中山大学数据科学与计算机学院本科生实验报告

（2019 学年秋季学期）

课程名称：**计算机组成原理实验**　　任课教师：　　助教：

年级&班级	19 计科超算	专业(方向)	
学号	18324034	姓名	林天皓
电话	17820688242	Email	Linth5@mail2.sysu.edu.cn
开始日期	2020.9.11	完成日期	2020.9.18

一、实验题目

VIVADO 模块化设计

二、实验目的

1.对 Verilog HDL 的模块化设计做初步了解。

2.体会直流设计“自顶向下”设计思想。

三、实验内容

1 实验原理

先设计系统的顶成模块，包括行为级描述，功能模拟和性能评估，在本次实验中即为受到开关控制的 LED 循环点亮的跑马灯模块设计，所需资源少，性能足够，在实现过程中需要用到输入时钟和开关状态并且根据顺序输出信号。接下来对各个功能模块进行划分。在本次实验中,通过上学期数字电路实验的学习,可以将划分为时钟分频模块,计数器模块,3-8 译码器模块，并通过时钟信号的不断改变使得 3-8 译码器的输出按照顺序循环往复。设计后，逐步检验与模拟小模块的实现效果是否达到要求，最后进行综合，将输出与输入变量绑定到实验版的引脚上，下载程序观察实验效果并验证是否完成实验要求。

2. 实验步骤

首先实现时钟分频模块。由于开发板的默认主频为 100Mhz。要使得肉眼容易观察到流水灯的变化，我们在这里将高频率的时钟信号，通过软件分频为 1hz，即为 0.5s 时钟信号翻转一次。

首先计算原始主频的周期

$$T = \frac{1}{100\times10^6} = 10^{-8}\,s$$

然后计算需要多少次时钟周期使得分频后的时钟信号翻转一次

$$n = \frac{0.5s}{10^{-8}\,s} = 5\times10^7$$

通过计算，将我们的结果写为 verilog 函数。为了保证同步性， 在时钟的翻转过程中采用非阻塞赋值的方式。

实现如下:

```
module clock_div(
    input clk,
    output reg clk_sys =0
    );
    reg [25:0] div_counter =0;
    always @(posedge clk) begin
        if(div_counter >= 50000000) begin
            clk_sys<=~clk_sys;
            div_counter <=0;
        end
        else begin
            div_counter <= div_counter +1;
        end
        $display("div_counter:",div_counter," clk_sys",clk_sys);
    end
endmodule
```

时钟分频模块完成。

接下来，实现 8 进制计数器模块。该模块输入分频后的时钟信号，输出三位二进制数字表示当前计数器的状态，同时还需要有一个清零的过程来控制整个模块的开关。由于该模块也是一个时序逻辑电路，所以与分频模块相同，使用非阻塞赋值。

实现如下：

```
module counter8(
    input clk,
    input reset,
    output reg[2:0] count
    );
    always @(posedge clk or negedge reset) begin
        if(reset == 0)begin
            count <= 0;
        end
        else begin
            if(count==0)
                count<=7;
            else begin
                count <= count-1;
            end
        end
    end
endmodule
```

8 进制计数器模块设计完成。

然后实现 3-8 译码器模块。该模块接收一个三位二进制数字，输出 8 位二进制数。由于该模块是一个组合逻辑电路，采用阻塞赋值的方式。

实现如下：

```
module decoder38(
    input A,
    input B,
    input C,
    input Enable,
    output reg Y0,
    output reg Y1,
    output reg Y2,
    output reg Y3,
```

```
    output reg Y4,
    output reg Y5,
    output reg Y6,
    output reg Y7
    );
    always @(A or B or C or Enable) begin
        if(!Enable) begin
            {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0}=8'b0000_0000;
        end
        else begin
            case({C,B,A})
            3'b000: {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0}=8'b0000_0001;
            3'b001: {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0}=8'b0000_0010;
            3'b010: {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0}=8'b0000_0100;
            3'b011: {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0}=8'b0000_1000;
            3'b100: {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0}=8'b0001_0000;
            3'b101: {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0}=8'b0010_0000;
            3'b110: {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0}=8'b0100_0000;
            3'b111: {Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0}=8'b1000_0000;
            endcase
        end
    end
endmodule
```

