实验七：总线实验

实验时间： 2017年12月11日 第十四周 星期一

实验者：16307130194 陈中钰 16级 计算机科学技术学院

座位号：30

指导老师：唐志强

1 实验目的

- 了解并掌握寄存器的原理和设计，加深对时序电路的理解
- 掌握 RAM 的储存功能的实现机制
- 能熟练设计、运用多于1位宽度的多路选择器
- 能理解实验的数据通路，加深模块化的理解，并能熟练进行电路微操作

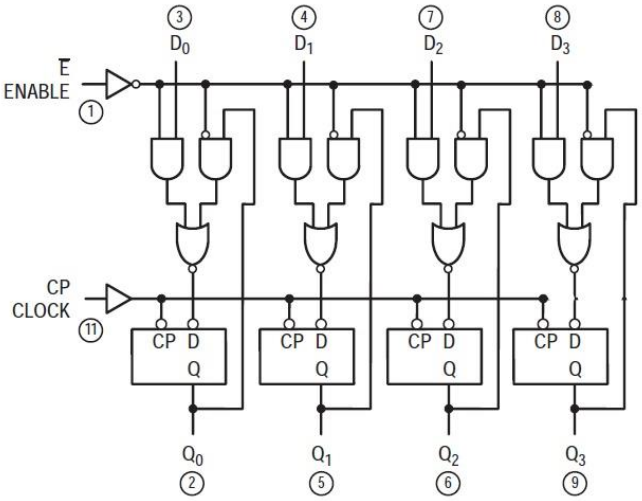
2 实验原理

2.1 74LS377寄存器：带使能的4位宽 D 触发器

- 输入：输入信号 D[3:0]，开关输入使能信号 EN，时钟信号 CLK；输出：输出信号 Q[3:0]，并在对应的 LED 上显示
- 状态表

EN	D[i]	Q[i](n)	Q[i](n+1)	操作
0	0	0	0	无变化
0	0	1	1	
0	1	0	0	
0	1	1	1	
1	0	0	0	置数
1	0	1	0	
1	1	0	1	
1	1	1	1	

- 具体实现：当 EN=0时，Q 不变；当 EN=1时，Q<=D 置数
- 逻辑电路图



2.2 RAM（4位）

- 输入：时钟信号 clk，控制写入信号 wen，地址 addr[3:0]，输入数据 din[3:0]；

输出：输出数据 qout[3:0]

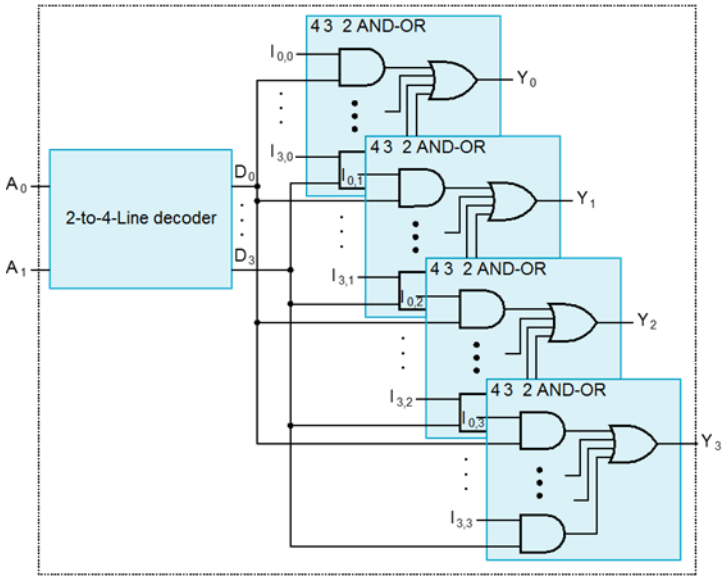
- 存储机制：模块中定义了 reg[3:0]ram[0:15]，也就是16个 reg 型的 ram[3:0]变量，这些变量从0~15编号，而这些编号也就是地址，每个地址对应一个 ram[3:0]变量，也就是对应一个存储在 RAM 中的值。通过 SW[4:1]输入二进制地址 addr[3:0]，值的范围为0000~1111，也就是0~15，恰好与编号一一对应，故 ram[addr]就是存储在 addr 的4位二进制数
- 写入：当 wen 控制信号为1，且时钟上升沿到达时，会把 din 中的4位二进制数值写入 ram[addr]，为同步过程，可以通过 always@()语句实现
- 读取：异步向 qout 输出 ram[addr]的值，并在对应 LED 上显示

