西南大學

本科毕业论文（设计）

题　　目 基于 CPLD 的自动售货控制系统设计与实现

学　　院 电子信息工程学院

专　　业 通信工程

年　　级 2011 级

学　　号 222011315220076

姓　　名 冯思远

指 导 教 师 王 飞

成　　绩

2014 年 4 月 30 日

基于 CPLD 的自动售货控制系统设计与实现

冯思远

西南大学电子信息工程学院，重庆 400715

摘要：本文通过分析当前街道自动售货需求，设计出一种基于 CPLD 的多功能自动售货控制系统，从而来满足客户选择商品单价和种类、退币和找零的使用需求，以及维护人员统计找零硬币剩余量、售出金额，控制库存数量的使用需求。功能设计通过原理图输入和 VHDL 语言编程方法实现，仿真通过矢量波形文件实现。设计和仿真使用的软件工具都集成于 Altera 公司生产的 EDA 软件 QUARTUSII 当中。当下载目标文件到 Altera 公司生产的 CPLD 芯片(MAXII EPM1270T144C5)并完成芯片和外围电路的焊接工作后，系统运作良好并实现所有设计的功能。控制系统的实现证实了基于 CPLD 的模块化设计的方法对解决现有问题具有可行性和简易性。

关键词：自动售货；控制系统；QUARTUS II；CPLD；模块化设计；

Design and Implementation of Vending Machine Control System Based on CPLD Programming

FENG Siyuan

College of Electronic Information Engineering, Southwest China University, Chongqing 400715, China

Abstract: By analyzing the current demands of automatic selling in the streets, we design a multifunctional control system so as to meet the requirements of users, such as price and product selection，coin return or change, and the requirements of maintenance staff, e.g., the remaining amount of coins, the sold amount, and the amount of inventory. The functional design is implemented by using the Block Diagram Mode and the VHDL language. The simulation is implemented by Vector Waveform Files. The design and simulation software tools, are integrated in EDA software QUARTUS II which is issued by Altera. The circuit behaves well and all functions are implemented after downloading the program into the EPM1270T144C5 (MAXII device of CPLD) and finishing the work of welding the chip with a peripheral circuit. The successful realization of this design demonstrates that the modularized design method based on CPLD is feasible and simple to solve the existing problems.

Key words:vending machine; control system; QUARTUS II; CPLD; modularized design;

目录

第一章 引言

1.1 自动售货机的发展分析

自动售货机是一种全新的商业零售形式，20 世纪 70 年代从日本和欧美发展起来。由自动售货机的发展趋势来看，它的出现是劳动密集型的产业构造向技术密集型社会转变的产物。大量生产、大量消费以及消费模式和销售环境的变化，要求出现新的流通渠道，诸如超市、百货购物中心等。但由于上述流通渠道人工费用不断上升，场地较为局限以及购物不够便利等因素的制约，使得无人自动售货机作为一种必须的机器应运而生。

我国引进自动售货机是从 1993 年开始，1995 年在北京诞生了我国第一台国产自动售货机，1999 年自动售货机已大规模投入市场使用。虽然与发达国家相比，我国的自动售货机在市场中运营的数量还比较少，但是我国也出现 10 多家自主研发、制造和运营自动售货机的企业。据中国自动售货机市场行情分析的一份资料表明：我国自动售货机市场走势已明朗，即从沿海经济发达地区向内陆各大中城市普及。自动售货机这种全新的零售模式正得到国人的认可和青睐。

中国的自动售货机将发展成为一个庞大的产业。作为一种新型商业形态和广告媒体，自动售货机从 1999 大规模投入中国市场以来，受到了年轻人的欢迎。2004 年之后中国的自动售货机产业实现了快速增长，奥运会和世博会为自动售货机市场带来了巨大商机，2008 年后进入了快速发展阶段。自动售货机将在中国零售业掀起继百货商店、 超市之后的第三次零售业革命[1-2]。

本设计通过分析自动售货普遍的控制管理需求，旨在设计出一种多功能控制系统来满足客户选择商品单价和种类、找零/退币等需求，以及维护人员提醒补充库存、统计售出金额和监视剩余找零数量等需求。

1.2 CPLD 的应用与发展

PLD 发展于上世纪 70 年代，目前已形成了很多不同种类的产品。20 世纪 70 年代末美国 NMI 公司最先推出可编程阵列逻辑(PAL)，器件工作效率高，输出结构种类多，设计十分灵活。但由于种类变得十分丰富，给使用、生产带来不便。20 世纪 80 年代初，

Lattice 公司在 PAL 的基础上发明了通用阵列逻辑(GAL)，其具有可电擦写、可重复编程并且可设置加密等特点[3]。但由于 PAL、GAL 结构过于简单使得它们只能实现规模较小的电路。为弥补 PLD 只能设计小规模电路这一不足，20 世纪 80 年代中期，Lattice、Altera 等公司推出了复杂可编程逻辑器件 CPLD(Complex Programmable Logic Device)，并且其已广泛应用于网络、仪器仪表、汽车电子、数控机床、航天测控设备等方面。

CPLD 从 PAL 和 GAL 器件发展出来，相对来讲规模大，结构复杂，属于大规模集成电路范围。通常来讲，CPLD 是一种用户根据自身需求而自行构造逻辑功能的数字集成电路。其基本设计方法是借助集成 EDA 的软件开发平台，通过原理图、硬件描述语言等方法，生成相应的目标文件，通过下载将程序传送到目标芯片中，实现数字系统的设计。

CPLD 具有编程灵活、集成度高、设计开发周期短、适用范围宽、开发工具先进、设计制造成本低、对设计者的硬件经验要求低、标准产品无需测试、保密性强、价格大众化等特点，可实现较大规模的电路设计，因此被广泛应用于产品的原型设计和产品生产之中。几乎所有应用中小规模通用数字集成电路的场合均可应用 CPLD 器件。CPLD 器件已成为电子产品不可缺少的组成部分，它的设计和应用成为电子工程师必备的一项技能。因此本设计采用 CPLD 作为控制器件，采用模块化设计方法，并通过键盘电路、点阵等外围电路实现系统控制功能。

第二章 总体方案设计

2.1 系统主要功能

自动售货机作为一种完全独立的售货设备，集接受货币、用户自选商品、售出商品和找零等功能于一体[4]。本控制系统是以 CPLD 为主要控制芯片的自动售货控制系统，其包括若干模拟钱币输入按钮、若干单价和种类选择按钮、用户退币/找零按钮、若干数据显示界面、若干功能指示灯。

本系统可分为用户操作界面和维护人员操作界面。在用户操作界面有 5 个独立按键，对应不同类型钱币输入模拟和退币/找零操作模拟。有 4*4 矩阵键盘，代表商品单价和商品种类选择按键，根据用户的意愿确定购买目标，用户可以自由组合。有 2 位 7 段数码管，显示应找零金额数。在维护人员操作界面有 6 位 7 段数码管，显示找零硬币剩余数、输出金额总数。用户、维护人员操作界面共用 8*8 点阵，表示各模块相应指示灯。

本控制系统将实现如下功能：

1.用户使用功能

⑴ 能识别 20 元、10 元、5 元纸币单次输入和 1 元硬币累加输入；

⑵ 当输入金额满足所需钱数时相应的“可选择单价”指示灯亮；

⑶ 当检测到钱币输入信号时，显示相应输入钱数；

⑷ 选择不同单价商品时相应的“已选择单价”指示灯亮；

⑸ 在商品单价选择阶段可记录最后一次选择类别，避免误选操作；

⑹ 完成商品单价选择后进行商品种类选择时，输出“出库”信号并显示余额；

⑺ 当余额大于 0 时等待用户继续购买或者退币操作；

⑻ 当用户按下找零/退币按钮时执行找零或退币操作，并重置系统；

2. 系统控制功能

⑴ 当用户投币后锁定输入钱币类型，使其他类型钱币输入无效；

⑵ 根据用户输入的钱币类型控制纸币、硬币投币口的开闭；

⑶ 当用户投币后执行取消购买操作时，产生相应钱币的退币信号；

⑷ 当存在消费且应找零数为 0 时系统自动复位；

⑸ 设置每类商品初始数量，当完成一次购买操作后库存数量减 1，当库存数量为 0

时控制相应的“售空”指示灯亮，并使用户无法选择购买该商品；

(6) 设置 1 元找零硬币初始数量，并实时统计找零硬币剩余数量，且当硬币数较低时输出“硬币数量不足”信号；

(7) 设置“售出金额总数”数值显示，当维护人员在添加商品和取走钱币时可以显示售货机在两次时间间隔内收入总金额数；

(8) 当维护人员切断芯片电源后，初始化库存数量和找零硬币数量；

2.2 系统工作流程

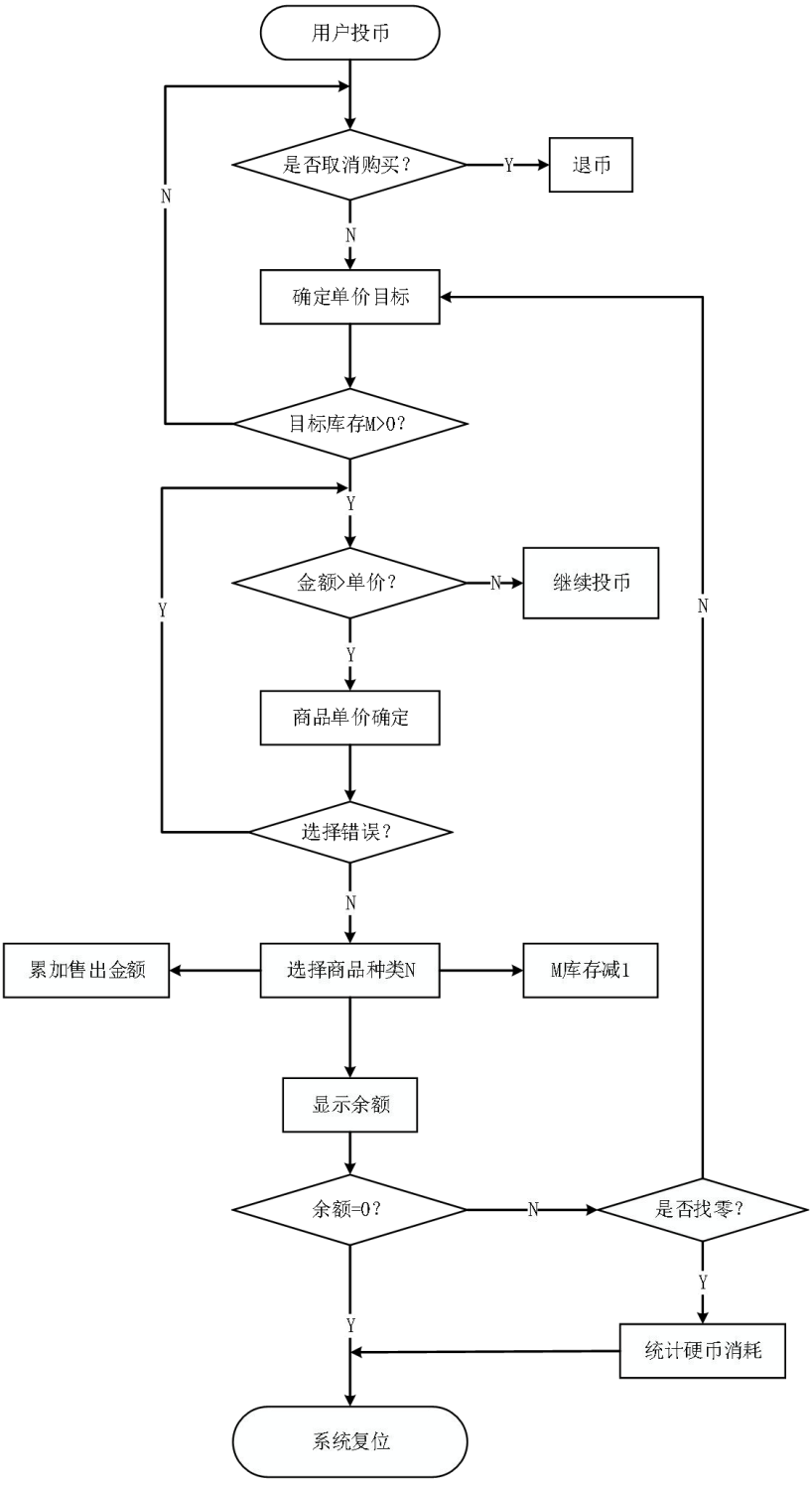


图 2.1 自动售货控制系统工作流程图

Fig. 2.1 The workflow chart of vending machine control system

根据图 2-1 所示流程，自动售货控制系统一次工作流程如下：芯片通电后系统保持待命状态。当用户进行购买操作时，根据用户投入的不同钱数显示已投入金额，若用户取消购买按下退币/找零按钮则执行返还相应钱币操作。在用户确定购买目标商品时扣除相应钱数，此时已投币数更新为应找零数，同时累加已售出金额并统计库存剩余量。当用户购买完毕按下退币/找零按钮时，统计剩余找零硬币数量。当应找零数量为 0 时系统自动复位，等待下次购买流程开始。

