**数字系统（课程）设计**

**实 验 报 告**

|  |  |
| --- | --- |
| **课程名称：** |  |
| **学生姓名：** |  |
| **学生学号：** |  |
| **学生专业：** |  |
| **开课学期：** |  |
| **实验成绩：** |  |

**电子信息学院**

**2016年5月**

**电子与信息学院本科教学实验室**

**学生实验安全操作规则**

为培养学生严谨的学习作风，营造安全的实验实践学习氛围，确保学生人身和仪器设备安全，顺利完成实验任务，特制定以下规则。

1、不得赤脚或穿拖鞋进入实验室，不得将食物带入实验室，不准随地丢弃废纸、废弃物，保持实验室清洁。

2、实验中不得触摸裸露的接线柱、接线片、导线，不得用表棒、镊子、剪刀等工具插入电源插座。严禁带电接线、拆线或改接线路。

3、电烙铁在通电而不用时，应始终置于烙铁架内，严禁将其随意摆在桌边或地上；留意烙铁头避开导线及附近的仪器设备和书籍用品；人体不可触碰烙铁头，以防烫伤或漏电事故。中途离开实验室或实验完毕，要及时断电，以免发生火灾事故。

4、实验接线完毕，要仔细复查，确认无误后方可接通电源。

5、不要动用与本次实验无关的设备、仪器、工具和原材料。

6、实验结束后，实验装置、仪器仪表及电烙铁要断电，将实验用过的有关的仪器、工具、导线及电缆整理好，放回原处；并将座椅推至实验台下方可离场。

7、遇到事故应立即断电，并及时向实验指导教师报告。

8、严守纪律，遵守实验室规章和安全制度，严格执行操作规则。

9、提高防火防盗意识，做好防火防盗工作。

电子与信息学院实验中心

二〇一六年七月

**本人已阅读上述《学生实验安全操作规则》，知悉规则条款，并承诺认真执行。**

承 诺 人：

学 院：

专业/班级：

时 间：

**目 录**

[实验一 Quartus软件的使用；VHDL程序结构的熟悉 1](#_Toc449456884)

[实验二 基本电路模块设计（组合、时序） 5](#_Toc449456885)

[实验三 基于状态机的交通灯控制 12](#_Toc449456886)

[实验四 按键控制的状态机设计 19](#_Toc449456887)

# Quartus软件的使用；VHDL程序结构的熟悉

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地 点：** | 30号 楼 | 506房； | **实验台号：** | 40 |
| **实验日期与时间：** | 2016年10月14日 | | **评 分：** |  |
| **预习检查纪录：** |  | | **批改教师：** |  |

报告内容：

1. 实验要求：

1. 熟悉Quartus软件的使用

2. 熟悉VHDL程序结构

3. 掌握VHDL设计电路的仿真、综合、测试过程

1. 实验内容：

按新教材“VHDL数字系统设计”第6章实例（P119页开始）操作，掌握建立工程、设计输入、编译、时序仿真等开发流程；至有仿真结果为止。要理解VHDL代码意思，将软件和硬件结合起来，掌握开发流程。

1. 实验设计原理；

图 1 quartus开发流程

四, 实验过程记录（流程图或者实验逻辑思路过程）

**1. 建立工程**

启动quartus软件之后直接新建工程，将工程文件名命名为light\_water，开发板选择CycloneⅡ系列的EP2C8Q208C8的芯片，工程建好之后，得到结果如下：

