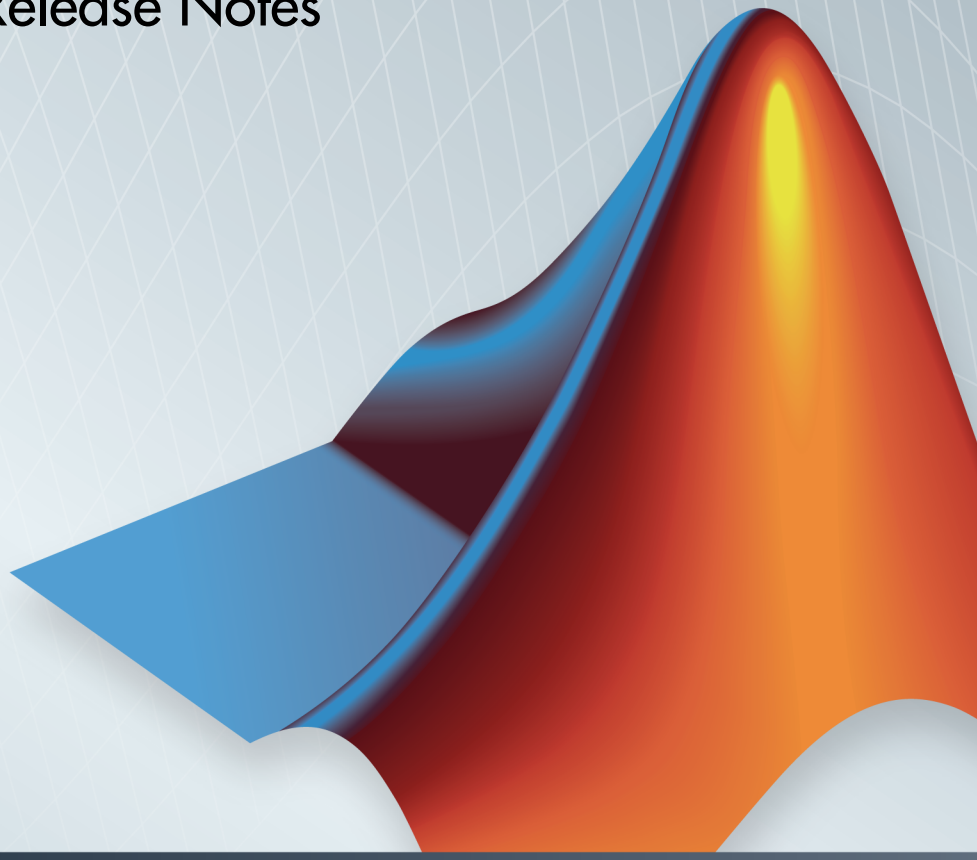
HDL Coder™ Release Notes



MATLAB®

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

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

HDL Coder™ Release Notes

Contents

R2014b

R2014a

R2013b

R2013a

R2012b

R2012a

R2014b

Clock-rate pipelining to optimize timing in multi-cycle paths

In the Simulink® to HDL workflow, for speed optimizations that insert pipeline registers, the coder identifies multi-cycle paths in your design and inserts pipeline registers at the clock rate instead of the data rate. When the optimization is in a slow-rate region or multi-cycle path of the design, clock rate pipelining enables the software to perform optimizations without adding extra latency, or by adding minimal latency. It also enables optimizations such as pipelining and floating-point library mapping inside feedback loops.

Clock-rate pipelining is enabled by default. You can disable clock-rate pipelining in one of the following ways:

- In the HDL Workflow Advisor, in the **HDL Code Generation > Set Code Generation Options > Set Advanced Options > Optimization** tab, select **Clock-rate pipelining**.
- At the command line, use `makehdl` or `hdlset_param` to set the `ClockRatePipelining` parameter to `off`.

For details, see “Clock-Rate Pipelining”.

Support for Xilinx Vivado

The HDL Coder™ software is now tested with Xilinx® Vivado® Design Suite 2013.4. You can:

- Generate a custom IP core for the Zynq®-7000 platform and automatically integrate it into a Vivado project for use with IP Integrator.
- Program FPGA hardware supported by Vivado using the HDL Workflow Advisor.
- Perform back-annotation analysis of your design.
- Generate synthesis scripts.

IP core generation for Altera SoC platform

You can generate a custom IP core with an AXI4 interface for the Altera® SoC platform.

HDL Coder can also insert your custom IP core into a predefined Qsys project to target the Altera Cyclone V SoC development kit or Arrow SoCKit development board. The

coder can connect the IP core to the ARM processor via the AXI interface within the project.

The software provides add-on support for Altera SoC hardware via the HDL Coder Support Package for Altera SoC Platform. For more details, see "HDL Coder Support Package for Altera SoC Platform".

Custom or legacy HDL code integration in the MATLAB to HDL workflow

You can use a black box System object™, `hdl.BlackBox`, to integrate custom HDL code into your design in the MATLAB® to HDL workflow. For example, you can integrate handwritten or legacy HDL code that you previously generated from MATLAB code or a Simulink model.

