**数字电路与逻辑设计**

**课程设计报告**

|  |  |  |
| --- | --- | --- |
| **课程设计实验部分：实验完成情况、时间**  **（亮点、完成、部分完成）** | | **总分**  **（实验部分70% +报告30%）** |
| **检查结果** |  |  |
| **检查名次** |  |
| **检查老师** |  |

**报告人 ： \*\*\*\*\***

**实验指导教师： \*\*\*\*\*\***

**报告批阅教师：**

**计算机科学与技术学院**

**2017 年 10 月 17 日**

**数字逻辑课程设计学生工作表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **班 级** | **姓 名** | **学 号** | | **验收时间（教师填写）** |
|  |  |  | |  |
| **（学生填写）** | | | **课设进度记录（学生填写）** | |
| **（1）主要工作的描述**  处于待机状态时，数码管显示“HELLO”，投币后进入购买状态。  售货机可以接受1元、5元和10元纸币，可以售卖2.5元和5元的商品，投币和购买操作都能多次进行。当前消费额和余额都在数码管上显示。按下确认购买，进入找零状态，显示找零金额；也可以中途按下取消键，取消购买，退回所投币值。8s以后，自动返回初始状态，可以进行下一次购买。  任何时候按键使reset = 0，售卖机关闭；reset = 1时进入初始状态，但是原有的数据清零。所以在购买时不允许按下reset键，这会使得购买失败，无法出货和找零，发生“吞钱”事故。  **（2）难点、亮点**  可以接受1、5、10元三种币值；  可以购买多件商品；  显示金额上限为199.5元；  一次按键可以很短也可以很长，都只算作一次输入；  完成购买后可延时回到初态。 | | | 日期 | 进度 |
| 10.09 | 上午接受任务，下午确定设计题目； |
| 10.10 | 上午资料阅读，下午和晚上绘制状态图； |
| 10.11 | 上午完成顶层模块，下午完成状态机模块； |
| 10.12 | 上午完成显示模块，下午完善显示模块，显  示“hello”字样；  示“hello |
| 10.13 | 上午调试并下板测试显示模块，所有数据都显示正确；下午测试状态机模块，状态转换正确； |
| 10.14 | 下午下板测试，投币和购买操作完成；晚上完成计时模块。所有模块都编写完成。 |
| 10.16 | 上午初步检查功能，功能基本正确，但会显示多余的数字“0”，修改以后全部功能都正确。下午写仿真文件。 |
| **实验平台故障记录（学生填写，请注明实验平台的编号）**  平台编号03，无故障。 | |

**重要说明**

1. **时间安排：课内**2周。
2. **验收准备：**
   1. 完成本表学生应该填写部分；
   2. 每位学生必须都能**以独自完成的方式**应对任何形式的验收；
   3. 完成课程设计报告书（**格式参见模板）**；
   4. 将源程序和报告的电子文档交班长。
3. **检查过程：** 
   1. 提交验收准备材料，请求老师验收，之后按验收老师的要求做；
   2. 在开发平台上根据验收老师的要求进行演示；
   3. 检查过程中独立回答老师提出的相关问题；
   4. 验收老师有权根据具体情况调整验收的内容与方式；
   5. 验收完成后关闭电源，整理好设备。
4. **评分标准：**
5. 在完成控制器基本要求外，有**亮点**为加分项；
6. 在规定时间内完成控制器基本要求；
7. 在规定时间内完成控制器**部分**基本要求；
8. 检查时间。
9. **课程设计判定为不合格的一些情形： （本人已阅读此条款1-5项：签名 ）**
   1. **请人代做或冒名顶替者；**
   2. **替人做且不听劝告者；**
   3. **课程设计报告内容抄袭或雷同者；**
   4. **课程设计报告内容与实际实验内容不一致者；**
   5. **课程设计代码抄袭者。**

目 录

[1. 课程设计概述 1](#_Toc496022694)

[1.1 课设目的 1](#_Toc496022695)

[1.2 课设要求 1](#_Toc496022696)

[1.3 课设任务 1](#_Toc496022697)

[1.4 实验环境 2](#_Toc496022698)

[2. 基本方案设计 3](#_Toc496022699)

[2.1 题目选择 3](#_Toc496022700)

[2.2 设计内容 3](#_Toc496022701)

[2.3 设计思路 3](#_Toc496022702)

[2.4 代码实现 6](#_Toc496022703)

[2.5 仿真过程 16](#_Toc496022704)

[2.6 主要故障 24](#_Toc496022705)

[2.7 功能测试 26](#_Toc496022706)

[2.8 实验中遇到的主要问题及解决方法 29](#_Toc496022707)

[2.9 层次设计的体会 30](#_Toc496022708)

[3. 总结与心得 31](#_Toc496022709)

[3.1 课设总结 31](#_Toc496022710)

[3.2 课设心得 31](#_Toc496022711)

[4 参考文献 32](#_Toc496022712)

[附 录 自动售货机系统约束文件 33](#_Toc496022713)

# 课程设计概述

## **课设目的**

（1）掌握Vivado软件的使用方法；

（2）熟悉FPGA器件的使用方法；

（3）用Verilog HDL进行较复杂逻辑电路的设计和调试；

（4）学习数字系统的设计方法；

（5）通过规范化的实验报告，培养学生良好的文档习惯以及撰写规范文档的能力。

## **课设要求**

（1）能够全面地应用课程中所学的基本理论和基本方法，完成从设计逻辑电路到设计简单数字系统的过渡；

（2）能力独立思考、独立查阅资料，独立设计规定的系统；

（3）能够独立地完成实施过程，包括电路设计、调试、排除故障、仿真和下载验证。

## **课设任务**

**本次课程设计每组要完成两个设计任务，具体参见数字逻辑课程设计题目。**

（1）制定出详细设计方案，认真记载毕业设计工作日记；

（2）通过Verilog HDL完成规定的设计任务，采取模块化、层次化的设计方法设计电路，然后进行编译和仿真，认真记录实施过程中遇到的各自故障以及解决方法，保证设计的正确性；

（3）生成bit文件，下载到开发板上，通过实际线路进行验证设计的正确性；

（4）撰写设计报告，并对存在的问题进行分析、提出改进意见。

## **实验环境**

**开发环境为**Vivado 2015.2软件和开发板NEXYS 4（芯片为XC7A100TCSG324-1，封装为CSG3242）。Vivado 2015.2是使用Xilinx FPGA必备的设计工具。它可以完成FPGA开发的全部流程，包括设计输入、仿真、 综合、布局布线、生成bit文件、配置以及在线调试等功能。

Nexys4 开发板简介：如图1-1所示，它是一款简单易用的数字电路开发平台，可以支持在课堂环境中来设计一些行业应用。大规模、高容量的FPGA，海量的外部存储，各种USB、以太网、以及其它接口、这些让Nexys4-DDR能够满足从入门级组合逻辑电路到强大的嵌入式系统的设计。同时，板上集成的加速度、温度传感器，MEMs数字麦克风，扬声器放大器以及人量的I/O设备，让Vexys4-DDR不需要增添额外组件而用于各种各样的设计。

**注意：开发板提供的时钟信号频率为100Mhz，对应的引脚封装编号为“E3”。**

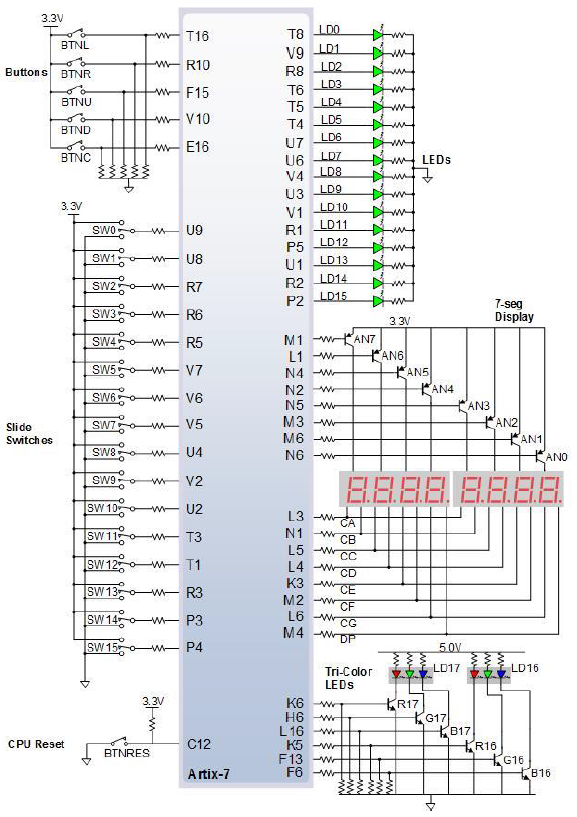


图 1-1 Nexys4通用I / O设备

# 基本方案设计

### **题目选择**

本人选择自动售货机设计。

### **设计内容**

设计并利用Nexys4 开发板来实现一个自动售货机，它具有下述功能：

（1）能识别1元、5元和10元纸币，总投币金额不大于99元。

（2）销售价格分别为2.5元和5元的两类商品，一次可以购买1到2两种商品，且数量任意，前提是总价值不大于99.5元。

（3）在数码管上显示当前消费额和余额，两者均用两位整数和一位小数显示。

（4）投入钱币后，可以按键选择商品，每按一次，商品数加一；当商品总价值大于投入的钱币金额时，按键无效，且溢出提示灯亮，提示用户不可继续投币或购买。

（5）确认投币和选货完毕以后，按下确认购买按键，数码管显示找零金额；若按下取消键，则找回所有已投入的钱，也由数码管显示。显示5s后自动回到初态，以备下次购买。

### **设计思路**

综合分析设计内容，自动售货机的结构设计如下。

**（1）控制器**

饮料自动投币售卖机控制器结构参见图2-1所示，自动销售机的电源开关同时作为电路总清零信号（Reset），当Reset = Off不工作、为On时电路进入初始状态，且电源指示灯亮（又称售卖机运行指示灯），控制器初始状态为“S0”并且显示“Hello”。

a．Reset作为电路总清零信号；

b．系统时钟（clk）作为系统的同步信号；

c．币值识别（coin\_val）：经由人民币识别器给出的每次投入的币值(具体定义：coin\_val =“b0001”表示1元，coin\_val =“b0101”表示5元，coin\_val =“b1010”表示10元)；

