

****

**工程综合设计（I）报告**

**题目： 基于FPGA的电子日历设计**

**报告人： 陈亦 学号： 20193715 班级： 弘深19电气电子**

**同组人： 杨涛 学号： 20193817 班级： 弘深19电气电子**

**重庆大学电气工程学院**

**2020年6月**

**完成情况及成绩：**

**指导教师：**

**目 录**

[**1工程综合设计概述 1**](#_Toc76333516)

[**1.1设计目的及意义 1**](#_Toc76333517)

[**1.2 设计要求 1**](#_Toc76333518)

[**1.3 设计任务 1**](#_Toc76333519)

[**1.4设计环境 1**](#_Toc76333520)

[**2工程综合设计内容 2**](#_Toc76333521)

[**2.1设计思路 2**](#_Toc76333522)

[**2.2设计方案及原理分析 3**](#_Toc76333523)

[**2.3电路（程序）仿真结果及分析 6**](#_Toc76333524)

[**2.4实物（程序）调试结果及分析 8**](#_Toc76333525)

[**2.5设计创新点 11**](#_Toc76333526)

[**2.6设计中遇到的主要问题及解决方法 11**](#_Toc76333527)

[**3总结与心得 12**](#_Toc76333528)

[**4参考文献（5-8篇） 13**](#_Toc76333529)

[**附 录1（源程序，加注释） 14**](#_Toc76333530)

## 1工程综合设计概述

### 1.1设计目的及意义

1. 学会Verilog HDL语言，掌握verilog HDL的语法规则

2. 掌握用VIVADO软件编译verilog HDL的方法与步骤

3. 熟悉EGO1开发板的构成与功能，基于Verilog HDL进行较复杂数字逻

辑系统的设计、仿真和调试；

4. 通过规范化撰写设计报告，培养科技写作与电子文书工具的应用能力。

### 1.2 设计要求

1. 小组（2人）合作，查阅资料，设计规定要求与功能的电子系统

2. 完成程序设计、仿真、调试、排除故障和下载验证

3. 每人独立撰写规范的实验报告，注明自己的贡献度，截止于第20周

五下午3点交报告和源程序给老师留存。

### 1.3 设计任务

1. 各组制定出详细设计方案，明确成员各自分工，认真记载设计工作日记；

2. 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。

3. 认真记录在设计过程中遇到的主要困难、故障以及解决方法，在设计报告 中展示经验积累；

4. 生成bit文件，下载到开发板上，验证设计的正确性；

5. 撰写设计报告。

6. 对于没有完成要求的设计，要求对存在的问题进行分析，提出改进意见。

### 1.4设计环境

设计采用Xilinx Artix-7系列XC7A35T-1CSG324C板卡，板卡电源采用Type-C/外接直流电源，系统时钟资源为100MHZ(P17)，存储资源有一块8Mbit的SRAM，以及一块SPI Flash (N25Q64-3.3v)。

## 2工程综合设计内容

### 2.1设计思路

采用多个模块相互配合的方式，涉及到的模块有分频模块、消抖模块、计时模块（日、月、年计时模块、判断模块）、备注模块、时间同步、显示模块、顶层模块。

**顶层模块**：输入有100MHz时钟信号，复位信号，停止（调时）、上、下、左、右五个按键，调速、同步、时间切换、备忘四个开关。按下复位键，日历复位，时间变为1949年10月1日15点整。按下停止（调时）键后，进入调整时间模式，可按左右键进行调整时间的选择（年、月、日），按上下可进行日期的加减。四个开关中，备忘开关优先级最高，当此开关闭合，进入备注模式，设定好时间后断开此开关，当时间到达设定时间后，板上的灯亮。时间切换开关可以切换运行的时间（日历模式或是时钟模式）。调速开关为开发过程中为方便调试设计，后发现该功能较为有趣便留下，可以实现日历运行速度的增减。同步开关可以通过串口和树莓派的网络功能将实时时间传输给FPGA开发板，实现时间同步的功能。

**具体各模块的设计思路大体如下：**计时模块是本次项目的核心，包括日计时模块、月计时模块、年计时模块、判断模块四个子模块。判断模块判断当前所在是否为闰年二月、大小月，此输出结果传输给日计时模块，决定在某一日（28日、29日、30日或31日）产生进位。日、月、日计时模块和判断模块四个模块共打包进一个“计时模块”中，实现计数的功能。另外，调整时间功能也融入进该模块中。在设计之初，本小组打算采用实验书中通过一系列if语句嵌套的方式进行设计，但后发现此方法1. 不利于修改，即不利于判断是否为大小月，闰年二月。2. 代码冗长，不利于阅读。因此改而采用现各模块配合的方法。

为方便设计，备注模块可以理解为计数模块的一个复制品，只改动一些输入值。此方法设计简单，但可能存在对FPGA中有限资源的浪费情况，但由于本项目也是一个较小的工程，因此不用担心资源受限的情况。

**时间同步模块**采用串口传输数据，在树莓派端联网获取当前时间(精确到秒)，通过串口发送器下发给FPGA端的串口接收器，每一次日期时间信息共56比特，故至少需要分多帧传输。串口顶层模块负责对多帧数据的起始和结束进行判断，每一次接受完所有56帧日期时间数据并确认无误时，才允许计时模块读取该时间进行同步。物理上，使用连接线将树莓派端作为tx的GPIO引脚与EGO1端作为rx的GPIO引脚相连。

其余模块都只对《实验指导书》中的例程进行了略微修改。显示模块：因为涉及“调整时间”的功能，显示模块需要增加一个“闪烁”功能来显示某位（年或月或日）是出于“可调状态”，因此需要通过计时模块输入“是否可调”和“当前可调位”的信息，增加闪烁功能。消抖模块：使用到了FPGA中的五个按键，实现了一个模块对多个按键消抖的功能，输出量为一个五位宽的向量，分别表示暂停、上、下、左、右。分频模块：与例程类似。

以上模块中，最核心的计时模块总框架，即日、月、年计时模块、特殊月份处理模块(能够实现判断大小月，闰年二月的信息)的基础功能由杨涛搭建，陈亦在该基础上进行修改，增加了调整时间的功能。其他模块中，时间同步模块由杨涛完成，消抖、分频、显示、备注模块由陈亦完成。

### 2.2设计方案及原理分析

**计时模块：**计时模块为本次项目最基础、核心的模块。整体基本工作流程如下图所示：

**图示

描述已自动生成**

图1：计时模块基础功能

除了上图所展示计时功能之外，本模块还能够实现1. 装载初值 2. 调整时间。这两项功能。下面，本文将以日计时器（Dcounter）为例，阐述工作原理。月、日计时器工作原理类似。

图示

描述已自动生成 日记计时器共有9个输入，2个输出。比较月和日计时器，日计时器多了一个输入，该输入为判断模块的判断结果。9个输入中，有2个时钟信号：ini\_clk和clk，前者为未分频的100MHz时钟信号，后者为1Hz的分频后的信号；2个按键信号：btn\_ena和[1:0]btns前者为按键使能位，即当前状态是否为可调状态，后者控制日期的增减；1个复位信号；1个使能信号：使能信号对于日计时器（Dcounter）来说没有作用，但对日、年计时器来说至关重要，该输入即为上一级（月计时器的上一级为日计时器，年计时器的上一级为月计时器）的进位信号；2个赋值信号：load和data，即赋值使能位和赋值日期；1个判断信号：即判断当前是否闰年二月，是否大小月。2个输出信号中，1个位进位信号：输出进位信息，传送至下一级计时器的使能信号；1个日期信号：即当天日期的8421BCD码。具体流程图如右图所示。

设置两个输入时钟信号ini\_clk和clk，前者为细粒度的时钟信号，是为了能够让该日历在调时功能下能够实时修改日信息，后者是为了提供进位信息。若只采用一个粗粒度的1Hz的clk，则会导致在调时功能对时间的增减操作，只能在“一天结束”后才起作用，这对一个产品来说是非常致命的错误。

图2：日计时器模块功能流程图

**时间同步模块：**由于在FPGA直接进行网络编程比较繁琐，并且EGO1实验本身并不附带以太网接口，本方案则退而求其次，采用网络+串口(UART)的形式，配合外设(树莓派4B)，在树莓派端联网获取数据，通过GPIO口连接FPGA，使用串口传输的形式将时间信息下发给FPGA实验板，以实现实时同步数据的功能。

物理连接上，我们首先通过连接线将在树莓派端ttyAMA1，即串口2的发送脚(TXD)与FPGA的串口接收器(本实验约束到EGO1的H17引脚上)相连。

其次在树莓派端，我们采用Python编程。程序流程上，如下：1.首先使用time库获取时间数据；2.转换成为16进制形式；3.通过Serial库很方便地初始化串口；4.通过Serial库将时间数据传输到TXD口上。对于第1、2、4条，每0.5秒左右重复获取一次，保证每一秒的数据都能得到刷新，而又不会由于刷新太快导致显示产生闪烁，影响视觉效果。程序流程图如下：

