

数字电子技术综合实验报告

**题 目 厨房助手——定时器**

**电气工程 学院 电气工程及其自动化 系**

**学生姓名 班级 电气22 学号 222**

**指导教师 孙敏**

2024年11月

**实验题目：厨房助手——定时器**

**学生姓名：**

**指导教师：孙敏**

摘 要

本系统实现了一个定时器，使用Verilog HDL语言，在Vivado软件中进行开发，并通过EGO1可编程逻辑器件完成硬件实现。系统采用模块化架构，分为多个功能模块，包括秒脉冲生成、按键消抖、正/倒计时、初始时间设定、数码管显示和LED灯控制等。该系统的主要功能包括正计时和倒计时两种模式，用户可以通过按键在两者之间切换，并进行开始、暂停、清零等操作，并通过数码管显示当前时间，由LED灯提示当前的工作模式或计时结束。

**关 键 词：FPGA；Verilog HDL；有限状态机；定时器**

目 录

[1 前 言 1](#_Toc183989416)

[2 系统设计 2](#_Toc183989417)

[2.1 系统总体结构 2](#_Toc183989418)

[2.1.1系统总体结构介绍 2](#_Toc183989419)

[2.1.2系统RTL顶层逻辑图 2](#_Toc183989420)

[2.1.3系统功能设计 3](#_Toc183989421)

[2.1.4系统输入、输出设计 3](#_Toc183989422)

[2.1.5系统子模块划分 4](#_Toc183989423)

[2.2 系统模块具体介绍 5](#_Toc183989424)

[2.2.1系统有限状态机模块设计（FSM.v） 5](#_Toc183989425)

[2.2.2 消抖模块（debounce.v） 7](#_Toc183989426)

[2.2.3 秒脉冲发生器模块（sec\_pulse.v） 8](#_Toc183989427)

[2.2.4 倒计时初值设定模块模块（Countdown\_time\_setup.v） 9](#_Toc183989428)

[2.2.5 可逆计时器（reversible\_cnt.v） 10](#_Toc183989429)

[2.2.6 数码管显示器（display.v） 12](#_Toc183989430)

[2.2.7 控制LED提示灯模块（Countdown.v） 13](#_Toc183989431)

[2.2.8 顶层模块（kitchen\_timer\_top.v） 14](#_Toc183989432)

[3 系统调试 16](#_Toc183989433)

[3.1 使用时难以区分计时状态 16](#_Toc183989434)

[3.2 无法正常倒计时 16](#_Toc183989435)

[4 使用说明 18](#_Toc183989436)

[5 总结 19](#_Toc183989437)

[5.1 所作工作 19](#_Toc183989438)

[5.2 课程收获 19](#_Toc183989439)

[5.3 心得体会 20](#_Toc183989440)

[5.4 课程建议 20](#_Toc183989441)

[附 录 21](#_Toc183989442)

在每一章的末尾插入下一章的MathType的章标记（打印前将其字体颜色变为白色，在打印预览中看不见即可）：

# 前 言

FPGA作为一种广泛应用的高密度可编程逻辑器件，具备丰富的布线资源、可重复编程和高集成度等优势，特别适合初学者进行模块化设计与实现。在FPGA上设计电路和系统需要使用硬件描述语言（HDL）。作为一种与C语言类似的硬件描述语言，Verilog HDL因其易学易用的特点，在本课程中被选为主要工具。

本实验基于Vivado和EGO1平台，设计了一个厨房定时器系统，采用模块化设计，具备正/倒计时、初值设定和状态提醒等功能。该系统设计简单、易上手，充分展示了实验者对已学知识的掌握以及分析与解决问题的能力。

# 系统设计

## 系统总体结构

### 2.1.1系统总体结构介绍

本系统采用模块化设计，具有更清晰的程序结构，且便于修改、调试和增加功能。各功能模块的名称如下所示：

1.顶层模块：kitchen\_timer\_top.v

2.秒脉冲发生器：sec\_pulse.v

3.可逆计时器：reversible\_cnt.v

4.倒计时初值设定模块：Countdown\_time\_setup.v

5.控制LED提示灯模块：Countdown\_LED.v

6.数码管显示器：display.v

7.消抖模块：debounce.v

系统思维导图如图2-1所示。



图 2‑1 系统总体思维导图

### 2.1.2系统RTL顶层逻辑图

系统RTL顶层逻辑图如图2-2所示。

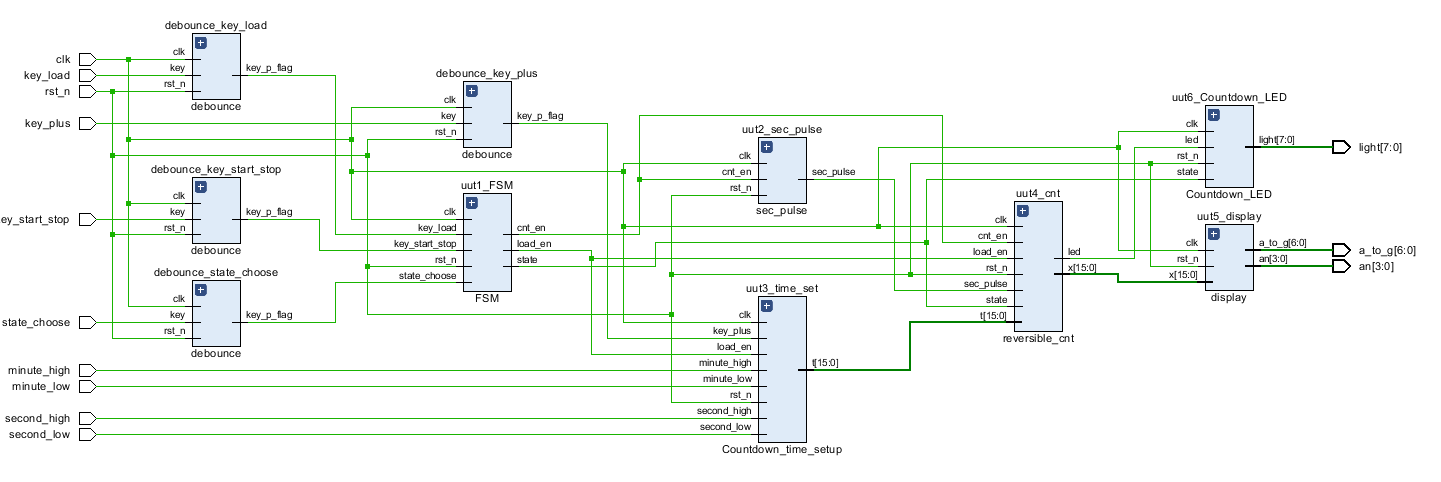


图 2‑2 系统RTL逻辑图

### 2.1.3系统功能设计

“厨房助手——定时器”的作用主要是帮助用户在厨房中进行时间管理，确保烹饪过程中的各个步骤按时完成。具体来说，定时器的作用包括基础功能和附加功能两个部分。分别为以下几点：

**基础功能：**

1. 正计时：初始状态为“0000”，按键控制开始、暂停、清零（非系统复位）。
2. 数码管显示时间：最大为99分59秒，当到达最大时间时，提醒灯LD1的所有灯亮，且显示停在99分59秒。

**附加功能：**

1. 切换计时方式且显示：按键控制倒计时与正计时切换。正计时LD1的0-3灯亮，倒计时LD1的4-7灯亮。
2. 设置定时：按键控制倒计时时间，最大为99分59秒。
3. 倒计时：按键控制开始、暂停、清零（非系统复位），并且可以在倒计时中随时按键复位到设置的时间。
4. 倒计时结束提示：提醒灯LD1的所有灯亮，并停在0。

具体地，各个模块对应的功能实现如表2-1所示。

表2‑1 模块与功能

|  |  |
| --- | --- |
| 正计时/倒计时 | reversible\_cnt.v |
| 数码管显示时间 | display.v |
| 设置倒计时定时 | Countdown\_time\_setup.v |
| LED灯显示 | Countdown\_LED.v |

### 2.1.4系统输入、输出设计

输入端口分别为：

* clk：系统时钟信号（接EGO1自带的100MHz时钟信号源）
* rst\_n：异步复位信号
* key\_start\_stop：开始/暂停按键信号
* key\_load：清零按键信号
* key\_plus：倒计时初始时间设定加一信号
* state\_choose：正计时/倒计时切换信号
* minute\_high：倒计时初始时间设定-选中分钟高位
* minute\_low：倒计时初始时间设定-选中分钟低位
* second\_high：倒计时初始时间设定-选中秒钟高位
* second\_low：倒计时初始时间设定-选中秒钟低位

其中，key\_start\_stop，key\_load，key\_plus和state\_choose是由FPGA的按键控制，因此均需要先经过消抖模块处理，如图2-3的RTL图所示：

