**实验13**

**时序逻辑元件**

# **实验介绍**

本次实验我们将介绍一些基础的时序逻辑元件。时序逻辑元件是构建各种数字电路和系统的基础。它们能够根据输入信号的变化，按照预定的时间顺序产生输出。这种特性使得时序逻辑元件在数字信号处理、数据通信、计算机架构以及许多其他领域都有广泛的应用。

触发器是最基本的时序逻辑元件之一，它有两个稳定的状态，通常被称为"0"和"1"。当触发器的输入端接收到特定的信号时，它会从当前状态翻转到另一个状态。这种翻转可以是瞬间的，也可以是延迟的，具体取决于触发器的类型和配置。常见的触发器类型包括RS触发器、JK触发器、D触发器和T触发器等。

计数器是一种能够存储和计算输入脉冲数量的时序逻辑元件。每当计数器的输入端接收到一个脉冲信号，它的内部状态就会改变，通常是通过增加或减少一个计数值来实现的。计数器在数字电路设计中非常有用，可以用于实现定时、计数、序列发生等多种功能。

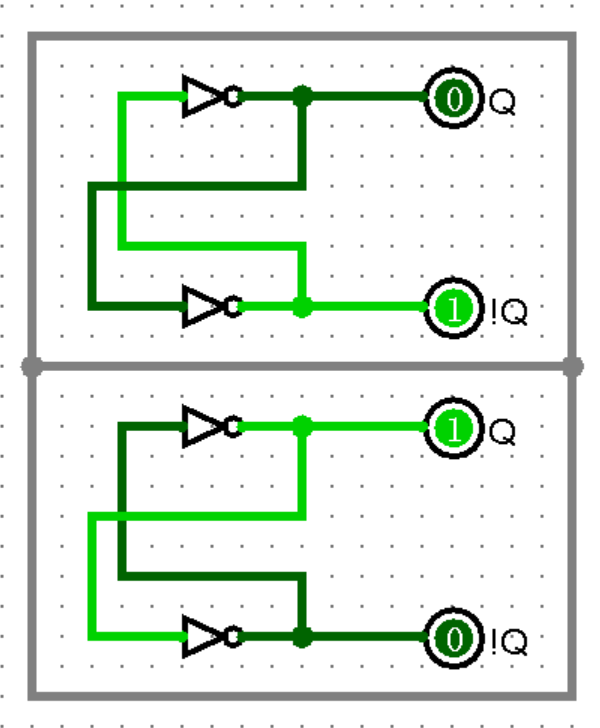
此外，我们还会讨论寄存器（Register）和移位寄存器（Shift Register）。寄存器是一种能够存储多位二进制信息的时序逻辑元件。它通常具有多个输入端、输出端和控制端，可以执行多种操作，如数据加载、数据保存和数据传输等。移位寄存器则是一种特殊的寄存器，它可以在时钟信号的控制下，将存储在其中的数据逐位向左或向右移动。移位寄存器在数据通信、序列检测和数据存储等方面都有广泛的应用。

# **实验内容**

## 1 锁存器与触发器

### ****1.1 双稳态电路****

我们首先观察下面的电路：

[](https://soc.ustc.edu.cn/Digital/lab5/figs/circuit_1.png)

从电路结构可知，若 Q=0，经过下面的非门取反，可得 ~Q=1，再经过上方的非门反馈到输入端，又保证了 Q=0。由于两个非门首尾相接的逻辑锁定，因而电路能自行保持在 Q=0,~Q=1 的状态，对应上方电路的输出结果。反之，若 Q=1,~Q=0，则对应下方电路的输出结果。