For an example that shows how to use `hdl.BlackBox`, see "Integrate Custom HDL Code Into MATLAB Design".

Custom HDL code for IP core generation from MATLAB

You can integrate custom HDL code, such as handwritten or legacy HDL code, into your design in the MATLAB to HDL IP core generation workflow. Use one or more `hdl.BlackBox` System objects in your MATLAB design, and add the HDL source files in the **Additional source files** field.

To learn how to use the **Additional source files** field, "Generate a Board-Independent IP Core from MATLAB".

Model reference as DUT for code generation

You can directly generate code for a model reference, without placing it in a Subsystem block. Previously, the code generation DUT had to be a Subsystem block.

Code generation for HDL optimized FFT/IFFT System object and HDL optimized Complex to Magnitude-Angle System object and block

You can generate code for the `dsp.HDLFFT` and `dsp.HDLIFFT` System objects, Complex to Magnitude-Angle HDL Optimized block, and `dsp.ComplexToMagnitudeAngle` System object, which are available in the DSP System Toolbox™.

Added features to HDL optimized FFT/IFFT blocks, including reduced latency

For details of the updates to the FFT HDL Optimized and IFFT HDL Optimized blocks, see the DSP System Toolbox release notes.

Compatibility Considerations

The FFT HDL Optimized and IFFT HDL Optimized blocks take fewer cycles to compute one frame of output than in previous releases. For instance, for the default 1024-point FFT, the latency in R2014a was 1589 cycles whereas in R2014b the latency is 1148. The latency is displayed on the block icon.

If you have manually matched latency paths in models using the R2014a version of the FFT HDL Optimized and IFFT HDL Optimized block, adjust the delay on those paths to accommodate the lower FFT latency.

Tunable parameter support for Gain and Constant blocks

The coder generates a top-level DUT port for each tunable parameter in your DUT that you use as the **Gain** parameter in a Gain block, or the **Constant value** parameter in a Constant block.

For details, see “Generate Code For Tunable Parameters”.

Code generation for Stateflow active state output

If you enable active state output to show child activity or leaf state activity for a Stateflow® block, the coder generates code for the active state output. See “Active State Output”.

Coding standards customization

If you enable HDL coding standard rule checking, you can enable or disable specific rules. You can specify rule parameters. For example, you can specify the maximum nesting depth for if-else statements. See `HDL Coding Standard Customization`.

HDL Designer script generation

You can now generate a lint tool script for Mentor Graphics® HDL Designer.

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".

Clock enable minimization for code generated from MATLAB designs

You can minimize clock enable logic in your generated code by setting the `MinimizeClockEnables` property of the `coder.HdlConfig` object to `true`, or by enabling the **Minimize clock enables** option in the HDL Workflow Advisor.

For details, see "Minimize Clock Enables".

HDL Block Properties dialog box shows only valid architectures

For each block supported for code generation, the HDL Block Properties dialog box **Architecture** drop-down list shows only the architectures that are valid for the block based on mask parameter settings. Previously, all architectures were available for selection regardless of mask parameter settings, and invalid settings caused errors during code generation.

HDL Reciprocal block with Newton-Raphson Implementation

The HDL Reciprocal block is available with Simulink. Use this block to implement division operations in models intended for HDL code generation. HDL Reciprocal has two Newton-Raphson HDL implementations, `ReciprocalNewton` and `ReciprocalNewtonSingleRate`. The new implementations use fewer hardware resources and can achieve higher clock frequency than Divide or Math Function HDL block implementations.

For the Divide and Math Function blocks, the names of the Newton-Raphson HDL block implementations have changed:

- `RecipNewton` is now `ReciprocalRsqrtBasedNewton`.
- `RecipNewtonSingleRate` is now `ReciprocalRsqrtBasedNewtonSingleRate`.

If you open a model from a previous release, HDL Coder automatically maps the `RecipNewton` and `RecipNewtonSingleRate` implementation names to `ReciprocalRsqrtBasedNewton` and `ReciprocalRsqrtBasedNewtonSingleRate`, respectively.

To learn about the HDL Reciprocal block, see HDL Reciprocal.

To learn about the `ReciprocalNewton` and `ReciprocalNewtonSingleRate` implementations, see HDL Reciprocal.

Serializer1D and Deserializer1D blocks

The following new blocks are available from the HDL Operations library for simulation and code generation:

- Serializer1D
- Deserializer1D

Additional blocks supported for code generation

The following blocks are now supported for code generation:

- Backlash
- Bus assignment
- Coulomb and Viscous Friction
- DC Blocker
- Dead Zone/Dead Zone Dynamic
- Discrete PID Controller
- Hit Crossing
- HDL Reciprocal
- Serializer1D/Deserializer1D
- Wrap to Zero

Traceable names for RAM blocks and port signals

When you generate code for a RAM block from the HDL Operations library, the name in the generated code reflects the name in the model. Similarly, port signal names in the generated code are inherited from your model.

