实验 10 综合实验

张芷苒 PB21081601

01 实验目的

完成实验文档中第四题，通过 LED 点阵实现汉字的循环显示 。本实验最终使用了两个屏幕，滚动显示“这是 zzr 的数电大作业”。

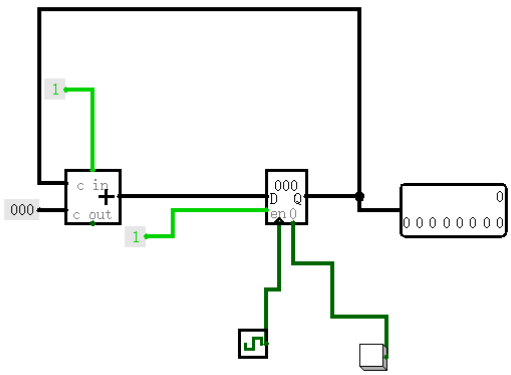
02 实验环境

- logisim,
- vscode

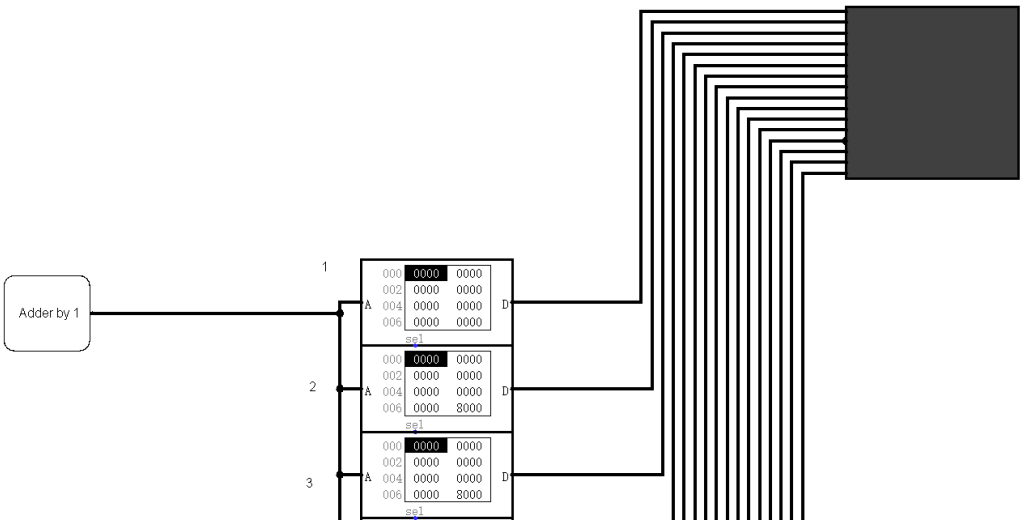
03 实验设计

3.1 电路设计

在Logisim中，用Adder和Register原件设计一个9位的自增加法器，原理图如下：



然后利用此加法器外接16个ROM原件来控制这16个ROM原件的状态变化，这16个ROM原件再分别与LED显示屏的16个接口相连来控制LED的亮灭情况。 由于图片太长，截图只体现了一部分：



注意到ROM中可以储存许多不同的状态，故可利用每一个状态来表示汉字不同的位置，从而实现汉字的循环滚动功能。

3.2 滚动循环的实现

ROM有许多不同的状态值，每一个状态都对应着不同的led显示情况，要实现滚动，就要实现每一个状态为上一个状态整体向右平移一位。

经计算与观察可得，某状态所对应的16进制数值除以2即为整体向右平移1格所对应的下一个状态的16进制值，乘2即为左移一个下一个状态所对应的16进制值。以下程序可以计算每一个16进制滚动各个左右平移状态数值：

```
void left(int x){
    int t[16] = {0};
    x *= 2;
    for (int i = 0; i < 16; i++){
        t[15 - i] = x;
        x *= 2;
    }
    for (int i = 0; i < 16; i++){
        if (i == 8)
        cout << endl;
        cout << hex << t[i] << " ";
        }
    cout << endl;
}

void right(int x){
    for (int i = 0; i < 16; i++){
        if (i == 8)
        cout << endl;
        cout /*<<setw(4)<<setfill('0')*/ << hex << x << " ";
        x /= 2;
        //x *= 2;
```

```
    }
    cout << endl;
}
```

