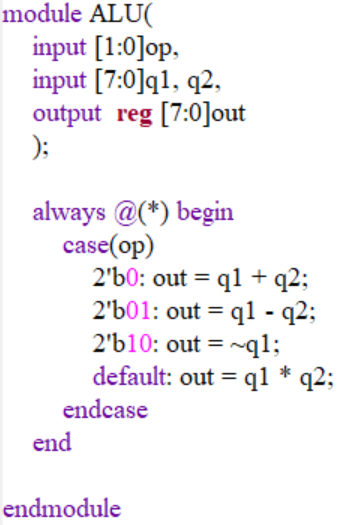
**实验报告——寄存器指令初步**

**一、实验设计**

本次实验相较前两次比较复杂一些，在本次实验中，我一共建了4个module，下面分别解释每个module的功能和实现，并在截图中给出部分代码实现：

1. **module ALU**，这个模块实现的是要求中的四种操作，+/-/\*/取反：、
   1. **输入**：操作符op（2-bit），和两个操作数q1和q2（8-bit）
   2. **输出**：两个操作数（或其中之一）在操作符下的运算结果，也是8-bit
   3. 因为共有四种运算，所以此处op用2位表示，0表示q1+q2，1表示q1-q2，2表示对q1取反，3表示q1\*q2
   4. 因为具体实现在第一次lab中已经基本完成，因此直接上代码：



1. **module regFile**，这个module对应的是文档中的RegFile，主要实现对32个寄存器内容的存储，读取n1和n2表示寄存器（以下简称n1寄存器和n2寄存器）中的值，将DI写入n1对应的寄存器
   1. **输入**：表示复位的reset(1-bit)，表示是否写入的WE(1-bit)，表示时钟的clk(1-bit),两个寄存器的地址n1和n2(5-bit)，要写入n1表示的寄存器的值DI(8-bit)
   2. **输出**：当对n1和n2寄存器读的时候，返回两个寄存器中的值q1和q2(8-bit)
   3. 寄存器中的值用一个包含32个8-bit的数的**二维数组data**表示



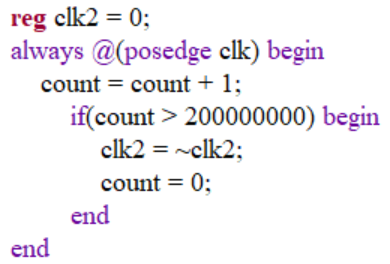
* 1. **分频处理**：

为了能够看到q值的变化，我们对clk做了分频处理，降低clk的频率到能够接受的范围（2-3s）；

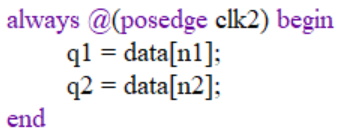
像文档中给出的那样，借助了一个新的时钟clk2和一个计数器count，初始值都为0，在每次clk的上升沿时，都会计数器增加一，当count加到2e8时，对clk2取反，count复位（变为0）；

这样的过程表示count个clk对应的时钟周期，clk2变化一次，之后利用clk2的上升沿和下降沿执行相应的读写操作，相当于将时钟周期降低为原来的1/count

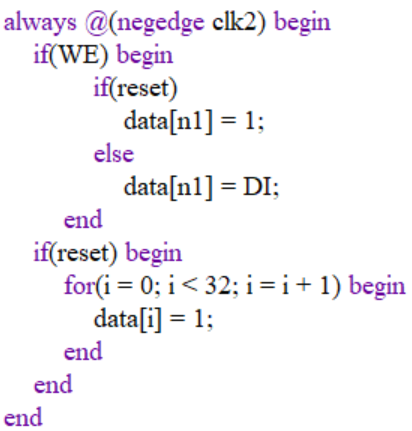
实现代码如下：



* 1. 时钟上升沿读出n1寄存器和n2寄存器的值，借助分频处理中产生的clk2，当clk2处于上升沿时，读出数据：



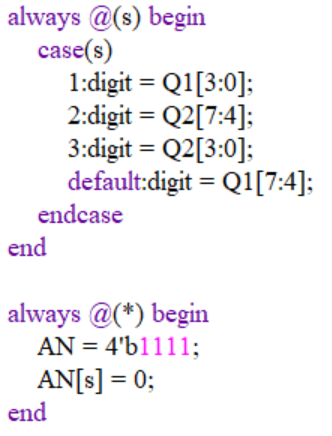
* 1. 时钟下降沿时将DI值写入n1寄存器，若reset=1，将所有寄存器的值复位（变为1）：