2.3 软硬件方案确定

2.3.1 硬件方案确定

根据上述提出的功能要求，自动售货控制系统的控制芯片有多种方案可供选择。可编程芯片最广泛使用的有现场可编程门阵列(FPGA)和复杂可编程逻辑器件(CPLD)。除此之外嵌入式微处理器(MPU/MCU/DSP/SoC)也可以作为控制芯片使用。

本系统设计采用了 CPLD 中较为典型的 MAXII 系列 CPLD 芯片作为控制芯片。该芯片较于 FPGA 而言系统断电后，编程信息不丢失，更适合完成各种算法和组合逻辑，并且时序延迟是均匀且可预测的[5]。相较于嵌入式微处理器而言具有更加丰富的端口资源，且处理速度更快。

本系统设计使用的是 Altera 公司生产的 MAXII 系列 EPM1270T144C5。该系列产品具有低成本，低功耗、瞬时上电，非易失体系结构、高容量、高性能等特点。该型号芯片封装有 144-pin TQFP、256-pin FBGA、256-pin MBGA 三种封装格式，本设计采用的是 144-pin TQFP 此种封装。芯片引脚结构图如图 2.2 所示。

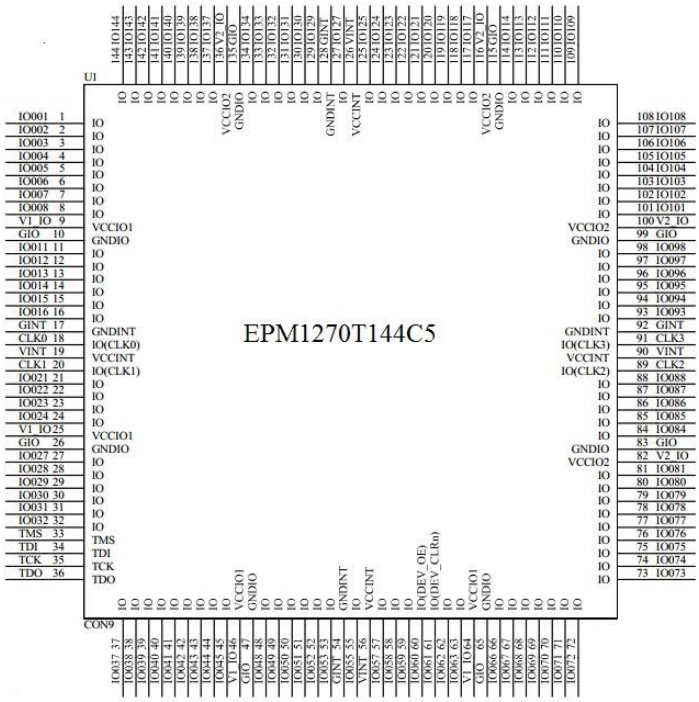


图 2.2 EPM1270T144C5 引脚结构图

Fig. 2.2 EPM1270T144C5 pin structure

2.3.2 软件方案确定

硬件描述语言 HDL 是 EDA 技术的重要组成部分，目前常用的 HDL 主要有 VHDL、Verilog HDL、SystemVerilog 和 SystemC。其中 Verilog、VHDL 在现在 EDA 设计中使用最多，也得到几乎所有的主流 EDA 工具的支持。

VHDL(Very-High-Speed Integrated Circuit Hardware Description Language)于 1983 年由美国国防部发起创建，并在 1987 年作为“IEEE 标准 1076”发布。从此 VHDL 成为硬件描述语言的业界标准之一。此后，VHDL 在电子设计领域得到了广泛应用，并与 Verilog 一起逐步取代了其他的非标准硬件描述语言。

Verilog HDL 是硬件描述语言一种，用于数字电子系统设计。它允许设计者用它来进行各种级别的逻辑设计，可以用它进行数字逻辑系统的仿真验证、时序分析、逻辑综合。它是目前应用最广泛的一种硬件描述语言。

Verilog HDL 和 VHDL 作为描述硬件电路设计语言，其共同点在于：能形式化地抽象表示电路的行为和结构，能支持逻辑设计及关于中层次与范围的描述，可借用高级语言的精巧结构来简化电路行为的描述，具有电路仿真与验证机制从而来保证设计的正确性。与 Verilog 相比，VHDL 有下列优势：

（1） 语法比 Verilog 严谨，通过 EDA 工具自动语法检查，易排除设计中的许多疏忽。

（2） 有很好的行为级描述能力和一定的系统级描述能力，而 Verilog 建模时，行为与系统级抽象及相关描述能力不及 VHDL。

常用的集成 EDA(Electronic Design Automation) 开发环境有 Altera 公司的 MAX+PLUS II、Quartus II 等。Quartus II 适用于大规模逻辑电路设计。其界面友好，集成化程度高，易学易用。设计流程主要为设计输入、设计编译、设计仿真、和设计下载等过程。该软件支持多种编辑输入方法，包括图形编辑输入法、VHDL、Verilog HDL 等文本编辑输入法[6-8]。

综上所述，本设计最终确定采用 Quartus II 作为集成开发环境，系统功能设计通过 VHDL 语言和原理图输入方法实现，仿真通过 Quartus II 内置波形仿真工具实现。

第三章 硬件设计

3.1 时钟频率电路设计

由于软件设计中的模块部分功能需要用到时序节拍，因此硬件电路中采用 50MHz 有源晶振作为系统主时钟。本设计使用了 EPM1270T144C5 时钟引脚中的 GCLK0，通过连接芯片的全局时钟网络来驱动整个设备[9]。时钟频率电路如图 3.1 所示。

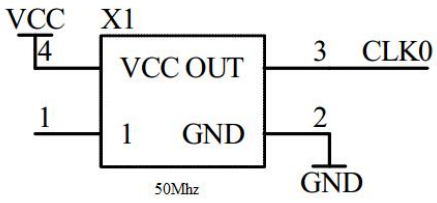


图 3.1 时钟频率电路

Fig. 3.1 Clock circuit

3.2 键盘电路设计

键盘是最常用也是最主要的输入设备，通过键盘可以向系统发出命令、输入数据。键盘应用主要是独立键盘和矩阵键盘。

本设计将购买操作分步，通过 4*4 矩阵键盘表示 8 个单价选择按键和 8 个种类选择按键，代替独立的 64 个相应单价不同种类按键，同时由于采用矩阵方式，因此购买功能仅用了 8 个 I/O 端口，在避免误操作的同时节省了三分之四的 I/O 端口。不同类型钱币输入信号、退币/找零操作通过另外 5 个独立按键产生相应电平信号模拟输入。

本设计电路输入信号设定为低电平有效，矩阵键盘、独立按键的上拉电阻阻值设置为 4.7K，设置低电平有效且限流避免烧毁芯片[10]。按键电路如图 3.2 所示。

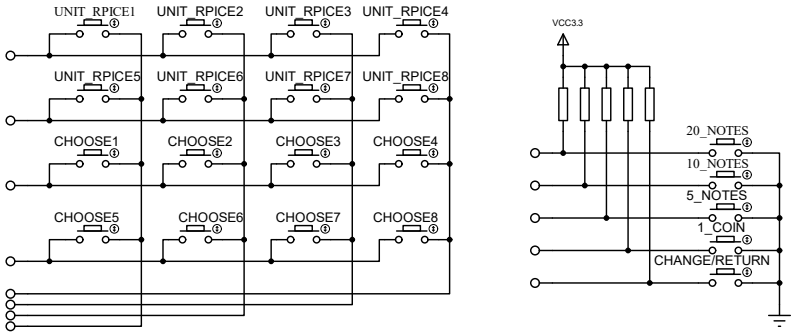


图 3.2 按键电路

Fig. 3.2 Keyboard circuit

3.3 电源电路设计

电源电路设计主要解决如何将常用的5V 直流电压转换为芯片工作所需的3.3V 直流电压。本设计主要元件为 1A 低压差稳压器 AMS1117-3.3[11]，通过该元件和若干电解电容、电阻实现将 5V 直流电转换为 3.3V 直流电以供系统使用。

此外，通过控制系统电源的通断可以实现本设计的硬件复位，即开关可初始化库存数和 1 元找零硬币数，因此可视为维护人员复位接口。电源电路如图 3.3 所示。

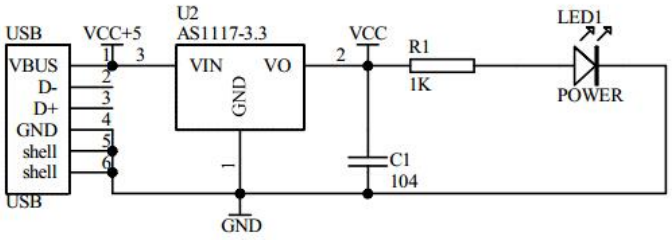


图 3.3 电源电路

Fig. 3.3 System power circuit

3.4 显示电路设计

显示电路由数据显示电路和指示灯显示电路组成。数据显示功能由 8 位 7 段数码管实现，当进行相应操作时，8 位 7 段数码管依次排序显示应找零数、找零硬币剩余数、已售出金额总数。指示灯显示功能由 8*8 点阵实现，系统根据不同控制信号控制点阵相应 LED 灯显示。显示电路如图 3.4 所示。

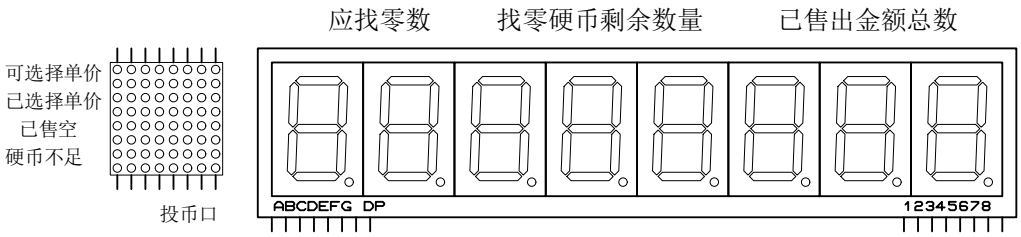


图 3.4 显示电路

Fig. 3.4 Display circuit

第四章　软件设计

自动售货控制系统的软件设计采用模块化设计，具有两个辅助模块和六个控制模块。辅助模块有：消抖模块、分频模块、控制模块、数据显示模块。功能模块有：金额输入模块、商品选择模块、找零模块、库存控制模块、钱币管理模块、售出金额控制模块。

4.1 辅助模块

4.1.1 消抖模块

当我们进行按键操作时，由于机械触点的弹性及电压突跳等原因，在触点闭合或开启的瞬间会出现电压抖动，实际应用中如果不进行处理将会造成误触发。因此为了准确识别按键次数，对每一次按键只作一次响应，因此要去除抖动，消除按键过程中产生的“毛刺”现象。

本设计基于数字电路的复杂可编程逻辑器件 CPLD，简单的 RC 消抖电路不便于在数字电路中集成，即使能集成，所占的版图面积也较大。因此主要考虑使用 D 触发器实现的消抖电路[12-13]。

一般人按键时间为 100ms，按下时间可估算为 50ms，不稳定噪声一般在 5ms~10ms。设置采样频率 CLK 为 100Hz，当按键操作时间大于或等于 CLK 时钟周期的三倍才输出一个正脉冲，因而避免了按键抖动而引起的计数错误。

消抖电路原理图如图 4.1 所示。

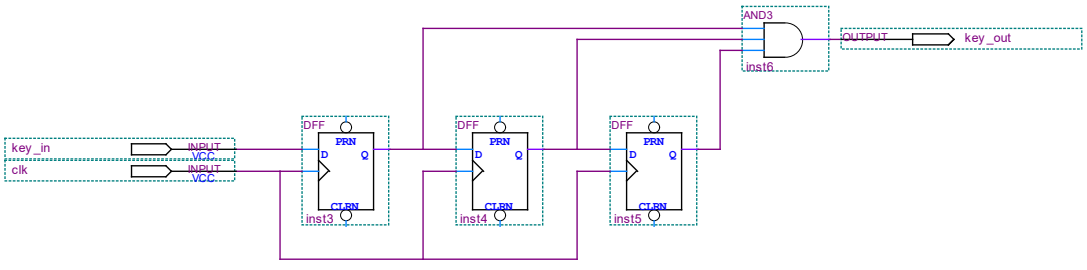


图 4.1 消抖模块原理图

Fig. 4.1 Schematic diagram of debouncing module

4.1.2 分频模块

本系统设计需要时钟信号的模块有消抖模块、售出金额控制模块、找零模块中的自动清零子程序。在此分析具体所需频率需求。

消抖模块：需求分析在 4.1.1 小结已详述，其所需频率为 100Hz，在此不再赘述。

售出金额控制模块：要求数码管动态显示，则要利用人的视觉暂留现象。一个数码管所要显示的字符需要在 1 秒内点亮 24 次以上才能感觉上认为该数码管没有熄灭。找零金额、售出金额为 8 位 BCD 码显示的总和，因此 8 只数码管要实现这种效果，则频率至少应为 192Hz 以上[14-15]。为了减少闪烁现象，达到较好的显示效果，取频率为 1KHz，每个数码管每秒显示 125 次。

