# Using Quartus II Verilog HDL & VHDL Integrated Synthesis

December 2002, ver. 1.2 Application Note 238

## Introduction

The Altera® Quartus® II software includes improved integrated synthesis that fully supports the Verilog HDL and VHDL languages and provides options to control the synthesis process. With this synthesis support, the Quartus II software provides a complete, easy-to-use, standalone solution for system-on-a-programmable-chip (SOPC) designs.

In addition to describing Quartus II synthesis support, this application note explains how you can improve Quartus II synthesis results by:

- Using Quartus II synthesis options
- Using design planning practices
- Using synchronous design methodologies
- Following Altera-recommended guidelines for writing HDL code
- Following guidelines for using architectural features of Altera devices (instantiating and inferring Altera megafunctions)

☞ The dialog boxes and settings described in this document reflect the Quartus II software version 2.2. If you are using another version of Quartus II software, the settings may be different. Refer to Quartus II Help for your version for information on the appropriate dialog boxes and settings.

## Verilog HDL & VHDL Support

The Quartus II software's integrated synthesis fully supports Verilog HDL and VHDL synthesizable language features, as well as some compiler directives.

For information on specific syntax features and language constructs, refer to the "Quartus II Verilog HDL Support" and "Quartus II VHDL Support" topics in Quartus II Help.

### Verilog HDL

The Quartus II Compiler's Logic Synthesizer module supports the Verilog-2001 standard (IEEE Std. 1364-1995) and some Verilog-2001 standard (IEEE Std. 1364-2001) constructs. You can select which standard to use in the in the **Settings** dialog box (Assignments menu). Select **Verilog HDL Input** under **HDL Input Settings** in the **Category** list. The Compiler uses Verilog-2001 by default.

The code samples provided in this document follow the Verilog-2001 standard.

Verilog-2001 support is a new feature in the Quartus II software version 2.1 and later. Supported constructs include:

- Generate statements: `generate` and `genvar`
- `localparam` constants
- Preprocessor statements such as `'elsif`, `'line`, `'ifdef`, and `'file`
- Signed declarations for all variables
- Operators such as `**`, `<<<`, and `>>>`
- Attributes using the syntax `(* name = value *)`
- Indexed part selects using `+:` and `-:`
- Combinational logic sensitivity wild card token `@*`

Refer to Quartus II Help for a complete listing of supported constructs.

The Quartus II software supports case-sensitive Verilog HDL code, per the Verilog HDL standard. Before version 2.1, the Quartus II software did not support case-sensitive module names.

## VHDL

The Quartus II Compiler's Logic Synthesizer module supports the VHDL 1987 (IEEE Std. 1076-1987) and 1993 (IEEE Std. 1076-1993) standards. You can select which standard to use in the in the **Settings** dialog box (Assignments menu). Select **VHDL Input** under **HDL Input Settings** in the **Category** list. The Compiler uses VHDL 1993 by default.

The code samples provided in this document follow the VHDL 1993 standard.

The Quartus II software now supports VHDL libraries differently from older versions of the Quartus II or MAX+PLUS® II software. In version 2.1 and later, standard IEEE and vendor VHDL libraries and packages can be called from VHDL code within the Quartus II software.

The **IEEE** library includes the standard VHDL packages **std_logic_1164**, **numeric_std**, and **numeric_bit**. The **STD** library is part of the VHDL language standard and includes packages **standard** (included in every project by default) and **textio**. For compatibility with older designs, the Quartus II software also supports vendor-specific packages/libraries, including:

- Synopsys packages such as **std_logic_arith** and **std_logic_unsigned** in the **IEEE** library

- Mentor Graphics packages such as **std_logic_arith** in the **ARITHMETIC** library
- Altera packages such as **maxplus2**, **altera_mf_components**, and **lpm_components** in the **ALTERA** library

For a complete listing of library and package support, refer to the "Using Quartus II Packages" topic in Quartus II Help.