2.3 4位宽4-1 MUX

- 输入：选择信号 s[1:0]，被选择的4个4位宽向量 i0[3:0]，i1[3:0]，i2[3:0]，i3[3:0]；输出选择出的长度为4的向量 y[3:0]
- s[1:0]=00，01，10，11时分别对应选择数据通路中该器件的 a，b，c，d 输入
- 可以通过 case()语句，对 s[1:0]的值进行判断，把对应输入向量赋值给输出向量
- 由于 MUX 是异步的，因此所有输入值都在 always@()语句的敏感表中，可以用*代替
- 选择表格

控制		输出
s[1]	s[0]	y
0	0	i0 / a
0	1	i1 / b
1	0	i2 / c
1	1	i3 / d

- 逻辑电路图



2.4 4位宽2-1 MUX

- 输入：选择信号 s，被选择的2个4位宽向量 i0[3:0]，i1[3:0]；输出被选择的长度为4的向量 y[3:0]
- s=0，1时分别对应选择数据通路中的该器件的 a，b 输入

- 可以通过 case()语句，对 s 的值进行判断，把对应输入向量赋值给输出向量
- 由于 MUX 是异步的，因此所有输入值都在 always@()语句的敏感表中，可以用* 代替
- 选择表格

控制 s	输出 y
0	i0 / a
1	i1 / b

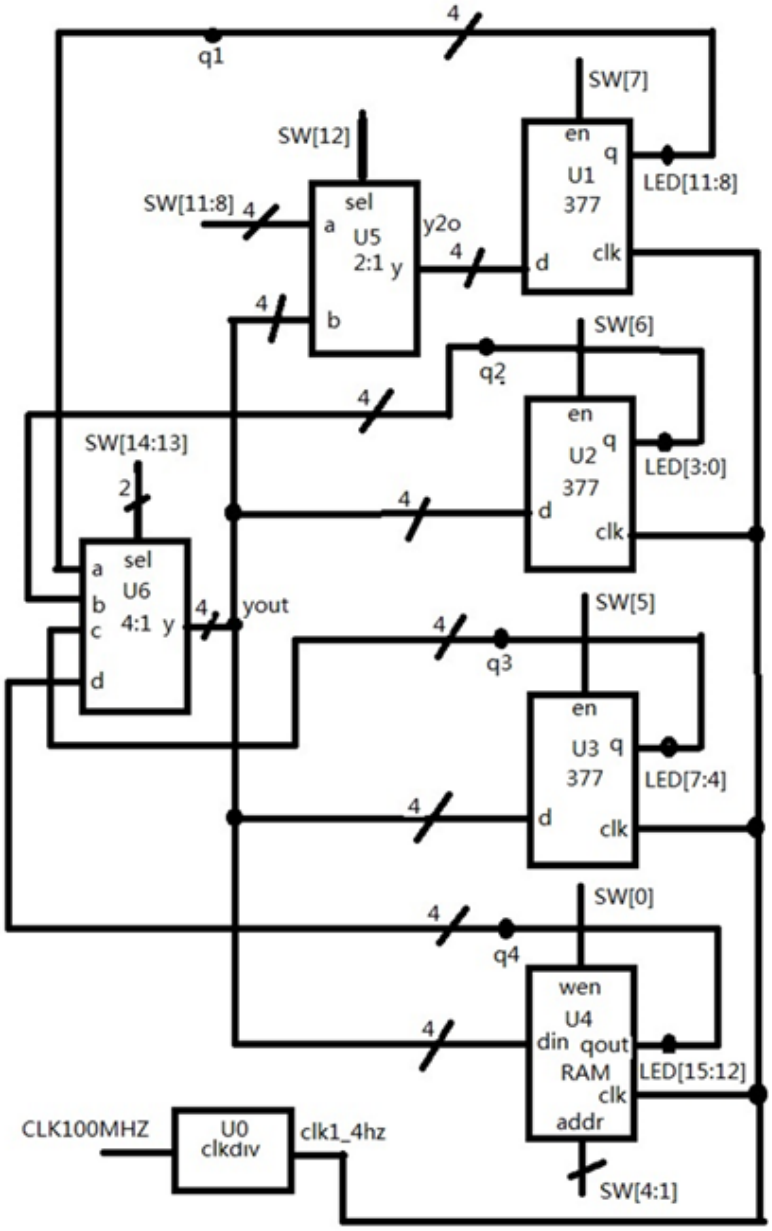
2.5 时钟分频模块

- 取的是 q[24]的值，输出时钟信号约为4Hz

3 实验分析

- U0为时钟分频模块；U1，U2，U3为3个4位宽寄存器；U4为 RAM；U5为4位宽的2-1MUX；U6为4位宽的4-1MUX
- U0把时钟频率从100MHz 降低到约4Hz，并控制 U1~U4
- 2-1多路复用器 U5的 SW[11:8]是唯一的数据输入处，通过调 SW[11:8]输入4位二进制数，通过调节 SW[12]控制 MUX 选择 a 或 b 输入，并传送给 U1寄存器
- 通过 SW[7]使能控制 U1，当 SW[7]=1时，U1的值被输入值更新，并输出到 LED[11:8]以及4-1MUX 的输入端 a
- 通过调节 U6的 SW[14:13]可控制 MUX 选择 a，b，c，d 输入中的一个，并传送给2-1MUX 的 b 输入、U2、U3、U4的输入端
- 当 U2的 SW[6]=1时 U2被触发，U2储存的值被4-1MUX 传来的值更新，并输出到 LED[3:0]以及4-1MUX 的输入端 b
- 当 U3的 SW[5]=1时 U3被触发，U3储存的值被4-1MUX 传来的值更新，并输出到 LED[7:4]以及4-1MUX 的输入端 c
- 当 U4的 SW[0]=1时 U4的写入被触发，U4中储存在地址 SW[4:1]处的值被4-1MUX 输出值更新，并输出到 LED[15:12]以及4-1MUX 的 d 输入端；其他时候，不断输出地址 SW[4:1]处储存的值到4-1MUX 的 d 输入端，并在 LED[15:12]显示

- 逻辑电路图



4 实验内容

4.1 设计思想

- 独立实现各个模块
- 把各个模块的输入、输出连接起来

4.2 Verilog 代码

```
module bus(input clk, input [14:0] SW, output [15:0] LED);
    wire myclk;
    wire [3:0] d1;
    wire [3:0] y;
    wire [15:0] light;