自动清零子程序：要求在已存在消费且应找零数目为 0 时累计数个脉冲后输出瞬时低电平信号，使得相应模块完成复位操作，该过程应该在数十毫秒内完成。

综上所述，将系统 50MHz 主时钟分频为 100Hz 和 1KHz 能满足本设计要求。

分频模块如图 4.2 所示。

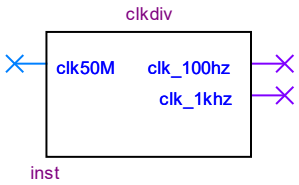


图 4.2 分频模块

Fig. 4.2 Clock-devision module

4.1.3 控制模块

控制模块实现功能包括矩阵键盘识别控制和指示灯控制。矩阵键盘的行及列的扫描控制和译码是设计的主要部分[16]。矩阵键盘扫描控制采用逐行扫描查询法，将行线按照时钟频率依次置低电平，然后检测列线状态。如果有按键按下，则该列的电平输出低电平，由此确定按键位置。当识别按键成功时产生相应译码信号以控制商品选择模块。点阵控制同样采用逐行扫描方法，根据各个模块的输入信号控制点阵相应 LED 灯亮灭。

控制模块如图 4.3 所示。

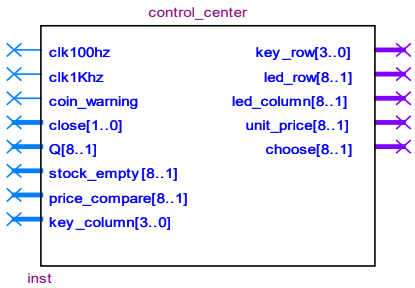


图 4.3 控制模块

Fig. 4.3 Control-center module

控制模块接口名称及含义如表 4.1

I/O 名称	接口含义
clk100/1Khz	100/1Khz 周期信号输入
coin_warning/close	硬币数量不足/关闭投币口信号输入
Q/stock_epmty/price_compare	已选择单价/库存/相应单价对比信号输入
key_column/row	键盘列/行信号输入/输出
led_row/column	点阵行/列信号输出
unit_price/choose	单价/种类选择信号译码输出

表 4.1 控制模块 I/O 接口名称及含义

Table 4.1 Names and meanings of I/O interface for control-center module

4.1.4 数据显示模块

数据显示模块接收来自找零模块、钱币管理模块、售出金额控制模块相应的 BCD 码信号，并控制 8 位数码管段码和位码随着 1Khz 频率实现应找零数、找零硬币剩余数、已售出金额数动态显示。动态显示模块如图 4.4 所示。

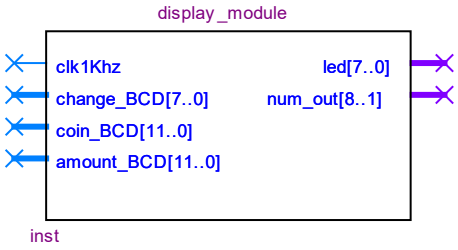


图 4.4 显示模块

Fig. 4.4 Display-module

数据显示模块接口名称及含义如表 4.2

I/O 名称	接口含义
clk1Khz	1Khz 周期信号输入
change/coin_BCD	应找零金额/1 元找零硬币剩余数量 3 位 BCD 码输入
amount_BCD	售出金额控制 2 位 BCD 码输出
led/ num_out	数码管 8 位段码/位码控制输出

表 4.2 显示模块 I/O 接口名称及含义

Table 4.2 Names and meanings of I/O interface for display-module

4.2 功能模块

4.2.1 金额输入模块

金额输入模块为用户使用功能实现模块。能够识别输入钱币金额并统计输出钱币总额。因为商品最高单价为 8 元，因此每次购买只能输入同种类型钱币，且不考虑 20 元、10 元和 5 元纸币的累加输入。本模块主要由组合逻辑电路和时序逻辑电路组合而成[17]。金额输入模块如图 4.5 所示。

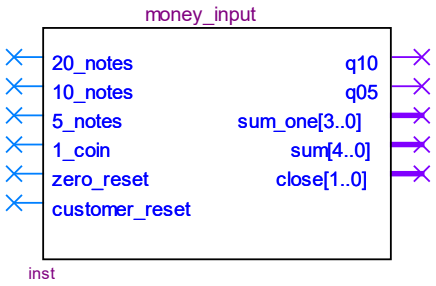


图 4.5 金额输入控制模块

Fig. 4.5 Money-input control module

1. 金额输入模块接口名称及含义如表 4.3

I/O 名称	接口含义
20/10/5_notes	20 元、10 元、5 元纸币信号输入,低电平有效
1_coin	1 元硬币信号输入，低电平有效
zero/ customer _reset	自动清零/用户退币找零信号输入，低电平有效
q10/q05	10/5 元纸币已投入信号输出
sum	5 位二进制钱币数值累加总和输出
sum_one	4 位二进制 1 元硬币累加总和输出
close	关闭硬币或纸币投币口，低电平驱动

表 4.3 金额输入控制模块 I/O 接口名称及含义

Table 4.3 Names and meanings of I/O interface for money-input control module

2. 金额控制模块功能

（1）检测 20/10/5_notes 电平，当用户投入 20 元纸币时产生下降沿跳变，转换为 5 位二进制数“10100”。当用户投入 10 元纸币时产生下降沿跳变，转换为 5 位二进制数“01010”，且 q10＝**1**。当用户投入 5 元纸币时产生下降沿跳变，转换为 5 位二进制数“00101”，且 q05＝**1**。同时通过门电路使得所有类型钱币再次输入无效。

（2）检测 1_coin 电平，当存在上升沿跳变时计数，并输出 4 位二进制计数累加值，同时使得纸币类型输入无效。当累加值等于“1000”时，所有类型钱币输入无效。

（3）接收来自 3 种不同钱币相应二进制数目并通过接口 sum 输出。

（4）检测 zero_reset 电平，当 zero_reset＝**0** 时自动复位。

（5）检测 customer_reset 电平，当用户按下退币按钮而产生下降沿跳变时，完成金额控制模块的复位操作。

（6）当投入钱币为纸币，则 close＝“00”。当投入钱币为硬币，则 close＝“01”，当 1 元硬币累计输入 8 次时，close＝“00”。关闭投币口功能由“投币口”指示灯模拟显示，通过观察指示灯状态验证该功能是否实现。

3. 金额控制模块理想状态下功能仿真结果如图 4.6 所示。

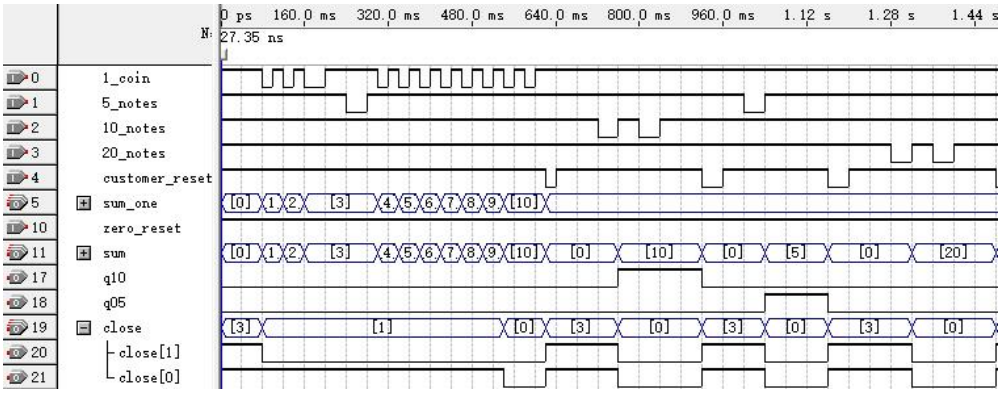


图 4.6 金额控制模块功能仿真结果

Fig. 4.6 Functional simulation of money-input control module

金额控制模块原理图见附录。

4.2.2 商品选择模块

商品选择模块为用户操作控制和系统控制辅助二者结合模块。具有对比输入金额、检测库存、防止用户误操作、产生最终决定购买瞬时电平、产生出库瞬时电平等多项功能。商品选择模块主要由基于逻辑电路的价格选择子模块及基于 VHDL 编程的选择控制子模块、出库控制子模块结合实现，商品单价、种类数量可随具体需求更改。模块如图 4.7 所示。

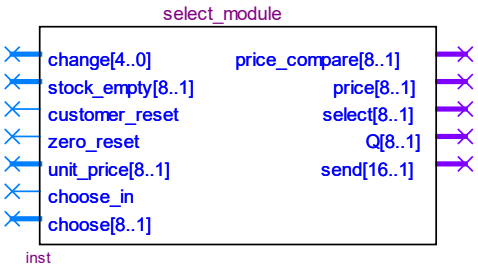


图 4.7 商品选择模块

Fig. 4.7 Product selection module

1. 商品选择模块接口名称及含义如表 4.4

I/O 名称	接口含义
change	5 位二进制找零金额数输入
stock_empty	8 种商品单价库存状态信号输入
zero /customer_reset	自动清零/用户退币找零信号输入，低电平驱动
unit_price/ choose	8 种商品单价/种类选择输入
choose_in	商品种类选择总输入，高电平驱动
price_compare	余额已满足相应单价信号输出
select	相应单价最终决定购买信号输出
Q	8 种单价已选择信号输出
send	64 种商品行线、列线控制信号输出

表 4.4 商品选择控制模块 I/O 接口名称及含义

Table 4.4 Names and meanings of I/O interface for select-module

2. 选择控制子模块功能

（1）检测接口 change，当数目满足相应单价金额时 price_compare 输出相应高电平。

（2）检测 Q 中 **1** 的数量，当数量大于 1 时判定为误操作，输出复位信号到价格选择子模块，Q 输出相应低电平，用户可以重新选择。

3. 价格选择子模块功能

（1）检测 price_compare 电平，当相应位为高电平时相应价格选择操作有效，且控制相应的“可选择单价”指示灯亮。

（2）检测 stock_empty 电平，当相应位为高电平时相应价格选择操作有效。

（3）检测 unit_price 电平，当为下降沿跳变时，相应的 Q 输出持续高电平，控制对应的“已选择单价”指示灯亮。

（4）检测 choose_in 电平，当为高电平且以上相应接口均为高电平时，select 输出相应高电平信号到找零模块，并通过或非门输出瞬时低电平到商品选择模块的复位接口，因此 select 随后马上复位全部输出低电平。

4. 出库控制子模块功能

检测 choose 和 Q 电平，本设计设定有 8 种不同单价，每种单价下有 8 类不同商品，因此总共有 64 种不同商品可供用户选择。当用户在完成单价选择步骤，按下相应商品种类选择按钮即可确定用户最终选择购买何种单价下的哪类商品，此时通过 send 接口输出相应瞬时行线、列线出货信号。

商品选择模块原理图、源程序见附录。

4.2.3 找零模块

找零模块是系统控制功能主要模块。具有识别 4 位二进制金额总数、随着商品选择模块的瞬时高电平信号实现相应累减操作、输出二进制应找零数并转换成相应的 BCD 码输出等功能[18]。找零模块由 VHDL 编程实现，如图 4.8 所示。

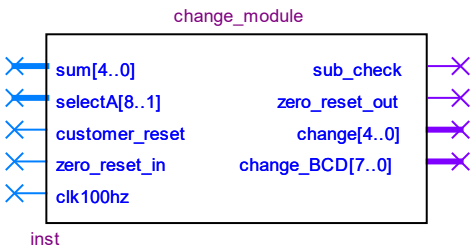


图 4.8 找零模块

Fig. 4.8 Change-module

1. 找零模块接口名称及含义如表 4.5

I/O 名称	接口含义
sum	5 位二进制钱币数值累加总和输入
selectA	相应单价最终决定购买信号输入
zero /customer_reset	自动清零/用户退币找零信号输入，低电平驱动
clk100hz	100hz 周期信号输入
sub_check	已存在消费信号输出
zero_reset_out	自动清零信号输出
change	5 位二进制找零金额数输出
change_BCD	应找零金额 3 位 BCD 码输出

表 4.5 找零显示模块 I/O 接口名称及含义

Table 4.5 Names and meanings of I/O interface for change-control module

2. 找零模块功能

（1）检测 sum 二进制总和以及 selectA 电平跳变，当发生下降沿跳变时累计减去相应已成功交易金额，并通过 change 输出 5 位二进制应找零数，同时转换为 2 位 BCD 码通过 change_BCD 串行输出。

