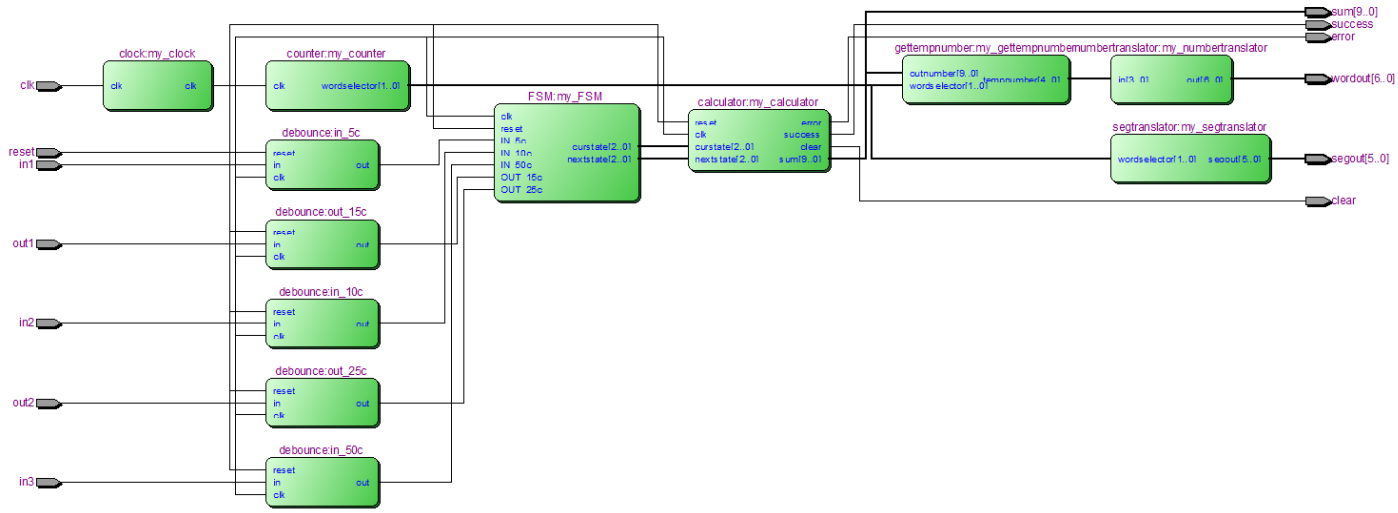
EDA2 实验报告

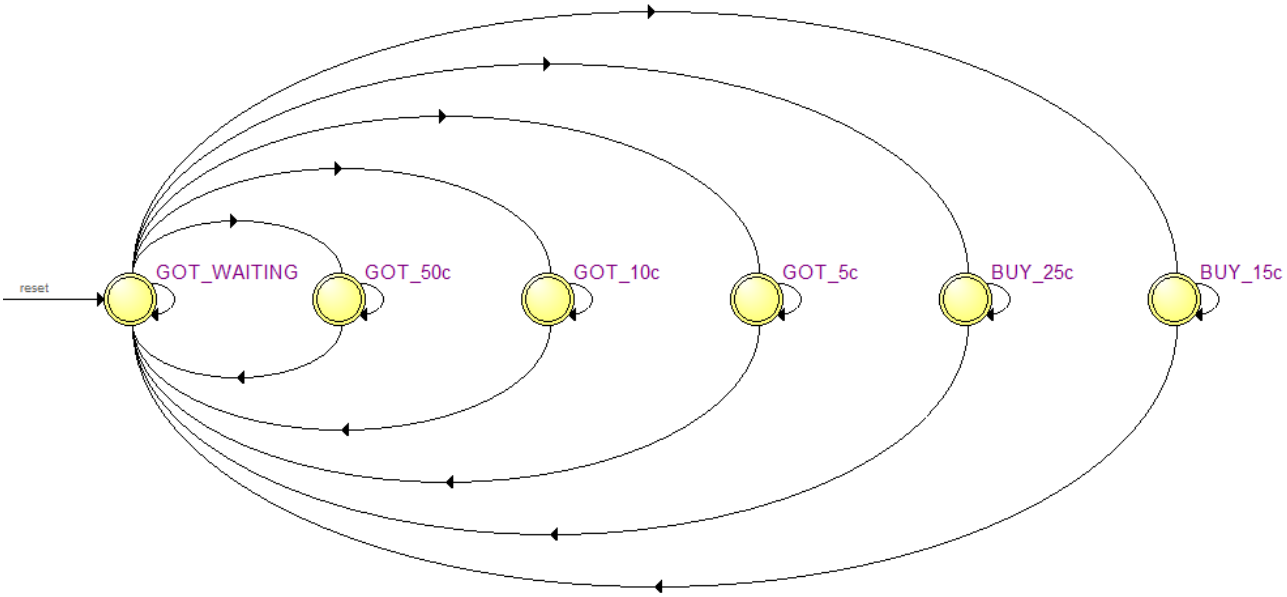
徐浩博 2020010108 软件 02

一、RTL 图

整体电路结构图如下：



二 状态转换图与说明



【GOT_WAITING】　基础状态，等待投币或者购买信号

当收到投入 5 元信号 IN_5c 时，跳入 GOT_5c 状态

当收到投入 10 元信号 IN_10c 时，跳入 GOT_10c 状态

当收到投入 50 元信号 IN_50c 时，跳入 GOT_50c 状态

当收到购买 15 元商品信号 GOT_15c 时，跳入 BUY_15c 状态

当收到购买 25 元商品信号 GOT_25c 时，跳入 BUY_25c 状态
当没有收到任何信号时，仍然保持 GOT_WAITING 状态

【GOT_5c】收到投入 5 元信号 IN_5c 后的状态
当收到投入 5 元信号 IN_5c 时，保持 GOT_5c 状态
当投入 5 元信号 IN_5c 消失时，跳回 GOT_WAITING 状态

【GOT_5c】收到投入 10 元信号 IN_10c 后的状态
当收到投入 10 元信号 IN_10c 时，保持 GOT_10c 状态
当投入 10 元信号 IN_10c 消失时，跳回 GOT_WAITING 状态

【GOT_50c】收到投入 50 元信号 IN_50c 后的状态
当收到投入 50 元信号 IN_50c 时，保持 GOT_50c 状态
当投入 50 元信号 IN_50c 消失时，跳回 GOT_WAITING 状态

【BUY_15c】收到购买 15 元商品信号 BUY_15c 后的状态
当收到购买 15 元商品信号 BUY_15c 时，保持 BUY_15c 状态
当购买 15 元商品信号 BUY_15c 消失时，跳回 GOT_WAITING 状态

