数字逻辑与计算机组成实验

LAB 07：状态机及键盘输入

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

（一）实验目的

自行设计状态机，实现单个按键的 ASCII 码显示。

基本要求

- 七段数码管低两位显示当前按键的键码，中间两位显示对应的 ASCII 码 （转换可以考虑自行设计一个 ROM 并初始化）。只需完成字符和数字键的 输入，不需要实现组合键和小键盘。
- 当按键松开时，七段数码管的低四位全灭。
- 七段数码管的最高两位显示按键的总次数。按住不放只算一次按键。只考虑顺序按下和放开的情况，不考虑同时按多个键的情况。

高级要求

- 七段数码管低两位显示当前按键的键码，中间两位显示对应的 ASCII 码 （转换可以考虑自行设计一个 ROM 并初始化）。只需完成字符和数字键的 输入，不需要实现组合键和小键盘。
- 当按键松开时，七段数码管的低四位全灭。
- 七段数码管的最高两位显示按键的总次数。按住不放只算一次按键。只考虑顺序按下和放开的情况，不考虑同时按多个键的情况。

（二）实验原理

有限状态机 FSM（Finite State Machine）

有限状态机 FSM（Finite State Machine）简称状态机，是一个在有限个状态间进行转换和动作的计算模型。有限状态机含有一个起始状态、一个输入列表（列表中包含了所有可能的输入信号序列）、一个状态转移函数和一个输出端，状态机在工作时由状态转移函数根据当前状态和输入信号确定下一个状态和输出。状态机一般都从起始状态开始，根据输入信号由状态转移函数决定状态机的下一个状态。

有限状态机是数字电路系统中十分重要的电路模块，是一种输出取决于过去输入和当前输入的时序逻辑电路，它是组合逻辑电路和时序逻辑电路的组合。其中组合逻辑分为两个部分，一个是用于产生有限状态机下一个状态的次态逻辑，另一个是用于产生输出信号的输出逻辑，次态逻辑的功能是确定有限状态机的下一个状态；输出逻辑的功能是确定有限状态机的输出。除了输入和输出外，状态机还有一组具有 “记忆” 功能的寄存器，这些寄存器的功能是记忆有限状态机的内部状态，常被称作状态寄存器。

（三）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

（四）实验过程

模型概述

在键盘控制器 ps2_keyboard 的基础上实现对键盘上各按键的输出——显示键码及其对应的 ASCII 码，并显示按键的总次数。

数字抽象

1. 输入：
   - clk —— 时钟信号，与分频器的输出时钟信号连接
   - en —— 使能信号
2. 输出：
   - LED [4:0] —— 判断溢出、按键是否被按下、shift & caps & ctrl 是否被按下
   - AN [7:0] —— 七段 LED 数码管
   - HEX [6:0] —— 数码管上的 LED
3. inout 双向端口：
   - PS2_CLK —— PS/2 接口信号线，传输时钟
   - PS2_DATA —— PS/2 接口信号线，传输数据

设计思路 & 设计代码

ps2_keyboard.v

PS/2 键盘控制器：负责接收键盘送来的数据。

scancode_ram.v

1. 相应键码的 ASCII 码以 ROM 形式存放，与键码是一一对应的关系。
2. 根据 shift、caps 键的状态对 ASCII 码的输出进行赋值。
3. 初始化文件详见项目文件夹的 scancode/文件夹中的.txt 文件。

```
module scancode_ram(
    input clk,
    input [7:0] din,
    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] dout
    );

    reg [7:0] ascii [255:0];
    reg [7:0] ascii_shift [255:0];
    reg [7:0] ascii_caps [255:0];

initial
begin
    $readmemh("F:/DigitalDesign2023/lab07/scancode/ascii.txt", ascii, 0, 255);
    $readmemh("F:/DigitalDesign2023/lab07/scancode/ascii_shift.txt", ascii_shift, 0, 255);
    $readmemh("F:/DigitalDesign2023/lab07/scancode/ascii_caps.txt", ascii_caps, 0, 255);
end

always @ (*)
begin
    if(shift_state && !caps_state)
        dout <= ascii_shift[din];
    else if(caps_state && !shift_state)
        dout <= ascii_caps[din];
    else
        dout <= ascii[din];
end
endmodule
```

