

**EDA(II) 实验报告**

**­——多功能数字钟设计**

|  |  |  |  |
| --- | --- | --- | --- |
| **作 者:** | 施尘玥 | **学 号：** | 9151040G0507 |
| **学 院:** | 电光学院 | | |
| **专 业:** | 通信工程 | | |
| **题 目:** | EDA(II)实验报告 | | |
|  | ——多功能数字钟设计 | | |

**指导者：** 蒋立平

2017 年 12 月

**目录**

[一、设计要求说明 5](#_Toc500227270)

[1、题目简介 5](#_Toc500227271)

[2、设计要求（包括基本和提高） 5](#_Toc500227272)

[二、方案论证 6](#_Toc500227273)

[三、数字钟各子模块的设计原理 6](#_Toc500227274)

[1、分频模块 6](#_Toc500227275)

[1.1 分频实现原理 7](#_Toc500227276)

[1.2 VHDL代码 7](#_Toc500227277)

[1.3 封装与仿真波形 8](#_Toc500227278)

[2、总计时模块 9](#_Toc500227279)

[2.1消颤模块 10](#_Toc500227280)

[2.2 校星期、时、分模块 11](#_Toc500227281)

[2.3 保持模块 12](#_Toc500227284)

[2.4 模7、模24、模60模块 13](#_Toc500227285)

[2.4.1 实现原理 13](#_Toc500227286)

[2.4.2 模24VHDL代码 13](#_Toc500227287)

[2.5 报时模块 15](#_Toc500227288)

[2.6 清零功能 16](#_Toc500227289)

[3.闹钟模块 17](#_Toc500227290)

[3.1 消颤模块 18](#_Toc500227291)

[3.2 模24、模60模块 18](#_Toc500227292)

[3.3 比较模块 18](#_Toc500227293)

[3.3.1 实现原理 18](#_Toc500227294)

[3.3.2 比较模块VHDL代码 18](#_Toc500227295)

[3.4 选择输出模块 19](#_Toc500227296)

[4.秒表模块 20](#_Toc500227297)

[5.音乐模块 21](#_Toc500227298)

[5.1 实现原理 21](#_Toc500227299)

[5.2 音乐模块VHDL代码 22](#_Toc500227300)

[6.显示模块 27](#_Toc500227301)

[6.1 模8模块 27](#_Toc500227302)

[6.2 八选一模块 28](#_Toc500227303)

[6.2.1 实现原理 28](#_Toc500227304)

[6.2.2 八选一模块VHDL代码 28](#_Toc500227305)

[6.3三八译码器模块 29](#_Toc500227306)

[6.3.1 实现原理 29](#_Toc500227307)

[6.3.2 三八译码器模块VHDL代码 29](#_Toc500227308)

[6.4七段译码器模块模块 30](#_Toc500227309)

[6.3.1 实现原理 30](#_Toc500227310)

[6.3.2七段译码器模块VHDL代码 30](#_Toc500227311)

[7.复用模块 31](#_Toc500227312)

[7.1 校星期、时、分开关复用模块 31](#_Toc500227313)

[7.2 闹钟开关复用模块 32](#_Toc500227314)

[7.3 显示复用模块 33](#_Toc500227315)

[8.多功能数字钟总电路图 34](#_Toc500227316)

[四、调试，仿真，编程下载 35](#_Toc500227317)

[五、实验过程中遇到的问题及解决方法 35](#_Toc500227318)

[六、实验感想 36](#_Toc500227319)

[七、参考文献 37](#_Toc500227320)

# 摘要

数字钟是一种用数字电路技术实现时、分、秒计时的钟表。与机械钟相比具有更高的准确性和直观性，具有更长的使用寿命，已得到广泛的使用。数字钟的设计方法有许多种，例如可用中小规模集成电路组成电子钟，也可以利用专用的电子钟芯片配以显示电路及其所需要的外围电路组成电子钟，还可以利用单片机来实现电子钟等等。

本文基于 QuartusⅡ在 FPGA 上用 VHDL 语言编程实现数字钟，其具有 24 小时计时，对时钟的星期、时、分、秒进行校对，时钟保持，时钟清零，整点报时，设置闹钟，秒表计时等功能。

本文采用自顶向下的模块式设计方法，将整个电路分为分频模块，校星期、时、分模块，清零保持模块，计时模块，闹钟模块，秒表模块，音乐模块，显示模块。在设计时对每个模块所需要的功能进行仿真验证或直接下载到实验平台验证。在验证了功能的正确性后，将各个模块相互连接组成一个总的电路。最后对总的电路分配好引脚，下载到实验板上，在满足基本功能的情况下进行了创新，达到了不错的效果。

# Abstract

Digital clock is a time clock with digital circuit technology. Compared with mechanical clock, it has higher accuracy and intuitiveness, has longer service life and has been widely used. Design method of digital clock has many kinds, such as the available composition of small and medium scale integrated circuit, electronic clock, also can use special electronic clock chip with the peripheral circuit of electronic clock display circuit and its need, also can take advantage of the single chip microcomputer to realize electronic clock and so on.

Based on Quartus Ⅱ on FPGA with VHDL language programming to realize digital clock, it has 24 hours timer, week of clock, proofreading, minutes and seconds, when the clock keep, clock reset, the hour, set the alarm clock, stopwatch, and other functions.

This article adopts the modular design of the top-down method, dividing the whole circuit is divided into modules, the school week, when, module, reset module, timing module, clock module, a stopwatch module, music module, display module. The functions required for each module are simulated or downloaded directly to the experimental platform verification during the design. After verifying the correctness of the function, each module is connected to form a total circuit. Finally, the overall circuit distribution is good for the pin, download to the experiment board, and the innovation is carried out in the case of the basic function, which achieves a good effect.

# 一、设计要求说明

## 1、题目简介

设计一个多功能数字钟，可以完成00:00:00到23:59:59的计时功能，并在控制电路的作用下具有保持、清零、快速校时、快速校分、整点报时等功能。

设计可增加闹钟、12小时计时、秒表、日历、音乐报时、发光管闪烁效果等功能。

## 2、设计要求（包括基本和提高）

1.能进行正常的时、分、秒计时功能；

2.分别由七个数码管显示时分秒以及星期，从左到右分别为星期、时十位、时个位、分十位、分个位、秒十位、秒个位；星期显示为1、2、3、4、5、6、7分别表示周一到周日；

3.开关的分配

* K1是计时电路的使能开关（K1=0正常工作，K1=1保持不变）；
* K2是计时电路的清零开关（K2=0正常工作，K2=1清零）；
* K3、K4是多功能数字钟的校时复用开关（K3=0且K4=0时正常工作，K3=1且K4=0时可以快速校分，K3=0且K4=1时可以快速校时，K3=1且K4=1时可以快速校星期）；
* K5、K6是闹钟电路的复用开关（K5=0且K6=0时数码管显示正常计时，闹铃为歌曲《小星星》；K5=1时且K6=0时数码管显示闹钟设置界面，此时可以用开关K3,K4设置闹钟时间，闹铃为歌曲《小星星》；K5=0时且K6=1时数码管显示正常计时界面，闹铃为歌曲《十年》；K5=1时且K6=1时关闭闹钟）；
* K7是秒表电路的显示开关（K7=0时数码管显示正常计时，K7=1时数码管显示秒表计时界面，此时可以使用K1对秒表进行计时）；
* K8是秒表电路的清零开关（K8=0秒表正常计时，K8=1秒表电路清零）

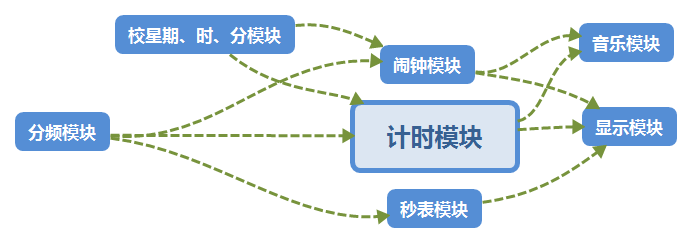
4.整点报时功能（当时钟计到59’53”时开始报时，在59’53”, 59’55”,59’57”时报时频率为500Hz，59’59”时报时频率为1KHz）；

