**东 北 大 学**

**课程设计报告**



课程名称: 数字电子技术课程设计

设计题目： 八位模型机

指导教师： 闫\*\*\*\*

学生姓名： \*\*\*\* 学号： 2016\*\*\*\*

专 业： 计算机科学与技术 班级： \*\*\*\*

设计日期： 2018 年7月8日~2018 年7月14日

# 课程设计（论文）任务书

|  |
| --- |
| **课程设计（论文）题目：基于FPGA的8位模型机的设计与实现** |
| **基本内容 基本要求：**  用VHDL设计一个8位模型机系统,包括设计实现指令系统、指令格式、寻址方式、寄存器结构、数据表示方式、存储器系统、运算器、控制器、输入输出设备控制器等,用VHDL硬件语言进行逻辑设计与实现. |

摘要

计算机技术发展到现在,已经成为了人类社会不可缺少的技术工具.去[银行](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C" \o "银行)、去办证件、入学就医、政府部门办公、超市购物、交朋友娱乐等,都离不开计算机和计算机技术.而模型计算机是我们日常使用的计算机的简化版,功能上相近,而且更容易实现,这次实现的目标：８位模型计算机是我们学习数字电子技术的绝佳练习对象.

通过实现８位模型计算机可以很好的了解计算机内部的架构,帮助我们去了解运算方法的基本原理与[运算器](https://baike.baidu.com/item/%E8%BF%90%E7%AE%97%E5%99%A8/2667320" \t "https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E6%8A%80%E6%9C%AF/_blank)设计、[指令系统](https://baike.baidu.com/item/%E6%8C%87%E4%BB%A4%E7%B3%BB%E7%BB%9F/3220297" \t "https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E6%8A%80%E6%9C%AF/_blank)、[中央处理器](https://baike.baidu.com/item/%E4%B8%AD%E5%A4%AE%E5%A4%84%E7%90%86%E5%99%A8/284033" \t "https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E6%8A%80%E6%9C%AF/_blank)(CPU)设计、[流水线原理](https://baike.baidu.com/item/%E6%B5%81%E6%B0%B4%E7%BA%BF%E5%8E%9F%E7%90%86/3225581" \t "https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E6%8A%80%E6%9C%AF/_blank)及其在CPU设计中的应用、存储体系、总线与输入输出.

本次课程设计采用ALINX公司生产的AX301黑金开发板,这块开发板搭载了很多设备,此次设计所用到的只有其中的FPGA芯片(EP4CE6F17C8),50MHz有源晶振,LED灯,数码管,按键和蜂鸣器.利用Quartus Prime 17.1这款EDA来编写VHDL语言源代码和画Block Diagram/Schematic原理图,最后用EDA下载到板子上完成测试.

**关键词：**FPGA, 8位模型机,Quartus Prime 17.1,VHDL,AX301

目录

[课程设计（论文）任务书 i](#_Toc519507917)

[摘要 2](#_Toc519507918)

[第1章 绪论 6](#_Toc519507919)

[1.1 8位模型机简介 6](#_Toc519507920)

[1.2设计主要内容 6](#_Toc519507921)

[1.2.1 指令系统 6](#_Toc519507922)

[1.2.2 控制系统 6](#_Toc519507923)

[1.2.3 总线系统 6](#_Toc519507924)

[1.2.4 时序系统 7](#_Toc519507925)

[1.2.5 输入输出系统 7](#_Toc519507926)

[第2章 系统设计 8](#_Toc519507927)

[2.1系统组成 8](#_Toc519507928)

[2.2 CPU模块 8](#_Toc519507929)

[2.2.1 中央控制器(Center Controller) 9](#_Toc519507930)

[2.2.2 节拍器(Beater) 14](#_Toc519507931)

[2.2.3 延时器(Delayer) 16](#_Toc519507932)

[2.2.4 数据寄存器(Data Register,Dreg) 17](#_Toc519507933)

[2.2.5 算术逻辑单元(Algorithm Logic Unit,ALU) 19](#_Toc519507934)

[2.2.6 只读存储器(Read Only Memory,ROM) 21](#_Toc519507935)

[2.2.7 随机访问存储器(Randomly Access Memory,RAM) 22](#_Toc519507936)

[2.2.8 栈存储器(Stack) 24](#_Toc519507937)

[2.3输入输出模块 25](#_Toc519507938)

[2.3.1按键消抖单元(Key AntiShake Unit) 26](#_Toc519507939)

[2.3.2蜂鸣器控制器(Buzzer Controller) 27](#_Toc519507940)

[2.3.3数码管扫描器(Number Scanner) 28](#_Toc519507941)

[2.4模块结合 30](#_Toc519507942)

[第3章 系统仿真 31](#_Toc519507943)

[3.1 按键防抖单元仿真 31](#_Toc519507944)

[3.2 数据寄存器仿真 31](#_Toc519507945)

[3.3 算术逻辑单元仿真 31](#_Toc519507946)

[3.4 节拍器仿真 32](#_Toc519507947)

[3.5 栈存储器仿真 32](#_Toc519507948)

[3.6 整体仿真 33](#_Toc519507949)

[第4章 系统实现 38](#_Toc519507950)

[第5章 结论 41](#_Toc519507951)

[参考文献 42](#_Toc519507952)

[附录A RTL图 43](#_Toc519507953)

[A1. 按键防抖单元的RTL图 43](#_Toc519507954)

[A2. 数据寄存器的RTL图 43](#_Toc519507955)

[A3. 48译码器RTL图 44](#_Toc519507956)

[A4. 数码管扫描器RTL图 45](#_Toc519507957)

[A5. 随机访问存储器 45](#_Toc519507958)

[A6. 节拍器RTL图 46](#_Toc519507959)

[A7. 栈存储器RTL图 47](#_Toc519507960)

[A8. 算术逻辑单元RTL图 47](#_Toc519507961)

[A9. 只读存储器RTL图 48](#_Toc519507962)

[A10. 延时器RTL图 48](#_Toc519507963)

[A11. 中央控制器RTL图 49](#_Toc519507964)

[A12. 蜂鸣器控制器RTL图 49](#_Toc519507965)

[附录B VHDL程序清单 50](#_Toc519507966)

[B1. AntiShake.vhd 50](#_Toc519507967)

[B2. Led\_Decoder.vhd 50](#_Toc519507968)

[B3. dreg.vhd 51](#_Toc519507969)

[B4. alu.vhd 52](#_Toc519507970)

[B5. beater.vhd 54](#_Toc519507971)

[B6. buzzer\_controller.vhd 55](#_Toc519507972)

[B7. controller.vhd 58](#_Toc519507973)

[B8. delayer.vhd 79](#_Toc519507974)

[B9. memory.vhd 81](#_Toc519507975)

[B10. rom.vhd 82](#_Toc519507976)

[B11. stack.vhd 110](#_Toc519507977)

[B12. num\_scan.vhd 111](#_Toc519507978)

[附录C 实物测试照片 114](#_Toc519507979)

# 第1章 绪论

## 1.1 8位模型机简介

所谓模型计算机就是一计算机实际结构为基础,将其简化,能对输入的信息进行处理运算,更便于分析设计.随着微电子技术的进步,现代计算机主要由运算器,控制器,存储器,输入设备,输出设备五大部分组成.计算机能完成用户要求是按照提前设计好的指令进行的,指令是计算机执行具体操作的命令.一条指令就是机器语言的一个语句,用它来说明机器硬件应完成什么样的基本操作.让预设指令在这些部件中按顺序执行达到预期目的.计算机执行一条指令分为三步进行：

第1步是取指令,将要执行的指令从存储器中取到控制器中；

第2步是执行指令,对所取的指令通过判断语句进入对应的执行块,然后按节拍器的输出执行一系列微指令完成指令的目标；

第3步是执行善后,根据不同情况将一些状态变量置为无效值和对cs:ip指令地址寄存器的操作.

## 1.2设计主要内容

可执行指令的八位模型机,可以分成指令系统,控制系统,总线系统,时序系统,输入输出系统.

### 1.2.1 指令系统

指令被设计成了8位,因此一共有256条指令可供使用,其中有128条送数指令,64条寄存器相互传值指令,6种单操作数指令,20条固定功能指令

### 1.2.2 控制系统

中央控制器能控制其他元器件,通过一系列使能赋值可以完成控制,比如送数,先将寄存器读控制端打开,然后向总线写值.最后再将读控制端关闭

### 1.2.3 总线系统

此次设计中只有一条数据总线,其他总线以其他形式存在,比如控制总线的存在形式就是中央控制器对其他元件的使能线的总和,地址总线则以cs:ip寄存器组组成.

### 1.2.4 时序系统

时序系统有两部分组成:beat和state,也就是用节拍器和有限状态自动机进行时序控制,首先在控制内部用state来决定指令执行的阶段,然后控制器根据state来控制节拍器的输出,根据节拍器来执行控制指令.

### 1.2.5 输入输出系统

此次设计中控制了4个LED灯,6个数码管,蜂鸣器和4个按键.按键和消抖模块组成输入系统直接和中央控制器相连,4个LED不做处理的连到中央控制器上充当输出端,6个数码管被数码管扫描器按一定频率扫描输出,然后数码管扫描器和中央控制器相连,蜂鸣器被蜂鸣器控制器控制以便发出不同频率,响度的声音了,然后蜂鸣器控制器直接跟控制器相连,LED,数码管和蜂鸣器构成了输出系统.

# 第2章 系统设计

## 2.1系统组成

模型机被划分成了十一个模块,分别是算术逻辑单元,中央控制器,只读存储器,随机访问存储器,栈存储器,数据寄存器,按键消抖单元,数码管扫描器 ,蜂鸣器控制器,延时器,节拍器.

## 2.2 CPU模块

CPU由中央控制器,节拍器,延时器,算术逻辑单元和寄存器组组成,中央控制器接受时序信号:节拍,然后向寄存器组发送读/写使能信号或向延时器发送延时信号,寄存器组返回数据给中央控制器/算术逻辑单元,然后中央控制器根据情况控制节拍器继续计数或清零或控制计算指令给算术逻辑单元.

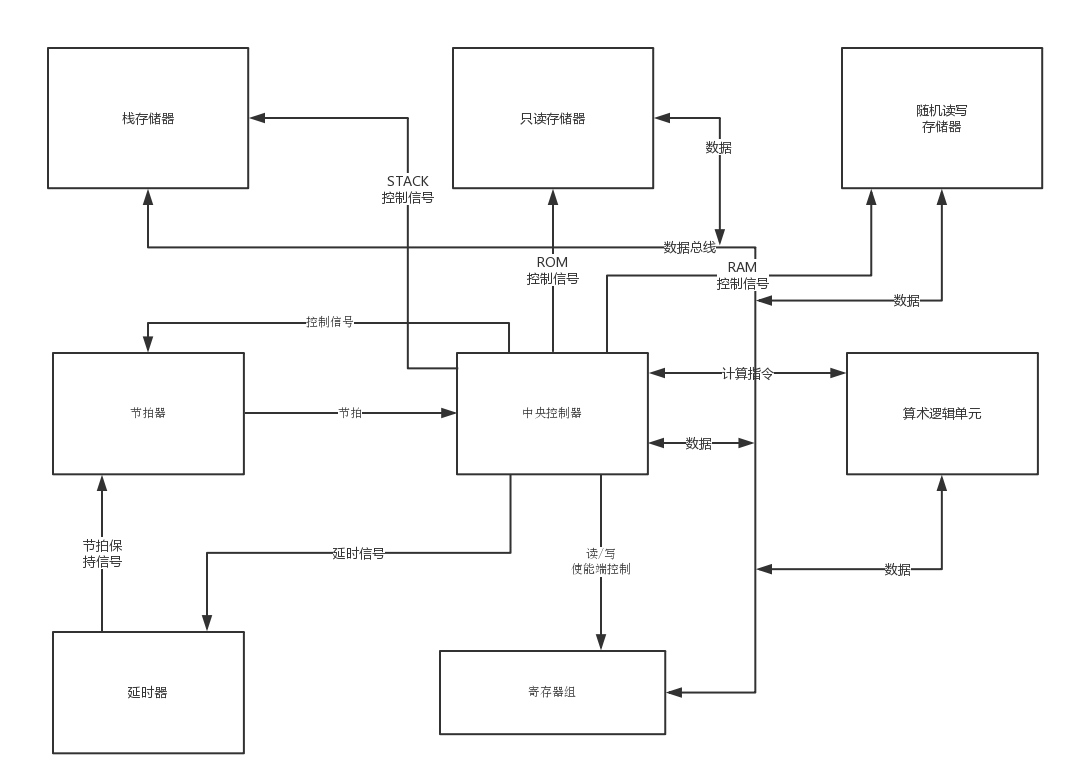


图2.1 CPU整体结构图

表2.1指令系统

|  |  |  |
| --- | --- | --- |
| 汇编语句 | 操作符 | 功能 |
| mov ix,abcdefg | 1abcdefg | 送abcdefg给ix |
| mov r1,r2 | 01sssddd | 送ddd所指示的寄存器给sss所指示的寄存器 |
| jmp cs:px | 00100000 | 跳到同段的px位置执行 |
| jmp ax:px | 00100001 | 跳到ax段的px位置执行 |
| jc cs:px | 00100010 | 如果CF=1,跳转到同段的px位置执行 |
| stos | 00100011 | 将dx的值存入RAM中ds段的si位置 |
| jz cs:px | 00100100 | 如果ZF=1,跳转到同段的px位置执行 |
| j | 00100101 | 把栈顶送给第一操作数寄存器,给标志位赋值 |
| js cs:px | 00100110 | 如果SF=1,跳转到同段的px位置执行 |
| jodd cs:px | 00100111 | 如果ODD=1,跳转到同段的px位置执行 |
| push r | 00101sss | 把sss号寄存器压栈 |
| pop r | 00110sss | 把sss号寄存器弹栈 |
| inc r | 00111sss | sss号寄存器自增1 |
| dec r | 00011sss | sss号寄存器自减1 |
| out id | 00010ooo | 把dx的内容输出到ooo号设备  0~5:数码管 6:led灯 7:蜂鸣器  如果是7,则固定是让蜂鸣器反转鸣叫状态 |
| xor ax,px | 00001111 | ax = ax 异或px |
| mode | 00001110 | 指令执行态从rom到ram的相互转化  默认是rom态,执行一次转到ram态 |
| shl ax,1 | 00001101 | ax左移一位 |
| or ax,px | 00001100 | ax = ax或px |
| lods | 00001011 | 读入RAM的ds段的si号单元到dx |
| shr ax,1 | 00001010 | ax 右移一位 |
| wait | 00001001 | 等待输入设备,若状态发生,则执行下一条指令 |
| and ax,px | 00001000 | ax = ax 与 px |
| in id | 000001ii | 读入ii号输入设备的状态 |
| jz ax:px | 00000010 | 如果ZF=1,则跳转到ax段的px |
| reset | 00000011 | 把状态寄存器设为11111111 |
| add ax,px | 00000001 | ax=ax+px |
| delay | 00000000 | 延时dx个毫秒 |

### 2.2.1 中央控制器(Center Controller)

1) 简介:

中央主控制器是一个管控所有外围器件的控制器,逻辑均在控制器内部,该模块拥有对存储器组,寄存器组,输入输出设备的端口,它受clk信号的上升沿控制.

2) 框图:

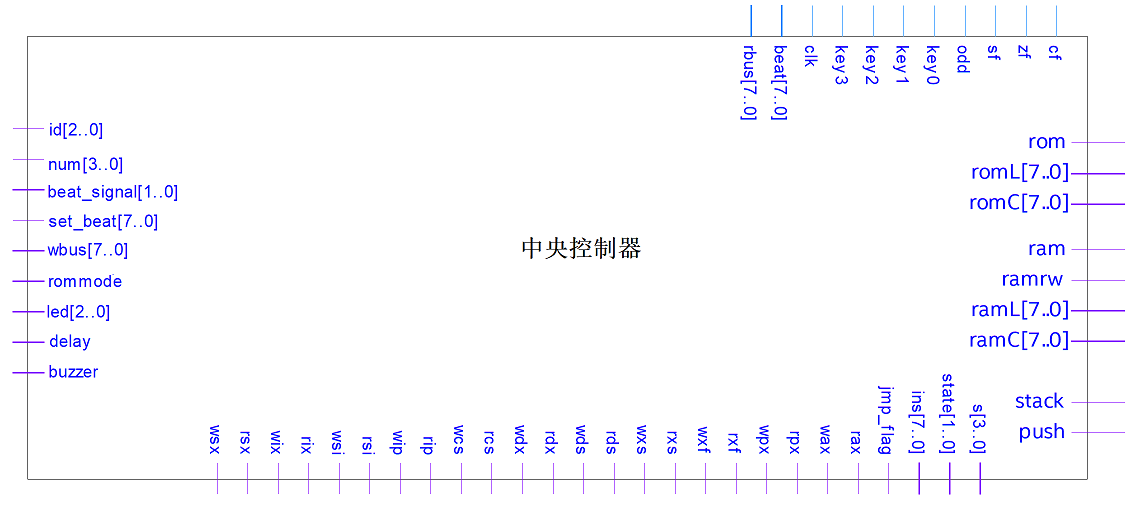


图2.2 中央控制器端口图

3) 端口解释:

(1)输入:

clk:50MHz时钟输入

rbus:总线的读取端,通过总线可以读取到指令

cf,zf,sf,odd:ALU的标志位,用于跳转指令

key0,key1,key2,key3:按键消抖模块的输出

beat:节拍

(2)输出:

rom:控制只读存储器的使能端

romL,romC:控制读取只读存储器的行和列

ram,ramrw:控制随机访问存储器的使能端和读写模式

ramL,ramC:控制读取随机访问存储器的行和列

stack,push:控制栈存储器的使能端和模式

s:ALU的模式

state:用于debug的内部状态变量

ins:用于debug的内部指令变量

jmp\_flag:用于debug的内部跳转标志变量

rax,wax, rpx,wpx,rxf,wxf,rxs,wxs,rds,wds,rdx,wdx,rcs,

wcs,rip,wip,rsi,wsi,rix,wix,rsx,wsx :控制寄存器组的读/写使能端组

id:控制更新数码管扫描器的第几个数据

num:给数码管扫描器的数据

beat\_signal:节拍器控制信号

set\_beat:给节拍器的数据

wbus:给总线写的数据

rommode:与LED0相邻,用于指示读取指令的模式(rom/ram)

led:控制LED1,LED2,LED3

delay:给延时器的使能信号

buzzer:给蜂鸣器的使能信号

4)流程图:

