# Moore模型自动售货机

### **题目要求**

一杯饮料1.5元，每次只能投入0.5元或1元的硬币，可以找零，要求能自启动。

### 设计方案

1. 确定模型：为了防止输出受到输入扰动的干扰，应选择Moore模型
2. 变量：

输入变量：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X1X0 | 00 | 01 | 11 | 10 |
| 含义 | 不投入硬币 | 投入0.5元 | 禁止输入 | 投入1元 |

状态和输出

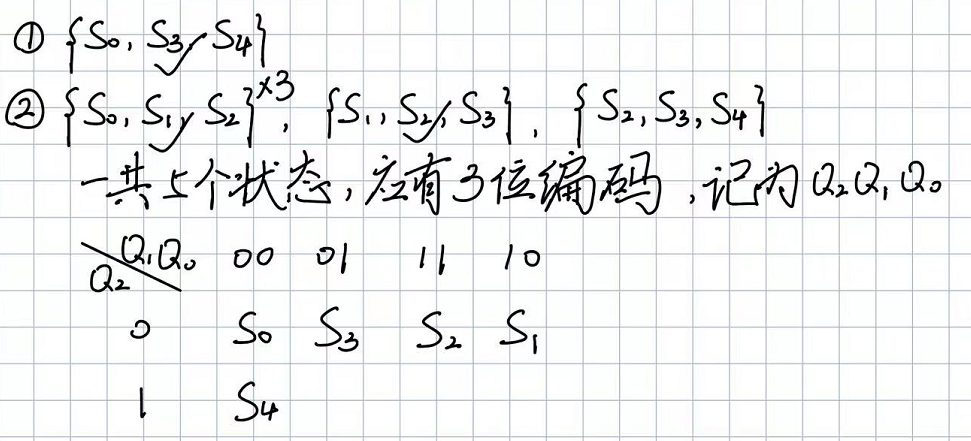
|  |  |  |  |
| --- | --- | --- | --- |
| 状态 | 含义 | 输出Z1Z2 | 含义 |
| S0 | 初始状态，收了0元 | 00 | 无响应 |
| S1 | 收了0.5元 | 00 | 无响应 |
| S2 | 收了1元 | 00 | 无响应 |
| S3 | 收了1.5元 | 10 | 输出一杯饮料 |
| S4 | 收了2元 | 11 | 输出一杯饮料，并找零（0.5元） |

1. 状态转换表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 现态 | 次态 | | | | 输出Z1Z2 |
| X1X0=00 | X1X0=01 | X1X0=11 | X1X0=10 |
| S0 | S0 | S1 | d | S2 | 00 |
| S1 | S1 | S2 | d | S3 | 00 |
| S2 | S2 | S3 | d | S4 | 00 |
| S3 | S0 | S1 | d | S2 | 10 |
| S4 | S0 | S1 | d | S2 | 11 |

状态编码：

根据教材建议的状态编码规则，编码如下：



### 设计思想和代码说明

1. 三段式有限状态机FSM对应代码为Drink\_machine.v，二段式为Drink\_machine2.v。

为了防止一个硬币无限白嫖（按键不松开，对应现实情况为硬币卡在了检测口）的情况发生，我给输入X和次态next\_state分别做了一个寄存器，输出分别为pre\_X和pre\_next\_state。当X和pre\_X相同时，说明输入没有变化，对应的暂态next\_state也就和pre\_next\_state相同。

1. 三段式顶层文件对应代码为Drink\_machine\_top.v，二段式为Drink\_machine\_top2.v
2. 时钟分频module对应代码为Clkdiv.v，生成三个时钟：

CP\_2ms：用于数码管的频闪

CP\_20ms：功能有二。

1. 查阅相关资料得知，按键抖动时间不超过20ms，所以以20ms为周期的时钟可以用于消抖。
2. 作为FSM状态转移的触发信号。因为光按键消抖就得20ms，所以FSM状态更新的时间周期不能太短，否则X和next\_state可能会出现亚稳态；但也不能太长，否则一旦用户比较急，投币速度快，就会出现吞币的情况。综合考量后，将FSM的时钟周期与获取输入的时钟周期同步起来最佳。
3. 按键消抖module对应代码为Key\_dejetter.v
4. 七段数码管module对应代码为x7seg.v，主要功能是显示输出和现态。

