实验一 组合逻辑实验

## 一、系统设计

本次实验实现了三个组合逻辑电路：4位数据传输，数码管显示，7位汉明码纠错。由于系统功能简单，避免赘述，三个简单的结构用图1.1表达。

图1.1 系统框图

实验根据lab1指导中的步骤完成，即先完成MUX和DEMUX模块，然后完成数码管显示模块，最后完成汉明码纠错与显示。

这一顺序是合理的，数码管的选通信号输入S和输出O\_led就是一个DEMUX分配器。数码管的驱动只输入了选通信号I，实际上可以认为是已经内置好了MUX的16个7位输入，并选择一个进行输出。最后汉明码纠错的功能模块就是简单的组合逻辑，而显示模块直接调用数码管显示模块即可。

## 二、实现细节

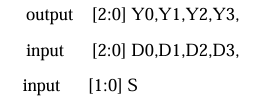
由于端口定义在实验指导中已经给出，作者在实现时几乎没有修改，故此处直接截取指导中的端口定义。值得注意的是，在图2.3中，端口ERR\_adr与端口an意义重复，在模块内部将ERR\_adr用作选通数码管地址an，故实际实现时删去了端口。

图2.1 数据传输模块端口定义

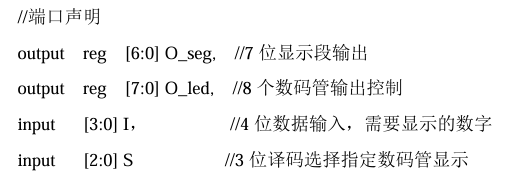
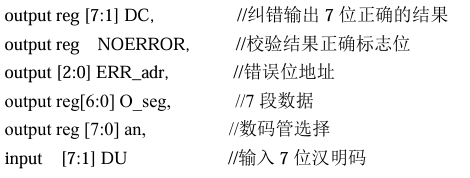
图2.2 数码管显示模块端口定义

图2.3 汉明码纠错与显示模块端口定义

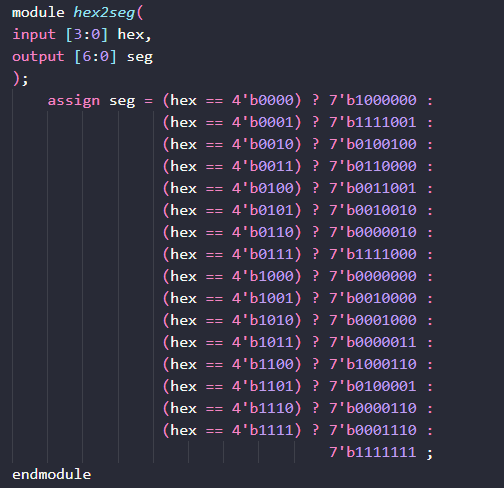
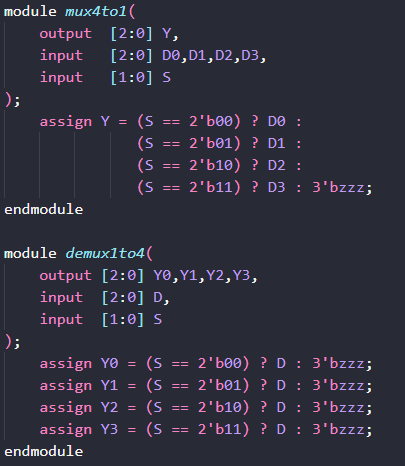
具体实现采用了数据流建模的方式构建了组合逻辑，例如图2.4中的mux、 demux、hex2seg模块。

