2023FPGA 国赛能力测试题

注：题目二选一即可，所有题目只验收仿真波形。

一、电梯控制电路

设计一个电梯控制电路，要求：

1）大楼总共有 40 层，初始楼层为 1 层，当目标楼层符合条件，且与当前楼层不同时，电梯向目标楼层行进。为方便仿真，每个经过一个 clk，电梯经过一个楼层；

2）电梯只有目标楼层按键和紧急按键。不考虑同时按下多个目标楼层按键的情况。

3）2-9 层为未开发区域，正常情况下不能设定为目标楼层，但是在紧急情况下，按下 2-9 的楼层号后，如果检测到紧急按键 key 的上升沿，此时可以到达这些楼层；

4）电梯限定只能承载 13 人、且不超过 1000Kg，人数或重量超标时均发出警报信号。

- 输入信号：目标楼层 H[7:0]、紧急按键key、当前人数 num[3:0]、重量[9:0](KG)；
- 输出信号：当前楼层 N[7:0]，警报信号alert。

程序头：

```
module elevator#(parameter floor =
    40)( input clk, rst_n,
    input key, [7:0]H,[3:0]num, [9:0]weight,
    output reg [7:0]N,
    output alert
    );
endmodule
```

- **Benchmark**

```
module elevator_tb(
);
reg clk, rst_n, key;
reg [7:0]H;
reg [3:0]num;
reg [9:0]weight;
wire [7:0]N;
wire alert;

elevator test(
    .clk(clk), .rst_n(rst_n),
    .key(key), .H(H), .num(num), .weight(weight),
    .N(N),
    .alert(alert)
);
initial begin
    clk = 0;
    rst_n = 0;
    #15
```

```
        rst_n = 1;
        forever
        #5 clk = ~clk;
end
initial begin
        key = 0;
        H = 0;
        num = 0;
        weight = 0;
        #20
        H = 30;
        num = 1;
        weight = 50;
        #400
        H = 8;
        #10
        key = 1;
        @ (negedge clk) ;key = 0;
        #300
        H = 12;
        num = 14;
        #20
        num = 10;
        weight = 1001;
        #50
        weight = 900;
end
endmodule
```

答案如下：

这是一个针对2023年FPGA国赛能力测试题——“电梯控制电路”的详细设计文档。包含了**全注释的Verilog代码**、**设计原理说明**以及**仿真波形分析**。

