

# Installing Running Simulation and Debugging with VCS & Verdi

## Installing Dependencies

### Starting with the su command

yum install [libnsl](#)

yum install redhat-lsb-core

**libnsl** → legacy network services library, needed for some binaries

**redhat-lsb-core** → Linux Standard Base, provides lsb\_release and compatibility tools

## How to compile, run, debug verilog codes using Synopsys VCS and Verdi?

### 1. Setting up environment:

```
source /home/SynopsysInstalledTools/SetupFiles/bashrc
```

```
source /home/SynopsysInstalledTools/SetupFiles/vcs_setup
```

The above commands load environment variables and tool setup scripts so VCS and Verdi can run properly

### 2. Compiling the Design/Testbench:

```
vcs -full64 full_adder_tb.v -debug_access+all -lca -kdb
```

This command compile the testbench using VCS

- **-full64** → for 64-bit compilation
- **-debug\_access+all** → Makes all signals/variables visible for debugging.  
Enables Debug visibility for all **variables, signals and hierarchy** in the design  
Without this, VCS may optimize away unused signals, and you won't be able to view them in **Verdi Waveforms**.

Recommended when you want **full observability**

#### Levels of access:

- **-debug\_access+none** → no debug access (fastest, but no visibility).
- **-debug\_access+vars** → only variables/registers.
- **-debug\_access+all** → everything (signals, variables, nets, hierarchy).
- **-lca** → Preserves combinational logic signals so you can see wires, not just registers.  
**Low-level Combinational Access**  
Ensures combinational signals like wires, intermediate nets are preserved for debugging.  
Without this Verdi may only show flip flops/register values, it loses visibility into combinational paths  
Needed to see all internal signals including wires in Verdi
- **-kdb** → options to support Verdi waveform debugging  
**Keep Debug Database**  
Instructs VCS to generate and keep a debug database that Verdi uses.

# Installing Running Simulation and Debugging with VCS & Verdi

This Database stores **signal mapping, hierarchy info, and Debug metadata**

Without this, verdi may fail to connect waveforms properly or won't display hierarchy correctly

Required for interactive debugging in verdi

## 3. Running the Simulation:

```
./simv verdi
```

This command runs the compiled design and prepares waveforms for Verdi, so that after simulation we can open novas.fsdb in Verdi for signal debugging

## 4. Opening Verdi for Debugging:

```
verdi -ssf novas.fsdb -nologo
```

verdi → launches the Synopsys Verdi GUI (waveform viewer and debugger)

-ssf novas.fsdb → loads the waveform dumpfile (novas.fsdb), that has all simulation results

.fsdb → **Fast Signal DataBase**, format used by verdi for efficient storage and access of waveform data

-nologo → skips the verdi startup logo screen

## 5. Generating waveforms in Verdi

Inside Verdi:

1. Go to “**Windows**” and then “**Interactive Debug Mode**”

This switches Verdi into a mode where we can interact with the simulator

2. Go to “**Simulation**” and then “**Invoke Simulator**”

This connects Verdi to the compiled simulation

3. Then again go to “**Simulation**” and then “**Run and Continue**”

This Starts or resumes the simulation

4. Go to “**View**” and then “**Signal List**”, select all the signals then, “**Add to waveform**” then “**New Waveform**” select the scale to 100.

This adds signals to the waveform window so you can visually observe their activity

## All the Commands in one go:

```
source /home/SynopsysInstalledTools/SetupFiles/bashrc
```

```
source /home/SynopsysInstalledTools/SetupFiles/vcs_setup
```

```
vcs -full64 <testbench>.v -debug_access+all -lca -kdb
```

```
./simv verdi
```

```
verdi -ssf novas.fsdb -nologo
```

**For generating waveforms after getting to Verdi GUI:**

Windows → Interactive Debug Mode

Simulation → Invoke Simulator

Simulation → Run/Continue

View → Signal List

# Installing Running Simulation and Debugging with VCS & Verdi

Select all, right click → Add to Waveform → New Waveform

## To save the waveform in .png

Go to file in the .fsdb window or the waveform window → then “Capture Window” → “Save as”  
choose your location, name your file and save

