HDL Coder™ Release Notes

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.

HDL Coder™ Release Notes

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.

Contents

R2013b

R2013a

R2012b

R2012a

R2013b

Version: 3.3

New Features: Yes

Bug Fixes: Yes

Model reference support and incremental code generation

You can generate HDL code from referenced models using the Model block. To use a referenced model in a subsystem intended for code generation, in the HDL Block Properties dialog box, set **Architecture** to **ModelReference**.

The coder incrementally generates code for referenced models according to the **Configuration Parameters dialog box > Model Referencing pane > Rebuild** options. However, the coder treats **If any changes detected** and **If any changes in known dependencies detected** as the same. For example, if you set **Rebuild** to either **If any changes detected** or **If any changes in known dependencies detected**, the coder regenerates code for referenced models only when the referenced models have changed.

To learn more, see "Model Referencing for HDL Code Generation".

Code generation for user-defined System objects

You can now generate HDL code from user-defined System objects written in MATLAB®. System objects enable you to create reusable HDL IP.

The `step` method specifies the HDL implementation behavior. It is the only System object™ method supported for HDL code generation.

User-defined System objects are not supported for automatic fixed-point conversion.

To learn how to define a custom System object, see "Generate Code for User-Defined System Objects".

RAM inference in conditional MATLAB code

The coder now infers RAM from persistent array variables accessed within conditional statements, such as if-else or switch-case statements, for both MATLAB designs and MATLAB Function blocks in Simulink®.

If you have nested conditional statements, the persistent array variables can map to RAM if accessed in the topmost conditional statement, but cannot map to RAM if accessed in a lower level nested conditional statement.

Code generation for subsystems containing Altera DSP Builder blocks

You can now generate HDL code for subsystems that include blocks from the Altera® DSP Builder Advanced Blockset.

For details, see “Create an Altera DSP Builder Subsystem”.

To see an example that shows HDL code generation for an Altera DSP Builder subsystem, see Using Altera DSP Builder Advanced Blockset with HDL Coder.

IP core integration into Xilinx EDK project for ZC702 and ZedBoard

When you generate an IP core from your MATLAB design or Simulink model, HDL Coder™ can automatically insert the IP core into a predefined Xilinx® ZC702 or ZedBoard™ EDK project for the Zynq®-7000 platform. The coder automatically connects the IP core to the AXI interface and ARM processor in the EDK project.

For an overview of the hardware and software codesign workflow, see “Hardware and Software Codesign Workflow”.

For an example that shows how to deploy your MATLAB design in hardware and software on the Zynq-7000 platform, see Getting Started with HW/SW Co-design Workflow for Xilinx Zynq Platform.

For an example that shows how to deploy your Simulink model in hardware and software on the Zynq-7000 platform, see Getting Started with HW/SW Co-design Workflow for Xilinx Zynq Platform.

FPGA Turnkey and IP Core generation in MATLAB to HDL workflow

You can now generate a custom IP core with an AXI4-Lite or AXI4-Stream Video interface from a MATLAB design. You can integrate the generated IP core into a larger design in your Xilinx EDK project.

You can also automatically program an Altera or Xilinx FPGA development board with code generated from your MATLAB design, using the HDL Workflow Advisor FPGA Turnkey workflow. To learn how to use this workflow, see "Program Standalone FPGA with FPGA Turnkey Workflow" and mlhdlc_tutorial_turnkey_led_blinking.

Previously, IP core generation and FPGA Turnkey were available only for the Simulink to HDL workflow.

Module or entity generation for local functions in MATLAB Function block

You can now generate instantiable Verilog® modules or VHDL® entities when you generate code for local functions in a MATLAB Function block, or for functions on your path that are called from within a MATLAB Function block.

To enable this feature, in the HDL Block Properties dialog box, set **InstantiateFunctions** to **on**. For details, see "InstantiateFunctions".

Bus signal inputs and outputs for MATLAB Function block and Stateflow charts

MATLAB Function blocks and Stateflow® charts with bus signal inputs or outputs are now supported for code generation. The bus must be defined with a bus object.

Coding style for improved ROM mapping

The coder now automatically inserts a no-reset register at the output of a constant matrix access. Many synthesis tools infer a ROM from this code pattern. For details, see "Map Matrices to ROM".

Reset port optimization

The coder no longer generates a top level reset port when the `ResetType` HDL block parameter is set to `none` for all RAM blocks in the DUT.

In previous releases, the generated code included a reset port even when the RAM reset logic was suppressed.

