Filter Design HDL Coder Release Notes

These release notes describe the Version 1.3 release of the Filter Design HDL Coder. Topics include

- "New Features" on page 1-2
- "Major Bug Reports" on page 1-8
- "Known Software and Documentation Problems" on page 1-9

The Filter Design HDL Coder Release Notes also provide information about the earlier versions of the product, in case you are upgrading from an earlier version:

- Chapter 2, "Filter Design HDL Coder 1.2 Release Notes"
- Chapter 3, "Filter Design HDL Coder 1.1 Release Notes"
- Chapter 4, "Filter Design HDL Coder 1.0 Release Notes"

Contents

Filter Design HDL Coder 1.3 Release Notes

These release notes provide the following information:

New Features

Generating Scripts for EDA Tools

Filter Design HDL Coder now supports generation of script files for third-party Electronic Design Automation (EDA) tools. These scripts let you compile and simulate generated HDL code and/or synthesize generated HDL code.

Using the defaults, you can automatically generate scripts for the following tools:

- Mentor Graphics ModelSim SE/PE HDL simulator
- The Synplify family of synthesis tools

You can customize both the names and the content of generated script files. To do this, you must use the `generatehdl` or `generatehdltb` function, and pass in the appropriate property name /property value arguments as described in "Customizing Script File Names" on page 1-3 and "Customizing Script Code" on page 1-3.

Default Script Generation

Script generation takes place automatically, as part of the code and test bench generation process (whether initiated from the command line or from the Generate HDL dialog). All script files are generated in the target directory.

When HDL code is generated for a filter *Hd*, The Filter Design HDL Coder writes the following script files:

- *Hd*`_compile.do`: ModelSim compilation script. This script contains commands to compile the generated filter code, but not to simulate it.
- *Hd*`_synplify.tcl`: Synplify synthesis script

When test bench code is generated for a filter *Hd*, Filter Design HDL Coder writes the following script files:

- *Hd*`_tb_compile.do`: ModelSim compilation script. This script contains commands to compile the generated filter and test bench code.

- `Hd_tb_sim.do`: ModelSim simulation script. This script contains commands to run a simulation of the generated filter and test bench code.

Customizing Script File Names

When HDL code is generated, script names are generated by appending a postfix string to the filter name *`Hd`*.

When test bench code is generated, script names are generated by appending a postfix string to the test bench name *`testbench_tb`*.

The postfix string depends on the type of script (compilation, simulation, or synthesis) being generated. The default postfix strings are shown in the table below. For each type of script, you can define your own postfix using the associated property.

Script Type	Property	Default Value
Compilation	`'HDLCompileFilePostfix'`	`'_compile.do'`
Simulation	`'HDLSimFilePostfix'`	`'_sim.do'`
Synthesis	`'HDLSynthFilePostfix'`	`'_synplify.tcl'`

In the following example, VHDL code is generated for the filter object `myfilt`. A custom postfix string is specified for the compilation script. The name of the generated compilation script will be `myfilt_test_compilation.do`.

```
generatehdl(myfilt, 'HDLCompileFilePostfix', '_test_compilation.do')
```

Customizing Script Code

A generated EDA script consists of three sections, which are generated and executed in the following order:

1. An initialization (`Init`) phase. The `Init` phase performs any required setup actions, such as creating a design library or a project file. Some arguments to the `Init` phase are implicit, for example the top-level entity or module name.

   Properties that apply to the `Init` phase are identified by the substring `Init` in the property name.

2 A command-per-file phase (`Cmd`). This phase of the script is called iteratively, once per generated HDL file or once per signal. On each call, a different file or signal name is passed in.

   Properties that apply to the `Cmd` phase are identified by the substring `Cmd` in the property name.

3 A termination phase (`Term`). This is the final execution phase of the script. One application of this phase is to execute a simulation of HDL code that was compiled in the `Cmd` phase. The `Term` phase takes no arguments.

   Properties that apply to the `Term` phase are identified by the substring `Term` in the property name.

`generatehdl` and `generatehdltb` generate scripts by passing format strings to the MATLAB `fprintf` function. Using the property name/property value pairs summarized in the table below, you can pass in customized format strings to `generatehdl` or `generatehdltb`.

You can use any legal `fprintf` formatting characters. For example, `'\n'` inserts a newline into the script file.

Some of these format strings can take arguments, such as the top-level entity or module name, or the names of the VHDL or Verilog files in the design. The `'HDLSimViewWaveCommand'` format string takes the top-level signal names as its argument.

Property Name and Default	Description
Name: `'HDLCompileInit'` Default:`'vlib work\n'`	Format string passed to `fprintf` to write the `Init` section of the compilation script.
Name: `'HDLCompileVHDLCmd'` Default: `'vcom %s %s\n'`	Format string passed to `fprintf` to write the `Cmd` section of the compilation script for VHDL files. The two arguments are the contents of the `'SimulatorFlags'` property and the filename of the current entity or module. To omit the flags, set `'SimulatorFlags'` to `''` (the default).

Property Name and Default	Description
Name: `'HDLCompileVerilogCmd'` Default: `'vlog %s %s\n'`	Format string passed to `fprintf` to write the `Cmd` section of the compilation script for Verilog files. The two arguments are the contents of the `'SimulatorFlags'` property and the filename of the current entity or module. To omit the flags, set `'SimulatorFlags'` to `''` (the default).
Name:`'HDLCompileTerm'` Default:`''`	Format string passed to `fprintf` to write the termination portion of the compilation script.

