Characterizing Cells and Writing a Technology Library File

Don Wichern

February 1, 2005

1 Introduction

The standard cell library we are creating needs to be integrated with the cadence synthesis tool, build gates. In order for build gates to correctly synthesize your circuit it needs to know timing and loading information about the standard cells in the library. You will need to generate simulation data in order to write your technology library.

We will write the technology file in the file format created by Synopsis. The reason for this is that the synopsis technology file format is much more readable than the build gates technology file format. After writing the synopsis style technology file we will translate it to the style build gates expects by using the translation tool `syn2tlf`.

1.1 Sample Technology Library

A sample synopsis technology library is included below. Do not let the size of the file scare you. All the information is straight forward and easy to understand.

```
library ( StandardCellLibrary ) {

  /*=== Library Level Attributes ===*/

  /* technology family */
  technology( cmos );

  /* delay model */
  delay_model : generic_cmos;

  /* documentation details */
  date : "24 June 2004";
  comment : "Standard Cell Library";
  revision : 00;

  /* library units */
  voltage_unit : "1V";
  current_unit : "1mA";
  pulling_resistance_unit : "1kohm";
  time_unit : "1ns";
  /* represetn capacitance in terms of the standard load unit of an inverter */
```

```
capacitive_load_unit( 0.0288, pf );

/* nominal operating condition */
nom_process : 1.0;
nom_temperature : 25.0;
nom_voltage : 5.0;

/* Delay threshold points */
output_threshold_pct_fall : 30 ;
input_threshold_pct_fall : 70 ;
output_threshold_pct_rise : 70 ;
input_threshold_pct_rise : 30 ;

/* Slew threshold points */
slew_derate_from_library : 1.0;
slew_lower_threshold_pct_fall : 10 ;
slew_upper_threshold_pct_fall : 90 ;
slew_lower_threshold_pct_rise : 10 ;
slew_upper_threshold_pct_rise : 90 ;

/* Cell swapping criteria */
in_place_swap_mode : match_footprint;

/*=== Default Attributes ===*/

/* Fanout */
default_fanout_load : 1.0 ;
default_max_fanout : 20.0 ;

/* Pin Capacitance */
default_inout_pin_cap : 1.0 ;
default_input_pin_cap : 1.0 ;
default_output_pin_cap : 0.0 ;

/* Fall and rise resistance (transition delay per unit load) */
default_inout_pin_fall_res : 0.0 ;
default_output_pin_fall_res : 0.0 ;
default_inout_pin_rise_res : 0.0 ;
default_output_pin_rise_res : 0.0 ;

/* Intrinsic fall and rise (propogation delay) */
default_intrinsic_fall : 1.0 ;
default_intrinsic_rise : 1.0 ;

/* Slope sensitivity factor (to account for input transition) */
default_slope_rise : 0.0 ;
default_slope_fall : 0.0 ;
```

```
/*=== Scaling Factors for delay calculation ===*/

/* process scaling factors */
k_process_pin_cap : 0.0 ;
k_process_intrinsic_fall : 0.0 ;
k_process_intrinsic_rise : 0.0 ;
k_process_hold_fall : 0.0 ;
k_process_hold_rise : 0.0 ;
k_process_setup_fall : 0.0 ;
k_process_setup_rise : 0.0 ;

/* temperature scaling factors */
k_temp_pin_cap : 0.0 ;
k_temp_intrinsic_fall : 0.0 ;
k_temp_intrinsic_rise : 0.0 ;
k_temp_hold_fall : 0.0 ;
k_temp_hold_rise : 0.0 ;
k_temp_setup_fall : 0.0 ;
k_temp_setup_rise : 0.0 ;

/* voltage scaling factors */
k_volt_pin_cap : 0.0 ;
k_volt_intrinsic_fall : 0.0 ;
k_volt_intrinsic_rise : 0.0 ;
k_volt_hold_fall : 0.0 ;
k_volt_hold_rise : 0.0 ;
k_volt_setup_fall : 0.0 ;
k_volt_setup_rise : 0.0 ;

/*=== Cell Definitions ===*/

/* inverter */
cell( INV ) {

  /* cell level simple attributes */
  area : 86.4;
  cell_footprint : "inv";

  /* pin groups in the cell */
  pin( A ) {
    /* pin level simple attributes */
    direction : input;
    capacitance : 0.25;
  } /* pin( A ) */

  pin( Out ) {
    /* pin level simple attributes */
    direction : output;
```

```
      function : "!A";

      /* timing group within the pin level */
      timing() {
        /* timing level simple attributes */
        related_pin : "A";
        intrinsic_rise : 0.135;
        intrinsic_fall : 0.142;
        rise_resistance : 0.421;
        fall_resistance : 0.428;
      } /* timing() */

    } /* pin( Out ) */

  } /* cell( INV ) */

} /* library ( UT_LP_AMI06 ) */
```

