数字电路与数字系统实验

EX11:字符输入界面

191220029 傅小龙
周一 5-6 节班
1830970417@qq.com
2020 年 11 月 29 日

目录

一、实验内容

1.1 实验要求

实现一个可以用键盘输入，并在 VGA 显示器上回显的交互界面。界面实现要求可以参考 DOS 字符界面，Window 命令行或 Linux 的字符终端。

基本要求

- 支持所有小写英文字母和数字输入，以及不用 Shift 即可输入的符号。
- 一直按压某个键时，重复输出该字符。
- 输入至行尾后自动换行，输入回车也换行。

可选扩展要求

- 可以显示光标，建议可以用显示闪烁的竖线或横线作为光标。
- 支持 BackSpace 键删除光标前的字符。
- BackSpace 删除至本行开始后，再按 BackSpace 可以删除回车键，光标停留在上一行末尾的非空字符后。
- 支持自动滚屏，即输入到最后一行后回车出现新空白行，并且所有已输入的行自动上移一行。
- 支持 Shift 键以及大小写字符输入。
- 支持方向键移动光标。
- 在行首显示命令提示符。

感兴趣的同学还可以考虑如何实现彩色字符、绘制 ASCII 艺术图或实现类似 Matrix 开头的字符雨效果。

1.2 实验工具

软件环境：

设计、编译、仿真：Quartus Prime Version 17.1.0 Build 590 10/25/2017 SJ Lite Edition

DE10_Standard_SystemBuilder

硬件环境：DE-10 Standard 开发平台

FPGA 芯片：Cyclone V 5CSXFC6D6F31C6

二、实验过程

2.1 模型概述

在前面实验中已经实现的键盘、VGA 控制器模块的基础上，使用 ROM 存储器存储字模点阵，使用 RAM 作为显存存储键盘已输入对应字符的点阵. 可以用 mif 文件对显存初始化以显示 ASCII 字符图.

本次实验中实现的功能包括 1.1 节中的所有基础要求以及可选扩展要求中的：①可以显示光标，建议可以用显示闪烁的竖线或横线作为光标; ②支持 BackSpace 键删除光标前的字符; ③BackSpace 删除至本行开始后，再按 BackSpace 可以删除回车键，光标停留在上一行末尾的非空字符后; ④支持 Shift 键以及大小写字符输入; ⑤支持方向键移动光标。

2.2 数字抽象

下图给出了各模块间的关系及其作用，其中 exp11ch 为顶层文件名.

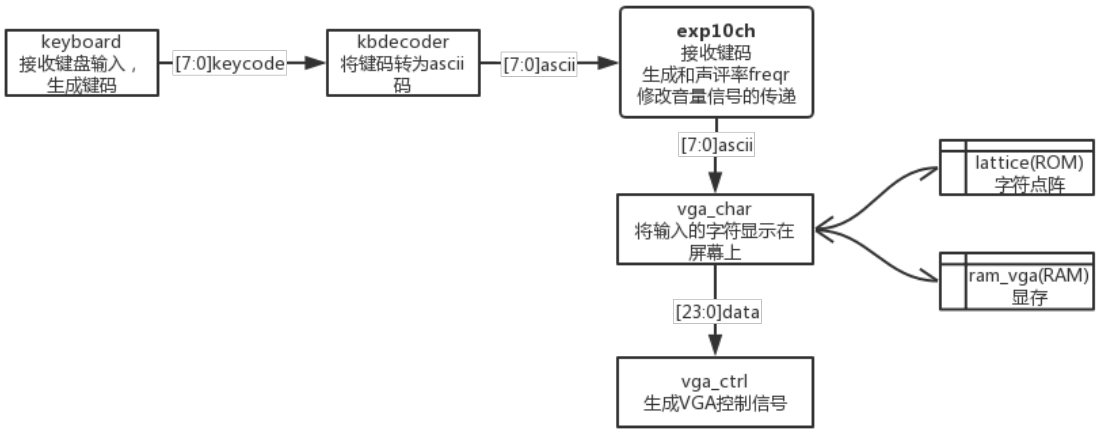


图 2-2-1 模块关系示意图

2.3 建立模型

2.3.1 vga_char 模块

vga_char 模块负责从只读存储器 lattice 中读取键盘输入对应的字符点阵并将之写入显存 RAM 以及光标闪烁色块的生成.

I)存储器的设置

由于每个字符的点阵大小为 16x9 像素点，共有 128 个字符，存放点阵的只读存储器 lattice 的规模设置为 4096x9bits，单口读，使用 IP 核生成，如图 2-3-1 所示. 对于分辨率为 640x480 像素的屏幕，可以显示 30 行 70 列的字符，每个单元存储 ASCII 码，故显存 RAM 的规模为 2100x8bits，使用双口读写，读写采用不同的时钟，使用 IP 核生成. 如图 2-3-2(a)(b)所示.

