

（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： 10

学生学号： 200111028

学生姓名： 伍文浩

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 |
| 在本次实验实验中，我们按照“自顶向下”的设计思路，设计完成了十六进制计算器。计算器功能如下：  1、清零功能：按下S1按钮，计算器清零，数码管全灭；松开S1按钮，数码管显示八位0，表示清零完毕。  2、计算功能：计算器有如下几个功能：加法、减法、乘法、求商除法、求余除法和乘方。运算功能通过拨码开关SW23-SW21输入，五个计算操作对应的输入分别是000,001,010,011,100,101。其余功能输入为无效输入。  3、操作数输入：除了乘方运算只有一个操作数输入（操作数1），其他操作都有两个操作数，操作数1由SW15-SW8输入、操作数2由SW7-SW0输入。  4、连续运算功能：计算器支持连续运算，在不执行清零操作的情况下，自动将上一次的计算结果复制给操作数1，连续运算时只需要输入操作数2。执行乘方操作时，将直接计算上一次计算结果的平方。  5、求值：计算器并不自动求值，设置好操作数和计算模式后，按下按键S2执行且仅执行一次计算操作。  6、显示功能：计算结果以十六进制的形式显示在数码管上。  7、报错功能：当输入的计算模式未定义时、或者当除法操作的操作数2为0时，计算器报错，数码管显示Error，按键S2失效，按下清零按钮S1恢复。 |
| 系统功能详细设计 |
| 硬件框图如下所示：  硬件框图 |
| 各模块描述 |
| 在本次实验中，我一共设计了四个模块：calculator\_top、calculator\_hex、calculator\_display、calculator\_anti\_shake。其中calculator\_top是主模块，用于统领各个模块、calculator\_anti\_shake用于实现button按键消抖、calculator\_hex是计算模块，实现计算功能、calculator\_display用于控制数码管显示计算结果。各个模块具体介绍如下，这里不calculator\_top做过多赘述。  **calculator\_top:**  如图所示，我在calculator\_top中定义了四条wire类型变量：clk\_g用于表示经过模块u\_clk\_div分频后的时钟信号；error用于记录计算过程中是否出现错误；cal\_result用于记录计算结果；rst\_n是置零信号rst的反。    **calculator\_anti\_shake:**  **变量说明**  module calculator\_anti\_shake(      input wire      clk\_g, 时钟信号（分频后）      input wire      rst\_n, 置零信号      input wire      button, 原button信号      output wire     button\_good 消抖后的button信号  );  **核心代码**  always@(posedge clk\_g or negedge rst\_n) begin      if (~rst\_n)         cnt <= 20'b0;      else if (cnt\_end)   cnt <= 20'b0;      else                cnt <= cnt + 1'b1;  end  always@(posedge clk\_g or negedge rst\_n) begin      if (~rst\_n)         button\_r\_next <= 1'b0;      else if (cnt\_end)   button\_r\_next <= button;  end  calculator\_anti\_shake是按键防抖模块，使用计数器100ms取一次button信号，由此实现按键防抖功能。  **calculator\_hex**  **变量说明**  module calculator\_hex(      input wire          clk\_g   ,      input wire          rst\_n   ,      input wire          button  ,      input wire [7:0]    num1    , 操作数1      input wire [7:0]    num2    , 操作数2      input wire [2:0]    func    , 运算模式      output reg          error   , 记录是否发生错误      output reg [31:0]   cal\_result 计算结果  );  **核心代码**          case (func)              3'b000:         //加法                  cal\_result  <= num1\_ff + num2\_ff;              3'b001:         //减法                  cal\_result  <= num1\_ff - num2\_ff;              3'b010:         //乘法                  cal\_result  <= num1\_ff \* num2\_ff;              3'b011:         //除法求商                  if (num2\_ff == 32'b0)   error       <= 1'b1;                  else                    cal\_result  <= num1\_ff / num2\_ff;              3'b100:         //除法求余                  if (num2\_ff == 32'b0)   error       <= 1'b1;                  else                    cal\_result  <= num1\_ff % num2\_ff;              3'b101:         //乘方                  cal\_result  <= num1\_ff \* num1\_ff;              default: begin                  cal\_result  <= 32'b0;                  error       <= 1'b1;                  end              endcase  calculator\_hex是我们的计算模块，刨去一些用于控制的if-else结构后，其主体部分就只有一个case语句，根据不同的func输入进行计算，并将结果保存到cal\_result中。  **calculator\_display**  **变量说明**  module calculator\_display(      input wire       clk\_g  , // 时钟信号      input wire       rst\_n  , // 清零信号      input wire       error  , // 报错信号      input wire[31:0] cal\_result ,// 计算结果，也是需要显示的结果      output reg [7:0] led\_en , // led使能端      output reg       led\_ca , // led\_cx控制八位数码管的明灭      output reg       led\_cb ,      output reg       led\_cc ,      output reg       led\_cd ,      output reg       led\_ce ,      output reg       led\_cf ,      output reg       led\_cg ,      output wire      led\_dp // 小数点的显示  );  **核心代码**  parameter   ZERO = 7'b1000000, ONE = 7'b1111001, TWO = 7'b0100100,              THREE = 7'b0110000, FOUR = 7'b0011001,FIVE = 7'b0010010,              SIX = 7'b0000010, SEVEN = 7'b1111000, EIGHT = 7'b0000000,              NINE = 7'b0011000, A = 7'b0001000, B = 7'b0000011, C = 7'b0100111,              D = 7'b0100001, E = 7'b0000110, F = 7'b0001110, NONE = 7'b1111111,              r = 7'b0101111, o = 7'b0100011, twcle = 10000;  先预置一些数码管显示需要的常量，用于显示各个数值。              