简单概括，这个电路与之前介绍的组合逻辑电路有三个主要的不同：

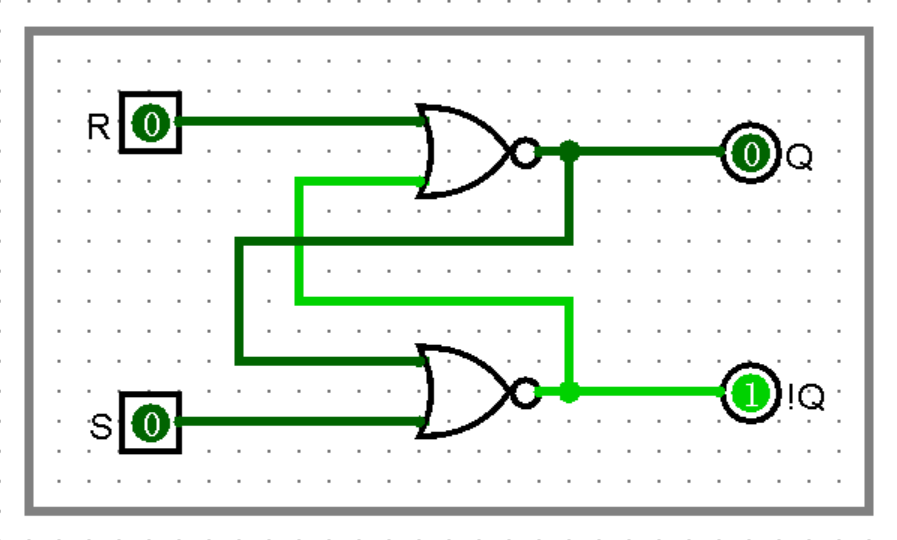
1. 没有输入端口。电路无法接收来自外界的输入信号，因而也无法改变自身的状态。
2. 有两种可能的状态。电路的结构是固定的，但是输出端有两种可能的输出结果。
3. 有反馈。两个非门的输出端口分别连接到了彼此的输入端口，即输出端会反馈影响到输入端。

像上面这样具有 0、1 两种逻辑状态，一旦进入其中一种状态，就能长期保持不变的单元电路，称为双稳态存储电路，简称双稳态电路。接下来所讨论的锁存器和触发器均属于双稳态电路。

### ****1.2 锁存器****

双稳态电路是众多时序逻辑电路的基础。这是因为它可以存储一定的信息。如果为其增加控制单元以改变内部的内容，我们就得到了锁存器。锁存器（Latch）是一种对脉冲电平敏感的双稳态电路，它具有0和1两个稳定状态，一旦状态被确定，就能自行保持，直到有外部特定输入脉冲电平作用在电路一定位置时，才有可能改变状态。这种特性可以用于置入和存储1位二进制数据。

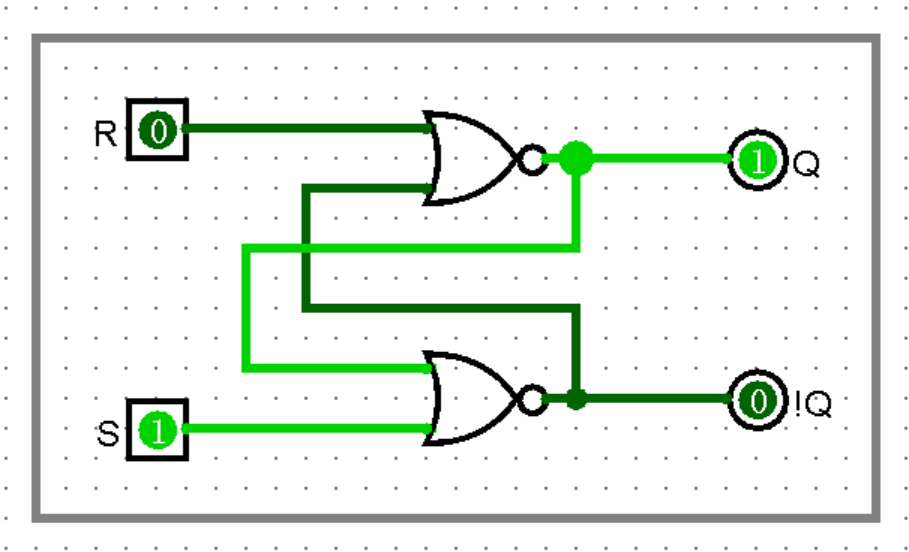
将双稳态电路的非门换成或非门，则构成下图所示的 RS 锁存器。它是一种具有最简单控制功能的双稳态电路。图中，S 和 R 是两个输入端，Q 和 ~Q是两个输出端。我们定义 Q=0 为锁存器的 0 状态，Q=1 为锁存器的 1 状态。

[](https://soc.ustc.edu.cn/Digital/lab5/figs/SR_Latch.png)

