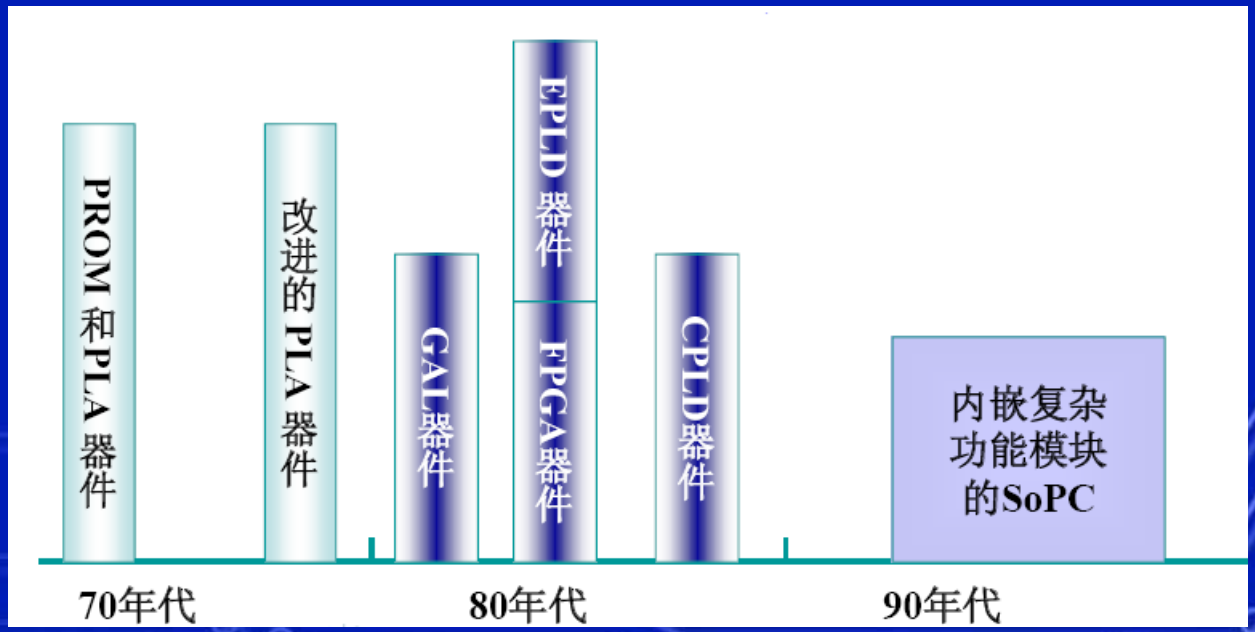
第2章　可编程逻辑器件

- 2.1可编程逻辑器件概述
- 2.2 通用阵列逻辑GAL
- 2.3 复杂可编程逻辑器件CPLD
- 2.4 现场可编程门阵列FPGA

2.1可编程逻辑器件概述

- PLD是可编程逻辑器件（Programmable Logic Devices）的英文缩写，是EDA得以实现的硬件基础，通过编程，可灵活方便地构建和修改数字系统。
- 可编程逻辑器件是集成电路技术发展的产物。自20世纪60年代以来，集成电路技术迅猛发展，数字集成电路已经历了从SSI、MSI、LSI到VLSI的发展过程，促进了可编程逻辑器件的飞速发展。



PLD的发展及现状

①ALTERA：主要产品：MAX3000/7000、FELX6K/10K、APEX20K、ACEX1K、Cyclone、Stratix等。开发工具MAX+PLUS II、Quartus II。仿真软件ModelSim。

②XILINX：FPGA的发明者。产品有XC9500/4000、Coolrunner(XPLA3)、Spartan、Vertex等系列，开发软件为Foundation和ISE。全球PLD/FPGA产品60%以上是由Altera和Xilinx提供的。可以讲Altera和Xilinx共同决定了PLD技术的发展方向。

③Lattice-Vantis：Lattice是ISP（In-System Programmability）技术的发明者。

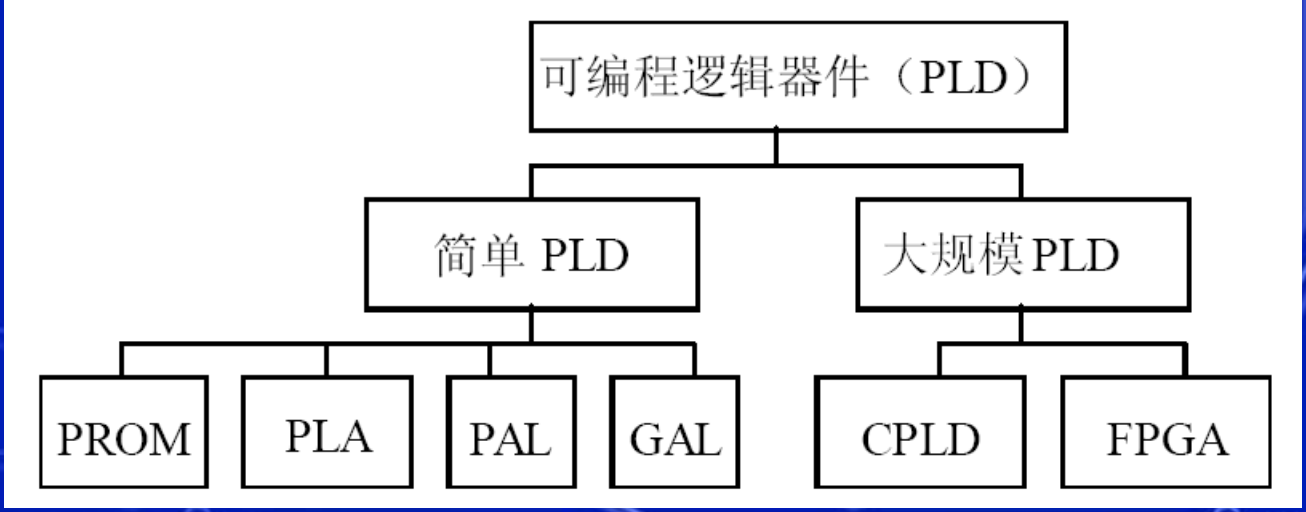
④ACTEL：反熔丝（一次性烧写）PLD的领导者。由于反熔丝PLD抗辐射、耐高低温、功耗低、速度快，所以在军品和宇航级上有较大优势。

⑤Quicklogic。⑥Lucent。

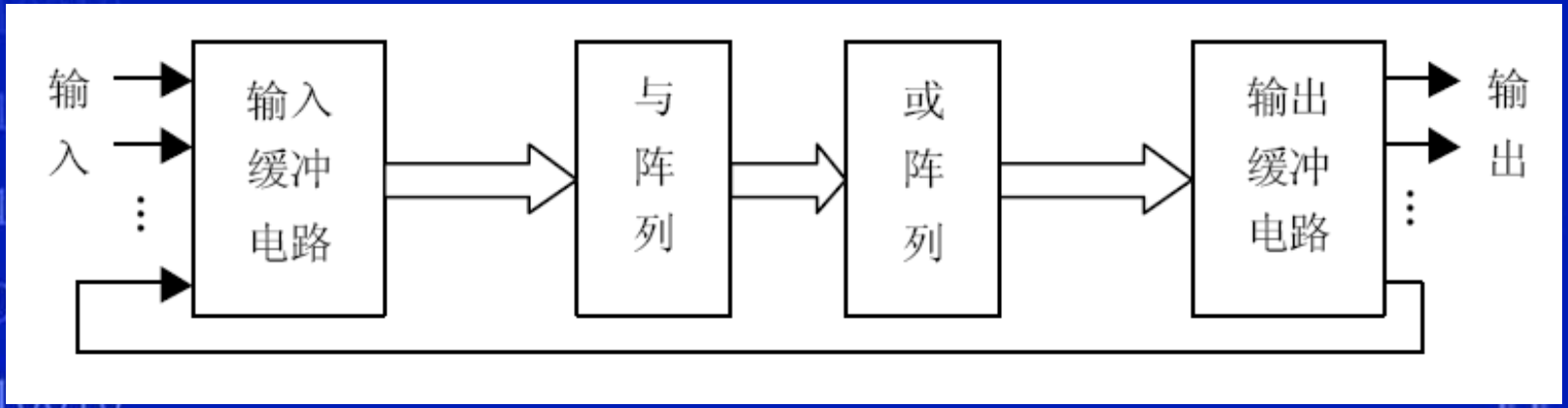
⑦ATMEL：中小规模PLD做得不错。ATMEL也做了一些与Altera和Xilinx兼容的片子，但在品质上与原厂家还是有一些差距，在高可靠性产品中使用较少，多用在低端产品上。

2.1.1 PLD分类

- 根据其集成度和结构复杂度的不同，大致可分为3类：
  - 简单可编程逻辑器件（Simply Programmable Logic Device，SPLD）
  - 大规模可编程逻辑器件
    - 复杂可编程逻辑器件（Complex Programmable Device，CPLD）
    - 现场可编程门阵列（Field Programmable Gate Array，FPGA）



SPLD基本原理结构框图



SPLD分类

- PLA(可编程逻辑阵列，Programmable Logic Array),70年代初产品，价格较贵，应用不广泛；
- PAL(可编程阵列逻辑，Programmable Array Logic),1977年美国MMI公司推出，价格较PLA便宜，应用比较广泛；
- GAL(通用阵列逻辑，Generic Array Logic), 1983年美国Lattice公司推出，价格便宜，应用最广泛。

分类	与阵列	或阵列	输出方式
PROM	固定	可编程	TS，OC
PLA	可编程	可编程	TS，OC
PAL	可编程	固定	TS，I/O，寄存器反馈
GAL	可编程	固定	用户定义

