这个小工程利用modelsim仿真波形图案来显示字符。效果如下：



用到的工具如下：

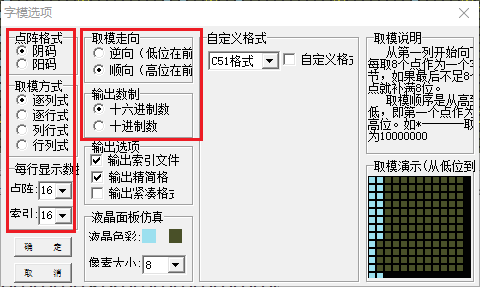
PCtoLCD2002 : 产生显示用的字符点阵

Quartus：代码编写，仿真文件生成。

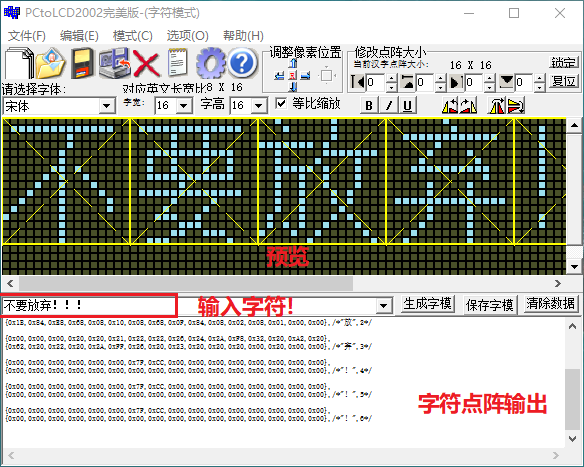
Modelsim：生成要显示的字符图案。

1. 字符点阵产生

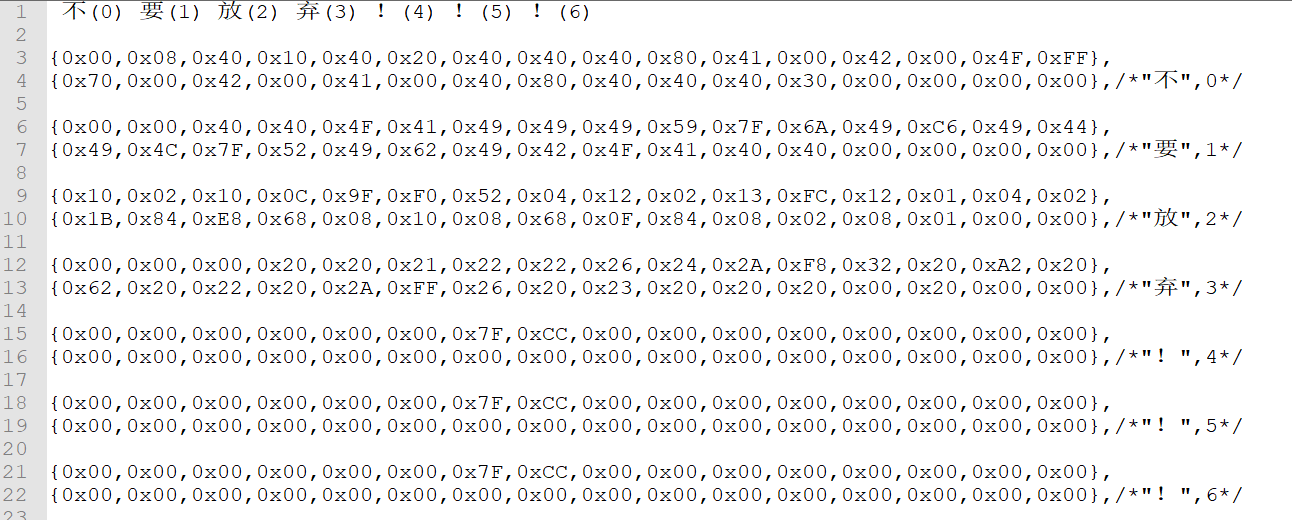
PCtoLCD2002端设置如下：



生成字符点阵：

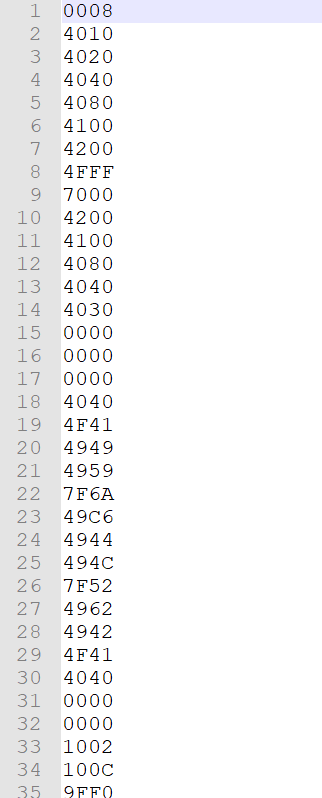


保持字模后，生成的字符点阵文件如下：

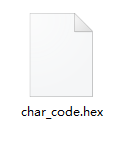


1. 字符点阵数据处理

将字符点阵文件中的提示信息，备注，标点符号等全部删除，只留下点阵的编码。并将相邻奇偶两行的数据调整到一行。效果如下：



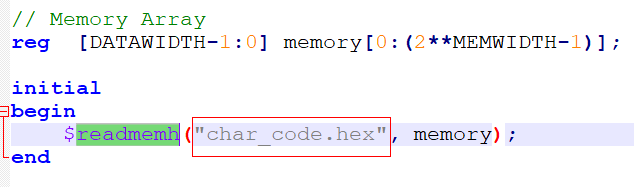
将处理后的字符文件重命名为 char\_code.hex



1. 代码编辑

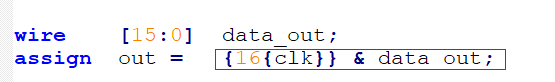
（1,）字符点阵数据储存

利用BRAM模块来储存字符点阵数据。利用 readmemh 函数加载字符点阵数据。



1. 波形生成

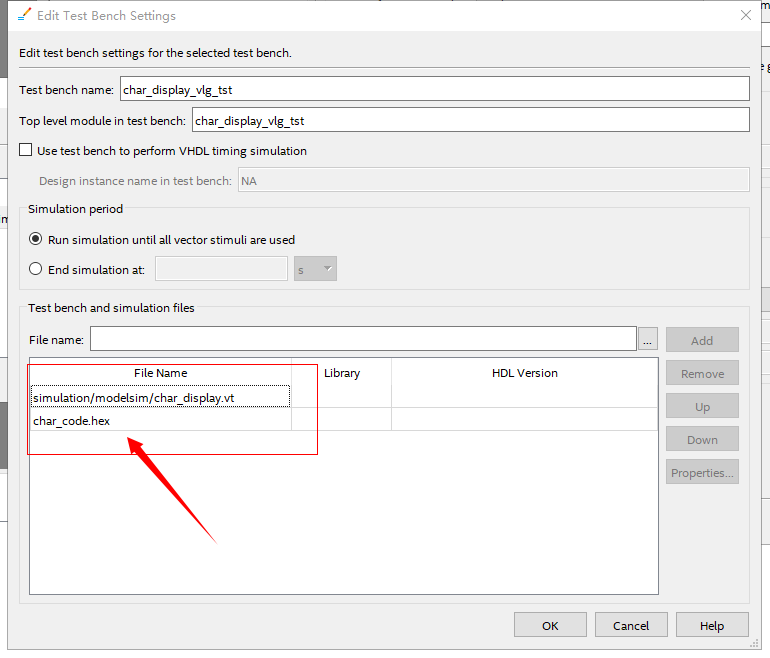
将从BRAM模块中读出的数据与时钟信号按位相与即可。



1. 仿真

编写仿真文件，运行仿真，展开 out 信号，缩放到合适比例，即可看到字符显示。效果即同文章开头所示。

【注】本工程是利用quartus设置的仿真环境，在仿真设置时，最好将 char\_code.hex 也添加进仿真文件列表中，仿真仿真软件找不到该文件。





更多细节请查看源代码！

1. 源代码

（1）char\_display 字符显示模块

module char\_display

(

input clk,

input rst\_n,

output [15:0] out

);

wire [15:0] data\_out;

reg [15:0] count;

wire [7:0] raddr = count[15:8];

assign out = {16{clk}} & data\_out;

BRAM BRAM\_inst

(

.clk ( clk ) , // input clk

.wr\_en ( 1'b0 ) , // input wr\_en

.waddr ( 1'b0 ) , // input [MEMWIDTH-1:0] waddr