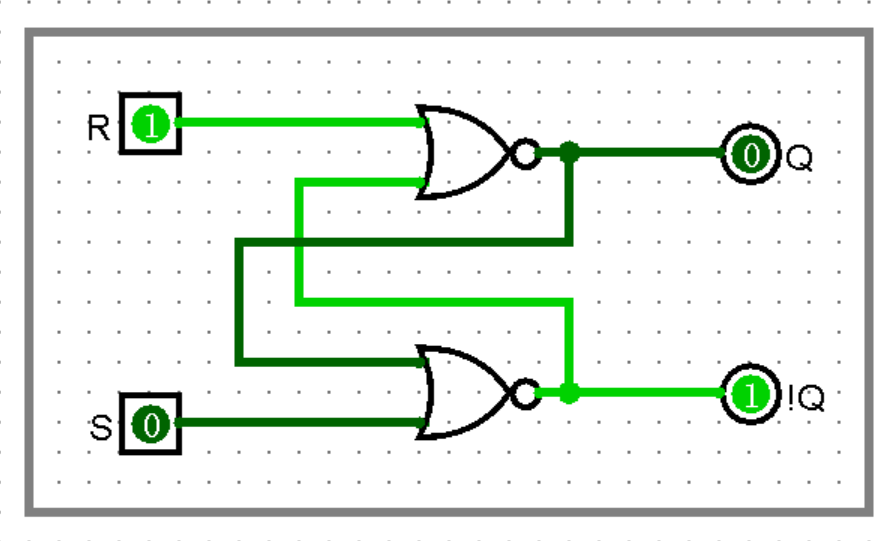
我们来分析一下电路的工作原理。

1. 当 S=0，R=0 时，此时或非门就相当于非门，因为另一端输入为 0 不影响或非运算的结果。RS 锁存器将会保持其原本的状态不变，可以存储 1bit 二进制数据。

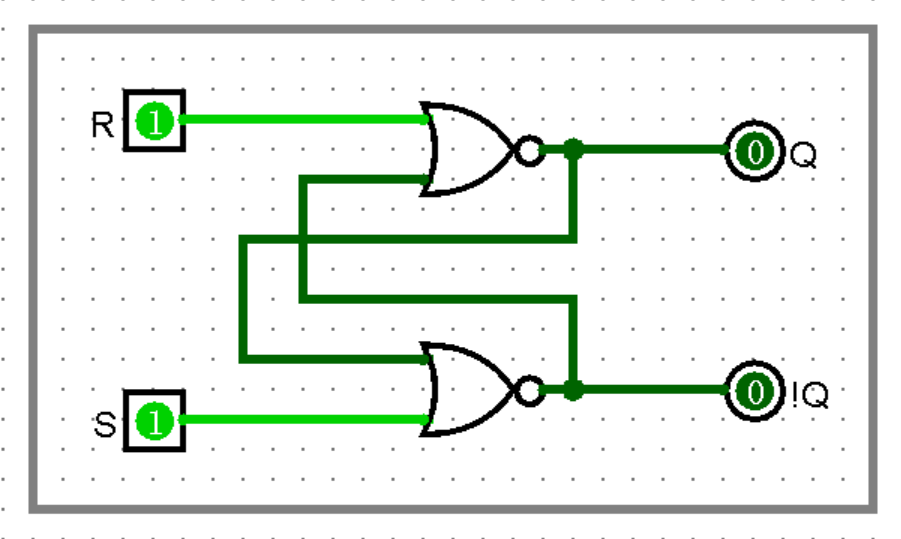
2. 当 S=1，R=0 时，如下图所示，此时 S 端对应的或非门固定输出 0，即~Q=0,Q=1。RS 锁存器处于 1 状态。我们称之为置位（Set）。

[](https://soc.ustc.edu.cn/Digital/lab5/figs/SR_Latch_S.png)

