数字电路与数字系统实验

实验十一

简单字符交互

计算机科学与技术系

191220008 陈南曈

924690736@qq.com

2020.12.4

一、实验目的

本实验将利用前面实现过的键盘和显示器功能来搭建一个简单的字符输入界面，通过该系统的实现深入理解多个模块之间的交互和接口的设计。

实验内容：

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：

①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。
②一直按压某个键时，重复输出该字符。
③输入至行尾后自动换行，输入回车也换行。

可选扩展要求（只需要完成五种以上即可满分）：

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。
③ 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。
④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。
⑤支持 Shift 键以及大小写字符输入。
⑥支持方向键移动光标。
⑦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。

二、实验原理（知识背景）

1、字符显示

从之前的 VGA 图片显示实验，我们体会到图形界面需要大量的资源支 持，在 FPGA 上实现高精度、高分辨率的图形界面在资源上有些捉襟见肘。但 是，FPGA 上的资源比几十年前的第一代 PC 机要丰富许多。用 FPGA 来实现一 个简单的字符输入和显示界面并不难。 字符显示界面只在屏幕上显示 ASCII 字符，其所需的资源比较少。首先， ASCII 字符用 7bit 表示，共 128 个字符。大部分情况下，我们会用 8bit 来表示 单个字符，所以一般系统会预留 256 个字符。我们可以在系统中预先存储这 256 个字符的字模点阵，如下图所示。

图 11-1: ASCII 字符字模

这里每个字符高为 16 个点，宽为 9 个点。因此单个字符可以用 16 个 9bit 数来表示，每个 9bit 数代表字符的一行，对应的点为“1”时显示白色，为“0”时显示黑色。因此，我们只需要 256 × 16 × 9 ≈ 37kbit 的空间即可存储整个点阵。同学们可以自己用高级语言生成点阵存储文件。我们也提供了可通过 $readmemh 语句读取的点阵文本文件，其中每 3 个 16 进制数 (共 12bit) 表示单个字符的一行，该行的 9 个点中的最左边点在 12bit 中的最低位（请注意高低位顺序），然后依次类推，最高的 3 个 bit 始终为 0。每个字符 16 行，共 256 个字符。

例如，ASCII 字符“A”的编码是 41h（十进制 65）。因此其字模对应的地址是 16 × 65 = 1040（文本文件起始从 1 行开始，因此在第 1041 行）。以 A 字符的第 4 行为例，文件中存储的是 038h，二进制对应是 0000 0011 1000。最低位为 0，所以左边第一个像素为 0，左起第 4 到第 6 个像素为 1。如下图所示，此处为方便显示颜色是黑白颠倒的。

有了字符点阵后，系统就不再需要记录屏幕上每个点的颜色信息了，只需要记录屏幕上显示的 ASCII 字符即可。在显示时，根据当前屏幕位置，确定应该显示那个字符，再查找对应的字符点阵即可完成显示。对于 640×480 的屏幕，可以显示 30 行（30×16=480），70 列（70 × 9=630）的 ASCII 字符。系统的显存只需要 30 × 70 大小，每单元存储 8bit 的 ASCII 字符即可。这样，我们的字符显存只需要 2.1kByte，加上点阵的 6.144kByte，总共只需要不到 10kByte 的存储，FPGA 片上的存储足够实现了。

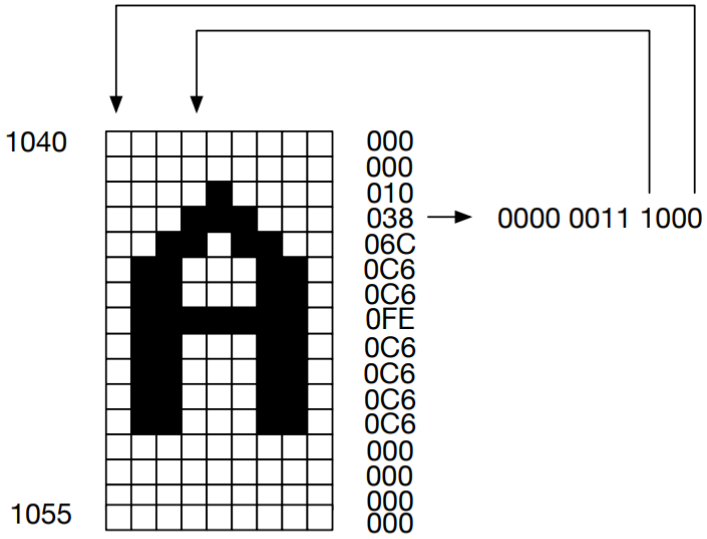


图 11-2: 字模 “A” 与存储器的关系

2、系统设计

（1）扫描显示

我们之前已经实现了 VGA 控制模块，该模块可以输出当前扫描到的行和列的位置信息，我们只需要稍加改动，即可让其输出当前扫描的位置对应 30×70 字符阵列的坐标 (0 ⩽ x ⩽ 69, 0 ⩽ y ⩽ 29)。利用该坐标，我们可以查询字符显存，获取对应字符的 ASCII 编码。利用 ASCII 编码，我们可以查询对应的点阵 ROM，再根据扫描线的行和列信息，可以知道当前扫描到的是字符内的哪个点。这时，可以根据该点对应的 bit 是 1 还是 0，选择输出白色还是黑色。