在FPGA端，我们没有(除蓝牙外)现成可用的串口模块，因此我们采用并改进正点原子设计的UART串口接收器(RXD)，基本配置为“8位数据位，1位停止位，波特率115200bps”(与树莓派端初始化配置相同)，并添加“1位偶校验位”的功能来减少传输扰动造成时间同步错误的可能。在此之上，由于日期+时间数据共14个数字，采用BCD码形式，需要传输14\*4=56 bits的数据，而每帧仅有8个数据位(bits)，所以一次日期时间数据至少分7帧才能够传输完成。对于这部分的设计，遵循同接受底层模块recv\_txd一样的方法，即采用状态机来实现。具体来讲，我们规定要想通过传输时间，首帧(第1帧)必须为起始位，即ASCII码的“~”，二进制表示为0111\_1110，任何时候收到此帧即开启接收过程；第2~9帧为日期数据；第10帧为校验位，必须为1000\_1000，否则判断为传输失败，将从头开始接收起始位；第11~16帧为时间数据。状态转换图如下：

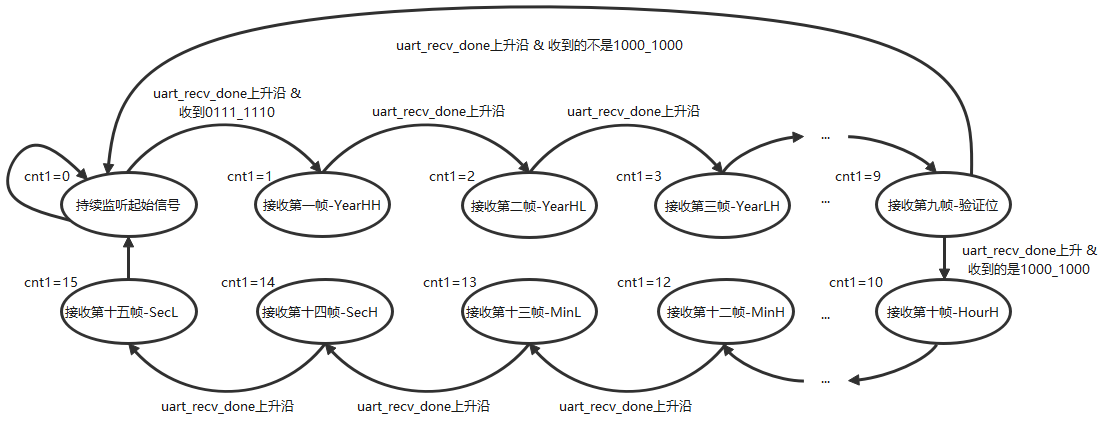


图4：串口多帧数据接收器的状态转换图

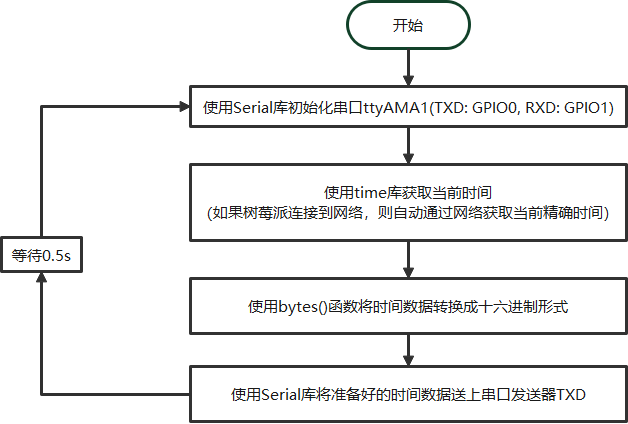


图3：树莓派端Python程序的程序流程图

具体每一帧的接收过程，由子模块uart\_recv完成，参考《领航者ZYNQ之 FPGA开发指南V1.3》P288-P306，这里不再赘述。但是需要明确：其中数据位可配置为5、6、7、8位，停止位可配置为1、1.5、2位，在原设计中采用的配置是：数据位为 8 位，停止位为 1 位，无校验位，波特率为 115200bps。

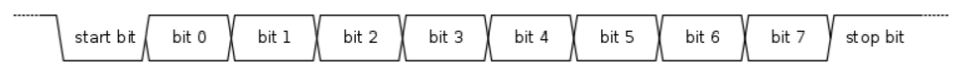


图5：一个典型的串口时序图（8为数据位，1位停止位，没有奇偶校验位）

在此之上，我们增加了一位奇校验位，以排除传输过程中受到干扰产生的无效数据。实现思路非常简单，将8个数据位做异或运算，得1即表示其中有奇数个1，则校验位置0；得0则有偶数个1，需要将校验位置1。

**分频模块：**该分频模块都采用偶数分频，需要把原先的100MHz的时钟信号分频为1Hz的信号。做一个加法计数器，当计数器从0技术到50M-1，即49999999时，输出时钟进行一次反转，如此循环。该1Hz的信号为日期的进位信号。另外，该模块还设置了一个50Hz的输出信号，设定的加法计数器的最大值为9999999，该信号输出给按键模块作为消抖。

**译码模块：**该模块中涉及到两个分频：一个是循环扫描的分频，另一个是闪烁分频。前者的分频系数为50000，频率约100Hz，后者闪烁分频的分频系数为10000000，闪烁频率约为5Hz。（因为此处时钟频率没有时钟分频要求这么高，可以不用在意一点点的误差）设置闪烁的原理非常简单，即对最后的位选输出信号和5Hz的闪烁时钟做一次“与”操作。即可实现闪烁功能。

### 2.3电路（程序）仿真结果及分析

各模块仿真结果如下：

**1. 时钟分频模块**

图形用户界面

低可信度描述已自动生成

输入信号clk1为100MHz，输出信号clk\_out为1Hz，clk\_btn为50Hz。实现分频。

**2. 按键消抖模块**

日程表

描述已自动生成

btn[5:0]分别为：复位、暂停（调时）、上、下、左、右。输入时钟clk为分频模块的clk\_btn，btns[5:0]为输入，out\_btn为输出，输出信号时钟晚输入信号至少两个时钟周期，至多三个时钟周期。实现了消抖功能。temp为仿真中为方便操作的中间变量，与实际源代码无关，可不予理会。

**3. 显示译码模块**

**表格

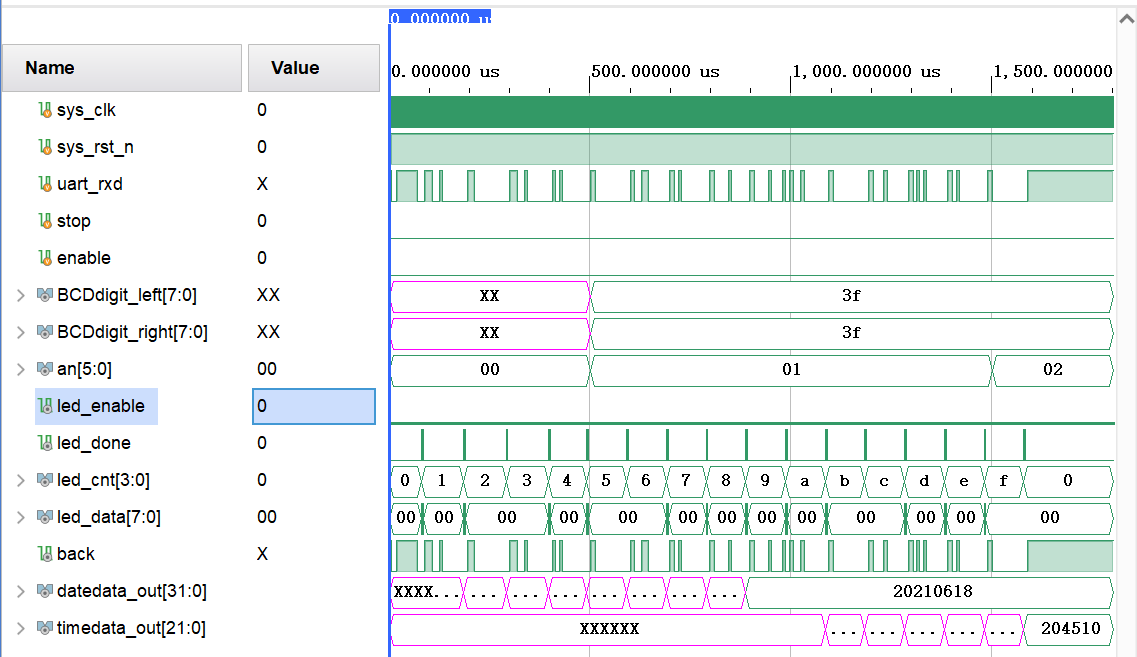
描述已自动生成**

显示模块仿真结果如上，因为该部分行为仿真实际意义不大，往往与上板后结果不一致，实验过程中小组直接上板进行验证。上图表示显示译码模块能够正确显示数字，1949.10.01。其中test\_code为8段led对应的相应数字，是在调试中为方便观察所添加，无实际意义。

