11 VHDL Compiler Directives

Synopsys has defined several methods of providing circuit design information directly in your VHDL source code.

- Using VHDL Compiler directives, you can direct the translation from VHDL to components with special VHDL comments. These synthetic comments turn translation on or off, specify one of several hard-wired resolution methods, and provide a means to map subprograms to hardware components.
- Using Synopsys-defined VHDL attributes, you can add synthesis-related signal and constraint information to ports, components, and entities. This information is used by Design Compiler during synthesis.
- Using embedded scripts, you can add one or more Design Compiler commands to your VHDL source with special VHDL comments. When your design is read into Design Compiler, VHDL Compiler translates your design. Embedded commands are then executed by Design Compiler.

To familiarize yourself with VHDL Compiler directives, consider the following topics:

Notation for VHDL Compiler Directives

VHDL Compiler directives are special VHDL comments (*synthetic comments*) that affect the actions of the Synopsys VHDL Compiler or Design Compiler. These comments are just a special case of regular VHDL comments, so they are ignored by other VHDL tools. Synthetic comments are used only to direct the actions of VHDL Compiler.

Synthetic comments begin just as regular comments do, with two hyphens (– –). If the word following these characters is `pragma` or `synopsys`, the remaining comment text is interpreted by VHDL Compiler as a directive.

Note:

VHDL Compiler displays a syntax error if an unrecognized directive is encountered after `– – synopsys` or `– – pragma`.

VHDL Compiler Directives

The three types of directives are

- Translation stop and start Directives

```
– – pragma translate_off
– – pragma translate_on
– – pragma synthesis_off
– – pragma synthesis_on
```

- Resolution function directives

```
– – pragma resolution_method wired_and
– – pragma resolution_method wired_or
– – pragma resolution_method three_state
```

- Component implication directives

```
– – pragma map_to_entity entity_name
– – pragma return_port_name port_name
```

There are other directives such as map_to operator which are used to drive inference of HDL operators such as ″*″, ″+″, and ″–″. Refer to the *DesignWare Developer Guide* for more information.

Translation Stop and Start Directives

Translation directives stop and start VHDL Compiler's translation of a VHDL source file.

```
– – pragma translate_off
– – pragma translate_on
```

The `translate_off` and `translate_on` directives instruct VHDL Compiler to stop and start synthesizing VHDL source code. The VHDL code between directives is, however, checked for syntax.

For situations where you need VHDL compiler to completely ignore the text in between directives, set the variable `hdlin_translate_off_skip_text = true`. Setting this variable ensures that VHDL Compiler treats the text between the directives as comments. This additional behavior of `translate_off` and `translate_on` has been added for reasons of backward compatibility and should be used with caution.

Translation is enabled at the beginning of each VHDL source file. You can use `translate_off` and `translate_on` directives anywhere in the text. If the directive `hdlin_translate_off_skip_text` is set to `true`, three things will happen.

- Code being analyzed using `vhdlan –spc` or `dc_shell analyze` might produce unexpected results when simulated in VSS. This is because VSS does not simulate any VHDL text between the directives.
- Synopsys software ignores all the VHDL code between the directives.
- If you try to simulate a VHDL design that has this variable `on` and also uses the directives, the Synopsys simulator displays a warning and continues. Synopsys does not guarantee that the same code will simulate on any other simulator.

The `synthesis_off` and `synthesis_on` directives are the recommended mechanisms for hiding simulation-only constructs from synthesis. Any text between these directives is checked for syntax, but no corresponding hardware is synthesized. The behavior of the `synthesis_off` and `synthesis_on` directives is not affected by the variable `hdlin_translate_off_skip_text`. The behavior of the directives `translate_off/on` and `synthesis_off/on` is the same, if the variable `hdlin_translate_off_skip_text` is `false`.

Example 11–1 shows how you can use the directives to protect a simulation driver.

Example 11–1 Using synthesis_on and synthesis_off Directives

```
-- The following test driver for entity EXAMPLE
--   should not be translated:
--
– – pragma synthesis_off
--   Translation stops

entity DRIVER is
end;

architecture VHDL of DRIVER is
    signal A, B : INTEGER range 0 to 255;
    signal SUM  : INTEGER range 0 to 511;

    component EXAMPLE
        port (A, B: in INTEGER range 0 to 255;
              SUM: out INTEGER range 0 to 511);
    end component;

begin
    U1: EXAMPLE port map(A, B, SUM);
    process
    begin
        for I in 0 to 255 loop
            for J in 0 to 255 loop
                A <= I;
                B <= J;
                wait for 10 ns;
                assert SUM = A + B;
            end loop;
        end loop;
    end process;
end;

– – pragma synthesis_on
--   Code from here on is translated
```

```
entity EXAMPLE is
    port (A, B: in INTEGER range 0 to 255;
          SUM: out INTEGER range 0 to 511);
end;

architecture VHDL of EXAMPLE is
begin
    SUM <= A + B;
end;
```

Resolution Function Directives

Resolution function directives determine the resolution function associated with resolved signals (see ‘‘Signal Declarations” in Chapter 3). VHDL Compiler does not currently support arbitrary resolution functions. It does support the following three methods:

```
– – pragma resolution_method wired_and
– – pragma resolution_method wired_or
– – pragma resolution_method three_state
```

Note:

Do not connect signals that use different resolution functions. VHDL Compiler supports only one resolution function per network.