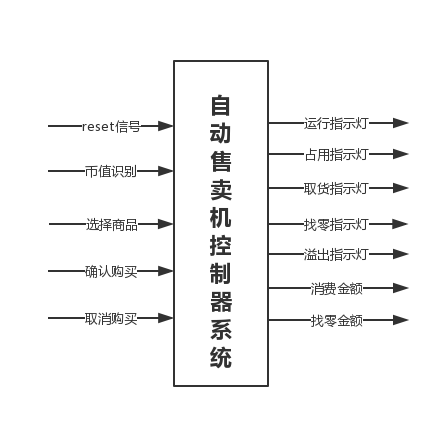


图2-1 自动投币售卖机控制器结构

d．选择商品（item）：item = “0” 表示选择价格为2.5元的商品，item = “1” 表示选择价格为5元的商品；

e．确认购买（assure）：assure = “1”表示用户确定购买所选商品；

f．取消操作信号（cancel\_flag）：cancel\_flag =“1”表示客户取消操作；

g．机器是否占用指示灯（hold\_ind）：hold\_ind =“1”表示该机器被占用；

h．取商品指示灯（drinktk\_ind）：drinktk\_ind =“1”表示客户可以取走商品；

i．找零与退币标志指示灯（charge\_ind）：charge\_ind =“1”表示找零提示；

j．消费金额（cost）：用户消费的金额，在0-99.5之间，用9位2进制码表示，高8位为整数部分，最低位为0表示没有小数部分，最低位为1表示小数部分为0.5；

k．找零与退币币值（charge\_val）：退给客户的币值，在0-99.5之间，用9位2进制码表示，高8位为整数部分，最低位为0表示没有小数部分，最低位为1表示小数部分为0.5；

j．自动投币售卖机的状态确定

S0 ：初始态；

S1 : 售卖状态；

S2 : 找零状态

S3 ：不工作；

**（2）控制器的状态图**

请同学给出自动投币售卖机的状态图并加以解释。见图2-2所示；



图2-2自动投币售卖机的状态图

**（3）控制器的模块图**

自动投币售卖机的模块图，参见图2-3所示。

（a）输入缓冲模块对所有的输入进行同步化处理，不改变输入的值；

（b）分频器模块对系统时钟进行降频，降频后的时钟作为显示模块的时钟；

（c）币值计算模块存储当前的累计币值，当有新的投币或者购买信号传入时，更新币值并输出给显示模块，当币值超出99.5元时，超出了显示范围，不可以继续操作，控制溢出指示灯亮；

（d）显示模块负责输出“HELLO”字样和当前金额；

（e）有限状态机根据输入的reset、币值、确认和超时信号进行状态转换，根据状态确定各指示灯的值，并把状态输出，供其它模块使用。

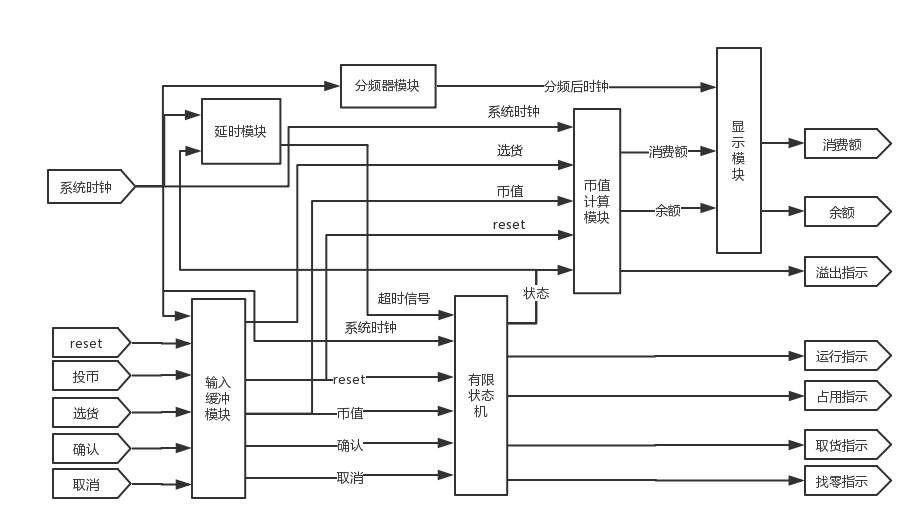


图2-3自动投币售卖机的模块图

### **代码实现**

自动售货机的顶层原理图如图2-4所示。

顶层模块下有输入缓冲模块、延时模块、状态机模块、币值计算模块、分频器模块和显示模块。

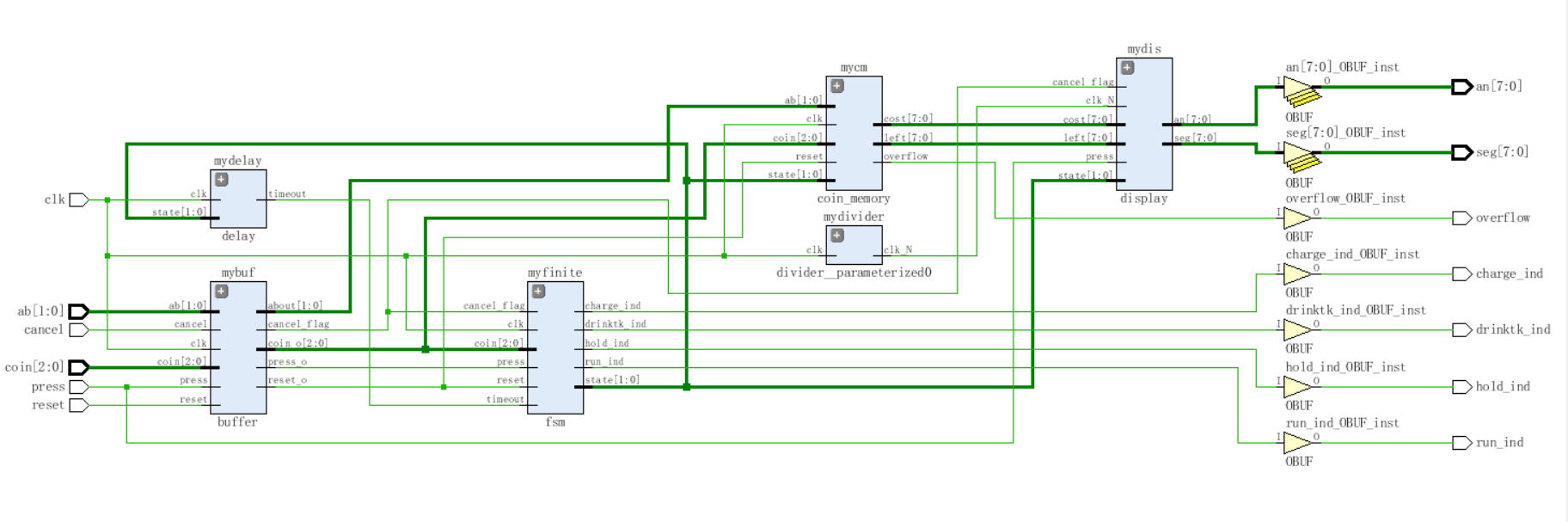


图2-4 自动售货机顶层原理图

1. 自动售货机顶层模块（用于整个售货机的控制）controller Verilog 代码

|  |
| --- |
| **程序2-1：顶层模块controller Verilog 代码** |
| `timescale 1ns / 1ps  module controller(  press, reset,  clk,ab,  cancel,coin,  run\_ind, hold\_ind,  drinktk\_ind, charge\_ind,overflow,  seg, an  );  input reset ; //清零  input clk; //系统时钟  input cancel; //取消信号  input press; //确认购买  input [2:0]coin; //投币  input [1:0]ab; //分别代表2.5和5元的商品  output run\_ind ; //运行指示灯  output hold\_ind; //占用指示灯  output drinktk\_ind; //取商品指示灯  output charge\_ind; //找零指示灯  output [7:0] seg; //七段显示输出  output [7:0] an; //片选信号  output overflow; //投币或购买溢出信号  wire [7:0]left; //剩余币值  wire [7:0]cost; //消费额  wire [2:0] coin\_o; //币值  wire [1:0] state; //状态  wire clk\_N; //显示模块用的时钟  wire cancel\_flag; //取消信号  wire reset\_o; //置位信号输出  wire press\_o; //确认购买信号输出  wire [1:0] about; //选择商品信号输出  wire timeout; //超时信号  //延时模块实例化  delay mydelay( .clk(clk), .state(state) , .timeout(timeout) );  //分频模块实例化  divider mydivider(.clk(clk), .clk\_N(clk\_N) );  //输入缓冲实例化  buffer mybuf (  .reset(reset), .press(press), .clk(clk), .ab( ab),  .cancel(cancel), .coin(coin),  .coin\_o(coin\_o), .cancel\_flag(cancel\_flag),  .reset\_o(reset\_o), .press\_o(press\_o), .about(about)  );  //总币值计算模块实例化  coin\_memory mycm( .reset(reset\_o), .clk(clk) , .ab(about), .coin(coin\_o),  .state(state), .cost(cost),.left(left), .overflow(overflow) );  //显示模块实例化  display mydis( .state(state),.cancel\_flag(cancel\_flag),  .press(press), .cost(cost), .left(left), .clk\_N(clk\_N) , .seg(seg), .an(an) );  //有限状态机实例化  fsm myfinite(  .reset(reset\_o), .clk(clk),  .coin(coin\_o),  .cancel\_flag(cancel\_flag), .press(press\_o), .timeout(timeout),  .hold\_ind(hold\_ind), .drinktk\_ind(drinktk\_ind),  .charge\_ind(charge\_ind), .run\_ind(run\_ind), .state(state) );  endmodule |

1. 底层模块buffer（用于读入数据）Verilog 代码

|  |
| --- |
| **程序2-2：底层模块**buffer**（用于读入数据）Verilog 代码** |
| `timescale 1ns / 1ps  module buffer(  reset, press,clk,ab,  cancel, coin,  coin\_o,  cancel\_flag,  reset\_o, press\_o,about  );  input reset; //置位信号  input cancel; //取消信号  input clk; //时钟  input press; //确认购买键  input [2:0] coin; //投币信号  input [1:0]ab; //选择商品的信号  output reg [1:0]about; //选择商品输出信号  output reg [2:0] coin\_o; //币值输出信号  output reg reset\_o; //置位信号输出  output reg cancel\_flag; //取消信号输出  output reg press\_o; //确认购买信号输出  always @(posedge clk) begin  //所有变量都不作处理，在时钟上升沿进行输出  //此模块仅起到对输入进行同步的作用  about <= ab;  reset\_o <= reset;  cancel\_flag <= cancel ;  press\_o <= press;  coin\_o <= coin;  end  endmodule |

