Proposal updated by Cliff Cummings - 11/20/2001
PROPOSAL: change all pragmas from "**rtl_synthesis**" to the simpler "**synthesis**"
PASSED: 20011106

Proposed by Cliff Cummings - 10/25/2001
PROPOSAL: Replace section 6 of the vlogrtl_D1.7 to the following:

Section 6 - additional work is required after the subset of synthesis attributes has been defined. I need feedback before investing more time in implementing this section.

Regards - Cliff

Pragmas

A *pragma* is a generic term used to define a construct with no predefined language semantics that influences how a synthesis tool should synthesize Verilog HDL code into a circuit. The only standard pragma style that may appear with the Verilog HDL code is a Verilog attribute instance.

6.1 Synthesis attributes

An attribute instance, as defined by the Verilog-2001 Standard (IEEE std 1364-2001) is a set of one or more comma separated attributes, with or without assignment to the attribute, enclosed within the reserved **(*** and ***)** Verilog tokens.

Per the Verilog-2001 Standard, “An attribute_instance can appear in the Verilog description as a prefix attached to a declaration, a module item, a statement, or a port connection. It can appear as a suffix to an operator or a Verilog function name in an expression.”

It is recommended that if a synthesis tool supports pragmas to control the structure of the synthesized netlist or to give direction to the synthesis tool, that attributes be used to convey the required information and that the first attribute within the attribute instance be **synthesis** followed by a comma separated list of synthesis-related attributes. For example:

(* synthesis, <any_company_specific_attribute**=**value_or_optional_value> ***)**

The attribute **synthesis** shall be listed as the first attribute in an attribute instance. By placing the **synthesis** attirbute first, a synthesis tool can more easily parse the attribute instance to determine if the rest of the attributes in the attribute instance are intended for the synthesis tool or for a different tool.

The following is a list of synthesis attributes that are considered part of the Verilog Language Synthesisizable Subset Standard and their functionality is described in the remainder of Section 6. If the attribute has an <optional_value>, the value does not change the behavior of the attribute. If the attribute has a <value>, the a value is required for this attribute. Some of the required values are described below. Additional vendor-specific attributes and attribute values may exist.

Cliff-Note: We might want to standardize a large number of the vendor synthesis attributes, provided that the vendors do not object to the standard using the attributes, just so multiple vendors do not implement the same attribute with different names. The exact descriptions will be added when the VSIWG decides which attributes should really be added.

This is a list of most of the common directives (in some cases the names have been modified but naming can be discussed by the VSIWG).

(* synthesis, full_case [**=** <optional_value>] ***)**
(* synthesis, parallel_case [**=** <optional_value>] ***)**
~~**(* synthesis, translate_off** [**=** <optional_value>] ***)**~~
~~**(* synthesis, translate_on** [**=** <optional_value>] ***)**~~

This is a list of most of the current Synopsys directives (in some cases the names have been modified but naming can be discussed by the VSIWG).

(* synthesis, fsm_state_vector = <value> ***) - value is the one item is the list - no synthesis interpretation implied**

(* synthesis, fsm_enum = <value> ***)**

(* synthesis, async_set_reset = <value> ***)**
(* synthesis, sync_set_reset = <value> ***)**
(* synthesis, one_cold = <value> ***)**
(* synthesis, one_hot = <value> ***)**
(* synthesis, infer_multibit = <value> ***)**
(* synthesis, dont_infer_multibit = <value> ***)**
(* synthesis, label [**=** <optional_value>] ***)**
(* synthesis, infer_mux [**=** <optional_value>] ***)**
(* synthesis, resource = <value> ***)**
(* synthesis, map_to_module = <value> ***)**
(* synthesis, implementation = <value> ***)**
(* synthesis, operator_on_line = <value> ***)**

This is a list of most of the current Synplicity directives (in some cases the names have been modified but naming can be discussed by the VSIWG).

~~**(* synthesis, ram_style =** <value> ***)**~~
~~**(* synthesis, rom_style =** <value> ***)**~~
~~**(* synthesis, logic_block** [**=** <optional_value>] ***)**~~
(* synthesis, black_box [**=** <optional_value>] ***) - combination of don't_touch and don't even look at it - ask Synplicity**

(* synthesis, fsm_encoding [**=** <value>] ***)**
(* synthesis, keep_signal [**=** <optional_value>] ***)**
(* synthesis, preserve_signal [**=** <optional_value>] ***)**
(* synthesis, resource_sharing [**=** <optional_value>] ***)**
(* synthesis, fsm_extract [**=** <optional_value>] ***)**
(* synthesis, port_probe [**=** <optional_value>] ***)**
(* synthesis, no_prune [**=** <optional_value>] ***)**

Multiple comma separated synthesis attributes may be added to the same attribute instance without repeating the keyword synthesis before each additional attribute. Example:

```
(* synthesis, full_case, parallel_case *)
case (state)
  ...
endcase
```

Note that in the above example, the use of the full_case and parallel_case attributes is generally not recommended.

