第2章　HDL指南

本章提供HDL语言的速成指南。

2.1　模块

模块是Verilog 的基本描述单位，用于描述某个设计的功能或结构及其与其他模块通信的外部端口。一个设计的结构可使用开关级原语、门级原语和用户定义的原语方式描述；设计的数据流行为使用连续赋值语句进行描述；时序行为使用过程结构描述。一个模块可以在另一个模块中使用。

一个模块的基本语法如下：

```
module module_name (port_list);
  Declarations:
    reg, wire, parameter,
    input, output, inout,
    function, task, . . .
  Statements:
    Initial statement
    Always statement
    Module instantiation
    Gate instantiation
    UDP instantiation
    Continuous assignment
endmodule
```

说明部分用于定义不同的项，例如模块描述中使用的寄存器和参数。语句定义设计的功能和结构。说明部分和语句可以散布在模块中的任何地方；但是变量、寄存器、线网和参数等的说明部分必须在使用前出现。为了使模块描述清晰和具有良好的可读性，最好将所有的说明部分放在语句前。本书中的所有实例都遵守这一规范。

图2-1为建模一个半加器电路的模块的简单实例。

```
module HalfAdder (A, B, Sum, Carry);
  input A, B;
  output Sum, Carry;

  assign #2 Sum = A ^ B;
  assign #5 Carry = A & B;
endmodule
```

模块的名字是*HalfAdder*。模块有4个端口：两个输入端口*A*和*B*，两个输出端口*Sum*和*Carry*。由于没有定义端口的位数，所有端口大小都为1位；同时，由于没有各端口的数据类型说明，这四个端口都是线网数据类型。

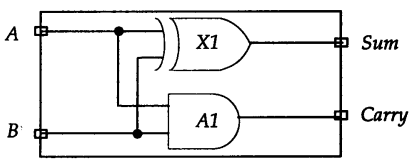

图2-1　半加器电路

模块包含两条描述半加器数据流行为的连续赋值

语句。从这种意义上讲，这些语句在模块中出现的顺序无关紧要，这些语句是并发的。每条语句的执行顺序依赖于发生在变量A和B上的事件。

在模块中，可用下述方式描述一个设计：

1) 数据流方式;

2) 行为方式;

3) 结构方式;

4) 上述描述方式的混合。

下面几节通过实例讲述这些设计描述方式。不过有必要首先对 Verilog HDL的时延作简要介绍。

2.2 时延

Verilog HDL模型中的所有时延都根据时间单位定义。 下面是带时延的连续赋值语句实例。

```
assign #2 Sum = A ^ B;
```

#2指2个时间单位。

使用编译指令将时间单位与物理时间相关联。这样的编译器指令需在模块描述前定义，如下所示:

```
` timescale 1ns /100ps
```

此语句说明时延时间单位为1ns并且时间精度为100ps (时间精度是指所有的时延必须被限定在0.1ns内)。 如果此编译器指令所在的模块包含上面的连续赋值语句，#2 代表2ns。

如果没有这样的编译器指令，Verilog HDL 模拟器会指定一个缺省时间单位。 IEEE Verilog HDL 标准中没有规定缺省时间单位。

2.3 数据流描述方式

用数据流描述方式对一个设计建模的最基本的机制就是使用连续赋值语句。在连续赋值语句中，某个值被指派给线网变量。 连续赋值语句的语法为:

```
assign [delay] LHS_net = RHS_ expression;
```

右边表达式使用的操作数无论何时发生变化，右边表达式都重新计算，并且在指定的时延后变化值被赋予左边表达式的线网变量。时延定义了右边表达式操作数变化与赋值给左边表达式之间的持续时间。如果没有定义时延值，缺省时延为0。

