2

Description Styles

A Verilog circuit description can be one of two types: structural or functional. A structural description explains the physical makeup of the circuit, detailing gates and the connections between them. A functional description, also referred to as an RTL (register transfer level) description, describes what the circuit does.

This chapter covers the following topics:

- Design Hierarchy
- Structural Descriptions
- Functional Descriptions
- Mixing Structural and Functional Descriptions
- Design Constraints

- Register Selection
- Asynchronous Designs

Design Hierarchy

Synopsys HDL Compiler maintains the hierarchical boundaries you define when you use structural Verilog. These boundaries have two major effects:

- Each module specified in your HDL description is synthesized separately and maintained as a distinct design. The constraints for the design are maintained, and each module can be optimized separately in Design Compiler.
- Module instantiations within HDL descriptions are maintained during input. The instance name you assign to user-defined components is carried through to the gate-level implementation.

Chapter 3, “Structural Descriptions,” discusses modules and module instantiations.

Note:

HDL Compiler does not automatically maintain (create) the hierarchy of other, nonstructural Verilog constructs such as blocks, loops, functions, and tasks. These elements of an HDL description are translated in the context of their design. After reading in a Verilog design, you can use the `group -hdl_block` command to group the gates in a block, function, or task. For information on how to use the `group` command with Verilog designs, see the Synopsys `group` man page.

The choice of hierarchical boundaries has a significant effect on the quality of the synthesized design. Using Design Compiler, you can optimize a design while preserving these hierarchical boundaries. However, Design Compiler only partially optimizes logic across hierarchical modules. Full optimization is possible across those parts of the design hierarchy that are collapsed in Design Compiler.

Structural Descriptions

The structural elements of a Verilog structural description are generic logic gates, library-specific components, and user-defined components connected by wires. In one way, a structural description can be viewed as a simple netlist composed of nets that connect instantiations of gates. However, unlike in a netlist, nets in the structural description can be driven by an arbitrary expression that describes the value assigned to the net. A statement that drives an arbitrary expression onto a net is called a continuous assignment. Continuous assignments are convenient links between pure netlist descriptions and functional descriptions.

A Verilog structural description can define a range of hierarchical and gate-level constructs, including module definitions, module instantiations, and netlist connections. See Chapter 3, “Structural Descriptions,” for more information.

Functional Descriptions

The functional elements of a Verilog description are function declarations, task statements, and always blocks. These elements describe the function of the circuit but do not describe its physical makeup or layout. The choice of gates and components is left entirely to Design Compiler.

You can construct functional descriptions with the Verilog functional constructs described in Chapter 5, “Functional Descriptions.” These constructs can appear within functions or always blocks. Functions imply only combinational logic; always blocks can imply either combinational or sequential logic.

Although many Verilog functional constructs (for example, for loops and multiple assignments to the same variable) appear sequential in nature, they describe combinational-logic networks. Other functional constructs imply sequential-logic networks. Latches and registers are inferred from these constructs. See Chapter 6, “Register, Multibit, Multiplexer, and Three-State Inference,” for details.

Mixing Structural and Functional Descriptions

When you use a functional description style in a design, you typically describe the combinational portions of the design in Verilog functions, always blocks, and assignments. The complexity of the logic determines whether you use one or many functions.

Example 2-1 shows how structural and functional description styles are mixed in a design specification. In Example 2-1, the function `detect_logic` determines whether the input bit is a 0 or a 1. After making this determination, detect_logic sets ns to the next state of the machine. An always block infers flip-flops to hold the state information between clock cycles.

You can specify elements of a design directly as module instantiations at the structural level. For example, see the three-state buffer `t1` in Example 2-1. (Note that three-states can be inferred. For more information, refer to “Three-State Inference” on page 6-73.) You can also use this description style to identify the wires and ports that carry information from one part of the design to another.

