

浙江大学

本科实验报告

课程名称:	数字逻辑设计
姓 名:	
学 院:	计算机科学与技术学院
专 业:	计算机科学与技术
邮 箱:	
QQ 号:	
电 话:	
指导教师:	蔡铭
报告日期:	2023 年 11 月 25 日

浙江大学实验报告

课程名称：_____ 数字逻辑设计 _____ 实验类型：_____ 综合 _____

实验项目名称：_____ 多路选择器设计及应用 _____

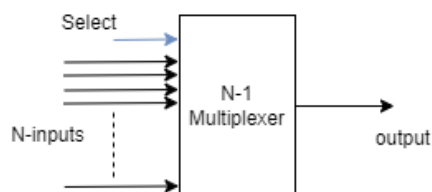
学生姓名：_____ 学号：_____ 同组学生姓名：_____

实验地点：_____ 紫金港东四 509 室 _____ 实验日期：_____ 2023 _____ 年 _____ 11 _____ 月 _____ 8 _____ 日

一、操作方法和实验步骤

实验背景

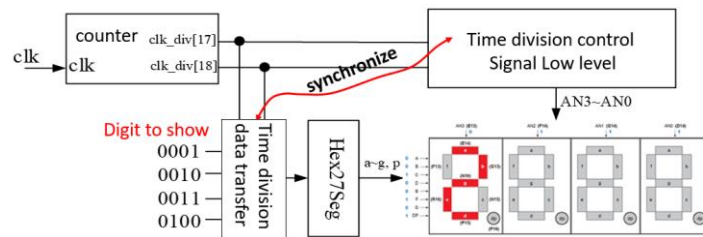
- 多路选择器：在多个输入信号中选择一个进行输出。它有多个数据输入端，一组选择信号和一个数据输出端。



- 以 4-1 Multiplexer 为例，数据输入的端口为 D0, D1, D2, D3，选择信号为 S0, S1，数据输出端口 Y，当 S1S0 分别为 0, 1, 2, 3 时，数据输出端口将分别输出 D0, D1, D2, D3 的值。
- 时钟分频：从一个时钟源获得多个不同频率的时钟信号。
- 四位七段数码管的动态显示：由于板上的四个七段数码管共用管脚，因此它在同时只能显示一样的数字。但是利用 **视觉暂留效应**，通过扫描的方式，我们每次只亮起一个数码管并显示对应数字，四个数码管循环显示。当扫描的频率比较高时，我们在视觉上就能看到四个数字同时显示。

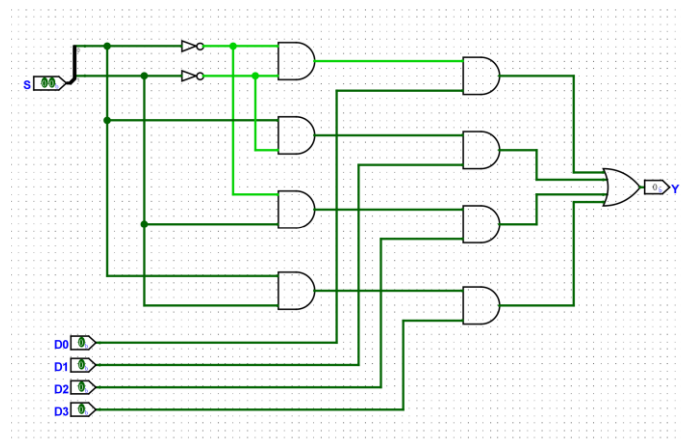
利用 Multiplexer，我们将四个十六进制数字输入作为 4-1 多路选择器的**数据输入**，将时钟分频器输出的某两位连续信号作为**选择信号**，即可周期性的选择出要打印的数字。

需要注意的是，使能信号的选择需要和输出数字的选择同步，不然打印数字的位置和数值可能和预期不符。



多路选择器的实现

- 首先，我们实现一位的 4-1 Multiplexer，在这里，为了熟悉它的电路实现，我使用了电路图的方式完成。



- 接下来，我们将一位多路选择器拓展为四位，这里我使用 Verilog 实现。

```

module Mux4to1b4 (
    input [1:0] s,
    input [3:0] I0,
    input [3:0] I1,
    input [3:0] I2,
    input [3:0] I3,
    output reg [3:0] o
);
    always @(*) begin
        case(s)
            2'b00: o = I0;
            2'b01: o = I1;
            2'b10: o = I2;
            2'b11: o = I3;
            default: o = 4'bzzzz;
        endcase
    end
endmodule

