

Release Notes for HDL Coder™

How to Contact MathWorks



www.mathworks.com Web
comp.soft-sys.matlab Newsgroup
www.mathworks.com/contact_TS.html Technical Support



suggest@mathworks.com Product enhancement suggestions
bugs@mathworks.com Bug reports
doc@mathworks.com Documentation error reports
service@mathworks.com Order status, license renewals, passcodes
info@mathworks.com Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Release Notes for HDL Coder™

© COPYRIGHT 2012–2013 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

R2013a

Static range analysis for floating-point to fixed-point conversion	2
User-specified pipeline insertion for MATLAB variables ..	3
Resource sharing and streaming without over clocking ...	4
Generation of custom IP core with AXI4 interface	5
Coprocessor synchronization in FPGA Turnkey and IP Core Generation workflows	6
Code generation for System objects in a MATLAB Function block	7
Resource sharing for the MATLAB Function block	8
Finer control for delay balancing	9
Complex multiplication optimizations in the Product block	10
Speedgoat IO331 Spartan-6 FPGA board for FPGA Turnkey workflow	11
Cosimulation and FPGA-in-the-Loop for MATLAB HDL code generation	12
HDL coding standard report and lint tool script generation	13
Output folder structure includes model name	14
File I/O to read test bench data in Verilog	15
Prefix for module or entity name	16
Single rate Newton-Raphson architecture for Sqrt, Reciprocal Sqrt	17
Additional System objects supported for code generation ..	18
Additional blocks supported for code generation	19
Functionality being removed	20

R2012b

Input parameter constants and structures in floating-point to fixed-point conversion	22
RAM, biquad filter, and demodulator System objects	23
Generation of MATLAB Function block in the MATLAB to HDL workflow	24

HDL code generation for Reed Solomon encoder and decoder, CRC detector, and multichannel Discrete FIR filter	25
Targeting of custom FPGA boards	26
Optimizations for MATLAB Function blocks and black boxes	27
Generate Xilinx System Generator Black Box block from MATLAB	28
Save and restore HDL-related model parameters	29
Command-line interface for MATLAB-to-HDL code generation	30
User-specifiable clock enable toggle rate in test bench	31
RAM mapping for dsp.Delay System object	32
Code generation for Repeat block with multiple clocks ...	33
Automatic verification with cosimulation using HDL Coder	34
ML605 Board Added To Turnkey Workflow	35

R2012a

Product Name Change and Extended Capability	38
Code Generation from MATLAB	39
Code Generation from Any Level of Subsystem Hierarchy	40
Automated Subsystem Hierarchy Flattening	41
Support for Discrete Transfer Fcn Block	42
User Option to Constrain Registers on Output Ports	43
Distributed Pipelining for Sum of Elements, Product of Elements, and MinMax Blocks	44
MATLAB Function Block Enhancements	45
Automated Code Generation from Xilinx System Generator for DSP Blocks	46
Altera Quartus II 11.0 Support in HDL Workflow Advisor	47
Automated Mapping to Xilinx and Altera Floating Point Libraries	48
Vector Data Type for PCI Interface Data Transfers Between xPC Target and FPGA	49
New Global Property to Select RAM Architecture	50
Turnkey Workflow for Altera Boards	51
HDL Support For Bus Creator and Bus Selector Blocks ..	52
HDL Support For HDL CRC Generator Block	53

HDL Support for Programmable Filter Coefficients	54
Synchronous Multiclock Code Generation for CIC Decimators and Interpolators	55
Filter Block Resource Report Participation	56
HDL Block Properties Interface Allows Choice of Filter Architecture	58
HDL Support for FIR Filters With Serial Architectures and Complex Inputs	60
HDL Support for External Reset Added for Proportional-Integral-Derivative (PID) and Discrete Time Integrator (DTI) Blocks	61

R2013a

Version: 3.2
New Features: Yes
Bug Fixes: No

Static range analysis for floating-point to fixed-point conversion

The coder can now use static range analysis to derive fixed-point data types for your floating-point MATLAB® code.

The redesigned interface for floating-point to fixed-point conversion enables you to use simulation with multiple test benches, static range analysis, or both, to determine fixed-point data types for your MATLAB variables.

For details, see “Automated Fixed-Point Conversion”.

