

第八章 CPLD 器件应用

随着生产工艺的逐步提高以及 CPLD 开发系统的不断完善, CPLD 器件容量也由几百门飞速发展到百万门以上,使得一个复杂数字系统完全可以在一个芯片中实现。HDL 语言的使用,以及强大的 EDA 综合仿真工具,使 CPLD 的开发形象、快捷,特别是 MAX+PLUS 10.0 和 QUARTUS 2.0 以上版本,图形输入和文本输入可以方便地结合在一起,错误定位使用方便,极大地加快了开发速度。因此 CPLD 在数字系统、微机系统、控制系统以及通信系统中或得广泛应用。

8.1 CPLD 在数字系统中的应用

CPLD 器件由于本身是一种纯数字器件,因此中小规模数字器件完成的设计可以完全由一片 CPLD 器件所取代。CPLD 在数字系统中的典型应用主要有数字中的设计、数字频率计的设计、A/D 及 D/A 转换时序设计、键盘扫描与去抖动设计等。下面以 0.1Hz-50MHz 的频率计为例介绍 CPLD 器件在频率计设计中的应用。

- 1、设计内容:频率采样、16 进制转换成 BCD 码、数码管显示。
- 2、开发系统:MAX+PLUS 10.0
- 3、使用的器件:ACEX EP1K100QC208-3
- 4、实验系统:CPLDEE-4 实验开发系统

- (1) 频率采样:由于测量的范围较宽,因此分成两种处理方式:频率大于等于 1000Hz 的采用直接计数方式,即在一秒内对被测频率所计的数即得频率值。等频率小于 1000Hz 时采用脉宽测量的方式,即首先将被测信号进行二分频,使得被测信号的正负脉宽相等,然后利用正脉宽对 1MHz 的标准频率进行采样,频率值 $f=100\text{MHz}/\text{采样值}$,这样可以得到小数点后两位的值。如图 8-1 所示:

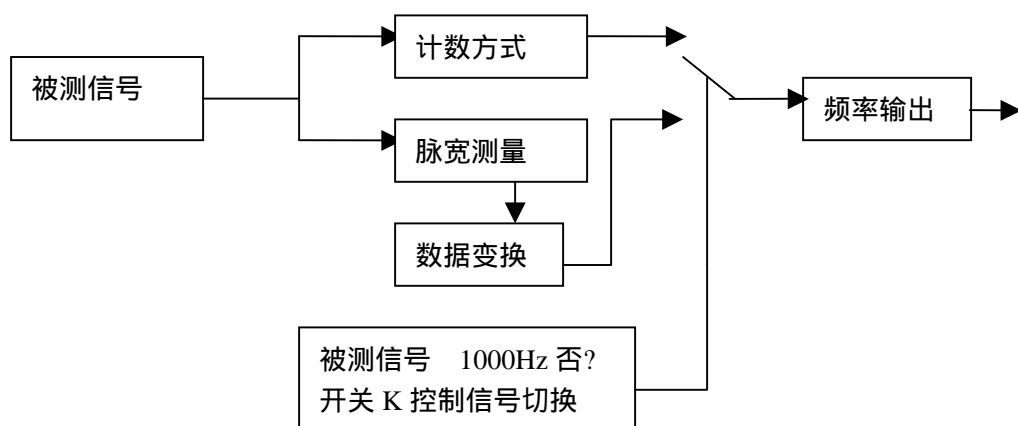


图 8-1 频率测量示意图

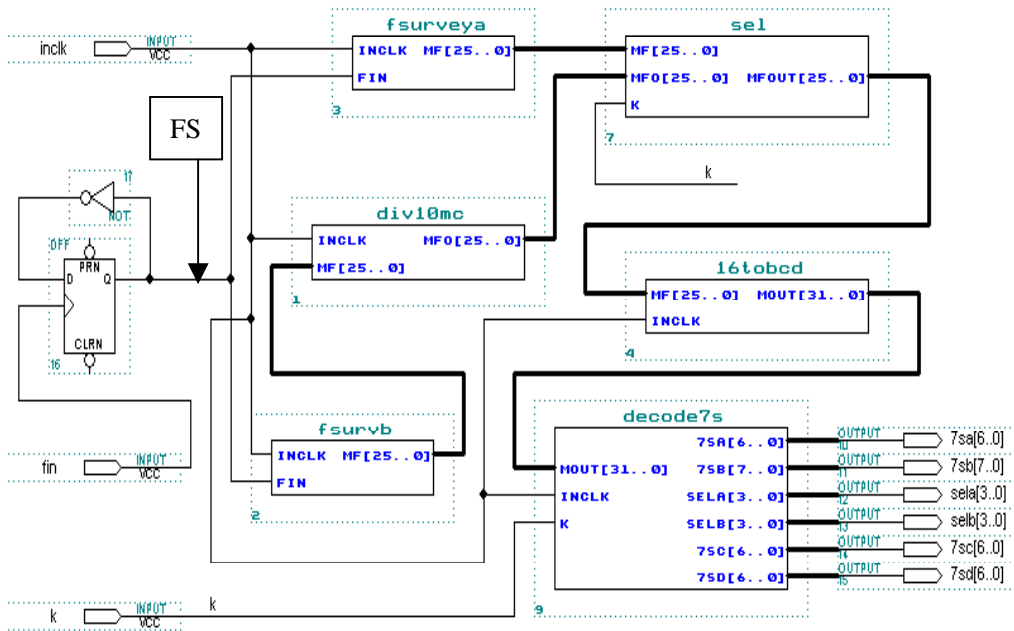


图 8-2 频率计模块示意图

图 8-2 中详细描述了各个模块的功能及连接关系。其中信号 *inclk* 为系统时钟，这里采用 40MHz，*fin* 为被测信号，其范围为 0.1Hz 至 50MHz，二分频后的 FS 作为测量的输入时钟，开关 *k* 实现计数方式与脉宽测量方式的切换。模块 *fsurveya* 实现计数方式测量，原理是将系统时钟 *inclk* 分频，得到高、低电平分别 2 秒的方波，在高电平时对被测信号计数，在低电平时将计数值输出。其 HDL 描述如下：

```

subdesign fsurveya
(inclk,fin: input;
 mf[25..0] : output;
 )
variable
ma[26..0]:dff;
fd:dff;
st[1..0]:dff;
mc[25..0],mf[25..0]:dff;
begin
ma[].clk=inclk;fd.clk=inclk;
st[].clk=inclk;mc[].clk=fin;mf[].clk=inclk;
if ma[]==79999999 then
ma[]=0;fd=!fd;
else
ma[]=ma[]+1; fd=fd;

```

```

        end if;
case st[] is
  when 0=>
    mc[]=0;mf[]=mf[];
if fd then
  st[]=1;
else
  st[]=0;
end if;
when 1=>
  MF[]=MF[];
  if fd then
    st[]=1;
    mc[]=mc[]+1;
  else
    st[]=2;mc[]=mc[];
  end if;
when 2=>
  mf[]=mc[];mc[]=mc[];
  st[]=0;
end case;
end;

```

模块 fsurvb 完成脉宽测量方式，为了提高测量精度采用 1MHz 作为计数时钟，因此首先将系统时钟 inclk 分频得到 1MHz 的时钟，在 FS 为高电平时计数，低电平时将测量数据输出，其 HDL 描述如下：

```

subdesign fsurvb
(inclk,fin: input;
 mf[25..0]: output;
)
variable
 ma[5..0],mf[25..0],mc[25..0]:dff;
 mfd:      dff;
 st[1..0]:dff;
begin
  ma[].clk=inclk;mfd.clk=inclk;st[].clk=inclk;
 mf[].clk=inclk;mc[].clk=mfd;
  if ma[]==19 then
    ma[]=0;mfd=!mfd;

```

```

else
    ma[]=ma[]+1;mfd=mfd;
end if;
case st[] is
when 0=>
    mf[]=mf[];mc[]=0;
    if fin then
        st[]=1;
    else
        st[]=0;
    end if;
when 1=>
    mf[]=mf[];
    if fin then
        mc[]=mc[]+1;st[]=1;
    else
        st[]=2;mc[]=mc[];
    end if;
when 2=>
    mf[]=mc[];mc[]=mc[];
    st[]=0;
end case;
end ;

```

模块 div10mc 完成脉宽测量数据到频率的转换，公式为频率值 $f=100\text{MHz}/\text{采样值}$ ，HDL 描述如下：

```

include "divide32.inc";
subdesign div10mc
(inclk,mf[25..0]):input;
    mfo[25..0]:    output;
)
variable
mdiv32:divide32;
begin
mdiv32.inclk=inclk;mdiv32.a[]=100000000;mdiv32.b[]=(0,mf[]);
mfo[]=mdiv32.c[25..0];
end;

