数字逻辑与计算机组成实验

LAB 09：字符输入界面

郑凯琳 205220025
205220025@smail.nju.edu.cn

（一）实验目的

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

扩展要求：

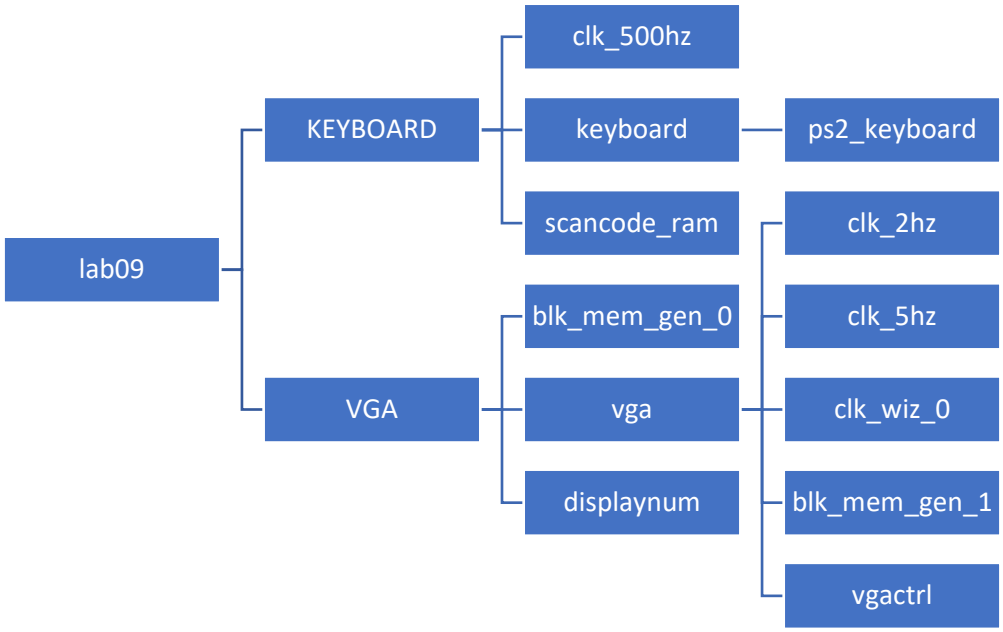
- 可以显示光标，建议可以用显示闪烁的竖线或横线作为光标。
- 支持 BackSpace 键删除光标前的字符。
- BackSpace 删除至本行开始后，再按 BackSpace 可以删除回车键，光标停留在上一行末尾的非空字符后
- 支持 Shift 键以及大小写字符输入。
- 支持方向键移动光标。
- 在行首显示命令提示符。

（二）实验环境/器材等

硬件器材：Nexys A7-100T 开发板
软件平台：Vivado 开发平台

（三）实验过程

模型概述



设计思路 & 设计代码

lab09.v

键盘处理：

1. clkgen_500hz 模块：该模块通过输入的时钟信号（clk）生成一个 500Hz 的时钟信号（clk_500hz）。这个时钟信号将在后续的计时操作中使用。
2. service_key_pressed 信号：根据键盘输入（key_pressed）和键码（keycode），如果键码为退格键（8'h66，即 ASCII 码为 0x66）、回车键（8'h5a，即 ASCII 码为 0x5a）或制表键（8'h29，即 ASCII 码为 0x29），则 service_key_pressed 被置为高电平。
3. write_sym 信号和计时器：write_sym 是一个标志位，用于指示是否可以写入字符到 VGA 显示模块。计时器（counter）用于控制写入字符的时机。

- 如果 Shift 键处于按下状态，并且按键计数器（press_counter_wire）为 1，则将 write_sym 置为 0，表示禁止写入字符。
- 如果 write_sym 为 0，计时器为 0，且键被按下并且要写入的字符（sym_to_write）不为 0，或者存在服务键被按下（service_key_pressed 为真），则执行下面的操作：
    - 将计时器设置为一个较大的值（26'd50000000）。
    - 将 write_sym 置为 1，表示允许写入字符。
- 如果计时器大于 1 且键被按下并且要写入的字符不为 0，或者存在服务键被按下，则递减计时器的值。
- 如果计时器为 1 且键被按下并且要写入的字符不为 0，或者存在服务键被按下，则在 500Hz 的时钟信号（clk_500hz）上翻转 write_sym 的值。
- 如果以上条件都不满足，则将计时器和 write_sym 复位为 0。

4. keyboard 模块：这个模块用于处理键盘输入，根据时钟信号、复位信号、PS2 接口时钟和数据、Shift 键状态等输入，输出键盘的按键状态、按键计数器以及其他控制信号。
5. scancode_ram 模块：这个模块用于接收键盘输入的键码，并根据 Shift 键和大写锁定键的状态将键码转换为相应的字符码（sym_to_write）。

```
clkgen_500hz clk500(
    .clkin(clk),
    .clkout(clk_500hz)
);

always @ (posedge clk)
begin
    service_key_pressed = key_pressed && ((keycode == 8'h66) || (keycode == 8'h5a) || (keycode == 8'h29));

    if (shift_state && press_counter_wire == 1)
        write_sym = 1'b0;
    else if (write_sym == 1'b0 && counter == 0 &&
        (key_pressed && sym_to_write != 8'b0 || service_key_pressed))
    begin
        counter = 26'd50000000;
        write_sym = 1'b1;
    end
    else if (counter > 26'd1 &&
        (key_pressed && sym_to_write != 8'b0 || service_key_pressed))
    begin
        counter = counter - 26'd1;
        write_sym = 1'b0;
    end
    else if (counter == 26'd1 && (key_pressed && sym_to_write != 8'b0 || service_key_pressed))
    begin
        if (clk_500hz)
            write_sym = ~write_sym;
    end
    else
    begin
        counter = 0;
        write_sym = 1'b0;
    end
end
```

