Synopsys

- Will be used to synthesize RTL VHDL/Verilog to Verilog netlists
- ‘swsetup synopsys’ to place binaries on path
- Online documentation is all PDF /opt/ecad/synopsys/default/doc/online
- doc/online/library -- library compiler documentation
- doc/online/synth – synthesis tools compiler
  - doc/online/synth/dcug - design compiler user guide
  - doc/online/synth/dctut – design compiler tutorial
  - doc/online/synth/dcrm – design compiler reference manual

BR 6/00 1

Library File

- To synthesize an RTL Verilog/VHDL file to a netlist, need two input files:
  - The RTL file itself
  - A compiled library file (mylib.db). The .db format is a binary format used by Synopsys for compiled designs/libraries
- The library file specifies what cells are in your library, their functions, pin names, timing/power characteristics
  - The ASCII form of the library is usually has a ‘.lib’ extension.
  - The attached zip file has two sample libraries in it
  - More examples of libraries are under /opt/ecad/synopsys/default/libraries/syn/lsi_10k.lib

BR 6/00 2

.lib to .db Format

Before a library can be used, it must be compiled. The *dc_shell* program is a command line interface to Synopsys synthesis.

To compile a .lib file to .db format, do (i.e., sc_cadence.lib to sc_cadence.db)

```
%  dc_shell

dc_shell >   read_lib sc_cadence.lib

dc_shell >   write_lib sc_cadence

dc_shell >   quit

%
```

dc_shell scripts

- The most common usage of *dc_shell* is to give it a script file with *dc_shell* commands:

```
% dc_shell  -f    scriptfile
```

- The commands on the previous slide could be placed in a file called 'makelib.script':

```
read_lib sc_cadence.lib
write_lib  sc_cadence
quit
```

- To compile the library via the script, do:

```
% dc_shell  -f   makelib.script
```

RTL to Netlist of gates

dc_shell is also used to synthesize an RTL description to a netlist of gates. A sample script file is shown below:

```
link_library=target_library={sc_cadence.db}
read -f vhdl rtl/arbiter.vhd
dont_touch_network clk
compile -ungroup_all
write -f verilog -output gate/arbiter_cadence.v
quit
```

This script reads a VHDL file called 'rtl/arbiter.vhd', synthesizes it to the *sc_cadence* library, and writes the resulting gate-level netlist out as a verilog file.

The 'dont_touch_network clk' statement prevents buffers from being placed on the clock network. No synthesis constraints are specified, so minimum area is used.

Minimum Delay Constraint

```
link_library=target_library={sc_cadence.db}
read -f vhdl rtl/arbiter.vhd
dont_touch_network clk
max_delay 0.1 –from all_registers(-clock_pins)  -to all_registers(-data_pins)
compile -ungroup_all
write -f verilog -output gate/arbiter_cadence.v
report_timing –path short –delay max –from all_registers(-clock_pins)
                  –to all_registers(-data_pins) –max_paths 3 –nworst 1
quit
```

max_delay command used to set timing constratint. Here we are minimizing the register to register delay.

report_timing command used to report delays along selected paths.

Minimum Delay/ Max Fanout Constraints

```
link_library=target_library={sc_cadence.db}
read -f vhdl rtl/arbiter.vhd
dont_touch_network clk
max_delay 0.1 –from all_registers(-clock_pins)  -to all_registers(-data_pins)
compile -ungroup_all
dont_touch_network clk
set_max_fanout 8.0 find(design, arbiter)
max_delay 0.1 –from all_registers(-clock_pins)  -to all_registers(-data_pins)
compile –incremental_mapping
write -f verilog -output gate/arbiter_cadence.v
report_timing –path short –delay max –from all_registers(-clock_pins)
                  –to all_registers(-data_pins) –max_paths 3 –nworst 1
quit
```

Two step process. Units of *max_fanout* are whatever 'load units' were specified in the library file.   -*incremental_mapping* used to not repeat entire synthesis process.

