Simulink® HDL Coder™ Release Notes

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

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
Latest Version V2.2 (R2011b)	Yes Details	Yes Summary	Bug Reports
V2.1 (R2011a)	Yes Details	None	Bug Reports
V2.0 (R2010b)	Yes Details	Yes Summary	Bug Reports
V1.7 (R2010a)	Yes Details	Yes Summary	Bug Reports
V1.6 (R2009b)	Yes Details	Yes Summary	None
V1.5 (R2009a)	Yes Details	Yes Summary	None

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®). Determine if enhancements, bugs, or compatibility considerations in other products impact you.

If you are upgrading from a software version other than the most recent one, review the current release notes and all interim versions. For example, when you upgrade from V1.0 to V1.2, review the release notes for V1.1 and V1.2.

What Is 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 release appear under Bug Reports at the MathWorks Web site. Bug fixes can sometimes result in incompatibilities, so review the fixed bugs in Bug Reports for any compatibility impact.

Fixed Bugs and Known Problems

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. Bug Reports include 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.

Documentation on the MathWorks Web Site

Related documentation is available on `mathworks.com` for the latest release and for previous releases:

- Latest product documentation
- Archived documentation

Version 2.2 (R2011b) Simulink HDL Coder Software

This table summarizes what's new in Version 2.2 (R2011b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Bug Reports

New features and changes introduced in this version are:

- "HDL Verification with FPGA-in-the-Loop Added to HDL Workflow Advisor" on page 5
- "DCM Generation for FPGA Turnkey Workflow in HDL Workflow Advisor" on page 6
- "RAM Mapping for Persistent Arrays in MATLAB Function Blocks" on page 6
- "Option to Reuse HDL Code for Masked Subsystems with Tunable Parameters" on page 6
- "Distributed Pipelining Inside MATLAB Function Blocks with Persistent Variables" on page 8
- "Option to Inline or Instantiate HDL Code from MATLAB Function Blocks" on page 8
- "Option to Minimize Intermediate Signals in HDL Code" on page 9
- "Option to Exclude Time/Date Information in HDL File Header" on page 11
- "Delay Balancing Now Enabled By Default" on page 12
- "Custom Block Libraries Now Supported for HDL Code Optimization" on page 12
- "Generation of Validation Model Added to HDL Workflow Advisor" on page 13

- “Compatibility with Third-Party FPGA Synthesis Tools” on page 14
- “Critical Path Annotation Available When Place and Route Fails” on page 14
- “HDL Code Generation for MATLAB Function Blocks with Local Structure Variables” on page 14
- “Black Box Implementation Enables Specification of Library for Loading VHDL Component” on page 15
- “Reorganization of HDL Code Generation Options in Configuration Parameters Dialog Box” on page 16
- “Conversion of Error and Warning Message Identifiers” on page 16
- “Serial/Partly Serial/Distributed Arithmetic Architectures for Interpolation Filters” on page 17
- “Resource Sharing for Filter Blocks” on page 18
- “Multiplier Input and Output Pipelining in Serial Implementations” on page 18
- “HDL Support for Viterbi Decoder Block Delayed Reset” on page 19
- “Functions and Function Elements Being Removed” on page 21

HDL Verification with FPGA-in-the-Loop Added to HDL Workflow Advisor

In R2011b, you can verify FPGA designs with FPGA-in-the-Loop (FIL) and the HDL Workflow Advisor. The HDL Workflow Advisor, using the model that you provide, guides you through specific tasks to set FPGA-in-the-Loop simulation options. Then, working with EDA Simulator Link™ software, it creates the necessary programming file and downloads it to your selected development board. FIL generates a new verification model based on your original model. The link software then inserts the FIL block and compares its outputs to the outputs of the original subsystem used to generate the HDL code. You can then run and test your FPGA implementation on the development board using Simulink. See "Performing FPGA-in-the-Loop Simulation with HDL Workflow Advisor" for details. You will require an EDA Simulator Link license for FIL simulation.

FIL has been tested with: Xilinx® ISE 13.1. It supports Windows® 32, Windows 64, Linux® 32, and Linux 64.

DCM Generation for FPGA Turnkey Workflow in HDL Workflow Advisor

You can now generate a clock module to change the clock frequency of an FPGA design in the HDL Workflow Advisor FPGA turnkey workflow.

The new **Set Target Frequency** task enables you to specify system clock frequency for an FPGA turnkey target. This task is available when you set **Target workflow** to `FPGA Turnkey` in the **Set Target Device and Synthesis Tool** task.

RAM Mapping for Persistent Arrays in MATLAB Function Blocks

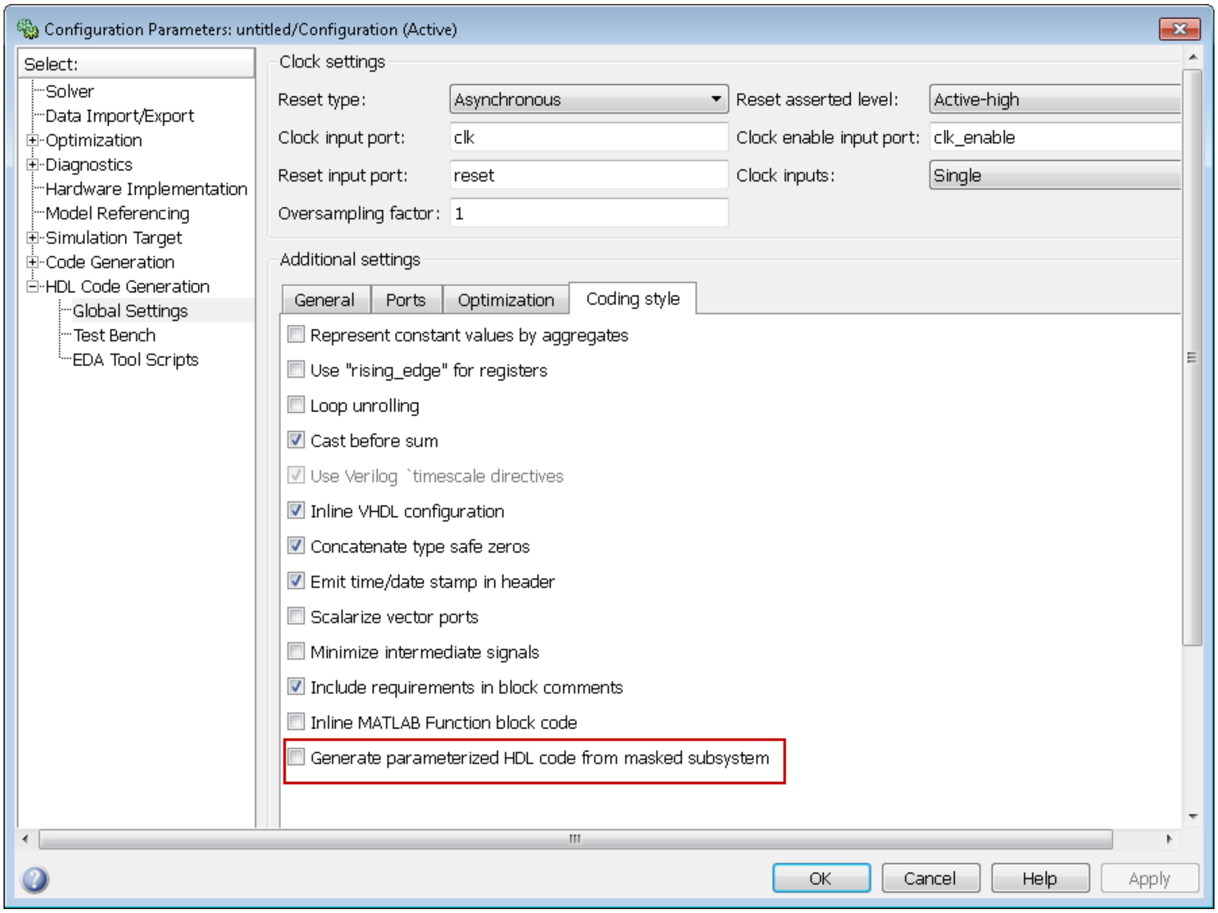
In R2011b, you can enable mapping of persistent arrays in MATLAB Function blocks to RAMs in the generated HDL code, instead of mapping to registers. This enhancement enables the design of more area-efficient hardware implementations.

A new block implementation parameter, `MapPersistentVarsToRAM`, specifies whether or not to allow RAM mapping for the MATLAB Function block. The model-level parameter, `RAMMappingThreshold`, specifies the minimum RAM size required for mapping the persistent array to a RAM instead of a register. You can specify the RAM mapping threshold at the command line or in the Configuration Parameters dialog box.

For more information, see "MapPersistentVarsToRAM" in the Simulink® HDL Coder™ documentation.

Option to Reuse HDL Code for Masked Subsystems with Tunable Parameters

R2011b provides an option to generate a single HDL file for multiple masked subsystems with tunable parameters that differ only in value.


Configuration Parameters: untitled/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
- HDL Code Generation
  - 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	Clock inputs:	Single
Oversampling factor:	1		

Additional settings

General | Ports | Optimization | Coding style

- [ ] 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
- [✓] Emit time/date stamp in header
- [ ] Scalarize vector ports
- [ ] Minimize intermediate signals
- [✓] Include requirements in block comments
- [ ] Inline MATLAB Function block code
- [ ] Generate parameterized HDL code from masked subsystem

You can now reuse HDL code and avoid generating a large number of files for masked subsystems. In previous releases, a separate HDL file was always generated for each subsystem when mask parameters differed in value.

For more information, see MaskParameterAsGeneric in the Simulink HDL Coder documentation.

Distributed Pipelining Inside MATLAB Function Blocks with Persistent Variables

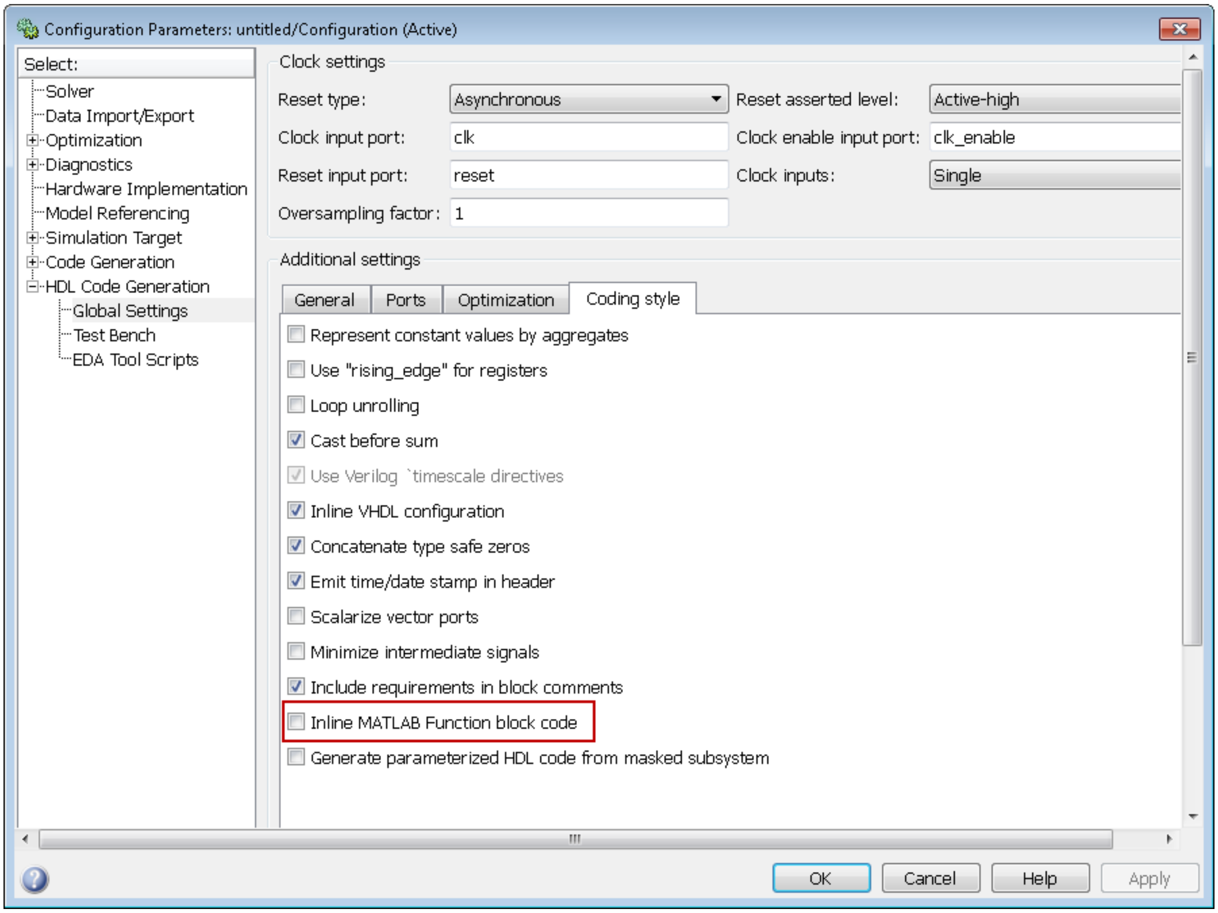
In previous releases, the coder inserted pipeline registers only at the outputs for a MATLAB Function block that used persistent variables. In R2011b, when you set `DistributedPipelining` to `on` and set `OutputPipeline` to a positive integer, the coder distributes pipeline registers inside a MATLAB Function block that uses persistent variables. This enhancement provides more opportunities to apply retiming during RTL synthesis.

For more information, see:

- "DistributedPipelining"
- "Distributed Pipeline Insertion for MATLAB Function Blocks"

Option to Inline or Instantiate HDL Code from MATLAB Function Blocks

R2011b provides an option to control whether to inline or instantiate the HDL code generated from MATLAB Function blocks.


Configuration Parameters: untitled/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
- HDL Code Generation
  - 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	Clock inputs:	Single
Oversampling factor:	1		

Additional settings

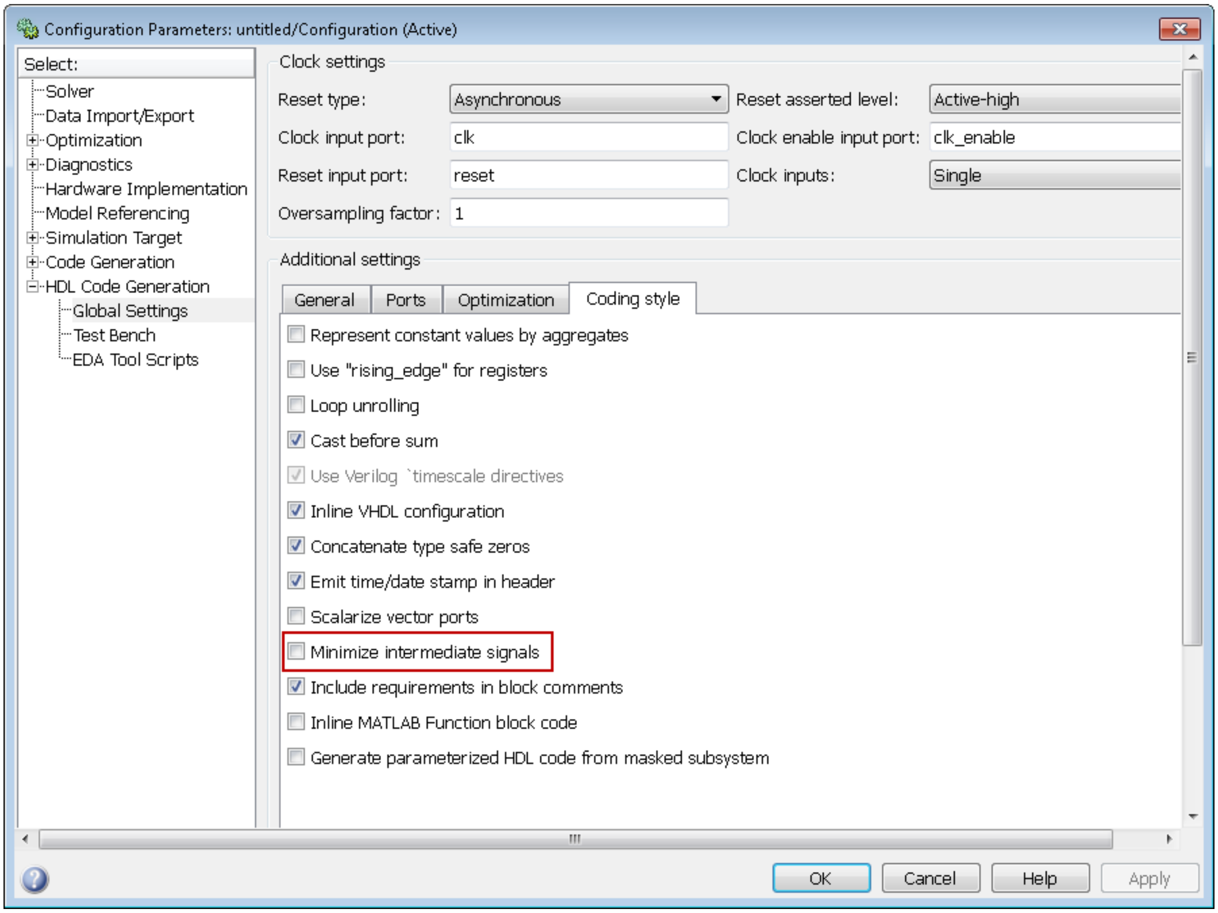
- [ ] Represent constant values by aggregates
- [ ] Use "rising_edge" for registers
- [ ] Loop unrolling
- [✓] Cast before sum
- [✓] Use Verilog `timescale directives
- [✓] Inline VHDL configuration
- [✓] Concatenate type safe zeros
- [✓] Emit time/date stamp in header
- [ ] Scalarize vector ports
- [ ] Minimize intermediate signals
- [✓] Include requirements in block comments
- [ ] Inline MATLAB Function block code
- [ ] Generate parameterized HDL code from masked subsystem

This enhancement enables you to avoid instantiation of HDL code for custom blocks you create with MATLAB Function blocks.

