ECE 3450: Digital Electronics
Fall 2009, Synopsys Design Compiler Tutorial

Dr. Pallav Gupta
pallav.gupta@villanova.edu

I Introduction

This tutorial provides a brief introduction to using Synopsys Design Compiler[1] - an industry-standard tool for synthesizing gate-level netlists from designs specified in a hardware modeling language such as Verilog or VHDL. We have done mostly custom design so far in this course. This is because the datapath of our simplified MIPS processor has a regular structure, and considerable effort can be spent to optimize it manually. However, the MIPS controller is a finite state machine that comprises of random logic. In practice, it is one of the most difficult part of a design. The controller logic changes every time there is a bug fix or new feature(s) added. Thus, it is cumbersome to keep redesigning the logic by hand!

The goal of synthesis is automatically synthesize the logic structure of a circuit, given some design constraints, and map that logic onto a set of cells (*i.e.*, the target technology library). Given this level of automation, the designer can focus his/her efforts on modeling the behavior of the circuit correctly.

Modern logic synthesis tools have evolved to a point that entire designs can be synthesized. For example, we could have synthesized the MIPS datapath as well. However, in microprocessor design, experienced logic and circuit designers are still employed to squeeze out every inch of performance possible in the datapath. The resulting design is significantly better than what CAD tools can currently produce. The cost is design time and labor. However, in application specific integrated circuits (ASICs) (*e.g.*, the chip in a cell phone, PlayStation, *etc*) design, the entire design may be synthesized and mapped using a standard cell library.

II Getting Started

You should have the following files available for this tutorial:

```
/
├─controller.v                    Verilog description of controller
└─std_vill.lib                   Villanova standard cell library
```

The `controller.v` contains the description of the controller to be synthesized. For reference, it is included in the Appendix.

The `std_vill.lib` contains the description, area, and timing specifications of the cells in our library. Recall, that we currently have the following cells: `std_nand2`, `std_nand3`, `std_nor2`, `std_nor3`, `std_inv`, `std_aoi`, `std_latch`. Open the file and examine its content carefully. The important parameters are the following:

1. Nominal voltage and temperature: These are the normal operating environment conditions of the cells. In this case, they are 5V and 25°C, respectively.
2. Units: All the parameters have units. For example, time is specified in `ns`, while voltage and current are specified in `V` and μ`A`, respectively.

[1] http://www.synopsys.com

3. Operating conditions: Various operating environment conditions are defined including worst case (WC) and best case (BC) scenarios. Furthermore, three categories are defined: commercial (COM), industrial (IND), and military (MIL). Synthesis results vary depending upon the targeted operating condition. For example, worst case commercial (WCCOM) has worst case voltage and temperature of 4.75V and 70°C (*i.e.*, the voltage on the circuit could drop to 4.75V while operating in an environment with temperatures reaching 70°C), respectively.
4. Wire load models: These models help the tool estimate the delays on interconnects (wires). They are developed through statistical information specific to a particular technology process. However, in submicron ($< 1\mu$m) technologies, these models are not very accurate. They will suffice for our purpose. Synopsys has developed "topographical" technology that allows the synthesis engine to do a "virtual" layout of the circuit to estimate wire delays.
5. Cell description: Each cell has a name, input/output ports, logic function, area, input capacitance and output timing parameters.

The synthesis tool uses this information to map the synthesized logic onto a a set of cells that meet the design constraints. Modern libraries contain hundreds of cells including multiplexers, latches and flip-flops, buffers, half/full adder, static ram (SRAM), *etc.* Typically, the library is provided by semiconductor vendors.

III Using Design Vision

The Synopsys tools are installed on k2.ece.villanova.edu. In order to run them, you will need to remotely login to the machine using an `ssh` client. If you are outside of Villanova, you will need to login to http://gateway.villanova.edu first to establish a secure connection before you can connect to `k2`.

If you are on a Windows system, you will also need to enable X11 forwarding/tunneling to allow GUI applications running remotely to display locally. Consult the documentation of your system. Software such as TightVNC (http://www.tighvnc.com) or MobaXVT (http://mobaxvt.mobatek.net/en/) might be of assistance. Linux or Mac OS X users do not have to worry about this.

