

（深圳）

实验报告

开课学期： 2021秋季

课程名称：数字逻辑设计（实验）

实验名称： 十六进制计算器设计

实验性质： 综合设计型

实验学时： 6 地点： T2506

学生班级： 计算机学院6班

学生学号： 200110619

学生姓名： 梁鑫嵘

评阅教师：

报告成绩：

实验与创新实践教育中心制

2021年12月

注：本设计报告中各个部分如果页数不够，请大家自行扩页，原则是一定要把报告写详细，能说明设计的成果和特色。报告中应该叙述设计中的每个模块。设计报告将是评定每个人成绩的重要组成部分（**设计内容及报告写作**都作为评分依据）。

|  |
| --- |
| 设计的功能描述 |
| 概述基本功能、详细描述自行扩展的功能  本系统设计使用Xilinx® XC7A100T系列FPGA，实现了一个十六进制的计算器，支持加、减、乘、求商，求余，平方六种运算，且支持连续运算功能。 |
| 系统功能详细设计 |
| 用硬件框图描述系统主要功能及各模块之间的相互关系    calculator\_alu  led\_display |
| 各模块描述 |
| 包括模块功能，输入、输出端口、变量含义及主要设计代码  一、模块clk\_div: 由Xilinx® IP Clocking Wizard生成。  1. 输入输出端口：      clk\_div u\_clk\_div (      .clk\_in1  (clk),        *// 输入时钟*      .clk\_out1 (clk\_g),      *// 输出时钟*      .locked   (locked)      *// 时钟是否已经初始化完成，进入稳定状态*      );  二、模块calculator\_hex：负责计算流程的调度  1. 输入输出端口：      calculator\_hex u\_calculator\_hex (      .clk        (clk\_g),    *// 时钟*      .rst        (rst || !(locked)), *// 复位，必须等待时钟初始化完毕*      .button     (button),   *// 开始计算/继续计算信号*      .num1       (num1),     *// 操作数1*      .num2       (num2),     *// 操作数2*      .func       (func),     *// 操作类型*      .cal\_result (cal\_result),   *// 计算结果*      .done       (done)      *// 是否计算完成，可以显示*      );  2. 变量含义      reg calculating;    *// 是否正在计算*      reg calculating2;   *// 上一个周期是否正在计算*      reg done\_reg;       *// 是否计算完*      reg button\_reg;     *// 上一个周期是否发出开始计算指令*      wire alu\_done;      *// ALU是否完成计算*      reg [WIDTH\_NUM1-1:0] alu\_op1;   *// ALU使用的操作数1*      reg [WIDTH\_NUM2-1:0] alu\_op2;   *// ALU使用的操作数2*      reg [WIDTH\_RESULT-1:0] cal\_result\_reg;  *// 计算结果寄存器*      wire [WIDTH\_RESULT-1:0] result; *// ALU计算结果*      assign cal\_result = cal\_result\_reg; *// 连接寄存器和对应线网*      assign done = done\_reg;  3. 主要代码  *// 初始化完成*  *// 记录本时钟状态到寄存器，下周期使用*  calculating2 <= calculating;  button\_reg <= button;  if (calculating && calculating2) begin      if (alu\_done) begin  *// ALU计算完成，设置结果寄存器，*  *// 并且向display模块发送显示信号*          done\_reg <= 1'b1;          cal\_result\_reg <= result;          calculating <= 1'b0;      end  end  else begin      if (done\_reg) begin  *// 控制计算完成信号只发送一个时钟周期*          done\_reg <= 1'b0;      end      else begin          if (button && !button\_reg) begin  *// 在开始计算信号上升沿向ALU发送开始计算信号*              calculating <= 1'b1;              alu\_op1 <= num1;              alu\_op2 <= num2;          end      end  end  三、模块：calculator\_alu负责实际计算功能，可选支持多周期流水计算  1. 输入输出端口  input wire clk,  input wire rst,  input wire start,  input wire [2:0] alu\_op,  input wire [WIDTH\_NUM1-1:0] op1,  input wire [WIDTH\_NUM2-1:0] op2,  output wire done,  output wire [WIDTH\_RESULT-1:0] result  2. 变量含义  *// 计算类型*  parameter OP\_ADD        = 3'b000;  parameter OP\_MINUS      = 3'b001;  parameter OP\_MUL        = 3'b010;  parameter OP\_DIV        = 3'b011;  parameter OP\_MOD        = 3'b100;  parameter OP\_SQUARE     = 3'b101;  *// 结果寄存器初始值*  parameter INIT\_RESULT   = {(WIDTH\_RESULT){1'b0}};  reg started;    *// 是否已经开始计算*  reg done\_reg;   *// 计算完成寄存器*  reg started2;   *// 上一个周期是否已经开始计算*  reg [WIDTH\_RESULT-1:0] result\_reg;  *// 结果寄存器*  reg continuous; *// 是否使用上一次结果进行计算*  wire [WIDTH\_NUM1-1:0] a;    *// 实际计算使用的操作数1*  wire [WIDTH\_NUM2-1:0] b;    *// 实际计算使用的操作数2*  *// 如果已经计算了一次，就使用上一次结果作为操作数1*  assign a = continuous ? result\_reg : op1;  assign b = op2;  *// 寄存器和线网连线*  assign done = done\_reg;  assign result = result\_reg;  3. 