实验原理

1、D触发器

D触发器的RTL代码和约束代码根据组的模块接口自行编写，仿真代码如下：

```
`timescale 1ns/1ps

module testbench ();

reg clk;
reg clr;
reg en ;
reg d  ;
wire q;

initial begin
    clr = 1'b1;     // 初始复位，所有输入初始化
    en  = 1'b0;
    clk = 0;
    d   = 0;

    #10;            // 写入
    clr = 1'b0;
    en  = 1'b1;
    d   = 1'b1;
    #10 d = 1'b0;
    #10 d = 1'b1;

    #10;            //读取
    en = 1'b0;
    d  = 1'b0;

    #10 clr = 1'b1; //异步清零

    #50 $finish;
end

always #5 clk = ~clk;   //生成时钟

dff u_dff(              //模块例化
    .clk(clk),
    .clr(clr),
    .en(en),
    .d(d),
    .q(q)
);

endmodule
```

下板自行测试写入、读取、清零三种状态：

- 写入：SW23往上拨使能打开，上下拨动SW0改变数据输入，输出GLD0需同步变化；
- 读取：SW23往下拨使能关闭，上下拨动SW0改变数据输入，输出GLD0无变化；
- 清零：也叫复位，在GLD0亮的情况下，按下按键开关S1，GLD0灭。

后续实验都会有全局复位信号，复位信号接按键开关非拨码开关，按键开关不按的时候对应输出0，按下输出1，复位时，所有的时序电路都要回到初始状态。

2、时钟和复位信号

所有时序逻辑都需要时钟clk信号，实验中，clk信号接到FPGA开发板的Y18引脚，该引脚会输出一个稳定的100MHz的时钟，后续实验的clk信号都按照此方式处理，不再重复介绍。

所有的时序逻辑都应有复位，即本实验的清零clr，通常触发器叫清零，更一般的是叫复位rst（reset的简写）。复位是对整个电路所有的时序逻辑复位，而不是只对其中一个模块复位。

3、寄存器文件

CPU的指令集架构总是定义了一些通用寄存器（riscv指令集是32个通用寄存器），用于在内存与CPU运算部件之间暂存数据，是CPU的核心单元。这一些寄存器组成了寄存器文件（register file），也叫寄存器堆。

寄存器文件的功能是，将输入的数据写入指定的寄存器，也可以读取指定寄存器的数据。本实验实现的是简化版的寄存器堆，只有一个读端口一个写端口。读端口是异步逻辑，只要地址改变，输出的数据立即变，不需要与时钟边沿同步。

8个8位的寄存器可使用如下向量类型的数组定义，reg 后的 [7:0]表示8位宽，regfile后的 [7:0]表示有8个元素，可直接用regfile[0]、regfile[1]的方式访问对应元素，regfile[0]表示第0个8位的寄存器。注意接口信号不能定义成数组，接口信号定义成数组会有语法warning，综合的时候也会失败。

```
reg [7:0] regfile [7:0];
```

> **注意**
>
> 时钟clk请使用上升沿posedge触发，工程中时钟很少用negedge触发
>
> 按键开关，只要按一下，不是一直按着！！！
>
> 所有实验中未注明的细节可自行决定

写完代码，生成bit文件，上板测试写入、读取、清零三种功能，推荐的测试步骤如下，具体的寄存器编号和数据在课上检查时自行决定：

- 所有拨码开关都往下拨，输入低电平。
- 写入：拨动SW22-SW20决定写的寄存器编号，再拨动SW7-SW0改变待写入的数据，写入的数据不能用全0测试。再将使能信号SW23往上拨将数据写入，再把SW23往下拨关闭使能。重复以上操作，向另外一个寄存器写入数据。
- 读取：先确定SW23往下拨关闭使能，上下拨动SW10-SW8改变读取的寄存器编号，与上一步写入的编号一致，观察LED灯显示的数据是否与写入的数据一致。
- 清零：在LED灯亮的情况下，按一下按键开关S1，所有的LED灯灭掉。
- 读取：再换一个寄存器编号读取，看LED是否能亮。