The Quartus II software does not support user-defined precompiled libraries because standard third-party synthesis tools do not support these precompiled libraries and these libraries do not provide significant reduction in compilation times. In older Quartus II versions, you declared precompiled libraries in the **VHDL Input** tab of the **General Settings** dialog box (Project menu).

To call a user-defined VHDL package in Quartus II version 2.1 and later, indicate the library and package name using the `LIBRARY` and `USE` commands. You can use any name for your library, including `work`; therefore, you can use current software versions for projects developed with older versions of Altera software that used precompiled libraries without the need to modify any code. To compile these projects, include the VHDL package in your Quartus II project by going to the **Add/Remove** page under **Files & Directories** in the **Settings** dialog box (Assignments menu). The package must be listed before other files that use the package because it must be analyzed by the Quartus II Compiler first.

## Synthesis Options

The Quartus II software provides a number of options to guide the synthesis process and achieve optimal results. You can use compiler directives, attributes, and Quartus II logic options to control synthesis.

Versions of Quartus II software earlier than 2.1 did not support compiler directives or attributes; the software treated these options as comments. Quartus II behavior is different if designs compiled in earlier versions of the software included these synthesis options. You may need to change your code now that the software recognizes these options.

This section defines three types of synthesis options, compiler directives, attributes, and Quartus II logic options. It also describes each of the following options:

- Translate Off & On
- Read Comments as HDL
- Full Case
- Parallel Case
- Preserve Registers

- Maximum Fan-Out
- Optimization Technique
- State Machine Processing
- Preserve Hierarchical Boundary
- Power-Up High
- Power-Up Don't Care
- Remove Duplicate Logic
- Remove Duplicate Registers
- Remove Redundant Logic Cells

## Compiler Directives

The Quartus II software supports compiler directives, also called pragmas. You include compiler directives in Verilog HDL or VHDL code as comments. These directives are not Verilog HDL or VHDL commands; however, synthesis tools use them to drive the synthesis process in a particular manner. Other tools such as simulators ignore these directives and treat them as comments.

You can enter compiler directives in your code using the following syntax, where *directive* and *value* are variables and the entry within brackets is optional.

**Verilog HDL:**

```
// synthesis <directive> { <value> }
or
/* synthesis <directive> { <value> } */
```

**VHDL:**

```
-- synthesis <directive> { <value> }
```

☞ In addition to the `synthesis` keyword shown above, the keywords `pragma`, `synopsys`, and `exemplar` are supported in both Verilog HDL and VHDL for compatibility with other synthesis tools.

## Attributes

The Quartus II software supports attributes, also known as pragmas or directives. Attributes are similar to compiler directives in that they drive the synthesis process; however, attributes always apply to a specific design element. Some attributes are also available as Quartus II logic options.

**Table 1. Synthesis Options as Compiler Directives, Attributes & Logic Options**

| Synthesis Option | Compiler Directive | Attribute | Quartus II Logic Option |
|---|---|---|---|
| Translate Off and On | `translate_off` `translate_on` | – | – |
| Read Comments as HDL | `read_comments_as_HDL` | – | – |
| Full Case | – | `full_case` | – |
| Parallel Case | – | `parallel_case` | – |
| Preserve Registers | – | `preserve` `syn_preserve` | Preserve Registers |
| Maximum Fan-Out | – | `maxfan` `syn_maxfan` | Maximum Fan-Out |
| Optimization Technique | – | – | Optimization Technique |
| State Machine Processing | – | – | State Machine Processing |
| Preserve Hierarchical Boundary | – | – | Preserve Hierarchical Boundary |
| Power-Up High | – | – | Power-Up High |
| Power-Up Don't Care | – | – | Power-Up Don't Care |
| Remove Duplicate Logic | – | – | Remove Duplicate Logic |
| Remove Duplicate Registers | – | – | Remove Duplicate Registers |
| Remove Redundant Logic Cells | – | – | Remove Redundant Logic Cells |

☞ Because Verilog HDL is case-sensitive, compiler directives and attributes are also case sensitive.

