深 圳 大 学 实 验 报 告

课程名称：硬件描述语言及数字系统设计

实验项目名称：实验五 **LCD1602**

学院：医学院

专业：生物医学工程

指导教师：但果、董磊

报告人：陈焕鑫 学号：**2016222042** 班级：生工 **2** 班

实验时间：**2018.11.21**

实验报告提交时间：**2018.12.18**

教务部制

一、实验平台：

安装了 ISE 软件的 PC 计算机

硬件描述语言及数字系统设计实验平台

JTAG 下载线

二、实验目的：

当你完成整个项目之后，你将会学会一下内容：

- 掌握 1602LCD 的工作原理
- 能够使用 1602LCD 进行显示

三、实验内容：

掌握 1602LCD 的工作原理

用 VHDL 代码描述一个与 1602LCD 相关的程序，并编写测试激励进行测试

编写引脚约束文件，用 ISE 软件生成.bit 文件，下载到硬件描述语言及数字系统设计实验平台进行板级验证

四、实验原理：

液晶显示的原理是利用液晶的物理特性，通过电压对其显示区域进行控制，有电就有显示，这样即可以显示出图形， LCD1602 的显示容量为 16×2 个字符。

LCD1602 各引脚的功能说明如下表 2-3 所示。

第 1 脚：VSS 为地电源；

第 2 脚：VDD 接 5V 正电源；

第 3 脚：V0 为液晶显示器对比度调整端，接正电源时对比度最弱，接地时对比度最高；

第 4 脚：RS 为寄存器选择，高电平时选择数据寄存器、低电平时选择指令寄存器；

第 5 脚：R/W 为读写信号线，高电平时进行读操作，低电平时进行写操作。当 RS 和 R/W 共同为低电平时可以写入指令或者显示地址，当 RS 为低电平 R/W 为高电平时可以读忙信号，当 RS 为高电平 R/W 为低电平时可以写入数据；

第 6 脚：E 端为使能端，当 E 端由高电平跳变成低电平时，液晶模块执行命令；

第 7~14 脚：D0~D7 为 8 位双向数据线；

第 15 脚：背光源正极；

第 16 脚：背光源负极。

表2-3 LCD1602引脚说明

引脚	符号	功能说明
1	VSS	一般接地
2	VDD	接电源（+5V）
3	VO	液晶显示器对比度调整端，接正电源时对比度最弱，接地电源时对比度最高（对比度过高时会产生“鬼影”，使用时可以通过一个 10K 的电位器调整对比度）。
4	RS	RS 为寄存器选择，高电平 1 时选择数据寄存器、低电平 0 时选择指令寄存器。
5	R/W	R/W 为读写信号线，高电平(1)时进行读操作，低电平(0)时进行写操作。
6	E	E(或 EN)端为使能(enable)端，下降沿使能。
7	DB0	低 4 位三态、双向数据总线 0 位（最低位）
8	DB1	低 4 位三态、双向数据总线 1 位
9	DB2	低 4 位三态、双向数据总线 2 位
10	DB3	低 4 位三态、双向数据总线 3 位
11	DB4	高 4 位三态、双向数据总线 4 位
12	DB5	高 4 位三态、双向数据总线 5 位
13	DB6	高 4 位三态、双向数据总线 6 位
14	DB7	高 4 位三态、双向数据总线 7 位（最高位）（也是 busy flag）
15	BLA	背光电源正极
16	BLK	背光电源负极

LCD1602 液晶模块内部的控制器共有 11 条控制指令，如表 2-4 所示：

表2-4 LCD1602液晶模块控制指令

序号	指令功能	指令编码										执行时间
		RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
1	清屏	0	0	0	0	0	0	0	0	0	1	1.64ms
2	光标返回	0	0	0	0	0	0	0	0	1	*	1.64ms
3	进入模式设置	0	0	0	0	0	0	0	1	I/D	S	40μs
4	显示开/关控制	0	0	0	0	0	0	1	D	C	B	40μs
5	光标或显示屏移动	0	0	0	0	0	1	S/C	R/L	*	*	40μs
6	功能设定	0	0	0	0	1	DL	N	F	*	*	40μs
7	设定CGRAM地址	0	0	0	1	CGRAM地址(6位)						40μs
8	设定DDRAM地址	0	0	1	DDRAM地址(7位)							40μs
9	读忙标志或AC内容	0	1	BF	地址计数器（AC）内容(7位)							40μs
10	写数到CG/DDRAM	1	0	要写的数据								40μs
11	从CG/DDRAM读数	1	1	读出的数据								40μs

LCD1602 液晶模块的读写操作、屏幕和光标的操作都是通过指令编程来实现的。（说明： 1 为高电平、 0 为低电平）

指令 1：清屏指令

功能：

（1）清除液晶显示屏，使用 20H（空格的 ASCII 码）填满 DDRAM；
（2）光标归位，即将光标撤回到液晶显示屏的左上方；
（3）将 DDRAM 的地址计数器（AC）设置为 00H。

指令 2：光标返回指令

功能：

（1）保持 DDRAM 的内容不变；
（2）光标归位，即将光标撤回到液晶显示屏的左上方；
（3）将 DDRAM 的地址计数器（AC）设置为 00H。

指令 3：进入模式设置指令

I/D： AC 的加/减

当 I/D=1，光标右移， DDRAM 的 AC 加 1。

当 I/D=0，光标左移， DDRAM 的 AC 减 1。

S：显示的移位

当 S=1、 I/D=1 时，画面整体左移；

当 S=1、 I/D=0 时，画面整体右移。

指令 2：光标复位，光标返回到地址 00H。

指令 3：光标和显示模式设置

I/D：光标移动方向，高电平右移，低电平左移

S:屏幕上所有文字是否左移或者右移。高电平表示有效，低电平则无效

指令 4：显示开/关控制指令

功能：控制整体显示屏开/关、光标开/关以及光标是否闪。

D（控制整体显示屏开/关）： 0→整体显示关 1→整体显示开

C（控制光标开/关）： 0→无光标 1→有光标

B（控制光标是否闪）： 0→光标闪烁 1→光标不闪烁

0x08

0x0C

0x0E

0x0F 光标

指令 4：显示开关控制。

D：控制整体显示的开与关，高电平表示开显示，低电平表示关显示

C：控制光标的开与关，高电平表示有光标，低电平表示无光标

B：控制光标是否闪烁，高电平闪烁，低电平不闪烁。

指令 5：光标或显示移位

S/C：高电平时移动显示的文字，低电平时移动光标。

指令 6：功能设置命令

DL：高电平时为 4 位总线，低电平时为 8 位总线

N：低电平时为单行显示，高电平时双行显示

F: 低电平时显示 5x7 的点阵字符，高电平时显示 5x10 的点阵字符。

指令 7：字符发生器 RAM 地址设置。

指令 8： DDRAM 地址设置。

指令 9：读忙信号和光标地址

BF：为忙标志位，高电平表示忙，此时模块不能接收命令或者数据，如果为低电平表示不忙。

指令 10：写数据。

指令 11：读数据

液晶显示模块是一个慢显示器件，所以在执行每条指令之前一定要确认模块的忙标志为低电平，表示不忙，否则此指令失效。要显示字符时要先输入显示字符地址，也就是告诉模块在哪里显示字符， 图 2-35 是 1602 的内部显示地址