This tutorial assumes you are operating from a Linux/Mac OS X machine. Type the following on a terminal:

```
# Connect to server. Use the user/pass assigned to you. -Y is to enable X11 forwarding.
[Kanchenjunga:~] pgupta% ssh -Y pgupta@k2.ece.villanova.edu
pgupta@k2.ece.villanova.edu's password:
Warning: No xauth data; using fake authentication data for X11 forwarding.
Last login: Wed May  6 21:53:03 2009 from 153.104.2.197

# Create a directory called controller.
[pgupta@k2 ~]$ mkdir -p controller

% Change directory.
[pgupta@k2 ~]$ cd controller

# Copy controller.v and std_vill.lib into this directory.
# If the files sit on your local machine, use a SSH FTP client to transfer them.
[pgupta@k2 controller]$ cp ../tmp/controller.v ../tmp/std_vill.lib .
[pgupta@k2 controller]$ ls -l
total 28
-rw-rw-r--  1 pgupta pgupta  8062 May  6 22:14 controller.v
-rw-r--r--  1 pgupta pgupta 10408 May  6 22:14 std_vill.lib

# Start Design Vision in GUI mode. Make sure you are in the controller directory.
[pgupta@k2 controller]$ design_vision -gui
```

The last command starts the Synopsys Design Vision Environment which is a collection of tools including Design Compiler. If your X11 forwarding is working properly, you should see a GUI pop up that looks like Fig. 1. The main components of the application are text-annotated. All of the viewers will not be visible because you have not loaded a design yet.

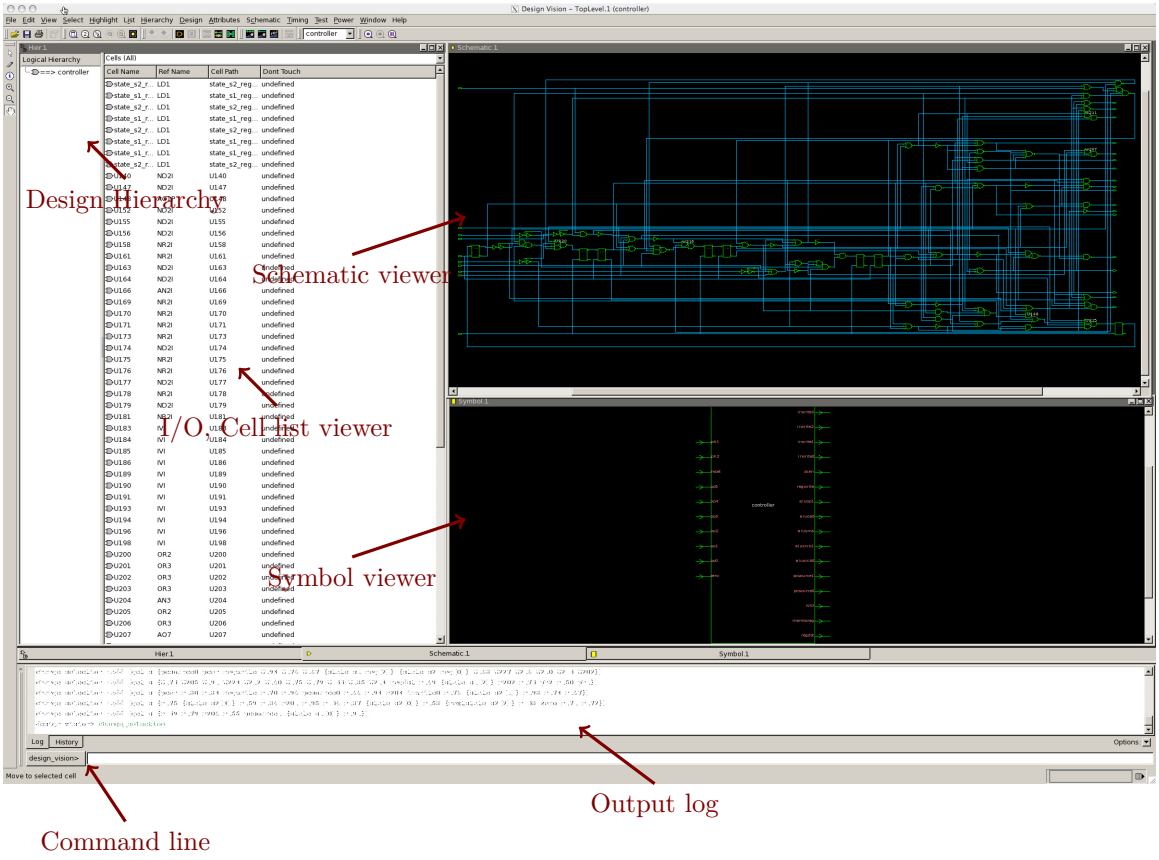

Figure 1: Synopsys Design Vision Environment.

.

III.A The Target Technology Library

Before we can perform synthesis, we need to do a little preparatory work and convert the cell library into a format that Synopsys can understand. This is done by reading the library and writing it back out into a different format. In order to do this, we need to supply some commands. There are two ways to input commands. You can either type them on the command prompt `design_vision>` at the bottom of the GUI application or you can enter them on the terminal in which you started Design Vision. You'll notice a `design_vision>` prompt there as well.

