中国科学技术大学计算机学院

《数字电路实验》报告

实验题目：FPGA 实验平台及 IP 核使用

学生姓名：_____徐奥_____

学生学号：_____PB20061343_____

完成日期：____2021 年 11 月 25 日____

计算机实验教学中心制

2020 年 09 月

【实验题目】

学习 FPGA 实验平台，学习如何使用 IP 核进行电路设计

【实验练习】

题目 1. 例化一个 16*8bit 的 ROM，并对其进行初始化，输入端口由 4 个开关控制，输出端口连接到七段数码管上。

如图 1，设置 ROM 的初始值，与图 2 所示的值对应，实现将由开关输入的 4 位二进制数，转化为七段数码管的输出

```
memory_initialization_radix=2;
memory_initialization_vector=0111111 0000110 1011011 1001111 1100110 1101101
1111101 0000111 1111111 1101111 1110111 1111100 0111001 1011110 1111001 1110001
0111111;
```

图 1

```
case(sw)
    4'b0000:led<=7'b0111111;
    4'b0001:led<=7'b0000110;
    4'b0010:led<=7'b1011011;
    4'b0011:led<=7'b1001111;
    4'b0100:led<=7'b1100110;
    4'b0101:led<=7'b1101101;
    4'b0110:led<=7'b1111101;
    4'b0111:led<=7'b0000111;
    4'b1000:led<=7'b1111111;
    4'b1001:led<=7'b1101111;
    4'b1010:led<=7'b1110111;
    4'b1011:led<=7'b1111100;
    4'b1100:led<=7'b0111001;
    4'b1101:led<=7'b1011110;
    4'b1110:led<=7'b1111001;
    4'b1111:led<=7'b1110001;
    default:led<=7'b0111111;
endcase
```

