嵌入式系统结构与设计课程实验

实验报告 2

Lab2、Lab3、Lab4 及 Lab6 四次实验的实验报告

邱迪聪 / 11331262

2013/10/16

目录

一、实验目的

- 熟悉使用 ISE 软件设计并仿真；
- 学会程序下载。

二、Lab2 (二位比较器的设计与实现)

1、使用 Verilog 语句表达二位比较器的逻辑

实验题目已经给出了二位比较器的真值表，如下：

b[1]	b[0]	a[1]	d[0]	a_eq_b	a_gt_b	a_lt_b
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	0
1	0	1	1	0	1	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	1	0	0

我们可以直接使用这个真值表，通过组合逻辑语句来描述整个二位比较器的逻辑。可是这样的做法非常耗费精力，而且浪费时间。我相信 Verilog 语言能够被大家广泛地使用，必然提供了针对比较等逻辑的非常简便的语句描述，而且网上必然会有相关的介绍。因此我就上网查找了一下相关的内容，以及 Verilog 语言的相关说明文档。

经过查找，我发现了 Verilog 有表达比较关系的逻辑语句，且使用方法与 C 语言非常相似。故我把比较器的逻辑设计为如下 Verilog 语句：

```
assign a_eq_b = a == b;
assign a_gt_b = a > b;
assign a_lt_b = a < b;
```

其中 a 和 b 均为二位数组，a_eq_b 表示相等输出，a_gt_b 表示大于关系的输出，a_lt_b 表示小于关系的输出。然后再将以上语句放入相应的模块即可。

到此为止，二位比较器的运行逻辑已经编写完成。

2、二位比较器的模拟运行

为了模拟测试编写的代码是否正确，下面需要编写相应的测试程序。

利用程序自动生成测试框架。为了进行完整性测试，需要对每个可能的输入组合都进行一次测试并观察输出是否符合设想。为了简化程序的编写，我在询问 TA 并在网上查阅相关帮助文档后，得知了 Verilog 语言的循环语句 always 语句。

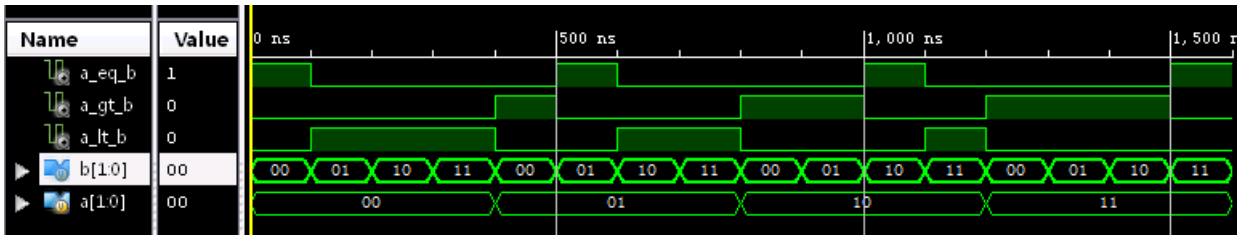
在生成的框架的初始化部分，加入如下初始化语句：

```
b = 2'b 00;
a = 2'b 11;
```

目的是在编写循环语句后，进行运行时图像看起来是从 b=00，a=00 的时刻开始的。

为了进行完整性测试，我设定让 b 每 100ns 进行加一操作，而在 b 的高位进行下降沿跳变时，对 a 进行加一操作。整个测试过程进行 1600ns 则可进行一次完整性测试。

运行模拟器后显示的图像如下：



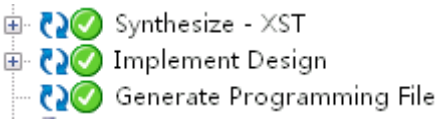
模拟结果与设想符合，故测试通过。

3、综合、实现及程序生成

编写管脚约束，如下：

```
NET "b[1]" LOC = "T5";
NET "b[0]" LOC = "V8";
NET "a[1]" LOC = "U8";
NET "a[0]" LOC = "N8";
NET "a_eq_b" LOC = "T11";
NET "a_gt_b" LOC = "R11";
NET "a_lt_b" LOC = "N11";
```