1.简单可编程逻辑器件SPLD

- 简单可编程逻辑器件SPLD属于集成度和结构复杂度都比较小的可编程逻辑器件。
- 特点是具有可编程的与阵列、不可编程的或阵列、输出逻辑宏单元OLMC（Output Logic Macro Cell）和输入输出逻辑单元IOC（In Output Cell）。
- 这类器件适合于规模较小的逻辑设计，典型器件有Lattice生产的GAL16V8、GAL22V10等。

2.复杂可编程逻辑器件

- 复杂可编程逻辑器件CPLD是阵列型高密度PLD器件，大多采用了乘积项、EEPROM(或Flash)工艺等技术，其集成度大于GAL22V10，具有高密度、高速度和低功耗等特点。
- 此类器件有更大的与阵列和或阵列，增加了大量的宏单元和布线资源，触发器的数量明显增多，多用于较大规模的逻辑设计。

3.现场可编程门阵列

- 现场可编程门阵列（Field Programmable Gate Array, FPGA），是集成度和结构复杂度最高的可编程逻辑器件，大部分FPGA采用基于SRAM的查找表LUT（Look Up Table）逻辑结构形式，且其内部采用矩阵式结构分布，并拥有更多的触发器和布线资源，多用于10,000门以上的大规模设计，适合做复杂的时序逻辑，如数字信号处理和各种算法。
- 目前常用器件主要有Xilinx公司和Altera公司。

2.1.2 PLD的开发流程

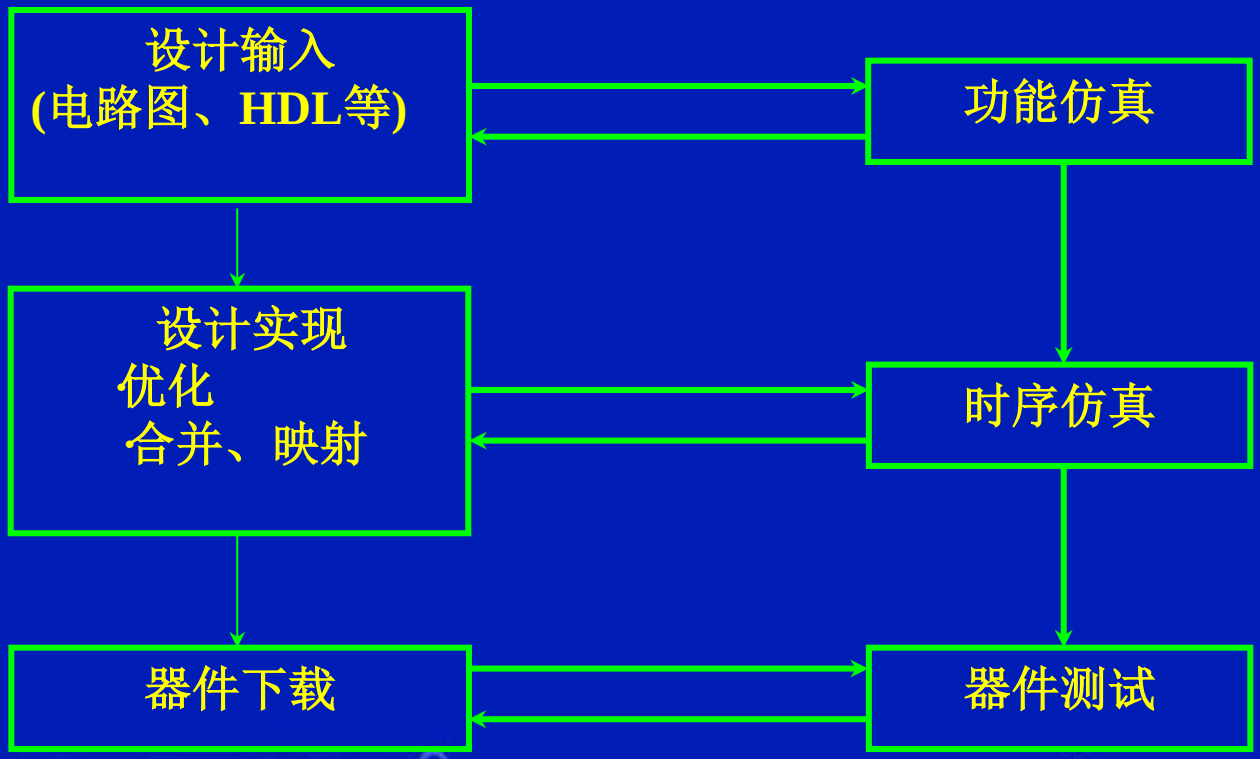


图2-1-1 PLD的一般开发流程

2.1.2 PLD的开发流程

- 设计输入是由设计者对器件所实现的数字系统的逻辑功能进行描述。设计输入一般采用一种或几种设计输入源文件来描述设计对象，以实现设计要求。常用的设计输入方法有原理图输入、状态机输入、HDL（Hardware Description Language）等输入方式。其中HDL是使用最为普遍的一种设计方法，具有较好地可读性和可移植性。在设计过程中，往往采用层次化设计方法，分模块、分层次地进行设计描述，即采用自定向下的设计方法。
- 设计实现是指从设计输入文件到熔丝图文件或位图文件的编译过程。设计实现阶段需要完成逻辑分割、器件布局和布线工作，这些一般是由可编程逻辑器件的开发系统自动完成的，而设计者可以根据设计需要用一些约束来进行干预，也可以使用编辑功能直接修改设计布局、布线结构。

2.1.2 PLD的开发流程

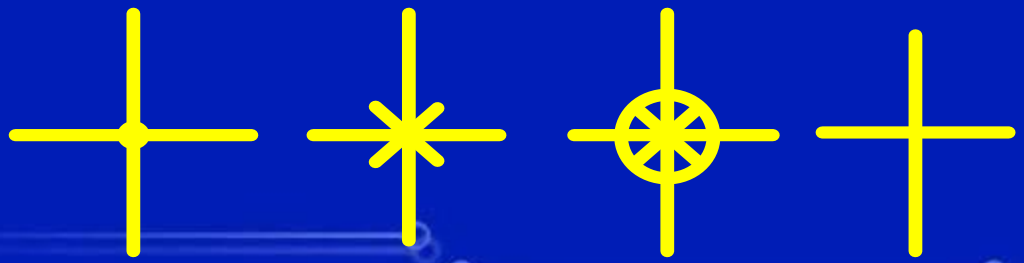
- 功能仿真主要是对所设计的电路及所有输入的电路进行功能验证，以便设计者及时修改设计中存在问题，实现设计要求。一般可以利用波形编辑器对所需检验的单个功能模块或系统输入一些数字信号，通过仿真软件得到输出波形，从而可以检验设计的正确性。需要指出的是功能仿真一般不包含设计器件的信息，是一种理想的仿真，只能验证设计的逻辑功能，与时序无关。
- 时序仿真是在完成布局、布线之后进行的，这时仿真工具会根据设计实现结果，给出设计中各个信号之间的逻辑功能和时序关系，以便设计者对设计的可靠性和稳定性进行评价。一般情况下通过时序仿真可以发现在功能仿真过程中不能发现的问题，例如竞争-冒险等问题。

2.1.2 PLD的开发流程

- 器件下载是将设计实现所给出的最终设计结果文件，即编程和配置数据文件，写入或加载到设计目标芯片的过程。目前主要下载方式是边界扫描方式，即通过专用的下载电缆，将设计结果的数据文件写入到目标芯片。
- 器件测试是确定设计的目标芯片是否符合设计要求，能否满足系统的工作需要。如果发现问题，则需要重新回到设计输入修改设计，直到满足系统要求。

2.1.3 PLD的逻辑表示

- 1.逻辑阵列交叉点的逻辑表示
- 实体连接，即行线与列线相互连接在一起，是不可以编程的，用实点表示。图2-1-2(b)的行线与列线在交叉点处采用×或连接，表示该交叉点是个可编程点。若PLD器件是采用熔丝工艺的，则器件出厂时，可编程点处的熔丝都处于接通状态，因此在可编程点上处处都打×或。图2-1-2(c)表示可编程点被用户编程后，熔丝被烧断的情况。此时熔丝烧断的可编程点上的×消失，行线和列线不相接。

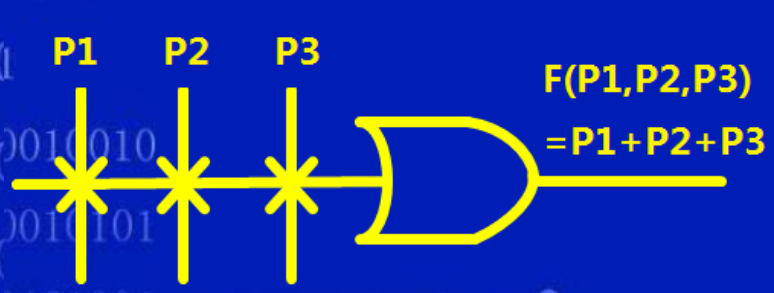


(a) 实体连接 (b) 可编程连接 (c)编程后熔丝烧断

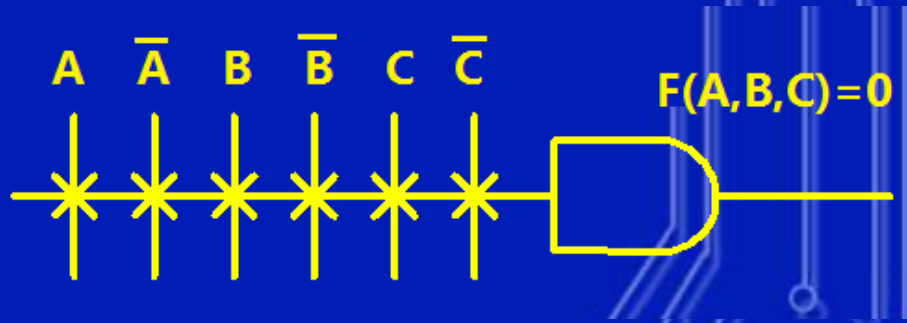
图2-1-2 阵列交叉点的PLD表示法

2.逻辑阵列的PLD表示

- PLD的逻辑阵列中通常包含与阵列和或阵列。
- 图2-1-3(a)是一个可编程与阵列的一般表示形式。与阵列的所有输入变量都称为输入项，并画成与行线垂直的列线以表示与阵列的输入，与阵列的输出称为乘积项。
- 图2-1-3(b)是一个可编程或阵列的一般表示形式，同与阵列表示方法相似。或阵列的输入常常是与阵列的乘积项输出，或阵列的输出是编程后保留熔丝各支路输入乘积项的逻辑或。



