Simulink® HDL Coder™ Release Notes

How to Contact The 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.

Simulink® HDL Coder™ Release Notes

Contents

Summary by Version

This table provides quick access to what's new in each version. For clarification, see "Using Release Notes" on page 1.

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Latest Version V1.4 (R2008b)	Yes Details	Yes Summary	Bug Reports	Printable Release Notes: PDF Current product documentation
V1.3 (R2008a)	Yes Details	Yes Summary	Bug Reports	No
V1.2 (R2007b)	Yes Details	Yes Summary	Bug Reports	No
V1.1 (R2007a)	Yes Details	No	Bug Reports	No

Using Release Notes

Use release notes when upgrading to a newer version to learn about:

- New features
- Changes
- Potential impact on your existing files and practices

Review the release notes for other MathWorks™ products required for this product (for example, MATLAB® or Simulink®) for enhancements, bugs, and compatibility considerations that also might impact you.

If you are upgrading from a software version other than the most recent one, review the release notes for all interim versions, not just for the version you are installing. For example, when upgrading from V1.0 to V1.2, review the release notes for V1.1 and V1.2.

What's in the Release Notes

New Features and Changes

- New functionality
- Changes to existing functionality

Version Compatibility Considerations

When a new feature or change introduces a reported incompatibility between versions, the **Compatibility Considerations** subsection explains the impact.

Compatibility issues reported after the product is released appear under Bug Reports at the MathWorks Web site. Bug fixes can sometimes result in incompatibilities, so you should also review the fixed bugs in Bug Reports for any compatibility impact.

Fixed Bugs and Known Problems

The MathWorks offers a user-searchable Bug Reports database so you can view Bug Reports. The development team updates this database at release time and as more information becomes available. This includes provisions for any known workarounds or file replacements. Information is available for bugs existing in or fixed in Release 14SP2 or later. Information is not available for all bugs in earlier releases.

Access Bug Reports using your MathWorks Account.

About Functions and Properties Being Removed

This section lists functions or properties removed or in the process of being removed. Functions and properties typically go through several stages across multiple releases before being completely removed. This provides time for you to make adjustments to your code.

- Announcement — The Release Notes announce the planned removal, but there are no functional changes; the function runs as it did before.
- Warning — When you run the function, it displays a warning message indicating it will be removed in a future release; otherwise the function runs as it did before.
- Error — When you run the function, it produces an error. The error message indicates the function was removed and suggests a replacement function, if one is available.
- Removal — When you run the function, it fails. The error message is the standard message when MATLAB does not recognize an entry.

Functions and properties might be in a stage for one or more releases before moving to another stage. Functions and properties are listed in the Functions and Properties Being Removed section only when they enter a new stage and their behavior changes. For example, if a function displayed a warning in the previous release and errors in this release, it appears on the list. If it continues to display a warning, it does not appear on the list because there was no change between the releases.

Not all functions and properties go through all stages. For example, a function's impending removal might not be announced, but instead, the first notification might be that the function displays a warning.

The Release Notes include actions you can take to mitigate the effects of function or property removal, such as adapting your code to use a replacement function.

Version 1.4 (R2008b) Simulink HDL Coder Software

This table summarizes what's new in Version 1.4 (R2008b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Bug Reports	Printable Release Notes: PDF Current product documentation

New features and changes introduced in this version are:

- "New hdldemolib Blocks Support FFT, HDL Counter, and Bitwise Operators" on page 5
- "Additional Simulink Blocks Supported for HDL Code Generation" on page 6
- "Complex Signals Supported for Additional Blocks" on page 7
- "Code Annotation Support" on page 8
- "New Constant Block Implementation Indicates Hi-Z or Unknown States" on page 8
- "New Test Bench Reference Postfix Option" on page 9
- "New Default HDL Implementations for Selected Blocks" on page 11
- "Default Entity Conflict Postfix Changed" on page 12
- "New DistributedPipelining Implementation Parameter for Embedded MATLAB Function Blocks and Stateflow Charts" on page 12
- "Coefficient Multiplier Optimization for Digital Filter, FIR Decimation, and FIR Interpolation Filters" on page 13
- "hdlnewblackbox Function Generates Black Box Control Statements" on page 14

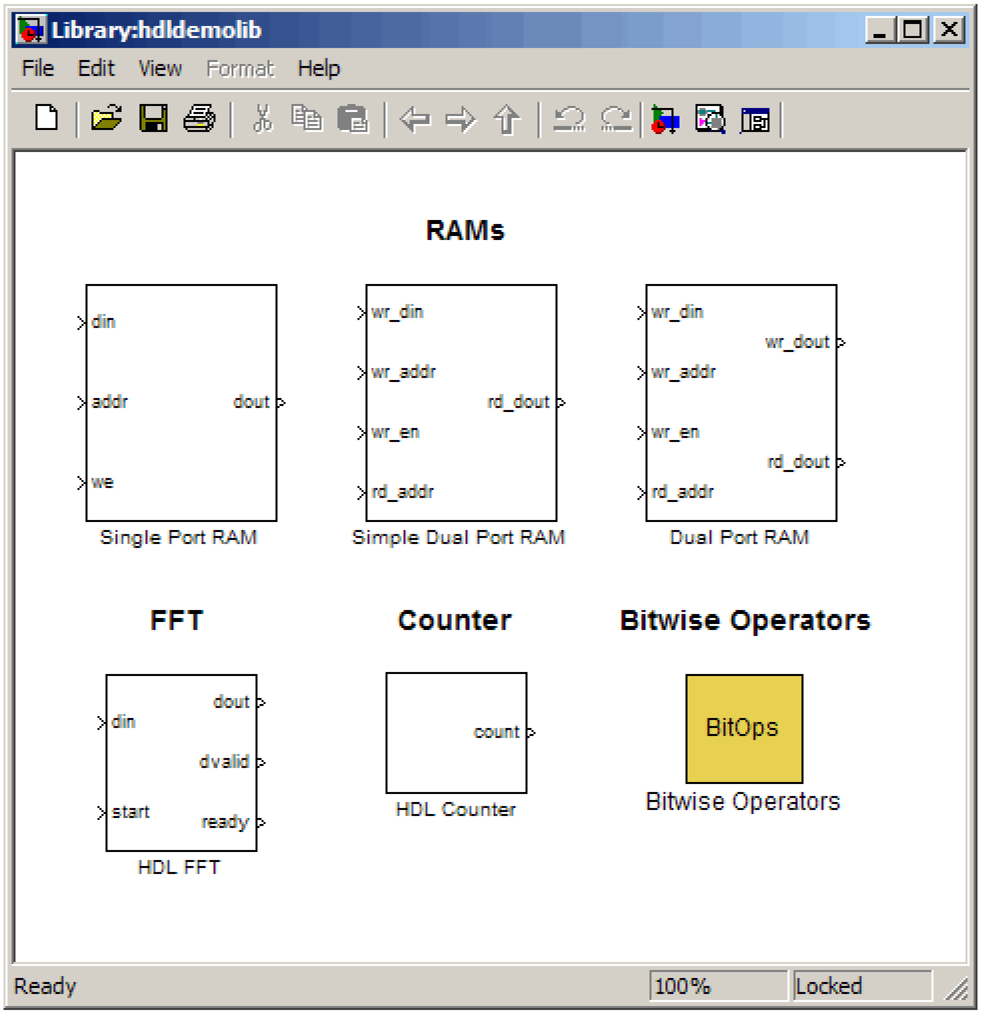
- "hdlnewcontrolfile Function Optionally Returns Result to String" on page 15
- "-novopt Flag Added to Default Simulation Command in Generated Compilation Scripts" on page 15

New hdldemolib Blocks Support FFT, HDL Counter, and Bitwise Operators

The `hdldemolib` library now includes HDL-specific block implementations supporting simulation and code generation for:

- Counter with count-limited and free-running modes (see "HDL Counter" in the Simulink® HDL Coder™ documentation)
- Minimum resource FFT (see "HDL FFT" in the Simulink HDL Coder documentation)
- Bitwise operations, including bit slice, bit reduction, bit concatenation, bit shift, and bit rotation (see "Bitwise Operators" in the Simulink HDL Coder documentation)

The following figure shows the `hdldemolib` library window. See "The hdldemolib Block Library" in the Simulink HDL Coder documentation for more information about the library.



Additional Simulink Blocks Supported for HDL Code Generation

The coder now supports the following blocks for HDL code generation:

- Signal Processing Blockset/Multirate Filters/CIC Interpolation

- Signal Processing Blockset/Multirate Filters/FIR Interpolation

  (See the demo “Digital Down Converter for HDL Code Generation” for an example of the use of this block.)
- Signal Processing Blockset/Filtering /Adaptive Filters/LMS Filter

  (See the demo “Adaptive Noise Canceler with LMS Filter” for an example of the use of this block.)
- simulink/Logic and Bit Operations/Extract Bits
- simulink/Math Operations/Math Function (now supports `hermitian`, and `transpose` functions for HDL code generation)
- simulink/Model-Wide Utilities/DocBlock
- Stateflow® Truth Table

In addition, several HDL-specific block implementations have been added to the `hdldemolib` library. See “New hdldemolib Blocks Support FFT, HDL Counter, and Bitwise Operators” on page 5.

See “Summary of Block Implementations” in the Simulink HDL Coder documentation for a complete listing of blocks that are currently supported for HDL code generation.

