哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

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：计算机类 4 班

学生学号：200110428

学生姓名：杨杰睿

评阅教师：郑海刚

报告成绩：

实验与创新实践教育中心制

2021 年 12 月

设计的功能描述

基本功能

实现十六进制计算器基本功能，包含加、减、乘、除、取模、平方。输入数值 num1 和 num2 由[15:8], [7:0]位拨码开关给出，运算的种类由[23:21]位拨码开关给定。

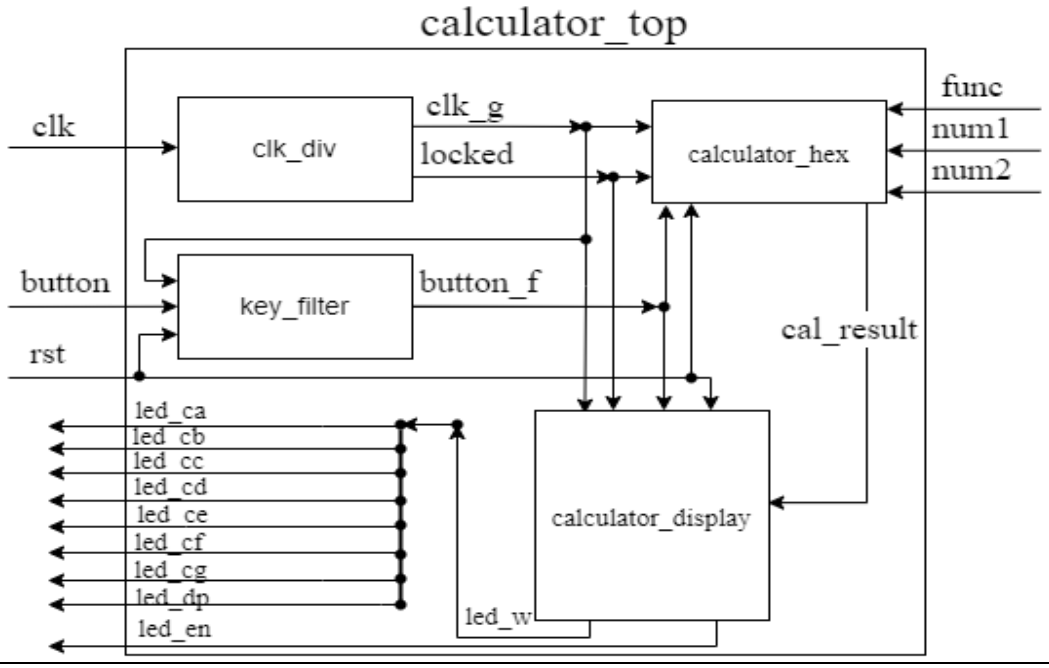
扩展功能

按键消抖模块 key_filter.v，该模块功能为实现按键消抖，防止按键时持续的周期造成运算不受控制的进行。设计逻辑经两轮迭代，初版消抖模块基本实现消抖功能，但设计粗糙，对于长按没有进行处理；终版提交的消抖模块，正确完整的实现了消抖功能，避免了按键时信号的毛刺对结果的影响，也避免了按键持续时非期望连续计算的进行。

