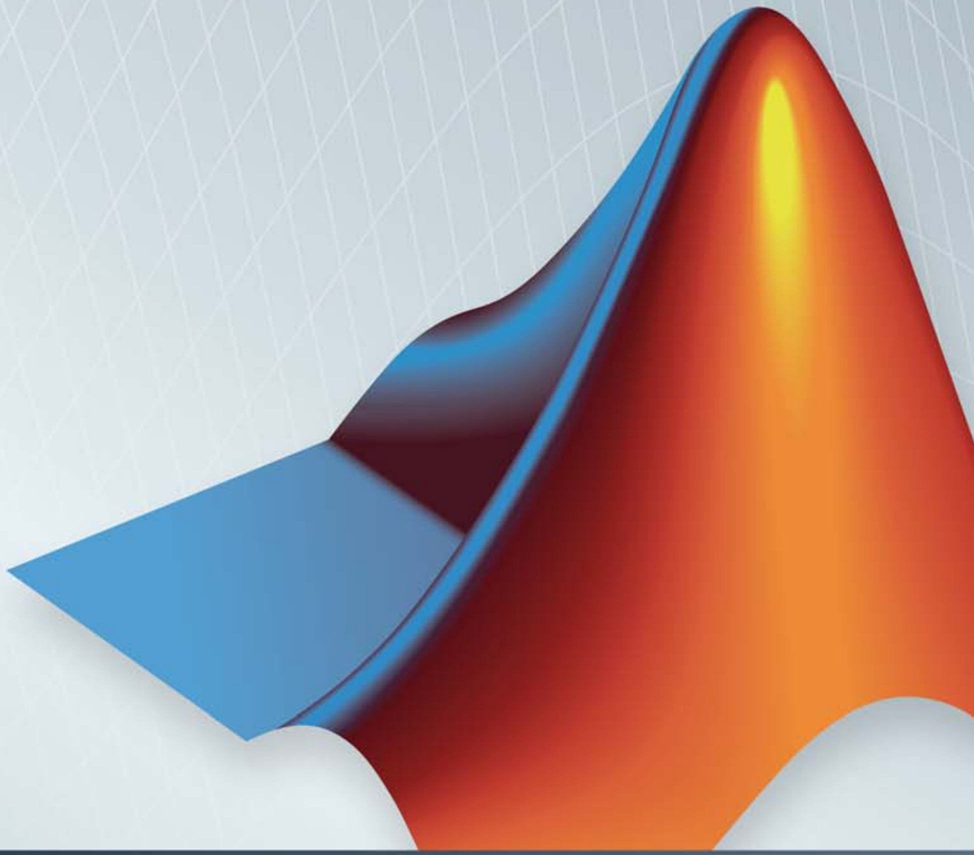
HDL Coder™

Reference

R2014**a**



MATLAB®

MathWorks®

Revision History

March 2013	Online only	New for Version 3.2 (R2013a)
September 2013	Online only	Revised for Version 3.3 (R2013b)
March 2014	Online only	Revised for Version 3.4 (Release 2014a)

Contents

Functions — Alphabetical List

checkhdl

Purpose

Check subsystem or model for HDL code generation compatibility

Syntax

```
checkhdl(bdroot)
checkhdl('modelname')
checkhdl('subsysname')
checkhdl(gcb)
output = checkhdl('system')
```

Description

`checkhdl` generates an HDL Code Generation Check Report, saves the report to the target folder, and displays the report in a new window. Before generating HDL code, use `checkhdl` to check your subsystems or models.

The report lists compatibility errors with a link to each block or subsystem that caused a problem. To highlight and display incompatible blocks, click each link in the report while keeping the model open.

The report file name is `*system*_report.html`. `*system*` is the name of the subsystem or model passed in to `checkhdl`.

When a model or subsystem passes `checkhdl`, that does not imply code generation will complete. `checkhdl` does not verify all block parameters.

`checkhdl(bdroot)` examines the current model for HDL code generation compatibility.

`checkhdl('*modelname*')` examines the specified model, `*modelname*`.

`checkhdl('*subsysname*')` examines a subsystem. `*subsysname*` is the full block path for a subsystem at any level of the model hierarchy.

`checkhdl(gcb)` examines the currently selected subsystem.

```
output = checkhdl('system')
```

does not generate a report. Instead, it returns a `1xN` struct array with one entry for each error, warning, or message. `*system*` specifies a model or the full block path for a subsystem at any level of the model hierarchy.

`checkhdl` reports three levels of compatibility problems:

- *Errors*: cause the code generation process to terminate. The report must not contain errors to continue with HDL code generation.
- *Warnings*: indicate problems in the generated code, but allow HDL code generation to continue.
- *Messages*: indication that some data types have special treatment. For example, the coder automatically converts single-precision floating-point data types to double-precision because VHDL® and Verilog® do not support single-precision data types.

Examples

Check the subsystem `symmetric_fir` within the model `sfir_fixed` for HDL code generation compatibility and generate a compatibility report.

```
checkhdl('sfir_fixed/symmetric_fir')
```

Check the subsystem `symmetric_fir_err` within the model `sfir_fixed_err` for HDL code generation compatibility, and return information on problems encountered in the struct `output`.

```
output = checkhdl('sfir_fixed_err/symmetric_fir_err')
### Starting HDL Check.
...
### HDL Check Complete with 4 errors, warnings and messages.
```

The following MATLAB® commands display the top-level structure of the struct `output`, and its first cell.

```
output =

1x4 struct array with fields:
    path
    type
    message
    level


output(1)
```

```
ans = 

       path: 'sfir_fixed_err/symmetric_fir_err/Product'
       type: 'block'
    message: 'Unhandled mixed double and non-double datatypes at ports of block'
      level: 'Error'
```

See Also

`makehdl`

Tutorials

- “Selecting and Checking a Subsystem for HDL Compatibility”

Purpose

Display HDL Workflow Advisor

Syntax

```
hdladvisor(gcb)
hdladvisor(subsystem)
hdladvisor(model,'SystemSelector')
```

Description

`hdladvisor(gcb)` starts the HDL Workflow Advisor, passing the currently selected subsystem within the current model as the DUT to be checked.

`hdladvisor(`*`subsystem`*`)` starts the HDL Workflow Advisor, passing in the path to a specified subsystem within the model.

`hdladvisor(`*`model`*`,'SystemSelector')` opens a System Selector window that lets you select a subsystem to be opened into the HDL Workflow Advisor as the device under test (DUT) to be checked.

Examples

Open the subsystem `symmetric_fir` within the model `sfir_fixed` into the HDL Workflow Advisor.

```
hdladvisor('sfir_fixed/symmetric_fir')
```

Open a System Selector window to select a subsystem within the current model. Then open the selected subsystem into the HDL Workflow Advisor.

```
hdladvisor(gcs,'SystemSelector')
```

Alternatives

You can also open the HDL Workflow Advisor from the your model window by selecting **Code > HDL Code > HDL Workflow Advisor**.

See Also

"What Is the HDL Workflow Advisor?" | "Using the HDL Workflow Advisor Window"

hdlapplycontrolfile

Purpose

Apply control file settings to model

Note `hdlapplycontrolfile` is not recommended. Use `hdlset_param` and `hdlget_param` instead.

Syntax

```
hdlapplycontrolfile(modelname, controlfilename)
hdlapplycontrolfile(dutname, controlfilename)
```

Description

`hdlapplycontrolfile(modelname, controlfilename)` applies the settings in the specified control file to the specified model.

`hdlapplycontrolfile(dutname, controlfilename)` applies the settings in the specified control file to a specified subsystem (the device under test, or DUT) within the current model.

Tips

- As of release R2010b, use of control files is not recommended, and the coder does not support the attachment of a control file to a new model. Instead, the coder now saves non-default block implementation and implementation parameter settings to the model itself. This eliminates the need to load and save a separate control file. The coder provides the `hdlapplycontrolfile` utility as a quick way to transfer HDL settings from existing models that have attached control files to other models.
- After you apply control file settings to a model, be sure to save the model.
- If you have existing models with attached control files, you should convert them to the current format. To do this, simply open the model and save it. Saving a model clears its attachment to its control file, but the control file itself is preserved so that you can apply it to other models if you wish.

  For backward compatibility, the coder continues to support models that have attached control files. See "READ THIS FIRST: Control File Compatibility and Conversion Issues" for further information.

- Some control files are designed to be generic, and do not specify the DUT using `generateHDLFor`. To apply settings from such a control file, you must supply a full path to the desired DUT using the `dutname` argument.

Input Arguments

modelname

Name of the target model, to which control file settings are applied.

Default: None

controlfilename

Name of the control file containing hdl settings to be applied

Default: None

dutname

Full path to the top-level subsystem (the device under test or DUT) within the target model.

Default: None

Examples

Apply settings from `sfir_fixed_control.m` to the open model `sfir_fixed_newVersion`.

```
hdlapplycontrolfile('sfir_fixed_newVersion','sfir_fixed_control.m')
Successfully loaded control file 'sfir_fixed_control.m' ...
```

Apply settings from `sfir_fixed_control.m` to the subsystem `symmetric_fir` within the open model`sfir_fixed_newVersion`.

```
  hdlapplycontrolfile('sfir_fixed_newVersion/symmetric_fir','sfir_fixed_control.m')
Successfully loaded control file 'sfir_fixed_control.m' ...
```

See Also

| "READ THIS FIRST: Control File Compatibility and Conversion Issues"

Purpose

Automatic iterative HDL design optimization

Syntax

```
hdlcoder.optimizeDesign(model, optimizationCfg)
hdlcoder.optimizeDesign(model, cpGuidanceFile)
```

Description

`hdlcoder.optimizeDesign(model, optimizationCfg)` automatically optimizes your generated HDL code based on the optimization configuration you specify.

`hdlcoder.optimizeDesign(model, cpGuidanceFile)` regenerates the optimized HDL code without rerunning the iterative optimization, by using data from a previous run of `hdlcoder.optimizeDesign`.

Input Arguments

model - Model name

string

Model name, specified as a string.

Example: `'sfir_fixed'`

optimizationCfg - Optimization configuration

`hdlcoder.OptimizationConfig`

Optimization configuration, specified as an `hdlcoder.OptimizationConfig` object.

cpGuidanceFile - File containing saved optimization data

`''` (default) | string

File that contains saved data from the final optimization iteration, including relative path, specified as a string. Use this file to regenerate optimized code without rerunning the iterative optimization.

The file name is `cpGuidance.mat`. You can find the file in the iteration folder name that starts with `Final`, which is a subfolder of `hdlexpl`.

Example:
`'hdlexpl/Final-11-Dec-2013-23-17-10/cpGuidance.mat'`

Examples

Maximize clock frequency

Maximize the clock frequency for a model, `sfir_fixed`, by performing up to 10 optimization iterations.

Open the model and specify the DUT subsystem.

```
model = 'sfir_fixed';
dutSubsys = 'symmetric_fir';
open_system(model);
hdlset_param(model,'HDLSubsystem',[model,'/',dutSubsys]);
```

Set your synthesis tool and target device options.

```
hdlset_param (model,'SynthesisTool','Xilinx ISE', ...
                    'SynthesisToolChipFamily','Zynq', ...
                    'SynthesisToolDeviceName','xc7z030', ...
                    'SynthesisToolPackageName','fbg484', ...
                    'SynthesisToolSpeedValue','-3')
```

Enable HDL test bench generation.

```
hdlset_param(model,'GenerateHDLTestBench','on');
```

Save your model.

You must save your model if you want to regenerate code later without rerunning the iterative optimizations, or resume your run if it is interrupted. When you use `hdlcoder.optimizeDesign` to regenerate code or resume an interrupted run, the coder checks the model checksum and generates an error if the model has changed.

Create an optimization configuration object, `oc`.

```
oc = hdlcoder.OptimizationConfig;
```

Set the iteration limit to 10.

```
oc.IterationLimit = 10;
```

Optimize the model.

```
hdlcoder.optimizeDesign(model,oc)
```

```
hdlset_param('sfir_fixed', 'HDLSubsystem', 'sfir_fixed/symmetric_fir'
hdlset_param('sfir_fixed', 'SynthesisTool', 'Xilinx ISE');
hdlset_param('sfir_fixed', 'SynthesisToolChipFamily', 'Zynq');
hdlset_param('sfir_fixed', 'SynthesisToolDeviceName', 'xc7z030');
hdlset_param('sfir_fixed', 'SynthesisToolPackageName', 'fbg484');
hdlset_param('sfir_fixed', 'SynthesisToolSpeedValue', '-3');

Iteration 0
Generate and synthesize HDL code ...
(CP ns) 16.26 (Constraint ns) 5.85 (Elapsed s) 143.66 Iteration 1
Generate and synthesize HDL code ...
(CP ns) 16.26 (Constraint ns) 5.85 (Elapsed s) 278.72 Iteration 2
Generate and synthesize HDL code ...
(CP ns) 10.25 (Constraint ns) 12.73 (Elapsed s) 427.22 Iteration 3
Generate and synthesize HDL code ...
(CP ns) 9.55 (Constraint ns) 9.73 (Elapsed s) 584.37 Iteration 4
Generate and synthesize HDL code ...
(CP ns) 9.55 (Constraint ns) 9.38 (Elapsed s) 741.04 Iteration 5
Generate and synthesize HDL code ...
Exiting because critical path cannot be further improved.
Summary report: summary.html
Achieved Critical Path (CP) Latency : 9.55 ns  Elapsed : 741.04 s
Iteration 0: (CP ns) 16.26 (Constraint ns) 5.85 (Elapsed s) 143.66
Iteration 1: (CP ns) 16.26 (Constraint ns) 5.85 (Elapsed s) 278.72
Iteration 2: (CP ns) 10.25 (Constraint ns) 12.73 (Elapsed s) 427.22
Iteration 3: (CP ns) 9.55 (Constraint ns) 9.73 (Elapsed s) 584.37
Iteration 4: (CP ns) 9.55 (Constraint ns) 9.38 (Elapsed s) 741.04
Final results are saved in
    /tmp/hdlsrc/sfir_fixed/hdlexpl/Final-07-Jan-2014-17-04-41
Validation model: gm_sfir_fixed_vnl
```

Then coder stops after five iterations because the fourth and fifth iterations had the same critical path, which indicates that the coder has found the minimum critical path. The design's maximum clock frequency after optimization is 1 / 9.55 ns, or 104.71 MHz.

Optimize for specific clock frequency

Optimize a model, `sfir_fixed`, to a specific clock frequency, 50 MHz, by performing up to 10 optimization iterations, and do not generate an HDL test bench.

Open the model and specify the DUT subsystem.

```
model = 'sfir_fixed';
dutSubsys = 'symmetric_fir';
open_system(model);
hdlset_param(model,'HDLSubsystem',[model,'/',dutSubsys]);
```

Set your synthesis tool and target device options.

```
hdlset_param (model,'SynthesisTool','Xilinx ISE', ...
                    'SynthesisToolChipFamily','Zynq', ...
                    'SynthesisToolDeviceName','xc7z030', ...
                    'SynthesisToolPackageName','fbg484', ...
                    'SynthesisToolSpeedValue','-3')
```

Disable HDL test bench generation.

```
hdlset_param(model,'GenerateHDLTestBench','off');
```

Save your model.

You must save your model if you want to regenerate code later without rerunning the iterative optimizations, or resume your run if it is interrupted. When you use `hdlcoder.optimizeDesign` to regenerate code or resume an interrupted run, the coder checks the model checksum and generates an error if the model has changed.

Create an optimization configuration object, `oc`.

```
oc = hdlcoder.OptimizationConfig;
```

Configure the automatic iterative optimization to stop after it reaches a clock frequency of 50MHz, or 10 iterations, whichever comes first.

```
oc.ExplorationMode = ...
    hdlcoder.OptimizationConfig.ExplorationMode.TargetFrequency;
oc.TargetFrequency = 50;
oc.IterationLimit = 10; =
```

Optimize the model.

```
hdlcoder.optimizeDesign(model,oc)
```

```
hdlset_param('sfir_fixed', 'GenerateHDLTestBench', 'off');
hdlset_param('sfir_fixed', 'HDLSubsystem', 'sfir_fixed/symmetric_fir')
hdlset_param('sfir_fixed', 'SynthesisTool', 'Xilinx ISE');
hdlset_param('sfir_fixed', 'SynthesisToolChipFamily', 'Zynq');
hdlset_param('sfir_fixed', 'SynthesisToolDeviceName', 'xc7z030');
hdlset_param('sfir_fixed', 'SynthesisToolPackageName', 'fbg484');
hdlset_param('sfir_fixed', 'SynthesisToolSpeedValue', '-3');

Iteration 0
Generate and synthesize HDL code ...
(CP ns) 16.26 (Constraint ns) 20.00 (Elapsed s) 134.02 Iteration 1
Generate and synthesize HDL code ...
Exiting because constraint (20.00 ns) has been met (16.26 ns).
Summary report: summary.html
Achieved Critical Path (CP) Latency : 16.26 ns  Elapsed : 134.02 s
Iteration 0: (CP ns) 16.26 (Constraint ns) 20.00 (Elapsed s) 134.02
Final results are saved in
    /tmp/hdlsrc/sfir_fixed/hdlexpl/Final-07-Jan-2014-17-07-14
Validation model: gm_sfir_fixed_vnl
```

Then coder stops after one iteration because it has achieved the target clock frequency. The critical path is 16.26 ns, a clock frequency of 61.50 GHz.

Resume clock frequency optimization using saved data

Run additional optimization iterations for a model, `sfir_fixed`, using saved iteration data, because you terminated in the middle of a previous run.

Open the model and specify the DUT subsystem.

```
model = 'sfir_fixed';
dutSubsys = 'symmetric_fir';
open_system(model);
hdlset_param(model,'HDLSubsystem',[model,'/',dutSubsys]);
```

Set your synthesis tool and target device options to the same values as in the interrupted run.

```
hdlset_param (model,'SynthesisTool','Xilinx ISE', ...
                    'SynthesisToolChipFamily','Zynq', ...
                    'SynthesisToolDeviceName','xc7z030', ...
                    'SynthesisToolPackageName','fbg484', ...
                    'SynthesisToolSpeedValue','-3')
```

Enable HDL test bench generation.

```
hdlset_param(model,'GenerateHDLTestBench','on');
```

Create an optimization configuration object, `oc`.

```
oc = hdlcoder.OptimizationConfig;
```

Configure the automatic iterative optimization to run using data from the first iteration of a previous run.

```
oc.ResumptionPoint = 'Iter5-07-Jan-2014-17-04-29';
```

Optimize the model.

```
hdlcoder.optimizeDesign(model,oc)
```

```
hdlset_param('sfir_fixed', 'HDLSubsystem', 'sfir_fixed/symmetric_fir');
hdlset_param('sfir_fixed', 'SynthesisTool', 'Xilinx ISE');
hdlset_param('sfir_fixed', 'SynthesisToolChipFamily', 'Zynq');
hdlset_param('sfir_fixed', 'SynthesisToolDeviceName', 'xc7z030');
hdlset_param('sfir_fixed', 'SynthesisToolPackageName', 'fbg484');
hdlset_param('sfir_fixed', 'SynthesisToolSpeedValue', '-3');

Try to resume from resumption point: Iter5-07-Jan-2014-17-04-29
Iteration 5
Generate and synthesize HDL code ...
Exiting because critical path cannot be further improved.
Summary report: summary.html
Achieved Critical Path (CP) Latency : 9.55 ns  Elapsed : 741.04 s
Iteration 0: (CP ns) 16.26 (Constraint ns) 5.85 (Elapsed s) 143.66
Iteration 1: (CP ns) 16.26 (Constraint ns) 5.85 (Elapsed s) 278.72
Iteration 2: (CP ns) 10.25 (Constraint ns) 12.73 (Elapsed s) 427.22
Iteration 3: (CP ns) 9.55 (Constraint ns) 9.73 (Elapsed s) 584.37
Iteration 4: (CP ns) 9.55 (Constraint ns) 9.38 (Elapsed s) 741.04
Final results are saved in
    /tmp/hdlsrc/sfir_fixed/hdlexpl/Final-07-Jan-2014-17-07-30
Validation model: gm_sfir_fixed_vnl
```

Then coder stops after one additional iteration because it has achieved the target clock frequency. The critical path is 9.55 ns, or a clock frequency of 104.71 MHz.

Regenerate code using original design and saved optimization data

Regenerate HDL code using the original model, `sfir_fixed`, and saved data from the final iteration of a previous optimization run.

Open the model and specify the DUT subsystem.

```
model = 'sfir_fixed';
dutSubsys = 'symmetric_fir';
open_system(model);
hdlset_param(model,'HDLSubsystem',[model,'/',dutSubsys]);
```

Set your synthesis tool and target device options to the same values as in the original run.

```
hdlset_param (model,'SynthesisTool','Xilinx ISE', ...
                    'SynthesisToolChipFamily','Zynq', ...
                    'SynthesisToolDeviceName','xc7z030', ...
                    'SynthesisToolPackageName','fbg484', ...
                    'SynthesisToolSpeedValue','-3')
```

Regenerate HDL code using saved optimization data from `cpGuidance.mat`.

```
hdlcoder.optimizeDesign(model,
  'hdlsrc/sfir_fixed/hdlexpl/Final-19-Dec-2013-23-05-04/cpGuidance.mat')

Final results are saved in
  /tmp/hdlsrc/sfir_fixed/hdlexpl/Final-07-Jan-2014-17-16-52
Validation model: gm_sfir_fixed_vnl
```

See Also

`hdlcoder.OptimizationConfig` |

Functions

`hdlcoder.supportedDevices`

Properties

`SynthesisToolSynthesisToolDeviceNameSynthesisToolChipFamilySynthesisToolP`

Concepts

- “Automatic Iterative Optimization”

Purpose

Show supported target hardware and device details

Syntax

```
hdlcoder.supportedDevices
```

Description

`hdlcoder.supportedDevices` shows a link to a report that contains device and device property names for target devices supported by your synthesis tool.

You can use the supported target device information to set `SynthesisToolChipFamily`, `SynthesisToolDeviceName`, `SynthesisToolPackageName`, and `SynthesisToolSpeedValue` for your model.

To see the report link, you must have a synthesis tool set up. If you have more than one synthesis tool available, you see a different report link for each synthesis tool.

Examples

Set the target device for your model

In this example, you set the target device for a model, `sfir_fixed`. Two synthesis tools are available, Altera® Quartus II and Xilinx® ISE. The target device is a Xilinx Virtex-6 XC6VLX130T FPGA.

Show the supported target device reports.

```
hdlcoder.supportedDevices
```

```
Altera QUARTUS II Device List
Xilinx ISE Device List
```

Click the `Xilinx ISE Device List` link to open the supported target device report and view details for your target device.

Open the model, `sfir_fixed`.

```
sfir_fixed
```

Set the `SynthesisToolChipFamily`, `SynthesisToolDeviceName`, `SynthesisToolPackageName`, and `SynthesisToolSpeedValue` model parameters based on details from the supported target device report.

```
hdlset_param ('sfir_fixed',
              'SynthesisToolChipFamily', 'Virtex6',
              'SynthesisToolDeviceName','xc6vlx130t',
              'SynthesisToolPackageName', 'ff484',
              'SynthesisToolSpeedValue', '-1')
```

View the nondefault parameters for your model, including target device information.

```
hdldispmdlparams
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
HDL CodeGen Parameters (non-default)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

SynthesisTool                      : 'Xilinx ISE'
SynthesisToolChipFamily            : 'Virtex6'
SynthesisToolDeviceName            : 'xc6vlx130t'
SynthesisToolPackageName           : 'ff484'
SynthesisToolSpeedValue            : -1
```

See Also

`SynthesisToolChipFamily` | `SynthesisToolDeviceName` | `SynthesisToolPackageName` | `SynthesisToolSpeedValue`

Concepts

- "Synthesis Tool Path Setup"

Purpose

Display HDL block parameters with nondefault values

Syntax

```
hdldispblkparams(path)
hdldispblkparams(path,'all')
```

Description

`hdldispblkparams(path)` displays, for the specified block, the names and values of HDL parameters that have nondefault values.

`hdldispblkparams(path,'all')` displays, for the specified block, the names and values of all HDL block parameters.

Input Arguments

path

Path to a block or subsystem in the current model.

Default: None

'all'

If you pass in the string `'all'`, `hdldispblkparams` displays the names and values of all HDL properties of the specified block.

Examples

The following example displays nondefault HDL block parameter settings for a Sum of Elements block).

```
hdldispblkparams('simplevectorsum/vsum/Sum of Elements')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
HDL Block Parameters ('simplevectorsum/vsum/Sum of Elements')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Implementation

 Architecture : Linear

Implementation Parameters

 InputPipeline : 1
```

The following example displays HDL block parameters and values for the currently selected block, (a Sum of Elements block).

```
hdldispblkparams(gcb,'all')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
HDL Block Parameters ('simplevectorsum/vsum/Sum of
Elements')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Implementation

 Architecture  : Linear

Implementation Parameters

 InputPipeline : 0
 OutputPipeline : 0
```

See Also

"Set and View HDL Block Parameters"

hdldispmdlparams

Purpose

Display HDL model parameters with nondefault values

Syntax

```
hdldispmdlparams(model)
hdldispmdlparams(model,'all')
```

Description

`hdldispmdlparams(model)` displays, for the specified model, the names and values of HDL parameters that have nondefault values.

`hdldispmdlparams(model,'all')` displays the names and values of all HDL parameters for the specified model.

Input Arguments

model

Name of an open model.

Default: None

'all'

If you pass in the string`'all'` , `hdldispmdlparams` displays the names and values of all HDL properties of the specified model.

Examples

The following example displays HDL properties of the current model that have nondefault values.

```
 hdldispmdlparams(bdroot)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
HDL CodeGen Parameters (non-default)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

CodeGenerationOutput           : 'GenerateHDLCodeAndDisplayGeneratedModel'
HDLSubsystem                   : 'simplevectorsum_2atomics/Subsystem'
OptimizationReport             : 'on'
ResetInputPort                 : 'rst'
ResetType                      : 'Synchronous'
```

The following example displays HDL properties and values of the current model.

```
 hdldispmdlparams(bdroot,'all')

%%%%%%%%%%%%%%%%%%%%%%%%%
HDL CodeGen Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%

AddPipelineRegisters                  : 'off'
Backannotation                        : 'on'
BlockGenerateLabel                    : '_gen'
CheckHDL                              : 'off'
ClockEnableInputPort                  : 'clk_enable'
.
.
.
VerilogFileExtension                  : '.v'
```

See Also

"View HDL Model Parameters"

hdlget_param

Purpose

Return value of specified HDL block-level parameter for specified block

Syntax

```
p = hdlget_param(block_path,prop)
```

Description

`p = hdlget_param(block_path,prop)` gets the value of a specified HDL property of a block or subsystem, and returns the value to the output variable.

Input Arguments

block_path

Path to a block or subsystem in the current model.

Default: None

prop

A string designating one of the following:

- The name of an HDL block property of the block or subsystem specified by `block_path`.
- `'all'` : If `prop` is set to `'all'`, `hdlget_param` returns `Name,Value` pairs for HDL properties of the specified block.

Default: None

Tips

- Use `hdlget_param` only to obtain the value of HDL block parameters (see "HDL Block Properties" for a list of block implementation parameters). Use `hdldispmdlparams` to see the values of HDL model parameters. To obtain the value of general model parameters, use the `get_param` function.

Output Arguments

p

`p` receives the value of the HDL block property specified by `prop`. The data type and dimensions of `p` depend on the data type and dimensions of the value returned. If `prop` is set to `'all'`, `p` is a cell array.

Examples

In the following example `hdlget_param` returns the value of the HDL block parameter `OutputPipeline` to the variable `p`.

```
  p = hdlget_param(gcb,'OutputPipeline')

p =

     3
```

In the following example `hdlget_param` returns HDL block parameters and values for the current block to the cell array `p`.

```
p = hdlget_param(gcb,'all')

p =

    'Architecture'    'Linear'    'InputPipeline'    [0]    'OutputPipeline'    [0]
```

See Also

`hdlset_param` | `hdlsaveparams` | `hdlrestoreparams`

Purpose

Create library of blocks that support HDL code generation

Syntax

```
hdllib
hdllib('html')
```

Description

`hdllib` creates a library of blocks that are compatible with HDL code generation. Use blocks from this library to build models that are compatible with the coder.

The default library name is `hdlsupported`. After you generate the library, you can save it to a folder of your choice.

Regenerate the library each time you install a new release to keep it current.

`hdllib('html')` creates a library of blocks that are compatible with HDL code generation, and generates two additional HTML reports: a categorized list of blocks (`hdlblklist.html`), and a table of blocks and their HDL code generation parameters (`hdlsupported.html`).

Examples

Create a supported blocks library

To create a library that contains blocks supported for HDL code generation:

```
hdllib
```

The `hdlsupported` block library opens.

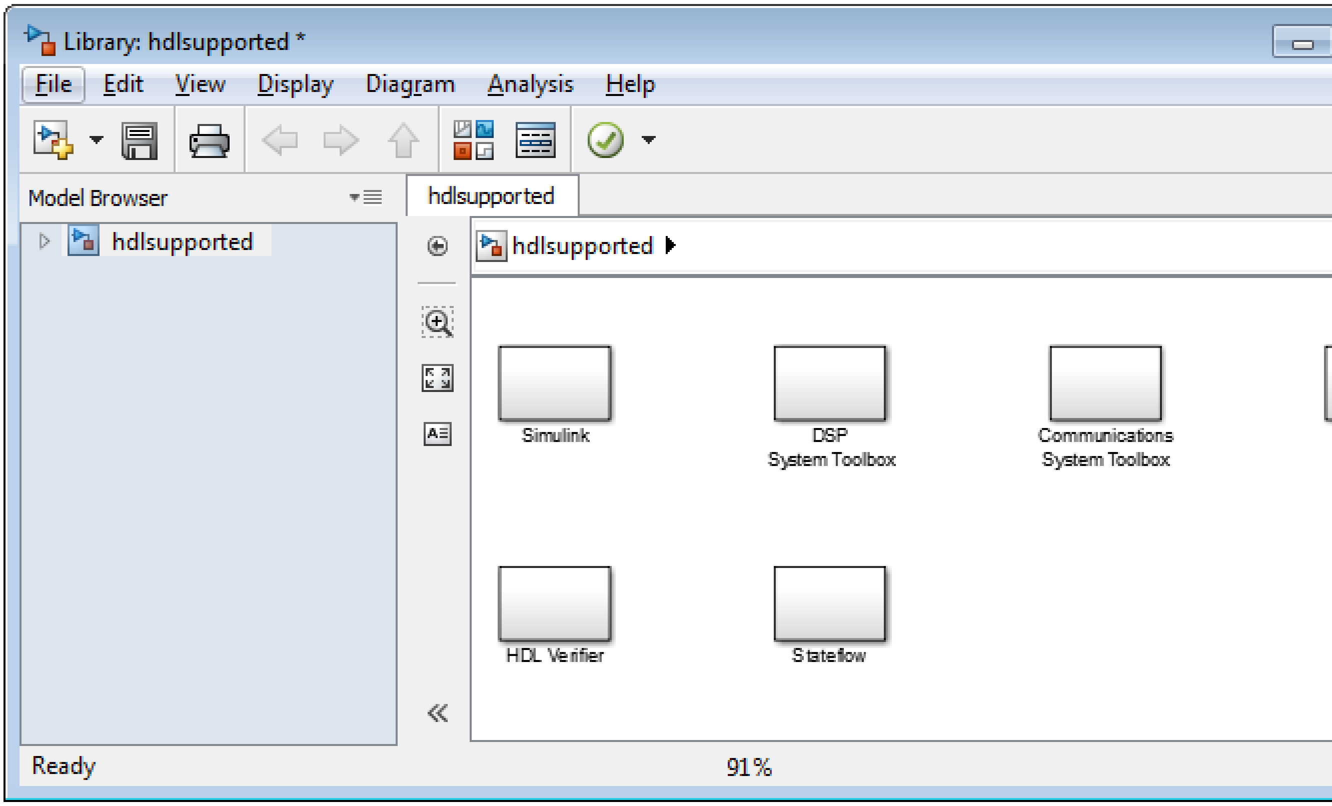
Create a supported blocks library and HTML reports

To create a library and HTML reports showing blocks supported for HDL code generation:

```
hdllib('html')
```

```
### HDL supported block list hdlblklist.html
### HDL implementation list hdlsupported.html
```

The `hdlsupported` library opens. To view the reports, click the `hdlblklist.html` and `hdlsupported.html` links.



See Also "HDL Block Properties"

Purpose

Generate customizable control file from selected subsystem or blocks

---

Note `hdlnewblackbox` is not recommended. Use `hdlset_param` and `hdlget_param` instead.

Syntax

```
hdlnewblackbox
hdlnewblackbox('blockpath')
hdlnewblackbox({'blockpath1','blockpath2',...'blockpathN'})
[cmd, impl] = hdlnewblackbox
[cmd, impl] = hdlnewblackbox('blockpath')
[cmd, impl] = hdlnewblackbox({'blockpath1','blockpath2',
   ...'blockpathN'})
[cmd, impl, params] = hdlnewblackbox
[cmd, impl, params] = hdlnewblackbox('blockpath')
[cmd, impl, params] = hdlnewblackbox({'blockpath1','blockpath2',
   ...'blockpathN'})
```

Description

The `hdlnewblackbox` utility helps you construct `forEach` calls for use in code generation control files when generating black box interfaces. Given a selection of one or more blocks from your model, `hdlnewblackbox` returns the following as string data in the MATLAB workspace for each selected block:

- A `forEach` call coded with the `modelscope`, `blocktype`, and default implementation class (`SubsystemBlackBoxHDLInstantiation`) arguments for the block.
- (Optional) a cell array of strings enumerating the available implementations classes for the subsystem.
- (Optional) A cell array of cell arrays of strings enumerating the names of implementation parameters corresponding to the implementation classes. `hdlnewblackbox` does not list data types and other details of implementation parameters.

`hdlnewblackbox` returns a `forEach` call for each selected block in the model.

`hdlnewblackbox('blockpath')` returns a `forEach` call for the block specified by the `'blockpath'` argument. The `'blockpath'` argument is a string specifying the full Simulink® path to the desired block.

`hdlnewblackbox({'blockpath1','blockpath2',...'blockpathN'})` returns a `forEach` call for the blocks specified by the `{'blockpath1','blockpath2',...'blockpathN'}` arguments. The `{'blockpath1','blockpath2',...'blockpathN'}` arguments are passed as a cell array of strings, each string specifying the full Simulink path to a desired block.

`[cmd, impl] = hdlnewblackbox` returns a `forEach` call for each selected block in the model to the string variable `cmd`. The call also returns `impl`, a cell array of cell arrays of strings enumerating the available implementations for the block.

`[cmd, impl] = hdlnewblackbox('blockpath')` returns a `forEach` call for the block specified by the `'blockpath'` argument to the string variable `cmd`. The call also returns `impl`, a cell array of cell arrays of strings enumerating the available implementations for the block. The `'blockpath'` argument is a string specifying the full Simulink path to the desired block.

`[cmd, impl] = hdlnewblackbox({'blockpath1','blockpath2', ...'blockpathN'})` returns a `forEach` call for the blocks specified by the `{'blockpath1','blockpath2',...'blockpathN'}` arguments to the string variable `cmd`. The call also returns `impl`, a cell array of cell arrays of strings enumerating the available implementations for the block. The `{'blockpath1','blockpath2',...'blockpathN'}` arguments are passed as a cell array of strings, each string specifying the full Simulink path to a desired block.

`[cmd, impl, params] = hdlnewblackbox` returns a `forEach` call for each selected block in the model to the string variable `cmd`. The call also returns:

- `impl`, a cell array of cell arrays of strings enumerating the available implementations for the block.

- `params`, a cell array of cell arrays of strings enumerating the available implementation parameters corresponding to each implementation.

`[cmd, impl, params] = hdlnewblackbox('blockpath')` returns a `forEach` call for the block specified by the `'blockpath'` argument to the string variable `cmd`. The call also returns:

- `impl`, a cell array of cell arrays of strings enumerating the available implementations for the block.
- `params`, a cell array of cell arrays of strings enumerating the available implementation parameters corresponding to each implementation.

The `'blockpath'` argument is a string specifying the full Simulink path to the desired block.

`[cmd, impl, params] = hdlnewblackbox({'blockpath1','blockpath2', ...'blockpathN'})` returns a `forEach` call for the blocks specified by the `{'blockpath1','blockpath2',...'blockpathN'}` arguments to the string variable `cmd`. The call also returns:

- `impl`, a cell array of cell arrays of strings enumerating the available implementations for the block.
- `params`, a cell array of cell arrays of strings enumerating the available implementation parameters corresponding to each implementation.

The `{'blockpath1','blockpath2',...'blockpathN'}` arguments are passed as a cell array of strings, each string specifying the full Simulink path to a desired block.

Tips

After invoking `hdlnewblackbox`, you will generally want to insert the `forEach` calls returned by the function into a control file, and use the implementation information returned to specify a nondefault block implementation.