```

时钟分频

- 时钟分频器的输出在每个 **时钟信号上升沿** 自增。复位信号为同步复位，当时钟信号的正边沿来到且复位信号有效时（高电平有效），进行一次复位。

```
module clkdiv(  
    input          clk,  
    input          rst, // Active-high  
    output reg [31:0] div_res  
);  
  
    always @(posedge clk) begin // When postive edge of `clk` comes  
        if(rst == 1'b1) begin  
            div_res <= 32'b0;  
        end else begin  
            div_res <= div_res + 32'b1; // Increase `div_res` by 1  
        end  
    end  
endmodule
```

计分模块

- 接下来，我们实现 **Create Number**，对七段数码管的显示进行初始化，并且在每次按下对应按钮时对数字输入自增 1。

```
module CreateNumber(  
    input [3:0] btn,  
    output reg [15:0] num  
);  
  
    wire [3:0] A, B, C, D;  
  
    initial num <= 16'b1010_1011_1100_1101;  
  
    assign A = num[15:12] + 4'b1;  
    assign B = num[11: 8] + 4'b1;  
    assign C = num[ 7: 4] + 4'b1;  
    assign D = num[ 3: 0] + 4'b1;  
  
    always @(posedge btn[0]) num[15:12] <= A;  
    always @(posedge btn[1]) num[11: 8] <= B;  
    always @(posedge btn[2]) num[ 7: 4] <= C;  
    always @(posedge btn[3]) num[ 3: 0] <= D;  
  
endmodule
```

显示同步

- 这个模块将实现数字的动态扫描过程，让数字与使能信号的选择实现同步。

```
module DisplaySync(  
    input [15:0] Hexs,  
    input [1:0] Scan,  
    input [3:0] Point,
```

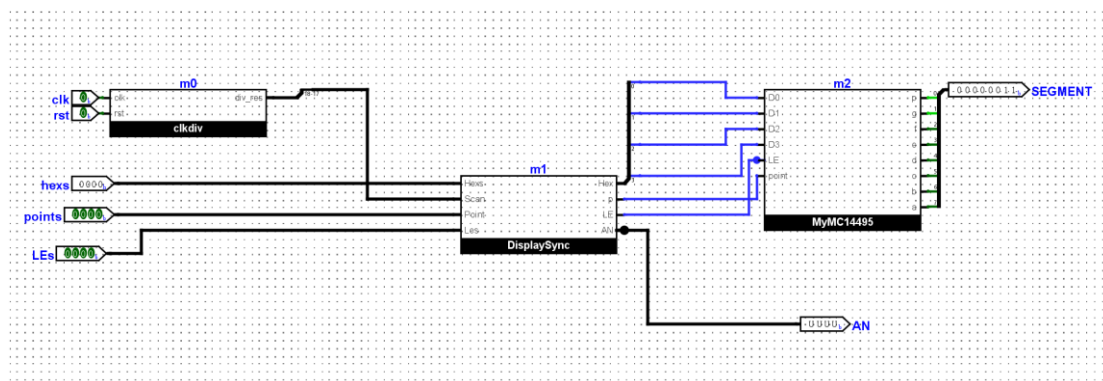
```

    input  [3:0] Les,
    output reg[3:0] Hex,
    output reg p,LE,
    output reg[3:0] AN
);
always @* begin
    case (Scan)
        2'b00 : begin    Hex <= Hexs[3:0];
                        p = Point[0];
                        LE = Les[0];
                        AN <= 4'b 1110;
                    end
        2'b01 : begin    Hex <= Hexs[7:4];
                        p = Point[1];
                        LE = Les[1];
                        AN <= 4'b 1101;
                    end
        2'b10 : begin    Hex <= Hexs[11:8];
                        p = Point[2];
                        LE = Les[2];
                        AN <= 4'b 1011;
                    end
        2'b11 : begin    Hex <= Hexs[15:12];
                        p = Point[3];
                        LE = Les[3];
                        AN <= 4'b 0111;
                    end
    endcase
end
endmodule