显示器处理：

1. always @ (posedge write_sym)块：这个块是一个时钟边沿触发的过程，用于处理 VGA 显示位置和字符数据的更新。
- 如果键码（keycode）为回车键（8'h5a，即 ASCII 码为 0x5a），执行以下操作：
    - 将当前写入位置的行结束位置（row_end_pos[write_pos / 64]）设为当前写入位置在行内的偏移量（write_pos % 64）。
    - 将写入位置（write_pos）更新为下一行的起始位置，即当前位置加上 64 减去当前行的结束位置。
- 如果键码为退格键（8'h66，即 ASCII 码为 0x66），执行以下操作：
    - 如果写入位置大于 0，执行以下操作：
- 如果写入位置在行的开始位置（write_pos % 64 == 0）：
    - 将写入位置更新为上一行的末尾位置，即当前位置减去（64 减去上一行的结束位置）。
    - 将上一行的结束位置（row_end_pos[write_pos / 64 - 1]）设为 0。
    - 否则，将写入位置减 1。
- 如果键码既不是回车键也不是退格键，执行以下操作：
    - 将写入位置加 1。
- 如果写入位置是行的结束位置（write_pos % 63 == 0），将当前行的结束位置（row_end_pos[write_pos / 64]）设为 63。
2. vga 模块：这个模块用于控制 VGA 显示器的输出。它接收主时钟信号 clk、开关信号 SW、写入位置 write_pos、字符数据 ascii_code、字符索引 sym_idx、LED 状态信号 LED 以及 VGA 的红色、绿色、蓝色、垂直同步和水平同步信号。具体的功能和实现细节需要查看 vga 模块的实现代码。
3. displaynum 模块：这个模块用于控制数码管的显示。它接收一个 16 位数字作为输入，通过时钟信号 clk 和数码管的使能信号 AN 和段选信号 HEX 来控制数码管的显示。输入的数字由按键计数器、符号数据、键码组成，通过连接和组合形成一个 16 位数字。

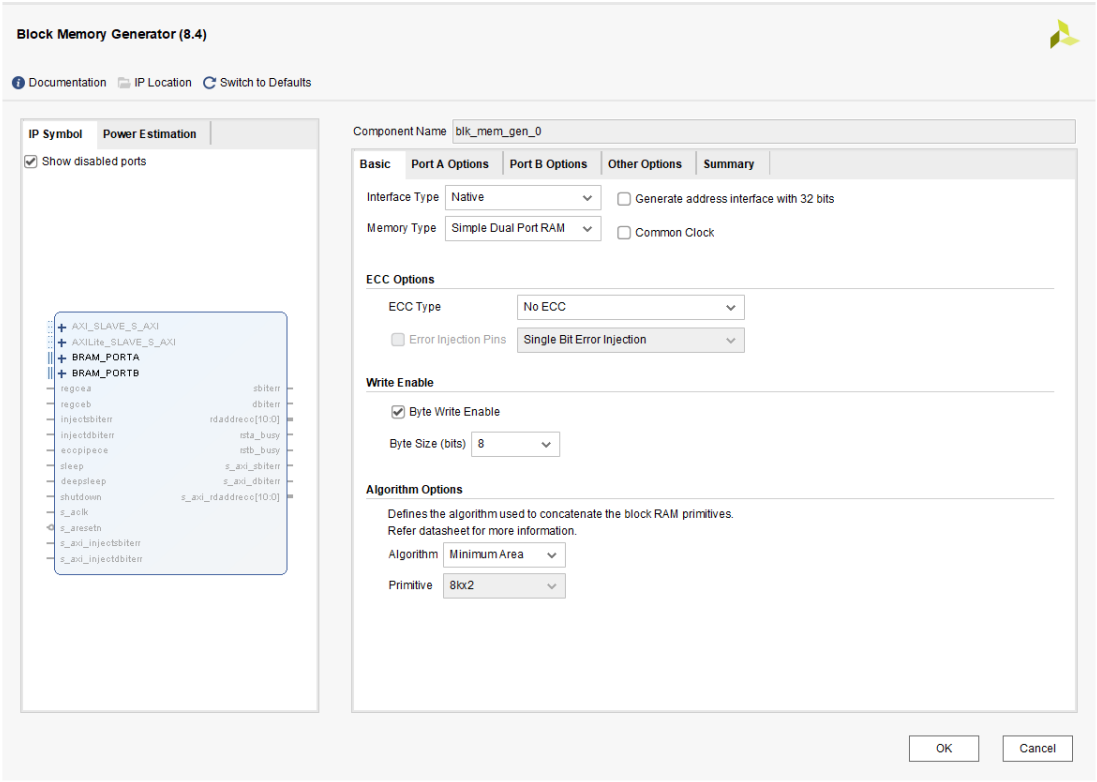
- num 信号为输入的 16 位数字，由按键计数器、8 位零、符号数据和键码组成。
- AN_active_map 为数码管的使能信号映射，当任意键被按下时，所有数码管均被使能。
- AN 信号为数码管的使能信号，用于控制数码管的显示。
- HEX 信号为段选信号，用于控制数码管显示的数字。