1. 底层模块fsm（有限状态机）Verilog 代码

|  |
| --- |
| **程序2-3：底层模块fsm（有限状态机）Verilog 代码** |
| `timescale 1ns / 1ps  module fsm(  reset, clk, coin,  cancel\_flag,press, timeout,  hold\_ind, drinktk\_ind,  charge\_ind, run\_ind,state  );  input reset; //置位信号  input clk; //时钟  input press; //确认购买  input timeout; //超时信号  input cancel\_flag; //取消购买  input [2:0] coin; //币值信号  output reg run\_ind; //运行指示灯  output reg hold\_ind; //占用指示  output reg drinktk\_ind; //取物品指示  output reg charge\_ind; //找零指示  output reg [1:0] state; //状态  reg [1:0] statenext; //状态中间变量  parameter s1=1,s2=2,s3=3,soff= 0;  initial state <= soff;  // CombLogic  always @(posedge clk )begin  case(state)  //初态，投币时进入购买状态  s1:begin  if(coin>4'b0000) statenext <= s2;  else statenext <= s1;  run\_ind <= 1;  hold\_ind <= 0;  drinktk\_ind <= 0;  charge\_ind <= 0;  end  //购买状态，确认或取消时进入找零状态  s2:begin  if(press | cancel\_flag ) statenext <= s3;  else statenext <= s2;  run\_ind <= 1;  hold\_ind <= 1;  drinktk\_ind <= 0;  charge\_ind <= 0;  end  //找零状态，5s以后回到初态  s3:begin  if(timeout) statenext <= s1;  else statenext <= s3;  run\_ind <= 1;  hold\_ind <= 1;  drinktk\_ind <= 1;  charge\_ind <= 1;  end  //不工作状态，reset时进入初态  soff:begin  if( reset) statenext <= s1;  else statenext <= soff;  run\_ind <= 0;  hold\_ind <= 0;  drinktk\_ind <= 0;  charge\_ind <= 0;  end  endcase  end  // StateReg  //时钟上升沿进行状态转换  always @(posedge clk) begin  if(reset==0) state <= soff;  else state <= statenext;  end  endmodule |

1. 底层模块divider（用于对系统时钟进行分频）Verilog 代码

|  |
| --- |
| **程序2-4：底层模块divider（用于对系统时钟进行分频）Verilog 代码** |
| `timescale 1ns / 1ps  module divider  #(parameter N = 100\_000)  (clk, clk\_N);  input clk; // 系统时钟  output reg clk\_N; // 分频后的时钟  reg [31:0] counter; //计数器变量  initial begin  counter = 0;  clk\_N = 0;  end  always @(posedge clk) begin // 时钟上升沿  if(counter == N/2 - 1)begin      counter = 0;      clk\_N = ~clk\_N;  end  else    counter = counter + 1;  end  endmodule |

（5）底层模块coin\_memory（用于计算并保存当前金额）Verilog 代码

|  |
| --- |
| **程序2-5：底层模块**coin\_memory**（用于计算并保存当前金额）Verilog 代码** |
| `timescale 1ns / 1ps  module coin\_memory(  reset, clk,  ab, coin,state,  cost, left, overflow  );  input reset; //清零信号  input [1:0] state; //状态  input [2:0] coin; //输入币值信号  input clk; //时钟信号  input [1:0]ab; //选商品信号  output reg [7:0] left; //余额输出  output reg [7:0] cost; //消费额  output reg overflow; //投币或者购买溢出信号  reg [1:0]abold; //选商品信号的旧值  reg [2:0] coin\_old; //币值信号的旧值  reg [7:0] temp; //一次消费的金额  reg [4:0] coint; //一次投入的金额  initial begin  left <= 8'b00000000;  cost <= 8'b00000000;  abold <= 2'b00;  coin\_old <= 3'b000;  temp <= 8'b00000000;  overflow <= 1'b0;  coint <= 5'b00000;  end  always @(posedge clk) begin  //不工作状态和初态时金额为0  if( reset==0 || state==2'b01 ) begin  cost = 8'b00000000;  left = 8'b00000000;  end  else begin  //选择5元商品  //当选货信号的新值和旧值不同时触发  if( ab[1]!= abold[1] ) begin  temp = ab[1] ? 8'b00001010 : 0;  //商品价格大于余额，不可购买，溢出  if( temp > left ) overflow <= 1'b1;  else begin  //消费额将要超出99.5元，不可显示，溢出  if( cost + temp > 8'b11000111 ) overflow <= 1'b1;  //计算消费额和余额  else begin  cost = cost + temp ;  overflow <= 1'b0;  left = left - temp;  end  end  end  else cost = cost;  //选择2.5元商品，情况可类比5元商品  if( ab[0]!= abold[0] ) begin  temp = ab[0] ? 8'b00000101 : 0;  if( temp > left ) overflow <= 1'b1;  else begin  if( cost + temp > 8'b11000111 ) overflow <= 1'b1;  else begin  cost = cost + temp ;  overflow <= 1'b0;  left = left - temp;  end  end  end  else cost = cost;  //输入币值发生变化时触发  if( coin != coin\_old ) begin  //coin的3位分别对应1元、5元和10元  case(coin)  3'b001: coint = 5'b00010;  3'b010: coint = 5'b01010;  3'b100: coint = 5'b10100;  default:coint = 5'b00000;  endcase  //余额将要大于99.5元时，溢出  if( left + coint > 8'b11000111 ) overflow <= 1'b1;  else begin  overflow <= 1'b0;  left = left + coint;  end  end  else left = left;  //用变量保存币值和选货信号的旧值  abold <= ab;  coin\_old <= coin;  end  end  endmodule |

（6）底层模块delay（用于计时8s）Verilog 代码

|  |
| --- |
| **程序2-6：底层模块delay（用于计时8s）Verilog 代码** |
| `timescale 1ns / 1ps  module delay(  clk,state,timeout  );  input clk; //系统时钟  input [1:0]state; //系统状态  output reg timeout; //是否超时  wire clk\_N; //分频后的时钟  reg [2:0] count; //计数变量  //把时钟频率降到1Hz  divider #( 100\_000\_000 ) dilay( clk, clk\_N );  initial begin //输出初始化  timeout <= 1'b0;  count <= 3'b000;  end  always @( posedge clk\_N ) begin  case( state)  //找零状态时，开始计时，8s后输出超时信号  2'b11:begin  if( count == 3'b111 ) begin  count = 3'b000;  timeout = 1'b1;  end  else begin  count = count + 1'b1;  timeout = 1'b0;  end  end  //其他状态不计时  default: begin  timeout = 1'b0;  count = 3'b000;  end  endcase  end  endmodule |

（7）底层模块delay（用于计时8s）Verilog 代码

|  |
| --- |
| **程序2-6：底层模块delay（用于计时8s）Verilog 代码** |
| `timescale 1ns / 1ps  module display(  state, cancel\_flag,press,  cost,  left,  clk\_N,  seg, an  );  input clk\_N; // 时钟  input press; //确认信号  input cancel\_flag; //取消信号  input [1:0] state; //状态机状态  input [7:0] cost, left; //要显示的钱数  output reg[7:0] seg; // 分别对应CA、CB、CC、CD、CE、CF、CG和DP  output reg [7:0] an; // 8位数码管片选信号  reg [3:0]code; //待显示数字二进制码  reg [2:0]num; //计数器输出  reg [2:0]num\_t; //计数器输出的旧值  reg [8:0] total; //总金额  initial begin  total <= 9'b000000000;  an <= 8'b11111111;  num <= 3'b000;  seg <= 8'b11111111;  code <= 4'b1111;  num <= 3'b000;  end  always @( posedge clk\_N)begin  num\_t = num;  if(num[2:0] == 3'b111) num[2:0] = 3'b000;// 计时器  else num[2:0] = num[2:0] + 1;  case(state)  //初态显示“HELLO”  2'b01:begin  total = 9'b000000000;  case(num)           3'b000: code[3:0] = 4'b0000;  3'b001: code[3:0] = 4'b1100;  3'b010: code[3:0] = 4'b1100;  3'b011: code[3:0] = 4'b1011;  3'b100: code[3:0] = 4'b1010;  default:code[3:0] = 4'b1111;  endcase  end  //购买状态，显示消费额和余额  2'b10:begin  total = 9'b000000000;  case(num)  3'b000: code[3:0] = (left[0]) ? 4'b0101 : 4'b0000;  3'b001: code[3:0] = left[7:1] % 4'b1010;  3'b010: code[3:0] = (left[7:1] / 4'b1010)?(left[7:1] / 4'b1010):4'b1111;  3'b100: code[3:0] = cost[0] ? 4'b0101 : 4'b0000;  3'b101: code[3:0] = cost[7:1] % 4'b1010;  3'b110: code[3:0] = (cost[7:1] / 4'b1010)?(cost[7:1] / 4'b1010):4'b1111;  default: code[3:0] = 4'b1111;  endcase  end  //找零状态，显示找零金额  2'b11:begin  if( cancel\_flag) total[8:0] = cost[7:0] + left[7:0];  else if(press) total[8:0] = {1'b0+left[7:0]};  else total = total;  case( num)  3'b000: code = total[0] ? 4'b0101 : 4'b0000;  3'b001: code = total[8:1] % 8'b00001010;  3'b010: code = ( (total[8:1] % 8'b01100100 ) / 8'b00001010 ) ?  ( (total[8:1] % 8'b01100100 ) / 8'b00001010 ) : 4'b1111;  3'b011: code = (total[8:1] / 8'b01100100) ? (total[8:1] / 8'b01100100) : 4'b1111;  default:code = 4'b1111;  endcase  end  //不工作状态，不显示  2'b00:begin  total = 9'b000000000;  code[3:0] = 4'b1111;  end  endcase  //片选信号发生改变时触发  if( num[2:0] != num\_t[2:0] ) begin  case(num)  3'b000: an[7:0] = 8'b11111110;  3'b001: an[7:0] = 8'b11111101;  3'b010: an[7:0] = 8'b11111011;  3'b011: an[7:0] = 8'b11110111;  3'b100: an[7:0] = 8'b11101111;  3'b101: an[7:0] = 8'b11011111;  3'b110: an[7:0] = 8'b10111111;  3'b111: an[7:0] = 8'b01111111;  endcase  end  else an = an;  //将待显示数字的二进制代码转换为七段数码管编码  case(code[3:0])  4'b0000: seg[7:0] = 8'b11000000;  4'b0001: seg[7:0] = 8'b11111001;  4'b0010: seg[7:0] = 8'b10100100;  4'b0011: seg[7:0] = 8'b10110000;  4'b0100: seg[7:0] = 8'b10011001;  4'b0101: seg[7:0] = 8'b10010010;  4'b0110: seg[7:0] = 8'b10000010;  4'b0111: seg[7:0] = 8'b11111000;  4'b1000: seg[7:0] = 8'b10000000;  4'b1001: seg[7:0] = 8'b10010000;  4'b1010: seg[7:0] = 8'b10001001; //H  4'b1011: seg[7:0] = 8'b10000110; //E  4'b1100: seg[7:0] = 8'b11000111; //L  default: seg[7:0] = 8'b11111111; //全灭  endcase  case(state)  //购买状态时，第1位和第5位显示小数点  2'b10:begin  case(num)  3'b001: seg = seg - 8'b10000000;  3'b101: seg = seg - 8'b10000000;  default:seg = seg;  endcase  end  //找零状态时，第1位显示小数点  2'b11:begin  case(num)  3'b001: seg = seg - 8'b10000000;  default:seg = seg;  endcase  end  default: seg[7] <= 1'b1;  endcase  end  endmodule |