3. 当 S=0，R=1 时，如下图所示。此时 R 端对应的或非门固定输出 0，即 Q=0。RS 锁存器处于 0 状态。我们称之为复位（Reset）。

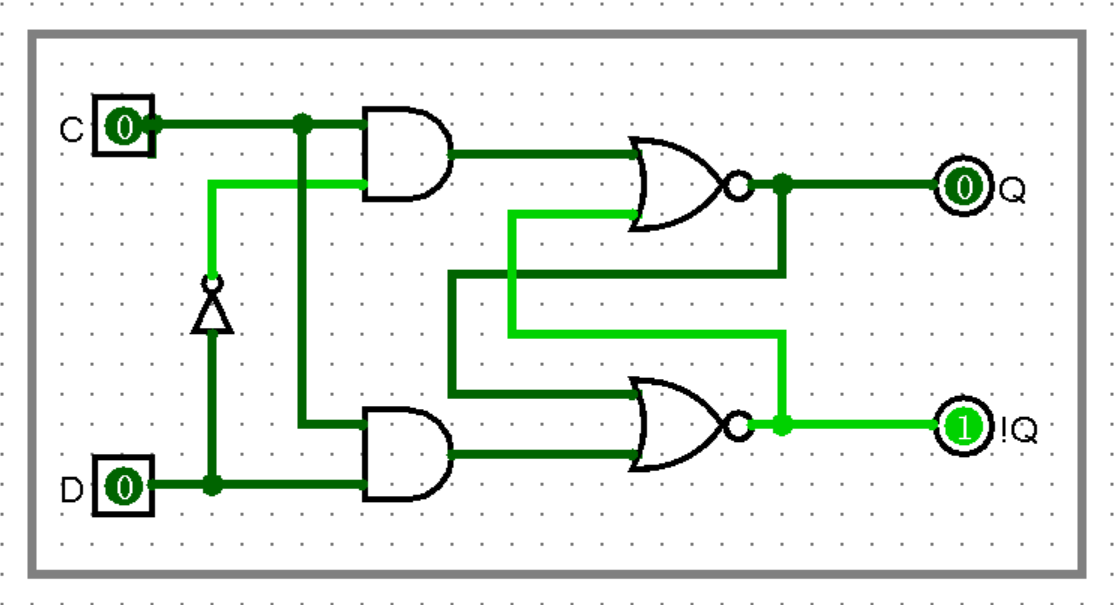
[](https://soc.ustc.edu.cn/Digital/lab5/figs/SR_Latch_R.png)

4. 当 S=1，R=1 时，如下图所示，此时 S 端和 R 端对应的或非门都固定输出 0，即 Q=0,~Q=0。此时锁存器处于非 0 非 1 的未定义状态，违背了 Q 和 ~Q 始终相反的设计初衷。

[](https://soc.ustc.edu.cn/Digital/lab5/figs/SR_Latch_SR.png)

当 RS 锁存器的 S 端和 R 端均为 1 时，电路处于未定义状态。此时如果 S 先变为 0，则相当于复位 Reset，若 R 先变为 0，则相当于置位 Set。若二者同时变为 0，则电路会根据或非门的延迟高低决定最终应当跳转到的状态，而这是我们无法预知的。为了保证 RS 锁存器始终处于有效的工作状态，我们一般约定 S 端和 R 端不同时为 1，即 S·R≡0。

除了 RS 锁存器，我们还将学习另一种锁存器：D 锁存器。与 RS 锁存器不同，D 锁存器在工作中不存在非定义状态，因而得到广泛应用。

[](https://soc.ustc.edu.cn/Digital/lab5/figs/D_Latch.png)

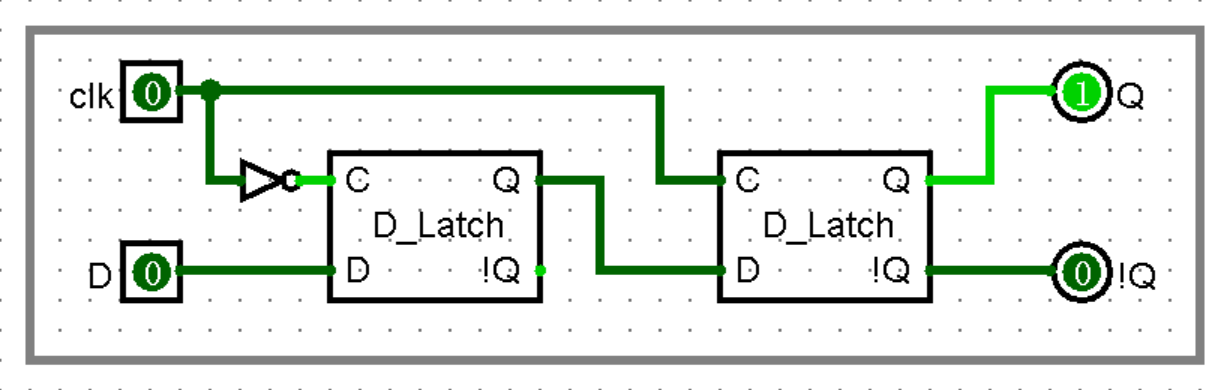
如上图所示，D 锁存器在 RS 锁存器的基础上，引入了两个与门和一个非门。除此之外，我们还引入了一个新的控制信号 C。

当 C=0 时，无论 D 端的输入是什么，与门的输出都为 0。此时相当于 RS 锁存器的输入为 S=0，R=0，则 D 锁存器处于保持状态；当 D=1 时，我们便可以忽略与门。此时 RS 锁存器的输入为 S=D，R=~D，显然有 S⋅R=D⋅~D≡0。因此 D 锁存器没有未定义状态，且内部存储的数值与 D 端的输入保持一致。

