目 录

内 容

- 第一节 硬件描述语言
- 第二节 Verilog HDL设计入门
- 第三节 Verilog基础知识
- 第四节 行为描述
- 第五节 数据流描述
- 第六节 结构描述
- 第七节 用户自定义元件
- 第八节 其他论题
- 第九节 仿真
- 第十节 实例分析

第五节 数据流描述

5.1 连续赋值语句

5.2 时延

5.3 线网时延

5.4 举例

第五节 数据流描述

5.1 连续赋值语句

5.2 连续赋值时延

5.3 线网时延

5.4 举例

数据流描述

- 在数字电路中，信号经过组合逻辑时会类似于数据流动，即信号从输入流向输出，并不会在其中存储。当输入发生变化时，总会在一定时间以后体现在输出端。同样，可以模拟数字电路这一特性，对其进行建模，这种建模方式通常称为数据流描述。数据流描述中最基本的语句是assign连续赋值语句。

连续赋值语句

- 在Verilog HDL中有两种赋值方式：连续赋值和过程赋值。连续赋值用于数据流行为建模；过程赋值用于顺序行为建模。
- assign连续赋值语句专门针对连线进行赋值，考虑了连线驱动强度与延时时间后，完整的连续赋值语句形式为：

  assign *（strength0，sthength1）#（delay）* **赋值表达式**

  如：assign c=a&b;
- 只要表达式的操作数有事件发生，即只要有数值上的变化，就会对表达式进行计算；如果计算的结果也发生变化，就会将新的数值赋给左边的线型变量 。

连续赋值与过程赋值

- 赋值对象不同

  连续赋值语句用于对线型变量赋值；过程赋值语句完成对寄存器变量赋值。

- 赋值过程实现方式不同

  线型变量一旦被连续赋值语句赋值后，赋值语句右端表达式中的信号有任何变化，都将随时反映到左端的线型变量中；过程赋值语句只有在语句执行到时，赋值过程才进行一次，且赋值过程的具体执行时刻还受到定时控制和延时模式等多方面的影响。

- 语句出现的位置不同

  连续赋值语句不能出现在任何一个过程块中；过程赋值语句只能出现在过程块中。

- 语句结构不同

  连续赋值语句以关键词assign为先导，语句中的赋值算符只有阻塞型一种形式；过程赋值语句不需要相应的先导关键词，语句中的赋值算符分阻塞型和非阻塞型两种。

- 冲突处理方式不同

  一条连线可被多条连续赋值语句同时驱动，最后的结果依据连线类型的不同有相应的冲突处理方式；寄存器变量在同一时刻只允许一条过程赋值语句对其进行赋值。

第五节 数据流描述

5.1 连续赋值语句

5.2 连续赋值时延

5.3 线网时延

5.4 举例

连续赋值时延

- 门级时延表达的是从门的输入端发生变化到输出端发生变化的门传输延时。
- 连线时延则直接体现了任何被驱动信号在连线上的传输延时。
- **assign**语句的时延表达了赋值表达式右端的变化反映到左端连线上的信号发生变化的延时时间。

连续赋值时延（Cont'd）

- 上升延时（输出变为1）
- 下降延时（输出变为0）
- 关闭延时（输出变为z，高阻态）
- 输出变成x的延时

说明：

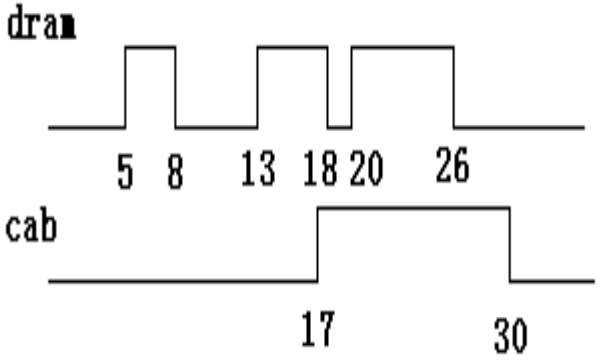
- 缺省取最小值

- 在一些电路模型中，延时分为最大、典型和最小3种情况，可以采用min：typ：max的格式来表示

连续赋值时延说明

- **assign** w = shreck & tech;
- **assign** # 4 cab = dram;
- **assign** # (4, 8, 6) ask = quick;
- **assign** # (4:5:6, 3:4:5) A_xor_wire = eq0 ^ eq1;