然后依次运行综合、实现及代码生成程序。运行均成功：



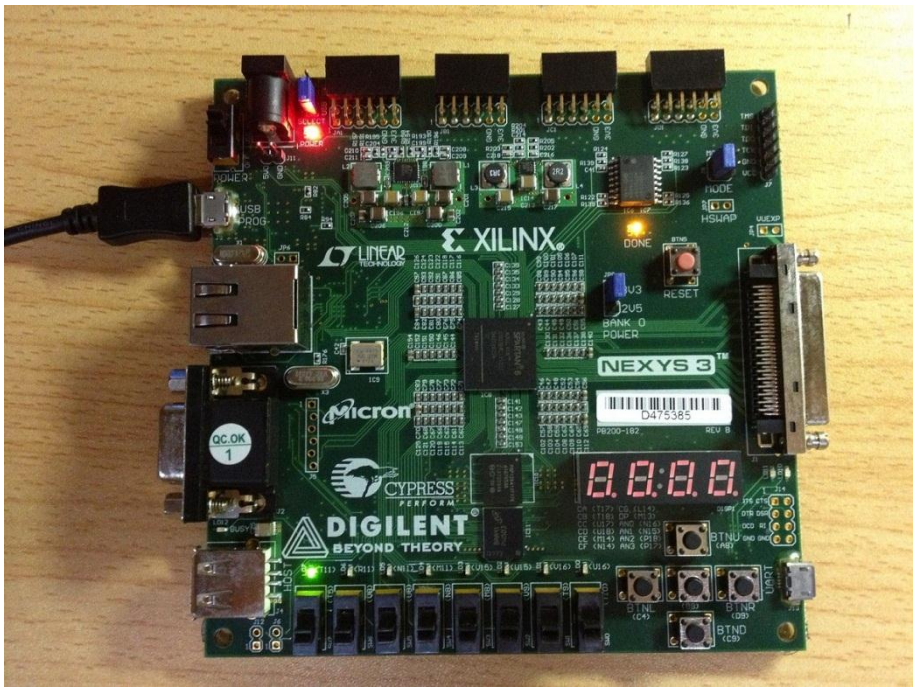
经过这几个步骤后，得到.bit 文件。

4、真机测试

使用 Adapt 上传二进制程序到开发板后，挑选三个样例进行测试。

b=00, a=00:

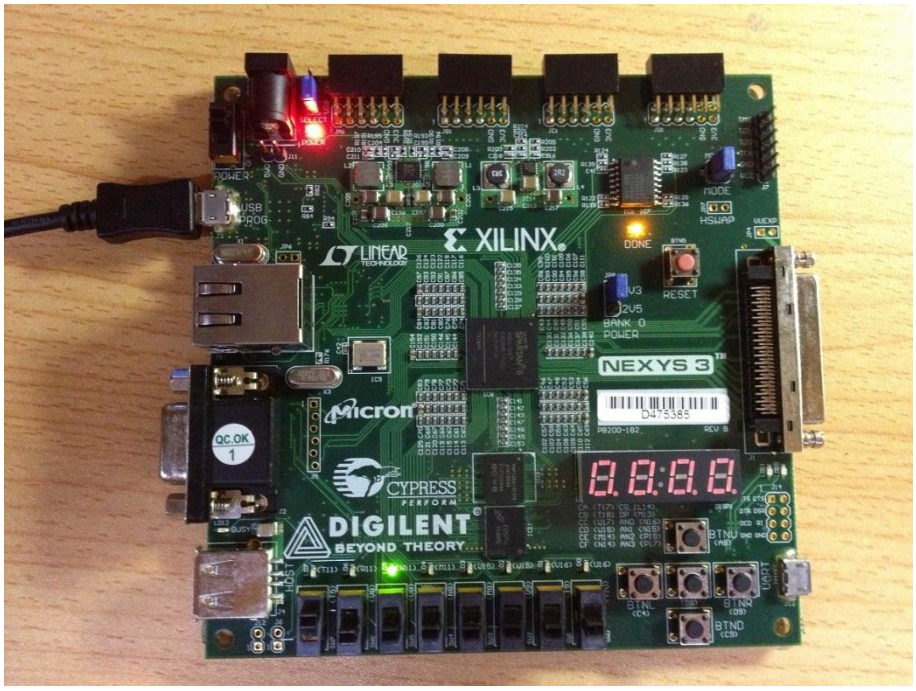
预计 a_eq_b=1, a_gt_b=0, a_lt_b=0。



测试结果正如预期。

b=10, a=00:

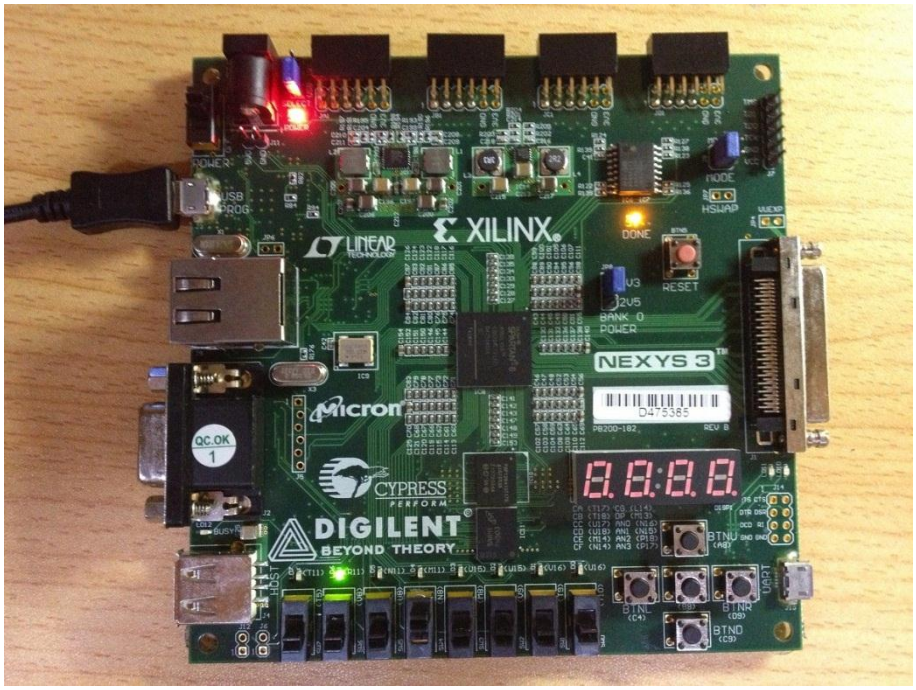
预计 a_eq_b=0, a_gt_b=0, a_lt_b=1。



测试结果正如预期。

b=00, a=01:

预计 a_eq_b=0, a_gt_b=1, a_lt_b=0。

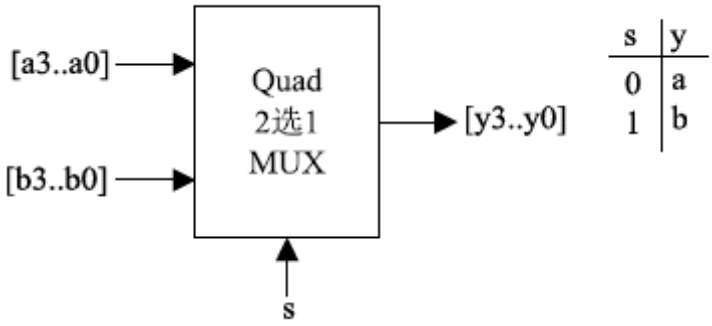


测试结果正如预期。

三、Lab3 (4 位 2 选 1 多路选择器的设计与实现)

1、使用 Verilog 语句表达多路选择器的逻辑

根据要求实现的选择器的电路图及真值表，如下：



可以很容易地利用 Verilog 语言中的一个三元操作符实现。对应的 Verilog 语句如下：

```
assign y = (s) ? (b) : (a);
```

在补充完整程序逻辑模块后即完成了程序的编写。

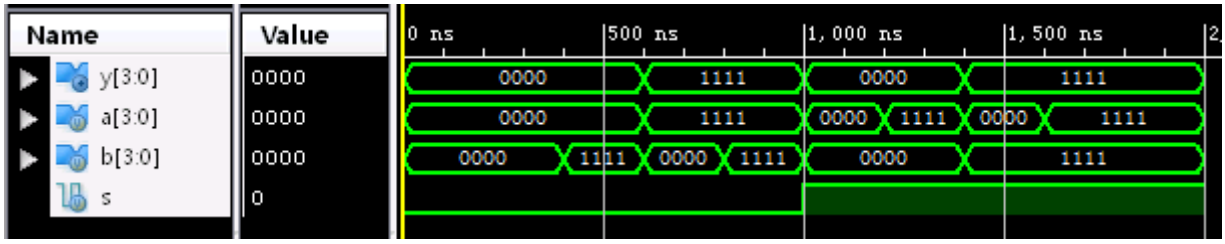
2、多路选择器的模拟运行

该程序有多个输入口与多个输出口，每个独立测试没有必要。只需要确定 s 对 y 的输出具有决定性的选择作用即可。故相应只需要测试八个情况即可：

```
#100
s <= 0;
a <= 4'b 0000;
b <= 4'b 0000;

#200
a <= 4'b 0000;
b <= 4'b 1111;

#200
a <= 4'b 1111;
b <= 4'b 0000;

#200
a <= 4'b 1111;
```

```
b <= 4'b 1111;


#200
s <= 1;
a <= 4'b 0000;
b <= 4'b 0000;

#200
a <= 4'b 1111;
b <= 4'b 0000;

#200
a <= 4'b 0000;
b <= 4'b 1111;

#200
a <= 4'b 1111;
b <= 4'b 1111;
```

运行后，模拟的结果如下：



模拟结果显示，当 s=0 时无论 b 怎么变化，y 输出总是只与 a 相关；当 s=1 时，无论 a 怎么变化，y 输出总是只与 b 相关。故达到效果，测试通过。

3、综合、实现及程序生成

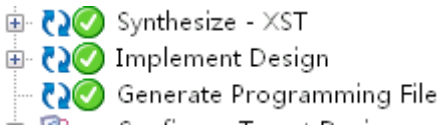
管脚约束如下：

```
NET "y[3]" LOC = "T11";
NET "y[2]" LOC = "R11";
NET "y[1]" LOC = "N11";
NET "y[0]" LOC = "M11";
NET "s" LOC = "B8";
NET "b[3]" LOC = "T5";
NET "b[2]" LOC = "V8";
NET "b[1]" LOC = "U8";
```

```
NET "b[0]" LOC = "N8";
NET "a[3]" LOC = "M8";
NET "a[2]" LOC = "V9";
NET "a[1]" LOC = "T9";
NET "a[0]" LOC = "T10";
```

该约束将靠左边的四个按钮归给 b[3:0]，将靠右边的四个按钮归给 a[3:0]，将右边呈十字的五个按钮的中间那个按钮归给 s，将前四个 LED 等归给 y[3:0]。

