EE4415 Integrated Digital Design

Lab (Optional) : Simulation Using VCS

Introduction

The first part of this lab manual will introduce you with the Verilog compiler and simulator from Synopsys. In the second part, you are required to design a 6-bit counter using Verilog and simulate the design using Synopsys VCS.

Part I: Introduction to Synopsys VCS

Synopsys VCS is introduced in this part where VCS stands for Verilog Compiled Simulator. Depending upon platform, VCS first generates C code from the Verilog source, then it compiles and links the object files to the simulation engine to create an executable. The command `vcs –help` will list compile options, runtime options and environment variables. The basic invocation is as follows:

```
vcs file1.v file2.v file3.v filen.v
```

and the file "simv" is the Verilog executable generated by VCS. The commonly used command line options are listed below:

`-I`	Compile for interactive simulation
`-Mupdate`	Incremental compilation (only changed files are compiled)
`-R`	Run after compilation
`-RI`	Run interactively (with GUI) after compilation
`-line`	Enable line, next and trace commands
`-f <filename>`	Read host command arguments from file
`-l <filename>`	Set log file name
`-s`	Stop simulation before it begins; enter interactive mode

In the following, VCS is introduced with a simple example. The Verilog code is shown below.

```
//simple.v
```

```
module simple;

reg clk, a;
wire b;

initial           // initialize registers and generate the clk
   begin
      a = 0;
      clk = 0;
      forever #25 clk = !clk;
   end

always @ (posedge clk)
   a = !a;  //toggle a every clock cycle

inv_things inv1 (a,b);  //instantiation of module inv_things

always @ (b)
   $display ("a = %b and b = %b",a,b);

initial
   #1000 $stop;         // stop after 1000 time steps

endmodule

module inv_things (a,b);

   input a;
   output b;
   reg b;

   always @ (a)
      b <= #5!a;  // invert a,  delay 5 and assign to b

endmodule
```

A great way to study/debug your HDL code is by means of the GUI (like having a programmable logic analyzer hooked to the simulator). VirSim GUI is introduced in the following by using the previous example.

1. To start, invoke VCS by typing:

```
vcs simple.v –RI –line –s –Mupdate
```

You will see the software interface that looks like the following captured screen. There are five subordinate windows:

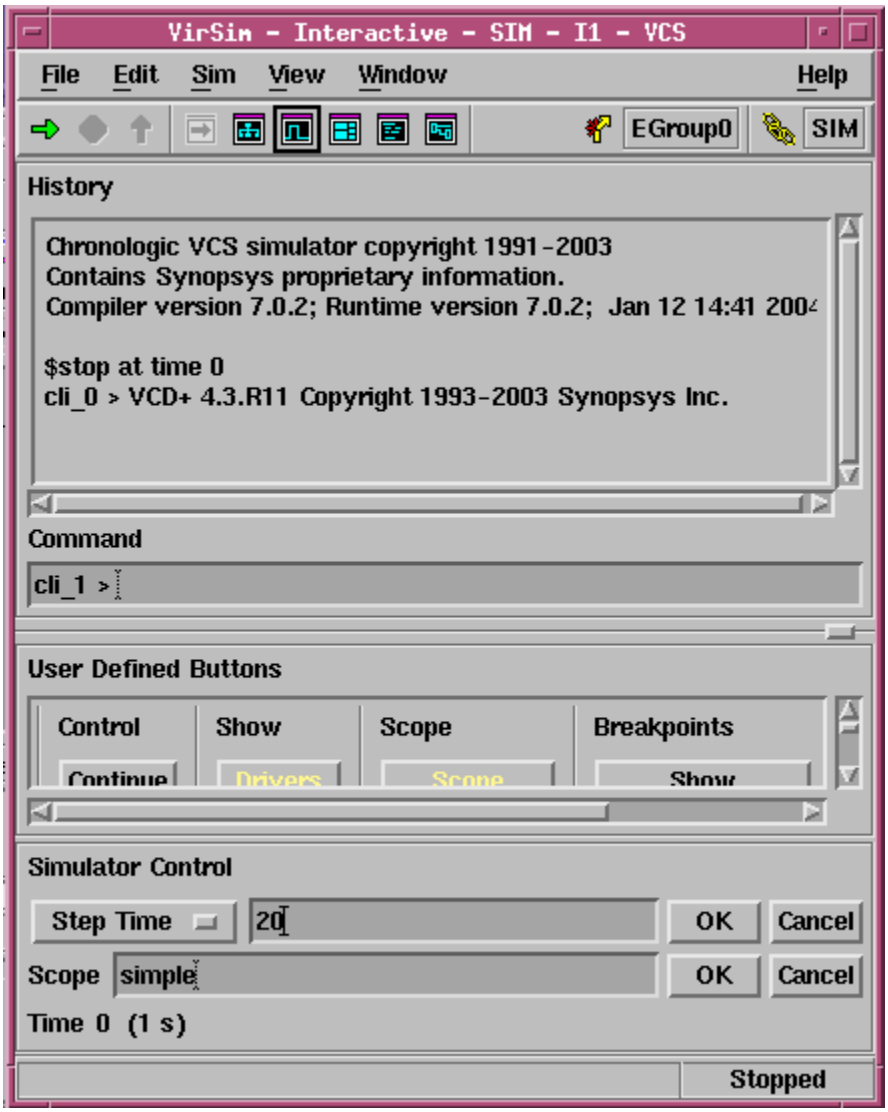
Hierarchy: navigate around the design hierarchy

