哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2024 秋季

课程名称：数字逻辑设计（实验）

实验名称：综合实验

实验性质：综合设计型

实验学时：6 地点：T2 612

学生班级：计算机与电子通信 7 班

学生学号：2023311704

学生姓名：王昕远

评阅教师：

报告成绩：

实验与创新实践教育中心制

2024 年 10 月

设计的功能描述

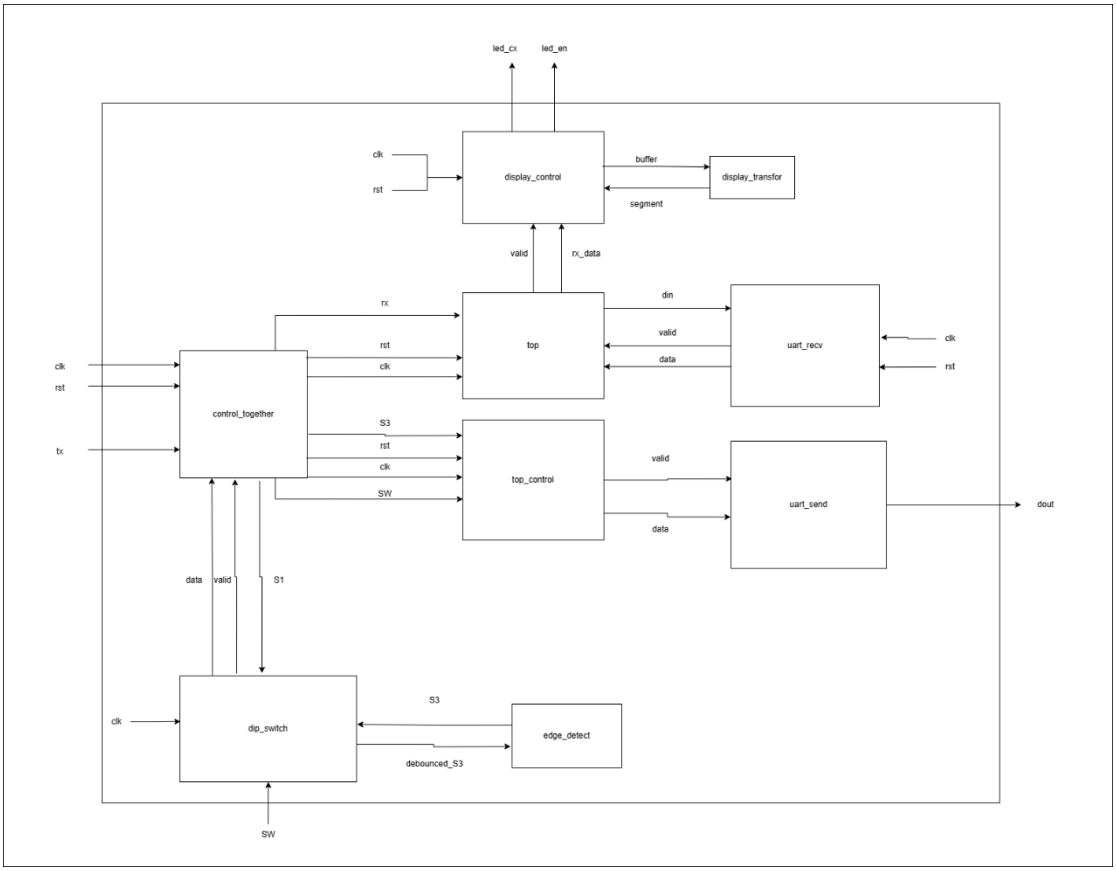
概述基本功能

1．从拨码开关 SW 输入八位数据，通过串口软件显示 ASCII 码对应字符。

2.通过串口文件输入字符，在数码管上面显示。

系统设计

用硬件框图描述系统主要功能及各模块之间的相互关系



Control_together 负责顶层控制，进行电路的连接

Top 是接受控制的顶层模块

Uart_recv 负责接受并解析数据转存为数码管可以识别的数据

Display_control 负责控制数码管的显示

Display_transfor 负责数码管显示的转译

Top_control 负责控制发送的顶层模块

Uart_send 负责向外发送数据

Dip_switch 负责读取拨码开关的数值

Edge_detect 负责检测上升沿

模块设计与实现

（1） dip_switch 读取 SW 拨码开关的输入，将其存入 data 中并且每摁一次 S3 按钮进行一次输出。实现逻辑：S1 进行复位，进行对 S3 的按键去抖动然后检测上升沿表示输入有效。

```
always @(posedge clk or posedge S1)
    if (S1)
    begin
        data <= 0;
        valid <= 0;
    end
    else if (pos_edge)
    begin
        data <= SW;
        valid <= 1;
    end
    else if (valid)
    begin
        valid <= 0;
    end
```

输入信号：clk 时钟信号，SW 拨码按钮，S1 复位，S3 启动
输出信号：valid 数据有效信号，data 数据

（2） edge_detect 边缘上升沿检测，如果是上升沿输出有效。

```
module edge_detect(
    input wire clk,
    input wire rst,
    input wire signal,
    output wire pos_edge
);
```

```
    reg sig_r0, sig_r1, sig_r2;

    always @ (posedge clk or posedge rst)
    begin
        if(rst)
            sig_r0 <= 0;
        else
            sig_r0 <= signal;
    end

    always @ (posedge clk or posedge rst)
    begin
        if(rst)
            sig_r1 <= 0;
        else
            sig_r1 <= sig_r0;
    end

    always @ (posedge clk or posedge rst)
    begin
        if(rst)
            sig_r2 <= 0;
