2024 FPGA 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、 十二小时制电子钟设计

创作一个 12 小时制的电子钟，带上午/下午指示器。要求如下:

（1） 顶层模块命名为 top。

（2） 输入时钟信号为 i_clk。

（3） 复位信号 i_rst 将电子钟复位至 00:00 am。

（4） 使能信号为 i_ena。

（5） o_pm 为 0 表示上午，o_pm 为 1 表示下午。

（6） o_hh、o_mm 和 o_ss 均为两位的 BCD 码，分别代表小时（00-11）、分钟（00-59）和秒（00-59）。

（7） 复位 i_rst 的优先级高于使能 i_ena，即使未使能也能进行复位。

程序头：

```
module top(
    input           i_clk,
    input           i_rst,
    input           i_ena,
    output          o_pm,
    output [7 : 0] o_hh,
    output [7 : 0] o_mm,
    output [7 : 0] o_ss
    );
endmodule
```

Benchmark：

```
module tb();

reg             i_clk = 0;
reg             i_rst = 1;
reg             i_ena = 1;
wire            o_pm;
wire [7 : 0] o_hh;
wire [7 : 0] o_mm;
wire [7 : 0] o_ss;

top test(
```

```
        .i_clk (i_clk),
        .i_rst (i_rst),
        .i_ena (i_ena),
        .o_pm    (o_pm ),
        .o_hh    (o_hh ),
        .o_mm    (o_mm ),
        .o_ss    (o_ss )
);

always #1 i_clk = ~i_clk;

initial begin
        #3 i_rst <= 0;
        #7 i_rst <= 1;
        #3 i_rst <= 0;
        #1000000;
        $stop;
end

endmodule
```

答案如下：

这是一个带有**详细注释的代码版本**以及一份完整的**设计说明文档**。这份文档旨在帮助你理解代码的每一个细节，包括 BCD 码的处理方式、进位逻辑以及时序控制。

一、 详细注释版 Verilog 代码

code Verilog
downloadcontent_copy
expand_less

```
    module top(
    // --- 端口定义 ---
    input           i_clk,   // 系统时钟信号
    input           i_rst,   // 高电平有效复位信号
    input           i_ena,   // 使能信号，高电平有效
    output          o_pm,    // 上午/下午指示：0=AM, 1=PM
    output [7 : 0] o_hh,     // 小时输出 (BCD 码, 范围 00-11)
    output [7 : 0] o_mm,     // 分钟输出 (BCD 码, 范围 00-59)
    output [7 : 0] o_ss      // 秒钟输出 (BCD 码, 范围 00-59)
);

    // --- 内部寄存器定义 ---
    // 使用寄存器(reg)来存储当前的时间状态
    reg [7:0] r_hh; // 小时寄存器
    reg [7:0] r_mm; // 分钟寄存器
    reg [7:0] r_ss; // 秒钟寄存器
    reg       r_pm; // AM/PM 状态寄存器

    // --- 输出赋值 ---
    // 将内部寄存器的值实时连接到输出端口
    assign o_hh = r_hh;
    assign o_mm = r_mm;
    assign o_ss = r_ss;
    assign o_pm = r_pm;

    // --- 核心计数逻辑 ---
    // 敏感列表包含 i_clk 上升沿和 i_rst 上升沿
    // 这意味着复位是“异步”的，一旦 i_rst 变高，立即复位，不需要等待时钟
    always @(posedge i_clk or posedge i_rst) begin
        // 1. 复位逻辑 (最高优先级)
        if (i_rst) begin
            r_hh <= 8'h00; // 复位时间为 00:00:00
            r_mm <= 8'h00;
            r_ss <= 8'h00;
            r_pm <= 1'b0;  // 复位默认为上午 (AM)
```

```
end
// 2. 使能逻辑 (仅当未复位且 i_ena 为 1 时执行)
else if (i_ena) begin

    // --- 秒计数部分 (00-59) ---
    // 检查秒个位是否到达 9
    if (r_ss[3:0] == 4'd9) begin
        r_ss[3:0] <= 4'd0; // 个位清零

        // 检查秒十位是否到达 5 (即当前是 59 秒)
        if (r_ss[7:4] == 4'd5) begin
            r_ss[7:4] <= 4'd0; // 十位清零，秒数归零

            // --- 分计数部分 (00-59) ---
            // 秒进位，开始处理分钟。检查分个位是否到达 9
            if (r_mm[3:0] == 4'd9) begin
                r_mm[3:0] <= 4'd0; // 分个位清零

                // 检查分十位是否到达 5 (即当前是 59 分)
                if (r_mm[7:4] == 4'd5) begin
                    r_mm[7:4] <= 4'd0; // 分十位清零，分钟归零

                    // --- 时计数部分 (00-11) ---
                    // 分进位，开始处理小时。
                    // 12 小时制特殊逻辑：如果当前是 11 点，下一刻变 00
点
                    if (r_hh == 8'h11) begin
                        r_hh <= 8'h00; // 11 -> 00
                        r_pm <= ~r_pm; // 翻转 AM/PM 状态
                    end
                    // 如果小时个位是 9 (例如 09 点)，下一刻变 10 点
                    else if (r_hh[3:0] == 4'd9) begin
                        r_hh[3:0] <= 4'd0;                    // 个位清零
                        r_hh[7:4] <= r_hh[7:4] + 1'b1; // 十位加 1
                    end
                    // 其他情况，小时个位直接加 1 (例如 00->01, 10->11)
                    else begin
                        r_hh[3:0] <= r_hh[3:0] + 1'b1;
                    end
                end
                // 分十位没满 5，十位加 1
                else begin
                    r_mm[7:4] <= r_mm[7:4] + 1'b1;
                end
```

```
                end
                // 分个位没满 9，个位加 1
                else begin
                    r_mm[3:0] <= r_mm[3:0] + 1'b1;
                end
            end
            // 秒十位没满 5，十位加 1
            else begin
                r_ss[7:4] <= r_ss[7:4] + 1'b1;
            end
        end
        // 秒个位没满 9，个位加 1
        else begin
            r_ss[3:0] <= r_ss[3:0] + 1'b1;
        end
    end
end

endmodule
```