分别进行综合、实现以及程序生成，运行结果如下：



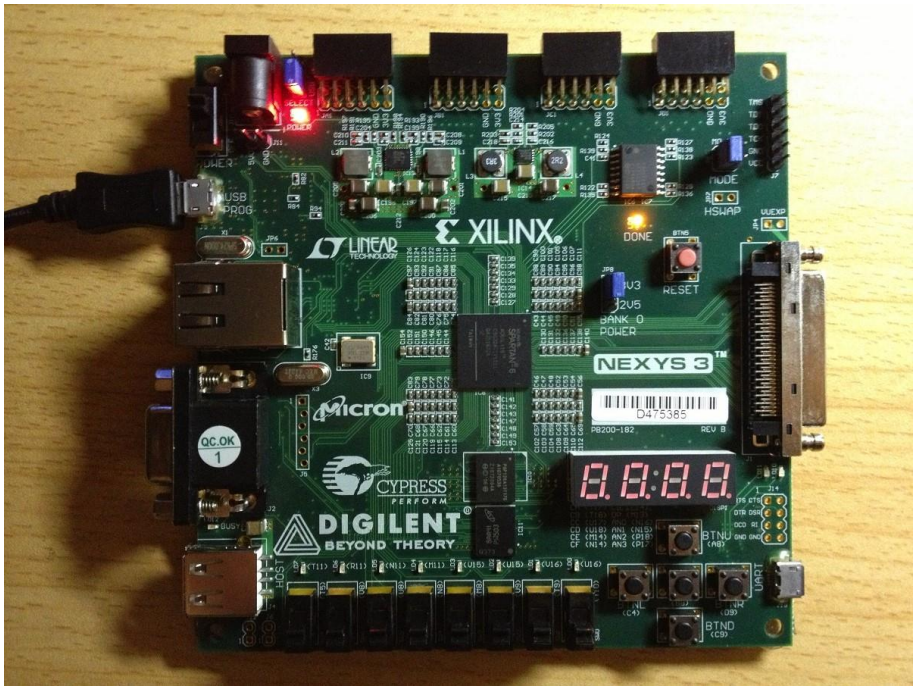
表明生成成功，并得到.bit 文件。

4、真机测试

将.bit 文件上传到开发板上，并进行几个样例测试。

s=0, b=0000, a=0000:

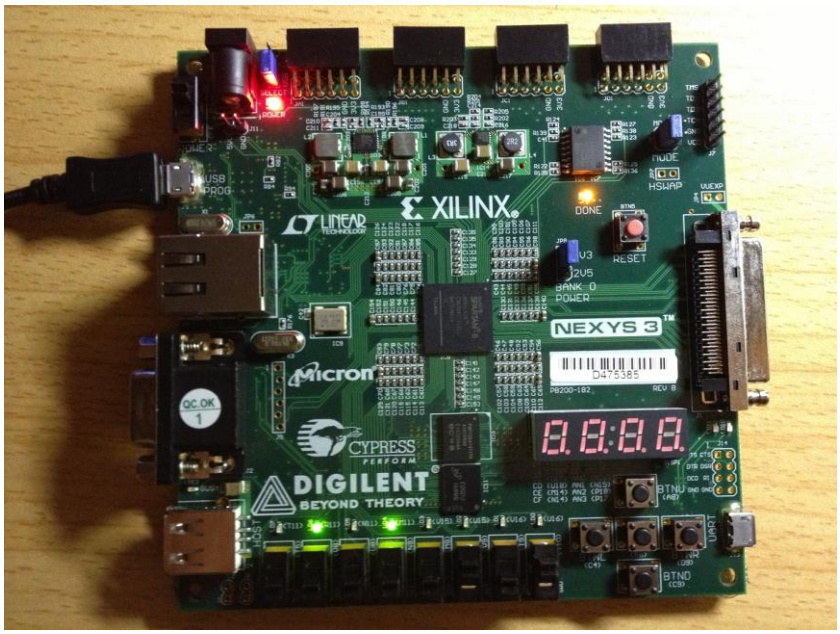
预期输出为 0000。



输出结果正如预期。

s=0, b=0000, a=0101:

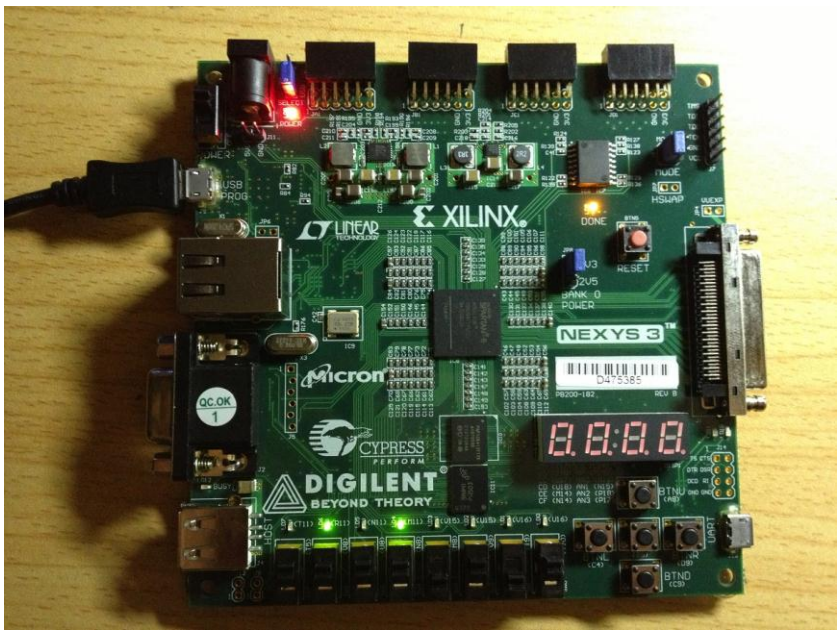
预期输出为 0101。



输出结果正如预期。

s=0, b=1111, a=0101:

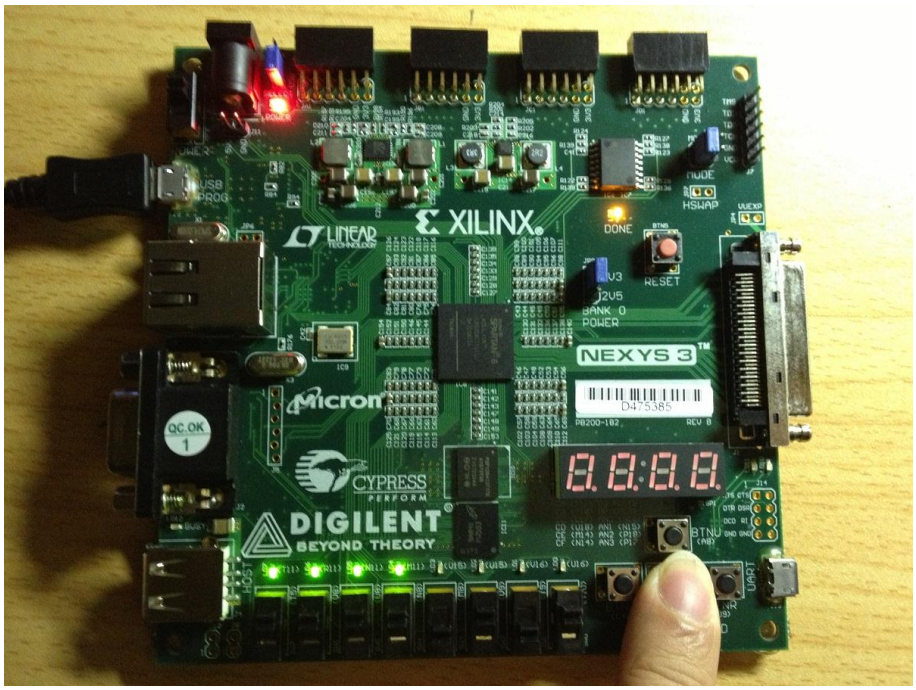
预期输出为 0101。



输出结果正如预期。

s=1, b=1111, a=0101:

预期输出结果为 1111。



输出结果正如预期。

四、Lab4 (7 段译码器的设计与实现)

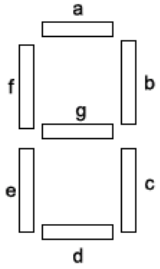
1、使用 Verilog 语句表达 7 段译码器的逻辑

7 段译码器需要建立一个由数字到数码管显示的映射。该映射可以首先通过真值表进行得出，再通过卡诺图进行化简。可是化简操作非常耗费脑力与时间，为了可以充分利用 ISE 的性能，故我直接将真值表写入到程序中，让 ISE 自动进行电路优化。题目提供的真值表如下：

X	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
B	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
D	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

1 = off

0 = on



我通过或操作对使某段数码管不显示的数据集合进行了编写，由于数码管要发光，则对应的输入应该为 0，故需要用或操作对不显示的内容进行汇集。例如对 a 段数码管，不显示的数据有 1、4、11(B)、13(D)，则对应的程序如下：

```
assign a = X==1 | X==4 | X==11 | X==13;
```

另外 6 段数码管以此类推。

因为在开发板上的数码管有小数点，为了方便进行消除，故我在该模块的输出中提供可高电平及低电平输出：

```
module DEC_7seg( ..., output wire act, neg )
```

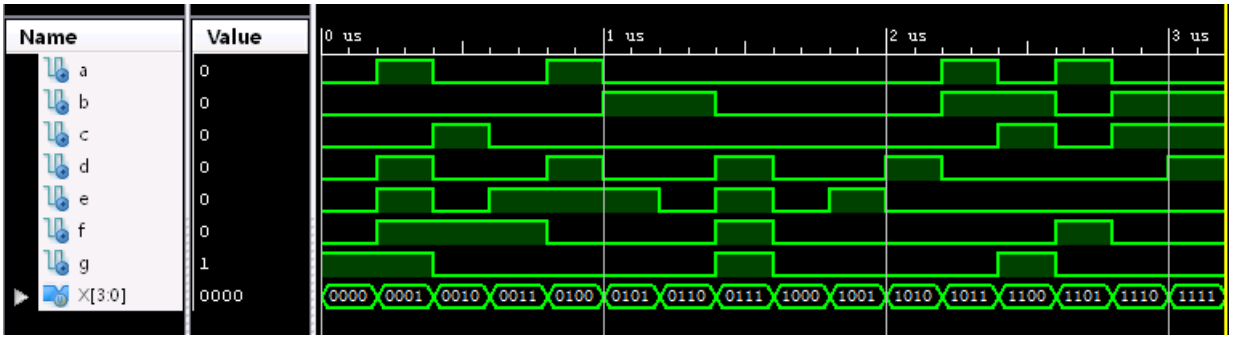
act 及 neg 两个参数并不是程序逻辑本身，只是为了方便进行显示而设计的。

2、7 段译码器的模拟运行

7 段译码器的输入有 16 种可能，为了进行完整性测试，故我设计了一个循环，让输入每 200ns 加一。核心测试程序如下：

```
always begin
    #200
    X = X + 1;
end
```

运行模拟后，得到的波形如下：



波形经过对照后，符合预期设想。

3、综合、实现及程序生成

该此实验我使用了开发板上的数码管进行显示，相应的管脚约束如下：

```
// Signal
NET "X[3]" LOC = "T5";
NET "X[2]" LOC = "V8";
```

```
NET "X[1]" LOC = "U8";
NET "X[0]" LOC = "N8";

// 7-segment decoder
NET "a" LOC = "T17";
NET "b" LOC = "T18";
NET "c" LOC = "U17";
NET "d" LOC = "U18";
NET "e" LOC = "M14";
NET "f" LOC = "N14";
NET "g" LOC = "L14";

// Initial setting
NET "act" LOC = "M13";
```