Name: `'HDLSimInit'`
Default:
`['onbreak resume\n',...`
`'onerror resume\n']`
	Format string passed to `fprintf` to write the initialization portion of the simulation script.

Name: `'HDLSimCmd'` Default: `'vsim work.%s\n'`	Format string passed to `fprintf` to write the simulation command. The implicit argument is the top-level module or entity name.
Name: `'HDLSimViewWaveCmd'` Default: `'add wave sim:%s\n'`	Format string passed to `fprintf` to write the simulation command waveform viewing command. The top-level module or entity signal names are implicit arguments.
Name: `'HDLSimTerm'` Default: `'run -all\n'`	Format string passed to `fprintf` to write the `Term` portion of the simulation script

Property Name and Default	Description
Name: `'HDLSynthInit'` Default: `'project -new %s.prj\n'`	Format string passed to `fprintf` to write the 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.
Name: `'HDLSynthCmd'` Default: `'add_file %s\n'`	Format string passed to `fprintf` to write the `Cmd` section of the synthesis script. The argument is the filename of the entity or module.

Name: `'HDLSynthTerm'`
Default:
`['set_option -technology VIRTEX2\n',...`
`'set_option -part XC2V500\n',...`
`'set_option -synthesis_onoff_pragma 0\n',...`
`'set_option -frequency auto\n',...`
`'project -run synthesis\n']`
	Format string passed to `fprintf` to write the `Term` portion of the synthesis script.

Example. The following example specifies a ModelSim command for the `Init` phase of a compilation script for VHDL code generated from the filter `myfilt`.

```
generatehdl(myfilt, 'HDLCompileInit', 'vlib mydesignlib\n')
```

The resultant script, `myfilt_compile.do`, is shown below.

```
vlib mydesignlib
vcom  myfilt.vhd
```

Mixed- Language Scripts

The Filter Design HDL Coder allows most combinations of filter and test bench languages. For example, it is possible to generate VHDL filter code and a Verilog test bench file, as in the following commands:

```
generatehdl(myfilt, 'TargetLanguage', 'VHDL')
generatetb(myfilt, 'Verilog')
```

The listing below shows the generated test bench compilation script for the above case (`myfilt_tb_compile.do`). The script contains the correct language-specific compile command for the generated filter and test bench code.

```
vlib work
vcom  myfilt.vhd
vlog  myfilt_tb.v
```

Note that there are two simulation compile `Cmd` properties (`'HDLCompileVHDLCmd'`, `'HDLCompileVerilogCmd'`) allowing you to customize the compilation command for each supported target language.

Note that you can specify generation of *both* VHDL and Verilog test bench code, via the Generate HDL dialog. In this case, the test bench compilation script will default to the Verilog compilation command.

Note that the generation of ModelSim .do test bench files, which is controlled by the **ModelSim .do file** option, is independent from the generation of script files.

Test Bench Generation Improved for Multirate Filters

The speed of generation of large test bench files for multirate filters has been improved significantly for this release.

Major Bug Reports

To view major bug fixes made in R14SP3 for Filter Design HDL Coder, use the Bug Reports interface on the MathWorks Web site.

Note If you are not already logged in to Access Login, when you link to the Bug Reports interface (see below), you will be prompted to log in or create an Access Login account.

After you are logged in, use this Bug Fixes link. You will see the bug report for Filter Design HDL Coder. The report is sorted with fixed bugs listed first, and then open bugs.

If you are viewing these release notes in PDF form on the MathWorks Web site, you can refer to the HTML form of the release notes on the MathWorks Web site and use the link provided.

Known Software and Documentation Problems

To view important open bugs in R14SP3 for Filter Design HDL Coder, use the Bug Reports interface on the MathWorks Web site.

Note If you are not already logged in to Access Login, when you link to the Bug Reports interface (see below), you will be prompted to log in or create an Access Login account.

After you are logged in, use this Open Bugs link. You will see the bug report for Filter Design HDL Coder. The report is sorted with fixed bugs listed first, and then open bugs. You can select the **Status** column to list the open bugs first.

If you are viewing these release notes in PDF form on the MathWorks Web site, you can refer to the HTML form of the release notes on the MathWorks Web site and use the link provided.

Filter Design HDL Coder 1.2 Release Notes

These release notes provide the following information:

New Features

This section describes new features in the Filter Design HDL Coder 1.2.

Additional Multirate and Discrete Filter Types Supported

The Filter Design HDL Coder now adds code generation support for the following multirate and discrete filter types:

- Direct-Form FIR Polyphase Interpolator (`mfilt.firinterp`)
- Direct-Form FIR Polyphase Decimator (`mfilt.firdecim`)
- FIR Hold Interpolator (`mfilt.holdinterp`)
- FIR Linear Interpolator (`mfilt.linearinterp`)
- Discrete-Time Scalar (`dfilt.scalar`)

For a complete list of filter structures supported for code generation, see "Key Features and Components" in the Filter Design HDL Coder online documentation.

Code Generation Support for Interpolating Filters in Cascades

In the previous release, only decimators and/or single-rate filter structures could be included in a cascade for code generation purposes.