主要代码  *// 初始化完成*  *// 记录上一周期start状态*  started2 <= started;  if (start && !started && !started2) begin  *// 开始计算标志*      done\_reg <= 1'b0;      started <= 1'b1;  end  else begin      if (started) begin  *// 当前正在计算，根据alu\_op判断计算类型*  *// `done\_reg <= 1'b1;`表示完成计算*  *// `started <= 1'b0;`清除开始计算标志*  *// `continuous <= 1'b1;`标志下次使用上次结果进行计算*  *// 如此设计是为乘除法等可能出现的多周期计算做准备，*  *// 如果不使用片上DSP，也可以用流水线等方法多周期完成计算*          case (alu\_op)          OP\_ADD: begin              done\_reg <= 1'b1;              continuous <= 1'b1;              started <= 1'b0;              result\_reg <= a + b;          end          OP\_MINUS: begin              done\_reg <= 1'b1;              continuous <= 1'b1;              started <= 1'b0;              result\_reg <= a - b;          end          OP\_MUL: begin              done\_reg <= 1'b1;              continuous <= 1'b1;              started <= 1'b0;              result\_reg <= a \* b;          end          OP\_DIV: begin              done\_reg <= 1'b1;              continuous <= 1'b1;              started <= 1'b0;              result\_reg <= a / b;          end          OP\_MOD: begin              done\_reg <= 1'b1;              continuous <= 1'b1;              started <= 1'b0;              result\_reg <= a % b;          end          OP\_SQUARE: begin              done\_reg <= 1'b1;              continuous <= 1'b1;              started <= 1'b0;              result\_reg <= a \* a;          end          default: begin              done\_reg <= 1'b1;              continuous <= 1'b1;              started <= 1'b0;              result\_reg <= INIT\_RESULT;          end          endcase      end  end  四、模块calculator\_display：数码管显示和结果计算的中间层  1. 输入输出端口  calculator\_display u\_calculator\_display (  .clk        (clk\_g),    *// 时钟*  .rst        (rst || !(locked)), *// 复位，必须等待时钟初始化完毕*  .cal\_result (cal\_result),   *// 计算结果*  .done       (done),     *// 计算完成信号*  .led\_en     (led\_en),   *// led 数码管使能*  .led\_ca     (led\_ca),   *// led 数码管引脚*  .led\_cb     (led\_cb),  .led\_cc     (led\_cc),  .led\_cd     (led\_cd),  .led\_ce     (led\_ce),  .led\_cf     (led\_cf),  .led\_cg     (led\_cg),  .led\_dp     (led\_dp)  );  2. 变量含义  reg [31:0] values;  *// 计算结果寄存器*  wire [7:0] led\_cx;  *// 将led 数码管输入输出端口集合为向量*  reg started;        *// 是否开始*  wire dismiss;       *// 是否忽略显示*  assign dismiss = (rst || (~started) || done);  assign {led\_dp, led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} = dismiss ? (~8'd0) : led\_cx;  3. 主要代码  always @(posedge clk or posedge rst) begin      if (rst) begin          started <= 1'b1;          values  <= 64'b0;      end      else begin          if (done) values <= cal\_result;      end  end  led\_display #(      .DELAY(DELAY)  ) u\_led\_display (      .clk(clk),      .rst(rst),      .values(values),      .led\_cx(led\_cx),      .led\_en(led\_en)  );  五、模块：led\_display数码管显示模块  1. 