

# **FPGA 应用开发**

## 实验一 Quartus II 软件操作(二)

### 一、实验目的

- (1) 掌握Quartus II文本输入法设计电路的步骤。
- (2) 掌握Quartus II混合输入法进行电路层次化设计。
- (3) 掌握在 Quartus II 中调用 ModelSim 进行仿真

### 二、实验内容及步骤

#### 1. Quartus II 文本输入法设计电路实例

首先要建立设计项目。

第 1 步：打开 QuartusII。

第 2 步：新建一个空项目。

执行 File->New Project Wizard 命令，进入新建项目向导。如图 4-1 所示，填入项目的名称，默认项目保存路径在 Quartus 安装下，也可修改为其他地址，视具体情况而定。

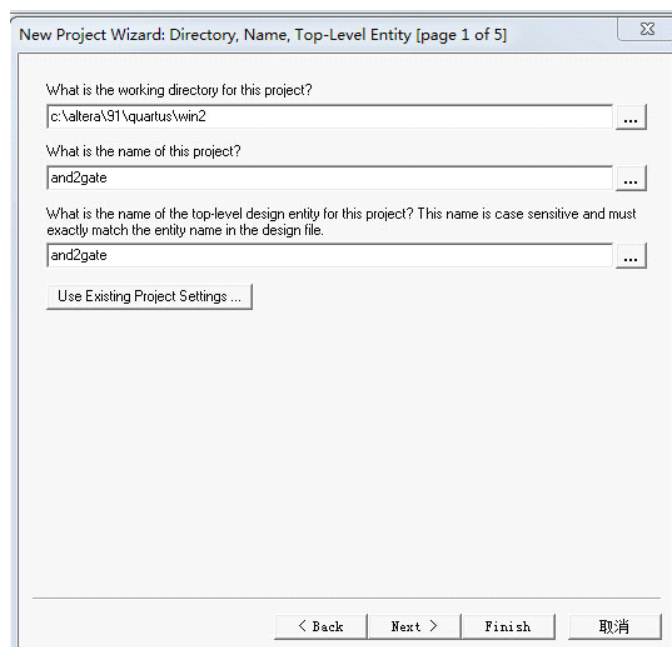


图4-1 新建项目向导

第3步：执行Next，进入向导的下一页进行项目内文件的添加操作，如果没有文件需要添加，则直接按Next即可。

第4步：指定CPLD/FPGA器件，如图4-2所示，选择芯片系列为“CycloneII”，型号为“EP2C35F672C6N”。选择型号时，可直接在列表框中查找，也可通过指定封装方式(Package)为“FBGA”、引脚数(Pin count)为“672”以及速度等级(Speed grade)为“6”这3个参数值来进行筛选。

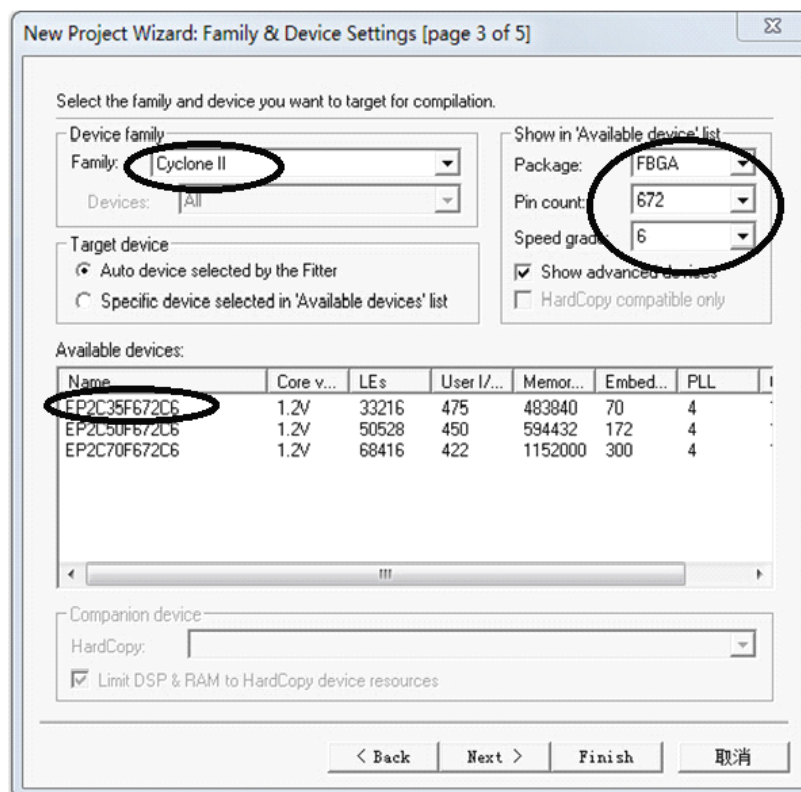


图 4-2 器件选择

第5步：向导的后面几步不做更改，直接按Next即可，最后按Finish结束向导。到此即完成了一个项目的新建工作。

第6步：新建一个Verilog HDL文件。

由于之前建立的项目还是一个空项目，所以接着需要为项目新建文件。执行File->New命令，在“Device Design Files”选项页中选择“Verilog HDL File”，然后点击OK按钮。这时自动新建一个名为Verilog1.v的文档，执行File->Save As命令，将文档另存为and2gate.v文件，结果如图4-3所示。

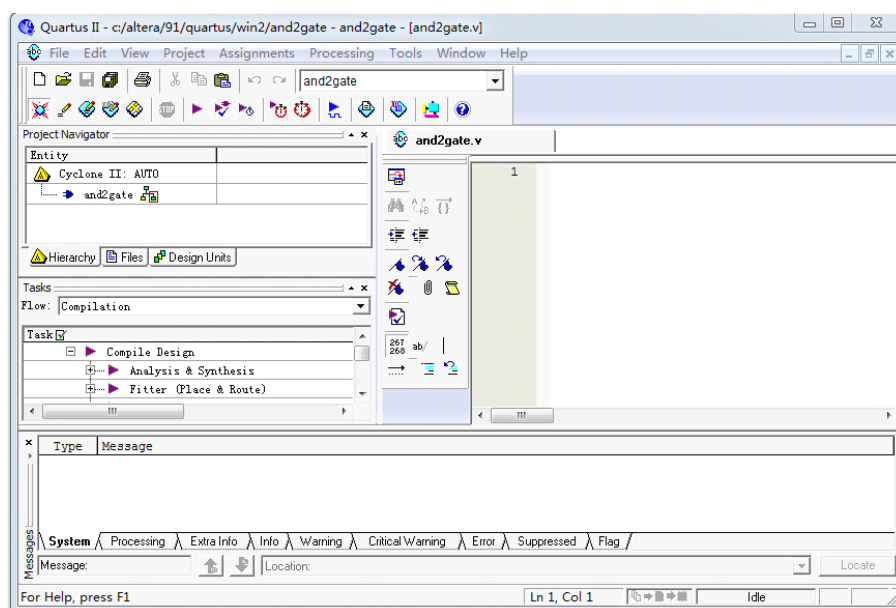


图4-3 新建Verilog HDL文件

第7步：代码输入。

在and2gate.v代码编辑窗口内输入以下代码：

```
module and2gate (y, a, b);  
input  a,b;  
output y;  
reg   y;  
always @ ( a or b)  
    y<=a & b;  
endmodule
```

第8步：代码的语法检查和编译。

.....（略）

此处与第三部分第一个实验（Quartus II图形输入软件操作）步骤中第11步～第20步一样。其中引脚分配参考表4-1。

表4-1 引脚分配

信号	FPGA引脚	DE2板上器件
a	PIN_N25	SW0
b	PIN_N26	SW1
y	PIN_AE22	LEDG0

第9步：程序下载（配置FPGA）。

用USB连接线连接DE2的USB Blaster端口和电脑即可进行程序的下载。在DE2平台上，可以对FPGA进行两种模式配置：一种是JTAG模式，通过USB Blaster直接配置FPGA，但掉电后，FPGA中的配置内容会丢失，再次上电需要用电脑重新配置；另一种是在AS模式下，通过USB Blaster对DE2平台上的串行配置器件EPCS16进行编程，平台上电后，EPCS16会自动配置FPGA。通过DE2平台上的SW19选择配置模式，SW19置于RUN位置，即选择JTAG模式配置；置于PROG位置，则选择AS模式对EPCS16进行编程。

JTAG模式配置：

1)用USB连接线连接DE2和电脑，将SW19置于RUN位置。选择Tools->Programmer命令，打开配置窗口，如图4-4所示。

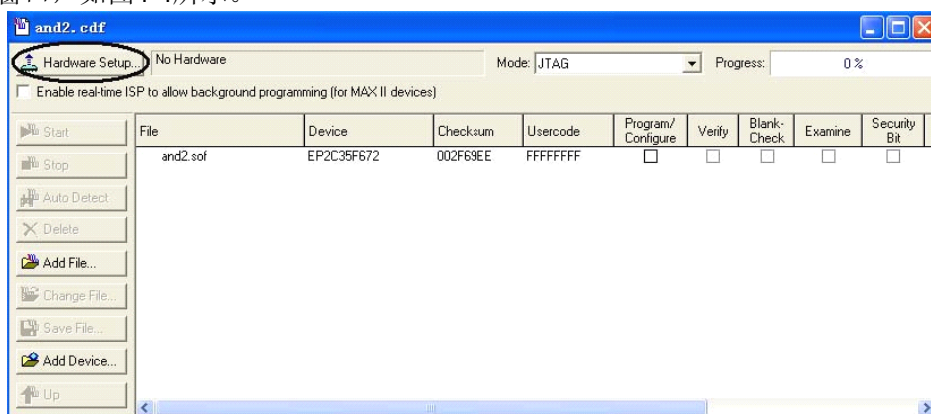


图4-4 下载配置窗口

2)图中第一列显示“No Hardware”，说明未指定硬件设备，单击Hardware Setup按钮，打开硬件设置窗口，如图4-5所示。双击列表框中的USB-Blaster，然后点击Close按钮，完成硬件设置。

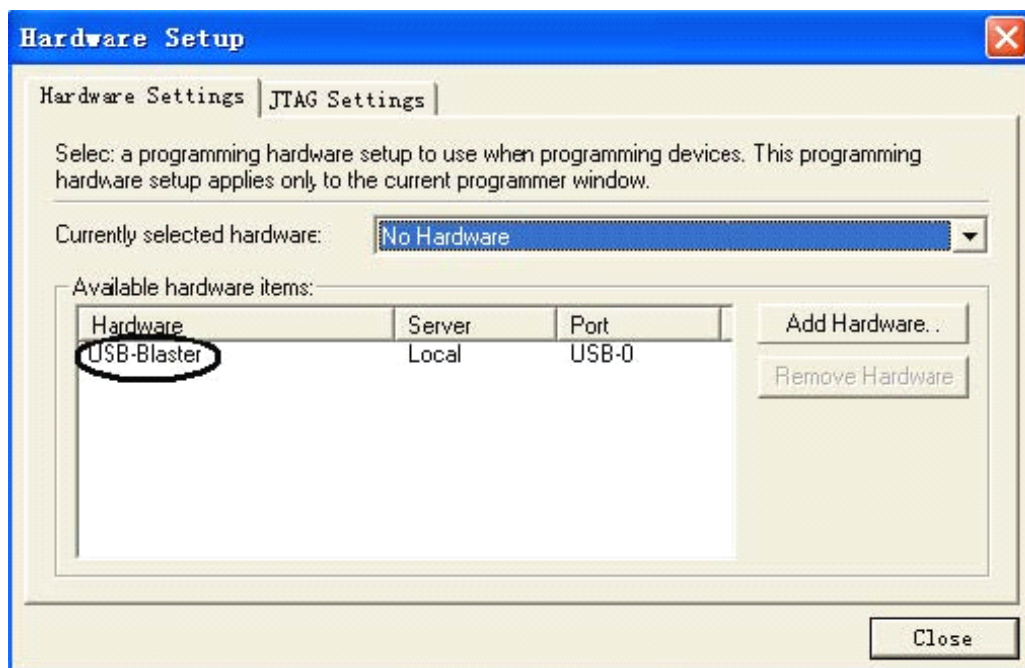


图4-5 硬件设置窗口

3)从图4-6可以看出，硬件已经设置完成，而且待配置的文件也已经在文件列表中。然后选中Program/Config选项，单击Start按钮，开始编程。编程结束后，即可在DE2上验证，将SW0和SW1置于1的位置，可以看到LEDG0灯亮。

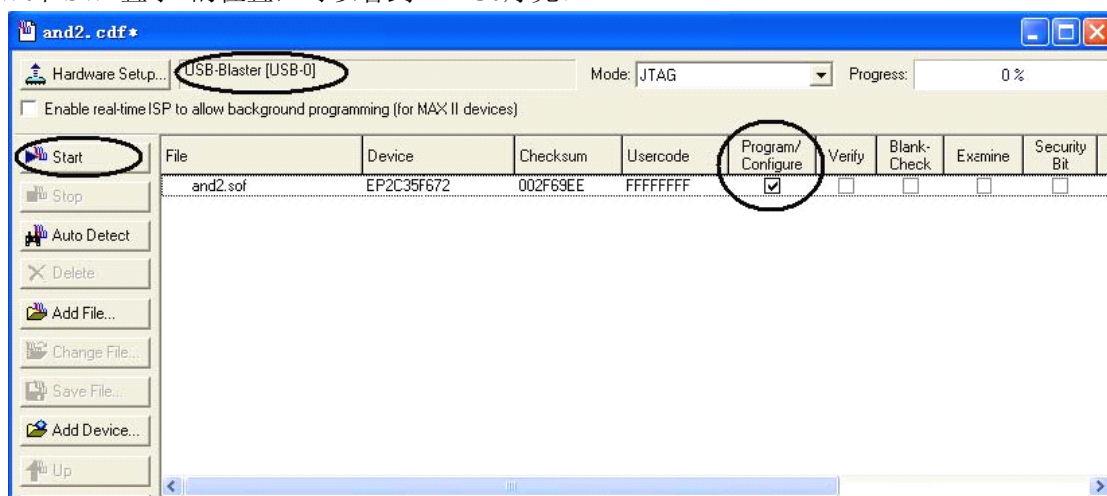


图4-6 下载配置窗口

AS模式配置：

1)首先需要设置串口配置器件，选择Assignments->Settings命令，打开设置窗口如图4-7所示。



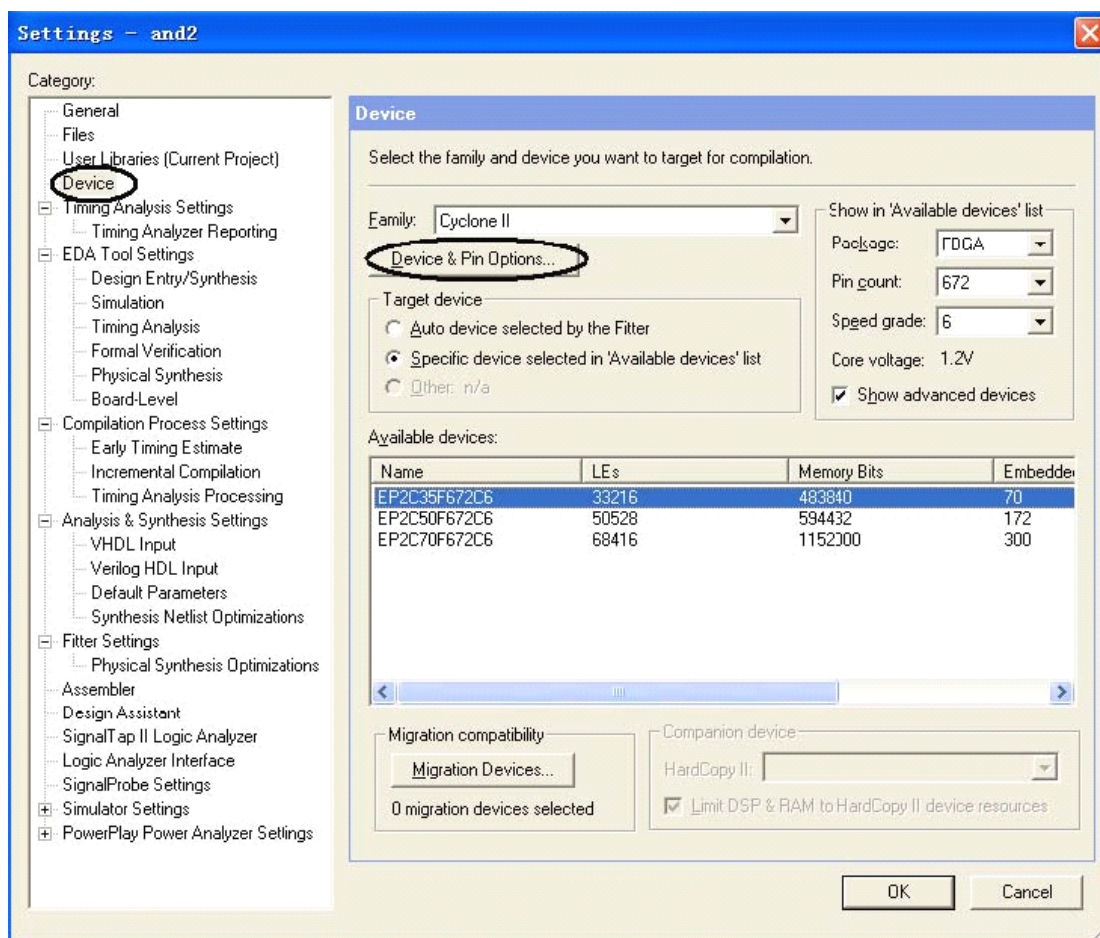


图4-7 串口配置器件设置窗口

2)单击 Device&Pin Options..按钮，打开器件及引脚选项窗口，如图 4-8 所示。切换到 Configuration页。在Configuration Device下拉框中选择“EPCS16”，单击OK按钮结束配置。

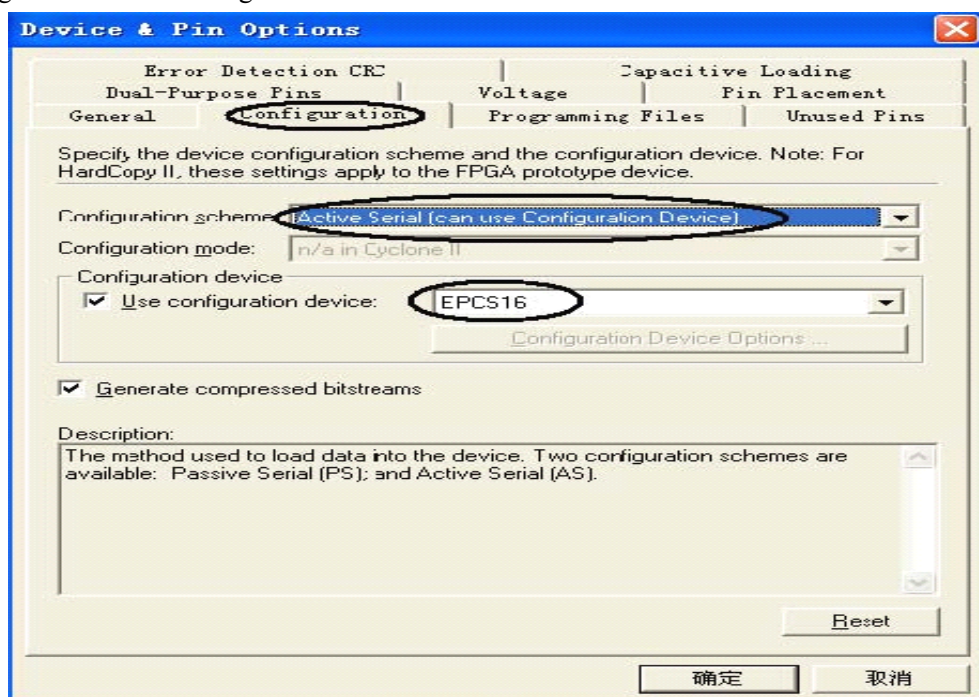


图4-8 器件及引脚选项窗口

3)将DE2上的SW19置于PROG位置。重新选择Tools->Programmer命令，打开编程窗口，在Mode下拉框中选择“Active Serial Programming”，这时会弹出图4-9所示的对话框，提示是否清除现有编程器件，选择“是”即可。

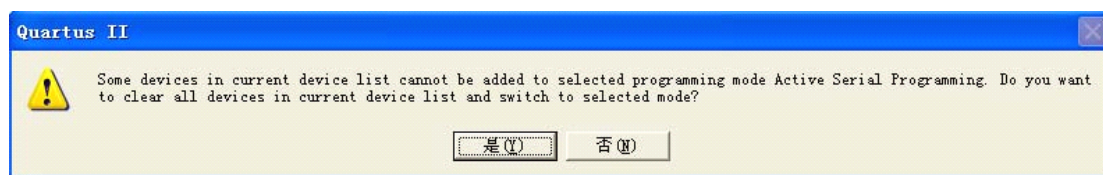


图4-9 提示对话框

4)接着需要重新添加配置文件，单击Add Files按钮，添加and2.pof配置文件。选中Program/Config选项，如图4-10所示。单击Start按钮，开始编程。编程结束后，将SW19置于RUN位置，再进行测试。

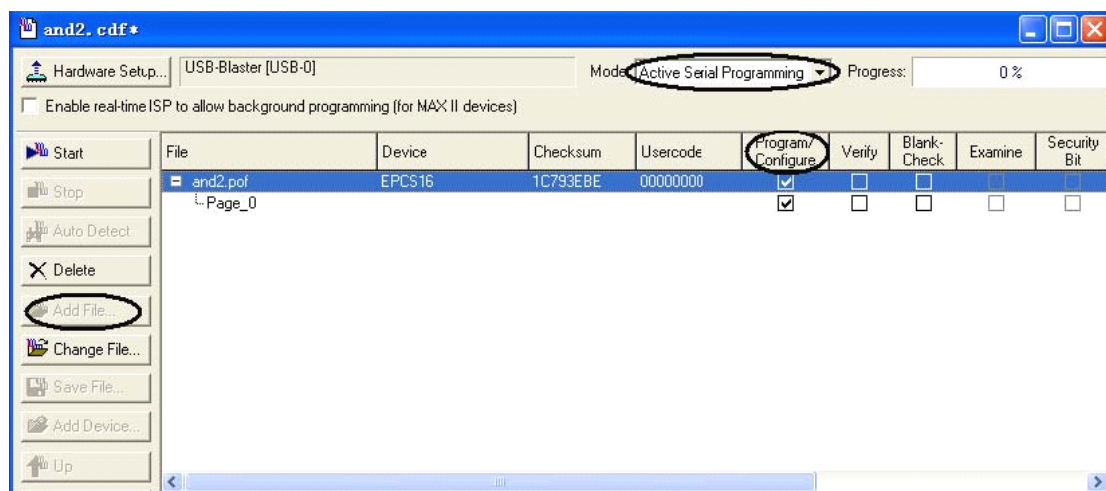


图4-10下载配置窗口

## 2 混合输入法完成层次化设计实例

采用混合输入法完成由与门和三态门组合成的三态与门。

### (1) 三态门

电路中共有2个输入信号：数据输入信号din和三态使能信号en。还有一个输出信号dout。三态门的逻辑功能是：当en='1'时，dout<=din；当en='0'时，dout<='Z'。

Verilog HDL程序如下：

```
module trigate ( dout, din,en );
input  din, en;
output dout;
reg  dout;
always @ ( en or din)
begin
    if (en)
        dout<=din;
    else
        dout<=1'bZ;
    end
endmodule
```

实验步骤如下：

第1步：在前面那个项目的基础上新建一个verilog HDL文件，起名为trigate.v，并输入上面的源程序。

第2步：在项目导向（Project Navigator）窗口中，如图4-11所示。选择文件（Files）管理页面，点开Device Design Files项，右击trigate.v文件，选择“Set as Top-Level Entity”选项。目的是将trigate.v文件设为项目的顶层实体。