Waveform: display simulation output as waves

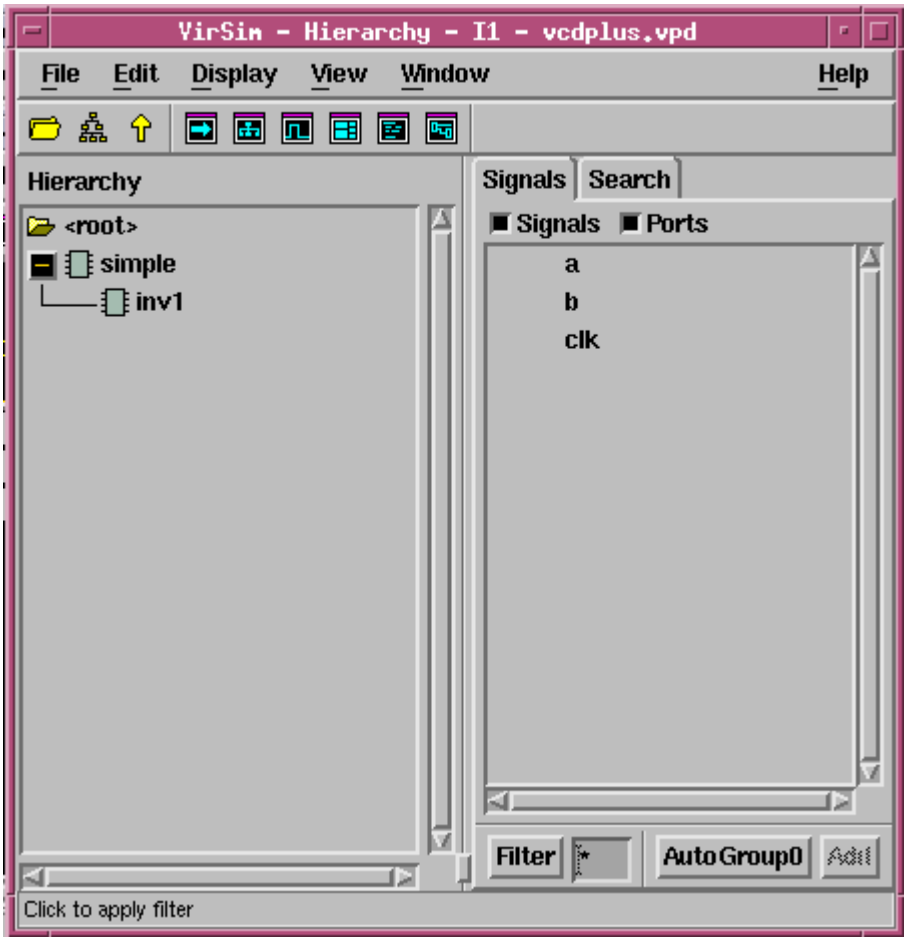
Register: create custom views of signals

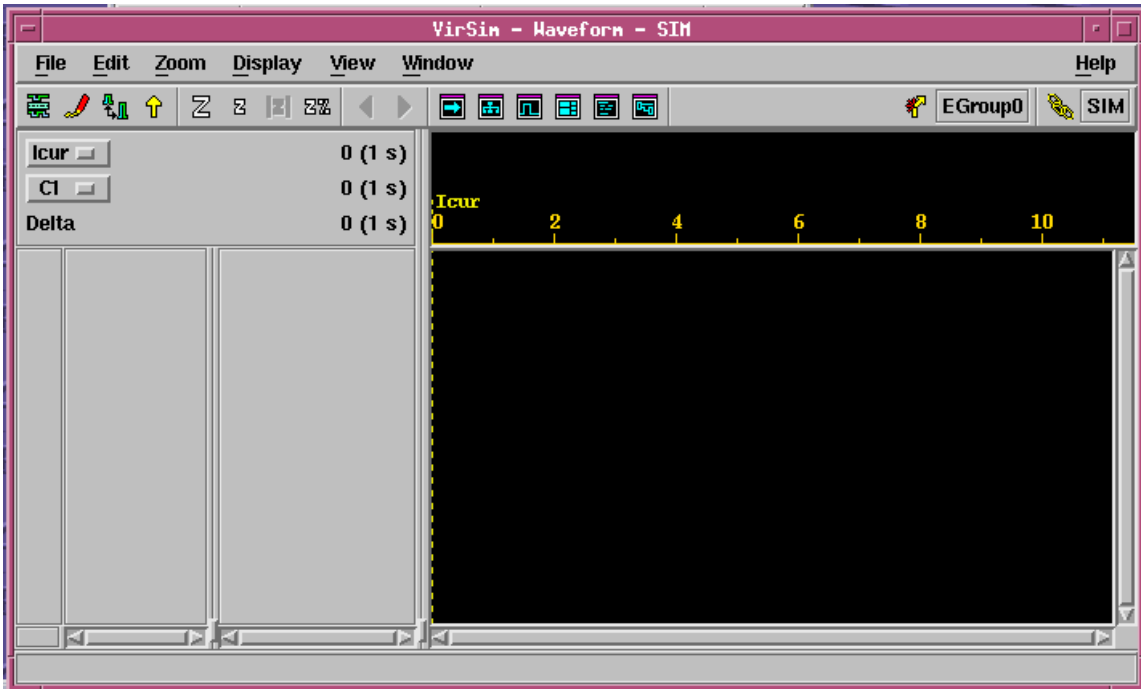
Source: show source code, set breakpoint etc.

Logic: show connectivity info up & down hierarchy

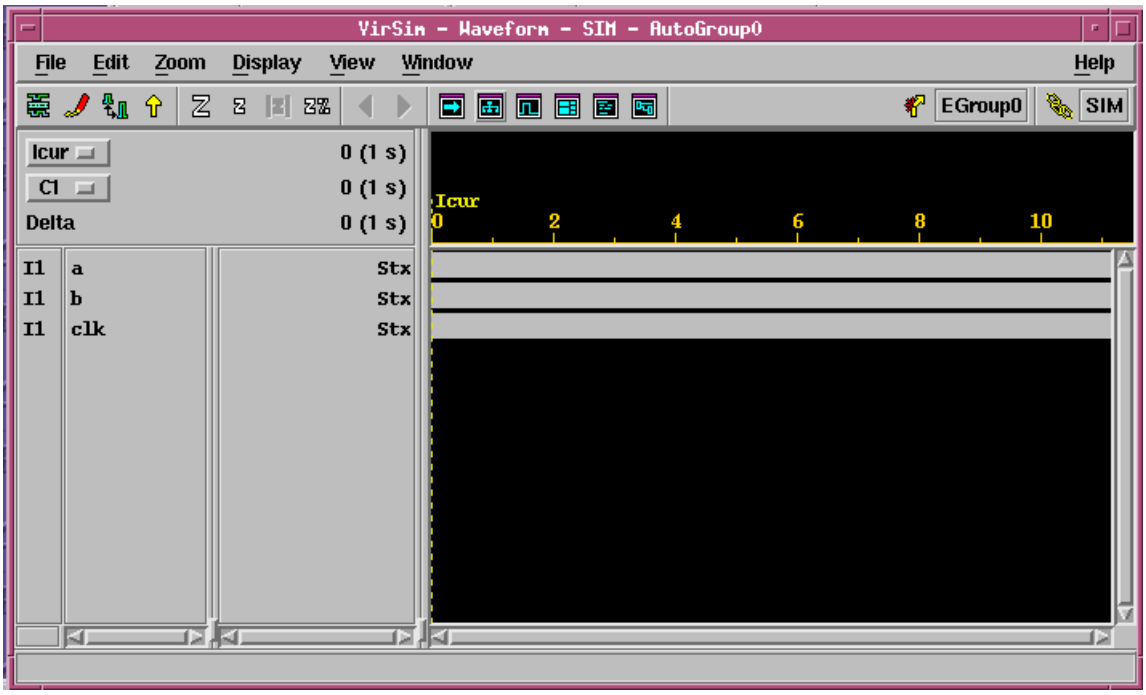


2. Left click the hierarchy & waveform buttons, two new windows will open. Hierarchy Browser window shows the structure of your design graphically. Waveform window looks like a logic analyzer connected to your design.