Type the following on the GUI command prompt:

```
1 design_vision> read_lib std_vill.lib
2 Reading '/home/pgupta/controller/std_vill.lib' ...
3 Technology library 'std_vill' read successfully
4 1
5 design_vision> write_lib -format db -output std_vill.db std_vill
6 Wrote the 'std_vill' library to '/home/pgupta/controller/std_vill.db' successfully.
7 1
```

The first command reads the cell library, while the second command writes it into `std_vill.db` using the Synopsys database format. If you check the `controller` directory, you should see this file present there now.

Type `help` to see the list of available commands. If you need to see usage information for a particular command, type `<command> -help`. For example, try `write_lib -help`. You can also view the online documentation at http://k2.ece.villanova.edu/syn. Note that the online documentation does not work properly in Firefox. Try a different browser.

III.B Performing Synthesis

In order to synthesize a design, we must perform the following steps:

1. Setup the target, link, and symbol libraries.
2. Read in a design.
3. Setup the process environment operating conditions.
4. Setup conditions on the input/output boundary of the design.
5. Setup optimization constraints.
6. Compile the design and map it using the technology library.
7. Perform incremental synthesis if necessary.
8. View the synthesis reports.

III.B.1 Setting up Libraries

To setup the libraries, choose File → Setup. A dialog box will open up. Replace the link, target, and symbol libraries with `'* std_vill.db'`, `'std_vill.db'`, and `'generic.sdb'`, respectively (without the quotes). The link library is used to define any technology input to the synthesis process. The `'*'` is necessary as it tells Design Compiler to search existing databases in its memory first. Do NOT remove it! The target library is the technology library. The symbol library provides icons for the cells of the target or link libraries in the Schematic Viewer. Finally, do not modify the search path. Otherwise, the tool will have difficulty finding the libraries. Click OK to apply the changes.

III.B.2 Reading the Design

Next, we can read in the design file using the `read_file` command. Since the source is a Verilog file, we need to let the command know the input format. Type the following and observe the command log.

```
design_vision> read_file -format verilog controller.v
Loading db file '/home/pgupta/controller/std_vill.db'
Loading db file '/usr/local/synopsys/current/synthesis/libraries/syn/gtech.db'
Loading db file '/usr/local/synopsys/current/synthesis/libraries/syn/standard.sldb'
  Loading link library 'std_vill'
  Loading link library 'gtech'
Loading verilog file '/home/pgupta/controller/controller.v'
Detecting input file type automatically (-rtl or -netlist).
Running DC verilog reader
Reading with Presto HDL Compiler (equivalent to -rtl option).
Running PRESTO HDLC
Compiling source file /home/pgupta/controller/controller.v
Warning:  /home/pgupta/controller/controller.v:110: Case statement is not a full case. (ELAB-909)

Statistics for case statements in always block at line 86 in file
        '/home/pgupta/controller/controller.v'
================================================
|           Line           |  full/ parallel  |
================================================
|            90            |    auto/auto     |
|           110            |    user/auto     |
================================================

Statistics for case statements in always block at line 130 in file
        '/home/pgupta/controller/controller.v'
================================================
|           Line           |  full/ parallel  |
================================================
```

```
|           148            |    auto/auto     |
===============================================

Inferred memory devices in process
        in routine controller line 80 in file
                '/home/pgupta/controller/controller.v'.
===============================================================================
|    Register Name    | Type  | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|    state_s1_reg     | Latch |   4   |  Y  | N  | N  | N  | -  | -  | -  |
===============================================================================

Inferred memory devices in process
        in routine controller line 82 in file
                '/home/pgupta/controller/controller.v'.
===============================================================================
|    Register Name    | Type  | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|    state_s2_reg     | Latch |   4   |  Y  | N  | N  | N  | -  | -  | -  |
===============================================================================
Presto compilation completed successfully.
Current design is now '/home/pgupta/controller/controller.db:controller'
Loaded 1 design.
Current design is 'controller'.
controller
Current design is 'controller'.
```

The tool reads in the libraries and the design file. It then compiles the design to generate an **initial** circuit. However, this circuit is a **starting** point for optimization. It is **not** the final result! Note that there is a warning displayed. In general, you should be **extremely paranoid** about any warnings. They usually indicate possible problems that will come back to haunt you later if you are not careful. In this case, it is safe to ignore the warning.