For each RAM block of a particular size, the coder generates an HDL module. The module file name reflects the name and size of the RAM block or persistent variable in your design.

For example, suppose `DPRAM_foo` is the name of a Dual Port RAM block in your model. The generated code for the instance is:

```
u_DPRAM_foo : DualPortRAM_Wrapper_256x8b
```

The RAM module name and wrapper name also match the name of the Simulink block:

```
DualPortRAM_256x8b.vhd
DualPortRAM_Wrapper_256x8b.vhd
```

`for-generate` statements in generated VHDL code

When you generate VHDL® code for block architectures that use replicated structures, the coder generates `for-generate` statements for better readability. For example, VHDL code generated for the Add and Product blocks uses `for-generate` statements.

Composite user-defined System object support

You can generate code for user-defined System objects that contain child user-defined System objects.

System object output and update method support

You can generate code for the output and update methods in user-defined System objects that inherit from the `matlab.system.mixin.Nondirect` class.

RAM mapping for user-defined System object private properties

Private properties in user-defined System objects can map to RAM. For details, see “Implement RAM Using a Persistent Array or System object Properties”.

hdlram renamed to hdl.RAM

The `hdlram` System object has been renamed to `hdl.RAM` and is now available with MATLAB. Previously, `hdlram` required a Fixed-Point Designer™ license.

Compatibility Considerations

If you open a design that uses `hdlram`, the software displays a warning. For continued compatibility with future releases, replace instances of `hdlram` with `hdl.RAM`.

Target platform interface mapping information saved with model

For the IP core generation workflow, FPGA turnkey workflow, or Simulink Real-Time™ FPGA I/O workflow, when you map each of your DUT top-level ports to a platform interface, HDL Coder saves the interface mapping information as port properties in your model. The coder also saves workflow and target platform information with the model.

For DUT Inport and Outport blocks, you can set and view **IOInterface** and **IOInterfaceMapping** with the HDL Block Properties dialog box or `hdlset_param` and `hdlget_param`.

For an example that shows how to configure target platform interface settings, see “Save Target Hardware Settings in Model”.

Highlighting for feedback loops that inhibit optimizations

You can generate a MATLAB script that highlights feedback loops that may inhibit delay balancing or speed and area optimizations. The script highlights feedback loops in your original model and generated model.

You can also save the highlighting information in a MATLAB script.

For details, see “Find Feedback Loops”.

Optimizations available for conditional-execution subsystems

The following optimizations are now supported for enabled subsystems and triggered subsystems:

- Resource sharing
- Streaming
- Constrained overclocking
- Floating-point library mapping
- Hierarchy flattening

- Delay balancing
- Automatic iterative optimization

Variable pipelining in conditional MATLAB code

HDL code generation now supports variable pipelining inside conditional MATLAB code for the MATLAB to HDL workflow and the MATLAB Function block in the Simulink to HDL workflow.

2-D matrix types in HDL generated for MATLAB matrices

When you have matrices in your MATLAB code, you can generate 2-D matrices in HDL code. By default, the software generates HDL vectors with additional index computation logic, which can use more area in the synthesized hardware than HDL matrices.

To generate 2-D matrix types in HDL in the MATLAB to HDL workflow:

- In the HDL Workflow Advisor, in the **HDL Code Generation** > **Coding Style** tab, select **Use matrix types in HDL code**.
- At the command line, set the `UseMatrixTypesInHDL` property of the `coder.HdlConfig` object to `true`.

Previously, 2-D matrix types could be generated in HDL for the Simulink MATLAB Function block, but not in the MATLAB to HDL workflow.

Optimizations available with `UseMatrixTypesInHDL` for MATLAB Function block

When you enable 2-D matrix types in the generated HDL code, for the MATLAB to HDL workflow and Simulink to HDL workflow, speed and area optimizations are available. Previously, for the MATLAB Function block, the `UseMatrixTypesInHDL` parameter was incompatible with speed and area optimizations.

Validation model generation regardless of delay balancing results

When you enable the **Generate validation model** option, HDL Coder generates the validation model even if delay balancing is unsuccessful. In previous releases, if delay balancing was unsuccessful, the coder did not generate the validation model.

Documentation installation with hardware support package

Starting in R2014b, each hardware support package that you install comes with its own documentation. For a list of support packages available for HDL Coder, with links to documentation, see “HDL Coder Supported Hardware”.

Functionality Being Removed or Changed

Functionality	What Happens When You Use This Functionality	Use This Functionality Instead	Compatibility Considerations
HDL Streaming FFT	Still runs. The block is forwarded from hdldemolib to dsp.obselete.	FFT HDL Optimized block, which is available in the DSP System Toolbox.	This block will be removed in a future release.
HDL FFT	Still runs. This block is renamed “HDL Minimum Resource FFT”. It is forwarded from hdldemolib dsp.obselete.	FFT HDL Optimized block, which is available in the DSP System Toolbox.	This block will be removed in a future release.

