设计寄存器和计数器

简介

多个触发器（flip-flops）在同一时钟下组合在一起，来保存相关信息的电路称为寄存器。就像触发器一样, 寄存器也可以有其它的控制信号。你将了解具有附加控制信号的寄存器的行为。计数器是广泛使用的时序电路。在本次实验中，你将用几种方法设计寄存器和计数器。 *请参考Vivado 教程上关于如何使用Vivado创建工程和验证电路。*

目标

完成本次实验后，你将有能力：

- 设计各种类型的寄存器
- 设计各种类型的计数器

寄存器 Part 1

在计算机系统中，相关信息常常在同时被存储。 **寄存器（register ）**以这样的方式存储信息比特，即系统可以在同一时间写入或读出所有的比特。寄存器的例子包含数据、地址、控制和状态。简单的寄存器数据的输入引脚和输出引脚分开，但它们用相同的时钟源。一个简单寄存器的设计如下。

```
module Register (input [3:0] D, input Clk, output reg [3:0] Q);
   always @(posedge Clk)
         Q <= D;
endmodule
```

注意到这和一个简单的有着多数据端口的D触发器类似。

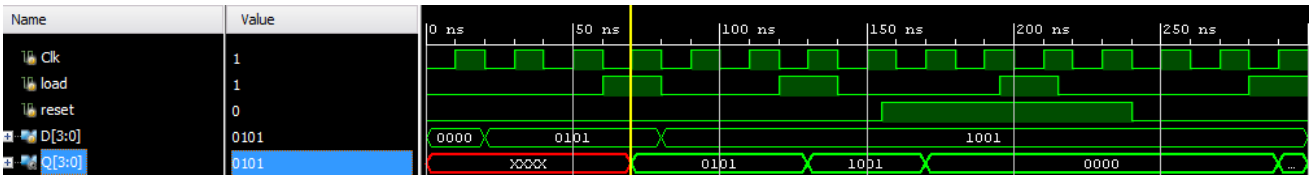
这个简单的寄存器会在每个时钟周期工作，保存需要的信息。然而，在有的情况下，需要只有在特定条件发生时，寄存器内容才被更新。比如，在计算机系统中的状态寄存器只在特定的指令执行时才更新。在这种情况下，寄存器的时钟需要用一个控制信号控制。这样的寄存器需要包含一个时钟使能引脚。下面是这种寄存器的设计。

```
module Register_with_synch_load_behavior(input [3:0] D, input Clk, input
load, output reg [3:0] Q);
   always @(posedge Clk)
      if (load)
         Q <= D;
endmodule
```

另一个希望寄存器包含的特性是，当特定条件发生时，寄存器会重置存储的内容。 下面是一个包含同步的（synchronous）重置和载入信号（重置的优先级高于载入）的简单寄存器的设计。

```
module Register_with_synch_reset_load_behavior(input [3:0] D, input Clk,
input reset, input load, output reg [3:0] Q);
   always @(posedge Clk)
      if (reset)
      begin
         Q <= 4'b0;
      end else if (load)
      begin
        Q <= D;
      end
endmodule
```

1-1. 用上面的例子，设计一个4-bit寄存器，包含同步的重置和载入信号。编写一个测试用例并对你的设计仿真。对Clk, D input, reset, load,和output Q赋值。在硬件中验证你的设计。



1-1-1. 打开Vivado，创建一个工程命名为*lab6_1_1*。

1-1-2. 创建Verilog module并添加包含同步重置和载入信号的4-bit寄存器。使用上面例子中提供的代码。

1-1-3. 编写一个测试用例，仿真**300ns**，并分析输出。

1-1-4. 添加开发板相对应的XDC文件，编辑XDC文件，加入相关的引脚，将 *Clk* 赋给 ***SW15***，*D* input 给 ***SW3-SW0***，*reset* 给 ***SW4***, *load* 给 ***SW5***，*Q* 给 ***LED3- LED0***。

1-1-5. 把下面这行代码加入XDC文件，使SW15 允许被当作时钟使用。

```
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets { clk }];
```

