

Logic Synthesis and Simulation

Dr. L.G. Johnson

1.0 Introduction

Logic synthesis with the Cadence Ambit synthesizer takes a circuit description given in behavioral (it is not necessary to synthesize structural descriptions) hardware description language (VHDL or Verilog) and produces a netlist (structural description) of an implementation of the circuit out of logic gates contained in a library. The Cadence Verilog simulator uses the netlist as input so that the function and timing of the circuit can be verified.

The synthesizer is also capable of meeting timing constraints imposed by the user. It always gives the minimum possible area that meets the timing constraint. If no timing constraint is given, it gives the minimum area circuit.

The time delay modeling done by Ambit is quite sophisticated taking into account the loading of other gates and the interconnect wiring delays. This level of accuracy is needed to be sure that the real circuit will meet the time delay constraints when it is fabricated in modern submicron silicon technology. Unfortunately, the amount of effort required of the synthesizer can be quite extensive when the synthesizer is required to pass on the timing information it uses (in a sdf file). For example, a 32 x 32 register file required 20 minutes of CPU time and created a 10MB sdf file.

In class work where design is not carried further than logic gate synthesis, we can use a simpler model of fixed delays for each logic gate. This saves considerable computer resources. It also allows the reuse of previously synthesized pieces of the circuit. Since loading effects are not modeled, one can just put together previously synthesized modules that already meet the timing constraints. This would not work if a real circuit is to be fabricated, since it may be necessary to resynthesize the entire circuit as a unit to be sure that loading effects have not violated timing constraints.

The examples in the following sections assume that the simplified fixed delay model is being used. The lower level behavioral modules are synthesized individually, then the structural netlists are combined in a higher level structural description without concern for loading effects.

Logic synthesis is done with the Cadence Ambit synthesizer using an interface program called the **bgx_shell**.

1. The `bgx_shell` environment is invoked by typing:
 `bgx_shell`
 at the unix prompt.

2. You should get the command prompt as shown below:
bgx_shell[1]
3. Respond with any unix command or an Ambit synthesis command.
4. After synthesizing your design, exit the bgx_shell by typing
exit

Alternatively, you can create a file, for example project.tcl, with the synthesis commands for a project and give it to the bgx_shell to execute by typing the following at the **unix** prompt.

```
bgx_shell < project.tcl
```

This method is advisable when synthesizing your circuit repeatedly as so often happens during the debugging process.

2.0 Synthesis for Combinational Logic

When synthesizing using the simplified timing model, it is usually best to separate out all registers and latches which leaves a purely combinational logic circuit. The timing constraints can then be specified as the maximum circuit delays from each input to each output. The Ambit synthesis commands in the following example specify the same worst case maximum delay from every input to every output.

```
do_remove_design -all
read_ver filename.v
read_vhdl filename.vhd
do_build_generic
set_data_arrival_time 0.0 [ find -input * ]
set_data_required_time 8.3 [find -output *]
read_alf filename.alf
set_wire_load_mode enclosed
do_optimize
write_ver -hier filename.v
```

The synthesis will be done and a hierarchical structural netlist will be formed in Verilog. The netlist is written in Verilog so that the Cadence Verilog simulator can be used to verify the functionality and speed of the circuit.

Note: The data arrival time (8.3ns above) and the data required time (0ns above) depend on the design that you are currently working on and are subject to change from one design to another as required by the project. It is the difference of these two numbers that determines the timing constraint.

Each of the commands written above are explained below:

1. do_remove_design -all

This serves to clear all the previous compilations of designs and makes the synthesizer re-compile each module.

2. `read_ver filename.v, read_vhdl filename.vhd`

Reads and loads the behavioral Verilog and VHDL files. This operation analyzes and parses the files.

Note: If there are any syntax or semantics errors in your code they will be displayed after the execution of these commands.

3. `do_build_generic`

This command performs the technology independent optimization (multi-level logic minimization).

4. `set_data_arrival_time 0.0 [find -input *]`

This means an edge on a circuit input arrives at the specified data arrival time. In this case edges arrive at all inputs at the same time.

5. `set_data_required_time 8.3 [find -output *]`

This means that the propagation delay time for any input to reach an output should be the specified required time (here it is 8.3ns). In this case the required time is the same for all outputs.

6. `read_alf filename.alf`

This loads the synthesis library file describing the logic gate library. The logic function of each gate is described along with timing information. Your instructor will tell you which library file to use.

7. `set_wire_load_mode enclosed`

This gives an estimate of the interconnect delay model to be used while building the circuit. This model uses a wire delay of zero.

8. `do_optimize`

This synthesizes the circuit by mapping the generic circuit into the logic gates in the library. The circuit is optimized to meet the timing constraints and minimize area.

9. `write_ver -hier filename.v`

This creates the hierarchical netlist in verilog format.

Sample Ambit synthesis commands for a 32 bit ALU are shown below.

```
do_remove_design -all
read_vhdl alu_cell.vhd
read_vhdl alu4.vhd
read_vhdl alu32.vhd
do_build_generic
set_data_arrival_time 0.0 [find -input *]
set_data_required_time 10 [find -output *]
read_alf ami350hxsc3.alf
set_wire_load_mode enclosed
```

```
do_optimize
write_hier -hier alu32.v
```

3.0 Synthesis for Latches and Registers

When using the simplified timing model, registers and latches are synthesized separately from combinational logic. Timing constraints are not imposed on latches and registers since they are already imposed on the combinational logic. The Ambient synthesis commands for a 32 bit register are as follows.

```
do_remove_design -all
read_vhdl reg32.vhd
do_build_generic
read_alf ami350hxsc3.alf
set_wire_load_mode enclosed
do_optimize
write_hier -hier reg32.v
```

Note that no timing constraints were given. The register delay and setup time can be used to modify the data arrival and data required times for the combinational logic. For example, suppose the alu32 module is to work with this register in a clock period of 10 nsec with a 0.5 nsec clock to Q delay and a 0.2 nsec setup time. The alu32 module should be re-synthesized with a 0.5 nsec data arrival time and a $10 - 0.2 = 9.8$ nsec data required time.

4.0 Simulation

After the behavioral modules have been synthesized, they can be put together in a high level structural module and synthesized. A test bench module will also be needed to provide circuit inputs and print or plot outputs. The following are the steps for simulating the design. We will use alu32 with registers added on the inputs and outputs as an example.

```
verilog regalu.test.v regalu.v reg32.v alu32.v \
-v /app1/cadence/ami/verilog/lib/ami350hxsc.lib
```

where

regalu.test.v is the test bench file,

regalu.v is the high level structural description of the circuit (written in Verilog) that puts the previously synthesized modules together,

reg32.v and alu32.v are the netlists (written in Verilog) for the previously synthesized modules,

and the “.lib” file describes the functionality and timing of the gates in the logic library.