DSP Guide for FPGAs

Lattice Semiconductor Corporation
5555 NE Moore Court
Hillsboro, OR 97124
(503) 268-8000

September 2009

Copyright

Trademarks

Disclaimers

of any correction if such be made. LSC recommends its customers obtain the latest version of the relevant information to establish, before ordering, that the information being relied upon is current.

Type Conventions Used in This Document

Convention	Meaning or Use
Bold	Items in the user interface that you select or click. Text that you type into the user interface.
<*Italic*>	Variables in commands, code syntax, and path names.
Ctrl+L	Press the two keys at the same time.
`Courier`	Code examples. Messages, reports, and prompts from the software.
`...`	Omitted material in a line of code.
`.` `.` `.`	Omitted lines in code and report examples.
`[ ]`	Optional items in syntax descriptions. In bus specifications, the brackets are required.
`( )`	Grouped items in syntax descriptions.
`{ }`	Repeatable items in syntax descriptions.
`|`	A choice between items in syntax descriptions.

Contents

1

Introduction

This book provides in-depth discussion of a variety of topics relating to high-performance digital signal processing (DSP) for Lattice Semiconductor FPGAs. More tips can be found in technical notes on the Lattice Semiconductor web site: www.latticesemi.com.

Digital Signal Processing The LatticeECP, LatticeECP2/M, LatticeECP3, and LatticeXP2 device families provide dedicated high-performance sysDSP blocks/slices on-chip for digital signal processing functions. The following two chapters discuss the design flow to fully utilize these blocks.

- Chapter 2 describes the basic modes of DSP design, explains the HDL based design flow, and lists the related Technical Notes for you to reference.
- Chapter 3 explains how to use The MathWorks MATLAB® and Simulink® modeling environment, in conjunction with the ispLeverDSP software, to add sysDSP blocks/slices to your designs. The chapter also describes the wide variety of Lattice Semiconductor sysDSP blocks/slices available to use with The MathWorks MATLAB/Simulink modeling environment, with specifications for each block.

2

DSP Design Using the HDL Flow for Lattice sysDSP FPGAs

This chapter discusses using an HDL flow to create designs that target the sysDSP blocks/slices of Lattice devices. The sysDSP blocks/slices of the Lattice sysDSP devices can be targeted in a number of ways.

- Use IPexpress to create modules that implement sysDSP elements. These modules can then be used in HDL designs as appropriate.
- Create HDL code of certain functions into a design's HDL, allowing the synthesis tools to inference the use of a sysDSP block/slice.
- Instantiate sysDSP primitives directly in the HDL source code.

Related Documentation

For detailed information on the DSP design flow for different Lattice FPGA devices, refer to the following Usage Guides on the Lattice Semiconductor Web site:

- TN1057 – LatticeECP-DSP sysDSP Usage Guide

  (http://www.latticesemi.com/lit/docs/technotes/tn1057.pdf)
- TN1107 – LatticeECP2/M sysDSP Usage Guide

  (http://www.latticesemi.com/lit/docs/technotes/tn1107.pdf)
- TN1140 – LatticeXP2 sysDSP Usage Guide

  (http://www.latticesemi.com/documents/TN1140.pdf)
- TN1182 – LatticeECP3 sysDSP Usage Guide

  (http://www.latticesemi.com/documents/tn1182.pdf)

3

DSP Design with MATLAB/ Simulink for Lattice FPGAs

Introduction

Lattice FPGAs provide dedicated high-performance DSP (digital signal processing) blocks on-chip. To unleash the high performance of the chips, Lattice Semiconductor offers ispLeverDSP as part of the ispLEVER design software for modeling and designing DSP systems in a visual data flow environment. In addition to have readily a great deal of functional abstraction for system modeling, DSP designers can also easily map the system model to a faithful hardware implementation without substantially compromising the quality of either the functional representation or the performance of the hardware implementation.

DSP System Design with The MathWorks MATLAB/Simulink

The design of a DSP system usually starts with high-level models that attempt to model real-world events and effects, processing blocks to perform the desired system operation, and models to manipulate the results and convey the resulting information in a manner that most easily allows the designer to determine if the system meets his requirements. The designer then typically must modify the system to conform to some rules that allow him to construct it using real-world devices at real-world system costs. Usually this involves several different design environments, depending upon the target hardware that is used to implement the system.

Often the design must be modified along the way in order to meet some constraint that may not have been known when the original system was designed. This usually means that the designer (or designers, as is more often the case), must take this new information back into the original design environment, modify and re-verify the system operation, and then follow the design process back through to the point where the original problem was encountered.

The MathWorks MATLAB/Simulink is a platform for multi-domain simulation and Model-Based Design of dynamic systems. It provides an interactive graphical environment and a customizable set of block libraries that let you accurately design, simulate, implement, and test control, signal processing, communications, and other time-varying systems. The ispLeverDSP block library consisting of Sinks, Sources, Nonlinear, and Linear components can be pulled into a design, configured and simulated with ease. The advantage of this approach over other simulation models is that Functional Blocks can be easily pulled into a design, graphically connected and configured in a fraction of the time that would be required with many simulation tools that may require entering complex equations and tables of values to get the simulation to run.

The MathWorks MATLAB /Simulink modeling environment, in conjunction with the ispLeverDSP, allows the entire design process to be done in a single environment. This greatly enhances productivity, reduces errors, and allows more what-if scenarios. Designers can more quickly determine the most optimal solution, resulting in faster time to market and lower system cost. The ispLeverDSP software allows designers to develop and verify designs at a high system level while guaranteeing device performance.

Getting Started

The ispLeverDSP software consists of Lattice device-specific blocks and the required applications to translate the design from a Simulink Behavioral Model to an RTL hierarchical design that can be implemented in the Lattice device-specific architecture, guaranteeing the desired performance and functionality desired from the Simulink based design. The Lattice basic blocks support all Lattice FPGA devices.

Each ispLeverDSP block has a menu called a mask that allows customization of the block through the setting of block parameters. The menu entries will be either a check box to select a feature, a dropdown menu with two or more selections, or a box to fill in the desired parameter value. For the latter, anything entered into the box will be evaluated as a MATLAB expression. Take care when entering in values because a type can easily be taken to be a variable reference that likely will not return the desired value, nor will a warning or an error be produced.

Most blocks in the library perform a mathematical or logical operation on one or more inputs. For most of these blocks, you can decide to allow for a full precision operation, or you may limit the word growth by limiting the number of output bits. When the latter option is selected, then the quantization method and overflow/underflow treatment can be selected. Word widths of arbitrary number of bits are supported. Additionally, most blocks allow the latency to be selected from zero clock cycles to up to 20 clock cycles.

You can refer to the following subsections for how to get started with the ispLeverDSP software:

- Installation
- Removing Old ispLeverDSP Blockset

Installation

The ispLeverDSP blockset is part of the ispLEVER software and is installed with the ispLEVER software. Refer to “ispLEVER Installation Notice” for more information on installing ispLEVER software.

In order to use ispLeverDSP blocksets, you have to install both ispLEVER and MATLAB/Simulink, and manually add ispLeverDSP paths into MATLAB search path as follows:

1. Start the MATLAB software.
2. Choose **File > Set Path** to open the Set Path dialog box.
3. Add `$ispTOOLS\ispLeverDSP` and `$ispTOOLS\ispFPGA\bin\nt` into MATLAB search path list, where `$ispTOOLS` is the directory of your ispLEVER software.

Note

The ispLEVER software and the MATLAB/Simulink are not necessary to be installed on the same disk drive.

In order to view the ispLeverDSP Help, you need manually copy help files from ispLEVER directory into MATLAB directory:

1. Browse to the $ispTOOLS\ispLeverDSP directory, where $ispTOOLS is the directory of your ispLEVER software.
2. Highlight and copy the **help** folder.
3. Inside the MATLAB directory, paste the **help** folder on top of the existing help folder.
4. In the Confirm Folder Replace dialog box, choose **Yes to All**.

To run ispLeverDSP blockset in MATLAB/Simulink, do the following:

1. Start the MATLAB software.
2. In the MATLAB command window, type **simulink**, or click the Simulink icon.

When typing **simulink** in the Command Window, a Simulink session will begin and display the Simulink Library Browser window.

Removing Old ispLeverDSP Blockset

If you have multiple versions of ispLEVER software installed, you will have multiple versions of ispLeverDSP blockset. The ispLeverDSP installer does not remove paths of old ispLeverDSP blockset from the MATLAB/Simulink software. This may result in old blockset being used instead of the new one. Follow these steps to manually remove paths of old ispLeverDSP from MATLAB/Simulink:

1. In MATLAB desktop, choose **File > Set Path**.

   The Set Path dialog box appears.

2. In MATLAB search path list, remove old ispLeverDSP paths. Save and close the dialog box.

Tutorials

Several tutorials in PDF-format are provided to show you how to create system designs with Lattice devices, using the Lattice design blocks within the MATLAB® Simulink software. The tutorials address designers who are already familiar with system modeling and the Simulink environment as well as those who are new to DSP design and Simulink tools.

The tutorials can be accessed directly from the first page of ispLEVER Help. Under the caption of **Tutorials**, click on the links entitled "**System Design Using ispLeverDSP**," "**DSP Floating Point to Fixed Point Conversion**," and "**DSP: Using Upsampling and Downsampling for Color Space Conversion**."

You can also find the tutorials in your ispLEVER install directory:

- ispcpld\Tutorial**ispleverdsp_tutorial.pdf**
- ispcpld\Tutorial**dsp_floating_point_tutor.pdf**
- ispcpld\Tutorial**dsp_upsample_downsample_tutor.pdf**

System Level Design with ispLeverDSP

A DSP design usually starts with a mathematical modeling using floating point number and concludes with a hardware realization using fixed-point numbers. ispLeverDSP automates the translation of the design into efficient hardware and Lattice Simulink blockset supports both floating point and fixed-point data types, either as a whole or mixed in the same design. In the next sections, we describe the capabilities of ispLeverDSP software and the ispLeverDSP design flow.

This section has the following subsections:

- Arithmetic Data Types
- Sample Rate and Clock Generation
- ispLeverDSP Design Flow
- Hints and Tricks

Arithmetic Data Types

Floating Point Data Types

Although floating point data types cannot be converted to hardware, this allows for a much smoother transition to a fixed-point implementation. Rounding and saturation effects can be more carefully controlled by selective conversion to a fixed-point representation. Floating point operation can be selected on a block-by-block basis, for the entire design, or by inheritance. When inheritance is selected and, if a block has any input that is floating point, the block will also become a floating point block.

Fixed Point Data Types

All fixed-point operations are performed using arbitrary precision fixed point math. This allows you to explicitly control all rounding operations, unlike floating point math. In arbitrary precision fixed point math, the word length will grow as required to maintain full precision.

For example, the full precision output for a multiplier with two 9 bit inputs will be 18 bits. You can determine when to limit the output size according to your specific requirements.

The fixed-point data type is specified by the number format, either signed or unsigned, the total number of bits inclusive of the sign bit as a positive value, and the binary point as a positive or zero value. The maximum number of bits that can be specified in fixed point representation is 150.

The specification of the fixed-point representation usually is needed only at data sources, as most blocks will inherit the characteristics of the driving block. Full fixed-point precision can be specified by most blocks, avoiding most sources of rounding and saturation effects. If desired, the bit growth can be controlled by specifying a maximum number of bits for the block function. In this case, you must specify how to handle possible saturation and rounding effects.

For saturation, you can specify that the output be saturated, that the values wrap around, or that an error be reported. Note that the wrap around mode will not require any additional hardware be generated. For rounding, you can specify that the result be truncated, be rounded to the nearest value, or rounded using convergent rounding. Convergent rounding will eliminate the bias inherent in the round to nearest mode, but will require additional hardware. The truncate mode requires no additional hardware.

In cases where feedback loops exist, it may be necessary to insert an additional block in order to fix the precision for the loop. If the ispLeverDSP software detects a feedback loop and it cannot determine the bit width, then an error will be indicated.

Signed or Unsigned Fixed Point Representation

You have the option of specifying either signed or unsigned fixed-point representation. Most blocks will automatically resolve mixed sign operations by converting the unsigned value to a signed representation. Additionally, values will usually be automatically extended in bit width as required. Most blocks can also automatically align the binary points when the inputs are of different values.

The ispLeverDSP software can annotate the Simulink model with text indicating the settings of the data type at each block output. This display is enabled by setting the “Port data types” option in the “Tools” menu. The text displayed is as follows:

```
(u)lfp#_#
```

where “(u)” indicates an optional “u” that is displayed if the data type is unsigned, the first “#” is the total number of bits, and the second “#” is the binary point.

Floating Point to Fixed Point Design Conversion

The following steps allow you to convert a floating point design to a fixed point design.

1. Create your design in MATLAB/Simulink.
2. Check the “Override With Doubles” box in the Lattice Generator block, as shown in Generator. When this is checked, all the blocks will output double precision.
3. Verify the floating point design performance using Simulink simulation.

   Note

   If floating point results are correct, continue with next step. If floating point results are not correct, check your design for errors.

4. Set precision settings for each block in the design.
5. Uncheck the “Override With Doubles” box in the Lattice Generator block to obtain a fixed point design.
6. Select appropriate precision for the fixed point design to meet design objective in terms of performance and hardware resource constraints.

Sample Rate and Clock Generation

All blocks located between the Lattice Gateway In and Lattice Gateway Out blocks must be Lattice ispLeverDSP blocks. The sample period must be specified at all Lattice Gateway In blocks. All other Lattice blocks can either inherit their sample rates from their inputs, or the sample rate can be explicitly set for each block. All sample rates must maintain an integer relationship to the fastest sample rate, or system rate. If a block receives multiple sample rates, it will be assigned the fastest sample rate.

Usually a change in the sample rate should be accomplished through the use of the Upsample or Downsample blocks. This insures that the data is correctly sampled during sample rate changes.

Sample Rate Probe

The Sample Rate Probe can be used to convert the sample rate of any signal to a double value that can then be displayed. This block is only for simulation use and will not have any effect on the generated hardware.

In the hardware generation, a single clock will be generated at the system sample rate. Slower sample rates are produced by automatically generating clock enables at the lower rates. The Clock Enable Probe can be used to view the clock enables for any signal in the system. This can also be used to create additional signals for use by the system. Note that this will create additional hardware in the design.

Clock Probe

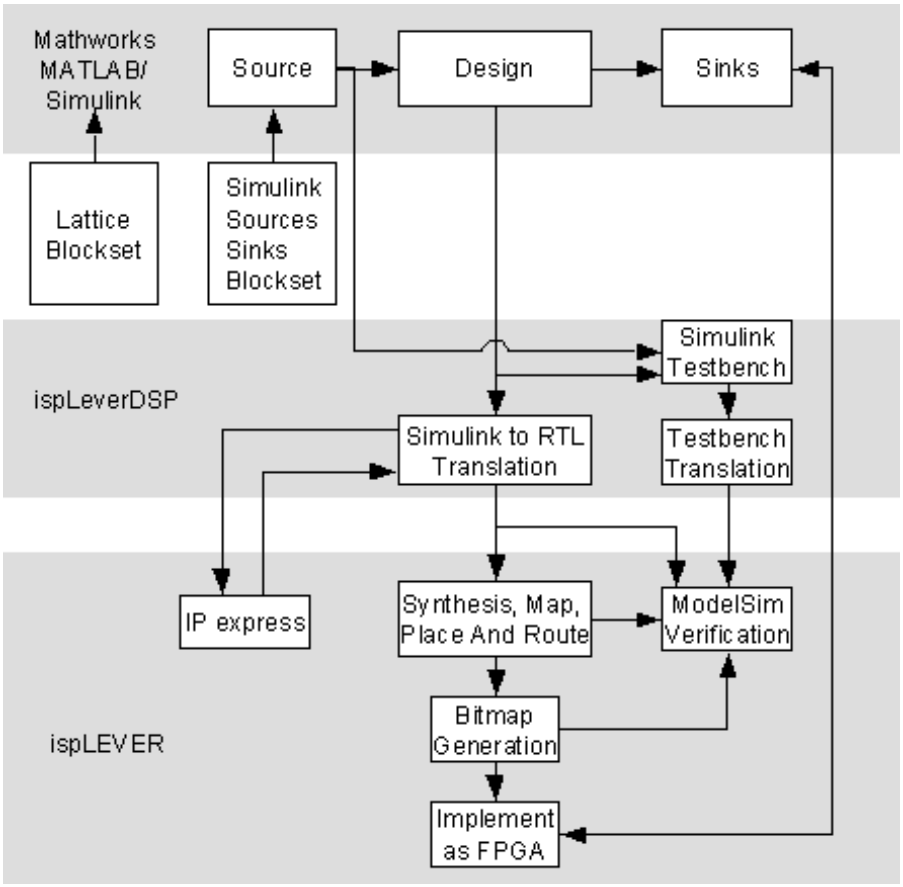
The Clock Probe can be used to view the effective clock signal present for any signal. This block is only for simulation use and will not have any effect on the generated hardware.

ispLeverDSP Design Flow

A typical ispLeverDSP design flow goes through the following steps:

1. Design Modeling
2. Translation
3. Verification
4. FPGA Fitting
5. Hardware Implementation

The following figure shows the ispLeverDSP design flow.



Design Modeling

The design is constructed using combinations of Simulink blocks and Lattice blocks. Gateway In blocks at the inputs and Gateway Out blocks at the outputs delimit the boundary of the design. Gateway blocks can appear at any level of the design. All blocks between the Gateway blocks must be blocks from the Lattice blockset. The Generator will target everything between these blocks to the FPGA by producing synthesizable code in HDL that perform the specified functions. The Generator block supports both Verilog HDL and VHDL, and must be included during the initial Simulink modeling phase.

The Lattice Gateway In and Lattice Gateway Out blocks are used to interface the rest of the design to other non-Lattice blocks such as the Simulink Source and Sink blockset.

The Simulink Source Blocks are used to stimulate the design, ultimately producing an HDL equivalent testbench that can be used when verifying the design using ModelSim.

The Simulink Sink Blocks are used to monitor the output of the system which will then be translated to a set of equivalent set of expected vectors which can be used to automatically compare the Simulink simulation results with the ModelSim simulation.

Most Simulink blocks produce double values. These are converted to a fixed precision by the Gateway In blocks in order to produce a fixed precision implementation for the design. The designer has full control over the final precision, which will have a direct impact on the size and performance of the design.

All Lattice blocks also have a mode in which the full double value can be passed through the block in order to validate the design without concern for rounding and saturation issues. Most blocks also allow full precision to be retained regardless of the actual precision. This is useful in that the precision need only be set at key points in the design; the remaining blocks will pass that precision setting through without introducing quantization or overflow effects.

Translation

The Generator block must be included in the top level of the design to provide the necessary control to target for a Lattice FPGA. The Generator block provides options to automatically generate an HDL testbench for functional simulation, perform functional simulation with Modelsim, generate ispLEVER project files, generate scripts for various synthesis tools available in ispLever, select specific target Lattice FPGA devices, and allow timing preferences to be specified. Once the Simulink design has been completed and verified to the designer's satisfaction, the Generate Structural HDL box should be checked in the Generator block.

The simulation should then be re-run, and once completed, the netlist will be generated. If any design rules have been violated, they will be flagged at this point. The Generator block produces an HDL netlist that can be sent to the ispLEVER tool for fitting into the selected FPGA.

Verification

There are several options for HDL verification. Once the structural HDL has been generated, an HDL testbench can be created in order to verify the design. The design can then be automatically simulated and verified with ModelSim using the generated testbench. The simulation can be selected to run either in a GUI window or as a command line. The command line is useful for automated testing and for checking of previously working simulations.

If the Generate Testbench box is checked in the Generator block, the data passing through the Gateway blocks will be used as stimulus and response data. A testbench will be automatically generated using that data.

This testbench, along with the generated structural HDL and data files, can then be simulated in ModelSim. If the HDL response differs from the Simulink response, it will be flagged as an error. In addition, when the Generate Testbench box is checked, a “Run ModelSim” selection appears in the dialog box. You can then choose to either:

- not automatically run ModelSim,
- run ModelSim in GUI mode,
- run ModelSim in command line mode.

If “run ModelSim in GUI mode” or “run ModelSim in command line mode” are selected, ModelSim will automatically be run with the appropriate files in the selected mode when the structural HDL and testbench have been generated. The GUI mode is most useful for initial debugging. Once the simulation is passing, the command line mode provides a way to perform quick automated testing to verify subsequent simulations.

In this mode, the simulation is run and the results are saved in a file called “transcript” in the target directory. The transcript file can then be examined to determine if the simulation passed. A sample of the error message from a transcript file for a failing simulation follows.

```
# ** Error: compare error #    Time: 150 ns  Iteration: 1
Instance: /test_fifo_noae_tb/uut_lattice_gateway_out1
# ** Error: compare error #    Time: 150 ns  Iteration: 1
Instance: /test_fifo_noae_tb/uut_lattice_gateway_out3
# ** Error: compare error #    Time: 150 ns  Iteration: 1
Instance: /test_fifo_noae_tb/uut_lattice_gateway_out4
# ** Note: Finished with simulation #    Time: 480 ns
Iteration: 1  Instance: /test_fifo_noae_tb/
uut_lattice_gateway_out
# ** Failure: #    Time: 480 ns  Iteration: 1  Process: /
test_fifo_noae_tb/uut_lattice_gateway_out/main
File: test_fifo_noae_tb.vhd # Break at test_fifo_noae_tb.vhd
line 204 # Stopped at
test_fifo_noae_tb.vhd line 204 # Simulation Breakpoint: Break
at test_fifo_noae_tb.vhd
line 204 # MACRO ./test_fifo_noae.udo PAUSED at line 11
```

Note

When the simulation completes, you will see a statement that says “Failure” in the ModelSim transcript window. This is due to the assertion statement inside the testbench to end simulation. This is a normal condition and does not indicate that simulation did not complete successfully.

It is possible to get a simulation mismatch under certain conditions. For example, reading of an uninitialized memory location will result in unknowns in the HDL simulation being compared to zeros from the Simulink simulation. This will result in a simulation mismatch.

A simulation mismatch can be prevented by insuring that only initialized memory locations are read. Failure to properly reset a block may also result in unknowns in the simulation that can cause simulation mismatches.

FPGA Fitting

Once the HDL simulation has been verified, you can then proceed to fit the design into an FPGA using ispLEVER. All necessary files are located in the target directory. The remaining steps are no different from the normal method of fitting a VHDL/Verilog design. Please refer to the ispLEVER documentation for instructions on how to accomplish this task.

Due to the manner in which the HDL code is written, some warnings may be issued during synthesis. These warnings are the result of synthesis optimization of generic code. These warnings do not affect the correctness of

the final result. Simulation of the post-synthesis netlist using the generated testbench should always be done to insure that any warnings are spurious.

Hardware Implementation

A number of blocks can be implemented using the ispDSP block, which will normally result in superior performance and efficiency. There are two options available to implement these blocks, and the resulting performance and utilization will differ depending upon the method used.

For blocks that can be implemented in the ispDSP block, there is a **Use DSP Block** option. If this box is checked, the netlist will call the IPexpress tool to implement the function. Depending upon the other options chosen, this may include internal pipelining. Note that by using this method, it is possible to run out of ispDSP blocks, causing the fitter to fail. However, in general, this will result in the highest performance provided sufficient latency is included to take advantage of the internal pipelining that is available in the ispDSP block.

If the **Use DSP Block** is not checked, then synthesizable code will be produced for the block. The chosen synthesis tool will then determine when the ispDSP block can be used, and whether it can use the internal pipelines. Note that different synthesis tools may give different results, and different version of the tools may also differ in their mapping schemes. The advantage of this method is that if there is no ispDSP block available, the tool will use the FPGA fabric to implement the design. This will result in a higher probability for fitting success. It may, however, also result in significantly lower performance depending upon the capability of the synthesis tool to effectively map to the sysDSP block.

Hints and Tricks

This section gives some suggestions to make design entry and verification easier.

Enable Port Data Types

To display the signals data format, enable the data type display as follows:

1. Select 'Format' from menu bar.
2. Select 'Port/Signal Displays'.
3. Select 'Port Data Types'.

Synthesis Strategy

Most blocks are synthesized directly from HDL RTL code, which gets mapped to the logic blocks by the synthesis tool. Most memory blocks will use calls to the Module Manager to generate the required function by using the EMBs (Extended Memory Blocks). For any block that utilizes a multiplier, it will normally be synthesized using calls to the Module Manager to implement the function in the ispDSP block. This can be forced by checking the “Use DSP Block” option if it occurs in the block mask. If this is not checked, some blocks will allow the synthesis tool to determine if the function will fit into a sysDSP block. If not, it will implement the function in the logic blocks. As a result, performance will vary depending upon the final implementation taken. If you want the highest performance, always make sure the “Use DSP Block” box is checked. This is the default behavior for blocks in the library. However, if you copy a block from another model, you may no get this behavior by default.

Board Level Support

The ispLEVER software includes board level support codes that are generic for any Lattice Evaluation Board. To run co-simulation with desired Lattice Evaluation Board, you need to generate and drop the board related data files into the property directory. The following are the steps that you should follow.

1. From Lattice Technical Support, obtain the .pkt and .bsm files for the specific Lattice Evaluation Board.
2. Generate the .xcf file as follows:
    a. In the “Implementation” tab of the Generator Parameters dialog box, select the target device.
    b. Run Simulink simulation with Active HDL as the HDL simulation tool.
    c. Find the .syn file in your working directory and load it in ispLEVER.
    d. In ispLEVER, generate either the .bit or the .jed file.
    e. Connect the Evaluation Board to the computer.
    f. Open the ispVM System software, add the device, fill in the data file generated in Step 2d, and test your board. Make sure there is a .xcf file generated in your working directory.
3. In the “Board Level Support” tab of the Generator Parameters dialog box, choose **User Specified Board** from the “Select Board” drop-down list. Enter the .package, .pkt, and .bsm file names associated with the desired board. Do not include the file extension.
4. Drop the .package, .pkt, and .bsm files in the $ispTOOLS\ispLeverDSP directory, where $ispTOOLS is the directory of your ispLEVER software.
5. Run hardware co-simulation.

Interpreting and Fixing Errors

On occasion, you may receive an error that you don’t understand. This section is intended to help in those situations when the problem is a frequently occurring one, and a solution has been found.

```
“Data size error with data width in '%s', detected size=%d.
Check for feedback loop that does not have a defined precision”
```

This error may occur because there is a feedback loop and the tool cannot determine the data width within the loop. The solution is to include a “Convert” block within the loop, and specify the width in the dialog box after selecting “User Defined” for the Precision. Make sure that “Wraparound” is selected for the saturation mode, and that “Truncate” is selected for the quantization mode so no additional hardware is generated.

Blocks - Categorical List

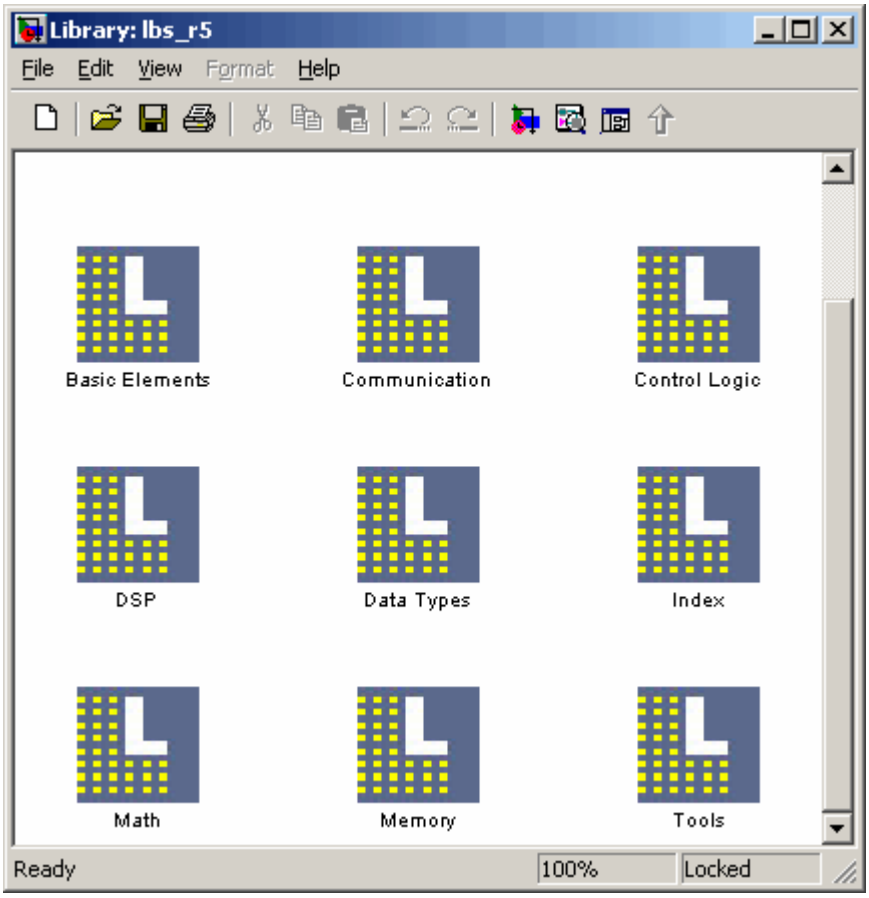
The Lattice ispLeverDSP blocks are grouped into libraries according to their function. Some blocks with broad applicability (e.g., the Gateway I/O blocks) are linked into multiple libraries. You can find the following libraries in the Simulink Library Browser:

Table 1: Blocks Organized in Libraries

Library Name	Brief Description
Index	Includes every block in the Lattice Blockset.
Basic Elements	Includes standard building blocks for digital logic.
Control Logic	Includes blocks used for control circuitry and state machines.
Data Types	Includes blocks that convert data types (includes gateways).
DSP	Includes digital signal processing (DSP) blocks.
Math	Includes blocks that implement mathematical functions.
Memory	Includes memories.
Tools	Includes “utility” blocks, e.g., code generation (Generator block), resource estimation, HDL co-simulation, etc.

Accessing the Libraries

You can access the main library of the ispLeverDSP Blockset by entering **lbs_r5** in the MATLAB Command Window. From the main library, you can access sub libraries by double-clicking their icons.


Library: lbs_r5
File Edit View Format Help
Basic Elements
Communication
Control Logic
DSP
Data Types
Index
Math
Memory
Tools
Ready
100%
Locked

On Windows platforms, you can also use the Simulink Library Browser to access libraries of the Communications Blockset. To open the Simulink Library Browser, enter simulink in the MATLAB Command Window.

Basic Elements

Library of basic elements Includes standard building blocks for digital logic.

Table 2: Basic Elements Blocks

Block Name	Brief Description
Addressable Shift Register	Create an addressable chain of registers.
Black Box	Enable MTI or board based co-simulation.
Clock Enable Probe	Create a clock enable for a signal.
Concat	Concatenate two buses into a single bus.
Constant	Specify a constant value.
Convert	Convert the input to a different data type or reduce the number of bits by performing rounding and/or saturation.
Counter	Create various types of counters.
Delay	Delay Input Value.
Downsample	Downsample the input.
Expression	Evaluate a logical combination of two or more inputs.
Gateway In	Define device input port, and provide a conversion from double to Lattice Fixed Point (lfp) data format.
Gateway Out	Define device output port, and provide a conversion from Lattice Fixed Point (lfp) data format to a double.
Generator	Generate structural HDL and testbench for system. An option is provided to run Modelsim to verify the generated HDL code.
Inverter	Invert an input signal.
LFSR	A shift register which modifies itself with feedback.
Logical	Perform the specified logical function on two or more inputs.
Mux	Multiplex 2 or more inputs.
Parallel to Serial	Convert an input word into a series of serial output bits.
Register	Delay Input Value by one cycle. The initial/reset value may be specified.
Relational	Perform a comparison between two inputs.
Serial to Parallel	Convert a series of input bits to parallel output.
Slice	Extract a range of bits from the input signal.

Table 2: Basic Elements Blocks (Continued)

Block Name	Brief Description
Sync	Synchronize between two to four inputs.
Upsample	Upsample the input.

Control Logic

Library of Control Logic includes blocks used for control circuitry and state machines.

Table 3: Control Logic Blocks

Block Name	Brief Description
Counter	Create various types of counters.
Expression	Evaluate a logical combination of two or more inputs.
Inverter	Invert an input signal.
Logical	Perform the specified logical function on two or more inputs.
Mux	Multiplex 2 or more inputs.
Register	Delay Input Value by one cycle. The initial/reset value may be specified.
Relational	Perform a comparison between two inputs.
Shift	Shift an input value by the specified number of bits.
Slice	Extract a range of bits from the input signal.

Data Types

Library of Data Types includes blocks that convert data types (includes gateways).

Table 4: Data Types Blocks

Block Name	Brief Description
Concat	Concatenate two buses into a single bus.
Convert	Convert the input to a different data type or reduce the number of bits by performing rounding and/or saturation.
Gateway In	Define device input port, and provide a conversion from double to Lattice Fixed Point (lfp) data format.
Gateway Out	Define device output port, and provide a conversion from Lattice Fixed Point (lfp) data format to a double.
Parallel to Serial	Convert an input word into a series of serial output bits.
Scale	Scale an input value by a power of 2. The value produced is <input>*2^scale.
Serial to Parallel	Convert a series of input bits to parallel output.
Shift	Shift an input value by the specified number of bits.
Slice	Extract a range of bits from the input signal.

DSP

DSP library includes digital signal processing (DSP) blocks.

Table 5: DSP Blocks

Block Name	Brief Description
CIC	Parameterizable Cascaded Integrator-Comb Filter IP core, used to achieve arbitrary and large sample rate changes in digital systems.
Convolutional Encoder	A parameterizable IP core for convolutional encoding of a continuous or burst input data stream.
Cordic	Parameterizable Coordinate Rotation Digital Computer IP core.
CSC	A parameterizable IP core for converting signals from one color space to another color space.
DA_FIR	Parameterizable Distributed Arithmetic Finite Impulse Response Filter Generator IP core.
FFT	Parameterizable Fast Fourier Transform IP core.
FIR	Parameterizable FIR Filter IP core.
Gamma	A parameterizable IP core that enables pre-distortion correction made to images or video frames to offset the non-linear behavior of display systems, such as CRT displays.
Interleaver Deinterleaver	A parameterizable IP core to overcome correlated channel noise such as burst error or fading.
NCO	A parameterizable IP core to offer several advantages over other types of oscillators in terms of accuracy, stability and reliability.
Viterbi Decoder	A parameterizable core for decoding different combinations of convolutionally encoded sequences.

Math

Math library includes blocks that implement mathematical functions.

Table 6: Math Blocks

Block Name	Brief Description
Accumulator	Accumulate sum or difference.
AddSub	Add or subtract two inputs.
CMult	Multiply an input by a constant.
Constant	Specify a constant value.
Convert	Convert the input to a different data type or reduce the number of bits by performing rounding and/or saturation.

Table 6: Math Blocks

Block Name	Brief Description
Counter	Create various types of counters.
Expression	Evaluate a logical combination of two or more inputs.
Inverter	Invert an input signal.
Logical	Perform the specified logical function on two or more inputs.
Mac	Multiply two inputs and accumulate the product.
Mult	Multiply two inputs.
Negate	Negate an input signal. (Performs 2's complement on input value).
Relational	Perform a comparison between two inputs.
Scale	Scale an input value by a power of 2. The value produced is <input>*2^scale.
Shift	Shifts an input value by the specified number of bits.
Threshold	Output 1 if the input is positive and a –1 if the input is negative.

Memory

Memory library includes memories or related blocks.

Table 7: Memory Blocks

Block Name	Brief Description
Addressable Shift Register	Create an addressable chain of registers.
Delay	Delay Input Value.
Dual Port RAM	Dual Port Memory.
FIFO	FIFO (First In First Out).
LFSR	A shift register which modifies itself with feedback.
Register	Delay Input Value by one cycle. The initial/reset value may be specified.
ROM	Read-only Memory (ROM).
Single Port RAM	Single-Port Memory.

Tools

Tools library includes “utility” blocks, e.g., code generation (Generator block), resource estimation, HDL co-simulation, etc.

Table 8: Tools Blocks

Block Name	Brief Description
Black Box	Enable MTI or board based co-simulation.
Clock Enable Probe	Create a clock enable for a signal.
Clock Probe	Create a clock signal for simulation viewing.
Generator	Generate structural HDL and testbench for system. An option is provided to run Modelsim to verify the generated HDL code.
Print	Print an internal signal as a double in the MATLAB window.
Signal Probe	Output a double representation of the input.

Blocks - Alphabetical List

The following table lists the Lattice blocks currently available in MATLAB/ Simulink in alphabetical order.

Table 9:

Block Name	Brief Description
Absolute	Absolute value of the input is asserted to the output.
Accumulator	Accumulate sum or difference.
Addressable Shift Register	Create an addressable chain of registers.
AddSub	Add or subtract two inputs.
Black Box	Enable MTI or board based co-simulation.
CIC	Parameterizable Cascaded Integrator-Comb Filter IP core, used to achieve arbitrary and large sample rate changes in digital systems.
Clock Enable Probe	Create a clock enable for a signal.
Clock Probe	Create a clock signal for simulation viewing.
CMult	Multiply an input by a constant.
Concat	Concatenate two buses into a single bus.
Constant	Specify a constant value.
Convert	Convert the input to a different data type or reduce the number of bits by performing rounding and/or saturation.
Convolutional Encoder	A parameterizable IP core for convolutional encoding of a continuous or burst input data stream.
Cordic	Parameterizable Coordinate Rotation Digital Computer IP core.
Counter	Create various types of counters.
CSC	A parameterizable IP core for converting signals from one color space to another color space.
DA_FIR	Parameterizable Distributed Arithmetic Finite Impulse Response Filter Generator IP core.
Delay	Delay Input Value.
Downsample	Downsample the input.
Dual Port RAM	Dual Port Memory.
Expression	Evaluate a logical combination of two or more inputs.
FFT	Parameterizable Fast Fourier Transform IP core.
FIFO	FIFO (First In First Out).

Table 9:

Block Name	Brief Description
FIR	Parameterizable FIR Filter IP core.
Gamma	A parameterizable IP core that enables pre-distortion correction made to images or video frames to offset the non-linear behavior of display systems, such as CRT displays.
Gateway In	Define device input port, and provide a conversion from double to Lattice Fixed Point (lfp) data format.
Gateway Out	Define device output port, and provide a conversion from Lattice Fixed Point (lfp) data format to a double.
Generator	Generate structural HDL and testbench for system. An option is provided to run Modelsim to verify the generated HDL code.
Indeterminate Probe	A double value of one is asserted at output if the data on the probe input is indeterminate. Otherwise, the probe output is a double value of zero.
Interleaver Deinterleaver	A parameterizable IP core to overcome correlated channel noise such as burst error or fading.
Inverter	Invert an input signal.
LFSR	A shift register which modifies itself with feedback.
Logical	Perform the specified logical function on two or more inputs.
Mac	Multiply two inputs and accumulate the product.
MacDSP	Multiply two inputs and accumulate the product.
MinMax	Find the minimum or maximum of a set of inputs.
Mult	Multiply two inputs.
MultAdd2	Compute a sum of two products.
MultAdd4	Compute a sum of four products.
Mux	Multiplex 2 or more inputs.
NCO	A parameterizable IP core to offer several advantages over other types of oscillators in terms of accuracy, stability and reliability.
Negate	Negate an input signal. (Performs 2's complement on input value).
Parallel to Serial	Convert an input word into a series of serial output bits.
Parameterizable Mux	A multiplex of 2 or more inputs with a user-specified parameter as the select value.
Print	Print an internal signal as a double in the MATLAB window.

Table 9:

Block Name	Brief Description
Register	Delay Input Value by one cycle. The initial/reset value may be specified.
Reinterpret	Change the data type of an input without changing its bit pattern.
Relational	Perform a comparison between two inputs.
Replicate Bit	Input bits are replicated n times to an output bus.
ROM	Read-only Memory (ROM).
Sample Rate Probe	Report sample rate of input.
Scale	Scale an input value by a power of 2. The value produced is <input>*2^scale.
Serial to Parallel	Convert a series of input bits to parallel output.
Shift	Shift an input value by the specified number of bits.
Signal Probe	Output a double representation of the input.
Single Port RAM	Single-Port Memory.
Slice	Extract a range of bits from the input signal.
Subsystem Generator	Provide a way of specifying options at a localized scope level.
Sync	Synchronize between two to four inputs.
Threshold	Output 1 if the input is positive and a –1 if the input is negative.
Upsample	Upsample the input.
Variable Shift Register	The output is shifted left or right by a number of bit positions with respect to the input. Shift mode may be either arithmetic or end-around.
Viterbi Decoder	A parameterizable core for decoding different combinations of convolutionally encoded sequences.

Common Parameters for ispLeverDSP Blocks

A number of blocks have certain parameters in common. If the parameter appears in the block, it will always have the same meaning regardless of the block.

Precision = [Full, User Defined]

If The User-Defined setting allows the user to specify output width and the binary point location of the output word. If **Full** is selected, the number of bits used will be automatically selected to guarantee that no precision is lost in the data. If **User Defined** is selected, the block mask will expand to display additional parameters for the user to specify.

Output Width

Number of bits in the output word. A positive integer that specifies the total number of bits that will be used to represent the data in two's complement format. Valid values are from 1 to 150. If Precision is “User Defined,” this value may be different from the width of the input ports. If so, then the Quantization, Overflow, and Numeric Format parameters determine how the conversion from input to output is accomplished.

Binary Point

A positive integer that specifies the position of the binary point in the output data word. Valid values are from 0 to the output width.

Numeric Format

Selection between **Signed** or **Unsigned**. It defines the format of the data value to be signed or unsigned. (The statement that signed treats the MSB as a positive or negative number should be added – 0 for positive – 1 for negative).

Quantization

Selection between **Truncate**, **Round to Nearest**, or **Convergent Rounding**. This parameter is only applicable when Precision is “User-Defined” and the selected value is smaller than the input ports. Three settings exist and determine the manner in which the input format value is shortened to the output format. Truncation, Round to Nearest, and Convergent Rounding. Truncation shortens the format by simply discarding the excess bits of the larger input format over the output format. Rounding to Nearest adds or subtracts an lsb if the excess bits sum to an absolute value that exceeds an lsb of the output format. Convergent Rounding is the same as Round to Nearest except when the excess bits add to exactly one-half an lsb, in which case a refinement to the Round to Nearest algorithm is employed. If the number of output fractional bits is less than the input, then quantization will be done according to this selection. Note that choosing any selection other than “Truncation” will result in an increase in the hardware requirements and potentially a lower overall performance.

Overflow Treatment

Selection between **Saturate**, **Wraparound**, or **Report Error**. If the number of integer output bits is less than that of the input, then any overflow or underflow

will be treated according to this selection. If “Report Error” is selected, the simulation will terminate on overflow with an error message indicating the block in which it was detected. Note that choosing “Saturate” will result in an increase in the hardware requirements and potentially a lower overall performance.

Latency

This is the latency desired for the block. Some blocks may have a minimum latency greater than zero. For most blocks, values up to 20 are allowed. Note that blocks that can be implemented in the ispLeverDSP block have a latency of up to 3 clock cycles built in, so there is no additional hardware cost for using this. For all other blocks, and for latencies greater than 3 for the sysDSP block, additional latency will increase the hardware requirement but generally will improve performance.

Override with Doubles?

If this checkbox is selected, then the block will produce the full precision of a double value. Note that when this is checked for any block, you cannot create a netlist, as the design cannot be targeted to an FPGA device.

Use Explicit Sample Period?

If this box is selected, you are prompted for a sample period as shown below. Otherwise, the sample period will be inherited from the driving block. If the block has multiple inputs, and the two inputs have differing sample rates, then an error will be issued..

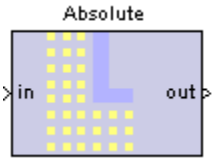
Provide Enable Port?

If the latency is greater than zero, then you may choose to explicitly generate an enable port for the pipeline registers. Check this box to provide an enable pin to the block. All counter functions except a reset (if provided) are disabled when this pin is low.

Provide Reset Port?

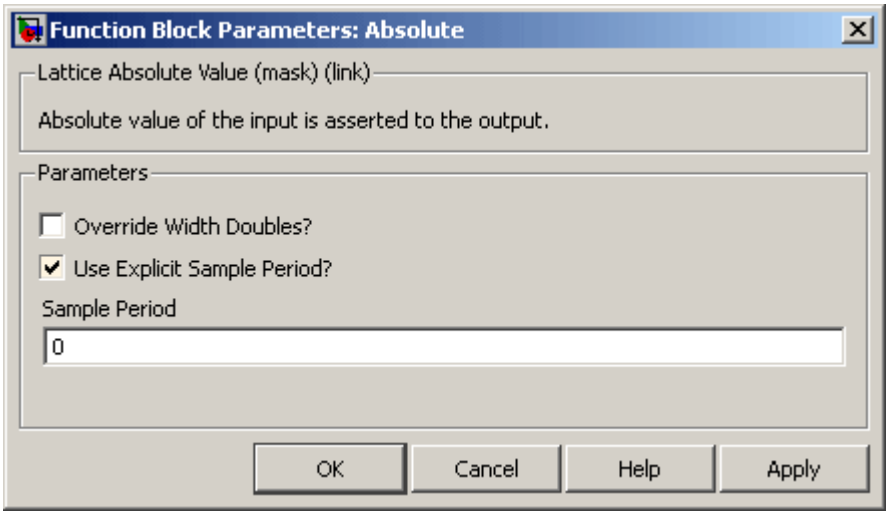
If the latency is greater than zero, then you may choose to generate a reset port for clearing the pipeline registers. The reset for all blocks is synchronous. Check this box to provide a reset pin to the block. When asserted, the output will be loaded to the initial value. This is a synchronous reset.