【BUY_25c】 收到购买 25 元商品信号 BUY_25c 后的状态
当收到购买 25 元商品信号 BUY_25c 时，保持 BUY_25c 状态
当购买 25 元商品信号 BUY_25c 消失时，跳回 GOT_WAITING 状态

三、各模块功能及实现方法

【分频模块】该部分由一个部分组成

clock:

功能：将高频率的晶振信号分频至适合电路工作的低频时钟信号

引脚：

input clk：40MHz 振动频率，接 FPGA 板时钟信号引脚

output clk_：250Hz 晶振，接 FPGA 后续所有模块

实现方法：count 每遇到一个 clk 上升沿则加一，count 的高位即可分频

【消抖模块】该部分由一个部分组成

debounce：

功能：对于输入的按键和拨码开关信号进行消抖

引脚：

input clk：时钟信号

input reset：重置信号

input in：需要消抖的信号

output out：消抖后的信号

实现方法：采用计数器的方法。时钟上升沿时比较上一次上升沿的输入信号与此次的输入信号，如果不一致则重新计数，一致则计数器加一；计数器累加到一定值，则说明输入信号在一段时间内已经稳定，此时可以将输入信号输出。

【状态机模块（核心模块）】该部分由两个部分组成

FSM：

功能：有限状态机，处理输入，并实现状态转换

引脚：

input clk：时钟信号

input reset：重置信号
input IN_5c：投入五元的信号
input IN_10c：投入十元的信号
input IN_50c：投入五十元的信号
input OUT_15c：购买十五元商品的信号
input OUT_25c：购买二十五元商品的信号
output curstate：当前状态机状态
output nextstate：状态机下一个状态