```

在模块 div10mc 中为了完成除法运算，调用了标准的 32 位除法运算器模块 divide32,divide32 模块的 HDL 描述如下：

```

    subdesign divide32
(a[31..0],b[31..0],inclk:input;
  c[31..0]          : output;
)
variable
ma[32..0],mb[32..0],dc[32..0]:dff;
mc[2..0],md[4..0],c[31..0]:dff;
begin
(dc[],ma[],mb[],mc[],md[]).clk=inclk;c[].clk=inclk;
case mc[] is
  when 0=>
mb[]=(0,b[]);ma[]=(0,a[]);
mc[]=1;md[]=md[];c[]=c[];
  when 1=>
ma[]=ma[];c[]=c[];
if mb[]==0 then
mc[]=0;md[]=md[];
else
md[]=md[]+1;
if mb[31]==gnd then
  for i in 31 to 1 generate
    mb[i]=mb[i-1];
  end generate;
mc[]=1;
else
mb[]=mb[];mc[]=2;
end if;
end if;
  when 2=>
mb[]=mb[];md[]=md[];c[]=c[];
if ma[]>=mb[] then
ma[]=ma[]-mb[];
mc[]=4;dc[]=dc[]+1;
else
mc[]=3;dc[]=dc[];ma[]=ma[];
end if;
  when 3=>
mc[]=4;c[]=c[];

```

```

for n in 32 to 1 generate
ma[n]=ma[n-1];dc[n]=dc[n-1];
end generate;
mb[]=mb[];md[]=md[]-1;
when 4 =>
ma[]=ma[];dc[]=dc[];mb[]=mb[];c[]=c[];
if md[]==1 then
mc[]=5;md[]=md[];
else
mc[]=2;md[]=md[];
end if;
when 5=>
ma[]=ma[];c[]=c[];
if ma[31]==vcc then
dc[]=dc[]+1;
else
dc[]=dc[];
end if;
mc[]=6;
when 6=>
dc[]=dc[];mc[]=0;c[]=dc[31..0];
end case;
end;

```

模块 sel 完成计数方式测量与脉宽测量方式得到的数据之间的切换,其中 K 为高电平时选择计数方式数据, K 为低电平时选择脉宽测量方式数据, HDL 描述如下:

```

subdesign sel
(mf[25..0],mfo[25..0],k:input;
mfout[25..0]:output;
)
begin
if k then
mfout[]=mf[];
else
mfout[]=mfo[];
end if;
end ;

```

模块 16tobcd 完成十六进制到 BCD 码的转换, HDL 描述如下:


```

        else
            moutx[15..12]=moutx[15..12]+1;moutx[31..16]=moutx[31..16];
        end if;
    else
        moutx[11..8]=moutx[11..8]+1;moutx[31..12]=moutx[31..12];
    end if;
else
    moutx[31..8]=moutx[31..8];
    if ma[]>9 then
        st[]=1;ma[]=ma[]-10; moutx[7..4]=moutx[7..4]+1;
    else
        moutx[7..4]=moutx[7..4];moutx[3..0]=ma[3..0];
        st[]=2;
    end if;
end if;
when 2=>
    mout[]=moutx[];moutx[]=moutx[];st[]=0;
end case;
end ;

```

模块 decode7s 主要任务完成 7 段共阴极数码管的译码及驱动，其中 8 位 BCD 码由 mout[31..0]输入，7SA[6..0]、7SB[7..0]分别为两个扫描显示的段输出，sela[3..0]、selb[3..0]为八个扫描显示的位选择。7sc[6..0]、7sd[6..0]为两个静态显示数码管的段输出，显示“HZ”字样。整个设计采用高位灭零的显示方式。HDL 描述如下：

```

subdesign decode7s
(mout[31..0],inclk,k:input;
7sa[6..0],7sb[7..0],sela[3..0],selb[3..0]:output;
7sc[6..0],7sd[6..0]:output;
)
variable
ma[9..0],f,sta[1..0],stb[1..0],mda[3..0],mdb[3..0]:dff;
begin
7sc[]=h"37";7sd[]=h"6d";mda[].clk=inclk;mdb[].clk=inclk;
ma[].clk=inclk;sta[].clk=f;stb[].clk=f;f.clk=inclk;
    if ma[]==1000 then
        ma[]=0;f=!f;
    else
        ma[]=ma[]+1; f=f;
    end if;
end if;

```

```

sta[]=sta[]+1; stb[]=stb[]+1;
case sta[] is
when 0 =>
    mda[]=mout[31..28];
    if mout[31..28]==0 then
        sela[]=0;
    else
        sela[]=8;
    end if;
when 1=>
    mda[]=mout[27..24];
    if mout[31..24]==0 then
        sela[]=0;
    else
        sela[]=4;
    end if;
when 2=>
    mda[]=mout[23..20];
    if mout[31..20]==0 then
        sela[]=0;
    else
        sela[]=2;
    end if;
when 3=>
    mda[]=mout[19..16];
    if mout[31..16]==0 then
        sela[]=0;
    else
        sela[]=1;
    end if;
end case;
case stb[] is
when 0 =>
    mdb[]=mout[15..12];
    if mout[31..12]==0 then
        selb[]=0;
    else
        selb[]=8;

```

```

end if;
when 1=>
    mdb[]=mout[11..8];
    if k then
        7sb7=gnd;
    else
        7sb7=vcc;
    end if;
if mout[31..8]==0 then
    selb[]=0;
else
    selb[]=4;
end if;
when 2=>
    mdb[]=mout[7..4]; selb[]=2;
    when 3=>
        mdb[]=mout[3..0]; selb[]=1;
end case;
table
    mda[]=> 7sa[];
    0=>h"3f";
    1=>h"06";
    2=>h"5b";
    3=>h"4f";
    4=>h"66";
    5=>h"6d";
    6=>h"7d";
    7=>h"07";
    8=>h"7f";
    9=>h"6f";
end table;
table
    mdb[]=> 7sb[];
    0=>h"3f";
    1=>h"06";
    2=>h"5b";
    3=>h"4f";
    4=>h"66";

```

```

5=>h"6d";
6=>h"7d";
7=>h"07";
8=>h"7f";
9=>h"6f";
end table;
end ;

```

由于分频系数太大，软件仿真很困难，因此采用 CPLDEE-4 实验开发系统进行硬件仿真，仿真结果满足设计要求。

8.2 CPLD 器件在人机接口中的应用

人机接口电路在微机系统中应用非常广泛，其主要接口是键盘和显示器，在 CPLD 系统的开发中人机接口电路也是一个重要的环节。数码管的显示（包括静态和串行扫描显示）前面已经讨论的比较详细，本节内容主要讨论矩阵排列方式的键盘的扫描、按键去抖动、键码识别等内容。下面介绍 4X4 矩阵扫描、去抖、键码识别及显示。主要原理是先将系统时钟 $inclock$ (22MHz) 分频至 5ms 作为键查询时钟 $keyclkout$ ， $keyclkout$ 十分频后得到 50ms 时钟 $chuckout$ 作为触发时钟，在该时钟高电平的期间送出列扫描数据，然后将行数据送到去抖电路进行去抖后读入，根据行、列数据之间的关系确定键值。键盘布局如图 8-3 所示，去抖电路如图 8-4 所示：在该电路中采用 5ms 的时钟接收输入数据，如果连续三次数据为零，则可以确认数据是稳定的并可以接收。