Absolute



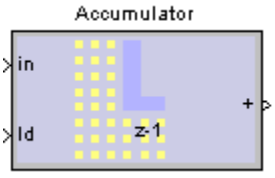
The Absolute Block has one input and one output. The output value is the absolute value of the input value. If the input is a positive value, signed or unsigned, with any combination of integer and fractional parts, the output value is the same. If the input has a negative value, however, the block calculates the positive value having the same magnitude as the input and asserts it to the output. Width, Arithmetic Type, and Binary point of the input are inherited from the source. Width, Arithmetic Type and Binary point of the output are the same as the input.

The user configurable parameters for this block are available in the following dialog box.



Parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

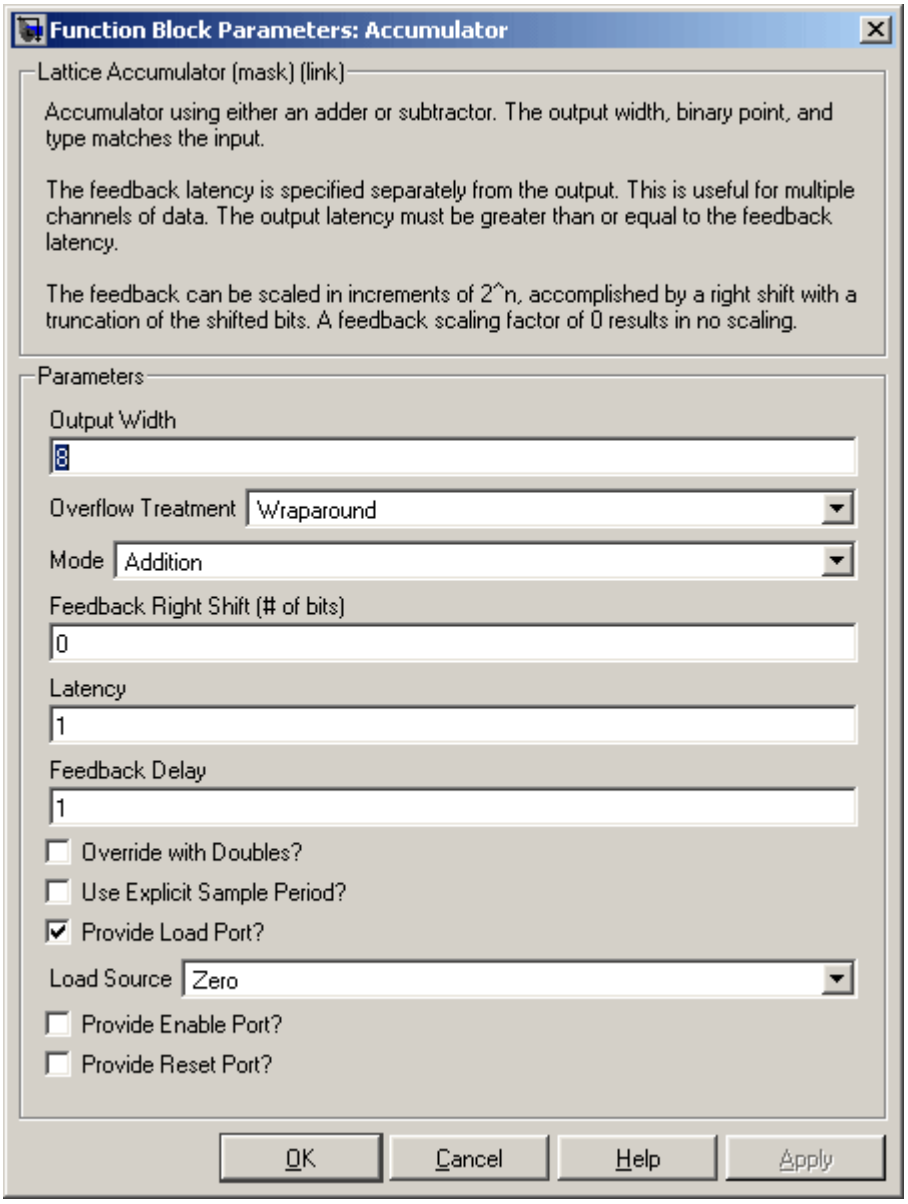
Accumulator



The Accumulator block performs an accumulation if the values that appear at the input. The accumulation can be either by addition or subtraction. For subtraction, the output is subtracted from the input. The output can be scaled in binary increments prior to accumulation with the input. The output width will be equal to the input width. If treatment for overflow can be selected as either wrap around or saturate.

The feedback delay is specified independent from the output latency. The output latency must be greater than or equal to the feedback delay.

The user configurable parameters for this block are available in the following dialog box.



Block parameters include:

- **Output Width**:

  This must be equal to the width of the input or an error will result.

- **Mode**:

  Selection between Addition and Subtraction. This selects whether the output is added to or subtracted from the input.

- **Feedback Right Shift (# of bits)**:

  This determines the scaling for the output prior to accumulation. This must be a positive integer less than the input width.

- **Feedback Delay**:

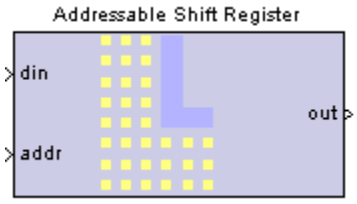
This is the delay for the output prior to being accumulated. This parameter must have a minimum value of 1 and a maximum of 50.

- **Load Source= [Input, Zero, Alternate Input]**:

  This parameter will be enabled when the optional load port is selected. This parameter selects the source for loading. If Alternate Input is selected, a Din port will be added to the model.

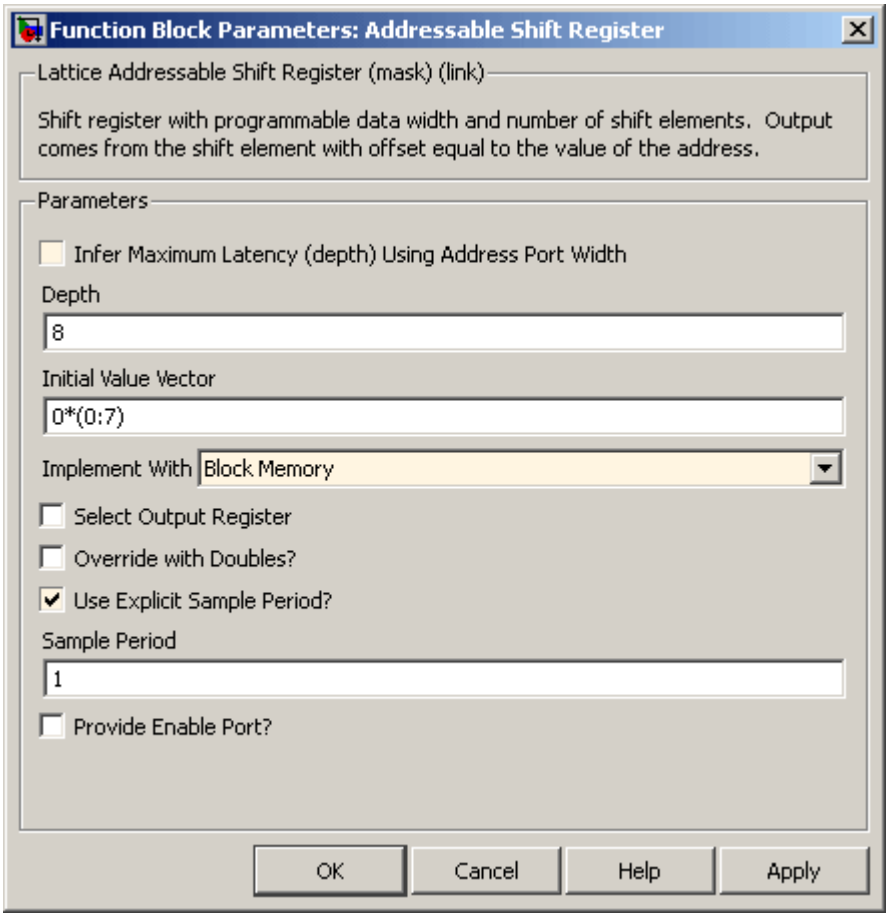
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Addressable Shift Register



The Addressable Shift Register block implements a parameterizable delay shift register that can be addressed at sample rates that are different from the rate at which the data in the register is shifted. The data in the register is shifted at the rate of the data input, and addressed at the rate of the address input. The width of the register is inferred from the data input. The depth if the registers can be from 2 to 1024. The depth can be inferred from the width of the address input if desired. The register can be initialized to a user-defined value.

The user configurable parameters for this block are available in the following dialog box.



Block parameters are:

- **Infer Maximum Latency (depth) Using Address Port Width**:

  Choose per design specification.

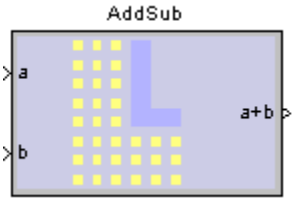
- **Depth**: Choose per design specification.
- **Initial Value Vector**: Choose per design specification.
- **Implement With**:

  Selection between Distributed Memory, Block Memory or PFU registers. Choose per design specification.

- **Select Output Register**: Choose per design specification.

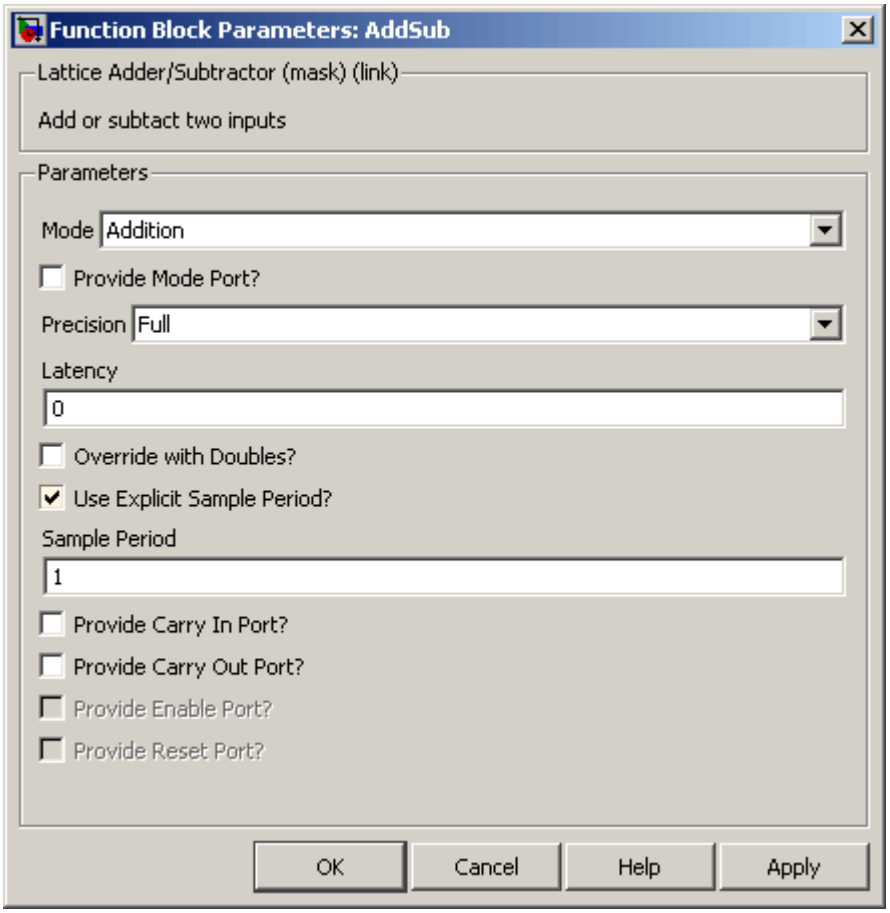
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

AddSub



The Adder/Subtractor block performs either addition or subtraction of two input values. For subtraction, the B input is subtracted from the A input. The output may be either full precision, or you can select the output characteristics. The input binary points are aligned. If one of the inputs is unsigned and the other signed, the unsigned input will be sign extended, and the resulting output will be signed. The hardware implementation will always match the Simulink model, so if the two inputs differ in width, the narrow input will be padded and/or sign extended as required.

The user configurable parameters for this block are available in the following dialog box.



Block parameters are:

- **Mode = [Addition, Subtraction]**:

  Selects whether an addition of a subtraction is performed.

- **Provide Mode Port?**: Choose per design specification.
- **Provide Carry In Port?**: Choose per design specification.

- **Provide Carry Out Port?**: Choose per design specification.

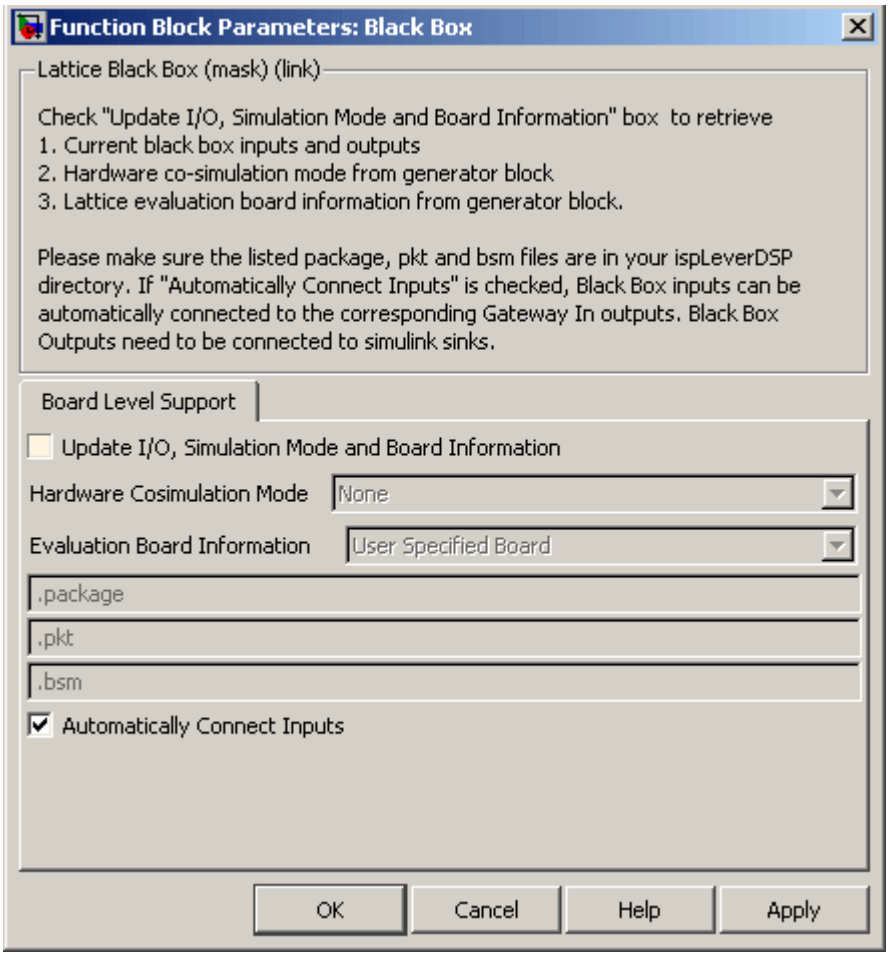
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Black Box

Black Box

The Black Box block is used to enable MTI or board based co-simulation.

The user configurable parameters for the Black Box block are available in the following dialog box.



Parameters used by this block are:

- **Update I/O and Simulation Mode and Board Information**:

  If this option is selected, Black Box will setup its input and output ports with proper names from the corresponding Gateway In and Gateway Out blocks. If “Disregard for Hardware Co-simulation” option in a Gateway block is checked, this Gateway will not be displayed in the Black box GUI

mask. Black Box will also update the following option **Hardware Co-simulation Mode** by retrieving the hardware co-simulation information from the Generator block. After the whole process is finished, the option returns to unchecked status for the next update.

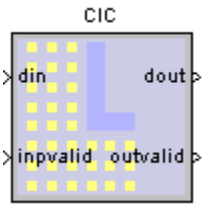
- **Hardware Co-simulation Mode**:

  When the previous option **Update I/O and Simulation Mode** is selected, Black Box will retrieve the hardware co-simulation information from the Generator block. If the option **Generate Bitstream** is checked in the Generator Block, this option will display “None”; if the option **Board Initialization and Verification** is checked in the Generator block, this option will display “Batch Mode”; if the option **Real-time Hardware Cosimulation** is checked in the Generator block, this option will display “Dynamic Mode”.

- **Automatically Connect Inputs**:

  If this option is checked, after the option **Update I/O and Simulation Mode** is selected, the Black Box inputs can be automatically connected to the corresponding Gateway In outputs. Black Box Outputs need to be manually connected to Simulink sinks by users.

CIC



Cascaded Integrator-Comb (CIC) filters, also known as Hogenauer filters, are used to achieve arbitrary and large sample rate changes in digital systems. These filters are used as decimation or interpolation filters and can be efficiently implemented without multipliers, utilizing only adders and subtractors.

A CIC filter is typically used in applications where the system sample rate is much larger than the bandwidth occupied by the signal. They are commonly used to build Digital Down Converters (DDCs) and Digital Up Converters (DUCs). Some applications that use the CIC filter include software designed radios, cable modems, satellite receivers, 3G base stations, and radar systems.

Lattice provides a widely parameterizable CIC filter that supports multiple channels with run-time programmable rates and differential delay parameters.

The CIC IP core has the following features:

- 1- to 32-bit input data width
- 1 to 8 cascaded stages
- 1 to 4 cycles differential delay, run-time programmable for both decimation and interpolation
- 2 to 16,384 decimation and interpolation sampling rate factor, run-time programmable rates for both decimation and interpolation
- Multi-channel (up to 4 channels) support for both decimation and interpolation

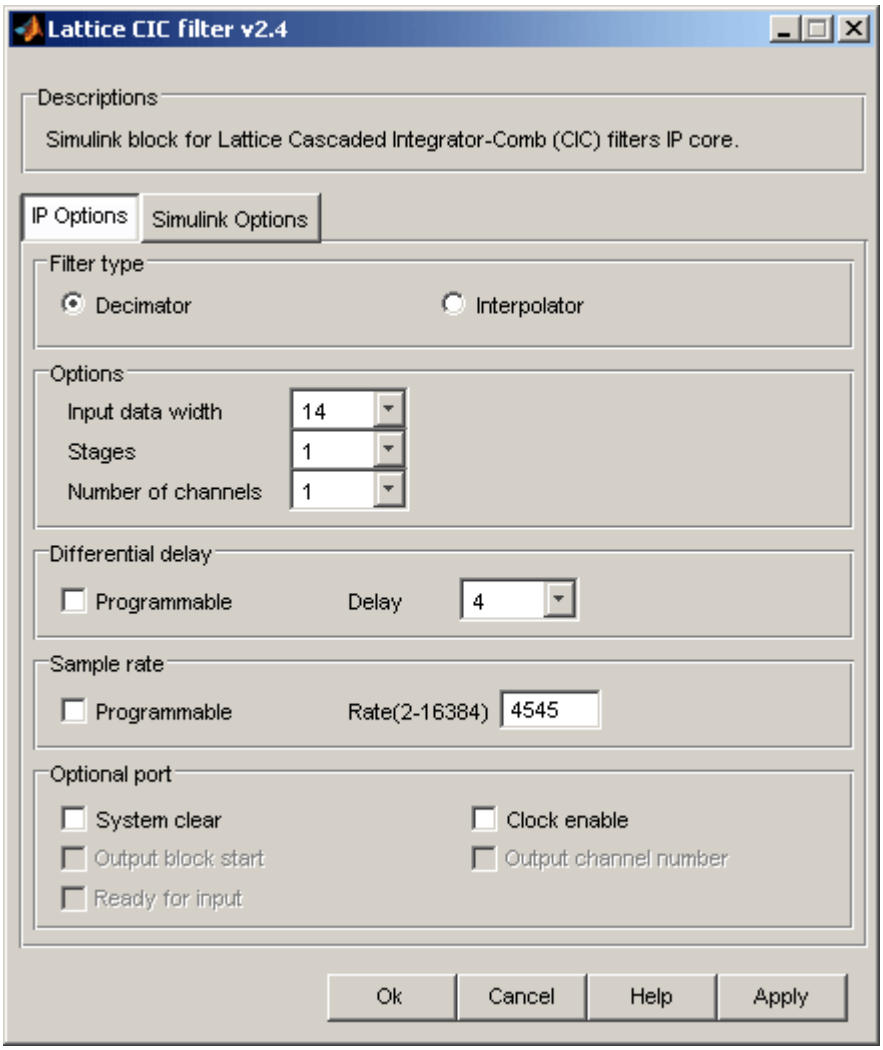
- Fully synchronous, single-clock design

Note

This IP core is available for evaluation in the MATLAB/Simulink environment. A netlist will be generated for simulation purposes. An IP license must be purchased to implement the IP into a Lattice FPGA. Information on all Lattice IP Modules, including data sheets, brochures, and downloads, can be found on the Lattice Semiconductor Corporation web site at the following URL:

http://www.latticesemi.com/products/intellectualproperty/index.cfm

The user configurable parameters for CIC Filter are available in the following dialog box.



IP Options Tab

- **Filter type**:

  Specifies whether the filter is a decimator or an interpolator.

- **Options**:

  Specifies input data with, filter stage, and channel number.

- **Differential delay**:
  - **Programmable**: Specifies whether this parameter is a variable.
  - **Delay (Max delay)**:

    Specifies the differential delay. If **Programmable** is selected, this specifies the max delay.

- **Sample rate**:
  - **Programmable**: Specifies whether this parameter is a variable.
  - **Rate (Max rate)**:

    Specifies the sample rate. If **Programmable** is selected, this specifies the max rate.

  - **Min rate**:

    Specifies the minimal value of sample rate when **Programmable** is selected.

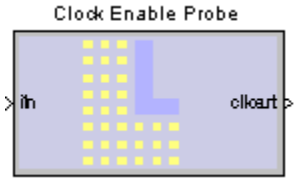
- **Optional port**: Specifies optional ports.

Simulink Options Tab

- **Manually connect to asynchronous reset port**:

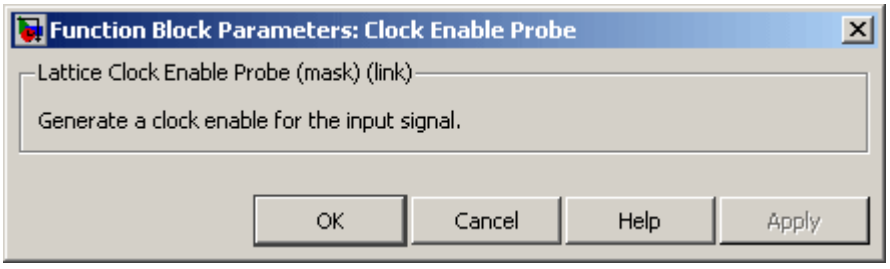
  Select this option to connect asynchronous reset port manually. Otherwise, the port will not be present.

Clock Enable Probe

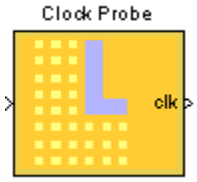


The Clock Enable Probe block generates a clock enable signal for the input signal. The output can then drive other Lattice blocks in order to generate control signals. The input can be connected to any signal, and will generate an unsigned single-bit output that will toggle at a rate corresponding to the sample rate of the input signal. If the input signal is at the system rate, the output will be permanently high.

The user configurable parameters for the Clock Enable Probe block are available in the following dialog box.



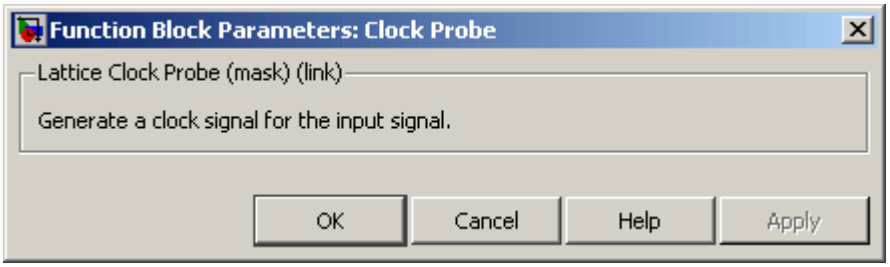
Clock Probe



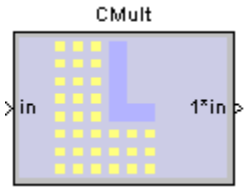
The Clock Probe block generates a clock signal for the input signal. The output is a double that takes on the value of either 0 or 1 depending on the state of the clock.

The user configurable parameters for the Clock Probe block are available in the following dialog box.

.

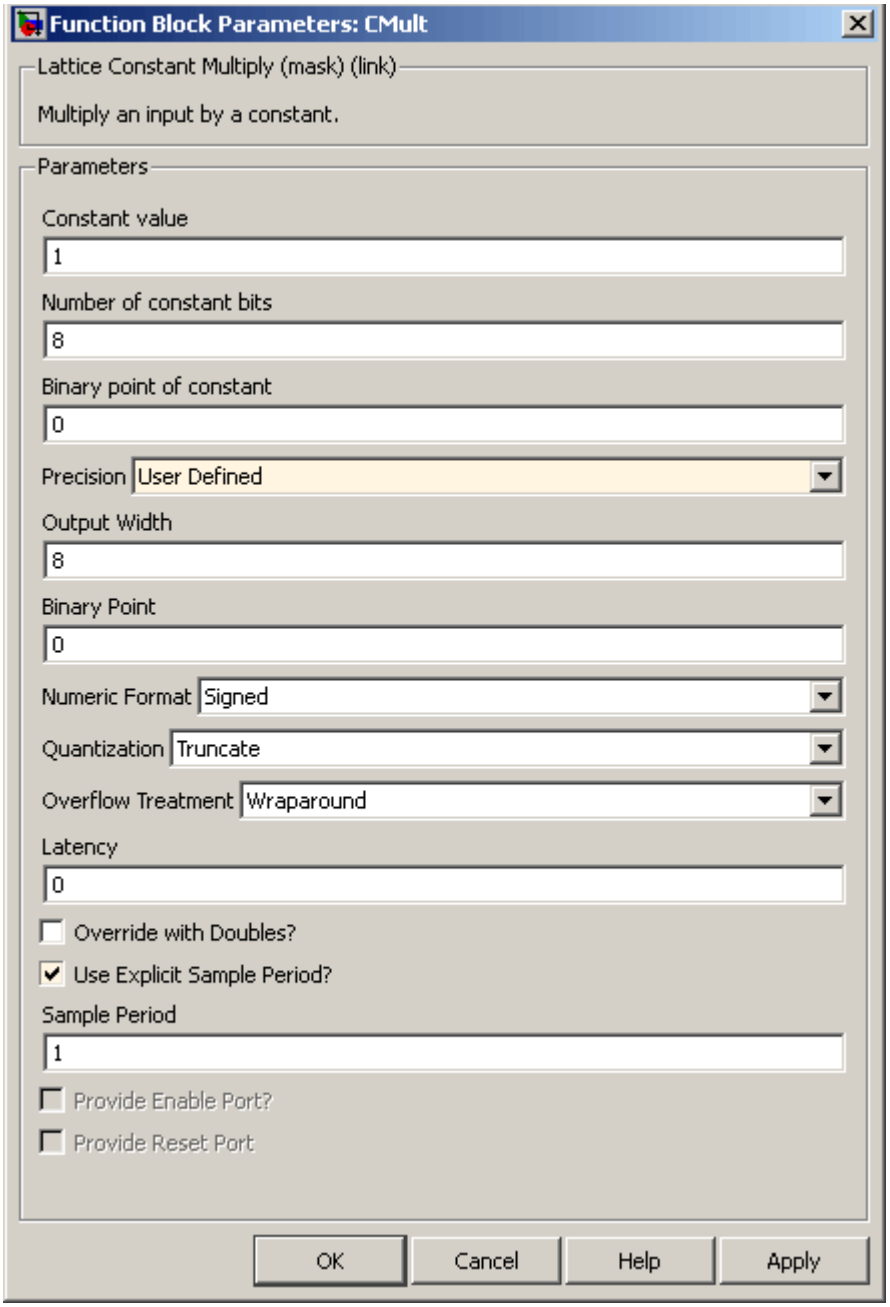


CMult



The Constant Multiply block multiplies the input by a specified constant value.

The user configurable parameters for the Constant Multiply block are available in the following dialog box.



Block parameters are:

- **Constant value**:

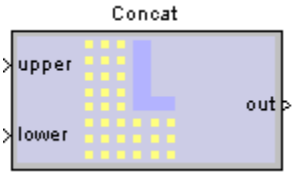
  This is a double precision number that will be converted to the specified format. If the value exceeds the range achievable by the selected format an error will be issued.

- **Number of constant bits**:

  A positive integer that specifies the number of bits to be used for the constant. If the value exceeds the number of bits specified, and error will be issued. Valid values are from 1 to 150.

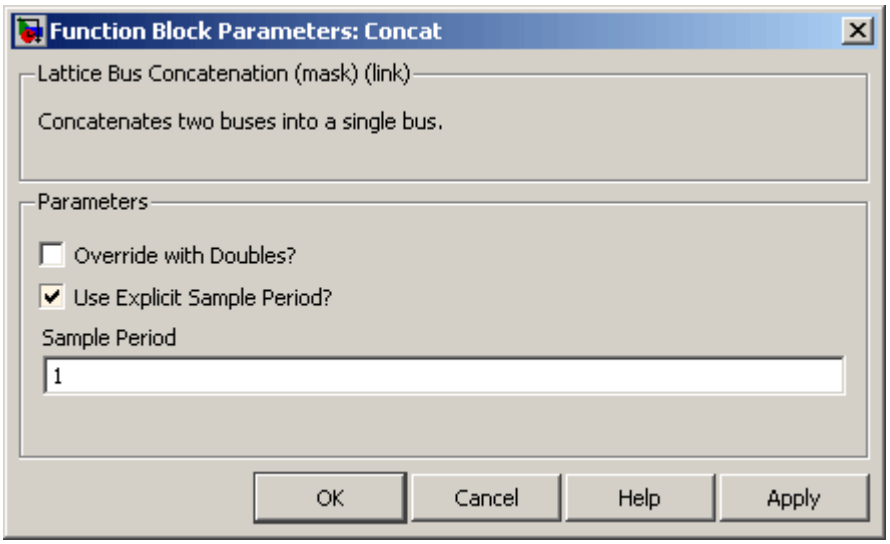
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Concat



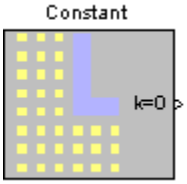
The Bus Concatenation block concatenates two busses into a single output bus. The inputs must be unsigned, and the binary points must be set to zero. The output will be an unsigned value with the binary point at zero.

The user configurable parameters for the Bus Concatenation block are available in the following dialog box.



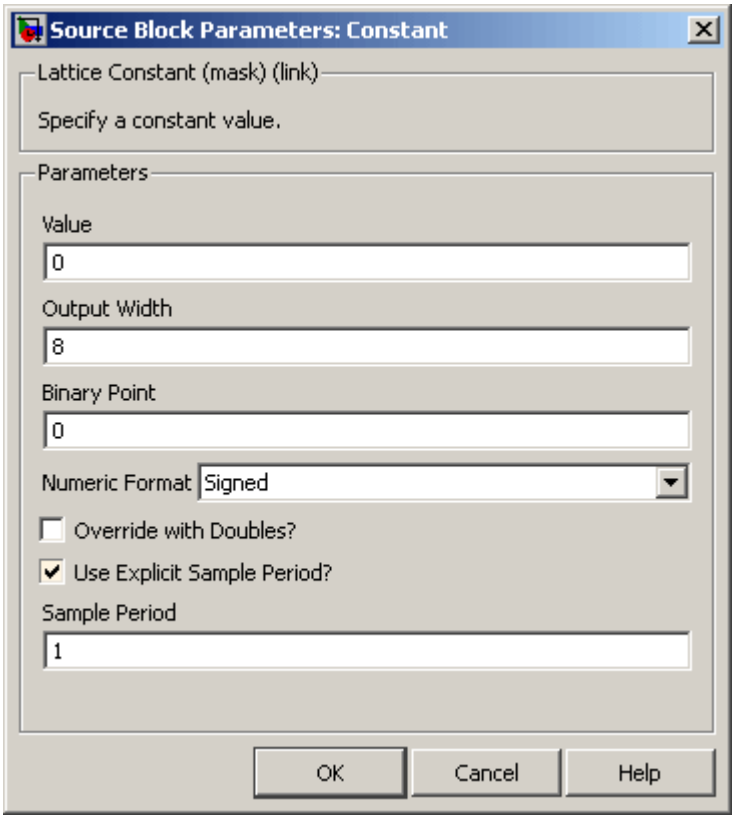
Parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Constant



The Constant block provides a constant value to other blocks. It must not be connected to a Gateway Out block.

The user configurable parameters for the Constant block are available in the following dialog box.



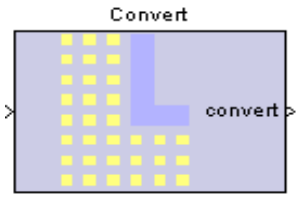
The block parameters are:

- **Value:**

  This is a double precision number that will be converted to the specified format. If the value exceeds the range achievable by the selected format an error will be issued.

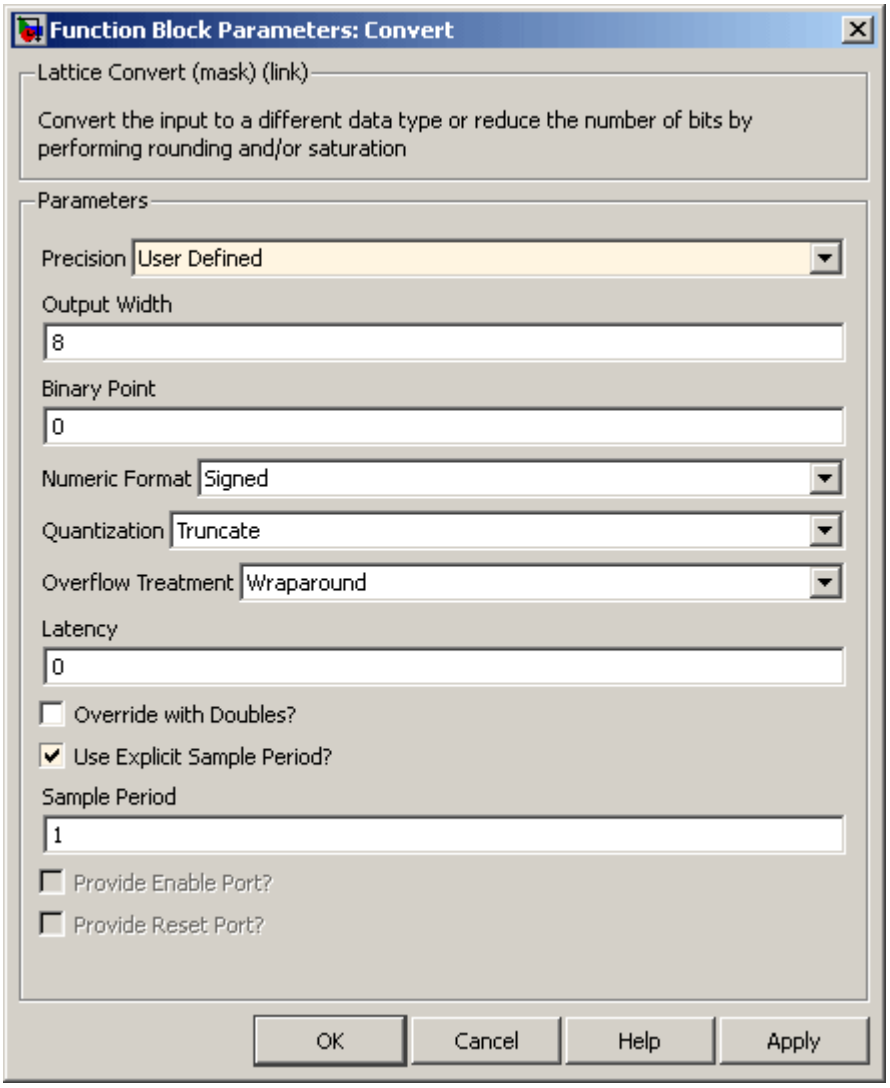
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Convert



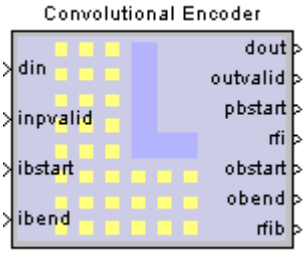
The Convert block converts from one format to another, performing rounding and/or saturation as required.

The user configurable parameters for the Convert block are available in the following dialog box.



See description of the parameters in Common Parameters for ispLeverDSP Blocks.

Convolutional Encoder



Lattice's Convolutional Encoder IP core is a parameterizable core for convolutional encoding of a continuous or burst input data stream. The core allows different code rates, constraint lengths and supports puncturing. It can operate in continuous or block mode, whichever is required by the channel. Either tail-biting or zero-flushing convolutional codes can be generated when the core works in the block mode. All these configurable parameters, including operation mode, generator polynomials, puncturing block size and puncturing pattern can be defined by the user to suit the needs of the application. Lattice's Convolutional Encoder IP is compatible with many networking and wireless standards that use convolutional encoding. It has the following features:

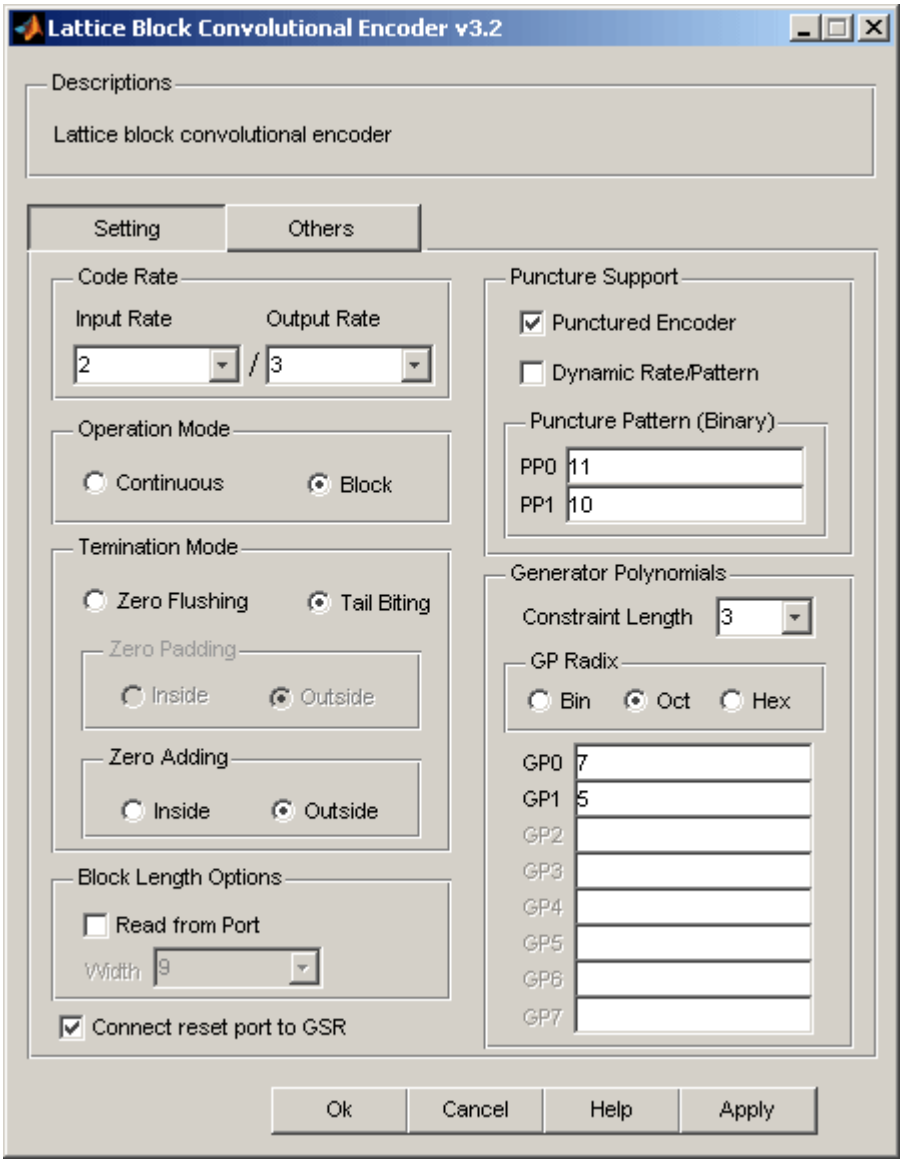
- Compatible with the Following Standards: IEEE 802.16-2004, IEEE 802.11a, 3GPP, 3GPP2 and DVB-S
- Supports Both Continuous and Block Encoding
- Variable Constraint Length from 3 to 9
- Supports Both Zero Flushing and Tail Biting Termination Modes
- Supports Both Internal and External Zero Padding in Zero Flushing Mode
- Supports Both Internal and External Tail Adding in Tail Biting Mode
- Supports a Wide Range of Programmable Code Rates (input_rate/ output_rate)
- User Defined Generator Polynomials
- Output Puncturing with Unrestricted, User Programmable Puncture Patterns
- Supports Dynamic Puncturing Mode, in Which Both the Code Rate and Puncture Patterns Can Be Varied through Ports
- Punctured Code Rate Can Be Programmed to k/n, Where k Can Be from 2 to 12 and n Can Be from k+1 to 2k-1; Additionally, Rate 1/2 is Supported in Dynamic Puncture Mode

Note

This IP core is available for evaluation in the MATLAB/Simulink environment. A netlist will be generated for simulation purposes. An IP license must be purchased to implement the IP into a Lattice FPGA. Information on all Lattice IP Modules, including data sheets, brochures, and downloads, can be found on the Lattice Semiconductor Corporation web site at the following URL:

http://www.latticesemi.com/products/intellectualproperty/index.cfm

The user configurable parameters for Block Convolutional Encoder are available in the following dialog box.



The block parameters:

- **Code Rate**: The code rate of convolutional encoder is expressed using two values: input rate (numerator of the code rate) and output rate (denominator of the code rate).
  - **Input Rate (Max Input Rate)**:

    This defines the input symbol rate for the encoder. The input rate for non-punctured encoder is always 1. For punctured encoders with fixed puncture rate, the input rate can be any value from 2 to 12.

    If “Dynamic Rate/Pattern” under “Puncture Support” is checked, it specifies a dynamic puncturing encoder. In this mode, the input rate is applied through a port. The parameter “Input Rate” is replaced by parameter “Max Input Rate” automatically. The parameter “Max Input Rate” defines the maximum value for the input rate and can be set to any value between 1 and 12.

- **Output Rate (Max Output Rate)**:

  This defines the output symbol rate for the encoder. The output rate for non-punctured encoder can be any value between 2 and 8. For fixed puncturing encoder, the output rate can be any value from k+1 to 2k-1, where k is the input rate.

  The parameter “Output Rate” is changed to “Max Output Rate,” when “Dynamic Rate/Pattern” under “Puncture Support” is checked. It defines the maximum value of the output rate when it is varied through the port. The Max Output Rate is from mir+1 to 2*mir-1, where mir is the value of Max Input Rate. It is always set as 2 when Max Input Rate is 1.

- **Puncture Support**:

  The encoder supports punctured or non-punctured data. A punctured encoder is generated if “Punctured Encoder” is checked in the GUI. The encoder can perform either fixed puncturing or dynamic puncturing, which is decided by the “Dynamic Rate/Pattern” option. For a fixed puncturing encoder (the option is unchecked), the puncture pattern composed of PP0 and PP1 is defined by the user. The total number of 1’s in both puncture patterns must equal the output rate. For a dynamic puncturing encoder, the puncture pattern is set through the input ports of the IP core.

- **Operation Mode**:

  The user selects either continuous or block encoding mode.

- **Termination Mode**:

  When the operation mode is “Block,” the user can select either “Zero Flushing” or “Tail Biting” termination mode. In Zero Flushing mode, zeros are padded at the end of the input data block. In Tail Biting mode, the encoder is initialized with the tail of the block.

  - **Zero Padding Mode**:

    When the Zero Flushing termination mode is used, the user can select either “Inside” padding or “Outside” padding. In “Inside” padding, K-1 zeros are added to the data stream by the IP core itself. In the “Outside” padding mode, the user needs to terminate a block with appropriate number of zeros.

  - **Tail Adding Mode**:

    When the Tail Biting termination mode is used, the user can select either “Inside” adding or “Outside” adding. If “Inside” mode is chosen, the core stores the entire block of data and uses the last k-1 data samples to initialize the state of the encoder, to result in a tail-biting code. If “Outside” mode is chosen, the user is expected to add the tail of the block to its beginning, and then supply the appended block at the input of the encoder.

- **Block Length Options**: For the block encoding, the block length can be set in one of the following two ways:

  - **Block length read from the port**:

    If “Read from Port” is checked, the user sets the block length through the blklen port. The width of the blklen port can be from 4 to 16 for

zero-flushing termination block, and from 4 to 9 for tail-biting termination block;

- **Duration defined by signals ibstart and ibend**:

  When “Read from Port” option is unchecked, the start and end of a block is defined by signal ibstart and ibend. If both “Tail Biting” and “Tail Adding Inside” options are selected for the core, the block length is always read through blklen port.