(a) 可编程的与阵列PLD表示



(b) 可编程的或阵列PLD表示

图2-1-3 可编程的与阵列和或阵列PLD表示

2.2 通用阵列逻辑GAL

1.GAL的基本阵列结构

- 通用阵列逻辑（General Array Logic，GAL）是Lattice公司于1985年首先推出的可编程逻辑器件。它采用了电擦除、电可编程的E2CMOS工艺制作，保证了GAL的高速度和低功耗，存取速度为12~40 ns，可以用电信号擦除并反复编程上百次。
- GAL器件是由可编程的与阵列、不可编程的或阵列、可编程的输出逻辑宏单元(OLMC)3部分主要电路构成。GAL器件可以实现老一代器件所有的各种输出电路工作模式，因此称为通用可编程逻辑器件。
- 常用的GAL器件有多种型号，这些器件的基本结构是相同的，只是内部可编程逻辑资源的多少不同而已，下面以GAL16V8为例对其结构加以介绍。

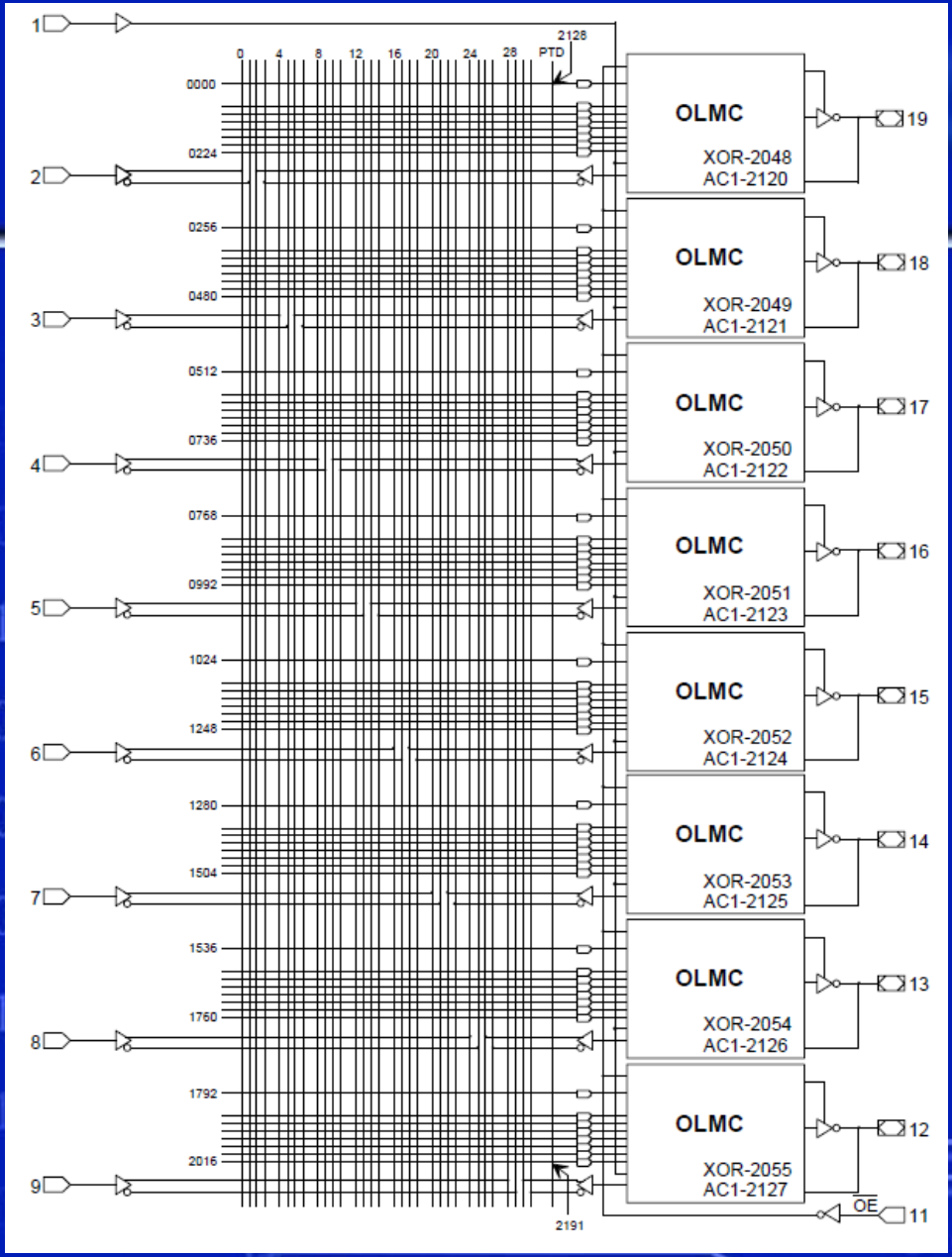


图2-2-1 GAL16V8内部结构图

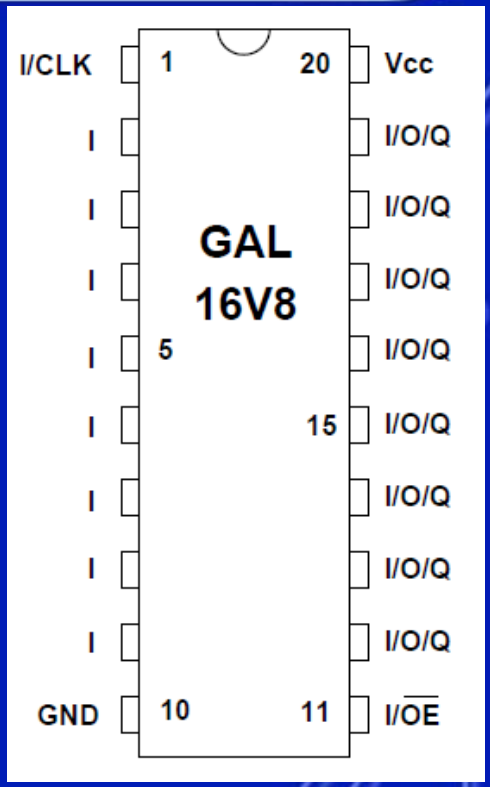


图2-2-2 GAL16V8引脚图

GAL16V8结构特点

① 包含 8个输入缓冲器，8个反馈缓冲器，8个输出三态缓冲器。

② 包含8个输出逻辑宏单元(OLMC12～OLMC19)。

③ 由8×8 个与门构成的与阵列，共形成64 个乘积项，与阵列共分8个阵列块。每个阵列块有8条行线，每条行线各接一个与门。与门的输出称为乘积项(与项)。每一个阵列块中最上面一个与门的输出称为第一与项。每个与门有32个输入项，由8个输入的原变量、反变量和8个反馈信号的原变量、反变量组成，故可编程与阵列共有32×8×8=2048 个可编程单元。

④ 1号引脚(I/CLK)经一级缓冲器引至OLMC的CLK端。

⑤ 8个OLMC的内部电路结构完全相同，外部引线稍有不同，2，3，4，5，6，7，8，9各引脚是专用输入引脚，1，11，12，13，14，17，18，19各引脚可通过编程组态为输入引脚。也就是说共有16个引脚可设置为输入。而12，13，14，15，16，17，18，19共有8个引脚可做输出引脚。这也是GAL16V8命名的由来。

GAL16V8结构特点

⑥ GAL16V8具有82位的控制字，可以通过编程控制OLMC的各种模式及输出组态，满足用户对各种输出电路形式的需要。这82位控制字分别是：

- *SYN*：1位的同步控制字；
- *AC*0：1位的结构控制字；
- *AC*1（*n*）：8位的结构控制字；
- *XOR*（*n*）：8位的极性控制字；
- *PTD*：64位的乘积项禁止控制字。

2.GAL的工作模式和逻辑组态

- GAL16V8系列器件的OLMC一共有3种工作模式，分为7种组态。三种工作模式是寄存器模式、复杂模式、简单模式。寄存器模式根据不同需求可以配置为寄存器输出组态和组合输出组态；复杂模式可以配置为有反馈组合输出和无反馈组合输出；简单模式可以配置为无反馈组合输出组态、本级组合输出邻级输入组态以及相邻输入组态。

表2-2-1 OLMC的工作模式和逻辑组态

OLMC的工作模式	逻辑组态
寄存器模式	寄存器输出组态 组合输出组态
复杂模式	有反馈组合输出 无反馈组合输出
简单模式	无反馈组合输出组态 本级组合输出邻级输入组态 相邻输入组态