The following sections provide more information on each option shown in Table 1.

### Translate Off & On

The `translate_off` and `translate_on` compiler directives indicate whether the Quartus II software or a third-party synthesis tool should compile a portion of HDL code that is not relevant for synthesis. The `translate_off` directive marks the beginning of code that the synthesis tool should ignore; the `translate_on` directive indicates that synthesis should resume. A common use of these directives is to indicate a portion of code that is intended for simulation only. The synthesis tool reads synthesis-specific directives and processes them during synthesis; however, third-party simulation tools read the directives as comments and ignore them. Figures 1 and 2 show examples of these directives.

***Figure 1. Verilog HDL Example of Translate On & Off***

```
// synthesis translate_off
parameter       tpd = 2;    // Delay for simulation

#tpd;
// synthesis translate_on
```

***Figure 2. VHDL Example of Translate On & Off***

```
-- synthesis translate_off
use std.textio.all;
-- synthesis translate_on
```

### *Read Comments as HDL*

The `read_comments_as_HDL` compiler directive indicates that the Quartus II software should compile a portion of HDL code that is commented out. This directive allows you to comment out portions of HDL source code that are not relevant for simulation, while instructing the Quartus II software to read and synthesize that same source code. Setting the `read_comments_as_HDL` directive to `on` marks the beginning of commented code that the synthesis tool should read; setting the `read_comments_as_HDL` directive to `off` indicates the end of the code.

☞ You can use the directive with `translate_off` and `translate_on` to create one HDL source file that includes both a megafunction instantiation for synthesis and a behavioral description for simulation.

In Figures 3 and 4, the commented code enclosed by `read_comments_as_HDL` is visible to the Quartus II Compiler and is synthesized.

☞ Compiler directives are case-sensitive in Verilog HDL; you must match the case of the directive as shown in Figure 3

***Figure 3. Verilog HDL Example of Read Comments as HDL***

```
// synthesis read_comments_as_HDL on
// my_rom lpm_rom (.address (address),
//    .data      (data));
// synthesis read_comments_as_HDL off
```

Verilog-2001 syntax also accepts the following statements in the `case` header instead of the comment form as shown in Figure 6:

```
(* parallel_case *) casez (sel)
```

### *Preserve Registers*

This attribute and logic option directs the compiler not to minimize or remove a specified register during synthesis optimization or sequential netlist optimization. Sequential optimizations can eliminate redundant registers and registers with constant drivers. This option can preserve a register so you can observe it during simulation or with the SignalTap® II logic analyzer. Additionally, it can preserve registers if you are creating a preliminary version of the design in which secondary signals are not specified. The option cannot preserve registers that have no fan-out.

You can set the Preserve Registers logic option in the Quartus II user interface or you can set the `preserve` attribute in your HDL code as shown below. In this example, the `my_reg` register is preserved.

☞ In addition to `preserve`, the Quartus II software supports the `syn_preserve` attribute name for compatibility with other synthesis tools.

**Verilog HDL:**

```
reg my_reg /* synthesis preserve */;
```

**Verilog-2001:**

```
(* preserve *) reg my_reg;
```

**VHDL:**

```
signal my_reg : stdlogic;

attribute preserve : boolean;
attribute preserve of my_reg : signal is true;
```

☞ Setting the Preserve Registers logic option in the Quartus II software version 2.2 does not affect registers that are removed during the analysis and elaboration stage of compilation (before logic synthesis). To fully preserve the register throughout compilation, use the HDL attribute instead of the logic option.