### ****1.3 触发器****

D 锁存器看起来十分完美了，但当 D 端输入不是那么平滑，存在一定的「抖动」时，锁存器内部便会跟着进行状态抖动。换而言之，D 锁存器的稳定性较差。有没有什么办法能够使得其具有良好的稳定性呢？分析可知，D 锁存器在 C=1 的一段时间都可以进行更新，从而带来了不稳定性。如果我们能够限制 D 锁存器仅在很小的一段时间进行更新呢？假定 C 端信号仅在很小的一段时间内保持 1，其他时刻都为 0。此时信号的突变间隙大于高电平维持长度，因而无法将干扰结果写入锁存器。

然而，我们无法无限制地提升 C 端信号的变化频率，因此学者们换了一个思路：不是在高电平时写入，而是在低电平转换为高电平的瞬间写入。这就得到了触发器。

[](https://soc.ustc.edu.cn/Digital/lab5/figs/D_FF.png)

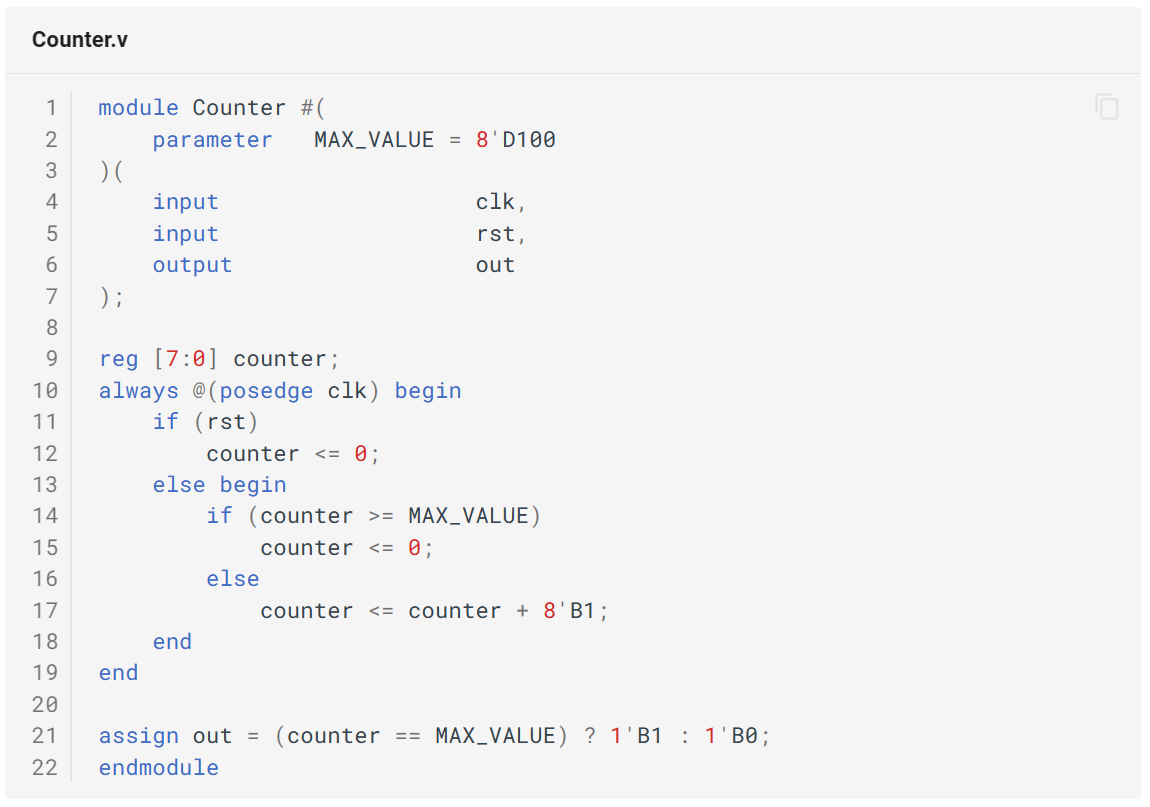
如上图所示，通过两个 D 锁存器级联，并加入一个非门，就得到了 D 触发器（D flip flop）。这里我们让控制信号以一定的周期进行高低电平翻转，类似于一个时钟 Clock 信号，因此记作 clk。电路的分析如下：

