VLSI Design
Tutorial - Icarus Verilog & GTKWave

Prof. Abhishek Srivastava

CVEST, IIIT Hyderabad

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

About Icarus Verilog & GTKWave

- ***Icarus Verilog*** is a Verilog simulation and synthesis tool.
  – It operates as a compiler, compiling source code written in Verilog (IEEE-1364) into some target format.
  – For batch simulation, the compiler can generate an intermediate form called vvp assembly. This intermediate form is executed by the "vvp" command.
  – For synthesis, the compiler generates netlists in the desired format.
- ***GTKWave*** is a fully featured GTK+ based wave viewer for Verilog VCD/EVCD files.

Instructions to Install iverilog & GTKWave

- Instructions for **Ubuntu/Linux (Recommended)** :
  To install iverilog : Execute the following bash command :
  – *sudo apt-get install iverilog*
  To install GTKWave : Execute the following bash command :
  – *sudo apt-get install gtkwave*

Instructions to run iverilog & GTKWave

- **To run iverilog** :
  - Open the terminal.
  - Go to the directory containing the verilog code and testbench
  - Type *iverilog -o <output> <testbench.v> <code.v>*.
  - To execute the output file on terminal : Type *vvp <output>*.
- **To run GTKWave** :
  - Open the terminal.
  - Go to the directory containing the $< filename.vcd >$.
  - Type *gtkwave <filename.vcd> &*.
  - In the left pane of the pop-up window, click on the
  - Signals appear in the lower left side of the window
  - Drag and drop the signals to be viewed in the waveform window
  - Go to *Time > Zoom > Zoom Best Fit* to see the whole waveform

Example verilog codes and testbench - 1

- 1x2 *Decoder* verilog code

```
`timescale 1ns / 1ps
module two_decoder(a,en,y1,y2);
input a,en;
output y1,y2;
and A1(y1,~a,en);
and A2(y2,a,en);
endmodule
```

Figure: two_decoder.v

- 1x2 *Decoder* verilog testbench

```
`timescale 1ns / 1ps
module two_decoder_test;

	// Inputs
	reg a;
	reg en;

	// Outputs
	wire y1;
	wire y2;

	// Instantiate the Unit Under Test (UUT)
	two_decoder uut (
		.a(a),
		.en(en),
		.y1(y1),
		.y2(y2)
	);

	initial begin
		$dumpfile("two_decoder_test.vcd");
		$dumpvars(0,two_decoder_test);
		// Initialize Inputs
		a = 1'b0;
		en = 1'b0;
		#100 $finish;
		end

		always #5 a = ~a;
		always #10 en = ~en;
		always@(a)
		$monitor("time = %t \t y1 = %d \t y2 = %d",$time,y1,y2);


endmodule
```

Figure: two_decoder_test.v

Example verilog codes and testbench - 2

- 1x4 *Demux* verilog code

```
`timescale 1ns / 1ps
module two_demux(y,S,a,b);
input y,S;
output a,b;
and A1(a,~S,y);
and A2(b,S,y);
endmodule

module demux_four(y,S_0,S_1,a,b,c,d);
input y,S_0,S_1;
output a,b,c,d;
wire p,q;
two_demux B1(y,S_1,p,q);
two_demux B2(p,S_0,a,b);
two_demux B3(q,S_0,c,d);
endmodule
```

Figure: four_demux.v

- 1x4 *Demux* verilog testbench

```
`timescale 1ns / 1ps
module demux_four_test;

	// Inputs
	reg y;
	reg S_0;
	reg S_1;

	// Outputs
	wire a;
	wire b;
	wire c;
	wire d;

	// Instantiate the Unit Under Test (UUT)
	demux_four uut (
		.y(y),
		.S_0(S_0),
		.S_1(S_1),
		.a(a),
		.b(b),
		.c(c),
		.d(d)
	);

	initial begin
		// Initialize Inputs
		$dumpfile("demux_four_test.vcd");
		$dumpvars(0,demux_four_test);
		y = 0;
		S_0 = 0;
		S_1 = 0;
		#100 $finish;
		end
		// Add stimulus here
		always #20 y = ~y;
		always #5 S_0 = ~S_0;
		always #10 S_1 = ~S_1;
		always@(y or S_0 or S_1)
		$monitor("time = %t \t y = %d \t a = %d \t b = %d c = %d \t d = %d",$time,y,a,b,c,d);


endmodule
```