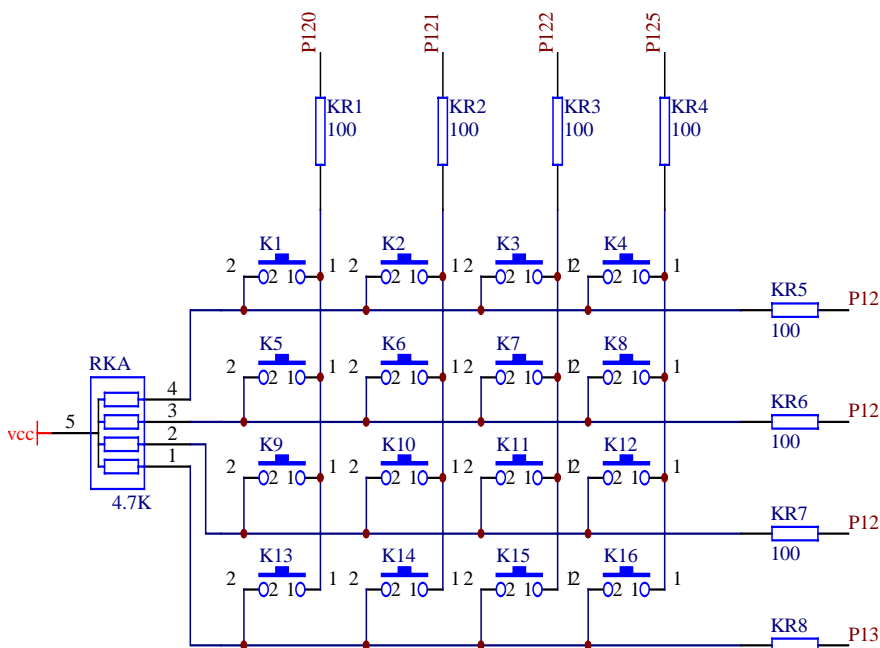


图 8-3 键盘电路示意图

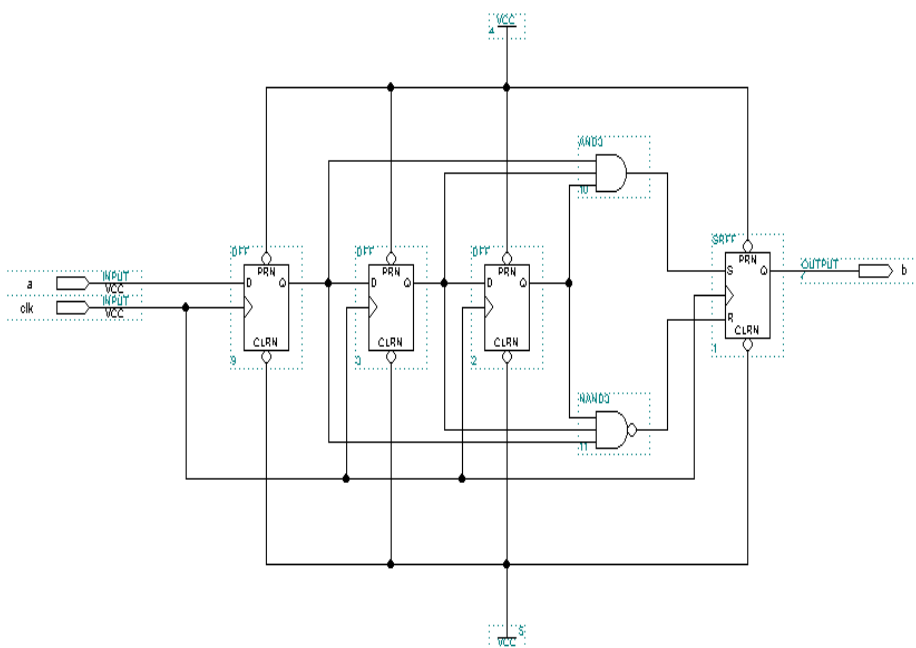


图 8-4 键盘去抖电路

以上是一个比较通用的抖动消除电路，它由 D 触发器与 RS 触发器构成。要求信号稳定的时间由 D 触发器的个数来决定。这个单元电路被命名为 tinglmove，并被作为元件在下面的 VHDL 程序中引用。下面是完整的键盘串行扫描显示电路的 VHDL 语言描述：

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
ENTITY key2 IS
    PORT (inclk      :IN std_logic;           --输入时钟信号
          inkey     :IN  std_logic_vector(0 to 3); --输入按键信号
          outkey    :OUT std_logic_vector( 0 to 3); --输出键盘扫描信号
          outled    :OUT std_logic_vector(7 downto 0) --LED 显示
    );
END key2;
ARCHITECTURE art OF key2 IS
    COMPONENT tinglmove --去抖电路引入
        PORT (a,clk: IN std_logic      ;
              b  : OUT std_logic) ;
    END COMPONENT;
    SIGNAL keyclk      :std_logic_vector(16 downto 0) ;

```

```

SIGNAL chuclk          :std_logic_vector (2 downto 0) ;
SIGNAL keyclkout,chuclkout :std_logic  ; --键盘去抖脉冲，串行扫描产生
                                          脉冲
SIGNAL chuout :std_logic_vector(0 to 3)  ;      --扫描信号
SIGNAL inkeymap :std_logic_vector(0 to 3)  ;--按键去抖后的信号
SIGNAL keyout      :std_logic_vector(0 to 7)  ;--
BEGIN
    roll: FOR i IN 0 TO 3 GENERATE      --生成四个去抖单元电路
        movskipX: tinglmove PORT MAP (inkey(i),keyclkout, inkeymap(i));
        END GENERATE;
clk_key:PROCESS(inclk)
    BEGIN
        if(inclk'event and inclk='1') then
            if keyclk=54999 then
                keyclk<="0000000000000000";
                keyclkout<=not keyclkout;
            else
                keyclk<=keyclk+1;
            end if;
        end if;
    END PROCESS clk_key;
clk_chu:PROCESS(keyclkout)
    BEGIN
        IF (keyclkout'EVENT AND keyclkout = '1') THEN
            IF chuclk=4 THEN
                chuclk<= "000";
                chuclkout<=not chuclkout;
            ELSE
                chuclk<=chuclk+1;
            END IF;
        END IF;
    END PROCESS clk_chu;
clk_chu_out:PROCESS(chuclkout)
    BEGIN
        IF (chuclkout'EVENT AND chuclkout='1') THEN
            IF chuout="1110" THEN
                IF inkeymap/="1111" THEN
                    keyout<=chuout&inkeymap ;
                
```

```

        END IF;
        chuout<="1101";
    ELSIF chuout="1101" THEN
        IF inkeymap/="1111" THEN
            keyout<=chuout&inkeymap ;
        END IF;
        chuout<="1011";
    ELSIF chuout="1011" THEN
        IF inkeymap/="1111" THEN
            keyout<=chuout&inkeymap ;
        END IF;
        chuout<="0111" ;
    ELSIF chuout="0111" THEN
        IF inkeymap/="1111" THEN
            keyout<=chuout&inkeymap ;
        END IF;
        chuout<="1110";
    ELSE
        chuout<="1110";
    END IF;
END IF;
END PROCESS clk_chu_out;
outkey<=chuout;
out_led:PROCESS(keyout)
BEGIN
    case keyout(0 to 3) is
        when "0111" => case keyout(4 to 7) is
            when "0111"=> outled<=x"7e";
            when "1011"=> outled<=x"33";
            when "1101"=> outled<=x"7f";
            when "1110"=> outled<=x"4e";
            when others=> outled<=x"00";
        end case;
    when "1011" => case keyout(4 to 7) is
        when "0111"=> outled<=x"30";
        when "1011"=> outled<=x"5b";
        when "1101"=> outled<=x"7b";
        when "1110"=> outled<=x"3d";
    end case;
END PROCESS;

```

