上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

数字逻辑设计

LAB1 – 组合逻辑电路设计

姓名：邱一航

学号：520030910155

1. 8-to-3 Encoder

1.1 实验电路原理

8-3 普通编码器共有 8 个输入 A0~A7，3 个输出 Y0~Y2，输出结果是有效（此处“有效”认为是高电平）输入信号的编码。8-3 普通编码器要求任何时刻有且仅有 1 个输入信号有效（高电平），其余输入信号均需为无效（低电平）。特别地，我们这次要求在无效输入下，输出为默认输出（三个输出全部低电平）。

8-3 普通编码器在合法输入下的真值表如下图：

Inputs								Outputs		
A0	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**Y2**	**Y1**	**Y0**
1	0	0	0	0	0	0	0	**0**	**0**	**0**
0	1	0	0	0	0	0	0	**0**	**0**	**1**
0	0	1	0	0	0	0	0	**0**	**1**	**0**
0	0	0	1	0	0	0	0	**0**	**1**	**1**
0	0	0	0	1	0	0	0	**1**	**0**	**0**
0	0	0	0	0	1	0	0	**1**	**0**	**1**
0	0	0	0	0	0	1	0	**1**	**1**	**0**
0	0	0	0	0	0	0	1	**1**	**1**	**1**

表 1　8-3 普通编码器的真值表

特别地，输入为非法输入（有 2 个及以上输入信号有效，或输入信号全部无效）时，输出应为 **000**。

1.2 电路实现与关键代码讨论

本次实验中，笔者采用了行为级建模的方式。

笔者原本打算用数据流建模，但是有了非法输入默认输出 000 的要求后，数据流建模变得较为复杂，因此最后仍使用行为级建模。

由于 8-3 普通编码器中的各种输入之间并不存在优先级，因此使用 case 语句即可非常简单地实现其功能。

8-3 普通编码器的 Verilog 代码如下：

```
/* ================================[Encoder8_3.v]================================ */
```

```
`timescale 1ns/100ps
module encoder8_3(a,y);

//Input and Output Definition.
input [7:0] a;
output[2:0] y;
wire[7:0] a;
reg [2:0] y;

// 原本想搞一搞数据流建模的（如下），然而多了非法输入默认值0这个要求之后就只好行为级建模用 case 语句了。
// 不能实现默认输出 000 的数据流建模如下。
/*
assign y[0] = a[1]|a[3]|a[5]|a[7];
assign y[1] = a[2]|a[3]|a[6]|a[7];
assign y[2] = a[4]|a[5]|a[6]|a[7];
*/
//所以现在用行为级建模 case 语句了，如下。
always @(a)
begin
  case (a)
    8'b00000010: y=3'b001;
    8'b00000100: y=3'b010;
    8'b00001000: y=3'b011;
    8'b00010000: y=3'b100;
    8'b00100000: y=3'b101;
    8'b01000000: y=3'b110;
    8'b10000000: y=3'b111;
    default:     y=3'b000;
    // including invalid inputs and the case of valid input a=8'b00000001.
  endcase
end

endmodule
```

```
/* ============================END of [Encoder8_3.v]============================ */
```

在 8-3 普通编码器的 testbench 中，我们理应测试所有的合法输入和所有非法输入。一个显然的想法是将所有可能的输入遍历一遍。

测试平台的 Verilog 代码如下：

```
/* ============================[Encoder8_3_tb_full.v]============================ */
```

```
`timescale 1ns/100ps
module encoder8_3_tb_full;
reg [7:0] a_in;
wire[2:0] y_out;
reg [1:0] count;
encoder8_3 ENCODER(
    .a(a_in),
    .y(y_out)
);

initial fork
    a_in = 0;
    count = 0;
join

// A thorough test is as follows.
always #5
begin
    if (a_in==0)       a_in = 1;
    else       a_in = a_in + 1;
//To test valid inputs only, change this line to “a_in = a_in<<1;”
end

endmodule
```

```
/* ========================END of [Encoder8_3_tb_full.v]======================== */
```

由于全部可能输入共2^8种，合法输入仅有 8 个，出现的概率较低；为了更好地确认输入波形是否正确，笔者另写了一版 testbench。该版本的 testbench 每次在测试完所有合法输入后随机生成四个输入来测试。

