# 数字电路实验二报告

编程题：

1. 2选1多路选择器

模块源码：

module mux2to1(w0,w1,s,f);

input w0,w1,s;

output f;

assign f=s?w1:w0;

endmodule

测试源码：

`timescale 1ns/1ps

module tb\_mux2to1;

reg w0\_test;

reg w1\_test;

reg s\_test;

wire f\_test;

initial

begin

w0\_test=0;

w1\_test=0;

s\_test=0;

end

always #80 s\_test=~s\_test;

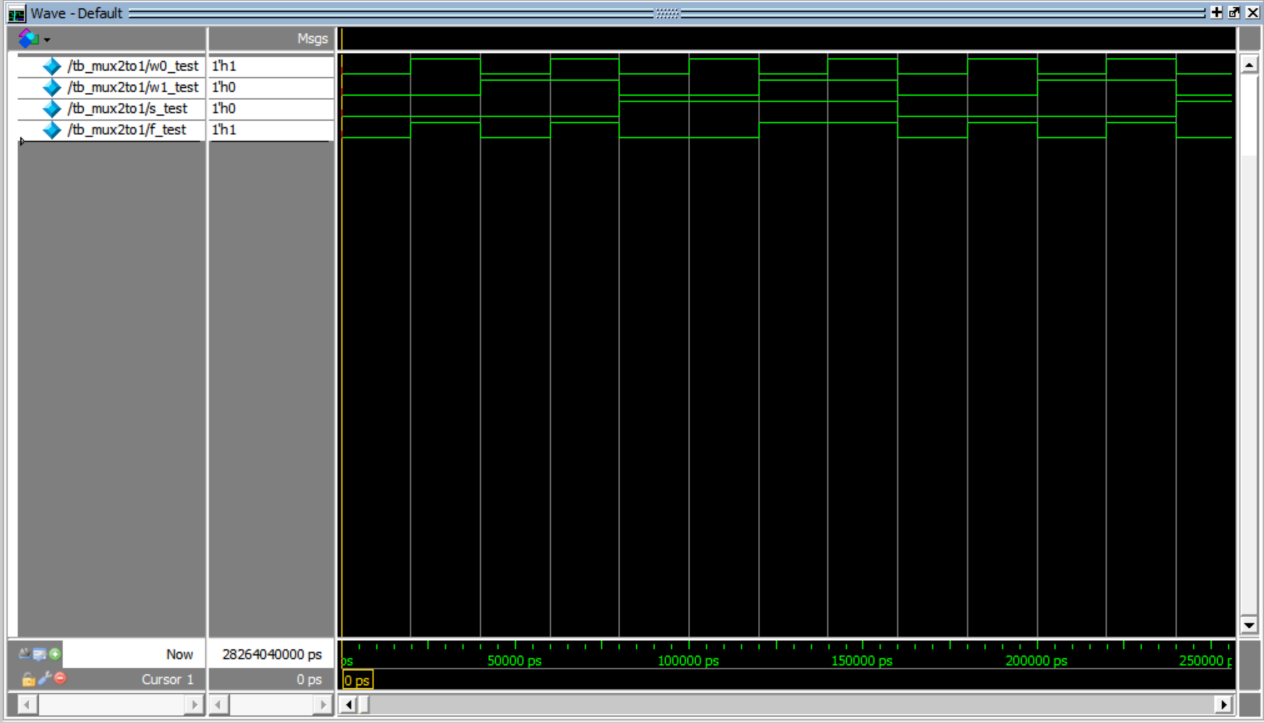
always #20 w0\_test=~w0\_test;

always #40 w1\_test=~w1\_test;

mux2to1 UUT\_mux2to1(.w0(w0\_test),.w1(w1\_test),.s(s\_test),.f(f\_test));

endmodule

仿真波形：



波形分析：

0~80ns内，s\_test=0,f\_test的波形跟w0\_test的波形一样，说明f\_test=w0\_test;

80~160ns内，s\_test=1,f\_test的波形跟w1\_test的波形一样，说明f\_test=w1\_test;

也即，当s=0时，f=w0；当s=1时，f=w1。s作为选择信号，完成了2选1的功能。

1. 2-4译码器

模块源码：

module dec2to4(W,En,Y);

input [1:0]W;

input En;

output reg[0:3]Y;

always @(W,En)

case({En,W})

3'b100:Y=4'b1000;

3'b101:Y=4'b0100;

3'b110:Y=4'b0010;

3'b111:Y=4'b0001;

default:Y=4'b0000;

endcase

endmodule

测试源码：

`timescale 1ns/1ps

module tb\_dec2to4;

reg [1:0]W\_test;

reg En\_test;

wire [0:3]Y\_test;

initial

begin

En\_test=0;

#40 En\_test=1;

end

initial

begin

W\_test=2'b00;