4、仿真文件和模块例化

仿真文件的结构大家可以参照D触发器的仿真文件。

- 第一行通常是 `timescale 1ns/1ps`，表明仿真的时间单位和精度，也就是#1 延迟多少时间，这里是 1ns 。
- 接着是仿真模块定义 `module testbench ();`，无需输入输出信号，模块名可以自定义，模块名与文件名保持一致比较规范。
- 再是信号定义和信号构造，信号的变化根据被测模块的功能和测试用例构造。没有规律的信号变化使用initial块构造，有规律、有逻辑关系的信号通过always块或者assign语句赋值。
- 时序电路仿真需要产生一个时钟信号，生成的方式如以下代码所示：定义时钟信号clk，首先在initial块中将clk初始化为0，另用一个always块周期性的让clk信号翻转，周期根据实际需要调整，clk初始化也可以在定义时初始化 `reg clk = 0;` ，仅限在仿真的时候这样写，功能代码不建议。将时间单位设定为1ns，以下代码生成周期为10ns的时钟，即频率是100MHz，clk每隔5ns翻转一次。

```
reg clk;
initial begin
    clk = 0;
end
always #5 clk = ~clk;
```

- 最后RTL模块的例化，即调用功能模块作为被测试单元（DUT，design under test），模块例化的语法如下，明确定义将信号连接到模块端口，例化后的对象称为模块的实例（instance）。

```
<module_name>  <instance_name> (
    //被例化的端口与例化的端口
    .<port_name1> (<signal_name1>),
    .<port_name2> (<signal_name2>)
);
```

- D触发器的例化如下：

```
dff u_dff(
    .clk(clk),
    .clr(clr),
    .en (en ),
    .d  (d  ),
    .q  (q  )
);
```

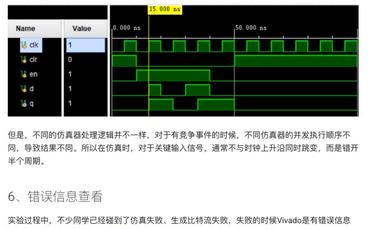
5、功能仿真采样时刻

对于功能仿真，没有考虑建立时间、保持时间等时序，波形是理想化的。仿真中可能会碰到有些输入变化跟时钟上升沿同步变化，这种情况是采样变化前的值还是变化后的值？在Vivado中会在时钟上升沿后的值，赋给其他信号。

对于D触发器，仿真代码如下稍作改动

```
`timescale 1ns/1ps

module testbench (
);

reg clk;
reg clr;
reg en;
reg d;
wire q;

initial begin
    clr = 1'b1; // 初始状态为复位态
    en = 1'b0;
    clk = 0;
    d = 0;
    #10;        // 测试跟随状态
    clr = 1'b0;
    en = 1'b1;
    #5 d = 1'b1;  // 延后d的变化与时钟边沿对齐
    #5 d = 1'b0;
    #10;
    d = 1'b1;
    #10;        //测试保持状态
    en = 1'b0;
    d = 1'b0;
    #10;        //测试异步清零
    clr = 1'b1;
    #50
    $finish;
end

always #5 clk = ~clk;   //生成时钟

dff u_dff(
    .clk(clk),
    .clr(clr),
    .en(en),
    .d(d),
    .q(q)
);

endmodule
```

运行仿真查看波形，如下图所示，可以看到输出没变同开始的仿真逻辑的输出，15ns的位置，d的变化与时钟同步，输出q为1用的是d变化后的值。



但是，不同的仿真器处理逻辑并不一样，对于有竞争事件的时候，不同仿真器的并发执行顺序不同，导致结果不同。所以在仿真时，对于关键输入信号，通常不与时钟上升沿同时跳变，而是错开半个周期。

6、错误信息查看

实验过程中，不少同学已经碰到了仿真失败、生成比特流失败，失败的时候Vivado是有错误信息提示，具体请查看常见问题与错误信息