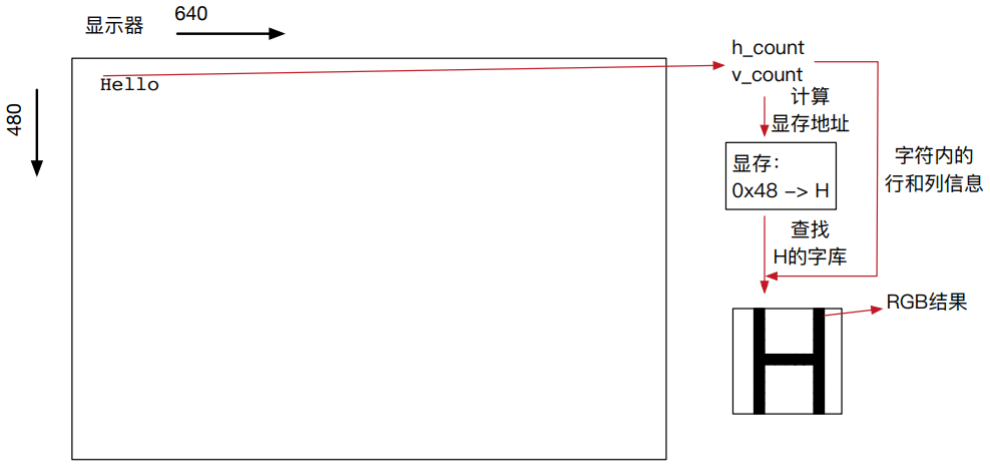


图 11-3: 字符显示流程示意图

我们将显示的过程总结如下：

1. 根据当前扫描位置，获取对应的字符的 x，y 坐标，以及扫描到单个字符点阵内的行列信息
2. 根据字符的 x，y 坐标，查询字符显存，获取对应 ASCII 编码
3. 根据 ASCII 编码和字符内的行信息，查询点阵 ROM，获取对应行的 9bit 数据
4. 根据字符内的列信息，取出对应的 bit，并根据该 bit 设置颜色。此处可以显示黑底白字或其他彩色字符，只需要按自己的需求分别设置背景颜色和字符颜色即可。

由于 VGA 的扫描频率是 25MHz，每个点扫过的时间非常短。因此，在扫过一个点的时间内要完成这一系列操作需要仔细地设计时序。首先，FPGA 在这么短的时间内是很难完成乘除法的。因此，设计时要合理选择存储器的大小 和寻址方式，将地址计算简化为简单的计数和逻辑操作。其次，对每个点我们要查询多个存储器，每个存储器的查询方式需要合理设计：是采用 RAM 还是 ROM？是上升沿读出还是下降沿读出？这些都有可能影响系统最终的性能。存在性能问题的显示有可能会出现彩色边缘或者模糊的现象，这些都说明系统实现上存在潜在问题。

- 如果用寄存器来实现所需的存储器而不用片上的 M10K 或 MLAB 有可能会占用大量资源，造成 FPGA 资源紧张，编译时间大大增加。

（2）显存读写

对于键盘输入，我们可以复用之前实现的键盘控制器。在键盘有输入的时候对字符显存进行改写，将按键对应的 ASCII 码写入显存的合适位置，这样输入就可以直接反馈到屏幕上了。