The Verilog-2001 Standard also allows multiple attribute instances to be placed before legal, attribute-prefixed statements. Example:

```
(* synthesis, full_case, parallel_case *)
(* synthesis, fsm_encoding = gray *)
case (state)
  ...
endcase
```

Note that in the above example, all three attributes could also have be placed in the same attribute instance.

```
(* synthesis, resource = "r0", map_to_module = "DW01_INC",
   implementation = "cla", operator_on_line = "A1" *)
```

~~The Verilog-2001 Standard also allows multiple attribute instances to be placed before legal, attribute-prefixed statements.~~ Only synthesis attributes shall be placed in an attribute instance with other synthesis attributes. Non-sysnthesis attributes can be placed along with synthesis attributes before legal attribute prefixed statements and no predetermined placement-order of mixed synthesis and non-synthesis attribute instances shall be imposed by synthesis tools.

6.2 Metacomments deprecated

(Note: section 6.2 - PASSED: 20011106)

Prior to the acceptance of the Verilog-2001 Standard (IEEE Std 1364-2001), it was common practice to include synthesis pragmas embedded within a comment, for example: `// synthesis full_case`. The practice of embedding pragmas into a comment meant that any synthesis tool that accepted such pragmas were required to partially or fully parse all comments within a Verilog RTL design just to determine if the comment contained a directive to the synthesis tool.

The Verilog-2001 Standard introduced attributes to discourage the practice of putting directives into comments and to replace them with a set of tokens (attribute delimiters) that could then be parsed for tool-specific information.

The practice of putting pragmas into comments is highly discouraged and deprecated for the Verilog Synthesis Standard. Tool vendors are encouraged to take advantage of the new attribute mechanism introduced by and incorporated into Verilog-2001.

6.3 Compiler directives and implicit-synthesis defined macros

(Note: section 6.3 - PASSED: 20011106)

Synthesis tools shall define a Verilog macro definition for the macro named **`SYNTHESIS`** before reading any Verilog synthesis source files. This is equivalent to adding the following macro definition to the front of a Verilog input stream:

```
`define SYNTHESIS
```

This macro definition makes it possible for Verilog design engineers to add conditionally compiled code to their design that will be read and interpreted by synthesis tools but that by default will be ignored by simulators (unless the Verilog simulation input stream also defines the SYNTHESIS text macro).

Example:

```
module ram (q, d, a, clk, we);
```

```
    output [7:0] q;
    input  [7:0] d;
    input  [6:0] a;
    input  clk, we;

    `ifndef SYNTHESIS
      // RTL model of a ram device for pre-synthesis simulation
      reg [7:0] mem [127:0];
      always @(posedge clk) if(we) mem[a] <= d;

      assign q = mem[a];
    `else
      // Instantiation of an actual ram block for synthesis
      xram ram1 (.dout(q), .din(d), .addr(a), .ck(clk), .we(we));
    `endif
  endmodule
```

The use of the above conditional compilation capability removes the need to use the deprecated "translate_off/translate_on" synthesis directives.

6.4 ~~Conditional compilation attributes~~"translate_on/translate_off" pragmas deprecated

Prior to approval of this Standard (IEEE Std 1364.1), it was common practice to include to directives to instruct the synthesis tool to ignore a block of Verilog code enclosed by "translate_off/translate_on" synthesis pragmas.

The practice of a synthesis tool ignoring Verilog source code by enclosing the code within "translate_off/translate_on" pragmas is highly discouraged and deprecated for the 1364.1 Standard. Users are encouraged to take advantage of SYNTHESIS macro definition and `ifdef SYNTHESIS and `ifndef SYNTHESIS compiler directives (see section 6.3) to exclude blocks of Verilog code from being read and compiled by synthesis tools.

~~Two attributes provide for conditional compilation control.~~

~~a) **(* synthesis, translate_off [= <optional_value>] *)**~~

~~b) **(* synthesis, translate_on [= <optional_value>] *)**~~

~~The optional value shall be ignored by synthesis tools.~~

~~A synthesis tool should ignore any attribute instance (other than **(* synthesis, translate_on *)**) or any Verilog HDL construct after the **(* synthesis, translate_off *)** directive and before the **(* synthesis, translate_on *)** directive. The attributes can appear in any combination of upper- and lower-case letters and can be inserted in any place where it is legal to insert a Verilog attribute.~~

6.5 Case decoding attributes

(Note: section 6.2 - PASSED: 20011106)

The following attributes shall be supported for decoding **case** statements and shall have the following interpretations:

a) **(* synthesis, full_case** [**=** <optional_value>] ***)**

The optional value shall be ignored by synthesis tools.

This attribute shall inform the synthesis tool that for all unspecified case choices, the outputs assigned within the case statement can be treated as synthesis "don't care" assignments.

NoteWarning: This synthesis attribute is providing different information to the synthesis tool than is known by the simulation tool and can cause a pre-synthesis simulation to differ with a post-synthesis simulation.

NoteWarning: Contrary to popular belief, this synthesis attribute does not remove all latches that could be inferred by a Verilog case statement. If multiple outputs are assigned by the specified case items, but not all outputs are assigned by all of the specified case items, a latch will be inferred even if the full_case attribute has been added to the case statement..