**4. 时间同步模块**

cnt1作为状态机的状态变量，同时标志着接收到的帧数，对于准确接收多帧数据十分重要，此处将其用led\_cnt引出，便于观察，其有效状态为0~15。

同理datedata\_out和timedata\_out也是将内部的datedata和timedata引出，当接收到第8帧、15帧数据的下一个上升沿时，这两个变量储存的分别是当前的完整日期数据和时间数据，将可供计数模块读取。



下图为收到起始信号cnt1置1。并可见uart\_rev\_data(即引出的led\_data)的数据在到下一帧数据到达时会被清空，大部分时间其实是处于00状态，这也是为什么要用datedata和timedata将收到的数据及时暂存起来。

图片包含 日历

描述已自动生成

### 2.4实物（程序）调试结果及分析

1. 正常日历：显示时间为1949.10.01。

电子零件

中度可信度描述已自动生成

2. 后台时钟：此时的时间为15：49：41

电子仪器

中度可信度描述已自动生成

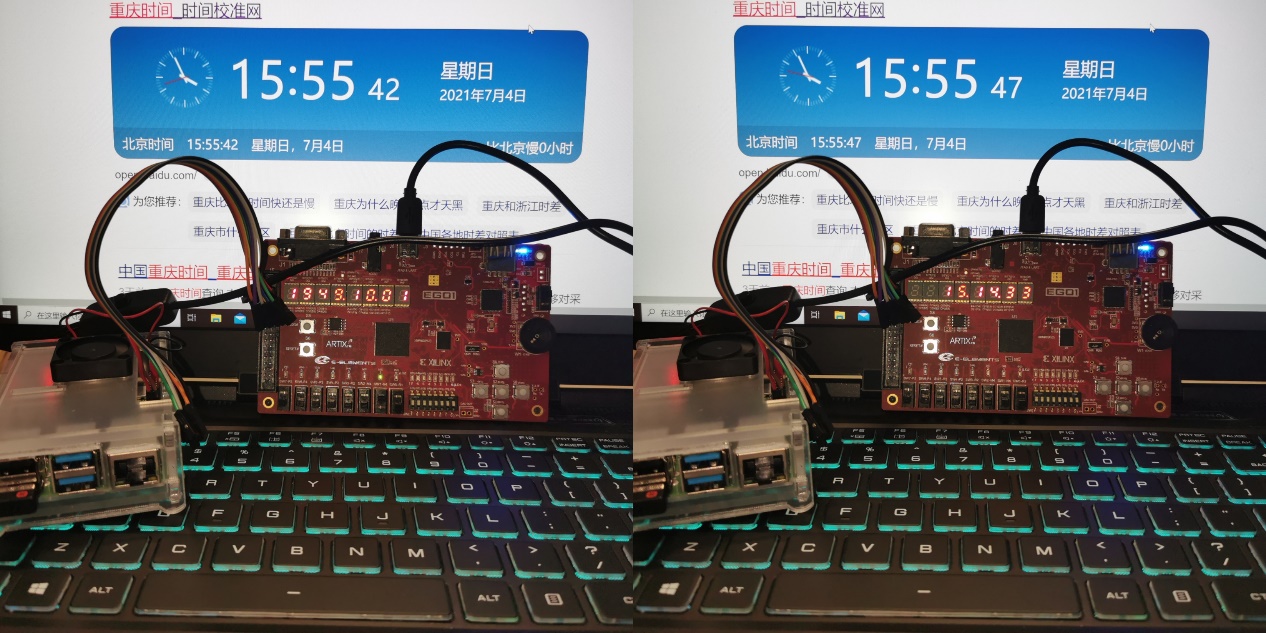
3. 备注功能：当时间达到设定时间时（此次设定时间为1949.12.06，即图中所示时间）小灯K3、M1、L1、K6亮。

电子零件

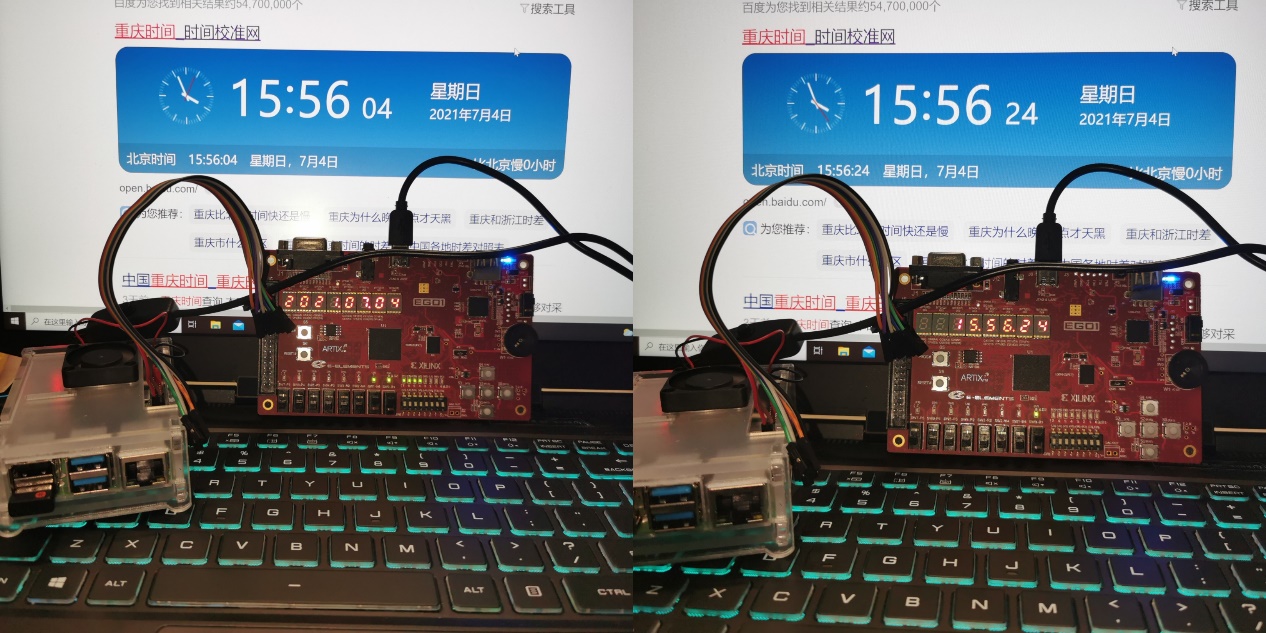
中度可信度描述已自动生成

4.时间同步功能

同步开启前(复位状态)：



同步开启后(误差在秒级以内)：



### 2.5设计创新点

1. 可实现对年、月、日时间的分别单独调整。

2. 通过数码管的闪烁来告知使用者当前可调的时间单位。

3. 可实现精确的网络时间同步，并可自动识别所在时区。

### 2.6设计中遇到的主要问题及解决方法

此次设计主要面对两大问题：1. 对并行语句的理解。 2. 行为仿真与上板验证结果不符。

问题一主要发生在设计的开始阶段，虽然数电实验中已经做过几个FPGA实验，但由于当时几乎都是书上现成的代码，只要做一些小小的修改就能完成实验要求，因此对Verilog并行语句并没有太过深入的理解。设计之初，在我的潜意识中还是将Verilog语句当成了串行语句，把写python的思维带到了Verilog中。因此在编写过程中发生了一个致命错误，在多个always块中对同一个变量进行赋值。在电路中，多个信号同时对一个触发器进行赋值是明显不可能存在的现象，Vivado理应会报错（但当时第一次调试的时候它并没有报错，甚至通过了行为仿真）。后通过CSDN等网络论坛详细了解了Verilog语法的原理，将多个always块合并至一个always块中，统一使用非阻塞赋值成功解决了该问题。

问题二几乎是伴随着整个项目全过程。当我花了一个一个下午加晚上成功理解了Verilog语法并解决了第一个问题之后，我本以为未来会是一条康庄大道。可随之而来的是更加令人崩溃的问题，这个问题一共纠缠了我整整三天的时间。前文提到将多个语句合并到一个always块中能有效解决问题一。但这带来一个新的问题，always块中的触发信号会非常多，如时钟信号、复位信号、五个按键信号等等，都会作为触发信号，但是这些信号中，能够稳定、且能派上用场的，只有前两者。如果把按键信号当作触发信号，行为仿真可以通过，但上板后该日历的“时间调整”功能会非常不稳定，往往会按一个按键连跳几十，甚至几百个单位（起初甚至以为是消抖模块没有做好，后发现不是）。后通过“打拍”，即通过另一个always块中的两个寄存器检测上升沿，在“主always块”中统一使用系统时钟和复位信号来作触发信号来解决该问题。我们分析问题，认为可能的原因有二：1. FPGA中查找表后接的FF输出信号不稳定，不能直接做为触发信号。2. 时序不收敛。在思考后，我们通过排除法认为是时序不收敛的可能性更大，因为如果是信号不稳定，打拍检测上升沿的方法应该也无法解决此问题。但因为设计处于考试周，时间紧迫，没有充足的时间学习Vivado中的时序分析工具。这也是我们小组计划在假期期间拓展学习的部分。

