Introduction to EPIC tools

Nanosim is a transistor-level power simulator and analyzer for CMOS and BiCMOS circuit designs. As more components are integrated into a tiny silicon chip, power consumption becomes a major concern. Nanosim is used to find excessive consumption of power through current analysis. In CMOS Nanosim calculates the switching current, caused by switching and node capacitance, the leakage current, caused by floating nodes, and short circuit current, caused by rise/fall times and dc contentions.
Pathmill is an analysis tool you can use to find critical paths and verify timing in semiconductor designs. It employs a static and mixed-level timing analysis to process transistor, gates, and timing models. It calculates timing delays, performs path searches, and checks timing requirements simultaneously.
Timemill catches design errors and timing problems during the design phase. This gives you the flexibility of more iterations to optimize your design. The switch and transistor-level models closely simulate the physical behavior of MOS devices.
NOTE: Filename extensions do matter. ".spi" != ".cir"

Environment Setup for EPIC tools

To run epic tools, you need to add following line in your *.cshrc* file or just type it at Unix shell prompt.

source /mentor/csu_setup/epic_setup.csu

Procedure To Run Nanosim

To run nanosim, you need to have *Spice Netlist file* and to create *nanosim configuration* file and *vector* file. The steps to run Nanosim are:

1. Make an *epic* directory under your layout design directory, copy the Eldo netlist file *design.cir* to the epic directory, and go to *epic* directory. Make a copy of your netlist file as *design.spi* then edit this file and remove all lines except the FET definitions (line staring with M_$) and the .end line.
2. Create *design.vec* file
3. Create *cfg_pw* file
4. Run Nanosim by using the following command\

   nanosim -n design.spi -nvec design.vec -c cfg_pw -o nanosim -p /class/EE571/epic_tech/tech*

5. After running nanosim, you can check your design or run-time errors in *nanosim.log* or *nanosim.err*. The *nanosim.out* file contains the results of power analysis of your design.

Procedure To Run Timemill

To run timemill you also need to have *Spice Netlist file* and to create *timemill configuration* file, *vector* file, and *command* file. The *vector* and *command* file is the same as the ones for the powermill. The steps to run Timemill are:

1. Make an *epic* directory under your layout design directory, copy the Eldo netlist file *design.cir* to the epic directory, and go to *epic* directory. Make a copy of your netlist file as *design.spi* then edit this file and remove all lines except the FET definitions (line staring with M_$) and the .end line.
2. Create *design.cmd* file
3. Create *design.vec* file
4. Create *cfg_tm* file
5. Run timemill by using the following command\

   timemill -n design.spi design.cmd -c cfg_tm -o timemill -p /class/EE571/epic_tech/tech*

6. After running timemill, you can check your design or run-time errors in *timemill.log* or *timemill.err*. The *timemill.out* file contains the timing analysis results of your design.

Procedure To Run Pathmill

The required files for running pathmill are *Spice netlist* and *pathmill configuration files*. The steps to run Pathmill are:

1. Make an *epic* directory under your layout design directory, copy the Eldo netlist file *design.cir* to the epic directory, and go to *epic* directory. Make a copy of your netlist file as *design.spi* then edit this file and remove all lines except the FET definitions (line staring with M_$) and the .end line.
2. Create *cfg_pm* file
3. Run pathmill by using the following command\

   pathmill -n design.spi -c cfg_pm -o pathmill -p /class/EE571/epic_tech/tech*

4. After running pathmill, you can check your design or run-time errors by looking *pathmill.log* or *pathmill.err*. The *pathmill.out* file contains the pathmill analysis results of your design.

Example files to be created

This section gives you some examples that you need to create to run powermill, timemill, and pathmill.

Command file : .cmd

The first example of command file can be used if your design has clock signal as an input.

```
 (is=clk) (en=CKa)  (ot=CK) (ov=0) (sv=0, 40, 80, 1,1);
 (is=vec) (en=dff1.vec)  (ot=Din);
```

The 1st line is a epic clock stimulus(**clk**) command. This command defines a clock with the name **CK** and clock period is 80ns.
The syntax of **clk** command is:\

```
 (is=clk) (en=element_name} (ot= signal_list}) [(ov=initial_value)]
       (sv=start_time, toggle_time, period, rise_time, fall_time);
```

The 2nd line is a vector stimulus(**vec**) command. This command provides a method for reading in a file containing input stimuli and output comparison. The syntax of **vec** command is: \

```
 (is=vec) (en=vector_file) (ot=signal_list)
```

The *vector_file* specifies the file to be read. The *signal_list* specifies the node to be forced or compared.
Next example is a command file that does not contain any clock signals.

```
  (is=vec) (en=ring.vec) (ot=in1,in2,in3,in4,in5,in6);
```