Complex Signals Supported for Additional Blocks

In the previous release, the coder introduced support for use of complex signals with a limited set of blocks. In R2008b, the coder supports complex signals for these additional blocks:

- dspadpt3/LMS Filter
- dspsigattribs/Frame Conversion
- dspsigops/Delay (`DSPDelayHDLEmission` implementation)
- hdldemolib/Dual Port RAM
- hdldemolib/Simple Dual Port RAM
- hdldemolib/Single Port RAM
- hdldemolib/HDL FFT

- simulink/Commonly Used Blocks/Relational Operator (~= and == operators only)
- simulink/Discrete/Memory
- simulink/Discrete/Zero-Order Hold
- simulink/Logic and Bit Operations/Compare To Constant
- simulink/Logic and Bit Operations/Compare To Zero
- simulink/Lookup Tables/Lookup Table (`LookupHDLInstantiation` implementation)
- simulink/Math Operations/Assignment
- simulink/Math Operations/Math Function (`hermitian`, `transpose`)
- simulink/Signal Attributes/Signal Specification

See "Blocks That Support Complex Data" in the Simulink HDL Coder documentation for a complete listing of blocks that support complex signals.

Code Annotation Support

The coder now lets you add text annotations to generated code, in the form of comments. There are two ways to add annotations to your code:

- Enter text directly on the block diagram as Simulink annotations.
- Place a DocBlock at the desired level of your model and enter text comments.

See "Annotating Generated Code" in the Simulink HDL Coder documentation for further information.

New Constant Block Implementation Indicates Hi-Z or Unknown States

The coder now supports an implementation for the built-in/Constant block (`hdldefaults.ConstantSpecialHDLEmission`), which you can use to indicate when a constant signal is in high-impedance (`'Z'`) or unknown (`'X'`) state. The implementation provides the `{Value}` parameter to indicate the state, as follows:

- `{Value, 'Z'}`: If the signal is in a high-impedance state, the Constant block emits the character `'Z'` for each bit in the signal. For example, for a 4-bit signal, `'ZZZZ'` would be emitted.

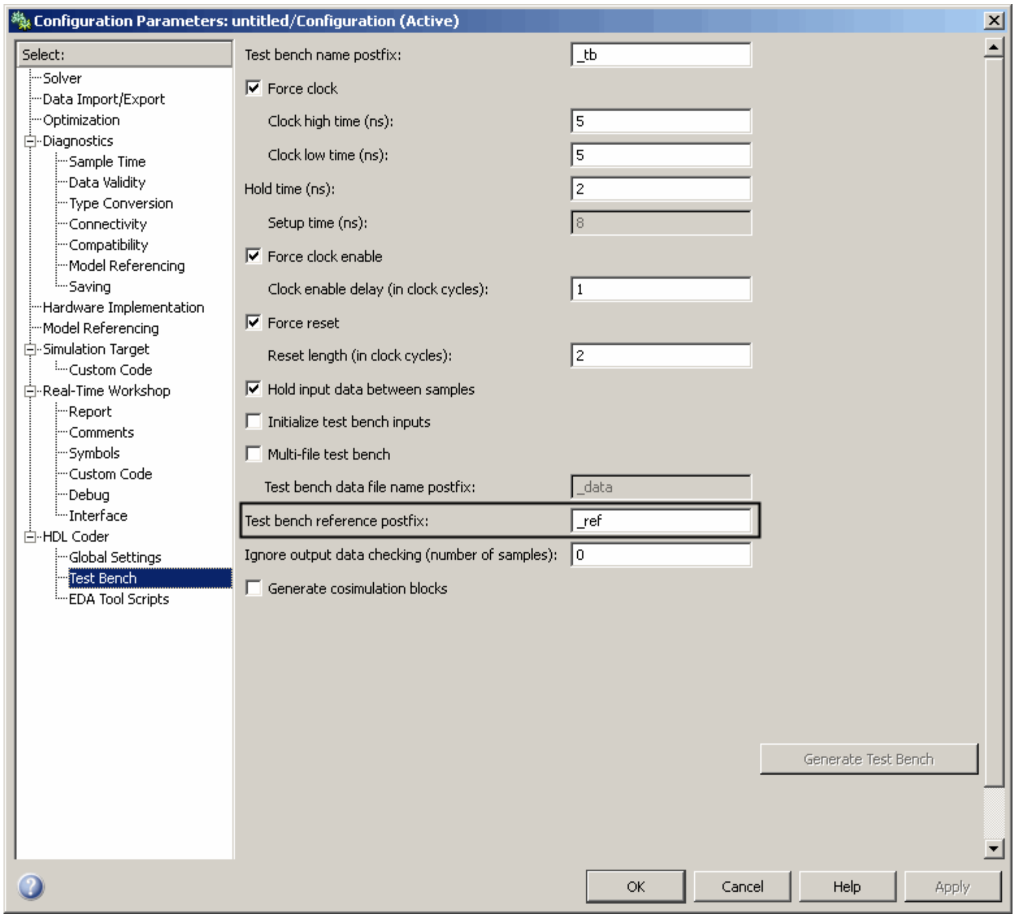
  `{Value, 'Z'}` is the default value for this implementation.
- `{Value, 'X'}`: If the signal is in an unknown state, the Constant block emits the character `'X'` for each bit in the signal. For example, for a 4-bit signal, `'XXXX'` would be emitted.

`hdldefaults.ConstantSpecialHDLEmission` does not support the `double` data type.

See also “Blocks with Multiple Implementations” in the Simulink HDL Coder documentation.

New Test Bench Reference Postfix Option

The new **Test bench reference postfix** option (shown in the following figure) lets you customize the names of reference signals generated in test bench code by specifying a string to be appended to reference signal names. The default string is`'_ref'`.


Configuration Parameters: untitled/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
  - Sample Time
  - Data Validity
  - Type Conversion
  - Connectivity
  - Compatibility
  - Model Referencing
  - Saving
- Hardware Implementation
- Model Referencing
- Simulation Target
  - Custom Code
- Real-Time Workshop
  - Report
  - Comments
  - Symbols
  - Custom Code
  - Debug
  - Interface
- HDL Coder
  - Global Settings
  - Test Bench
  - EDA Tool Scripts

Test bench name postfix:	_tb
[✓] Force clock	
Clock high time (ns):	5
Clock low time (ns):	5
Hold time (ns):	2
Setup time (ns):	8
[✓] Force clock enable	
Clock enable delay (in clock cycles):	1
[✓] Force reset	
Reset length (in clock cycles):	2
[✓] Hold input data between samples	
[ ] Initialize test bench inputs	
[ ] Multi-file test bench	
Test bench data file name postfix:	_data
Test bench reference postfix:	_ref
Ignore output data checking (number of samples):	0
[ ] Generate cosimulation blocks	

If you generate test bench code via the `makehdltb` function, use the `Testbenchreferencepostfix` property (see TestBenchReferencePostFix in the in the Simulink HDL Coder documentation) to specify the postfix string.

New Default HDL Implementations for Selected Blocks

The default HDL implementations for certain blocks has been changed. The following table lists these blocks, as well as their new default implementations and previous default implementations. All listed implementation classes belong to the package `hdldefaults`.

Block	Default Implementation Before Release R2008b	New Default Implementation
simulink/Commonly Used Blocks/Data Type Conversion	DataTypeConversionHDLEmission	DataTypeConversionRTW
simulink/Commonly Used Blocks/Product	ProductLinearHDLEmission	ProductRTW
simulink/Math Operations/Divide	ProductLinearHDLEmission	ProductRTW
simulink/Math Operations/Product of Elements	ProductLinearHDLEmission	ProductRTW
simulink/Commonly Used Blocks/Sum	SumLinearHDLEmission	SumRTW
simulink/Math Operations/Add	SumLinearHDLEmission	SumRTW
simulink/Math Operations/Sum of Elements	SumLinearHDLEmission	SumRTW
simulink/Math Operations/Subtract	SumLinearHDLEmission	SumRTW
simulink/Commonly Used Blocks/Unit Delay	UnitDelayHDLEmission	UnitDelayRTW
simulink/Math Operations/MinMax	MinMaxTreeHDLEmission	MinMaxTree
dspstat3/Maximum	MinMaxTreeHDLEmission	MinMaxTree
dspstat3/Minimum	MinMaxTreeHDLEmission	MinMaxTree

Compatibility Considerations

If your models use default HDL block implementations for the affected blocks, the coder will now default to the new implementations. The new implementations are compatible with the previous implementations and will produce identical results.

The older implementations for the listed blocks will be supported for a limited number of future releases. If your control files explicitly reference the previous default implementation for any of the affected blocks, the coder will continue to use the referenced implementation. You should consider removing or changing such references in your control files to use the new implementations.

Default Entity Conflict Postfix Changed

The default value for the **Entity conflict postfix** property (and the corresponding CLI property, `EntityConflictPostfix`) has been changed from `'_entity'` to `'_block'`.

Compatibility Considerations

If your models or scripts rely on the previous default value (`'_entity'`) for the **Entity conflict postfix** property, you will need to explicitly set the property value to `'_entity'`.

New DistributedPipelining Implementation Parameter for Embedded MATLAB Function Blocks and Stateflow Charts

In the previous release, the coder introduced automatic pipeline insertion, a special optimization for HDL code generated from Embedded MATLAB™ Function blocks or Stateflow charts. This optimization was enabled implicitly by specifying the `{'OutputPipeline', nStages}` parameter in a control file for these blocks.