User-specified pipeline insertion for MATLAB variables

You can now specify pipeline register insertion for variables in your MATLAB code. This feature is available in both the MATLAB to HDL workflow and the MATLAB Function block.

To learn how to pipeline variables in the MATLAB to HDL workflow, see “Pipeline MATLAB Variables”.

To learn how to pipeline variables in the MATLAB Function block, see “Pipeline Variables in the MATLAB Function Block”.

Resource sharing and streaming without over clocking

You can now constrain the resource sharing and streaming optimizations to prevent or reduce overclocking. The coder optimizes your design based on two parameters that you specify: maximum oversampling ratio, `MaxOversampling`, and maximum computation latency, `MaxComputationLatency`.

For single-rate resource sharing or streaming, you can set `MaxOversampling` to 1.

To learn more about constrained overclocking, maximum oversampling ratio, and maximum computation latency, see:

- “Optimization With Constrained Overclocking”
- “Maximum Oversampling Ratio”
- “Maximum Computation Latency”

Generation of custom IP core with AXI4 interface

You can now generate custom IP cores with an AXI4-Lite or AXI4-Stream Video interface. You can integrate these custom IP cores with your design in a Xilinx® EDK environment for the Xilinx Zynq®-7000 Platform.

For more details, see “Custom IP Core Generation”.

To view an example that shows how to generate a custom IP core, at the command line, enter:

```
hdlcoder_ip_core_led_blinking
```

Coprocessor synchronization in FPGA Turnkey and IP Core Generation workflows

The coder can now automatically synchronize communication and data transfers between your processor and FPGA. You can use the new **Processor/FPGA synchronization mode** in the FPGA Turnkey workflow with xPC Target™, or when you generate a custom IP core.

For more details, see “Processor and FPGA Synchronization”.

Code generation for System objects in a MATLAB Function block

You can now generate code from a MATLAB Function block containing System objects.

For details, see System Objects under MATLAB Language Support, in “MATLAB Function Block Usage”.

Resource sharing for the MATLAB Function block

You can now specify a resource sharing factor for the MATLAB Function block to share multipliers in the MATLAB code.

For details, see “Resource Sharing” and “Specify Resource Sharing”.

Finer control for delay balancing

You can now disable delay balancing for a subsystem within your DUT subsystem.

For details, see “Balance Delays”.

Complex multiplication optimizations in the Product block

You can now share multipliers used in a single complex multiplication in the Product block. Distributed pipelining can also move registers between the multiply and add stages of a complex multiplication.

Speedgoat IO331 Spartan-6 FPGA board for FPGA Turnkey workflow

You can now use the Speedgoat IO331 Spartan-6 FPGA board in the FPGA Turnkey workflow with xPC Target.

You must have an xPC Target license to use this feature.

Cosimulation and FPGA-in-the-Loop for MATLAB HDL code generation

With the MATLAB HDL Workflow Advisor, the HDL Verification step includes automation for the following workflows:

- **Verify with HDL Test Bench:** Create a standalone test bench. You can choose to simulate a model using ModelSim® or Incisive® with a vector file created by the Workflow Advisor.
- **Verify with Cosimulation:** Cosimulate the DUT in ModelSim or Incisive with the test bench in MATLAB.
- **Verify with FPGA-in-the-Loop:** Create the FPGA programming file and test bench, and, optionally, download it to your selected development board.

You must have an HDL Verifier™ license to use these workflows.

HDL coding standard report and lint tool script generation

You can now generate a report that shows how well your generated HDL code conforms to an industry coding standard. Errors and warnings in the report link to elements in your original design so you can fix problems.

You can also generate third-party lint tool scripts to use to check your generated HDL code. In this release, you can generate LEDA, Spyglass, and generic scripts.

To learn more about the coding standard report, see “HDL Coding Standard Report”.

To learn how to generate a coding standard report and lint tool script in the Simulink® to HDL workflow, see:

- “Generate an HDL Coding Standard Report”
- “Generate an HDL Lint Tool Script”

To learn how to generate a coding standard report and lint tool script in the MATLAB to HDL workflow, see:

- “Generate an HDL Coding Standard Report”
- “Generate an HDL Lint Tool Script”

Output folder structure includes model name

Compatibility Considerations: Yes

When you generate code for a subsystem within a model, the output folder structure now includes the model name.

For example, if you generate code for a subsystem in a model, `Mymodel`, the output folder is `hdlsrc/Mymodel`.