存储 ASCII 码的字符显存会经常被 VGA 扫描模块高速读取，而键盘模块需要对显存进行写入，需要注意两者的协调。

三、实验环境/器材等

1) 软件环境：

Quartus (Quartus Prime 17.1) Lite Edition

2) 硬件环境：

DE10-Standard 开发板
FPGA 部分：

Intel Cyclone V SE 5CSXFC6D6 F31C6N

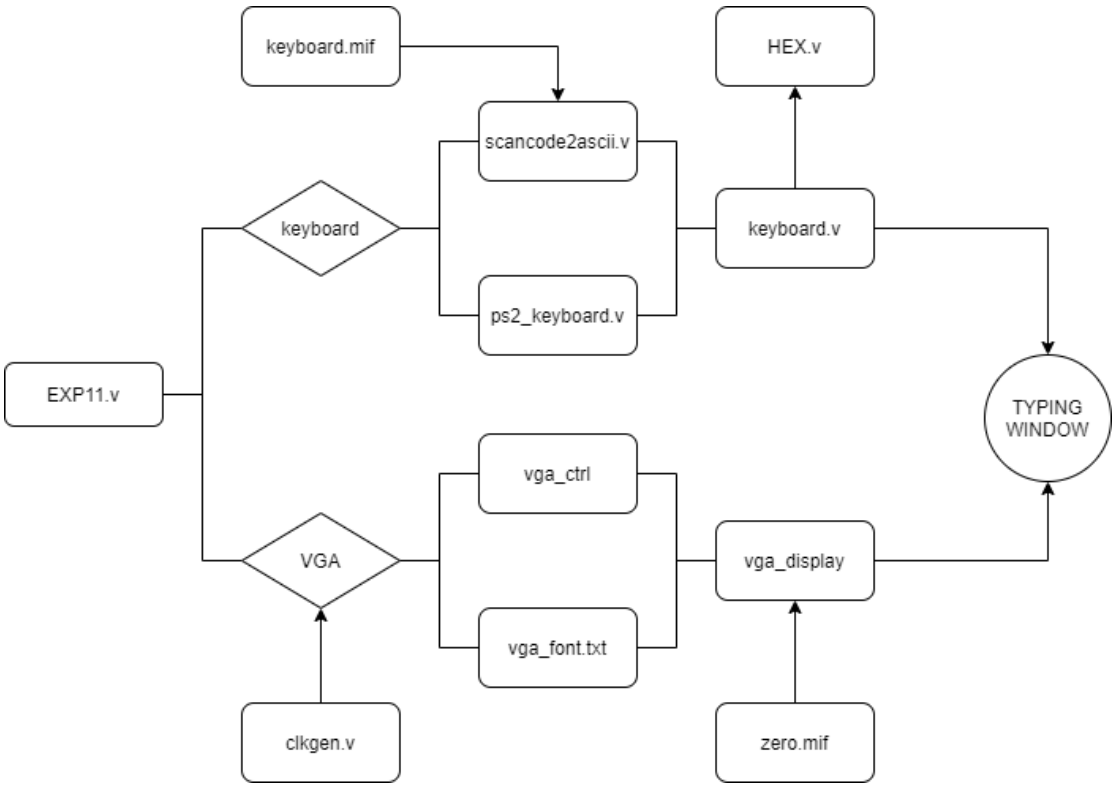
- 110K 逻辑单元
- 5,761Kbit RAM

HPS 部分：

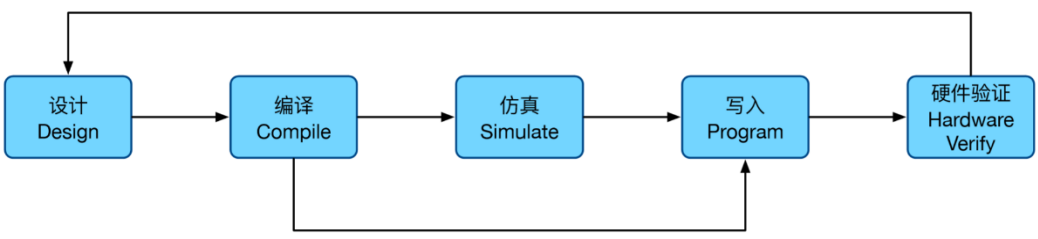
Dual-core ARM Cortex A9

- 925MHz
- 1GB DDR

四、程序代码或流程图



五、实验步骤/过程



设计：

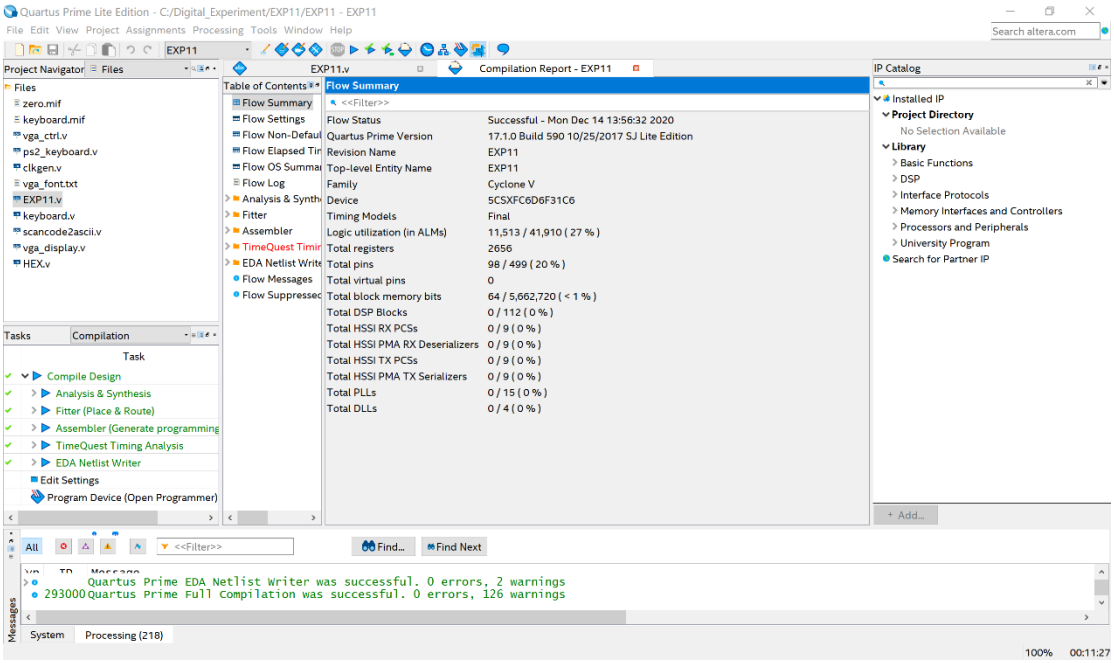
```
module EXP11(

   input                      CLOCK_50,
   input            [9:0]     SW,
   input            [3:0]     KEY,

   output           [9:0]     LEDR,

   output           [6:0]     HEX0,
   output           [6:0]     HEX1,
   output           [6:0]     HEX2,
   output           [6:0]     HEX3,
   output           [6:0]     HEX4,
   output           [6:0]     HEX5,

   output                     VGA_CLK,
   output           [7:0]     VGA_R,
   output           [7:0]     VGA_G,
   output           [7:0]     VGA_B,
   output                     VGA_HS,
   output                     VGA_VS,
   output                     VGA_SYNC_N,
   output                     VGA_BLANK_N,

   inout                      PS2_CLK,
   inout                      PS2_DAT

   );



   wire [11:0] vga_data;
   wire [7:0] vga_r;
   wire [7:0] vga_g;
   wire [7:0] vga_b;
   wire [9:0] h_addr;
   wire [9:0] v_addr;
   wire [7:0] data;
   wire [7:0] count;
   wire [7:0] ascii;
   wire [7:0] scancode;
   wire press_flag;
   wire [3:0] special_char;
   wire ready;
   wire nextdata_n;
   wire overflow;

   assign VGA_R = vga_r;
   assign VGA_G = vga_g;
   assign VGA_B = vga_b;

   initial