（2）检测 customer_reset 电平，当用户按下退币按钮而产生下降沿跳变时，完成找零模块的复位操作。

（3）当用户完成购买流程且应找零数目为 0 时，找零模块的 zero_reset_out＝**0**，zero_reset_in＝**0**，且其余模块的 zero_reset＝**0**，实现本模块和其余模块的自动复位。

（4）检测 clk100hz 电平，配合自动清零子程序输出清零信号。

（5）当用户完成一次购买流程时，sub_check＝**1**，未购买或复位后 sub_check＝**0**。

3. 找零模块理想状态下功能仿真结果如图 4.9 所示。

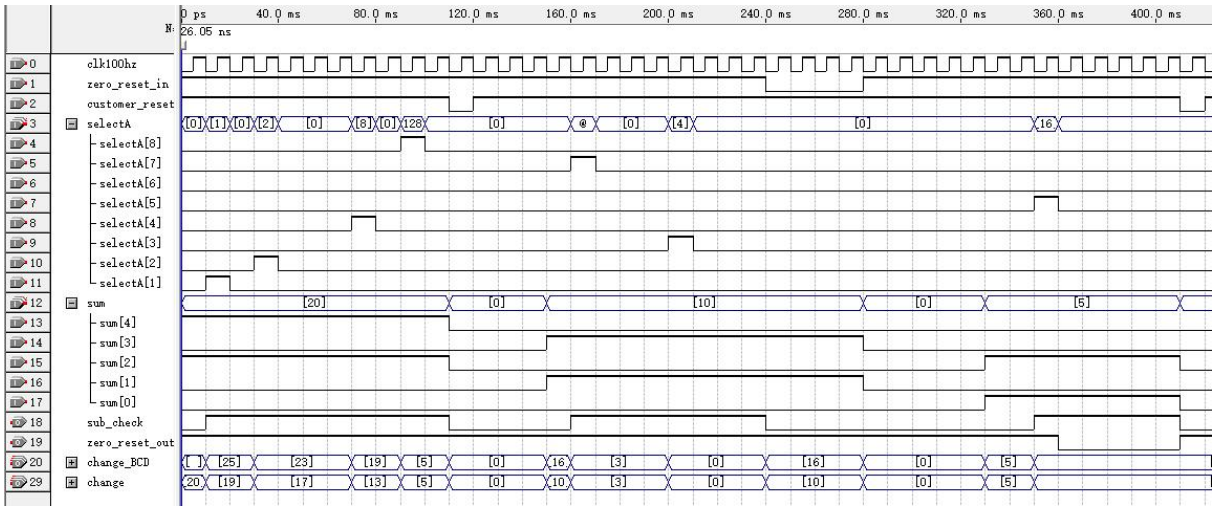


图 4.9 找零显示模块功能仿真结果

Fig. 4.9 Functional simulation of change-control module

找零模块源程序见附录。

4.2.4 库存控制模块

库存控制模块是系统控制辅助模块，实时检测不同单价商品各自剩余数，当该单价商品剩余数为 0 时输出“售空”指示灯，并使得用户无法购买该单价商品。本模块由 VHDL 编程实现，初始库存数目可根据不同使用要求而更改，以辅助商品模块实现库存管理功能。库存控制模块如图 4.10 所示。

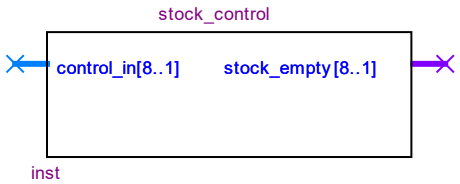


图 4.10 库存模块

Fig. 4.10 Stock-control module

1. 库存控制模块接口名称及含义如表 4.6

I/O 名称	接口含义
control_in	相应单价最终决定购买信号输入
stock_empty	8 种商品单价库存状态信号输出

表 4.6 库存控制模块 I/O 接口名称及含义

Table 4.6 Names and meanings of I/O interface for stock-control module

2. 库存控制模块功能

（1）设定不同单价商品各自初始库存值为 80，此时输出相应高电平信号到商品选择模块输入接口 stock_empty。

（2）当任意单价商品库存为 0 时，stock_empty 输出相应低电平信号到商品选择模块输入接口，并控制相应“售空”指示灯亮。

3. 库存控制模块理想状态下功能仿真结果如图 4.11 所示。

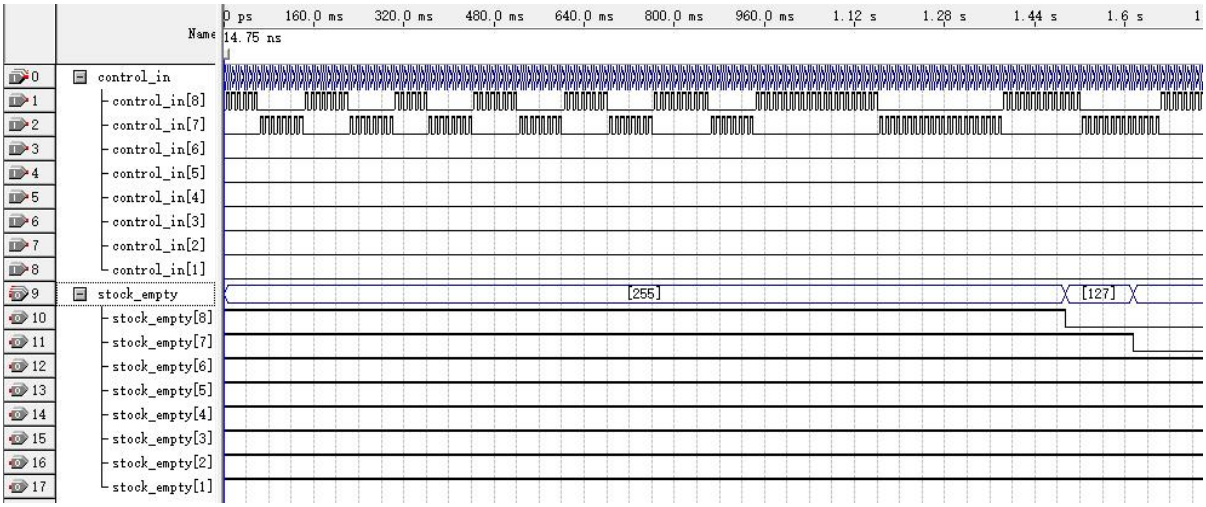


图 4.11 库存控制模块功能仿真结果

Fig. 4.11 Functional simulation of stock-control module

库存控制模块源程序见附录。

4.2.5 钱币管理模块

钱币管理模块是系统控制功能实现模块。能够统计用于找零的硬币剩余数量、控制相应指示灯以及判断有无消费，实现用户退币功能。本模块由 VHDL 编程实现，初始硬币剩余数量可根据不同使用要求而更改，模块如图 4.12 所示。

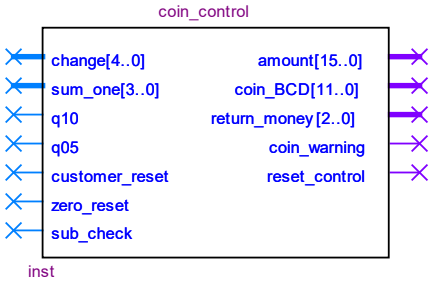


图 4.12 钱币管理模块

Fig. 4.12 Coin-control module

1. 钱币管理模块接口名称及含义如表 4.7

I/O 名称	接口含义
change	5 位二进制找零金额数输入
sum_one	4 位二进制 1 元硬币累加总和输入
q10/ q05	10/5 元纸币已投入信号输入
zero/ customer _reset	自动清零/用户退币找零信号输入，低电平有效
sub_check	已存在消费信号输入
coin_BCD	1 元找零硬币剩余数量 3 位 BCD 码输出
return_money	20 元、10 元、5 元纸币退币信号输出
coin_warning	硬币数量不足信号输出
reset_control	控制用户找零信号输出

表 4.7 钱币管理模块 I/O 接口名称及含义

Table 4.7 Names and meanings of I/O interface for money-control module

2. 钱币管理模块功能

（1）设定 1 元找零硬币初始数量为 200，此时接口 coin_warning＝**0**，此时“硬币不足”指示灯保持灭状态。

（2）检测 sub_check 和 customer_reset 电平，当 customer_reset＝**0** 时，若 sub_check＝**1** 则执行找零操作，反之执行退币操作。

（3）当执行退币操作时：

①若 change＝“10100”则判断未消费，应返还纸币 20 元，return20 输出瞬时高电平。

②若 change＝“01010”且 q10＝ 1 、sub_check＝**0**，则判断未消费，应返还纸币 10 元，return10 输出瞬时高电平。

③若 change＝“00101”且 q05＝ 1 、sub_check＝**0**，则判断未消费，应返还纸币 5 元，return05 输出瞬时高电平。

（4）当执行找零操作时，若 sum_one＞“0000”，则输入类型为硬币，因此

关系式记为：

$$\mathbf{amount_{now}=amount_{before}+sum_{one}-change}$$

当执行找零操作时，若 sum_one＝“0000”，则输入类型为纸币，因此

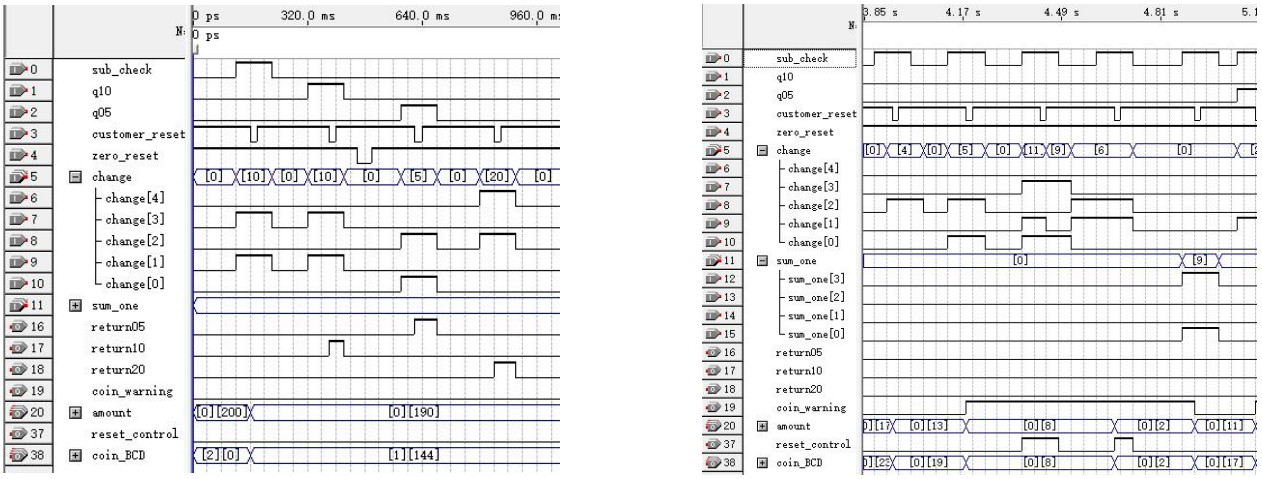
关系式记为：

$$\mathbf{amount_{now}=amount_{before}-change}$$

（5）当 $amount_{now}$≤10 时，coin_warning＝**1**，控制“硬币不足”指示灯亮。

当 $amount_{now}$＜change 时，若投币类型为硬币，则退币操作随时有效，若投币类型为纸币且存在消费则找零操作无效，但可继续购买直到 $amount_{now}$≥change。

（6）将累加总和转换为 3 位 BCD 码，并将 3 位 BCD 通过 coin_BCD 串行输出。



(a) (b)

3. 钱币管理模块理想状态下功能仿真结果如图 4.13 所示。

图 4.13 钱币管理模块功能仿真结果

(a)硬币数量大于 10 时仿真图 (b)硬币数量小于 10 时仿真图

Fig. 4.13 Functional simulation of coin-control module

(a)When number of coins are more than 10 (b)When number of coins are less than 10

库存控制模块源程序见附录。

4.2.6 售出金额控制模块

售出金额统计模块是系统控制功能实现模块。当商品选择模块输出瞬时高电平时累加相应单价钱数，但不会随着用户找零/退币操作清零，以便为维护人员显示售货机在两次维护时间间隔内收入总金额数[19]。本模块由 VHDL 编程实现，如图 4.14 所示。