可以利用之前实验所发字库16进制点阵查询获取每个汉字的16进制点阵。

为了更快的获取每个状态所对应的16进制数值，使用如下程序：

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
using namespace std;

int Atoint(string s, int radix)
{
    int ans = 0;
    for (int i = 0; i < s.size(); i++)
    {
        char t = s[i];
        if (t >= '0' && t <= '9')
            ans = ans * radix + t - '0';
        else
            ans = ans * radix + t - 'a' + 10;
    }
    return ans;
}

int num_length(int n)
{
    int sum = 0;
    while (n)
    {
        sum++;
        n /= 10;
    }
    return sum;
}

void left_gun(int x)
{
    int t[16] = {0};
    x *= 2;
    for (int i = 0; i < 16; i++)
    {
        t[15 - i] = x;
        x *= 2;
```

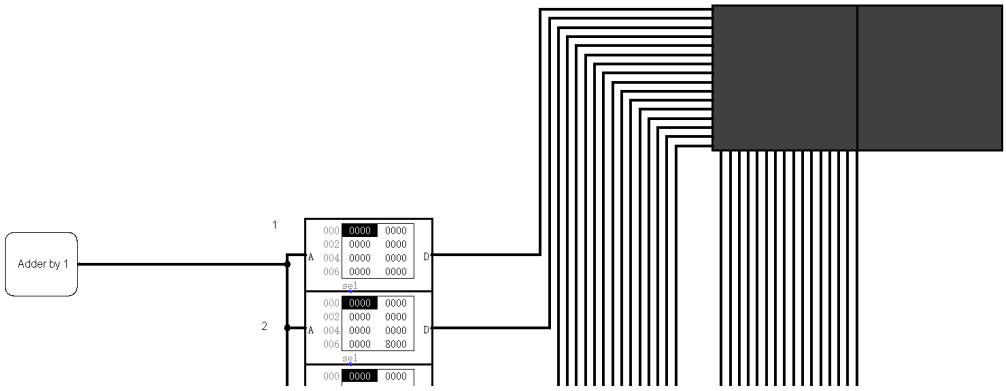
```
    }
    for (int i = 0; i < 16; i++)
    {
        if (i == 8)
            cout << endl;
        cout << hex << t[i] << " ";
    }
    cout << endl;
}

void right_gun(int x)
{
    for (int i = 0; i < 16; i++)
    {
        if (i == 8)
            cout << endl;
        cout /*<<setw(4)<<setfill('0')*/ << hex << x << " ";
        x /= 2;
        //x *= 2;
    }
    cout << endl;
}
int main()
{
    string str[16];
    string line;
    ifstream file;
    file.open("C:/Users/Lenovo/Desktop/hexnum.txt");
    int i = 0;
    while (getline(file, line))
    {
        str[i++] = line;
    }
    int n = str->size() / 4;
    int x[18];
    for (int k = 0; k < n; k++)
    {
        int j = 0;
        string s;
        for (int i = 0; i < str->size(); i++)
        {
            if (i % 4 == 0 && i != 0)
            {
                x[j++] = Atoint(s, 16);
                s.clear();
            }
            s += str[k][i];
        }
```