图 2 工程建立结果

**2. 设计输入**

**2.1 计数器脉冲模块设计**

按照实验原理的算法，先建立VHDL FILE文件，实现计数器脉冲模块的功能，代码截图如下：

图 3 计数器脉冲模块代码

其中，文件命名保持了与实体名的一致。再生成相应的电路模块结果如下：

图 4 计数器脉冲模块

其中CLK为时钟信号，RST为复位信号，PUL为脉冲输出，每经过10000000个CLK周期就输出脉冲信号。

**2.2 流水灯模块设计**

先建立VHDL FILE文件，实现流水灯模块的功能，代码截图如下：

图 5 流水灯模块代码

其中，文件命名保持了与实体名的一致。再生成相应的电路模块结果如下：

图 6 流水灯模块

其中PUL为脉冲输入信号，RST为复位信号，随着PUL信号的接收，六个流水灯被轮流点亮，实现流水灯效果。

**2.3 整体系统电路设计**

最后，建立.bdf文件，画好电路图如下：

图 7 系统电路图

**3. 编译**

先将.bdf文件设置为顶层文件，编译结果如下：

图 8 编译成功结果

**4. 时序仿真**

输入CLK的周期为20ns，得到仿真结果如下

图 9 仿真结果

五，实验结果和分析

实验结果如图8所示，经观察，仿真结果符合设计要求。需要注意的是，由于Quartus自带仿真软件仿真时间过短，所以这里将原代码20ms所需要的计数数值改小，以便进行仿真。

**（实验报告作品相片粘贴页）**

（工程软件截图、仿真结果或者实验平台运行效果照片）

# 基本电路模块设计（组合、时序）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地 点：** | 30 楼 | 506 房； | **实验台号：** | 40 |
| **实验日期与时间：** | 2016.10.28 | | **评 分：** |  |
| **预习检查纪录：** |  | | **批改教师：** |  |

报告内容：

一，实验要求：

用VHDL语言设计组合逻辑、时序逻辑电路模块

1. 实验内容：

组合逻辑、时序逻辑基本模块电路设计

1、3-8 译码器

设计要求：

1) 3个译码输入端A、B、C，分别对应开发板上表示为ON DIP的2,3,4

2) EN为使能端（低电平有效），对应开发板上表示为 ON DIP的1管脚

3) Y为译码输出，8位位矢量类型。D3至D10

4) 输入采用电平开关，译码输出采用LED指示灯显示

5）管脚对应见附件：信号分配表

2、分频器实验

设计要求：

1) 将EDA板上的系统时钟50MHz分频为1Hz的时钟信号

2) 占空比为50%

3) 利用流水灯点亮程序，在 EDA 板上观察效果

4) 实体命名为clkdiv\_(班级号)\_(班级序号)

1. 实验设计原理；

1. 38译码器原理

根据如下译码表，很容易将输入ABC转化为输出Y

2. 分频器原理

对输入时钟进行计数，当计数到频率的一半时，输出脉冲跳转，当计数完整个频率时，输出脉冲极性再次跳转，由此可以实现占空比为50%的时钟信号分频。

对于已经分频的信号，用简单的case语句即可实现流水灯。

四, 实验过程记录（流程图或者实验逻辑思路过程）

1. 38译码器实现流程图如下

2. 分频器流程图

五，实验结果和分析

实验结果：

实现了3-8译码，并将50MHz的频率分成1Hz，驱动了流水灯。

分析：

1. 在对38译码器进行时序仿真时，需采用function仿真，否则若用timing仿真，结果的波形会有延时，也没有漂亮的方波，甚至有毛刺，如下图所示：

2. 在对分频器进行仿真时，由于quartus无法仿真出50MHz的分频，所以应该减小输入频率以实现分频。

3. 打代码时，由于能力有限，所以不应该追求快速，而应该一个模块一个模块地逐步实现，步步为营不操之过急，在debug时也是从一个一个小模块排除起。

六，附录（打印源代码页）

1. 38译码器代码

library ieee;

use ieee.std\_logic\_1164.all;

entity decoder38\_dl\_38 is

port(a,b,c:in std\_logic;

en:in std\_logic;

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

end entity;

architecture behav of decoder38\_dl\_38 is

signal abc:std\_logic\_vector(2 downto 0);

begin

abc<=a&b&c;

process(en,abc)

begin

if en='0' then

case abc is

when "000"=>y<="11111110";

when "001"=>y<="11111101";

when "010"=>y<="11111011";

when "011"=>y<="11110111";

when "100"=>y<="11101111";

when "101"=>y<="11011111";

when "110"=>y<="10111111";

when "111"=>y<="01111111";

when others=>y<="XXXXXXXX";

end case;

else

y<="11111111";

end if;

end process;

end behav;

