

第9章 结构建模

本章讲述 Verilog HDL 中的结构建模方式。结构建模方式用以下三种实例语句描述：

- Gate 实例语句
- UDP 实例语句
- Module 实例语句

第5章和第6章已经讨论了门级建模方式和 UDP 建模方式，本章讲述模块实例语句。

9.1 模块

Verilog HDL 中，基本单元定义成模块形式，如下所示：

```
module module_name(port_list);  
    Declarations_and_Statements  
endmodule
```

端口队列 *port_list* 列出了该模块通过哪些端口与外部模块通信。

9.2 端口

模块的端口可以是输入端口、输出端口或双向端口。缺省的端口类型为线网类型（即 wire 类型）。但是，端口可被显式地指定为线网。输出或输入输出端口能够被重新声明为 reg 型寄存器。无论是在线网说明还是寄存器说明中，线网或寄存器必须与端口说明中指定的长度相同。下面是一些端口说明实例。

```
module Micro (PC, Instr, NextAddr;  
    //端口说明  
    input [3:1] PC;  
    output [1:8] Instr;  
    inout [16:1] NextAddr;  
  
    //重新说明端口类型：  
    wire [16:1] NextAddr; //该说明是可选的，但如果指定了，就必须与它的端口说明保持相同长度。  
  
    reg [1:8] Instr;  
    //Instr已被重新说明为reg类型，因此它能在always 语句或在initial语句中赋值。  
    ...  
endmodule
```

9.3 模块实例语句

一个模块能够在另外一个模块中被引用，这样就建立了描述的层次。模块实例语句形式如下：

```
module_name instance_name(port_associations);
```

信号端口可以通过位置或名称关联；但是关联方式不能够混合使用。端口关联形式如下：

```
port_expr          //通过位置。
.PortName (port_expr) //通过名称。
```

port_expr可以是以下的任何类型：

- 1) 标识符 (reg或net)
- 2) 位选择
- 3) 部分选择
- 4) 上述类型的合并
- 5) 表达式 (只适用于输入端口)

在位置关联中，端口表达式按指定的顺序与模块中的端口关联。在通过名称实现的关联中，模块端口和端口表达式的关联被显式地指定，因此端口的关联顺序并不重要。下例使用两个半加器模块构造全加器；逻辑图如图 9-1所示。

```
module HA(A,B,S,C);
  input A,B;
  output S,C;
  parameter AND_DELAY = 1, XOR_DELAY = 2;

  assign #XOR_DELAY S = A ^ B;
  assign #AND_DELAY C = A & B;
endmodule
```

```
module FA(P, Q, Cin, Sum, Cout);
  input P, Q, Cin;
  output Sum, Cout;
  parameter OR_DELAY = 1;
  wire S1, C1, C2;

  //两个模块实例语句
  HA h1 (P, Q, S1, C1); //通过位置关联。
  HA h2 (.A(Cin), .S(Sum), .B(S1), .C(C2)); //通过端口与信号的名字关联。
  //门实例语句：
  or #OR_DELAY O1 (Cout, C1, C2);
endmodule
```

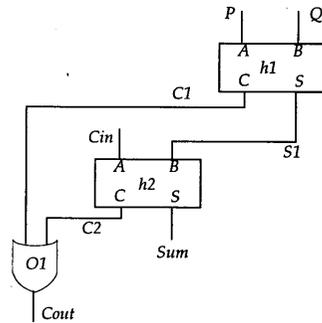


图9-1 使用两个半加器模块构造的全加器

在第一个模块实例语句中，HA是模块的名字，h1是实例名称，并且端口按位置关联，即信号P与模块（HA）的端口A连接，信号Q与端口B连接，S1与S连接，C1与模块端口C连接。在第二个实例中，端口按名称关联，即模块（HA）和端口表达式间的连接是显示地定义的。

下例是使用不同端口表达式形式的模块实例语句。

```
Micro M1 (UdIn[3:0], {WrN, RdN}, Status[0], Status[1],
  & UdOut [0:7], TxData);
```

这个实例语句表示端口表达式可以是标识符（TxData）、位选择（Status[0]）、部分位选择（UdIn[3:0]）、合并（{WrN,RdN}）或一个表达式（& udOut[0:7]）；表达式只能够连接到输入端口。