   begin
      VGA_SYNC_N = 0;
   end

ps2_keyboard my_ps2_keyboard(CLOCK_50, SW[0], PS2_CLK, PS2_DAT, data, ready, nextdata_
keyboard my_keyboard(CLOCK_50, SW[0], ready, data, nextdata_n, count, ascii, scancode,
HEX my_hex1(press_flag, scancode[3:0], HEX0);
HEX my_hex2(press_flag, scancode[7:4], HEX1);
clkgen #(.clk_freq(25000000)) my_clkgen(CLOCK_50, ~KEY[0], 1'b1, VGA_CLK);
vga_display my_vga_display(CLOCK_50, KEY[0], press_flag, special_char, ascill, h_addr,
vga_ctrl my_vga_ctrl(VGA_CLK, ~KEY[0], vga_data, h_addr, v_addr, VGA_HS, VGA_VS, VGA_B

endmodule
```

```
module vga_display(clk, clrn, press_flag, special_char, char_ascii, h_addr, v_addr, v

    input clk;
    input clrn;
    input [3:0] special_char;
    input [7:0] char_ascii;
    input [9:0] h_addr;
    input [9:0] v_addr;
    input press_flag;

    output reg [11:0] vga_data;//VGA颜色数据

    reg [29:0] line_flag;//是否显示该行
    reg flush_clk;//屏幕刷新时钟
    reg cursor_blink;//光标是否闪烁
    reg [6:0] now_x;//当前x坐标指针
    reg [6:0] now_y;//当前y坐标指针
    reg [6:0] now_max_x;//文末x坐标指针
    reg [6:0] now_max_y;//文末y坐标指针

    wire [9:0] char_x;//30×70矩阵x坐标
    wire [9:0] char_y;//30×70矩阵y坐标
    wire [3:0] content_x;//字模点阵x坐标
    wire [11:0] content_y;//字模点阵y坐标
    wire [11:0] matrix_line;//当前字模点阵行号
    wire [7:0] curr_char;//当前字符的acsii
    wire [1:0] curr_color;//当前字符颜色
    wire [9:0] cursor_area_h;//光标像素横坐标
    wire [9:0] cursor_area_v;//光标像素纵坐标

    reg [7:0] screen[0:29][0:69];//字符ascii矩阵
    reg [1:0] color[0:29][0:69];//字符颜色矩阵
    reg [11:0] const_color = 0;//当前字符实际颜色
    reg [11:0] vga_font[4095:0];//字模点阵
    reg [7:0] cmd[cmd_size - 1:0];//每行开头字符
```