Pipeline registers between adder or multiplier and rounding or saturation logic

The coder now places a pipeline register between an adder or multiplier and associated rounding or saturation logic when distributing pipelining registers. This register placement can significantly improve clock frequency.

Coding style improvements according to industry standard guidelines

The coder now follows these industry standard coding style guidelines when generating HDL code:

- Division by a power of 2 becomes a bit shift operation.
- Constants with double data types in the original design are automatically converted to their canonical fixed-point types as long as there is no loss of precision.
- SystemVerilog keywords are treated as reserved words.
- Intermediate signals and latches are reduced when **HDLCodingStandard** is set to **Industry**.

- Real data types generate warnings, except when you target an FPGA floating-point library.

Coding standard report target language enhancement and text file format

HDL Coder now generates the coding standard report according to target language. Coding standard errors, warnings, and messages that do not pertain to your target language no longer appear in the report.

The coding standard report is generated in text file format, in addition to HTML format, to enable easier comparison between multiple runs.

Load constants from MAT-files

HDL Coder now generates code for the `coder.load` function, which you can use to load compile-time constants from a MAT-file. You no longer have to manually type in constants that were stored in a MAT-file.

To learn how to use `coder.load` for HDL code generation, see "Load constants from a MAT-File".

UI for SpyGlass, Leda, and custom lint tool script generation

You can now use the UI to generate Atrenta SpyGlass, Synopsys® Leda, or custom lint scripts in the Simulink-to-HDL and MATLAB-to-HDL workflows.

To learn about HDL lint script generation for your Simulink design, see "Generate an HDL Lint Tool Script".

To learn about HDL lint script generation for your MATLAB design, see "Generate an HDL Lint Tool Script".

Synthesis tool addition and detection after MATLAB-to-HDL project creation

You can now set up and add a synthesis tool after creating a MATLAB-to-HDL project without having to close and reopen the project. In the HDL Workflow Advisor, in the **Set Code Generation Target** task, click **Refresh list** to detect and add the new tool. For details, see "Add Synthesis Tool for Current MATLAB Session".

Synthesis script generation for Microsemi Libero and other synthesis tools

You can now generate a Microsemi Libero or custom synthesis tool script during Simulink-to-HDL and MATLAB-to-HDL code generation.

In the MATLAB-to-HDL workflow, you can now generate synthesis tool scripts customized for Xilinx ISE, Microsemi Libero, Mentor Graphics® Precision, Altera Quartus II, and Synopsys Synplify Pro®. The coder populates the scripts with default options, but you can further customize the scripts as needed. In previous releases, you had to enter the synthesis tool commands manually. For details, see "Generate Synthesis Scripts".

File I/O to read test bench data in VHDL and Verilog

You can now specify the generated VHDL or Verilog 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 requiring long simulations and large test vectors.

This feature is available for Simulink-to-HDL and MATLAB-to-HDL code generation.

To learn about test bench generation with file I/O in the Simulink-to-HDL workflow, see "Generate Test Bench With File I/O".

To learn about test bench generation with file I/O in the MATLAB-to-HDL workflow, see "Generate Test Bench With File I/O".

Floating-point library mapping for mixed floating-point and fixed-point designs

When you enable FPGA target-specific floating-point library mapping, you can now generate code from a design containing both floating-point and fixed-point components. The coder determines whether to map to a floating-point IP block based on the data types in your model.

xPC Target FPGA I/O workflow separate from FPGA Turnkey workflow

The HDL Workflow Advisor target workflow that programs Speedgoat boards to run with xPC Target™ is now called the **xPC Target FPGA I/O** workflow. This workflow is separate from the FPGA Turnkey workflow for Altera and Xilinx FPGA boards.

For an example that shows how to use the xPC Target FPGA I/O workflow, see “Generate xPC Target Interface for Speedgoat Boards”.

AXM-A75 AD/DA module for Speedgoat IO331 FPGA board

The AXM-A75 AD/DA module for Speedgoat IO331 FPGA board is now available as a hardware target for the **xPC Target FPGA I/O** workflow.

Speedgoat IO321 and IO321-5 target hardware support

The xPC Target FPGA I/O workflow now supports the Speedgoat IO321 board and its variant, Speedgoat IO321-5, as separate hardware targets. Previously, the name of the IO321-5 board was IO325.

To learn more about the IO321 and IO321-5 boards, see Speedgoat IO321.

