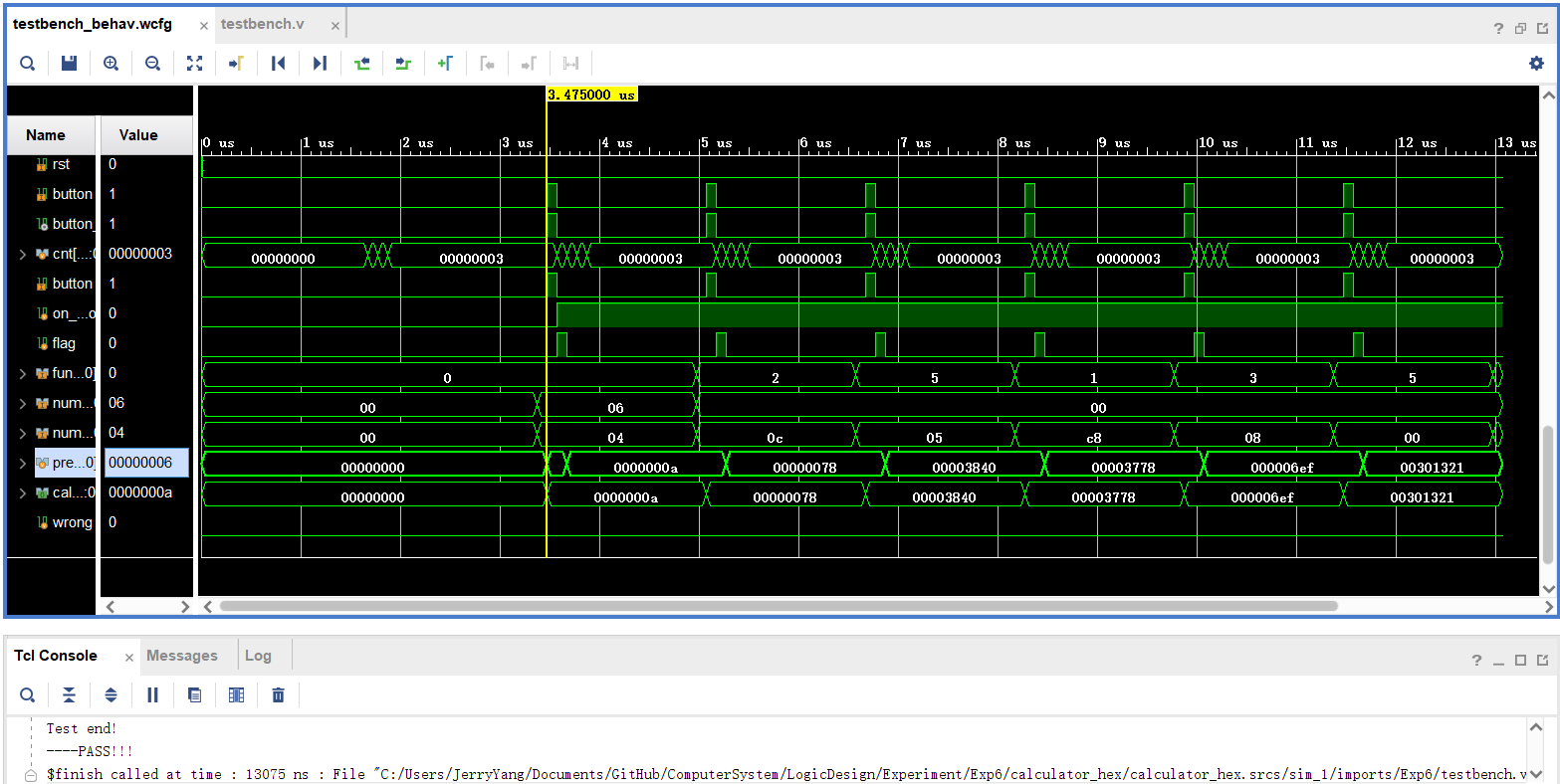
## 实验6 十六进制计算器设计

### calculator\_hex

#### 仿真波形



#### 波形分析

##### 模块定义

**calculator\_hex**实验是实现十六进制计算器的加、减、乘、除、模和平方的功能，默认进行连续运算，模块定义如下：

* **calculator\_top.v** 电路总控制模块，调用各模块功能最后实现运算并显示在数码管上。
* **clk\_div** 时钟IP核，用于时钟分频，分频时钟clk\_g的频率为10MHz
* **key\_filter.v** 消抖模块，用于在上板操作时，避免了按键时不稳定脉冲导致的波形抖动，也避免持续按键导致的非期望的连续计算
* **calculator\_hex.v** 实现计算器核心功能的模块，用于完成加减乘除模平方的运算，默认进行连续运算，按下rst后重置寄存器，每按下一次button进行一次运算
* **calculator\_display.v** 实现将计算器得出的32位2进制数cal\_result转为16进制数在8位数码管上显示

##### 输入输出定义

* **calculator\_top.v**
* module calculator\_top (input wire clk,  
   input wire rst,  
   input wire button,  
   input wire [2:0] func,  
   input wire [7:0] num1,  
   input wire [7:0] num2,  
   output wire [7:0] led\_en,  
   output wire led\_ca,  
   output wire led\_cb,  
   output wire led\_cc,  
   output wire led\_cd,  
   output wire led\_ce,  
   output wire led\_cf,  
   output wire led\_cg,  
   output wire led\_dp);  
   ...  
  endmodule
* **key\_filter.v**
* module key\_filter(input wire rst,  
   input wire clk,  
   input wire button,  
   output reg button\_f);  
   ...  
  endmodule
* **calculator\_hex.v**
* module calculator\_hex (input wire clk,  