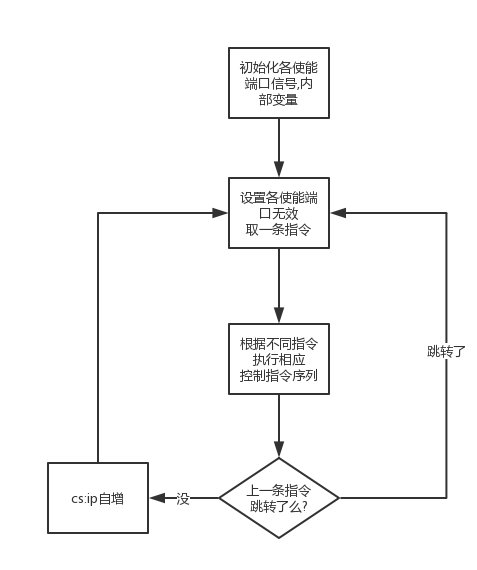


图2.3 中央控制器内部状态机流程图

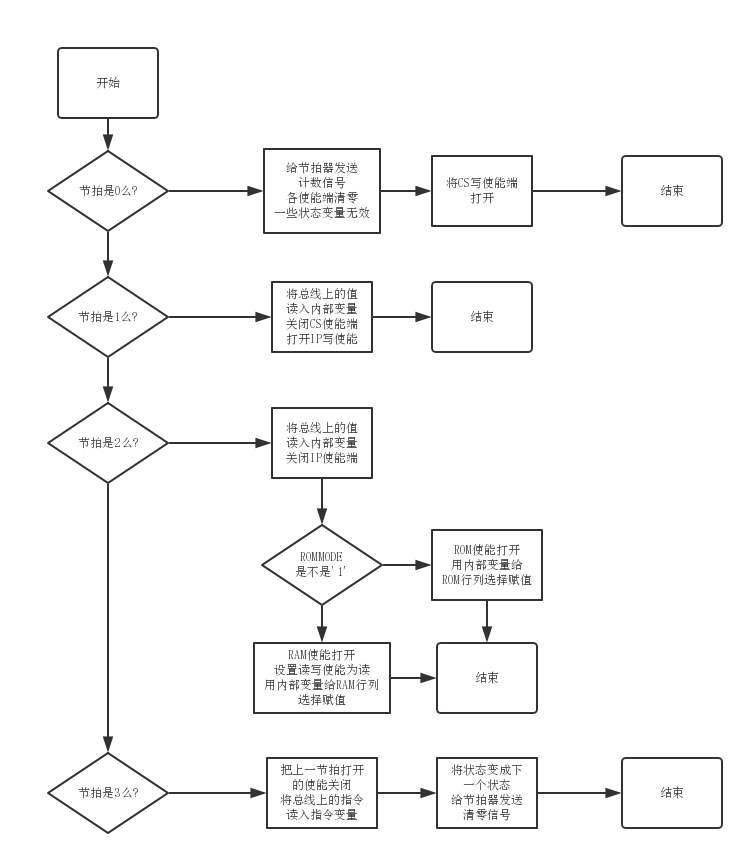


图2.4 取指令控制序列流程图

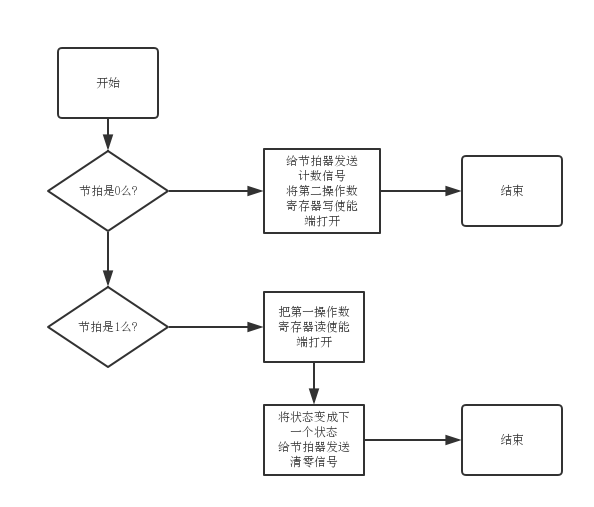


图2.5 mov r1,r2控制指令序列举例

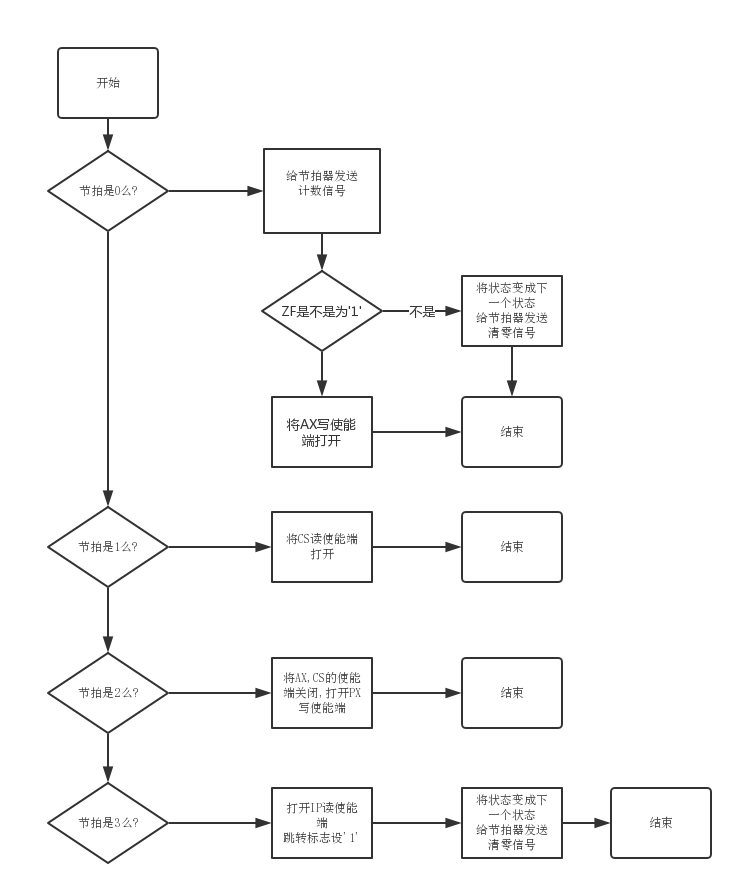


图2.6 jz ax:px控制指令序列举例

备注:由于指令太多而且解释指令的方式是使用VHDL语言的if..elsif..elsif.......endif来实现的,所以在状态机流程图中就不详细展开,只举2例说明控制指令序列.

### 2.2.2 节拍器(Beater)

1) 简介:

节拍器是一个受中央主控制器和延时器控制的计数器,一共有4种工作模式,计数/清零/保持/置数,它受clk信号的下降沿控制.

2) 框图:

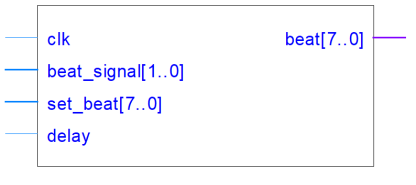


图2.7 节拍器端口图

3) 端口解释:

(1)输入:

clk:50MHz时钟输入

beat\_signal:控制器给的模式信号

set\_beat:置数模式所置的数

delay:延时器给的保持信号

(2)输出:

beat:给控制器的节拍(整数型)从0~255

4) 流程图:

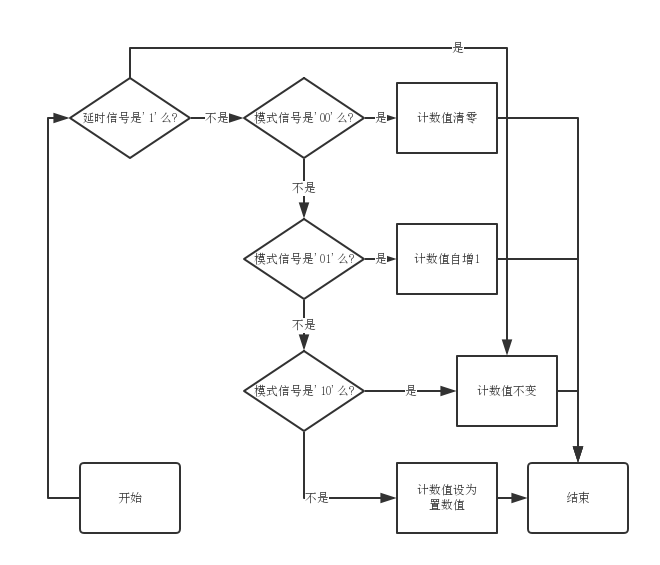


图2.8 节拍器流程图

### 2.2.3 延时器(Delayer)

1) 简介:

延时器是一个根据DX的值延时的器件,内部用计数器实现,可以延时DXms,它受clk信号的下降沿控制.

2) 框图:

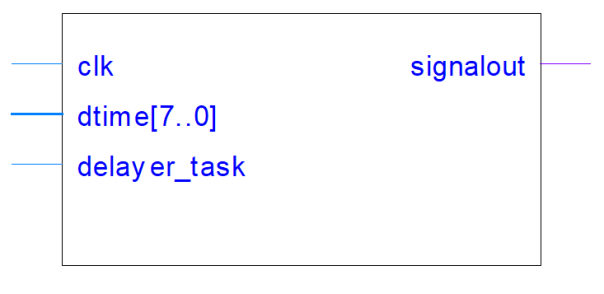


图2.9 延时器端口图

3) 端口解释:

(1)输入:

clk:50MHz时钟输入

dtime:DX输入

delayer\_task:延时器的使能端

(2)输出:

signalout:给节拍器的输出,为’0’时,节拍器进入保持模式

4) 流程图:

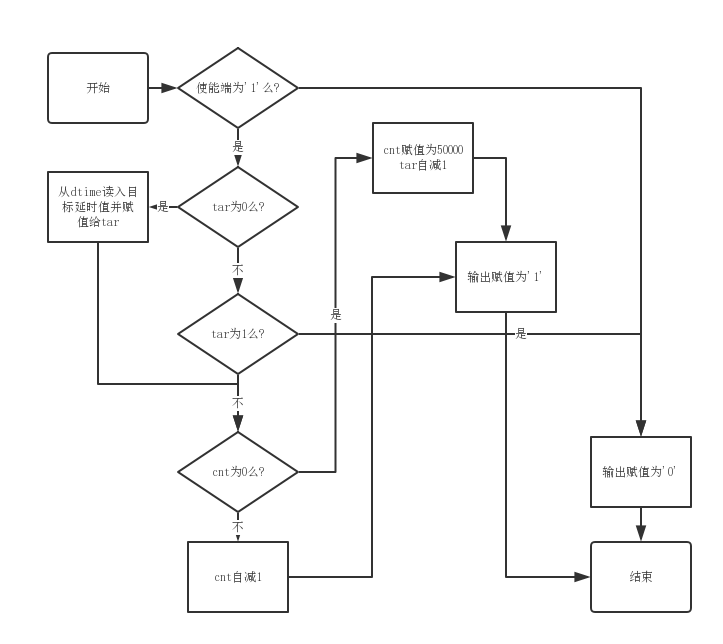


图2.10 延时器流程图

### 2.2.4 数据寄存器(Data Register,Dreg)

1) 简介:

数据寄存器是一个具有读/写总线功能的存储8位数据的存储器,它受clk信号的下降沿控制.

2) 框图:

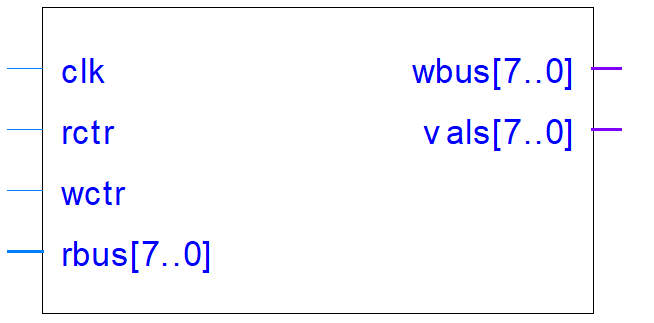


图2.11 寄存器端口图

3) 端口解释:

(1)输入:

clk:50MHz时钟输入

rbus:总线的读取端

rctr:控制器的读控制端,为’1’时,通过rbus读入总线上的数据

wctr:控制器的写控制端,为’1’时,通过wbus向总线写数据

(2)输出:

wbus:总线的输出端

vals:内容的直接输出端

4) 流程图:

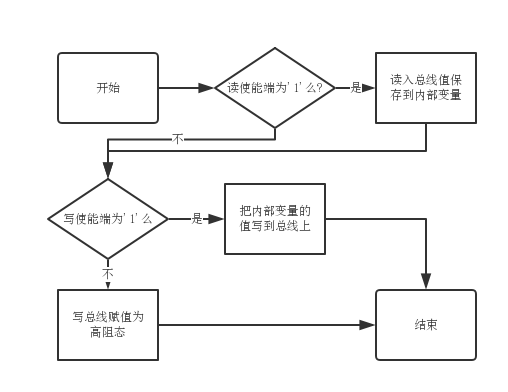


图2.12 数据寄存器流程图

### 2.2.5 算术逻辑单元(Algorithm Logic Unit,ALU)

1) 简介:

算术逻辑单元是一个具有配合第一操作数寄存器运算功能的寄存器,为实现算术运算和逻辑运算提供方便,该模块除了拥有数据寄存器有的clk,rbus,wbus,rctr,wctr,vals端口,还有独有的xf,cf,zf,sf,odd端口,它的电路是组合逻辑电路

2) 框图:

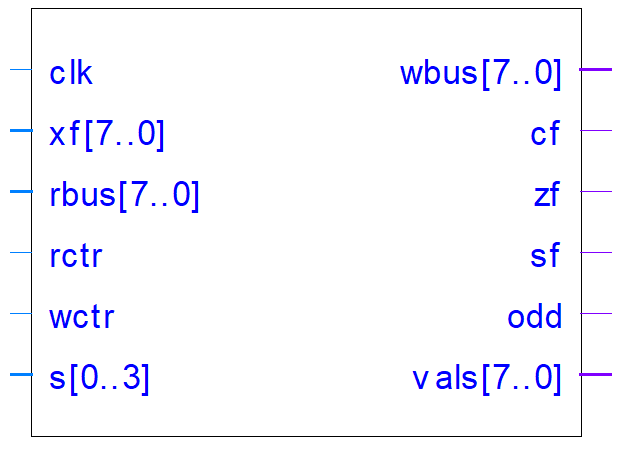


图2.13 算术逻辑单元端口图

3) 端口解释:

(1)输入:

clk:50MHz时钟输入

rbus:总线的读取端

rctr:控制器的读控制端,为’1’时,通过rbus读入总线上的数据

wctr:控制器的写控制端,为’1’时,通过wbus向总线写数据

xf:第一操作数寄存器的值

(2)输出:

wbus:总线的输出端

cf:Carry Flag,输出给控制器的进位标志位

zf:Zero Flag,输出给控制器的零标志位

sf:Sign Flag,输出给控制器的符号标志位

odd:Odd Flag,输出给控制器的奇标志位

4) 流程图:

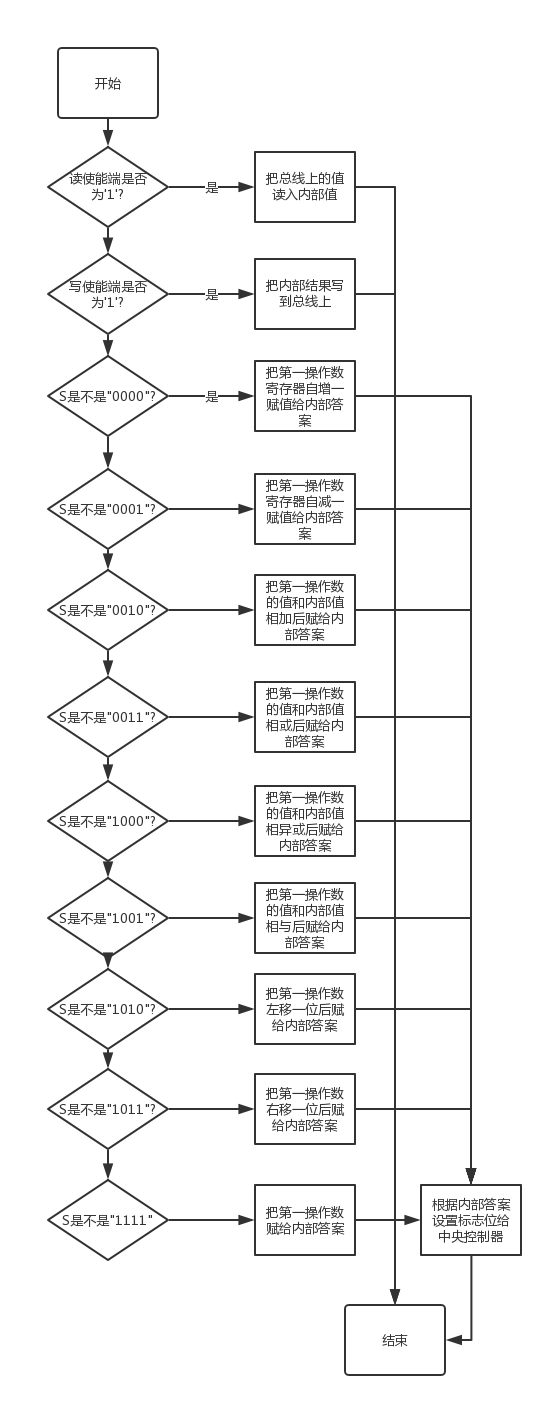


图2.14 算术逻辑单元流程图

### 2.2.6 只读存储器(Read Only Memory,ROM)

1) 简介:

只读存储器是一个只能读数据不能被写数据的存储器,它受clk信号的下降沿控制.