1-1-6. 综合你的设计。

1-1-7. 实现你的设计

查看Project Summary和Utilization table，注意到1个BUFG和 11 个IO被使用了。

1-1-8. 生成比特流文件，将其下载到Basys3或Nexys4 DDR开发板，并验证功能。

在一些仿真中，有必要把寄存器设置到一个预设值。对于这种情况，需要使用另一个控制信号，称为设置（set）。一般情况下，在这种寄存器中，重置信号会比设置信号拥有更高优先级，而设置信号比载入信号拥有更高优先级。

1-2. 设计一个4-bit包含同步重置，设置和载入信号的寄存器。分配好Clk, D input, reset, set, load和 output Q。在硬件中验证功能。

1-2-1. 打开Vivado，创建一个工程命名为*lab6_1_2*.

1-2-2. 创建Verilog module并添加包含同步重置，设置和载入信号的4-bit寄存器。

1-2-3. 添加开发板相对应的XDC文件，编辑XDC文件，加入相关的引脚。注意：你可能需要为你选择的拨码开关的Clk pin加入CLOCK_DEDITCATED_ROUTE 特性（property）。

1-2-4. 综合你的设计。

1-2-5. 实现你的设计。

查看 Project Summary并注意使用资源。理解输出的结果。

1-2-6. 生成比特流文件，将其下载到Basys3或Nexys4 DDR开发板，并验证功能。

上述的寄存器的分类是并行寄存器（parallel registers）。还有一种寄存器称为移位寄存器（shift registers）。移位寄存器是当控制信号生效时，存储的二进制数据会左移或右移的寄存器。移位寄存器可以被进一步分类为并行载入串行输出，串行载入并行输出和串行载入串行输出。它们可能有也可能没有重置信号。

在Xilinx FPGA，LUT可以被用作串行移位寄存器；如果代码写对的话，1个LUT就可以用作1bit输入1bit输出的SRL32，因而可以提供非常经济的设计（而不是串联32个触发器）。它可以有也可以没有使能信号。当使能信号生效时，内部存储的数据会移位1bit，1bit新的数据也会移入。这是一个简单的没有使能信号的1bit串行移入移出寄存器的设计。在这个设计中移入的bit需要经过32个时钟周期移出。这个设计可以用来实现一个延迟线。

```
module simple_one_bit_serial_shift_register_behavior(input Clk, input
ShiftIn, output ShiftOut);
    reg [31:0] shift_reg;

    always @(posedge Clk)
          shift_reg <= {shift_reg[30:0], ShiftIn};
    assign ShiftOut = shift_reg[31];

endmodule
```

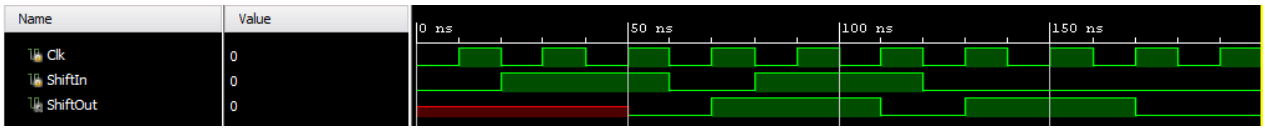
如果我们要实现一个少于32个时钟周期的延迟线，可以修改上面这个设计。下面是一个3时钟周期的延迟线。

```
module delay_line3_behavior(input Clk, input ShiftIn, output ShiftOut);
    reg [2:0] shift_reg;

    always @(posedge Clk)
          shift_reg <= {shift_reg[1:0], ShiftIn};
    assign ShiftOut = shift_reg[2];

endmodule
```

1-3. 使用上面的代码，设计一个1-bit的延迟线移位寄存器。编写一个测试用例并使用下图的信号刺激来对你的设计仿真。分配Clk，ShiftIn和output ShiftOut。在硬件上验证功能。



1-3-1. 打开Vivado，创建一个工程命名为*lab6_1_3*.

1-3-2. 创建Verilog module，使用上面的代码，添加1-bit延迟线移位寄存器。

1-3-3. 编写一个测试用例，仿真**200ns**.