3. GAL编程

- 厂家出厂的GAL芯片不具任何逻辑功能，必须借助GAL的开发软件和硬件设备对其进行编程写入，才能使空白的GAL芯片具有预期的逻辑功能。对GAL器件进行编程时，硬件环境需要有一台计算机，另外还要配置GAL编程写入器，通常称为编程器。
- GAL器件的编程主要有两种方式。一种是使用第三方提供的通用编程器及相应的编程工具软件对器件进行编程。通过此种方法在完成对器件的编程操作的同时，还可以完成对器件的擦除、读回、加密等辅助操作。GAL器件被加密后，如果对其进行读出操作，器件将进行自我保护。
- 另一种编程方式是在系统编程（In System Programmable，ISP）。所谓在系统编程就是可以将器件先焊接到目标系统中，只要提供四条专用的编程引脚（MODE、SCLK、DIN、DOUT）就可以对器件进行编程了，这种编程方式不再需要编程器，仅需要一条下载电缆就可以对器件进行编程，而且同样具有擦除、读回、加密等辅助功能。但是这种编程方式仅支持具有在系统编程功能的GAL器件，例如ispGAL20V8，ispGAL22V10等器件。

GAL器件设计步骤：

① 按逻辑要求选择器件类型，主要考虑输入输出管脚数量。

② 选择一种合适的编程软件编制相应的源文件。

③ 经过相应的编译程序生成.JED文件（熔丝图文件）。

④ 利用相应的编程方式对GAL进行编程，并且可以进行检验及对GAL进行加密。

2.3 复杂可编程逻辑器件CPLD

- 复杂可编程逻辑器件 (Complex Programmable Logic Device， CPLD)，是从PAL和GAL器件发展出来的器件，相对而言规模大，结构复杂，属于大规模集成电路范围。
- 使用者可以根据自己需要自行构造逻辑功能的数字电路或数字系统。
- 设计方法是借助集成开发软件平台，用原理图、硬件描述语言等方法，生成相应的目标文件，通过编程器将代码传送到目标芯片中，实现设计的数字系统。
- 常用的CPLD器件有：
  - Xilinx公司的XC95144、XC9572
  - ALTERA公司的EPM240T100、EPM570T144

2.3 复杂可编程逻辑器件CPLD

- CPLD MAX II有EPM240、EPM570和EPM1270器件。
- CPLD MAX II适合需要较多I/O与LCD显示器、小键盘、闪存和存储器接口的应用。

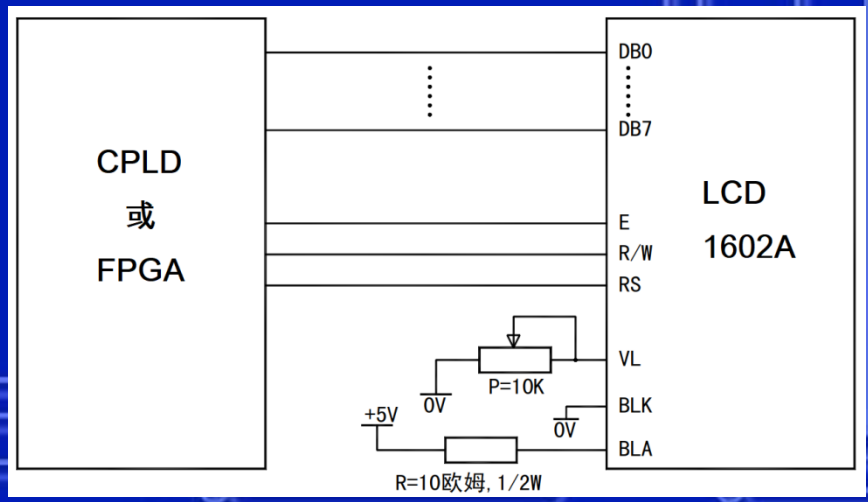
例2-3-1 Altera CPLD应用实例：在CPLD EPM240T100 开发板的LCD1602上，显示“Digital Logic”。

- 如果需要自己设计CPLD开发板，需要详细了解要用的CPLD芯片工作原理等；如果买开发板，需要了解开发板的原理图，所用器件连接方式，编制程序。
- 在CPLD的EPM240T100开发板上，连接LCD1602原理图，用于管脚配置Pin Planner。

LCD1602A接口信号说明

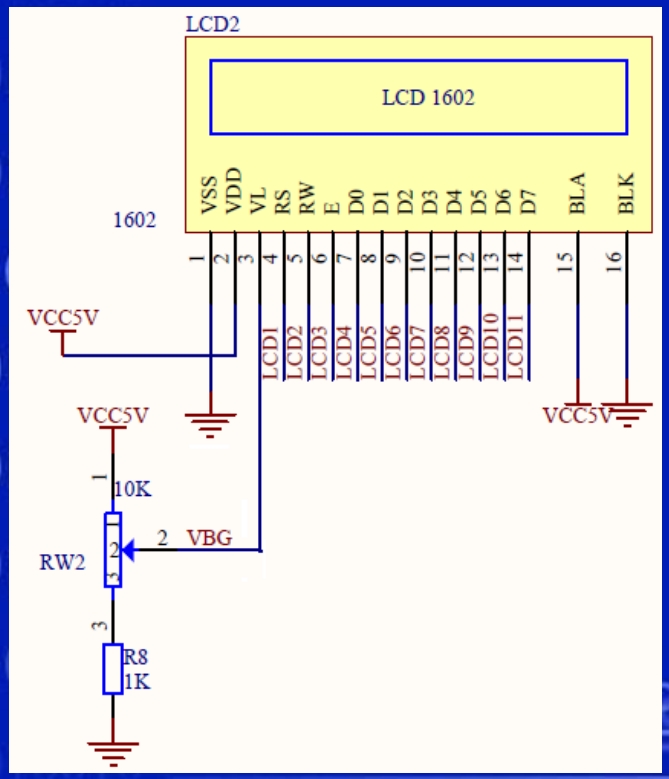
编号	符号	引脚说明	编号	符号	引脚说明
1	VSS	电源地	9	D2	Data I/O
2	VDD	电源正极	10	D3	Data I/O
3	VL	液晶显示偏压信号	11	D4	Data I/O
4	RS	数据/命令选择端（H/L）	12	D5	Data I/O
5	R/W	读/写选择端（H/L）	13	D6	Data I/O
6	E	使能信号	14	D7	Data I/O
7	DO	Data I/O	15	BLA	背光源正极
8	D1	Data I/O	16	BLK	背光源负极

LCD1602A与开发板的连接



例2-3-1 CPLD应用实例：
在CPLD EPM240T100 开发板的LCD1602上，显示“Digital Logic”。

- 如果需要自己设计CPLD开发板，需要详细了解要用的CPLD芯片工作原理等；如果买开发板，需要了解开发板的原理图，所用器件连接方式，编制程序。
- 在CPLD的EPM240T100开发板上，连接LCD1602原理图，用于管脚配置Pin Planner。




```
set_location_assignment PIN_6   -to  lcd[1]
set_location_assignment PIN_5   -to  lcd[2]
set_location_assignment PIN_4   -to  lcd[3]
set_location_assignment PIN_3   -to  lcd[4]
set_location_assignment PIN_2   -to  lcd[5]
set_location_assignment PIN_1   -to  lcd[6]
set_location_assignment PIN_100 -to lcd[7]
set_location_assignment PIN_99  -to  lcd[8]
set_location_assignment PIN_98  -to  lcd[9]
set_location_assignment PIN_97  -to  lcd[10]
set_location_assignment PIN_96  -to  lcd[11]
set_location_assignment PIN_12  -to  clk
```

- 在编程之前，需要阅读LCD1602的原理手册，了解LCD1602工作原理。
- 要在LCD1602上显示“Digital Logic”,需首先了解LCD1602控制指令，以及指令发出的顺序。

指令如下：

1. 清显示，指令码01H，光标复位到地址00H位置 。
2. 光标复位，光标返回到地址00H。
3. 光标和显示位置设置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右移，高电平表示有效，低电平表示无效。
4. 显示开关控制。D：控制整体的显示开与关，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。
5. 光标或显示移位 S/C ：高电平时显示移动的文字，低电平时移动光标。
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，高电平时为双行显示，F：低电平时显示5X7的点阵字符，高电平时显示5X10的显示字符。
7. 字符发生器RAM地址设置。
8. DDRAM地址设置。
9. 读忙信号和光标地址 BF：忙标志位，高电平表示忙，此时模块不能接收命令或数据，如果为低电平表示不忙。

LCD1602液晶模块内部的控制器控制指令

序号	指令	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	清显示	0	0	0	0	0	0	0	0	0	1
2	光标返回	0	0	0	0	0	0	0	0	1	*
3	置输入模式	0	0	0	0	0	0	0	1	I/D	S
4	显示开/关控制	0	0	0	0	0	0	1	D	C	B
5	光标或字符移位	0	0	0	0	0	1	S/C	R/L	*	*
6	置功能	0	0	0	0	1	DL	N	F	*	*
7	置字符发生存贮器地址	0	0	0	1	字符发生存贮器地址					
8	置数据存贮器地址	0	0	1	显示数据存贮器地址						
9	读忙标志或地址	0	1	BF	计数器地址						
10	写数到 CGRAM 或 DDRAM）	1	0	要写的数据内容							
11	从 CGRAM或 DDRAM读数	1	1	读出的数据内容							