2) 框图:

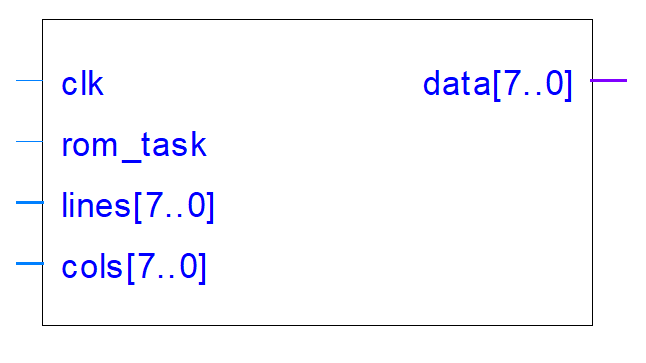


图2.15 只读存储器端口图

3) 端口解释:

(1)输入:

clk:50MHz时钟输入

rom\_task:ROM的使能端,’1’有效

lines:表明需要的数据的段地址(二进制补码形式)

cols:表明需要的数据的偏移地址(二进制补码形式)

(2)输出:

data:总线的输出端

4) 流程图:

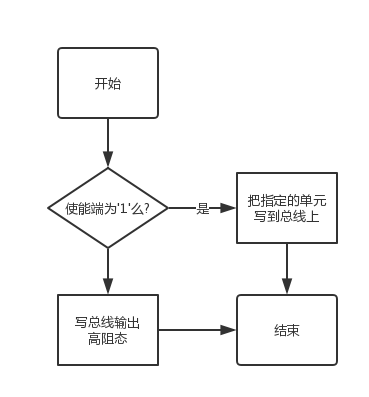


图2.16 只读存储器流程图

5) 备注:代码中预设的大小是5x256Byte,可以根据要求调大,和内存的大小加起来最大可以到120x256Byte左右

### 2.2.7 随机访问存储器(Randomly Access Memory,RAM)

1) 简介:

随机访问存储器是一个可读可写的存储器,它受clk信号的下降沿控制.

2) 框图:

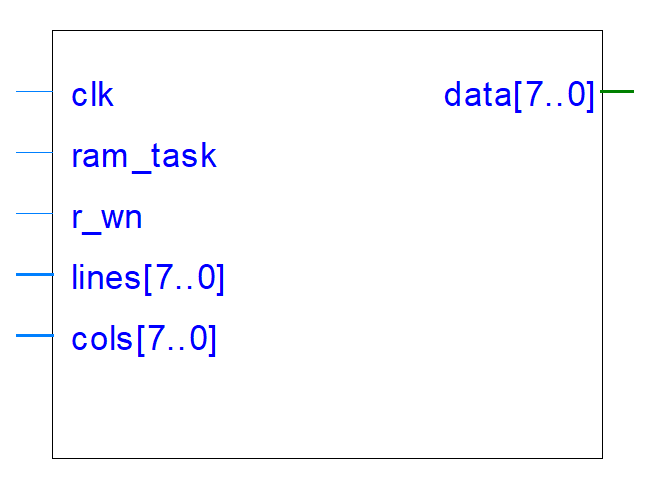


图2.17 随机访问存储器端口图

3) 端口解释:

(1)输入:

clk:50MHz时钟输入

ram\_task:RAM的使能端,’1’有效

r\_wn:表明是读还是写,’1’为读取

lines:表明读/写的数据的段地址(二进制补码形式)

cols:表明读/写的数据的偏移地址(二进制补码形式)

(2)输入输出:

data:总线的输入输出端口

4) 流程图:

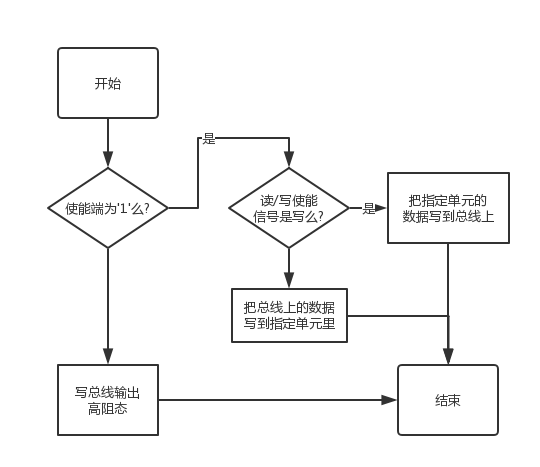


图2.18 随机访问存储器流程图

5) 备注:预设的大小是16x256Byte,可以调大到100左右

### 2.2.8 栈存储器(Stack)

1) 简介:

栈存储器是一个配合push/pop指令的先进后出存储器,它受clk信号的下降沿控制.

2) 框图:

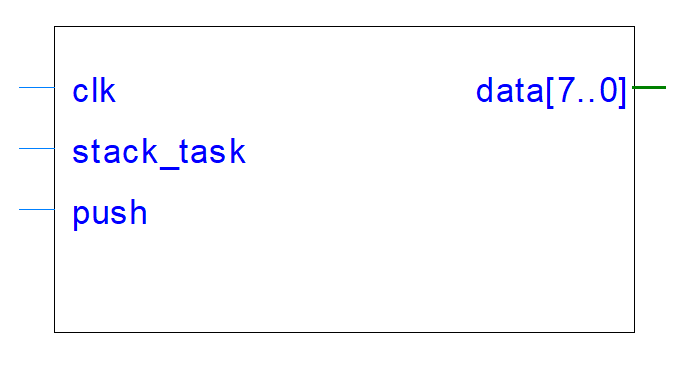


图2.19 栈存储器端口图

3) 端口解释:

(1)输入:

clk:50MHz时钟输入

stack:栈的使能端,’1’有效

push:表明数据是压栈还是弹栈,’1’为压栈

(2)输出:

data:总线的输入输出端

4) 流程图:

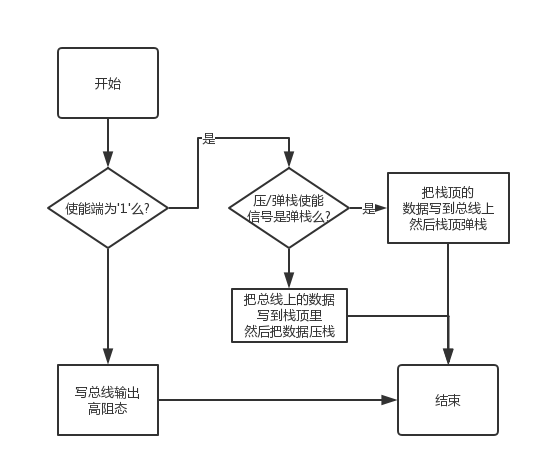


图2.20 栈存储器流程图

5) 备注:预设的大小是128Byte,可以调大

# 2.3输入输出模块

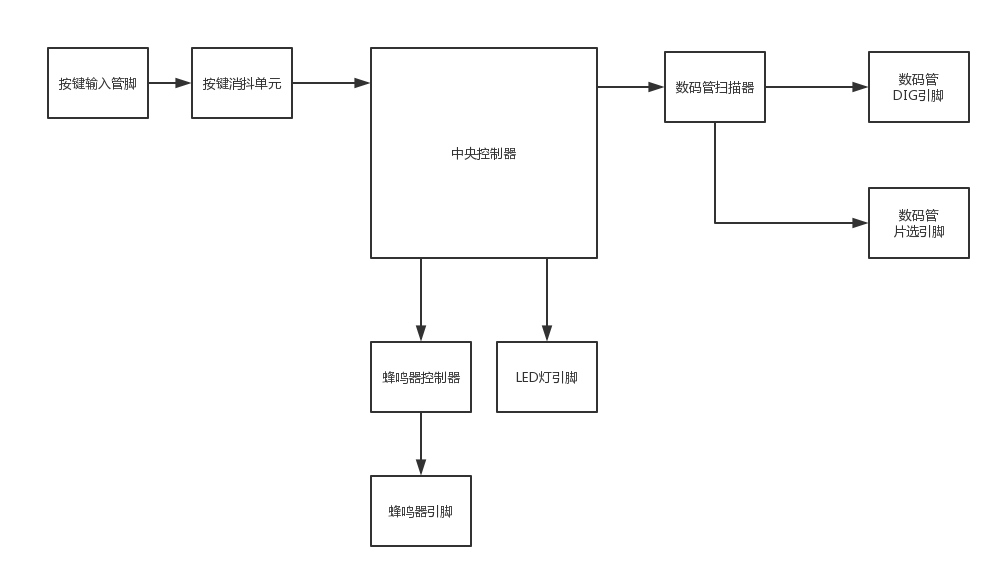


图2.21 输入输出系统整体结构图

### 2.3.1按键消抖单元(Key AntiShake Unit)

1) 简介:

按键消抖单元是将按键硬件输入中的抖动消除后输出的单元,是时序逻辑电路,内部设有计时器,若一个硬件信号稳定10ms以上就输出这个硬件信号,它受clk信号的下降沿控制.

2) 框图:

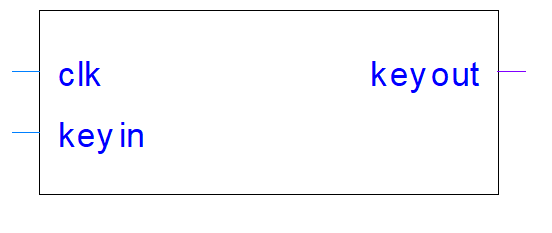


图2.22 按键消抖单元端口图

3) 端口解释:

(1)输入:

clk:50MHz时钟输入

keyin:硬件信号输入

(2)输出:

keyout:消抖后按键信号输出

4) 流程图:

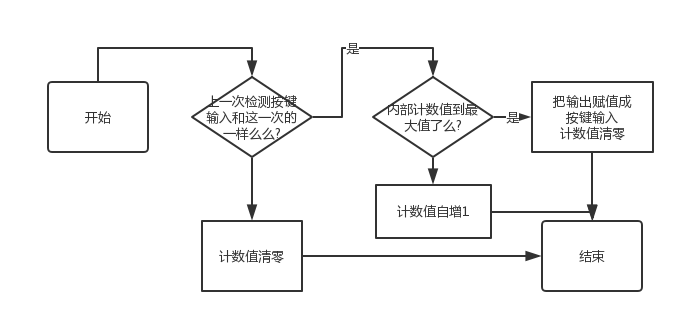


图2.23 按键消抖单元流程图

### 2.3.2蜂鸣器控制器(Buzzer Controller)

1) 简介:

蜂鸣器控制器是一个控制蜂鸣器发出声音的频率和响度的元件,它受clk信号的下降沿控制.

2) 框图:

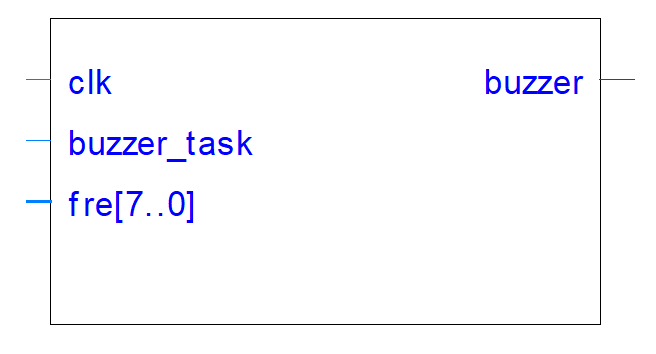


图2.24 蜂鸣器端口图

3) 端口解释:

(1)输入:

clk:50MHz时钟输入

buzzer\_task:蜂鸣器使能端,’0’有效

fre:直接与AX相连,接受AX的控制

(2)输出:

buzzer:真正蜂鸣器的输出端

4) 流程图:

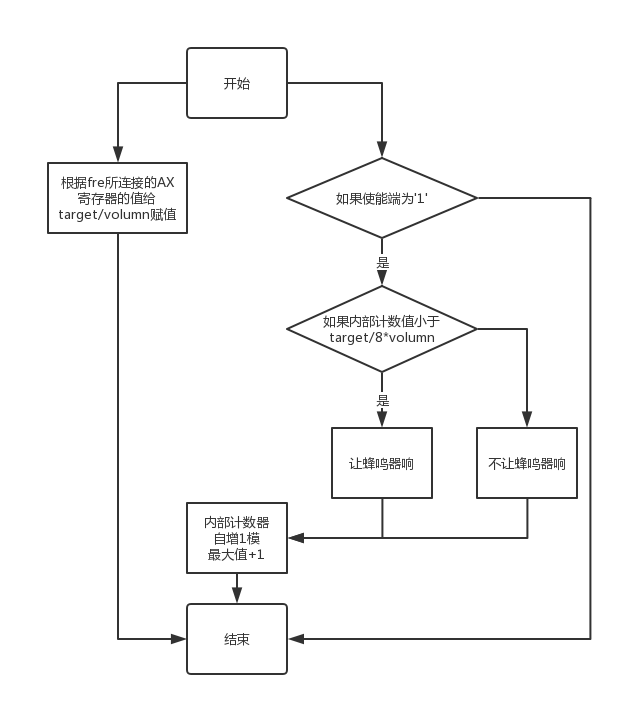


图2.25 蜂鸣器控制器流程图

5) 备注:AX的高三位控制音量,"111"为最响,"000"为静音,后五位是控制频率的,"00000"是低1音,"11011"是倍高7音

### 2.3.3数码管扫描器(Number Scanner)

1) 简介:

数码管扫描器是一个以一定频率扫描数码管并把内部存储的值给译码器让数码管显示相应的值的器件,它受clk信号的下降沿控制.

2) 框图:

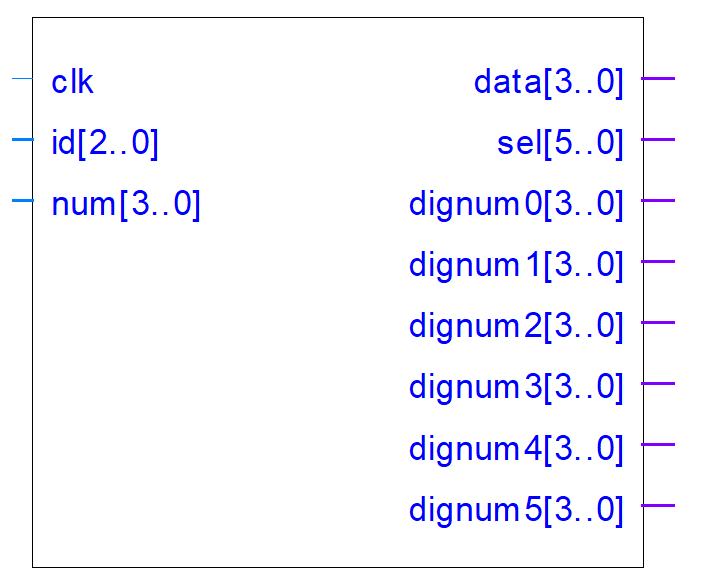


图2.26 数码管扫描器端口图

3) 端口解释:

(1)输入:

clk:50MHz时钟输入

id:需更新的数码管编号(“000"~“101")

num:更新的数字

(2)输出:

data:给译码器的数字

sel:接到数码管的片选上,用于选通某一数码管

dignum0,dignum1,dignum2,dignum3,dignum4,dignum5:仅用于debug

4) 流程图:

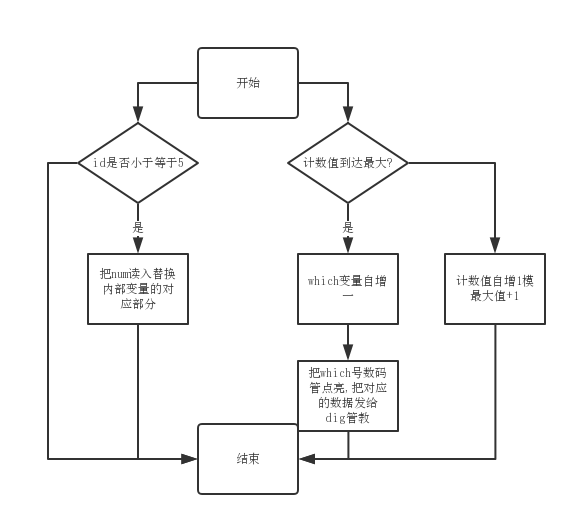


图2.27 数码管扫描器流程图

5) 备注:每个数码管的扫描频率:200Hz

## 2.4模块结合

模块间依靠中央控制器相连,时序系统给控制系统发节拍,控制系统控制其他系统干活,完成初始化后周而复始的取指令,执行指令,指令结束后善后.用指令完成切换指令执行空间形态,等待输入设备状态变化和精准延时.

# 第3章 系统仿真

## 3.1 按键防抖单元仿真

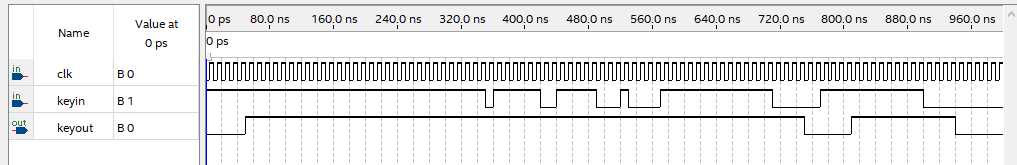


图3.1 按键防抖单元仿真结果

解读:因为是从0到3的计数没所以要超过5个cp才能够算通过

备注:由于按键防抖单元有计数变量但是这个计数变量可以调整而且有意义,所以我把这个计数变量cnt从原来的500000调整到了3,以便更加直观地看到结果.