In the current release, the new `DistributedPipelining` parameter lets you explicitly enable or disable pipeline insertion, independently from the `OutputPipeline` parameter. The control file listed in the following example specifies two pipeline registers, with `DistributedPipelining` enabled.

```
function c = pipeline_control

c = hdlnewcontrol(mfilename);

c.forEach('*',...
    'eml_lib/Embedded MATLAB Function', {},...
 'hdlstateflow.StateflowHDLInstantiation', {'OutputPipeline', 2, 'DistributedPipelining', 'on'});
```

The `DistributedPipelining` property applies only to Embedded MATLAB Function blocks or Stateflow charts within a subsystem.

For detailed information, see “Distributed Pipeline Insertion” in the Simulink HDL Coder documentation.

Compatibility Considerations

If your existing control files specified automatic pipelining implicitly using the `OutputPipeline` parameter, you should change your control files to specify automatic pipelining explicitly as in the following code excerpt:

```
c.forEach('*',...
    'eml_lib/Embedded MATLAB Function', {},...
 'hdlstateflow.StateflowHDLInstantiation', {'OutputPipeline', 2, 'DistributedPipelining', 'on'});
```

Coefficient Multiplier Optimization for Digital Filter, FIR Decimation, and FIR Interpolation Filters

The `CoeffMultipliers` implementation parameter lets you specify use of canonic signed digit (CSD) or factored CSD optimizations for processing coefficient multiplier operations in code generated for certain filter blocks. Specify the `CoeffMultipliers` parameter in a control file using the following syntax:

- `{'CoeffMultipliers', 'csd'}`: Use CSD techniques to replace multiplier operations with shift and add operations. CSD techniques minimize the number of addition operations required for constant multiplication by representing binary numbers with a minimum count of nonzero digits. This decreases the area used by the filter while maintaining or increasing clock speed.

- `{'CoeffMultipliers', 'factored-csd'}`: Use factored CSD techniques, which replace multiplier operations with shift and add operations on prime factors of the coefficients. This option lets you achieve a greater filter area reduction than CSD, at the cost of decreasing clock speed.
- `{'CoeffMultipliers', 'multipliers'}` (default): Retain multiplier operations.

The coder supports `CoeffMultipliers` for the filter block implementations shown in the following table.

Block	Implementation
dsparch4/Digital Filter	hdldefaults.DigitalFilterHDLInstantiation
dspmlti4/FIR Decimation	hdldefaults.FIRDecimationHDLInstantiation
dspmlti4/FIR Interpolation	hdldefaults.FIRInterpolationHDLInstantiation

See also "Block Implementation Parameters" in the Simulink HDL Coder documentation.

hdlnewblackbox Function Generates Black Box Control Statements

The `hdlnewblackbox` function provides a simple way to create the control file statements that are required to generate black box interfaces for one or more subsystems.

Given a selection of one or more subsystems from your model, `hdlnewblackbox` returns the following as string data in the MATLAB workspace for each selected subsystem:

- A `forEach` call coded with the correct `modelscope`, `blocktype`, and default implementation class (`SubsystemBlackBoxHDLInstantiation`) arguments for the block.
- (Optional) A cell array of strings enumerating the available implementations classes for the subsystem, in `package.class` form.
- (Optional) A cell array of cell arrays of strings enumerating the names of implementation parameters (if any) corresponding to the implementation

classes. `hdlnewblackbox` does not list data types and other details of implementation parameters.

For further information, see "Generating a Black Box Interface for a Subsystem" in the Simulink HDL Coder documentation.

hdlnewcontrolfile Function Optionally Returns Result to String

The `hdlnewcontrolfile` function (optionally) now can return control statements to a string variable.

To return control statements as text in the string variable `t`, instead of returning a control file, use the following syntax:

```
t = hdlnewcontrolfile(...)
```

See also hdlnewcontrolfile in the Simulink HDL Coder documentation.

-novopt Flag Added to Default Simulation Command in Generated Compilation Scripts

For improved operation with the ModelSim® (version 6.2 and later) simulator, the default values of the `HDLSimCmd` property string (and the **Simulation Command** GUI option) now includes the `-novopt` flag, as follows:

```
'vsim -novopt work.%s\n'
```

The `-novopt` flag directs the ModelSim simulator not to perform optimizations that remove signals from the simulation view.

Compatibility Considerations

If you are using ModelSim 6.0 or an earlier version, you should set the `HDLSimCmd` property string (or the **Simulation Command** GUI option) to omit the `-novopt` option, as follows:

```
'vsim work.%s\n'
```

Version 1.3 (R2008a) Simulink HDL Coder Software

This table summarizes what's new in V1.3 (R2008a):

New Features and Changes	**Version Compatibility Considerations**	**Fixed Bugs and Known Problems**	**Related Documentation at Web Site**
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Bug Reports	No

New features and changes introduced in this version are:

- "Complex Data Type Support" on page 17
- "Test Bench Enhancements" on page 18
- "Additional Blocks Supported for HDL Code Generation" on page 20
- "Enhanced Pipelining Support" on page 21
- "Additional RAM Blocks" on page 23
- "Enhanced Math Function and Divide Block Support" on page 24
- "Optional Suppression of Reset Logic Generation for Selected Delay Blocks" on page 24
- "Enhanced Embedded MATLAB Function Block Support" on page 25
- "Stateflow Chart Support Supports Complex Data Type" on page 28
- "hdlnewcontrolfile Function Generates Control Files Automatically" on page 28
- "Integrating FPGA Vendor Tools with Simulink® HDL Coder" on page 29
- "Timing Controller Optimization for Multirate Models" on page 29
- "Enhanced modelscope Syntax Increases Portability of Control Files" on page 30

- "“What's This?” Context-Sensitive Help Available for Simulink Configuration Parameters Dialog” on page 31
- “Limited Variable-Step Solver Support” on page 32

Complex Data Type Support

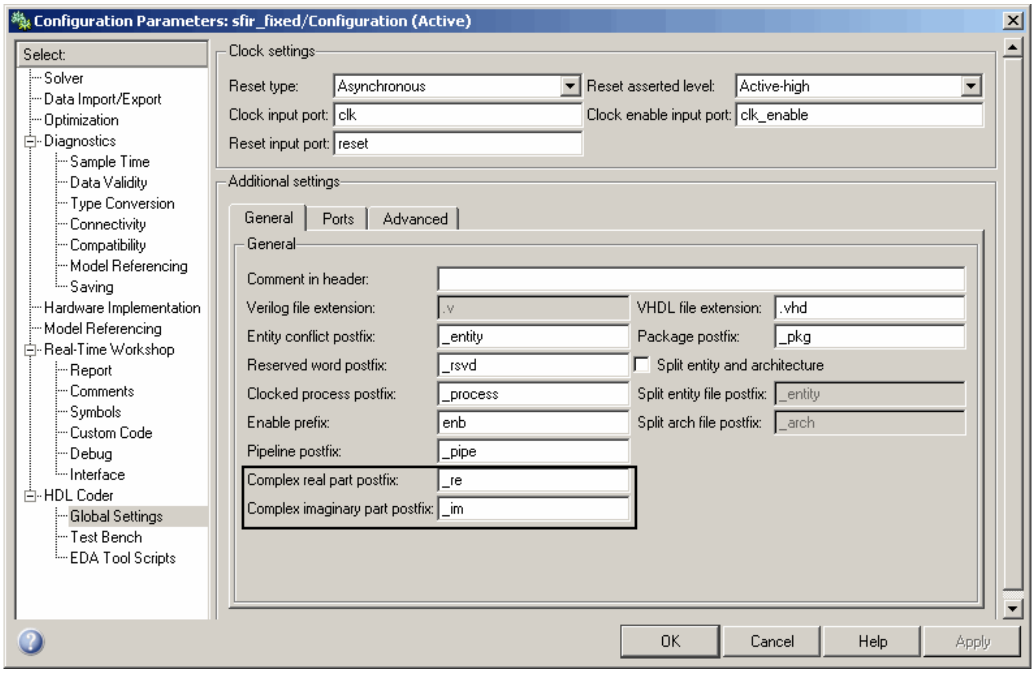
The coder now supports use of signals of complex data type.

You can use complex signals in the test bench without restriction.

In the device under test (DUT) selected for HDL code generation, support for complex signals is limited to a subset of the blocks supported by the coder. Some restrictions apply for some of these blocks. These blocks are listed in “Blocks That Support Complex Data”.

New Options Supporting Complex Data Types

Two new code generation options have been added to help you customize naming conventions for the real and imaginary components of complex signals in generated HDL code. These options are available to the **Global Settings / General** pane in the **HDL Coder** pane of the Configuration Parameters dialog box, as shown in the following figure.



Clock settings

Reset type: Asynchronous
Reset asserted level: Active-high
Clock input port: clk
Clock enable input port: clk_enable
Reset input port: reset

Additional settings

General

Comment in header:
Verilog file extension: .v
VHDL file extension: .vhd
Entity conflict postfix: _entity
Package postfix: _pkg
Reserved word postfix: _rsvd
Split entity and architecture
Clocked process postfix: _process
Split entity file postfix: _entity
Enable prefix: enb
Split arch file postfix: _arch
Pipeline postfix: _pipe
Complex real part postfix: _re
Complex imaginary part postfix: _im

The **Complex real part postfix** option (and the corresponding `ComplexRealPostfix` CLI property) specifies a string to be appended to the names generated for the real part of complex signals. The default postfix is `'_re'`. See also "Complex real part postfix".

The **Complex imaginary part postfix** option (and the corresponding `ComplexImagPostfix` CLI property) specifies a string to be appended to the names generated for the imaginary part of complex signals. The default postfix is `'_im'`. See also "Complex imaginary part postfix".