二、 设计说明文档

1. 功能概述

本设计实现了一个基于 FPGA/Verilog 的 12 小时制数字电子钟。该模块能够对输入的时钟信号进行计数，输出当前的小时、分钟和秒，并提供上午（AM）和下午（PM）的状态指示。输出格式采用 **8421 BCD 码**（Binary-Coded Decimal），方便直接驱动数码管显示或与其他十进制系统接口。

2. 端口说明

端口名	方向	位宽	说明
i_clk	Input	1	**系统时钟**：计数的基准信号。
i_rst	Input	1	**复位信号**：高电平有效。优先级最高，强制将时间重置为 00:00:00 AM。
i_ena	Input	1	**使能信号**：高电平有效。当该信号为高时，时钟正常走时；为低时，时间保持暂停。
o_pm	Output	1	**AM/PM 指示**：0 代表上午 (AM)，1 代表下午 (PM)。
o_hh	Output	8	**小时**：BCD 编码，高 4 位为十位，低 4 位为个位。范围 00 ~ 11。
o_mm	Output	8	**分钟**：BCD 编码。范围 00 ~ 59。
o_ss	Output	8	**秒钟**：BCD 编码。范围 00 ~ 59。

3. 核心设计原理

为了方便显示，设计没有使用二进制计数后再转换，而是直接采用 **BCD 码计数**。

- **数据结构**：每个时间单位（时、分、秒）使用 8 位 寄存器。
    - [3:0] (低 4 位)：表示个位 (0-9)。

    - [7:4] (高 4 位)：表示十位 (0-5 或 0-1)。
- **进位规则**：
    - 当低 4 位等于 9 时，低 4 位归零，高 4 位加 1。
    - 当低 4 位等于 9 且高 4 位达到最大值（秒/分为 5，时为 1）时，整个字节归零，并向上一级单位进位。