   input wire locked,  
   input wire rst,  
   input wire button,  
   input wire [2:0] func,  
   input wire [7:0] num1,  
   input wire [7:0] num2,  
   output reg [31:0] cal\_result);  
   ...  
  endmodule
* **calculator\_display.v**
* module calculator\_display(input wire clk,  
   input wire rst,  
   input wire locked,  
   input wire button,  
   input wire [31:0] cal\_result,  
   output reg [7:0] led\_en,  
   output wire [7:0] led\_w);  
   ...  
  endmodule

##### 信号定义

只对模块中自定义信号和部分给定信号进行说明

**calculator\_top.v**

wire clk\_g ; // 分频时钟  
 wire locked ; // 时钟锁定信号，当分频时钟输出稳定后为高电平  
 wire button\_f ; // button消除抖动后的信号，作为其他模块的button输入信号  
 wire on\_button ; // 开始计算后处于工作状态的标记信号  
 wire [31:0] cal\_result; // 传递32位二进制计算结果的信号  
 wire [7:0] led\_w ; // \_w代表wire类型，数码管段选信号

**key\_filter.v**

// parameter CNT\_MAX = 32'd1000; // 计数器最大数值，控制两次按键时间间隔  
 parameter CNT\_MAX = 32'd3; // simulation  
 reg [31:0] cnt = 0 ; // 两次按键最小时间间隔计数，用于消除毛刺信号  
 wire rst\_n = ~rst; // rst\_n下降沿复位

**calculator\_hex.v**

wire rst\_n = ~rst; // rst\_n下降沿复位   
 reg on\_button = 0 ; // 开始计算后处于工作状态的标记信号  
 reg flag = 0 ; // 用于标记按下button后的第一个时钟周期  
 reg [31:0] prev\_result = 0 ; // 连续运算时，用于保存中间计算结果