Component Implication Directives

Component implication directives map VHDL subprograms onto existing components or VHDL entities. These directives are described under ‘‘Subprograms” in Chapter 6:

```
– – pragma map_to_entity entity_name
– – pragma return_port_name port_name
```

Synthesis Attributes and Constraints

Design Compiler synthesis attributes and constraints are

attribute
: A characteristic of a design object, or of an object's context. For example, one attribute of an input port is its signal's expected arrival time. This is its `ARRIVAL` attribute.

  You can set attributes for input ports, output ports, and individual logic cells (VHDL components).

constraint
: A goal for optimization. For example, one major constraint on a design is its maximum input-to-output (transition) delay. This is its `MAX_DELAY` constraint.

  You can set constraints for entities and output ports.

When a design's external environment (such as signal arrival times) or its synthesis requirements (such as maximum circuit delay) are already known, the information can be added as VHDL attribute values in the VHDL source. You can also set synthesis attribute and constraint values directly in Design Compiler or Design Analyzer.

Synthesis attributes and constraints are described in the *Design Compiler Family Reference Manual*.

About VHDL Attributes

A VHDL attribute associates a named value with a VHDL object. VHDL Compiler defines several kinds of attributes for VHDL object classes `signal` (port), `component` (cell), and `entity` (design).

Note:

VHDL Compiler supports several predefined VHDL attributes for arrays, as described in Chapter 4 under ‘‘Array Types.” VHDL Compiler also supports reading two predefined VHDL `BOOLEAN` attributes for clock signals, `’event` and `’stable`, as described in Chapter 8, ‘‘Register and Three-State Inference.”

A VHDL attribute is defined with a name and a type. The syntax is

```
attribute name : type ;
```

For example:

```
attribute LOGIC_ONE : boolean;
```

A VHDL attribute value is then associated with one or more objects of a given class. The syntax is

```
attribute name of objects : class is value ;
```

where *objects* is a list of one or more object names separated by commas. *class* is `signal`, `component`, or `entity`. *value* is any appropriate value of the attribute’s declared type. For example

```
attribute LOGIC_ONE of INPUT7 : signal is TRUE;
```

The value of a VHDL object attribute is returned by

```
object_name’attribute_name
```

For example:

```
if (MY_ARRAY’LEFT = 0) . . .
```

Synthesis Attributes Defined for VHDL Compiler

VHDL Compiler supports the definition and setting of the attributes listed in the following sections. Other definitions are ignored. Reading attributes with the *object_name'attribute_name* construct is supported only for certain predefined attributes. See Names in Appendix C for a complete list. When you use the VHDL Compiler-defined attributes, you need to insert

```
use SYNOPSYS.ATTRIBUTES.all;
```

The definition for this package is in the `attributes.vhd` file described in Appendix B.

Example 11–2 shows the entity declaration used in Examples 11–3 and 11–4.

Example 11–2 Sample Entity Declaration

```
entity EXAMPLE is
  port (A, B:  in BIT;
        C:     in BIT_VECTOR (1 to 3);
        X, Y: out BIT;
        Z:    out BIT_VECTOR (1 to 3));
end EXAMPLE;
```

Using VHDL Attributes to Specify Synthesis Constraints

Two basic circuit constraints are the maximum expected area (max_area constraint), and the maximum allowable delay to an output (max_delay constraint).

Example 11–3 shows how to ask for the smallest possible circuit by using VHDL design (entity) attribute MAX_AREA with a value of 0.0.

Example 11–3 Circuit Area Constraint

```
entity EXAMPLE is
  port (A, B:  in BIT;
        C:     in BIT_VECTOR (1 to 3);
        X, Y: out BIT;
        Z:    out BIT_VECTOR (1 to 3));

  attribute MAX_AREA of EXAMPLE : entity is 0.0;
end EXAMPLE;
```

Example 11–4 shows how to ask for a circuit with a maximum delay of 10 (technology library time units), by using VHDL attribute MAX_DELAY, with a value of 10.0, on all output ports.