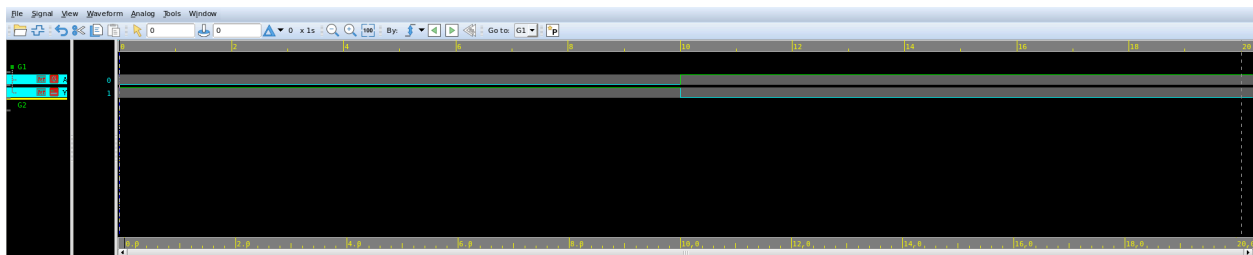
## Inverter (NOT Gate):

### not\_gate.v

```
module not_gate(  
    input A,  
    output Y);  
    assign Y = ~A;  
Endmodule ``
```

### not\_gate\_tb.v

```
`include "not_gate.v"  
module not_gate_tb;  
    reg A;  
    wire Y;  
    not_gate dut(.A(A), .Y(Y));  
initial begin  
    $fsdbDumpvars();  
    A = 0;  
    #10;  
    $display("Time=%0t : A=%b, Y=%b", $time, A, Y);  
    A = 1;  
    #10;  
    $display("Time=%0t : A=%b, Y=%b", $time, A, Y);  
    $finish;  
end  
endmodule
```



# Installing Running Simulation and Debugging with VCS & Verdi

## OR Gate:

```
module or_gate (input A, B, output Y);  
    assign Y = A | B;  
    // or(A, B, Y);  
endmodule
```

endmodule

## or\_gate\_tb.v

```
`include "or_gate.v"
```

```
module or_gate_tb;
```

```
    reg A, B;
```

```
    wire Y;
```

```
    or_gate dut(.A(A), .B(B), .Y(Y));
```

```
initial begin
```

```
    $fsdbDumpvars();
```

```
    A = 0; B = 0;
```

```
    #10;
```

```
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);
```

```
    A = 0; B = 1;
```

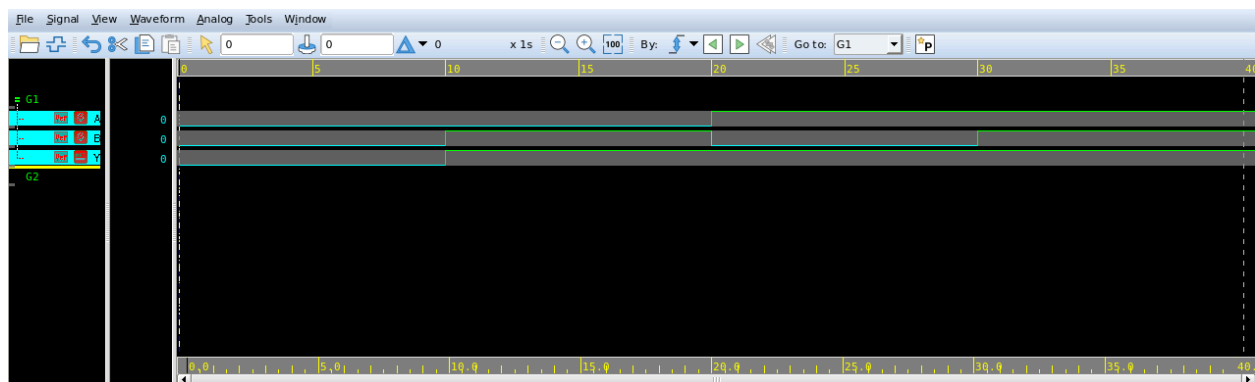
```
    #10;
```

```
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);
```

```
    A = 1; B = 0;
```

```
    #10;
```

```
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);
```