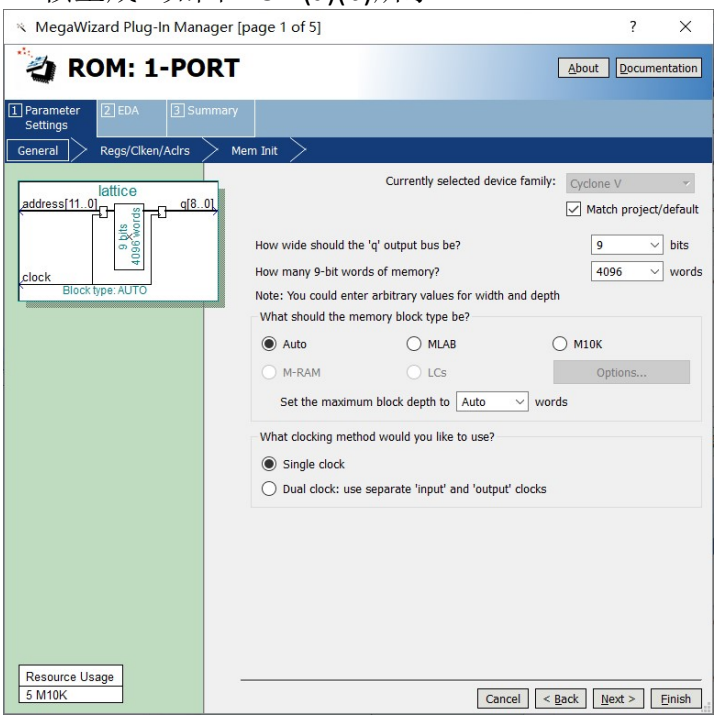


图 2-3-1 lattice 的设置

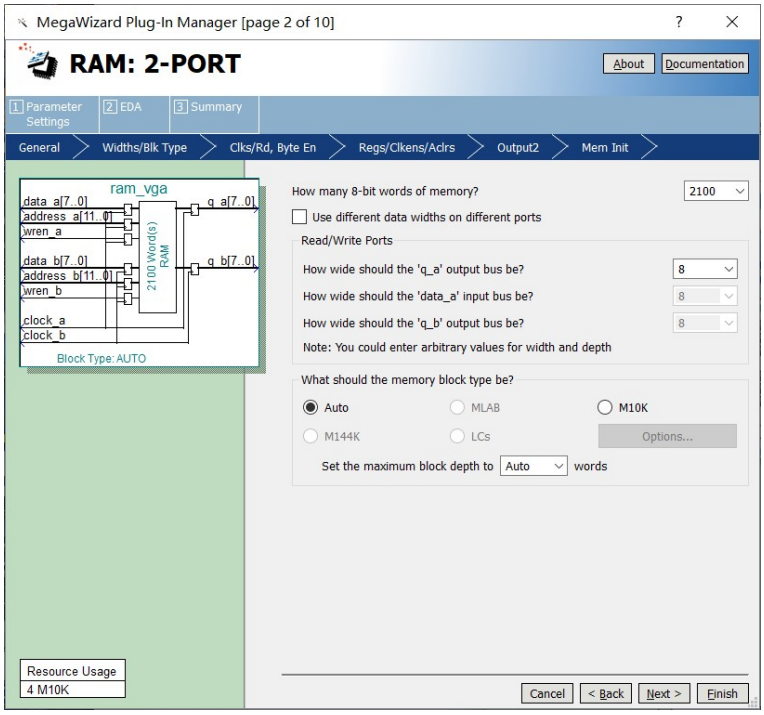


图 2-3-2（a） ram_vga 的设置

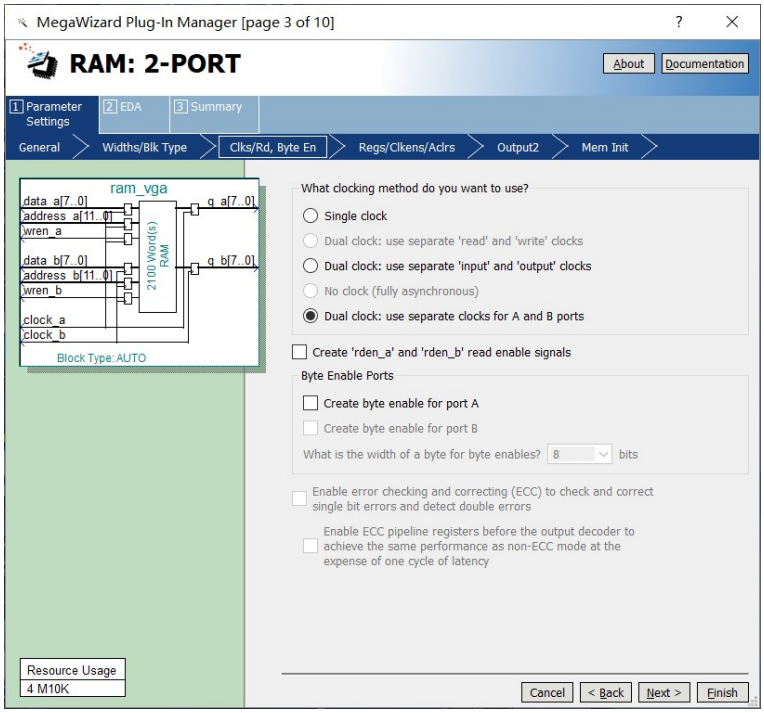


图 2-3-2（b） ram_vga 的设置

II)将键盘输入写入显存

使用频率较低的时钟（40Hz）来处理由顶层文件传入的 ascii 码. 通过 reg 型变量[3:0]cnt 来对该部分经过的 40Hz 低频时钟周期数进行计数, reg 型变量[11:0]index 用于记录当前输入的位置（对应屏幕上显示的第 index 个位置的 ascii 字符）, reg 型变量[11:0]ram_index[29:0]用于记录每一行（共 30 行）最后一个有效字符的位置，以便在删除字符时能够使 index 能够回到正确的位置.

有按键按下时 cnt 每经过一个 40Hz 时钟周期将会加一，达到 4'd15 时（0.375s）认为该键长按，将会持续对显存写入被按下的键码对应的 ascii 码. 若输入的 ascii 码为 0 则为无效操作，cnt 置 0. 对于功能型按键的输入不应该写入显存，比如说 Caps_Lock 键、Backspace 键等，故在遇到此类输入时需要将 ram_vga 的写使能 wen 置 0，遇到有实际显示意义的字符再将 wen 置 1. 考虑到器件可能存在的时序上的延迟, ram_vga 的写使能设置采用阻塞式赋值.