```

        when others=> outled<=x"00";
    end case;
when "1101" => case    keyout(4 to 7) is
    when "0111"=> outled<=x"6d";
    when "1011"=> outled<=x"5f";
    when "1101"=> outled<=x"77";
    when "1110"=> outled<=x"4f";
    when others=> outled<=x"00";
end case;
when "1110" => case    keyout(4 to 7) is
    when "0111"=> outled<=x"79";
    when "1011"=> outled<=x"70";
    when "1101"=> outled<=x"1f";
    when "1110"=> outled<=x"47";
    when others=> outled<=x"00";
end case;
when others => outled<=x"00";
end case;
END PROCESS out_led;
end art;

```

程序说明：这个系统中引入时钟频率为 10M---40M 之间或更高时都能很好的工作。如果系统时钟不在此范围，只要改变分频系数使 keyclkout 的周期约为 5ms 即可。

8.3 CPLD 在微机系统中的应用

由于目前 CPLD 器件使用灵活，可编程管脚丰富，而且与微机相关 IPCORE 也越来越多，用 CPLD 与微机配合可以极大地缩短开发周期和上市时间，单片机与 CPLD 配合可以相互弥补对方的不足。

- (1) 单片机可以作为 FPGA 的配置系统，由于 FPGA 的容量大，下载次数不受限制，价格便宜因此获得广泛应用。但 FPGA 的数据掉电丢失，需用存储器解决上电重新配置问题，而专用配置存储器的价格昂贵，普通的存储器又无法解决读写时序，因此用单片机加存储器解决上电重新配置是一种比较理想的方案。
- (2) 单片机系统一般是由一个单片机加外围电路组成，系统复杂，外围电路同样比较复杂，而且需有 PCB 板系统。如果用 CPLD/FPGA 构架单片机外围系统，配合 EDA 软件进行设计既经济、灵活又极大提高开发速度。
- (3) 单片机与 CPLD 可以相互弥补对方的不足。单片机的子程序可以循环调用，随着处理事物的增多仅仅增加程序存储器的空间即可，而 CPLD 不同，多一个事件处理即多增加内部资源，因此可以利用 CPLD 作为单片机的外围器件并处理一些需快速处理的数据，单片机处理繁多的慢速数据。

因此 CPLD 器件在微机系统获得广泛应用。本节以 CPLD 器件作为 MCS51 单片机的接口器件研究单片机、CPLD 配合在数据采集、数据传输、人机接口、波形产生等各方面的应用接口的模块如图 8-5、8-6 所示：

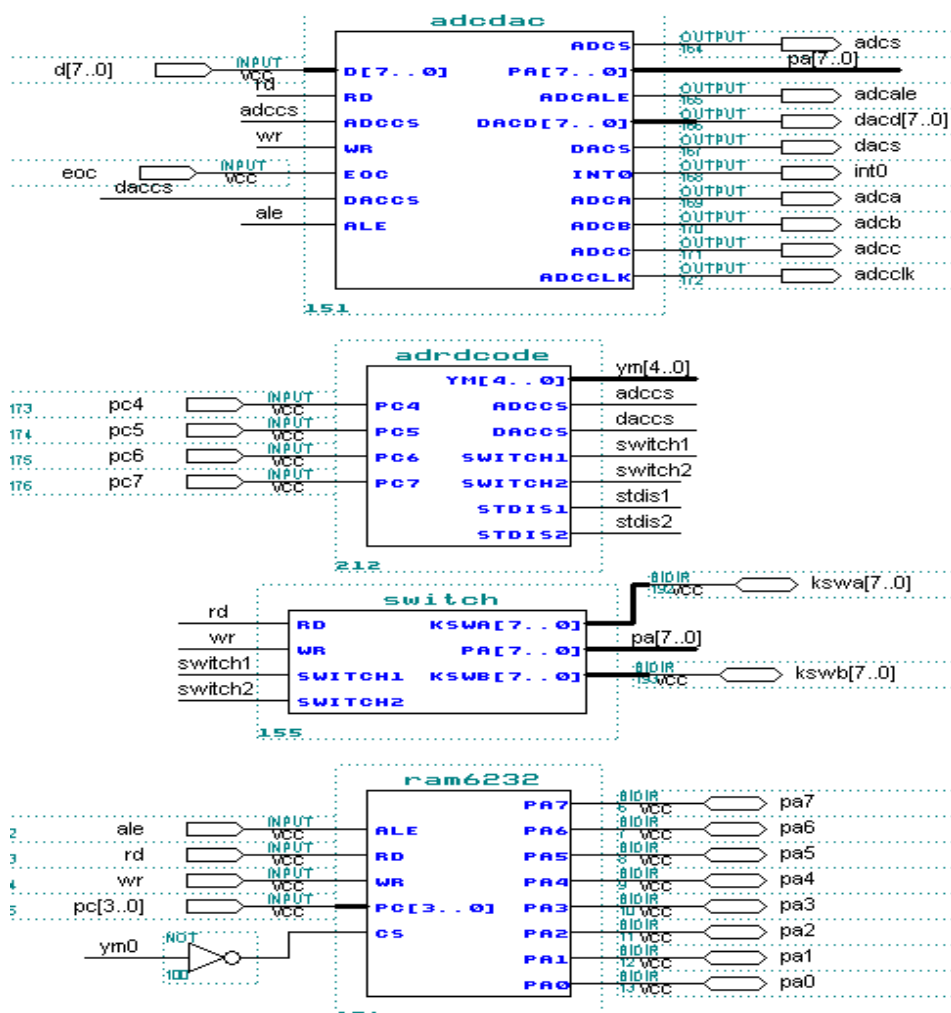


图 8-5 通过 CPLD 构建开关、RAM、A/D、D/A 与单片机的接口

