**实验报告——自动售货机**

**备注：该文档中一些截图可能不清晰，因此在提交的压缩包里面同时提供了独立于文档的截图，便于助教查看**

**一、需求分析**

下面列出的是针对文档中没有明确规定的需求，并给出我的选择方案：

1. 因为实验要求中并不要求自动售货机有找零功能，因此为了简单起见，在具体实验中，我并没有考虑同时买了两件及以上商品的情况；因为六种商品中最贵的商品为13元，那么我只考虑输入金额最多为12.5+5=17.5，因此对应已经输入的金额数要用三部分表示，第一部分表示小数点之后的部分，第二部分表示整数，因为四个比特最大表示数字为15，因此需要第三部分来表示最高位整数
2. 文档中没有明确规定商品编号和输入金额是用同一个7段数码管显示还是用两个7段数码管分别显示，为了使这个简陋的自动售货机更易于使用，在此次实验中，我的选择是用AN[4]显示选定的商品编号，用AN[2]、AN[1]和AN[0]共同组成了输入金额，其中AN[0]显示的是小数部分，因此整数部分AN[1]对应的小数点要显示，最高位整数AN[2]的小数点不需要显示
3. 因为实验中有六种商品，因此用3个switch表示，那么对应的000和111都是无效的商品编号，对于这两个无效的商品编号我有三个处理，一是当选定这两个无效编号，对应显示商品编号的7段数码管都显示为0；二是当选定这两个无效编号时，不允许输入货币，也就是按下货币输入的四个按钮无效；三是倘若在购买编号为1-6的商品中切换商品编号为0或者7，已经输入的货币不清零
4. 文档里面并没有明确说明当购买一件商品过程中放弃购买时，已经输入的货币是否可以退回，所以我考虑了日常生活中的实际情况，设置了一个reset位，用一个button来控制，当reset被按下时，已经输入的货币清零
5. 文档中并没有提到表示交易完成的等要亮多久，因此我在此次实验中的处理是，当一次交易完成之后，表示此次交易完成的灯到下次交易开始时才会息掉；表示下次交易开始有两种情况：一是改变了商品编号；二是保持当前商品编号不变，重新开始输入货币。因为既改变商品编号，又输入货币按照改变的顺序，一定属于这两种情况的某一种，因此对此种情况，不再分类讨论
6. 通过以上分析，我们可以得出系统的总输入和总输出：
   1. 总输入：时钟、reset、商品编号、当前输入的金额
   2. 总输出：8个七段数码管的显示、表示当前交易是否完成的LED灯的显示

**二、实验设计**

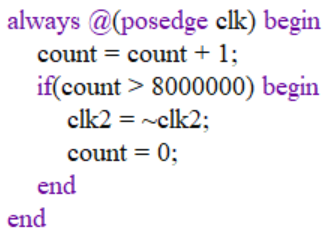
**1.module paid：**

**a)输入：**clk(表示时钟，1-bit)；hasDone(表示此次交易是否完成，1-bit)；reset(表示对已经输入的货币是否清零，1-bit)；No(表示选定的商品编号，3-bit)；money(表示当前输入的货币金额，包括5角、1块、2块和5块，4-bit)

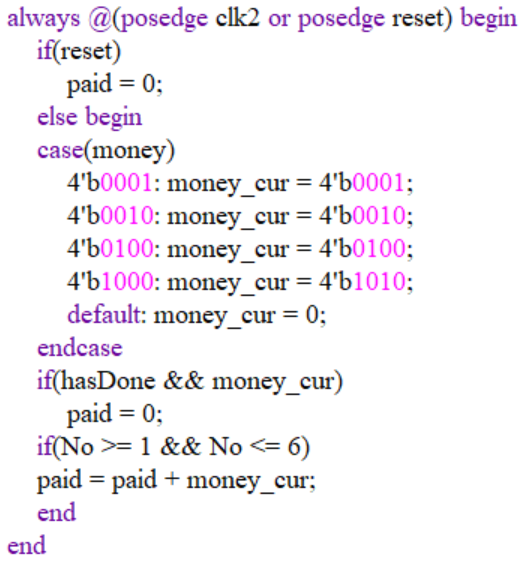
**b)输出：**paid(表示已经输入的货币总额，根据需求分析的第一条，此处选用长度为5-bit)