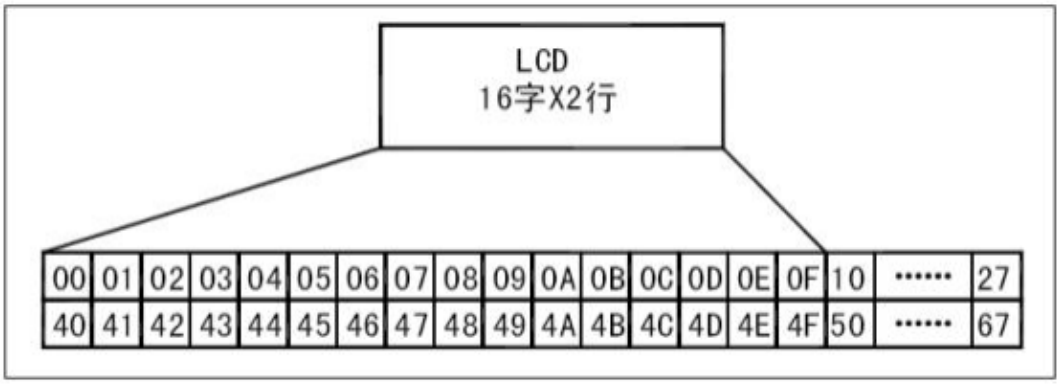
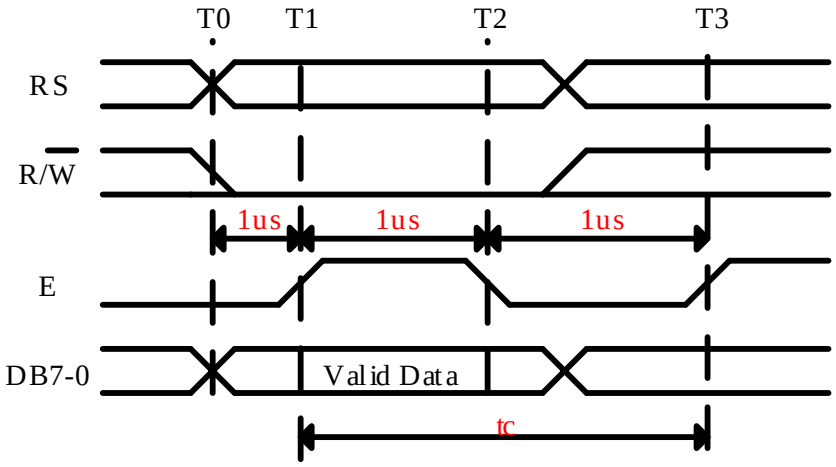


图 2-35

例如第二行第一个字符的地址是 40H，那么是否直接写入 40H 就可以将光标定位在第二行第一个字符的位置呢？这样不行，因为写入显示地址时要求最高位 D7 恒定为高电平 1 所以实际写入的数据应该是 01000000B（40H）
+10000000B(80H)=11000000B(C0H)。

LCD1602 写操作时序如下图所示：



LCD1602 在执行每条指令之前一般都要确认模块的忙标志。但是，由于现在很多国产的液晶忙标志读出来的时候都是错误的，因此，本教程建议不用读取忙标志，而采用每次给充分的时间执行指令。上图就是推荐的写操作时序图，在 T0 时刻，RS、R/W、DB7-0 信号有效，E 为 0，经过 1us，在 T1 时刻，E 拉高，再过 1us，在 T2 时刻，E 拉低，最后，再经过 1us，在 T3 时刻，E 拉高，表示写操作结束。

五、实验方法步骤及 VHDL 代码：

首先，根据分频原理进行代码设计，设计一个 1Hz 的时钟信号源，用于 clcok 的计时

```
----------------------------------------------------------------------------------
--模块名称：clk_gen_1hz
--摘要提示：
--当前版本：1.0.0
--模块作者：
--完成日期：2018 年 12 月 12 日
--内容提要：
--需要注意：
----------------------------------------------------------------------------------
--取代版本：
--模块作者：
--完成日期：
--修改内容：
--修改文件：
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity clk_gen_1hz is
   generic(
    CNT_MAX  : integer := 50000000;--计数最大值
    CNT_HALF : integer := 2500000  --计数一半
   );

    Port (
    rst_n_i : in  STD_LOGIC;
      clk_i : in  STD_LOGIC;
      clk_o : out STD_LOGIC
   );
end clk_gen_1hz;

architecture rtl of clk_gen_1hz is
  signal s_cnt : integer range 0 to CNT_MAX := 0;
begin

  process(clk_i, rst_n_i, s_cnt)
  begin
    if(rst_n_i = '0')then        --复位信号有效时
      s_cnt <= 0;
      clk_o <= '0';
```

```
    elsif rising_edge(clk_i)then  --时钟上升沿时
      if(s_cnt = CNT_MAX)then
        s_cnt <= 0;
        clk_o <= '0';              --拉低
      elsif(s_cnt = CNT_HALF)then
        s_cnt <= s_cnt + 1;
        clk_o <= '1';              --拉高
      else
        s_cnt <= s_cnt + 1;
      end if;
    end if;
  end process;
end rtl;
```

根据分频原理进行代码设计，设计一个 1MHz 的时钟信号源，用于 LCD1602 的扫描。

```
----------------------------------------------------------------------------------
--模块名称: clk_gen_1Mhz
--摘要提示:
--当前版本: 1.0.0
--模块作者:
--完成日期: 2018 年 12 月 12 日
--内容提要:
--需要注意:
----------------------------------------------------------------------------------
--取代版本:
--模块作者:
--完成日期:
--修改内容:
--修改文件:
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--用于产生频率为 1MHz 的时钟信号
entity clk_gen_1MHz is
  generic(
    CNT_MAX  : integer := 50;  --计数最大值
    CNT_HALF : integer := 25   --计数一半
  );

  Port(
    rst_n_i :   in    STD_LOGIC;
    clk_i :     in    STD_LOGIC;
    clk_o :     out   STD_LOGIC
  );
```

```
end clk_gen_1MHz;

architecture rtl of clk_gen_1MHz is
  signal s_cnt   :  integer range 0 to CNT_MAX := 0;

begin
  process(clk_i, rst_n_i, s_cnt)
  begin
    if(rst_n_i = '0')then        --复位信号有效时
      s_cnt <= 0;
      clk_o <= '0';
    elsif rising_edge(clk_i) then --时钟上升沿时
      if (s_cnt = CNT_MAX) then
        s_cnt <= 0;
        clk_o <= '0';                --拉低
      elsif (s_cnt = CNT_HALF) then
        s_cnt <= s_cnt + 1;
        clk_o <= '1';                --拉高
      else
        s_cnt <= s_cnt + 1;
      end if;
    end if;
  end process;
end rtl;
```

秒钟计数器模块-sec_counter，每接收到一次 1Hz 的上升沿信号，秒钟计数器就加 1，直到计数达到 59 时，再接收到 1Hz 信号计数器将清零，同时分钟进位端输出“1”。

```
--------------------------------------------------------------------------------
--模块名称：sec_counter
--摘要提示：
--当前版本：1.0.0
--模块作者：
--完成日期：2018 年 12 月 12 日
--内容提要：
--需要注意：
--------------------------------------------------------------------------------
--取代版本：
--模块作者：
--完成日期：
--修改内容：
--修改文件：
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sec_counter is
    Port (
        clk_i : in  STD_LOGIC;
        rst_n_i : in  STD_LOGIC;
        sec_cnt_o : out  STD_LOGIC_VECTOR (5 downto 0);
        min_carry_o : out  STD_LOGIC
      );
end sec_counter;

architecture rtl of sec_counter is
  signal s_carry : std_logic := '0';
  signal s_sec   : std_logic_vector(5 downto 0) := "000000";

begin
  process(clk_i, rst_n_i)
  begin
    if(rst_n_i = '0')then
      s_carry <= '0';
      s_sec   <= "000000";
    elsif rising_edge(clk_i)then
      if(s_sec = "111011")then
        s_sec <= "000000";          --秒钟清零
        s_carry <= '1';             --向前进位
      else
        s_sec <= s_sec + "000001";  --秒钟加 1
        s_carry <= '0';
      end if;
    end if;
  end process;
  sec_cnt_o <= s_sec;
  min_carry_o <= s_carry;
end rtl;
```

分钟计数器模块-min_counter，每接收到一次分钟进位输出的上升沿信号，分钟计数器就加 1，直到计数达到 59 时，再接收到分钟进位信号计数器将清零，同时时钟进位端输出“1”。

```
--------------------------------------------------------------------------------
--模块名称: min_counter
--摘要提示:
--当前版本: 1.0.0
--模块作者:
```

```
--完成日期：2018年12月12日
--内容提要：
--需要注意：
--------------------------------------------------------------------------------
--取代版本：
--模块作者：
--完成日期：
--修改内容：
--修改文件：
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity min_counter is
    Port (
        clk_i :       in    STD_LOGIC;
        rst_n_i :     in    STD_LOGIC;
        min_cnt_o :   out   STD_LOGIC_VECTOR (5 downto 0);
        hour_carry_o : out  STD_LOGIC
      );
end min_counter;

architecture rtl of min_counter is
  signal s_carry  : std_logic := '0';
  signal s_min  : std_logic_vector(5 downto 0) := "000000";

begin
  process(clk_i, rst_n_i)
  begin
    if(rst_n_i = '0')then
      s_carry <= '0';
      s_min   <= "000000";
    elsif rising_edge(clk_i)then
      if(s_min = "111011")then
        s_min <= "000000";          --分钟清零
        s_carry <= '1';             --向前进位
      else
        s_min <= s_min + "000001"; --分钟加1
        s_carry <= '0';
      end if;
    end if;
  end process;
```

```
  min_cnt_o <= s_min;
  hour_carry_o <= s_carry;
end rtl;
```

