# **实验二 Vivado环境下时序逻辑电路的设计与仿真**

## 一、 实验目的

1． 理解触发器和计数器的概念，掌握这些时序器件的 Verilog HDL 语言程序设计的方法。

## 二、 实验内容

1. 触发器（Ｄ型）的设计与仿真 ；
2. 计数器（递增、递减）的设计与仿真 ；
3. Gray 编码和 One-hot 编码两种状态机(FSM)设计时序控制电路；
4. 采用两种不同的状态机(FSM)结构来设计时序控制电路后进行仿真。

## 三、 实验要求

1. 在 Vivado 环境下进行时序电路的设计；
2. 在 Vivado 环境下进行时序电路的仿真；

## 四、 实验步骤

1. **D 触发器的实现**

在各种复杂的数字电路中， 不但需要对输入信号进行算术运算和逻辑运算， 还经常需要将这些信号和运算结果保存起来。因此，需要使用具有记忆功能的基本逻辑单元，能够存储一位信号的基本单元电路就被称为触发器。根据电路结构形式和 控制方式的不同，可以将触发器分为 D 触发器、JK 触发器、T 触发器等等。这里只介绍常用的Ｄ型触发器，其他类型触发器请有兴趣的同学自己实现。

在数字电路中，D 触发器是最为简单也是最为常用的一种基本时序逻辑电路，它是构成数字电路系统的基础。大体可分为如下几类：基本的 D 触发器； 同步复位的 D 触发器；异步复位的 D 触发器；同步置位/复位的 D 触发器； 异步置位/复位的 D 触发器

下面先分别介绍各个 D 触发器的具体工作原理，然后再介绍具体操作步骤。

1. **基本的 D 触发器**

在数字电路中，一个基本的上升沿 D 触发器的逻辑电路符号如图 2.1 所示，其功能表如表 2.1 所示。

根据下面的电路符号和功能表不难看出，一个基本的 D 触发器的工作原理为：当时钟信号的上升沿到来时，输入端口 D 的数据将传递给输出端口 Q 和输出端口 Q。在此，输出端口 Q 和输出端口 Q 除了反相之外，其他特性都是相同的。

下面给出具体操作过程：

1. 利用向导，建立一个新项目，工程名为 exper3。
2. 新建一个 Verilog HDL 文件，并输入源程序：

module async\_rddf(clk, d,q,qb); input clk, d;

output q,qb;

reg q,qb;

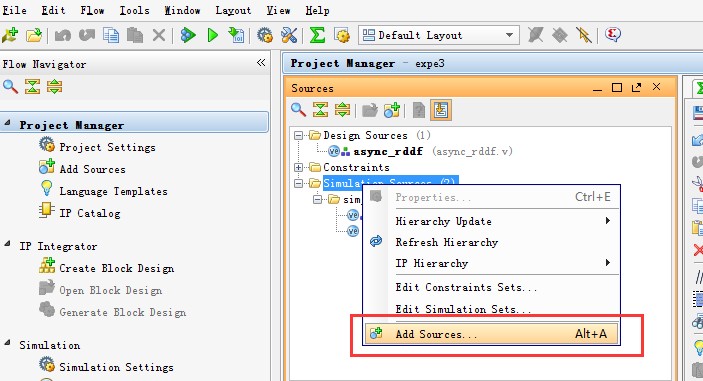
always@(posedge clk) begin q<=d;

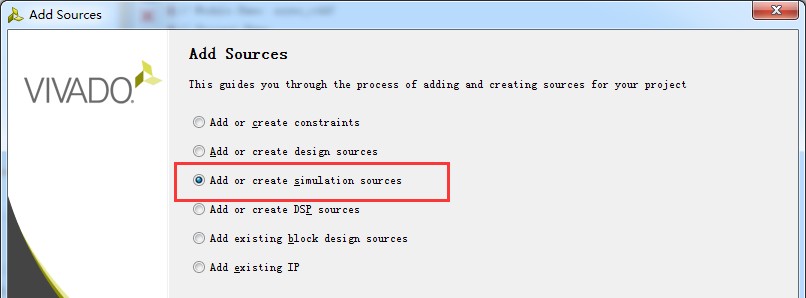
qb<=~d;

end endmodule

1. 对源程序进行语法检查和编译；
2. 进行时序仿真；

在 Simulation Source 上右击，在弹出的菜单中点击 Add Source，在弹出的对话框中选择 Add or create simulaton sources，如图 2.2 所示：