**calculator\_display.v**

// parameter SCAN\_CNT\_MAX = 20'd1\_0000; // 扫描时间间隔计数最大时钟周期数  
 parameter SCAN\_CNT\_MAX = 20'd5; // simulation  
 wire rst\_n = ~rst ; // rst\_n下降沿复位  
 reg on\_button = 0 ; // 开始计算后，处于工作状态的标记信号  
 reg [7:0] cur\_code = 8'hff; // 当前结果位置数值显示到数码管上对应的编码  
 reg [20:0] scan\_cnt = 0 ; // 扫描计数器  
 reg [2:0] scan\_pos = 0 ; // 当前扫描的位置  
 wire [3:0] result [7:0] ; // 重新分割32位2进制计算结果的位，分割成8个4位数值，只是为了人直接的阅读方面

##### 主要代码

* 核心模块调用代码 **calculator\_top.v**
* module calculator\_top (input wire clk,  
   input wire rst,  
   input wire button,  
   input wire [2:0] func,  
   input wire [7:0] num1,  
   input wire [7:0] num2,  
   output wire [7:0] led\_en,  
   output wire led\_ca,  
   output wire led\_cb,  
   output wire led\_cc,  
   output wire led\_cd,  
   output wire led\_ce,  
   output wire led\_cf,  
   output wire led\_cg,  
   output wire led\_dp);  
     
   wire clk\_g ; // 分频时钟  
   wire locked ; // 时钟锁定信号，当分频时钟输出稳定后为高电平  
   wire button\_f ; // button消除抖动后的信号，作为其他模块的button输入信号  
   wire on\_button ; // 开始计算后处于工作状态的标记信号  
   wire [31:0] cal\_result; // 传递32位二进制计算结果的信号  
   wire [7:0] led\_w ; // \_w代表wire类型，数码管段选信号  
     
   assign {led\_dp,led\_cg,led\_cf,led\_ce,led\_cd,led\_cc,led\_cb,led\_ca} = led\_w;  
    
   clk\_div u\_clk\_div (  
   .clk\_in1 (clk),  
   .clk\_out1 (clk\_g),  
   .locked (locked)  
   );  
     
   key\_filter u\_key\_filter(  
   .rst(rst),  
   .clk(clk\_g),  
   .button(button),  
   .button\_f(button\_f)  
   );  
     
   calculator\_hex u\_calculator\_hex (  
   .clk (clk\_g),  
   .rst (rst), //add your own code  
   .locked(locked),  
   .button(button\_f),  
   .func(func),  
   .num1(num1),  
   .num2(num2),  
   .cal\_result (cal\_result)  
   );  
     
   calculator\_display u\_calculator\_display (  
   .clk(clk\_g),  
   .rst(rst),  
   .locked(locked),  
   .button(button\_f),  
   .cal\_result (cal\_result),  
   .led\_en(led\_en),  
   .led\_w(led\_w)  
   );  
     
  endmodule
* 计算器功能实现代码**calculator\_hex.v**
* module calculator\_hex (input wire clk,
    
   input wire locked,
    
   input wire rst,
    
   input wire button,
    
   input wire [2:0] func,
    
   input wire [7:0] num1,
    
   input wire [7:0] num2,
    
   output reg [31:0] cal\_result);
    
    
   wire rst\_n = ~rst; // rst\_n下降沿复位
    
   reg on\_button = 0 ; // 开始计算后处于工作状态的标记信号
    
   reg flag = 0 ; // 用于标记按下button后的第一个时钟周期
    
   reg [31:0] prev\_result = 0 ; // 连续运算时，用于保存中间计算结果
    
    
   always @(posedge clk, negedge rst\_n) begin // on\_button, prev\_result
    
   if (~rst\_n || !locked)
    
   on\_button <= 0;
    
   else if (button)
    
   on\_button <= 1;
    
   else
    
   on\_button <= on\_button;
    
   end
    
   always @(posedge clk, negedge rst\_n) begin
    
   if (~rst\_n)
    
   flag <= 0;
    
   else if (button)
    
   flag <= 1;
    
   else
    
   flag <= 0;
    
   end
    
   always @(posedge clk, negedge rst\_n) begin
    
   if (~rst\_n)
    
   prev\_result <= 0;
    
   else if (!on\_button)
    