Distributed pipelining improvements with loop unrolling in MATLAB Function block

When you enable distributed pipelining for a MATLAB Function block without persistent variables, set the **Loop Optimization** option to `Unrolling` for better timing results.

HDL Counter has specifiable start value

You can now specify a start value for the HDL Counter block. When the counter initializes or wraps around, it counts from the specified start value.

Maximum 32-bit address for RAM

For the Single Port RAM block, Simple Dual Port RAM block, Dual Port RAM block, and `hdlram` System object, the maximum address width is now 32 bits. For more information, see:

- `hdlram`
- "RAM Blocks"

Removing HDL Support for NCO Block

Compatibility Considerations: Yes

HDL support for the NCO block will be removed in a future release. Use the NCO HDL Optimized block instead.

Compatibility Considerations

In the current release, if you generate HDL code for the NCO block, a warning message appears. In a future release, any attempt to generate HDL code for the NCO block will cause an error.

Fixed-point file name change

Compatibility Considerations: Yes

The suffix for generated fixed-point files is now `_fixpt`. Previously, the suffix was `_FixPt`.

Compatibility Considerations

If you have MATLAB-to-HDL projects from previous releases that depend on the generated fixed-point file name, you can use the `FixPtFileNameSuffix` property to set the suffix to `_FixPt`.

Support package for Xilinx Zynq-7000 platform

Generate a custom IP core for the ZC702 or ZedBoard on the Xilinx Zynq-7000 platform using the IP core generation workflow.

To install this support package for MATLAB-to-HDL code generation:

1. In the HDL Workflow Advisor, in the **Select Code Generation Target** task, set **Workflow** to **IP Core Generation**.
2. For **Platform**, select **Get more**.
3. Use Support Package Installer to install the HDL Coder Support Package for Xilinx Zynq-7000 Platform.

To install this support package for Simulink-to-HDL code generation:

1. In the HDL Workflow Advisor, in the **Set Target > Set Target Device and Synthesis Tool** task, set **Target workflow** to **IP Core Generation**.
2. For **Target platform**, select **Get more**.
3. Use Support Package Installer to install the HDL Coder Support Package for Xilinx Zynq-7000 Platform.

Support package for Altera FPGA boards

Compatibility Considerations: Yes

Program Altera FPGA boards with your generated HDL code using the FPGA Turnkey workflow.

To install this support package for MATLAB-to-HDL code generation:

1. In the HDL Workflow Advisor, in the **Select Code Generation Target** task, set **Workflow** to **FPGA Turnkey**.
2. For **Platform**, select **Get more boards**.
3. Use Support Package Installer to install the HDL Coder Support Package for Altera FPGA Boards.

To install this support package for Simulink-to-HDL code generation:

1. In the HDL Workflow Advisor, in the **Set Target > Set Target Device and Synthesis Tool** task, set **Target workflow** to **FPGA Turnkey**.
2. For **Target platform**, select **Get more boards**.
3. Use Support Package Installer to install the HDL Coder Support Package for Altera FPGA Boards.

Compatibility Considerations

Previous versions of HDL Coder had built-in support for Altera FPGA boards in the FPGA Turnkey workflow. The current version of HDL Coder does not have built-in support for Altera FPGA boards. To get support for Altera FPGA boards, install the HDL Coder Support Package for Altera FPGA Boards.

Support package for Xilinx FPGA boards

Compatibility Considerations: Yes

Program Xilinx FPGA boards with your generated HDL code using the FPGA Turnkey workflow.

To install this support package for MATLAB-to-HDL code generation:

1. In the HDL Workflow Advisor, in the **Select Code Generation Target** task, set **Workflow** to **FPGA Turnkey**.
2. For **Platform**, select **Get more boards**.
3. Use Support Package Installer to install the HDL Coder Support Package for Xilinx FPGA Boards.

To install this support package for Simulink-to-HDL code generation:

1. In the HDL Workflow Advisor, in the **Set Target > Set Target Device and Synthesis Tool** task, set **Target workflow** to **FPGA Turnkey**.
2. For **Target platform**, select **Get more boards**.
3. Use Support Package Installer to install the HDL Coder Support Package for Xilinx FPGA Boards.

Compatibility Considerations

Previous versions of HDL Coder had built-in support for Xilinx FPGA boards in the FPGA Turnkey workflow. The current version of HDL Coder does not have built-in support for Xilinx FPGA boards. To get support for Xilinx FPGA boards, install the HDL Coder Support Package for Xilinx FPGA Boards.

