Using DesignWare Building Block IP

Using DesignWare Building Block IP does not require you to alter your design flow. However, you must understand how to do the following:

- Set up your environment to use DesignWare Building Block IP
- Integrate DesignWare Building Block IP into your design
- Simulate and synthesize a design that contains DesignWare Building Block IP

This document provides the information you need to use DesignWare Building Block IP in your design. For more detailed information, including advanced topics such as managing the synthetic library cache, refer to the *DesignWare Building Block IP User Guide*.

DesignWare Building Block IP Directory Structure

Figure 1-1 depicts the Synopsys root directory structure that contains the components, packages, and other files required by DesignWare Building Block IP. The UNIX environment variable $SYNOPSYS points to the Synopsys root directory under which all Synopsys tools and DesignWare Building Block IP files are installed.

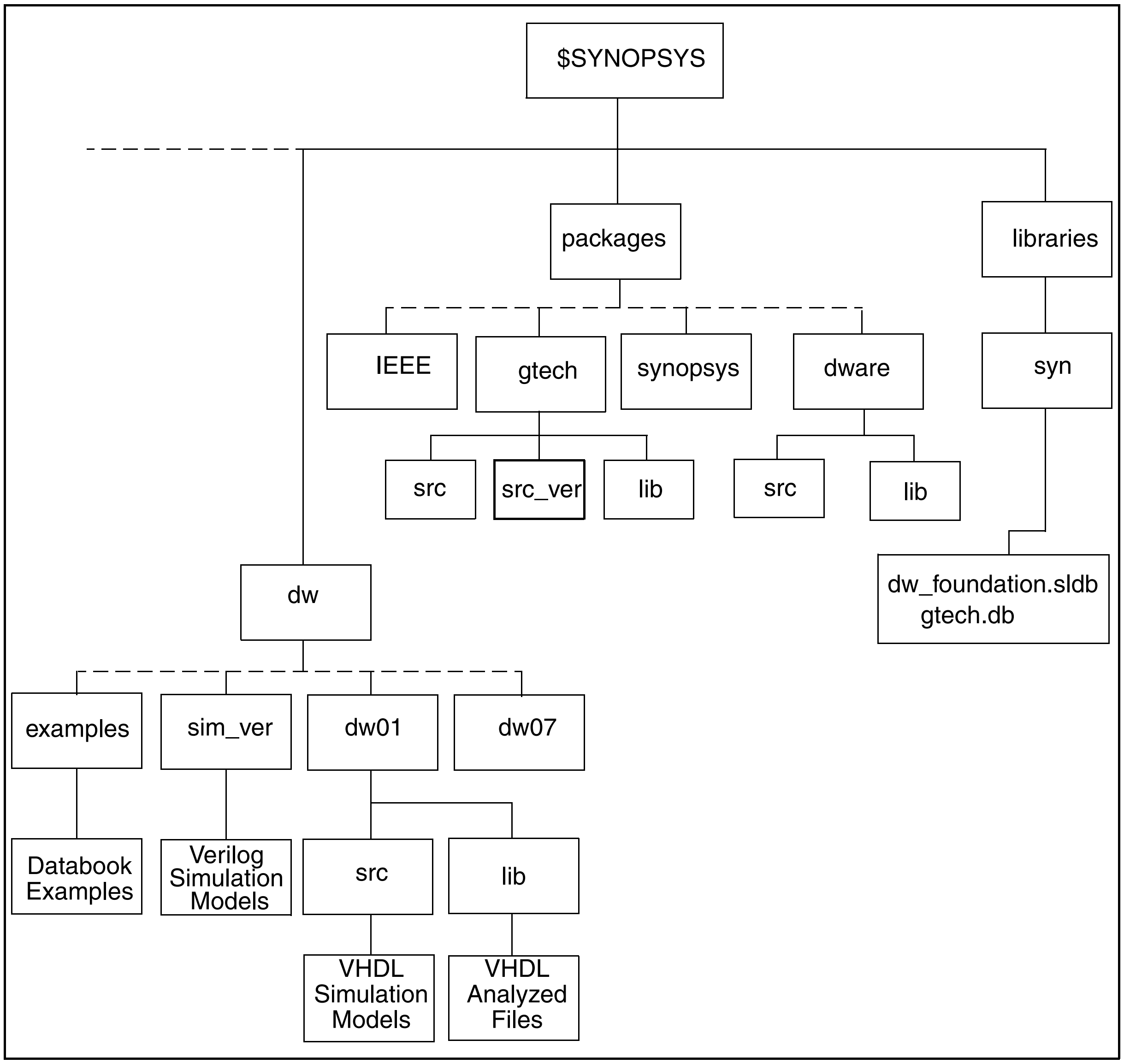
Synopsys recommends that references to files within this directory structure use the $SYNOPSYS environment variable so that they are easily relocated without requiring changes to setup files. Furthermore, the default Synopsys setup files reference the $SYNOPSYS environment variable.