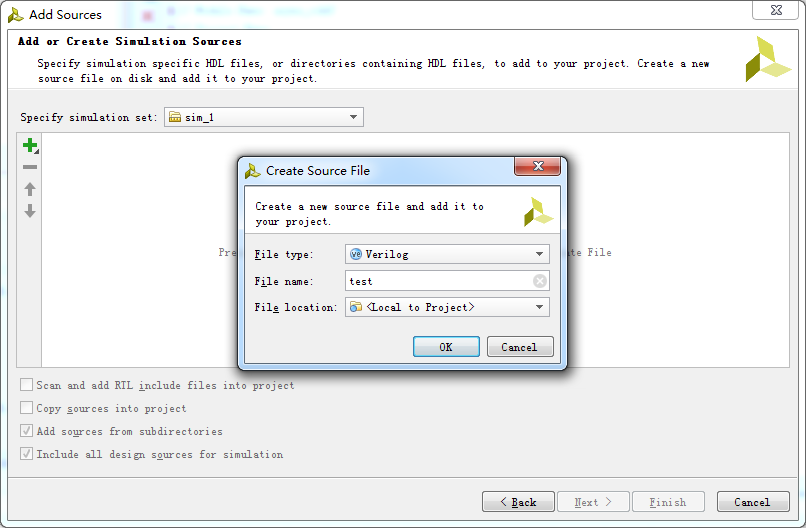


图 2.2 创建仿真源文件

创建完成后，输入仿真程序如下：

module test;

// Inputs reg clk; reg d;

// Outputs wire q; wire qb;

// 实例化控制电路uut;

async\_rddf uut (

.clk(clk),

.d(d),

.q(q),

.qb(qb)

);

initial begin

// Initialize Inputs clk = 0;

d = 0;

// Wait 100 ns

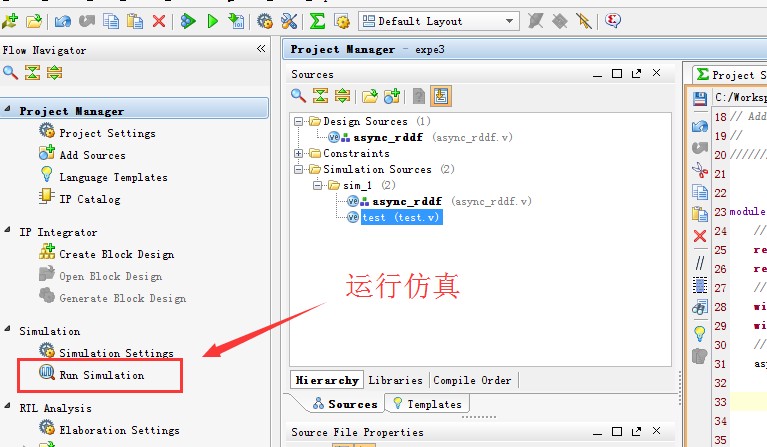
#100;