Compatibility Considerations

If you have scripts that depend on a specific output folder structure, you must update them with the new structure.

File I/O to read test bench data in Verilog

You can now specify the generated HDL test bench to use file I/O to read input stimulus and output response data during simulation, instead of including data constants in the test bench code. Doing so improves scalability for designs needing long simulations.

This feature is available when Verilog® is the target language.

For details, see “Test Bench Generation with File I/O”.

Prefix for module or entity name

You can now specify a prefix for every module or entity name in the generated HDL code. This feature helps you to avoid name clashes when you want to have multiple instances of the HDL code generated from the same block. For details, see `ModulePrefix`.

Single rate Newton-Raphson architecture for Sqrt, Reciprocal Sqrt

The Sqrt, Reciprocal Sqrt, reciprocal Divide, and reciprocal Math Function blocks now have a single-rate pipelined architecture. The new architecture enables you to use the high-speed Newton-Raphson algorithm without multirate or overclocking.

The following table lists each block with its new block implementation.

Block	Implementation Name	Details
Sqrt	SqrtNewtonSingleRate	See “Sqrt”.
Reciprocal Sqrt	RecipSqrtNewtonSingleRate	See “Reciprocal Sqrt”.
Divide (reciprocal)	RecipNewtonSingleRate	See “Divide (reciprocal)”.
Math Function (reciprocal)	RecipNewtonSingleRate	See “Math Function (reciprocal)”.

Additional System objects supported for code generation

Effective with this release, the following System objects provide HDL code generation:

- `comm.HDLCRCGenerator`
- `comm.HDLCRCDetector`
- `comm.HDLRSEncoder`
- `comm.HDLRSDecoder`
- `dsp.HDLNCO`

Additional blocks supported for code generation

The following blocks are now supported for HDL code generation:

- NCO HDL Optimized
- Bias
- Relay
- Dot Product
- Sum with more than two inputs with different signs
- MinMax with multiple input data types

Functionality being removed

Compatibility Considerations: Yes

Property Name	What Happens When You Use This Property?	Use This Property Instead	Compatibility Considerations
RAMStyle	Error	RAMArchitecture	The new property syntax differs. Replace existing instances of RAMStyle with the correct RAMArchitecture syntax.
GainImpls	Error	"ConstMultiplierOptimization"	The "ConstMultiplierOptimization" property syntax differs. Replace existing instances of GainImpls with the correct "ConstMultiplierOptimization" syntax.

R2012b

Version: 3.1
New Features: Yes
Bug Fixes: No

Input parameter constants and structures in floating-point to fixed-point conversion

Floating-point to fixed-point conversion now supports structures and constant value inputs.

RAM, biquad filter, and demodulator System objects

HDL RAM System object

With release 2012b, you can use the `hdlram` System object™ for modeling and generating fixed-point code for RAMs in FPGAs and ASICs. The `hdlram` System object provides simulation capability in MATLAB for Dual Port, Simple Dual Port, and Single Port RAM. The System object also generates RTL code that can be inferred as a RAM by most synthesis tools.

To learn how to model and generate RAMs using the `hdlram` System object, see [Model and Generate RAM with `hdlram`](#).

HDL System object support for biquad filters

HDL support has been added for the following System object:

- `dsp.BiquadFilter`

HDL support with demodulator System objects

HDL support has been added for the following System objects:

- `comm.BPSKDemodulator`
- `comm.QPSKDemodulator`
- `comm.PSKDemodulator`
- `comm.RectangularQAMDemodulator`
- `comm.RectangularQAMModulator`

Generation of MATLAB Function block in the MATLAB to HDL workflow

You can now generate a MATLAB Function block during the MATLAB to HDL workflow. You can use the generated block for further design, simulation, and code generation in Simulink.

For details, see [MATLAB Function Block Generation](#).

HDL code generation for Reed Solomon encoder and decoder, CRC detector, and multichannel Discrete FIR filter

HDL code generation

In R2012b, HDL code generation support has been added for the following blocks:

- General CRC Syndrome Detector HDL Optimized
For an example of using the HDL-optimized CRC generator and detector blocks, see [Using HDL Optimized CRC Library Blocks](#).
- Integer-Input RS Encoder HDL Optimized
- Integer-Output RS Decoder HDL Optimized

Multichannel Discrete FIR filters

The Discrete FIR Filter block accepts vector input and supports multichannel implementation for better resource utilization.