3-8 译码器模块设计完成。

最后实现顶层模块设计，将需要用到的所有输入输出做为参数，这部分主要把之前设计的小模块实例化，对于时钟信号和分频时钟信号，使用 wire 声明为无逻辑连线。

实现如下：

```
module led_8lights(
    input clock, input reset,
    output Y0, output Y1, output Y2, output Y3,
    output Y4, output Y5,output Y6, output Y7
    );
    wire clk_sys;
    wire [2:0] count; // counter
    counter8 U0( .clk(clk_sys), .reset(reset), .count(count) );
    // clock
    clock_div U1( .clk(clock), .clk_sys(clk_sys) );
    // 38decoder
    decoder38 U2(
        .A(count[0]), .B(count[1]), .C(count[2]),
```

```
        .Enable(reset),
        .Y0(Y0), .Y1(Y1), .Y2(Y2), .Y3(Y3),
        .Y4(Y4), .Y5(Y5), .Y6(Y6), .Y7(Y7)
); endmodule
```

至此，完成了所有模块设计工作。

完整模块的 RTL 图如下

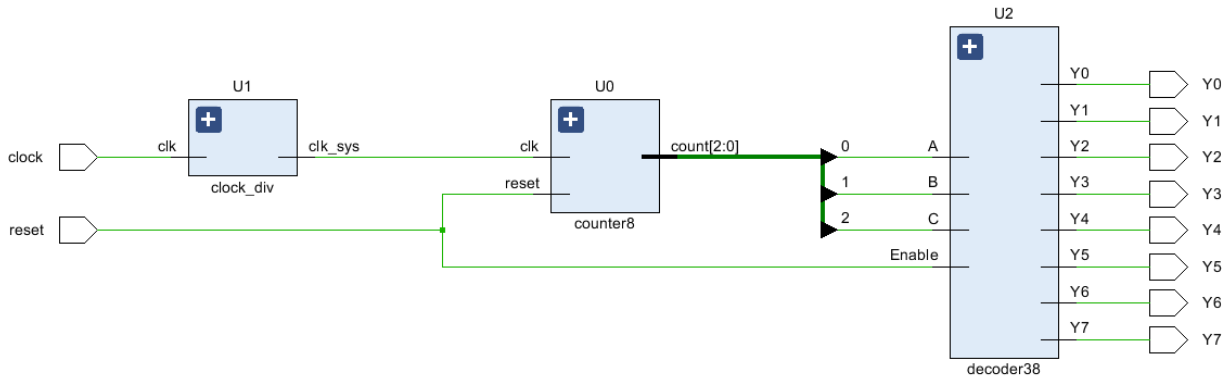

图-1：综合所生成的RTL图

下面进行仿真测试。在仿真模块中，通过实例化顶层模块，对 clock 信号按照周期进行翻转来测试电路的功能。

模拟仿真代码如下:

```
module led_sim();
    //input
    reg clock;
    reg reset;
    //output
    wire Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7;
    led_8lights uut(.clock(clock),.reset(reset),
    .Y0(Y0),.Y1(Y1),.Y2(Y2),.Y3(Y3),
    .Y4(Y4),.Y5(Y5),.Y6(Y6),.Y7(Y7)
    );
    always #5 clock = ~clock;
    initial begin
        clock      = 0;
        reset      = 0;
        #100 reset = 1;
```

```
    end
endmodule
```