图2.4 数据流建模实例

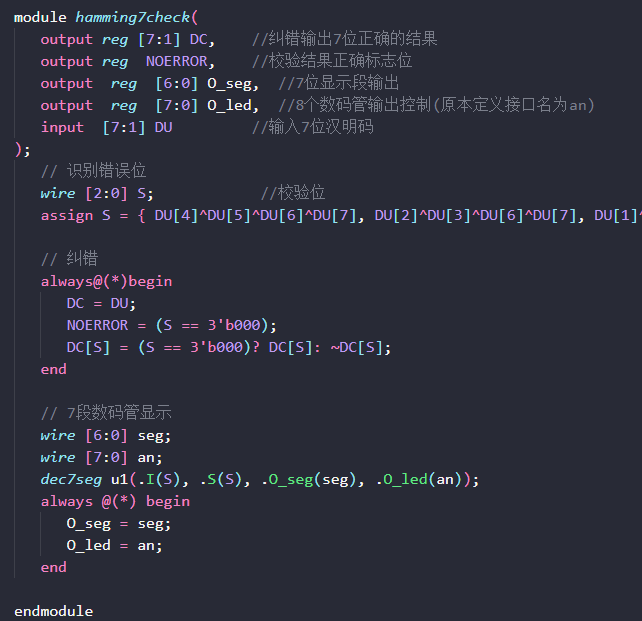
得益于模块的设计的优势，在顶层模块中直接实例化已有功能模块进行组合，让整体设计结构清晰。例如图2.5中的汉明码纠错模块，直接调用前一部分的数码管模块，提高可读性。

图2.5 层次化设计实例

## 三、实验结果

3.1 波形仿真

3.1.1 数据传输波形

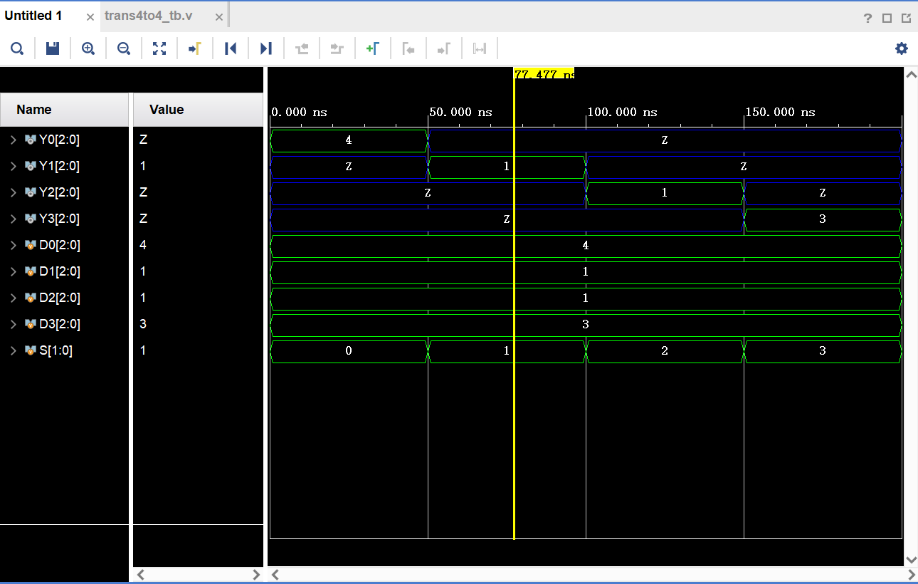
波形仿真结果见图3.1.1，可以发现随着选通信号S变化，被选通的输出Y[2:0]的值与对应被选中的输入D[2:0]的值相同。

