Verilog Codes for Study

1 Logic Gates

1.1 AND Gate

1.1.1 Question

Design an AND gate.

1.1.2 Verilog Code

```
module andgate (a, b, y);
input a, b;
output y;
assign y = a & b;
endmodule
```

1.1.3 Test Bench

```
module andgate_tb;
wire t_y;
reg t_a, t_b;
andgate my_gate( .a(t_a), .b(t_b), .y(t_y) );
initial
begin
$monitor("A=%b, B=%b ---> Y=%b", t_a, t_b, t_y);
t_a = 1'b0;
t_b = 1'b0;
#5
t_a = 1'b0;
t_b = 1'b1;
#5
t_a = 1'b1;
t_b = 1'b0;
#5
t_a = 1'b1;
t_b = 1'b1;
end
endmodule
```

1.2 OR Gate

1.2.1 Question

Design an OR gate.

1.2.2 Verilog Code

```
module orgate(input a, input b, output y);
assign y = a | b;
endmodule
```

1.2.3 Test Bench

```
module orgate_tb;
wire t_y;
reg tb_a, tb_b;
orgate mygate( .a(tb_a), .b(tb_b), .y(t_y) );
initial
begin
$monitor("A=%b, B=%b ---> Y=%b", tb_a, tb_b, t_y);
tb_a = 1'b0;
tb_b = 1'b0;
#5
tb_a = 1'b0;
tb_b = 1'b1;
#5
tb_a = 1'b1;
tb_b = 1'b0;
#5
tb_a = 1'b1;
tb_b = 1'b1;
end
endmodule
```

1.3 NOT Gate

1.3.1 Question

Design a NOT gate.

1.3.2 Verilog Code

```
module not_gate(
input a,
output z);
assign z = ~a;
endmodule
```

1.3.3 Test Bench

```
module notgate_tb;
wire tb_z;
reg tb_a;
not_gate mygate( .a(tb_a), .z(tb_z));
initial
begin
$monitor("A=%b ---> Z=%b", tb_a, tb_z);
tb_a = 1'b0;
#5
tb_a = 1'b1;
end
endmodule
```

1.4 NOR Gate

1.4.1 Question

Design a NOR gate.

1.4.2 Verilog Code

```
module nor_gate(
input a,
input b,
output q);
assign q = ~(a | b);
endmodule
```

1.4.3 Test Bench

```
module nor_tb;
wire tb_q;
reg tb_a,tb_b;
nor_gate mygate( .a(tb_a), .b(tb_b), .q(tb_q));
initial
begin
$monitor("A=%b, B=%b ---> Q=%b", tb_a, tb_b, tb_q);
tb_a = 1'b0;
tb_b = 1'b0;
#5
tb_a = 1'b0;
tb_b = 1'b1;
#5
tb_a = 1'b1;
tb_b = 1'b0;
#5
tb_a = 1'b1;
tb_b = 1'b1;
end
endmodule
```

1.5 NAND Gate

1.5.1 Question

Design a NAND gate.

1.5.2 Verilog Code

```
module nand_gate(
input a,
input b,
output y );
assign y = ~(a & b);
endmodule
```

1.5.3 Test Bench

```
module nand_tb;
wire tb_y;
reg tb_a, tb_b;
nand_gate mygate (a(tb_a), .b(tb_b), .y(tb_y));
initial
begin
$monitor("A=%b, B=%b ---> Y=%b", tb_a, tb_b, tb_y);
tb_a = 1'b0;
tb_b = 1'b0;
#5
tb_a = 1'b0;
```

```
tb_b = 1'b1;
#5
tb_a = 1'b1;
tb_b = 1'b0;
#5
tb_a = 1'b1;
tb_b = 1'b1;
end
endmodule
```

1.6 XOR Gate

1.6.1 Question

Design a XOR gate.

1.6.2 Verilog Code

```
module xor_gate(
input a,
input b,
output y );
assign y = a ^ b;
endmodule
```

1.6.3 Test Bench

```
module xor_tb;
wire tb_y;
reg tb_a, tb_b;
xor_gate mygate (a(tb_a), b(tb_b), .y(tb_y));
initial
begin
$monitor("A=%b, B=%b ---> Y=%b", tb_a, tb_b, tb_y);
tb_a = 1'b0;
tb_b = 1'b0;
#5
tb_a = 1'b0;
tb_b = 1'b1;
#5
tb_a = 1'b1;
tb_b = 1'b0;
#5
tb_a = 1'b1;
tb_b = 1'b1;
end
endmodule
```

1.7 XNOR Gate

1.7.1 Question

Design an XNOR gate.