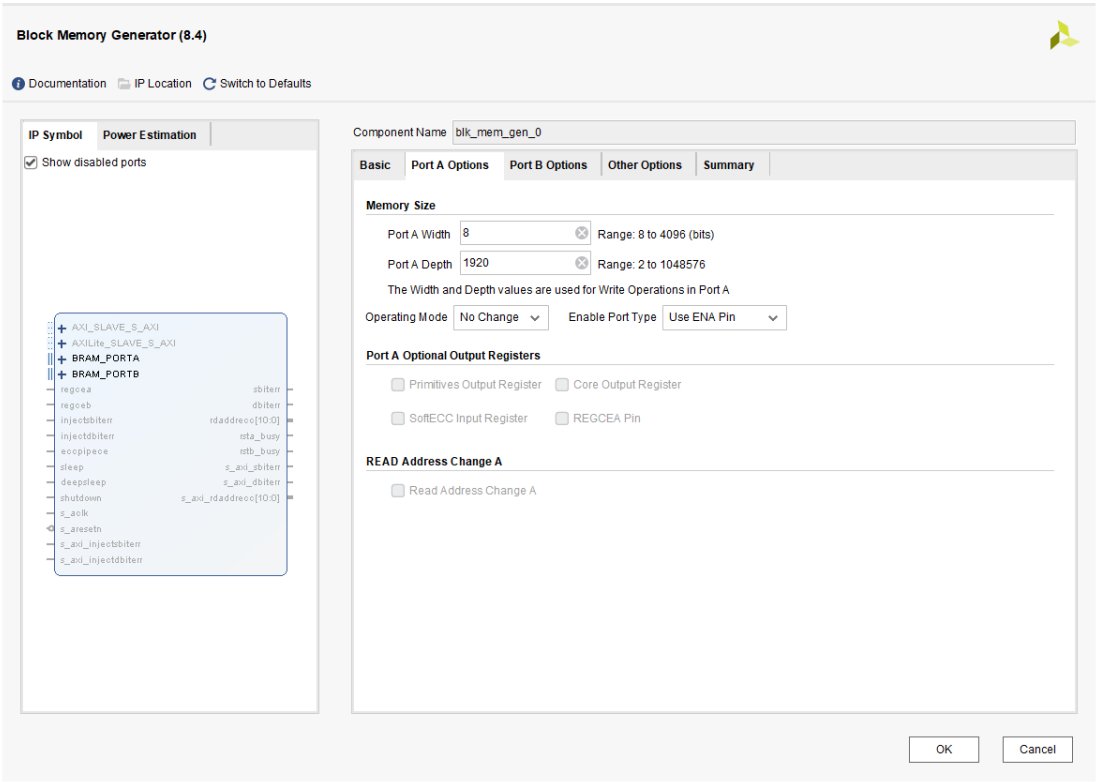
4. num_disp_active_map 信号：这是一个 8 位信号，用于控制数码管的使能。如果任意键被按下，所有数码管都被使能（8'b11111111），否则只有前四个数码管被使能（8'b11110000）。

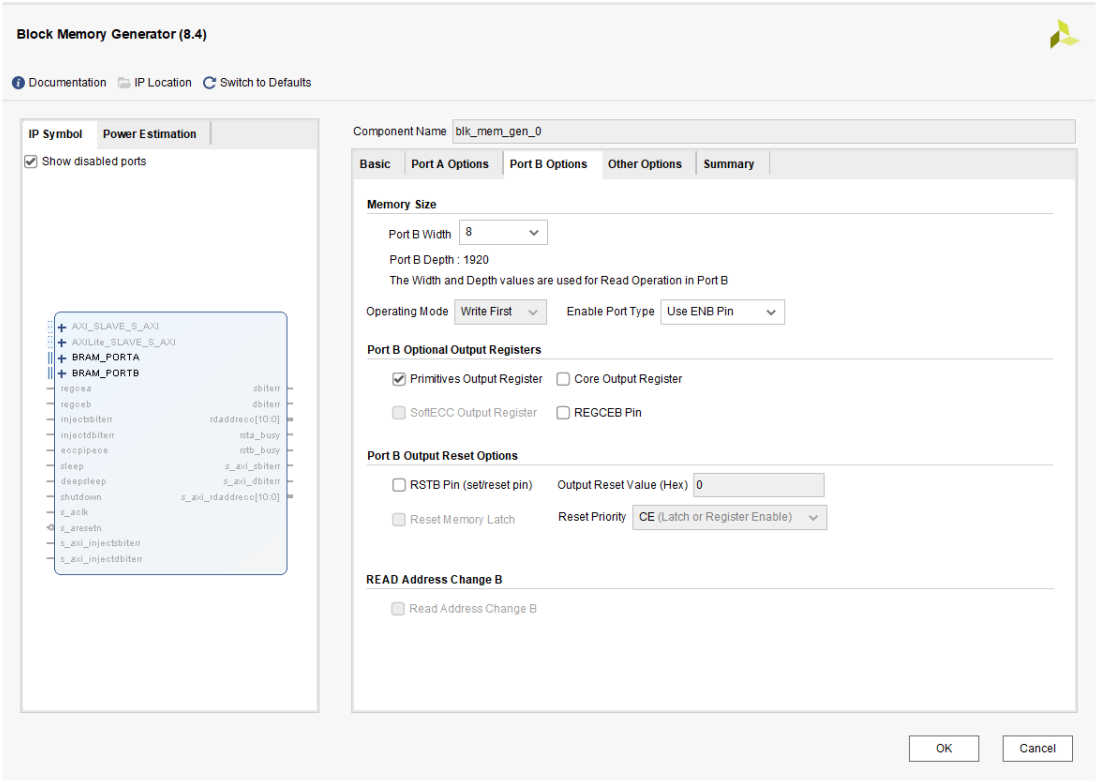
```
blk_mem_gen_0 ram(
    .clka(write_sym),
    .clkb(clk),
    .addra(write_pos),
    .addrb(sym_idx),
    .dina(sym_to_write),
    .doutb(ascii_code),
    .ena(1),
    .enb(1),
    .wea(1)
);

always @ (posedge write_sym)
begin
    if (keycode == 8'h5a)        // Enter
    begin
        row_end_pos[write_pos / 64] <= (write_pos % 64);
        write_pos <= (write_pos + 64) - (write_pos + 64) % 64;
    end
    else if (keycode == 8'h66)  // Backspace
    begin
        if (write_pos > 0)
            if (write_pos % 64 == 0)
            begin
                write_pos <= write_pos - (64 - row_end_pos[write_pos / 64 - 1]);
                row_end_pos[write_pos / 64 - 1] <= 8'd0;
            end
            else
                write_pos <= write_pos - 1;
    end
    else
    begin
        write_pos <= write_pos + 1;
        if (write_pos % 63 == 0)
            row_end_pos[write_pos / 64] <= 8'd63;
    end
end

    vga _vga_(
        .clk(clk),
        .SW(SW),
        .write_pos(write_pos),
        .ascii_code(ascii_code),
        .sym_idx(sym_idx),
        .LED(LED),
        .VGA_R(VGA_R),
        .VGA_G(VGA_G),
        .VGA_B(VGA_B),
        .VGA_VS(VGA_VS),
        .VGA_HS(VGA_HS)
    );

    displaynum dpnum(
        .num({press_counter_wire, 8'd0, sym_to_write, keycode}),
        .clk(clk),
        .AN_active_map(num_disp_active_map),
        .AN(AN),
        .HEX(HEX)
    );

    assign num_disp_active_map = (key_pressed)?8'b11111111:8'b11110000;
```