**c)**该模块实现的功能是，保存当前已经输入的货币总数，并且根据商品编号、当前输入的货币金额、当前交易是否完成、是否将已输入的货币清零以及时钟等输入，对保存的输入金额做相应的改变

**d)防抖动处理：**此处的防抖动处理我利用的是文档中给出的方法，因为我利用的是板子上的E3引脚，对应的频率是100MHz，所以我取count的最大值为8000000，每到clk的时钟上升沿，就将count加一，当count达到最大值时，对clk2取反，这样clk2的时钟周期就降到了12.5Hz，每到clk2的上升沿获取一下按钮状态，就相当于每隔80ms获取一次按钮状态，一定程度上避免了在按钮仍然抖动的时候获取按钮的状态而得到无效的值；代码如下：



**e)关键代码解析：**这一模块除了必要的防抖动处理，另外一部分重要的就是利用低频的clk2和五个按钮（4个表示输入金额，1个表示reset）对保存已输入金额的寄存器中的值做出对应的改变，首先上代码：



①always的敏感列表，除了clk2的时钟上升沿之外，另外一个就是表示reset的上升沿，也就是从0变为1的时候

②在块的内部，首先判断是否是reset触发的，如果是，就将存储已输入金额总数的paid寄存器中的值清零

③如果不是，就首先通过模块输入的money判断当前的输入金额，也就是money\_cur，因为不考虑多个按钮同时按下的情况，因此在default中，多个按钮同时按下，也表示当前输入金额为0；

④之后，就是最关键的地方，如果当前hasDone为1并且money\_cur不为0，也就是输入了新的金额，那么就将paid寄存器清零；这一步的意思是，如果当前交易已经完成了，那么如果接收到了新的非0的货币输入，就认为开始了一次新的交易，那么paid就要清零

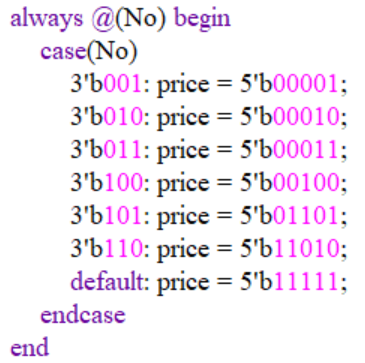
⑤判断No是否有效，也就是是否∈[1,6]，如果在，就对paid加上当前输入金额数，如果不在，就不处理；意味着，当No从一个有效的商品编号切换为0或者7时，已输入金额不会清零，但是此时不能再投入货币

**2.module compDone：**

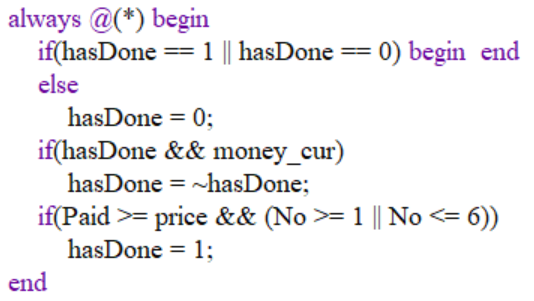
**a)输入：**No(表示当前选定的商品编号，3-bit)，money(表示当前投入的金额，4-bit)，Paid(表示当前已输入的金额总数，6-bit)

**b)输出：**hasDone(表示在已输入金额为Paid的情况下，此次交易是否完成)

**c)**显然，这个模块要通过比较No商品的价格和已输入金额Paid的值判断此次交易是否完成，因此，每当No变化，我们都要获取一下No商品的价格price(5-bit)，便于之后的比较，对于无效的商品编号，取价格为最大，0x1F：



**d)关键代码解析：**该模块的作用就是判断当前交易是否完成，输出一个表示是否完成的hasDone，因此代码实现如下：



①首先第一部分很巧妙，hasDone是没有初始值的，在程序刚开始执行时，hasDone处于不确定状态(x)，为了避免之后取反出问题，我需要在此时对hasDone赋一个初始值0；因为verilog中，不确定值无法直接通过是否等于x而判断，因此，我判断hasDone既不是0，也不是1，就认为此时hasDone的值是不确定的，赋值为0

②因为在需求分析中提到了表示交易已完成的LED灯亮多长时间的问题，在此次实验中我令LED灯一直亮，直到下一次交易开始，对应的代码实现就是当hasDone=1，也就是交易已完成时，只有新的money输入，才认为开始了一次新的交易，改变hasDone为0；对应代码中的money\_cur与paid模块中的意义相同，都是表示当前输入的金额数，保持统一，此处也忽略了多种货币同时输入

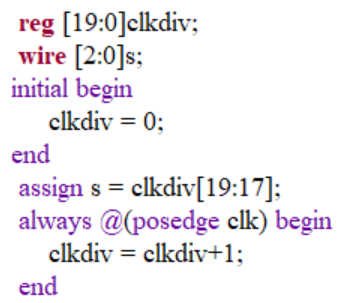
③如果已输入金额数Paid大于price，并且No是有效的商品编号，认为交易完成，hasDone赋值为1