## 3.2 数据寄存器仿真

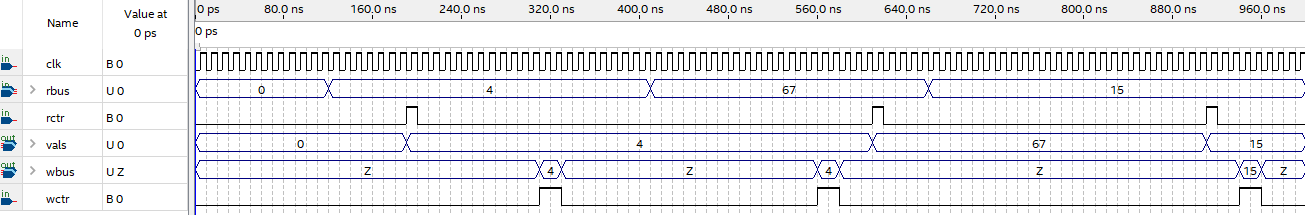


图3.2 数据寄存器仿真结果

解读:在cp下降沿一旦rctr有效,立刻把总线上的值读入,在cp下降沿一旦wctr有效,立刻把内部值写到总线上

## 3.3 算术逻辑单元仿真

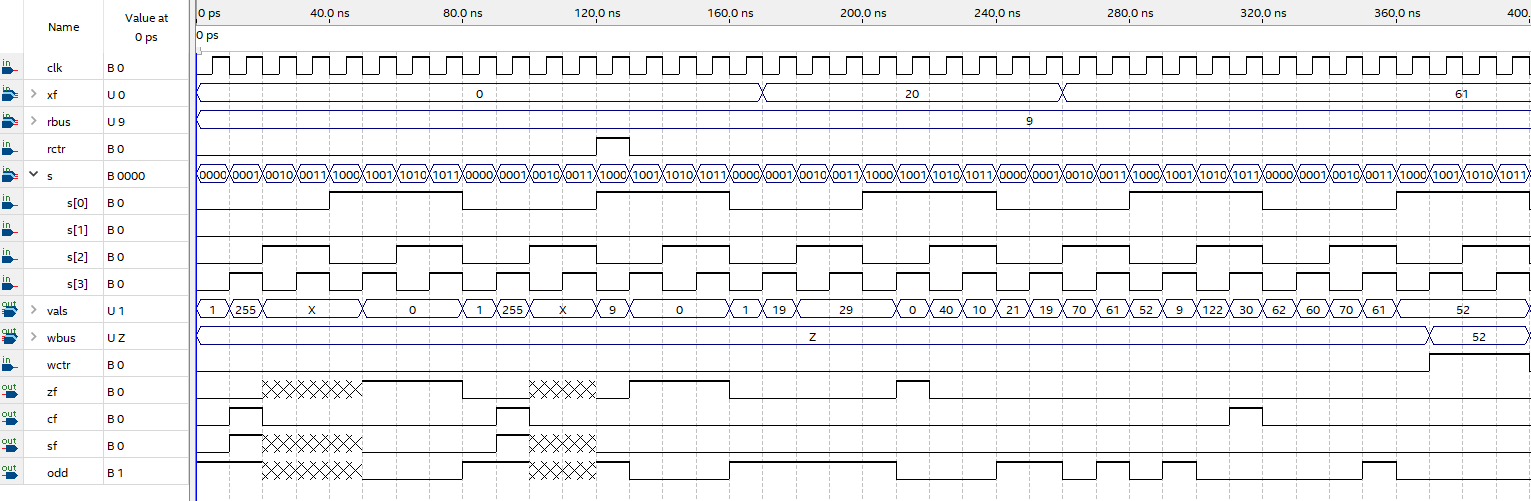


图3.3 算数逻辑单元仿真结果

解读:在rctr之前的一些状态之所以zf,sf,odd会出现强未知的状态是因为ALU内部第二操作数值还没有读入,一些需要两个操作数的操作例如相加,相异或等均不能执行,当rctr为’1’时立刻把总线上的值赋值给第二操作数,之后就没有强未知的状态了.

## 3.4 节拍器仿真

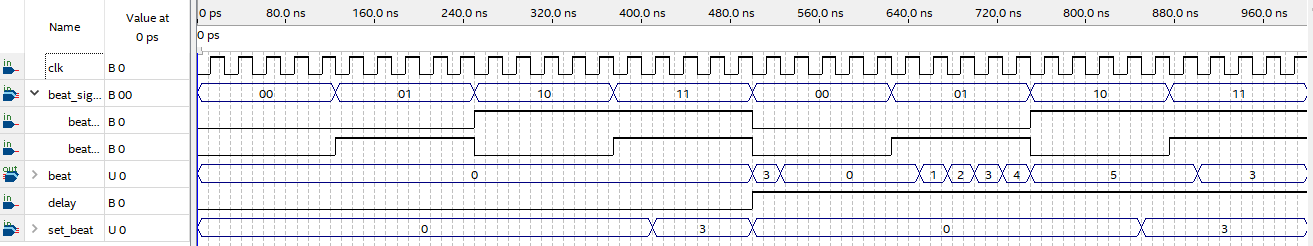


图3.4 节拍器仿真结果

解读:可以看到前半截delay为’0’时,无论节拍信号是什么都没有用,节拍均不变,delay为’1’时,一个瞬间信号还是"11"而且set\_beat为3,所以有一个cp的时间节拍是3,之后信号为"00",节拍清零,"01"顺序计数,"10"保持,"11"置数.

## 3.5 栈存储器仿真

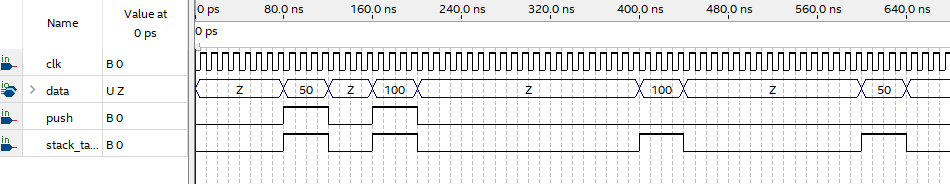


图3.5 栈存储器仿真结果

解读:可以看到当使能信号有效且压/弹栈信号为’1’(压栈)的时候,并没有什么现象,当后两个弹栈信号来临的时候,总线上果然出现了之前压入的50和100,完全符合栈的特性---后进先出.

## 3.6 整体仿真

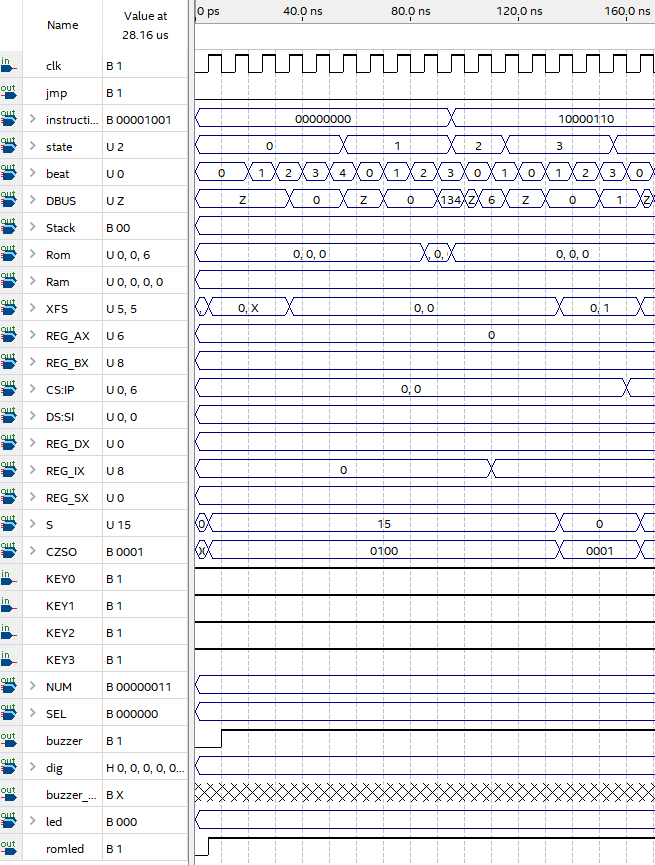


图3.6 整体仿真结果(1/5)

解读:在0~160ns做了两件事,一是state为0时的初始化,二是执行完了10000110,送110给ix,可以看到110ns,ix的值发生了变化,具体是什么可看下一张.

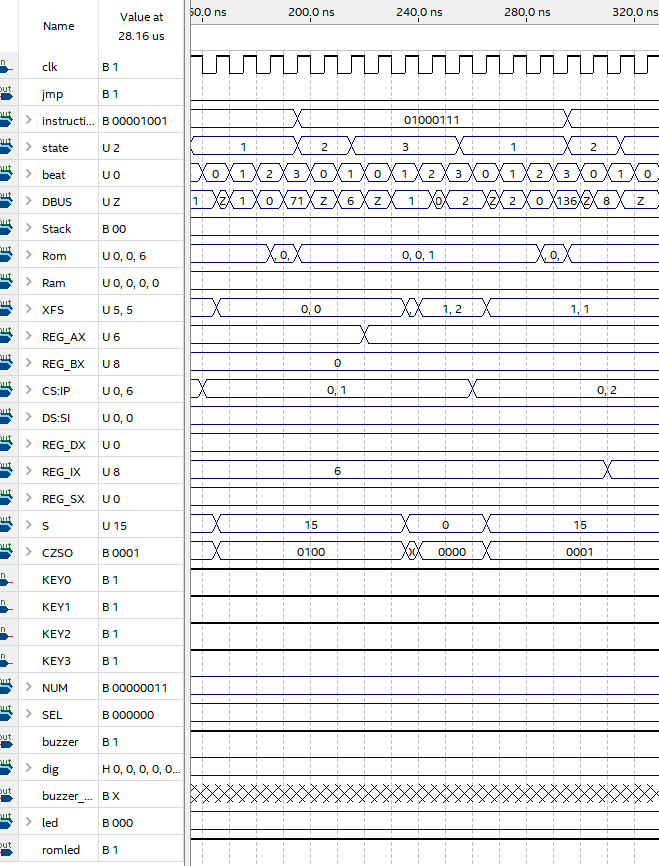


图3.7 整体仿真结果(2/5)

解读:在160~320ns完成了上一条指令ip的自增从0变1,然后执行了01000111,也就是mov ix,ax,可以看到在220ns处,ax发生了变化,具体结果可以看下一张仿真结果.又执行了10001000的一部分,可以看到在310ns处ix又发生了变化.应该是ix变成了8.

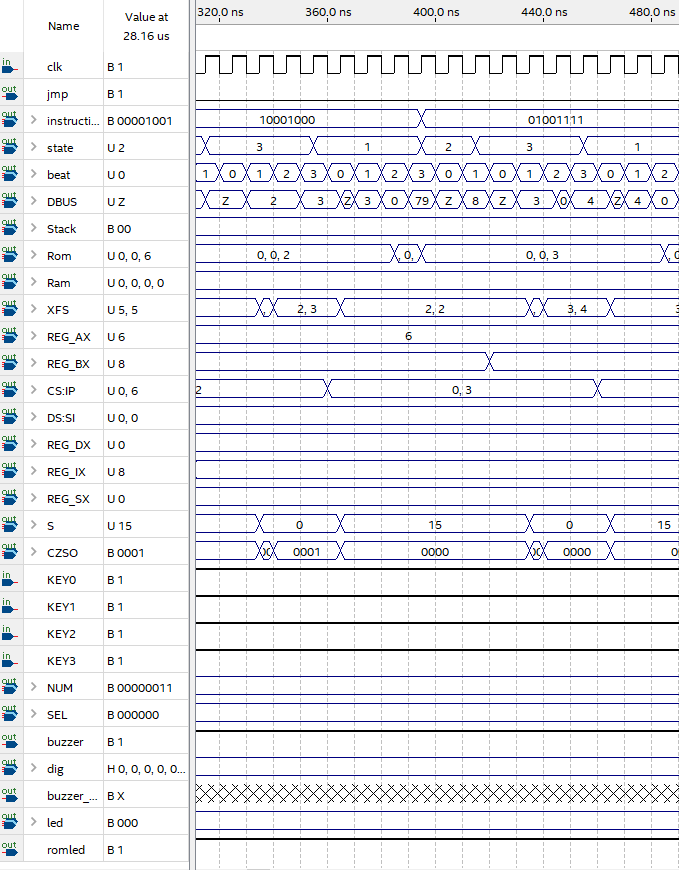


图3.8 整体仿真结果(3/5)

解读:在320~480ns时间里,执行完了上一次刚开始执行的10001000(mov ix,8)和执行了01001111(mov px,ix),可以看到420ns处bx的值发生了变化.cs:ip的自增一直都很正常,都是在最后一个状态state=3的最后一个节拍beat=3处发生自增.

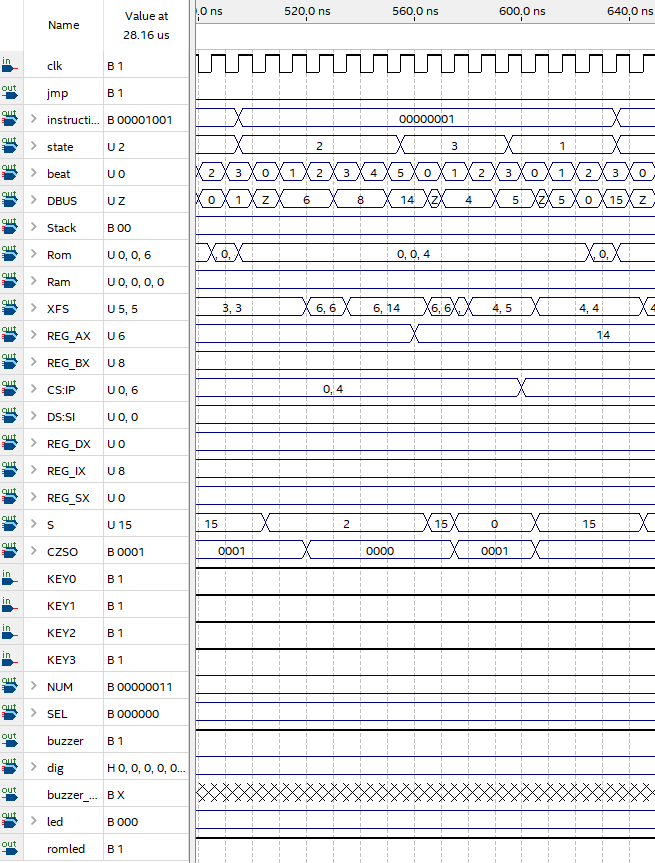


图3.9 整体仿真结果(4/5)

解读:在480~640ns时间里,执行完了上一次执行的01001111(mov px,ix)和执行00000001(add ax,px)了,可以看到505~565ns之间s为2,意味着alu执行的任务是相加,在560ns处答案也就是ax=6+px=8的结果14赋值给了ax.

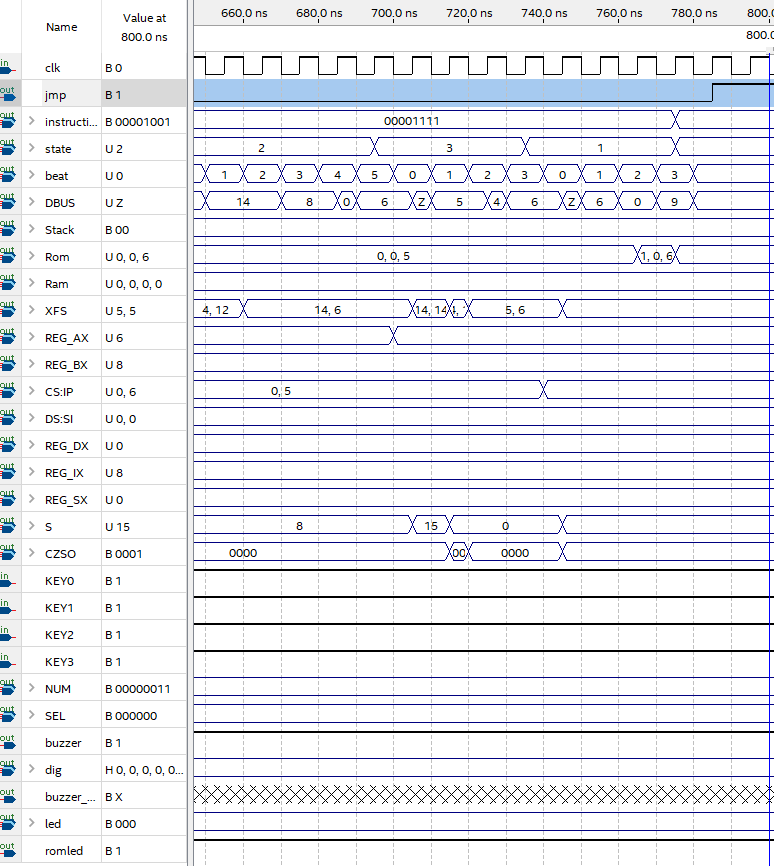


图3.10 整体仿真结果(5/5)

解读:最后两条指令是00001111(xor ax,px)和00001001(wait),当异或的state=2阶段执行完后,ax果然又变回了6,可以在左边的value at 800ns看到.因为14^8=6.最后一条指令wait直接把跳转标记设为’1’,所以ip也不自增了,又把节拍给控制住为清零状态,所以最后只有没有按键输入就会一直停在这条指令.

# 第4章 系统实现

由于整个系统的不可拆分性,不能单独下载验证,单独验证没有可视化结果.所以还是整体系统验证比较简单.

由于详细的部分老师已经验收过了,这里展示一个样例程序的验证,默认加法程序.