        else
            sig_r2 <= sig_r1;
    end
    assign pos_edge = ~sig_r2 & sig_r1;
endmodule
```

输入信号：clk 时钟，rst 复位，signal 去抖信号

输出信号：pos_edge 上升沿

（3） uart_send 数据发送模块，通过状态机进行发送信号

```
        always @(posedge clk or posedge rst) begin
        if(rst) current_state <= IDLE;
        else if(valid)begin
                current_state <= START;
                tx_data <= data;
        end
        else if(tick)    current_state <= next_state;
        end

        always @(*) begin
        case (current_state)
            IDLE: if(valid) next_state = START;
                    else next_state = IDLE;
```

```
                START:next_state = DATA;
                DATA: if(bit_cnt < 7) next_state = DATA;
                        else if(bit_cnt == 7) next_state = STOP;
                STOP: next_state = IDLE;
            endcase
            end

            always @(posedge clk or posedge rst) begin
            if(rst) begin
                dout <= 1;
                bit_cnt <= 0;
            end
            else begin
                case(current_state)
                    IDLE :      dout <= 1;
                    START :    dout <= 0;
                    DATA : begin
                                dout <= tx_data[bit_cnt];
                                if (tick) bit_cnt <= bit_cnt + 1;
                            end
                    STOP :      dout <= 1;
                endcase
            end
        end
```

重点在于实现状态机，通过三段式来实现状态机。
信号说明：时钟信号 clk，数据有效信号 valid，复位信号 rst，需要发送的数据 data[7:0]，输出信号 dout
定义四个状态：IDLE，START，DATA，STOP
波特率信号:baud tick