时钟计数器模块-hour_counter，每收到一次时钟进位输出的上升沿信号，时钟计数器就加 1，直到计数达到 23 时，在接收到时钟进位信号计数器将清零。

```
--------------------------------------------------------------------------------
--模块名称：hour_counter
--摘要提示：
--当前版本：1.0.0
--模块作者：
--完成日期：2018 年 12 月 12 日
--内容提要：
--需要注意：
--------------------------------------------------------------------------------
--取代版本：
--模块作者：
--完成日期：
--修改内容：
--修改文件：
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity hour_counter is
    Port (
        clk_i : in  STD_LOGIC;
        rst_n_i : in  STD_LOGIC;
        hour_cnt_o : out  STD_LOGIC_VECTOR (5 downto 0)
      );
end hour_counter;

architecture rtl of hour_counter is
  signal s_hour : std_logic_vector(5 downto 0) := "000000";

begin
  process(clk_i, rst_n_i)
  begin
    if(rst_n_i = '0')then
      s_hour <= "000000";
    elsif rising_edge(clk_i)then
      if(s_hour = "10111")then
```

```
      s_hour <= "000000";          --时钟清零
    else
      s_hour <= s_hour + "000001";--时钟加 1
    end if;
  end if;
 end process;
 hour_cnt_o <= s_hour;
end rtl;
```

求商模块-calc_mod，接收计数器输出的信号，转换为整型，求得除以 10 之后的商，将以 BCD 码的格式将其输出。

```
--------------------------------------------------------------------------------
--模块名称：calc_mod
--摘要提示：
--当前版本：1.0.0
--模块作者：
--完成日期：2018 年 12 月 12 日
--内容提要：
--需要注意：
--------------------------------------------------------------------------------
--取代版本：
--模块作者：
--完成日期：
--修改内容：
--修改文件：
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity calc_mod is
   Port (
      data_i : in  STD_LOGIC_VECTOR (5 downto 0);
      data_o : out  STD_LOGIC_VECTOR (3 downto 0)
    );
end calc_mod;

architecture rtl of calc_mod is
 signal s_data : integer range 0 to 59;
 signal s_mod  : integer range 0 to 9 ;
begin

 s_data <= conv_integer(data_i);
```

```
  process(s_data)
  begin
    if  (s_data >= 0  and s_data <= 9) then
      s_mod <= 0;
    elsif (s_data >= 10 and s_data <= 19) then
      s_mod <= 1;
    elsif (s_data >= 20 and s_data <= 29) then
      s_mod <= 2;
    elsif (s_data >= 30 and s_data <= 39) then
      s_mod <= 3;
    elsif (s_data >= 40 and s_data <= 49) then
      s_mod <= 4;
    elsif (s_data >= 50 and s_data <= 59) then
      s_mod <= 5;
    end if;
  end process;
    data_o <= conv_std_logic_vector(s_mod,4); --转换成 std_logic_vector

end rtl;
```

求模模块-calc_rem，接收计数器输出的信号，转换为整型，求得除以 10 之后的余数，并将其以 BCD 码的格式输出。

```
----------------------------------------------------------------------------
--模块名称：calc_rem
--摘要提示：
--当前版本：1.0.0
--模块作者：
--完成日期：2018 年 12 月 12 日
--内容提要：
--需要注意：
----------------------------------------------------------------------------
--取代版本：
--模块作者：
--完成日期：
--修改内容：
--修改文件：
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity calc_rem is
```

```
    Port (
        data_i : in  STD_LOGIC_VECTOR (5 downto 0);
        data_o : out  STD_LOGIC_VECTOR (3 downto 0)
       );
end calc_rem;

architecture rtl of calc_rem is
  signal s_data : integer range 0 to 59;
  signal s_rem : integer range 0 to 9;

begin
  s_data <= conv_integer(data_i);

  process(data_i)
  begin
    if    (s_data >= 0  and s_data <= 9) then
      s_rem <= s_data;
    elsif (s_data >= 10 and s_data <= 19) then
      s_rem <= s_data - 10;
    elsif (s_data >= 20 and s_data <= 29) then
      s_rem <= s_data - 20;
    elsif (s_data >= 30 and s_data <= 39) then
      s_rem <= s_data - 30;
    elsif (s_data >= 40 and s_data <= 49) then
      s_rem <= s_data - 40;
    elsif (s_data >= 50 and s_data <= 59) then
      s_rem <= s_data - 50;
    end if;
  end process;
  data_o <= conv_std_logic_vector(s_rem,4);
end rtl;
```

