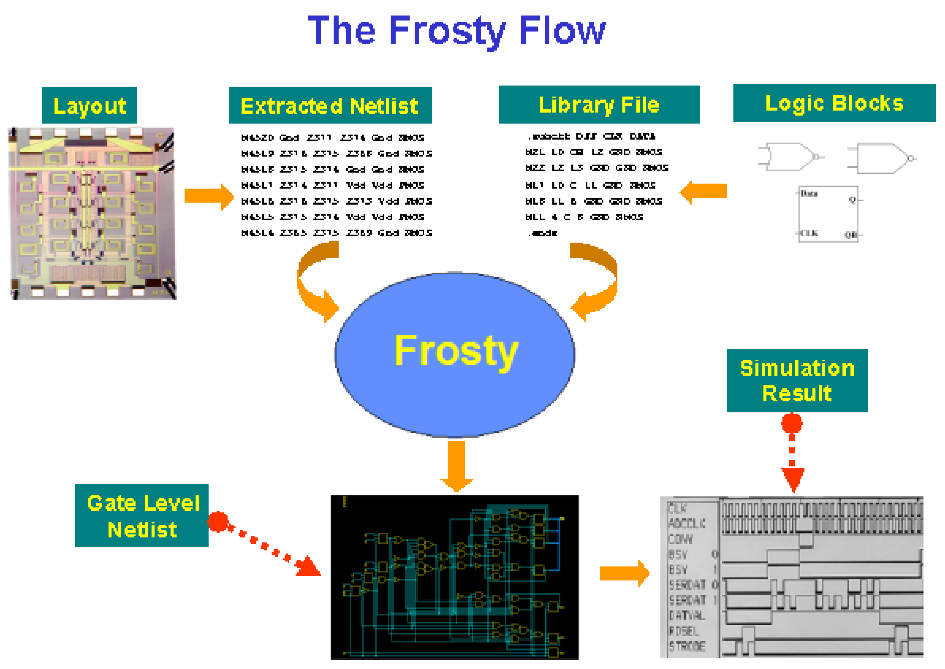
The FROSTY User Guide

Lei Yang
Department of Electrical Engineering
University of Washington
Seattle, WA 98105
May 22, 2003

1. Description

Circuit recognition and extraction is a very important task in VLSI CAD field. This kind of program is widely used in many commercial CAD tools for post-layout simulation, layout function verification, formal verification and design for test.

FROSTY is an automatic CMOS circuit recognition and extraction tool. It reads in two files — file1 and file2. File1 is the description of an ***object circuit***, FROSTY automatically recognize all the standard CMOS gates, such as INV, NAND2, AOI12… in the file1. In file2, user can define some higher level digital blocks, such as DFF, Latch, adder, etc. FROSTY automatically checks whether there are instances of the digital blocks in file1. Finally, FROSTY outputs a Verilog format RTL level netlist and a header file (It contains the functional definitions of all used standard CMOS gates), the two files can be simulated in any digital simulators.



FROSTY read in file format is standard SPICE format, which is compatible with industry standard. The algorithm used by FROSTY is a two-step algorithm. In the first step, structural recognition algorithm is used to recognize all the standard CMOS gates in ***object circuit***. Then in the second step, pattern matching algorithm is performed to extract all the user defined higher level blocks.

FROSTY is written in C++ and runs under the UNIX operating system. It compiles using g++ on Sun SparcStations, but may need some modifications to use other machines and compilers.

FROSTY is research software, please send bug report to yanglei@ee.washington.edu.

2. Usage

There are 2 input files and 2 output files for FROSTY, the file names are specified on the UNIX command line:

```
%> extractor  <flat netlist>  <library flie>  <Output netlist>  <header file>
```

1) Flat netlist *(input)* — the input flat netlist of the ***object circuit***, SPICE format.
2) Library file *(input)* — SPICE format, user can define digital blocks need to be recognized in this library file.
3) Output netlist *(output)* — Verilog format RTL netlist output from FROSTY.
4) Header file *(output)* — Define the Verilog format behavior model for the standard CMOS gates in the output netlist.