```
            cout<<"ROM"<<k+1<<":"<<endl;
            for (int i = 0; i < 16; i++)
            {
                left_gun(x[i]);
                right_gun(x[i]);
            }
            cout<<endl;
        }
    }
```

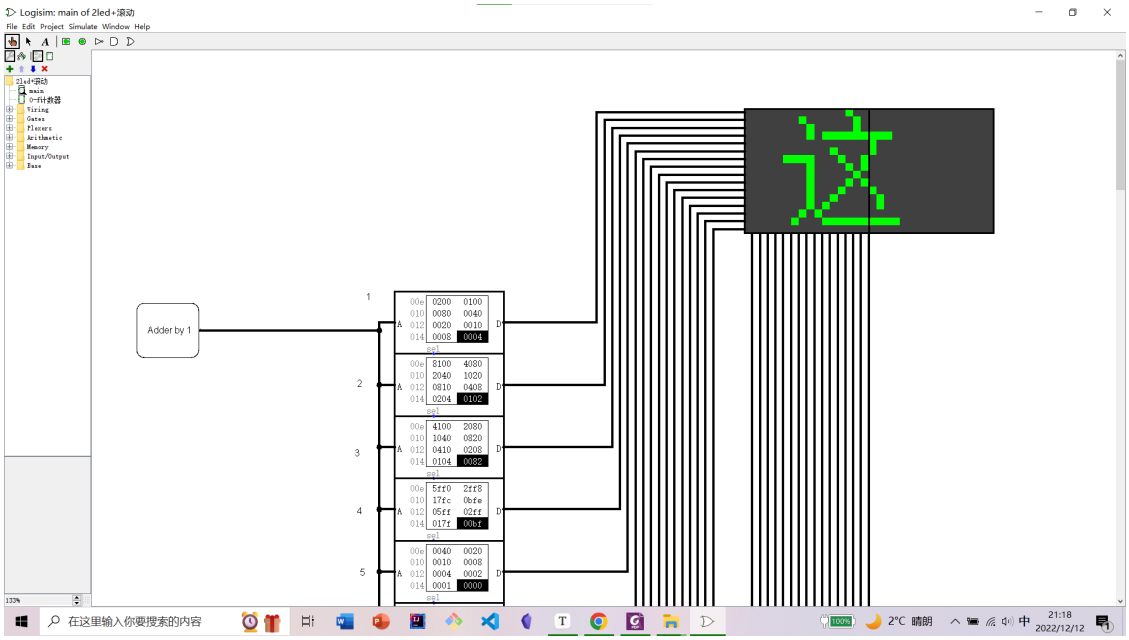
3.3 双屏扩展

为了使滚动效果更好，可以可以通过增加ROM元件的数目来控制更多显示屏的加入。要注意不同显示屏直接的衔接：



3.4 最终效果

最终效果的静态图如下图所示：



04 总结与思考

- 巩固了一学期以来所学的知识
- ~~不过 verilog debug 的能力貌似没有多大提升~~
- 最后一次任务量较大，少说花了几十个小时吧，感觉开放式选题有点难以下手，希望老师以后可以先构建几个框架供参考之类的
- 本来想做一个 myISA 和 ICS 联动一下，可惜能力不足事与愿违，详见“补充说明”

05 补充说明

本次实验，一开始是没有打算完成实验手册上的题摆烂的，本来写了一个可以实现LC3指令集的小ISA，但是在时序上有错误，没能达到预期结果。代码没有报错，我自己也找不出问题所在。但是在这个项目上花了大量时间，所以把代码和答题思路贴在下面，表示一下诚恳的态度。

另外还有一个问题就是，由于实验手册从未介绍过 uart，以及 RTS, CTS信号使用，所以我的构想在理论层面有一点实现难度。后来助教在群里发了相关内容，但是时间比较紧，来不及进行下一步调试了，比较可惜。

5.1 myISA 概要

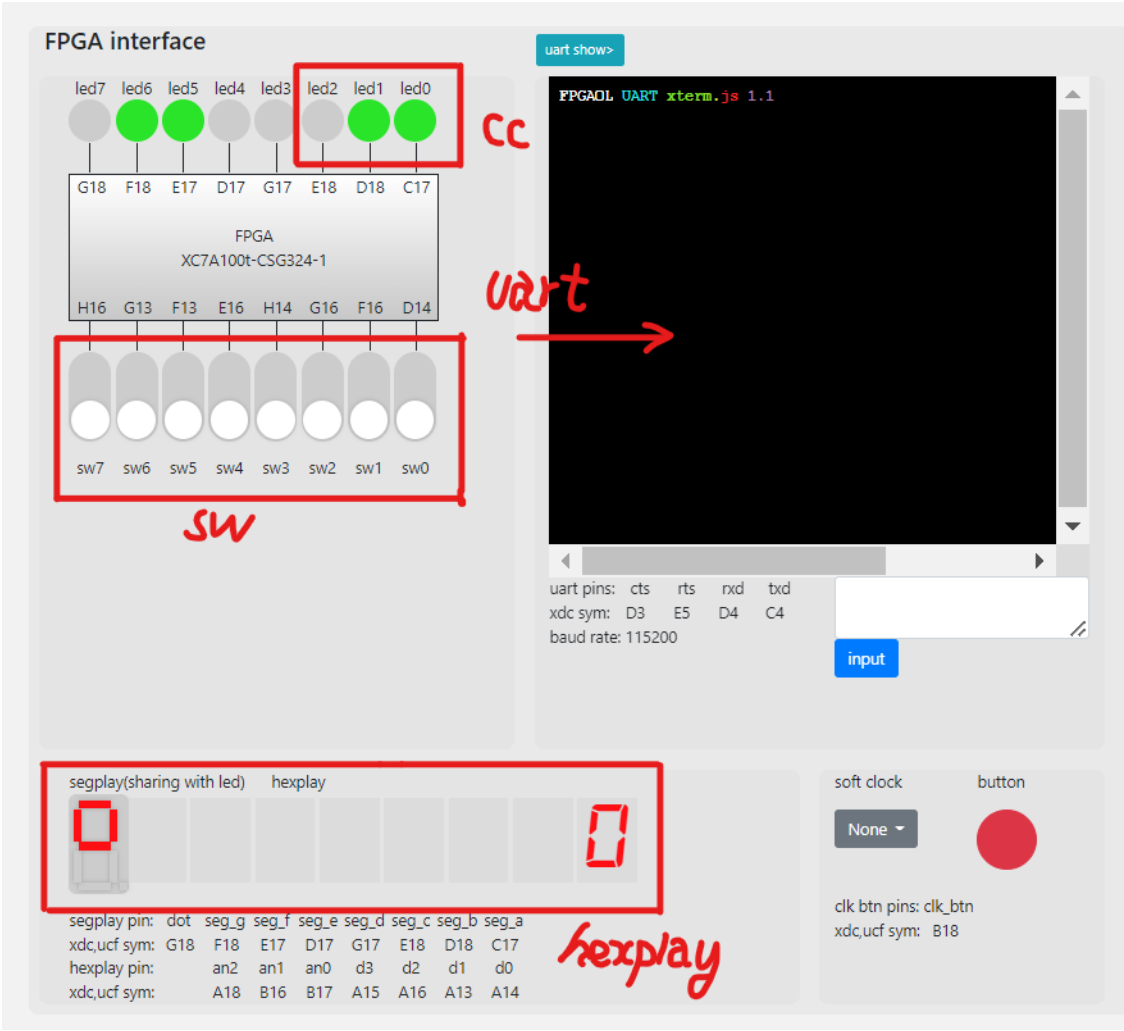
此实验将依托 fpgaol 在线平台，以有限状态自动机的方式实现一个支持编程的 myISA 指令集。用户可以通过查阅 myISA 指令表来操作内存和临时变量寄存器的值来进行编程。myISA 给出了内存组织方式、寄存器组、指令集（包括操作码、数据类型、寻址模式）等信息。myISA 的可寻址空间大小是 64，寻址基本单位是 16 位。myISA 和大多数的机器一样，提供了临时存储空间（R[7:0]）。

myISA 中的一条指令分为两个部分：操作码（做什么）和操作数（对谁操作）。 所有指令可以分为三类：运算（operate）、数据搬移（data movement）和控制（control）。运算类指令负责处理信息； 数据搬移类指令则负责在内存和寄存器之间以及内存/寄存器和 IO 设备之间转移信息； 控制类指令负责改变指令执行的顺序，即它们能让程序随时跳转至另一个地方继续执行（而不是常规的顺序向下执行）。

myISA 指令表：

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD+	0001				DR			SR1			0			SR2		
ADD+	0001				DR			SR1			1	imm5				
AND+	0101				DR			SR1			0			SR2		
AND+	0101				DR			SR1			1	imm5				
BR	1000				n	z	p	PCoffset9								
JMP	1100							BaseR								
JSR	0100				1	PCoffset11										
JSRR	0100				0			BaseR								
LD+	0010				DR			PCoffset9								
LDI+	1010				DR			PCoffset9								

fpgaol 各功能区使用说明：



- CC：条件码显示区， 显示当前条件码（高电平有效）
- hexplay：当 sw[7:0]全为 0 时，显示上一条输入进内存的指令（ 以低 4 位 16 进制呈现）；当 sw[7:0]有 1 时，显示为 1 的最低位为编号的寄存器内容
- sw：开关，控制 hexplay 的显示内容
- uart：串口输入，输入指令

5.2 实验设计

本实验设计的状态机有如下 4 个状态：

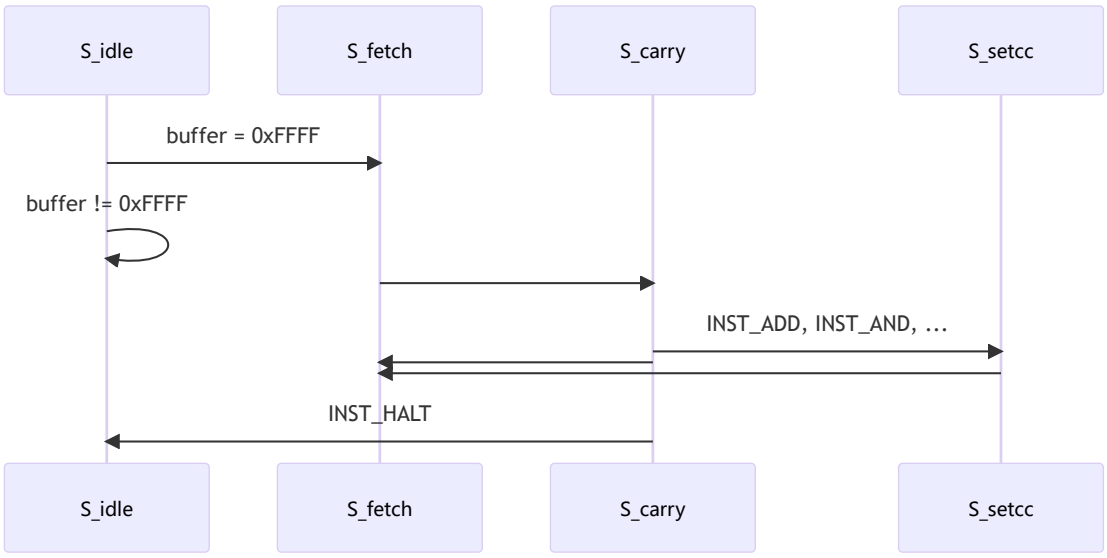
```
S_idle = 2'd0; 静止状态
S_fetch = 2’d1; 取指令状态
S_carry = 2’d2; 执行指令状态
S_setcc = 2'd3; 设置条件码状态
```

用户通过串口在静止状态输入指令，指令格式如下：

