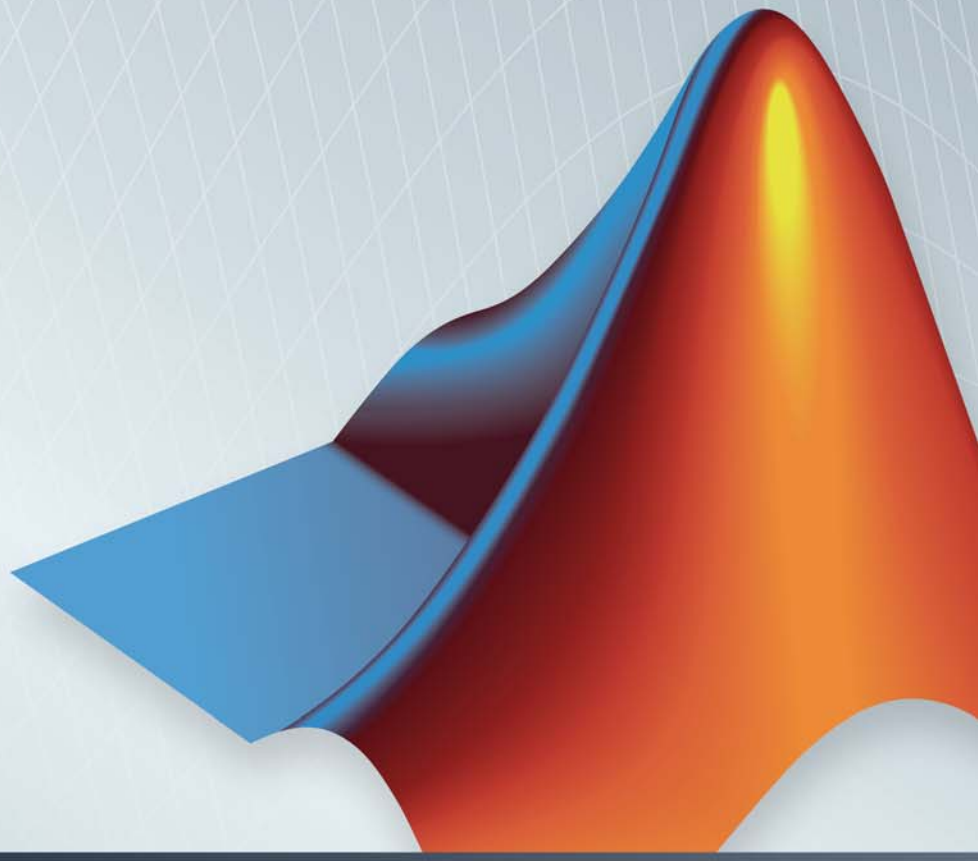
HDL Coder™

Reference

R2013**b**



MATLAB®

MathWorks®

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support

`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

HDL Coder™ Reference

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

March 2013	Online only	New for Version 3.2 (R2013a)
September 2013	Online only	Revised for Version 3.3 (R2013b)

Contents

1

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. For example, the presence of an unsupported block in the model raises a warning. In this case, the coder attempts to proceed as if the block were not present in the design. This might lead to errors later on in the code generation process.
- *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

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”

hdldispblkparams

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 “Block Implementation Parameters” 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`