Note: Emacs nicely highlights the syntax in this file if you turn on c-mode. Another note: this sample technology file wasn't made for the geometries we are using in class. While the format and generic parameters in the file will not change, the specific capacitance and timing information will.

The final section of this document explains every section of the technology library file. Because a few of the sections are harder to generate data for, we will focus on them more exhaustively. They are the sections you need to customize: the capacitive load unit, the delay threshold points and the cell function.

1.2 Capacitive Unit Load

The capacitive load unit lets us define all the capacitances in the circuit in terms of the standard load which is defined as four inverters. Every cell defines its pin capacitance in terms of this load. If you look at the inverter section of the technology library you will see that the input capacitance on pin A is 0.25, one quarter of the standard four inverter load.

1.3 Rise and Fall Timing Measurements

The delay threshold point defines the rise and fall percentages for propagation timing measurements and the slew threshold point defines our rise and fall percentages for resistance timing measurements. Propagation delay is usually measured as the time between a 50% input change and 50% output change, but in order to avoid negative delay caused by slow changing outputs we will measure the time between a 30% input change and 70% output change. Propagation delay is measured when not driving a load. on the other hand, resistance timing is measured while driving a load. For this measurement, calculate the time between a 10% input change and a 90% output change. The synthesis tool uses these times along with the capacitance of the standard load to calculate the rise and fall resistances.

1.4 Cell Function

The cell function contains the logical operation the cell performs. For logic gates such as nand, nor and inv, this is fairly easy. Defining cell functions for flip-flops and other sequential elements is a

little trickier. Sequential elements and tristate devices are explained very well at the UofU VLSI course web page. Go to http://www.cs.utah.edu/classes/cs5710/tutorials/appendixA.htm to read more about these types of cells.

2 Capacitive Unit Load

Let's calculate the capacitive load unit for our standard cell library. The input pins of a standard cell connect to the gate of one or more nmos and/or pmos transistors. An oxide layer insulates the gate poly layer from the substrate creating a capacitor. Mosis reports process parameter results from fabrication processes and from these reports we know the value of gate oxide capacitance per unit area for the AMI_ABN process is

$$C_{gox} = 1.1ff/u^2.$$

The total capacitance at an input pin is the sum of the areas $(W * L)$ of the transistor gates multiplied by C_{gox}.

The inverter input pin, A, is connected to both nmos and pmos transistors with dimensions $W/L = 3.6/1.2$ and $W/L = 10.8/1.2$. The total capacitance equals

$$\begin{aligned} C_{pinA} &= [(W_1 * L_1) + (W_2 + L_2)]C_{gox} \\ &= [(3.6 * 1.2) + (10.8 * 1.2)] * 1.1 * 10^{-15} \\ &= 0.019 * 10^{-12}. \end{aligned}$$

Calculating the standard capacitive unit load simply involves calculating the input capacitance for an input pin connected to four inverters.

$$\begin{aligned} C_{standardload} &= [(W_1 * L_1) + (W_2 + L_2)]C_{gox} \\ &= [(14.4 * 1.2) + (43.2 * 1.2)] * 1.1 * 10^{-15} \\ &= 0.076 * 10^{-12} \end{aligned}$$

The normalized capacitance of a single inverter is therefore

$$C_{pinA(normalized)} = 0.019pf/0.076pf = 0.25.$$

Remember, when calculating pin capacitances, first determine how many transistors the pin connects to, second, add up the areas of the transistors, third, multiply by C_{gox} and fourth, divide by the capacitive unit load.