keyboard.v

顶层模块：

1. 在 ready 信号为 1 时，键码输出才有效。
2. 获取有效的键码输出后，需将 next_data_n 信号置为 0 以准备接收下一个有效信号。
3. 根据队列 FIFO 思想，设置 prev_code 和 cur_code，更好地判断各个键码。
4. release_flag 记录是否有 F0 键码输出。
5. 判断：如 prev_code 或 cur_code 是 F0 键码，则显示其键码。
6. 按键的总次数 —— 当 prev_code 和 cur_code 不相等，且 prev_code 和 cur_code 都不是断码。
7. 对于 shift、ctrl 按键的设置 —— 若 release_flag 为 0，则按键按下。
8. 对于 caps 按键的设置 —— caps 键按下、松开后，状态改变一次。

```
always @ (posedge clk)
begin
    next_data_n = 0;
    if (ready)
    begin
        prev_code = cur_code;
        cur_code = code;

        if (release_flag) begin
            release_flag = 0;
            prev_code = 0;
            if (cur_code == 8'h58)
                caps_state_n = ~caps_state_n;
        end

        if (cur_code != prev_code && cur_code != 8'hF0 && prev_code != 8'hF0)
            cnt = cnt + 1;

        if (cur_code == prev_code || prev_code == 0 || cur_code == 8'hF0) begin
            if (cur_code == 8'h12) shift_state_n = 1;
            if (cur_code == 8'h14) ctrl_state_n = 1;
//              if (cur_code == 8'h58) caps_state_n = 1;
//              begin
//                  if(caps_state_n == 1)
//                      caps_state_n = ~caps_state_n;
//              end
//              if (key_pressed == 0 || (non_service_key_pressed)) press_counter = press_counter + 1;
            key_pressed = 1;
        end
    end


    // 判断
    if (prev_code == 8'hF0)
    begin
        release_flag = 1;
        if (cur_code == 8'h12) shift_state_n = 0;
        if (cur_code == 8'h14) ctrl_state_n = 0;
//          if (cur_code == 8'h58) caps_state_n = 0;
        key_pressed = 0 || ctrl_state_n || shift_state_n;
//          if(caps_state_n == 0)begin
//              caps_state_n = ~caps_state_n;
//          end
    end
end
```

```
always @ (*)
begin
// 低两位：当前按键的键码
if(Q) begin
    if (clk_1ms == 3'b000)
    begin
    AN = 8'b11111110;
    case(tmp[3:0])...
    end
    else if (clk_1ms == 3'b001)
    begin
    AN = 8'b11111101;
    case(tmp[7:4])...
    end
// 中间两位：对应的ASCII码
    else if (clk_1ms == 3'b010)
    begin
    AN = 8'b11110111;
    case(ascii_res[3:0])...
    end
    else if (clk_1ms == 3'b011)
    begin
    AN = 8'b11101111;
    case(ascii_res[7:4])...
    end
// 高两位：按键的总次数
    else if (clk_1ms == 3'b100)
    begin
    AN = 8'b10111111;
    case(cnt[3:0])...
    end
    else if (clk_1ms == 3'b101)
    begin
    AN = 8'b01111111;
    case(cnt[7:4])...
    end
end
```

9. 七段 LED 数码管：
    a. 低两位 —— 当前按键的键码
    b. 中间两位 —— 对应的 ASCII 码
    c. 高两位 —— 按键的总次数

测试代码

pdf 里的键盘测试代码

- ps2_keyboard_model 模块：模块中主要是 kbd_sendcode task
- 分别将 ps2_keyboard_model 和 ps2_keyboard 实例化，并连接起来。

也对顶层文件进行实例化：

```
module lab08_test();
    reg clk;
    reg en;
    wire [4:0] LED;
    wire [7:0] AN;
    wire [6:0] HEX;
    reg PS2_CLK;
    reg PS2_DATA;

    lab08 t1(
        .clk(clk),
        .en(en),
        .LED(LED),
        .AN(AN),
        .HEX(HEX),
        .PS2_CLK(PS2_CLK),
        .PS2_DATA(PS2_DATA));
```