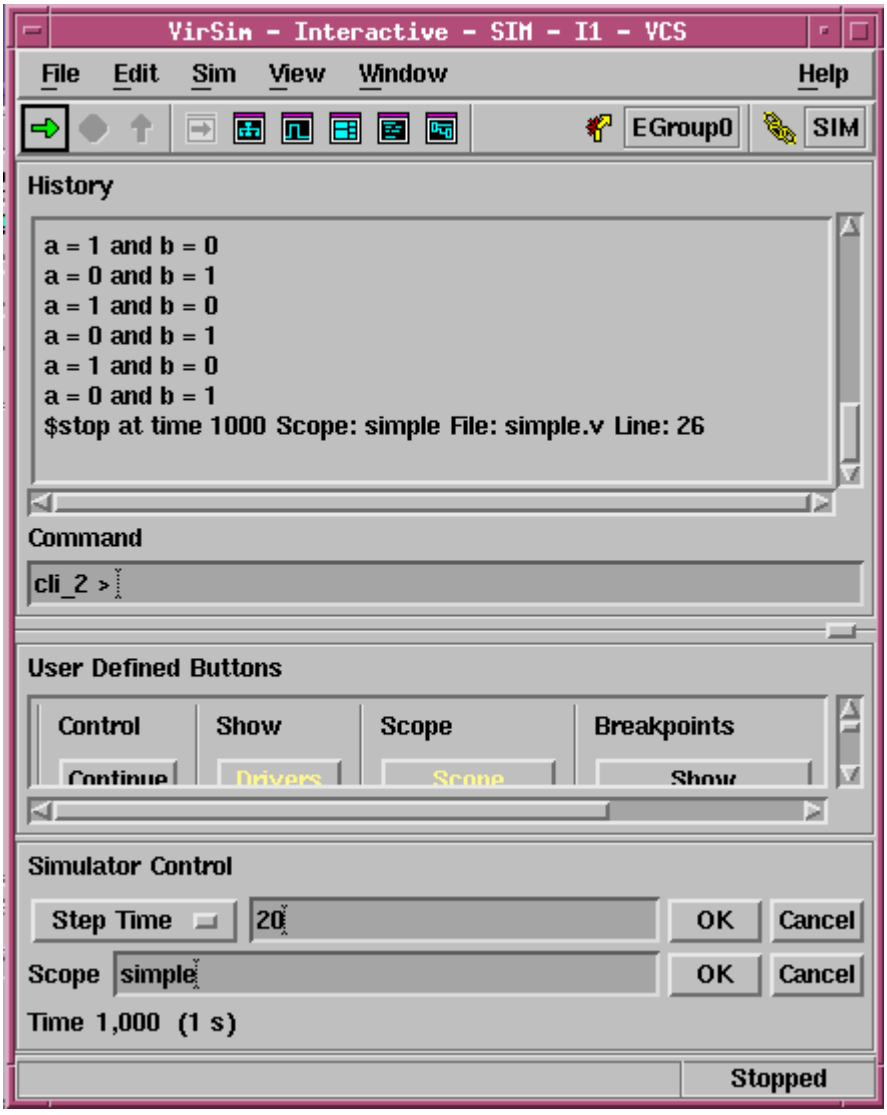

VirSim - Hierarchy - I1 - vcdplus.vpd
File
Edit
Display
View
Window
Help
Hierarchy
<root>
simple
inv1
Signals
Search
Signals
Ports
a
b
clk
Filter
AutoGroup0
Click to apply filter


VirSim - Waveform - SIM
File
Edit
Zoom
Display
View
Window
Help
EGroup0
SIM
Icur
0 (1 s)
C1
0 (1 s)
Delta
0 (1 s)
Icur
0
2
4
6
8
10

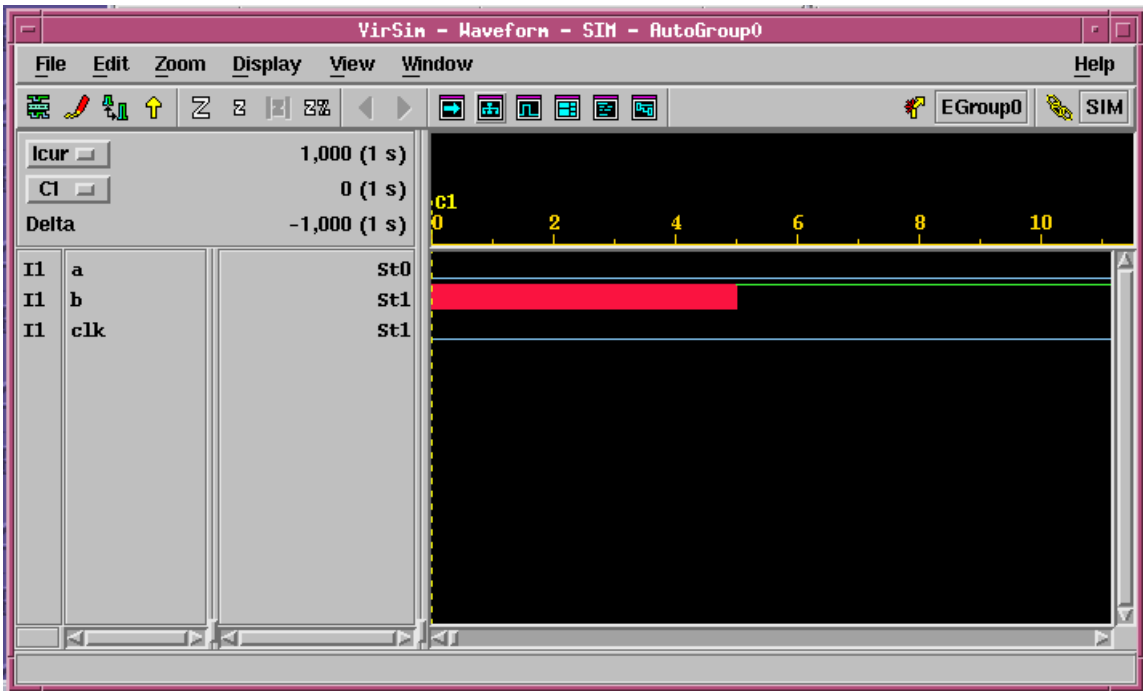
3. Select the gray simple icon from the Hierarchy Browser window using the Middle Mouse Button and drag-drop it into the Waveform window. Notice how Waveform window updates with the name of all signals in module simple (a, b & CLK).