（4） uart_recv 数据接收模块，通过状态机对数据进行接收。
重点在于如何实现接受的状态机

```
always @(posedge clk or posedge rst) begin
        if (rst) begin
            state <= IDLE;
            cnt <= 0;
            bit_index <= 0;
            shift_reg <= 0;
            data <= 8'b0;
            valid <= 0;
        end else begin
            case (state)
```

```
                IDLE: begin
                    valid <= 0;
                    cnt <= 0;       // 移动到起始位
                    if (din == 0) begin    // 检测起始位 (din 变低)
                        state <= START;
                        // cnt <= 0;            // 移动到起始位
                    end
                end

                START: begin
                    if (cnt == cnt_half) begin
                        state <= DATA;       // 转换到 DATA 状态
                        cnt <= 0;            // 重置计数器用于数据位
                        bit_index <= 0;
                    end else begin
                        cnt <= cnt + 1;
                    end
                end

                DATA: begin
                    if (cnt == cnt_end) begin
                        shift_reg[bit_index] <= din; // 在中间采样当前位
                        cnt <= 0;            // 重置计数器
                        if (bit_index == 7) begin
                            state <= STOP;           // 所有位接收完成，
进入 STOP 状态
                        end else begin
                            bit_index <= bit_index + 1;
                        end
                    end else begin
                        cnt <= cnt + 1;
                    end
                end

                STOP: begin
                    if (cnt == cnt_end) begin
                        if (din == 1) begin          // 检测有效的停止位
                            data <= shift_reg;       // 存储接收的数据
                            valid <= 1;              // 数据有效标志置为
高
                        end
                        state <= IDLE;               // 返回到 IDLE 状
态
                    end else begin
```

```
                        cnt <= cnt + 1;
                    end
                end

                default: state <= IDLE;
            endcase
        end
    end
```

输入信号：clk 时钟信号，rst 复位信号，din 输出信号，valid 数据有效信号
输出信号：data 数据

（5） display_control 控制数码管的展示，先将数据缓存然后依次进行显示，选择数码管显示，在选择该输入的数据。
难点在于如何将缓存数据进行依次的显示。

```
reg [10:0] count;
reg [1:0] data_flag;
integer i,n,s;
always @(posedge clk or posedge rst)
begin
    if (rst)
    begin
        // 复位时清空缓存
        data_flag <= 0;
        data_former <= 5'h1f;
        data_latter <= 5'h1f;
        count <= 0;
        for (i = 0; i < 8; i = i + 1)
        begin
            buffer[i] <= 5'h1f;
        end
    end
    else if (valid)
    begin
        data_former <= rx_data[7:4];
        data_latter <= rx_data[3:0];
        data_flag <= 1;

    end
    else if (data_flag)
    begin
        for (n = 7; n - 1 > 0; n = n - 1)
        begin
            buffer[n] <= buffer[n-2];
```

```
        end
        count <= count +1;
        buffer[1] <= data_former;
        buffer[0] <= data_latter;
        data_flag <= 0;
      end
   end
```