1-3-4. 添加开发板相对应的XDC文件，编辑XDC文件，加入相关的引脚。注意：你可能需要为你选择的拨码开关的Clk pin加入CLOCK_DEDITCATED_ROUTE 特性（property）。

1-3-5. 综合你的设计。

1-3-6. 实现你的设计。

查看 Project Summary并注意使用资源。理解输出的结果。

1-3-7. 生成比特流文件，将其下载到Basys3或Nexys4 DDR开发板，并验证功能。

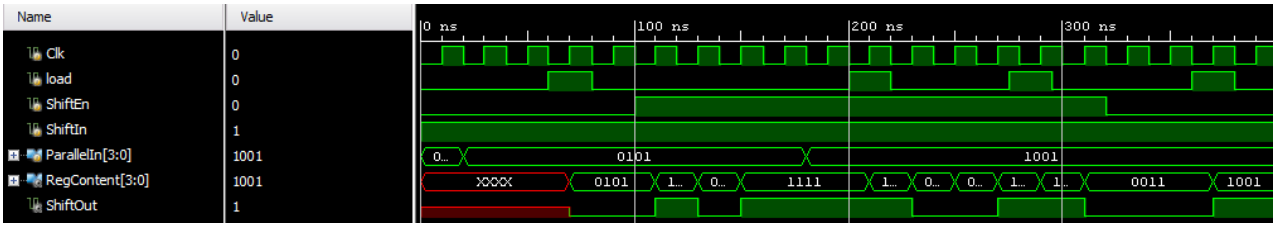
下面的代码设计了一个4-bit并行输入，带有载入和移位使能信号的向左移位寄存器 。

```
module Parallel_in_serial_out_load_enable_behavior(input Clk, input ShiftIn,
input [3:0] ParallelIn, input load, input ShiftEn, output ShiftOut, output
[3:0] RegContent);
    reg [3:0] shift_reg;

always @(posedge Clk)
       if(load)
          shift_reg <= ParallelIn;
       else if (ShiftEn)
          shift_reg <= {shift_reg[2:0], ShiftIn};
    assign ShiftOut = shift_reg[3];
    assign RegContent = shift_reg;

endmodule
```

1-4. 使用上面的代码，设计一个4-bit并行输入的向左移位寄存器。使用下图的信号刺激，编写一个测试用例并对你的设计仿真。分配Clk, ParallelIn, load, ShiftEn, ShiftIn, RegContent和ShiftOut.。在硬件上验证功能。



1-4-1. 打开Vivado，创建一个工程命名为*lab6_1_4*.

1-4-2. 创建Verilog module，使用上面的代码，添加4-bit并行输入的向左移位寄存器。

1-4-3. 编写一个测试用例，仿真**400ns**。

1-4-4. 添加开发板相对应的XDC文件，编辑XDC文件，加入相关的引脚。注意：你可能需要为你选择的拨码开关的Clk pin加入CLOCK_DEDITCATED_ROUTE 特性（property）。

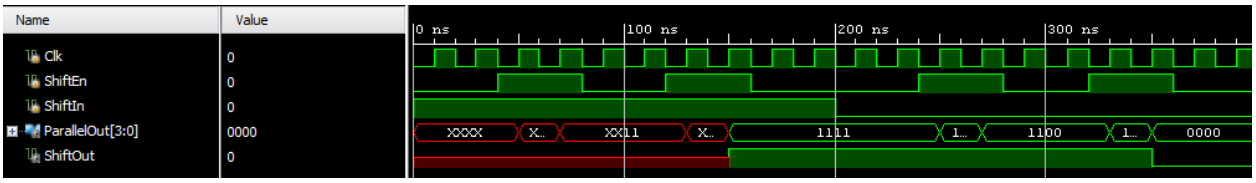
1-4-5. 综合你的设计。

1-4-6. 实现你的设计。

查看 Project Summary并注意使用资源。理解输出的结果。

1-4-7. 生成比特流文件，将其下载到Basys3或Nexys4 DDR开发板，并验证功能。

1-5. 设计一个4-bit串行输入并行输出移位寄存器。 编写测试用例并对设计仿真。分配Clk, ShiftEn, ShiftIn, ParallelOut和ShiftOut。在硬件上验证功能。



1-5-1. 打开Vivado，创建一个工程命名为*lab6_1_5*.