Test Bench Enhancements

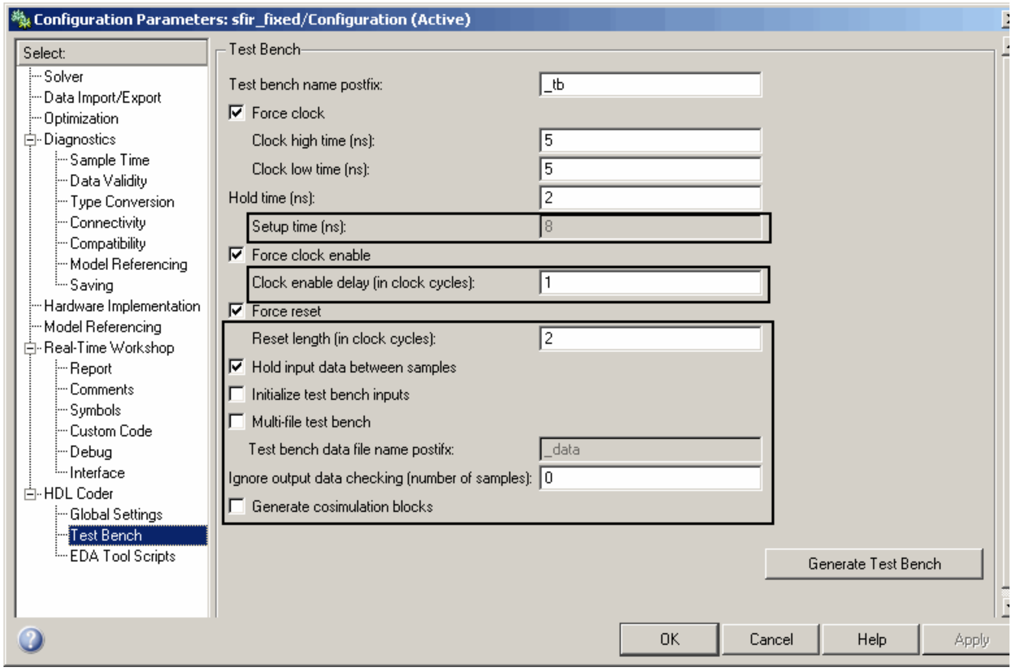
This release includes significant enhancements to test bench generation.

Test Bench Supports Complex Data Type

You can use complex signals in the test bench without restriction. Use of complex signals within the DUT is limited to a subset of supported blocks. See also "Complex Data Type Support" on page 17.

New Test Bench Options and Properties

A number of options have been added to the **HDL Coder / Test Bench** pane of the Configuration Parameters dialog box, as shown in the following figure.



Test bench name postfix: _tb
☑ Force clock
Clock high time (ns): 5
Clock low time (ns): 5
Hold time (ns): 2
Setup time (ns): 8
☑ Force clock enable
Clock enable delay (in clock cycles): 1
☑ Force reset
Reset length (in clock cycles): 2
☑ Hold input data between samples
☐ Initialize test bench inputs
☐ Multi-file test bench
Test bench data file name postfix: _data
Ignore output data checking (number of samples): 0
☐ Generate cosimulation blocks

Most of the new options have a corresponding command-line property. The following table lists the new options and their corresponding CLI properties, and provides hyperlinks to the relevant documentation.

GUI Option	Command-line Property
Setup time: See "Setup time (ns)"	This is a display-only field. It does not have a corresponding user-settable command-line property.
Clock enable delay (in clock cycles): See "Clock enable delay (in clock cycles)"	`TestBenchClockEnableDelay`
Reset length : See "Reset length (in clock cycles)"	`ResetLength`
Hold input data between samples: See "Hold input data between samples"	`HoldInputDataBetweenSamples`
Initialize test bench inputs: See "Initialize test bench inputs"	`InitializeTestBenchInputs`
Multi-file test bench : See "Multi-file test bench"	`MultifileTestBench`
Test bench data file name postfix : See "Test bench data file name postfix"	`TestBenchDataPostFix`
Ignore test bench data checking: See "Ignore output data checking (number of samples)"	`IgnoreDataChecking`
Generate cosimulation blocks: See "Generate cosimulation blocks"	`GenerateCoSimBlock`

Additional Blocks Supported for HDL Code Generation

The coder now supports the following blocks for HDL code generation:

- Communications Blockset/Comm Sources/Sequence Generators/PN Sequence Generator

  (This block requires Communications Blockset™.)

- Signal Processing Blockset/Multirate Filters/CIC Decimation
- Signal Processing Blockset/Multirate Filters/FIR Decimation
- Signal Processing Blockset/Signal Operations/NCO
- Signal Processing Blockset/Signal Processing Sources/Sine Wave
- Simulink/Discontinuities/Saturation
- Simulink/Discrete/Discrete-Time Integrator
- Simulink/Math Operations/Real-Imag to Complex
- Simulink/Math Operations/Complex to Real-Imag
- Simple Dual Port RAM (see also “Additional RAM Blocks” on page 23.)
- Single Port RAM (see also “Additional RAM Blocks” on page 23.)

See“Summary of Block Implementations” for a complete listing of blocks that are currently supported for HDL code generation.

Enhanced Pipelining Support

In the previous release, the coder introduced output pipelining support for many block implementations (see “OutputPipeline”). In this release, pipelining support has been significantly expanded and enhanced. The following sections discuss new pipelining features.

Input Pipelining

You can now specify generation of input pipeline registers for selected blocks. To do this, invoke the new block implementation parameter `{'InputPipeline', nStages}` in a control file. The parameter value (`nStages`) specifies the number of input pipeline stages (pipeline depth) in the generated code. See “InputPipeline” for further information.

Most HDL block implementations support `InputPipeline`. See “Summary of Block Implementations” for a complete list of block implemenations and their parameters.

Automatic Pipeline Insertion for Embedded MATLAB Function Block and Stateflow Chart

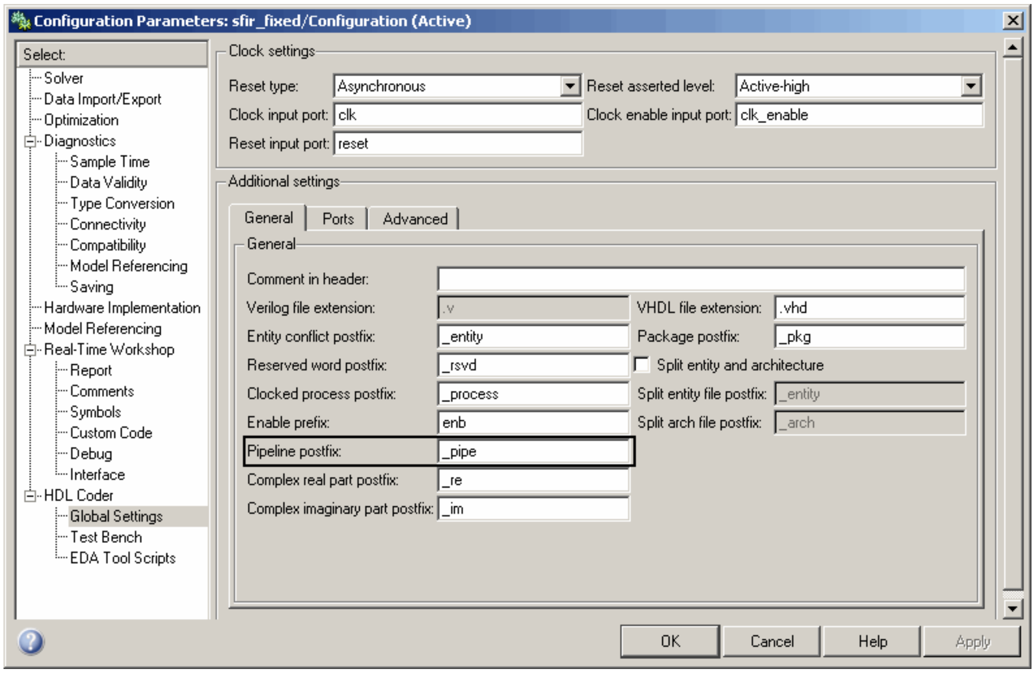
In this release, the coder introduces *automatic pipeline insertion*, a special optimization for HDL code generated from Embedded MATLAB Function blocks or Stateflow charts. Automatic pipeline insertion is performed when the `{'OutputPipeline', nStages}` parameter is specified for these blocks. When you specify `OutputPipeline`, the coder inserts internal pipeline stages into the HDL code generated for these blocks (rather than at the output of the HDL code) whenever possible. The `nStages` argument defines the number of pipeline stages to be inserted.

Automatic pipeline insertion lets you achieve higher clock rates in your HDL applications, at the cost of some latency caused by the introduction of pipeline registers.

See "Distributed Pipeline Insertion" for a detailed description of this feature.

Customizable Pipeline Register Names

When generating code for pipeline registers, the coder appends a postfix string to names of input or output pipeline registers. The default postfix string is `_pipe`. You can now customize the postfix string. To specify the postfix, use the **Pipeline postfix** option in the **Global Settings / General** pane in the **HDL Coder** pane of the Configuration Parameters dialog box (see the following figure). Alternatively, you can pass the desired postfix string in the `makehdl` property `PipelinePostfix`. See "Pipeline postfix" for an example.