9.3.1 悬空端口

在实例语句中，悬空端口可通过将端口表达式表示为空白来指定为悬空端口，例如：

```
DFF d1 (.Q(QS), .Qbar(), .Data(D),
        .Preset(), .Clock(CK)); //名称对应方式。
```

```
DFF d2 (QS, , D, , CK; //位置对应方式。
//输出端口Qbar悬空。
//输入端口Preset打开, 其值设定为z。
```

在这两个实例语句中, 端口 *Qbar* 和 *Preset* 悬空。

模块的输入端悬空, 值为高阻态 *z*。模块的输出端口悬空, 表示该输出端口废弃不用。

9.3.2 不同的端口长度

当端口和局部端口表达式的长度不同时, 端口通过无符号数的右对齐或截断方式进行匹配。例如:

```
module Child(Pba, Ppy);
    input [5:0] Pba;
    output [2:0] Ppy;
    ...
endmodule
```

```
module Top;
    wire [1:2] Bdl;
    wire [2:6] Mpr;

    Child C1(Bdl, Mpr);
endmodule
```

在对 *Child* 模块的实例中, *Bdl*[2] 连接到 *Pba*[0], *Bdl*[1] 连接到 *Pba*[1], 余下的输入端口 *Pba*[5]、*Pba*[4] 和 *Pba*[3] 悬空, 因此为高阻态 *z*。与之相似, *Mpr*[6] 连接到 *Ppy*[0], *Mpr*[5] 连接到 *Ppy*[1], *Mpr*[4] 连接到 *Ppy*[2]。参见图 9-2。

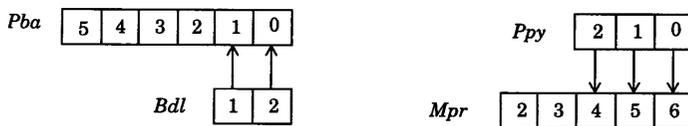


图9-2 端口匹配

9.3.3 模块参数值

当某个模块在另一个模块内被引用时, 高层模块能够改变低层模块的参数值。模块参数值的改变可采用下述两种方式:

- 1) 参数定义语句 (*defparam*) ;
- 2) 带参数值的模块引用。

1. 参数定义语句

参数定义语句形式如下:

```
defparam hier_path_name1= value1,
        hier_path_name2= value2, ...;
```

较低层模块中的层次路径名参数可以使用如下语句显式定义 (层次路径名在下一章中讲

述)。下面是一个例。模块FA和HA已在本节前面描述过。

```

module TOP (NewA,NewB,NewS,NewC);
  input NewA,NewB;
  output NewS,NewC;
  defparam Ha1.XOR_DELAY = 5,
            //实例Ha1中的参数XOR_DELAY。
            Ha1.AND_DELAY = 2;
            //实例Ha1中参数的AND_DELAY。

  HA Ha1 (NewA, NewB, NewS,NewC)
endmodule

module TOP2 (NewP, NewQ, NewCin,NewSum,NewCout
  input NewP, NewQ, NewCin
  output NewSum,NewCout;
  defparam Fa1.h1.XOR_DELAY = 2,
            //实例Fa1的实例h1中的参数XOR_DELAY。
            Fa1.h1.AND_DELAY = 3,
            //实例Fa1的实例h1中的参数AND_DELAY。
            Fa1.OR_DELAY = 3;
            //实例Fa1中的参数OR_DELAY )
  FA Fa1 (NewP, NewQ, NewCin,NewSum,NewCout)
endmodule

```

2. 带参数值的模块引用

在这种方法中，模块实例语句自身包含有新的参数值。下面的例子在前几节中也出现过，本例中采用带参数的模块引用方式。

```

module TOP3 (NewA,NewB,NewS,NewC);
  input NewA,NewB;
  output NewS,NewC;

  HA #(5,2) Ha1(NewA,NewB,NewS,NewC);
  //第1个值5赋给参数AND_DELAY,该参数在模块HA中说明。
  //第2个值2赋给参数 XOR_DELAY,该参数在模块HA中说明。
endmodule

module TOP4(NewP,NewQ,NewCin,NewCout);
  input NewP,NewQ,NewCin;
  output NewSum,NewCount;
  defparam Fa1.h1.XOR_DELAY = 2, //实例Fa1中实例h1的参数XOR_DELAY。
            Fa1.h1.AND_DELAY = 3; //实例Fa1中实例h1的参数AND_DELAY。

  FA #(3) Fa1(NewP,NewQ,NewCin,NewCout);
  //值3是参数OR_DELAY的新值。
endmodule