Inserting IO Pads

For a complete chip design (and not just a standard cell block), I/O pads have to placed on the 'pins'. This can be done by reading the gate-level netlist, and using the 'insert_pads' command:

```
link_library=target_library={sc_cadence.db}
read -f verilog gate/arbiter_cadence.v
set_port_is_pad {clk reset breq bgrant bbusy}
insert_pads
verilogout_single_bit = true
write -f verilog -output gate/arbiterTop.v
quit
```

Requires that you have 'pad' cells in your library. *verilogout_single_bit* variable used to write busses out as individual signals in top level netlist interface.

Help information within *dc_shell*

- From within dc_shell can do:

```
dc_shell>   help  command
```

and will list help on that particular command.

Library File

The library definition file (i.e., sc_cadence.lib) is broken into two sections: a header section that defines attributes to be used by all cells in the library, and cell section that has a definition for each cell in the library.

A cell’s definition defines attributes about the cell such as pin names, area, functionality, timing, power, etc.

The following slides contains some examples from .lib files.

Cell Functionality – and2

```
cell (and2) {
  area : 434.7;
  pin(A1) {
    direction : input;
    capacitance : 2.141;
  }
  pin(B1) {
    direction : input;
    capacitance : 1.948;
  }
  pin(O) {
    direction : output;
    function : "A1 * B1";
   }
  }
```

pin type

input pin capacitance

output pin function

Timing has been removed from this example.

Cell Functionality – nand2

```
cell (nand2_4) {
  area : 359.1;
 pin(A1) {
    direction : input;
    capacitance : 12.547;
  }
  pin(B1) {
    direction : input;
    capacitance : 12.259;
  }
  pin(O) {
    direction : output;
    function : "(A1 * B1)'";
  }
 }
```

Timing has been removed from this example.

Single quote used for inversion

Cell Functionality - DFF

```
cell (dfr) {
  area : 4819.5;
  ff(IQ,IQN) {
    next_state : "DATA1";
    clocked_on : "CLK2’";
    clear : "RST3’";
  }
  pin(DATA1) {
    direction : input;
    capacitance : 51.289;
   }
  pin(CLK2) {
    direction : input;
    capacitance : 52.305;
  }
  pin(RST3) {
    direction : input;
    capacitance : 28.602;
  }
pin(Q) {
    direction : output;
    function : "IQ";
  }
```

Inversion on clock, falling edge triggerred.

Inversion on reset, low true reset.

BR 6/00 13

Hints on naming cells/pins

- Netlist produced by Synopsys may go through many different CAD tools on the way to silicon
  - Naming conventions between CAD tool differ, so need to be conservative in naming conventions
- Use all lower case
  - some CAD tools will accept mixed case, some will not.
- Do not use special symbols except for underscores
- Start with alpha character (a-z) as first character
- Keep names short, simple
- For pin names, do not end with a numeric, end with an alpha
  - some CAD tools will use an ending numeric value as a bus index
- Be consistent between cells

BR 6/00 14

Synopsys Timing Models

- Synopsys supports multiple timing models
- Example libs in .zip file use lookup table approach based on input slope, output capacitive load
  - This is known as the non-linear CMOS delay model, documentation is found in doc/online/library/lcug2 (look at *toc.pdf* first).
- Examples under Synopsys installation directories use RC delay approach.

Nonlinear Delay Model

- Two components to delay (Dtotal)
  - Dcell (delay due to cell itself)
  - Dconnect (delay due to interconnect)
- In this project will ignore the Dconnect term (it will be zero if we do not explicity specify a value for it).
- In computing the Dcell (Cell delay), can do this in two ways
  - As a sum of two delays: propagation time + transistion time. Both values are found from lookup tables.
  - As a single delay (cell_rise, cell_fall) read from a lookup table
  - Both delay models also compute an output transistion time for a signal in addition to delay.
- These slides will discuss the ‘single delay’ model.

Two Dimensional Lookup Table Template

The non-linear CMOS model can use a 1-D, 2-D, or 3-D lookup table. For 2-D tables, one axis must always be input transistion time, the 2nd axis can be different choices, one of which is output net total capacitance load.