DesignWare Building Block IP Files in $SYNOPSYS/dw/dwxx

The dw subdirectory under $SYNOPSYS contains one directory for each family of designs (dw01, dw02, and so forth). Each dw*xx* directory, in turn, contains the following subdirectories:

- src
  VHDL source for components packages, entities, simulation models, and encrypted source for synthesis architectures.
- src_ver
  Verilog source for simulation models. They are also linked to $SYNOPSYS/dw/sim_ver.
- lib
  Analyzed source files for simulation and synthesis.

Figure 1-1 Synopsys Root Directory Structure



Setting Up Your DesignWare Building Block IP Environment

To access and use DesignWare Building Block IP with Synopsys synthesis and simulation tools, you need to access the DesignWare Building Block IP synthetic libraries. This is done by setting the dc_shell-t variables synthetic_library and link_library.

Accessing the Synthetic Libraries

The DesignWare Building Block IP synthetic libraries contain the synthetic modules, operators, and bindings that link the DesignWare Building Block IP to Synopsys synthesis tools.

Set the dc_shell-t variables synthetic_library and link_library to include the synthetic libraries for the components you want to use, or all DesignWare Building Block IP.

The following code illustrates how to set the synthetic_library and link_library variables to include the synthetic libraries for individual DesignWare Building Block IP libraries DW01, DW02, DW03, DW04, DW06 and DW07. All DesignWare Building Block IP libraries are included when dw_foundation is used. You must set the variables for the libraries you want to access; they are not set by default.

Execute the following commands from the dc_shell-t command line, in a dc_shell-t command script, or in your .synopsys_dc.setup file:

```
set synthetic_library [list dw_foundation.sldb]
set link_library [concat $target_library $synthetic_library]
set search_path [concat $search_path [list \
   [format "%s%s" $synopsys_root "/dw/sim_ver"]]]
set synlib_wait_for_design_license [list "DesignWare"]
```

Integrating IP into Your Design

You can access DesignWare Building Block IP through HDL source code, either Verilog or VHDL. The mechanisms for accessing components are inferencing and instantiation.

Using Design Compiler (DC), any DesignWare Building Block IP can be instantiated in your HDL code. You can also infer all Combinational IP with HDL operators and/or functions. However, you cannot infer sequential components for DC. Examples are available in the $SYNOPSYS/dw/examples directory in ASCII format.

The datasheet for each IP provides examples of how to instantiate the IP in VHDL and in Verilog. If the IP is inferable, the datasheet also provides examples of how to infer the component for DC.

Currently, there is no graphical entry tool to support the entry of components.

Inferring DesignWare Building Block IP

DesignWare Building Block IP provides a mechanism to infer components through a series of mappings and bindings that relate an HDL operator or function to one or more IP. Inferred IP are subject to high-level synthesis optimization algorithms (resource sharing, arithmetic optimization, and implementation selection).