图3.1.1 数据传输波形

3.1.2 数码管显示波形

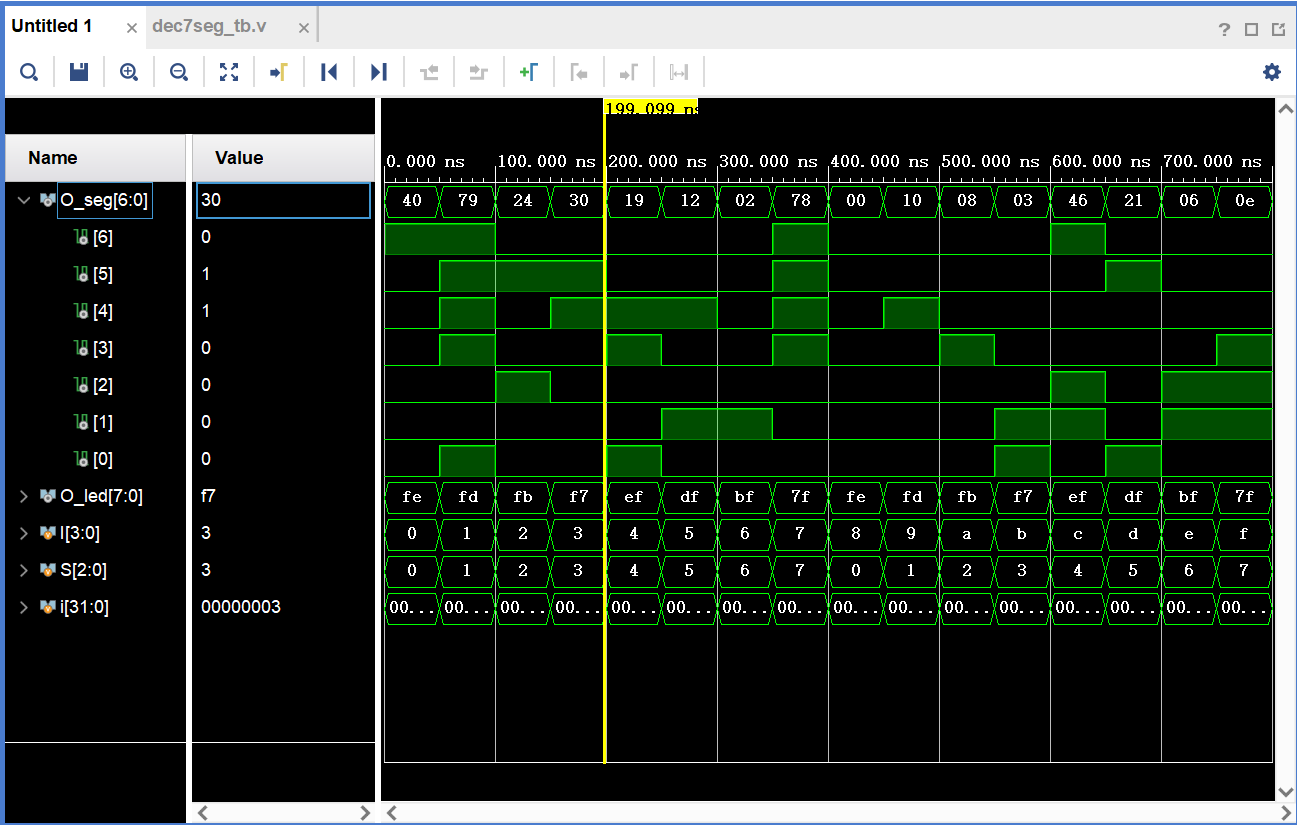
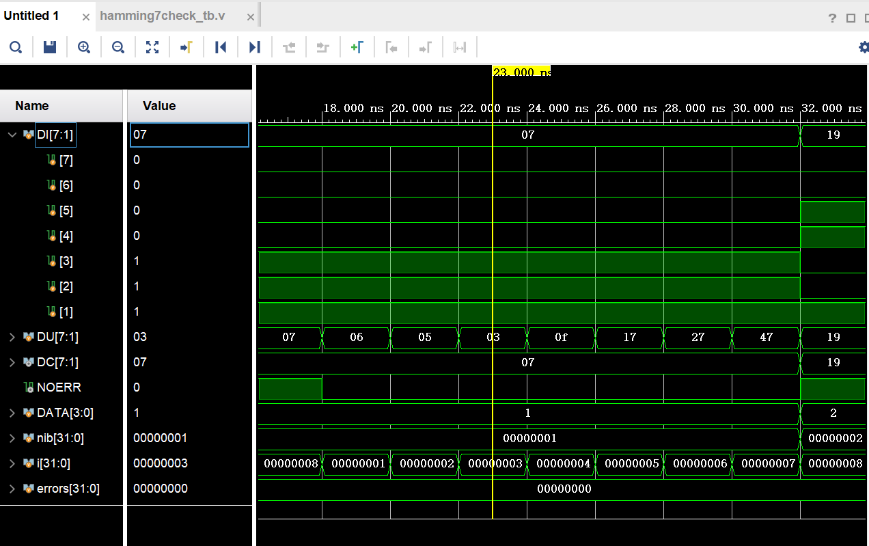
波形仿真结果见图3.1.2，随着数码管选通信号I变化，O\_led对应变化；随着输入待显示数据变化，O\_seg信号的值查表知是正确显示的值。