IP 核（blk_mem_gen_0）

由于每个字符的点阵大小为 16 x 9 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 4096 x 9bits，单口读。对于分辨率为 640 x 480 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 2100 x 8bits，使用双口读写，读写采用不同的时钟。


Block Memory Generator (8.4)
Documentation
IP Location
Switch to Defaults
IP Symbol
Power Estimation
Show disabled ports
AXI_SLAVE_S_AXI
AXILite_SLAVE_S_AXI
BRAM_PORTA
BRAM_PORTB
regcea
regceb
injectsbiterr
injectdbiterr
eccpipece
sleep
deepsleep
shutdown
s_aclk
s_aresetn
s_axi_injectsbiterr
s_axi_injectdbiterr
sbiterr
dbiterr
rdaddrecc[10:0]
rsta_busy
rstb_busy
s_axi_sbiterr
s_axi_dbiterr
s_axi_rdaddrecc[10:0]
Component Name
blk_mem_gen_0
Basic
Port A Options
Port B Options
Other Options
Summary
Interface Type
Native
Generate address interface with 32 bits
Memory Type
Simple Dual Port RAM
Common Clock
ECC Options
ECC Type
No ECC
Error Injection Pins
Single Bit Error Injection
Write Enable
Byte Write Enable
Byte Size (bits)
8
Algorithm Options
Defines the algorithm used to concatenate the block RAM primitives.
Refer datasheet for more information.
Algorithm
Minimum Area
Primitive
8kx2
OK
Cancel


Block Memory Generator (8.4)
Documentation
IP Location
Switch to Defaults
IP Symbol
Power Estimation
Show disabled ports
AXI_SLAVE_S_AXI
AXILite_SLAVE_S_AXI
BRAM_PORTA
BRAM_PORTB
regcea
regceb
injectsbiterr
injectdbiterr
eccpipece
sleep
deepsleep
shutdown
s_aclk
s_aresetn
s_axi_injectsbiterr
s_axi_injectdbiterr
sbiterr
dbiterr
rdaddrecc[10:0]
rsta_busy
rstb_busy
s_axi_sbiterr
s_axi_dbiterr
s_axi_rdaddrecc[10:0]
Component Name
blk_mem_gen_0
Basic
Port A Options
Port B Options
Other Options
Summary
Memory Size
Port A Width
8
Range: 8 to 4096 (bits)
Port A Depth
1920
Range: 2 to 1048576
The Width and Depth values are used for Write Operations in Port A
Operating Mode
No Change
Enable Port Type
Use ENA Pin
Port A Optional Output Registers
Primitives Output Register
Core Output Register
SoftECC Input Register
REGCEA Pin
READ Address Change A
Read Address Change A
OK
Cancel



keyboard.v

和之前 lab07 差不多。

vga.v

1. 内部变量和信号：
- vga_clk：VGA 时钟信号。
- pxl_color_code：像素颜色代码，指示要显示的像素颜色。
- bg_color：背景颜色。
- text_color：文本颜色。
- h_addr：水平地址。
- v_addr：垂直地址。
- hsync：水平同步信号。
- vsync：垂直同步信号。
- valid：有效信号。
- addr：地址。
- bitmap：位图数据。
- counter：计数器。
- bit_idx：位索引。
- bit_idx_start：起始位索引。

2. 时钟生成：

- clkgen_2hz 模块用于生成 2Hz 的时钟信号。
- clkgen_5hz 模块用于生成 5Hz 的时钟信号。
- clk_wiz_0 模块用于生成 VGA 时钟信号。

3. 时钟上升沿触发的过程：

- 当 valid 为 1 时执行以下操作：
    - 如果 bit_idx 等于 bit_idx_start + 8'd9，表示当前字符的位图已经全部显示完毕，需要更新索引和位索引：
        - 如果当前字符的索引加 1 后对 64 取模等于 0，表示已经显示完一行字符，需要更新行和字符索引。
        - 否则，更新字符索引和位索引。
    - 否则，仅更新位索引。