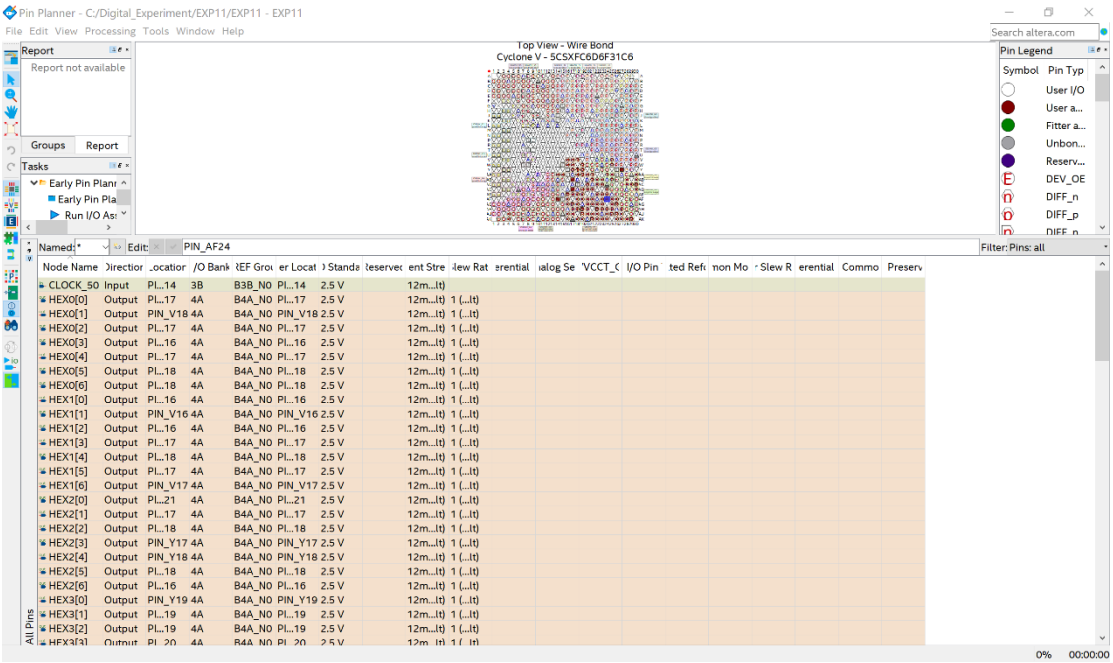
```
module keyboard(clk, clrn, ready, data, nextdata_n, count, ascii, scancode, press_fla

    input clk;
    input clrn;
    input ready;
    input [7:0] data;

    output reg nextdata_n;
    output reg [7:0] count = 0;
    output reg [7:0] ascii;
    output reg [7:0] scancode;
    output reg press_flag;
    output reg shift_flag;
    output reg ctrl_flag;
    output reg capslock_flag;
    output reg capital_flag;
    output reg [3:0] special_char;

    wire [7:0] ascii_copy;

    scancode2ascii my_ascii(data, ascii_copy);
    (* ram_init_file = "zero.mif" *) reg helper[255:0];

    reg predata = 1; //是否持续按键
    reg [7:0] const = 8'hE0;
    reg [6:0] count_clk = 0;
    reg clk_large = 0;
```

测试：无

编译：

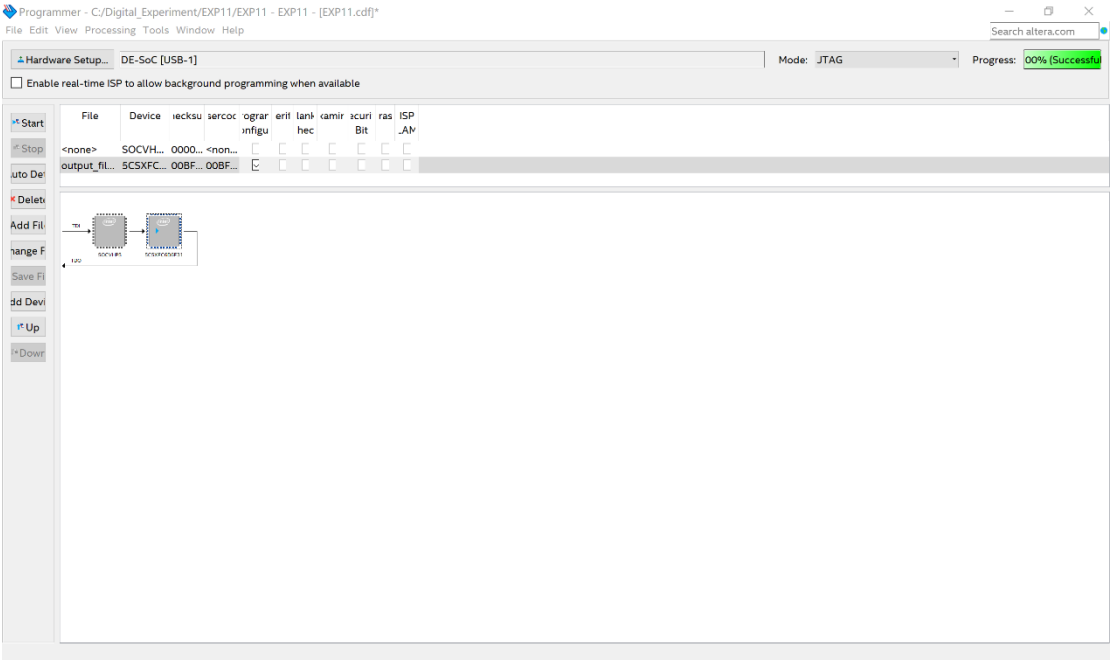

Quartus Prime Lite Edition - C:/Digital_Experiment/EXP11/EXP11 - EXP11
Flow Summary
Flow Status Successful - Mon Dec 14 13:56:32 2020
Quartus Prime Version 17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name EXP11
Top-level Entity Name EXP11
Family Cyclone V
Device 5CSXFC6D6F31C6
Timing Models Final
Logic utilization (in ALMs) 11,513 / 41,910 (27 %)
Total registers 2656
Total pins 98 / 499 (20 %)
Total virtual pins 0
Total block memory bits 64 / 5,662,720 (< 1 %)
Total DSP Blocks 0 / 112 (0 %)
Total HSSI RX PCSs 0 / 9 (0 %)
Total HSSI PMA RX Deserializers 0 / 9 (0 %)
Total HSSI TX PCSs 0 / 9 (0 %)
Total HSSI PMA TX Serializers 0 / 9 (0 %)
Total PLLs 0 / 15 (0 %)
Total DLLs 0 / 4 (0 %)
Quartus Prime EDA Netlist Writer was successful. 0 errors, 2 warnings
293000 Quartus Prime Full Compilation was successful. 0 errors, 126 warnings