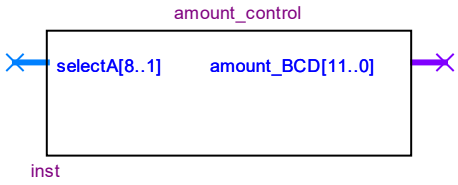


图 4.14 售出金额控制模块

Fig. 4.14 Amount-control module

1. 售出金额控制模块接口名称及含义如表 4.8

I/O 名称	接口含义
select	相应单价最终决定购买信号输入
amount_BCD	售出金额控制 2 位 BCD 码输出

表 4.8 售出金额控制模块 I/O 接口名称及含义

Table 4.8 Names and meanings of I/O interface for amount-control module

2. 售出金额控制模块功能

（1）检测 select 电平跳变，当发生下降沿跳变时累加相应已成功交易金额。

（2）将累加总和转换为 3 位 BCD 码，并将 3 位 BCD 通过 amount_BCD 串行输出。

3. 售出金额控制模块理想状态下功能仿真结果如图 4.15 所示。

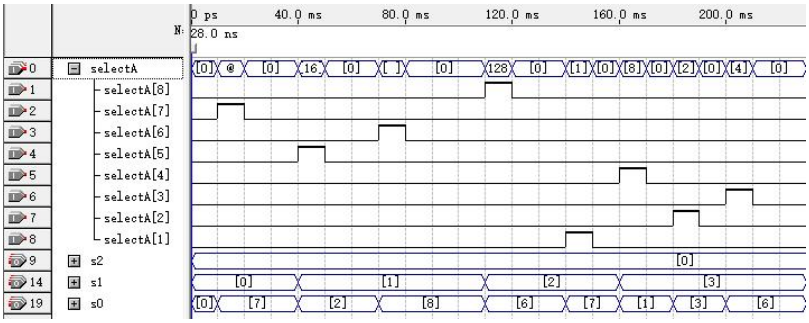


图 4.15 售出金额控制模块功能仿真结果

Fig. 4.15 Functional simulation of amount-control module

售出金额控制模块源程序见附录。

第五章　系统综合调试

5.1 系统已实现功能

当完成上述硬件和软件的设计后，我们应通过实际电路验证设计能否在实际电路中运行。通过观察实际运行结果来验证设计的可行性和完整性。功能验证主要包括：

1. 观察相应数据是否正常且正确显示。
2. 观察相应指示灯是否正确指示。
3. 观察按下相应按键是否能被正确译码和控制。
4. 观察相应操作是否存在漏洞和不足。

5.1 系统运行效果

1. 通电时初始状态显示

图 5.1 通电初始状态

Fig. 5.1 The initial state of electricity

通电后数码管依次显示此时应找零数为 0、找零硬币剩余数为 200，售出金额总数为 0，如图 5.1 所示。模拟投币口状态指示灯亮。初始状态显示符合本设计功能需求。

2. 投币、单价选择操作

图 5.2 10 元纸币输入

Fig. 5.2 10 notes input

图 5.4 单价错误选择（2 元）

Fig. 5.4 Error selection of unit price (2 yuan)

图 5.3 单价正确选择（7 元）

Fig. 5.3 Correct selection of unit price (7 yuan)

按下“10 元纸币”模拟按钮，此时应找零数为 10，如图 5.2。模拟投币口状态指示灯全灭，“可选择单价”指示灯全亮。此时误选单价类型为“2 元”商品，重新按下单价类型为“7 元”商品，如图 5.4、图 5.3。“已选择单价”指示灯依次正确显示。投币、单价选择操作符合本设计功能需求。

3. 种类选择、找零操作

图 5.5 商品成功购买（种类 8）

Fig. 5.5 Purchase success (choose 8)

图 5.6 进行找零操作后状态

Fig. 5.6 The status after change operation

按下商品种类为“8”的商品，此时应找零数为 3。1-3 元区间的“可选择单价”指示灯亮，售出金额更新为 7 元，如图 5.5。此时按下“找零/退币”按钮，此时应找零数为 0。“可选择单价”指示灯全灭，找零硬币剩余数更新为 197，如图 5.6 所示。种类选择、找零操作符合本设计功能需求。

4. 库存控制显示

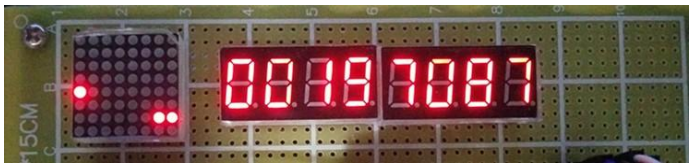


图 5.7 库存不足指示（1 元）

Fig. 5.7 Indication of inventory shortage (1 yuan)

继续上述操作，当重复执行购买“1 元”单价商品 80 次后，1 元商品“售空”指示灯亮。此时由于模拟操作期间无找零操作，因此找零硬币剩余数保持为 197，售出金额更新为 87，如图 5.7 所示。库存控制显示符合本设计功能需求。

5. 找零硬币控制

图 5.8 找零硬币不足时禁止找零

Fig. 5.8 Prohibition of change when coin is insufficient

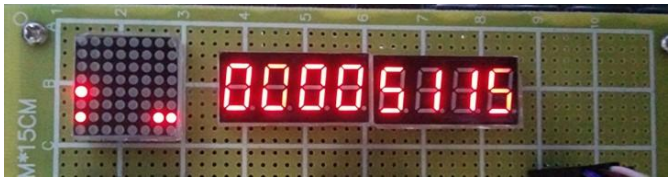


图 5.9 找零硬币不足指示（1）

Fig. 5.9 Indication of change coin shortage（1）

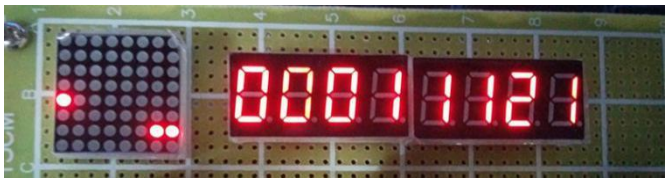


图 5.10 找零硬币不足指示（2）

Fig. 5.10 Indication of change coin shortage（2）

当找零硬币剩余数小于应找零数时，找零操作无效，但可继续购买直到找零硬币剩余数大于或等于应找零数，如图 5.8。当找零硬币剩余数小于或等于 10 时，找零硬币不足指示灯亮，如图 5.9。若下次购买流程输入钱币类型为纸币，交易成功后累加相应硬币数，如图 5.10。

总结

本系统成功地实现了最初所设想的自动售货的控制功能，通过实际电路测试，可以看出功能实现情况非常理想。本设计通过 EDA 软件开发平台 QUARTUSII 设计了一种可满足客户选择商品单价和种类、找零/退币等需求，以及维护人员提醒补充库存、统计售出金额和监视剩余找零数量等需求的多功能自动售货控制系统。通过观察外围硬件电路，如 8 位数码管和 8*8 点阵的实际运行结果，成功验证了设计的可行性和完整性。

系统在保证正常运行的情况下还达到了最初设计的功能。本次毕业设计通过 EPM1270T144C5，实现了基于 CPLD 的自动售货控制系统，将软件设计及仿真和硬件电路布设成功结合于一体。本系统主要完成了以下功能：

用户使用功能：

- 能识别 20 元、10 元、5 元纸币单次输入和 1 元硬币累加输入；
- 当输入金额满足所需钱数时亮相应的“允许购买”指示灯；
- 当检测到钱数输入信号时，显示相应输入钱数；
- 选择不同单价商品时相应的“已选择单价”指示灯亮；
- 在商品单价选择阶段可记录最后一次选择类别，避免误选操作；
- 完成商品单价选择后进行商品种类选择时，输出“出库”信号并显示余额；
- 当余额大于 0 时等待用户继续购买或者退币操作；
- 当用户按下找零/退币按钮时执行找零或退币操作，并重置系统；

系统控制功能：

- 当用户投币后锁定输入钱币类型，使其他类型钱币输入无效；
- 当用户投币后执行取消购买操作时，产生相应钱币的退币信号；
- 当存在消费且应找零数为 0 时系统自动复位；
- 设置每类商品初始数量，当完成一次购买操作后库存数量减 1，当库存数量为 0 时控制相应的“售空”指示灯亮，并使用户无法选择购买该商品；
- 设置 1 元找零硬币初始数量，当硬币数较低时输出“硬币数量不足”信号；
- 设置“售出金额总数”数值显示，当维护人员在添加商品和取走钱币时可以显示售货机在两次时间间隔内收入总金额数；
- 当维护人员通断芯片电源后，初始化库存数量和找零硬币数量。

本设计系统将实际识别钱币和控制出库等实际复杂操作功能通过输入或输出相应电平模拟或表现。因此本论文所表述的关于自动售货控制系统的设计和实现，而非自动售货机的设计和实现是合理的定位及表述。

本系统设计的重点在于软件设计，系统的主干功能通过原理图输入方式实现。原理图输入方法的优点在于直观易用且稳定性较高，由功能强大、分门别类的期间库支撑。但因为期间库元件通用性差，导致其移植性差，因此如若要实现某些细节功能，原理图设计的方法显得不够灵活和全面，此时需要通过 VHLD 语言来完成。本系统中 VHDL 设计的模块大多采用串行处理的方式，若在某一环节出错，则整个系统就无法正常运作。通过原理图输入和 VHDL 语言结合的模块化设计方法可以大幅度简化系统层次结构，节省芯片储存空间，并且提高系统的稳定性。

通过这次毕业设计，我有了非常大的收获。不仅熟悉和掌握了常见数字电路芯片的使用以及掌握 VHDL 语言的编写，并且在实现软硬件结合的这个阶段里，培养了个人分析和纠正错误以及学习阅读的能力，从而为成为一名硬件工程师打下了坚实基础。

本系统设计还成功证明基于 CPLD 的模块化设计方法对解决现有问题是可行且简易的。本系统设计在功能上还可以有比较大的提升潜力，主要归纳为以下几点：

(1) 系统显示采用液晶显示，使金额、库存等信息通过更智能和人性化的方式显示。

(2) 硬币找零信号实现，能够根据余额输出简单、稳定且易实现的电平信号。

(3) 系统数据云服务化，能通过网络上传相关信息，智能分析客户购买需求。

(4) 增加温控和光控感应控制以及相应语音提示，使得自动售货系统能更人性化为用户和维护人员使用。

当以后相应的硬件条件允许的条件内可以考虑增设和扩展以上功能，使得系统功能更加完善和实用。

参考文献

[1] 汤建铨. 自动售货机经济运营体系[J]. 现代经济信息，2010，17: 31.
[2] 白丽. 自动售货机:第三次零售业革命[J]. 电子商务，2005，03: 64-65.
[3] 李丙成. 可编程逻辑器件 CPLD/FPGA 的发展概述[J]. 中国科技博览，2009，30: 171-172.
[4] 陈萌，叶桦，达飞鹏. 自动售货机主控制器及执行机构的设计与实现[J]. 东南大学学报（自然科学版），2007, 37(z1): 24-28.
[5] 王维博，吴自恒. 用 FPGA/CPLD 实现 EDA 设计[J]. 四川工业学院学报，2004，02: 6-8.
[6] 潘松. EDA 技术实用教程[M]. 第四版. 科学出版社，2010:4-5.
[7] 姜宇柏，王开军. 面向 CPLD/FPGA 的 VHDL 设计[M]. 机械工业出版社，2007: 28-30.
[8] 陈忠平. 基于 Quartus II 的 FPGA/CPLD 设计与实践[M]. 电子工业出版社，2010:1-3.
[9] 俞一鸣. Altera 可编程逻辑器件的应用与设计[M]. 机械工业出版社，2007: 8-9.
[10] Altera Corporation. MAX II Device Handbook [EB/OL]. 2008:52.
[11] AMS Co.Ltd. 1A Adjustable/Fixed Low Dropout Linear Regulator [EB/OL]. 2012.
[12] 张友木. 基于 VHDL 语言的几种消抖电路的设计[J]. 山西电子技术，2011，01: 61-63.
[13] 黄秋萍，王汉祥，李富华. 无抖动开关的研究与设计[J]. 电子与封装，2010，11: 26-28.
[14] 樊勇. 浅谈视觉暂留现象[J]. 初中生世界(八年级物理)，2011(Z6): 59-60.
[15] 许永贤. 基于 CPLD 的任意整数半整数分频器设计[J]. 电子工程师，2006(04): 27-28.
[16] 张喜凤，屈宝鹏. 基于 VHDL 的矩阵键盘及显示电路设计[J]. 现代电子技术，2010，16: 14-15.
[17] Mano M, Ciletti M D. Digital design[M]. 5 edition ed. Prentice Hall, 2012:198-202.
[18] 阎石. 数字电子技术基础[M]. 高等教育出版社，2006: 181-182.
[19] 李大社. 基于 Quartus II 的 FPGA/CPLD 设计实例精解[M]. 电子工业出版社，2010:216-219.