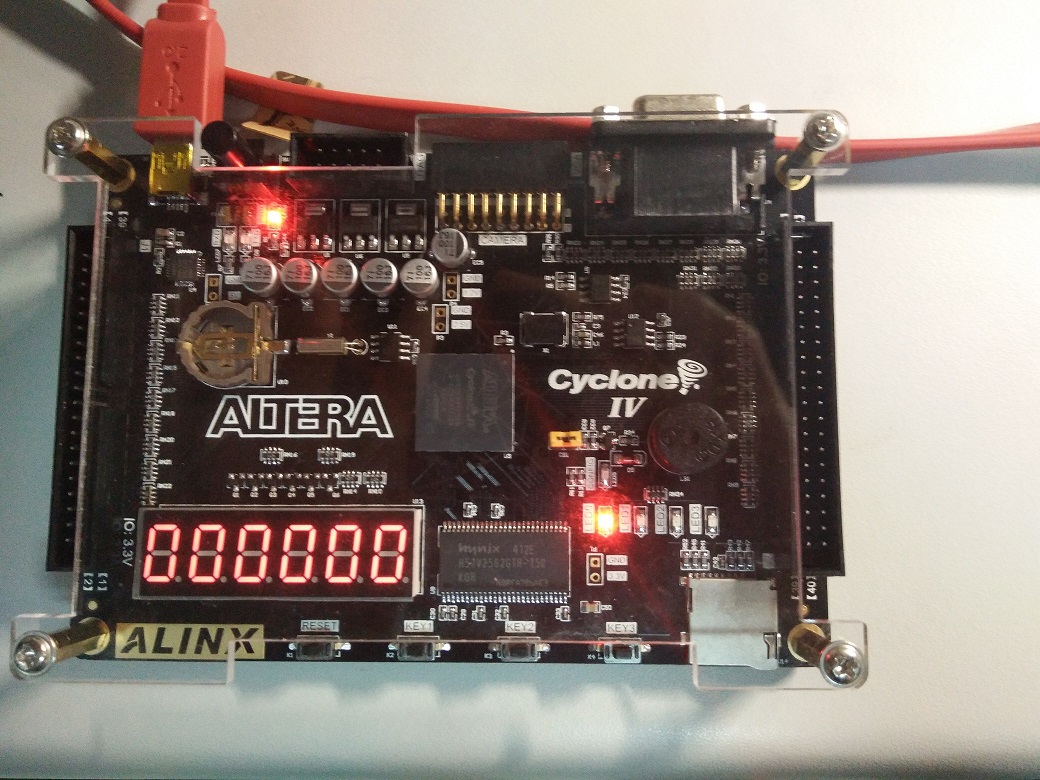


图4.1 图4.2

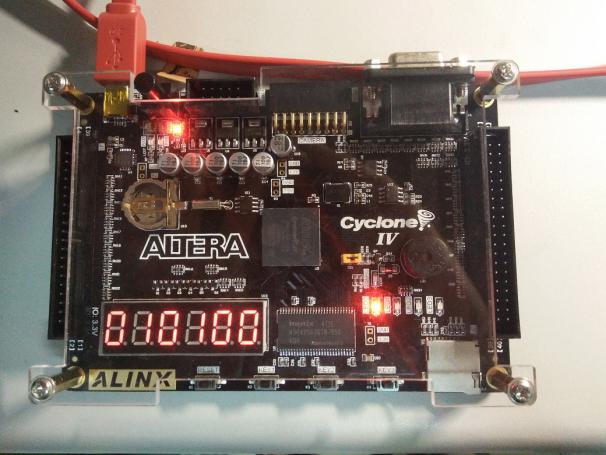


图4.3 图4.4

图4.1,4.2,4.3代表0:0,1:0,1:1号单元数据为零

图4.4代表设置读入1号预制程序,那么接下来0段应该有跳转指令序列,1段有默认加法指令序列

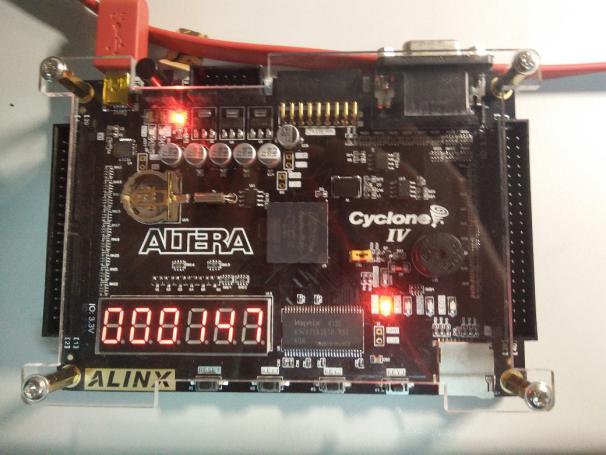
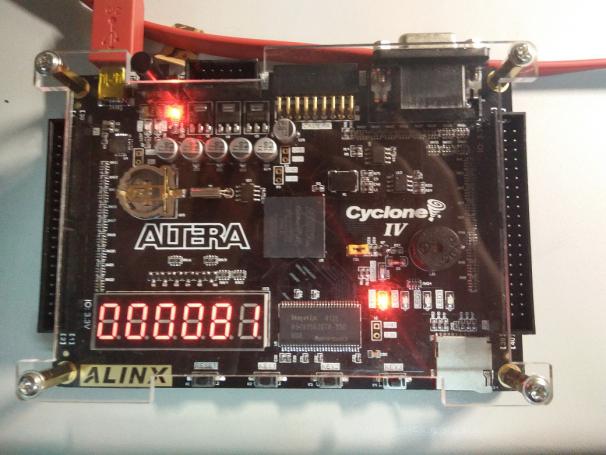


图4.5 图4.6

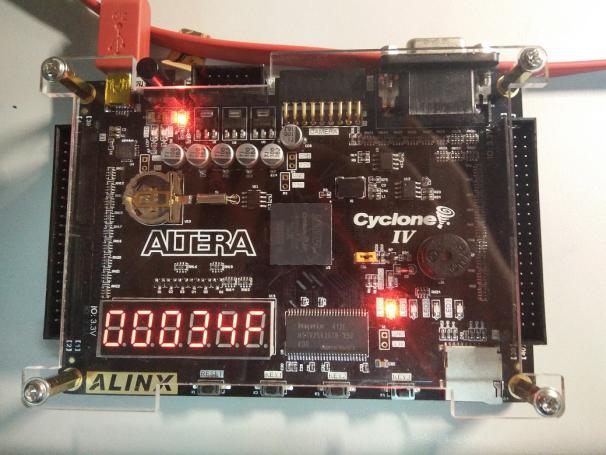
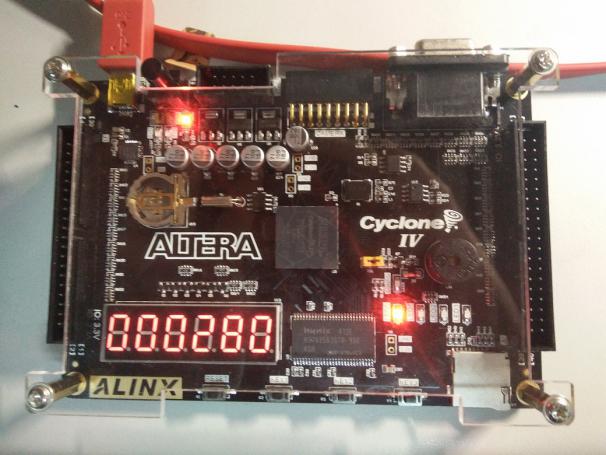


图4.7 图4.8

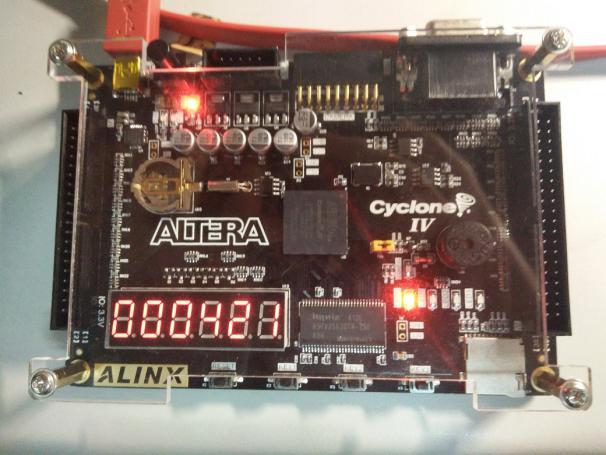


图4.9

图4.5~4.9表明了0段的前5条指令分别是"mov ix,1","mov ax,ix","mov ix,0","mov px,ix","jmp ax:px",也就是跳到1段的0号单元执行.恰巧就是加法程序的起始位置.

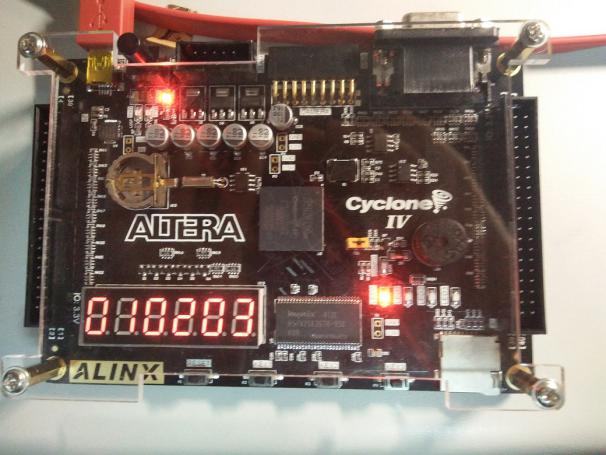
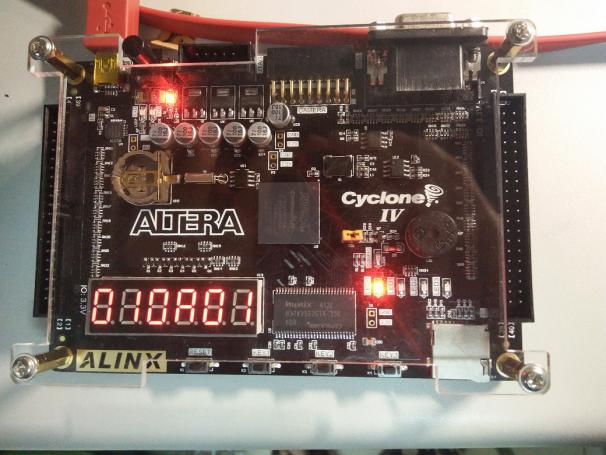


图4.10 图4.11

图4.10 表明01:0A单元是00000001,也就是add ax,px说明这段程序就是ax+px的演示程序.

图4.11 展示的是结果:1+2=3

# 第5章 结论

这次的课程设计让我受益匪浅,首先掌握了硬件设计语言VHDL,其次是知道了如何在开发板上进行硬件开发,然后是知晓了计算机的工作过程,最后是提高了自己的debug能力,感觉非常有收获.

由于没有学过计算机组成原理,所以这次的8位模型计算机显得"与众不同",没有pc也没有μIR,IR等一系列器件.我清楚的明白我不懂为什么要有那些器件,但是我知道汇编语言的工作过程,指令就在CS:IP指令地址组控制的单元,每次读取一条指令执行后改变CS:IP的值.

但是做模型机还是很麻烦,组件很多,控制器很复杂,在做的过程中采用了二级控制器分现在的中央控制器为主控制器和从控制器,这样的话两个控制器体积和功能均能下降但是由于分开来了,指令的执行周期大大增加(相比于现在的架构),最后忍痛推翻重新写,合并了两个控制器,就成了现在所看到的中央控制器.

控制器大了有利有弊,好处就是给外围元件发控制信号只需要一级就能连上,也就是只要等待半个CP就能有响应,下一个节拍马上就可以进行处理,但是相应的写的东西就更复杂了.

指令周期短了很有好处,第一是能够减少确认指令有效性的仿真时间,这个很重要,因为这里只有一种debug方式,那就是仿真.有的时候需要仿真十几条语句才能够全方面的验证一个指令是完全正确的.那么就得等十几分钟,但是如果缩短了的话,时间也能缩短很多,只要等几分钟就可以了.第二是能够加快仿真完后波形图的放大/缩小和移动速度,不是我乱说,如果一个仿真完的波形图是100μs的那么拖动,放大,缩小很痛苦,因为响应时间太长了.如果仿真时间短且波形图不大,放大缩小就很快很流畅,提高了debug的体验.

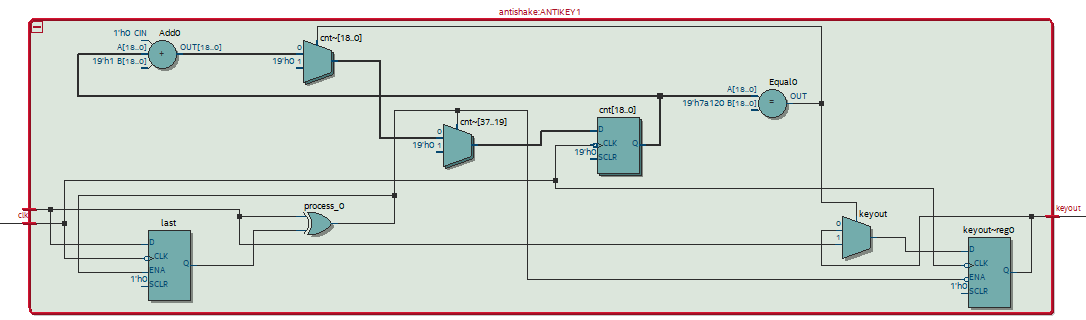
虽然在设计的过程中遇到了很多问题,自己都不是能很好的解决.但是通过同学和老师的帮助和指导完美的解决了我的问题.虽然在整个设计不是很完善,但是总的来说这次课程设计还是很成功的,我从中获得了不少,学习了将理论与实际相结合,既检验了知识还锻炼了思维和设计操作能力.在此次的课程设计中,我要感谢我的同学和老师,多亏你们不辞辛苦指导,才让我顺利的完成了这次课程设计,得到了如此多的收获.

# 参考文献

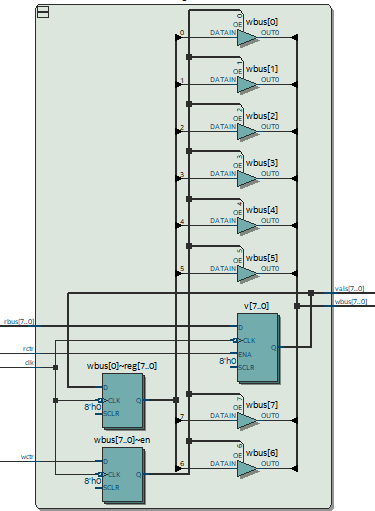
1. 黄任. VHDL入门.解惑.经典实例.经验总结(M) .北京：北京航空航天出版社,2005.