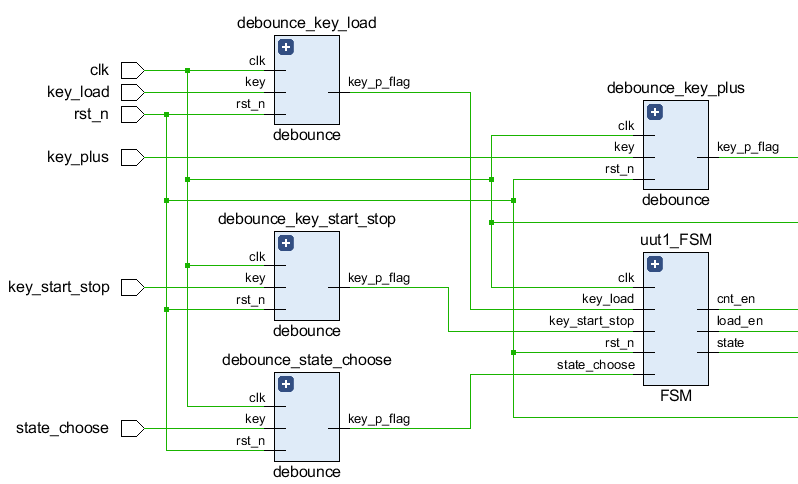


图 2‑3 系统RTL逻辑图——输入部分

系统输出分别为：

* a\_to\_g [6:0]：七位数码管
* an [3:0]：位选信号
* light [7:0]：LED灯

由控制LED提示灯模块和数码管显示器模块输出，如图2-4的RTL图所示。

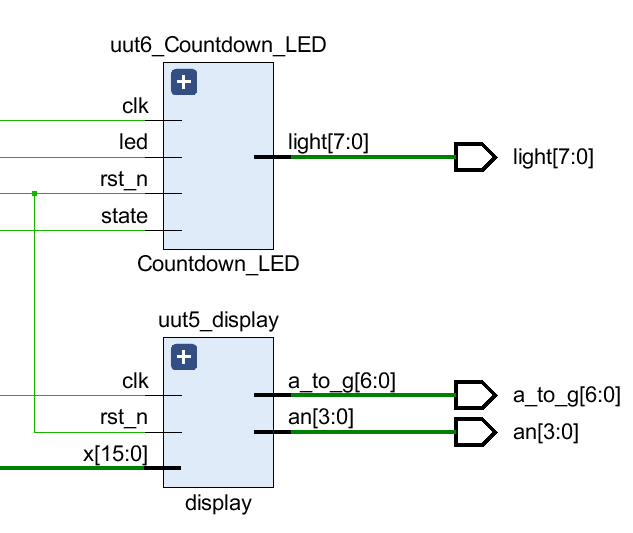


图 2‑4 系统RTL逻辑图——输出部分

### 2.1.5系统子模块划分

本系统可以划分为以下七个子模块：

* 1. **按键防抖模块**：实现对输入按键信号的防抖处理。
  2. **秒脉冲模块**：生成秒级脉冲信号，用于计时。
  3. **状态机模块**：负责各个系统状态的转移与管理。
  4. **正/倒计时计数器模块**：实现正计时和倒计时功能。
  5. **倒计时初值设定模块**：用于设定倒计时的初始值。
  6. **数码管输出模块**：负责将计时信息显示在数码管上。
  7. **LED输出模块**：控制LED灯的显示，提供状态指示或提醒功能。

七个子模块在顶层模块中灵活调用，以实现系统的各项功能。子模块在系统中的布局如图2-5所示。

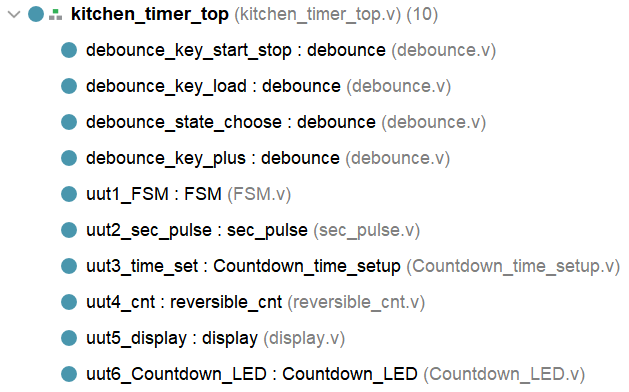


图 2‑5 系统模块

## 系统模块具体介绍

### 2.2.1系统有限状态机模块设计（FSM.v）

厨房定时器系统中涉及到多种操作模式和状态转化，例如：计时开始、暂停、清零、倒计时开始、倒计时结束等。为了高效管理这些不同的操作状态及其相应的转化，本系统采用有限状态机（FSM）来实现。

根据系统功能的需求，厨房定时器的状态被划分为六个主要状态，分别表示正计时模式和倒计时模式下的不同操作状态。如表2-2所示。

表2‑2 FSM状态说明

|  |  |  |  |
| --- | --- | --- | --- |
| **正计时** | | **倒计时** | |
| **WORK\_INIT0** | 初始复位状态 | **WORK\_INIT1** | 初始复位状态 |
| **COUNT0** | 计时状态 | **COUNT1** | 计时状态 |
| **PAUSE0** | 暂停状态 | **PAUSE1** | 暂停状态 |

为了实现状态转换，本系统使用两个reg [2:0]类型的变量来分别存储现态和次态：

* cstate：存现态
* nstate：存次态

状态转换基于三个输入信号，这些信号控制系统的操作模式和状态切换：

* state\_choose：模式选择信号，可以切换正计时和倒计时
* key\_start\_stop：暂停、恢复计时信号
* key\_load：复位信号（非系统复位），返回初态

系统输出信号由三个主要信号组成，这些信号与各个操作模式和状态直接相关：

* state：模式输出信号，正计时为0，倒计时为1
* cnt\_en：计数使能信号
* load\_en：判断是否为初态的信号

基于上述状态（现态、次态）、条件（输入）和动作（输出）的分析，本系统的有限状态机转换图如图2-6所示。

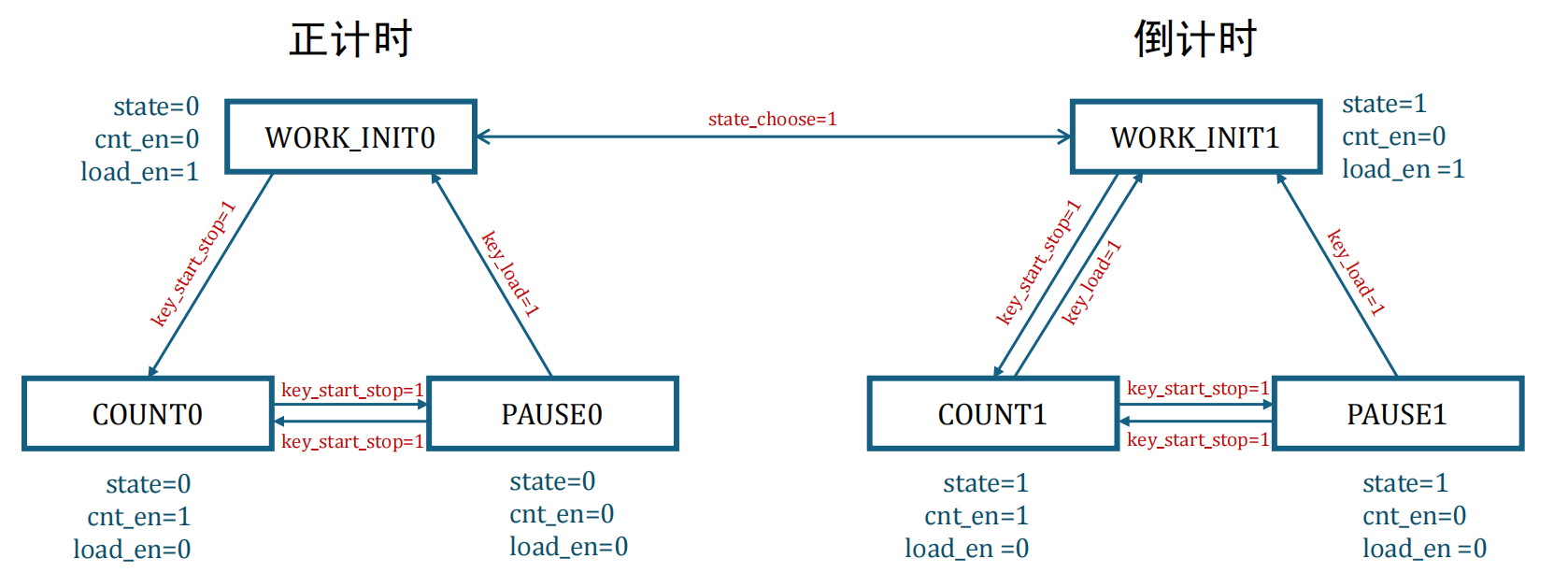


图 2‑6 系统有限状态机转换图

有限状态机的工作原理可以总结为如下几点：

1. 在初始状态下，系统默认处于正计时模式（state = 0）。当用户选择切换到倒计时模式时，可以按state\_choose 对应的按键，触发模式切换，系统进入倒计时模式（state = 1）。
2. 在计时状态下，系统根据 key\_start\_stop 信号进行暂停和恢复操作。当用户按下暂停按钮时，系统进入暂停状态（PAUSE0 或 PAUSE1），计时停止。按下恢复按钮时，系统从暂停状态恢复到计时状态。
3. key\_load 信号用于将系统复位，返回到初始状态。其中，正计时的初始状态是00:00，而倒计时的初始状态是由用户定时设计的。特别的，倒计时过程中，可以直接按key\_load 复位；正计时只能在暂停计时的状态下按键复位。