图 2

引脚约束文件中只需要 4 个开关和 led 对应的引脚。

生成 Bitstream 文件烧写到 FPGA 平台上后，效果如下图

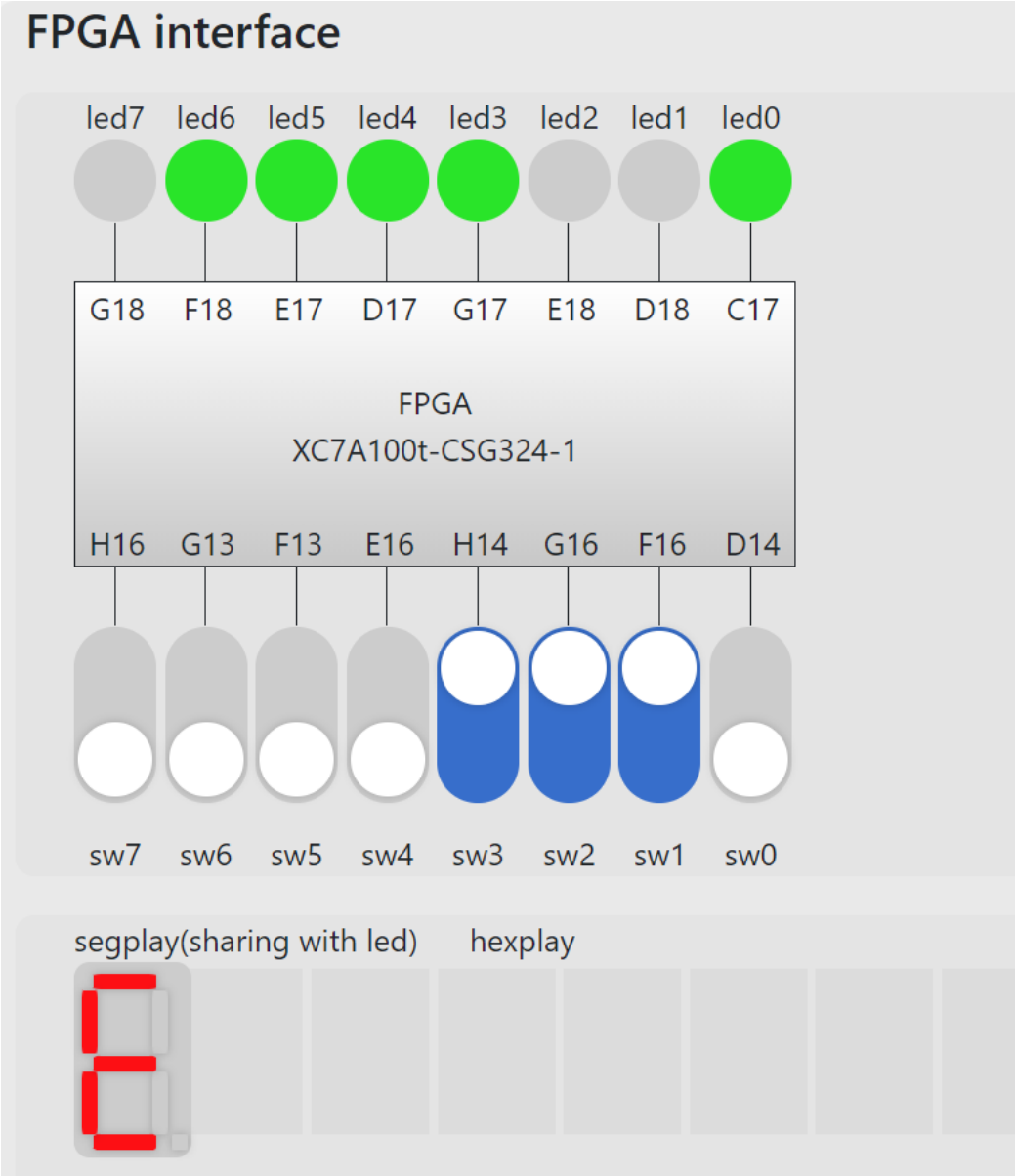

图 3

题目 2. 采用 8 个开关作为输入，两个十六进制数码管作为输出，采用时分复用的方式将开关的十六进制数值在两个数码管上显示出来，例如高四位全为 1，低四位全为 0 时，数码管显示“F0”。

设计思路：

每四个开关对应一个 4 位二进制数，将其与数码管的引脚相连。利用人的视觉暂留效应，不断刷新显示两个数码管的值，使得在看上去为

两个数字同时显示。在代码中设置一个标志 a，当 a=0 时，显示低位的数码管，当 a=1 时，显示高位的数码管

代码如下图

```
module q2_v(
    input [7:0] sw,
    input clk,
    output reg [3:0] hexplay_data,
    output reg [2:0] hexplay_an
    );

    reg a;
    reg [19:0] cnt;
    wire pulse;

    always@(posedge clk)
        begin
            if(cnt>= 1000000)
                cnt <= 0;
            else
                cnt <= cnt + 1;
        end
```

```
    assign pulse = (cnt == 1);
    always @(posedge clk)
    begin
        if(pulse)
            a <= ~a;
    end

    always@(*)
    begin
        if(a==0)
            hexplay_an<=3'b000;
        else
            hexplay_an<=3'b001;
    end

    always@(*)
    begin
        if(a==0)
            hexplay_data <= {sw[3],sw[2],sw[1],sw[0]};
        else
            hexplay_data <= {sw[7],sw[6],sw[5],sw[4]};
    end

endmodule
```

图 4

引脚约束文件为

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];

## FPGAOL SWITCH

set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }];
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];
set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }];
set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }];
set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 } [get_ports { sw[4] }];
set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 } [get_ports { sw[5] }];
set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 } [get_ports { sw[6] }];
set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { sw[7] }];