hdllib

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

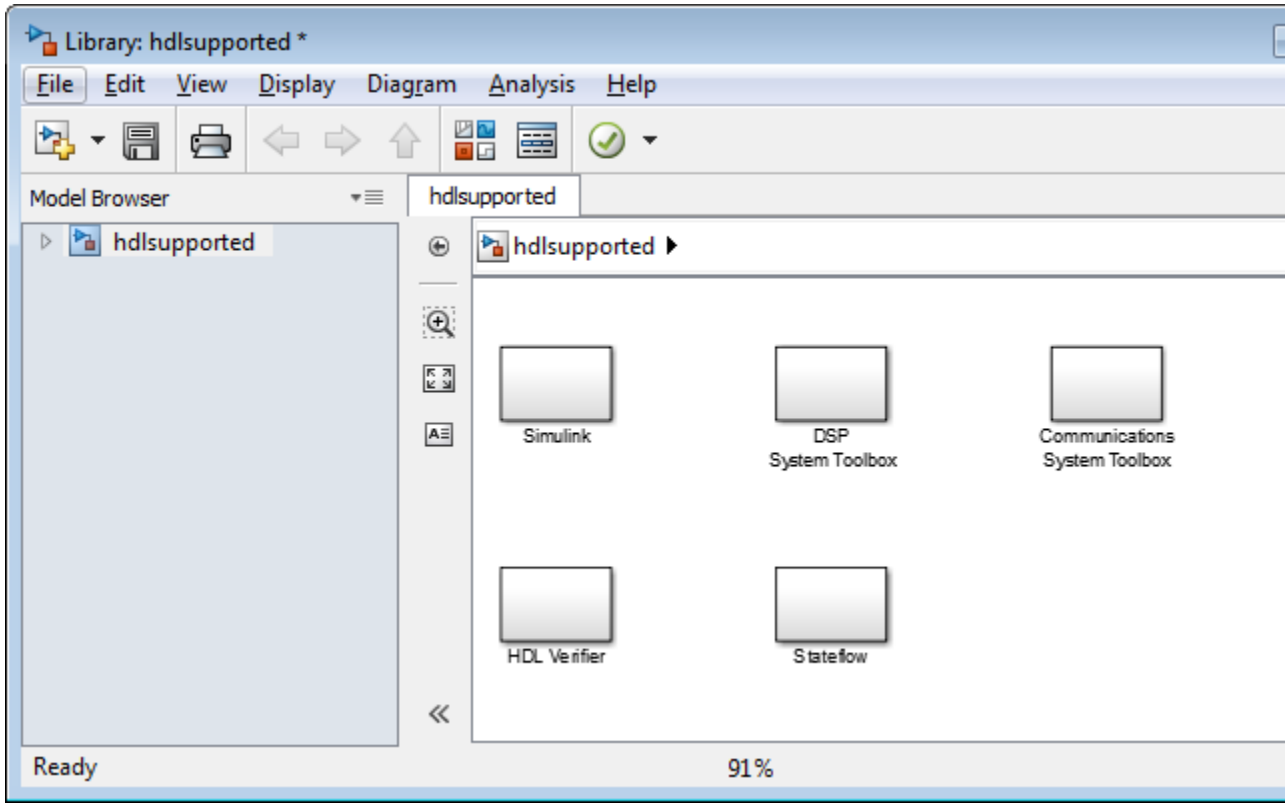
Create a supported blocks library and HTML reports

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

"Blocks Supported for HDL Code Generation" | "Block Implementation Parameters" | "Open the hdldemolib Library" |

hdlnewblackbox

Purpose

Generate customizable control file from selected subsystem or blocks

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;
```

hdlnewcontrol

Purpose

Construct code generation control object for use in control file

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

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'});
```

Purpose

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

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 “Block Implementation Parameters” 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`

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`

hdlset_param

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 "Block Implementation Parameters" 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"

hdlsetup

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`.

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`.

'HierarchicalDistPipelining' - Hierarchical distributed pipelining

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

For more information, see `HierarchicalDistPipelining`.

'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` function to clock registers

`'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

'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`.

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) | `'SpyGlass'` | `'Leda'` | `'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

'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
### Begin Verilog Code Generation for 'sfir_fixed'.
### Working on sfir_fixed/symmetric_fir as hdlsrc\sfir_fixed\symmetric_fi
### 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
### Begin VHDL Code Generation for 'sfir_fixed'.
### Working on sfir_fixed/symmetric_fir as hdlsrc\sfir_fixed\symmetric_fi
### 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
### Begin Verilog Code Generation for 'sfir_fixed'.
### Working on sfir_fixed/symmetric_fir as hdlsrc\sfir_fixed\symmetric
### 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

Start Support Package Installer and install support for third-party hardware or software