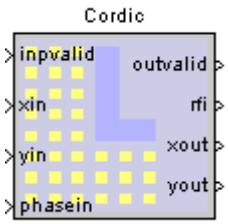
- **Generator Polynomials**:

  GP0, GP1, GP2, GP3, GP4, GP5, GP6 and GP7 are generator polynomials. For non-puncturing encoders, the number of generator polynomials is always equal to the output rate. For puncturing encoders, the number of generator polynomials is 2. The polynomial values can be provided in any of the three radixes: binary (“Bin”), octal (“Oct”), or hexadecimal (“Hex”).

  - **Constraint Length**:

    This defines the constraint register length. The value can be any integer from 3 to 9.

Cordic



The Lattice CORDIC (Coordinate Rotation Digital Computer) IP is configurable and several functions can be implemented in the IP core: Rotation, Translation, Sin and Cos, and Arctan. Two architecture configurations are available for the arithmetic unit: Parallel, with single cycle data throughput, and Word-serial, with multiple cycles throughput. The input data, output data widths, and iterative number are configurable over a wide range. The IP core uses full internal precision while allowing variable output precision with several choices for rounding.

The CORDIC core has the following features:

- Functional configurations
  - Vector rotation (polar to rectangular)
  - Vector translation (rectangular to polar)
  - Sin and Cos
  - Arctan
- Input data widths from 8 to 32 bits
- Iterative number from 4 to 32
- Optional pre-rotation module
- Optional amplitude compensation scaling module to compensate for CORDIC algorithm’s output amplitude scale factor
- Selectable rounding: truncation, rounding up, rounding away from zero, and convergent rounding

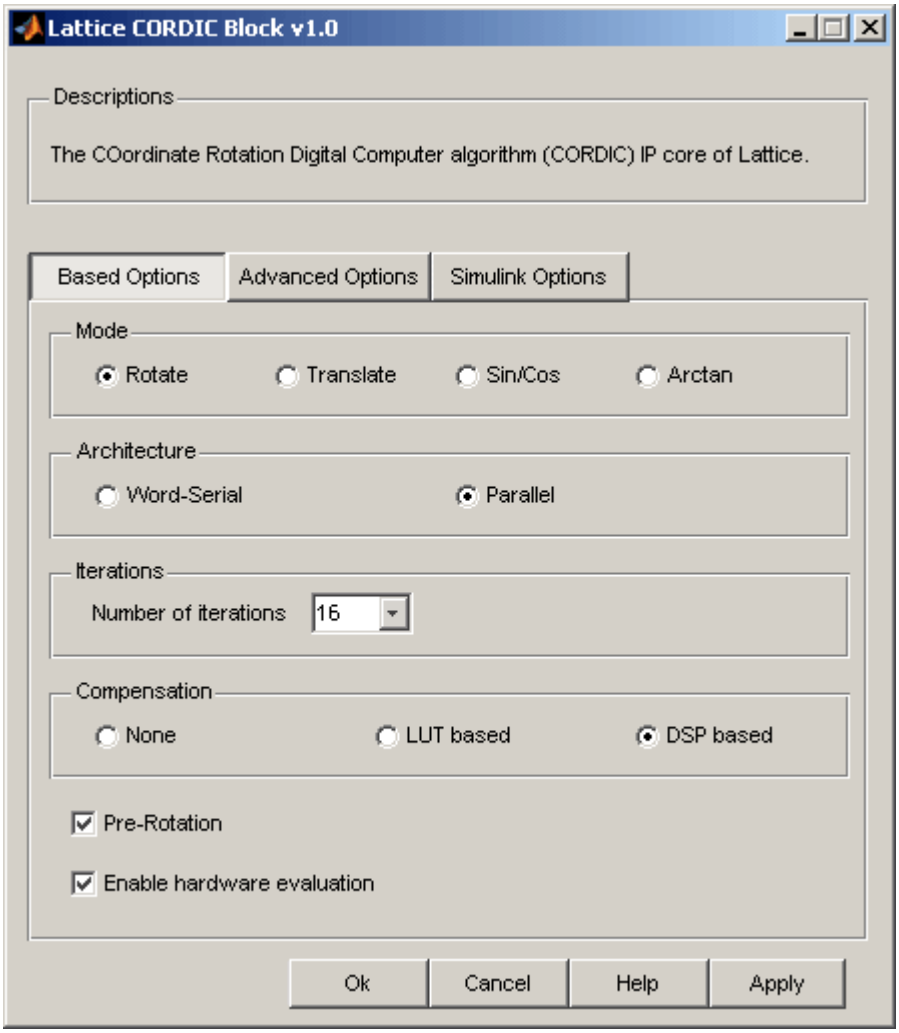
- Parallel architectural configuration for high throughput
- Word serial architectural configuration for small area
- Signed 2's complement data
- Optional control signals: clock enable (ce) and synchronous reset (sr)
- Full precision arithmetic

Note

This IP core is available for evaluation in the MATLAB/Simulink environment. A netlist will be generated for simulation purposes. An IP license must be purchased to implement the IP into a Lattice FPGA. Information on all Lattice IP Modules, including data sheets, brochures, and downloads, can be found on the Lattice Semiconductor Corporation web site at the following URL:

http://www.latticesemi.com/products/intellectualproperty/index.cfm

The user configurable parameters for the CORDIC block are available in the following dialog box.



Based Options Tab

Filter specifications

- **Mode**: Specifies the operation mode.
- **Architecture**: Specifies the implement method.
- **Iterations**:

  Specifies the number of iterations. Value range: 4 to 32.
- **Compensation**: Specifies one of the following compensation method.
  - **None**: No compensation.
  - **LUT based**: Uses LUT based multiplier to do the compensation.
  - **DSP based**: Uses sysDSP block to do the compensation.
- **Pre-Rotation**: Specifies whether pre-rotation is used.
- **Enable hardware evaluation**: For hardware evaluation use.

Advanced Options Tab

Data Width

- **Input data width**:

  Specifies input data width. Value range: 8 to 32. Default: 16.
- **Output width**:

  Specifies output data width. Value range: 8 to 32. Default: 16.
- **Rounding method**:

  Four types of rounding method are provided: Truncation, Rounding up Rounding away from zero, and Convergent rounding.
- **Option ports**:

  Specifies if a synchronous reset port or a clock enable port is needed in the IP.
- **Synthesis options**: Specifies synthesis options.
- **Frequency constraints**:

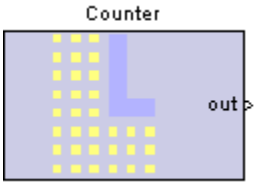
  Specifies frequency constraints for synthesis use. Unit is MHz.
- **Pipeline & Retiming**: Do not recommend.

Simulink Options Tab

- **Manually connect to asynchronous reset port**:

  Select this option to connect asynchronous reset port manually. Otherwise, the port will not be present.

Counter



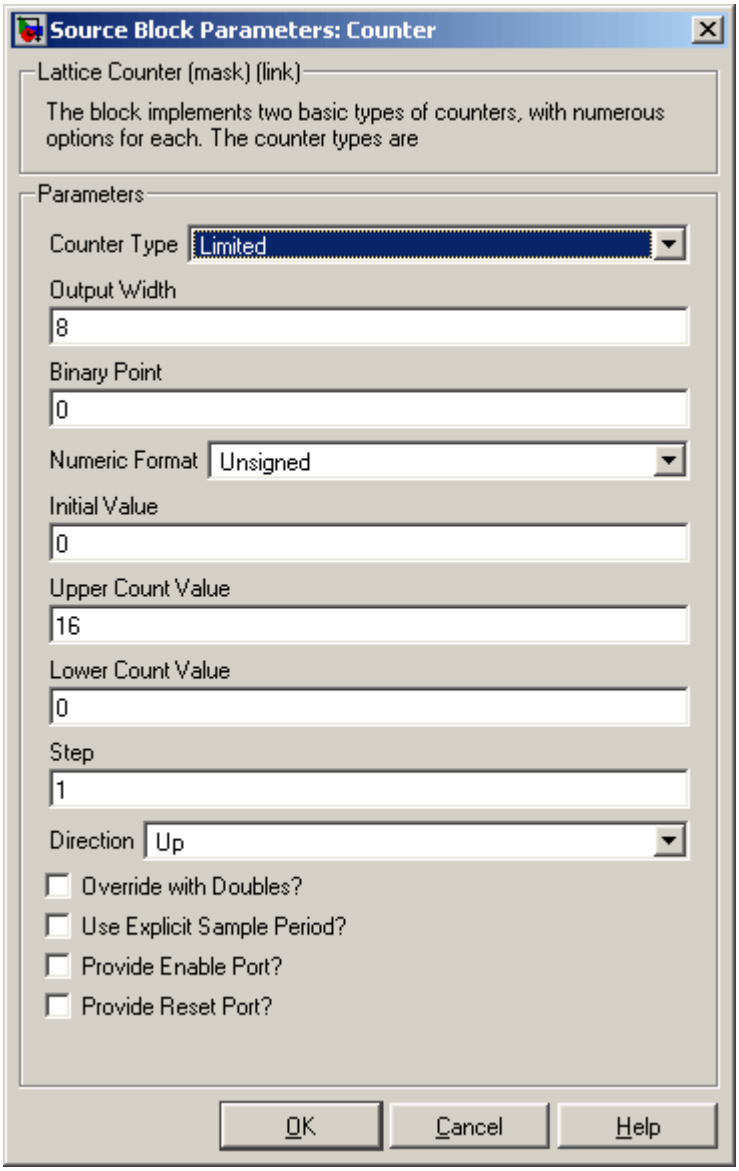
The Counter block provides a count function. There are two basic types of counters, either a limited counter or a modulo counter. Both counter modes support an arbitrary step value; loading from an external source; resetting to an initial value; and either up or down counting.

The limited counter counts between two specified count values. When the upper value is reached, the counter is reset to the lower limit value. In this mode, the difference between the upper and lower values must be an evenly divisible by the step.

The modulo counter counts by a specified modulo. If the modulo is a 2^n value, then the hardware required is minimized, with a resulting higher performance. In this mode, the difference between the upper and lower values will be the modulo-1 for the counter.

The counter can implement practical counters up to 52 bits in length. This is due to the fact that the parameters are of type double and are limited to 52 bits of precision.

The user configurable parameters for the Counter block are available in the following dialog box.



Block parameters are:

- **Counter Type = [Limited, Modulo]**:

  You can select from either a limited counter or a modulo counter.

- **Initial Value**:

  This specifies the starting value of the counter and the value that the counter is reset to if a reset port is selected and active. This is a double precision number that will be converted to the specified format. If the value exceeds the range achievable by the selected format an error will be issued. For limited counting, this value must be an even multiple of the count range. Otherwise an error will be issued.

- **Upper Value**:

  This specifies the upper count value of the counter. This is a double precision number that will be converted to the specified format. If the

value exceeds the range achievable by the selected format an error will be issued. For limited counting, this value must be an even multiple of the count range. Otherwise an error will be issued.

- **Lower Value**:

  This specifies the lower count value of the counter. This is a double precision number that will be converted to the specified format. If the value exceeds the range achievable by the selected format an error will be issued.

- **Step**:

  This specifies the count increment. This is a double precision number that will be converted to the specified format. If the value exceeds the range achievable by the selected format an error will be issued.

- **Direction = [Up, Down]**:

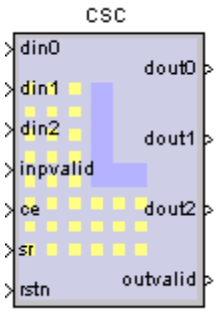
  This specifies the count direction. This is a double precision number that will be converted to the specified format. If the value exceeds the range achievable by the selected format an error will be issued.

- **Provide Load Port?**:

  Check this box to provide a load and data input pin to the block. When the load pin goes high, the data on the data in pin will be loaded into the counter. If an enable pin is provided, it must also be high in order to load the data. This selection in only available for the modulo counter.

Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

CSC



Lattice's Color Space Converter (CSC) IP core converts signals from one color space to another color space. This capability is often required to ensure compatibility with display devices or to make the image data amenable for compression or transmission. CSC is used in video and image display systems including televisions, computer monitors, color printers, video telephony and surveillance systems. CSC is also used in the implementation of many standards-based video/image compression and processing applications, including NTSC/PAL/SECAM television standards, JPEG, and MPEG systems.

The Lattice CSC IP core is widely parameterizable and can support any custom color space conversion requirement. Furthermore, several commonly used color space conversion methods are provided as ready-to-use configurations.

The Lattice CSC IP core supports all Lattice FPGA devices. You can refer to the Lattice Color Space Converter IP Core User Manual on Lattice web site for more detailed information.

The Lattice Color Space Converter IP Core has the following features:

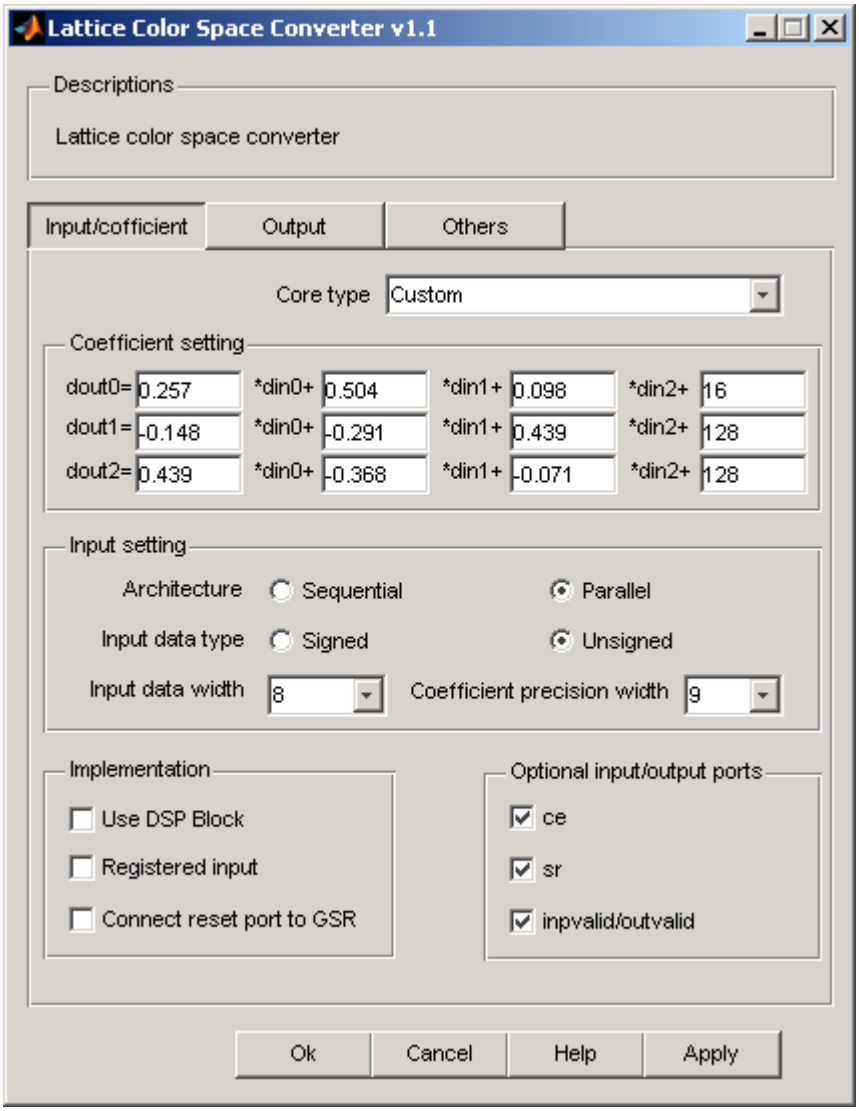
- Input data width of 8, 10, 12, and 16 bits
- Signed or unsigned input data
- Supports standard configurations as well as custom configurations
- Precision of parameterized coefficients from 9 to 18 bits
- Full precision as well as limited precision output
- Programmable precision and rounding options for the output
- Option for sequential or parallel architecture for area or throughput optimization
- Configurable sysDSP block-based or look-up-table (LUT) based multiplier implementations
- Registered input option available for input set-up time improvement

Note

This IP core is available for evaluation in the MATLAB/Simulink environment. A netlist will be generated for simulation purposes. An IP license must be purchased to implement the IP into a Lattice FPGA. Information on all Lattice IP Modules, including data sheets, brochures, and downloads, can be found on the Lattice Semiconductor Corporation web site at the following URL:

http://www.latticesemi.com/products/intellectualproperty/index.cfm

The user configurable parameters for Color Space Converter are available in the following dialog box.



Input/coefficient Tab

- **Core Type**: Selection between custom and pre-defined standard configurations. The following pre-defined standard configurations are available.
  - Computer R'G'B' <-> Y'CbCr: SDTV
  - Computer R'G'B' <-> Y'CbCr: HDTV
  - Studio R'G'B' <-> V'CbCr: SDTV
  - Studio R'G'B' <-> V'CbCr: HDTV
  - Computer R'G'B' <-> V'UV
  - Computer R'G'B' <-> Y'IQ
  - Y'IQ -> Y'UV
- **Architecture**: Selection between parallel and sequential implementation.

- **Input Data Type**: Signed or unsigned input data type.
- **Input Data Width**: inheritance, 8, 10, 12, 16
- **Coefficient Precision Width**:

  Number of bits used to convert floating point coefficients into fixed point. Range is 9, 10, ..., 18.

- **Use DSP Block**:

  If this option is checked then core will use the sysDSP Block for implementation.

- **Registered Input**:

  If this option is selected, the inputs are registered. The setup time of core inputs will be improved by registering the inputs. This option is useful when the input data is provided on the device pins.

- **Connect reset port to GSR**:

  If this option is checked, the GSR is instantiated and used to route the CSC's rstn input. Using GSR improves the utilization and performance of the CSC IP. However, if GSR is used, an active input in rstn will reset most of the FPGA components as well. This option must be checked to enable the hardware evaluation capability for this IP.

- **Optional Input Ports**:
  - **ce**: Input port ce is added to the core when checked.
  - **sr**: Input port sr is added to the core when checked.
  - **inpvalid/outpvalid**:

    Input ports inpvalid and outvalid are added to the core when checked. If architecture is selected as Sequential, then inpvalid/outvalid is always checked. If architecture is selected as Parallel, then this is optional.

Output Tab

- **Output Precision**: Full precision or limited precision for the output data.
- **Output Width**: 5-15 for limited output precision and 19 for full output precision.
- **Overflow**:

  This parameter is available when some MSBs have to be removed from the result for the limited precision output. If the overflow option is "Truncation", the excess MSBs are removed and the remaining LSBs are not changed. If saturation option is selected, the output is saturated based on the removed MSBs.

- **Underflow**:

  This parameter is available when some LSBs have to be removed from the result for the limited precision output. If the underflow option is "Truncation", the excess LSBs are removed and the remaining output is not affected by the removed LSBs. If rounding option is selected, the output is rounded based on the removed LSBs.

- **Starting LSB (0-13)**:

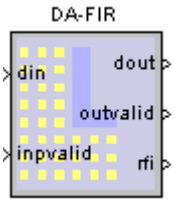
  This parameter defines the starting least significant bit of the full precision output that becomes LSB of the limited precision output. This parameter gets enabled only when output precision is selected as limited.

Others Tab

- **Manually connect reset port**:

  Select this option to connect asynchronous reset port manually. Otherwise, the port will not be present.

DA_FIR



The Lattice DA-FIR (Distributed Arithmetic Finite Impulse Response) Filter IP core is a widely configurable, multi-channel FIR filter. It uses distributed arithmetic algorithms implemented in FPGA Look Up Table (LUT) or Embedded Block Memory (EBR) to efficiently support the sum-of-product calculations required to perform the filter function.

In addition to single rate filters, the IP core also supports a range of polyphase interpolation and decimation filters. The DA-FIR Filter IP core supports as high as 32 channels, with each having up to 1024 taps. The input data, coefficient, and output data widths are configurable over a wide range. The IP core uses full internal precision while allowing variable output precision with several choices for saturation and rounding.

The DA-FIR IP core has the following features:

- Variable number of taps up to 1024
- Multi-channel support (up to 32 channels)
- Polyphase interpolating/decimating filters
- Halfband filters
- Interpolation and Decimation ratios from 2 to 32
- Input data widths from 4 to 32 bits
- Coefficient widths from 4 to 32 bits
- Signed or unsigned data and coefficients
- Selectable rounding: truncation, rounding away from zero, convergent rounding
- Optional saturation logic for overflow handling
- Full precision arithmetic
- Specification of fractional inputs and outputs
- Support for both serial and parallel filters, with user specified degree of parallelism
- Configurable pipelining to increase performance
- Optimizations based on filter characteristics (symmetry and halfband)

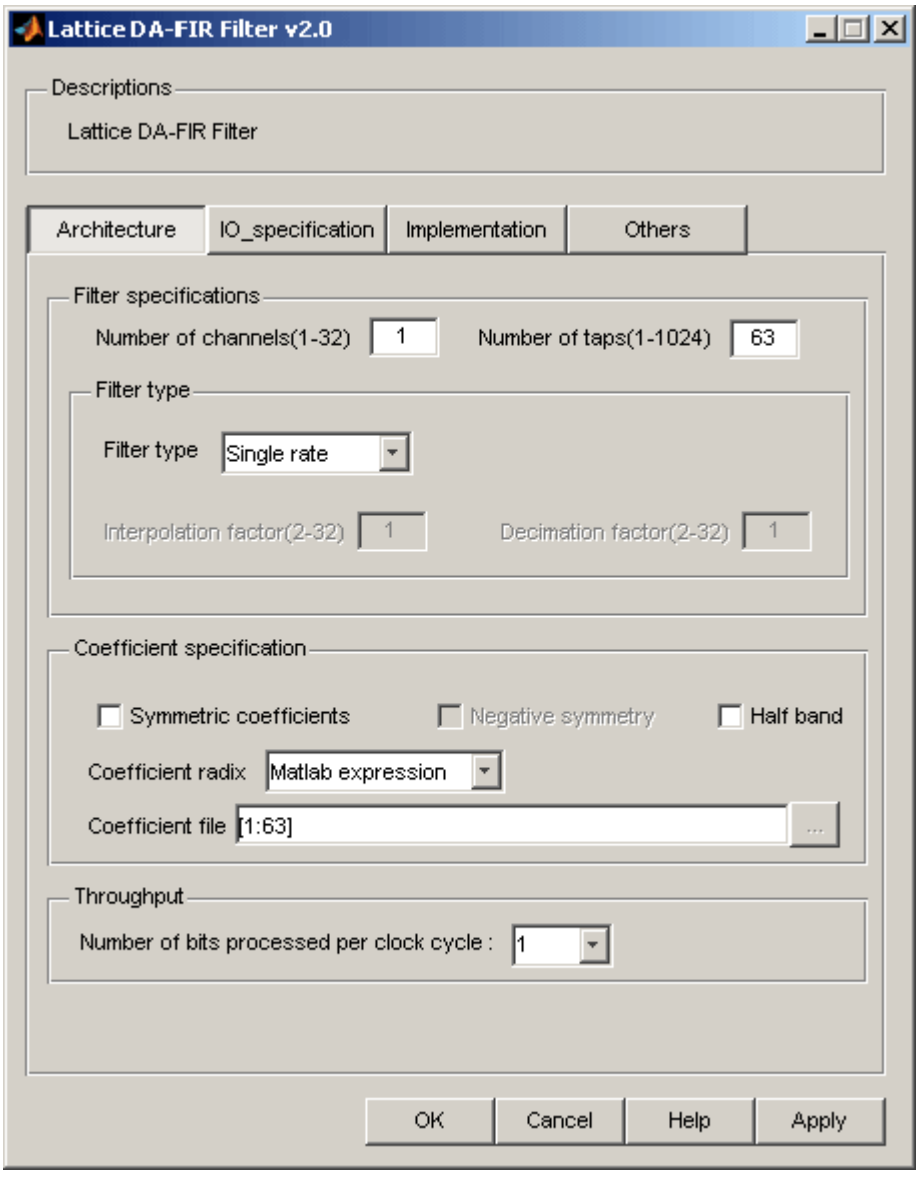
- Handshake signals to facilitate smooth interfacing

Note

This IP core is available for evaluation in the MATLAB/Simulink environment. A netlist will be generated for simulation purposes. An IP license must be purchased to implement the IP into a Lattice FPGA. Information on all Lattice IP Modules, including data sheets, brochures, and downloads, can be found on the Lattice Semiconductor Corporation web site at the following URL:

http://www.latticesemi.com/products/intellectualproperty/index.cfm

The user configurable parameters for DA-FIR Filter are available in the following dialog box.



Architecture Tab

Filter specifications:

- **Number of channels**: Specifies the number of channels.
- **Number of taps**: Specifies the number of taps.
- **Filter type**:

  Specifies whether the filter is single rate, interpolate, or decimator.
- **Interpolator factor**: Specifies the value of the interpolation factor.
- **Decimator factor**: Specifies the value of the decimation factor.

Coefficient specification:

- **Symmetric coefficients**:

  Specifies whether the coefficients are symmetric. If this is selected, only one half of the number of coefficients (if number of taps is odd, the half value is rounded to the next higher integer) is read from the initialization files.
- **Negative symmetry**:

  If this is selected, the coefficients are considered to be negative symmetric. The second half of the coefficients are made equal to the negative of the corresponding first-half coefficients.
- **Half band**:

  Specifies whether a half band filter is realized. If this is selected, only one half of the number of coefficients (if the number of taps is odd, the half value is rounded to the next higher integer) is read from the initialization file.
- **Coefficient radix**:

  Radix for the coefficients in the coefficients file. For floating point radix and Matlab expression, the coefficients can be signed or unsigned floating point value. For decimal radix, the negative values have a preceding unary minus sign. For hexadecimal (Hex) and binary radices, the negative values must be written in two’s complement form using exactly as many digits as specified by the coefficients width parameter.
- **Coefficient file**:

  Specifies the name and location of the coefficients file or Matlab expression.

Throughput:

- **Number of bits processed per clock cycle**:

  Number of bits processed at a time. This defines the degree of parallelism. This value must be a factor of input data width for non-symmetric architecture, or input data width+1 if it is symmetric.

IO_specification Tab

Data:

- **Input data type**:

  Specifies the input data type as signed or unsigned. If the type is signed, the data is interpreted as a two's complement number.

- **Input data width**: Width of input data. Value range: 4 to 32. Default: 16.

- **Input data binary points position**:

  Specifies the location of the binary point in the input data. This number specifies the bit position of the binary point from the LSB of the input data. If the number is zero, the point is right after LSB; if positive, it is to the left of LSB and if negative, it is to the right of LSB. Value range: -2 to <*Input data width*> + 2. Default: 0.

Coefficient:

- **Coefficient type**:

  Specifies the coefficients type as signed or unsigned. If the type is signed, the coefficient data is interpreted as a two's complement number.

- **Coefficient width**: Coefficient width. Value range: 4 to 32. Default:16.

- **Coefficient binary points position**:

  Specifies the location of the binary point in the coefficients. This number specifies the bit position of the binary point from the LSB of the coefficients. If the number is zero, the point is right after LSB; if positive, it is to the left of LSB and if negative, it is to the right of LSB. Value range: -2 to <*Coefficient width*> + 2. Default: 0.

Output:

- **Output width**:

  Specifies output width. The maximum full precision output width is defined by <*Max output width*> = <*Input data width*> + <*Coefficients width*> +ceil(Log2(<*Number of taps*>)). The core's output is usually a part of the full precision output equal to the **Output width** and extracted based on the different binary point position parameters.

  The format for the internal full precision output is displayed as a static text next to the Output width control in the GUI. The format is displayed as `W.F`, where `W` is the full precision output width and `F` is the location of the binary point from the LSB of the full precision output, counted to the left.

  Value range: 4 to <*Max output width*>. Default: <*Full precision output width*>.

- **Output binary point position**:

  This number specifies the bit position of the binary point from the LSB of the actual core output. If the number is zero, the point is right after LSB; if positive, it is to the left of LSB and if negative, it is to the right of LSB. This number, together with the parameter **Output width** determines how the actual core output is extracted from the true full precision output. The precision control parameters **Overflow** and **Rounding** are applied

respectively when MSBs and LSBs are discarded from the true full precision output.

Value range: (4 + *<Input data binary point position>* + *<coefficient binary point position>* – *<Max output width>*) to (*<Output width>* + *<Input data binary point position>* + *<Coefficient binary point position>* – 4). Default: 0.

Precision control:

- **Overflow**:

  Specifies what kind of overflow control is to be used. This parameter is available whenever there is a need to drop some of the MSBs from the true output. If the selection is **Saturation**, the output value is clipped to the maximum, if positive or minimum, if negative, while discarding the MSBs. If the selection is **Wrap-around**, the MSBs are simply discarded without making any correction.

- **Rounding**:

  Specifies the rounding method when there is a need to drop one or more LSBs from the true output.

Implementation Tab

Memory type:

- **Coefficient memory type**:

  Selects the type of memory that is used for storing the coefficients. If the selection is **EBR**, EBR memories are used for storing the coefficients. If the selection is **Distributed**, distributed memories are used for storing coefficients.

- **Input buffer type**: Memory type for the input buffer.

- **Output buffer type**: Memory type for the output buffer.

Performance:

- **Area(1) – Speed(4)**:

  Specifies the optimization level for speed. The optimization level of performance can be specified from 1 to 4. The adder trees and accumulators are implemented as pipelined adder. Each input word of pipelined adder is split into one slice when level 2 is specified, two slices when level 3 is specified, and four slices when level 4 is specified. When level 1 is specified, there is no register in the pipelined adder for low performance application.

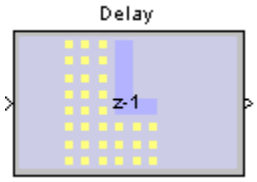
  The default optimization level is 2.

Others Tab

- **Manually connect to asynchronous reset port**:

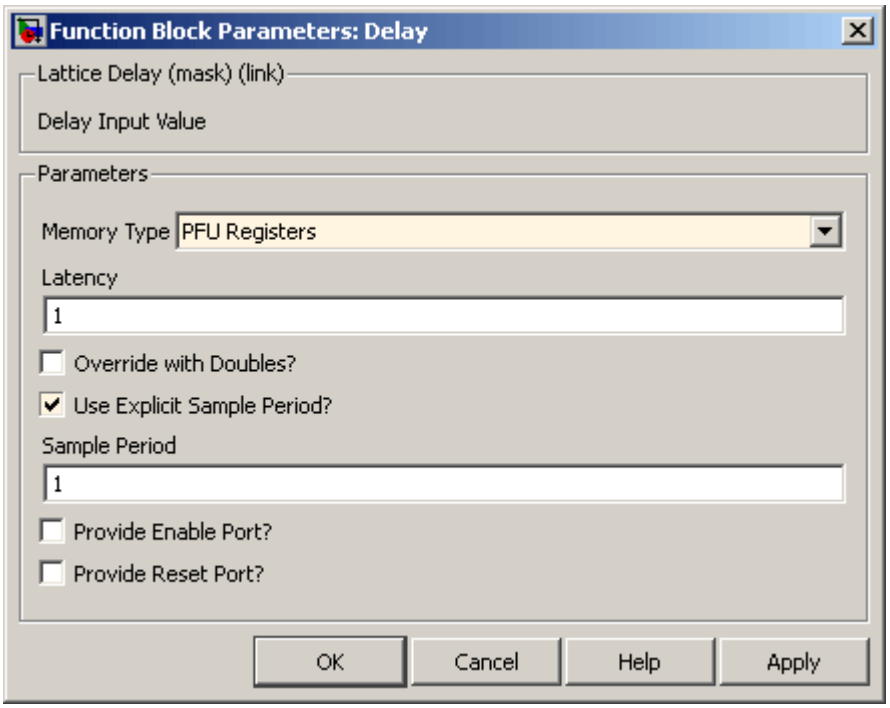
  Select this option to connect asynchronous reset port manually. Otherwise, the port will not be present.

Delay



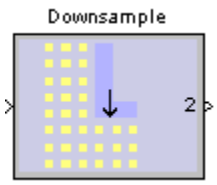
The Delay block provides a selectable latency of from 1 to 20 clock cycles. The initial value of the delay registers will be zeroes.

The user configurable parameters for the Delay block are available in the following dialog box.



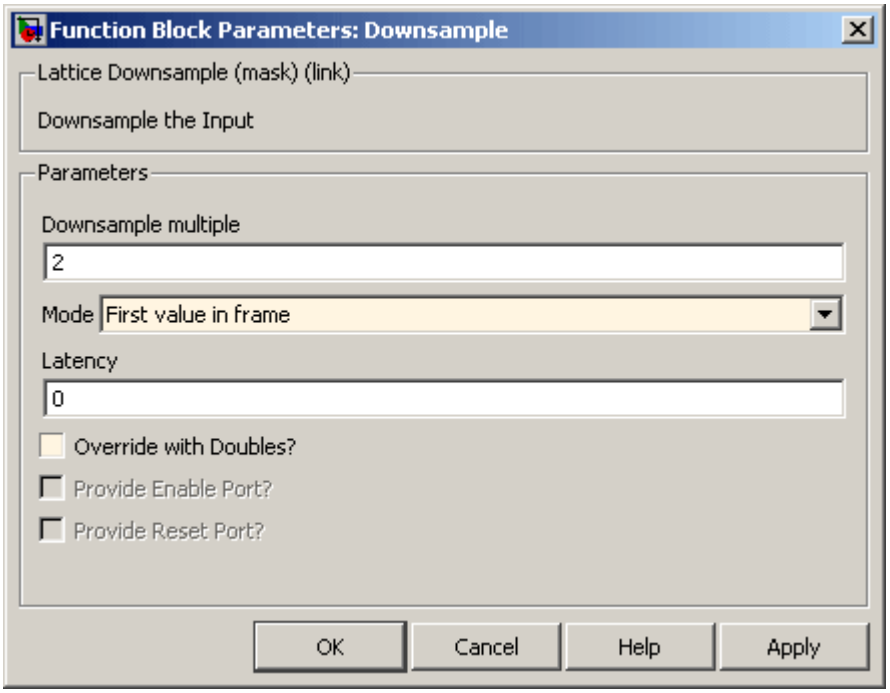
See description of the parameters in Common Parameters for ispLeverDSP Blocks.

Downsample



The Downsample block scales the input sample rate provided by the Downsample multiple.

The user configurable parameters for the Downsample block are available in the following dialog box.



Block parameters are:

- **Downsample multiple**:

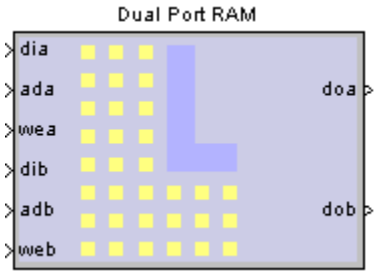
  This value is a positive integer greater than or equal to 2 that is the input rate divided by the output rate.

- **Mode**:

  Selection between First value in Frame and Last value in frame. Only “First value in frame” is currently supported. In this mode, the first of “n” sequential input values is repeated at the output, where “n” is the downsample multiple.

Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Dual Port RAM



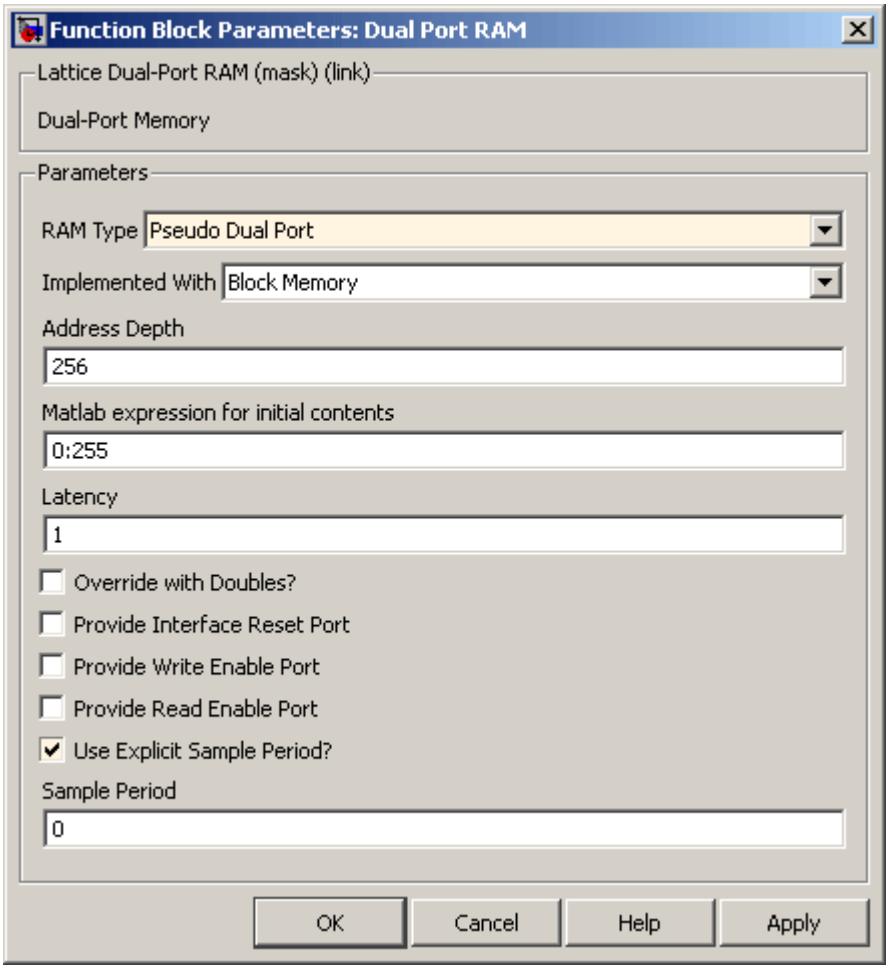
The Dual Port Ram block provides two implementations:

- Pseudo-Dual Port RAM
- True Dual Port RAM

For more detailed information, see technical notes of specific device architecture on Lattice web site.

Pseudo Dual Port RAM

The user configurable parameters for the Pseudo Dual Port RAM block are available in the following dialog box.



Block parameters are:

- **Ram Type**:

  Choose from Pseudo Dual Port RAM or True Dual Port RAM.

- **Implemented With**:

  Choose between Block Memory or Distributed Memory.

- **Address Depth**: Depth of memory.

- **MATLAB Expression for Initial Contents**:

  Initial value for the memory data. This can be any legal MATLAB expression that produces a vector of the appropriate length. If the result of the evaluation is less than the number of words, the remaining words will be set to zero. If it is larger than the number of words, the excess values will be ignored.

- **Provide Interface Reset Port?**:

Select or deselect. Deselect always set to constant 0.

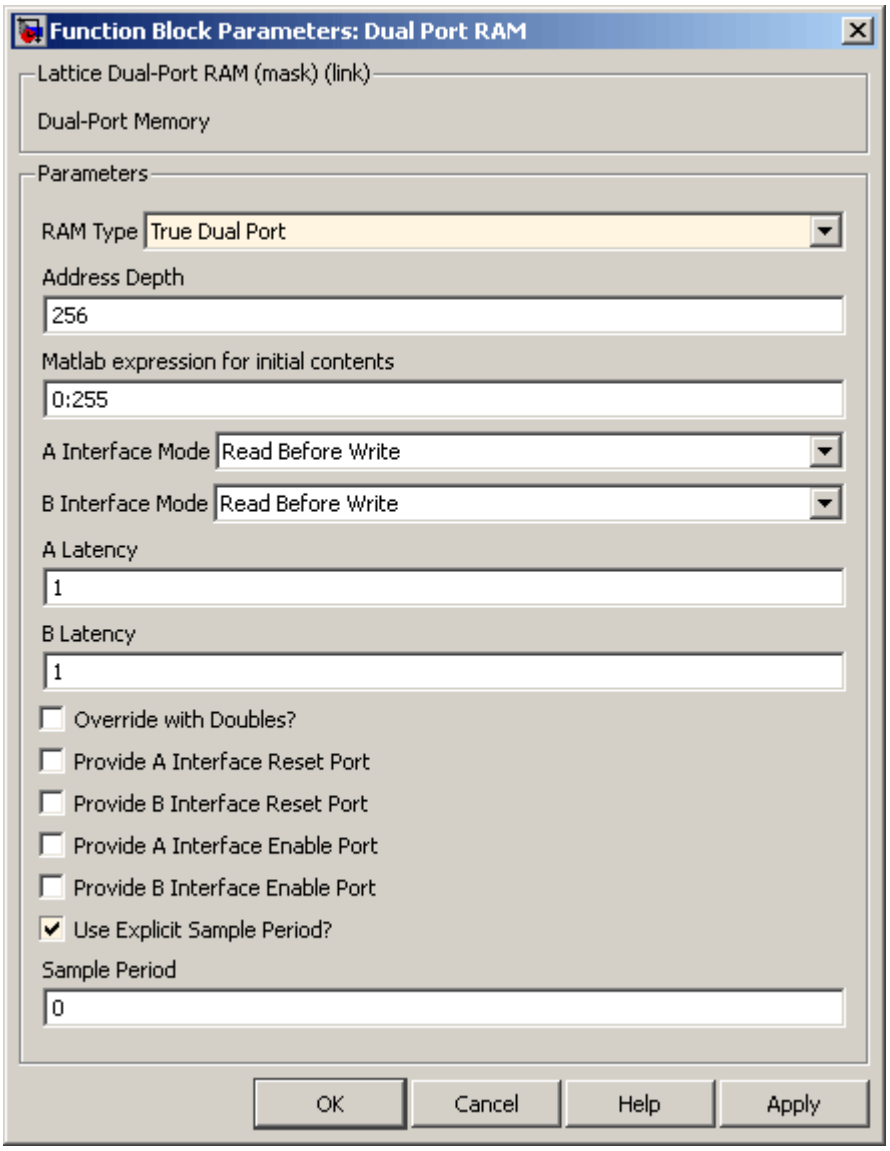
- **Provide Write Enable Port?**:

  Select or deselect. Deselect always set to constant 1.

Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

True Dual Port RAM

The user configurable parameters for the Pseudo Dual Port RAM block are available in the following dialog box.



Parameters specific to this block are:

- **RAM Type**:

  Choose from Pseudo Dual Port RAM or True Dual Port RAM.

- **Address Depth**: Depth of memory.
- **MATLAB Expression for Initial Contents**:

  Initial value for the memory data. This can be any legal MATLAB expression that produces a vector of the appropriate length. If the result of the evaluation is less than the number of words, the remaining words will be set to zero. If it is larger than the number of words, the excess values will be ignored.
- **A Interface Mode**:

  Memory options include Read Before Write, Normal, and Write Through.
- **B Interface Mode**:

  Memory options include Read Before Write, Normal, and Write Through.
- **Provide A Interface Reset Port**:

  Select or deselect. Deselect always set to constant 0.
- **Provide B Interface Reset Port**:

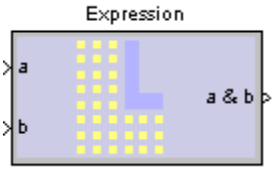
  Select or deselect. Deselect always set to constant 0.
- **Provide A Interface Enable Port**:

  Select or deselect. Deselect always set to constant 1.
- **Provide B Interface Enable Port**:

  Select or deselect. Deselect always set to constant 1.

Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Expression



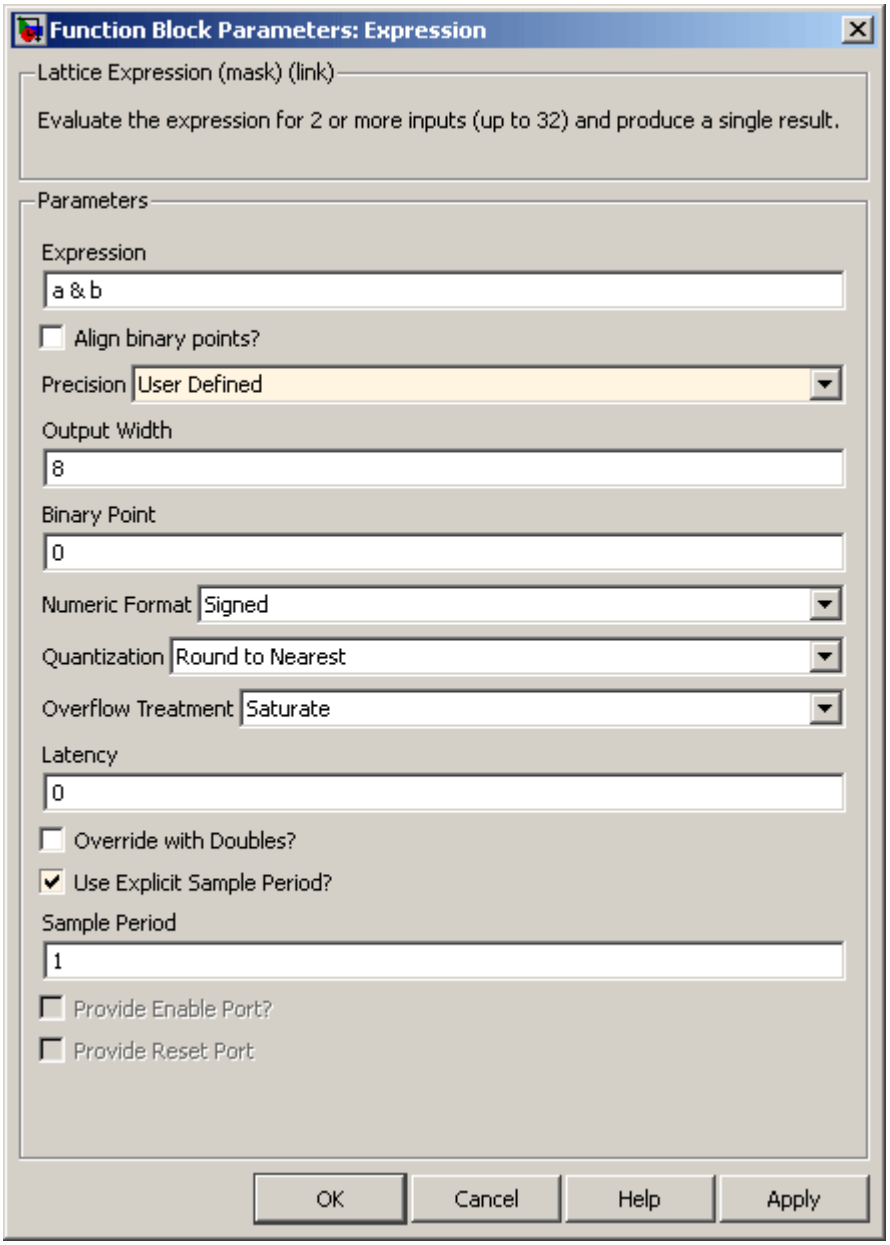
The Expression block provides a user-defined logical function that is performed on from 2 to 32 inputs. All inputs must be of the same width, binary point, and numeric format, otherwise an error will be issued.