For more information, see InlineMATLABBlockCode in the Simulink HDL Coder documentation.

Option to Minimize Intermediate Signals in HDL Code

R2011b provides an option to minimize intermediate signals.


Configuration Parameters: untitled/Configuration (Active)

Clock settings

Reset type:	Asynchronous	Reset asserted level:	Active-high
Clock input port:	clk	Clock enable input port:	clk_enable
Reset input port:	reset	Clock inputs:	Single
Oversampling factor:	1		

Additional settings

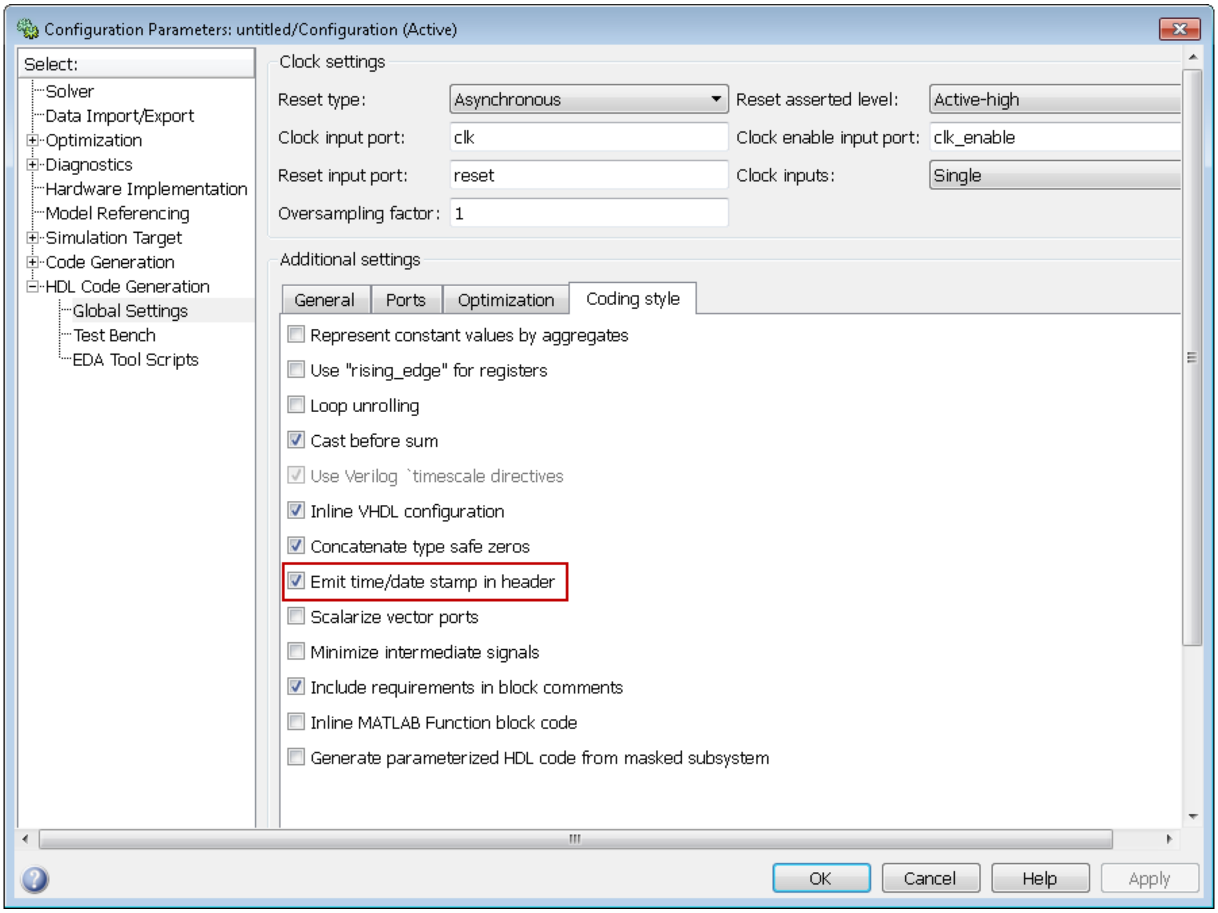
- [ ] Represent constant values by aggregates
- [ ] Use "rising_edge" for registers
- [ ] Loop unrolling
- [x] Cast before sum
- [x] Use Verilog `timescale directives
- [x] Inline VHDL configuration
- [x] Concatenate type safe zeros
- [x] Emit time/date stamp in header
- [ ] Scalarize vector ports
- [ ] Minimize intermediate signals
- [x] Include requirements in block comments
- [ ] Inline MATLAB Function block code
- [ ] Generate parameterized HDL code from masked subsystem

By removing intermediate signals and omitting declarations for those signals, you can enable better code coverage. For new models, this option is `off` by default. When you open existing models, this option is also `off` to preserve backward compatibility.

For more information, see MinimizeIntermediateSignals in the Simulink HDL Coder documentation.

Option to Exclude Time/Date Information in HDL File Header

R2011b provides an option to exclude time and date information in the generated HDL file header.


Configuration Parameters: untitled/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
- HDL Code Generation
  - 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	Clock inputs:	Single
Oversampling factor:	1		

Additional settings

General | Ports | Optimization | Coding style

- [ ] Represent constant values by aggregates
- [ ] Use "rising_edge" for registers
- [ ] Loop unrolling
- [x] Cast before sum
- [x] Use Verilog `timescale directives
- [x] Inline VHDL configuration
- [x] Concatenate type safe zeros
- [x] Emit time/date stamp in header
- [ ] Scalarize vector ports
- [ ] Minimize intermediate signals
- [x] Include requirements in block comments
- [ ] Inline MATLAB Function block code
- [ ] Generate parameterized HDL code from masked subsystem

By excluding the time and date information in the file header, you can more easily determine if two HDL files contain identical code. You can also avoid extraneous revisions of the same file when checking in HDL files to a source code management (SCM) system. For new models, this option is on by default.

When you open existing models, this option is also `on` to preserve backward compatibility.

For more information, see DateComment in the Simulink HDL Coder documentation.

Delay Balancing Now Enabled By Default

In R2011b, new models enable delay balancing by default. Upon detection of new delays introduced along one path, the coder inserts matching delays on all other paths. For more information, see "Delay Balancing" in the Simulink HDL Coder documentation.

Additional Supported Blocks for Delay Balancing

The following blocks are now supported for delay balancing:

- HDL Counter
- Counter Free-Running
- Math Function (`conj`, `reciprocal`, `transpose`, `hermitian`)
- Extract Bits
- Discrete-Time Integrator

Compatibility Considerations

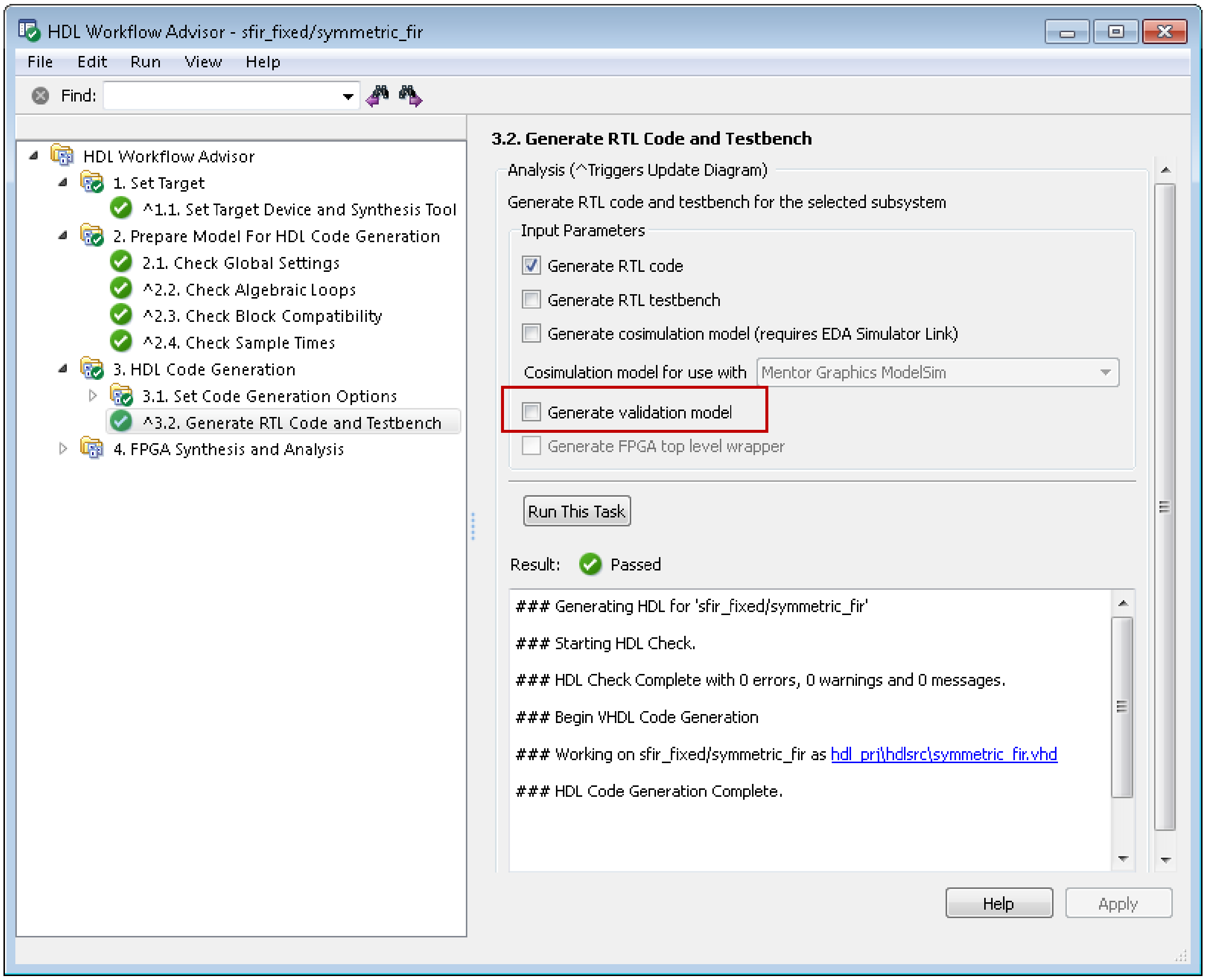
When you open existing models that disable delay balancing, those models now enable delay balancing by default. The generated HDL code for these models is functionally different. To disable delay balancing for a model before generating HDL code, change the model parameter `BalanceDelays` to `off`.

Custom Block Libraries Now Supported for HDL Code Optimization

You can now set HDL block properties in blocks from your custom block library to optimize the generated HDL code. Set HDL block properties in the **HDL Block Properties** dialog box, or use `hdlset_param`.

Generation of Validation Model Added to HDL Workflow Advisor

The **Generate RTL Code and Testbench** task in the HDL Workflow Advisor now provides an option to generate a validation model.


HDL Workflow Advisor - sfir_fixed/symmetric_fir
File Edit Run View Help
Find:
HDL Workflow Advisor
1. Set Target
^1.1. Set Target Device and Synthesis Tool
2. Prepare Model For HDL Code Generation
2.1. Check Global Settings
^2.2. Check Algebraic Loops
^2.3. Check Block Compatibility
^2.4. Check Sample Times
3. HDL Code Generation
3.1. Set Code Generation Options
^3.2. Generate RTL Code and Testbench
4. FPGA Synthesis and Analysis
Run This Task
Help
Apply

3.2. Generate RTL Code and Testbench

Analysis (^Triggers Update Diagram)

Generate RTL code and testbench for the selected subsystem

Input Parameters

- [✓] Generate RTL code
- [ ] Generate RTL testbench
- [ ] Generate cosimulation model (requires EDA Simulator Link)

Cosimulation model for use with: Mentor Graphics ModelSim

- [ ] Generate validation model
- [ ] Generate FPGA top level wrapper

Result: ✓ Passed

```
### Generating HDL for 'sfir_fixed/symmetric_fir'
### Starting HDL Check.
### HDL Check Complete with 0 errors, 0 warnings and 0 messages.
### Begin VHDL Code Generation
### Working on sfir_fixed/symmetric_fir as hdl_prj\hdlsrc\symmetric_fir.vhd
### HDL Code Generation Complete.
```

The validation model contains the DUT from the original model and the DUT from the generated cosimulation model. Using the validation model, you can verify that the output of the optimized DUT is bit-true to the results produced by the original DUT. With a validation model, you can observe the effects of streaming, resource sharing, and delay balancing.

For more information, see:

- “Generate RTL Code and Testbench”
- “Generating a Simulink Model for Cosimulation with an HDL Simulator”

Compatibility with Third-Party FPGA Synthesis Tools

In R2011b, the HDL Workflow Advisor is compatible with Xilinx ISE 13.1. For more information, see “HDL Workflow Advisor Compatibility with Third-Party Tools” in the Simulink HDL Coder documentation.

Critical Path Annotation Available When Place and Route Fails

The HDL Workflow Advisor now includes an option to ignore place and route errors in the **Perform Place and Route** task. When you select **Ignore place and route errors**, critical path annotation is available in the **Annotate Model with Synthesis Result** task even when place and route fails. In previous releases, when place and route failed in the ISE, critical path annotation was not available in the HDL Workflow Advisor.

HDL Code Generation for MATLAB Function Blocks with Local Structure Variables

In R2011b, you can generate HDL code for MATLAB Function blocks that contain local variables of `struct` type. For example, you can create local structures and pass them as inputs to subfunctions:

```
function out = fcn(u, v, w, x, y, z)
%#codegen

% Create a 'structure' variable from scalar or matrix variables
arr1 = [x y];
s_var = struct('su', u, 'sv', v, 'sw', w, 'valid', x, 'mvar', arr1, 'check_var', z);

% Pass the structure to a function
```

```
out = bus_logic(s_var);

function y = bus_logic(s_var)

% Access the elements of the structure in various contexts
switch s_var.check_var
    case 1
        a = s_var.mvar(1);
    case 2
        a = -s_var.mvar(2);
    otherwise
        a = uint8(10);
end

if a > 10
    y =  s_var.sv + s_var.sw;
elseif a < -10
    y =  s_var.sv - s_var.sw;
elseif a == 10
    y = bitand(s_var.sv, s_var.sw);
else
    y = s_var.valid;
end
```

The coder does not support structures as inputs or outputs of MATLAB Function blocks.

Black Box Implementation Enables Specification of Library for Loading VHDL Component

For black box implementations of subsystems, you can now specify the library from which to load a VHDL component. The following example specifies `mylib` as the library:

```
hdlset_param(gcb, 'VHDLComponentLibrary', 'mylib');
```

You can specify a VHDL component library other than `work`.

For more information, see:

- "Black Box Implementation for Subsystem Blocks"
- "Customizing the Generated Interface"

Reorganization of HDL Code Generation Options in Configuration Parameters Dialog Box

In R2011b, the following changes to the **HDL Code Generation** options appear:

- Two new parameters: **Generate HDL code** and **Generate validation model**

  These two check boxes replace the option buttons that previously appeared in the **Code generation output** section. The command-line property `CodeGenerationOutput` continues to work with `makehdl`, `hdlget_param`, and `hdl_setparam`.
- Migration of **Include requirements in block comments** to the **Global Settings > Coding style** section
- Migration of parameters in the **Global Settings > Advanced** section to the **Global Settings > Optimization** and **Global Settings > Coding style** sections

As a result of this reorganization, you can locate options for HDL code generation more easily. For more information, see "Code Generation Options in the Simulink HDL Coder Dialog Boxes".

Conversion of Error and Warning Message Identifiers

For R2011b, error and warning message identifiers have changed in Simulink HDL Coder.

Compatibility Considerations

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages.

For example, the `hdlcoder:workflow:IllegalNumArguments` identifier has changed to `HDLShared:hdldialog:HDLWAUsage`. If your code checks for `hdlcoder:workflow:IllegalNumArguments`, you must update it to check for `HDLShared:hdldialog:HDLWAUsage` instead.

To determine the identifier for a warning that appears at the MATLAB prompt, run the following command just after you see the warning:

```
[MSG,MSGID] = lastwarn;
```

This command saves the message identifier to the variable `MSGID`.

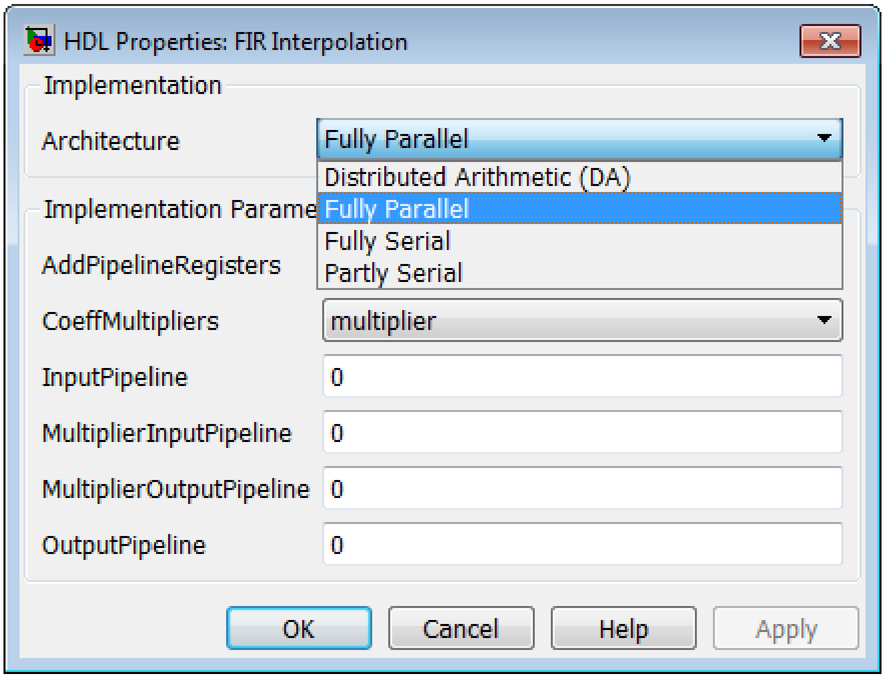
To determine the identifier for an error that appears at the MATLAB prompt, run the following commands just after you see the error:

```
exception = MException.last;
MSGID = exception.identifier;
```

Note Warning messages indicate a potential issue with your model or code. While you can turn off a warning, a suggested alternative is to change your model or code so it runs warning-free.