Now that the design has been read, go to the Logical Hierarchy panel in the GUI and select `controller`. Click on the icon Create Design Schematic. This will generate the current logic schematic of the circuit. Click on the icon Create Symbol View next to the the Create Design Schematic icon. This will create a symbol for the controller module with the input/output ports.

Go to the Cell List Viewer and select Cells (All). This will show all the cells contained in the current circuit. Note the reference names such as `GTECH_NOT`, `GTECH_OR2`, *etc.* These gates are clearly not present in our standard cell library. Design Compiler is currently using a generic technology library to represent the circuit. We have not optimized and and performed technology mapping using our library. If you select a particular cell, that cell will get highlighted in the schematic.

Go to the Cell List Viewer and select Pins/Ports. This will list all the inputs/outputs of the `controller` module. Selecting a pin will highlight in the schematic. Nets are the ultra hip term used in the chip design community for wires.

Now that you are familiar with the basic components of the GUI, spend some time browsing through the various menu items. Keep in mind that only a few of the commands are available through the GUI. The bulk of the commands are available via the command prompt. This is because the commands are typically written in scripts to automate the entire process. We will talk about this in Section III.E.

III.B.3 Setting up Operating Conditions

Choose Attributes → Operating Environment → Operating Conditions to setup the process environment. In the dialog box, ensure that Single is selected, `std_vill` is the Library, and WCCOM (worst case commercial) is the Condition. Observe the command log and you will see that typing **set_operating_conditions -library std_vill WCCOM** would have had the same effect.

Choose Attributes → Operating Environment → Wire Load and select 20x20. This is the wire model that will be

used to calculate net delays. Note that the command **set_wire_load_model -name 20x20 -library std_vill** would have had the same effect.

Remember that the process environment conditions are set for the top module in a design hierarchy only. In this case, we only have a single module.

III.B.4 Setting up Boundary Design Conditions

The next thing we need to do is setup boundary conditions at the inputs/outputs of the controller. We need to specify the drive strength for the input pins and the load being driven by the output pins. These conditions help the tool ensure that the synthesized circuit can drive the given output load. Since the output pins of the controller go into the zipper (which is responsible for driving the signals onto the datapath), it is not necessary for them to drive a big load. Thus, we can set an output load of 4 on these pins. For the inputs, we set the drive strength to 1.

It is very cumbersome to set the aforementioned conditions individually on each input/output pin. Thus, we make use of the command line to do this quickly. Type the following:

```
% Set the drive strength of all inputs to be 1.
design_vision> set_drive 1 [get_ports [all_inputs]]
1
% Set the output load of each output to be 4.
design_vision> set_load 4 [get_ports [all_outputs]]
1
```

III.B.5 Setting up Optimization Constraints

The final step is to set the optimization constraints. In our case, we want to minimize the delay of our circuit. In practice, there are many paths through a circuit. However, only a subset of these paths are actually **critical**. As the design progresses, designers usually know what the critical paths are, and focus their efforts on minimizing its delay (to increase the operation frequency).

In our case, we do not know what the critical path in the controller is. If we knew it, we would set the constraints on only those input/output pins that lie on the critical path! What we are going to do is tell the tool that it must optimize the circuit in such a way that the maximum delay from any input to output be no more than 10 units. It is possible that the tool fails to synthesize such a circuit. If that were the case we would relax the constraint (set it to say, 15) and repeat the process.

To set the optimization constraint, type the following:

```
% Set the maximum delay of the circuit to be 10.
design_vision> set_max_delay -from [get_ports [all_inputs]] -to [get_ports [all_outputs]] 10.0
1
```

III.B.6 Performing the Optimization and Technology Mapping

We can now do the actual optimization and technology mapping. Type the following:

```
% Perform optimization.
design_vision> compile -exact_map -map_effort high -area_effort high
Information: Evaluating DesignWare library utilization. (UISN-27)

============================================================================
| DesignWare Building Block Library       |      Version       | Available |
============================================================================
| Basic DW Building Blocks                | B-2008.09-DWBB_0902 |    *     |
| Licensed DW Building Blocks             |                     |          |
============================================================================