5.界面人性化，星期与时分秒间用短横线间隔显示；

6.增加闹钟设置功能，可选择播放音乐《小星星》或《十年》，并可手动关闭闹钟；

7.增加秒表功能，可以准确计时。

# 二、方案论证



按照功能，可以将时钟分为以下模块：分频模块、校星期、时、分模块、清零保持模块、计时模块、闹钟模块、秒表模块、音乐模块、显示模块。其中，消颤功能属于计时模块与秒表模块自身的功能。根据各个部分的功能要求，各个大模块分成小的基本模块，以实现相应的功，模块间联系如图。

# 三、数字钟各子模块的设计原理

## 1、分频模块

分频的基本设计思想是将实验箱提供的48MHZ的高频信号通过分频器变成所需的1Hz、2Hz、100Hz、500Hz和1kHz的信号，1000Hz 用于报时最后一声，500Hz 用于报时的前几声，100Hz用于秒表计时，2Hz 用于快速校分，1Hz 用于正常计时模块。

### 1.1 分频实现原理

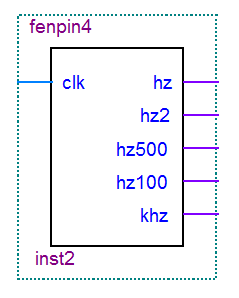
对时钟信号进行上升沿检测，并记录检测的数（从 0 开始计数）。如果是 *N* 分频，每当计数到 时将信号翻转一次，将信号输出。

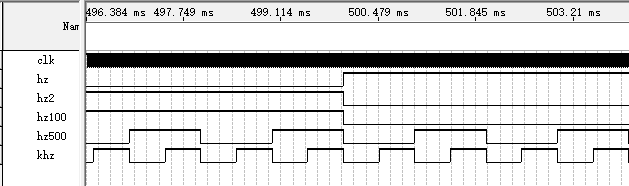
### 1.2 VHDL代码

|  |
| --- |
| **LIBRARY** IEEE**;**  **USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**  **ENTITY** fenpin4 **IS**  **PORT(**clk**:IN** STD\_LOGIC**;**  hz**:buffer** STD\_LOGIC**;**  hz2**:buffer** STD\_LOGIC**;**  hz500**:buffer** STD\_LOGIC**;**  hz100**:buffer** STD\_LOGIC**;**  khz**:buffer** STD\_LOGIC**);**  **END** fenpin4**;**  **ARCHITECTURE** beh **OF** fenpin4 **IS**  **SIGNAL** count**:**integer **range** 0 **to** 24000000**;**  **SIGNAL** count1**:**integer **range** 0 **to** 24000000**;**  **SIGNAL** count2**:**integer **range** 0 **to** 24000000**;**  **SIGNAL** count3**:**integer **range** 0 **to** 24000000**;**  **SIGNAL** count4**:**integer **range** 0 **to** 24000000**;**  **BEGIN**  **PROCESS(**clk**)**  **BEGIN**  **IF(**clk**=**'1'**)THEN**  count**<=**count**+**1**;**  count1**<=**count1**+**1**;**  count2**<=**count2**+**1**;**  count3**<=**count3**+**1**;**  count4**<=**count4**+**1**;**  **IF(**count**=**23999999**)THEN**  count**<=**0**;**  hz**<=**NOT hz**;**  **END** **IF;**  **IF(**count1**=**11999999**)THEN**  count1**<=**0**;**  hz2**<=**NOT hz2**;**  **END** **IF;**  **IF(**count2**=**23999**)THEN**  count2**<=**0**;**  khz**<=**NOT khz**;**  **END** **IF;**  **IF(**count4**=**239999**)THEN**  count4**<=**0**;**  hz100**<=**NOT hz100**;**  **END** **IF;**  **IF(**count3**=**47999**)THEN**  count3**<=**0**;**  hz500**<=**NOT hz500**;**  **END** **IF;**  **END** **IF;**  **END** **PROCESS;**  **END** beh**;** |

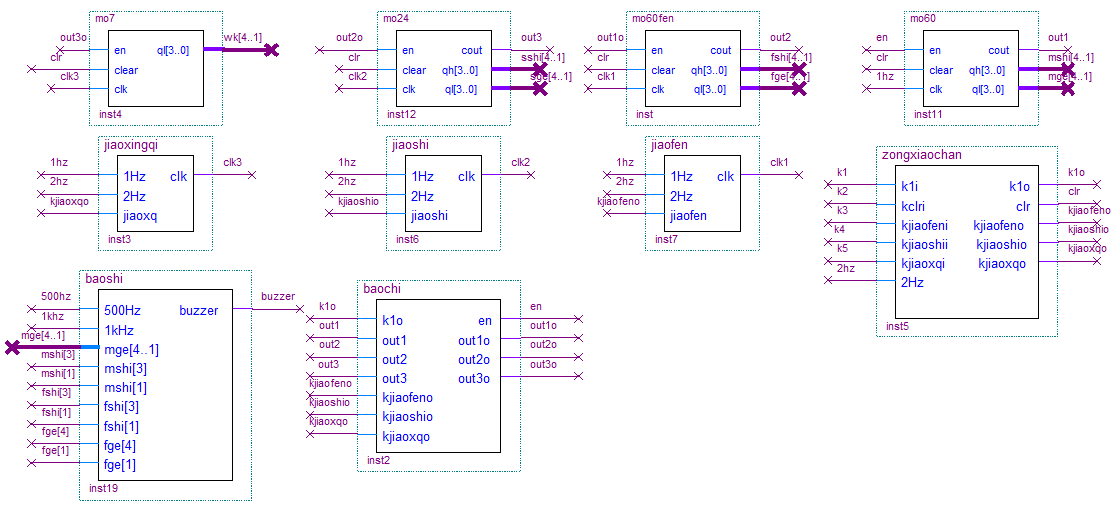
### 1.3 封装与仿真波形

将分频模块封装成如下所示：



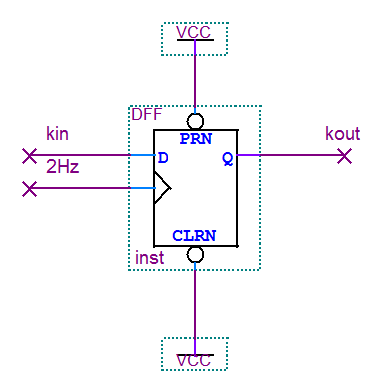
分频模块仿真波形如下所示：

## 2、总计时模块

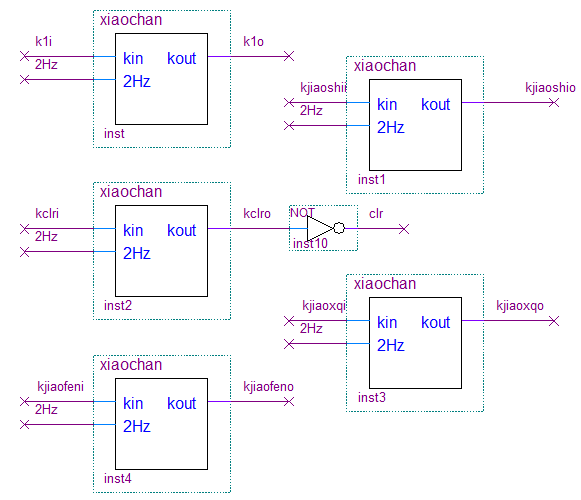
该模块由消颤模块，校星期、时、分模块，保持模块，模7、模24、模60模块、报时模块组成。总原理图如下所示：