Serial/Partly Serial/Distributed Arithmetic Architectures for Interpolation Filters

The code now supports serial, partly serial, and distributed arithmetic (DA) filter implementations for the dspmlti4/FIR Interpolation block. See "Summary of Block Implementations" in the Simulink HDL Coder documentation for a list of the implementations and their associated parameters.


HDL Properties: FIR Interpolation

Implementation

Architecture: Fully Parallel

Distributed Arithmetic (DA)
Fully Parallel
Fully Serial
Partly Serial

Implementation Parame

AddPipelineRegisters

CoeffMultipliers: multiplier

InputPipeline: 0

MultiplierInputPipeline: 0

MultiplierOutputPipeline: 0

OutputPipeline: 0

Resource Sharing for Filter Blocks

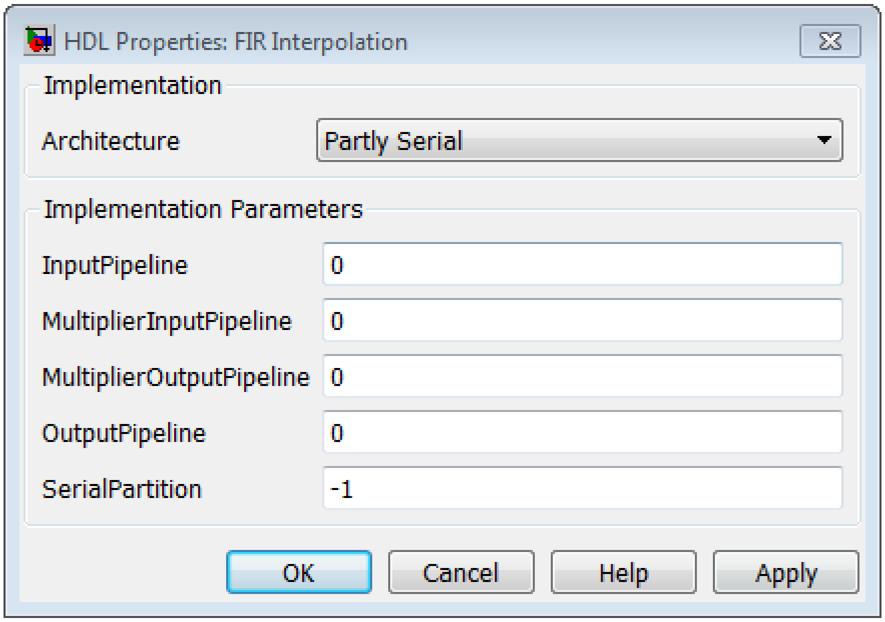
Resource sharing is available for filters with serial partitions. See "Streaming, Resource Sharing, and Delay Balancing" in the Simulink HDL Coder documentation.

Multiplier Input and Output Pipelining in Serial Implementations

The Generate HDL dialog box has these new parameter options—**MultiplierInputPipeline** and **MultiplierOutputPipeline**. These options support multiplier pipeline registers for serial architectures. Benefits of this feature include:

- Additional pipeline stages added to multipliers
- User control over how many and where pipeline stages are added

- Compatibility with all FIR-based structures


HDL Properties: FIR Interpolation

Implementation

Architecture: Partly Serial

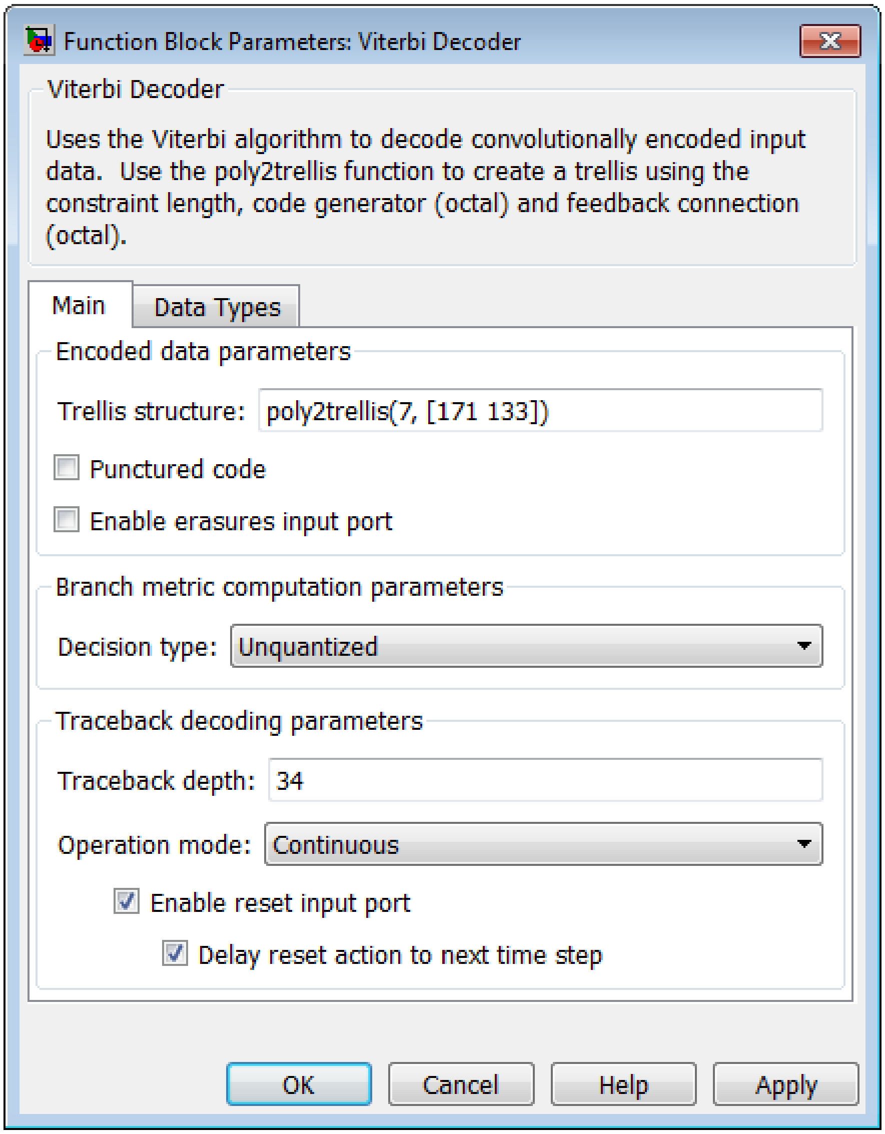
Implementation Parameters

InputPipeline	0
MultiplierInputPipeline	0
MultiplierOutputPipeline	0
OutputPipeline	0
SerialPartition	-1

HDL Support for Viterbi Decoder Block Delayed Reset

The Viterbi Decoder block has a delayed reset option, which allows you to generate HDL code for the reset port.

Select **Enable reset input port**, and then select **Delay reset action to next time step**. The Viterbi Decoder resets after decoding the incoming data. The **Delay reset action to next time step** check box appears only when you set the **Operation mode** parameter to `Continuous`. You must also select register-based traceback architecture on the HDL block properties dialog box.



Function Block Parameters: Viterbi Decoder

Viterbi Decoder

Uses the Viterbi algorithm to decode convolutionally encoded input data. Use the poly2trellis function to create a trellis using the constraint length, code generator (octal) and feedback connection (octal).

Main | Data Types

Encoded data parameters

Trellis structure: poly2trellis(7, [171 133])

☐ Punctured code

☐ Enable erasures input port

Branch metric computation parameters

Decision type: Unquantized

Traceback decoding parameters

Traceback depth: 34

Operation mode: Continuous

☑ Enable reset input port

☑ Delay reset action to next time step

Functions and Function Elements Being Removed

Function or Function Element Name	What Happens When you use the Function or Element?	Use This Instead	Compatibility Considerations
HDL Cosimulation block—Discovery	Errors	Support for Synopsys® Discovery™ has been removed.	Remove or replace the Discovery HDL Cosimulation block in your Simulink models.

Version 2.1 (R2011a) Simulink HDL Coder Software

This table summarizes what's new in Version 2.1 (R2011a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	None	Bug Reports

New features and changes introduced in this version are:

- "Synchronous Multiclock Code Generation" on page 23
- "GUI Support for Scalarize Vector Ports Option for VHDL" on page 23
- "GUI Support for Balancing Delays" on page 24
- "Delay Balancing Support for Filter and High-Level Blockset Blocks " on page 24
- "Enhanced CORDIC Algorithm Support" on page 24
- "Enhanced Retiming Features" on page 24
- "Enhanced Resource Sharing" on page 26
- "Enhanced Resource Utilization Report" on page 27
- "Enhanced Synthesis Script Generation" on page 27
- "RAM Mapping for Integer Delay Blocks" on page 29
- "Generic Parameter Passing to Subsystems With BlackBox Interface" on page 29
- "HDL Workflow Advisor Integrated FPGA Development Workflow" on page 30
- "Enhanced Support for Black Box Implementations" on page 30
- "Viterbi Decoder Enhancements" on page 30
- "Support for From and Goto Blocks at Any Level in Model" on page 30

Synchronous Multiclock Code Generation

R2011a supports synchronous multiple-clock code generation. You can specify multiple clocks in one of the following ways:

- Use the new model-level parameter `ClockInputs` with the function `makehdl` and specify the value as `'Multiple'`.
- In the **Clock settings** section of the **HDL Code Generation > Global Settings** pane in the 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 new `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.

GUI Support for Scalarize Vector Ports Option for VHDL

The **Scalarize Vector Ports** option provides GUI support for the `ScalarizePorts` property. **Scalarize Vector Ports** is located in the **Advanced** section of the **HDL Code Generation > Global Settings** pane of the Configuration Parameters dialog box.

Scalarize Vector Ports lets you control how the coder generates VHDL code for vector ports. When you select **Scalarize Vector Ports**, the coder flattens each vector port into a structure of scalar ports.

This option is available when the selected **Language** for code generation is VHDL.

See ScalarizePorts in the Simulink HDL Coder documentation for details.

GUI Support for Balancing Delays

The **Balance Delays** option provides GUI support for the `BalanceDelays` property. **Balance Delays** is located in the **Advanced** section of the **HDL Code Generation > Global Settings** pane of the Configuration Parameters dialog box.

When you select **Balance Delays**, if the coder detects the introduction of new delays along one path, it ensures that matching delays are inserted on all other paths. See "Delay Balancing" in the Simulink HDL Coder documentation for details.

Delay Balancing Support for Filter and High-Level Blockset Blocks

Support for delay balancing is now available for filter and high-level blockset blocks. This addition removes a limitation in this feature from an earlier release. See "Delay Balancing" in the Simulink HDL Coder documentation.

Enhanced CORDIC Algorithm Support

Simulink HDL Coder now adds HDL code generation support for:

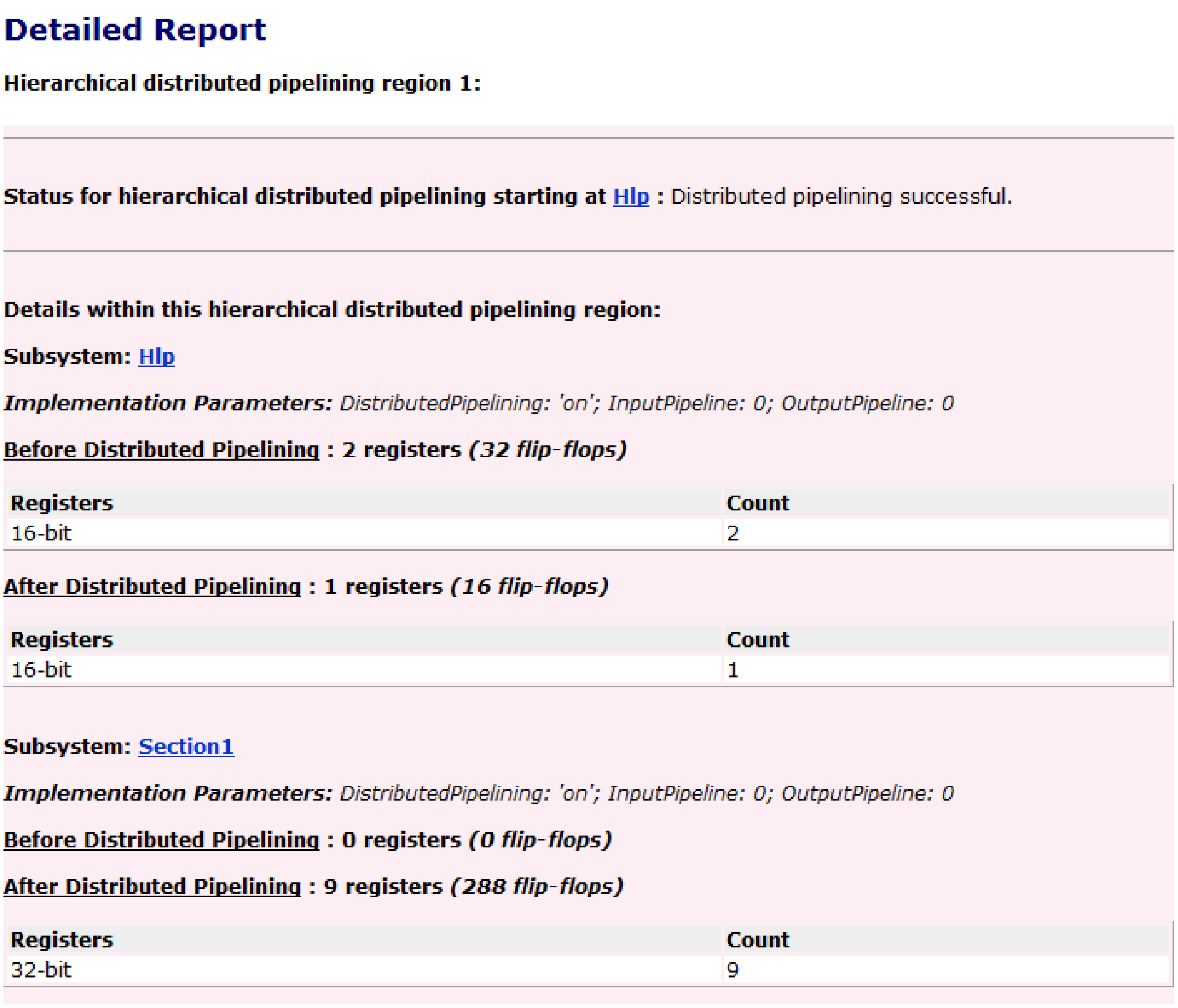
- The `cos+jsin` (complex exponential) function of the Trigonometric Function block, using the CORDIC approximation
- The CORDIC approximation method of the Magnitude-Angle to Complex block
- Use of unsigned data types with CORDIC approximation methods for `sin` and `cos` functions
- The `cordicrotate` function of the MATLAB Function block

See also "Trigonometric Function Block Requirements and Restrictions" in the Simulink HDL Coder documentation.

Enhanced Retiming Features

R2011a enhancements to distributed pipelining enable retiming to be applied across a subsystem hierarchy.

- The coder supports a new global parameter `HierarchicalDistPipelining`, which is `off` by default.
  - If `HierarchicalDistPipelining` is `on`, the coder applies retiming hierarchically down until `DistributedPipelining` is `off` for a given subsystem. The coder preserves the original subsystem hierarchy, which enables you to trace the changes that occur during pipelining for nested subsystems.
  - If `HierarchicalDistPipelining` is `off`, the previous functionality applies. That is, distribution happens only within a subsystem.
- If you set the `OptimizationReport` property to `on`, the coder adopts an enhanced reporting mechanism to provide the following information:
  - If `HierarchicalDistPipelining` is `on`, the Optimization Report uses colored sections to distinguish between different regions where the coder applies hierarchical distributed pipelining:


Detailed Report

Hierarchical distributed pipelining region 1:

Status for hierarchical distributed pipelining starting at Hlp : Distributed pipelining successful.

Details within this hierarchical distributed pipelining region:

Subsystem: Hlp

Implementation Parameters: *DistributedPipelining: 'on'; InputPipeline: 0; OutputPipeline: 0*

Before Distributed Pipelining : 2 registers *(32 flip-flops)*

Registers	Count
16-bit	2

After Distributed Pipelining : 1 registers *(16 flip-flops)*

Registers	Count
16-bit	1

Subsystem: Section1

Implementation Parameters: *DistributedPipelining: 'on'; InputPipeline: 0; OutputPipeline: 0*

Before Distributed Pipelining : 0 registers *(0 flip-flops)*

After Distributed Pipelining : 9 registers *(288 flip-flops)*

Registers	Count
32-bit	9

- If distributed pipelining fails, the coder gives you information that might help you correct the failure mode.

Enhanced Resource Sharing

R2011a resource sharing is enhanced as follows:

- Previously, atomic subsystems could be used in data-dependent sharing only if they did not contain any state elements. The removal of this restriction allows sharing in designs like IIR filters.
- Previously, the coder did not allow blocks contained in any feedback loop to be shared. In R2011a, you can share blocks within a feedback loop, provided that there are sufficient delays (Unit Delays or Integer Delays) available within the feedback loop to enable semantics-preserving resource sharing.

  To construct a sharable feedback loop, connect a Unit Delay or Integer Delay to the output of all Gain and Product blocks within the loop.