Examples

```
% Return a forEach call for a specific subsystem to the MATLAB workspace
hdlnewblackbox('sfir_fixed/symmetric_fir');
%
% Return forEach calls for all currently selected blocks to the MATLAB workspace
```

```
hdlnewblackbox;
%
% Return forEach calls, implementation names, and implementation parameter names
% for all currently selected blocks to string variables
[cmd,impl,parms] = hdlnewblackbox;
```

Purpose

Construct code generation control object for use in control file

---

Note `hdlnewcontrol` is not recommended. Use `hdlset_param` and `hdlget_param` instead.

Syntax

```
object = hdlnewcontrol(mfilename)
```

Description

`object = hdlnewcontrol(mfilename)` constructs and returns a control generation control object (`object`) that is linked to a code generation control file.

The argument to `hdlnewcontrol` is the name of the control file itself. Use the `mfilename` function to pass in the file name string.

Tip The `hdlnewcontrol` function constructs an instance of the class `slhdlcoder.ConfigurationContainer`. `hdlnewcontrol` is a wrapper function provided to let you instantiate such objects. You should not directly call the constructor of the class.

In your control files, use only the public methods of the class `slhdlcoder.ConfigurationContainer`. Public methods are described in this document. Other methods of this class are for MathWorks® internal development use only.

See also

- "READ THIS FIRST: Control File Compatibility and Conversion Issues"

Purpose

Generate customizable control file from selected subsystem or blocks

Note `hdlnewcontrolfile` is not recommended. Use `hdlset_param` and `hdlget_param` instead.

Syntax

```
hdlnewcontrolfile
hdlnewcontrolfile('blockpath')
hdlnewcontrolfile({'blockpath1','blockpath2',...'blockpathN'})
t = hdlnewcontrolfile(...)
```

Description

The coder provides the `hdlnewcontrolfile` utility to help you construct code generation control files. Given a selection of one or more blocks from your model, `hdlnewcontrolfile` generates a control file containing:

- A `c.generateHDLFor` call specifying the full path to the currently selected block or subsystem from which code is to be generated.
- `c.forEach` calls for the selected blocks that have HDL implementations.
- Comments providing information about supported implementations and parameters for selected blocks that have HDL implementations.
- `c.set` calls for global HDL Coder™ options that are set to nondefault values.

Generated control files are automatically opened as untitled files in the MATLAB editor for further customization. The file naming sequence for successively generated control files is `Untitled1, Untitled2,...UntitledN`.

`hdlnewcontrolfile` returns a control file containing a `forEach` statement and comments for each selected block in the model.

`hdlnewcontrolfile('blockpath')` returns a control file containing a `forEach` statement and comments for the block specified by the `'blockpath'` argument. The `'blockpath'` argument is a string specifying the full Simulink path to the desired block.

`hdlnewcontrolfile({'blockpath1','blockpath2',...'blockpathN'})` returns a control file containing a `forEach` statement and comments for the blocks specified by the `{'blockpath1','blockpath2',...'blockpathN'}` arguments. The `{'blockpath1','blockpath2',...'blockpathN'}` arguments are passed as a cell array of strings, each string specifying the full Simulink path to a desired block.

`t = hdlnewcontrolfile(...)` returns control statements as text in the string variable `t`, instead of returning a control file.

Tips

You can use the generated control file as:

- A starting point for development of a customized control file.
- A source of information or documentation of the HDL code generation parameter settings in the model.

Examples

```
% Generate control file for a specific block
hdlnewcontrolfile('sfir_fixed/symmetric_fir/Product1');
%
% Generate a control file for all currently selected blocks
hdlnewcontrolfile;
%
% Generate a control file for two specific blocks
hdlnewcontrolfile({'sfir_fixed/symmetric_fir/Add1',...
'sfir_fixed/symmetric_fir/Product2'});
```

hdlnewforeach

Purpose

Generate `forEach` calls for insertion into code generation control files

Note `hdlnewforeach` is not recommended. Use `hdlset_param` and `hdlget_param` instead.

Syntax

```
hdlnewforeach
hdlnewforeach('blockpath')
hdlnewforeach({'blockpath1','blockpath2',...})
[cmd, impl] = hdlnewforeach
[cmd, impl] = hdlnewforeach('blockpath')
[cmd, impl] = hdlnewforeach({'blockpath1','blockpath2',...})
[cmd, impl, parms] = hdlnewforeach
[cmd, impl, parms] = hdlnewforeach('blockpath')
[cmd, impl, parms] = hdlnewforeach({'blockpath1','blockpath2',...})
```

Description

The coder provides the `hdlnewforeach` utility to help you construct `forEach` calls for use in code generation control files. Given a selection of one or more blocks from your model, `hdlnewforeach` returns the following for each selected block, as string data in the MATLAB workspace:

- A `forEach` call coded with the `modelscope`, `blocktype`, and default implementation arguments for the block.
- (Optional) A cell array of cell arrays of strings enumerating the available implementations for the block.
- (Optional) A cell array of cell arrays of strings enumerating the names of implementation parameters corresponding to the block implementations. See “HDL Block Properties” for that data types and other details of block implementation parameters.

`hdlnewforeach` returns a `forEach` call for each selected block in the model. Each call is returned as a string.

`hdlnewforeach('blockpath')` returns a `forEach` call for a specified block in the model. The call is returned as a string.

The `'blockpath'` argument is a string specifying the full path to the desired block.

`hdlnewforeach({'blockpath1','blockpath2',...})` returns a `forEach` call for each specified block in the model. Each call is returned as a string.

The `{'blockpath1','blockpath2',...}` argument is a cell array of strings, each of which specifies the full path to a desired block.

`[cmd, impl] = hdlnewforeach` returns a `forEach` call for each selected block in the model to the string variable `cmd`. In addition, the call returns a cell array of cell arrays of strings (`impl`) enumerating the available implementations for the block.

`[cmd, impl] = hdlnewforeach('blockpath')` returns a `forEach` call for a specified block in the model to the string variable `cmd`. In addition, the call returns a cell array of cell arrays of strings (`impl`) enumerating the available implementations for the block.

The `'blockpath'` argument is a string specifying the full path to the desired block.

`[cmd, impl] =`
`hdlnewforeach({'blockpath1','blockpath2',...})` returns a `forEach` call for each specified block in the model to the string variable `cmd`. In addition, the call returns a cell array of cell arrays of strings (`impl`) enumerating the available implementations for the block.

The `{'blockpath1','blockpath2',...}` argument is a cell array of strings, each of which specifies the full path to a desired block.

`[cmd, impl, parms] = hdlnewforeach` returns a `forEach` call for each selected block in the model to the string variable `cmd`. In addition, the call returns:

- A cell array of cell arrays of strings (`impl`) enumerating the available implementations for the block.
- A cell array of cell arrays of strings (`parms`) enumerating the available implementation parameters corresponding to each implementation.

`[cmd, impl, parms] = hdlnewforeach('blockpath')` returns a `forEach` call for a specified block in the model to the string variable `cmd`. In addition, the call returns:

- A cell array of cell arrays of strings (`impl`) enumerating the available implementations for the block.
- A cell array of cell arrays of strings (`parms`) enumerating the available implementation parameters corresponding to each implementation.

The `'blockpath'` argument is a string specifying the full path to the desired block.

`[cmd, impl, parms] = hdlnewforeach({'blockpath1','blockpath2',...})` returns a `forEach` call for each specified block in the model to the string variable `cmd`. In addition, the call returns:

- A cell array of cell arrays of strings (`impl`) enumerating the available implementations for the block.
- A cell array of cell arrays of strings (`parms`) enumerating the available implementation parameters corresponding to each implementation.

The `{'blockpath1','blockpath2',...}` argument is a cell array of strings, each of which specifies the full path to a desired block.

Tips

`hdlnewforeach` returns an empty string for blocks that do not have an HDL implementation. `hdlnewforeach` also returns an empty string for subsystems, which are a special case. Subsystems do not have a default implementation class, but special-purpose subsystems implementations are provided (see “External Component Interfaces”).

After invoking `hdlnewforeach`, you will generally want to insert the `forEach` calls returned by the function into a control file, and use the implementation and parameter information returned to specify a nondefault block implementation. See “Generating Selection/Action Statements with the hdlnewforeach Function” for a worked example.

Examples

The following example generates `forEach` commands for two explicitly specified blocks.

```
hdlnewforeach({'sfir_fixed/symmetric_fir/Add4',...
'sfir_fixed/symmetric_fir/Product2'})

ans =

c.forEach('./symmetric_fir/Add4',...
 'built-in/Sum', {},...
 'default', {}); % Default architecture is 'Linear'

c.forEach('./symmetric_fir/Product2',...
 'built-in/Product', {},...
 'default', {}); % Default architecture is 'Linear'
```

The following example generates a `forEach` command for an explicitly specified Sum block. The implementation and parameters information returned is listed after the `forEach` command.

```
c.forEach('./symmetric_fir/Add4',...
 'built-in/Sum', {},...
 'default', {}); % Default architecture is 'Linear'




impl =

    {3x1 cell}


parms =

    {1x2 cell}    {1x2 cell}    {1x2 cell} >> parms{1:4}

>> impl{1}

ans =
    'Linear'
    'Cascade'
    'Tree'ans =

>> parms{1:3}

ans =

    'InputPipeline'    'OutputPipeline'


ans =

    'InputPipeline'    'OutputPipeline'


ans =

    'InputPipeline'    'OutputPipeline'
```

hdlrestoreparams

Purpose

Restore block- and model-level HDL parameters to model

Syntax

```
hdlrestoreparams(subsys)
hdlrestoreparams(subsys,filename)
```

Description

`hdlrestoreparams(subsys)` restores to the specified model the default block- and model-level HDL settings.

`hdlrestoreparams(subsys,filename)` restores to the specified model the block- and model-level HDL settings from a previously saved file.

Input Arguments

subsys - Subsystem name

string

Subsystem name, specified as a string, with full hierarchical path.

Example: `'modelname/subsysTarget'`

Example: `'modelname/subsysA/subsysB/subsysTarget'`

filename - Name of file

string

Name of file containing previously saved HDL model parameters.

Example: 'mymodel_saved_params.m'

Examples

Reset and Restore HDL-Related Model Parameters

Open the model.

```
sfir_fixed
```

Verify that model parameters have default values.

```
hdlsaveparams('sfir_fixed/symmetric_fir')
```

```
hdlset_param('sfir_fixed', 'HDLSubsystem', 'sfir_fixed');
```

Set HDL-related model parameters for the `symmetric_fir` subsystem.

```
hdlset_param('sfir_fixed/symmetric_fir', 'SharingFactor', 3)
hdlset_param('sfir_fixed/symmetric_fir/Product',
             'InputPipeline', 5)
```

Verify that model parameters are set.

```
hdlsaveparams('sfir_fixed/symmetric_fir')
```

```
hdlset_param('sfir_fixed', 'HDLSubsystem',
             'sfir_fixed/symmetric_fir');
hdlset_param('sfir_fixed/symmetric_fir', 'SharingFactor', 3);
hdlset_param('sfir_fixed/symmetric_fir/Product',
             'InputPipeline', 5);
```

Save the model parameters to a MATLAB script, `sfir_saved_params.m`.

```
hdlsaveparams('sfir_fixed/symmetric_fir',
              'sfir_saved_params.m')
```

Reset HDL-related model parameters to default values.

```
hdlrestoreparams('sfir_fixed/symmetric_fir')
```

Verify that model parameters have default values.

```
hdlsaveparams('sfir_fixed/symmetric_fir')
```

```
hdlset_param('sfir_fixed', 'HDLSubsystem',
             'sfir_fixed');
```

Restore the saved model parameters from `sfir_saved_params.m`.

```
hdlrestoreparams('sfir_fixed/symmetric_fir',
                 'sfir_saved_params.m')
```

Verify that the saved model parameters are restored.

```
hdlsaveparams('sfir_fixed/symmetric_fir')
```

```
hdlset_param('sfir_fixed', 'HDLSubsystem',
             'sfir_fixed/symmetric_fir');
hdlset_param('sfir_fixed/symmetric_fir', 'SharingFactor', 3);
hdlset_param('sfir_fixed/symmetric_fir/Product',
             'InputPipeline', 5);
```

See Also

`hdlsaveparams`

hdlsaveparams

Purpose

Save nondefault block- and model-level HDL parameters

Syntax

```
hdlsaveparams(subsys)
hdlsaveparams(subsys,filename)
```

Description

`hdlsaveparams(subsys)` displays nondefault block- and model-level HDL parameters.

`hdlsaveparams(subsys,filename)` saves nondefault block- and model-level HDL parameters to a MATLAB script.

Input Arguments

subsys - Subsystem name

string

Subsystem name, specified as a string, with full hierarchical path.

Example: `'modelname/subsysTarget'`

Example: `'modelname/subsysA/subsysB/subsysTarget'`

filename - Name of file

string

Name of file to which you are saving model parameters, specified as a string.

Example: 'mymodel_saved_params.m'

Examples

Display HDL-Related Nondefault Model Parameters

Open the model.

```
sfir_fixed
```

Set HDL-related model parameters for the `symmetric_fir` subsystem.

```
hdlset_param('sfir_fixed/symmetric_fir', 'SharingFactor', 3)
hdlset_param('sfir_fixed/symmetric_fir/Product', 'InputPipeline', 5)
```

Display HDL-related nondefault model parameters for the `symmetric_fir` subsystem.

```
hdlsaveparams('sfir_fixed/symmetric_fir')
```

```
hdlset_param('sfir_fixed', 'HDLSubsystem', 'sfir_fixed/symmetric_fir'
hdlset_param('sfir_fixed/symmetric_fir', 'SharingFactor', 3);
hdlset_param('sfir_fixed/symmetric_fir/Product', 'InputPipeline', 5);
```

The output identifies the subsystem and displays its HDL-related parameter values.

Save and Restore HDL-Related Model Parameters

Open the model.

```
sfir_fixed
```

Verify that model parameters have default values.

```
hdlsaveparams('sfir_fixed/symmetric_fir')
```

```
hdlset_param('sfir_fixed', 'HDLSubsystem', 'sfir_fixed');
```

Set HDL-related model parameters for the `symmetric_fir` subsystem.

```
hdlset_param('sfir_fixed/symmetric_fir', 'SharingFactor', 3)
hdlset_param('sfir_fixed/symmetric_fir/Product',
             'InputPipeline', 5)
```

Verify that model parameters are set.

```
hdlsaveparams('sfir_fixed/symmetric_fir')
```

```
hdlset_param('sfir_fixed', 'HDLSubsystem',
             'sfir_fixed/symmetric_fir');
hdlset_param('sfir_fixed/symmetric_fir', 'SharingFactor', 3);
hdlset_param('sfir_fixed/symmetric_fir/Product',
             'InputPipeline', 5);
```

Save the model parameters to a MATLAB script, `sfir_saved_params.m`.

```
hdlsaveparams('sfir_fixed/symmetric_fir',
              'sfir_saved_params.m')
```

Reset HDL-related model parameters to default values.

```
hdlrestoreparams('sfir_fixed/symmetric_fir')
```

Verify that model parameters have default values.

```
hdlsaveparams('sfir_fixed/symmetric_fir')

hdlset_param('sfir_fixed', 'HDLSubsystem',
             'sfir_fixed');
```

Restore the saved model parameters from `sfir_saved_params.m`.

```
hdlrestoreparams('sfir_fixed/symmetric_fir',
                 'sfir_saved_params.m')
```

Verify that the saved model parameters are restored.

```
hdlsaveparams('sfir_fixed/symmetric_fir')

hdlset_param('sfir_fixed', 'HDLSubsystem',
             'sfir_fixed/symmetric_fir');
hdlset_param('sfir_fixed/symmetric_fir', 'SharingFactor', 3);
hdlset_param('sfir_fixed/symmetric_fir/Product',
             'InputPipeline', 5);
```

See Also

`hdlrestoreparams`

Purpose

Set HDL-related parameters at model or block level

Syntax

```
hdlset_param(path,Name,Value)
```

Description

`hdlset_param(path,Name,Value)` sets HDL-related parameters in the block or model referenced by `path`. The parameters to be set, and their values, are specified by one or more `Name,Value` pair arguments. You can specify several name and value pair arguments in any order as `Name1,Value1, ,NameN,ValueN`.

Tips

- When you set multiple parameters on the same model or block, use a single `hdlset_param` command with multiple pairs of arguments, rather than multiple `hdlset_param` commands. This technique is more efficient because using a single call requires evaluating parameters only once.
- To set HDL block parameters for multiple blocks, use the `find_system` function to locate the blocks of interest. Then, use a loop to iterate over the blocks and call `hdlset_param` to set the desired parameters.

Input Arguments

path

Path to the model or block for which `hdlset_param` is to set one or more parameter values.

Default: None

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'Name'

`Name` is a string specifying the name of one of the following:

- A model-level HDL-related property. See Properties — Alphabetical List for a list of model-level properties, their data types and their default values.
- An HDL block property, such as an implementation name or an implementation parameter. See “HDL Block Properties” for a list of block implementation parameters.

Default: None

'Value'

`Value` is a value to be applied to the corresponding property in a `Name,Value` argument.

Default: Default value is dependent on the property.

Examples

The following example uses the `sfir_fixed` model to demonstrate how to locate a group of blocks in a subsystem and specify the same output pipeline depth for each of the blocks.

```
open sfir_fixed;
prodblocks = find_system('sfir_fixed/symmetric_fir', 'BlockType', 'Product');
for ii=1:length(prodblocks), hdlset_param(prodblocks{ii}, 'OutputPipeline', 2), end;
```

See Also

`hdlget_param` | `hdlsaveparams` | `hdlrestoreparams`

How To

- “Set and View HDL Block Parameters”
- “Set HDL Block Parameters for Multiple Blocks”

Purpose

Set up model parameters for HDL code generation

Syntax

`hdlsetup('`*`modelname`*`')`

Description

`hdlsetup('`*`modelname`*`')` sets the parameters of the model specified by *`modelname`* to common default values for HDL code generation. After using `hdlsetup`, you can use `set_param` to modify these default settings.

Open the model before you invoke the `hdlsetup` command.

To see which model parameters are affected by `hdlsetup`, open `hdlsetup.m`.

How hdlsetup Configures Solver Options

`hdlsetup` configures **Solver** options used by the coder. These options are:

- **Type**: `Fixed-step`. This is the recommended solver type for most HDL applications.

  The coder also supports variable-step solvers under the following conditions:

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

- **Solver**: `Discrete (no continuous states)`. You can use other fixed-step solvers, but this option is usually best for simulating discrete systems.

- **Tasking mode**: `SingleTasking`. The coder does not support multitasking mode.

  Do not set **Tasking mode** to `Auto`.

makehdl

Purpose

Generate HDL RTL code from model or subsystem

Syntax

```
makehdl(model)
makehdl(model,Name,Value)
```

Description

`makehdl(model)` generates HDL code from the specified model.

`makehdl(model,Name,Value)` generates HDL code from the specified model with options specified by one or more name-value pair arguments.

Input Arguments

model - Model or subsystem name

string

Model or subsystem name, specified as top-level model name or subsystem name with full hierarchical path.

Example: `'top_level_name'`

Example: `'top_level_name/subsysA/subsysB/codegen_subsys_name'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `` `TargetLanguage','Verilog' ``

Basic Options

'TargetLanguage' - Target language

`'VHDL'` (default) | `'Verilog'`

For more information, see `TargetLanguage`.

'TargetDirectory' - Output directory

'hdlsrc' (default) | string

For more information, see `TargetDirectory`.

'CheckHDL' - Check HDL code generation compatibility
'off' (default) | 'on'

For more information, see `CheckHDL`.

'GenerateHDLCode' - Generate HDL code
'on' (default) | 'off'

For more information, see `GenerateHDLCode`.

'SplitEntityArch' - Split VHDL entity and architecture into separate files
'off' (default) | 'on'

For more information, see `SplitEntityArch`.

'Verbosity' - Level of message detail
1 (default) | 0

For more information, see `Verbosity`.

Report Generation

'Traceability' - Generate report with mapping links between HDL and model
'off' (default) | 'on'

For more information, see `Traceability`.

'ResourceReport' - Resource utilization report generation
'off' (default) | 'on'

For more information, see `ResourceReport`.

'OptimizationReport' - Optimization report generation
'off' (default) | 'on'

For more information, see `OptimizationReport`.

'GenerateWebview' - Include model Web view

`'on'` (default) | `'off'`

For more information, see `GenerateWebview`.

Speed and Area Optimization

'BalanceDelays' - Delay balancing

`'on'` (default) | `'off'`

For more information, see `BalanceDelays`.

'DistributedPipeliningPriority' - Specify priority for distributed pipelining algorithm

`'NumericalIntegrity'` (default) | `'Performance'`

For more information, see `DistributedPipeliningPriority`.

'HierarchicalDistPipelining' - Hierarchical distributed pipelining

`'off'` (default) | `'on'`

For more information, see `HierarchicalDistPipelining`.

'PreserveDesignDelays' - Prevent distributed pipelining from moving design delays

`'off'` (default) | `'on'`

For more information, see `PreserveDesignDelays`.

'MaxOversampling' - Limit the maximum sample rate

0 (default) | N, where N is an integer greater than 1

For more information, see `MaxOversampling`.

'MaxComputationLatency' - Specify the maximum number of time steps for which your DUT inputs are guaranteed to be stable

1 (default) | N, where N is an integer greater than 1

For more information, see `MaxComputationLatency`.

'MinimizeClockEnables' - Omit clock enable logic for single-rate designs

`'off'` (default) | `'on'`

For more information, see `MinimizeClockEnables`.

'RAMMappingThreshold' - Minimum RAM size for mapping to RAMs instead of registers

256 (default) | positive integer

The minimum RAM size required for mapping to RAMs instead of registers, specified in bits.

For more information, see `RAMMappingThreshold`.

Coding Style

'UserComment' - HDL file header comment

string

For more information, see `UserComment`.

'UseAggregatesForConst' - Represent constant values with aggregates

`'off'` (default) | `'on'`

For more information, see `UseAggregatesForConst`.

'UseRisingEdge' - Use VHDL `rising_edge` or `falling_edge` function to detect clock transitions

`'off'` (default) | `'on'`

For more information, see `UseRisingEdge`.

'LoopUnrolling' - Unroll VHDL `FOR` and `GENERATE` loops

`'off'` (default) | `'on'`

For more information, see `LoopUnrolling`.

'UseVerilogTimescale' - Generate `'timescale` compiler directives

`'on'` (default) | `'off'`

For more information, see `UseVerilogTimescale`.

'InlineConfigurations' - Include VHDL configurations

`'on'` (default) | `'off'`

For more information, see `InlineConfigurations`.

'SafeZeroConcat' - Type-safe syntax for concatenated zeros

`'on'` (default) | `'off'`

For more information, see `SafeZeroConcat`.

'DateComment' - Include time stamp in header

`'on'` (default) | `'off'`

For more information, see `DateComment`.

'ScalarizePorts' - Flatten vector ports into scalar ports

`'off'` (default) | `'on'`

For more information, see `ScalarizePorts`.

'MinimizeIntermediateSignals' - Minimize intermediate signals

`'off'` (default) | `'on'`

For more information, see `MinimizeIntermediateSignals`.

'RequirementComments' - Link from code generation reports to requirement documents

`'on'` (default) | `'off'`

For more information, see `RequirementComments`.

'InlineMATLABBlockCode' - Inline HDL code for MATLAB Function blocks

`'off'` (default) | `'on'`

For more information, see `InlineMATLABBlockCode`.

'MaskParameterAsGeneric' - Reusable code generation for subsystems with identical mask parameters
`'off'` (default) | `'on'`

For more information, see `MaskParameterAsGeneric`.

'InitializeBlockRAM' - Initial signal value generation for RAM blocks
`'on'` (default) | `'off'`

For more information, see `InitializeBlockRAM`.

'RAMArchitecture' - RAM architecture
`'WithClockEnable'` (default) | `'WithoutClockEnable'`

For more information, see `RAMArchitecture`.

'HandleAtomicSubsystem' - Reusable code generation for identical atomic subsystems
`'on'` (default) | `'off'`

For more information, see `HandleAtomicSubsystem`.

Clocks and Reset

'ClockEdge' - Active clock edge
`'Rising'` (default) | `'Falling'`

For more information, see `ClockEdge`.

'ClockInputs' - Single or multiple clock inputs
`'Single'` (default) | `'Multiple'`

Single or multiple clock inputs, specified as a string.

For more information, see `ClockInputs`.

'Oversampling' - Oversampling factor for global clock
1 (default) | integer greater than or equal to 0

Frequency of global oversampling clock, specified as an integer multiple of the model's base rate.

For more information, see `Oversampling`.

'ResetAssertedLevel' - Asserted (active) level of reset

`'active-high'` (default) | `'active-low'`

For more information, see `ResetAssertedLevel`.

'ResetType' - Reset type

`'async'` (default) | `'sync'`

For more information, see `ResetType`.

'TriggerAsClock' - Use trigger signal as clock in triggered subsystems

`'off'` (default) | `'on'`

For more information, see `TriggerAsClock`.

'TimingControllerArch' - Generate reset for timing controller

`'default'` (default) | `'resettable'`

For more information, see `TimingControllerArch`.

Test Bench

'Verbosity' - Level of message detail

0 (default) | `n`

For more information, see `Verbosity`.

'GenerateCoSimBlock' - Generate HDL Cosimulation block

`'off'` (default) | `'on'`

Generate an HDL Cosimulation block so you can simulate the DUT in Simulink with an HDL simulator.

For more information, see `GenerateCoSimBlock`.

'GenerateCoSimModel' - Generate HDL Cosimulation model
`'ModelSim'` (default) | `'Incisive'`

Generate a model containing an HDL Cosimulation block for the specified HDL simulator.

For more information, see `GenerateCoSimModel`.

'GenerateValidationModel' - Generate validation model
`'off'` (default) | `'on'`

For more information, see `GenerateValidationModel`.

'SimulatorFlags' - Options for generated compilation scripts
string

For more information, see `SimulatorFlags`.

'TestBenchReferencePostFix' - Suffix for test bench reference signals
`'_ref'` (default) | string

For more information, see `TestBenchReferencePostFix`.

Script Generation

'EDAScriptGeneration' - Enable or disable script generation for third-party tools
`'on'` (default) | `'off'`

For more information, see `EDAScriptGeneration`.

'HDLCompileInit' - Compilation script initialization string
`'vlib work\n'` (default) | string

For more information, see `HDLCompileInit`.

'HDLCompileTerm' - Compilation script termination string
`''` (default) | string

For more information, see `HDLCompileTerm`.

'HDLCompileFilePostfix' - Postfix for compilation script file name
`'_compile.do'` (default) | string

For more information, see `HDLCompileFilePostfix`.

'HDLCompileVerilogCmd' - Verilog compilation command
`'vlog %s %s\n'` (default) | string

Verilog compilation command, specified as a string. The `SimulatorFlags` name-value pair specifies the first argument, and the module name specifies the second argument.

For more information, see `HDLCompileVerilogCmd`.

'HDLCompileVHDLCmd' - VHDL compilation command
`'vcom %s %s\n'` (default) | string

VHDL compilation command, specified as a string. The `SimulatorFlags` name-value pair specifies the first argument, and the entity name specifies the second argument.

For more information, see `HDLCompileVerilogCmd`.

'HDLLintTool' - HDL lint tool
`'None'` (default) | `'AscentLint'` | `'Leda'` | `'SpyGlass'` | `'Custom'`

HDL lint tool, specified as a string.

For more information, see `HDLLintTool`.

'HDLLintInit' - HDL lint initialization string
string

HDL lint initialization, specified as a string. The default is derived from the `HDLLintTool` name-value pair.

For more information, see `HDLLintInit`.

'HDLLintCmd' - HDL lint command
string

HDL lint command, specified as a string. The default is derived from the `HDLLintTool` name-value pair.

For more information, see `HDLLintCmd`.

'HDLLintTerm' - HDL lint termination string

string

HDL lint termination, specified as a string. The default is derived from the `HDLLintTool` name-value pair.

For more information, see `HDLLintTerm`.

'HDLSynthTool' - Synthesis tool

`'None'` (default) | `'ISE'` | `'Precision'` | `'Quartus'` | `'Synplify'` | `'Custom'`

HDL synthesis tool, specified as a string.

For more information, see `HDLSynthTool`.

'HDLSynthCmd' - HDL synthesis command

string

HDL synthesis command, specified as a string. The default is derived from the `HDLSynthTool` name-value pair.

For more information, see `HDLSynthCmd`.

'HDLSynthFilePostfix' - Postfix for synthesis script file name

string

HDL synthesis script file name postfix, specified as a string. The default is derived from the `HDLSynthTool` name-value pair.

For more information, see `HDLSynthFilePostfix`.

'HDLSynthInit' - Synthesis script initialization string

string

Initialization for the HDL synthesis script, specified as a string. The default is derived from the `HDLSynthTool` name-value pair.

For more information, see `HDLSynthInit`.

'HDLSynthTerm' - Synthesis script termination string
string

Termination string for the HDL synthesis script. The default is derived from the `HDLSynthTool` name-value pair.

For more information, see `HDLSynthTerm`.

Generated Model

'CodeGenerationOutput' - Display and generation of generated model
`'GenerateHDLCode'` (default) | `'GenerateHDLCodeAndDisplayGeneratedModel'` | `'DisplayGeneratedModelOnly'`

For more information, see `CodeGenerationOutput`.

'GeneratedModelName' - Generated model name
same as original model name (default) | string

For more information, see `GeneratedModelName`.

'GeneratedModelNamePrefix' - Prefix for generated model name
`'gm_'` (default) | string

For more information, see `GeneratedModelNamePrefix`.

'HighlightAncestors' - Highlight parent blocks of generated model blocks differing from original model
`'on'` (default) | `'off'`

For more information, see `HighlightAncestors`.

'HighlightColor' - Color of highlighted blocks in generated model
`'cyan'` (default) | `'yellow'` | `'magenta'` | `'red'` | `'green'` | `'blue'` | `'white'` | `'magenta'` | `'black'`

For more information, see `HighlightColor`.

Synthesis

'SynthesisTool' - Synthesis tool

`''` (default) | `'Altera Quartus II'` | `'Xilinx ISE'`

For more information, see `SynthesisTool`.

'MulticyclePathInfo' - Multicycle path constraint file generation

`'off'` (default) | `'on'`

For more information, see `MulticyclePathInfo`.

Port Names and Types

'ClockEnableInputPort' - Clock enable input port name

`'clk_enable'` (default) | string

Clock enable input port name, specified as a string.

For more information, see `ClockEnableInputPort`.

'ClockEnableOutputPort' - Clock enable output port name

`'ce_out'` (default) | string

Clock enable output port name, specified as a string.

For more information, see `ClockEnableOutputPort`.

'ClockInputPort' - Clock input port name

`'clk'` (default) | string

Clock input port name, specified as a string.

For more information, see `ClockInputPort`.

'InputType' - HDL data type for input ports

`'signed/unsigned'` | `'wire'` or `'std_logic_vector'` (default)

HDL data type for input ports, specified as a string.

VHDL inputs can have `'std_logic_vector'` or `'signed/unsigned'` data type. Verilog inputs must be `'wire'`.

For more information, see `InputType`.

'OutputType' - HDL data type for output ports

`'Same as input data type'` (default) | `'std_logic_vector'` | `'signed/unsigned'` | `'wire'`

HDL data type for output ports, specified as a string.

VHDL output can be `'Same as input data type'`, `'std_logic_vector'` or `'signed/unsigned'`. Verilog output must be `'wire'`.

For more information, see `OutputType`.

'ResetInputPort' - Reset input port name

`'reset'` (default) | string

Reset input port name, specified as a string.

For more information, see `ResetInputPort`.

File and Variable Names

'VerilogFileExtension' - Verilog file extension

`'.v'` (default) | string

For more information, see `VerilogFileExtension`.

'VHDLFileExtension' - VHDL file extension

`'.vhd'` (default) | string

For more information, see `VHDLFileExtension`.

'VHDLArchitectureName' - VHDL architecture name

`'rtl'` (default) | string

For more information, see `VHDLArchitectureName`.

'VHDLLibraryName' - VHDL library name
`'work'` (default) | string

For more information, see `VHDLLibraryName`.

'SplitEntityFilePostfix' - Postfix for VHDL entity file names
`'_entity'` (default) | string

For more information, see `SplitEntityFilePostfix`.

'SplitArchFilePostfix' - Postfix for VHDL architecture file names
`'_arch'` (default) | string

For more information, see `SplitArchFilePostfix`.

'PackagePostfix' - Postfix for package file name
`'_pkg'` (default) | string

For more information, see `PackagePostfix`.

'HDLMapFilePostfix' - Postfix for mapping file
`'_map.txt'` (default) | string

For more information, see `HDLMapFilePostfix`.

'BlockGenerateLabel' - Block label postfix for VHDL `GENERATE` statements
`'_gen'` (default) | string

For more information, see `BlockGenerateLabel`.

'ClockProcessPostfix' - Postfix for clock process names
`'_process'` (default) | string

For more information, see `ClockProcessPostfix`.

'ComplexImagPostfix' - Postfix for imaginary part of complex signal
`'_im'` (default) | string

For more information, see `ComplexImagPostfix`.

'ComplexRealPostfix' - Postfix for imaginary part of complex signal names
`'_re'` (default) | string

For more information, see `ComplexRealPostfix`.

'EntityConflictPostfix' - Postfix for duplicate VHDL entity or Verilog module names
`'_block'` (default) | string

For more information, see `EntityConflictPostfix`.

'InstanceGenerateLabel' - Instance section label postfix for VHDL `GENERATE` statements
`'_gen'` (default) | string

For more information, see `InstanceGenerateLabel`.

'InstancePostfix' - Postfix for generated component instance names
`''` (default) | string

For more information, see `InstancePostfix`.

'InstancePrefix' - Prefix for generated component instance names
`'u_'` (default) | string

For more information, see `InstancePrefix`.

'OutputGenerateLabel' - Output assignment label postfix for VHDL `GENERATE` statements
`'outputgen'` (default) | string

For more information, see `OutputGenerateLabel`.

'PipelinePostfix' - Postfix for input and output pipeline register names
`'_pipe'` (default) | string

For more information, see `PipelinePostfix`.

'ReservedWordPostfix' - Postfix for names conflicting with VHDL or Verilog reserved words
`'_rsvd'` (default) | string

For more information, see `ReservedWordPostfix`.

'TimingControllerPostfix' - Postfix for timing controller name
`'_tc'` (default) | string

For more information, see `TimingControllerPostfix`.

'VectorPrefix' - Prefix for vector names
`'vector_of_'` (default) | string

For more information, see `VectorPrefix`.

'EnablePrefix' - Prefix for internal enable signals
`'enb'` (default) | string

Prefix for internal clock enable and control flow enable signals, specified as a string.

For more information, see `EnablePrefix`.

'ModulePrefix' - Specify a prefix for every module or entity name in the generated HDL code. The coder also applies this prefix to generated script file names
`''` (default) | string

For more information, see `ModulePrefix`.

Examples

Generate VHDL for the Current Model

Generate VHDL code for the current model.

Generate HDL code for the current model with code generation options set to default values.

```
makehdl(bdroot)
```

The generated VHDL code is saved in the `hdlsrc` folder.

Generate Verilog for a Subsystem Within a Model

Generate Verilog for the subsystem `symmetric_fir` within the model `sfir_fixed`.

Open the `sfir_fixed` model.

```
sfir_fixed;
```

The model opens in a new Simulink window.

Generate Verilog for the `symmetric_fir` subsystem.

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

```
### Generating HDL for 'sfir_fixed/symmetric_fir'.
### Starting HDL check.
### HDL check for 'sfir_fixed' complete with 0 errors, 0 warnings,
    and 0 messages.
### Begin Verilog Code Generation for 'sfir_fixed'.
### Working on sfir_fixed/symmetric_fir as
    hdlsrc\sfir_fixed\symmetric_fir.v
### HDL code generation complete.
```

The generated Verilog code for the `symmetric_fir` subsystem is saved in `hdlsrc\sfir_fixed\symmetric_fir.v`.

Close the model.

```
bdclose('sfir_fixed');
```

Check Subsystem for Compatibility with HDL Code Generation

Check that the subsystem `symmetric_fir` is compatible with HDL code generation, then generate HDL.

Open the `sfir_fixed` model.

```
sfir_fixed;
```

The model opens in a new Simulink window.

Check the `symmetric_fir` subsystem for compatibility with HDL code generation. Generate code with code generation options set to default values.

```
makehdl('sfir_fixed/symmetric_fir','CheckHDL','on')
```

The generated VHDL code for the `symmetric_fir` subsystem is saved in `hdlsrc\sfir_fixed\symmetric_fir.vhd`.

Close the model.

```
bdclose('sfir_fixed');
```

See Also

`makehdltb` | `checkhdl`

makehdltb

Purpose

Generate HDL test bench from model or subsystem

Syntax

```
makehdltb(subsys)
makehdltb(subsys,Name,Value)
```

Description

`makehdltb(subsys)` generates an HDL test bench from the specified subsystem.

`makehdltb(subsys,Name,Value)` generates an HDL test bench from the specified subsystem with options specified by one or more name-value pair arguments.

Input Arguments

subsys - Subsystem name
string

Subsystem name, specified as a string, with full hierarchical path.

Example: `'modelname/subsysTarget'`

Example: `'modelname/subsysA/subsysB/subsysTarget'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `` `TargetLanguage','Verilog' ``

Basic Options

'TargetLanguage' - Target language
`'VHDL'` (default) | `'Verilog'`

For more information, see `TargetLanguage`.