- With vector input and channel sharing option `on`, the block supports multichannel fully parallel FIR, including direct form FIR, sym/antisym FIR, and FIRT. Support for all implementation parameters, for example: multiplier pipeline, add pipeline registers.
- With vector input and channel sharing option `off`, the block instantiates one filter implementation for each channel. If the input vector size is N , N identical filters are instantiated.

Applies to the fully parallel architecture option for FIR filters only.

Targeting of custom FPGA boards

The FPGA Board Manager and New FPGA Board Wizard allow you to add custom board information so that you can use FIL simulation with an FPGA board that is not one of the pre-registered boards. See “FPGA Board Customization”.

Optimizations for MATLAB Function blocks and black boxes

The resource sharing optimization now operates on MATLAB Function blocks. For details, see [Specify Resource Sharing](#).

The delay balancing and distributed pipelining optimizations now operate on black box subsystems. To learn how to specify latency and enable distributed pipelining for a black box subsystem, see [Customize the Generated Interface](#).

Generate Xilinx System Generator Black Box block from MATLAB

You can now generate a Xilinx System Generator Black Box block during the MATLAB-to-HDL workflow. You can use the generated block for further design, simulation, and code generation in Simulink.

For details, see [Xilinx System Generator Black Box Block Generation](#).

Save and restore HDL-related model parameters

Two new functions, `hdlsaveparams` and `hdlrestoreparams`, enable you to save and restore nondefault HDL-related model parameters. Using these functions, you can perform multiple iterations on your design to optimize the generated code.

For details, see `hdlsaveparams` and `hdlrestoreparams`.

Command-line interface for MATLAB-to-HDL code generation

You can now convert your MATLAB code from floating-point to fixed-point and generate HDL code using the command-line interface.

To learn how to use the command line interface, open the tutorial:

```
showdemo mlhdlc_tutorial_cli
```

User-specifiable clock enable toggle rate in test bench

You can now specify the clock enable toggle rate in your test bench to match your input data rate or improve test coverage.

To learn how to specify your test bench clock enable toggle rate, see [Test Bench Clock Enable Toggle Rate Specification](#).

RAM mapping for dsp.Delay System object

The dsp.Delay System object now maps to RAM if the RAM mapping optimization is enabled and the delay size meets the RAM mapping threshold.

To learn how to map the dsp.Delay System object to RAM, see [Map Persistent Arrays and dsp.Delay to RAM](#).

Code generation for Repeat block with multiple clocks

You can now generate code for the DSP System Toolbox™ Repeat block in a model with multiple clocks.

Automatic verification with cosimulation using HDL Coder

With the HDL Coder™ HDL Workflow Advisor, you can automatically verify using your Simulink test bench with the new verification step **Run Cosimulation Test Bench**. During verification, the HDL Workflow Advisor and HDL Verifier verify the generated HDL using cosimulation between the HDL Simulator and the Simulink test bench. See Automatic Verification in the HDL Verifier documentation.

ML605 Board Added To Turnkey Workflow

The Xilinx Virtex-6 FPGA ML605 board has been added for Turnkey Workflow in the HDL Workflow Advisor.

R2012a

Version: 3.0
New Features: Yes
Bug Fixes: No

Product Name Change and Extended Capability

HDL Coder replaces Simulink HDL Coder and adds the HDL code generation capability directly from MATLAB.

To generate HDL code from MATLAB, you need the following products:

- HDL Coder
- MATLAB Coder™
- Fixed-Point Toolbox™
- MATLAB

To generate HDL code from Simulink, you need the following products:

- HDL Coder
- MATLAB Coder
- Fixed-Point Toolbox
- Simulink Fixed Point™
- Simulink
- MATLAB

Code Generation from MATLAB

You can now generate HDL code directly from MATLAB code.

This workflow provides:

- Verilog or VHDL® code generation from MATLAB code.
- Test bench generation from MATLAB scripts.
- Automated conversion from floating point code to fixed point code.
- Automated HDL verification through integration with ModelSim and ISim.
- HDL code generation for a subset of System objects from the Communications System Toolbox™ and DSP System Toolbox.
- A traceability report mapping generated HDL code to your original MATLAB code.