See also "Streaming, Resource Sharing, and Delay Balancing" in the Simulink HDL Coder documentation.

Enhanced Resource Utilization Report

The Resource Utilization Report now provides information on additional resources:

- Multiplexers (MUXes), including number and bit width of MUXes
- RAMs. including number of RAMs and their depth and bit width

Enhanced Synthesis Script Generation

R2011a lets you generate synthesis scripts specifically designed for your choice of one of the following synthesis tools:

- Xilinx ISE
- Mentor Graphics® Precision
- Altera® Quartus II
- Synopsys®Synplify Pro®

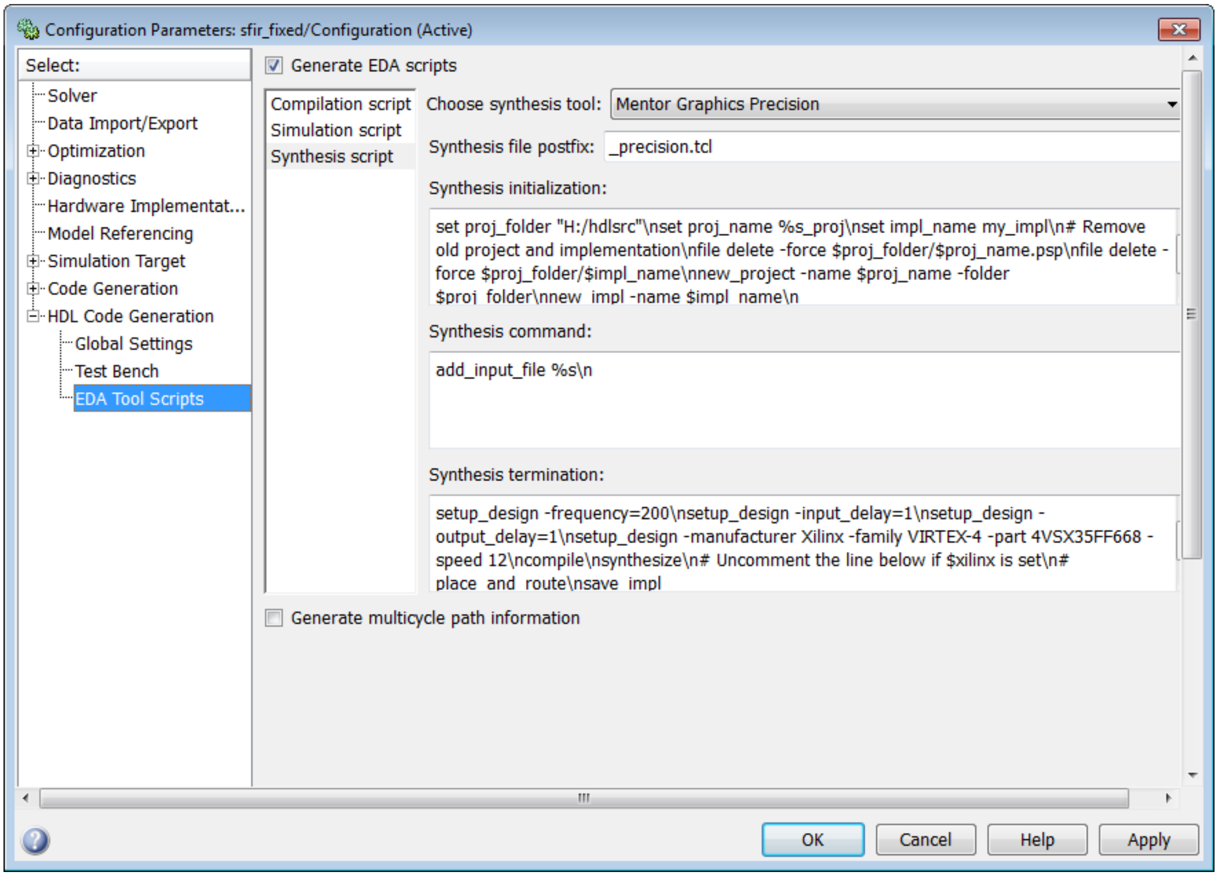
You can select a synthesis tool in one of the following ways:

- In the **HDL Code Generation > EDA Tool Scripts** pane of the Configuration Parameters dialog box, select a synthesis tool from the **Choose synthesis tool** pull-down menu.
- In a `makehdl` command, set the value of the new `'HDLSynthTool'` property.

When you select a synthesis tool, the coder:

- Enables synthesis script generation.
- Enters a file name postfix (specific to the chosen synthesis tool) into the **Synthesis file postfix** field.
- Enters strings (specific to the chosen synthesis tool) into the initialization, command, and termination fields.

The following figure shows the **Synthesis script** pane, with default option values entered for the Mentor Graphics Precision tool.



☑ Generate EDA scripts

Choose synthesis tool: Mentor Graphics Precision

Synthesis file postfix: _precision.tcl

Synthesis initialization:

```
set proj_folder "H:/hdlsrc"\nset proj_name %s_proj\nset impl_name my_impl\n# Remove
old project and implementation\nfile delete -force $proj_folder/$proj_name.psp\nfile delete -
force $proj_folder/$impl_name\nnew_project -name $proj_name -folder
$proj_folder\nnew_impl -name $impl_name\n
```

Synthesis command:

```
add_input_file %s\n
```

Synthesis termination:

```
setup_design -frequency=200\nsetup_design -input_delay=1\nsetup_design -
output_delay=1\nsetup_design -manufacturer Xilinx -family VIRTEX-4 -part 4VSX35FF668 -
speed 12\ncompile\nsynthesize\n# Uncomment the line below if $xilinx is set\n#
place_and_route\nsave_impl
```

☐ Generate multicycle path information

See also “Synthesis Script Options” in the Simulink HDL Coder documentation.

RAM Mapping for Integer Delay Blocks

In R2011a, you can enable mapping of Integer Delay blocks to RAMs in the generated HDL code, instead of mapping to shift registers. This enhancement enables the design of more area-efficient hardware implementations.
A new block-level parameter `UseRAM` specifies whether or not to allow RAM mapping for an Integer Delay block. A new model-level parameter `RAMMappingThreshold` specifies the minimum RAM size required for mapping the delay to a RAM instead of a shift register.

For more information, see:

- “UseRAM”
- RAMMappingThreshold

Generic Parameter Passing to Subsystems With BlackBox Interface

The `BlackBox` implementation for subsystems now includes the `'GenericList'` implementation parameter. Using `'GenericList'`, you can pass a list of parameter/value pairs (with optional data type specification) in string format to any subsystem having a `BlackBox` implementation.

You specify `'GenericList'` as a cell array of string data. Each element of the cell array is another cell array, of the form `{'Name', 'Value', 'Type'}`.`'Type'` is optional. If you omit `'Type'`, `'integer'` is passed as the data type.

The following example specifies two generic parameters, named `'Width'` and `'CntToNum'`, to be passed to the `BlackBox` interface generated for the current subsystem. The data type for `'Width'` is specified. The data type for `'CntToNum'` is not specified.

```
   hdlset_param(gcb, 'GenericList', '{{''Width'', ''8'', ''integer''}, {''C
```

HDL Workflow Advisor Integrated FPGA Development Workflow

In R2011a, the HDL Workflow Advisor supports an FPGA development workflow that integrates the following tasks:

- **Set Target Device and Synthesis Tool**: Selection of a target FPGA development board and supporting synthesis tool. A large number of development boards are now supported.
- **Set Target Interface**: Specification of I/O interface for the target board
- **Download to Target**, **Program Target Device**: Synthesis of generated HDL code and programming of the target board
- **Generate xPC Interface**: Generate xPC Target interface subsystem, which automatically creates an interface between the xPC Target™ system and Speedgoat FPGA boards.

For more information, see "Automated Workflows for Specific Target Devices and Synthesis Tools".

Enhanced Support for Black Box Implementations

In R2011a, virtual and atomic subsystem blocks of custom libraries that are below the level of the DUT also work with black box implementations. For more information, see "Black Box Implementation for Subsystem Blocks".

Viterbi Decoder Enhancements

The Viterbi decoder now supports RAM-based traceback for HDL code generation.

Support for From and Goto Blocks at Any Level in Model

In previous releases, the coder supported From and Goto blocks only if the connected blocks were located at the same subsystem level.

In R2011a, the coder supports use of From and Goto blocks at any level in the model hierarchy inside of the DUT.

Version 2.0 (R2010b) Simulink HDL Coder Software

This table summarizes what's new in Version 2.0 (R2010b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Bug Reports

New features and changes introduced in this version are:

- "HDL Parameters Now Saved to Model, Eliminating Need For Control Files" on page 32
- "Additional Simulink Blocks Supported for HDL Code Generation" on page 35
- "Resource Streaming and Sharing Optimizations Conserve Chip Area" on page 36
- "Delay Balancing" on page 37
- "New Timing Controller Naming Convention Avoids Name Clashes" on page 38
- "Scalarized Ports Option for VHDL" on page 38
- "Pipelining Improvements for Filter Blocks" on page 39
- "Reusable Code Generation for Atomic Subsystems" on page 40
- "Resource Utilization and and Optimization Reports" on page 41
- "Limitation on Generated Verilog Black Box Interfaces Removed" on page 44
- "Model Blocks Within Enabled and Triggered Subsystems Supported" on page 45
- "InitializeBlockRAM Property Controls Generation of Initial Signal Values for RAMs" on page 45

- “AddClockEnablePort Implementation Parameter for RAM Blocks Removed” on page 45
- “Do Not Use Floor Rounding Mode for Signed Integer Division” on page 45

HDL Parameters Now Saved to Model, Eliminating Need For Control Files

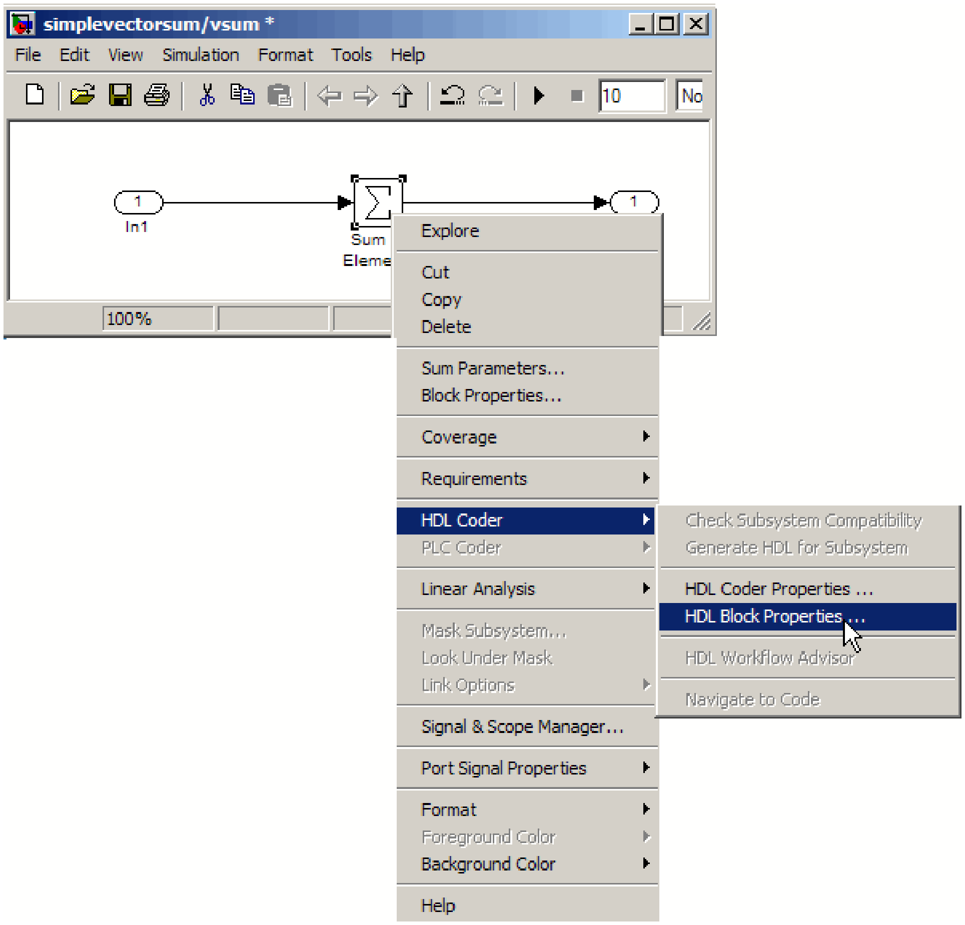
In R2010b, the coder saves all non-default HDL-related model settings, block implementation selections and implementation parameter settings to the model file. This eliminates the need to maintain a separate control file. Because the coder saves only the non-default parameter settings, the loading and saving of models is more efficient.

As of release R2010b, the coder does not support the attachment of a control file to a new model. If you have existing models with attached control files, you should convert them to the current format and remove control file linkage. The “Compatibility Considerations” on page 35 section of this release note describes this simple process.

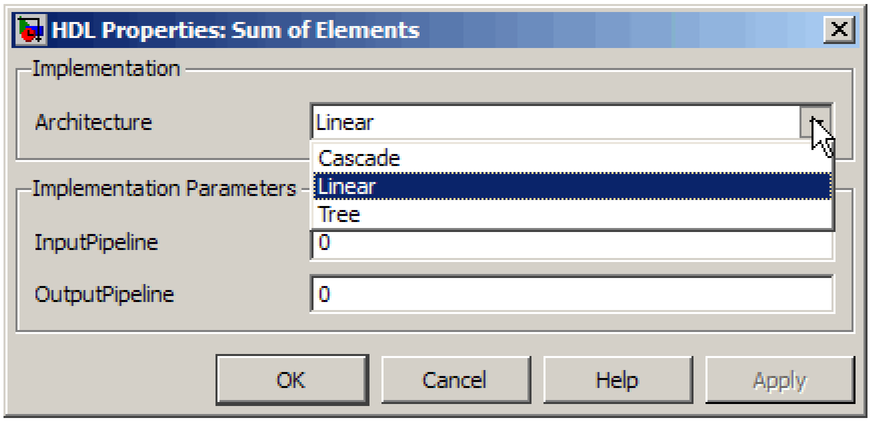
In R2010b, the coder provides both GUI enhancements and utility functions that let you select block implementations, set HDL-related model and block parameters, and perform other functions that previously required control files. “Setting HDL Block Properties in the GUI” on page 32 and “Setting and Getting HDL Block and Model Properties Programatically” on page 34 summarize these enhancements.

Setting HDL Block Properties in the GUI

R2010b lets you select block implementations and set implementation parameters using the new **HDL Properties** dialog box. This dialog box is available via the **HDL Coder** block context menu. The following figure shows this menu when accessed from a Sum of Elements block.



When you select **HDL Block Properties**, the **HDL Properties** dialog box for the block opens. The following figure shows the dialog box for a Sum of Elements block. The **Implementation** section of the dialog box lets you select one of three block implementations. The **Implementation Parameters** section of the dialog box and lets you view and set implementation parameters. For this block, all implementations support the **InputPipeline** and **OutputPipeline** parameters.



For further information, see "Selecting Block Implementations and Setting Implementation Parameters with the HDL Block Properties Dialog Box" in the Simulink HDL Coder documentation.

Setting and Getting HDL Block and Model Properties Programatically

The following new functions provided by R2010b let you report or set HDL-related property values at the block and model levels:

- hdlset_param: Set HDL-related parameters at the model or block level.
- hdlget_param: Return the value of specified HDL block-level parameter (or of all parameters) for a specified block.
- hdldispblkparams: Display HDL-related block parameters that have nondefault values, or all HDL-related block parameters for a specified block.
- hdldispmdlparams: Display HDL-related model parameters that have nondefault values, or all HDL-related model parameters.
- hdlapplycontrolfile: Apply settings from a control file to a model or subsystem.

See also "Specifying Block Implementations and Parameters for HDL Code Generation" in the Simulink HDL Coder documentation:

Compatibility Considerations

For backward compatibility, the coder continues to support code generation for existing models that have attached control files. The recommended practice is to convert such models to the current format and remove control file linkage.

To convert a model that has an attached control file:

1. Open the model. When the coder opens a model that has an attached control file, it loads and sets parameters as specified in the control file, and clears the control file linkage from the model. During this process, the coder displays the following messages:

   ```
   Found HDL control file attached to the model 'test_model' ...
   Loading control file 'test_model_control' ...
   Successfully loaded control file 'test_model_control.m' ...
   Please consider saving the model to make changes permanent ...
   Detaching the HDL control file from the model...
   ```

2. Save the model. The model now preserves all non-default settings. The next time you open the model, the coder will not display any control file status messages.

Additional Simulink Blocks Supported for HDL Code Generation

The coder now supports the blocks listed in the following table for HDL code generation.

“Summary of Block Implementations” in the Simulink HDL Coder documentation gives a complete listing of blocks that the coder supports for HDL code generation.