'TargetDirectory' - Output directory

`'hdlsrc'` (default) | string

For more information, see `TargetDirectory`.

'SplitEntityArch' - Split VHDL entity and architecture into separate files

`'off'` (default) | `'on'`

For more information, see `SplitEntityArch`.

Test Bench

'ForceClock' - Force clock input

`'on'` (default) | `'off'`

Specify that the generated test bench drives the clock enable input based on `ClockLowTime` and `ClockHighTime`.

For more information, see `ForceClock`.

'ClockHighTime' - Clock high time

5 (default) | positive integer

Clock high time during a clock period, specified in nanoseconds.

For more information, see `ClockHighTime`.

'ClockLowTime' - Clock low time

5 (default) | positive integer

Clock low time during a clock period, specified in nanoseconds.

For more information, see `ClockLowTime`.

'ForceClockEnable' - Force clock enable input

`'on'` (default) | `'off'`

Specify that the generated test bench drives the clock enable input.

For more information, see `ForceClockEnable`.

'ClockInputs' - Single or multiple clock inputs

`'Single'` (default) | `'Multiple'`

Single or multiple clock inputs, specified as a string.

For more information, see `ClockInputs`.

'ForceReset' - Force reset input
`'on'` (default) | `'off'`

Specify that the generated test bench drives the reset input.

For more information, see `ForceReset`.

'ResetLength' - Reset asserted time length
2 (default) | integer greater than or equal to 0

Length of time that reset is asserted, specified as the number of clock cycles.

For more information, see `ResetLength`.

'ResetAssertedLevel' - Asserted (active) level of reset
`'active-high'` (default) | `'active-low'`

For more information, see `ResetAssertedLevel`.

'HoldInputDataBetweenSamples' - Hold valid data for signals clocked at slower rate
`'on'` (default) | `'off'`

For more information, see `HoldInputDataBetweenSamples`.

'HoldTime' - Hold time for inputs and forced reset
2 (default) | positive integer

Hold time for inputs and forced reset, specified in nanoseconds.

For more information, see `HoldTime`.

'IgnoreDataChecking' - Time to wait after clock enable before checking output data
0 (default) | positive integer

Time after clock enable is asserted before starting output data checks, specified in number of samples.

For more information, see `IgnoreDataChecking`.

'InitializeTestBenchInputs' - Initialize test bench inputs to `0`
`'off'` (default) | `'on'`

For more information, see `InitializeTestBenchInputs`.

'MultifileTestBench' - Divide generated test bench into helper functions, data, and HDL test bench files
`'off'` (default) | `'on'`

For more information, see `MultifileTestBench`.

'UseFileIOInTestBench' - Use file I/O to read/write test bench data
`'off'` (default) | `'on'`

For more information, see `UseFileIOInTestBench`.

'TestBenchClockEnableDelay' - Number of clock cycles between deassertion of reset and assertion of clock enable
1 (default) | positive integer

For more information, see `TestBenchClockEnableDelay`.

'TestBenchDataPostFix' - Postfix for test bench data file name
`'_data'` (default) | string

For more information, see `TestBenchDataPostFix`.

'TestBenchPostFix' - Suffix for test bench name
`'_tb'` (default) | string

For more information, see `TestBenchPostFix`.

'GenerateCoSimBlock' - Generate HDL Cosimulation block
`'off'` (default) | `'on'`

Generate an HDL Cosimulation block so you can simulate the DUT in Simulink with an HDL simulator.

For more information, see `GenerateCoSimBlock`.

'GenerateCoSimModel' - Generate HDL Cosimulation model
`'ModelSim'` (default) | `'Incisive'`

Generate a model containing an HDL Cosimulation block for the specified HDL simulator.

For more information, see `GenerateCoSimModel`.

Coding Style

'UseVerilogTimescale' - Generate `'timescale` compiler directives
`'on'` (default) | `'off'`

For more information, see `UseVerilogTimescale`.

'DateComment' - Include time stamp in header
`'on'` (default) | `'off'`

For more information, see `DateComment`.

'InlineConfigurations' - Include VHDL configurations
`'on'` (default) | `'off'`

For more information, see `InlineConfigurations`.

'ScalarizePorts' - Flatten vector ports into scalar ports
`'off'` (default) | `'on'`

For more information, see `ScalarizePorts`.

Script Generation

'HDLCompileInit' - Compilation script initialization string
`'vlib work\n'` (default) | string

For more information, see `HDLCompileInit`.

'HDLCompileTerm' - Compilation script termination string
`''` (default) | string

For more information, see `HDLCompileTerm`.

'HDLCompileFilePostfix' - Postfix for compilation script file name
`'_compile.do'` (default) | string

For more information, see `HDLCompileFilePostfix`.

'HDLCompileVerilogCmd' - Verilog compilation command
`'vlog %s %s\n'` (default) | string

Verilog compilation command, specified as a string. The `SimulatorFlags` name-value pair specifies the first argument, and the module name specifies the second argument.

For more information, see `HDLCompileVerilogCmd`.

'HDLCompileVHDLCmd' - VHDL compilation command
`'vcom %s %s\n'` (default) | string

VHDL compilation command, specified as a string. The `SimulatorFlags` name-value pair specifies the first argument, and the entity name specifies the second argument.

For more information, see `HDLCompileVerilogCmd`.

'HDLSimCmd' - HDL simulation command
`'vsim -novopt work.%s\n'` (default) | string

The HDL simulation command, specified as a string. Your top-level module or entity name is automatically used as the argument.

For more information, see `HDLSimCmd`.

'HDLSimInit' - HDL simulation script initialization string
`['onbreak resume\n', 'onerror resume\n']` (default) | string

Initialization for the HDL simulation script, specified as a string.

For more information, see `HDLSimInit`.

'HDLSimTerm' - HDL simulation script termination string

`'run -all'` (default) | string

The termination string for the HDL simulation command.

For more information, see `HDLSimTerm`.

'HDLSimFilePostfix' - Postscript for HDL simulation script

`'_sim.do'` (default) | string

For more information, see `HDLSimFilePostfix`.

'HDLSimViewWaveCmd' - HDL simulation waveform viewing command

`'add wave sim:%s\n'` (default) | string

Waveform viewing command, specified as a string. The implicit argument adds the signal paths for the DUT top-level input, output, and output reference signals.

For more information, see `HDLSimViewWaveCmd`.

Port Names and Types

'ClockEnableInputPort' - Clock enable input port name

`'clk_enable'` (default) | string

Clock enable input port name, specified as a string.

For more information, see `ClockEnableInputPort`.

'ClockEnableOutputPort' - Clock enable output port name

`'ce_out'` (default) | string

Clock enable output port name, specified as a string.

For more information, see `ClockEnableOutputPort`.

'ClockInputPort' - Clock input port name

`'clk'` (default) | string

Clock input port name, specified as a string.

For more information, see `ClockInputPort`.

'ResetInputPort' - Reset input port name

`'reset'` (default) | string

Reset input port name, specified as a string.

For more information, see `ResetInputPort`.

File and Variable Names

'VerilogFileExtension' - Verilog file extension

`'.v'` (default) | string

For more information, see `VerilogFileExtension`.

'VHDLFileExtension' - VHDL file extension

`'.vhd'` (default) | string

For more information, see `VHDLFileExtension`.

'VHDLArchitectureName' - VHDL architecture name

`'rtl'` (default) | string

For more information, see `VHDLArchitectureName`.

'VHDLLibraryName' - VHDL library name

`'work'` (default) | string

For more information, see `VHDLLibraryName`.

'SplitEntityFilePostfix' - Postfix for VHDL entity file names

`'_entity'` (default) | string

For more information, see `SplitEntityFilePostfix`.

'SplitArchFilePostfix' - Postfix for VHDL architecture file names

`'_arch'` (default) | string

For more information, see `SplitArchFilePostfix`.

'PackagePostfix' - Postfix for package file name

`'_pkg'` (default) | string

For more information, see `PackagePostfix`.

'ComplexImagPostfix' - Postfix for imaginary part of complex signal

`'_im'` (default) | string

For more information, see `ComplexImagPostfix`.

'ComplexRealPostfix' - Postfix for imaginary part of complex signal names

`'_re'` (default) | string

For more information, see `ComplexRealPostfix`.

'EnablePrefix' - Prefix for internal enable signals

`'enb'` (default) | string

Prefix for internal clock enable and control flow enable signals, specified as a string.

For more information, see `EnablePrefix`.

Examples

Generate VHDL Test Bench

Generate VHDL DUT and test bench for a subsystem.

Use `makehdl` to generate VHDL code for the subsystem `symmetric_fir`.

```
makehdl('sfir_fixed/symmetric_fir')
```

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

```
### HDL code generation complete.
```

After `makehdl` is complete, use `makehdltb` to generate a VHDL test bench for the same subsystem.

```
makehdltb('sfir_fixed/symmetric_fir')
```

```
### Begin TestBench generation.
### Generating HDL TestBench for 'sfir_fixed/symmetric_fir'.
### Begin simulation of the model 'gm_sfir_fixed'...
### Collecting data...
### Generating test bench: hdlsrc\sfir_fixed\symmetric_fir_tb.vhd
### Creating stimulus vectors...
### HDL TestBench generation complete.
```

The generated VHDL test bench code is saved in the `hdlsrc` folder.

Generate Verilog Test Bench

Generate Verilog DUT and test bench for a subsystem.

Use `makehdl` to generate Verilog code for the subsystem `symmetric_fir`.

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

```
### Generating HDL for 'sfir_fixed/symmetric_fir'.
### Starting HDL check.
### HDL check for 'sfir_fixed' complete with 0 errors, 0 warnings,
    and 0 messages.
### Begin Verilog Code Generation for 'sfir_fixed'.
### Working on sfir_fixed/symmetric_fir as
    hdlsrc\sfir_fixed\symmetric_fir.v
### HDL code generation complete.
```

After `makehdl` is complete, use `makehdltb` to generate a Verilog test bench for the same subsystem.

```
makehdltb('sfir_fixed/symmetric_fir','TargetLanguage','Verilog')
```

```
### Begin TestBench generation.
### Generating HDL TestBench for 'sfir_fixed/symmetric_fir'.
### Begin simulation of the model 'gm_sfir_fixed'...
### Collecting data...
### Generating test bench: hdlsrc\sfir_fixed\symmetric_fir_tb.v
### Creating stimulus vectors...
### HDL TestBench generation complete.
```

The generated Verilog test bench code is saved in the `hdlsrc\sfir_fixed` folder.

See Also

`makehdl`

Purpose

Find and install support for third-party hardware or software

Syntax

`supportPackageInstaller`

Description

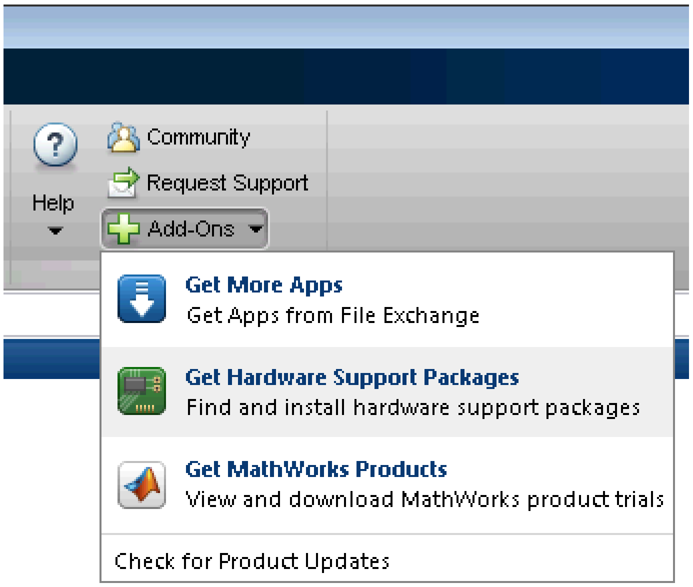
The `supportPackageInstaller` function opens *Support Package Installer*.

Support Package Installer can install *support packages*, which add support for specific third-party hardware or software to specific MathWorks products.

To see a list of available support packages, run Support Package Installer and advance to the second screen.

You can also start Support Package Installer in one of the following ways:

- On the MATLAB toolstrip, click **Add-Ons > Get Hardware Support Packages**.


Help
Community
Request Support
Add-Ons
Get More Apps
Get Apps from File Exchange
Get Hardware Support Packages
Find and install hardware support packages
Get MathWorks Products
View and download MathWorks product trials
Check for Product Updates

- Double-click a support package installation file (`*.mlpkginstall`).

See Also

`targetUpdater` | `matlabshared.supportpkg.checkForUpdate` | `matlabshared.supportpkg.getInstalled`

Supported Blocks

1-D Lookup Table

Purpose

1-D Lookup Table implementations, properties, and restrictions for HDL code generation

Description

The 1-D Lookup Table block is a one-dimensional version of the n-D Lookup Table block. For HDL code generation information, see n-D Lookup Table.

2-D Lookup Table

Purpose

2-D Lookup Table implementations, properties, and restrictions for HDL code generation

Description

The 2-D Lookup Table block is a one-dimensional version of the n-D Lookup Table block. For HDL code generation information, see n-D Lookup Table.

Purpose

Abs implementations, properties, and restrictions for HDL code generation

Description

The Abs block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Abs.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Add implementations, properties, and restrictions for HDL code generation

Description

The Add block is the same as the Sum block. For HDL code generation information, see Sum.

Assertion

Purpose

Assertion implementations, properties, and restrictions for HDL code generation

Description

The Assertion block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Assertion.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

Assignment implementations, properties, and restrictions for HDL code generation

Description

The Assignment block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Assignment.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Atomic Subsystem

Purpose

Atomic Subsystem implementations, properties, and restrictions for HDL code generation

Description

The Atomic Subsystem block is a version of the Subsystem block. For HDL code generation information, see Subsystem.

For information on the Simulink simulation behavior and block parameters, see Atomic Subsystem.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Bias implementations, properties, and restrictions for HDL code generation

Description

The Bias block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Bias.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Bit Clear implementations, properties, and restrictions for HDL code generation

Description

The Bit Clear block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Bit Clear.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Bit Concat implementations, properties, and restrictions for HDL code generation

Description

The Bit Concat block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Bit Concat.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Bit Reduce implementations, properties, and restrictions for HDL code generation

Description

The Bit Reduce block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Bit Reduce.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Bit Rotate implementations, properties, and restrictions for HDL code generation

Description

The Bit Rotate block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Bit Rotate.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Bit Set implementations, properties, and restrictions for HDL code generation

Description

The Bit Set block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Bit Set.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Bit Shift implementations, properties, and restrictions for HDL code generation

Description

The Bit Shift block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Bit Shift.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Bit Slice implementations, properties, and restrictions for HDL code generation

Description

The Bit Slice block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Bit Slice.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Bitwise Operator implementations, properties, and restrictions for HDL code generation

Description

The Bitwise Operator block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Bitwise Operator.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Biquad Filter

Purpose

Biquad Filter implementations, properties, and restrictions for HDL code generation

Description

The Biquad Filter block is available with DSP System Toolbox™.

For information on the Simulink simulation behavior and block parameters, see Biquad Filter.

HDL Implementations

Programmable Filter Support

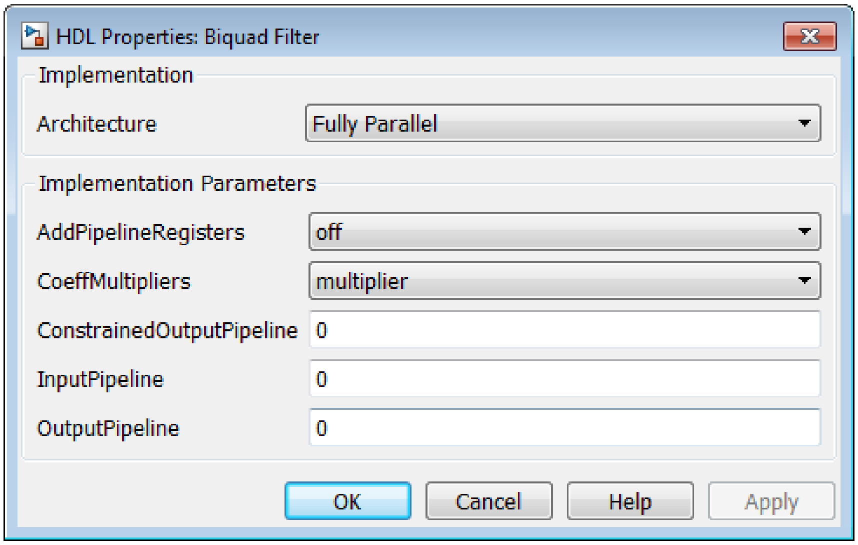
The coder supports programmable filters for Biquad Filters. Fully parallel and applicable serial architectures are supported.

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

Serial Architecture Support

Biquad Filter block supports fully parallel, fully serial, and partly serial architectures for `Direct form I` and `Direct form II` filter structures. Serial architecture is not supported for `Direct form I transposed` and `Direct form II transposed` filter structures.

- Fully Parallel (default): AddPipelineRegisters, CoeffMultipliers, ConstrainedOutputPipeline, InputPipeline, OutputPipeline



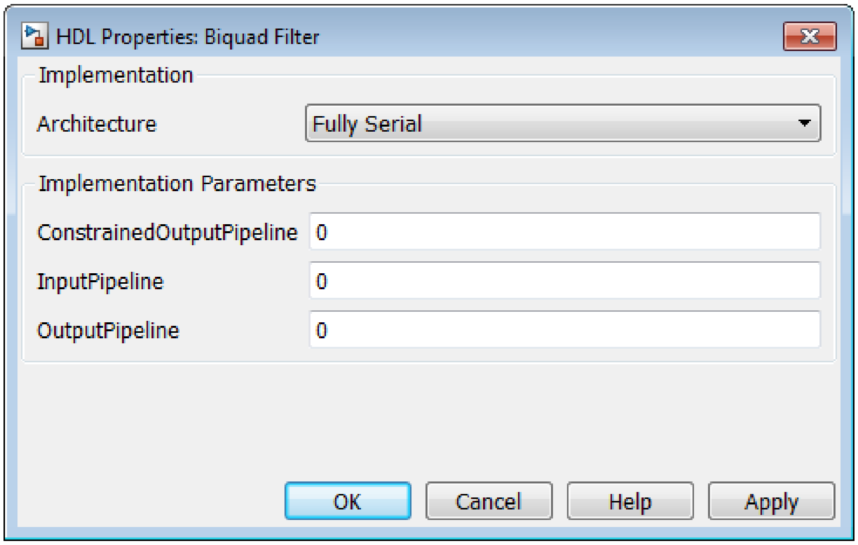
Implementation

Architecture	Fully Parallel

Implementation Parameters

AddPipelineRegisters	off
CoeffMultipliers	multiplier
ConstrainedOutputPipeline	0
InputPipeline	0
OutputPipeline	0

- Fully Serial: ConstrainedOutputPipeline, InputPipeline, OutputPipeline


HDL Properties: Biquad Filter

Implementation

Architecture: Fully Serial

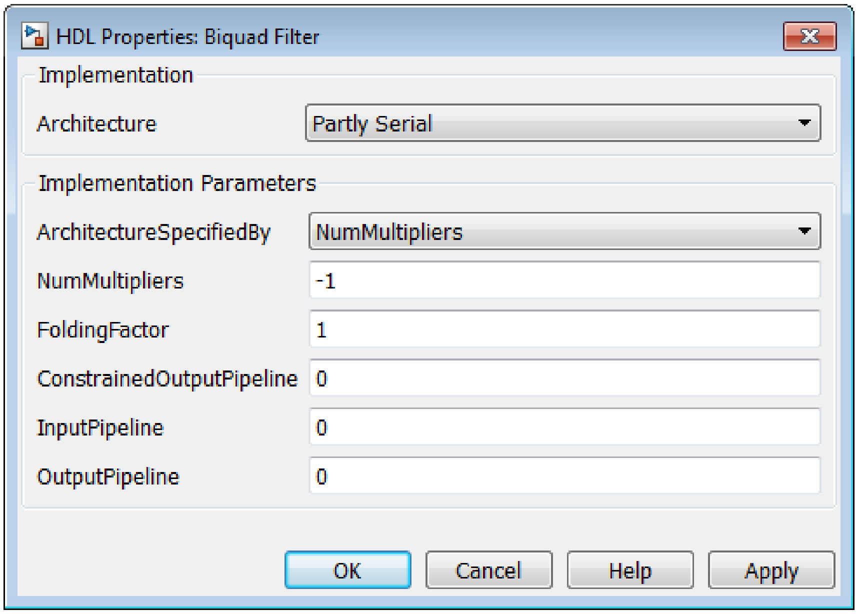
Implementation Parameters

ConstrainedOutputPipeline: 0

InputPipeline: 0

OutputPipeline: 0

- Partly Serial: ArchitectureSpecifiedBy, NumMultipliers, FoldingFactor, ConstrainedOutputPipeline, InputPipeline, OutputPipeline



Architecture: Partly Serial

ArchitectureSpecifiedBy: NumMultipliers

NumMultipliers: -1

FoldingFactor: 1

ConstrainedOutputPipeline: 0

InputPipeline: 0

OutputPipeline: 0

AddPipelineRegisters Support

When you use **AddPipelineRegisters**, registers are placed based on filter implementation. The pipeline register placement determines the latency.

Implementation	Pipeline Register Placement	Latency (clock cycles)
Biquad Filter	Pipeline registers are added between the filter sections.	Where `NS` is number of sections: `NS-1`

HDL Filter Properties

For HDL filter property descriptions, see “HDL Filter Block Properties”.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Complex Coefficients and Data Support

The coder supports use of complex coefficients and complex input signals for all filter structures of the Biquad Filter block, except decimators and interpolators. 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	Complex Data and Coefficients
`dfilt.dffir`	Y	Y	Y
`dfilt.dfsymfir`	Y	Y	Y
`dfilt.dfasymfir`	Y	Y	Y
`dfilt.dffirt`	Y	Y	Y
`dfilt.scalar`	Y	Y	Y
`dfilt.delay`	Y	N/A	N/A
`mfilt.cicdecim`	Y	N/A	N/A
`mfilt.cicinterp`	Y	N/A	N/A
`mfilt.firdecim`	Y	Y	N
`mfilt.firinterp`	Y	Y	N
`dfilt.df1sos`	Y	Y	Y
`dfilt.df1tsos`	Y	Y	Y
`dfilt.df2sos`	Y	Y	Y
`dfilt.df2tsos`	Y	Y	Y

Restrictions

- Data vector and frame inputs are not supported for HDL code generation.

- **Initial conditions** must be set to zero. HDL code generation is not supported for nonzero initial states.
- **Optimize unity scale values** must be selected.

Programmable filters are not supported for:

- `CoeffMultipliers` as `csd` or `factored-csd`

BPSK Demodulator Baseband

Purpose

BPSK Demodulator Baseband implementations, properties, and restrictions for HDL code generation

Description

The BPSK Demodulator Baseband block is available with Communications System Toolbox™.

For information on the Simulink simulation behavior and block parameters, see BPSK Demodulator Baseband.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

BPSK Modulator Baseband

Purpose

BPSK Modulator Baseband implementations, properties, and restrictions for HDL code generation

Description

The BPSK Modulator Baseband block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see BPSK Modulator Baseband.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Bus Creator

Purpose

Bus Creator implementations, properties, and restrictions for HDL code generation

Description

The Bus Creator block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Bus Creator.

HDL Implementations

This block has a single default HDL architecture.

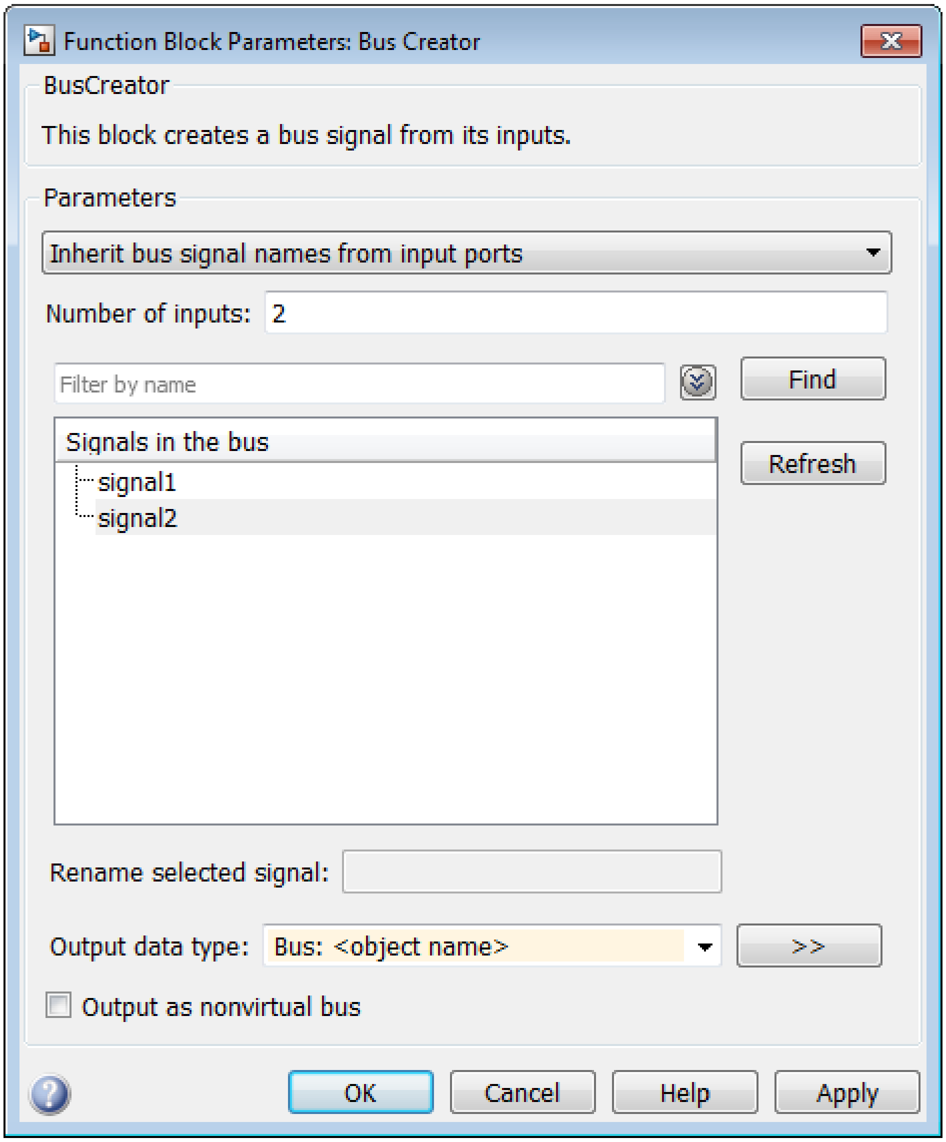
HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

Setup

- Set the **Simulation > Configuration Parameters > Diagnostics > Connectivity→Mux blocks used to create bus signals** parameter to `error`. For details, see “Prevent Bus and Mux Mixtures”.
- Make sure **Output as nonvirtual bus** is *not* checked.
- Make sure Bus Creator output is a BusObject.


Function Block Parameters: Bus Creator

BusCreator

This block creates a bus signal from its inputs.

Parameters

Inherit bus signal names from input ports

Number of inputs: 2

Rename selected signal:

Output data type: Bus: <object name>

☐ Output as nonvirtual bus

Concepts

- "Buses"

Purpose

Bus Selector implementations, properties, and restrictions for HDL code generation

Description

The Bus Selector block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Bus Selector.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

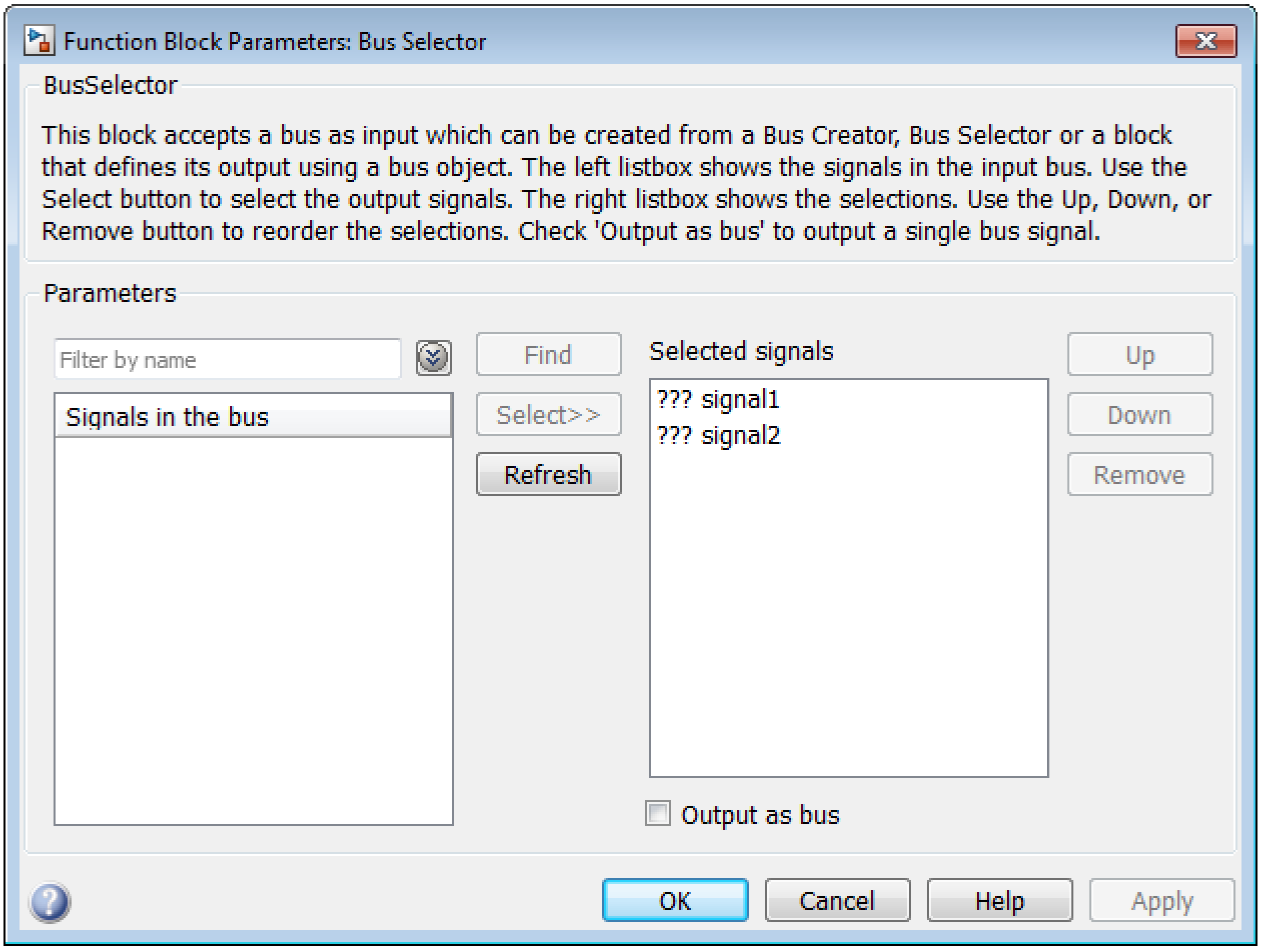
Restrictions

- “Setup” on page 2-29
- “Limitations” on page 2-30

Setup

- Set the **Simulation > Configuration Parameters > Diagnostics > Connectivity→Mux blocks used to create bus signals** parameter to `error`. For details, see “Prevent Bus and Mux Mixtures”.

- In the Bus Selector dialog, make sure **Output as a bus** is *not* checked.



Function Block Parameters: Bus Selector

BusSelector

This block accepts a bus as input which can be created from a Bus Creator, Bus Selector or a block that defines its output using a bus object. The left listbox shows the signals in the input bus. Use the Select button to select the output signals. The right listbox shows the selections. Use the Up, Down, or Remove button to reorder the selections. Check 'Output as bus' to output a single bus signal.

Parameters

Filter by name

Signals in the bus

Selected signals

??? signal1
??? signal2

☐ Output as bus

Limitations

You cannot select **Output as bus** for HDL code generation.

Concepts

- "Buses"

Purpose

Chart implementations, properties, and restrictions for HDL code generation

Description

The Chart block is available with Stateflow®.

For information on the Simulink simulation behavior and block parameters, see Chart.

HDL Implementations

This block has a single default HDL architecture.

Registered Output

If you want to insert an output register that delays the chart output by a simulation cycle, use the OutputPipeline block property.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Restrictions

- “Location of Charts in the Model” on page 2-31
- “Data Types” on page 2-32
- “Chart Initialization” on page 2-32
- “Imported Code” on page 2-33
- “Input and Output Events” on page 2-33
- “Loops” on page 2-34
- “Other Restrictions” on page 2-34

Location of Charts in the Model

A chart intended for HDL code generation must be part of a Simulink subsystem. If the chart for which you want to generate code is at the root level of your model, embed the chart in a subsystem and connect the relevant signals to the subsystem inputs and outputs.

Data Types

The current release supports a subset of MATLAB data types in charts intended for use in HDL code generation. Supported data types are

- Signed and unsigned integer
- Double and single

Note Results obtained from HDL code generated for models using double or single data types might not be bit-true to results obtained from simulation of the original model.

- Fixed point
- Boolean

Note Multidimensional arrays of these types are supported, with the exception of data types assigned to ports. Port data types must be either scalar or vector.

Chart Initialization

You must enable the chart property **Execute (enter) Chart at Initialization**. This option executes the update chart function immediately following chart initialization. The option is required for HDL because outputs must be available at time 0 (hardware reset). "Execution of a Chart at Initialization" describes existing restrictions under this property.

The reset action must not entail the delay of combinatorial logic. Therefore, do not perform arithmetic in initialization actions.

The chart property **Initialize Outputs Every Time Chart Wakes Up** controls whether or not output is persistent.

Selecting **Initialize Outputs Every Time Chart Wakes Up** generates HDL code that is more readable and has better synthesis results.

Imported Code

A chart intended for HDL code generation must be entirely self-contained. The following restrictions apply:

- Do not call MATLAB functions other than `min` or `max`.
- Do not use MATLAB workspace data.
- Do not call C math functions. There is no HDL counterpart to the C math library.
- If the **Enable bit operations** property is disabled, do not use the exponentiation operator (`^`). The exponentiation operator is implemented with the C Math Library function `pow`.
- Do not include custom code. Information entered in the **Simulation Target > Custom Code** pane of the Configuration Parameters dialog box is ignored.
- Do not share data (via Data Store Memory blocks) between charts. The coder does not map such global data to HDL, because HDL does not support global data.

Input and Output Events

The coder supports the use of input and output events with Stateflow charts, subject to the following constraints:

- You can define and use one and only one input event per Stateflow chart. (There is no restriction on the number of output events you can use.)
- The coder does not support HDL code generation for charts that have a single input event, and which also have nonzero initial values on the chart's output ports.

- All input and output events must be edge-triggered.

For detailed information on input and output events, see “Activate a Stateflow Chart Using Input Events”and “Activate a Simulink Block Using Output Events” in the Stateflow documentation.

Loops

Do not explicitly use loops other than `for` loops in a chart intended for HDL code generation. Observe the following restrictions on `for` loops:

- The data type of the loop counter variable must be `int32`.
- The coder supports only constant-bounded loops.

The `for` loop example, `sf_for`, shows a design pattern for a `for` loop using a graphical function.

Other Restrictions

The coder imposes a number of additional restrictions on the use of classic chart features. These limitations exist because HDL does not support some features of general-purpose sequential programming languages.

- Do not define local events in a chart from which HDL code is to be generated.

  Do not use the following implicit events:

  - `enter`
  - `exit`
  - `change`

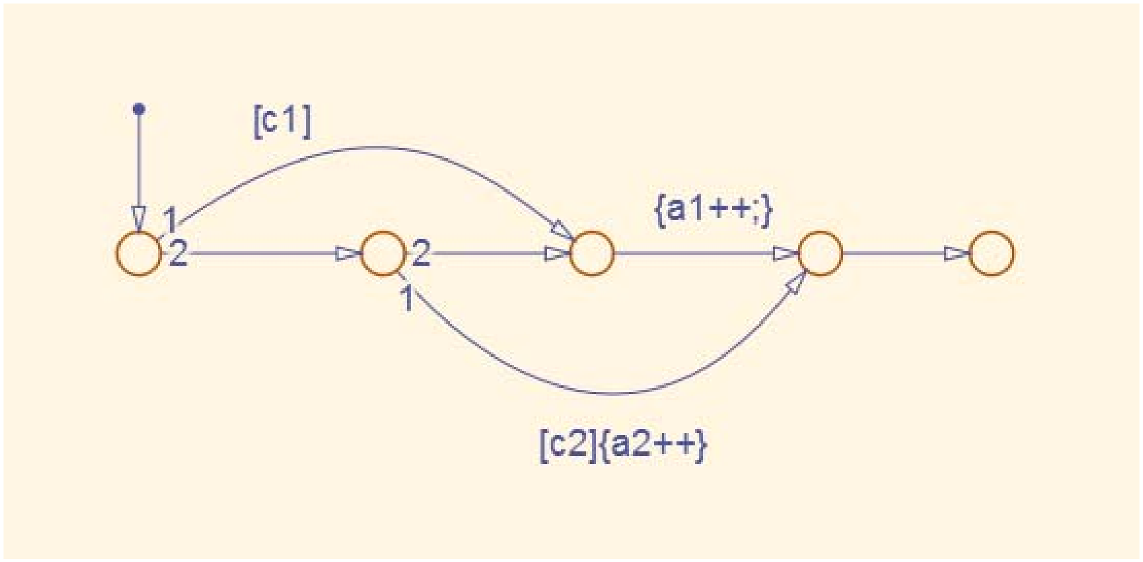
  You can use the following implicit events:

  - `wakeup`
  - `tick`

Temporal logic can be used provided the base events are limited to these types of implicit events.

