LED 流水灯与按键控制实验报告

石禹

2023K8009926030

中国科学院大学

§ 1 小组成员与分工

- 成员: 石禹、许书闻、潘瀚霖
- 分工: 为保证每个小组成员均能够完整地对整个实验内容进行学习和理解，本小组的所有成员均分别独立地完成了所有的实验内容，包括了代码编写、调试和实验报告的撰写。

§ 2 实验目的

使用新起点开发板上的四个按键控制四个 LED 灯。不同按键按下时，四个 LED 灯显示不同效果。具体效果为: 无按键按下时，LED 灯全灭；按键 1 按下时，LED 灯显示自右向左的流水效果；按键 2 按下时，LED 灯显示自左向右的流水效果；按键 3 按下时，四个 LED 灯同时闪烁；按键 4 按下时，LED 灯全亮。

§ 3 LED 流水灯实验

在设计 LED 流水灯的过程中，需要设计 0.2 秒计时器，同时利用位移寄存器实现流水灯的效果。其 Verilog 代码如下:

```
module ProjectOne(
        input sys_clk,//系统时钟
        input sys_rst_n,//系统复位低电平有效
        output reg [3:0] led //四个led灯
);

reg[23:0] counter;

always @(posedge sys_clk or negedge sys_rst_n) begin
        if (!sys_rst_n)
                counter <=24'd0;
        else if (counter < 24'd1000_0000)
                counter <= counter+1'd1;
        else
```

```
                counter <= 24'd0;
end

always @(posedge sys_clk or negedge sys_rst_n) begin
        if (!sys_rst_n)
                led <=4'b0001;
        else if(counter ==24'd1000_0000)
                led[3:0]<= {led[2:0],led[3]};
        else
                led <= led;
end

endmodule
```

编译完成后，将生成的 SOF 文件下载到开发板上进行测试，测试结果以视频方式呈现，链接如下: `https://www.bilibili.com/video/BV15p7oz1Etn`

§ 4 按键控制 LED 灯实验

1. 分配引脚:

   实验中需要用到的变量为:sys clk, sys rst n, key[0], key[1], key[2], key[3], led[0], led[1], led[2], led[3]。各变量所分配的引脚如下表所示:

信号名	方向	管脚	端口说明
sys clk	input	M2	系统时钟、50M
sys rst n	input	M1	系统复位、低有效
key[0]	input	E16	按键
key[1]	input	E15	按键
key[2]	input	M15	按键
key[3]	input	M16	按键
led[0]	output	D11	LED
led[1]	output	C11	LED
led[2]	output	E10	LED
led[3]	output	F9	LED

由于手动分配引脚较为繁琐，且容易由于操作失误导致错误。于是我编写了 TCL 脚本进行引脚的自动分配操作。上表对应的 TCL 脚本代码如下:

```
set_location_assignment PIN_M2 -to sys_clk
set_location_assignment PIN_M1 -to sys_rst_n
set_location_assignment PIN_E16 -to key[0]
set_location_assignment PIN_E15 -to key[1]
set_location_assignment PIN_M15 -to key[2]
set_location_assignment PIN_M16 -to key[3]
set_location_assignment PIN_D11 -to led[0]
set_location_assignment PIN_C11 -to led[1]
set_location_assignment PIN_E10 -to led[2]
set_location_assignment PIN_F9 -to led[3]
```

通过运行上述脚本，Quartus 会自动将引脚分配到对应的管脚上，此时打开引脚分配器可以看到所有引脚均已被正确分配。

2. 编写基础程序:

定义模块与端口、变量:

```
module key_led(
input                sys_clk   ,
input                sys_rst_n ,
input        [3:0]   key ,
output reg   [3:0]   led
);
reg   [23:0] cnt;
reg   [1:0]  led_control;
```

在基础实验代码中，LED 在流水效果和闪烁效果在时间间隔均为 0.2 秒，因此需要在程序中定义一个 0.2s 的计数器，即每隔 0.2s，状态计数器加一。计数器的时钟频率为 50MHz，因此计数器的计数值为 0.2s*50MHz=10000000。计数器的 Verilog 代码如下:

```
always @(posedge sys_clk or negedge sys_rst_n)
begin
```

```
    if(!sys_rst_n)
        cnt<=24'd9_999_999;
    else if(cnt<24'd9_999_999)
        cnt<=cnt+1;
    else
        cnt<=0;
end
```

由于实验需要对四个 LED 灯进行有规律的控制，因此需要进行 LED 灯的状态选择。其 Verilog 代码如下:

```
always @(posedge sys_clk or negedge sys_rst_n)
begin
    if(!sys_rst_n)
        led_control<=2'b00;
    else if(cnt==24'd9_999_999)
        led_control<=led_control+1;
    else
        led_control<=led_control;
end
```