测试平台的 Verilog 代码如下：

```
/* ================================[Encoder8_3_tb.v]============================ */
`timescale 1ns/100ps
module encoder8_3_tb;
reg [7:0] a_in;
wire[2:0] y_out;
reg [1:0] count;
encoder8_3 ENCODER(
    .a(a_in),
    .y(y_out)
);

initial fork
    a_in = 0;
    count = 0;
join

always #5
// We also test our encoder on invalid inputs.
begin
    if (a_in==0)
    begin
        a_in = {$random}%256;
        // We test four random inputs.
        // The 1st random input.
        count = 1;
    end
    else if ((count>0) && (count<3))
    // We test four random inputs.
    // The 2nd, 3rd, or 4th random input.
    begin
        a_in = {$random}%256;
        count = count+1;
    end
    else if (count==3)
    begin
        a_in = 8'b00000001;
        count = 0;
    end
    else
        a_in = a_in<<1;
end

endmodule

/* ============================END of [Encoder8_3_tb.v]========================== */
```

实际上，使用前一版本的 testbench 已足以证明 8-3 普通编码器的正确性。

（若需要考虑不同延时可能造成的区别，则以不同步长改变输入将全部可能输入遍历一遍即可，即枚举所有相邻信号的可能性。）

此处在 testbench 中引入随机数仅仅是为了使合法输入和非法输入出现的概率不至于过小，这样仿真波形的正确性更易于辨认。

1.3 仿真波形结果讨论

8-3 普通编码器程序在测试平台上的输出仿真波形如下。使用软件为 ModelSim 2020.3。

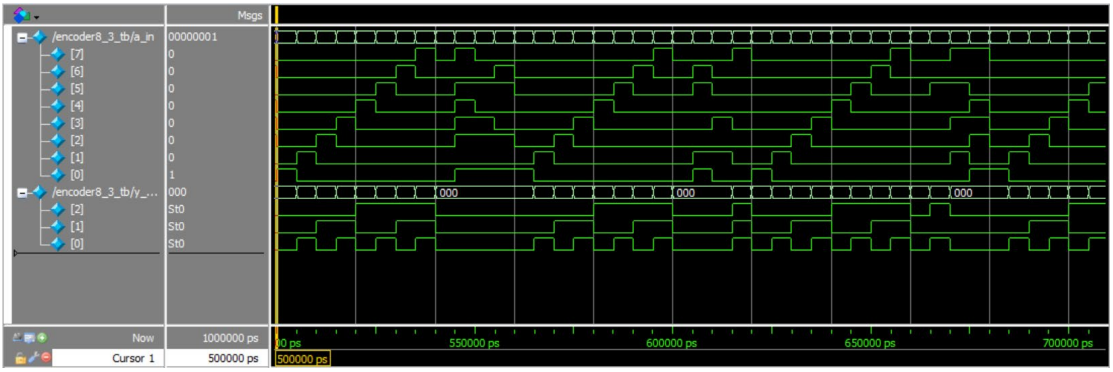


图 1　8-3 普通编码器仿真波形

可以看出，500ns~540ns、560ns~600ns、620~660ns、680~720ns 范围内，输入波形是合法的（即有且仅有一个输入信号为有效信号（高电平））。此区段内，编码器的输出为对应有效输入信号的编码（输出显示为从 000~111 变化）。

在 540ns~560ns、600ns~620ns、660ns~680ns 区段内，输入为随机生成的输入。因此，除了 615ns~620ns、665~670ns 输入波形是合法的（分别只有 A7、A5 有效），对应输出为 111 和 101；其余输入均为非法输入，输出为 000，符合要求。

2. 9-to-4 Priority Encoder

2.1 实验电路原理

由 **1.1** 和 **1.2** 可知，对普通编码器而言，非法输入出现的概率很高，因此要保证编码器正常工作，我们需要对输入信号指定优先级，即使用优先编码器。