2. 分频器代码

分频器部分

library ieee;

use ieee.std\_logic\_1164.all;

entity div\_dl\_38 is

port(clk:in std\_logic;

q:out std\_logic);

end entity;

architecture behav of div\_dl\_38 is

begin

process(clk)

variable time:integer range 0 to 4;

begin

if rising\_edge(clk) then time:=time+1;

end if;

if time=4 then q<='1';time:=0;

elsif time=2 then q<='0';

elsif time=0 then q<='1';--赋初值

end if;

end process;

end behav;

流水灯部分

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity clkdiv\_dl\_38 is

port(q:in std\_logic;

led:out std\_logic\_vector(5 downto 0));

end entity;

architecture behav1 of clkdiv\_dl\_38 is

signal i:std\_logic\_vector(2 downto 0):="000";

begin

process(q)

begin

if rising\_edge(q) then

i<=i+'1';

if i="101" then

i<="000";

end if;

case i is

when "000"=>led<="111110";

when "001"=>led<="111101";

when "010"=>led<="111011";

when "011"=>led<="110111";

when "100"=>led<="101111";

when "101"=>led<="011111";

when others=>null;

end case;

end if;

end process;

end behav1;

**（实验报告作品相片粘贴页）**

（工程软件截图、仿真结果或者实验平台运行效果照片）

1. 38译码器部分

电路连接图

仿真结果

实验结果

2. 分频器部分

电路连接图

仿真结果

实验结果

# 实验三 基于状态机的交通灯控制

报告内容：

1. 实验要求：

运用状态机设计红黄绿交通灯控制电路系统

1. 实验内容：

1、 开发板上三个led等分别代表公路上红黄绿三种颜色交通灯，D3代表红灯，D5代表黄灯，D7代表绿灯 。D3，D5，D7 对应开发板的IO口为120，118，116 。

2、 交通灯状态机初始状态为红灯，交通灯工作过程依次是红 绿 黄 红。

3、 为了方便观察，本次实验要求红灯的显示时间为9s，绿灯显示时间为6s ，黄灯显示时间为3s ，时间需要倒计时，在数码管Q4上显示。Q4的使能端（低电平有效）对应开发板IO口为168，八段led对应IO口为 A~H (144,158,162,160,159,156,163,161)。

4、 编程之前要求同学们先画好ASM图。

5、 1Hz分频模块请采用第二次实验中的内容，7段码显示模块请参考书本151页内容。

6、 顶层模块命名方式，traffic\_light\_(班级)\_(班级序号) 。

1. 实验设计原理；

先通过分频器得到1hz的时钟脉冲，根据时钟脉冲依次点亮红黄绿灯，并输出对应计数器，将计数器的数值转换为七段码以便数码管显示，由于电路板要求输入串行数据，因此先将并行数据转换为串行数据（7Hz时钟脉冲），最后将数据流传入单片机实现电路功能。实验原理如下：

四, 实验过程记录（流程图或者实验逻辑思路过程）

1. 设计分频器获取1Hz时钟脉冲。

设计原理与上一实验相同

2. 根据时钟脉冲输出交通灯状态

根据实验内容要求，交通灯控制电路可以作为一个有限状态机来设计，用R、G、Y分别代表红、绿、黄三种交通灯状态，sub代表计数减一的状态，asm图如下：

图 10 交通灯asm图

3. 数据编码转为七段码

八段led译码列表如下，根据表格将当前计数值转换为可被数码管显示的七段码。

4. 并行数据转为串行数据

此部分应实现的功能为，对于输入的频率为1Hz的脉冲，可将上述的8位并行led译码转化为8位串行的led译码。为此需要频率为8Hz的脉冲时钟，此时钟频率的获得原理与上一实验相同，不再赘述。

五，实验结果和分析

仿真结果如下

分析实验结果可知，红、绿、黄灯分别点亮9秒、6秒和3秒，并通过计数器输出计数时间，仿真结果正确。

六，附录（打印源代码页）

1. 分频器代码（仿真版本）

library ieee;

use ieee.std\_logic\_1164.all;

entity div\_dl\_38 is

port(clk:in std\_logic;

q:out std\_logic);

end entity;

architecture behav1 of div\_dl\_38 is

begin

process(clk)

variable time:integer range 0 to 4;

begin

if rising\_edge(clk) then time:=time+1;

end if;

if time=4 then q<='1';time:=0;

elsif time=2 then q<='0';

elsif time=0 then q<='1';--赋初值

end if;

end process;

end behav1;