if (error)          //when error occurs                  case(led\_num)                      4'd7:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= NONE;                      4'd6:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= NONE;                      4'd5:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= NONE;                      4'd4:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= E;                      4'd3:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= r;                      4'd2:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= r;                      4'd1:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= o;                      4'd0:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= r;                      default:    {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= NONE;                      endcase              else                  case({cal\_result[4\*led\_num+3],cal\_result[4\*led\_num+2],cal\_result[4\*led\_num+1],cal\_result[4\*led\_num]})                      4'h0:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= ZERO;                      4'h1:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= ONE;                      4'h2:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= TWO;                      4'h3:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= THREE;                      4'h4:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= FOUR;                      4'h5:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= FIVE;                      4'h6:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= SIX;                      4'h7:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= SEVEN;                      4'h8:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= EIGHT;                      4'h9:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= NINE;                      4'hA:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= A;                      4'hB:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= B;                      4'hC:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= C;                      4'hD:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= D;                      4'hE:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= E;                      4'hF:   {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= F;                      default:    {led\_cg, led\_cf, led\_ce, led\_cd, led\_cc, led\_cb, led\_ca} <= NONE;                      endcase  calculator\_display模块是我们的led显示模块。设计思路还是和之前的实验四一样，通过计数器计数使每个七位数码管只显示2ms。主体部分根据当前显示的数码管的编号（led\_num），使用case语句在cal\_result中一次取四位，再根据取值对应的十六进制数设置数码管显示。 |
| 调试报告 |
| 仿真波形截图如下：  仿真波形  波形分析：  如图所示7.195us时设置操作数1和操作数2分别为8’h06和8’h04，运算模式为3’b000，即执行加法运算。7.495us计算完毕，结果为32’h0000000a，运算正确。  8.795us，操作数1为上一次计算结果32’h0000000a，设置操作数2为8’h0e，运算模式为3’b010，即乘法运算。计算结果为32’h00000078，计算正确。  10.395us，操作数1为上一次计算结果32’h00000078，设置操作数2为8’h05，运算模式为3’b101，即乘方操作。计算结果为32’h00003840，计算正确。后续计算过程同理，这里不多赘述。 |
| 设计过程中遇到的问题及解决方法 |
| 本次实验主要遇到了三个问题：  一、如何从普通运算模式切换到连续运算模式。在第一次计算时，操作数1和操作数2都由拨码开关输入，但是在接下来的运算中，操作数1都继承上一次计算的结果，只有操作数2从拨码开关输入。解决方法是          if (state == 1'b0) begin              num1\_ff <= num1;              num2\_ff <= num2;              end          else if (state == 1'b1) begin              num1\_ff <= cal\_result;              num2\_ff <= num2;              end  使用32位寄存器num1\_ff和num2\_ff转存操作数1和2，并且在执行完第一次计算操作后将state赋值为1，此后，num1\_ff从上一次计算结果取值。  二、在上板时，只要button处于按下状态，计算器就会一直进行运算。这一问题出现的原因是，一开始我将button==1作为进行运算的条件，但是没有注意到在按下button时会进行多次计算。解决方法是，使用一个寄存器记录是否应当进行计算，而且把开始计算的判定条件改为松开button并且未进行计算，计算结束后将该变量置为0；    三、消除按键抖动。其实按键防抖本来不是实验要求的内容，个人本来也没有打算做。但是，在上板的过程中发现，由于按键抖动，即使经过上一步的处理，按下button后计算器仍然偶尔进行2~3次计算。于是，我在最终的设计中添加了一个按键防抖的模块，使得最终保证按下一次button计算器仅计算一次。消抖原理很简单，通过查阅资料可知，按键抖动一般不超过10ms，于是我们每100ms检测一次按键状态，将button的值赋给线网button\_good，再使用button\_good作为新的button基本可以消除按键抖动。 |
| 课程设计总结 |
| **设计总结**  这个学期，我们在数字逻辑实验课上设计完成了六个小实验。刚开始的时候“人生地不熟”面对实验内容手足无措，写出了许多幼稚的代码。不过，随着代码量的提升，许多设计最终写起来也慢慢变得得心应手。总的来说，还是得多写代码多上机房。  总的设计上，我尽量写出精简的代码去实现实验的内容，在所有的实验中，每个模块都在100行以内完成了。  **需改进的内容**  对我自身而言，需要改进的内容可能就是，容易把所有功能写到一个always块里面，程序可读性有待提高。然后提一点对课程的小意见，这个学期主要是理论课老师在讲解verilog的语法，个人觉得好像没有怎么起到知识讲解的作用，第一节实验课都有点懵，不知道该写什么。希望可以分配一点理论课的课时到实验上面，然后由实验课老师在实践中讲解verilog的一些语法知识，或许这样我们上手做实验的时候会比较容易上手。  **收获**  verilog是我们学习到的第一门硬件语言，它和我们之前学习到的其他语言不同之处在于：软件上的调试通过只代表了实验成功了一半，还要在上板时跑出相应的功能才意味着实验设计的结束。时常有过了仿真过不了板的情况发生。把硬件和软件结合起来是我们在这门课上学习到的最宝贵的经验。 |