- 如果在例中出现以下情况：在右端的值传输到左端之前，右端的值又发生变化，则其值的变化在时延间隔中将被虑掉，即这种变化无法传输到左端。



第五节 数据流描述

5.1 连续赋值语句

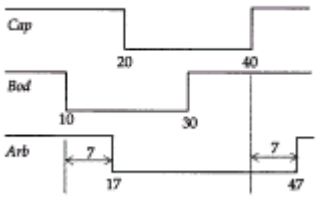
5.2 连续赋值时延

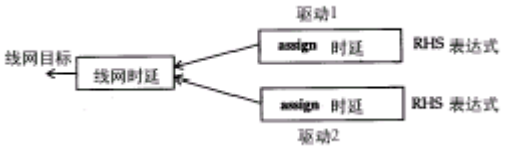
5.3 线网时延

5.4 举例

wire #5 arb; **assign** #2 arb = bod & cap;

- 线网信号本身的时延为5个单位，加上连续赋值的延迟，总的延时是7个时间单位。也就说假设在某一时刻t，Bod上事件的发生使得右端表达式的值发生改变，因此在(t+7)时刻，arb上的值才发生改变。
- 如果时延在线网说明赋值中出现，那么时延不是线网时延，而是赋值时延。下面例子中线网说明赋值，2个时间单位是赋值时延，而非线网时延。





wire ＃2 arb=bod∩

第五节 数据流描述

5.1 连续赋值语句

5.2 连续赋值时延

5.3 线网时延

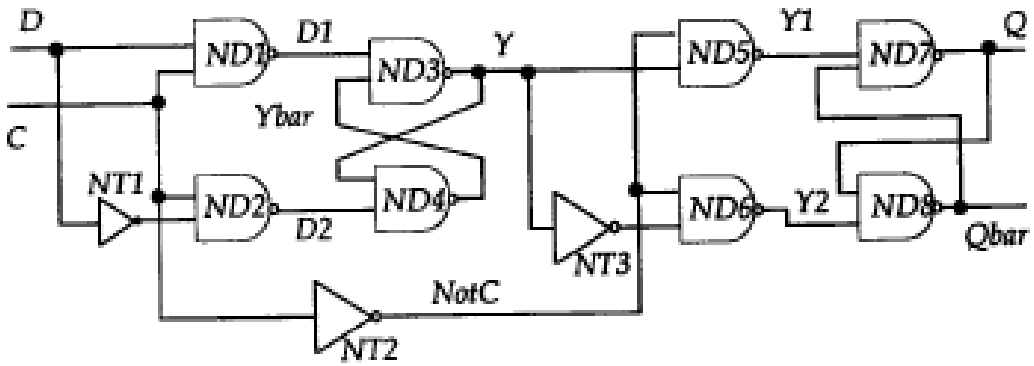
5.4 举例

数值比较器的数据流描述

```
module MagnitudeComparator (A，B，AgtB，AeqB，AltB ) ;
  parameter BUS = 8;
  parameter EQ_DELAY = 5，LT_DELAY = 8，GT_DELAY = 8;
  input  [1 : BUS]A，B;
  output AgtB，AeqB，AltB ;
  assign #EQ_DELAY AeqB = A = = B;
  assign #GT_DELAY AgtB= A > B;
  assign #LT_DELAY AltB= A < B;
endmodule
```