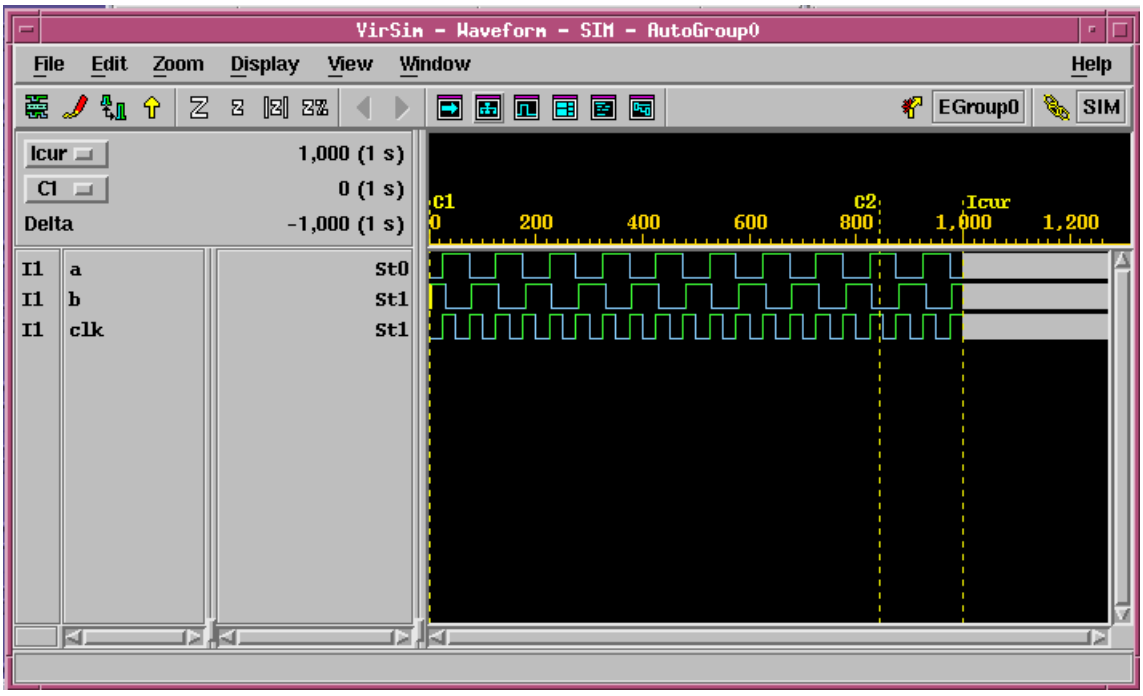
4. Click the run simulation button in the Interactive Window; you will see the simulation output appears in the Interactive window.



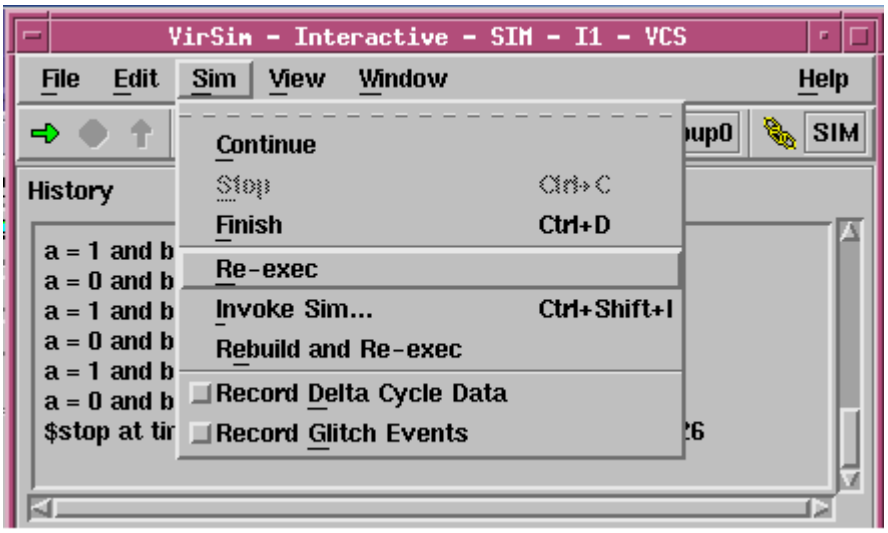
Also notice Waveform window update with signal data.



5. Handy buttons allow zoom in, out etc. click the Z% button now and select 100% from the popup. Notice that the window redraws to show all signal activity logged so far.

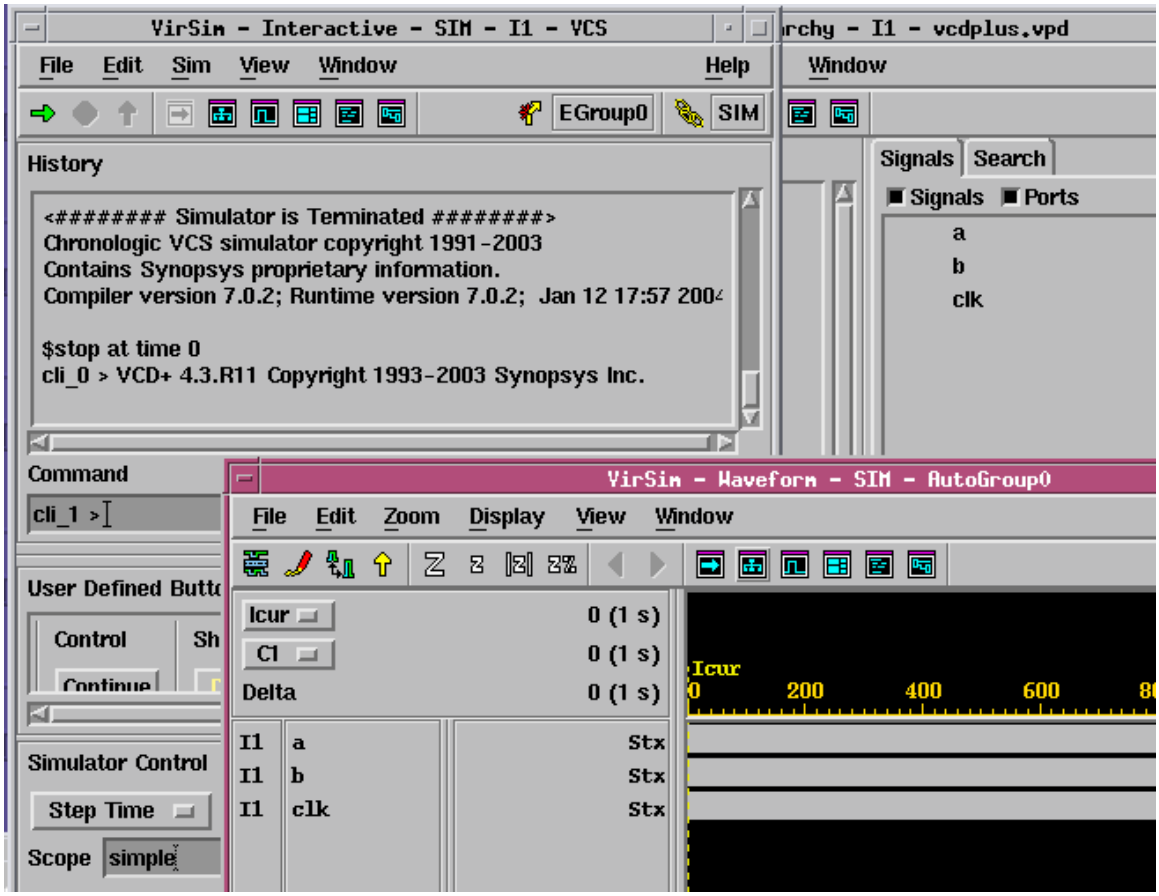


6. In the Interactive window, click the Sim pull-down & choose Re-exec. The simulator resets to time zero and the Waveform window grays out all waveforms again because no data has been logged.