### 2.1消颤模块

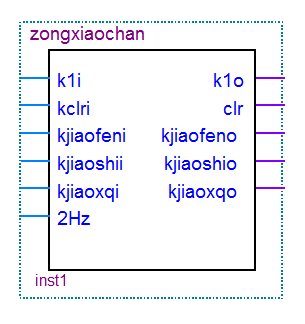
由于开关的接通和关闭有可能产生毛刺，出现竞争冒险现象，所以要对所有的开关进行消颤处理，本次实验需要对K1-K5五个开关（其中K5后来转为K3与K4进行复用产生的开关）进行消颤。具体是通过一个D触发器对开关信号进行延时，从而得到稳定可靠的开关信号，具体电路如下所示：



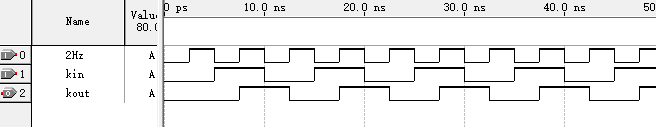
D触发器的时钟接2Hz信号，两个控制端都接高电平，这样时钟边沿到来时输出跟随输入，相当于对输入信号做了一个延时处理。



封装之后的消颤模块为：

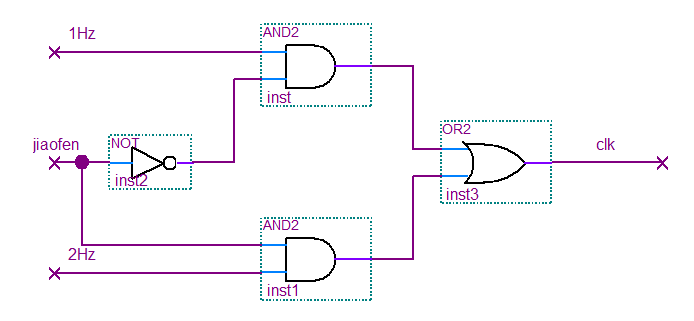


其仿真波形为：

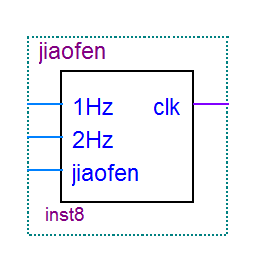


### 2.2 校星期、时、分模块

校分电路由二输入与门、非门和或门这三种逻辑门构成。可知：当K3=0（校分开关）时时钟端clk输入1Hz的信号即时钟正常计数，当K3=1时时钟端clk输入2Hz的信号即进行快速校分。具体电路如下图所示：



校星期电路、校时电路与校分电路原理类似，可得校分模块封装如下：



通过对仿真波形的分析来验证以上控制电路的正确性：

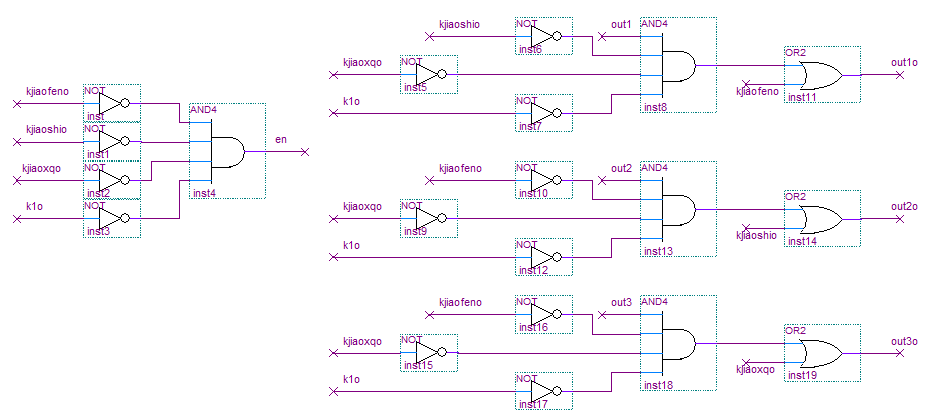
K=0时，正常计数；

### 较分波形图0

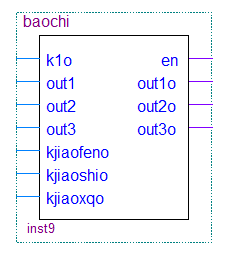
k=1时，快速较分。

## 较分波形图

### 2.3 保持模块

保持功能是通过逻辑门电路控制秒、分、时以及星期计数器的使能端实现的。正常情况下，K1=0，此时通过一个非门接到使能端则计数器正常计数，而当K1=1时，使能端变成0，此时计数器停止计数，从而实现计时保持功能。

可得保持模块封装如下：



### 2.4 模7、模24、模60模块

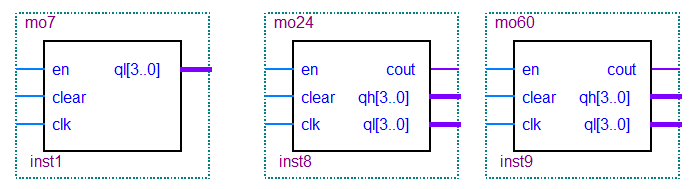
#### 2.4.1 实现原理

利用IF语句，对计数的条件进行判断，例如模7，在计数到0111时跳转到0001，即可实现数字1到7的循环计数。

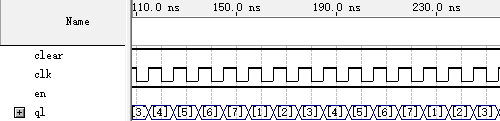
#### 2.4.2 模24VHDL代码

|  |
| --- |
| **library** ieee**;**  **use** ieee**.**std\_logic\_1164**.all;**  **use** ieee**.**std\_logic\_unsigned**.all;**  **ENTITY** mo24 **IS**  **PORT**  **(** en **:IN** std\_logic**;**  clear**:IN** std\_logic**;**  clk **:IN** std\_logic**;**  cout **:out** std\_logic**;**  qh **:buffer** std\_logic\_vector**(**3 **downto** 0**);**  ql **:buffer** std\_logic\_vector**(**3 **downto** 0**)**  **);**  **END** mo24**;**  **ARCHITECTURE** beh **OF** mo24 **IS**  **BEGIN**  cout**<=**'1'**when(**qh**=**"0010"and ql**=**"0011"and en**=**'1'**)else**'0'**;**  **PROCESS(**clk**,**clear**)**  **BEGIN**  **IF(**clear**=**'0'**)THEN**  qh**<=**"0000"**;**  ql**<=**"0000"**;**  **ELSIF(**clk'**EVENT** AND clk**=**'1'**)THEN**  **if(**en**=**'1'**)then**  **if((**ql**=**3 and qh**=**2**)** or ql**=**9**)** **then**  ql**<=**"0000"**;**  **if(**qh**=**2**)then**  qh**<=**"0000"**;**  **else**  qh**<=**qh**+**1**;**  **end** **if;**  **else**  ql**<=**ql**+**1**;**  **end** **if;**  **end** **if;**  **END** **IF;**  **END** **PROCESS;**  **END** beh**;** |

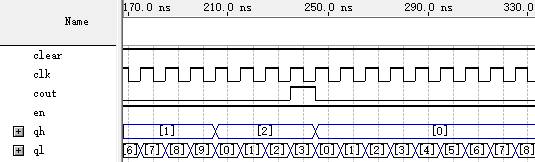
* 模7和模60的VHDL代码与模24的类似，仅在IF的判断语句处有差别。因此本文省略模7和模60的VHDL代码。