由上所述，ram_vga 的写地址为 index，写时钟为应为 40Hz 的时钟信号，写使能为 wen. 普通字符的输入需要将 index 加 1，并设置写使能为 1.

```
1   //add one char
2   wen = 1;
3   index <= index + 1;
```

若当前行输入已至行尾，则需要记录当前行的有效字符结束位置对应的 index.

```
1   if((index + 1) % 70 == 0)begin
2   //current line is to be full
3       ram_index[(index - 1) / 70] <= index;
4   end
```

若输入回车，则需要将 index 移动至下一行的起始位置并记录旧行的有效字符的结束位置.

```
1   if(ascii == 8'h0d)begin//hit enter
2       backspace_state <= 0;
```

```
         //store this line's ending position
        ram_index[index/70] <= index + 1;
        //move cursur to next line
         index <= index + 70 - (index % 70);
         enter_state <= 0;
end
```

若输入退格，则需要将 index 减 1 以起到删除字符的效果，当 index 退到行首时，需要将将 index 回退至 ram_index 中存储的上一行行尾位置对应的 index 值.

```
if (ascii == 8'h08) begin//hit backspace
    if(backspace_state == 0)begin
        index <= index - 1;
        backspace_state <= 1'b1;
    end
    else begin //backspace_state == 1'b0
        if(index % 70 == 0)begin
            //return to last line's end
            index <= preindex;
        end
        else begin
            //delete one char
        index <= index - 1;
        end
    end
end
```

其中, preindex 为 ram_index 中对应当前 index 上一行的行末位置.其取值如下：

```
always @(posedge clk_keyboard)begin
    if(index < 12'd70)begin
        preindex <= 0;
    end
    else begin
        preindex <= ram_index[(index/70)-1];
    end
end
```

若输入方向键，则将写使能置 0 并按相应方向修改 index 的值. ←键将 index 减 1, →键将 index 加 1, ↓键将 index 加 70 并记录旧行的行尾位置. ↑键将 index 减去 70. 为保证 index 的值始终在屏幕显示区域内，即 index 的值应满足 index < 2100. 故当 index 大于 2100 时将 index 置 0，故当输入到当前页面末尾后继续输入，光标将跳转置页面起始位置覆盖之前写的位置.

综合上面所述，给出将 ascii 码写入显存的对应 Verilog 语言实现如下:

```
 always @(posedge clk_keyboard) begin
    if(ascii != 0)
       begin
          if(cnt == 0 ) //short hit
          begin
```

```
6                   if(ascii == 8'h0d)begin//hit enter
7                       wen = 0;
8                       backspace_state <= 0;
9                       //store this line's ending position
10                      ram_index[index/70] <= index + 1;
11                      //move cursur to next line
12                      index <= index + 70 - (index % 70);
13                      enter_state <= 0;
14                  end
15                  else if (ascii == 8'h08) begin//hit backspace
16                          wen = 0;
17                          if(index % 70 == 0)begin
18                          //return to last line's end
19                          index <= preindex;
20                          end
21                          else begin
22                              //delete one char
23                              index <= index - 1;
24                              end
25                          end
26                      end
27                  else begin//hit key
28                          if((index + 1) % 70 == 0)begin
29                              //current line is to be full
30                              ram_index[(index - 1) / 70] <= index;
31                          end begin
32                              if(ascii == 8'h38) begin
33                                  //move cursur to last line
34                                  index <= index - 70;
35                                  wen = 0;
36                              end
37                              else if(ascii == 8'h32) begin
40                                  //move cursur to next line
41                                  //store this line's ending position
42                                  ram_index[index / 70] <= index + 1;
43                                  index <= index + 70;
44                                  wen = 0;
45                              end
46                              else if(ascii == 8'h34) begin
47                                  //left move cursur
48                                  index <= index - 1;
49                                  wen = 0;
50                              end
51                              else if(ascii == 8'h36) begin
```

```
52                          //right move  cursur
53                          index <= index + 1;
54                          wen = 0;
55                      end
56                      else if(ascii == 8'h14)begin
57                          //skip caps_lock
58                          wen = 0;
59                          index <= index;
60                      end
61                      else begin
62                          //add one char
63                          wen = 1;
64                          index <= index + 1;
65                      end
66                  end
67              end
68              backspace_state <= 0;
69              enter_state <= 1;
70          end
71          cnt <= cnt + 1;
72      end
73      else if(cnt == 4'd15)begin //long press
74              if(ascii == 8'h0d)  //enter
75                 index <= index + 70 - (index % 70) - 1;
76              else if (ascii == 8'h08)begin //backspace
77                  if(index % 70 == 0)begin
78             //this line has nothing left, ret to last line
79                      index <= preindex;
80                  end
81                  else begin
82                      //delete one char
83                      index <= index - 1;
84                  end
85              end
86             else begin
87                  if((index + 1) % 70 == 0)begin //end of line
88                      //store this line's ending position
89                      ram_index[(index-1)/70] <= index;
90                  end
91                  if(ascii == 8'h38) begin
92                        //move cursur to last line
93                        index <= index - 70;
94                        wen = 0;
95                    end
```

```
96                          else if(ascii == 8'h32) begin
97                              //move cursur to next line
98                              //store this line's position
99                              ram_index[index/70] <= index + 1;
100                             index <= index + 70;
101                             wen = 0;
102                         end
103                         else if(ascii == 8'h34) begin
104                             //left move cursur
105                             index <= index - 1;
106                             wen = 0;
107                         end
108                         else if(ascii == 8'h36) begin
109                             //right move  cursur
110                             index <= index + 1;
111                             wen = 0;
112                         end
113                         else if(ascii == 8'h14)begin
114                             //skip caps_lock
115                             wen = 0;
116                             index <= index;
117                         end
118                         else begin
119                             //add one char
120                             wen = 1;
121                             index <= index + 1;
122                         end
123                     end
124                 end
125             else begin //waiting if long press
126                 index <= index;
127                 cnt <= cnt + 1;
128             end
129             ascii_data <= ascii;
130         end
131     else begin //reset cnt
132         index <= index;
133           cnt <= 0;
134      end
135     if(index >= 2100) //return to the start of the page if
136 reaching the end
137         index <= 0;
138 end
```