Example 11–4 Circuit Delay Constraint

```
entity EXAMPLE is
  port (A, B:  in BIT;
        C:     in BIT_VECTOR (1 to 3);
        X, Y: out BIT;
        Z:    out BIT_VECTOR (1 to 3));

  attribute MAX_DELAY of X, Y, Z : signal is 10.0;
end EXAMPLE;
```

The MAX_DELAY attribute in Example 11–4 is a multiple constraint applying to outputs X, Y, and Z together, rather than three separate constraints. For more information, see the discussion of constraints in the *Design Compiler Family Reference Manual*.

Input Port Attributes

Synthesis attributes for input ports are defined in VHDL as follows.

```
attribute ARRIVAL : real;
  -- Expected signal arrival time, in technology
  -- library time units. Sets both RISE_ARRIVAL and
  -- FALL_ARRIVAL.
attribute RISE_ARRIVAL : real;
  -- Input signal’s rise time.
attribute FALL_ARRIVAL : real;
  -- Input signal’s fall time.

attribute DRIVE_STRENGTH : real;
  -- Input signal’s drive strength, in technology
  -- library load units. Sets both RISE_DRIVE and
  -- FALL_DRIVE.

attribute FALL_DRIVE : real;
  -- Input signal’s drive strength while rising.
attribute RISE_DRIVE : real;
  -- Input signal’s drive strength while falling.

attribute EQUAL : boolean;
  -- Applied to pairs of input ports; true
  -- if both ports are logically equal.
attribute OPPOSITE : boolean;
  -- Applied to pairs of input ports; true
  -- if the two ports are logically opposite.

attribute LOGIC_ONE : boolean;
  -- True if the input port is always at logic one.
attribute LOGIC_ZERO : boolean;
  -- True if the input port is always at logic zero.

attribute DONT_TOUCH_NETWORK : boolean;
  -- True if the network connected to the input
  -- port is to be excluded from optimization.
```

Example 11–5 shows Synopsys VHDL attribute definitions and equivalent Design Compiler commands for all input port attributes. Note that C is declared as

```
C:in BIT_VECTOR(1 to 3)
```

and the `read_array_naming_style` variable is set to its default value, `"_"`. The attributes `EQUAL` and `OPPOSITE` are used only for pairs of single–bit ports (signals).

Example 11–5 Attributes of Input Ports

```
attribute ARRIVAL of A : signal is 1.5;
set_arrival 1.5 A

attribute FALL_ARRIVAL of A, B : signal is 1.5;
set_arrival –fall 1.5 {A B}

attribute RISE_ARRIVAL of C : signal is 1.5;
set_arrival –rise 1.5 {C_1 C_2 C_3}

attribute DRIVE_STRENGTH of A, B : signal is 0.01;
set_drive .01 {A B}

attribute FALL_DRIVE of B : signal is 0.01;
set_drive –fall .01 B

attribute RISE_DRIVE of A : signal is 0.01;
set_drive –rise .01 A

attribute EQUAL of A, B : signal is TRUE;
set_equal A B

attribute OPPOSITE of A, B: signal is TRUE;
set_opposite A B

attribute LOGIC_ONE of A : signal is TRUE;
set_logic_one A

attribute LOGIC_ZERO of A, B: signal is TRUE;
set_logic_zero {A B}

attribute DONT_TOUCH_NETWORK of A : signal is TRUE;
dont_touch_network A
```

Output Port Attributes

Synthesis attributes for output ports are defined in VHDL as

```
attribute LOAD : real;
  -- Loading on output port, in library load units.

attribute UNCONNECTED : boolean;
  -- May be set to true if the output port is not
  -- connected to external circuitry.
```

By using the entity declaration from Example 11–2, Example 11–6 shows VHDL definitions and equivalent Design Compiler commands for both output port attributes. Note that `z` is declared as

```
Z:out BIT_VECTOR(1 to 3)
```

Example 11–6 Attributes of Output Ports

```
attribute LOAD of Y, Z : signal is 5.0;
set_load 5 {Y Z_1 Z_2 Z_3}

attribute UNCONNECTED of X : signal is TRUE;
set_unconnected X
```

Cell (VHDL Component Instantiation) Attribute

Synthesis attributes for cells (VHDL component instantiations) are defined in VHDL as

```
attribute DONT_TOUCH : boolean;
  -- True if the instance is not to be optimized.
```

Example 11–7 shows a VHDL definition and an equivalent Design Compiler command for the cell (VHDL component instance) attribute. The `dont_touch` attribute is stored on a component only if it is set to `true`. A `dont_touch` attribute cannot be set to `false`.

Example 11–7 dont_touch Attribute for Component Instantiation

```
attribute DONT_TOUCH of INSTANCE : label is TRUE;
dont_touch INSTANCE
```

Design Constraints

Synthesis constraints for designs are defined in VHDL as

```
attribute MAX_AREA :       real;
  -- Maximum desired area, in technology library
  -- area units.

attribute MAX_TRANSITION : real;
  -- Maximum allowable transition time for any network
  -- in the design, in technology library time units.
```

By using the entity declaration from Example 11–2, Example 11–8 shows VHDL definitions and equivalent Design Compiler commands for both design constraints.