* 当 clk 为低电平时，前一个锁存器处于更新状态，此时 D 端输入可以直接写入前一个锁存器。后一个锁存器处于保持状态，无论前一个锁存器输出如何，后一个锁存器均保持自身原先的数值不变。
* 当 clk 为高电平时，前一个锁存器处于保存状态，此时 D 端输入无法写入前一个锁存器。后一个锁存器处于更新状态，将会写入前一个锁存器的值。这个时候毛刺信号均无法影响到后一个锁存器，因而增强了电路的稳定性。

通过非门，两个 D 锁存器的时钟存在一个 180° 的相位差（也就是是相差半个时钟周期），从而实现，只在时钟上升沿的时候读取输入并输出，其他时候输入的变化不会传导到输出端，去除了输入可能存在的毛刺，得到了稳定的输出。

## 2 其他时序元件

下面是我们已经很熟悉的代码：



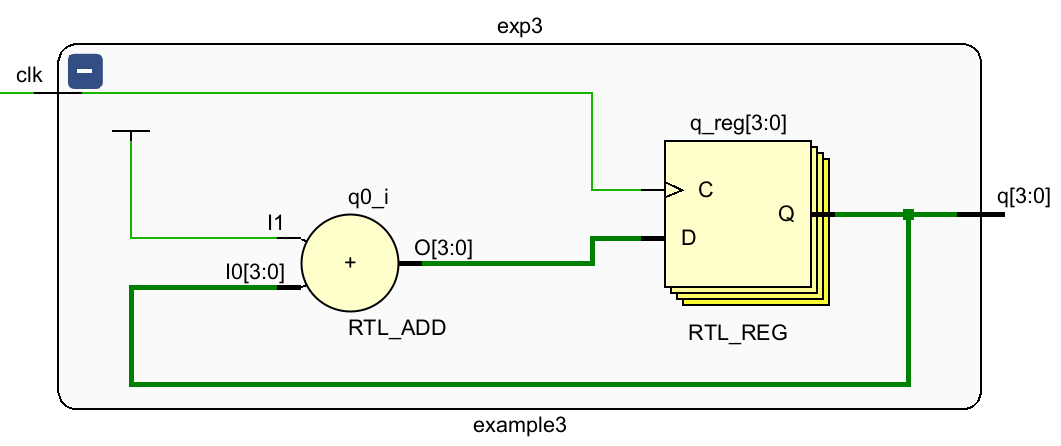
我们知道，这段代码包含了一个基本的计数器，以我们设定的间隔输出高电平脉冲。

### ****2.1 计数器****

我们先前也提到过，always 语句可以使用 posedge 和 negedge 关键字指定电平变化的事件触发。下面是一个简单的自增计数器的 Verilog 代码实现：



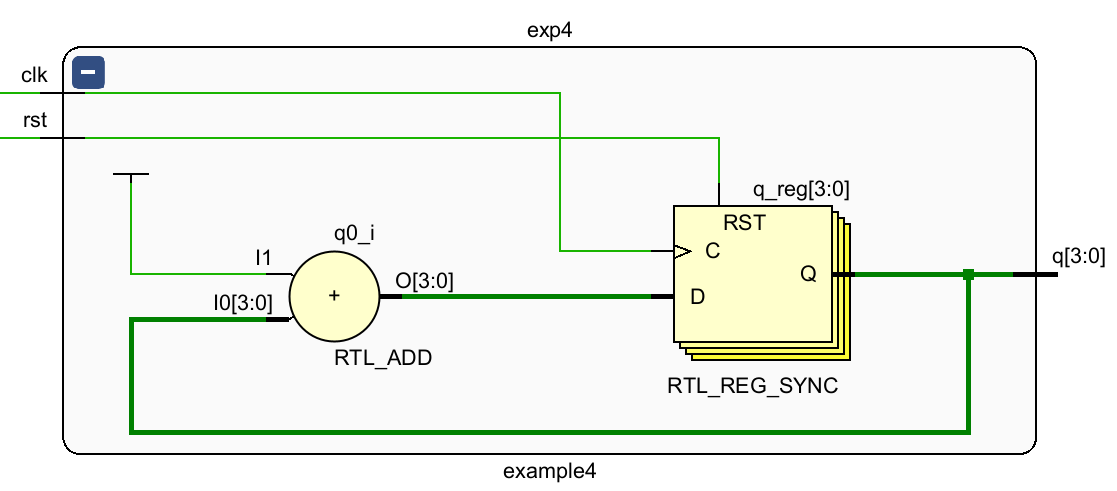
变量 q 这个时候被综合成一个寄存器，而不是普通的导线。这是一个时序逻辑单元，它应该被综合成一个计数器，每当时钟的上升沿，q 自增一。这段代码在 RTL 综合出的结果如下：