The Filter Design HDL Coder 1.2 now supports code generation for cascades that include interpolators. You can generate code for cascades that combine the following filter types:

- Decimators and/or single-rate filter structures
- Interpolators and/or single-rate filter structures

Code generation for cascades that include both decimators and interpolators is not currently supported, however.

See also “Generating Code for Cascade Filters” in the Filter Design HDL Coder online documentation.

Upgrading from an Earlier Release

This section describes issues involved in upgrading from the Filter Design HDL Coder Version 1.1 to Version 1.2.

InitializeRealSignals Property and GUI Option Removed

The Filter Design HDL Coder Version 1.2 always initializes signals of type `REAL` with a value of `0.0`.

In previous releases, initialization code for real signals was generated optionally. Generation of such initialization code was controlled by the `InitializeRealSignals` property and the corresponding **Initialize real signals** option in the **Advanced** pane of the **HDL Options** dialog box. The **Initialize real signals** option is no longer supported and has been removed from the **Advanced** pane. The `InitializeRealSignals` property is set to `'on'` and is no longer user settable.

Filter Design HDL Coder 1.1 Release Notes

These release notes provide the following information:

New Features

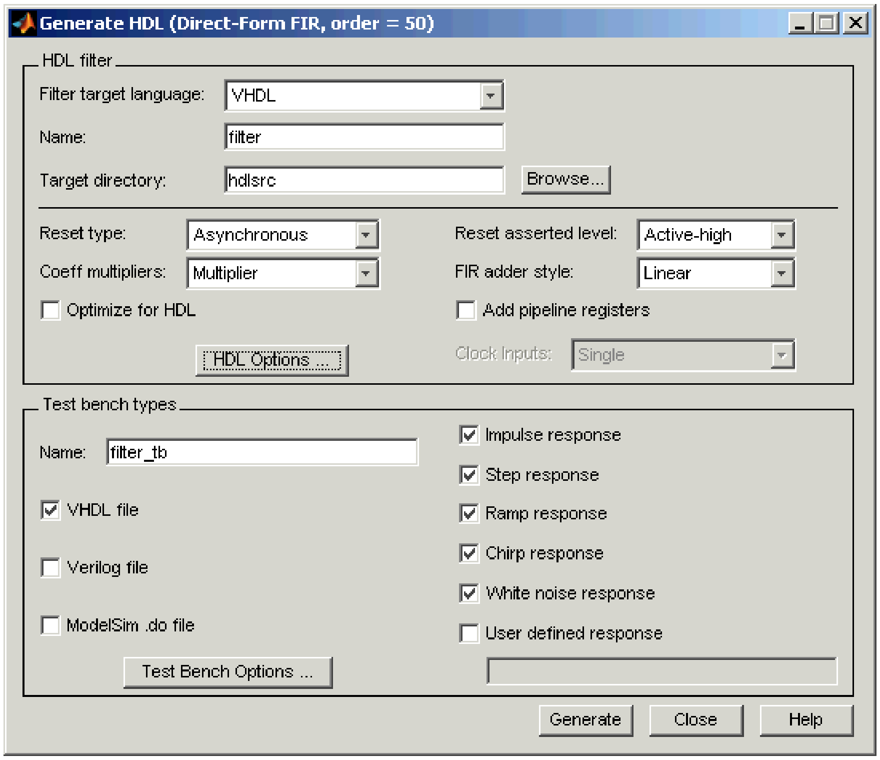
This section describes new features in the Filter Design HDL Coder 1.1.

Graphical User Interface Additions and Enhancements

The Filter Design HDL Coder GUI supports several new code generation options, and its layout and organization have been improved. The sections below show the current appearance of each dialog box that has been changed, and summarize new options and other revisions.

Generate HDL Dialog Box

This figure shows the default appearance of the **Generate HDL** dialog box.



Generate HDL (Direct-Form FIR, order = 50)

HDL filter

Filter target language: VHDL

Name: filter

Target directory: hdlsrc

Reset type: Asynchronous

Reset asserted level: Active-high

Coeff multipliers: Multiplier

FIR adder style: Linear

☐ Optimize for HDL

☐ Add pipeline registers

Clock Inputs: Single

Test bench types

Name: filter_tb

☑ VHDL file

☐ Verilog file

☐ ModelSim .do file

☑ Impulse response

☑ Step response

☑ Ramp response

☑ Chirp response

☑ White noise response

☐ User defined response

The **Generate HDL** dialog box incorporates the following changes:

- A target directory **Browse** button has been added. The **Browse** button opens a file browser dialog that lets you locate (or create) the directory into which generated code is written. The path to the selected directory is entered into the **Target directory** field.
- The **Generate** and **Close** buttons replace the **OK**, **Cancel**, and **Apply** buttons.