模7、模24、模60的封装如下：

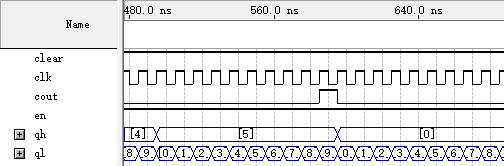
模7仿真波形为：



模24仿真波形为：



模60仿真波形为：



### 2.5 报时模块

整点报时电路要求完成数字钟在每 59 分 53 秒、59 分 55 秒、59 分 57 秒时低音报时，在 59 分 59 秒时高音报时，这就要求要使用 1000Hz、500Hz 两种信号分别驱动。在电路设计中，将响铃条件分别用与门相连，当检测到特定时间序列时则响起铃声。

下面列出各个时刻的各位输出状态，以分析报时条件，

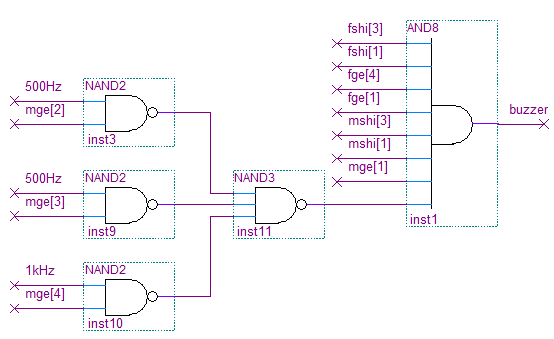
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 时刻 | 分十位 | 分个位 | 秒十位 | 秒个位 | 频率 |
| 59分53秒 | 0101 | 1001 | 0101 | 0011 | 500Hz |
| 59分55秒 | 0101 | 1001 | 0101 | 0101 | 500Hz |
| 59分57秒 | 0101 | 1001 | 0101 | 0111 | 500Hz |
| 59分59秒 | 0101 | 1001 | 0101 | 1001 | 1kHz |

通过观察发现,蜂鸣器要响至少必须满足59分50秒，且秒个位必须为奇数，因此有满足此条件的表达式：

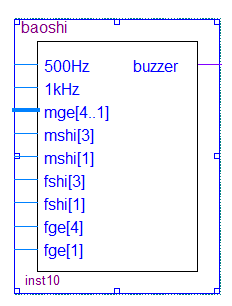
在满足F1的基础上，还要满足mge[2]、mge[3]、mge[4]中至少有一个不为0蜂鸣器才会响，而且只有在mge[4]不为0时蜂鸣器发出1000Hz响的声，其余3种情况都为500Hz的响声，所以又有表达式F2必须满足：

所以蜂鸣器输入端的逻辑表达式为*。*

其电路图为：



封装为：

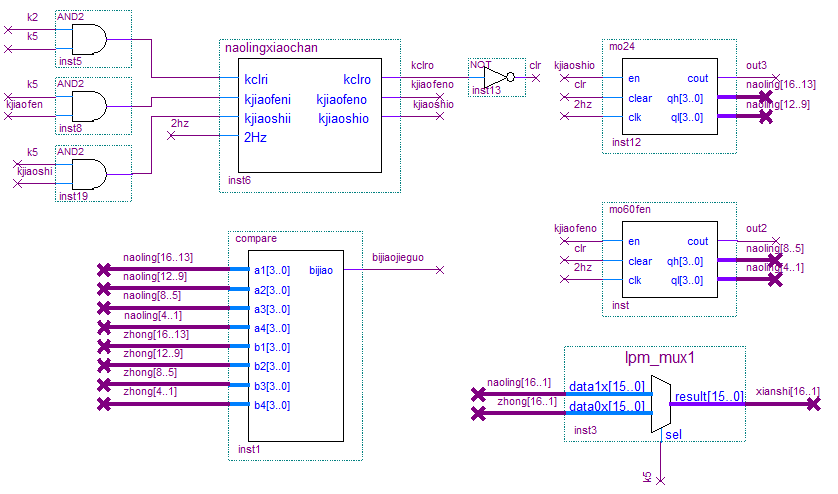


### 2.6 清零功能

清零功能是通过控制各位计数器清零端的电平高低来实现的。只需使清零开关按下时各计数器的清零端均接入低电平，而清零开关断开时各清零端均接入高电平即可。

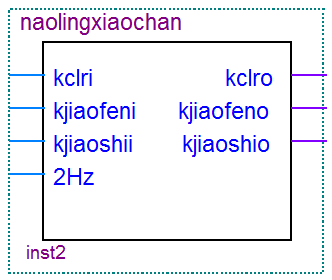
## 3.闹钟模块

该模块由消颤模块，模24、模60模块、比较模块、选择输出模块组成。总原理图如下所示：



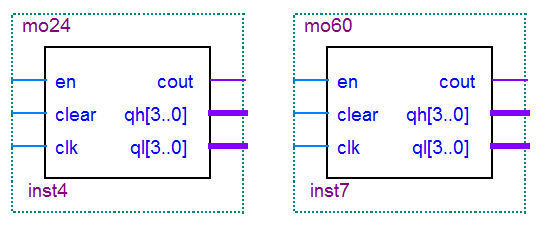
### 3.1 消颤模块

闹钟模块的消颤原理与总计时模块的相同，在此不做赘述，仅给出封装图：



### 3.2 模24、模60模块

闹钟模块的模24、60与总计时模块的相同，在此不做赘述，仅给出封装图：



### 3.3 比较模块

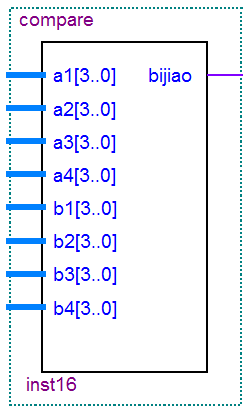
#### 3.3.1 实现原理

将总计时模块的时分位与闹钟模块的时分位进行比较，如果相等就输出1，不等就输出0。

#### 3.3.2 比较模块VHDL代码

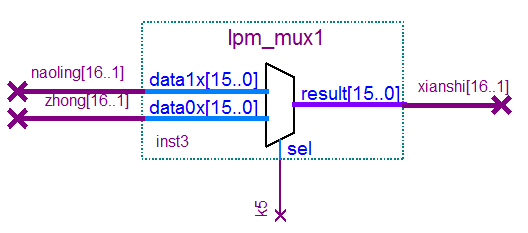
|  |
| --- |
| **LIBRARY** IEEE**;**  **USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**  **ENTITY** compare **is**  **PORT(**a1**,**a2**,**a3**,**a4**,**b1**,**b2**,**b3**,**b4**:in** std\_logic\_vector**(**3 **downto** 0**);**  bijiao**:buffer** bit**);**  **end** compare**;**  **architecture** beh **of** compare **is**  **signal** a**,**b**:**std\_logic\_vector**(**15 **downto** 0**);**  **begin**  a**<=**a4**&**a3**&**a2**&**a1**;**  b**<=**b4**&**b3**&**b2**&**b1**;**  **process(**a**,**b**)**  **begin**  **if** **(**a**=**b**)then**  bijiao**<=**'1'**;**  **else** bijiao**<=**'0'**;**  **end** **if;**  **end** **process;**  **end** beh**;** |