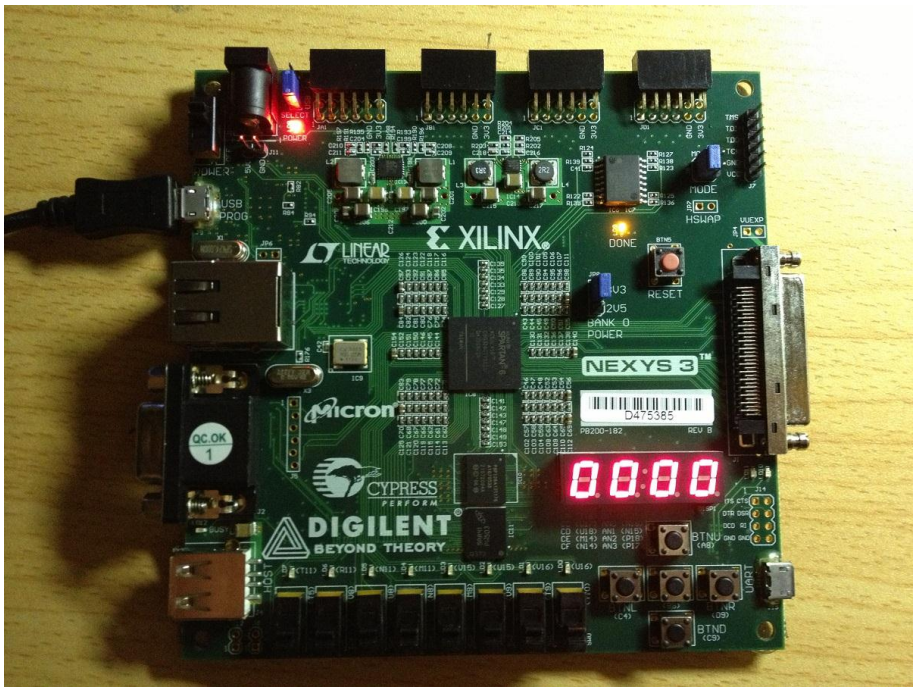
由于暂时知识有限，还不知道 AN0-AN3 端口需要如何使用才能够让仅有一个数码管进行显示，故现在这个约束，会同时让 4 个数码管进行显示。

经过编译后，程序显示成功通过，并得到对应的.bit 文件。

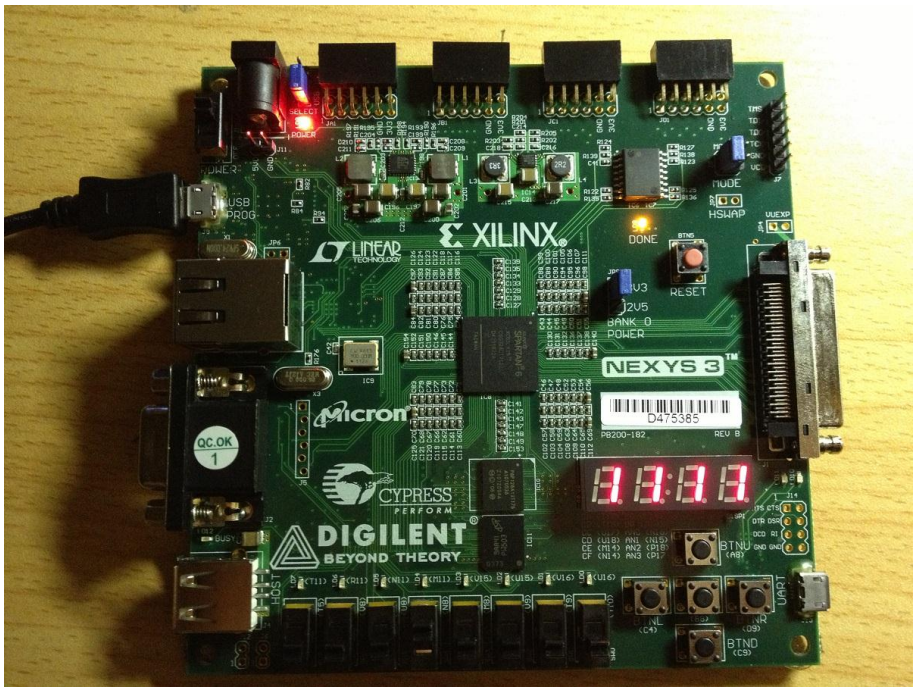
4、真机测试

将.bit 文件上传到开发板上，然后在这里我进行了完整的测试。

以下是输入为 0 的测试结果：



然后拨动开关，将输入设置为 1，结果如下：



重复上面操作，并得到之后输入为 2-15 的对应显示效果：

均满足设计需求。

五、Lab6 (3-8 译码器的设计与实现)

1、使用 Verilog 语句表达 3-8 译码器的逻辑

3-8 译码器的逻辑在我目前的知识看来，没有太多的便捷解法，只能够直接对输出进行赋值了。其真值表如下：

a2	a1	a0	y0	y1	y2	y3	y4	y5	y6	y7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

只需要将对应的二进制数分别映射到译码输出即可。

```
assign y[0] = !a[2] & !a[1] & !a[0];
assign y[1] = !a[2] & !a[1] & a[0];
assign y[2] = !a[2] & a[1] & !a[0];
assign y[3] = !a[2] & a[1] & a[0];
assign y[4] = a[2] & !a[1] & !a[0];
assign y[5] = a[2] & !a[1] & a[0];
assign y[6] = a[2] & a[1] & !a[0];
assign y[7] = a[2] & a[1] & a[0];
```