# Installing Running Simulation and Debugging with VCS & Verdi

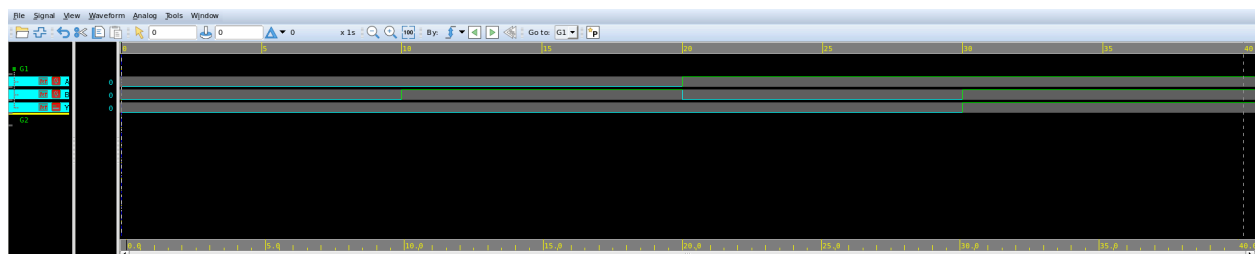
## AND Gate:

### and\_gate.v

```
module and_gate (input A, B, output Y);  
    assign Y = A&B;  
    // and(A,B,Y);  
endmodule
```

### and\_gate\_tb.v

```
`include "and_gate.v"  
module and_gate_tb;  
    reg A, B;  
    wire Y;  
    and_gate dut(.A(A), .B(B), .Y(Y));  
initial begin  
    $fsdbDumpvars();  
    A = 0; B = 0;  
    #10;  
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);  
    A = 0; B = 1;  
    #10;  
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);  
    A = 1; B = 0;  
    #10;  
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);  
    A = 1; B = 1;  
    #10;  
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);  
    $finish;  
end  
endmodule
```



# Installing Running Simulation and Debugging with VCS & Verdi

## NAND Gate:

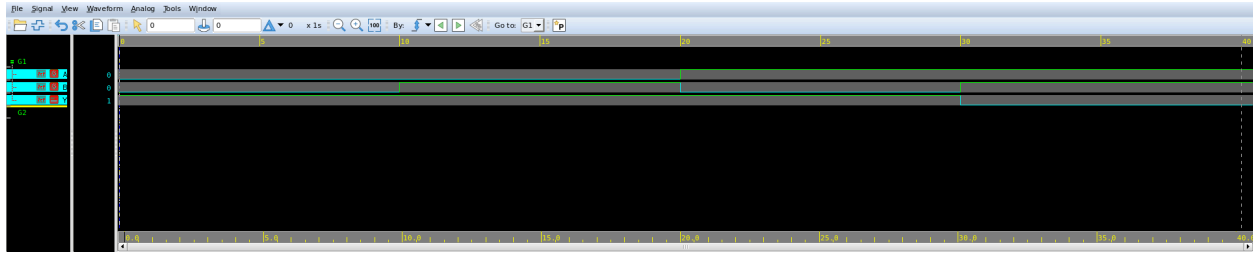
### nand\_gate.v

```
module nand_gate (  
    input A, B,  
    output Y);  
    assign Y = ~(A&B);  
// and(A,B,Y);  
endmodule
```

### nand\_gate\_tb.v

```
`include "nand_gate.v"  
module nand_gate_tb;  
    reg A, B;  
    wire Y;  
    nand_gate dut(.A(A), .B(B), .Y(Y));  
initial begin  
    $fsdbDumpvars();  
    A = 0; B = 0;  
    #10;  
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);  
    A = 0; B = 1;  
    #10;  
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);  
    A = 1; B = 0;  
    #10;  
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);  
    A = 1; B = 1;  
    #10;  
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);  
    $finish;  
end  
endmodule
```

# Installing Running Simulation and Debugging with VCS & Verdi



## NOR Gate:

### **nor\_gate.v**

```
module nor_gate(  
    input A,B,  
    output Y  
);  
    assign Y = ~(A | B);
```

```
endmodule
```

### **nand\_gate\_tb.v**

```
`include "nor_gate.v"
```

```
module nor_gate_tb;
```

```
    reg A, B;
```

```
    wire Y;
```

```
    nor_gate dut(.A(A), .B(B), .Y(Y));
```

```
initial begin
```

```
    $fsdbDumpvars();
```

```
    A = 0; B = 0;
```

```
    #10;
```

```
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);
```

```
    A = 0; B = 1;
```

```
    #10;
```

```
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);
```

```
    A = 1; B = 0;
```

```
    #10;
```

```
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);
```

```
    A = 1; B = 1;
```

```
    #10;
```