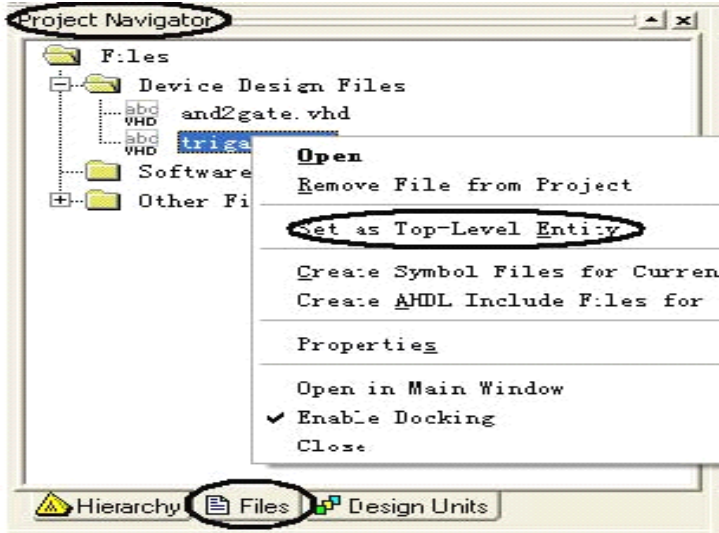


图4-11 项目导向（Project Navigator）窗口

第3步：对源程序进行语法检查，直到程序无误。

第4步：功能仿真，新建矢量波形图，起名为trigate.vwf，仿真结果如图4-12所示。

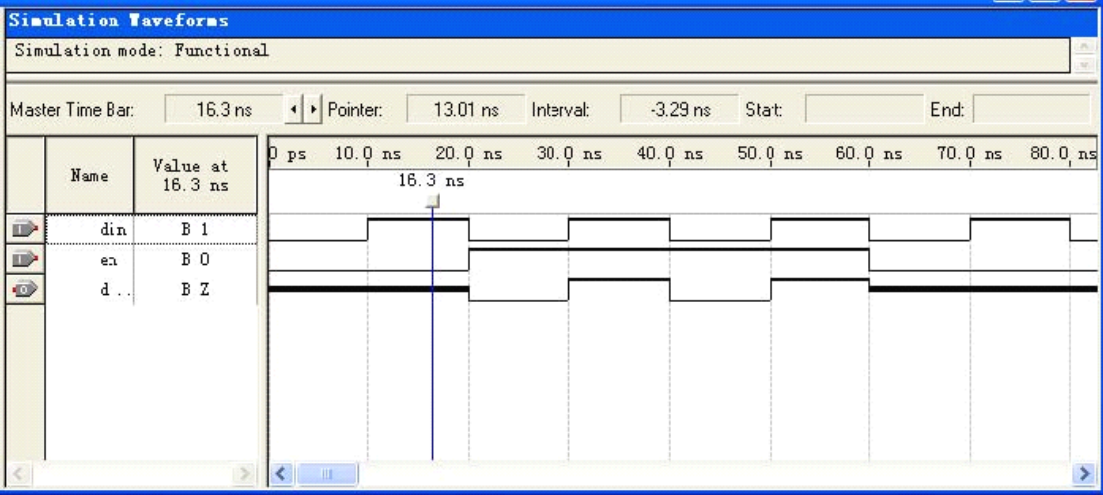


图4-12 仿真结果

第5步：按照表4-2进行引脚分配。重新编译，并下载。

表4-2 引脚分配

信号	FPGA引脚	DE2板上器件
din	PIN_N25	SW0
en	PIN_N26	SW1
dout	PIN_AE22	LEDG0

## (2) 三态与门

利用前面已完成的与门和三态门组合成一个三态与门。与前面两个例子不同的是，在这里不是采用文本编辑器完成设计输入，而是采用图形编辑器。QuartusII 的原理图输入设



计法可以与传统的数字电路设计法接轨，即把传统方法得到的设计电路的原理图，用 EDA 平台完成设计电路的输入、仿真验证和综合，最后编程下载到可编程逻辑器件 (FPGA/CPLD) 或专用集成电路 (ASIC) 中。在 EDA 设计中,不必进行传统电路设计过程的布局布线、绘制印刷电路板、电路焊接、电路加电测试等，从而提高了设计效率，降低了设计成本，减轻了设计者的劳动强度。然而，原理图输入设计法的优点不仅如此，它还可以方便地实现数字系统的层次化设计，这是传统设计方法无法比拟的。层次化设计也称为"自底向上"的设计，即将一个大的设计项目分解为若干个子项目或若干个层次来完成。先从底层的电路设计开始,然后从高层次的设计中逐级调用低层次的设计结果，直至顶层系统电路的实现。对于每个层次的设计结果，都经过严格的仿真验证，以尽量减少系统设计中的错误。每个层次的设计均可以用原理图输入法实现，也可以用其他方法(如HDL文本输入法)实现，这种方法称为"混合设计输入法"。层次化设计为大型系统设计及SOC或SOPC的设计提供了方便、直观的设计途径。

操作步骤如下：

第1步：首先将上述两个Verilog HDL文件生成成为符号（Symbol），以供后续步骤使用。在图4-13所示的项目导向（Project Navigator）窗口中，右击and2gate.vhd，选择Create Symbol Files for Current File命令，即生成了and2gate符号。用同样的方法生成trigate符号。

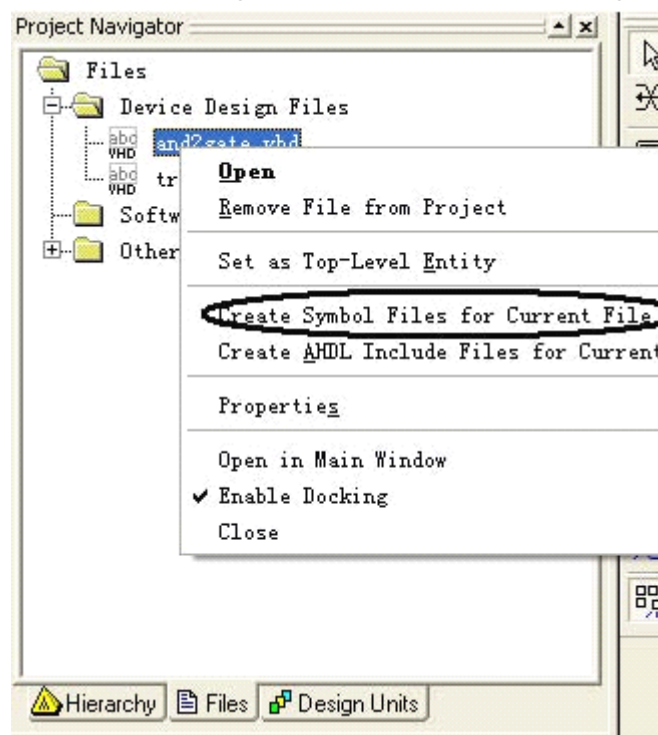


图4-13 项目导向（Project Navigator）窗口

第2步：新建一个图形文件。选择File->New命令，选择“Diagram/Schematic File”，点击OK按钮完成。将该图形文件另存为tri\_and\_gate.bdf。图形编辑窗口如图4-14所示，窗口左边是图形编辑工具条。

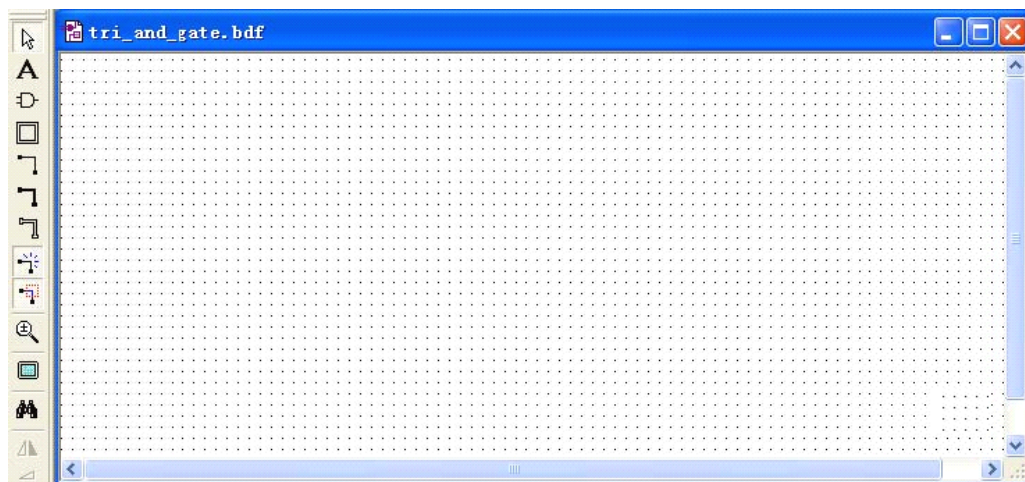


图4-14 图形编辑窗口

第3步：在图形编辑窗口的空白处双击，打开符号库，如图4-15所示。展开Project项，可以看到有两个之前生成的符号分别是and2gate和trigate。选择and2gate，单击OK按钮，该符号就会出现在图形编辑窗口，单击左键即在窗口内放置该符号。用同样的方法放置trigate符号。

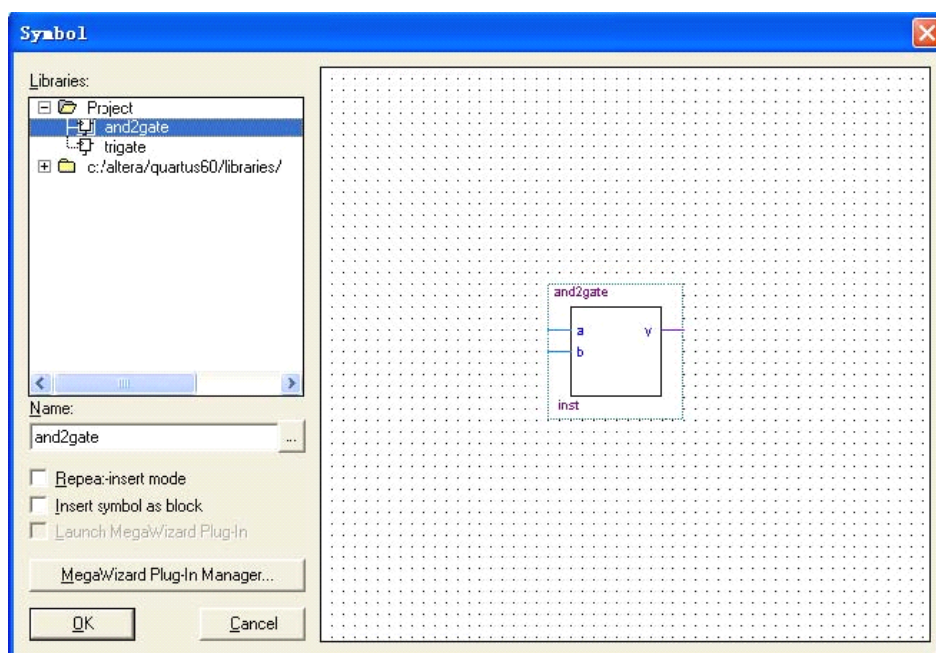



图4-15 符号库

第4步：再次打开符号库，在name输入栏中输入“input”，符号库自动在库中找到输入（input）符号，并选中“Repeat-insert mode”点击OK按钮，如图4-16所示。可反复在编辑窗口中放入输入符号，直到单击右键取消放置为止。由于输入信号一共有3个，所以需要放入3个输入符号，并将3个输入符号命名为dina、dinb和en。用同样的方法放置1个输出（output）符号，并命名为dout。再选择工具栏中的  按钮，将各符号连接起来，结果如图 4 -17所示。

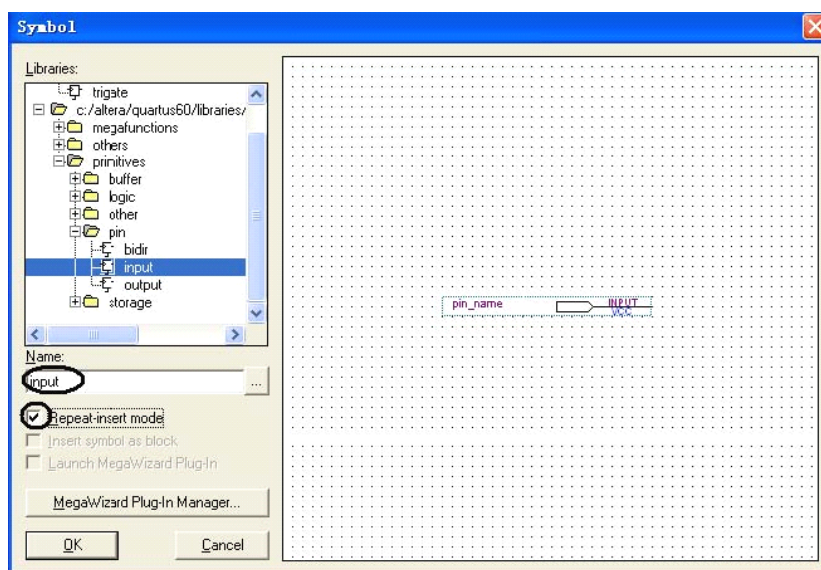


图4-16 input 输入端符号

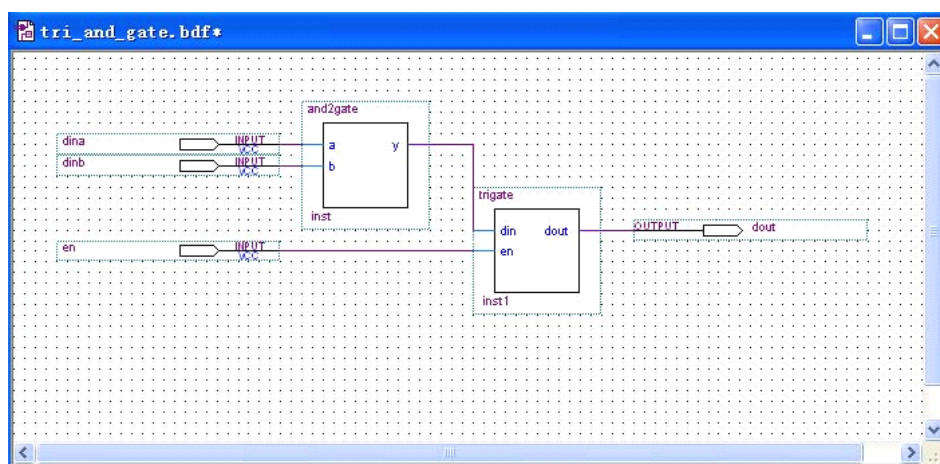


图4-17 三态与门原理图

第5步：保存图形文件，并将tri\_and\_gate.bdf设置为顶层实体。再次编译项目文件，并进行功能仿真，仿真结果如图4-18所示。

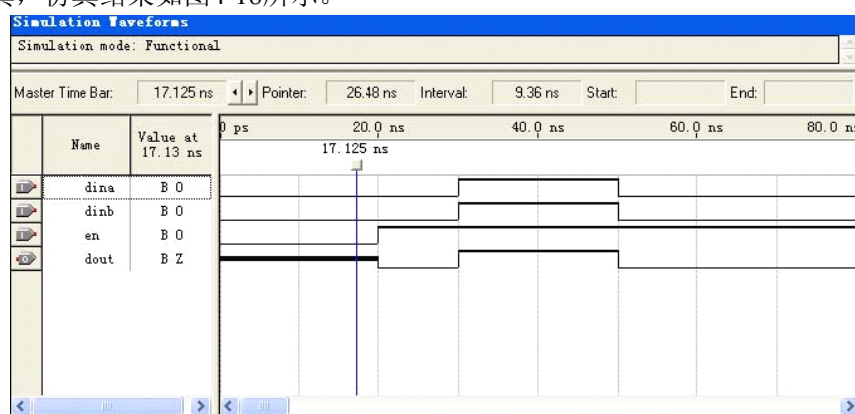


图4-18 仿真结果

第6步：按照表4-3分配引脚，重新编译并下载验证。

表4-3 引脚分配

信号	FPGA引脚	DE2上的器件
dina	PIN_N25	SW0
dinb	PIN_N26	SW1
en	PIN_P25	SW2
dout	PIN_AE22	LEDG0

3.在 Quartus II 中调用 ModelSim 进行仿真

(1) Quartus II 的相关设置

1)在 Quartus II 中指明仿真工具及路径

在 Quartus II 中执行 Tools\Options,打开 EDA Tool Options 选项卡。我们使用 ModelSim 进行仿真，所以在 ModelSim 对应的 Location of Executable 中选择 ModelSim 的安装路径 E:\Modelsim\win32（视具体情况而定）,如图 4-19 所示。

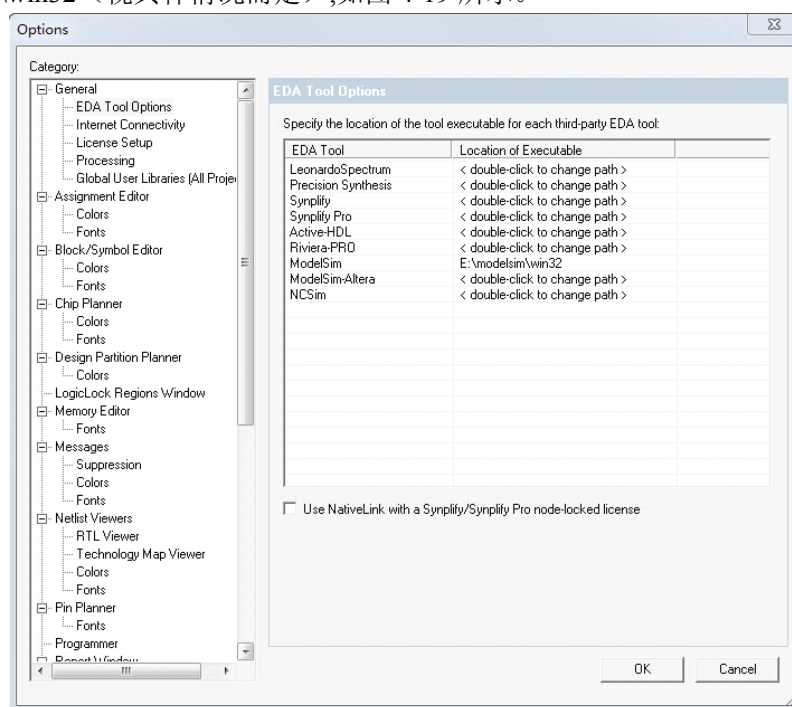


图 4-19 ModelSim 的安装路径

2)在 Quartus II 中编译所需的元器件库

①在 ModelSim 安装路径 E:\Modelsim 中新建文件夹 modelsimSE\_lib,用于存放编译的文件。

②在 Quartus II 中执行 Tools\Launch EDA Simulation Library Compiler,在打开的界面中 Executable location 一项选择 ModelSim 的安装路径 E:\Modelsim\win32，在 Available families 中根据自己需要选择可能用的器件系列，如 cyclone、stratix 等；在这里可以多选择一些，以备后用，并且一次编译后，以后就不用再编译了。Library Language 一项勾选 Verilog; Output directory 选择 E:\Modelsim\modelsimSE\_lib;设置好的界面如图 4-20 所示。

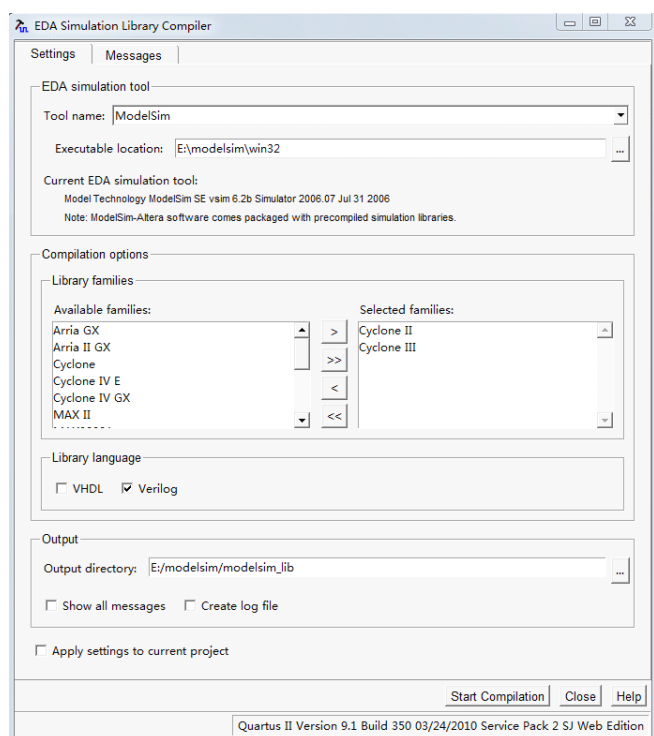


图 4-20 元器件库编译设置窗口

③在图 4-20 所示界面中点击 **Start Compilation**,开始编译。

④编译完成后如图 4-21 所示，依次点击确定、close。

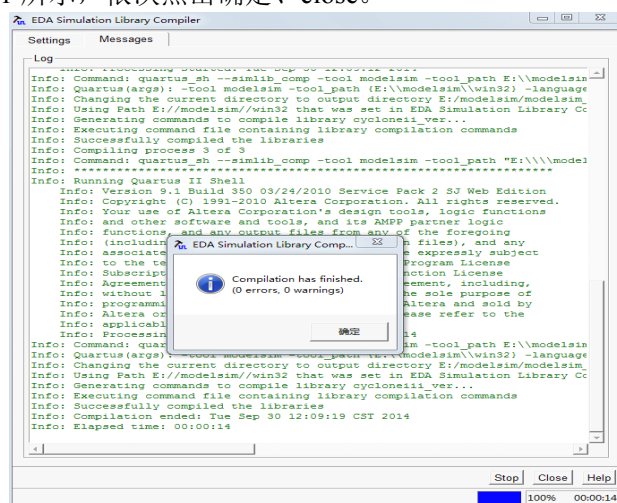


图 4-21 仿真库编译状态

(2)在 Quartus II 中建立工程及仿真

1)在 Quartus II 中建立工程 **counter**,并指明仿真工具为 **Modelsim(Verilog)**,这一步可以在 **New Project Wizard** 中指定,如图 4-22 所示;也可以先建立好工程在 **Assignments/Setting/EDA Tool Settings** 中的 **simulation** 选项中设置,如图 4-23 所示。



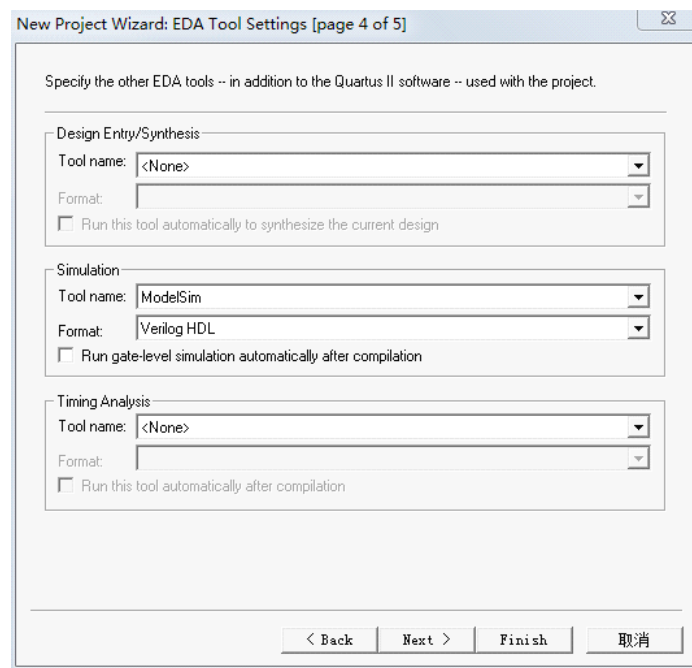


图 4-22 在 New Project Wizard 中指定仿真工具

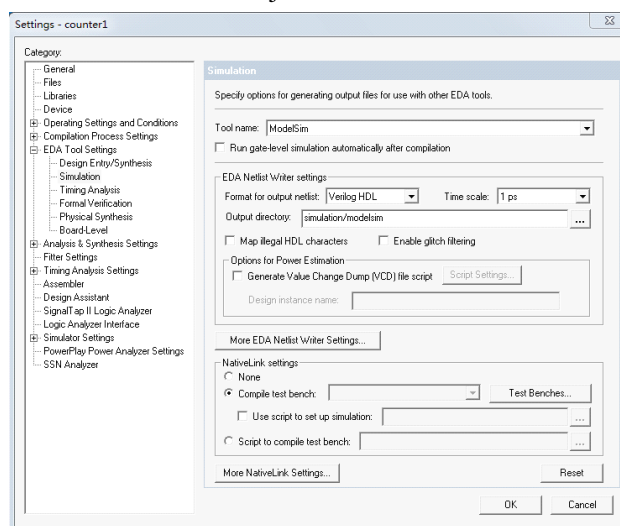


图 4-23 在 EDA Tool Settings 中指定仿真工具

2)在 Quartus II 工程中输入顶层文件 counter.v 代码

```
`timescale 1ns/1ps //时间单位/时间精度
module counter(
    input        clk,
    input        rst,
    output reg[3:0] cnt, //时间计数器
    output        div_2, //2 分频
    output        div_4, //4 分频
    output        div_8  //8 分频
);
```

```
always @ (posedge clk or posedge rst)
begin
```

```

if(rst) begin
    cnt<=4'h0;
    end
else begin
    cnt<=cnt+1'b1;
    end
end
assign div_2=cnt[0];
assign div_4=cnt[1];
assign div_8=cnt[2];
endmodule