Note Absolute-time temporal logic is not supported for HDL code generation.

- Do not use recursion through graphical functions. The coder does not currently support recursion.
- Avoid unstructured code. Although charts allow unstructured code to be written (through transition flow diagrams and graphical functions), this usage results in `goto` statements and multiple function return statements. HDL does not support either `goto` statements or multiple function return statements. Therefore, do not use unstructured flow diagrams, such as the flow diagram shown in the following figure.



- Do not read from output ports if you do not have the **Initialize Outputs Every Time Chart Wakes Up** chart option selected.
- Do not use Data Store Memory objects.
- Do not use pointer (&) or indirection (*) operators. See the discussion of "Pointer and Address Operations".
- If a chart gets a runtime overflow error during simulation, it is possible to disable data range error checking and generate HDL code for the chart. However, in such cases results obtained from the generated HDL code might not be bit-true to results obtained from the simulation. Recommended practice is to enable overflow checking and eliminate overflow conditions from the model during simulation.

See Also

State Transition Table | Truth Table

Related Examples

- "Generate HDL for Mealy and Moore Finite State Machines"
- "Design Patterns Using Advanced Chart Features"

Concepts

- "Hardware Realization of Stateflow Semantics"

Purpose

Check Dynamic Gap implementations, properties, and restrictions for HDL code generation

Description

The Check Dynamic Gap block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Check Dynamic Gap.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Check Dynamic Range

Purpose

Check Dynamic Range implementations, properties, and restrictions for HDL code generation

Description

The Check Dynamic Range block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Check Dynamic Range.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

Check Static Gap implementations, properties, and restrictions for HDL code generation

Description

The Check Static Gap block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Check Static Gap.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Check Static Range

Purpose

Check Static Range implementations, properties, and restrictions for HDL code generation

Description

The Check Static Range block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Check Static Range.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Check Discrete Gradient

Purpose

Check Discrete Gradient implementations, properties, and restrictions for HDL code generation

Description

The Check Discrete Gradient block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Check Discrete Gradient.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Check Dynamic Lower Bound

Purpose

Check Dynamic Lower Bound implementations, properties, and restrictions for HDL code generation

Description

The Check Dynamic Lower Bound block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Check Dynamic Lower Bound.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

Check Dynamic Upper Bound implementations, properties, and restrictions for HDL code generation

Description

The Check Dynamic Upper Bound block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Check Dynamic Upper Bound.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Check Input Resolution

Purpose

Check Input Resolution implementations, properties, and restrictions for HDL code generation

Description

The Check Input Resolution block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Check Input Resolution.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

Check Static Lower Bound implementations, properties, and restrictions for HDL code generation

Description

The Check Static Lower Bound block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Check Static Lower Bound.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Check Static Upper Bound

Purpose

Check Static Upper Bound implementations, properties, and restrictions for HDL code generation

Description

The Check Static Upper Bound block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Check Static Upper Bound.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

CIC Decimation implementations, properties, and restrictions for HDL code generation

Description

The CIC Decimation block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see CIC Decimation.

The coder supports both **Coefficient source** options (**Dialog parameters** or **Multirate filter object (MFILT)**).

When you select **Multirate filter object (MFILT)**, you can enter either a filter object name or a direct filter specification in the **Multirate filter variable** field.

HDL Implementations

AddPipelineRegisters Support

When you use **AddPipelineRegisters**, registers are placed based on filter implementation. The pipeline register placement determines the latency.

Implementation	Pipeline Register Placement	Latency (clock cycles)
CIC Decimation	A pipeline register is added between the comb stages of the differentiators .	Where `NS` is number of sections (at the input side): `NS-1`

HDL Filter Properties

For HDL filter property descriptions, see "HDL Filter Block Properties".

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Restrictions

- Vector and frame inputs are not supported for HDL code generation.

- When you select **Multirate filter object (MFILT)**, the filter object specified in the **Multirate filter variable** field must be a `mfilt.cicdecim` object. If you specify some other type of filter object, an error will occur.
- When you select **Dialog parameters**, the **Filter Structure** option `Zero-latency decimator` is not supported for HDL code generation. Select `Decimator` in the **Filter Structure** pulldown menu.

Purpose

CIC Interpolation implementations, properties, and restrictions for HDL code generation

Description

The CIC Interpolation block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see CIC Interpolation.

The coder supports both **Coefficient source** options (**Dialog parameters** or **Multirate filter object (MFILT)**). When you select **Multirate filter object (MFILT)**, you can enter either a filter object name or a direct filter specification in the **Multirate filter variable** field.

HDL Implementations

AddPipelineRegisters Support

When you use **AddPipelineRegisters**, registers are placed based on filter implementation. The pipeline register placement determines the latency.

Implementation	Pipeline Register Placement	Latency (clock cycles)
CIC Interpolation	A pipeline register is added between the comb stages of the differentiators .	Where `NS` is number of sections (at the input side): `NS`

HDL Filter Properties

For HDL filter property descriptions, see “HDL Filter Block Properties”.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

- Vector and frame inputs are not supported for HDL code generation.
- When you select **Multirate filter object (MFILT)**, the filter object specified in the **Multirate filter variable** field must be a

`mfilt.cicinterp` object. If you specify some other type of filter object, an error will occur.

- When you select **Dialog parameters**, the **Filter Structure** option `Zero-latency interpolator` is not supported for HDL code generation. Select `Interpolator` in the **Filter Structure** dropdown menu.

Compare To Constant

Purpose

Compare To Constant implementations, properties, and restrictions for HDL code generation

Description

The Compare To Constant block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Compare To Constant.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Compare To Zero

Purpose

Compare To Zero implementations, properties, and restrictions for HDL code generation

Description

The Compare To Zero block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Compare To Zero.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Complex to Real-Imag

Purpose

Complex to Real-Imag implementations, properties, and restrictions for HDL code generation

Description

The Complex to Real-Imag block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Complex to Real-Imag.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Complex Data Support

This block supports code generation for complex signals.

Constant

Purpose

Constant implementations, properties, and restrictions for HDL code generation

Description

The Constant block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Constant.

HDL Implementations

Implementations	Parameters	Description
`default Constant`	None	This implementation emits the value of the Constant block.
`Logic Value`	None	By default, this implementation emits the character `'Z'` for each bit in the signal. For example, for a 4-bit signal, the implementation would emit `'ZZZZ'`.
	`{'Value', 'Z'}`	Use this parameter value if the signal is in a high-impedance state. This implementation emits the character `'Z'` for each bit in the signal. For example, for a 4-bit signal, the implementation would emit `'ZZZZ'`.
	`{'Value', 'X'}`	Use this parameter value if the signal is in an unknown state. This implementation emits the character `'X'` for each bit in the signal. For example, for a 4-bit signal, the implementation would emit `'XXXX'`.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Restrictions

- The `Logic Value` implementation does not support the `double` data type. If you specify this implementation for a Constant of type `double`, a code-generation error occurs.
- For **Sample time**, enter -1. Delay balancing does not support an `inf` sample time.

Constellation Diagram

Purpose

Constellation Diagram implementations, properties, and restrictions for HDL code generation

Description

The Constellation Diagram block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Constellation Diagram.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

Convert 1-D to 2-D implementations, properties, and restrictions for HDL code generation

Description

The Convert 1-D to 2-D block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Convert 1-D to 2-D.

HDL Implementations

This block has a pass-through implementation.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Convolutional Deinterleaver

Purpose

Convolutional Deinterleaver implementations, properties, and restrictions for HDL code generation

Description

The Convolutional Deinterleaver block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Convolutional Deinterleaver.

HDL Implementations

- "Shift Register Based Implementation" on page 2-58
- "RAM Based Implementation" on page 2-58

Shift Register Based Implementation

The default implementation for the Convolutional Deinterleaver block is shift register based. If you want to suppress generation of reset logic, set the implementation parameter `ResetType` to`'none'`.

Note that when you set `ResetType` to`'none'`, reset is not applied to the shift registers. Mismatches between Simulink and the generated code occur for some number of samples during the initial phase, when registers are not fully loaded. To avoid spurious test bench errors, determine the number of samples required to fully load the shift registers. Then, set the **Ignore output data checking (number of samples)** option accordingly. (You can use the `IgnoreDataChecking` property for this purpose, if you are using the command-line interface.)

RAM Based Implementation

When you select the `RAM` implementation for a Convolutional Deinterleaver block, the coder uses RAM resources instead of shift registers.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Restrictions

When you select the `RAM` implementation:

- Double or single data types are not supported for either input or output signals.
- **Initial conditions** for the block must be set to zero.
- At least two rows of interleaving are required.

Convolutional Encoder

Purpose

Convolutional Encoder implementations, properties, and restrictions for HDL code generation

Description

The Convolutional Encoder block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Convolutional Encoder.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

Input data requirements:

- Must be sample-based,
- Must have `boolean` or `ufix1` data type.

The coder supports only the following coding rates:

- ½ to 1/7
- 2/3

The coder supports only constraint lengths for 3 to 9.

Trellis structure must be specified by the `poly2trellis` function.

The coder supports the following **Operation mode** settings:

- `Continuous`
- `Reset on nonzero input via port`

  If you select this mode, you must select the **Delay reset action to next time step** option. When you select this option, the

Convolutional Encoder block finishes its current computation before executing a reset.

Convolutional Interleaver

Purpose

Convolutional Interleaver implementations, properties, and restrictions for HDL code generation

Description

The Convolutional Interleaver block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Convolutional Interleaver.

HDL Implementations

- "Shift Register Based Implementation" on page 2-62
- "RAM Based Implementation" on page 2-62

Shift Register Based Implementation

The default implementation for the Convolutional Interleaver block is shift register based. If you want to suppress generation of reset logic, set the implementation parameter `ResetType` to `'none'`.

Note that when you set `ResetType` to `'none'`, reset is not applied to the shift registers. Mismatches between Simulink and the generated code occur for some number of samples during the initial phase, when registers are not fully loaded. To avoid spurious test bench errors, determine the number of samples required to fully load the shift registers. Then, set the **Ignore output data checking (number of samples)** option accordingly. (You can use the `IgnoreDataChecking` property for this purpose, if you are using the command-line interface.)

RAM Based Implementation

When you select the `RAM` implementation for a Convolutional Interleaver block, the coder uses RAM resources instead of shift registers.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Restrictions

When you select the `RAM` implementation:

- Double or single data types are not supported for either input or output signals.

- **Initial conditions** for the block must be set to zero.
- At least two rows of interleaving are required.

Purpose

Cosine implementations, properties, and restrictions for HDL code generation

Description

The Cosine block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Sine, Cosine.

HDL Implementations

The HDL code implements Cosine using the quarter-wave lookup table you specify in the Simulink block parameters.

To avoid generating a division operator (`/`) in the HDL code, for **Number of data points for lookup table**, enter `(2^`*`n`*`)+1`, where *n* is an integer.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Restrictions

This block does not have restrictions for HDL code generation.

If you see the following warnings for the Sine or Cosine block, you can ignore them:

- `HDL code generation for the Lookup Table (n-D) block does not support out-of-range inputs.  Set the "Diagnostic for out of range input" block parameter to "Error" to suppress this warning.`
- `Using linear interpolation on the Lookup Table (n-D) block, may require using a divide operator in the generated HDL, which may not be synthesizable.`

Purpose

Counter Free-Running implementations, properties, and restrictions for HDL code generation

Description

The Counter Free-Running block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Counter Free-Running.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose Counter Limited implementations, properties, and restrictions for HDL code generation

Description The Counter Limited block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Counter Limited.

HDL Implementations This block has a single default HDL architecture.

HDL Block Properties For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Data Type Conversion implementations, properties, and restrictions for HDL code generation

Description

The Data Type Conversion block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Data Type Conversion.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Restrictions

If you configure a Data Type Conversion block for double to fixed-point or fixed-point to double conversion, a warning displays during code generation.

Data Type Duplicate

Purpose

Data Type Duplicate implementations, properties, and restrictions for HDL code generation

Description

The Data Type Duplicate block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Data Type Duplicate.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

Data Type Propagation implementations, properties, and restrictions for HDL code generation

Description

The Data Type Propagation block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Data Type Propagation.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

Decrement Real World implementations, properties, and restrictions for HDL code generation

Description

The Decrement Real World block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Decrement Real World.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Decrement Stored Integer implementations, properties, and restrictions for HDL code generation

Description

The Decrement Stored Integer block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Decrement Stored Integer.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Delay implementations, properties, and restrictions for HDL code generation

Description

The Delay block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Delay.

HDL Implementations

To generate a reset port in the HDL code, set **External reset** to `Level`.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Delay implementations, properties, and restrictions for HDL code generation

Description

The Delay block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Delay.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Demux implementations, properties, and restrictions for HDL code generation

Description

The Demux block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Demux.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Digital Filter implementations, properties, and restrictions for HDL code generation

Description

The Digital Filter block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Digital Filter.

HDL Implementations

When you specify `SerialPartition` and `ReuseAccum` for a Digital Filter block, observe the following constraints.

- If you specify **Dialog parameters** as the `Coefficient source`:
  - Set **Transfer function type** to `FIR (all zeros)`.
  - Select **Filter structure** as one of : `Direct form,`, `Direct form symmetric`, or `Direct form asymmetric`.

- If you specify **Discrete-time filter object** as the `Coefficient source`, the filter object must be one of the following:
  - `dfilt.dffir`
  - `dfilt.dfsymfir`
  - `dfilt.dfasymfir`

Distributed Arithmetic Support

Distributed Arithmetic properties **DALUTPartition** and **DARadix** are supported for the following filter structures.

Implementation	Supported FIR Structures
default	• dfilt.dffir • dfilt.dfsymfir • dfilt.dfasymfir

AddPipelineRegisters Support

When you use **AddPipelineRegisters**, registers are placed based on filter implementation. The pipeline register placement determines the latency.

Implementation	Pipeline Register Placement	Latency (clock cycles)
FIR, Asymmetric FIR, and Symmetric FIR filters	A pipeline register is added between levels of a tree-based adder.	Where `FL` is the filter length: `ceil(log2(FL)`
FIR Transposed	A pipeline register is added after the products.	1
IIR SOS	Pipeline registers are added between the filter sections.	Where `NS` is number of sections: `NS-1`

HDL Filter Properties

For HDL filter property descriptions, see "HDL Filter Block Properties".

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Complex Coefficients and Data Support

The coder supports use of complex coefficients and complex input signals for all filter structures of the Digital Filter block, except decimators and interpolators. 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	Complex Data and Coefficients
`dfilt.dffir`	Y	Y	Y
`dfilt.dfsymfir`	Y	Y	Y
`dfilt.dfasymfir`	Y	Y	Y
`dfilt.dffirt`	Y	Y	Y
`dfilt.scalar`	Y	Y	Y
`dfilt.delay`	Y	N/A	N/A
`mfilt.cicdecim`	Y	N/A	N/A
`mfilt.cicinterp`	Y	N/A	N/A
`mfilt.firdecim`	Y	Y	N
`mfilt.firinterp`	Y	Y	N
`dfilt.df1sos`	Y	Y	Y
`dfilt.df1tsos`	Y	Y	Y
`dfilt.df2sos`	Y	Y	Y
`dfilt.df2tsos`	Y	Y	Y

Restrictions

- If you select the Digital Filter block **Discrete-time filter object** option, you must have the DSP System Toolbox software to generate code for the block.
- **Initial conditions** must be set to zero. HDL code generation is not supported for nonzero initial states.
- The coder does not support the Digital Filter block **Input port(s)** option for HDL code generation.

Direct Lookup Table (n-D)

Purpose

Direct Lookup Table (n-D) implementations, properties, and restrictions for HDL code generation

Description

The Direct Lookup Table (n-D) block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Direct Lookup Table (n-D).

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Restrictions

- "Required Block Settings" on page 2-78
- "Table Data Typing and Sizing" on page 2-78

Required Block Settings

- **Number of table dimensions**: The coder supports a maximum dimension of 2.
- **Inputs select this object from table**: Select `Element`.
- **Make table an input**: Clear this check box.
- **Diagnostic for out-of-range input**: Select `Error`. If you select other options, the coder displays a warning.

Table Data Typing and Sizing

- It is good practice to size each dimension in the table to be a power of two. The coder issues a warning if the length of a dimension (*except* the innermost dimension) is not a power of two. By following this practice, you can avoid multiplications during table indexing operations and realize a more efficient table in hardware.

- Table data must resolve to a nonfloating-point data type. The coder examines the output port to verify that its data type meets this requirement.
- All ports on the block require scalar values.

Discrete FIR Filter

Purpose

Discrete FIR Filter implementations, properties, and restrictions for HDL code generation

Description

The Discrete FIR Filter block is available with Simulink, but a DSP System Toolbox license is required to use a filter structure other than Direct Form.

For information on the Simulink simulation behavior and block parameters, see Discrete FIR Filter.

Multichannel Filter Support

The coder supports the use of vector inputs for Discrete FIR Filters.

1 Connect vector signals to Discrete FIR block input port.

2 Specify **Input processing** as `Elements as channels (sample based)`.

3 Specify architecture and implementation parameters.

4 Specify channel sharing option as on for fully parallel support.

5 Generate HDL code.

Programmable Filter Support

The coder supports programmable filters for Discrete FIR Filters. Fully parallel and applicable serial architectures are supported.

1 Select **Input port(s**) as coefficient source on the filter block mask.

2 Connect the coefficient port with a vector signal.

3 Specify the implementation architecture and parameters from the HDL Coder property interface.

4 Generate HDL code.

HDL Implementations

When you specify `SerialPartition` and `ReuseAccum` for a Discrete FIR Filter block, select **Filter structure** as one of the following:

- `Direct form`
- `Direct form symmetric`
- `Direct form asymmetric`

Distributed Arithmetic Support

Distributed Arithmetic properties **DALUTPartition** and **DARadix**are supported for the following filter structures.

Implementation	Supported FIR Structures
default	• dfilt.dffir • dfilt.dfsymfir • dfilt.dfasymfir

HDL Filter Properties

For HDL filter property descriptions, see “HDL Filter Block Properties”.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

- The coder does not support unsigned inputs for the Discrete FIR Filter block.
- **Initial conditions** must be set to zero. HDL code generation is not supported for nonzero initial states.
- The coder does not support the following options of the Discrete FIR Filter block:
  - **Filter Structure** : `Lattice MA`
- **CoeffMultipliers** options are supported only when using a fully parallel architecture. The **CoeffMultipliers** property is hidden

from the HDL Block Properties dialog box when you select a serial architecture.

Programmable filters are not supported for:

- Implementations for which you specify the coefficients by dialog parameters (for example, complex input and coefficients with serial architecture)
- Distributed Arithmetic (DA)
- CoeffMultipliers as `csd` or `factored-csd`

Related Examples

- Generate HDL Code for FIR Programmable Filter

Purpose

Discrete PID Controller implementations, properties, and restrictions for HDL code generation

Description

The Discrete PID Controller block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Discrete PID Controller.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Discrete Transfer Fcn

Purpose

Discrete Transfer Fcn implementations, properties, and restrictions for HDL code generation

Description

The Discrete Transfer Fcn block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Discrete Transfer Fcn.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

- You must use the **Inherit: Inherit via internal rule** option for data type propagation if, and only if, the input data type is double.
- Frame, matrix, and vector input data types are not supported.
- The leading denominator coefficient (a0) must be 1 or -1.

The Discrete Transfer Fcn block cannot participate in the following optimizations:

- Resource sharing
- Distributed pipelining

Purpose

Discrete-Time Eye Diagram Scope implementations, properties, and restrictions for HDL code generation

Description

The Discrete-Time Eye Diagram Scope block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Discrete-Time Eye Diagram Scope.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Discrete-Time Signal Trajectory Scope

Purpose

Discrete-Time Signal Trajectory Scope implementations, properties, and restrictions for HDL code generation

Description

The Discrete-Time Signal Trajectory Scope block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Discrete-Time Signal Trajectory Scope.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

Discrete-Time Integrator implementations, properties, and restrictions for HDL code generation

Description

The Discrete-Time Integrator block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Discrete-Time Integrator.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Restrictions

- Use of state ports is not supported for HDL code generation. Clear the **Show state port** option.
- Use of external initial conditions is not supported for HDL code generation. Set **Initial condition source** to `Internal`.
- Width of input and output signals must not exceed 32 bits.

Display

Purpose

Display implementations, properties, and restrictions for HDL code generation

Description

The Display block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Display.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

Divide implementations, properties, and restrictions for HDL code generation

Description

The Divide block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Divide.

HDL Implementations

To perform an HDL-optimized divide operation, connect a Product block to a Divide block in reciprocal mode. For information about the Divide block in reciprocal mode, see "Reciprocal Mode" on page 2-89.

Default Mode

In default mode, the Divide block supports only integer data types for HDL code generation.

Implementations	Parameters	Description
`default` `Linear`	None	Generate a divide (`/`) operator in the HDL code.

Reciprocal Mode

The Divide block is in reciprocal mode when **Number of Inputs** is set to `/`. In reciprocal mode, the Divide block has the HDL block implementations described in the following table.

Implementations	Parameters	Description
`default` `Linear`	None	When you compute a reciprocal, use the HDL divide (/) operator to implement the division.
`RecipNewton`	`Iterations`	Use the iterative Newton method. Select this option to optimize area. The default value for `Iterations` is 3. The recommended value for `Iterations` is between 2 and 10. The coder generates a message if `Iterations` is outside the recommended range.
`RecipNewtonSingleRate`	`Iterations`	Use the single rate pipelined Newton method. Select this option to optimize speed, or if you want a single rate implementation. The default value for `Iterations` is 3. The recommended value for `Iterations` is between 2 and 10. The coder generates a message if `Iterations` is outside the recommended range.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Complex Data Support

This block does not support code generation for division with complex signals.

Restrictions

When you use the Divide block in reciprocal mode, the following restrictions apply:

- The input must be scalar and must have integer or fixed-point (signed or unsigned) data type.
- The output must be scalar and have integer or fixed-point (signed or unsigned) data type.
- Only the `Zero` rounding mode is supported.
- The **Saturate on integer overflow** option on the block must be selected.

DocBlock

Purpose

DocBlock implementations, properties, and restrictions for HDL code generation

Description

The DocBlock block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see DocBlock.

HDL Implementations

This block has a single default HDL architecture.

Restrictions

Set the **Document type** parameter of the DocBlock to `Text`. The coder does not support the `HTML` or `RTF` options.

Related Examples

- “Generate Code with Annotations or Comments”

Purpose

Dot Product implementations, properties, and restrictions for HDL code generation

Description

The Dot Product block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Dot Product.

HDL Implementations

Implementations	Description
`Linear` (default)	Generates a chain of N operations (multipliers) for N inputs.
`Tree`	This implementation has minimal latency but is large and slow. It generates a tree-shaped structure of multipliers.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Downsample

Purpose

Downsample implementations, properties, and restrictions for HDL code generation

Description

The Downsample block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Downsample.

Best Practices

It is good practice to follow the Downsample block with a unit delay. This will prevent the code generator from inserting an extra bypass register in the HDL code. See “HDL Optimized QPSK Transmitter” for an example.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose DSP Constant (Obsolete) implementations, properties, and restrictions for HDL code generation

Description HDL support for the DSP Constant (Obsolete) block will be removed in a future release. Use the Constant block instead.

Dual Port RAM

Purpose

Dual Port RAM implementations, properties, and restrictions for HDL code generation

Description

The Dual Port RAM block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Dual Port RAM.

HDL Implementations

This block has a single default HDL architecture.

HDL code generated for RAM blocks has:

- A latency of 1 clock cycle for read data output.
- No reset signal, since some synthesis tools do not infer a RAM from HDL code if it includes a reset.

Code generation for a RAM block creates a separate file, `blockname.ext`, where `blockname` is derived from the name of the RAM block, and `ext` is the target language filename extension.

RAM Initialization

Code generated to initialize a RAM is intended for simulation only, and may be ignored by synthesis tools.

Implement RAM With or Without Clock Enable

The HDL block property, `RAMArchitecture` , enables or suppresses generation of clock enable logic for all RAM blocks in a subsystem. You can set `RAMArchitecture` to the following values:

- `WithClockEnable` (default): Generates RAMs using HDL templates that include a clock enable signal, and an empty RAM wrapper.
- `WithoutClockEnable`: Generates RAMs without clock enables, and a RAM wrapper that implements the clock enable logic.

Some synthesis tools may not infer RAMs with a clock enable. Set `RAMArchitecture` to `'WithoutClockEnable'` if your synthesis tool does not support RAM structures with a clock enable, and cannot map your generated HDL code to FPGA RAM resources. To learn how to generate RAMs without clock enables for your design, see the Getting Started with RAM and ROM example. To open the example, type the following command at the MATLAB prompt:

```
hdlcoderramrom
```

RAM Inference Limitations

If you use RAM blocks to perform concurrent read and write operations, you should verify the read-during-write behavior in hardware. The read-during-write behavior of the RAM blocks in Simulink matches that of the generated behavioral HDL code. However, a synthesis tool may not follow the same behavior during RAM inference, causing the read-during-write behavior in hardware to differ from the behavior of the Simulink model or generated HDL code.

In addition, your synthesis tool may not map the generated code to RAM for the following reasons:

- Small RAM size: your synthesis tool may use registers to implement a small RAM for better performance.
- A clock enable signal is present. You can suppress generation of a clock enable signal in RAM blocks, as described in “Implement RAM With or Without Clock Enable” on page 2-196.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Dual Rate Dual Port RAM

Purpose

Dual Rate Dual Port RAM implementations, properties, and restrictions for HDL code generation

Description

The Dual Rate Dual Port RAM block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Dual Rate Dual Port RAM.

HDL Implementations

This block has a single default HDL architecture.

HDL code generated for RAM blocks has:

- A latency of 1 clock cycle for read data output.
- No reset signal, since some synthesis tools do not infer a RAM from HDL code if it includes a reset.

Code generation for a RAM block creates a separate file, `*blockname.ext*`, where `*blockname*` is derived from the name of the RAM block, and `*ext*` is the target language filename extension.

RAM Initialization

Code generated to initialize a RAM is intended for simulation only, and may be ignored by synthesis tools.

Implement RAM With or Without Clock Enable

The HDL block property, `RAMArchitecture` , enables or suppresses generation of clock enable logic for all RAM blocks in a subsystem. You can set `RAMArchitecture` to the following values:

- `WithClockEnable` (default): Generates RAMs using HDL templates that include a clock enable signal, and an empty RAM wrapper.
- `WithoutClockEnable`: Generates RAMs without clock enables, and a RAM wrapper that implements the clock enable logic.

Some synthesis tools may not infer RAMs with a clock enable. Set `RAMArchitecture` to `WithoutClockEnable` if your synthesis tool does not support RAM structures with a clock enable, and cannot map your generated HDL code to FPGA RAM resources.

RAM Inference Limitations

If you use RAM blocks to perform concurrent read and write operations, you should verify the read-during-write behavior in hardware. The read-during-write behavior of the RAM blocks in Simulink matches that of the generated behavioral HDL code. However, a synthesis tool may not follow the same behavior during RAM inference, causing the read-during-write behavior in hardware to differ from the behavior of the Simulink model or generated HDL code.

In addition, your synthesis tool may not map the generated code to RAM for the following reasons:

- Small RAM size: your synthesis tool may use registers to implement a small RAM for better performance.
- A clock enable signal is present. You can suppress generation of a clock enable signal in RAM blocks, as described in “Implement RAM With or Without Clock Enable” on page 2-196.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Enable implementations, properties, and restrictions for HDL code generation

Description

The Enable block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Enable.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

See Also

Enabled Subsystem

Purpose

Enabled Subsystem implementations, properties, and restrictions for HDL code generation

Description

An enabled subsystem is a subsystem that receives a control signal via an Enable block. The enabled subsystem executes at each simulation step where the control signal has a positive value.

For detailed information on how to construct and configure enabled subsystems, see “Create an Enabled Subsystem” in the Simulink documentation.

Best Practices

It is good practice to consider the following when using enabled subsystems in models targeted for HDL code generation:

- For synthesis results to match Simulink results, the Enable port should be driven by registered logic (with a synchronous clock) on the FPGA.
- It is good practice to put unit delays on Enabled Subsystem output signals. This will prevent the code generator from inserting extra bypass registers in the HDL code.
- The use of enabled subsystems can affect synthesis results in the following ways:
  - In some cases the system clock speed may drop by a small percentage.
  - Generated code will use more resources, scaling with the number of enabled subsystem instances and the number of output ports per subsystem.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

The coder supports HDL code generation for enabled subsystems that meet the following conditions:

- The DUT (i.e., the top-level subsystem for which code is generated) must not be an enabled subsystem.
- The coder does not support subsystems that are *both* triggered *and* enabled for HDL code generation.
- The enable signal must be a scalar.
- The data type of the enable signal must be either `boolean` or `ufix1`.
- Outputs of the enabled subsystem must have an initial value of 0.
- All inputs and outputs of the enabled subsystem (including the enable signal) must run at the same rate.
- The **Show output port** parameter of the Enable block must be set to `Off`.
- The **States when enabling** parameter of the Enable block must be set to `held` (i.e., the Enable block does not reset states when enabled).
- The **Output when disabled** parameter for the enabled subsystem output port(s) must be set to `held` (i.e., the enabled subsystem does not reset output values when disabled).
- If the DUT contains the following blocks, `RAMArchitecture` must be set to `WithClockEnable`:
  - Dual Port RAM
  - Simple Dual Port RAM
  - Single Port RAM
- The following blocks are not supported in enabled subsystems targeted for HDL code generation:
  - CIC Decimation
  - CIC Interpolation
  - FIR Decimation

- FIR Interpolation
- Downsample
- Upsample
- HDL Cosimulation blocks for HDL Verifier™
- Rate Transition
- FFT
- HDL Streaming FFT

Example

The Automatic Gain Controller example shows how you can use enabled subsystems in HDL code generation. To open the example, enter:

```
hdlcoder_agc
```

See Also

Enable | Subsystem

Enumerated Constant

Purpose

Enumerated Constant implementations, properties, and restrictions for HDL code generation

Description

The Enumerated Constant block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Enumerated Constant.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Error Rate Calculation implementations, properties, and restrictions for HDL code generation

Description

The Error Rate Calculation block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Error Rate Calculation.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Extract Bits

Purpose

Extract Bits implementations, properties, and restrictions for HDL code generation

Description

The Extract Bits block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Extract Bits.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

FFT HDL Optimized implementations, properties, and restrictions for HDL code generation

Description

The FFT HDL Optimized block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see FFT HDL Optimized.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

FIR Decimation

Purpose

FIR Decimation implementations, properties, and restrictions for HDL code generation

Description

The FIR Decimation block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see FIR Decimation.

The coder supports both **Coefficient source** options (**Dialog parameters** or **Multirate filter object (MFILT)**).

When you select **Multirate filter object (MFILT)**, you can enter either a filter object name or a direct filter specification in the **Multirate filter variable** field.

HDL Implementations

Observe the following limitations for FIR Decimation filters:

- The coder supports `SerialPartition` only for the FIR Direct Form structure.
- Accumulator reuse is not supported.

Distributed Arithmetic Support

Distributed Arithmetic properties **DALUTPartition** and **DARadix** are supported for the following filter structures.

Implementation	Supported FIR Structures
default	mfilt.firdecim

AddPipelineRegisters Support

When you use **AddPipelineRegisters**, registers are placed based on filter implementation. The pipeline register placement determines the latency.

Implementation	Pipeline Register Placement	Latency (clock cycles)
FIR Decimation	One pipeline register is added between levels of a tree-based adder, and one is added after the products.	Where `NZ` is the number of non-zero coefficients: `ceil(log2(NZ))`

HDL Filter Properties

For HDL filter property descriptions, see “HDL Filter Block Properties”.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

- **Initial conditions** must be set to zero. HDL code generation is not supported for nonzero initial states.
- Vector and frame inputs are not supported for HDL code generation.
- When you select **Multirate filter object (MFILT)**, the filter object specified in the **Multirate filter variable** field must be either a `mfilt.firdecim` object or a `mfilt.firtdecim` object. If you specify some other type of filter object, an error will occur.
- When you select **Dialog parameters**, the following fixed-point options are not supported for HDL code generation:
  - `Slope and Bias scaling`
  - `Inherit via internal rule`
- **CoeffMultipliers** options are supported only when using a fully parallel architecture. The **CoeffMultipliers** property is hidden from the HDL Block Properties dialog box when you select a serial architecture.

FIR Interpolation

Purpose

FIR Interpolation implementations, properties, and restrictions for HDL code generation

Description

The FIR Interpolation block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see FIR Interpolation.

The coder supports both **Coefficient source** options (**Dialog parameters** or **Multirate filter object (MFILT)**). When you select **Multirate filter object (MFILT)**, you can enter either a filter object name or a direct filter specification in the **Multirate filter variable** field.

HDL Implementations

The `SerialPartition` property is set automatically for you on the FIR Interpolation Block when you select `Fully Serial` architecture.

Distributed Arithmetic Support

Distributed Arithmetic properties **DALUTPartition** and **DARadix** are supported for the following filter structures.

Implementation	Supported FIR Structures
Fully Parallel (default) Partly Serial Fully Serial Distributed Arithmetic (DA)	mfilt.firinterp

AddPipelineRegisters Support

When you use **AddPipelineRegisters**, registers are placed based on filter implementation. The pipeline register placement determines the latency.

Implementation	Pipeline Register Placement	Latency (clock cycles)
FIR Interpolation	A pipeline register is added between levels of a tree-based adder.	Where `PL` is polyphse filter length: `ceil(log2(PL))-1`

HDL Filter Properties

For HDL filter property descriptions, see “HDL Filter Block Properties”.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

- **Initial conditions** must be set to zero. HDL code generation is not supported for nonzero initial states.
- Vector and frame inputs are not supported for HDL code generation.
- When you select **Multirate filter object (MFILT)**, the filter object specified in the **Multirate filter variable** field must be a `mfilt.firinterp` object. If you specify some other type of filter object, an error will occur.
- When you select **Dialog parameters**, the following fixed-point options are not supported for HDL code generation:
  - **Coefficients**: `Slope and Bias scaling`
  - **Product Output**: `Inherit via internal rule`
- **CoeffMultipliers** options are supported only when using a fully parallel architecture. The **CoeffMultipliers** property is hidden from the HDL Block Properties dialog box when you select a serial architecture.

Floating Scope

Purpose

Floating Scope implementations, properties, and restrictions for HDL code generation

Description

The Floating Scope block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Floating Scope.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Frame Conversion

Purpose

Frame Conversion implementations, properties, and restrictions for HDL code generation

Description

The Frame Conversion block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Frame Conversion.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

From

Purpose

From implementations, properties, and restrictions for HDL code generation

Description

The From block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see From.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose Gain implementations, properties, and restrictions for HDL code generation

Description The Gain block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Gain.

HDL Implementations

ConstMultiplierOptimization	Description
none(*Default*)	By default, the coder does not perform CSD or FCSD optimizations. Code generated for the Gain block retains multiplier operations.
csd	When you specify this option, the generated code decreases the area used by the model while maintaining or increasing clock speed, using canonic signed digit (CSD) techniques. CSD replaces multiplier operations with add and subtract operations. CSD minimizes the number of addition operations required for constant multiplication by representing binary numbers with a minimum count of nonzero digits.
fcsd	This option uses factored CSD (FCSD) techniques, which replace multiplier operations with shift and add/subtract operations on certain factors of the operands. These factors are generally prime but can also be a number close to a power of 2, which favors area reduction. FCSD lets you achieve a greater area reduction than CSD, at the cost of decreasing clock speed.
auto	When you specify this option, the coder chooses between the CSD or FCSD optimizations. The coder chooses the optimization that yields the most area-efficient implementation, based on the number of adders

ConstMultiplierOptimization	Description
	required. When you specify auto, the coder does not use multipliers, unless conditions are such that CSD or FCSD optimizations are not possible (for example, if the design uses floating-point arithmetic).

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

General CRC Generator HDL Optimized implementations, properties, and restrictions for HDL code generation

Description

The General CRC Generator HDL Optimized block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see General CRC Generator HDL Optimized.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

General CRC Syndrome Detector HDL Optimized

Purpose

General CRC Syndrome Detector HDL Optimized implementations, properties, and restrictions for HDL code generation

Description

The General CRC Syndrome Detector HDL Optimized block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see General CRC Syndrome Detector HDL Optimized.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

General Multiplexed Deinterleaver

Purpose

General Multiplexed Deinterleaver implementations, properties, and restrictions for HDL code generation

Description

The General Multiplexed Deinterleaver block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see General Multiplexed Deinterleaver.