1-5-2. 创建Verilog module，添加4-bit串行输入并行输出移位寄存器的设计。

1-5-3. 编写一个测试用例，仿真**400ns.**

1-5-4. 添加开发板相对应的XDC文件，编辑XDC文件，加入相关的引脚。注意：你可能需要为你选择的拨码开关的Clk pin加入CLOCK_DEDITCATED_ROUTE 特性（property）。

1-5-5. 综合你的设计。

1-5-6. 实现你的设计。

查看 Project Summary并注意使用资源。理解输出的结果。

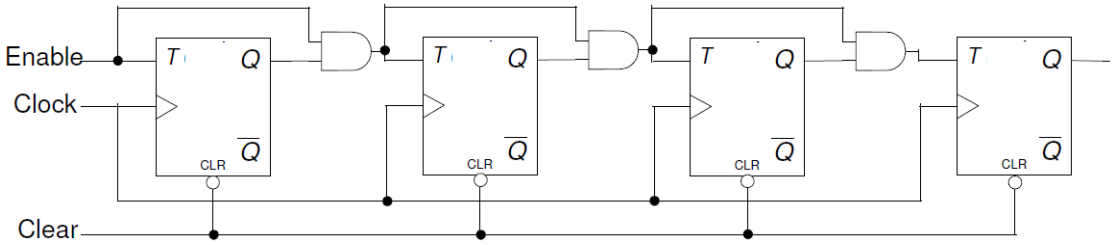
1-5-7. 生成比特流文件，将其下载到Basys3或Nexys4 DDR开发板，并验证功能。

计数器 **Part 2**

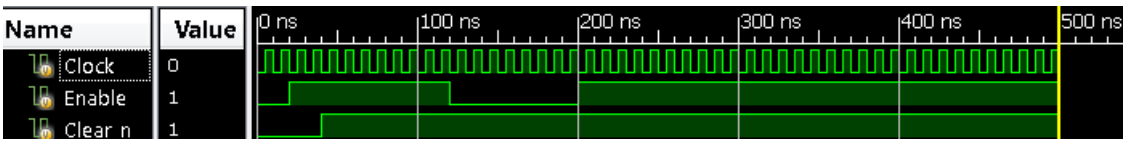
计数器可以是异步的也可以是同步的。异步计数器只使用事件信号来对事件计数。而同步计数器，使用共有的时钟信号，因而当多个触发器必须改变状态时，状态的改变同时发生。

二进制计数器是一种简单的计数器，当使能信号生效时会计数，当重置信号生效时会重置。当然，重置信号比使能信号的优先级高。

下面的电路图展示了这样的计数器。需要注意的是，清除信号是异步低电平生效的，而使能信号是同步高电平生效的。



2-1. 用T触发器（T flip-flops）设计一个8-bit计数器，将上面的结构扩展至8-bits。你的设计应该是分层级的，用行为级建模风格设计T触发器，其余部分用数据流级或门级建模风格。
编写一个测试用例并验证设计。分配Clock input, Clear_n, Enable和 Q。实现设计并在硬件上验证功能。



2-1-1. 打开Vivado，创建一个工程命名为*lab6_2_1*.

2-1-2. 创建并添加Verilog module以提供所需的功能。

2-1-3. 编写一个测试用例，并验证设计。

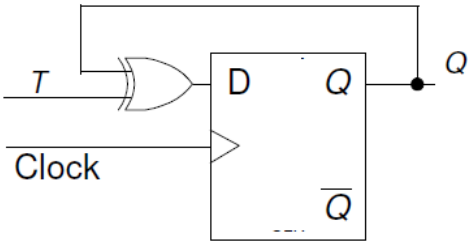
2-1-4. 添加开发板相对应的XDC文件，编辑XDC文件，加入相关的引脚。注意：你可能需要为你选择的拨码开关的Clk pin加入CLOCK_DEDITCATED_ROUTE 特性（property）。