### 2.5 **仿真过程**

为了验证设计的正确性，对顶层模块、状态机、显示模块和币值计算等模块进行了仿真，具体过程如下：

**（1）顶层模块仿真**

目的：验证系统的基本功能，包括投币、选货、购买和延时回到初始状态。

输入：共6个，分别为：reset（置位）、clk（系统时钟）、cancel（取消操作）、press（确认购买）、coin（投币）、ab（选货）；

输出：共7个，分别为：run\_ind(运行指示)、hold\_ind（占用指示）、drinktk\_ind（取货指示）、charge\_ind（找零指示）、seg（七段显示输出）、an（片选信号）、overflow（溢出指示）。

**顶层模块仿真文件如下：**

|  |
| --- |
| **程序2-7：自动售货机主模块仿真文件** |
| `timescale 1ns / 1ps  module top\_test( );  reg reset ; //清零  reg clk; //系统时钟  reg cancel; //取消信号  reg press; //确认购买  reg [2:0]coin; //投币  reg [1:0]ab; //分别代表2.5和5元的商品  wire run\_ind ; //运行指示灯  wire hold\_ind; //占用指示灯  wire drinktk\_ind; //取商品指示灯  wire charge\_ind; //找零指示灯  wire [7:0] seg; //七段显示输出  wire [7:0] an; //片选信号  wire overflow; //投币或购买溢出信号  controller test\_cont( press, reset, clk,ab,cancel, coin,  run\_ind, hold\_ind, drinktk\_ind, charge\_ind,overflow, seg, an  );  always #5 clk = ~ clk;  initial begin  reset = 0;  clk = 0;  cancel = 0;  press = 0;  coin = 2'b00;  ab = 2'b00;  #15 reset = 1;  //投币  #120 coin = 3'b001;  #15 coin = 3'b010;  #15 coin = 3'b100;  #15 coin = 3'b000;  //选货  ab = 2'b01;  #15 ab = 2'b00;  #15 ab = 2'b01;  #15 ab = 2'b10;  #15 ab = 2'b00;  //购买  #20 press = 1;  #15 press = 0;  end  endmodule |

主模块仿真图如图2-5所示。

确认购买

购买2×2.5+5=10元商品

投币1+5+10=16元



找零6.0元

自动回到初始状态，显示“HELLO”

显示“HELLO”

图2-5 自动售货机主模块仿真测试图

a. 零时刻reset= 0，然后另reset = 1，进入初始状态，显示“HELLO”，七段数码管的编码为“HELLO”编码；

b. 依次投入1元、5元和10元，共计16元，数码管显示16元；

c. 依次购买2.5元、2.5元和5元的商品，共三件商品，消费10元，数码管显示10元；

d. 按下确认购买按钮，press = 1，交易完成，数码管显示找零金额6元；

e. 5s以后，系统自动回到初始状态，数码管重新显示“HELLO”。

**（2）状态机模块仿真**

目的：验证状态机的基本功能，在各种输入条件下的状态转换和指示信号输出。

输入：共6个，分别为：reset（置位）、clk（系统时钟）、cancel（取消操作）、press（确认购买）、coin（投币）、ab（选货）；

输出：共7个，分别为：run\_ind(运行指示)、hold\_ind（占用指示）、drinktk\_ind（取货指示）、charge\_ind（找零指示）、seg（七段显示输出）、an（片选信号）、overflow（溢出指示）。

**状态机模块仿真文件如下：**

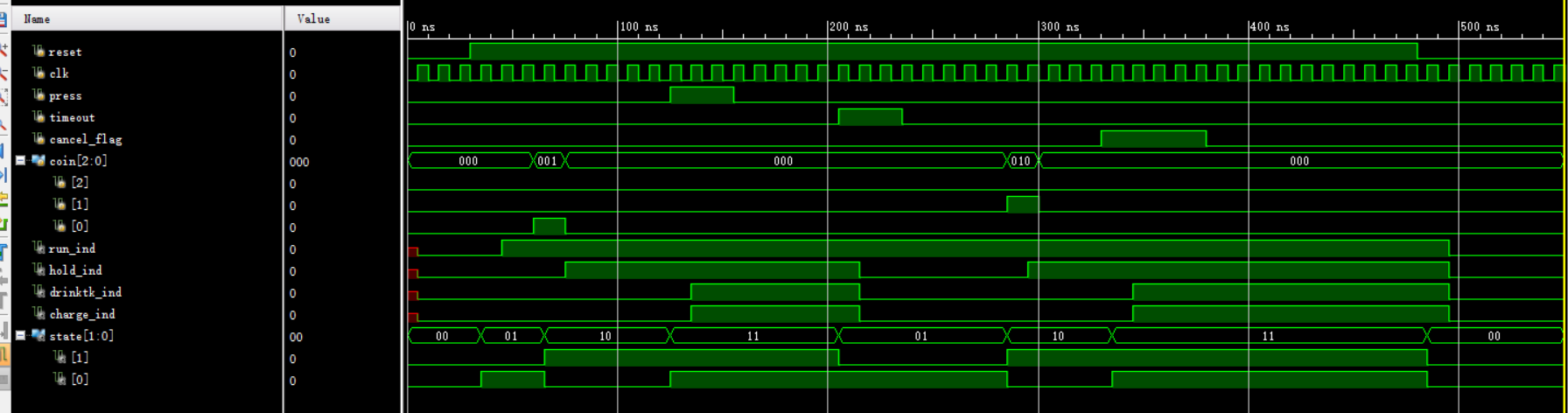
|  |
| --- |
| **程序2-8：状态机模块仿真文件** |
| `timescale 1ns / 1ps  module fsm\_test( );  reg reset; //置位信号  reg clk; //时钟  reg press; //确认购买  reg timeout; //超时信号  reg cancel\_flag; //取消购买  reg [2:0] coin; //币值信号  wire run\_ind; //运行指示灯  wire hold\_ind; //占用指示  wire drinktk\_ind; //取物品指示  wire charge\_ind; //找零指示  wire [1:0] state; //状态  fsm fsmtest(  .reset(reset), .clk(clk),  .coin(coin),  .cancel\_flag(cancel\_flag), .press(press), .timeout(timeout),  .hold\_ind(hold\_ind), .drinktk\_ind(drinktk\_ind),  .charge\_ind(charge\_ind), .run\_ind(run\_ind), .state(state) );  initial begin  reset = 0;  clk = 0;  press = 0;  timeout = 0;  cancel\_flag = 0;  coin = 3'b000;  #30 reset = 1;  #30 coin = 3'b001;  #15 coin = 3'b000;  #50 press = 1;  #30 press = 0;  #50 timeout = 1;  #30 timeout = 0;  #50 coin = 3'b010;  #15 coin = 3'b000;  #30 cancel\_flag = 1;  #50 cancel\_flag = 0;  #100 reset = 0;  end  always #5 clk = ~clk;  endmodule |

状态机模块仿真图如图2-6所示。

再次投币

确认购买，进入找零状态

有投币信号，进入购买状态



取消购买，进入找零状态

不工作状态，灯全灭

图2-6 自动售货机状态机模块仿真测试图

a. reset=0，处于不工作的状态，state = 2b00，所有指示灯都不亮；

b. reset = 1，进入初始状态，state= 2b01，运行指示灯亮；

c. 投入1元钱，进入购买状态，state= 2b10，运行指示灯和占用指示灯亮；

d. 按下确认购买按钮，进入找零状态，state = 2b11，找零指示灯亮；

e. 再次投币，又进入购买状态，然后按下取消按钮，也进入找零状态，找回所有投入的钱。

**（3）币值计算模块仿真**

目的：验证币值计算模块的基本功能，在投币和购买信号下实时输出总金额。

输入：共5个，分别为：reset（置位）、clk（系统时钟）、coin（币值）、state（状态机状态）、ab（选货）；

输出：共3个，分别为：left (余额)、cost（消费额）、overflow（溢出指示）。

**币值计算模块仿真文件如下：**

|  |
| --- |
| **程序2-9：币值计算模块仿真文件** |
| `timescale 1ns / 1ps  module coin\_test( );  reg reset; //清零信号  reg [1:0] state; //状态  reg [2:0] coin; //输入币值信号  reg clk; //时钟信号  reg [1:0]ab; //选商品信号  wire [7:0] left; //余额输出  wire [7:0] cost; //消费额  wire overflow; //投币或者购买溢出信号  coin\_memory cointest(  reset, clk,  ab, coin,state,  cost, left, overflow  );  initial begin  clk = 0;  state = 2'b00;  reset = 0;  coin = 3'b000;  ab = 2'b00;  #45 reset = 1;  #50 coin = 3'b001;  #30 coin = 3'b100;  #20 coin = 3'b000;  #50 ab= 2'b10;  #35 ab = 2'b01;  #50 ab = 2'b00;  #20 ab = 2'b10;  #20 ab = 2'b00;  #20 ab = 2'b10;  #20 ab = 2'b00;  #30 reset = 0;  end  always #5 clk = ~clk;  endmodule |

