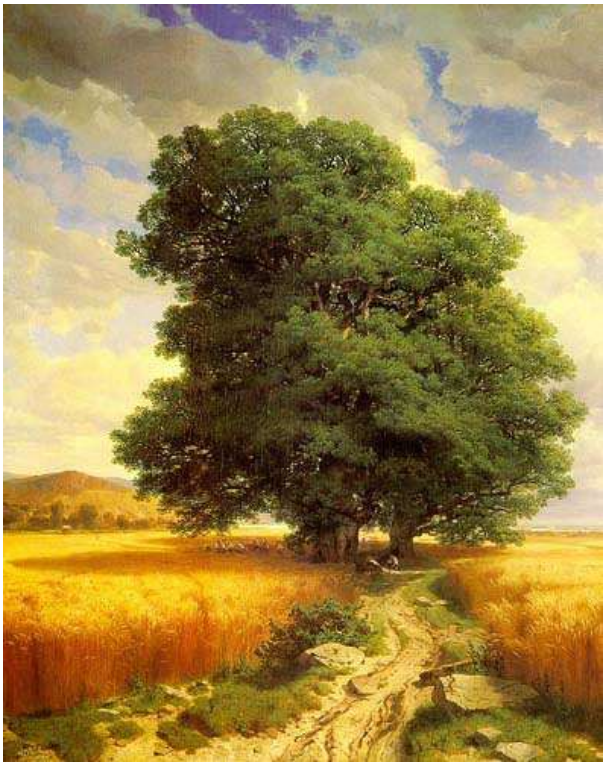
1 | 基础练习



有橡树的风景（亚历山德拉·卡拉梅）

树干总是一成不变，枝叶却纷披而伸展。

~ · ~

1.1 | 实验目的

本实验的目的是熟悉实验环境，学习如何安装 Vivado、如何使用 Vivado 2023.2 创建工程、代码编辑、RTL 分析、仿真等一系列设计流程，并在数码管上显示十进制数字。

1.2 | 实验内容

1.2.1 | Vivado 设计流程学习

学习视频和文档资料，了解 Vivado 设计流程和功能：

- Ego 五分钟快速上手-HDL 流程视频（https://www.bilibili.com/video/av24701535/）
- Ego 五分钟搭建你的数字积木视频（https://www.bilibili.com/video/av10888635）
- Vivado 安装说明视频（https://www.bilibili.com/video/av35948660）
- Vivado 设计流程中的基本概念.pdf
- 什么是约束文件.ppt
- EGO1 开发板手册，了解 FPGA 芯片管脚与外设管脚的连接关系

1.2.2 | Vivado 设计流程练习–数码管

4 个拨码开关控制 1 个数码管，选择板子上的 4 个拨码开关作为数据输入，选择板子上的 1 个数码管作为输出，将输入的 4 位二进制转换为 10 进制数输出显示到数码管。

Listing 2: 数码管显示模块（segMsg）

```
// 这是数码管显示模块
`timescale 1ns / 1ps
module segMsg(
    input key1,
    input key2,
    input key3,
    input key4,
    output reg [3:0] pos,
    output reg [7:0] seg
    );
wire [3:0] data = {key4,key3,key2,key1};

always@(data) begin
    pos=4'b0001;
    case(data)
        4'b0000:seg = 8'b0011_1111;
        4'b0001:seg = 8'b0000_0110;
        4'b0010:seg = 8'b0101_1011;
        4'b0011:seg = 8'b0100_1111;
        4'b0100:seg = 8'b0110_0110;
        4'b0101:seg = 8'b0110_1101;
        4'b0110:seg = 8'b0111_1101;
        4'b0111:seg = 8'b0000_0111;
        4'b1000:seg = 8'b0111_1111;
        4'b1001:seg = 8'b0110_1111;
        default :seg = 8'b0000_1000;
     endcase
  end
endmodule
```

1.2.3 | Vivado 设计流程练习–半加器和全加器

Listing 3: 半加器（Halfadd）

```
`timescale 1ns / 1ps

module HalfAdder(
    input wire A,
    input wire B,
    output wire Sum,
    output wire Carry
);
    assign Sum = A ^ B;
    assign Carry = A & B;
```

```
endmodule
```