```

```
Warning: Operating condition WCCOM set on design controller has different process,
voltage and temperatures parameters than the parameters at which target library
std_vill is characterized. Delays may be inaccurate as a result. (OPT-998)

  Beginning Pass 1 Mapping
  ------------------------
  Processing 'controller'

  Updating timing information
Information: Updating design information... (UID-85)

  Beginning Mapping Optimizations  (High effort)
  -------------------------------

   ELAPSED            WORST NEG TOTAL NEG  DESIGN
    TIME      AREA      SLACK     SLACK   RULE COST         ENDPOINT
  --------- --------- --------- --------- --------- -------------------------
    0:00:00     159.0      0.00       0.0       0.0
    0:00:00     159.0      0.00       0.0       0.0
    0:00:00     159.0      0.00       0.0       0.0
    0:00:00     159.0      0.00       0.0       0.0
    0:00:00     159.0      0.00       0.0       0.0
    0:00:00     159.0      0.00       0.0       0.0
    0:00:00     159.0      0.00       0.0       0.0
    0:00:00     159.0      0.00       0.0       0.0


  Beginning Delay Optimization Phase
  ----------------------------------

   ELAPSED            WORST NEG TOTAL NEG  DESIGN
    TIME      AREA      SLACK     SLACK   RULE COST         ENDPOINT
  --------- --------- --------- --------- --------- -------------------------
    0:00:00     159.0      0.00       0.0       0.0


  Beginning Area-Recovery Phase  (cleanup)
  ------------------------------

   ELAPSED            WORST NEG TOTAL NEG  DESIGN
    TIME      AREA      SLACK     SLACK   RULE COST         ENDPOINT
  --------- --------- --------- --------- --------- -------------------------
    0:00:00     159.0      0.00       0.0       0.0
    0:00:00     159.0      0.00       0.0       0.0
    0:00:00     153.0      0.00       0.0       0.0
    0:00:00     149.0      0.00       0.0       0.0
    0:00:00     148.0      0.00       0.0       0.0
    0:00:00     147.0      0.00       0.0       0.0
    0:00:00     147.0      0.00       0.0       0.0
    0:00:00     147.0      0.00       0.0       0.0
    0:00:00     147.0      0.00       0.0       0.0
    0:00:00     147.0      0.00       0.0       0.0
    0:00:00     147.0      0.00       0.0       0.0
    0:00:00     147.0      0.00       0.0       0.0
    0:00:00     147.0      0.00       0.0       0.0
    0:00:00     147.0      0.00       0.0       0.0
    0:00:00     147.0      0.00       0.0       0.0
Loading db file '/home/pgupta/controller/std_vill.db'

  Optimization Complete
  ---------------------
1
Current design is 'controller'.
```

We told the tool to put in maximum effort in optimizing the circuit for best area while meeting the given delay constraints. We also specified that it should put in maximum effort in technology mapping. From the report, you can see that the tool generated a circuit with an area of 147. We could try running the same command

again on this circuit to further improve the result. This is called **incremental improvement** or **refinement**. However, we are content with this result.

You may have noticed that during the optimization, the Schematic Viewer window disappeared. Now that the circuit structure has changed, we need to generate a new schematic by clicking on the Create Design Schematic icon. Observe the structure of this circuit and you will realize that is looks significantly different from the original circuit. If you look at all the cells in the Cells (All) panel, you will observe that the reference name for each cell is that from our library only. They are all `std_*`.

III.C Viewing Reports

The tool generates various reports for the designer to examine to see synthesis results. Before trusting the results, it is worthwhile to check the design to ensure it has no errors. Type **check_design** to ensure that everything is OK.

You can view the reports from the various menu items in the GUI. Alternatively, type the following:

```
1
2  % Area report.
3  design_vision> report -area
4  report_area
5
6  ****************************************
7  Report : area
8  Design : controller
9  Version: B-2008.09-SP4
10 Date   : Thu May  7 02:58:40 2009
11 ****************************************
12
13 Information: Updating design information... (UID-85)
14 Library(s) Used:
15
16     std_vill (File: /home/pgupta/controller/std_vill.db)
17
18 Number of ports:               28
19 Number of nets:                103
20 Number of cells:                93
21 Number of references:           7
22
23 Combinational area:        107.000000
24 Noncombinational area:      40.000000
25 Net Interconnect area:      undefined  (Wire load has zero net area)
26
27 Total cell area:           147.000000
28 Total area:                 undefined
29 1
30 % Timing report
31 design_vision> report -timing
32
33 ****************************************
34 Report : timing
35         -path full
36         -delay max
37         -max_paths 1
38 Design : controller
39 Version: B-2008.09-SP4
40 Date   : Thu May  7 02:58:44 2009
41 ****************************************
42
43 Operating Conditions: WCCOM   Library: std_vill
44 Wire Load Model Mode: top
45
46   Startpoint: zero (input port)
47   Endpoint: pcen (output port)
48   Path Group: default
```

```
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  controller         20x20                 std_vill

  Point                                    Incr       Path
  -----------------------------------------------------------
  input external delay                     0.00       0.00 f
  zero (in)                                1.86       1.86 f
  U26/Y (std_aoi)                          2.43       4.29 r
  U25/Y (std_nand2)                        1.02       5.31 f
  pcen (out)                               0.00       5.31 f
  data arrival time                                   5.31

  max_delay                               10.00      10.00
  output external delay                    0.00      10.00
  data required time                                 10.00
  -----------------------------------------------------------
  data required time                                 10.00
  data arrival time                                  -5.31
  -----------------------------------------------------------
  slack (MET)                                         4.69


1
design_vision>
```