币值计算模块仿真图如图2-7所示。

a. reset = 0，处于不工作状态，币值为0；

b. reset = 1，进入初始状态，币值仍然为0；

c. 依次投入10元和1元，进入购买状态，共计11元，余额为11元，消费额为0元；

d. 依次选择2.5元和5元的商品，消费额7.5元，余额3.5元；

e. 继续购买5元商品，余额只有3.5元，无法购买，此时溢出指示灯亮，提示用户不要继续购买。

投币11元，显示余额为11元

初始状态，币值为0



欲购买5元商品，余额不足，溢出指示灯亮

消费7.5元，余额3.5元

图2-7 币值计算模块仿真测试图

**（4）显示模块仿真**

目的：验证显示模块的基本功能，在不同状态下显示不同的值。

输入：共6个，分别为：clk（系统时钟）、press（确认购买）、state（状态机状态）、cancel\_flag（取消购买）、cost（消费额）、left（余额）；

输出：共2个，分别为：seg（七段数码管待显示数字的编码）、an（数码管的片选信号）。

**显示模块仿真文件如下：**

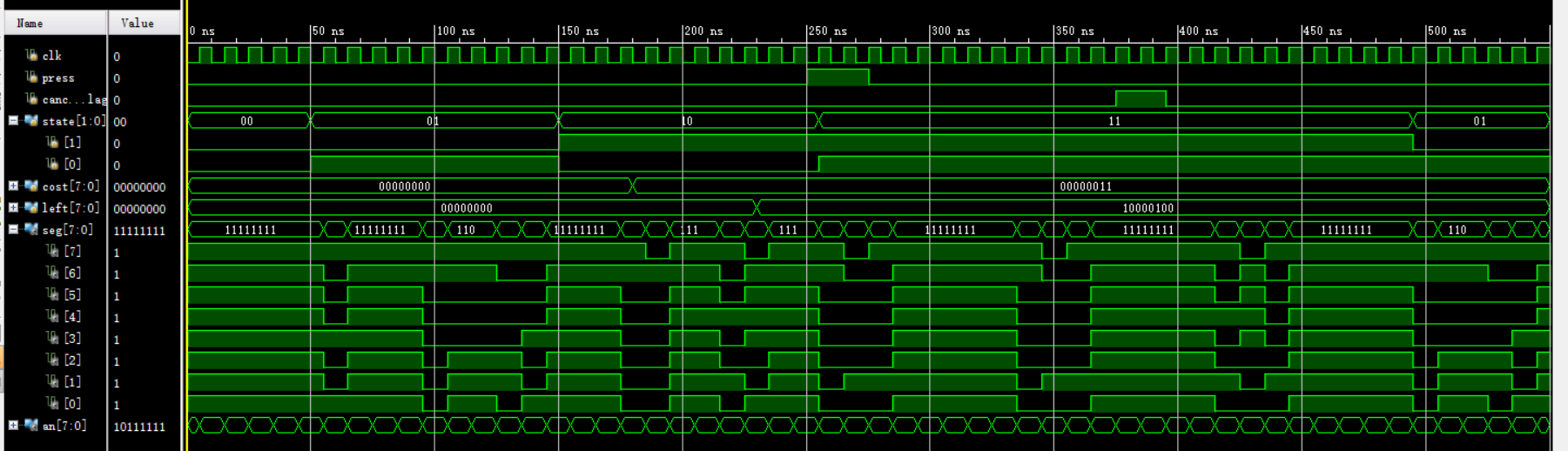
|  |
| --- |
| **程序2-10：显示模块仿真文件** |
| `timescale 1ns / 1ps  module display\_testbench( );  reg clk; // 时钟  reg press; //确认信号  reg cancel\_flag; //取消信号  reg [1:0] state; //状态机状态  reg [7:0] cost; //消费额  reg [7:0] left; //余额  wire [7:0] seg; // 分别对应CA、CB、CC、CD、CE、CF、CG和DP  wire [7:0] an; // 8位数码管片选信号  display mydisplay(  .state(state), .cancel\_flag(cancel\_flag),  .press(press), .cost(cost), .left(left),  .clk\_N(clk) , .seg(seg), .an(an) );  always begin  #5 clk <= ~clk;  end  initial begin  clk = 1'b0;  press = 1'b0;  state = 2'b00;  cost = 8'b00000000;  left = 8'b00000000;  cancel\_flag = 1'b0;  #50 state <= 2'b01;  #100 state <=2'b10;  #30 cost <= 8'b00000011;  #50 left <= 8'b10000100;  #20 press <= 1'b1;  #5 state <= 2'b11;  #20 press <= 1'b0;  #100 cancel\_flag <= 1'b1;  #20 cancel\_flag <= 1'b0;  #100 state <= 2'b01;  end  endmodule |

显示模块仿真图如图2-8所示。

回到初态，显示“HELLO”

初态，显示“HELLO”

不工作，显示全灭

****

取消购买，显示所投金额

确认购买，显示余额

图2-8 显示模块仿真测试图

a. 首先为不工作状态，数码管应该全灭，所以seg编码为全1；

整点报时5个单位时间

b. 进入初始状态，应该显示“HELLO”，seg编码为“HELLO”对应编码；

c. 进入购买状态，且cost（消费额）和left（余额）都不为0，故数码管显示消费额和余额；

d. 按下确认购买，进入找零状态，找零金额即为余额，由数码管显示；

e. 按下取消购买，进入找零状态，找回消费额与余额的和，即所有投入的钱，由数码管显示；

f. 5s以后，自动回到初始状态，再次显示“HELLO”。

### **主要故障**

（1）故障1（错误实例）

**问题描述**：显示“HELLO”失败，数码管为微亮状态。

**问题分析**：数码管的显示利用了人的视觉暂留，当频率过低时会看到图像闪烁，频率过高则会出现微亮状态，所以应该是分频器调用时参数错误。

**解决方法**：修改分频器参数，使得分频后的时钟频率在100Hz左右。

**解决情况：**成功显示“HELLO”。

（2）故障2（错误实例）

**问题描述**：显示消费额和找零金额时，有时会在不需要出现小数点的位置显示小数点，需要小数点的位置则没有小数点。

**问题分析**：经过观察和记录，错误的输出主要为几组(括号内为需要输出的值)：155（15.5）、1.1.0（11.0）、0.0.0（00.0）……这些数据都有一个共同点，即存在相邻的两位数字是相同的，这就是问题所在。因为在显示模块中，8个数码管是从右至左依次循环点亮的，显示值必须随着数码管的移动而变化，这里就是因为没有变化，导致相邻两位输出了同样的数字，小数点出现了两次。

**解决方法**：在显示模块中修改待输出数字的代码，当数码管编码变化时，输出的数字也变化。

**解决情况：**修改以后，数字和小数点都不再有错误。

（3）故障3（错误实例）

**问题描述**：购买过程中将reset置为0，进入不工作的状态，再使reset=1，又进入购买状态，发现原来的币值还被保留，没有被清零。

**问题分析**：reset是强制置零信号，不论当前在进行何种操作，只要reset=0，必须强制清零，这里的reset并没有起到清零的作用，仅仅是表面上不工作，并未改变寄存器中变量的值。

**解决方法**：应该将reset信号作为币值计算模块的输入，当reset=0时，不管投入何种币值，都使总币值为0，这样，再次打开时，原来的币值不会被保留，会被清零。

**解决情况**：在币值计算模块中添加reset输入后，再次上板测试，reset信号可以清除所有信号，再次打开时不受原来币值影响。

（4）故障4（错误实例）

**问题描述**：按键时间过长时，数码管显示的余额会迅速增加，相当于连续投币。

**问题分析**：这是因为人的按键时间至少为0.2s，而时钟的频率达到了上万Hz，一次按键可以被时钟上升沿读取若干次，每次读取都算作投币或购买，显然是不合理的，所以不能使用电平方式判断投币和购买信号，只有当投币和购买信号在上升沿时才算作有效输入。

**解决方法**：可以使用异步电路实现，当投币和购买信号上升沿时修改币值，但是本系统的尽量使用同步时序电路，所以放弃此方法。改用一个中间变量保存上一个时钟周期中投币信号的值，当下一个时钟上升沿到来时，判断当前投币信号是否与旧值相等，假如不想等则修改币值。