The expression is formed from identifiers logical operators, and optional precedence operators. Identifiers must be single upper or lowercase alphanumeric characters. Legal logical operators are “&” (and), “|” (or), “^“(xor), and “~” (not). In addition, the precedence operators “(“and“)” may be used. The operators are evaluated in order of precedence, or from left to right for equal precedence. The precedence order is”~”, “&”, “^”, and “|”. For example, the expression “~a | b & c” is identical to “(~a) | (b & c)”.

The use of the precedence operator is highly recommended to remove ambiguity.

Ports are placed on the model and labeled in the order in which they occur in the expression using the specified identifiers.

The user configurable parameters for the Logical block are available in the following dialog box.



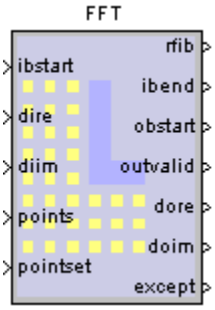
Parameters specific to this block are:

- **Expression**:

  An expression formed of legal identifiers and operators.

Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

FFT



Lattice's FFT Compiler IP core can be employed for implementing high speed N point Fast Fourier Transform, where $N = 2^k$ and k is from 6 to 14. This IP core can be configured to perform forward FFT, inverse FFT (IFFT) or port selectable forward/inverse FFT. The FFT compiler offers two choices of implementation: high performance (Streaming I/O) and low resource (Burst I/O). In the high performance implementation, the FFT 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 lesser slices (logic unit of Lattice FPGA devices) and EBR (Embedded Block RAM) resources or if the device is too small to accommodate high performance version.

To account for the data growth in fine register length implementations, the FFT compiler allows one of three fixed scaling options after each radix-2 stage of the FFT computation. The low resource version also supports dynamic stage-wise scale factors and block floating point arithmetic. The number of FFT points can also be varied dynamically through a port in the low resource implementation.

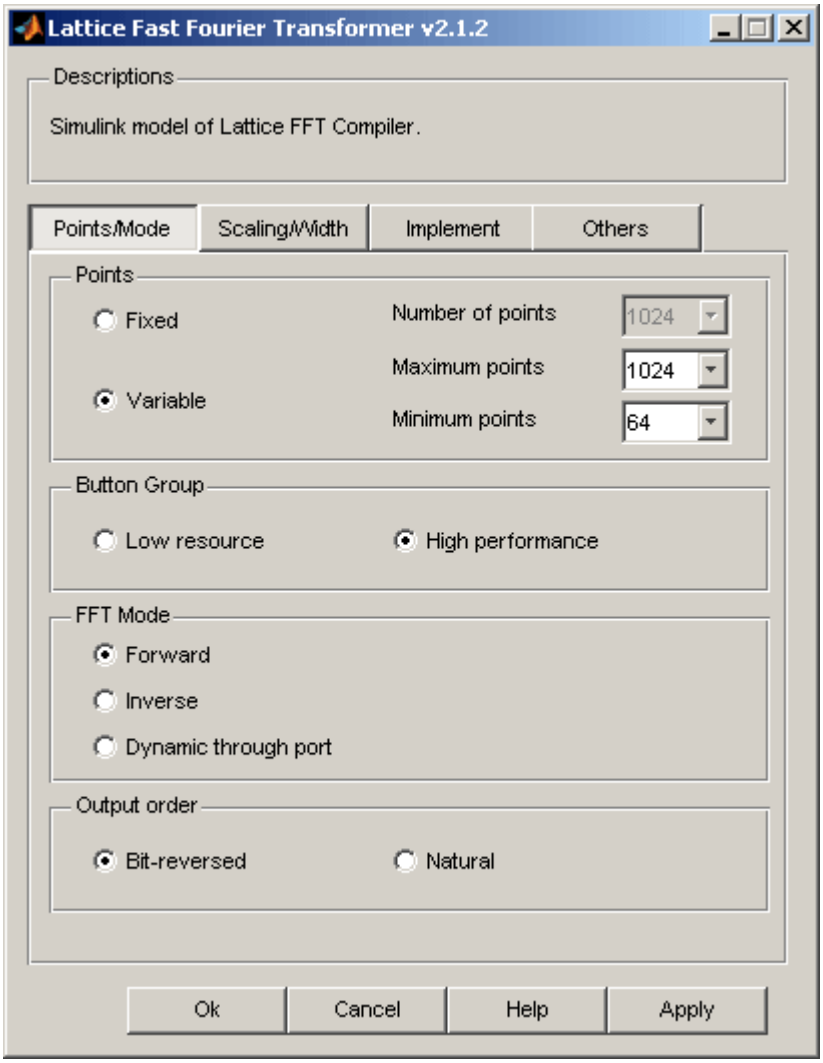
The Lattice FFT Compiler IP Core supports LatticeECP/EC and LatticeECP2/M devices. For further information, you can refer to the User's Manual on Lattice web site.

Note

This IP core is available for evaluation in the MATLAB/Simulink environment. A netlist will be generated for simulation purposes. An IP license must be purchased to implement the IP into a Lattice FPGA. Information on all Lattice IP Modules, including data sheets, brochures, and downloads, can be found on the Lattice Semiconductor Corporation web site at the following URL:

http://www.latticesemi.com/products/intellectualproperty/index.cfm

The user configurable parameters for FFT Compiler are available in the following dialog box.



Points/Mode Tab

- **Number of Points**:

  Specifies the number of FFT points if Points Variability is “fixed.”

- **Maximum Points**:

  Denotes the maximum for the points range if Points Variability is “variable.”

- **Minimum points**:

  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**:

  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**:

  Specifies whether the output data is in bit-reversed or natural order.

Scaling/Width Tab

- **Scaling Mode**:

  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 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 2 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.

- **Input Data Width**:

  Input data width (width of either of the components: real or imaginary).

- **Twiddle Factor Width**:

  Twiddle factor width (width of either of the components: real or imaginary).

- **Precision Reduction Method**:

  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 2 if FFT Mode is dynamic) to truncation. Setting the last stage to truncation results in better throughput.

Implement Tab

- **Multiplier Type**:

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

- **Adder Pipeline**:

  This option is used to specify an additional pipeline after the adders.

- **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 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.

- **Connect reset port to GSR**:

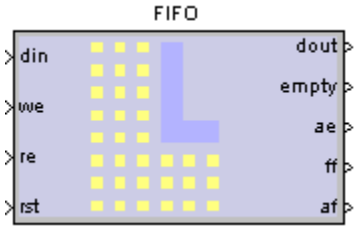
  If this option is checked, the GSR is instantiated and used to route the FFT compiler's rstn input. Using GSR improves the utilization and performance of the FFT compiler. However, if GSR is used, an active input in rstn will reset most of the FPGA components as well. This option must be checked to enable the Hardware evaluation capability for this IP.

Others Tab

- **Manually connect asynchronous reset port**:

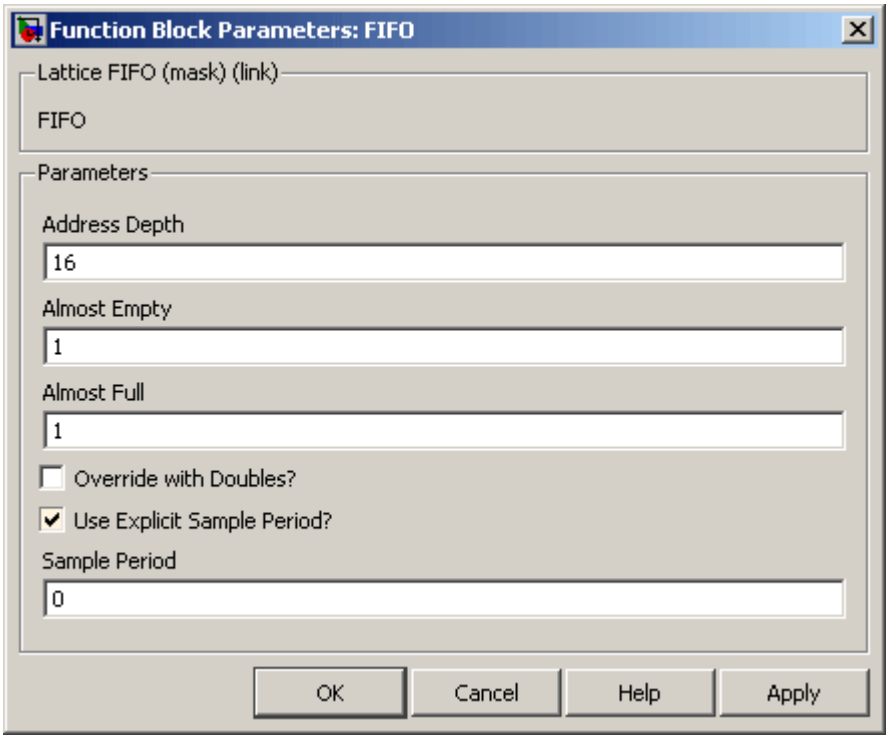
  Select this option to connect asynchronous reset port manually. Otherwise, the port will not be present.

FIFO



The FIFO block implements a First-In, First-Out memory queue. See the technical notes of individual device family on memory usage for detailed specification.

The user configurable parameters for the FIFO block are available in the following dialog box.



Block parameters are:

- **Address Depth**:

  Address depth is a positive integer, greater than 0, which specifies the number of memory words.

- **Almost Empty**:

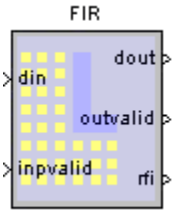
  A positive integer that specifies the number of words away from empty that the almost empty flag output will go high. Valid values are from 0 to the number of words –1.

- **Almost Full**:

  A positive integer that specifies the number of words away from full that the almost full flag output will go high. Valid values are from 0 to the number of words –1.

Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

FIR



The Lattice FIR (Finite Impulse Response) Filter IP core is a widely configurable, multi-channel FIR filter, implemented using high performance sysDSP™ blocks available in Lattice devices. In addition to single rate filters, the IP core also supports a range of polyphase decimation and interpolation filters. The utilization versus throughput trade-off can be controlled by specifying the number of multipliers used for implementing the filter. For example, using one multiplier results in the best resource utilization, whereas using as many multipliers as the number of taps results in the best throughput. The FIR Filter IP core supports as high as 256 channels, with each having up to 2048 taps.

The input data, coefficient and output data widths are configurable over a wide range. The IP core uses full internal precision while allowing variable output precision with several choices for saturation and rounding. The coefficients of the filter can be specified at generation time and/or re-loadable during run-time through input ports.

The FIR IP core has the following features:

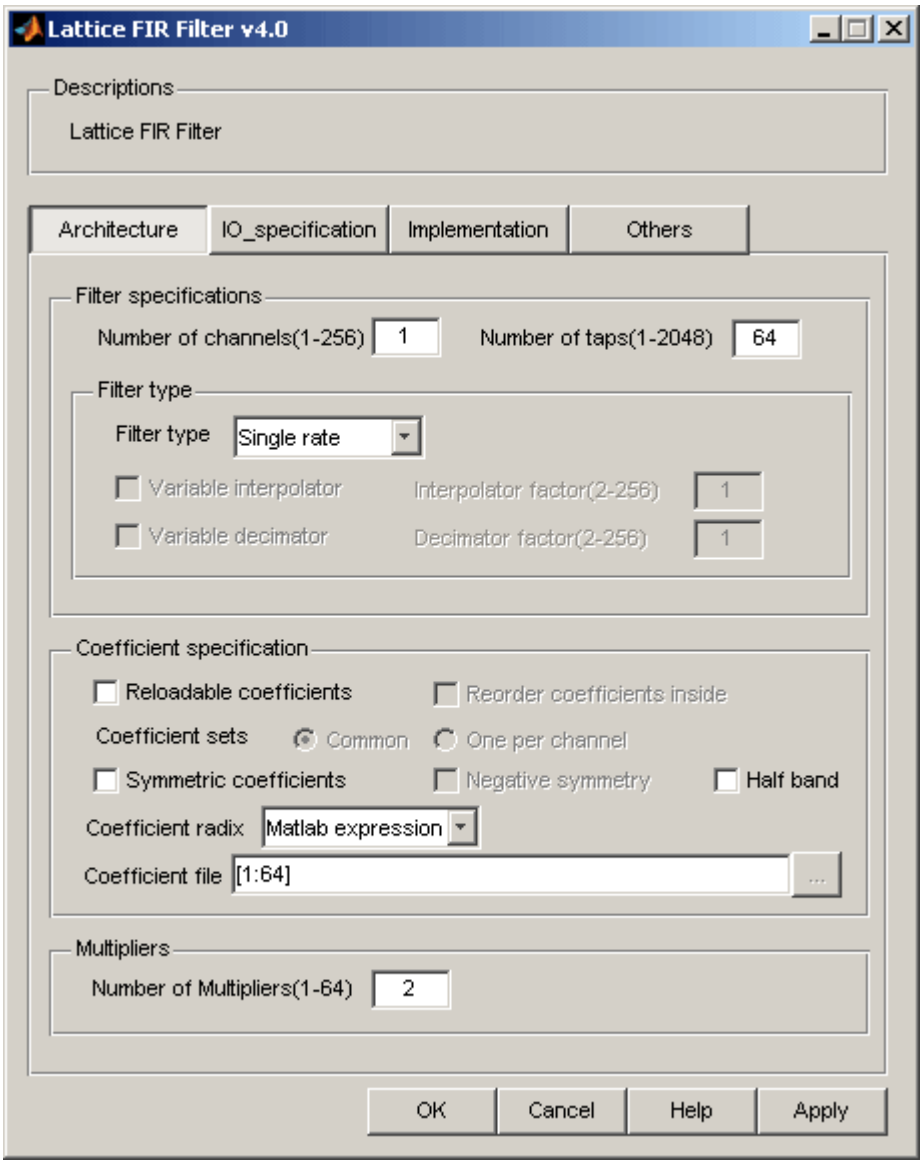
- Variable number of taps up to 2048
- Input and coefficients widths of 4 to 32 bits
- Multi-channel support for up to 256 channels
- Decimation and Interpolation ratios from 2 to 256
- Support for half-band filter
- Configurable parallelism from full to serial
- Signed or unsigned data and coefficients
- Coefficients symmetry and negative symmetry optimization
- Re-loadable coefficients support
- Full precision arithmetic
- Selectable output width and precision
- Selectable overflow: wrap-around or saturation
- Selectable rounding: truncation, round towards zero, round away from zero, round to nearest, and convergent rounding
- Width and precision specified using fixed point notations
- Handshake signals to facilitate smooth interfacing

Note

This IP core is available for evaluation in the MATLAB/Simulink environment. A netlist will be generated for simulation purposes. An IP license must be purchased to implement the IP into a Lattice FPGA. Information on all Lattice IP Modules, including data sheets, brochures, and downloads, can be found on the Lattice Semiconductor Corporation web site at the following URL:

http://www.latticesemi.com/products/intellectualproperty/index.cfm

The user configurable parameters for FIR Filter are available in the following dialog box.



Architecture Tab

Filter specifications

- **Number of channels**: Number of channels.
- **Number of taps**: Number of taps.
- **Filter type**:

  Specifies whether the filter is single rate, interpolator or decimator.
- **Variable interpolator**:

Specifies whether the interpolation factor is fixed at the time of IP generation or variable during run-time. If this is checked, the interpolation factor is set through the input port `ifactor` when `factorset` is high.

- **Interpolator factor**: Value of the fixed interpolation factor.
- **Variable decimator**:

  Specifies whether the decimation factor is fixed at the time of IP generation or variable during run-time. If this is checked, the decimation factor is set through the input port `dfactor` when `factorset` is high.

- **Decimator factor**: Value of the fixed decimation factor.

Coefficient specification

- **Reloadable coefficients**:

  Specifies whether the coefficients are fixed or reloadable. If checked, the coefficients can be reloaded during core operation using the input port `coeffin`.

- **Reorder coefficients inside**:

  When coefficients are reloadable, they need to be entered in a particular order. The reordering can be done using the program supplied along with the IP core. However, the core also provides for optional hardware reordering at the expense of additional hardware resources. If this option is selected, the coefficients can be entered in the normal sequence to the core and the core will internally reorder hem as required. This option is not available when **Filter type** is interpolator and **Symmetric coefficients** is enabled.

- **Coefficient sets**:

  Specifies whether the same coefficient set is used for all channels or an independent coefficient set is used for each channel.

- **Symmetric coefficients**:

  Specifies whether the coefficients are symmetric. If this is checked only one half of the number of coefficients (if number of taps is odd, the half value is rounded to the next higher integer) is read from the initialization file.

- **Negative symmetry**:

  If this is checked, the coefficients are considered to be negative symmetric. That is the second half of the coefficients are made equal to the negative of the corresponding first-half coefficients.

- **Half band**:

  Specifies whether a half band filter is realized. If this is checked only one half of the number of coefficients (if the number of taps is odd, the half value is rounded to the next higher integer) is read from the initialization file.

- **Coefficient radix**:

  Radix for the coefficients in the coefficients file. For decimal radix, the negative values have a preceding unary minus sign. For hexadecimal (Hex) and binary radices, the negative values must be written in two’s

complement form using exactly as many digits as specified by the coefficients width parameter.

- **Coefficient file**:

  Specifies the name and location of the coefficients file or Matlab expression.

Multipliers

- **Number of Multipliers**:

  Number of multipliers used to implement the FIR filter. This parameter determines the throughput and resource utilization of the FIR filter.

IO_specification Tab

Data

- **Input data type**:

  Specifies the input data type as signed or unsigned. If the type is signed, the data is interpreted as a two's complement number.

- **Input data width**: Input data width. Value range: 4 to 32. Default: 16.

- **Input data binary points position**:

  Specifies the location of the binary point in the input data. This number specifies the bit position of the binary point from the LSB of the input data. If the number is zero, the point is right after LSB, if positive, it is to the left of LSB and if negative, it is to the right of LSB. Value range: -2 to <*Input data width*> + 2. Default: 0.

Coefficient

- **Coefficient type**:

  Specifies the coefficients type as signed or unsigned. If the type is signed, the coefficient data is interpreted as a two's complement number.

- **Coefficient width**: Coefficient width. Value range: 4 to 32. Default: 16.

- **Coefficient binary points position**:

  Specifies the location of the binary point in the coefficients. This number specifies the bit position of the binary point from the LSB of the coefficients. If the number is zero, the point is right after LSB, if positive, it is to the left of LSB and if negative, it is to the right of LSB. Value range: -2 to <*Coefficient width*> + 2. Default: 0.

Output

- **Output width**:

  Output width. The maximum full precision output width is defined by <*Max Output Width*> = <*Input data width*> + <*Coefficients width*> +ceil(Log2(<*Number of taps*>)). The core's output is usually a part of the full precision output equal to the **Output width** and extracted based on the different binary point position parameters.

  The format for the internal full precision output is displayed as a static text next to the Output width control in the GUI. The format is displayed as

W.*F*, where *W* is the full precision output width and *F* is the location of the binary point from the LSB of the full precision output, counted to the left.

Value range: 4 to <*Max Output Width*>. Default: <*Input data width*>.

- **Output binary point position**:

  This number specifies the bit position of the binary point from the LSB of the actual core output. If the number is zero, the point is right after LSB, if positive, it is to the left of LSB and if negative, it is to the right of LSB. This number, together with the parameter **Output width** determines how the actual core output is extracted from the true full precision output. The precision control parameters **Overflow** and **Rounding** are applied respectively when MSBs and LSBs are discarded from the true full precision output.

  Value range: (4 + <*Input data binary point position*> + <*coefficient binary point position*> – <*Max output width*>) to (<*Output width*> + <*Input data binary point position*> + <*Coefficient binary point position*> – 4). Default: 0.

Precision control

- **Overflow**:

  Specifies what kind of overflow control is to be used. This parameter is available whenever there is a need to drop some of the MSBs from the true output. If the selection is **Saturation**, the output value is clipped to the maximum, if positive or minimum, if negative, while discarding the MSBs. If the selection is **Wrap-around**, the MSBs are simply discarded without making any correction.

- **Rounding**:

  Specifies the rounding method when there is a need to drop one or more LSBs from the true output.

Implementation Tab

Memory type

- **Data memory type**:

  Selects the type of memory that is used for storing the data. If the selection is **EBR**, Lattice Embedded Block RAM memories are used for storing the data. If the selection is **Distributed**, look-up-table based distributed memories are used for storing data. If **Auto** is selected, EBR memories are used for memory sizes deeper than 128 locations and distributed memories are used for all other memories.

- **Coefficient memory type**:

  Selects the type of memory that is used for storing the coefficients. If the selection is **EBR**, EBR memories are used for storing the coefficients. If the selection is **Distributed**, distributed, distributed memories are used for storing coefficients. If **Auto** is selected, EBR memories are used for memory sizes deeper than 128 locations and distributed memories are used for all other memories.

- **Input buffer type**: Memory type for the input buffer.
- **Output buffer type**: Memory type for the output buffer.

Optional ports

- **Synchronous reset (sr)**:

  Specifies if a synchronous reset port is needed in the IP. Synchronous reset signal resets all the registers in the FIR filter IP core.

- **Clock enable (ce)**:

  Specifies if a clock enable port is needed in the IP. Clock enable control can be used for power saving when the core is not used. Use of clock enable port increases the resource utilization and may affect the performance due to the increased routing congestion.

Optimization

- **Area** & **Speed**:

  Specifies the optimization method. If **Area** is selected, the core is optimized for lower resource utilization and if **Speed** is selected, the core is optimized for higher performance, but with slightly higher resource utilization.

GSR

- **Connect reset port to GSR**:

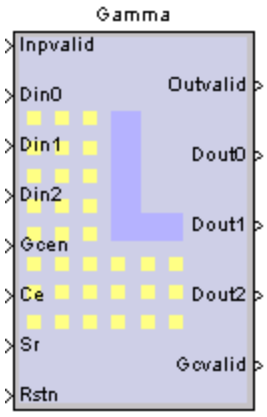
  If this option is checked, the GSR is instantiated and used to route the IP core's `rstn` input. Using GSR improves the utilization and performance of the IP core. However, if GSR is used, an active input at `rstn` will reset most of the FPGA components as well. This option must be checked to enable the hardware evaluation capability for this IP core.

Others Tab

- **Manually connect to asynchronous reset port**:

  Select this option to connect asynchronous reset port manually. Otherwise, the port will not be present.

Gamma



Gamma correction is a kind of pre-distortion correction made to images or video frames to offset the non-linear behavior of display systems, such as cathode ray tube (CRT) displays. A characteristic of CRT displays is that the intensity they generate is not a linear function of the input voltage. Instead the intensity is proportional to a power of the signal amplitude, also referred to as gamma. Gamma is usually greater than 1 and hence the displays have a lower gain at low intensities and progressively larger gain at higher intensities. Lattice's Gamma Corrector IP core multiplies the input signal with the inverse of the display transfer function which results in a linear intensity response with respect to the original input signal.

The Lattice Gamma Corrector IP core is widely parameterizable and supports both CRT and non-CRT systems. Furthermore, both the Gamma Corrector IP core and Lattice's CSC can be implemented to satisfy many of the requirements for video pre-processing in most display systems.

The Lattice Gamma Corrector IP core supports LatticeECP2/M devices. You can refer to the Lattice Gamma Corrector IP Core User Manual on Lattice web site for more detailed information.

The Lattice Color Space Converter IP Core has the following features:

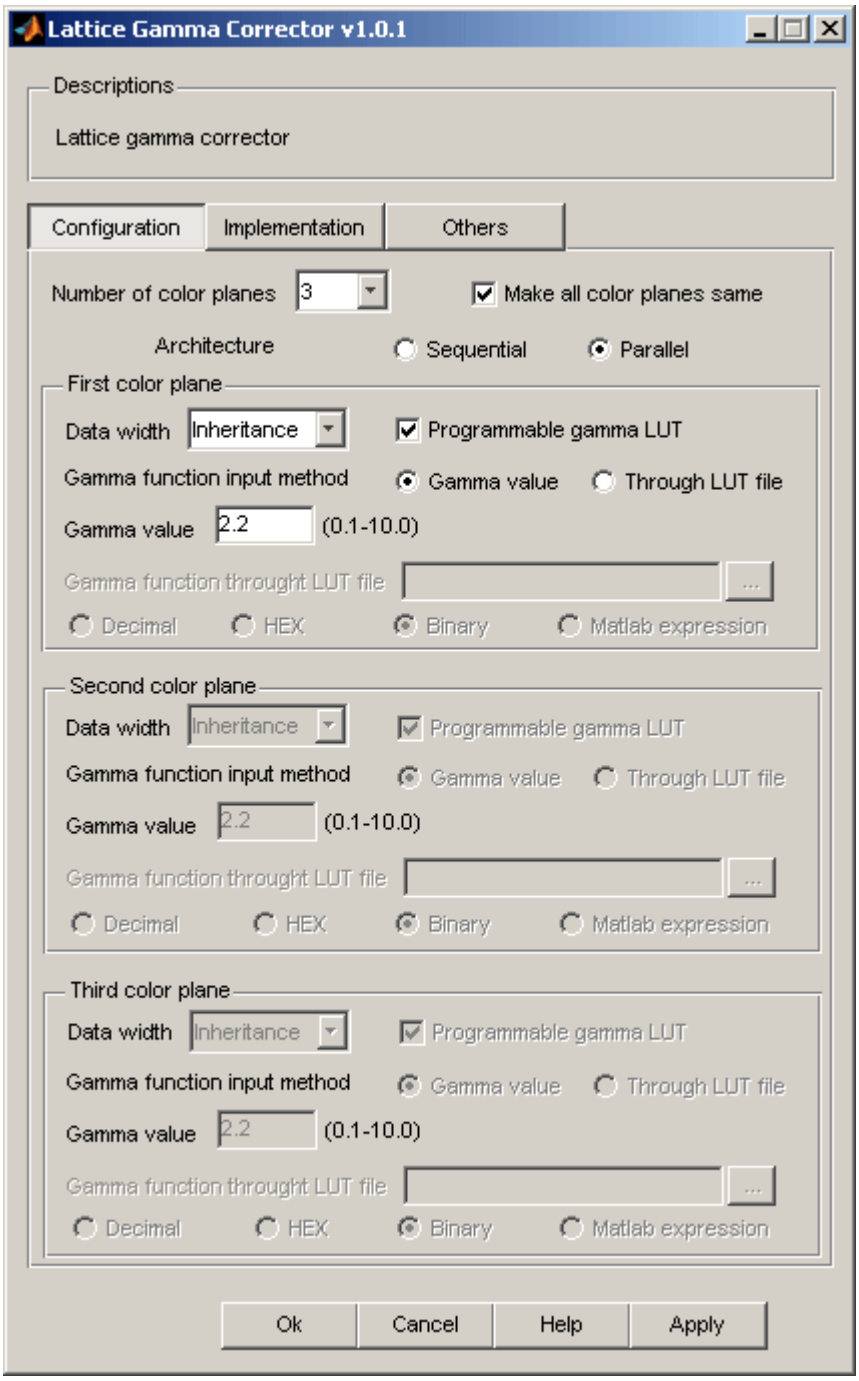
- Configurable number of color planes: 1 to 3
- Configurable number of bits per color plane: 4 to 12
- Option to specify gamma correction characteristics as an equation using a gamma value or by the actual mapping values of the look-up table
- Gamma correction look-up table can be run-time programmable
- Optimized gamma look-up table memory when same gamma correction is used for multiple color planes
- Gamma correction enable/disable control
- Option for sequential or parallel architecture for area or throughput trade-off
- Registered input option for input set-up time improvement

Note

This IP core is available for evaluation in the MATLAB/Simulink environment. A netlist will be generated for simulation purposes. An IP license must be purchased to implement the IP into a Lattice FPGA. Information on all Lattice IP Modules, including data sheets, brochures, and downloads, can be found on the Lattice Semiconductor Corporation web site at the following URL:

http://www.latticesemi.com/products/intellectualproperty/index.cfm

The user configurable parameters for Gamma Corrector are available in the following dialog box.



Configuration Tab

- **Number of Color Planes**: 1, 2, 3
- **Make all color planes same**:

Select this option will make all the other color plane parameters the same as those of the first color plane.

- **Architecture**: Selection between parallel and sequential implementation.
- **Gamma function input method**:

  The method for specifying the gamma function. The function can be specified by the value of the gamma in the gamma correction equation or by the actual gamma mapping values for all the pixel values in the input range.

- **Gamma Value (0.1-10.0)**:

  This gamma value is used to create the gamma LUT using Equation (2). This parameter is available when the Gamma function input method is selected as Gamma value.

- **Gamma function through file**:

  This browse button is enabled when the Gamma function input methods set to Through LUT file. The gamma LUT values will be read from the text file specified. You can further specify the format of the file, whether it is decimal, hexadecimal, binary or MATLAB expression.

- **Programmable Gamma LUT**:

  This parameter is used to indicate whether the gamma LUT is also programmable through the input port.

Implementation Tab

- **Add bypass function**:

  Selecting this option will add the dynamic gamma correction bypass functionality. Input port gcen and output port gcvalid are added to the Gamma Corrector IP.

- **Registered Input**:

  If this option is selected, the inputs are registered. The setup time of core inputs will be improved by registering the inputs. This option is useful when the input data is provided on the device pins.

- **Connect reset port to GSR**:

  If this option is checked, the GSR is instantiated and used to route the CSC's rstn input. Using GSR improves the utilization and performance of the CSC IP. However, if GSR is used, an active input in rstn will reset most of the FPGA components as well. This option must be checked to enable the hardware evaluation capability for this IP.

- **Memory type**:

  Selection between EBR, Distributed and Automatic type. This parameter influences the type of memory used to implement the gamma LUT. If the EBR option is selected then the device's EBR (Embedded Block RAM) resources are used for the gamma LUT, if the data width is greater than 4. If the Distributed option is selected, distributed memory (realized using FPGA's LUTs) is used for the gamma LUT if the data width is less than 9. If the Automatic option is selected, The memory resources are automatically selected in an optimal way, typically using distributed

memories for data widths less than 7 and block memories for higher data widths. There are exceptions to the this split of EBR/distributed memory usage for complex configurations that require optimization of the gamma LUT for multiple color planes.

- **Optional Input Ports**:
  - **ce:** Optional clock enable input port ce is added to the IP core if this option is checked.
  - **sr:** Optional synchronous reset input port sr is added to the IP core if this option is checked.
- **Output Latency**:

  Range 4 - 6. This static display shows the output latency for the selected core configuration.

Others Tab

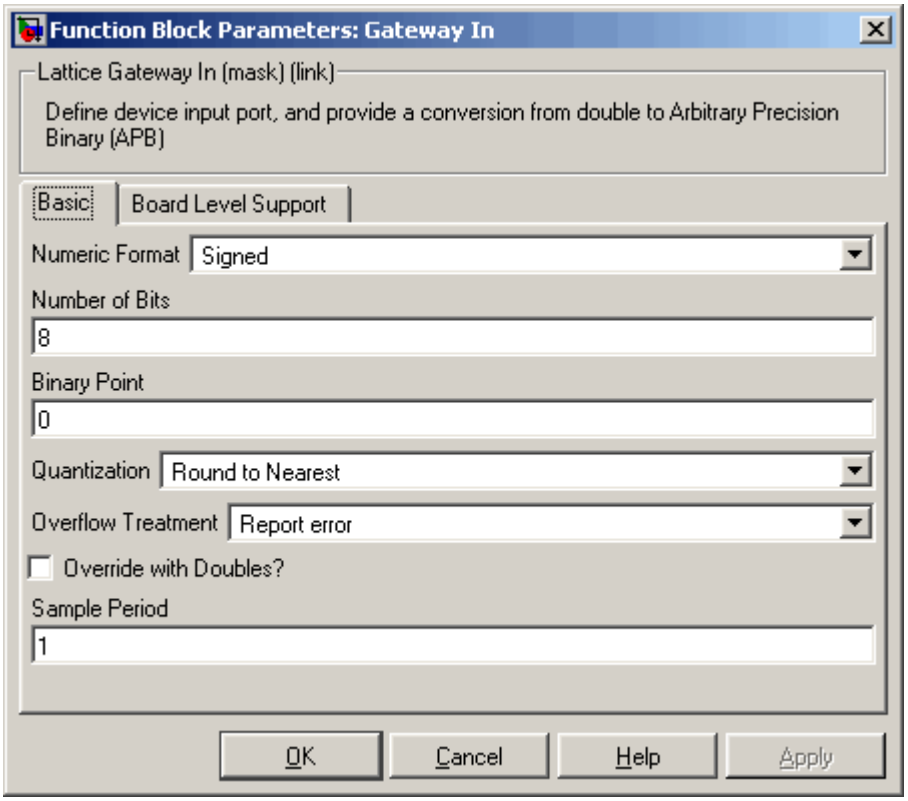
- **Manually connect reset port**:

  Select this option to connect asynchronous reset port manually. Otherwise, the port will not be present.

Gateway In

The Gateway In block defines a device input port, and provides a conversion from double to Lattice Fixed Point (LFP).

The user configurable parameters for the Gateway In block has two tabs and are available in the following dialog box.



Basic Tab

- **Number of Bits**: The default value is 8.

Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Board Level Support Tab

- **Disregard for Hardware Co-simulation**:

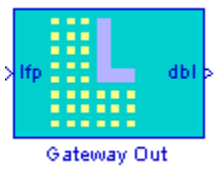
  If this option is selected, the Gateway In will not show up in Black Box mask.

- **Input FPGA Pin Locations**:

  Once this option is selected, the next option is visible and the user can input the Pin Location for this input port.

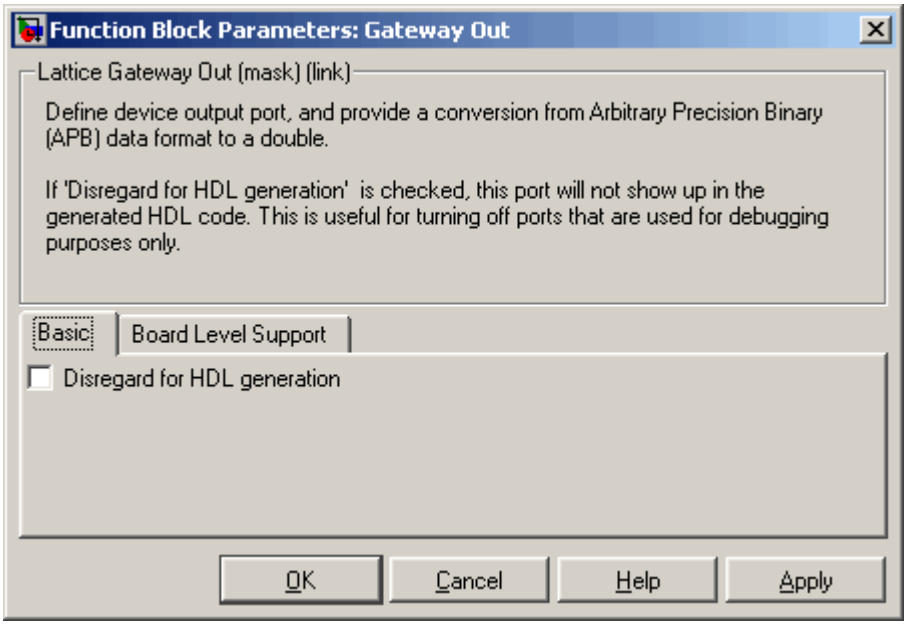
- **Pin Location Input**: Input pin location for this input port.

Gateway Out



The Gateway Out block defines a device output port, and provides a conversion from Lattice Fixed Point (LFP) data format back to a double.

The user configurable parameters for the Gateway Out block has two tabs and are available in the following dialog box.



Basic Tab

- **Disregard for HDL generation**:

  If this option is selected, this port will not show up in the generated HDL code. This is useful for turning off ports that are used for debugging purposes only.

Board Level Support Tab

- **Disregard for Hardware Co-simulation**:

  If this option is selected, the Gateway In will not show up in Black Box mask.

- **Input FPGA Pin Locations**:

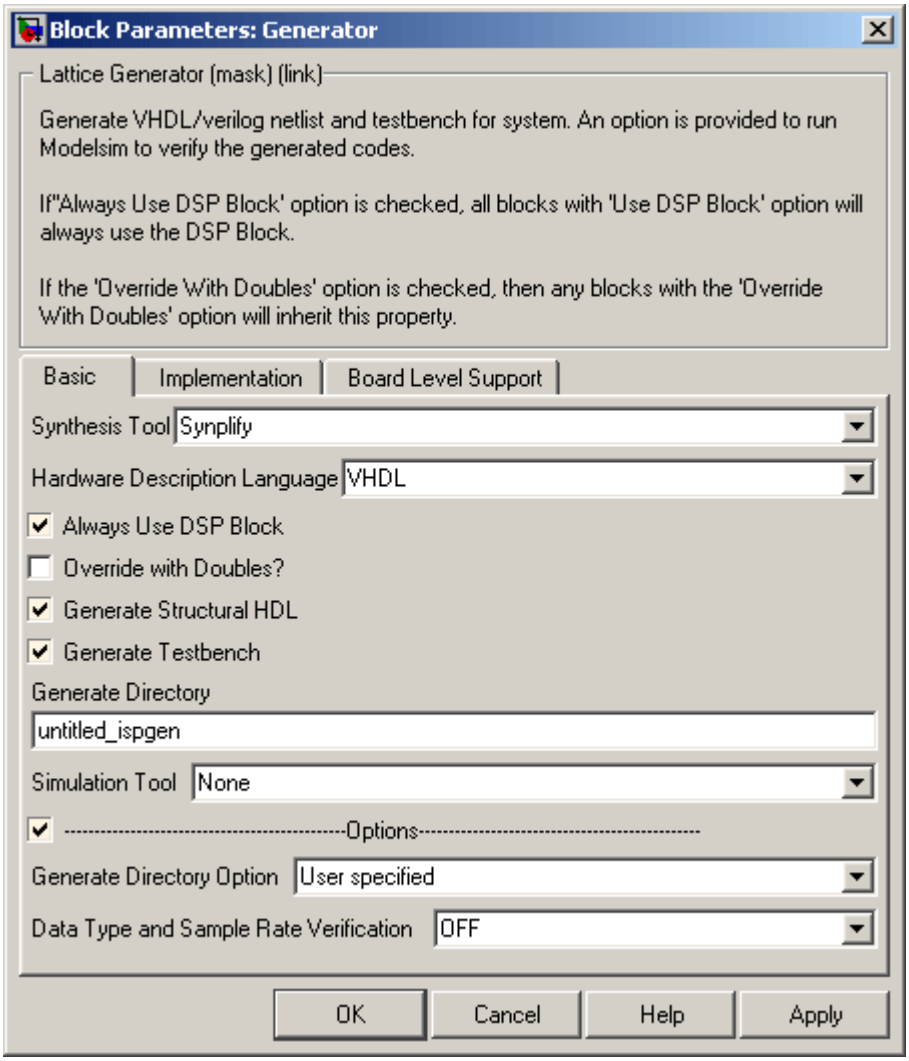
  Once this option is selected, the next option is visible and the user can input the Pin Location for this input port.

- **Pin Location Input**: Input pin location for this input port.

Generator

The Generator block must be included in a Simulink model to enable the generation of structural HDL and testbench. The Generator block supports both VHDL and Verilog HDL. An option is provided to run Modelsim to verify the generated VHDL/Verilog code. A Simulink simulation run must be started to generate structural VHDL/Verilog code via the Generator block. There are also additional options to specify the target device as well as user specified timing preferences.

The user configurable parameters for the Generator block has three tabs and are available in the following dialog box.



Basic Tab

- **Synthesis Tool = [Synplify, Precision]**:

  Select the synthesis tool. (Only Synplify or Precision are supported).

- **Hardware Description Language = [VHDL, Verilog]**

- **Generate Testbench**:

  Generate a VHDL/Verilog testbench that can be used to verify the Simulink model. If the “Generate Testbench” option is selected, then an additional “Simulation Tool” dropdown box will appear in the Generator dialog box.

- **Generate Directory**:

  Specify the output directory for the VHDL/Verilog files generated.

- **Simulation Tool = [None, Active-HDL GUI, Active-HDL Command Line, Modelsim GUI, Modelsim Command Line]**:

  Verify the generated VHDL/Verilog code with Active-HDL or ModelSim.

- **None**:

  Do not run Modelsim or Active-HDL. MATLAB will still simulate.

- **Active-HDL GUI**:

  Run Active-HDL in GUI Mode

- **Active-HDL Command Line**:

  Run Active-HDL in command line mode (no GUI windows will appear)

- **Modelsim GUI**:

  Run Modelsim in GUI Mode

- **Modelsim Command Line**:

  Run Modelsim in command line mode (no GUI windows will appear)

Implementation Tab

- **Select Device**:

  Select the target FPGA device. If the “Select Device” option is selected, then the next five parameters to select a device will appear in the Generator dialog box. The 5 parameters in the Select Device section are:

  - **Family**:

    Select the target FPGA device family.

  - **Device**:

    Select the part name. The appropriate list of part names will be displayed for the selected device family.

  - **Speed Grade**:

    Select the speed grade. The appropriate list of speed grades will be displayed for the selected part name.

  - **Package Type**:

    Select the package type. The appropriate list of available packages will be displayed for the selected part name.

  - **Operating Conditions = [Commercial, Industrial]**:

    Select the operating condition. The appropriate display of available operating conditions will be displayed for the selected speed grade.

- **Select Preferences**:

  Specify timing preferences for the target FPGA device. If the “Select Preferences” option is selected, then the next three parameters to select the timing preferences will appear in the Generator dialog box. The three parameters are:

  - **Frequency (in MHz)**:

    Specify the clock frequency in MHz.

  - **Input Setup Time (in ns)**:

    Specify the input setup time in nanoseconds.

- **Clock to Output Time (in ns)**:

  Specify the clock to output time in nanoseconds.

Board Level Support Tab

- **Select Board**:

  User can select various supported Lattice Evaluation Boards. Currently, only "**ECP2-50E DSP Evaluation Board**" is supported.

- **Generate Bitstream**:

  This option can be checked and triggered by running Simulink simulation. This option runs synthesis tool specified in "Basic/Synthesis Tool," edif2ngd, ngdbuild, map, par and finally generate bit stream that is ready to be downloaded to the evaluation board.

- **Board Initialization and Verification**:

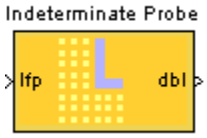
  This option can be checked and triggered by running Simulink simulation. This option runs all the procedures in the previous option **Generate Bitstream**, downloads bit stream to the board, gather the input data from previous Simulink simulation, sends input data to the board, and saves the hardware simulation results into files. After running Simulink simulation again, "Black Box" is able to read and display the hardware simulation results. Note that, the number of hardware simulation clock cycles is limited to the previous Simulink simulation time.

- **Real-time Hardware Co-simulation**:

  This option can be checked and triggered by running Simulink simulation. Finishing the previous option **Board Initialization and Verification** is a prerequisite. After this option is selected, the hardware cosimulation is in a free-running status. During each Simulink simulation time step, "Black Box" receives the current input data from Simulink, sends them to the board, reads back the hardware simulation results and displays them

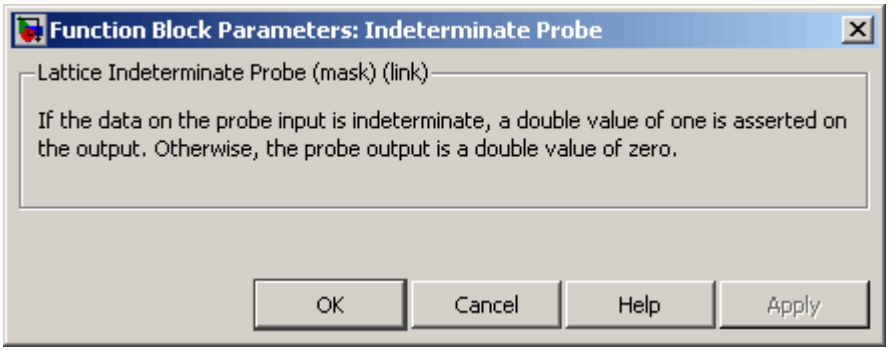
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Indeterminate Probe

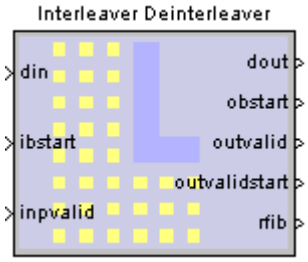


The Indeterminate Probe detects whether its input value is in transition (indeterminate). The output value has a data type of double, and its value indicates whether the input value is stable or indeterminate. If the input is stable, the output value is "0"; if the input value is indeterminate, the output value is "1".