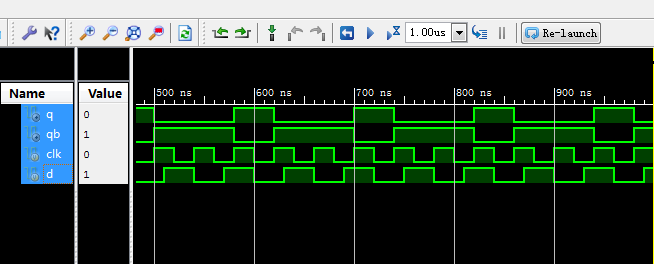
// Add stimulus here

end

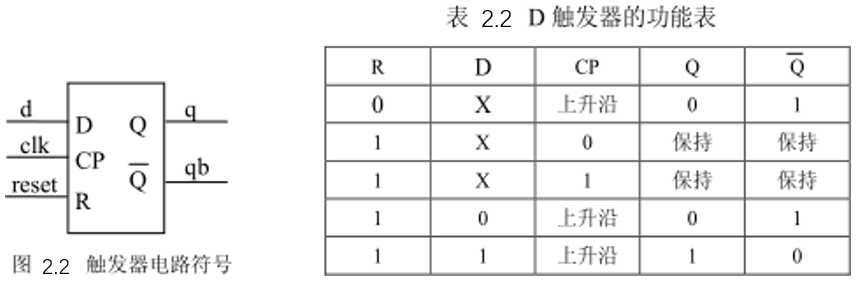
always #20 clk=~clk;

always #30 d=~d; endmodule

输入完成后点击左侧的 Run Simulation 进行仿真

仿真结果如下图所示（仅供参考）。

1. **同步复位的 D 触发器**



在数字电路中，一种常见的带有同步复位控制端口的上升沿 D 触发器的逻辑电路符号如图 2.3 所示，它的功能表如表 2.2 所示。不难看出，只有在时钟信号的上升沿到来并且复位控制端口的信号有效时，D 触发器才进行复位操作， 即将输出端口 Q 的值置为逻辑 0，而把输出端口 Q 的值置为逻辑 1。

源程序如下：

module sync\_rddf(clk,reset,d,q,qb); input clk,reset,d;

output q,qb; reg q,qb;

always @(posedge clk) begin if(!reset) begin

q<=0;

qb<=1; end

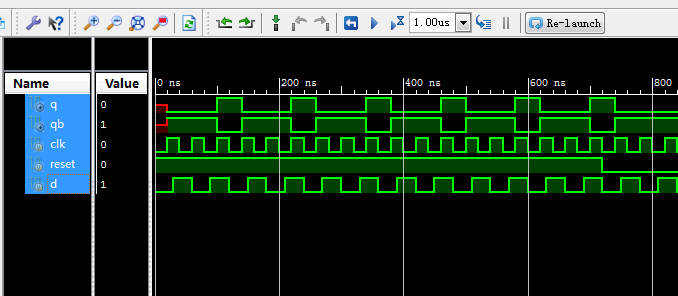
else begin

q<=d; qb<=~d;

end end

endmodule

仿真结果如下（仅供参考）：



仿真结果说明：

当复位信号 reset 为高时，同步复位 D 触发器与基本 D 触发器所实现的功能一致。

1. **异步复位的 D 触发器**

常见的带有异步复位控制端口的上升沿 D 触发器的逻辑电路符号如图 2.4 所示，它的功能表如表 2.3 所示。不难看出，只要复位控制端口的信号有效，D 触发器就会立即进行复位操作。可见，这时的复位操作是与时钟信号无关的。



源程序如下：

module async\_rddf(clk,reset,d,q,qb); input clk,reset,d;

output q,qb; reg q,qb;

always @(posedge clk or negedge reset) begin if(!reset) begin

q<=0;

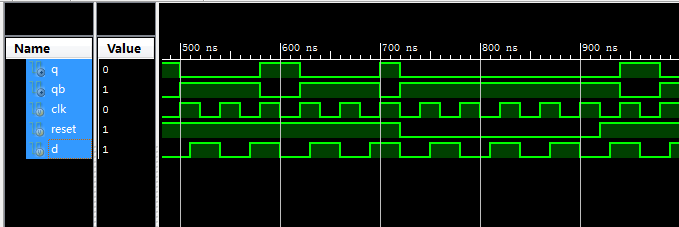
qb<=1; end

else begin q<=d; qb<=~d;

end end

endmodule

仿真结果如下（仅供参考）：



仿真结果说明：

观察同步复位 D 触发器与异步复位 D 触发器的仿真结果，其区别是显而易见的：如果不考虑器件本身的延迟，异步复位 D 触发器的 reset 信号为 0 时，输出信号 q 直接复位，不受时钟信号的影响。