```

模块实例语句中参数值的顺序必须与较低层被引用的模块中说明的参数顺序匹配。在模块TOP3中，AND_DELAY已被设置为5，XOR_DELAY已被设置为2。

模块TOP3和TOP4解释说明了带参数的模块引用只能用于将参数值向下传递一个层次（例如，OR_DELAY），但是参数定义语句能够用于替换层次中任意一层的参数值。

应注意到：在带参数的模块引用中，参数的指定方式与门级实例语句中时延的定义方式

相似；但由于对复杂模块的引用时，其实例语句不能像对门实例语句那样指定时延，故此处不会导致混淆。

参数值还可以表示长度。下面是通用的 $M \times N$ 乘法器建模的实例。

```
module Multiplier(Opd_1,Opd_2,Result);
    parameter EM = 4,EN = 2; // 默认值
    input [EM:1] Opd_1;
    input [EN:1] Opd_2;
    output [EM+EN:1] Result;

    assign Result = Opd_1 * Opd_2;
endmodule
```

这个带参数的乘法器可在另一个设计中使用，下面是 8×6 乘法器模块的带参数引用方式：

```
wire [1:8] Pipe_Reg;
wire [1:6] Dbus;
wire [1:14] Addr_Counter;
...
Multiplier #(8,6) M1(Pipe_Reg,Dbus,Addr_Counter);
```

第1个值8指定了参数 EM 的新值，第2个值6指定了参数 EN 的新值。

9.4 外部端口

在迄今为止所见到的模块定义中，端口表列举出了模块外部可见的端口。例如，

```
module Scram_A(Arb,Ctrl,Mem_Blk,Byte);
    input[0:3] Arb;
    input Ctrl;
    input [8:0] Mem_Blk;
    output [0:3] Byte;
    ...
endmodule
```

Arb 、 $Ctrl$ 、 Mem_Blk 和 $Byte$ 为模块端口。这些端口同时也是外部端口，即在实例中，当采用名称关联方式时，外部端口名称用于指定相互连接。下面是模块 $Scram_A$ 的实例。

```
Scram_A SX.Byte(B1),.Mem_Blk(M1),.Ctrl(C1),.Arb(A1));
```

在模块 $Scram_A$ 中，外部端口名称隐式地指定。Verilog HDL 中提供显式方式指定外部端口名称。这可以通过按如下形式指定一个端口来完成：

```
.external_port_name(internal_port_name)
```

下面是同一个例子，只不过是显式地指定外部端口。

```
module Scram_B (.Data(Arb),.Control(Ctrl),
               .Mem_Word(Mem_Blk),.Addr(Byte));

    input [0:3] Arb;
    input Ctrl;
    input [8:0] Mem_Blk;
    output [0:3] Byte;
    ...
endmodule
```

模块 $Scram_B$ 在此实例中指定的外部端口是 $Data$ 、 $Control$ 、 Mem_Word 和 $Addr$ 。端口表显式地表明了外部端口和内部端口之间的连接。注意外部端口无需声明，但是模块的内部端口

却必须声明。外部端口在模块内不可见，但是却要在模块实例语句中使用，而内部端口因为在模块中可见，所以必须在模块中说明。在模块实例语句中，外部端口的使用如下所示：

```
Scram_B S1 (.Addr(A1),.Data(D1),.Control(C1),
           .Mem_Word(M1));
```