Operator Inferencing

To infer IP with a VHDL operator or Verilog operator, see the following examples. For both VHDL and Verilog, the port connections and parameter definitions are inferred from the context in which the IP is inferred.

VHDL:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity  DW01_add_oper is
  generic(wordlength: integer := 8);
  port(in1, in2 : in STD_LOGIC_VECTOR(wordlength-1 downto 0);
       sum      : out STD_LOGIC_VECTOR(wordlength-1 downto 0));
```

```
end DW01_add_oper;

architecture oper of DW01_add_oper is
  signal in1_signed, in2_signed, sum_signed: SIGNED(wordlength-1 downto 0);
begin
  in1_signed <= SIGNED(in1);
  in2_signed <= SIGNED(in2);
  -- infer the "+" addition operator
  sum_signed <= in1_signed + in2_signed;
  sum <= STD_LOGIC_VECTOR(sum_signed);
end oper;
```

Verilog:

```
module DW01_add_oper(in1,in2,sum);
  parameter wordlength = 8;

  input [wordlength-1:0] in1,in2;
  output [wordlength-1:0] sum;

  assign  sum = in1 + in2;
endmodule
```

Function Inferencing

For many Combinational IP, there are VHDL and Verilog functions that infer the IP. For example, the function named DWF_sin infers the component DW02_mac.

To infer IP with a VHDL function, refer to the following example. For VHDL function inferencing, the port connections and parameter definitions are inferred from the context in which the IP is inferred.

VHDL

```
library IEEE, DWARE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use DWARE.DW_foundation_arith.all;

entity DW02_mac_func is
  generic(wordlength1 : integer:=8; wordlength2: integer := 8);
  port(func_A   : in std_logic_vector(wordlength1-1 downto 0);
       func_B   : in std_logic_vector(wordlength2-1 downto 0);
       func_C   : in std_logic_vector(wordlength1+wordlength2-1 downto 0);
       func_TC  : in std_logic;
       MAC_func : out std_logic_vector(wordlength1+wordlength2-1 downto 0) );
end DW02_mac_func;

architecture func of DW02_mac_func is
```

```
begin

  process (func_A,func_B,func_C, func_TC)
  begin
    if func_TC = '1'  then
      MAC_func <= std_logic_vector(DWF_mac(SIGNED(func_A),
                                    SIGNED(func_B), SIGNED(func_C)) );
    else
      MAC_func <= std_logic_vector(DWF_mac(UNSIGNED(func_A),
                                    UNSIGNED(func_B), UNSIGNED(func_C)) );
    end if;
  end process;
end func;
```

To infer IP with a Verilog function, refer to the following example. For Verilog function inferencing, the port connections are inferred from the context in which the IP is inferred. However, you must pass parameter values to the function that infers the IP, and use a Verilog `` `include `` statement to include the file that defines the function, as shown below.

Verilog

```
module DW02_mac_func (func_A, func_B, func_C, func_TC, MAC_func);
  parameter func_A_width = 8;
  parameter func_B_width = 8;

  // Passes the widths to the multiplier-accumulator function
  parameter A_width = func_A_width;
  parameter B_width = func_B_width;

  // Please add search_path = search_path + {synopsys_root + "/dw/sim_ver"}
  // to your .synopsys_dc.setup file (for synthesis) and add
  // +incdir+$SYNOPSYS/dw/sim_ver+ to your verilog simulator command line
  // (for simulation).
  `include "DW02_mac_function.inc"

  input [func_A_width-1 : 0]               func_A;
  input [func_B_width-1 : 0]               func_B;
  input [func_A_width+func_B_width-1 : 0] func_C;
  input                                    func_TC;

  output [func_A_width+func_B_width-1 : 0] MAC_func;

  assign MAC_func = (func_TC) ? DWF_mac_tc(func_A, func_B, func_C) :
                                DWF_mac_uns(func_A, func_B, func_C);

endmodule
```

Instantiating DesignWare Building Block IP

DesignWare Building Block IP that are sequential cannot be inferred with Design Compiler; they must be instantiated. Combinational designs that can be inferred can also be instantiated. To instantiate IP, you specify an instance of a cell type, configure it with appropriate parameters, and interconnect it within your design. The following two examples show how to instantiate in VHDL and Verilog, respectively.

Instantiated IP are subject to implementation selection, but are not subject to resource sharing and arithmetic optimization.

VHDL