1. **同步置位/复位的 D 触发器**

同时带有置位控制和复位控制端口的 D 触发器也是经常使用的，同样它也具有同步异步两种方式。这里我们给出同步置位/复位的 D 触发器的源程序及仿真结果，请读者根据已经介绍的内容自己实现异步置位/复位的 D 触发器。

带有同步置位/复位端口的上升沿 D 触发器的逻辑电路符号如图 2.5 所示， 它的功能表如表 2.4 所示。不难看出，只有在时钟信号的上升沿到来并且同步置位/复位端口的信号有效时，D 触发器才可以进行置位或者复位操作。



源程序如下：

module sync\_rsddf(clk,reset,set,d,q,qb); input clk,reset,set;

input d; output q,qb; reg q,qb;

always@(posedge clk) begin if(!set && reset) begin

q<=1;

qb<=0; end

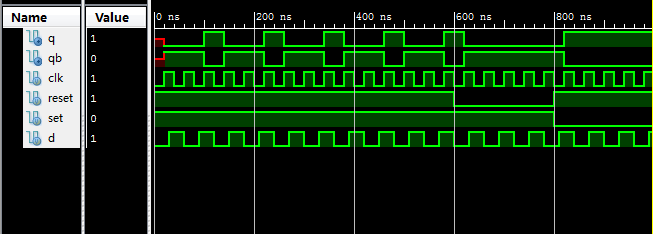
else if(set && !reset) begin q<=0;

qb<=1; end

else begin q<=d; qb<=~d;

end end

endmodule

仿真结果如下（仅供参考）：

1. **计数器的实现**
2. **加法计数器**

加法计数器的动作是， 每次时钟脉冲信号 clk 为上升沿时， 计数器会将计数值加 1。 以图 2.6 为例， 它是 2bits 的计数器，所以计数值（由 Q2Q1 组成） ，依次是 0，1，2，3，0，1…，周而复始。

在图 2.6 的波形图里，透露了这样几个信息：

1. 一个两 bit 计数器，它所能计数的范围是 0~3（22-1） 。同理，n bits 的计数器所能计数的范围是 0~2n -1。
2. 分别由 Q1、Q2 得到的波形频率是时钟脉冲信号 clk 的 1/2、1/4，亦即是将时钟脉冲信号的 clk 频率除 2、除 4。因此图 2.6 又常被称为除 4 计数器。
3. 由上讨论推广可知，n bits 计数器可获得的信号至多是频率除 2n 的结果。

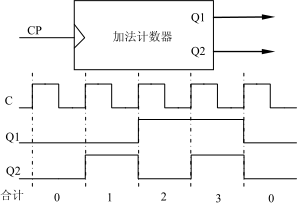


图 2.6、加法计数器的相关波形

源程序如下：

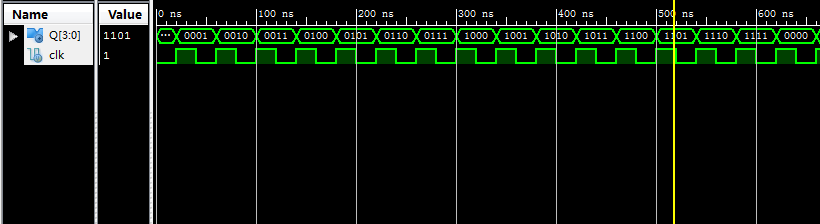
module addcounter(clk,Q) ;

input clk ; output[3 :0] Q ;

reg[3 :0] Q ;

always @(posedge clk)begin Q<=Q+1 ;

end endmodule

仿真结果如下（仅供参考）：

1. **减法计数器**

减法计数器的计数方式改成 15，14…0，因为仅是和加法计数器的计数方向不同，其它完全是一样的，因此，减法计数器的 Verilog HDL 语言描述，只需要将前面加法计数器的程序稍作修改即可。在此不再多说。