4. 像素颜色代码：

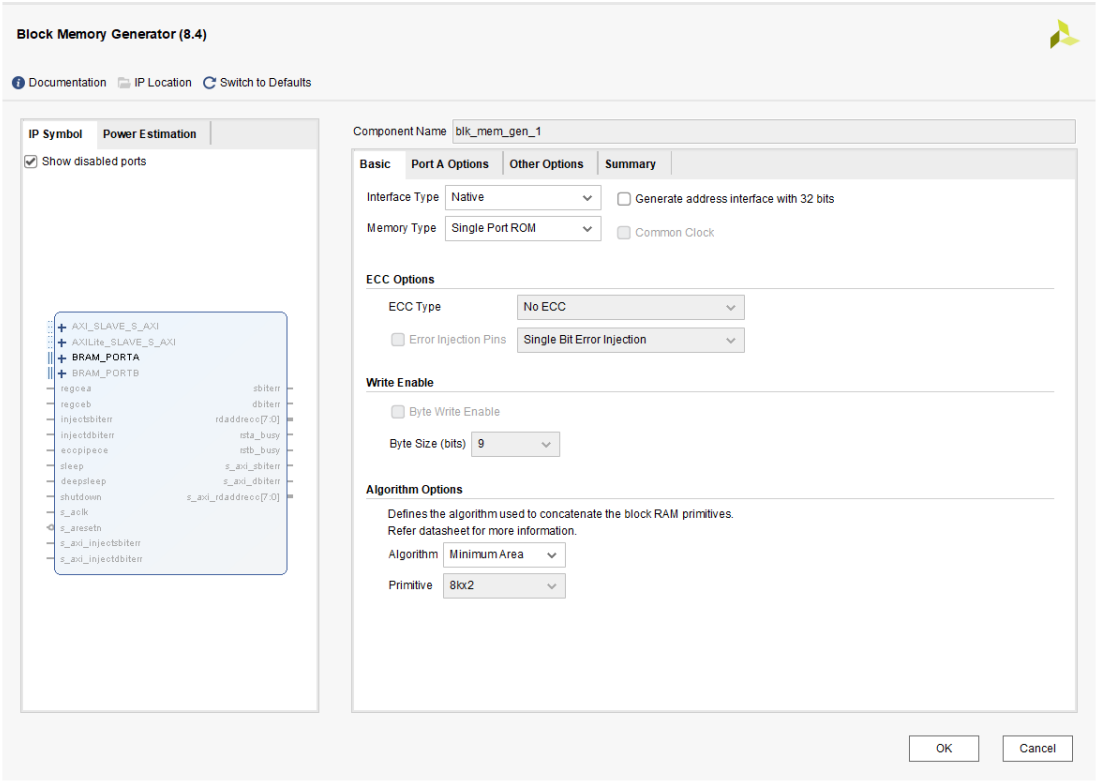
- 根据 write_pos 和 sym_idx 等条件判断，将 pxl_color_code 设置为要显示的像素颜色。
- 如果 write_pos 等于 sym_idx 且 bit_idx 模 10 小于等于 5，将 pxl_color_code 设置为文本颜色；否则，从位图数据中获取像素颜色。