Floating point for FIL and HDL cosimulation test bench generation

With the R2013b release, HDL Coder HDL workflow advisor for Simulink supports double and single data types on the DUT interface for test bench generation using HDL Verifier™.

Additional FPGA board support for FIL verification, including Xilinx KC705 and Altera DSP Development Kit, Stratix V edition

Several FPGA boards have been added to the HDL Verifier FPGA board support packages, including Xilinx KC705 and Altera DSP Development Kit,

Stratix V edition. You can select these boards for FIL verification using the HDL workflow advisor for Simulink.

R2013a

Version: 3.2

New Features: Yes

Bug Fixes: Yes

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 new 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 betweenxPC 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

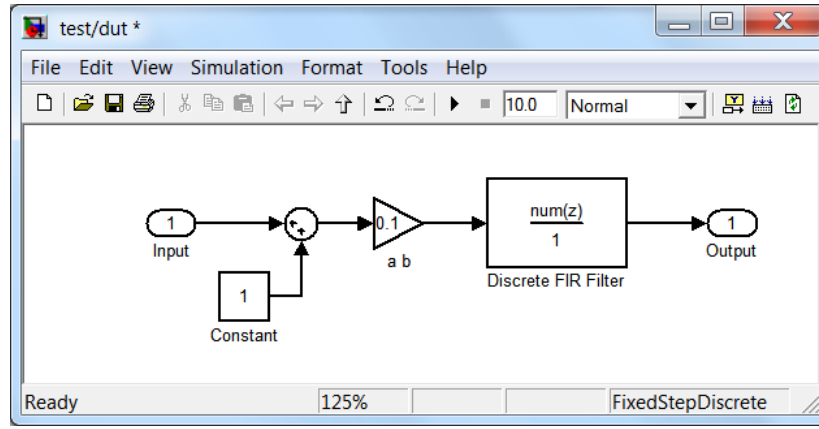
one clock input for each rate in the DUT. The number of timing controllers generated in multiple-clock mode depends on the design in the DUT.

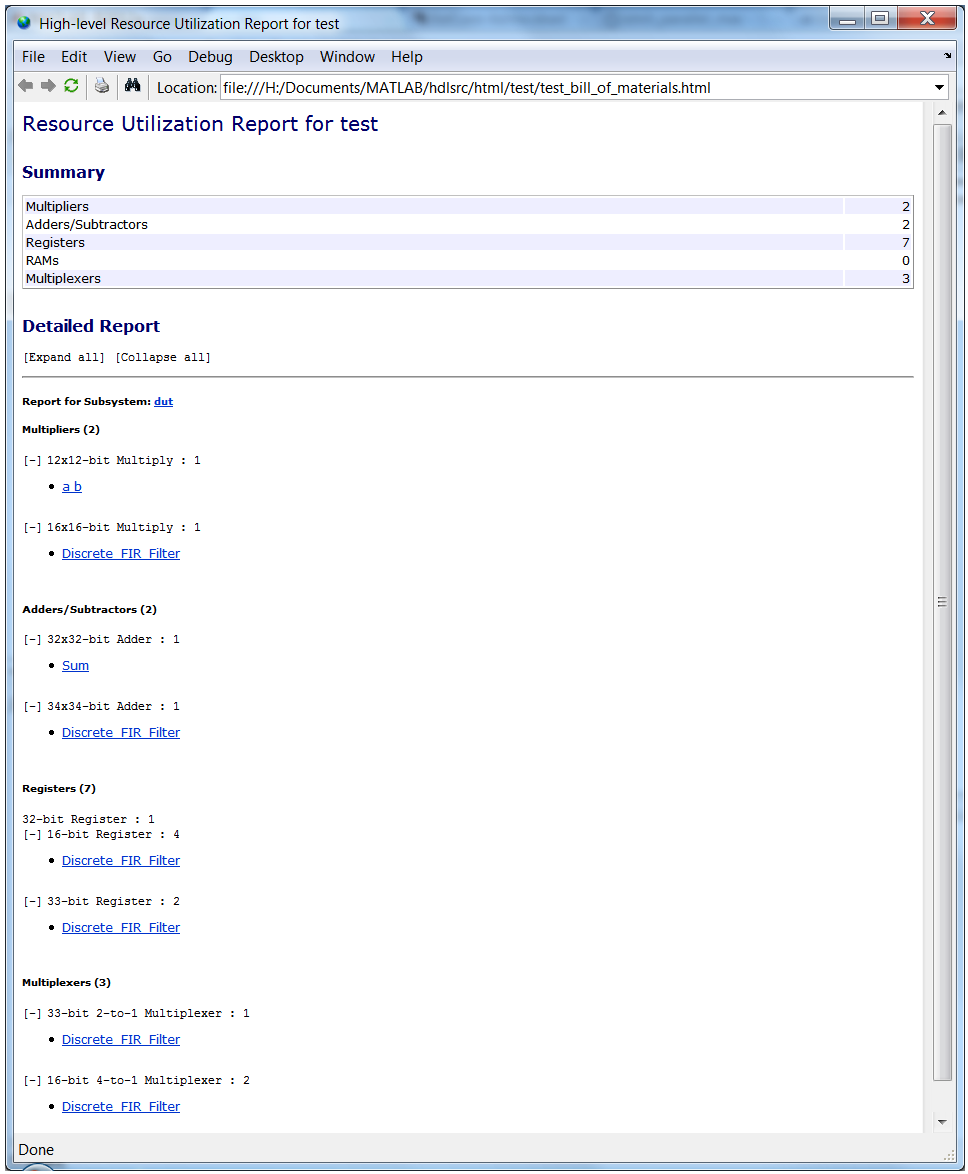
The `ClockInputs` parameter supports the values 'Single' and 'Multiple', where the default is 'Single'. In the default single-clock mode, the coder behavior is unchanged from previous releases.