```
01010...01010101 输入一个 16bit 指令，最后一行指令必须是 HALT
1111111111111111 表示包括 HALT 在内的所有指令都输入完毕，开始执行
```

设置一个串口缓冲区寄存器 buffer[16:0]和指针 buffer_ptr 指向下一个待写入的 buffer 位置，buffer_ptr 从 buffer 高位写到低位，即初始状态是 buffer_ptr = 15。每当 rx_data 不是换行符，就向 buffer 中写，否则初始化 buffer_ptr = 15。当读到换行符时，说明这一行指令已经被完全读到 buffer 中，然后需要判断：如果 buffer 的内容是 0xFFFF，表示用户输入结束了，不应该写入内存中；否则需要将 buffer 内容存入 mem[PC]，然后使得 PC 自增。

状态机的第一部分是现态（cs）和次态（ns）的转换部分，遵从如下状态图：



S_idle 状态下，需要对 buffer 进行判断以切换到 S_fetch 状态，否则保持 S_idle 状态；S_fetch 状态下，无条件切换到 S_carry 状态；S_carry 状态需要判断 IR 中操作码以确定是否跳转到 S_setcc 状态和 S_idle 状态（操作码为 INST_HALT 时），否则跳转到 S_fetch 状态； S_setcc 状态无条件跳转到 S_fetch 状态。

状态机的第二部分是时序状态转换部分。在每个 posedge clk，此部分使 cs 转化为 ns 状态。如果 cs 是 S_idle 而 ns 是 S_fetch，保险起见还要将 buffer 清零。

状态机的第三部分是逻辑处理部分。这一部分指示了在各个状态我们的状态机还要执行的其他的操作。

- S_idle

首先是展示寄存器状态。通过开关 sw[7:0]以独热码形式来控制七段数码管显示内容，如 sw[7:0] = 8’b10000000 表示七段数码管应当展示寄存器 R7 的内容，格式为：R700XXXX，XXXX 为 R7 内容的十六进制编码。

其次是完成对 buffer 缓冲区的写入和对内存指令的写入。

- S_fetch

将 mem[PC]读入 IR，并使 PC 自增。

- S_carry

根据 IR 内容完成相应操作。 这里的执行操作以 LC-3（以 *Introduction to Computing Systems: from Bits & Gates to C/C++ & Beyond, 3rd edition* 书中所述为准）为基础，将未定义的 unused 指令定义为 INST_RSF，即右移一位指令（高位补逻辑 0），并修改条件码；将在此不会用到的 INST_RTI 重定义为 INST_MOD， 可以方便地取余数。各指令具体的实现操作如下：