图2-2显示了使用数据流描述方式对2-4解码器电路的建模的实例模型。

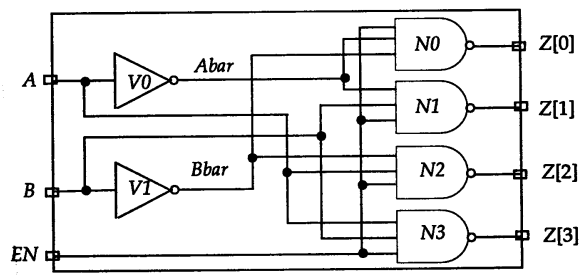

图2-2 2-4解码器电路

```
`timescale 1ns/ 1ns
module Decoder2x4 (A, B, EN, Z);
  input A, B, EN;
  output [ 0 :3] Z;
  wire Abar, Bbar;

  assign #1 Abar = ~ A;                          / 语句 1。
  assign #1 Bbar = ~ B;                          / 语句 2。
  assign #2 Z[0] = ~ (Abar & Bbar & EN) ;   / /语句 3。
  assign #2 Z[1] = ~ (Abar & B & EN) ;      / /语句 4。
  assign #2 Z[2] = ~ (A & Bbar & EN) ;      / /语句 5。
  assign #2 Z[3] = ~ (A & B & EN) ;         / /语句 6。
endmodule
```

以反引号“ ` ”开始的第一条语句是编译器指令，编译器指令`timescale 将模块中所有时延的单位设置为1 ns，时间精度为1 ns。例如，在连续赋值语句中时延值#1和#2分别对应时延1 ns和2 ns。

模块*Decoder2x4*有3个输入端口和1个4位输出端口。线网类型说明了两个连线型变量 *Abar* 和*Bbar* (连线类型是线网类型的一种)。此外，模块包含6个连续赋值语句。

参见图2-3中的波形图。当*EN*在第5 ns变化时,语句3、4、5和6执行。这是因为*EN*是这些连续赋值语句中右边表达式的操作数。 *Z*[0]在第7 ns时被赋予新值0。当*A*在第15 ns变化时，语句1、5和6执行。执行语句5和6不影响*Z*[0]和*Z*[1]的取值。执行语句5导致*Z*[2]值在第17 ns变为0。执行语句1导致*Abar*在第16 ns被重新赋值。由于*Abar*的改变，反过来又导致*Z*[0]值在第18 ns变为1。

请注意连续赋值语句是如何对电路的数据流行为建模的；这种建模方式是隐式而非显式的建模方式。此外,连续赋值语句是并发执行的，也就是说各语句的执行顺序与其在描述中出现的顺序无关。

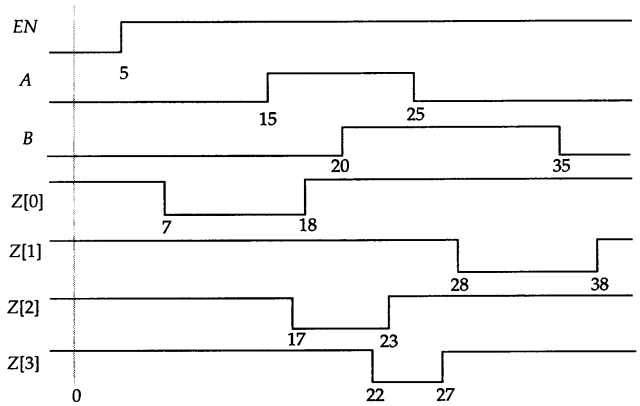

图2-3 连续赋值语句实例

2.4 行为描述方式

设计的行为功能使用下述过程语句结构描述：

1) initial语句：此语句只执行一次。

2) always语句：此语句总是循环执行，或者说此语句重复执行。

只有寄存器类型数据能够在这两种语句中被赋值。寄存器类型数据在被赋新值前保持原有值不变。所有的初始化语句和always语句在0时刻并发执行。

下例为always语句对1位全加器电路建模的示例，如图2-4。

```
module FA_Seq (A, B, Cin, Sum, Cout);
  input A, B, Cin;
  output Sum, Cout;
  reg Sum, Cout;
  reg T1, T2, T3;
  always
    @ ( A or B or Cin ) begin
      Sum = (A ^ B) ^ Cin;
      T1 = A & Cin;
      T2 = B & Cin;
      T3 = A & B;
      Cout = (T1| T2) | T3;
    end
endmodule
```