**解决情况**：修改以后再次测试，无论按键时间多长，都只算作一次输入。

（5）故障5（错误实例）

**问题描述**：币值显示时预留了两位整数，当实际金额只有一位整数时，十位上会显示为0，如05.5，这个0是多余的，影响美观。

**问题分析**：在显示模块中，待显示的值都是用公式算出的，有可能为0，也可能非0，当最高位为0时，也会正常显示，就导致了前述情况的出现。

**解决方法**：在现实模块中判断最高位是否为0，若为0则不显示。

**解决情况**：高位不再有多余的0出现，显示结果显得简洁许多。

### **功能测试**

共进行了 4项测试，它们分别为： 显示“HELLO”测试、投币和购买测试、取消购买测试和溢出测试。

（1）显示“HELLO”测试

将reset置为0，所有指示灯和数码管全灭，如图2-1（a）所示。

将reset置为1，进入初始状态，显示“HELLO”，运行指示灯亮，如图2-1（b）所示。



图2-1（a）不工作状态

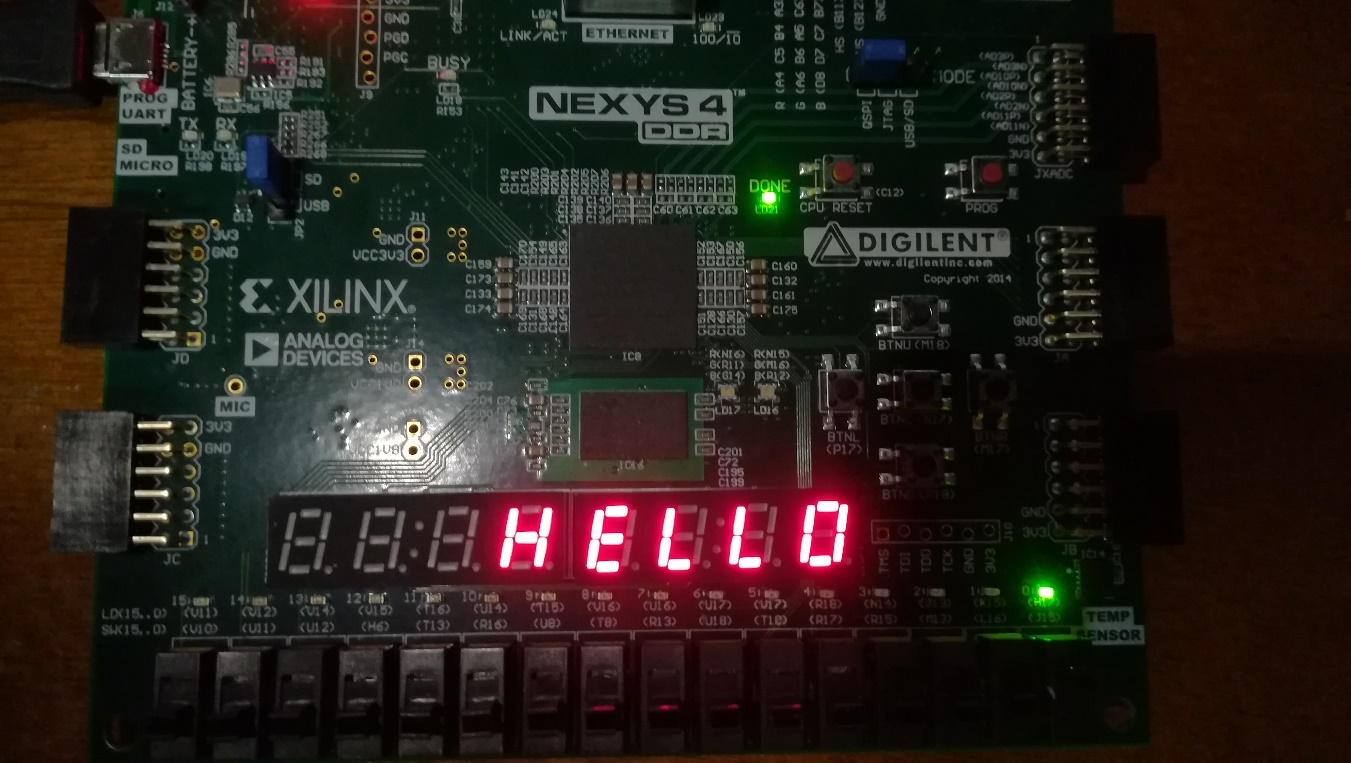


图2-1（b）初始状态显示“HELLO”