Figure: four_demux_test.v

Example verilog codes and testbench - 3

- $D - Flipflop$ verilog code

```
`timescale 1ns / 1ps
module dff(clk,rst,D,Q);
input clk,rst,D;
output reg Q;
always@(posedge clk or negedge rst)
begin
if(!rst)
Q <=  1'b0;
else
Q <= D;
end
endmodule
```

Figure: dff.v

- $D - Flipflop$ verilog testbench

```
`timescale 1ns / 1ps
module dff_test;

	// Inputs
	reg clk;
	reg rst;
	reg D;

	// Outputs
	wire Q;

	// Instantiate the Unit Under Test (UUT)
	dff uut (
		.clk(clk),
		.rst(rst),
		.D(D),
		.Q(Q)
	);

	initial begin
		// Initialize Inputs
		$dumpfile("dff_test.vcd");
		$dumpvars(0,dff_test);
		clk = 1;
		rst = 0;
		D = 0;
		#100 $finish;
		end

	always #5 clk = ~clk;
	always#20 rst = ~rst;
	always#10 D = ~D;
	always@(D or rst)
	$monitor("time = %t \t rst = %b \t D = %b \t Q = %b",$time,rst,D,Q);
endmodule
```

Figure: dff_test.v

Executing 1x2 Decoder

- Using **iverilog**
  - Open the terminal
  - Go to the directory containing verilog codes
  - Type *iverilog -o two_dec two_decoder_test.v two_decoder.v*
  - Type *vvp two_dec*
- Using **GTKWave**
  - In the terminal type *gtkwave two_decoder_test.vcd &*
  - Click on *two_decoder_test* in the left pane of the pop up window
  - Drag and drop all the signals into waveform window
  - Go to *Time > Zoom > Zoom Best Fit* to see the whole waveform