另外设计过程中还有一些其他小问题的存在。一、 在设计消抖模块时，尝试把“复位”键也放进消抖里，结果发现按下复位之后，输出一直是“复位”，直到按下另外一个按键后原先的“复位”才被取消掉。后发现代码中的btn0，1，2寄存器会一直出0，直到另一个按键按下之后才能停止“复位”。将复位的优先级置于消抖模块之上，即消抖模块无需对“复位”进行消抖，在此思路的指导下成功完成多个按键消抖的模块。二、 由于在一开始设计中，计时模块的always块中的触发信号为分频后的1Hz的时钟信号，因此在设计“时间调整”拓展功能时，发现按键按下后只有等到下一个时钟周期到来才会改变时间，这意味着在现实生活中，只有等到一天结束才能够响应“时间调整”的操作。这是不能接受的错误。后将计时模块的always块中的触发信号由分频后的1Hz时钟换成了100MHz的系统时钟，即可实现“实时响应”的效果。而原先的1Hz时钟，则通过打拍检测上升沿，融入到always块中。三、 时间错乱，会出现9月30日->10月31日->11月1日这种错误的计时。该问题是由于内部“日、月、年计时器”响应时间冲突所产生的，同样，在对1Hz的时钟进行打拍后即解决了此问题。四、 由于代码编写规范性的欠缺（特别是date和data这两个长相相近的单词让我饱受折磨）也消耗了我大量的查错时间。

## 3总结与心得

本次课程设计大作业，对于我来说是一次很好的锻炼机会。所谓课程设计，那就要求我们根据题目，自顶向下地完成一套可实现的方案，并对方案进行解读。《数字电子技术基础》作为一门基础性的专业课，是一门理论原理与实际应用结合得很紧密的课程。此次课程设计提升了我对FPGA原理的理解，Verilog语言的掌握，也加强了我分析问题、解决问题的能力。一切的一切都使我受益良多。

整个过程中并没有感受到很明显的进步，但是回想刚开始拿到课设题目那是，甚至连Verilog代码都无法独立编写，到后来能够独立编写几百行的代码，这就是进步最好的证明。但是此次课设还是存在一些可以改进之处：1. 所用的按键、开关过多。其实本打算考试结束之后做一个简单的500ms内检测按键连按两次的模块，来减少按键的使用。但是考完试之后实在是心情浮躁，直接放飞自我，也让这个计划不了了之。2. 还可以用到许多外设，如VGA、蜂鸣器等；对FPGA的原理理解还有所欠缺，如还未掌握时序分析的方法，这些也是我打算在暑假进一步学习拓展的内容。

另外在完成作业的过程中，非常感谢陈洁老师的辛勤指导和整个阶段的陪伴，感谢同组成员杨涛同学在繁忙考试周仍几番熬夜为此次设计做出了非常重要的贡献，感谢计算机学院代韵涛同学陪我在一食堂度过了一个非常折磨的调试夜晚。没有他们的帮助我无法完成此次作业。

最后，希望自己能够在未来的学习中保持这份热情，无限努力，永远进步。

送上最好的祝福

陈亦

注：在2.1中的最后一段中有本次设计的任务分工，在此再强调一下：最核心的计时模块总框架，即日、月、年计时模块、特殊月份处理模块(能够实现判断大小月，闰年二月的信息)的基础功能由杨涛搭建，陈亦在该基础上进行修改，增加了调整时间的功能。其他模块中，时间同步模块由杨涛完成，消抖、分频、显示、备注模块由陈亦完成。

## 4参考文献（5-8篇）

[1] EGO1用户手册, 依元素科技有限公司, [2018.04 .ver2.2](https://blog.csdn.net/dingpan119/article/details/107848199?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_baidulandingword-4&spm=1001.2101.3001.4242)

[2] 领航者ZYNQ之FPGA开发指南V1.3 正点原子, 第十四章 UART串口通信实验

[3] Raspberry Pi – Configuration - UART configuration, <https://www.raspberrypi.orgdocumentationconfigurationuart.md>

[4] CSDN博客-树莓派 4 UART多串口配置通信, TEDxPY, <https://blog.csdn.net/weixin_40796925/article/details/107907991?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_title-5&spm=1001.2101.3001.4242>

[5] CSDN博客-树莓派4B串口通信, dingpan119, <https://blog.csdn.net/dingpan119/article/details/107848199?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_baidulandingword-4&spm=1001.2101.3001.4242>

## 附 录1（源程序，加注释）

module top\_file(

//基本。

input clk,

input rst,

input stop\_key,

input up,

input down,

input left,

input right,

//备忘

input noteORdate,

//时钟

input dateORtime,

//串口

input uart\_rxd,

input matchOnline,

//加速

input speedUP,

//显示

output [7:0] seg\_left,

output [7:0] seg\_right,

output [7:0] an,

//leds

output noteORdate\_led,

output dateORtime\_led,

output matchOnline\_led,

output [3:0] led\_cnt,

output speedUP\_led,

output back,

output [3:0] tixingdeng

);

wire [31:0] note\_date;

wire [31:0] disp\_date;

wire [31:0] disp\_time;

wire [23:0] part\_time;

reg [31:0] disp\_data;

wire [7:0]seg\_data;

wire clk\_out;

wire clk\_btn;

wire [4:0]btn\_out;

reg stop\_info;

reg [2:0]pos\_info;

assign seg\_left = seg\_data;

assign seg\_right = seg\_data;

//串口同步数据

wire [31:0] datedata\_UART;

wire [23:0] timedata\_UART;

//leds

assign noteORdate\_led = noteORdate;

assign dateORtime\_led = dateORtime;

assign matchOnline\_led = matchOnline;

assign speedUP\_led = speedUP;

wire c\_time;//时分秒模块进位信号

time\_div u\_time\_div(

.clk\_in(clk),

.clk\_out(clk\_out),

.clk\_btn(clk\_btn)

);

key\_debounce u\_key\_debounce(

.clk\_btn(clk\_btn),

.rst(rst), //reset

.stop\_key(stop\_key),

.up(up),

.down(down),

.left(left),

.right(right),

.btn\_out(btn\_out)

);

wire [2:0] pos\_info\_main;

wire stop\_info\_main;

main\_counter u\_main\_counter(

.ini\_clk(clk),

.clk(c\_time),

.arstn(rst),

.btns(btn\_out),

.load(matchOnline&&(led\_cnt==4'd0)),

.load\_date(datedata\_UART),

.disp\_date(disp\_date),

.stop\_info(stop\_info\_main),

.pos\_info(pos\_info\_main)

);

hms\_counter hms\_cnt0(

.clk(clk),

.arstn(rst),

.stop(1'b0),

.load(matchOnline&&(led\_cnt==4'd0)),

.data(timedata\_UART),

.speedUP(speedUP),

.disp\_time(part\_time),

.c(c\_time)

);

wire [2:0]pos\_info\_note;

wire stop\_info\_note;

note\_module u\_note\_module(

.note\_key(noteORdate),

.ini\_clk(clk),

.clk(1'b1),

.arstn(rst),

.btns(btn\_out),

.load(0),

.load\_date(32'b0),

.disp\_date(note\_date),

.stop\_info(stop\_info\_note),

.pos\_info(pos\_info\_note)

);

assign disp\_time = {8'hFF, part\_time};//时间高位补非BCD码

//切换日历/时钟功能, dateORtime=1显示时间，否则显示日期

always @ (\*) begin

if (noteORdate) begin

disp\_data = note\_date;

pos\_info = pos\_info\_note;

stop\_info = stop\_info\_note;

end

else if (dateORtime) begin

disp\_data = disp\_date;

pos\_info = pos\_info\_main;

stop\_info = stop\_info\_main;

end

else begin

disp\_data = disp\_time;

pos\_info = pos\_info\_main;

stop\_info = stop\_info\_main;

end

end

smg u\_smg(

.clk(clk),

.rst(rst),

.stop\_state(stop\_info),

.btn\_pos(pos\_info),

.true\_date(disp\_data),

.point(8'b00010100),

.seg(seg\_data),

.an(an)

);

//串口接收器

main\_UART Receivor0 (

.sys\_clk(clk),

.sys\_rst\_n(rst),

.uart\_rxd(uart\_rxd),

.datedata\_out(datedata\_UART),

.timedata\_out(timedata\_UART),

.led\_cnt(led\_cnt)