Listing 4: 全加器（Fulladd）

```
`timescale 1ns / 1ps

module FullAdder(
    input wire A,
    input wire B,
    input wire Cin,
    output wire Sum,
    output wire Cout
);
    wire sum_ha1, carry_ha1, carry_ha2;

    HalfAdder HA1 (
        .A(A),
        .B(B),
        .Sum(sum_ha1),
        .Carry(carry_ha1)
    );

    HalfAdder HA2 (
        .A(sum_ha1),
        .B(Cin),
        .Sum(Sum),
        .Carry(carry_ha2)
    );

    assign Cout = carry_ha1 | carry_ha2;
endmodule
```

请大家根据上面的一位全加器和一位半加器，设计出一个两位半加器并通过仿真验证，仿真代码会在后面给出。需要注意的是，你设计的两位半加器的接口和模块名称需要和下面一致，否则无法直接使用后续的仿真代码（当然，我们鼓励自己独立完成仿真代码，这在验收的时候也会给助教留下一个好印象）。

Listing 5: 两位半加器接口示例（TwoBitAdder）

```
module TwoBitAdder(
    input wire [1:0] A,
    input wire [1:0] B,
    output wire [1:0] Sum,
    output wire Cout
);
```

1.3 | 仿真补充说明

本书已在 lab0 中详细介绍了 vivado 的使用方法和设计流程，因此不在本章节赘述仿真的步骤，仅介绍编写 testbench（测试用例）的教程。

1.3.1 | testbench 介绍

Verilog 代码设计完成后，还需要进行重要的步骤，即逻辑功能仿真。仿真激励文件称之为 testbench，放在各设计模块的顶层，以便对模块进行系统性的例化调用进行仿真。

初学者可能对仿真的重要性没有概念，但要知道，数字电路行业会具体划分设计工程师和验证工程师。其重要性可见一般。

1.3.2 | testbench 实例（仅供参考）

编写 testbench 包括但不限于以下步骤：

1. 设置时间单位和时间精度
2. 定义 testbench 模块
3. 定义输入输出信号
4. 依据被测模块的内容来设计测试模块
5. 实例化待测试模块
6. 结束仿真

示例代码：

Listing 6: 数码管仿真模块（segMsg_tb）

```
`timescale 1ns / 1ps//时间单位和时间精度
module segMsg_tb;    //模块
reg key1;//输入输出信号
reg key2;
reg key3;
reg key4;
wire [3:0] pos;
wire [7:0] seg;

initial begin //初始化
    #100
    key1 = 1'b0; key2 = 1'b0;key3 = 1'b0;key4 = 1'b0;
    #100;
    key1 = 1'b1; key2 = 1'b1;key3 = 1'b1;key4 = 1'b1;
    #100;
    forever #20 {key4, key3, key2, key1} = {$random} % 4'b1111;// 生成16位的随机数
end

initial begin
    #10000;
    $finish; // 或使用 $stop;
end

segMsg WSW_seg(key1,key2,key3,key4,pos,seg);//实例化

endmodule
```

Listing 7: 两位半加器仿真模块（TwoBitAdder_tb）

```
`timescale 1ns / 1ps
```

```
2
3  module tb_adder;
4      reg [1:0] A;
5      reg [1:0] B;
6      wire [1:0] Sum;
7      wire Cout;
8
9      // 实例化两位加法器
10     TwoBitAdder add (
11         .A(A),
12         .B(B),
13         .Sum(Sum),
14         .Cout(Cout)
15     );
16
17     initial begin
18         // 测试用例1: 01 + 01 = 10
19         A = 2'b01;  // 01
20         B = 2'b01;  // 01
21         #10;
22         $display("Test 1: %d + %d = %d, Sum = %b, Cout = %b", A, B, A+B, Sum, Cout)
               ;
23
24         // 测试用例2: 11 + 11 = [1]10
25         A = 2'b11;  // 11
26         B = 2'b11;  // 11
27         #10;
28         $display("Test 2: %d + %d = %d, Sum = %b, Cout = %b", A, B, A+B, Sum, Cout)
               ;
29
30         // 测试用例3: 11 + 01 = [1]00
31         A = 2'b11;  // 11
32         B = 2'b01;  // 01
33         #10;
34         $display("Test 3: %d + %d = %d, Sum = %b, Cout = %b", A, B, A+B, Sum, Cout)
               ;
35
36         // 测试用例4: 00 + 00 = 00
37         A = 2'b00;  // 00
38         B = 2'b00;  // 00
39         #10;
40         $display("Test 4: %d + %d = %d, Sum = %b, Cout = %b", A, B, A+B, Sum, Cout)
               ;
41
42         $stop;
43     end
44 endmodule
```