从以上照片可以看出，reset键可以控制整个系统的开闭，且初始状态时“HELLO”字样显示正确。

（2）投币和购买测试

多次按下投币按钮，任意投入1元、5元和10元，最后累计42元，此时进入购买状态，运行指示灯和机器占用指示灯亮，如图2-2（a）所示。

连续按下两次5元商品的购买按钮，再按一次2.5元的购买按钮，消费额共12.5元，余额为29.5元，如图2-2（b）所示。

将确认购买开关闭合，此前所选的所有商品都发生交易，进入找零状态，运行指示灯、占用指示灯、找零指示灯和取货指示灯都亮，找零金额为29.5元，如图2-2（c）所示。

购买完成以后，计时器开始计时，8s后自动回到初始状态，如图2-2（d）所示。

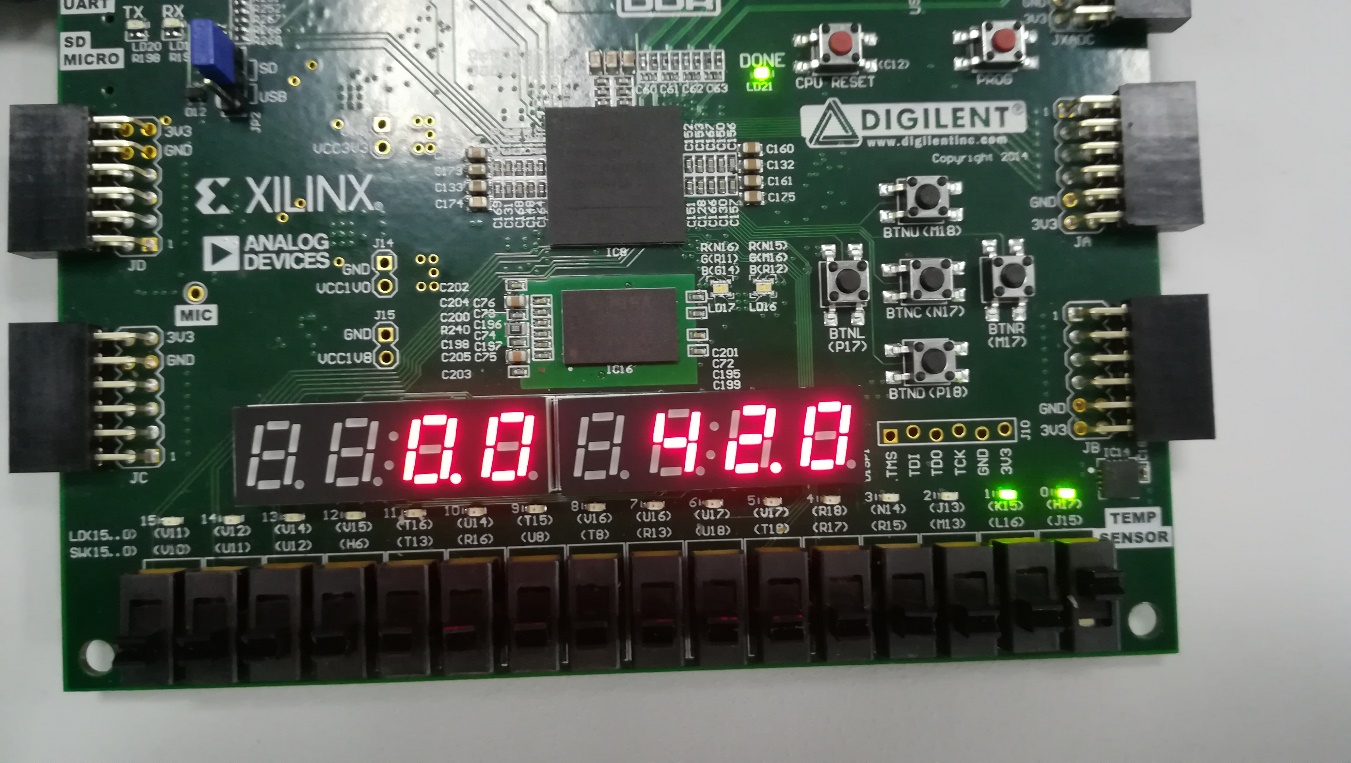


图2-2(a) 投币且尚未购买状态

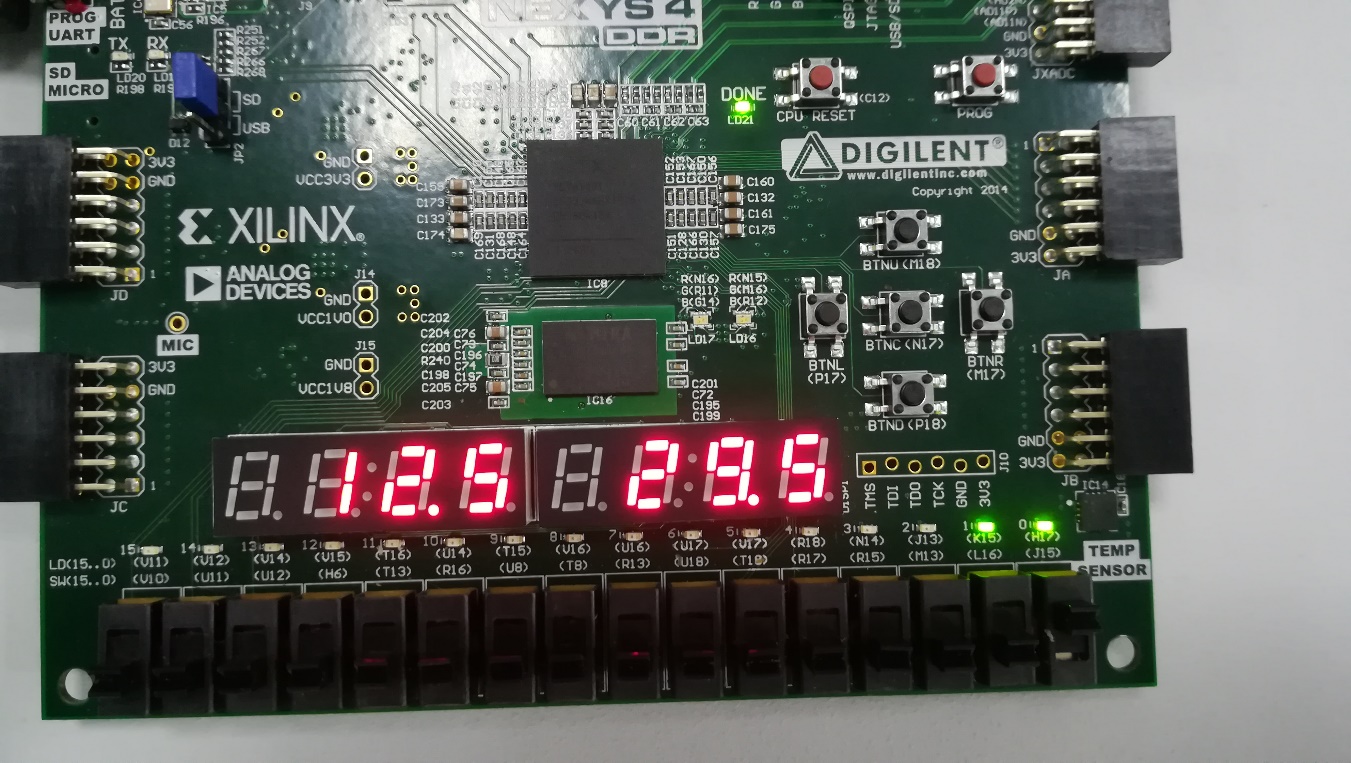


图2-2（b） 投币且购买状态



图2-2（c） 找零状态

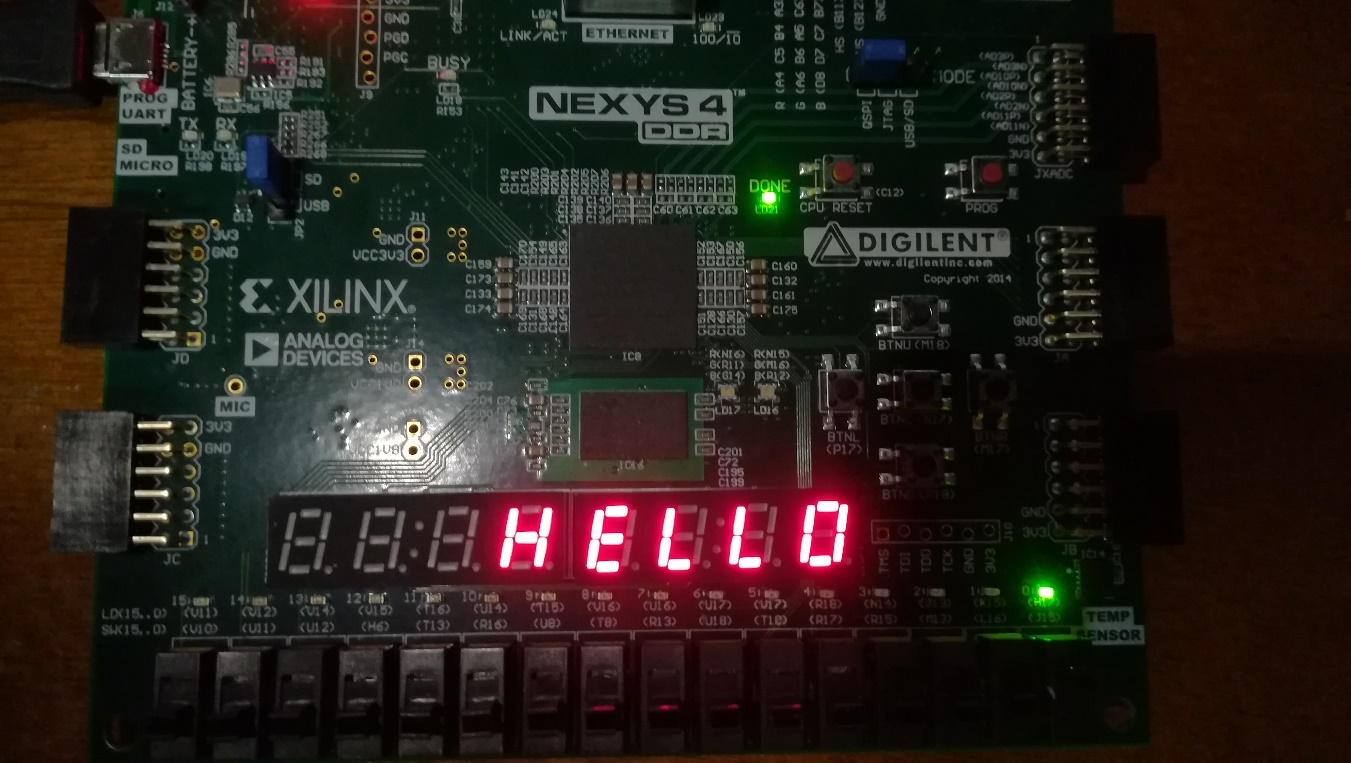


图2-2（d） 购买完成回到初态

根据上述测试结果可知，购买操作功能正确。

（3）取消购买测试

任意投入足够多的钱，如72元，购买总价格为17.5的多件商品，如图2-3（a）所示。

按下取消按钮，将所投入的72元全部找回，如图2-3（b）所示。

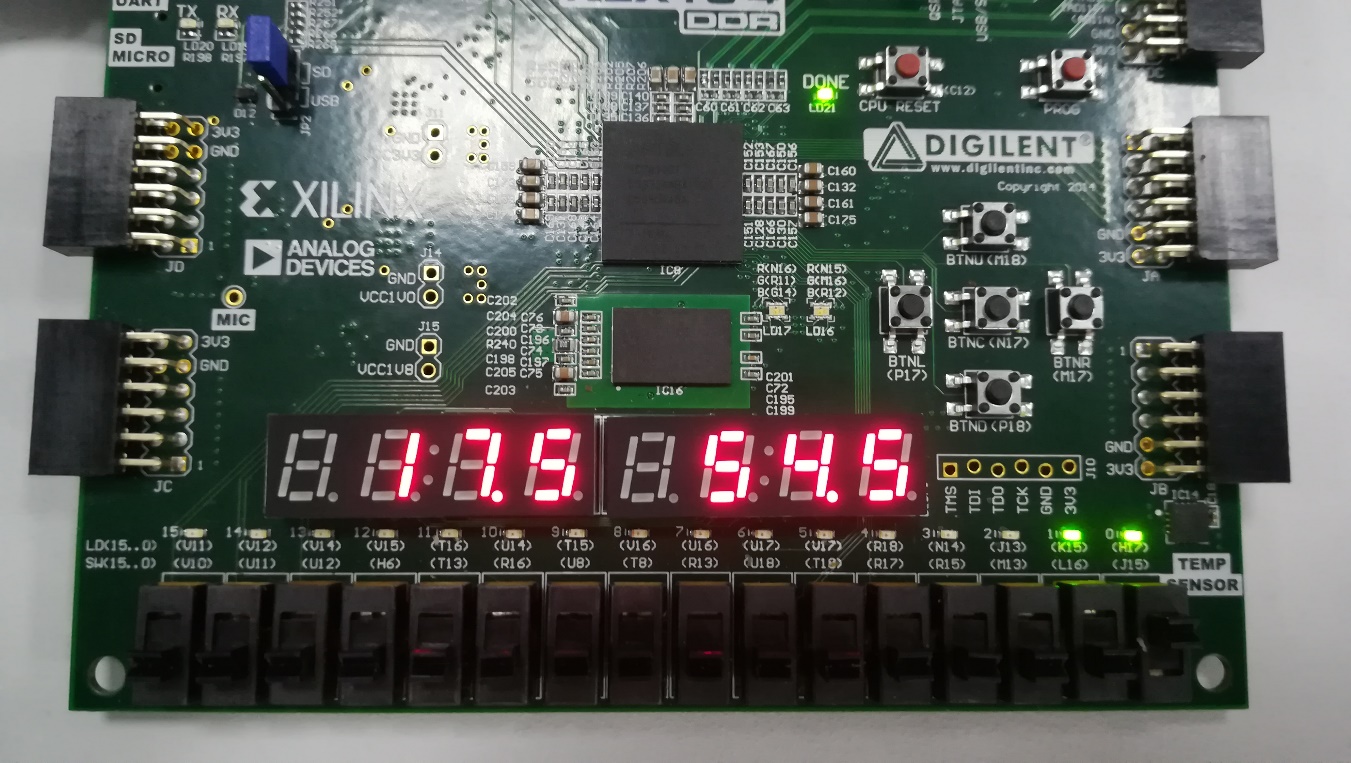


图2-3（a） 投币并购买

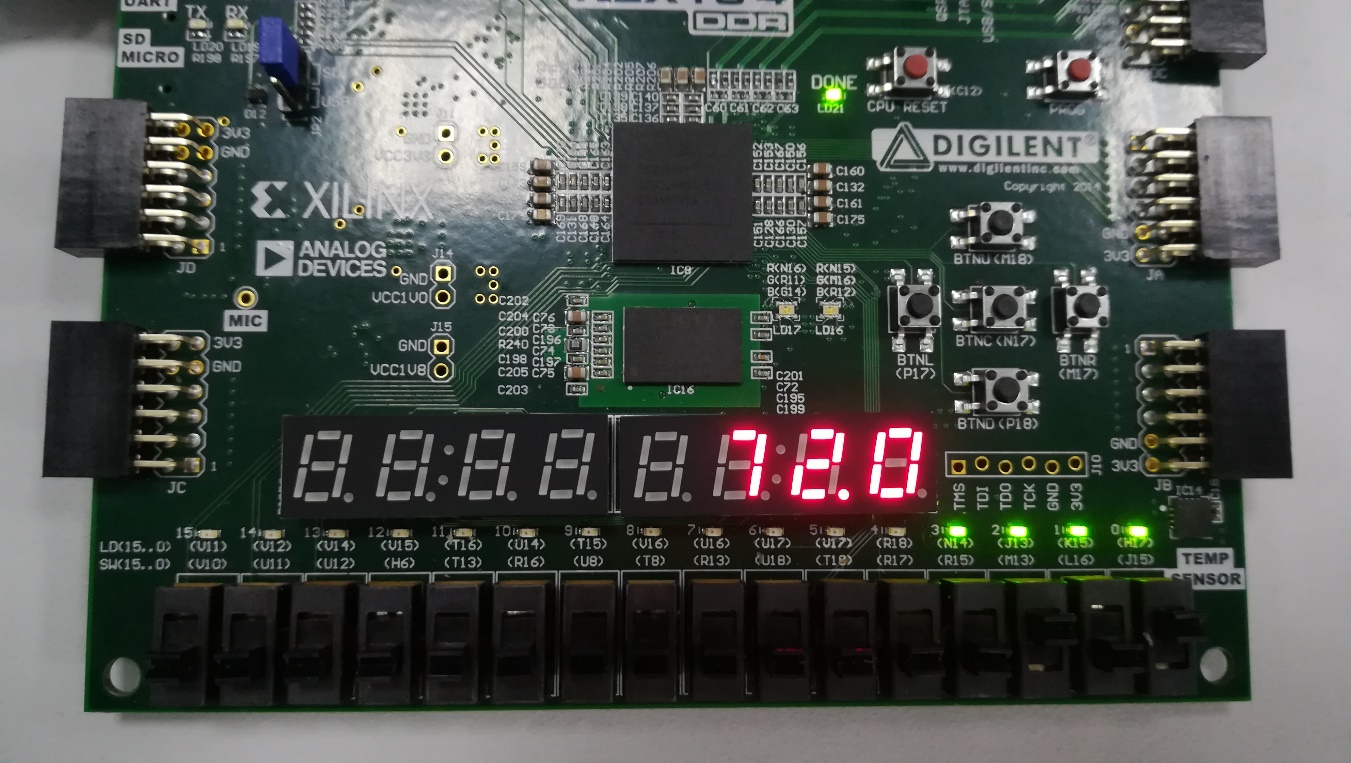


图2-3（b） 取消购买并退币

取消操作以后，计时器也开始计时，8s后回到初态，与确认购买时相同，所以此处略去该图。

由上述操作可知，取消购买操作的功能正确。

（4）溢出测试

投入1元，选则购买5元的商品，此时余额不足，无法购买，溢出（下溢）指示灯亮，如图2-4（a）所示。

显示消费额和余额的数码管都为3位，两位整数和一位小数，所以最大可显示金额为99.5，当再次投币后的总金额将达到100时，将不再增加币值，溢出指示灯亮(上溢)，如图2-4（b）所示。