### *Maximum Fan-Out*

This attribute and logic option directs the compiler to control the number of destinations fed by a node. The fan-out count of the node will not exceed the value specified for the maximum number of fan-out. You can apply this option to a register, pin, or a logic cell buffer. This option is useful for reducing the load of critical signals, which can improve performance. Additionally, you can use this option to instruct the compiler to duplicate or replicate a register that feeds nodes in different locations on the target device. Duplicating the register may allow the Fitter to place these new registers closer to their destination logic, minimizing routing delay.

This option is available for all devices supported in the Quartus II software except for MAX®, FLEX®, ACEX®, and Mercury™ devices.

You can set the Maximum Fanout logic option in the Quartus II user interface, or you can set the `maxfan` attribute in your HDL code as shown below. In this example, the compiler duplicates the `clk_gen` register so its fan-out is not greater than 50.

☞ In addition to `maxfan`, the Quartus II software supports the `syn_maxfan` attribute name for compatibility with other synthesis tools.

**Verilog HDL:**

```
reg clk_gen /* synthesis maxfan 50 */;
```

**Verilog-2001:**

```
(* maxfan = 50 *) reg clk_gen;
```

**VHDL:**

```
signal clk_gen : stdlogic;

attribute maxfan : integer ;
attribute maxfan of clk_gen : integer is 50;
```

### *Optimization Technique*

This logic option specifies the overall goal for logic optimization, i.e., whether to attempt to achieve maximum speed performance or minimum area usage during compilation. Table 2 lists the settings for this option, which you can only apply to a design entity.

***Table 2. Optimization Technique Settings***

| Setting | Description |
|---|---|
| **Area** | The Compiler makes the design as small as possible to minimize resource usage. |
| **Speed** | The Compiler chooses a design implementation that has the fastest $f_{MAX}$. |

The default setting varies by target device family, and is generally optimized to get the best area/speed trade-off. Results are design and device-dependent and can vary depending on which design or device family you use.

### *State Machine Processing*

This logic option specifies the processing style used to compile a state machine. Table 3 lists the settings for this option, which you can apply to a state machine name or to a design entity containing a state machine.

***Table 3. State Machine Processing Settings***

| Setting | Description |
|---|---|
| **Auto** (Default) | Allows the Compiler to choose the best encoding for the state machine. |
| **Minimal Bits** | Uses the least number of bits to encode the state machine. |
| **One-Hot** | Encodes the state machine in the one-hot style. |
| **User-Encoded** | Encodes the state machine in the manner specified by the user. |

☞ The Compiler in the Quartus II software versions 2.1 and 2.2 does not report Verilog HDL state machines. The software correctly reads and implements state machine logic during synthesis, however, the Compiler does not report state information and you cannot change the encoding using Quartus II logic options.

- If this option is turned on for an output or bidirectional pin, it is transferred automatically to the register that feeds the pin if the following conditions are present:
  - There is no intervening logic, other than inversion, between the register and the pin.
  - The register does not fan out to any other logic.

### *Power-Up Don't Care*

This logic option causes registers to power up with a "don't care" logic level (X), or the logic level most appropriate for the design. You might use this option to allow the Compiler to change the power-up level of a register to minimize your design's area usage.

For example, a register may have its D input tied to VCC. If you turned this option off, the register powers up low even though it goes high at the first clock signal. If you turned this option on, the Compiler sets the power-up value of the register to high and, therefore, can eliminate the register and connect the output of the register to VCC. If the Compiler makes this type of optimization, it issues you a message indicating it is doing so.

This project-wide option does not apply to registers that have the **Power-Up High** logic option set (either on or off).

☞ Versions of the Quartus II software earlier than version 2.1 did not include this option. If you compile an older design that relies on registers to power-up to a specific level, the Compiler may synthesize the design differently. Turn off the **Power-Up Don't Care** option if you want your design to use the power-up behavior of older versions of Quartus II software.

### *Remove Duplicate Logic*

If you turn on this option, the Compiler removes logic if it is identical to other logic in the design. If two functions generate the same logic, the Compiler removes the second one and the first one fans out to the second one's destinations. Additionally, if the deleted logic function has different logic option assignments, the Compiler ignores them. This option is turned on by default.