设计采用了嵌套的 if-else 结构来实现级联：

1. **秒 (Seconds)**：每个时钟周期（当 i_ena=1）增加。当计数到 59 后，下一周期归零，并触发分钟加 1。
2. **分 (Minutes)**：仅在秒从 59 翻转到 00 的时刻增加。当计数到 59 后，下一周期归零，并触发小时加 1。
3. **时 (Hours)**：仅在分从 59 翻转到 00 的时刻增加。

小时的计数逻辑与分秒不同，它是在 00 到 11 之间循环：

- **常规增加**：如 00 -> 01, 09 -> 10。
- **翻转点**：当小时为 11 (BCD: 8'h11) 且分钟进位时：
    - 小时变为 00。
    - o_pm 信号取反（上午变下午，或下午变上午）。
- **异步复位**：代码中使用 always @(posedge i_clk or posedge i_rst)。只要 i_rst 为高，无论时钟沿是否到来，电路立即复位。
- **优先级**：if (i_rst) 位于逻辑的最外层，因此复位优先级高于 i_ena。即使 i_ena 为 0（暂停状态），复位信号依然有效。

4. 仿真波形预期

在 Benchmark 仿真中，你应该能观察到以下现象：

1. **复位**：i_rst 变高时，所有输出瞬间变为 0。
2. **计数**：i_rst 变低且 i_ena 为高后，o_ss 每个时钟周期加 1。
3. **进位**：
    - o_ss 从 59 (hex: 59) 变为 00 时，o_mm 加 1。
    - o_mm 从 59 变为 00 时，o_hh 加 1。
4. **AM/PM 切换**：当 o_hh 为 11 且 o_mm 为 59，o_ss 为 59 时，下一个时钟沿 o_hh 变为 00，同时 o_pm 翻转。

5. 硬件实现注意事项

- **时钟频率**：本设计假设 i_clk 为 1Hz 的脉冲信号（即一秒一个脉冲）。如果在实际 FPGA 开发板上使用（例如 50MHz 晶振），则需要在 top 模块前添加一个分频模块，将 50MHz 分频为 1Hz 后再输入给 i_clk。
- **资源占用**：由于采用了 BCD 编码和简单的加法逻辑，该设计在 FPGA 上的资源占用极低（少量的 LUTs 和 Flip-Flops）。

二、 游戏机自动计费功能设计

实现一个游戏机计费程序的设计。某游戏机具有多个模式，价格不同，要求如下:

（1） 普通模式每分钟 1 元，畅玩模式每分钟收费 2 元，体验模式不计费但是只能体验一次,且时间限制为 5 分钟,体验结束回到普通模式(reset 后体验次数重置)；

（2） 默认情况下为普通模式(boost 为 2'b01)，在 boost 为 2'b10 时进入畅玩模式，在 boost 为 2'b11 时进入体验模式，boost 为 2'b00 时关闭游戏机（等待状态)；

（3） 游戏机采用预付费模式，输入端口 money 的数值为预付费用，在 set 信号有效时，将 money 的数值读入。输出端口 remain 的数值为剩余费用，当费用小于 10 元时，黄色信号灯 yellow 亮起。当费用不足时，红色信号灯 red 亮起;

（4） 在游戏过程中可以通过 set 端口续费。每次 set 信号有效将此时刻 money 的数值加到 remain 之中。

注:在程序中以每个时钟周期代表一分钟，每个单位大小表示 1 元。

程序头：

```
module game_count(
    input rst_n,
    input clk,
    input [9:0]money,
    input set,
    input [1:0]boost,
    output [9:0]remain,
    output yellow,
    output red
);
endmodule
```

Benchmark：

```
    module tbtb();
    reg rst_n;
    reg clk;
    reg [9:0]money;
    reg set;
    reg [1:0]boost;

    wire [9:0]remain;
    wire yellow;
```

```
wire red;

game_count test(
.clk(clk),
.rst_n(rst_n),
.money(money),
.set(set),
.boost(boost),
.remain(remain),
.yellow(yellow),
.red(red)
);

initial begin
clk = 0;
rst_n = 0;

set = 0;
boost = 2'b01;                  //普通模式
money = 10'd0;

#15 rst_n = 1;
#10 money = 10'd15;
#10 set = 1;                    //充值 15 元
#10 set = 0;
#50 boost = 2'b11;              //体验模式

#30    money = 10'd15;
#70    set = 1;                 //再充值 1 次
#10    set = 0;
#50    boost = 2'b10;           //畅玩模式
#50    boost = 2'b11;           //第二次进入体验模式，按题意应当无法进入
#10    boost = 2'b00;           //关机
#20    boost = 2'b01;           //重新进入普通模式一直到红灯亮
end
always #5 clk = ~clk;

endmodule
```