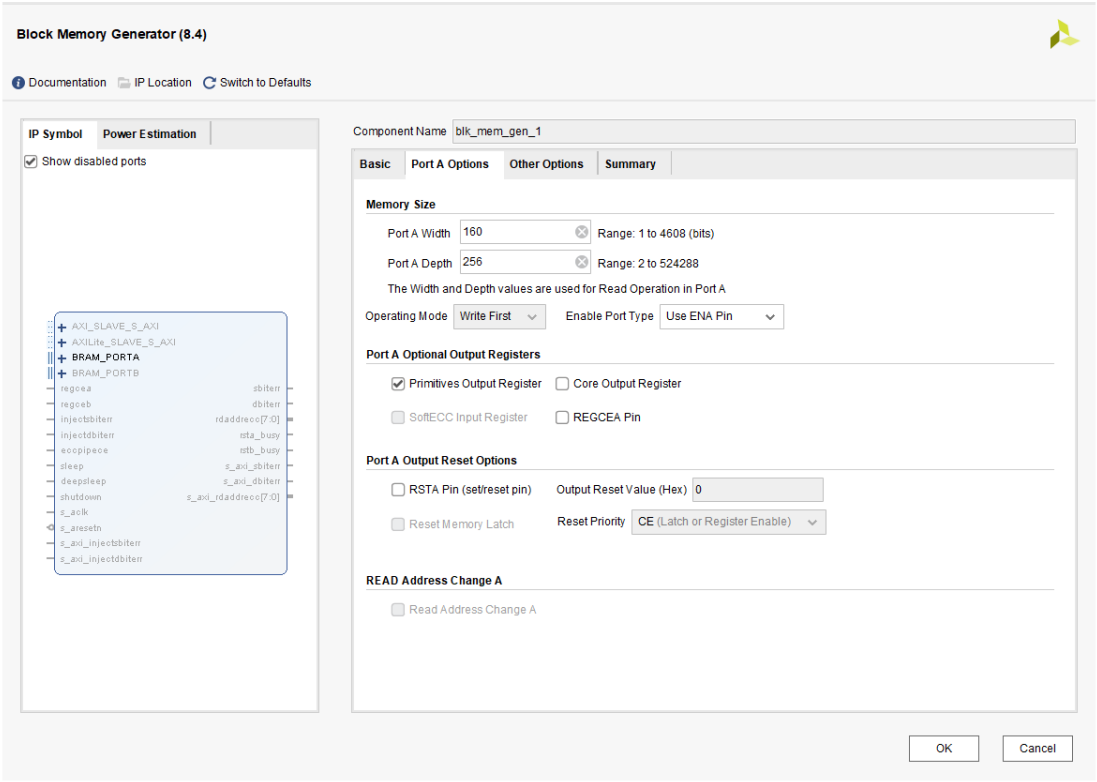
5. VGA 同步信号：

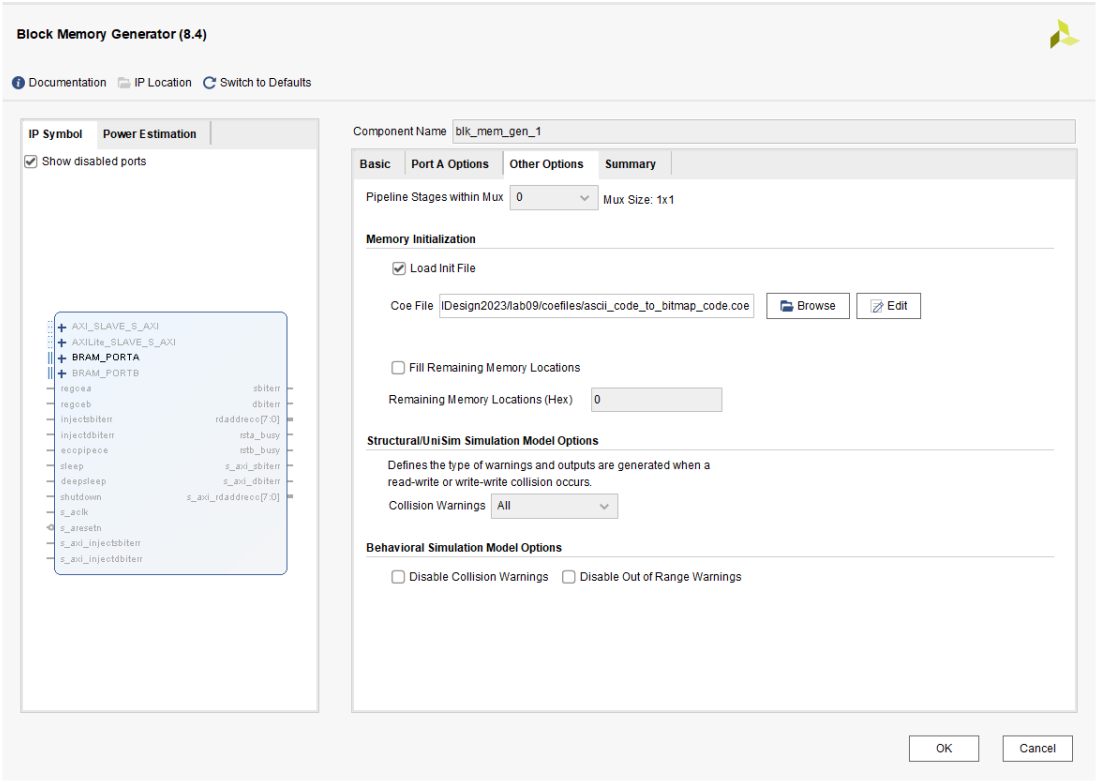
- 将 VGA_VS 设置为垂直同步信号。
- 将 VGA_HS 设置为水平同步信号。

```
always @ (posedge vga_clk)
begin
    if (valid)
    begin
        if (bit_idx == bit_idx_start + 8'd9)
        begin
            if ((sym_idx + 12'd1) % 12'd64 == 0)
            begin
                if (bit_idx == 8'd159)
                begin
                    bit_idx_start = 8'd0;
                    if (sym_idx == 12'd1919)
                        sym_idx = 12'd0;
                    else
                        sym_idx = sym_idx + 12'd1;
                end
                else
                begin
                    bit_idx_start = bit_idx_start + 8'd10;
                    sym_idx = sym_idx - 12'd63;
                end
            end
            else
            begin
                sym_idx = sym_idx + 12'd1;
            end

            bit_idx = bit_idx_start;
        end
        else
        begin
            bit_idx = bit_idx + 8'd1;
        end
    end
end

assign pxl_color_code = (write_pos == sym_idx && bit_idx % 10 <= 5)?
                            (clk_2hz)?
                                1'b1:
                                1'b0
                            :
                            bitmap[bit_idx];

assign VGA_VS = vsync;
assign VGA_HS = hsync;
```

IP 核（blk_mem_gen_1）



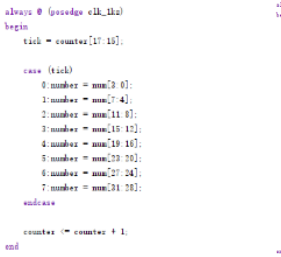


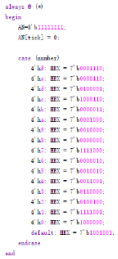


displaynum.v

数字显示模块，用于将输入的 32 位数字显示在数码管上。

1. 1kHz 时钟生成：
- clkgen_1kz 模块用于生成 1kHz 的时钟信号 clk_1kz。
2. 时钟上升沿触发的过程：
- 在每个 1kHz 时钟的上升沿，根据 counter 的高位（17:15）更新 tick 的值。
- 使用 case 语句将输入数字 num 分为 8 个部分，并在每个时钟周期中根据 tick 的值选择相应部分的数字。
- 计数器 counter 递增。
3. 数码管显示：
- AN 始终被设置为 8'b11111111，即所有位选均不激活。
- 根据 number 的值，使用 case 语句将对应的 7 段显示码赋值给 HEX，以显示对应的数字。
- 如果 number 的值不在 0 到 F 之间（即默认情况），则显示一个特殊的错误码。