```
INST_ADD:begin
if(IR[5] == 1'b1)begin
R[IR[11:9]] <= R[IR[8:6]] + {{11{IR[4]}}, IR[4:0]};
end
else begin
R[IR[11:9]] <= R[IR[8:6]] + R[IR[2:0]];
end
end
INST_AND:begin
if(IR[5] == 1'b1)begin
R[IR[11:9]] <= R[IR[8:6]] & {{11{IR[4]}}, IR[4:0]};
end
else begin
R[IR[11:9]] <= R[IR[8:6]] & R[IR[2:0]];
end
end
INST_BR:begin
if((IR[11] & CC[2]) | (IR[10] & CC[1]) | (IR[9] & CC[0])) begin
PC <= PC + IR[5:0];
R[3'b110] <= 16'hFFFF;
end
end
INST_JMP:begin
PC <= R[IR[8:6]][5:0];
end
INST_JSR:begin
R[7] <= PC;
if(IR[11]) PC <= PC + IR[5:0];
else PC <= R[IR[8:6]][5:0];
end
INST_LD:begin
R[IR[11:9]] <= mem[PC + IR[5:0]];
end
INST_LDI:begin
R[IR[11:9]] <= mem[mem[PC + IR[5:0]][5:0]];
end
INST_LDR:begin
R[IR[11:9]] <= mem[R[IR[8:6]][5:0]+IR[5:0]];
end7 / 11
INST_LEA:begin
R[IR[11:9]] <= PC + IR[5:0];
end
```

```
INST_NOT:begin
R[IR[11:9]] <= ~R[IR[8:6]];
end
INST_ST:begin
mem[PC + IR[5:0]] <= R[IR[11:9]];
end
INST_STI:begin
mem[mem[PC + IR[5:0]][5:0]] <= R[IR[11:9]];
end
INST_STR:begin
mem[R[IR[8:6]][5:0] + IR[5:0]] <= R[IR[11:9]];
end
INST_RSF:begin
R[IR[11:9]] <= {1'b0, R[IR[8:6]][7:1]};
end
INST_TRAP:begin
//PC <= ENDL;
end
INST_MOD:begin
R[IR[11:9]] <= R[IR[8:6]] % R[IR[2:0]];
end
```

- S_setcc

根据 IR 中 DR 寄存器内容设置条件码。

5.3 代码

设计文件：