Syntax

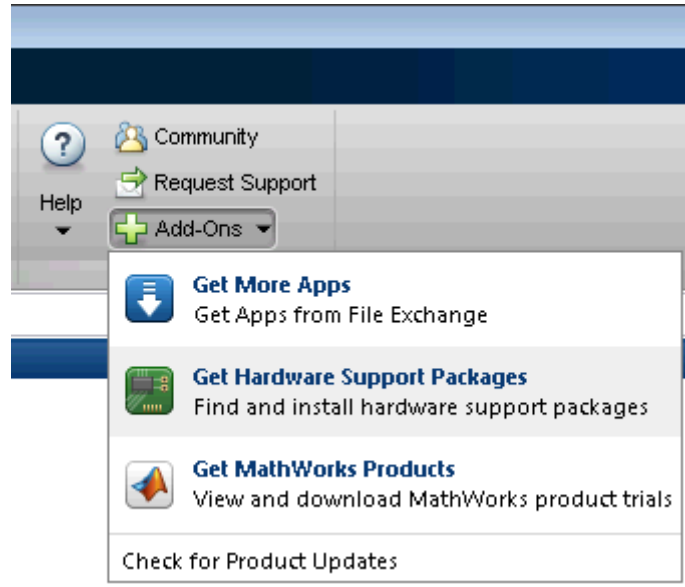
```
supportPackageInstaller
```

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**.



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

2

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 set this property using `hdlset_param` or `makehdl`.

You can further control delay balancing within the model by disabling or enabling delay balancing for subsystems within the model.

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.

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.

See Also

`checkhdl`, `makehdl`

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.

See Also

`ClockInputPort`, `InputType`, `OutputType`, `ResetInputPort`

ClockEnableOutputPort property

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.

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'`.

See Also

`ClockLowTime`, `ForceClock`, `ForceClockEnable`, `ForceReset`, `HoldTime`

ClockInputs property

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.

Example

The following example specifies the generation of multiple clocks.

```
makehdl(gcb, 'ClockInputs','Multiple');
```

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.

See Also

`ClockEnableInputPort`, `InputType`, `OutputType`

ClockLowTime property

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'`.

See Also

`ClockHighTime`, `ForceClock`, `ForceClockEnable`, `ForceReset`, `HoldTime`

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
  .
  .
  .
```

See Also

`PackagePostfix`, `ReservedWordPostfix`

CodeGenerationOutput property

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.

See Also

"Defaults and Options for Generated Models"

Purpose

Specify string to append to imaginary part of complex signal names

Settings

`'string'`

Default: `'_im'`.

See Also

`ComplexRealPostfix`

ComplexRealPostfix property

Purpose

Specify string to append to real part of complex signal names

Settings

`'string'`

Default: `'_re'`.

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.

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.

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;
```

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 `MyFilt`, the coder names the first entity `MyFilt` and the second instance `MyFilt_block`.

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.

See Also

`ClockLowTime`, `ClockHighTime`, `ForceClockEnable`, `ForceReset`, `HoldTime`

ForceClockEnable property

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.

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.

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.

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.

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 `Generatemodelname` to override the default.

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.

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.

See Also

“Generate HDL Code Using the Configuration Parameters Dialog Box”

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

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.

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.

See Also

"Generate Reusable Code for Atomic Subsystems"

HDLCodingStandard property

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.

Purpose

Specify string written to initialization section of compilation script

Settings

`'string'`

Default: `'vlib work\n'`.

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 (`''`).

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`.

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).

See Also

"Generate Scripts for Compilation, Simulation, and Synthesis"

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).

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`.

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.