模块*FA_Seq*有三个输入和两个输出。由于*Sum*、*Cout*、*T1*、*T2*和*T3*在always语句中被赋值，它们被说明为reg类型(reg是寄存器数据类型的一种)。always语句中有一个与事件控制(紧跟在字符@后面的表达式)。相关联的顺序过程(begin-end对)。这意味着只要*A*、*B*或*Cin*上发生事件，即*A*、*B*或*Cin*之一的值发生变化，顺序过程就执行。在顺序过程中的语句顺序执行，并且在顺序过程执行结束后被挂起。顺序过程执行完成后，always语句再次等待*A*、*B*或*Cin*上发生的事件。

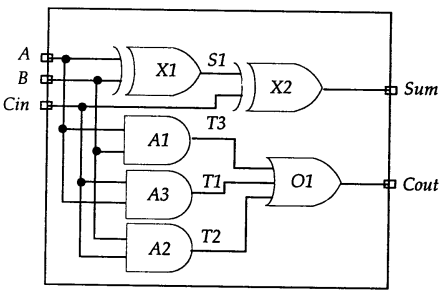

图2-4 1位全加器电路

在顺序过程中出现的语句是过程赋值模块化的实例。模块化过程赋值在下一条语句执行前完成执行。过程赋值可以有一个可选的时延。

时延可以细分为两种类型：

1) 语句间时延：这是时延语句执行的时延。

2) 语句内时延：这是右边表达式数值计算与左边表达式赋值间的时延。

下面是语句间时延的示例：

```
Sum = (A ^ B) ^ Cin;
#4 T1 = A & Cin;
```

在第二条语句中的时延规定赋值延迟4个时间单位执行。就是说，在第一条语句执行后等待4个时间单位，然后执行第二条语句。下面是语句内时延的示例。

```
Sum = #3 (A^ B) ^ Cin;
```

这个赋值中的时延意味着首先计算右边表达式的值，等待3个时间单位，然后赋值给*Sum*。

如果在过程赋值中未定义时延，缺省值为0时延，也就是说，赋值立即发生。这种形式以及在always语句中指定语句的其他形式将在第8章中详细讨论。

下面是initial语句的示例：

```
`timescale 1ns / 1ns
```

```
module Test (Pop, Pid);
  output Pop, Pid;
  reg Pop, Pid;

  initial
    begin
      Pop = 0;            //语句 1。
      Pid = 0;            //语句 2。
      Pop = #5 1;         //语句 3。
      Pid = #3 1;         //语句 4。
      Pop = #6 0;         //语句 5。
      Pid = #2 0;         //语句 6。
  end
endmodule
```

这一模块产生如图2-5所示的波形。initial语句包含一个顺序过程。这一顺序过程在 0 ns时开始执行，并且在顺序过程中所有语句全部执行完毕后，initial语句永远挂起。这一顺序过程包含带有定义语句内时延的分组过程赋值的实例。语句 1和2在0 ns时执行。第三条语句也在 0 时刻执行,导致*Pop* 在第5 ns时被赋值。语句4在第5 ns执行,并且*Pid* 在第8 ns被赋值。同样，Pop在14 ns被赋值0，Pid在第16 ns被赋值0。第6条语句执行后，initial语句永远被挂起。第 8 章将更详细地讲解initial语句。

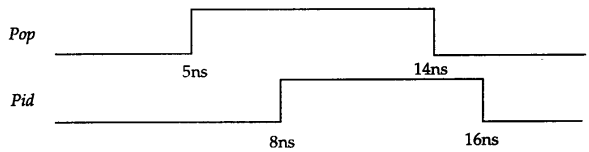

图2-5 *Test* 模块的输出波形

2.5 结构化描述形式

在Verilog HDL中可使用如下方式描述结构：

1) 内置门原语(在门级)；
2) 开关级原语(在晶体管级)；
3) 用户定义的原语(在门级)；
4) 模块实例 (创建层次结构)。

通过使用线网来相互连接。下面的结构描述形式使用内置门原语描述的全加器电路实例。该实例基于图2-4所示的逻辑图。

```
module FA_Str (A, B, Cin, Sum, Cout);
  input A, B, Cin;
  output Sum, Cout;
  wire S1, T1, T2, T3;

  xor
    X1 (S1, A, B),
    X2 (Sum, S1, Cin);

  and
```

```
    A1 (T3, A, B),
    A2 (T2, B, Cin),
    A3 (T1, A, Cin),

  or
    O1 (Cout, T1, T2, T3);
endmodule
```