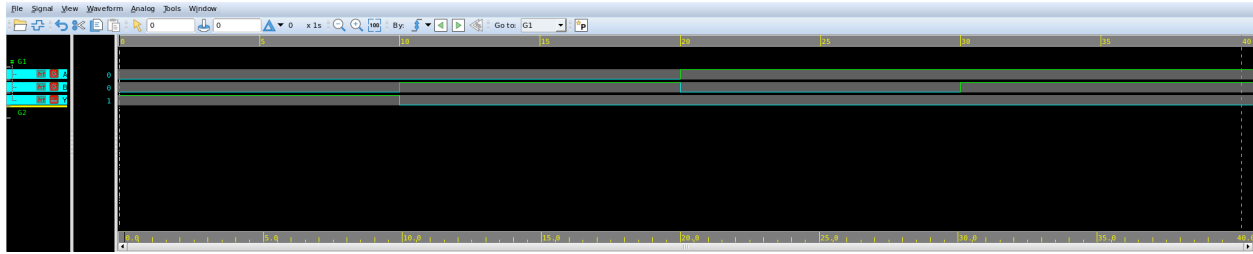
```
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);
```

```
    $finish;
```

```
end
```

```
endmodule
```

# Installing Running Simulation and Debugging with VCS & Verdi



## XOR Gate:

### xor\_gate.v

```
module xor_gate(  
    input A,B,  
    output Y  
);  
    assign Y = A ^ B;
```

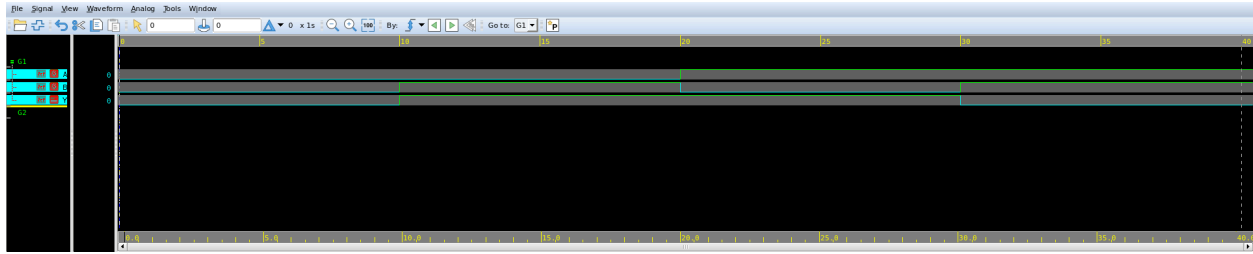
```
endmodule
```

### xor\_gate\_tb.v

```
`include "xor_gate.v"  
module xor_gate_tb;  
    reg A, B;  
    wire Y;  
    xor_gate dut(.A(A), .B(B), .Y(Y));  
initial begin  
    $fsdbDumpvars();  
    A = 0; B = 0;  
    #10;  
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);  
    A = 0; B = 1;  
    #10;  
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);  
    A = 1; B = 0;  
    #10;  
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);  
    A = 1; B = 1;  
    #10;  
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);  
    $finish;  
end  
endmodule
```



# Installing Running Simulation and Debugging with VCS & Verdi



## XNOR Gate:

### xnor\_gate.v

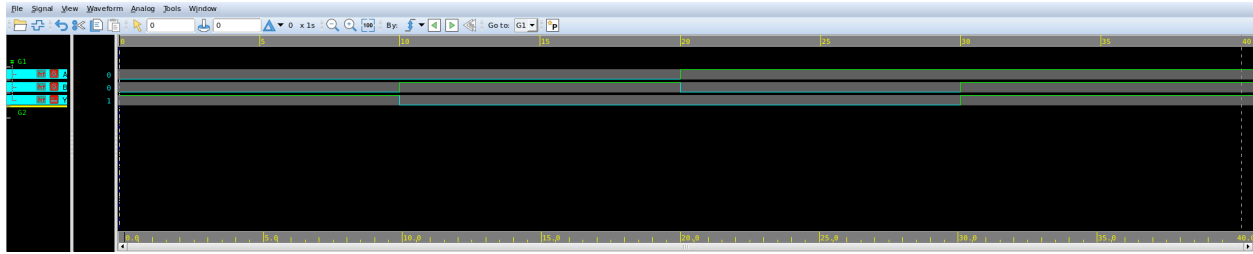
```
module xnor_gate(
    input A,B,
    output Y
);
    assign Y = ~(A ^ B);
```

```
endmodule
```