   // prev\_result <= (func == 3'd5) ? 0 : num1; // not on\_button, prev square = 0, else prev = num1
    
   prev\_result <= num1; // new requirement given during class
    
   else if (on\_button && flag)
    
   prev\_result <= cal\_result;
    
   else
    
   prev\_result <= prev\_result;
    
   end
    
   always @(\*) begin // cal\_result
    
   if (~rst\_n)
    
   cal\_result = 0;
    
   else if (button)
    
   case(func)
    
   3'd0: cal\_result = prev\_result + num2; // add
    
   3'd1: cal\_result = prev\_result - num2; // substract
    
   3'd2: cal\_result = prev\_result \* num2; // time
    
   3'd3: cal\_result = prev\_result / num2; // divide
    
   3'd4: cal\_result = prev\_result % num2; // mod
    
   3'd5: cal\_result = prev\_result \* prev\_result; // square
    
   default: cal\_result = 0;
    
   endcase
    
   else
    
   cal\_result = cal\_result;
    
   end
    
  endmodule
* 数码管显示代码**calculator\_display.v**
* module calculator\_display(input wire clk,
    
   input wire rst,
    
   input wire locked,
    
   input wire button,
    
   input wire [31:0] cal\_result,
    
   output reg [7:0] led\_en,
    
   output wire [7:0] led\_w);
    
   // parameter SCAN\_CNT\_MAX = 20'd1\_0000; // 扫描时间间隔计数最大时钟周期数
    
   parameter SCAN\_CNT\_MAX = 20'd5; // simulation
    
    
   wire rst\_n = ~rst ; // rst\_n下降沿复位
    
   reg on\_button = 0 ; // 开始计算后，处于工作状态的标记信号
    
   reg [7:0] cur\_code = 8'hff; // 当前结果位置数值显示到数码管上对应的编码
    
   reg [20:0] scan\_cnt = 0 ; // 扫描计数器
    
   reg [2:0] scan\_pos = 0 ; // 当前扫描的位置
    
   wire [3:0] result [7:0] ; // 重新分割32位2进制计算结果的位，分割成8个4位数值，只是为了人直接的阅读方面
    
    
   generate
    
   genvar i;
    
   for (i = 0; i< 8; i = i + 1) begin: result\_block
    
   assign result[i] = cal\_result[i\*4 +: 4]; // just for read convenience
    
   end
    
   endgenerate
    
   always @(posedge clk, negedge rst\_n) begin // on\_button
    
   if (~rst\_n || !locked)
    
   on\_button <= 0;
    
   else if (button)
    
   on\_button <= 1;
    
   else
    
   on\_button <= on\_button;
    
   end
    
   always @(posedge clk, negedge rst\_n) begin // scan\_cnt
    
   if (~rst\_n || !on\_button)
    
   scan\_cnt <= 0;
    
   else if (on\_button) begin
    
   if (scan\_cnt == SCAN\_CNT\_MAX-1)
    
   scan\_cnt <= 0;
    
   else
    
   scan\_cnt <= scan\_cnt + 1;
    
   end
    
   else
    
   scan\_cnt <= 0;
    
   end
    
   always @(posedge clk, negedge rst\_n) begin // scan\_pos
    
   if (~rst\_n || !on\_button)
    
   scan\_pos <= 0;
    
   else if (on\_button && scan\_cnt == SCAN\_CNT\_MAX-1)
    
   scan\_pos <= scan\_pos + 1;
    
   else if (on\_button && scan\_cnt < SCAN\_CNT\_MAX-1)
    
   scan\_pos <= scan\_pos;
    
   else
    
   scan\_pos <= 0;
    
   end
    
   always @(posedge clk, negedge rst\_n) begin // led\_en
    
   if (~rst\_n || !on\_button)
    
   led\_en <= 8'd255;
    
   else if (on\_button)
    
   case (scan\_pos)
    