# 附录A RTL图

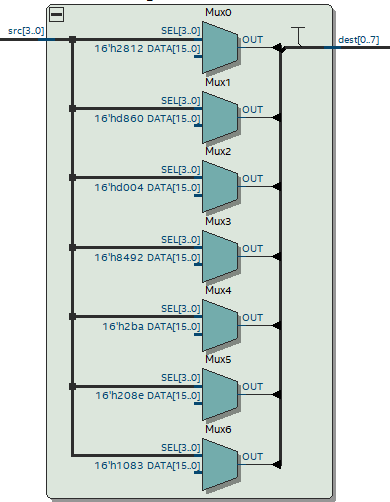
## A1. 按键防抖单元的RTL图



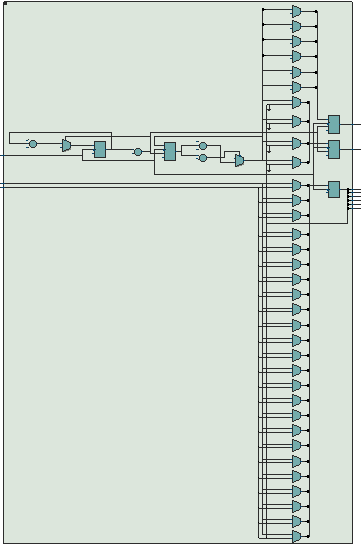
## A2. 数据寄存器的RTL图



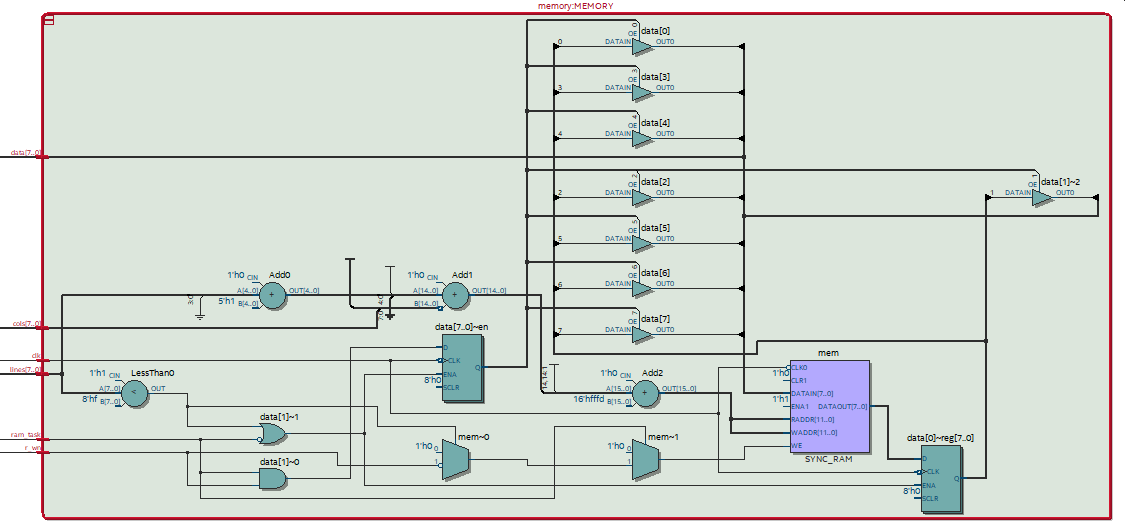
## A3. 48译码器RTL图



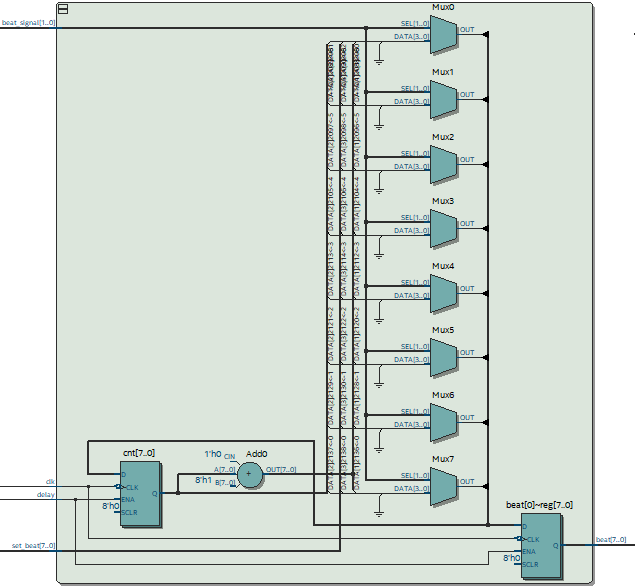
## A4. 数码管扫描器RTL图



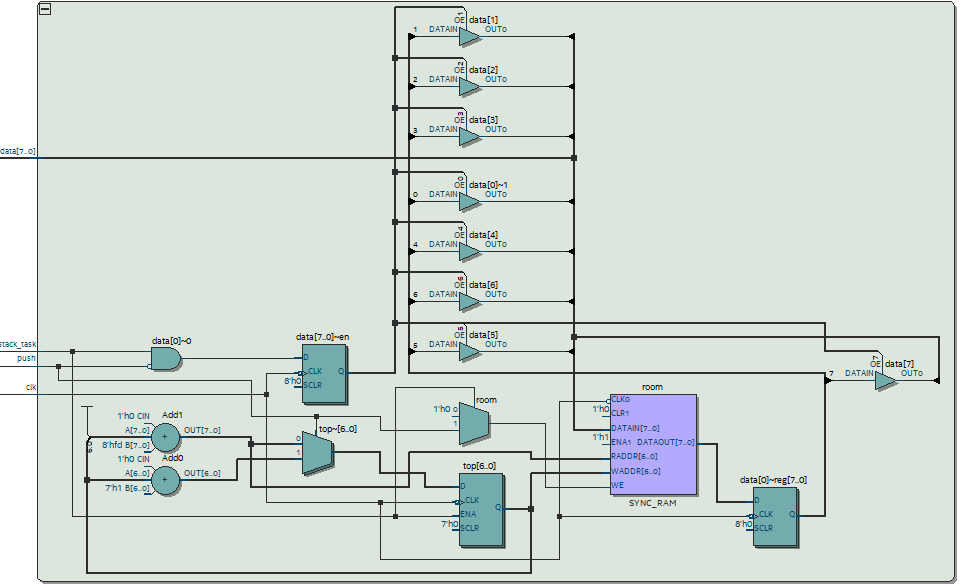
## A5. 随机访问存储器



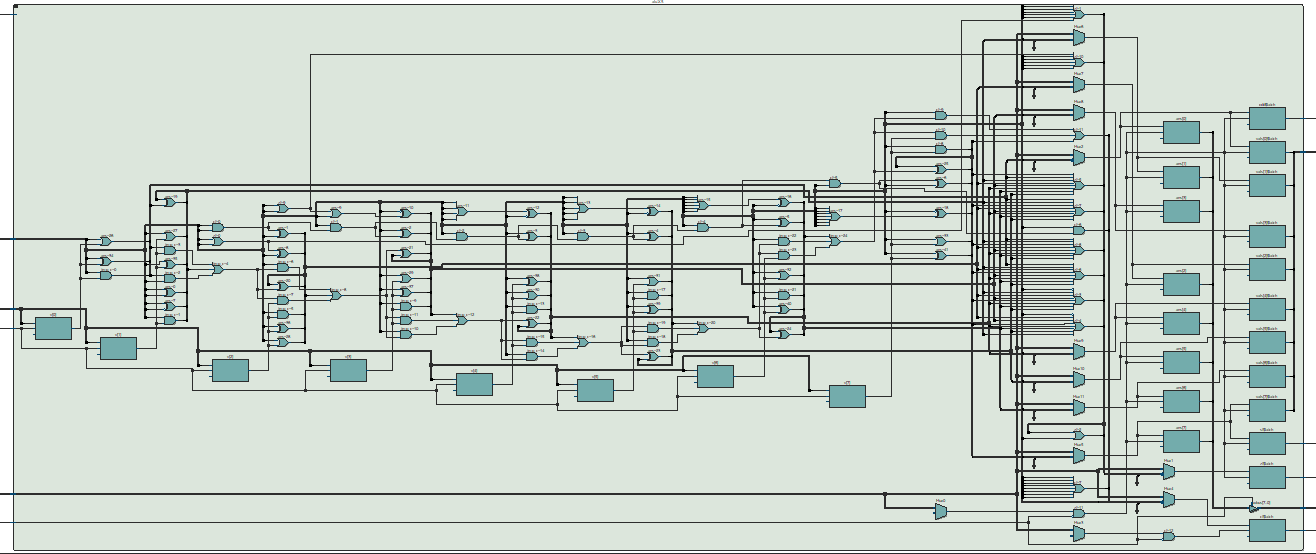
## A6. 节拍器RTL图



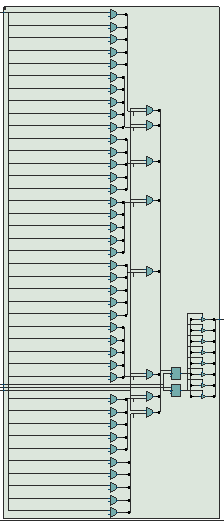
## A7. 栈存储器RTL图



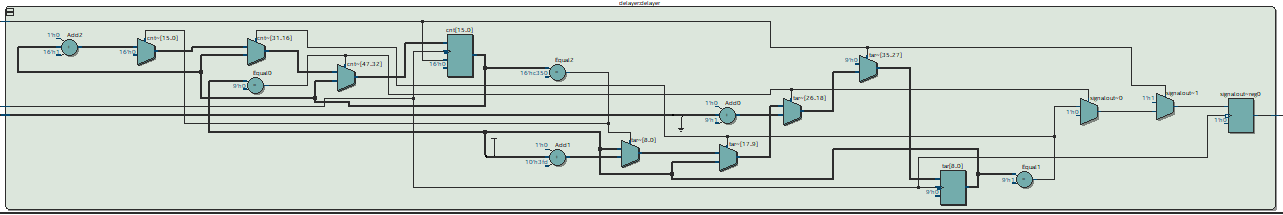
## A8. 算术逻辑单元RTL图



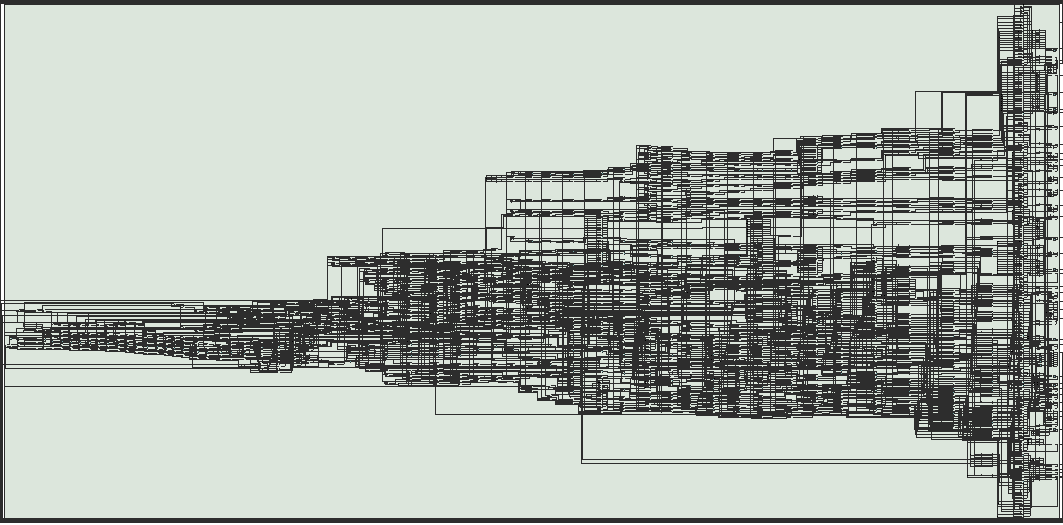
## A9. 只读存储器RTL图



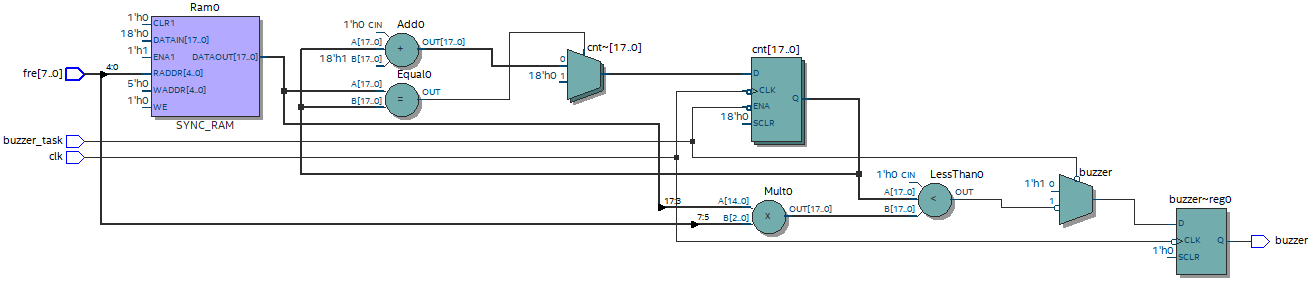
## A10. 延时器RTL图



## A11. 中央控制器RTL图



## A12. 蜂鸣器控制器RTL图



# 附录B VHDL程序清单

## B1. AntiShake.vhd

library ieee;

use ieee.std\_logic\_1164.all;

entity antishake is

port(

clk:in std\_logic;

keyin:in std\_logic;

keyout:out std\_logic

);

end;

architecture bhv of antishake is

signal last:std\_logic;

signal cnt:integer range 0 to 500000;

begin

process(clk)begin

if(falling\_edge(clk))then

if(last/=keyin)then

last<=keyin;

cnt<=0;

else

if(cnt=500000)then

keyout<=keyin;

cnt<=0;

else

cnt<=cnt+1;

end if;

end if;

end if;

end process;

end;

## B2. Led\_Decoder.vhd

library ieee;

use ieee.std\_logic\_1164.all;

entity led\_decoder is

port(

src:in std\_logic\_vector(3 downto 0);

dest:out std\_logic\_vector(0 to 7)

);

end;

architecture decode of led\_decoder is begin

process(src)begin

case src is

when"0000"=>dest<="00000011";

when"0001"=>dest<="10011111";

when"0010"=>dest<="00100101";

when"0011"=>dest<="00001101";

when"0100"=>dest<="10011001";

when"0101"=>dest<="01001001";

when"0110"=>dest<="01000001";

when"0111"=>dest<="00011111";

when"1000"=>dest<="00000001";

when"1001"=>dest<="00001001";

when"1010"=>dest<="00010001";

when"1011"=>dest<="11000001";

when"1100"=>dest<="01100011";

when"1101"=>dest<="10000101";

when"1110"=>dest<="01100001";

when"1111"=>dest<="01110001";

end case;

end process;

end;

## B3. dreg.vhd

library ieee;

use ieee.std\_logic\_1164.all;

entity dreg is

port(

clk:in std\_logic;

rctr:in std\_logic;

wctr:in std\_logic;

rbus:in std\_logic\_vector(7 downto 0);

wbus:out std\_logic\_vector(7 downto 0);

vals:out std\_logic\_vector(7 downto 0)

);

end;

architecture bhv of dreg is

signal v:std\_logic\_vector(7 downto 0);

begin

vals<=v;

process(clk)begin

if(falling\_edge(clk))then

if(rctr='1')then

v<=rbus;

end if;

if(wctr='1')then

wbus<=v;

else

wbus<=(others=>'Z');

end if;

end if;

end process;

end;

## B4. alu.vhd

library ieee;

use ieee.std\_logic\_1164.all;

entity alu is

port(

clk:in std\_logic;

xf:in std\_logic\_vector(7 downto 0);

rbus:in std\_logic\_vector(7 downto 0);

rctr:in std\_logic;

wctr:in std\_logic;

s:in std\_logic\_vector(0 to 3);

wbus:out std\_logic\_vector(7 downto 0);

cf,zf,sf,odd:out std\_logic;

vals:out std\_logic\_vector(7 downto 0)

);

end;

architecture bhv of alu is

begin

process(rctr,rbus,wctr,xf,s)

variable ans,v:std\_logic\_vector(7 downto 0);

variable tmp\_c:std\_logic\_vector(6 downto 0);

begin

if(rctr='1')then

v:=rbus;

end if;

if(wctr='1')then

wbus<=ans;

else

wbus<="ZZZZZZZZ";

case s is

when"1111"=>

ans:=xf;

vals<=ans;

sf<=ans(7);

zf<=not(ans(0)or ans(1)or ans(2)or ans(3)or ans(4)or ans(5)or ans(6)or ans(7));

odd<=ans(0);

when"0000"=> -->inc

ans(0):=xf(0) xor '1';

ans(1):=xf(1) xor xf(0);

ans(2):=xf(2) xor (xf(0) and xf(1));

ans(3):=xf(3) xor (xf(0) and xf(1) and xf(2));

ans(4):=xf(4) xor (xf(0) and xf(1) and xf(2) and xf(3));

ans(5):=xf(5) xor (xf(0) and xf(1) and xf(2) and xf(3) and xf(4));

ans(6):=xf(6) xor (xf(0) and xf(1) and xf(2) and xf(3) and xf(4) and xf(5));

ans(7):=xf(7) xor (xf(0) and xf(1) and xf(2) and xf(3) and xf(4) and xf(5) and xf(6));

cf<=xf(0) and xf(1) and xf(2) and xf(3) and xf(4) and xf(5) and xf(6) and xf(7);

vals<=ans;

sf<=ans(7);

zf<=not(ans(0)or ans(1)or ans(2)or ans(3)or ans(4)or ans(5)or ans(6)or ans(7));

odd<=ans(0);

when"0001"=> -->dec

ans(0):=xf(0) xor '1';

ans(1):=xf(1) xor '1' xor xf(0);

ans(2):=xf(2) xor '1' xor (xf(1)or xf(0));

ans(3):=xf(3) xor '1' xor (xf(2)or xf(1)or xf(0));

ans(4):=xf(4) xor '1' xor (xf(3)or xf(2)or xf(1)or xf(0));

ans(5):=xf(5) xor '1' xor (xf(4)or xf(3)or xf(2)or xf(1)or xf(0));

ans(6):=xf(6) xor '1' xor (xf(5)or xf(4)or xf(3)or xf(2)or xf(1)or xf(0));

ans(7):=xf(7) xor '1' xor (xf(6)or xf(5)or xf(4)or xf(3)or xf(2)or xf(1)or xf(0));

cf<=not(xf(7)or xf(6)or xf(5)or xf(4)or xf(3)or xf(2)or xf(1)or xf(0));

vals<=ans;

sf<=ans(7);

zf<=not(ans(0)or ans(1)or ans(2)or ans(3)or ans(4)or ans(5)or ans(6)or ans(7));

odd<=ans(0);

when"0010"=> -->add

tmp\_c(0):=xf(0) and v(0);

tmp\_c(1):=(xf(1) and v(1))or(tmp\_c(0) and xf(1))or(tmp\_c(0) and v(1));

tmp\_c(2):=(xf(2) and v(2))or(tmp\_c(1) and xf(2))or(tmp\_c(1) and v(2));

tmp\_c(3):=(xf(3) and v(3))or(tmp\_c(2) and xf(3))or(tmp\_c(2) and v(3));

tmp\_c(4):=(xf(4) and v(4))or(tmp\_c(3) and xf(4))or(tmp\_c(3) and v(4));

tmp\_c(5):=(xf(5) and v(5))or(tmp\_c(4) and xf(5))or(tmp\_c(4) and v(5));

tmp\_c(6):=(xf(6) and v(6))or(tmp\_c(5) and xf(6))or(tmp\_c(5) and v(6));

ans(0):=xf(0) xor v(0);

ans(1):=xf(1) xor v(1) xor tmp\_c(0);

ans(2):=xf(2) xor v(2) xor tmp\_c(1);

ans(3):=xf(3) xor v(3) xor tmp\_c(2);

ans(4):=xf(4) xor v(4) xor tmp\_c(3);

ans(5):=xf(5) xor v(5) xor tmp\_c(4);

ans(6):=xf(6) xor v(6) xor tmp\_c(5);

ans(7):=xf(7) xor v(7) xor tmp\_c(6);

cf<=(xf(7) and v(7))or(tmp\_c(6) and xf(7))or(tmp\_c(6) and v(7));

vals<=ans;

sf<=ans(7);

zf<=not(ans(0)or ans(1)or ans(2)or ans(3)or ans(4)or ans(5)or ans(6)or ans(7));

odd<=ans(0);

when"0011"=> -->or

ans:=xf or v;

vals<=ans;

sf<=ans(7);

zf<=not(ans(0)or ans(1)or ans(2)or ans(3)or ans(4)or ans(5)or ans(6)or ans(7));

odd<=ans(0);

when"1000"=> -->xor

ans:=xf xor v;

vals<=ans;

sf<=ans(7);

zf<=not(ans(0)or ans(1)or ans(2)or ans(3)or ans(4)or ans(5)or ans(6)or ans(7));

odd<=ans(0);

when"1001"=> -->and

ans:=xf and v;

vals<=ans;

sf<=ans(7);

zf<=not(ans(0)or ans(1)or ans(2)or ans(3)or ans(4)or ans(5)or ans(6)or ans(7));

odd<=ans(0);

when"1010"=> -->shl

ans:=xf(6 downto 0)&'0';

cf<=xf(7);

vals<=ans;

sf<=ans(7);

zf<=not(ans(0)or ans(1)or ans(2)or ans(3)or ans(4)or ans(5)or ans(6)or ans(7));

odd<=ans(0);

when"1011"=> -->shr

ans:='0'&xf(7 downto 1);

cf<=xf(0);

vals<=ans;

sf<=ans(7);

zf<=not(ans(0)or ans(1)or ans(2)or ans(3)or ans(4)or ans(5)or ans(6)or ans(7));

odd<=ans(0);

when others=>

end case;

end if;

end process;

end;

## B5. beater.vhd

library ieee;

use ieee.std\_logic\_1164.all;

entity beater is

port(

clk:in std\_logic;

beat\_signal:in std\_logic\_vector(1 downto 0);

set\_beat:in integer range 0 to 255;

beat:out integer range 0 to 255;

delay:in std\_logic

);

end;

architecture bhv of beater is

begin

process(clk,delay)

variable cnt:integer range 0 to 255;

begin

if(falling\_edge(clk) and delay='1')then

case beat\_signal is

when"00"=>cnt:=0;

when"01"=>cnt:=cnt+1;

when"10"=>cnt:=cnt;

when"11"=>cnt:=set\_beat;

end case;

beat<=cnt;

end if;

end process;

end;

## B6. buzzer\_controller.vhd

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity buzzer\_controller is

port(

clk:in std\_logic;

buzzer\_task:in std\_logic;

buzzer:out std\_logic;

fre:in std\_logic\_vector(7 downto 0)