HDL Implementations

The implementation for the General Multiplexed Deinterleaver block is shift register based. If you want to suppress generation of reset logic, set the implementation parameter `ResetType` to`none`.

Note that when you set `ResetType` to`none`, reset is not applied to the shift registers. Mismatches between Simulink and the generated code occur for some number of samples during the initial phase, when registers are not fully loaded. To avoid spurious test bench errors, determine the number of samples required to fully load the shift registers. Then, set the **Ignore output data checking (number of samples)** option accordingly. (You can use the `IgnoreDataChecking` property for this purpose, if you are using the command-line interface.)

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

General Multiplexed Interleaver

Purpose

General Multiplexed Interleaver implementations, properties, and restrictions for HDL code generation

Description

The General Multiplexed Interleaver block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see General Multiplexed Interleaver.

HDL Implementations

The implementation for the General Multiplexed Interleaver block is shift register based. If you want to suppress generation of reset logic, set the implementation parameter `ResetType` to`'none'`.

Note that when you set `ResetType` to`'none'`, reset is not applied to the shift registers. Mismatches between Simulink and the generated code occur for some number of samples during the initial phase, when registers are not fully loaded. To avoid spurious test bench errors, determine the number of samples required to fully load the shift registers. Then, set the **Ignore output data checking (number of samples)** option accordingly. (You can use the `IgnoreDataChecking` property for this purpose, if you are using the command-line interface.)

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Goto implementations, properties, and restrictions for HDL code generation

Description

The Goto block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Goto.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Ground implementations, properties, and restrictions for HDL code generation

Description

The Ground block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Ground.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

HDL Cosimulation

Purpose

HDL Cosimulation implementations, properties, and restrictions for HDL code generation

Description

The HDL Cosimulation block is available with HDL Verifier.

For information on the Simulink simulation behavior and block parameters, see HDL Cosimulation.

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

- HDL Verifier for use with Mentor Graphics® ModelSim®
- HDL Verifier for use with Cadence Incisive®

Each of the HDL Cosimulation blocks cosimulates a hardware component by applying input signals to, and reading output signals from, an HDL model that executes under an HDL simulator.

See the "Define HDL Cosimulation Block Interface" section of the HDL Verifier documentation for information on timing, latency, data typing, frame-based processing, and other issues that may be of concern to you when setting up an HDL cosimulation.

You can use an HDL Cosimulation block with the coder to generate an interface to your manually written or legacy HDL code. When an HDL Cosimulation block is included in a model, the coder generates a VHDL or Verilog interface, depending on the selected target language.

When the target language is VHDL, the generated interface includes:

- An entity definition. The entity defines ports (input, output, and clock) corresponding in name and data type to the ports configured on the HDL Cosimulation block. Clock enable and reset ports are also declared.
- An RTL architecture including a component declaration, a component configuration declaring signals corresponding to signals connected to the HDL Cosimulation ports, and a component instantiation.
- Port assignment statements as required by the model.

When the target language is Verilog, the generated interface includes:

- A module defining ports (input, output, and clock) corresponding in name and data type to the ports configured on the HDL Cosimulation block. The module also defines clock enable and reset ports, and `wire` declarations corresponding to signals connected to the HDL Cosimulation ports.
- A module instance.
- Port assignment statements as required by the model.

The requirements for using the HDL Cosimulation block for code generation are the same as those for cosimulation. If you want to check these conditions before initiating code generation, select **Simulation > Update Diagram**.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Concepts

- "Generate a Cosimulation Model"

Purpose

HDL Counter implementations, properties, and restrictions for HDL code generation

Description

The HDL Counter block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see HDL Counter.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

HDL FIFO

Purpose

HDL FIFO implementations, properties, and restrictions for HDL code generation

Description

The HDL FIFO block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see HDL FIFO.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

IFFT HDL Optimized implementations, properties, and restrictions for HDL code generation

Description

The IFFT HDL Optimized block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see IFFT HDL Optimized.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Increment Real World implementations, properties, and restrictions for HDL code generation

Description

The Increment Real World block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Increment Real World.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Increment Stored Integer implementations, properties, and restrictions for HDL code generation

Description

The Increment Stored Integer block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Increment Stored Integer.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Index Vector

Purpose

Index Vector implementations, properties, and restrictions for HDL code generation

Description

The Index Vector block is a Multiport Switch block with **Number of data ports** set to 1. For HDL code generation information, see Multiport Switch.

Purpose

Inport implementations, properties, and restrictions for HDL code generation

Description

The Inport block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Inport.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

BidirectionalPort

BidirectionalPort Setting	Description
`on`	Specify the port as bidirectional. The following requirements apply: • The port must be in a Subsystem block with black box implementation. • There must also be no logic between the bidirectional port and the corresponding top-level DUT subsystem port. For more information, see “Specify Bidirectional Ports”.
`off` (default)	Do not specify the port as bidirectional.

Integer-Input RS Encoder HDL Optimized

Purpose

Integer-Input RS Encoder HDL Optimized implementations, properties, and restrictions for HDL code generation

Description

The Integer-Input RS Encoder HDL Optimized block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Integer-Input RS Encoder HDL Optimized.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Integer-Output RS Decoder HDL Optimized implementations, properties, and restrictions for HDL code generation

Description

The Integer-Output RS Decoder HDL Optimized block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Integer-Output RS Decoder HDL Optimized.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

LMS Filter

Purpose

LMS Filter implementations, properties, and restrictions for HDL code generation

Description

The LMS Filter block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see LMS Filter.

HDL Implementations

By default, the LMS Filter implementation uses a linear sum for the FIR section of the filter.

The LMS Filter implements a tree summation (which has a shorter critical path) under the following conditions:

- The LMS Filter is used with real data
- The word length of the Accumulator **W'u** data type is at least `ceil(log2(filter length))` bits wider than the word length of the Product W'u data type
- The Accumulator **W'u** data type has the same fraction length as the Product **W'u** data type

HDL Filter Properties

For HDL filter property descriptions, see "HDL Filter Block Properties".

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Complex Data Support

This block supports code generation for complex signals.

Restrictions

- The coder does not support the `Normalized LMS` algorithm of the LMS Filter.
- The `Reset` port supports only `Boolean` and `unsigned` inputs.

- The `Adapt` port supports only `Boolean` inputs.
- **Filter length** must be greater than or equal to 2.

Purpose

Logical Operator implementations, properties, and restrictions for HDL code generation

Description

The Logical Operator block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Logical Operator.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

M-PSK Demodulator Baseband implementations, properties, and restrictions for HDL code generation

Description

The M-PSK Demodulator Baseband block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see M-PSK Demodulator Baseband.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

M-PSK Modulator Baseband

Purpose

M-PSK Modulator Baseband implementations, properties, and restrictions for HDL code generation

Description

The M-PSK Modulator Baseband block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see M-PSK Modulator Baseband.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Magnitude-Angle to Complex implementations, properties, and restrictions for HDL code generation

Description

The Magnitude-Angle to Complex block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Magnitude-Angle to Complex.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Restrictions

The Magnitude-Angle to Complex block supports HDL code generation when you set **Approximation method** to `CORDIC`.

Math Function

Purpose

Math Function implementations, properties, and restrictions for HDL code generation

Description

The Math Function block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Math Function.

HDL Implementations

conj

Implementations	Description
`ComplexConjugate`	Compute complex conjugate. See Math Function in the Simulink documentation.

hermitian

Implementations	Description
`Hermitian`	Compute hermitian. See Math Function in the Simulink documentation.

reciprocal

Implementations	Parameters	Description
`Math` (default) `Reciprocal`	None	Compute reciprocal as `1/N`, using the HDL divide (`/`) operator to implement the division.
`RecipNewton`	`Iterations`	Use the iterative Newton method. Select this option to optimize area. The default value for `Iterations` is 3. The recommended value for `Iterations` is between 2 and 10. The coder generates a message

Implementations	Parameters	Description
		if `Iterations` is outside the recommended range.
`RecipNewtonSingleRate`	`Iterations`	Use the single rate pipelined Newton method. Select this option to optimize speed, or if you want a single rate implementation. The default value for `Iterations` is 3. The recommended value for `Iterations` is between 2 and 10. The coder generates a message if `Iterations` is outside the recommended range.

transpose

Implementations	Description
`Transpose`	Compute array transpose. See Math Function in the Simulink documentation.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Complex Data Support

The `conj`, `hermitian`, and `transpose` functions support complex data.

Restrictions

When you use a `reciprocal` implementation:

- Input must be scalar and must have integer or fixed-point (signed or unsigned) data type.

- The output must be scalar and have integer or fixed-point (signed or unsigned) data type.
- Only the `Zero` rounding mode is supported.
- The **Saturate on integer overflow** option on the block must be selected.

Purpose

MATLAB Function implementations, properties, and restrictions for HDL code generation

Description

The MATLAB Function block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see MATLAB Function.

Best Practices

- “Design Guidelines for the MATLAB Function Block”
- “Generate Instantiable Code for Functions”
- “Optimize Loops in the MATLAB Function Block”
- “Pipeline Variables in the MATLAB Function Block”

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

See also “Complex Data Type Support”.

Restrictions

- If the block contains a System object™, block inputs cannot have non-discrete (constant or `Inf`) sample time.

For the MATLAB language subset supported for HDL code generation from a MATLAB Function block, see:

- “Data Types and Scope”
- “Operators”
- “Control Flow Statements”
- “Persistent Variables”

- “Persistent Array Variables”
- “System Objects”
- “Complex Data Type Support”
- “Fixed-Point Bitwise Functions”
- “Fixed-Point Run-Time Library Functions”

Related Examples

- “Code Generation from a MATLAB Function Block”
- “MATLAB Function Block Design Patterns for HDL”
- “Distributed Pipeline Insertion for MATLAB Function Blocks”

Concepts

- “HDL Applications for the MATLAB Function Block”

Purpose

MATLAB System implementations, properties, and restrictions for HDL code generation

Description

You can define a System object and use it in a MATLAB System block for HDL code generation.

The MATLAB System block is available with Simulink.

For information on the Simulink behavior and block parameters, see MATLAB System.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Restrictions

- The DUT subsystem must be single-rate.
- Inputs cannot have non-discrete (constant or `Inf`) sample time.
- Predefined System objects are not supported. For example, System objects that are available with DSP System Toolbox or Communications System Toolbox, such as `hdlram`, are not supported.
- The System object must be a user-defined System object that supports HDL code generation. For information about user-defined System objects and requirements for HDL code generation, see "System Objects".

Related Examples

- "Generate Code for User-Defined System Objects"

Concepts

- "System Objects"

Matrix Concatenate

Purpose

Matrix Concatenate implementations, properties, and restrictions for HDL code generation

Description

The Matrix Concatenate block is the same as the Vector Concatenate block with **Mode** set to `Multidimensional array`. For HDL code generation information, see Vector Concatenate.

Purpose

Matrix Viewer implementations, properties, and restrictions for HDL code generation

Description

The Matrix Viewer block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Matrix Viewer.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Maximum

Purpose

Maximum implementations, properties, and restrictions for HDL code generation

Description

The Maximum block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Maximum.

HDL Implementations

Implementations	Description
`default` `Tree`	The `Tree` implementation is large and slow but has minimal latency.
`Cascade`	This implementation is optimized for latency * area, with medium speed. See “Cascade Implementation Best Practices”.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Memory implementations, properties, and restrictions for HDL code generation

Description

The Memory block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Memory.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Minimum

Purpose

Minimum implementations, properties, and restrictions for HDL code generation

Description

The Minimum block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Minimum.

HDL Implementations

Implementations	Description
`default` `Tree`	The `Tree` implementation is large and slow but has minimal latency.
`Cascade`	This implementation is optimized for latency * area, with medium speed. See “Cascade Implementation Best Practices”.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

MinMax implementations, properties, and restrictions for HDL code generation

Description

The MinMax block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see MinMax.

HDL Implementations

Implementations	Description
`default` `Tree`	The `Tree` implementation is large and slow but has minimal latency.
`Cascade`	This implementation is optimized for latency * area, with medium speed. See “Cascade Implementation Best Practices”.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Model

Purpose

Model implementations, properties, and restrictions for HDL code generation

Description

The Model block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Model.

HDL Implementations

Implementations	Description
`ModelReference` (default)	Use the `ModelReference` implementation when you want to generate code from a referenced model and any nested models. For more information, see “How To Generate Code for a Referenced Model”.
`BlackBox`	Use the `BlackBox` implementation to instantiate an HDL wrapper, or black box interface, for legacy or external HDL code. If you specify a black box interface, the coder does not attempt to generate HDL code for the referenced model. For more information, see “Generate Black Box Interface for Referenced Model”.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

When you generate HDL code for referenced models, the following limitations apply:

- Block parameters for the Model block must be set to their default values.

- If multiple model references refer to the same model, their HDL block properties must be the same.
- Referenced models cannot be protected models.
- Hierarchical distributed pipelining must be disabled.

The coder cannot move registers across a model reference. Therefore, referenced models may inhibit the following optimizations:

- Distributed pipelining
- Constrained output pipelining

The coder cannot apply the streaming optimization to a model reference.

The coder can apply the resource sharing optimization to share referenced model instances. However, this optimization can be applied only when all model references that point to the same referenced model have the same rate after optimizations and rate propagation. The model reference final rate may differ from the original rate, but all model references that point to the same referenced model must have the same final rate.

Concepts

- "Model Referencing for HDL Code Generation"
- "Generate Black Box Interface for Referenced Model"

Model Info

Purpose

Model Info implementations, properties, and restrictions for HDL code generation

Description

The Model Info block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Model Info.

HDL Implementations

This block has a single default HDL architecture.

Purpose

Model Variants implementations, properties, and restrictions for HDL code generation

Description

The Model Variants block is a version of the Model block. For HDL code generation information, see Model.

Purpose

Multiport Selector implementations, properties, and restrictions for HDL code generation

Description

The Multiport Selector block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Multiport Selector.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Multiport Switch implementations, properties, and restrictions for HDL code generation

Description

The Multiport Switch block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Multiport Switch.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

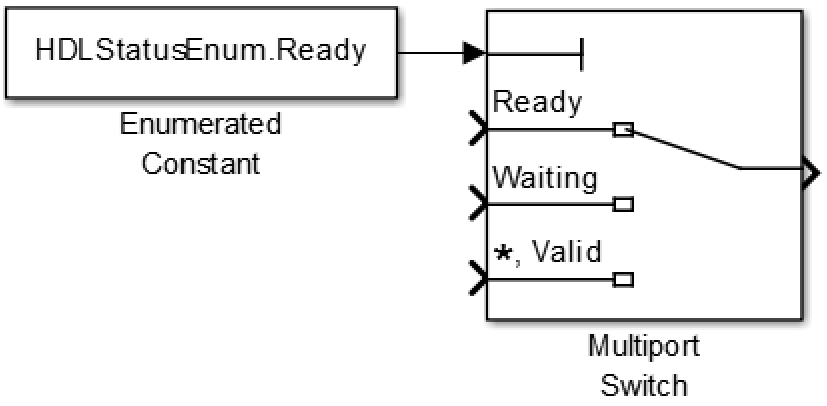
For HDL block property descriptions, see "HDL Block Properties".

Complex Data Support

This block supports code generation for complex signals.

Example

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


HDLStatusEnum.Ready
Enumerated Constant
Ready
Waiting
*, Valid
Multiport Switch

Purpose

Mux implementations, properties, and restrictions for HDL code generation

Description

The Mux block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Mux.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Restrictions

Buses are not supported for HDL code generation.

Purpose

n-D Lookup Table implementations, properties, and restrictions for HDL code generation

Description

The n-D Lookup Table block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see n-D Lookup Table.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Complex Data Support

This block supports code generation for complex signals.

Restrictions

- "Required Block Settings" on page 2-160
- "Avoid Generation of Divide Operator" on page 2-161
- "Table Data Typing and Sizing" on page 2-161

Required Block Settings

- **Number of table dimensions**: The coder supports a maximum dimension of 2.
- **Index search method**: Select `Evenly spaced points`.
- **Extrapolation method**: The coder supports only `Clip`. The coder does not support extrapolation beyond the table bounds.
- **Interpolation method**: The coder supports only `Flat` or `Linear`.
- **Diagnostic for out-of-range input**: Select `Error`. If you select other options, the coder displays a warning.

- **Use last table value for inputs at or above last breakpoint**: Select this check box.
- **Require all inputs to have the same data type**: Select this check box.
- **Fraction**: Select `Inherit:  Inherit via internal rule`.
- **Integer rounding mode**: Select `Zero`, `Floor`, or `Simplest`.

Avoid Generation of Divide Operator

The coder gives a warning if it encounters conditions under which a division operation is required to match the model's simulation behavior. The conditions described in this section will cause this block to emit a divide operator. When you use this block for HDL code generation, you should avoid the following conditions:

- If the block is configured to use interpolation, a division operator will be required. To avoid this, set **Interpolation method** : to `Flat`.
- The second way depends on the table spacing. HDL code generation requires the block to use the "Evenly Spaced Points" algorithm. The block mapping from the input data type to the zero-based table index in general requires a division. When the breakpoint spacing is an exact power of 2, this divide is implemented as a shift instead of as a divide. To adjust the breakpoint spacing, you can adjust the number of breakpoints in the table and/or the difference between the left and right bounds of the breakpoint range.

Table Data Typing and Sizing

- It is good practice to structure your table such that the spacing between breakpoints is a power of two. The coder issues a warning if the breakpoint spacing does not meet this condition. When the breakpoint spacing is a power of two, you can replace division operations in the prelookup step with right-shift operations.
- Table data must resolve to a nonfloating-point data type.
- All ports on the block require scalar values.

Purpose

NCO implementations, properties, and restrictions for HDL code generation

Description

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

Purpose

NCO HDL Optimized implementations, properties, and restrictions for HDL code generation

Description

The NCO HDL Optimized block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see NCO HDL Optimized.

HDL Implementations

This block has a single default HDL architecture.

HDL Filter Properties

For HDL filter property descriptions, see “HDL Filter Block Properties”.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Outport

Purpose

Outport implementations, properties, and restrictions for HDL code generation

Description

The Outport block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Outport.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

BidirectionalPort

BidirectionalPort Setting	Description
`on`	Specify the port as bidirectional. The following requirements apply: • The port must be in a Subsystem block with black box implementation. • There must also be no logic between the bidirectional port and the corresponding top-level DUT subsystem port. For more information, see "Specify Bidirectional Ports".
`off` (default)	Do not specify the port as bidirectional.

Purpose

PN Sequence Generator implementations, properties, and restrictions for HDL code generation

Description

The PN Sequence Generator block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see PN Sequence Generator.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

Inputs:

- You can select `Input port` as the **Output mask source** on the block. However, in this case the `Mask` input signal must be a vector of data type `ufix1`.
- If **Reset on nonzero input** is selected, the input to the `Rst` port must have data type `Boolean`.

Outputs:

- Outputs of type `double` are not supported for HDL code generation. All other output types (including bit packed outputs) are supported.

Prelookup

Purpose

Prelookup implementations, properties, and restrictions for HDL code generation

Description

The Prelookup block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Prelookup.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Restrictions

- "Required Block Settings" on page 2-166
- "Table Data Typing and Sizing" on page 2-166

Required Block Settings

- **Breakpoint data**: For **Source**, select `Dialog`.
- **Index search method**: Select `Evenly spaced points`.
- **Extrapolation method**: Select `Clip`.
- **Diagnostic for out-of-range input**: Select `Error`.
- **Use last breakpoint for input at or above upper limit**: Select this check box.
- **Breakpoint**: For **Data Type**, select `Inherit:  Same as input`.
- **Integer rounding mode**: Select `Zero`, `Floor`, or `Simplest`.

Table Data Typing and Sizing

- It is good practice to structure your table such that the spacing between breakpoints is a power of two. The coder issues a warning if the breakpoint spacing does not meet this condition. When the breakpoint spacing is a power of two, you can replace division operations in the prelookup step with right-shift operations.

- All ports on the block require scalar values.
- The coder permits floating-point data for breakpoints.

Product

Purpose

Product implementations, properties, and restrictions for HDL code generation

Description

The Product block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Product.

HDL Implementations

Implementations	Description
`Linear` (default)	Generates a chain of N operations (multipliers) for N inputs.
`Tree`	This implementation has minimal latency but is large and slow. It generates a tree-shaped structure of multipliers. **Note**: Product blocks that have a vector input with two or more elements support Tree and Cascade.
`Cascade`	This implementation optimizes latency * area and is faster than the `Tree` implementation. It computes partial products and cascades multipliers. **Note**: Product blocks that have a vector input with two or more elements support Tree and Cascade. See “Cascade Implementation Best Practices”.

Divide Mode

For block implementations of the Product block in divide mode, see Divide.

Note The Product block is in divide mode when the **Number of Inputs** is set to `*/`.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

The default (linear) implementation supports complex data.

Complex division is not supported.

Product of Elements

Purpose

Product of Elements implementations, properties, and restrictions for HDL code generation

Description

The Product of Elements block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Product of Elements.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

The default (linear) implementation supports complex data.

Complex division is not supported.

Purpose

QPSK Demodulator Baseband implementations, properties, and restrictions for HDL code generation

Description

The QPSK Demodulator Baseband block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see QPSK Demodulator Baseband.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

QPSK Modulator Baseband

Purpose

QPSK Modulator Baseband implementations, properties, and restrictions for HDL code generation

Description

The QPSK Modulator Baseband block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see QPSK Modulator Baseband.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Purpose

Rate Transition implementations, properties, and restrictions for HDL code generation

Description

The Rate Transition block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Rate Transition.

Best Practices

It is good practice to follow the Rate Transition block with a unit delay. This will prevent the code generator from inserting an extra bypass register in the HDL code.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Real-Imag to Complex

Purpose

Real-Imag to Complex implementations, properties, and restrictions for HDL code generation

Description

The Real-Imag to Complex block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Real-Imag to Complex.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Reciprocal implementations, properties, and restrictions for HDL code generation

Description

The Reciprocal block is the Divide block with **Number of Inputs** is set to `/`. For HDL code generation information, see Divide.

Reciprocal Sqrt

Purpose

Reciprocal Sqrt implementations, properties, and restrictions for HDL code generation

Description

The Reciprocal Sqrt block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Reciprocal Sqrt.

HDL Implementations

Implementations	Description
`SqrtFunction` (default) `RecipSqrtNewton`	Use the iterative Newton method. Select this option to optimize area.
`RecipSqrtNewtonSingleRate`	Use the single rate pipelined Newton method. Select this option to optimize speed, or if you want a single rate implementation.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Restrictions

- In the Block Parameters dialog box, in the **Algorithm** tab, for **Method**, you must select `Newton-Raphson`.
- Input must be an unsigned scalar value.
- Output is a fixed-point scalar value.

Rectangular QAM Demodulator Baseband

Purpose

Rectangular QAM Demodulator Baseband implementations, properties, and restrictions for HDL code generation

Description

The Rectangular QAM Demodulator Baseband block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Rectangular QAM Demodulator Baseband.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

- The block does not support single or double data types for HDL code generation.
- The coder supports the following **Output type** options:
  - `Integer`
  - `Bit` is supported only if the **Decision Type** selected is `Hard decision`.
- The coder requires that **Normalization Method** be set to `Minimum Distance Between Symbols`, with a **Minimum distance** of 2.
- The coder requires that **Phase offset (rad)** be set to a value that is multiple a of `pi`/4.

Rectangular QAM Modulator Baseband

Purpose

Rectangular QAM Modulator Baseband implementations, properties, and restrictions for HDL code generation

Description

The Rectangular QAM Modulator Baseband block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Rectangular QAM Modulator Baseband.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Restrictions

- The block does not support single or double data types for HDL code generation.
- When **Input Type** is set to `Bit`, the block does not support HDL code generation for input types other than `boolean` or `ufix1`.

The Rectangular QAM Modulator Baseband block does not support HDL code generation when the input type is set to `Bit` but the block input is actually multibit (`uint16`, for example).

Purpose Relational Operator implementations, properties, and restrictions for HDL code generation

Description The Relational Operator block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Relational Operator.

HDL Implementations This block has a single default HDL architecture.

HDL Block Properties For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support The ~= and == operators are supported for code generation.

Purpose

Relay implementations, properties, and restrictions for HDL code generation

Description

The Relay block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Relay.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Repeat implementations, properties, and restrictions for HDL code generation

Description

The Repeat block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Repeat.

Best Practices

The Repeat block uses fewer hardware resources than the Upsample block. Use the Repeat block if your algorithm does not require zero-padding upsampling.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Purpose

Reshape implementations, properties, and restrictions for HDL code generation

Description

The Reshape block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Reshape.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Saturation implementations, properties, and restrictions for HDL code generation

Description

The Saturation block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Saturation.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Purpose

Saturation Dynamic implementations, properties, and restrictions for HDL code generation

Description

The Saturation Dynamic block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Saturation Dynamic.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Scope implementations, properties, and restrictions for HDL code generation

Description

The Scope block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Scope.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

Selector implementations, properties, and restrictions for HDL code generation

Description

The Selector block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Selector.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Shift Arithmetic implementations, properties, and restrictions for HDL code generation

Description

The Shift Arithmetic block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Shift Arithmetic.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Restrictions

In the Function Block Parameters dialog box, for **Bits to shift**, you must set **Source** to **Dialog**. The **Input port** option is not supported for HDL code generation.

Sign

Purpose Sign implementations, properties, and restrictions for HDL code generation

Description The Sign block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Sign.

HDL Implementations This block has a single default HDL architecture.

HDL Block Properties For HDL block property descriptions, see “HDL Block Properties”.

Signal Conversion

Purpose

Signal Conversion implementations, properties, and restrictions for HDL code generation

Description

The Signal Conversion block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Signal Conversion.

HDL Implementations

This block has a pass-through implementation.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Signal Specification

Purpose

Signal Specification implementations, properties, and restrictions for HDL code generation

Description

The Signal Specification block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Signal Specification.

HDL Implementations

This block has a pass-through implementation.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Signal To Workspace

Purpose

Signal To Workspace implementations, properties, and restrictions for HDL code generation

Description

The Signal To Workspace block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Signal To Workspace.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Simple Dual Port RAM

Purpose

Simple Dual Port RAM implementations, properties, and restrictions for HDL code generation

Description

The Simple Dual Port RAM block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Simple Dual Port RAM.

HDL Implementations

This block has a single default HDL architecture.

HDL code generated for RAM blocks has:

- A latency of 1 clock cycle for read data output.
- No reset signal, since some synthesis tools do not infer a RAM from HDL code if it includes a reset.

Code generation for a RAM block creates a separate file, `*blockname.ext*`, where `*blockname*` is derived from the name of the RAM block, and `*ext*` is the target language filename extension.

RAM Initialization

Code generated to initialize a RAM is intended for simulation only, and may be ignored by synthesis tools.

Implement RAM With or Without Clock Enable

The HDL block property, `RAMArchitecture` , enables or suppresses generation of clock enable logic for all RAM blocks in a subsystem. You can set `RAMArchitecture` to the following values:

- `WithClockEnable` (default): Generates RAMs using HDL templates that include a clock enable signal, and an empty RAM wrapper.
- `WithoutClockEnable`: Generates RAMs without clock enables, and a RAM wrapper that implements the clock enable logic.

Some synthesis tools may not infer RAMs with a clock enable. Set `RAMArchitecture` to `WithoutClockEnable` if your synthesis tool does not support RAM structures with a clock enable, and cannot map your generated HDL code to FPGA RAM resources. To learn how to generate RAMs without clock enables for your design, see the Getting Started with RAM and ROM example. To open the example, type the following command at the MATLAB prompt:

```
hdlcoderramrom
```

RAM Inference Limitations

If you use RAM blocks to perform concurrent read and write operations, you should verify the read-during-write behavior in hardware. The read-during-write behavior of the RAM blocks in Simulink matches that of the generated behavioral HDL code. However, a synthesis tool may not follow the same behavior during RAM inference, causing the read-during-write behavior in hardware to differ from the behavior of the Simulink model or generated HDL code.

In addition, your synthesis tool may not map the generated code to RAM for the following reasons:

- Small RAM size: your synthesis tool may use registers to implement a small RAM for better performance.
- A clock enable signal is present. You can suppress generation of a clock enable signal in RAM blocks, as described in “Implement RAM With or Without Clock Enable” on page 2-196.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Sine implementations, properties, and restrictions for HDL code generation

Description

The Sine block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Sine, Cosine.

HDL Implementations

The HDL code implements Sine using the quarter-wave lookup table you specify in the Simulink block parameters.

To avoid generating a division operator (`/`) in the HDL code, for **Number of data points for lookup table**, enter `(2^`*`n`*`)+1`, where *n* is an integer.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

This block does not have restrictions for HDL code generation.

If you see the following warnings for the Sine or Cosine block, you can ignore them:

-
  ```
  HDL code generation for the Lookup Table (n-D) block does
  not support out-of-range inputs.  Set the "Diagnostic
  for out of range input" block parameter to "Error" to
  suppress this warning.
  ```

-
  ```
  Using linear interpolation on the Lookup Table (n-D)
  block, may require using a divide operator in the
  generated HDL, which may not be synthesizable.
  ```

Purpose

Sine Wave implementations, properties, and restrictions for HDL code generation

Description

The Sine Wave block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Sine Wave.

HDL Implementations

This block has a single default HDL architecture.

Restrictions

For HDL code generation, you must select the following Sine Wave block settings:

- **Computation method**: `Table lookup`
- **Sample mode**: `Discrete`

Output:

- The output port cannot have data types `single` or `double`.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Single Port RAM implementations, properties, and restrictions for HDL code generation

Description

The Single Port RAM block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Single Port RAM.

HDL Implementations

This block has a single default HDL architecture.

HDL code generated for RAM blocks has:

- A latency of 1 clock cycle for read data output.
- No reset signal, since some synthesis tools do not infer a RAM from HDL code if it includes a reset.

Code generation for a RAM block creates a separate file, `*blockname.ext*`, where `*blockname*` is derived from the name of the RAM block, and `*ext*` is the target language filename extension.

RAM Initialization

Code generated to initialize a RAM is intended for simulation only, and may be ignored by synthesis tools.

Implement RAM With or Without Clock Enable

The HDL block property, `RAMArchitecture` , enables or suppresses generation of clock enable logic for all RAM blocks in a subsystem. You can set `RAMArchitecture` to the following values:

- `WithClockEnable` (default): Generates RAMs using HDL templates that include a clock enable signal, and an empty RAM wrapper.
- `WithoutClockEnable`: Generates RAMs without clock enables, and a RAM wrapper that implements the clock enable logic.

Some synthesis tools may not infer RAMs with a clock enable. Set `RAMArchitecture` to `WithoutClockEnable` if your synthesis tool does not support RAM structures with a clock enable, and cannot map your generated HDL code to FPGA RAM resources. To learn how to generate RAMs without clock enables for your design, see the Getting Started with RAM and ROM example. To open the example, type the following command at the MATLAB prompt:

```
hdlcoderramrom
```

RAM Inference Limitations

Depending on your synthesis tool and target device, the setting of **Output data during write** may affect the result of RAM inference.

If you use RAM blocks to perform concurrent read and write operations, you should verify the read-during-write behavior in hardware. The read-during-write behavior of the RAM blocks in Simulink matches that of the generated behavioral HDL code. However, a synthesis tool may not follow the same behavior during RAM inference, causing the read-during-write behavior in hardware to differ from the behavior of the Simulink model or generated HDL code.

In addition, your synthesis tool may not map the generated code to RAM for the following reasons:

- Small RAM size: your synthesis tool may use registers to implement a small RAM for better performance.
- A clock enable signal is present. You can suppress generation of a clock enable signal in RAM blocks, as described in "Implement RAM With or Without Clock Enable" on page 2-196.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Complex Data Support

This block supports code generation for complex signals.

Spectrum Analyzer

Purpose

Spectrum Analyzer implementations, properties, and restrictions for HDL code generation

Description

The Spectrum Analyzer block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Spectrum Analyzer.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

Sqrt implementations, properties, and restrictions for HDL code generation

Description

The Sqrt block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Sqrt.

HDL Implementations

Implementation	Parameter	Description	
`SqrtFunction` (default) `SqrtBitset`	`UseMultiplier`	`on`	Use a multiply/add algorithm (Simulink default algorithm).
		`off`	Use a bitset shift/addition algorithm.
`SqrtNewton`	`Iterations`	Use the iterative Newton method. Select this option to optimize area. The default value for `Iterations` is 3. The recommended value for `Iterations` is between 2 and 10. The coder generates a message if `Iterations` is outside the recommended range.	
`SqrtNewtonSingleRate`	`Iterations`	Use the single rate pipelined Newton method. Select this option to optimize speed, or if you want a single rate implementation. The default value for `Iterations` is 3. The recommended value for `Iterations` is between 2 and 10. The coder generates a message if `Iterations` is outside the recommended range.	
`SqrtTargetLibrary`	None	Use the Altera or Xilinx target library.	

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

- Input must be an unsigned scalar value.
- Output is a fixed-point scalar value.

Purpose State Transition Table implementations, properties, and restrictions for HDL code generation

Description The State Transition Table block is available with Stateflow.

For information on the Simulink simulation behavior and block parameters, see State Transition Table.

HDL Implementations This block has a single default HDL architecture.

HDL Block Properties For HDL block property descriptions, see “HDL Block Properties”.

See Also Chart | Truth Table

Stop Simulation

Purpose

Stop Simulation implementations, properties, and restrictions for HDL code generation

Description

The Stop Simulation block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Stop Simulation.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

Subsystem implementations, properties, and restrictions for HDL code generation

Description

The Subsystem block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Subsystem.

HDL Implementations

Implementation	Description
`Module` (default)	Generate code for the subsystem and the blocks within the subsystem.
`AlteraBlackBox`	Specify an Altera DSP Builder Subsystem.
`BlackBox`	Generate a black-box interface. That is, the generated HDL code includes only the input/output port definitions for the subsystem. In this way, you can use a subsystem in your model to generate an interface to existing manually written HDL code. The black-box interface generated for subsystems is similar to the interface generated for Model blocks, but without generation of clock signals.
`No HDL`	Remove the subsystem from the generated code. You can use the subsystem in simulation but treat it as a "no-op" in the HDL code.
`XilinxBlackBox`	Specify a Xilinx System Generator Subsystem.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Concepts

- "External Component Interfaces"
- "Generate Black Box Interface for Subsystem"
- "Create a Xilinx System Generator Subsystem"
- "Create an Altera DSP Builder Subsystem"

Purpose

Subtract implementations, properties, and restrictions for HDL code generation

Description

The Subtract block is a Sum block with **List of signs** set to `+-`. For HDL code generation information, see Sum.

For information on the Simulink simulation behavior and block parameters, see Subtract.

Purpose

Sum implementations, properties, and restrictions for HDL code generation

Description

The Sum block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Sum.

HDL Implementation

Implementations	Description
`default` `Linear`	Generates a chain of N operations (adders) for N inputs.
`Tree`	This implementation has minimal latency but is large and slow. Generates a tree-shaped structure of adders.
`Cascade`	This implementation optimizes latency * area and is faster than the `Tree` implementation. It computes partial sums and cascades adders. See "Cascade Implementation Best Practices".

The coder supports Tree and Cascade implementations for Sum blocks that have a single vector input with multiple elements.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Complex Data Support

Only the default (linear) implementation supports complex data.

Sum of Elements

Purpose

Sum of Elements implementations, properties, and restrictions for HDL code generation

Description

The Sum of Elements block is a Sum block with more than two inputs. For HDL code generation information, see Sum.

For information on the Simulink simulation behavior and block parameters, see Sum of Elements.

Purpose

Switch implementations, properties, and restrictions for HDL code generation

Description

The Switch block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Switch.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Tapped Delay

Purpose

Tapped Delay implementations, properties, and restrictions for HDL code generation

Description

The Tapped Delay block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Tapped Delay.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Terminator implementations, properties, and restrictions for HDL code generation

Description

The Terminator block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Terminator.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Time Scope

Purpose

Time Scope implementations, properties, and restrictions for HDL code generation

Description

The Time Scope block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Time Scope.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

To File implementations, properties, and restrictions for HDL code generation

Description

The To File block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see To File.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

To VCD File implementations, properties, and restrictions for HDL code generation

Description

The To VCD File block is available with HDL Verifier.

For information on the Simulink simulation behavior and block parameters, see To VCD File.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

To Workspace implementations, properties, and restrictions for HDL code generation

Description

The To Workspace block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see To Workspace.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Trigger

Purpose

Trigger implementations, properties, and restrictions for HDL code generation

Description

The Trigger block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Trigger.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

See Also

Triggered Subsystem

Purpose

Triggered Subsystem implementations, properties, and restrictions for HDL code generation

Description

A triggered subsystem is a subsystem that receives a control signal via a Trigger block. The triggered subsystem executes for one cycle each time a trigger event occurs. For detailed information on how to define trigger events and configure triggered subsystems, see "Create a Triggered Subsystem" in the Simulink documentation.

Best Practices

Consider the following when using triggered subsystems in models targeted for HDL code generation:

- For synthesis results to match Simulink results, the trigger port should be driven by registered logic (with a synchronous clock) on the FPGA.
- It is good practice to put unit delays on Triggered Subsystem output signals. This will prevent the code generator from inserting extra bypass registers in the HDL code.
- The use of triggered subsystems can affect synthesis results in the following ways:
  - In some cases the system clock speed may drop by a small percentage.
  - Generated code will use more resources, scaling with the number of triggered subsystem instances and the number of output ports per subsystem.