左边两个管子显示输出Z：输出饮料显示10，输出饮料并找零显示11。

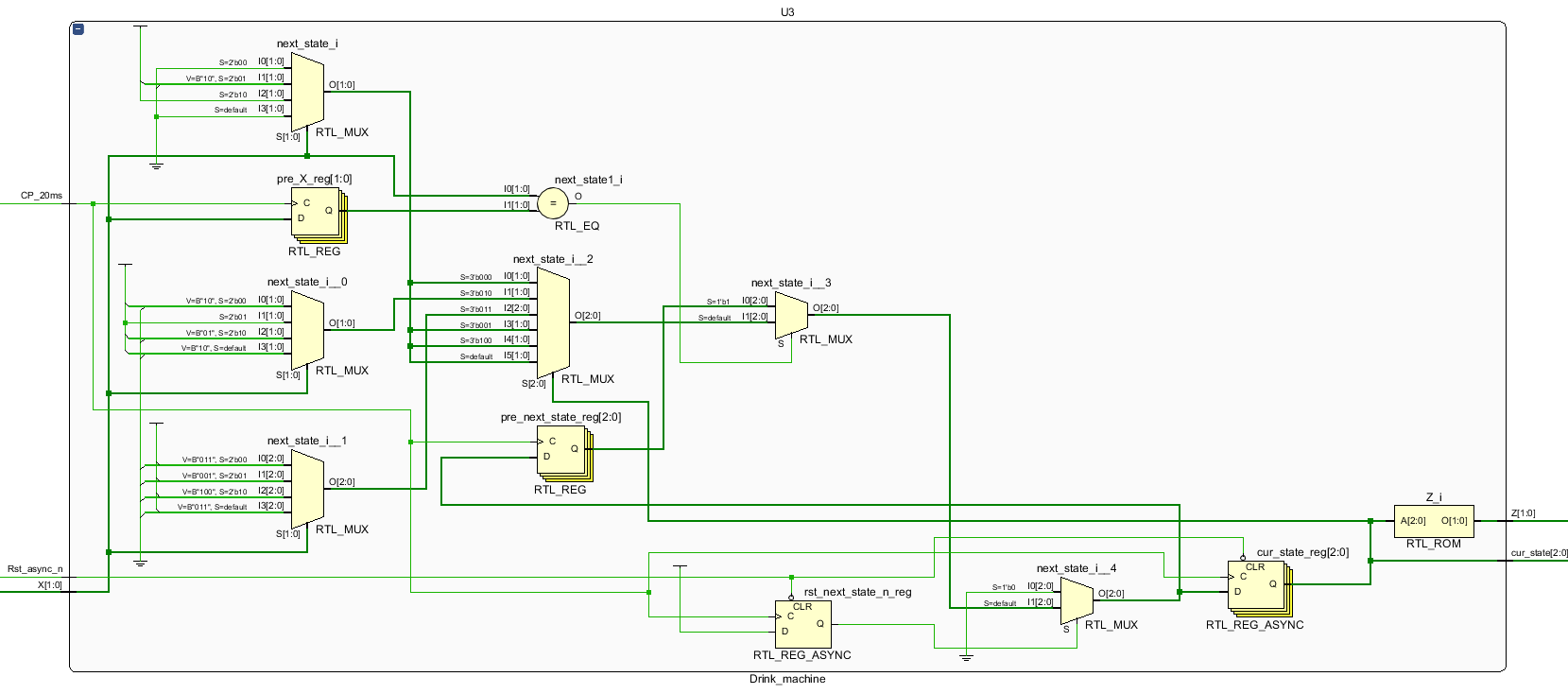
右边两个管子显示现态：已收0.5元显示05，已收1元显示10，已收1.5元显示15，已收2元显示20。