由此，LED 灯的状态就可以通过 led control 的值来进行周期变化了。随后需要引入按键的状态控制 LED 灯的状态。其 Verilog 代码如下:

```
always @(posedge sys_clk or negedge sys_rst_n)
begin
    if(!sys_rst_n) begin
        led<=4'b 0000;
    end
    else if(key[0]== 0)
        case (led_control)
            2'b00    : led<=4'b1000;
            2'b01    : led<=4'b0100;
            2'b10    : led<=4'b0010;
            2'b11    : led<=4'b0001;
            default  : led<=4'b0000;
```

```
        endcase
    else if (key[1]==0)
        case (led_control)
            2'b00    : led<=4'b0001;
            2'b01    : led<=4'b0010;
            2'b10    : led<=4'b0100;
            2'b11    : led<=4'b1000;
            default  : led<=4'b0000;
        endcase
    else if (key[2]==0)
        case (led_control)
            2'b00    : led<=4'b1111;
            2'b01    : led<=4'b0000;
            2'b10    : led<=4'b1111;
            2'b11    : led<=4'b0000;
            default  : led<=4'b0000;
        endcase
    else if (key[3]==0)
          led=4'b1111;
    else
        led<=4'b0000;
end
```

综上所述，完整的 Verilog 代码如下:

```
module key_led(
    input               sys_clk   ,
    input               sys_rst_n ,
    input       [3:0]   key ,
    output  reg [3:0]   led
);
reg  [23:0] cnt;
reg  [1:0]  led_control;
always @(posedge sys_clk or negedge sys_rst_n)
begin
```

```
    if(!sys_rst_n)
        cnt<=24'd9_999_999;
    else if(cnt<24'd9_999_999)
        cnt<=cnt+1;
    else
        cnt<=0;
end
always @(posedge sys_clk or negedge sys_rst_n)
begin
    if(!sys_rst_n)
        led_control<=2'b00;
    else if(cnt==24'd9_999_999)
        led_control<=led_control+1;
    else
        led_control<=led_control;
end
always @(posedge sys_clk or negedge sys_rst_n)
begin
    if(!sys_rst_n) begin
        led<=4'b 0000;
    end
    else if(key[0]== 0)
        case (led_control)
            2'b00    : led<=4'b1000;
            2'b01    : led<=4'b0100;
            2'b10    : led<=4'b0010;
            2'b11    : led<=4'b0001;
            default  : led<=4'b0000;
        endcase
    else if (key[1]==0)
        case (led_control)
            2'b00    : led<=4'b0001;
            2'b01    : led<=4'b0010;
```

```
                2'b10    : led<=4'b0100;
                2'b11    : led<=4'b1000;
                default  : led<=4'b0000;
            endcase
        else if (key[2]==0)
            case (led_control)
                2'b00    : led<=4'b1111;
                2'b01    : led<=4'b0000;
                2'b10    : led<=4'b1111;
                2'b11    : led<=4'b0000;
                default  : led<=4'b0000;
            endcase
        else if (key[3]==0)
              led=4'b1111;
        else
            led<=4'b0000;
    end

    endmodule
```

至此，实验的代码编写完成，进行编译后，将生成的 SOF 文件下载到开发板上进行测试。

3. 固化程序:

   在实验完成后，我将实验代码进行固化，固化的操作步骤如下:

   (a) 选择“File”菜单下的“Convert Programming Files”选项。

   (b) 在弹出的对话框中，选择“SOF”文件作为输入文件。

   (c) 选择生成 JIC 文件的输出路径。

   (d) 在“Programming File Type”中选择“JIC”。

   (e) 点击“Start”按钮开始转换。

   (f) 等待转换完成后，点击“OK”按钮。

固化完成后，我将固化后的 JIC 文件下载到开发板上进行测试，测试结果与之前的实验结果一致。且在断开 USB 线后，LED 灯的状态仍然保持不变，说明固化成功。

4. 实验结果展示: 得到的实验结果由于以动态效果呈现，我将其拍摄成视频进行展示，视频链接如下: `https://www.bilibili.com/video/BV1bS79z8EGS`

§ 5 拓展实验内容

修改上述实验的按键逻辑，使其分别实现下述功能:

1. 按键 1，2，4 保持原功能不变，按下按键 3 会使 LED 灯按照从左向右的顺序流水，时间间隔设置为 1s。

   要完成这一部分实验，首先需要对计数器部分进行修改，将计数器的计数值改为 1s*50MHz=50000000。对应的 Verilog 代码如下:

```
parameter MAX_COUNT = 26'd49_999_999;

always @(posedge sys_clk or negedge sys_rst_n)
begin
        if(!sys_rst_n)
                cnt<=26'd0;
        else if(cnt==MAX_COUNT)
                cnt<=26'd0;
        else
                cnt<=cnt+1'b1;
end
```

   上述代码中，MAX COUNT 的值为 49999999，当 cnt 的值等于 MAX COUNT，计数次数达到 50000000 时，cnt 的值会被重置为 0，否则 cnt 的值会加 1。这样就实现了计数器的计数值为 1s。