```

数字显示

- 最后，我们将前面编写的所有子模块集成起来，就得到了最终的动态扫描模块。
- 使用时钟分频器获得合适的扫描信号，这里选用 `clkdiv[18:17]` 两位信号。
DisplaySync 模块选择合适的数字和使能信号，并将四位数字和小数点控制信号输出到 **MyMC14495** 模块实例中得到七段信息。
- 在这里，我使用电路图完成了子模块的连接。



顶层模块

- 最后，我们将之前所有的模块组合起来，由 CreateNumber 和 DisplayNumber 实现 计分板。

```
module top(  
    input clk,  
    input [7:0] SW,  
    input [3:0] btn,  
    output[3:0] AN,  
    output[7:0] SEGMENT,  
    output      BTN_X  
);  
  
    assign BTN_X = 1'b0;  
  
    wire [15:0] num;  
  
    CreateNumber create_inst(  
        .btn(btn),  
        .num(num)  
    );  
  
    DisplayNumber disp_inst(  
        .clk(clk),  
        .rst(1'b0),  
        .hexs(num),  
        .points(SW[7:4]),  
        .LEs(SW[3:0]),  
        .AN(AN),  
        .SEGMENT(SEGMENT)  
    );  
  
endmodule
```

二、实验结果与分析

Mux4to1b4 仿真激励

- 仿真激励代码如下：

```
module Mux4to1b4_tb();  
    // Input  
    reg [1:0] s;  
    reg [3:0] I0;  
    reg [3:0] I1;  
    reg [3:0] I2;  
    reg [3:0] I3;  
  
    // Output  
    wire[3:0] o;  
  
    Mux4to1b4 m (  
        .s(s),  
        .I0(I0),  
        .I1(I1),  
        .I2(I2),  
        .I3(I3),  
        .o(o)
```

```

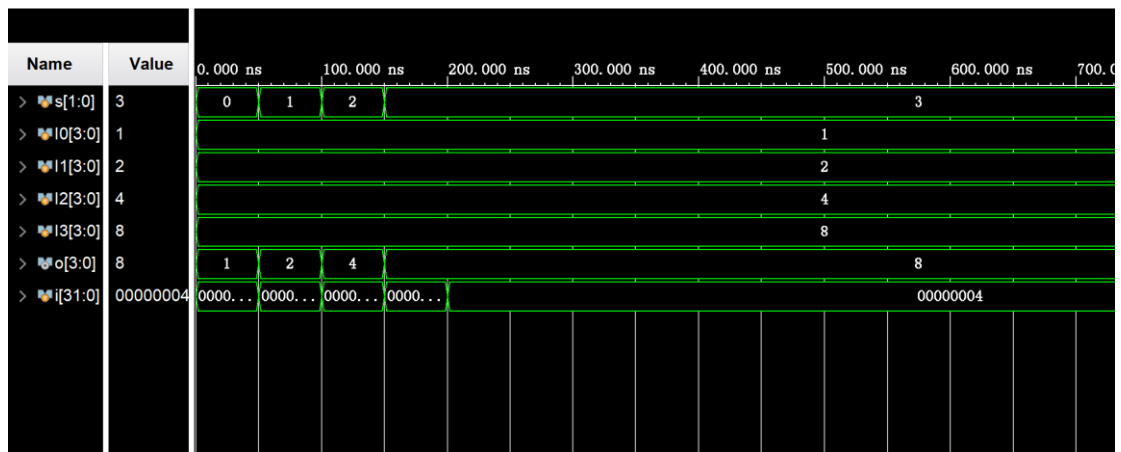
        .s(s),
        .IO(I0),
        .I1(I1),
        .I2(I2),
        .I3(I3),
        .o(o)
    );

integer i;
initial begin
    I0 <= 4'b0001;
    I1 <= 4'b0010;
    I2 <= 4'b0100;
    I3 <= 4'b1000;

    for(i=0; i<4; i=i+1) begin
        s = i;
        #50;
    end
end
endmodule

```

- 如图，I0, I1, I2, I3 的输入分别为 1, 2, 4, 8，我们让 S 的选择信号分别为 0, 1, 2, 3，可以看到输出 o 分别为 1, 2, 4, 8，实现了对输入信号的正确选择。



ScoreBoard 下板结果

- 下板的引脚约束文件如下:

```

# Main clock
set_property PACKAGE_PIN AC18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports clk]

create_clock -period 10.000 -name clk [get_ports "clk"]

# Switches as inputs
set_property PACKAGE_PIN AA10 [get_ports {SW[0]}]
set_property PACKAGE_PIN AB10 [get_ports {SW[1]}]