Need to define a *template* that will be used to hold our lookup table data. The general form is:

```
lu_table_template(name) {
  variable_1 : value;
  variable_2 : value;
  variable_3 : value;
  index_1 ("float, ..., float");
  index_2 ("float, ..., float");
  index_3 ("float, ..., float");
}
```

Lookup Table Template Example

```
lu_table_template(t4x3) {
   variable_1: total_output_net_capacitance ;
   variable_2: input_net_transistion ;
   index_1 {"5,  20, 60, 200"} ;
   index_2 {"0.01,  0.1,  2.0"} ;

 }
```

The units of a table can be anything that you desire as long as your data is consistent with those units. Typical units for capacitance is fF, any time units can be used for input transistion time (ps or ns typically).

The above template represents a lookup table with 12 entries. For each cell delay data table, 12 spice measurements would need to be made.

Adding Timing Data to Cell Definition

Delay timing data is added to the output pin definition of a cell.   Values for both output delay and output transistion need to be added. For AND2 example:

```
pin(O) {
    direction : output;
    function : "A1 * B1";
    timing() {
     cell_rise(t4x3){
       values("0.740,0.755,0.768",
              "0.803,0.823,0.838",
              "0.918,0.939,0.954,0.967”,
              “1.439,1.497,1.554,1.610”);
      }
     cell_fall(t4x3) {values(....)};
     rise_transistion(t4x3) {values(....)};
     fall_transistion(t4x3) {values(....)};
     related_pin: “A1” ;
     }
   timing () {  cell_rise.... cell_fall..
                rise_transistion... fall_transition...
              related_pin: “B1” ;
            }
```

Timing Data Format

- The timing data forms the lookup table for the delay function.
  - Each grouping of data (enclosed in quotes) corresponds to values for index_1, with index_2 used to select a value from the group
  - In this example, there were 4 capacitive load values, 3 input transistion values, so there were 4 groups of data, with 3 data values per group
- cell_rise, cell_fall are tplh, tphl propagation delays
- rise_transition, fall_transition are the new output transition values for the output signal
- The indexes in the lookup table should be chosen to represent the range of values seen in a design
  - Do not count on accurate timing results if the algorithm has to interpolate outside of these ranges.

Setup/Hold Timing

Can either use a 1-dimensional or 2-dimensional lookup table for setup/hold timing.

For 2-dimensional table, the two axes are transition time on data pin, transition time on clock pin. Same template used for both setup and hold.

```
lu_table_template(dff3x3) {
   variable_1: constrained_pin_transition ;
   variable_2: related_pin_transition ;
   index_1 {“0.01,  0.1,  2.0”} ;
   index_2 {“0.01,  0.1,  2.0”} ;
 }
```

data input transition

clock input transition

Setup/Hold Timing (cont).

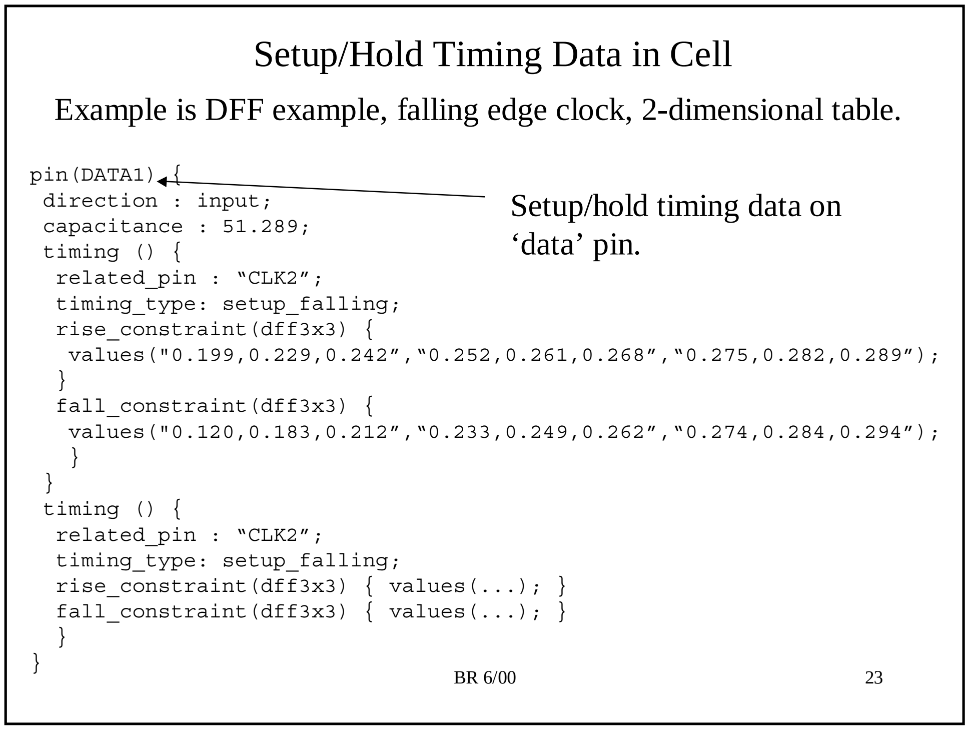
For 1-dimensional table, the axis is different depending on setup or hold time. Need seperate templates for setup/hold.

```
lu_table_template(setup_1d) {
   variable_1: constrained_pin_transition ;
   index_1 {“0.01  0.1  2.0”} ;
 }
```