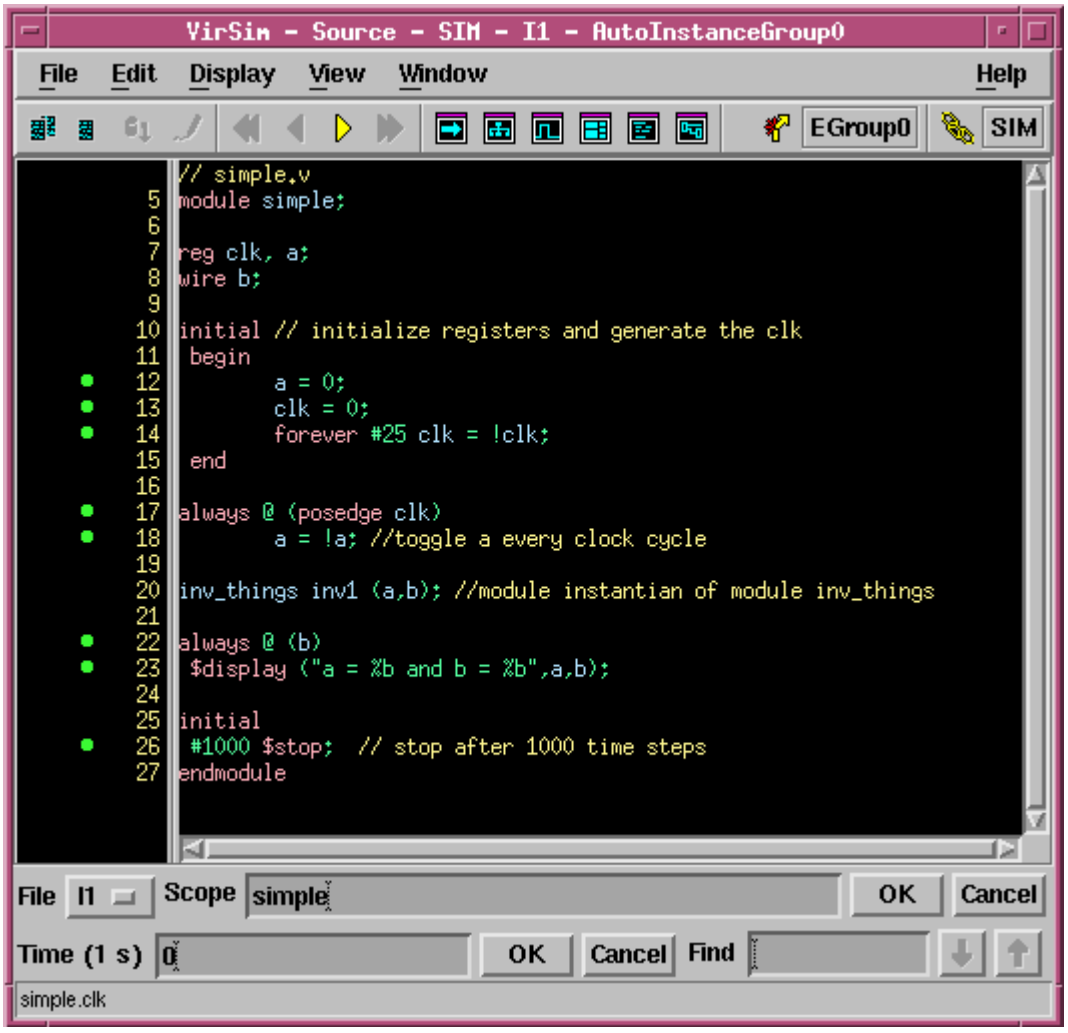
NOTE: Even if source-code has been changed, Re-exec simply restarts the SAME simulation; it does NOT recompile the source.

The following screen shows the Interactive window and the Waveform window after Re-execration.