实现方法：列出状态机的各个状态（如前所述），写出状态方程、驱动方程和输出方程，通过对当前状态和输入信号的讨论完成状态转移。

calculator：

功能：通过状态机状态进行余额计算并给出提醒信号

引脚：

input clk：时钟信号
input reset：重置信号
input curstate：当前状态机状态
input nextstate：状态机下一个状态
output success：购买成功，提醒购买者取走商品
output error：购买失败，余额不足提醒
output clear：退币，提醒购买者取走余额
output sum：售货机内余额

实现方法：通过对于状态机当前状态和下一个状态的讨论，完成相应的余额计算和提醒信号的任务，其中提醒信号安排了一个计数器，当提醒亮起并持续 100 个时钟上升沿时，提醒熄灭。

【显示模块】该部分由四个部分组成

counter：

功能：告知该哪一个数码管显示数字

引脚：

input clk：时钟信号
output wordselector：表示该第几个数码管显示数字

实现方法：两位二进制数计数器，每一个时钟信号上升沿计数器加一

gettempnumber：

功能：获得当前数码管需要显示的十进制数

引脚：

input outnumber：需要显示的最终结果（三位十进制数）
input wordselector：该第几个数码管显示数字
output tempnumber：当前数码管需要显示的十进制数

实现方法：利用 case 分支语句，对 wordselector 讨论以通过 outnumber 计算 tempnumber

numbertranslator：

功能：七段式数码管译码器

引脚：

input in：当前数码管该显示的十进制数
output out：数字对应的七段式数码管每一段的高低电平

实现方法：对 0-9 九个数字进行讨论，依次给出数码管每一段高低电平

segtranslator：

功能：获得位选信号

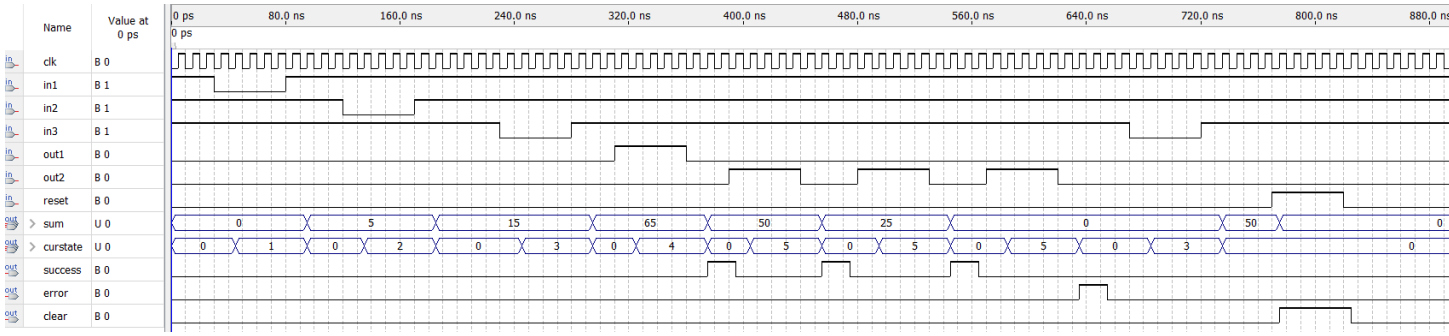
引脚：

input wordselector：该第几位数码管显示

output segout：位选信号

实现方法：对 wordselector 进行讨论，对于每一个数码管显示的情况均给出对应的位选信号

四 波形仿真图与说明



in1-in3 分别代表投币 5、10、50 元，以低电平为有效电平；out1、out2 分别代表购买 15、25 元商品，以高电平为有效电平。reset 是重置清零信号。sum 是余额，success 是取走货物信号，error 是余额不足信号，clear 是取走退币信号。

以上是一个购买的完整流程，我们以此来验证状态机以及余额计算的正确性。应该注意，仿真时为了体现出效果，我们将分频电路和消抖电路计数器的常数进行了更改。

1）in1 输入低电平，代表投币 5 元，经过若干时钟周期后 state 变为 GOT_5c(1)，当投币结束后 state 重回 GOT_WAITING(0)，sum 余额也变为 5 元

2）in2 输入低电平，代表投币 10 元，经过若干时钟周期后 state 变为 GOT_10c(2)，当投币结束后 state 重回 GOT_WAITING(0)，sum 余额也变为 15 元