```

3) 在 Quartus II 工程中输入 testbench 文件 counter\_test.v 代码

```

`timescale 1ns/1ps                                //仿真时间单位/时间精度
module counter_test();
reg      clk;
reg      rst;
wire[3:0] cnt;
wire      div_2;
wire      div_4;
wire      div_8;

parameter clk_cycle=10;                          //20M 时钟
parameter clk_hcycle=5;

counter dut(                                       //实例化待测试模块
    .clk(clk),
    .rst(rst),
    .cnt(cnt),
    .div_2(div_2),
    .div_4(div_4),
    .div_8(div_8)
);

initial begin
    clk=1'b1;
    end
always #clk_hcycle  clk=~clk;                    //产生时钟信号

initial begin                                     //产生复位信号
    rst=1'b1;
    #10                                           //延时 10ns 即 10 个时间单位后,rst 从 1 变为
0
    rst=1'b0;
    end

```

initial begin

```
$monitor($time,,clk,,rst,,cnt,,div_2,,div_4,,div_8);
```

```
#10000 $stop;
```

end

endmodule

4)在 Quartus II 中添加 testbench 文件

①打开 Assignments\Setting\EDA Tool Setting,单击 simulation 选项,在打开的界面中选中 compile testbench,如图 4-24 所示。

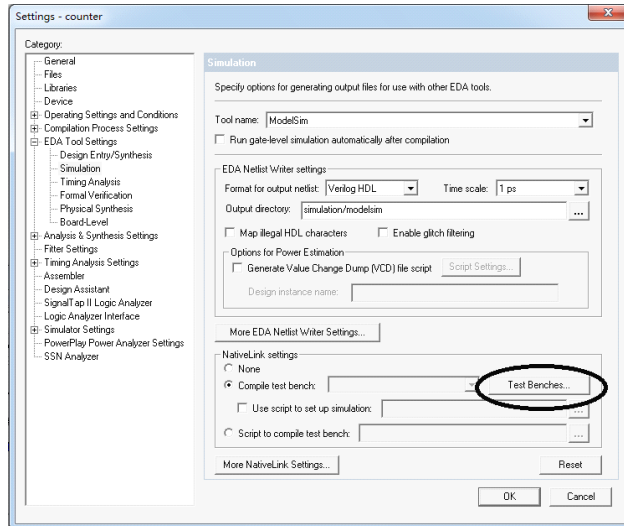


图 4-24 添加 testbench 文件窗口

②在图 4-24 界面中,单击 Test Benches...,弹出图 4-25 所示界面。

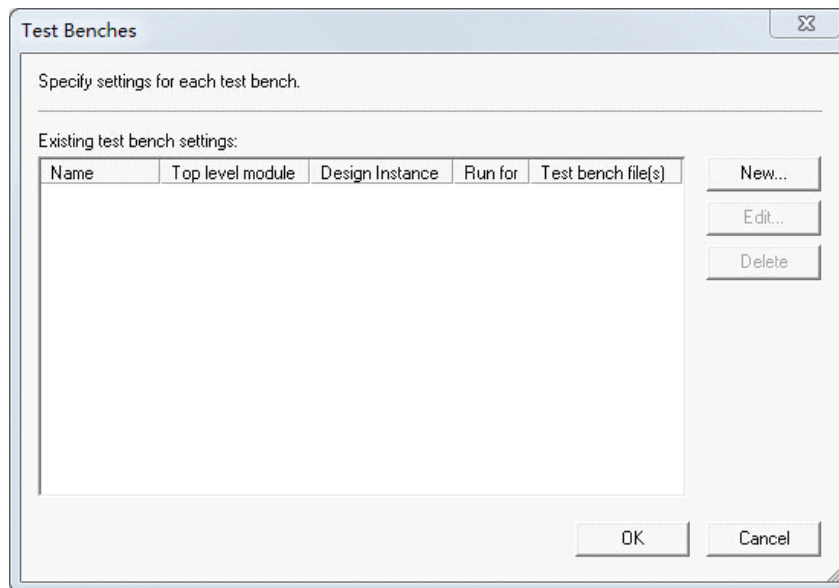


图 4-25 指定 Test Bench 窗口

③在图 4-25 界面中，单击 New,在弹出的界面中，按图 4-26 填入相关内容。

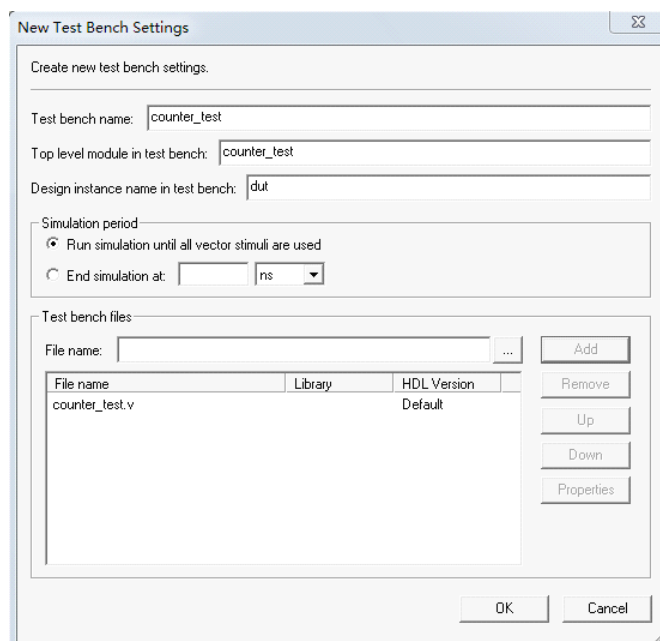


图 4-26 Test Bench 设置窗口

④后面几步不做更改，直接按 ok 即可

5)在 Quartus II 中全编译工程，这样在工作目录下会生成 simulation 文件夹，内部 modelSim 文件夹中有三个文件分别是 counter.vo（布局布线后的仿真模型文件），counter\_modelsim.xrf（交叉引用文件），counter\_v.sdo（标准延时输出文件）。

6)前仿真：在 Quartus II 中执行 Tools\Run EDA simulation Tool\EDA RTL Simulation Modelsim 会自动启动并完成前仿真。前仿真结果如图 4-27 所示。

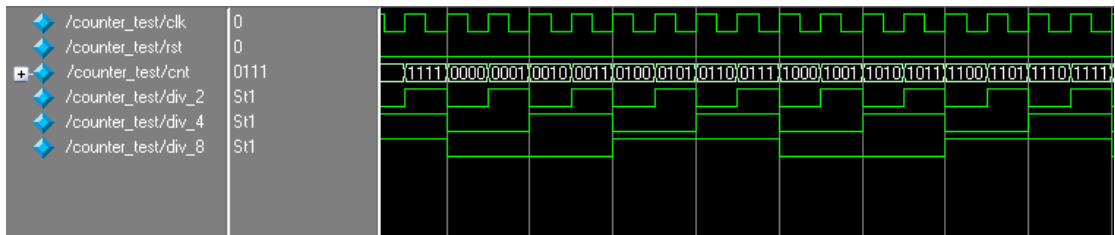


图 4-27 前仿真结果

7)后仿真：在 Quartus II 中执行 Tools\Run EDA simulation Tool\EDA Gate Level Simulation 会自动启动并完成仿真。后仿真结果如图 4-28 所示。

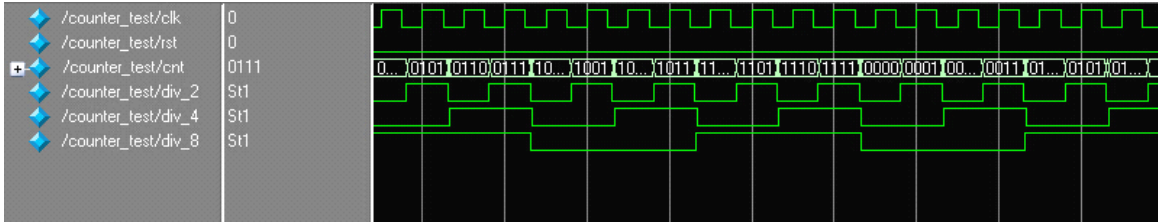


图 4-28 后仿真结果

### 三、实验报告与要求

- (1)完成实验内容中要求的各项任务。
- (2)记录每个步骤的软件仿真波形和硬件验证结果。并总结调试过程中出现的问题和解决方案。

## 实验二：简单的组合逻辑电路设计

### 一、实验目的

- (1) 掌握组合逻辑电路的设计方法。
- (2) 掌握同一项目下对指定文件的编译方法。
- (3) 加深PLD设计的过程，并比较原理图输入法和文本输入法的优劣。

### 二、实验内容及步骤

#### 1. 2选1的数据选择器

具体步骤：

第1步：新建一个Quartus项目。

第2步：在Quartus项目中新建一个Verilog HDL文件，并命名为mux\_2to1.v，实现2选1的电路功能，其真值表和电路如图4-29所示。即当s=1时，输出m=y；当s=0时，输出m=x。

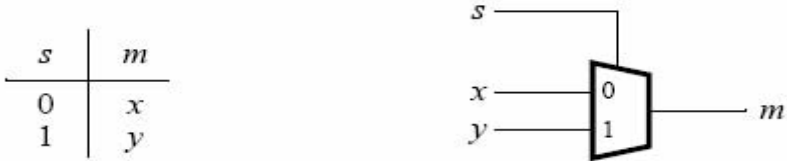


图4-29 2选1真值表和电路图



代码一： Verilog HDL程序代码如下。

```
module mux_2to1 (m,s,x,y);
input x,y,s;
output m;
reg m;
always @(s or x or y)
begin
if(s==1'b0)
m=x;
else
m=y;
end
endmodule
```

第3步：语法检查通过后，进行引脚分配，分配表如表4-4所示。然后再编译，下载验证。

表4-4 引脚分配

信号	FPGA引脚	DE2上的器件
s	PIN_N25	SW0
x	PIN_N26	SW1
y	PIN_P25	SW2
m	PIN_AE22	LEDG0

## 2. 8 位宽 2 选 1 的数据选择器

在完成2选1数据选择器之后，将信号x和y的位宽由1位扩展为8位。更改后的电路图如图4-30所述。

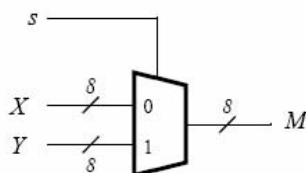


图4-30 8位宽2选1的数据选择器电路图

实验步骤如下：

第1步：在代码一中，端口说明部分更改为：

```
module mux_2to1_8bit (m,s,x,y);
input[7:0] x,y;
input s;
```

而逻辑功能描述部分代码不变，代码修改后另存为mux\_2to1\_8bit.v。

第2步：接着把mux\_2to1\_8bit.v设定为项目的顶层设计文件。实现方法如图4-31所示，在项目浏览器（Project Navigator）中选择文件（Files）页，选中mux\_2to1\_8bit.v，单击右键，选择“Set as Top-Level Entity”命令即可。

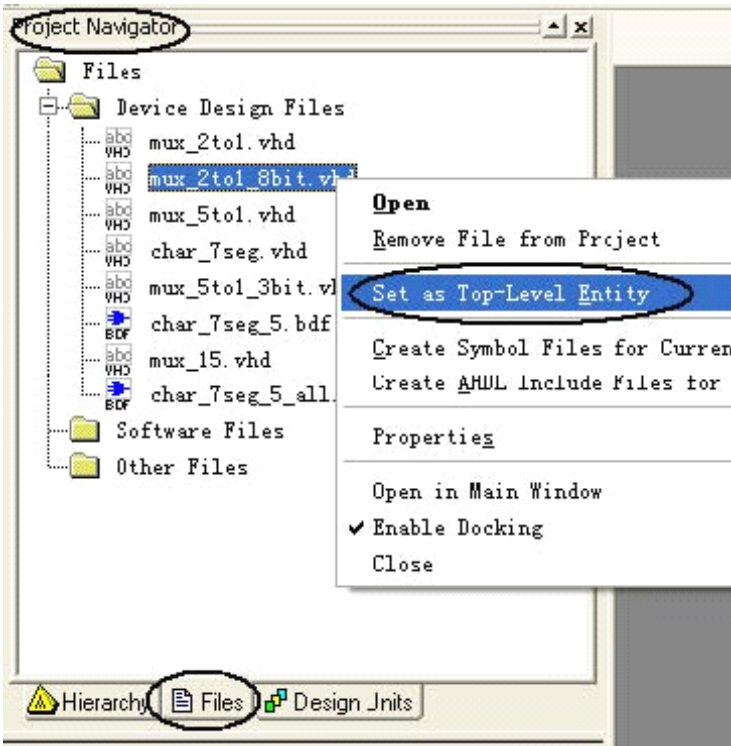


图4-31 顶层文件设置图

第3步：语法检查，检查通过后再进行引脚分配。

引脚数也由原来的4个增加到25个，引脚分配更改如表4-5所示。分配完后重新编译项目文件，并下载验证。

表4-5 引脚分配

信号	FPGA引脚	DE2上的器件
s	PIN_N25	SW0
X[7..0]	查附录表	SW8~SW1
Y[7..0]	同上	SW16~SW9
M[7..0]	同上	LEDG7~LEDG0

3. 4 选 1 的数据选择器

在完成2选1电路之后，将电路扩展为4选1数据选择器，其真值表及电路图如图4-32所示。

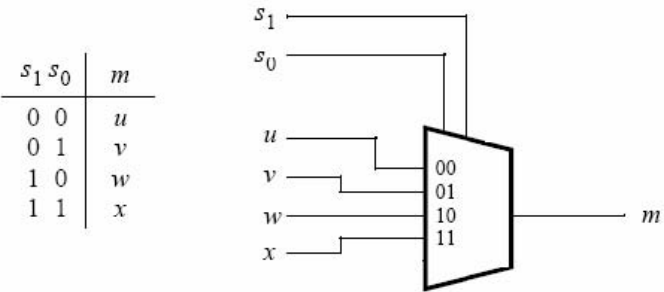


图4-32 4选1数据选择器真值表和电路图

代码修改如下：  
module mux\_4to1 (m,s,u,v,w,x);

```

input u,v,w,x;
input[1:0] s;
output m;
reg m;
always @(s or u or v or w or x)
begin
    case(s)
        2'b00: m=u;
        2'b01: m=v;
        2'b10: m=w;
        default: m=x;
    endcase
end
endmodule

```

文件另存为mux\_4to1.v。

接着将mux\_4to1.v设定为项目的顶层设计文件，再进行语法检查和引脚分配。

引脚分配表如表4-6所示，具体的FPGA引脚可通过查找附录获取。分配完后重新编译项目文件，并下载验证。

表4-6 引脚分配表

信号	DE2上的器件
S[1..0]	SW1~SW0
u	SW2
v	SW3
w	SW4
x	SW5
m	LEDG0

#### 4. 实现 3 位宽的 4 选 1 数据选择器。

电路如图4-33所示。代码完成后，另存为mux\_4to1\_3bit.vhd。

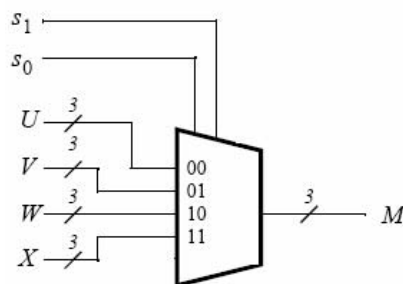


图4-33 3位宽的4选1数据选择器电路图

## 四、实验报告与要求

- (1)完成实验内容中要求的各项任务。
- (2)记录编写的代码或设计的原理图。
- (3)记录每个步骤的软件仿真波形和硬件验证结果。并总结调试过程中出现的问题和解决方

案。

### 实验三：七段数码管显示

#### 一、实验目的

- (1) 掌握七段数码管显示电路原理。
- (2) 掌握数码管显示简单字符、数字。
- (3) 掌握数码管和数字选择器构成组合逻辑电路。
- (4) 掌握元件例化语句的使用

#### 二、实验内容及步骤

##### 1. 显示简单字符

七段数码管显示电路如图4-34所示：

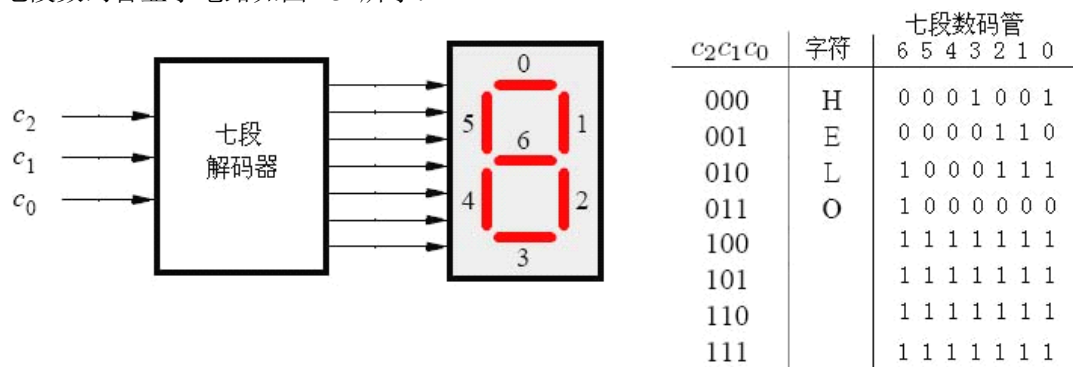


图4-34 电路图和真值表

图4-34中包含一个七段解码器模块， $c_2 \sim c_0$ 是解码器的3个输入，当输入值不同时，输出的字符也不同。如图4-34所示，当输入值为100~111时，输出空格，即数码管全暗。七段数码管的不同段位用数字0~6表示，注意七段数码管是共阳极的，即各管段输入低电平时，数码管亮；否则数码管暗。

具体实验步骤如下：

第1步：新建一个Quartus项目。

第2步：新建一个Verilog HDL文件，实现上述七段解码器。具体代码如下：

```
module char_7seg (hex,c);
input[2:0] c;
output[6:0] hex;
reg[6:0] hex;
always @(c)
begin
    case(c)
        3'b000: hex=7'b0001001;
        3'b001: hex=7'b0000110;
        3'b010: hex=7'b1000111;
        3'b011: hex=7'b1000000;
        default: hex=7'b1111111;
    endcase
end
endmodule
```

保存Verilog HDL文件，并命名为char\_7seg.v。  
 第3步：语法检查，通过后，进行引脚分配，如表4-7所示。

表4-7 引脚分配

信号	DE2上的器件
C[2..0]	SW2~SW0
hex[6..0]	HEX0[6..0]

第4步：编译项目，完成后下载到FPGA中，并验证其功能。

2. 显示0~9数字

在完成简单字符显示电路之后，设计一个用于显示0~9数字的七段数码管电路。电路如图4-35所示，c3~c0是七段数码器的输入，当输入0000~1001时，则输出0~9，真值表如图4-35所示；当输入1010~1111时，输出空格。

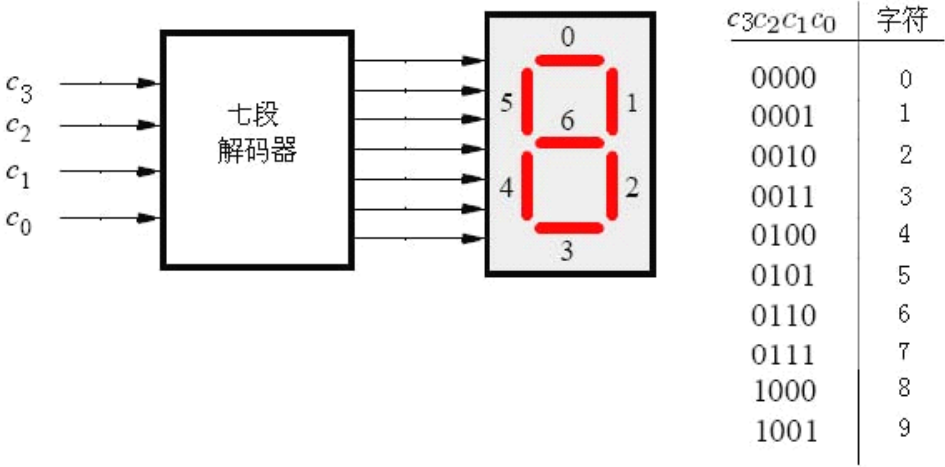


图4-35 电路图和真值表

对上述代码进行相应的修改，并将文件另存为num\_7seg.v。  
 接着将num\_7seg.v设定为项目的顶层设计文件，再进行语法检查和引脚分配。  
 引脚分配如表4-8所示，具体的FPGA引脚可通过查找附录获取。分配完后重新编译项目文件，并下载验证。

表4-8 引脚分配

信号	DE2上的器件
C[3..0]	SW3~SW0
hex[6..0]	HEX0[6..0]

3. 循环显示4个字符

循环显示4个字符的电路图如图4-36所示。

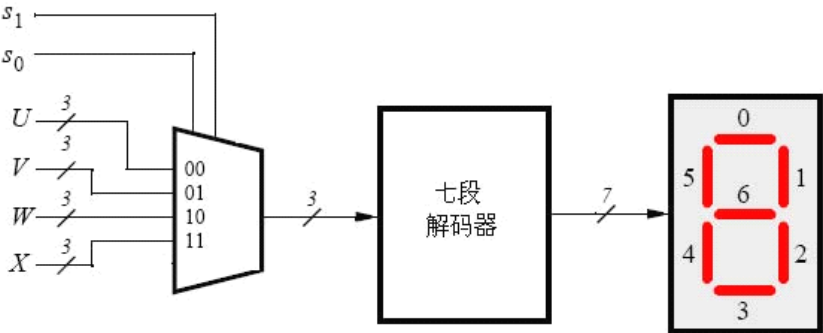




图4-36 电路图

电路的工作原理是，输入端U、V、W和X的输入值分别是000、001、010和011，通过s1和s0选择四个输入端其中一个作为七段解码器的输入值，从而显示H、L、E和O任一字符。

这个实验的实现方法有两个，一个是采用图形编辑的方法实现，与本节**实验一**中三态与门的实现方法类似。不过事先需要将**实验二**中的mux\_4to1\_3bit.v添加到本项目中，并为其创建一个符号；同时，为char\_7seg.v创建一个符号。另一个方法是采用Verilog HDL文本输入，但也需要事先将**实验二**中的mux\_4to1\_3bit.v添加到本项目中。

方法一：

图形文件的最终效果图如图4-37所示。图中的inst1和inst分别是mux\_4to1\_3bit.v和char\_7seg.v所生成的符号。

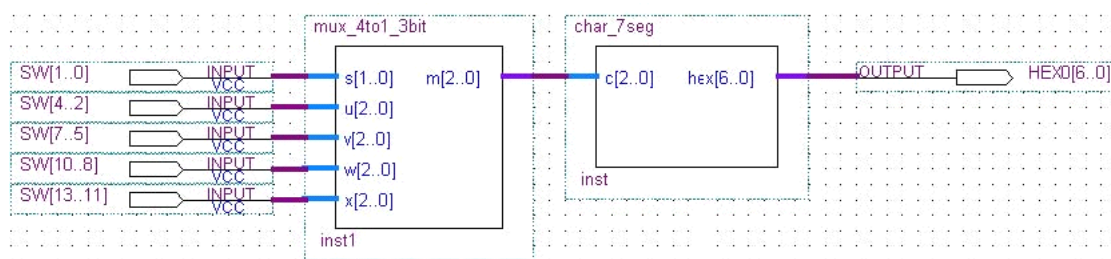


图4-37 循环显示4个字符的原理图

值得注意的是，s[1..0]与输入引脚SW[1..0]连接，其他输入端类似，而hex[6..0]与输出引脚HEX0[6..0]连接。这里的输入和输出引脚全部采用DE2的引脚名称（注意区分大小写），目的是可以省去手动分配引脚的工作。

第1步：首先将图形文件另存为char\_4to1\_7seg.bdf，并将该文件设定为项目的顶层设计文件，再进行语法检查。

第2步：检查通过后，打开引脚规划（Pin Planner）窗口，命令是Assignments->Pins。

第3步：选择Assignments->Import Assignments..命令，导入DE2\_pin\_assignments.csv文件，该文件是DE2板上所有引脚的分配。导入之后，可以发现所有输入和输出引脚已经自动完成引脚分配，原因是我们把引脚的名称设定为DE2默认的引脚名称。