输入信号：clk 信号，rst 复位，valid 数据有效，rx_data 输入数据
输出信号：led_en 输出的信号管，led_cx 输出的信号

（6） display_transfor 数码管的转码模块

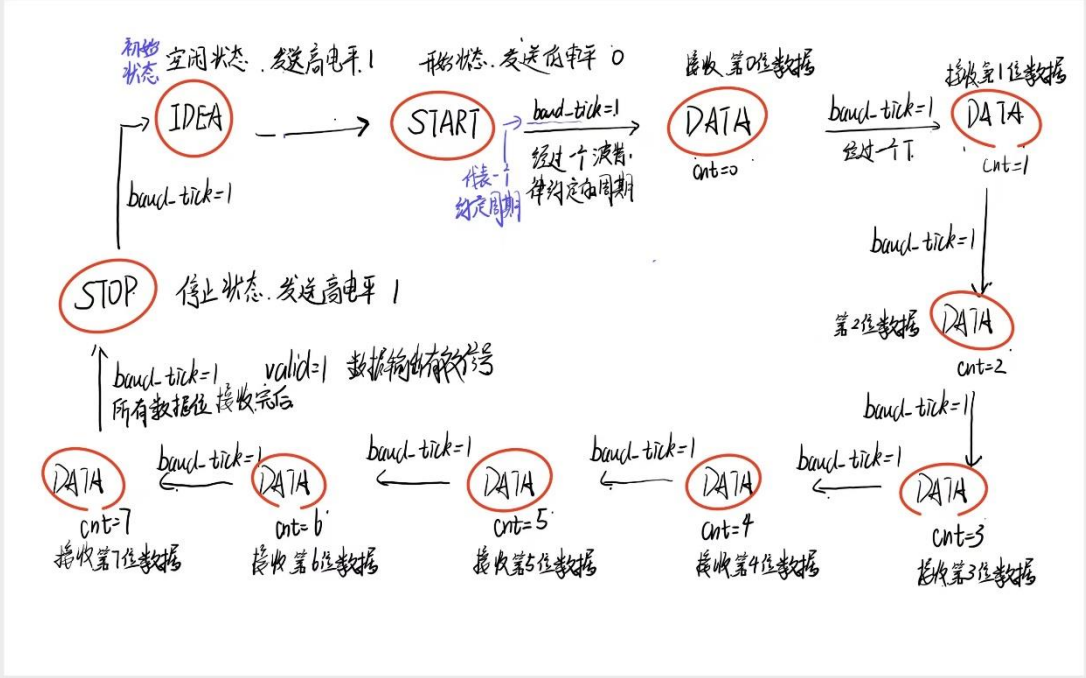
```
module dlsplay_transfor(
      input wire [4:0] digit,
      // input wire flag,
      output reg [7:0] segment
   );

   always @(*)
   begin
      if (digit == 5'h1f)
      segment =    8'b11111111; // blank
      else
      begin
         case (digit)
            4'h0:
               segment = 8'b00000011; // 0
            4'h1:
               segment = 8'b10011111; // 1
            4'h2:
               segment = 8'b00100101; // 2
            4'h3:
               segment = 8'b00001101; // 3
            4'h4:
               segment = 8'b10011001; // 4
            4'h5:
               segment = 8'b01001001; // 5
            4'h6:
               segment = 8'b01000001; // 6
            4'h7:
               segment = 8'b00011111; // 7
            4'h8:
               segment = 8'b00000001; // 8
            4'h9:
```

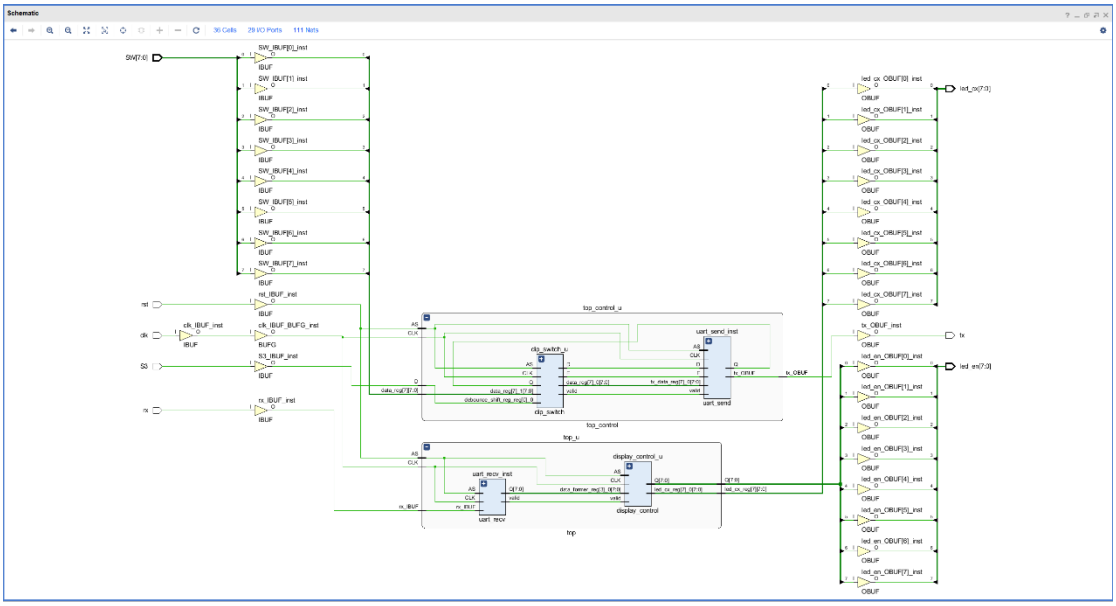
```
                segment = 8'b00001001; // 9
            4'hA:
                segment = 8'b00010001; // A
            4'hB:
                segment = 8'b11000001; // B
            4'hC:
                segment = 8'b11100101; // C
            4'hD:
                segment = 8'b10000101; // D
            4'hE:
                segment = 8'b01100001; // E
            4'hF:
                segment = 8'b01110001; // F
            default:
                segment = 8'b11111110; // blank
        endcase
    end
    // else
    //     segment =    8'b11111111; // blank
  end
endmodule
```