Clock settings

Reset type: Asynchronous
Reset asserted level: Active-high
Clock input port: clk
Clock enable input port: clk_enable
Reset input port: reset

Additional settings

General

Comment in header:
Verilog file extension: .v
VHDL file extension: .vhd
Entity conflict postfix: _entity
Package postfix: _pkg
Reserved word postfix: _rsvd
Split entity and architecture: ☐
Clocked process postfix: _process
Split entity file postfix: _entity
Enable prefix: enb
Split arch file postfix: _arch
Pipeline postfix: _pipe
Complex real part postfix: _re
Complex imaginary part postfix: _im

Additional RAM Blocks

The coder now supports two new RAM blocks, supplementing the previously supported Dual Port RAM block:

- Simple Dual Port RAM: This block is identical to the Dual Port RAM , but does not have a data output at the write port. If data output at the write port is not required, you can achieve better RAM inferring with synthesis tools by using the Simple Dual Port RAM block rather than the Dual Port RAM block.
- Single Port RAM: This block provides data input, write address and write enable, and data output ports. The block GUI includes a **Output data during write** drop-down menu, providing options that control how the generated RAM handles data that is read into the RAM during a write operation.

See “RAM Blocks” for detailed information on RAM blocks.

Enhanced Math Function and Divide Block Support

The coder now supports a wider range of functions and algorithms for the Math Function and Divide blocks, as follows:

- The Math Function block `reciprocal` operation is now supported. Implementations using either hardware divide (HDL `/` operator) or iterative Newton algorithm are available.
- The Math Function block `conj` function is now supported.
- The Math Function block `sqrt` function implementations now support a choice of multiply/add, bitset shift/addition, or iterative Newton algorithms.
- The Math Operations/Divide block `reciprocal` operation now supports implementations using either hardware divide (HDL `/` operator) or the iterative Newton algorithm.

See "Math Function Block Implementations" and "Divide Block Implementations" for further information.

Optional Suppression of Reset Logic Generation for Selected Delay Blocks

The new `{'ResetType','None'}` block implementation parameter lets you suppress generation of reset logic for selected delay blocks. The following blocks support this parameter:

- Integer Delay
- Tapped Delay
- Unit Delay
- Unit Delay Enabled

The following control file specifies suppression of reset logic for a specific unit delay block within the subsystem `resetnone_examp/HDLSubsystem`.

```
function c = resetnone_examp

% Control file for resetnone_examp
c = hdlnewcontrol(mfilename);
c.generateHDLFor('resetnone_examp/HDLSubsystem');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Suppress reset logic for Unit Delay block

c.forEach('resetnone_examp/HDLSubsystem/Unit Delay',...
 'built-in/UnitDelay', {},...
 'hdldefaults.UnitDelayHDLEmission', {'ResetType','none'});
```

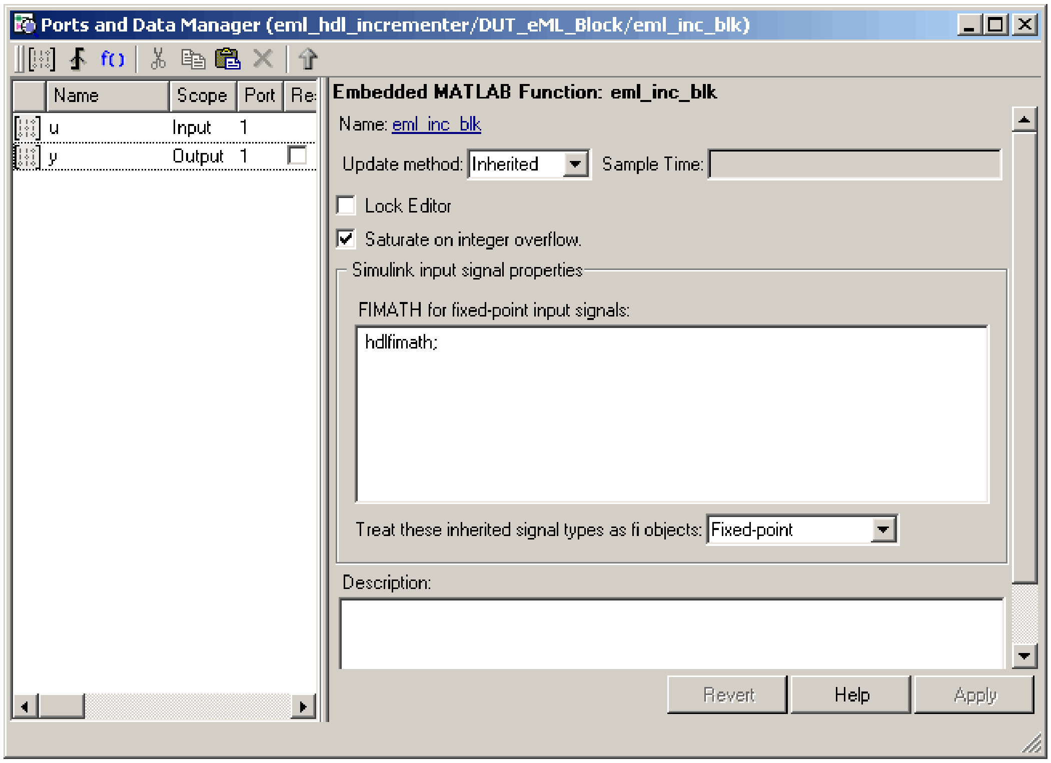
See ResetType for further information.

Enhanced Embedded MATLAB Function Block Support

HDL code generation support for the Embedded MATLAB Function block has been enhanced in Release 2008a, as discussed in the following sections.

hdlfimath Utility for Configuring Optimized FIMATH Settings

In this release, the coder provides the M-function `hdlfimath.m`, a utility that defines a FIMATH specification that is optimized for HDL code generation. When you configure an Embedded MATLAB Function Block for HDL code generation, it is strongly recommended that you replace the default **FIMATH for fixed-point signals** specification with a call to the `hdlfimath` function, as shown in the following figure.


Ports and Data Manager (eml_hdl_incrementer/DUT_eML_Block/eml_inc_blk)
Revert
Help
Apply

Name	Scope	Port	Re
u	Input	1	
y	Output	1	☐

Embedded MATLAB Function: eml_inc_blk

Name: eml_inc_blk

Update method: Inherited Sample Time:

☐ Lock Editor

☑ Saturate on integer overflow

Simulink input signal properties

FIMATH for fixed-point input signals:

```
hdlfimath;
```

Treat these inherited signal types as fi objects: Fixed-point

Description:

See "Use the hdlfimath Utility for Optimized FIMATH Settings" for further information.

Support for Complex Data Type

Embedded MATLAB Function block now supports use of complex data type for HDL code generation. All operators that support complex data types can be used in a Embedded MATLAB Function block code, subject to some restrictions. See the `eml_hdl_design_patterns` library for numerous examples showing applications of complex arithmetic in Embedded MATLAB Function blocks.

Support for Compiled External M-Functions on the Embedded MATLAB Path

You can now generate HDL code from Embedded MATLAB Function blocks that include compiled external M-functions. This feature lets you write reusable M-code that can be called from multiple Embedded MATLAB Function blocks.

Such functions must be defined in M-files that are on the Embedded MATLAB path, and must include the `%#eml` compilation directive. See "Adding the Compilation Directive %#eml" in the Embedded MATLAB documentation for complete details.

Support for Non-Tunable Parameter Arguments

An Embedded MATLAB function argument can be declared as a *parameter argument* by setting its **Scope** to `Parameter` in the Ports and Data Manager GUI.

Parameter arguments for Embedded MATLAB Function blocks do not appear as signal ports on the block. Parameter arguments do not take their values from signals in the Simulink model. Instead, their values come from parameters defined in a parent Simulink masked subsystem or variables defined in the MATLAB base workspace.

Only *nontunable* parameter arguments are supported for HDL code generation. If you declare parameter arguments in Embedded MATLAB function code that is intended for HDL code generation, be sure to clear the **Tunable** option for each parameter argument.

See also "Parameter Arguments in Embedded MATLAB Functions" in the Simulink documentation.

Enhanced Support for Fixed-Point Functions

Rounding Functions. The Embedded MATLAB Function block now supports the following Fixed-Point Toolbox™ rounding functions for HDL code generation:

- `ceil`

- `fix`
- `floor`
- `nearest`

See also “Supported Functions and Limitations of the Fixed-Point Embedded MATLAB Subset” in the Fixed-Point Toolboxdocumentation.

Other Functions. The Embedded MATLAB Function block now supports the following for HDL code generation:

- The `bitreplicate` function
- The `bitconcat` function now supports:
  - single-vector argument:

    ```
    bitconcat([a_vector]);
    ```

  - variable argument list:

    ```
    bitconcat(a,b,c,...);
    ```

For general information on these functions, see “Supported Functions and Limitations of the Fixed-Point Embedded MATLAB Subset” in the Fixed-Point Toolboxdocumentation.

Stateflow Chart Support Supports Complex Data Type

Stateflow charts now support the use of complex data types for HDL code generation. All operators that support complex data types can be used in a chart, without restriction.

See also “Stateflow HDL Code Generation Support”.

hdlnewcontrolfile Function Generates Control Files Automatically

The coder provides the new `hdlnewcontrolfile` utility to help you construct code generation control files. Given a selection of one or more blocks from your model, `hdlnewcontrolfile` generates a control file containing `forEach` statements and comments providing information about all supported implementations and parameters, for all selected blocks. The generated control file is automatically opened in the MATLAB Editor for further customization. See hdlnewcontrolfile for details.