图3.1.2 数码管显示波形

3.1.2 汉明码纠错波形

波形仿真结果见图3.1.3，以输入数据为1为例，对应数据位是0001，但输入数据DU位03（7’b0000011），说明检测出第三位出错，结果应该是0000111，数据位为0001，成功恢复。

图3.1.3 汉明码纠错波形

3.2 上板测试

3.2.1 数据传输上板结果

见图3.2.1，最右侧选通拨码为11，选通了第9-11个拨码。LED被成功对应点亮

图3.2.1 数据传输上板测试结果

3.2.2 数码管显示上板结果

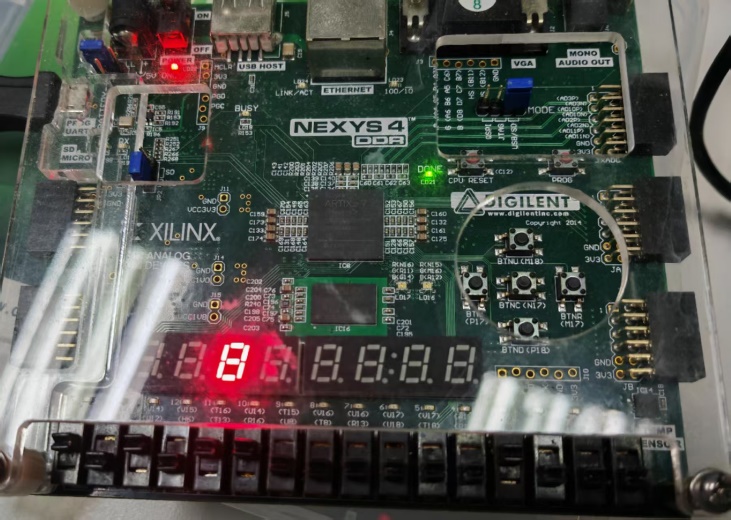
见图3.2.2，最右侧三位选通拨码为101，对应第5个LED亮。第四个拨码被拨动代表数据为1000，应显示8，与结果对应。

图3.2.2 数码管显示测试结果

3.2.3 汉明码纠错上板结果

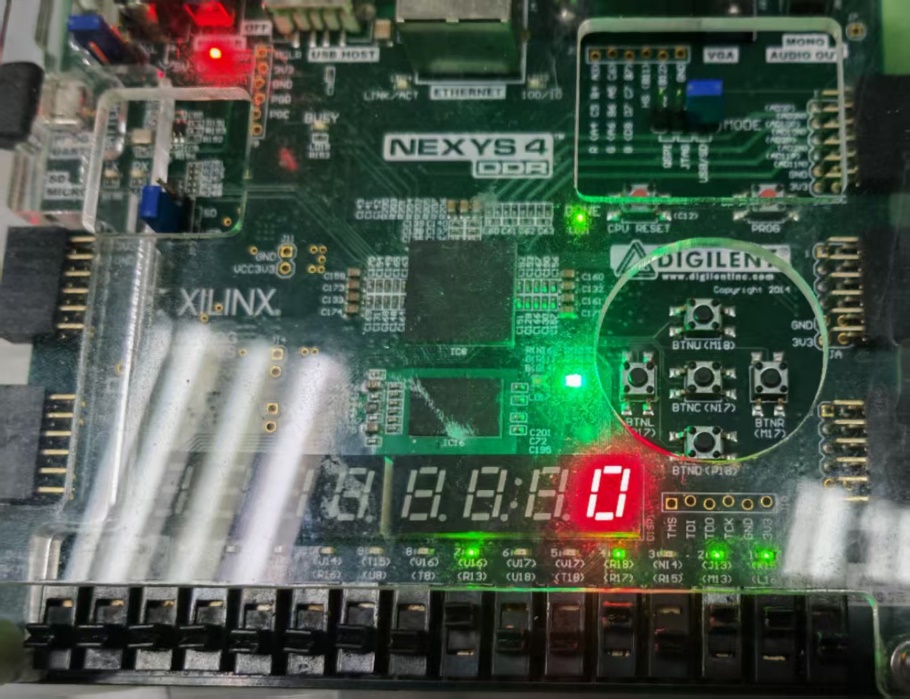
见图3.2.3，拨码输入为0010110，故障位为000，没有错误因此第0个数码管显示0，并且RGB显示绿灯，符合要求。