致谢

大学四年的学习生活马上要拉下帷幕了，在忙碌的学习生活中体会到了培养自主学习的可贵和必要。感激学院给我提供了一个吸收精神营养，培养专业技能的场所，不仅教导我专业知识本领，还指导我毕业后面对工作，面对挑战所应有的平和心态和职业素养。感激母校，不仅仅感激母校所提供优良的住宿和学习环境，还感激母校自然而然形成的浓厚学习氛围一直督促我不断进步。

在此我十分感谢我的指导老师王飞老师。王飞老师对于我本科毕业论文十分的关注，从一开始的开题报告到论文完成，王飞老师总是主动热情的和我沟通，实时了解我设计的进度和所需解决的问题。尤其是我的基础较为薄弱，老师对我另找课题的做法表示理解和支持，并且鼓励我要对自己做出的选择负责。这样的精神鼓舞使我十分触动，因此一直认真坚持并自主完成毕业论文，没有辜负老师对我的期望。

同时要感谢父母对于本科学业的大力支持，不但为我提供生活资金来源，使得我免于兼职工作的辛苦和焦虑，还在思想建设这方面起了非常大的作用。他们的关心和开导，使得我对生活和学习的挑战始终抱着积极的心态来面对，使我逐渐变得不畏惧挫折，勇于挑战困难。