3）in3 输入低电平，代表投币 50 元，经过若干时钟周期后 state 变为 GOT_50c(3)，当投币结束后 state 重回 GOT_WAITING(0)，sum 余额也变为 65 元

4）out1 输入高电平，代表购买 15 元商品，经过若干时钟周期后 state 变为 BUY_15c(4)，当购买结束后 state 重回 GOT_WAITING(0)，sum 余额也变为 50 元，并且给出了 success 提示

5）out2 输入高电平，代表购买 25 元商品，经过若干时钟周期后 state 变为 BUY_25c(5)，当购买结束后 state 重回 GOT_WAITING(0)，sum 余额也变为 25 元，并且给出了 success 提示

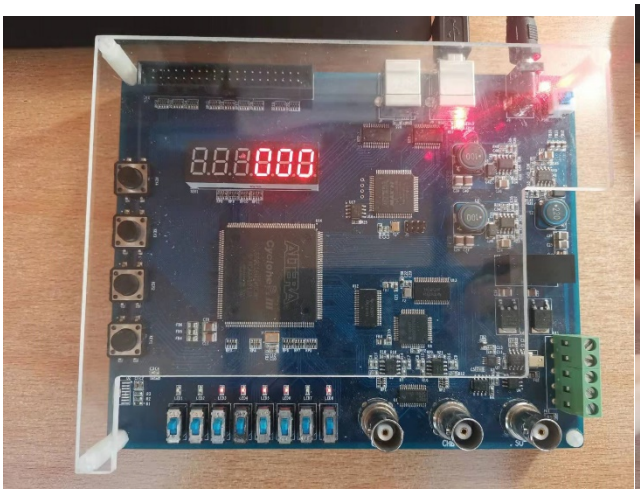
6）out2 输入高电平，代表购买 25 元商品，经过若干时钟周期后 state 变为 BUY_25c(5)，当购买结束后 state 重回 GOT_WAITING(0)，sum 余额也变为 0 元，并且给出了 success 提示

7）out2 输入高电平，代表购买 25 元商品，经过若干时钟周期后 state 变为 BUY_25c(5)，当购买结束后 state 重回 GOT_WAITING(0)，然而余额不足，给出了 error 提示

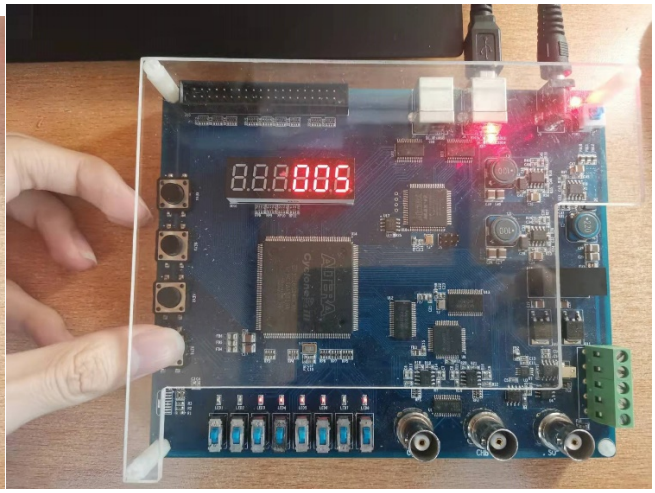
8）in3 输入低电平，代表投币 50 元，经过若干时钟周期后 state 变为 GOT_50c(3)，当投币结束后 state 重回 GOT_WAITING(0)，sum 余额也变为 50 元