   3'd0:led\_en <= ~8'd1;
    
   3'd1:led\_en <= ~8'd2;
    
   3'd2:led\_en <= ~8'd4;
    
   3'd3:led\_en <= ~8'd8;
    
   3'd4:led\_en <= ~8'd16;
    
   3'd5:led\_en <= ~8'd32;
    
   3'd6:led\_en <= ~8'd64;
    
   3'd7:led\_en <= ~8'd128;
    
   endcase
    
   else
    
   led\_en <= 8'd255;
    
   end
    
   always @(posedge clk, negedge rst\_n) begin // cur\_code
    
   if (~rst\_n || !on\_button)
    
   cur\_code <= 8'hff;
    
   else if (on\_button) begin
    
   case(result[scan\_pos])
    
   4'h0: cur\_code <= 8'hc0;
    
   4'h1: cur\_code <= 8'hf9;
    
   4'h2: cur\_code <= 8'ha4;
    
   4'h3: cur\_code <= 8'hb0;
    
   4'h4: cur\_code <= 8'h99;
    
   4'h5: cur\_code <= 8'h92;
    
   4'h6: cur\_code <= 8'h82;
    
   4'h7: cur\_code <= 8'hf8;
    
   4'h8: cur\_code <= 8'h80;
    
   4'h9: cur\_code <= 8'h98;
    
   4'ha: cur\_code <= 8'h88;
    
   4'hb: cur\_code <= 8'h83;
    
   4'hc: cur\_code <= 8'hc6;
    
   4'hd: cur\_code <= 8'ha1;
    
   4'he: cur\_code <= 8'h86;
    
   4'hf: cur\_code <= 8'h8e;
    
   endcase
    
   end
    
   else
    
   cur\_code <= 8'hff;
    
   end
    
    
   assign led\_w =
    
   (~rst\_n || !on\_button) ? 8'hff :
    
   on\_button ? cur\_code :
    
   8'hff; // using operator ?: directly instead of if-else statement
    
  endmodule
* 消抖模块代码**key\_filter.v**
* `timescale 1ns / 1ps
    
    
  module key\_filter(input wire rst,
    
   input wire clk,
    
   input wire button,
    
   output reg button\_f);
    
   // parameter CNT\_MAX = 32'd1000; // 计数器最大数值，控制两次按键时间间隔
    
   parameter CNT\_MAX = 32'd3; // simulation
    
    
   reg [31:0] cnt = 0 ; // 两次按键最小时间间隔计数，用于消除毛刺信号
    
   wire rst\_n = ~rst; // rst\_n下降沿复位
    
    
   always @(\*) begin
    
   if (~rst\_n)
    
   button\_f = 0;
    
   else if (cnt == CNT\_MAX && button)
    
   button\_f = 1;
    
   else
    
   button\_f = 0;
    
   end
    
   always @(posedge clk, negedge rst\_n) begin
    
   if (~rst\_n)
    
   cnt <= 0;
    
   else if (button) begin
    
   if (cnt == CNT\_MAX)
    
   cnt <= 0;
    
   else
    
   cnt <= cnt;
    
   end
    
   else begin
    
   if (cnt == CNT\_MAX)
    
   cnt <= cnt;
    
   else
    
   cnt <= cnt + 1;
    