III)从显存中读取已输入的字符并将之转化为点阵以及光标的显示

根据 vga_ctrl 模块提供的当前扫描点坐标([9:0]h_addr, [9:0]v_addr)确定正在扫描的字符位置[11:0]block_addr. 由前面所述的字符点阵大小可知这三者的关系满足

```
block_addr <= (v_addr >> 4) * 70 + ((h_addr) / 9);
```

[11:0]block_addr 即为 ram_vga 的读地址. ram_vga 的读输出由[7:0]ram_vga_ret 接收，[7:0]ram_vga_ret 即为当前扫描位置的字符的 ascii 码. 由该 ascii 码可算出该字符对应点阵在 lattice 中的（读）地址[11:0]address 应为

```
address <= (ram_vga_ret << 4) + (v_addr % 16);
```

lattice 的读输出由[8:0]font 接收, font 即为该字符在当前扫描行上的数据. 对应到扫描点上的数据应为 font[(h_addr + 2) % 9]. '+2'是对扫描位置的修正(否则将会造成最后一个字符的显示不完整). 若该扫描点对应的点阵数据为 0, 则传给 vga_ctrl 的 RGB 参数[23:0]data 为 24'h000000(黑色), 否则为全 1(白色).

光标的显示位置在当前输入位置 index 的下一格. 这里采用的显示区域为 `v_addr > (index / 70) * 16 && v_addr < (1 + index / 70) * 16 && h_addr > (index % 70 + 1) * 9 + 5 && h_addr < (index % 70 + 2` 需要注意的是若光标的显示区域过宽可能会遮挡住已经输入的字符. 光标的闪烁由一个低频时钟控制(5Hz).

综合上面所述, 显存读取部分以及光标的显示的 Verilog 实现如下:

```
1   always @(posedge clk_vga)begin
2       block_addr <= (v_addr >> 4) * 70 + ((h_addr) / 9);
3       address <= (ram_vga_ret << 4) + (v_addr % 16);
4
5       //show the cursor
6       //cursor is at the end of current line
7       if(v_addr > (index / 70) * 16 && v_addr < (1 + index / 70)
8   * 16
9           && h_addr > (index % 70 + 1) * 9 + 5 && h_addr < (index %
10  70 + 2) * 9)
11          if(clk_cursur) data <= 24'hffffff;
12          else data <= 24'h000000;
13
14      //show the font
15      else if(font[(h_addr + 2) % 9] == 1'b1)
16          data <= 24'hffffff;
17      else
18          data <= 24'h000000;
19
20      //current line: (index - 1)/70
21      //cursur pos: (index - 1) % 70
22  end
```

以上提到的任意频率的时钟为显示器实验中所使用的 clkgen 模块, 作用是将 50MHz 时钟信号转化为设置的频率. 其 Verilog 实现如下：

```
module clkgen(
input clkin,
input rst,
input clken,
output reg clkout
);
parameter clk_freq = 1000;
parameter countlimit = 50000000/2/clk_freq; // 自动计算计数次数

reg[31:0] clkcount;
always @ (posedge clkin)
if(rst) begin
   clkcount=0;
   clkout=1'b0;
end
else begin
if(clken)
   begin
      clkcount = clkcount + 1;
   if(clkcount >= countlimit) begin
      clkcount = 32'd0;
      clkout = ~clkout;
   end
   else
      clkout=clkout;
   end
   else begin
      clkcount=clkcount;
      clkout=clkout;
   end
end
```

2.3.2 vga_ctrl 模块

vga_ctrl 模块采用了 ex09 显示器实验中对 VGA 控制器的参考实现，故这里不再赘述. 主要功能是将 RGB 数据转化为 VGA 控制信号输出至显示器.

2.3.3 keyboard 模块

keyborad 模块采用了 ex08 PS2 键盘实验中给出的参考实现，故这里不再赘述. 主要功能是接收 PS2 键盘输入并将键码输出.

2.3.4 kbdecoder 模块

kbdecoder 模块中，相应键码的 ASCII 码用类似 ROM 的形式存放，与键码是一一对应关系. 再根据 shift, capslock 键的状态对 ASCII 码的输出进行赋值.

下面是 kbdecoder 模块的相关代码：

```
module kbdecoder(datain, dataout, shift_state, caps_state,
ctrl_state);
input [7:0]datain;
input shift_state; //shift 键是否被按下
input caps_state;  //caps 键是否被按下
input ctrl_state;  //ctrl 键是否被按下
output reg [7:0]dataout;

reg [7:0] asc [255:0];
reg [7:0] ascii_shift [255:0];
reg [7:0] ascii_caps [255:0];
reg [7:0] temp;

initial
begin
    $readmemh(".\\init\\ascii_init.txt", asc, 0, 255);
    $readmemh(".\\init\\ascii_init_shift.txt", ascii_shift,
0, 255);
    $readmemh(".\\init\\ascii_init_caps.txt", ascii_caps, 0,
255);

end

always @(*)
begin
    if(shift_state && !caps_state) begin
        dataout <= ascii_shift[datain];
    end
    else if(caps_state && !shift_state) begin
        dataout <= ascii_caps[datain];
    end
    else dataout <= asc[datain];
end
endmodule

```

初始化文件详见项目文件夹的 init/文件夹中的.txt 文件.