Vector file : .vec

This section shows two different vector file examples. The first is a vector file for a **D flip-flop** and the latter is a vector file for a ring oscillator.

```
                    ; Example vector file (D flip-flop)

                    type vec
                    signal D
radix        1
;
io           i
;            D
;            i
;   time(ns)  n
;
          0  1
         80  0
```

```
            110  1
            140  0
            150  1
            160  0
            170  1
            200  0
      ; end of file

 ; Example vector file for ring oscillator

           type vec

           signal A B C D E F
radix        11 11 11
io           ii ii ii
;
;            ii ii ii
;            nn nn nn
;            12 34 56
;
          0 10 10 11
         80 00 10 11
        110 10 10 10
        140 00 10 00
        150 10 10 11
        160 00 10 10
        170 10 10 01
        220 10 10 10
        270 00 10 11
        310 10 10 01
        370 00 10 10
        400 10 10 11
        450 00 10 11
        490 10 10 01
        560 00 10 11
        600 10 10 11
```

The **type** specifies that it is a vector file format. The **signal** specifies the input signals (order is important). The **radix** specifies the number of bits associated with each input signal. For the **io**, **i** indicates that the corresponding signal is an input and the stimulus is used to stimulate the circuit, **o** indicates output to be compared.

The following is the content that you would need to use as vector file for Static and PTL.

```
type vec
signal A[15-0] B[15-0] Cn[0]
radix 4444 4444 1
io iiii iiii i
;
```

```
; ii ii ii
; nn nn nn
; 12 34 56
;
0 0000 FFFF 0
80 FFFF FFFF 0
110 FFFF 0001 0
140 2222 1111 1
150 5555 5555 0
```

The following is the content that you would need to use as vector file for Dynamic.

```
type vec
signal A[15-0] B[15-0] Cn[0] CLK
radix 4444 4444 1 1
io iiii iiii i i
;
; ii ii ii i
; nn nn nn n
; 12 34 56 7
;
0 0000 FFFF 0 0
40 0000 FFFF 0 1
80 FFFF FFFF 0 0
110 FFFF 0001 0 0
120 FFFF 0001 0 1
140 2222 1111 1 1
150 5555 5555 0 1
160 5555 5555 0 0
```

Configuration files

In the following configuration commands the * mark is used to specify all nodes.

- Nanosim : cfg_pw
- Timemill : cfg_tm
- Pathmill : cfg_pm

Nanosim : cfg_pw

```
;example cfg_pw file
print_node_i *
print_node_v *
report_node_powr GND VDD
set_print_iwindow start=0n end=500n
```

The **print_node_i *** and **print_node_v *** commands request the node instantaneous current and voltage information for all nodes to be printed to the **nanosim.out** file. The

report_node_powr GND VDD command requests instantaneous, average, and rms current to be printed to **nanosim.log** and **nanosim.out** files. The **set_print_iwindow** command sets the time period during which to calculate the RMS and average currents.

Timemill : cfg_tm

```
;example cfg_tm file
print_node_logic *
print_node_v *
```

The first command prints all nodes in digital format; the second command prints all nodes as voltage waveforms. These results are stored in **timemill.out** file.

Nanosim : cfg_pm

```
;example cfg_pm file
source_node Din CLK
sink_node Q QB
```

The **source_node** and **sink_node** indicate input and output pin_names respectively.

Result Analysis - nanosim, pathmill

1. **Nanosim :** Look at the **nanosim.log** file. This file contains power analysis result of your design. You can find Average current and RMS current from VDD to GND. The power can be calculated by $P = VDD \times Current$.
2. **Pathmill :** Look at the **pathmill.out** file. This file contains pathmill analysis result of your design. You can find maximum delay and maximum delay path.
3. **Timemill :** Look at the **timemill.out**. This file contains timemill analysis result of your design. Follow the steps in the following section.

Timemill Output Display and Analysis

Unfortunately we don't have SimWave installed. You will have to analyze the timemill output by hand.
THIS INFO IS FOR ARCHIVAL PURPOSES:
The **timemill.out** can be displayed by using interactive waveform display tool **SimWave**. Output file displaying procedure is as follows:

1. Type **wd** in c-shell prompt
2. Select **File →Database →Load**
3. Click **epic** button and enter **timemill.out** in the **Filename** field.
4. Click **OK**
5. Select **Edit →Add Signals**. Then you can see the signal names in the **Signals** box.

6. Select signals which you want to display and click **Display Signals**. Then you will have **SimWave Browser**. **Display Signals** button is located at right side of the **Close Windows** button.

About this document ...

The command line arguments were:
latex2html epic.tex.
The translation was initiated by Geun Rae Cho on Mon Dec 4 13:55:18 MST 2000
This document was revised by Chinmay Gupte on Fri Apr 23 14:25:10 MST 2004

Latest version [done by Charles Thangaraj @ Tuesday, January 11, 2005 1:24:29 PM]