Spend some time studying the reports. The area report indicates the number of cells (93) used in the design. The total area of the circuit is 147 (not accounting for interconnect). The timing report indicates that the longest path in the circuit has a delay of 5.31 and goes from the input pin `in` via cells U26 and U2 to output `pcen`. There is a slack of 4.69 indicating that the circuit meets the specified timing requirements. If the slack were negative, it would indicate that the circuit fails to compute the outputs at the required time.

III.D Saving the Design, Netlist, and Constraints

The final step is to save the entire design should we wish to return to it later. Choose File → Save. This will save the design in `controller.ddc` in Synopsys format. This is equivalent to the command **write -hierarchy -format ddc**.

We also have the option of saving the structural circuit netlist as a Verilog or VHDL file. The netlist can then be fed into place-and-route tools such as Electric or Cadence to place and route the design. We prefer to save the netlist in VHDL format. Choose File → Save As. On the dialog box, change the format from Auto to VHDL. Type `controller` in the filename. Choose OK. This should save the netlist in `controller.vhdl`. Browse through this file to see the netlist. We can accomplish the same with the command **write -hierarchy -format vhdl -output controller.vhdl**.

Finally, we should also save the constraints and design setup should we need to re-synthesize the controller from scratch. Choose File → Design Setup and supply the filename `controller`. A file named `controller.dc` will be created. Browse through the file and compare it with everything we did above. The same can done with the command **write_script > controller.dc**.

III.E Writing Synthesis Scripts

Now that you have learned synthesis the hard way, it is time to learn the shortcuts. Designers usually have to synthesize a design many times to achieve convergence. This is because there are bug fixes in the logic, new features adfded, and the results do not match with post-synthesis layout. Thus, synthesis, technology mapping, and layout are an iterative process. It would be extremely painful for a designer to reenter the above commands each time he/she wanted to re-synthesize the design.

Designers usually write synthesis scripts that automate the procedure described above. The script is nothing but a text file containing the above commands. Once the script is written, it can be executed by the tool. In this way, it easy to automate the process. Furthermore, it is also possible to have different synthesis scripts with varying parameters that produce different results. This can be useful in analyzing trade-offs in a design to pick the best one suited for the designer's needs.

First, create a hidden file named `.synopsys_dc.setup` in the `controller` directory. This file is read during tool initialization to setup the libraries and search path. The contents of this file should be the following:

```
# Setup the search path, target/symbol/link libraries.
set search_path "$search_path"
set target_library "std_vill.db"
set link_library "* std_vill.db"
set symbol_library "generic.sdb"
```

Next, create a script file named `controller.scr` in the same directory. The contents of this file should be the following:

```
# Controller synthesis script
# Read design
read_file -format verilog controller.v
# Setup process environment
set_wire_load_model -name 10x10 -library std_vill
set_operating_conditions -library std_vill WCCOM
# Setup conditions on design boundary
set_drive 1 [get_ports [all_inputs]]
set_load 4 [get_ports [all_outputs]]
# Setup optimization constraints
set_max_delay -from [get_ports [all_inputs]] -to [get_ports [all_outputs]] 10.0
# Compile the design
compile -exact_map -map_effort high -area_effort high
# Check the design
check_design
# View reports
report -area > area.txt
report -timing > timing.txt
# Save the netlist
write -hierarchy -format vhdl -output controller.vhdl
# Save design constraint setup
write_script > controller.dc
# Save the design
write -hierarchy -format ddc
```

Now, restart `design_vision`. Then, type **source controller.scr** on the command prompt to run the script. The tool will start executing each command. It will write out the area and timing reports in `area.txt` and `timing.txt`, respectively. The netlist and design constraints setup will also be saved. You should get the same results as before. If there are any errors, look at the command log and fix the problems.

IV Parting Words

Congratulations on finishing this tutorial. This should have given you a quick introduction to Synopsys Design Compiler and how to synthesize designs. It should also have provided you with some experience in writing scripts to automate tasks. You can now apply your knowledge in synthesizing larger designs!

V Courses Forum

The forum is located at http://pandim.ece.villanova.edu/phpbbforum. Please use it to ask questions.