1.4 | Vivado 代码编辑和 RTL 分析

创建一个工程，自己指定工程位置和工程名称，新建空白源程序文件，依次完成下面代码编辑和 RTL 分析：

分别编写图1.1中的两个例子，观察 vivado 工具的 RTL 分析结果，截图放入实验报告；

```
1 module example1 (xl, x2, s, f);
2     input xl, x2, s;
3     output f;
4
5     not (k, s);
6     and (g, k, x1);
7     and (h, s, x2);
8     or  (f, g,  h);
9 endmodule
```

```
1  module example2 (x1, x2, x3, x4, f, g
       , h);
2      input x1, x2, x3, x4;
3      output f, g, h;
4
5      and (z1, x1, x3);
6      and (z2, x2, x4);
7      or  ( g, z1, z2);
8      or  (z3, x1,~x3);
9      or  (z4,~x2, x4);
10     and ( h, z3, z4);
11     or  ( f,  g,  h);
12 endmodule
```

图 1.1: 代码 1

分别编写图1.2例子，观察 vivado 工具的 RTL 分析结果，截图放入实验报告；

```
1 module example3(x1, x2, s, f);
2     input x1, x2, s;
3     output f;
4
5     assign f=(~s& x1)|(s & x2);
6 endmodule
```

```
1  module example4 (x1, x2, x3, x4, f, g
       , h);
2      input x1,x2, x3, x4;
3      output f, g, h;
4
5      assign g=(x1 & x3)|(x2 & x4);
6      assign h=(x1|~x3)&(~x2|x4);
7      assign f= g|h;
8  endmodule
```

图 1.2: 代码 2

分别编写图1.3例子，对比 vivado 工具的 RTL 分析结果，截图放入实验报告，给出你对结果的理解；

```
module fulladd1 (Cin, x, y, s, Cout);
    input Cin, x, y;
    output s, Cout;

    xor (s, x, y, Cin);
    and (z1, x, y);
    and (z2, x, Cin);
    and (z3, y, Cin);
    or (Cout, z1, z2, z3);
endmodule
```

```
module fulladd2 (Cin, x, y, s, Cout);
    input Cin, x, y;
    output s, Cout;

    assign s= x ^ y ^ Cin;
    assign Cout=(x & y)|(x & Cin)|(y
        & Cin);
endmodule
```

图 1.3: 代码 3

编写图1.4例子，观察 vivado 工具的 RTL 分析结果，截图放入实验报告，给出你对结果的理解；

```
module adder4 (carryin, x3, x2, x1, x0, y3, y2, y1, y0, s3, s2, s1, s0, carryout);
    input carryin, x3,x2,x1,x0, y3, y2,y1, y0;
    output s3,s2,s1, s0, carryout;

    fulladd stage0 (carryin, x0, y0, s0, c1);
    fulladd stagel (c1, x1, y1, s1, c2);
    fulladd stage2 (c2, x2, y2, s2, c3);
    fulladd stage3 (c3, x3, y3, s3, carryout);
endmodule

module fulladd (Cin, x, y, s, Cout);
    input Cin, x, y;
    output s, Cout;

    assign s=x ^ y ^ Cin;
    assign Cout=(x & y)|(x & Cin)|(y & Cin);
endmodule
```

图 1.4: 代码 4

1.5 | 实验要求

1. 观看提供的所有视频资料；学习 Vivado 设计流程中的基本概念.pdf、约束文件.ppt。在实验报告中回答以下问题：
   - [a] 描述 Vivado 的设计流程
   - [b] 什么是网表
   - [c] Vivado 设计流程中，Synthesis 的作用是什么？
   - [d] Vivado 设计流程中，Implementation 的作用是什么？
2. 在实验报告中提交1.2.2、1.2.3的仿真结果截图及板上运行结果照片；
3. 在实验报告中提交1.4的 RTL 详细设计图及相关文字说明；

4. 实验报告提交到 CG（202.204.62.165）平台；

5. 数码管板上运行结果和加法器仿真结果需要找助教或老师演示验收。两位半加器实验中，有余力者可以选择通过拨码开关输入数据，数码管显示结果来进行板上验证，验收成功后可以酌情加分；（未验收者实验 1 无成绩）