Normally output pins do not connect to any transistor gates, so they do not have any capacitance. If for some reason the output pin does connect to the gate of one or more transistors you need to follow the same steps in determining its capacitance. When does an output pin connect to a transistor gate? (Hint: think about flip-flops)

3 Rise and Fall Timing Measurements

Rise and fall timing measurements fit into two categories, propagation times and slew rates. The technology library uses intrinsic rise and fall to record propagation times and it uses rise and fall resistance to record slew rates. This section describes the post layout simulation environment used to measure intrinsic_rise, intrinsic_fall, rise_resistance and fall_resistance.

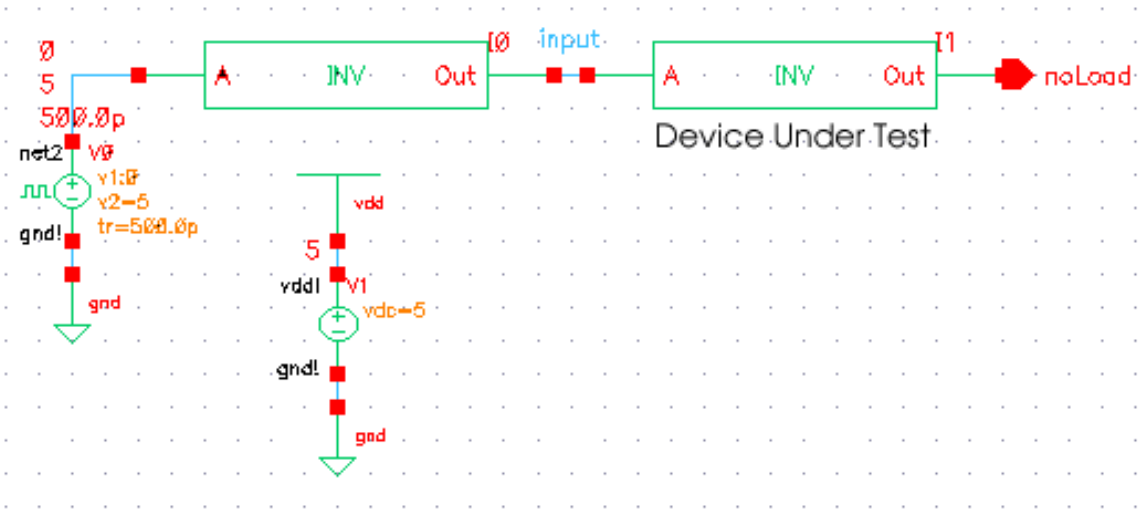

Figure 1: Propagation Rise and Fall Test Setup

3.1 Test Library

The first thing you need to do is create a new library called `testLib`. It will contain all the schematics you use for testing your standard cells. Link the library to the AMI12 technology library. Create two cells, one called `inv_test` for your inverter timing measurements and one called `inv4x` in which you need to create an inverter sized four times larger than normal. Make the widths $3.6 * 4 = 14.4$ and $10.8 * 4 = 43.2$. Also, remember to change the model types of the transistors to ami12N and ami12P.

3.2 Propagation Rise and Fall

When characterizing a cell's propagation delay, use the setup in Figure 1. Choose `vpulse` and `vdc` from NCSU_Analog_parts. Set the source parameters to the same values you used in the post layout simulation tutorial.

Notice that there are two inverters in the test setup. They are both instances of your inverter. The first one is used to simulate a real driving device and should use the schematic cell view. The second one is the device being tested and should use the analog_extracted view. Once again, follow the same steps you did in the post layout simulation tutorial to create a config cell and use the analog_extracted cell for one of the inverters.

After running your simulation, use the resulting waveforms to calculate the 30-70 rise and fall times for the input pin A. Write these down, you will need them when you write the technology library file. If the device under test has more than one input, i.e. a nand gate, then calculate the propagation rise and fall times separately for each input. For instance, to test B, hold A high and toggle B. Likewise, to test A hold B high and toggle A. Refer to the sample technology library file, notice how pin Out has a timing section related to pin A? Each input pin has its own timing section in the output pin section of the cell definition. If this were a nand gate, the output pin section would have two timing sections, one related to pin A and one to pin B.