Using the Signal Builder Block

When you connect outputs from a Signal Builder block to a triggered subsystem, you may need to use a Rate Transition block. To run all triggered subsystem ports at the same rate:

- If the trigger source is a Signal Builder block, but the other triggered subsystem inputs come from other sources, insert a Rate Transition block into the signal path before the trigger input.

- If all inputs (including the trigger) come from a Signal Builder block, they have the same rate, so special action is not required.

Using the Trigger As Clock

You can generate code that uses the trigger signal as a clock using the `TriggerAsClock` property. See "Use Trigger As Clock in Triggered Subsystems".

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Restrictions

The coder supports HDL code generation for triggered subsystems that meet the following conditions:

- The DUT (that is, the top-level subsystem for which code is generated) must not be a triggered subsystem.
- The coder does not support subsystems that are *both* triggered *and* enabled for HDL code generation.
- The trigger signal must be a scalar.
- The data type of the trigger signal must be either `boolean` or `ufix1`.
- Outputs of the triggered subsystem must have an initial value of 0.
- All inputs and outputs of the triggered subsystem (including the trigger signal) must run at the same rate. (See "Using the Signal Builder Block" on page 2-215 for information on a special case.)
- The **Show output port** parameter of the Trigger block must be set to `Off`.
- If the DUT contains the following blocks, `RAMArchitecture` must be set to `WithClockEnable`:
  - Dual Port RAM

  - Simple Dual Port RAM
  - Single Port RAM
- The following blocks are not supported in triggered subsystems targeted for HDL code generation:
  - Discrete-Time Integrator
  - CIC Decimation
  - CIC Interpolation
  - FIR Decimation
  - FIR Interpolation
  - Downsample
  - Upsample
  - HDL Cosimulation blocks for HDL Verifier
  - Rate Transition
  - FFT
  - HDL Streaming FFT

See Also

Trigger | Subsystem

Trigonometric Function

Purpose

Trigonometric Function implementations, properties, and restrictions for HDL code generation

Description

The Trigonometric Function block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Trigonometric Function.

HDL Implementations

You must use the default HDL implementation, `Trigonometric`.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Restrictions

The Trigonometric Function block supports HDL code generation for the following functions:

Trigonometric Function Block Implementation	Supported Functions	Supported Approximation Methods
`default` `Trigonometric`	`sin`	`CORDIC`
	`cos`	`CORDIC`
	`cos + jsin`	`CORDIC`
	`sincos`	`CORDIC`

For the `sin` and `cos` functions, unsigned data types are supported for CORDIC approximations.

The coder gives an error when:

- You select an unsupported function on the Trigonometric Function block.
- You select an **Approximation method** other than `CORDIC`.

See Also

`cordicsin` | `cordiccos` | `cordicsincos`

Purpose

Truth Table implementations, properties, and restrictions for HDL code generation

Description

The Truth Table block is available with Stateflow.

For information on the Simulink simulation behavior and block parameters, see Truth Table.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

See Also

Chart | Truth Table

Unary Minus

Purpose

Unary Minus implementations, properties, and restrictions for HDL code generation

Description

The Unary Minus block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Unary Minus.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Unit Delay implementations, properties, and restrictions for HDL code generation

Description

The Unit Delay block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Unit Delay.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Unit Delay Enabled implementations, properties, and restrictions for HDL code generation

Description

The Unit Delay Enabled block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Unit Delay Enabled.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Unit Delay Enabled Resettable implementations, properties, and restrictions for HDL code generation

Description

The Unit Delay Enabled Resettable block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Unit Delay Enabled Resettable.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Unit Delay Resettable

Purpose

Unit Delay Resettable implementations, properties, and restrictions for HDL code generation

Description

The Unit Delay Resettable block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Unit Delay Resettable.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Purpose

Upsample implementations, properties, and restrictions for HDL code generation

Description

The Upsample block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Upsample.

Best Practices

Consider whether your model can use the Repeat block instead of Upsample. The Repeat block uses fewer hardware resources, so it is a best practice to only use Upsample when your algorithm requires zero-padding upsampling.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Variable Selector

Purpose Variable Selector implementations, properties, and restrictions for HDL code generation

Description The Variable Selector block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Variable Selector.

HDL Implementations This block has a single default HDL architecture.

HDL Block Properties For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support This block supports code generation for complex signals.

Purpose

Variant Subsystem implementations, properties, and restrictions for HDL code generation

Description

The Variant Subsystem block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Variant Subsystem.

HDL Implementations

The coder generates code for only the active variant.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Vector Concatenate

Purpose

Vector Concatenate implementations, properties, and restrictions for HDL code generation

Description

The Vector Concatenate block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Vector Concatenate.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose

Vector Scope implementations, properties, and restrictions for HDL code generation

Description

The Vector Scope block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Vector Scope.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Purpose

Viterbi Decoder implementations, properties, and restrictions for HDL code generation

Description

The Viterbi Decoder block is available with Communications System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Viterbi Decoder.

The coder currently supports the following features of the Viterbi Decoder block:

- Non-recursive encoder/decoder with feed-forward trellis and simple shift register generation configuration
- Sample based input
- Decoder rates from 1/2 to 1/7
- Constraint length from 3 to 9

Pipelining the Traceback Unit

The Viterbi Decoder block decodes every bit by tracing back through a traceback depth that you define for the block. The block implements a complete traceback for each decision bit, using registers to store the minimum state index and branch decision in the traceback decoding unit. You can specify that the traceback decoding unit be pipelined in order to improve the speed of the generated circuit. You can add pipeline registers to the traceback unit by specifying the number of traceback stages per pipeline register. To do this, use the `TracebackStagesPerPipeline` implementation parameter.

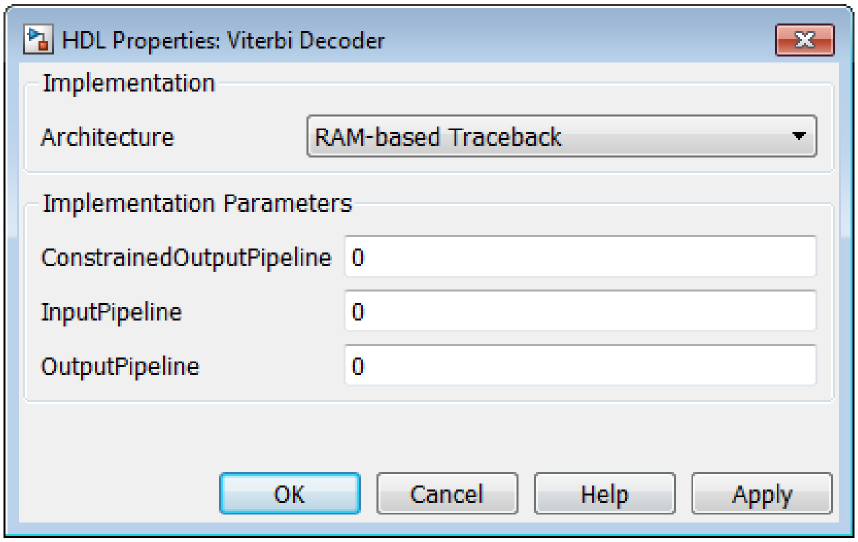
The `TracebackStagesPerPipeline` implementation parameter lets you balance the circuit performance based on system requirements. A smaller parameter value indicates the requirement to add more registers to increase the speed of the traceback circuit. Increasing the number results in a lower number of registers along with a decrease in the circuit speed.

See the "HDL Code Generation for Viterbi Decoder" example model for an example using `TracebackStagesPerPipeline`.

RAM-Based Traceback

Instead of using registers, you can choose to use RAMs to save the survivor branch information.

1. Set the HDL Architecture property of the Viterbi Decoder block to RAM-based Traceback.


HDL Properties: Viterbi Decoder

Implementation

Architecture: RAM-based Traceback

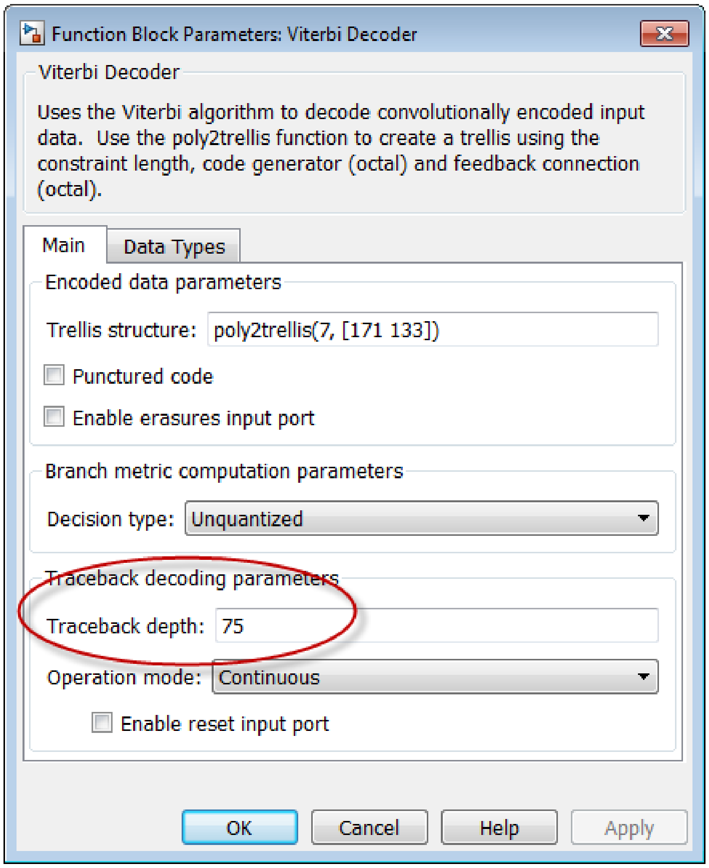
Implementation Parameters

ConstrainedOutputPipeline: 0

InputPipeline: 0

OutputPipeline: 0

2. Set the traceback depth on the Viterbi Decoder block mask.


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: 75

Operation mode: Continuous

☐ Enable reset input port

RAM-based traceback and register-based traceback differ in the following ways:

- The RAM-based implementation traces back through one set of data to find the initial state to decode the previous set of data. The register-based implementation combines the traceback and decode operations into one step and uses the best state found from the minimum operation as the decoding initial state.

- RAM-based implementation traces back through M samples, decodes the previous M bits in reverse order, and releases one bit in order at each clock cycle, whereas the register-based implementation decodes one bit after a complete traceback.

Because of the differences in the two traceback algorithms, the RAM-based implementation produces different numerical results than the register-based traceback. A longer traceback depth, for example, 10 times the constraint length, is recommended in the RAM-based traceback to achieve a similar bit error rate (BER) as the register-based implementation. The size of RAM required for the implementation depends on the trellis and the traceback depth.

See HDL Code Generation for Viterbi Decoder.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see "HDL Block Properties".

Restrictions

- **Punctured code**: Do not select this option. Punctured code requires frame-based input, which the coder does not support.
- **Decision type**: the coder does not support the `Unquantized` decision type.
- **Error if quantized input values are out of range**: The coder does not support this option.
- **Operation mode**: The coder supports only the `Continuous` mode.
- **Enable reset input port**: HDL support is provided when you enable both **Enable reset input port** and **Delay reset action to next time step**. You must select `Continuous` operation mode.

Input and Output Data Types

- When **Decision type** is set to `Soft decision`, the HDL implementation of the Viterbi Decoder block supports fixed-point inputs and output. For input, the fixed-point data type must be `ufixN`, where `N` is the number of soft decision bits. Signed built-in data types (`int8`, `int16`, `int32`) are not supported. For output, the HDL implementation of the Viterbi Decoder block supports block-supported output data types.
- When **Decision type** is set to `Hard decision`, the block supports input with data types `ufix1` and `Boolean`. For output, the HDL implementation of the Viterbi Decoder block supports block-supported output data types.
- The HDL implementation of the Viterbi Decoder block does not support double and single input data types are not supported. The block does not support floating point output for fixed-point inputs.

Example

The "HDL Code Generation for Viterbi Decoder" example demonstrates HDL code generation for a fixed-point Viterbi Decoder block, with pipelined traceback decoding. To open the example, type the following command:

```
showdemo commviterbihdl_m
```

Purpose

Waterfall implementations, properties, and restrictions for HDL code generation

Description

The Waterfall block is available with DSP System Toolbox.

For information on the Simulink simulation behavior and block parameters, see Waterfall.

HDL Implementations

The coder does not generate HDL code for this block when you use it in your model.

Zero-Order Hold

Purpose

Zero-Order Hold implementations, properties, and restrictions for HDL code generation

Description

The Zero-Order Hold block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see Zero-Order Hold.

HDL Implementations

This block has a single default HDL architecture.

HDL Block Properties

For HDL block property descriptions, see “HDL Block Properties”.

Complex Data Support

This block supports code generation for complex signals.

Purpose XY Graph implementations, properties, and restrictions for HDL code generation

Description The XY Graph block is available with Simulink.

For information on the Simulink simulation behavior and block parameters, see XY Graph.

HDL Implementations The coder does not generate HDL code for this block when you use it in your model.

Properties — Alphabetical List

BalanceDelays property

Purpose

Set delay balancing for the model

Settings

`'on'` (default)

Enable delay balancing for the model.

`'off'`

Disable delay balancing for the model.

Usage Notes

You can further control delay balancing within the model by disabling or enabling delay balancing for subsystems within the model.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

- "Delay Balancing"
- "BalanceDelays"

Purpose

Specify string to append to block labels used for HDL `GENERATE` statements

Settings

`'string'`

Default: `'_gen'`

Specify a postfix string to append to block labels used for HDL `GENERATE` statements.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`InstanceGenerateLabel`, `OutputGenerateLabel`

CheckHDL property

Purpose

Check model or subsystem for HDL code generation compatibility

Settings

`'on'`

Selected

Check the model or subsystem for HDL compatibility before generating code, and report problems encountered. This is equivalent to executing the `checkhdl` function before calling `makehdl`.

`'off'` (default)

Cleared (default)

Do not check the model or subsystem for HDL compatibility before generating code.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`checkhdl`, `makehdl`

Purpose

Specify active clock edge

Settings

`'Rising'` (default)

The rising clock edge triggers Verilog `always` or VHDL `process` blocks in the generated code.

`'Falling'`

The falling clock edge triggers Verilog `always` or VHDL `process` blocks in the generated code.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`ResetAssertedLevel`, `ClockInputPort`, `InputType`, `OutputType`, `ResetInputPort`

ClockEnableInputPort property

Purpose

Name HDL port for model's clock enable input signals

Settings

'string'

Default: 'clk_enable'

The string specifies the name for the model's clock enable input port.

If you override the default with (for example) the string 'filter_clock_enable' for the generating subsystem filter_subsys, the generated entity declaration might look as follows:

```
ENTITY filter_subsys IS
   PORT( clk                 :  IN  std_logic;
         filter_clock_enable :  IN  std_logic;
         reset               :  IN  std_logic;
         filter_subsys_in    :  IN  std_logic_vector (15 DOWNTO 0);
         filter_subsys_out   :  OUT std_logic_vector (15 DOWNTO 0);
         );
END filter_subsys;
```

If you specify a string that is a VHDL or Verilog reserved word, the code generator appends a reserved word postfix string to form a valid VHDL or Verilog identifier. For example, if you specify the reserved word signal, the resulting name string would be signal_rsvd. See ReservedWordPostfix for more information.

Usage Notes

The clock enable signal is asserted active high (1). Thus, the input value must be high for the generated entity's registers to be updated.

Set or View This Property

To set this property, use hdlset_param or makehdl. To view the property value, use hdlget_param.

See Also

ClockInputPort, InputType, OutputType, ResetInputPort

Purpose

Specify name of clock enable output port

Settings

`'string'`

Default: `'ce_out'`

The string specifies the name for the generated clock enable output port.

A clock enable output is generated when the design requires one.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

ClockHighTime property

Purpose

Specify period, in nanoseconds, during which test bench drives clock input signals high (1)

Settings

`ns`

Default: 5

The clock high time is expressed as a positive integer.

The `ClockHighTime` and `ClockLowTime` properties define the period and duty cycle for the clock signal. Using the defaults, the clock signal is a square wave (50% duty cycle) with a period of 10 ns.

Usage Notes

The coder ignores this property if `ForceClock` is set to `off`.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`ClockLowTime`, `ForceClock`, `ForceClockEnable`, `ForceReset`, `HoldTime`

Purpose

Specify generation of single or multiple clock inputs

Settings

`'Single'` (Default)

Generates a single clock input for the DUT. If the DUT is multirate, the input clock is the master clock rate, and a timing controller is synthesized to generate additional clocks as required.

`'Multiple'`

Generates a unique clock for each Simulink rate in the DUT. The number of timing controllers generated depends on the contents of the DUT.

Usage Notes

The oversample factor must be 1 (default) to specify multiple clocks.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Example

The following example specifies the generation of multiple clocks.

```
makehdl(gcb, 'ClockInputs','Multiple');
```

ClockInputPort property

Purpose

Name HDL port for model's clock input signals

Settings

'string'

Default: `'clk'`.

The string specifies the clock input port name.

If you override the default with (for example) the string `'filter_clock'` for the generated entity `my_filter`, the generated entity declaration might look as follows:

```
ENTITY my_filter IS
   PORT( filter_clock   :  IN  std_logic;
         clk_enable     :  IN  std_logic;
         reset          :  IN  std_logic;
         my_filter_in   :  IN  std_logic_vector (15 DOWNTO 0); -- sfix16_En15
         my_filter_out  :  OUT std_logic_vector (15 DOWNTO 0); -- sfix16_En15
         );
END my_filter;
```

If you specify a string that is a VHDL or Verilog reserved word, the code generator appends a reserved word postfix string to form a valid VHDL or Verilog identifier. For example, if you specify the reserved word `signal`, the resulting name string would be `signal_rsvd`. See `ReservedWordPostfix` for more information.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`ClockEnableInputPort`, `InputType`, `OutputType`

Purpose

Specify period, in nanoseconds, during which test bench drives clock input signals low (0)

Settings

Default: 5

The clock low time is expressed as a positive integer.

The `ClockHighTime` and `ClockLowTime` properties define the period and duty cycle for the clock signal. Using the defaults, the clock signal is a square wave (50% duty cycle) with a period of 10 ns.

Usage Notes

The coder ignores this property if `ForceClock` is set to `off`.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`ClockHighTime`, `ForceClock`, `ForceClockEnable`, `ForceReset`, `HoldTime`

ClockProcessPostfix property

Purpose

Specify string to append to HDL clock process names

Settings

`'string'`

Default: `'_process'`.

The coder uses process blocks for register operations. The label for each of these blocks is derived from a register name and the postfix `_process`. For example, the coder derives the label `delay_pipeline_process` in the following block declaration from the register name `delay_pipeline` and the default postfix string `_process`:

```
delay_pipeline_process : PROCESS (clk, reset)
BEGIN
   .
   .
   .
```

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`PackagePostfix`, `ReservedWordPostfix`

Purpose

Control production of generated code and display of generated model

Settings

`'string'`

Default: `'GenerateHDLCode'`

Generate code but do not display the generated model.

`'GenerateHDLCodeAndDisplayGeneratedModel'`

Generate both code and model, and display model when completed.

`'DisplayGeneratedModelOnly'`

Create and display generated model, but do not proceed to code generation.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Defaults and Options for Generated Models"

ComplexImagPostfix property

Purpose Specify string to append to imaginary part of complex signal names

Settings `'`*`string`*`'`

Default: `'_im'`.

Set or View This Property To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also `ComplexRealPostfix`

Purpose Specify string to append to real part of complex signal names

Settings `'string'`

Default: `'_re'`.

Set or View This Property To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also `ComplexImagPostfix`

Purpose

Specify whether to include time/date information in the generated HDL file header

Settings

'on' (default)

Include time/date information in the generated HDL file header.

```
-- -----------------------------------------------------------
--
-- File Name: hdlsrc\symmetric_fir.vhd
-- Created: 2011-02-14 07:21:36
--
```

'off'

Omit time/date information in the generated HDL file header.

```
-- -----------------------------------------------------------
--
-- File Name: hdlsrc\symmetric_fir.vhd
--
```

By omitting the time/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.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Specify priority for distributed pipelining algorithm

Settings

`'NumericalIntegrity'` (default)

Prioritize numerical integrity when distributing pipeline registers.

This option uses a conservative retiming algorithm that does not move registers across a component if the functional equivalence to the original design is unknown.

`'Performance'`

Prioritize performance over numerical integrity.

Use this option if your design requires a higher clock frequency and the Simulink behavior does not need to strictly match the generated code behavior.

This option uses a more aggressive retiming algorithm that moves registers across a component even if the modified design's functional equivalence to the original design is unknown.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

EDAScriptGeneration property

Purpose Enable or disable generation of script files for third-party tools

Settings `'on'` (default)

Enable generation of script files.

`'off'`

Disable generation of script files.

Set or View This Property To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also "Generate Scripts for Compilation, Simulation, and Synthesis"

Purpose

Specify base name string for internal clock enables in generated code

Settings

'*string*'

Default: 'enb'

Specify the string used as the base name for internal clock enables and other flow control signals in generated code.

Usage Notes

Where only a single clock enable is generated, EnablePrefix specifies the signal name for the internal clock enable signal.

In some cases multiple clock enables are generated (for example, when a cascade block implementation for certain blocks is specified). In such cases, EnablePrefix specifies a base signal name for the first clock enable that is generated. For other clock enable signals, numeric tags are appended to EnablePrefix to form unique signal names. For example, the following code fragment illustrates two clock enables that were generated when EnablePrefix was set to 'test_clk_enable' :

```
COMPONENT mysys_tc
    PORT( clk                   :   IN    std_logic;
          reset                 :   IN    std_logic;
          clk_enable            :   IN    std_logic;
          test_clk_enable       :   OUT   std_logic;
          test_clk_enable_5_1_0 :   OUT   std_logic
          );
  END COMPONENT;
```

Set or View This Property

To set this property, use hdlset_param or makehdl. To view the property value, use hdlget_param.

EntityConflictPostfix property

Purpose

Specify string to append to duplicate VHDL entity or Verilog module names

Settings

`'string'`

Default: `'_block'`

The specified postfix resolves duplicate VHDL entity or Verilog module names.

For example, if the coder detects two entities with the name `MyFilter`, the coder names the first entity `MyFilter` and the second entity `MyFilter_block`.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`PackagePostfix`, `ReservedWordPostfix`

Purpose

Specify whether test bench forces clock input signals

Settings

`'on'` (default)

Selected (default)

Specify that the test bench forces the clock input signals. When this option is set, the clock high and low time settings control the clock waveform.

`'off'`

Cleared

Specify that a user-defined external source forces the clock input signals.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`ClockLowTime`, `ClockHighTime`, `ForceClockEnable`, `ForceReset`, `HoldTime`

Purpose

Specify whether test bench forces clock enable input signals

Settings

`'on'` (default)

Selected (default)

Specify that the test bench forces the clock enable input signals to active high (1) or active low (0), depending on the setting of the clock enable input value.

`'off'`

Cleared

Specify that a user-defined external source forces the clock enable input signals.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`ClockHighTime`, `ClockLowTime`, `ForceClock`, `HoldTime`

Purpose

Specify whether test bench forces reset input signals

Settings

`'on'` (default)

Selected (default)

Specify that the test bench forces the reset input signals. If you enable this option, you can also specify a hold time to control the timing of a reset.

`'off'`

Cleared

Specify that a user-defined external source forces the reset input signals.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`ClockHighTime`, `ClockLowTime`, `ForceClock`, `HoldTime`

GenerateCoSimBlock property

Purpose

Generate HDL Cosimulation blocks for use in testing DUT

Settings

`'on'`

If your installation includes one or more of the following HDL simulation features, the coder generates an HDL Cosimulation block for each:

- HDL Verifier for use with Mentor Graphics ModelSim
- HDL Verifier for use with Cadence Incisive

The coder configures the generated HDL Cosimulation blocks to conform to the port and data type interface of the DUT selected for code generation. By connecting an HDL Cosimulation block to your model in place of the DUT, you can cosimulate your design with the desired simulator.

The coder appends the string specified by the `CosimLibPostfix` property to the names of the generated HDL Cosimulation blocks.

`'off'` (default)

Do not generate HDL Cosimulation blocks.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Generate model containing HDL Cosimulation block for use in testing DUT

Settings

`'ModelSim'` (default)

If your installation includes HDL Verifier for use with Mentor Graphics ModelSim, the coder generates and opens a Simulink model that contains an HDL Cosimulation block for Mentor Graphics ModelSim.

`'Incisive'`

If your installation includes HDL Verifier for use with Cadence Incisive, the coder generates and opens a Simulink model that contains an HDL Cosimulation block for Cadence Incisive.

`'None'`

Do not create a cosimulation model.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Generate a Cosimulation Model"

GeneratedModelName property

Purpose

Specify name of generated model

Settings

'string'

By default, the name of a generated model is the same as that of the original model. Assign a string value to `GeneratedModelName` to override the default.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Defaults and Options for Generated Models"

Purpose

Specify prefix to name of generated model

Settings

`'string'`

Default: `'gm_'`

The specified string is prepended to the name of the generated model.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Defaults and Options for Generated Models"

GenerateHDLCode property

Purpose

Generate HDL code

Settings

`'on'` (default)

Generate HDL code.

`'off'`

Do not generate HDL code.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Generate HDL Code Using the Configuration Parameters Dialog Box"

GenerateValidationModel property

Purpose

Generate validation model with HDL code

Settings

`'on'`

Generate a validation model that highlights generated delays and other differences between your original model and the generated model. With a validation model, you can observe the effects of streaming, resource sharing, and delay balancing.

`'off'` (default)

Do not generate a validation model.

Usage Notes

If you enable generation of a validation model, also enable delay balancing to keep the generated DUT model synchronized with the original DUT model. Mismatches between delays in the original DUT model and delays in the generated DUT model cause validation to fail.

You can set this property using `hdlset_param` or `makehdl`.

You can also generate a validation model by selecting one of the following check boxes:

- **Generate validation model** in the **HDL Code Generation** pane of the Configuration Parameters dialog box
- **Generate validation model** in the **Generate RTL Code and Testbench** task of the HDL Workflow Advisor

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

- “Delay Balancing”
- `BalanceDelays`

GenerateWebview property

Purpose

Include model Web view in the code generation report

Settings

`'on'`

Include model Web view in the code generation report.

`'off'` (default)

Omit model Web view in the code generation report.

Usage Notes

With a model Web view, you can click a link in the generated code to highlight the corresponding block in the model.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Web View of Model in Code Generation Report"

Purpose

Enable reusable code generation for identical atomic subsystems

Settings

`'on'` (default)

Generate reusable code for identical atomic subsystems.

`'off'`

Do not generate reusable code for identical atomic subsystems.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Generate Reusable Code for Atomic Subsystems"

Purpose

Generate HDL code that follows the specified coding standard.

Settings

`'None'` (default)

Generate generic synthesizable HDL code.

`'Industry'`

Generate HDL code that follows the industry standard rules supported by the coder. When this option is enabled, the coder generates a standard compliance report.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose Specify string written to initialization section of compilation script

Settings `'string'`

Default: `'vlib work\n'`.

Set or View This Property To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also "Generate Scripts for Compilation, Simulation, and Synthesis"

HDLCompileTerm property

Purpose Specify string written to termination section of compilation script

Settings `'string'`

The default is the null string (`''`).

Set or View This Property To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also "Generate Scripts for Compilation, Simulation, and Synthesis"

Purpose

Specify postfix string appended to file name for generated Mentor Graphics ModelSim compilation scripts

Settings

'string'

Default:'_compile.do'.

For example, if the name of the device under test or test bench is my_design, the coder adds the postfix _compile.do to form the name my_design_compile.do.

Set or View This Property

To set this property, use hdlset_param or makehdl. To view the property value, use hdlget_param.

HDLCompileVerilogCmd property

Purpose

Specify command string written to compilation script for Verilog files

Settings

`'string'`

Default: `'vlog %s %s\n'`.

The two arguments are the contents of the `SimulatorFlags` property and the file name of the current module. To omit the flags, set `SimulatorFlags` to `''` (the default).

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Generate Scripts for Compilation, Simulation, and Synthesis"

HDLCompileVHDLCmd property

Purpose

Specify command string written to compilation script for VHDL files

Settings

`'string'`

Default: `'vcom %s %s\n'`.

The two arguments are the contents of the `SimulatorFlags` property and the file name of the current entity. To omit the flags, set `SimulatorFlags` to `''` (the default).

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Generate Scripts for Compilation, Simulation, and Synthesis"

HDLControlFiles property

Purpose

Attach code generation control file to model

Settings

`{'string'}`

Pass in a cell array containing a string that specifies a control file to be attached to the current model. Defaults are

- File name extension: `.m`
- Location of file: the control file must be on the MATLAB path or in the current working folder. Therefore you need only specify the file name; do not specify path information.

  The following example specifies a control file, using the default for the file name extension.

  ```
  makehdl(gcb, 'HDLControlFiles', {'dct8config'});
  ```

  Specify a control file that is on the MATLAB path, or in the current working folder. You may need to modify the MATLAB path so that the desired control file is on the path before generating code. Then attach the control file to the model.

Note The current release supports specification of a single control file.

Usage Notes

To clear the property (so that control files are not invoked during code generation), pass in a cell array containing the null string, as in the following example:

```
makehdl(gcb,'HDLControlFiles',{''});
```

See Also

For a detailed description of the structure and use of control files, see "Code Generation Control Objects and Methods".

Purpose Specify postfix string appended to file name for generated mapping file

Settings `'string'`

Default: `'_map.txt'`.

For example, if the name of the device under test is `my_design`, the coder adds the postfix `_map.txt` to form the name `my_design_map.txt`.

Set or View This Property To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

HDLSimCmd property

Purpose

Specify simulation command written to simulation script

Settings

'*string*'

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

The implicit argument is the top-level module or entity name.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Generate Scripts for Compilation, Simulation, and Synthesis"

Purpose

Specify string written to initialization section of simulation script

Settings

'string'

The default string is

```
['onbreak resume\n',...
'onerror resume\n']
```

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

“Generate Scripts for Compilation, Simulation, and Synthesis”

HDLSimFilePostfix property

Purpose

Specify postfix string appended to file name for generated Mentor Graphics ModelSim simulation scripts

Settings

'*string*'

Default: `_sim.do`.

For example, if the name of your test bench file is `my_design`, the coder adds the postfix `_sim.do` to form the name `my_design_tb_sim.do`.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose Specify string written to termination section of simulation script

Settings `'string'`

Default: `'run -all\n'`.

Set or View This Property To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also "Generate Scripts for Compilation, Simulation, and Synthesis"

HDLSimViewWaveCmd property

Purpose Specify waveform viewing command written to simulation script

Settings `'string'`

Default: `'add wave sim:%s\n'`

The implicit argument adds the signal paths for the DUT top-level input, output, and output reference signals.

Set or View This Property To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also "Generate Scripts for Compilation, Simulation, and Synthesis"

Purpose

Specify command written to HDL lint script

Settings

'string'

Default: `''`

Specify the HDL lint tool command in the Tcl script. The command string must contain `%s`, which is a placeholder for the HDL file name.

Dependencies

If `HDLLintCmd` is set to the default value, `''`, and you set `HDLLintCmd` to one of the supported third-party tools, the coder automatically inserts a tool-specific default command in the Tcl script.

Usage

If you set `HDLLintTool` to `Custom`, you must use `%s` as a placeholder for the HDL file name in the generated Tcl script. Specify `HDLLintCmd` using the following format:

```
custom_lint_tool_command -option1 -option2 %s
```

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`HDLLintTool`, `HDLLintInit`, `HDLLintTerm`, “Generate an HDL Lint Tool Script”

Purpose

Specify HDL lint script initialization string

Settings

`'string'`

Default: `''`

Specify the HDL lint script initialization string.

Dependencies

If `HDLLintInit` is set to the default value, `''`, and you set `HDLLintCmd` to one of the supported third-party tools, the coder automatically inserts a tool-specific default initialization string in the Tcl script.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`HDLLintTool`, `HDLLintCmd`, `HDLLintTerm`, “Generate an HDL Lint Tool Script”

Purpose

Specify HDL lint script termination string

Settings

'*string*'

Default: ''

Specify the HDL lint script termination string.

Dependencies

If `HDLLintTerm` is set to the default value, `''`, and you set `HDLLintCmd` to one of the supported third-party tools, the coder automatically inserts a tool-specific default termination string in the Tcl script.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`HDLLintTool`, `HDLLintCmd`, `HDLLintInit`, "Generate an HDL Lint Tool Script"

HDLLintTool property

Purpose

Select HDL lint tool for which the coder generates scripts

Settings

`'string'`

Default: `'None'`.

`HDLLintTool` enables or disables generation of scripts for third-party HDL lint tools. By default, the coder does not generate a lint script.

To generate a script for one of the supported lint tools, set `HDLLintTool` to one of the following strings:

HDLLintTool Option	Lint Tool
`'None'`	None. Lint script generation is disabled.
`'AscentLint'`	Real Intent Ascent Lint
`'Leda'`	Synopsys® Leda
`'SpyGlass'`	Atrenta SpyGlass
`'Custom'`	A custom lint tool.

Dependencies

If you set `HDLLintTool` to one of the supported third-party tools, you can generate a Tcl script without setting `HDLLintInit`, `HDLLintCmd`, and `HDLLintTerm` to nondefault values. If the `HDLLintInit`, `HDLLintCmd`, and `HDLLintTerm` have default values, the coder automatically writes tool-specific default initialization, command, and termination strings to the Tcl script.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

Properties

`HDLLintCmdHDLLintInitHDLLintTerm`

Related Examples

- "Generate an HDL Lint Tool Script"

HDLSynthCmd property

Purpose

Specify command written to synthesis script

Settings

`'string'`

Default: none.

Your choice of synthesis tool (see `HDLSynthTool`) sets the synthesis command string. The default string is a format string passed to `fprintf` to write the command section of the synthesis script. The implicit argument is the top-level module or entity name. The content of the string is specific to the selected synthesis tool.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`HDLSynthTool`, `HDLSynthInit`, `HDLSynthTerm`, `HDLSynthFilePostfix`, "Generate Scripts for Compilation, Simulation, and Synthesis"

Purpose

Specify postfix string appended to file name for generated synthesis scripts

Settings

'string'

Default: The value of `HDLSynthFilePostfix` normally defaults to a string that corresponds to the synthesis tool that `HDLSynthTool` specifies.

For example, if the value of `HDLSynthTool` is `'Synplify'`, `HDLSynthFilePostfix` defaults to the string `'_synplify.tcl'`. Then, if the name of the device under test is `my_design`, the coder adds the postfix `_synplify.tcl` to form the synthesis script file name `my_design_synplify.tcl`.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`HDLSynthTool`, `HDLSynthCmd`, `HDLSynthInit`, `HDLSynthTerm`, "Generate Scripts for Compilation, Simulation, and Synthesis"

Purpose

Specify string written to initialization section of synthesis script

Settings

'string'

Default: none

Your choice of synthesis tool (see `HDLSynthTool`) sets the synthesis initialization string. The default string is a format string passed to `fprintf` to write the initialization section of the synthesis script. The default string is a synthesis project creation command. The implicit argument is the top-level module or entity name. The content of the string is specific to the selected synthesis tool.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`HDLSynthTool`, `HDLSynthCmd`, `HDLSynthTerm`, `HDLSynthFilePostfix`, "Generate Scripts for Compilation, Simulation, and Synthesis"

Purpose

Specify string written to termination section of synthesis script

Settings

`'string'`

Default: none

Your choice of synthesis tool (see `HDLSynthTool`) sets the synthesis termination string. The default string is a format string passed to `fprintf` to write the termination and clean up section of the synthesis script. This section does not take arguments. The content of the string is specific to the selected synthesis tool.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`HDLSynthTool`, `HDLSynthCmd`, `HDLSynthInit`, `HDLSynthFilePostfix`, "Generate Scripts for Compilation, Simulation, and Synthesis"

HDLSynthTool property

Purpose

Select synthesis tool for which the coder generates scripts

Settings

'*string*'

Default: 'None'.

HDLSynthTool enables or disables generation of scripts for third-party synthesis tools. By default, the coder does not generate a synthesis script. To generate a script for one of the supported synthesis tools, set HDLSynthTool to one of the following strings:

Tip The value of HDLSynthTool also sets the postfix string (HDLSynthFilePostfix) that the coder appends to generated synthesis script file names.

Choice of HDLSynthTool Value...	Generates Script For...	Sets HDLSynthFilePostfix To...
'None'	N/A; script generation disabled	N/A
'ISE'	Xilinx ISE	'_ise.tcl'
'Libero'	Microsemi Libero	'_libero.tcl'
'Precision'	Mentor Graphics Precision	'_precision.tcl'
'Quartus'	Altera Quartus II	'_quartus.tcl'
'Synplify'	Synopsys Synplify Pro®	'_synplify.tcl'
'Custom'	A custom synthesis tool	'_custom.tcl'

Set or View This Property

To set this property, use hdlset_param or makehdl. To view the property value, use hdlget_param.