The user configurable parameters for the Indeterminate Probe are available in the following dialog box.



Interleaver Deinterleaver



Interleaving is a technique commonly used in communication systems to overcome correlated channel noise such as burst error or fading. The interleaver rearranges input data such that consecutive data are split among different blocks. At the receiver end, the interleaved data is arranged back into the original sequence by the de-interleaver. As a result of interleaving, correlated noise introduced in the transmission channel appears to be statistically independent at the receiver and thus allows better error correction.

The Lattice Interleaver/De-interleaver IP core supports rectangular block type and convolutional architectures. Rectangular interleaving arranges the input data row-wise in a matrix. The interleaved data is obtained by reading the columns of the matrix. Convolutional interleaving feeds the input data to a number of branches, each of which has a shift register with pre-defined length. The output data is taken from the branch outputs. Lattice's Convolutional Interleaver/De-interleaver IP Cores are compliant with ATSC and DVB standards, while the Rectangular Interleaver/De-interleaver is compliant with IEEE 802.16a standard.

Features:

- High Performance and Area Efficient Symbol Interleaver/De-interleaver
- Supports Multiple Standards, Such as DVB, ATSC and IEEE 802.16
- Convolutional and Rectangular Block Type Architectures Available
- Fully Synchronous Design Using a Single Clock
- Symbol Size from 1 to 256 Bits
- Full Handshake Capability for Input and Output Interfaces
- Rectangular Block Type Features
  - Variable block size
  - Variable number of rows
  - Variable number of columns
  - Row permutations

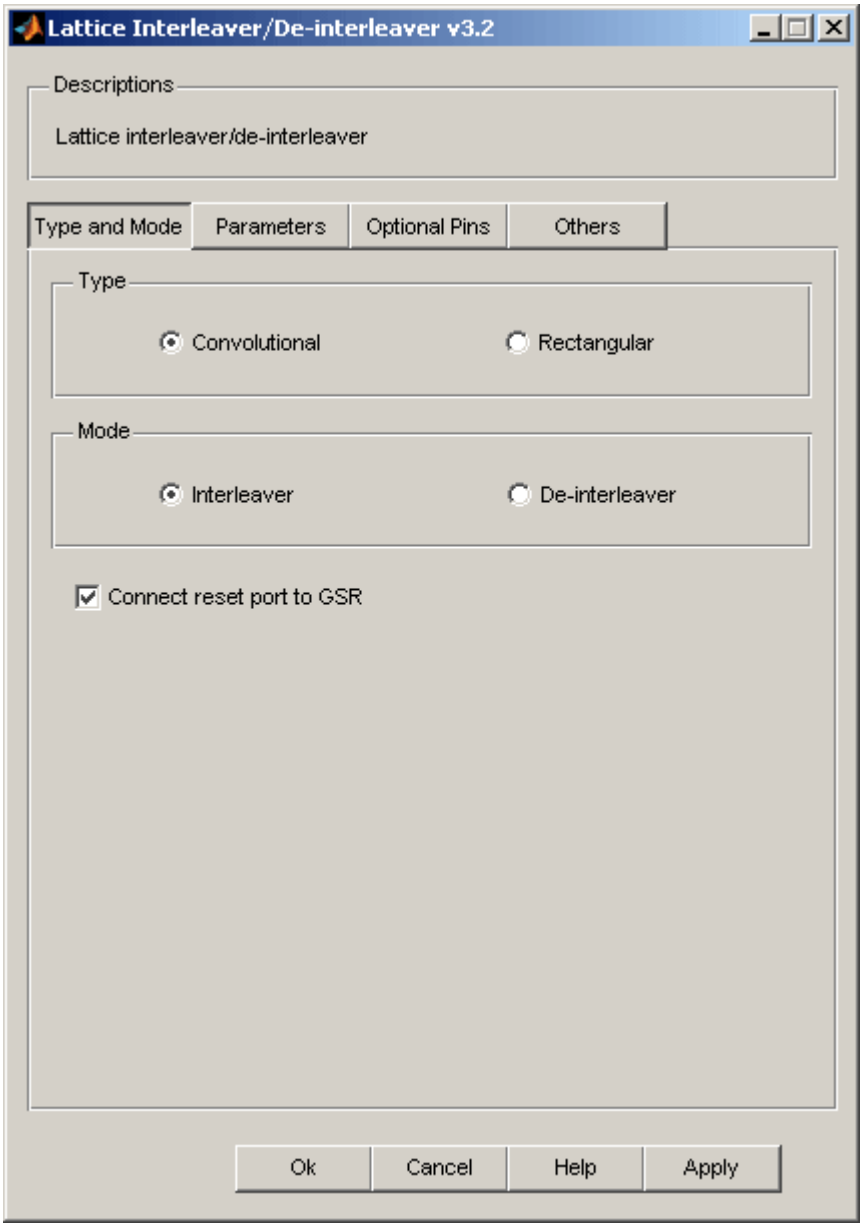
- Column permutations
- Convolutional Type Features
  - User-configurable number of branches
- User-configurable branch length

Note

This IP core is available for evaluation in the MATLAB/Simulink environment. A netlist will be generated for simulation purposes. An IP license must be purchased to implement the IP into a Lattice FPGA. Information on all Lattice IP Modules, including data sheets, brochures, and downloads, can be found on the Lattice Semiconductor Corporation web site at the following URL:

http://www.latticesemi.com/products/intellectualproperty/index.cfm

The user configurable parameters for Interleaver/De-interleaver are available in the following dialog box.



Convolutional Interleaver/De-interleaver Parameters:

- **Mode**: Interleaver (default) or De-interleaver.
- **Symbol Width**:

  Data width of the input symbols. Range is 1-256. Default is 8.
- **Number of Branches**:

  Number of branches of the commutator arm. Range for ECP/EC is 7-256. Range for ispXPGA is 7-181. Default is 12.
- **Branch Length**:

  The difference in storage elements for consecutive branches. Range is 1-780. Default is 17.

Rectangular Interleaver De-interleaver Parameters:

- **Mode**: Interleaver (default) or De-interleaver
- **Symbol Width**:

  Data width of the input symbols. Range is 1-256. Default is 8.
- **Block Size Type**:

  Block size input to the core. There are three options for giving block size value.

  - **Constant**:

    In this configuration the block size value is constant and assigned a value during core configuration.
  - **Row*Col**:

    In this configuration, block size value is computed form the Row and Column values.
  - **Variable**:

    In this configuration, block size value is given through an input port.

  Selectable values are Constant (default), Row*Col, or Variable.
- **Block Size**:

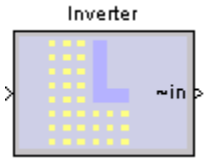
  This parameter is only available if Block Size Type = Constant. Block size input to the core: The block size value should be such that the last symbol in the block should come on the last row. Therefore block size value should be greater than (Col*(Row-1)) and less than or equal to (Col*Row). When block size type is constant, number of rows and number of columns is also constant. Inter-row permutations and inter-column permutations are supported when block size value is Col*Row. Col is number of columns and Row is number of rows. Range for ECP/EC: 9-65536. Range for ispXPGA: 9-16384. Default: 4096.
- **Number of Columns**:

  Number of columns used in the core. Inter-row permutations and inter-column permutations are supported when block size value is Col*Row. Range: 2-256. Default: 256.
- **Number of Rows**:

  Number of rows used in the core. Inter-row permutations and inter-column permutations are supported when block size value is Col*Row. Range: 4-256. Default: 16.
- **Row Permutations**:

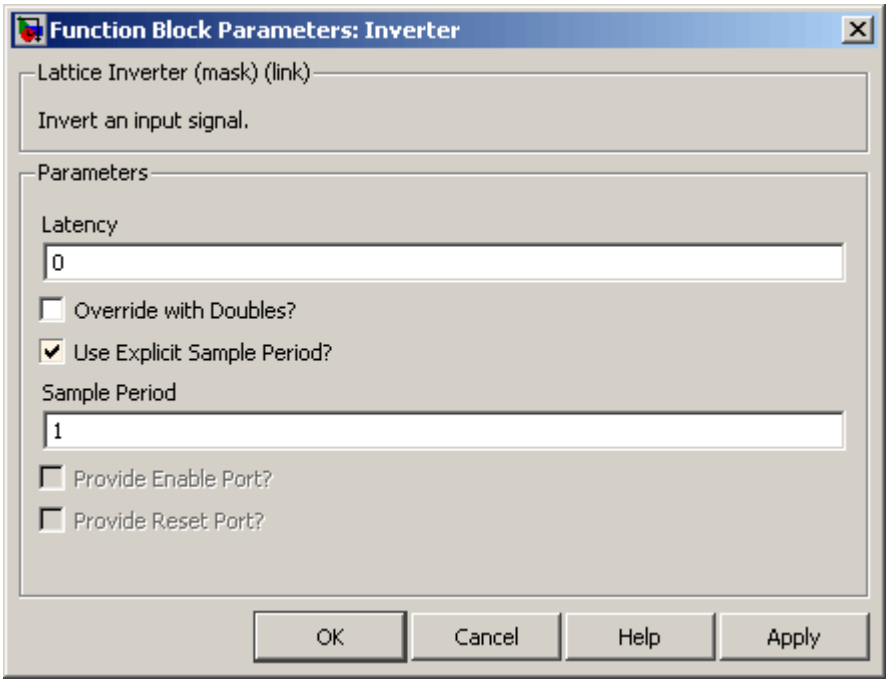
  Rows can be permutated if required. Inter-row permutations and inter-column permutations are supported when block size value is Col*Row. Range: Yes or No. Default: No.

Inverter



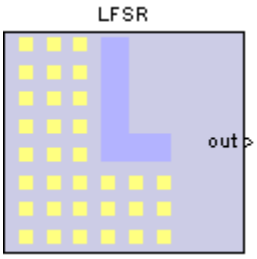
The Inverter block provides an output that is the logical inverse of the input. The Inverter provides a bitwise operation.

The user configurable parameters for the Inverter block are available in the following dialog box.



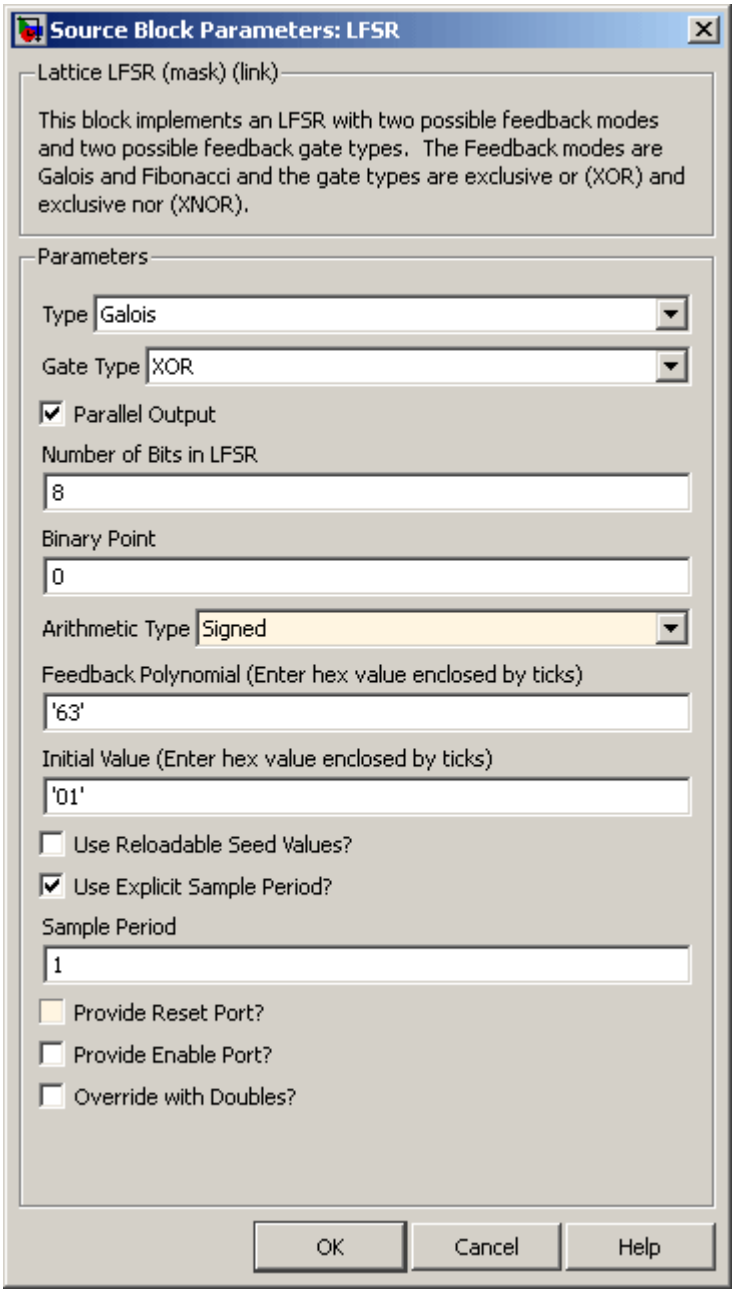
The parameters are explained in Common Parameters for ispLeverDSP Blocks.

LFSR



The Lattice Linear Feedback Shift Register (LFSR) can be implemented with Galois Feedback or Fibonacci Feedback. Feedback gates may be either exclusive or, or exclusive nor. The shift register's initial value may be specified as a constant or it may be loaded from an input port.

The user configurable parameters for the Feedback Shift Register block are available in the following dialog box.



Block parameters are:

- **Type**:

  This parameter selects Galois feedback or Fibonacci feedback. Galois feedback implements an exclusive or/nor at each shift register element corresponding to a non-zero feedback coefficient. Fibonacci feedback calculates the exclusive or/nor of all shift register elements corresponding to non-zero feedback coefficients and inputs the result to the least significant shift register element

- **Gate Type**:

  Choices are XOR (exclusive or) or XNOR (exclusive nor).

- **Parallel Output**:

  If the check box is checked, Parallel Output is selected. Otherwise serial output is selected.

- **Number of bits in LFSR**:

  This is the length of the shift register in bits.

- **Feedback Polynomial**:

  A hexadecimal value enclosed by ticks denotes the feedback polynomial. Coefficients of the feedback polynomial are the one bits in the binary equivalent of the hex value. The most significant bit in the feedback polynomial is implicit because it is always 1, hence the hex parameter value omits it. For example, x8 + x6 + x5 + x1 + 1 might be n as the feedback polynomial for an 8-bit shift register. The hex equivalent is 163, and would be expressed as ‘63’.

- **Initial Value**:

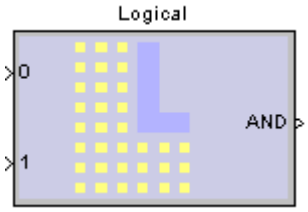
  This is the value taken by the shift register prior to the first shift operation. It must be expressed as a hex value enclosed by ticks and must correspond to value of the Number of bits in LFSR parameter.

- **Use Reloadable Seed**:

  If the option is selected, the din port is defined. The value on din is a parallel value having the same number of bits as the shift register, and presents the value of the reloadable seed to the block. The reloadable seed is loaded at initialization, and whenever a new shift sequence is started.

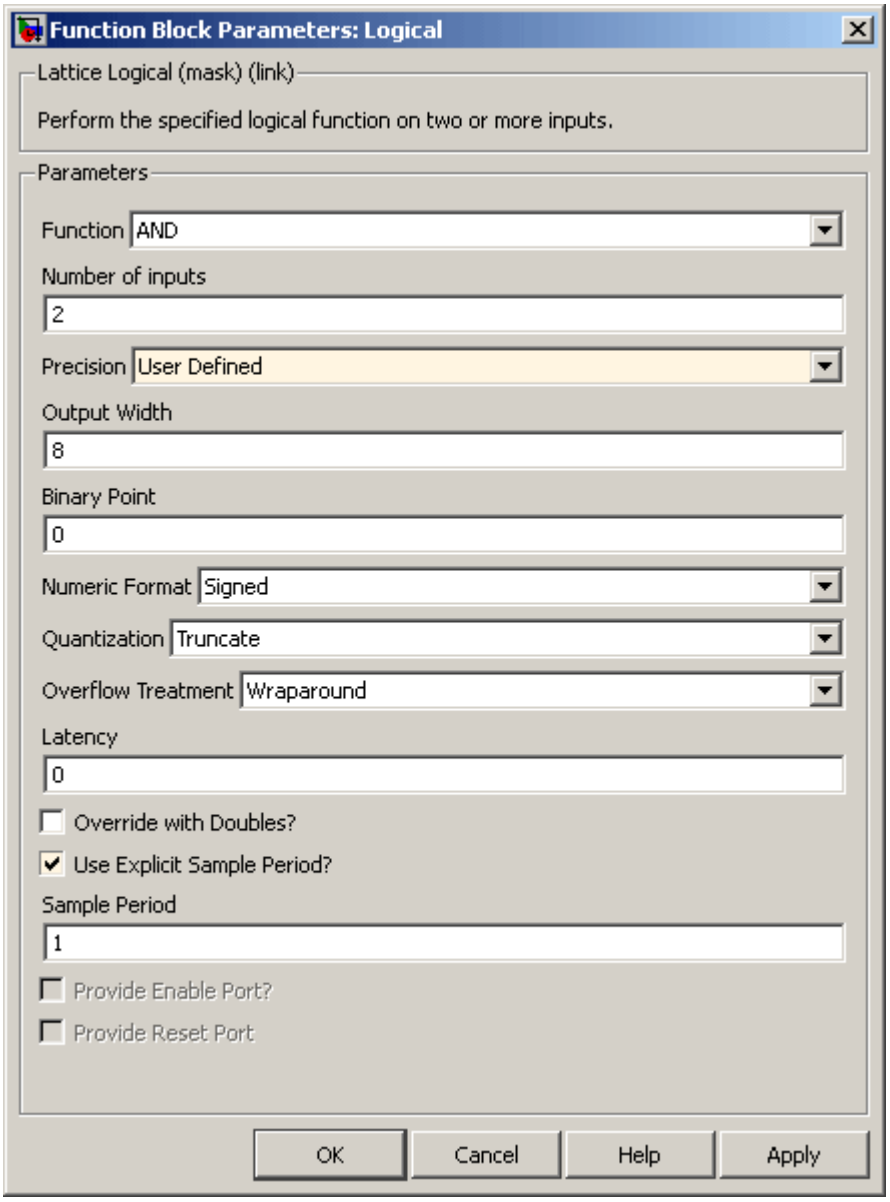
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Logical



The Logical block provides a logical function that is performed on from 2 to 4 inputs. All inputs must be of the same width, binary point, and numeric format, otherwise an error will be issued.

The user configurable parameters for the Logical block are available in the following dialog box.



The block parameters are:

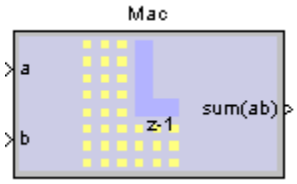
- **Function = [AND, OR, XOR, NAND, NOR, XNOR]**:

  The logical function to be performed.

- **Number of inputs**: An integer value from 2 to 4.

Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Mac



The Multiply/Accumulate block multiplies two inputs and accumulates the product. The accumulator may be initialized from either the multiplier output, an additional input, or with zeroes.

If full precision is selected, the accumulator output will be 34 bits if the multiplier inputs are both 9 bits or less, or 52 bits if the multiplier outputs are both 18 bits or less. If the “Use DSP Block” is checked, the ispDSP block will be used to implement the entire function.

If user defined precision is selected, the accumulator output width can be set to any value up to the maximum supported bit width (150 bits). If the “Use DSP Block” is checked, and the selected width is 34 bits and both multiplier inputs are 9 bits or less, or the selected width is 52 bits and both multiplier inputs are 18 bits or less, the ispDSP block will be used to implement the entire function.

In all cases, if the optional reset pin is selected, or the optional load pin is selected and the load source is other than the multiplier output, the sysDSP block cannot be used to implement the complete function,

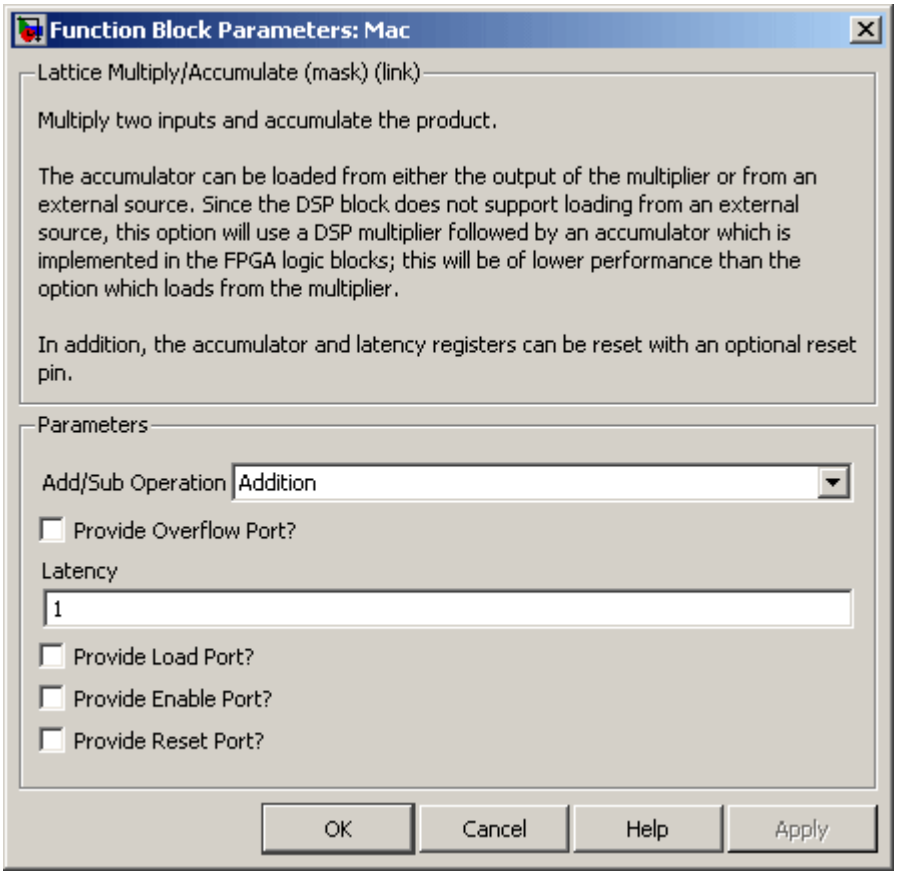
If the ispDSP block is not used to implement the entire function, it will use the ispDSP block to implement the multiply if “Use DSP Block” is checked. If not, the synthesis tool will choose how to implement the multiply. The remaining functionality will be implemented in the FPGA fabric.

If a latency greater than 1 is selected and “Use DSP Block” is checked, the internal pipelines in the ispDSP block will be used as shown in the following table.

Table 10: Multiply/Accumulate Pipeline Usage

Latency	Pipelines Used
1	Output as Accumulator register
2	Input and Output as Accumulator register.
3	All
>3	All + FPGA registers

The user configurable parameters for this block are available in the following dialog box.



The block parameters are:

- **Add/Sub Operation**:

  Allows the implementation of adder or subtractor. Default is set to Addition.

- **Provide Load Port?**:

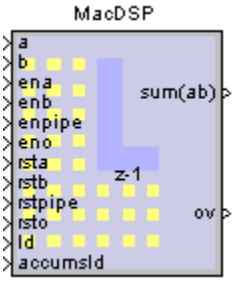
  This enables an optional port for initializing and loading the accumulator. If this is selected, the Load Mode parameter will be enabled for selecting the source for loading.

  - **Load Source = [Input, Alternate Input]**:

    This parameter will be enabled when the optional load port is selected. This parameter selects the source for loading. If Alternate Input is selected, a Din port will be added to the model.

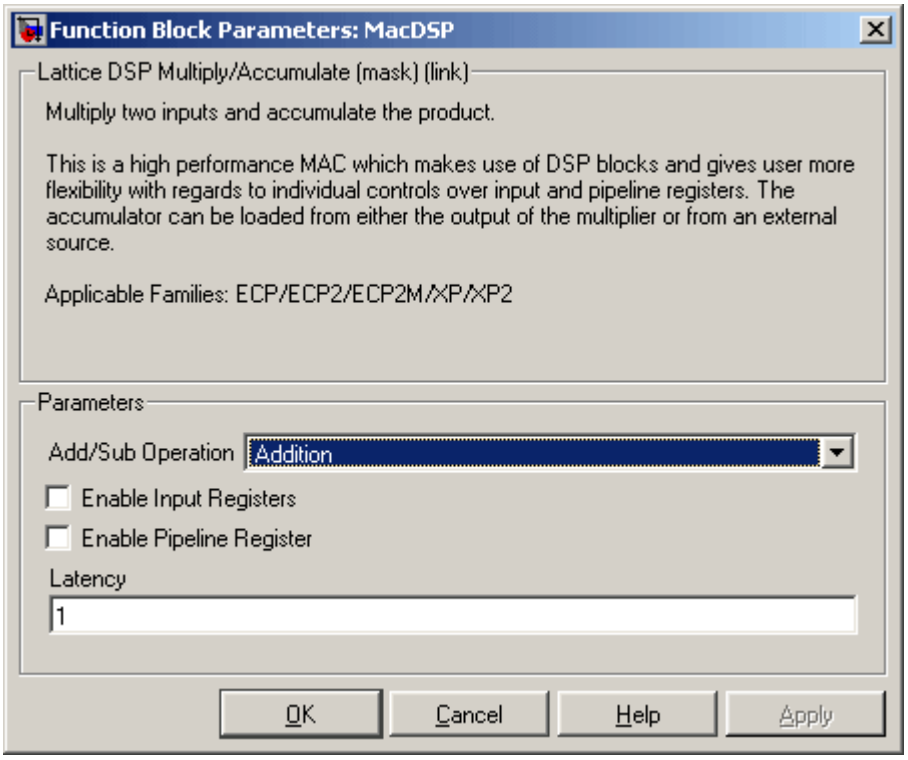
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

MacDSP



The DSP Multiply Accumulate (MacDSP) block multiplies two inputs and accumulates the product. This is a high performance MAC which makes use of sysDSP blocks and gives user more flexibility with regards to individual controls over input and pipeline registers. The accumulator can be loaded from either the output of the multiplier or from an external source.

The user configurable parameters for this block are available in the following dialog box.



The block parameters are:

- **Add/Sub Operation**:

  Allows the implementation of adder or subtractor. Default is set to Addition.

- **Enable Input Registers**:

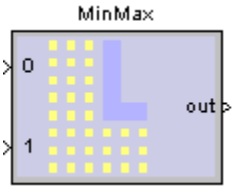
  Allows the implementation of registers on the inputs.

- **Enable Pipeline Register**:

  Allows the implementation of pipeline registers.

Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

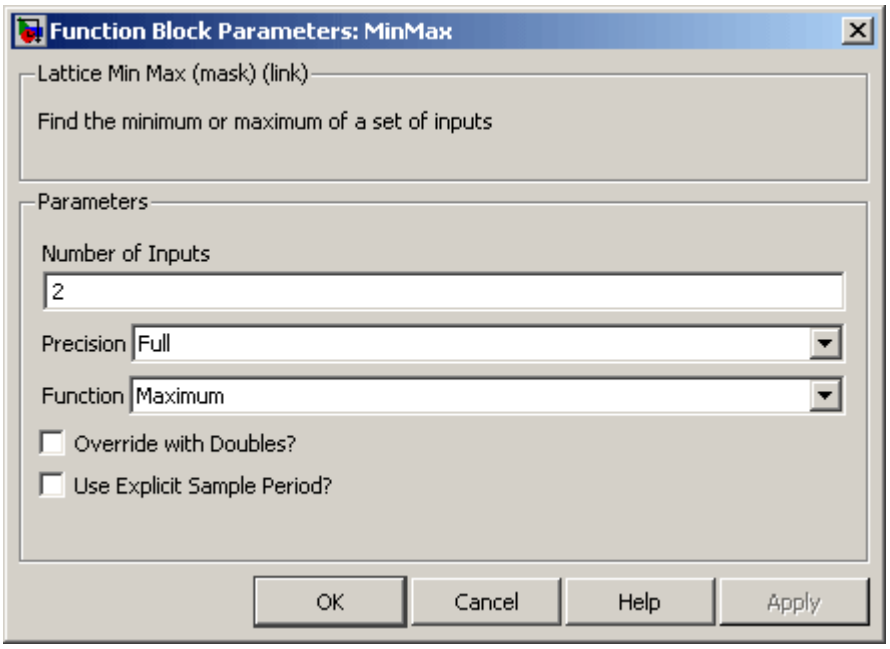
MinMax



The MinMax block will propagate the minimum or maximum value from the set of inputs to the output during each sample. The user can specify the number of inputs to be between 2 to 32. The inputs may have different formats with respect to width binary point and sign.

The output precision depends on whether full or user-defined precision is specified. The generated VHDL/Verilog code will perform the necessary binary point alignment and sign extension for the block to function correctly. If user-defined precision is specified, the full-precision value will be converted to the user-specified output format.

The user configurable parameters for MinMax are available in the following dialog box.



Block parameters are:

- **Number of Inputs**:

  This is the number of input ports assigned to the block. The number is user-selectable, but must be between 2 and 32. If the entered value is outside this range an error message is generated.

- **Precision**:

  Two values are possible: Full or User Defined. If Full is selected, output precision is computed based on the inputs. User Defined precision allows you to define output width, binary point, numeric type, quantization, and overflow treatment.

- **Output Width**:

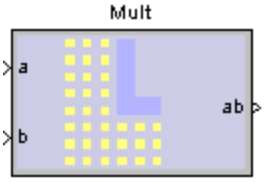
Width, in bits, of the output word. This parameter is accessible when User Defined Precision is selected.

- **Function**:

  This parameter selects the mode of operation. If Minimum is selected, the block always outputs the minimum input value. If Maximum is selected, the block outputs the maximum output value.

Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Mult



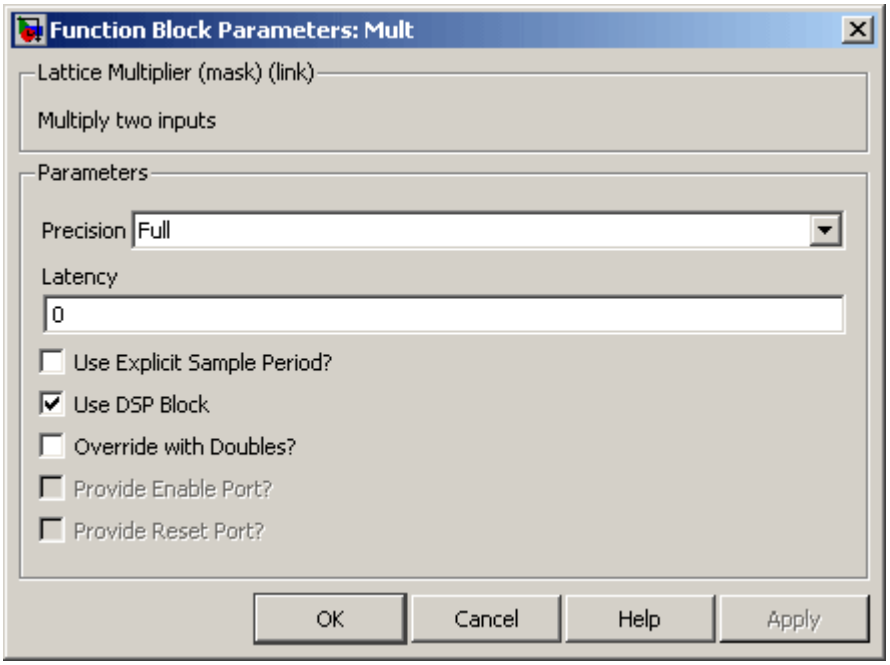
The Multiplier block provides a product of two input values. The multipliers can be implemented in either the sysDSP block or the FPGA fabric. If “Use DSP Block” is checked, the ispDSP block will be used to implement the multiply; otherwise the synthesis tool will choose how to implement the multiply.

If a latency greater than zero is selected and “Use DSP Block” is checked, the internal pipelines in the ispDSP block will be used as shown in the following table:

Table 11: Multiplier Pipeline Usage

Latency	Pipelines Used
0	None
1	Output
2	Input and Output
3	All
>3	All + FPGA registers

The user configurable parameters for the Multiplier block are available in the following dialog box.



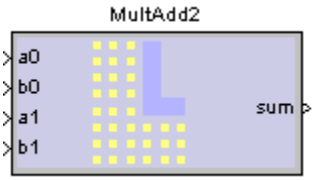
The block parameters are:

- **Use DSP Block**:

  If this box is unchecked, the multiplier will be implemented using the most appropriate method as determined by the synthesis tool. If checked, the multiplier will be implemented in the sysDSP block.

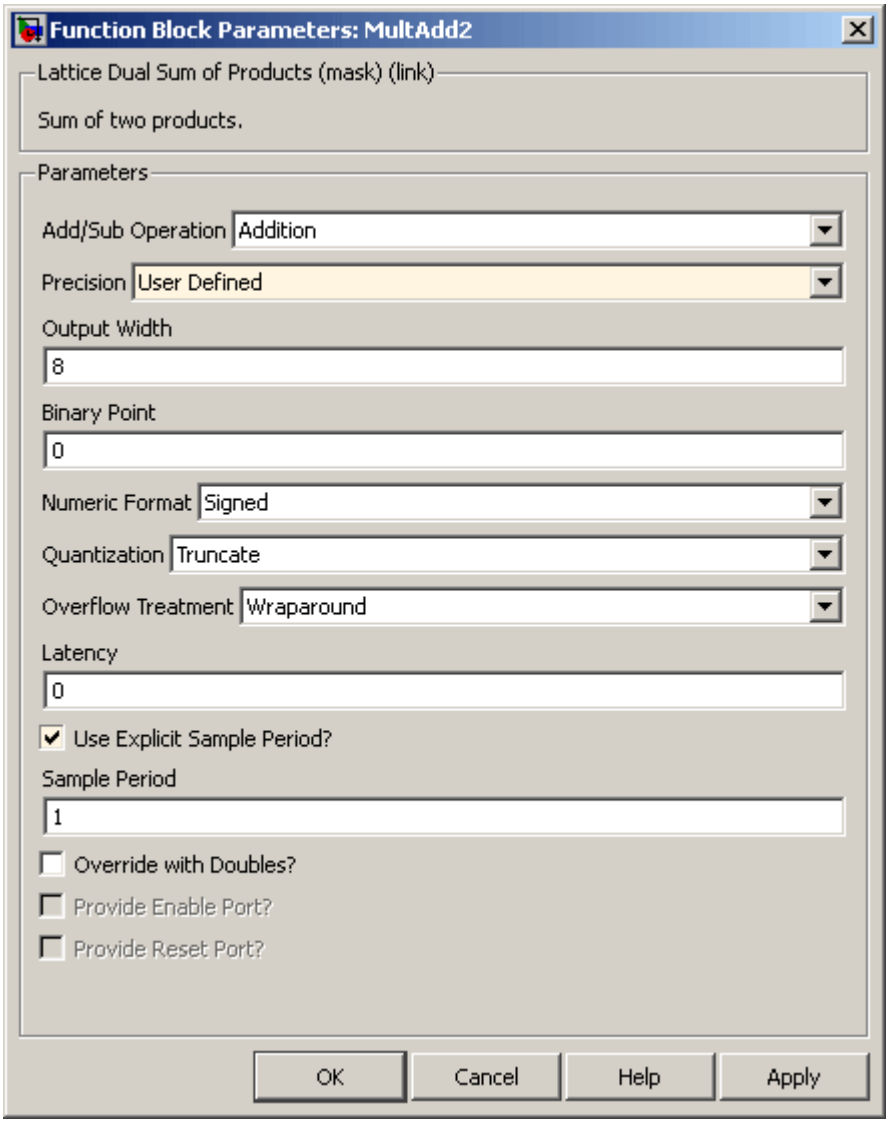
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

MultAdd2



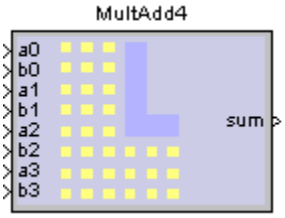
The MultAdd2 block provides a single block that output the sum of two products. This block will always be implemented in the ispDSP block.

The user configurable parameters for the MultAdd2 block are available in the following dialog box.



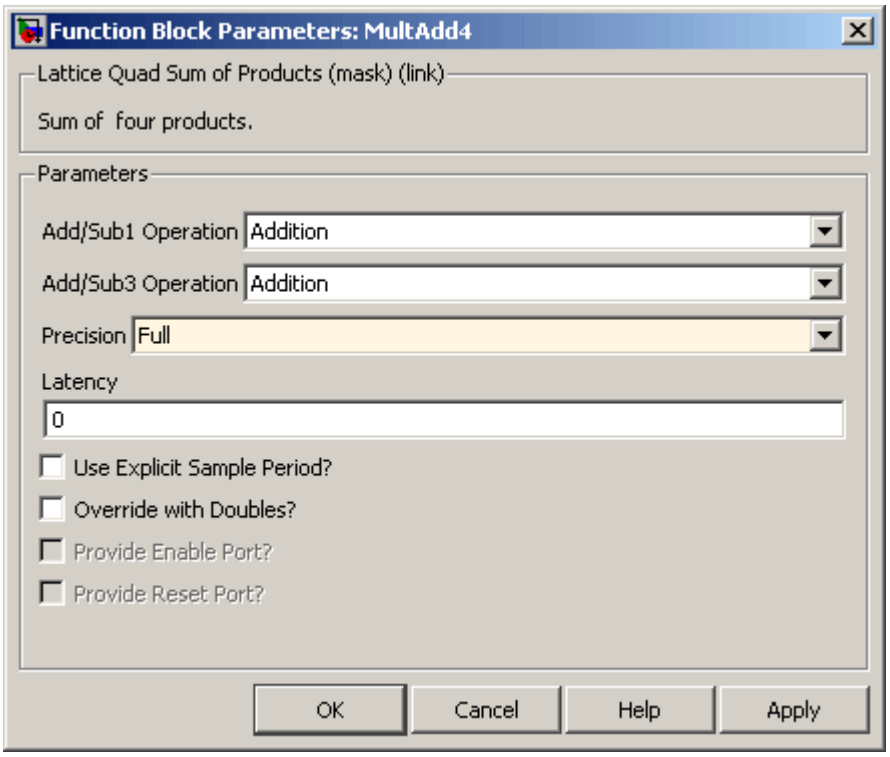
The block parameters are explained in Common Parameters for ispLeverDSP Blocks.

MultAdd4



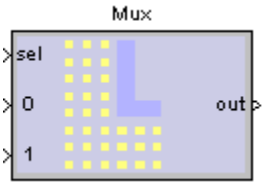
The MultAdd4 block provides a single block that output the sum of four products. This block will always be implemented in a sysDSP block.

The user configurable parameters for the MultAdd4 block are available in the following dialog box.



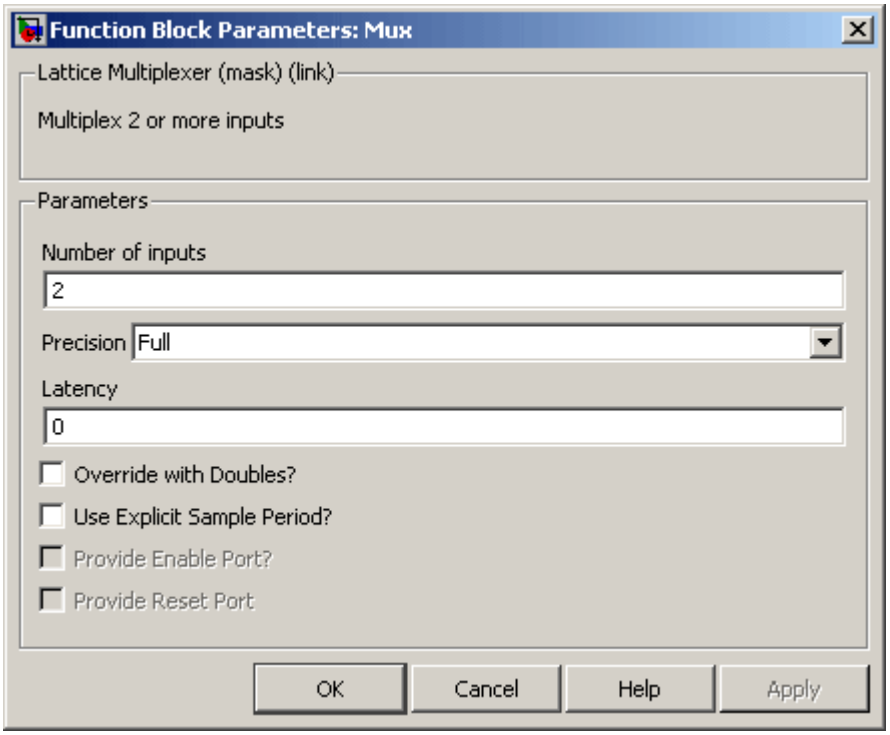
The block parameters are explained in Common Parameters for ispLeverDSP Blocks.

Mux



The Multiplexer block provides the ability to select one input from up to 32 different inputs. The select input must be unsigned. If the number of bits of the select input does not match the number of inputs for the mux, then an error is issued. All inputs must be of the same width, binary point, and numeric format, otherwise an error will be issued.

The user configurable parameters for the Multiplexer block are available in the following dialog box.



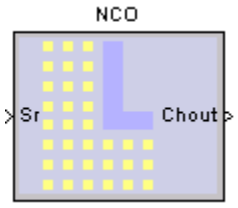
The block parameters are:

- **Number of inputs**:

  A positive integer that specifies the number of inputs for the mux. The value must be in the range of 2 to 32 inclusive.

Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

NCO



Numerically Controlled Oscillators (NCOs), a.k.a. Direct Digital Synthesizers (DDS), offer several advantages over other types of oscillators in terms of accuracy, stability and reliability. In addition, NCOs provide a flexible architecture that enables easy programmability (on the fly frequency and phase changes) and provides a trade-off between performance and resource requirements. This diagram shows a generic implementation diagram for the NCO. The diagram shows a single channel NCO, with FSK and PSK inputs and a full wave look-up table. The optional blocks for dithering and trigonometric correction are also shown in the diagram.

The Lattice NCO IP core supports all LatticeECP/EC, LatticeXP, LatticeSC/M, LatticeECP2/M devices. You can refer to the NCO IP Core User Manual on the Lattice web site for more information.

The NCO IP core has the following features:

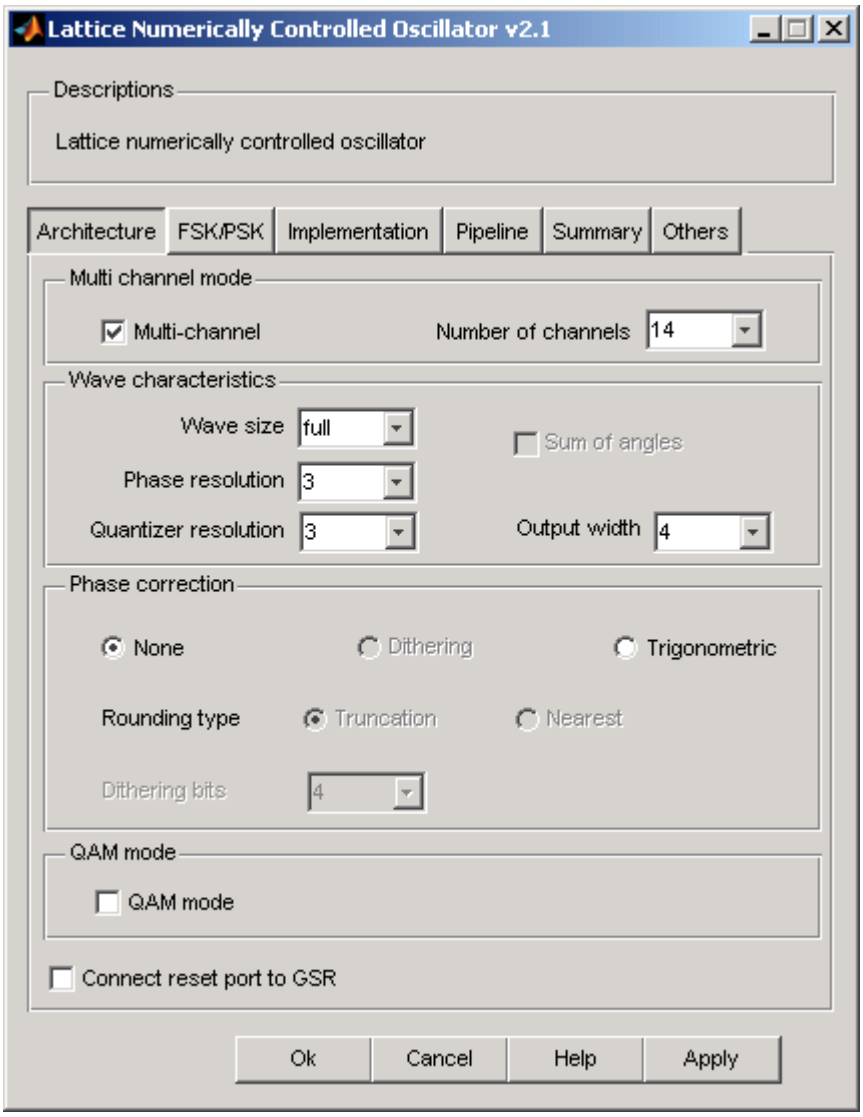
- Multi-channel support up to 16 channels
- Variable phase increment
- Variable phase offset input
- Variable phase resolution
- Variable phase quantization (3 to 20 bits)
- Variable amplitude resolution (4 to 32 bits)
- Full wave, 1/2 wave, or 1/4 wave architecture
- Optional sum of angles (SOA) optimization for memory
- Optional Phase Dithering Correction (up to 4 bits)
- Optional Trigonometric Correction for SFDR up to 115 dB
- Optional Quadrature Amplitude Modulation (QAM).