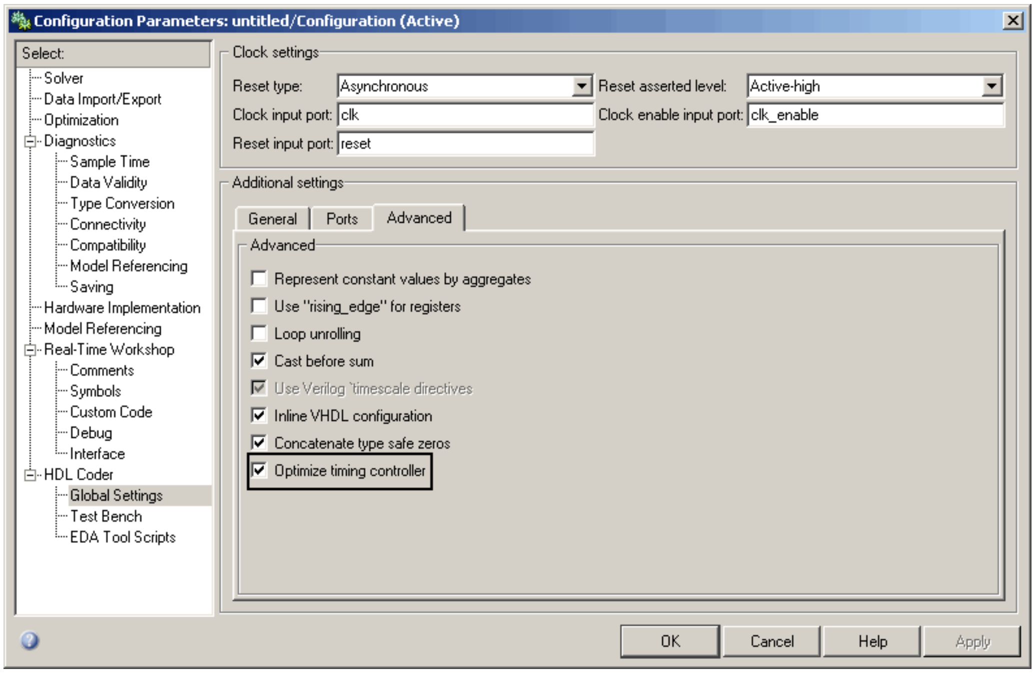
Integrating FPGA Vendor Tools with Simulink HDL Coder

You can now integrate Simulink HDL Coder with third-party FPGA vendor tools for HDL code generation. For detailed information on how to do this, see the Simulink HDL Coder Technical literature page: http://www.mathworks.com/products/slhdlcoder/technicalliterature.html.

Timing Controller Optimization for Multirate Models

The new **Optimize timing controller** option (and the corresponding `OptimizeTimingController` CLI property) optimizes generated `TimingController` entities for speed and code size by generating multiple counters (one counter for each rate in the model) in the timing controller code. The benefit of this optimization is that it generates faster logic, and reduces generated code size.

By default, the **Optimize timing controller** option is selected, as shown in the following figure.



Clock settings

Reset type: Asynchronous
Reset asserted level: Active-high
Clock input port: clk
Clock enable input port: clk_enable
Reset input port: reset

Additional settings

Advanced

- [ ] Represent constant values by aggregates
- [ ] Use "rising_edge" for registers
- [ ] Loop unrolling
- [✓] Cast before sum
- [✓] Use Verilog `timescale directives
- [✓] Inline VHDL configuration
- [✓] Concatenate type safe zeros
- [✓] Optimize timing controller

See “Optimize timing controller” for further details.

Enhanced modelscope Syntax Increases Portability of Control Files

The `modelscope` argument to the `forEach` and `forAll` control file methods has been enhanced to allow use of the period (.) to represent the root-level model. This lets you represent the current model as an abstraction, instead of explicitly coding the model name, as in the following example:

```
cfg.forEach( './Subsystem/MinMax', ...
    'built-in/MinMax', {}, ...
    'hdldefaults.MinMaxCascadeHDLEmission');
```

If you represent the model in this way, and then save the model under a different name, the control file does not require any change. Using the period to represent the root-level model makes the modelscope independent of the model name, and therefore more portable.

See also “Representation of the Root Model in modelscopes” in the Simulink HDL Coder User’s Guide.

Compatibility Considerations

When you save HDL code generation settings to a control file, the control file contains a `generateHDLFor` statement that specifies the path to the DUT specified in the **Generate HDL for** field. In previous releases, the root-level model in this path was represented by an explicit model name reference. In release 2008a, by default, the root-level model is represented by the period, as described above.

If you resave model settings to an existing control file, be aware that such explicit references to root-level model name will be changed to the period syntax, in accordance with this new default. This will not affect the operation of your existing control files in any way.

“What’s This?” Context-Sensitive Help Available for Simulink Configuration Parameters Dialog

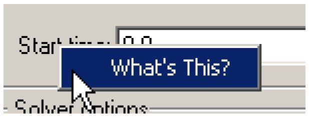
R2008a introduces “What’s This?” context-sensitive help for parameters that appear in the Simulink Configuration Parameters dialog. This feature provides quick access to a detailed description of the parameters, saving you the time it would take to find the information in the Help browser.

To use the "What’s This?" help, do the following:

1. Place your cursor over the label of a parameter.

2. Right-click. A **What’s This?** context menu appears.

   For example, the following figure shows the **What’s This?** context menu appearing after a right-click on the **Start time** parameter in the **Solver** pane.



3. Click **What's This?** A context-sensitive help window appears showing a description of the parameter.

Limited Variable-Step Solver Support

In previous releases, only fixed-step solvers were supported for HDL code generation. In the current release, you can select a variable-step **Solver type** for your model, under the following limited conditions:

- The device under test (DUT) is single-rate.
- The sample times of all signals driving the DUT are greater than 0.

Version 1.2 (R2007b) Simulink HDL Coder Software

This table summarizes what's new in V1.2 (R2007b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Bug Reports	No

New features and changes introduced in this version are:

- "HDL Code Generation for Single-Clock Multirate Models" on page 33
- "Additional Blocks Supported for HDL Code Generation" on page 34
- "Dual Port RAM Block Supported for Simulation and Code Generation" on page 35
- "Block Implementation Parameters Include Output Pipelining" on page 35
- "Summary of GUI Updates" on page 36
- "Digital Filter Block Restriction Removed" on page 38
- "Support for New Embedded MATLAB Bitwise Functions" on page 39
- "Default Hardware Target for Synthesis Scripts Updated to Virtex-4 " on page 39

HDL Code Generation for Single-Clock Multirate Models

The coder now supports HDL code generation for single-clock, single-tasking multirate models. Your model can include blocks running at multiple sample rates:

- Within in the device under test (DUT)

- In the test bench driving the DUT
- In both the test bench and the DUT

Multirate code generation support is described in detail in "Generating HDL Code for Multirate Models" in the documentation.

Additional Blocks Supported for Multirate Code Generation

The following blocks, frequently used in construction of multirate models, are now supported for HDL code generation:

- Signal Attributes/Rate Transition
- Signal Processing Blockset/Signal Operations/Downsample
- Signal Processing Blockset/Signal Operations/Upsample

New Property Added in Support of Multirate Code Generation

To support multirate code generation, a new `makehdl` property, `HoldInputDataBetweenSamples`, has been added. This property determines how long (in terms of base rate clock cycles) data values for subrate signals are held in a valid state. See `HoldInputDataBetweenSamples` for details.

Requirements and Restrictions for Multirate Code Generation

Certain requirements and restrictions apply to the use of multirate models for HDL code generation. See "Configuring Multirate Models for HDL Code Generation" for further information.

Additional Blocks Supported for HDL Code Generation

The coder now supports the following blocks for HDL code generation:

- Additional Math & Discrete/Additional Discrete/Unit Delay Enabled
- Math Operations/Divide
- Math Operations/Math Function (`sqrt` function only)
- Signal Attributes/Rate Transition

- Signal Processing Blockset/Signal Operations/Downsample
- Signal Processing Blockset/Signal Operations/Upsample
- Dual Port RAM (For information on this new block, see also "Dual Port RAM Block Supported for Simulation and Code Generation" on page 35.)

See "Summary of Block Implementations" for a complete listing of blocks that are currently supported for HDL code generation.

Dual Port RAM Block Supported for Simulation and Code Generation

The coder now provides the Dual Port RAM Block for use in simulation and code generation.

The Dual Port RAM block lets you:

- Simulate the behavior of a dual-port RAM with registered outputs in your model.
- Generate an interface to the inputs and outputs of the RAM in HDL code.

See "RAM Blocks" for full details.

Block Implementation Parameters Include Output Pipelining

The coder now supports *block implementation parameters*, which let you control details of the code generated for specific block implementations. Block implementation parameters are passed as property/value pairs to `forEach` or `forAll` calls in a code generation control file.

Supported Block Implementation Parameters

Block implementation parameters supported in the current release include:

- `'OutputPipeline', nStages`: This parameter lets you specify a pipelined implementation for selected blocks. The parameter value (`nStages`) specifies the number of pipeline stages (pipeline depth) in the generated code. `OutputPipeline` is supported by most Simulink HDL Coder HDL Coder block implementations.

- Interface generation parameters let you customize features of an interface generated for the following block types:
  - simulink/Ports & Subsystems/Model
  - built-in/Subsystem
  - lfilinklib/HDL Cosimulation
  - modelsimlib/HDL Cosimulation

  For example, you can specify generation of a black box interface for a subsystem, and pass in parameters that specify the generation and naming of clock, reset, and other ports in HDL code. Interface generation parameters are described in "Customizing the Generated Interface".

For more information on block implementation parameters, see the following sections in the documentation:

- "Specifying Block Implementations and Parameters in the Control File"
- "Block Implementation Parameters"
- "Summary of Block Implementations"

Using hdlnewforeach to Find Block Implementation Parameters

Given a selection of one or more blocks from your model, the `hdlnewforeach` function returns information about the available HDL implementations for each block.