See Also

`HDLSynthCmd`, `HDLSynthInit`, `HDLSynthTerm`, `HDLSynthFilePostfix`, “Generate Scripts for Compilation, Simulation, and Synthesis”

HierarchicalDistPipelining property

Purpose

Specify whether to apply retiming across a subsystem hierarchy

Settings

`'on'`

Enable retiming across a subsystem hierarchy. The coder applies retiming hierarchically down, until it reaches a subsystem where **DistributedPipelining** is `off`.

`'off'` (default)

Distribute pipelining only within a subsystem.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"DistributedPipelining"

Purpose

Highlight ancestors of blocks in generated model that differ from original model

Settings

`'on'` (default)

Highlight blocks in a generated model that differ from the original model, and their ancestor (parent) blocks in the model hierarchy. The `HighlightColor` property specifies the highlight color.

`'off'`

Highlight only the blocks in a generated model that differ from the original model without highlighting their ancestor (parent) blocks in the model hierarchy.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`HighlightColor`

Concepts

- “Defaults and Options for Generated Models”

Purpose

Specify color for highlighted blocks in generated model

Settings

`'string'`

Default: `'cyan'`.

Specify the color as one of the following color string values:

- `'cyan'`
- `'yellow'`
- `'magenta'`
- `'red'`
- `'green'`
- `'blue'`
- `'white'`
- `'black'`

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Defaults and Options for Generated Models"

Purpose

Specify how long subrate signal values are held in valid state

Settings

`'on'` (default)

Data values for subrate signals are held in a valid state across N base-rate clock cycles, where N is the number of base-rate clock cycles that elapse per subrate sample period and N >= 2.

`'off'`

Data values for subrate signals are held in a valid state for only one base-rate clock cycle. For the subsequent base-rate cycles, data is in an unknown state (expressed as `'X'`) until leading edge of the next subrate sample period.

Usage Notes

In most cases, the default (`'on'`) is the best setting for this property. This setting matches the behavior of a Simulink simulation, in which subrate signals are held valid through each base-rate clock period.

In some cases (for example modeling memory or memory interfaces), it is desirable to set `HoldInputDataBetweenSamples` to `'off'`. In this way, you can obtain diagnostic information about when data is in an invalid (`'X'`) state.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`HoldTime`, “Code Generation from Multirate Models”

Purpose

Specify hold time for input signals and forced reset input signals

Settings

`ns`

Default: 2

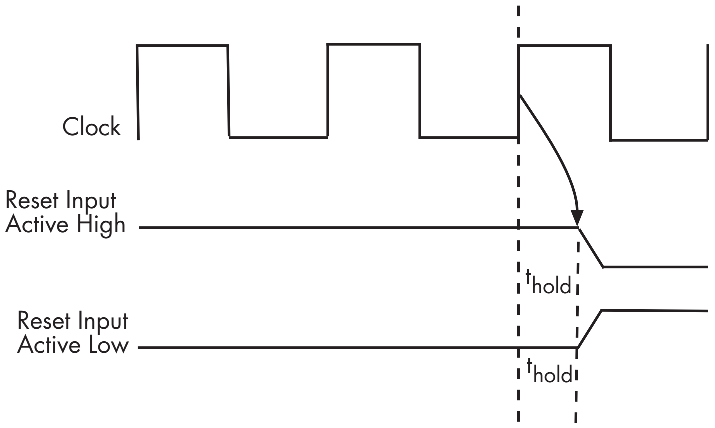
Specify the number of nanoseconds during which the model's data input signals and forced reset input signals are held past the clock rising edge.

The hold time is expressed as a positive integer.

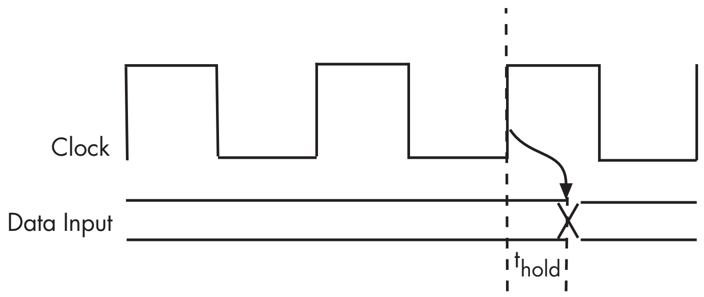
This option applies to reset input signals only if forced resets are enabled.

Usage Notes

The hold time is the amount of time that reset input signals and input data are held past the clock rising edge. The following figures show the application of a hold time (t_{hold}) for reset and data input signals when the signals are forced to active high and active low.



Hold Time for Reset Input Signals



Hold Time for Data Input Signals

Note A reset signal is always asserted for two cycles plus t_{hold}.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`ClockHighTime`, `ClockLowTime`, `ForceClock`

IgnoreDataChecking property

Purpose

Specify number of samples during which output data checking is suppressed

Settings

N

Default: `0`.

N must be a positive integer.

When *N* > 0, the test bench suppresses output data checking for the first *N* output samples after the clock enable output (`ce_out`) is asserted.

Usage Notes

When using pipelined block implementations, output data may be in an invalid state for some number of samples. To avoid spurious test bench errors, determine this number and set `IgnoreDataChecking` accordingly.

Be careful to specify *N* as a number of samples, not as a number of clock cycles. For a single-rate model, these are equivalent, but they are not equivalent for a multirate model.

You should use `IgnoreDataChecking` in cases where there is a state (register) initial condition in the HDL code that does not match the Simulink state, including the following specific cases:

- When you set the`DistributedPipelining` parameter to `'on'` for the MATLAB Function block (see “Distributed Pipeline Insertion for MATLAB Function Blocks”).
- When you set the `ResetType` parameter to `'None'` (see “ResetType”) for the following block types:
  - commcnvintrlv2/Convolutional Deinterleaver
  - commcnvintrlv2/Convolutional Interleaver
  - commcnvintrlv2/General Multiplexed Deinterleaver
  - commcnvintrlv2/General Multiplexed Interleaver
  - dspsigops/Delay

  - simulink/Additional Math & Discrete/Additional Discrete/Unit Delay Enabled
  - simulink/Commonly Used Blocks/Unit Delay
  - simulink/Discrete/Delay
  - simulink/Discrete/Memory
  - simulink/Discrete/Tapped Delay
  - simulink/User-Defined Functions/MATLAB Function
  - sflib/Chart
  - sflib/Truth Table
- When generating a black box interface to existing manually-written HDL code.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

InitializeBlockRAM property

Purpose

Enable or suppress generation of initial signal value for RAM blocks

Settings

`'on'` (default)

For RAM blocks, generate initial values of `'0'` for both the RAM signal and the output temporary signal.

`'off'`

For RAM blocks, do not generate initial values for either the RAM signal or the output temporary signal.

Usage Notes

This property applies to RAM blocks in the HDL Operations block library:

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

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`IgnoreDataChecking`

Purpose

Specify initial value driven on test bench inputs before data is asserted to DUT

Settings

`'on'`

Initial value driven on test bench inputs is`'0'`.

`'off'` (default)

Initial value driven on test bench inputs is `'X'` (unknown).

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

InlineConfigurations property

Purpose

Specify whether generated VHDL code includes inline configurations

Settings

`'on'` (default)

Selected (default)

Include VHDL configurations in files that instantiate a component.

`'off'`

Cleared

Suppress the generation of configurations and require user-supplied external configurations. Use this setting if you are creating your own VHDL configuration files.

Usage Notes

VHDL configurations can be either inline with the rest of the VHDL code for an entity or external in separate VHDL source files. By default, the coder includes configurations for a model within the generated VHDL code. If you are creating your own VHDL configuration files, you should suppress the generation of inline configurations.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`LoopUnrolling`, `SafeZeroConcat`, `UseAggregatesForConst`, `UseRisingEdge`

Purpose

Inline HDL code for MATLAB Function blocks

Settings

`'on'`

Inline HDL code for MATLAB Function blocks to avoid instantiation of code for custom blocks.

`'off'` (default)

Instantiate HDL code for MATLAB Function blocks and do not inline.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Examples

Enable inlining of HDL code:

```
mdl = 'my_custom_block_model';
hdlset_param(mdl,'InlineMATLABBlockCode','on');
```

Enable instantiation of HDL code:

```
mdl = 'my_custom_block_model';
hdlset_param(mdl,'InlineMATLABBlockCode','off');
```

Purpose

Specify HDL data type for model input ports

Settings

Default (for VHDL):`'std_logic_vector'`

Default (for VHDL): **std_logic_vector**

Specifies VHDL type `STD_LOGIC_VECTOR` for the model's input ports.

`'signed/unsigned'`

signed/unsigned

Specifies VHDL type `SIGNED` or `UNSIGNED` for the model's input ports.

`'wire'` (Verilog)

wire (Verilog)

If the target language is Verilog, the data type for all ports is `wire`. This property is not modifiable in this case.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`ClockEnableInputPort`, `OutputType`

Purpose

Specify string to append to instance section labels in VHDL `GENERATE` statements

Settings

`'string'`

Default: `'_gen'`

Specify a postfix string to append to instance section labels in VHDL `GENERATE` statements.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`BlockGenerateLabel`, `OutputGenerateLabel`

Purpose

Specify string appended to generated component instance names

Settings

`'string'`

Default: `''` (no postfix appended)

Specify a string to be appended to component instance names in generated code.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Specify string prefixed to generated component instance names

Settings

'string'

Default: `'u_'`

Specify a string to be prefixed to component instance names in generated code.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

LoopUnrolling property

Purpose

Specify whether VHDL `FOR` and `GENERATE` loops are unrolled and omitted from generated VHDL code

Settings

`'on'`

Selected

Unroll and omit `FOR` and `GENERATE` loops from the generated VHDL code.

In Verilog code, loops are always unrolled.

If you are using an electronic design automation (EDA) tool that does not support `GENERATE` loops, you can enable this option to omit loops from your generated VHDL code.

`'off'` (default)

Cleared (default)

Include `FOR` and `GENERATE` loops in the generated VHDL code.

Usage Notes

The setting of this option does not affect results obtained from simulation or synthesis of generated VHDL code.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`InlineConfigurations`, `SafeZeroConcat`, `UseAggregatesForConst`, `UseRisingEdge`

Purpose

Generate reusable HDL code for subsystems with identical mask parameters that differ only in value

Settings

`'on'`

Generate one HDL file for multiple masked subsystems with different values for tunable mask parameters. The coder automatically detects atomic subsystems with tunable mask parameters that are sharable.

Inside the subsystem, you can use the mask parameter only in the following blocks and parameters:

Block	Parameter	Limitation
Constant	**Constant value** on the Main tab of the dialog box	None
Gain	**Gain** on the Main tab of the dialog box	**Parameter data type** should be the same for all Gain blocks.

`'off'` (default)

Generate a separate HDL file for each masked subsystem.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

MaxComputationLatency property

Purpose

Specify the maximum number of time steps for which your DUT inputs are guaranteed to be stable

Settings

1 (default)

DUT input data can change every cycle.

N, where N is an integer greater than 1

DUT input data can change every N cycles.

Usage Notes

Use with `MaxOversampling` to prevent or reduce overclocking by constraining resource sharing and streaming optimizations.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Limit the maximum sample rate

Settings

0 (default)

Do not set a limit on the maximum sample rate.

1

Do not allow oversampling.

N, where N is an integer greater than 1

Allow oversampling up to N times the original model sample rate.

Usage Notes

Use with `MaxComputationLatency` to prevent or reduce overclocking by constraining resource sharing and streaming optimizations.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Select RAM architecture with or without clock enable for all RAMs in DUT subsystem

Settings

`'WithClockEnable'` (default)

Generate RAMs with clock enable.

`'WithoutClockEnable'`

Generate RAMs without clock enable.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Omit generation of clock enable logic for single-rate designs

Settings

`'on'`

Omit generation of clock enable logic for single-rate designs, wherever possible (see “Usage Notes” on page 3-77). The following VHDL code example does not define or examine a clock enable signal. When the clock signal (`clk`) goes high, the current signal value is output.

```
Unit_Delay_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay_out1 <= to_signed(0, 32);
    ELSIF clk'EVENT AND clk = '1' THEN
      Unit_Delay_out1 <= In1_signed;
    END IF;
  END PROCESS Unit_Delay_process;
```

`'off'` (default)

Generate clock enable logic. The following VHDL code extract represents a register with a clock enable (`enb`)

```
Unit_Delay_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay_out1 <= to_signed(0, 32);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        Unit_Delay_out1 <= In1_signed;
      END IF;
    END IF;
  END PROCESS Unit_Delay_process;
```

Usage Notes

In some cases, the coder emits clock enables even when `MinimizeClockEnables` is `'on'`. These cases are:

- Registers inside Enabled, State-Enabled, and Triggered subsystems.

- Multirate models.
- The coder emits clock enables for the following blocks:
  - commseqgen2/PN Sequence Generator
  - dspsigops/NCO

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

  - dspsrcs4/Sine Wave
  - hdldemolib/HDL FFT
  - built-in/DiscreteFir
  - dspmlti4/CIC Decimation
  - dspmlti4/CIC Interpolation
  - dspmlti4/FIR Decimation
  - dspmlti4/FIR Interpolation
  - dspadpt3/LMS Filter
  - dsparch4/Biquad Filter
  - dsparch4/Digital Filter

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Specify whether to optimize HDL code for debuggability or code coverage

Settings

`'on'`

Optimize for code coverage by minimizing intermediate signals. For example, suppose that the generated code with this setting *off* is:

```
const3 <= to_signed(24, 7);
subtractor_sub_cast <= resize(const3, 8);
subtractor_sub_cast_1 <= resize(delayout, 8);
subtractor_sub_temp <= subtractor_sub_cast - subtractor_sub_cast_1;
```

With this setting *on*, the output code is optimized to:

```
subtractor_sub_temp <= 24 - (resize(delayout, 8));
```

The intermediate signals `const3`, `subtractor_sub_cast`, and `subtractor_sub_cast_1` are removed.

`'off'` (default)

Optimize for debuggability by preserving intermediate signals.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

ModulePrefix property

Purpose

Specify prefix string for DUT module or entity name

Settings

`'string'`

Default: `''`

Specify a prefix for every module or entity name in the generated HDL code. The coder also applies this prefix to generated script file names.

Usage Notes

You can specify the module name prefix to avoid name collisions if you plan to instantiate the generated HDL code multiple times in a larger system.

For example, suppose you have a DUT, `myDut`, containing an internal module, `myUnit`. You can prefix the modules within your design with the string, `unit1_`, by entering the following command:

```
hdlset_param ('path/to/myDut', 'ModulePrefix','unit1_')
```

In the generated code, your HDL module names are `unit1_myDut` and `unit1_myUnit`, with corresponding HDL file names. Generated script file names also have the `unit1_` prefix.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

MulticyclePathInfo property

Purpose

Generate text file that reports multicycle path constraint information for use with synthesis tools

Settings

`'on'`

Selected

Generate a multicycle path information file.

`'off'` (default)

Do not generate a multicycle path information file.

Usage Notes

The file name for the multicycle path information file derives from the name of the DUT and the postfix string `'_constraints'`, as follows:

`DUTname``_constraints.txt`

For example, if the DUT name is `symmetric_fir`, the name of the multicycle path information file is `symmetric_fir_constraints.txt`.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Generate Multicycle Path Information Files"

MultifileTestBench property

Purpose

Divide generated test bench into helper functions, data, and HDL test bench code files

Settings

`'on'`

Write separate files for test bench code, helper functions, and test bench data. The file names are derived from the name of the DUT, the `TestBenchPostfix` property, and the `TestBenchDataPostfix` property as follows:

`DUTname_TestBenchPostfix_TestBenchDataPostfix`

For example, if the DUT name is `symmetric_fir`, and the target language is VHDL, the default test bench file names are:

- `symmetric_fir_tb.vhd`: test bench code
- `symmetric_fir_tb_pkg.vhd`: helper functions package
- `symmetric_fir_tb_data.vhd`: data package

If the DUT name is `symmetric_fir` and the target language is Verilog, the default test bench file names are:

- `symmetric_fir_tb.v`: test bench code
- `symmetric_fir_tb_pkg.v`: helper functions package
- `symmetric_fir_tb_data.v`: test bench data

`'off'` (default)

Write a single test bench file containing the HDL test bench code and helper functions and test bench data.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`TestBenchPostFix`, `TestBenchDataPostFix`

OptimizationReport property

Purpose

Display HTML optimization report

Settings

`'on'`

Create and display an HTML optimization report.

`'off'` (default)

Do not create an HTML optimization report.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Create and Use Code Generation Reports"

OptimizeTimingController property

Purpose

Optimize timing controller entity by implementing separate counters per rate

Settings

`'on'` (default)

A timing controller code file is generated if required by the design, for example:

- When code is generated for a multirate model.
- When a cascade block implementation for certain blocks is specified.

This file contains a module defining timing signals (clock, reset, external clock enable inputs and clock enable output) in a separate entity or module. In a multirate model, the timing controller entity generates the required rates from a single master clock using one or more counters and multiple clock enables.

When `OptimizeTimingController` is set `'on'` (the default), the coder generates multiple counters (one counter for each rate in the model). The benefit of this optimization is that it generates faster logic, and the size of the generated code is usually much smaller.

`'off'`

When `OptimizeTimingController` is set `'off'`, the timing controller uses one counter to generate the rates in the model.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Code Generation from Multirate Models", `EnablePrefix`, `TimingControllerPostfix`

Purpose

Specify string that labels output assignment block for VHDL `GENERATE` statements

Settings

'string'

Default: `'outputgen'`

Specify a postfix string to append to output assignment block labels in VHDL `GENERATE` statements.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`BlockGenerateLabel`, `OutputGenerateLabel`

OutputType property

Purpose

Specify HDL data type for model output ports

Settings

`'Same as input data type'` (VHDL default)

Same as input data type (VHDL default)

Output ports have the same type as the specified input port type.

`'std_logic_vector'`

std_logic_vector

Output ports have VHDL type `STD_LOGIC_VECTOR`.

`'signed/unsigned'`

signed/unsigned

Output ports have type `SIGNED` or `UNSIGNED`.

`'wire'` (Verilog)

wire (Verilog)

If the target language is Verilog, the data type for all ports is `wire`. This property is not modifiable in this case.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`ClockEnableInputPort`, `InputType`

Oversampling property

Purpose

Specify frequency of global oversampling clock as a multiple of model base rate

Settings

`N`

Default: 1.

`N` must be an integer greater than or equal to `0`.

`Oversampling` specifies `N`, the *oversampling factor* of a global oversampling clock. The oversampling factor expresses the global oversampling clock rate as a multiple of your model's base rate.

When you specify an oversampling factor greater than 1, the coder generates the global oversampling clock and derives the required timing signals from the clock signal. By default, the coder does not generate a global oversampling clock.

Generation of the global oversampling clock affects only generated HDL code. The clock does not affect the simulation behavior of your model.

If you want to generate a global oversampling clock:

- The oversampling factor must be an integer greater than or equal to 1.
- In a multirate DUT, the other rates in the DUT must divide evenly into the global oversampling rate.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Generate a Global Oversampling Clock"

PackagePostfix property

Purpose

Specify string to append to specified model or subsystem name to form name of package file

Settings

'string'

Default: `'_pkg'`

The coder applies this option only if a package file is required for the design.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`ClockProcessPostfix`, `EntityConflictPostfix`, `ReservedWordPostfix`

Purpose

Specify string to append to names of input or output pipeline registers generated for pipelined block implementations

Settings

'string'

Default: `'_pipe'`

When you specify a generation of input and/or output pipeline registers for selected blocks, the coder appends the string specified by the `PipelinePostfix` property when generating code for such pipeline registers.

For example, suppose you specify a pipelined output implementation for a Product block in a model, as in the following code:

```
 hdlset_param('sfir_fixed/symmetric_fir/Product','OutputPipeline', 2)
```

The following `makehdl` command specifies that the coder appends`'testpipe'` to generated pipeline register names.

```
makehdl(gcs,'PipelinePostfix','testpipe');
```

The following excerpt from generated VHDL code shows process the `PROCESS` code, with postfixed identifiers, that implements two pipeline stages:

```
Product_outtestpipe_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Product_outtestpipe_reg <= (OTHERS => to_signed(0, 33));
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        Product_outtestpipe_reg(0) <= Product_out1;
        Product_outtestpipe_reg(1) <= Product_outtestpipe_reg(0);
      END IF;
    END IF;
  END PROCESS Product_outtestpipe_process;
```

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"HDL Block Properties", "InputPipeline", "OutputPipeline"

Purpose

Enable to prevent distributed pipelining from moving design delays

Settings

`'on'`

Prevent distributed pipelining from moving design delays, such as:

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

`'off'` (default)

Allow distributed pipelining to move design delays.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Concepts

- "Distributed Pipelining and Hierarchical Distributed Pipelining"

RAMMappingThreshold property

Purpose

Specify the minimum RAM size required for mapping to RAMs instead of registers

Settings

`N`

Default: 256.

`N` must be an integer greater than or equal to 0.

`RAMMappingThreshold` defines the minimum RAM size required for mapping to RAM instead of registers. This threshold applies to:

- Delay blocks
- Persistent variables in MATLAB Function blocks

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Example

To change the RAM mapping threshold for a model, use the `hdlset_param` function. For example:

```
hdlset_param('sfir_fixed', 'RAMMappingThreshold', 1024);
```

That command sets the threshold for the `sfir_fixed` model to 1024 bits.

See Also

- “UseRAM” in the HDL Coder documentation
- “MapPersistentVarsToRAM” in the HDL Coder documentation

Purpose

Enable or disable generation of hyperlinked requirements comments in HTML code generation reports

Settings

`'on'` (default)

If the model includes requirements comments, generate hyperlinked requirements comments within the HTML code generation report. The comments link to the corresponding requirements documents.

`'off'`

When generating an HTML code generation report, render requirements as comments within the generated code

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Create and Use Code Generation Reports", "Generate Code with Annotations or Comments", `Traceability`

ReservedWordPostfix property

Purpose

Specify string appended to identifiers for entities, signals, constants, or other model elements that conflict with VHDL or Verilog reserved words

Settings

`'string'`

Default: `'_rsvd'`.

The reserved word postfix is applied identifiers (for entities, signals, constants, or other model elements) that conflict with VHDL or Verilog reserved words. For example, if your generating model contains a signal named `mod`, the coder adds the postfix `_rsvd` to form the name `mod_rsvd`.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`ClockProcessPostfix`, `EntityConflictPostfix`, `ReservedWordPostfix`

Purpose

Specify asserted (active) level of reset input signal

Settings

`'active-high'` (default)

Active-high (default)

Specify that the reset input signal must be driven high (1) to reset registers in the model. For example, the following code fragment checks whether `reset` is active high before populating the `delay_pipeline` register:

```
Delay_Pipeline_Process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    delay_pipeline(0 TO 50) <= (OTHERS => (OTHERS => '0'));
.
.
.
```

`'active-low'`

Active-low

Specify that the reset input signal must be driven low (0) to reset registers in the model. For example, the following code fragment checks whether `reset` is active low before populating the `delay_pipeline` register:

```
Delay_Pipeline_Process : PROCESS (clk, reset)
BEGIN
  IF reset = '0' THEN
    delay_pipeline(0 TO 50) <= (OTHERS => (OTHERS => '0'));
.
.
.
```

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also `ResetType`, `ClockInputPort`, `ClockEdge`

Purpose

Name HDL port for model's reset input

Settings

'string'

Default: 'reset'.

The string specifies the name for the model's reset input port. If you override the default with (for example) the string 'chip_reset' for the generating system myfilter, the generated entity declaration might look as follows:

```
ENTITY myfilter IS
   PORT( clk            :  IN  std_logic;
         clk_enable     :  IN  std_logic;
         chip_reset     :  IN  std_logic;
         myfilter_in    :  IN  std_logic_vector (15 DOWNTO 0);
         myfilter_out   :  OUT std_logic_vector (15 DOWNTO 0);
         );
END myfilter;
```

If you specify a string that is a VHDL or Verilog reserved word, the code generator appends a reserved word postfix string to form a valid VHDL or Verilog identifier. For example, if you specify the reserved word signal, the resulting name string would be signal_rsvd. See ReservedWordPostfix for more information.

Usage Notes

If the reset asserted level is set to active high, the reset input signal is asserted active high (1) and the input value must be high (1) for the entity's registers to be reset. If the reset asserted level is set to active low, the reset input signal is asserted active low (0) and the input value must be low (0) for the entity's registers to be reset.

Set or View This Property

To set this property, use hdlset_param or makehdl. To view the property value, use hdlget_param.

See Also `ClockEnableInputPort`, `InputType`, `OutputType`

Purpose

Define length of time (in clock cycles) during which reset is asserted

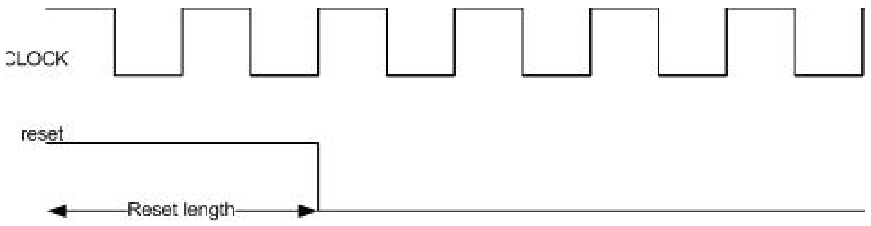
Settings

N

Default: 2.

N must be an integer greater than or equal to 0.

`Resetlength` defines *N*, the number of clock cycles during which reset is asserted. The following figure illustrates the default case, in which the reset signal (active-high) is asserted for 2 clock cycles.



Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Specify whether to use asynchronous or synchronous reset logic when generating HDL code for registers

Settings

`'async'` (default)

Asynchronous (default)

Use asynchronous reset logic. The following process block, generated by a Unit Delay block, illustrates the use of asynchronous resets. When the reset signal is asserted, the process block performs a reset, without checking for a clock event.

```
Unit_Delay1_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay1_out1 <= (OTHERS => '0');
    ELSIF clk'event AND clk = '1' THEN
      IF clk_enable = '1' THEN
        Unit_Delay1_out1 <= signed(x_in);
      END IF;
    END IF;
  END PROCESS Unit_Delay1_process;
```

`'sync'`

Synchronous

Use synchronous reset logic. Code for a synchronous reset follows. The following process block, generated by a Unit Delay block, checks for a clock event, the rising edge, before performing a reset:

```
Unit_Delay1_process : PROCESS (clk)
  BEGIN
    IF rising_edge(clk) THEN
      IF reset = '1' THEN
        Unit_Delay1_out1 <= (OTHERS => '0');
      ELSIF clk_enable = '1' THEN
      Unit_Delay1_out1 <= signed(x_in);
    END IF;
  END IF;
END PROCESS Unit_Delay1_process;
```

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`ResetAssertedLevel`

ResourceReport property

Purpose

Display HTML resource utilization report

Settings

`'on'`

Create and display an HTML resource utilization report (bill of materials).

`'off'` (default)

Do not create an HTML resource utilization report.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Create and Use Code Generation Reports"

Purpose

Specify syntax for concatenated zeros in generated VHDL code

Settings

`'on'` (default)

Selected (default)

Use the type-safe syntax, `'0' & '0'`, for concatenated zeros. Typically, this syntax is preferred.

`'off'`

Cleared

Use the syntax `"000000..."` for concatenated zeros. This syntax can be easier to read and is more compact, but it can lead to ambiguous types.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`LoopUnrolling`, `UseAggregatesForConst`, `UseRisingEdge`

Purpose

Flatten vector ports into structure of scalar ports in VHDL code

Settings

`'on'`

When generating code for a vector port, generate a structure of scalar ports

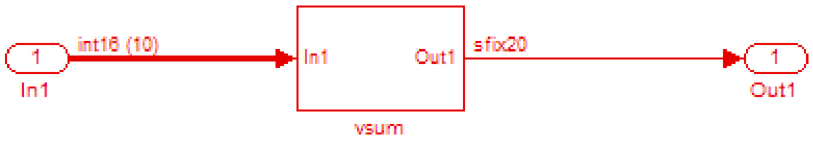
`'off'` (default)

Do not generate a structure of scalar ports for a vector port.

Usage Notes

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

For example, consider the subsystem `vsum` in the following figure.



By default, `ScalarizePorts` is `'off'`. The coder generates a type definition and port declaration for the vector port `In1` like the following:

```
PACKAGE simplevectorsum_pkg IS
  TYPE vector_of_std_logic_vector16 IS ARRAY (NATURAL RANGE <>)
     OF std_logic_vector(15 DOWNTO 0);
  TYPE vector_of_signed16 IS ARRAY (NATURAL RANGE <>) OF signed(15 DOWNTO 0);
END simplevectorsum_pkg;
.
.
.
ENTITY vsum IS
```

```
  PORT( In1     :  IN    vector_of_std_logic_vector16(0 TO 9);  -- int16 [10]
        Out1    :  OUT   std_logic_vector(19 DOWNTO 0)  -- sfix20
        );
END vsum;
```

Under VHDL typing rules two types declared in this manner are not compatible across design units. This may cause problems if you need to interface two or more generated VHDL code modules.

You can flatten such a vector port into a structure of scalar ports by enabling `ScalarizePorts` in your `makehdl` command, as in the following example.

```
makehdl(gcs,'ScalarizePorts','on')
```

The listing below shows the generated ports.

```
ENTITY vsum IS
  PORT( In1_0              :   IN    std_logic_vector(15 DOWNTO 0);  -- int16
        In1_1              :   IN    std_logic_vector(15 DOWNTO 0);  -- int16
        In1_2              :   IN    std_logic_vector(15 DOWNTO 0);  -- int16
        In1_3              :   IN    std_logic_vector(15 DOWNTO 0);  -- int16
        In1_4              :   IN    std_logic_vector(15 DOWNTO 0);  -- int16
        In1_5              :   IN    std_logic_vector(15 DOWNTO 0);  -- int16
        In1_6              :   IN    std_logic_vector(15 DOWNTO 0);  -- int16
        In1_7              :   IN    std_logic_vector(15 DOWNTO 0);  -- int16
        In1_8              :   IN    std_logic_vector(15 DOWNTO 0);  -- int16
        In1_9              :   IN    std_logic_vector(15 DOWNTO 0);  -- int16
        Out1               :   OUT   std_logic_vector(19 DOWNTO 0)  -- sfix20
        );
END vsum;
```

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Generate Black Box Interface for Referenced Model"

SimulatorFlags property

Purpose Specify simulator flags to apply to generated compilation scripts

Settings `'string'`

Default: `''`

Specify options that are specific to your application and the simulator you are using. For example, if you must use the 1076–1993 VHDL compiler, specify the flag `-93`.

Usage Notes The flags you specify with this option are added to the compilation command in generated compilation scripts. The simulation command string is specified by the `HDLCompileVHDLCmd` or `HDLCompileVerilogCmd` properties.

Set or View This Property To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Specify string to append to specified name to form name of file containing model VHDL architecture

Settings

`'string'`

Default: `'_arch'`.

This option applies only if you direct the coder to place the generated VHDL entity and architecture code in separate files.

Usage Notes

The option applies only if you direct the coder to place the filter's entity and architecture in separate files.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`SplitEntityArch`, `SplitEntityFilePostfix`

SplitEntityArch property

Purpose

Specify whether generated VHDL entity and architecture code is written to single VHDL file or to separate files

Settings

`'on'`

Selected

Write the generated VHDL code to a single file.

`'off'`(default)

Cleared (default)

Write the code for the generated VHDL entity and architecture to separate files.

The names of the entity and architecture files derive from the base file name (as specified by the generating model or subsystem name). By default, postfix strings identifying the file as an entity (`_entity`) or architecture (`_arch`) are appended to the base file name. You can override the default and specify your own postfix string.

For example, instead of all generated code residing in `MyFIR.vhd`, you can specify that the code reside in `MyFIR_entity.vhd` and `MyFIR_arch.vhd`.

Note This property is specific to VHDL code generation. It does not apply to Verilog code generation and should not be enabled when generating Verilog code.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`SplitArchFilePostfix`, `SplitEntityFilePostfix`

Purpose

Specify string to append to specified model name to form name of generated VHDL entity file

Settings

`'string'`

Default: `'_entity'`

This option applies only if you direct the coder to place the generated VHDL entity and architecture code in separate files.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`SplitArchFilePostfix`, `SplitEntityArch`

SynthesisTool property

Purpose

Specify synthesis tool

Settings

`''` (default)

If you do not specify a synthesis tool, the default is `''`.

`'Altera Quartus II'`

Specify Altera Quartus II as your synthesis tool.

`'Xilinx ISE'`

Specify Xilinx ISE as your synthesis tool.

Usage

To specify Altera Quartus II as the synthesis tool for a DUT subsystem, `myDUT`:

```
hdlset_param (myDUT, 'SynthesisTool', 'Altera Quartus II')
```

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

Properties

`SynthesisToolDeviceNameSynthesisToolPackageNameSynthesisToolSpeedValue`

Purpose

Specify target device chip family name

Settings

'string'

Default: `''`

Specify the target device chip family name for your model.

To find the chip family name for your target device:

1. At the MATLAB command line, enter:

   ```
   hdlcoder.supportedDevices
   ```

2. Open the linked report and find your target device details.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

Functions

`hdlcoder.supportedDevices`

Properties

`SynthesisToolDeviceNameSynthesisToolPackageNameSynthesisToolSpeedValue`

SynthesisToolDeviceName property

Purpose

Specify target device name

Settings

`'string'`

Default: `''`

Specify the target device name for your model.

To find the name for your target device:

1. At the MATLAB command line, enter:

   ```
   hdlcoder.supportedDevices
   ```

2. Open the linked report and find your target device details.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

Functions

`hdlcoder.supportedDevices`

Properties

`SynthesisToolChipFamilySynthesisToolPackageNameSynthesisToolSpeedValue`

Purpose

Specify target device package name

Settings

`'string'`

Default: `''`

Specify the target device package name for your model.

To find the package name for your target device:

1. At the MATLAB command line, enter:

   ```
   hdlcoder.supportedDevices
   ```

2. Open the linked report and find your target device details.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

Functions

`hdlcoder.supportedDevices`

Properties

`SynthesisToolChipFamilySynthesisToolDeviceNameSynthesisToolSpeedValue`

SynthesisToolSpeedValue property

Purpose

Specify target device speed value

Settings

'string'

Default: `''`

Specify the target device speed value for your model.

To find the speed value for your target device:

1. At the MATLAB command line, enter:

   ```
   hdlcoder.supportedDevices
   ```

2. Open the linked report and find your target device details.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

Functions

`hdlcoder.supportedDevices`

Properties

`SynthesisToolChipFamilySynthesisToolDeviceNameSynthesisToolPackageName`

Purpose

Identify folder into which the coder writes generated output files

Settings

`'string'`

Default: `'hdlsrc'`

Specify a subfolder under the current working folder into which the coder writes generated files. The string can specify a complete path name.

If the target folder does not exist, the coder creates it.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`VerilogFileExtension`, `VHDLFileExtension`

TargetLanguage property

Purpose

Specify HDL language to use for generated code

Settings

`'VHDL'` (default)

VHDL (default)

Generate VHDL code.

`'Verilog'`

Verilog

Generate Verilog code.

The generated HDL code complies with the following standards:

- VHDL-1993 (IEEE® 1076-1993) or later
- Verilog-2001 (IEEE 1364-2001) or later

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Define elapsed time in clock cycles between deassertion of reset and assertion of clock enable

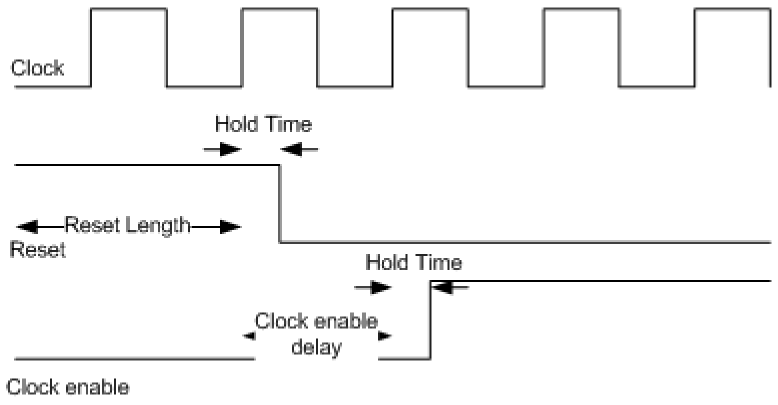
Settings

N (integer number of clock cycles)

Default: 1

The `TestBenchClockEnableDelay` property specifies a delay time *N*, expressed in base-rate clock cycles (the default value is 1) elapsed between the time the reset signal is deasserted and the time the clock enable signal is first asserted. `TestBenchClockEnableDelay` works in conjunction with the `HoldTime` property; after deassertion of reset, the clock enable goes high after a delay of *N* base-rate clock cycles plus the delay specified by `HoldTime`.

In the figure below, the reset signal (active-high) deasserts after the interval labelled `Hold Time`. The clock enable asserts after a further interval labelled `Clock enable delay`.



Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`HoldTime`, `ResetLength`

TestBenchDataPostFix property

Purpose

Specify suffix added to test bench data file name when generating multifile test bench

Settings

'string'

Default: `'_data'`.

The coder applies `TestBenchDataPostFix` only when generating a multi-file test bench (i.e. when `MultifileTestBench` is `'on'`).