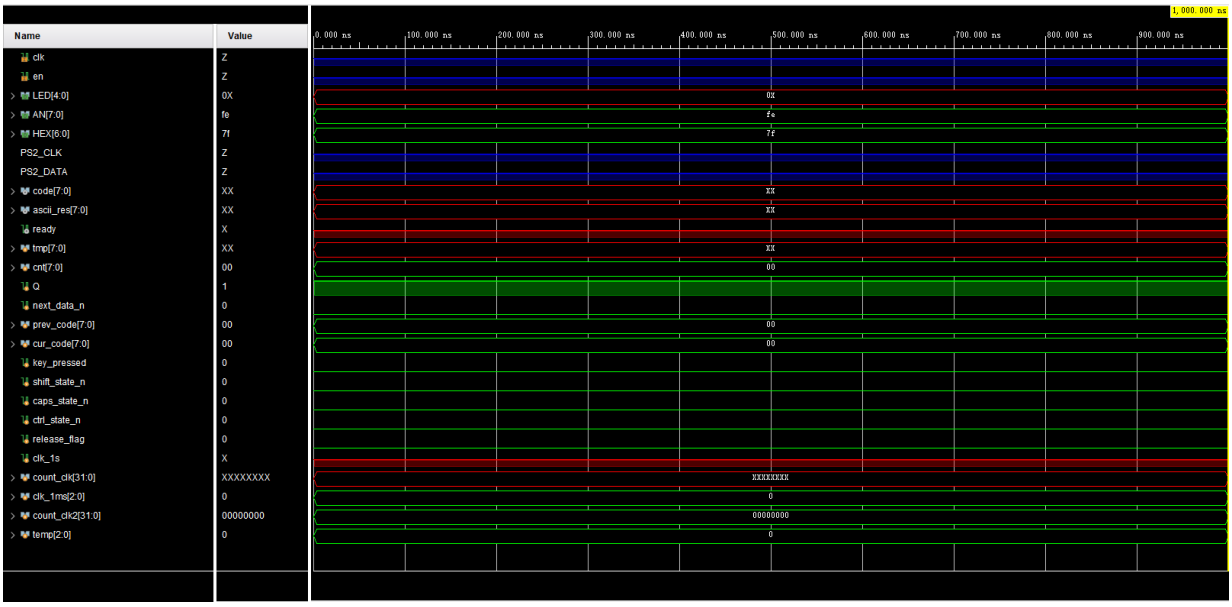
硬件实现（引脚分配）

```
## 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 { en }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
#set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { PS2_CLK }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
#set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { PS2_DATA }]; #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 E15   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[1] }]; #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[2] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { LED[3] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { LED[4] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]
```

```
##USB HID (PS/2)
set_property -dict { PACKAGE_PIN F4    IOSTANDARD LVCMOS33 } [get_ports { PS2_CLK }]; #IO_L13P_T2_MRCC_35 Sch=ps2_clk
set_property -dict { PACKAGE_PIN B2    IOSTANDARD LVCMOS33 } [get_ports { PS2_DATA }]; #IO_L10N_T1_AD15N_35 Sch=ps2_data

##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { HEX[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { HEX[1] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { HEX[2] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { HEX[3] }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { HEX[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { HEX[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { HEX[6] }]; #IO_L4P_T0_D04_14 Sch=cg
#set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { DP }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { AN[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { AN[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { AN[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { AN[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { AN[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { AN[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { AN[6] }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { AN[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
```

（五）实验结果

仿真结果:

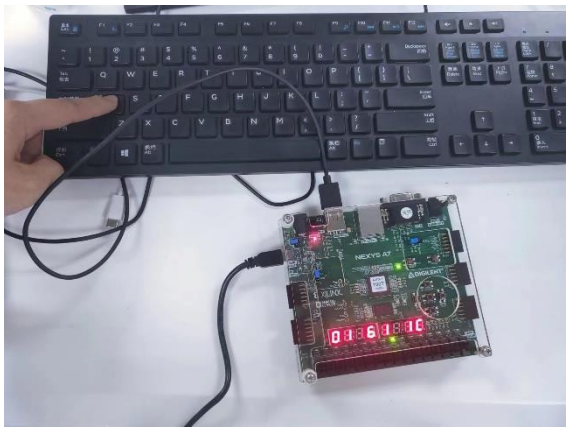


FGPA 结果:

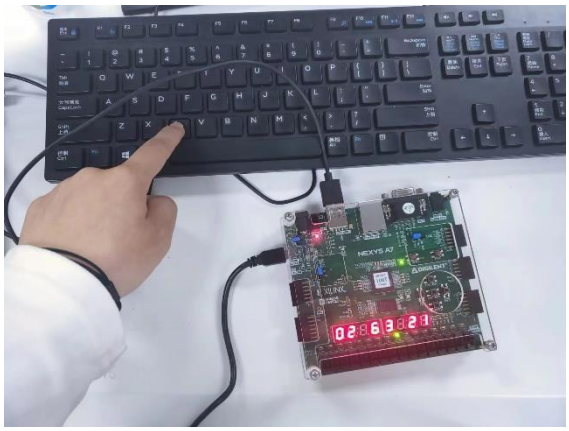
初始状态

字符输入

a

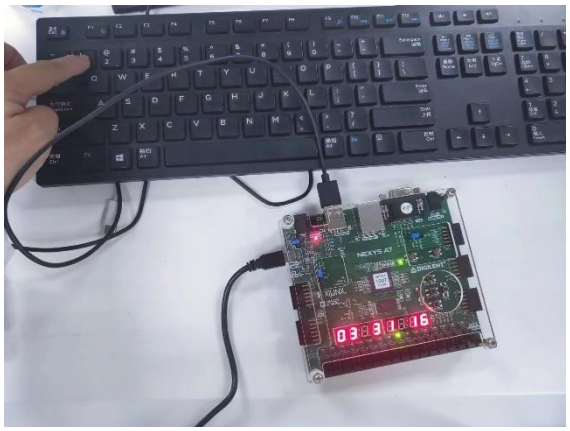


c

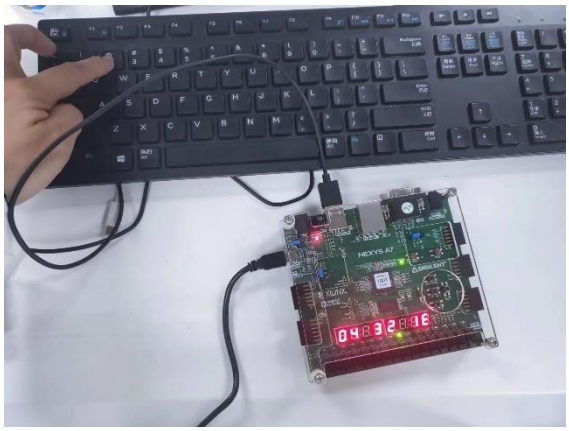


数字输入

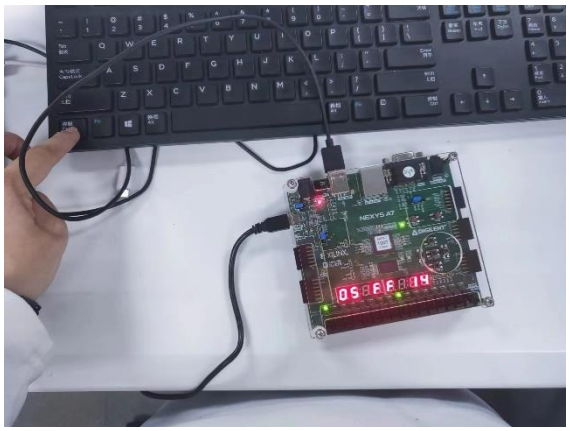
1



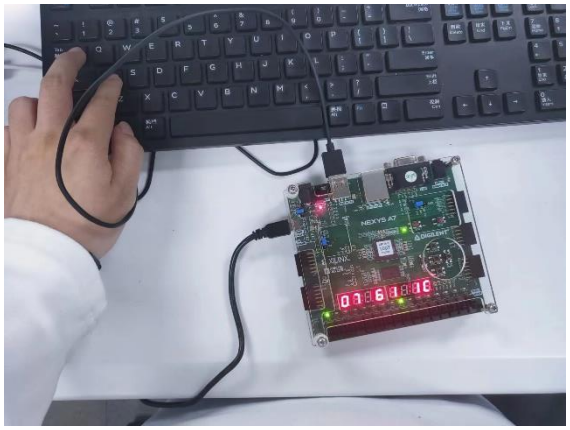
2



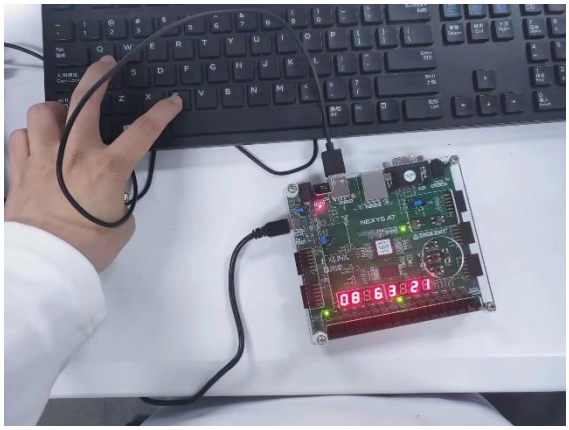
Ctrl 按键



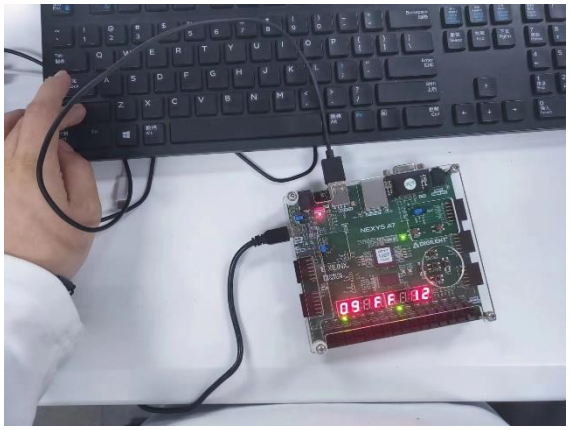
Ctrl + a



Ctrl + c

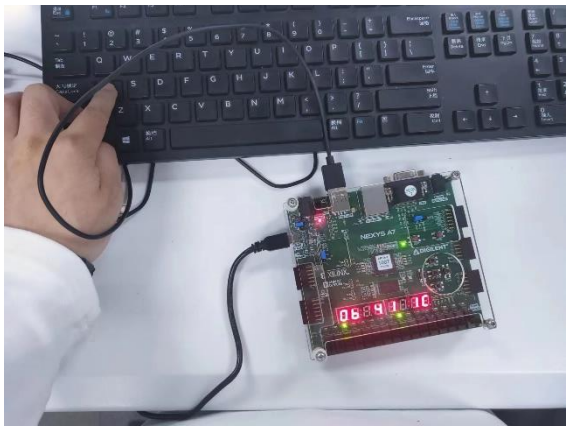


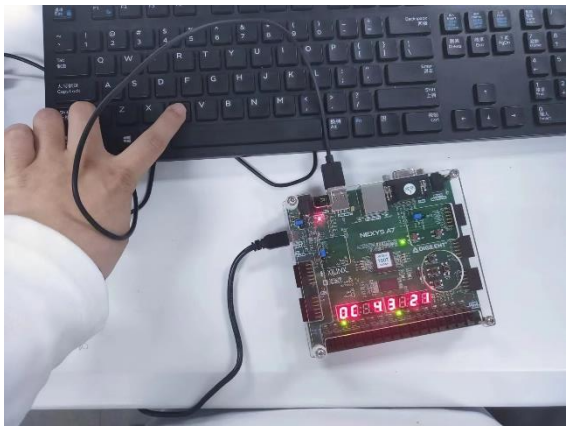
Shift 按键



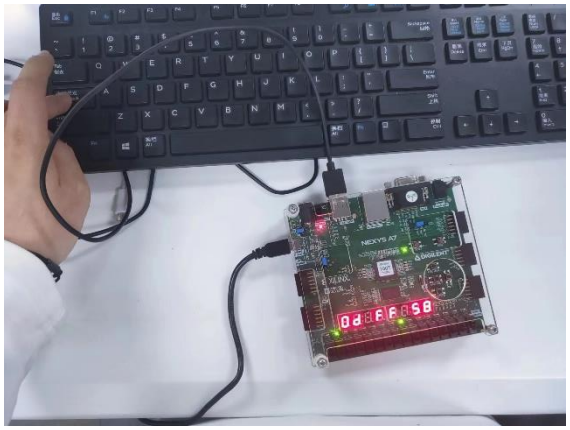
Shift + a

Shift + c



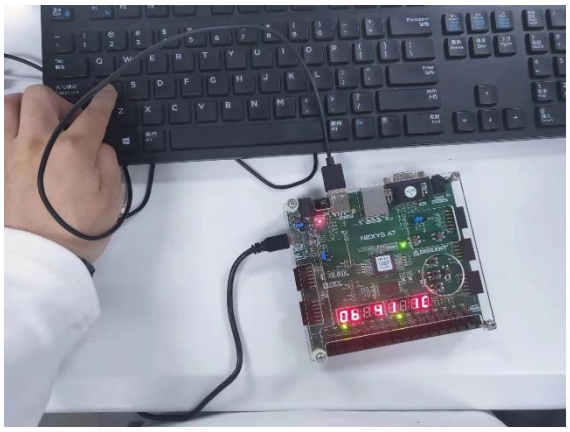


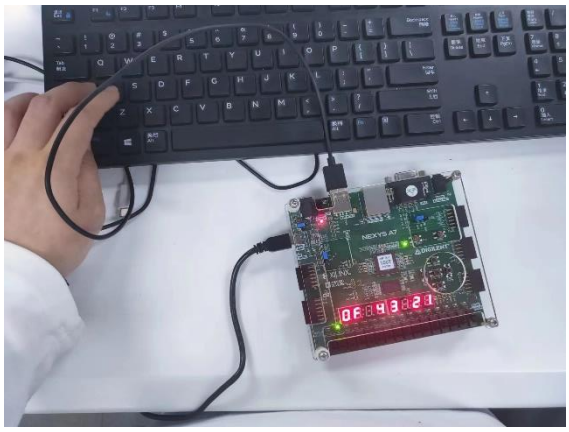
Caps 按键



A

C





（六）实验中遇到的问题及解决方法

问题：caps 的设置