输入信号：digit 信号
输出信号：segment 转码后的数据

接收模块状态转移图

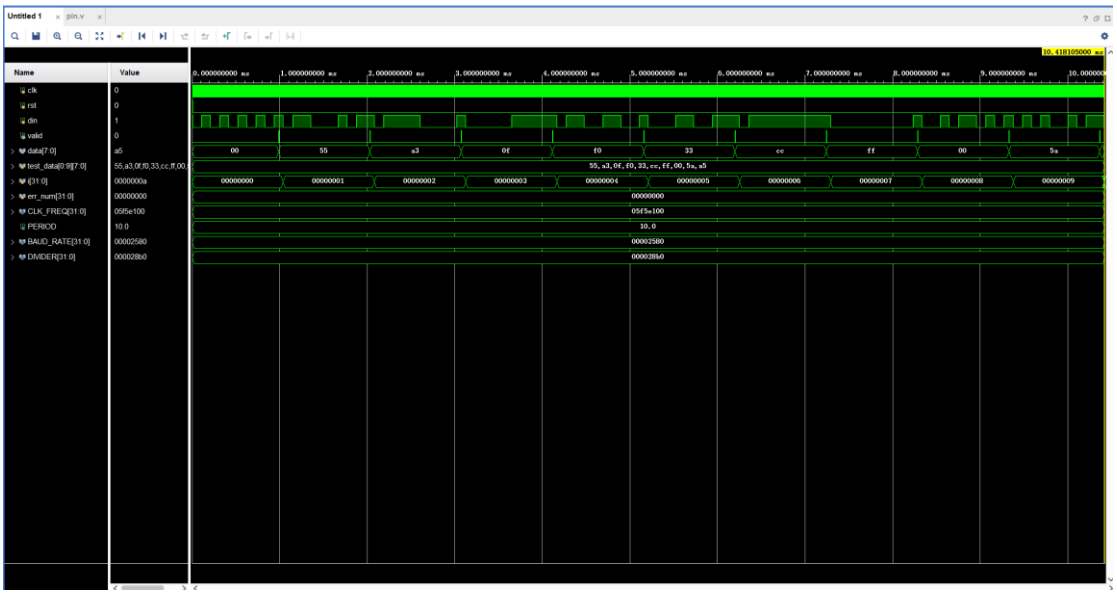


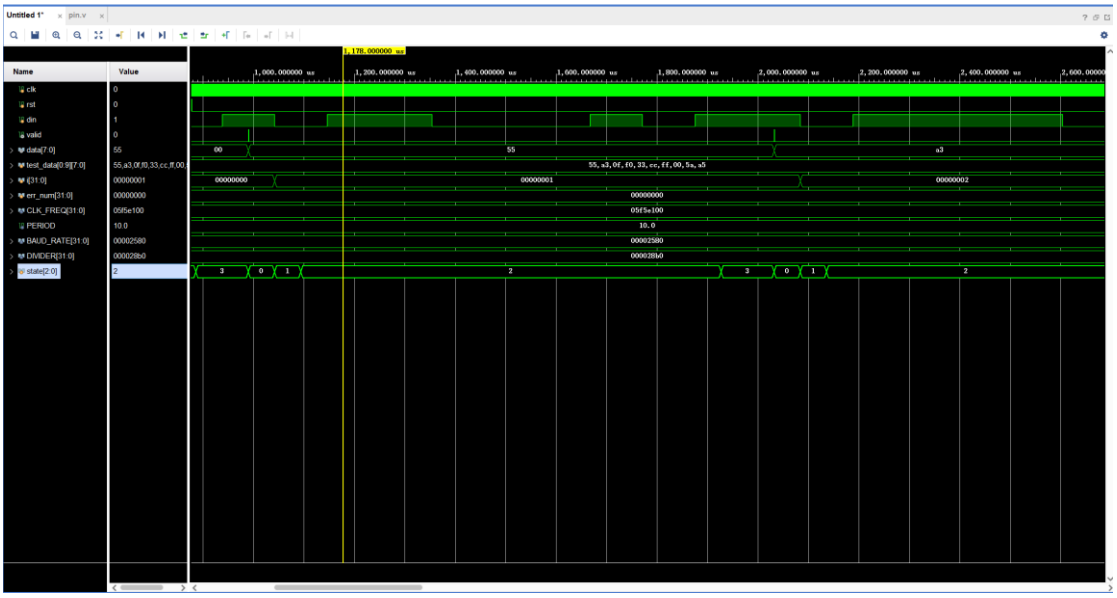
顶层模块的状态分析图



调试报告

仿真波形截图及仿真分析





通过全部十组数据的测试。收到新的数据的时候，经过半个波特率周期状态变为 START（1）状态，然后经过一个周期转变为 DATA（2）状态，在每个输入（din）的中间时刻进行读取数据，读取 8 位数据时进入 STOP（3）状态，停止接受数据，再经过一个周期后

将 valid 输出置为 1 并且状态改为 DELA（0），该次输入结束，data 写入。

设计过程中遇到的问题及解决方法

在实现将输入显示在数码管的功能时，一直存在输入的数据会产生暂存而出现延迟一组显示的问题。

```
reg [4:0] buffer [7:0];   // 存储最近接收的 8 个字符
wire [4:0] data_former, data_latter;
wire [7:0] seg_data [7:0];   // 8 位数码管输出数据
assign data_former = rx_data[7:4];
assign data_latter = rx_data[3:0];
reg [3:0] count;
integer i,n;

always @(*) begin
   // 检测复位信号
   if (rst) begin
      for (i = 0; i < 8; i = i + 1) begin
         buffer[i] = 5'h1f;
      end
      count = 0;
   end else if (valid) begin
      // 右移缓存数据并插入新数据
      for (n = 7; n > 1; n = n - 1) begin
         buffer[n] = buffer[n-2];
      end
```

```
        buffer[1] = data_former;
        buffer[0] = data_latter;
        count = count + 1;