在这一实例中，模块包含门的实例语句，也就是说包含内置门 xor、and和or 的实例语句。门实例由线网类型变量 *S1*、*T1*、*T2*和*T3*互连。由于没有指定的顺序，门实例语句可以以任何顺序出现；图中显示了纯结构； xor、and和or是内置门原语；*X1*、*X2*、*A1*等是实例名称。紧跟在每个门后的信号列表是它的互连；列表中的第一个是门输出，余下的是输入。例如， *S1*与xor 门实例*X1*的输出连接，而*A*和*B*与实例*X1*的输入连接。

4位全加器可以使用 4个1位全加器模块描述，描述的逻辑图如图 2-6所示。下面是4位全加器的结构描述形式。

```
module FourBitFA (FA, FB, FCin, FSum, FCout);
  parameter SIZE = 4;
  input [SIZE:1] FA, FB;
  output [SIZE:1] FSum
  input FCin;
  input FCout;
  wire [ 1: SIZE-1] FTemp;
FA_Str
  FA1( .A (FA[1]), .B(FB[1]), .Cin(FCin),
     .Sum(FSum[1]), .Cout(FTemp[2])),
  FA2( .A (FA[2]), .B(FB[2]), .Cin(FTemp[1]),
     .Sum(FSum[2]), .Cout(FTemp[2])),
  FA3(FA[3], FB[3], FTemp[2], FSum[3], FTemp[3],
  FA4(FA[4], FB[4], FTemp[3], FSum[4], FCout);
endmodule
```

在这一实例中，模块实例用于建模 4位全加器。在模块实例语句中，端口可以与名称或位置关联。前两个实例 *FA1*和*FA2*使用命名关联方式，也就是说，端口的名称和它连接的线网被显式描述（每一个的形式都为“ *.port_name* (*net_name*)）。最后两个实例语句，实例 *FA3*和*FA4*使用位置关联方式将端口与线网关联。这里关联的顺序很重要，例如，在实例 *FA4*中，第一个*FA*[4]与*FA_Str* 的端口*A*连接，第二个*FB*[4]与*FA_Str* 的端口*B*连接，余下的由此类推。

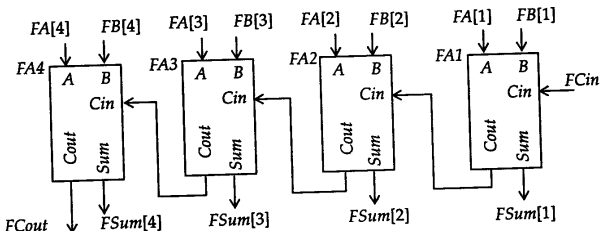

图2-6 4位全加器

2.6 混合设计描述方式

在模块中，结构的和行为的结构可以自由混合。也就是说，模块描述中可以包含实例化

的门、模块实例化语句、连续赋值语句以及 always语句和initial语句的混合。它们之间可以相互包含。来自always语句和initial语句（切记只有寄存器类型数据可以在这两种语句中赋值）的值能够驱动门或开关，而来自于门或连续赋值语句（只能驱动线网）的值能够反过来用于触发always语句和initial语句。

下面是混合设计方式的1位全加器实例。

```
module FA_Mix (A, B, Cin, Sum, Cout);
  input A,B, Cin;
  output Sum, Cout;
  reg Cout;
  reg T1, T2, T3;
  wire S1;

  xor X1(S1, A, B);              //门实例语句。

  always
    @ ( A or B or Cin ) begin  // always 语句。
      T1 = A & Cin;
      T2 = B & Cin;
      T3 = A & B;
      Cout = (T1| T2) | T3;
    end

  assign Sum = S1 ^ Cin;  // 连续赋值语句。
endmodule
```

只要*A*或*B*上有事件发生，门实例语句即被执行。只要 *A*、*B*或*Cin*上有事件发生，就执行always 语句，并且只要*S1*或*Cin*上有事件发生，就执行连续赋值语句。