对于 shift, capslock 按键的状态设计和 next_data_n 信号的设置在顶层文件中给出如下实现：

```
always@(posedge CLOCK_50) begin
    if(ready == 1 && next_data_n == 1)begin
        temp <= keycode;
        next_data_n <= 0;
        if(keycode == 8'hf0) begin//realse
            release_flag <= 1;
            cnt <= cnt + 1;
        end
        else if(keycode == 8'h12 || keycode == 8'h59)
begin//shift
            if(release_flag) begin
                shift_state <= 0;
                e_out <= 0;
                release_flag <= 0;
            end
            else begin
                shift_state <= 1;
                e_out <= 1;
            end
        end
        else if(keycode == 8'h14) begin //ctrl
            if(release_flag) begin
                ctrl_state <= 0;
                e_out <= 0;
                release_flag <= 0;
            end
            else begin
                ctrl_state <= 1;
                e_out <= 1;
            end
        end
        else begin
            if(release_flag) begin
                e_out <= 0;
                release_flag <= 0;
                if(keycode == 8'h58) caps_state <= ~caps_state;
                else;
            end
            else e_out <= 1;
        end
    end
    else begin
        next_data_n <= 1;
    end
```

需要注意的是 keyboard 模块给出的键盘码输出并不一定是有效的，只有在 ready 和 next_data_n 信号都为 1 时才有效（Line2）.获取有效的键码输出后，需要将 next_data_n 信号置 0 以准备接受下一个有效信号.

对于 shift, ctrl 按键状态的设置：若 release_flag 为零，则说明是按键按下，将对应的标志信号置 1，否则置 0 并设置数码管的使能信号为 0(Line 9-31).

对于 caps_lock 按键状态的设置：caps_lock 键按下、松开后 caps_lock 按键的状态改变一次（Line 33-38）.

