第9章 Verilog HDL模型的不同抽象级别

概述

Verilog模型可以是实际电路不同级别的抽象。所谓不同的抽象级别，实际上是指同一个物理电路，可以在不同的层次上用Verilog语言来描述它，如果只从行为和功能的角度来描述某一电路模块，就称为行为模块；如果从电路结构的角度来描述该电路模块，就称为结构模块。抽象的级别和它们对应的模块类型常可以分为以下5种

(1) 系统级(system)
(2) 算法级(algorithmic)
(3) RTL级(RegisterTransferLevel):
(4) 门级(gate-level):
(5) 开关级(switch-level)

系统级、算法级和RTL级是属于行为级的，门级是属于结构级的。

对于数字系统的逻辑设计工程师而言，熟练地掌握门级、RTL级、算法级、系统级是非常重要的。而对于电路基本部件（如门、缓冲器、驱动器等）库的设计者而言，则需要掌握用户自定义源语元件（UDP）和开关级的描述。

一个复杂电路的完整Verilog HDL模型是由若干个Verilog HDL模块构成的，每一个模块又可以由若干个子模块构成。这些模块可以分别用不同抽象级别的Verilog HDL描述，在一个模块中也可以有多种级别的描述。利用Verilog HDL语言结构所提供的这种功能就可以构造一个模块间的清晰层次结构来描述极其复杂的大型设计。