第4步：编译下载。

方法二：

通过 Verilog HDL 代码实现，这里需要用到元件实例化语句(元件调用),调用 mux\_4to1\_3bit模块和char\_7seg模块 代码如下。

```
moulde char_4to1_7segv (hex0,s,U,V,W,X);
input [2:0] U,V,W,X;
input [1:0] s;
output [6:0] hex0;
wire [2:0] m;
mux_4to1 A1 (m,s,U,V,W,X);
char_7seg A2 (hex0,m);.....
endmoule
```

将文件另存为char\_4to1\_7segv.v，并将其设定为项目的顶层设计文件。再进行语法检查、引脚分配和编译下载。

### 三、实验报告与要求

- (1)完成实验内容中要求的各项任务。
- (2)记录编写的代码或设计的原理图。

(3)记录每个实验内容的硬件验证结果，并总结调试过程中出现的问题和解决方案。

## 实验四：BCD 码显示及运算

### 一、实验目的

- (1) 掌握根据电路功能表设计电路的Verilog HDL代码。
- (2) 掌握用图形编辑方法和文本编辑方法实现层次化设计。
- (3) 掌握设计BCD加法电路

### 二、实验内容及步骤

#### 1. 二进制码到 BCD 码的转换

二进制码与BCD码之间的转换关系如表4-9所示。

表4-9 二进制码与BCD码转换

Binary Value	Decimal digits	
0000	0	0
0001	0	1
0010	0	2
...	...	
1001	0	9
1010	1	0
1011	1	1
1100	1	2
1101	1	3
1110	1	4
1111	1	5

表4-9中将4位二进制输入 $V=v_3v_2v_1v_0$ 转换成2位十进制 $D=d_1d_0$ ，实现办法是用SW[3..0]作为二进制输入，而用HEX1和HEX0作为十进制输出的显示。从表4-9中可以看出，当 $V \leq 9$ 时， $d_1=0$ 、 $d_0=V$ ；反之， $d_1=1$ 、 $d_0=V-10$ 。

实验步骤如下：

第1步：新建一个Quartus项目。

第2步：建立一个Verilog HDL文件，根据上述工作原理编写代码以实现所要求的电路，文件另存为bin\_bcd.v。

第3步：完成代码转换之后，需要将BCD码在数码管上显示，所以需要在项目中添加本节实验三中完成的num\_7seg.v文件。

第4步：采用图形编辑方法或元件调用方法都可以完成最终的电路功能。

第5步：编译并下载验证。

#### 2. 1位BCD加法器

电路原理是输入两个BCD码A和B以及1位进位输入cin，输出是BCD码的和sum以及1位进位输出cout。例如当 $A=1001$ （9）、 $B=1001$ （9）、 $cin=1$ 时， $cout=1$ ， $sum=1001$ （9）。电路的输出最大值也就是19。

1位BCD加法器可以利用两个二进制加法器实现，其电路如图4-38所示。在Verilog HDL中，二进制加法可以直接用 $A+B$ 实现。

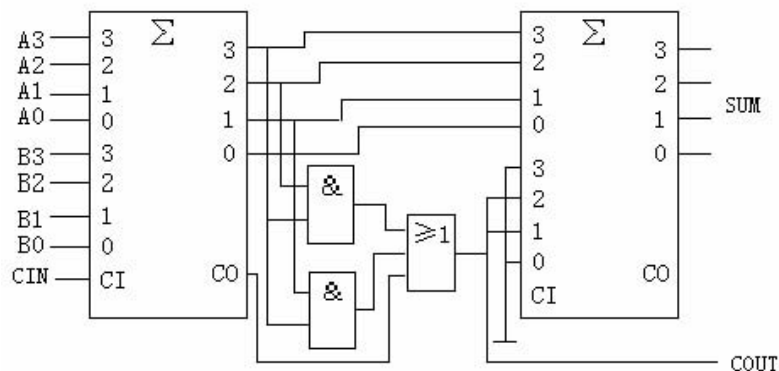


图4-38 两个二进制加法器电路图

程序的部分代码如下：

```

wire [3:0] m;
wire CO,c;
assign {CO,m}=A+B+CIN;
assign c=CO | (m[3]&m[2]) | (m[3]&m[1]);
assign COUT=c;
assign SUM=m[3:0]+{1'b0,c,c,1'b0};

```

这个程序完全是按照图4-38得出的，文件另存为bcd\_add\_1bit.v。由于需要将结果在数码管上显示，所以需要在项目中添加本节实验三中完成的num\_7seg.v文件。

验证电路时可以用SW[0]作为CIN输入端，SW[4..1]、SW[8..5]分别作为A和B的输入端，HEX0作为SUM的输出端，LEDG[0]作为COUT的输出端。

另外一种方法：设计一个BCD码加法器（两个 4 bit）

```

module add1(ina,inb,cin,cout,sum)
input [3:0] ina,inb;
input cin;
output [1:0] cout;
output [3:0] sum;
assign {cout,sum}=((ina+inb+cin)>9)? (ina+inb+cin+6): (ina+inb+cin);
endmodule

```

解释上述代码的BCD码加法器计算原理。

### 3. 2 位 BCD 加法器

从1位BCD加法器扩展为2位BCD加法器，可以采用图形编辑器和Verilog HDL文本输入两种方法实现。输入两个2位BCD码 $A_1A_0$ 和 $B_1B_0$ 以及1位进位输入cin，输出2位BCD码之和 $S_1S_0$ 和1位进位输出cout。验证电路时可用SW[8..1]表示 $A_1A_0$ ，SW[16..9]表示 $B_1B_0$ ，SW[0]表示cin；HEX1和HEX0表示 $S_1S_0$ ，LEDG[0]表示cout。

方法1：采用图形编辑器的方法，最终效果图如图4-39所示。

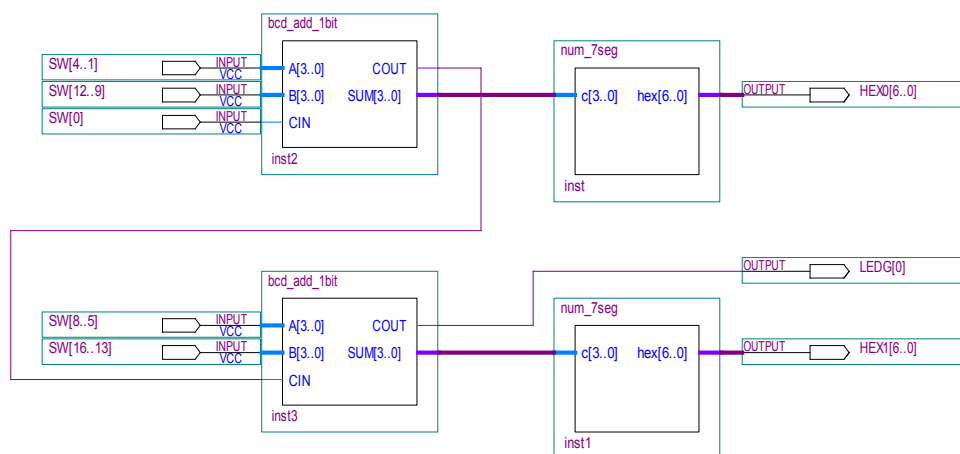


图4-39 2位BCD加法器电路图

方法2：采用1位BCD加法器的设计思路重新设计一个2位BCD加法器，以下是实现2位BCD加法器的伪代码，可作为编程的思路。

```

1 T0 = A0 + B0
2 if(T0 > 9) then
3 Z0 = 10;
4 c1 = 1;
5 else
6 Z0 = 0;
7 c1 = 0;
8 end if
9 S0 = T0 - Z0
10 T1 = A1 + B1 + c1
11 if (T1 > 9) then
12 Z1 = 10;
13 c2 = 1;
14 else
15 Z1 = 0;
16 c2 = 0;
17 end if
18 S1 = T1 - Z1
19 2 = c2

```

### 三、实验报告与要求

- (1) 完成实验内容中要求的各项任务。
- (2) 记录编写的代码或设计的原理图以及软件仿真图。
- (3) 解释步骤2中1位BCD加法器第二种方法的BCD码加法器计算原理。
- (4) 记录每个实验内容的硬件验证结果，并总结调试过程中出现的问题和解决方案。

## 实验五 分频器

### 一、实验目的

- (1) 掌握分频电路设计方法。
- (2) 掌握嵌入式锁相环宏功能模块的使用方法

### 二、实验内容及步骤

CPLD/FPGA与单片机相比，一个非常明显的优势就在于它的高速性。但是因为很多外围器件的驱动需要低频的时钟（若时钟频率太高，则键盘扫描容易出错，七段数码管会闪烁和不稳定等），所以常常需要用到分频电路，其目的是用一个时钟频率生成另一个时钟频率。常用的分频电路分为偶数倍分频和奇数倍分频两种。

#### 1. 偶数倍分频

偶数倍分频的原理十分简单，例如8分频率电路设计

第1步：新建一个Verilog HDL文件，并另存为clk\_div8.v。

第2步：Verilog HDL代码如下：

```
module clk_div8 (clkout,clkkin);
input clkkin;
output clkout;
reg clkout;
reg[1:0] counter;
always @(posedge clkkin)
begin
    if( counter[1:0] == 2'd3)
        begin //每计到4个（0~3）上升沿，输出信号翻转一次
            counter <= 0;
            clkout <= ~ clkout;
        end
    else
        counter[1:0] <= counter[1:0] + 1'b1;
    end
endmodule
```

第3步：语法检查通过后，再进行功能仿真。仿真结果应该如图4-40所示。

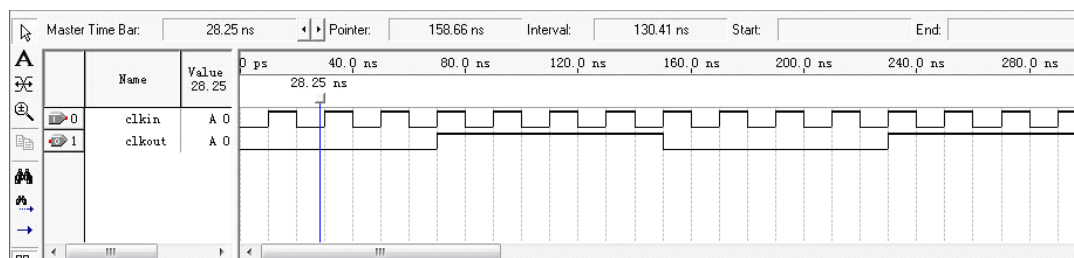


图 4-40 仿真结果

上面给出的是一个8分频率电路，其他倍频数的分频电路可以通过修改Counter的上限值N得到。一般的计算规则是：对一个2x 分频的电路来说，counter上限值是N=x-1(从0计到x-1恰好为x次，每x个上升沿翻转一次就实现了2x分频)。如果希望产生一个下降沿翻转的分频电路，只需将rising\_edge( Clkin)改成falling\_edge(Clkin)即可。

第4步：将DE2平台上的50MHz时钟分频为1Hz频率的时钟，将文件另存为clk\_1\_gen.v。





Plug-In Manager[page1]"对话框的第1页面。

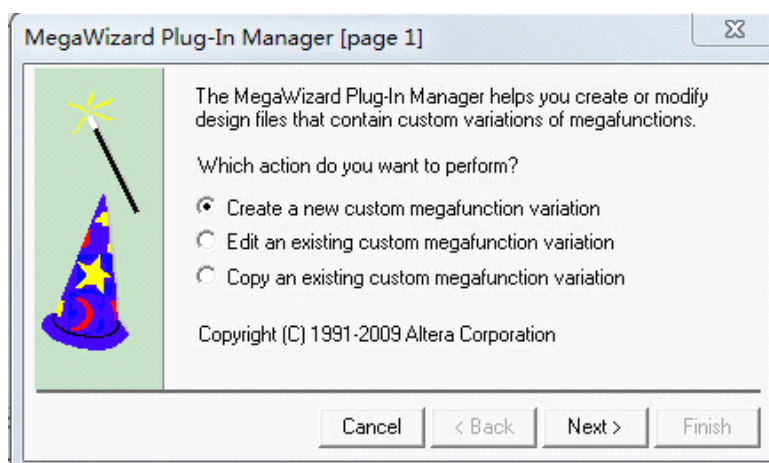


图 4-42 "MegaWizard Plug-In Manager[page1]" 对话框

在对话框中,选中 "Create a new custom megafunction variation" 项,创建一个新的强函数定制。在此对话框中还可以选择编辑一个现有的强函数定制或复制一个现有的强函数定制。用鼠标左键单击 MegaWizard 插件管理器对话框下方的"Next"按钮,弹出如图4-43 所示的"MegaWizard Plug-In Manager[page 2a]"对话框。在此对话框中,用鼠标左键选中强函数列表中的"I/O"选项下的"ALTPLL"项,表示将创建一个新的嵌入式锁相环设计项目。在对话框中的 "Which device family will you be using?" 栏中,选择编程下载目标芯片的类型,例如 "Cyclone II"。在对话框的 "Which type of output file do you want to create?" 栏下选择生成设计文件的类型,有 AHDL、VHDL和 Verilog HDL 三种HDL文件类型可选。例如,选择 "Verilog HDL",则可生成嵌入式锁相环的Verilog HDL设计文件。在对话框的 "What name do you want for the output file?" 栏中填入设计文件的路径和文件名,例如 "D:\mypll\mypll.v"。

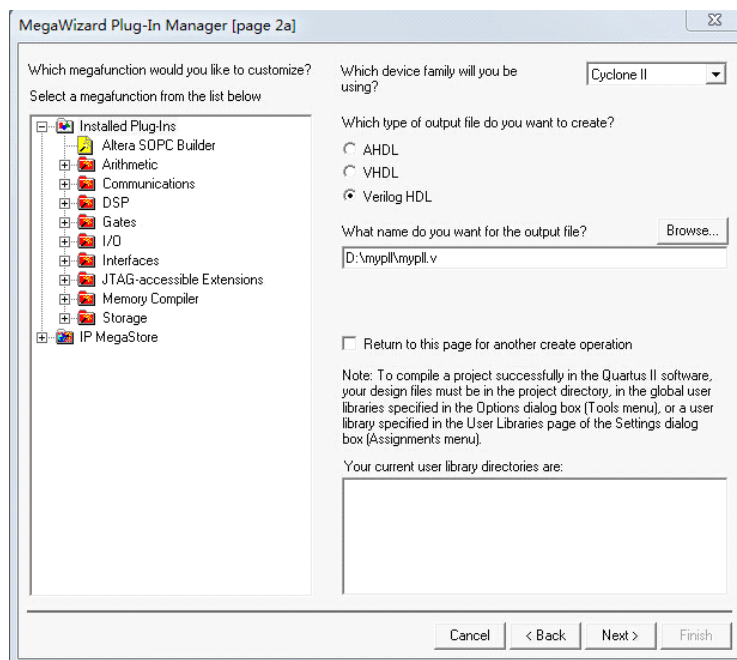


图4-43 "MegaWizard Plug-In Manager[page 2a]"对话框

完成 "MegaWizard Plug-In Manager[page 2a]"对话框的设置后,用鼠标左键单击对话框下方的"Next"按钮,弹出如图4-44所示对话框。在对话框的左边显示了嵌入式锁相环的元件

图, 元件包括外部时钟输入端 `inclk0`、复位输入端 `areset`、倍频(或分频)输出端 `c0` 和相位锁定输出端 `locked`。在对话框的 "What is the frequency of the `inclk0` input?" 栏中填入输入时钟的频率, 此频率需要根据选择的目标芯片来决定, 不能过低也不能过高, 对于 CycloneII 系列芯片, 输入时钟的频率可选择 50MHz。对话框其他栏目中的内容可以选择默认。

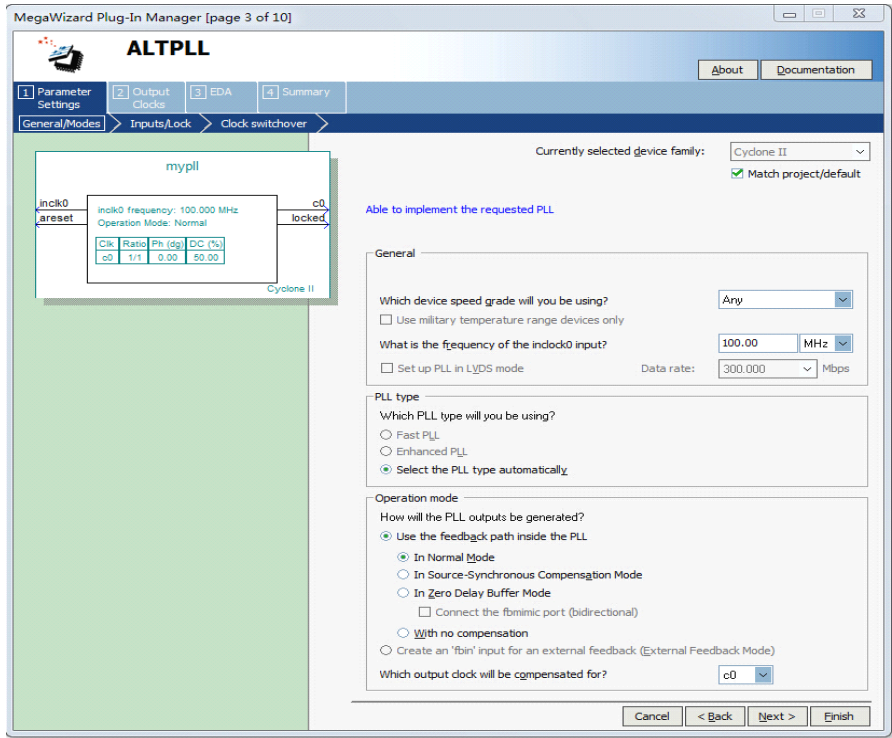


图 4-44 "MegaWizard Plug-In Manager[page 3]"对话框

完成图4-44所示的对话框的设置后, 用鼠标左键单击对话框下方的 "Next" 按钮, 弹出如图4-45所示对话框。此对话框主要用于添加其他控制输入端, 如添加相位/频率选择控制端 `pfdena`, 本例设计不添加该输入端。

用鼠标左键单击对话框下方的 "Next" 按钮, 弹出的对话框主要用于添加第2个时钟输入端 `inclk1`, 本例设计对该对话框的设置保持默认, 不添加该输出端。

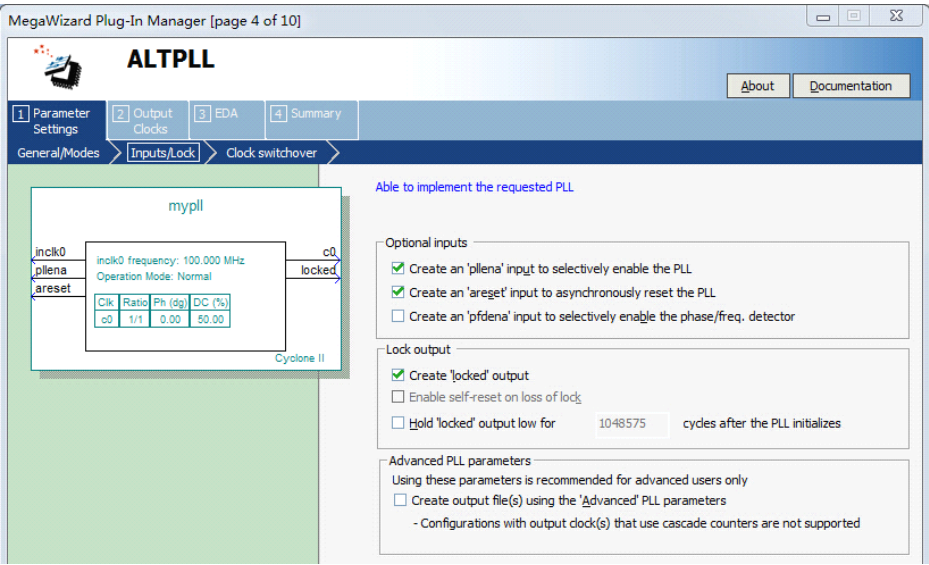


图4-45 "MegaWizard Plug-In Manager[page 4]"对话框

用鼠标左键单击"Next"按钮,弹出如图4-46所示对话框,此对话框主要用于设置输出时钟c0的相关参数,如倍频数、分频比、占空比等。在话框的"Clock multiplication factor" 栏中可选择时钟的倍频数,如选择倍频数为"2",则c0的时钟频率为100MHz。也可以在 "Clock division factor" 栏目中选择c0的分频比,如选择分频比为"2",则c0的输出频率为25MHz。

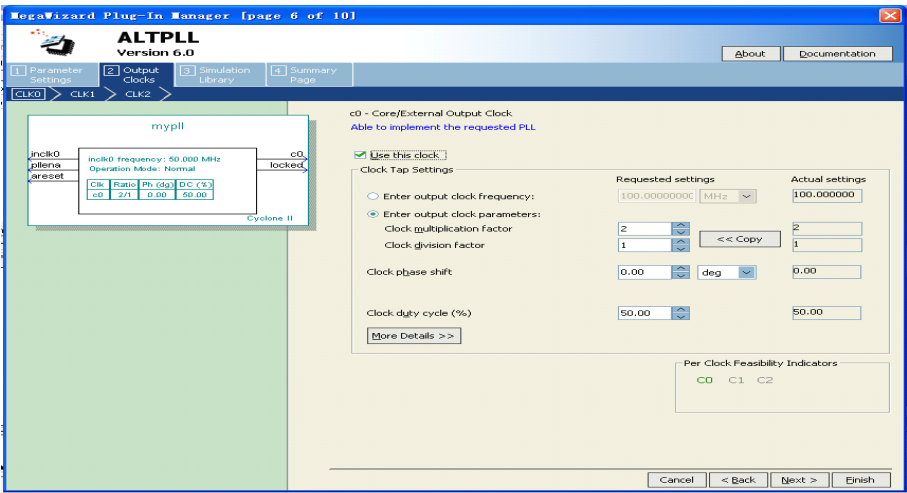


图 4-46 "MegaWizard Plug-In Manager[page 6]"对话框

用鼠标左键单击图 4-46 对话框下的"Next" 按钮,弹出如图 4-47 所示的对话框,这是嵌入式锁相环设计的最后一个对话框,用于选择输出设计文件,此对话框设置可保持默认"。

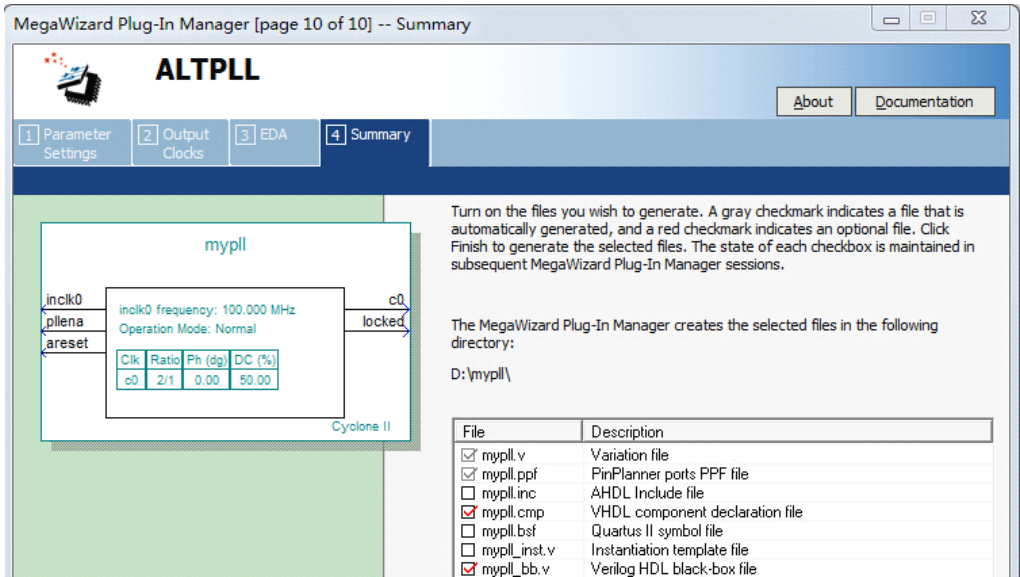


图 4-47 "MegaWizard Plug-In Manager[page 10]"对话框

用鼠标左键单击图 4-47 所示对话框下方的"Finish" 按钮,完成嵌入式锁相环的设计。

注意：在锁相环参数的设置过程中,应注意每个对话框上方出现的提示信息,如果出现 "Able to implement the requested PLL" 信息,则说明设置的参数是可以接受的；如果出现 "Cannot implement the requested PLL" 信息时,则表示设置的参数是不可接受的,需要及时修改。