2.4 分析/综合

分析/综合实验成功，如下图所示：

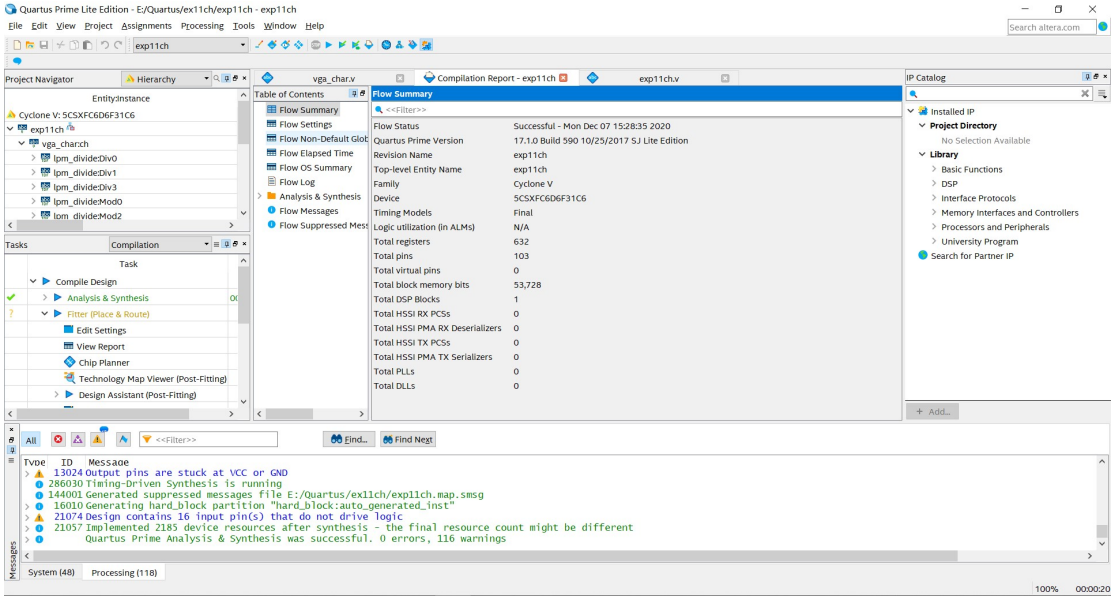


图 2-4-1：分析/综合成功

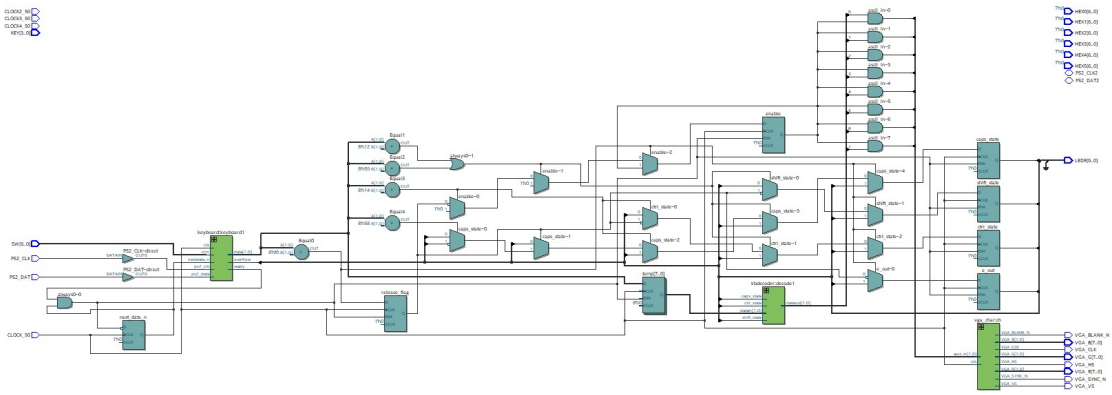
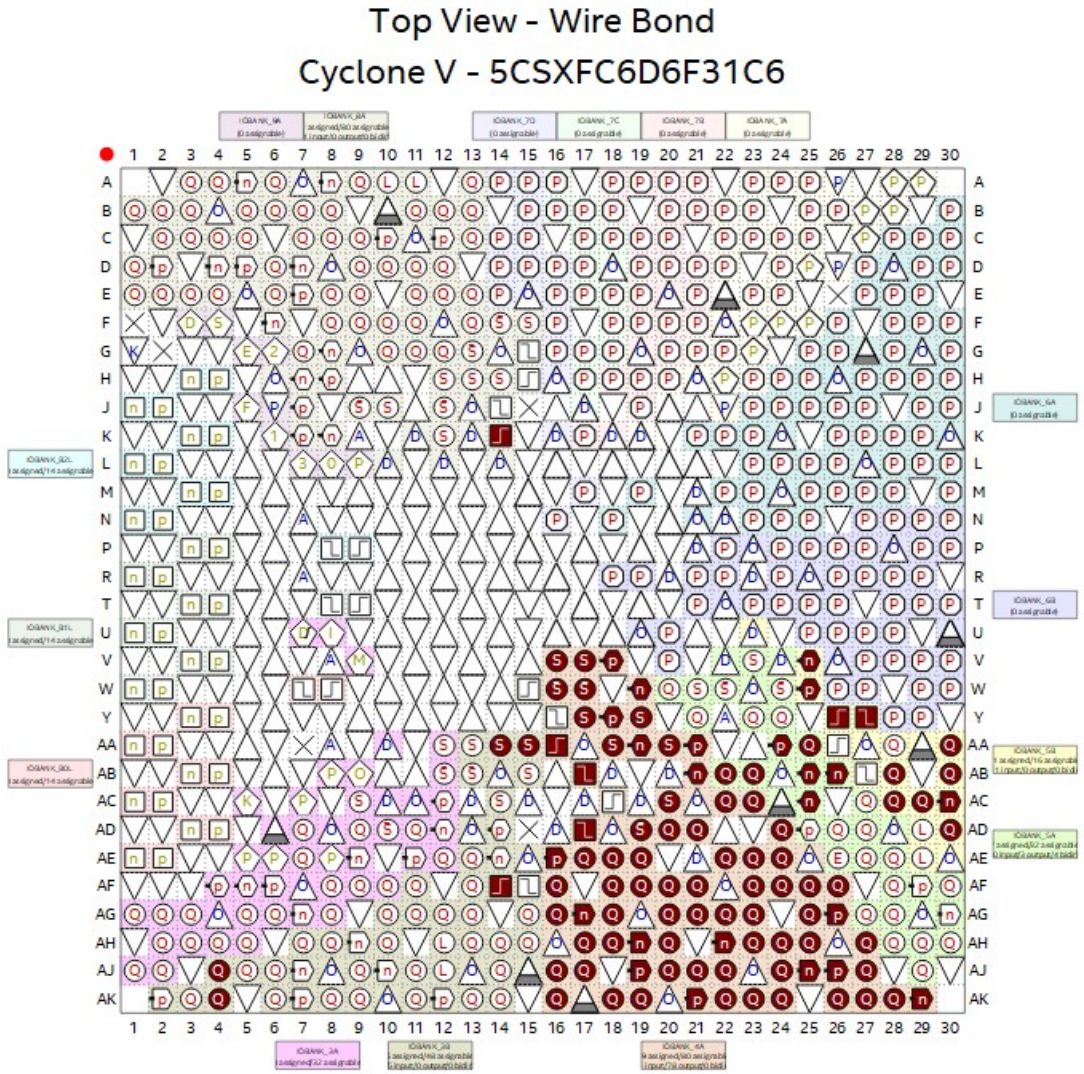


图 2-4-2:RTL 视图

2.5 分配引脚

引脚分配使用 DE10_Standard_SystemBuilder 生成。



CLOCK2_50	Input	PIN_AA16	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	
CLOCK3_50	Input	PIN_Y26	5B	B5B_N0	2.5 V	12mA ...ault)	
CLOCK4_50	Input	PIN_K14	8A	B8A_N0	3.3-V LVTTL	16mA ...ault)	
CLOCK_50	Input	PIN_AF14	3B	B3B_N0	3.3-V LVTTL	16mA ...ault)	
HEX0[6]	Output	PIN_AH18	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX0[5]	Output	PIN_AG18	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX0[4]	Output	PIN_AH17	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX0[3]	Output	PIN_AG16	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX0[2]	Output	PIN_AG17	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX0[1]	Output	PIN_V18	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX0[0]	Output	PIN_W17	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX1[6]	Output	PIN_V17	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX1[5]	Output	PIN_AE17	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX1[4]	Output	PIN_AE18	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX1[3]	Output	PIN_AD17	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX1[2]	Output	PIN_AE16	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX1[1]	Output	PIN_V16	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX1[0]	Output	PIN_AF16	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX2[6]	Output	PIN_W16	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX2[5]	Output	PIN_AF18	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX2[4]	Output	PIN_Y18	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX2[3]	Output	PIN_Y17	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX2[2]	Output	PIN_AA18	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX2[1]	Output	PIN_AB17	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX2[0]	Output	PIN_AA21	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX3[6]	Output	PIN_AD20	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX3[5]	Output	PIN_AA19	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)