在图 8-5 中采用层次设计方式，其中模块 adcdac 完成 ADC0809 数/模转换、DAC0832 模/数转换与单片机的接口；模块 key4X4 完成 4X4 键盘与单片机的接口，键盘电路如图 8-3 所示；模块 switch 完成 16 个数位开关与单片机的接口；模块 ram6232 是利用 CPLD 内部的 EAB 技术生成可以完全替代 4kram 的存储器；模块 adrdcode 是利用单片机 P2 口的 p2.4、p2.5、p2.6、p2.7 进行译码将 64k 的读写空间进行划分，其中地址空间 0-0FFFH 译码产生 ym0 控制 RAM 的读写；1000H 译码产生 ym1；2000H 译码产生 ym2 分别控制键盘的扫描和回读；3000H，4000H 译码产生 ym3,ym4 控制八个串行扫描数码管的显示，其连接方式见图 8.6；5000H，6000H 译码产生 adccs、daccs 控制 ADC0809 和 DAC0832 数据接口；7000H、8000H 译码产生 switch1、switch2

信号控制 16 个开关的读写(这里采用的实验系统中 16 个开关和 16 个状态指示 LED 是复用的方式,因此既可以输入,也可以输出);9000H、0A000H 译码产生 stdis1、stdis2 分别控制两个静态数码管的显示。

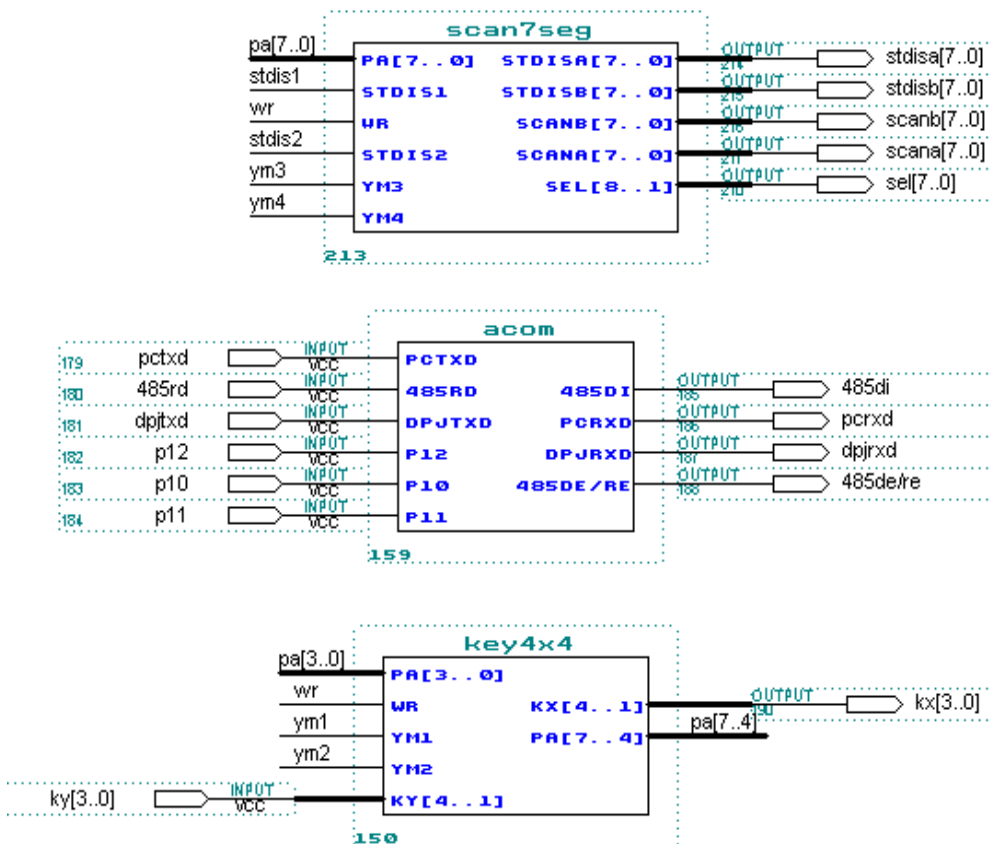


图 8-6 通过 CPLD 构建显示、通信、键盘与单片机的接口

下面分别介绍各个模块的原理：

- 1、 adcdac 模块 :由于 ADC0809 采样与 DCA0832 的数据输入端都使用单片机的 P0 口,因此采用 74244 三态总线进行隔离,读写选通由 rd、wr、adccs、daccs、eoc、ale 信号以及它们之间组合逻辑关系构成,该部分内容是常规的单片机接口信号,这里不在累述。pa0、pa1、pa2 分别为单片机的 p0.0、p0.1、p0.2 与 ale、74373 配合产生 ADC0809 的八路模拟通道的选择信号。ADC0809 的采样时钟 adclk 由单片机的 ale 经 74163 计数器 5 分频后产生。adcs、dacs 为 ADC0809 与 DAC0832 的片选端,这里采用直接接地的方式。具体的电路连接方式如图 8-7 所示：

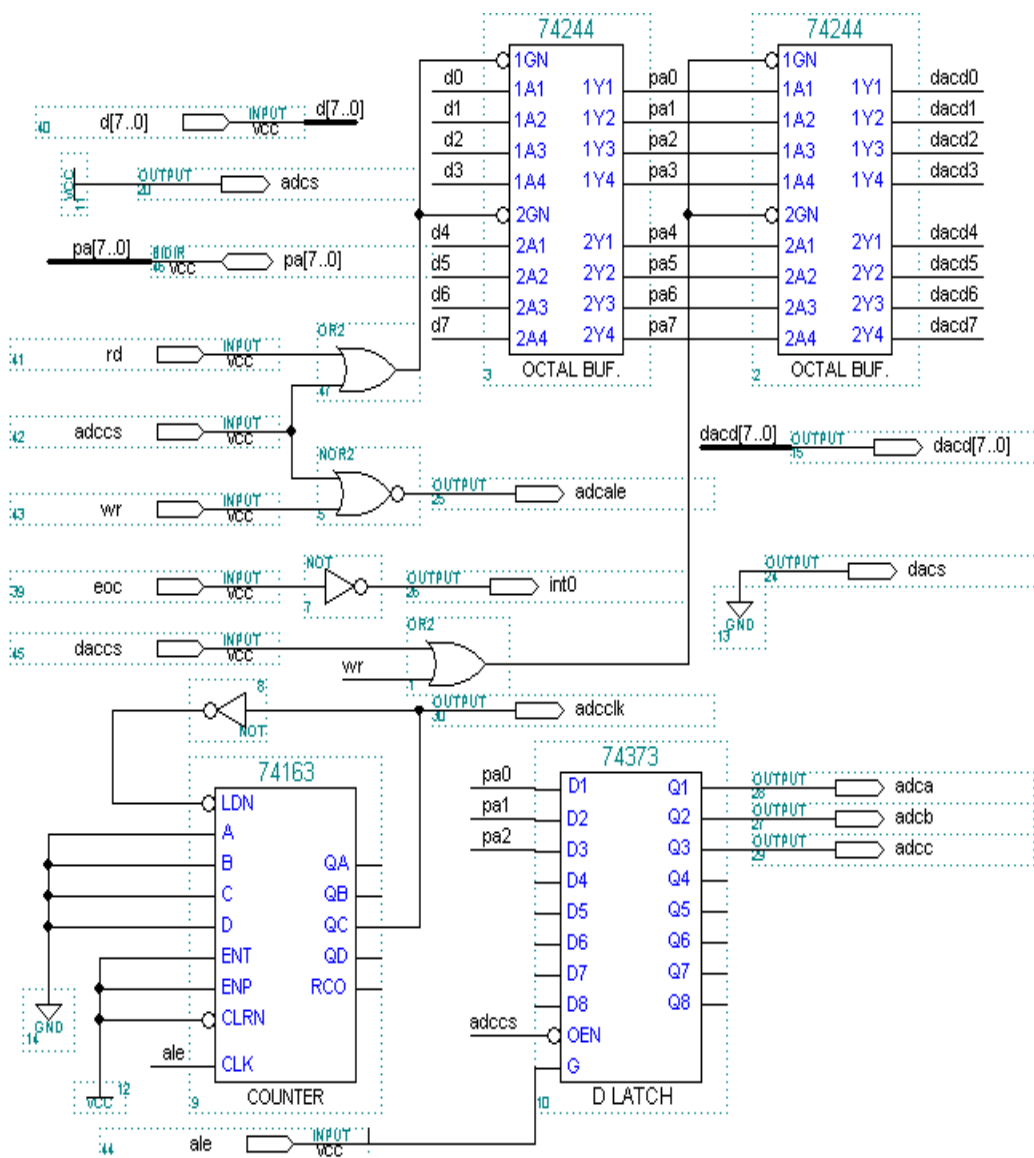


图 8-7 ADC0809 与 DAC0832 的接口模块

- 2、`adrdcode` 模块 `adrdcode` 模块的原理比较简单，利用单片机的 P2 口的 $p2.4, p2.5, p2.6, p2.7$ 四根地址线与两片 74138 级连进行译码将 I/O 空间划分为 16 个部分，这里只使用了其中的 11 个，还有 5 个地址空间没有使用，可以留着作为扩展空间，如液晶显示器、PC 机的并行接口等，划分后的地址空间上面已经做了详细介绍。电路如图 8-8 所示：

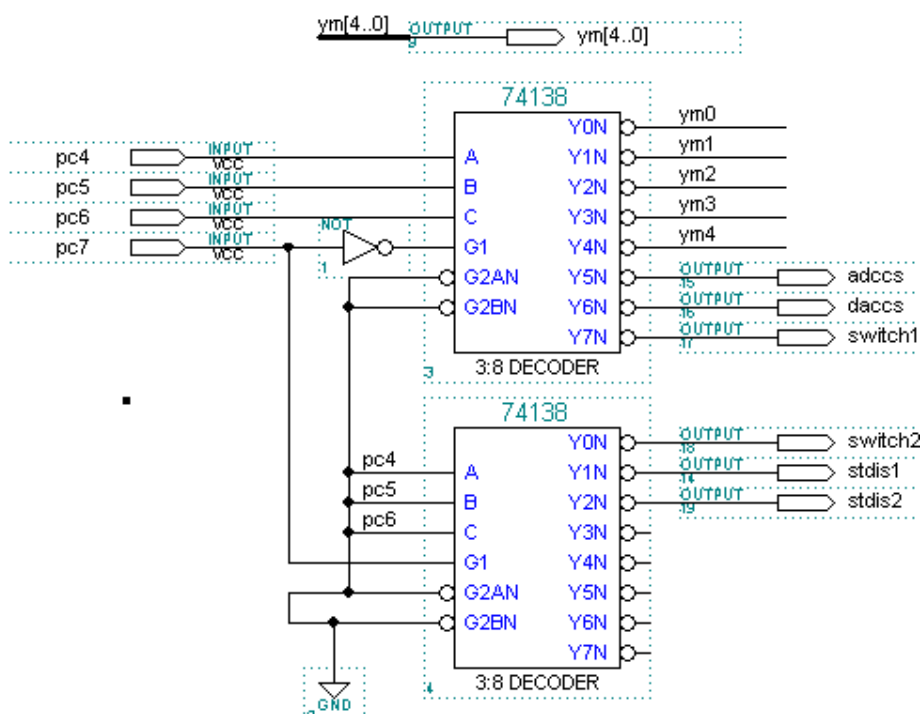


图 8-8 地址译码器

- 3、 switch 模块 :在 switch 模块中主要利用单片机实现 16 个开关和 16 个 LED 指示灯的读写，开关、指示灯与 CPLD 的连接方式如图 8.9 所示：

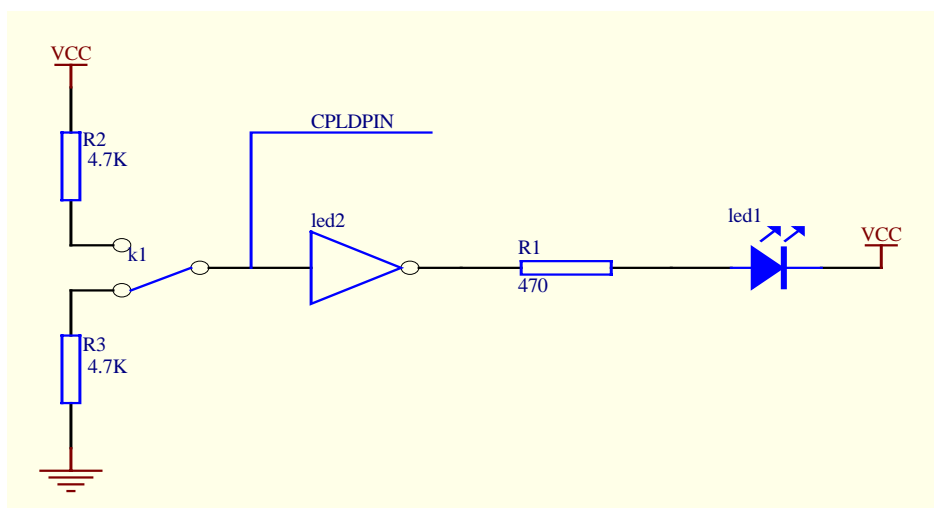


图 8-9 开关与 LED 指示灯与 CPLD 的物理连接

图 8-10 是 switch 模块的具体内部电路，在该电路中由于输入、输出都是由单片机的 P0 口操作，因此输入、输出需要 74244 三态总线隔离，控制逻辑由 rd、wr、

switch1、switch2 配合产生。

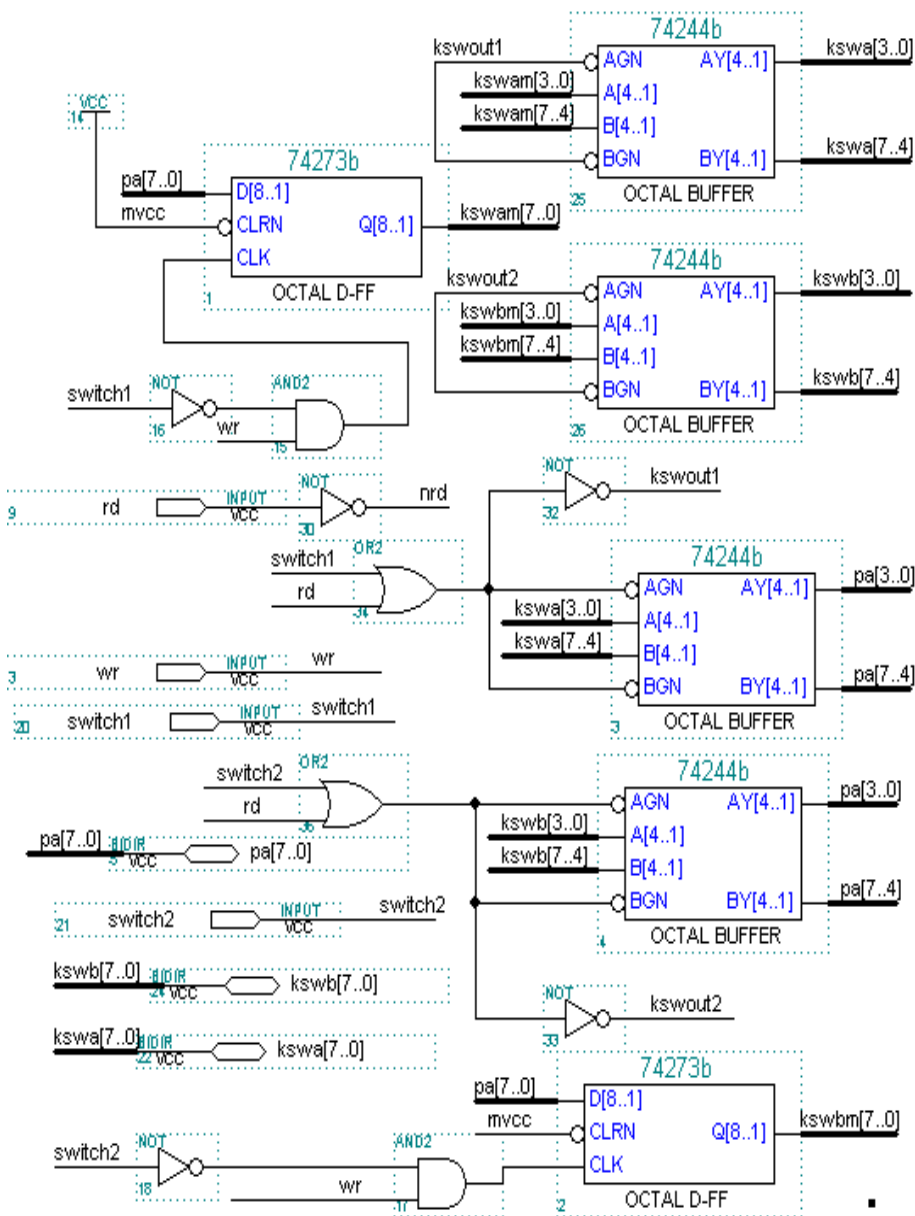


图 8-10 switch 模块的内部电路

4、ram6232 模块 :该模块是利用 CPLD 器件内部的 EAB 技术构造的 4K 的 RAM , 其功能可以完全替代 ram6232 , 其 HDL 描述如下 :