LCD1602 显示模块-LCD1602_Disp，该模块采用了状态机，将

```
--------------------------------------------------------------------------------
--模块名称：LCD1602_Disp
--摘要提示：
--当前版本：1.0.0
--模块作者：
--完成日期：2018 年 12 月 12 日
--内容提要：
--需要注意：
--------------------------------------------------------------------------------
--取代版本：
--模块作者：
```

```
--完成日期:
--修改内容:
--修改文件:
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity LCD1602_Disp is
  generic(
    LCD_COM   : STD_LOGIC := '0';
    LCD_DATA  : STD_LOGIC := '1';
    LCD_READ  : STD_LOGIC := '1';
    LCD_WRITE : STD_LOGIC := '0';
    LCD_LOW   : STD_LOGIC := '0';
    LCD_HIGH  : STD_LOGIC := '1'
  );

  port(
    clk_50mhz_i : in  std_logic;
    rst_n_i     : in  std_logic;
   disp_hour_h_i : in std_logic_vector(3 downto 0);
   disp_hour_l_i : in std_logic_vector(3 downto 0);
   disp_min_h_i : in std_logic_vector(3 downto 0);
   disp_min_l_i : in std_logic_vector(3 downto 0);
   disp_sec_h_i : in std_logic_vector(3 downto 0);
   disp_sec_l_i : in std_logic_vector(3 downto 0);
    lcd_rs_o    : out std_logic;
    lcd_wr_o    : out std_logic;
    lcd_en_o    : out std_logic;
    lcd_data_o  : out std_logic_vector(7 downto 0)
  );
end LCD1602_Disp;

Architecture rtl of LCD1602_Disp is
  component clk_gen_1MHz is
    port(
      rst_n_i : in  std_logic;
      clk_i   : in  std_logic;
      clk_o   : out std_logic
    );
  end component;
```

```
function char_to_integer(data : character) return integer is
  variable rlt : integer range 32 to 125;
begin
  case data is
    when ' ' =>    rlt := 32;
    when '!' =>    rlt := 33;
    --when '"' =>    rlt := 34;
    when '#' =>    rlt := 35;
    when '$' =>    rlt := 36;
    when '%' =>    rlt := 37;
    when '&' =>    rlt := 38;
    when ''' =>    rlt := 39;
    when '(' =>    rlt := 40;
    when ')' =>    rlt := 41;
    when '*' =>    rlt := 42;
    when '+' =>    rlt := 43;
    when ',' =>    rlt := 44;
    when '-' =>    rlt := 45;
    when '.' =>    rlt := 46;
    when '/' =>    rlt := 47;
    when '0' =>    rlt := 48;
    when '1' =>    rlt := 49;
    when '2' =>    rlt := 50;
    when '3' =>    rlt := 51;
    when '4' =>    rlt := 52;
    when '5' =>    rlt := 53;
    when '6' =>    rlt := 54;
    when '7' =>    rlt := 55;
    when '8' =>    rlt := 56;
    when '9' =>    rlt := 57;
    when ':' =>    rlt := 58;
    when ';' =>    rlt := 59;
    when '<' =>    rlt := 60;
    when '=' =>    rlt := 61;
    when '>' =>    rlt := 62;
    when '?' =>    rlt := 63;
    when '@' =>    rlt := 64;
    when 'A' =>    rlt := 65;
    when 'B' =>    rlt := 66;
    when 'C' =>    rlt := 67;
    when 'D' =>    rlt := 68;
    when 'E' =>    rlt := 69;
    when 'F' =>    rlt := 70;
    when 'G' =>    rlt := 71;
```

```
    when 'H' =>    rlt := 72;
    when 'I' =>    rlt := 73;
    when 'J' =>    rlt := 74;
    when 'K' =>    rlt := 75;
    when 'L' =>    rlt := 76;
    when 'M' =>    rlt := 77;
    when 'N' =>    rlt := 78;
    when 'O' =>    rlt := 79;
    when 'P' =>    rlt := 80;
    when 'Q' =>    rlt := 81;
    when 'R' =>    rlt := 82;
    when 'S' =>    rlt := 83;
    when 'T' =>    rlt := 84;
    when 'U' =>    rlt := 85;
    when 'V' =>    rlt := 86;
    when 'W' =>    rlt := 87;
    when 'X' =>    rlt := 88;
    when 'Y' =>    rlt := 89;
    when 'Z' =>    rlt := 90;
    when '[' =>    rlt := 91;
    when '\' =>    rlt := 92;
    when ']' =>    rlt := 93;
    when '^' =>    rlt := 94;
    when '_' =>    rlt := 95;
    when '`' =>    rlt := 96;
    when 'a' =>    rlt := 97;
    when 'b' =>    rlt := 98;
    when 'c' =>    rlt := 99;
    when 'd' =>    rlt := 100;
    when 'e' =>    rlt := 101;
    when 'f' =>    rlt := 102;
    when 'g' =>    rlt := 103;
    when 'h' =>    rlt := 104;
    when 'i' =>    rlt := 105;
    when 'j' =>    rlt := 106;
    when 'k' =>    rlt := 107;
    when 'l' =>    rlt := 108;
    when 'm' =>    rlt := 109;
    when 'n' =>    rlt := 110;
    when 'o' =>    rlt := 111;
    when 'p' =>    rlt := 112;
    when 'q' =>    rlt := 113;
    when 'r' =>    rlt := 114;
    when 's' =>    rlt := 115;
```

```
      when 't' =>    rlt := 116;
      when 'u' =>    rlt := 117;
      when 'v' =>    rlt := 118;
      when 'w' =>    rlt := 119;
      when 'x' =>    rlt := 120;
      when 'y' =>    rlt := 121;
      when 'z' =>    rlt := 122;
      when '{' =>    rlt := 123;
      when '|' =>    rlt := 124;
      when '}' =>    rlt := 125;
      when others => rlt := 32;
    end case;

    return rlt;
  end char_to_integer;

  type state is(s_rst, s_38H, s_08H, s_01H, s_06H, s_0CH, set_addr, write_data);
  signal pr_state, nx_state : state;

  signal s_cnt     : INTEGER range 0 to 15000 := 0;
  signal s_cnt_max : INTEGER range 0 to 15000 := 0;

  signal s_clk_1MHz : std_logic := '0';

  signal s_addr_cnt : INTEGER range 0 to 31 := 0;
  signal s_start_addr_cnt : std_logic := LCD_LOW;
  signal s_lcd_ddram_addr : std_logic_vector(7 downto 0) := "10000000";
  signal s_lcd_ddram_data : std_logic_vector(7 downto 0) := "00100000";

  signal s_hour_h : std_logic_vector(7 downto 0);
  signal s_hour_l : std_logic_vector(7 downto 0);
  signal s_min_h : std_logic_vector(7 downto 0);
  signal s_min_l : std_logic_vector(7 downto 0);
  signal s_sec_h : std_logic_vector(7 downto 0);
  signal s_sec_l : std_logic_vector(7 downto 0);

begin
  U_Clk_gen_1MHz : clk_gen_1MHz
  port map(
    rst_n_i => rst_n_i,
    clk_i   => clk_50mhz_i,
    clk_o   => s_clk_1MHz
  );
------------------lower section---------------------
```

```
  process(s_clk_1MHz, rst_n_i)

  begin
    if(rst_n_i = '0')then
      pr_state <= s_rst;
    elsif (s_clk_1MHz'event and s_clk_1MHz = '1')then
      pr_state <= nx_state;
    end if;
  end process;
------------------upper section----------------------
  process(pr_state, s_cnt, s_clk_1MHz)

  begin
    --default values

    case pr_state is
      when s_rst      =>                  --起始状态
      nx_state         <= s_rst;
      lcd_wr_o         <= LCD_WRITE;
      lcd_rs_o         <= LCD_DATA;
      lcd_data_o       <= "11111111";
      lcd_en_o         <= LCD_LOW;
      s_cnt_max        <= 0;
      s_start_addr_cnt  <= LCD_LOW;
        nx_state <= s_38H;
      when s_38H      =>                   --设定 8 位数据总线
        s_cnt_max <= 15000;
        if(s_cnt < s_cnt_max - 3)then
          nx_state <= s_38H;
        elsif(s_cnt = s_cnt_max - 3)then
          lcd_rs_o <= LCD_COM;
          lcd_data_o <= "00111000";
          nx_state <= s_38H;
        elsif(s_cnt = s_cnt_max - 2)then
          lcd_rs_o <= LCD_COM;
          lcd_data_o <= "00111000";
          lcd_en_o <= LCD_HIGH;
          nx_state <= s_38H;
        elsif(s_cnt = s_cnt_max - 1)then
          lcd_rs_o <= LCD_COM;
          lcd_data_o <= "00111000";
          --lcd_en_o <= LCD_LOW;
          nx_state <= s_38H;
        elsif(s_cnt = s_cnt_max)then
```

```
        nx_state <= s_08H;
      end if;

    when s_08H      =>                  --显示开总命令
      s_cnt_max <= 40;
      if(s_cnt < s_cnt_max - 3)then
        nx_state <= s_08H;
      elsif(s_cnt = s_cnt_max - 3)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= "00001000";
        nx_state <= s_08H;
      elsif(s_cnt = s_cnt_max - 2)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= "00001000";
        lcd_en_o <= LCD_HIGH;
        nx_state <= s_08H;
      elsif(s_cnt = s_cnt_max - 1)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= "00001000";