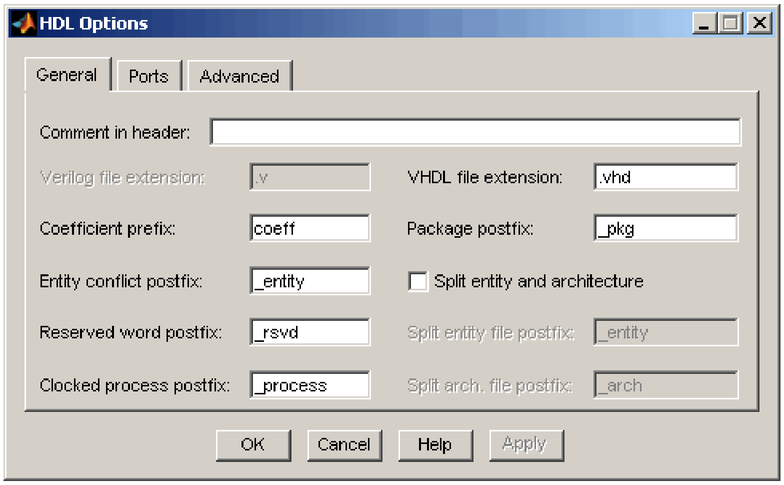
  The **Generate** button registers all option setting changes in the **Generate HDL** dialog (and its sub-dialogs) and initiates code generation.

  The **Close** button closes the **Generate HDL** dialog (and its sub-dialogs) without registering option setting changes.
- A new pull-down menu, **Clock inputs**, supports code generation for multirate filters (see "Multirate Filter Support" on page 3-6). **Clock inputs** is disabled by default. When a multirate filter is designed in FDATool, **Clock inputs** is enabled.
- The **HDL Options** button, which opens the **HDL Options** dialog box, replaces the **More Options** button.

HDL Options Dialog Box

The **HDL Options** dialog box incorporates changes to the **General** and **Ports** panes.

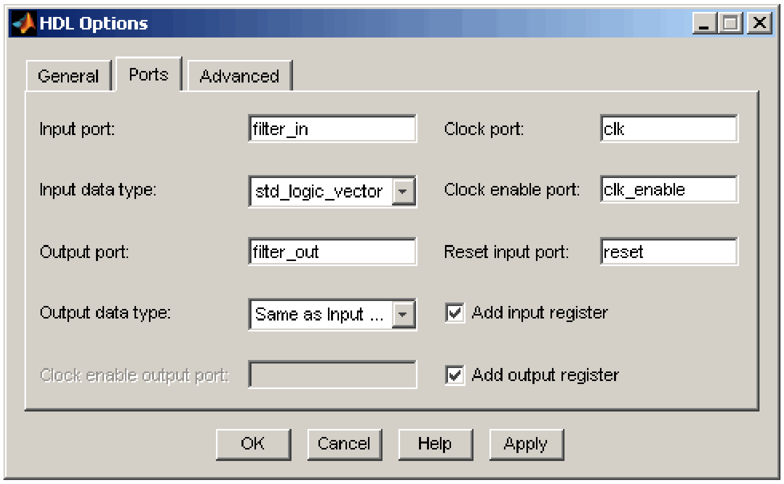
General Options. This figure shows the default appearance of the **General** pane of the **HDL Options** dialog box.



Comment in header:
Verilog file extension: .v
VHDL file extension: .vhd
Coefficient prefix: coeff
Package postfix: _pkg
Entity conflict postfix: _entity
☐ Split entity and architecture
Reserved word postfix: _rsvd
Split entity file postfix: _entity
Clocked process postfix: _process
Split arch. file postfix: _arch

In this pane, the **Coefficient name** edit field has been relabeled as **Coefficient prefix**. The function of the option is unchanged. As in previous releases, this field specifies a string to be used as the prefix for filter coefficient names. See also "CoeffPrefix Property Replaces CoeffName Property" on page 3-14.

Ports Options. This figure shows the default appearance of the **Ports** pane of the **HDL Options** dialog box.



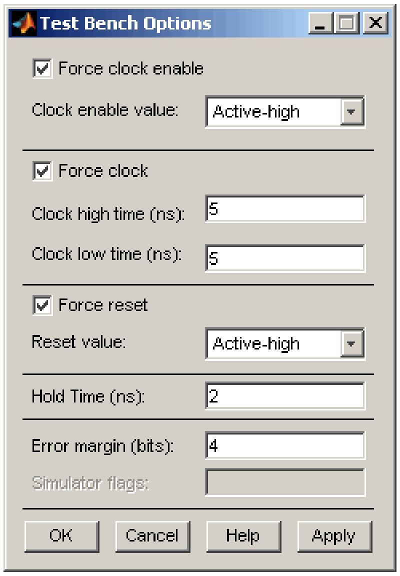
Input port: filter_in
Clock port: clk
Input data type: std_logic_vector
Clock enable port: clk_enable
Output port: filter_out
Reset input port: reset
Output data type: Same as Input ...
☑ Add input register
Clock enable output port:
☑ Add output register

A new option, **Clock enable output port**, supports code generation for multirate filters. **Clock enable output port** is disabled by default (as shown). See "Setting the Clock Enable Output Name" on page 3-11 for details on the use of this option.

Test Bench Options Dialog Box

The behavior of the **Error margin (bits)** field of the **Test Bench Options** dialog box has changed for certain cases.

For fixed-point filters, the **Error margin (bits)** field is now initialized to a default value of 4 when it is first enabled, as shown in the figure below.