VI Errors

I usually write precise tutorials and bug-free code. However, I am human (do not be surprised) and do make mistakes. In addition, CAD tools get updated frequently and the interface might change, rendering parts of the writeup ineffective. If you find any mistakes or inconsistencies while doing this tutorial, please bring it to my attention **immediately**.

Appendix

```
//                              -*- Mode: Verilog -*-
// Filename         : controller.v
// Description      : Description of the controller logic for our simplified MIPS processor.
// Author           : Pallav Gupta
// Created On       : Thu Oct 23 20:34:04 2008
// Last Modified On: Time-stamp: <2009-05-05 17:17:50 pgupta>
// Update Count     : 0
// Status           : Unknown, Use with caution!

// This module describes a controller for a multicycle MIPS processor like that
// given in Patterson and Hennessy. It contains the control FSM and the
// additional AND/OR logic for branching. It does not contain the alucontrol
// logic.


// Using busses would be cleaner, but is not supported well by the Electric
// Silicon Compiler.

// reference time unit is 1 nanosecond
`timescale 1ns/1ps

module controller(/*AUTOARG*/
   // Outputs
   memread, memwrite, irwrite3, irwrite2, irwrite1, irwrite0, pcen, regwrite,
   aluop1, aluop0, alusrca, alusrcb1, alusrcb0, pcsource1, pcsource0, iord,
   memtoreg, regdst,
   // Inputs
   ph1, ph2, reset, op5, op4, op3, op2, op1, op0, zero
   );

   input ph1, ph2;
   input reset;
   input op5, op4, op3, op2, op1, op0;
   input zero;
   output reg memread;
   output reg memwrite;
   output reg irwrite3, irwrite2, irwrite1, irwrite0;
   output     pcen;
   output reg regwrite;
   output reg aluop1, aluop0;
   output reg alusrca;
   output reg alusrcb1, alusrcb0;
   output reg pcsource1, pcsource0;
   output reg iord;
   output reg memtoreg;
   output reg regdst;

   // multicycle state machine state definitions
   parameter    FETCH1 = 4'b0000;  // instruction fetch (4 cycles, 8-bit datapath)
   parameter    FETCH2 = 4'b0001;
   parameter    FETCH3 = 4'b0010;
   parameter    FETCH4 = 4'b0011;
   parameter    DECODE = 4'b0100;  // instruction decode
   parameter    MEMADR = 4'b0101;  // memory address computation
   parameter    LBRD   = 4'b0110;  // load byte read
   parameter    LBWR   = 4'b0111;  // load byte writeback
   parameter    SBWR   = 4'b1000;  // store writeback
   parameter    RTYPEEX= 4'b1001;  // R-type instruction execution
   parameter    RTYPEWR= 4'b1010;  // R-type instruction writeback
   parameter    BEQEX  = 4'b1011;  // branch on equal execution
   parameter    JEX    = 4'b1100;  // jump execution
   // add extra states for ADDI here, use unique code (continue above sequence)
   parameter    ADDIEX = 4'b1101;
   parameter    ADDIWR = 4'b1110;
```

```
// instruction opcodes (do not modify!)
parameter    LB      = 6'b100000;
parameter    SB      = 6'b101000;
parameter    RTYPE   = 6'b000000;
parameter    BEQ     = 6'b000100;
parameter    J       = 6'b000010;
parameter    ADDI    = 6'b001000;

// internal signals
reg [3:0] nextstate_s2;
reg [3:0] state_s1, state_s2;
reg       pcwrite, pcwritecond;

// update state regiter (comprised of master/slave latch)
always @(/*AUTOSENSE*/nextstate_s2 or ph2) // update master latch
  if (ph2) state_s1 = nextstate_s2;
always @(/*AUTOSENSE*/ph1 or state_s1) // update slave latch
  if (ph1) state_s2 = state_s1;

// next state logic
always @(/*AUTOSENSE*/op0 or op1 or op2 or op3 or op4 or op5 or reset
         or state_s2)
  // state transition figure is given in the write up
  if (reset) nextstate_s2 = FETCH1; // synchronous reset
  else case (state_s2)
         // fetch
         FETCH1: nextstate_s2 = FETCH2;
         FETCH2: nextstate_s2 = FETCH3;
         FETCH3: nextstate_s2 = FETCH4;
         FETCH4: nextstate_s2 = DECODE;

         // decode, look at opcode
         DECODE: case ({op5, op4, op3, op2, op1, op0})
