



第2章

程序结构与数据类型

2.1 Verilog程序结构

【例2-1】

```
module MUX41a (a, b, c, d, s1, s0, y);  
    input  a, b, c, d ;  
    input  s1, s0 ;  
    output y ;  
  
    reg y ;  
  
    always @ ( a or b or c or d or s1 or s0 )  
        begin      : MUX41  //块语句开始  
            case( { s1, s0 } )  
                2'b00 : y <= a ;  
                2'b01 : y <= b ;  
                2'b10 : y <= c ;  
                2'b11 : y <= d ;  
                default : y <= a ;  
            endcase  
        end  
  
endmodule
```

电路模块
端口说明
和定义段

信号类型
定义段

具体描述电
路功能的
Verilog语句

电路模
块功能
描述段

Verilog
表述的可
综合的完
整电路模
块

2.1 Verilog程序结构

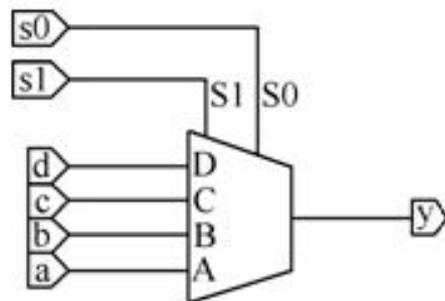


图 2-1 4 选 1 多路选择器

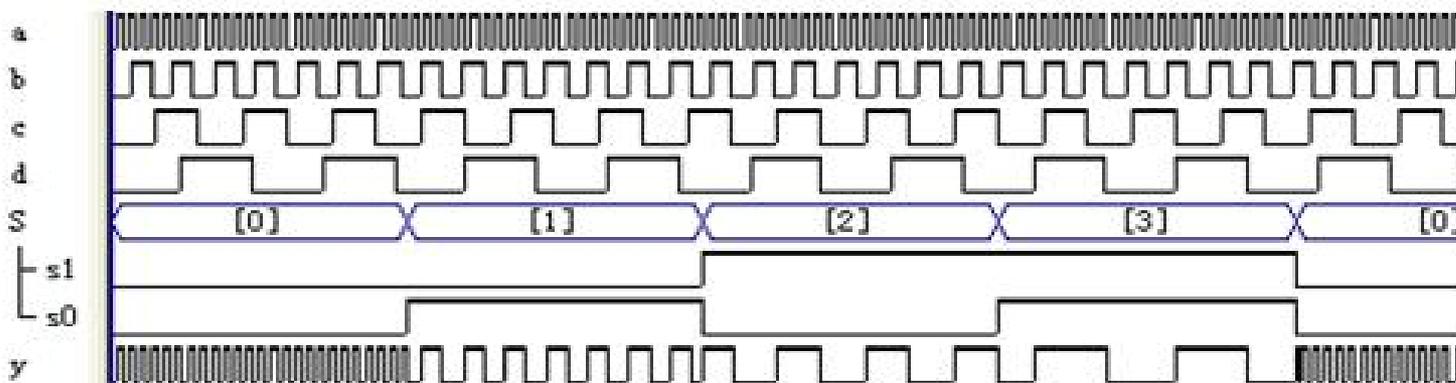


图 2-2 4 选 1 多路选择器 MUX41a 的时序波形

2.1 Verilog程序结构

2.1.1 Verilog的模块的表达方式

模块语句的一般格式

```
module 模块名 (模块端口名表) ;  
    模块端口和模块功能描述 ;  
endmodule
```

2.1 Verilog程序结构

2.1.2 Verilog模块的端口信号名和端口模式

端口定义语句的一般格式

```
input  端口名 1, 端口名 2, ... ;  
output 端口名 1, 端口名 2, ... ;  
inout  端口名 1, 端口名 2, ... ;  
input [msb : lsb] 端口名 1, 端口名 2, ... ;
```