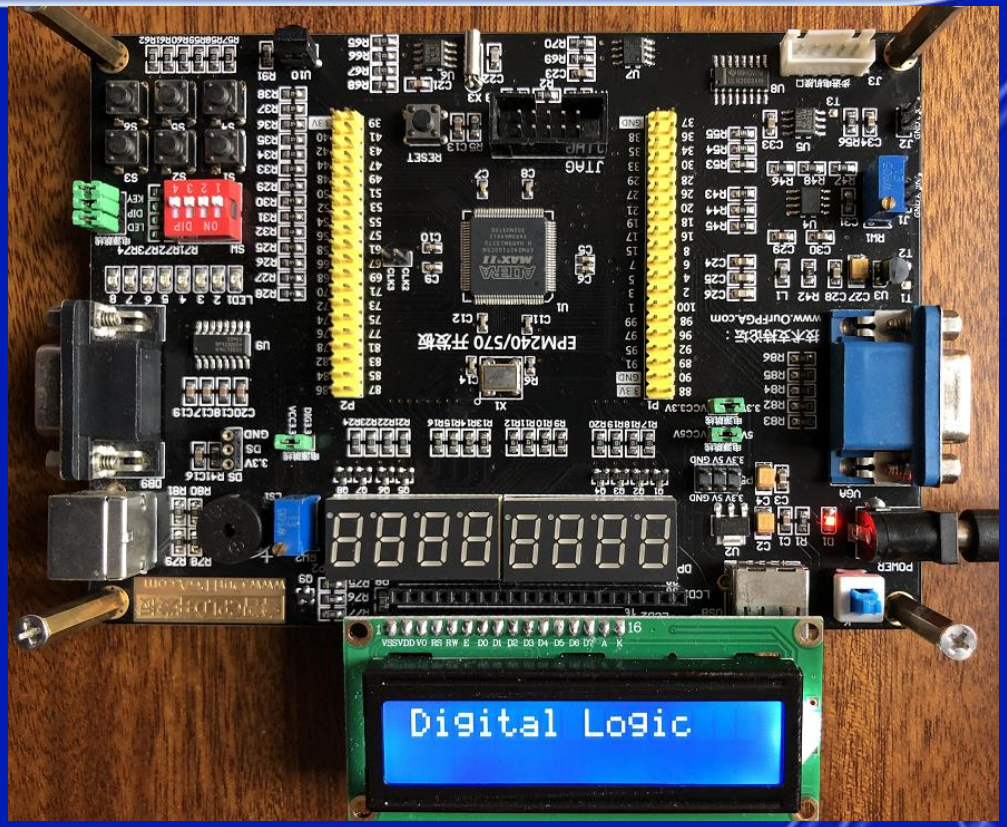
CPLD LCD1602显示“Digital Logic”源程序

```
//在1602上显示Digital Logic
module lcd(
input clk,                  //50MHz系统时钟
output reg [7:0] data, //8位数据
output reg  rs, //数据或命令选择 命令为0
output wire rw, //读写，写为0
output wire en  //使能
);
//寄存器类型说明
 reg e,clk_382;
 reg [16:0] counter;
 reg [4:0] current, next;
 reg [1:0] cnt;
//功能定义
assign en = clk_382 | e;
assign rw = 0;
//将50MHz时钟进行131072分频，为382Hz
always @ ( posedge clk ) begin
  if(counter != 17'h1ffff)
    counter <= counter + 1'b1;
    else
    counter <= 17'h0;
    end
//生成382Hz时钟clk_382
always @ ( posedge clk ) begin
  if( counter == 17'hffff )
     clk_382 <= ~clk_382;
     end
```

```
//设置并输出数据
always @(posedge clk_382) begin
  current = next;
  case( current )
 //置双行5X10显示字符
    0:    begin  rs<=0; data<=8'h30; next<=1; end
 //显示开关控制，开显示，无光标，不闪烁
    1:    begin  rs<=0; data<=8'h0c; next<=2; end
 //光标右移
    2:    begin  rs<=0; data<=8'h6;  next<=3; end
 //清显示
    3:    begin  rs<=0; data<=8'h1;  next<=4; end
 //发送数据
    4:    begin  rs<=1; data<="D";   next<=5;  end
    5:    begin  rs<=1; data<="i";   next<=6;  end
    6:    begin  rs<=1; data<="g";   next<=7;  end
    7:    begin  rs<=1; data<="i";   next<=8;  end
    8:    begin  rs<=1; data<="t";   next<=9;  end
    9:    begin  rs<=1; data<="a";   next<=10; end
    10:   begin  rs<=1; data<="l";   next<=11; end
    11:   begin  rs<=1; data<=" ";   next<=12; end
    12:   begin  rs<=1; data<="L";   next<=13; end
    13:   begin  rs<=1; data<="o";   next<=14; end
    14:   begin  rs<=1; data<="g";   next<=15; end
    15:   begin  rs<=1; data<="i";   next<=16; end
    16:   begin  rs<=1; data<="c";   next<=17; end
    17:   begin  rs<=0; data<=8'h00;
          if(cnt!=2'b10)  begin
             e <= 0; next <= 0; cnt <= cnt+1;
             end
             else  begin
                 next <= 17; e <= 1;
             end
          end
    default:  next <= 0;
    endcase
end
endmodule
```

程序编译通过后，配置管脚，下载运行后得到运行结果。

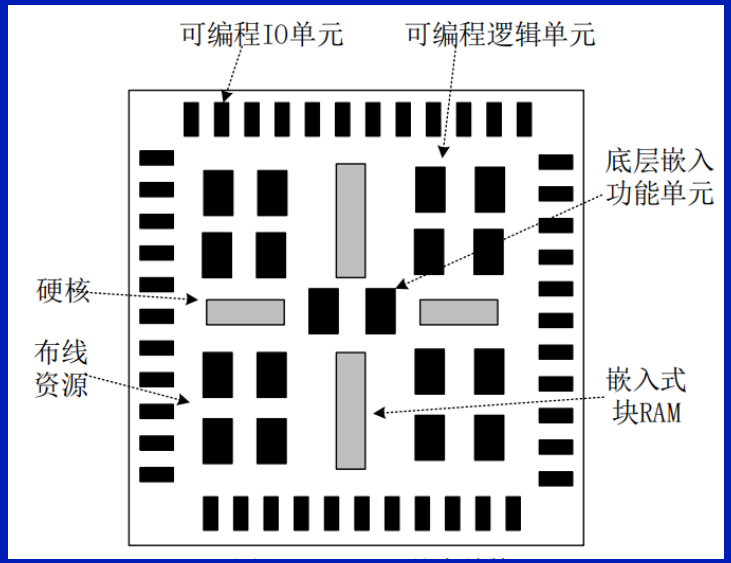
Node Name	Direction	Location
clk	Input	PIN_12
data[7]	Output	PIN_96
data[6]	Output	PIN_97
data[5]	Output	PIN_98
data[4]	Output	PIN_99
data[3]	Output	PIN_100
data[2]	Output	PIN_1
data[1]	Output	PIN_2
data[0]	Output	PIN_3
en	Output	PIN_4
rs	Output	PIN_6
rw	Output	PIN_5



2.4 现场可编程门阵列FPGA

- 现场可编程门阵列 (Field Programmable Gate Array, FPGA)与SPLD和CPLD相比，具有更高的密度、更快的工作速度和更大的编程灵活性，被广泛应用于各种电子类产品中。
- 现在应用较为广泛的FPGA主要生产厂家是Xilinx和Altera。FPGA 由Xilinx公司发明的，高端系列Xilinx有一定优势，Altera价格非常有优势。
- FPGA是采用查找表(LUT)结构的可编程逻辑器件的统称，大部分FPGA采用基于SRAM的查找表逻辑结构形式，但不同公司的产品结构也有差异。
- 常用的FPGA QFP封装器件有：
  - Xilinx公司的Spartan6
  - ALTERA公司的Cyclone IV

- FPGA组成
  - 可编程输入/输出单元
  - 基本可编程逻辑单元
  - 嵌入式块 RAM
  - 布线资源
  - 底层嵌入功能单元



FPGA组成：输入/输出单元（I/O Block，IOB）

- 是芯片与外界电路的接口，I/O 单元被 设计为可编程模式，即通过软件的灵活配置，可以适配不同的电气标准与 I/O 物理特性，可以调整匹配阻抗 特性、上下拉电阻、以及调整驱动电流的大小等。
- Artix7系列，用户 可用的 IO 为 250 个，可以满足大多数设计需求。从 1.2v 到 3.3v 都支持，可支持 DDR3 的 800Mb/S 高速输入输出。

FPGA组成：可配置逻辑模块（Configurable Logic Block，CLB）

- 是可编程逻辑的主体，根据设计灵活地改变其内部连接与配置，完成不同 逻辑功能。
- FPGA一般是基于 SRAM工艺的，其基本可编程逻辑单元几乎都是由查找表（LUT，Look Up Table） 和寄存器（Register）组成。
- Xilinx ARTIX 7系列 FPGA 内部查找表为 6 输入，查找表一般完成纯组合逻辑功能。FPGA 内部寄存器结构相当灵活，可以配置为带同步/异步复位或置位，时钟使能的触发器，也可以配置成 锁存器，FPGA 依赖寄存器完成同步时序逻辑设计。
- 每个 CLB 里包含2个逻辑片（Slice），每个 Slice 由 4 个查找表（LUT）、8 个触发器(FF)和其它一些逻辑所组成的。

- **FPGA组成：嵌入式块 RAM**
  - FPGA 内部嵌入可编程 RAM 模块，提高了FPGA 的应用范围和使用灵活性。不同器件的内嵌块 RAM 的结构不同。
  - Artix-7系列20~135 个双端口块 RAM，每个 RAM 有 36 Kb 容量、并且每个块 RAM有两个完全独立的端口，它们共享存储的数据。
  - 除了块 RAM，可以灵活地将 LUT 配置成 RAM、ROM、FIFO 等存储结构，这种技术被称 为分布式 RAM。
  - 根据设计需求，块 RAM 的数量和配置方式是器件选型的一个重要标准。