Block	Notes
hdldemolib/HDL FIFO	This block implements a first-in first-out (FIFO) register. See “HDL FIFO”
Communications System Toolbox™/Channel Coding/Convolutional /Convolutional Encoder	See “Convolutional Encoder Block Requirements and Restrictions”

Block	Notes
Communications System Toolbox/Digital Baseband Modulation/AM: • Rectangular QAM Demodulator Baseband • Rectangular QAM Modulator Baseband	See “Rectangular QAM Demodulator Baseband Block Requirements and Restrictions” See “Rectangular QAM Modulator Baseband Block Requirements and Restrictions”
Communications System Toolbox/Interleaving/Convolutional: • Convolutional Deinterleaver • Convolutional Interleaver	Release 2010b adds RAM-based implementations for these blocks. See “Convolutional Interleaver and Deinterleaver Block Requirements and Restrictions”
Communications System Toolbox/Interleaving/Convolutional: • General Multiplexed Deinterleaver • General Multiplexed Interleaver	See “General Multiplexed Interleaver and Deinterleaver Block Requirements and Restrictions”

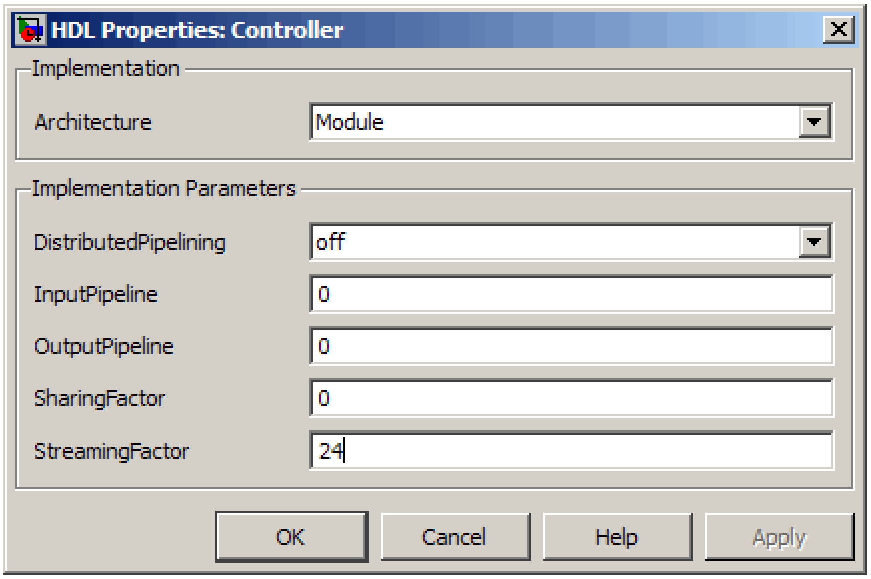
Resource Streaming and Sharing Optimizations Conserve Chip Area

R2010b introduces two related techniques that help you to conserve hardware resources and chip area:

Streaming is an optimization in which the coder transforms a vector data path to a scalar data path (or to several smaller-sized vector data paths) that executes at a faster rate. The generated code saves chip area by multiplexing the data over a smaller number of hardware resources. In effect, streaming allows some number of computations to share a hardware resource.

Resource sharing is an optimization in which the coder identifies multiple functionally equivalent resources and shares a single resource among them to perform their operations. This technique can realize a substantial reduction in chip area. For example, the generated code may use only one multiplier to perform the operations of several identically-configured multipliers from the original model. The coder achieves this by multiplexing the shared data over the shared resource.

The coder applies streaming and sharing at the subsystem level. You request streaming or sharing by specifying the subsystem HDL parameters `StreamingFactor` or `SharingFactor`. You can set these properties in the HDL Properties dialog for a subsystem, as shown in the following figure.


HDL Properties: Controller

Implementation

Architecture: Module

Implementation Parameters

DistributedPipelining: off

InputPipeline: 0

OutputPipeline: 0

SharingFactor: 0

StreamingFactor: 24

See "Streaming, Resource Sharing, and Delay Balancing" in the Simulink HDL Coder documentation for details.

Delay Balancing

In R2010b, the coder supports *delay balancing*, an option that corrects problems that occur when optimizations introduce delays along one path in a model, but equivalent delays are not introduced on other, parallel signal paths. When you enable delay balancing, if the coder detects introduction of new delays along one path, it ensures that matching delays are inserted on all other paths. When delay balancing is enabled, the coder guarantees that the generated model is functionally equivalent to the original model.

See “Delay Balancing” in the Simulink HDL Coder documentation for details.

New Timing Controller Naming Convention Avoids Name Clashes

The coder generates a timing controller code file if required by the design, for example when generating code for a multirate model.

In previous releases the timing controller file was always named `Timing_Controller.vhd` or `Timing_Controller.v`. This naming convention could cause name clashes between timing controllers generated in separate `makehdl` runs from models within a Model Reference block.

Release 2010b avoids such naming clashes by using a new naming convention for timing controllers, as follows:

- The coder supports a new string property, `TimingControllerPostfix`. The default value for `TimingControllerPostfix` is `'_tc'`.
- The timing controller name derives from the name of the subsystem that is selected for code generation (the DUT) as `DUTname+TimingControllerPostfix`.

For example, if the name of the DUT is `'symmetric_fir'`, the default name for the associated timing controller would be `'symmetric_fir_tc'`.

See also TimingControllerPostfix in the Simulink HDL Coder documentation.

Scalarized Ports Option for VHDL

The new `ScalarizePorts` property for `makehdl` lets you control how the coder generates VHDL code for vector ports.

When you set `ScalarizePorts` to`'on'`, the coder flattens each vector port into a structure of scalar ports.

You can use `ScalarizePorts` to generate non-conficting port definitions `ScalarizePorts` if you encounter typing or naming conflicts between vector ports when interfacing two or more generated VHDL code modules.

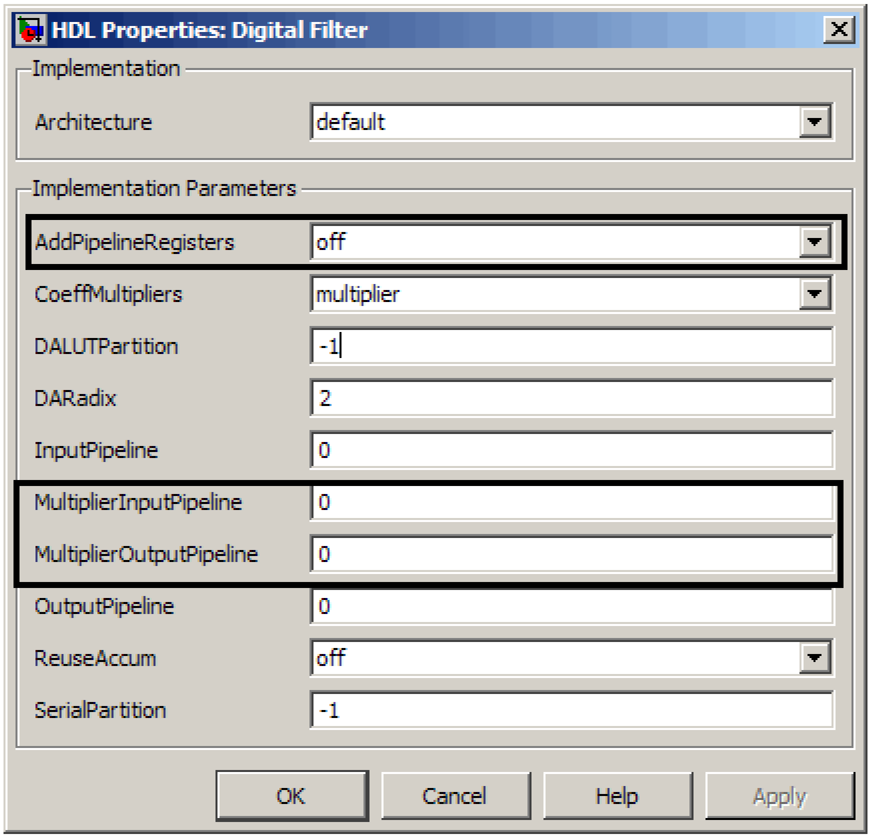
See ScalarizePorts in the Simulink HDL Coder documentation for details.

Pipelining Improvements for Filter Blocks

In R2010b, three new implementation parameters for filter blocks provide improved pipelining support. The new implementation parameters are:

- `AddPipelineRegisters` (Default: `off`): Inserts a pipeline register between stages of computation in a filter.
- `MultiplierInputPipeline` (Default: 0): Generates a specified number of pipeline stages at multiplier inputs for FIR filter structures.
- `MultiplierOutputPipeline` (Default: 0): Generates a specified number of pipeline stages at multiplier outputs for FIR filter structures.

The following figure shows these parameters, set to their default values, in the HDL Block Properties dialog box for a Digital Filter block.



Implementation

Architecture	default

Implementation Parameters

AddPipelineRegisters	off
CoeffMultipliers	multiplier
DALUTPartition	-1
DARadix	2
InputPipeline	0
MultiplierInputPipeline	0
MultiplierOutputPipeline	0
OutputPipeline	0
ReuseAccum	off
SerialPartition	-1

See “Pipelining Implementation Parameters for Filter Blocks”in the Simulink HDL Coder documentation for details.

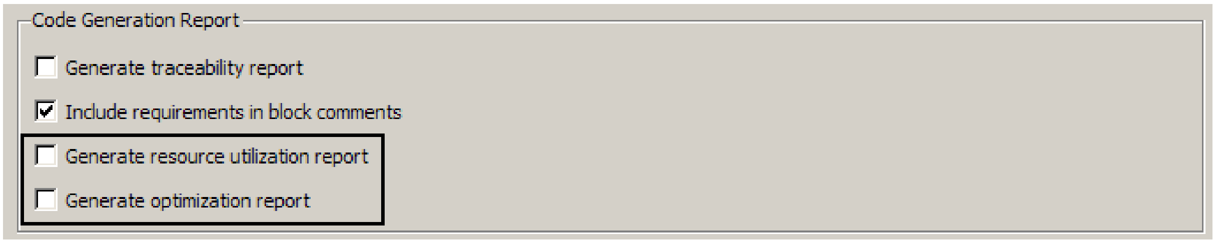
Reusable Code Generation for Atomic Subsystems

The new `HandleAtomicSubsystem` property for `makehdl` lets you generate reusable code for atomic subsystems that are identical. By generating reusable code, you can often eliminate the creation of numerous redundant source code files generated for identical subsystems. `HandleAtomicSubsystem` is enabled by default.

See “Generating Reusable Code for Atomic Subsystems” in the Simulink HDL Coder documentation for details.

Resource Utilization and and Optimization Reports

The coder now supports generation of two additional report sections to the HDL Code Generation report. You can add these sections to your reports by selecting the options highlighted in the following figure.


Code Generation Report

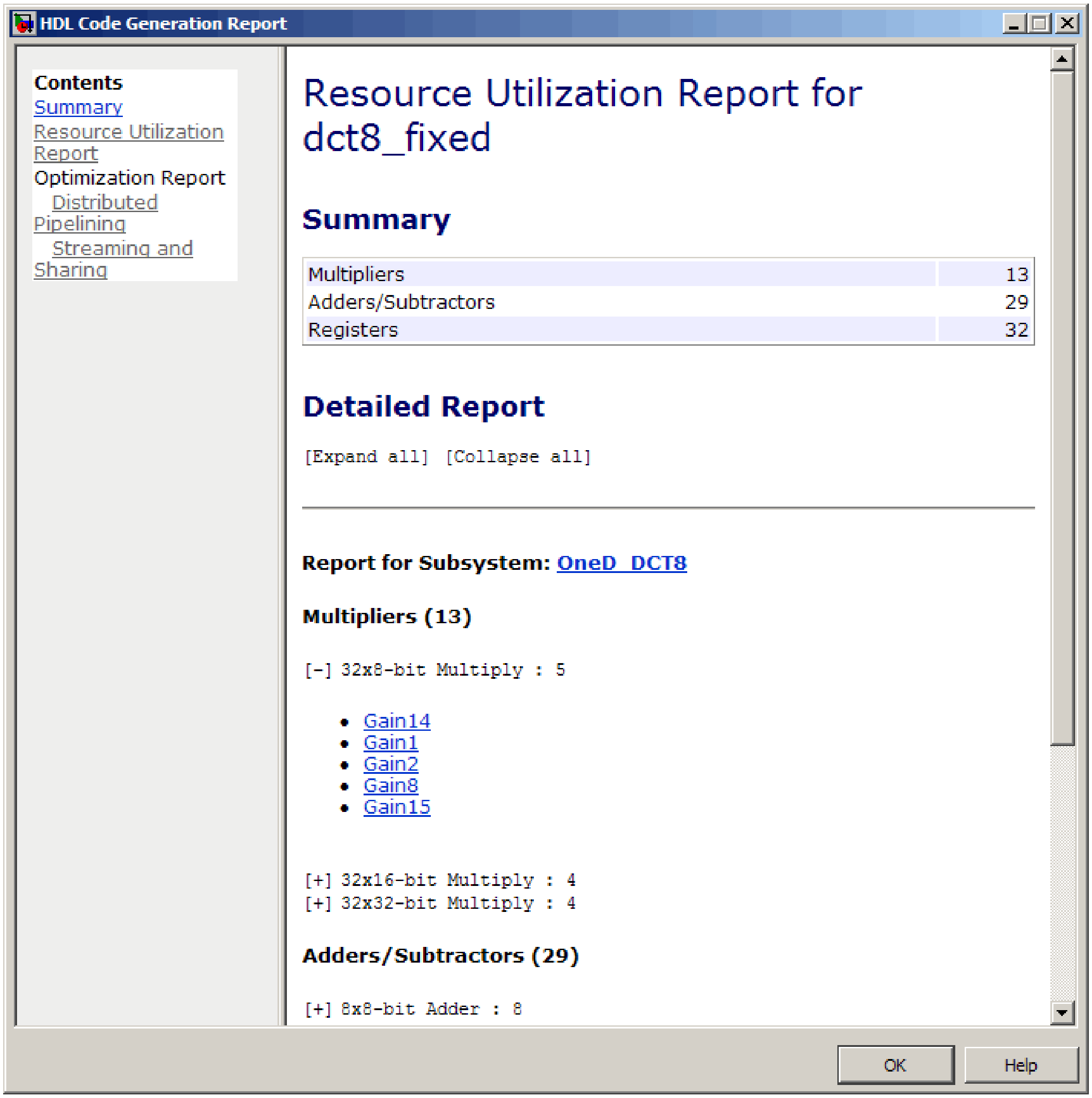
☐ Generate traceability report

☑ Include requirements in block comments

☐ Generate resource utilization report

☐ Generate optimization report

When you select **Generate resource utilization report**, the coder adds a Resource Utilization Report section. The Resource Utilization Report summarizes multipliers, adders/subtractors, and registers consumed by the device under test (DUT). It also includes a detailed report on resources used by each subsystem. The detailed report includes (wherever possible) links back to corresponding blocks in your model.



Resource Utilization Report for dct8_fixed

Summary

Multipliers	13
Adders/Subtractors	29
Registers	32

Detailed Report

```
[Expand all]  [Collapse all]
```

Report for Subsystem: OneD_DCT8

Multipliers (13)

```
[-] 32x8-bit Multiply : 5
```

- Gain14
- Gain1
- Gain2
- Gain8
- Gain15

```
[+] 32x16-bit Multiply : 4
[+] 32x32-bit Multiply : 4
```

Adders/Subtractors (29)

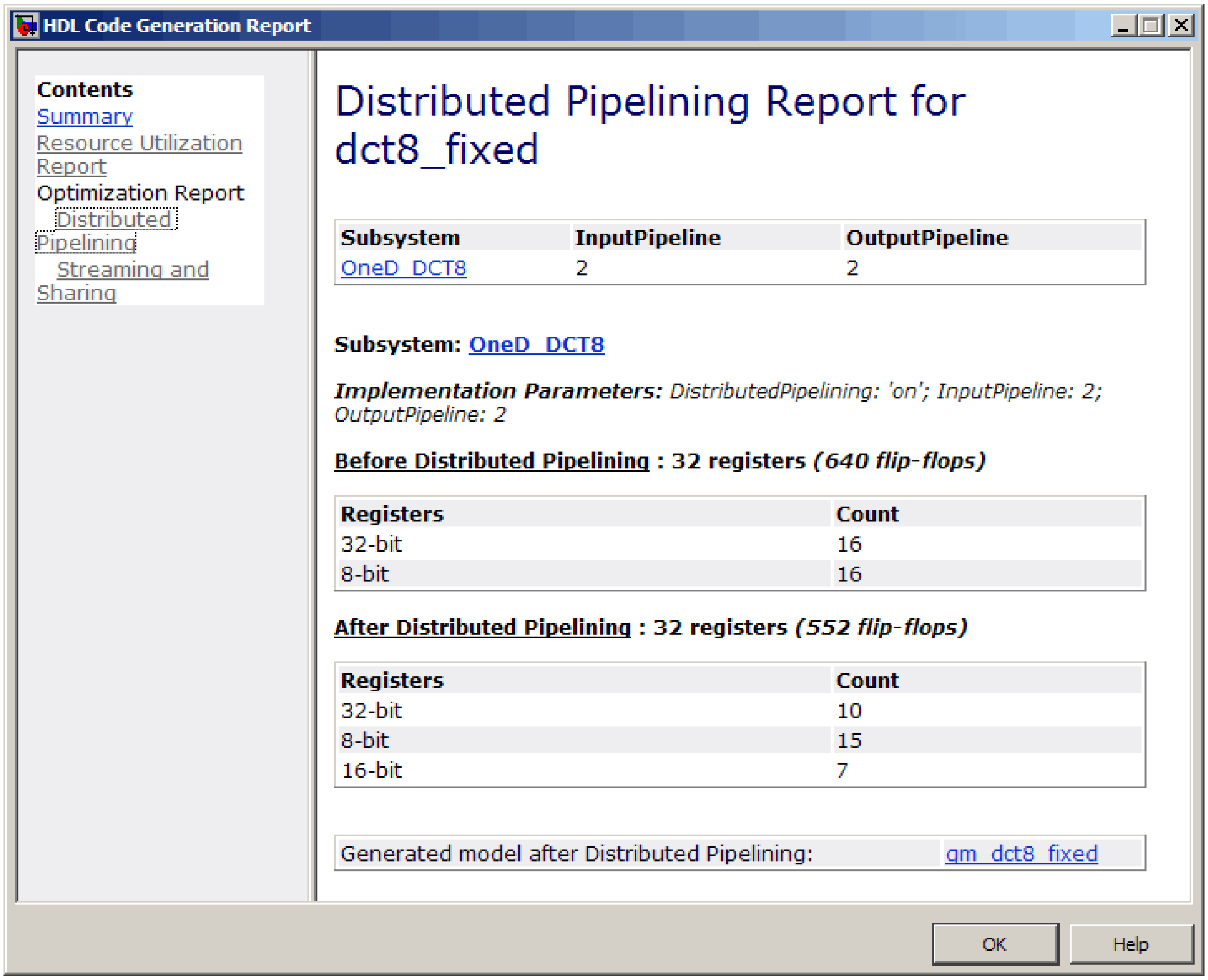
```
[+] 8x8-bit Adder : 8
```

When you select **Generate optimization report**, the coder adds an Optimization Report section, with two subsections:

Distributed Pipelining: this subsection shows details of subsystem-level distributed pipelining (if any subsystems have the `DistributedPipelining` option enabled). Details include comparative listings of registers and flip-flops before and after applying the distributed pipelining transform.