    clkdiv U0(clk,myclk);
    register U1(myclk,SW[7],d1[3:0],light[11:8]);
    assign LED[11:8]=light[11:8];
```

```
    register U2(myclk,SW[6],y[3:0],light[3:0]);
    assign LED[3:0]=light[3:0];
    register U3(myclk,SW[5],y[3:0],light[7:4]);
    assign LED[7:4]=light[7:4];
    ram2 U4(myclk,SW[0],SW[4:1],y[3:0],light[15:12]);
    assign LED[15:12]=light[15:12];
    MUX_2_1 U5(SW[12],SW[11:8],y[3:0],d1[3:0]);
    MUX_4_1
U6(SW[14:13],light[11:8],light[3:0],light[7:4],light[15:12],y[3:0]);
endmodule

module register(input myclk, input en, input [3:0] d, output reg [3:0]
q);
    always@(posedge myclk,posedge en)
    begin
        if(en)
            q<=d;
    end
endmodule

module ram2(input clk, input wen, input [3:0] addr, input [3:0] din,
output [3:0] qout);
    reg[3:0]ram[0:15];
    always@(posedge clk)
      if(wen)
        ram[addr]<=din;
      assign qout=ram[addr];
endmodule

module clkdiv(input mclk, output clk1_4hz);
    reg [27:0]q;
    always@(posedge mclk)
         q<=q+1;
    assign clk1_4hz=q[24];
endmodule

module MUX_4_1(input [1:0] s, input [3:0] i0, input [3:0] i1, input
[3:0] i2, input [3:0] i3, output reg [3:0] y);
    always@(*)
    begin
        case(s)
        2'b00:y<=i0;
        2'b01:y<=i1;
        2'b10:y<=i2;
```

```
        2'b11:y<=i3;
        endcase
    end
endmodule

module MUX_2_1(input s, input [3:0] i0, input [3:0] i1, output reg
[3:0] y);
    always@(*)
    begin
        case(s)
        1'b0:y<=i0;
        1'b1:y<=i1;
        endcase
    end
endmodule
```