   其次需要对 LED 灯的状态选择进行修改，将 LED 灯的状态选择改为从左向右流水。修改部分 Verilog 代码如下:

```
else if(key[2]==0)
```

```
case(led_control)
    2'b00:led<=4'b0001;
    2'b01:led<=4'b0010;
    2'b10:led<=4'b0100;
    2'b11:led<=4'b1000;
    default:led<=4'b0000;
endcase
```

即将按键 3 按下时对应的 LED 灯状态修改为与按键 2 按下时对应的 LED 灯状态相同。该部分的实验结果视频链接如下: `https://www.bilibili.com/video/BV1Gy7dzkEYD`

2. 按下对应按键，松开按钮 LED 也会保持按键对应的逻辑运行。

   要完成这一部分实验，需要设计一个寄存器，用于储存上一个按下的按键状态，通过对寄存器状态的读取，来设计 LED 灯的状态选择。寄存器部分的 Verilog 代码如下:

```
always @ (posedge sys_clk or negedge sys_rst_n)
begin
    if(!sys_rst_n)
        last_key<=4'b1111;
    else if(key!=4'b1111)
        last_key<=key;
end
```

   设计完寄存器后，需要对 LED 灯的状态选择进行修改，将 LED 灯的状态选择改为读取寄存器的值。修改部分 Verilog 代码如下:

```
always @(posedge sys_clk or negedge sys_rst_n)
begin
    if(!sys_rst_n) begin
        led<=4'b 0000;
    end
    else if(last_key[0]==0)
        case(led_control)
            2'b00:led<=4'b1000;
```

```
                2'b01:led<=4'b0100;
                2'b10:led<=4'b0010;
                2'b11:led<=4'b0001;
                default:led<=4'b0000;
            endcase
        else if(last_key[1]==0)
            case(led_control)
                2'b00:led<=4'b0001;
                2'b01:led<=4'b0010;
                2'b10:led<=4'b0100;
                2'b11:led<=4'b1000;
                default:led<=4'b0000;
            endcase
        else if(last_key[2]==0)
            case(led_control)
                2'b00:led<=4'b1111;
                2'b01:led<=4'b0000;
                2'b10:led<=4'b1111;
                2'b11:led<=4'b0000;
                default:led<=4'b0000;
            endcase
        else if(last_key[3]==0)
            led<=4'b1111;
        else
            led<=4'b0000;
end
```

至此,LED 灯的状态选择就可以通过读取寄存器的值来进行选择了。该部分的实验结果视频链接如下: https://www.bilibili.com/video/BV1JJEczPE65

§ 6 实验总结与感想

本次实验中，我学习了如何使用 Verilog 语言进行 LED 灯的控制，掌握了按键的使用方法和 LED 灯的状态选择方法。在基础实验过程中，我需要学习编写 TCL 脚本进行引脚的自动分配操作，了解计数器的设计方法和 LED 灯的状态选择方法。其中需要了解计算机的时钟频率和计数器的计数值之间的关系，掌握 Verilog 语言的基本语法和编程方法。在这个过程中，我发现 Verilog 的语法与 C 语言存在许多的相同之处，在记住一些细微的差别之后，让我能够较为轻松地上手 Verilog 的编程。

在拓展实验中，第一部分的实验内容目的在于让我们了解计数器设计过程中的基本逻辑，实现对计数器的各项参数进行修改与重设计，实现不同的计数时间间隔。同时还需要了解 Verilog 语言中条件判断语句的基本语法，从而实现对目标按钮的功能的修改。

第二部分的实验内容目的在于让我们了解寄存器的设计过程，了解寄存器的基本逻辑和使用方法。在实验中，我需要设计一个寄存器，用于储存上一个按下的按键状态，通过对寄存器状态的读取，来设计 LED 灯的状态选择。这一部分的实验建立在第一部分的基础上，需要我们完全熟悉条件语句的逻辑后，更改 LED 灯的状态选择程序的逻辑。同时寄存器的设计要求我们能够引入一个新的变量，动态记录当前的按键状态，并在 LED 灯的状态选择中进行使用。

在实验过程中，由于对 Verilog 语言的语法不够熟悉，导致在编写代码时出现了一些错误，例如在计数器的设计中，最大值设置错误，导致计数器无法正常工作。在求助 AI 工具后，我及时修正了漏洞并完成了实验。同样在固化程序的过程中，由于第一次操作不够熟练，我在生成 JIC 文件时没有选择正确的输入文件，导致生成的 JIC 文件无法正常下载到开发板上。经过多次尝试后，我终于成功地将 JIC 文件下载到开发板上。

通过本次实验，我对 Verilog 语言有了更深入的了解，掌握了 LED 灯的控制方法和按键的使用方法。同时也对计数器和寄存器的设计有了更深入的理解。