解决方法：

不要设置它为按下或松开，因为它将保持不变，直到下次再按下。

需特别在 release_flag 为 1 时，将其状态转变。

```
            if (cur_code == prev_code || prev_code == 0 || cur_code == 8'hF0) begin
                if (cur_code == 8'h12) shift_state_n = 1;
                if (cur_code == 8'h14) ctrl_state_n = 1;
//                if (cur_code == 8'h58) caps_state_n = 1;
//                begin
//                    if(caps_state_n == 1)
//                        caps_state_n = ~caps_state_n;
//                end
//                if (key_pressed == 0 || (non_service_key_pressed)) press_counter = press_counter + 1;
                key_pressed = 1;
            end

        // 判断
        if (prev_code == 8'hF0)
        begin
            release_flag = 1;
            if (cur_code == 8'h12) shift_state_n = 0;
            if (cur_code == 8'h14) ctrl_state_n = 0;
//            if (cur_code == 8'h58) caps_state_n = 0;
            key_pressed = 0 || ctrl_state_n || shift_state_n;
//            if(caps_state_n == 0)begin
//                caps_state_n = ~caps_state_n;
//            end
        end
    end
```

*//注释//为错误想法。

```
            if (release_flag) begin
                release_flag = 0;
                prev_code = 0;
                if (cur_code == 8'h58)
                    caps_state_n = ~caps_state_n;
            end
```

正确想法。