比较模块封装如下：



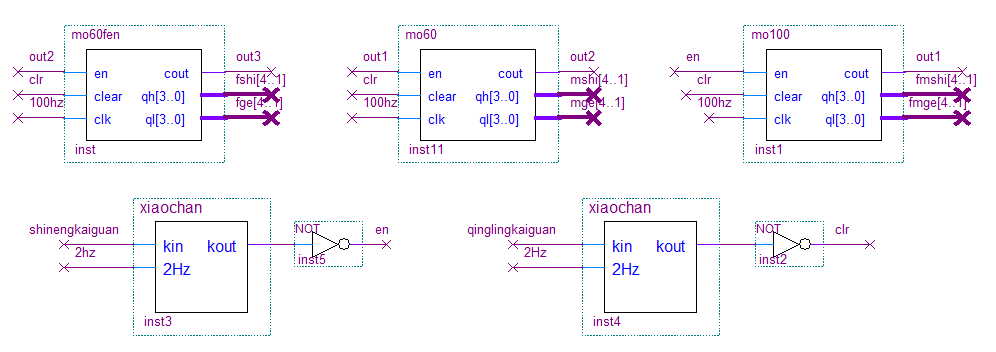
### 3.4 选择输出模块

通过调用lpm\_mux模块，对闹钟模块的时分位和总计时模块的时分位进行数据选择，用K5控制数码管的显示。



## 4.秒表模块

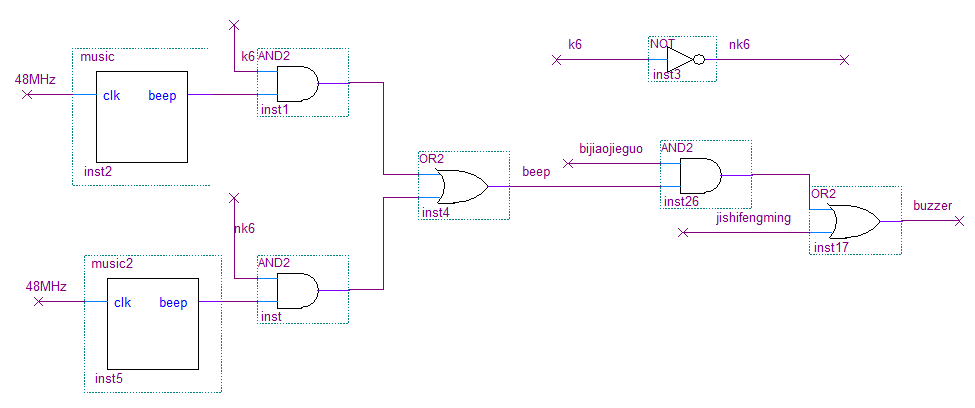
该模块由消颤模块，模60、100模块组成。总原理图如下所示：



消颤模块、模60、100模块的原理均与总计时模块的原理相似，在此不进行赘述。

## 5.音乐模块

该模块由两个音乐模块组成。总原理图如下所示：



### 5.1 实现原理

* 音调

音乐的产生主要是通过分频输出高低不同的脉冲信号来控制蜂鸣器发音，乐曲的十二平均律规定：每2个八度音（如简谱中的中音1与高音1之间的频率相差一倍。在2个八度音之间，又可分为12个半音，每2个半音的频率比为。另外，简谱中的低音6的频率为440Hz，音符7到1之间、3到4之间为半音，其余为全音。由此可计算出简谱中从低音1至高音7之间每个音符的频率，如下表所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 音符 | 频率Hz | 音符 | 频率Hz | 音符 | 频率Hz |
| 低1Do | 262 | 中1Do | 523 | 高1Do | 1047 |
| 低2Re | 294 | 中2Re | 587 | 高2Re | 1175 |
| 低3Mi | 330 | 中3Mi | 659 | 高3Mi | 1319 |
| 低4Fa | 349 | 中4Fa | 698 | 高4Fa | 1397 |
| 低5So | 392 | 中5So | 784 | 高5So | 1568 |
| 低6La | 440 | 中6La | 880 | 高6La | 1760 |
| 低7Si | 494 | 中7Si | 988 | 高7Si | 1967 |

基于上表对 48MHz 进行分频。根据分频模块的原理，分*N*频时，计数到，可以计算出音调与计数的关系。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 音符 | 频率Hz | 音符 | 频率Hz | 音符 | 频率Hz |
| 低1Do | 91742 | 中1Do | 45861 | 高1Do | 22932 |
| 低2Re | 81715 | 中2Re | 40864 | 高2Re | 20429 |
| 低3Mi | 72814 | 中3Mi | 36401 | 高3Mi | 18201 |
| 低4Fa | 68727 | 中4Fa | 34358 | 高4Fa | 17183 |
| 低5So | 61223 | 中5So | 30611 | 高5So | 15305 |
| 低6La | 54544 | 中6La | 27271 | 高6La | 13635 |
| 低7Si | 48592 | 中7Si | 24295 | 高7Si | 12148 |

* 节拍

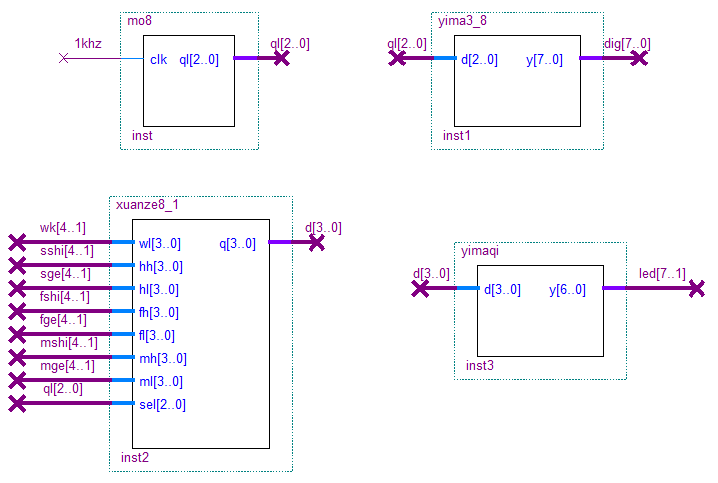
简谱里将音符分为全音符、二分音符、四分音符、十六分音符、三十二分音符等。将四分音符作为一拍，产生一个4Hz 的频率信号，表示一个节拍250ms，用一个变量对整首歌进行计数，这个变量也就代表了时间长度，每加一前进 250ms. 一个音符有几个节拍就连续播放此音符多少次。