**3.module display：**

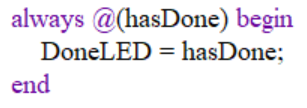
**a)输入：**clk(表示时钟，1-bit)，hasDone(表示交易是否完成，1-bit)，No(表示选定的商品编号，3-bit)，Paid(表示已输入金额数，6-bit)

**b)输出：**DoneLED(代表的是表示交易是否完成的LED，1-bit)，DP(七段数码管对应的小数点，1-bit)，AN(8个七段数码管，8-bit)，A2G(七段数码管的七位，7-bit)

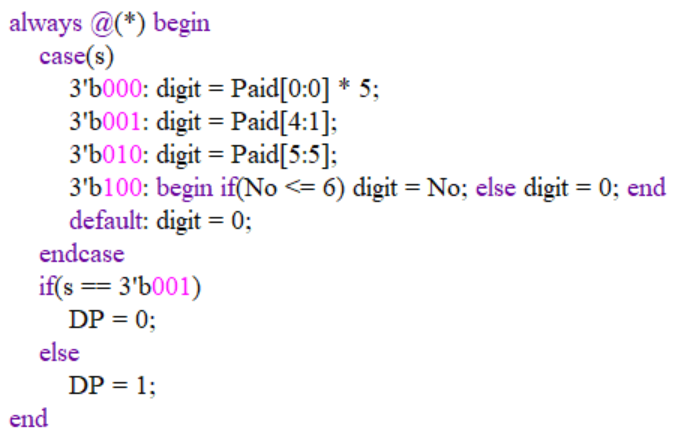
**c)**根据七段数码管的原理，此处也要对八个七段数码管做分时显示，和LAB3做相同的处理，因为此次需要显示的七段数码管有8个（除需求分析第二条提到的四个，其余全是0），所以s取3-bit：



**d)用DoneLED显示交易是否完成**，每次hasDone变化，就对DoneLED重新按照hasDone的状态赋值：



**e)获取每一个七段数码管的显示数值**，根据s判断：



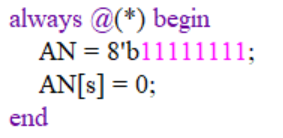
①当s=0时，显示的小数部分。digit表示七段数码管显示的数值，因为此处是定点小数，也就是小数点固定的小数，因此，我们约定Paid的第0位总是表示输入金额的小数部分，为1表示5角，为0表示输入金额恰好为整数，那么对应显示小数位时，要对该值乘5得到实际表示的金额数来显示

②当s=1时，显示的是低四位的整数部分，s=2时显示已输入金额的最高位整数部分

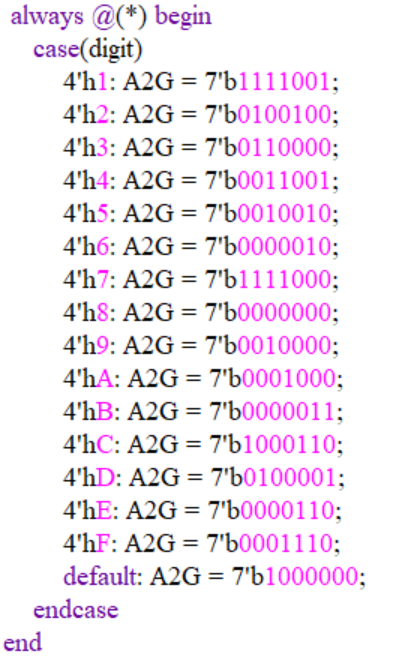
③当s=4时，显示商品编号，对于无效的商品编号0和7，统一显示为0，这也就是代码中对No是否在[1,6]范围中做分别处理的意义

④小数点，只有s=1时，对应位才显示小数点

**f)获取当前显示的是哪一个七段数码管**，也是根据s判断的，此处应格外注意是低电平有效：



**g)对应每一个七段数码管应该如何显示**，在前两步得到当前应该显示哪一个七段数码管和对应要显示的内容之后，就应该处理如何显示的问题了，此处与上次LAB相同，都是根据digit的数值选择a-g七段分别是否显示：



**4.module VEM：**

**a)输入：**clk(表示时钟，1-bit，对应E3引脚)，

reset(表示已输入金额是否清零，1-bit，对应一个按钮)，

No(表示当前选定的商品编号，3-bit，对应3个switch)，

money(表示当前输入的金额，4-bit，对应4个button)