NoteWarning: Adding a default statement to a case statement negates the effect of the full_case attribute.

NoteWarning: The use of the full_case synthesis attribute is generally discouraged.

b) **`(* synthesis, parallel_case`** [**`=`** <optional_value>] **`*)`**

The optional value shall be ignored by synthesis tools.

This attribute shall inform the synthesis tool that all case items are to be tested, even if more than one case item could potentially match the case expression.

NoteWarning: This synthesis attribute is providing different information to the synthesis tool than is known by the simulation tool and can cause a pre-synthesis simulation to differ with a post-synthesis simulation.

NoteWarning: Verilog case statements can have overlapping case items (a case expression could match more than one case item), and the first case item that matches the case expression will cause the statement for that case item to be executed and an implied break insures that no other case item will be tested against the case expression for the current pass through the case statement. The Verilog statement for the matched case item is the only Verilog code that will be executed during the current pass of the case statement.

NoteWarning: The parallel_case attribute directs the synthesis tool to test each and every case item in the case statement every time the case statement is executed. This directive causes the synthesis tool to remove and priority that might be assigned to the case statement by testing every case item, even if more than one case item matches the case expression. This behavior differs from the behavior of standard Verilog simulation.

NoteWarning: The parallel_case attribute is commonly used to remove priority encoders from the gate-level implementation of an RTL case statement. Unfortunately, the RTL case statement may still simulate like a priority encoder, causing a mismatch between pre-synthesis and post-synthesis simulations.

NoteWarning: Adding a default statement to a case statement does not negate the effect of the parallel_case attribute.

WarningNote: The use of the parallel_case synthesis attribute is generally discouraged. One exception is the careful implementation of a onehot Verilog state machine design.

c) **`(* synthesis, full_case, parallel_case *)`**

The full_case and parallel_case attributes may also appear as a single attribute instance, as shown above. The order in which they appear shall not be of importance.

Warning –Note: Strictly speaking, full_case should not be needed by any tool. It’s purpose is to communicate to the tool some information which is also available from alternative modeling styles. The

risk is that the user could be wrong about the ‘fullness’ of the case, and, if so, the results will not match simulation. For example,

```
always @(sel)
  (* synthesis, full_case *) case (sel)
    2’b01: out = op1;
    2’b10: out = op2;
    2’b11: out = op3;
  endcase
```

is synthesis-equivalent to the much safer,

```
always @(sel) begin
  out = ’bx;
  case (sel)
    2’b01: out = op1;
    2’b10: out = op2;
    2’b11: out = op3;
  endcase
end
```

6.6 RAM inference attributes

The following attributes may be supported to assist in the selection of the style of an inferred RAM device:

a) **(* synthesis, ram_block** [**=** <optional_value>] ***)**

~~The optional value shall be ignored by synthesis tools.~~

The <optional_value> may be ignored by synthesis tools.

~~Other **block_ram** values may be defined by synthesis tool vendors.~~

6.7 ROM inference attributes

The following attributes may be supported to assist in the selection of the style of an inferred ROM device:

a) **(* synthesis, rom_block** [**=** <optional_value>] ***)**

The ~~optional value~~<optional_value> ~~shall~~ may be ignored by synthesis tools.

~~The definitions of **block_rom**, <??? other ???> is target dependent and defined by the synthesis tool vendor.~~

~~Other **block_rom** values may be defined by synthesis tool vendors.~~

6.8 Logic inference attribute

The following attribute may be supported to assist to infer discrete logic for a particular RTL coding style as opposed to another type of logic such as ROMs or multiplexers:

a) **(* synthesis, logic_block** [**=** <optional_value>] ***)**

The optional value shall be ignored by synthesis tools.

6.9 <other> attributes

Descriptions to follow

Pragma List

Synplicity

syn_black_box	Defines a black box for synthesis.
syn_encoding	Specifies the encoding style for state machines.
syn_keep	Prevents the internal signal from being removed during synthesis and optimization.
syn_preserve	Prevents sequential optimizations across a flip-flop boundary during optimization, and preserves the signal.
syn_sharing	Specifies resource sharing of operators.
syn_state_machine	Determines if the FSM Compiler extracts a structure as a state machine.
translate_off/translate_on	Specifies sections of code to exclude from synthesis, such as simulation-specific code.
syn_probe	Drag a net to ports so that it can be probed.
syn_rom_style	specify the style of rom to infer.
syn_ramstyle	same as above, but for rams.
syn_noprune	Controls the automatic removal of instances that have outputs that are not driven.

Synopsys

state_vector	*vector_name*
enum	*enum_name*
return_port_name	*port_name*
template	
async_set_reset	"signal_name_list"
async_set_reset_local block_label	"signal_name_list"
async_set_reset_local_all	"block_label_list"
sync_set_reset	"signal_name_list"
sync_set_reset_local block_label	"signal_name_list"
sync_set_reset_local_all	"block_label_list"
one_cold	"signal_name_list"
one_hot	"signal_name_list"
dc_script_begin	
dc_script_end	
infer_multibit	"signal_name_list"
dont_infer_multibit	"signal_name_list"
label identifier	resource identifier
label_applies_to	LABEL