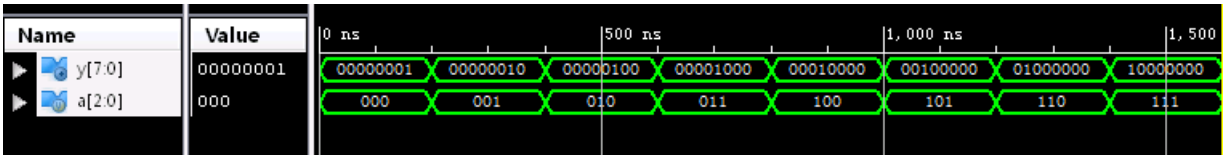
完善模块的其他内容后，程序部分编写完成。

2、3-8 译码器的模拟运行

为了进行完整性测试，我的程序使用了一个循环语句，每 200ns 对 X 加一，则进行 1600ns 后，即可得到完整性的测试结果。测试语句如下：

```
always begin
    #200
    a = a+1;
end
```

运行模拟，得到如下模拟测试波形：



该段波形显示，输入从 000 至 111 变化的过程中，输出依次由 00000001 变化到 10000000，符合译码器的工作原理及需求，故该程序通过模拟测试。

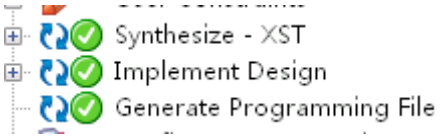
3、综合、实现及程序生成

管脚约束如下：

```
// Input
NET "a[2]" LOC = "T5";
NET "a[1]" LOC = "V8";
NET "a[0]" LOC = "U8";

// Output
NET "y[7]" LOC = "T11";
NET "y[6]" LOC = "R11";
NET "y[5]" LOC = "N11";
NET "y[4]" LOC = "M11";
NET "y[3]" LOC = "V15";
NET "y[2]" LOC = "U15";
NET "y[1]" LOC = "V16";
NET "y[0]" LOC = "U16";
```

该段约束将前三个按钮分配给 a，依次将八个 LED 灯分配给 y[7]至 y[0]作为输出显示。

进行综合、实现及程序生成操作后，结果如下：



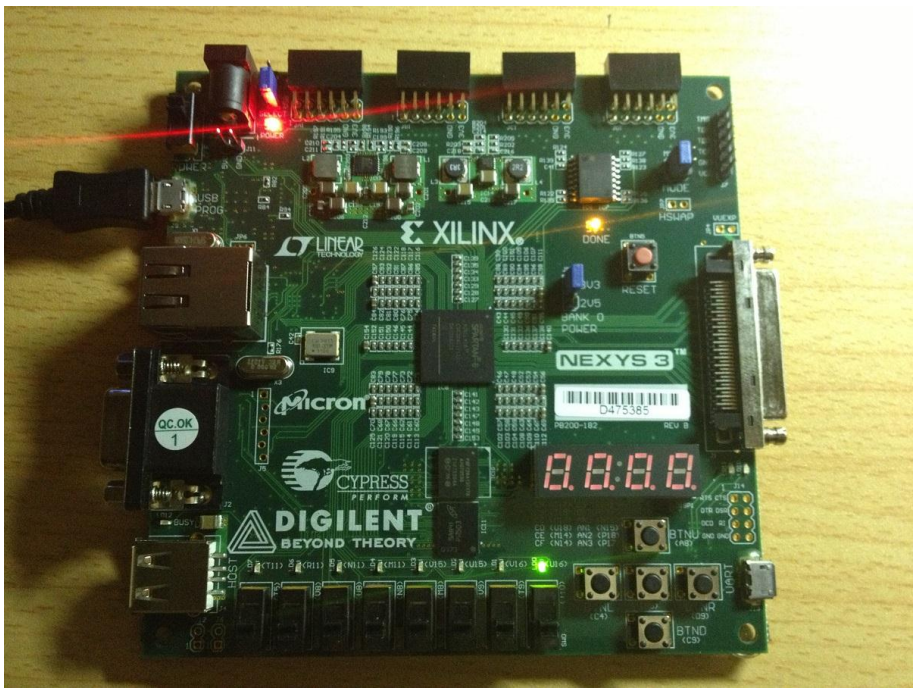
程序通过编译，并得到对应的.bit 文件。

4、真机测试

将生成的.bit 文件上传到开发板后，挑选几个样例进行测试。

a=000:

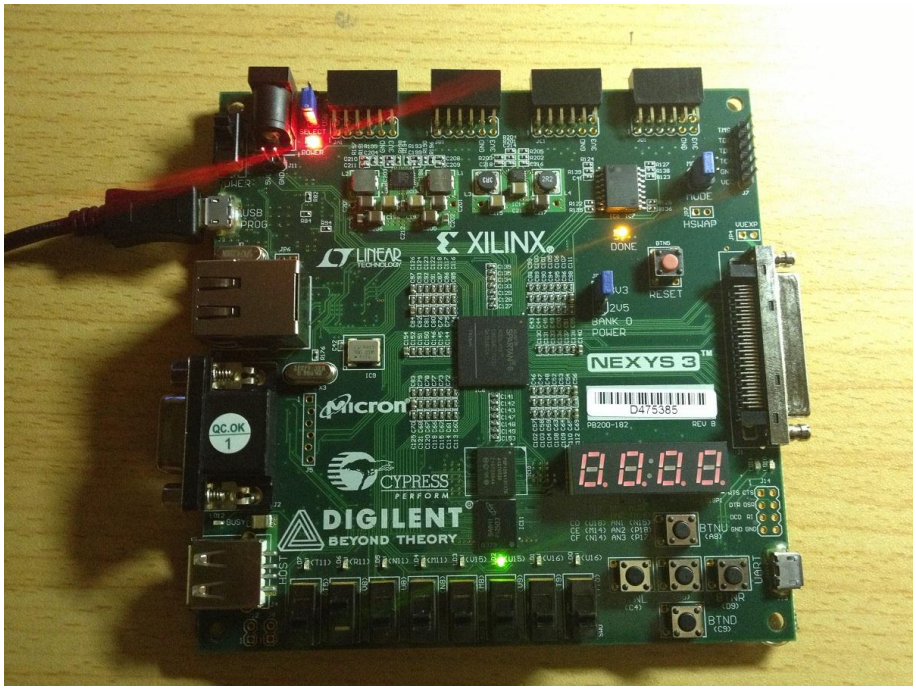
预期输出为 00000001。



输出结果符合预期。

a=010:

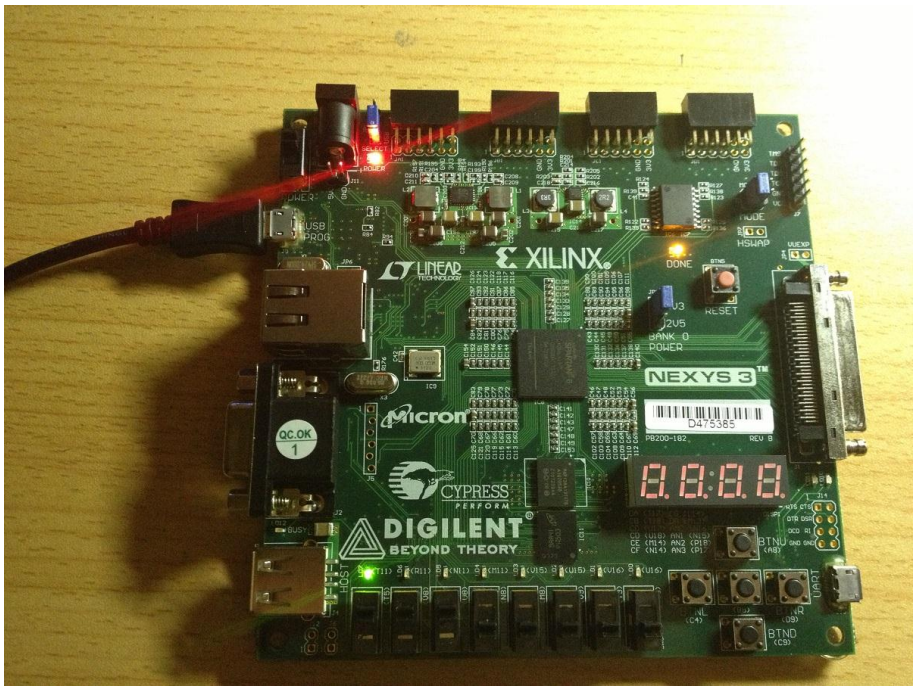
预期输出为 00000100。



输出结果符合预期。

a=111:

预期输出为 10000000。



输出结果符合预期。

六、实验感想

这一次的实验，一共做了 4 个，说实话虽然都不难，可是的确耗费了好多时间去做。

虽然这些实验在数字电路的实验课上都做过了，可是那个是利用逻辑电路来做的，每做一个功能，要么就要有现成的芯片，要么就要通过现成的芯片去搭出这样一个功能模块，非常地不方便、折腾人。然而在嵌入式系统课上，我们学习了如此厉害的工具，能够使用编程语言，就实现了电路的排布设计，而且电路的优化问题，全部都已经实现了自动化。这是一个非常伟大的工具，它能够节省我们非常多的时间，让我们更关注设计本身。

很多人都很机械，看到实验课件上给出了真值表，就想都不想马上动手按照真值表来干。或者有些同学进化得不完全，居然用卡诺图来化简真值表，然后再通过复杂的电路运算来表达出题目的需求。这真是诋毁了 ISE 这套工具的优化功能，并且小看了 Verilog 的威力了。我想的是，既然 Verilog 这么多人使用，一定是对每个基本功能都提供了非常便捷的操作的。所以经过询问我们的 TA，并上网查找资料，我学到了不少更深入的 Verilog 的语法。当然这些语法有不少在一开始就给我们讲解过了，可是光讲光听是没用的，即使老师都有讲过，没使用过的话，马上就忘记了。与其说我在网上找到了很多关于 Verilog 的新内容，不如说我是在复习第一次课的内容罢了。

经过这次高强度的实验，我发现 Verilog 实在太强大了，简直就是一个电路化的 C 语言，什么东西都提供得好好的，只不过是使用起来还需要纠结一下相关的语法。毕竟这个语法看起来更像是 VB 或者 Python 什么的。不过它能提供的自动化功能，已经是让我们的效率提高了不知道多少倍了。

邱迪聪

2013 年 10 月 16 日

附录 1：附件列表

lab2 COMP_2bit

\COMP_2BIT.v

\COMP_2BIT_ctr.ucf

\COMP_2BIT_test.v

lab3 MUX_4bit_2to1

\MUX_4bit_2to1.v

\MUX_4bit_2to1_ctr.ucf

\MUX_4bit_2to1_test.v

lab4 DEC_7seg

\DEC_7seg.v

\DEC_7seg_ctr.ucf

\DEC_7seg_test.v

lab6 DEC_3to8

\DEC_3to8.v

\DEC_3to8_ctr.ucf

\DEC_3to8_test.v

其中 X.v 文件为程序模块文件，X_ctr.ucf 为管脚约束文件，X_test.v 为模拟测试代码文件。