最后衷心感谢同班一起奋斗的同学、教导过我的各位老师对我的鼎力帮助和支持，真诚且庄重的跟大家说声：感激不尽！

冯思远

2015 年 4 月 30 日

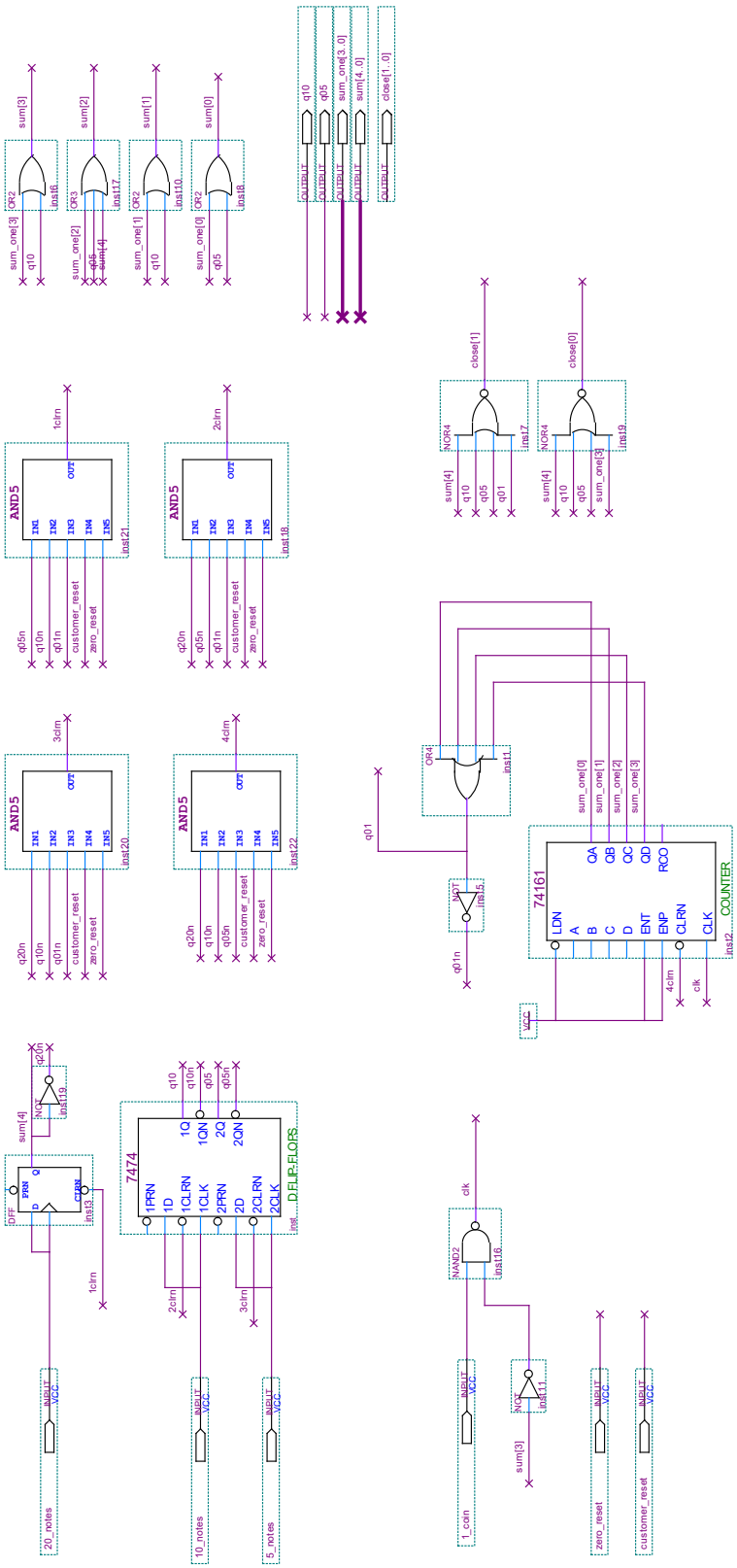


图 F1 金额控制模块原理图

Fig. F1 Schematic diagram of money-input control module

附录 1：控制模块原理图.bdf

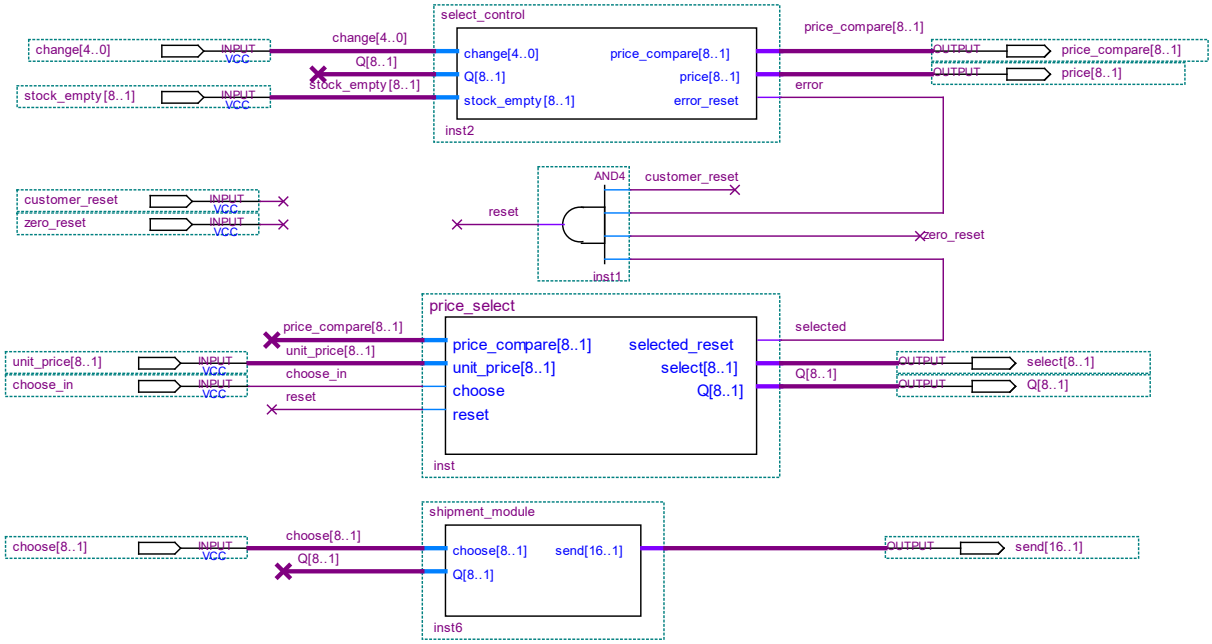


图 F2 商品选择控制模块原理图

Fig. F2 Schematic diagram of select-module

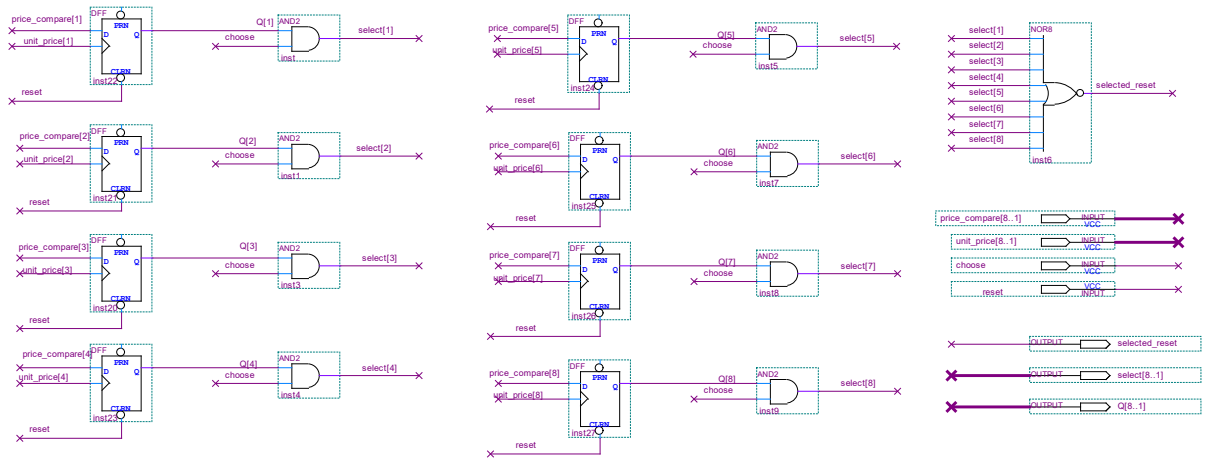


图 F3 价格选择子程序原理图

Fig. Schematic diagram of price-select subroutine

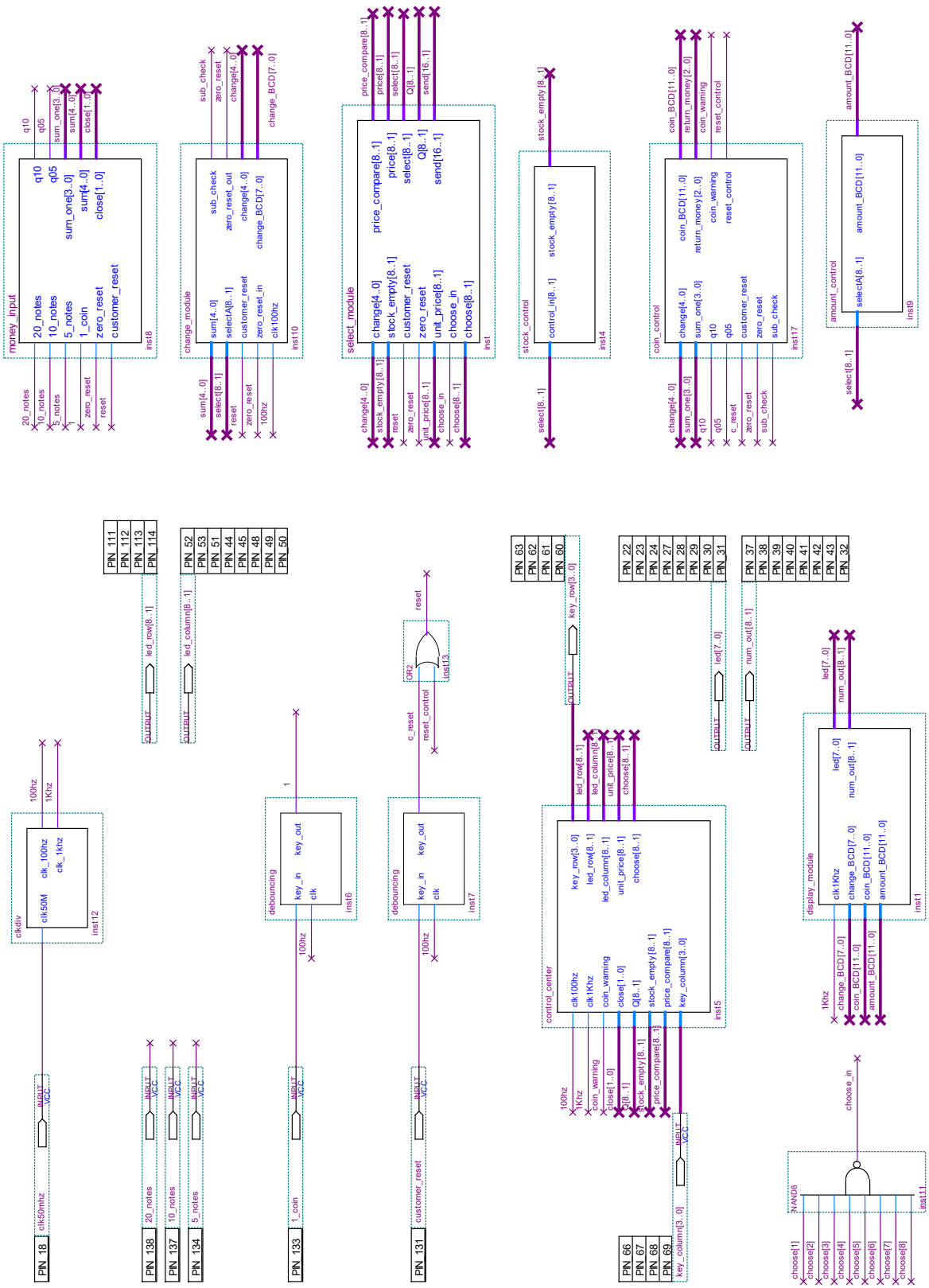


图 F4 自动售货控制模块原理图

Fig. F4 Schematic diagram of vending machine control system

```
--*********************************
--****商品选择控制模块：
--*********************************


--*********************************
--****选择控制子模块.vhd
--*********************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity select_control is
port(change:in std_logic_vector(4 downto 0);
        Q,stock_empty:in std_logic_vector(8 downto 1);
        price_compare,price:out std_logic_vector(8 downto 1);
        error_reset:out std_logic);
end entity select_control;
architecture beh of select_control is
signal clk:std_logic;
signal p: std_logic_vector(8 downto 1);
begin
price<=p;
price_compare<=stock_empty and p;
process(change)
begin
case(change) is                                  --投入金额满足相应单价时，使得
   when "00000" =>p<= "00000000";                 --相应单价可被选择
   when "00001" =>p<= "00000001";
   when "00010" =>p<= "00000011";
   when "00011" =>p<= "00000111";
   when "00100" =>p<= "00001111";
   when "00101" =>p<= "00011111";
   when "00110" =>p<= "00111111";
   when "00111" =>p<= "01111111";
   when others =>p<= null;
   end case;

if change>="01000" then
     p<="11111111";
     end if;
end process;


process(Q)                                        --当存在误选时自动清零
begin
case(Q) is
   when "00000000" =>error_reset<= '1';
   when "00000001" =>error_reset<= '1';
   when "00000010" =>error_reset<= '1';
   when "00000100" =>error_reset<= '1';
   when "00001000" =>error_reset<= '1';
   when "00010000" =>error_reset<= '1';
   when "00100000" =>error_reset<= '1';
   when "01000000" =>error_reset<= '1';
   when "10000000" =>error_reset<= '1';
   when others    =>error_reset<= '0';
   end case;
end process;
end architecture beh;
```