Note

This IP core is available for evaluation in the MATLAB/Simulink environment. A netlist will be generated for simulation purposes. An IP license must be purchased to implement the IP into a Lattice FPGA. Information on all Lattice IP Modules, including data sheets, brochures, and downloads, can be found on the Lattice Semiconductor Corporation web site at the following URL:

http://www.latticesemi.com/products/intellectualproperty/index.cfm

The user configurable parameters for NCO are available in the following dialog box.



Architecture Tab

- **Multi-channel**: Determines whether multiple channels are supported.
- **Number of Channels**:

  Range is 2 - 16. This parameter denotes the number of NCO channels. Valid only if Multi-channel is selected.

- **Wave size**:

  Selection between full, half and quarter. This parameter determines size of the sine wave stored in the look-up table.

- **Sum of angles**:

  Determines whether sum of angles method is used for memory reduction.

- **Phase resolution**:

  Range is 3 - 32. Maximum phase resolution of the NCO expressed in bits. This also defines the accumulator width.

- **Quantizer resolution**:

  Phase quantizer resolution: The output of the phase accumulator is quantized to this resolution before addressing the trigonometric look-up table. This also determines the depth of the trigonometric look-up table. The maximum value supported is 16 bits without sum of angles usage and 20 bits if the sum of angles method is employed. This resolution must be less than or equal to Phase resolution.

- **Output width**:

  Range is 4 - 32. Width of the output data. This parameter defines the width of sine and cosine outputs.

- **Phase correction**:

  Select between None, Dithering, and Trigonometric. Phase correction method for SFDR improvement. “Trigonometric” option is not available if Sum of angles is selected.

- **Rounding type**:

  Selection between Truncation and Nearest. Rounding type used for quantizing the phase accumulator output. This is valid only if Phase correction is “None”.

- **Dithering bits**:

  Number of dithering bits. This is used only if Phase correction is “Dithering”.

- **QAM mode**:

  This parameter indicates whether Quadrature Amplitude Modulation functionality is required. If “Yes”, QAM input and output ports are added to the IP and the parameter QAM input port width must be defined by user.

- **Connect reset port to GSR**:

  If this option is checked, the GSR is instantiated and used to route the NCO’s input. Using GSR rstn improves the utilization and performance of the NCO. However if GSR is used, an active input in rstn will reset most of the FPGA components as well. This option must be checked to enable the hardware evaluation capability.

FSK/PSK Tab

- **FSK input**:

  Select between Constant and Variable. This parameter defines whether the FSK input is a constant or a variable. If “Variable” is selected, FSK input ports are added and the parameter FSK input port width must be defined by the user. If “Constant”, the Phase increment parameter must be defined.

- **Phase increment for each channel**:

Range is 1 to 2^(Phase resolution-1). Phase increment value. This value determines the phase increment that is added to the phase accumulator at every clock. This decides the frequency of the output waveform. In multi-channel modes, a phase increment must be specified for each channel.

- **FSK input port width**:

  Width of the fskin port. This must be less than the parameter Phase resolution. Range is 3 to (Phase resolution -1). Default is 16.

- **PSK input**:

  Select between None, Constant, and Variable. This parameter determines if Phase Shift Keying input is used and if used, whether it is a constant or variable. If “Constant” is selected, a fixed value defined by Phase offset is used for the increment. If “Variable”, PSK input ports are added and the user must define the parameter PSK input port width.

- **Phase offset for each channel**:

  Phase offset value. Determines the phase offset that is added to the accumulated phase at every clock. This decides the phase of the output waveform. In multi-channel modes, a phase offset must be specified for each channel. Range from 1 to 2^(Phase resolution).

- **PSK input port width**:

  Width of the pskin port. This must be equal to or less than the parameter Phase resolution. Range 3 to Phase resolution. Default is 16.

Implementation Tab

- **Memory Type**:

  This parameter defines whether block or distributed memories are used. It provides the user with additional flexibility of memory/logic resource utilization.

- **Data output ports**:

  - **Sine**:

    This parameter determines whether the sine output port is available in the core. If QAM mode is “No” and Cosine is “No”, then Sine has to be “Yes”.

  - **Cosine**:

    This parameter determines whether the cosine output port is available in the core. If QAM mode is “No” and Sine is “No”, then Cosine has to be “Yes”.

- **Polarity**:

  This parameter defines polarity of the sine or cosine output. It could be positive or negative.

- **Optional I/O Ports**:

  - **ce**:

    Determines whether the input port ce (clock enable) is present.

- **sr**:

  Determines whether the input port sr (synchronous reset) is present.

- **clear**:

  Determines whether the input port clear is present. This signal clears the phase accumulator (or presets the accumulator with the fixed phase offset, if provided).

- **phout**:

  This option determines whether the optional phase output is required. If “Yes”, the output port phout is added.

- **outvalid**:

  This option determines whether the output port out-valid is present.

- **qout**:

  This option determines whether qout port is present. This port is available only if QAM mode is selected.

- **Use DSP Block**:

  This parameter defines whether sysDSP blocks are used. This option is available only for Trigonometric correction, Sum of angles and QAM modes.

Pipeline Tab

- **Register after phase shift adder**:

  This option places a register after the phase shift adder if PSK input is “Variable”. This prevents any performance degradation due to phase adder, but the output is delayed by one more clock cycle.

- **Register after phase dithering block**:

  This option places a register after the dithering block if “Dithering” is chosen for phase correction. This prevents any performance degradation due to phase dithering, but the output is delayed by one more clock cycle.

- **Register after phase quantizer**:

  This option places a register after the phase quantizer if Wave size is “quarter”. This prevents any performance degradation in quarter-wave modes, but the output is delayed by one more clock cycle.

- **Memory output register**:

  This option selects the optional memory output register in the sysMEM™ EBR block RAMs. This improves the performance of the trigonometric look-up tables, especially when multiple sysMEM EBR blocks are used for the look-up-table. The output is delayed by one more clock cycle if this option is chosen.

- **Additional memory data register for half and quarter waves:**

  This option places an additional register in the memory data path. This prevents any performance degradation in half-wave or quarter-wave modes. The output is delayed by one more clock cycle if this option is chosen.

Summary Tab

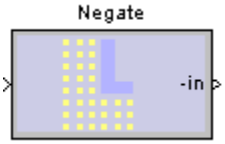
This tab shows a summary of your configurations.

Others Tab

- **Manually connect reset port**:

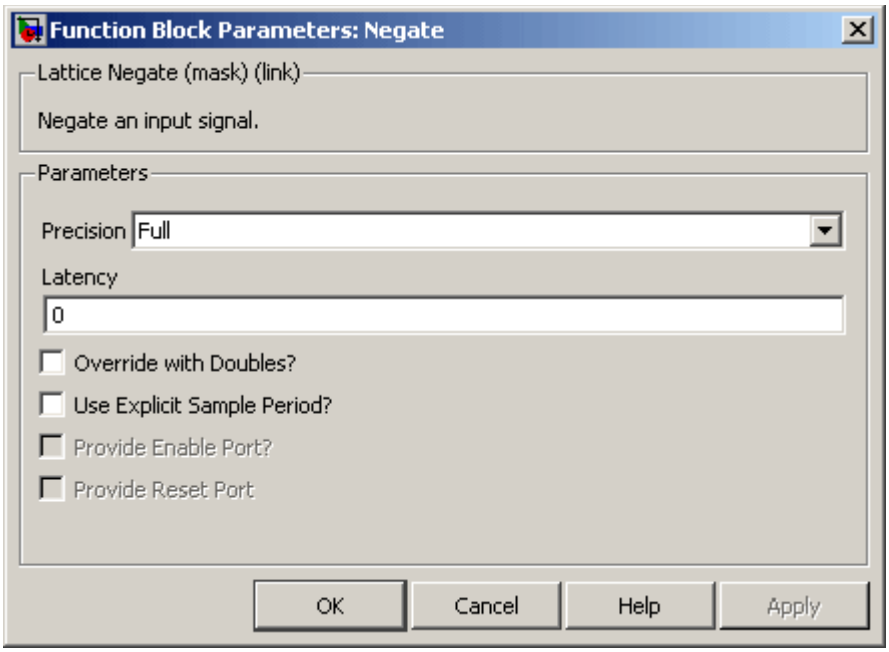
  Select this option to connect asynchronous reset port manually. Otherwise, the port will not be present.

Negate



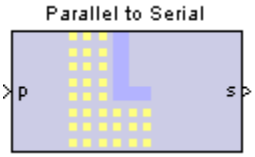
The Negate block provides an output that is the arithmetic negation of the input. The output is the 2's complement of the input.

The user configurable parameters for the Negate block are available in the following dialog box.



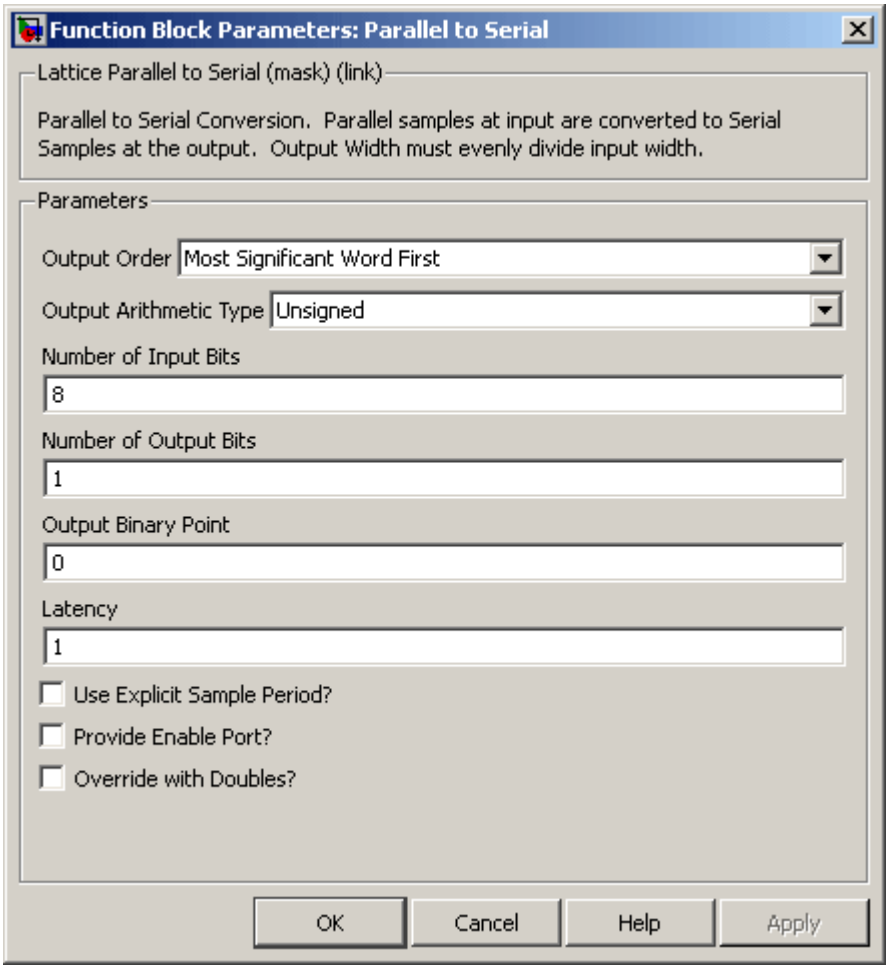
The block parameters are explained in Common Parameters for ispLeverDSP Blocks.

Parallel to Serial



The Parallel to Serial Block converts an input word into a series of serial output bits. Both the number of input bits and the number of output bits are selectable via parameters to the block, but the number of input bits must always be an integral multiple of the number of output bits. The block can be implemented so that either the most significant or the least significant bits of the input word are the first bits to appear on the output.

The user configurable parameters for the Parallel to Serial Block are available in the following dialog box.



Block parameters are:

- **Output Order**:

  If Most Significant Word First is selected, the most significant bits in the input word are the first values to appear at the output. If Least Significant

Word is selected the least significant bits in the input word are the first values to appear at the output.

- **Number of Input Bits**:

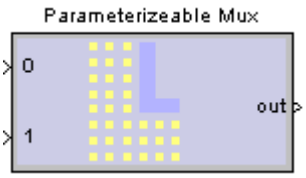
  This is the width, in bits, of the input word. It must always be evenly divisible by the number of output bits.

- **Number of Output Bits**:

  This is the width of the value seen at the output port. It must always evenly divide the number of input bits.

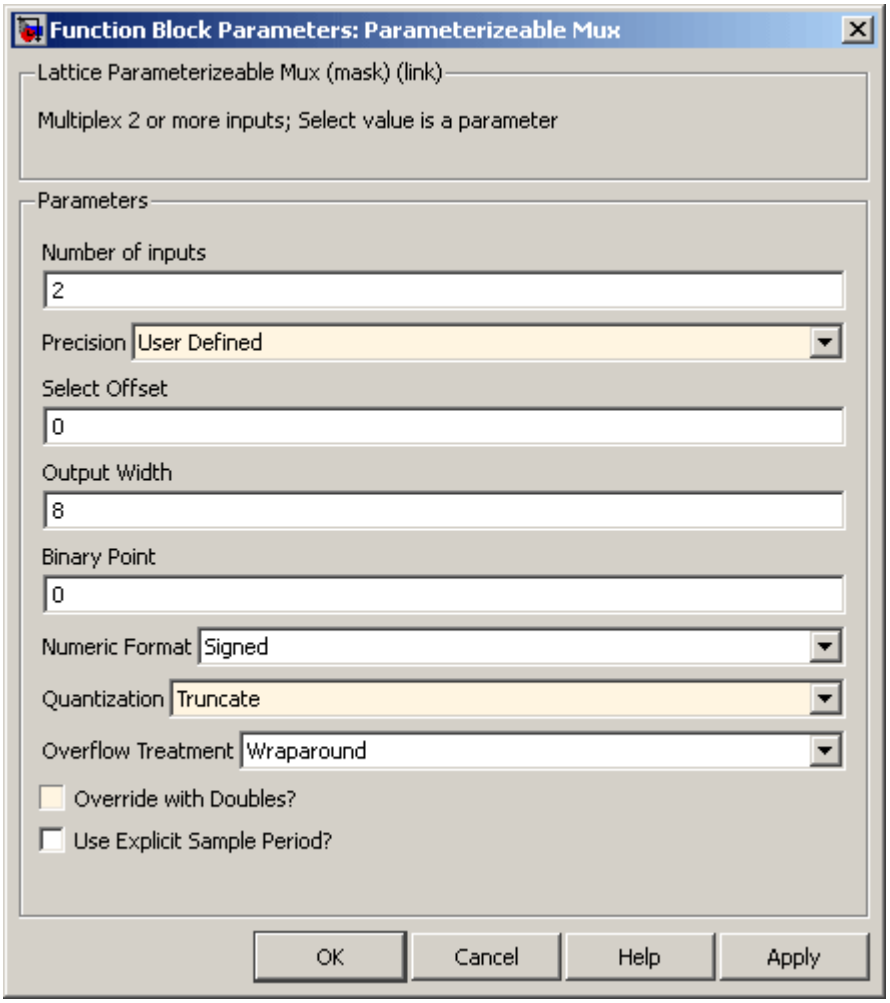
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Parameterizable Mux



The Parameterizable Multiplexer is a multiplexer which has a user-specified number of inputs and one output. Each input may be up to 52 bits wide, but all inputs must all have the same width. The width of the select input must be large enough to address each of the data inputs and may be either user-defined or automatically calculated from the number of data inputs. If the user-defined option is chosen for the select input and the user chooses a control width value too small to span the total number of inputs, an error message is generated If a user-defined width is specified that is larger that necessary to span all the inputs, then excess bits in the width are ignored by the block's logic. Regardless of the width of the select input, offsets range in value from 0 to one less than the number of inputs.

The user configurable parameters for the Serial to Parallel Block are available in the following dialog box.:



The block parameters are:

- **Number of Inputs**:

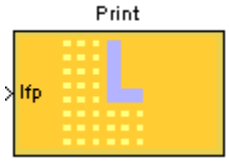
  This value is the number of inputs to the mode. All inputs must have the same width and the same binary point position, otherwise an error message is generated.

- **Select Offset**:

  This value addresses the input port that will be directed to the output port. Since this is a user-selected parameter, and does not change during a simulation, it implies that the implementation amounts to a hardwired connection of the selected input port to the output port.

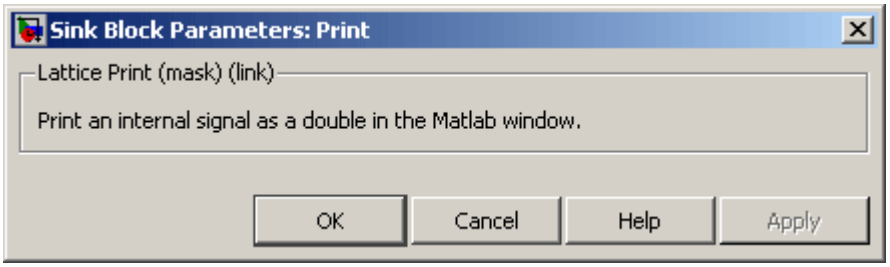
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Print

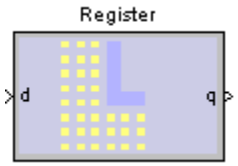


The Output Print block prints an internal signal as a double in the MATLAB window.

The user configurable parameters for the Output Print block are available in the following dialog box.

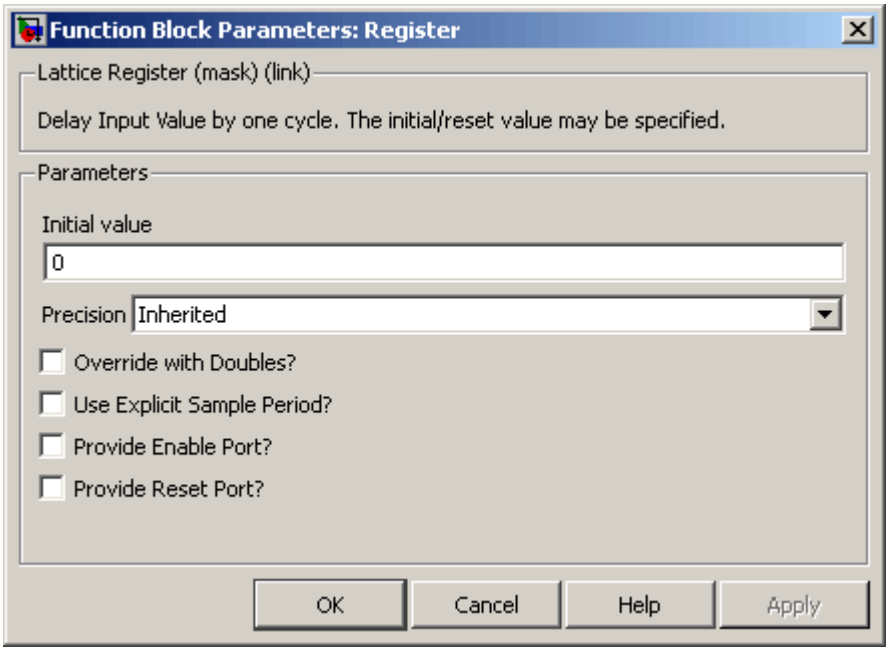


Register



The Register block provides a presettable D-type register.

The user configurable parameters for the Register block are available in the following dialog box.



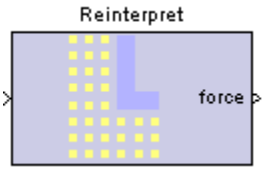
The block parameters are:

- **Initial value**:

This is the reset value for the register.

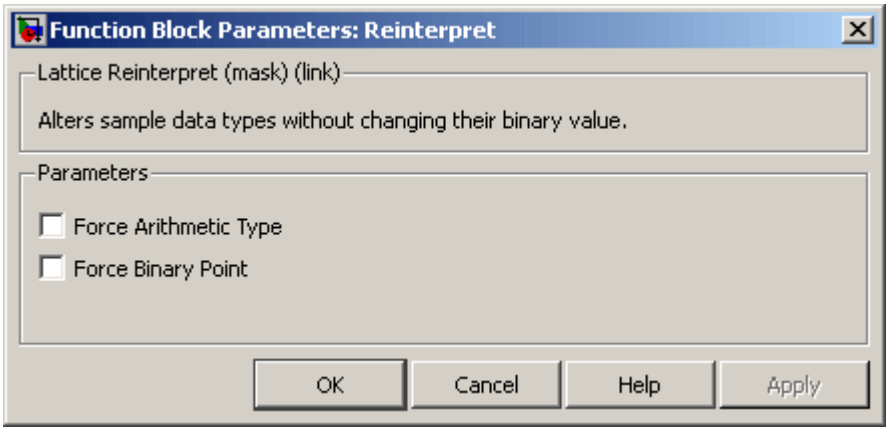
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Reinterpret



The Reinterpret Block Provides the means to change the data type of an input without changing its bit pattern. Signed or unsigned format may be forced regardless of the input format, and the binary point may be placed at an arbitrary location in the output.

The user configurable parameters for the Reinterpret block are available in the following dialog box.



The block parameters are:

- **Force Arithmetic Type**:

  When this box is checked the Output Arithmetic Type is exposed in the mask. When it is unchecked the Output Arithmetic Type parameter is hidden in the mask, and the block passes the format of the input to the output.

- **Output Arithmetic Type**:

  When enabled by checking the Force Arithmetic Type box, this parameter offers the choice of forcing the output format to Signed or Unsigned.

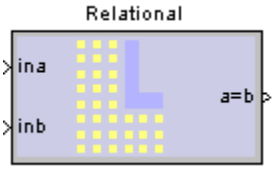
- **Force Binary Point**:

  Checking the Force Binary Point Box exposes the Binary Point parameter in the mask. Leaving it unchecked keeps the Force Binary Point parameter hidden in the mask, with the result that the binary point of the input is passed to the output.

- **Binary Point**:

  When enabled by checking the Force Binary Point box, this parameter forces the output binary point to the entered value.

Relational



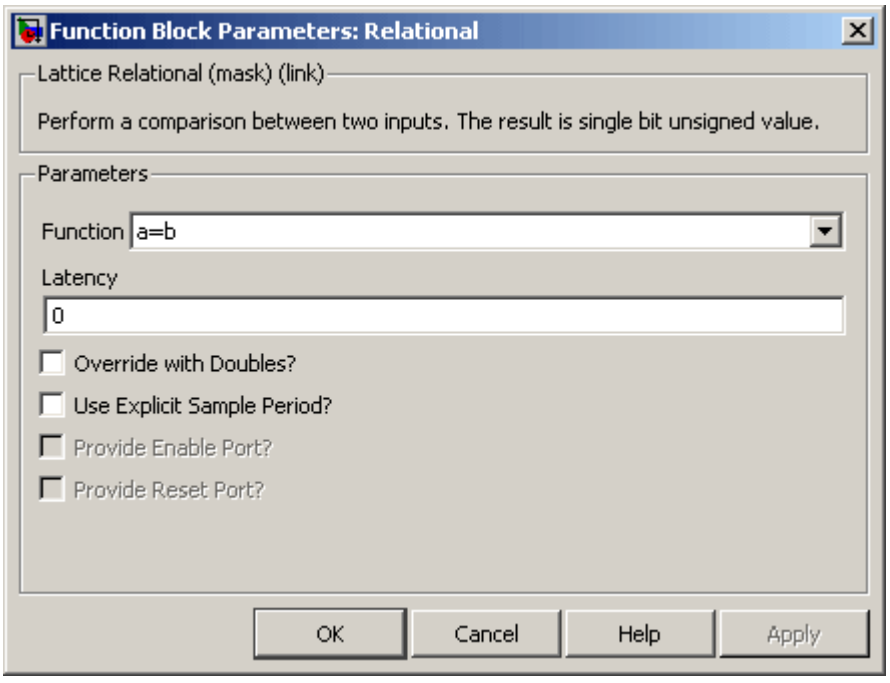
The Relational block performs a comparison between two inputs. The result is a single bit unsigned value.

The comparisons that can be performed are:

- equal (a=b)
- not equal (a!=b)
- greater than (a>b)
- greater than or equal (a>=b)
- less than (a<b)
- less than or equal (a<=b)

All inputs must be of the same width, binary point, and numeric format, otherwise an error will be issued.

The user configurable parameters for the Relational block are available in the following dialog box.



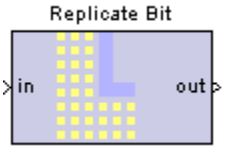
The block parameters are:

- **Function = [a=b, a!=b, a>b, a>=b, a<b, a<=b]**:

  This selects the comparison function to be performed.

Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

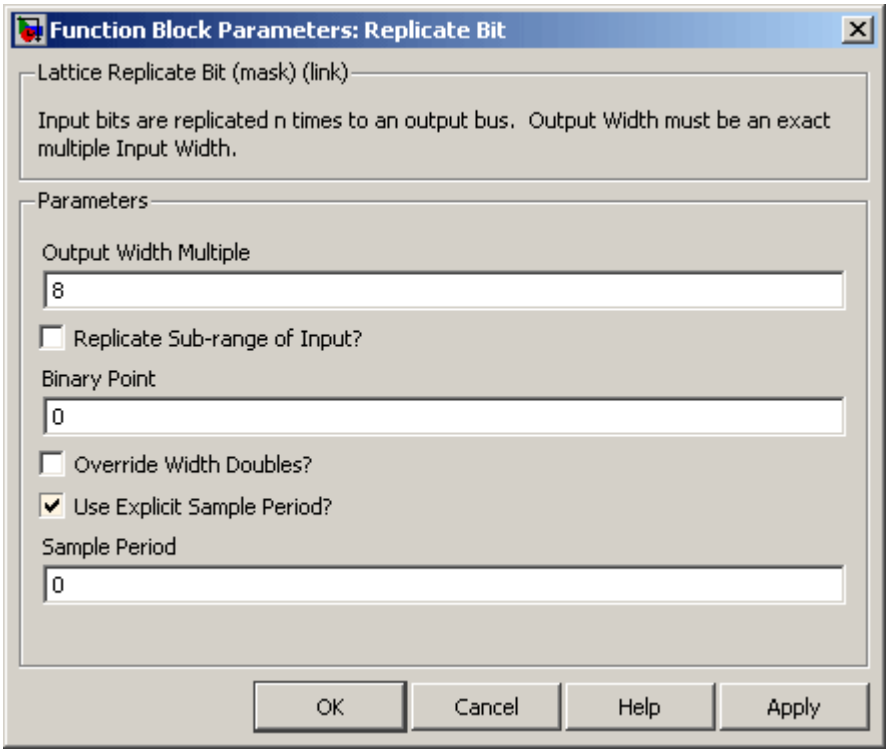
Replicate Bit



This block has one input and one output. It fans multiple copies of an input to the output such that the output is a parallel word containing n copies of the input. The input may consist of one or more bits; and the n umber of bits in the output is an integral multiple, n, of the number of input bits. In no case may the number of input or output bits exceed 52 bits (the width of a double data type in MATLAB).

Inputs may have either signed or unsigned numeric format. The output numeric format is user-defined, and may be either signed or unsigned.

The user configurable parameters for the Replicate Bit Block are available in the following dialog box.



Block parameters are:

- **Output Width Multiple**:

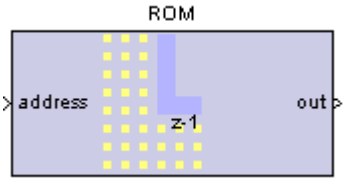
  This number, multiplied by the number of input bits gives the number of bits in the output word. The value entered for this parameter must be an integer greater than or equal to one, but must not so large as to cause the Output Width to exceed 52 bits.

- **Replicate Sub-range of Input?**:

  Choose per design specification.

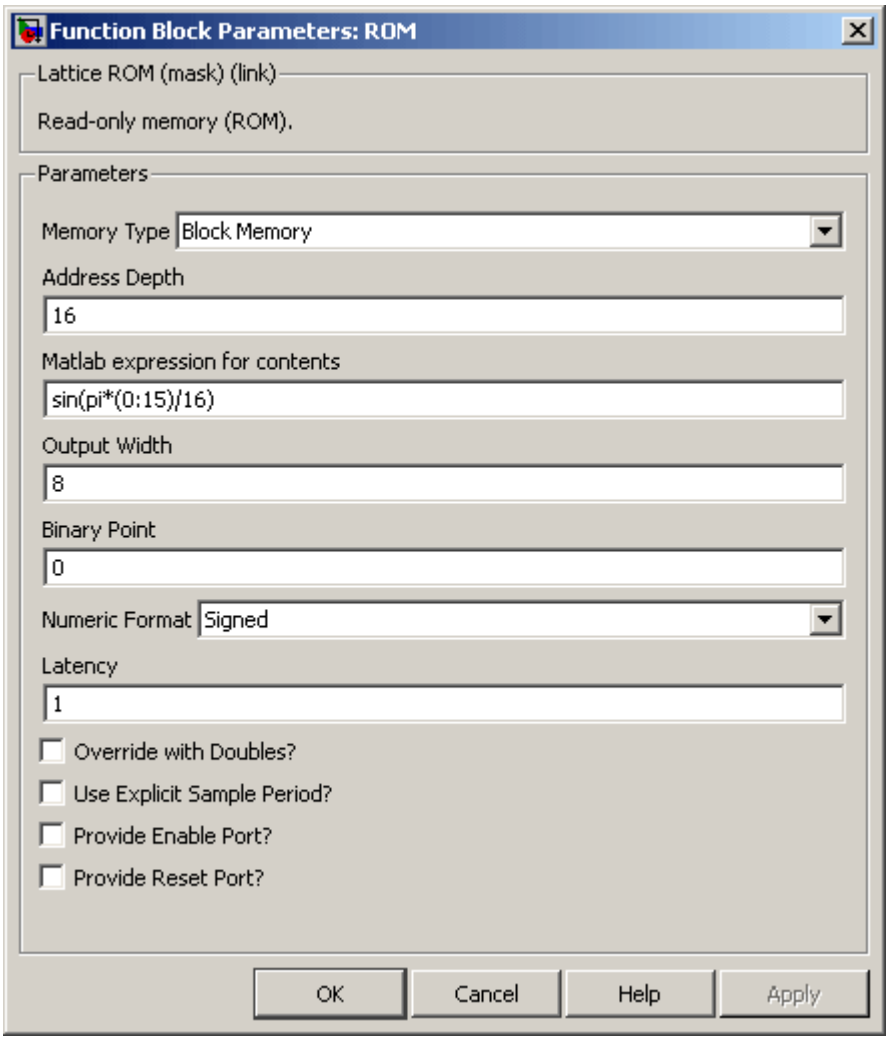
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

ROM



The ROM block provides a Read-Only Memory block.

The user configurable parameters for the ROM block are available in the following dialog box.



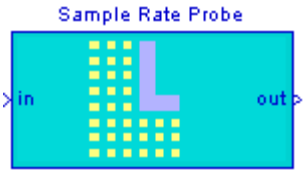
The block parameters are:

- **Memory Type = [Block Memory, Distributed Memory]**
- **Address Depth**: Depth of memory.
- **MATLAB Expression for Initial Contents**:

The value for the ROM data. This can be any legal MATLAB expression that produces a vector of the appropriate length. If the result of the evaluation is less than the number of words, the remaining words will be set to zero. If it is larger than the number of words, the excess values will be ignored.

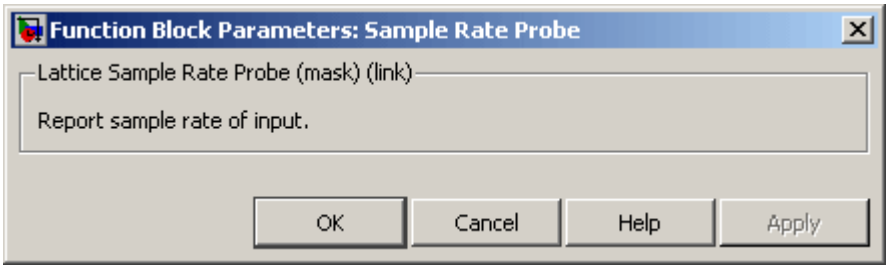
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Sample Rate Probe

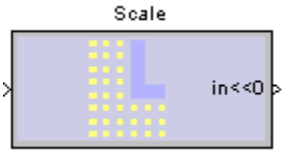


The Sample Rate Probe block outputs the sample rate of the input as a double value.

The user configurable parameters for the Sample Rate Probe block are available in the following dialog box.

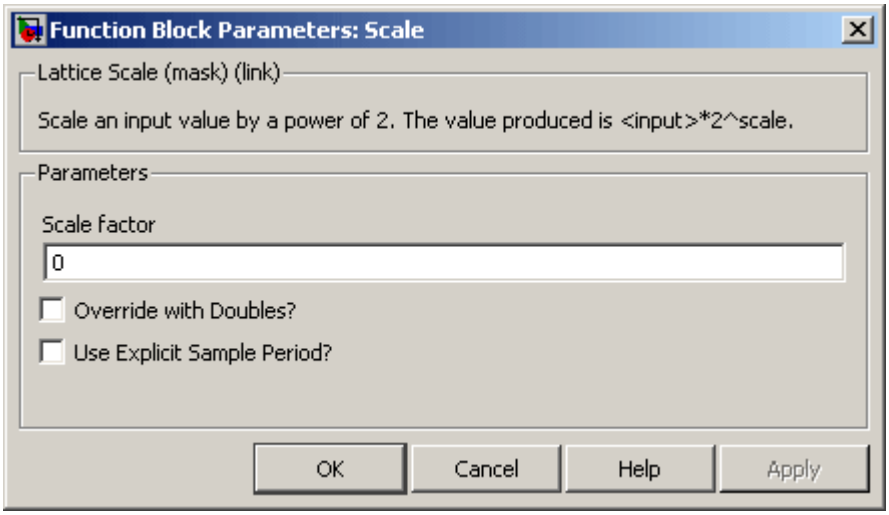


Scale



The Scale block scales the input value by a specified amount. This block only affects the value of the binary point, and does not generate any hardware.

The user configurable parameters for the Scale block are available in the following dialog box.



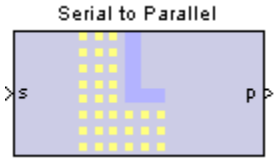
The block parameters are:

- **Scale factor**:

  This is an integer that indicates how many bits to scale the input. A positive value scales the value up, and a negative value scales down. The output value is input*2^(scale factor).

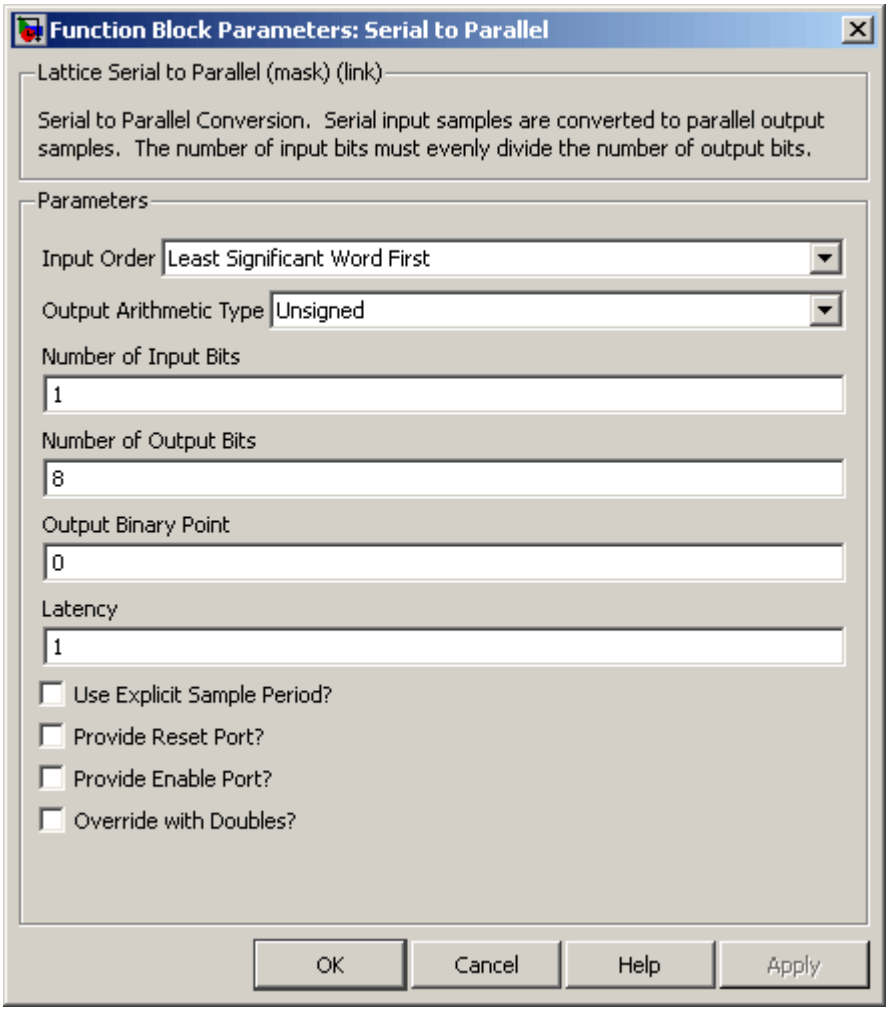
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Serial to Parallel



The Serial to Parallel block converts a series of input bits to parallel output. Both the number of input bits and the number of output bits are selectable, but the number of output bits must always be an integer multiple of the number of inputs bits. Input bits may be accumulated in modes which cause the first input bits to appear as the most significant or the least significant bits in the output word. The output can be selected as a signed or unsigned value and the position of the output binary point can be designated as well.

The user configurable parameters for the Serial to Parallel Block are available in the following dialog box.



Block parameters are:

- **Input Order**:

  Selecting Least Significant Word first cause the first inputs bits to be received on the input to be accumulated into the least significant bit positions of the output word. Conversely, selecting Most Significant Word first causes the first input bit received on the input port to be accumulated into the most significant bit positions of the output word.

- **Output Arithmetic Type**:

  Possible choices for this parameter are signed and unsigned. The parameter is applied to the parallel output value.

- **Number of Input Bits**:

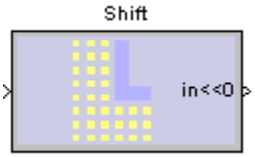
  This is the width, in bits, of the input word. Its value can range from 1 to the output width, subject to the constraint that the number of input bits evenly divide the number of output bits.

- **Number of Output Bits**:

  This is the width, in bits of the output word. It must be an integral multiple of the number of input bits.

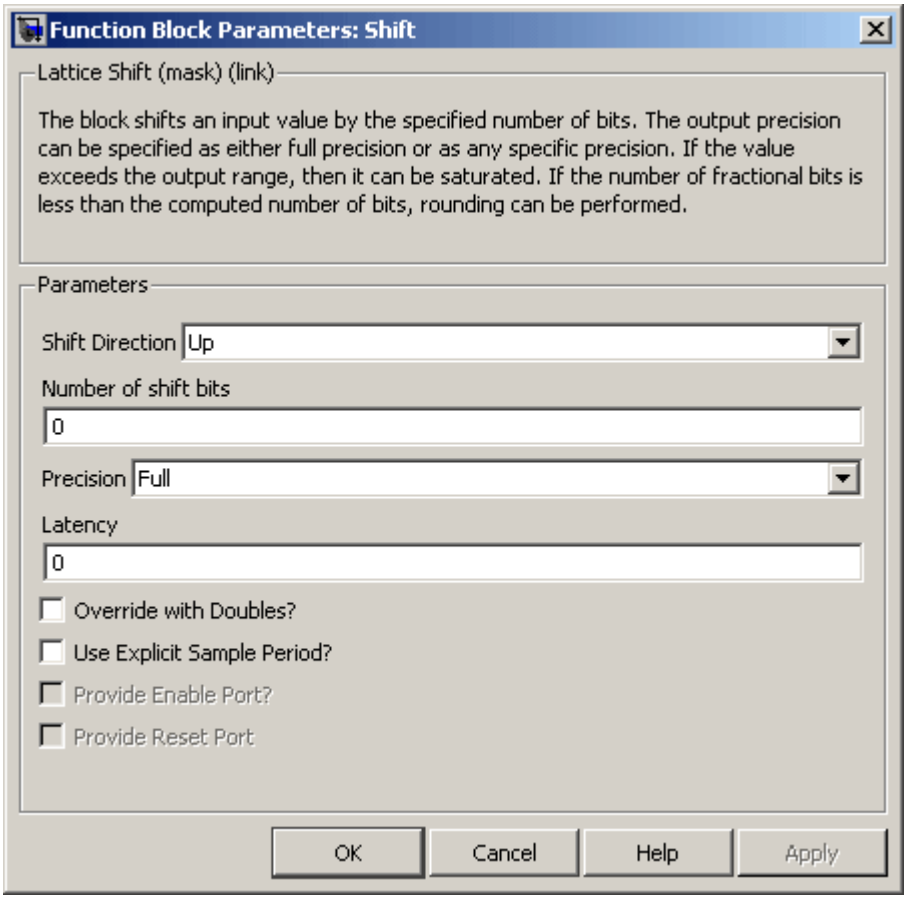
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Shift



The Shift block shifts an input value by the specified number of bits. The Down shift moves the input toward the least significant bit, with appropriate sign extension. Bits shifted beyond the output width are discarded. The Up shift moves the input toward the most significant bit with zero padding of the least significant bits. Bits shifted beyond the output width are discarded.

The user configurable parameters for the Shift block are available in the following dialog box.



The block parameters are:

- **Shift direction = [Up, Down]**:

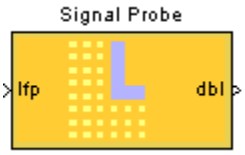
  Direction of the shift operation.

- **Number of shift bits**:

This is an integer that indicates how many bits to shift the input. The absolute value must be less than the number of output bits.

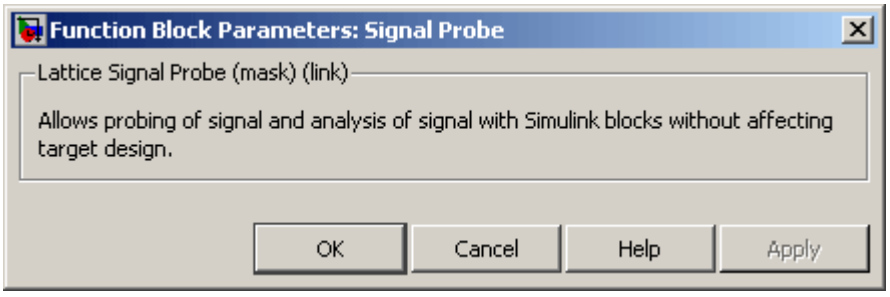
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Signal Probe

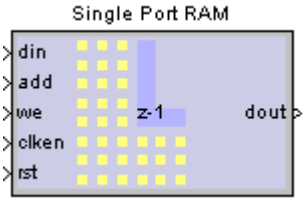


The Signal Probe block converts the input to a double value. It has no impact on hardware, and does not affect the generated testbench. This is useful for examining signals during the simulation phase without affecting the netlisting process.

The user configurable parameters for the Sample Rate Probe block are available in the following dialog box.

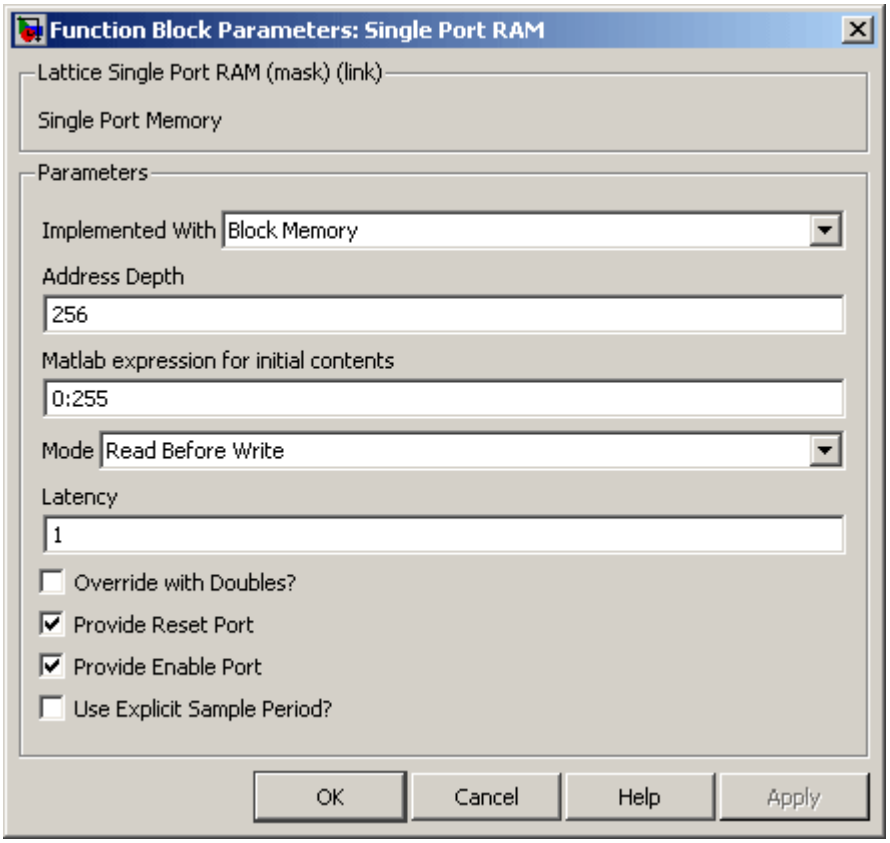


Single Port RAM



The Single Port Ram block provides a single port memory. The write mode for the memory will always be read-before-write. The reset pin will always be a synchronous reset. For more detailed information, see technical notes of specific device family on Lattice web site.