Example 11–8 Design Constraints

```
attribute MAX_AREA of EXAMPLE : entity is 500.0;
max_area 500.0

attribute MAX_TRANSITION of EXAMPLE : entity is 3.0;
set_max_transition 3.0
```

Output Port Constraints

Synthesis constraints for output ports are defined in VHDL as

```
attribute MAX_DELAY :      real;
  -- Maximum allowable delay time, from any
  -- input signal connected to the output
  -- port, in technology library time units.
  -- Sets both MAX_RISE_DELAY and MAX_FALL_DELAY.

attribute MAX_RISE_DELAY : real;
  -- Maximum allowable delay time before
  -- the output port’s signal rises.

attribute MAX_FALL_DELAY : real;
  -- Maximum allowable delay time before
  -- the output port’s signal falls.
```

```
attribute MIN_DELAY :      real;
  -- Minimum allowable delay time, from any
  -- input signal connected to the output
  -- port, in technology library time units.
  -- Sets both MIN_RISE_DELAY and MIN_FALL_DELAY.

attribute MIN_RISE_DELAY : real;
  -- Minimum allowable delay time before
  -- the output port's signal rises.

attribute MIN_FALL_DELAY : real;
  -- Minimum allowable delay time before
  -- the output port's signal falls.
```

By using the entity declaration from Example 11–2, Example 11–9 shows VHDL definitions and equivalent Design Compiler commands for all output port constraints. Note that Z is declared as

```
Z:out BIT_VECTOR(1 to 3)
```

Example 11–9 Constraints on Output Ports

```
attribute MAX_DELAY of X : signal is 20.0;
max_delay 20.0 X

attribute MAX_FALL_DELAY of X, Y : signal is 20.0;
max_delay –fall 20.0 {X Y}

attribute MAX_RISE_DELAY of Z : signal is 20.0;
max_delay –rise 20.0 {Z_1 Z_2 Z_3}

attribute MIN_DELAY of X, Z : signal is 5.0;
min_delay 5.0 {X Z_1 Z_2 Z_3}

attribute MIN_FALL_DELAY of Y : signal is 5.0;
min_delay –fall 5.0 Y

attribute MIN_RISE_DELAY of X : signal is 5.0;
min_delay –rise 5.0 X
```

Embedded Constraint and Attribute Commands

You can embed Design Compiler attribute and constraint commands, usually entered at the `dc_shell` prompt, in your VHDL source code.

Use VHDL Compiler directives `– – pragma dc_script_begin` and `– – pragma dc_script_end` to indicate the start and end of a set (script) of embedded commands. Between these directives, prefix the constraint or attribute command with the VHDL comment characters (two hyphens, – –), as shown in Example 11–10.

Example 11–10 Embedding Constraints and Attributes

```
– – pragma dc_script_begin
– – set_max_area 2500.0
– – set_drive –rise 1 port_b
– – pragma dc_script_end
```

Note:

`dc_shell` command comments outside the scope of `– – pragma dc_script_begin` and `– – pragma dc_script_end` are treated as regular VHDL comments. These comments are not passed on to Design Compiler.

Note the following limitations on the scope of constraints and attributes:

- Do not use embedded scripts inside packages or function declarations.
- Constraints and attributes declared outside an entity or architecture apply to all subsequent entities declared in the file.
- Constraints and attributes declared inside an entity or architecture apply only to the enclosing entity or architecture.

You have access to the full `dc_shell` command language, including control flow commands (`if`, `while`) and search commands (`find`, `get_attribute`). For information on writing `dc_shell` scripts, see the *Design Compiler Family Reference Manual*.

In an embedded `dc_shell` script include only commands that set constraints and attributes. Do not use action commands such as `compile`, `ungroup`, or `report`.

Note:

> Attributes or constraints set in the embedded script become effective *after* the `read -format vhdl` command has completed processing. Therefore, even variables affecting the `read` process itself are not effective before or during that `read`. For example, if you set the variable `hdlin_suppress_warnings` to a list of message codes in an embedded script, the assignment becomes effective only after the `read` command finishes, and does not affect the messages reported as the source file is read in.
>
> Refer to Chapter 12, “Processing the VHDL Source Files,” for information on using the `analyze` and `elaborate` commands as an alternative to the `read` command.

Design Compiler performs error checking after the `read` command completes processing. Errors in embedded `dc_shell` commands are therefore reported at the end of the `read` command.

Names in an embedded script are case-sensitive, unlike the rest of your VHDL file.