2. 根据时钟脉冲输出交通灯状态

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.ALL;

USE IEEE.std\_logic\_arith.ALL;

ENTITY traffic\_led\_dl\_38 IS

PORT(q:IN std\_logic;

reset:IN std\_logic;

rled,gled,yled:OUT std\_logic;

counter:OUT std\_logic\_vector(3 DOWNTO 0));

END ENTITY;

ARCHITECTURE nan OF traffic\_led\_dl\_38 IS

SIGNAL mtime:integer RANGE 0 TO 9;

BEGIN

PROCESS(q,reset)

TYPE state\_type IS(R,G,Y);

VARIABLE state:state\_type;

BEGIN

IF reset='1'

THEN state:=R;

rled<='1';

gled<='0';

yled<='0';

mtime<=9;

counter<=conv\_std\_logic\_vector(mtime,4);

ELSIF q'event and q='1' THEN

CASE state IS

WHEN R=>rled<='1';gled<='0'; yled<='0';

mtime<=mtime-1;

counter<=conv\_std\_logic\_vector(mtime,4);

IF(mtime=1)THEN state:=G; mtime<=6;

ELSE state:=R;

END IF;

WHEN G=>gled<='1'; rled<='0';yled<='0';

mtime<=mtime-1;

counter<=conv\_std\_logic\_vector(mtime,4);

IF(mtime=1)THEN state:=Y; mtime<=3;

ELSE state:=G;

END IF;

WHEN Y=>yled<='1'; gled<='0'; rled<='0';

mtime<=mtime-1;

counter<=conv\_std\_logic\_vector(mtime,4);

IF(mtime=1)THEN state:=R; mtime<=9;

ELSE state:=Y;

END IF;

END CASE;

END IF;

END PROCESS;

END nan;

3. 数据编码转为七段码

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.ALL;

ENTITY seven\_seg IS

PORT(counter:IN std\_logic\_vector(3 DOWNTO 0);

seven\_seg\_input:OUT std\_logic\_vector(7 DOWNTO 0));

END entity;

ARCHITECTURE led\_arch OF seven\_seg IS

BEGIN

PROCESS(counter)

BEGIN

CASE counter IS

WHEN"0000"=>seven\_seg\_input<="00000010";

WHEN"0001"=>seven\_seg\_input<="10011110";

WHEN"0010"=>seven\_seg\_input<="00100100";

WHEN"0011"=>seven\_seg\_input<="00001100";

WHEN"0100"=>seven\_seg\_input<="10011000";

WHEN"0101"=>seven\_seg\_input<="01001000";

WHEN"0110"=>seven\_seg\_input<="01000000";

WHEN"0111"=>seven\_seg\_input<="00011110";

WHEN"1000"=>seven\_seg\_input<="00000000";

WHEN"1001"=>seven\_seg\_input<="00001000";

WHEN OTHERS=>seven\_seg\_input<="11111111";

END CASE;

END PROCESS;

END led\_arch;

4. 并串转换

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity p2s is

port(

reset : in std\_logic;

clk : in std\_logic;

start : in std\_logic; --low active,data\_in valid

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

data\_valid : out std\_logic; --high active,output data valid

ready : out std\_logic; --low active,ready to recieve data

q : out std\_logic

);

end p2s;

architecture Behavioral of p2s is

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

signal cnt : std\_logic\_vector(3 downto 0);

signal reg\_en : std\_logic;

signal shift\_start : std\_logic;

type state is (idle,recieve,shift,finish);

signal current\_state, next\_state : state;

begin

counter: process(reset,clk,shift\_start)

begin

if(reset = '0') then

cnt <= (others => '0');

elsif(clk'event and clk = '1') then

if(shift\_start = '0') then

cnt <= cnt + 1;