9-4 优先编码器共有 9 个输入 A0~A8、4 个输出 Y0~Y3，输出是最优先的有效（高电平）输入信号的编码。其中，A0 到 A8 优先级逐渐递增。

当输入信号有多个有效（高电平）时，编码器只对最优先的有效输入信号进行编码，忽视所有不如该信号优先的输入信号。

9-4 优先编码器的真值表见下页。（其中 x 表示 don’t-care 项，即无关项）

Inputs									Outputs			
A0	A1	A2	A3	A4	A5	A6	A7	A8	Y3	Y2	Y1	Y0
x	x	x	x	x	x	x	x	1	**1**	**0**	**0**	**1**
x	x	x	x	x	x	x	1	0	**1**	**0**	**0**	**0**
x	x	x	x	x	x	1	0	0	**0**	**1**	**1**	**1**
x	x	x	x	x	1	0	0	0	**0**	**1**	**1**	**0**
x	x	x	x	1	0	0	0	0	**0**	**1**	**0**	**1**
x	x	x	1	0	0	0	0	0	**0**	**1**	**0**	**0**
x	x	1	0	0	0	0	0	0	**0**	**0**	**1**	**1**
x	1	0	0	0	0	0	0	0	**0**	**0**	**1**	**0**
1	0	0	0	0	0	0	0	0	**0**	**0**	**0**	**1**
0	0	0	0	0	0	0	0	0	**0**	**0**	**0**	**0**

表 2　9-4 优先编码器的真值表

2.2 电路实现与关键代码讨论

采用行为级建模。

由于有优先级的存在，因此使用 if 语句嵌套的方式实现优先编码器的功能。

9-4 优先编码器的 Verilog 代码如下：

```
/* ================================[Encoder9_4.v]============================== */
    `timescale 1ns/100ps
    module encoder9_4(a,y);

    // Input and output definition.
    input [8:0] a;
    output[3:0] y;
    wire[8:0] a;
    reg [3:0] y;

    always @(a)
    begin
        if (a[8]==1)
            y = 4'b1001;
        else if (a[7]==1)
            y = 4'b1000;
        else if (a[6]==1)
            y = 4'b0111;
        else if (a[5]==1)
            y = 4'b0110;
        else if (a[4]==1)
            y = 4'b0101;
        else if (a[3]==1)
            y = 4'b0100;
        else if (a[2]==1)
            y = 4'b0011;
        else if (a[1]==1)
            y = 4'b0010;
        else if (a[0]==1)
            y = 4'b0001;
        else
            y = 4'b0000;
    end

    endmodule
```

```
/* ============================END of [Encoder9_4.v]============================ */
```

在 9-4 优先编码器的 testbench 中，我们理应测试所有的合法输入和所有非法输入。一个显然的想法是将所有可能的输入遍历一遍。该方案的实现见下方代码中的注释部分。

但很显然，这样会导致不同输出的出现概率不同（输出为 1001 的概率显然最高）。为了使各种不同输出的出现概率相对均衡一些，我们通过以下方式生成输入。

首先生成“标准”的仅有一个输入信号有效的输入，记该有效输入信号为 Ai。则对随机数取 2^i 模可以得到随机的 A0~A(i-1)输入序列。将两者相加即可得到最优先输入为 Ai，且可能有多个有效输入信号的随机输入。

对每组“标准”的输入，我们生成 4 个随机输入用以测试。

9-4 优先编码器 testbench 的 Verilog 代码如下。

```
/* ============================[Encoder9_4_tb.v]============================ */
```

```
`timescale 1ns/100ps
module encoder_9_4_tb;
reg [8:0] a_in, a_gen;
wire[3:0] y_out;
encoder9_4 ENCODER(
    .a(a_in),
    .y(y_out)
);
initial fork
    a_gen = 0;
    a_in  = 0;
join

// To test all possible inputs in order,
use the following block.
/*
always #5
begin
    if (a_in==0)        a_in = 1;
    else        a_in = a_in + 1;
end
*/
always #5
// We also test our encoder on invalid
inputs.
begin
    if (a_gen==0)
    begin
        a_in = a_gen;
        #10 a_gen = 1;
    end
    else
    begin
        a_in = a_gen;
        // We generate three random inputs.
        #5 a_in = a_gen + {$random} % a_gen;
        #5 a_in = a_gen + {$random} % a_gen;
        #5 a_in = a_gen + {$random} % a_gen;
        a_gen = a_gen << 1;
    end
end

endmodule
```

```
/* ============================END of [Encoder9_4_tb.v]============================ */
```