);

assign back = uart\_rxd;

reg[3:0] regtixingdeng;

always@(\*) begin

if (disp\_data == note\_date)

regtixingdeng=4'b1111;

else

regtixingdeng = 4'b0000;

end

assign tixingdeng = regtixingdeng;

// test\_led2bcd u\_test\_led2bcd(

// .code(seg\_data[6:0]),

// .bcd\_code(test\_code)

// );

endmodule

module main\_counter(

input ini\_clk, //100MHz的原始时钟。

input clk,

input arstn,

input[4:0] btns, //按键，由key\_debounce输出得到。

input load, //装载初值的使能信号。

input [31:0] load\_date, //装的初值。

output [31:0] disp\_date, //显示的值。最后输出的是bcd?。

output stop\_info, //停止信息。

//设计之初是想搞停止的，但感觉对日历来说停止实在是没啥用，这个后来就变成了“时间调整的使能位”。

output [2:0]pos\_info //按键所在位置。（调年，还是月，还是日？）

);

wire [13:0] year;

wire [3:0] month;

wire [4:0] day;

wire c\_D, c\_M, c\_Y; //进位信息。

wire [1:0] exp\_code; //大小月，闰年二月信息。

reg stop\_state; //就是上面的stop\_info

reg[2:0] dmy\_btnena; //就是上面的pos\_info，但有点不一样。

//“时间调整”使能位。

always@(posedge btns[4] or negedge arstn) begin

if (!arstn)

stop\_state <=0;

else

stop\_state <= ~stop\_state;

end

assign stop\_info = stop\_state;

//打拍。

reg btn0\_d0,btn0\_d1;

always@(posedge ini\_clk or negedge arstn) begin

if (!arstn) begin

btn0\_d0 <= 0;

btn0\_d1 <= 0;

end

else begin

btn0\_d0 <= btns[0];

btn0\_d1 <= btn0\_d0;

end

end

reg btn1\_d0,btn1\_d1;

always@(posedge ini\_clk or negedge arstn) begin

if (!arstn) begin

btn1\_d0 <= 0;

btn1\_d1 <= 0;

end

else begin

btn1\_d0 <= btns[1];

btn1\_d1 <= btn1\_d0;

end

end

//刚开始不是ini\_clk检测的。是直接用posedge btn[0]这个检测，但是信号非常非常不稳定。

always @(posedge ini\_clk or negedge arstn) begin

if (~arstn)

dmy\_btnena <= 3'b001;

else if (stop\_info) begin

if (btn0\_d0&(!btn0\_d1)) begin

dmy\_btnena <= {dmy\_btnena[0],dmy\_btnena[2:1]}; //循环右移。

end

else if (btn1\_d0&(!btn1\_d1)) begin

dmy\_btnena <= {dmy\_btnena[1:0],dmy\_btnena[2]}; //循环左移。

end

end

else begin

dmy\_btnena <= dmy\_btnena;

end

end

assign pos\_info = {dmy\_btnena[2]&stop\_info,dmy\_btnena[1]&stop\_info,dmy\_btnena[0]&stop\_info};

Dcounter dayCounter (.ini\_clk(ini\_clk), .clk(clk), .btns(btns[3:2]),

.btn\_ena(pos\_info[0]), .arstn(arstn),

.ena(1'b1), .load(load), .data(load\_date[7:0]),

.exp\_code(exp\_code), .c\_D(c\_D), .day\_out(day));

Mcounter monthCounter (.ini\_clk(ini\_clk), .clk(clk), .btns(btns[3:2]),

.btn\_ena(pos\_info[1]), .arstn(arstn),

.ena(c\_D), .load(load), .data(load\_date[15:8]),

.c\_M(c\_M), .month\_out(month));

Ycounter yearCounter (.ini\_clk(ini\_clk), .clk(clk), .btns(btns[3:2]),

.btn\_ena(pos\_info[2]), .arstn(arstn),

.ena(c\_D&c\_M), .load(load), .data(load\_date[31:16]),

.c\_Y(c\_Y), .year\_out(year));

ExceptFor except0 (.year(year), .month(month), .exp\_code(exp\_code));

//转化位BCD码。

wire [3:0] YearHH;

wire [3:0] YearHL;

wire [3:0] YearLH;

wire [3:0] YearLL;

wire [3:0] MonthH;

wire [3:0] MonthL;

wire [3:0] DayH;

wire [3:0] DayL;

assign YearHH = year/1000;

assign YearHL = year/100 - YearHH\*10;

assign YearLH = year/10 - YearHL\*10 - YearHH\*100;

assign YearLL = year%10;

assign MonthH = month/10;

assign MonthL = month%10;

assign DayH = day/10;

assign DayL = day%10;

assign disp\_date = {

YearHH, YearHL,

YearLH, YearLL,

MonthH, MonthL,

DayH , DayL

};

endmodule

module Dcounter(

input ini\_clk, //原始的100MHz的时钟。

input clk, //分频后的1Hz的时钟。

input[1:0] btns, //按键，负责控制上和下。两个就够了。

input btn\_ena, //按键使能位，只有当stop\_info == 1 时才使能。

input arstn, //复位

input ena, //使能。现在用不到。（在月和年模块里就是前一个模块的进位信号）

input load, //load==1,赋初值。

input[7:0] data, //赋的初值

input[1:0] exp\_code,//输入月份信息（大小月份，闰年二月啥的）

output c\_D, //进位信号，送给Mcounter的ena位。

output[4:0] day\_out //输出的天。

);

reg [4:0] state; //2^5=32>31

wire [4:0] day;

wire up,down;

assign up = btn\_ena & btns[1];

assign down = btn\_ena & btns[0];

assign day = state + 1'b1; //没有第0天。

// 打拍。

reg up\_d0, up\_d1;

always @ (posedge ini\_clk or negedge arstn) begin

if (~arstn) begin

up\_d0 <= 1'b0;

up\_d1 <= 1'b0;

end

else begin

up\_d0 <= up;

up\_d1 <= up\_d0;

end

end

reg down\_d0, down\_d1;

always @ (posedge ini\_clk or negedge arstn) begin

if (~arstn) begin

down\_d0 <= 1'b0;

down\_d1 <= 1'b0;

end

else begin

down\_d0 <= down;

down\_d1 <= down\_d0;

end

end

reg clk\_d0,clk\_d1;

always @(posedge ini\_clk or negedge arstn) begin

if (~arstn) begin

clk\_d0 <= 0;

clk\_d1 <= 0;

end

else begin

clk\_d0 <= clk;

clk\_d1 <= clk\_d0;

end

end

//计时。

always @ (posedge ini\_clk or negedge arstn) begin

if (~arstn) begin

state <= 5'd0; //复位就给他1949.10.01。新中国成立啦！

end

else if (load) begin

state <= 10\*data[7:4]+data[3:0]-1;

end

else if (clk\_d0&(~clk\_d1)) begin

if (((state>=5'd27)&&(exp\_code==2'b00))||

((state>=5'd28)&&(exp\_code==2'b01))||

((state>=5'd29)&&(exp\_code==2'b10))||

((state>=5'd30)&&(exp\_code==2'b11)))

state <= 5'd0;

else

state <= state + 1'b1;

end

else if (up\_d0&(~up\_d1)) begin

if (((state>=5'd27)&&(exp\_code==2'b00))||

((state>=5'd28)&&(exp\_code==2'b01))||

((state>=5'd29)&&(exp\_code==2'b10))||

((state>=5'd30)&&(exp\_code==2'b11)))

state <= 5'd0;

else

state <= state + 1'b1;

end

else if (down\_d0&(~down\_d1)) begin

if (state == 5'd0)

state <= 5'd0;

else

state <= state - 1'b1;

end

else

state <= state;

end

// 原本想异步实现加减天数的，但是发现：

// 会导致：1.在不同always块中对同一个变量赋值，反应到电路里就是同一时刻不同信号给同一个输入端。

// 2. 这种异步的方式就算“行为仿真”成功了，综合出来的电路也会出问题。根本跑不了代码。

// 3. 信号及其不稳定。这里怀疑是因为通过LUT和FF出来的信号就是不稳定的，不能放在always块里做边缘检测量。人工打拍后才行。

// always @(posedge (up) or posedge (down)) begin

// if (up == 1) begin

// if (((state>=5'd27)&&(exp\_code==2'b00))||

// ((state>=5'd28)&&(exp\_code==2'b01))||

// ((state>=5'd29)&&(exp\_code==2'b10))||

// ((state>=5'd30)&&(exp\_code==2'b11)))

// state <= 5'd0;

// else

// state <= state + 1;

// end

// else if (down ==1) begin

// if (state == 0)

// state <= 5'd0;

// else

// state <= state-1;