1. **有限状态机(FSM) 的实现**

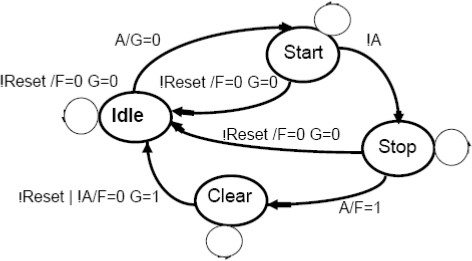
有限状态机是由寄存器组和组合逻辑构成的硬件时序电路，其状态（即由寄存器组的 1 和 0 的组合状态所构成的有限个状态）只可能在同一时钟跳变沿的情况下才能从一个状态转向另一个状态，究竟转向哪一状态还是留在原状态不但取决于各个输入值，还取决于当前所在状态。（这里指的是米里 Mealy 型有限状态机，而莫尔 Moore 型有限状态机究竟转向哪一状态只取决于当前状态。）

图2.7、用三种不同编码所实现的状态图

【例 1】 采用 Gray 编码的状态机源程序：

module ztj(clock,reset,a,g,f

);

input a,reset,clock; output g,f;

reg g,f; reg[1:0] state;

parameter start=2'b00,

stop =2'b01, clear=2'b10, idle =2'b11;

always@(posedge clock) begin

if(!reset)

begin state<=idle; f<=0; g<=0;

end

else

begin case(state)

start: if(!a)

state<=stop; else state<=start;

stop : if(a)

else

begin state<=clear; f<=1;

end

state<=stop;

clear: if(!a)

else

begin state<=idle; f<=0; g<=1;

end

state<=clear;

idle : if(a)

begin

end

end

endcase

else

state<=start; g<=0;

end state<=idle;

endmodule

【例 2】采用 One-hot 编码的状态机源程序：

module fsm (Clock, Reset, A, F, G); input Clock, Reset, A; output F,G; reg F,G;

reg [3:0] state ; parameter Idle = 4'b1000, Start = 4'b0100,

Stop = 4'b0010, Clear = 4'b0001;

always @(posedge Clock) begin

if (!Reset) begin

state <= Idle; F<=0; G<=0; end

else begin

case (state)

Idle: begin

if (A) begin state <= Start; G<=0;

end

else state <= Idle;

end

Start: begin

if (!A) state <= Stop; else state <= Start;

end

Stop: begin

if (A) begin

state <= Clear; F <= 1;

end

else state <= Stop; end

Clear: begin

if (!A) begin state <=Idle; F<=0; G<=1;

end

else state <= Clear; end

default: state <=Idle; endcase

end end

endmodule

例 1 中采用 Gray 编码，例 2 中采用的是 One-hot 编码。究竟采用哪一种编码好要看具体情况而定。对于用 FPGA 实现的有限状态机建议采用 One-hot 码，因为虽然采用 One-hot 编码多用了两个触发器，但所用组合电路可省下许多，因而使电路的速度和可靠性有显著提高，而总的单元数并无显著增加。采用了 One-hot 编码后有了多余的状态，就有一些不可到达的状态，为此在 CASE 语句的最后需要增加 default 分支项，以确保多余状态能回到 Idle 状态。

【例 3】利用状态机编写的流水灯源程序：

module led(clk,data,sw); input clk, sw; output[3:0] data;

reg clk1s;

parameter max=5000000; reg[1:0] state=2'b00; reg[30:0] n;

reg[3:0] data;

always @(posedge clk)begin if(n==max)begin

if(!clk1s) clk1s<=1'b1;

else clk1s<=1'b0; n<=0;

end

else n<=n+1; end

always @(posedge clk1s)begin

case(state) 2'b00:begin

state<=2'b01; if(sw)begin

data<=4'b1000;

end

else begin

data<=4'b0111;

end

end

2'b01:begin state<=2'b10; if(sw)begin

data<=4'b0100;

end

else begin

data<=4'b1011;

end end

2'b10:begin state<=2'b11; if(sw)begin

data<=4'b0010;

end

else begin

data<=4'b1101;

end end

2'b11:begin state<=2'b00; if(sw)begin

data<=4'b0001;

end

else begin

data<=4'b1110;

end end endcase

end endmodule

例三是采用 Gray 码编码的流水灯程序，通过拨码开关sw值的设置，来控制显示两种不同变化的流水灯。

## 五、 实验结果

**1.加法计数器**

（1）.给关键代码添加注释，或者给出模块代码的流程图。

（2）.编程与自己的学号有关联的测试程序（也称激励文件），在测试程序中关键代码后给出注释。可以将延迟时间，对应于自己的学号，或者将学号\*2作为延迟时间，或者将学号的后6位作为延迟时间……