### xnor\_gate\_tb.v

```
`include "xnor_gate.v"
module xnor_gate_tb;
    reg A, B;
    wire Y;
    xnor_gate dut(.A(A), .B(B), .Y(Y));
initial begin
    $fsdbDumpvars();
    A = 0; B = 0;
    #10;
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);
    A = 0; B = 1;
    #10;
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);
    A = 1; B = 0;
    #10;
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);
    A = 1; B = 1;
    #10;
    $display("Time = %0t: A = %b, B = %b, Y = %b", $time, A, B, Y);
    $finish;
end
endmodule
```

# Installing Running Simulation and Debugging with VCS & Verdi



## SR FlipFlop

### sr\_ff:

```
module sr_ff(input clk, s, r, output reg q);
always @ (posedge clk) begin
    case ({s, r})
        2'b00:
            q <= q;
        2'b01:
            q <= 0;
        2'b10:
            q <= 1;
        2'b11:
            q <= 1'bx;
    endcase
end
```

end

endmodule

### sr\_ff\_tb:

```
`include "sr_ff.v"
```

```
module sr_ff_tb();
```

```
    reg clk, s, r;
```

```
    wire q;
```

```
    sr_ff uut(.clk(clk),
```

```
    .s(s),
```

```
    .r(r),
```

```
    .q(q));
```

```
initial begin
```

```
    clk = 0;
```

```
    forever #5 clk = ~clk; // 10 time unit period
```

```
end
```

```
initial begin
```

```
    $fsdbDumpvars;
```

# Installing Running Simulation and Debugging with VCS & Verdi

```
// Initial state
s = 0; r = 0;
#12 s = 1; r = 0; // Set
$display("Time = %0t, s = %b, r = %b, q = %b", $time, s, r, q);
#20 s = 0; r = 0; // Hold
$display("Time = %0t, s = %b, r = %b, q = %b", $time, s, r, q);
#20 s = 0; r = 1; // Reset
$display("Time = %0t, s = %b, r = %b, q = %b", $time, s, r, q);
#20 s = 0; r = 0; // Hold
$display("Time = %0t, s = %b, r = %b, q = %b", $time, s, r, q);
#20 s = 1; r = 1; // Invalid
$display("Time = %0t, s = %b, r = %b, q = %b", $time, s, r, q);
#20 s = 0; r = 0; // Back to Hold
$display("Time = %0t, s = %b, r = %b, q = %b", $time, s, r, q);

#20 $finish;

end
endmodule
```



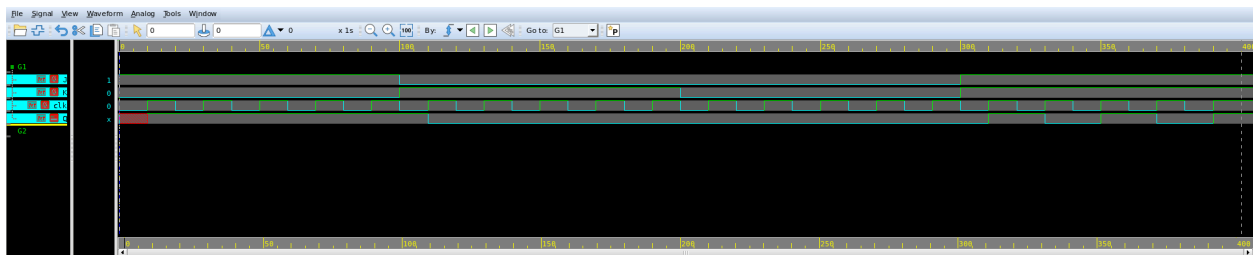
## JKFlip Flop:

```
module jk_ff(input clk, J, K, output reg Q);
always @ (posedge clk) begin
    case({J, K})
        2'b00: Q <= Q;
        2'b01: Q <= 1'b0;
        2'b10: Q <= 1'b1;
        2'b11: Q <= ~Q;
        default:
            Q <= Q;
    endcase
end
endmodule

jk_ff_tb:
`include "jk_ff.v"
```