Operator	Description
’	invert previous expression
!	invert following expression
^	logical XOR
*	logical AND
&	logical AND
space	logical AND
+	logical OR
|	logical OR
1	signal tied to logic 1
0	signal tied to logic 0

Table 1: Cell Function Operators

3.3 Rise and Fall Resistance

Measure the rise and fall resistances by removing the noLoad output from the device under test (DUT), wiring the `inv4x` to the output of the DUT and creating a new output pin on the output of the DUT called Load. Name the wire in between the DUT and the `inv4x` output. Run the simulation again and this time measure the 10-90 rise and fall times of the DUT. Record these for use in the technology library file. Similarly to the propagation time measurements, each input pin is timed separately for the resistance measurements.

4 Cell Function

The function attribute in the cell definition section of the technology library file defines the functionality of the cell. The synthesis tool, build gates, uses the function to correctly construct the operation you code in your verilog. Write the functions for your logical gates using the operators in Table 1.

A couple examples, the inverter has function : “!A”, a nand gate would have function : “!(A&B)”.

Describing functionality of sequential cells is more complicated and is described in detail at http://www.cs.utah.edu/classes/cs5710/tutorials/appendixA.htm.

5 Examining a Technology Library File

5.1 Create the File

You should put your technology file in your standard cell library folder. Navigate to the folder named stdCellLib and use your favorite text editor to create a file called `stdCellLib.lib`. By now you have an example of a technology library file and a good idea of what goes in one. We are going to write the library portion for your file as well as the inverter section. As a side note, you only need one file for your entire library. As you create and add more cells to the library, simply add a new cell definition section for each new cell.

5.2 Write the File

What follows is a pretty exhaustive description of the file.

Case Sensitivity and Comments - The library file is case sensitive so if you get some funny errors down the line this may be the cause. The commenting style is similar to c or verilog. A slash asterisk starts a comment and an asterisk slash ends one. Note: writing your file in emacs with the c-mode on is a great idea.

```
/* this is a comment in the library file */
```

Library Name - It will be appropriate to name the library stdCellLib. Go ahead and enter this into your cell as

```
library( stdCellLib ) {
        :
        :
}.
```

As you progress through the following steps, fill in appropriate details to the above framework in order to form a complete technology library file. Note that the details provided create a general library file, the file has many more commands that could be included to add more functionality. If you are interested, you can find more out about those by searching out a synopsis technology library file reference manual.

Library level simple and complex attributes -

- Technology Family - The technology we are following is CMOS and this is specified as follows

  ```
  technology(cmos);
  ```

  If specified, this attribute assignment should be the first one in the library source file.

- Delay Model - We need to specify a delay model for calculation of cell timing details. The available options are: generic_cmos, table_lookup, piecewise_cmos, cmos2, and dcm (delay calculation mode). We select generic_cmos. This is the next attribute specified after technology. The table_lookup method provides more accurate timing information, but if you want to use it you are on your own.

  ```
  delay_model : generic_cmos;
  ```

- Documentation Details - These attributes are for library documentation purposes.

  ```
  date : "date of library creation in any format";
  comment : "any single line comment about the library";
  revision : revision number in any format;
  ```

- Library Units - Sets the units for current, voltage, capacitance, time, etc used in the library. Capacitance units can be specified in farads or in terms of the standard 4x inverter load. We will use the latter method and so will specify capacitances in terms of this standard load.

  ```
  voltage_unit : "1V";
  current_unit : "1mA";
  pulling_resistance_unit : "1kohm";
  time_unit : "1ns";
  capacitive_load_unit(0.076,pf);
  ```

Remember to select the units so that they are consistent with each other. Note that in the library there is no direct way to specify the resistance unit. When we calculated rise_resistance and fall_resistance we specified them in nanoseconds. Well $R = t/C$ and ns/pf is in the $kOhm$ range. Using a similar approach make sure the other units are consistent with each other.

- Nominal Operating Condition - The nominal temperature, process and voltage values under which the library is characterized are specified here.

```
nom_process : 1.0;
nom_temperature : 25.0;
nom_voltage : 5.0;
```

Temperature is always in degrees celsius.