// end

// end

assign day\_out = day;

assign c\_D = (((day==5'd28)&&(exp\_code==2'b00))||

((day==5'd29)&&(exp\_code==2'b01))||

((day==5'd30)&&(exp\_code==2'b10))||

((day==5'd31)&&(exp\_code==2'b11)));

endmodule

module Mcounter(

// 和Dcounter的输入几乎一个道理。

input ini\_clk,

input clk,

input[1:0] btns,

input btn\_ena,

input arstn,

input ena, //Dcounter的进位信号。

input load,

input[7:0] data,

output c\_M,

output[3:0] month\_out

);

reg [3:0] state; //2^4=16>12

wire [3:0] month;

assign month = state + 1'b1;

// reg cc;

// assign c\_M = cc;

wire up,down;

assign up = btn\_ena & btns[1];

assign down = btn\_ena & btns[0];

// 打拍。

reg up\_d0, up\_d1;

always @ (posedge ini\_clk or negedge arstn) begin

if (~arstn) begin

up\_d0 <= 1'b0;

up\_d1 <= 1'b0;

end

else begin

up\_d0 <= up;

up\_d1 <= up\_d0;

end

end

reg down\_d0,down\_d1;

always @ (posedge ini\_clk or negedge arstn) begin

if (~arstn) begin

down\_d0 <= 1'b0;

down\_d1 <= 1'b0;

end

else begin

down\_d0 <= down;

down\_d1 <= down\_d0;

end

end

reg clk\_d0,clk\_d1;

always @(posedge ini\_clk or negedge arstn) begin

if (~arstn) begin

clk\_d0 <= 0;

clk\_d1 <= 0;

end

else begin

clk\_d0 <= clk;

clk\_d1 <= clk\_d0;

end

end

always @ (posedge ini\_clk or negedge arstn or posedge load) begin

if (~arstn) begin

state <= 4'd9; //复位就给他1949.10.01。新中国成立啦！

// cc <= 1'b0;

end

else if (load) begin

state <= 10\*data[7:4]+data[3:0]-1;

end

else if (ena&clk\_d0&(~clk\_d1)) begin

if (state >= 4'd11)

state <= 4'd0;

else

state <= state + 1'b1;

end

else if (up\_d0&(~up\_d1)) begin

if (state >= 4'd11)

state <= 4'd0;

else

state <= state + 1'b1;

end

else if (down\_d0&(~down\_d1)) begin

if (state == 4'd0)

state <= 4'd0;

else

state <= state-1'b1;

end

else

state <= state;

end

//月份的数值。

assign month\_out = month;

//月份的进位信号。

assign c\_M = (month==4'd12);

endmodule

module Ycounter(

input ini\_clk,

input clk,

input[1:0] btns,

input btn\_ena,

input arstn,

input ena,

input load,

input[15:0] data,

output c\_Y,

output[13:0] year\_out

);

reg [13:0] state; //2^5=32>31

wire [13:0] year;

assign year = state + 1'b1; //呃，年份没有0年。（其实是有的，甚至有负的哈哈哈）

wire up,down;

assign up = btn\_ena & btns[1];

assign down = btn\_ena & btns[0];

//打拍，虽然不知道为啥打了就会很稳定。打就对了。

reg up\_d0, up\_d1;

always @ (posedge ini\_clk or negedge arstn) begin

if (~arstn) begin

up\_d0 <= 1'b0;

up\_d1 <= 1'b0;

end

else begin

up\_d0 <= up;

up\_d1 <= up\_d0;

end

end

reg down\_d0,down\_d1;

always @ (posedge ini\_clk or negedge arstn) begin

if (!arstn) begin

down\_d0 <= 1'b0;

down\_d1 <= 1'b0;

end

else begin

down\_d0 <= down;

down\_d1 <= down\_d0;

end

end

reg clk\_d0,clk\_d1;

always @(posedge ini\_clk or negedge arstn) begin

if (~arstn) begin

clk\_d0 <= 0;

clk\_d1 <= 0;

end

else begin

clk\_d0 <= clk;

clk\_d1 <= clk\_d0;

end

end

always @ (posedge ini\_clk or negedge arstn or posedge load) begin

if (~arstn) begin

state <= 14'd1948; //复位就给他1949.10.01。新中国成立啦！

end

else if (load) begin

state <= 1000\*data[15:12]+100\*data[11:8]+10\*data[7:4]+data[3:0]-1;

end

else if (ena&clk\_d0&(~clk\_d1)) begin

if (state>=14'd9999)

state <= 14'd0;

else

state <= state + 1'b1;

end

else if (up\_d0&(~up\_d1)) begin

if (state >= 14'd9999)

state <= 14'b0;

else

state <= state + 1'b1;

end

else if (down&(~down\_d1)) begin

if (state == 14'd0000)

state <= 14'd0;

else

state <= state - 1'b1;

end

else

state <= state;

end

//年的数值。

assign year\_out = year;

//年的进位信号。（其实没啥用，已经进无可进了）

assign c\_Y = (year==14'd9999);

endmodule

module ExceptFor(

input[13:0] year,

input[3:0] month,

output reg [1:0] exp\_code

// output reg LeapYear, BigMonth, Feb

);

// 特殊情况判断表

// exp\_code=00 平年2月-28天

// exp\_code=01 闰年2月-29天

// exp\_code=10 小月-30天

// exp\_code=11 大月-31天

reg LeapYear; //是否闰年

reg BigMonth, Feb; //是否大月，是否二月

//是否闰年

always @ (\*) begin

if (year%4==0) begin

if (year%100!=0)

LeapYear = 1'b1;

else if (year%400==0) begin

if (year%3200!=0)

LeapYear = 1'b1;

else

LeapYear = 1'b0;

end

end

else

LeapYear = 1'b0;

end

//是否大月

always @ (\*) begin

if (

(month==4'd1)||

(month==4'd3)||

(month==4'd5)||

(month==4'd7)||

(month==4'd8)||

(month==4'd10)||

(month==4'd12)

)

BigMonth = 1'b1;

else

BigMonth = 1'b0;

end

//是否二月

always @ (\*) begin

Feb = (month==4'd2);

end

// assign Feb = (month==4'd2);

//特殊情况码

always @ (\*) begin

if (Feb) begin

if (~LeapYear)

exp\_code = 2'b00;

else

exp\_code = 2'b01;

end

else begin

if (~BigMonth)

exp\_code = 2'b10;

else

exp\_code = 2'b11;

end

end

endmodule

module hms\_counter(

input clk,

input arstn,

input stop,

input load,

input [23:0] data,

input speedUP,

output[23:0] disp\_time,

output c

);

reg [19:0] divclk\_cnt;

reg divclk;

reg[3:0] sec\_l;

reg[3:0] sec\_h;

reg[3:0] min\_l;

reg[3:0] min\_h;

reg[3:0] hour\_l;

reg[3:0] hour\_h;

//调试用加速模块。

parameter MAXCNT = 1000000;

parameter MINCNT = 1156;

reg [19:0] time\_cnt;

always @ (\*) begin

if (speedUP)

time\_cnt = MINCNT;

else

time\_cnt = MAXCNT;

end

assign disp\_time = {hour\_h, hour\_l, min\_h, min\_l, sec\_h, sec\_l};

//鏉堟挸鍙嗛弮鍫曟寭10MHZ閿涘瞼娲伴惃鍕闁??100Hz閿涘矂?鏍у帥鐎圭偟骞?1M閸掑棝??

always @ (posedge clk or negedge arstn) begin

if(!arstn) begin

divclk <= 1'b0;

divclk\_cnt <= 20'd0;

end

//閸嬭埖鏆熼崚鍡??鎴礄1M閸掑棝?鎴礆閿涘奔绗傞梽?1M/2-1

else if (divclk\_cnt==time\_cnt/2-1'b1) begin

divclk <= ~divclk;

divclk\_cnt <= 20'd0;

end

else begin

divclk\_cnt <= divclk\_cnt + 1'b1;

end

end

//閺嗗倸浠犻崝鐔诲厴闁??

reg state;

reg stop\_now, stop\_last;

always @ (posedge clk or negedge arstn) begin

stop\_now <= stop;

stop\_last <= stop\_now;

if (!arstn)

state <= 1'b0;

else if ({stop\_last, stop\_now}==2'b01)

state <= state^stop;

else

state <= state;

end

always @ (posedge divclk or negedge arstn or posedge load) begin

if (!arstn)

{hour\_h, hour\_l,

min\_h, min\_l,

sec\_h, sec\_l} <= 24'h15\_00\_00;

else if (load)

{hour\_h, hour\_l,

min\_h, min\_l,

sec\_h, sec\_l} <= data;

else if (!state) begin

sec\_l <= sec\_l + 1'b1;

if (sec\_l==4'd9) begin

sec\_l <= 4'd0;

sec\_h <= sec\_h + 1'b1;

if (sec\_h==4'd5) begin

sec\_h <= 4'd0;

min\_l <= min\_l + 1'b1;