# Installing Running Simulation and Debugging with VCS & Verdi

```
module jk_ff_tb();
    reg clk, J, K;
    wire Q;
    jk_ff uut(.clk(clk),
        .J(J),
        .K(K),
        .Q(Q));
initial begin
    clk = 0;
    forever #5 clk = ~clk; // 10 time unit period
end
initial begin
    $fsdbDumpvars;
    // Initial state
    J = 0; K = 0;
    #12 J = 0; K = 1; // Set
    #2 $display("Time=%0t J=%b K=%b Q=%b", $time, J, K, Q);
    #20 J = 1; K = 0; // Hold
    #2 $display("Time=%0t J=%b K=%b Q=%b", $time, J, K, Q);
    #20 J = 1; K = 1; // Reset
    #2 $display("Time=%0t J=%b K=%b Q=%b", $time, J, K, Q);
    #20 J = 0; K = 0; // Hold
    #2 $display("Time=%0t J=%b K=%b Q=%b", $time, J, K, Q);
    #20 J = 0; K = 1; // Toggle
    #2 $display("Time=%0t J=%b K=%b Q=%b", $time, J, K, Q);
    #20 J = 1; K = 0; // Hold
    #2 $display("Time=%0t J=%b K=%b Q=%b", $time, J, K, Q);
    #20 $finish;
end
endmodule
```



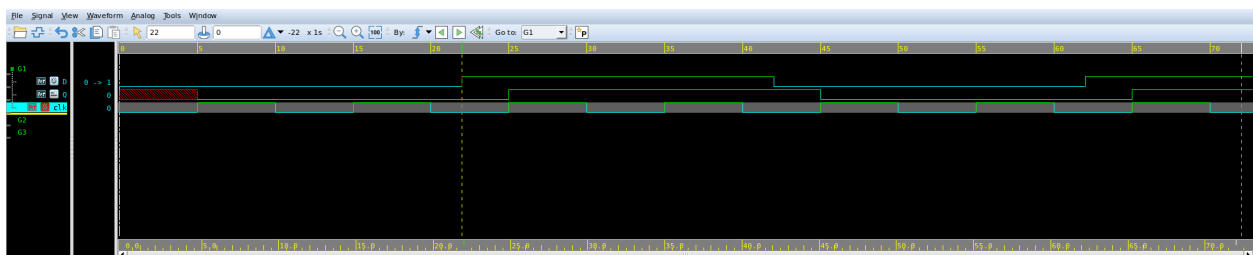
## D Flip Flop:

### d\_ff.v

```
module d_ff(input clk, D, output reg Q);
always @ (posedge clk) begin
    Q <= D;
end
```

# Installing Running Simulation and Debugging with VCS & Verdi

```
end
endmodule
d_ff_tb.v:
`include "d_ff.v";
module d_ff_tb;
    reg clk, D;
    wire Q;
    d_ff uut(.clk(clk), .D(D), .Q(Q));
initial begin
    clk = 0;
    forever #5 clk = ~clk; // 10 time unit period
end
initial begin
    $fsdbDumpvars;
    D = 1'b0; #12
    D = 1'b0;
    #10
    $display("Time = %0t, D = %b, Q = %b", $time, D, Q);
    D = 1'b1;
    #10
    $display("Time = %0t, D = %b, Q = %b", $time, D, Q);
    D = 1'b1;
    #10
    $display("Time = %0t, D = %b, Q = %b", $time, D, Q);
    D = 1'b0;
    #10
    $display("Time = %0t, D = %b, Q = %b", $time, D, Q);
    D = 1'b0;
    #10
    $display("Time = %0t, D = %b, Q = %b", $time, D, Q);
    D = 1'b1;
    #10
    $display("Time = %0t, D = %b, Q = %b", $time, D, Q);
    $finish;
end
endmodule
```