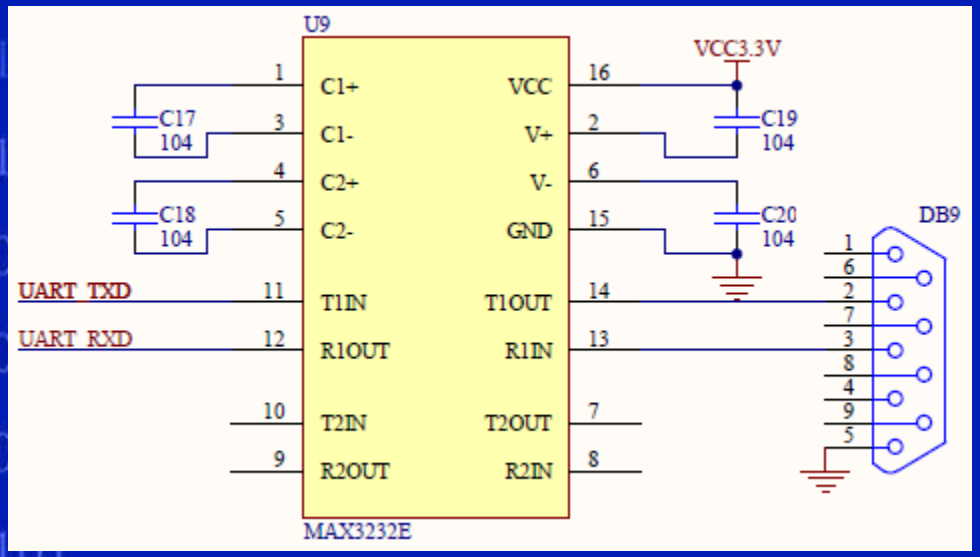
- **FPGA组成：布线资源**
  - 布线资源连通 FPGA 内部的所有单元，而连线的长度和工艺决定着信号在连线上的驱动能力和传输速 度。
  - FPGA 芯片内部有丰富的布线资源，根据工艺、长度、宽度和分布位置的不同而划分为 4 类：
    - 第一类是全局布线资源，用于芯片内部全局时钟和全局复位/置位的布线；
    - 第二类是长线资源，用以完成芯片 Bank 间的高速信号和第二全局时钟信号的布线；
    - 第三类是短线资源，用于完成基本逻辑单元之间的逻辑互连和布线；
    - 第四类是分布式的布线资源，用于专有时钟、复位等控制信号线。
  - 在实际中设计者不需要直接选择布线资源，布局布线器可自动地根据输入逻辑网表的拓扑结构和约束条件 选择布线资源来连通各个模块单元。

- **FPGA组成：底层嵌入功能单元**
  - 底层嵌入功能单元的概念比较笼统，这里指的是通用程度较高的嵌入式功能模块，比如 PLL （Phase Locked Loop）、DLL（Delay Locked Loop）、DSP、CPU 等。
  - 随着 FPGA 的发展，这些模块被越 来越多地嵌入到 FPGA 的内部，以满足不同场合的需求。

例2-4-1 Altera FPGA应用实例：

EP4CE6E22C6开发板的UART应用，通过串口接收PC发送字符，将收到的字符发送给PC 。

解：了解Altera 的EP4CE6E22C6开发板的UART原理图以及管脚连接，用于输入输出管脚配置。

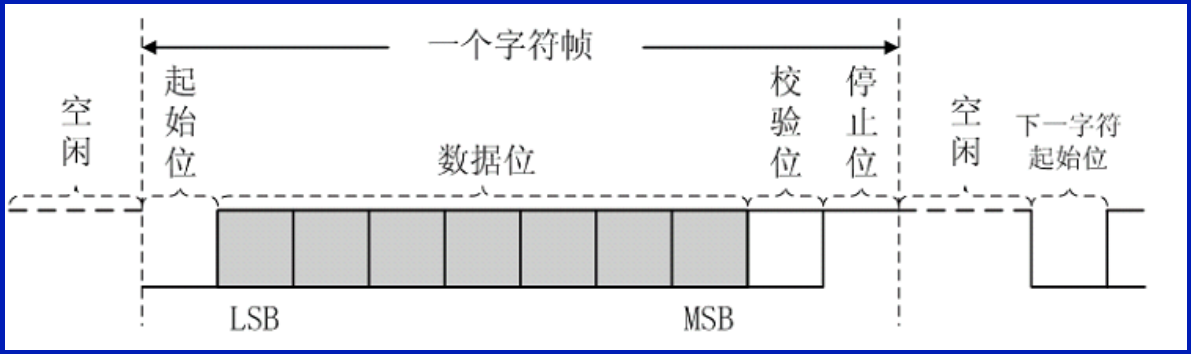


UART	FPGA引脚号
UART_TXD	114
UART_RXD	115

了解UART工作原理。

① 在串口的总线上默认状态是“高电平”，一帧数据开始传输需要先拉低电平，输出0。之后是8个数据位，最后是校验位和停止位。

② 串口传输过程需双方约定传输速度，称“波特率”。常用波特率有9600 bps 和 115200 bps，“9600 bps”表示每秒可以传输9600位。



串口传输一帧11位数据的时序图

开始程序设计。

主程序 uart_top.v

```
1   //开发板通过串口接收PC发送字符，然后将收到的字符发送给PC
2   module uart_top(
3       input           sys_clk,            //外部50M时钟
4       input           sys_rst_n,          //外部复位信号，低有效
5       input           uart_rxd,           //UART接收端口
6       output          uart_txd            //UART发送端口
7       );
8   wire        uart_en_w;                  //UART发送使能
9   wire [7:0] uart_data_w;                 //UART发送数据
10  uart_recv(
11      .sys_clk        (sys_clk),
12      .sys_rst_n      (sys_rst_n),
13      .uart_rxd       (uart_rxd),
14      .uart_done      (uart_en_w),
15      .uart_data      (uart_data_w)
16      );
17  uart_send(
18      .sys_clk        (sys_clk),
19      .sys_rst_n      (sys_rst_n),
20      .uart_en        (uart_en_w),
21      .uart_din       (uart_data_w),
22      .uart_txd       (uart_txd)
23      );
24  endmodule
```

开始程序设计。

接收子程序 uart_recv.v (1/3)

```
1   //接收子程序
2   module uart_recv(
3       input             sys_clk,                  //系统时钟
4       input             sys_rst_n,                //系统复位，低电平有效
5       input             uart_rxd,                 //UART接收端口
6       output  reg       uart_done,                //接收一帧数据完成标志信号
7       output  reg [7:0] uart_data                 //接收的数据
8       );
9
10  //parameter define
11  parameter  CLK_FREQ = 50000000;                 //系统时钟频率
12  parameter  UART_BPS = 9600;                     //串口波特率
13  localparam BPS_CNT  = CLK_FREQ/UART_BPS;        //为得到指定波特率，
14                                                  //需要对系统时钟计数BPS_CNT次
15  //reg define
16  reg        uart_rxd_d0;
17  reg        uart_rxd_d1;
18  reg [15:0] clk_cnt;                             //系统时钟计数器
19  reg [ 3:0] rx_cnt;                              //接收数据计数器
20  reg        rx_flag;                             //接收过程标志信号
21  reg [ 7:0] rxdata;                              //接收数据寄存器
22  //wire define
23  wire       start_flag;
24  //捕获接收端口下降沿(起始位)，得到一个时钟周期的脉冲信号
25  assign  start_flag = uart_rxd_d1 & (~uart_rxd_d0);
26  //对UART接收端口的数据延迟两个时钟周期
27  always @(posedge sys_clk or negedge sys_rst_n) begin
28      if (!sys_rst_n) begin
29          uart_rxd_d0 <= 1'b0;
30          uart_rxd_d1 <= 1'b0;
31      end
32      else begin
33          uart_rxd_d0  <= uart_rxd;
34          uart_rxd_d1  <= uart_rxd_d0;
35      end
36  end
```

开始程序设计。

接收子程序 uart_recv.v(2/3)

```
37    //当脉冲信号start_flag到达时，进入接收过程
38    always @(posedge sys_clk or negedge sys_rst_n) begin
39        if (!sys_rst_n)
40            rx_flag <= 1'b0;
41        else begin
42            if(start_flag)                                  //检测到起始位
43                rx_flag <= 1'b1;                            //进入接收过程，标志位rx_flag拉高
44            else if((rx_cnt == 4'd9)&&(clk_cnt == BPS_CNT/2))
45                rx_flag <= 1'b0;                            //计数到停止位中间时，停止接收过程
46            else
47                rx_flag <= rx_flag;
48        end
49    end
50    //进入接收过程后，启动系统时钟计数器与接收数据计数器
51    always @(posedge sys_clk or negedge sys_rst_n) begin
52        if (!sys_rst_n) begin
53            clk_cnt <= 16'd0;
54            rx_cnt  <= 4'd0;
55        end
56        else if ( rx_flag ) begin                           //处于接收过程
57            if (clk_cnt < BPS_CNT - 1) begin
58                clk_cnt <= clk_cnt + 1'b1;
59                rx_cnt  <= rx_cnt;
60            end
61            else begin
62                clk_cnt <= 16'd0;                           //对系统时钟计数达一个波特率周期后清零
63                rx_cnt  <= rx_cnt + 1'b1;                   //此时接收数据计数器加1
64            end
65        end
66        else begin                                          //接收过程结束，计数器清零
67            clk_cnt <= 16'd0;
68            rx_cnt  <= 4'd0;
69        end
70    end
```