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']
```

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`.

Purpose

Specify string written to termination section of simulation script

Settings

`'string'`

Default: `'run -all\n'`.

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.

See Also

"Generate Scripts for Compilation, Simulation, and Synthesis"

Purpose

Specify command written to HDL lint script

Settings

`'string'`

Default: none.

Specify the HDL lint tool command.

See Also

`HDLLintTool`, `HDLLintInit`, `HDLLintTerm`, "Generate an HDL Lint Tool Script"

HDLLintInit property

Purpose

Specify HDL lint script initialization string

Settings

`'string'`

Default: none.

Specify the HDL lint script initialization string.

See Also

`HDLLintTool`, `HDLLintCmd`, `HDLLintTerm`, "Generate an HDL Lint Tool Script"

Purpose

Specify HDL lint script termination string

Settings

`'string'`

Default: none.

Specify the HDL lint script termination string.

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.
`'SpyGlass'`	Atrenta Spyglass
`'Leda'`	Synopsys® Leda
`'Custom'`	A custom lint tool.

See Also

`HDLLintInit`, `HDLLintCmd`, `HDLLintTerm`, "Generate an HDL Lint Tool Script"

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 `Cmd` 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.

See Also

`HDLSynthTool`, `HDLSynthInit`, `HDLSynthTerm`, `HDLSynthFilePostfix`, "Generate Scripts for Compilation, Simulation, and Synthesis"

HDLSynthFilePostfix property

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`.

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 `Init` 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.

See Also

`HDLSynthTool`, `HDLSynthCmd`, `HDLSynthTerm`, `HDLSynthFilePostfix`, "Generate Scripts for Compilation, Simulation, and Synthesis"

HDLSynthTerm property

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 `Term` section of the synthesis script. The `Term` section does not take arguments. The content of the string is specific to the selected synthesis tool.

See Also

`HDLSynthTool`, `HDLSynthCmd`, `HDLSynthInit`, `HDLSynthFilePostfix`, "Generate Scripts for Compilation, Simulation, and Synthesis"

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'`

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.

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.

`'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.

See Also

"Defaults and Options for Generated Models"

HighlightColor property

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'`

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. (`N` is `>= 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.

See Also

`HoldTime`, “Code Generation from Multirate Models”

HoldTime property

Purpose

Specify hold time for input signals and forced reset input signals

Settings

ns

Default: 2

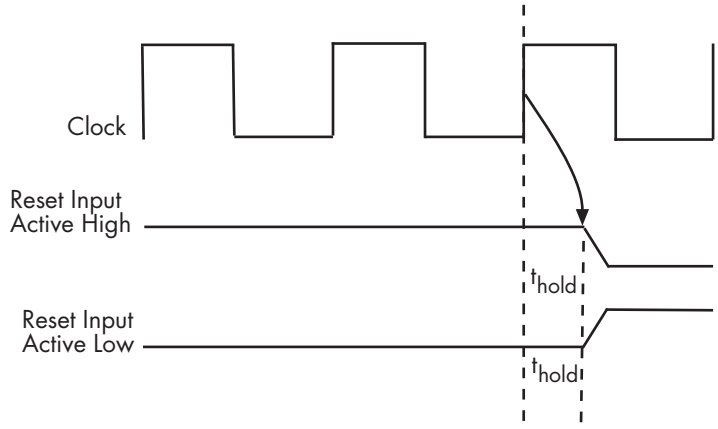
Specify the number of nanoseconds during which the model's data input signals and forced reset input signals are held past the clock rising edge.

The hold time is expressed as a positive integer.

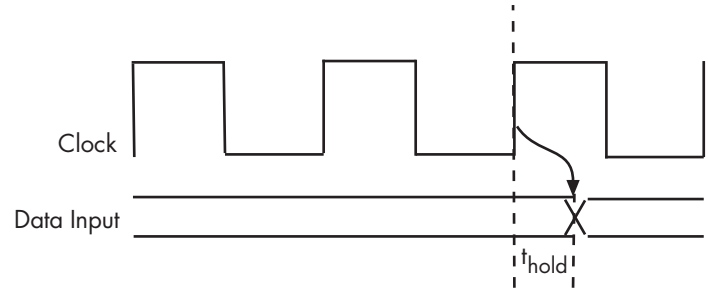
This option applies to reset input signals only if forced resets are enabled.

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}.

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 specify the `'DistributedPipelining','on'` parameter for the MATLAB Function block (see "Distributed Pipeline Insertion for MATLAB Function Blocks").
- When you specify the `'ResetType','None'` parameter (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.

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 `hdldemolib` library (see also "RAM Blocks"). The library provides three type of RAM blocks:

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

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).

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.

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.

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');
```

InputType property

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.