else

cnt <= (others => '0');

end if;

end if;

end process counter;

fsm: block

begin

sync: process(reset,clk)

begin

if(reset= '0') then

current\_state <= idle;

elsif(clk'event and clk = '1') then

current\_state <= next\_state;

end if;

end process sync;

comb: process(current\_state,cnt,start)

begin

case current\_state is

when idle =>

ready <= '0';

reg\_en <= '1';

shift\_start <= '1';

data\_valid <= '1';

if(start = '0') then

reg\_en <= '0';

next\_state <= recieve;

else

next\_state <= idle;

end if;

when recieve =>

reg\_en <= '1';

ready <= '1';

data\_valid <= '0';

shift\_start <= '0';

next\_state <= shift;

when shift =>

reg\_en <= '1';

ready <= '1';

data\_valid <= '0';

if(cnt = 8) then

shift\_start <= '1';

next\_state <= finish;

else

shift\_start <= '0';

next\_state <= shift;

end if;

when finish =>

reg\_en <= '1';

ready <= '0';

data\_valid <= '1';

shift\_start <= '1';

next\_state <= idle;

when others =>

next\_state <= idle;

end case;

end process comb;

end block fsm;

data\_channel: process(reset,clk)

begin

if(reset = '0') then

reg <= (others => '0');

q <= '0';

elsif(clk'event and clk = '1') then

if(reg\_en = '0') then

reg <= data\_in;

elsif(shift\_start = '0') then

q <= reg(7);

for i in 7 downto 1 loop --shift register

reg(i) <= reg(i - 1);

end loop;

reg(0) <= '0';

else

q <= '0';

end if;

end if;

end process data\_channel;

end Behavioral;

**（实验报告作品相片粘贴页）**

（工程软件截图、仿真结果或者实验平台运行效果照片）

1. 整体电路图

2. 仿真结果

3. 实验结果

# 实验四 按键控制的状态机设计

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地 点：** | 楼 | 房； | **实验台号：** |  |
| **实验日期与时间：** |  | | **评 分：** |  |
| **预习检查纪录：** |  | | **批改教师：** |  |

报告内容：

1. 实验要求：

运用状态机设计按键控制数码管显示的电路系统

1. 实验内容：

1、 按键控制数码管显示，当按下 S2 时， Q4 数码管显示 1； 再次按下 S2， Q4 数码管显示 2； 第三次按下 S2， Q4 数码管显示 3； 第四次按下 S2 时， Q4 数码管显示 4.第五次按下 S2 时 Q4 又从 1 开始显示，如此反复。 Q4 使能对应 IO 口为 168（低电平有效），按键 S2 对应 IO 口为 125（按下为低电平）。

2、 八段 led 对应 IO 口为 A~H (144,158,162,160,159,156,163,161)。

3、 检测按键是否按下需要消抖，消抖程序请同学们参考附录程序，参考程序是利用四个按键控制四个 LED 灯的亮灭。请同学们在读懂参考程序的基础上将其改编为符合要求 1 的程序。

4、 参考程序是普通的控制程序，本实验要求同学们利用状态机来编写程序。

5、 顶层模块命名方式， key\_state\_(班级)\_(班级序号)。

1. 实验设计原理；

用延时消抖的方法检测按键输入，实现每来一个按键就输出一个脉冲信号。用三进程状态机实现按键控制数码管显示，其中，根据时钟变化决定时序进程；根据是否有按键脉冲信号决定次态逻辑的变化；根据现态的值决定输出给数码管的比特串。最后，用实验三的seg程序把输入的十六位二进制数显示在数码管。原理方框图如下所示：