### 5.2 音乐模块VHDL代码

|  |
| --- |
| **library** ieee**;**  **use** ieee**.**std\_logic\_1164**.all;**  **ENTITY** music2 **IS**  **PORT(**  clk **:** **in** std\_logic**;**  beep**:** **out** std\_logic  **);**  **end** music2**;**  **architecture** beh **of** music2 **is**  **type** state\_type **is** **(**do\_l**,**re\_l**,**mi\_l**,**fa\_l**,**sol\_l**,**la\_l**,**si\_l**,**do\_m**,**re\_m**,**mi\_m**,**fa\_m**,**sol\_m**,**la\_m**,**si\_m**,**do\_h**,**re\_h**,**mi\_h**,**fa\_h**,**sol\_h**,**la\_h**,**si\_h**,**none**);**  **signal** counter **:** integer **range** 0 **to** 100000 **:=** 0**;**  **signal** count **:** integer **range** 0 **to** 99 **:=** 0**;**  **signal** beep\_reg**:** std\_logic**;**  **signal** clk1khz**,**clk4hz **:** std\_logic**;**  **signal** note **:** state\_type **;**  **begin**  beep **<=** beep\_reg**;**  beep\_pro **:** **process(**clk**)**  **variable** cnt **:** integer **range** 0 **to** 100000 **:=** 0**;**  **begin**  **if** clk'**event** and clk**=**'1' **then**  **if** cnt **<** counter **then**  cnt **:=** cnt **+** 1 **;**  **else**  cnt **:=** 0 **;** beep\_reg **<=** not beep\_reg**;**  **end** **if;**  **end** **if;**  **end** **process** beep\_pro**;**  clk1khz\_pro **:** **process(**clk**)**  **variable** cnt **:** integer **range** 0 **to** 23999**;**  **begin**  **if** clk'**event** and clk**=**'1' **then**  **if** cnt **=** 23999 **then**  cnt **:=** 0 **;** clk1khz **<=** not clk1khz**;**  **else**  cnt **:=** cnt **+** 1**;**  **end** **if;**  **end** **if;**  **end** **process** clk1khz\_pro**;**  clk4hz\_pro **:process(**clk1khz**)**  **variable** cnt **:** integer **range** 0 **to** 124 **:=** 0**;**  **begin**  **if** clk1khz'**event** and clk1khz **=** '1' **then**  **if** cnt **=** 124 **then**  cnt **:=** 0 **;** clk4hz **<=** not clk4hz**;**  **else**  cnt **:=** cnt **+** 1**;**  **end** **if;**  **end** **if;**  **end** **process** clk4hz\_pro**;**  count\_pro **:** **process(**clk4hz**)**  **begin**  **if** clk4hz'**event** and clk4hz **=**'1' **then**  **if** count **<=** 66 **then**  count **<=** count **+** 1**;**  **else**  count **<=** 0**;**  **end** **if;**  **end** **if;**  **end** **process** count\_pro**;**  note\_pro **:** **process(**note**)**  **begin**  **case** note **is**  **when** do\_l **=>** counter **<=** 91742 **;**  **when** re\_l **=>** counter **<=** 81715 **;**  **when** mi\_l **=>** counter **<=** 72814 **;**  **when** fa\_l **=>** counter **<=** 68727 **;**  **when** sol\_l **=>** counter **<=**61223 **;**  **when** la\_l **=>** counter **<=** 54544 **;**  **when** si\_l **=>** counter **<=** 48592 **;**  **when** do\_m **=>** counter **<=** 45861 **;**  **when** re\_m **=>** counter **<=** 40864 **;**  **when** mi\_m **=>** counter **<=** 36401 **;**  **when** fa\_m **=>** counter **<=** 34358 **;**  **when** sol\_m **=>** counter **<=**30611 **;**  **when** la\_m **=>** counter **<=** 27271 **;**  **when** si\_m **=>** counter **<=** 24295 **;**  **when** do\_h **=>** counter **<=** 22932 **;**  **when** re\_h **=>** counter **<=** 20429 **;**  **when** mi\_h **=>** counter **<=** 18201 **;**  **when** fa\_h **=>** counter **<=** 17183 **;**  **when** sol\_h **=>** counter **<=**15305 **;**  **when** la\_h **=>** counter **<=** 13635 **;**  **when** si\_h **=>** counter **<=** 12148 **;**  **when** **others** **=>** counter **<=** 0**;**  **end** **case;**  **end** **process** note\_pro**;**  music\_pro **:** **process(**count**)**  **begin**  **case** count **is** **when** 0 **=>** note **<=** do\_m **;**  **when** 1 **=>** note **<=** do\_m**;**  **when** 2 **=>** note **<=** do\_m**;**  **when** 3 **=>** note **<=** do\_m**;**  **when** 4 **=>** note **<=** sol\_m**;**  **when** 5 **=>** note **<=** sol\_m**;**  **when** 6 **=>** note **<=** sol\_m**;**  **when** 7 **=>** note **<=** sol\_m**;**  **when** 8 **=>** note **<=** la\_m**;**  **when** 9 **=>** note **<=** la\_m**;**  **when** 10 **=>** note **<=** la\_m**;**  **when** 11 **=>** note **<=** la\_m**;**  **when** 12 **=>** note **<=** sol\_m**;**  **when** 13 **=>** note **<=** sol\_m**;**  **when** 14 **=>** note **<=** sol\_m**;**  **when** 15 **=>** note **<=** sol\_m**;**  **when** 16 **=>** note **<=** fa\_m **;**  **when** 17 **=>** note **<=** fa\_m **;**  **when** 18 **=>** note **<=** fa\_m **;**  **when** 19 **=>** note **<=** fa\_m **;**  **when** 20 **=>** note **<=** mi\_m **;**  **when** 21 **=>** note **<=** mi\_m **;**  **when** 22 **=>** note **<=** mi\_m **;**  **when** 23 **=>** note **<=** mi\_m **;**  **when** 24 **=>** note **<=** re\_m**;**  **when** 25 **=>** note **<=** re\_m**;**  **when** 26 **=>** note **<=** re\_m**;**  **when** 27 **=>** note **<=** re\_m**;**  **when** 28 **=>** note **<=** do\_m**;**  **when** 29 **=>** note **<=** do\_m**;**  **when** 30 **=>** note **<=** do\_m**;**  **when** 31 **=>** note **<=** do\_m**;**  **when** 32 **=>** note **<=** sol\_m**;**  **when** 33 **=>** note **<=** sol\_m**;**  **when** 34 **=>** note **<=** sol\_m**;**  **when** 35 **=>** note **<=** sol\_m**;**  **when** 36 **=>** note **<=** fa\_m **;**  **when** 37 **=>** note **<=** fa\_m **;**  **when** 38 **=>** note **<=** fa\_m **;**  **when** 39 **=>** note **<=** fa\_m **;**  **when** 40 **=>** note **<=** mi\_m**;**  **when** 41 **=>** note **<=** mi\_m**;**  **when** 42 **=>** note **<=** mi\_m**;**  **when** 43 **=>** note **<=** mi\_m**;**  **when** 44 **=>** note **<=** re\_m**;**  **when** 45 **=>** note **<=** re\_m**;**  **when** 46 **=>** note **<=** re\_m**;**  **when** 47 **=>** note **<=** re\_m**;**  **when** 48 **=>** note **<=** sol\_m**;**  **when** 49 **=>** note **<=** sol\_m**;**  **when** 50 **=>** note **<=** sol\_m**;**  **when** 51 **=>** note **<=** sol\_m**;**  **when** 52 **=>** note **<=** fa\_m**;**  **when** 53 **=>** note **<=** fa\_m**;**  **when** 54 **=>** note **<=** fa\_m**;**  **when** 55 **=>** note **<=** fa\_m**;**  **when** 56 **=>** note **<=** mi\_m**;**  **when** 57 **=>** note **<=** mi\_m**;**  **when** 58 **=>** note **<=** mi\_m**;**  **when** 59 **=>** note **<=** mi\_m**;**  **when** 60 **=>** note **<=** re\_m**;**  **when** 61 **=>** note **<=** re\_m**;**  **when** 62 **=>** note **<=** re\_m**;**  **when** 63 **=>** note **<=** re\_m**;**  **when** 64 **=>** note **<=** do\_m**;**  **when** 65 **=>** note **<=** do\_m**;**  **when** 66 **=>** note **<=** do\_m**;**  **when** **others** **=>** note **<=** none**;**  **end** **case;**  **end** **process** music\_pro**;**  **end** beh**;** |

## 6.显示模块

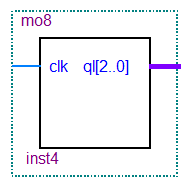
在数码管上的显示为动态扫描，每一个时间只能有一个数码管有效。当扫描的频率比较大时，就可以认为所有的数码管都是亮的。数码管有八个，数字有七个，为了美观，在星期与时分秒中间加入了短横线。显示模块由模8模块、八选一模块、七段译码器模块、三八译码器模块组成。总原理图如下图所示：



### 6.1 模8模块

模8计数器是为了只让一个数码管有效，且作为八选一选择器的输入端，选择输出的数字是多少。

模8的原理和代码与计时电路中模7的原理类似，在此不进行赘述，仅给出封装图：



### 6.2 八选一模块

#### 6.2.1 实现原理

八选一模块用来选择当前有效的数码管应该显示时钟的什么数字，星期、短横线、时十位、时个位、分十位、分个位、十位、秒个位，这八种状态。其中短横线为1111来代替，在后面的七段译码器模块中，将1111译为在数码管中只有 g 段亮，即为短横线。