The user configurable parameters for the Single Port RAM block are available in the following dialog box.



The block parameters are:

- **Implemented With**:

  Choose between Block Memory or Distributed Memory.

- **Address Depth**: Depth of memory.

- **MATLAB Expression for Initial Contents**:

  Initial value for the memory data. This can be any legal MATLAB expression that produces a vector of the appropriate length. If the result of the evaluation is less than the number of words, the remaining words will be set to zero. If it is larger than the number of words, the excess values will be ignored.

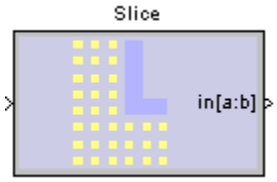
- **Mode**:

  Memory options include Read Before Write, Normal, and Write Through.

- **Latency**: The minimum programmable latency is 1.

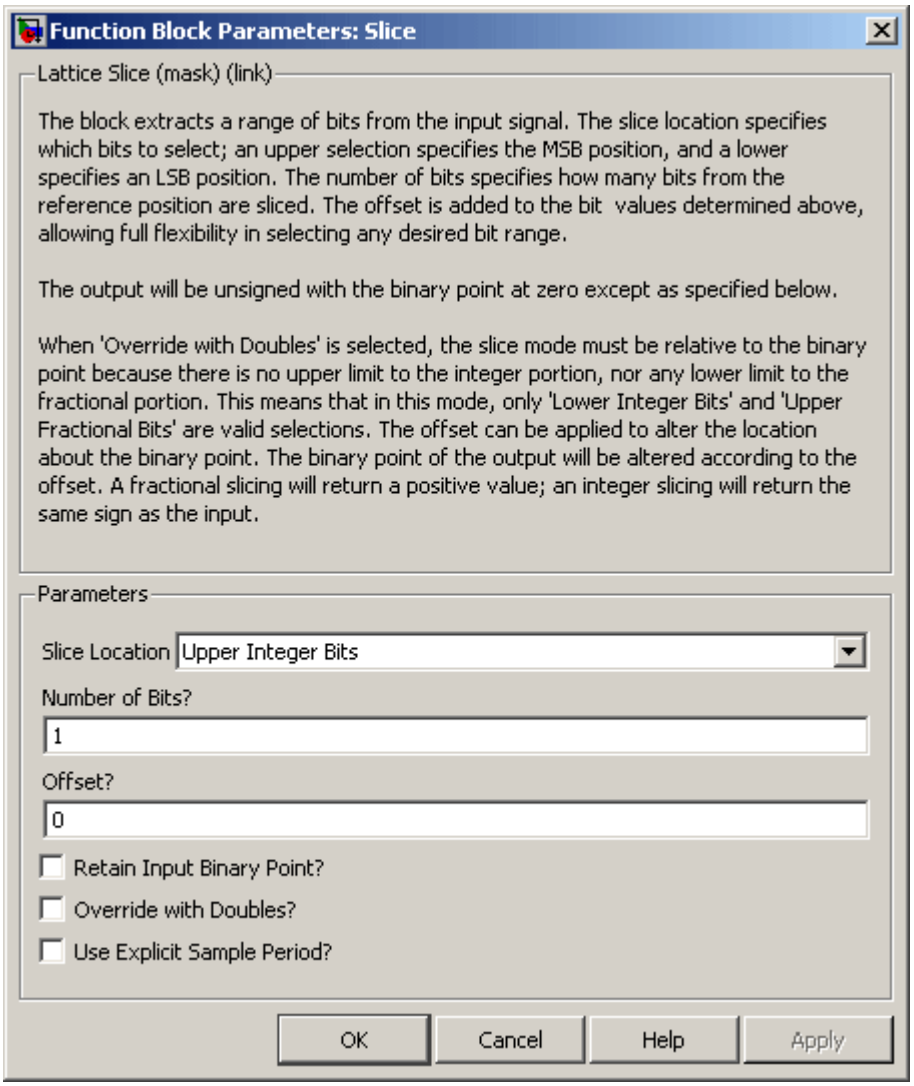
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Slice



The Slice block allows the extraction of a portion of a bus. Any number of bits from 1 to the bus width can be extracted. The output data type will be unsigned with the binary point at zero. If the input is signed, then the sign bit is included in the integer bits.

The user configurable parameters for the Slice block are available in the following dialog box.



The block parameters are:

- **Slice Location = [Upper Integer Bits, Lower Integer Bits, Upper Fractional Bits, Lower Fractional Bits]**:

  This determines the reference point for the slicing. If “Upper” is selected, then the uppermost n bits of the specified word fragment (integer or

fractional) are sliced, where n is the number of desired bits. If “Lower” is selected, then the lowermost n bits are sliced.

- **Number of Bits**:

  This specifies the number of bits to be sliced.

- **Offset**:

  This value is added to or subtracted from the bit location calculated from the above two parameters. If an “Upper” selection is made, a positive offset selects bits below the upper selection (offset is subtracted from the upper bit location); if a “Lower” selection is made, then a positive offset selects bits above the lower selection (offset is added to the lower bit location). This allows slicing of the bus anywhere regardless of the reference point.

  For example, if the input word is 8 bits in width, with the binary point at 4, then all of the following will select the same 2 bits (bits 6:5):

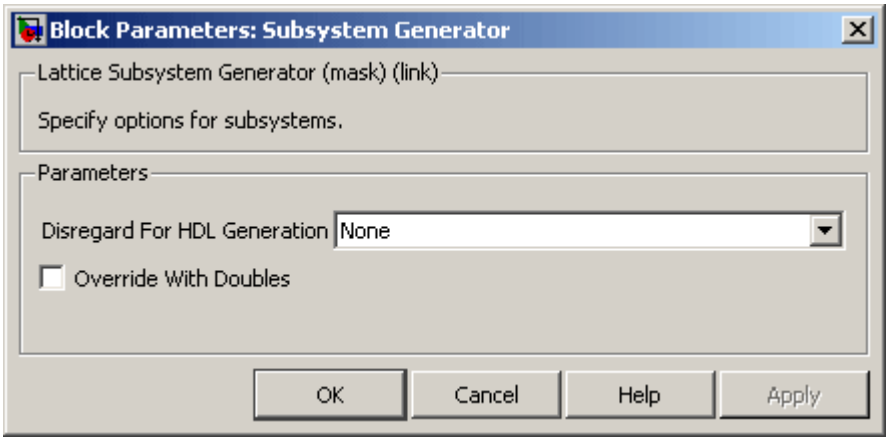
  - Upper Integer Bits, 2 bits, Offset=1
  - Lower Integer Bits, 2 bits, Offset=1
  - Upper Fractional Bits, 2 bits, Offset=-3
  - Lower Fractional Bits, 2 bits, Offset=5

Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Subsystem Generator

The Subsystem Generator block provides various options at the subsystem level. This block provides a way of specifying options at a localized scope level. The options specified in a Subsystem Generator block apply to all blocks within the subsystem. This includes any lower level subsystem blocks within the subsystem in which the Subsystem Generator block is placed.

The Subsystem Generator mask are available in the following dialog box.



The block parameters are:

- **Disregard For HDL Generation = [None, Entire Subsystem, Gateway Outs Only]**:

  This option is provided to allow localized scope control of the HDL generation.

  - **Entire Subsystem**

    This option will prevent any HDL code from being generation for all blocks plus any lower-level blocks in the subsystem.

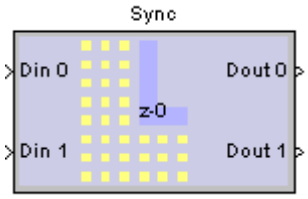
  - **Gateway Outs Only**

    This option will remove Gateway Out blocks from the portmaps of the generated HDL code.

- **Override With Doubles**:

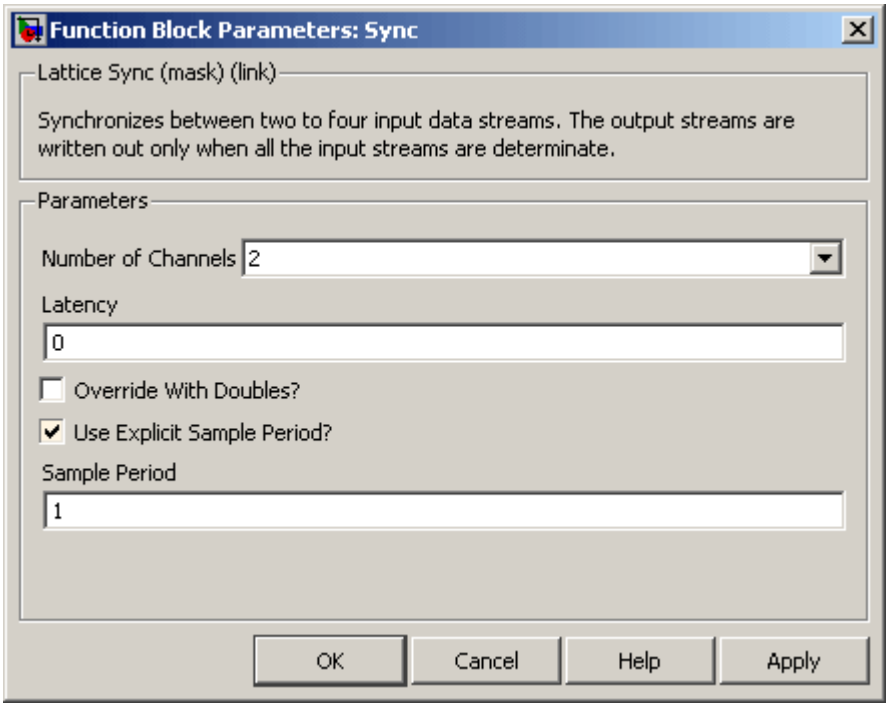
  This option is provided to allow 'Override With Doubles' to be specified at the subsystem level. If the option is selected, this option overrides the 'Override With Doubles' option in all blocks plus lower-level blocks in the subsystem.

Sync



The Sync block takes two, three, or four input channels and synchronizes them. The data is then presented in synchronized form on corresponding output channels. This is accomplished by inserting delay lines between the input and output of each channel. Factors for the delay lines are automatically computed such that the first valid data sample of each input channel is presented simultaneously at the output channels. An optional Latency parameter specifies a minimum delay for all channels.

The user configurable parameters for the Sync block are available in the following dialog box.



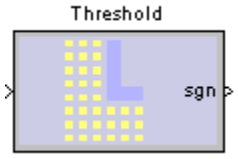
The block parameters are:

- **Number of Channels**:

  Specify the number of input and output channels to be synchronized.

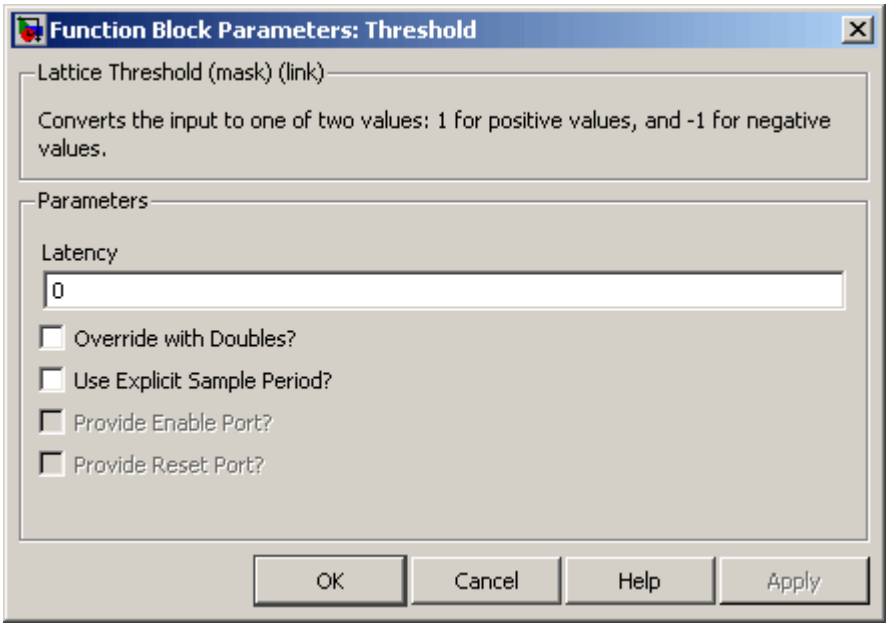
Other parameters used by this block are explained in the Common Parameters for ispLeverDSP Blocks.

Threshold



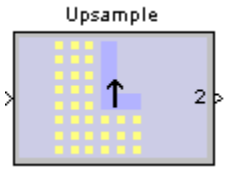
The Threshold block provides a two-bit signed output that is the sign of the input.

The user configurable parameters for the Threshold block are available in the following dialog box.



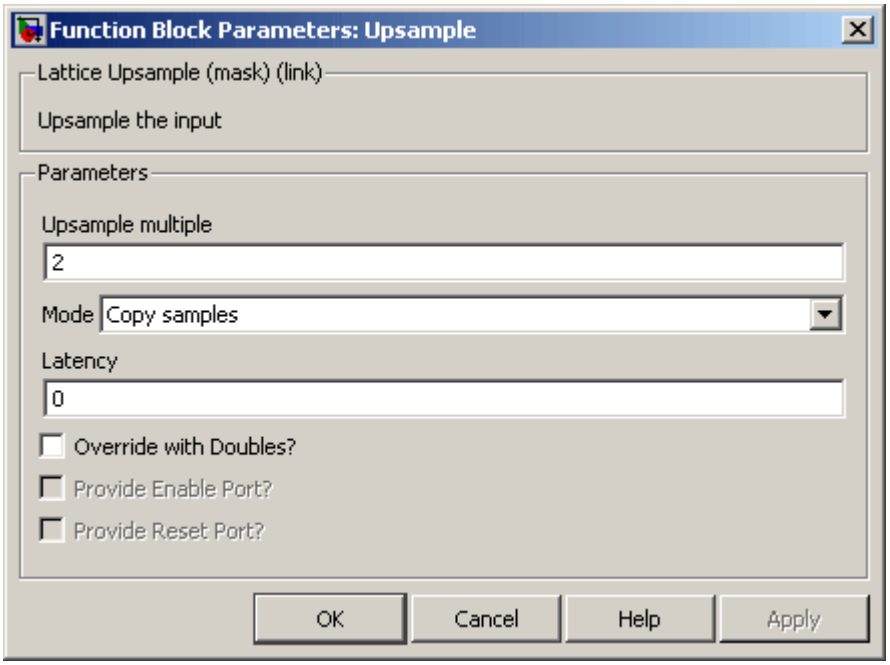
The block parameters are explained in Common Parameters for ispLeverDSP Blocks.

Upsample



The Upsample block creates an output that is a multiple of the input sample rate.

The user configurable parameters for the Upsample block are available in the following dialog box.



The block parameters are:

- **Upsample multiple**:

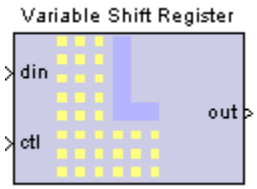
  This value is an integer that is the output rate divided by the input rate. Upsample maximum is 64.

- **Mode = [Copy samples, Insert zeroes]**:

  If “Copy samples” is selected, then the input is replicated for the inserted samples. If “Insert zeroes” is selected, then the input is copied for one cycle, and zeroes are output for all remaining cycles.

Other parameters used by this block are explained in Common Parameters for ispLeverDSP Blocks.

Variable Shift Register

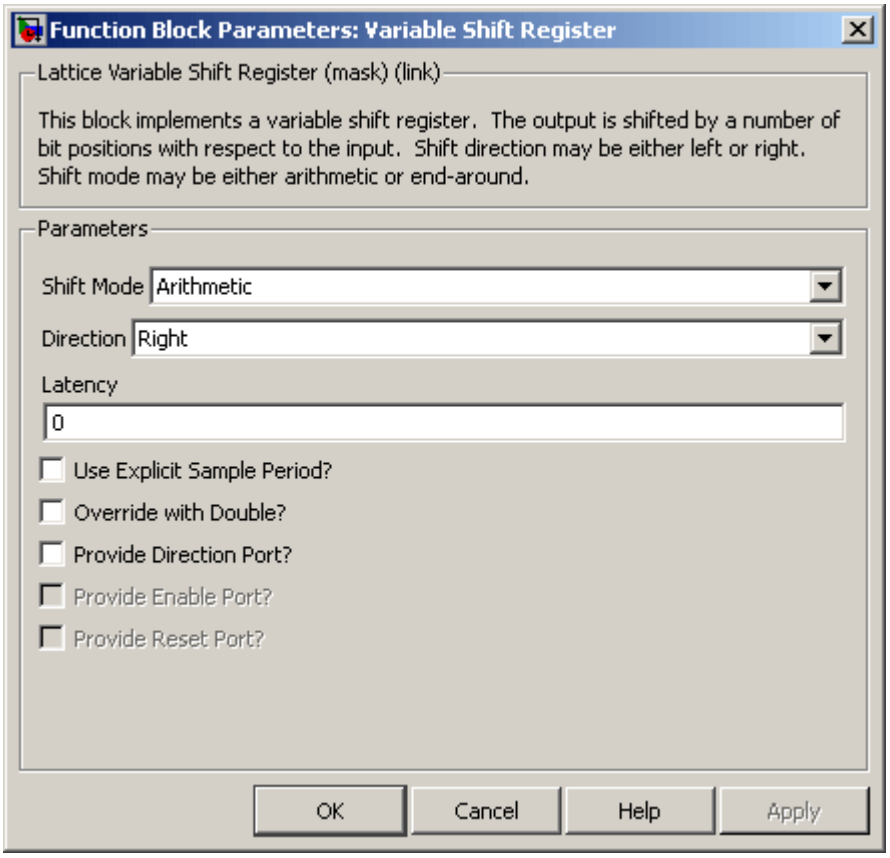


A minimum of two inputs must be present: a data input and a control input. The data input is a binary word having 1 to 52 bits and the control input is a 1- to 6-bit value specifying the number of bit positions the input value is shifted before assertion to the output. An optional input port for direction allows the choice between left and right shifting to be made dynamically. If the Direction port is defined, a ‘0’ denotes left shifting and ‘1’ denotes right shifting. The output word has the same bit width as the input word. Other optional inputs include enable and reset—these are selected via mask parameters.

Specific shift register parameters include: direction (this is used if the Direction port is not defined and hidden if the Direction port is defined), Mode (chooses arithmetic or end-around shifting), and the bit widths of input data and offset input ports.

Parameters in this block that are in common with other ispLeverDSP blocks are Arithmetic Format, Override to Double, Latency, and Binary Point.

The user configurable parameters for the Variable Shift Register Block are available in the following dialog box.



Block parameters are:

- **Mode**:

  Arithmetic, or End-Around modes are choices for this parameter. Left or Right shifting may be selected for either mode via the Direction Port (if defined) or the direction parameter.

  - **Arithmetic Mode**:

    If Right shifting is selected, the most significant bits of the result assume the value of the sign (msb) bit of the input value. If Left shifting is selected, lsbs of the shifted result are filled with zeros.

  - **End-Around Mode**:

    Also called barrel shifting, bits shifted out of least or bit significant bit positions are shifted back into the most or least significant, respectively, bit positions. In right shifting bits shifted out of the input word in the lsbs appear at the msbs of the result; in Left shifting bits shifted off the msbs are shifted back into the result at the lsbs.

- **Direction**:

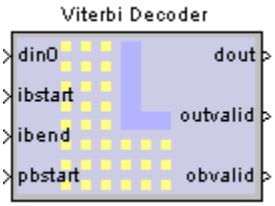
This parameter controls whether shifting is Left or Right.

- **Direction Port?**:

  If the box is checked, the Direction Port is defined, and the Direction Parameter (described above) is hidden from the user. The Direction Port, when defined, is a single-bit input: a state of zero (‘0’) commands Left shifting, and a state of one (‘1’) denotes Right shifting.

Other parameters used by this block are explained in Common Parameters for ispLeverDSP Blocks.

Viterbi Decoder



The Viterbi Decoder IP core is a parameterizable core for decoding different combinations of convolutionally encoded sequences. The decoder core supports various code rates, constraint lengths and generator polynomials. It also allows soft-decision decoding and is capable of decoding punctured codes. The core can operate in continuous or block mode, whichever is required by the channel. Either tail-biting or zero-flushing convolutional codes can be decoded in the block mode. All the configurable parameters, including operation mode, generator polynomials, puncturing block size and puncturing pattern can be defined by the user to suit the needs of their application. The architectural details of the core are given in the next section. Lattice’s Block Viterbi Decoder IP is compatible with many networking and wireless standards that use convolutional encoding at the encoder and Viterbi decoding at the decoder. It has the following features:

- Compatible with the following standards: IEEE 802.16-2004 SC PHY/ OFDM PHY, IEEEE802.11a, 3GPP, 3GPP2, and DVB-S
- Supports multiple code rates: 1/2, 1/3, ..., 1/7 for non-punctured codes, 2/ 3, 3/4, ..., 12/13 for punctured codes, and from m/(m+1) to m/(2m-1), where m is from 1 to 12, for dynamic punctured codes
- Variable constraint length from 3 to 9
- Supports dynamically variable code rates and puncture patterns
- Dynamic BER estimation option
- One-clock synchronous design
- Hard or parameterizable soft decision decoding. Hard and soft decision for non-punctured codes and soft decision for punctured codes
- Fully parallel or hybrid implementations. For a hybrid implementation, the degree of parallelism is parameterizable
- Parameterizable trace-back length
- Signed and unsigned representations for soft decision data
- Supports parameterized puncturing patterns
- Supports both continuous and block data input
- Supports both Tail Biting and Zero Flushing block convolutional codes

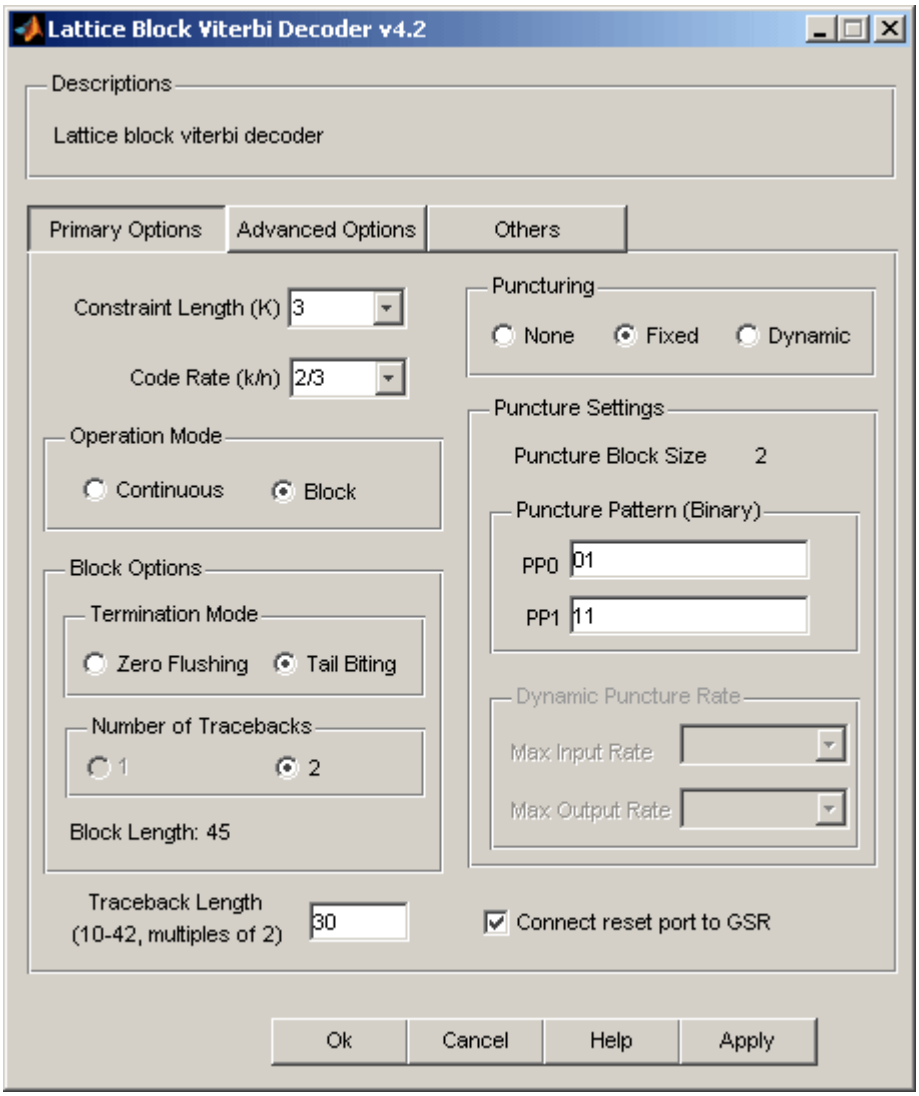
- Supports both one and two traceback schemes to cater to different coding scenarios

Note

This IP core is available for evaluation in the MATLAB/Simulink environment. A netlist will be generated for simulation purposes. An IP license must be purchased to implement the IP into a Lattice FPGA. Information on all Lattice IP Modules, including data sheets, brochures, and downloads, can be found on the Lattice Semiconductor Corporation web site at the following URL:

http://www.latticesemi.com/products/intellectualproperty/index.cfm

The user configurable parameters for Viterbi Decoder are available in the following dialog box.



Primary Options:

- **Constraint Length**

Constraint length is equal to the number of input data values (present and past) used to generate the convolutional code in the encoder. The value can be any integer from 3 to 9.

- **Code Rate**

  This is the symbol output rate of the encoder, defined as the number of output bits per input bit in the encoder. For non-punctured decoder, this can be set from 1/2 to 1/7. For fixed puncturing decoder, this can be set to *m*/*m*+1, where *m* can range from 2 to 12.

- **Operation Mode**

  The operation mode of the decoder is either continuous or block.

- **Traceback Length**

  Traceback length is the number of trellis states the decoder traces back for performing decoding. The traceback length must be between 3k to 14k. The range is further restricted by the value of some block related parameters.

- **Connect reset port to GSR**

  If this option is checked, the GSR is instantiated and used to route the Viterbi decoder's rstn input. Using GSR improves the utilization and performance of the Viterbi decoder. However, if GSR is used an active input in rstn will reset most of the FPGA components as well. This option must be checked to enable the hardware evaluation capability for this IP.

Block Options:

- **Termination Mode**

  This is the termination mode used for the convolutional coding of the input block. This parameter is required for block operation modes.

- **Number of Tracebacks**

  Number of tracebacks performed for decoding. This option is available only when zero-flushing termination mode is used.

Puncture Settings:

- **Puncturing**

  This option specifies whether input data is punctured or not. If the input is punctured, the decoder can be set to use either fixed puncture settings or dynamically variable puncture settings.

- **Puncture Pattern**

  Puncture pattern for fixed puncturing decoders. For dynamic puncture decoders, this pattern is applied through the input port. PP0 and PP1 are each k bits wide binary patterns.

- **Max Input Rate**

  This is the maximum value for the numerator, k, of the code rate, when the puncture rate is dynamically set through port.

- **Max Output Rate**

This is the maximum value for the denominator, n, of the code rate, when the puncture rate is dynamically set through port.

Generator Polynomials:

- **Radix**

  The number system in which the generator polynomials are specified.

- **GP0-GP6**

  Generator polynomials used for generating the convolutional code. The number of these polynomials is equal to 2 for punctured decoders and n for non-punctured decoders. The width of each polynomial is equal to constraint length, K.

Implementation:

- **Implementation Method**

  The implementation method can be either “parallel” or “hybrid.” In the parallel implementation, the decoder can produce one output data in one cycle. In hybrid implementations, it takes multiple clock cycles to generate each output data, but a smaller number of device resources are used.

- **Hybrid Index**

  This controls the resource-throughput trade-off in hybrid implementations. It takes 2^(Hybrid Index) cycles to produce one output data. For example, if Hybrid Index is 4, It takes 16 (2^4) cycles to produce one output data.

Inputs:

- **Decoder Input**

  Specifies whether the decoder is fed with a hard decision or soft decision input. For punctured decoders, this option is not available and decoder has to be fed with soft decision inputs.

- **Soft Width**

  Input data width for soft decision inputs.

- **Data Type**

  Specifies whether the input data type is represented in sign-magnitude form (signed) or unsigned offset form (unsigned).

BER (Bit Error Rate):

- **BER Monitor**

  Specifies whether the optional bit error rate (BER) monitor is added to the Viterbi decoder.

- **BER Period**

  This determines the duration for which the BER is accumulated. The BER value starts accumulating from zero for up to 2^(BER Period) clock cycles. After this period, the accumulated value is placed on the BER output port. The BER value is then reset and the monitor starts accumulating again.

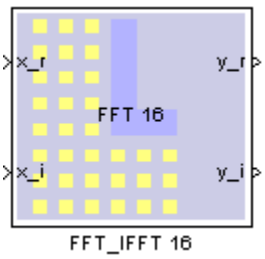
List of Reference Designs

Lattice ispLeverDSP Reference designs are fully parameterizable DSP reference designs for MATLAB/Simulink. The designs were developed using ispLeverDSP design flow and are optimized for high performance in Lattice devices. The following table describes the Lattice Reference Designs currently available for MATLAB/Simulink,

Table 12: Lattice Reference Blockset

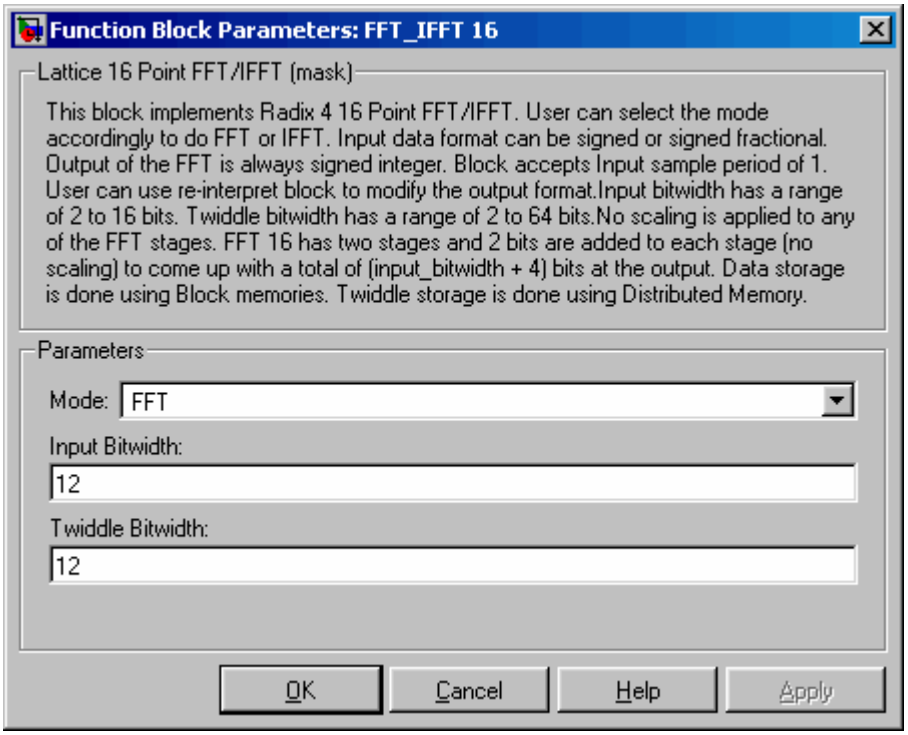
Block Name	Brief Description
16 Point FFT/IFFT	This block implements Radix 4 16 Point FFT/IFFT.
One Multiplier Based MAC FIR	This is a 1 Multiplier based MAC FIR filter implementation.
Two Multiplier Based MAC FIR	This is a two multiplier based MAC FIR implementation that effectively doubles the throughput of the FIR filter compared to single multiplier based implementation.
Four Multiplier Based MAC FIR	This is a four multiplier based MAC FIR implementation that effectively quadruples the throughput of the FIR filter compared to single multiplier based implementation.
One Multiplier Based MAC FIR for Even Coefficient Symmetry	This is a single multiplier based MAC FIR implementation that uses the coefficient symmetry property of the FIR filter for even number of taps to increase throughput and save on resource utilization.
8 Bit, 8 Tap Distributed Arithmetic FIR	This is 8 Bit, 8 Tap Distributed Arithmetic FIR Filter
8 Bit, 16 Tap Distributed Arithmetic FI	This is 8 Bit, 16 Tap Distributed Arithmetic FIR Filter
8 Bit, 32 Tap Distributed Arithmetic FIR	This is 8 Bit, 32 Tap Distributed Arithmetic FIR Filter
8 Bit, 64 Tap Distributed Arithmetic FIR	This is 8 Bit, 64 Tap Distributed Arithmetic FIR Filter
8 Bit, 128 Tap Distributed Arithmetic FIR	This is 8 Bit, 128 Tap Distributed Arithmetic FIR Filter
8-Channel 16-Tap 16-Multiplier Transpose FIR	The design implements an eight-channel, 16-tap and 16-multiplier transpose FIR filter.
Matrix 8x8 Matrix Multiplier	This design implements a 8x8 Matrix Multiplier.
Multi-Channel 1-Multiplier Based MAC FIR	This is a multi-channel single-multiplier based MAC FIR filter.

16 Point FFT/IFFT



This block implements Radix 4 16 Point FFT/IFFT. User can select the mode accordingly to do FFT or IFFT. Input data format can be signed or signed fractional. Output of the FFT is always signed integer. Block accepts Input sample period of 1. User can use re-interpret block to modify the output format. Input bit width has a range of 2 to 16 bits. Twiddle bit width has a range of 2 to 64 bits. No scaling is applied to any of the FFT stages. FFT 16 has two stages and 2 bits are added to each stage (no scaling) to come up with a total of (input_bitwidth + 4) bits at the output. Data storage is done using Block memories. Twiddle storage is done using Distributed Memory.

The user configurable parameters for the FFT_IFFT 16 are available in the following dialog box.



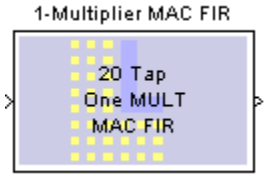
The block parameters are:

- **Mode = [FFT, IFFT]**
- **Input Bitwidth**:

  Input bitwidth has a range of 2 to 16 bits.
- **Twiddle Bitwidth**:

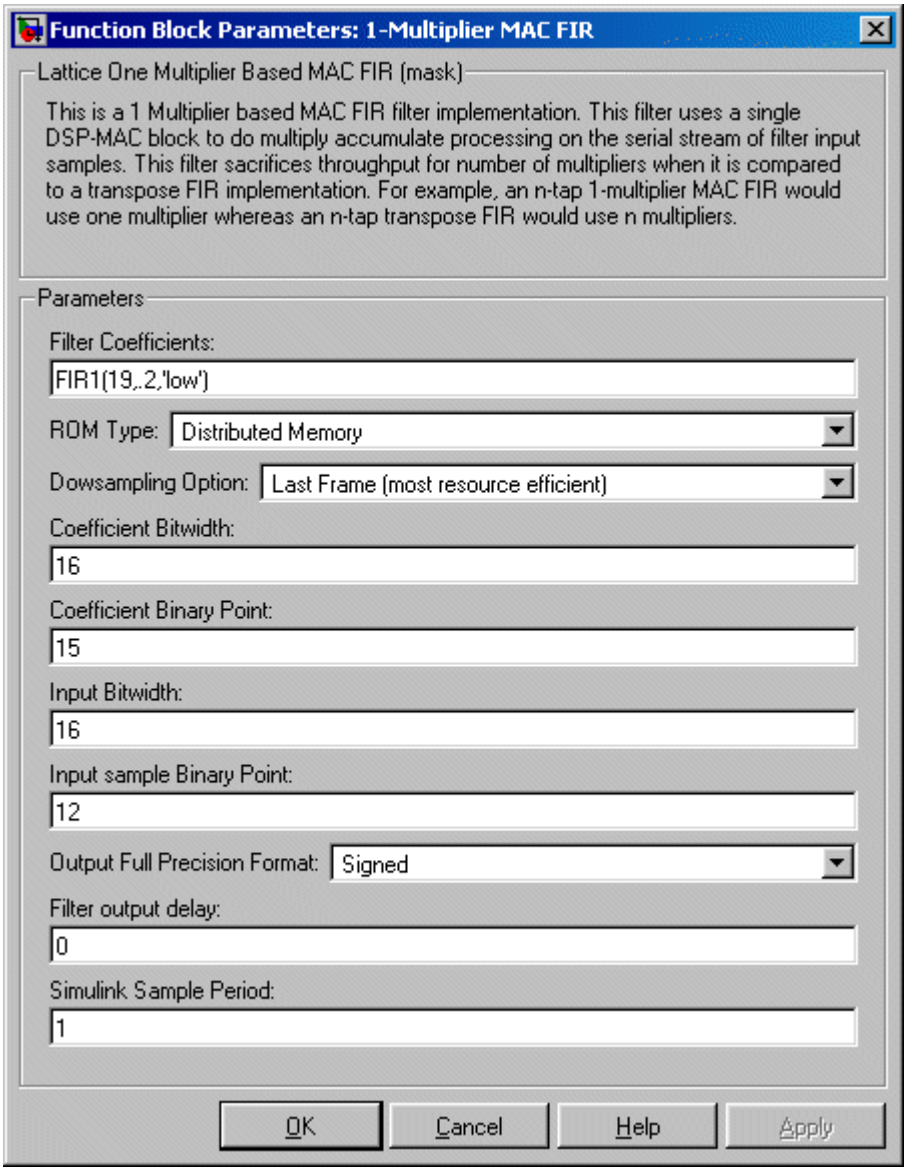
  Twiddle bitwidth has a range of 2 to 64 bits.

One Multiplier Based MAC FIR



This is a 1 Multiplier based MAC FIR filter implementation. This filter uses a single DSP-MAC block to do multiply accumulate processing on the serial stream of filter input samples. The MAC FIR filter implementation sacrifices throughput for number of multipliers when compared to a transpose FIR implementation. For example, an n-tap 1-Multiplier MAC FIR would use one multiplier whereas a n-tap transpose FIR would use n multipliers. This filter implementation uses one ROM for storing the filter coefficients.

The user configurable parameters for the 1 Multiplier based MAC FIR are available in the following dialog box.



The block parameters are:

- **Filter Coefficients**:

  Accepts MATLAB expression of any type. By default, the coefficients are produced by the MATLAB function FIR1. If you do not have the MATLAB Signal Processing Toolbox, you must enter your own coefficients. The length of the coefficient vector determines the number of filter taps.

- **ROM Type = [Clock Memory, Distributed Memory]**
- **Downsampling Option**:

  Select between Last Frame (resource efficient) or First Frame (speed efficient) modes for the down sampler used in the design.

- **Coefficient Bit width**:

  Choose per filter design specification.

- **Coefficient Binary Point**:

  Choose per filter design specification.

- **Input Bitwidth**:

  Choose per filter design specification.

- **Input Sample Binary Point**:

  Choose per filter design specification.

- **Output Full Precision Format**:

  The filter has full precision output with selectable Signed or Unsigned data type format.

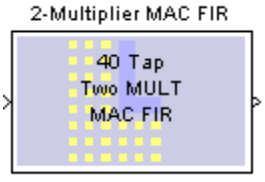
- **Filter Output Delay**:

  The filter output delay is the delay you can set to align the filter output with an external circuit. This delay is additional to inherent filter delay that is defined by the implementation.

- **Simulink Sample Period**:

  The Simulink sample period is the input sample period for the FIR. The sampling period of the input source must match the value entered for the Simulink Sample Period.

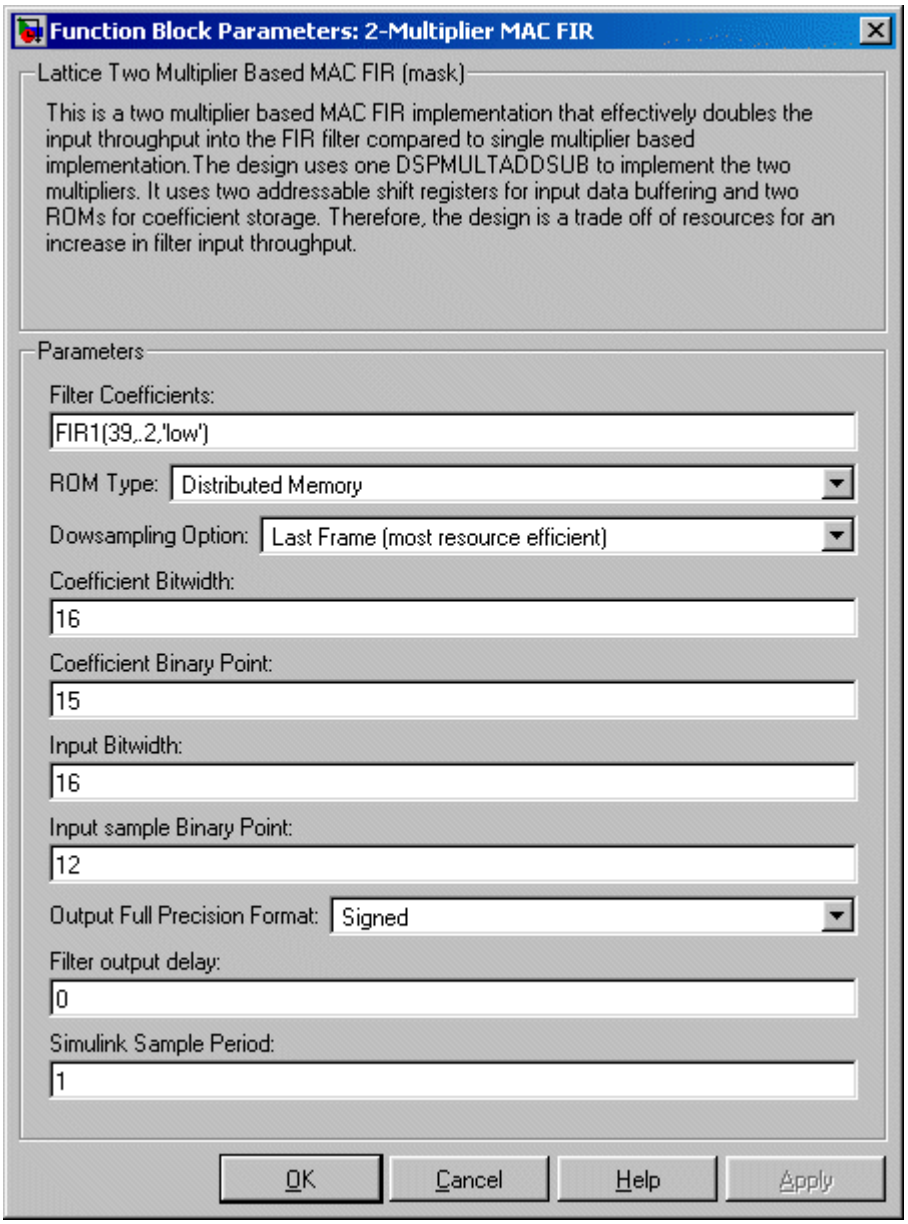
Two Multiplier Based MAC FIR



This is a two multiplier based MAC FIR implementation that effectively doubles the throughput of the FIR filter compared to single multiplier based implementation. The design uses one DSPMULTADDSUB to implement the two multipliers. It uses two addressable shift registers for input data buffering and two ROMs for coefficient storage. Therefore, the design is a trade off of resources for an increase in filter throughput.

The user configurable parameters for the Lattice Two multiplier based MAC FIR are available in the following dialog box.

.



The block parameters are:

- **Filter Coefficients**:

  Accepts MATLAB expression of any type. By default, the coefficients are produced by the MATLAB function FIR1. If you do not have the MATLAB Signal Processing Toolbox, you must enter your own coefficients. The length of the coefficient vector determines the number of filter taps.

- **ROM Type = [Block Memory, Distributed Memory]**
- **Downsampling Option**:

  Select between Last Frame (resource efficient) or First Frame (speed efficient) modes for the down sampler used in the design.

- **Coefficient Bit width**:

  Choose per filter design specification.

- **Coefficient Binary Point**:

  Choose per filter design specification.

- **Input Bitwidth**:

  Choose per filter design specification.

- **Input Sample Binary Point**:

  Choose per filter design specification.

- **Output Full Precision Format**:

  The filter has full precision output with selectable Signed or Unsigned data type format.

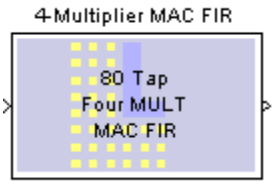
- **Filter Output Delay**:

  The filter output delay is the delay you can set to align the filter output with an external circuit. This delay is additional to inherent filter delay that is defined by the implementation.