## T Flip Flop:

# Installing Running Simulation and Debugging with VCS & Verdi

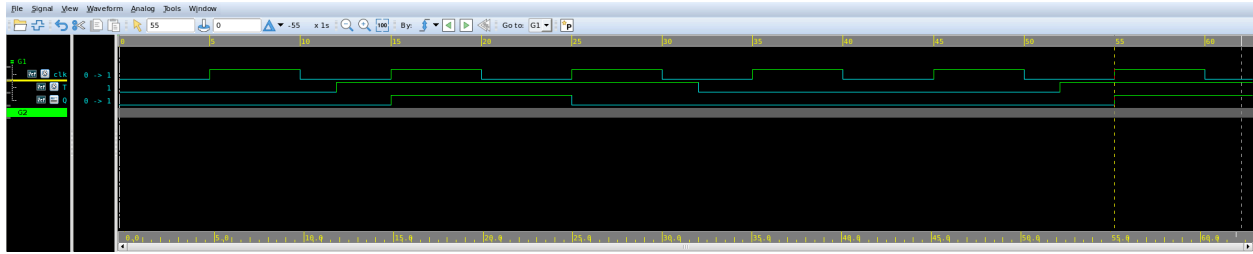
## **t\_ff.v**

```
module t_ff (input clk, T, output reg Q);
    initial Q = 0;
    always @(posedge clk) begin
        if (T == 1)
            Q = ~Q;
        else
            Q <= Q;
        end
    endmodule
```

## **t\_ff\_tb.v**

```
`include "t_ff.v"
module t_ff_tb();
    reg clk, T;
    wire Q;
    t_ff uut (.clk(clk), .T(T), .Q(Q));
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
        end
    initial begin
        $fsdbDumpvars();
        T = 1'b0; #12;
        $display("Time=%0t, T=%b, Q=%b", $time, T, Q);
        T = 1'b1; #10;
        $display("Time=%0t, T=%b, Q=%b", $time, T, Q);
        T = 1'b1; #10;
        $display("Time=%0t, T=%b, Q=%b", $time, T, Q);
        T = 1'b0; #10;
        $display("Time=%0t, T=%b, Q=%b", $time, T, Q);
        T = 1'b0; #10;
        $display("Time=%0t, T=%b, Q=%b", $time, T, Q);
        T = 1'b1; #10;
        $display("Time=%0t, T=%b, Q=%b", $time, T, Q);
        $finish;
    end
endmodule
```

# Installing Running Simulation and Debugging with VCS & Verdi



## Full Adder:

### fulladder.v

```
module full_adder (  
    input A, B, C_in,  
    output Y, C_out);  
    assign Y = A^B^C_in;  
    assign C_out = (A&B) | (C_in & (A^B));  
endmodule
```

### fulladder\_tb.v

```
`include "full_adder.v"  
module full_adder_tb;  
    reg A, B, C_in;  
    wire Y, C_out;  
    full_adder uut(.A(A), .B(B), .C_in(C_in), .Y(Y), .C_out(C_out));  
initial begin  
    $fsdbDumpvars();  
    A = 0; B = 0; C_in = 0;  
    #10;  
    $display("Time=%0t : A=%b, B=%b, C_in = %b, Y=%b, C_out=%b", $time, A, B, C_in, Y, C_out);  
    A = 0; B = 0; C_in = 1;  
    #10;  
    $display("Time=%0t : A=%b, B=%b, C_in = %b, Y=%b, C_out=%b", $time, A, B, C_in, Y, C_out);  
    A = 0; B = 1; C_in = 0;  
    #10;  
    $display("Time=%0t : A=%b, B=%b, C_in = %b, Y=%b, C_out=%b", $time, A, B, C_in, Y, C_out);  
    A = 0; B = 1; C_in = 1;  
    #10;  
    $display("Time=%0t : A=%b, B=%b, C_in = %b, Y=%b, C_out=%b", $time, A, B, C_in, Y, C_out);
```

# Installing Running Simulation and Debugging with VCS & Verdi

```
A = 1; B = 0; C_in = 0;
#10;
$display("Time=%0t : A=%b, B=%b, C_in = %b, Y=%b, C_out=%b", $time, A, B, C_in, Y, C_out);
A = 1; B = 0; C_in = 1;
#10;
$display("Time=%0t : A=%b, B=%b, C_in = %b, Y=%b, C_out=%b", $time, A, B, C_in, Y, C_out);
A = 1; B = 1; C_in = 0;
#10;
$display("Time=%0t : A=%b, B=%b, C_in = %b, Y=%b, C_out=%b", $time, A, B, C_in, Y, C_out);
A = 1; B = 1; C_in = 1;
#10;
$display("Time=%0t : A=%b, B=%b, C_in = %b, Y=%b, C_out=%b", $time, A, B, C_in, Y, C_out);
$finish;
end
endmodule
```