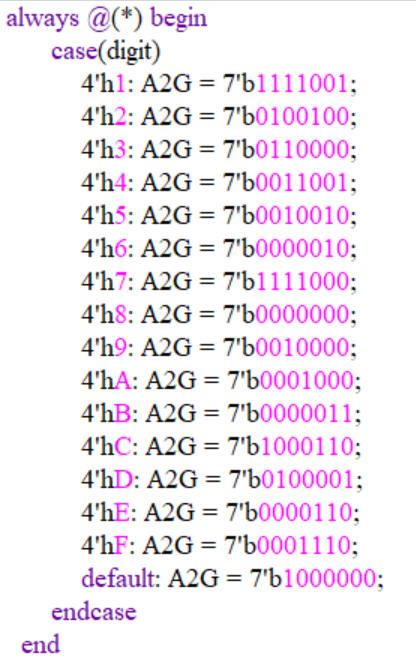
* 1. **注**：此处之所以将reset和写入操作放在同一个always块中，是因为verilog不允许在多个always块中对同一个数据对象做赋值；因为reset也是一种特殊的写操作（所有寄存器都写入1），所以此处和DI的写操作放在一起，这不一定是解决信号多驱动问题的最佳解决方案，但是这是我目前想到的最好的实践

1. **module display**，该模块实现了用七段显示管显示Q1和Q2
   1. **输入**：时钟clk(1-bit)，两个要显示的数值Q1和Q2(8-bit)
   2. **输出**：是否显示小数点DP(1-bit)，两个8位的数用四个七段显示管显示AN(4-bit)，每一位表示对应七段显示管是否被选择，七段显示管对应的七个管脚A2G(7-bit,a,b,c,d,e,f,g)
   3. **分时显示**：根据七段显示管的原理，如果要是每一个七段显示管显示不同的信息，需要做分时处理，借助一个20位的clkdiv，在每一次时钟上升沿，都对clkdiv增加一，那么，因为权重不同，对应位变化的频率也不同，所以可以用不同的位表示不同的频率，也就是不同的周期

在本次实验中，对于显示管的刷新频率，我取了190Hz（高于人眼所能接受的最低频率，24Hz）,对应于clkdiv也就是第18位，剩余两位s(2-bit)用来表示显示Q1和Q2的那一部分，显示在哪一个七段显示管上，s与Q1和Q2以及与四个七段显示管的对应关系如下：