主从触发器的数据流描述

```
module MSDFF_DF(D，C，Q，Qbar) ;
  input  D，C;
  output Q，Qbar;
  wire NotC，NotD，NotY，Y，D1，D2，Ybar，Y1，Y2 ;
  assign Not D = ~ D;
  assign Not C = ~ C;
  assign Not Y = ~ Y;
  assign D1 = ~ (D & C) ;
  assign D2 = ~ (C & NotD) ;
  assign Y = ~ (D1 & Ybar ) ;
  assign Ybar = ~ (Y & D2) ;
  assign Y1 = ~ (Y & NotC ) ;
  assign Y2 = ~ (NotY & NotC) ;
  assign Q = ~ (Qbar & Y1) ;
  assign Qbar = ~ (Y2 & Q) ;
endmodule
```



小 结

- 连续赋值语句是连续驱动的，只要输入发生变化，都会导致该语句的重新计算。
- 使用assign对组合逻辑建模，由于assign语句以连线作为操作对象，它的许多细节与连线的性质有关，只有线型变量才能在assign语句中被赋值。
- assign语句本身构成一个独立的并行运行进程，与行为语句块（always和initial）、其他连续赋值语句、门级模型之间是并行的。
- 事实上，连续赋值语句作为一种细化到对模块内部具体连线描述的语句，有一定的结构描述的含义。但它又没有给出赋值语句本身实现的内部结构形式，应当属于行为描述的范畴。
- 它是一种介于行为描述与结构描述之间的一种中间描述。这充分体现了Verilog HDL可以对各个抽象层次进行描述的语言特点。

内 容

- 第一节 硬件描述语言
- 第二节 Verilog HDL设计入门
- 第三节 Verilog基础知识
- 第四节 行为描述
- 第五节 数据流描述
- 第六节 结构描述
- 第七节 用户自定义元件
- 第八节 其他论题
- 第九节 仿真
- 第十节 实例分析

第六节 结构描述

6.1 模块

6.2 模块调用

- 6.2.1 模块调用的基本方式
- 6.2.2 悬空端口
- 6.2.3 不同的端口长度
- 6.2.4 模块参数值

6.3 外部端口

6.4 结构描述方法

6.5 举例

第六节 结构描述

模 块

模块的定义形式如下：

module *模块名(端口列表)；*

端口类型说明；

数据类型说明；

描述体；

endmodule

- 通过Verilog HDL模块的调用，可以构成任何具有复杂结构的电路。而这种以结构方式建立的硬件模型可以进行仿真，也可以进行综合。

模块示例

【例6.1】二位加法器的模块

```
module  add2bit (in1，in2，sum)；//模块定义行
input  in1，in2；
output  [1:0] sum；                    // 端口类型说明
wire  in1，in2；
reg  [1:0] sum；                       //数据类型说明
always  @ (in1 or in2)
  begin
    sum = in1 + in2；
    $display (“The sum of %b and %b is %d (time = %d)”，      //描述体
    in1，in2，sum，$time)；
  end

endmodule //结束行
```

第六节 结构描述

第六节 结构描述

模块调用形式

模块调用的基本形式为：

模块名 调用名 (端口名表项)

- 由于描述的是具体的硬件逻辑，每个模块都表示一个具有特定功能的电路块。因此每当它被其它模块调用时，该模块内部被调用的电路块就被复制一次。
- 如果在当前模块中多次调用同一个模块，则需要用不同的调用名。
- Verilog HDL的模块调用和C语言的函数调用相似，也存在形参和实参的结合问题。

模块调用方式

- 模块的调用方式可大致分为两种：位置关联调用方式、端口名关联调用方式。

【例6.2】模块调用的例子。

```
module ha (out1，out2，in1，in2)；
input in1，in2；
output out1，out2；
……
endmodule
```