R2014a

Code generation for enumeration data types

You can generate code for Simulink, MATLAB, or Stateflow enumerations within your design. In the current release, you cannot generate code if your design uses enumerations at the top-level DUT ports.

To learn more about code generation support for enumerations in Simulink designs, see Enumerations.

To learn more about code generation support for enumerations in MATLAB designs, see Data Types and Scope.

ZC706 target for IP core generation and integration into Xilinx EDK project

You can target the Xilinx Zynq-7000 AP ZC706 Evaluation Board for IP core generation and Xilinx EDK project integration. After you install the HDL Coder Support Package for Xilinx Zynq-7000 Platform, ZC706 hardware support is available.

Automatic iterative clock frequency optimization

You can use the `hdlcoder.optimizeDesign` function to achieve either your target clock frequency or a maximum clock frequency. Based on your clock frequency goal and target device, the software iteratively generates and synthesizes code, retrieves back annotation data, and inserts delays into your Simulink model to break the critical path.

To learn more, see Automatic Iterative Optimization.

Code generation for FFT HDL Optimized and IFFT HDL Optimized blocks

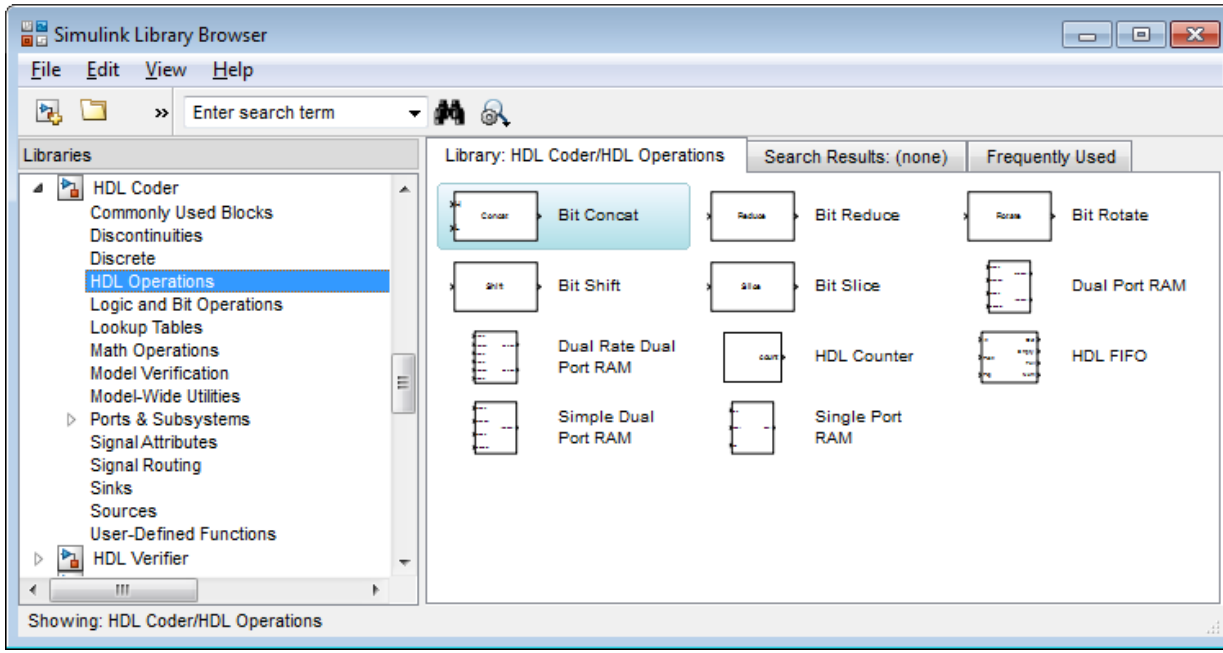
You can generate code for the FFT HDL Optimized and IFFT HDL Optimized blocks, which are available in the DSP System Toolbox.

HDL block library in Simulink

The HDL Coder library, which contains blocks supported for HDL code generation, is available in Simulink. When you create a model using the HDL Coder library, the blocks are preconfigured with settings suitable for code generation.

The HDL Operations library, previously called `hdldemolib`, is available in Simulink as part of the HDL Coder library. Previously, the HDL Operations blocks were available only with an HDL Coder license.

To view the HDL Operations block library from the Simulink Library Browser, open the **HDL Coder** folder and select **HDL Operations**.



Bus support improvements

You can generate code for designs that contain:

- DUT ports connected to buses.
- Buses that are not defined with a bus object.
- Nonvirtual buses.

To learn more, see Buses.

Variant Subsystem support for configurable models

You can generate code for designs containing Variant Subsystem blocks. Using Variant Subsystem blocks enables you to explore and generate code for different component implementations and design configurations.

Trigger signal can clock triggered subsystems

You can generate code that uses the trigger signals in Triggered Subsystem blocks as clocks. Using triggers as clocks enables you to partition your design into different clock regions in the generated code, but can cause a timing mismatch during testbench simulation.