9）reset 输入低电平，代表清零，此时 state 重回 GOT_WAITING(0)，sum 余额也变为 0 元

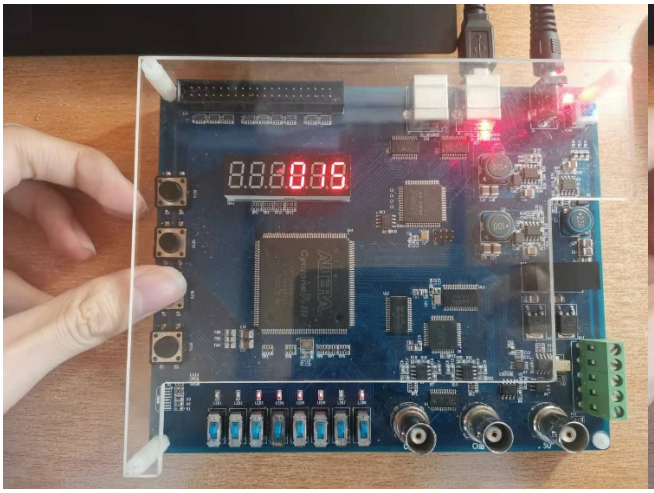
五 效果展示



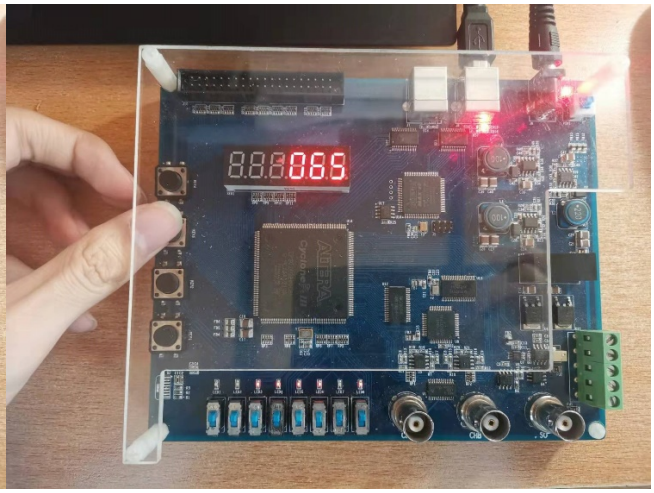
1）初态



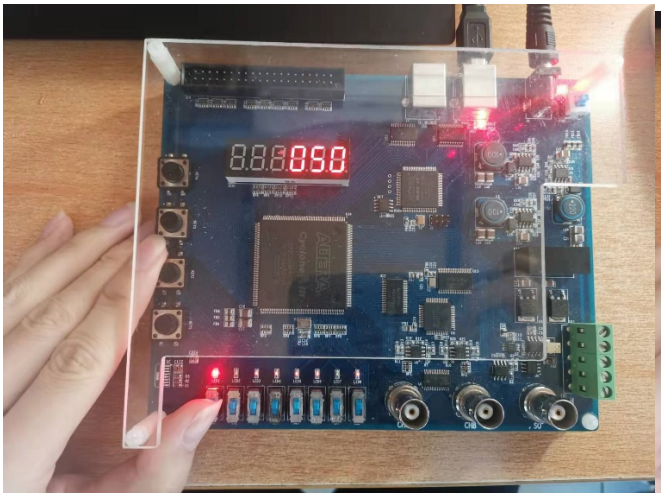
2）投币 5 元



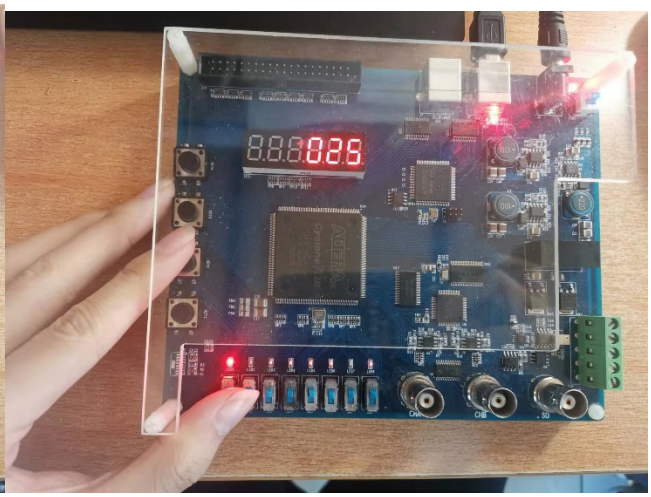
3）投币 10 元



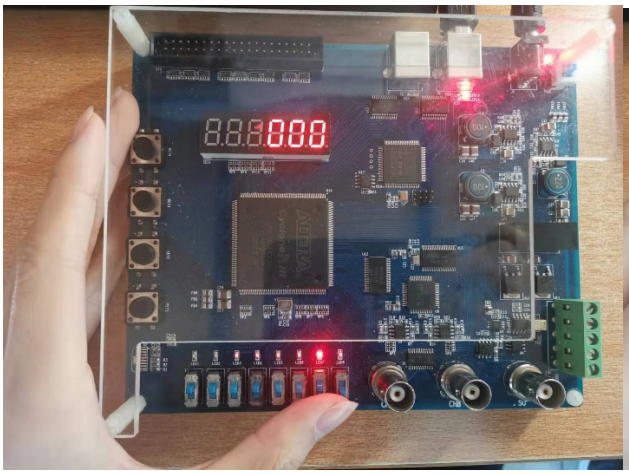
4）投币 50 元



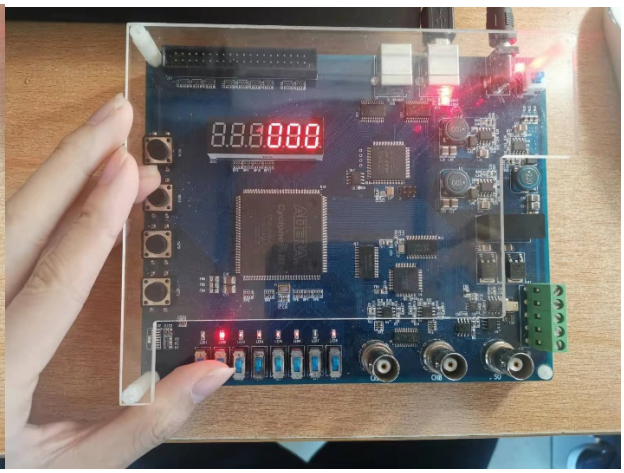
5）购买 15 元商品，给出取货提示



6）购买 25 元商品，给出取货提示



7）退币，给出取币提示



8）购买 25 元商品，报错

六 实验中遇到的问题与解决方案

1. 在写有限状态机时，我将阻塞赋值和非阻塞赋值搞混，导致状态出错：

我通过翻阅老师的讲义和上网查找相关资料，搞清了阻塞赋值和非阻塞赋值的含义：

1）阻塞赋值用等号（=），一般对应电路中的组合逻辑赋值，等号右端的结果会立刻赋值给左端。

2）非阻塞赋值用小于等于号（<=）表示，一般对应电路中的时序逻辑赋值，等号右端的结果不会立刻赋值给左端。

3）在 always 语句中，阻塞赋值等号左端的参数如果参与该模块的其他运算，则按照赋值后的结果参与运算，而非阻塞赋值等号左端的参数依旧按照未赋值前的结果参与运算。