When turned on, this option also removes all duplicate registers, like the **Remove Duplicate Registers** option. If you do not want the Compiler to remove certain registers when this option is turned on, turn off the **Remove Duplicate Registers** option for those registers. See Table 5 for additional details.

- For Verilog HDL, use the `parallel_case` attribute with case statements for one-hot functionality. Use `if` and `else_if` statements for all intentional priority encoders.
- Separate the state machine logic from all arithmetic functions and data paths, including assigning output values.

☞ The Compiler in the Quartus II software versions 2.1 and 2.2 does not report Verilog HDL state machines. The software correctly reads and implements state machine logic during synthesis, however, the Compiler does not report state information and you cannot change the encoding using Quartus II logic options.

## Architecture-Specific Coding Style Guidelines

This section discusses coding style guidelines to ensure optimal synthesis results when using architectural features of Altera devices. The section also provides code examples for inferring Altera megafunctions from HDL code in the Quartus II software.

### Altera Megafunctions

Altera provides parameterizable megafunctions that are optimized for Altera device architectures. Megafunctions include the library of parameterized modules (LPM), device-specific embedded megafunctions such as phase-locked loops (PLLs), Stratix DSP blocks, LVDS drivers, intellectual property (IP) available as Altera MegaCore® functions, and IP available from Altera Megafunction Partners Program (AMPP$^{SM}$) partners.

Using megafunctions instead of coding your own logic can save valuable design time. Additionally, these functions can offer more efficient logic synthesis and device implementation. It is easy to scale megafunctions to different sizes by simply setting parameters.

You must use megafunctions to access some Altera device-specific features, such as memory, DSP blocks, LVDS drivers, PLLs, and DDIO circuitry. You can use megafunctions by instantiating them in your HDL code or inferring them from generic HDL code.

### Instantiating Altera Megafunctions in HDL Code

You can instantiate Altera megafunctions in your HDL design by:

- Using the MegaWizard® Plug-In Manager to parameterize the function and create a wrapper file.
- Instantiating the function using the port and parameter definition.

### *Using the MegaWizard Plug-In Manager*

Altera recommends that you use the Quartus II MegaWizard Plug-In Manager to instantiate megafunctions. The wizard provides a graphical interface for customizing and parameterizing megafunctions, and ensures that you set all megafunction parameters properly. When you finish setting parameters, the wizard generates a VHDL or Verilog HDL wrapper file that instantiates the megafunction with the correct parameters (it also creates a Component Declaration file for VHDL). You can then instantiate the wrapper file in your HDL code.

Altera strongly recommends that you use the wizard for complex megafunctions such as PLLs and LVDS drivers.

Table 7 lists the files the MegaWizard Plug-In Manager generates and describes each file.

***Table 7. MegaWizard Plug-In Manager Generated Files***

| File | Description |
|---|---|
| *<output file>*.**bsf** | Symbol for the megafunction used in the Quartus II schematic editor. |
| *<output file>*.**cmp** | Component Declaration File. |
| *<output file>*.**inc** | Include File for the module in the megafunction wrapper file. |
| *<output file>*.**tdf** *(1)* | Megafunction wrapper file for instantiation in an AHDL design. |
| *<output file>*.**vhd** *(2)* | Megafunction wrapper file for instantiation in a VHDL design. |
| *<output file>*.**v** *(3)* | Megafunction wrapper file for instantiation in a Verilog HDL design. |
| *<output file>*_**bb.v** *(3)* | Hollow-body declaration of the module in the megafunction wrapper file used in Verilog HDL designs to specify port directions when using third-party synthesis tools. |
| *<output file>*_**inst.tdf** *(1)* | Sample AHDL instantiation of the subdesign in the megafunction wrapper file. |
| *<output file>*_**inst.vhd** *(2)* | Sample VHDL instantiation of the entity in the megafunction wrapper file. |
| *<output file>*_**inst.v** *(3)* | Sample Verilog HDL instantiation of the module in the megafunction wrapper file. |

*Notes to Table 7:*

(1) The wizard only generates this file if you select AHDL output files.
(2) The wizard only generates this file if you select VHDL output files.
(3) The wizard only generates this file if you select Verilog HDL output files.