引脚分配：

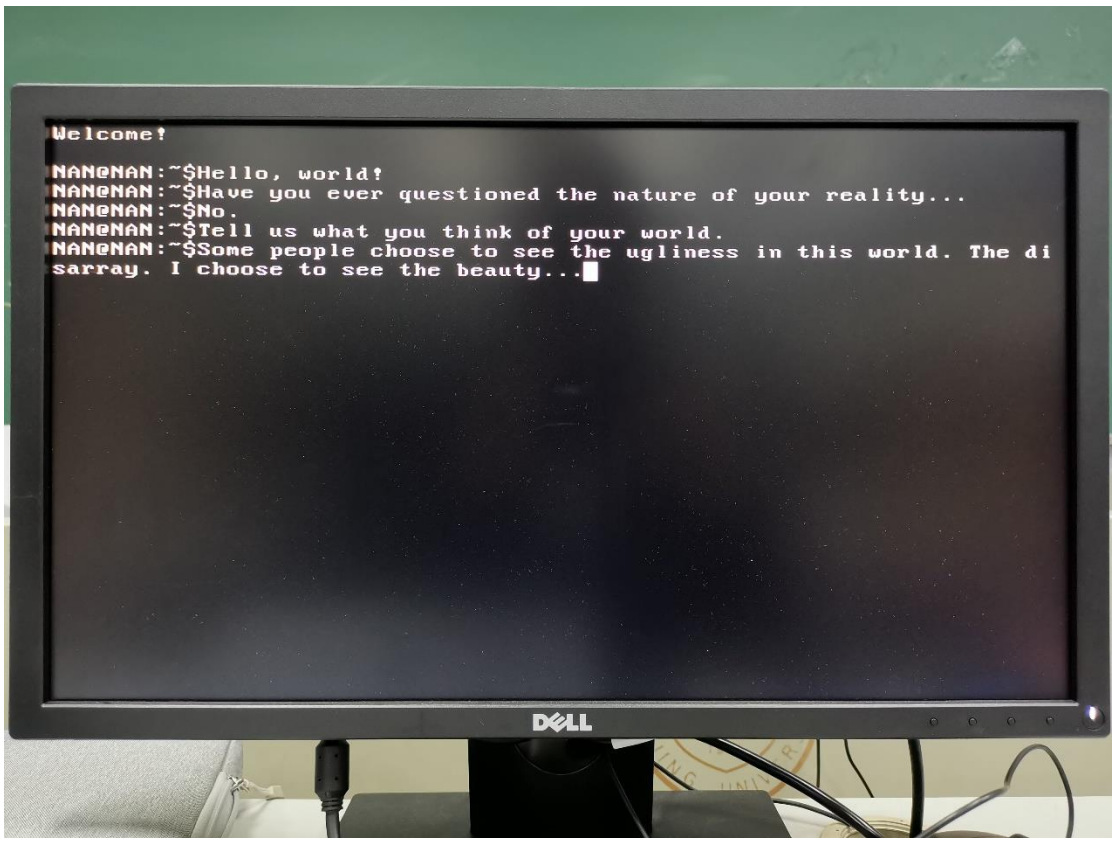

Pin Planner - C:/Digital_Experiment/EXP11/EXP11 - EXP11
Top View - Wire Bond
Cyclone V - 5CSXFC6D6F31C6