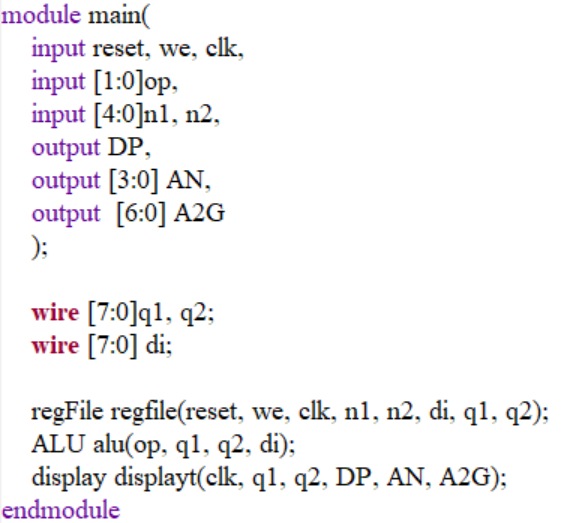


* 1. 根据s得到对应要显示的数值digit之后，可以通过case语句选择{a,b,c,d,e,f,g}的显示方式：



* 1. **注**：我们实验中用到的电路板是低电平有效

1. **module main**：这一模块实现的是顶层的控制，输入就是总输入，输出就是总输出，不再赘述，直接上代码：

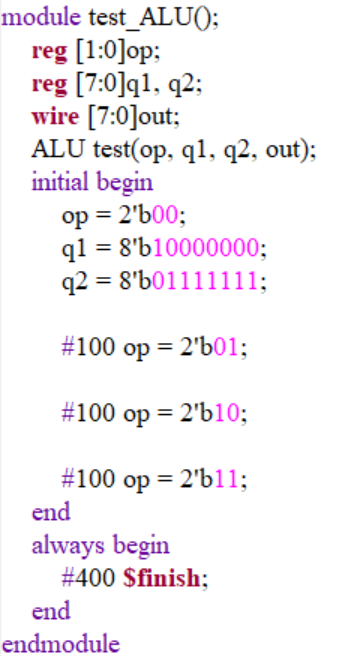
****

**二、仿真波形**

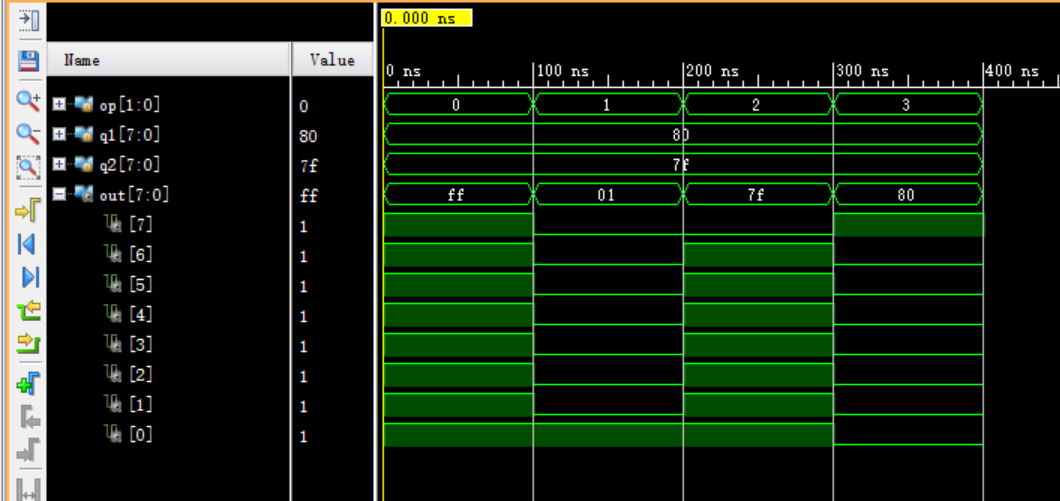