[](https://soc.ustc.edu.cn/Digital/lab5/figs/rtl_reg_1.png)

如果我们引入同步复位信号，对应的代码变为



Verilog 可以自动识别，并将其连接到寄存器的复位端。

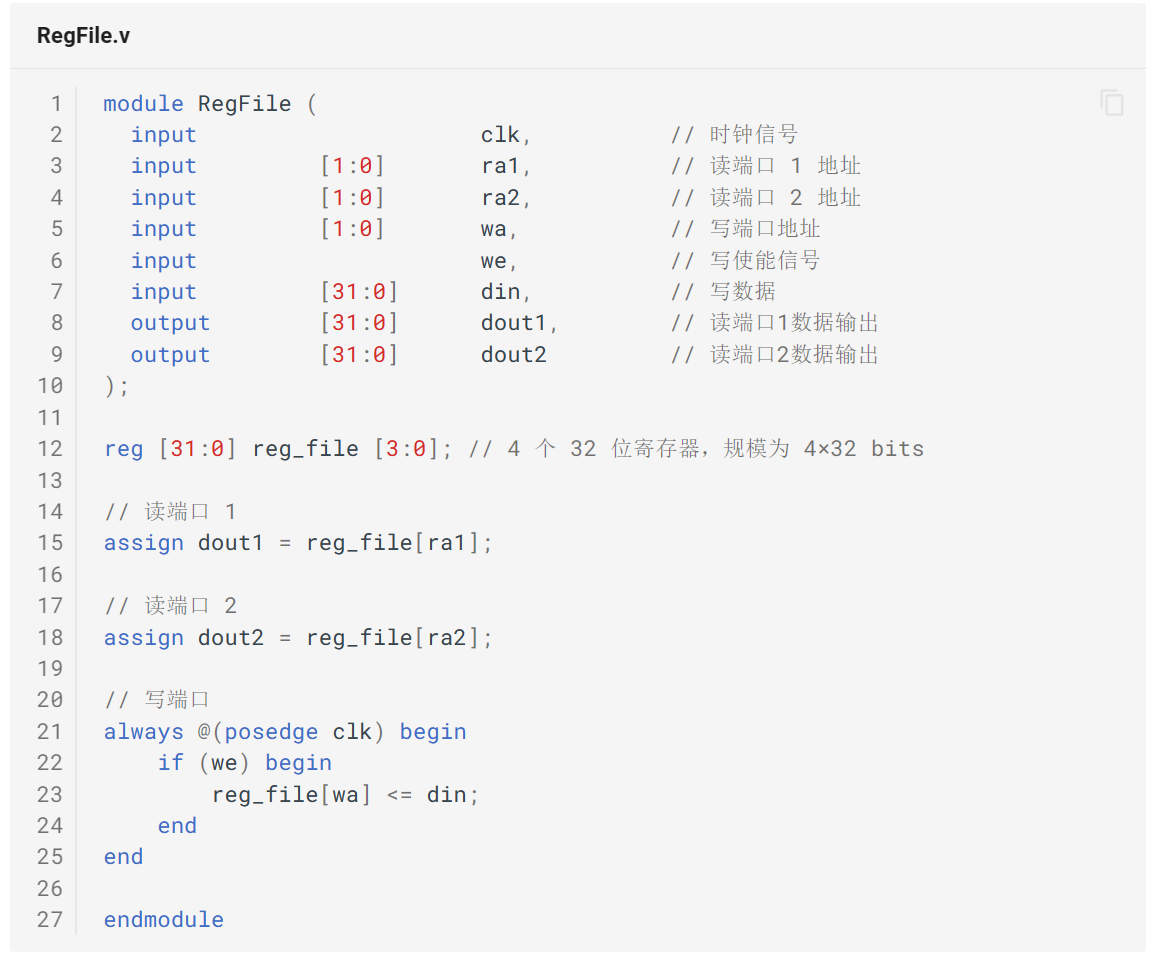
[](https://soc.ustc.edu.cn/Digital/lab5/figs/rtl_reg_2.png)