一、 详细注释版 Verilog 代码

code Verilog
downloadcontent_copy
expand_less

```
   module elevator#(
   parameter floor = 40 // 参数定义：大楼总层数为40层
)(
   // --- 输入端口 ---
   input          clk,    // 系统时钟信号
   input          rst_n,  // 系统复位信号，低电平有效
   input          key,    // 紧急按键信号
```

```
    input      [7:0] H,       // 目标楼层输入 (Input Floor)
    input      [3:0] num,     // 电梯内当前人数
    input      [9:0] weight,  // 电梯内当前重量 (KG)

    // --- 输出端口 ---
    output reg [7:0] N,       // 电梯当前所在楼层 (Current Floor)
    output           alert    // 警报信号：1表示超载或超员，0表示正常
);

    // --- 内部信号定义 ---
    reg [7:0] target_floor; // 内部寄存器：用于锁存有效的“目的楼层”
    reg       key_d1;       // 延迟寄存器：用于存储上一拍的按键状态
    wire      key_rise;     // 线网：表示按键的上升沿脉冲

    // ============================================================
    // 1. 警报逻辑 (组合逻辑)
    // ============================================================
    // 题目要求：人数 > 13 或 重量 > 1000Kg 时发出警报
    // 这是一个实时检测逻辑，不依赖时钟边沿，立即响应
    assign alert = (num > 4'd13) || (weight > 10'd1000);

    // ============================================================
    // 2. 按键边沿检测逻辑
    // ============================================================
    // 为了检测 key 信号从 0 变 1 的瞬间（上升沿），我们需要记录它的上一个状态
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            key_d1 <= 1'b0;
        else
            key_d1 <= key; // 每个时钟上升沿，将当前 key 值存入 key_d1
    end

    // 当前为高(1) 且 上一刻为低(0) = 上升沿
    assign key_rise = key & (~key_d1);

    // ============================================================
    // 3. 主控制逻辑 (楼层更新与移动)
    // ============================================================
    always @(posedge clk or negedge rst_n) begin
        // --- 复位状态 ---
        if (!rst_n) begin
            N           <= 8'd1; // 初始楼层设为 1
            target_floor <= 8'd1; // 初始目标也设为 1，电梯静止
        end
        else begin
            // ------------------------------------------------
```

```
        // A. 目标楼层设定逻辑 (Target Logic)
        // ----------------------------------------------------
        // 首先判断输入楼层 H 是否在合法的物理范围内 (1-40层)
        if (H >= 8'd1 && H <= floor) begin

            // 情况1：受限区域 (2层 到 9层)
            if (H >= 8'd2 && H <= 8'd9) begin
                // 只有当检测到紧急按键上升沿 (key_rise) 时，才接受这个目标
                if (key_rise) begin
                    target_floor <= H;
                end
                // 如果没有按键，target_floor 保持不变，忽略当前的 H 输入
            end

            // 情况2：正常区域 (1层 或 10-40层)
            else begin
                // 只要输入改变，立即更新目标楼层
                target_floor <= H;
            end
        end

        // ----------------------------------------------------
        // B. 电梯运动逻辑 (Movement Logic)
        // ----------------------------------------------------
        // 安全机制：只有在没有警报 (alert == 0) 时电梯才允许移动
        if (!alert) begin
            // 如果当前楼层 < 目标楼层，向上运行
            if (N < target_floor) begin
                N <= N + 1'b1;
            end
            // 如果当前楼层 > 目标楼层，向下运行
            else if (N > target_floor) begin
                N <= N - 1'b1;
            end
            // 如果 N == target_floor，保持不变 (else 分支省略即保持)
        end
        // 如果 alert == 1，电梯状态保持不变（即悬停在当前楼层或停止运行）
    end
end

endmodule
```

二、 设计说明文档

1. 功能概述

本模块实现了一个简易的40层电梯控制器。它能够根据输入的目标楼层控制电梯上下移动，同时具备特殊楼层（未开发区域）的访问权限控制，以及针对人数和重量的超载安

全报警功能。

2. 端口定义

端口名	方向	位宽	说明
clk	Input	1	**系统时钟**：电梯运行的基准，仿真中每1个clk移动一层。
rst_n	Input	1	**复位信号**：低电平有效，复位后电梯回到1层。
key	Input	1	**紧急按键**：用于解锁2-9层的访问权限。
H	Input	8	**目标楼层**：用户想要去的楼层号。
num	Input	4	**人数**：电梯内当前乘客数量。
weight	Input	10	**重量**：电梯内当前载重 (KG)。
N	Output	8	**当前楼层**：电梯实时所在的位置。
alert	Output	1	**警报**：1=超载/超员，0=正常。

3. 核心逻辑详解

- **状态比较**：系统内部维护一个 target_floor 寄存器。每个时钟周期，系统比较当前楼层 N 与 target_floor 的关系。
    - N < target_floor：电梯上行 (N = N + 1)。
    - N > target_floor：电梯下行 (N = N - 1)。
    - N == target_floor：电梯停止。
- **仿真简化**：题目要求“每个经过一个 clk，电梯经过一个楼层”，因此没有设计加减速或开关门延迟，直接在时钟沿进行加减操作。
- **需求**：2-9层为未开发区域，正常输入无效。只有配合紧急按键 key 才能前往。
- **实现**：
    - 设计了一个边沿检测电路 (key_rise)。
    - 当输入 H 处于 [2, 9] 区间时，逻辑判断条件变严：仅当 key_rise 为高时，才将 H 赋值给 target_floor。
    - 如果 H 在 [2, 9] 区间但没有按键，target_floor 保持旧值，电梯会继续前往之前的目标或停留在原地，实现了“忽略无效请求”的功能。
- **阈值**：人数上限 13人，重量上限 1000KG。
- **逻辑**：alert = (num > 13) || (weight > 1000)。
- **联锁反应**：在 Verilog 代码中，电梯移动逻辑被包裹在 if (!alert) 块中。这意味着一旦触发警报，电梯将立即停止更新 N 的值（即紧急制动/停止运行），直到超载情况解除。

4. 仿真波形预期分析 (基于 Benchmark)