#20

W\_test=2'b01;

#20

W\_test=2'b11;

#20

W\_test=2'b10;

#20

W\_test=2'b00;

#20

W\_test=2'b01;

#20

W\_test=2'b11;

#20

W\_test=2'b10;

#20

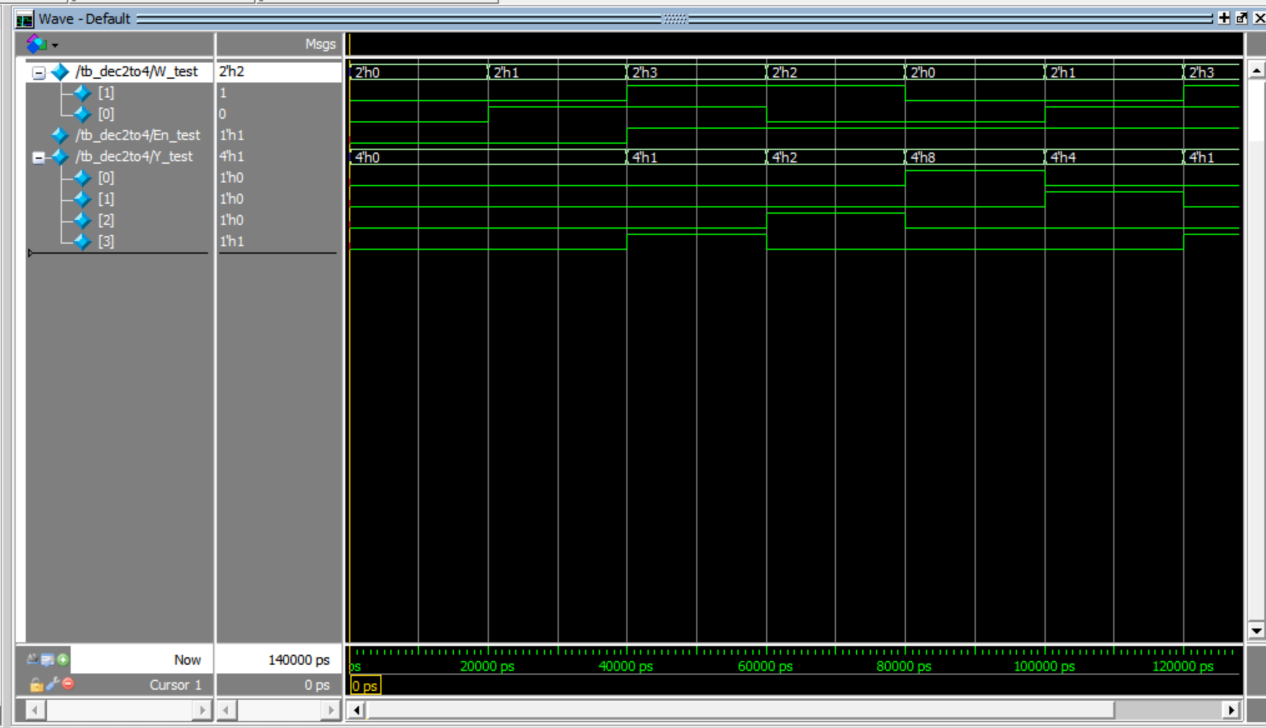
$finish;

end

dec2to4 UUT\_dec2to4(.W(W\_test),.En(En\_test),.Y(Y\_test));

endmodule

仿真波形：



波形分析：

0~40ns内，E=0，此时不论W\_test如何变化，Y\_test=0；

40ns后，E=1,

40~60ns内，W\_test=3，Y\_test=1=4’b0001；

60~80ns内，W\_test=2，Y\_test=2=4’b0010；

80~100ns内，W\_test=0，Y\_test=8=4’b1000；

100~120ns内，W\_test=1，Y\_test=4=4’b0100；

1. 8-3编码器

模块源码：

module priority(W,Y,z);

input [3:0] W;

output reg [1:0] Y;

output reg z;

always @(W)

begin

z=1;

casex(W)

4'b1xxx:Y=3;

4'b01xx:Y=2;

4'b001x:Y=1;

4'b0001:Y=0;

default:

begin

z=0;

Y=2'bx;

end

endcase

end

endmodule

测试源码：

`timescale 1ns/1ps

module tb\_priority;

reg [3:0] W\_test;

wire [1:0] Y\_test;

wire z\_test;

initial

begin

W\_test=4'b1000;

#40 W\_test=4'b0100;

#40 W\_test=4'b0010;

#40 W\_test=4'b0001;

#40 W\_test=4'b0000;