由于篇幅原因，状态机的代码将在附录中列出。代码实现的过程图如图2-7。

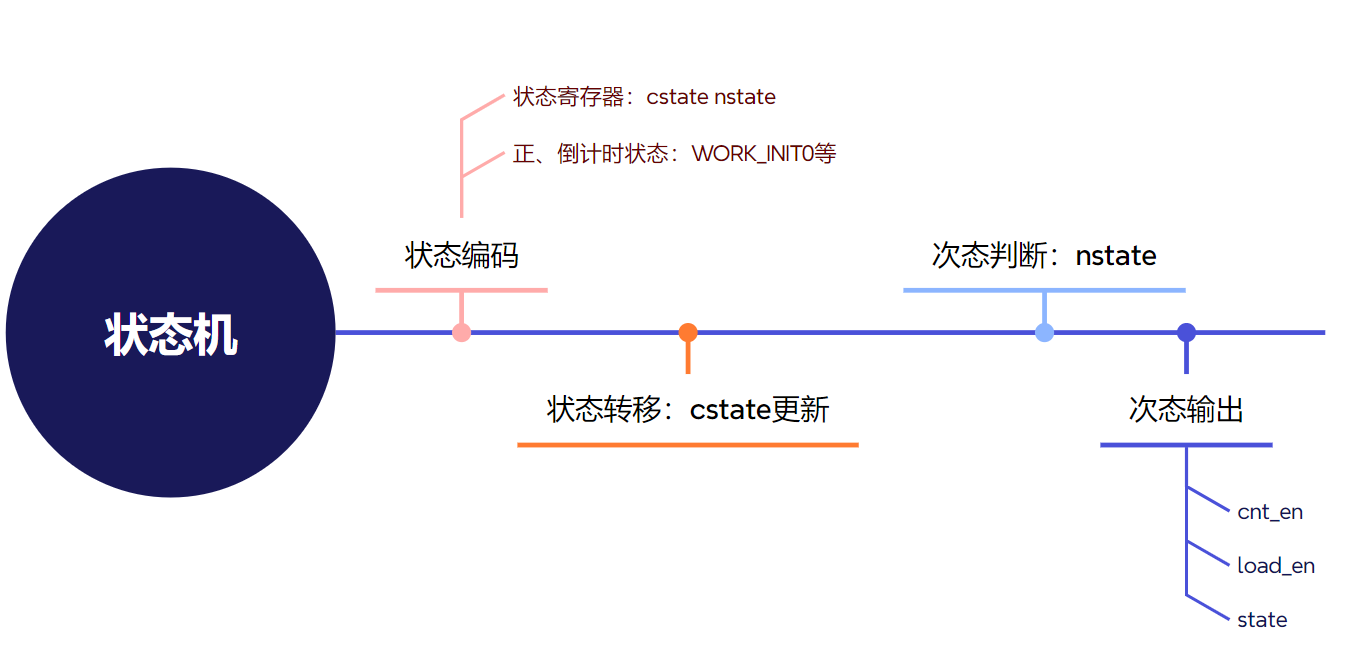


图 2‑7 FSM代码实现流程图

### 2.2.2 消抖模块（debounce.v）

在本系统中，我们采用了**延时计数器**的方式来进行软件消抖，具体的工作原理是：每当按键状态发生变化时，启动一个延时计数器，只有当该计数器计数完毕且按键状态未改变时，才认为按键状态发生了稳定变化。

输入、输出变量表如表2-3所示。

表2‑3 消抖模块输入、输出变量

|  |  |
| --- | --- |
| 变量 | 功能 |
| input clk | 系统时钟信号 |
| input rst\_n | 异步复位信号 |
| input key | 按键输入 |
| output reg key\_p\_flag | 按键上升沿标志信号 |

其中，设置了以 1500000 个时钟周期（15ms）的变量DELAY\_TIME。

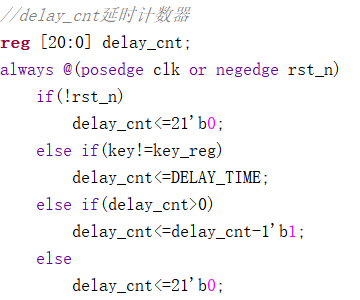
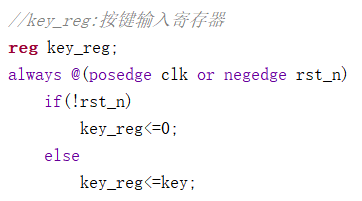
本设计的消抖模块包含了以下几个关键部分，其变量如表2-4所示。

表2‑4 消抖模块中间变量

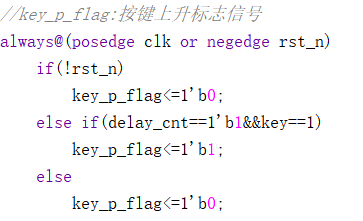
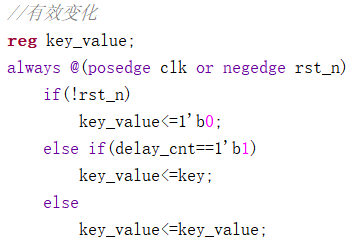
|  |  |
| --- | --- |
| 按键输入寄存器（key\_reg） | 用于捕捉按键信号的变化，并将其保存在寄存器中。每个时钟周期，都会将当前按键信号的状态保存在该寄存器中。 |
| 延时计数器（delay\_cnt） | 用于产生一个延时，确保按键信号经过一段时间的稳定后才认为是有效的。计数器会在每次按键信号变化时重置，并开始计时。只有当计数器完成设定的延时时间后，才认为按键信号稳定下来。 |
| 按键状态寄存器（key\_value） | 在延时计数器完成计时后，读取按键的当前状态，判断按键是否发生了有效变化。 |
| 按键上升沿标志信号（key\_p\_flag） | 用于标识按键的有效变化，主要在检测到按键从未按下到按下（上升沿）时，产生一个脉冲信号作为触发标志。 |

工作流程具体如下：

1. 初始状态：在模块复位后，所有寄存器清零，按键状态初始化。
2. 按键检测：当按键信号变化时，寄存器 key\_reg 捕捉当前按键状态，并触发延时计数器delay\_cnt开始倒计时，直到计数器倒计时完成。此时，如果按键信号仍然没有变化，才认为按键信号稳定，继续执行后续逻辑。



1. 标志生成：一旦按键稳定，key\_value 被更新，并根据按键的状态生成上升沿标志信号 key\_p\_flag，表示按键有效输入。



### 2.2.3 秒脉冲发生器模块（sec\_pulse.v）

本模块主要功能是基于输入的时钟信号（clk），产生一个周期为1秒的脉冲信号（sec\_pulse）。基本思路是，通过计数器对输入时钟进行分频，生成一个每秒钟脉冲一次的输出信号，常用于计时或触发操作。且受到异步复位信号（rst\_n）和暂停/恢复使能控制（cnt\_en）控制。

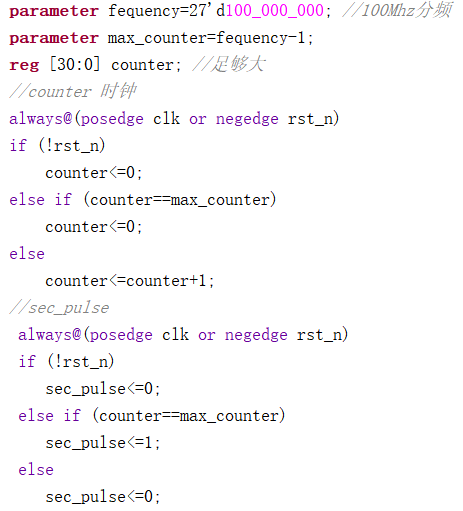
输入、输出变量表如表2-5所示。

表2‑5 秒脉冲发生器模块输入、输出变量

|  |  |
| --- | --- |
| 变量 | 功能 |
| input clk | 系统时钟信号 |
| input rst\_n | 异步复位信号 |
| input cnt\_en | 暂停/恢复使能信号 |
| output reg sec\_pulse | 秒脉冲信号 |