Test Bench Options

✓ Force clock enable

Clock enable value: Active-high

✓ Force clock

Clock high time (ns): 5

Clock low time (ns): 5

✓ Force reset

Reset value: Active-high

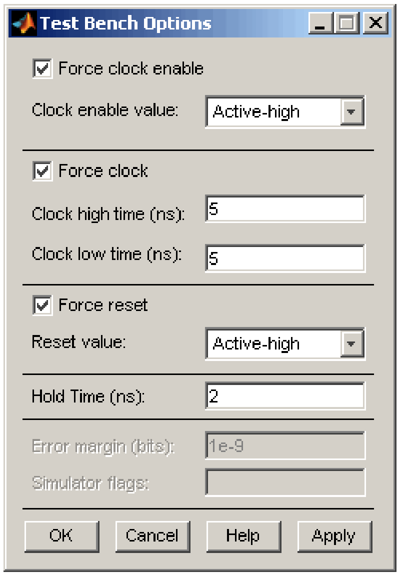
Hold Time (ns): 2

Error margin (bits): 4

Simulator flags:

Error margin (bits) is normally disabled and empty. It becomes enabled if you select certain optimizations (for example, **Optimize for HDL**) that can potentially cause the generated HDL filter to produce numeric results that differ from the results of the original MATLAB filter. As long as it is enabled, **Error margin (bits)** displays its current value. If **Error margin (bits)** becomes disabled again, it displays an empty field.

For double-precision floating-point filters, the **Error margin (bits)** value is fixed at `1e-9`. This value is displayed as shown in the figure below; the **Error margin (bits)** field is disabled to indicate that value cannot be changed.


Test Bench Options

[✓] Force clock enable

Clock enable value: Active-high

[✓] Force clock

Clock high time (ns): 5

Clock low time (ns): 5

[✓] Force reset

Reset value: Active-high

Hold Time (ns): 2

Error margin (bits): 1e-9

Simulator flags:

Multirate Filter Support

The Filter Design HDL Coder now supports code generation for several types of multirate filters:

- Cascaded Integrator Comb (CIC) interpolation (`mfilt.cicdecim`)
- Cascaded Integrator Comb (CIC) decimation (`mfilt.cicinterp`)
- Direct-Form Transposed FIR Polyphase Decimator (`mfilt.firtdecim`)

To generate multirate filter code, you must first select and design one of the supported filter types in the multirate design panel of FDATool. See "Designing Multirate Filters in FDATool" in the Filter Design Toolbox documentation for information about multirate filter design. After you have created the filter, open the **Generate HDL** dialog and set the desired code generation properties.

To support multirate filter code generation, the Filter Design HDL Coder provides new GUI options. These are described in “Code Generation Options for Multirate Filters” on page 3-7.

The Filter Design HDL Coder also defines new multirate filter code generation properties for `generatehdl`. These properties are described in “generatehdl Properties for Multirate Filters” on page 3-11.

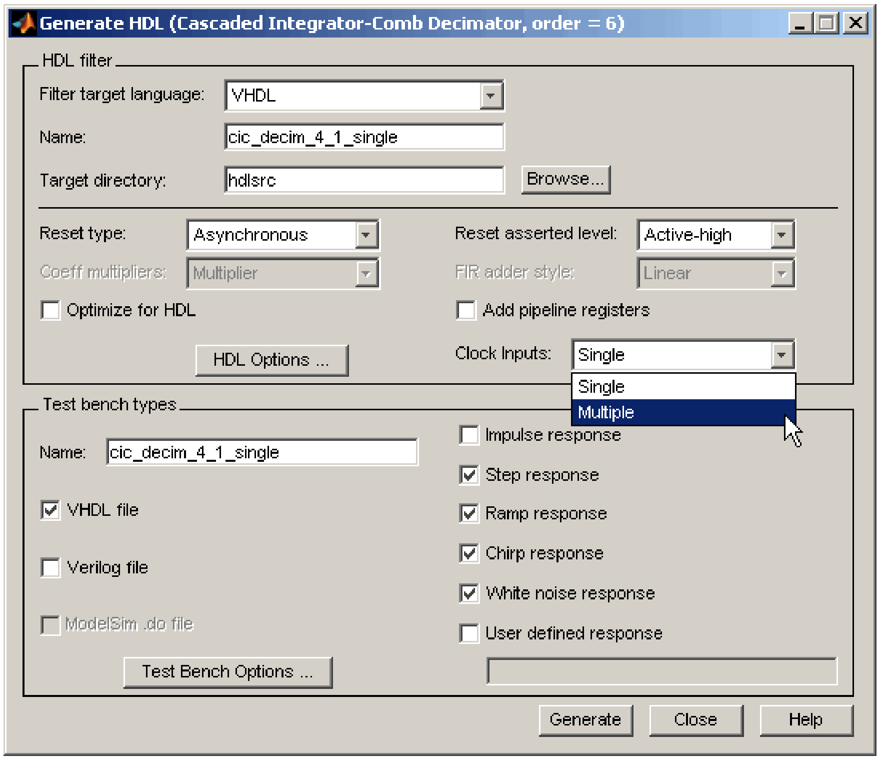
Code Generation Options for Multirate Filters

When a multirate filter (of one of the supported types listed previously) is designed in FDATool, the enabled/disabled state of several options in the **Generate HDL** dialog changes:

- The **Clock inputs** pulldown menu is enabled. This menu provides two alternatives for generating clock inputs for multirate filters, as discussed below.
- The **ModelSim .do file** option is disabled. Generation of ModelSim .do test bench files is not supported for multirate filters.
- For CIC filters, the **Coefficient multipliers** option is disabled. Coefficient multipliers are not used in CIC filters.