2.7 设计模拟

Verilog HDL不仅提供描述设计的能力，而且提供对激励、控制、存储响应和设计验证的建模能力。激励和控制可用初始化语句产生。验证运行过程中的响应可以作为“变化时保存”或作为选通的数据存储。最后，设计验证可以通过在初始化语句中写入相应的语句自动与期望的响应值比较完成。

下面是测试模块*Top*的例子。该例子测试2.3节中讲到的*FA_Seq*模块。

```
`timescale 1ns/1ns
module Top;          // 一个模块可以有一个空的端口列表。
  reg PA, PB, PCi;
  wire PCo, PSum;

  // 正在测试的实例化模块：
  FA_Seq F1(PA, PB, PCi, PSum, PCo);      // 定位。

  initial
    begin: ONLY_ONCE
      reg [3:0] Pal;
       //需要4位，Pal才能取值8。

      for (Pal = 0; Pal < 8; Pal = Pal + 1)
```

```
        begin
          {PA, PB, PCi} = Pal;
          #5 $display ("PA, PB, PCi = %b%b%b, PA, PB, PCi,
                       " : : : PCo, PSum=%b%b", PCo, PSum);
        end
      end
    endmodule
```

在测试模块描述中使用位置关联方式将模块实例语句中的信号与模块中的端口相连接。也就是说，*PA*连接到模块*FA_Seq*的端口*A*，*PB*连接到模块*FA_Seq*的端口*B*，依此类推。注意初始化语句中使用了一个for循环语句，在*PA*、*PB*和*PCi*上产生波形。for 循环中的第一条赋值语句用于表示合并的目标。自右向左，右端各相应的位赋给左端的参数。初始化语句还包含有一个预先定义好的系统任务。系统任务 $display将输入以特定的格式打印输出。

系统任务 $display调用中的时延控制规定 $display任务在5个时间单位后执行。这 5个时间单位基本上代表了逻辑处理时间。即是输入向量的加载至观察到模块在测试条件下输出之间的延迟时间。

这一模型中还有另外一个细微差别。 *Pal*在初始化语句内被局部定义。为完成这一功能，初始化语句中的顺序过程（ begin-end）必须标记。在这种情况下 , ONLY_ONCE是顺序过程标记。如果在顺序过程内没有局部声明的变量，就不需要该标记。测试模块产生的波形如图2-7显示。下面是测试模块产生的输出。

```
PA, PB, PCi = 000 ::: PCo, PSum = 00
PA, PB, PCi = 001 ::: PCo, PSum = 01
PA, PB, PCi = 010 ::: PCo, PSum = 01
PA, PB, PCi = 011 ::: PCo, PSum = 10
PA, PB, PCi = 100 ::: PCo, PSum = 01
PA, PB, PCi = 101 ::: PCo, PSum = 10
PA, PB, PCi = 110 ::: PCo, PSum = 10
PA, PB, PCi = 111 ::: PCo, PSum = 11
```

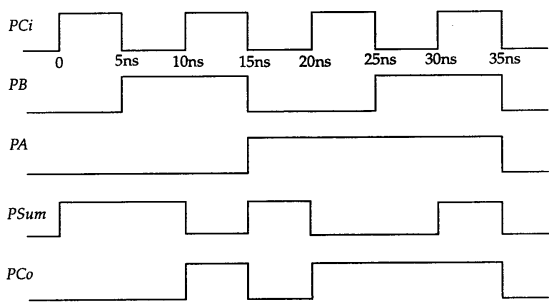

图2-7 测试模块*Top*执行产生的波形

验证与非门交叉连接构成的*RS_FF*模块的测试模块如图2-8所示。

```
`timescale 10ns/1ns
module RS_FF (Q, Qbar, R, S);
  output Q, Qbar;
  input R, S;

  nand #1 (Q, R, Qbar);
```