#40 $finish;

end

priority UUT\_priority(.W(W\_test),.Y(Y\_test),.z(z\_test));

endmodule

仿真波形：



波形分析：

0~40ns内，W\_test=8=4’b1000，此时Y\_test=3，z=1；

40~80ns内，W\_test=4=4’b0100，此时Y\_test=2，z=1；

80~120ns内，W\_test=2=4’b0010，此时Y\_test=1，z=1；

120~160ns内，W\_test=1=4’b0001，此时Y\_test=0，z=1；

160~200ns内，W\_test=0=4’b0000，此时Y\_test的值不确定，z=0；

也即当W不是独热码时，z=0，说明输出无效，否则z=1，Y=对应的编码。

1. 1位二进制比较器

模块源码：

module compare(A,B,AeqB,AgtB,AltB);

input A,B;

output reg AeqB,AgtB,AltB;

always @(A,B)

begin

AeqB=0;

AgtB=0;

AltB=0;

if(A==B)

AeqB=1;

else if(A>B)

AgtB=1;

else

AltB=1;

end

endmodule

测试源码：

`timescale 1ns/1ps

module tb\_compare;

reg A\_test;

reg B\_test;

wire AeqB\_test,AgtB\_test,AltB\_test;

initial

A\_test=0;

always #20 A\_test=~A\_test;

initial

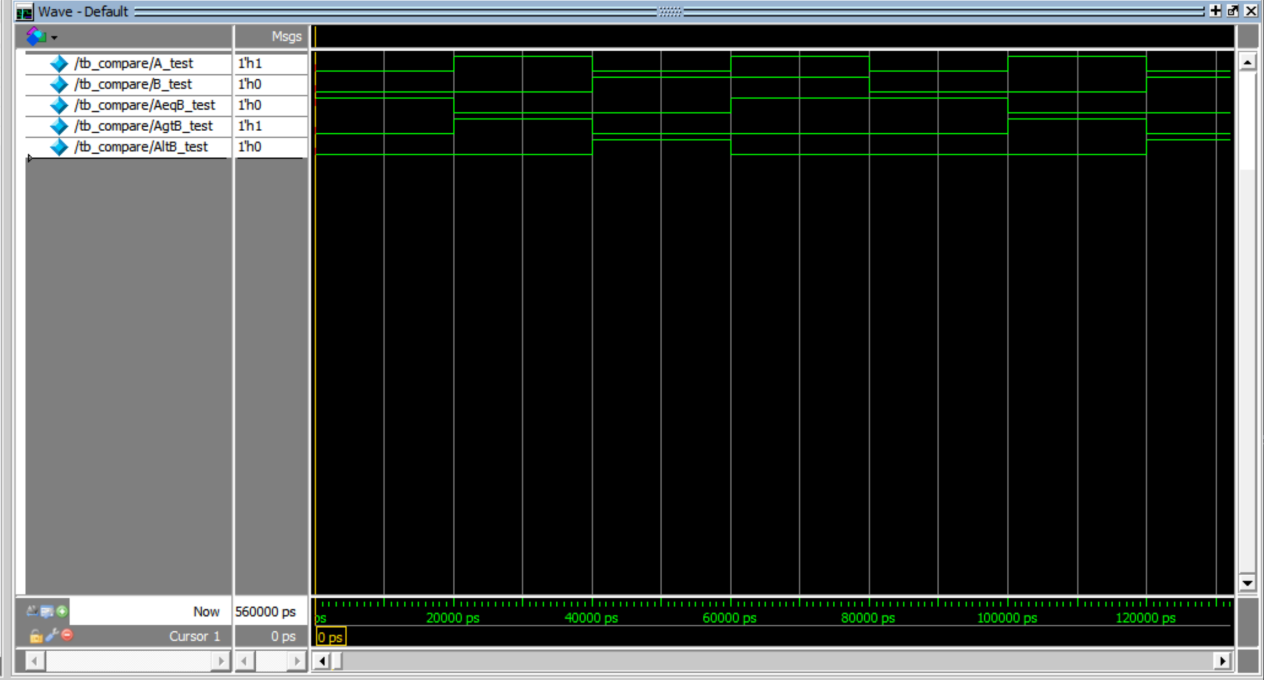
B\_test=0;

always #40 B\_test=~B\_test;

compare UUT\_compare(.A(A\_test),.B(B\_test),.AeqB(AeqB\_test),.AgtB(AgtB\_test),.AltB(AltB\_test));

endmodule

仿真波形：



波形分析：

A\_test每20ns变化一次,B\_test每40ns变化一次。

0~20ns内，A\_test=0，B\_test=0，此时AeqB\_test=1，AgtB\_test=0，AltB\_test=0；

20~40ns内，A\_test=1，B\_test=0，此时AeqB\_test=0，AgtB\_test=1，AltB\_test=0；

40~60ns内，A\_test=0，B\_test=1，此时AeqB\_test=0，AgtB\_test=0，AltB\_test=1；

60~80ns内，A\_test=1，B\_test=1，此时AeqB\_test=1，AgtB\_test=0，AltB\_test=0；

也即当A=B时，AeqB=1，说明二者相等；当A=1,B=0时，AgtB=1，说明A比B大；当A=0，B=1时，AltB=1，说明A比B小。

1. 2+2位简单全加器

模块源码：

module adder2(carryin,X,Y,S,carryout);