  However, the **Coefficient multipliers** option is enabled for Direct-Form Transposed FIR Polyphase Decimator (`mfilt.firtdecim`) filters.
- For CIC filters, the **FIR adder style** option is disabled, since CIC filters do not require a final adder.

The figure below shows the default settings of the **Generate HDL** dialog options when a supported CIC filter has been designed in FDATool.



Generate HDL (Cascaded Integrator-Comb Decimator, order = 6)

HDL filter

Filter target language: VHDL

Name: cic_decim_4_1_single

Target directory: hdlsrc

Reset type: Asynchronous

Reset asserted level: Active-high

Coeff multipliers: Multiplier

FIR adder style: Linear

☐ Optimize for HDL

☐ Add pipeline registers

Clock Inputs: Single

Test bench types

Name: cic_decim_4_1_single

☑ VHDL file

☐ Verilog file

☐ ModelSim .do file

☐ Impulse response

☑ Step response

☑ Ramp response

☑ Chirp response

☑ White noise response

☐ User defined response

The **Clock inputs** menu choices are

- `Single` : When `Single` is selected, the `ENTITY` declaration for the filter defines a single clock input with an associated clock enable input and clock enable output. The generated code maintains a counter that controls the timing of data transfers to the filter output (for decimation filters) or input (for interpolation filters). The counter is, in effect, a secondary clock enable whose rate is determined by the filter's decimation or interpolation factor.

  The `Single` option is primarily intended for FPGAs. It provides a self-contained solution for multirate filters, and does not require you to provide any additional code.

  The following code excerpts were generated from a CIC decimation filter having a decimation factor of 4, with **Clock inputs** set to `Single`.

  The `ENTITY` declaration is as follows:

```
ENTITY cic_decim_4_1_single IS
   PORT( clk            :   IN    std_logic;
         clk_enable     :   IN    std_logic;
         reset          :   IN    std_logic;
         filter_in      :   IN    std_logic_vector(15 DOWNTO 0); -- sfix16_En15
         filter_out     :   OUT   std_logic_vector(15 DOWNTO 0); -- sfix16_En15
         ce_out         :   OUT   std_logic
         );

END cic_decim_4_1_single;
```

The signal `counter` is maintained by the clock enable output process (`ce_output`). Every 4th clock cycle, `counter` is toggled to 1.

```
ce_output : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      cur_count <= to_unsigned(0, 4);
    ELSIF clk'event AND clk = '1' THEN
      IF clk_enable = '1' THEN
        IF cur_count = 3 THEN
          cur_count <= to_unsigned(0, 4);
        ELSE
          cur_count <= cur_count + 1;
        END IF;
      END IF;
    END IF;
  END PROCESS ce_output;

  counter <= '1' WHEN cur_count = 1 AND clk_enable = '1' ELSE '0';
```

The following code excerpt illustrates a typical use of the `counter` signal, in this case to time the filter output.

```
output_reg_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      output_register <= (OTHERS => '0');
    ELSIF clk'event AND clk = '1' THEN
      IF counter = '1' THEN
```

```
        output_register <= section_out4;
      END IF;
    END IF;
  END PROCESS output_reg_process;
```

- `Multiple` :When `Multiple` is selected, the `ENTITY` declaration for the filter defines separate clock inputs (each with an associated clock enable input) for each rate of a multirate filter. (For currently supported multirate filters, there are two such rates).

  The generated code assumes that the clocks are driven at the appropriate rates. You are responsible for ensuring that the clocks run at the correct relative rates for the filter's decimation or interpolation factor. To see an example of such code, generate test bench code for your multirate filter and examine the `clk_gen` processes for each clock.

  The `Multiple` option is intended for ASICs and FPGAs. It provides more flexibility than the `Single` option, but assumes that you will provide higher-level code for driving your filter's clocks.

  Note that no synchronizers between multiple clock domains are provided.

  When `Multiple` is selected, clock enable outputs are not generated; therefore the **Clock enable output port** field of the **HDL Options** dialog is disabled.

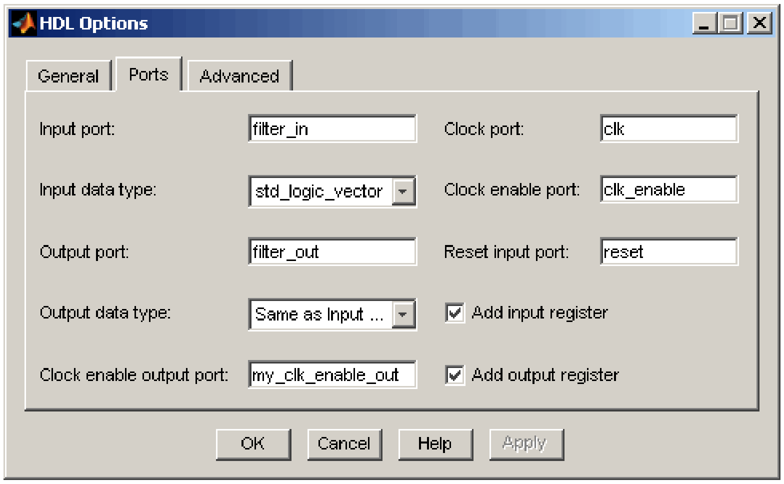
  The following `ENTITY` declaration was generated from a CIC decimation filter with **Clock inputs** set to `Multiple`.

```
ENTITY cic_decim_4_1_multi IS
   PORT( clk              :   IN    std_logic;
         clk_enable       :   IN    std_logic;
         reset            :   IN    std_logic;
         filter_in        :   IN    std_logic_vector(15 DOWNTO 0); -- sfix16_En15
         clk1             :   IN    std_logic;
         clk_enable1      :   IN    std_logic;
         reset1           :   IN    std_logic;
         filter_out       :   OUT   std_logic_vector(15 DOWNTO 0)  -- sfix16_En15
         );

END cic_decim_4_1_multi;
```