   end
    
   end
    
  endmodule

##### 波形时序分析

* 对于第一次计算，相较于接下来的计算稍微特殊，在时钟分频之后，分频时钟的频率为10MHz，时钟周期为100ns

| * clk(ns) | * rst | * button | * on\_button | * flag | * func | * num1 | * num2 | * prev\_result | * cal\_result |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| * 0-20(+) | * 1=>0 | * 0 | * 0 | * 0 | * 0 | * 0 | * 0 | * 0 | * 0 |
| * 20(+)-3375(+) | * 0 | * 0 | * 0 | * 0 | * 0 | * 0=>'h06 | * 0=>'h04 | * 0 | * 0 |
| * 3375(+)-3475(+) | * 0 | * 0=>1 | * 0 | * 0 | * 0 | * 'h06 | * 'h04 | * 0=>'h6 | * 0=>'ha |
| * 3475(+)-3575(+) | * 0 | * 1=>0 | * 0=>1 | * 0=>1 | * 0 | * 'h06 | * 'h04 | * 'h6 | * 'ha |
| * 3575(+)-3675(+) | * 0 | * 0 | * 1 | * 1=>0 | * 0 | * 'h06 | * 'h04 | * 'h6=>'ha | * 'ha |
| * 3675(+)-4975(+) | * 0 | * 0 | * 1 | * 0 | * 0=>'h2 | * 'h06=>0 | * 'h04=>'h0c | * 'ha | * 'ha |

* 由上表可见，在开始第一次计算之前（3375ns之前，之后只更新了num1和num2信号的值），对应各数值均为0，rst之后的状态就是此时的状态
  + 当第一次按下button时（3475ns），在当前时钟周期由时序逻辑控制的prev\_result将num1数值存入，组合逻辑控制的cal\_result将prev\_result和num2进行运算，得到结果'ha
  + 在第一次按下button后的第一个时钟上沿到来的时候（3575ns），运算结果稳定一周期，该周期由上一周期on\_button尚未修改确定（即标记了第一次按下button前的时间），对应到代码如下：
  + always @(posedge clk, negedge rst\_n) begin  
     if (~rst\_n)  
     prev\_result <= 0;  
     else if (!on\_button)  
     prev\_result <= num1; // new requirement given during class  
     else if (on\_button && flag)  
     prev\_result <= cal\_result;  
     else  
     prev\_result <= prev\_result;  
     end
  + 在第一次按下button后的第二个时钟上沿到来的时候（3675ns），prev\_result更新为先前计算结果，flag信号上一周期被更改的1确定。
* 对于此后的连续运算过程，与第一次计算无异，但prev\_result的更新只由flag高电平的1个时钟周期控制
* 波形给出的计算过程如下：

| * cnt | * func | * prev\_result | * num1 | * num2 | * cal\_result |
| --- | --- | --- | --- | --- | --- |
| * 0 | * 0 | * 'h00000006 | * 'h06 | * 'h04 | * 'h0000000a |
| * 1 | * 'h2 | * 'h0000000a | * 0 | * 'h0c | * 'h00000078 |
| * 2 | * 'h5 | * 'h00000078 | * 0 | * 'h05 | * 'h00003840 |
| * 3 | * 'h1 | * 'h00003840 | * 0 | * 'hc8 | * 'h00003778 |
| * 4 | * 'h3 | * 'h00003778 | * 0 | * 'h08 | * 'h000006ef |
| * 5 | * 'h5 | * 'h000006ef | * 0 | * 'h00 | * 'h00301321 |

* 对应于十进制计算过程如下：

| * cnt | * func | * prev\_result | * num1 | * num2 | * cal\_result |
| --- | --- | --- | --- | --- | --- |
| * 0 | * 加法 | * 6 | * 6 | * 4 | * 10 |
| * 1 | * 乘法 | * 10 | * 0 | * 12 | * 120 |
| * 2 | * 平方 | * 120 | * 0 | * 5（与结果无关） | * 14400 |
| * 3 | * 减法 | * 14400 | * 0 | * 200 | * 14200 |
| * 4 | * 除法 | * 14200 | * 0 | * 8 | * 1775 |
| * 5 | * 平方 | * 1775 | * 0 | * 0 | * 3150625 |

综上所述，本次实验完整的实现了十六进制计算器的基本功能，同时使用按键消抖模块避免了毛刺信号对按键的干扰，也避免了长按时对按键的错误判断。仿真通过，上板验证良好，说明本实验完成良好。