1.7.2 Verilog Code

```
module xnor_gate(
input a,
input b,
output y );
```

```
assign y = ~(a ^ b);
endmodule
```

1.7.3 Test Bench

```
module xnor_tb;
wire tb_y;
reg tb_a, tb_b;
xnor_gate mygate (a(tb_a), b(tb_b), .y(tb_y));
initial
begin
$monitor("A=%b, B=%b ---> Y=%b", tb_a, tb_b, tb_y);
tb_a = 1'b0;
tb_b = 1'b0;
#5
tb_a = 1'b0;
tb_b = 1'b1;
#5
tb_a = 1'b1;
tb_b = 1'b0;
#5
tb_a = 1'b1;
tb_b = 1'b1;
end
endmodule
```

2 Arithmetic Circuits

2.1 Subtractor Circuit

2.1.1 Question

Design a subtractor circuit.

2.1.2 Verilog Code

```
module subtractor (X,Y,Z, borrow);
input[15:0] X,Y;
output[15:0] Z;
output borrow;
assign {borrow, Z} = X - Y;
endmodule
```

2.1.3 Test Bench

```
module subtest;
reg[15:0] X,Y;
wire[15:0] Z;
wire B;
subtractor ADDER(X,Y,Z,B);
initial
begin
$dumpfile("adder.vcd");
$dumpvars(0,subtest);
$monitor($time, "X=%h, Y=%h, Z=%h, Bo=%b",X,Y,Z,B);
#5 X=16'hfff0; Y=16'h8000;
#5 X=16'h0000; Y=16'hfed1;
#5 X=16'hAAAA; Y=16'h55555;
#5 $finish;
```

```
end
endmodule
```

2.2 Full Subtractor Circuit

2.2.1 Question

Design a full subtractor circuit.

2.2.2 Verilog Code

```
module fullsub(X,Y,Z, Sub, Borrow);
input X,Y,Z;
output Sub;
output Borrow;
assign Sub = X ^ Y ^ Z;
assign Borrow = ~X*Y + ~(X^Y)*Z;
endmodule
```

2.2.3 Test Bench

```
module subtest;
reg X,Y,Z;
wire D;
wire B;
fullsub Subtractor (X,Y,Z,D,B);
initial
begin
$dumpfile("subtest.vcd");
$dumpvars(0,subtest);
$monitor("X=%b Y=%b Z=%b ---> D=%b and B=%b", X,Y,Z,D,B);
X=1'b0;
Y=1'b0;
Z=1'b0;
#5
X=1'b0;
Y=1'b0;
Z=1'b1;
#5
X=1'b0;
Y=1'b1;
Z=1'b0;
#5
X=1'b0;
Y=1'b1;
Z=1'b1;
#5
X=1'b1;
Y=1'b0;
Z=1'b0;
#5
X=1'b1;
Y=1'b0;
Z=1'b1;
#5
X=1'b1;
Y=1'b1;
Z=1'b0;
#5
```

```
X=1'b1;
Y=1'b1;
Z=1'b1;
$finish;
end
endmodule
```

2.3 Adder Circuit

2.3.1 Question

Design an adder circuit.

2.3.2 Verilog Code

```
module adder(X,Y,Z, Carry);
input[15:0] X,Y;
output[15:0] Z;
output Carry;
assign {Carry,Z} = X + Y;
endmodule
```

2.3.3 Test Bench

```
module addtest;
reg[15:0] X,Y;
wire[15:0] Z;
wire CY;
adder ADDER (X, Y, Z, CY);
initial
begin
$dumpfile("adder.vcd");
$dumpvars(0,addtest);
$monitor($time, "X=%h, Y=%h, Z=%h, CY=%b",X,Y,Z,CY);
#5
X=16'h8fff;
Y=16'h8000;
#5
X=16'hfffe;
Y=16'h0000;
#5
X=16'hAAAA;
Y=16'h5555;
#5 $finish;
end
endmodule
```

2.4 Full Adder Circuit

2.4.1 Question

Design a full adder circuit.