.raddr ( raddr ) , // input [MEMWIDTH-1:0] raddr

.data\_in ( 8'b0 ) , // input [DATAWIDTH-1:0] data\_in

.data\_out ( data\_out ) // output [DATAWIDTH-1:0] data\_out

);

defparam BRAM\_inst.MEMWIDTH = 8;

defparam BRAM\_inst.DATAWIDTH = 16;

always@(posedge clk,negedge rst\_n)

begin

if(!rst\_n)

count <= 'b0;

else

count <= count + 1'b1;

end

Endmodule

（2）BRAM 模块

////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////

module BRAM

#(

parameter MEMWIDTH = 20, //1024 \* data

parameter DATAWIDTH = 1

)

(

input wire clk,

input wire wr\_en, //写有效信号

input wire [MEMWIDTH-1:0] waddr, //写地址

input wire [MEMWIDTH-1:0] raddr, //读地址

input wire [DATAWIDTH-1:0] data\_in, //写入的数据

output reg [DATAWIDTH-1:0] data\_out //读出的数据

);

// Memory Array

reg [DATAWIDTH-1:0] memory[0:(2\*\*MEMWIDTH-1)];

initial

begin

$readmemh("char\_code.hex", memory);

end

always@(posedge clk)

begin

if( wr\_en )

memory[waddr] <= data\_in;

data\_out <= memory[raddr];

end

endmodule

1. 仿真代码

`timescale 1 ps/ 1 ps

module char\_display\_vlg\_tst();

reg clk;

reg rst\_n;

wire [15:0] out;

char\_display i1 (

.clk(clk),

.out(out),

.rst\_n(rst\_n)

);

initial

begin

clk = 0;

rst\_n = 0;

#100 rst\_n = 1;

#1000000 $stop(2);

end

always #10 clk = ~clk;

endmodule