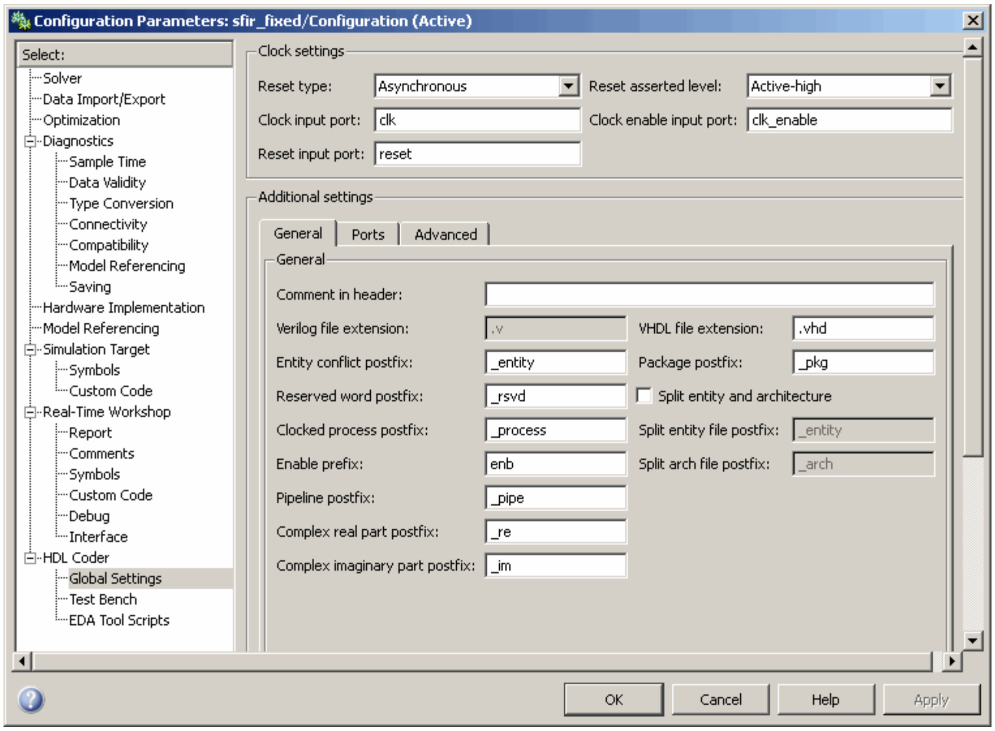
In the current release, the information returned by `hdlnewforeach` has been expanded. `hdlnewforeach` now returns an optional cell array of strings specifying the parameter(s) corresponding to each block implementation.

See "Generating Selection/Action Statements with the hdlnewforeach Function" for details.

Summary of GUI Updates

The following updates have been made to the Simulink HDL Coder GUI:

- The **Enable prefix** option is now supported by the GUI as well as by the `EnablePrefix` command-line property. See "Enable prefix" for details on this option.



Configuration Parameters: sfir_fixed/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
  - Sample Time
  - Data Validity
  - Type Conversion
  - Connectivity
  - Compatibility
  - Model Referencing
  - Saving
- Hardware Implementation
- Model Referencing
- Simulation Target
  - Symbols
  - Custom Code
- Real-Time Workshop
  - Report
  - Comments
  - Symbols
  - Custom Code
  - Debug
  - Interface
- HDL Coder
  - Global Settings
  - Test Bench
  - EDA Tool Scripts

Clock settings

Reset type: Asynchronous
Reset asserted level: Active-high
Clock input port: clk
Clock enable input port: clk_enable
Reset input port: reset

Additional settings

General | Ports | Advanced

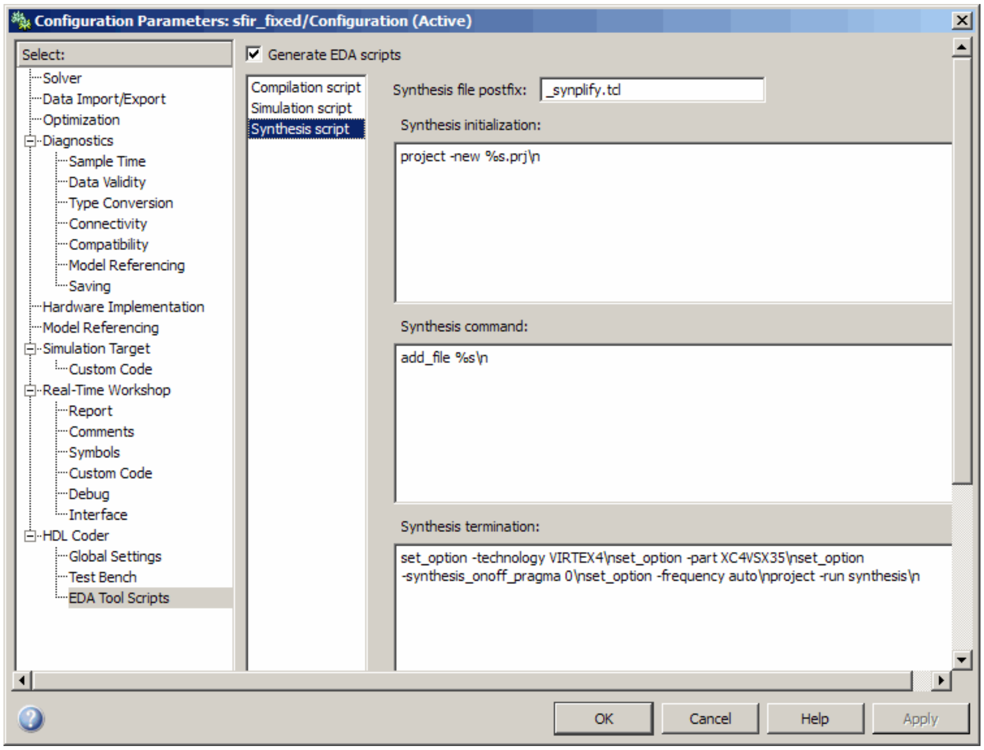
General

Comment in header:
Verilog file extension: .v
VHDL file extension: .vhd
Entity conflict postfix: _entity
Package postfix: _pkg
Reserved word postfix: _rsvd
Split entity and architecture: ☐
Clocked process postfix: _process
Split entity file postfix: _entity
Enable prefix: enb
Split arch file postfix: _arch
Pipeline postfix: _pipe
Complex real part postfix: _re
Complex imaginary part postfix: _im

- The default value for the **Synthesis termination** field of the EDA Tool Scripts dialog box has changed, as shown in the following figure. The default hardware target string in generated synthesis scripts now specifies
  - `technology` option: `VIRTEX4`

    In previous releases, this option defaulted to `VIRTEX2`.
  - `part` option: `XC4VSX35`

    In previous releases, this option defaulted to `XC2V500`.



Generate EDA scripts: ✓

Synthesis file postfix: _synplify.tcl

Synthesis initialization:

```
project -new %s.prj\n
```

Synthesis command:

```
add_file %s\n
```

Synthesis termination:

```
set_option -technology VIRTEX4\nset_option -part XC4VSX35\nset_option
-synthesis_onoff_pragma 0\nset_option -frequency auto\nproject -run synthesis\n
```

See also "Default Hardware Target for Synthesis Scripts Updated to Virtex-4 " on page 39.

Digital Filter Block Restriction Removed

In previous releases, Filter Design HDL Coder™ software was required to generate HDL code for the Digital Filter block when the **Dialog parameters** option was selected in the **Coefficient source** option group. This requirement has been removed.

In the current release, the HDL code generation requirements for the Digital Filter block vary according to the **Coefficient source** option you select, as follows:

- **Dialog parameters**: No additional toolboxes or blocksets required for HDL code generation.

- **Discrete-time filter object**: Filter Design HDL Coder software required.
- **Input port(s)**: This option is not supported for HDL code generation.

Support for New Embedded MATLAB Bitwise Functions

The code supports the new Embedded MATLAB fixed-point bitwise functions introduced in R2007b. Many of these functions map directly to HDL bitwise operators, resulting in very efficient HDL code. See "Using Fixed-Point Bitwise Functions" for examples of the use of these functions in HDL code generation.

For general information on Embedded MATLAB bitwise functions, see "Bitwise Operations" in the Fixed-Point Toolbox documentation.

Compatibility Considerations

In previous releases, the return type of the `bitget` function was `ufix8`. For more efficient HDL code generation, the return data type of the `bitget` function has been changed to `ufix1`. If your existing Embedded MATLAB code performs type casts to adapt values returned from `bitget` for HDL code generation, you may be able to eliminate these type casts.

Default Hardware Target for Synthesis Scripts Updated to Virtex-4

The default hardware target string in generated synthesis scripts now specifies

- `technology` option: `VIRTEX4`

  In previous releases, this option defaulted to `VIRTEX2`.