实际上，使用注释中的 testbench 实现即可测试所有可能的输入，足以证明 9-4 优先编码器功能实现的正确性。使用现有版本仅仅是为了使各种输出出现的可能性相对均衡，更易于辨认仿真波形的正确性。

2.3 仿真波形结果讨论

9-4 优先编码器程序在测试平台上的输出仿真波形如下。使用软件为 ModelSim 2020.3。

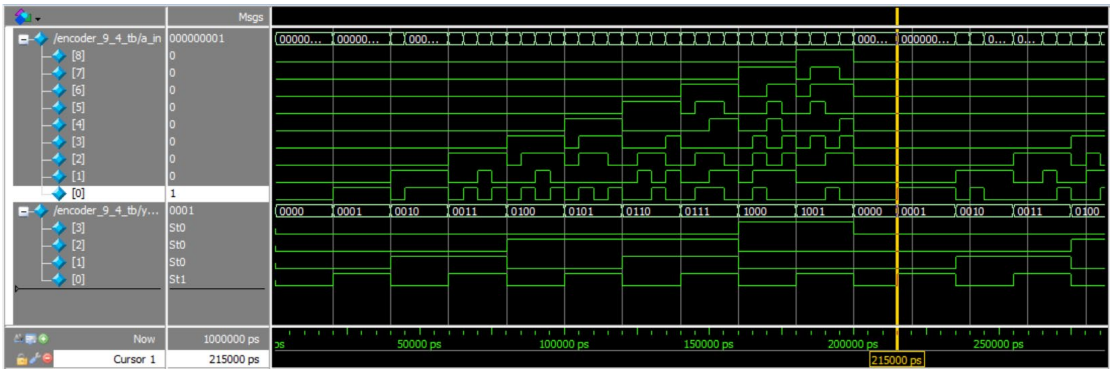
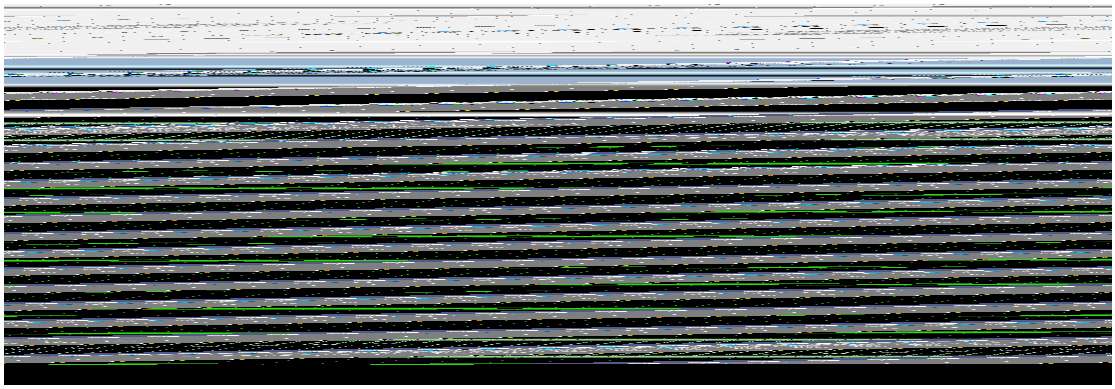


图 2　9-4 优先编码器仿真波形

由图可见，在 0~200ns 区段内，依次测试了全无效输入、最优先有效位为 A0~A8 的情况，输出均表现正常。从 200ns 开始再次依次测试全无效输入、最优先有效位为 A0~A8 的情况，输出结果符合要求。

三、实验中的问题与建议

实验中基本上没有遇到什么比较麻烦的问题。不过，ModelSim 的图片导出功能疑似有问题，如下图。因此最后被迫用截图替代。



建议：

1）喜欢腾讯会议 1 对 1 小组讨论的实验课答疑形式，希望能保留；

2）作业提交的说明文字似乎有歧义（无法确定需要提交的只是实验报告还是包含实验报告、源代码和生成的波形图的压缩文件）。

四、Reference

对 testbench 的一些改进（引入随机数）的灵感来源来自**季弋琨**。