For example, In the working directory, there are 2 files — “PSM.ckt” and “PSM.lib”. “PSM.ckt” is the SPICE format description of PSM circuit, and “PSM.lib” defines some digital blocks contained in “PSM.ckt”. User can input the following command:

```
%> extractor  PSM.ckt   PSM.lib  PSM.out  header.v
```

After FROSTY finishes the running, two more output files can be found in the working directory — “PSM.out” and “header.v”. “PSM.out” is an RTL level description of PSM circuit, “header.v” defines Verilog format behavior model of all the standard CMOS gates.

3. Installation

The executable file of FROSTY, all the test circuits, one post-layout simulation example are compressed to a "frosty-demo.tar" file. To install FROSTY, just copy the file to your working directory, and run the following command:

%> tar xvf frosty-demo.tar

After running the above command, a new directory call "FROSTY-DEMO" will show up. This directory include the following:

1) Tutorial.pdf

Tutorial file of FROSTY.

2) extractor

This is the executable file of FROSTY, please refer to Part 3 to see the usage of FROSTY.

3) Boeing_Test_Circuit

There are 3 series of test circuits from Boeing in this directory:

--- CEGRP

A series of CEGRP circuits and CEGRP.lib file. In this directory, user can test FROSTY by input the command:

% ../../extractor CEGRP.ckt CEGRP.lib CEGRP.out header.v

or input

% ../../extractor CEGRP3.ckt CEGRP.lib CEGRP.out header.v

--- DFGRP

A series of DFGRP circuits and DFGRP.lib file. To test them, user can input:

% ../../extractor DFGRP4.ckt DFGRP.lib DFGRP.out header.v

--- PSM

A series of PSM circuits and PSM.lib file.

4) PSM-Post_layout_simulation

In this directory, an example - PSM is used to show FROSTY in the flow of post layout simulation. There are 3 directories:

--- VHDL_simulation

PSM VHDL source code and testbench (provided by Boeing). They can be simulated in VHDL simulator, for example, Active-VHDL.

--- Hspice_simulation

PSM Flat netlist extracted from layout (provided by Boeing). It can be simulated in Hspice to get the post-layout waveform.

--- Verilog_simulation

FROSTY extracts all the gates and blocks outputs a Verilog format block level netlist.

4. Input/Output file syntax

The 2 input files are SPICE format:

1) Input file 1

Input file 1 is a flat netlist description of the object circuit. Now FROSTY requires this netlist is a flat netlist. In the future, FROSTY will also support hierarchy netlist.

In this netlist, user need to use ".subckt" command to specify the input ports and output ports of the ***object circuit***. Otherwise, FROSTY can not give the correct port information in the output Verilog format file. Also, in this input netlist, user need to use ".model" file to specify the MOSFET is NMOS or PMOS.

An example is in the following.

```
.subckt sdfa4 a b c e f g Q QBAR
m1 VDD a CLK VDD PMOS L=2.900000e-06 W=6.000000e-07
m2 CLK a GND GND NMOS L=2.900000e-06 W=6.000000e-07
m3 VDD b D VDD PMOS L=2.900000e-06 W=6.000000e-07
m4 1 c GND GND NMOS L=2.900000e-06 W=6.000000e-07
m5 D b 1 GND NMOS L=2.900000e-06 W=6.000000e-07
m6 VDD c D VDD PMOS L=2.900000e-06 W=6.000000e-07
m7 VDD g SCANIN VDD PMOS L=2.900000e-06 W=6.000000e-07
m8 SCANIN g GND GND NMOS L=2.900000e-06 W=6.000000e-07
m9 VDD e 2 VDD PMOS L=2.900000e-06 W=6.000000e-07
m10 TEST f GND GND NMOS L=2.900000e-06 W=6.000000e-07
m11 TEST e GND GND NMOS L=2.900000e-06 W=6.000000e-07
m12 2 f TEST VDD PMOS L=2.900000e-06 W=6.000000e-07
m13 3 4 5 GND NMOS L=1.450000e-06 W=6.000000e-07
m14 6 7 8 VDD PMOS L=1.400000e-06 W=6.000000e-07
m15 8 9 VDD VDD PMOS L=2.200000e-06 W=6.000000e-07
m16 9 6 VDD VDD PMOS L=2.200000e-06 W=6.000000e-07
m17 6 4 10 VDD PMOS L=2.200000e-06 W=6.000000e-07
m18 3 4 11 VDD PMOS L=2.200000e-06 W=6.000000e-07
m19 11 10 VDD VDD PMOS L=2.200000e-06 W=6.000000e-07
m20 10 3 VDD VDD PMOS L=2.200000e-06 W=6.000000e-07
.model NMOS nmos1 level=1 k=0.2 Vth=0.4 lambda=0.002
.model PMOS pmos1 level=1 k=0.2 Vth=0.4 lambda=0.002
```

```
.ends sdfa4
```