        --lcd_en_o <= LCD_LOW;
        nx_state <= s_08H;
      elsif(s_cnt = s_cnt_max)then
        nx_state <= s_01H;
      end if;

    when s_01H      =>                  --清屏指令
      s_cnt_max <= 40;
      if(s_cnt < s_cnt_max - 3)then
        nx_state <= s_01H;
      elsif(s_cnt = s_cnt_max - 3)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= "00000001";
        nx_state <= s_01H;
      elsif(s_cnt = s_cnt_max - 2)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= "00000001";
        lcd_en_o <= LCD_HIGH;
        nx_state <= s_01H;
      elsif(s_cnt = s_cnt_max - 1)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= "00000001";
        --lcd_en_o <= LCD_LOW;
        nx_state <= s_01H;
      elsif(s_cnt = s_cnt_max)then
```

```
        nx_state <= s_06H;
      end if;

    when s_06H     =>                --光标右移 1 位
      s_cnt_max <= 1640;
      if(s_cnt < s_cnt_max - 3)then
        nx_state <= s_06H;
      elsif(s_cnt = s_cnt_max - 3)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= "00000110";
        nx_state <= s_06H;
      elsif(s_cnt = s_cnt_max - 2)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= "00000110";
        lcd_en_o <= LCD_HIGH;
        nx_state <= s_06H;
      elsif(s_cnt = s_cnt_max - 1)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= "00000110";
        --lcd_en_o <= LCD_LOW;
        nx_state <= s_06H;
      elsif(s_cnt = s_cnt_max)then
        nx_state <= s_0CH;
      end if;

    when s_0CH     =>                --整体显示屏开
      s_cnt_max <= 40;
      if(s_cnt < s_cnt_max - 3)then
        nx_state <= s_0CH;
      elsif(s_cnt = s_cnt_max - 3)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= "00001100";
        nx_state <= s_0CH;
      elsif(s_cnt = s_cnt_max - 2)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= "00001100";
        lcd_en_o <= LCD_HIGH;
        nx_state <= s_0CH;
      elsif(s_cnt = s_cnt_max - 1)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= "00001100";
        --lcd_en_o <= LCD_LOW;
        nx_state <= s_0CH;
      elsif(s_cnt = s_cnt_max)then
```

```
        nx_state <= set_addr;
      end if;

    when set_addr   =>                  --设置光标地址
      s_cnt_max <= 40;
      if(s_cnt < s_cnt_max - 3)then
        nx_state <= set_addr;
      elsif(s_cnt = s_cnt_max - 3)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= s_lcd_ddram_addr;
        nx_state <= set_addr;
      elsif(s_cnt = s_cnt_max - 2)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= s_lcd_ddram_addr;
        lcd_en_o <= LCD_HIGH;
        nx_state <= set_addr;
      elsif(s_cnt = s_cnt_max - 1)then
        lcd_rs_o <= LCD_COM;
        lcd_data_o <= s_lcd_ddram_addr;
        --lcd_en_o <= LCD_LOW;
        nx_state <= set_addr;
      elsif(s_cnt = s_cnt_max)then
        nx_state <= write_data;
      end if;

    when write_data =>                  --数据线写数据
      s_cnt_max <= 40;
      if(s_cnt < s_cnt_max - 3)then
        nx_state <= write_data;
      elsif(s_cnt = s_cnt_max - 3)then
        lcd_rs_o <= LCD_DATA;
        lcd_data_o <= s_lcd_ddram_data;
        nx_state <= write_data;
      elsif(s_cnt = s_cnt_max - 2)then
        lcd_rs_o <= LCD_DATA;
        lcd_data_o <= s_lcd_ddram_data;
        lcd_rs_o <= LCD_DATA;
        lcd_en_o <= LCD_HIGH;
        nx_state <= write_data;
      elsif(s_cnt = s_cnt_max - 1)then
        lcd_rs_o <= LCD_DATA;
        lcd_data_o <= s_lcd_ddram_data;
        lcd_rs_o <= LCD_DATA;
        --lcd_en_o <= LCD_LOW;
```

```
        nx_state <= write_data;
      elsif(s_cnt = s_cnt_max)then
        nx_state <= set_addr;
        s_start_addr_cnt <= LCD_HIGH;
      end if;

    end case;

  end process;

-----------------cnt_process-----------------------
  process(rst_n_i, s_cnt, s_clk_1MHz)

  begin
    if(rst_n_i = '0')then
      s_cnt <= 0;
    elsif(s_clk_1MHz'event and s_clk_1MHz = '1')then
      if(s_cnt < s_cnt_max)then
        s_cnt <= s_cnt + 1;
      elsif(s_cnt >= s_cnt_max)then
        s_cnt <= 0;
      end if;
    end if;
  end process;

-------------------------------------------------
  process(rst_n_i, s_start_addr_cnt, s_clk_1MHz, s_addr_cnt)

  begin
    if(rst_n_i = '0')then
      s_addr_cnt <= 0;
    elsif(s_start_addr_cnt = '1')then
      if(s_clk_1MHz'event and s_clk_1MHz = '1')then
        if(s_addr_cnt >= 32)then
          s_addr_cnt <= 0;
        else
          s_addr_cnt <= s_addr_cnt + 1;
        end if;
      end if;
    end if;
  end process;

-------------------------------------------------
  process(s_addr_cnt)
```

```
begin
  if(s_addr_cnt >= 0 and s_addr_cnt <= 15)then
    s_lcd_ddram_addr <= "10000000" + conv_std_logic_vector(s_addr_cnt, 8);
  elsif(s_addr_cnt >= 16 and s_addr_cnt <= 31)then
    s_lcd_ddram_addr <= "11000000" + conv_std_logic_vector(s_addr_cnt - 16, 8);
  end if;
end process;

s_hour_h <= "0011" & disp_hour_h_i;
s_hour_l <= "0011" & disp_hour_l_i;
s_min_h <=  "0011" & disp_min_h_i;
s_min_l <=  "0011" & disp_min_l_i;
s_sec_h <=  "0011" & disp_sec_h_i;
s_sec_l <=  "0011" & disp_sec_l_i;

-------------------------------------------------------
process(s_addr_cnt)

begin
  case s_addr_cnt is
    when 0 =>  s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 1 =>  s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 2 =>  s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 3 =>  s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 4 =>  s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer('S'),8);
    when 5 =>  s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer('Z'),8);
    when 6 =>  s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer('U'),8);
    when 7 =>  s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 8 =>  s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 9 =>  s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer('C'),8);
    when 10 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer('H'),8);
    when 11 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer('X'),8);
    when 12 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 13 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 14 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 15 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 16 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 17 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 18 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 19 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
    when 20 => s_lcd_ddram_data <= s_hour_h;  --动态显示时钟高位
    when 21 => s_lcd_ddram_data <= s_hour_l;  --动态显示时钟低位
    when 22 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer('-'),8);
```

```
      when 23 => s_lcd_ddram_data <= s_min_h;  --动态显示分钟高位
      when 24 => s_lcd_ddram_data <= s_min_l; --动态显示分钟低位
      when 25 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer('-'),8);
      when 26 => s_lcd_ddram_data <= s_sec_h;  --动态显示秒钟高位
      when 27 => s_lcd_ddram_data <= s_sec_l;  --动态显示秒钟低位
      when 28 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
      when 29 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
      when 30 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
      when 31 => s_lcd_ddram_data <= conv_std_logic_vector(char_to_integer(' '),8);
      when others => s_lcd_ddram_data <= "00100000";
    end case;
  end process;
-------------------------------------------------
end rtl;
```