模块调用方式

- 模块调用采用位置关联调用方式，只需按序列出实例的端口名。模块实例语句如下：

  ha h1(P，Q，S，C)；

  在本语句中，ha是模块名，h1 是实例名称，并且端口按序关联。

- 模块调用采用端口名关联调用方式，则无需按序排列端口名，但实例的端口信号和被调用模块的端口信号必须一一列出。说明方式如下：

  .定义时端口名（调用时与之相连的信号名）

  模块实例语句如下：

  ha h2(.in2(C)，.in1(S)，.out1(P)，.out2(Q),)

  由于端口之间的对应关系十分清楚，因此端口名的排列顺序可随意改变。

第六节 结构描述

悬空端口

- 悬空端口即其端口表达式为空白的端口。模块实例语句如下：

  ha h3 (P，Q，，C)；

  ha h4 (.in2(C)，.in1(S)，.out1()，.out2(Q))；

- 在这两个实例语句中，端口S和out1悬空。若模块的输入端口悬空，则其值为高阻态z；而模块的输出端口悬空，则表示该输出端口废弃不用。

第六节 结构描述

不同的端口位长

- 当端口和局部端口表达式的长度不一致时，端口的匹配是通过以下两种方法来解决的：无符号数的右对齐或截断方式。

例如：

module Child(A,B) ;

intput [4:0] A;

output [2:0] B;

……

endmodule

module TOP;

wire [1:2] C;

wire [2:6] D;

Child CH1 (C,D);

endmodule

- 端口匹配情况如下，C[2]连接到A[0]，C[1]连接到A[1]，余下的输入端口均悬空,即其值为高阻态z。而D[6]连接到B[0]，D[5]连接到B[1]，D[4]连接到B[2]。

第六节 结构描述

模块参数化

- 设计采用参数化的设计方法，在模块中引入参数、用文字量来代替数字的表示方法，可以增加程序的可读性。
- 更重要的是大大增强了模块的可重用性。

参数化模块的调用

- 当某个模块被其它模块调用时，上层模块可以改变下层模块的参数值。改变的方法有两种：

  参数重定义语句和带参数值的模块调用。

例子：

```
module  HI(out1，out2，
                in1，in2)；
……
parameter  p1 =2；  p2 =3；
                p3 =5；
……
endmodule
```

参数重定义语句(defparam)

参数重定义语句的表达形式如下：

defparam *分级名 = 值;*

以下是采用参数重定义语句的例子。

```
module TOP1(N1，N2，M1，M2);
……
HI k1(out1，out2，in1，in2);
defparam k1.p1 =4；k1.p2 =2；k1.p3
    =6;
……
endmodule
```

- 在任意地方完成对任意特定模块参数的改写，比如在一个模块中调用某个参数化模块，而在另一个独立的模块中完成对它的参数重定义，则其分级名是从调用模块开始的全路径分级名。

带参数值的模块调用

- 带参数值的模块调用，即在调用的同时，完成对被调用模块参数的改变。

以下是采用带参数值的模块调用的例子。

```
module TOP2(N1，N2，M1，M2)；
……
HI ＃(4，2，6) k1(out1，out2，in1，in2)；
……
endmodule
```

第六节 结构描述

外部端口

下面是一个采用显式的方式来指定外部端口例子：

```
module  Scram2 (.Data(A)，.Con(C)，.Mem(M)，.Addr(B))；
input  [0:3] A；
input  C；
input [8:0] M；
output  [0:3] B；
……
endmodule
```

- 模块Scram2在本例中，指定了外部端口Data、Con、Mem和Addr。端口表显式地说明了外部端口和内部端口之间的连接与映射关系。外部端口无需声明，它在模块内不可见，但要在模块实例语句中使用；而内部端口必须声明，因为它在模块内可见。

外部端口

- 在模块实例语句中，外部端口的使用如下：

Scram2 S1(.Addr(A1)，.Data(D1)，.Con(C1)，.Mem(M1))；

- 如果模块端口通过位置连接，则模块实例语句中不能使用外部端口名称。
- 模块中若无内部端口，则模块在被调用时，其外部端口悬空，不与内部信号相连。例如，

module Scram5 (.Data()，.Con(C)，.Mem(M[0]，M[1])，.Addr())；
input C；
input [8:0] M；
……
endmodule