2) Input file 2

Input file 2 defines all the digital blocks. The format is also SPICE format, however, keywords ".macromodel" and ".endm" are used to specify the beginning and end of a digital block.

The following is an example of an MUX.

```
.macromodel MUX3 IN0 IN1 IN2 S0 S1 Y
M0 NODEN5 CNODE GND GND NMOS W=2.900U L=0.600U
M1 Y S1BAR NODEN6 GND NMOS W=2.900U L=0.600U
M2 NODEN6 DNODE GND GND NMOS W=2.900U L=0.600U
M3 S1BAR S1 GND GND NMOS W=1.450U L=0.600U
M4 S0BAR S0 GND GND NMOS W=1.450U L=0.600U
M5 CNODE IN2 GND GND NMOS W=1.450U L=0.600U
M6 NODEN3 S0BAR GND GND NMOS W=2.900U L=0.600U
M7 DNODE IN0 NODEN3 GND NMOS W=2.900U L=0.600U
M8 DNODE S0 NODEN4 GND NMOS W=2.900U L=0.600U
M9 NODEN4 IN1 GND GND NMOS W=2.900U L=0.600U
M10 Y S1 NODEN5 GND NMOS W=2.900U L=0.600U
M11 NODEP5 CNODE VDD VDD PMOS W=3.050U L=0.600U
M12 Y DNODE NODEP6 VDD PMOS W=3.050U L=0.600U
M13 NODEP6 S1 VDD VDD PMOS W=3.050U L=0.600U
M14 S1BAR S1 VDD VDD PMOS W=2.200U L=0.600U
M15 S0BAR S0 VDD VDD PMOS W=2.200U L=0.600U
M16 CNODE IN2 VDD VDD PMOS W=2.200U L=0.600U
M17 NODEP3 S0 VDD VDD PMOS W=3.050U L=0.600U
M18 DNODE IN0 NODEP3 VDD PMOS W=3.050U L=0.600U
M19 DNODE S0BAR NODEP4 VDD PMOS W=3.050U L=0.600U
M20 NODEP4 IN1 VDD VDD PMOS W=3.050U L=0.600U
M21 Y S1BAR NODEP5 VDD PMOS W=3.050U L=0.600U
.model NMOS nmos1 level=1 k=0.2 Vth=0.4 lambda=0.002
.model PMOS pmos1 level=1 k=0.2 Vth=0.4 lambda=0.002
.endm MUX3
```

The 2 output files are Verilog format.

3) Output file 1

Output file 1 is a Verilog format RTL level netlist. The following is an example:

```
module sdfa4 (g, f, e, c, b, a, QBAR, Q);
input g, f, e, c, b, a;
output QBAR, Q;
 INV  U1 ( .a(a), .out(CLK) );
 NOR2  U2 ( .a(e), .b(f), .out(TEST) );
 NAND2  U3 ( .a(b), .b(c), .out(D) );
 INV  U4 ( .a(g), .out(SCANIN) );
 DFF  U5 ( .SCANIN(SCANIN), .CLK(CLK), .D(D), .TEST(TEST), .QBAR(QBAR), .Q(Q) );
Endmodule
```

4) Output file 2

Output file 2 defines the behavior models of all the standard CMOS gates. The following is an example of an NOR2:

```
module NAND2(a, b, out);
input a, b;
output out;
assign out = ~(a & b);
endmodule
```

Notice here FROSTY only defines all the behavior models of the standard CMOS gates. For simulation purpose, use should manually add the behavior description of the digital blocks that are defined in the library file. For example, if user defines an DFF in the library file, he should write the Verilog model of the DFF, shown in the following:

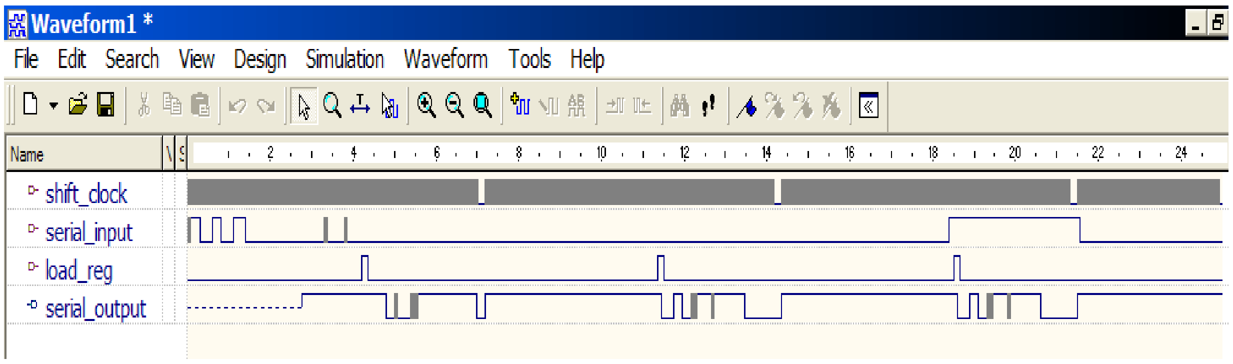
```
module DFFP(DATA, CLK, PRB, Q, QB);
input DATA, CLK, PRB;
output Q, QB;
reg Q, QB;
always @(posedge CLK or negedge PRB)
begin
  if (PRB == 1'b0)
    begin
     Q <= 1'b1;
     QB <= 1'b0;
    end
  else
   begin
    Q <= DATA;
    QB <= ~DATA;
   end
end
endmodule
```

3. An example to shown the flow of post layout simulation

In this part, an example, PSM, is illustrated to show how to use FROSTY in the flow of post layout simulation.

1) VHDL simulation

PSM is written in VHDL in Boeing. Using the PSM VHDL source code and testbench, we can simulate them in Active-VHDL and get the following waveform:

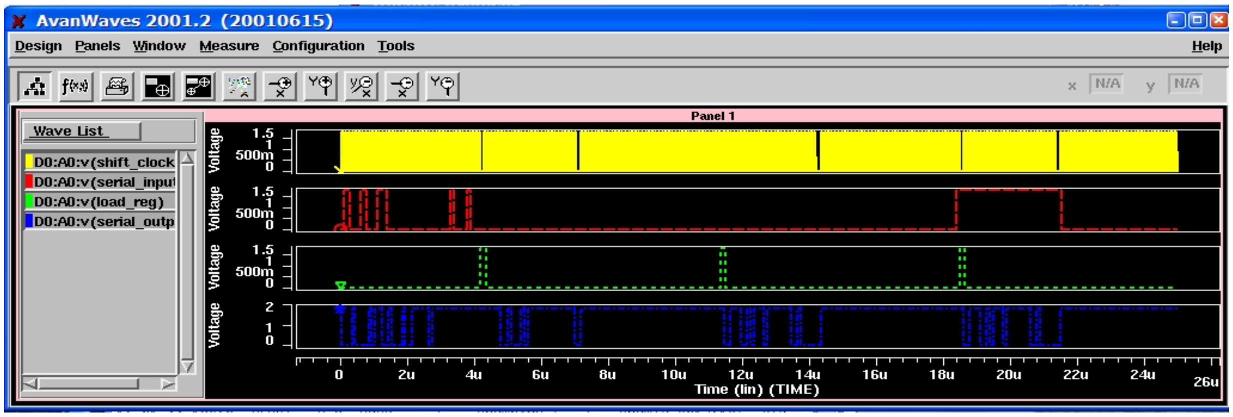


2) HSPICE simulation

Also Boeing provides the flat netlist of PSM extracted from PSM layout. To run the HSPICE simulation, just go to the directory "./FROSTY-DEMO/PSM-Post_layout_simulation/Hspice_simulation" and run the following command:

% hspice psm.sp

This Hspice simulation will take almost 2 hours in our Workstation (900M CPU, 16GB RAM). You can use AWAVES to see the waveform:



3) Verilog simulation

To run the Verilog simulation, we need to get the PSM circuit RTL level netlist from FROSTY, go to directory: "FROSTY-DEMO/Boeing_Test_Circuit/PSM " , run the following command:

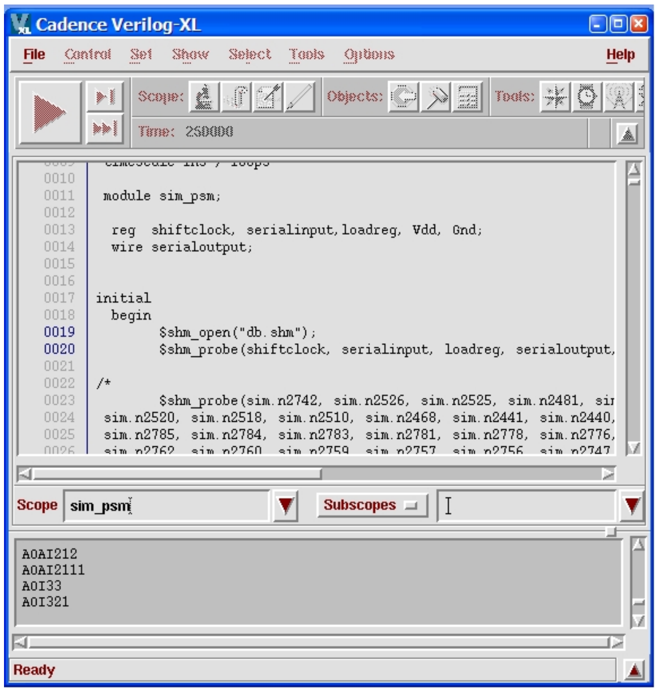
% ../../extractor PSM.ckt PSM.lib PSM.out header.v

After the running, you will see FROSTY output 2 files, one is RTL level netlist "PSM.out", the other one is gate model definition file "header.v", which define the Verilog model of the standard CMOS gates, such as INV, NOR2…..

Copy the above two files "PSM.out" "header.v" to the directory "FROSTY-DEMO/PSM-Post_layout_simulation/Verilog_simulation". In this directory, we have two other files "Subcircuit.model" "sim_psm.v". "Subcircuit.model" define the Verilog model of the higher level blocks in PSM design, such as DFF, latch, MUX and so on. "sim_psm.v" is the simulation file for PSM design. Run the following command:

% verilog sim_psm.v Subcircuit.model header.v PSM.out +gui &

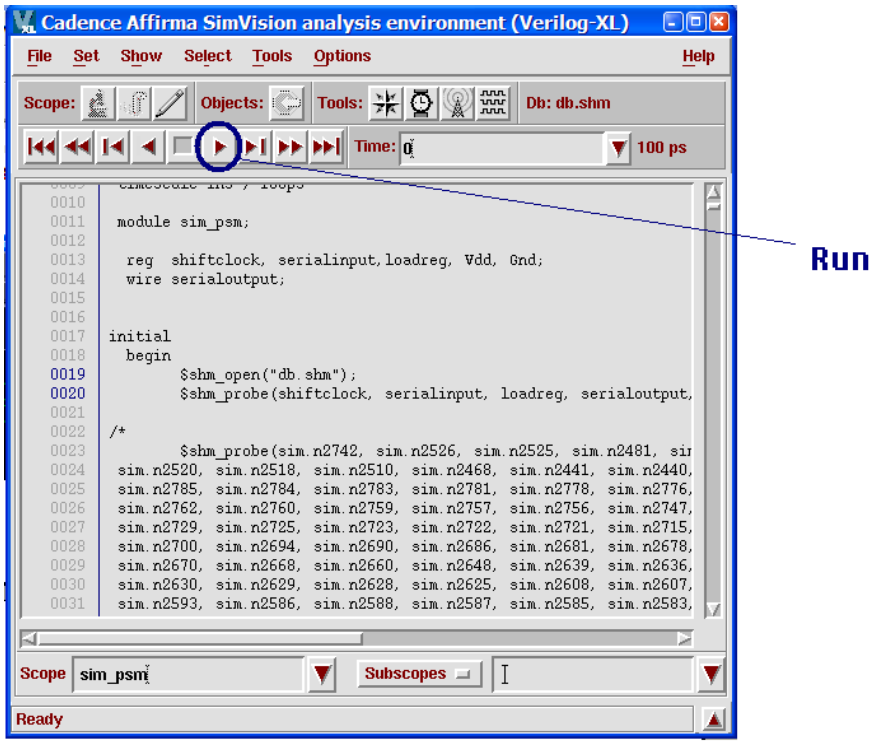
We can see the Verilog-XL GUI interface show up after running the above the command, shown in the following.



```
 module sim_psm;

  reg  shiftclock, serialinput,loadreg, Vdd, Gnd;
  wire serialoutput;


initial
  begin
        $shm_open("db.shm");
        $shm_probe(shiftclock, serialinput, loadreg, serialoutput,

/*
        $shm_probe(sim.n2742, sim.n2526, sim.n2525, sim.n2481, si
 sim.n2520, sim.n2518, sim.n2510, sim.n2468, sim.n2441, sim.n2440,
 sim.n2785, sim.n2784, sim.n2783, sim.n2781, sim.n2778, sim.n2776,
```

```
AOAI212
AOAI2111
AOI33
AOI321
```

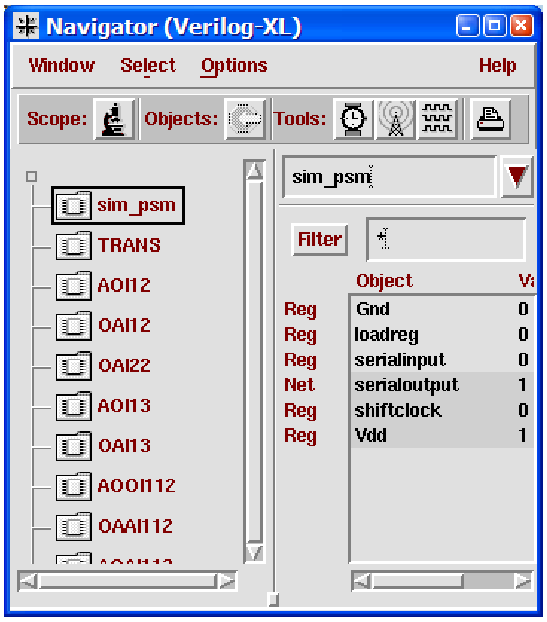
In the GUI interface, select "File-> Post Processing", a window will pop up, click "Yes", then we can enter into the "Post Processing" enviroment. Shown in the following:


Cadence Affirma SimVision analysis environment (Verilog-XL)
File Set Show Select Tools Options Help
Scope: Objects: Tools: Db: db.shm
Time: 0 100 ps
Run
Scope sim_psm Subscopes
Ready

```
 module sim_psm;

  reg  shiftclock, serialinput,loadreg, Vdd, Gnd;
  wire serialoutput;


initial
  begin
        $shm_open("db.shm");
        $shm_probe(shiftclock, serialinput, loadreg, serialoutput,

/*
        $shm_probe(sim.n2742, sim.n2526, sim.n2525, sim.n2481, si
 sim.n2520, sim.n2518, sim.n2510, sim.n2468, sim.n2441, sim.n2440,
 sim.n2785, sim.n2784, sim.n2783, sim.n2781, sim.n2778, sim.n2776,
 sim.n2762, sim.n2760, sim.n2759, sim.n2757, sim.n2756, sim.n2747,
 sim.n2729, sim.n2725, sim.n2723, sim.n2722, sim.n2721, sim.n2715,
 sim.n2700, sim.n2694, sim.n2690, sim.n2686, sim.n2681, sim.n2678,
 sim.n2670, sim.n2668, sim.n2660, sim.n2648, sim.n2639, sim.n2636,
 sim.n2630, sim.n2629, sim.n2628, sim.n2625, sim.n2608, sim.n2607,
 sim.n2593, sim.n2586, sim.n2588, sim.n2587, sim.n2585, sim.n2583,
```