开始程序设计。

接收子程序 uart_recv.v(3/3)

```
71    //根据接收数据计数器来寄存uart接收端口数据
72    always @(posedge sys_clk or negedge sys_rst_n) begin
73        if ( !sys_rst_n)
74            rxdata <= 8'd0;
75        else if(rx_flag)                                    //系统处于接收过程
76            if (clk_cnt == BPS_CNT/2) begin                 //判断系统时钟计数器计数到数据位中间
77                case ( rx_cnt )
78                 4'd1 : rxdata[0] <= uart_rxd_d1;    //寄存数据位最低位
79                 4'd2 : rxdata[1] <= uart_rxd_d1;
80                 4'd3 : rxdata[2] <= uart_rxd_d1;
81                 4'd4 : rxdata[3] <= uart_rxd_d1;
82                 4'd5 : rxdata[4] <= uart_rxd_d1;
83                 4'd6 : rxdata[5] <= uart_rxd_d1;
84                 4'd7 : rxdata[6] <= uart_rxd_d1;
85                 4'd8 : rxdata[7] <= uart_rxd_d1;    //寄存数据位最高位
86                default:;
87                endcase
88            end
89            else
90                rxdata <= rxdata;
91        else
92            rxdata <= 8'd0;
93    end
94    //数据接收完毕后给出标志信号并寄存输出接收到的数据
95    always @(posedge sys_clk or negedge sys_rst_n) begin
96        if (!sys_rst_n) begin
97            uart_data <= 8'd0;
98            uart_done <= 1'b0;
99        end
100       else if(rx_cnt == 4'd9) begin                       //接收数据计数器计数到停止位时
101           uart_data <= rxdata;                            //寄存输出接收到的数据
102           uart_done <= 1'b1;                              //并将接收完成标志位拉高
103       end
104       else begin
105           uart_data <= 8'd0;
106           uart_done <= 1'b0;
107       end
108   end
```

开始程序设计。

发送子程序 uart_send.v(1/3)

```
1  //发送子程序
2  module uart_send(
3      input          sys_clk,                  //系统时钟
4      input          sys_rst_n,                //系统复位，低电平有效
5      input          uart_en,                  //发送使能信号
6      input  [7:0]   uart_din,                 //待发送数据
7      output  reg    uart_txd                  //UART发送端口
8      );
9  //parameter define
10 parameter  CLK_FREQ = 50000000;              //系统时钟频率
11 parameter  UART_BPS = 9600;                  //串口波特率
12 localparam BPS_CNT  = CLK_FREQ/UART_BPS;     //为得到指定波特率，对系统时钟计数BPS_CNT次
13 //reg define
14 reg         uart_en_d0;
15 reg         uart_en_d1;
16 reg [15:0] clk_cnt;                          //系统时钟计数器
17 reg [ 3:0] tx_cnt;                           //发送数据计数器
18 reg        tx_flag;                          //发送过程标志信号
19 reg [ 7:0] tx_data;                          //寄存发送数据
20 //wire define
21 wire        en_flag;
22 //捕获uart_en上升沿，得到一个时钟周期的脉冲信号
23 assign en_flag = (~uart_en_d1) & uart_en_d0;
24 //对发送使能信号uart_en延迟两个时钟周期
25 always @(posedge sys_clk or negedge sys_rst_n) begin
26     if (!sys_rst_n) begin
27         uart_en_d0 <= 1'b0;
28         uart_en_d1 <= 1'b0;
29     end
30     else begin
31         uart_en_d0 <= uart_en;
32         uart_en_d1 <= uart_en_d0;
33     end
34 end
```

开始程序设计。

发送子程序 uart_send.v(2/3)

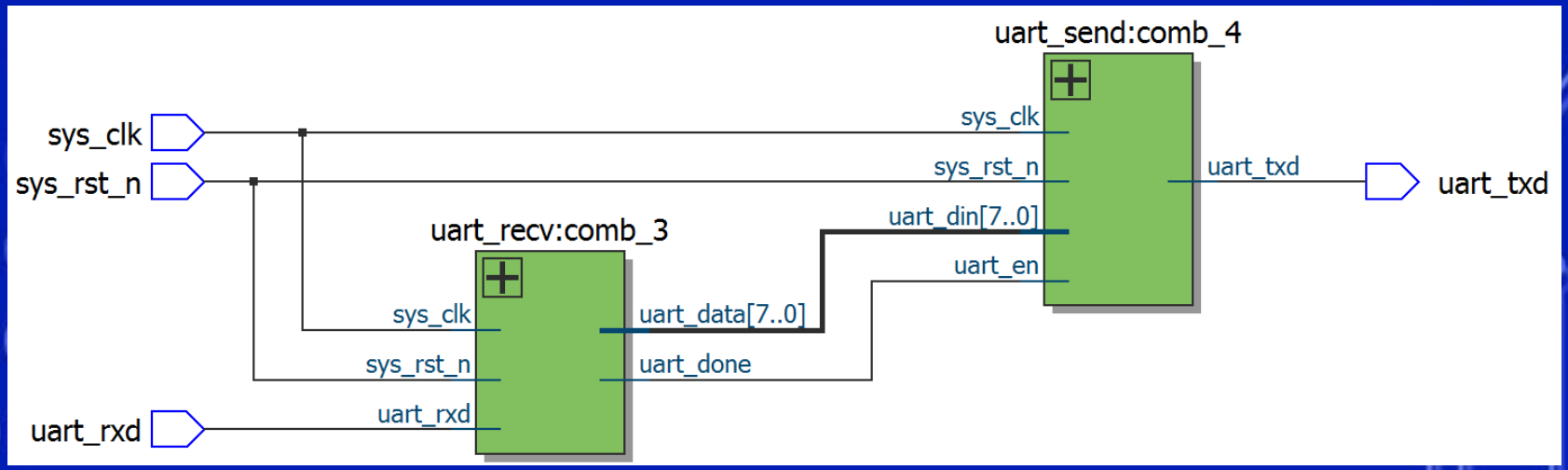
```
35    //当脉冲信号en_flag到达时,寄存待发送的数据，并进入发送过程
36    always @(posedge sys_clk or negedge sys_rst_n) begin
37        if (!sys_rst_n) begin
38            tx_flag <= 1'b0;
39            tx_data <= 8'd0;
40        end
41        else if (en_flag) begin                    //检测到发送使能上升沿
42                tx_flag <= 1'b1;                   //进入发送过程，标志位tx_flag拉高
43                tx_data <= uart_din;               //寄存待发送的数据
44            end
45            else
46            if ((tx_cnt == 4'd9)&&(clk_cnt == BPS_CNT/2))
47            begin                                  //计数到停止位中间时，停止发送过程
48                tx_flag <= 1'b0;                   //发送过程结束，标志位tx_flag拉低
49                tx_data <= 8'd0;
50            end
51            else begin
52                tx_flag <= tx_flag;
53                tx_data <= tx_data;
54            end
55    end
56    //进入发送过程后，启动系统时钟计数器与发送数据计数器
57    always @(posedge sys_clk or negedge sys_rst_n) begin
58        if (!sys_rst_n) begin
59            clk_cnt <= 16'd0;
60            tx_cnt  <= 4'd0;
61        end
62        else if (tx_flag) begin                    //处于发送过程
63            if (clk_cnt < BPS_CNT - 1) begin
64                clk_cnt <= clk_cnt + 1'b1;
65                tx_cnt  <= tx_cnt;
66            end
67            else begin
68                clk_cnt <= 16'd0;                  //对系统时钟计数达一个波特率周期后清零
69                tx_cnt  <= tx_cnt + 1'b1;          //此时发送数据计数器加1
70            end
71        end
```

开始程序设计。

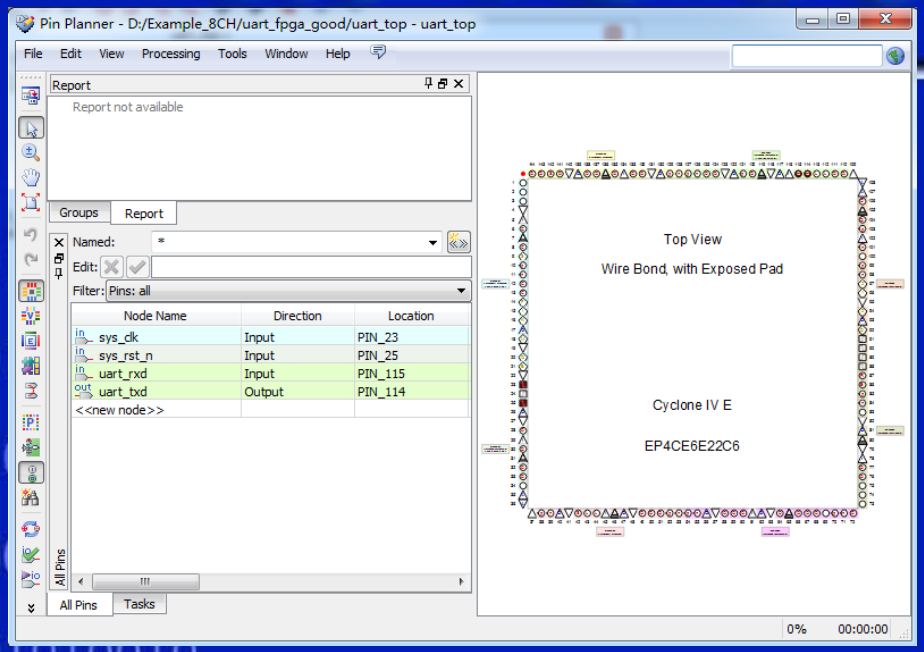
发送子程序 uart_send.v(3/3)

```
72      else begin                                  //发送过程结束
73          clk_cnt <= 16'd0;
74          tx_cnt  <= 4'd0;
75      end
76  end
77  //根据发送数据计数器来给uart发送端口赋值
78  always @(posedge sys_clk or negedge sys_rst_n) begin
79      if (!sys_rst_n)
80          uart_txd <= 1'b1;
81      else if (tx_flag)
82          case(tx_cnt)
83              4'd0: uart_txd <= 1'b0;             //起始位
84              4'd1: uart_txd <= tx_data[0];       //数据位最低位
85              4'd2: uart_txd <= tx_data[1];
86              4'd3: uart_txd <= tx_data[2];
87              4'd4: uart_txd <= tx_data[3];
88              4'd5: uart_txd <= tx_data[4];
89              4'd6: uart_txd <= tx_data[5];
90              4'd7: uart_txd <= tx_data[6];
91              4'd8: uart_txd <= tx_data[7];       //数据位最高位
92              4'd9: uart_txd <= 1'b1;             //停止位
93              default: ;
94          endcase
95      else
96          uart_txd <= 1'b1;                       //空闲时发送端口为高电平
97  end
98  endmodule
```