```
library IEEE,DWARE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_foundation_comp.all;

entity DW01_add_inst is
  generic ( inst_width : NATURAL := 8 );
  port ( inst_A    : in std_logic_vector(inst_width-1 downto 0);
         inst_B    : in std_logic_vector(inst_width-1 downto 0);
         inst_CI   : in std_logic;
         SUM_inst  : out std_logic_vector(inst_width-1 downto 0);
         CO_inst   : out std_logic );
end DW01_add_inst;

architecture inst of DW01_add_inst is
begin

  -- Instance of DW01_add
  U1 : DW01_add
  generic map ( width => inst_width )
  port map ( A => inst_A, B => inst_B, CI => inst_CI,
             SUM => SUM_inst, CO => CO_inst );
end inst;

-- pragma translate_off
configuration DW01_add_inst_cfg_inst of DW01_add_inst is
  for inst
  end for; -- inst
end DW01_add_inst_cfg_inst;
-- pragma translate_on
```

Verilog

```
module DW01_add_inst( inst_A, inst_B, inst_CI, SUM_inst, CO_inst );

  parameter width = 8;
```

```
    input [width-1 : 0] inst_A;
    input [width-1 : 0] inst_B;
    input inst_CI;
    output [width-1 : 0] SUM_inst;
    output CO_inst;

    // Instance of DW01_add
    DW01_add #(width)
      U1 (.A(inst_A), .B(inst_B), .CI(inst_CI), .SUM(SUM_inst), .CO(CO_inst) );

  endmodule
```

Parameter and Port Mapping

Component instantiations require the declaration of actual parameters and port connections. You can use name-based, explicit mapping, as illustrated for VHDL in the previous DW01_add_inst VHDL example on page 6, or position-based, implicit mapping, as illustrated for Verilog in the previous Verilog DW01_add_inst example. Synopsys recommends that you use name-based mapping to avoid erroneous connections. If you use position-based mapping, you must list the parameters and ports in exactly the same order that the parameters and ports are declared in the component entity file (DW*xx*_entity.vhd). For ease of use, all DesignWare Building Block IP datasheets list the ports in Table 1 - Pin Description, in the same order as found in their entity file.

Actual arguments must be passed as parameters; default values are not guaranteed. To understand the port and parameter (generic) definitions and their types, refer to the pin and parameter description tables in the appropriate datasheets.

Do not leave input pins open. Relying on synthesis to tie open input pins "high" or "low" may lead to inconsistent results between pre- and post-synthesis simulation. Stuck-at inputs are constants that are optimized away during boundary optimization, which is on by default for DesignWare Building Block IP. Outputs can either be connected or left "open."

Referencing Libraries for VHDL Simulation

The mechanism of accessing components in a library is different between synthesis and simulation. Synthesis does not require the declaration in the source HDL of a library containing components to be instantiated or inferred. Rather, synthesis relies on link_library and other declarations described in the topic titled "Accessing the Synthetic Libraries" on page 2.

To simulate DesignWare Building Block IP that are instantiated in VHDL, you must set the VHDL `library` and `use` constructs to access the components in the appropriate functional libraries.

DesignWare Building Block IP are declared, and thus accessed, through the DW*xx*_components packages. The `use` construct establishes a reference to a package through a logical file name, then a package name. The keyword `all` denotes that all exported declarations are to be included. The following example includes all DesignWare Building Block IP:

```
LIBRARY IEEE, DWARE, DW01, DW02, DW03, DW04, DW06, DW07;
USE IEEE.std_logic_arith.all, DW01.DW01_components.all,
    DW02.DW02_components.all, DW03.DW03_components.all,
    DW04.DW04_components.all, DW06.DW06_components.all,
    DW07.DW07_components.all, DWARE.DWpackages.all;
```

For Synopsys VCS MX, the design library is mapped to a UNIX directory using the corresponding design library declaration in the .synopsys_sim.setup file. This directory includes the components package. The specific functional library and package references required for VHDL are shown in the IP datasheet code examples.

Accessing DesignWare Building Block IP Usage Examples

Each DesignWare Building Block IP datasheet includes Verilog and VHDL code examples for inferring (if applicable) and instantiating components.

You can access DesignWare Building Block IP datasheets on the Web, where there is an easy-to-use search engine for locating components needed for your designs. Just use the "Search for IP" box in the upper left of this page:
http://www.synopsys.com/designware.

You can also access a complete listing of DesignWare Building Block IP and related documentation.

Extracting the Appropriate Code Example

All examples shown in DesignWare Building Block IP datasheets are located in your $SYNOPSYS/dw/examples directory. All examples are in ASCII format.

Simulating with DesignWare Building Block IP

You can simulate DesignWare Building Block IP referenced in your design in any of the following ways:

- Simulate the non-synthesizable, RTL simulation model that is provided for the IP VHDL or Verilog source code.
- Analyze and simulate any of the synthesis implementations provided for the IP. The code for the synthesis implementations is provided in encrypted VHDL format. Therefore, you must use Synopsys VCS MX to simulate the synthesis implementations.
- Compile the design to your target technology, and simulate the gate-level netlist (VHDL or Verilog). You can also compile the design to GTECH, and simulate the design using the GTECH simulation models.

Simulating with Gate-Level GTECH Models

Most DesignWare Building Block IP include behavioral source code simulation models.

For IP that do not have a simulation model available, you can create a gate-level VHDL or Verilog simulation model for your implementation of the IP. To do so, set your target library to GTECH, then elaborate and compile the IP with your parameter values and write out the gate-level netlist in either VHDL or Verilog.

For example, the following dc_shell-t script creates a Verilog simulation model for an instance of DW_ecc with the parameter values `width = 16`, `chkbits = 6`, and `synd_sel = 1`:

```
set target_library [list gtech.db ]
elaborate DW_ecc -param {16, 6, 1} -lib dw04   -arch str
compile -map_effort low
ungroup -all -flatten
write -f verilog -o DW_ecc_sim_16_6_1.v
```

The gate-level simulation model that you create references GTECH IP. The VHDL simulation models for GTECH IP are located in $SYNOPSYS/packages/gtech/src/*.vhd. The Verilog simulation models for GTECH IP are located in the directory:

$SYNOPSYS/packages/gtech/src_ver/

In this directory is a file that contains the models of all GTECH cells. That file is: gtech_lib.v

If you are using VCS MX, you can reference pre-analyzed VHDL simulation models for GTECH IP through `library` and `use` statements, as shown in the following example:

```
LIBRARY ;
USE GTECH.GTECH_components.all;
```

Simulating with RTL Source Code Simulation Models

For most IP, there are non-synthesizable simulation models available in VHDL and Verilog source code. You can simulate these simulation models using your chosen VHDL or Verilog simulator. For VCS MX users, there is also a pre-analyzed VHDL configuration for the IP's VHDL simulation model. The simulation models are written at the register-transfer level of abstraction for fast simulation.

Post-Synthesis Gate-Level Simulation

After your design has been synthesized, DesignWare Building Block IP are mapped into your target technology and integrated into the design. To simulate the design at this level, you can write the design out in the format of your choice, compatible with the simulator you will be employing.

At this point, you have a structural netlist containing interconnections of cells from the technology library that can be simulated on any simulation platform that interfaces to Design Compiler. (In general, this means any VHDL or Verilog simulator or gate-level simulator accepting VHDL/Verilog or EDIF 2.0.0.)

Layout and Back-Annotation

After floorplanning or layout, capacitance can be back-annotated to the design containing components in the same manner as a netlist.

Note that when laying out a design containing DesignWare Building Block IP, you may wish to specify individual IP as "soft groups." This means that cells contained within these modules should be physically located near one another. This is particularly true of multipliers, multiplier-accumulators, FIFOs, RAMs, registers, and so forth.

Issues Specific to VHDL Simulation

In VHDL, multiple architectures may be associated with one entity. For DesignWare Building Block IP, there is one simulation architecture and one or more synthesis architectures. You can select among architectures using configurations.

A configuration specifies an architecture to be bound to given instances of an entity. These configurations are analyzed after the source files on which they depend are analyzed.

Simulating with Synopsys VCS MX

For each IP, there is a pre-analyzed configuration for the RTL simulation model in the $SYNOPSYS/dw/dw*xx*/lib directory. These models are analyzed specifically for simulation with VCS MX. For example, to simulate the behavioral simulation model of DW01_add with VCS MX, use the pre-analyzed configuration $SYNOPSYS/dw/dw01/lib/DW01_ADD_CFG_SIM.

If you are using VCS MX, you can also analyze and simulate the encrypted RTL source code for any of the IP's synthesis implementations (for example, DW01_add_rpl.vhd.e).

VHDL Simulators Other Than VCS MX

All necessary packages, entities, and behavioral simulation architectures ("SIM") are supplied in VHDL source code to support VHDL simulators other than VCS MX. For example, to simulate a behavioral simulation model with a VHDL simulator other than VCS MX, analyze and simulate the VHDL source code for the simulation model (e.g., $SYNOPSYS/dw/dw01/src/DW01_add_sim.vhd).

All of the VHDL source files are IEEE VHDL 1076-1987 compliant.

The synthesis architectures are encrypted and, therefore, can only be analyzed and simulated with VCS MX. However, references to entities, other components, generics, and ports are retained in un-encrypted VHDL source code for compatibility and ease of debugging.

Components package, entities, and architectures are located in the src subdirectory of the associated DesignWare Building Block (DWBB) library. Logical library-to-path associations should be defined as described in the topic titled "Setting Up Your DesignWare Building Block IP Environment" on page 2, with the appropriate files, syntax, and so forth, used to reflect the requirements of your specific simulator.

DWBB components depend on VHDL packages IEEE and dware. Some architectures also depend on the gtech package and, possibly, other DWBB components, as shown in Figure 1-2.

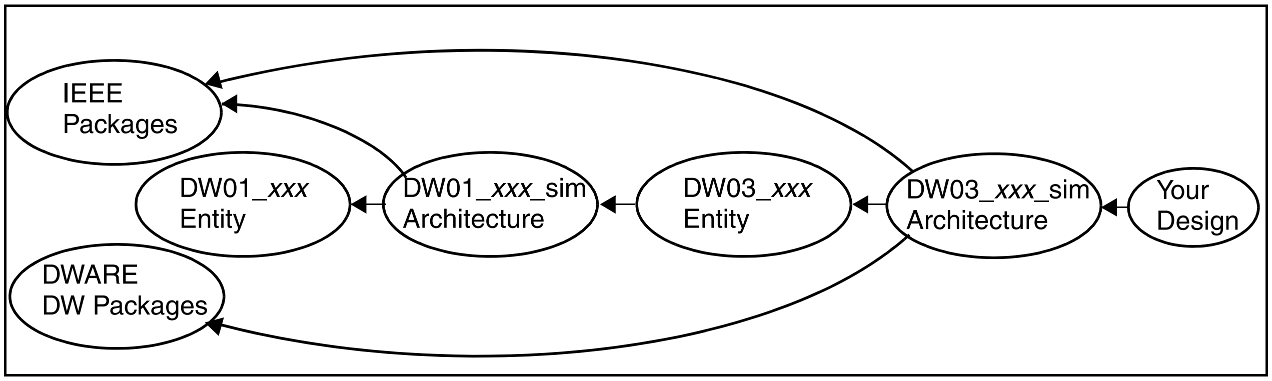
To resolve dependencies, analyze VHDL source files in the following order:

1. Packages (IEEE and dware) and GTECH, if needed.
2. DesignWare Building Block IP.
   - ❑ Analyze libraries in increasing order (for example, analyze DW01 first, then DW02, DW03, and so on).
   - ❑ Analyze entities before respective architectures.
3. Your design, in dependent order.

The DesignWare Building Block IP installation procedure analyzes the IP in the proper order for you.

Because some IP depend on others, all DesignWare Building Block IP should be analyzed once. It is assumed that the IEEE packages supplied with your simulator are used, since these may be specifically optimized.

Figure 1-2 Dependency Relationship Example



Issues Specific to Verilog Simulation

Verilog behavioral simulation models are supplied in Verilog source code (.v files).

The .v files are standard Verilog files that use default parameter values. To override the default parameter values, pass your desired parameter values when you instantiate the IP in your Verilog source code. See the individual IP datasheets for examples.

Verilog models are provided only for behavioral simulation. However, post-synthesis simulation can be supported by writing out a technology-specific Verilog netlist or RTL equations.

Verilog simulation models of GTECH cells are provided in Verilog source code in the file $SYNOPSYS/packages/gtech/src_ver/gtech_lib.v.

Simulating DesignWare Building Block IP

There is no need to modify the setup files if you are simulating with the VCS MX simulator because all the packages are in your default .synopsys_sim.setup file. Provide the following command line option to simulate DesignWare Building Block IP with Verilog simulators:

```
-y $SYNOPSYS/dw/sim_ver +libext+.v+ +incdir+$SYNOPSYS/sim_ver+
```

If your design includes GTECH cells, also include the following option:

```
-y $SYNOPSYS/packages/gtech/src_ver +libext+.v+
```

Compiling the Design

Set constraints on your design and compile using the strategy you are planning to use on the remainder of your circuit. Determine beforehand if you will use technology-specific DesignWare Building Block IP libraries from your ASIC or FPGA vendor. These libraries allow your RTL descriptions to infer technology-specific cells in the implementation selection process, if they are available. Therefore, components crafted with MacroCell from the target-technology are selected in a potentially more optimal way than those that are generic. Check with your vendor to determine if they support these libraries.

Using these libraries simply requires that they be installed, and that the synthetic_library and search_path dc_shell-t variables be set correctly. Specific details are available from ASIC/FPGA vendors who support DesignWare Building Block IP technology-specific libraries.

If you need to change your constraints after compilation, and consequently after implementations have been selected, then you have two options. You can do the following:

1. Read in the original RTL-level code, re-apply the new constraints and recompile, or
2. Use incremental implementation selection.

The implementation selection process, because it is processing an unmapped design, makes assumptions about the driving and driven circuitry to which a DesignWare Building Block IP interfaces. Incremental implementation selection is based on more accurate, mapped logic that interfaces to the IP to be evaluated. One restriction to this is that while paths can be evaluated, actual loads and drives to the IP are not considered; rather, estimates are employed.

For more information on incremental implementation selection, refer to the *DesignWare Building Block IP User Guide*.

For more information on compile strategies, refer to the following Synopsys publications:

- *(V)HDL Compiler Reference Manual*
- *Design Compiler Family Reference Manual*

Instance Name Generation

When a parameterized DesignWare Building Block IP is elaborated, an instance name is generated by the tool using the naming convention shown as follows:

DesignName_module_param1_value_paramn_value

DesignName - is the entity/module name of the containing design,
module - is the design name,
param1 - is the formal name of the parameter, and
value - is the actual parameter argument bound.

This instance name is important, as it represents a name-based tag that is used during other synthesis processes.

This can result in long instance names that may be inconvenient. Furthermore, if your ASIC vendor imposes name limitations and hashes names to shorter lengths, duplicate names may result. You can control the instance names by using the dc_shell-t command change_names after compiling to shorten the names. Or, you can set one or more of the following dc_shell-t variables to control the format in which the names are generated:

```
template_naming_style
template_parameter_style
template_separator_style
```

These generated names appear in your final netlist as you set them.

For example, changing the value of template_parameter_style from `%s%d` (the default) to `%d` results in names of the following format:

```
DesignName_module_p1value_pnvalue
```

For more information regarding change_names and the above dc_shell-t variables, enter "`help` *command_name*" at the dc_shell-t command line or refer to the appropriate command(s) in the *Design Compiler Family Reference Manual*.

Hierarchy

DWBB components may or may not be hierarchical, depending on the specific component and the value of parameters passed to it. Although dependent on the complexity of the function, instances with parameter widths of 4 or less generally result in flattened components. Parameter values greater than 4 introduce one level of hierarchy. Some hierarchical components are built with other DWBB components instantiated. Other hierarchical components employ additional hierarchy for better and faster synthesis results, and potentially result in an optimal layout. If you wish to remove one or more levels of hierarchy, execute the ungroup command after compiling the design.

Writing Out the Design

After you compile the design, create a netlist of the design for gate-level simulation. The approach here is the same as with any mapped design. If you do not require accurate timing in your simulation, simply execute the dc_shell-t write command, specifying the appropriate arguments.

This approach also works if you are using an external delay calculator, such as one provided by your ASIC vendor, to calculate and annotate unique, pre-layout delays into the netlist.

However, if you wish to use the delay calculation built into the Synopsys timing analyzer, DesignTime, then create a delay annotation file by executing the write_timing command with a VHDL or Verilog context, depending on the simulator you are using.

Reading Designs Back in After Layout

After floorplanning and/or layout, you will back-annotate the delays, based on actual wire and gate loading capacitance, into Design Compiler in order to:

- Perform static timing analysis, and
- Incrementally compile your design in synthesis to resolve hold or setup time violations, or constraint violations.

This can be accomplished if your layout tool, or the layout tool used by your vendor, writes Standard Delay Format (SDF) files or writes out a dc_shell-t script file that sets loads on all nets in the design.

If an SDF file is to be imported, execute the read_timing command using the appropriate arguments. Note that the interface to the layout tool is critical for optimal design results. Refer to the *Design Compiler Family Reference Manual* for more information on read_timing.

Test Considerations

IP are verified for DFT Compiler compliance and are designed to support high fault coverage for ATPG using a multiplexed flip-flop approach, unless noted. IP may work with other Design For Test (DFT) tools, especially those utilizing a multiplexed flip-flop approach. However, alternative schemes have not been verified.

The criterion for testability of IP is that ATPG render fault coverage of at least 95% for a standalone design. For certain IP, area has been traded off for test coverage. This is specified in the datasheets for the applicable IP. This means that for these IP, a high-testability equivalent that meets the DesignWare Building Block IP criteria is also available, though at a higher area cost.

Multiplexed flip-flop is the recommended scan style for DesignWare Building Block IP. Setting the dc_shell-t variable set_scan_configuration to "multiplexed_flip_flop" enables this style of scan.

To improve controllability and observability, some DesignWare Building Block IP have special requirements, such as specific ways in which to multiplex input select/control lines to achieve higher fault coverage. Specific test considerations are unique to certain IP, and are detailed at the end of the applicable datasheets.

Related Topics

- DesignWare Building Blocks Documentation Overview

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043

www.synopsys.com