- **Simulink Sample Period**:

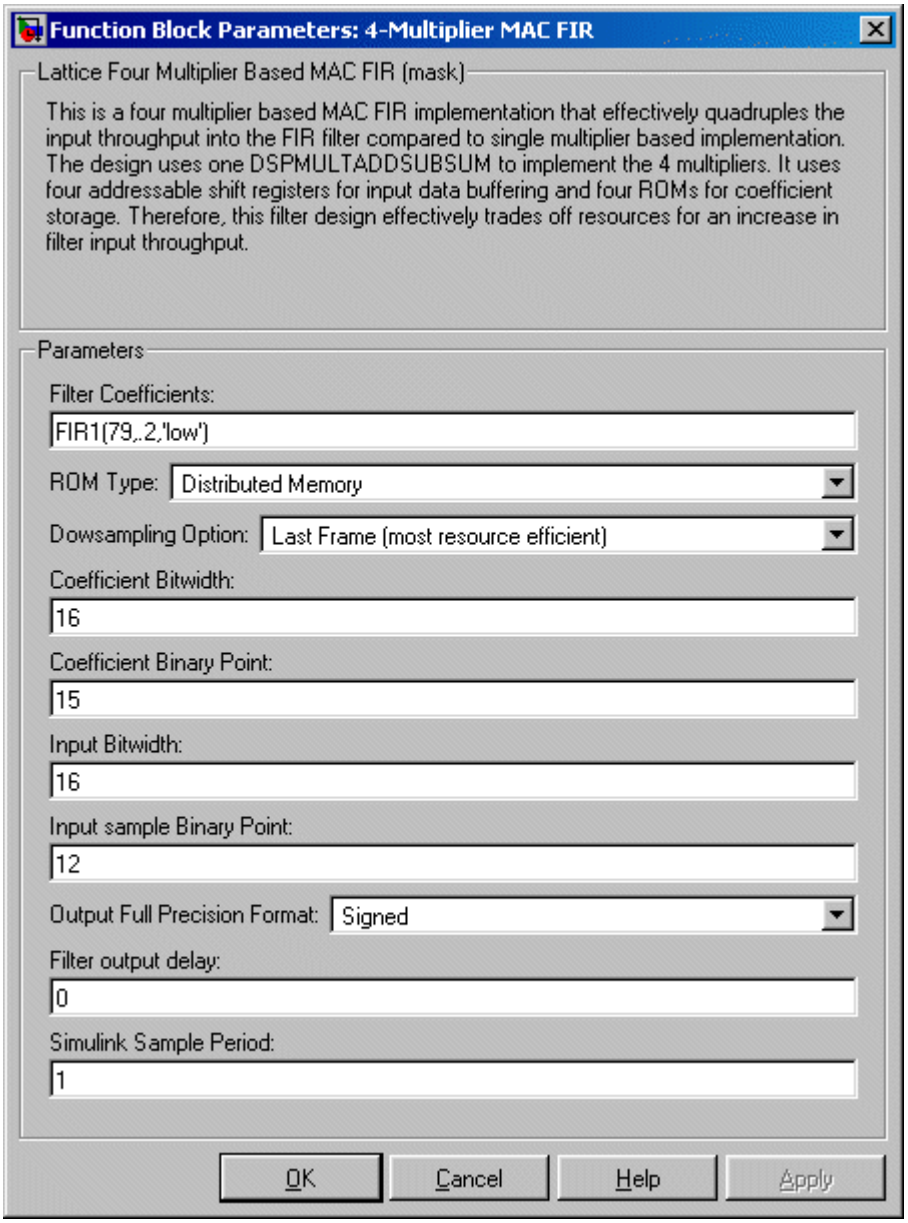
  The Simulink sample period is the input sample period for the FIR. The sampling period of the input source must match the value entered for the Simulink Sample Period.

Four Multiplier Based MAC FIR



This is a four multiplier based MAC FIR implementation that effectively quadruples the throughput of the FIR filter compared to single multiplier based implementation. The design uses one DSPMULTADDSUBSUM to implement the 4 multipliers. It uses four addressable shift registers for input data buffering and four ROMs for coefficient storage. Therefore, this filter design effectively trades off resources for a higher filter throughput.

The user configurable parameters for the Four Multiplier Based MAC FIR are available in the following dialog box.



The block parameters are:

- **Filter Coefficients**:

  Accepts MATLAB expression of any type. By default, the coefficients are produced by the MATLAB function FIR1. If you do not have the MATLAB Signal Processing Toolbox, you must enter your own coefficients. The length of the coefficient vector determines the number of filter taps.

- **ROM Type = [Block Memory, Distributed Memory]**
- **Downsampling Option**:

  Select between Last Frame (resource efficient) or First Frame (speed efficient) modes for the down sampler used in the design.

- **Coefficient Bit width**:

Choose per filter design specification.

- **Coefficient Binary Point**:

  Choose per filter design specification.

- **Input Bitwidth**:

  Choose per filter design specification.

- **Input Sample Binary Point**:

  Choose per filter design specification.

- **Output Full Precision Format**:

  The filter has full precision output with selectable Signed or Unsigned data type format.

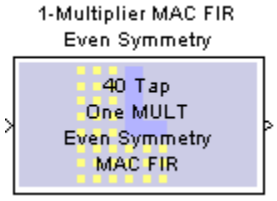
- **Filter Output Delay**:

  The filter output delay is the delay you can set to align the filter output with an external circuit. This delay is additional to inherent filter delay that is defined by the implementation.

- **Simulink Sample Period**:

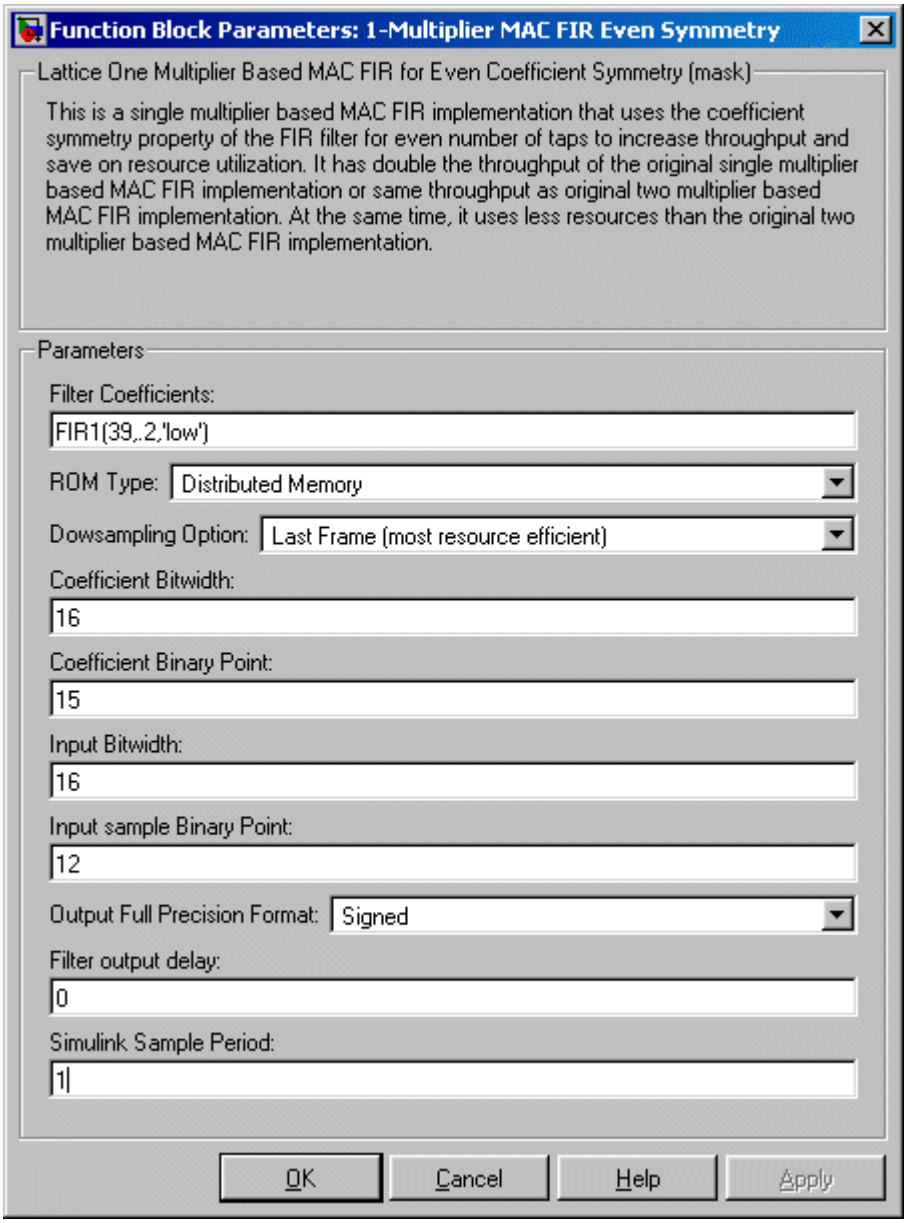
  The Simulink sample period is the input sample period for the FIR. The sampling period of the input source must match the value entered for the Simulink Sample Period.

One Multiplier Based MAC FIR for Even Coefficient Symmetry



This is a single multiplier based MAC FIR implementation that uses the coefficient symmetry property of the FIR filter for even number of taps to increase throughput and save on resource utilization. It has double the throughput of the original single multiplier based MAC FIR implementation and same throughput as original two multiplier based MAC FIR implementation. But, uses much less resources than the original two multiplier based MAC FIR implementation. To be specific, it uses one ROM and one Multiplier where as two multiplier implementation uses two ROMs and two Multipliers.

The user configurable parameters for the Lattice One Multiplier Based MAC FIR for Even Coefficient Symmetry are available in the following dialog box.



The block parameters are:

- **Filter Coefficients**:

  Accepts MATLAB expression of any type. By default, the coefficients are produced by the MATLAB function FIR1. If you do not have the MATLAB Signal Processing Toolbox, you must enter your own coefficients. The length of the coefficient vector determines the number of filter taps.

- **ROM Type = [Block Memory, Distributed Memory]**
- **Downsampling Option**:

  Select between Last Frame (resource efficient) or First Frame (speed efficient) modes for the down sampler used in the design.

- **Coefficient Bit width**:

Choose per filter design specification.

- **Coefficient Binary Point**:

  Choose per filter design specification.

- **Input Bitwidth**:

  Choose per filter design specification.

- **Input Sample Binary Point**:

  Choose per filter design specification.

- **Output Full Precision Format**:

  The filter has full precision output with selectable Signed or Unsigned data type format.

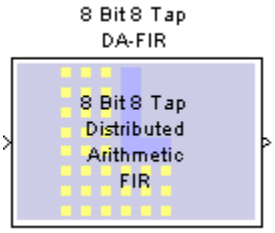
- **Filter Output Delay**:

  The filter output delay is the delay you can set to align the filter output with an external circuit. This delay is additional to inherent filter delay that is defined by the implementation.

- **Simulink Sample Period**:

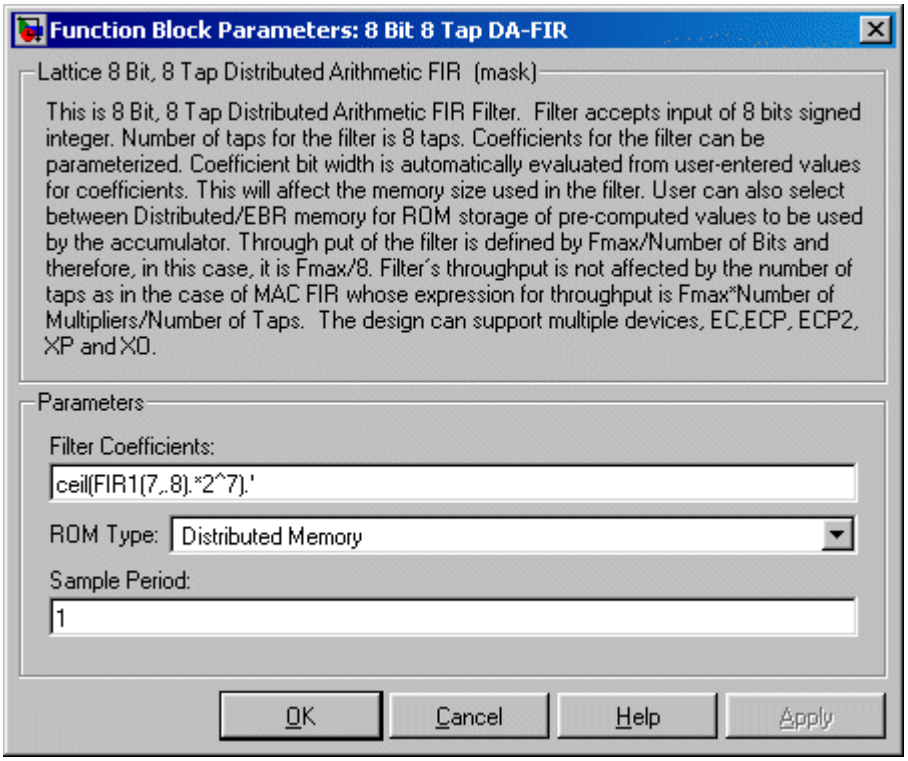
  The Simulink sample period is the input sample period for the FIR. The sampling period of the input source must match the value entered for the Simulink Sample Period.

8 Bit, 8 Tap Distributed Arithmetic FIR



This is 8 Bit, 8 Tap Distributed Arithmetic FIR Filter. Filter accepts input of 8 bits signed integer. Filter has fixed number of taps of 8. Coefficient values for the filter can be parameterized and has to be in signed integer format. Coefficient bit width is automatically evaluated from user-entered values for coefficients. This will affect the memory size used in the filter. User can also select between Distributed/EBR memory for ROM storage of pre-computed values to be used by the accumulator. Throughput of DA FIR is defined by Fmax/Number of Bits and therefore, in this case, it is Fmax/8. Filter's throughput is not affected by the number of taps as in the case of MAC FIR whose expression for throughput is Fmax*Number of Multipliers/Number of Taps. The design can support multiple devices, EC,ECP, ECP2, XP and XO.

The user configurable parameters for the Lattice 8 Bit 8 Tap DA-FIR are available in the following dialog box.



The block parameters are:

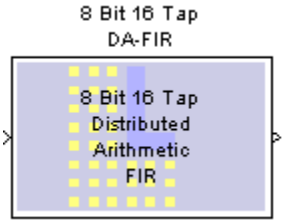
- **Filter Coefficients**:

  Accepts MATLAB expression of any type. By default, the coefficients are produced by the MATLAB function FIR1. If you do not have the MATLAB Signal Processing Toolbox, you must enter your own coefficients. The length of the coefficient vector determines the number of filter taps.

- **ROM Type = [Block Memory, Distributed Memory]**
- **Sample Period**:

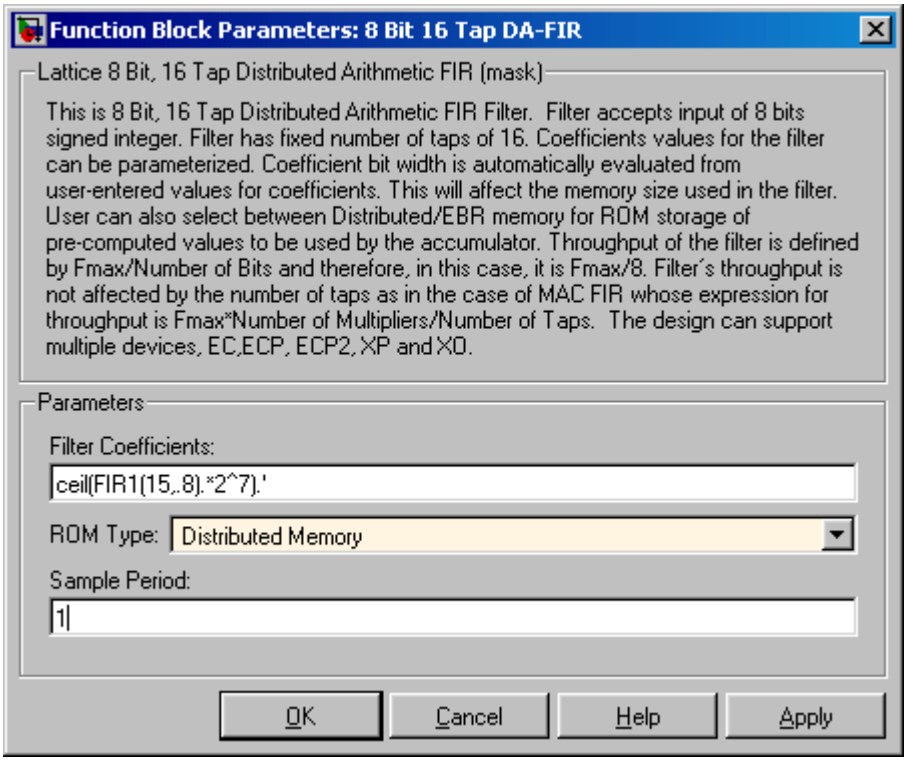
  Choose per filter design specification.

8 Bit, 16 Tap Distributed Arithmetic FI



This is 8 Bit, 16 Tap Distributed Arithmetic FIR Filter. Filter accepts input of 8 bits signed integer. Filter has fixed number of taps of 16. Coefficient values for the filter can be parameterized and has to be in signed integer format. Coefficient bit width is automatically evaluated from user-entered values for coefficients. This will affect the memory size used in the filter. User can also select between Distributed/EBR memory for ROM storage of pre-computed values to be used by the accumulator. Throughput of DA FIR is defined by Fmax/Number of Bits and therefore, in this case, it is Fmax/8. Filter's throughput is not affected by the number of taps as in the case of MAC FIR whose expression for throughput is Fmax*Number of Multipliers/Number of Taps. The design can support multiple devices, EC,ECP, ECP2, XP and XO.

The user configurable parameters for the Lattice 8 Bit 16 Tap DA-FIR are available in the following dialog box.



The block parameters are:

- **Filter Coefficients**:

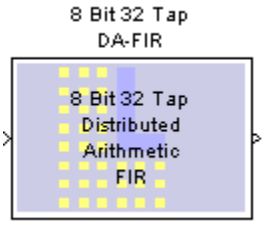
  Accepts MATLAB expression of any type. By default, the coefficients are produced by the MATLAB function FIR1. If you do not have the MATLAB Signal Processing Toolbox, you must enter your own coefficients. The length of the coefficient vector determines the number of filter taps.

- **ROM Type = [Block Memory, Distributed Memory]**

- **Sample Period**:

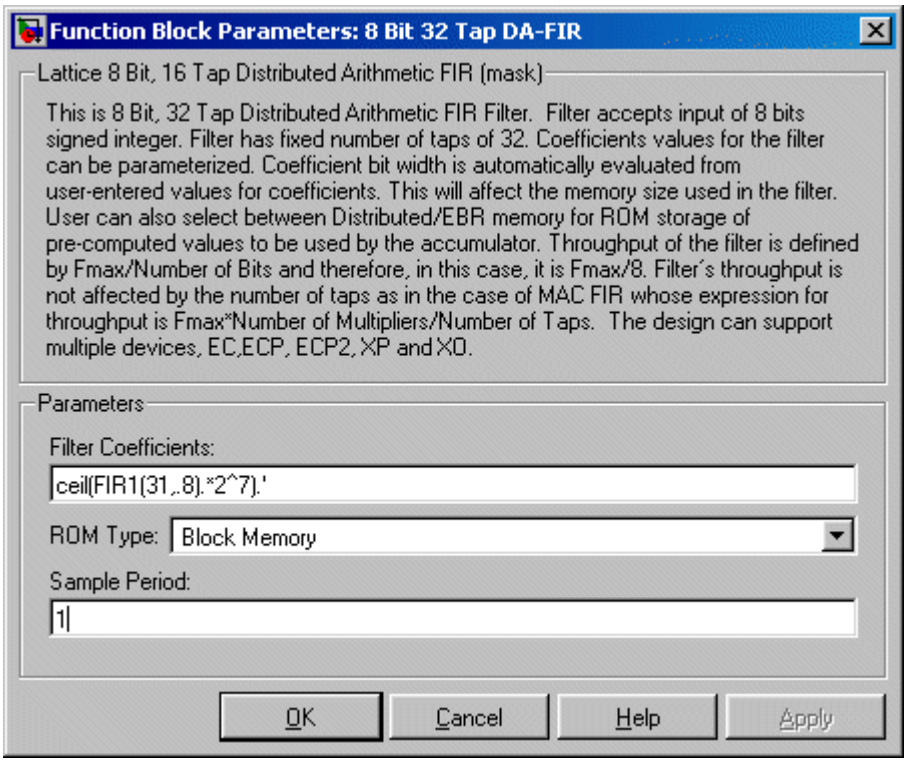
  Choose per filter design specification.

8 Bit, 32 Tap Distributed Arithmetic FIR



This is 8 Bit, 32 Tap Distributed Arithmetic FIR Filter. Filter accepts input of 8 bits signed integer. Filter has fixed number of taps of 32. Coefficient values for the filter can be parameterized and has to be in signed integer format. Coefficient bit width is automatically evaluated from user-entered values for coefficients. This will affect the memory size used in the filter. User can also select between Distributed/EBR memory for ROM storage of pre-computed values to be used by the accumulator. Throughput of DA FIR is defined by Fmax/ Number of Bits and therefore, in this case, it is Fmax/8. Filter's throughput is not affected by the number of taps as in the case of MAC FIR whose expression for throughput is Fmax*Number of Multipliers/Number of Taps. The design can support multiple devices, EC,ECP, ECP2, XP and XO.

The user configurable parameters for the Lattice 8 Bit 32 Tap DA-FIR are available in the following dialog box.



The block parameters are:

- **Filter Coefficients**:

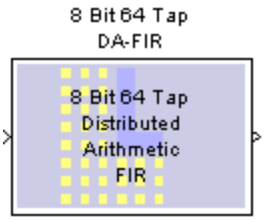
  Accepts MATLAB expression of any type. By default, the coefficients are produced by the MATLAB function FIR1. If you do not have the MATLAB

Signal Processing Toolbox, you must enter your own coefficients. The length of the coefficient vector determines the number of filter taps.

- **ROM Type = [Block Memory, Distributed Memory]**
- **Sample Period**:

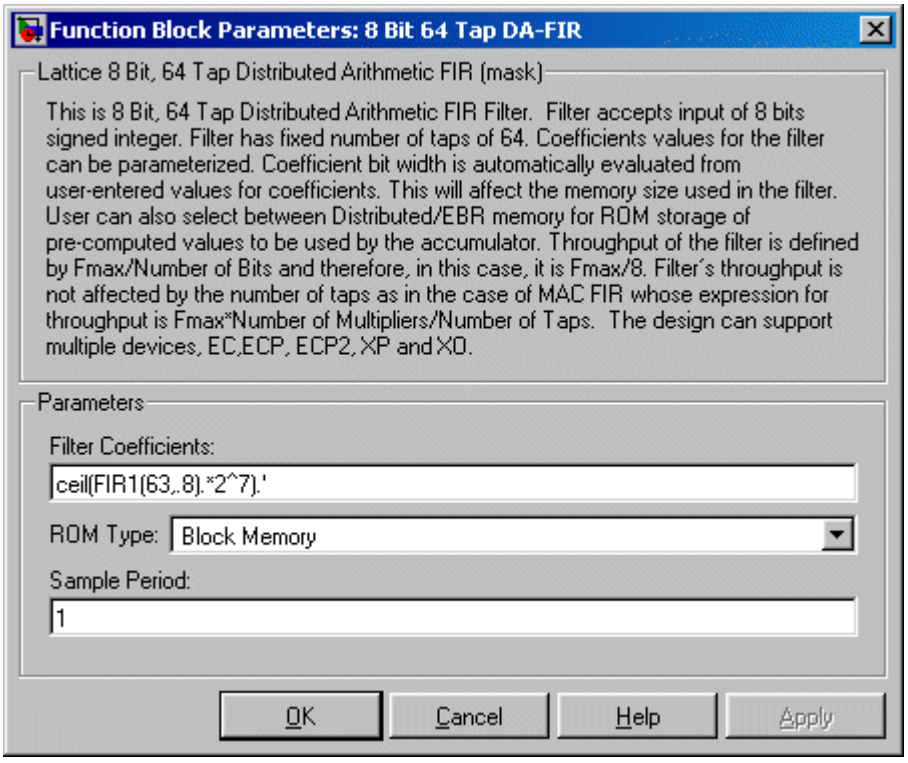
  Choose per filter design specification.

8 Bit, 64 Tap Distributed Arithmetic FIR



This is 8 Bit, 64 Tap Distributed Arithmetic FIR Filter. Filter accepts input of 8 bits signed integer. Filter has fixed number of taps of 64. Coefficient values for the filter can be parameterized and has to be in signed integer format. Coefficient bit width is automatically evaluated from user-entered values for coefficients. This will affect the memory size used in the filter. User can also select between Distributed/EBR memory for ROM storage of pre-computed values to be used by the accumulator. Throughput of DA FIR is defined by Fmax/ Number of Bits and therefore, in this case, it is Fmax/8. Filter's throughput is not affected by the number of taps as in the case of MAC FIR whose expression for throughput is Fmax*Number of Multipliers/Number of Taps. The design can support multiple devices, EC,ECP, ECP2, XP and XO.

The user configurable parameters for the Lattice 8 Bit 64 Tap DA-FIR are available in the following dialog box.



The block parameters are:

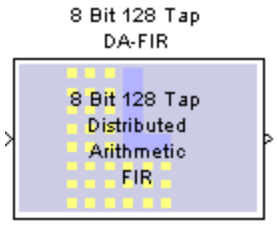
- **Filter Coefficients**:

  Accepts MATLAB expression of any type. By default, the coefficients are produced by the MATLAB function FIR1. If you do not have the MATLAB Signal Processing Toolbox, you must enter your own coefficients. The length of the coefficient vector determines the number of filter taps.

- **ROM Type = [Block Memory, Distributed Memory]**
- **Sample Period**:

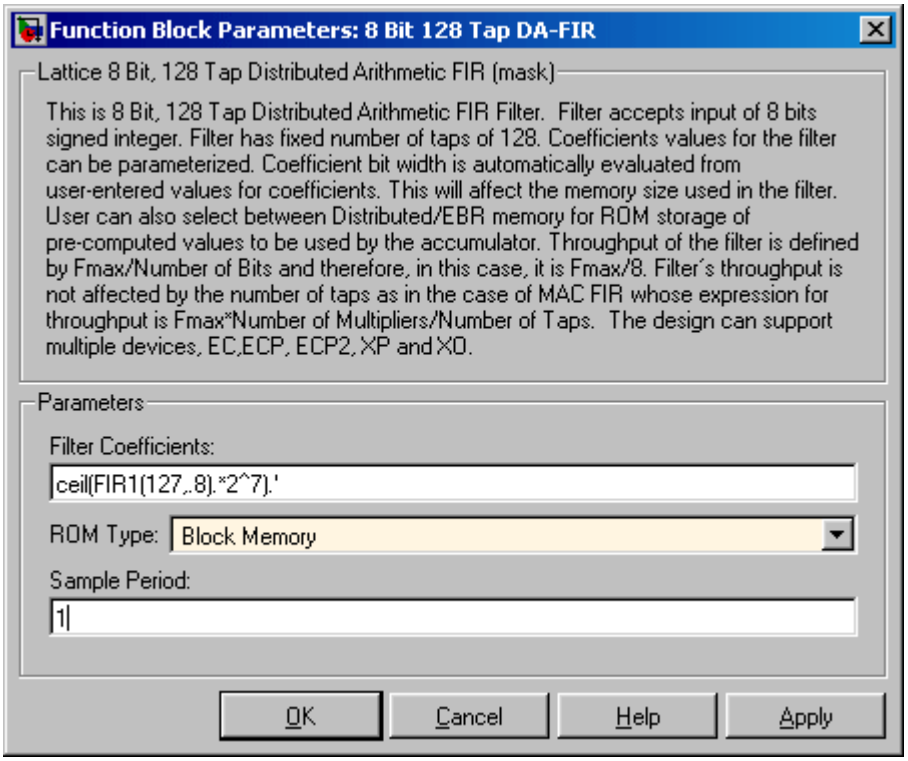
  Choose per filter design specification.

8 Bit, 128 Tap Distributed Arithmetic FIR



This is 8 Bit, 128 Tap Distributed Arithmetic FIR Filter. Filter accepts input of 8 bits signed integer. Filter has fixed number of taps of 128. Coefficient values for the filter can be parameterized and has to be in signed integer format. Coefficient bit width is automatically evaluated from user-entered values for coefficients. This will affect the memory size used in the filter. User can also select between Distributed/EBR memory for ROM storage of pre-computed values to be used by the accumulator. Throughput of DA FIR is defined by Fmax/Number of Bits and therefore, in this case, it is Fmax/8. Filter's throughput is not affected by the number of taps as in the case of MAC FIR whose expression for throughput is Fmax*Number of Multipliers/Number of Taps. The design can support multiple devices, EC,ECP, ECP2, XP and XO.

The user configurable parameters for the Lattice 8 Bit 128 Tap DA-FIR are available in the following dialog box.



The block parameters are:

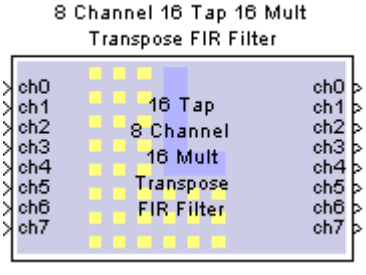
- **Filter Coefficients**:

  Accepts MATLAB expression of any type. By default, the coefficients are produced by the MATLAB function FIR1. If you do not have the MATLAB Signal Processing Toolbox, you must enter your own coefficients. The length of the coefficient vector determines the number of filter taps.

- **ROM Type = [Block Memory, Distributed Memory]**
- **Sample Period**:

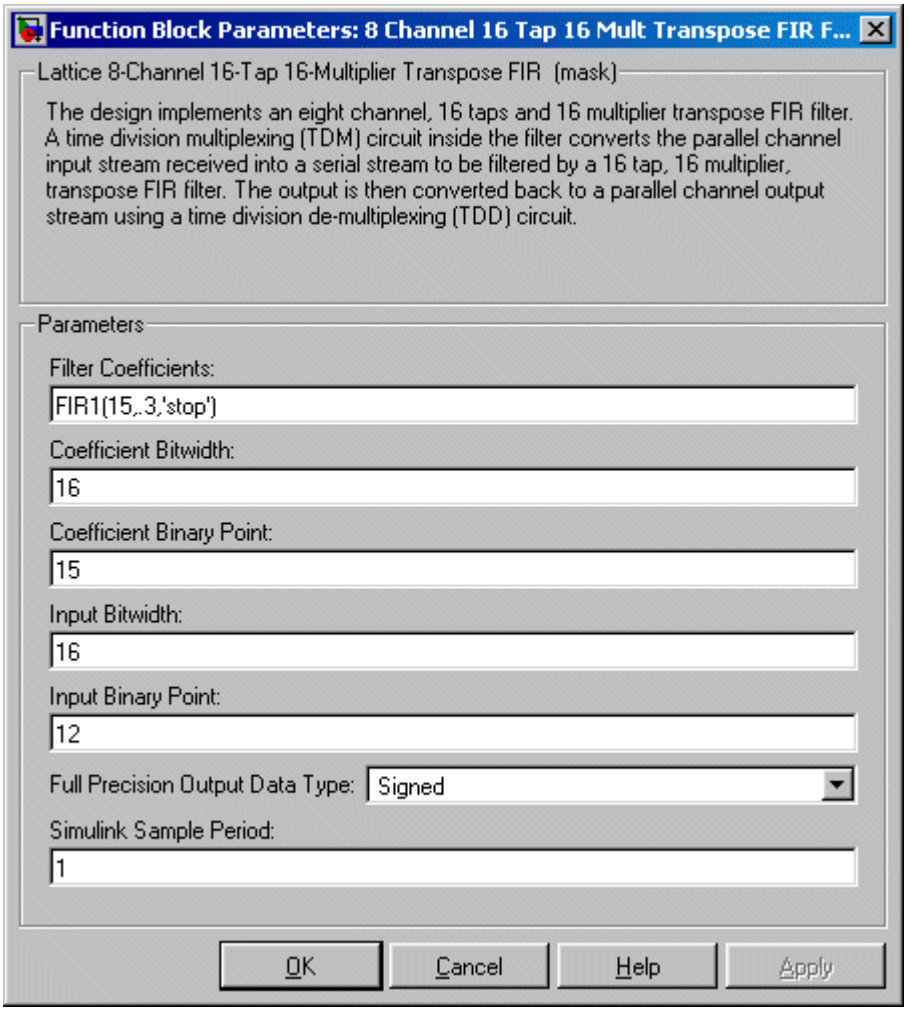
  Choose per filter design specification.

8-Channel 16-Tap 16-Multiplier Transpose FIR



The design implements an eight-channel, 16-tap and 16-multiplier transpose FIR filter. A Time Division Multiplexing (TDM) circuit inside the filter converts the parallel channel input data stream into a serial stream to be filtered by a 16-tap, 16-multiplier transpose FIR filter. The output is then converted back to a parallel channel output data stream using a time division de-multiplexing (TDD) circuit.

The user configurable parameters for the Lattice 8 Channel 16 Tap 16 Multiplier Transpose FIR are available in the following dialog box.



The block parameters are:

- **Filter Coefficients**:

  Accepts MATLAB expression of any type. By default, the coefficients are produced by the MATLAB function FIR1. If you do not have the MATLAB Signal Processing Toolbox, you must enter your own coefficients. The length of the coefficient vector determines the number of filter taps.

- **ROM Type = [Block Memory, Distributed Memory]**
- **Downsampling Option**:

  Select between Last Frame (resource efficient) or First Frame (speed efficient) modes for the down sampler used in the design.

- **Coefficient Bit width**:

  Choose per filter design specification.

- **Coefficient Binary Point**:

  Choose per filter design specification.

- **Input Bitwidth**:

Choose per filter design specification.

- **Input Sample Binary Point**:

  Choose per filter design specification.

- **Output Full Precision Format**:

  The filter has full precision output with selectable Signed or Unsigned data type format.

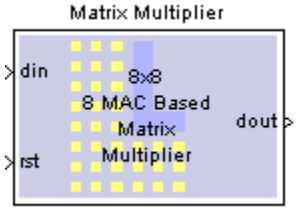
- **Filter Output Delay**:

  The filter output delay is the delay you can set to align the filter output with an external circuit. This delay is additional to inherent filter delay that is defined by the implementation.

- **Simulink Sample Period**:

  The Simulink sample period is the input sample period for the FIR. The sampling period of the input source must match the value entered for the Simulink Sample Period.

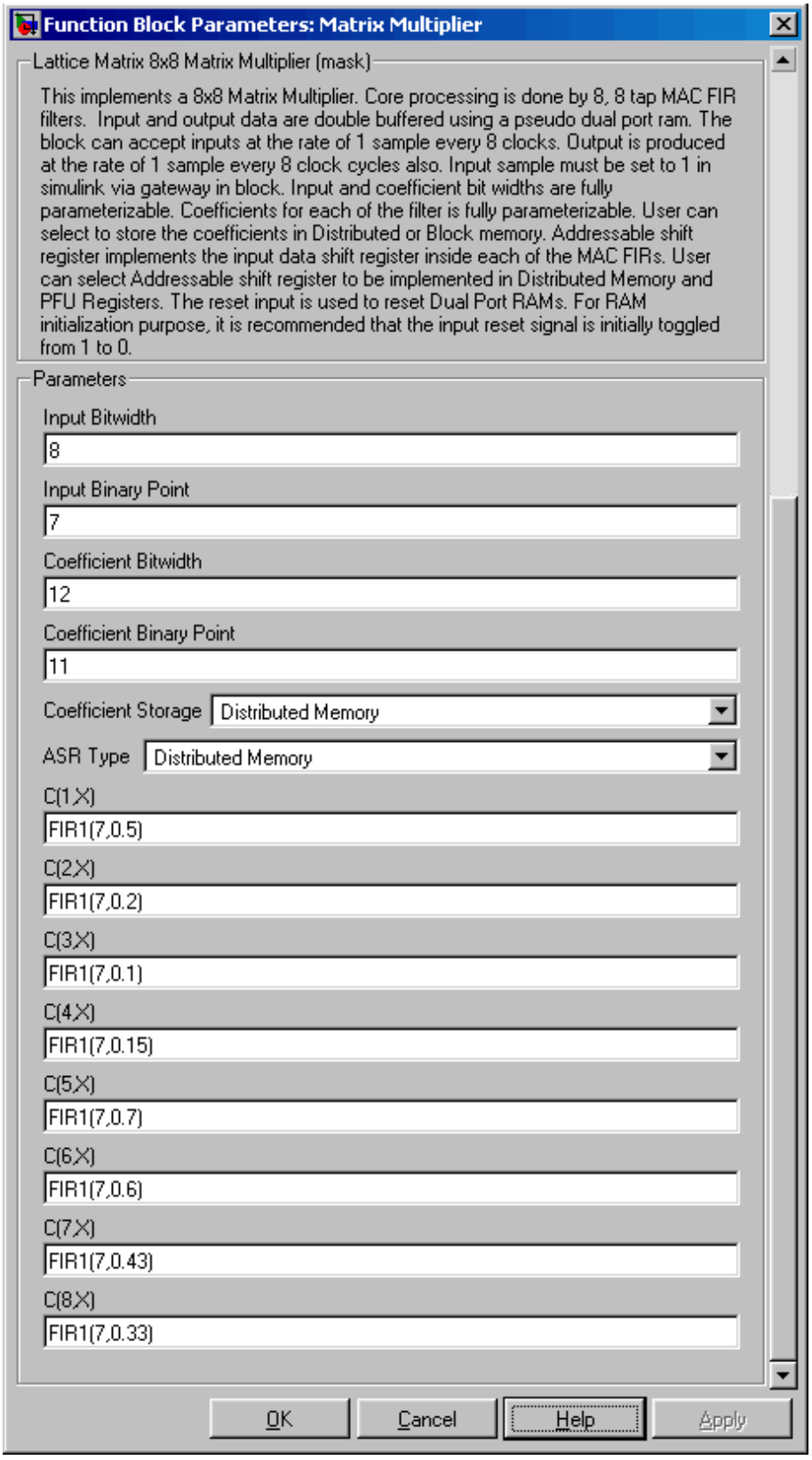
Matrix 8x8 Matrix Multiplier



This implements a 8x8 Matrix Multiplier. Core processing is done by 8, 8 tap MAC FIR filters. Input and output data are double buffered using a pseudo dual port ram. The block can accept input at the rate of 1 sample every 8 clocks. Output is produced at the rate of 1 sample every 8 clock cycles also. Input sample must be set to 1 in Simulink via gateway in block. Input and coefficient bit widths are fully parameterizable. Coefficients for each of the filter is fully parameterizable. User can select to store the coefficients in Distributed or Block memory.

Addressable shift register implements the input data shift register inside each of the MAC FIRs. User can select Addressable shift register to be implemented in Distributed Memory and PFU Registers. The reset input is used to reset Dual Port RAMs. For RAM initialization purpose, it is recommended that the input reset signal is initially toggled from 1 to 0.

The user configurable parameters for Lattice Matrix 8x8 Matrix Multiplier are available in the following dialog box.



The block parameters are:

- **Input Bitwidth**:

  Choose per filter design specification.

- **Input Binary Point**:

  Choose per filter design specification.

- **Coefficient Bitwidth**:

  Choose per filter design specification.

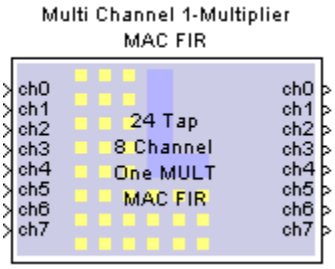
- **Coefficient Binary Point**:

  Choose per filter design specification.

- **Coefficient Storage = [Block Memory, Distributed Memory]**
- **ASR Type = [Distributed Memory, PFU Registers]**
- **C(1,X), C(2,X), C(3,X), C(4,X), C(5,X), C(6,X), C(7,X), C(8,X)**:

  These define coefficients of the Matrix Multiplier. Each dialog accept MATLAB expression of any type. Length of the resulting coefficient vector must be 8. By default, the coefficients are produced by the MATLAB function FIR1. If you do not have the MATLAB Signal Processing Toolbox, you must enter your own coefficients.

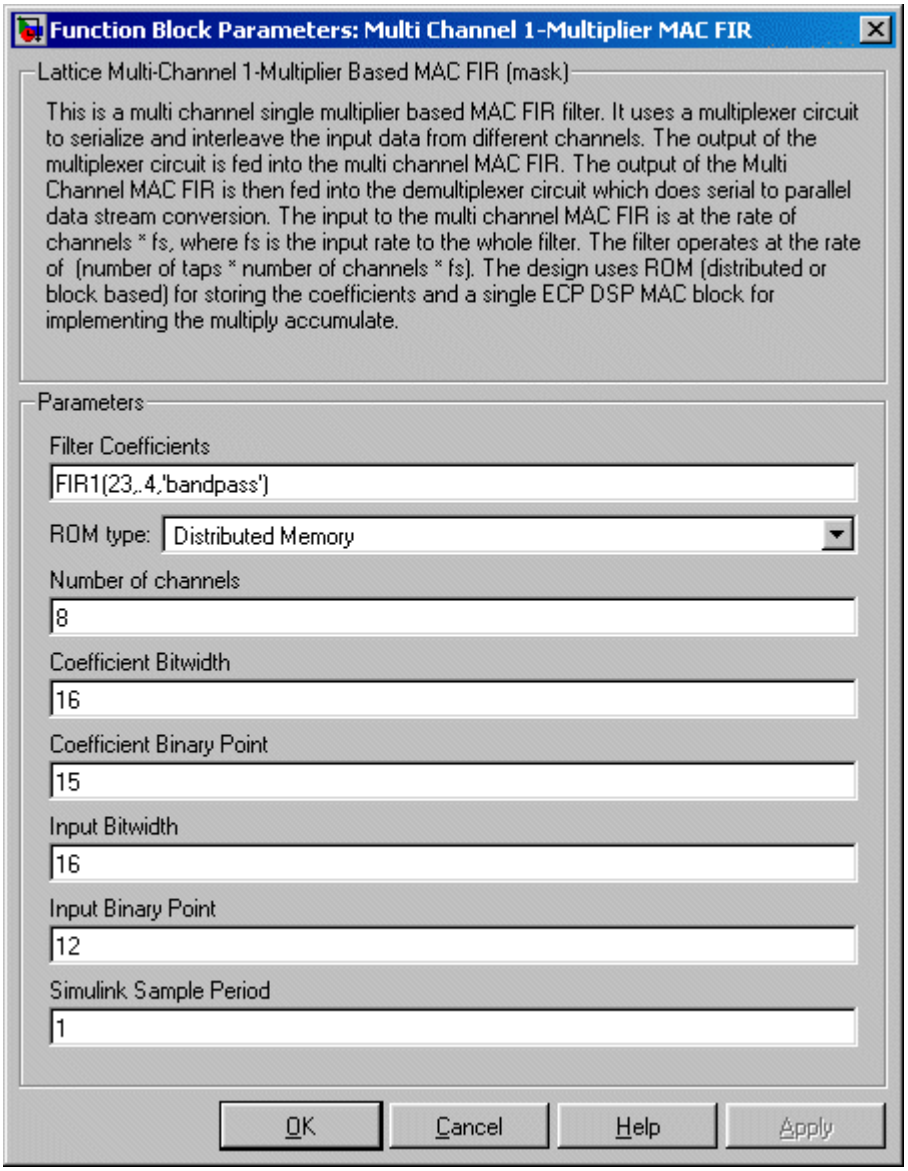
Multi-Channel 1-Multiplier Based MAC FIR



This is a multi-channel single-multiplier based MAC FIR filter. It uses a time division multiplexing (TDM) circuit to serialize the parallel channel input data stream. The output of the time division multiplexing circuit is fed into multi-channel MAC FIR. The output of the Multi Channel MAC FIR is then fed into the time division de-multiplexing (TDD) circuit, which does serial to parallel channel data stream conversion.

The input to the filter arrives at Simulink sample period defined on the filter. The input to the multi-channel MAC FIR is at the rate of Number of Channels * fs, where fs is the input sampling rate in to the filter. The filter operates at the rate of, number of taps * number of channels * fs. The design uses ROM (distributed or block based) for storing the coefficients and a single DSP MAC block for implementing the multiply accumulate.

The user configurable parameters for Lattice Multi-Channel 1-Multiplier Based MAC FIR are available in the following dialog box.



The block parameters are:

- **Filter Coefficients**:

  Accepts MATLAB expression of any type. By default, the coefficients are produced by the MATLAB function FIR1. If you do not have the MATLAB Signal Processing Toolbox, you must enter your own coefficients. The length of the coefficient vector determines the number of filter taps.

- **ROM Type = [Block Memory, Distributed Memory]**
- **Downsampling Option**:

  Select between Last Frame (resource efficient) or First Frame (speed efficient) modes for the down sampler used in the design.

- **Coefficient Bit width**:

  Choose per filter design specification.

- **Coefficient Binary Point**:

  Choose per filter design specification.

- **Input Bitwidth**:

  Choose per filter design specification.

- **Input Sample Binary Point**:

  Choose per filter design specification.

- **Output Full Precision Format**:

  The filter has full precision output with selectable Signed or Unsigned data type format.

- **Filter Output Delay**:

  The filter output delay is the delay you can set to align the filter output with an external circuit. This delay is additional to inherent filter delay that is defined by the implementation.

- **Simulink Sample Period**:

  The Simulink sample period is the input sample period for the FIR. The sampling period of the input source must match the value entered for the Simulink Sample Period.