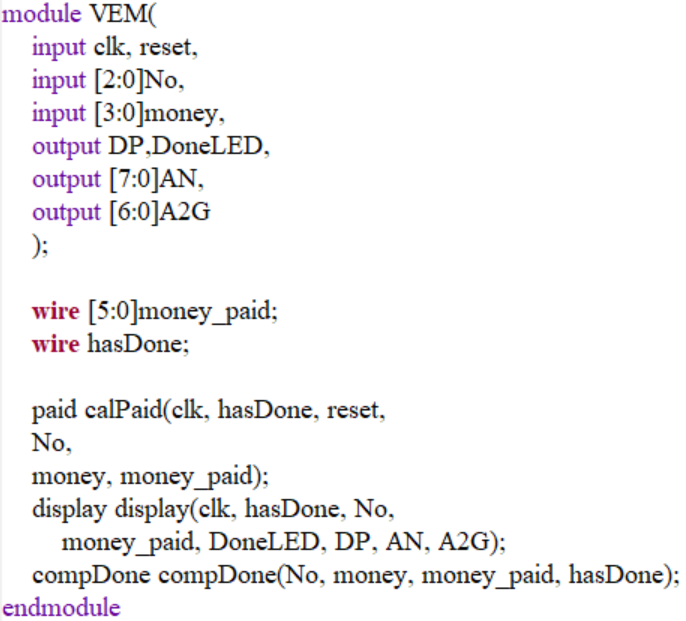
**b)输出：**DoneLED(显示当前交易是否完成，1-bit，对应一个LED灯)，

DP(表示小数点，1-bit)，

AN(对应8个七段数码管，8-bit)，

A2G(表示七段数码管的数值如何显示，7-bit)

**c)**该模块是顶层模块，调用其他三个模块，形成一个自动售货机的系统：



**三、仿真波形**

**1.module paid：**

**a)测试用例：**根据需求分析，在此次实验中，测试了一下7中情况：

①商品编号无效，但是money有效🡪paid = 0

②商品编号为4，money = 1 🡪 paid = 1

③商品编号从4变为2，没有money输入 🡪 paid保持1

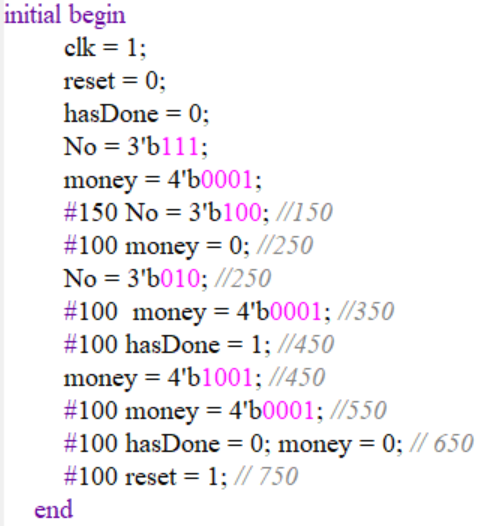
④保持商品编号为2不变，money = 1 🡪 paid = 2，之后hasDone从0变为1