硬件实现（引脚分配）

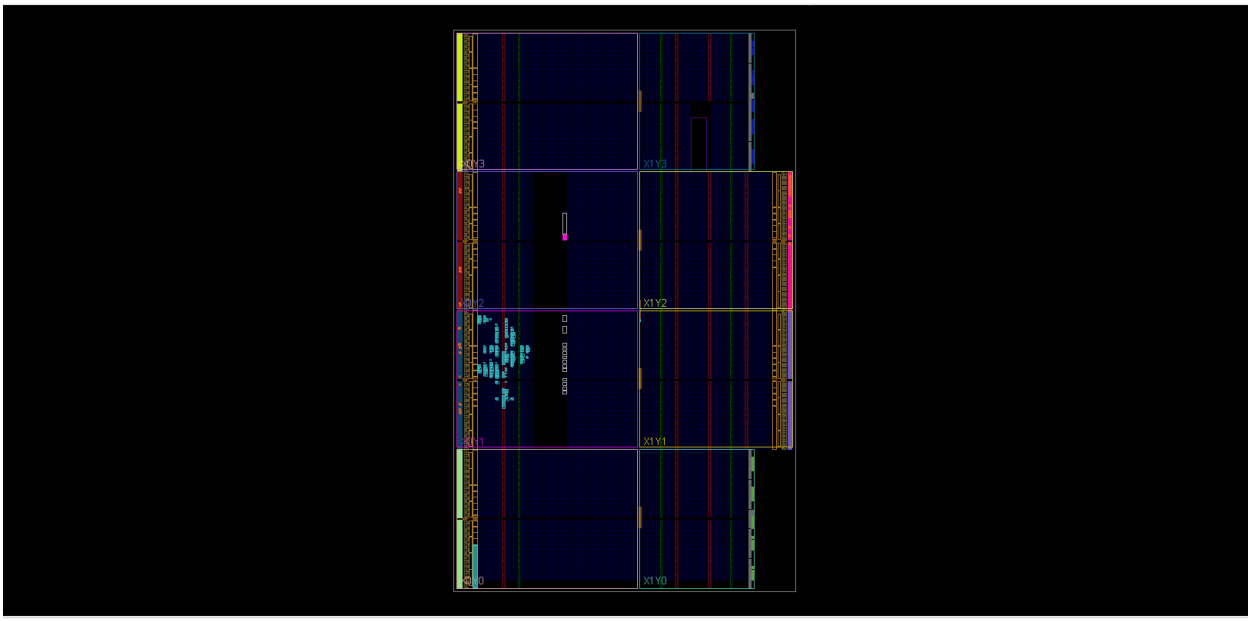
```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];


##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
#set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
#set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { LED[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { LED[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { LED[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { LED[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { LED[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { LED[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { LED[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { LED[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { LED[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { LED[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
#set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { LED[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { LED[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
#set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { LED[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
#set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { LED[13] }]; #IO_L23N_T3_A04_D20_14 Sch=led[13]
#set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { LED[14] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
#set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { LED[15] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]
```

```
##USB-RS232 Interface
set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports { uart_rxd_out }]; #IO_L10N_T1_AD11N_35 Sch=uart_rxd_out
set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports { uart_txd_in }]; #IO_L10P_T1_AD11P_35 Sch=uart_txd_in
##USB HID (PS/2)
```