For details, see Use Trigger As Clock in Triggered Subsystems.

2-D matrix types in code generated for MATLAB Function block

You can now generate 2-D matrices in HDL code when you have MATLAB matrices in a MATLAB Function block. By default, the software generates HDL vectors with additional index computation logic, which can use more area in the synthesized hardware than HDL matrices.

For details, see UseMatrixTypesInHDL.

64-bit data support

You can generate code for `uint64` and `int64` data types in MATLAB code, both in the MATLAB-to-HDL workflow and for the MATLAB Function block in the Simulink-to-HDL workflow.

MATLAB Function block ports must use `sfix64` or `ufix64` types for 64-bit data, because `uint64` and `int64` are not yet supported in Simulink.

HDL code generation from MATLAB System block

The MATLAB System block, which you use to include System objects in Simulink models, now supports HDL code generation.

For details, see MATLAB System.

System object methods in conditional code

HDL code generation now supports System object `step` method calls inside conditional code regions.

`Persistent` keyword not needed in HDL code generation

If your MATLAB code includes a System object that does not have states, you do not need to include the `persistent` keyword for HDL code generation.

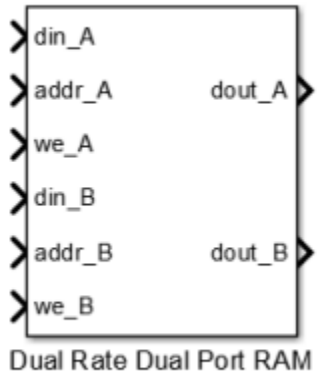
For details, see Limitations of HDL Code Generation for System Objects.

Dual Rate Dual Port RAM block

A new block, Dual Rate Dual Port RAM, is available for simulation and code generation.

The Dual Rate Dual RAM supports two simultaneous read or write accesses at two Simulink rates. When you generate code, the Dual Rate Dual Port RAM block infers a dual-clock dual-port RAM in most FPGAs.

To view the block, open the HDL Operations block library.



For more information about the block, see Dual Rate Dual Port RAM. For HDL code generation details, see Dual Rate Dual Port RAM.

Negative edge clocking

You can clock your design on the falling edge of the clock.

To generate code that clocks your design on the negative edge of the clock, in the Configuration Parameters dialog box, for **HDL Code Generation > Global Settings > Clock Edge**, select **Falling edge**.

Alternatively, at the command line, set the `ClockEdge` property to `'Falling'` using `makehdl` or `hdlset_param`.

For details, see ClockEdge.

Bidirectional port specification

You can specify bidirectional ports for Subsystem blocks that have Architecture set to `BlackBox`. In the FPGA Turnkey workflow, you can use the bidirectional ports to connect to external RAM.

In the generated code, the ports have the Verilog® or VHDL `inout` keyword. However, Simulink does not support bidirectional ports, so you cannot simulate the bidirectional behavior in Simulink.

To learn more, see Specify Bidirectional Ports.

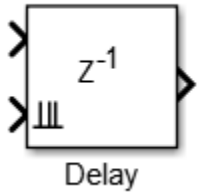
Port names in generated code match signal names

You can use the **Icon display** block parameter on Inport and Outport blocks to make your code more readable. When you set the **Icon display** parameter to **Signal name**, **Port number**, or **Port number and signal name**, the port names in the generated code match the display names of the connected signals.

Additional blocks and block implementations supported for code generation

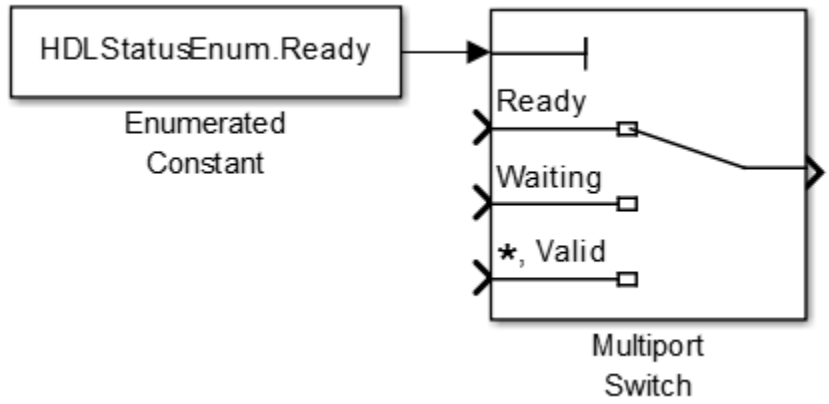
The following blocks and block implementations are now supported for code generation:

- Sine, Cosine
- Enumerated Constant
- Delay with **External reset** set to `Level`.



- Multiport Switch with enumerated type at control input.

  You can set **Data port order** to **Specify indices**, and enter enumeration values for the **Data port indices**. For example, you can connect the Enumerated Constant block to the Multiport Switch control port and use the enumerated types as data port indices.