The MATLAB to HDL workflow provides the following automated HDL code optimizations:

- Area optimizations: RAM mapping for persistent array variables, loop streaming, resource sharing, and constant multiplier optimization.
- Speed optimizations: input pipelining, output pipelining, and distributed pipelining.

The coder can also generate a resource utilization report, with RAM usage and the number of adders, multipliers, and muxes in your design.

See also [HDL Code Generation from MATLAB](#).

Code Generation from Any Level of Subsystem Hierarchy

You can now generate HDL code from a subsystem at any level of the subsystem hierarchy. In previous releases, you could generate HDL code from the top-level subsystem only.

This feature also enables you to check any level subsystem for code generation compatibility, and to automatically generate a testbench.

Automated Subsystem Hierarchy Flattening

You can now generate code with a flattened subsystem hierarchy, while preserving hierarchy in nested subsystems.

This option enables you to perform more extensive area and speed optimization on the flattened component. It also enables you to reduce the number of HDL output files.

See also [Hierarchy Flattening](#).

Support for Discrete Transfer Fcn Block

You can now generate HDL code from the Discrete Transfer Fcn block.

For details, see Discrete Transfer Fcn Requirements and Restrictions.

User Option to Constrain Registers on Output Ports

A new property, `ConstrainedOutputPipeline`, enables you to specify the number of registers you wish to have on an output port without introducing additional delay on the input to output path. The coder redistributes existing delays within your design to try to meet the constraint. This behavior is different from the `OutputPipeline` property, which introduces additional delay on the input to output path.

If the coder is unable to meet the constraint using existing delays, it reports the difference between the number of desired and actual output registers in the timing report.

Distributed Pipelining for Sum of Elements, Product of Elements, and MinMax Blocks

The Sum of Elements, Product of Elements, and MinMax blocks can now participate in distributed pipelining if their architecture is set to Tree.

MATLAB Function Block Enhancements

Multiple Accesses to RAMs Mapped from Persistent Variables

You can now perform multiple reads and writes to a persistent variable, and the persistent variable will still be mapped to RAM. In previous releases, a RAM mapped from a persistent variable could be accessed only once.

Streaming for MATLAB Loops and Vector Operations

You can now perform streaming on MATLAB loops and loops created from vector operations for improved area efficiency.

For details, see [Loop Optimization](#).

Loop Unrolling for MATLAB Loops and Vector Operations

You can now unroll user-written MATLAB loops and loops created from vector operations. This enables the coder to perform area and speed optimizations on the unrolled loops.

For details, see [Loop Optimization](#).

Automated Code Generation from Xilinx System Generator for DSP Blocks

You can now automatically generate HDL code from subsystems containing Xilinx System Generator for DSP blocks.

For details, see [Code Generation with Xilinx System Generator Subsystems](#).

Altera Quartus II 11.0 Support in HDL Workflow Advisor

The HDL Workflow Advisor has now been tested with Altera® Quartus II 11.0. In previous releases, the HDL Workflow Advisor was tested with Altera Quartus II 9.1.

Automated Mapping to Xilinx and Altera Floating Point Libraries

The coder can now map Simulink floating point operations to synthesizable floating point Altera Megafunctions and Xilinx LogiCORE IP Floating Point Operator v5.0 blocks. To learn more, see [FPGA Target-Specific Floating-Point Library Mapping](#).

For a list of supported Altera Megafunction blocks, see [Supported Altera Floating-Point Library Blocks](#).

For a list of supported Xilinx LogicCORE IP blocks, see [Supported Xilinx Floating-Point Library Blocks](#).

Vector Data Type for PCI Interface Data Transfers Between xPC Target and FPGA

In the FPGA Turnkey workflow, you can now use vector data types with the **Scalarize Vector Ports** option to automatically generate PCI DMA transfers on the PCI interface between xPC Target and FPGA. You no longer need to manually insert multiplexers, demultiplexers and provide synchronization logic for vector data transfers.

If the **Scalarize Vector Ports** option is disabled when the code generation subsystem has vector ports, the coder displays an error.

New Global Property to Select RAM Architecture

Compatibility Considerations: Yes

There is a new global property, `RAMArchitecture`, that enables you to generate RAMs either with or without clock enables. This property applies to every RAM in your design, and replaces the block level property, `RAMStyle`. By default, RAMs are generated with clock enables.