```

```

set_property PACKAGE_PIN AA13 [get_ports {SW[2]}]
set_property PACKAGE_PIN AA12 [get_ports {SW[3]}]
set_property PACKAGE_PIN Y13 [get_ports {SW[4]}]
set_property PACKAGE_PIN Y12 [get_ports {SW[5]}]
set_property PACKAGE_PIN AD11 [get_ports {SW[6]}]
set_property PACKAGE_PIN AD10 [get_ports {SW[7]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[0]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[1]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[2]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[3]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[4]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[5]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[6]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[7]}]

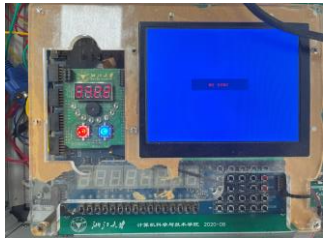


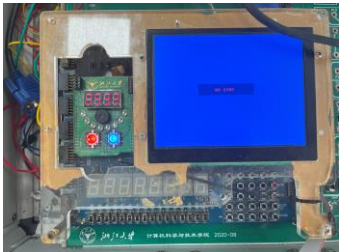
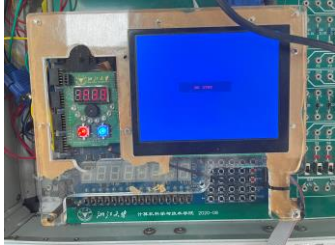



# Key as inputs
set_property PACKAGE_PIN W16 [get_ports BTN_X]
set_property IOSTANDARD LVCMOS18 [get_ports BTN_X]
set_property PACKAGE_PIN V18 [get_ports {btn[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {btn[3]}]
set_property PACKAGE_PIN V19 [get_ports {btn[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {btn[2]}]
set_property PACKAGE_PIN V14 [get_ports {btn[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {btn[1]}]
set_property PACKAGE_PIN W14 [get_ports {btn[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {btn[0]}]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets btn*]

# Arduino-Segment & AN
set_property PACKAGE_PIN AD21 [get_ports {AN[0]}]
set_property PACKAGE_PIN AC21 [get_ports {AN[1]}]
set_property PACKAGE_PIN AB21 [get_ports {AN[2]}]
set_property PACKAGE_PIN AC22 [get_ports {AN[3]}]
set_property PACKAGE_PIN AB22 [get_ports {SEGMENT[0]}]
set_property PACKAGE_PIN AD24 [get_ports {SEGMENT[1]}]
set_property PACKAGE_PIN AD23 [get_ports {SEGMENT[2]}]
set_property PACKAGE_PIN Y21 [get_ports {SEGMENT[3]}]
set_property PACKAGE_PIN W20 [get_ports {SEGMENT[4]}]
set_property PACKAGE_PIN AC24 [get_ports {SEGMENT[5]}]
set_property PACKAGE_PIN AC23 [get_ports {SEGMENT[6]}]
set_property PACKAGE_PIN AA22 [get_ports {SEGMENT[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[7]}]

```


• 下板结果

	<p>首先，我们看到计分板可以实现正常的四位数字显示</p>
	<p>接下来，我们实现最后一位数字的自增一</p>
	<p>倒数第二位的自增一</p>
	<p>第二位的自增一</p>
	<p>第一位的自增一</p>
	<div></div>

三、讨论、心得

这次实验应该是花费时间最多的一次试验了，由于计分板当中有非常多的子模块，刚刚拿到手的时候对不同模块的功能和它们之间的联系都不是太明白，因此在刚开始进行设计的时候也是一头雾水。最后跟着实验文档也算是磕磕绊绊地写完了。其中有一些让我印象深刻的点：

1. 对 Verilog 代码还不是很熟悉，有些代码是助教直接给出的，但是由于我看不懂，不能够很好地理解它的实现方式，对功能也是一知半解。
2. 在下板的过程中，由于我对按钮的映射和约束方式不是很了解，因此尝试了很久才摸索出来到底是按哪些按钮才能实现计分板的自增。而且在实验过程中会出现按一下按钮，自增很多下的情况，这是由于我们没有在代码中加入防抖动模块，因此最后的显示效果并不尽人意。
3. 应该要慢慢转变电路图中连线的思维，转向 Verilog 的连线方法和实现逻辑，才能在接下来的实验当中更加地顺利。