在运行 elevator_tb 仿真时，应观察到以下关键时间点的行为：

1. **复位阶段 (0 - 15ns)**
    - rst_n 为低，N 被强制置为 1，alert 为 0。
2. **正常上行 (20ns - 400ns)**
    - 输入：H=30, num=1, weight=50。
    - 行为：H 是合法楼层，target_floor 更新为 30。
    - 波形：N 从 1 开始，每个时钟周期加 1，呈现阶梯状上升，直到 N=30 停止。
3. **受限区域访问失败 (400ns - 430ns)**
    - 输入：H=8 (受限层), key=0。
    - 行为：由于没有按键信号，target_floor **保持为 30**。

- 波形：N 保持在 30 不变，电梯不动作。

4. **受限区域访问成功 (430ns - 730ns)**
   - 输入：key 产生一个脉冲 (0->1->0)，此时 H 仍为 8。
   - 行为：检测到上升沿，target_floor 更新为 8。
   - 波形：N 开始从 30 递减，每个时钟周期减 1，直到 N=8。
5. **超员/超载报警 (730ns - 800ns)**
   - 输入：H=12。电梯本应向上运行。
   - 事件A：num=14 (超员)。
     - 波形：alert 变高 (1)。电梯停止移动（N 保持不变）。
   - 事件B：num 恢复，但 weight=1001 (超重)。
     - 波形：alert 保持高 (1)。电梯继续停止。
   - 恢复：weight=900。
     - 波形：alert 变低 (0)。电梯恢复运行，N 开始增加，前往 12 层。

二、非重叠序列检测

设计一个串行序列检测器，用来检测序列 011010，要求：

1、以每六个输入为一组，不检测重复序列，例如第一位数据不符合，则不考虑后五位。一直到第七位数据即下一组信号的第一位开始检测。

2、六位数据输入完成后，当信号满足该序列，在最后一位数据输入的下一拍指示信号 match 拉高一个周期，当不满足时，指示信号 not_match 拉高一个周期。

程序头：

```
module
    sequence_detect( input
    clk,
    input rst_n,
    input data,
    output reg match,
    output reg not_match
    );
endmodule
```

- **Benchmark**

```
module testbench();
    reg clk, rst_n, D_in;
    wire match, not_match;
    reg [23:0] data;
    initial begin
        clk=0;
        rst_n=0;
        D_in=0;
        data=24'b011100_011010_011110_011101;
        #5;
        rst_n=1;
    end
always #10 clk<=~clk;
always@(posedge clk)begin
    D_in <= data[23];
    data <= {data[22:0],data[23]};
end
    sequence_detect test(
        .clk(clk),
        .rst_n(rst_n),
        .data(D_in),
        .match(match),
        .not_match(not_match)
    );
Endmodule
```

答案如下：

这是一个符合要求的非重叠序列检测器设计。
该设计采用**计数器+移位寄存器**的架构。由于题目要求“每六个输入为一组”，这是一种典型的块（Block）检测方式，而不是传统的滑动窗口检测。

1. 设计代码 (sequence_detect.v)

code Verilog
downloadcontent_copy
expand_less

```
module sequence_detect(
input      clk,
input      rst_n,
input      data,
output reg match,
output reg not_match
);

// 定义目标序列
localparam [5:0] TARGET_SEQ = 6'b011010;

// 内部计数器，用于记录当前是每组6位数据的第几位 (0~5)
reg [2:0] cnt;
// 移位寄存器，用于缓存输入的序列
reg [5:0] shift_reg;

always @(posedge clk or negedge rst_n) begin
   if (!rst_n) begin
      cnt       <= 3'd0;
      shift_reg <= 6'b0;
      match     <= 1'b0;
      not_match <= 1'b0;
   end
   else begin
      // 默认拉低输出信号（产生脉冲）
      match     <= 1'b0;
      not_match <= 1'b0;

      // 数据移位进入寄存器
      // {shift_reg[4:0], data} 构成了包含当前最新输入在内的6位数据
      shift_reg <= {shift_reg[4:0], data};

      // 计数器逻辑
      if (cnt == 3'd5) begin
         // 一组6位数据接收完毕
         cnt <= 3'd0; // 计数器归零，准备下一组

         // 比较逻辑：检查当前移位寄存器里的值（包含刚进来的data）是否匹配
         // 注意：此时 shift_reg 尚未更新完成，所以要用组合逻辑拼接判断
         if ({shift_reg[4:0], data} == TARGET_SEQ) begin
            match <= 1'b1;     // 匹配，下一拍拉高 match
         end
         else begin
            not_match <= 1'b1; // 不匹配，下一拍拉高 not_match
         end
      end
      else begin
```

```
            // 组内计数
            cnt <= cnt + 1'b1;
        end
    end
end

endmodule
```