To generate RAMs without clock enables, set `RAMArchitecture` to `'WithoutClockEnable'`. To generate RAMs with clock enables, either use the default, or set `RAMArchitecture` to `'WithClockEnable'`. For more information, see [Implement RAMs With or Without Clock Enable](#).

Compatibility Considerations

The coder now ignores the block level property, `RAMStyle`.

If a block's `RAMStyle` property is set, the coder generates a warning.

Turnkey Workflow for Altera Boards

HDL Workflow Advisor now supports Altera FPGA design software and the following Altera development kits and boards:

- Altera Arria II GX FPGA development kit
- Altera Cyclone III FPGA development kit
- Altera Cyclone IV GX FPGA development kit
- Altera DE2-115 development and education board

This workflow has been tested with Altera Quartus II 11.0.

HDL Support For Bus Creator and Bus Selector Blocks

Release R2012a provides HDL code generation for the Bus Creator and Bus Selector blocks. You must use these blocks for your buses if you want HDL support.

HDL Support For HDL CRC Generator Block

Release R2012a provides HDL code generation for the new HDL CRC Generator block.

HDL Support for Programmable Filter Coefficients

When using filter blocks to generate HDL code, you can specify coefficients from input port(s). This feature applies to FIR and BiQuad filter blocks only. Fully Parallel and all serial architectures are supported.

Follow these directions to use programmable filters:

- 1 Select Input port(s) as coefficient source from the filter block mask.
- 2 Connect the coefficient port with a vector signal.
- 3 Specify the implementation architecture and parameters from the HDL Coder property interface.
- 4 Generate HDL code.

Notes

- For fully parallel implementations, the coefficients ports are connected to the dedicated MAC directly.
- For serial implementation, the coefficients ports first go to a mux, and then to the MAC. The mux decides the coefficients that used at current time instant
- For Discrete FIR filters, this feature is not supported under the following conditions:
 - Implementations having coefficients specified by dialog parameters (for example, complex input and coefficients with serial architecture)
 - Filters using DA architecture
 - CoeffMultipliers specified as `csd` or `factored-csd`
- For Biquad filters, this feature is not supported when CoeffMultipliers are specified as `csd` or `factored-csd`.

Synchronous Multiclock Code Generation for CIC Decimators and Interpolators

You can specify multiple clocks in one of the following ways:

- Use the model-level parameter `ClockInputs` with the function `makehdl` and specify the value as 'Multiple'.
- In the Clock settings section of the **Global Settings** pane in the HDL Code Generation Configuration Parameters dialog box, set **Clock inputs** to `Multiple`.

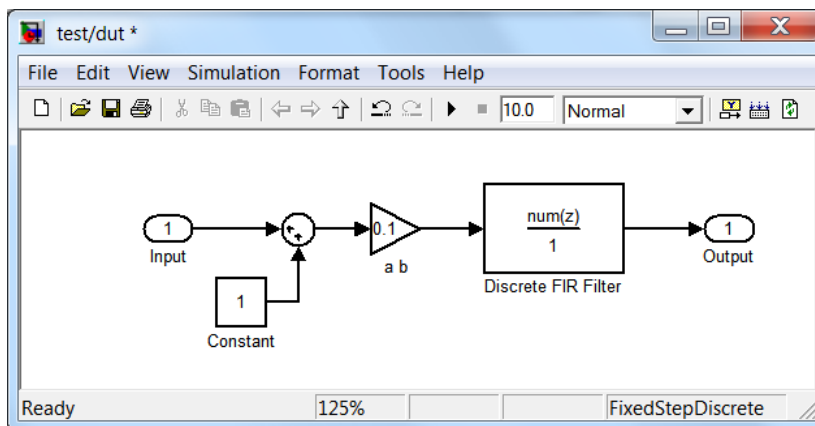
When you use single-clock mode, HDL code generated from multirate models uses a single master clock that corresponds to the base rate of the DUT. When you use multiple-clock mode, HDL code generated from multirate models use one clock input for each rate in the DUT. The number of timing controllers generated in multiple-clock mode depends on the design in the DUT.