图3.2.3 汉明码纠错测试结果

四、反思与总结

4.1 挫折与进步本部分包含遇到的问题、解决方法、实验感想

第一次亲手写verilog时总是进行语法报错，与课前测试平台以及其他语言不同，编译器不会在命令栏给出错误行数(有的编译器/解释器还会给出具体的错误原因)，Vivado需要访问给出的日志文件可以看到错误行数与大致的错误原因(尽管对于老手来说可能一眼就知道错在哪里，但是我对于仅有的以据syntax error还是会愣住)，在此总结了一些经验：

1. 在verilog中写begin end要像在C中写{}一样顺手 。

2. 行为级的语言，例如if、case等分支语句，必须在always块中描述。如果想在always外部使用分支，请用三目运算符或组合逻辑表达。

3. wire的本质只是连线，用assign来对其“赋值”。Wire不可以在always块中赋值。Wire可以被赋值为reg，这本质上实在寄存器上连了一根名为wire的导线。

4. Reg可以在always中赋值，对于wire赋值给reg，通常用阻塞赋值，对于reg赋值给reg通常采用非阻塞赋值。

5. 不可以在always块中进行实例化！！！

6. 传入实例化的端口只能是wire型，因为代表了与实例化模块的物理连接，尽管实例化模块的端口被声明为了reg型。

4.2 思考题

1. 设计32位比较器，分析资源占用

比较器输入32bits DEST和32bits SRC，输出结果标志位L, S, E，分别代表DEST > SRC，DEST < SRC，DEST = SRC。具体实现有两个思路，一个是从MSB到LSB逐位比较判断，实现方法是将两个输入逐位异或，异或输出和DEST对应位相与（即异或输出选通该位输出）得到输出32 bits comp，最后将所有的异或相与得到32bits equal。显然，comp代表不同的位数中，DEST对应的位数是0还是1，对于最高的不同位（由equal可知）若为0则小于，若为1则大于，若comp全0且equal全0代表相等。

另一个思路是直接用32位减法器，根据结果的标志位分析大小。

由于Vivado提供了比较运算，因此EDA工具也可以直接根据我们输入的>综合出最优的电路。

资源占用情况，由于实现方法都是纯组合逻辑，只占用LUT资源。

对于方案一每一位并行的比较，只是实例化了32个异或门，32个与门，还有一个多输入的与门（可能需要多级级联）。

方案二生成的组合逻辑更庞大，因为这个电路还可以实现加法减法，实现比较器是顺手的事。

2. 设计32位译码器，分析资源占用

输入5位二进制编码，输出32位独热码，可以直接用case语句生成。电路实际优化应该是多个选择器，占用资源应只有LUT。

3. 用8位数码管展示学号，每秒移动一位

首先根据系统时钟分频得到1Hz的时钟，然后设计一个位宽4的，长度位9的循环移位寄存器（学号一共9位）。每次1s移位一次。取前8个数据，转换为七段数码管显示的数据输出即可。

细节上，循环移位寄存器可以声明一个reg [3:0] stu\_num [8:0]数组，然后用一套时序逻辑控制其循环移位。数码管需要一个更高频率的时钟驱扫描8个数码管。

4. 设计8位信息位的汉明码纠错

8位信息位，需要4位校验位，因此需要12位汉明码。校验位分别在第1，2，4，8位，具体的判断方法和4位信息位类似，因此不再赘述。