4.3 改进

- 不必使用 light[15:0]来作为 LED[15:0]的中间量，在模块中，LED[15:0]信号可以直接作为输入、输出信号，代码会更简洁

4.4 实验操作（输入1111，0000两个值，分别存在 U2，U3寄存器，并利用 RAM 为中介进行值的交换）

- SW[14:0]=15'b000000000000000置零
- SW[11:8]=4'b1111，而2-1MUX 的选择控制 SW[12]=0，选择输出的是 a，也就是开关 SW[11:8]输入的值1111，输出到 U1的输入口，使 SW[7]=1触发 U1，LED[11:8]全亮，表示1111存入 U1，SW[7]=0关闭 U1
- 此时 SW[14:13]==2'b00，选择输出的是 a，也就是 U1的输出值1111，并输出到 U2、U3、U4的输入口，使 SW[6]=1触发 U2，LED[3:0]全亮，表示1111存入 U2，关闭 SW[6]
- SW[11:8]=4'b0000，而2-1MUX 的选择控制 SW[12]=0，选择输出的是 a，也就是开关 SW[11:8]输入的值0000，输出到 U1的输入口，SW[7]=1触发 U1，LED[11:8]全暗，表示0000存入 U1，SW[7]=0关闭 U1
- 此时 SW[14:13]==2'b00，选择输出的是 a，也就是 U1的输出值1111，并输出到 U2、U3、U4的输入口，使 SW[5]=1触发 U3，LED[7:4]仍然是暗的，表示0000存入了 U3，关闭 SW[5]
- SW[14:13]=2'b01，使4-1MUX 选择来自 U2输出的 b 输入1111，输出到 y，也就是输出到 U2、U3、U4的输入口
- SW[0]=1触发 U4，而 SW[4:1]为0000，此时把输入值1111写入地址0000处，并输出到 LED[15:12]，全亮，并输出到4-1MUX 的 d 输入端
- SW[14:13]=2'b10，选择来自 U3输出的 c 输入0000，并输出到 y，也就是输出到 U2、U3、U4的输入口，使 SW[6]=1，触发 U2，把0000写入 U2，LED[3:0]全灭，表示0000存入 U2，SW[6]=0关闭 U2
- SW[14:13]=2'b11，选择来自 U4在地址0000储存的值的输出1111，并输出到 y，也就是输出到 U2、U3、U4的输入口，使 SW[5]=1触发 U3，把1111写入 U3，LED[7:4]全亮，表示1111存入 U3，SW[5]=0关闭 U3
- 操作完成

5 实验结论

- 实现了把1111写入 U2，0000写入 U3，并利用 RAM 为中介，交换 U2、U3中的值
- 寄存器具有储存、转移的功能
- RAM 具有储存多个数据的功能

6 实验感想

- 最初疏忽没有把4-1MUX 的 y 输出和 U2、U3的 d 输入连接起来，导致 SW 改变时 U2、U3完全没有反应，后来通过仔细检查才发现并更正
- 虽然4位宽的 D 触发器可以使用4个1位宽 D 触发器以及若干与门、或门，通过分层设计、模块化的思想来实现，但是这样做反而会复杂化，更不直观，且使得 debug 更困难；4-1MUX、2-1MUX 也是如此。因此需要利用 Verilog 语言的便捷之处，以简化问题以及代码设计，以避免粗暴的硬件翻译，也更直观，便于 debug
- 要能熟练进行微操作，则必须要十分清楚各个器件的作用，了解各个输入、输出，以及整个数据通路的关系，并明晰操作流程和效果
- 在较大电路的设计中，模块化具有很强的优势，能使得整个电路虽然庞大但是不乱