Setting the Clock Enable Output Name. A clock enable output is generated when `Single` is selected from the **Clock inputs** menu in the **Generate HDL** dialog. The default name for the clock enable output is `ce_out`.

To change the name of the clock enable output, enter the desired name into the **Clock enable output port** field of the **HDL Options** dialog, as shown below.



Input port: filter_in
Clock port: clk
Input data type: std_logic_vector
Clock enable port: clk_enable
Output port: filter_out
Reset input port: reset
Output data type: Same as Input ...
☑ Add input register
Clock enable output port: my_clk_enable_out
☑ Add output register

Note that the **Clock enable output port** field is disabled when multiple clocks are being generated.

Generating Test Bench Code. You can generate VHDL or Verilog test bench files for multirate filters. Generation of ModelSim .do test bench files is not supported for multirate filters, and the **ModelSim .do file** option of the **Generate HDL** dialog is disabled.

generatehdl Properties for Multirate Filters

If you are using `generatehdl` to generate code for a multirate filter, you can set the following properties to specify clock generation options:

- `ClockInputs` (String): Select generation of single or multiple clock inputs for multirate filters.

Valid values are `'Single'` or `'Multiple'`.

Default: `'Single'`.

- `ClockEnableOutputPort` (String): Specifies the name of the clock enable output port. Note that the clock enable output port is generated only when `ClockInputs` is set to `'Single'`.

  Default: `'ce_out'`.

Cascade Filter Support

The Filter Design HDL Coder now supports code generation for the following types of cascade filters:

- Multirate cascade of filter objects (`mfilt.cascade`)
- Cascade of discrete-time filter objects (`dfilt.cascade`)

To generate cascade filter code:

1. Instantiate the filter stages and cascade them in the MATLAB workspace (see the Filter Design Toolbox documentation for the `mfilt.cascade` and `mfilt.cascade` filter objects).

   The Filter Design HDL Coder currently imposes certain limitations on the filter types allowed in a cascade filter. See "Rules and Limitations for Code Generation with Cascade Filters" on page 3-13 before creating your filter stages and cascade filter object.

2. Import the cascade filter object into FDATool, as described in "Importing and Exporting Quantized Filters" in the Filter Design Toolbox documentation.

3. After you have imported the filter, open the **Generate HDL** dialog, set the desired code generation properties, and generate code. See "Rules and Limitations for Code Generation with Cascade Filters" on page 3-13

4. Note that the Filter Design HDL Coder generates separate HDL code files for each stage of the cascade, in addition to the top-level code for the cascade filter itself. The filter stage code files are identified by appending the string `_stage1`, `_stage2`, ... `_stage`*`N`* to the filter name.

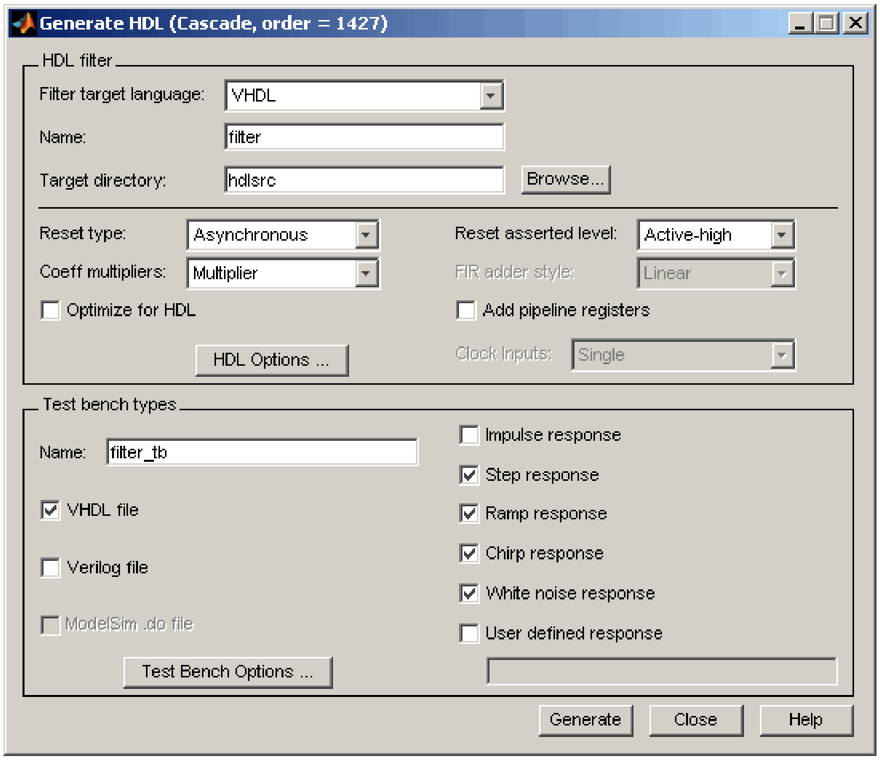
Rules and Limitations for Code Generation with Cascade Filters