2.4.2 Verilog Code

```
module fulladder (X,Y,Z, Sum, Carry);
input X,Y,Z;
output Sum;
output Carry;
assign Sum = X ^ Y ^ Z;
```

```
assign Carry = X*Y + (X^Y)*Z;
endmodule
```

2.4.3 Test Bench

```
module addtest;
reg X,Y,Z;
wire S;
wire C;
fulladder ADDER(X,Y,Z,S,C);
initial
begin
$dumpfile("adder.vcd");
$dumpvars(0,addtest);
$monitor("X=%b Y=%b Z=%b ----> S=%b and C=%b",X,Y,Z,S,C);
X=1'b0;
Y=1'b0;
Z=1'b0;
#5
X=1'b0;
Y=1'b0;
Z=1'b1;
#5
X=1'b0;
Y=1'b1;
Z=1'b0;
#5
X=1'b0;
Y=1'b1;
Z=1'b1;
#5
X=1'b1;
Y=1'b0;
Z=1'b0;
#5
X=1'b1;
Y=1'b0;
Z=1'b1;
#5
X=1'b1;
Y=1'b1;
Z=1'b0;
#5
X=1'b1;
Y=1'b1;
Z=1'b1;
$finish;
end
endmodule
```

2.5 Ripple Carry Adder

2.5.1 Question

Design a ripple carry adder.

2.5.2 Verilog Code

```
module addblock(X,Y,Z, Sum, Carry);
```

```
input X,Y,Z;
output Sum;
output Carry;
assign Sum = X ^ Y ^ Z;
assign Carry = X*Y + (X^Y)*Z;
endmodule

module ripple(a,b,cin,s,cout);
input [3:0] a,b;
input cin;
output [3:0] s;
output [3:0] cout;
addblock Adder (a[0],b[0], cin,s[0],cout[0]);
addblock Adder1(a[1],b[1], cout[0], s[1], cout[1]);
addblock Adder2(a[2],b[2], cout[1], s[2],cout[2]);
addblock Adder3(a[3], b[3], cout[2], s[3],cout[3]);
endmodule
```

2.5.3 Test Bench

```
module ripple_tb;
reg [3:0] A,B;
reg CIN;
wire [3:0] S;
wire [3:0] COUT;
ripple rippleadd(A, B, CIN, S, COUT);
initial
begin
$monitor("A=%b B=%h CIN=%b -----> S=%b and COUT=%b", A,B,CIN,S,COUT[3]);
CIN = 0;
A = 4'b1010;
B = 4'b1100;
#5
A = 4'b1111;
B = 4'b1100;
#5
A = 4'b1011;
B = 4'b0100;
#5
A = 4'b1111;
B = 4'b0001;
end
endmodule
```

2.6 Carry Look-ahead Adder

2.6.1 Question

Design a carry look-ahead adder.

2.6.2 Verilog Code

```
module CLA (a,b,cin, S,cout);
input [3:0] a, b;
input cin;
output [3:0] S;
output cout;
output [3:0] c;
assign c[0] = cin;
```

```
assign c[1] = (a[0] & b[0]) | ((a[0]^b[0]) & c[0]);
assign c[2] = (a[1] & b[1]) | ((a[1]^b[1]) & c[1]);
assign c[3] = (a[2] & b[2]) | ((a[2]^b[2]) & c[2]);
assign cout = (a[3] & b[3]) | ((a[3]^b[3]) & c[3]);
assign S = a^b^c;
endmodule
```

2.6.3 Test Bench

```
module CLA_tb;
reg [3:0] A, B;
reg Cin;
wire [3:0] S;
wire Cout;
CLA cla_adder(A, B, Cin, S, Cout);
initial
begin
$monitor("A=%b, B=%b, Cin = %b --> S = %b, Cout =%b", A, B, Cin, S, Cout);
A=4'b1010;
B=4'b1100;
Cin = 0;
#5
A=4'b1111;
B=4'b1100;
Cin = 1;
#5
A=4'b1011;
B=4'b0100;
Cin = 0;
#5
A=4'b1111;
B=4'b0001;
Cin = 1;
end
endmodule
```

3 Multiplexer

3.1 16 to 1 Multiplexer

3.1.1 Question

Design a 16 to 1 Multiplexer.

3.1.2 Verilog Code

```
module mux16to1(in, sel, out);
input[15:0] in;
input[3:0] sel;
output out;
assign out=in[sel];
endmodule
```

3.1.3 Test Bench

```
module muxtest;
reg[15:0] A; reg[3:0] S; wire F;
mux16to1 M(.in(A),.sel(S),.out(F));
initial
```

```
begin
$dumpfile("mux16to1.vcd");
$dumpvars(0,muxtest);
$monitor($time, "A=%h S=%h, F=%b",A,S,F);
#5 A=16'h3f0a;
S=4'h0;
#5 S=4'h1;
#5 S=4'h2;
#5 S=4'h3;
#5 S=4'h4;
#5
S=4'h5;
#5
S=4'h6;
#5 S=4'h7;
#5 S=4'h8;
#5 S=4'h9;
#5 S=4'ha;
#5 S=4'hb;
#5 S=4'hc;
#5 S=4'hd;
#5 S=4'he;
#5 S=4'hf;
#5 $finish;
end
endmodule
```