工作原理如下：

1. 时钟分频：模块通过一个计数器counter对输入的时钟信号进行计数，直到达到设定的分频值（如100MHz时为1秒）。
2. 脉冲输出：每当计数器达到最大值时，输出一个高电平脉冲信号sec\_pulse。
3. 使能控制：cnt\_en信号控制计数器是否工作。只有在 cnt\_en=1时，计数器才会开始计时，即有限状态机所输出的“计时模式”。



工作流程可以概括为：计数器每次增加1，当其值达到max\_counter时，sec\_pulse 输出一个高电平脉冲。

### 倒计时初值设定模块（Countdown\_time\_setup.v）

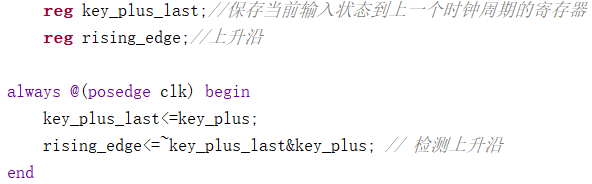
Countdown\_time\_setup 模块用于设定倒计时初值。它通过输入的信号 (second\_low、second\_high、minute\_low、minute\_high) 来选中增加哪一位的初值，由FPGA的开关控制。每次按下key\_plus时，被选中的那一位的倒计时初值会增加，直到达到最大值后，按下key\_plus会回到0。本模块包括了按键处理、计数控制和复位操作，其中按键处理不是必须的，与前面消抖模块功能重复。

输入、输出变量表如表2-6所示。

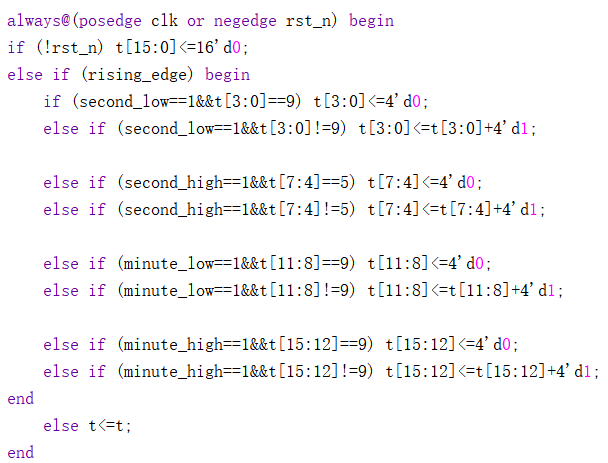
表2‑6 倒计时初值设定模块输入、输出变量

|  |  |
| --- | --- |
| 变量 | 功能 |
| input clk | 系统时钟信号 |
| input rst\_n | 异步复位信号 |
| input load\_en | 复位使能信号 |
| input second\_low, | 选中秒钟低位信号 |
| input second\_high | 选中秒钟高位信号 |
| input minute\_low | 选中分钟低位信号 |
| input minute\_high | 选中分钟高位信号 |
| input key\_plus | 增加1按键信号 |
| output reg [15:0] t | 初始时间存储器 |

首先，代码设置了边沿触发消抖功能。事实上，在系统模块化的时候，key\_plus按键已经使用了消抖模块，因此本步骤是多余的。



然后，由最低位开始逐一设置加一功能。在使用定时器的时候，如果同时将两位选中位数开关打开，只有低位会有效增加。复位时，初始时间存储器会归零。



生成的初始时间存储器会输出送至计数器，完成后续的倒计时功能。

总的来说，可以将工作原理概括为：

* 按键输入检测：通过key\_plus输入信号检测按键的上升沿，控制倒计时初值的增加。
* 秒和分钟设置：分别通过second\_low、second\_high、minute\_low和minute\_high 来设置秒和分钟的每一位（低位和高位）。秒和分钟的个位与十位通过输入信号分别控制。
* 每次按下对应按键时，相关的秒或分钟位会增加1，超过最大值（9或5）后会回绕到0。

### 可逆计时器（reversible\_cnt.v）

reversible\_cnt模块实现了一个可正计时和倒计时的计数器。根据输入的状态信号(state)，它可以在正计时和倒计时模式之间切换。在正计时模式下，计数器从0开始递增，最大可以达到99:59，并且停住；在倒计时模式下，计数器从设定的初值开始递减，直到00:00停下。模块还包括一个提醒LED灯信号(led)，不同状态的LED灯状态也不一样，后续在控制LED模块会专门介绍。

输入、输出变量表如表2-7所示。

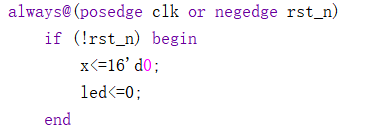
表2‑7 可逆计时器模块输入、输出变量

|  |  |
| --- | --- |
| 变量 | 功能 |
| input clk | 系统时钟信号 |
| input rst\_n | 异步复位信号 |
| input sec\_pulse | 秒脉冲信号 |
| input state | 状态信号 |
| input cnt\_en | 暂停/恢复使能信号 |
| input load\_en | 复位使能信号 |
| input [15:0] t | 倒计时初始时间存储器 |
| output reg [15:0] x | 输出实时时间 |
| output reg led | 提醒LED灯信号 |

工作流程可以概括为以下三部分：

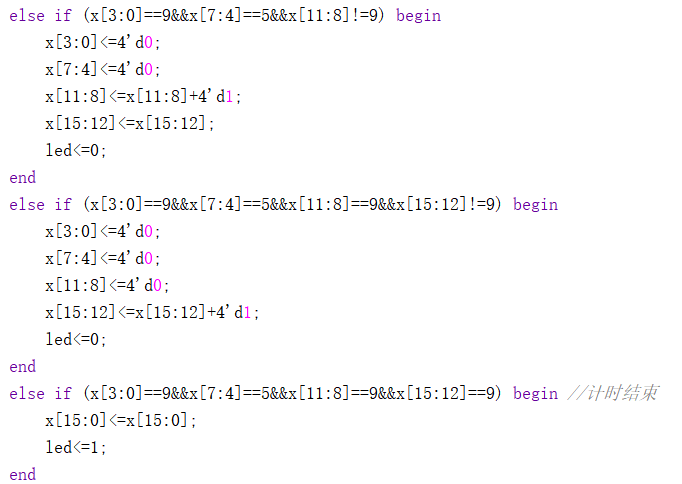
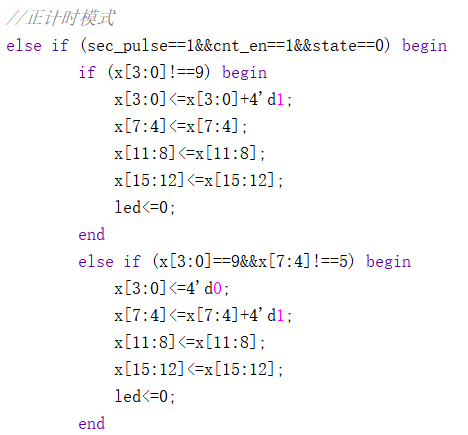
* 复位：

在复位信号rst\_n=0时，计数器实时输出时间x被重置为 0，led信号归零。



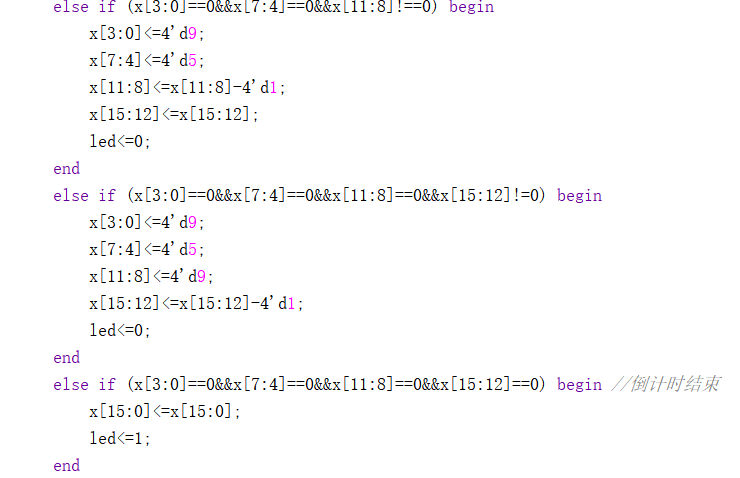
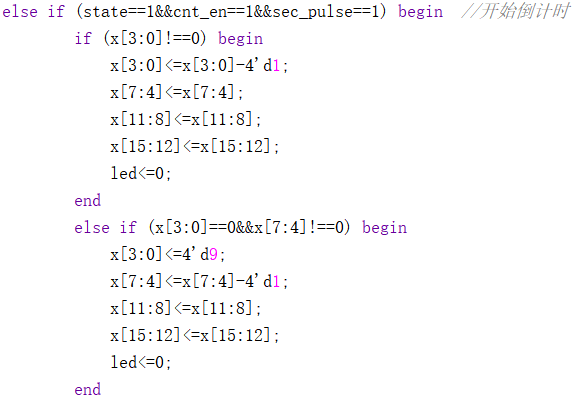
* 正计时 (state = 0)：