The following rules and limitations apply to code generation with cascade filters:

- You can generate code for cascades that combine the following filter types:
  - Decimators and/or single-rate filter structures
  - Interpolators and/or single-rate filter structures

  Code generation for cascades that include both decimators and interpolators is not currently supported, however. If unsupported filter structures are included in the cascade, code generation is disallowed.
- For code generation, only a flat (single-level) cascade structure is allowed. Nesting of cascade filters is disallowed.
- By default, all input /output registers are removed from the stages of the cascade in generated code, except for the input of the first stage and the output of the final stage. However, if the **Add pipeline registers** option in **Generate HDL** dialog is selected, the output registers for each stage are generated, and internal pipeline registers may be added, depending on the filter structures.
- When a cascade filter is created in FDATool, the enabled/disabled state of several options in the **Generate HDL** dialog changes:
  - The **ModelSim .do file** option is disabled. Generation of ModelSim .do test bench files is not supported for multirate filters.
  - The **FIR adder style** option is disabled. If you require tree adders for FIR filters in a cascade, select the **Add pipeline registers** option (since pipelines require tree style FIR adders).

The figure below shows the default settings of the **Generate HDL** dialog options when a cascade filter has been designed in FDATool.



Generate HDL (Cascade, order = 1427)

HDL filter

Filter target language: VHDL

Name: filter

Target directory: hdlsrc

Reset type: Asynchronous

Reset asserted level: Active-high

Coeff multipliers: Multiplier

FIR adder style: Linear

☐ Optimize for HDL

☐ Add pipeline registers

Clock Inputs: Single

Test bench types

Name: filter_tb

☑ VHDL file

☐ Verilog file

☐ ModelSim .do file

☐ Impulse response

☑ Step response

☑ Ramp response

☑ Chirp response

☑ White noise response

☐ User defined response

CoeffPrefix Property Replaces CoeffName Property

The `CoeffName`property has been renamed to `CoeffPrefix`. The purpose of the `CoeffPrefix` remains unchanged. This property specifies a string to be used as the prefix for filter coefficient names.

For backward compatibility, `CoeffName` is still supported. Existing code that uses `CoeffName` will run without change. However, we recommend updating your code to use the current property name.

Filter Design HDL Coder 1.0 Release Notes

These release notes provide the following information:

Introduction to the Filter Design HDL Coder

The Filter Design HDL Coder generates hardware description language (HDL) code and test benches for filters you design with FDATool. The Filter Design HDL Coder accelerates the development of application-specific integrated circuit (ASIC) and field programmable gate array (FPGA) filter designs and bridges the gap between system-level design and hardware development by generating HDL code based on algorithms developed in MATLAB. Currently, system designers and hardware developers use HDLs, such as very high speed integrated circuit (VHSIC) hardware definition language (VHDL) and Verilog, to develop hardware designs. Although HDLs provide a proven method for hardware design, the task of coding filter designs and hardware designs in general, is labor intensive and the use of these languages for algorithm and system-level design is not optimal.

Using the Filter Design HDL Coder, system architects and designers can spend more time on fine-tuning algorithms and models through rapid prototyping and experimentation and less time on HDL coding. The architects and designers can efficiently design, analyze, simulate, and transfer system designs to hardware developers. In a typical use scenario, an architect or designer uses the Filter Design Toolbox, its Filter Design and Analysis Tool (FDATool), and the Filter Design HDL Coder to design and configure a filter. Then, with the click of a button, the Filter Design HDL Coder generates a VHDL or Verilog implementation of the design and specified test benches. The generated code adheres to a clean HDL coding style that enables architects and designers to quickly address customizations, as needed. The test bench feature increases confidence in the correctness of the generated code and saves potential time spent on test bench implementation.

The Filter Design HDL Coder includes the following features:

- Graphical user interface (GUI) plug-in to the Filter Design and Analysis Tool (FDATool)
- MATLAB command line interface
- Support for the following filter structures:
  - Finite impulse response (FIR)
  - Antisymmetric FIR

  - Transposed FIR
  - Symmetric FIR
  - Second-order section (SOS) infinite impulse response (IIR) Direct Form I
  - SOS IIR Direct Form I transposed
  - SOS IIR Direct Form II
  - SOS IIR Direct Form II transposed
  - Cascaded Integrator Comb (CIC) interpolation
  - Cascaded Integrator Comb (CIC) decimation
  - Direct-Form Transposed FIR Polyphase Decimator
  - Direct-Form FIR Polyphase Interpolator
  - Direct-Form FIR Polyphase Decimator
  - FIR Hold Interpolator
  - FIR Linear Interpolator
  - Discrete-Time Scalar
  - Cascade filters
- Generation of code that adheres to a clean HDL coding style
- Options for controlling the contents and style of the generated HDL code
- Options for optimizing numeric results of generated HDL code
- Test bench generation for validating the generated HDL filter code
- Portable VHDL, portable Verilog, and ModelSim Tcl/Tk DO file test bench options