完成嵌入式锁相环的设计后,打开嵌入式锁相环的 Verilog HDL 设计文件 mypll.v,首先对设计文件进行编译,然后仿真设计文件。嵌入式锁相环的仿真波形如图 4-48 所示。

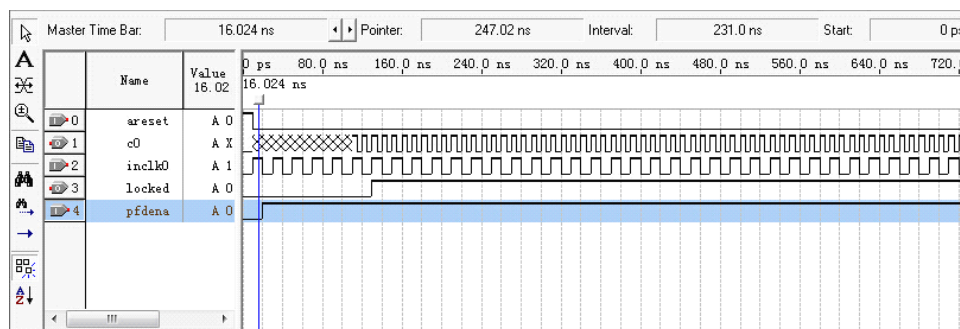


图 4-48 嵌入式锁相环的仿真波形

在仿真波形中,inclk0 是外部时钟输入端,在设置仿真输入时钟的频率时,其频率不应与实际设计电路的输入时钟频率有太大的差异。例如,设计电路时钟频率为 50MHz,则仿真输入时钟的频率也应选择在 50MHz(周期为 20ns) 范围内,否则将得不到仿真结果。

### 三、实验报告与要求

- (1) 完成实验内容中要求的各项任务。
- (2) 记录文件clk\_1\_gen.v中的代码。
- (3) 记录7分频电路代码。
- (4) 记录每个实验内容的波形仿真结果、硬件验证结果,并总结调试过程中出现的问题和解决方案。

## 实验六：计数器与时钟电路设计

### 一、实验目的

- (1) 掌握计数器硬件描述语言设计方法。
- (2) 掌握计数器LPM实现方法。
- (3) 掌握层次化设计方法实现时钟电路。

### 二、实验内容及步骤

#### 1. 计数器

在Verilog HDL中,可以用 $Q=Q+1$ 简单地实现一个计数器,也可以用LPM来实现。下面分别对这两种方法进行介绍。

实现一个8位计数器。计数器从“00000000”开始计到“11111111”,计数器的模是256。计数器模块还需要包含一个时钟clock、一个使能信号en、一个异步清0信号aclr和一个同步数据加载信号sload。模块符号如图4-49所示。

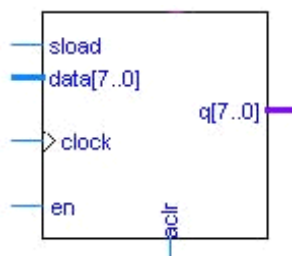




图4-49 模块符号

方法一：建立项目，并完成下面代码

Verilog HDL代码如下：

```
module counter_8bit (q, data, clock,en,sload,aclr);
input clock,en,sload,aclr;
input [7:0] data;
output [7:0] q;
reg [7:0] q;
always @ (posedge clock or posedge aclr)
begin
    if (aclr)
        q<=0;
    else if (sload)
        q<=data;
    else if (en)
        q<=q+1'b1;
end
endmodule
```

Verilog HDL文件另存为counter\_8bit.v，并将其设定为项目的最顶层文件，再进行语法检查。

引脚分配：用KEY[0]表示clock，SW[7..0]表示data，SW[8~10]分别表示en、sload和aclr；LEDR[7..0]表示q。将程序下载到开发板上进行硬件验证。

设计要求：修改上述代码，把计数器的模更改为100，并进行验证。

方法二：使用LPM实现8位计数器。

LPM是指参数化功能模块，用LPM可以非常方便快捷地实现一个计数器。

第1步：选择Tools->MegaWizard Plug-In Manager命令，打开如图4-50所示的对话框。

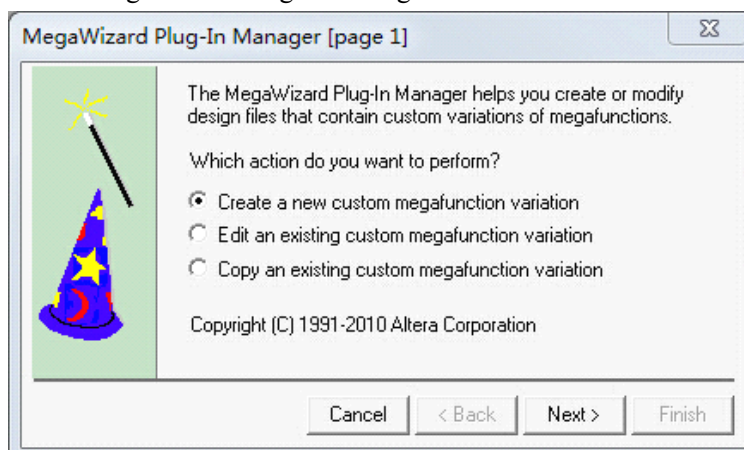


图4-50 "MegaWizard Plug-In Manager[page 1]"对话框

第2步：直接单击Next按钮，出现如图4-51所示的对话框。在对话框左边的选择框中选择“LPM\_COUNTER”，在输出文件类型单选框中选中“Verilog HDL”，并输入文件名为“counter\_lpm”。

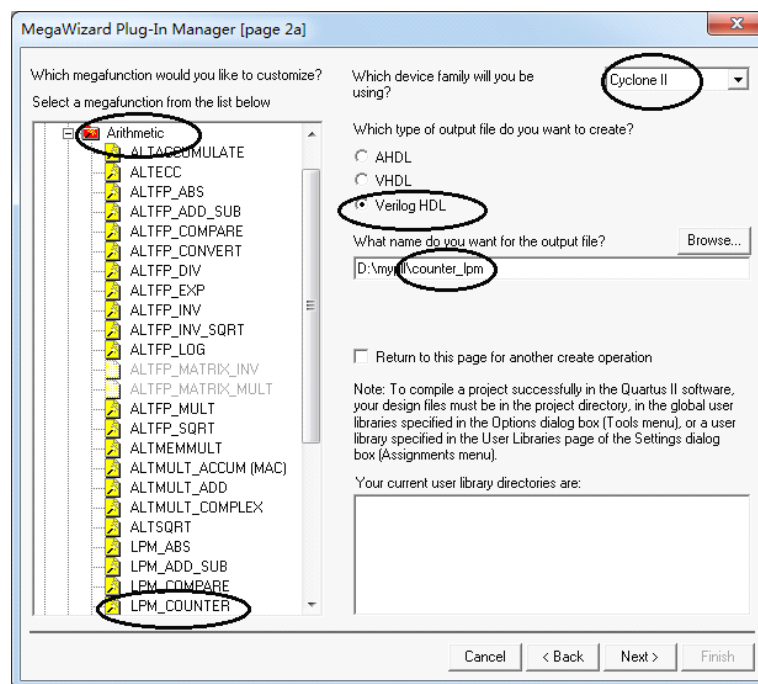


图4-51 "MegaWizard Plug-In Manager[page 2a]"对话框

第3步：完成设置后直接单击Next按钮，打开如图4-52所示的对话框。在输出位数的下拉框中选择“8 bits”，在计数方向的单选框中选中“Up only”。这个设置表示生成的计数器是8位加法计数器。

第4步：独立设计模为七的计数器

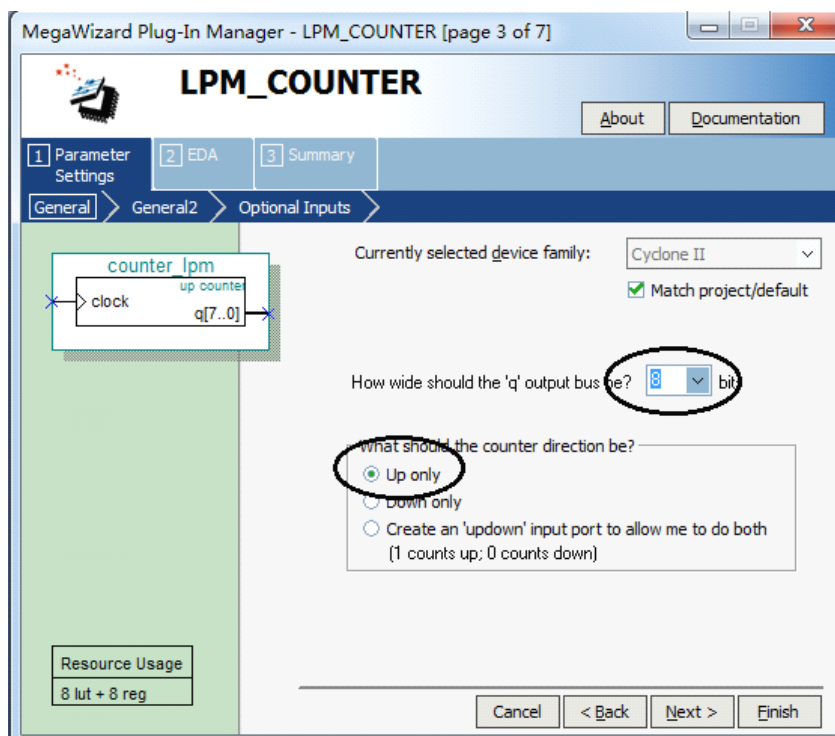


图4-52 "MegaWizard Plug-In Manager[page 3]"对话框

第4步：单击Next按钮后，出现如图4-53所示的对话框。在该对话框中选择添加额外的端口，在这里选中“Count Enable”选项，表示添加了一个计数使能端口，此时在左边的图形符号中可以看到多了一个“cnt\_en”的引脚。

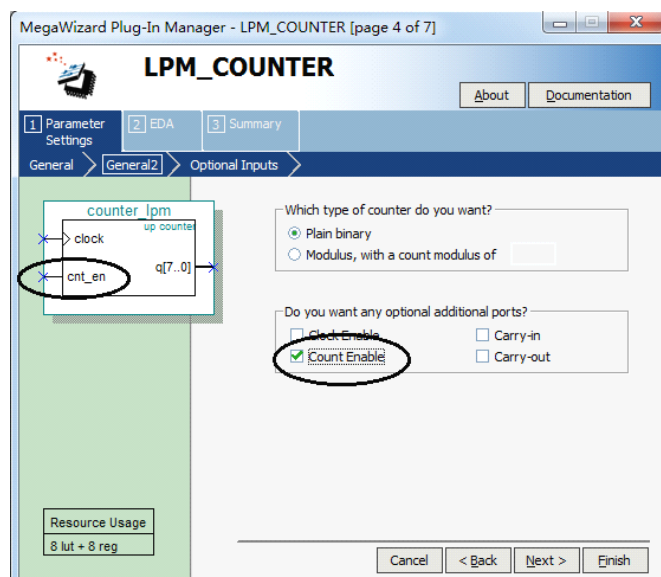


图4-53 "MegaWizard Plug-In Manager[page 4]"对话框

第5步：单击Next按钮，打开如图4-54所示的下一个对话框。在同步输入（Synchronous inputs）处选择“Load”，在异步输入（Asynchronous inputs）处选择“Clear”。表示在计数器中添加了一个同步置数端和一个异步清0端，在左边的图形符号中可以看到又添加了一个aclr、sload和用于置数用的data[7..0]。

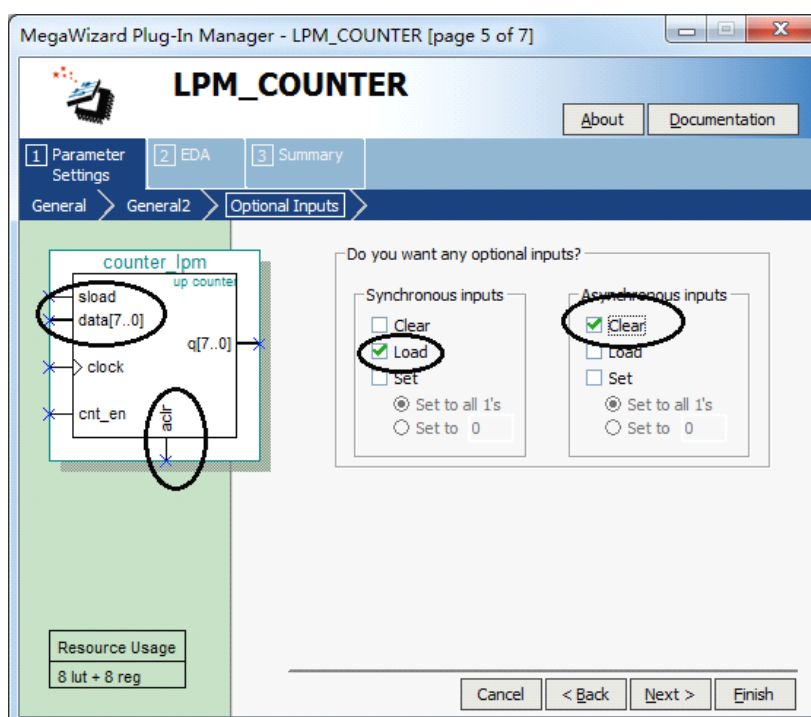


图4-54 "MegaWizard Plug-In Manager[page 5]"对话框

第6步：继续单击Next按钮直到结束为止。到此即完成了一个8位计数器的设计，同时生成了一个Verilog HDL文件此counter\_lpm.v。

第7步：接着需要将生成的counter\_lpm.v文件添加到项目中，如图4-55所示，在项目浏览器窗口中，右击“Device Design Files”，在下拉菜单中选择“Add/Remove Files in Project”命令。



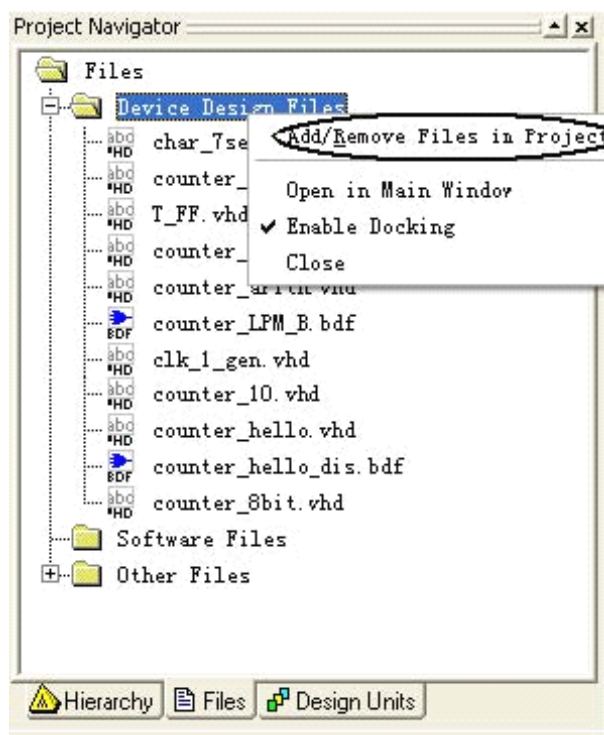


图4-55 项目浏览器窗口

第8步：选择添加文件命令后，打开如图4-56所示的对话框。在“File name”处可直接输入将添加的文件名，或通过点击右边的... 浏览按钮，打开浏览窗口，选择需要添加的文件。然后点击右边的Add按钮，即完成。

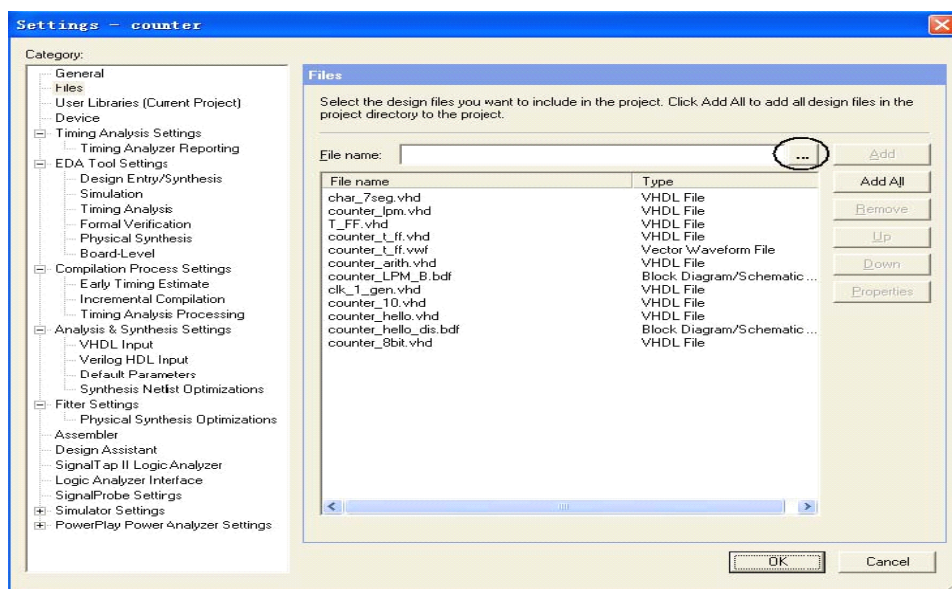


图4-56 添加文件窗口

第9步：将counter\_1pm.v设定为顶层设计文件，进行语法检查后，执行与方法一相同的操作即可。

### 3.时钟电路

利用上面设计好的计数器和分频器设计一个实时的时钟。一共需要1个模24计数器、2个模6计数器、2个模10计数器、一个生成1Hz的分频器和6个数码管解码器。最终用HEX5~HEX4显示小时（0~23），用HEX3~HEX2显示分钟（0~59），用HEX1~HEX0显示秒

钟（0~59）。

下面是模24的计数器的设计：

第1步：新建一个Verilog HDL文件，并另存为counter\_24.v。

第2步：输入以下代码：

```
module counter(q1,q0, clk,en,clr);
input clk,en,clr;
output [3:0]q1,q0;
reg [3:0] q1,q0;
always @ (posedge clk or negedge clr)
begin
    if (!clr)
    begin
        q0<=0;q1<=0;
    end
    else if (en)
    begin
        if (((q1>2) || (q0>9) || ((q1==2) && (q0>=3))))
        begin
            q0<=0; q1<=0;
        end
        else if (q0==9)
        begin
            q0<=0; q1<=q1+1'b1;
        end
        else q0<=q0+1'b1;
    end
end
endmodule
```

第3步：语法检查，通过后直接生成符号。

第4步：采用图形编辑器，将几个模块连接起来构成一个时钟。最终效果如图4-57所示。

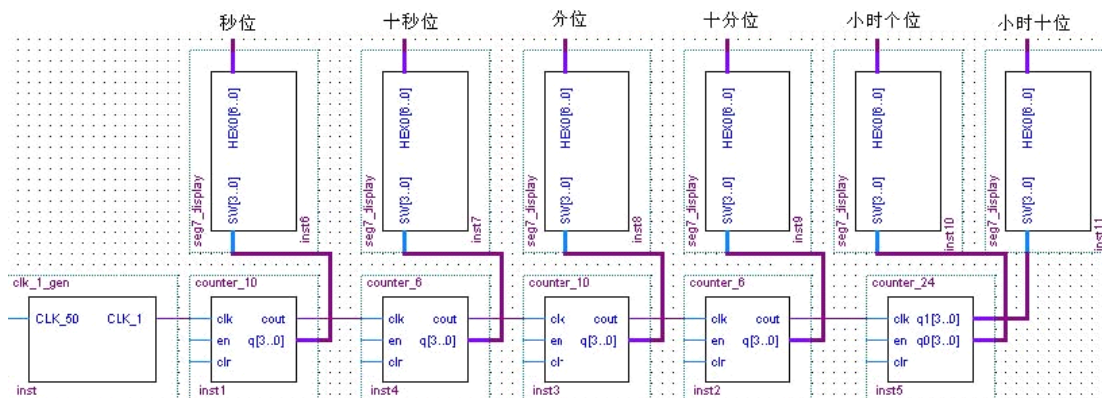


图4-57 顶层文件电路图

第5步：在加上相应的输入和输出端口，保存为clock.bdf。

第6步：语法分析、编译和下载。

### 三、实验报告与要求

- (1) 完成实验内容中要求的各项任务。
- (2) 记录模为100、24、6、10的计数器代码
- (3) 记录时钟电路完整的电路图
- (4) 记录每个实验内容的波形仿真结果、硬件验证结果，并总结调试过程中出现的问题和解决方案

## 实验七：存储器的设计

### 一、实验目的

- (1) 掌握用LPM、Verilog HDL、片外RAM实现存储器。
- (2) 掌握存储器初始化方法。
- (3) 掌握存储器的简单使用。

### 二、实验内容及步骤

在计算机系统中，一般都提供一定数量的存储器。在用FPGA实现的系统中，除可以使用FPGA本身提供的存储器资源外，还可以使用FPGA的外部扩充存储器。本实验要求设计一个 $32 \times 8$  RAM，如图4-58所示，它包含5位地址、8位数据口和一个写控制端口。实现这个存储器有两种方法：一种是采用FPGA上的存储器块实现。EP2C35 FPGA片内共有105个M4K RAM块，每个M4K包含4096位，支持 $4K \times 1$ 、 $2K \times 2$ 、 $1K \times 4$ 和 $512 \times 8$ 四种配置。本实验选择 $512 \times 8$ 配置，而且只使用RAM中的32个字节。另一种是采用外部存储器芯片实现。

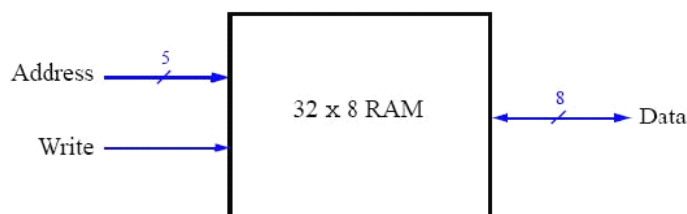


图4-58  $32 \times 8$  RAM示意图

每个M4K都有专用的寄存器用于所有输入、输出与时钟同步，因此在使用M4K时，要让输入、输出端口之一或全部与输入时钟同步。改进之后的 $32 \times 8$  RAM模块如图4-59所示。

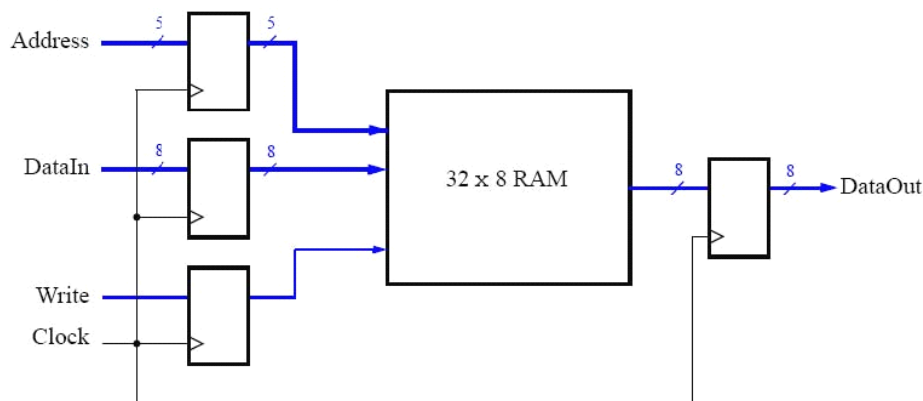


图4-59 32×8 RAM 电路图

## 1. LPM实现32×8 RAM

常用的逻辑电路如加法器、计数器、寄存器和存储器都可调用QuartusII提供的参数化功能模块LPM实现。本实验采用LPM altsyncram实现存储器。

第1步：新建一个Quartus项目，并命名为lpm\_sram。

第2步：新建一个图形文件，并命名为lpm\_sram.bdf。打开符号（symbol）库，如图4-60所示。从左边符号库中选中“altsyncram”，点击“ok”