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.

See Also

`BlockGenerateLabel`, `OutputGenerateLabel`

InstancePostfix property

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.

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.

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.

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.

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.

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.

RAMArchitecture property

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.

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 2-73). 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

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.

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.

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`.

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.

See Also

`TestBenchPostFix`, `TestBenchDataPostFix`

Purpose

Display HTML optimization report

Settings

`'on'`

Create and display an HTML optimization report.

`'off'` (default)

Do not create an HTML optimization report.

See Also

"Create and Use Code Generation Reports"

OptimizeTimingController property

Purpose

Optimize timing controller entity for speed and code size 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.

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.

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.

See Also

`ClockEnableInputPort`, `InputType`

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.

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.

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;
```

See Also

“Block Implementation Parameters”, “InputPipeline”, “OutputPipeline”

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 RAMs instead of registers. This threshold applies to:

- Delay blocks
- Persistent variables in MATLAB Function blocks

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

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`.

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'));
.
.
.
```

See Also

`ResetType`

ResetInputPort property

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.

See Also

`ClockEnableInputPort`, `InputType`, `OutputType`

Purpose

Define length of time (in clock cycles) during which reset is asserted

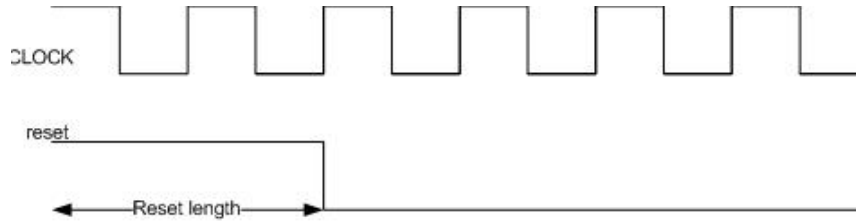
Settings

`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.



ResetType property

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;
```

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.

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.

See Also

`LoopUnrolling`, `UseAggregatesForConst`, `UseRisingEdge`

ScalarizePorts property

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;
```

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.

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.

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.

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.

See Also

`SplitArchFilePostfix`, `SplitEntityArch`

TargetDirectory property

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.

See Also

`VerilogFileExtension`, `VHDLFileExtension`

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

TestBenchClockEnableDelay property

Purpose

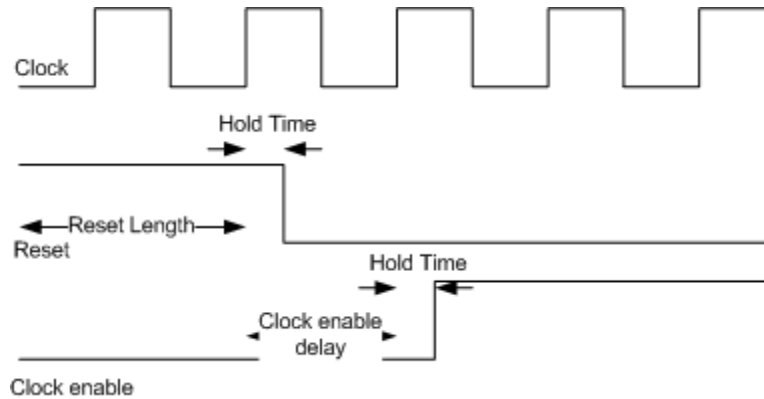
Define elapsed time in clock cycles between deassertion of reset and assertion of clock enable

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`.



See Also

`HoldTime`, `ResetLength`

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 set `'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`.

See Also

`MultifileTestBench`, `TestBenchPostFix`

TestBenchPostFix property

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`.

See Also

`MultifileTestBench`, `TestBenchDataPostFix`

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`.

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.

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.