For more information about how to use the MegaWizard Plug-In Manager, refer to Quartus II Help.

### *Using the Port & Parameter Definition*

You can instantiate the megafunction directly in your Verilog HDL or VHDL code by calling the function like any other module or component. In VHDL, you also need to use a Component Declaration. Refer to Quartus II Help (or your IP documentation) for a list of the megafunctions's ports and parameters. Help also provides a sample VHDL Component Declaration.

## Inferring Megafunctions from HDL Code

The Quartus II Logic Synthesizer automatically recognizes certain types of HDL code and infers the appropriate megafunction. That is, the software uses the Altera megafunction code when compiling your design even though you did not specifically instantiate the megafunction. The software uses inference because the megafunctions are optimized for Altera devices, so the area and/or performance may be better than generic HDL code. Additionally, you must use megafunctions to access certain architecture-specific features—such as RAM, DSP blocks, and shift registers—that generally provide improved performance compared to basic logic elements.

The following sections describe the types of logic that the Quartus II Logic Synthesizer recognizes and maps to megafunctions. The software only infers these specific functions that are described by HDL code. The software cannot infer other megafunctions, such as PLLs and LVDS drivers, from HDL code because these functions cannot be fully or efficiently described in HDL. In some cases, the Quartus II software has an option that you can use to disable inference.

### *Counters*

The Quartus II Logic Synthesizer looks for any set of registers that feeds itself through a plus-one adder, a minus-one adder, or both, and converts the registers and logic to an `lpm_counter` megafunction. If the design also has logic implementing counter signals, the software can recognize them as well. Specifically, the Quartus II software recognizes the following signals:

- Asynchronous clear
- Asynchronous set (only to all ones)
- Asynchronous load
- Count enable
- Synchronous clear
- Synchronous set (only to all ones)
- Synchronous load

- Clock enable
- Up/down

Figures 13 and 14 show simple Verilog HDL and VHDL counter examples with different control signals.

***Figure 13. Verilog HDL Counter with Count Enable & Asynchronous Clear***

```
module counter
(
      clk,
      reset,
      result,
      ena
);

      input clk;
      input reset;
      input ena;
      output [7:0] result;

      reg [7:0] result;

      always @(posedge clk or posedge reset)
      begin
            if (reset)
                  result = 0;
            else if (ena)
                  result = result + 1;
      end
endmodule
```

### *Multipliers*

The Quartus II Logic Synthesizer finds multipliers and converts them to `lpm_mult` megafunctions. For devices with DSP blocks, the software may implement the `lpm_mult` function in a DSP block instead of LEs, depending on device utilization. The Quartus II Fitter may also place input and output registers in DSP blocks (i.e., perform register packing) to improve performance and LE utilization.

For more information on the DSP block and which functions it can implement, refer to the appropriate Altera device family data sheet and the DSP Solutions Center on the Altera web site at **http://www.altera.com**.

Figures 17 through 20 show Verilog HDL and VHDL examples for unsigned and signed multipliers that the Quartus II Compiler infers as an `lpm_mult`. Each example fits into one DSP block 9-bit element (using no LEs for registers when register packing occurs).

☞ The signed declaration in Verilog HDL is a feature of the Verilog-2001 standard.

***Figure 17. Verilog HDL Unsigned Multiplier***

```
module unsigned_mult (out, a, b);

      output [15:0] out;
      input  [7:0] a;
      input  [7:0] b;

      assign out = a * b;

endmodule
```