HEX3[4]	Output	PIN_AC20	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX3[3]	Output	PIN_AA20	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX3[2]	Output	PIN_AD19	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX3[1]	Output	PIN_W19	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX3[0]	Output	PIN_Y19	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX4[6]	Output	PIN_AH22	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX4[5]	Output	PIN_AF23	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX4[4]	Output	PIN_AG23	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX4[3]	Output	PIN_AE23	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX4[2]	Output	PIN_AE22	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX4[1]	Output	PIN_AG22	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX4[0]	Output	PIN_AD21	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX5[6]	Output	PIN_AB21	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX5[5]	Output	PIN_AF19	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX5[4]	Output	PIN_AE19	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX5[3]	Output	PIN_AG20	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX5[2]	Output	PIN_AF20	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX5[1]	Output	PIN_AG21	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
HEX5[0]	Output	PIN_AF21	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
KEY[3]	Input	PIN_AA15	3B	B3B_N0	3.3-V LVTTL	16mA ...ault)	
KEY[2]	Input	PIN_AA14	3B	B3B_N0	3.3-V LVTTL	16mA ...ault)	
KEY[1]	Input	PIN_AK4	3B	B3B_N0	3.3-V LVTTL	16mA ...ault)	
KEY[0]	Input	PIN_AJ4	3B	B3B_N0	3.3-V LVTTL	16mA ...ault)	
LEDR[9]	Output	PIN_AC22	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[8]	Output	PIN_AB22	5A	B5A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[7]	Output	PIN_AF24	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[6]	Output	PIN_AE24	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[5]	Output	PIN_AF25	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[4]	Output	PIN_AG25	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[3]	Output	PIN_AD24	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[2]	Output	PIN_AC23	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[1]	Output	PIN_AB23	5A	B5A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[0]	Output	PIN_AA24	5A	B5A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
PS2_CLK	Bidir	PIN_AB25	5A	B5A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
PS2_CLK2	Bidir	PIN_AC25	5A	B5A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
PS2_DAT	Bidir	PIN_AA25	5A	B5A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
PS2_DAT2	Bidir	PIN_AB26	5A	B5A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
SW[9]	Input	PIN_AA30	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[8]	Input	PIN_AC29	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[7]	Input	PIN_AD30	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[6]	Input	PIN_AC28	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[5]	Input	PIN_V25	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[4]	Input	PIN_W25	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[3]	Input	PIN_AC30	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[2]	Input	PIN_AB28	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[1]	Input	PIN_Y27	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[0]	Input	PIN_AB30	5B	B5B_N0	2.5 V	12mA ...ault)	
VGA_B[7]	Output	PIN_AK16	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_B[6]	Output	PIN_AJ16	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_B[5]	Output	PIN_AJ17	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_B[4]	Output	PIN_AH19	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_B[3]	Output	PIN_AJ19	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_B[2]	Output	PIN_AH20	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_B[1]	Output	PIN_AJ20	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_B[0]	Output	PIN_AJ21	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_BLANK_N	Output	PIN_AK22	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_CLK	Output	PIN_AK21	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_G[7]	Output	PIN_AH23	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_G[6]	Output	PIN_AK23	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_G[5]	Output	PIN_AH24	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_G[4]	Output	PIN_AJ24	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_G[3]	Output	PIN_AK24	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_G[2]	Output	PIN_AH25	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_G[1]	Output	PIN_AJ25	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_G[0]	Output	PIN_AK26	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_HS	Output	PIN_AK19	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_R[7]	Output	PIN_AJ26	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_R[6]	Output	PIN_AG26	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_R[5]	Output	PIN_AF26	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_R[4]	Output	PIN_AH27	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_R[3]	Output	PIN_AJ27	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_R[2]	Output	PIN_AK27	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_R[1]	Output	PIN_AK28	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_R[0]	Output	PIN_AK29	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_SYNC_N	Output	PIN_AJ22	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
VGA_VS	Output	PIN_AK18	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)