See Also

"Create and Use Code Generation Reports", "Generate Code with Annotations or Comments", `RequirementComments`

UseAggregatesForConst property

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);
```

See Also

`LoopUnrolling`, `SafeZeroConcat`, `UseRisingEdge`

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.

UserComment property

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.
--
-- -------------------------------------------------------------
```

Purpose

Specify VHDL coding style used to check for rising edges when operating on registers

Settings

`'on'`

Selected

Use the VHDL `rising_edge` function to check for rising edges when operating on registers. The following code, generated from a Unit Delay block, tests `rising_edge` as shown in the following `PROCESS` block:

```
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)

Check for clock events when operating on registers. The following code, generated from a Unit Delay block, checks for a clock event as shown in the `ELSIF` statement of the following `PROCESS` block:

```
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;
```

Usage Notes

The two coding styles have different simulation behavior when the clock transitions from `'X'` to `'1'`.

See Also

`LoopUnrolling`, `SafeZeroConcat`, `UseAggregatesForConst`

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.

See Also

`LoopUnrolling`, `SafeZeroConcat`, `UseAggregatesForConst`, `UseRisingEdge`

VectorPrefix property

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.

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.

VerilogFileExtension property

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`

Purpose

Specify architecture name for generated HDL code

Settings

`'string'`

The default architecture name is `'rtl'`.

VHDLFileExtension property

Purpose

Specify file type extension for generated VHDL files

Settings

`'string'`

The default file type extension for generated VHDL files is `.vhd`.

See Also

`TargetLanguage`

Purpose

Specify name of target library for generated HDL code

Settings

`'string'`

The default target library name is `'work'`.

See Also

`HDLCompileInit`

VHDLLibraryName

3

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_nam
   e
```

Description

`codegen -config hdlcfg matlab_design_name` generates HDL code from MATLAB code.

`codegen -config hdlcfg -float2fixed fixptcfg matlab_design_nam e` 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 an 'hdl' config with default se
```

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"

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.

Specify Function Replacement in Fixed-Point Conversion Configuration Object

Suppose that:

- The function `myfunc` calls a local function `myadd`.
- The test function `mytest` calls `myfunc`.
- You want to replace calls to `myadd` with the fixed-point function `fi_myadd`.

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 `mytest`.

```
fixptcfg.TestBenchName = 'mytest';
```

Specify that the floating-point function, `myadd`, should be replaced with the fixed-point function, `fi_myadd`.

```
fixptcfg.addFunctionReplacement('myadd', 'fi_myadd');
```

Create a code generation configuration object to generate a standalone C static library.

```
cfg = coder.config('lib');
```

Convert the floating-point MATLAB function, `myfunc`, to fixed-point, and generate C code.

```
codegen -float2fixed fixptcfg -config cfg myfunc
```

When you generate code, the code generation software replaces instances of `myadd` with `fi_myadd` 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` | `codegencoder.FixptConfig` | | `coder.config` | `codegen`

4

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.

A `coder.FixptConfig` object contains the configuration parameters that the MATLAB Coder™ `codegen` function requires to convert floating-point MATLAB code to fixed-point MATLAB code during 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`

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
```

Generate Fixed-Point C 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 `dti_test`.

```
fixptcfg.TestBenchName = 'dti_test';
```

Create a code generation configuration object to generate a standalone C static library.

```
cfg = coder.config('lib');
```

Convert a floating-point MATLAB function to fixed-point C code. In this example, the MATLAB function name is `dti`.

```
codegen -float2fixed fixptcfg -config cfg 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”.

You can convert floating-point MATLAB code to fixed-point code using the MATLAB Coder app. Open the app using one of these methods:

- On the **Apps** tab, in the **Code Generation** section, click **MATLAB Coder**.
- Use the `coder` command.

See "Convert MATLAB Code to Fixed-Point C Code".

See Also

`coder.HdlConfig` | `coder.config` | `codegencoder.codeConfig` | | `coder.config` | `codegen` | `coder`

Related Examples

- "Generate HDL Code from MATLAB Code Using the Command Line Interface"
- "C Code Generation at the Command Line"

coder.HdlConfig

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) | `'SpyGlass'` | `'Leda'` |`'Custom'`

HDLLintInit

HDL lint script initialization string.

Value: string

HDLLintCmd

HDL lint script command.

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`

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'`

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 an 'hdl' config with default se
```

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 an 'hdl' config with default setti
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"