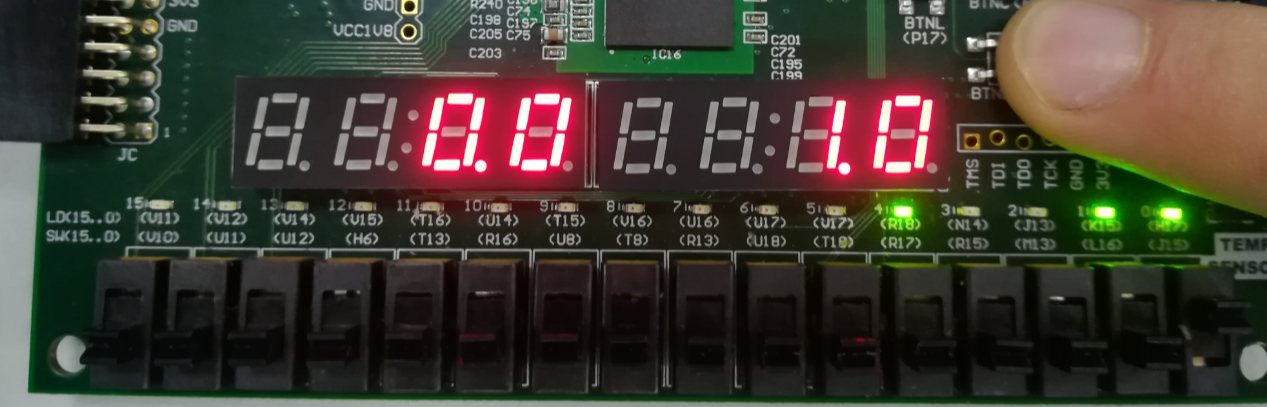


图2-4（a） 余额不足无法购买



图2-4(b) 投币金额溢出

当消费额超出99.5元时，不论余额为为多少，也不许继续购买，继续进行购买操作时溢出指示灯亮，提示用户停止购买，与前述两种情况类似。

综上，售卖机的防止溢出操作功能正常，可以防止顾客的不当操作并实时提醒。

### **实验中遇到的主要问题及解决方法**

表2-1 主要问题及解决方法

|  |  |
| --- | --- |
| 问题 | 解决方案 |
| 如何使输入同步化 | 编写输入缓冲模块，对输入进行处理，使其他功能模块不直接接收输入，而是处理已经同步化处理的输入信号。 |
| 有限状态机如何实现 | 使用“三段式”的方法，用一个中间变量statenext存储下一个状态的值，等到时钟上升沿到来时，将statenext赋值给state。 |
| 如何获得100Hz的时钟 | 使用分频器，对100MHz的系统时钟进行降频。 |
| 如何计算投入的钱币总值 | 使用一个寄存器模块，能够记录此前已经投入的总金额，且在再次投币时，能够立即更新寄存器的值。 |
| 如何显示“HELLO”字样 | 仿照10进制数的输出，给字幕“H”、“L”和“E”编出对应的8位数码管编码，字母就可以像数字一样进行输出。 |
| 如何分别在不同状态显示“HELLO”以及消费额 | 将状态机的状态作为显示模块的输入，显示模块可以根据状态来决定具体显示那种信息。寄存器模块计算的金额也需要传给显示模块。 |
| 如何在购买以后回到初态 | 为了良好的用户体验，我们不能要求顾客购买完成以后按某个键回到初始态，所以必须自动回到初始状态，这要使用一个计时模块，在进入找零状态时开始计时，若干秒之后跳转到初始状态 |
| 如何显示小数点 | 正常数字的8位二进制码最高位为1，对应小数点不亮，在需要显示小数点时，将二进制码减去8’b10000000即可使小数点点亮。 |

### **2.8 层次设计的体会**

系统由一个顶层模块controller来整合实现整个自动售货机系统的所有功能，其下有5个模块：buffer（输入缓冲）、divider（分频器）、delay（延时模块）、coin\_memory（币值计算模块）、fsm（有限状态机）和display（显示模块）。各模块功能和相互之间的参数传递如下。

输入缓冲模块负责将所有的输入整合到时钟上升沿时刻，把异步信号变成同步信号并输出，供其它模块使用。

分频器模块将系统时钟降频到100Hz左右，供显示模块使用，此模块可以放在显示模块内嵌套调用，而不作为一个单独的模块。

币值计算模块接收投币信号和购买信号，计算投币和购买以后的余额以及消费额，将金额输出，供显示模块使用。

有限状态机模块将币值、确认购买、取消购买和reset信号作为输入，根据输入信号进行状态的转变，并输出机器运行的若干个指示信号。

显示模块将状态机的输出状态作为输入，在不同状态下分别显示不同的值。初始状态显示“HELLO”；购买状态接收来自币值计算模块的金额值，并将其输出；找零状态显示找零金额。

顶层模块中调用其他底层模块，使得各部分功能并发进行。各模块之间的互相依赖，互相利用对方的输出作为自己的输入，关系紧密。

# 总结与心得

## **课设总结**

为了实现自动售货机系统的功能要求，作了如下几点工作：

（1）首先进行资料查阅，了解自动售货机的工作原理，然后联系显示，参考生活中的自动售货机，对其进行修改和提升，设计出一个功能更完备的售货机。

（2）确定输入和输出参数，并确定机器工作的状态，由此作出状态图，确定状态转移关系。根据状态图，弄清机器工作机理和流程，由此可以确定主要分成哪些模块，从而作出模块图。根据模块图，完成顶层模块，其中包含输入缓冲、延时、分频、计算、状态机和显示模块。

（3）完成顶层模块后，开始子模块的设计，在实现过程中，发现子模块的输入输出参数设计得不够合理，进行了多次修改，以使得各个模块之间的划分和协作最为合理。

（4）最初只写出了子模块的框架，不计较其中的功能，使整个系统能够运行起来，再去增加详细的功能。当顶层模块综合出错时，对子模块逐个进行仿真，从而缩小错误的范围。

（5）在粗略完成了所有的功能后，交由老师检查，对于某些不合理的部分，再进行修改，使系统的健壮性更强，用户体验更好。

## **课设心得**

（1）万事开头难，最开始设计模块和和顶层模块的两天进度最慢，因为没有理清思路，不能站在足够的高度掌握这个系统，当清楚了系统的结构以后，做起来就很顺手了。

（2）在碰到难题的时候，找班上的同学进行讨论，看他们是如何处理那些棘手的问题，对于自己的实现很有帮助，经常恍然大悟。

（3）要把一个系统的功能分成很合理的几个模块，互相之间的参数传输关系合理，不是一件容易的事，需要很强的逻辑性，也需要些许的运气。

（4）课程设计对于强化verilog语言的编程能力锻炼有极大的帮助，上学期在verilog实验和数字逻辑实验中没有弄清楚的问题，大都在课程设计的过程中明白了，做完以后感觉很有成就感。

# **4 参考文献**

**教学参考书：**

[1]欧阳星明，于俊清. 数字逻辑. 武汉：华中科技大学出版社，2012

[2]白中英，谢松云. 数字逻辑. 北京：科学出版社，2013

[3]徐光辉，程东旭，黄如. 基于FPGA的嵌入式开发与应用. 北京:电子工业出版社, 2006.

[4]Guy Even. Digital Logic Design: A Rigorous Approach, Cambridge University Press, 2012

[5]David Harris. Digital Design and Computer Architecture(Second Edition), Morgan Kaufmann, 2012

[6]Stephen Brown. 数字逻辑基础与Verilog设计, 机械工业出版社, 2009

**课外文献阅读 ：**

[1]Introductory Digital Systems Laboratory

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-111-introductory-digital-systems-laboratory-fall-2002

[2]Digital Design Fabrication

http://ocw.mit.edu/courses/architecture/4-510-digital-design-fabrication-fall-2008/

[3]Analysis and Design of Digital Integrated Circuits

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-374-analysis-and-design-of-digital-integrated-circuits-fall-2003/

# **附 录 自动售货机系统约束文件**

## Clock signal

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

##Switches

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { reset }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { press }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { cancel }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { run\_ind }]; #IO\_L18P\_T2\_A24\_15 Sch=led[0]

set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { hold\_ind }]; #IO\_L24P\_T3\_RS1\_15 Sch=led[1]

set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { drinktk\_ind }]; #IO\_L17N\_T2\_A25\_15 Sch=led[2]

set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { charge\_ind }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]

set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { overflow }]; #IO\_L7P\_T1\_D09\_14 Sch=led[4]

##7 segment display

set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { seg[0] }]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca

set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { seg[1] }]; #IO\_25\_14 Sch=cb

set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { seg[2] }]; #IO\_25\_15 Sch=cc

set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { seg[3] }]; #IO\_L17P\_T2\_A26\_15 Sch=cd

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { seg[4] }]; #IO\_L13P\_T2\_MRCC\_14 Sch=ce

set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { seg[5] }]; #IO\_L19P\_T3\_A10\_D26\_14 Sch=cf

set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { seg[6] }]; #IO\_L4P\_T0\_D04\_14 Sch=cg

set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { seg[7] }]; #IO\_L19N\_T3\_A21\_VREF\_15 Sch=dp

set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { an[0] }]; #IO\_L23P\_T3\_FOE\_B\_15 Sch=an[0]

set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { an[1] }]; #IO\_L23N\_T3\_FWE\_B\_15 Sch=an[1]

set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { an[2] }]; #IO\_L24P\_T3\_A01\_D17\_14 Sch=an[2]

set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { an[3] }]; #IO\_L19P\_T3\_A22\_15 Sch=an[3]

set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { an[4] }]; #IO\_L8N\_T1\_D12\_14 Sch=an[4]

set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { an[5] }]; #IO\_L14P\_T2\_SRCC\_14 Sch=an[5]

set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { an[6] }]; #IO\_L23P\_T3\_35 Sch=an[6]

set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { an[7] }]; #IO\_L23N\_T3\_A02\_D18\_14 Sch=an[7]

##Buttons

set\_property -dict { PACKAGE\_PIN N17 IOSTANDARD LVCMOS33 } [get\_ports { coin[1] }]; #IO\_L9P\_T1\_DQS\_14 Sch=btnc

set\_property -dict { PACKAGE\_PIN M18 IOSTANDARD LVCMOS33 } [get\_ports { ab[0] }]; #IO\_L4N\_T0\_D05\_14 Sch=btnu

set\_property -dict { PACKAGE\_PIN P17 IOSTANDARD LVCMOS33 } [get\_ports { coin[0] }]; #IO\_L12P\_T1\_MRCC\_14 Sch=btnl

set\_property -dict { PACKAGE\_PIN M17 IOSTANDARD LVCMOS33 } [get\_ports { coin[2] }]; #IO\_L10N\_T1\_D15\_14 Sch=btnr

set\_property -dict { PACKAGE\_PIN P18 IOSTANDARD LVCMOS33 } [get\_ports { ab[1] }]; #IO\_L9N\_T1\_DQS\_D13\_14 Sch=btnd