//濞夈劍鍓版稉鈧崚鍡涙寭60缁夋帪绱濋崚?59缁夋帞娈戞稉瀣╃娑??濞岃儻?浣界箻娴??

if (min\_l==4'd9) begin

min\_l <= 4'd0;

min\_h <= min\_h + 1'b1;

if (min\_h==4'd5) begin

min\_h <= 4'd0;

hour\_l <= hour\_l + 1'b1;

//閸氬瞼鎮婇敍灞肩鐏忓繑妞?23閸掑棝鎸?

if (hour\_l==4'd3) begin

hour\_l <= 4'd0;

hour\_h <= hour\_h + 1'b1;

if(hour\_h==4'd2) begin

hour\_h <= 4'd0;

end

end

end

end

end

end

end

else

{hour\_h, hour\_l,min\_h, min\_l,sec\_h, sec\_l} <= {hour\_h, hour\_l,min\_h, min\_l,sec\_h, sec\_l};

end

assign c = ({hour\_h, hour\_l,min\_h, min\_l, sec\_h, sec\_l}==24'h235959);

endmodule

module main\_UART(

input sys\_clk, //外部100M时钟

input sys\_rst\_n, //外部复位信号，低有效

input uart\_rxd, //UART接收端口

output[31:0] datedata\_out,

output[23:0] timedata\_out,

output [3:0] led\_cnt

);

//parameter define

parameter CLK\_FREQ = 100\_000000; //定义系统时钟频率

parameter UART\_BPS = 115200; //定义串口波特率

localparam BPS\_CNT = CLK\_FREQ/UART\_BPS; //为得到指定波特率，

//需要对系统时钟计数BPS\_CNT次

//wire define

wire uart\_recv\_done; //UART接收完成

wire [7:0] uart\_recv\_data; //UART接收数据

//串口接收模块

uart\_recv #(

.CLK\_FREQ (CLK\_FREQ), //设置系统时钟频率

.UART\_BPS (UART\_BPS)) //设置串口接收波特率

u\_uart\_recv(

.sys\_clk (sys\_clk),

.sys\_rst\_n (sys\_rst\_n),

.uart\_rxd (uart\_rxd),

.uart\_done (uart\_recv\_done),

.uart\_data (uart\_recv\_data)

);

reg [3:0] cnt1; //2^4=16

reg [31:0] datedata;

reg [23:0] timedata;

wire StartFlag;

//打拍处理

reg uart\_recv\_done\_d0, uart\_recv\_done\_d1;

//捕获接收端口上升沿(起始位)，得到一个时钟周期的脉冲信号

assign StartFlag = (uart\_recv\_done\_d0) & (~uart\_recv\_done\_d1);

//延迟两个时钟周期

always @(posedge sys\_clk or negedge sys\_rst\_n) begin

if (!sys\_rst\_n) begin

uart\_recv\_done\_d0 <= 1'b0;

uart\_recv\_done\_d1 <= 1'b0;

end

else begin

uart\_recv\_done\_d0 <= uart\_recv\_done;

uart\_recv\_done\_d1 <= uart\_recv\_done\_d0;

end

end

//接收16帧数据，首帧起始位，第1~8帧日期，第9帧校验位，第11~15帧时间

always @ (posedge sys\_clk or negedge sys\_rst\_n) begin

if (!sys\_rst\_n)

cnt1 <= 4'd0;

else if (StartFlag) begin

case(cnt1)

//启始标志

4'd0: begin

if (uart\_recv\_data==8'b0111\_1110)

cnt1 <= 4'd1;

end

//寄存日期数据

4'd8 : begin

datedata[3:0] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

4'd7 : begin

datedata[7:4] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

4'd6 : begin

datedata[11:8] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

4'd5 : begin

datedata[15:12] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

4'd4 : begin

datedata[19:16] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

4'd3 : begin

datedata[23:20] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

4'd2 : begin

datedata[27:24] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

4'd1 : begin

datedata[31:28] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

//检验位

4'd9 : begin

if (uart\_recv\_data!=8'b1000\_1000)

cnt1 <= 4'd0;

else

cnt1 <= cnt1 + 1;

end

//寄存时间数据

4'd15 : begin

timedata[3:0] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

4'd14 : begin

timedata[7:4] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

4'd13 : begin

timedata[11:8] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

4'd12 : begin

timedata[15:12] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

4'd11 : begin

timedata[19:16] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

//结束位

4'd10 : begin

timedata[23:20] <= uart\_recv\_data[3:0];

cnt1 <= cnt1 + 1;

end

endcase

end

end

assign datedata\_out = datedata;

assign timedata\_out = timedata;

assign led\_cnt = cnt1;

endmodule

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Copyright (c)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//原子哥在线教学平台：www.yuanzige.com

//技术支持：www.openedv.com

//淘宝店铺：http://openedv.taobao.com

//关注微信公众平台微信号："正点原子"，免费获取ZYNQ & FPGA & STM32 & LINUX资料。

//版权所有，盗版必究。

//Copyright(C) 正点原子 2018-2028

//All rights reserved

//----------------------------------------------------------------------------------------

// File name: uart\_recv

// Last modified Date: 2019/10/9 9:56:36

// Last Version: V1.1

// Descriptions: UART串口接收模块

//----------------------------------------------------------------------------------------

// Created by: 正点原子

// Created date: 2019/10/9 9:56:36

// Version: V1.0

// Descriptions: The original version

//

//----------------------------------------------------------------------------------------

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//连copyright都有，专业！

module uart\_recv(

input sys\_clk, //系统时钟

input sys\_rst\_n, //系统复位，低电平有效

input uart\_rxd, //UART接收端口

output reg uart\_done, //接收一帧数据完成标志。

output reg rx\_flag, //接收过程标志信号

output reg [ 3:0] rx\_cnt, //接收数据计数

output reg [ 7:0] rxdata,

output reg [7:0] uart\_data //接收的数??

);

//parameter define

// parameter CLK\_FREQ = 50000000; //系统时钟频率

parameter CLK\_FREQ = 100\_000000; //系统时钟频率。

parameter UART\_BPS = 115200; //串口波特率。

localparam BPS\_CNT = CLK\_FREQ/UART\_BPS; //为得到指定串口的波特率。

//需要对系统时钟计数BPS\_CNT??

//reg define

reg uart\_rxd\_d0;

reg uart\_rxd\_d1;

reg [15:0] clk\_cnt; //系统时钟计数??

//wire define

wire start\_flag;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\* main code

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//捕获接收端口下降沿(起始位)，得到一个时钟周期的脉冲信号

assign start\_flag = uart\_rxd\_d1 & (~uart\_rxd\_d0);

//对UART接收端口的数据延迟两个时钟周期

always @(posedge sys\_clk or negedge sys\_rst\_n) begin

if (!sys\_rst\_n) begin

uart\_rxd\_d0 <= 1'b0;

uart\_rxd\_d1 <= 1'b0;

end

else begin

uart\_rxd\_d0 <= uart\_rxd;

uart\_rxd\_d1 <= uart\_rxd\_d0;

end

end

//当脉冲信号start\_flag到达时，进入接收过程

always @(posedge sys\_clk or negedge sys\_rst\_n) begin

if (!sys\_rst\_n)

rx\_flag <= 1'b0;

else begin

if(start\_flag) //检测到起始位。

rx\_flag <= 1'b1; //进入接收过程，标志位rx\_flag拉高

//计数到停止位中间时，停止接收过程

else if((rx\_cnt == 4'd10) && (clk\_cnt == BPS\_CNT/2))

rx\_flag <= 1'b0; //接收过程结束，标志位rx\_flag拉低

else

rx\_flag <= rx\_flag;

end

end

//进入接收过程后，启动系统时钟计数??

always @(posedge sys\_clk or negedge sys\_rst\_n) begin

if (!sys\_rst\_n)

clk\_cnt <= 16'd0;

else if ( rx\_flag ) begin //处于接收过程

if (clk\_cnt < BPS\_CNT - 1)

clk\_cnt <= clk\_cnt + 1'b1;

else

clk\_cnt <= 16'd0; //对系统时钟计数达一个波特率周期后清??

end

else

clk\_cnt <= 16'd0; //接收过程结束，计数器清零

end

//进入接收过程后，启动接收数据计数??

always @(posedge sys\_clk or negedge sys\_rst\_n) begin

if (!sys\_rst\_n)

rx\_cnt <= 4'd0;

else if ( rx\_flag ) begin //处于接收过程

if (clk\_cnt == BPS\_CNT - 1) //对系统时钟计数达一个波特率周期

rx\_cnt <= rx\_cnt + 1'b1; //此时接收数据计数器加1

else

rx\_cnt <= rx\_cnt;

end

else

rx\_cnt <= 4'd0; //接收过程结束，计数器清零

end

//根据接收数据计数器来寄存uart接收端口数据

always @(posedge sys\_clk or negedge sys\_rst\_n) begin

if ( !sys\_rst\_n)

rxdata <= 8'd0;

else if(rx\_flag) //系统处于接收过程

if (clk\_cnt == BPS\_CNT/2) begin //判断系统时钟计数器计数到数据位中??