The HDL FIFO block no longer requires a DSP System Toolbox license. The HDL FIFO block is available in the HDL Operations library.

Errors instead of warnings for blocks not supported for code generation

If your design contains blocks or block architectures that are not supported for HDL code generation, the software shows an error and does not generate code. Previously, the

software showed a warning, but still generated code, with black box interfaces for the unsupported blocks or block architectures.

Compatibility Considerations

If you want to generate code for models containing unsupported blocks or block architectures, you must **Comment out** the unsupported blocks in Simulink.

ModelReference default architecture for Model block

The Model block default architecture is `ModelReference`. Previously, the default architecture was `BlackBox`.

Compatibility Considerations

When you open a model created in a previous release, a Model block in that design changes architecture from `BlackBox` to `ModelReference` if all the HDL block properties are set to default settings.

To keep the `BlackBox` architecture for Model blocks, use one of the following workarounds:

- Open the model using the current release, specify the `BlackBox` architecture for the affected Model blocks, and save the model.
- Open the model using a previous release, specify a nondefault setting for each Model block, and save the model.

Reset port optimization

The coder does not generate a top-level reset port when the code generation subsystem does not contain resettable delays or blocks.

To generate code without a top-level reset port:

- Set the `ResetType` HDL block parameter to `none` for all blocks in the DUT with the `ResetType` parameter.
- For MATLAB Function blocks in your DUT, do not enable block level HDL optimizations, which insert resettable registers.

For details, see ResetType.

Reset for timing controller

You can generate a reset port for the timing controller, which generates the clock, clock enable, and reset signals in a multirate DUT.

To generate a reset port for the timing controller, set the `TimingControllerArch` property to `resettable` using `makehdl` or `hdlset_param`.

To learn more, see Generate Reset for Timing Controller.

Synthesis attributes for multipliers

You can now generate code that includes synthesis attributes to specify multipliers in your design that you want to map to DSPs or logic in hardware. If you specify resource sharing, the software does not share multipliers that have different synthesis attribute settings.

For Xilinx targets, the generated code uses the `use_dsp48` attribute. For Altera targets, the generated code uses the `multstyle` attribute.

For details, see DSPStyle.

Ascent Lint script generation

You can now generate a lint tool script for Real Intent Ascent Lint.

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.

RAM mapping scheduler improvements

The RAM mapping scheduling algorithm now minimizes overclocking when your MATLAB code maps to multiple RAMs. In addition, multiple persistent variables with cyclic read-write dependencies can now map to RAM.

Incremental code generation and synthesis

In the Simulink-to-HDL workflow, and hardware-software codesign workflow, HDL Coder does not rerun code generation or synthesis tasks unless you have changed your model or other hardware-related project settings. You can save time when you want to regenerate HDL code or FPGA programming files without changing your model, code generation options, or hardware target.

Similarly, in the hardware and software codesign workflow, when you modify the embedded software part of your design without changing the hardware part, HDL Coder does not rerun HDL code generation or synthesis tasks.

When the coder skips code generation or synthesis tasks, the HDL Workflow Advisor shows a message. The message contains a link you can click to force the coder to rerun the task.

Custom HDL code for IP core generation

You can integrate custom HDL code into your design in the Simulink-to-HDL IP core generation workflow. You can integrate handwritten or legacy HDL code into an IP core that you generate from a Simulink model.

To include custom HDL code in your IP core design, use one or more Model or Subsystem blocks with Architecture set to `BlackBox`. Use the **Additional source files** field in the HDL Workflow Advisor to specify corresponding HDL file names.

For details of the IP core generation workflow, see Generate a Custom IP Core from Simulink.

Performance-prioritized retiming

When you enable distributed pipelining, you can specify a priority for **Distributed pipelining priority**: **Numerical integrity**, or **Performance**. In the previous release, the distributed pipelining algorithm prioritized numerical integrity.

For details, see DistributedPipeliningPriority.

Retiming without moving user-created design delays

You can use the **Preserve design delays** option to prevent distributed pipelining from moving design delays in your Simulink or MATLAB design. If you specify **Preserve design delays**, distributed pipelining does not move the following design delays:

- Persistent variable in MATLAB code, a MATLAB Function block, or a Stateflow Chart
- Unit Delay block
- Integer Delay block
- Memory block
- Delay block from DSP System Toolbox
- `dsp.Delay` System object from DSP System Toolbox

For details, see PreserveDesignDelays.

Resource sharing factor can be greater than number of shareable resources

With the resource sharing area optimization, the software shares the maximum number of shareable resources within your overclocking constraints, even if the sharing factor that you specify is not an integer divisor of the number of shareable resources. This capability can increase resource sharing, and therefore reduce area.

For example, if your subsystem has 11 multipliers, and you set **SharingFactor** to 4, the coder can implement your design with 3 multipliers: 2 multipliers shared 4 ways, and 1 multiplier shared 3 ways. In the previous release, the coder implemented the design with 5 multipliers: 2 multipliers shared 4 ways, and 3 unshared multipliers. The resulting implementation requires overclocking by a factor of 4.