**1. module ALU：**

①**测试代码**：

该模块比较简单，在LAB1中已经实现过了，因此在此处只做了简单的测试，取了Q1=0x80，Q2=0x7F，分别做加减乘和取反运算



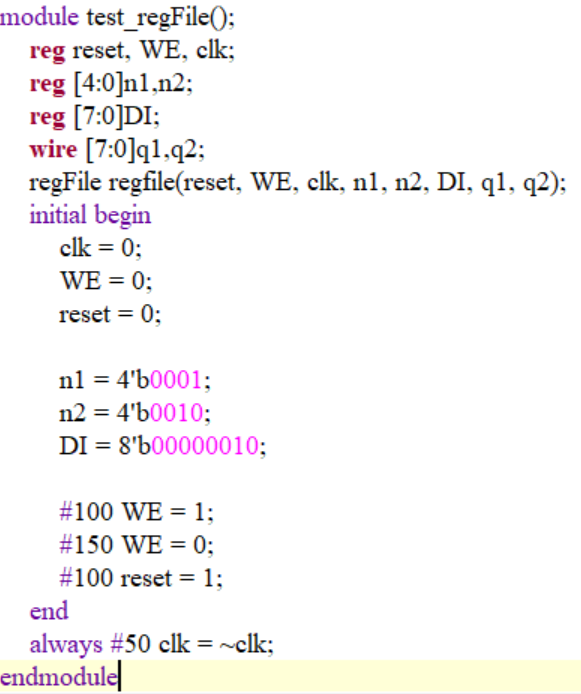
②**仿真波形**：



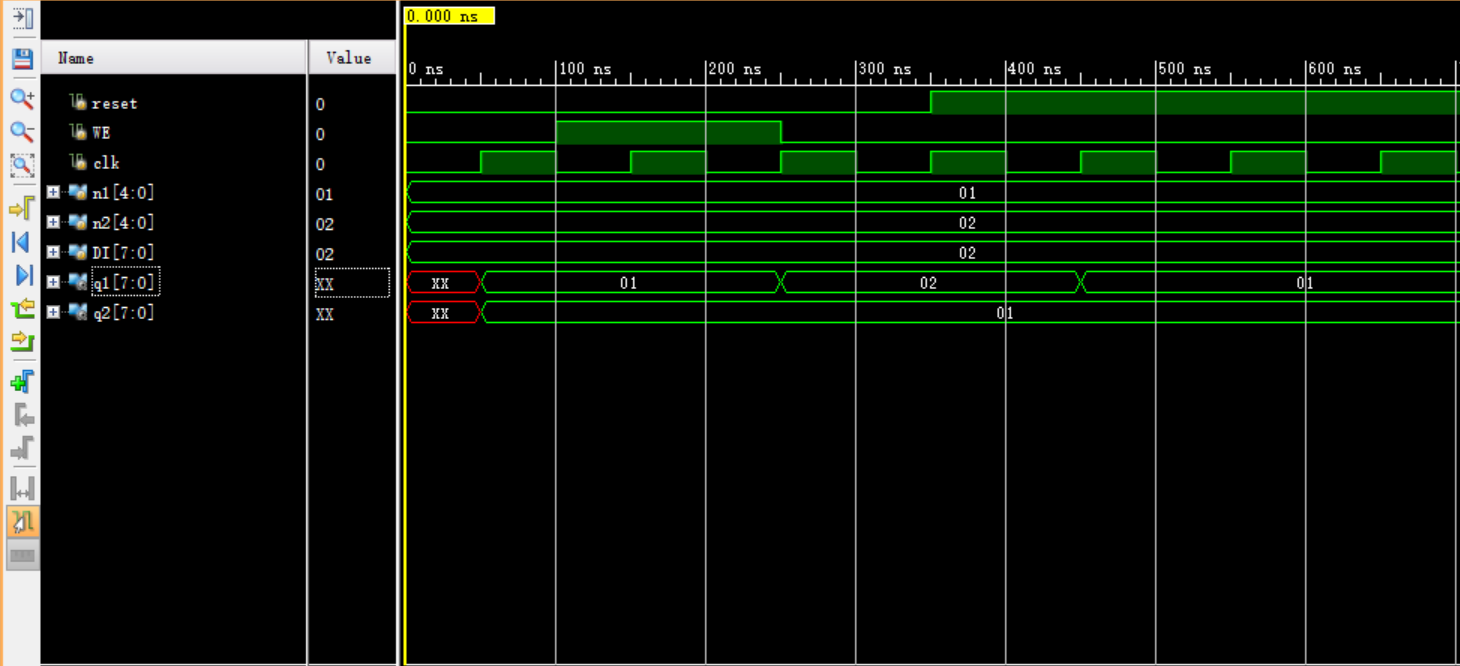
**2. module regFile：**

①**测试代码**：

对该模块的测试，我测试了初始状态写的地址为1和2两个寄存器的值，WE设为1后，将值2写入地址为1的寄存器之后，对应寄存器的值变化，以及reset之后，两个寄存器的值