- Delay Threshold Points - Earlier, when we characterized the inverter, we decided that a 30% input to 70% output change represents a better propagation delay than a 50% to 50% change. These threshold points are specified as follows.

```
output_threshold_pct_fall : 30;
input_threshold_pct_fall : 70;
output_threshold_pct_rise : 70;
input_threshold_pct_rise : 30;
```

Realize that 0% always corresponds to logic '0' and 1% always corresponds to logic '1'. If not specified the default value is 50%.

- Slew Threshold Points - Similar to delay threshold points, slew threshold points set the points used for rise and fall time measurements.

```
slew_lower_threshold_pct_fall : 10;
slew_upper_threshold_pct_fall : 90;
slew_lower_threshold_pct_rise : 10;
slew_upper_threshold_pct_rise : 90;
```

- Cell Swapping Criteria - Sometimes it is useful to design a library with multiple realizations of the same cell. These cells have different names, but their input pins, output pins and logical functions are the same. For instance, one inverter is designed with minimum transistor sizes and another is designed to drive a high load. The `cell_footprint` attribute tells the tool the cells are functionally equivalent. It then chooses the cell based on the functionality that best realizes the circuit. Activate this by specifying the following attribute.

```
in_place_swap_mode : match_footprint;
```

The other option is `no_swapping`.

Default Attributes - These are the default values for the entire cell library. They are in effect unless specifically overridden in a cell description.

- Fanout - These specify the load on the input pins of a cell and the maximum fanout of an output pin in terms of a standard 4x inverter load.

```
default_fanout_load :  1.0;
default_max_fanout  : 20.0;
```

- Pin Capacitance - Specifies the default capacitance values of every pin.

```
default_inout_pin_cap  : 1.0;
default_input_pin_cap  : 1.0;
default_output_pin_cap : 0.0;
```

- Rise and Fall Resistances - These specify the default resistance values pullup and pull-down transistor stacks. Calculate them in terms of transition delay (rise or fall time) per unit load. The cell attributes `rise_resistance` and `fall_resistance` override these defaults.

```
default_inout_pin_fall_res  : 0.0;
default_output_pin_fall_res : 0.0;
default_inout_pin_rise_res  : 0.0;
default_output_pin_rise_res : 0.0;
```

- Intrinsic Rise and Fall - Default low-to-high and high-to-low propagation delays while driving no load. The cell attributes `intrinsic_rise` and `intrinsic_fall` override these default values.

```
default_intrinsic_rise : 1.0;
default_intrinsic_fall : 1.0;
```

- Slope Sensitivity Factor - Accounts for the delay caused by the ramp nature of input signals.

```
default_slope_rise : 0.0;
default_slope_fall : 0.0;
```

Scaling Factors for Delay Calculations - The timing library file specifies delay factors with respect to nominal operating conditions. The actual operating conditions require different delay factors and calculate them using scaling factors. Our library ignores delay variations due to changes in operating conditions and therefore sets the scaling factors to zero.

```
/* process scaling factors */
k_process_pin_cap : 0.0 ;
k_process_intrinsic_fall : 0.0 ;
k_process_intrinsic_rise : 0.0 ;
k_process_hold_fall : 0.0 ;
k_process_hold_rise : 0.0 ;
k_process_setup_fall : 0.0 ;
k_process_setup_rise : 0.0 ;

/* temperature scaling factors */
k_temp_pin_cap : 0.0 ;
k_temp_intrinsic_fall : 0.0 ;
k_temp_intrinsic_rise : 0.0 ;
k_temp_hold_fall : 0.0 ;
k_temp_hold_rise : 0.0 ;
k_temp_setup_fall : 0.0 ;
k_temp_setup_rise : 0.0 ;
```

```
/* voltage scaling factors */
k_volt_pin_cap : 0.0 ;
k_volt_intrinsic_fall : 0.0 ;
k_volt_intrinsic_rise : 0.0 ;
k_volt_hold_fall : 0.0 ;
k_volt_hold_rise : 0.0 ;
k_volt_setup_fall : 0.0 ;
k_volt_setup_rise : 0.0 ;
```

Cell Definitions - This section compromises the main section of the technology library file. It describes individual cells in the library and each cell has its own section. An inverter cell section uses the following framework.

```
/* inverter cell */
cell(INV) {
  :
  :
}
```

Each cell describes cell level simple attributes and pin groups with specific timing information.