                   LB: nextstate_s2 = MEMADR;
                   SB: nextstate_s2 = MEMADR;
                   RTYPE: nextstate_s2 = RTYPEEX;
                   BEQ: nextstate_s2 = BEQEX;
                   J: nextstate_s2 = JEX;
                   // add support for ADDI here

                   default: nextstate_s2 = FETCH1;
                 endcase // case ({op5, op4, op3, op2, op1, op0})

         // memory adress compute (for load/store)
         MEMADR: case ({op5, op4, op3, op2, op1, op0}) // synopsys full_case
                   LB: nextstate_s2 = LBRD;
                   SB: nextstate_s2 = SBWR;
                   // no default needed becaus of full_case directive
                 endcase // case ({op5, op4, op3, op2, op1, op0})

         LBRD: nextstate_s2 = LBWR;
         LBWR: nextstate_s2 = FETCH1;
         SBWR: nextstate_s2 = FETCH1;
         RTYPEEX: nextstate_s2 = RTYPEWR;
         RTYPEWR: nextstate_s2 = FETCH1;
         BEQEX: nextstate_s2 = FETCH1;
         JEX: nextstate_s2 = FETCH1;
         // add support for ADDI here

         default: nextstate_s2 = FETCH1;
       endcase

// output logic
always @(/*AUTOSENSE*/state_s2)
  begin
     // provide default values for signals not specified
     memread = 0;
     memwrite = 0;
     irwrite3 = 0; irwrite2 = 0; irwrite1 = 0; irwrite0 = 0;
     pcwrite = 0;
     pcwritecond = 0;
     regwrite = 0;
     alusrca = 0;
     alusrcb1 = 0; alusrcb0 = 0;
     aluop1 = 0; aluop0 = 0;
     pcsource1 = 0; pcsource0 = 0;
     iord = 0;
     memtoreg = 0;
     regdst = 0;

     // specify the outputs for reach state according to FSM
```

```
case (state_s2)
  FETCH1: begin
     memread = 1;
     alusrca = 0;
     iord = 0;
     irwrite3 = 1; // endianness
     alusrcb1 = 0; alusrcb0 = 1;
     aluop1 = 0; aluop0 = 0;
     pcwrite = 1;
     pcsource1 = 0; pcsource0 = 0;
  end
  FETCH2: begin
     memread = 1;
     alusrca = 0;
     iord = 0;
     irwrite2 = 1; // endianness
     alusrcb1 = 0; alusrcb0 = 1;
     aluop1 = 0; aluop0 = 0;
     pcwrite = 1;
     pcsource1 = 0; pcsource0 = 0;
  end
  FETCH3: begin
     memread = 1;
     alusrca = 0;
     iord = 0;
     irwrite1 = 1; // endianness
     alusrcb1 = 0; alusrcb0 = 1;
     aluop1 = 0; aluop0 = 0;
     pcwrite = 1;
     pcsource1 = 0; pcsource0 = 0;
  end
  FETCH4: begin
     memread = 1;
     alusrca = 0;
     iord = 0;
     irwrite0 = 1; // endianness
     alusrcb1 = 0; alusrcb0 = 1;
     aluop1 = 0; aluop0 = 0;
     pcwrite = 1;
     pcsource1 = 0; pcsource0 = 0;
  end
  DECODE: begin
     alusrca = 0;
     alusrcb1 = 1; alusrcb0 = 1;
     aluop1 = 0; aluop0 = 0;
  end
  MEMADR: begin
     alusrca = 1;
     alusrcb1 = 1; alusrcb0 = 0;
     aluop1 = 0; aluop0 = 0;
  end
  LBRD: begin
     memread = 1;
     iord = 1;
  end
  LBWR: begin
     regdst = 0;
     regwrite = 1;
     memtoreg = 1;
  end
  SBWR: begin
     memwrite = 1;
     iord = 1;
  end
  RTYPEEX: begin
     alusrca = 1;
     alusrcb1 = 0; alusrcb0 = 0;
     aluop1 = 1; aluop0 = 0;
  end
  RTYPEWR: begin
     regdst = 1;
     regwrite = 1;
     memtoreg = 0;
  end
  BEQEX: begin
     alusrca = 1;
     alusrcb1 = 0; alusrcb0 = 0;
     aluop1 = 0; aluop0 = 1;
     pcwritecond = 1;
     pcsource1 = 0; pcsource0 = 1;
  end
```

```
            JEX: begin
               pcwrite = 1;
               aluop0 = 1; // not logically required, but a hack to ensure aluop0
                           // and pcsource0 aren't always identical.  If they
                           // were identical, Synopsys would optimize one away,
                           // which confuses the Silicon Compiler.
                pcsource1 = 1; pcsource0 = 0;
            end
            // add support for ADDI here

            default: begin
            end
         endcase // case (state_s2)
      end // always @ (state_s2)

   // compute pcen, the write enable for the program counter
   assign pcen = pcwrite | (pcwritecond & zero);
endmodule
```