### ****2.2 寄存器堆****

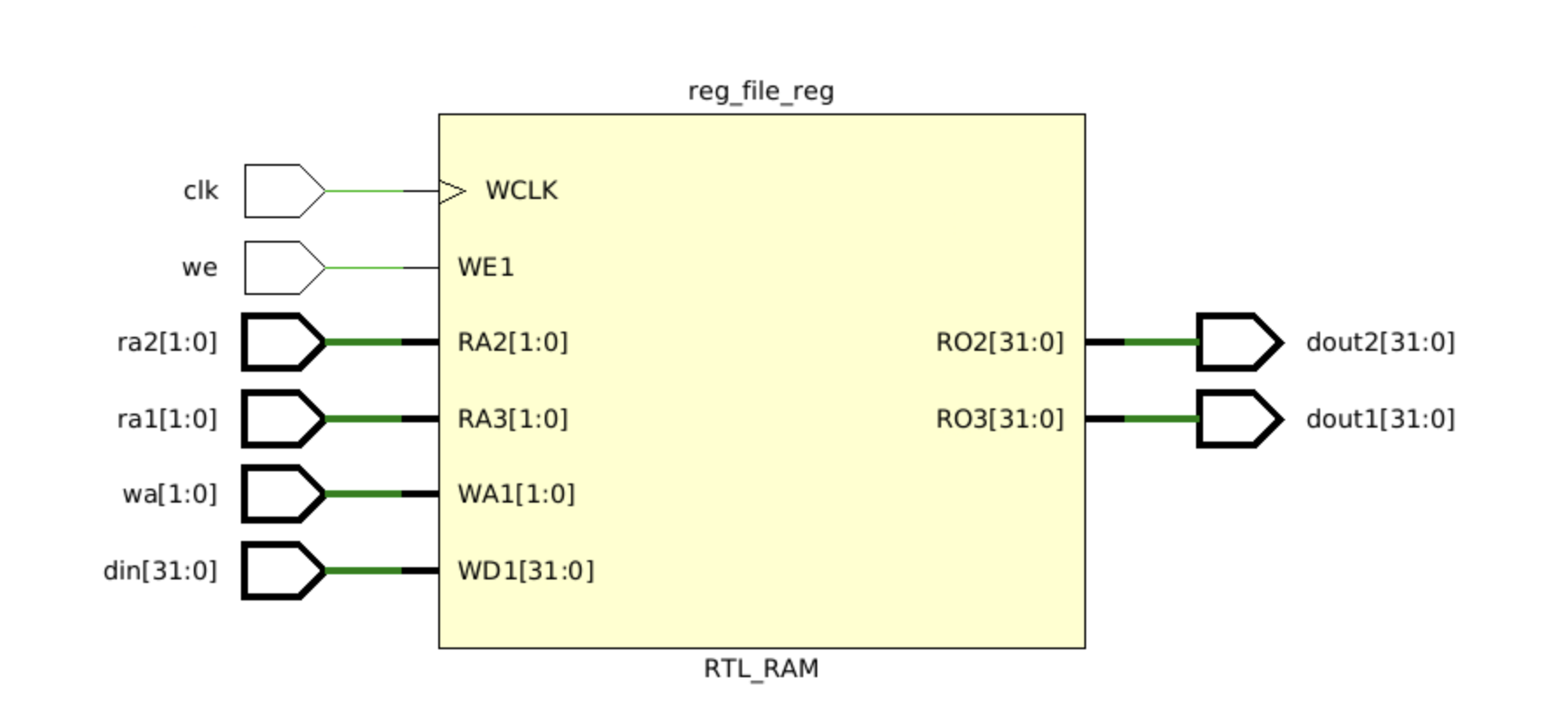
寄存器堆（Register File）是由多个可读写的寄存器组成的一种纯粹存储器件。

寄存器堆常用于 CPU 中作为计算机存储结构的最底层，它可以提高数据读取和写入的速度，从而加速 CPU 的运行效率。

以下是一个寄存器堆的 Verilog 代码实现：



从以下电路综合图可以看到，vivado 把我们的较为规范的寄存器堆直接综合成了 RAM（Random Access Memory），这也印证了寄存器堆的重要性。

[](https://soc.ustc.edu.cn/Digital/lab5/figs/rtl_reg_file.png)

# 思考与练习

1. 请参照实验文档中对寄存器堆的描述，设计并编写一个寄存器堆，要求满足以下功能：

1. 寄存器堆的规模为 32x32bits；
2. 0 号寄存器始终保持 0；
3. 当同时读写时，能读到正写入的最新数据。

编号低的功能优先级更高（3号情况要在满足2号情况下才能实现）。本题要求自行编写仿真文件进行测试（不需要很复杂的仿真，能凸显模块特点即可），代码框架如下：