clock 模块，将各个小部件组装起来，形成一个 LCD1602 时钟。

```
--------------------------------------------------------------------------------
--模块名称：clock
--摘要提示：
--当前版本：1.0.0
--模块作者：
--完成日期：2018 年 12 月 12 日
--内容提要：
--需要注意：
--------------------------------------------------------------------------------
--取代版本：
--模块作者：
--完成日期：
--修改内容：
--修改文件：
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity clock is
  port(
    clk_50mhz_i : in std_logic;
    rst_n_i : in std_logic;
    lcd_rs_o  : out  std_logic;
    lcd_wr_o  : out  std_logic;
    lcd_en_o  : out  std_logic;
    lcd_data_o  : out  std_logic_vector(7 downto 0)
  );
end clock;
```

```
architecture rtl of clock is
  component clk_gen_1hz is
    port(
      rst_n_i :   in    STD_LOGIC;
      clk_i :     in    STD_LOGIC;
      clk_o :     out    STD_LOGIC
    );
  end component;

  component sec_counter is
    port(
      clk_i : in  STD_LOGIC;
      rst_n_i : in  STD_LOGIC;
      sec_cnt_o : out  STD_LOGIC_VECTOR (5 downto 0);
      min_carry_o : out  STD_LOGIC
    );
  end component;

  component min_counter is
    port(
      clk_i :      in    STD_LOGIC;
      rst_n_i :    in    STD_LOGIC;
      min_cnt_o :   out    STD_LOGIC_VECTOR (5 downto 0);
      hour_carry_o : out   STD_LOGIC
    );
  end component;

  component hour_counter is
    port(
      clk_i : in  STD_LOGIC;
      rst_n_i : in  STD_LOGIC;
      hour_cnt_o : out  STD_LOGIC_VECTOR (5 downto 0)
    );
  end component;

  component calc_mod is
    port(
      data_i : in  STD_LOGIC_VECTOR (5 downto 0);
      data_o : out  STD_LOGIC_VECTOR (3 downto 0)
    );
  end component;

  component calc_rem is
```

```
    port(
      data_i : in  STD_LOGIC_VECTOR (5 downto 0);
      data_o : out  STD_LOGIC_VECTOR (3 downto 0)
    );
  end component;

  component LCD1602_Disp is
  port(
    clk_50mhz_i : in  std_logic;
    rst_n_i     : in  std_logic;
    disp_hour_h_i : in std_logic_vector(3 downto 0);
    disp_hour_l_i : in std_logic_vector(3 downto 0);
    disp_min_h_i : in std_logic_vector(3 downto 0);
    disp_min_l_i : in std_logic_vector(3 downto 0);
    disp_sec_h_i : in std_logic_vector(3 downto 0);
    disp_sec_l_i : in std_logic_vector(3 downto 0);
    lcd_rs_o    : out std_logic;
    lcd_wr_o    : out std_logic;
    lcd_en_o    : out std_logic;
    lcd_data_o  : out std_logic_vector(7 downto 0)
  );
  end component;

  signal s_clk_1hz : std_logic;
  signal s_1hz  : std_logic;
  signal s_1min  :  std_logic;
  signal s_1hour :  std_logic;
  signal s_calc_sec : std_logic_vector(5 downto 0);
  signal s_calc_min : std_logic_vector(5 downto 0);
  signal s_calc_hour : std_logic_vector(5 downto 0);
  signal s_sec_h : std_logic_vector(3 downto 0);
  signal s_sec_l : std_logic_vector(3 downto 0);
  signal s_min_h : std_logic_vector(3 downto 0);
  signal s_min_l : std_logic_vector(3 downto 0);
  signal s_hour_h : std_logic_vector(3 downto 0);
  signal s_hour_l : std_logic_vector(3 downto 0);

begin
  i_clk_gen_1hz : clk_gen_1hz
  port map(
    rst_n_i => rst_n_i,
    clk_i   => clk_50mhz_i,
    clk_o   => s_clk_1hz
  );
```

```
i_sec_counter : sec_counter
port map(
  clk_i      => s_clk_1hz,
  rst_n_i    => rst_n_i,
  sec_cnt_o  => s_calc_sec,
  min_carry_o => s_1min
);

i_min_counter : min_counter
port map(
  clk_i => s_1min,
  rst_n_i => rst_n_i,
  min_cnt_o => s_calc_min,
  hour_carry_o => s_1hour
);

i_hour_counter : hour_counter
port map(
  clk_i => s_1hour,
  rst_n_i => rst_n_i,
  hour_cnt_o => s_calc_hour
);

i_sec_mod : calc_mod
port map(
  data_i => s_calc_sec,
  data_o => s_sec_h
);

i_min_mod : calc_mod
port map(
  data_i => s_calc_min,
  data_o => s_min_h
);

i_hour_mod : calc_mod
port map(
  data_i => s_calc_hour,
  data_o => s_hour_h
);

i_sec_rem : calc_rem
port map(
```

```
    data_i => s_calc_sec,
    data_o => s_sec_l
  );

  i_min_rem : calc_rem
  port map(
    data_i => s_calc_min,
    data_o => s_min_l
  );

  i_hour_rem : calc_rem
  port map(
    data_i => s_calc_hour,
    data_o => s_hour_l
  );

  i_LCD1602_Disp : LCD1602_Disp
  port map(
    clk_50mhz_i => clk_50mhz_i,
    rst_n_i     => rst_n_i,
    disp_hour_h_i => s_hour_h,
    disp_hour_l_i => s_hour_l,
    disp_min_h_i => s_min_h,
    disp_min_l_i => s_min_l,
    disp_sec_h_i => s_sec_h,
    disp_sec_l_i => s_sec_l,
    lcd_rs_o    => lcd_rs_o,
    lcd_wr_o    => lcd_wr_o,
    lcd_en_o    => lcd_en_o,
    lcd_data_o  => lcd_data_o
  );
end rtl;
```