```

INCLUDE "YY.INC";
SUBDESIGN RAM6232
(ale,rd,wr,pc[3..0],CS:input;

```

```

pa7,Pa6,pa5,pa4,pa3,pa2,pa1,pa0 :bidir;
)
variable
myy: yy;
md[7..0]:LATCH;
begin
md[0].ENA=ale;
md[0]=pa[7..0];
myy.ad[7..0]=md[0];
myy.ad[11..8]=pc[3..0];
myy.rd=rd;
myy.we=wr;
myy.cs=cs;
pa[7..0]=myy.dio[0];
end;

```

在该模块的描述中，将一个标准的 4K RAM 模块 YY 与 ALE 以及锁存器 md[7..0]配合实现数据总线、地址总线的分离。4KRAM 的电路如图 8-11 所示：

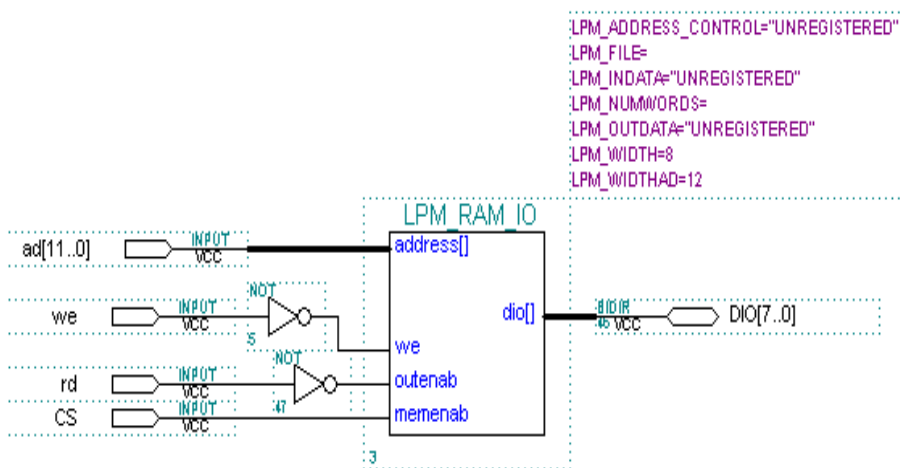


图 8-11 利用 CPLD 器件的 EAB 技术实现 4K RAM

5、scan7seg 模块的原理：在该模块中可以控制 10 个数码管的显示，整个图形分为上、下两个块，上半部分由单片机的 P0 口、stdis1、stdis2、wr 以及两个 74273 配合驱动两个静态数码管的显示。下半部分主要用来驱动 8 个串行扫描数码管。由 ym3 和 ym4 决定段驱动和位扫描的地址空间，scana[7..0],scanb[7..0]各驱动四个数码管的八段，sel[8..1]驱动八个数码管的位。具体电路如图 8-12 所示：

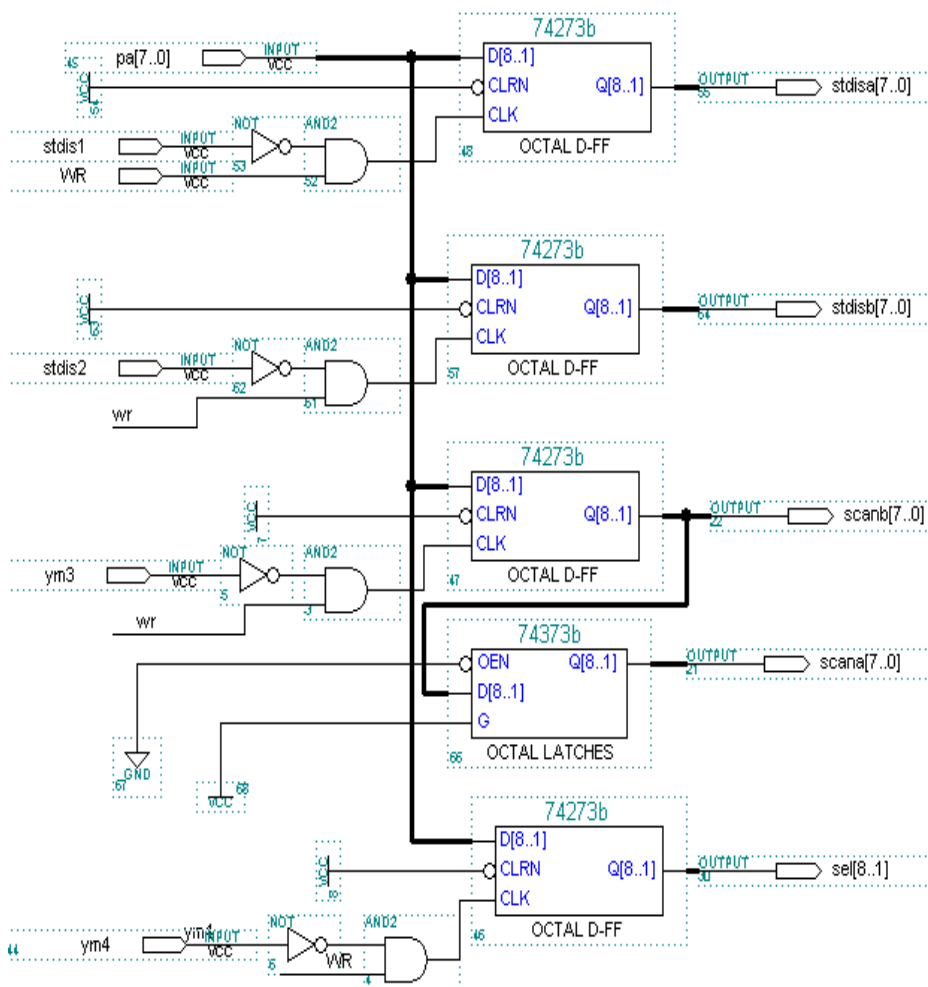


图 8-12 数码管的选择与驱动

6、key4x4 模块的原理：该模块的功能是利用单片机的 P0 口，进行 16 个键盘的扫描和查询并形成编码。键盘的排列方式见图 8.3，将 P0 口分成高、低四位，其中低四位值经 74273 锁存后输出列扫描线，扫描码为“0111”、“1011”、“1101”、“1110”，并依次循环。键盘的行线经 74244 隔离和缓冲后送 P0 口的高四位，单片机依据扫描码和回读行线的值就可以确定被按下的键的位置，并进行编码。键扫描的时序由 ym1 和 wr 产生的逻辑关系确定，键值的回读时序由 ym2 和 rd 确定，具体电路如图 8-13 所示。这里仅仅介绍键盘与 CPLD 单片机接口的例子，实际利用 CPLD 进行扫描、去抖、编码也是非常方便的，在本章的前面内容已经介绍了键盘的 VHDL 处理，在第四章 AHDL 语言部分也介绍了利用 AHDL 进行键盘处理的程序。

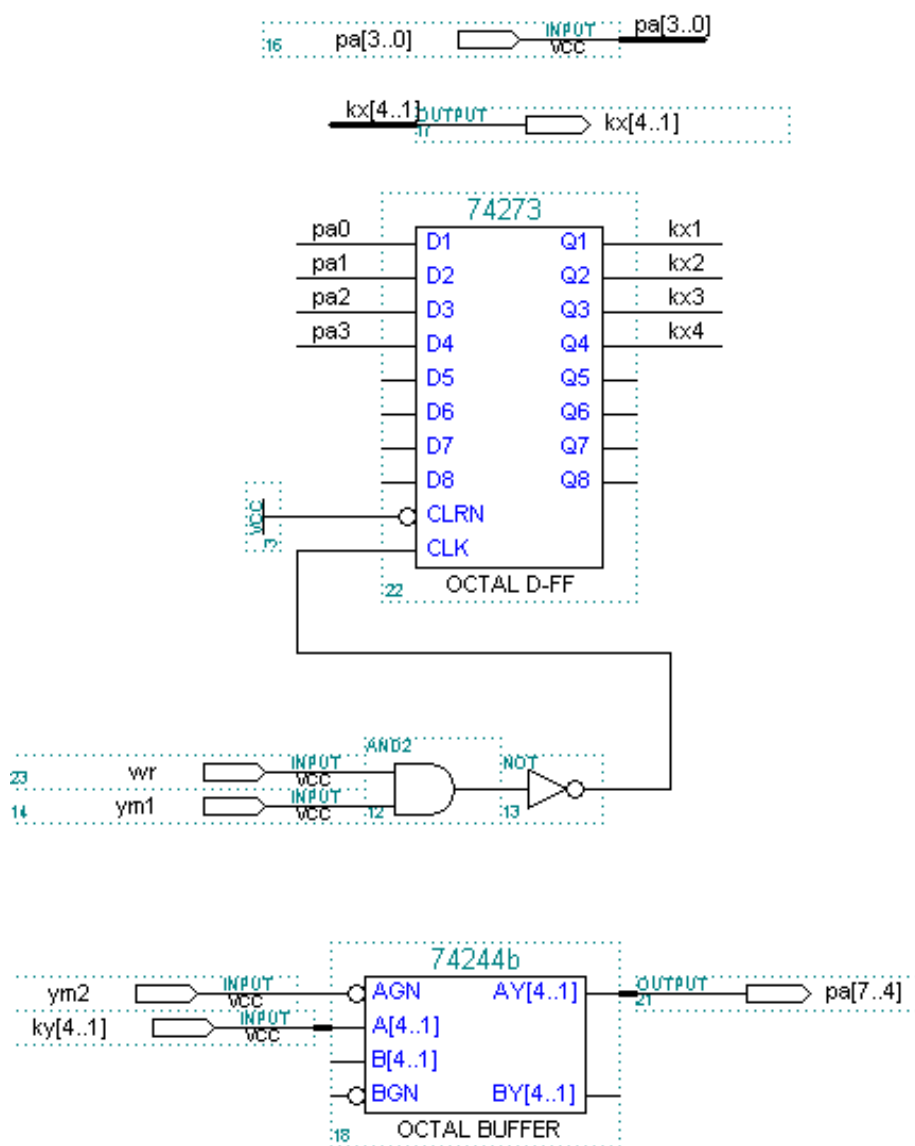


图 8-13 键盘的扫描与回读

7、acom 模块的原理：该模块主要研究异步串行通信的内容，主要由三个部分：(1) PC 机的异步串行口经 RS232 与 CPLD 相连 (2) 单片机的异步串行口与 CPLD 相连 (3) CPLD 与 MAX487 相连。acom 模块内容主要研究如何控制 PC 机到单片机，PC 机到 485，单片机到 485 的通信。其中单片机的 P1.0、P1.1 经 74138 译码后产生控制逻辑，当 P1.1、P1.0 组合的值为“00”时选通 PC 机到 485 的直通方式，“01”时选通 PC 机与单片机之间的相互通信，“10”时选通单片机与 485 的连接，实现单片机的远距离传输与多机通信。P1.2 控制 485 的收、发，P1.2 为“0”时接收，为“1”发送。具体电路见图 8-14 所示：

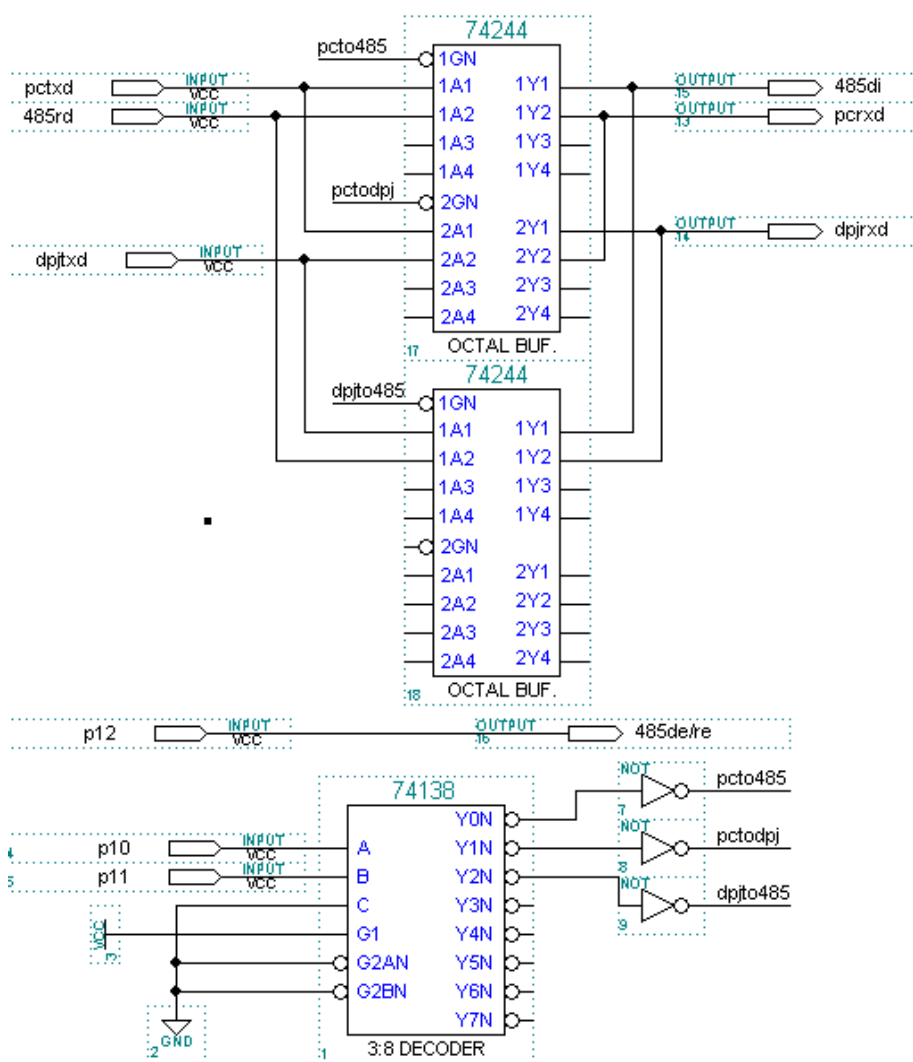


图 8-14 通信接口电路

8.4 CPLD 器件在通信中的应用