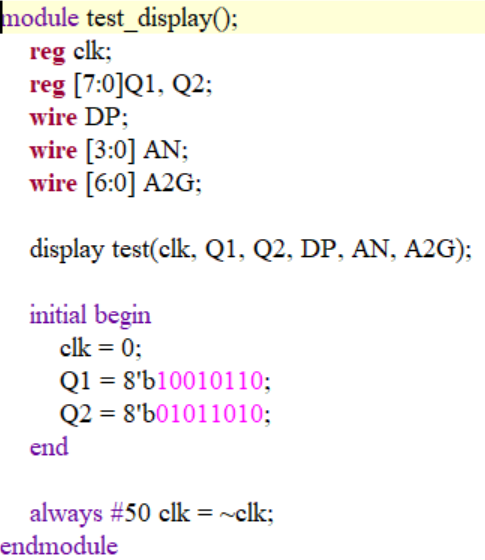
②**仿真波形**：下图中q1的三个值对应未写入，写入和reset三个测试



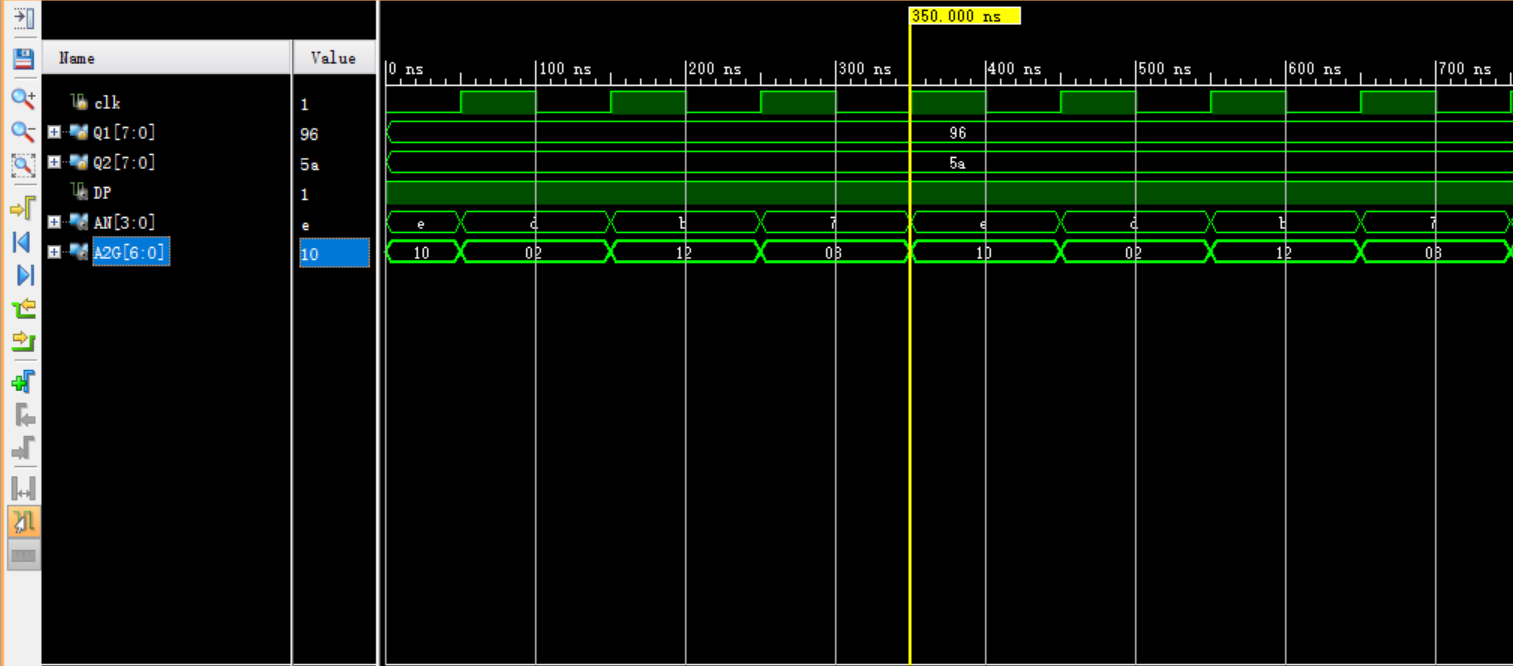
**3. module display：**

①**测试代码**：

该测试比较简单，设定了Q1=0x96，Q2=0x5A，将这两个数显示出来：



②**仿真波形**：



**4. module main:**

① **测试代码**：

因为涉及四种基本运算，所以我对这四种运算都进行了测试，每一个测试用到两个测试用例

**以下列出的代码截图都在同一个initial块中，在模块中的顺序与下面列举顺序相同**，整体的操作顺序就是（data是regfile数组）

data[1]+data[1],

data[2]+data[1],

data[1]-data[1],

data[2]-data[1],

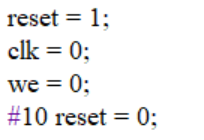
data[3]\*data[1],

data[4]\*data[2],

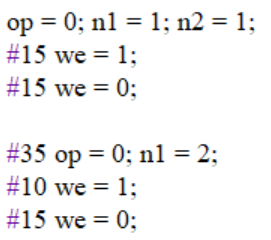
data[3]取反，

data[4]取反

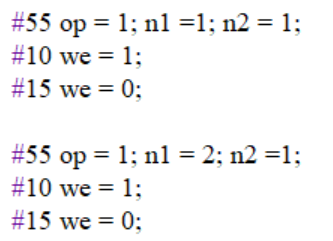
1. **初始化**：



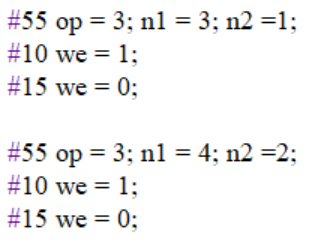
1. **加法**：第一种测试的是同一个寄存器上的加法；第二种测试的是不同寄存器上的加法



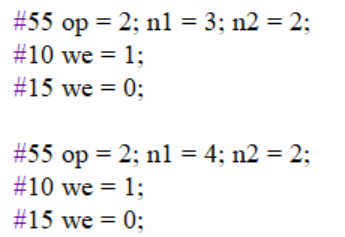
1. **减法**：第一个测的是同一个寄存器上的减法，第二个测的是不同寄存器上的减法