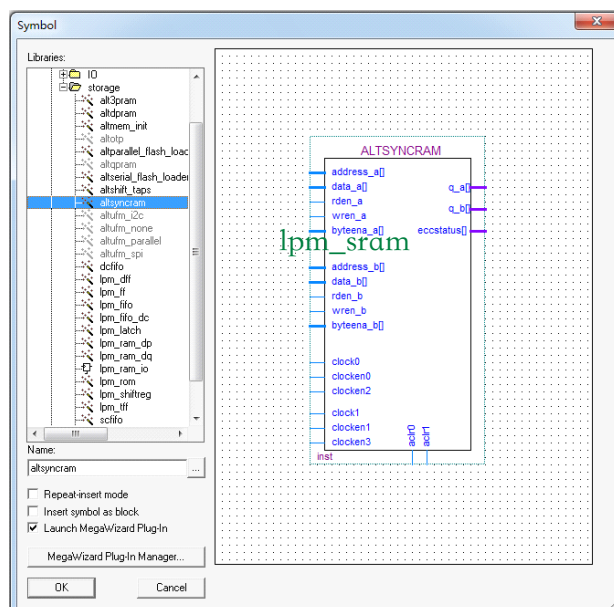


图4-60 符号库

出现如图4-61所示的界面，

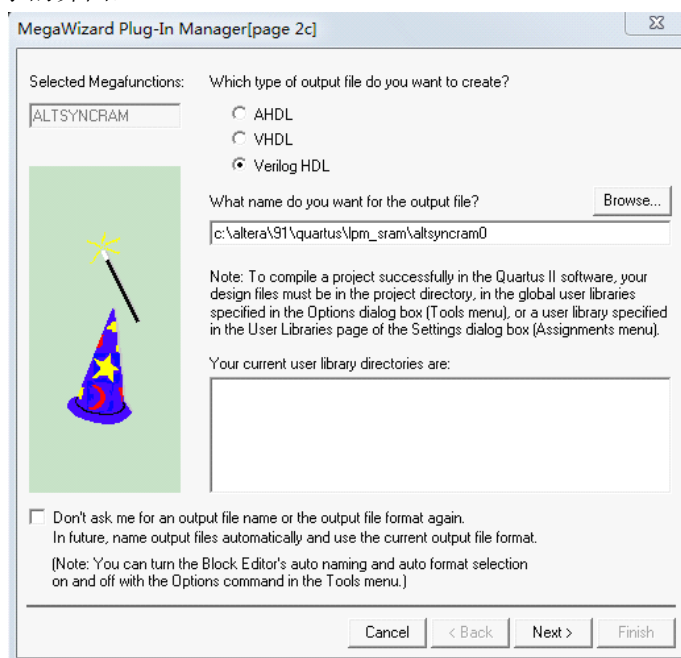


图4-61 "MegaWizard Plug-In Manager- altsyncram [page 2c]"对话框

第4步：点击Next按钮，在下一个对话框中选择“with one read/write port”模式，如图4-62所示。

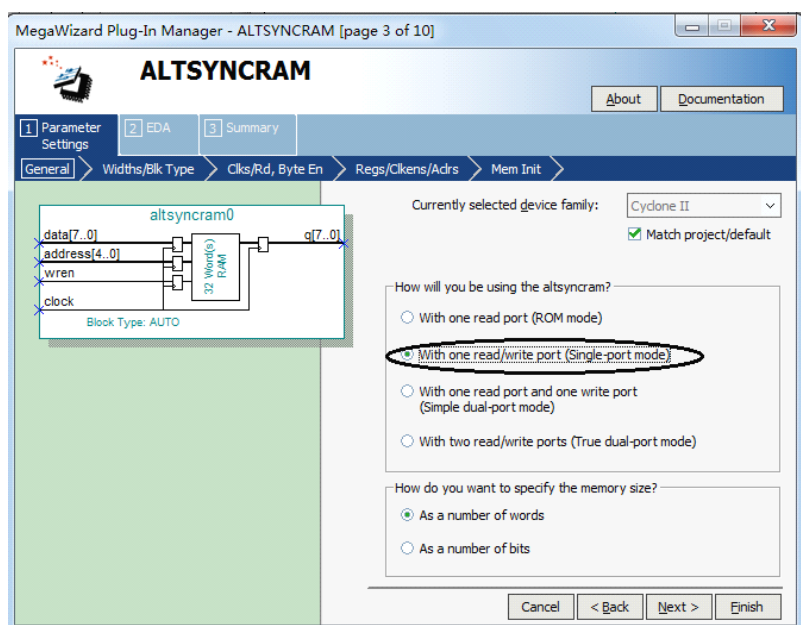


图4-62 "MegaWizard Plug-In Manager- altsyncram [page 3]"对话框  
第5步：点击Next按钮，在下一个对话框中选择存储器容量是32个字节，如图4-63所示。

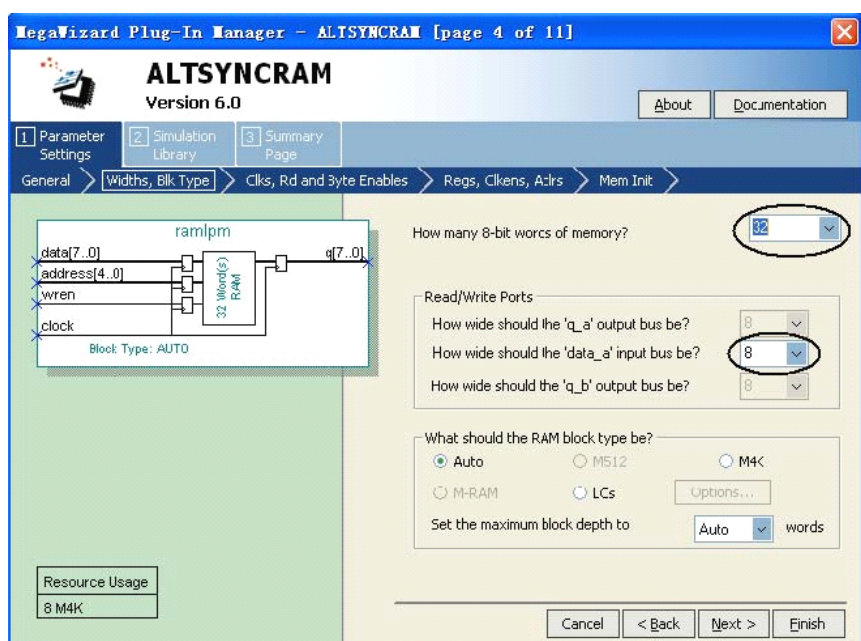


图4-63 "MegaWizard Plug-In Manager- altsyncram [page 4]"对话框  
第6步：其他对话框均采用默认值即可，最后按Finish按钮即完成RAM模块的设计。  
第8步：添加相应的输入和输出端口，端口命名如图4-64所示。

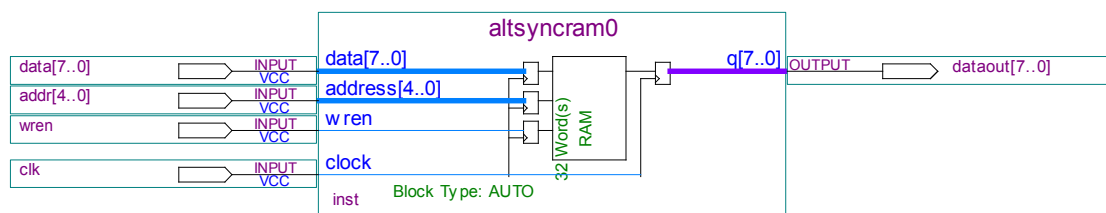


图4-64 altsyncram端口命名  
第9步：进行语法检查，检查通过后，打开Pin Planner，用SW7~SW0作为数据输入、



SW12~SW8作为地址的输入、SW13作为Write信号、结果显示在LEDG上。

第10步：开始编译并下载。

第11步：验证实验结果。首先，写数据到存储器中。例如设置地址值为"00000"，即SW12~SW8="00000"；写入的数据为"01"H，即SW7~SW0="00000001"；再将SW13置1，表示开始写，写完后重新置0等待下一次写操作。用同样的方法，将表4-10中的数据写入到相应的存储地址中。接着是从存储器中将数据读出，将SW13置0，设置不同的地址值，就会在LEDG上显示之前写入的数据了。

表4-10 存储器中的数据分配

地址	数据
00000	01H
00001	02H
00010	04H
00011	08H
00100	10H
00101	20H
00110	40H

第12步：存储器中的数据除了可以用上述方法写入外，还可以通过存储器初始化文件（MIF文件）进行初始化。双击LPM模块，点击Next按钮直到“page 5 of 7”这一对话框如图4-65所示。选择“yes”选项，并输入MIF文件“32B.mif”，然后点击Finish按钮结束即可。

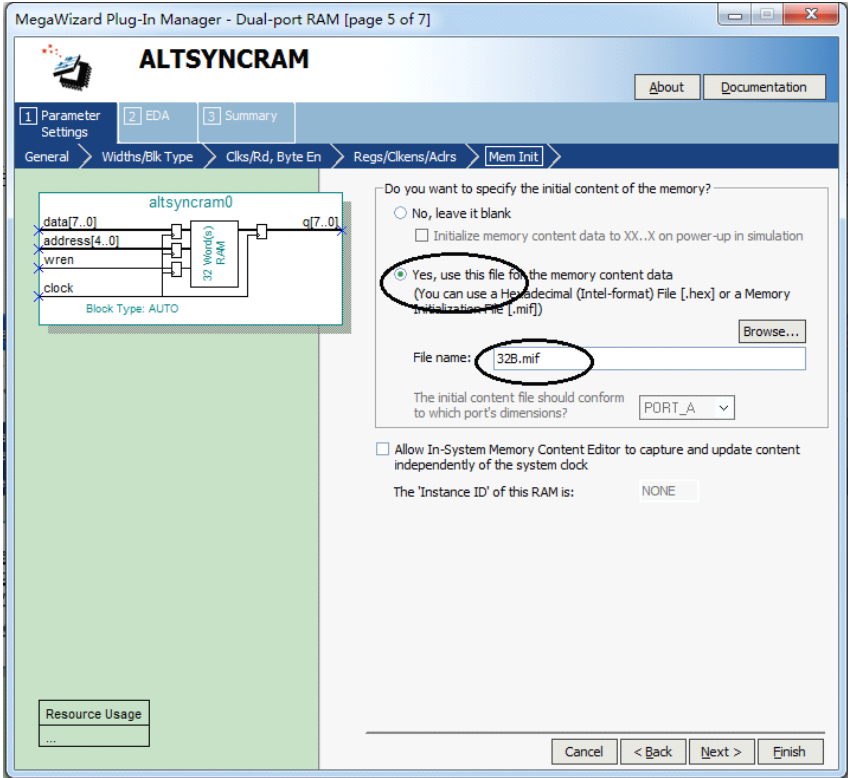


图4-65 "MegaWizard Plug-In Manager- altsyncram [page 5]"对话框

第13步：接着用记事本生成一个32B.mif文件，文件内容格式如下

WIDTH=8;

DEPTH=32;

```

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;
CONTENT BEGIN
00 : 01;
01 : 02;
02 : 03;
03 : 04;
.....
END;

```

其中的 WIDTH 是指数据带宽，这里设为 8 位；DEPTH 是指存储器容量，这里设为 32 个单元；“ADDRESS\_RADIX=HEX”表示以下地址的表示形式是十六进制；再以“CONTENT BEGIN”表示以下将开始存储器数据的输入，而以“END;”结束；“00:01 ...”表示地址“00H”处的数据是“01H”。

此外，用上述方法只能自动生成一些简单数据，对于复杂数据，比如波形（如正弦波）数据，需要在存储器初值设定文件的界面上逐个将数据填入。利用 C 语言可以生成存储器初值设定文件(.mif)中的数据。能生成正弦波数据的 C 语言源程序(myram.c)如下：

```

#include <stdio.h>
#include "math.h"
main()
{int i,k;
for(i=0;i<256;i++)
{k=128+128*sin(360.0*i/256.0*3.1415926/180);
printf("%d:%d;\n",i,k);
}

Return;
}

```

在源程序中,i 表示 8 位计数器提供的地址(从 0 到 255 变化), 由于正弦波的一个周期是从 0° 到 259° , 因此 i 对应的角度是  $360*i/256$  。另外, 存储器中的数据是 8 位无符号数, 因此在正弦函数前增加了 128 的倍数和 128 的增量, 使 0° 对应的 8 位无符号数的值为 128(表示正弦值 0),90° 对应的值为 255(表示正弦值 1),270° 对应的值为 0(表示正弦值 -1),依此类推。

把 myram.c 文件编译成可执行文件后, 在 DOS(Windows 的命令提示符)环境下执行命令 :

```
myram>myram_1.mif
```

则将 myram 文件执行的结果保存在 myram\_1.mif 文件（该文件可以任意命名，也可以不加文件属性）中。以“记事本”方式打开 myram\_1.mif 文件，将其内容复制到存储器初值设定文件(.mif)中的数据中，代替源文件中的地址和数据。

注意：由于原来的存储器初值设定文件 (.mif) 中的地址基数选用 "Hexadecimal"(十六进制), 而用 C 语言生成的地址基数是十进制, 因此, 需要把 mydds.mif 中的 "ADDRESS\_RADIX=HEX;" 语句修改为 "ADDRESS\_RADIX=DEC;" , 表示地址基数为十进制。如果原来的存储器初值设定文件中的地址基数选用十进制, 则不需要修改。在 Quartus II 环境下打开修改后的 mydds.mif, 其存储的数据即为正弦波的数据。

## 2.用 Verilog HDL 实现

除了使用 LPM 模块实现 RAM 外, 还可以通过 Verilog HDL 代码实现。在 Verilog HDL



中可以定义存储器的数据类型，如  $32 \times 8$  RAM 可以用以下语句实现：

```
module sram_32B (dataout, datain, addr, clk, wren);
input clk, wren;
input [4:0] addr;
input [7:0] datain;
output [7:0] dataout;
reg [7:0] dataout;
reg [7:0] sram [31:0];
always @ (posedge clk)
begin
    if ( wren)
        sram[addr] <=datain ; //Write data
    else
        dataout<=sram[addr]; //Write data
    end
endmodule
```

在 Verilog HDL 程序中也可以对定义的 RAM 进行数据初始化，只需在 RAM 的定义前加上以下语句即可：

```
(* ramstyle="no_rw_check, m4k", ram_init_file=" sram_32B.mif" *) reg [7:0] sram [31:0];
```

3.用片外 RAM 实现

在DE2平台上还集成了一个SRAM芯片IS16LV25616AL-10，这是一个 $256K \times 16$ 位的SRAM。这个SRAM芯片的接口包括了18位地址口A17~A0，16位双向数据口I/O15~I/O0，还包括CE、OE、WE、UB和LB等控制信号，且这些控制信号都是低电平有效。具体功能见表4-11。

表4-11 SRAM芯片控制信号功能表

引脚名称	功能
CE	芯片使能信号
OE	输出使能信号，可以在读操作或所有操作中置为低电平
WE	写使能信号，写操作时置为低电平
UB	高字节选择
LB	低字节选择

本实验的目的是用SRAM芯片实现 $32 \times 8$ 的RAM，此时FPGA与SRAM和外围接口的连接关系如图4-66所示。

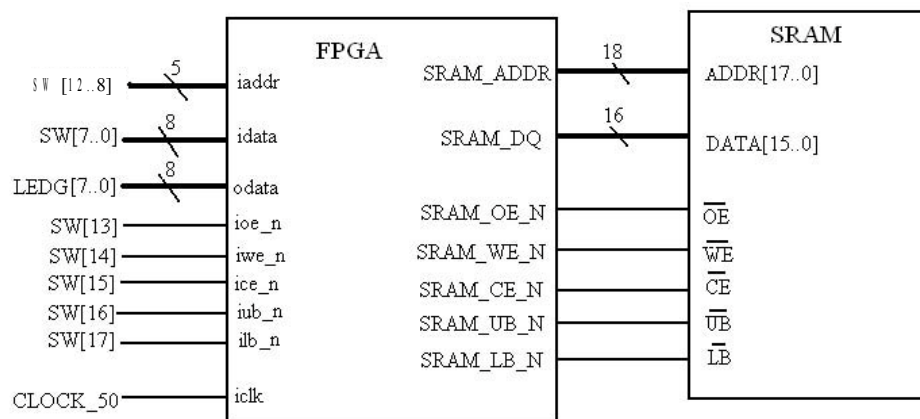


图4-66 FPGA与SRAM和外围接口的连接关系

从图4-66可以看出本实验没有使用SRAM的所有数据及地址引脚，所以在程序中将不需要使用的引脚接地。

新建一个Verilog HDL文件，并命名为ext\_sram.v。

Verilog HDL部分程序如下：

```
.....
sram_lb_n <= ilb_n;
sram_ub_n <= iub_n;
sram_oe_n <= ioe_n;
sram_ce_n <= ice_n;
sram_we_n <= iwe_n;
sram_addr <= {13'b0, iaddr};
odata <= sram_dq[7:0];
if (!iwe_n)
    sram_dq <= {8'b0, idata[7:0]};
else
    sram_dq <= 16'bZ;
.....
```

#### 4.SRAM 的应用

从以上3种实现RAM的方法中选择一种方法实现64×8的RAM。并要求在RAM的第0～15单元内分别写入1～16，可以通过手动输入也可以通过存储器初始化文件（MIF）完成。

接着按顺序从第0～15单元每一秒钟读出一个单元的数据，并显示在LED上。

### 三、实验报告与要求

- (1) 完成实验内容中要求的各项任务。
- (2) 记录ext\_sram.v代码。
- (3) 记录第四步（SRAM的应用）中的代码或原理图。
- (4) 记录每个实验内容的硬件验证结果，并总结调试过程中出现的问题和解决方案。

## 实验九：交通灯控制器设计

### 一、实验目的

根据设计要求设计数字系统。

### 二、实验内容

设计要求：设计一个简单的十字路口交通灯控制器。交通灯分东西和南北两个方向，均通过数码管和指示灯指示当前的状态。设两个方向的流量相当，红灯时间 45s，绿灯时间 40s，黄灯时间 5s。完成代码编写、软件仿真和硬件验证。

设计提示：从交通灯的工作机理来看，无论是东西方向还是南北方向，都是一个减法计数器。只不过计数时还要判断红绿灯情况，再设置计数器的模值。

表 4-12 所示为一个初始状态和 4 个跳变状态。交通灯控制器工作时状态将在 4 个状态间循环跳变，整个交通灯则完全按照减计数器原理进行设计。

表 4-12 交通灯控制器状态变换表

状态	当前计数值			下一个 CLOCK 到来时新模值		
	东西 方向指示	南北 方向指示	东西 - 南北 方向指示	东西 方向指示	南北 方向指示	东西-南北 方向指示
初始	0	0		45	40	红-绿
1	6	1	红-绿	5	5	红-黄
2	1	1	红-黄	40	45	绿-红
3	1	6	绿-红	5	5	黄-红
4	1	1		45	40	红-绿

输入/输出引脚列表如表 4-13 所示。

表 4-13 交通灯控制器输入/输出引脚列表

输入信号				
序号	信号名称	位宽	端口类型	备注
1	clk	1	I	系统时钟
2	urgency	1	I	系统紧急信号
3	east_west	8	I	东西方向时钟计数
4	south_north	8	I	南北方向的时钟计数
输出信号				
1	led	6	O	交通指示灯

### 三、实验报告与要求

- (1) 记录交通灯控制器的代码或者原理图，并给出代码相应的解释。
- (2) 记录仿真波形。
- (3) 记录硬件验证结果，并总结调试过程中出现的问题和解决方案

## 实验十 抢答器

### 一、实验目的

- (1) 掌握根据设计要求编写源代码。
- (2) 掌握根据仿真要求编写测试代码。
- (3) 掌握在Quartus II中调用ModelSim进行仿真。

## 二、实验内容

### 1.设计要求

抢答器可容纳 4 组参赛者抢答，每组设置一个抢答按键。要求：抢答器具有第一抢答信号的鉴别和锁存功能，当按下复位键 reset 后开始抢答，抢答器监测到首先按下按钮的选手并点亮对应的 led 灯，此后其余选手的抢答无效。完成该抢答器源代码和测试代码编写，并进行软件仿真和 DE2 开发硬件验证。

### 2. 设计提示

(1)引脚分布图或者基本框图如图 4-67 所示。

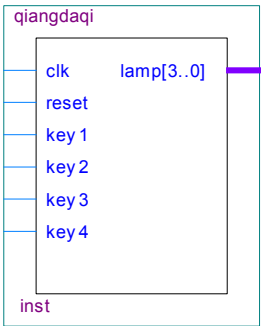


图 4-67 抢答器引脚分布图

(2)输入/输出引脚列表如表 4-14 所示。

表 4-14 抢答器输入/输出引脚列表

输入信号				
序号	信号名称	位宽	端口类型	备注
1	clk	1	I	系统时钟
2	reset	1	I	异步复位
3	key1	1	I	1 号按键
4	key2	1	I	2 号按键
5	key3	1	I	3 号按键
6	key4	1	I	4 号按键
输出信号				
1	lamp	4	O	抢答结果显示灯

(3) 输入/输出的关系

Input: clk, reset 和 key1, key2, key3, key4

Output: lamp

- ① Reset 表征抢答开始。
- ② key1、key2、key3、key4 分别对应 4 组选手的抢答按键。
- ② lamp 表示 4 个抢答结果显示灯。

设计中要求的锁存功能可以用计数器 count 和寄存器 enable 来实现。每当有选手按下按钮时，计数器 count 的值+1，同时 enable 记录选手的序号。抢答器算法参考流程如图 4-68 所示。

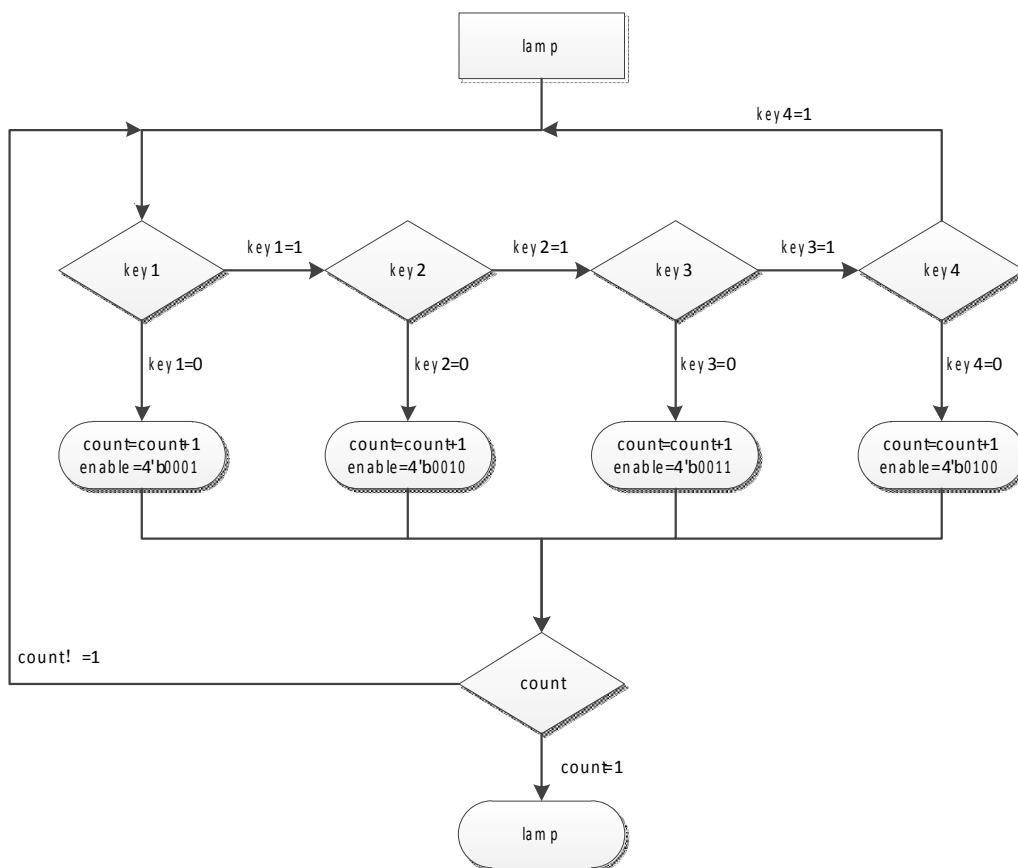


图 4-68 抢答器算法参考流程图

#### (4) 设计注意事项

- ① 设计中注意计数器 `count` 的设计容量，由于锁存抢答结果的功能要求，在下次抢答，即再次按下 `reset` 之前，要保留抢答的结果。这就要求 `count` 足够大，不然在 `count` 重新计数后抢答结果就会改变。
- ② 设计中注意时钟频率的问题，由于系统时钟频率过快，会导致计数器 `count` 在短时间内重置，从而影响抢答结果。所以需要分频使时钟频率降低，然而值得注意的是，当时钟频率过慢时，又会造成抢答器反应不灵敏。请设计者认真思考计算并给出合适的分频倍数。
- ③ 设计中注意按键的消抖问题，每次按键后需要隔一段时间才能进行下次按键操作，依据此项原则给出适合的消抖模块。设计者应注意消抖模块的设计可与之前的分频模块设计结合，产生较为合理的设计。