附录 2：控制模块源程序.vhd

```
--*********************************
--****出库控制子模块.vhd
--*********************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity shipment_module is
port(choose,Q:in std_logic_vector (8 downto 1);
        send:out std_logic_vector(16 downto 1));
end entity    shipment_module;
architecture beh of    shipment_module is
signal row:std_logic_vector(8 downto 1);
signal column:std_logic_vector(8 downto 1);

begin
process(Q,choose)
begin

case Q is
when "00000000" =>row<="00000000";
when "00000001" =>row<="10000000";
when "00000010" =>row<="01000000";
when "00000100" =>row<="00100000";
when "00001000" =>row<="00010000";
when "00010000" =>row<="00001000";
when "00100000" =>row<="00000100";
when "01000000" =>row<="00000010";
when "10000000" =>row<="00000001";
when others =>null;
end case;

case choose is
when "11111111" =>column<="00000000";
when "01111111" =>column<="10000000";
when "10111111" =>column<="01000000";
when "11011111" =>column<="00100000";
when "11101111" =>column<="00010000";
when "11110111" =>column<="00001000";
when "11111011" =>column<="00000100";
when "11111101" =>column<="00000010";
when "11111110" =>column<="00000001";
when others =>null;
end case;
end process;
send<=row&column;
end architecture beh;

--*********************************
--****找零显示模块源程.vhd
--*********************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity change_module is
port(sum:in std_logic_vector (4 downto 0);
        selectA :in std_logic_vector (8 downto 1);
        customer_reset,zero_reset_in,clk100hz:in std_logic;
        sub_check,zero_reset_out: out std_logic;
        change: out std_logic_vector(4 downto 0);
        change_BCD: out std_logic_vector(7 downto 0));
end entity change_module;
```

```
 architecture beh of change_module is
signal b,s1,s0,q:std_logic_vector(3 downto 0);
signal c,t,z,clk:std_logic;
signal sub,sub8,sub7,sub6,sub5,sub4,sub3,sub2,sub1:integer range 0 to 20;
signal remain,amount:integer range 0 to 20;
begin
 c<=customer_reset;
 z<=zero_reset_in;
 change_BCD<=s1&s0;
 amount<=conv_integer(sum);
 process(selectA(8),c,z)
 begin
 if (c and z)<='0' then                          --检测用户找零/退币按钮、自动清零子程序，当
     sub8<=0;                                    --有低电平输入时累加和清零
     elsif rising_edge(selectA(8)) then          --检测 selectA[8],累计上升沿次数，每次自动+8
     sub8<=sub8+8;
     end if;
 end process;

 process(selectA(7),c)
 begin
 if (c and z)<='0' then
     sub7<=0;                                     --检测 selectA[7],累计上升沿次数，每次自动+7
     elsif rising_edge(selectA(7)) then
     sub7<=sub7+7;
     end if;
 end process;

 process(selectA(6),c)
 begin
 if (c and z)<='0' then
     sub6<=0;
     elsif rising_edge(selectA(6)) then          --检测 selectA[6],累计上升沿次数，每次自动+6
     sub6<=sub6+6;
     end if;
 end process;

 process(selectA(5),c)
 begin
 if (c and z)<='0' then
     sub5<=0;
     elsif rising_edge(selectA(5)) then          --检测 selectA[5],累计上升沿次数，每次自动+5
     sub5<=sub5+5;
     end if;
 end process;

 process(selectA(4),c)
 begin
 if   (c and z)<='0' then
     sub4<=0;
     elsif rising_edge(selectA(4)) then          --检测 selectA[4],累计上升沿次数，每次自动+4
     sub4<=sub4+4 ;
     end if;
 end process;
 process(selectA(3),c)
 begin
 if   (c and z)<='0' then
     sub3<=0;
     elsif rising_edge(selectA(3)) then          --检测 selectA[3],累计上升沿次数，每次自动+3
     sub3<=sub3+3 ;
     end if;
 end process;
```

```
process(selectA(2),c)
  begin
  if    (c and z)<='0' then
        sub2<=0;
        elsif rising_edge(selectA(2)) then            --检测 selectA[2],累计上升沿次数，每次自动+2
        sub2<=sub2+2 ;
        end if;
  end process;

  process(selectA(1),c)
  begin
  if    (c and z)<='0' then
        sub1<=0;
        elsif rising_edge(selectA(1)) then          --检测 selectA[1],累计上升沿次数，每次自动+1
        sub1<=sub1+1 ;
        end if;
  end process;
sub<=sub8+sub7+sub6+sub5+sub4+sub3+sub2+sub1;                    --累加所有已成功消费金额

process(amount,remain)                                          --计算找应零数 m
  begin
  if amount>=sub then
        remain<=amount-sub ;
        else remain<=0;
        end if;
        end process;
process(s1,s0)
variable ten,one :integer range 0 to 9;
begin                                       --应找零数转换 3 位 BCD 码输出
ten:=remain/10;
one:=remain mod 10;
s1<=conv_std_logic_vector(ten,4);
s0<=conv_std_logic_vector(one,4);
end process;
process(sub)                                                --输出已存在消费信号
begin
if sub<=0 then
sub_check<='0';
else sub_check<='1';
end if;
end process;

process(sum,t,sub)                                          --自动清零子程序
begin
if amount/=0 then                                           --当已存在消费且应找零数为 0 时开始记录脉冲数
    if amount<=sub then                                          --当脉冲数累计 3 次时输出低电平
    if falling_edge(clk) then                               --此时该低电平使得系统复位，使得输入钱归零
      if b="0011" then                                          --此时自动清零信号马上回到高电平
           t<='0';
           else b<=b+1;
           t<='1';
           end if;
         end if;
       else t<='1';
       end if;
     else t<='1';
  end if;
end process;
clk<=clk100hz;
zero_reset_out<=t;
change<=conv_std_logic_vector(remain,5);
end architecture beh;
```

```
--*********************************
--****库存管理模块源程序.vhd
--*********************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity stock_control is
port(control_in:in std_logic_vector(8 downto 1);
        stock_empty:out std_logic_vector(8 downto 1));
end entity stock_control;
architecture beh of stock_control is
signal t1,t2,t3,t4,t5,t6,t7,t8:std_logic_vector(6 downto 0);
signal clk:std_logic_vector(8 downto 1);
begin

--累加 1-8 元单价商品上升沿数目
--当该单价商品第 80 个上升沿到达时输出高电平
process(control_in(1))
begin
  if rising_edge(control_in(1)) then
    if t1>="1001111" then
       clk(1)<='1';
       else t1<=t1+1;
       clk(1) <='0';
       end if;
     end if;
end process;

process(control_in(2))
begin
  if rising_edge(control_in(2)) then
     if t2>="1001111" then
       clk(2)<='1';
       else t2<=t2+1;
       clk(2) <='0';
       end if;
     end if;
end process;

process(control_in(3))
begin
  if rising_edge(control_in(3)) then
     if t3>="1001111" then
       clk(3)<='1';
       else t3<=t3+1;
       clk(3) <='0';
       end if;
     end if;
  end process;

process(control_in(4))
begin
if rising_edge(control_in(4)) then
     if t4>="1001111" then
       clk(4)<='1';
       else t4<=t4+1;
       clk(4) <='0';
       end if;
     end if;
end process;
```

```
process(control_in(5))
begin
  if rising_edge(control_in(5)) then
     if t5>="1001111" then
       clk(5)<='1';
       else t5<=t5+1;
       clk(5) <='0';
       end if;
     end if;
  end process;

process(control_in(6))
begin
  if rising_edge(control_in(6)) then
     if t6>="1001111" then
       clk(6)<='1';
       else t6<=t6+1;
       clk(6) <='0';
       end if;
     end if;
  end process;

process(control_in(7))
begin
  if rising_edge(control_in(7)) then
     if t7>="1001111" then
       clk(7)<='1';
       else t7<=t7+1;
       clk(7) <='0';
       end if;
     end if;
  end process;

process(control_in(8))
begin
  if rising_edge(control_in(8)) then
     if t8>="1001111" then
       clk(8)<='1';
       else t8<=t8+1;
       clk(8) <='0';
       end if;
     end if;
  end process;
  stock_empty   <= not clk;   --若库存为 0 则低电平输出，使得商品选择模块输入无效
  end architecture beh;

--*********************************
--****钱币控制模块源程.vhd
--*********************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity coin_control is
port(change:in std_logic_vector(4 downto 0);
        sum_one:in std_logic_vector (3 downto 0);
        q10,q05,customer_reset,zero_reset,sub_check:in std_logic;
        coin_BCD:out std_logic_vector(11 downto 0);
        return_money:out std_logic_vector(2 downto 0);
        coin_warning:out std_logic;
        reset_control:out std_logic);
end entity coin_control;
```

```
architecture beh of coin_control is
signal r20,r05,r10,warning,clk1,clk2:std_logic;
signal control:std_logic;
signal Q,s,s2,s1,s0:std_logic_vector(3 downto 0);
signal a:std_logic_vector (15 downto 0);
signal num:integer range 0 to 511;          --目前硬币剩余数
signal m:integer:= 200;                        --1 元找零硬币初始数量为 200
signal c:integer range 0 to 20;               --应找零数

begin
process(clk1)
begin
s<=sum_one;
clk1<=(customer_reset and zero_reset) or (not sub_check);
clk2<=customer_reset;
c<=conv_integer(change);
coin_BCD<=s2&s1&s0;

if    num<c then
            control<='1';
     else control<='0';
     end if;


if sum_one<="0000" then
reset_control<=control and    sub_check;
else reset_control<='0';
end if;

  --****找零程序****--
if sub_check<='1' then
  if falling_edge(clk1) then       --当找零电平出现下降沿跳变时，
       if s<="0000" then                 --如果投入钱币类型为纸币
            if num>=c then                --当前剩余数大于或等于应找零数时时
            a<=a+c;                          --则目前累计支出硬币数＝过去累计+此次找零数
            end if;
            else a<=a+c-s;              --如果投入钱币类型为硬币
          end if;                          --则目前累计支出硬币数＝过去累计+此次投入硬币数-应找零数
     end if;
   end if;
end process;

process(num)                                --当硬币数超过 200 时
variable b: integer range -255 to 255;
begin
b:=conv_integer(a);
if sub_check<='1' then
     if b>=0 then
     num<=m-b;
     else num<=m+(ABS b);              --当前硬币数=200+硬币投入数
     end if;
end if;
end process;

process(num)                                                --当前硬币剩余数量转换为 3 位 BCD 码输出
variable hundred,ten,one :integer range 0 to 9;
begin
hundred:=num/100;
ten:=(num mod 100)/10;
one:=num mod 10;
```

```
s2<=conv_std_logic_vector(hundred,4);          --s2,s1,s0 一次代表剩余数的百位、十位、各位
s1<=conv_std_logic_vector(ten,4);
s0<=conv_std_logic_vector(one,4);
end process;

process(r20)                                    --20 元纸币退币操作
begin
   if c/=20 then                                --当应找零数为 10 时，则判断未消费返还 10 元纸币
        r20<='0';
      elsif clk2<='0' then
           r20<='1';
        end if;
end process;

process(r10)                                    --10 元纸币退币操作
begin
if q10<='1' then                                --当应找零数量为 10，且投入钱币为纸币 10 元时
     if c<=10 then                              --则判断未消费返还 10 元纸币
       if clk2<='0' then
            r10<='1';
          if c/=10 then                         --如果应找零数不为 10，则不执行
     r10<='0';
     end if;
     end if;
     end if;
     end if;
if q10<='0' then
     r10<='0';
     end if;
end process;

process(r05)                                    --5 元纸币退币操作
begin
if q05<='1' then                                --当应找零数量为 5，且投入钱币为纸币 5 元时
     if c<=5 then                               --则判断未消费返还 5 元纸币
       if clk2<='0' then
            r05<='1';
          if c/=5 then                          --如果应找零数不为 5，则不执行
     r05<='0';
     end if;
     end if;
     end if;
     end if;
  if q05<='0' then
     r05<='0';
     end if;
end process;

process(num)
begin
if num>=0 then                                  --当 1 元硬币剩余量小于等于 10 时，输出高电平
     if num<11 then
     coin_warning<='1';
     else coin_warning<='0';                    --反之输出高电平
     end if;
end if;
end process;
return_money<=r20 & r10 & r05;
end architecture beh;
```

```
--********************************
--****售出金额显示模块源程.vhd
--********************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity amount_control is
port(selectA:in std_logic_vector (8 downto 1);
        amount_BCD:out std_logic_vector (11 downto 0));
end entity amount_control;
architecture beh of amount_control is
signal t,t8,t7,t6,t5,t4,t3,t2,t1:integer range 0 to 511;
signal q,s2,s1,s0:std_logic_vector(3 downto 0);
signal num:std_logic_vector(8 downto 1);

begin
amount_BCD<=s2&s1&s0;
process(selectA(8))
begin
if rising_edge(selectA(8)) then              --检测 selectA[8],累计上升沿次数，每次自动+8
       t8<=t8+8;
       end if;
  end process;

process(selectA(7))
begin
if rising_edge(selectA(7)) then              --检测 selectA[7],累计上升沿次数，每次自动+7
       t7<=t7+7;
       end if;
  end process;

process(selectA(6))
begin
if rising_edge(selectA(6)) then              --检测 selectA[6],累计上升沿次数，每次自动+6
       t6<=t6+6;
       end if;
  end process;

process(selectA(5))
begin
if rising_edge(selectA(5)) then              --检测 selectA[5],累计上升沿次数，每次自动+5
       t5<=t5+5;
       end if;
  end process;
process(selectA(4))
begin
if rising_edge(selectA(4)) then              --检测 selectA[4],累计上升沿次数，每次自动+4
       t4<=t4+4 ;
       end if;
  end process;

process(selectA(3))
  begin
  if rising_edge(selectA(3)) then            --检测 selectA[3],累计上升沿次数，每次自动+3
       t3<=t3+3 ;
       end if;
  end process;
```

```
process(selectA(2))
begin
if rising_edge(selectA(2)) then                --检测 selectA[2],累计上升沿次数，每次自动+2
      t2<=t2+2 ;
      end if;
  end process;


process(selectA(1))
begin
if rising_edge(selectA(1)) then                --检测 selectA[1],累计上升沿次数，每次自动+1
      t1<=t1+1 ;
      end if;
  end process;

t<=t8+t7+t6+t5+t4+t3+t2+t1;                              --累加所有已成功消费金额


process(t)                                              --售出金额总和转换 3 位 BCD 码输出
variable hundred,ten,one :integer range 0 to 9;
begin
hundred:=t/100;
ten:=(t mod 100)/10;
one:=t mod 10;

s2<=conv_std_logic_vector(hundred,4);
s1<=conv_std_logic_vector(ten,4);
s0<=conv_std_logic_vector(one,4);
end process;
end architecture beh;

--**********************************
--****控制模块源程序.vhd
--**********************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity control_center is
port (clk100hz,clk1Khz,coin_warning: in std_logic;
        close: in std_logic_vector(1 downto 0);
        Q,stock_empty,price_compare:in std_logic_vector(8 downto 1);
        key_column:in std_logic_vector(3 downto 0);
        key_row:out std_logic_vector(3 downto 0);
        led_row:out std_logic_vector(8 downto 1);
        led_column:out std_logic_vector(8 downto 1);
        unit_price,choose:out std_logic_vector(8 downto 1));
end entity control_center;
architecture beh of control_center is
signal clk1:std_logic_vector(1 downto 0);
signal clk2,control:std_logic_vector(2 downto 0);
signal scan1:std_logic_vector(3 downto 0);
signal stock,scan2,price:std_logic_vector(8 downto 1);

begin
key_row<=scan1;
led_row<=scan2;
stock<=( stock_empty(1)&stock_empty(2)&stock_empty(3)&stock_empty(4)
          &stock_empty(5)&stock_empty(6)&stock_empty(7)&stock_empty(8));
```

```
price<=( price_compare(1)&price_compare(2)&price_compare(3)&price_compare(4)
          &price_compare(5)&price_compare(6)&price_compare(7)&price_compare(8));
control<=coin_warning & close;
process(clk100hz,clk1)
begin
if rising_edge(clk100hz) then
    clk1<= clk1 + 1;
    end if;
end process;

process(clk1Khz,clk2)
begin
if rising_edge(clk1Khz) then
    clk2<= clk2 + 1;
    end if;
end process;

process(clk1)
begin
case clk1 is                                    --键盘逐行扫描检测
  when "00"=> scan1<="1110";
  when "01"=> scan1<="1101";
  when "10"=> scan1<="1011";
  when "11"=> scan1<="0111";
  end case;

  case clk2 is
  when "000"=> scan2<="10000000";       --点阵逐行扫描控制
  when "001"=> scan2<="10000000";
  when "010"=> scan2<="00100000";
  when "011"=> scan2<="00100000";
  when "100"=> scan2<="00001000";
  when "101"=> scan2<="00000100";
  when "110"=> scan2<="00000010";
  when "111"=> scan2<="00000001";
  end case;
  end process;

process(clk100hz)                               --键盘译码程序
begin
if rising_edge(clk100hz) then
    case scan1 is    -
          when "1110" =>
            case key_column is                  --键盘 1 至 2 行代表 8 种不同商品单价按键
                        when "1110" =>unit_price<= "11111110";
                        when "1101" =>unit_price<= "11111101";
                        when "1011" =>unit_price<= "11111011";
                        when "0111" =>unit_price<= "11110111";
                        when "1111" =>unit_price<= "11111111";
                        when others =>null;
                        end case;
          when "1101" =>
            case key_column is
                        when "1110" =>unit_price<= "11101111";
                        when "1101" =>unit_price<= "11011111";
                        when "1011" =>unit_price<= "10111111";
                        when "0111" =>unit_price<= "01111111";
                        when "1111" =>unit_price<= "11111111";
                        when others =>null;
                        end case;
```

```
        when "1011" =>                  --键盘 3 至 4 行代表 8 种不同商品类型按键
             case key_column is
                  when "1110" =>choose<= "11111110";
                  when "1101" =>choose<= "11111101";
                  when "1011" =>choose<= "11111011";
                  when "0111" =>choose<= "11110111";
                  when "1111" =>choose<= "11111111";
                  when others =>null;
                  end case;
        when "0111" =>
             case key_column is
                  when "1110" =>choose<= "11101111";
                  when "1101" =>choose<= "11011111";
                  when "1011" =>choose<= "10111111";
                  when "0111" =>choose<= "01111111";
                  when "1111" =>choose<= "11111111";
                  when others =>null;
                  end case;
        when others =>
                  price<= null;
                  choose<=null;
        end case;
        end if;

case scan2 is                           --点阵译码控制
when "10000000"=>
             led_column<=not price; --第一行代表可选择单价指示
when "01000000"=>
             led_column<="11111111";
when "00100000"=>                       --第三行代表已选择单价指示
      case Q is
             when "00000000"=>led_column<="11111111";
             when "00000001"=>led_column<="01111111";
             when "00000010"=>led_column<="10111111";
             when "00000100"=>led_column<="11011111";
             when "00001000"=>led_column<="11101111";
             when "00010000"=>led_column<="11110111";
             when "00100000"=>led_column<="11111011";
             when "01000000"=>led_column<="11111101";
             when "10000000"=>led_column<="11111110";
             when others       =>led_column<=null;
             end case;
when "00010000"=>
             led_column<="11111111";
                when "00001000"=>          --第五行代表库存状况指示
             led_column<= stock;
when "00000100"=>
             led_column<="11111111";
                when "00000010"=>          --第七行代表硬币不足和投币口指示
      case control is
             when "000"     =>led_column<="11111111";
             when "001"     =>led_column<="11111110";
             when "010"     =>led_column<="11111101";
             when "011"     =>led_column<="11111100";
             when "100"     =>led_column<="01111111";
             when "101"     =>led_column<="01111110";
             when "110"     =>led_column<="01111101";
             when "111"     =>led_column<="01111100";
             when others =>led_column<=null;
             end case;
```

```
when "00000001"=>
                led_column<="11111111";
when others =>null;
end case;
end process;
end architecture beh;

--********************************
--****数据显示模块.vhd
--********************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity display_module is
port(clk1Khz:in std_logic;
      change_BCD:in std_logic_vector(7 downto 0);
      coin_BCD,amount_BCD:in std_logic_vector(11 downto 0);
      led :out std_logic_vector (7 downto 0);
      num_out:out std_logic_vector (8 downto 1));
end entity display_module;
architecture beh of display_module is
signal change_s1,change_s0,q,clk:std_logic_vector(3 downto 0);
signal amount_s2,amount_s1,amount_s0:std_logic_vector(3 downto 0);
signal coin_s2,coin_s1,coin_s0:std_logic_vector(3 downto 0);
signal num: std_logic_vector (8 downto 1);
begin
num_out<=num;
change_s1<=change_BCD(7 downto 4);
change_s0<=change_BCD(3 downto 0);
amount_s2<=amount_BCD(11 downto 8);
amount_s1<=amount_BCD(7 downto 4);
amount_s0<=amount_BCD(3 downto 0);
coin_s2<=coin_BCD(11 downto 8);
coin_s1<=coin_BCD(7 downto 4);
coin_s0<=coin_BCD(3 downto 0);

process(clk1Khz,clk)
begin
  if rising_edge(clk1Khz) then
       clk<=clk+1;
       end if;
end process;

process(clk)
begin
case(clk) is                                          --控制数码管位码
when "0000" =>num<="01111111";
when "0001" =>num<="01111111";
when "0010" =>num<="10111111";
when "0011" =>num<="10111111";
when "0100" =>num<="11011111";
when "0101" =>num<="11011111";
when "0110" =>num<="11101111";
when "0111" =>num<="11101111";
when "1000" =>num<="11110111";
when "1001" =>num<="11110111";
when "1010" =>num<="11111011";
when "1011" =>num<="11111011";
when "1100" =>num<="11111101";
```

```
when "1101" =>num<="11111101";
when "1110" =>num<="11111110";
when "1111" =>num<="11111110";
when others=>null;
end case;
end process;

process(clk)
begin

case(num) is                                    --数码管从左到右依次显示
when    "01111111" =>q<=change_s1;              --应找零数（2 位）
when    "10111111" =>q<=change_s0;
when    "11011111" =>q<=coin_s2;                --找零硬币剩余数（3 位）
when    "11101111" =>q<=coin_s1;
when    "11110111" =>q<=coin_s0;
when    "11111011" =>q<=amount_s2;              --售出金额总数（3 位）
when    "11111101" =>q<=amount_s1;
when    "11111110" =>q<=amount_s0;
when others=>null;
end case;
end process;


process(q)                                      --控制数码管段码
  BEGIN
  case(q) is
    when "0000" =>led<= "11111100";
    when "0001" =>led<= "01100000";
    when "0010" =>led<= "11011010";
    when "0011" =>led<= "11110010";
    when "0100" =>led<= "01100110";
    when "0101" =>led<= "10110110";
    when "0110" =>led<= "10111110";
    when "0111" =>led<= "11100000";
    when "1000" =>led<= "11111110";
    when "1001" =>led<= "11110110";
    when "1010" =>led<= "11101110";
    when others =>null;
    end case;
  end process;
  end architecture beh;
```