四, 实验过程记录（流程图或者实验逻辑思路过程）

1. 按键检测程序设计

延时消抖算法如下：

图 11 按键检测算法

生成电路模块如下：

图 12 按键检测电路模块

2. 按键控制数码管显示

设计三进程状态机，其中输入信号为按键输入，输出信号为s1,s2,s3,s4，分别表示数码管显示数字1、2、3、4，画出asm图如下：

图 13 按键控制状态机asm图

生成电路模块如下：

图 14 按键控制电路模块

五，实验结果和分析

1. 按键延时消抖结果

仿真结果如下

分析：

为了方便仿真，将延时消抖的时间缩短为2个时钟，从图中可见，按键出现后的两个时钟上升沿即出现了按键检测脉冲，长度为一个时钟周期长度，这符合我们的预期效果。

2. 按键控制结果

仿真结果如下

分析：

从图中看见，每来一个按键脉冲，数码管的显示数值就增加一，并从1到4循环，这符合我们的预期。

六，附录（打印源代码页）

1. 读取键值代码（延时消抖）

library ieee;

use ieee.std\_logic\_1164.all;

entity key\_read IS

generic(n:integer :=3);--n=250000 for 5ms

PORT(clk,key\_in:in std\_logic;

key\_out:out std\_logic);

end entity;

architecture behav of key\_read is

begin

PROCESS (clk,key\_in)

variable count1:integer range n downto 0:=0;

begin

if key\_in='0' then

if rising\_edge(clk) then

if count1<n then count1:=count1+1;

else count1:=count1;

end if;

if count1=n-1 then key\_out<='1';

else key\_out<='0';

end if;

end if;

else count1:=0;

end if;

end process;

end behav;

2. 按键控制代码

library ieee;

use ieee.std\_logic\_1164.all;

entity key\_state\_dl\_38 is

port(clk,key\_out:in std\_logic;

seg\_in:out std\_logic\_vector(15 downto 0));

end entity;

architecture behav2 of key\_state\_dl\_38 is

type state is(s1,s2,s3,s4);

signal present\_state,next\_state:state:=s1;

begin

--seq进程，产生状态寄存器的时序进程

seq:process(clk)

begin

if rising\_edge(clk)then

present\_state<=next\_state;

end if;

end process seq;

--ns进程，产生次态进程

ns:process(key\_out)

begin

case present\_state is

when s1=>

if(key\_out='1')then next\_state<=s2;

else next\_state<=s1;

end if;

when s2=>

if(key\_out='1')then next\_state<=s3;

else next\_state<=s2;

end if;

when s3=>

if(key\_out='1')then next\_state<=s4;

else next\_state<=s3;

end if;

when s4=>

if(key\_out='1')then next\_state<=s1;

else next\_state<=s4;

end if;

end case;

end process ns;

--op进程，产生输出逻辑

op:process(present\_state)

begin

case present\_state is

when s1=>seg\_in<="0000000000000001";

when s2=>seg\_in<="0000000000000010";

when s3=>seg\_in<="0000000000000011";

when s4=>seg\_in<="0000000000000100";

end case;

end process op;

end behav2;

3. seg显示代码

module seg(

clk,rst\_n,en,

idis\_data,

ds\_stcp,ds\_shcp,ds\_data

);

input clk; //25M??????

input rst\_n; //??????????

input en;

input [15:0] idis\_data ;

output ds\_stcp; //74HC595????????????????????????

output ds\_shcp; //74HC595?????????????????????

output ds\_data; //74HC595???????

//????? 0~F ??????

parameter SEG\_NUM0 = 8'hc0,

SEG\_NUM1 = 8'hf9,

SEG\_NUM2 = 8'ha4,

SEG\_NUM3 = 8'hb0,

SEG\_NUM4 = 8'h99,

SEG\_NUM5 = 8'h92,

SEG\_NUM6 = 8'h82,

SEG\_NUM7 = 8'hF8,

SEG\_NUM8 = 8'h80,

SEG\_NUM9 = 8'h90,

SEG\_NUMA = 8'h88,

SEG\_NUMB = 8'h83,

SEG\_NUMC = 8'hc6,

SEG\_NUMD = 8'ha1,

SEG\_NUME = 8'h86,

SEG\_NUMF = 8'h8e;

//????? 0~7????

parameter SEG\_WE0 = 8'b1111\_1110,

SEG\_WE1 = 8'b1111\_1101,

SEG\_WE2 = 8'b1111\_1011,

SEG\_WE3 = 8'b1111\_0111;