### 细节记录

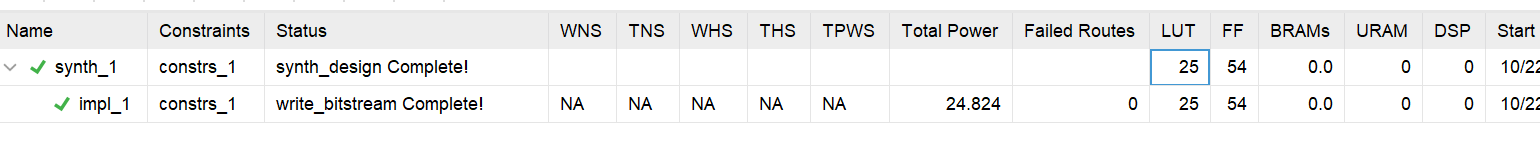
1. 强化赋初值意识，可以减少很多意外bug
2. 不能在不同的always块里给同一个变量赋值
3. If-else和case语句都要罗列完整，操作对象保持不变也不能直接else，而是要利用寄存器，使其等于上一个状态，否则会生成锁存器，发生意外bug。
4. Moore状态机由于状态多，其激励方程（状态转移规律）会比Mealy模型复杂，但输出方程比较简单。
5. 对于按键状态机，触发信号周期和按键消抖所用的时钟周期最好保持相同。