六、综合、仿真结果及分析：

RTL 级电路综合：

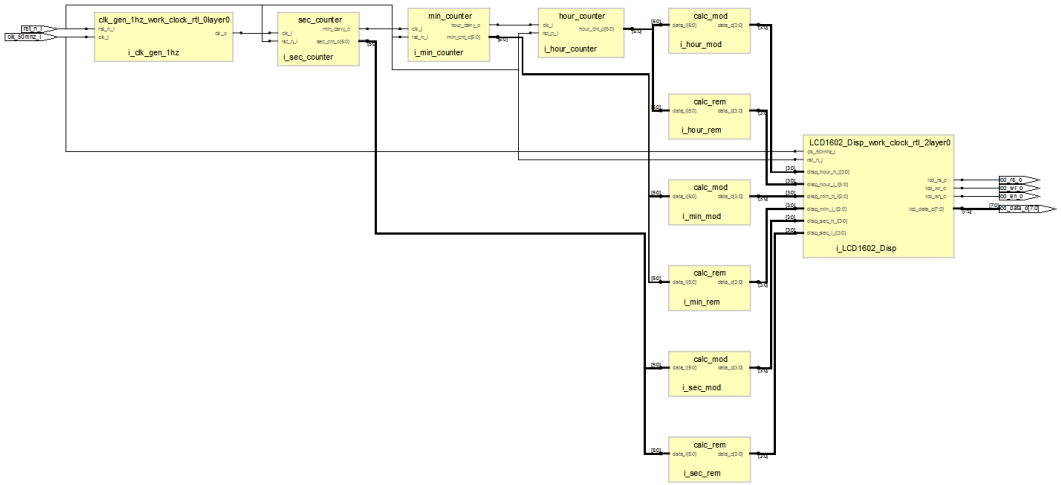


图 6-1 整体 RTL 级电路

1Hz 时钟信号发生器电路图：

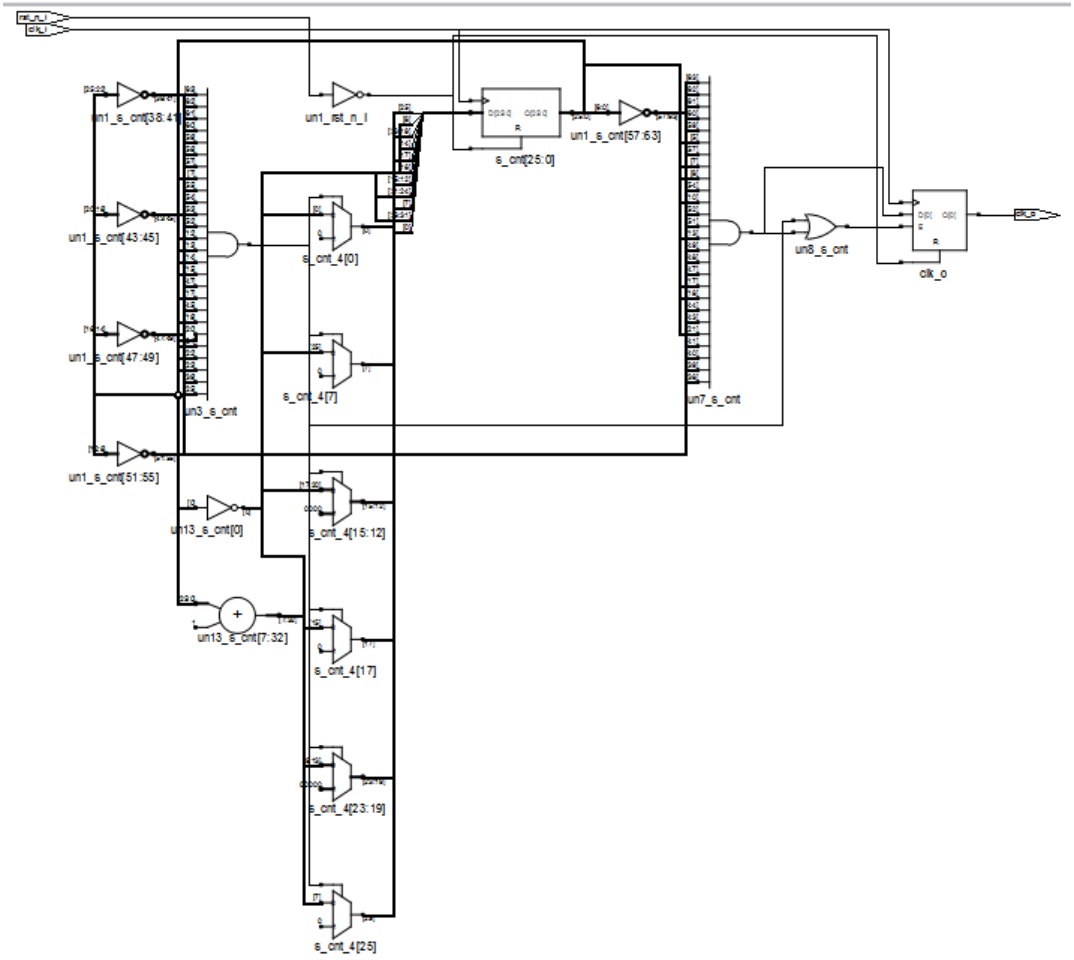


图 6-2　1Hz 时钟信号产生电路

秒钟计数器电路图：

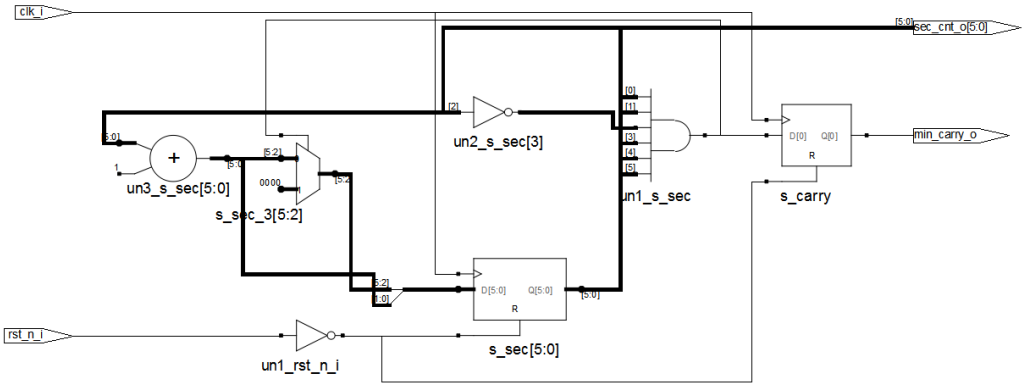


图 6-3 秒钟计数器电路

分钟计数器电路图：

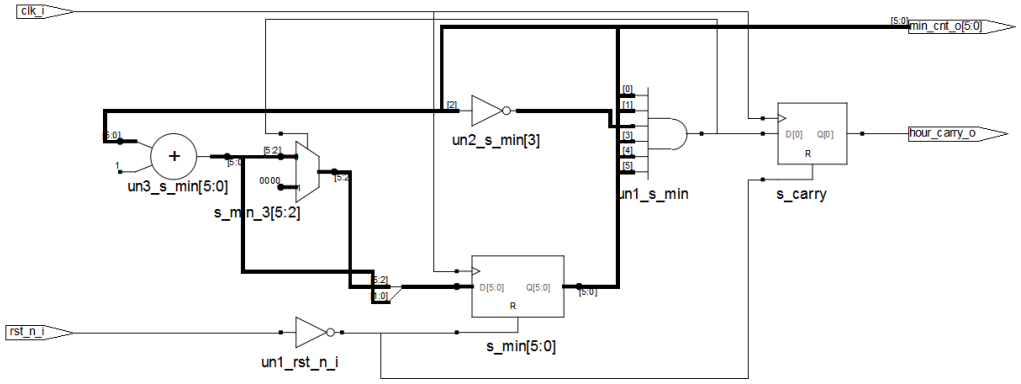


图 6-4 分钟计数器电路

时钟计数器电路图：

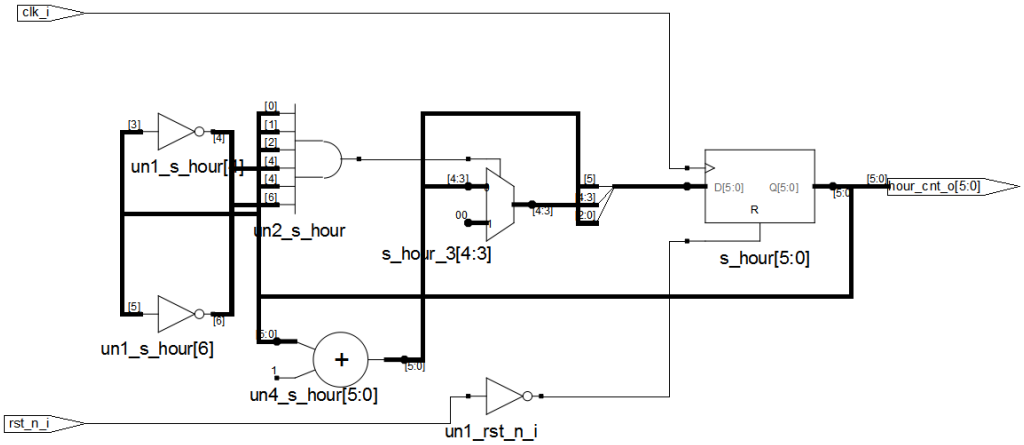


图 6-5 时钟计数器电路

求商模块电路图：

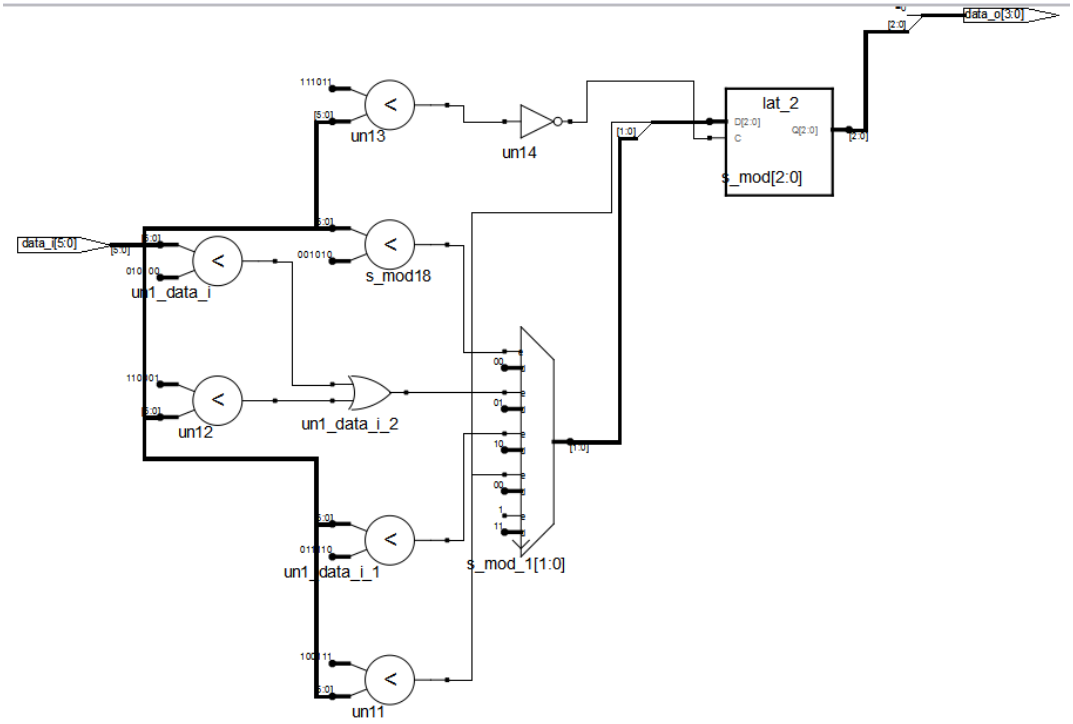


图 6-6 求商模块电路

求余模块电路图：

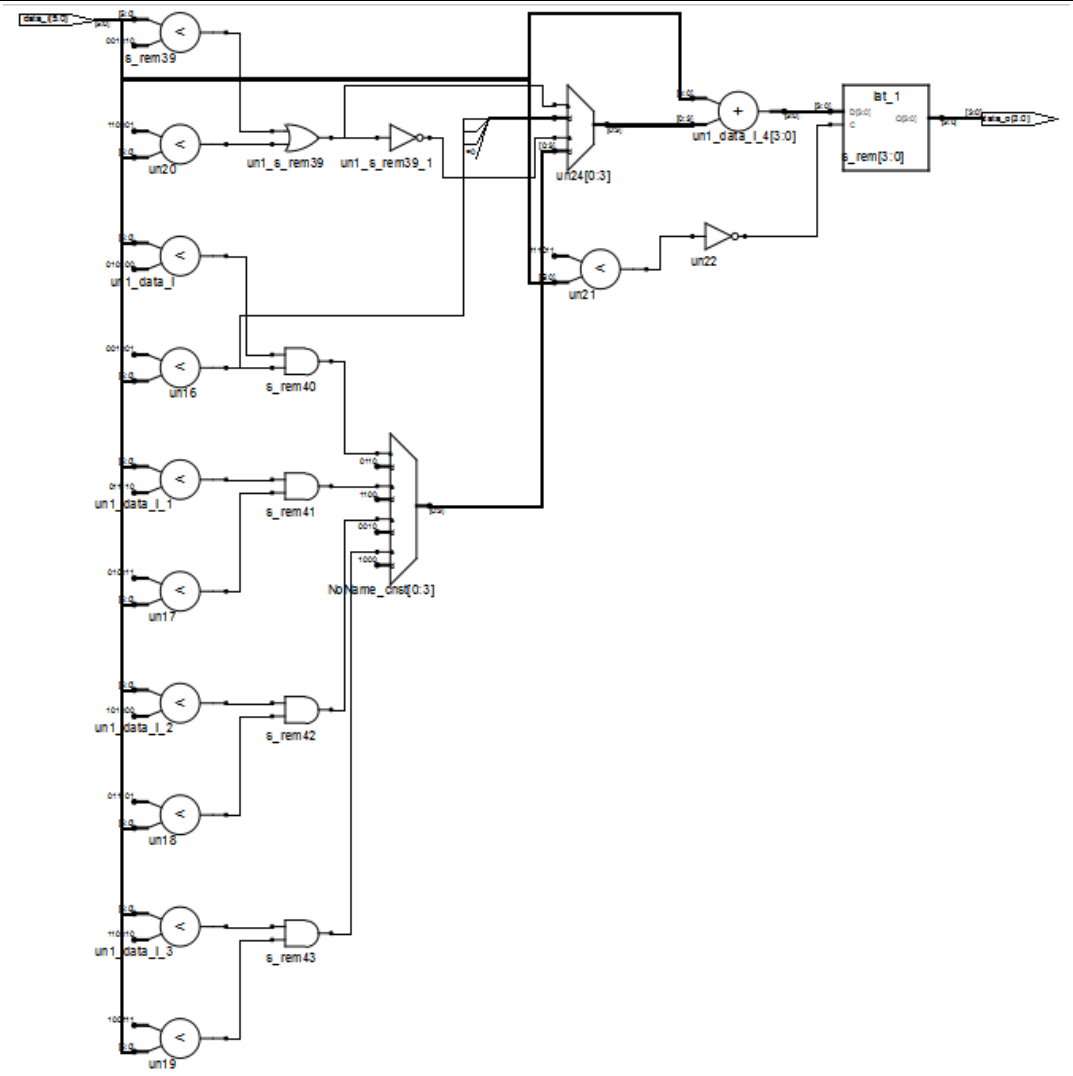


图 6-7 求余模块电路

2、电路仿真

测试激励 Test Bench 代码如下：

```
----------------------------------------------------------------------------------
--模块名称：LCD1602_Disp_tb
--摘要提示：LCD1602_Disp 测试激励
--当前版本：1.0.0
--模块作者：
--完成日期：2018 年 12 月 12 日
--内容提要：
--需要注意：
----------------------------------------------------------------------------------
--取代版本：
--模块作者：
--完成日期：
--修改内容：
--修改文件：
----------------------------------------------------------------------------------
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY LCD1602_Disp_tb IS
END LCD1602_Disp_tb;

ARCHITECTURE behavior OF LCD1602_Disp_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT LCD1602_Disp
    PORT(
         clk_50mhz_i : IN  std_logic;
         rst_n_i : IN  std_logic;
         disp_hour_h_i : IN  std_logic_vector(3 downto 0);
         disp_hour_l_i : IN  std_logic_vector(3 downto 0);
         disp_min_h_i : IN  std_logic_vector(3 downto 0);
         disp_min_l_i : IN  std_logic_vector(3 downto 0);
         disp_sec_h_i : IN  std_logic_vector(3 downto 0);
         disp_sec_l_i : IN  std_logic_vector(3 downto 0);
         lcd_rs_o : OUT  std_logic;
         lcd_wr_o : OUT  std_logic;
         lcd_en_o : OUT  std_logic;
         lcd_data_o : OUT  std_logic_vector(7 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk_50mhz_i : std_logic := '0';
   signal rst_n_i : std_logic := '0';
   signal disp_hour_h_i : std_logic_vector(3 downto 0) := (others => '0');
   signal disp_hour_l_i : std_logic_vector(3 downto 0) := (others => '0');
   signal disp_min_h_i : std_logic_vector(3 downto 0) := (others => '0');
   signal disp_min_l_i : std_logic_vector(3 downto 0) := (others => '0');
   signal disp_sec_h_i : std_logic_vector(3 downto 0) := (others => '0');
   signal disp_sec_l_i : std_logic_vector(3 downto 0) := (others => '0');

  --Outputs
   signal lcd_rs_o : std_logic;
   signal lcd_wr_o : std_logic;