- 一个内部端口还可以与多个外部端口连接，例如，

module FanOut (.A(in)，.B(out)，.C(out))；
input in；
output out；
……
endmodule

第六节 结构描述

Verilog结构描述方法

- Verilog HDL的结构描述只是忠实地将图形方式的连接关系转变为相应的文字而已，如果已经完成了一个模块的逻辑电路图设计，那么按照下面给出的步骤，就可以很方便地转换成与之等价的Verilog HDL结构描述模块：

  1.给电路图中的每个输入输出引脚赋以端口名；

  2.给电路图中的每条内部连线信号取上各自的连线名；

  3.给电路图中的每个逻辑单元取一个单元名（即调用名）；

  4.给所要描述的电路模块确定一个模块名；

  5.用module定义相应模块名的结构描述，并将逻辑图中所有的输入输出端口名列入端口名表项中，再完成对各端口的输入输出类型说明；

  6.依照电路图中的连接关系，确定各单元之间端口信号的连接，完成对电路图内部的结构描述；

  7.最后用endmodule结束模块描述全过程。

Verilog结构描述方法（Cont’d）

- Verilog HDL内含的基本原语元件（Basic Primitives）有26种，其中14种为门级原语元件，12种为开关级原语元件。
- 门级原语元件包括：
  - and、nand、or、nor、xor、xnor
  - buf、not
  - bufif1、bufif0、notif1、notif0
  - pullup、pulldown

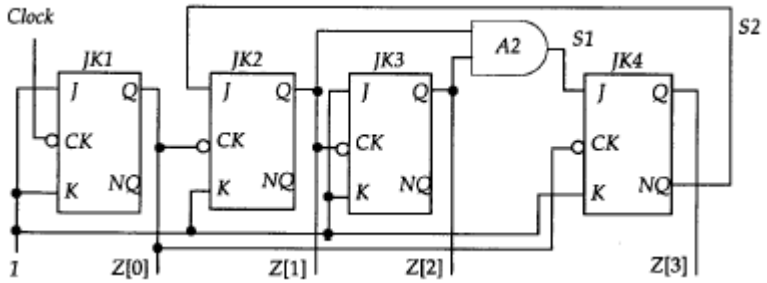
Verilog结构描述方法（Cont'd）

- 如果描述中用到的逻辑单元是Verilog HDL内含的基本原语元件，则：
  - 调用名允许省略，仿真系统在模拟过程中会自动对这些没有指定调用名的基本门赋以一个形式为“基本门名$序列号”的缺省名加以标识。
  - 允许在调用的同时给出门的延时参数与驱动强度说明。
- 以上2点对调用由module定义生成的模块时均不适用。

第六节 结构描述

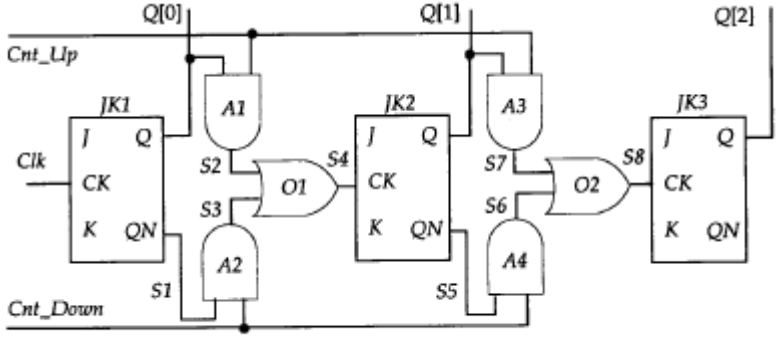
【例6.3】十进制记数器的结构描述

```
module Decade_Ctr (Clock，Z)；
 input Clock；
 output [0:3] Z；
 wire S1，S2；
 and A1 (S1，Z[2]，Z[1])；
 JK_FF
  JK1 (.J(1`b1)，.K(1`b1)，.CK(Clock)，.Q(Z[0])，.NQ() ),
  JK2 (.J(S2)，.K(1`b1)，.CK(Z[0])，.Q(Z[1])，.NQ() ),
  JK3 (.J(1`b1)，.K(1`b1)，.CK(Z[1])，.Q(Z[2])，.NQ() ),
  JK4 (.J(S1)，.K(1`b1)，.CK(Z[0])，.Q(Z[3])，.NQ(S2) )；
endmodule
```