For setup time, vary transition time on data input, use a fast transition time for clock

```
lu_table_template(hold_1d) {
   variable_1: related_pin_transition ;
   index_1 {“0.01  0.1  2.0”} ;
 }
```

For hold time, vary transition time on clock input, use a fast transition time for data



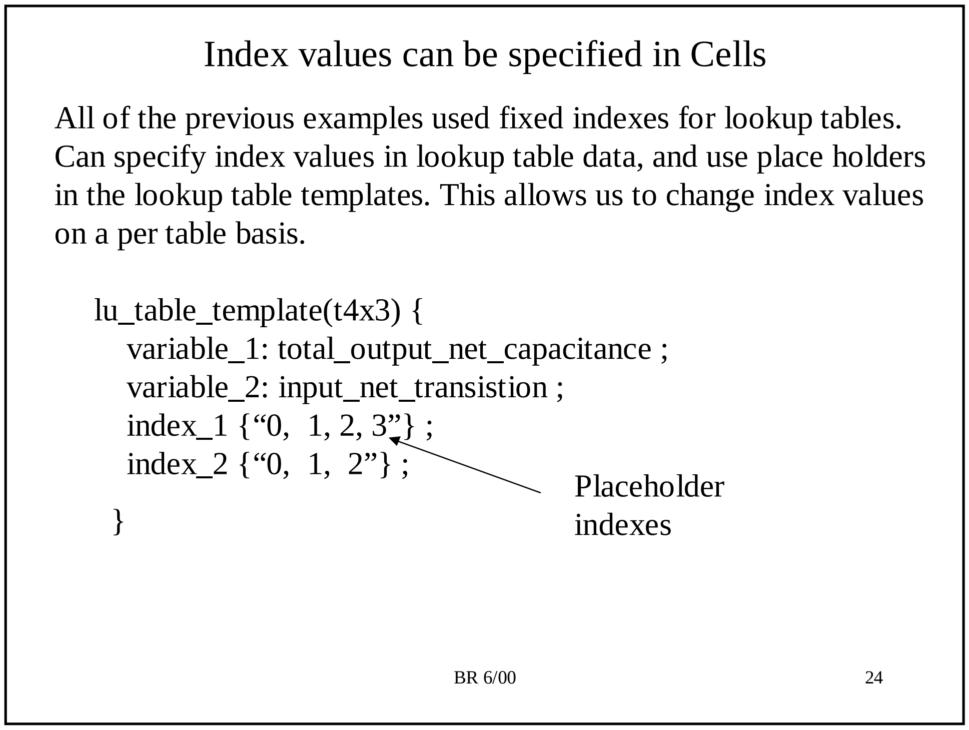
Setup/Hold Timing Data in Cell

Example is DFF example, falling edge clock, 2-dimensional table.

```
pin(DATA1) {
 direction : input;
 capacitance : 51.289;
 timing () {
  related_pin : “CLK2”;
  timing_type: setup_falling;
  rise_constraint(dff3x3) {
   values("0.199,0.229,0.242”,“0.252,0.261,0.268”,“0.275,0.282,0.289”);
  }
  fall_constraint(dff3x3) {
   values("0.120,0.183,0.212”,“0.233,0.249,0.262”,“0.274,0.284,0.294”);
   }
 }
 timing () {
  related_pin : “CLK2”;
  timing_type: setup_falling;
  rise_constraint(dff3x3) { values(...); }
  fall_constraint(dff3x3) { values(...); }
  }
}
```

Setup/hold timing data on ‘data’ pin.