答案如下

这是一个完善的设计文档和带有详细注释的代码。这份文档旨在解释设计思路、状态跳转逻辑以及计费算法，适合作为工程文档或课程设计报告使用。

游戏机自动计费系统设计说明书

1. 功能概述

本模块 game_count 实现了一个基于预付费模式的游戏机计费控制系统。系统根据用户选择的不同模式（普通、畅玩、体验）进行实时扣费，并根据剩余金额控制状态指示灯（黄灯预警、红灯停机）。系统支持运行时动态充值。

2. 端口定义

端口名	方向	位宽	描述
clk	Input	1	系统时钟，1 个时钟周期代表实际时间的 1 分钟
rst_n	Input	1	异步复位信号，低电平有效，用于重置余额和体验状态
money	Input	10	预付/充值金额输入端口
set	Input	1	充值使能信号，高电平有效。有效时将 money 累加至余额
boost	Input	2	模式控制信号： 00: 关机/等待 01: 普通模式 10: 畅玩模式 11: 体验模式
remain	Output	10	当前剩余金额（余额）
yellow	Output	1	余额预警灯，当余额 < 10 时亮起
red	Output	1	余额不足灯，当余额不足以支付当前模式费用时亮起

3. 详细逻辑设计

3.1 模式与费率控制

系统通过组合逻辑根据 boost 信号计算当前的每分钟扣费 (current_cost)：

- **关机模式 (00)**: 费用为 **0 元/分**。
- **普通模式 (01)**: 费用为 **1 元/分**。
- **畅玩模式 (10)**: 费用为 **2 元/分**。
- **体验模式 (11)**:
    - **未体验过**: 前 5 分钟费用为 **0 元/分**。
    - **体验结束**: 超过 5 分钟后，自动切换为普通模式计费 (**1 元/分**)。
    - **状态记忆**: 系统内部维护 trial_used 标志位。一旦体验过（无论是时间耗尽还是中途切换模式），该标志位永久置 1（直到复位），防止用户通过反复切换模式来无限“白嫖”体验时间。

3.2 计费与充值算法

计费逻辑在时钟上升沿触发，遵循以下优先级：

1. **充值优先**: 如果 set 信号有效，先将 money 加入 remain。
2. **扣费判断**: 计算 (当前余额 + 充值金额) 是否大于等于 current_cost。
    - **足够**: 新余额 = 旧余额 + 充值金额 - 本次扣费。

- **不足**: 新余额 = 旧余额 + 充值金额 (不扣费，游戏暂停)。

3. **红灯逻辑**: 只要 remain < current_cost 且机器未关机, 红灯即亮起, 提示用户充值。

3.3 状态指示

- **Yellow**: 纯粹的余额阈值判断 (remain < 10)。
- **Red**: 功能性判断。如果红灯亮起，意味着下一分钟的费用无法支付，计费逻辑会暂停扣款，直到用户充值使余额满足要求。