case ( rx\_cnt )

4'd1 : rxdata[0] <= uart\_rxd\_d1; //寄存数据位最低位

4'd2 : rxdata[1] <= uart\_rxd\_d1;

4'd3 : rxdata[2] <= uart\_rxd\_d1;

4'd4 : rxdata[3] <= uart\_rxd\_d1;

4'd5 : rxdata[4] <= uart\_rxd\_d1;

4'd6 : rxdata[5] <= uart\_rxd\_d1;

4'd7 : rxdata[6] <= uart\_rxd\_d1;

4'd8 : rxdata[7] <= uart\_rxd\_d1; //寄存数据位最高位

default:;

endcase

end

else

rxdata <= rxdata;

else

rxdata <= 8'd0;

end

//数据接收完毕后给出标志信号并寄存输出接收到的数据

always @(posedge sys\_clk or negedge sys\_rst\_n) begin

if (!sys\_rst\_n) begin

uart\_data <= 8'd0;

uart\_done <= 1'b0;

end

else if(rx\_cnt == 4'd9) begin //接收数据计数器计数到停止位时

if (((^rxdata)^uart\_rxd\_d1)==1'b0) //如果满足偶校验位

uart\_data <= rxdata; //寄存输出接收到的数据

else

uart\_data <= uart\_data;

end

else if(rx\_cnt == 4'd10)

uart\_done <= 1'b1; //并将接收完成标志位拉??

else begin

uart\_data <= 8'd0;

uart\_done <= 1'b0;

end

end

endmodule

// 直接从main\_counter里复制过来修改。

module note\_module(

input note\_key, //从R2接过来的note信号。

input ini\_clk, //100MHz的原始时钟。

input clk, //1Hz的分频后的时钟。

input arstn,

input[4:0] btns, //按键，由key\_debounce输出得到。

input load, //装载初值的使能信号。

input [31:0] load\_date, //装的初值。

output [31:0] disp\_date, //显示的值。最后输出的是bcd?。

output stop\_info, //停止信息。

//设计之初是想搞停止的，但感觉对日历来说停止实在是没啥用，这个后来就变成了“时间调整的使能位”。

output [2:0]pos\_info //按键所在位置。（调年，还是月，还是日？）

);

wire [13:0] year;

wire [3:0] month;

wire [4:0] day;

wire c\_D, c\_M, c\_Y; //进位信息。

wire [1:0] exp\_code; //大小月，闰年二月信息。

reg stop\_state; //就是上面的stop\_info

reg[2:0] dmy\_btnena; //就是上面的pos\_info，但有点不一样。

//“时间调整”使能位。

// always@(posedge btns[4] or negedge arstn) begin

// if (!arstn)

// stop\_state <=0;

// else

// stop\_state <= ~stop\_state;

// end

// assign stop\_info = stop\_state;

assign stop\_info = note\_key;

//打拍。

reg btn0\_d0,btn0\_d1;

always@(posedge ini\_clk or negedge arstn) begin

if (!arstn) begin

btn0\_d0 <= 0;

btn0\_d1 <= 0;

end

else begin

btn0\_d0 <= btns[0];

btn0\_d1 <= btn0\_d0;

end

end

reg btn1\_d0,btn1\_d1;

always@(posedge ini\_clk or negedge arstn) begin

if (!arstn) begin

btn1\_d0 <= 0;

btn1\_d1 <= 0;

end

else begin

btn1\_d0 <= btns[1];

btn1\_d1 <= btn1\_d0;

end

end

//刚开始不是ini\_clk检测的。是直接用posedge btn[0]这个检测，但是信号非常非常不稳定。

always @(posedge ini\_clk or negedge arstn) begin

if (~arstn)

dmy\_btnena <= 3'b001;

else if (stop\_info) begin

if (btn0\_d0&(!btn0\_d1)) begin

dmy\_btnena <= {dmy\_btnena[0],dmy\_btnena[2:1]}; //循环右移。

end

else if (btn1\_d0&(!btn1\_d1)) begin

dmy\_btnena <= {dmy\_btnena[1:0],dmy\_btnena[2]}; //循环左移。

end

end

else begin

dmy\_btnena <= dmy\_btnena;

end

end

assign pos\_info = {dmy\_btnena[2]&stop\_info,dmy\_btnena[1]&stop\_info,dmy\_btnena[0]&stop\_info};

Dcounter dayCounter (.ini\_clk(ini\_clk), .clk(clk), .btns(btns[3:2]),

.btn\_ena(pos\_info[0]), .arstn(arstn),

.ena(1'b1), .load(load), .data(load\_date[7:0]),

.exp\_code(exp\_code), .c\_D(c\_D), .day\_out(day));

Mcounter monthCounter (.ini\_clk(ini\_clk), .clk(clk), .btns(btns[3:2]),

.btn\_ena(pos\_info[1]), .arstn(arstn),

.ena(c\_D), .load(load), .data(load\_date[15:8]),

.c\_M(c\_M), .month\_out(month));

Ycounter yearCounter (.ini\_clk(ini\_clk), .clk(clk), .btns(btns[3:2]),

.btn\_ena(pos\_info[2]), .arstn(arstn),

.ena(c\_D&c\_M), .load(load), .data(load\_date[31:16]),

.c\_Y(c\_Y), .year\_out(year));

ExceptFor except0 (.year(year), .month(month), .exp\_code(exp\_code));

//转化位BCD码。

wire [3:0] YearHH;

wire [3:0] YearHL;

wire [3:0] YearLH;

wire [3:0] YearLL;

wire [3:0] MonthH;

wire [3:0] MonthL;

wire [3:0] DayH;

wire [3:0] DayL;

assign YearHH = year/1000;

assign YearHL = year/100 - YearHH\*10;

assign YearLH = year/10 - YearHL\*10 - YearHH\*100;

assign YearLL = year%10;

assign MonthH = month/10;

assign MonthL = month%10;

assign DayH = day/10;

assign DayL = day%10;

assign disp\_date = {

YearHH, YearHL,

YearLH, YearLL,

MonthH, MonthL,

DayH , DayL

};

endmodule

//消抖模块，这个没啥好说的。

//中上下左右五个按键。用btn\_out[4:0]输出。

module key\_debounce(

input clk\_btn,

input rst,

input stop\_key,

input up,

input down,

input left,

input right,

output [4:0] btn\_out

);

reg btn0;

reg btn1;

reg btn2;

always@ (posedge clk\_btn or negedge rst)

begin

if (!rst) begin

btn0<=1'b0;

btn1<=1'b0;

btn2<=1'b0;

end

else begin

btn0 <= stop\_key | up | right | down | left;

btn1 <= btn0;

btn2 <= btn1;

end

end

assign btn\_out = {stop\_key&btn0&btn1&btn2,up&btn0&btn1&btn2,down&btn0&btn1&btn2,left&btn0&btn1&btn2,right&btn0&btn1&btn2};

endmodule

//时钟分频模块。

module time\_div(clk\_in,clk\_out,clk\_btn);

//按键的消抖用的是20ms的。

input clk\_in;

output reg clk\_out=0; //正常的1Hz的。

output reg clk\_btn=0; //给按键消抖用的。

reg[26:0] clk\_div\_cnt=0;

reg[19:0] clk\_div\_cnt2=0;

always @ (posedge clk\_in)

begin

if (clk\_div\_cnt==27'd49999999)

begin

clk\_out <= ~clk\_out;

clk\_div\_cnt<=0;

end

else

clk\_div\_cnt<=clk\_div\_cnt+1;

end

always @(posedge clk\_in)

begin

if (clk\_div\_cnt2 == 20'd999999) begin

clk\_btn<=~clk\_btn;

clk\_div\_cnt2 <= 0;

end

else

clk\_div\_cnt2 <= clk\_div\_cnt2 + 1;

end

endmodule

//24h的。

//module time\_div(clk\_in,clk\_out);

// input clk\_in;

// output reg clk\_out=0;

// reg[26:0] clk\_div\_cnt=0;

// reg[16:0] temp\_div=0;

// always @ (posedge clk\_in)

// begin

// if (clk\_div\_cnt==99999999)

// begin

// temp\_div = temp\_div + 1;

// clk\_div\_cnt=0;

// end

// else begin

// clk\_div\_cnt=clk\_div\_cnt+1;

// end

// if (temp\_div == 86400/2)

// begin

// clk\_out = !clk\_out;

// clk\_div\_cnt = 0;

// temp\_div = 0;

// end

// end

// endmodule