// SEG\_WE4 = 8'b1110\_1111,

// SEG\_WE5 = 8'b1101\_1111,

// SEG\_WE6 = 8'b1011\_1111,

// SEG\_WE7 = 8'b0111\_1111;

wire en;

reg clk\_div\_2;

reg clk1;

always@(en) //use enable siganl to break the module

begin

if (en==1)

clk1<=clk;

else clk1<=clk1;

end

always@(posedge clk1 or negedge rst\_n)

if(!rst\_n)

clk\_div\_2<=1'b0;

else

clk\_div\_2<=~clk\_div\_2;

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

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

//??????????

reg[3:0] seg\_num; //??????

reg[7:0] seg\_duan; //7???????????????8??

reg[7:0] seg\_wei; //7????????

reg[7:0] cnt\_4; //?????

//?????

always @(posedge clk\_div\_2 or negedge rst\_n)

if(!rst\_n) cnt\_4 <= 8'd0;

else cnt\_4 <= cnt\_4+1'b1;

//????

always @(posedge clk\_div\_2 or negedge rst\_n)

if(!rst\_n) seg\_num <= 8'h00;

else

case(cnt\_4[7:6])

2'b00: seg\_num <= idis\_data[3:0];

2'b01: seg\_num <= idis\_data[7:4];

2'b10: seg\_num <= idis\_data[11:8];

2'b11: seg\_num <= idis\_data[15:12];

default: ;

endcase

//??????

reg flag;

always @(posedge clk\_div\_2 or negedge rst\_n)

if(!rst\_n) begin seg\_duan <= 8'hff;

// flag<=1'b0;

end

//else if(flag) begin seg\_duan<=8'hff;

// flag<=~flag;

// end

else

case(seg\_num)

4'h0: seg\_duan <= SEG\_NUM0;

4'h1: seg\_duan <= SEG\_NUM1;

4'h2: seg\_duan <= SEG\_NUM2;

4'h3: seg\_duan <= SEG\_NUM3;

4'h4: seg\_duan <= SEG\_NUM4;

4'h5: seg\_duan <= SEG\_NUM5;

4'h6: seg\_duan <= SEG\_NUM6;

4'h7: seg\_duan <= SEG\_NUM7;

4'h8: seg\_duan <= SEG\_NUM8;

4'h9: seg\_duan <= SEG\_NUM9;

4'ha: seg\_duan <= SEG\_NUMA;

4'hb: seg\_duan <= SEG\_NUMB;

4'hc: seg\_duan <= SEG\_NUMC;

4'hd: seg\_duan <= SEG\_NUMD;

4'he: seg\_duan <= SEG\_NUME;

4'hf: seg\_duan <= SEG\_NUMF;

default: ;

endcase

//????

always @(cnt\_4[7:6])

case(cnt\_4[7:6])

2'b00: seg\_wei <= SEG\_WE0;

2'b01: seg\_wei <= SEG\_WE1;

2'b10: seg\_wei <= SEG\_WE2;

2'b11: seg\_wei <= SEG\_WE3;

default: seg\_wei <= 8'b0000\_0000;

endcase

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

//74HC95????

reg ds\_stcpr; //74HC595????????????????????????

reg ds\_shcpr; //74HC595?????????????????????

reg ds\_datar; //74HC595???????

//????????

always @(posedge clk\_div\_2 or negedge rst\_n)

if(!rst\_n) ds\_shcpr <= 1'b0;

else if((cnt\_4 > 8'h02 && cnt\_4 <= 8'h22) || (cnt\_4 > 8'h24 && cnt\_4 <= 8'h44)

|| (cnt\_4 > 8'h46 && cnt\_4 <= 8'h66) || (cnt\_4 > 8'h68 && cnt\_4 <= 8'h88)

|| (cnt\_4 > 8'h90 && cnt\_4 <= 8'hb0) || (cnt\_4 > 8'hb2 && cnt\_4 <= 8'hd2)

|| (cnt\_4 > 8'hd4 && cnt\_4 <= 8'hf4))

ds\_shcpr <= ~ds\_shcpr;

else ds\_shcpr<=1'b0;

//????????