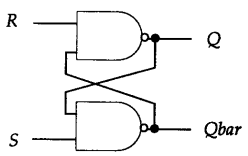

图2-8 交叉连接的与非门

```
    nand #1 (Qbar, S, Q);
      //在门实例语句中，实例名称是可选的。
endmodule

module Test;
  reg TS, TR;
  wire TQ, TQb;

  //测试模块的实例语句：
  RS_FF NSTA (.Q(TQ), .S(TS), .R(TR), .Qbar(TQb));
    //采用端口名相关联的连接方式。

  // 加载激励：
  initial
   begin:
    TR = 0;
    TS = 0;
    #5 TS = 1;
    #5 TS = 0;
    TR = 1;
    #5 TS = 1;
    TR = 0;
    #5 TS = 0;
    #5 TR = 1;
  end
  //输出显示：
  initial
    $monitor ("At time %t ," , $time,
      "TR = %b, TS=%b, TQ=%b, TQb= %b", TR, TS, TQ, TQb);
endmodule
```

RS_FF模块描述了设计的结构。在门实例语句中使用门时延；例如，第一个实例语句中的门时延为1个时间单位。该门时延意味着如果 R或Qbar假定在T时刻变化，Q将在T+1时刻获得计算结果值。

模块Test是一个测试模块。测试模块中的 RS_FF用实例语句说明其端口用端口名关联方式连接。在这一模块中有两条初始化语句。第一个初始化语句只简单地产生 TS和TR上的波形。这一初始化语句包含带有语句间时延的程序块过程赋值语句。

第二条初始化语句调用系统任务 $monitor。这一系统任务调用的功能是只要参数表中指定的变量值发生变化就打印指定的字符串。产生的相应波形如图 2-9所示。下面是测试模块产生的输出。请注意`timescale指令在时延上的影响。

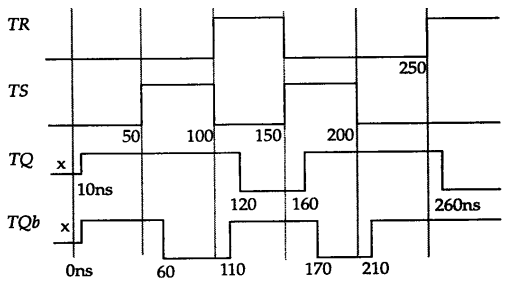

图2-9 Test模块产生的波形

```
At time      0, TR=0, TS=0, TQ=x, TQb= x
At time     10, TR=0, TS=0, TQ=1, TQb= 1
At time     50, TR=0, TS=1, TQ=1, TQb= 1
At time     60, TR=0, TS=1, TQ=1, TQb= 0
At time    100, TR=1, TS=0, TQ=1, TQb= 0
At time    110, TR=1, TS=0, TQ=1, TQb= 1
At time    120, TR=1, TS=0, TQ=0, TQb= 1
At time    150, TR=0, TS=1, TQ=0, TQb= 1
At time    160, TR=0, TS=1, TQ=1, TQb= 1
At time    170, TR=0, TS=1, TQ=1, TQb= 0
At time    200, TR=0, TS=0, TQ=1, TQb= 0
At time    210, TR=0, TS=0, TQ=1, TQb= 1
At time    250, TR=1, TS=0, TQ=1, TQb= 1
At time    260, TR=1, TS=0, TQ=0, TQb= 1
```

后面的章节将更详细地讲述这些主题。

习题

1. 在数据流描述方式中使用什么语句描述一个设计？
2. 使用`timescale 编译器指令的目的是什么？举出一个实例。
3. 在过程赋值语句中可以定义哪两种时延？请举例详细说明。
4. 采用数据流描述方式描述图2-4中所示的1位全加器。
5. initial语句与always 语句的关键区别是什么？
6. 写出产生图2-10所示波形的变量*BullsEye*的初始化语句。

图2-10 变量BullsEye 的波形

7. 采用结构描述方式描写图2-2中所示的2-4译码器。
8. 为2.3节中描述的模块*Decode2x4*编写一个测试验证程序。
9. 列出你在Verilog HDL模型中使用的两类赋值语句。
10. 在顺序过程中何时需要定义标记？
11. 使用数据流描述方式编写图 2-11所示的异或逻辑的Verilog HDL描述，并使用规定的时延。
12. 找出下面连续赋值语句的错误。

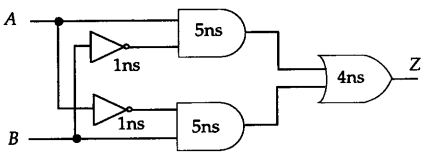

图2-11 异或逻辑

```
assign Reset = #2 ^ WriteBus;
```