Streaming and Sharing: this subsection shows both summary and detailed information about all subsystems for which sharing or stream is requested.

The following figure shows the distributed pipelining subsection of a typical Optimization Report.



Distributed Pipelining Report for dct8_fixed

Subsystem	InputPipeline	OutputPipeline
OneD_DCT8	2	2

Subsystem: OneD_DCT8

Implementation Parameters: *DistributedPipelining: 'on'; InputPipeline: 2; OutputPipeline: 2*

Before Distributed Pipelining : 32 registers *(640 flip-flops)*

Registers	Count
32-bit	16
8-bit	16

After Distributed Pipelining : 32 registers *(552 flip-flops)*

Registers	Count
32-bit	10
8-bit	15
16-bit	7

Generated model after Distributed Pipelining:	gm_dct8_fixed

See "Creating and Using Code Generation Reports" in the Simulink HDL Coder documentation for further information.

Limitation on Generated Verilog Black Box Interfaces Removed

In previous releases, where Verilog was specified as the target language, the coder supported only scalar ports for code generation with the Subsystem black box implementation (`BlackBox`). Release R2010b removes this restriction.

The restriction still applies to some other block types. See “Limitation on Generated Verilog Interfaces” in the Simulink HDL Coder documentation for further information.

Model Blocks Within Enabled and Triggered Subsystems Supported

The coder now supports HDL code generation for Enabled Subsystem and Triggered Subsystem blocks that contain Model blocks.

InitializeBlockRAM Property Controls Generation of Initial Signal Values for RAMs

The new `InitializeBlockRAM` property for the `makehdl` function lets you enable or suppress generation of initial signal values for RAM blocks.

See also InitializeBlockRAM in the Simulink HDL Coder documentation.

AddClockEnablePort Implementation Parameter for RAM Blocks Removed

The `AddClockEnablePort` implementation parameter for the Dual Port RAM and Single Port RAM blocks was deprecated in Release 2009b.

In R2010b, the coder no longer supports `AddClockEnablePort` for RAM blocks.

Do Not Use Floor Rounding Mode for Signed Integer Division

The coder now displays an error message at code generation time if it encounters use of `'floor'` rounding mode for signed integer division . The HDL division operator does not support `'floor'` rounding mode.

To avoid this error, use `'fix'` mode for signed integer division operations, or else change the signed integer division operations to unsigned integer division.

Version 1.7 (R2010a) Simulink HDL Coder Software

This table summarizes what's new in Version 1.7 (R2010a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	None

New features and changes introduced in this version are:

- "Simplified Syntax for Specification of Block Implementations in Control Files" on page 47
- "HDL Workflow Advisor" on page 48
- "Additional Simulink Blocks Supported for HDL Code Generation" on page 50
- "CORDIC Algorithm Supported for Trigonometric Functions (sin, cos, sincos)" on page 51
- "Option to Minimize Generation of Clock Enables " on page 51
- "VHDLArchitectureName Property Supports Specification of Architecture Name" on page 51
- "VHDLLibraryName Property Supports Specification of Target Library " on page 51
- "Output Pipelining Now Supported for Subsystems" on page 52
- "Distributed Pipelining Now Supported for Subsystems" on page 52
- "CSD and Factored CSD Optimizations for Constant Multiplications" on page 52
- "Enhanced Gain Block Support" on page 52
- "FIR Decimation Filter Supports Distributed Arithmetic Architecture" on page 53

- “Serial, Partly Serial and Cascade Serial Architectures Supported for FIR Filter Implementations” on page 53
- “InstancePostfix Property Allows Specification of Extension to Postfix String” on page 53

Simplified Syntax for Specification of Block Implementations in Control Files

In R2010a, the coder supports a simplified syntax for specifying block implementations in a control file. The new syntax lets you specify a block implementation using simple keywords, instead of `package.class` notation. The new implementation keywords are generic, rather than block-specific. This approach lets you use the same keyword to specify implementation types such as `Tree`, `Cascade`, or `Linear` for all blocks that support such implementations. For example, the following control file specifies that the coder uses a cascade implementation for all Sum blocks and all Product blocks in the model.

```
function cfg = controlFile
cfg = hdlnewcontrol(mfilename);


cfg.forEach('*',...
 'built-in/Sum', {},...
 'Cascade', {});
cfg.forEach('*',...
  'built-in/Product', {},...
  'Cascade', {});
```

To specify the default implementation for any block, simply use the keyword `'default'`, as in the following example.

```
function cfg = controlFile
cfg = hdlnewcontrol(mfilename);


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

Refer to the Simulink HDL Coder documentation for a complete listing of supported blocks and their implementations.

Compatibility Considerations

In previous releases, control files specified block implementations using `package.class` syntax. For example, the following control file specifies the cascade implementation for Sum blocks, using `package.class` syntax.

```
function cfg = controlFile
 cfg = hdlnewcontrol(mfilename);


 cfg.forEach('*',...
  'built-in/Sum', {},...
  'hdldefaults.SumCascadeHDLEmission', {});
```

The coder continues to support control files that use `package.class` syntax. However, we strongly recommend that you convert existing control files to the new syntax. To convert an existing control file:

- Open a model that is linked to the control file.
- Open the Configuration Parameters dialog box and select the **HDL Coder** pane.
- Click **Generate** to generate HDL code for the model. The code generation process updates in-memory information that will be written to your updated control file.
- In the **Code generation control file** subpane, click **Save**. This overwrites the existing control file. The updated control file will use the new syntax.

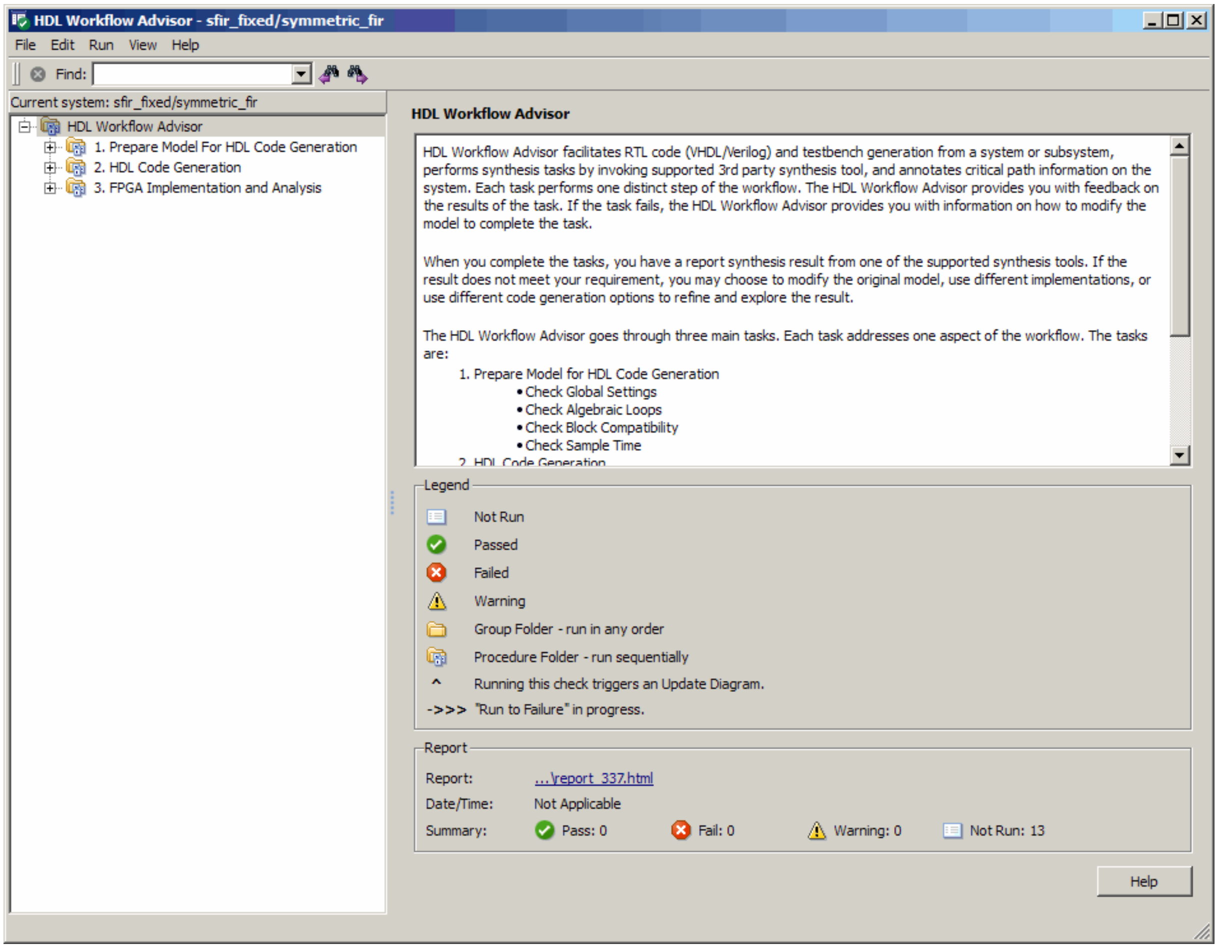
HDL Workflow Advisor

The HDL Workflow Advisor is a tool that supports all stages of the FPGA design process, including the following:

- Checking the Simulink model for HDL code generation compatibility
- HDL code and test bench generation

- Synthesis and timing analysis through integration with third-party synthesis tools (r2010a supports Xilinx ISE)
- Back annotation of the Simulink model with critical path and other information obtained during synthesis.

The following figure shows the top-level HDL Workflow Advisor window.



Current system: sfir_fixed/symmetric_fir

- HDL Workflow Advisor
  - 1. Prepare Model For HDL Code Generation
  - 2. HDL Code Generation
  - 3. FPGA Implementation and Analysis

HDL Workflow Advisor

HDL Workflow Advisor facilitates RTL code (VHDL/Verilog) and testbench generation from a system or subsystem, performs synthesis tasks by invoking supported 3rd party synthesis tool, and annotates critical path information on the system. Each task performs one distinct step of the workflow. The HDL Workflow Advisor provides you with feedback on the results of the task. If the task fails, the HDL Workflow Advisor provides you with information on how to modify the model to complete the task.

When you complete the tasks, you have a report synthesis result from one of the supported synthesis tools. If the result does not meet your requirement, you may choose to modify the original model, use different implementations, or use different code generation options to refine and explore the result.

The HDL Workflow Advisor goes through three main tasks. Each task addresses one aspect of the workflow. The tasks are:

1. Prepare Model for HDL Code Generation
   - Check Global Settings
   - Check Algebraic Loops
   - Check Block Compatibility
   - Check Sample Time

2. HDL Code Generation

Legend

☰	Not Run
✓	Passed
✗	Failed
⚠	Warning
📁	Group Folder - run in any order
📁	Procedure Folder - run sequentially
^	Running this check triggers an Update Diagram.
->>>	"Run to Failure" in progress.

Report

Report: ...\report_337.html
Date/Time: Not Applicable
Summary: ✓ Pass: 0 ✗ Fail: 0 ⚠ Warning: 0 ☰ Not Run: 13

See "Using the HDL Workflow Advisor" for further information.

Additional Simulink Blocks Supported for HDL Code Generation

The coder now supports the blocks listed in the following table for HDL code generation.

Block	Notes
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay Enabled Resettable	
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay Resettable	
simulink/Math Operations/Trigonometric Function	See also "CORDIC Algorithm Supported for Trigonometric Functions (sin, cos, sincos)" on page 51.
Signal Processing Blockset/Signal Operations/Repeat	
Communications System Toolbox/Digital Baseband Modulation/PM: • PSK Modulators (BPSK,M-PSK,QPSK) • PSK Demodulators (BPSK,M-PSK,QPSK)	
Communications System Toolbox/Interleaving/Convolutional: • Convolutional Interleaver • Convolutional Deinterleaver	"Convolutional Interleaver and Deinterleaver Block Requirements and Restrictions"
Communications System Toolbox/Error Detection and Correction/Convolutional/Viterbi Decoder	"Viterbi Decoder Block Requirements and Restrictions"

"Summary of Block Implementations" in the Simulink HDL Coder documentation gives a complete listing of blocks that the coder supports for HDL code generation.

CORDIC Algorithm Supported for Trigonometric Functions (sin, cos, sincos)

The Simulink Trigonometric Function block now supports the CORDIC algorithm for the `sin`,`cos`, and `sincos` functions.

Simulink HDL Coder HDL Coder now supports HDL code generation for the Trigonometric Function block for the `sin`,`cos`, and `sincos` functions. To generate HDL code for one these functions, select the Trigonometric Function block, you must set the **Approximation method** parameter to `CORDIC`.

See also "Trigonometric Function Block Requirements and Restrictions" in the Simulink HDL Coder documentation.

Option to Minimize Generation of Clock Enables

The new **Minimize clock enables** options lets you suppress generation of clock enable logic for single-rate designs, wherever possible. If your target device does not have registers with clock enables, you may want to consider selecting this option.

You can also use the command-line property `MinimizeClockEnables` to suppress generation of clock enable logic.

See also "Minimize clock enables" in the Simulink HDL Coder documentation.

VHDLArchitectureName Property Supports Specification of Architecture Name

The new `VHDLArchitectureName` property lets you specify the architecture name for generated HDL code. The default architecture name is `'rtl'`.

VHDLLibraryName Property Supports Specification of Target Library

The new `VHDLLibraryName` property lets you specify the name of the target library for generated HDL code. The default target library name is `'work'`.

Output Pipelining Now Supported for Subsystems

The coder now supports the `OutputPipeline` property for subsystems.

For detailed information, see “OutputPipeline” and “DistributedPipelining” in the Simulink HDL Coder documentation.

Distributed Pipelining Now Supported for Subsystems

In the previous release, the coder supported the `DistributedPipelining` property for Embedded MATLAB® Function blocks or Stateflow® charts within a subsystem.. In R2010a, the coder also supports this property for any subsystem.

For detailed information, see “DistributedPipelining” in the Simulink HDL Coder documentation.

CSD and Factored CSD Optimizations for Constant Multiplications

You can now specify Canonic Signed Digit (CSD) and Factored Canonic Signed Digit (FCSD) techniques to optimize multiplication operations involving constants.

The `ConstMultiplierOptimization` implementation supports CSD and FCSD optimizations for the following blocks:

- Gain
- Stateflow chart
- Truth Table
- Embedded MATLAB

See also “ConstMultiplierOptimization”.

Enhanced Gain Block Support

The coder now supports the following for HDL code generation for the Gain block:

- Use of `Matrix (k*u) (u vector)` mode for the **Gain** parameter.
- If you specify the implementation parameter `ConstMultiplierOptimization, 'auto')` for the Gain block, the coder automatically selects CSD or FCSD implementations based on the number of required adders.

FIR Decimation Filter Supports Distributed Arithmetic Architecture

The code now supports distributed arithmetic (DA) filter implementations for the dspmlti4/FIR Decimation block. See "Distributed Arithmetic Implementation Parameters for Digital Filter Blocks" in the Simulink HDL Coder documentation for details.

Serial, Partly Serial and Cascade Serial Architectures Supported for FIR Filter Implementations

The coder now supports serial, partly serial and cascade serial architectures for the following blocks:

- dsparch4/Digital Filter (FIR structures only)
- simulink/Discrete/Discrete FIR Filter
- dspmlti4/FIR Decimation

You can specify serial architectures using the `SerialPartition` and `ReuseAccum` implementation parameters. See"Speed vs. Area Optimizations for FIR Filter Implementations" for further information.]

InstancePostfix Property Allows Specification of Extension to Postfix String

In R2010a, the coder supports the `InstancePostfix`. `InstancePostfix` lets you specify a string appended after component instance names in generated code. The default value for `InstancePostfix` is `''`(no postfix added).

Version 1.6 (R2009b) Simulink HDL Coder Software

This table summarizes what's new in Version 1.6 (R2009b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	None

New features and changes introduced in this version are:

- "AddClockEnablePort Implementation Parameter for RAM Blocks Deprecated" on page 61
- "Additional Lookup Table Blocks Supported" on page 61
- "Discrete FIR Filter Supports Distributed Arithmetic Architecture" on page 62
- "Generation of Multicycle Path Constraint Information" on page 62
- "Biquad Filter and Digital Filter Blocks Support Complex Input Data and Coefficients" on page 63
- "Support for Adding or Removing HDL Configuration Component" on page 64

Triggered Subsystems Support for HDL Code Generation

The coder now supports HDL code generation for triggered subsystems. See "Code Generation for Enabled and Triggered Subsystems" in the Simulink HDL Coder documentation for further information.

Stateflow Events Support for HDL Code Generation

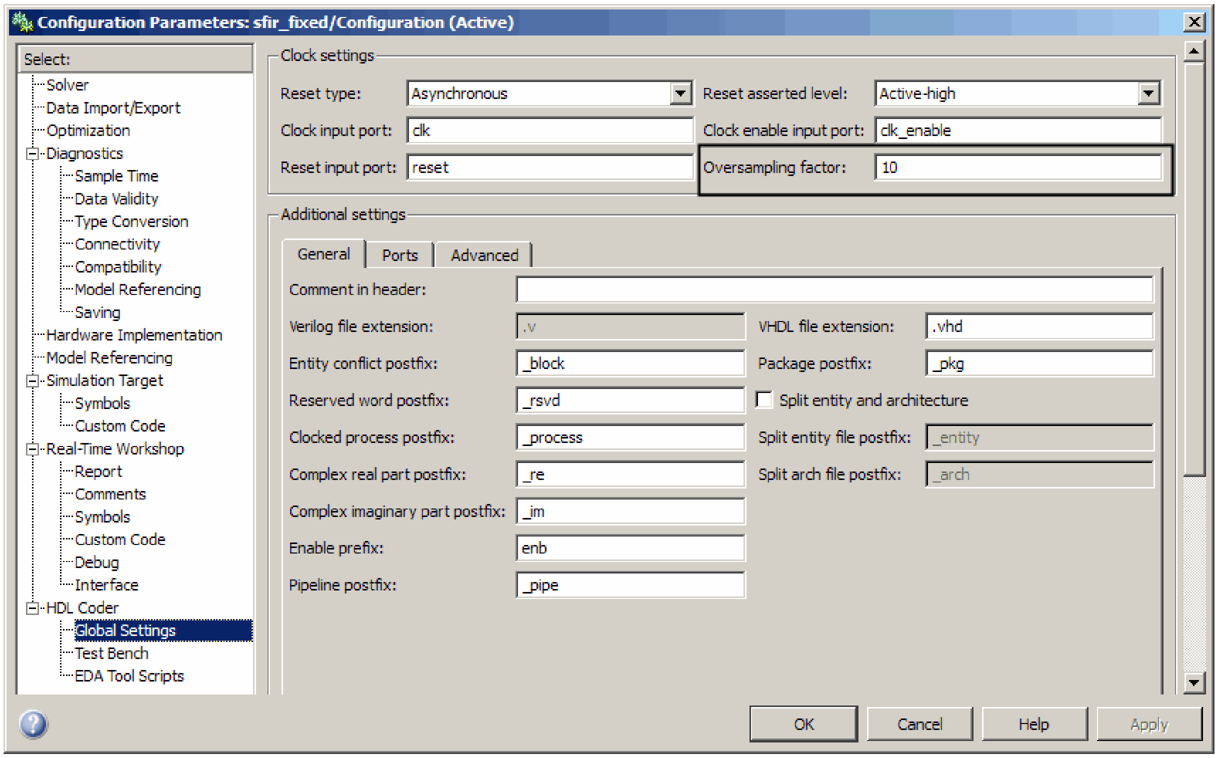
The coder now supports a single input event and unlimited output events in Stateflow charts. for further information, see "Using Input and Output Events" in the Simulink HDL Coder documentation.

Support for Global Oversampling Clock

You can now generate global clock logic that allows you to integrate your DUT into a larger system easily, without using Upsample or Downsample blocks.

To generate global clock logic, you specify an *oversampling factor*. The oversampling factor expresses the desired rate of the global oversampling clock as a multiple of the base rate of the model. When you specify an oversampling factor, the coder generates the global oversampling clock. Then, it derives the required timing signals from the clock signal. Generation of the global oversampling clock affects only generated HDL code. The clock does not affect the simulation behavior of your model.

You can specify the desired factor as the **Oversampling factor** option in the **Clock settings** section of the **Global Settings** pane of the Configuration Parameters dialog. The following figure shows the option. Alternatively, you can set the command-line property `'Oversampling'`.



Reset type:	Asynchronous	Reset asserted level:	Active-high
Clock input port:	clk	Clock enable input port:	clk_enable
Reset input port:	reset	Oversampling factor:	10

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

See “Generating a Global Oversampling Clock” in the Simulink HDL Coder documentation for further information.

Test Bench GUI Reorganized

The new **Testbench generation output** section of the GUI contains three new options:

- **HDL test bench**: Selecting this option enables generation of an HDL test bench, and also enables all options in the **Configuration** section of the **Test Bench** pane.
- **Cosimulation blocks**: Selecting this option enables generation of a model containing HDL Cosimulation block for use in testing the DUT. Selecting this option also enables all options in the **Configuration** section of the **Test Bench** pane.
- **Cosimulation model for use with**: This option enables generation of a model containing an HDL Cosimulation block for use in testing with a selected cosimulation tool. Selecting this option also enables all options in the **Configuration** section of the **Test Bench** pane.

To configure test bench options and generate test bench code, select one or more of the options of the **Testbench generation output** section. If you deselect all three options of the **Testbench generation output** section, the coder disables all options in the **Configuration** section of the **Test Bench** pane.

MATLAB Editor Supports VHDL and Verilog Syntax Highlighting

The MATLAB Editor now supports syntax highlighting for VHDL and Verilog code. See "Highlight Syntax to Help Ensure Correct Entries in the Command Window" in the MATLAB documentation for further information on syntax highlighting.

Hyperlinked Requirements Comments Included in HTML Code Generation Reports

The coder now renders requirements comments as hyperlinked comments within generated HTML code generation reports. See "Requirements Comments and Hyperlinks" in the Simulink HDL Coder documentation for further information.

HTML Code Generation Report from Root-Level Model Supported

In previous releases, the coder did not support generation of HTML code generation reports from the root-level model. R2009b removes this restriction. You can now generate reports for the root-level model as well as for subsystems, blocks, Stateflow charts, or Embedded MATLAB blocks.

Generation of Simulink Model for Cosimulation of Generated HDL Code

The coder now supports generation of a Simulink model configured for:

- Simulink simulation of your design
- Cosimulation of your design with an HDL simulator

The generated model includes a behavioral model of your design and a corresponding HDL Cosimulation block, configured to cosimulate the design using EDA Simulator Link. You can generate an HDL Cosimulation block for either of the following:

- EDA Simulator Link for use with Mentor GraphicsModelSim®
- EDA Simulator Link for use with Cadence Incisive®

See "Generating a Simulink Model for Cosimulation with an HDL Simulator" for further information.

Additional Simulink Blocks Supported for HDL Code Generation

The coder now supports the blocks listed in the following table for HDL code generation.

Block	Implementation
hdldemolib/HDL Streaming FFT	hdldefaults.FFT
Ports & Subsystems/Trigger	hdldefaults.TriggerPort
simulink/Discrete/Discrete FIR Filter	hdldefaults.DiscreteFIRFilterHDLInstantiation

Block	Implementation
simulink/Lookup Tables/Direct Lookup Table (n-D)	hdldefaults.DirectLookupTable
simulink/Lookup Tables/Lookup Table (n-D)	hdldefaults.LookupTableND
simulink/Lookup Tables/Prelookup	hdldefaults.PreLookup

"Summary of Block Implementations" in the Simulink HDL Coder documentation gives a complete listing of blocks that the coder supports for HDL code generation.

New hdldemolib Block Supports Streaming FFT

The new hdldemolib/HDL Streaming FFT block supports a Radix-2 DIF streaming FFT algorithm.

See "HDL Streaming FFT" in the Simulink HDL Coder documentation for details.

Algebraic Loops Disallowed for HDL Code Generation

The coder now checks for algebraic loops during the compatibility checking phase of the code generation process. If `makehdl` detects an algebraic loop inside the DUT, the coder displays an error message and ends the code generation process.

Compatibility Considerations

Restructure any of your models that contain algebraic loops such that algebraic loops do not occur. It is also good practice to set the **Algebraic loop** diagnostic in the **Diagnostics** pane of the Configuration Parameters dialog box to `error`.

DUT Argument Required for checkhdl and makehdl Commands

R2009b requires that calls to the following functions must specify the device under test (DUT):

- `checkhdl`
- `makehdl`

When you call `checkhdl` or `makehdl`, specify the DUT as the initial argument to these functions, as in the following example:

```
makehdl('sfir_fixed/symmetric_fir','TargetLanguage', 'Verilog');
```

As in previous releases, you can specify the DUT in any of the following forms:

- `bdroot`: the current model.
- `'modelname'`: an explicitly specified model.
- `'modelname/subsys'`: explicitly specified path to a subsystem.
- `gcb`: the currently selected subsystem

This requirement avoids certain ambiguities that occurred in calls to `checkhdl` or `makehdl` that did not pass in an explicit DUT argument.

In R2009b, the coder displays a warning if it encounters a call to `checkhdl` or `makehdl` without the DUT argument. In future releases, the coder will generate an error if it encounters a call to either of these functions without the DUT argument.

See also the checkhdl and makehdl function reference pages in the Simulink HDL Coder documentation.

Compatibility Considerations

If your MATLAB files contain any calls to `checkhdl` or `makehdl` that do not specify the DUT, modify them to pass in the DUT as the initial argument.

AddClockEnablePort Implementation Parameter for RAM Blocks Deprecated

The `AddClockEnablePort` implementation parameter for the Dual Port RAM and Single Port RAM blocks is deprecated. The coder issues an error message if it detects a reference to `AddClockEnablePort` in a control file.

Compatibility Considerations

If you use the `AddClockEnablePort` in a control file to suppress to generation of a clock enable signal for RAM blocks:

- Remove all references to `AddClockEnablePort` from your control files.
- Use the generic RAM templates instead. The generic RAM templates do not use a clock enable signal for RAM structures. The generic RAM template implements clock enable with logic in a wrapper around the RAM. Consider the generic RAM style if
  - Your synthesis tool does not support RAM structures with a clock enable
  - Your synthesis tool cannot map generated HDL code to FPGA RAM resources.

To learn how to use generic style RAM for your design, see the new Getting Started with RAM and ROM in Simulink demo. To open the demo, type the following command at the MATLAB prompt:

```
hdlcoderramrom
```

Additional Lookup Table Blocks Supported

The coder now supports the following lookup table (LUT) blocks for HDL code generation:

- simulink/Lookup Tables/Lookup Table (n-D)
- simulink/Lookup Tables/Prelookup
- simulink/Lookup Tables/Direct Lookup Table (n-D)

Expanded LUT functionality supported for these blocks includes:

- Tables of two dimensions

- Prelookup
- Interpolation
- Extrapolation

See "Support for Lookup Table Blocks in HDL Code Generation" in the Simulink HDL Coder documentation for details.

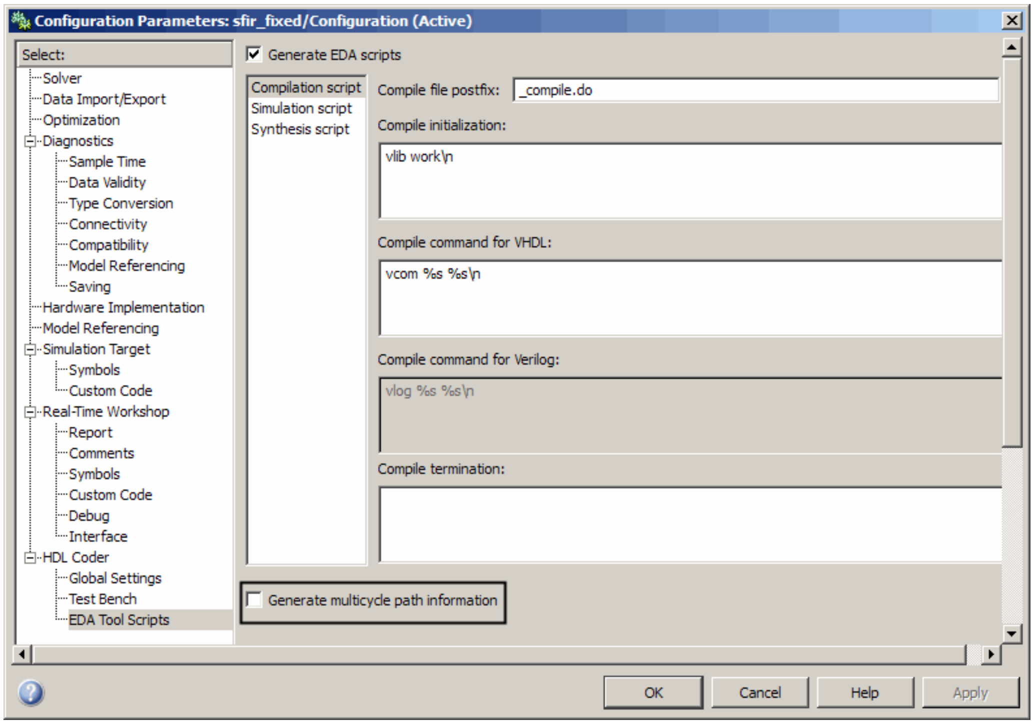
Discrete FIR Filter Supports Distributed Arithmetic Architecture

The code now supports distributed arithmetic (DA) filter implementations for the Discrete FIR Filter block. See "Distributed Arithmetic Implementation Parameters for Digital Filter Blocks" in the Simulink HDL Coder documentation for details.

Generation of Multicycle Path Constraint Information

The coder now supports generation of a text file that reports multicycle path constraint information. You can use this information with your synthesis tool.

To generate the file, select the **Generate multicycle path information** option in the **EDA Tool Scripts** pane of the Configuration Parameters dialog box. The following figure shows this option.



[✓] 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:

[ ] Generate multicycle path information

To generate a multicycle path constraint information file at the command line, set the `MulticyclePathInfo` property as shown in the following example.