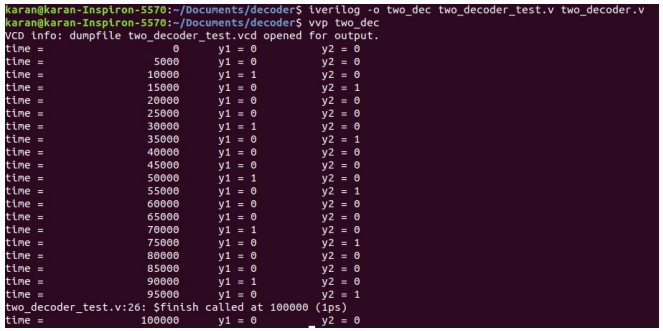

Figure: Output from iverilog

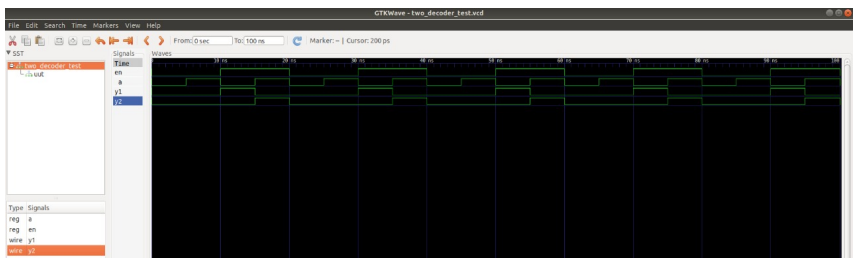

Figure: Timing Diagram in GTKWave

Executing 1x4 Demux

- Using **iverilog**
  - Open the terminal
  - Go to the directory containing verilog codes
  - Type *iverilog -o demux_four demux_four.v demux_four$_t$est.v*
  - Type *vvp demux_four*
- Using **GTKWave**
  - In the terminal type *gtkwave demux_four_test.vcd &*
  - Click on *demux_four_test* in the left pane of the pop up window
  - Drag and drop all the signals into waveform window
  - Go to *Time > Zoom > Zoom Best Fit* to see the whole waveform

```
karan@karan-Inspiron-5570:~/Documents/demux$ iverilog -o demux_four demux_four.v demux_four_test.v
karan@karan-Inspiron-5570:~/Documents/demux$ vvp demux_four
VCD info: dumpfile demux_four_test.vcd opened for output.
time =                    0      y = 0   a = 0   b = 0 c = 0     d = 0
time =                 5000      y = 0   a = 0   b = 0 c = 0     d = 0
time =                10000      y = 0   a = 0   b = 0 c = 0     d = 0
time =                15000      y = 0   a = 0   b = 0 c = 0     d = 0
time =                20000      y = 1   a = 1   b = 0 c = 0     d = 0
time =                25000      y = 1   a = 0   b = 1 c = 0     d = 0
time =                30000      y = 1   a = 0   b = 0 c = 1     d = 0
time =                35000      y = 1   a = 0   b = 0 c = 0     d = 1
time =                40000      y = 0   a = 0   b = 0 c = 0     d = 0
time =                45000      y = 0   a = 0   b = 0 c = 0     d = 0
time =                50000      y = 0   a = 0   b = 0 c = 0     d = 0
time =                55000      y = 0   a = 0   b = 0 c = 0     d = 0
time =                60000      y = 1   a = 1   b = 0 c = 0     d = 0
time =                65000      y = 1   a = 0   b = 1 c = 0     d = 0
time =                70000      y = 1   a = 0   b = 0 c = 1     d = 0
time =                75000      y = 1   a = 0   b = 0 c = 0     d = 1
time =                80000      y = 0   a = 0   b = 0 c = 0     d = 0
time =                85000      y = 0   a = 0   b = 0 c = 0     d = 0
time =                90000      y = 0   a = 0   b = 0 c = 0     d = 0
time =                95000      y = 0   a = 0   b = 0 c = 0     d = 0
demux_four_test.v:33: $finish called at 100000 (1ps)
time =               100000      y = 1   a = 1   b = 0 c = 0     d = 0
```

Figure: Output from iverilog

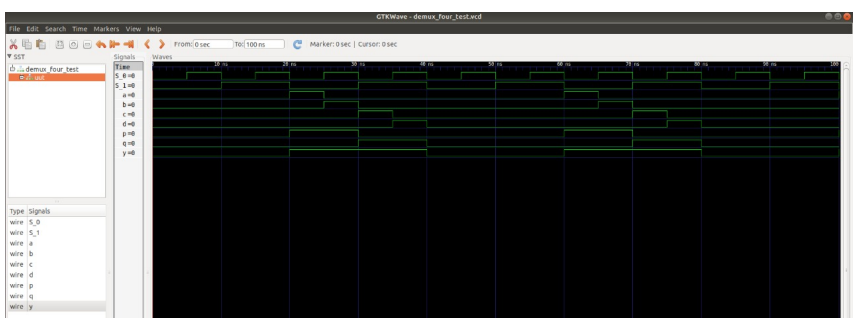

Figure: Timing Diagram in GTKWave

Executing D-Flip flop

- Using **iverilog**
  - Open the terminal
  - Go to the directory containing verilog codes
  - Type *iverilog -o dff dff.v dff$_t$est.v*
  - Type *vvp dff*
- Using **GTKWave**
  - In the terminal type *gtkwave dff_test.vcd &*
  - Click on *dff_test* in the left pane of the pop up window
  - Drag and drop all the signals into waveform window
  - Go to *Time > Zoom > Zoom Best Fit* to see the whole waveform

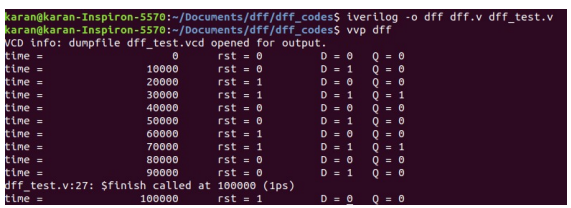

Figure: Output from iverilog

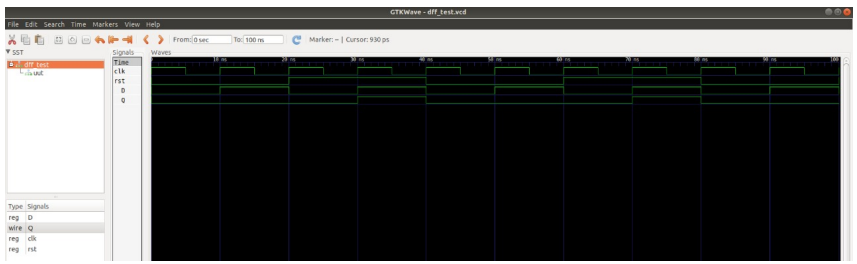

Figure: Timing Diagram in GTKWave

References

- For any queries in **Verilog** refer to : `http://www.asic-world.com/verilog/veritut.html`
- **Icarus Verilog** : `http://iverilog.icarus.com/`
- **GTKWave** : `http://gtkwave.sourceforge.net/`