1. **乘法**：第一个测的是包含0的乘法，第二个是一般乘法

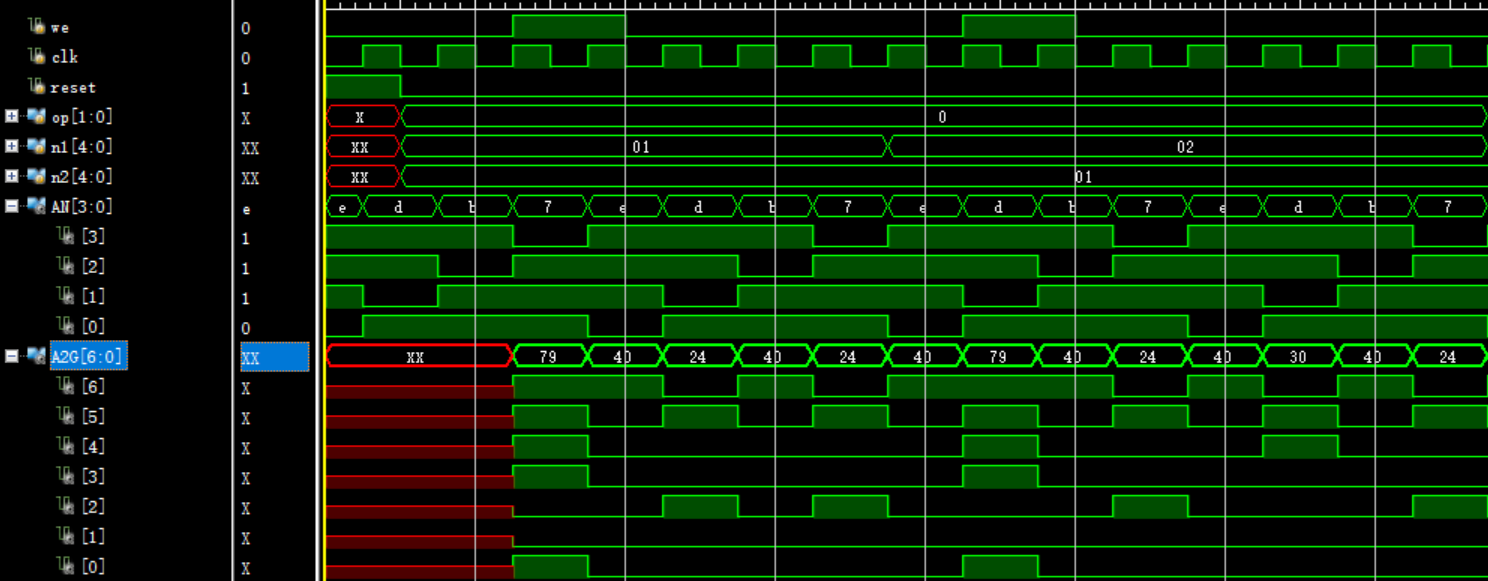


1. **取反**：第一个测的是对0的取反，第二个测试的对非0数的取反操作

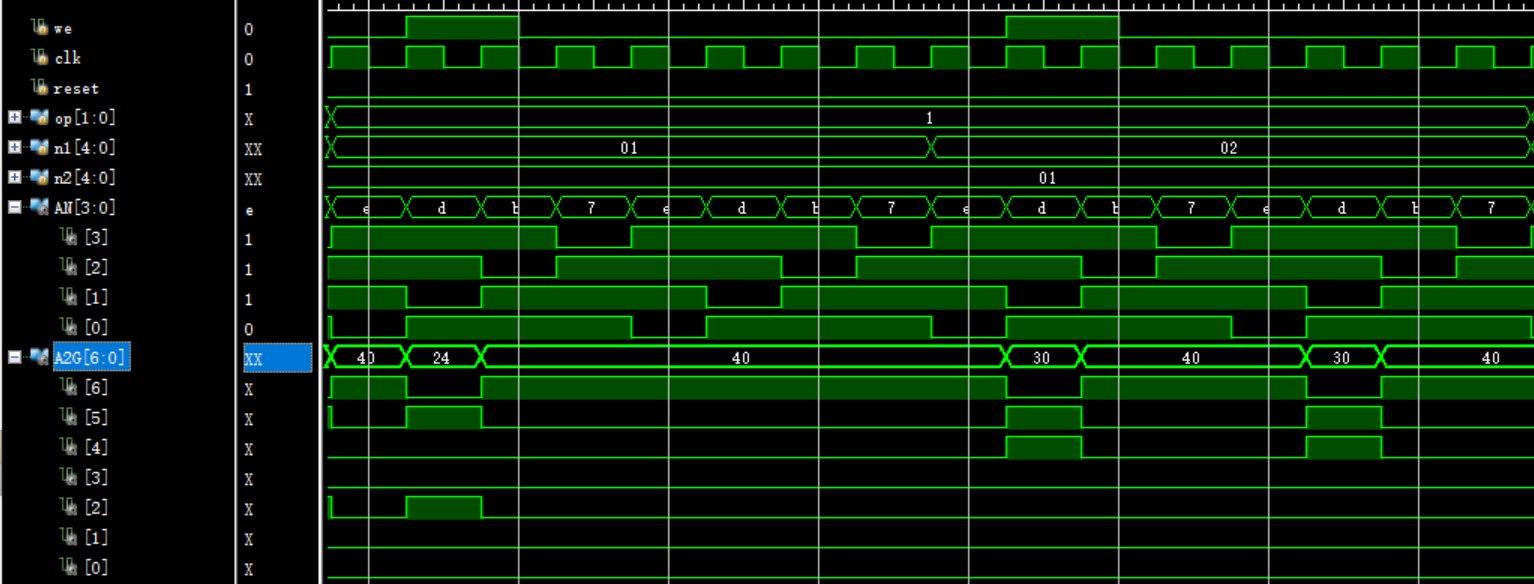


② **仿真波形**：（图中红色框框出的是每中测试用例下的q1和q2的值，前两位表示q1，后两位表示q2）

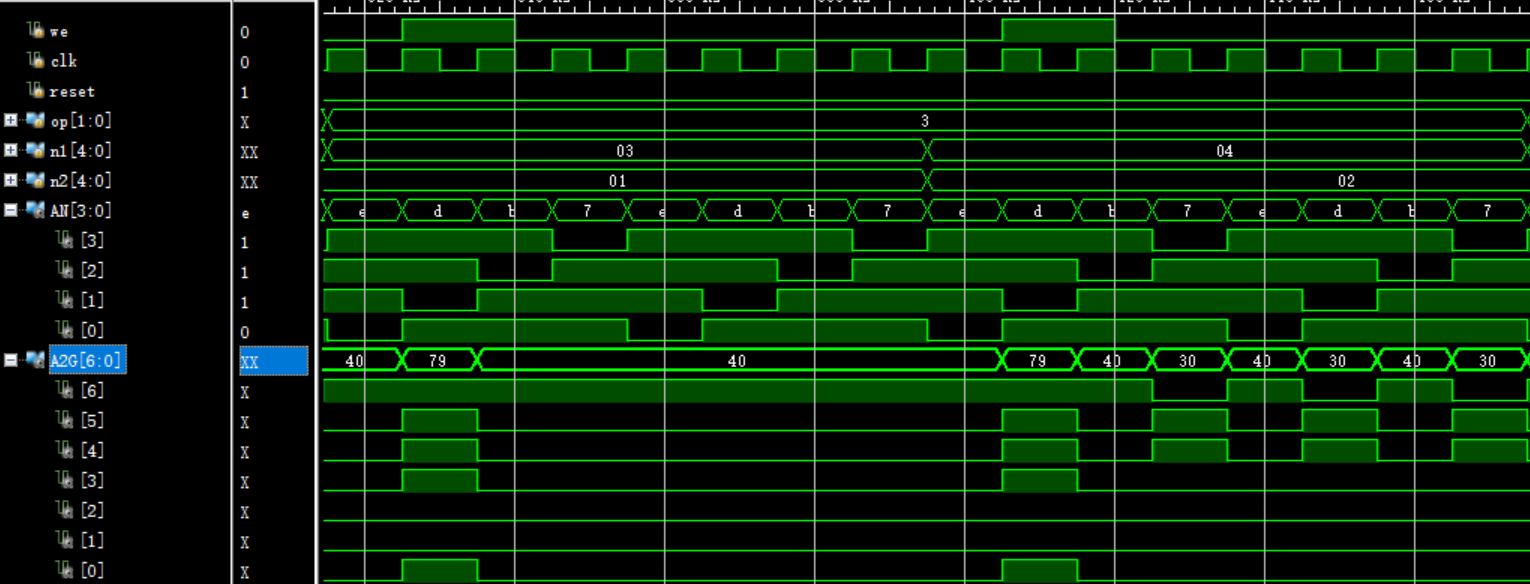
a> **加法**：



b> **减法**：



c> **乘法**：



d> **取反**：