图 2-5 引脚分配图

2.6 全编译

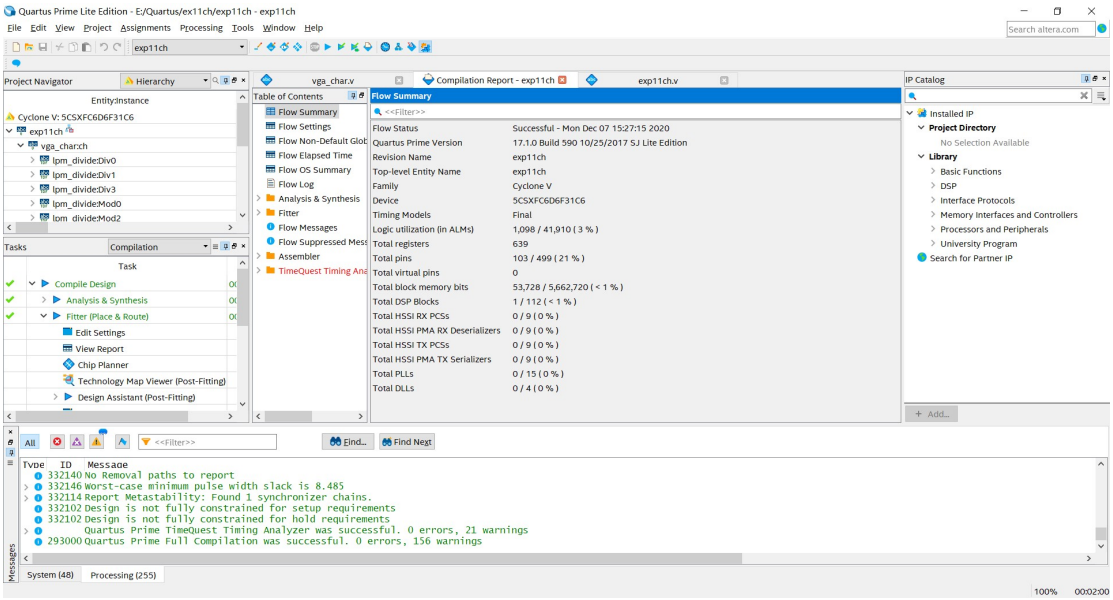


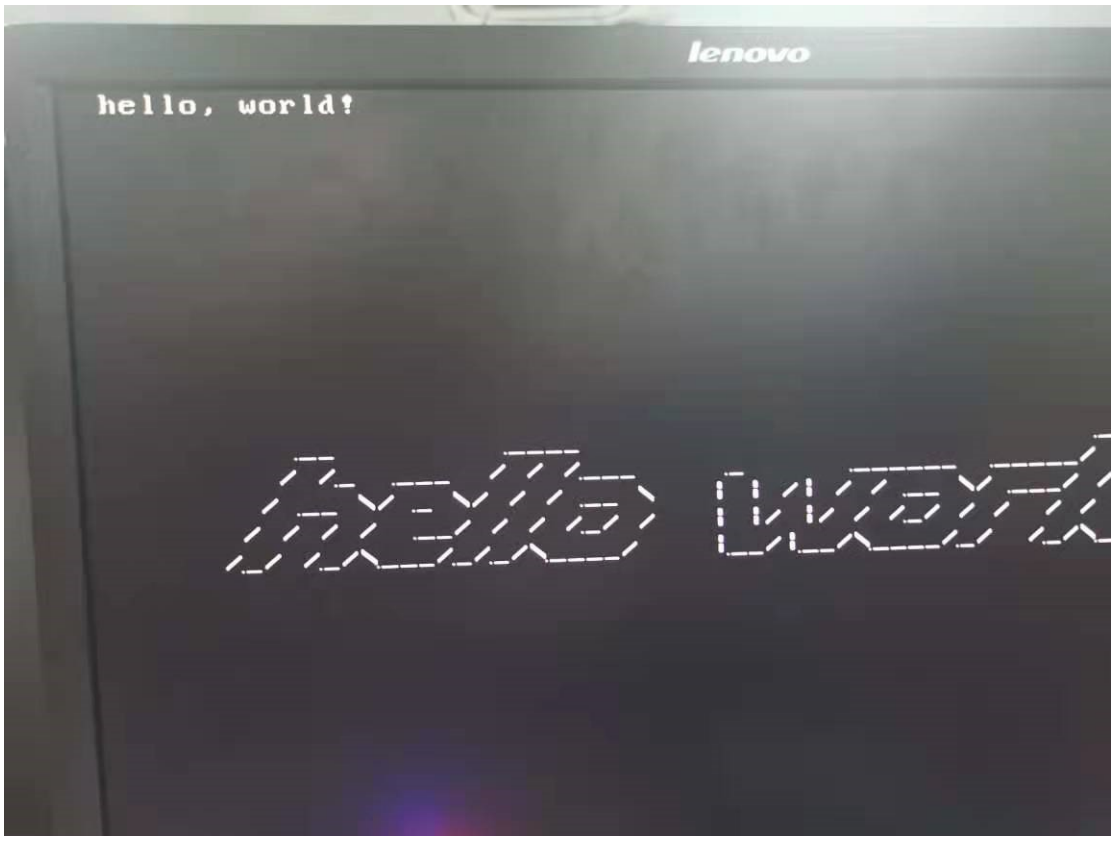
图 2-6 全编译成功

三、实验总结

本次实验设计了字符串输入界面. 结合之前实验中的 ROM、RAM 存储器, PS2 键盘，VGA 显示器，通过图 2-2-1 的方式与本实验中要设计的交互模块 vga_char 构成一个具有输入、显示字符功能的交互机器. 本次实验中需要注意的是对显存的读写的时序逻辑应分别与其对应的显示、字符输入功能的时序相一致. 显存的读写使能端的控制也应当与其相应的功能相符. 本次实验的实现中美中不足的是光标移动是不受当前输入的影响的, 即无法做到像文本编辑器一样限制光标移动的位置.

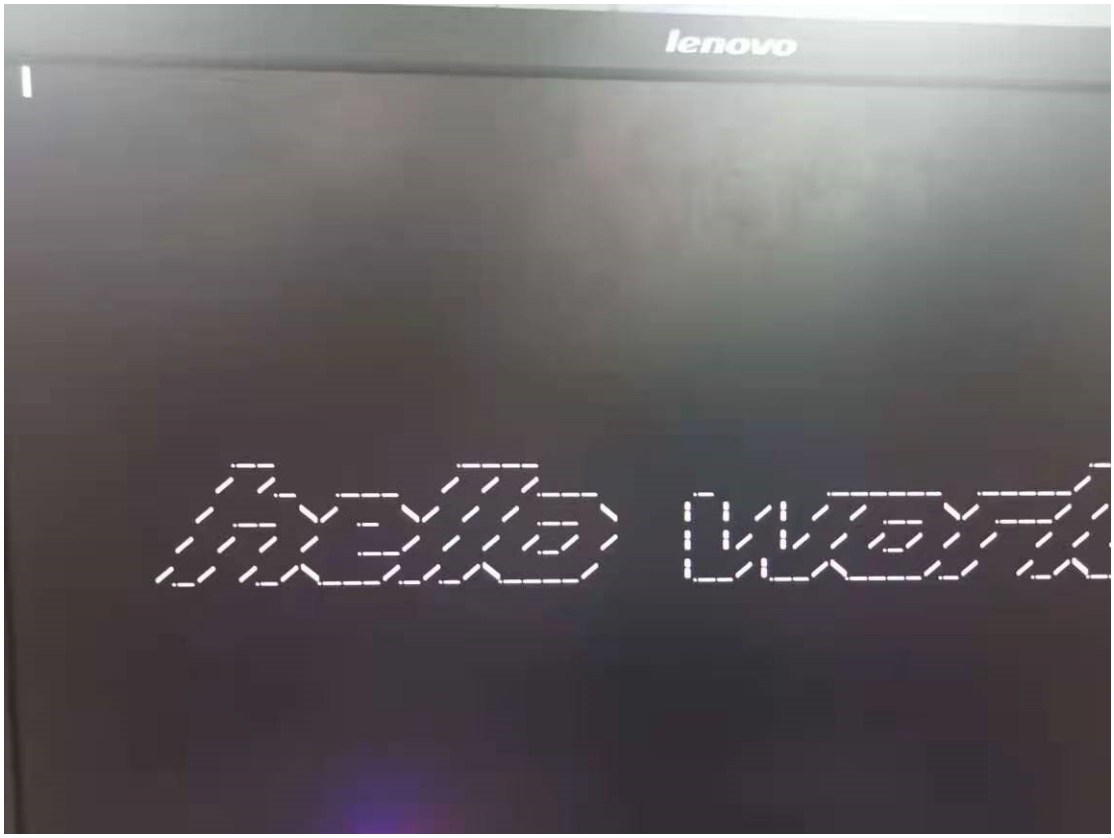
四、附

以下给出的是本实验的的图片：



附图 1 效果图 1

屏幕中央的艺术字”hello world!”字样由显存的 mif 初始化文件实现. 该 mif 文件在提交的项目文件夹中(./helloworld.mif).屏幕最上方的”hello, world!”字样由键盘输入. 由于拍摄原因附图 1 中并没有拍到光标. 故以附图 2 作为补充.



附图 1 效果图 2

由于长按效果不便以照片形式展现，这里不再赘述.