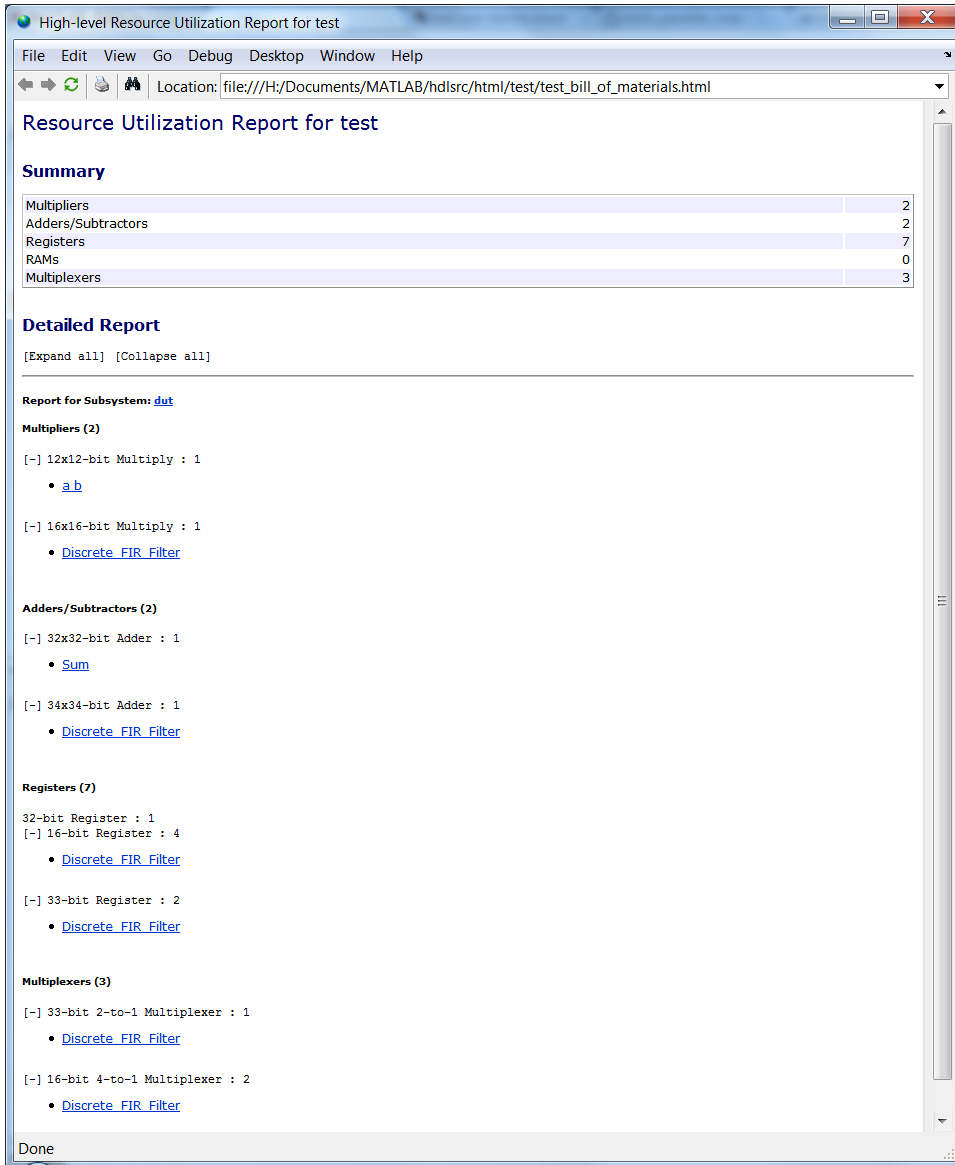
The `ClockInputs` parameter supports the values 'Single' and 'Multiple', where the default is 'Single'. In the default single-clock mode, the coder behavior is unchanged from previous releases.

Filter Block Resource Report Participation

Resource reports include the HDL resource usage for filter blocks. The report includes adders, subtractors, multipliers, multiplexers, registers. This feature covers all filter blocks, and all implementations for the block.

You can turn on the report feature using the command line (`ResourceReport`) or GUI (**Generate resource utilization report**). The following illustrations show a report for a model that includes a Discrete FIR Filter block.