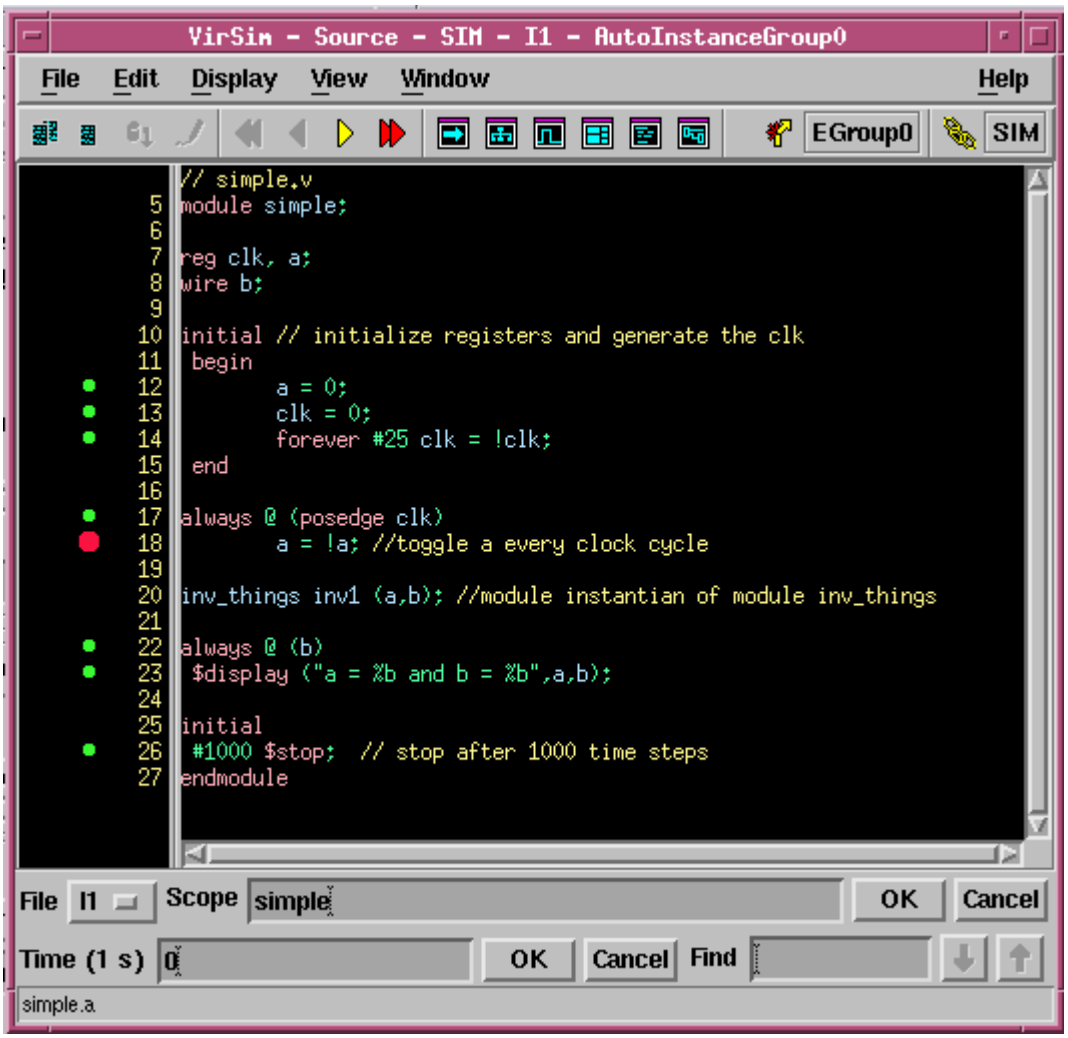

VirSim - Interactive - SIM - I1 - VCS
File Edit Sim View Window Help
EGroup0
SIM
History
<######## Simulator is Terminated ########>
Chronologic VCS simulator copyright 1991-2003
Contains Synopsys proprietary information.
Compiler version 7.0.2; Runtime version 7.0.2; Jan 12 17:57 200
$stop at time 0
cli_0 > VCD+ 4.3.R11 Copyright 1993-2003 Synopsys Inc.
Command
cli_1 >
User Defined Butt
Control
Continue
Simulator Control
Step Time
Scope simple
rchy - I1 - vcdplus.vpd
Window
Signals Search
Signals Ports
a
b
clk
VirSim - Waveform - SIM - AutoGroup0
File Edit Zoom Display View Window
Icur 0 (1 s)
C1 0 (1 s)
Delta 0 (1 s)
I1 a Stx
I1 b Stx
I1 clk Stx
Icur
0 200 400 600

7. Click the Source button in the Interactive window, the Source window opens, Source window shows code execution and allows you to set & clear line breakpoints. Notice that it is empty until you click-drag an icon from Hierarchy Browser into Source window just like as you did for Waveform window earlier. Click-drag the ***simple*** icon now.

Take notice of the green dots in the Source window. These indicate executable lines, or in other words, lines where a breakpoint may be set.



8. Click on the green dot for line 18. Notice that a red-dot appears to show a breakpoint has been set at line 18. (Do not do it now, but you can clear this breakpoint by clicking the red dot.)



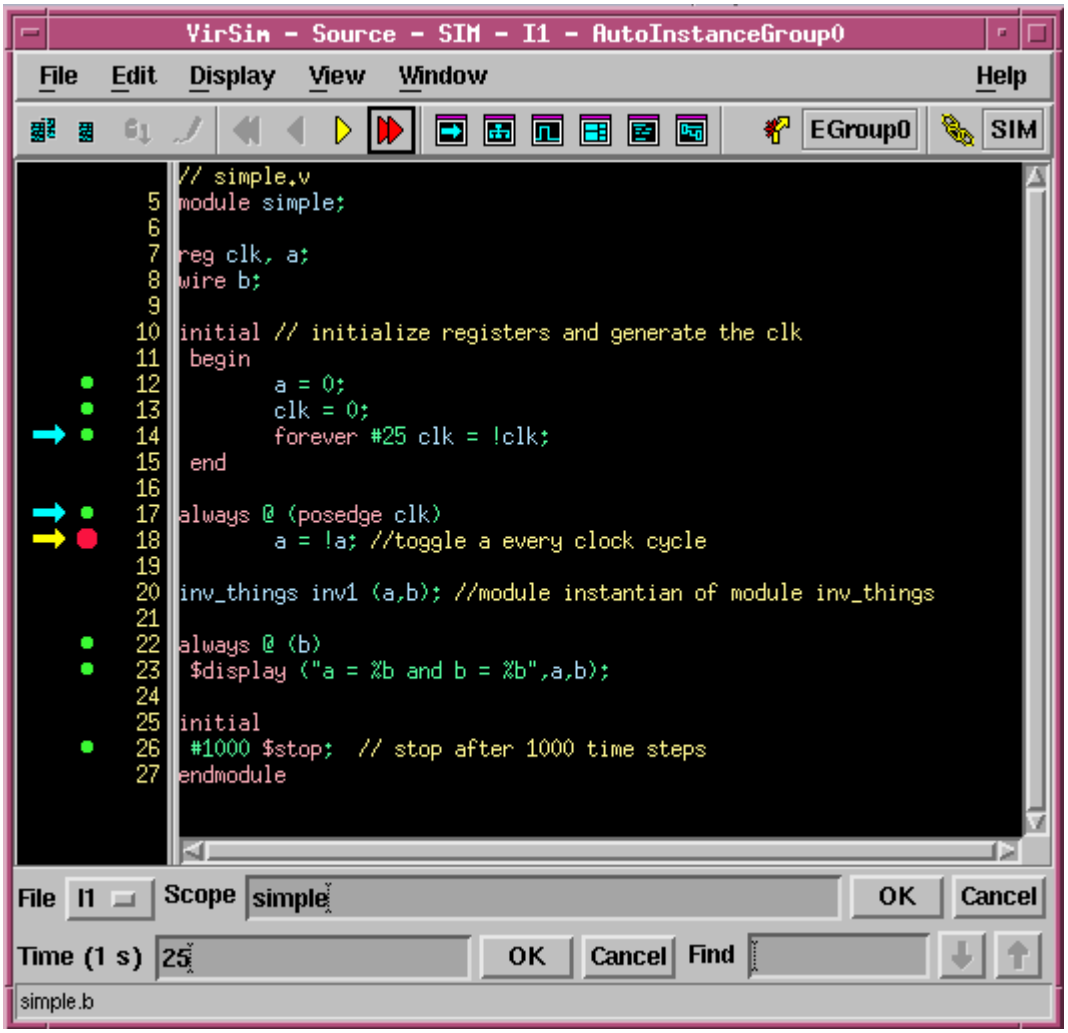
9. The red double-arrow icon at the top of source window is a quick way to advance simulation to the next breakpoint. Click it now!

If nothing happens, the problem is Source window is not part of the SIM group, so it can not control the simulation. Use the chain gadget link Source window with SIM and click the red double-arrow again. Notice that simulation hits a breakpoint at time 25, where line 18 is scheduled to be executed.