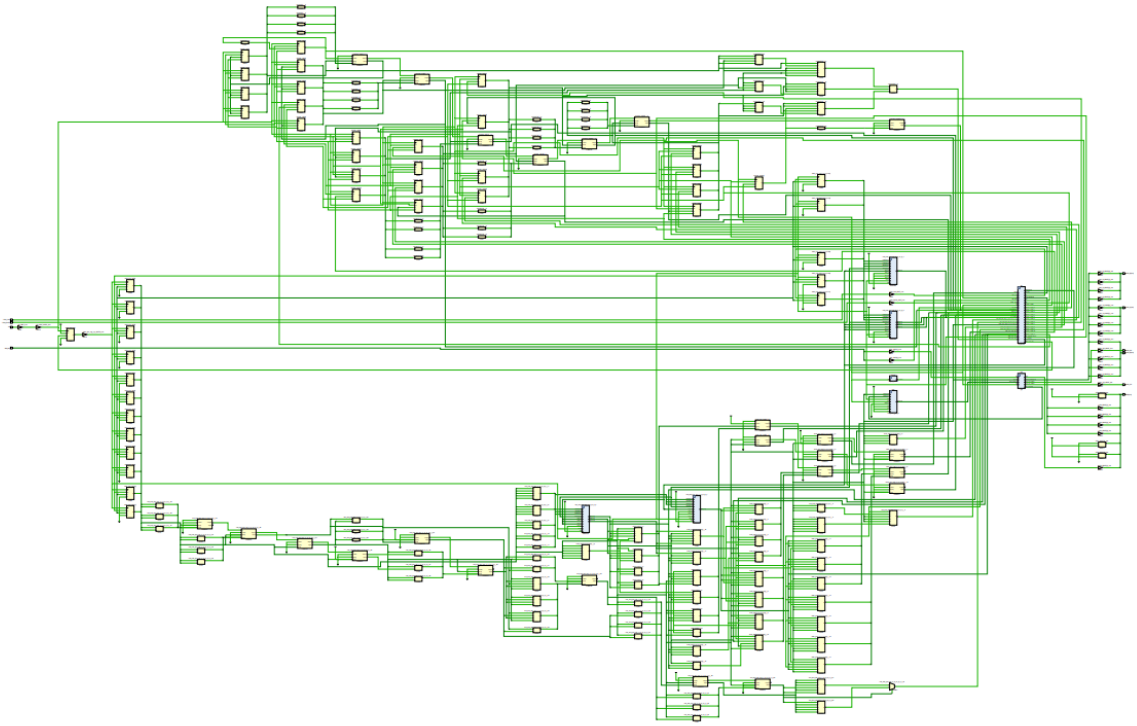
```
##VGA Connector
set_property -dict { PACKAGE_PIN A3    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[0] }]; #IO_L8N_T1_AD14N_35 Sch=vga_r[0]
set_property -dict { PACKAGE_PIN B4    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[1] }]; #IO_L7N_T1_AD6N_35 Sch=vga_r[1]
set_property -dict { PACKAGE_PIN C5    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[2] }]; #IO_L1N_T0_AD4N_35 Sch=vga_r[2]
set_property -dict { PACKAGE_PIN A4    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[3] }]; #IO_L9P_T1_AD14P_35 Sch=vga_r[3]
set_property -dict { PACKAGE_PIN C6    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[0] }]; #IO_L1P_T0_AD4P_35 Sch=vga_g[0]
set_property -dict { PACKAGE_PIN A5    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[1] }]; #IO_L3N_T0_DQS_AD5N_35 Sch=vga_g[1]
set_property -dict { PACKAGE_PIN B6    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[2] }]; #IO_L2N_T0_AD13N_35 Sch=vga_g[2]
set_property -dict { PACKAGE_PIN A6    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[3] }]; #IO_L3P_T0_DQS_AD5P_35 Sch=vga_g[3]
set_property -dict { PACKAGE_PIN B7    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[0] }]; #IO_L2P_T0_AD13P_35 Sch=vga_b[0]
set_property -dict { PACKAGE_PIN C7    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[1] }]; #IO_L4N_T0_35 Sch=vga_b[1]
set_property -dict { PACKAGE_PIN D7    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[2] }]; #IO_L6N_T0_VREF_35 Sch=vga_b[2]
set_property -dict { PACKAGE_PIN D8    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[3] }]; #IO_L4P_T0_35 Sch=vga_b[3]
set_property -dict { PACKAGE_PIN B11   IOSTANDARD LVCMOS33 } [get_ports { VGA_HS }]; #IO_L4P_T0_15 Sch=vga_hs
set_property -dict { PACKAGE_PIN B12   IOSTANDARD LVCMOS33 } [get_ports { VGA_VS }]; #IO_L3N_T0_DQS_AD1N_15 Sch=vga_vs
```

（四）实验结果

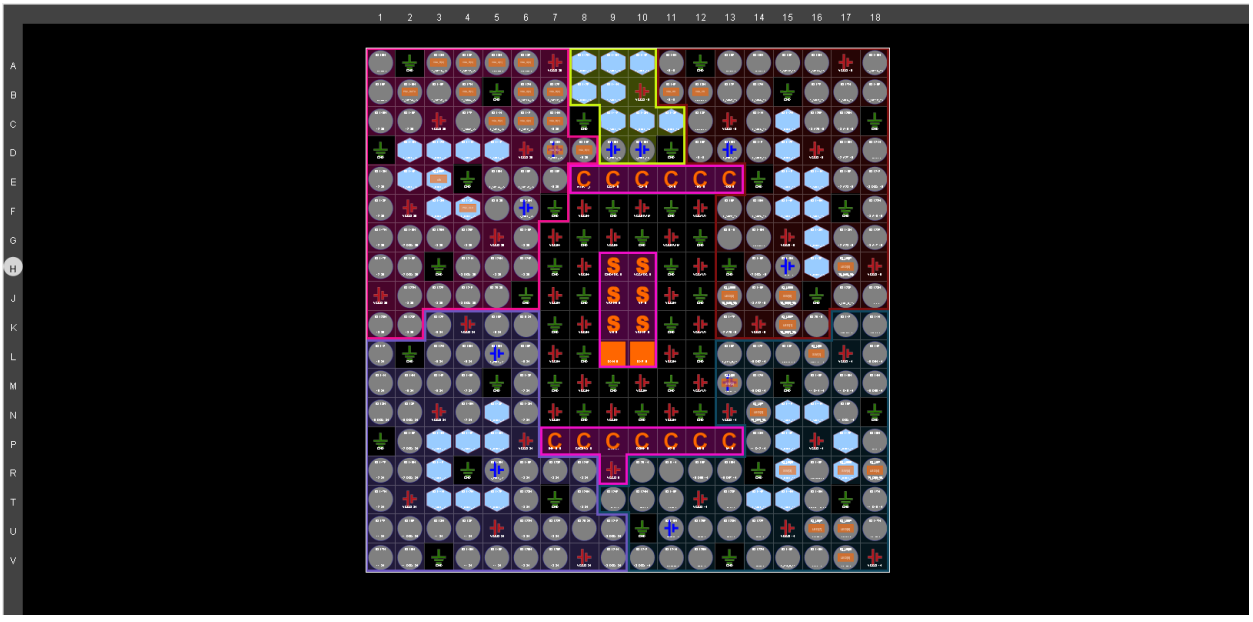
Device：



Schematic:



I/O Planning:



FGPA 结果：

（已附上视频）