High-level Resource Utilization Report for test

File Edit View Go Debug Desktop Window Help

Location: file:///H:/Documents/MATLAB/hdsrc/html/test/test_bill_of_materials.html

Resource Utilization Report for test

Summary

Multipliers	2
Adders/Subtractors	2
Registers	7
RAMs	0
Multiplexers	3

Detailed Report

[Expand all] [Collapse all]

Report for Subsystem: [dut](#)

Multipliers (2)

[-] 12x12-bit Multiply : 1

- [a_b](#)

[-] 16x16-bit Multiply : 1

- [Discrete FIR Filter](#)

Adders/Subtractors (2)

[-] 32x32-bit Adder : 1

- [Sum](#)

[-] 34x34-bit Adder : 1

- [Discrete FIR Filter](#)

Registers (7)

32-bit Register : 1

[-] 16-bit Register : 4

- [Discrete FIR Filter](#)

[-] 33-bit Register : 2

- [Discrete FIR Filter](#)

Multiplexers (3)

[-] 33-bit 2-to-1 Multiplexer : 1

- [Discrete FIR Filter](#)

[-] 16-bit 4-to-1 Multiplexer : 2

- [Discrete FIR Filter](#)

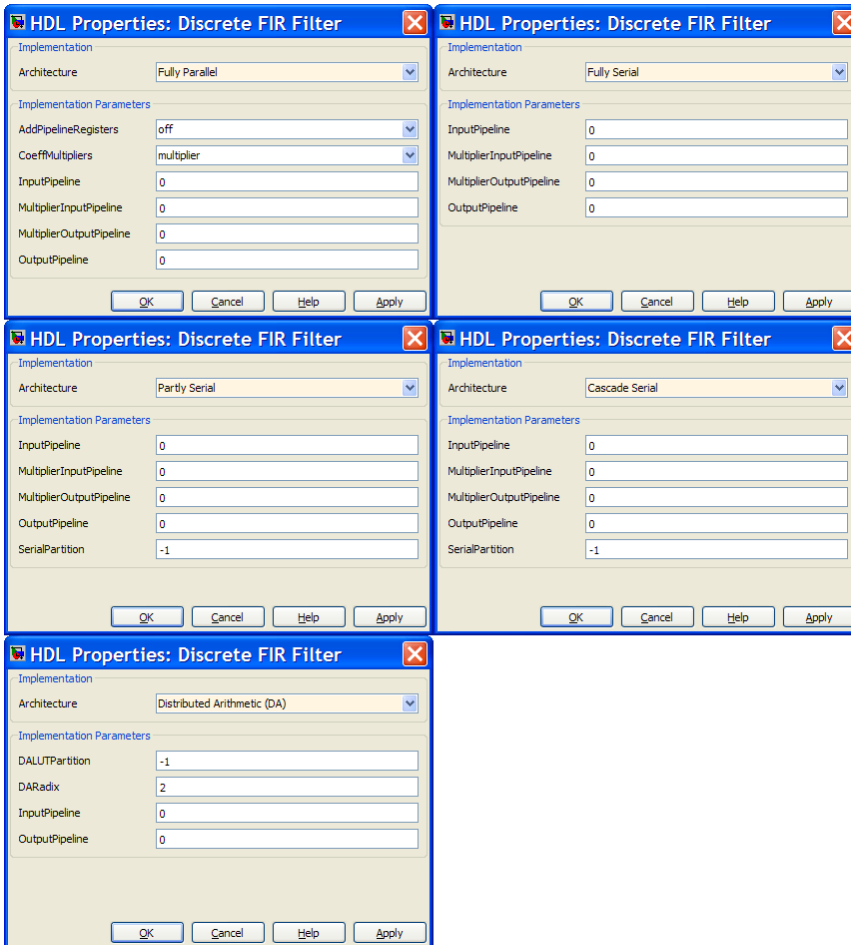
Done

HDL Block Properties Interface Allows Choice of Filter Architecture

You can choose from several filter architectures for FIR Decimation and Discrete FIR Filter blocks. Choices are:

- Fully Parallel
- Distributed Architecture (DA)
- Fully Serial
- Partly Serial
- Cascade Serial

The availability of architectures depends on the transfer function type and filter structure of filter blocks. For Partly Serial and DA, specify at least **SerialPartition** and **DALUTPartition**, respectively, so that these architectures are inferred. For example, if you select Distributed Architecture (DA), make sure to also set **DALUTPartition**.



HDL Support for FIR Filters With Serial Architectures and Complex Inputs

HDL support for serial implementations of a FIR block with complex inputs.

HDL Support for External Reset Added for Proportional-Integral-Derivative (PID) and Discrete Time Integrator (DTI) Blocks

External reset support added for level mode.