【例6.4】3位可逆计数器的逻辑结构

```
module  Up_Down (Clk，Cnt_Up，Cnt_Down，Q)；
input  Clk，Cnt_Down；
output  [0:2] Q；
wire  S1，S2，S3，S4，S5，S6，S7，S8；
JK_FF  JK1 (1`b1，1`b1，clk，Q[0]，S1),
        JK2 (1`b1，1`b1，S4，Q[1]，S5),
        JK3 (1`b1，1`b1,S8，Q[2])；
and A1 (S2，Cnt_Up，Q[0]),
     A2 (S3，S1，Cnt_Down),
     A3 (S7，Q[1]，Cnt_Up),
     A4 (S6，S7，Cnt_Down)；
or  O1 (S4，S2，S3),
    O2 (S8，S7，S6)；
endmodule
```



所有触发器的J、K输入端口与高电平相连

小结

- 结构描述是相对行为描述而言的另一种Verilog HDL的重要描述方法，结构描述的概念可以认为是在门级描述的基础上引申而来。
- 门级描述就是对由基本逻辑门级元件互连而成的具有一定功能的电路模块的描述，将这些基本逻辑门级元件用一个个功能模块加以替换，门级描述的概念就拓展到一般意义上的结构描述。
- 与行为描述侧重反映模块的行为特征相比，结构描述侧重反映模块内部的具体构造。

内 容

- 第一节 硬件描述语言
- 第二节 Verilog HDL设计入门
- 第三节 Verilog基础知识
- 第四节 行为描述
- 第五节 数据流描述
- 第六节 结构描述
- 第七节 用户自定义元件
- 第八节 其他论题
- 第九节 仿真
- 第十节 实例分析

第七节 用户自定义元件

7.1 UDP的定义

7.2 组合电路UDP

7.3 时序电路UDP

- 7.3.1 电平触发的时序电路UDP
- 7.3.2 边沿触发的时序电路UDP
- 7.3.3 边沿触发和电平触发混和的时序电路UDP

第七节 用户定义的原语

Why UDP？

- Verilog HDL已经把一些常用的逻辑门内含到语言内部，用户可以方便地调用到自己的设计描述中，这些门级单元被称之为基本原语元件（Primitives）。另外，Verilog还为用户提供自己扩充的方法，用户定义的原语元件（**User Defined Primitives**，UDP）。
- UDP 是一种紧凑的表示简单逻辑关系部件的方法。
- 由几个原语元件组成的逻辑可以用一个UDP表示。在仿真时使用这样的UDP来代替分散的原语元件可以节省计算资源，加快仿真速度。
- UDP 还可用于ASIC库中的基本元件（cell）设计，以及小规模芯片和中规模芯片的设计。

UDP定义形式

primitive *元件名 (输出端名，输入端1，输入端2，……);*

output *输出端类型说明;*

input *输入端类型说明;*

reg *输出端寄存器类说明;*

【initial *输出端初始化；】*

table

table表项;

endtable

endprimitive

注：**UDP不可综合**

UDP特点

- 方括号内的项可缺省。
- UDP只有一个输出端，且必须是第一个端口；但可以有一个或多个输入端，最多允许有 10 个。
- UDP 所有端口变量必须是标量，不允许使用双向端口。
- 不支持Z（高阻）逻辑值，若输入取z值，则当作x处理。
- 对于寄存器说明，只有输出端才可以被定义为寄存器类。
- initial语句是对输出端进行初始化，也不允许z值，缺省值是x。
- 不支持综合，即不能通过综合把它转变为门级结构逻辑。