**三、实验中遇到的问题和解决方法**

1.信号多驱动问题，指的是在不同的always块中对同一个数据对象进行写操作，这个问题出现在寄存器数组的两处写操作时，一处是reset，另外一处是将DI的值写入n1寄存器，解决方法就是将两个写入同一个always块中，上面对regFile的说明中已经提到。

2.端口连接问题，因为之前的两个lab中都是单一的module，不存在module的连接问题，但是在这个LAB中存在，去网上查了教程之后，概括如下：

将一个端口看成由相互链接的两个部分组成，一部分位于模块内部，另一部分位于模块外部。当在一个模块中调用另一个模块时：

输入端口：从模块内部来讲，输入端口必须为线网数据类型，从模块外部来看，输入端口可以连接到线网或者reg数据类型的变量。

输出端口：从模块内部来讲，输出端口可以是线网或者reg数据类型，从模块外部来看，输出必须连接到线网类型的变量，而不能连接到reg类型的变量。

3.我在写regfile时，为了解决信号多驱动问题，尝试将上升沿时间和下降沿事件合在一起写，结果出现了一些“怪异”的事情，查了资料了解到：如果在always的敏感列表中，同时包括一个信号的上升沿和它的下降沿的话，这两个事件会合并为一个电平事件。

**四、实验的收获与感想**

1. 掌握了分频处理的方法

2. 充分了解了七段显示管的原理，并可以用分时显示使七段显示管显示不同的数值

3. 了解了verilog中模块与模块连接时输入和输出的类型

4. 每一个模块都需要测试！！！

这次LAB相较前两次都比较复杂，尤其是七段显示管，了解了原理之后，具体实践仍有很多需要注意的地方。