在正计时模式下，每秒钟sec\_pulse脉冲触发计数器递增，秒钟的个位、十位，直到分钟的个位、十位依次递增。当计数器实时输出时间x达到最大值 99:59 时，计时结束，led信号变为高电平，并且在没有按键信号输入的前提下，保持此状态不变。



* 倒计时 (state = 1)：

在倒计时模式下，通过Countdown\_time\_setup 模块的输出t设置初值（load\_en=1时，将初值送至计数器），计数器从该初值开始递减。每秒sec\_pulse脉冲触发计数器递减，直到计数器值为0时，倒计时结束，led信号变为高电平，并且在没有按键信号输入的前提下，保持此状态不变。



### 数码管显示器（display.v）

display模块的功能是将输入的16位数值x显示在四个七段显示器上，采用时钟分频和位扫描技术，通过位选和七段段选信号控制四个七段显示器的显示内容。

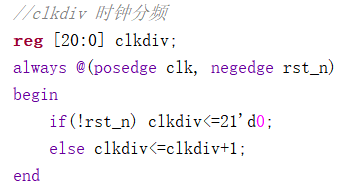
输入、输出变量表如表2-8所示。

表2‑8 数码管显示器模块输入、输出变量

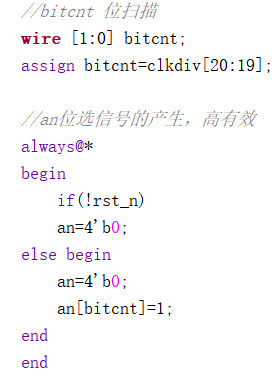
|  |  |
| --- | --- |
| 变量 | 功能 |
| input clk | 系统时钟信号 |
| input rst\_n | 异步复位信号 |
| input [15:0] x | 实时时间信号 |
| output reg [6:0] a\_to\_g | 数码管段信号 |
| output reg [3:0] an | 数码管位选信号 |

其工作原理及代码如下：

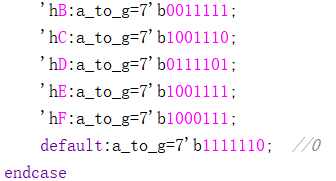
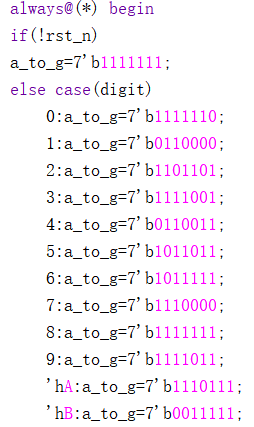
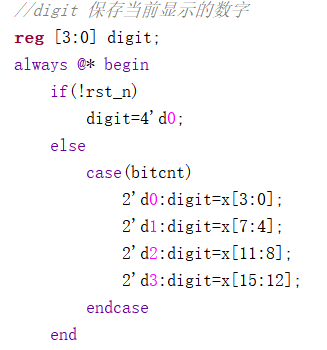
* 时钟分频：由于七段显示器的刷新频率较低，因此需要通过时钟分频将输入时钟降低到适合显示器刷新频率的信号。本模块使用一个21位的计数器 clkdiv 来进行时钟分频。



* 位扫描与位选：通过bitcnt信号，模块按顺序轮流选择四个七段显示器中的一个进行显示。an信号控制哪个显示器被选中，an的每一位对应一个显示器（高有效）。



* 数字选择与译码：根据bitcnt信号，选择对应的数字位（从x的低4位到高4位）。选择到的数字（0-9，A-F）会通过一个译码器（a\_to\_g）转化为七段显示器的段信号。每个数字的段信号会控制显示器上相应的LED开关，以显示对应的数字或字符。由于篇幅原因，这里的代码没有截全，详见附录。



### 控制LED提示灯模块（Countdown.v）

Countdown\_LED模块的主要功能是根据计时器的状态和倒计时结束标志，控制 8 个 LED 灯的显示。根据当前的计时状态（state判断正计时、倒计时）以及是否有结束计时的提醒LED灯信号（led）来点亮或熄灭不同的 LED 灯。

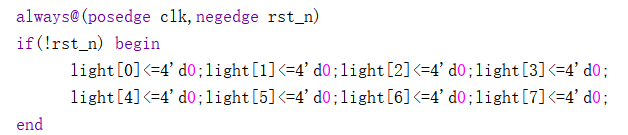
输入、输出变量表如表2-9所示。

表2‑9 控制LED提示灯模块输入、输出变量

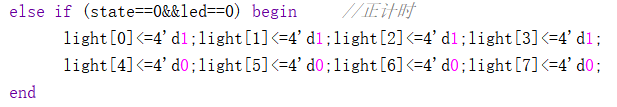
|  |  |
| --- | --- |
| 变量 | 功能 |
| input clk | 时钟信号 |
| input rst\_n | 复位信号 |
| input led | 提醒LED灯信号 |
| input state | 状态信号 |
| output reg [7:0] light | 输出LED灯显示 |

LED 控制逻辑如下：

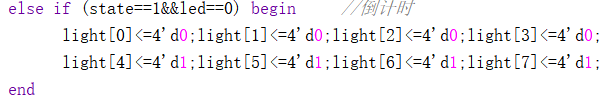
* 系统复位时，所有LED灯熄灭



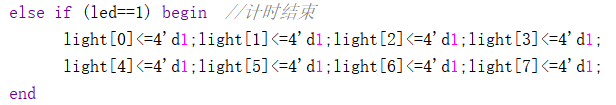
* 正计时模式（state=0且led=0），点亮前4个LED灯（light[0]到light[3]）



* 倒计时模式（state=1且led=0），点亮后4个LED灯（light[4]到light[7]）



* 计时结束（led=1，由可逆计数器结构判断和输出），点亮所有（8个）LED灯（light[0]到light[7]）



### 顶层模块（kitchen\_timer\_top.v）

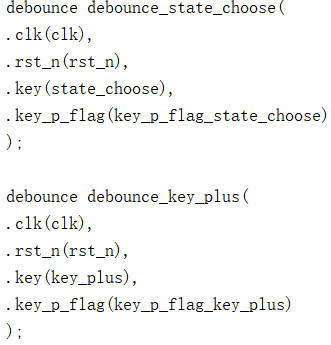
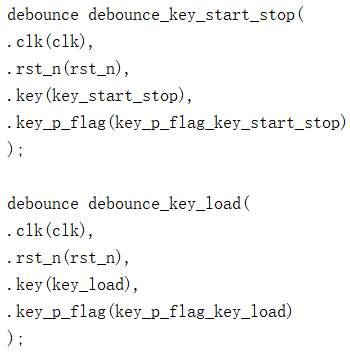
kitchen\_timer\_top是厨房定时器的顶层模块，负责将各个子模块进行连接和协同工作。该模块的输入包括按键输入、时钟信号和复位信号，而输出则包括显示和控制信号。由于输入、输出信号都已介绍，这里就不多赘述。

因为去抖模块的反复使用，本小节将补充解释顶层模块中的去抖部分，以及引入的一些用于去抖模块的中间变量。如下表2-10所示。

表2‑10 顶层模块模块输入、输出变量

|  |  |  |
| --- | --- | --- |
| 消抖模块 | 中间变量 | 功能 |
| debounce\_key\_start\_stop | key\_p\_flag\_key\_start\_stop | 去抖处理暂停/恢复按键 |
| debounce\_key\_load | key\_p\_flag\_key\_load | 去抖处理复位按键 |
| debounce\_state\_choose | key\_p\_flag\_state\_choose | 去抖处理模式选择按键 |
| debounce\_key\_plus | key\_p\_flag\_key\_plus | 去抖处理增加1按键按键 |

代码如下所示：



# 系统调试

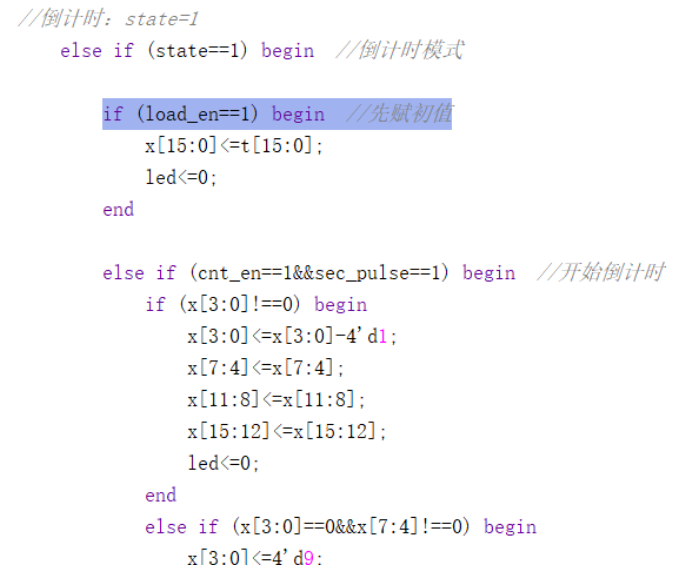
## 使用时难以区分计时状态

问题描述：在项目的初期设计中，LED灯没有在正计时和倒计时过程中做出相应的显示，这使得用户在使用时无法直观地了解当前计时器的状态。由于FPGA可能存在按键不灵敏的设备问题，当用户按下 state\_choose按键后，无法确定FPGA是否成功识别了按键信号。或者在多次按下state\_choose按键后，无法确定有效的按键信号的个数，不知道最终停在什么状态下。