);

end;

architecture bhv of buzzer\_controller is

signal cnt:integer range 0 to 200000;

signal target:integer range 0 to 200000;

signal volumn:integer range 0 to 7;

begin

process(clk)begin

if(falling\_edge(clk))then

if(buzzer\_task='0')then

if(cnt<target/8\*volumn)then

buzzer<='0';

else

buzzer<='1';

end if;

if(cnt=target)then

cnt<=0;

else

cnt<=cnt+1;

end if;

else

buzzer<='1';

end if;

end if;

end process;

process(fre)begin

volumn<=conv\_integer(fre(7 downto 5));

case conv\_integer(fre(4 downto 0)) is

when 00=>target<=50000000/261;

when 01=>target<=50000000/293;

when 02=>target<=50000000/329;

when 03=>target<=50000000/349;

when 04=>target<=50000000/392;

when 05=>target<=50000000/440;

when 06=>target<=50000000/499;

when 07=>target<=50000000/523;

when 08=>target<=50000000/587;

when 09=>target<=50000000/659;

when 10=>target<=50000000/698;

when 11=>target<=50000000/784;

when 12=>target<=50000000/880;

when 13=>target<=50000000/998;

when 14=>target<=50000000/1046;

when 15=>target<=50000000/1174;

when 16=>target<=50000000/1318;

when 17=>target<=50000000/1396;

when 18=>target<=50000000/1568;

when 19=>target<=50000000/1760;

when 20=>target<=50000000/1976;

when 21=>target<=50000000/2093;

when 22=>target<=50000000/2349;

when 23=>target<=50000000/2637;

when 24=>target<=50000000/2794;

when 25=>target<=50000000/3136;

when 26=>target<=50000000/3520;

when 27=>target<=50000000/3951;

when others=>target<=0;

end case;

end process;

end;

## B7. controller.vhd

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity controller is

port(

rom:out std\_logic;

romL,romC:out std\_logic\_vector(7 downto 0);

ram,ramrw:out std\_logic;

ramL,ramC:out std\_logic\_vector(7 downto 0);

stack,push:out std\_logic;

s:out std\_logic\_vector(3 downto 0);

cf,zf,sf,odd:in std\_logic;

key0,key1,key2,key3:in std\_logic;

state:buffer integer range 0 to 3;

ins:out std\_logic\_vector(7 downto 0);

clk:in std\_logic;

beat:in integer range 0 to 255;

jmp\_flag:buffer std\_logic;

rax,wax,

rpx,wpx,

rxf,wxf,

rxs,wxs,

rds,wds,

rdx,wdx,

rcs,wcs,

rip,wip,

rsi,wsi,

rix,wix,

rsx,wsx

:out std\_logic;

id:out integer range 0 to 7;

num:out std\_logic\_vector(3 downto 0);

beat\_signal:out std\_logic\_vector(1 downto 0);

set\_beat:out integer range 0 to 255;

rbus:in std\_logic\_vector(7 downto 0);

wbus:out std\_logic\_vector(7 downto 0);

rommode:buffer std\_logic;

led:out std\_logic\_vector(2 downto 0);

delay:out std\_logic;

buzzer:buffer std\_logic

);

end;

architecture controll of controller is

constant INIT:integer range 0 to 3:=0;

constant LOAD:integer range 0 to 3:=1;

constant EXEC:integer range 0 to 3:=2;

constant OVER:integer range 0 to 3:=3;

begin

process(clk)

variable cs,ip, --load

ds,si, --stos|lods

instruction --global

:std\_logic\_vector(7 downto 0);

variable cs\_flag, --over

fcf,fsf,fzf,fodd, --flag

last0,last1,last2,last3 --wait

:std\_logic;

variable key\_pressed --wait

:integer range 0 to 4;

begin

if(rising\_edge(clk))then

case state is

when INIT=>

case beat is

when 00=>

buzzer<='1';

beat\_signal<="01";s<="1111";

rommode<='1';delay<='0';

fcf:='0';fsf:='0';fzf:='0';fodd:='0';cs\_flag:='0';id<=7;

last0:=key0;last1:=key1;last2:=key2;last3:=key3;key\_pressed:=4;

rom<='0';ram<='0';stack<='0';

when 01=>wax<='0';wpx<='0';wds<='0';wdx<='0';wcs<='0';wip<='0';wsi<='0';wix<='0';wxf<='0';wxs<='0';wsx<='0';

when 02=>rax<='1';rpx<='1';rds<='1';rdx<='1';rcs<='1';rip<='1';rsi<='1';rix<='1';rxf<='1';rxs<='1';rsx<='1';

wbus<="00000000";

when 03=>rax<='0';rpx<='0';rds<='0';rdx<='0';rcs<='0';rip<='0';rsi<='0';rix<='0';rxf<='0';rxs<='0';rsx<='0';

when 04=>state<=LOAD;wbus<="ZZZZZZZZ";beat\_signal<="00";

when others=>state<=LOAD;wbus<="ZZZZZZZZ";beat\_signal<="00";

end case;

when LOAD=>

case beat is -->load instruction

when 00=>

wax<='0';wpx<='0';wds<='0';wdx<='0';wcs<='0'; wsi<='0';wix<='0';wxf<='0';wxs<='0';wsx<='0';

rax<='0';rpx<='0';rds<='0';rdx<='0';rcs<='0';rip<='0';rsi<='0';rix<='0';rxf<='0';rxs<='0';rsx<='0';

rom<='0';ram<='0';stack<='0';s<="1111";id<=7;

beat\_signal<="01";

jmp\_flag<='0';

cs\_flag:='0';

wip<='1';

when 01=>ip:=rbus;

wip<='0';

wcs<='1';

when 02=>cs:=rbus;

wcs<='0';

if(rommode='1')then

rom<='1';romL<=cs;romC<=ip;

else

ram<='1';ramL<=cs;ramC<=ip;ramrw<='1';

end if;

when 03=>instruction:=rbus;ins<=rbus;

rom<='0';ram<='0';

beat\_signal<="00";

state<=EXEC;

when others=>beat\_signal<="00";state<=EXEC;

end case;

when EXEC=>

if(instruction(7)='1')then -->mov ix,@i7<

case beat is

when 00=>beat\_signal<="01";rix<='1';wbus<='0'&instruction(6 downto 0);

when 01=>rix<='0';state<=OVER;beat\_signal<="00";wbus<="ZZZZZZZZ";

when others=>state<=OVER;beat\_signal<="00";wbus<="ZZZZZZZZ";

end case;

elsif(instruction(7 downto 8-2)="01")then -->mov @r,@r<

case beat is

when 00=>beat\_signal<="01";

case instruction(2 downto 0) is

when"000"=>wax<='1';

when"001"=>wpx<='1';

when"010"=>wds<='1';

when"011"=>wdx<='1';

when"100"=>wcs<='1';

when"101"=>wip<='1';

when"110"=>wsi<='1';

when"111"=>wix<='1';

end case;

when 01=>

case instruction(5 downto 3) is

when"000"=>rax<='1';

when"001"=>rpx<='1';

when"010"=>rds<='1';

when"011"=>rdx<='1';

when"100"=>rcs<='1';

when"101"=>rip<='1';

when"110"=>rsi<='1';

when"111"=>rix<='1';

end case;

case instruction(2 downto 0) is

when"000"=>wax<='0';

when"001"=>wpx<='0';

when"010"=>wds<='0';

when"011"=>wdx<='0';

when"100"=>wcs<='0';

when"101"=>wip<='0';

when"110"=>wsi<='0';

when"111"=>wix<='0';

end case;

state<=OVER;

beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00100000")then -->jmp cs:px<

case beat is

when 00=>beat\_signal<="01";wpx<='1';

when 01=>rip<='1';jmp\_flag<='1';

state<=OVER;

beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00100001")then -->jmp ax:px<

case beat is

when 00=>beat\_signal<="01";wax<='1';

when 01=>rcs<='1';

when 02=>wax<='0';rcs<='0';wpx<='1';

when 03=>rip<='1';jmp\_flag<='1';

state<=OVER;

beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00100010")then -->jc cs:px<

case beat is

when 00=>beat\_signal<="01";

if(fcf='0')then

state<=OVER;beat\_signal<="00";

else

wpx<='1';

end if;

when 01=>rip<='1';jmp\_flag<='1';

state<=OVER;

beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00100011")then -->stos<

case beat is

when 00=>beat\_signal<="01";wds<='1';

when 01=>ds:=rbus;wds<='0';wsi<='1';

when 02=>si:=rbus;wsi<='0';wdx<='1';

when 03=>ram<='1';ramrw<='0';ramL<=ds;ramC<=si;

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00100100")then -->jz cs:px<

case beat is

when 00=>beat\_signal<="01";

if(fzf='0')then

state<=OVER;beat\_signal<="00";

else

wpx<='1';

end if;

when 01=>rip<='1';jmp\_flag<='1';

state<=OVER;

beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00100101")then -->j<

case beat is

when 00=>beat\_signal<="01";

stack<='1';push<='0';

when 01=>rxf<='1';stack<='0';

when 02=>fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00100110")then -->js cs:px<

case beat is

when 00=>beat\_signal<="01";

if(fsf='0')then

state<=OVER;beat\_signal<="00";

else

wpx<='1';

end if;

when 01=>rip<='1';jmp\_flag<='1';

state<=OVER;

beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00100111")then -->jodd cs:px

case beat is

when 00=>beat\_signal<="01";

if(fodd='0')then

state<=OVER;beat\_signal<="00";

else

wpx<='1';

end if;

when 01=>rip<='1';jmp\_flag<='1';

state<=OVER;

beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction(7 downto 8-5)="00101")then -->push @r<

case beat is

when 00=>beat\_signal<="01";

case instruction(2 downto 0) is

when"000"=>wax<='1';

when"001"=>wpx<='1';

when"010"=>wds<='1';

when"011"=>wdx<='1';

when"100"=>wcs<='1';

when"101"=>wip<='1';

when"110"=>wsi<='1';

when"111"=>wix<='1';

end case;

when 01=>stack<='1';push<='1';

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction(7 downto 8-5)="00110")then -->pop @r<

case beat is

when 00=>beat\_signal<="01";

stack<='1';push<='0';

when 01=>

stack<='0';push<='0';

case instruction(2 downto 0) is

when"000"=>rax<='1';

when"001"=>rpx<='1';

when"010"=>rds<='1';

when"011"=>rdx<='1';

when"100"=>rcs<='1';

when"101"=>rip<='1';

when"110"=>rsi<='1';

when"111"=>rix<='1';

end case;

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction(7 downto 8-5)="00111")then -->inc @r<

case beat is

when 00=>beat\_signal<="01";

case instruction(2 downto 0) is

when"000"=>wax<='1';

when"001"=>wpx<='1';

when"010"=>wds<='1';

when"011"=>wdx<='1';

when"100"=>wcs<='1';

when"101"=>wip<='1';

when"110"=>wsi<='1';

when"111"=>wix<='1';

end case;

when 01=>rxf<='1';s<="0000";

when 02=>rxf<='0';

case instruction(2 downto 0) is

when"000"=>wax<='0';

when"001"=>wpx<='0';

when"010"=>wds<='0';

when"011"=>wdx<='0';

when"100"=>wcs<='0';

when"101"=>wip<='0';

when"110"=>wsi<='0';

when"111"=>wix<='0';

end case;

wxs<='1';

when 03=>fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;

case instruction(2 downto 0) is

when"000"=>rax<='1';

when"001"=>rpx<='1';

when"010"=>rds<='1';

when"011"=>rdx<='1';

when"100"=>rcs<='1';

when"101"=>rip<='1';

when"110"=>rsi<='1';

when"111"=>rix<='1';

end case;

when 04=>

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction(7 downto 8-5)="00011")then -->dec @r<

case beat is

when 00=>beat\_signal<="01";

case instruction(2 downto 0) is

when"000"=>wax<='1';

when"001"=>wpx<='1';

when"010"=>wds<='1';

when"011"=>wdx<='1';

when"100"=>wcs<='1';

when"101"=>wip<='1';

when"110"=>wsi<='1';

when"111"=>wix<='1';

end case;

when 01=>rxf<='1';s<="0001";

when 02=>rxf<='0';

case instruction(2 downto 0) is

when"000"=>wax<='0';

when"001"=>wpx<='0';

when"010"=>wds<='0';

when"011"=>wdx<='0';

when"100"=>wcs<='0';

when"101"=>wip<='0';

when"110"=>wsi<='0';

when"111"=>wix<='0';

end case;

wxs<='1';

when 03=>fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;

case instruction(2 downto 0) is

when"000"=>rax<='1';

when"001"=>rpx<='1';

when"010"=>rds<='1';

when"011"=>rdx<='1';

when"100"=>rcs<='1';

when"101"=>rip<='1';

when"110"=>rsi<='1';

when"111"=>rix<='1';

end case;

when 04=>

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction(7 downto 8-5)="00010")then -->out @o<

case beat is

when 00=>beat\_signal<="01";wdx<='1';

when 01=>

if(instruction(2 downto 0)="110")then

led<=rbus(2 downto 0);

elsif(instruction(2 downto 0)="111")then

buzzer<=not buzzer;

else

if(instruction(0)='0')then

num<=rbus(3 downto 0);

else

num<=rbus(7 downto 4);end if;

id<=conv\_integer(instruction(2 downto 0));

end if;

when 02=>

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00001111")then -->xor ax,px<

case beat is

when 00=>beat\_signal<="01";wax<='1';s<="1000";

when 01=>rxf<='1';

when 02=>rxf<='0';wax<='0';wpx<='1';

when 03=>rxs<='1';

when 04=>rxs<='0';wpx<='0';wxs<='1';fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;

when 05=>rax<='1';

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00001110")then -->mode<

wbus<="00000000";rcs<='1';rip<='1';rommode<=not rommode;jmp\_flag<='1';state<=OVER;beat\_signal<="00";

elsif(instruction="00001101")then -->shl ax,1<

case beat is

when 00=>beat\_signal<="01";wax<='1';s<="1010";

when 01=>rxf<='1';

when 02=>rxf<='0';wax<='0';wxs<='1';

when 03=>fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;rax<='1';

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00001100")then -->or ax,px<

case beat is

when 00=>beat\_signal<="01";wax<='1';s<="0011";

when 01=>rxf<='1';

when 02=>rxf<='0';wax<='0';wpx<='1';

when 03=>rxs<='1';

when 04=>rxs<='0';wpx<='0';wxs<='1';fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;

when 05=>rax<='1';

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00001011")then -->lods<

case beat is

when 00=>beat\_signal<="01";wds<='1';

when 01=>ds:=rbus;wds<='0';wsi<='1';

when 02=>si:=rbus;wsi<='0';

when 03=>ram<='1';ramrw<='1';ramL<=ds;ramC<=si;

when 04=>rdx<='1';ram<='0';

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00001010")then -->shr ax,1<

case beat is

when 00=>beat\_signal<="01";wax<='1';s<="1011";

when 01=>rxf<='1';

when 02=>rxf<='0';wax<='0';wxs<='1';

when 03=>fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;rax<='1';

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00001001")then -->wait<

if(key0/=last0 or key\_pressed=0)then

jmp\_flag<='0';

last0:=key0;

key\_pressed:=0;

case beat is

when 00=>beat\_signal<="01";

wsx<='1';s<="1000";

when 01=>rxf<='1';

when 02=>rxf<='0';wsx<='0';wbus<="00000001";rxs<='1';

when 03=>rxs<='0';wxs<='1';fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;wbus<="ZZZZZZZZ";

when 04=>rsx<='1';

state<=OVER;beat\_signal<="00";

key\_pressed:=4;

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(key1/=last1 or key\_pressed=1)then

jmp\_flag<='0';

last1:=key1;

key\_pressed:=1;

case beat is

when 00=>beat\_signal<="01";

wsx<='1';s<="1000";

when 01=>rxf<='1';

when 02=>rxf<='0';wsx<='0';wbus<="00000010";rxs<='1';

when 03=>rxs<='0';wxs<='1';fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;wbus<="ZZZZZZZZ";

when 04=>rsx<='1';

state<=OVER;beat\_signal<="00";

key\_pressed:=4;

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(key2/=last2 or key\_pressed=2)then

jmp\_flag<='0';

last2:=key2;

key\_pressed:=2;

case beat is

when 00=>beat\_signal<="01";

wsx<='1';s<="1000";

when 01=>rxf<='1';

when 02=>rxf<='0';wsx<='0';wbus<="00000100";rxs<='1';

when 03=>rxs<='0';wxs<='1';fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;wbus<="ZZZZZZZZ";

when 04=>rsx<='1';

state<=OVER;beat\_signal<="00";

key\_pressed:=4;

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(key3/=last3 or key\_pressed=3)then

jmp\_flag<='0';

last3:=key3;

key\_pressed:=3;

case beat is

when 00=>beat\_signal<="01";

wsx<='1';s<="1000";

when 01=>rxf<='1';

when 02=>rxf<='0';wsx<='0';wbus<="00001000";rxs<='1';

when 03=>rxs<='0';wxs<='1';fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;wbus<="ZZZZZZZZ";

when 04=>rsx<='1';

state<=OVER;beat\_signal<="00";

key\_pressed:=4;

when others=>state<=OVER;beat\_signal<="00";

end case;

else

jmp\_flag<='1';

key\_pressed:=4;

end if;

elsif(instruction="00001000")then -->and ax,px<

case beat is

when 00=>beat\_signal<="01";wax<='1';s<="1001";

when 01=>rxf<='1';

when 02=>rxf<='0';wax<='0';wpx<='1';

when 03=>rxs<='1';

when 04=>rxs<='0';wpx<='0';wxs<='1';fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;

when 05=>rax<='1';