2. 仿真测试代码 (testbench.v)
我对题目提供的Benchmark进行了微调（修正了全角引号等语法错误），并添加了波形输出以便观察。
code Verilog
downloadcontent_copy
expand_less

```
`timescale 1ns/1ns

module testbench();

    reg clk;
    reg rst_n;
    reg D_in;

    wire match;
    wire not_match;

    reg [23:0] data; // 存储测试向量

    // 实例化被测模块
    sequence_detect test(
        .clk(clk),
        .rst_n(rst_n),
        .data(D_in),
        .match(match),
        .not_match(not_match)
    );

    // 时钟生成
    initial begin
        clk = 0;
        forever #10 clk = ~clk;
    end

    // 激励逻辑
    initial begin
        rst_n = 0;
        D_in = 0;
        // 测试数据：011100 (Fail), 011010 (Pass), 011110 (Fail), 011101 (Fail)
        data = 24'b011100_011010_011110_011101;

        #5;
        rst_n = 1;

        // 等待数据发送完毕 (24位 * 20ns = 480ns)
        #600;
        $stop;
    end

    // 数据发送逻辑
```

```
    // 每个上升沿发送一位数据，从高位开始
    always @(posedge clk) begin
        if (rst_n) begin
            D_in <= data[23];
            data <= {data[22:0], data[23]}; // 循环移位，或者补0均可
        end
    end

endmodule
```

3. 设计说明与注释

1. 功能概述

本模块用于检测串行输入数据流中是否存在特定的 **6位** 序列 011010。

检测模式为 **非重叠（Non-overlapping）**，即每6个时钟周期作为一个独立的检测窗口。

2. 核心逻辑

- **计数器 (cnt)：**
  - 使用一个模6计数器（0~5）。
  - 当 cnt 为 0 到 4 时，模块仅进行数据移位存储。
  - 当 cnt 为 5 时，表示当前组的最后一位数据到达。此时进行比对，并决定下一周期的输出状态，同时将计数器重置为 0。
- **移位寄存器 (shift_reg)：**
  - 用于串行转并行，保存最近输入的比特流。
- **输出控制：**
  - 题目要求“在最后一位数据输入的下一拍指示信号”。
  - 在 Verilog 中，在 cnt == 5 的时刻赋值 match <= 1，由于是非阻塞赋值，match 信号会在下一个时钟上升沿（即下一组数据的第1位输入时刻）变为高电平，持续一个周期。这完美符合题目时序要求。

3. 题目特殊要求解析

"例如第一位数据不符合，则不考虑后五位。一直到第七位数据即下一组信号的第一位开始检测。"

这句话本质上描述的是 **固定分块（Fixed Block）** 的特性，而不是滑动窗口。

- **实现方式：**通过 cnt 计数器强制每6位对齐。
- **逻辑体现：**无论中间的数据是什么，比较逻辑只在 cnt == 5（每组结束时）触发一次。如果第一位就不对（例如目标是0，来了1），虽然电路仍在移位，但最终在第6位时的比对必然失败，从而触发 not_match，并等待计数器归零后开始下一组。这与题目描述的“不考虑后五位（即不进行中间的滑动匹配）”是一致的。

4. 仿真波形预期

输入数据为 011100_011010_011110_011101 (高位在前发送)。

1. **第1组 (011100)**:
   - 不匹配。
   - 在第6个时钟后，not_match 拉高一个周期。
2. **第2组 (011010)**:
   - **匹配**。
   - 在第12个时钟后，match 拉高一个周期。
3. **第3组 (011110)**:
   - 不匹配。
   - 在第18个时钟后，not_match 拉高一个周期。
4. **第4组 (011101)**:
   - 不匹配。
   - 在第24个时钟后，not_match 拉高一个周期。