第七节 用户定义的原语

组合电路UDP

【例7.1】用UDP描述二选一多路选择器。

```
primitive mux2_1 (out，int1，int2，sel);
output out;
intput int1，int2，sel;
table
// int1   int2   sel   ：out
    0      ?      1   ：   0 ;
    1      ?      1   ：   1 ;
    ?      0      0   ：   0 ;
    ?      1      0   ：   1 ;
    0      0      x   ：   0 ;
endtable
endprimitive
```

说明：

- table表项的第一行通常是标注信号排列顺序的注释。
- 问号表示该信号可以是0、1或x三种可能的逻辑状态。
- 输入端的次序和table表项的次序相一致，输出端由primitive语句中的第一列移至table表项的最后一列。
- 其实，UDP描述的table表，就相当于该电路的真值表。
- *table表中没有的组合默认为x。*即当输入项发生变化，但table表中却没有匹配的表项，则输出一个不定态x值。

组合电路UDP

【例7.2】用UDP描述的半加器。

```
primitive  halfadd ( s，a，b );
output  s;
intput  a，b;
table
//   a      b     :     s
       0      1     :     1 ;
       1      0     :     1 ;
       0      0     :     0 ;
       1      1     :     0 ;
endtable
endprimitive
```

组合电路UDP

【例7.3】全加器进位UDP描述。

```
primitive  add (s，a，b，c);
output  s;
intput  a，b，c;
table
//   a      b      c   :   s
     0      ?      0   :   0 ;
     1      ?      1   :   1 ;
     ?      0      0   :   0 ;
     1      1      ?   :   1 ;
     0      0      ?   :   0 ;
     ?      1      1   :   1 ;
endtable
endprimitive
```

- 如果某输入端变为x，由于table中找不到对应的表项，则输出x值。但实际上可能是定值。

- 因此表项要考虑周到各种输入情况。

第七节 用户定义的原语

7.1 UDP的定义

7.2 组合电路UDP

7.3 时序电路UDP

7.3.1 电平触发的时序电路UDP

7.3.2 边沿触发的时序电路UDP

7.3.3 边沿触发和电平触发混和的时序电路UDP

第七节 用户定义的原语

7.1 UDP的定义

7.2 组合电路UDP

7.3 时序电路UDP

7.3.1 电平触发的时序电路UDP

7.3.2 边沿触发的时序电路UDP

7.3.3 边沿触发和电平触发混和的时序电路UDP

电平触发的锁存器UDP

```
primitive  latch (q，clk，d);
output  q;
reg  q;
intput  d，  clk;
table
//    clk    d     :    state  :    q
      0     1     :     ?     :    1;
      0     0     :     ?     :    0;
      1     ?     :     ?     :    -;
endtable
endprimitive
```

说明：

1. “-”表示没有变化。
2. 当时钟clk=0时，q端输出为输入数据d的值；当时钟clk=1时，q端输出不变化。
3. 与组合电路UDP描述不同，它多了一列对内部状态的描述即用一个寄存器存储内部状态；并且多了对q为寄存器变量的说明。

【例7.4】用UDP描述的电平触发的T触发器。

```
primitive   t_trigger (q，clk，t)；
output  q；
reg  q；
intput  t，  clk；
table
//      clk     t       :     state   :     q
        ?       0       :      1      :     1；
        ?       0       :      0      :     0；
        1       1       :      0      :     1；
        1       1       :      1      :     0；
        0       1       :      0      :     0；
        0       1       :      1      :     1；
endtable
endprimitive
```