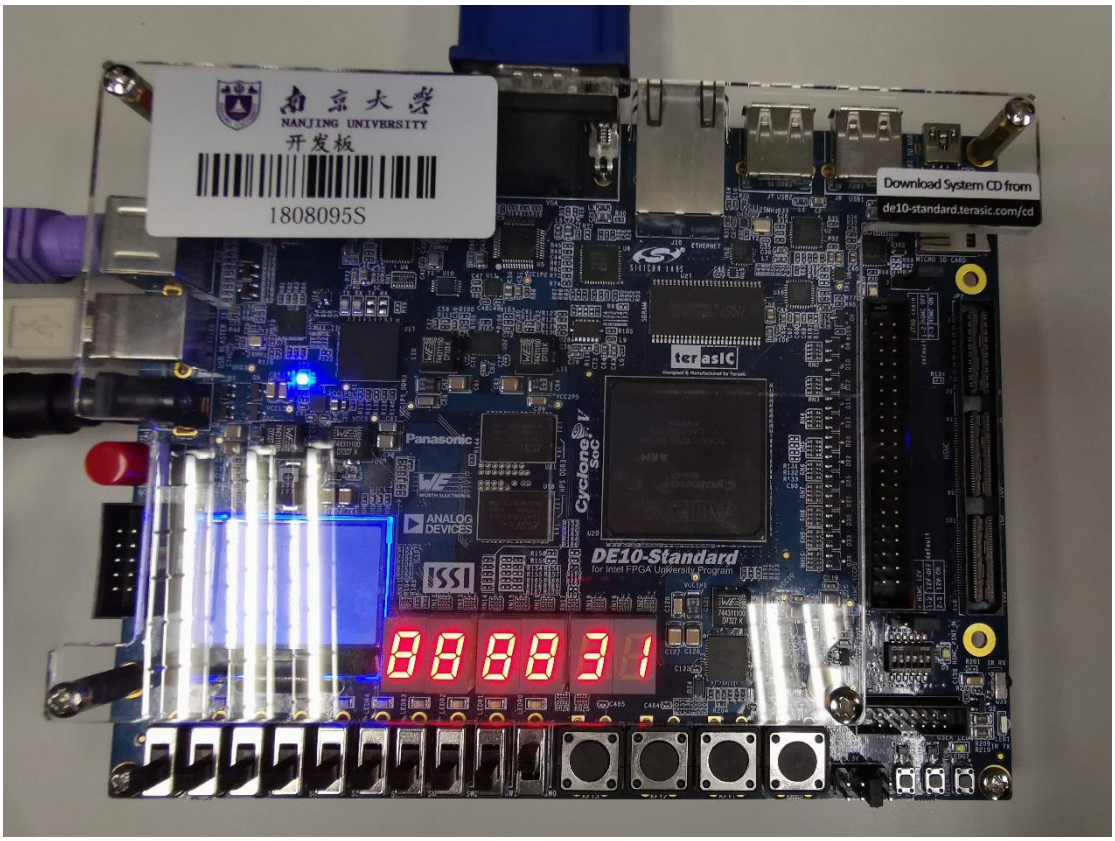
仿真：无

写入：


Programmer - C:/Digital_Experiment/EXP11/EXP11 - EXP11 - [EXP11.cdf]*
File Edit View Processing Tools Window Help
Search altera.com
Hardware Setup...
DE-SoC [USB-1]
Mode: JTAG
Progress:
00% (Successful
Enable real-time ISP to allow background programming when available
Start
Stop
Add Fil
Delete
Up
File
Device
Bit
<none>
SOCVH...
output_fil...
5CSXFC...

硬件验证：


Welcome!
NAN@NAN:~$Hello, world!
NAN@NAN:~$Have you ever questioned the nature of your reality...
NAN@NAN:~$No.
NAN@NAN:~$Tell us what you think of your world.
NAN@NAN:~$Some people choose to see the ugliness in this world. The di
sarray. I choose to see the beauty...
DELL



六、测试方法

该实验无法使用 Test Bench 进行仿真。
故采用直接通过屏幕显示情况反馈的字符显示、显示时序等来判断可能存在的 bug。

七、实验结果

经过很多次的硬件导入调试，最终使各个部分的小功能均满足预期效果。

总体上实现了一个可以用键盘输入，并在 VGA 显示器上回显的交互界面。交互界面与操作系统终端接面相类似，有命令行和欢迎界面。

支持所有小写英文字母、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。输入至行尾后可以自动换行，输入回车也能换行。

此外，还可以显示光标，用白色长方形方块作为闪烁光标，与 linux 终端光标效果类似。支持 BackSpace 键删除光标前的字符。且 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。可以用左右方向键移动光标。在行首显示命令提示符“NAN@NAN：~$”。

八、实验中遇到的问题及解决办法

1、本次实验工程规模较大，多出需要使用 ROM 或者 RAM，以及许多对大型二维数组的赋值，导致每一次在硬件的验证前都需要进行较长时间的编译，比较浪费时间。

解决办法：

①对于一些大型的数组，采用 mif 或 txt 写入的方式进行赋值，避免通过逐一赋值或循环赋值，而浪费较多时间。

例如：

```
//读入字模点阵
$readmemh("C:/Digital_Experiment/EXP11/vga_font.txt", vga_font, 0, 4095);
```

```
        (* ram_init_file = "zero.mif" *) reg helper[255:0];
```

```
  (* ram_init_file = "keyboard.mif" *) reg [7:0] ascii_rom[255:0];
```

②在 FPGA 中要计算乘除法操作耗费资源较多，且在这么短的时间内是很难完成乘除法，故需要换成移位、拼接和加减的方式，避免不必要的乘除法。

例如：

```
  assign cursor_area_h = {now_x[6:0], 3'b0} + now_x;//光标像素横坐标
```

```
      assign cursor_area_v = {now_y[5:0], 4'b0};//光标像素纵坐标
```

2、屏幕上正常显示欢迎界面和命令行，但输入字符时不显示字符。

解决办法：键盘输入时不显示字符，初步判断是键盘实现部分的时序逻辑出现问题，于是对 ps2_keyboard.v 和 keyboard.v 的代码进行了仔细的检查。但是，最终并未发现问题。因此，在无法确定问题所在的情况下，对所有代码进行了排查，最终发现，是 EXP11.v 中调用模块是传入的参数名字有误，“press_flag”打成了“pres_flag”，另一处“ascii”打成了“asci”，但这样的问题很难被发现，因为变量名错误的情况在编译时无法被自动检查出来，特别是当工程规模较大时，很难用人眼排查出来。