```
module rx(
    input clk, rst,
    input rx,
    output reg rx_vld,
    output reg [7:0] rx_data
);
    parameter DIV_CNT = 10'd867;
    parameter HDIV_CNT = 10'd433;
    parameter RX_CNT = 4'h8;
    parameter C_IDLE = 1'b0;
    parameter C_RX = 1'b1;
    reg curr_state;
    reg next_state;
    reg [9:0] div_cnt;
    reg [3:0] rx_cnt;
```

```
    reg
rx_reg_0,rx_reg_1,rx_reg_2,rx_reg_3,rx_reg_4,rx_reg_5,rx_reg_6,rx_r
eg_7;
    wire rx_pulse;
    always@(posedge clk or posedge rst)
        begin
            if(rst)
                curr_state <= C_IDLE;
            else
                curr_state <= next_state;
        end

    always@(*)
        begin
            case(curr_state)
                C_IDLE:
                    if(div_cnt == HDIV_CNT)
                        next_state = C_RX;
                    else
                        next_state = C_IDLE;
                C_RX:
                    if((div_cnt == DIV_CNT) && (rx_cnt >= RX_CNT))
                        next_state = C_IDLE;
                    else
                        next_state = C_RX;
            endcase
        end

    always@(posedge clk or posedge rst)
        begin
            if(rst)
                div_cnt <= 10'h0;
            else if(curr_state == C_IDLE)
                begin
                    if(rx==1'b1)
                        div_cnt <= 10'h0;
                    else if(div_cnt < HDIV_CNT)
                        div_cnt <= div_cnt + 10'h1;
                    else
                        div_cnt <= 10'h0;
                end
            else if(curr_state == C_RX)
                begin
                    if(div_cnt >= DIV_CNT)
                        div_cnt <= 10'h0;
                    else
                        div_cnt <= div_cnt + 10'h1;
                end
```

```
        end

    always@(posedge clk or posedge rst)
        begin
            if(rst)
                rx_cnt <= 4'h0;
            else if(curr_state == C_IDLE)
                rx_cnt <= 4'h0;
            else if((div_cnt == DIV_CNT) && (rx_cnt < 4'hF))
                rx_cnt <= rx_cnt + 1'b1;
        end

        assign rx_pulse = (curr_state == C_RX) && (div_cnt ==
DIV_CNT);

    always@(posedge clk)
        begin
            if(rx_pulse)
                begin
                    case(rx_cnt)
                        4'h0: rx_reg_0 <= rx;
                        4'h1: rx_reg_1 <= rx;
                        4'h2: rx_reg_2 <= rx;
                        4'h3: rx_reg_3 <= rx;
                        4'h4: rx_reg_4 <= rx;
                        4'h5: rx_reg_5 <= rx;
                        4'h6: rx_reg_6 <= rx;
                        4'h7: rx_reg_7 <= rx;
                    endcase
                end
        end

    always@(posedge clk or posedge rst)
        begin
            if(rst)
                begin
                    rx_vld <= 1'b0;
                    rx_data <= 8'h55;
                end
            else if((curr_state==C_RX)&&(next_state==C_IDLE))
                begin
                    rx_vld <= 1'b1;
                    rx_data <=
{rx_reg_7,rx_reg_6,rx_reg_5,rx_reg_4,rx_reg_3,rx_reg_2,rx_reg_1,rx_
reg_0};
                end
            else
                rx_vld <= 1'b0;
```

```
        end
endmodule

//顶层模块
module top(
    input clk,
    input rx,
    output reg [7:0] led,
    input [7:0] sw,
    output reg [2:0] hexplay_an,
    output reg [3:0] hexplay_data
);
    reg [31:0] data;//七段数码管
    //LC-3寄存器
    reg [15:0] mem[63:0];
    reg [5:0] PC;//没有ENDL，最后写一行HALT就行
    reg [5:0] ptr;
    reg [2:0] CC;
    reg [15:0] IR;
    reg [15:0] R[7:0];
    initial begin
        PC = 0;
        ptr = 0;
        CC = 3'b000;
        IR = 0;
        R[7] = 0;
        R[6] = 0;
        R[5] = 0;
        R[4] = 0;
        R[3] = 0;
        R[2] = 0;
        R[1] = 0;
        R[0] = 0;
    end
    wire [7:0] trap_vec8;
    wire ben;
    assign trap_vec8 = IR[7:0];

    //缓冲区
    reg [15:0] buffer = 0;
    reg [3:0] buffer_ptr;
    initial buffer_ptr = 4'b1111;

    wire [7:0] rx_data;
    rx rx_inst(.clk (clk),.rst (rst),.rx (rx),.rx_vld
(rx_vld),.rx_data (rx_data));