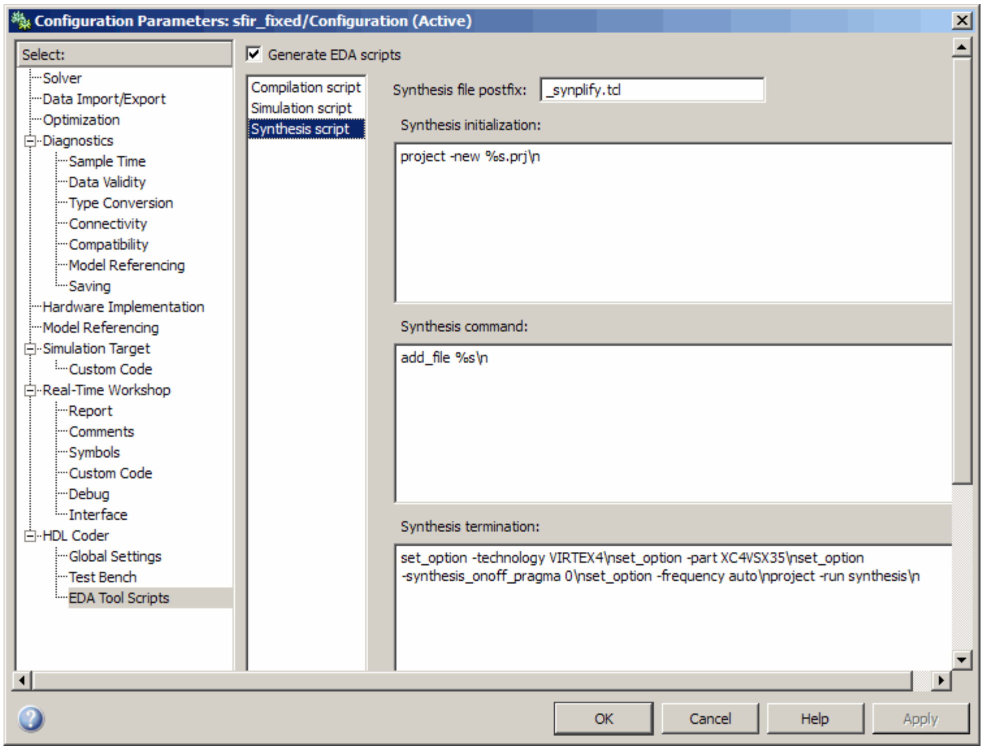
- `part` option: `XC4VSX35`

  In previous releases, this option defaulted to `XC2V500`.

These updates affect the default value for the `HDLSynthTerm` property. The default is:

```
['set_option -technology VIRTEX4\n',...
'set_option -part XC4VSX35\n',...
'set_option -synthesis_onoff_pragma 0\n',...
'set_option -frequency auto\n',...
'project -run synthesis\n']
```

The default value for the `HDLSynthTerm` property appears in the **Synthesis termination** field of the EDA Tool Scripts dialog box, as shown in the following figure.



Generate EDA scripts: ✓

Synthesis file postfix: _synplify.tcl

Synthesis initialization:

```
project -new %s.prj\n
```

Synthesis command:

```
add_file %s\n
```

Synthesis termination:

```
set_option -technology VIRTEX4\nset_option -part XC4VSX35\nset_option
-synthesis_onoff_pragma 0\nset_option -frequency auto\nproject -run synthesis\n
```

See also "Generating Scripts for HDL Simulators and Synthesis Tools".

Compatibility Considerations

If you have existing models that generate synthesis scripts using the previous defaults for `technology` or `part`, you may want to update your models and regenerate scripts.

Version 1.1 (R2007a) Simulink HDL Coder Software

This table summarizes what's new in V1.1 (R2007a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems	Related Documentation at Web Site
Yes Details below	No	Bug Reports	No

New features and changes introduced in this version are

- "Sign Block Supported for HDL Code Generation" on page 42
- "Link for Cadence Incisive HDL Cosimulation Block Supported" on page 42
- "GUI Support for Generation of EDA Tool Scripts" on page 43
- "Embedded MATLAB Function Block Supported for HDL Code Generation" on page 44
- "Stateflow HDL Code Generation Updates" on page 44

Sign Block Supported for HDL Code Generation

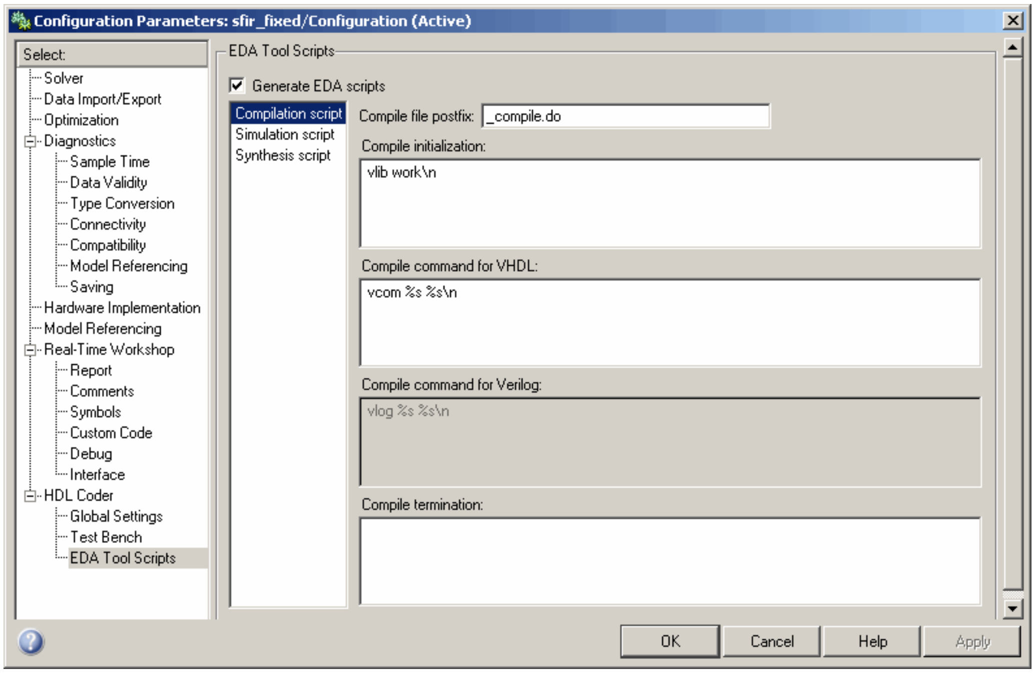
The Sign block (Simulink/Math Operations/Sign) is now supported for HDL code generation.

Link for Cadence Incisive HDL Cosimulation Block Supported

The coder now supports HDL code generation for the Link for Cadence® Incisive® HDL Cosimulation Block. You can use the HDL Cosimulation block with the coder to generate an interface to your manually written or legacy HDL code. When an HDL Cosimulation block is included in a model, the coder generates a VHDL or Verilog interface, depending on the selected target language. See "Code Generation for HDL Cosimulation Blocks" for details.

GUI Support for Generation of EDA Tool Scripts

The new **EDA Tool Scripts** pane of the GUI (shown in the following figure) lets you set all options that control generation of script files for third-party electronic design automation (EDA) tools. In previous releases, script generation options were available only through `makehdl` and `makehdltb` properties.



[✓] Generate EDA scripts

Compile file postfix: _compile.do

Compile initialization:

```
vlib work\n
```

Compile command for VHDL:

```
vcom %s %s\n
```

Compile command for Verilog:

```
vlog %s %s\n
```

Compile termination:

The list on the left of the **EDA Tool Scripts** pane lets you select from the following categories of options:

- **Compilation script**: Options related to customizing scripts for compilation of generated VHDL or Verilog code.
- **Simulation script**: Options related to customizing scripts for HDL simulators.
- **Synthesis script**: Options related to customizing scripts for synthesis tools.

See "Generating Scripts for HDL Simulators and Synthesis Tools" for detailed information on the **EDA Tool Scripts** options and on script generation in general.

Embedded MATLAB Function Block Supported for HDL Code Generation

The coder now supports synthesizable HDL code generation from the Embedded MATLAB Function block. See "Generating HDL Code with the Embedded MATLAB Function Block" for detailed information.

We are interested in getting your feedback on this introductory feature. Please send your responses to: hdlcoder_feedback@mathworks.com.

Stateflow HDL Code Generation Updates

This section describes some limitations on the use of Stateflow charts in HDL code generation have been removed in the current release. These are:

Restriction on Reading from Output Ports Removed

In the previous release, reading from output ports was disallowed. This restriction has been relaxed. You can now read from output ports if outputs are registered. (Outputs are registered if the **Initialize Outputs Every Time Chart Wakes Up** option is deselected.)

Stateflow Charts Fully Support Fixed Point Data Type

In the previous release, fixed-point data type support for Stateflow HDL code generation was limited to fixed point without scaling. This limitation has been removed. You can now use fixed-point data types without restriction in Stateflow charts intended for HDL code generation.

Compatibility Summary for Simulink HDL Coder Software

This table summarizes new features and changes that might cause incompatibilities when you upgrade from an earlier version, or when you use files on multiple versions. Details are provided in the description of the new feature or change.

Version (Release)	New Features and Changes with Version Compatibility Impact
Latest Version V1.4 (R2008b)	See the **Compatibility Considerations** subheading for this new feature or change: • "Default Entity Conflict Postfix Changed" on page 12 • "-novopt Flag Added to Default Simulation Command in Generated Compilation Scripts" on page 15 • "New DistributedPipelining Implementation Parameter for Embedded MATLAB Function Blocks and Stateflow Charts" on page 12
V1.3 (R2008a)	See the **Compatibility Considerations** subheading for this new feature or change: • "Enhanced modelscope Syntax Increases Portability of Control Files" on page 30

Version (Release)	New Features and Changes with Version Compatibility Impact
V1.2 (R2007b)	See the **Compatibility Considerations** subheading for this new feature or change: • "Default Hardware Target for Synthesis Scripts Updated to Virtex-4 " on page 39 • "Support for New Embedded MATLAB Bitwise Functions" on page 39
V1.1 (R2007a)	None