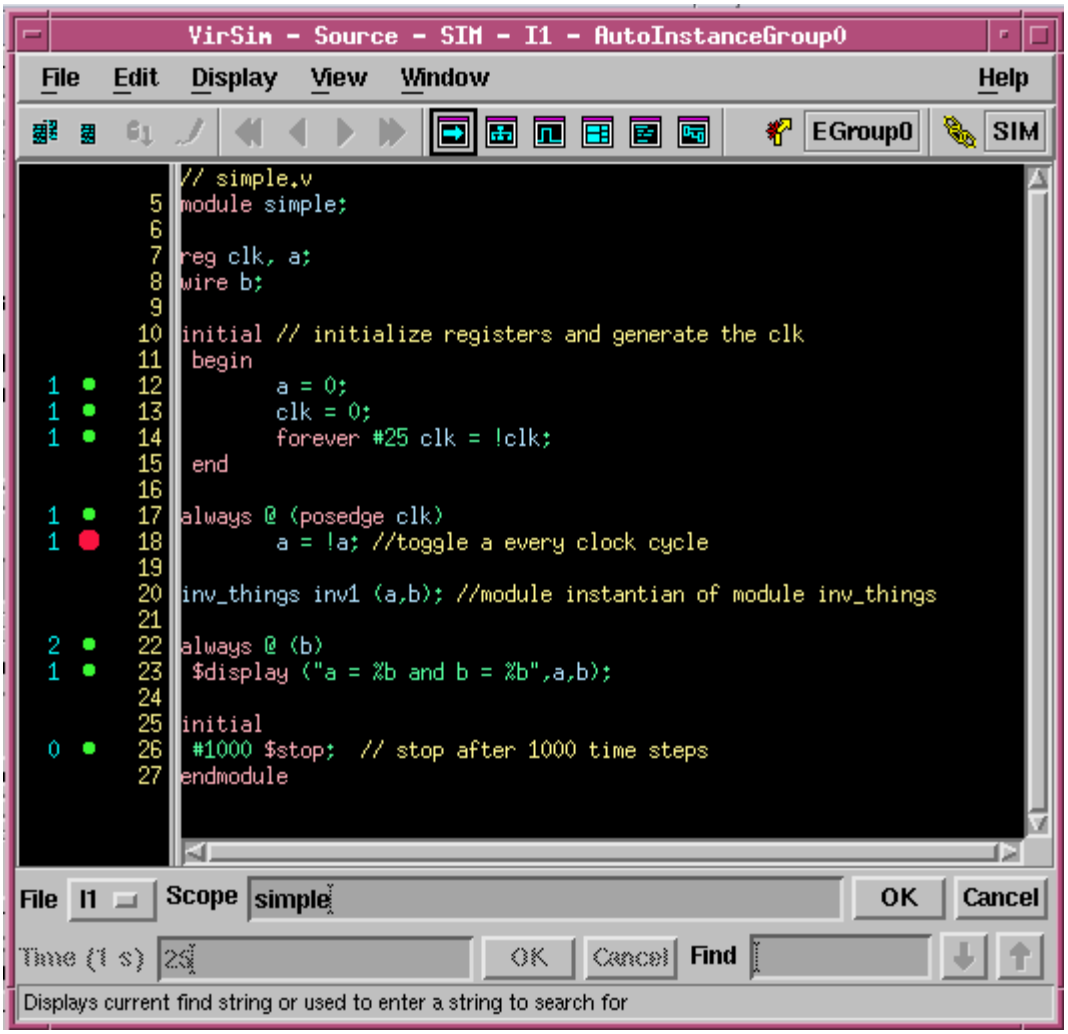
Take notice the arrow in source window:

Yellow: Indicates the currently executing time

Blue: Indicates other lines that have executed in the same time step



10. Code profiling. In the Source window click the display pull-down and select Snow Coverage. Notice the arrows have been replaced by numbers at the very left of the Source window. These numbers are a count of how many times the corresponding line of code has been executed in the simulation. This information can be very helpful in analyzing the performance of your code as well as the quality of your test vectors.



11. When you exit simulation, a new dialog box opens, asking if you want to save your current configuration. Click Yes and specify a filename (my.cfg) in the Save File Dialog window that opens. Notice the file is saved and the VirSim environment closes. Rerun VCS in the same way as before. In the Interactive window, click Load Configuration from the File pull-down. Select the configuration file you just saved (my.cfg) and click OK. VirSim returns itself to the saved configuration, windows, contents and all.

12. If you make a code change in your editor and want to re-simulate with the new code, in the Interactive window click the Sim pull-down and choose Rebuild & Re-exec. Notice that the simulator exits but the GUI remains open. Once simulator has recompiled your source code, it reconnects to the GUI. Next, click the Run arrow in the Interactive window to rerun the simulation and update all active GUI windows.

NOTE: To use Rebuild and Re-exec, you must have invoked VCS using the `–Mupdate` option.

Part II: The Design of a 6-bit Counter

1. Specifications

In this part, you are required to design a 6-bit ripple counter by using negative edge-triggered flip-flops. You should describe your design using Verilog Structural Model, i.e. describing the functionality of a device in terms of gates. A testbench is needed to test your design with a stimulus. The diagrams for the 6-bit ripple carry counter modules are shown in Fig. 2.1. Note that the counter is built with six T flip-flops.

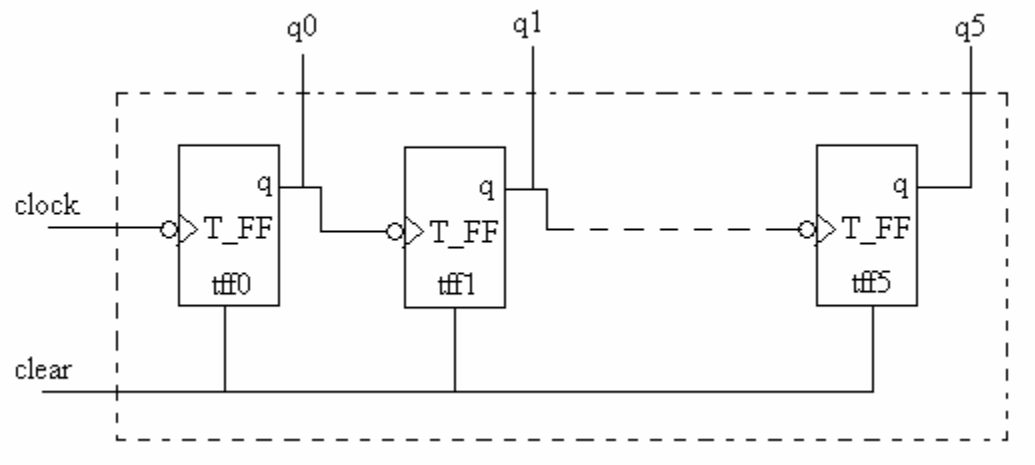


Fig. 2.1 Schematic of a 6-bit counter

Each T flip-flop is built with one D flip-flop and an inverter gate.