    //LC-3状态机
```

```
    reg [1:0] cs;//current_state
    reg [1:0] ns;//next_state
    parameter S_idle = 2'd0;
    parameter S_fetch = 2'd1;
    parameter S_carry = 2'd2;
    parameter S_setcc = 2'd3;
    initial cs = S_idle;

    parameter INST_ADD = 4'b0001;
    parameter INST_AND = 4'b0101;
    parameter INST_BR = 4'b1000;
    parameter INST_JMP = 4'b1100;
    parameter INST_JSR = 4'b0100;
    parameter INST_LD = 4'b0010;
    parameter INST_LDI = 4'b1010;
    parameter INST_LDR = 4'b0110;
    parameter INST_LEA = 4'b1110;
    parameter INST_NOT = 4'b1001;
    parameter INST_MOD = 4'b0000;
    parameter INST_ST = 4'b0011;
    parameter INST_STI = 4'b1011;
    parameter INST_STR = 4'b0111;
    parameter INST_TRAP = 4'b1111;//全1是输入结束标记，
1111000000000000，用&R来判断
    parameter INST_RSF = 4'b1101;//右移，1101 DR SR ...

    //状态机第一部分
    always@(*)begin
        case(cs)
            S_idle:begin
                if(rx_vld && rx_data == 8'h0a && buffer ==
16'b1111111111111111)
                    ns = S_fetch;
                else ns = S_idle;
            end
            S_fetch:begin
                ns = S_carry;
            end
            S_carry:begin
                case(IR[15:12])
                    INST_ADD, INST_AND, INST_LD, INST_LDI,
INST_LDR, INST_NOT, INST_RSF: ns = S_setcc;
                    INST_TRAP: ns = S_idle;
                    default: ns = S_fetch;
                endcase
            end
            S_setcc: ns = S_fetch;
            default: ns = S_idle;
```

```
        endcase
    end
    //状态机第二部分
    always@(posedge clk)begin
        if(cs == S_idle && ns == S_carry) begin
            //buffer <= 0;
        end
        cs <= ns;
    end

    //状态机第三部分
    always@(posedge clk)begin
        led <= {5'b0, CC};
        case(cs)
        S_idle: begin
            if(rx_vld)begin
                if(rx_data == 8'h0a)begin
                    buffer_ptr <= 4'b1111;
                    if(buffer!=16'b1111111111111111)begin
                        mem[ptr] <= buffer;
                        ptr <= ptr + 1;
                    end
                end
                else begin
                    buffer[buffer_ptr] <= rx_data[0];
                    buffer_ptr <= buffer_ptr - 1;
                end
            end
            casex(sw)
                8'bxxxxxxx1: data <= {4'ha,4'h0,8'h0,R[0]};
                8'bxxxxxx1x: data <= {4'ha,4'h1,8'h0,R[1]};
                8'bxxxxx1xx: data <= {4'ha,4'h2,8'h0,R[2]};
                8'bxxxx1xxx: data <= {4'ha,4'h3,8'h0,R[3]};
                8'bxxx1xxxx: data <= {4'ha,4'h4,8'h0,R[4]};
                8'bxx1xxxxx: data <= {4'ha,4'h5,8'h0,R[5]};
                8'bx1xxxxxx: data <= {4'ha,4'h6,8'h0,R[6]};
                8'b1xxxxxxx: data <= {4'ha,4'h7,8'h0,R[7]};
                default: data <= {4'hc,2'h0,ptr,4'h0,mem[ptr-1]};
            endcase
        end
        S_fetch:begin
            IR <= mem[PC];
            PC <= PC + 1;
        end
        S_setcc:begin
            if(R[IR[11:9]] == 0) CC <= 3'b010;
            else if(R[IR[11:9]][15]) CC <= 3'b100;
            else CC <= 3'b001;
```

```
        end
        S_carry:begin
            case(IR[15:12])
                INST_ADD:begin
                    if(IR[5] == 1'b1)begin
                        R[IR[11:9]] <= R[IR[8:6]] + {{11{IR[4]}},
IR[4:0]};
                    end
                    else begin
                        R[IR[11:9]] <= R[IR[8:6]] + R[IR[2:0]];
                    end
                end
                INST_AND:begin
                    if(IR[5] == 1'b1)begin
                        R[IR[11:9]] <= R[IR[8:6]] & {{11{IR[4]}},
IR[4:0]};
                    end
                    else begin
                        R[IR[11:9]] <= R[IR[8:6]] & R[IR[2:0]];
                    end
                end
                INST_BR:begin
                    if((IR[11] & CC[2]) | (IR[10] & CC[1]) | (IR[9]
& CC[0])) begin
                        PC <= PC + IR[5:0];
                        R[3'b110] <= 16'hFFFF;
                    end
                end
                INST_JMP:begin
                    PC <= R[IR[8:6]][5:0];
                end
                INST_JSR:begin
                    R[7] <= PC;
                    if(IR[11]) PC <= PC + IR[5:0];
                    else PC <= R[IR[8:6]][5:0];
                end
                INST_LD:begin
                    R[IR[11:9]] <= mem[PC + IR[5:0]];
                end
                INST_LDI:begin
                    R[IR[11:9]] <= mem[mem[PC + IR[5:0]][5:0]];
                end
                INST_LDR:begin
                    R[IR[11:9]] <= mem[R[IR[8:6]][5:0]+IR[5:0]];
                end
                INST_LEA:begin
                    R[IR[11:9]] <= PC + IR[5:0];
                end
```