To learn more, see Resource Sharing For Area Optimization.

Reduced area with multirate delay balancing

When the coder balances delays in a multirate model, it now inserts a single delay at the transition from a much faster rate to a slow rate, and passes through the data samples aligned with the slow rate. Previously, the coder inserted a large number of delays at the faster rate.

Serializer-deserializer and multiplexer-demultiplexer optimization

The coder removes back-to-back serializer-deserializer and multiplexer-demultiplexer pairs introduced by the implementation of optimizations such as resource sharing and streaming. This results in more area-efficient HDL code.

Synthesis and simulation tool addition and detection after opening HDL Workflow Advisor

In the Simulink-to-HDL workflow, you can set up and add a synthesis tool without having to close and reopen the HDL Workflow Advisor. In the HDL Workflow Advisor, in the **Set Target > Set Target Device and Synthesis Tool** task, click **Refresh** to detect and add the new tool.

You can also set up and add a simulation tool after creating a MATLAB-to-HDL project without having to close and reopen the project. In the HDL Workflow Advisor, in the **HDL Verification > Verify with HDL Test Bench** task, click **Refresh list** to detect and add the new tool.

Automatic C compiler setup

In earlier releases, to set up a compiler to accelerate test bench simulation for MATLAB algorithms, you were required to run `mex -setup`. Now, the code generation software automatically locates and uses a supported installed compiler. You can use `mex -setup` to change the default compiler. See Changing Default Compiler.

xPC Target is Simulink Real-Time

The **xPC Target FPGA I/O** workflow is now called the **Simulink Real-Time FPGA I/O** workflow. This change reflects the xPC Target™ product name change to Simulink Real-Time. For details about the product name change, see New product that combines the functionality of xPC Target and xPC Target Embedded Option.

Updates to supported software

HDL Coder has been tested with:

- Xilinx ISE 14.6

- Altera Quartus II 13.0 SP1

For a list of supported third-party tools and hardware, see Supported Third-Party Tools and Hardware.

Functionality Being Removed or Changed

You cannot save a model that uses an attached control file to apply HDL model or block parameters.

Since the R2010a release, if you open a model that uses a control file, the software shows a warning, and updates the model by applying the HDL parameters to your model and removing the control file. For continued compatibility with future releases, save the updated model.

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
hdlapplycontrolfile	Still runs	hdlset_param, hdlget_param	Do not use control files for model or block configuration. Instead, use hdlset_param and hdlget_param to configure your model.
hdlnewblackbox	Still runs	hdlset_param, hdlget_param	Do not use control files for model or block configuration. Instead, use hdlset_param and hdlget_param to configure your model.
hdlnewcontrol	Still runs	hdlset_param, hdlget_param	Do not use control files for model or block configuration. Instead, use hdlset_param and

Functionality	What Happens When You Use This Functionality	Use This Instead	Compatibility Considerations
			hdlget_param to configure your model.
hdlnewcontrolfile	Still runs	hdlset_param, hdlget_param	Do not use control files for model or block configuration. Instead, use hdlset_param and hdlget_param to configure your model.
hdlnewforeach	Still runs	hdlset_param, hdlget_param	Do not use control files for model or block configuration. Instead, use hdlset_param and hdlget_param to configure your model.

R2013b

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 Simulink Real-Time 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

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

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

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

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

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

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	`RecipSqrtNewtonSingleR`	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

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

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

Compatibility Considerations

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

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`.

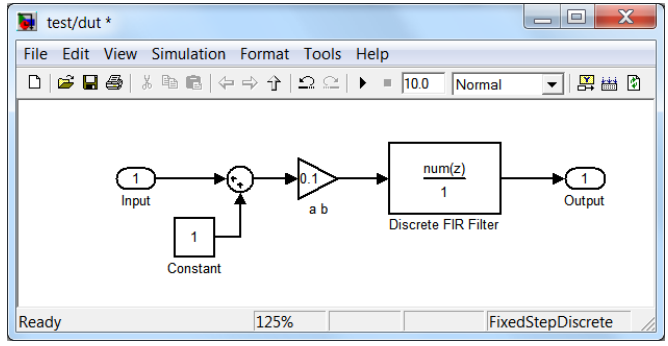
When you use single-clock mode, HDL code generated from multirate models uses a single master clock that corresponds to the base rate of the DUT. When you use multiple-clock mode, HDL code generated from multirate models use one clock input for each rate in the DUT. The number of timing controllers generated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