```
makehdl(gcb,'MulticyclePathInfo', 'on');
```

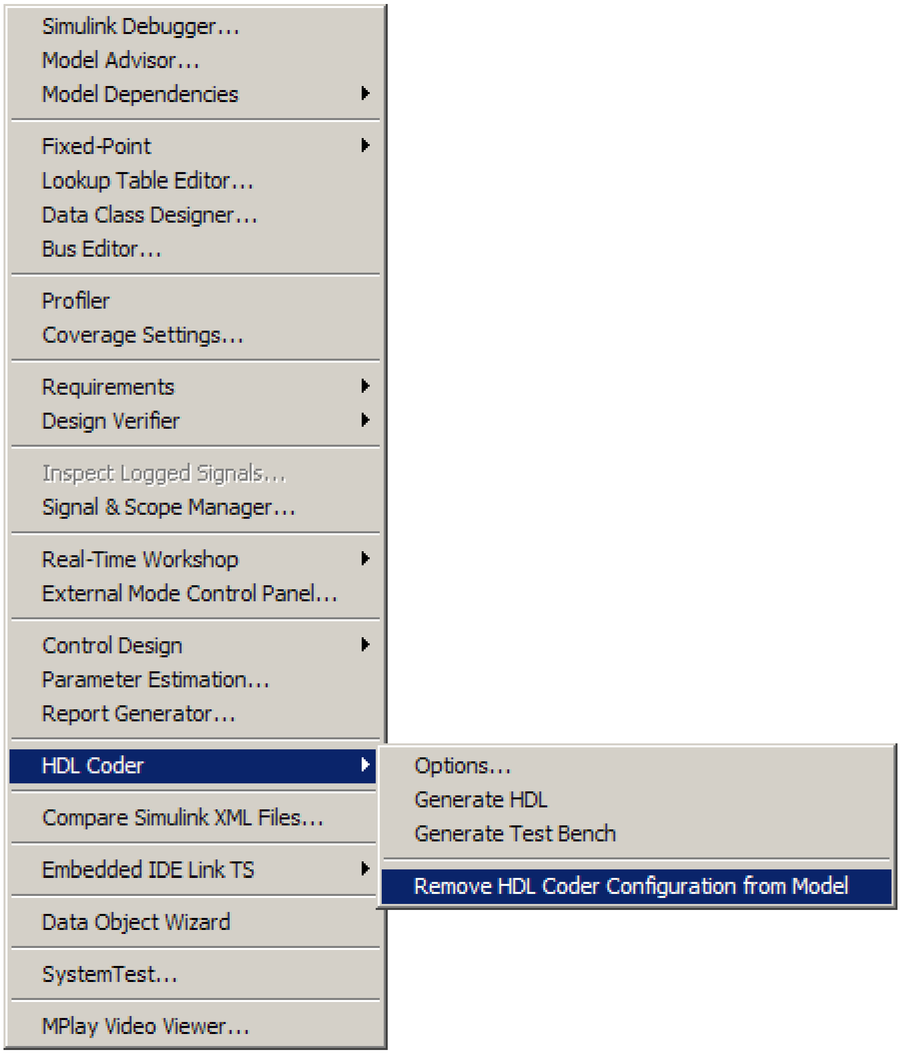
See "Generating Multicycle Path Information Files" in the Simulink HDL Coder documentation for detailed information.

Biquad Filter and Digital Filter Blocks Support Complex Input Data and Coefficients

The Biquad Filter and Digital Filter blocks now support complex input data and coefficients for all filter structures except decimators and interpolators.

Support for Adding or Removing HDL Configuration Component

The **HDL Coder** submenu of the **Tools** menu now supports addition or removal of the HDL Coder configuration component of a model. The following figure shows the **Remove HDL Configuration to Model** option.



See “Adding and Removing the HDL Configuration Component” Simulink HDL Coder documentation for more information.

Version 1.5 (R2009a) Simulink HDL Coder Software

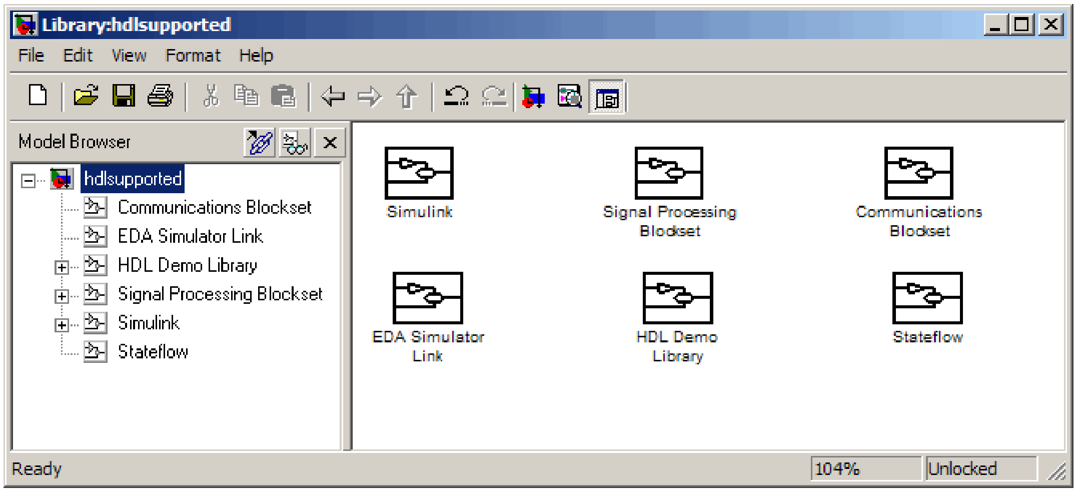
This table summarizes what's new in Version 1.5 (R2009a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	None

New features and changes introduced in this version are:

hdlsupported Library Reorganized

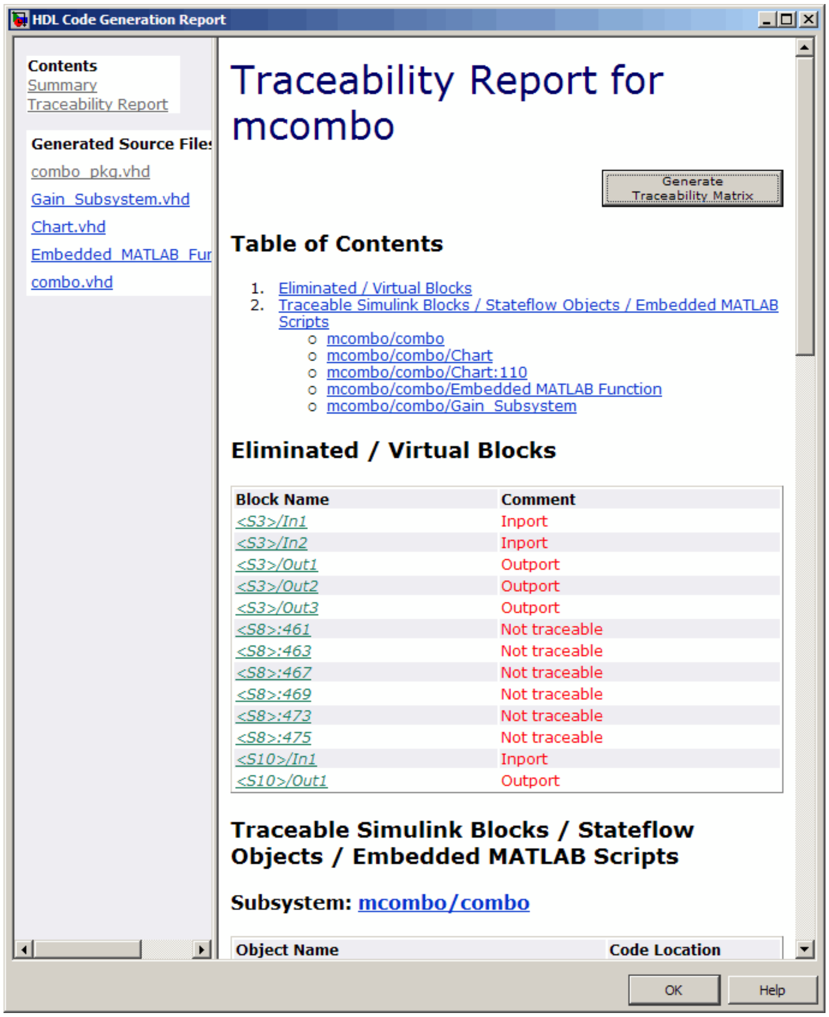
The `hdlsupported.mdl` block library has been reorganized into several sublibraries to help you locate the HDL-compatible blocks you need more easily. The following figure shows the top-level view of the `hdlsupported.mdl` library.



The set of supported blocks will change in future releases of the coder. To keep the `hdlsupported.mdl` current, you should rebuild the library each time you install a new release. See “Supported Blocks Library” in the Simulink HDL Coder documentation for more information.

HTML Code Generation Report

To help you navigate more easily between generated code and your source model, the coder provides a *traceability* option that lets you generate reports from either the GUI or the command line. When you enable traceability, the coder creates and displays an HTML code generation report during the code generation process. The following figure shows the top-level page of a typical report.



Contents
Summary
Traceability Report

Generated Source Files
combo_pkg.vhd
Gain_Subsystem.vhd
Chart.vhd
Embedded_MATLAB_Fur
combo.vhd

Traceability Report for mcombo

Table of Contents

1. Eliminated / Virtual Blocks
2. Traceable Simulink Blocks / Stateflow Objects / Embedded MATLAB Scripts
   - mcombo/combo
   - mcombo/combo/Chart
   - mcombo/combo/Chart:110
   - mcombo/combo/Embedded MATLAB Function
   - mcombo/combo/Gain_Subsystem

Eliminated / Virtual Blocks

Block Name	Comment
<S3>/In1	Inport
<S3>/In2	Inport
<S3>/Out1	Outport
<S3>/Out2	Outport
<S3>/Out3	Outport
<S8>:461	Not traceable
<S8>:463	Not traceable
<S8>:467	Not traceable
<S8>:469	Not traceable
<S8>:473	Not traceable
<S8>:475	Not traceable
<S10>/In1	Inport
<S10>/Out1	Outport

Traceable Simulink Blocks / Stateflow Objects / Embedded MATLAB Scripts

Subsystem: mcombo/combo

Object Name	Code Location

The report comprises several sections:

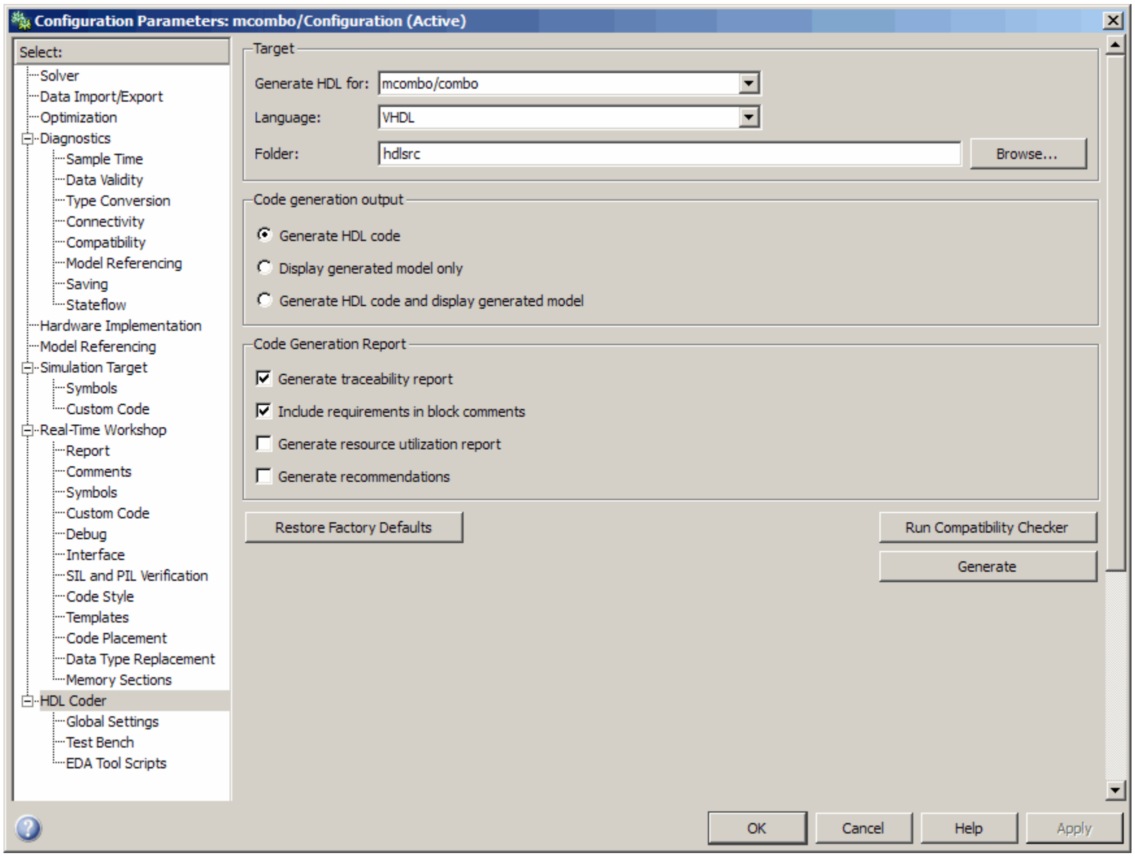
- The **Summary** section lists version and date information.
- The **Generated Source Files** table contains hyperlinks to that let you view generated HDL code in a MATLAB Web browser window. This view of the code includes hyperlinks that let you view the blocks or subsystems from which the code was generated. You can click the names of source code

files generated from your model to view their contents in a MATLAB Web browser window. The report supports two types of linkage between the model and generated code:

  - Code-to-model hyperlinks within the displayed source code let you view the blocks or subsystems from which the code was generated. Click on the hyperlinks to view the relevant blocks or subsystems in a Simulink model window.
  - Model-to-code linkage lets you view the generated code for any block in the model. To highlight a block's generated code in the HTML report, right-click the block and select **HDL Coder > Navigate to Code** from the context menu.

- The **Traceability Report** allows you to account for **Eliminated / Virtual Blocks** that are untraceable, versus the listed **Traceable Simulink Blocks / Stateflow Objects / Embedded MATLAB Scripts**, providing a complete mapping between model elements and code.

To enable generation of the HTML code generation report, select **Generate traceability report** in the **HDL Coder** pane of the Configuration Parameters dialog box, as shown in the following figure.



Target

Generate HDL for: mcombo/combo

Language: VHDL

Folder: hdlsrc

Code generation output

[✓ Generate HDL code]

Display generated model only

Generate HDL code and display generated model

Code Generation Report

[✓] Generate traceability report

[✓] Include requirements in block comments

[ ] Generate resource utilization report

[ ] Generate recommendations

See "Creating and Using Code Generation Reports" in the Simulink HDL Coder documentation for further information.

Additional Simulink Blocks Supported for HDL Code Generation

The coder now supports the blocks listed in the following table for HDL code generation.

Block	Implementation(s)
simulink/Additional Math & Discrete/ Additional Math: Increment - Decrement/Decrement Real World	default
simulink/Additional Math & Discrete/ Additional Math: Increment - Decrement/Increment Real World	default
simulink/Additional Math & Discrete/ Additional Math: Increment - Decrement/Decrement Store Integer	default
simulink/Additional Math & Discrete/ Additional Math: Increment - Decrement/Increment Store Integer	default
simulink/Discontinuities/Saturation Dynamic	default
simulink/Math Operations/Reciprocal Sqrt	default, SqrtFunction RecipSqrtNewton SqrtBitset SqrtNewton
Signal Routing/Go To	default
Signal Routing/From	default
dsparch4/Biquad Filter	default
Ports & Subsystems/Enable	default

Enabled Subsystems Supported for HDL Code Generation

The code now supports code generation for enabled subsystems, provided that they are configured as described in “Code Generation for Enabled and Triggered Subsystems” in the Simulink HDL Coder documentation.

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 R2009a	New Default Implementation
simulink/Commonly Used Blocks/Constant simulink/Commonly Used Blocks/Ground dspsrcs4/DSP Constant	ConstantHDLEmission	Constant
simulink/Commonly Used Blocks/Demux	DemuxHDLEmission	Demux
simulink/Commonly Used Blocks/Mux	MuxHDLEmission	Mux
simulink/Commonly Used Blocks/Switch	SwitchHDLEmission	SwitchRTW
simulink/Math Operations/Complex to Real-Imag	ComplexToRealImagHDLEmission	ComplexToRealImag
simulink/Math Operations/Real-Imag to Complex	RealImagtoComplexHDLEmission	RealImagtoComplex

See the Simulink HDL Coder documentation for a complete listing of blocks that are currently supported for HDL code generation.

Compatibility Considerations

If your models use default HDL block implementations for the affected blocks, the coder now defaults 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.

New HDL Implementations for Selected Blocks

A number of HDL block implementations have been changed. The following table lists these blocks, as well as their new implementations and the earlier implementations that they replace. All listed implementation classes belong to the package `hdldefaults`.

Block	Implementation Before R2009a	New Implementation
simulink/Math Operations/MinMax dspstat3/Maximum dspstat3/Minimum	MinMaxCascadeHDLEmission	MinMaxCascade
simulink/Commonly Used Blocks/Sum simulink/Math Operations/Sum of Elements	SumTreeHDLEmission	SumTree
simulink/Commonly Used Blocks/Product simulink/Math Operations/Product of Elements	ProductTreeHDLEmission	ProductTree
simulink/Commonly Used Blocks/Sum simulink/Math Operations/Sum of Elements	SumCascadeHDLEmission	SumCascade
simulink/Commonly Used Blocks/Product simulink/Math Operations/Product of Elements	ProductCascadeHDLEmission	ProductCascade

See the Simulink HDL Coder documentation for a complete listing of blocks that are currently supported for HDL code generation.

Compatibility Considerations

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 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.

Distributed Arithmetic Implementations for the Digital Filter Block

Distributed Arithmetic (DA) is a widely used technique for implementing sum-of-products computations without using multipliers. DA distributes multiply and accumulate operations across shifters, lookup tables (LUTs) and adders in such a way that conventional multipliers are not required. The coder now supports DA implementations for the following FIR structures of the Digital Filter block:

- `dfilt.dffir`
- `dfilt.dfsymfir`
- `dfilt.dfasymdir`

See "Block Implementation Parameters" in the Simulink HDL Coder documentation for further information.

Complex Data Supported for the Digital Filter Block

The coder supports complex coefficients and complex input signals for fully parallel FIR and CIC filter structures of the Digital Filter block. In many cases, you can use complex data and complex coefficients in combination. The following table shows the filter structures that support complex data and/or coefficients, and the permitted combinations.

Filter Structure	Complex Data	Complex Coefficients	Both Complex Data and Coefficients
`dfilt.dffir`	Y	Y	Y
`dfilt.dfsymfir`	Y	Y	Y
`dfilt.dfasymfir`	Y	Y	Y
`dfilt.dffirt`	Y	Y	Y
`mfilt.cicdecim`	Y	N/A	N/A
`mfilt.cicinterp`	Y	N/A	N/A
`mfilt.firdecim`	Y	Y	N
`mfilt.firinterp`	Y	Y	N

See “Blocks That Support Complex Data” for further information on how the coder supports use of complex data.

Requirements Comments Included in Generated Code

Requirements that you assign to Simulink blocks are now automatically included as comments in generated code. See the *Simulink® Verification and Validation™ User’s Guide* in the Simulink HDL Coder documentation for further information on requirements comments.

Restriction on fi and fimath Rounding Modes in Embedded MATLAB Function Block Removed

In previous releases, the coder did not support the `convergent` and `round` modes for the `fi` and `fimath` functions in Embedded MATLAB Function blocks.

This restriction has been removed; the coder now supports all `fi` and `fimath` rounding modes.

See also “Generating HDL Code with the MATLAB Function Block” in the Simulink HDL Coder documentation.

Restriction on for Loop Increment in Embedded MATLAB Function Block Removed

In previous releases, the use of `for` loops with an increment other than `1` in an Embedded MATLAB Function Block was not supported for HDL code generation.

This restriction has been removed. The coder now allows use of any increment in a `for` loop in an Embedded MATLAB Function Block.

See also "Generating HDL Code with the MATLAB Function Block" in the Simulink HDL Coder documentation.

Generic RAM Template Supports RAM Without a Clock Enable Signal

The `hdldemolib` library provides three type of RAM blocks:

- Dual Port RAM
- Simple Dual Port RAM
- Single Port RAM

These blocks (see "RAM Blocks" in the Simulink HDL Coder documentation) implement RAM structures using HDL templates that include a clock enable signal.

However, some synthesis tools do not support RAM inference with a clock enable. As an alternative, the coder now provides a generic style of HDL templates that do not use a clock enable signal for the RAM structures. The generic RAM template implements clock enable with logic in a wrapper around the RAM.

You may want to use the generic RAM style if your synthesis tool does not support RAM structures with a clock enable, and cannot map generated HDL code to FPGA RAM resources. To learn how to use generic style RAM for your design, see the new Getting Started with RAM and ROM in Simulink demo. To open the demo, type the following command at the MATLAB prompt:

```
hdlcoderramrom
```

Generating ROM with Lookup Table and Unit Delay Blocks

Simulink HDL Coder does not provide a ROM block, but you can easily build one using basic Simulink blocks. The new Getting Started with RAM and ROM in Simulink demo includes an example in which a ROM is built using a Lookup Table block and a Unit Delay block. To open the demo, type the following command at the MATLAB prompt:

```
hdlcoderramrom
```

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 V2.2 (R2011b)	See the **Compatibility Considerations** subheading for these new features or changes: • "Delay Balancing Now Enabled By Default" on page 12 • "Conversion of Error and Warning Message Identifiers" on page 16 • "Functions and Function Elements Being Removed" on page 21
V2.1 (R2011a)	None
V2.0 (R2010b)	See the **Compatibility Considerations** subheading for this new feature or change: • "HDL Parameters Now Saved to Model, Eliminating Need For Control Files" on page 32
V1.7 (R2010a)	See the **Compatibility Considerations** subheading for this new feature or change: • "Simplified Syntax for Specification of Block Implementations in Control Files" on page 47

Version (Release)	New Features and Changes with Version Compatibility Impact
V1.6 (R2009b)	See the **Compatibility Considerations** subheading for these new features or changes: • "DUT Argument Required for checkhdl and makehdl Commands" on page 60 • "Algebraic Loops Disallowed for HDL Code Generation" on page 59 • "AddClockEnablePort Implementation Parameter for RAM Blocks Deprecated" on page 61
V1.5 (R2009a)	See the **Compatibility Considerations** subheading for these new features or changes: • "New Default HDL Implementations for Selected Blocks" on page 72 • "New HDL Implementations for Selected Blocks" on page 73