- Cell Level Simple Attributes - The `area` attribute specifies cell area in no specific rigid units. For consistency, specify the area as the length and width of the non-overlapping section of a standard cell. Measure from the origin to the center of the `metal1-substrate` via directly above it for cell height (54 micros for a standard cell) and measure width as the distance between the origin and the center of the rightmost via (10.2 microns).

  ```
  area : 550.8;
  ```

  As discussed earlier in Cell Swapping Criteria, cells with identical pin names and functionality but different characteristics are identified by the same value for the `cell_footprint` attribute. Specify the inverter family by setting `cell_footprint` to `inverter`.

  ```
  cell_footprint : "inverter";
  ```

- Pins and Pin Level Simple Attributes - The inverter has two pins, A and Y. Pin A sees the gates of an nmos transistor and a pmos transistor, both of which contribute to its gate capacitance. We calculated this earlier. Here the capacitance is recorded in terms of the standard 4x inverter load, which makes a single inverter input's capacitance 0.25. The output pin does not internally connect to any transistor gates, therefore it does not record any pin capacitance. If it did internally connect to a transistor gate then specify its capacitance similarly to the input pin's capacitance.

  ```
  pin(A) {
    /* pin level simple attributes */
    direction : input;
    capacitance : 0.25;
  }

  pin(Y) {
  ```

```
  /* pin level simple attributes */
  direction : output;
    :
    :
}
```

- Cell Function - The `function` attribute defines the logical cell function. We described it in detail earlier. Each output pin needs a function section, add the following under the `direction` attribute.

  ```
  function : "!A";
  ```

- Cell Timing Information - Timing level simple attributes describe timing information of the output pins in relation to the input pins. Every output pin has a timing section for every input pin. For the inverter, pin Y has a timing section for pin A. For a nand gate, the output has a timing section for each input. We already calculated the timing information for the inverter, record it under the `function` attribute.

  ```
  /* timing group for pin A */
  timing() {
    /* timing level simple attributes */
    related_pin : "A";
    intrinsic_rise : value;
    intrinsic_fall : value;
    rise_resistance : value;
    fall_resistance : value;
  }
  ```

Adding Additional Cells - As you characterize more cells for the technology library file, add them under the inverter. Remember, every cell type has its own cell definition and you do not need to rewrite the library header information for each cell.

5.3 Compile the File

Save your file as `stdCellLib.lib` . Now convert the file to the format the synthesis tool, build gates, expects. Do this by running the `syn2tlf` tool with `stdCellLib.lib` as the argument. At a console prompt type:

```
> syn2tlf stdCellLib.lib
```

You should now have a stdCellLib.tlf file in your directory. Congratulations, you are almost through the tutorial!

6 Creating a Verilog Interface Description

In addition to the technology library file, the tools expect a verilog interface file containing a separate module for each cell in the library describing the input and output properties of every pin in the cell. The general file format follows:

```
module cell1_name (out1, out2, ... , in1, in2, ...);
  output out1, out2, ...;
```

```
  input  in1, in2, ...;
endmodule

module cell2_name (out1, out2, ... , in1, in2, ...);
  output out1, out2, ...;
  input  in1, in2, ...;
endmodule

   :
   :

module celln_name (out1, out2, ... , in1, in2, ...);
  output out1, out2, ...;
  input  in1, in2, ...;
endmodule
```

A verilog interface file for the `stdCellLib` with only one inverter cell looks like this:

```
module INV ( Y, A );
  output Y;
  input  A;
endmodule
```

Make a file in your `stdCellLib` directory called `stdCellLib.v`. Add the inverter cell to it. As you add additional cells to your library, make sure to include a module for each in the verilog interface description.

7 Conclusion

Creating a standard cell, characterizing it, recording all the data in the technology library file and writing the verilog interface is a long process. Refer to this file often as you create more cells for your library. Remember that cell library creating simplifies your life later by allowing you to write your circuits in an HDL and synthesize them without resorting to k-maps, state diagrams, schematic capture and lots of headaches when you need to redesign.

As always, if you have any trouble or find any errors in this tutorial let the TAs and the professor know.