      end
  end
```

寻找错误的地方应该是存在于缓存显示的代码块中，先将是时序逻辑改为组合逻辑以避免延迟的问题，经过调试发现无法解决，然后寻找定义 flag 变量根据 flag 的值来进行再次赋值以消除延时的问题，成功解决。

```
// 更新显示缓存
  reg [10:0] count;
  reg [1:0] data_flag;
  integer i,n,s;
  always @(posedge clk or posedge rst)
  begin
    if (rst)
    begin
      // 复位时清空缓存
      data_flag <= 0;
      data_former <= 5'h1f;
      data_latter <= 5'h1f;
      count <= 0;
      for (i = 0; i < 8; i = i + 1)
      begin
        buffer[i] <= 5'h1f;
      end
    end
    else if (valid)
    begin
      data_former <= rx_data[7:4];
      data_latter <= rx_data[3:0];
      data_flag <= 1;

    end
    else if (data_flag)
    begin
      for (n = 7; n - 1 > 0; n = n - 1)
      begin
```

```
            buffer[n] <= buffer[n-2];
        end
        count <= count +1;
        buffer[1] <= data_former;
        buffer[0] <= data_latter;
        data_flag <= 0;
    end
end
```

课程设计总结

（1）完成耗时 12 小时，写代码 7 小时，写报告 5 小时。

（2）课程收获：对数逻的内部元件有了更深的理解，对 verilog 的熟练程度有很大提高。

建议：希望对实验流程有更详尽的指导，刚上手的难度有点大。