⑤在④已经完成一次交易的基础上，保持商品编号不变，输入无效的money，paid不会清零

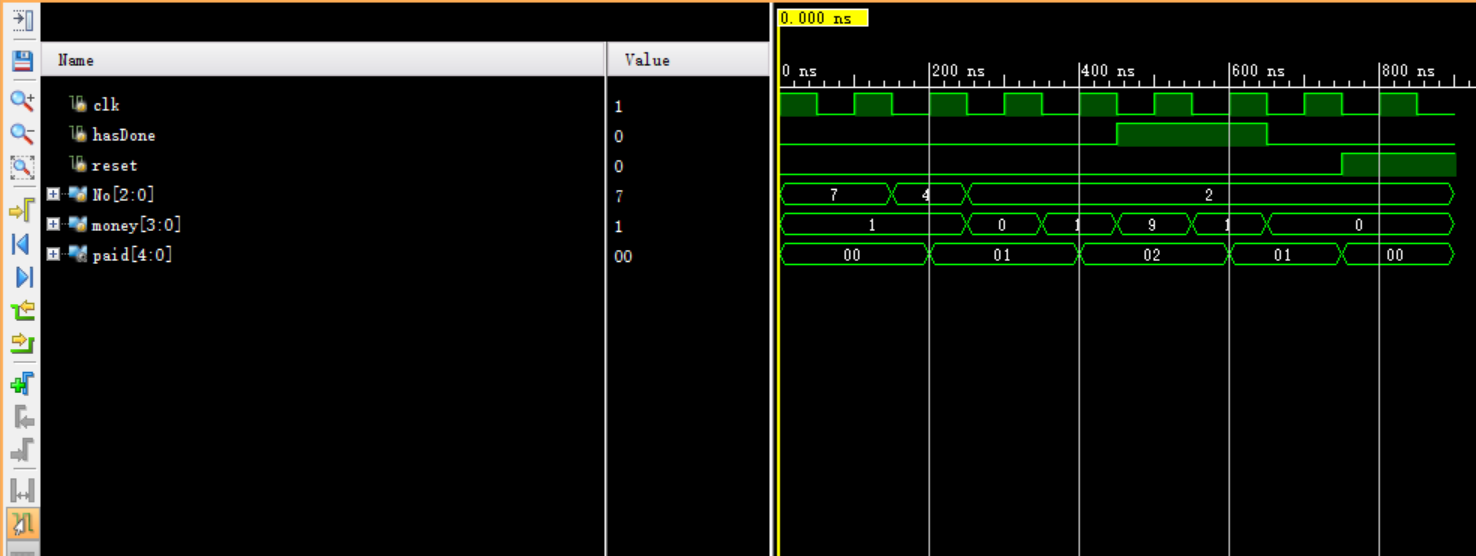
⑥保持商品编号不变，money = 1 🡪 paid清零之后变为1

⑦测试reset，将paid手动清零 🡪 paid = 0

**b)测试代码：**

****

**c)仿真波形：**

****

**2.module compDone：**

**a)测试用例：**该模块实现的功能是计算hasDone，判断当前交易是否完成，因此对于商品编号为4的情况，price=2测了一下四种情况：

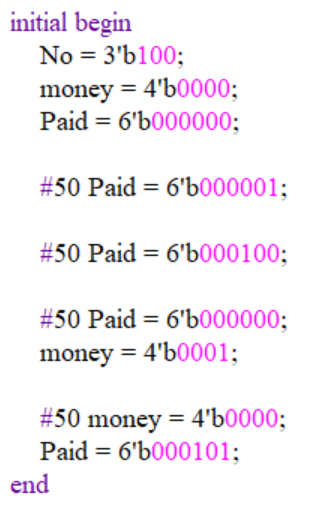
①已输入金额为5角 🡪 hasDone = 0

②已输入金额为2元 🡪 hasDone = 1

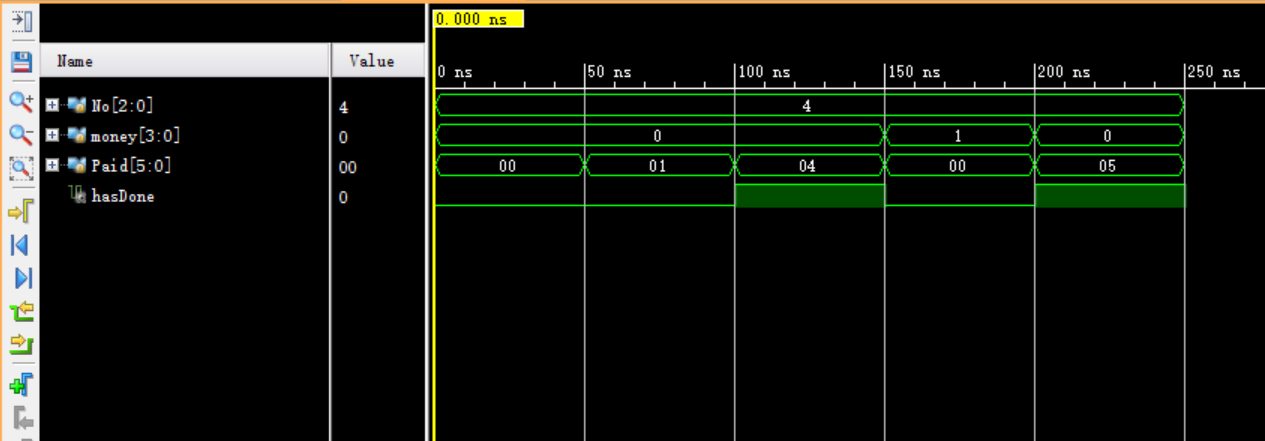
③在②交易已完成的情况下，测试输入新的money，hasDone的变化 🡪 hasDone = 0