（3）.对电路进行仿真，根据仿真波形，验证电路功能是否正确。将仿真后获得的波形图截图保存后，放到实验报告中。

（4）.在vivado中，通过项目管理窗口中的schematic菜单，生成与verilog代码对应的电路原理图，将该电路原理图放到实验报告中。

**2.减法计数器**

（1）.编写减法计数器的verilog代码，给关键代码添加注释，或者给出模块代码的流程图。

（2）.编程与自己的学号有关联的测试程序（也称激励文件），在测试程序中关键代码后给出注释。可以将延迟时间，对应于自己的学号，或者将学号\*2作为延迟时间，或者将学号的后6位作为延迟时间……。

（3）.对电路进行仿真，根据仿真波形，验证电路功能是否正确。将仿真后获得的波形图截图保存后，放到实验报告中。

（4）.在vivado中，通过项目管理窗口中的schematic菜单，生成与verilog代码对应的电路原理图，将该电路原理图放到实验报告中。

**3.有限状态机设计时序控制电路**

三种编码方式的仿真波形类似下图（仅供参考）：

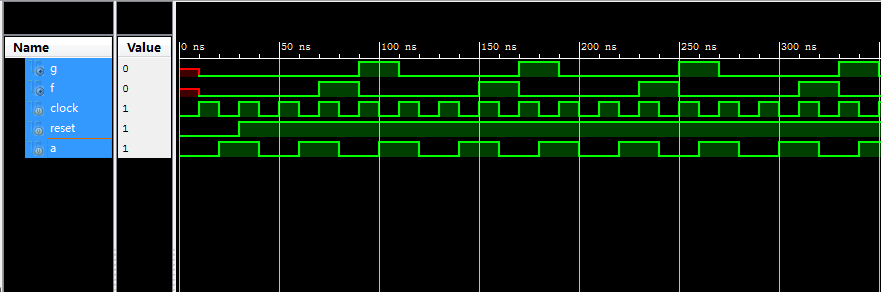


图 2.8 三种编码方式的仿真波形

流水灯的实验结果，通过拨码开关sw的选择可以显示两种方式的流水灯。

如果将上面例3的状态机代码，编译后生成的bit文件下载到EGO1开发板上，并且管脚分配表如下：（下载部分不需要完成）

|  |  |  |
| --- | --- | --- |
| 程序中管脚名 | 实际管脚 | 说明 |
| clk | P17 | clk时序电路时钟100MHZ |
| reset | P15 | 复位控制 |
| SW | P5 | 拨动开关 SW0 |
| DATA[0] | F6 | LED0 |
| DATA[1] | G4 | LED1 |
| DATA[2] | G3 | LED2 |
| DATA[3] | J4 | LED3 |

例3的实验结果如下（仅供参考）：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 拨动开关  sw0 值 | LED0 | LED1 | LED2 | LED3 |
| 1 | 从左至右逐次只亮一个灯 | | | |
| 0 | 从左至右逐次只灭一个灯 | | | |

要求：（1）.给verilog关键代码添加注释，或者给出模块代码的流程图。

（2）.编写与自己的学号有关联的测试程序（也称激励文件），所谓有关联，是指：在测试程序中关键代码后给出注释。可以将延迟时间，对应于自己的学号，或者将学号\*2作为延迟时间，或者将学号的后6位作为延迟时间……类似这样的，在程序中进行了相应注释就是满足要求的。

（3）.对电路进行仿真，根据仿真波形，验证电路功能是否正确。将仿真后获得的波形图截图保存后，放到实验报告中。

（4）.在vivado中，通过项目管理窗口中的schematic菜单，生成与verilog代码对应的电路原理图，将该电路原理图放到实验报告中。