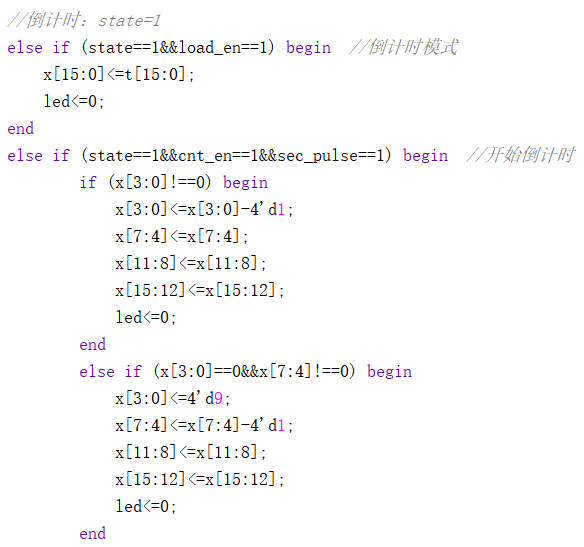
分析与解决：未设计完整的LED灯的状态反馈，导致用户无法明确判断当前系统的工作状态。为了有效解决这些问题，本系统设计LED灯在正计时和倒计时过程中的显示模式，通过led信号和state信号共同控制。本功能设计添加在Countdown\_LED.v模块，额外添加了正计时和倒计时的状态显示。

## 无法正常倒计时

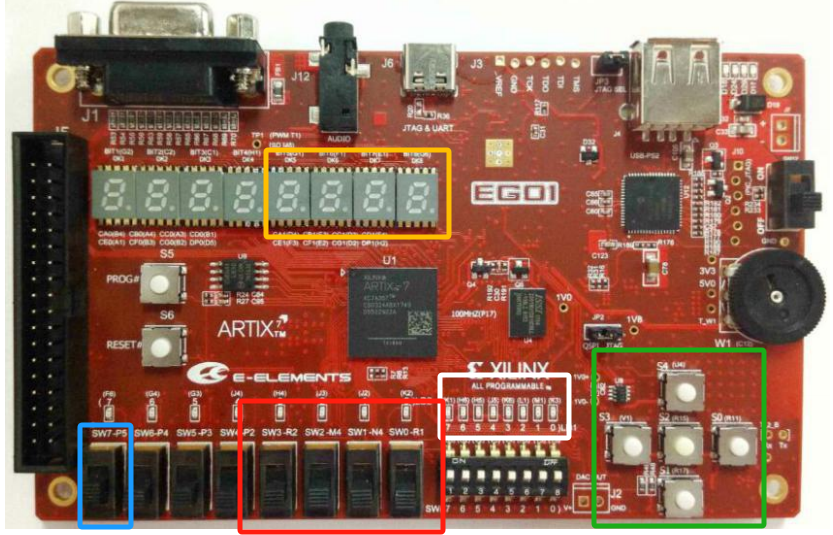
问题描述：在可逆计数器模块中，初始倒计时循环的代码如下图所示，但该代码未能正常运行。



分析与解决：if (load\_en==1)和else if (cnt\_en==1&&sec\_pulse==1)是并列的关系。当state==1,满足load\_en==1这个条件，执行完“倒计时初值设置”后，便跳出这个循环，不会进入else if (state==1)这个语句中，无法继续执行后续的倒计时。因此，需要将load\_en==1和cnt\_en==1&&sec\_pulse==1设置为并列的关系，用两个并列的else if。修改后的代码如下图所示：



# 使用说明



橙黄色框：显示时间数码管，从左到右分别为分钟十位、个位，秒钟十位、个位。

其余按键/开关功能如下表3-1所示。

表3-1 按键说明

|  |  |
| --- | --- |
| 引脚编号 | 功能 |
| SW7-P5（蓝色框） | 异步复位开关 |
| SW0-R1（红色框） | 选中秒钟低位开关 |
| SW1-N4（红色框） | 选中秒钟高位开关 |
| SW2-M4（红色框） | 选中分钟低位开关 |
| SW3-R2（红色框） | 选中分钟高位开关 |
| S0（绿色框） | 暂停/恢复计时按键 |
| S1（绿色框） | 复位按键 |
| S3（绿色框） | 切换正/倒计时按键 |
| S4（绿色框） | 初始时间增加1按键 |

# 总结

## 所作工作

本次综合设计利用Verilog HDL语言，以Vivado软件为工具、EGO1可编程逻辑器件为载体实现了一个用于厨房的定时器，旨在通过简单而有效的控制逻辑，帮助用户轻松管理时间，尤其适用于厨房等场景中的定时操作。系统主要由正计时和倒计时两种模式组成，并能够通过按键进行切换，具备了清晰的显示界面和直观的操作方式。通过数码管显示当前计时状态，用户可以清楚地查看时间进度，而通过LED灯的不同状态，系统能够提示当前所处的工作模式。本文介绍了该系统的功能、创新点以及实现过程，并在设计中充分考虑了用户需求，确保了系统的易用性与高效性。

系统的基础功能包括正计时功能，初始状态为“0000”，用户可以通过按键控制计时的开始、暂停和清零（非系统复位）。计时功能支持最多 99 分 59 秒的显示，满足常见计时需求。数码管清晰地展示了当前的计时进度，用户可以随时观察到时间的变化。正计时模式下，系统始终保持计时，用户可以在任何时刻暂停或清零计时，并通过按键恢复到初始状态。

附加功能方面，系统还提供了倒计时功能，用户可以通过按键在正计时和倒计时模式之间进行切换。倒计时模式下，用户可以设置最大为 99 分 59 秒的倒计时时间，并通过按键控制开始、暂停、清零操作。在倒计时过程中，系统允许用户随时按键复位到设定的初始时间，灵活应对不同的需求。特别地，倒计时结束时，系统通过提醒灯LD1的所有灯亮起，明确地提示用户倒计时已完成。

创新方面，本系统采用了通过LED灯状态来指示当前工作模式（正计时、倒计时或计时结束）。正计时模式下，LD1的0-3灯亮起；倒计时模式下，LD1的4-7灯亮起；倒计时结束时，LD1的所有灯都会亮起，直观地向用户传递状态信息。此设计不仅提升了用户的操作体验，也使得系统的状态转换更加清晰易懂。另一个创新点在于倒计时初始时间的设置，用户可以方便快捷地通过按键设定倒计时的时间，增强了系统的灵活性和实用性。

## 课程收获

在本次课程中，我们每周都会学习一个模块的设计与编程，包括数码管显示、消抖的实现、计数器设计、有限状态机等内容。课程的最后一周，我们完成了一项大作业，涵盖了从功能设计、编程实现到系统调试的整个过程。尽管在短短一周内完成如此复杂的任务对我来说是一个不小的挑战，但这一过程让我受益匪浅。在完成设计和编程的过程中，我不仅巩固了数电知识，还提高了我的动手实践能力，尤其是那些平时课堂上难以接触到的实战经验，这让我收获颇丰。通过此次实践，我更加深刻地理解了理论与实际应用之间的联系，并提高了解决问题的实际能力。

在编写程序的过程中，我系统地学习了Verilog HDL语言，并结合模块化的编程设计思想，将一个复杂的任务拆解成多个可以逐步解决的小问题。同时，学习到了用有限状态机来实现时序电路的设计。我学会了将复杂的系统设计简化成一系列清晰、可控的状态转移过程。这种设计思想培养了我更加严谨、条理清晰的解决问题的能力。

总的来说，通过此次课程，我对整个项目的设计流程有了更加全面的认识，同时也锻炼了我学习编程语言、利用网络搜索解决问题的能力，将为我今后的学习和工作奠定坚实的基础。

## 心得体会

这次综合设计在开始之前，我觉得事情非常多，没有信心自己能够独立完成。后来把任务拆解成一个个小任务，每天安排一部分，不懂的问题及时请教同学，最后还是圆满完成了。这让我学会了做计划、将复杂问题拆解成简单的小任务的能力。