通过以上内容的重新理解，我将阻塞赋值和非阻塞赋值重新调整，并获得了正确答案

2. 初始值初始化不会写

经查阅资料，我发现初始化可以在声明寄存器时一并赋初值，也可以使用 initial 语句块进行初值赋值。

同时，也可以采用老师上课传授的方法，在更新状态的 always 语句块中书写，这种方法比较考验书写的技巧，不如采用前一种方法方便。

3. 仿真时无法仿出正常波形

经过仔细检查，发现消抖电路要经过数十个波形才能给出输出，且分频电路也需要计数器计数参与，而 clk 间隔时间是有下限的，时间过短会导致仿真时程序卡死。因此我们需要检验状态机正确性时，适当调整参数，使得信号经过几个周期就可输入，同时分频电路也无需将分频倍数设定过高，则就可以仿真出正确的波形了。

4. output 传递的信号出错

仿真后发现 output 传回顶层电路的值不正确，经过排查 warning 发现，顶层电路中并未声明向量，而是将它当做了普通的导线，因此只能容纳 0/1，而一个多位向量传递给它当然会出现错误，这提醒我们一定要认真排查 warning 中的提示，不能有马虎！

5. 缺少模块化思维

最开始我在主函数中通过assign语句和always语句接线，但导致的后果是代码写到一半，逻辑已经异常

混乱了；于是我自顶而下开始重新设计，将需要实现的功能一个一个分类并且划分成模块，然后再自底向上实现小模块并在主函数中将各个模块组装，由此写出的代码异常清晰且非常利于维护。