9.1 门级结构

一个逻辑网络是由许多逻辑门和开关所组成，因此用逻辑门的模型来描述逻辑网络是最直观的。

Verilog HDL提供了一些门类型的关键字，可以用于门级结构建模。

9.1.1 与非门、或门和反向器及其说明语法

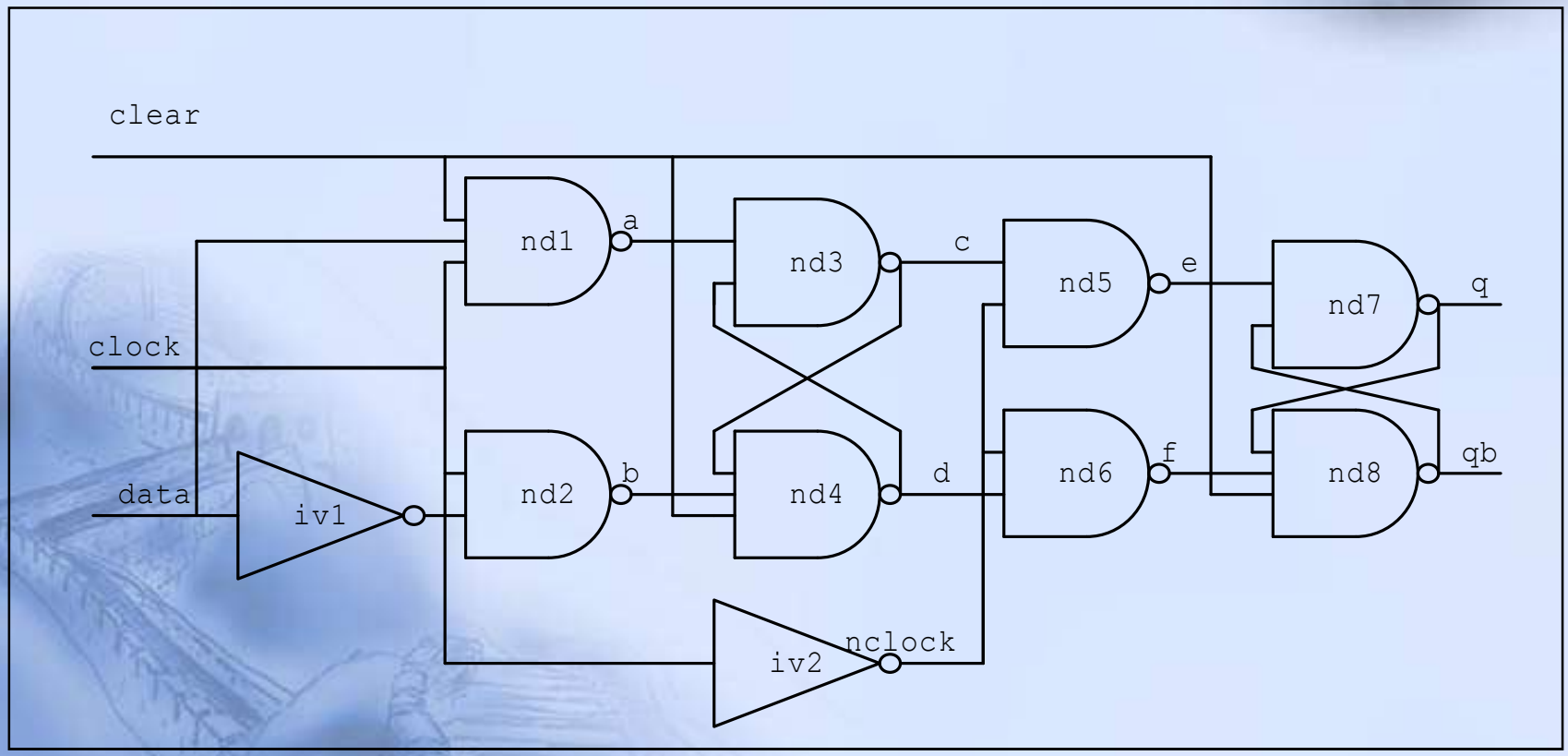
Verilog HDL中有关门类型的关键字共有26个之多。我们只介绍8个。

and 与门
nand 与非门
nor 或非门
or 或门
xor 异或门
xnor 异或非门
buf 缓冲器
not 非门

门声明语句的格式如下：
<门的类型>[<驱动能力><延时>]<门实例1>[，<门实例2>，…<门实例n>];

9.1.2用门级结构描述D触发器

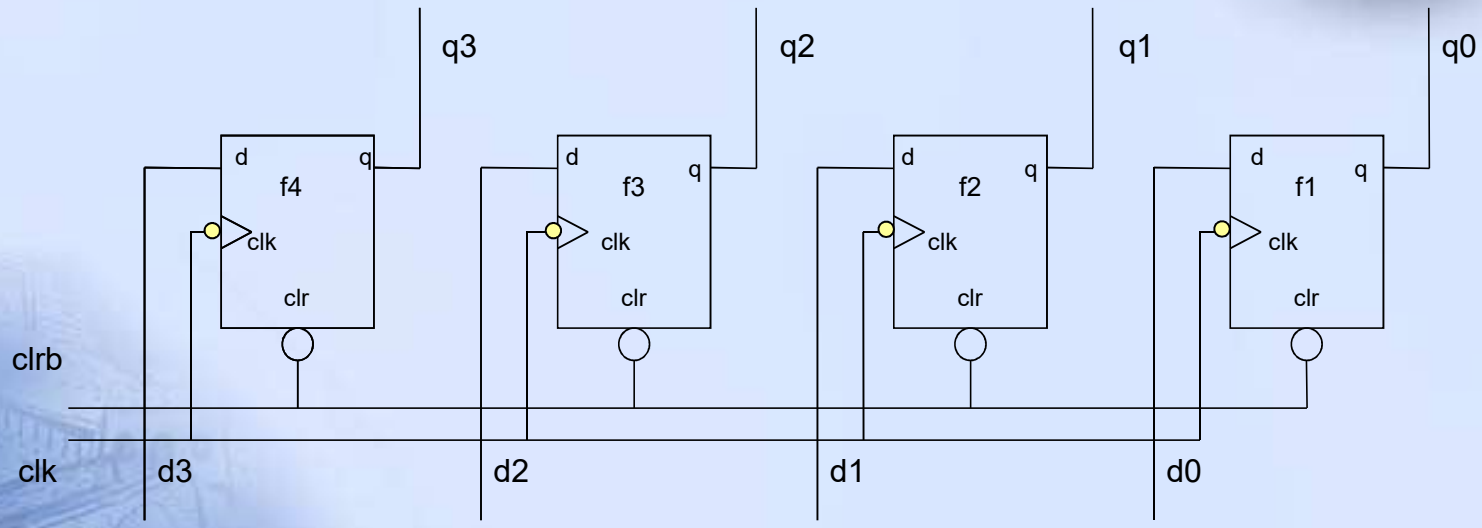
例9.1用基本逻辑单元组成D型主从触发器



```
module      flop(data,clock,clear,q,qb);
input       data,clock,clear;
output      q,qb;

nand  #10   nd1(a,data,clock,clear),
            nd2(b,ndata,clock),
            nd4(d,c,b,clear),
            nd5(e,c,nclock),
            nd6(f,d,nclock),
            nd8(qb,q,f,clear);
nand  #9    nd3(c,a,d),
            nd7(q,e,qb);
not   #10   iv1(ndata,data),
            iv2(nclock,clock);
endmodule
```