我特别感谢那些在我完成综合设计过程中给予我帮助的朋友。他们对我的编程给予的建议和指导，尤其是在调试代码时，帮助我解决了很多棘手的问题，也给我提供了不同的解决思路，从而高效推进项目。

编写程序和调试的过程确实充满了挑战，尤其是遇到问题时需要耐心和细心去解决。但当我终于完成了功能设计，我十分有成就感，而且我也渐渐地在学习、上手这一门编程语言中，体会到了用编程解决问题的乐趣。

## 课程建议

**加强实验过程中的实时反馈与指导：**在课程的实验环节中，我们遇到了一些设计和编程上的困难。如果能够在实验过程中安排几个助教实时指导，或者建一个答疑群便于我们在预习的时候讨论问题，这样能够帮助我们更高效地解决问题。

# 附 录

1. 顶层文件kitchen\_timer\_top.v

`timescale 1ns / 1ps

module kitchen\_timer\_top(

input clk,rst\_n,

input state\_choose, //正计时倒计时 模式选择按键

input key\_start\_stop, //开始暂停

input key\_load, //置数

input second\_low,second\_high,minute\_low,minute\_high, //控制调节按钮

input key\_plus, //调节按钮（加数）

output wire [6:0] a\_to\_g,

output wire [3:0] an,

output wire [7:0] light

);

//中间变量

wire state,cnt\_en,load\_en; //FSM

wire sec\_pulse; //sec\_pulse

wire [15:0]t; //Countdown\_time\_setup

wire [15:0]x,led; //cnt

wire key\_p\_flag\_key\_start\_stop,key\_p\_flag\_key\_load,key\_p\_flag\_state\_choose,key\_p\_flag\_key\_plus; //消抖的

debounce debounce\_key\_start\_stop(

.clk(clk),

.rst\_n(rst\_n),

.key(key\_start\_stop),

.key\_p\_flag(key\_p\_flag\_key\_start\_stop)

);

debounce debounce\_key\_load(

.clk(clk),

.rst\_n(rst\_n),

.key(key\_load),

.key\_p\_flag(key\_p\_flag\_key\_load)

);

debounce debounce\_state\_choose(

.clk(clk),

.rst\_n(rst\_n),

.key(state\_choose),

.key\_p\_flag(key\_p\_flag\_state\_choose)

);

debounce debounce\_key\_plus(

.clk(clk),

.rst\_n(rst\_n),

.key(key\_plus),

.key\_p\_flag(key\_p\_flag\_key\_plus)

);

FSM uut1\_FSM(

.clk(clk),

.rst\_n(rst\_n),

.key\_start\_stop(key\_p\_flag\_key\_start\_stop),

.key\_load(key\_p\_flag\_key\_load),

.state\_choose(key\_p\_flag\_state\_choose),

.state(state),

.cnt\_en(cnt\_en),

.load\_en(load\_en)

);

sec\_pulse uut2\_sec\_pulse(

.clk(clk),

.rst\_n(rst\_n),

.cnt\_en(cnt\_en),

.sec\_pulse(sec\_pulse)

);

Countdown\_time\_setup uut3\_time\_set(

.clk(clk),

.rst\_n(rst\_n),

.load\_en(load\_en),

.second\_low(second\_low),

.second\_high(second\_high),

.minute\_low(minute\_low),

.minute\_high(minute\_high),

.key\_plus(key\_p\_flag\_key\_plus),

.t(t)

);

reversible\_cnt uut4\_cnt(

.clk(clk),

.rst\_n(rst\_n),

.cnt\_en(cnt\_en),

.sec\_pulse(sec\_pulse),

.state(state),

.load\_en(load\_en),

.t(t),

.x(x),

.led(led)

);

display uut5\_display(

.clk(clk),

.rst\_n(rst\_n),

.x(x),

.a\_to\_g(a\_to\_g),

.an(an)

);

Countdown\_LED uut6\_Countdown\_LED(

.clk(clk),

.rst\_n(rst\_n),

.state(state),

.led(led),

.light(light)

);

endmodule

1. 消抖模块debounce.v

`timescale 1ns / 1ps

module debounce(

input clk,

input rst\_n,

input key,

output reg key\_p\_flag

);

parameter DELAY\_TIME=21'd1500000; //延时15ms

//key\_reg:按键输入寄存器

reg key\_reg;

always @(posedge clk or negedge rst\_n)

if(!rst\_n)

key\_reg<=0;

else

key\_reg<=key;

//delay\_cnt延时计数器

reg [20:0] delay\_cnt;

always @(posedge clk or negedge rst\_n)

if(!rst\_n)

delay\_cnt<=21'b0;

else if(key!=key\_reg)

delay\_cnt<=DELAY\_TIME;

else if(delay\_cnt>0)

delay\_cnt<=delay\_cnt-1'b1;

else

delay\_cnt<=21'b0;

//有效变化

reg key\_value;

always @(posedge clk or negedge rst\_n)

if(!rst\_n)

key\_value<=1'b0;

else if(delay\_cnt==1'b1)

key\_value<=key;

else

key\_value<=key\_value;

//key\_p\_flag:按键上升标志信号

always@(posedge clk or negedge rst\_n)

if(!rst\_n)

key\_p\_flag<=1'b0;

else if(delay\_cnt==1'b1&&key==1)

key\_p\_flag<=1'b1;

else

key\_p\_flag<=1'b0;

endmodule

1. 有限状态机FSM.v

`timescale 1ns / 1ps

module FSM(

input clk,

input rst\_n,

input state\_choose, //模式选择

input key\_start\_stop, //暂停恢复计时信号

input key\_load, //置数信号

output reg state, //模式输出：0=正计时,1=倒计时

output reg cnt\_en, //计数使能信号

output reg load\_en //有无置数

);

//状态编码

reg [2:0] cstate; //存现态

reg [2:0] nstate=0; //存次态

parameter

//正计时

WORK\_INIT0=3'b000,COUNT0=3'b001,PAUSE0=3'b010,

//倒计时

WORK\_INIT1=3'b100,COUNT1=3'b101,PAUSE1=3'b110;

//第一段，状态转移：cstate 更新

always @(posedge clk or negedge rst\_n)

if (!rst\_n)

cstate<=WORK\_INIT0;

else

cstate<=nstate;

//第二段，次态判断：nstate

always @ \*

begin

case (cstate)

//正计时 state=0

WORK\_INIT0:

if (key\_start\_stop==1) nstate=COUNT0;

else if (state\_choose==1) nstate=WORK\_INIT1;

else nstate = cstate;

COUNT0:

if (key\_start\_stop==1) nstate=PAUSE0;

else nstate = cstate;

PAUSE0:

if (key\_start\_stop==1) nstate=COUNT0;

else if (key\_load==1) nstate=WORK\_INIT0;

else nstate=cstate;

//倒计时 state=1

WORK\_INIT1:

if (key\_start\_stop==1) nstate=COUNT1;

else if (state\_choose==1) nstate=WORK\_INIT0;

else nstate=cstate;

COUNT1:

if (key\_start\_stop==1) nstate=PAUSE1;

else if (key\_load==1) nstate=WORK\_INIT1;

else nstate=cstate;

PAUSE1:

if (key\_start\_stop==1) nstate=COUNT1;

else if (key\_load==1) nstate=WORK\_INIT1;

else nstate=cstate;

default:nstate=WORK\_INIT0;

endcase

end

//第三段，次态输出

//cnt\_en:有无计数，在计数的话就是1

always @ (posedge clk or negedge rst\_n)

if (!rst\_n)

cnt\_en<=0;

else if(cstate==COUNT0)

cnt\_en<=1;

else if(cstate==COUNT1)

cnt\_en<=1;

else cnt\_en<=0;

//load\_en:有无置数，有的话就是1

always @ (posedge clk or negedge rst\_n)

if (!rst\_n)

load\_en<=1;

else if(cstate==COUNT1||cstate==PAUSE1)//倒计时

load\_en<=0;

else if(cstate==COUNT0||cstate==PAUSE0)//正计时

load\_en<=0;

else load\_en<=1;

//state：模式，正计时=0 倒计时=1

always @ (posedge clk or negedge rst\_n)

if (!rst\_n)

state<=0; //复位是先设置为正计时

else if (cstate==COUNT1) state<=1;

else if (cstate==PAUSE1) state<=1;

else if (cstate==WORK\_INIT1) state<=1;

else if (cstate==COUNT0) state<=0;

else if (cstate==PAUSE0) state<=0;

else if (cstate==WORK\_INIT0) state<=0;

else state<=0;

endmodule

1. 秒脉冲发生器sec\_pulse.v

`timescale 1ns / 1ps

module sec\_pulse(

input clk,

input rst\_n,

input cnt\_en, //暂停开始使能

output reg sec\_pulse

);

parameter fequency=27'd100\_000\_000; //100Mhz分频

parameter max\_counter=fequency-1;

reg [30:0] counter; //足够大

//counter 时钟

always@(posedge clk or negedge rst\_n)

if (!rst\_n)

counter<=0;

else if (counter==max\_counter)

counter<=0;

else

counter<=counter+1;