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction(7 downto 8-6)="000001")then -->in ss<

case beat is

when 00=>beat\_signal<="01";wsx<='1';s<="1001";

when 01=>rxf<='1';

when 02=>rxf<='0';wsx<='0';rxs<='1';

case instruction(1 downto 0) is

when"00"=>wbus<="00000001";

when"01"=>wbus<="00000010";

when"10"=>wbus<="00000100";

when"11"=>wbus<="00001000";

end case;

when 03=>state<=OVER;beat\_signal<="00";

fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00000011")then -->reset

rsx<='1';wbus<="11111111";state<=OVER;beat\_signal<="00";

elsif(instruction="00000010")then -->jz ax:px<

case beat is

when 00=>beat\_signal<="01";

if(fzf='0')then

state<=OVER;beat\_signal<="00";

else

wax<='1';

end if;

when 01=>rcs<='1';

when 02=>wax<='0';rcs<='0';wpx<='1';

when 03=>rip<='1';jmp\_flag<='1';

state<=OVER;

beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00000001")then -->add ax,px<

case beat is

when 00=>beat\_signal<="01";wax<='1';s<="0010";

when 01=>rxf<='1';

when 02=>rxf<='0';wax<='0';wpx<='1';

when 03=>rxs<='1';

when 04=>rxs<='0';wpx<='0';wxs<='1';fcf:=cf;fzf:=zf;fsf:=sf;fodd:=odd;

when 05=>rax<='1';

state<=OVER;beat\_signal<="00";

when others=>state<=OVER;beat\_signal<="00";

end case;

elsif(instruction="00000000")then -->delay<

case beat is

when 00=>beat\_signal<="01";

when 01=>delay<='1';

when 02=>

when 03=>

when 04=>state<=OVER;beat\_signal<="00";delay<='0';

when others=>state<=OVER;beat\_signal<="00";

end case;

end if;

when OVER=>

if(jmp\_flag='0')then -->process cs,ip

case beat is

when 00=>beat\_signal<="01";wip<='1';

wax<='0';wpx<='0';wds<='0';wdx<='0';wcs<='0'; wsi<='0';wix<='0';wxf<='0';wxs<='0';wsx<='0';

rax<='0';rpx<='0';rds<='0';rdx<='0';rcs<='0';rip<='0';rsi<='0';rix<='0';rxf<='0';rxs<='0';rsx<='0';

rom<='0';ram<='0';stack<='0';wbus<="ZZZZZZZZ";s<="1111";id<=7;

when 01=>rxf<='1';s<="0000";

when 02=>wip<='0';rxf<='0';wxs<='1';

when 03=>cs\_flag:=cf;rip<='1';

if(cs\_flag='0')then

state<=LOAD;

beat\_signal<="00";

end if;

when 04=>rip<='0';wxs<='0';wcs<='1';

when 05=>rxf<='1';s<="0000";

when 06=>rxf<='0';wcs<='0';wxs<='1';

when 07=>rcs<='1';state<=LOAD;beat\_signal<="00";

when others=>state<=LOAD;beat\_signal<="00";

end case;

else

wax<='0';wpx<='0';wds<='0';wdx<='0';wcs<='0';wip<='0';wsi<='0';wix<='0';wxf<='0';wxs<='0';wsx<='0';

rax<='0';rpx<='0';rds<='0';rdx<='0';rcs<='0';rip<='0';rsi<='0';rix<='0';rxf<='0';rxs<='0';rsx<='0';

rom<='0';ram<='0';stack<='0';wbus<="ZZZZZZZZ";s<="1111";id<=7;

state<=LOAD;beat\_signal<="00";

end if;

end case;

end if;

end process;

end;

## B8. delayer.vhd

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity delayer is

port(

clk:in std\_logic;

dtime:in std\_logic\_vector(7 downto 0);

delayer\_task:in std\_logic;

signalout:out std\_logic

);

end;

architecture bhv of delayer is

signal cnt:integer range 0 to 50000;

signal tar:integer range 0 to 256;

begin

process(clk)begin

if(falling\_edge(clk))then

if(delayer\_task='1')then

if(tar=0)then

tar<=conv\_integer(dtime)+1;

signalout<='0';

elsif(tar=1)then

signalout<='1';

else

if(cnt=50000)then

cnt<=0;

tar<=tar-1;

else

cnt<=cnt+1;

end if;

signalout<='0';

end if;

else

tar<=0;

signalout<='1';

end if;

end if;

end process;

end;

## B9. memory.vhd

library ieee;

use ieee.std\_logic\_1164.all;

entity memory is

port(

clk:in std\_logic;

ram\_task,r\_wn:in std\_logic;

lines,cols:in integer range 0 to 255;

data:inout std\_logic\_vector(7 downto 0)

);

end;

architecture bhv of memory is

type bank is array(0 to 255)of std\_logic\_vector(7 downto 0);

type banks is array(0 to 15)of bank;

begin

process(clk)

variable mem:banks;

begin

if(falling\_edge(clk))then

if(ram\_task='1')then

if(lines<=15)then

if(r\_wn='1')then --read

data<=mem(lines)(cols);

else --write

mem(lines)(cols):=data;

data<=(others=>'Z');

end if;

end if;

else

data<=(others=>'Z');

end if;

end if;

end process;

end;

## B10. rom.vhd

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity rom is

port(

clk:in std\_logic;

rom\_task:in std\_logic;

lines,cols:in std\_logic\_vector(7 downto 0);

data:out std\_logic\_vector(7 downto 0)

);

end;

architecture bhv of rom is

type bank is array(0 to 255)of std\_logic\_vector(7 downto 0);

type banks is array(0 to 4)of bank;

begin

process(clk)

constant mem:banks:=((

"00001001",-->wait<

"00001001",-->wait<

"00001001",-->wait<

"00001001",-->wait<

"00000011",-->reset<

"10000000",-->mov ix,0<

"01010111",-->mov ds,ix<

"01110111",-->mov si,ix<

"01011111",-->mov dx,ix<

"00010101",-->out 5<

"00010100",-->out 4<

"00010011",-->out 3<

"00010010",-->out 2<

"00010001",-->out 1<

"00010000",-->out 0<

"00001001",-->wait<

"00000101",-->in 1<

"10011111",-->mov ix,31<

"01001111",-->mov px,ix<

"00100100",-->jz cs:px<

"00000110",-->in 2<

"11010100",-->mov ix,84<

"01001111",-->mov px,ix<

"00100100",-->jz cs:px<

"00000111",-->in 3<

"11101010",-->mov ix,106<

"01001111",-->mov px,ix<

"00100100",-->jz cs:px<

"10001111",-->mov ix,15<

"01001111",-->mov px,ix<

"00100000",-->jmp cs:px<

"00000110",-->in 2<

"10000001",-->mov ix,1<

"01000111",-->mov ax,ix<

"10000000",-->mov ix,0<

"01001111",-->mov px,ix<

"00000010",-->jz ax:px<

"00000111",-->in 3<

"11111000",-->mov ix,120<

"01001111",-->mov px,ix<

"00100100",-->jz cs:px<

"00000100",-->in 0<

"11000000",-->mov ix,64<

"01001111",-->mov px,ix<

"00100100",-->jz cs:px<

"00111010",-->inc ds<

"01000010",-->mov ax,ds<

"10001000",-->mov ix,8<

"01001111",-->mov px,ix<

"00001000",-->and ax,px<

"10110111",-->mov ix,55<

"01001111",-->mov px,ix<

"00100100",-->jz cs:px<

"10000000",-->mov ix,0<

"01010111",-->mov ds,ix<

"01011010",-->mov dx,ds<

"00010101",-->out 5<

"00010100",-->out 4<

"00001011",-->lods<

"00010001",-->out 1<

"00010000",-->out 0<

"10001111",-->mov ix,15<

"01001111",-->mov px,ix<

"00100000",-->jmp cs:px<

"00101010",-->push ds<

"00100101",-->j<

"11001000",-->mov ix,72<

"01001111",-->mov px,ix<

"00100100",-->jz cs:px<

"11001010",-->mov ix,74<

"01001111",-->mov px,ix<

"00100000",-->jmp cs:px<

"10001000",-->mov ix,8<

"01010111",-->mov ds,ix<

"00011010",-->dec ds<

"01011010",-->mov dx,ds<

"00010101",-->out 5<

"00010100",-->out 4<

"00001011",-->lods<

"00010001",-->out 1<

"00010000",-->out 0<

"10001111",-->mov ix,15<

"01001111",-->mov px,ix<

"00100000",-->jmp cs:px<

"00000101",-->in 1<

"10000001",-->mov ix,1<

"01000111",-->mov ax,ix<

"10000000",-->mov ix,0<

"01001111",-->mov px,ix<

"00000010",-->jz ax:px<

"00000100",-->in 0<

"11100000",-->mov ix,96<

"01001111",-->mov px,ix<

"00100100",-->jz cs:px<

"00111110",-->inc si<

"00111110",-->inc si<

"00011110",-->dec si<

"01011110",-->mov dx,si<

"00010011",-->out 3<

"00010010",-->out 2<

"00001011",-->lods<

"00010001",-->out 1<

"00010000",-->out 0<

"10001111",-->mov ix,15<

"01001111",-->mov px,ix<

"00100000",-->jmp cs:px<

"01000011",-->mov ax,dx<

"00001101",-->shl ax,1<

"00000100",-->in 0<

"11110001",-->mov ix,113<

"01001111",-->mov px,ix<

"00100100",-->jz cs:px<

"00111000",-->inc ax<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00010001",-->out 1<

"00010000",-->out 0<

"10001111",-->mov ix,15<

"01001111",-->mov px,ix<

"00100000",-->jmp cs:px<

"10000000",-->mov ix,0<

"01010111",-->mov ds,ix<

"00011111",-->dec ix<

"01110111",-->mov si,ix<

"00001011",-->lods<

"01000011",-->mov ax,dx<

"10000111",-->mov ix,7<

"01001111",-->mov px,ix<

"00001000",-->and ax,px<

"10000000",-->mov ix,0<

"00011111",-->dec ix<

"00011111",-->dec ix<

"00011111",-->dec ix<

"01001111",-->mov px,ix<

"00100001",-->jmp ax:px<

"00000000",

"00000000",

"00000000",

"00000000",

"00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000",

"10101001",-->mov ix,41<

"01001111",-->mov px,ix<

"00100000"-->jmp cs:px<

),(--1

"00001110",-->mode<

"01010000",-->mov ds,ax<

"10000000",-->mov ix,0<

"01110111",-->mov si,ix<

"11111111",-->mov ix,127<

"00111111",-->inc ix<

"01001111",-->mov px,ix<

"10000001",-->mov ix,1<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11011111",-->mov ix,95<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000010",-->mov ix,2<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010101",-->mov ix,21<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010100",-->mov ix,20<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11000011",-->mov ix,67<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11011111",-->mov ix,95<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010011",-->mov ix,19<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010010",-->mov ix,18<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11001011",-->mov ix,75<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000001",-->mov ix,1<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11011000",-->mov ix,88<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010001",-->mov ix,17<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010000",-->mov ix,16<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10001110",-->mov ix,14<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01010111",-->mov ds,ix<

"01110111",-->mov si,ix<

"10000001",-->mov ix,1<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11000111",-->mov ix,71<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11001111",-->mov ix,79<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10100001",-->mov ix,33<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01010111",-->mov ds,ix<

"01110111",-->mov si,ix<

"01011111",-->mov dx,ix<

"00010101",-->out 5<

"00010100",-->out 4<

"00010011",-->out 3<

"00010010",-->out 2<

"00010001",-->out 1<

"00010000",-->out 0<

"10000000",-->mov ix,0<

"01000111",-->mov ax,ix<

"10001111",-->mov ix,15<

"01001111",-->mov px,ix<

"00100001",-->jmp ax:px<

"00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000",

"10000001",-->mov ix,1<

"01001111",-->mov px,ix<

"00100000"-->jmp cs:px<

),(--2

"01010000",-->mov ds,ax<

"10000000",-->mov ix,0<

"01110111",-->mov si,ix<

"11111111",-->mov ix,127<

"00111111",-->inc ix<

"01001111",-->mov px,ix<

"10000001",-->mov ix,1<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11011111",-->mov ix,95<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010101",-->mov ix,21<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010100",-->mov ix,20<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11000011",-->mov ix,67<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11011111",-->mov ix,95<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010011",-->mov ix,19<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010010",-->mov ix,18<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10111000",-->mov ix,56<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11011000",-->mov ix,88<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010001",-->mov ix,17<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010000",-->mov ix,16<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01010111",-->mov ds,ix<

"01110111",-->mov si,ix<

"10000010",-->mov ix,2<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11000111",-->mov ix,71<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11001111",-->mov ix,79<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10100001",-->mov ix,33<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10001110",-->mov ix,14<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01010111",-->mov ds,ix<

"01110111",-->mov si,ix<

"01011111",-->mov dx,ix<

"00010101",-->out 5<

"00010100",-->out 4<

"00010011",-->out 3<

"00010010",-->out 2<

"00010001",-->out 1<

"00010000",-->out 0<

"10000000",-->mov ix,0<

"01000111",-->mov ax,ix<

"10001111",-->mov ix,15<

"01001111",-->mov px,ix<

"00100001",-->jmp ax:px<

"00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000",

"10000000",-->mov ix,0<

"01001111",-->mov px,ix<

"00100000"-->jmp cs:px<

),(--3

"01010000",-->mov ds,ax<

"10000000",-->mov ix,0<

"01110111",-->mov si,ix<

"11111111",-->mov ix,127<

"00111111",-->inc ix<

"01001111",-->mov px,ix<

"11111111",-->mov ix,127<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11000111",-->mov ix,71<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10001101",-->mov ix,13<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10001101",-->mov ix,13<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10001101",-->mov ix,13<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11001000",-->mov ix,72<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000100",-->mov ix,4<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11000111",-->mov ix,71<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000001",-->mov ix,1<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11011000",-->mov ix,88<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010110",-->mov ix,22<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10011011",-->mov ix,27<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10011011",-->mov ix,27<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010110",-->mov ix,22<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10011011",-->mov ix,27<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010110",-->mov ix,22<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10011011",-->mov ix,27<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010110",-->mov ix,22<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10001110",-->mov ix,14<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01010111",-->mov ds,ix<

"01110111",-->mov si,ix<

"10000011",-->mov ix,3<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11000111",-->mov ix,71<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"01011001",-->mov dx,px<

"00100011",-->stos<

"00111110",-->inc si<

"11001111",-->mov ix,79<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10100001",-->mov ix,33<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"10000000",-->mov ix,0<

"01010111",-->mov ds,ix<

"01110111",-->mov si,ix<

"01011111",-->mov dx,ix<

"00010101",-->out 5<

"00010100",-->out 4<

"00010011",-->out 3<

"00010010",-->out 2<

"00010001",-->out 1<

"00010000",-->out 0<

"10000000",-->mov ix,0<

"01000111",-->mov ax,ix<

"10001111",-->mov ix,15<

"01001111",-->mov px,ix<

"00100001",-->jmp ax:px<

"00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000","00000000",

"10000000",-->mov ix,0<

"01001111",-->mov px,ix<

"00100000"-->jmp cs:px<

),(--4

"01010000",-->mov ds,ax<

"10000000",-->mov ix,0<

"01110111",-->mov si,ix<

"11111111",-->mov ix,127<

"00111111",-->inc ix<

"01001111",-->mov px,ix<

"10100000",-->mov ix,32<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11000111",-->mov ix,71<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11011111",-->mov ix,95<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10011011",-->mov ix,27<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010111",-->mov ix,23<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10111000",-->mov ix,56<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10111100",-->mov ix,60<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11001111",-->mov ix,79<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10001111",-->mov ix,15<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010100",-->mov ix,20<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11001111",-->mov ix,79<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10100100",-->mov ix,36<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10111100",-->mov ix,60<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11001111",-->mov ix,79<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10001111",-->mov ix,15<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000110",-->mov ix,6<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11001111",-->mov ix,79<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10100000",-->mov ix,32<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10010111",-->mov ix,23<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10001110",-->mov ix,14<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01010111",-->mov ds,ix<

"01110111",-->mov si,ix<

"10000100",-->mov ix,4<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11000111",-->mov ix,71<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01000111",-->mov ax,ix<

"00000001",-->add ax,px<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"11001111",-->mov ix,79<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10100001",-->mov ix,33<

"01000111",-->mov ax,ix<

"01011000",-->mov dx,ax<

"00100011",-->stos<

"00111110",-->inc si<

"10000000",-->mov ix,0<

"01010111",-->mov ds,ix<

"01110111",-->mov si,ix<

"01011111",-->mov dx,ix<

"00010101",-->out 5<

"00010100",-->out 4<

"00010011",-->out 3<

"00010010",-->out 2<

"00010001",-->out 1<

"00010000",-->out 0<

"10000000",-->mov ix,0<

"01000111",-->mov ax,ix<

"10001111",-->mov ix,15<

"01001111",-->mov px,ix<

"00100001",-->jmp ax:px<

"00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000", "00000000",

"10000000",-->mov ix,0<

"01001111",-->mov px,ix<

"00100000"-->jmp cs:px<

));

begin

if(falling\_edge(clk))then

if(rom\_task='1')then

data<=mem(conv\_integer(lines))(conv\_integer(cols));

else

data<=(others=>'Z');

end if;

end if;

end process;

end;

## B11. stack.vhd

library ieee;

use ieee.std\_logic\_1164.all;

entity stack is

port(

clk:in std\_logic;

stack\_task,push:in std\_logic;

data:inout std\_logic\_vector(7 downto 0)

);

end;

architecture bhv of stack is

subtype t\_vector is std\_logic\_vector(7 downto 0);

type t\_matrix is array(127 downto 0)of t\_vector;

begin

process(clk)

variable room:t\_matrix;

variable top:integer range 0 to 127;

begin

if(falling\_edge(clk))then

if(stack\_task='1')then

if(push='1')then

room(top):=data;

top:=top+1;

data<=(others=>'Z');

else

top:=top-1;

data<=room(top);

end if;

else

data<=(others=>'Z');

end if;

end if;

end process;

end;

## B12. num\_scan.vhd

library ieee;

use ieee.std\_logic\_1164.all;

entity num\_scan is

port(

clk:in std\_logic;

id:in integer range 0 to 7;

num:in std\_logic\_vector(3 downto 0);

data:out std\_logic\_vector(3 downto 0);

sel:out std\_logic\_vector(5 downto 0);

dignum0,dignum1,dignum2,dignum3,dignum4,dignum5

:out std\_logic\_vector(3 downto 0)

);

end;

architecture scan of num\_scan is

signal cnt:integer range 0 to 16383;

signal v:std\_logic\_vector(23 downto 0);

begin

dignum0<=v(3 downto 0);

dignum1<=v(7 downto 4);

dignum2<=v(11 downto 8);

dignum3<=v(15 downto 12);

dignum4<=v(19 downto 16);

dignum5<=v(23 downto 20);

process(clk)

variable which:integer range 0 to 5;

begin

if(falling\_edge(clk))then

case id is

when 0=>v(3 downto 0)<=num;

when 1=>v(7 downto 4)<=num;

when 2=>v(11 downto 8)<=num;

when 3=>v(15 downto 12)<=num;

when 4=>v(19 downto 16)<=num;

when 5=>v(23 downto 20)<=num;

when others=>

end case;

if(cnt=16383)then

cnt<=0;

if(which/=5)then

which:=which+1;

else

which:=0;

end if;

case which is

when 0=>sel<="111110";data<=v(3 downto 0);

when 1=>sel<="111101";data<=v(7 downto 4);

when 2=>sel<="111011";data<=v(11 downto 8);

when 3=>sel<="110111";data<=v(15 downto 12);

when 4=>sel<="101111";data<=v(19 downto 16);

when 5=>sel<="011111";data<=v(23 downto 20);

end case;

else

cnt<=cnt+1;

end if;

end if;

end process;

end;

# 附录C 实物测试照片