#### 3. 仿真程序（测试代码）要求

测试时 `key1`、`key2`、`key3`、`key4` 信号分别在不同时刻给一个低电平信号，以此模拟选手在比赛中进行抢答的情况，观察判断输出 `lamp` 是否正确对应输入激励中最先为低电平的序号。

附：仿真波形图和说明如图 4-69 所示。

### 三、实验报告与要求

- (1) 记录抢答器源代码、测试代码，并给出代码相应的解释。
- (2) 记录实验仿真波形和说明。
- (3) 记录DE2硬件验证结果，并总结调试过程中出现的问题和解决方案

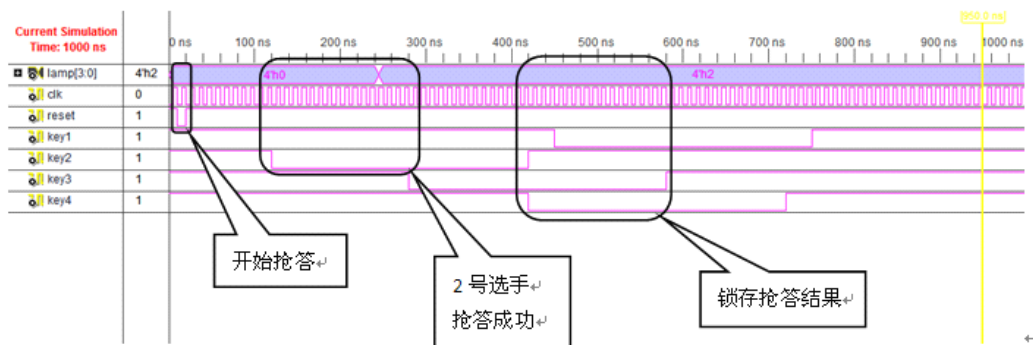


图 4-69 抢答器波形仿真图

## 实验十一 串行数据序列检测器

### 一、实验目的

- (1) 掌握根据设计要求编写源代码。
- (2) 掌握根据仿真要求编写测试代码。
- (3) 掌握在Quartus II中调用ModelSim进行仿真。

### 二、实验内容

#### 1. 设计要求

检测输入的串行数据序列，当检测到输入序列为 1011 时，LED 灯一直熄灭。完成源代码和测试代码编写，并进行软件仿真。

#### 2. 设计提示

- (1) 引脚分布图或者基本框图如图 4-70 所示。

Parameter	Value	Type	ED
MULTIPLE	5000000	Signed Integer	
S0	000	Unsigned Binary	
S1	001	Unsigned Binary	
S2	010	Unsigned Binary	
S3	011	Unsigned Binary	
S4	100	Unsigned Binary	

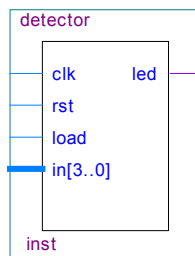


图 4-70 串行数据序列检测器引脚分布图

- (2) 输入/输出引脚列表如表 4-15 如所示。

表 4-15 串行数据序列检测器输入/输出引脚列表

输入信号				
序号	信号名称	位宽	端口类型	备注
1	clk	1	I	系统时钟
2	rst	1	I	复位信号
3	load	1	I	加载并行数据信号
4	in	4	I	并行输入的 4 位序列
输出信号				
1	led	1	O	检测到序列为 1011

### (3) 输入/输出的关系

Input: clk, rst, load, in

Output: led

In(3:0)为一个并行输入的4位序列，当load信号有效时，并行输入被存入移位寄存器shift\_register，接着产生串行序列输出serial\_out，检测到序列1011时led点亮。

### (4) 设计注意事项

由于采用并行数据输入，若load信号采用按键，加载数据时为避免加入多个输入的并行数据，可以将系统时钟clk进行分频得到一个合适的时钟q（例如周期为0.1s）。

### 3. 仿真程序（测试代码）要求

输入序列1011测试能否正确检测，同时验证输入控制键load是否工作。

附：仿真波形图与说明如图4-71所示。

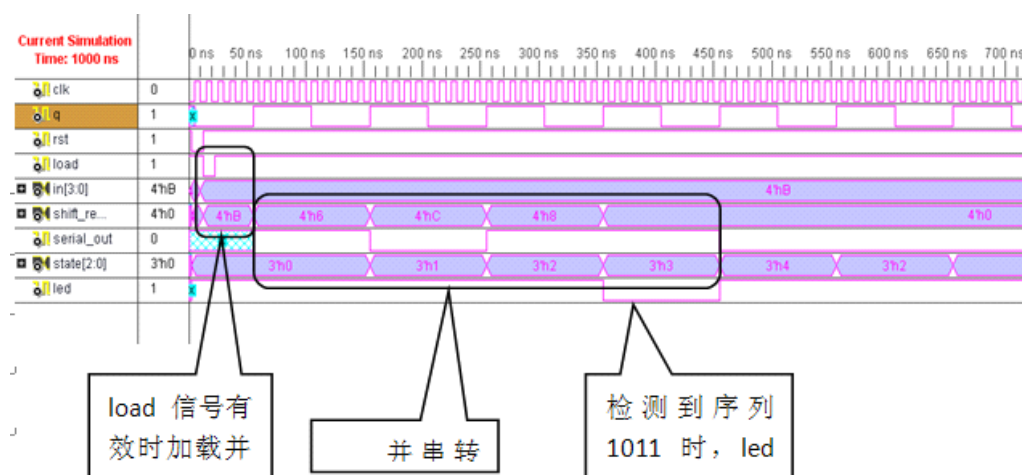


图 4-71 串行数据序列检测器波形仿真图

## 三、实验报告与要求

- (1) 记录串行数据序列检测器源代码、测试代码，并给出代码相应的解释。
- (2) 记录实验仿真波形和说明。
- (3) 记录DE2开发板硬件验证结果，并总结调试过程中出现的问题和解决方案

## 实验十二 汽车尾灯控制器

### 一、实验目的

- (1) 掌握根据汽车尾灯控制器设计要求编写源代码。



(2) 掌握根据仿真要求编写测试代码。

二、实验内容

1.设计要求

- ① 汽车尾部左右两侧各有 2 只尾灯，用作汽车行驶状态的方向指示标志。
- ② 汽车正常向前行驶时，4 只尾灯全部熄灭。
- ③ 当汽车要向左或向右转弯时，相应侧的 2 只尾灯从左向右依次闪烁。每个灯亮 1s，每个周期为 2s，另一侧的 2 只灯不亮。
- ④ 紧急刹车时，4 只尾灯全部闪，闪动频率为 1Hz。
- ⑤ 完成源代码和测试代码编写，并进行软件仿真和 DE2 开发硬件验证。

2.设计提示

(1) 引脚分布图或者基本框图如图 4-72 所示。

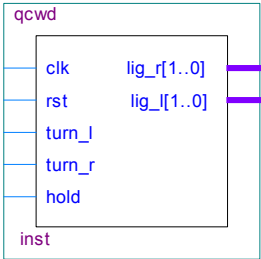


图 4-72 汽车尾灯控制器引脚分布图

(2) 输入/输出引脚列表如表 4-16 所示。

表 4-16 汽车尾灯控制器输入/输出引脚列表

输入信号				
序号	信号名称	位宽	端口类型	备注
1	clk	1	I	系统时钟
2	turn_l	1	I	左转向信号
3	turn_r	1	I	右转向信号
4	hold	1	I	刹车信号
输出信号				
1	lig_l[1:0]	2	O	左尾灯
2	lig_r[1:0]	2	O	右尾灯

(3)输入/输出关系

Input: clk, turn\_l, turn\_r, hold

Output: lig\_l[1:0], lig\_r[1:0]

- ①clk 为系统时钟，在代码中需要进行分频；
- ②turn\_l,turn\_r 和 hold 分别为输入控制信号，分别控制汽车尾灯在不同输入下的输出状态；
- ③lig\_l[1:0]和 lig\_r[1:0]为输出尾灯，其在不同的输入下，呈现不同的点亮状态。
- ④tl 和 tr 分别为左转和右转控制灯闪烁的寄存器。

(4)注意事项

- ① 分频时的计数器要设置好参数，系统提供的为 50MHz 的信号，分频产生的信号应为 1Hz。
- ② 控制优先设置时，应当将刹车设置为最优先，接下来依次为转向和前行。

3. 仿真程序（测试代码）要求

测试时依次对 hold、lig\_l 和 lig\_r 赋值，看对应的输出结果怎么样。

附：仿真波形图及说明如图 4-73 所示。

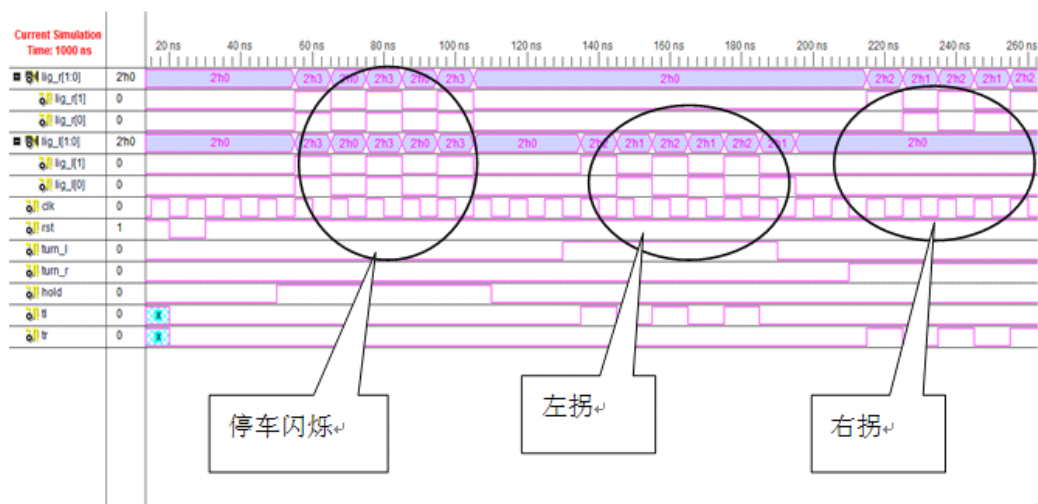


图 4-73 汽车尾灯控制器波形仿真图

### 三、实验报告与要求

- (1) 记录汽车尾灯控制器源代码、测试代码，并给出代码相应的解释。
- (2) 记录实验仿真波形和说明。
- (3) 记录DE2开发板验证结果，并总结调试过程中出现的问题和解决方案

## 实验十三 简易洗衣机控制器设计

### 一、实验目的

- (1) 掌握根据洗衣机控制器设计要求编写源代码。
- (2) 掌握根据仿真要求编写测试代码。

### 二、实验内容

#### 1. 设计要求

- ① 洗衣机正常的工作状态为待机（5s）→正转（60s）→待机（5s）→反转（60s）。
- ③ 可由用户设定循环次数，此处设计最大循环次数为 7 次。
- ④ 具有紧急情况处理功能，可在洗衣过程中直接打断工作状态转入待机状态，待紧急情况解除后重新设定并开始工作。
- ⑤ 为方便用户在洗衣过程中操作，该洗衣机还具备有暂停功能，当用户操作完成后，可继续上次未完成的工作。
- ⑥ 洗衣完成后即设定洗衣次数归零时，可报警告知用户。
- ⑦ 完成洗衣机控制器源代码和测试代码编写，并进行软件仿真和 DE2 开发板硬件验证。

#### 2. 设计提示

- (1) 引脚分布图或者基本框图如图 4-74 所示。

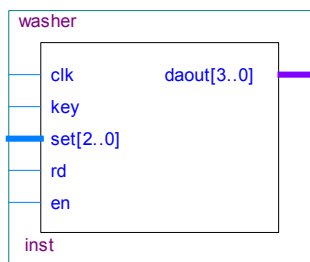


图 4-74 洗衣机控制器引脚分布图

(2) 输入/输出引脚列表如表 4-17 所示。

表 4-17 洗衣机控制器输入/输出引脚列表

输入信号				
序号	信号名称	位宽	端口类型	备注
1	clk	1	I	系统时钟
2	en	1	I	设定/工作
3	rd	1	I	复位
4	set	2	I	设定循环次数
5	key	1	I	选择输出按键
输出信号				
1	daout	4	O	经选择后的输出

(3)输入/输出关系

Input: clk, en, rd, set, key

Output: daout

- ① rd 为复位键，rd=1 时复位。rd=0 时，按下 en，可以设置循环次数 set。
- ② key 为选择输出信号。key=1，daout 输出警报器 alarm 和洗衣机的工作状态 lamp。key=0 时，daout 输出剩余循环次数 tim。

(4) 基本设计方案

整个洗衣机的设计可以由工作控制模块、循环次数计算模块以及报警器模块组成。其中工作控制模块主要实现洗衣机在正常工作的每次循环中各种工作状态之前的转换。循环次数计算模块则负责设定循环次数或者改变剩余循环次数值。报警模块则在正常工作结束后发出报警信号。

(5) 设计要点

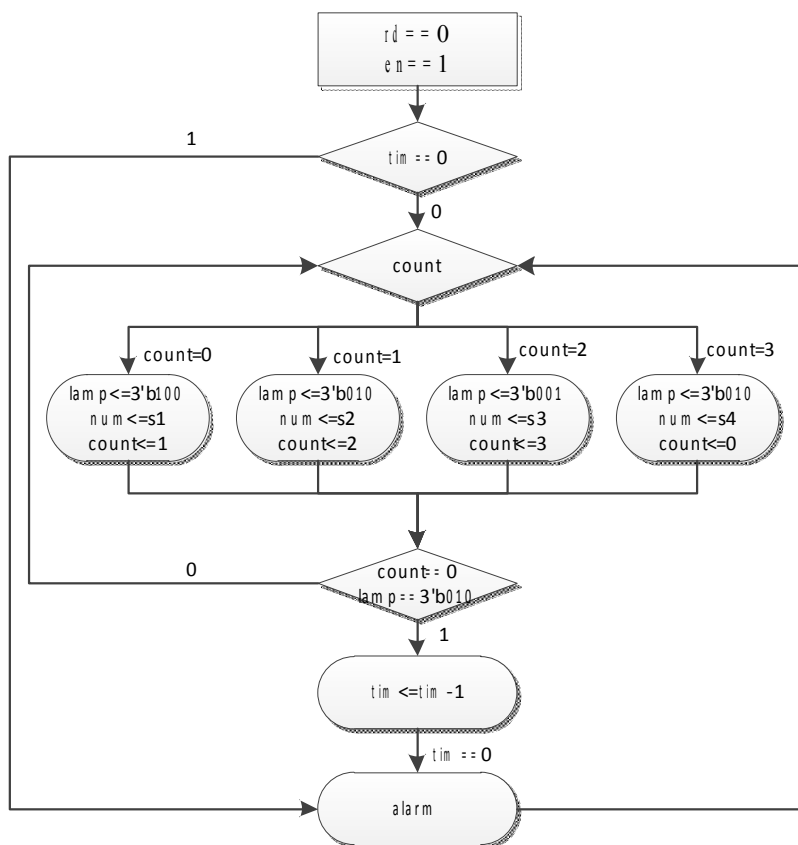


图 4-75 洗衣机工作控制模块流程图

洗衣机工作控制模块是整个设计的核心模块。它有 5s、60s 信号发生器、状态计数器、数据选择器和状态译码器等组成,能够自动执行工作状态顺序循环的指令。当  $rd=0$  且  $en=1$  时,开始工作。倒计时  $num=2$  时工作状态切换的使能信号  $temp$  置 0,同时使  $count$  的值改变,切换工作状态。当  $count=0$  且  $lamp=3'b010$  时,表示一次循环结束,则  $tim \leq tim-1$ 。当  $tim=0$  则  $alarm$  报警。

#### (6) 设计注意事项

- ① 设计中注意时钟频率的问题,由于系统时钟频率过快,会影响显示时观察结果,因此需要分频使得时钟频率降低,计算并得出合适的分频倍数。
- ② 设计中需使用一定方法使工作状态顺序循环切换。本设计中使用切换使能信号  $temp$  的低电平来改变工作状态  $count$  的值,从而实现工作状态的切换。

#### 3. 仿真程序(测试代码)要求

测试洗衣机控制器的循环工作次数是否正确设置  $set$ , 测试循环结束后是否报警。测试工作时  $en=0$  是否具有暂停工作的功能,  $rd$  是否具有重置的功能。

附: 仿真波形图及说明如图 4-76 所示。

### 三、实验报告与要求

- (1) 记录洗衣机源代码、测试代码,并给出代码相应的解释。
- (2) 记录实验仿真波形和说明。
- (3) 记录 DE2 开发板硬件验证结果,并总结调试过程中出现的问题和解决方案

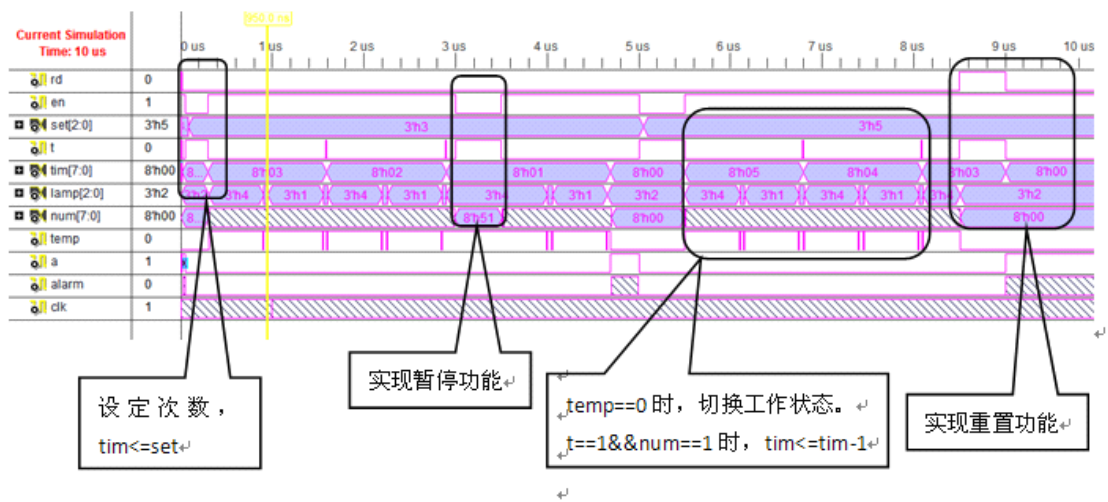


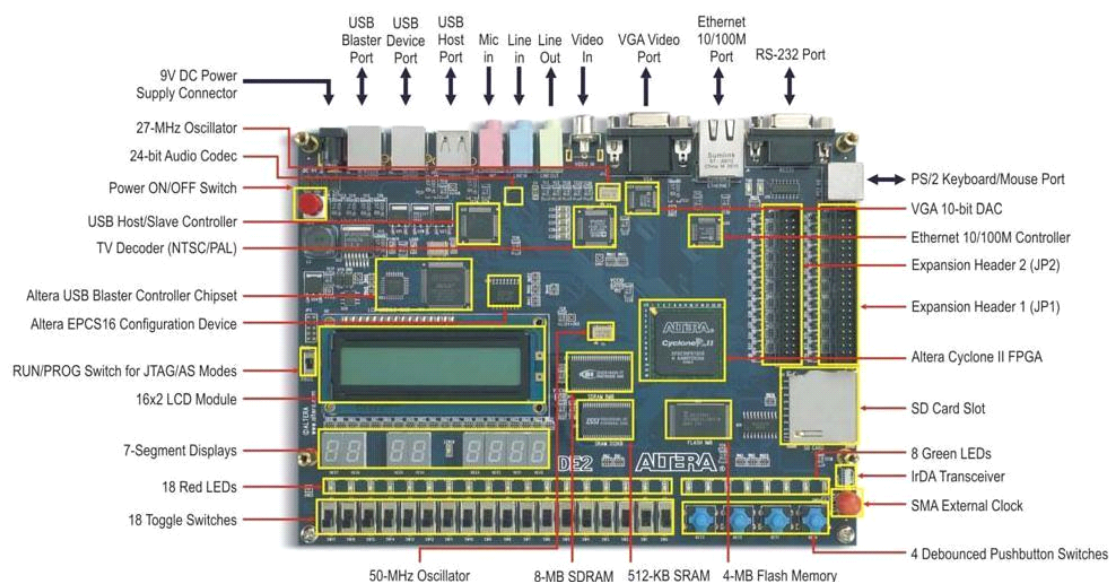
图 4-76 洗衣机波形仿真图

## 附录四 DE2 实验平台介绍

### 1.DE2 简介

DE2 实验平台是 Altera 公司针对大学和研究机构推出的 FPGA 开发平台，它为用户提供了丰富的外设，涵盖了常用的各类硬件和接口，如各类存储器、USB、以太网、视频、音频、SD 卡、液晶显示等，除此之外，DE2 还提供扩展接口供用户定制使用，可用于多媒体开发、SOPC 嵌入式系统和 DSP 等各类应用的实验和开发。

DE2 平台布局图如附图 4-1 所示。



附图 4-1 DE2 开发板

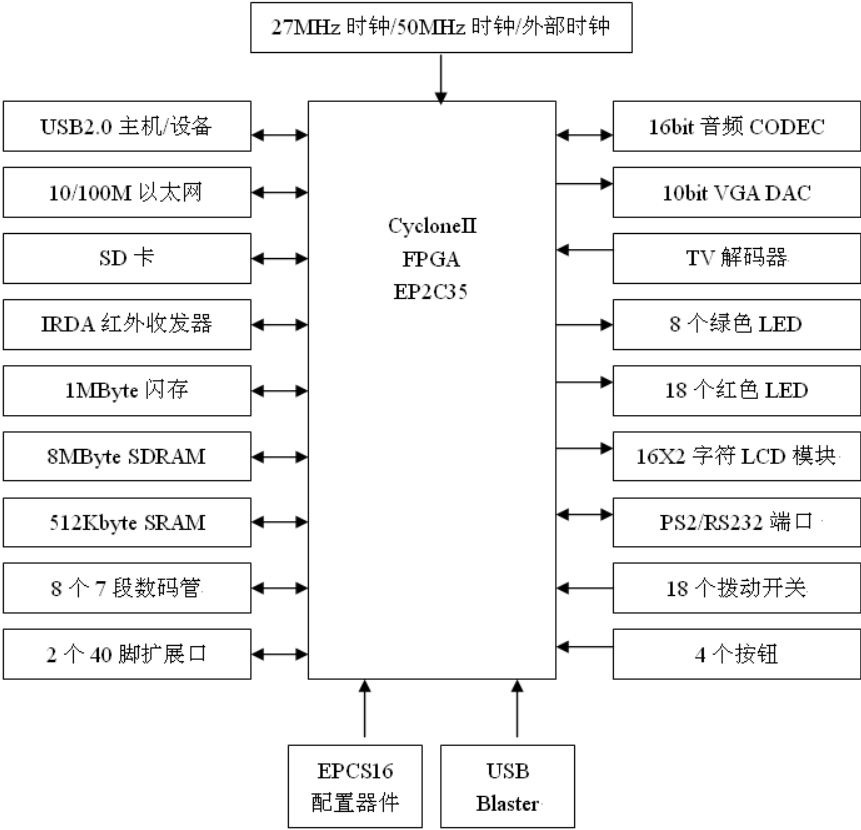
DE2 平台提供的主要资源有：

- ✧ Altera CycloneII 系列 FPGA 芯片 EP2C35F672C6 (U16)；
- ✧ 主动串行配置器件 EPCS16 (U30)；
- ✧ 编程调试接口 USB Blaster, 支持 JTAG 模式和 AS 模式, 其中 U25 是实现 USB Blaster 的 USB 接口芯片 FT245B; U26 为 CPLD 芯片 EPM3128, 用以实现 JTAG 模式和 AS 模式配置, 可以用 SW19 选择配置模式; USB Blaster 的 USB 口为 J9 ;
- ✧ 512K 字节 SRAM (U18)；
- ✧ 8M 字节 SDRAM (U17)；
- ✧ 1M 字节闪存 (U20)；
- ✧ SD 卡接口 (U19)；
- ✧ 4 个手动按钮 (KEY0-KEY3) 和 18 个拨动开关 (SW0-SW17)；
- ✧ 9 个绿色 LED (LEDG0-LEDG8) 和 18 个红色 LED (LEDR0-LEDR17)；
- ✧ 板上时钟源 (50MHz 晶振 Y1 和 27MHz 晶振 Y3), 外部时钟接口 (J5)；
- ✧ 音频编解码芯片 WM8371 (U1), 麦克风输入 (J1)、线路输入 (J2)、线路输出

