GATE LEVEL MODELING

1. FULL ADDER:
   1. DIAGRAM:

* 1. CODE:

module fulladd(sum, c\_out, a, b, c\_in);

output sum, c\_out;

input a, b, c\_in;

// Internal nets

wire s1, s2, c1;

xor (s1, a, b);

and (c1, a, b);

xor (sum, s1, c\_in);

and (s2, s1, c\_in);

or (c\_out, s2, c1);

endmodule

* 1. TEST BENCH:

module

1. AND GATE:
   1. CODE:

module andgate (c,a,b);

input a,b;

output c;

and a1(c,a,b);

endmodule

* 1. TEST BENCH:

module stimulus;

reg at;

reg bt;

wire ct;

// instantiate the design block

andgate t1(ct, at, bt);

initial

begin

#5 at = 1'b0; bt=1’b0;

#10 at = 1'b0; bt=1’b1;

#5 at = 1'b1; bt=1’b0;

#5 at = 1'b1; bt=1’b1;

end

endmodule

1. 4 BIT FULL ADDER:
   1. DIAGRAM:
   2. CODE:

module fulladd4(sum, c\_out, a, b, c\_in);

output [3:0] sum;

output c\_out;

input[3:0] a, b;

input c\_in;

// Internal nets

wire c1, c2, c3;

fulladd fa0(sum[0], c1, a[0], b[0], c\_in);

fulladd fa1(sum[1], c2, a[1], b[1], c1);

fulladd fa2(sum[2], c3, a[2], b[2], c2);

fulladd fa3(sum[3], c\_out, a[3], b[3], c3);

endmodule

* 1. TEST BENCH:

module stimulus;

// Set up variables

reg [3:0] A, B;

reg C\_IN;

wire [3:0] SUM;

wire C\_OUT;

fulladd4 FA1\_4(SUM, C\_OUT, A, B, C\_IN);

Initial

begin

A = 4'd0; B = 4'd0; C\_IN = 1'b0;

#5 A = 4'd3; B = 4'd4;

#5 A = 4'd2; B = 4'd5;

#5 A = 4'd9; B = 4'd9;

#5 A = 4'd10; B = 4'd15;

#5 A = 4'd10; B = 4'd5; C\_IN = 1'b1;

end

initial

begin

$monitor($time," A= %b, B=%b, C\_IN= %b, --- C\_OUT= %b, SUM= %b\n",A, B, C\_IN, C\_OUT, SUM);

end

endmodule

1. 4 TO 1 MUX:
   1. DIAGRAM:

* 1. CODE:

module mux4\_to\_1 (out, i0, i1, i2, i3, s1, s0);

output out;

input i0, i1, i2, i3;

input s1, s0;

wire s1n, s0n;

wire y0, y1, y2, y3;

not (s1n, s1);

not (s0n, s0);

and (y0, i0, s1n, s0n);

and (y1, i1, s1n, s0);

and (y2, i2, s1, s0n);

and (y3, i3, s1, s0);

or (out, y0, y1, y2, y3);

endmodule

* 1. TEST BENCH:

Mod

module stimulus;

reg IN0, IN1, IN2, IN3;

reg S1, S0;

wire OUTPUT;

mux4\_to\_1 mymux(OUTPUT, IN0, IN1, IN2, IN3, S1, S0);

initial

begin

IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;

#5 S1 = 0; S0 = 0;

#5 S1 = 0; S0 = 1;

#5 S1 = 1; S0 = 0;

#5 S1 = 1; S0 = 1;

end

endmodule

1. D FLIP FLOP:
   1. DIAGRAM:
   2. CODE:

module D-FF(q, d, clk, reset) ;

output q;

input d, clk, reset;

reg q;

always @(posedge reset or negedge clk)

if (reset)

q = 1'b0;

**else**

**q** = *d;*

endmodule

* 1. TEST BENCH:

1. T FLIP FLOP:
   1. DIAGRAM:
   2. CODE:

module T-FF(q, clk, reset) ;

output q;

input clk, reset;

wire d;

D-FF dff0 (q, d, clk, reset) ;

not nl(d, q);

endmodule

* 1. TEST BENCH:

1. 4 BIT RIPPLE CARRY COUNTER:
   1. DIAGRAM:
   2. CODE:

module ripple-carry-counter(q, clk, reset);

output [3:0] q;

input clk, reset;

T-FF tffO(q[0],clk,reset);

T-FF tff1(q[1] ,q[0], reset);

T-FF tff2 (q[2] ,q[1], reset) ;

T-FF tff3(q[3] ,q[2], reset) ;

endmodule

* 1. TEST BENCH:

module stimulus;

reg clk;

reg reset;

wire[3:0] q;

// instantiate the design block

ripple-carry-counter rl(q, clk, reset);

// Control the clk signal that drives the design block.Cycle time = 10ns

initial

clk = 1'b0; //set clk to 0

always

#5 clk = -clk; //toggle clk every 5 time units

// Control the reset signal that drives the design block

initial

begin

reset = 1'b1;

#15 reset = 1'b0;

#180 reset = 1'b1;

#10 reset = 1'b0;

#20 $finish; //terminate the simulation

end

// Monitor the outputs

initial

$monitor ($time, " Output q = %d" , q) ;

endmodule

1. SR LATCH:
   1. DIAGRAM:
   2. CODE:

// Module name and port list

// SR\_latch module

module SR\_latch(Q, Qbar, Sbar, Rbar);

//Port declarations

output Q, Qbar;

input Sbar, Rbar;

// Instantiate lower-level modules

// In this case, instantiate Verilog primitive nand gates

// Note, how the wires are connected in a cross-coupled fashion.

nand n1(Q, Sbar, Qbar);

nand n2(Qbar, Rbar, Q);

// endmodule statement

endmodule

* 1. TEST BENCH:

// Stimulus module

// Module name and port list is not present

module Top;

// Declarations of wire, reg, and other variables

wire q, qbar;

reg set, reset;

// Instantiate lower-level modules

// In this case, instantiate SR\_latch

// Feed inverted set and reset signals to the SR latch

SR\_latch m1(q, qbar, ~set, ~reset);

BEHAVIORAL MODELING

1. 4 TO 1 MUX:
   1. Verilog code:

module mux4\_to\_1 (out, i0, i1, i2, i3, s1, s0);

// Port declarations from the I/O diagram

output out;

input i0, i1, i2, i3;

input s1, s0;

reg out;

always @(s1 or s0 or i0 or i1 or i2 or i3)

case ({s1, s0}) //Switch based on concatenation of control signals

2’b00 : out = i0;

2’b01 : out = i1;

2’b10 : out = i2;

2’d11 : out = i3;

default: $display("Invalid control signals");

endcase

endmodule

1. 4-bit Ripple Carry Counter:
   1. Verilog code:

module stimulus;

// Set up variables

reg [3:0] A, B;

reg C\_IN;

wire [3:0] SUM;

wire C\_OUT;

fulladd4 FA1\_4(SUM, C\_OUT, A, B, C\_IN);

Initial

begin

A = 4'd0; B = 4'd0; C\_IN = 1'b0;

#5 A = 4'd3; B = 4'd4;

#5 A = 4'd2; B = 4'd5;

#5 A = 4'd9; B = 4'd9;

#5 A = 4'd10; B = 4'd15;

#5 A = 4'd10; B = 4'd5; C\_IN = 1'b1;

end

initial

begin

$monitor($time," A= %b, B=%b, C\_IN= %b, --- C\_OUT= %b, SUM= %b\n",A, B, C\_IN, C\_OUT, SUM);

end

endmodule