第七节 用户定义的原语

7.1 UDP的定义

7.2 组合电路UDP

7.3 时序电路UDP

- 7.3.1 电平触发的时序电路UDP
- ***7.3.2 边沿触发的时序电路UDP***
- 7.3.3 边沿触发和电平触发混和的时序电路UDP

边沿触发的时序电路UDP

【例7.5】上升D触发器UDP

```
primitive  d_ff (q，clk，d);
output  q;
reg  q;
intput  d，  clk;
table
//  clk    d       :    state    :     q
    (01)    0      :      ?      :     0;
    (01)    1      :      ?      :     1;
    (0x)    1      :      1      :     1;
    (0x)    0      :      0      :     0;
    (?0)    ?      :      ?      :     -;
     ?     (??)    :      ?      :     -;
endtable
endprimitive
```

在table表项中，

- 第一、二行表示：当时钟处于上升沿时，输出值只与输入数据d的状态有关；
- 第三、四行表示：当时钟从0转换到x时，如输入数据d和内部状态state相同，则输出为定态0或1，其道理同全加器进位位的UDP描述；
- 第五行表示：当时钟出现非上升沿跳变时，D触发器的输出不受输入数据状态的影响；
- 第六行表示：当时钟处于某个定态，则不管输入数据有何跳变，D触发器的输出端将保持不变。

UDP中的缩记符

缩略符号	说明
0	逻辑0
1	逻辑1
x	不定态
?	0、1或x
b	0或1
-	没有变化
(uv)	从状态u转换为状态v
*	同(??)，表示输入端有任意变化
r	同(01)，即上升沿
f	同(10)，即下降沿
p	(01)、(0x)或(x1)，包含x态的上升沿跳变
n	(10)、(1x)或(x0)，包含x态的下降沿跳变

【例7.6】根据缩记表改写上例。

```
primitive  d_ff (q，clk，d);
output  q;
reg  q;
intput  d，  clk;
table
//   clk     d        :     state    :     q
       r       0        :      ?       :     0;
       r       1        :      ?       :     1;
     (0x)      1        :      1       :     1;
     (0x)      0        :      0       :     0;
     (?0)      ?        :      ?       :     -;
       ?       *        :      ?       :     -;
endtable
endprimitive
```

第七节 用户定义的原语

7.1 UDP的定义

7.2 组合电路UDP

7.3 时序电路UDP

- 7.3.1 电平触发的时序电路UDP
- 7.3.2 边沿触发的时序电路UDP
- ***7.3.3 边沿触发和电平触发混和的时序电路UDP***

边沿触发和电平触发混和的时序电路UDP

【例7.7】带异步清零的D触发器的UDP描述

```
primitive  d_asyn_ff (q，clk，clr，d);
output  q;
reg  q;
intput  d，  clk，clr;
table
//  clk    clr     d       :     state    :     q
    r       0      0       :      ?       :     0;
    r       0      1       :      ?       :     1;
    (0x)    0      1       :      1       :     1;
    (0x)    0      0       :      0       :     0;
    (?0)    0      ?       :      ?       :     -;
     *      1      ?       :      ?       :     0;
     ?      1      ?       :      ?       :     0;
endtable
endprimitive
```

- 在同一个table中能混和电平触发和边沿触发项。如对于带异步清空的D触发器而言，其清零置位是电平触发，而触发器本身是边沿触发。我们的处理方法是，边沿触发在电平触发之前处理。

小结

- 使用 UDP可以在现有的Verilog 语言支持的原语元件的基础上编写新的源语元件。
- UDP 是一个独立元件， 不能用实例调用的方法调用其他的模块。
- UDP 既可以用来表示时序逻辑元件，也可以表示组合逻辑元件。
- UDP 的行为是使用真值表来描述的。
- 调用 UDP 的方式与调用Verilog语言提供的源语元件的方式相同。
- UDP 可用于ASIC库中的基本元件（cell）设计，以及小规模芯片和中规模芯片的设计。