always @(posedge clk\_div\_2 or negedge rst\_n)

if(!rst\_n) ds\_datar <= 1'b0;

else

case(cnt\_4)

8'h02,8'h46,8'h90,8'hd4: ds\_datar <= seg\_duan[7];

8'h04,8'h48,8'h92,8'hd6: ds\_datar <= seg\_duan[6];

8'h06,8'h4a,8'h94,8'hd8: ds\_datar <= seg\_duan[5];

8'h08,8'h4c,8'h96,8'hda: ds\_datar <= seg\_duan[4];

8'h0a,8'h4e,8'h98,8'hdc: ds\_datar <= seg\_duan[3];

8'h0c,8'h50,8'h9a,8'hde: ds\_datar <= seg\_duan[2];

8'h0e,8'h52,8'h9c,8'he0: ds\_datar <= seg\_duan[1];

8'h10,8'h54,8'h9e,8'he2: ds\_datar <= seg\_duan[0];

8'h12,8'h56,8'ha0,8'he4: ds\_datar <= seg\_wei[0];

8'h14,8'h58,8'ha2,8'he6: ds\_datar <= seg\_wei[1];

8'h16,8'h5a,8'ha4,8'he8: ds\_datar <= seg\_wei[2];

8'h18,8'h5c,8'ha6,8'hea: ds\_datar <= seg\_wei[3];

8'h1a,8'h5e,8'ha8,8'hec: ds\_datar <= seg\_wei[4];

8'h1c,8'h60,8'haa,8'hee: ds\_datar <= seg\_wei[5];

8'h1e,8'h62,8'hac,8'hf0: ds\_datar <= seg\_wei[6];

8'h20,8'h64,8'hae,8'hf2: ds\_datar <= seg\_wei[7];

8'h24,8'h68,8'hb2,: ds\_datar <= 1;

8'h26,8'h6a,8'hb4,: ds\_datar <= 1;

8'h28,8'h6c,8'hb6,: ds\_datar <= 1;

8'h2a,8'h6e,8'hb8,: ds\_datar <= 1;

8'h2c,8'h70,8'hba,: ds\_datar <= 1;

8'h2e,8'h72,8'hbc,: ds\_datar <= 1;

8'h30,8'h74,8'hbe,: ds\_datar <= 1;

8'h32,8'h76,8'hc0,: ds\_datar <= 1;

8'h34,8'h78,8'hc2,: ds\_datar <= 1;

8'h36,8'h7a,8'hc4,: ds\_datar <= 1;

8'h38,8'h7c,8'hc6,: ds\_datar <= 1;

8'h3a,8'h7e,8'hc8,: ds\_datar <= 1;

8'h3c,8'h80,8'hca,: ds\_datar <= 1;

8'h3e,8'h82,8'hcc,: ds\_datar <= 1;

8'h40,8'h84,8'hce,: ds\_datar <= 1;

8'h42,8'h86,8'hd0,: ds\_datar <= 1;

default: ds\_datar <= seg\_duan[0];

endcase

//????????

always @(posedge clk1 or negedge rst\_n)

if(!rst\_n) ds\_stcpr <= 1'b0;

else if((cnt\_4 == 8'h02) || (cnt\_4 == 8'h23) || (cnt\_4 == 8'h45) || (cnt\_4 == 8'h67) || (cnt\_4 == 8'h89)|| (cnt\_4 == 8'hb1)|| (cnt\_4 == 8'hd3)) ds\_stcpr <= 1'b0;

else if((cnt\_4 == 8'h22) || (cnt\_4 == 8'h44) || (cnt\_4 == 8'h66) || (cnt\_4 == 8'h88) || (cnt\_4 == 8'hb0)|| (cnt\_4 == 8'hd2)|| (cnt\_4 == 8'hf4)) ds\_stcpr <= 1'b1;

wire ds\_stcp = ds\_stcpr;

wire ds\_shcp = ds\_shcpr;

wire ds\_data = ds\_datar;

endmodule

**（实验报告作品相片粘贴页）**

（工程软件截图、仿真结果或者实验平台运行效果照片）

1. 按键检测仿结果

2. 按键控制仿真结果

3. 整体电路图

4. 实验结果