//sec\_pulse

always@(posedge clk or negedge rst\_n)

if (!rst\_n)

sec\_pulse<=0;

else if (counter==max\_counter)

sec\_pulse<=1;

else

sec\_pulse<=0;

endmodule

1. 可逆计数器reversible\_cnt.v

`timescale 1ns / 1ps

module reversible\_cnt(

input clk,

input rst\_n,

input sec\_pulse,

input state,

input cnt\_en,

input load\_en,

input [15:0] t, //倒计时初值

output reg [15:0] x,

output reg led //提醒LED

);

//正计时：state=0

always@(posedge clk or negedge rst\_n)

if (!rst\_n) begin

x<=16'd0;

led<=0;

end

//正计时模式

else if (sec\_pulse==1&&cnt\_en==1&&state==0) begin

if (x[3:0]!==9) begin

x[3:0]<=x[3:0]+4'd1;

x[7:4]<=x[7:4];

x[11:8]<=x[11:8];

x[15:12]<=x[15:12];

led<=0;

end

else if (x[3:0]==9&&x[7:4]!==5) begin

x[3:0]<=4'd0;

x[7:4]<=x[7:4]+4'd1;

x[11:8]<=x[11:8];

x[15:12]<=x[15:12];

led<=0;

end

else if (x[3:0]==9&&x[7:4]==5&&x[11:8]!=9) begin

x[3:0]<=4'd0;

x[7:4]<=4'd0;

x[11:8]<=x[11:8]+4'd1;

x[15:12]<=x[15:12];

led<=0;

end

else if (x[3:0]==9&&x[7:4]==5&&x[11:8]==9&&x[15:12]!=9) begin

x[3:0]<=4'd0;

x[7:4]<=4'd0;

x[11:8]<=4'd0;

x[15:12]<=x[15:12]+4'd1;

led<=0;

end

else if (x[3:0]==9&&x[7:4]==5&&x[11:8]==9&&x[15:12]==9) begin //计时结束

x[15:0]<=x[15:0];

led<=1;

end

end

else if (load\_en==1&&state==0)begin

x<=16'd0;

led<=0;

end

//倒计时：state=1

else if (state==1&&load\_en==1) begin //倒计时模式

x[15:0]<=t[15:0];

led<=0;

end

else if (state==1&&cnt\_en==1&&sec\_pulse==1) begin //开始倒计时

if (x[3:0]!==0) begin

x[3:0]<=x[3:0]-4'd1;

x[7:4]<=x[7:4];

x[11:8]<=x[11:8];

x[15:12]<=x[15:12];

led<=0;

end

else if (x[3:0]==0&&x[7:4]!==0) begin

x[3:0]<=4'd9;

x[7:4]<=x[7:4]-4'd1;

x[11:8]<=x[11:8];

x[15:12]<=x[15:12];

led<=0;

end

else if (x[3:0]==0&&x[7:4]==0&&x[11:8]!==0) begin

x[3:0]<=4'd9;

x[7:4]<=4'd5;

x[11:8]<=x[11:8]-4'd1;

x[15:12]<=x[15:12];

led<=0;

end

else if (x[3:0]==0&&x[7:4]==0&&x[11:8]==0&&x[15:12]!=0) begin

x[3:0]<=4'd9;

x[7:4]<=4'd5;

x[11:8]<=4'd9;

x[15:12]<=x[15:12]-4'd1;

led<=0;

end

else if (x[3:0]==0&&x[7:4]==0&&x[11:8]==0&&x[15:12]==0) begin //倒计时结束

x[15:0]<=x[15:0];

led<=1;

end

end

endmodule

1. 倒计时初始时间设定Countdown\_time\_setup.v

`timescale 1ns / 1ps

module Countdown\_time\_setup(

input clk,

input rst\_n,

input load\_en,

input second\_low,

input second\_high,

input minute\_low,

input minute\_high,

input key\_plus,

output reg [15:0] t //时间

);

reg key\_plus\_last;//保存当前输入状态到上一个时钟周期的寄存器

reg rising\_edge;//上升沿

always @(posedge clk) begin

key\_plus\_last<=key\_plus;

rising\_edge<=~key\_plus\_last&key\_plus; // 检测上升沿

end

always@(posedge clk or negedge rst\_n) begin

if (!rst\_n) t[15:0]<=16'd0;

else if (rising\_edge) begin

if (second\_low==1&&t[3:0]==9) t[3:0]<=4'd0;

else if (second\_low==1&&t[3:0]!=9) t[3:0]<=t[3:0]+4'd1;

else if (second\_high==1&&t[7:4]==5) t[7:4]<=4'd0;

else if (second\_high==1&&t[7:4]!=5) t[7:4]<=t[7:4]+4'd1;

else if (minute\_low==1&&t[11:8]==9) t[11:8]<=4'd0;

else if (minute\_low==1&&t[11:8]!=9) t[11:8]<=t[11:8]+4'd1;

else if (minute\_high==1&&t[15:12]==9) t[15:12]<=4'd0;

else if (minute\_high==1&&t[15:12]!=9) t[15:12]<=t[15:12]+4'd1;

end

else t<=t;

end

endmodule

1. 数码管显示器display.v

`timescale 1ns / 1ps

module display(

input clk,

input rst\_n,

input [15:0]x,

output reg [6:0]a\_to\_g, //控制7个管，段信号

output reg [3:0]an //位选，分钟、秒数各两个

);

//clkdiv 时钟分频

reg [20:0] clkdiv;

always @(posedge clk, negedge rst\_n)

begin

if(!rst\_n) clkdiv<=21'd0;

else clkdiv<=clkdiv+1;

end

//bitcnt 位扫描

wire [1:0] bitcnt;

assign bitcnt=clkdiv[20:19];

//an位选信号的产生，高有效

always@\*

begin

if(!rst\_n)

an=4'b0;

else begin

an=4'b0;

an[bitcnt]=1;

end

end

//digit 保存当前显示的数字

reg [3:0] digit;

always @\* begin

if(!rst\_n)

digit=4'd0;

else

case(bitcnt)

2'd0:digit=x[3:0];

2'd1:digit=x[7:4];

2'd2:digit=x[11:8];

2'd3:digit=x[15:12];

endcase

end

//译码

always@(\*) begin

if(!rst\_n)

a\_to\_g=7'b1111111;

else case(digit)

0:a\_to\_g=7'b1111110;

1:a\_to\_g=7'b0110000;

2:a\_to\_g=7'b1101101;

3:a\_to\_g=7'b1111001;

4:a\_to\_g=7'b0110011;

5:a\_to\_g=7'b1011011;

6:a\_to\_g=7'b1011111;

7:a\_to\_g=7'b1110000;

8:a\_to\_g=7'b1111111;

9:a\_to\_g=7'b1111011;

'hA:a\_to\_g=7'b1110111;

'hB:a\_to\_g=7'b0011111;

'hC:a\_to\_g=7'b1001110;

'hD:a\_to\_g=7'b0111101;

'hE:a\_to\_g=7'b1001111;

'hF:a\_to\_g=7'b1000111;

default:a\_to\_g=7'b1111110; //0

endcase

end

endmodule

1. LED显示模块Countdown\_LED.v

`timescale 1ns / 1ps

module Countdown\_LED(

input clk,

input rst\_n,

input led,

input state,

output reg [7:0] light

);

always@(posedge clk,negedge rst\_n)

if(!rst\_n) begin

light[0]<=4'd0;light[1]<=4'd0;light[2]<=4'd0;light[3]<=4'd0;

light[4]<=4'd0;light[5]<=4'd0;light[6]<=4'd0;light[7]<=4'd0;

end

else if (state==0&&led==0) begin //正计时

light[0]<=4'd1;light[1]<=4'd1;light[2]<=4'd1;light[3]<=4'd1;

light[4]<=4'd0;light[5]<=4'd0;light[6]<=4'd0;light[7]<=4'd0;

end

else if (state==1&&led==0) begin //倒计时

light[0]<=4'd0;light[1]<=4'd0;light[2]<=4'd0;light[3]<=4'd0;

light[4]<=4'd1;light[5]<=4'd1;light[6]<=4'd1;light[7]<=4'd1;

end

else if (led==1) begin //计时结束

light[0]<=4'd1;light[1]<=4'd1;light[2]<=4'd1;light[3]<=4'd1;

light[4]<=4'd1;light[5]<=4'd1;light[6]<=4'd1;light[7]<=4'd1;

end

else begin

light[0]<=4'd0;light[1]<=4'd0;light[2]<=4'd0;light[3]<=4'd0;

light[4]<=4'd0;light[5]<=4'd0;light[6]<=4'd0;light[7]<=4'd0;

end

endmodule