仿真验证通过后，对照 bayas3 实验板的原理图，将引脚绑定在芯片上，同时设定引脚电平为 3.3V 以驱动 LED。最后生成比特流下载到 FPGA 上

四、实验结果

仿真测试结果如图（注:由于仿真速度的原因，把分频后频率大幅度提高更快看到差异，验证正确性）。

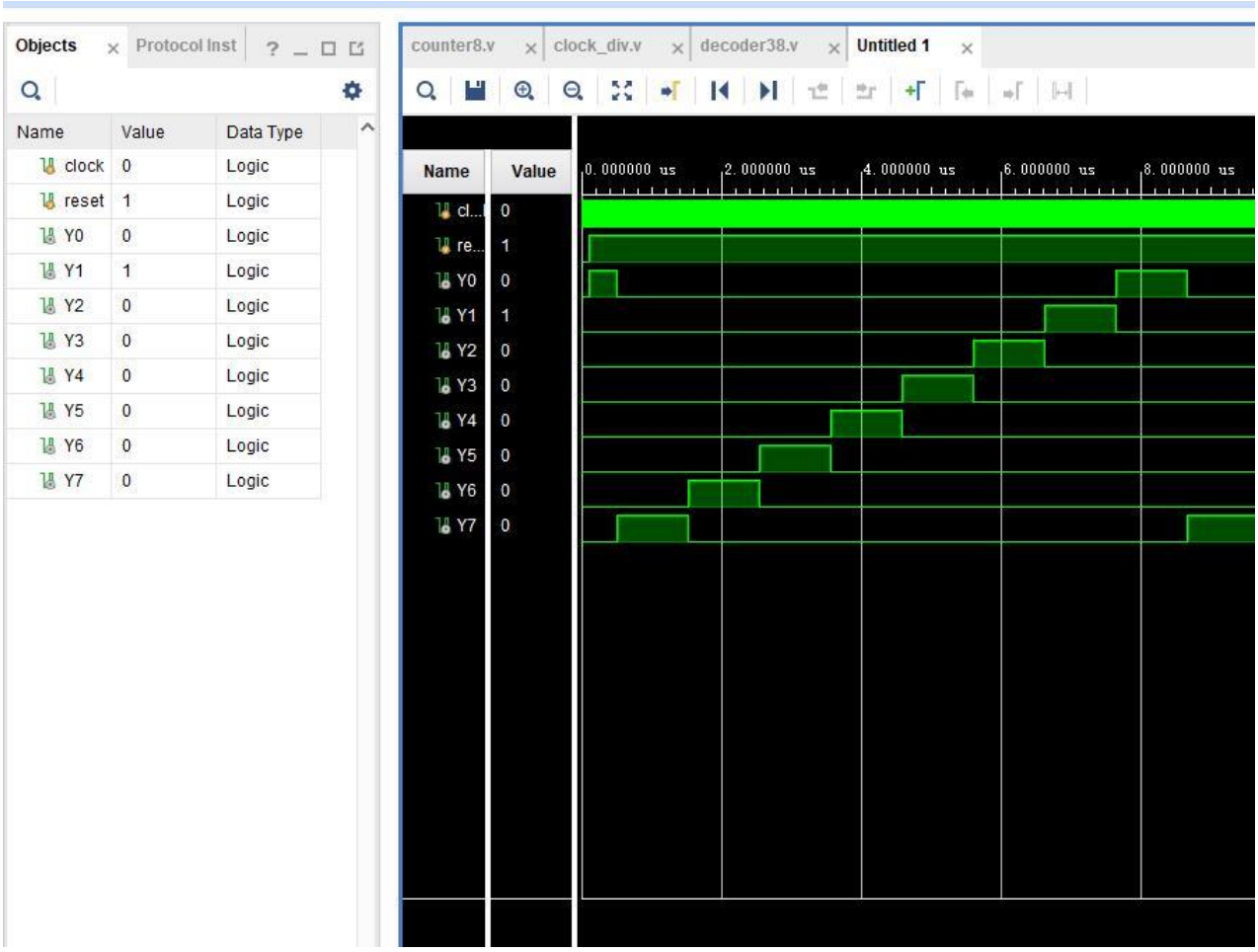

图-2：模拟分析状态波形图

分析：一开始的100ns中，reset为0，3-8译码器未使能所有LED关闭，100ns后，reset为1，计数器和3-8译码器正常工作，每间隔1us计数器的状态发生一次改变，8个LED轮流被点亮，计数

器能够循环计数。

总结:根据模拟结果，实现满足了设计所需要的功能。

实际成品效果展示。

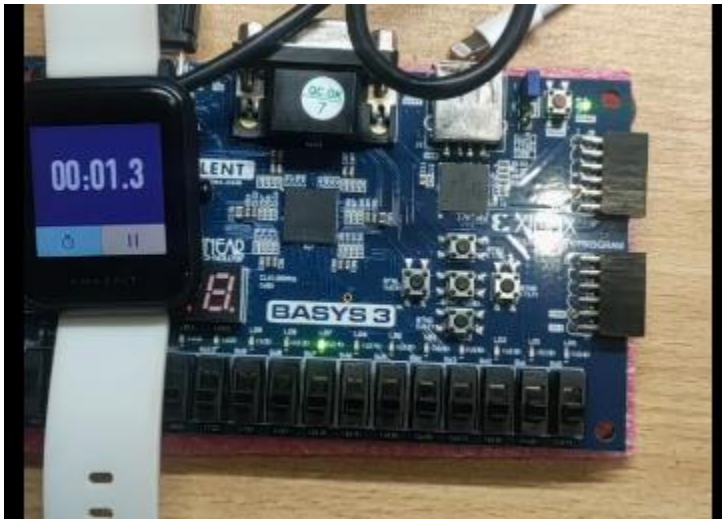

图-3：实物运行状态(左侧秒表时间为1.3s)