由于 HDL 语言的使用,使 CPLD 器件在数字设计中变的非常容易,因此 CPLD 器件在通信领域获得广泛应用。特别是在差错编码技术、信息加密技术、位同步信号提取技术、复接技术、串行异步通信等方面更是应用广泛。下面就以异步串行通信为例介绍 CPLD 在通信方面的应用。本节内容介绍如何将 CPLD 与 PC 机配合形成任意波形函数发生器,其主要模块介绍如下:模块 UNIVERSALRXD 是用 VHDL 语言描述的通用异步接收器,可以接受和 9 位数据,第九位数据由 CHEN 输出,为“0”表示接收的是数据码,为“1”表示接收的是命令码。FS[3..0]控制该模块的波特率,总共有 16 中选择(300—115200)。OUTEN 表示数据接收完成,OUTDATA[7..0]是输出的低八位数据,INCLK 是系统时钟(22.1184MHz),OUTCLK 是波特率的 96

倍，可以给其它模块使用。模块 WAVEVBIO 主要完成 ID 号的选择，这是在多机通信中必不可少的一个部件。其功能描述如下：ID[5..0]是本机的地址码，当 UNIVERSALRXD 模块第九位数据接收完毕，且第九位数据 CHEN 为“1”时，使能 WAVEVBIO 模块，将接收到的八位数据与本机的 ID 号比较，如果相等则表示本机选中，使 iden 为“1”并使能其它模块，以处理后续接收的数据，否则本机将接收的数据丢弃，其中命令码 INDATA[7..0]中高两位编码 INDATA[7..6] 是“01”时，低位数据为 ID 号；“10”时，低位数据为波特率调整系数；“11”时，低位数据为扫描频率系数。模块 WAVEVB-BPS 是波特率选择器，其主要原理是当本机的 ID 号，接收的第九位数据完成（INEN 有效）以及 CHEN 有效时，根据 INDATA[7..0]来调整接收器的波特率。波特率的调整码由 fs[3..0]输出。Csen 为 D/A 转换器的片选端，电路如图 8-15 所示：

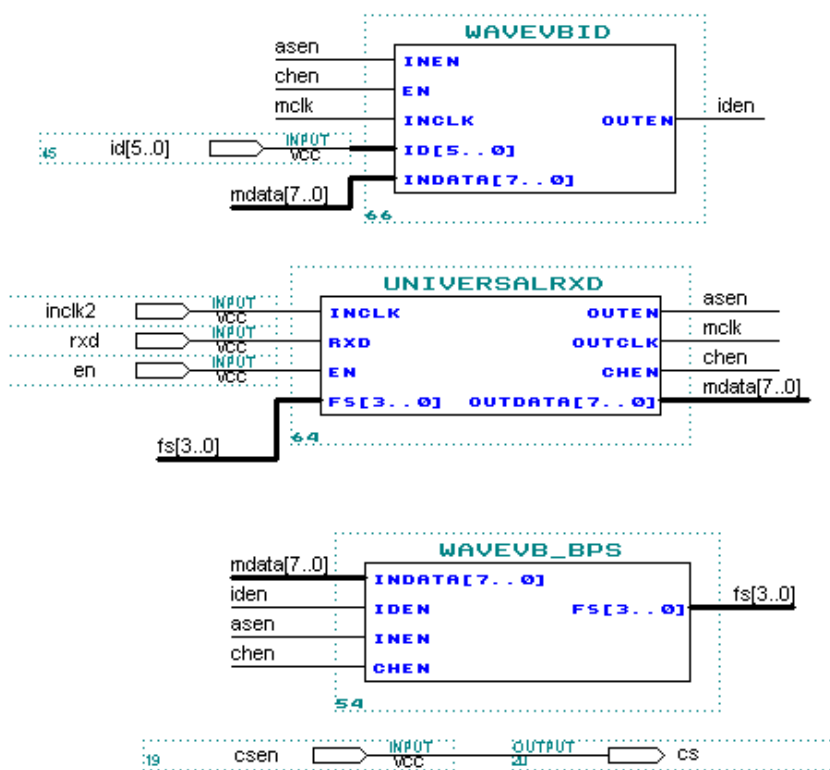


图 8-15 异步通信控制电路

模块 LPM-RAM-DP 是双口 RAM，其中 DATA[]，waddress[]，wren 控制数据的写入，是序逻辑由模块 WAVEVB-22 控制；LPM-RAM-DP 模块的 rdaddress[]、q[]控制数据读出，并送 D/A 转换器产生波形，波形的频率是由 rdaddress[]的变化快慢决定的；模块 WAVEVB-3 的功能是根据 infs[11..0]的数据调整 rdaddress[]的时钟；模块 WAVE-F 的功能是将接收的多字节数据拼接形成 insf[11..0]。具体电路见图 8-16 所示：

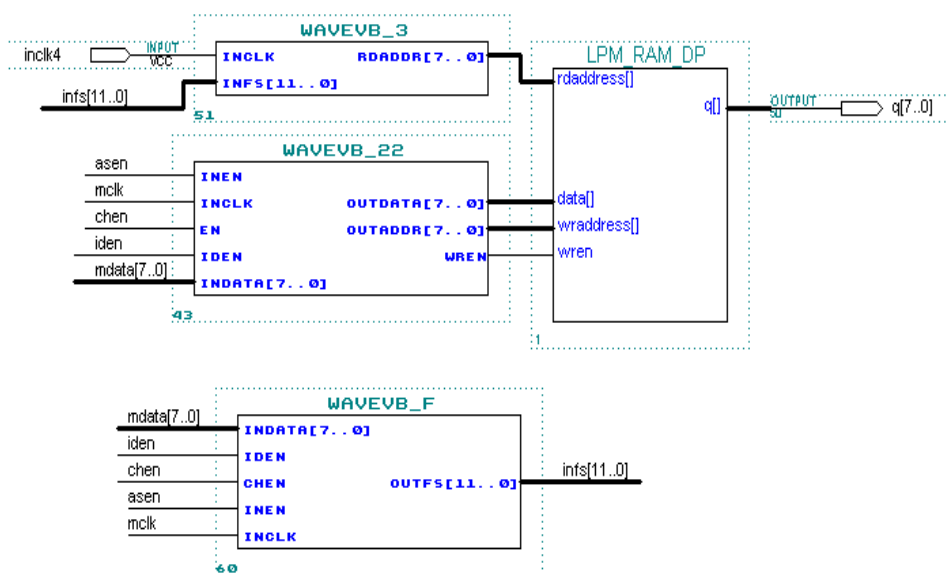


图 8-16 波形发生器电路模块图

模块 UNIVERSALRXD 的 VHDL 语言如下：

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY universalrxd IS
PORT (inclk,rxd,en      :IN STD_LOGIC;
      fs                :IN STD_LOGIC_VECTOR(3 DOWNT0 0);
      outen,outclk,chen :OUT STD_LOGIC;
      outdata          :OUT STD_LOGIC_VECTOR(7 DOWNT0 0)
      --outpe         :OUT STD_LOGIC
);
END universalrxd;
ARCHITECTURE art OF universalrxd IS
SIGNAL clk_1      :STD_LOGIC ;
SIGNAL b         :STD_LOGIC_VECTOR(7 DOWNT0 0) ;
SIGNAL d         :STD_LOGIC_VECTOR(3 DOWNT0 0) ;
SIGNAL K,ben,start :STD_LOGIC;
SIGNAL js0,js,js1 :STD_LOGIC_VECTOR(6 DOWNT0 0) ;
SIGNAL c         :STD_LOGIC_VECTOR(5 DOWNT0 0) ;
SIGNAL m         :STD_LOGIC_VECTOR(8 DOWNT0 0) ;
BEGIN