BR 6/00 23



Index values can be specified in Cells

All of the previous examples used fixed indexes for lookup tables. Can specify index values in lookup table data, and use place holders in the lookup table templates. This allows us to change index values on a per table basis.

```
lu_table_template(t4x3) {
   variable_1: total_output_net_capacitance ;
   variable_2: input_net_transistion ;
   index_1 {“0,  1, 2, 3”} ;
   index_2 {“0,  1,  2”} ;
 }
```

Placeholder indexes

BR 6/00 24

Index values specified in Cell (cont.)

```
pin(O) {
    direction : output;
    function : "A1 * B1";
    timing() {
     cell_rise(t4x3){
       index_1("5, 20, 60, 200");
       index_2("0.01, 0.1, 2.0");
       values("0.740,0.755,0.768",
              "0.803,0.823,0.838",
              "0.918,0.939,0.954,0.967",
              "1.439,1.497,1.554,1.610");
      }
     cell_fall(t4x3) {values(....)};
     rise_transistion(t4x3) {values(....)};
     fall_transistion(t4x3) {values(....)};
     related_pin: "A1" ;
     }
   timing () {  cell_rise.... cell_fall..
                rise_transistion... fall_transition...
              related_pin: "B1" ;
           }
```

Index values specified here.

Index values need to also be specified here.

BR 6/00 25

Picking Index Values

- Important not to let tool interpolate outside of indexes
- Transition index choice
  - leftmost value could transistion time of fastest cell (largest inverter) with no load
  - rightmost value could transition time of weakest output drive cell driving largest expected load (synthesis library file should restrict maximum fanout so that fanout load is bounded).
- Capacitance index choice
  - leftmost value could be smallest pin capacitance value in library (or 10% to 20% lower than this for margin).
  - rightmost value should be largest pin capacitance value in library X maximum allowed fanout (may want to add 10% to 20% for margin).
- Number of indexes?
  - How much SPICE are you willing to do?
  - Most characterization systems are automated.

BR 6/00 26

Environmental Scaling Factors

- Temperature, Process, Voltage affect timing
- Two approaches
  - Specify multiplicative factors (*k-factors*) that modify timing values based on Process, Temperature, Voltage
  - Have different synthesis libraries with different timing tables for different environments (nominal, slow, fast).
- Many K-factors available in Synopsys
  - K-factors are timing model dependent (one set for nonlinear model, one set for linear model, etc..)
  - Default values of K-factors are 0 which means that timing, voltage, process values do not affect timing calculations
  - Will not consider environmental conditions in this project.

Input Pad Specification

```
cell (IPAD_1) {
    area : 2973.6 ;
     pad_cell : true;

  pin ( A ) {
    direction : input ;
    capacitance : 85 ;
     is_pad : true ;

  }
  pin ( Y ) {
    direction : output ;
    function : "A" ;
   }
 }
```

Indicates that this pin is a pad.

Timing data for output pin not shown.

Output Pad Specification

```
cell (OPAD_1) {
    area : 2973.6 ;
     pad_cell : true;

  pin ( A ) {
    direction : input ;
    capacitance : 278 ;
 }
  pin ( Y ) {
    direction : output ;
    function : "A" ;
    is_pad : true ;
    drive_current : 0.05 ;
    }
 }
```

Timing data not shown.

Output pad pin must have a ‘drive_current’ specification. Units are whatever the *current_unit* attribute is set in library header.

For your Synopsys library

- Need minimum cell set for Synopsys (nand2, nor2, inverter, DFF with async preset/clear), logic0, logic1.
  - No timing information needed for logic0, logic1 cells
- Need to demonstrate that the synthesized netlist simulation matches RTL simulation (functional specification is correct)
- Need accurate timing information for non-IO cells
- Need to be able to produce a netlist with/without IO cells
- For IO cells, only need an Input pad, Output pad (no tri-state)