9.1.3 .由已经设计成的模块来构成更高一层的模块



四位寄存器电路结构图

```
`include "flop.v"
module      hardreg(d,clk,clrb,q);
input       clk,clrb;
input[3:0]        d;
output[3:0] q;

flop  f1(d[0],clk,clrb,q[0],),
      f2(d[1],clk,clrb,q[1],),
      f3(d[2],clk,clrb,q[2],),
      f4(d[3],clk,clrb,q[3],);
endmodule
```

9.2 Verilog HDL的行为描述建模

```
module hardreg(d,clk,clrb,q);
input        clk,clrb;
input[3:0]   d;
output[3:0] q;
reg [3:0] q;

    always @ (negedge clk or posedge clrb)
      begin
           if (clrb)
               q <= 0;
            else
               q <= d;
      end
endmodule
```

9.2.1 仅用于产生仿真测试信号的Verilog HDL行为描述建模

```
`include "flop.v"
`include "hardreg.v"
module  hardreg_top;
reg clock, clearb;
    //为产生测试用的时钟和清零信号需要寄存器
reg[3:0] data ;      //为产生测试用数据需要用寄存器
wire[3:0] qout;//为观察输出信号需要从模块实例端口中引出线

 `define  stim  #100 data=4'b //宏定义 stim,可使源程序简洁
  event  end_first_pass;   //定义事件end_first_pass

hardreg  reg_4bit
(.d(data), .clk(clock), .clrb(clearb), .q(qout));
```

```
initial
begin
 clock = 0;
 clearb = 1;
end

always #50 clock = ~clock;

always @(end_first_pass)
  clearb = ~clearb;

always @(negedge clock)
  $strobe("at time %0d clearb= %b data= %d qout= %d",
                        $time, clearb, data, qout);
```

```
initial
begin
  #55;
  repeat(4)          //重复四次产生下面的data变化
  begin
    data=4'b0000;
    `stim 0001;
    `stim 0010;
    `stim 0011;
    `stim 0100;
    `stim 0101;
    `stim 0110;
    `stim 0111;
    `stim 1000;
    `stim 1001;
```

```
`define  stim  #100 data=4'b
```

```
        `stim 1010;
        `stim 1011;
        `stim 1100;
        `stim 1101;

        `stim 1110;
        `stim 1111;
        #200-> end_first_pass;
    end

    $finish;              //结束仿真
end
endmodule
```

9.2.2 硬件描述语言的可综合性问题

所谓逻辑综合就其实质而言是设计流程中的一个阶段，在这一阶段中将较高级抽象层次的描述自动地转换成较低层次描述。就现在达到的水平而言，所谓逻辑综合就是通过综合器把HDL程序转换成标准的门级结构网表，而并非真实具体的电路。而真实具体的电路还需要利用ASIC和FPGA制造厂商的布局布线工具根据综合后生成的标准的门级结构网表来产生。为了能转换成标准的门级结构网表，HDL程序的编写必须符合特定综合器所要求的风格。由于门级结构、RTL级的HDL程序的综合是很成熟的技术，所有的综合器都支持这两个级别HDL程序的综合。

9.3 用户定义的原语

用户可以定义自己设计的基本逻辑元件的功能，也就是说，可以利用UDP来定义自己特色的用于仿真的基本逻辑元件模块并建立相应的原语库。这样，就可以与调用Verilog HDL基本逻辑元件同样的方法来调用原语库中相应的元件模块，并进行仿真。由于UDP是用查表的方法来确定其输出的，用仿真器进行仿真时，对它的处理速度较对一般用户编写的模块快得多。与一般的用户模块比较，UDP更为基本，它只能描述简单的能用真值表表示的组合或时序逻辑。

```
primitive udp_and(out,a,b); //原语名和端口列表

//端口声明语句
output out;//表示组合逻辑时一定不能声明为reg类型
input a,b; //输入端口声明

//状态表定义；以关键词table开始
table
  //状态表输入项的次序必须与输入端口列表一致
  // a     b       :      out;
     0     0       :      0;
     0     1       :      0;
     1     0       :      0;
     1     1       :      1;
endtable  //状态表定义结束

endprimitive //自定义原语udp_and的定义结束
```