   signal lcd_en_o : std_logic;
   signal lcd_data_o : std_logic_vector(7 downto 0);
```

```
   -- Clock period definitions
   constant clk_50mhz_i_period : time := 10 ns;

BEGIN

  -- Instantiate the Unit Under Test (UUT)
   uut: LCD1602_Disp PORT MAP (
          clk_50mhz_i => clk_50mhz_i,
          rst_n_i => rst_n_i,
          disp_hour_h_i => disp_hour_h_i,
          disp_hour_l_i => disp_hour_l_i,
          disp_min_h_i => disp_min_h_i,
          disp_min_l_i => disp_min_l_i,
          disp_sec_h_i => disp_sec_h_i,
          disp_sec_l_i => disp_sec_l_i,
          lcd_rs_o => lcd_rs_o,
          lcd_wr_o => lcd_wr_o,
          lcd_en_o => lcd_en_o,
          lcd_data_o => lcd_data_o
        );

   -- Clock process definitions
   clk_50mhz_i_process :process
   begin
    clk_50mhz_i <= '0';
    wait for clk_50mhz_i_period/2;
    clk_50mhz_i <= '1';
    wait for clk_50mhz_i_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;

      wait for clk_50mhz_i_period*10;

    rst_n_i <= '1';
      -- insert stimulus here

      wait;
   end process;
```

```
END;
```

仿真结果如图所示：

lcd_data_o 总线输出在发生变化。

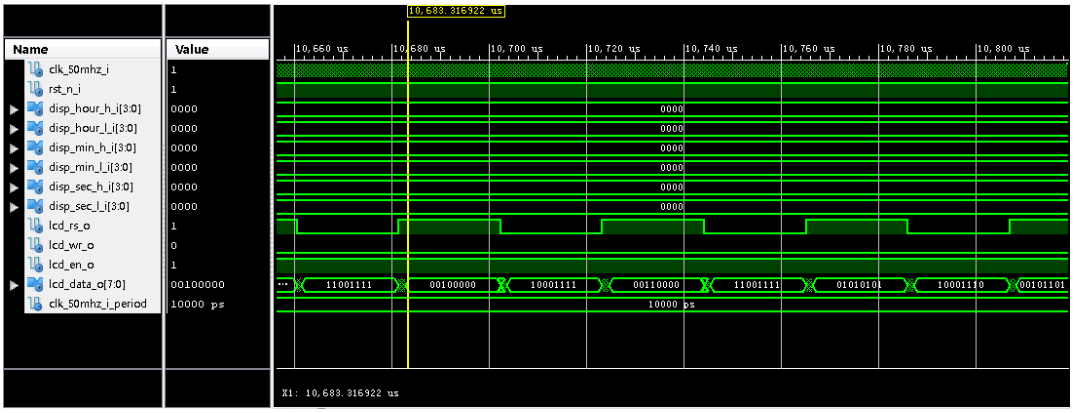


图 6-8 仿真结果

七、FPGA 板级验证及结果分析

1、在工程中加入引脚约束文件，代码如下

```
##-------------------------------------------------------------------
##模块名称: clock.ucf
##摘要提示:
##当前版本: 1.0.0
##模块作者:
##完成日期: 2018 年 12 月 12 日
##内容提要:
##需要注意:
##-------------------------------------------------------------------
##取代版本:
##模块作者:
##完成日期:
##修改内容:
##修改文件:
##-------------------------------------------------------------------
Net "clk_50mhz_i" LOC=P129; --系统主频
Net "rst_n_i"     LOC=P85;    --复位信号线
Net "lcd_rs_o"    LOC=P51;
Net "lcd_wr_o"    LOC=P50;
Net "lcd_en_o"    LOC=P44;
Net "lcd_data_o[0]" LOC=P43;
Net "lcd_data_o[1]" LOC=P39;
Net "lcd_data_o[2]" LOC=P35;
Net "lcd_data_o[3]" LOC=P34;
Net "lcd_data_o[4]" LOC=P33;
Net "lcd_data_o[5]" LOC=P32;
```

```
Net "lcd_data_o[6]" LOC=P31;
Net "lcd_data_o[7]" LOC=P26;
```

综合之后生成 bit 文件烧进实验箱中观察到如下图所示的现象：

LCD1602 上的时钟可以正常显示并运行。时分秒之间的进位也正常。

图 7-1 现象（1）

图 7-2 现象（2）

图 7-3 现象（3）

八、实验总结：

通过这次实验我掌握了 1602LCD 的工作原理以及状态机的使用方法，明白了使用状态机极大程度地方便了我们进行时序设计。并且能够使用 1602LCD 进行显示。在设计 LCD1602 时钟的时候，关于时钟计数的模块（时钟模块、分钟模块、秒钟模块、求商模块、求模模块）都是之间使用之前七段数码管完成的模块，十分方便，这使我更加理解了模块重用的好处，在设计模块的时候，应该尽量考虑可重用性和可移植性。

指导教师批阅意见：

成绩评定：

指导教师签字：

年　　月　　日

备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。