## FPGAOL HEXPLAY

set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[0] }];
set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[1] }];
set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[2] }];
set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[3] }];
set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33 } [get_ports { hexplay_an[0] }];
set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports { hexplay_an[1] }];
set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports { hexplay_an[2] }];
```

题目 3. 利用本实验中的时钟管理单元或周期脉冲技术，设计一个精度为 0.1 秒的计时器，用 4 位数码管显示出来，数码管从高到低，分别表示分钟、秒钟十位、秒钟个位、十分之一秒，该计时器具有复位功能（可采用按键或开关作为复位信号），复位时计数值为 1234

设计思路：

1. 在四个数码管上显示四位数字，与第二题思路类似，不断刷新显示。
2. 因为时钟频率为 100Mhz，所以一个时钟周期为 10ns，所以 0.1s=10000000 个时钟周期，设置一个计数标志，当它达到 10000000 时，数码管的最低为加一，并根据时间的进制规则设置进位，本次实验在进位中我采用了阻塞式赋值。

代码为：

```
module q3_v(
    input clk,
    input rst,
    output reg [3:0] hexplay_data,
    output reg [2:0] hexplay_an
    );

    reg [3:0] minute;
    reg [3:0] second_10;
    reg [3:0] second_1;
    reg [3:0] second_0_1;

    reg [26:0] timee;
    always@(posedge clk)
    begin
        if (timee >= 10000000)
            timee <= 0;
        else
            timee <= timee + 1;
    end
```

```
reg [19:0] cnt;
always@(posedge clk)
    begin
        if(cnt>= 20'd1000000)
            cnt <= 20'd0;
        else
            cnt <= cnt + 20'd1;
    end

assign pulse = (cnt == 20'd1);
reg [1:0] a;

always@(posedge clk)
begin
    if (pulse)
    begin
        if (a == 2'b11)
            a<=2'b00;
        else
            a<=a+2'b01;
    end
end
```

```
always @(posedge clk)
begin
    if(rst)
    begin
        minute <= 4'b0001;
        second_10 <= 4'b0010;
        second_1 <= 4'b0011;
        second_0_1 <= 4'b0100;
    end
```

```
        else
        begin
            if(timee == 1)
            begin
                second_0_1 = second_0_1 + 1;
                if(second_0_1 == 4'b1010)
                begin
                    second_0_1 = 4'b0000;
                    second_1 = second_1 + 1;
                    if(second_1 == 4'b1010)
                    begin
                        second_1 = 4'b0000;
                        second_10 = second_10 + 1;
                        if(second_10 == 4'b0110)
                        begin
                            second_10 = 4'b0000;
                            minute = minute + 1;
                        end
                    end
                end
            end
        end
    end
```

```
    always@(*)
    begin
    case(a)
        2'b00:begin
            hexplay_an=3'b000;
        end
        2'b01:begin
            hexplay_an=3'b001;
        end
        2'b10:begin
            hexplay_an=3'b010;
        end
        2'b11:begin
            hexplay_an=3'b011;
        end
    endcase
    end

    always @(posedge clk)
    begin
    case(a)
        2'b00:  hexplay_data <= {second_0_1[3],second_0_1[2],second_0_1[1],second_0_1[0]};
        2'b01:  hexplay_data <= {second_1[3],second_1[2],second_1[1],second_1[0]};
        2'b10:  hexplay_data <= {second_10[3],second_10[2],second_10[1],second_10[0]};
        2'b11:  hexplay_data <= {minute[3],minute[2],minute[1],minute[0]};
    endcase
    end
endmodule
```

【总结与思考】

1. 本次实验中，我将很大一部分时间耗费在第一问，原因是第一遍没有读懂题意，以为是需要在执行过程中给 ROM 赋值。但是题目的本意是 ROM 存储的值是在初始化时就设置好了的。
2. 本次实验的一大挑战就是变量与约束引脚的对应，不过在将它们对应的过程中，自己对于 FPGA 的原理也有了更深入的认识。比如 hexplay pin 中的 an0,an1,an2,我最初认为它们是独热码，但随着对引脚的学习，我了解到他们的是 3-8 译码器的输入。
3. 本次实验内容适中，难度中等，Verilog 代码量较前几次有了提升。
4. 建议增加第一题的题目描述，比如“你需要在 ROM 初始化时将它存储的值设置好，对于给定的四个开关的状态，七段数码管可以输出对应的十六进制数”。
5. 前面的实验介绍文档建议增加 Vivado 2019 版的部分截图