该按键消抖模块具有通用性，可方便的复用到其他应用到的模块中。

系统功能详细设计

硬件框图



各模块描述

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 (...);
    ...
    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 (...);
    ...
    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 <= num1;  // new requirement given durin
g 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; // subs
                3'd2: cal_result    = prev_result * num2; // time
                3'd3: cal_result    = prev_result / num2; // divi
                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
```

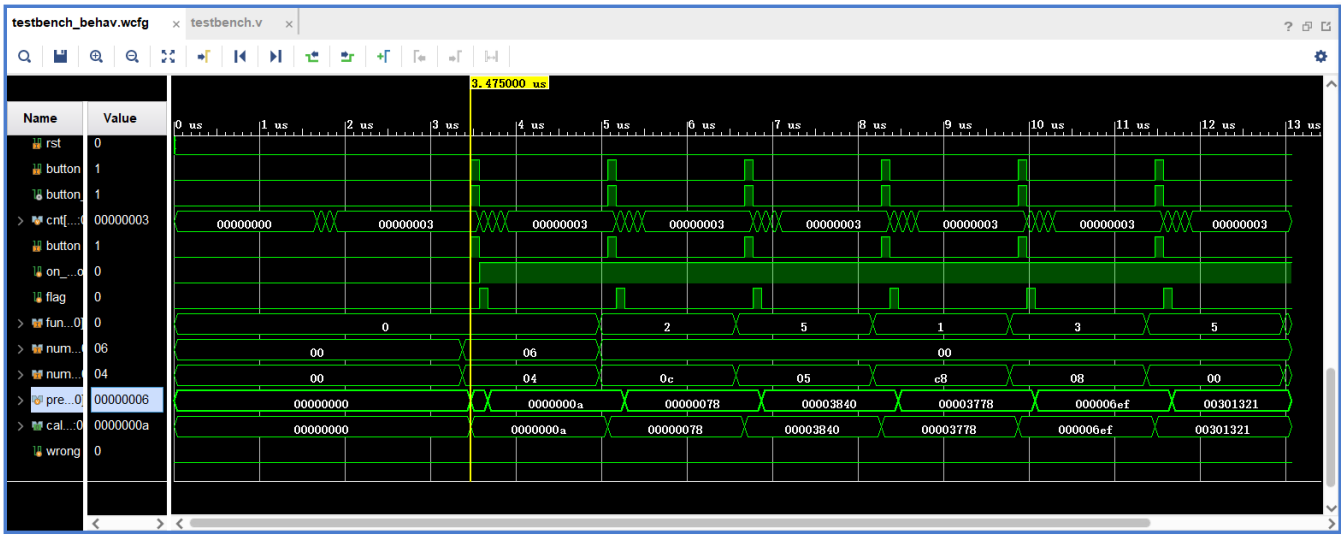
- 消抖模块代码 **key_filter.v**

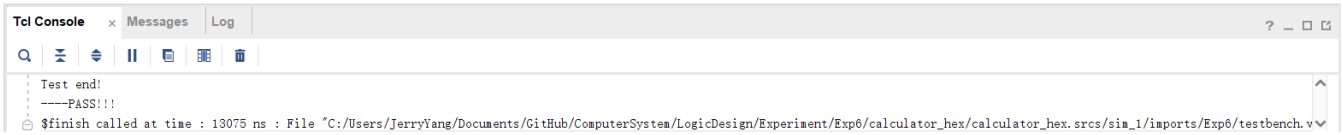
```
`timescale 1ns / 1ps

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
```

实验6 十六进制计算器设计 调试报告

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

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
```

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。

设计过程中遇到的问题及解决方法

遇到的问题级解决方案

事实上，本实验难度适中，仅代码量相比此前稍大（总计约 260 行），需要提前梳理好电路运行的逻辑，按部就班写完代码仿真上板即可完成实验。

在理解题意的方面上，出现了一点小的偏差，导致 1-2h 在对一小部分 testbench 的异常进行排错。事实上，所给定的 testbench 只对前 6 次进行了测试，之后是没有测试的，如果强行运行，会出现波形异常的情况，虽然此前尝试利用同步时序变化滞后一个周期的特点刻意制造延迟等手段来满足异常处的波形的计算，但后了解到是没有必要的部分，最终删去相关代码。

在上板时，比较典型的问题就是按下 button 的瞬间，会导致计算器进行上千万次的计算，而这显然不是我们所期望的，所以消抖模块对于上板正确是必要的，而对于仿真过程中，由于仿真设置的 button 信号没有异常抖动，仅需要调整消抖模块的 CNT_MAX 即可正常运行仿真。