分析:t=1s,最左边的led点亮。

图-4：实物运行状态(左侧秒表时间为2.1s)

分析:t=2s,最左边第二个的led点亮。

图-5：实物运行状态(左侧秒表时间为3.2s)

分析:t=2s,最左边第三个的led点亮。

更长时间的规律同样符合预期，此处不再赘述。

综上所述，该设计过程使用模块化设计自顶向下完成了实验。

五、实验感想

在这次实验流程中，初步尝试了模块化设计的思想，了解了 verilog 的基本语法如何使用，了解一个硬件描述电路通过模块化，生成实例，综合实现，最后连接上对应的引脚并生成写入文件，来完成硬件实现电路的效果。只有把层次分划分清楚了，整个实现的思路才能够有逻辑顺序。在了解 verilog 语法的过程中，了解了 verilog 中 reg 和 wire 的基本格式，assign 的赋值给 wire 变量，reg 格式通过<= 和= 的区别来完成阻塞的赋值和同步的赋值，正好对应了数字电路中组合逻辑电路与时间无关和特点与时序逻辑电路中引脚状态的变化和上一个状态有关的两种不同的电路方式。除此之外，还尝试了在 vscode 中编写 verilog 代码，可以使用插件来自动补全 begin 后面的 end，但是由于暂时还不会配置 verilog 的 language server 未能完成自动检查错误的功能，希望以后能够找到方法。再吐槽一下 vivado 的 ide 甚至不能在文件发生变化之后自动地重新载入代码文件，被微软家的 ide 惯坏了实在不太习惯。最终

在整个实验中逐步解决问题，最终完成了实验。

附录（流程图，注释过的代码）：

设计流程图如下

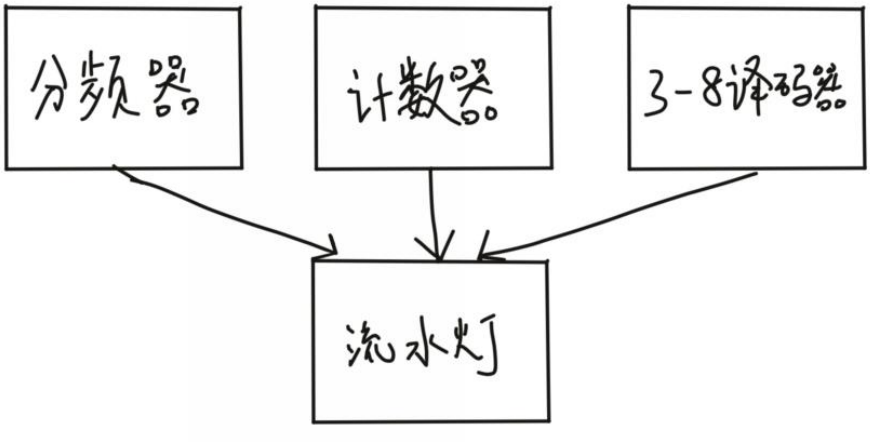

图-6：本次实验总设计流程图

代码大多数在上文中已经放出，这里仅给出上文中未出现的约束文件。

led_8lights.xdc 如下

```
set_property IOSTANDARD LVCMOS33 [get_ports clock]
set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property PACKAGE_PIN W5 [get_ports clock]
set_property PACKAGE_PIN R2 [get_ports reset]
set_property PACKAGE_PIN U16 [get_ports Y0]
set_property PACKAGE_PIN E19 [get_ports Y1]
set_property PACKAGE_PIN U19 [get_ports Y2]
set_property PACKAGE_PIN V19 [get_ports Y3]
set_property PACKAGE_PIN W18 [get_ports Y4]
set_property PACKAGE_PIN U15 [get_ports Y5]
set_property PACKAGE_PIN U14 [get_ports Y6]
set_property PACKAGE_PIN V14 [get_ports Y7]
set_property IOSTANDARD LVCMOS33 [get_ports Y0]
set_property IOSTANDARD LVCMOS33 [get_ports Y1]
set_property IOSTANDARD LVCMOS33 [get_ports Y2]
set_property IOSTANDARD LVCMOS33 [get_ports Y3]
set_property IOSTANDARD LVCMOS33 [get_ports Y4]
set_property IOSTANDARD LVCMOS33 [get_ports Y5]
set_property IOSTANDARD LVCMOS33 [get_ports Y6]
set_property IOSTANDARD LVCMOS33 [get_ports Y7]
```