#### 6.2.2 八选一模块VHDL代码

|  |
| --- |
| **library** ieee**;**  **use** ieee**.**std\_logic\_1164**.all;**  **use** ieee**.**std\_logic\_unsigned**.all;**  **entity** xuanze8\_1 **is**  **port(**wl**,**hh**,**hl**,**fh**,**fl**,**mh**,**ml**:in** std\_logic\_vector**(**3 **downto** 0**);**  sel**:in** std\_logic\_vector**(**2 **downto** 0**);**  q**:out** std\_logic\_vector**(**3 **downto** 0**));**  **end** xuanze8\_1**;**  **architecture** xuanze **of** xuanze8\_1 **is**  **begin**  **process(**sel**,**wl**,**hh**,**hl**,**fh**,**fl**,**mh**,**ml**)**  **begin**  **case** sel **is**  **when**"000"**=>**q**<=**wl**;**  **when**"001"**=>**q**<=**"1111"**;**  **when**"010"**=>**q**<=**hh**;**  **when**"011"**=>**q**<=**hl**;**  **when**"100"**=>**q**<=**fh**;**  **when**"101"**=>**q**<=**fl**;**  **when**"110"**=>**q**<=**mh**;**  **when**"111"**=>**q**<=**ml**;**  **when** **others=>**q**<=**"0000"**;**  **end** **case;**  **end** **process;**  **end** xuanze**;** |

可得八选一模块封装图如下：



### 6.3三八译码器模块

#### 6.3.1 实现原理

该译码器将计数器输出的三位数据译码成八位数据，作为数据分配器使用，控制数码管 的是否有效。

#### 6.3.2 三八译码器模块VHDL代码

|  |
| --- |
| **library** ieee**;**  **use** ieee**.**std\_logic\_1164**.all;**  **use** ieee**.**std\_logic\_unsigned**.all;**  **entity** yima3\_8 **is**  **port(**d**:in** std\_logic\_vector**(**2 **downto** 0**);**  y**:out** std\_logic\_vector**(**7 **downto** 0**));**  **end** yima3\_8**;**  **architecture** yima **of** yima3\_8 **is**  **begin**  **process(**d**)**  **begin**  **case** d **is**  **when**"000"**=>**y**<=**"01111111"**;**  **when**"001"**=>**y**<=**"10111111"**;**  **when**"010"**=>**y**<=**"11011111"**;**  **when**"011"**=>**y**<=**"11101111"**;**  **when**"100"**=>**y**<=**"11110111"**;**  **when**"101"**=>**y**<=**"11111011"**;**  **when**"110"**=>**y**<=**"11111101"**;**  **when**"111"**=>**y**<=**"11111110"**;**  **when** **others=>**y**<=**"11111111"**;**  **end** **case;**  **end** **process;**  **end** yima**;** |

可得三八译码器模块封装如下：



### 6.4七段译码器模块模块

#### 6.4.1 实现原理

该译码器用于将二进制的数字译码为数码管中 a、b、c、d、e、f、g 段的显示情况，能直观地显示出数字来。我们所用的数码管为共阳。

|  |  |
| --- | --- |
| 显示字型 | g,f,e,d,c,b,a |
| 0 | 1 0 0 0 0 0 0 |
| 1 | 1 1 1 1 0 0 1 |
| 2 | 0 1 0 0 1 0 0 |
| 3 | 0 1 1 0 0 0 0 |
| 4 | 0 0 1 0 0 1 0 |
| 5 | 0 0 1 0 0 1 0 |
| 6 | 0 0 0 0 0 1 0 |
| 7 | 1 1 1 1 0 0 0 |
| 8 | 0 0 0 0 0 0 0 |
| 9 | 0 0 1 0 0 0 0 |

#### 6.4.2七段译码器模块VHDL代码

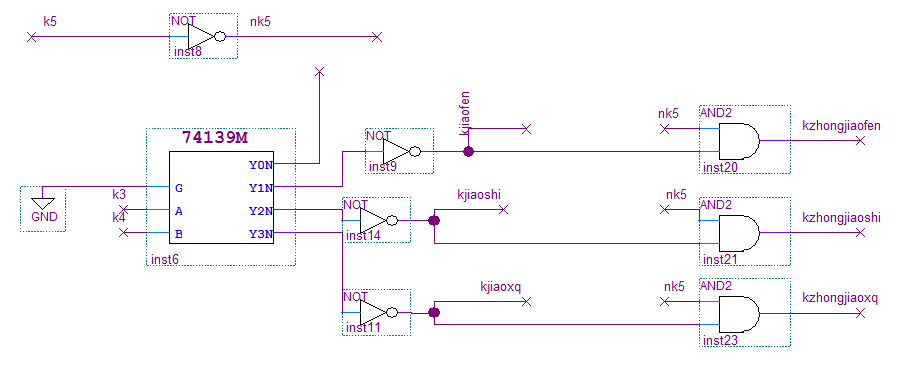
|  |
| --- |
| **library** ieee**;**  **use** ieee**.**std\_logic\_1164**.all;**  **use** ieee**.**std\_logic\_unsigned**.all;**  **entity** yimaqi **is**  **port(**d**:in** std\_logic\_vector**(**3 **downto** 0**);**  y**:out** std\_logic\_vector**(**6 **downto** 0**));**  **end** yimaqi**;**  **architecture** yima **of** yimaqi **is**  **begin**  **process(**d**)**  **begin**  **case** d **is**  **when**"0000"**=>**y**<=**"1000000"**;**  **when**"0001"**=>**y**<=**"1111001"**;**  **when**"0010"**=>**y**<=**"0100100"**;**  **when**"0011"**=>**y**<=**"0110000"**;**  **when**"0100"**=>**y**<=**"0011001"**;**  **when**"0101"**=>**y**<=**"0010010"**;**  **when**"0110"**=>**y**<=**"0000010"**;**  **when**"0111"**=>**y**<=**"1111000"**;**  **when**"1000"**=>**y**<=**"0000000"**;**  **when**"1001"**=>**y**<=**"0010000"**;**  **when**"1111"**=>**y**<=**"0111111"**;**  **when** **others=>**y**<=**"1111111"**;**  **end** **case;**  **end** **process;**  **end** yima**;** |

可得三八译码器模块封装如下：



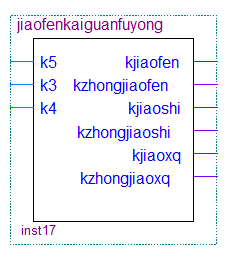
## 7.复用模块

### 7.1 校星期、时、分开关复用模块

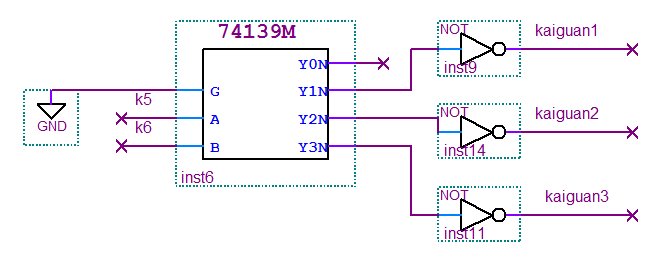


通过2-4译码器，将K3、K4开关进行复用，使K3=0且K4=0时正常工作，K3=1且K4=0时可以快速校分，K3=0且K4=1时可以快速校时，K3=1且K4=1时可以快速校星期，并将译码器结果与K5想与，实现与闹钟模块的复用，K5等于0时校星期、时、分开关对总计时模块有效，当K5等于1时，校星期、时、分开关对闹钟模块有效。