Example 2-1 Mixed Structural and Functional Descriptions

```
// This finite-state machine (Mealy type) reads one
// bit per clock cycle and detects three or more
// consecutive 1s.

module three_ones( signal, clock, detect, output_enable );
input signal, clock, output_enable;
output detect;

// Declare current state and next state variables.
reg [1:0] cs;
reg [1:0] ns;
wire ungated_detect;

// declare the symbolic names for states
parameter NO_ONES = 0, ONE_ONE = 1,
          TWO_ONES = 2, AT_LEAST_THREE_ONES = 3;

// ************* STRUCTURAL DESCRIPTION ****************
// Instance of a three-state gate that enables output
three_state t1 (ungated_detect, output_enable, detect);

// *************I*** ALWAYS BLOCK *******************
// always block infers flip-flops to hold the state of
// the FSM.
always @ ( posedge clock ) begin
     cs = ns;
end
```

```
// ************* FUNCTIONAL DESCRIPTION ****************
function detect_logic;
    input [1:0] cs;
    input signal;

    begin
        detect_logic = 0;    //default value
        if ( signal == 0 )   //bit is zero
            ns = NO_ONES;
        else                 //bit is one, increment state
            case (cs)
                NO_ONES: ns = ONE_ONE;
                ONE_ONE: ns = TWO_ONES;
                TWO_ONES, AT_LEAST_THREE_ONES:
                         begin
                             ns = AT_LEAST_THREE_ONES;
                             detect_logic = 1;
                         end
            endcase
    end
endfunction

// ************** assign STATEMENT **************
assign ungated_detect = detect_logic( cs, signal );
endmodule
```

For a structural or functional HDL description to be synthesized, it must follow the Synopsys synthesis policy, which has three parts:

- Design methodology
- Description style
- Language constructs

Design Methodology

Design methodology refers to the synthesis design process that uses HDL Compiler, Design Compiler, and Verilog HDL Simulator. This process is described in Chapter 1, “Introducing HDL Compiler for Verilog.”

Description Style

Use the HDL design and coding style that makes the best use of the synthesis process to obtain high-quality results from HDL Compiler and Design Compiler. See Chapter 8, “Writing Circuit Descriptions,” for guidelines.

Language Constructs

The third component of the Verilog synthesis policy is the set of Verilog constructs that describe your design, determine its architecture, and give consistently good results.

Synopsys uses HDL constructs that maximize coding flexibility while producing consistently good results. Although HDL Compiler can read the entire Verilog language, a few HDL constructs cannot be synthesized. These constructs are unsupported because they cannot be realized in logic. For example, you cannot use simulation time as a trigger, because time is an element of the simulation process and cannot be realized. “Unsupported Verilog Language Constructs” on page B-21 lists these constructs.

Design Constraints

You can describe the area and performance constraints for a design module directly in your Verilog description. HDL Compiler inputs constraints specified for a design when they are embedded in a Synopsys-defined HDL Compiler directive. By specifying constraints with your HDL description,

- You can control the optimization of a design module from within the Verilog description. Design Compiler attempts to optimize each module so that all design constraints are met.
- You can use the Verilog description to document important specification information.

Chapter 9, “HDL Compiler Directives,” covers HDL Compiler directives in detail.

Register Selection

The clocking scheme and the placement of registers are important architectural factors. There are two ways to define registers in your Verilog description. Each method has specific advantages.

- You can directly instantiate registers into a Verilog description, selecting from any element in your ASIC library.

  Clocking schemes can be arbitrarily complex. You can choose between a flip-flop and a latch-based architecture. The main disadvantages to this approach are that

  - The Verilog description is specific to a given technology, because you choose structural elements from that technology library. However, you can isolate the portion of your design with directly instantiated registers as a separate component (module) and then connect it to the rest of the design.
  - The description is more difficult to write.

- You can use some Verilog constructs to direct HDL Compiler to infer registers from the description.

  The advantages to this approach directly counter the disadvantages of the previous approach. With register inference, the Verilog description is much easier to write and is technology-independent. This method allows Design Compiler to select the type of component inferred, based on constraints. Therefore, if a specific component is necessary, use instantiation. Some types of registers and latches cannot be inferred.

See “Register Inference” on page 6-2 for a discussion of latch and register inference.

Asynchronous Designs

You can use HDL Compiler to construct asynchronous designs that use multiple or gated clocks. However, although these designs are logically and statistically correct, they may not simulate or operate correctly, because of race conditions.

“Synthesis Issues” on page 8-36 describes how to write Verilog descriptions of asynchronous designs.