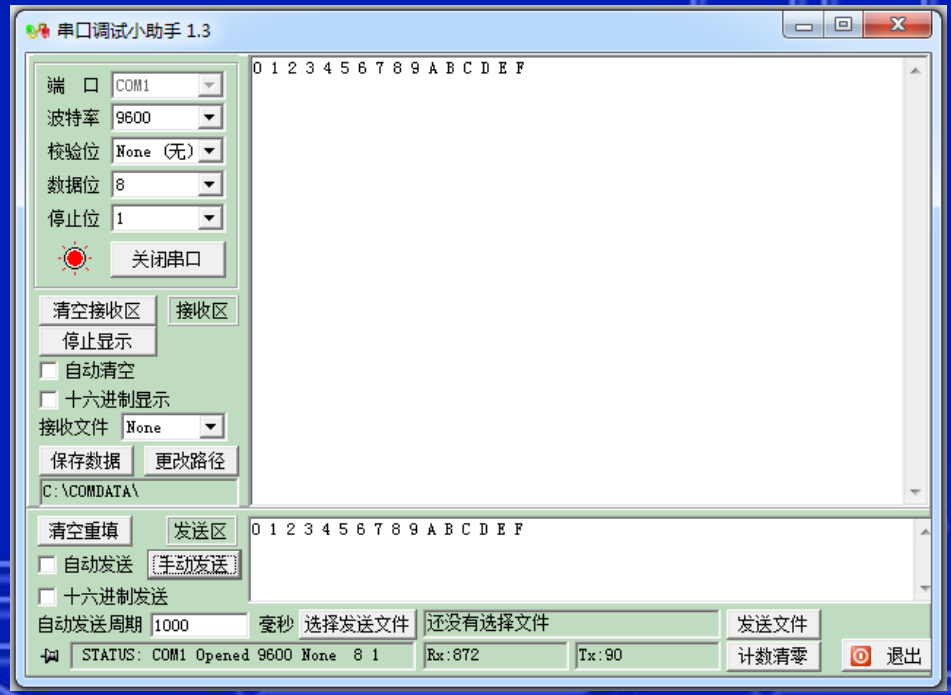
编译通过后得到的RTL Viewer



程序编译通过后，配置管脚，
下载运行后得到运行结果。

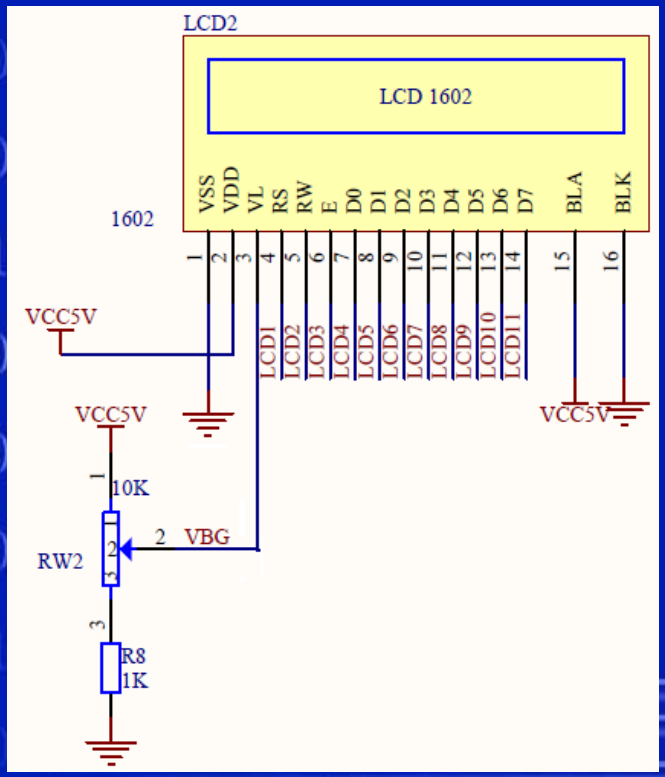


串口调试助手



例2-4-2 Xilinx FPGA应用实例：在FPGA Spartans6开发板的LCD1602上，显示“Digital Logic”。

- 如果需要自己设计FPGA开发板，需要详细了解要用的FPGA芯片工作原理等；如果买开发板，需要了解开发板的原理图，所用器件连接方式，编制程序。
- 在FPGA Spartans6开发板上，连接LCD1602原理图，用于管脚配置。

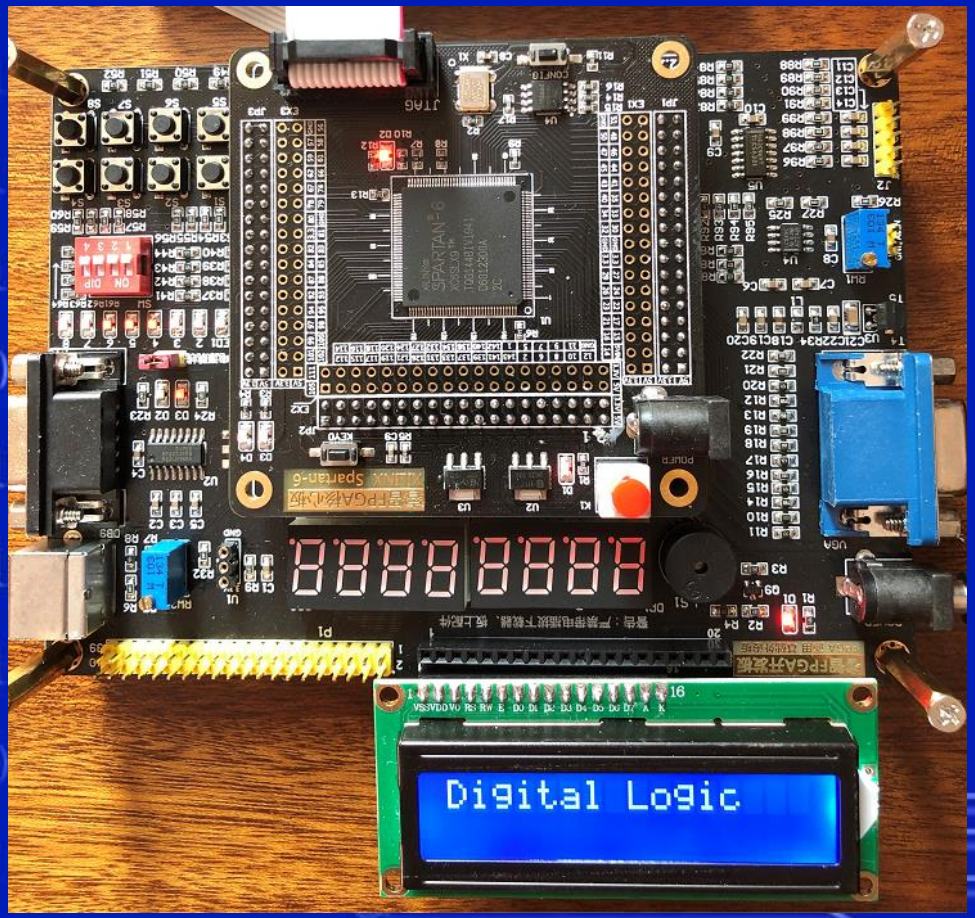



```
NET "clk" LOC=P55;

NET "rs" LOC=P131;
NET "rw" LOC=P134;
NET "en" LOC=P133;

NET "data<0>" LOC=P138;
NET "data<1>" LOC=P137;
NET "data<2>" LOC=P140;
NET "data<3>" LOC=P139;
NET "data<4>" LOC=P142;
NET "data<5>" LOC=P141;
NET "data<6>" LOC=P1;
NET "data<7>" LOC=P143;
```

管脚配置文件*.ucf

例2-4-2 Xilinx FPGA应用实例：
在FPGA Spartans6开发板的LCD1602上，显示“Digital Logic”。

Verilog源程序与例2-3-1相同。



FPGA和CPLD的开发应用选择

- 在FPGA和CPLD开发应用中，从以下几个方面考虑选型：
  - ① 应用需要的逻辑规模；
  - ② 应用的速度要求；
  - ③ 功耗：功耗通常由电压可以反应出来，功耗越低，电压也越低，一般来说，要选用低功耗、低电压的产品；
  - ④ 可靠性；
  - ⑤ 价格：要尽量选用价格低廉、易于购得的产品；
  - ⑥ 开发环境和开发人员熟悉程度。