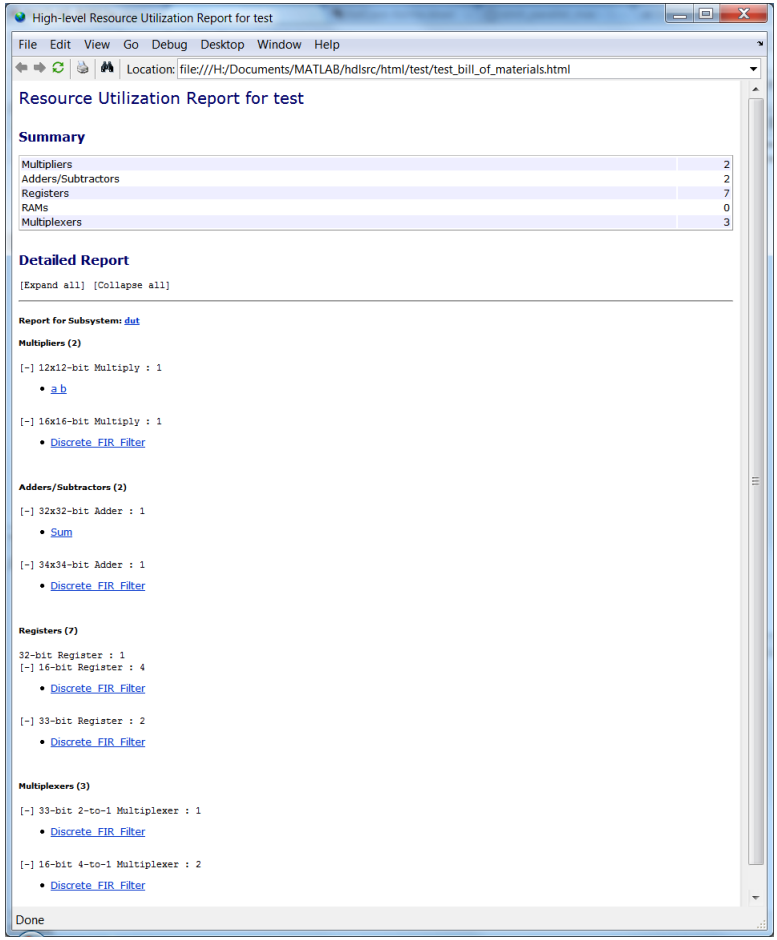
The `ClockInputs` parameter supports the values 'Single' and 'Multiple', where the default is 'Single'. In the default single-clock mode, the coder behavior is unchanged from previous releases.

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.





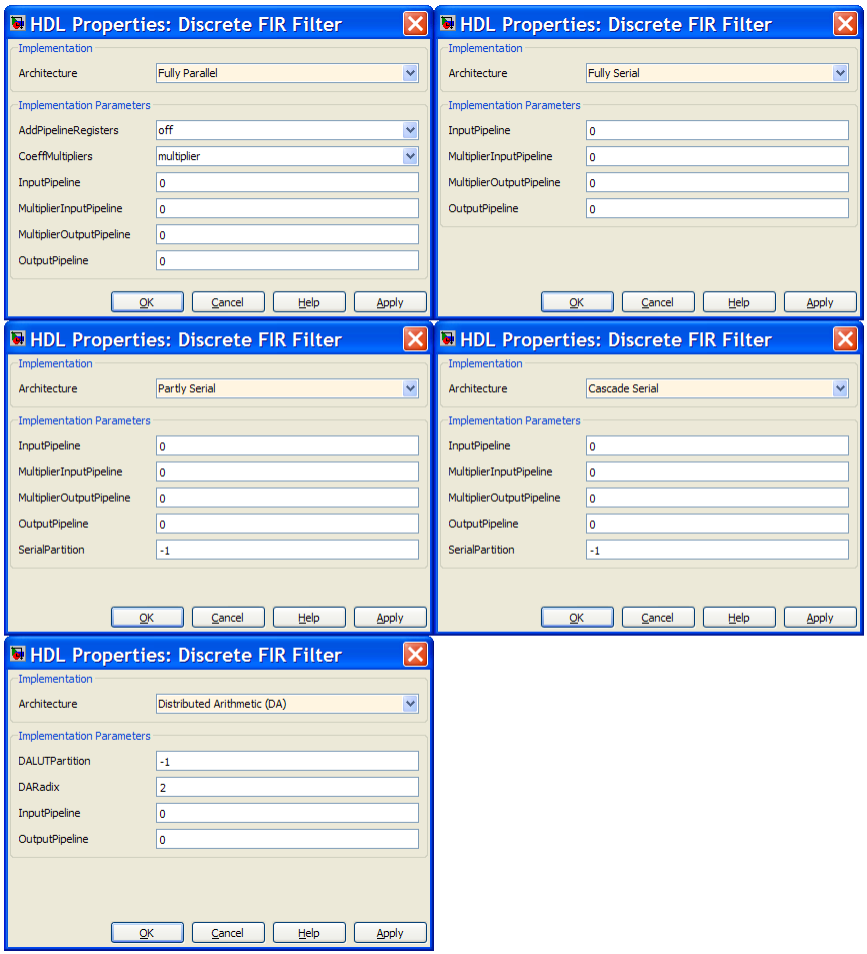
HDL Block Properties Interface Allows Choice of Filter Architecture

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.