    input carryin;

    input [1:0]X,Y;

    output [1:0]S;

    output carryout;

        wire C1;

    fulladd stage0(carryin,X[0],Y[0],S[0],C1);

    fulladd stage1(C1,X[1],Y[1],S[1],carryout);

endmodule

module fulladd(Cin,x,y,s,Cout);

    input Cin,x,y;

    output s,Cout;

    assign  s=x^y^Cin,

            Cout=(x&y)|(x&Cin)|(y&Cin);

endmodule

测试源码：

`timescale 1ns/1ns

module tb\_adder2;

    reg carryin\_test;

    reg [1:0]X\_test;

    reg [1:0]Y\_test;

    wire [1:0]S\_test;

    wire carryout\_test;

    initial

    begin

    carryin\_test=1;

    #20 carryin\_test=0;

    end

    initial

    begin

        X\_test=2'b00;

        Y\_test=2'b00;

        #40

        X\_test=2'b01;

        Y\_test=2'b00;

        #40

        X\_test=2'b01;

        Y\_test=2'b01;

        #40

        X\_test=2'b10;

        Y\_test=2'b01;

        #40

        X\_test=2'b10;

        Y\_test=2'b10;

        #40

        X\_test=2'b10;

        Y\_test=2'b11;

        #40

        X\_test=2'b11;

        Y\_test=2'b11;

        #40

        $finish;

    end

    adder2 UUT\_adder2(.carryin(carryin\_test),.X(X\_test),.Y(Y\_test),.S(S\_test),.carryout(carryout\_test));

endmodule

    adder2 UUT\_adder2(.carryin(carryin\_test),.X(X\_test),.Y(Y\_test),.S(S\_test),.carryout(carryout\_test));

endmodule

仿真波形：



波形分析：

0~20ns内，carryin\_test=1，X\_test=0，Y\_test=0，此时S\_test=1，carryout\_test=0；

20ns后，carryin\_test=0，

20~40内，X\_test=0，Y\_test=0，此时S\_test=0，carryout\_test=0；

40~80内，X\_test=1，Y\_test=0，此时S\_test=1，carryout\_test=0；

80~120内，X\_test=1，Y\_test=1，此时S\_test=2，carryout\_test=0；

120~160内，X\_test=2，Y\_test=1，此时S\_test=3，carryout\_test=0；

160~200内，X\_test=2，Y\_test=2，此时S\_test=0，carryout\_test=1；

200~240内，X\_test=2，Y\_test=3，此时S\_test=1，carryout\_test=1；

240~280内，X\_test=3，Y\_test=3，此时S\_test=2，carryout\_test=1；

也即实现了2+2位简单全加器的功能。

verilog编程与高级语言编程的主要区别：

1. 在verilog模块中所有过程（eg：initial块、always块）、连续赋值语句、实例引用都是并行的；而高级语言编程中基本上指令都是是串行执行的，每一行代码的执行都要等到前一行代码执行完毕后才能进行。
2. 在同一个模块中各个过程块、各条连续赋值语句和各条实例引用语句这三者出现的先后顺序没有关系；而高级语言编程中会影响代码的执行顺序。
3. 只有连续赋值语句（即用关键词assign引出的语句）和实例引用语句（即用已定义的模块名引出的语句），可以独立于过程块存在而在模块的功能定义部分。
4. 被实例引用的模块，其端口可以通过不同的连线或寄存器类型变量连接到别的模块相应的输出输入信号端，这与高级语言编程中的函数调用类似，但是高级语言编程中有返回值，而且调用函数不会改变原参数的值（指针传递除外）。
5. 时序分析：verilog编程中需要考虑信号在电路中传输时的传播延时，如线延时、器件延时。时延就是对延时特性的HDL描述。而高级编程语言中不需要考虑时延。
6. If else语句与case语句：verilog编程中如果使用if语句，最好有else分支；如果使用case语句，最好有default语句。即使需要锁存器，也通过else分支或default分支来显式说明。而c语言中不作特殊要求。

本次实验收获和心得：

本次实验学习了verilog编程规范和面向硬件电路的设计思想，明白了其与高级语言编程的一些区别，体会到了硬件设计并行执行的设计风格，完成了几道测试模块的编写、综合及仿真。懂得了在学习过程中不能一遇到困难就立刻寻求帮助，应该先自己搜索查找，独立解决，实在不行再去向他人寻求意见。