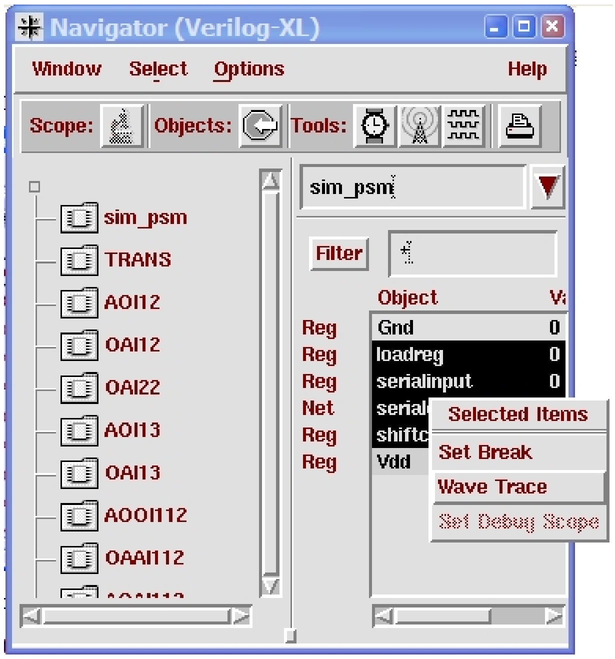
In the Post Processing window, click “Run” button, shown in the above figure, Verilog begins the simulation, we can see the simulation time in the Terminal window, about 0.5 s.

After the simulation is done, we can see the waveforms. Go to “Tools->Navigator”, the following window will pop up.



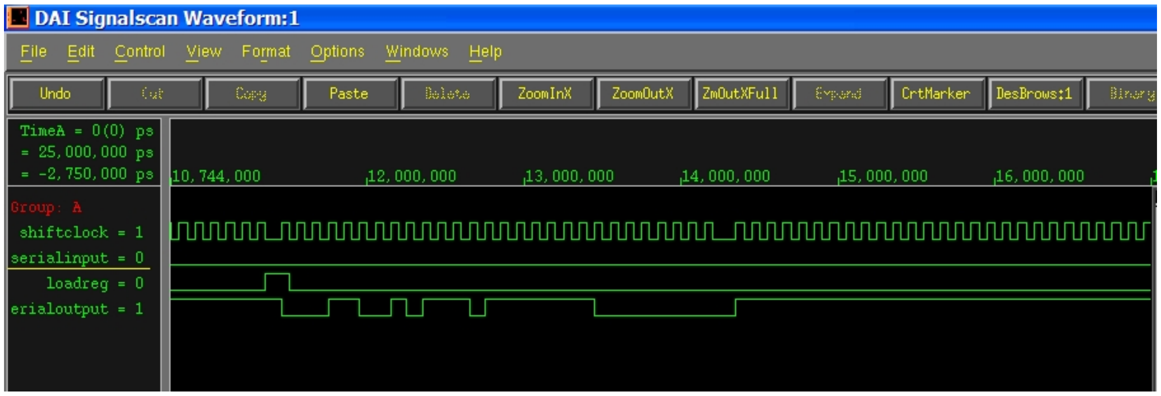
	Object	Va
Reg	Gnd	0
Reg	loadreg	0
Reg	serialinput	0
Net	serialoutput	1
Reg	shiftclock	0
Reg	Vdd	1

In the Navigator window, click the “sim_psm” in the left frame, so there are some signals show up in the right window. Select the signals and click right button, select “”, shown in the following figure:



	Object	Va
Reg	Gnd	0
Reg	loadreg	0
Reg	serialinput	0
Net	serial	
Reg	shiftc	
Reg	Vdd	

Then the waveform window will show up. Click the "ZmOutXFull", you can see the waveform show in the following, this waveform is totally same as the VHDL simulation result.



Summary:

From the above flow, we can see that using FROSTY, the post-layout time can be saved greatly. However, in this flow, the real delay information of the design is not inserted, which will be a future work of FROSTY.