在模块定义的端口表中，这两种概念不能混淆，即在模块定义中所有端口必须指定显式的端口名称，或者没有一个端口带有显式的端口名称。

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。

内部端口名称可以是标识符，也可以是下述类型的表达式：

- 位选择；
- 部分选择；
- 位选择、部分选择和标识符的合并。

例如，

```
module Scram_C (Arb[0:2],Ctrl,
               { Mem_Blк[0],Mem_Blк[1]},Byte[3]);
  input [0:3] Arb;
  input Ctrl;
  input [8:0] Mem_Blк;
  output [0:3] Byte;
  ...
endmodule
```

在Scram_C的模块定义中，端口表包括部分选择（Arb[0:2]）、标识符（Ctrl）、合并（{Mem_Blк[0], Mem_Blк[1]}）和位选择（Byte[3]）。在内部端口是位选择、部分选择或合并的情况下，没有隐式地指定外部端口名。因此，在这样的模块实例语句中，模块端口必须通过位置关联相互连接。例如，

```
Scram_C SYA (L1[4:6],CL,MMY[1:0],BT);
```

在这个实例语句中，端口通过位置关联相连接，因此L1[4:6]连接到Arb[0:2]，CL连接到Ctrl，MMY[1]连接到Mem_Blк[0]，MMY[0]连接到Mem_Blк[1]，BT连接到Byte[3]。

若使用端口名称关联（即当内部端口不是标识符时），必须对模块中的端口指定外部端口名。如下面的Scram_D模块定义所示。

```
module Scram_D (.Data(Arb[0:2]),.Control(Ctrl),
               .Mem_Word({Mem_Blк[0],Mem_Blк[1]}),
               .Addr(Byte[3]));
  input [0:3] Arb;
  input Ctrl;
  input [8:0] Mem_Blк;
  output [0:3] Byte;
  ...
endmodule
```

在Scram_D模块实例语句中，端口既能够使用位置连接，也能够使用名称连接，但是不能混合使用。下例的模块实例语句端口通过名称连接。

```
Scram_D SZ (.Data(L1[4:6],.Control(CL),
                 .Mem_Word(MMY[1:0],.Addr(BT)));
```

模块中可以只有外部端口而没有内部端口。即模块在引用中其外部端口可以悬空，不与内部信号相连。例如：

```

module Scram_E (.Data(), .Control (Ctrl),
                .Mem_Word ({ Mem_Blck[0], Mem_Blck [1]}),
                .Addr());

input Ctrl;
input [8:0] Mem_Blck;
...
endmodule

```

模块 *Scram_E* 有两个外部端口 *Data* 和 *Addr*，这两个端口在使用时被悬空。

一个内部端口是否能与多个外部端口连接？Verilog HDL 允许这样连接。例如，

```

module FanOut (.A(CtrlIn), .B(CondOut), .C(CondOut));
input CtrlIn;
output CondOut;

assign CondOut = CtrlIn;
endmodule

```

内部端口 *CondOut* 与两个外部端口 *B* 和 *C* 连接，所以 *CondOut* 的值在 *B* 和 *C* 上都出现。

9.5 举例

下例采用结构模型描述十进制计数器。十进制计数器的逻辑图如图 9-3 所示。

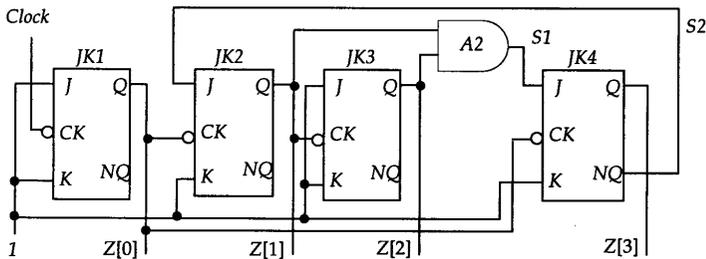


图9-3 十进制计数器

```

module Decade_Ctr (Clock,Z);
input Clock;
output [0:3] Z;
wire S1,S2;

and A1 (S1,Z[2],Z[1]); / 基本门实例语句。

// 4个模块实例语句:
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

```

注意常数作为输入端口信号的法，以及悬空端口。

下面是另一个例子，3位可逆计数器的逻辑结构如图 9-4 所示，其结构描述如下：

```

module Up_Down(Clk,Cnt_Up,Cnt_Down,Q);
input Clk,Cnt_Up,Cnt_Down;
output [0:2] Q;

```