```

```

    outen<=k; outclk<=clk_1; --outdata<=outdata_1;
P0:PROCESS(inclk)
BEGIN
IF(inclk'EVENT AND inclk='1') THEN
    CASE fs IS
        WHEN "0000"=>                                --115200 BPS
            clk_1<=NOT clk_1;
            --m<="000000000";
        WHEN "0001"=>                                --300 BPS
            IF m>=383 THEN
                m<="000000000" ;clk_1<=NOT clk_1;
            ELSE
                m<=m+1;
            END IF ;
        WHEN "0010"=>                                --600 BPS
            IF m>=191 THEN
                m<="000000000" ;clk_1<=NOT clk_1;
            ELSE
                m<=m+1;
            END IF ;
        WHEN "0011"=>                                --1200 BPS
            IF m>=95 THEN
                m<="000000000" ;clk_1<=NOT clk_1;
            ELSE
                m<=m+1;
            END IF ;
        WHEN "0100"=>                                --2400 BPS
            IF m>=47 THEN
                m<="000000000" ;clk_1<=NOT clk_1;
            ELSE
                m<=m+1;
            END IF ;
        WHEN "0101"=>                                --9600 BPS
            IF m>=11 THEN
                m<="000000000" ;clk_1<=NOT clk_1;
            ELSE
                m<=m+1;
            END IF;
    
```

```

WHEN "0110"=>                                --14400 BPS
    IF m>=7 THEN
        m<="000000000" ;clk_1<=NOT clk_1;
    ELSE
        m<=m+1;
    END IF ;
WHEN "0111"=>                                --19200 BPS
    IF m>=5 THEN
        m<="000000000" ;clk_1<=NOT clk_1;
    ELSE
        m<=m+1;
    END IF ;
WHEN "1000"=>                                --28800 BPS
    IF m>=3 THEN
        m<="000000000" ;clk_1<=NOT clk_1;
    ELSE
        m<=m+1;
    END IF ;
WHEN "1001"=>                                --38400 BPS
    IF m>=2 THEN
        m<="000000000" ;clk_1<=NOT clk_1;
    ELSE
        m<=m+1;
    END IF ;
WHEN "1010"=>                                --57600 BPS
    IF m>=1 THEN
        m<="000000000" ;clk_1<=NOT clk_1;
    ELSE
        m<=m+1;
    END IF ;
WHEN OTHERS=>
    clk_1<=not clk_1;
    --m<="000000000";

END CASE;
END IF;
END PROCESS P0;
P1:PROCESS(clk_1,en)
BEGIN

```

```

IF (clk_1'EVENT AND clk_1='1') THEN
  IF en='1' THEN
    IF ben='0' THEN
      K<='0';
      IF rxd='0' THEN
        IF start='0' THEN
          IF c=25 THEN
            C<="000000";
            start<='1';
          ELSE
            c<=c+1;
          END IF;
        END IF;
      ELSE
        C<="000000";
      END IF;
      IF start='1' THEN
        IF js=69 THEN
          js<="0000000";
          IF js0>45 THEN
            ben<='1';js0<="0000000";
          ELSE
            js0<="0000000";
          END IF;
        ELSE
          js<=js+1;
          IF rxd='0' THEN
            js0<=js0+1;
          END IF;
        END IF;
      END IF;
    ELSE
      start<='0';C<="000000";
      IF js=95 THEN
        IF d=0 THEN
          IF js0>js1 THEN
            b(0)<='0'; d<=d+1; k<='1';
            js<="0000000";js0<="0000000";js1<="0000000";
          END IF;
        END IF;
      END IF;
    END IF;
  END IF;
END IF;

```

```

ELSE
    b(0)<='1'; d<=d+1; k<='1';
    js<="0000000";js0<="0000000";js1<="0000000";
END IF;
ELSIF d=1 THEN
    IF js0>js1 THEN
        b(1)<='0'; d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    ELSE
        b(1)<='1'; d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    END IF;
ELSIF d=2 THEN
    IF js0>js1 THEN
        b(2)<='0'; d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    ELSE
        b(2)<='1'; d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    END IF;
ELSIF d=3 THEN
    IF js0>js1 THEN
        b(3)<='0'; d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    ELSE
        b(3)<='1'; d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    END IF;
ELSIF d=4 THEN
    IF js0>js1 THEN
        b(4)<='0'; d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    ELSE
        b(4)<='1'; d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    END IF;
ELSIF d=5 THEN
    IF js0>js1 THEN

```

```

        b(5)<='0'; d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    ELSE
        b(5)<='1'; d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    END IF;
ELSIF d=6 THEN
    IF js0>js1 THEN
        b(6)<='0'; d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    ELSE
        b(6)<='1'; d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    END IF;
ELSIF d=7 THEN
    IF js0>js1 THEN
        b(7)<='0'; d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    ELSE
        b(7)<='1'; d<=d+1 ;
        js<="0000000";js0<="0000000";js1<="0000000";
    END IF;
ELSE
    IF js0>js1 THEN
        chen<='0';d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    ELSE
        chen<='1'; d<=d+1 ;
        js<="0000000";js0<="0000000";js1<="0000000";
    END IF;
    --outdata_1<=b;
END IF ;
ELSE
    IF d=9 THEN
        outdata<=b;
        d<=d+1;
        js<="0000000";js0<="0000000";js1<="0000000";
    ELSIF d=10 THEN

```

```
d<="0000";
ben<='0';
js<="0000000";js0<="0000000";js1<="0000000";
    ELSE
        js<=js+1;
    IF ( js=46 or js=47 or js=48 ) THEN
        IF rxd='0' THEN
            js0<=js0+1;
        ELSE
            js1<=js1+1;
        END IF;
    END IF;
END IF;
END IF ;
END IF;
END IF;
END IF;
END PROCESS P1;
END art;
```