起初的消抖模块未能够处理在持续按键时不进行多次运算的问题，这一点在最终提交的代码上已经得到修改。当按下 button 时进行运算，持续按键不改变运算结果，只有释放之后再次按下才会使用先前的计算结果进行连续计算。消抖模块的实现逻辑，基本思路是限定两次 button 按下有设定的最小时间间隔并且持续按下时计数器值不发生改变或为 0，这里我设置的最小时间间隔是 1E-4s，也就是 1000 个分频时钟（10MHz）周期，经测试可以达到所期望的执行效果。

课程设计总结

总结

实验 6 十六进制计算器设计，可以视为是此前多次实验的复合产物，综合的锻炼并复习了此前实验的功能实现。

在实验 1 中，我们学习使用了 3-8 译码器的 verilog 代码实现，在那之后，实验 4 数码管控制器设计实验中同样的用到了译码器代码，并且此次实验 6，归功于此前实验 4 代码风格良好，直接复用了近 50 行数码管扫描代码，大大的提高了代码编写效率。

本次实验中要求使用时钟 IP，可以推测目的在于对使用 IP 核进行 verilog 编程的再一次熟悉。除此以外，大量使用的计数器用于时间控制，也是实验 2 熟悉的结果。

收获与待改进的方面

代码风格相比于最初几次的实验有了较大的改进。从第 2 次实验开始，因将所有的逻辑语句放在同一个 always 块中，当代码出错后难以排查，甚至逻辑无异常但是确实无法正确运行。那之后，在学长和同学以及老师的提醒下，养成了对于同一 always 块尽可能的只对一个变量进行赋值的习惯，偶尔的特例也是在严格思考过代码运行逻辑的情况下，以提高代码简洁程度来进行的少量改动（如几个 always 块有完全相同的判断条件）。

学会阅读了波形并调出自己模块的波形。因 testbench 并非自己编写，对于其代码逻辑的不熟悉往往不方便对代码进行调试，在学会将自己模块的信号波形调出进行观察以前，通常使用上板的方法进行调试，这样效率极其低下，浪费大量时间在等待比特流生成的过程中。

学会了报错信息阅读。仿真报错信息路径已经较为熟悉，elaborate 文件 xvlog 日志信息等已经深入脑海。对于日志的阅读能高效的帮助自己排除错误。

复用 xdc 文件。这一点是在实验 3 之后才意识到的，对于同样的开发板，尽管使用不同的电路元件进行编程，在信号命名习惯相同的情况下，通用的 xdc 文件是可以被总结出的，为此，经过前面几次的总结梳理得出了目前为止的通用 Exp_General.xdc 文件来提高约束文件编写效率。

使用 generate 来简化代码编写。Generate 不同于 for 语句的作用是，generate 语句相当于是将代码段展开硬嵌到最终的代码中，实际上并不会执行循环语句，所以，既起到了免于对重复部分编写代码的麻烦和单调工作，也避免了不可综合的风险。而对于 verilog 中一些小技巧的使用，也让我对向量的控制更加灵活，如 assign result[i] = cal_result[i*4 +: 4]; (出处为 generate 语句块中, genvar i;)，来控制特定连续 4 位的向量块赋值。

但是一些使用习惯有待改进。对于毛刺信号，据了解，利用任何计数器进行的时间控制都会有毛刺的产生，导致时间控制不完全准确，更建议的做法是只使用时钟 IP 而不做计数器，但由于使用计数器进行时钟控制几乎已经成为代码习惯的一部分，而当前的任务并没有如此高的精度要求，所以暂时搁置这一点不好的习惯。对于使用 rst_n 而不是 rst 是被建议的做法，理由是对生成实际电路更友好，这一点由于在写最后一次实验代码的中途被告知，因此也只在最后一次做出了相应的改动，仍然需要日后对这一意识的巩固。

本次实验收获颇多，很好的起到了最后一次大实验对于 verilog 代码能力的考查，在经过此前多次实验的打磨之后，本次实验得以顺利的完成。