```
                INST_NOT:begin
                    R[IR[11:9]] <= ~R[IR[8:6]];
                end
                INST_ST:begin
                    mem[PC + IR[5:0]] <= R[IR[11:9]];
                end
                INST_STI:begin
                    mem[mem[PC + IR[5:0]][5:0]] <= R[IR[11:9]];
                end
                INST_STR:begin
                    mem[R[IR[8:6]][5:0] + IR[5:0]] <= R[IR[11:9]];
                end
                INST_RSF:begin
                    R[IR[11:9]] <= {1'b0, R[IR[8:6]][7:1]};
                end
                INST_TRAP:begin
                    //PC <= ENDL;
                end
                INST_MOD:begin
                    R[IR[11:9]] <= R[IR[8:6]] % R[IR[2:0]];
                end
                //default: PC <= PC;
            endcase
        end
        endcase
    end


    //七段数码管

    reg [32:0] hexplay_cnt;
    always@(posedge clk) begin
        if (hexplay_cnt >= (2000000 / 8)) hexplay_cnt <= 0;
        else hexplay_cnt <= hexplay_cnt + 1;
    end
    always@(posedge clk) begin
        if (hexplay_cnt == 0)begin
            if (hexplay_an == 7)hexplay_an <= 0;
            else hexplay_an <= hexplay_an + 1;
        end
    end
    always@(*) begin
        case(hexplay_an)
            0: hexplay_data = data[3:0];
            1: hexplay_data = data[7:4];
            2: hexplay_data = data[11:8];
            3: hexplay_data = data[15:12];
            4: hexplay_data = data[19:16];
```

```
            5: hexplay_data = data[23:20];
            6: hexplay_data = data[27:24];
            7: hexplay_data = data[31:28];
        endcase
    end
endmodule
```

约束文件：

```
#CLK100MHZ
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 }
[get_ports { clk }];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports { clk }];
#uart
set_property -dict { PACKAGE_PIN C4 IOSTANDARD LVCMOS33 }
[get_ports { rx }];
#hexplay
set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33     }
[get_ports { hexplay_data[0] }];
set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVCMOS33     }
[get_ports { hexplay_data[1] }];
set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33     }
[get_ports { hexplay_data[2] }];
set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33     }
[get_ports { hexplay_data[3] }];
set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33     }
[get_ports { hexplay_an[0] }];
set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33     }
[get_ports { hexplay_an[1] }];
set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33     }
[get_ports { hexplay_an[2] }];

#Led
set_property -dict { PACKAGE_PIN G18 IOSTANDARD LVCMOS33 }
[get_ports { led[7] }];
set_property -dict { PACKAGE_PIN F18 IOSTANDARD LVCMOS33 }
[get_ports { led[6] }];
set_property -dict { PACKAGE_PIN E17 IOSTANDARD LVCMOS33 }
[get_ports { led[5] }];
set_property -dict { PACKAGE_PIN D17 IOSTANDARD LVCMOS33 }
[get_ports { led[4] }];
set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMOS33 }
[get_ports { led[3] }];
set_property -dict { PACKAGE_PIN E18 IOSTANDARD LVCMOS33 }
[get_ports { led[2] }];
```

```
set_property -dict { PACKAGE_PIN D18 IOSTANDARD LVCMOS33 }
[get_ports { led[1] }];
set_property -dict { PACKAGE_PIN C17 IOSTANDARD LVCMOS33 }
[get_ports { led[0] }];
##Switches

set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 }
[get_ports { sw[0] }]; #IO_L1P_T0_AD0P_15 Sch=jb[1]
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 }
[get_ports { sw[1] }]; #IO_L14N_T2_SRCC_15 Sch=jb[2]
set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 }
[get_ports { sw[2] }]; #IO_L13N_T2_MRCC_15 Sch=jb[3]
set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 }
[get_ports { sw[3] }]; #IO_L15P_T2_DQS_15 Sch=jb[4]
set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 }
[get_ports { sw[4] }]; #IO_L11N_T1_SRCC_15 Sch=jb[7]
set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 }
[get_ports { sw[5] }]; #IO_L5P_T0_AD9P_15 Sch=jb[8]
set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 }
[get_ports { sw[6] }]; #IO_0_15 Sch=jb[9]
set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 }
[get_ports { sw[7] }]; #IO_L13P_T2_MRCC_15 Sch=jb[10]
```