3、交互界面个功能正常显示，但屏幕中出现一些彩色的不规则线条。

解决办法：对每个点我们要查询多个存储器，每个存储器的查询方式需要合理设计：是采用 RAM 还是 ROM？是上升沿读出还是下降沿读出？这些都有可能影响系统最终的性能。存在性能问题的显示有可能会出现彩色边缘或者模糊的现象，这些都说明系统实现上存在潜在问题。因此，对于每个存储器的存取要规范，不要出现数组越界等非法操作。

4、键盘输入时，按一次显示多个字符或不显示字符。

解决办法：存储 ASCII 码的字符显存会经常被 VGA 扫描模块高速读取，而键盘模块需要对显存进行写入，需要注意两者的协调。因此设计时钟频率和读取时序时需要实现两者的统一，否则会造成部分时刻无法读取或读取延迟的情况。此外，键盘的时序逻辑是基础，必须保证其正确性。

5、字符显示位置与预期位置不同。

解决办法：当光标位置因删除、回车、左移、右移等操作而发生变化时，再次从键盘输入时应当从正确的位置开始显示。因此除了当前字符位置的指针 now_x 和 now_y 以外，还应该用另外的指针 now_max_x 和 now_max_y 保存文末字符的位置

6、命令行和非命令行的实现，以及左移右移光标。

解决办法：想要实现命令提示符和左移右移，不仅仅是将其显示出来，还需要一些实现逻辑上的改变。比如：删除到行首是不能删除命令提示符；回车换行后需要显示下一行命令提示符；输入多行字符时不需要显示命令提示符；等等。因此，上一个问题中使用的当前字符位置指针和文末位置指针就尤为重要，保证了命令提示符和左移右移光标的正确显示。

九、实验得到的启示

1、为了实现字符交互的功能，需要对键盘输入和显示器的实现有较为良好的理解，才能在本实验运用得当，不然极容易出现时序上的错误，导致字符无法输出或者显示器无法显示的情况。

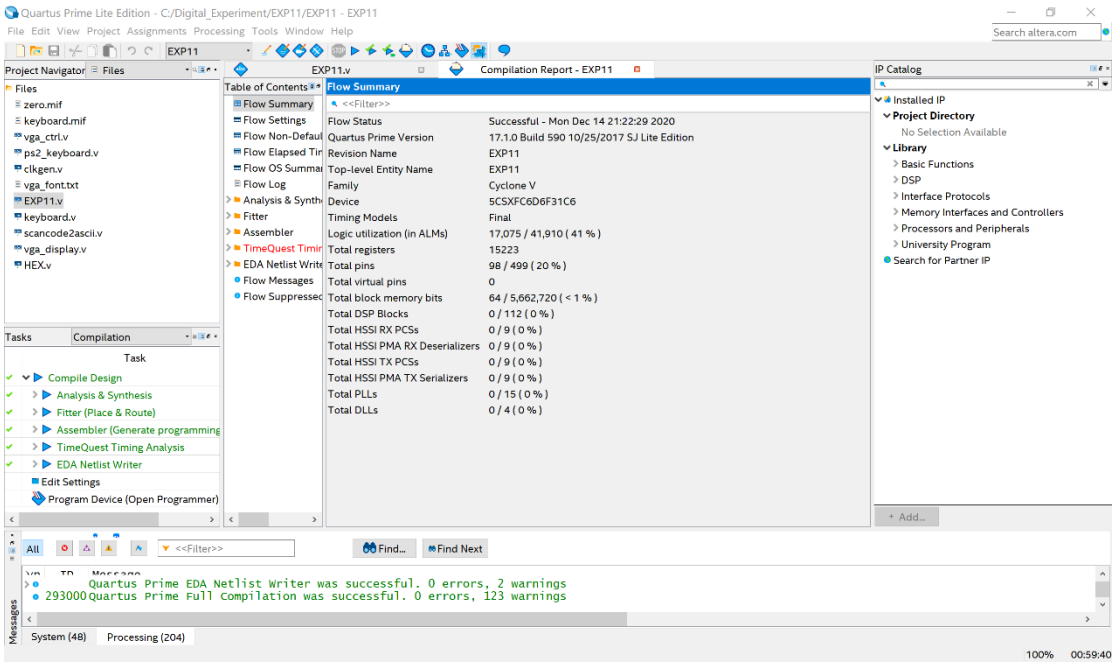
2、本次实验的讲义较短，刚开始做实验时可能没什么头绪，也不太能理解讲义中的做法。但在仔细思考后，由字符交互本身去逆向思考这个功能有哪些小功能，每一个小功能需要怎

样实现，就能够很好地和讲义联系起来了，理解起来也比较容易。

3、当需要实现的功能较为复杂时，可以每实现一个一部分时，就进行验证，以保证实验的每个阶段均成功实现，避免在最后一次性验证时，无法找出错误出处的情况。比如可以先写好键盘，通过数码管反馈，初步验证键盘时序逻辑是否正确，再实现显示器字符交互。

十、意见和建议

1、本实验编译时间太长，不易进行调试，可给出一些减少编译时长的具体做法示例。若不知如何减少编译时间则会出现下图情况：



编译了接近一个小时，但实际效果是正确的。

2、本实验工程规模稍大，可给出一些框架性的代码或者设计思路以供实验者参考。