For example, if the name of your DUT is `my_test`, and `TestBenchPostFix` has the default value `_tb`, the coder adds the postfix `_data` to form the test bench data file name `my_test_tb_data`.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`MultifileTestBench`, `TestBenchPostFix`

Purpose

Specify suffix to test bench name

Settings

'*string*'

Default: '_tb'.

For example, if the name of your DUT is my_test, the coder adds the postfix _tb to form the name my_test_tb.

Set or View This Property

To set this property, use hdlset_param or makehdl. To view the property value, use hdlget_param.

See Also

MultifileTestBench, TestBenchDataPostFix

TimingControllerArch property

Purpose

Generate reset for timing controller

Settings

`'resettable'`

Generate a reset for the timing controller. If you select this option, the `ClockInputs` property value must be `'Single'`.

`'default'` (default)

Do not generate a reset for the timing controller.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

Properties

`ClockInputs`

Related Examples

- “Generate Reset for Timing Controller”

Purpose

Specify suffix appended to DUT name to form timing controller name

Settings

`'`*`string`*`'`

Default: `'_tc'`.

A timing controller code file is generated if required by the design, for example:

- When code is generated for a multirate model.
- When a cascade block implementation for certain blocks is specified.

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 your DUT is `my_test`, in the default case the coder adds the postfix `_tc` to form the timing controller name `my_test_tc`.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`OptimizeTimingController`, "Code Generation from Multirate Models"

TestBenchReferencePostFix property

Purpose

Specify string appended to names of reference signals generated in test bench code

Settings

`'string'`

Default: `'_ref'`.

Reference signal data is represented as arrays in the generated test bench code. The string specified by `TestBenchReferencePostFix` is appended to the generated signal names.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Enable or disable creation of HTML code generation report with code-to-model and model-to-code hyperlinks

Settings

`'on'`

Create and display an HTML code generation report.

`'off'` (default)

Do not create an HTML code generation report.

Usage Notes

You can use the `RequirementComments` property to generate hyperlinked requirements comments within the HTML code generation report. The requirements comments link to the corresponding requirements documents for your model.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

"Create and Use Code Generation Reports", "Generate Code with Annotations or Comments", `RequirementComments`

TriggerAsClock property

Purpose

Use trigger signal in triggered subsystem as a clock

Settings

`'on'`

For triggered subsystems, use the trigger input signal as a clock in the generated HDL code.

`'off'` (default)

For triggered subsystems, do not use the trigger input signal as a clock in the generated HDL code.

Usage Example

Use `hdlset_param` or `makehdl` to set this property.

For example, to generate HDL code that uses the trigger signal as clock for triggered subsystems within the `sfir_fixed/symmetric_fir` DUT subsystem, enter:

```
makehdl ('sfir_fixed/symmetric_sfir','TriggerAsClock','on')
```

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Concepts

- “Use Trigger As Clock in Triggered Subsystems”

Purpose

Specify whether constants are represented by aggregates, including constants that are less than 32 bits

Settings

`'on'`

Selected

Specify that constants, including constants that are less than 32 bits, be represented by aggregates. The following VHDL code show a scalar less than 32 bits represented as an aggregate:

```
GainFactor_gainparam <= (14 => '1',  OTHERS => '0');
```

`'off'` (default)

Cleared(default)

Specify that the coder represent constants less than 32 bits as scalars and constants greater than or equal to 32 bits as aggregates. The following VHDL code was generated by default for a value less than 32 bits:

```
GainFactor_gainparam <= to_signed(16384, 16);
```

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`LoopUnrolling`, `SafeZeroConcat`, `UseRisingEdge`

UseFileIOInTestBench property

Purpose

Specify whether to use data files for reading and writing test bench stimulus and reference data

Settings

`'on'`

Selected

Create and use data files for reading and writing test bench stimulus and reference data.

`'off'` (default)

Cleared(default)

Generated test bench contains stimulus and reference data as constants.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Specify comment line in header of generated HDL and test bench files

Settings

'string'

The comment is generated in each of the generated code and test bench files. The code generator adds leading comment characters for the target language. When newlines or line feeds are included in the string, the code generator emits single-line comments for each newline.

For example, the following `makehdl` command adds two comment lines to the header in a generated VHDL file.

```
makehdl(gcb,'UserComment','This is a comment line.\nThis is a second line.')
```

The resulting header comment block for subsystem `symmetric_fir` would appear as follows:

```
-- -------------------------------------------------------------
--
-- Module: symmetric_fir
-- Simulink Path: sfir_fixed/symmetric_fir
-- Created: 2006-11-20 15:55:25
-- Hierarchy Level: 0
--
-- This is a comment line.
-- This is a second line.
--
-- Simulink model description for sfir_fixed:
  -- This model shows how to use HDL Coder to check, generate,
  -- and verify HDL for a fixed-point symmetric FIR filter.
--
-- -------------------------------------------------------------
```

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

UseRisingEdge property

Purpose

Specify VHDL coding style used to detect clock transitions

Settings

`'on'`

Selected

Generated code uses the VHDL `rising_edge` or `falling_edge` function to detect clock transitions.

For example, the following code, generated from a Unit Delay block, uses `rising_edge` to detect positive clock transitions:

```
Unit_Delay1_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay1_out1 <= (OTHERS => '0');
    ELSIF rising_edge(clk) THEN
      IF clk_enable = '1' THEN
        Unit_Delay1_out1 <= signed(x_in);
      END IF;
    END IF;
  END PROCESS Unit_Delay1_process;
```

`'off'` (default)

Cleared(default)

Generated code uses the `'event` syntax.

For example, the following code, generated from a Unit Delay block, uses `clk'event AND clk = '1'` to detect positive clock transitions:

```
Unit_Delay1_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay1_out1 <= (OTHERS => '0');
    ELSIF clk'event AND clk = '1' THEN
```

```
      IF clk_enable = '1' THEN
        Unit_Delay1_out1 <= signed(x_in);
      END IF;
    END IF;
  END PROCESS Unit_Delay1_process;
```

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`LoopUnrolling`, `SafeZeroConcat`, `UseAggregatesForConst`

UseVerilogTimescale property

Purpose

Use compiler `` `timescale `` directives in generated Verilog code

Settings

`'on'` (default)

Selected (default)

Use compiler `` `timescale `` directives in generated Verilog code.

`'off'`

Cleared

Suppress the use of compiler `` `timescale `` directives in generated Verilog code.

Usage Notes

The `` `timescale `` directive provides a way of specifying different delay values for multiple modules in a Verilog file. This setting does not affect the generated test bench.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`LoopUnrolling`, `SafeZeroConcat`, `UseAggregatesForConst`, `UseRisingEdge`

Purpose

Specify string prefixed to vector names in generated code

Settings

`'string'`

Default: `'vector_of_'`

Specify a string to be prefixed to vector names in generated code.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Verbosity property

Purpose

Specify level of detail for messages displayed during code generation

Settings

Default: 1

0

When `Verbosity` is set to 0, code generation progress messages are not displayed as code generation proceeds. When `Verbosity` is set to 1, more detailed progress messages are displayed.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Specify file type extension for generated Verilog files

Settings

`'string'`

The default file type extension for generated Verilog files is `.v`.

See Also

`TargetLanguage`

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

VHDLArchitectureName property

Purpose Specify architecture name for generated HDL code

Settings *'string'*

The default architecture name is `'rtl'`.

Set or View This Property To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

Purpose

Specify file type extension for generated VHDL files

Settings

'string'

The default file type extension for generated VHDL files is `.vhd`.

Set or View This Property

To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also

`TargetLanguage`

VHDLLibraryName property

Purpose Specify name of target library for generated HDL code

Settings `'`*`string`*`'`

The default target library name is `'work'`.

Set or View This Property To set this property, use `hdlset_param` or `makehdl`. To view the property value, use `hdlget_param`.

See Also `HDLCompileInit`

4

Classes reference for HDL code generation from Simulink

hdlcoder.OptimizationConfig

Purpose

`hdlcoder.optimizeDesign` configuration object

Description

Use the `hdlcoder.OptimizationConfig` object to set options for the `hdlcoder.optimizeDesign` function.

Maximum Clock Frequency Configuration

To configure `hdlcoder.optimizeDesign` to maximize the clock frequency of your design:

- Set `ExplorationMode` to `hdlcoder.OptimizationConfig.ExplorationMode.BestFrequency`.
- Set `ResumptionPoint` to the default, `''`.

You can optionally set `IterationLimit` and `TestbenchGeneration` to nondefault values. The coder ignores the `TargetFrequency` setting.

Target Clock Frequency Configuration

To configure `hdlcoder.optimizeDesign` to meet a target clock frequency:

- Set `ExplorationMode` to `hdlcoder.OptimizationConfig.ExplorationMode.TargetFrequency`.
- Set `TargetFrequency` to your target clock frequency.
- Set `ResumptionPoint` to the default, `''`

You can optionally set `IterationLimit` and `TestbenchGeneration` to nondefault values.

Resume From Interruption Configuration

To configure `hdlcoder.optimizeDesign` to resume after an interruption, specify `ResumptionPoint`.

When you set `ResumptionPoint` to a nondefault value, the other properties are ignored.

Construction

optimcfg = `hdlcoder.OptimizationConfig` creates an hdlcoder.OptimizationConfig object for automatic iterative HDL design optimization.

Properties

ExplorationMode - Optimization target mode

`hdlcoder.OptimizationConfig.ExplorationMode.BestFrequency` (default) | `hdlcoder.OptimizationConfig.ExplorationMode.TargetFrequency`

Optimization target mode, specified as one of these values:

`hdlcoder.OptimizationConfig.ExplorationMode.BestFrequency`	Optimizes the design to try to achieve the maximum clock frequency `hdlcoder.OptimizationConfig.Explorat` is the default.
`hdlcoder.OptimizationConfig.ExplorationMode.TargetFrequency`	Optimizes the design to try to achieve the specified target clock frequency

IterationLimit - Maximum number of iterations

1 (default) | positive integer

Maximum number of optimization iterations before exiting, specified as a positive integer.

If `ExplorationMode` is `hdlcoder.OptimizationConfig.ExplorationMode.BestFrequency`, the coder runs this number of iterations.

If `ExplorationMode` is `hdlcoder.OptimizationConfig.ExplorationMode.TargetFrequency`, the coder runs the number of iterations needed to meet the target frequency. Otherwise, the coder runs the maximum number of iterations.

ResumptionPoint - Folder containing optimization data from earlier iteration

`''` (default) | string

Name of folder that contains previously-generated optimization iteration data, specified as a string. The folder is a subfolder of `hdlexpl`, and the folder name begins with the string, `Iter`.

When you set `ResumptionPoint` to a nondefault value, `hdlcoder.optimizeDesign` ignores the other configuration object properties.

Example: `'Iter1-26-Sep-2013-10-19-13'`

TargetFrequency - Target clock frequency
`Inf` (default) | double

Target clock frequency, specified as a double in MHz. Specify when `ExplorationMode` is `hdlcoder.OptimizationConfig.ExplorationMode.TargetFrequency`.

Examples

Configure hdlcoder.optimizeDesign for maximum clock frequency

Open the model and specify the DUT subsystem.

```
model = 'sfir_fixed';
dutSubsys = 'symmetric_fir';
open_system(model);
hdlset_param(model,'HDLSubsystem',[model,'/',dutSubsys]);
```

Set your synthesis tool and target device options.

```
hdlset_param (model,'SynthesisTool','Xilinx ISE', ...
                    'SynthesisToolChipFamily','Zynq', ...
                    'SynthesisToolDeviceName','xc7z030', ...
                    'SynthesisToolPackageName','fbg484', ...
                    'SynthesisToolSpeedValue','-3')
```

Enable HDL test bench generation.

```
hdlset_param(model,'GenerateHDLTestBench','on');
```

Save your model.

You must save your model if you want to regenerate code later without rerunning the iterative optimizations, or resume your run if it is interrupted. When you use `hdlcoder.optimizeDesign` to regenerate code or resume an interrupted run, the coder checks the model checksum and generates an error if the model has changed.

Create an optimization configuration object, `oc`.

```
oc = hdlcoder.OptimizationConfig;
```

Set the iteration limit to 10.

```
oc.IterationLimit = 10;
```

Optimize the model.

```
hdlcoder.optimizeDesign(model,oc)
```

```
hdlset_param('sfir_fixed', 'HDLSubsystem', 'sfir_fixed/symmetric_fir'
hdlset_param('sfir_fixed', 'SynthesisTool', 'Xilinx ISE');
hdlset_param('sfir_fixed', 'SynthesisToolChipFamily', 'Zynq');
hdlset_param('sfir_fixed', 'SynthesisToolDeviceName', 'xc7z030');
hdlset_param('sfir_fixed', 'SynthesisToolPackageName', 'fbg484');
hdlset_param('sfir_fixed', 'SynthesisToolSpeedValue', '-3');

Iteration 0
Generate and synthesize HDL code ...
(CP ns) 16.26 (Constraint ns) 5.85 (Elapsed s) 143.66 Iteration 1
Generate and synthesize HDL code ...
(CP ns) 16.26 (Constraint ns) 5.85 (Elapsed s) 278.72 Iteration 2
Generate and synthesize HDL code ...
(CP ns) 10.25 (Constraint ns) 12.73 (Elapsed s) 427.22 Iteration 3
Generate and synthesize HDL code ...
(CP ns) 9.55 (Constraint ns) 9.73 (Elapsed s) 584.37 Iteration 4
Generate and synthesize HDL code ...
(CP ns) 9.55 (Constraint ns) 9.38 (Elapsed s) 741.04 Iteration 5
Generate and synthesize HDL code ...
```

```
Exiting because critical path cannot be further improved.
Summary report: summary.html
Achieved Critical Path (CP) Latency : 9.55 ns  Elapsed : 741.04 s
Iteration 0: (CP ns) 16.26 (Constraint ns) 5.85 (Elapsed s) 143.66
Iteration 1: (CP ns) 16.26 (Constraint ns) 5.85 (Elapsed s) 278.72
Iteration 2: (CP ns) 10.25 (Constraint ns) 12.73 (Elapsed s) 427.22
Iteration 3: (CP ns) 9.55 (Constraint ns) 9.73 (Elapsed s) 584.37
Iteration 4: (CP ns) 9.55 (Constraint ns) 9.38 (Elapsed s) 741.04
Final results are saved in
    /tmp/hdlsrc/sfir_fixed/hdlexpl/Final-07-Jan-2014-17-04-41
Validation model: gm_sfir_fixed_vnl
```

Then coder stops after five iterations because the fourth and fifth iterations had the same critical path, which indicates that the coder has found the minimum critical path. The design's maximum clock frequency after optimization is 1 / 9.55 ns, or 104.71 MHz.

Configure hdlcoder.optimizeDesign for target clock frequency

Open the model and specify the DUT subsystem.

```
model = 'sfir_fixed';
dutSubsys = 'symmetric_fir';
open_system(model);
hdlset_param(model,'HDLSubsystem',[model,'/',dutSubsys]);
```

Set your synthesis tool and target device options.

```
hdlset_param (model,'SynthesisTool','Xilinx ISE', ...
                    'SynthesisToolChipFamily','Zynq', ...
                    'SynthesisToolDeviceName','xc7z030', ...
                    'SynthesisToolPackageName','fbg484', ...
                    'SynthesisToolSpeedValue','-3')
```

Disable HDL test bench generation.

```
hdlset_param(model,'GenerateHDLTestBench','off');
```

Save your model.

You must save your model if you want to regenerate code later without rerunning the iterative optimizations, or resume your run if it is interrupted. When you use `hdlcoder.optimizeDesign` to regenerate code or resume an interrupted run, the coder checks the model checksum and generates an error if the model has changed.

Create an optimization configuration object, `oc`.

```
oc = hdlcoder.OptimizationConfig;
```

Configure the automatic iterative optimization to stop after it reaches a clock frequency of 50MHz, or 10 iterations, whichever comes first.

```
oc.ExplorationMode = ...
    hdlcoder.OptimizationConfig.ExplorationMode.TargetFrequency;
oc.TargetFrequency = 50;
oc.IterationLimit = 10; =
```

Optimize the model.

```
hdlcoder.optimizeDesign(model,oc)
```

```
hdlset_param('sfir_fixed', 'GenerateHDLTestBench', 'off');
hdlset_param('sfir_fixed', 'HDLSubsystem', 'sfir_fixed/symmetric_fir'
hdlset_param('sfir_fixed', 'SynthesisTool', 'Xilinx ISE');
hdlset_param('sfir_fixed', 'SynthesisToolChipFamily', 'Zynq');
hdlset_param('sfir_fixed', 'SynthesisToolDeviceName', 'xc7z030');
hdlset_param('sfir_fixed', 'SynthesisToolPackageName', 'fbg484');
hdlset_param('sfir_fixed', 'SynthesisToolSpeedValue', '-3');

Iteration 0
Generate and synthesize HDL code ...
(CP ns) 16.26 (Constraint ns) 20.00 (Elapsed s) 134.02 Iteration 1
Generate and synthesize HDL code ...
Exiting because constraint (20.00 ns) has been met (16.26 ns).
Summary report: summary.html
Achieved Critical Path (CP) Latency : 16.26 ns  Elapsed : 134.02 s
```

```
Iteration 0: (CP ns) 16.26 (Constraint ns) 20.00 (Elapsed s) 134.02
Final results are saved in
    /tmp/hdlsrc/sfir_fixed/hdlexpl/Final-07-Jan-2014-17-07-14
Validation model: gm_sfir_fixed_vnl
```

Then coder stops after one iteration because it has achieved the target clock frequency. The critical path is 16.26 ns, a clock frequency of 61.50 GHz.

Configure hdlcoder.optimizeDesign to resume from interruption

Open the model and specify the DUT subsystem.

```
model = 'sfir_fixed';
dutSubsys = 'symmetric_fir';
open_system(model);
hdlset_param(model,'HDLSubsystem',[model,'/',dutSubsys]);
```

Set your synthesis tool and target device options to the same values as in the interrupted run.

```
hdlset_param (model,'SynthesisTool','Xilinx ISE', ...
                    'SynthesisToolChipFamily','Zynq', ...
                    'SynthesisToolDeviceName','xc7z030', ...
                    'SynthesisToolPackageName','fbg484', ...
                    'SynthesisToolSpeedValue','-3')
```

Enable HDL test bench generation.

```
hdlset_param(model,'GenerateHDLTestBench','on');
```

Create an optimization configuration object, `oc`.

```
oc = hdlcoder.OptimizationConfig;
```

Configure the automatic iterative optimization to run using data from the first iteration of a previous run.

```
oc.ResumptionPoint = 'Iter5-07-Jan-2014-17-04-29';
```

Optimize the model.

```
hdlcoder.optimizeDesign(model,oc)
```

```
hdlset_param('sfir_fixed', 'HDLSubsystem', 'sfir_fixed/symmetric_fir'
hdlset_param('sfir_fixed', 'SynthesisTool', 'Xilinx ISE');
hdlset_param('sfir_fixed', 'SynthesisToolChipFamily', 'Zynq');
hdlset_param('sfir_fixed', 'SynthesisToolDeviceName', 'xc7z030');
hdlset_param('sfir_fixed', 'SynthesisToolPackageName', 'fbg484');
hdlset_param('sfir_fixed', 'SynthesisToolSpeedValue', '-3');

Try to resume from resumption point: Iter5-07-Jan-2014-17-04-29
Iteration 5
Generate and synthesize HDL code ...
Exiting because critical path cannot be further improved.
Summary report: summary.html
Achieved Critical Path (CP) Latency : 9.55 ns  Elapsed : 741.04 s
Iteration 0: (CP ns) 16.26 (Constraint ns) 5.85 (Elapsed s) 143.66
Iteration 1: (CP ns) 16.26 (Constraint ns) 5.85 (Elapsed s) 278.72
Iteration 2: (CP ns) 10.25 (Constraint ns) 12.73 (Elapsed s) 427.22
Iteration 3: (CP ns) 9.55 (Constraint ns) 9.73 (Elapsed s) 584.37
Iteration 4: (CP ns) 9.55 (Constraint ns) 9.38 (Elapsed s) 741.04
Final results are saved in
    /tmp/hdlsrc/sfir_fixed/hdlexpl/Final-07-Jan-2014-17-07-30
Validation model: gm_sfir_fixed_vnl
```

Then coder stops after one additional iteration because it has achieved the target clock frequency. The critical path is 9.55 ns, or a clock frequency of 104.71 MHz.

See Also

`hdlcoder.optimizeDesign`

Function Reference for HDL Code Generation from MATLAB

codegen

Purpose

Generate HDL code from MATLAB code

Syntax

```
codegen -config hdlcfg matlab_design_name
codegen -config hdlcfg -float2fixed
fixptcfg matlab_design_name
```

Description

`codegen -config hdlcfg matlab_design_name` generates HDL code from MATLAB code.

`codegen -config hdlcfg -float2fixed fixptcfg matlab_design_name` converts floating-point MATLAB code to fixed-point code, then generates HDL code.

Input Arguments

hdlcfg - HDL code generation configuration
`coder.HdlConfig`

HDL code generation configuration options, specified as a `coder.HdlConfig` object.

Create a `coder.HdlConfig` object using the HDL `coder.config` function.

matlab_design_name - MATLAB design function name
string

Name of top-level MATLAB function for which you want to generate HDL code.

fixptcfg - Floating-point to fixed-point conversion configuration
`coder.FixptConfig`

Floating-point to fixed-point conversion configuration options, specified as a `coder.FixptConfig` object.

Use `fixptcfg` when generating HDL code from floating-point MATLAB code. Create a `coder.FixptConfig` object using the HDL `coder.config` function.

Examples

Generate Verilog Code from MATLAB Code

Create a `coder.HdlConfig` object, `hdlcfg`.

```
hdlcfg = coder.config('hdl'); % Create a default 'hdl' config
```

Set the test bench name. In this example, the test bench function name is `mlhdlc_dti_tb`.

```
hdlcfg.TestBenchName = 'mlhdlc_dti_tb';
```

Set the target language to Verilog.

```
hdlcfg.TargetLanguage = 'Verilog';
```

Generate HDL code from your MATLAB design. In this example, the MATLAB design function name is `mlhdlc_dti`.

```
codegen -config hdlcfg mlhdlc_dti
```

Generate HDL Code from Floating-Point MATLAB Code

Create a `coder.FixptConfig` object, `fixptcfg`, with default settings.

```
fixptcfg = coder.config('fixpt');
```

Set the test bench name. In this example, the test bench function name is `mlhdlc_dti_tb`.

```
fixptcfg.TestBenchName = 'mlhdlc_dti_tb';
```

Create a `coder.HdlConfig` object, `hdlcfg`, with default settings.

```
hdlcfg = coder.config('hdl');
```

Convert your floating-point MATLAB design to fixed-point, and generate HDL code. In this example, the MATLAB design function name is `mlhdlc_dti`.

```
codegen -float2fixed fixptcfg -config hdlcfg mlhdlc_dti
```

See Also

`coder.FixptConfig` | `coder.HdlConfig` | `coder.config`

Related Examples

- “Generate HDL Code from MATLAB Code Using the Command Line Interface”

Purpose

Create HDL Coder code generation configuration objects

Syntax

```
config_obj = coder.config('hdl')
config_obj = coder.config('fixpt')
```

Description

`config_obj = coder.config('hdl')` creates a `coder.HdlConfig` configuration object for use with the HDL `codegen` function when generating HDL code from MATLAB code.

`config_obj = coder.config('fixpt')` creates a `coder.FixptConfig` configuration object for use with the HDL `codegen` function when generating HDL code from floating-point MATLAB code. The `coder.FixptConfig` object configures the floating-point to fixed-point conversion.

Examples

Generate HDL Code from Floating-Point MATLAB Code

Create a `coder.FixptConfig` object, `fixptcfg`, with default settings.

```
fixptcfg = coder.config('fixpt');
```

Set the test bench name. In this example, the test bench function name is `mlhdlc_dti_tb`.

```
fixptcfg.TestBenchName = 'mlhdlc_dti_tb';
```

Create a `coder.HdlConfig` object, `hdlcfg`, with default settings.

```
hdlcfg = coder.config('hdl');
```

Convert your floating-point MATLAB design to fixed-point, and generate HDL code. In this example, the MATLAB design function name is `mlhdlc_dti`.

```
codegen -float2fixed fixptcfg -config hdlcfg mlhdlc_dti
```

See Also

`coder.HdlConfig` | `coder.FixptConfig` | `codegen`

Related Examples

- “Generate HDL Code from MATLAB Code Using the Command Line Interface”

coder.FixptConfig.addFunctionReplacement

Purpose

Replace floating-point function with fixed-point function during fixed-point conversion

Syntax

`fxptcfg.addFunctionReplacement(floatFn,fixedFn)`

Description

`fxptcfg.addFunctionReplacement(floatFn,fixedFn)` specifies a function replacement in a `coder.FixptConfig` object. During floating-point to fixed-point conversion in the HDL code generation workflow, the code generation software replaces the specified floating-point function with the specified fixed-point function. The fixed-point function must be in the same folder as the floating-point function or on the MATLAB path.

Input Arguments

floatFn - Name of floating-point function

`''` (default) | string

Name of floating-point function, specified as a string.

fixedFn - Name of fixed-point function

`''` (default) | string

Name of fixed-point function, specified as a string.

Examples

Specify Function Replacement in Fixed-Point Conversion Configuration Object

Create a fixed-point code configuration object, `fxpCfg`, with a test bench, `myTestbenchName`.

```
fxpCfg = coder.config('fixpt');
fxpCfg.TestBenchName = 'myTestbenchName';
fxpCfg.addFunctionReplacement('min', 'fi_min');
codegen -float2fixed fxpCfg designName
```

Specify that the floating-point function, `min`, should be replaced with the fixed-point function, `fi_min`.

```
fxpCfg.addFunctionReplacement('min', 'fi_min');
```

When you generate code, the code generation software replaces instances of `min` with `fi_min` during floating-point to fixed-point conversion.

Alternatives

You can specify function replacements in the HDL Workflow Advisor. See “Function Replacements”.

See Also

`coder.FixptConfig` | `coder.config` | `codegen`

Class Reference for HDL Code Generation from MATLAB

coder.FixptConfig

Purpose

`codegen` floating-point to fixed-point conversion configuration object

Description

A `coder.FixptConfig` object contains the configuration parameters that the HDL `codegen` function requires to convert floating-point MATLAB code to fixed-point MATLAB code during HDL code generation. Use the `-float2fixed` option to pass this object to the `codegen` function.

Construction

`*fixptcfg* = coder.config('fixpt')` creates a `coder.FixptConfig` object for floating-point to fixed-point conversion during HDL code generation.

Properties

TestBenchName

Test bench function name, specified as a string. You must specify a test bench.

Values: `''` (default) | string

FixPtFileNameSuffix

Suffix for fixed-point file names.

Values: `'_fixpt'` | string

ProposeFractionLengthsForDefaultWordLength

Propose fixed-point types based on `DefaultWordLength`.

Values: `true` (default) | `false`

DefaultWordLength

Default fixed-point word length.

Values: 14 (default) | positive integer

ProposeWordLengthsForDefaultFractionLength

Propose fixed-point types based on `DefaultFractionLength`.

Values: `false` (default) | `true`

DefaultFractionLength

Default fixed-point fraction length.

Values: 4 (default) | positive integer

SafetyMargin

Safety margin percentage by which to increase the simulation range when proposing fixed-point types.

Values: 4 (default) | positive integer

LaunchNumericTypesReport

View the numeric types report after the coder has proposed fixed-point types.

Values: `true` (default) | `false`

LogIOForComparisonPlotting

Enable simulation data logging to plot the data differences introduced by fixed-point conversion.

Values: `true` (default) | `false`

DetectFixptOverflows

Enable detection of overflows using scaled doubles.

Values: `true`| `false` (default)

Methods

`addFunctionReplacement`	Replace floating-point function with fixed-point function during fixed-point conversion

Examples

Generate HDL Code from Floating-Point MATLAB Code

Create a `coder.FixptConfig` object, `fixptcfg`, with default settings.

```
fixptcfg = coder.config('fixpt');
```

Set the test bench name. In this example, the test bench function name is `mlhdlc_dti_tb`.

```
fixptcfg.TestBenchName = 'mlhdlc_dti_tb';
```

Create a `coder.HdlConfig` object, `hdlcfg`, with default settings.

```
hdlcfg = coder.config('hdl');
```

Convert your floating-point MATLAB design to fixed-point, and generate HDL code. In this example, the MATLAB design function name is `mlhdlc_dti`.

```
codegen -float2fixed fixptcfg -config hdlcfg mlhdlc_dti
```

Alternatives

You can also generate HDL code from MATLAB code using the HDL Workflow Advisor. For more information, see "HDL Code Generation from a MATLAB Algorithm".

See Also

`coder.HdlConfig` | `coder.config` | `codegen`

Related Examples

- "Generate HDL Code from MATLAB Code Using the Command Line Interface"

Purpose

HDL `codegen` configuration object

Description

A `coder.HdlConfig` object contains the configuration parameters that the HDL `codegen` function requires to generate HDL code. Use the `-config` option to pass this object to the `codegen` function.

Construction

`*hdlcfg* = coder.config('hdl')` creates a `coder.HdlConfig` object for HDL code generation.

Properties

Basic

GenerateHDLTestBench

Generate an HDL test bench, specified as a `logical`.

Values: `false` (default) | `true`

HDLCodingStandard

HDL coding standard to follow and check when generating code, specified as a string. Generates a compliance report showing errors, warnings, and messages.

Values: `'None'` (default) | `'Industry'`

HDLLintTool

HDL lint tool script to generate, specified as a string.

Values: `'None'` (default) | `'AscentLint'` | `'Leda'` | `'SpyGlass'` |`'Custom'`

HDLLintInit

HDL lint script initialization string.

Value: string

HDLLintCmd

HDL lint script command.

If you set `HDLLintTool` to `Custom`, you must use `%s` as a placeholder for the HDL file name in the generated Tcl script. Specify `HDLLintCmd` using the following format:

```
custom_lint_tool_command -option1 -option2 %s
```

Value: string

HDLLintTerm

HDL lint script termination string.

Value: string

InitializeBlockRAM

Specify whether to initialize all block RAM to `'0'` for simulation.

Values: `true` (default) | `false`

InlineConfigurations

Specify whether to include inline configurations in generated VHDL code.

When `true`, include VHDL configurations in files that instantiate a component.

When `false`, suppress the generation of configurations and require user-supplied external configurations. Set to `false` if you are creating your own VHDL configuration files.

Values: `true` (default) | `false`

ClockEdge

Specify active clock edge.

Values: `'Rising'` (default) | `'Falling'`

SimulateGeneratedCode

Simulate generated code, specified as a `logical`.

Values: `false` (default) | `true`

PartitionFunctions

Generate instantiable HDL code modules from functions.

Values: `false` (default) | `true`

SimulationIterationLimit

Maximum number of simulation iterations during test bench generation, specified as an integer. This property affects only test bench generation, not simulation during fixed-point conversion.

Values: unlimited (default) | positive integer

SimulationTool

Simulation tool name, specified as a string.

Values: `'ModelSim'` (default) | `'ISIM'`

SynthesisTool

Synthesis tool name, specified as a string.

Values: `'Xilinx ISE'` (default) | `'Altera Quartus II'`

SynthesisToolChipFamily

Synthesis target chip family name, specified as a string.

Values: `'Virtex4'` (default) | string

SynthesisToolDeviceName

Synthesis target device name, specified as a string.

Values: `'xc4vsx35'` (default) | string

SynthesisToolPackageName

Synthesis target package name, specified as a string.

Values: `'ff668'` (default) | string

SynthesisToolSpeedValue

Synthesis target speed, specified as a string.

Values: `'-10'` (default) | string

SynthesizeGeneratedCode

Synthesize generated code or not, specified as a `logical`.

Values: `false` (default) | `true`

TargetLanguage

Target language, specified as a string.

Values: `'VHDL'` (default) | `'Verilog'`

TestBenchName

Test bench function name, specified as a string. You must specify a test bench.

Values: `''` (default) | string

UseFileIOInTestBench

Create and use data files for reading and writing test bench input and output data.

Values: `'off'` (default) | `'on'`

DistributedPipeliningPriority

Priority for distributed pipelining algorithm, specified as a string.

`DistributedPipeliningPriority` **Value**	**Description**
`Numerical Integrity` (default)	Prioritize numerical integrity when distributing pipeline registers. This option uses a conservative retiming algorithm that does not move registers across a component if the functional

`DistributedPipeliningPriority` Value	Description
	equivalence to the original design is unknown.
`Performance`	Prioritize performance over numerical integrity. Use this option if your design requires a higher clock frequency and the MATLAB behavior does not need to strictly match the generated code behavior. This option uses a more aggressive retiming algorithm that moves registers across a component even if the modified design's functional equivalence to the original design is unknown.

Values: `'NumericalIntegrity'` (default) | `'Performance'`

PreserveDesignDelays

Prevent distributed pipelining from moving design delays or allow distributed pipelining to move design delays, specified as a `logical`.

Persistent variables and `dsp.Delay` System objects are design delays.

Values: `false` (default) | `true`

Cosimulation

GenerateCosimTestBench

Generate a cosimulation test bench or not, specified as a `logical`.

Values: `false` (default) | `true`

SimulateCosimTestBench

Simulate generated cosimulation test bench, specified as a `logical`. This option is ignored if `GenerateCosimTestBench` is `false`.

Values: `false` (default) | `true`

CosimClockEnableDelay

Time (in clock cycles) between deassertion of reset and assertion of clock enable.

Values: `0` (default)

CosimClockHighTime

The number of nanoseconds the clock is high.

Values: `5` (default)

CosimClockLowTime

The number of nanoseconds the clock is low.

Values: `5` (default)

CosimHoldTime

The hold time for input signals and forced reset signals, specified in nanoseconds.

Values: `2` (default)

CosimLogOutput

Log and plot outputs of the reference design function and HDL simulator.

Values: `false` (default) | `true`

CosimResetLength

Specify time (in clock cycles) between assertion and deassertion of reset.

Values: `2` (default)

CosimRunMode

HDL simulator run mode during simulation, specified as a string. When in Batch mode, you do not see the HDL simulator GUI, and the HDL simulator automatically shuts down after simulation.

Values: `Batch` (default) | `GUI`

CosimTool

HDL simulator for the generated cosim test bench, specified as a string.

Values: `ModelSim` (default) | `Incisive`

FPGA-in-the-loop

GenerateFILTestBench

Generate a FIL test bench or not, specified as a `logical`.

Values: `false` (default) | `true`

SimulateFILTestBench

Simulate generated cosimulation test bench, specified as a `logical`. This option is ignored if `GenerateCosimTestBench` is `false`.

Values: `false` (default) | `true`

FILBoardName

FPGA board name, specified as a string. You must override the default value and specify a valid board name.

Values: `'Choose a board'` (default) | string

FILBoardIPAddress

IP address of the FPGA board, specified as a string. You must enter a valid IP address.

Values: `192.168.0.2` (default) | string

FILBoardMACAddress

MAC address of the FPGA board, specified as a string. You must enter a valid MAC address.

Values: `00-0A-35-02-21-8A` (default) | string

FILAdditionalFiles

List of additional source files to include, specified as a string. Separate file names with a semi-colon (";").

Values: `''` (default) | string

FILLogOutputs

Log and plot outputs of the reference design function and FPGA.

Values: `false` (default) | `true`

Examples

Generate Verilog Code from MATLAB Code

Create a `coder.HdlConfig` object, `hdlcfg`.

```
hdlcfg = coder.config('hdl'); % Create a default 'hdl' config
```

Set the test bench name. In this example, the test bench function name is `mlhdlc_dti_tb`.

```
hdlcfg.TestBenchName = 'mlhdlc_dti_tb';
```

Set the target language to Verilog.

```
hdlcfg.TargetLanguage = 'Verilog';
```

Generate HDL code from your MATLAB design. In this example, the MATLAB design function name is `mlhdlc_dti`.

```
codegen -config hdlcfg mlhdlc_dti
```

Generate Cosim and FIL Test Benches

Create a `coder.FixptConfig` object with default settings and provide test bench name.

```
fixptcfg = coder.config('fixpt');
fixptcfg.TestBenchName = 'mlhdlc_sfir_tb';
```

Create a `coder.HdlConfig` object with default settings and set enable rate.

```
hdlcfg = coder.config('hdl'); % Create a default 'hdl' config
hdlcfg.EnableRate = 'DUTBaseRate';
```

Instruct MATLAB to generate a cosim test bench and a FIL test bench. Specify FPGA board name.

```
hdlcfg.GenerateCosimTestBench = true;
hdlcfg.FILBoardName = 'Xilinx Virtex-5 XUPV5-LX110T development board
hdlcfg.GenerateFILTestBench = true;
```

Perform code generation, Cosim test bench generation, and FIL test bench generation.

```
codegen -float2fixed fixptcfg -config hdlcfg mlhdlc_sfir
```

Alternatives

You can also generate HDL code from MATLAB code using the HDL Workflow Advisor. For more information, see “HDL Code Generation from a MATLAB Algorithm”.

See Also

`coder.FixptConfig` | `coder.config` | `codegen`

Related Examples

- “Generate HDL Code from MATLAB Code Using the Command Line Interface”