输入输出端口      input   wire clk,      input   wire rst,      input   wire [WIDTH\_RES-1:0] values,      output  wire [(WIDTH\_RES>>2)-1:0] led\_en,      output  wire [7:0] led\_cx  2. 变量含义      parameter DELAY = 5,        *// 切换使能延时周期数*      parameter WIDTH\_RES = 32,   *// 结果寄存器宽度*      parameter COUNT\_NUM = 8     *// 数码管数量*      reg [31:0] tim;         *// 切换使能延时计时器*      reg [7:0] cnt;          *// 当前显示数码管号*      reg [7:0] map [15:0];   *// 储存数码管显示值和亮灯的映射关系*      wire [3:0] val;         *// 当前应该显示的数码管值*      wire [7:0] val\_map;     *// 当前应该发光的数码管灯*      assign led\_en  = rst ? 8'b0 : ~(1 << cnt);  *// 当前使能数码管*      assign val     = values[(cnt<<2)+:4];       *// 当前数码管值*      assign val\_map = map[val];                  *// 当前数码管亮灯*      assign led\_cx  = ~val\_map;                  *// 因为是阴极所以需要取反*  3. 主要代码      always @ (posedge clk or posedge rst) begin          if (rst) begin  *// 初始化寄存器*              cnt <= 8'b0;              tim <= 32'b0;  *// 初始化数码管显示值和亮灯的映射数据*              map[4'h0] <= 8'b00111111;              map[4'h1] <= 8'b00000110;              map[4'h2] <= 8'b01011011;              map[4'h3] <= 8'b01001111;              map[4'h4] <= 8'b01100110;              map[4'h5] <= 8'b01101101;              map[4'h6] <= 8'b01111101;              map[4'h7] <= 8'b00000111;              map[4'h8] <= 8'b01111111;              map[4'h9] <= 8'b01100111;              map[4'ha] <= 8'b01110111;              map[4'hb] <= 8'b01111100;              map[4'hc] <= 8'b01011000;              map[4'hd] <= 8'b01011110;              map[4'he] <= 8'b01111001;              map[4'hf] <= 8'b01110001;          end          else begin  *// 两层计时器*              if (tim == DELAY) begin                  tim <= 32'b0;                  if (cnt == (COUNT\_NUM-1)) begin                      cnt <= 8'b0;                  end                  else begin                      cnt <= cnt + 8'b1;                  end              end              else begin                  tim <= tim + 32'b1;              end          end      end |
| 调试报告 |
| 仿真波形截图及仿真分析    仿真分析：  1. clk\_div模块工作正常：在clk启动之后，g\_clk生成了正确的10M时钟并且经过一段时间后locked拉起，系统复位完成。  2. 系统计算功能工作正常：  2.1. 按下button后开始计算  2.2. 5个周期后ALU结果计算完毕  2.3. 每次ALU使用一个周期(10M)计算完毕  3. 系统显示功能工作正常：  3.1. led片选正常工作，每隔DELAY切换一次使能信号  3.2. led显示功能正常，每次显示对应结果寄存器内的信号 |
| 设计过程中遇到的问题及解决方法 |
| 在设计过程中遇到了以下问题：  1. 怎样复用之前的模块？  模块复用可以通过模块实例化完成，实例化一个模块即在电路中添加了对应模块和电路连线。  2. 怎样提高模块复用率？  提高代码复用率可以有效地提高代码的鲁棒性和可移植性。在本设计系统中使用的是用parameter完成参数的定制和传递的方法提高代码的复用率，通过修改对应的parameter即可实例化不同适用场景的模块，极大地提高了代码的复用能力。  3. 怎样在电路中完成复杂计算？  仅仅凭简单门电路来完成复杂的计算任务是难度很大的，通常无法在一个周期内计算完成，需要更大的电路面积和更多的时钟周期，使用生成表或者流水线计算的方式才能完成。但是，Xilinx® FPGA为了提高系统设计灵活性，在片内加入了DSP模块等特殊设计的ASIC，其能够被周围的FPGA器件直接使用。在Xilinx® Vivado进行电路综合的时候，综合器会将“\*”“/”“%”等运算符综合成使用DSP模块进行计算的电路。为了使用片上DSP，我们需要降低系统中的其他部分的时钟频率，让DSP能够使用更多时钟周期完成计算工作。在本系统中，系统外部时钟输入为100M，但是控制和显示模块使用的时钟是经过Xilinx® IP Clocking Wizard生成的10M时钟。 |
| 课程设计总结 |
| 包括设计的总结和还需改进的内容以及收获  本次设计的总结：  在本次设计中，代码上使用了模块化参数化的设计，可以轻松地复用几个模块；测试上略微修改testbench，使之可以支持检查led数码管的输出正确性；计算原理上使用灵活的计算方法，可以扩展成多周期计算，增强了计算能力。  还需要改进的内容：  本次设计还有许多细节功能可以添加，如取消前导0、小数运算、按键防抖等，但是本设计暂未添加这些特性。  收获：学习了硬件电路前端的许多设计方法，积累了许多设计经验和技巧。 |