---

4. 详细代码注释

code Verilog
downloadcontent_copy
expand_less

```
    module game_count(
    // --- 输入端口 ---
    input rst_n,                  // 异步复位，低电平有效
    input clk,                    // 时钟信号 (1 cycle = 1 minute)
    input [9:0] money,            // 充值金额输入
    input set,                    // 充值触发信号
    input [1:0] boost,           // 模式选择信号

    // --- 输出端口 ---
    output reg [9:0] remain,// 剩余金额
    output yellow,                // 余额不足 10 元预警
    output red                     // 余额不足以支付当前费用报警
);

    // --- 内部寄存器定义 ---

    // trial_used: 体验模式状态标志
    // 0: 尚未使用体验资格
    // 1: 体验资格已使用（或已过期）
    reg trial_used;

    // trial_cnt: 体验时间计数器
    // 用于记录在体验模式下度过了多少分钟，最大计数到 5
    reg [2:0] trial_cnt;

    // current_cost: 当前每分钟需要扣除的费用
    // 由组合逻辑根据 boost 和 trial_used 实时计算
    reg [9:0] current_cost;

    //
============================================================
    // 逻辑块 1: 费率计算 (组合逻辑)
    // 功能: 根据输入模式和内部状态，决定当前时刻的扣费标准
```

```
    //
============================================================
    always @(*) begin
        case(boost)
            2'b00: current_cost = 10'd0; // 关机状态，不扣费

            2'b01: current_cost = 10'd1; // 普通模式，1 元/分

            2'b10: current_cost = 10'd2; // 畅玩模式，2 元/分

            2'b11: begin
                // 体验模式特殊逻辑:
                // 如果 trial_used 为 1，说明体验已结束，强制按普通模式(1 元)收费
                // 如果 trial_used 为 0，说明处于体验期内，免费(0 元)
                if (trial_used)
                    current_cost = 10'd1;
                else
                    current_cost = 10'd0;
            end

            default: current_cost = 10'd1; // 默认情况防锁死
        endcase
    end

    //
============================================================
    // 逻辑块 2: 指示灯控制 (组合逻辑)
    // 功能: 实时反馈余额状态
    //
============================================================

    // 黄灯：余额小于 10 元时常亮
    assign yellow = (remain < 10'd10);

    // 红灯：余额不足以支付下一分钟费用时亮起
    // 条件 1: 机器必须是开机状态 (boost != 00)
    // 条件 2: 剩余金额 < 当前所需费用
    assign red = (boost != 2'b00) && (remain < current_cost);

    //
============================================================
    // 逻辑块 3: 核心业务逻辑 (时序逻辑)
    // 功能: 处理复位、体验计时、充值及扣费
    //
```

```
============================================================
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // 复位逻辑：清空余额，重置体验资格
            remain <= 10'd0;
            trial_used <= 1'b0;
            trial_cnt <= 3'd0;
        end else begin
            // ------------------------------------------------
            // A. 体验模式状态机管理
            // ------------------------------------------------

            // 情况 1: 正在进行体验模式 (boost=11) 且 资格有效 (!trial_used)
            if (boost == 2'b11 && !trial_used) begin
                if (trial_cnt < 3'd4) begin
                    // 计数器未满 5 (0,1,2,3 -> 4)，继续计时
                    trial_cnt <= trial_cnt + 1'b1;
                end else begin
                    // 计数器达到 4，意味着当前是第 5 分钟。
                    // 下一分钟开始时，体验资格失效。
                    trial_used <= 1'b1;
                end
            end

            // 情况 2: 防作弊/状态清理
            // 如果用户在体验时间未结束时(trial_cnt > 0)切换到了其他模式(boost != 11)
            // 视为自动放弃剩余体验时间，标记资格已使用。
            if (boost != 2'b11 && trial_cnt > 0) begin
                trial_used <= 1'b1;
            end

            // ------------------------------------------------
            // B. 余额计算逻辑 (支持边玩边充)
            // ------------------------------------------------
            if (set) begin
                // --- 有充值操作 ---
                // 计算逻辑：(当前余额 + 充值金额) 是否够付 (当前费用)
                if ((remain + money) >= current_cost) begin
                    // 够付：更新余额 = 原余额 + 充值 - 扣费
                    remain <= remain + money - current_cost;
                end else begin
                    // 不够付：更新余额 = 原余额 + 充值
                    // (此时红灯会亮，因为 remain < current_cost)
                    remain <= remain + money;
```

```
                end
            end else begin
                // --- 无充值操作 ---
                // 检查当前余额是否足够支付
                if (remain >= current_cost) begin
                    // 够付：正常扣费
                    remain <= remain - current_cost;
                end
                // 不够付：remain 保持不变，等待充值
                // (else 分支隐含：remain <= remain)
            end
        end
    end

endmodule
```