④超额支付，money = 0， 已输入金额为2.5元 🡪 hasDone = 1

**b)测试代码：**

****

**c)仿真波形：**

****

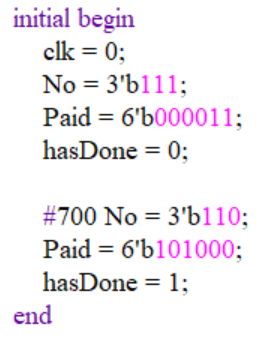
**3.module display：**

**a)测试用例：**该模块实现的是显示商品编号，已输入金额和交易完成状态，因此测了一下两种状态：

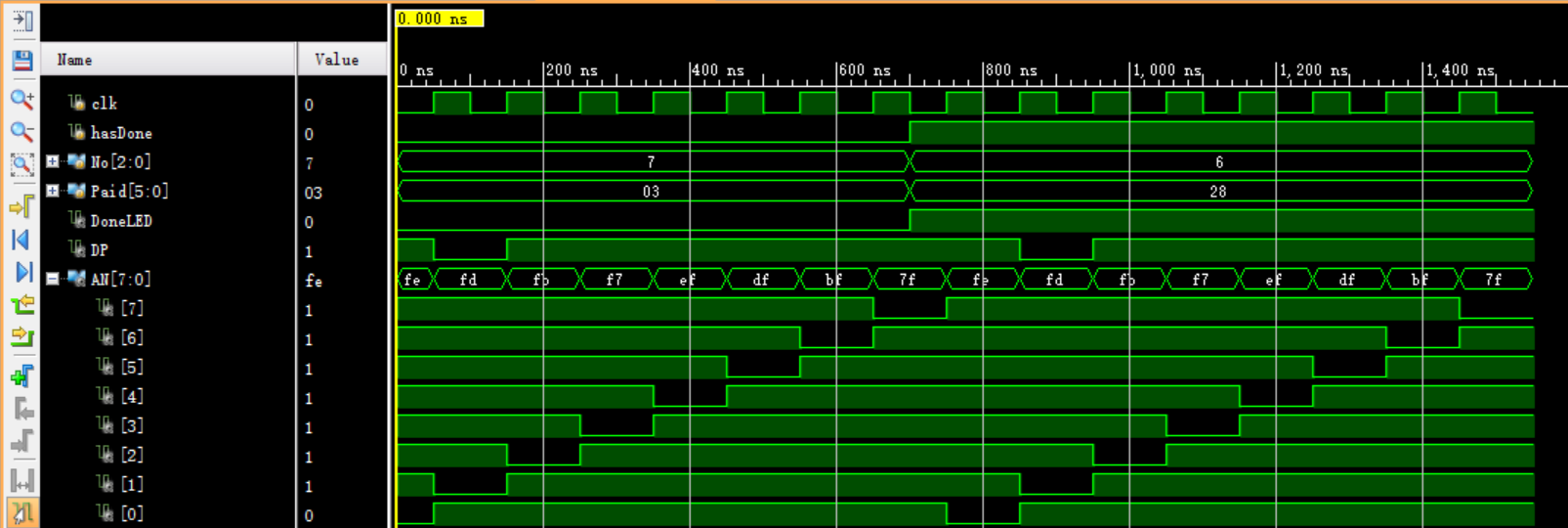
①商品编号为7，已输入金额为6’b000011，hasDone=0

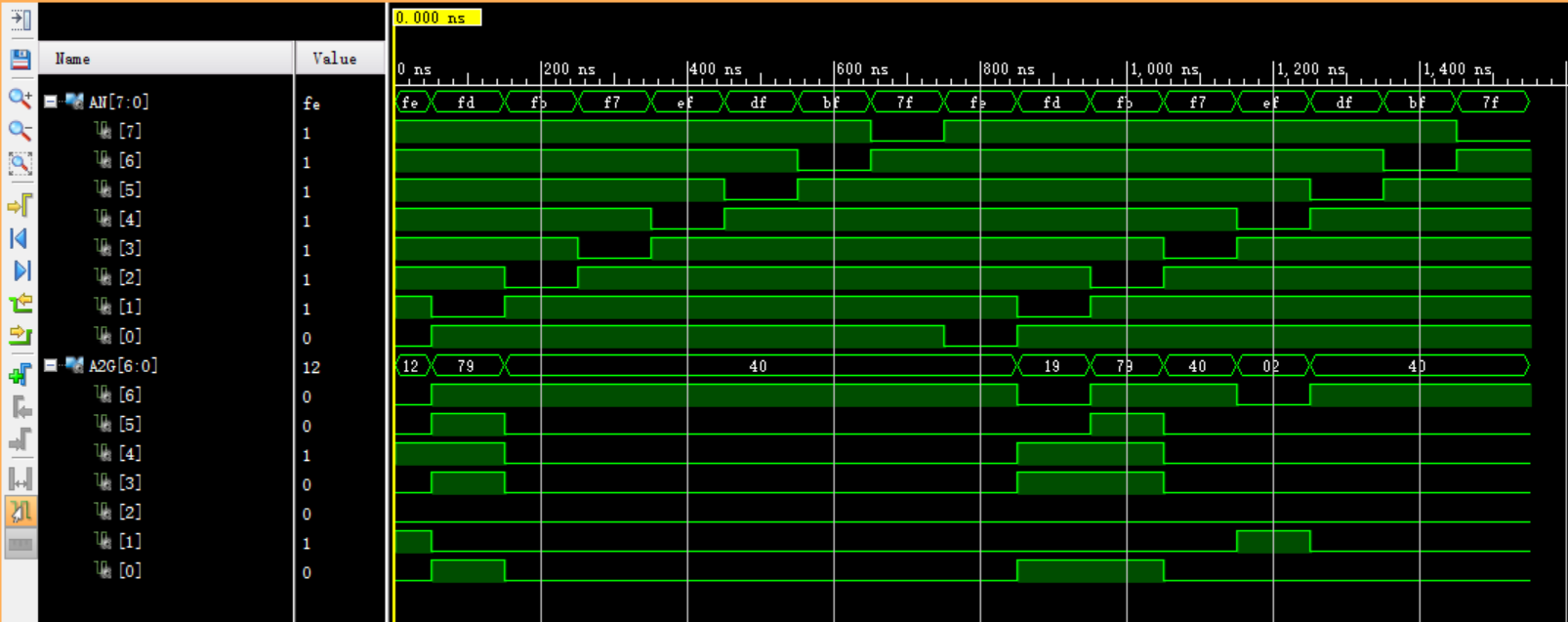
②商品编号为6，已输入金额为6’b101000，hasDone=1

**b)测试代码：**

****

**c)仿真波形：**

****

****

**4.module VEM：**

**a)测试用例：**该测试模拟了一个真实的可能场景，流程如下：

①商品编号选定为7，money为5角