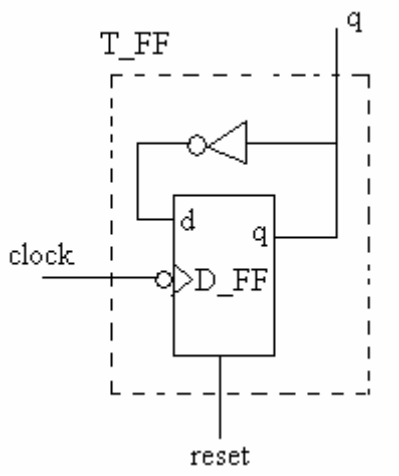


Fig. 2.2 Schematic of a T Flip-Flop

And finally, the D flip-flop constructed from basic logic gates is shown below.

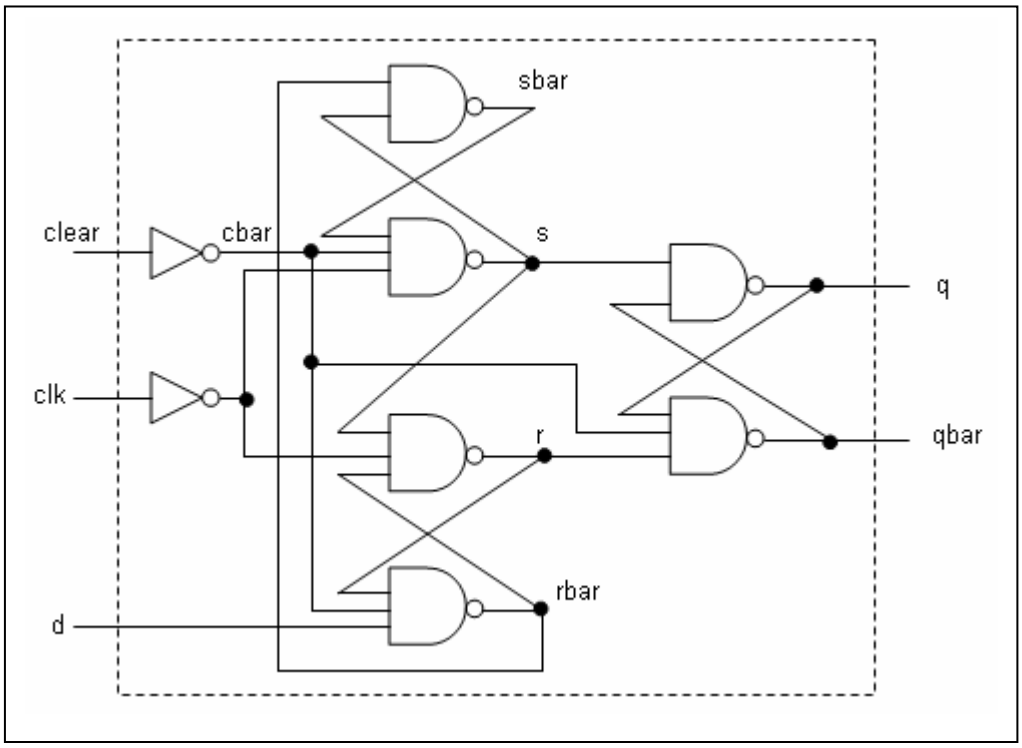


Fig. 2.3 Schematic of a D Flip-flop

2. Lab Report

Given the above diagrams, you need to write the corresponding Verilog codes, using structural model in a top-down fashion. Here are the steps:

1. Create a top module called "*counter*". The module should have two input ports, clock and clear, and 6 output ports, q0 to q5, as shown in Fig. 2.1. Your code should contain instantiation of six *T_FF* modules.
2. Create second module called "my555" which will be your clock generator. This module will have only one output port, clock.
3. Create third module called "*T_FF*" according to Fig. 2.2. You should using "*D_FF*" as the building block.
4. Create the lowest level module "*D_FF*" according to Fig. 2.3 using Verilog Primitives, NAND and NOT. Note that the pins of NAND and NOT are as follows:

- nand (out, in1, in2) or nand (out, in1, in2, in3)
- not (out, in)

5. To test the counter, you need to tie all components together to complete the design as shown in Fig. 2.4. A testbench called “Board” is required. The stimulus should check all 64 status of the counter. Note that the clear signal resets the count to zero. The simulation results should display the time stamp, the output “clear”, and q[0] to q[5].

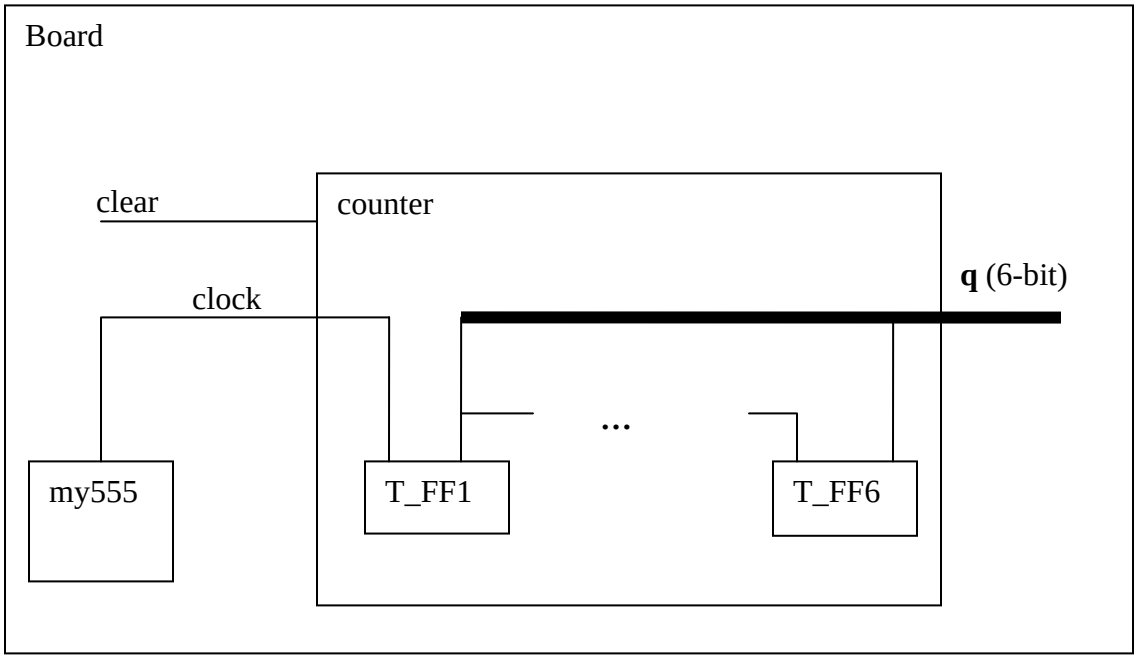


Fig. 2.4 The complete Counter for testing

Please include the Verilog codes and simulated results in your report.