2-1-5. 综合你的设计并在Synthesized Design查看原理图（schematic）。
指出用了什么资源，用了多少。

2-1-6. 实现你的设计。

2-1-7. 生成比特流文件，将其下载到Basys3或Nexys4 DDR开发板，并验证功能。

2-1-8. 计数器也可以用D触发器实现，因为T触发器可以用D触发器构造，如下图。



2-2. 用D触发器实现8-bit计数器。 你的设计应该是分层级的，用行为级建模风格定义D触发器，用数据流级建模风格从D触发器设计T触发器，实现其他的功能。分配Clock input, Clear_n, Enable和Q。实现设计并在硬件上验证功能。

2-2-1. 打开Vivado，创建一个工程命名为 *lab6_2_2*.

2-2-2. 创建并添加Verilog module以提供所需的功能。

2-2-3. 添加开发板相对应的XDC文件，编辑XDC文件，加入相关的引脚。注意：你可能需要为你选择的拨码开关的Clk pin加入CLOCK_DEDITCATED_ROUTE 特性（property）。

2-2-4. 综合你的设计并在Synthesized Design查看原理图（schematic）。

2-2-5. 实现你的设计。

2-2-6. 生成比特流文件，将其下载到Basys3或Nexys4 DDR开发板，并验证功能。

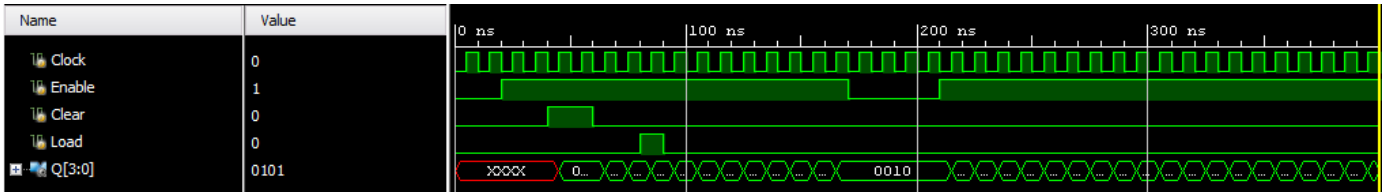
其他类型的二进制计数器包括(i) 上，(ii) 下和 (iii) 上下。它们可能都有计数使能和重置作为控制信号。有些情况下，你可能要求计数器从非0的地方开始计数增或计数减，要求计数器到达指定值时停下。这是一个4-bit计数器的例子，它从10 开始计数递减到 0。当计数值到0时，它会重新初始化到10。在任何时候，如果使能信号是低电平，计数器会暂停计数，直到使能信号恢复。假定在计数开始前，载入信号就生效来载入预设值。

```
reg [3:0] count;
wire cnt_done;

assign cnt_done = ~| count;
assign Q = count;

always @(posedge Clock)
  if (Clear)
     count <= 0;
  else if (Enable)
     if (Load | cnt_done)
        count <= 4'b1010;   // decimal 10
     else
        count <= count - 1;
```

2-3. 使用上面的代码，设计一个4-bit递减计数器，包含同步载入、使能和清除。编写测试用例（和下面波形图类似的）并验证功能。分配Clock input, Clear, Enable, Load和Q。实现设计并在硬件上验证功能。



2-3-1. 打开Vivado，创建一个工程命名为*lab6_2_3*.

2-3-2. 创建并添加Verilog module以提供所需的功能。

2-3-3. 编写一个测试用例，并验证设计。

2-3-4. 添加开发板相对应的XDC文件，编辑XDC文件，加入相关的引脚。注意：你可能需要为你选择的拨码开关的Clk pin加入CLOCK_DEDITCATED_ROUTE 特性（property）。

2-3-5. 综合你的设计并在Synthesized Design查看原理图（schematic）。

2-3-6. 实现你的设计。

2-3-7. 生成比特流文件，将其下载到Basys3或Nexys4 DDR开发板，并验证功能。

总结

本次实验中，你学习了各种寄存器和计数器是如何工作的。你设计这些组件并验证了它们的功能。这些组件被广泛用于处理器系统设计中。