②保持money为5角，商品编号由7变为4，money变为1元

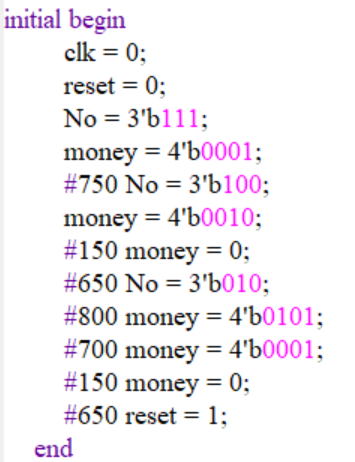
③商品编号由4变为2，此时交易完成

④在交易完成的基础上，再投入无效的money=4’b0101

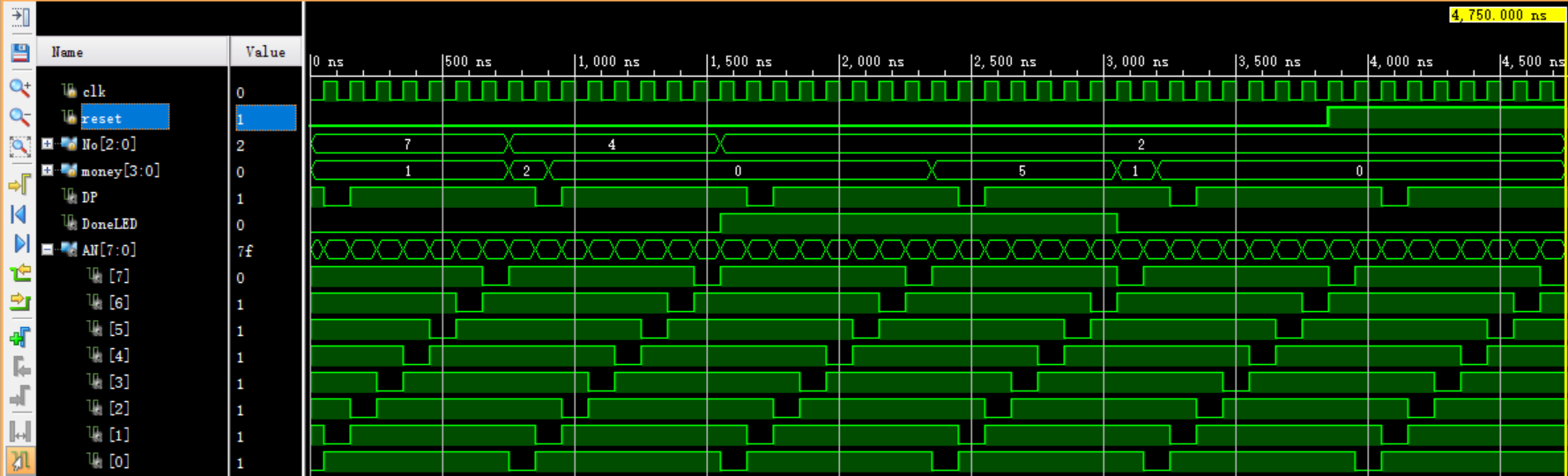
⑤投入有效的money为5角

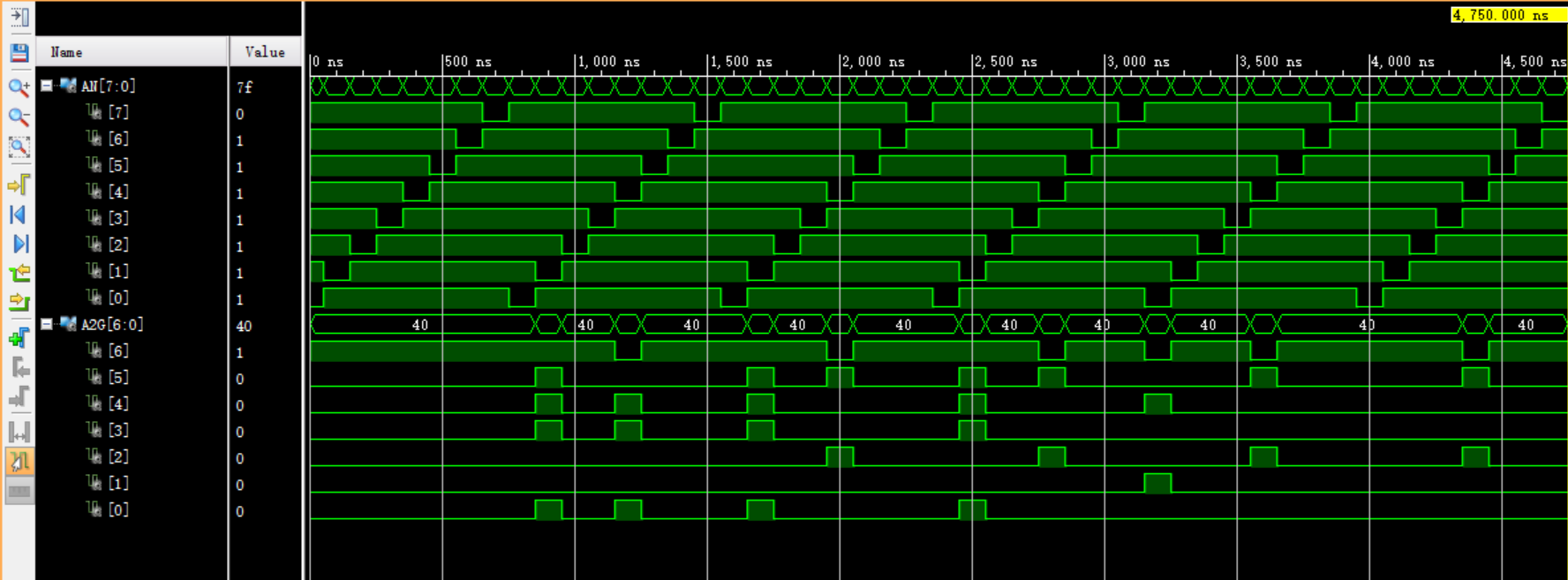
⑥reset

**b)测试代码：**

****

**c)仿真波形：**

****

****

**四、实验中遇到的问题及解决思路**

1.本次实验中，最重要的一个问题就是需求分析，因为不像之前的三次LAB，文档中直接体现了整体的电路图关系和总输入、总输出，其中的很多细节在文档中也没有明确给出，因此我参考了老师的意见，并且联系实际生活，在实验中给出我的处理

2.hasDone的初始值问题，刚开始我并没有对hasDone赋初始值，下板测试的时候，是正确的，但是，运行测试代码，查看仿真波形，发现hasDone在刚开一段时间内都是不确定状态，导致后面也出现了一小段异常，因此在compDone中添加了对hasDone赋初始值的代码

3.金额的小数部分显示问题，因为此次输入金额可能会包含一个0.5的小数部分，因此考虑到是定点小数，我对他们做了统一的处理，无论是money\_cur还是Paid，最后一位的1都表示是5角

**五、实验中的收获与感受**

整体感受是本次LAB较为简单，大部分功能都是之前LAB中已经实现的，比如：分频处理与上次Q1的显示中的分频处理原理相同，七段数码管的显示也已经在上次LAB中实践过了，此次LAB只不过是按照新的处理逻辑做了综合，并不复杂。