封装如下：



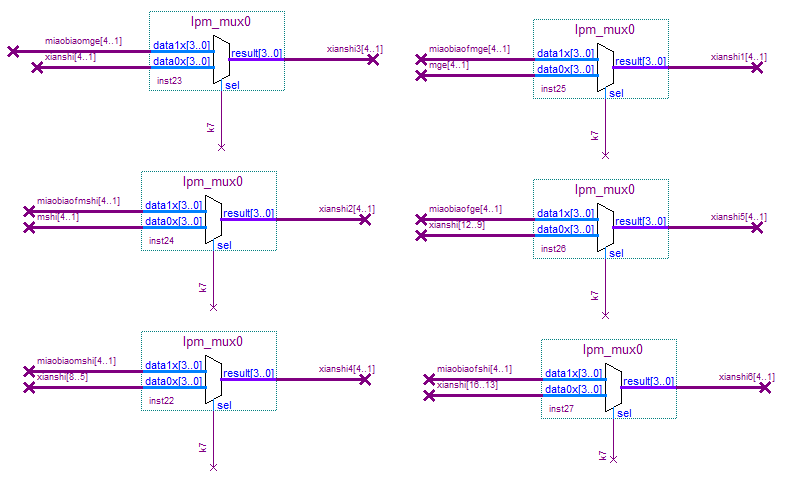
### 7.2 闹钟开关复用模块



通过2-4译码器，将K5、K6开关进行复用，K5=0且K6=0时数码管显示正常计时，闹铃为歌曲《小星星》；K5=1时且K6=0时数码管显示闹钟设置界面，此时可以用开关K3,K4设置闹钟时间，闹铃为歌曲《小星星》；K5=0时且K6=1时数码管显示正常计时界面，闹铃为歌曲《十年》；K5=1时且K6=1时关闭闹钟。封装如下：



### 7.3 显示复用模块

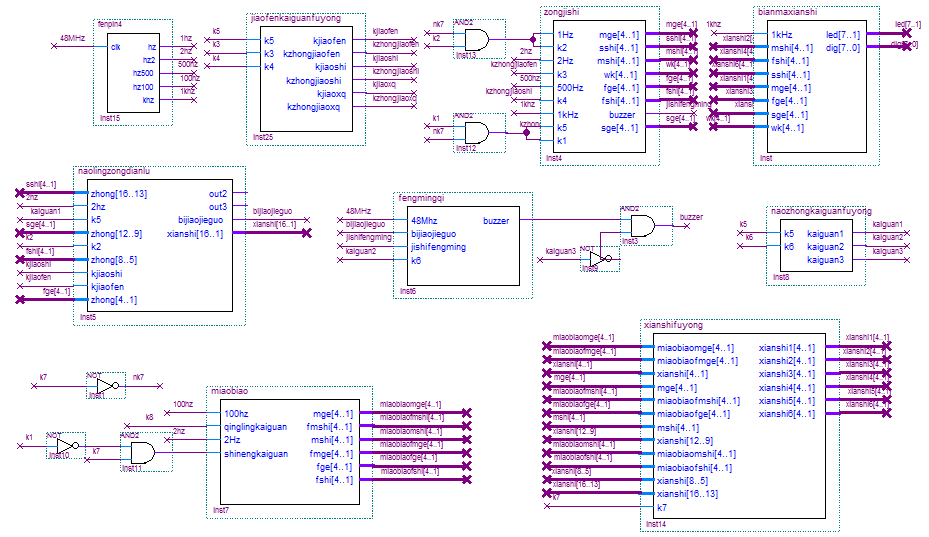


通过六个lpm\_mux模块实现显示复用， K7=1时显示秒表模块，k7=0时显示正常计时模块获闹钟模块，具体显示哪一个在闹钟模块中进行选择。

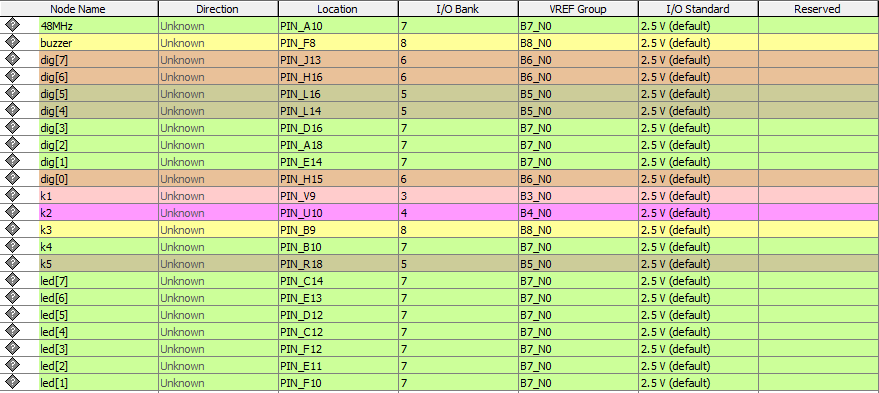
封装如下：



## 8.多功能数字钟总电路图



# 四、调试，仿真，编程下载

将编译好的程序经过管脚设定后即可下载到SmartSOPC实验系统中进行调试和验证。其中管脚设定如下图所示：

# 五、实验过程中遇到的问题及解决方法

总结一下，实验过程中遇到的问题大概有以下几个方面：

1、在仿真分频模块，由于我是写VHDL代码直接将48MHz直接分频为自己想要的频率，在仿真时我将CLK设置为48Mhz，仿真时间设为1s，但发现仿真速度极慢，一开始考虑是不是代码出了问题，后来发现是运算数太多，系统需要处理48M个数据，所以运行慢是正常的，在运行了两三个小时后终于成功仿真出了分频结果。但其实也可以下载到实验系统中观察LED的闪烁情况来进行仿真。

2、然后在实验室下载运行时也出现了让我头疼的情况，我的电路连接到实验室的实验板上时，没有任何的显示，这让我的心情有点烦躁。在冷静思考后，先重新检查了一下我的译码显示电路的设计，多次检查后依旧没有发现问题。后来我又重新检查管脚分配，欣喜地发现有一个管脚分配有问题，然而不幸的是，我改正错误后，还是无法显示。后来我更改了48MHz的管脚并进行了重新编译，就显示出来了。

3、最后我还想加一个秒表模块，秒表模块并不复杂，但我做好之后就是不显示，我认认真真检查了好几遍秒表模块，仿真也没有问题，在这个方面花了大量时间，心里也很苦恼，最后只好求助了学长，学长教会了我一种检查的方法，就是在可能出错的地方接VCC或者接地，然后看系统是否正常，一个一个排除排除错误的地方，最后竟然发现问题出在分频的100hz上，因为只有秒表模块需要100hz，这个频率是最后更改分频模块加的，其中有一个数字写错了，就导致100hz错误，秒表当然就不能正常运行啦。这个问题让我意识到一个小小的错误也很可能会影响整个系统，以后写代码还是要更加认真一点，也感谢学长的帮助，教会了我检查错误的方法。

# 六、实验感想

短暂的五天实验时间，我感触还是很深的。

通过这次实验，我对数字电路的知识有了更深的了解，将课本中的知识应用到了实际中。不过，这次实验更多是让我对 VHDL 语言有了更多的认识，对语法细节也关注了许多。不过对于语言的学习都是一个循序渐进的过程，曾经学过 c++ , matlab ,这对我迅速掌握语言有很大的帮助。

在实验过程中，我真真切切感受到了自顶向下的设计思想。在实验前，首先将整个设计的思路理清，明白要实现哪些功能，需要哪些模块，大体怎样布局。自顶向下，将一个功能繁多，设计复杂的电路拆分成几个简单地部分电路，然后开始动手设计，确保每一部分实现自己的功能，来保证最后完整电路的功能可以正确实现。

这次实验最后的结果还算圆满，但也存在一些小小的遗憾。这次实验学习了VHDL的编写方法，但还是没有做到全代码，部分还是借助了原理图来实现，希望在下星期的DDS实验中可以实现全代码的编写。

最后，非常感谢老师和周围同学在本次实验中给我的帮助，经过这次实验， 我真的收获了很多，体验到了设计电路的乐趣。同时，我相信这次经历也会给今后的学习和生活带来莫大的帮助。

# 七、参考文献

【1】数字逻辑电路与系统设计.蒋立平.北京:电子工业出版社,2008.7

【2】VHDL数字电路设计教程.佩德罗尼.北京:电子工业出版社, 2013.1

【3】VHDL应用教程. 杨光.北京:电子工业出版社, 2017.5