### 三段式和二段式比较

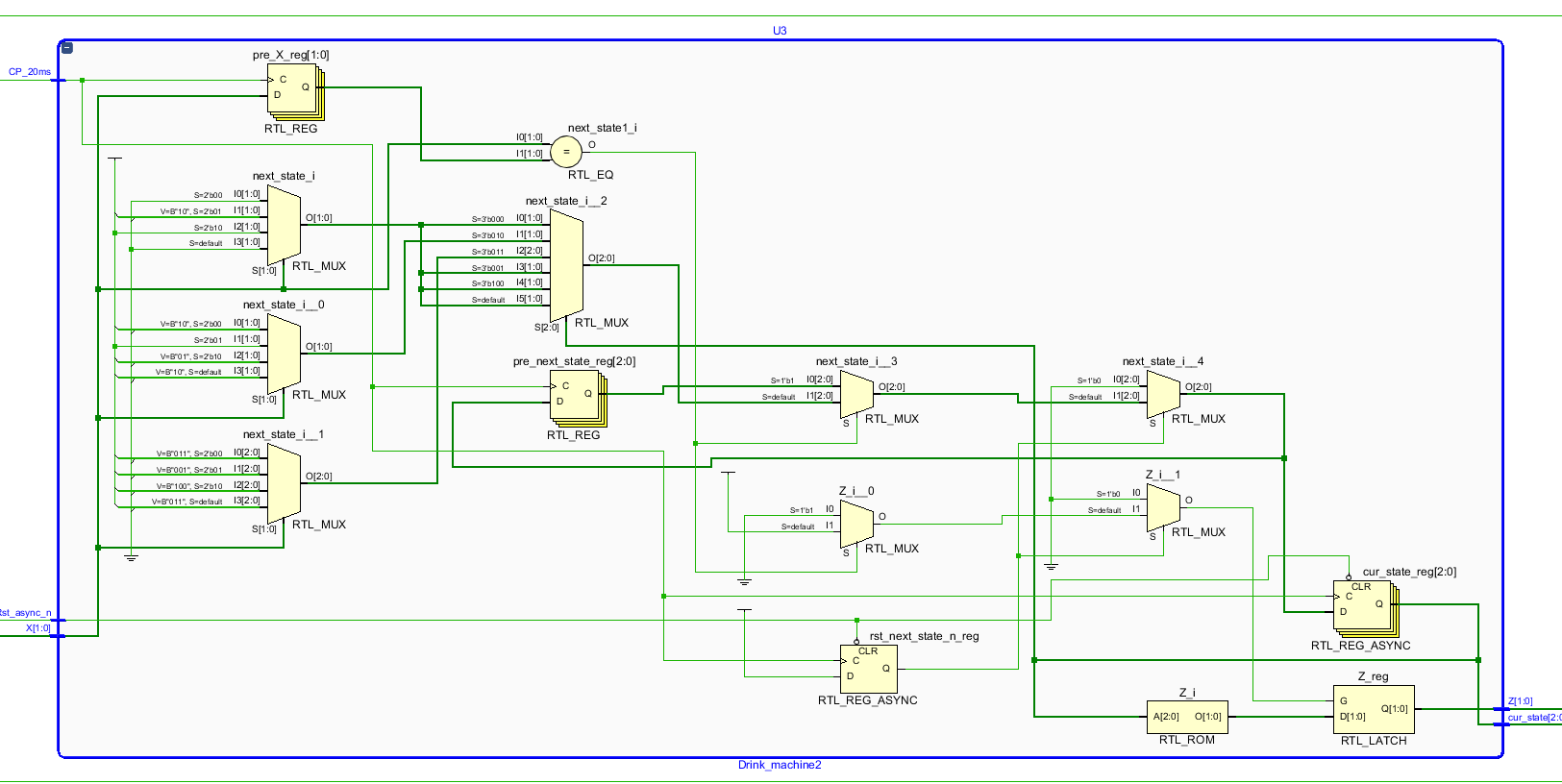
三段式电路图如下：



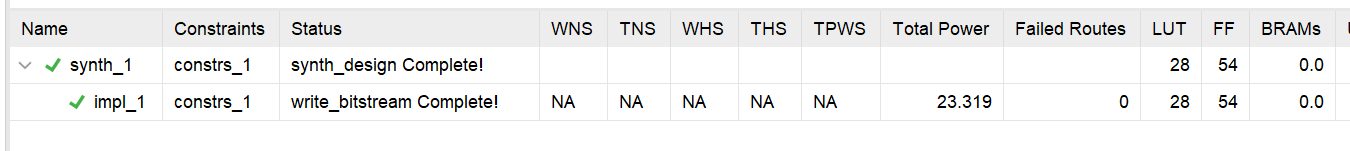
占用资源情况如下：



二段式电路图如下：



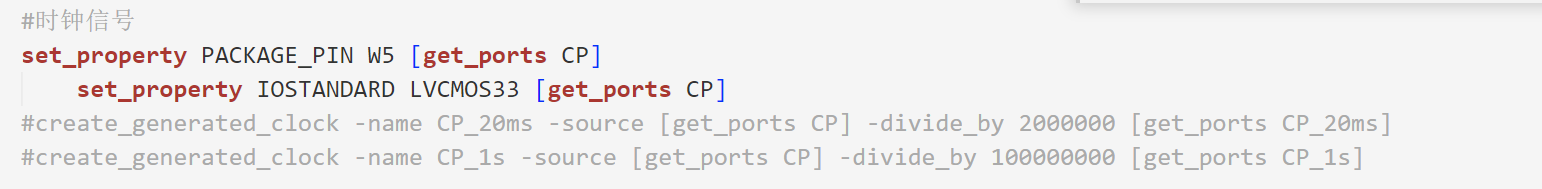
占用资源情况如下



比较可以得出，三段式的电路较简单，所占用资源较少。

### 问题

1. 上节课助教说用户自己分频得到的时钟应该在xdc文件里声明，我照做了之后报错，不知为何。对时钟进行分频是否需要我们专门设计一个module？还是说直接在引脚文件里生成即可？希望听助教详细讲解qwqqq



1. 在数电基础中，我们会对一个FSM进行充分的化简（状态编码、任意项处理、冗余态自启动方式选择等等），但这种化简相当繁琐，而且化简后反而增加了硬件描述语言的代码量。我因为本周事务繁忙，便只是对状态编码方案进行了筛选，并没有完成化简的全过程。是不是任何时候都应该充分化简后再上机呢？
2. 一组数据，如[1:0]X，当我们写posedge X的时候，是否要求X[1]和X[0]都出现posedge，X才算出现posedge？还是两者满足其一即可？