- (J3);
- ✧ VGA 数模转换芯片 ADV7123 (U34), VGA 输出接口 (J13);
- ✧ TV 解码器 ADV7178B (U33), TV 接口 (J12);
- ✧ 10/100M 以太网控制器 DM9000AE (U35), 网络接口 (J4);
- ✧ USB 主从控制器 ISP1362 (U31), USB 主机接口 (J10), 设备接口 (J11);
- ✧ RS232 收发器 (U15), DB9 连接器 (J6);
- ✧ PS/2 鼠标/键盘连接器 (J7);
- ✧ IRDA 红外收发器 (U14);
- ✧ 带二极管保护的 40 针扩展口 (JP1、JP2);
- ✧ 2X16 字符 LCD 模块 (U2);
- ✧ 总电源开关 (SW18), 直流 9V 供电 (J8)。

## 2. DE2 实验平台结构

DE2 平台上包含了丰富的硬件接口，组成结构如附图 4-2 所示



附图 4-2 DE2 组成结构图

DE2 平台的核心是 Altera 公司的 FPGA 芯片 EP2C35F672, 该芯片是 Altera 公司 CycloneII 系列产品之一。EP2C35F672 采用 Fineline BGA 672 脚的封装，可以提供多达 475 个 I/O 引脚供使用者使用。图 B-2 中各个模块的具体功能见附表 4-1。

附表 4-1 模块功能表

名称	功能
----	----



Cyclone II 2C35 FPGA	33,216 逻辑单元、105 * M4K RAM 35个乘法器、4个同步逻辑器 475个I/O口、672脚BGA封装 串行配置设备和USB Blaster电路 Altera EPCS16 串行配置设备 支持JTAG 和AS配置模式
SRAM	512Kbyte SRAM 存储芯片 256K x 16bits 架构 可作为Nios II 处理器与DE2 控制面板的存储器
SDRAM	8-Mbyte 单数据传输率同步动态随机存储芯片 1M x 16bits x 4banks 构架 可作为Nios II 处理器与DE2 控制面板的存储器
Flash memory	4-Mbyte 或非门闪存、8 位数据总线 可作为Nios II 处理器与DE2 控制面板的存储器
SD卡插槽 (SD card socket)	提供SPI 模式对SD卡的访问 可作为Nios II 处理器与DE2 控制面板的存储器
按钮开关 (Pushbutton switches)	施密特触发电路反跳 通常高电平，按下时产生低电平脉冲
拨动开关 (Toggle switch)	18 个拨动开关作为用户输入 由按下转为弹起时产生逻辑0; 由弹起转为按下时产生逻辑1
晶振输入 (Clock inputs)	50-MHz 、27-MHz晶体振荡器 SMS 外部时钟输入
音频编码转换器 (Audio CODEC)	Wolfson WM87312 24位音频编码转换器 串行输入、串行输出、麦克风输入插孔 采样频率: 8~96KHz 适用于MP3 播放器、录音机、个人数字助理(PDA)、智能电话(smart phone)等
视频输出 (VGA output)	采用3路10位高速视频数/模转换器ADV7123(240MHz) 带有15 脚高密度D 形接口 支持100Hz 刷新率时的高达1600 x 1200 分辨率 可与Cyclone II FPGA 联合使用实现高性能TV 编码器
NTFC/PAL TV 解 码 电 路 (NTFC/PAL TV decoder circuit)	采用ADV7181B多格式SDTV视频解码器 支持NTFC、PAL、SECAM制式 集成三个54MHz 9位模/数转换器(ADC) 27MHz 晶体振荡器输入作为时钟源 支持复合视频(CVBS)RCA 接口输入 适用: DVD录像机(DVD recorders)、液晶电视(LCD TV)、机顶盒(Set-top-boxes)、数字电视(Digital TV)、带接口的数字设备
10/100Mb/s 以 太 网 控 制 器 (Ethernet controller)	集成带有一个总处理器接口的MAC 和PHY 支持100Base-T和10Base-T 支持带10 Mb/s和100Mb/s自动MDIX的全双工操作 支持IP/TCP/UDP校验求和操作与校验

	支持半双工流控制的后压方式
USB 主/从控制器	完全遵从USB2.0 规范、支持高速数据传输 支持USB主机和设备 并行接口、支持NiosII
串行口(Serial ports)	一个RS-232 口、一个PS/2 口 DB-9作为RS-232串口连接器 PS/2连接器连接PS2鼠标或键盘
红 外 端 口 收 发 器 (IrDA transceiver)	拥有一个115.2Kb/s 的红外收发器 32mA LED 驱动电流、集成电磁干扰屏蔽 IEC825-11级眼保护、边沿侦测输入
两个40 针扩展跳线	72 个Cyclone II I/O 口和8 只电源和地线端连接到两个40 针扩展跳线 按照可接插标准40 针IDE 硬驱动排线标准设计

### 3.DE2 平台上的引脚连接

DE2 平台上 FPGA 芯片 EP2C35F672 与外围各接口的引脚连接是固定不变的,其连接关系见附表 4-2~附表 4-13。

附表 4-2 开关与 FPGA 芯片的引脚连接表

开关引脚	芯片引脚	开关引脚	芯片引脚	开关引脚	芯片引脚
SW[0]	PIN_N25	SW[8]	PIN_B13	SW[16]	PIN_V1
SW[1]	PIN_N26	SW[9]	PIN_A13	SW[17]	PIN_V2
SW[2]	PIN_P25	SW[10]	PIN_N1	KEY[0]	PIN_G26
SW[3]	PIN_AE14	SW[11]	PIN_P1	KEY[1]	PIN_N23
SW[4]	PIN_AF14	SW[12]	PIN_P2	KEY[2]	PIN_P23
SW[5]	PIN_AD13	SW[13]	PIN_T7	KEY[3]	PIN_W26
SW[6]	PIN_AC13	SW[14]	PIN_U3		
SW[7]	PIN_C13	SW[15]	PIN_U4		

附表 4-3 LED 与 FPGA 芯片的引脚连接表

LED引脚	芯片引脚	LED引脚	芯片引脚	LED引脚	芯片引脚
LEDR[0]	PIN_AE23	LEDR[9]	PIN_Y13	LEDG[0]	PIN_AE22
LEDR[1]	PIN_AF23	LEDR[10]	PIN_AA13	LEDG[1]	PIN_AF22
LEDR[2]	PIN_AB21	LEDR[11]	PIN_AC14	LEDG[2]	PIN_W19
LEDR[3]	PIN_AC22	LEDR[12]	PIN_AD15	LEDG[3]	PIN_V18
LEDR[4]	PIN_AD22	LEDR[13]	PIN_AE15	LEDG[4]	PIN_U18
LEDR[5]	PIN_AD23	LEDR[14]	PIN_AF13	LEDG[5]	PIN_U17
LEDR[6]	PIN_AD21	LEDR[15]	PIN_AE13	LEDG[6]	PIN_AA20
LEDR[7]	PIN_AC21	LEDR[16]	PIN_AE12	LEDG[7]	PIN_Y18
LEDR[8]	PIN_AA14	LEDR[17]	PIN_AD12	LEDG[8]	PIN_Y12

附表 4-4 7 段数码管 HEX 与 FPGA 芯片的引脚连接表

HEX引脚	芯片引脚	HEX引脚	芯片引脚	HEX引脚	芯片引脚
HEX0[0]	PIN_AF10	HEX2[5]	PIN_AB25	HEX5[3]	PIN_T9
HEX0[1]	PIN_AB12	HEX2[6]	PIN_Y24	HEX5[4]	PIN_R5
HEX0[2]	PIN_AC12	HEX3[0]	PIN_Y23	HEX5[5]	PIN_R4

HEX0[3]	PIN_AD11	HEX3[1]	PIN_AA25	HEX5[6]	PIN_R3
HEX0[4]	PIN_AE11	HEX3[2]	PIN_AA26	HEX6[0]	PIN_R2
HEX0[5]	PIN_V14	HEX3[3]	PIN_Y26	HEX6[1]	PIN_P4
HEX0[6]	PIN_V13	HEX3[4]	PIN_Y25	HEX6[2]	PIN_P3
HEX1[0]	PIN_V20	HEX3[5]	PIN_U22	HEX6[3]	PIN_M2
HEX1[1]	PIN_V21	HEX3[6]	PIN_W24	HEX6[4]	PIN_M3
HEX1[2]	PIN_W21	HEX4[0]	PIN_U9	HEX6[5]	PIN_M5
HEX1[3]	PIN_Y22	HEX4[1]	PIN_U1	HEX6[6]	PIN_M4
HEX1[4]	PIN_AA24	HEX4[2]	PIN_U2	HEX7[0]	PIN_L3
HEX1[5]	PIN_AA23	HEX4[3]	PIN_T4	HEX7[1]	PIN_L2
HEX1[6]	PIN_AB24	HEX4[4]	PIN_R7	HEX7[2]	PIN_L9
HEX2[0]	PIN_AB23	HEX4[5]	PIN_R6	HEX7[3]	PIN_L6
HEX2[1]	PIN_V22	HEX4[6]	PIN_T3	HEX7[4]	PIN_L7
HEX2[2]	PIN_AC25	HEX5[0]	PIN_T2	HEX7[5]	PIN_P9
HEX2[3]	PIN_AC26	HEX5[1]	PIN_P6	HEX7[6]	PIN_N9
HEX2[4]	PIN_AB26	HEX5[2]	PIN_P7		

附表 4-5 SRAM 与 FPGA 芯片的引脚连接表

SRAM引脚	芯片引脚	SRAM引脚	芯片引脚	SRAM引脚	芯片引脚
SRAM_ADDR[0]	PIN_AE4	SRAM_ADDR[13]	PIN_W8	SRAM_DQ[8]	PIN_AE7
SRAM_ADDR[1]	PIN_AF4	SRAM_ADDR[14]	PIN_W10	SRAM_DQ[9]	PIN_AF7
SRAM_ADDR[2]	PIN_AC5	SRAM_ADDR[15]	PIN_Y10	SRAM_DQ[10]	PIN_AE8
SRAM_ADDR[3]	PIN_AC6	SRAM_ADDR[16]	PIN_AB8	SRAM_DQ[11]	PIN_AF8
SRAM_ADDR[4]	PIN_AD4	SRAM_ADDR[17]	PIN_AC8	SRAM_DQ[12]	PIN_W11
SRAM_ADDR[5]	PIN_AD5	SRAM_DQ[0]	PIN_AD8	SRAM_DQ[13]	PIN_W12
SRAM_ADDR[6]	PIN_AE5	SRAM_DQ[1]	PIN_AE6	SRAM_DQ[14]	PIN_AC9
SRAM_ADDR[7]	PIN_AF5	SRAM_DQ[2]	PIN_AF6	SRAM_DQ[15]	PIN_AC10
SRAM_ADDR[8]	PIN_AD6	SRAM_DQ[3]	PIN_AA9	SRAM_WE_N	PIN_AE10
SRAM_ADDR[9]	PIN_AD7	SRAM_DQ[4]	PIN_AA10	SRAM_OE_N	PIN_AD10
SRAM_ADDR[10]	PIN_V10	SRAM_DQ[5]	PIN_AB10	SRAM_UB_N	PIN_AF9
SRAM_ADDR[11]	PIN_V9	SRAM_DQ[6]	PIN_AA11	SRAM_LB_N	PIN_AE9
SRAM_ADDR[12]	PIN_AC7	SRAM_DQ[7]	PIN_Y11	SRAM_CE_N	PIN_AC11

附表 4-6 SDRAM 与 FPGA 芯片的引脚连接表

SDRAM引脚	芯片引脚	SDRAM引脚	芯片引脚	SDRAM引脚	芯片引脚
DRAM_ADDR[0]	PIN_T6	DRAM_BA_1	PIN_AE3	DRAM_DQ[8]	PIN_W6
DRAM_ADDR[1]	PIN_V4	DRAM_CAS_N	PIN_AB3	DRAM_DQ[9]	PIN_AB2
DRAM_ADDR[2]	PIN_V3	DRAM_CKE	PIN_AA6	DRAM_DQ[10]	PIN_AB1
DRAM_ADDR[3]	PIN_W2	DRAM_CLK	PIN_AA7	DRAM_DQ[11]	PIN_AA4
DRAM_ADDR[4]	PIN_W1	DRAM_CS_N	PIN_AC3	DRAM_DQ[12]	PIN_AA3
DRAM_ADDR[5]	PIN_U6	DRAM_DQ[0]	PIN_V6	DRAM_DQ[13]	PIN_AC2
DRAM_ADDR[6]	PIN_U7	DRAM_DQ[1]	PIN_AA2	DRAM_DQ[14]	PIN_AC1
DRAM_ADDR[7]	PIN_U5	DRAM_DQ[2]	PIN_AA1	DRAM_DQ[15]	PIN_AA5
DRAM_ADDR[8]	PIN_W4	DRAM_DQ[3]	PIN_Y3	DRAM_LDQM	PIN_AD2
DRAM_ADDR[9]	PIN_W3	DRAM_DQ[4]	PIN_Y4	DRAM_UDQM	PIN_Y5

DRAM_ADDR[10]	PIN_Y1	DRAM_DQ[5]	PIN_R8	DRAM_RAS_N	PIN_AB4
DRAM_ADDR[11]	PIN_V5	DRAM_DQ[6]	PIN_T8	DRAM_WE_N	PIN_AD3
DRAM_BA_0	PIN_AE2	DRAM_DQ[7]	PIN_V7		

附表4-7 Flash与FPGA芯片的引脚连接表

Flash引脚	芯片引脚	Flash引脚	芯片引脚	Flash引脚	芯片引脚
FL_ADDR[0]	PIN_AC18	FL_ADDR[12]	PIN_W16	FL_DQ[0]	PIN_AD19
FL_ADDR[1]	PIN_AB18	FL_ADDR[13]	PIN_W15	FL_DQ[1]	PIN_AC19
FL_ADDR[2]	PIN_AE19	FL_ADDR[14]	PIN_AC16	FL_DQ[2]	PIN_AF20
FL_ADDR[3]	PIN_AF19	FL_ADDR[15]	PIN_AD16	FL_DQ[3]	PIN_AE20
FL_ADDR[4]	PIN_AE18	FL_ADDR[16]	PIN_AE16	FL_DQ[4]	PIN_AB20
FL_ADDR[5]	PIN_AF18	FL_ADDR[17]	PIN_AC15	FL_DQ[5]	PIN_AC20
FL_ADDR[6]	PIN_Y16	FL_ADDR[18]	PIN_AB15	FL_DQ[6]	PIN_AF21
FL_ADDR[7]	PIN_AA16	FL_ADDR[19]	PIN_AA15	FL_DQ[7]	PIN_AE21
FL_ADDR[8]	PIN_AD17	FL_ADDR[20]	PIN_Y15	FL_RST_N	PIN_AA18
FL_ADDR[9]	PIN_AC17	FL_ADDR[21]	PIN_Y14	FL_WE_N	PIN_AA17
FL_ADDR[10]	PIN_AE17	FL_CE_N	PIN_V17		
FL_ADDR[11]	PIN_AF17	FL_OE_N	PIN_W17		

附表 4-8 LCD 与 FPGA 芯片的引脚连接表

LCD引脚	芯片引脚	LCD引脚	芯片引脚	LCD引脚	芯片引脚
LCD_RW	PIN_K4	LCD_DATA[2]	PIN_H1	LCD_DATA[7]	PIN_H3
LCD_EN	PIN_K3	LCD_DATA[3]	PIN_H2	LCD_ON	PIN_L4
LCD_RS	PIN_K1	LCD_DATA[4]	PIN_J4	LCD_BLON	PIN_K2
LCD_DATA[0]	PIN_J1	LCD_DATA[5]	PIN_J3		
LCD_DATA[1]	PIN_J2	LCD_DATA[6]	PIN_H4		

附表 4-9 VGA 与 FPGA 芯片的引脚连接表

VGA引脚	芯片引脚	VGA引脚	芯片引脚	VGA引脚	芯片引脚
VGA_R[0]	PIN_C8	VGA_G[2]	PIN_C10	VGA_B[4]	PIN_J10
VGA_R[1]	PIN_F10	VGA_G[3]	PIN_D10	VGA_B[5]	PIN_J11
VGA_R[2]	PIN_G10	VGA_G[4]	PIN_B10	VGA_B[6]	PIN_C11
VGA_R[3]	PIN_D9	VGA_G[5]	PIN_A10	VGA_B[7]	PIN_B11
VGA_R[4]	PIN_C9	VGA_G[6]	PIN_G11	VGA_B[8]	PIN_C12
VGA_R[5]	PIN_A8	VGA_G[7]	PIN_D11	VGA_B[9]	PIN_B12
VGA_R[6]	PIN_H11	VGA_G[8]	PIN_E12	VGA_CLK	PIN_B8
VGA_R[7]	PIN_H12	VGA_G[9]	PIN_D12	VGA_BLANK	PIN_D6
VGA_R[8]	PIN_F11	VGA_B[0]	PIN_J13	VGA_HS	PIN_A7
VGA_R[9]	PIN_E10	VGA_B[1]	PIN_J14	VGA_VS	PIN_D8
VGA_G[0]	PIN_B9	VGA_B[2]	PIN_F12	VGA_SYNC	PIN_B7
VGA_G[1]	PIN_A9	VGA_B[3]	PIN_G12		

附表 4-10 时钟、各接口与 FPGA 芯片的引脚连接表

接口引脚	芯片引脚	接口引脚	芯片引脚	接口引脚	芯片引脚
CLOCK_27	PIN_D13	TD_DATA[4]	PIN_G9	SD_DAT	PIN_AD24
CLOCK_50	PIN_N2	TD_DATA[5]	PIN_F9	SD_DAT3	PIN_AC23
EXT_CLOCK	PIN_P26	TD_DATA[6]	PIN_D7	SD_CMD	PIN_Y21

PS2_CLK	PIN_D26	TD_DATA[7]	PIN_C7	SD_CLK	PIN_AD25
PS2_DAT	PIN_C24	TD_HS	PIN_D5	AUD_ADCLRCK	PIN_C5
UART_RXD	PIN_C25	TD_VS	PIN_K9	AUD_ADCDATA	PIN_B5
UART_TXD	PIN_B25	TD_RESET	PIN_C4	AUD_DACLK	PIN_C6
TD_DATA[0]	PIN_J9	I2C_SCLK	PIN_A6	AUD_DACDATA	PIN_A4
TD_DATA[1]	PIN_E8	I2C_SDAT	PIN_B6	AUD_XCK	PIN_A5
TD_DATA[2]	PIN_H8	IRDA_TXD	PIN_AE24	AUD_BCLK	PIN_B4
TD_DATA[3]	PIN_H10	IRDA_RXD	PIN_AE25		

附表 4-11 USB 控制器与 FPGA 芯片的引脚连接表

USB引脚	芯片引脚	USB引脚	芯片引脚	USB引脚	芯片引脚
OTG_ADDR[0]	PIN_K7	OTG_DATA[0]	PIN_F4	OTG_DATA[10]	PIN_K6
OTG_ADDR[1]	PIN_F2	OTG_DATA[1]	PIN_D2	OTG_DATA[11]	PIN_K5
OTG_INT0	PIN_B3	OTG_DATA[2]	PIN_D1	OTG_DATA[12]	PIN_G4
OTG_INT1	PIN_C3	OTG_DATA[3]	PIN_F7	OTG_DATA[13]	PIN_G3
OTG_DACK0_N	PIN_C2	OTG_DATA[4]	PIN_J5	OTG_DATA[14]	PIN_J6
OTG_DACK1_N	PIN_B2	OTG_DATA[5]	PIN_J8	OTG_DATA[15]	PIN_K8
OTG_DREQ0	PIN_F6	OTG_DATA[6]	PIN_J7	OTG_CS_N	PIN_F1
OTG_DREQ1	PIN_E5	OTG_DATA[7]	PIN_H6	OTG_RD_N	PIN_G2
OTG_FSPEED	PIN_F3	OTG_DATA[8]	PIN_E2	OTG_WR_N	PIN_G1
OTG_LSPEED	PIN_G6	OTG_DATA[9]	PIN_E1	OTG_RST_N	PIN_G5

附表 4-12 网络接口与 FPGA 芯片的引脚连接表

网络接口引脚	芯片引脚	网络接口引脚	芯片引脚	网络接口引脚	芯片引脚
ENET_DATA[0]	PIN_D17	ENET_DATA[8]	PIN_B20	ENET_CLK	PIN_B24
ENET_DATA[1]	PIN_C17	ENET_DATA[9]	PIN_A20	ENET_CMD	PIN_A21
ENET_DATA[2]	PIN_B18	ENET_DATA[10]	PIN_C19	ENET_CS_N	PIN_A23
ENET_DATA[3]	PIN_A18	ENET_DATA[11]	PIN_D19	ENET_INT	PIN_B21
ENET_DATA[4]	PIN_B17	ENET_DATA[12]	PIN_B19	ENET_RD_N	PIN_A22
ENET_DATA[5]	PIN_A17	ENET_DATA[13]	PIN_A19	ENET_WR_N	PIN_B22
ENET_DATA[6]	PIN_B16	ENET_DATA[14]	PIN_E18	ENET_RST_N	PIN_B23
ENET_DATA[7]	PIN_B15	ENET_DATA[15]	PIN_D18		

附表 4-13 扩展 IO 与 FPGA 芯片的引脚连接表

扩展IO引脚	芯片引脚	扩展IO引脚	芯片引脚	扩展IO引脚	芯片引脚
GPIO_0[0]	PIN_D25	GPIO_0[24]	PIN_K19	GPIO_1[12]	PIN_R25
GPIO_0[1]	PIN_J22	GPIO_0[25]	PIN_K21	GPIO_1[13]	PIN_R24
GPIO_0[2]	PIN_E26	GPIO_0[26]	PIN_K23	GPIO_1[14]	PIN_R20
GPIO_0[3]	PIN_E25	GPIO_0[27]	PIN_K24	GPIO_1[15]	PIN_T22
GPIO_0[4]	PIN_F24	GPIO_0[28]	PIN_L21	GPIO_1[16]	PIN_T23
GPIO_0[5]	PIN_F23	GPIO_0[29]	PIN_L20	GPIO_1[17]	PIN_T24
GPIO_0[6]	PIN_J21	GPIO_0[30]	PIN_J25	GPIO_1[18]	PIN_T25
GPIO_0[7]	PIN_J20	GPIO_0[31]	PIN_J26	GPIO_1[19]	PIN_T18
GPIO_0[8]	PIN_F25	GPIO_0[32]	PIN_L23	GPIO_1[20]	PIN_T21
GPIO_0[9]	PIN_F26	GPIO_0[33]	PIN_L24	GPIO_1[21]	PIN_T20
GPIO_0[10]	PIN_N18	GPIO_0[34]	PIN_L25	GPIO_1[22]	PIN_U26

GPIO_0[11]	PIN_P18	GPIO_0[35]	PIN_L19	GPIO_1[23]	PIN_U25
GPIO_0[12]	PIN_G23	GPIO_1[0]	PIN_K25	GPIO_1[24]	PIN_U23
GPIO_0[13]	PIN_G24	GPIO_1[1]	PIN_K26	GPIO_1[25]	PIN_U24
GPIO_0[14]	PIN_K22	GPIO_1[2]	PIN_M22	GPIO_1[26]	PIN_R19
GPIO_0[15]	PIN_G25	GPIO_1[3]	PIN_M23	GPIO_1[27]	PIN_T19
GPIO_0[16]	PIN_H23	GPIO_1[4]	PIN_M19	GPIO_1[28]	PIN_U20
GPIO_0[17]	PIN_H24	GPIO_1[5]	PIN_M20	GPIO_1[29]	PIN_U21
GPIO_0[18]	PIN_J23	GPIO_1[6]	PIN_N20	GPIO_1[30]	PIN_V26
GPIO_0[19]	PIN_J24	GPIO_1[7]	PIN_M21	GPIO_1[31]	PIN_V25
GPIO_0[20]	PIN_H25	GPIO_1[8]	PIN_M24	GPIO_1[32]	PIN_V24
GPIO_0[21]	PIN_H26	GPIO_1[9]	PIN_M25	GPIO_1[33]	PIN_V23
GPIO_0[22]	PIN_H19	GPIO_1[10]	PIN_N24	GPIO_1[34]	PIN_W25
GPIO_0[23]	PIN_K18	GPIO_1[11]	PIN_P24	GPIO_1[35]	PIN_W23

### 参考文献

- 1.王鲁杨.电子技术实验指导书(第二版), 北京: 中国电力出版社, 2013 年.
- 2.王久和, 李春云.电工电子实验教程(第二版), 北京: 电子工业出版社出版社, 2013 年.
- 3.