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.



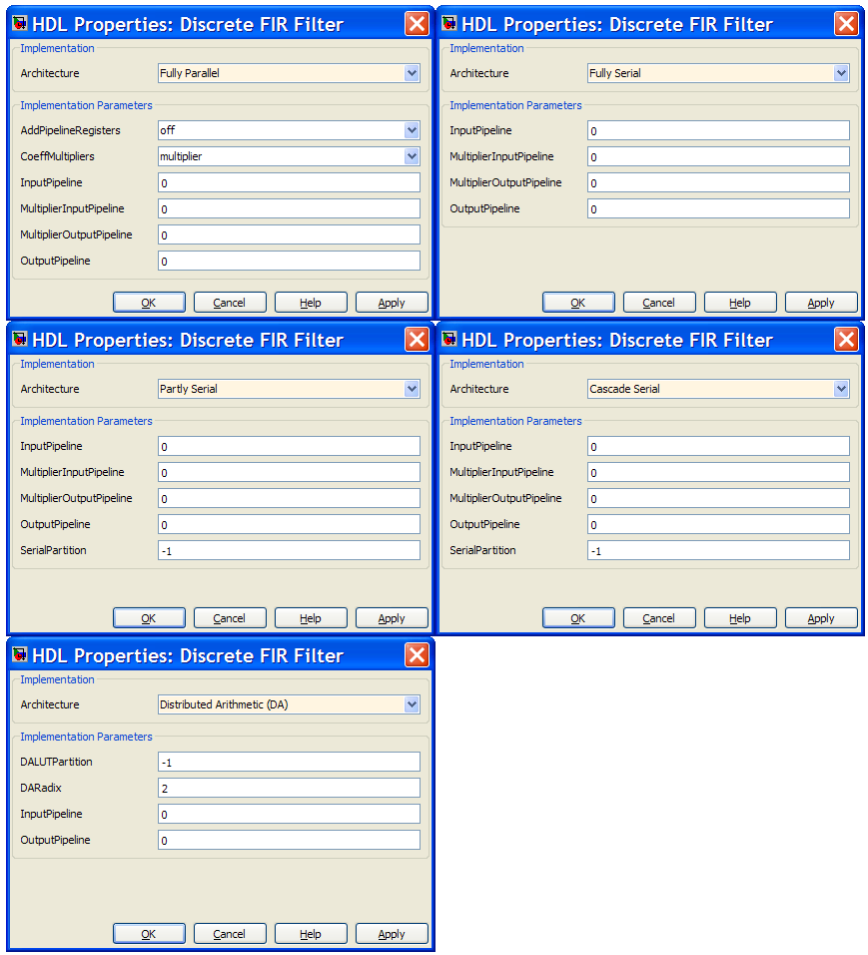

High-level Resource Utilization Report for test
File Edit View Go Debug Desktop Window Help
Location: file:///H:/Documents/MATLAB/hdlsrc/html/test/test_bill_of_materials.html
Resource Utilization Report for test
Summary
Multipliers 2
Adders/Subtractors 2
Registers 7
RAMs 0
Multiplexers 3
Detailed Report
[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
- Party 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.