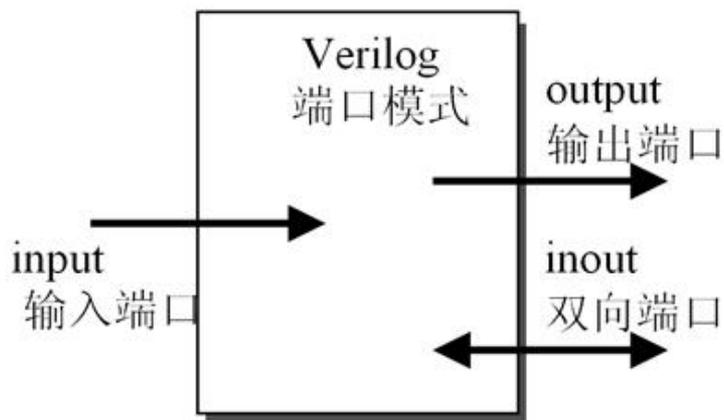


图2-3 Verilog端口模式图

2.1 Verilog程序结构

2.1.3 Verilog信号类型定义

```
reg [3:0] A,B; //定义A和B是4位位宽的寄存器类型信号
wire C,D;     //定义C和D是网线类型信号
```

【例 2-2】

```
module MUX41a (a,b,c,d,s1,s0,y);
    input a,b,c,d,s1,s0;      output y;
    wire [1:0] SEL;           // 定义2元素位矢量SEL为网线型变量wire
    wire AT, BT, CT, DT;     // 定义中间变量，以作连线或信号节点
    assign SEL = {s1,s0};     // 对s1,s0进行并位操作，即SEL[1]=s1; SEL[0]=s0
    assign AT = (SEL==2'D0); //assign语句中的输出变量必须是网线型变量
    assign BT = (SEL==2'D1);
    assign CT = (SEL==2'D2);
    assign DT = (SEL==2'D3);
    assign y = (a & AT) | (b & BT) | (c & CT) | (d & DT); //4个逻辑信号相或
endmodule
```

2.1.4 Verilog模块功能描述

2.2 Verilog 的数据类型

2.2.1 net 网线类型

2.2.2 wire 网线型变量的定义方法

```
wire 变量名 1, 变量名 2, . . . ;
```

```
wire [msb:lsb] 变量名 1, 变量名 2, . . . ;
```

```
wire [7:0] a ;
```

```
wire Y=a1 ^ a2 ; // a1 ^ a2 表示 a1 和 a2 进行逻辑异或操作, 符号^表示逻辑异或
```

```
wire a1, a2;
```

```
assign Y =a1 ^ a2;
```

2.2 Verilog数据类型

2.2.3 register寄存器类型

2.2.4 reg 寄存器型变量的定义方法

```
reg 变量名 1, 变量名 2, . . . ;  
reg [msb:lsb] 变量名 1, 变量名 2, . . . ;
```

```
module seg_7 (input [3:0] num, input en, output reg [6:0] seg );
```

2.2 Verilog数据类型

2.2.5 integer 类型变量定义方法

`integer` 标识符 1, 标识符 2, ..., 标识符 n [msb: lsb] ;

```
module EXAPL (R,G); //定义模块名是 EXAPL, 端口信号是 R 和 G
parameter S=4; //定义参数 S, parameter 是参数定义语句
output[2*S:1] R,G ; //定义两个 8 位输出变量, 即 R[8:1]和 G[8:1]
integer A, B[3:0]; //定义了 5 个 integer 类型: A、B[0]、B[1]、B[2]、B[3], 都是 32 位
reg[2*S:1] R,G; //定义两个端口为 8 位寄存器类型变量, 即 R[8:1]和 G[8:1]
always @( A, B ) begin // 过程语句起始
    B[2] = 367; // 整数完整赋值, 将 367 向 B[2]赋值, B[2]有 32 位二进制位数宽
    R=B[2]; // 32 位 integer 整数类型 B[2]赋给 8 位 reg 类型 R, B[2]高位被截
    A=-20; // 整数完整赋值, 因为 A 有 32 位, 将-20 赋予 A, 对应无符号数 65516
    G=A; // 32 位 integer 整数类型 A 赋给 8 位 reg 类型 G, A 高位被截
    B[0]= 3'B101; // 允许二进制数直接赋给 integer 类型 B[0]。
end
endmodule
```

2.2 Verilog数据类型

2.2.6 存储器类型

【例 2-3】

```
module RAM78 ( output wire[7:0] Q, //定义RAM的8位数据输出端口
               input wire[7:0] D, //定义RAM的8位数据输入端口
               input wire[6:0] A, //定义RAM的7位地址输入端口
               input wire CLK,WREN ) ; //定义时钟和写允许控制
    reg[7:0] mem[127:0] /* synthesis ram_init_file="DATA7X8.mif" */ ;
    always @(posedge CLK )
        if (WREN) mem[A] <= D; //在CLK上升沿将数据口D的数据锁入地址对应单元中
    assign Q = mem[A]; //同时，地址对应单元的数据被输出至端口
endmodule
```

2.2 Verilog数据类型

2.2.6 存储器类型

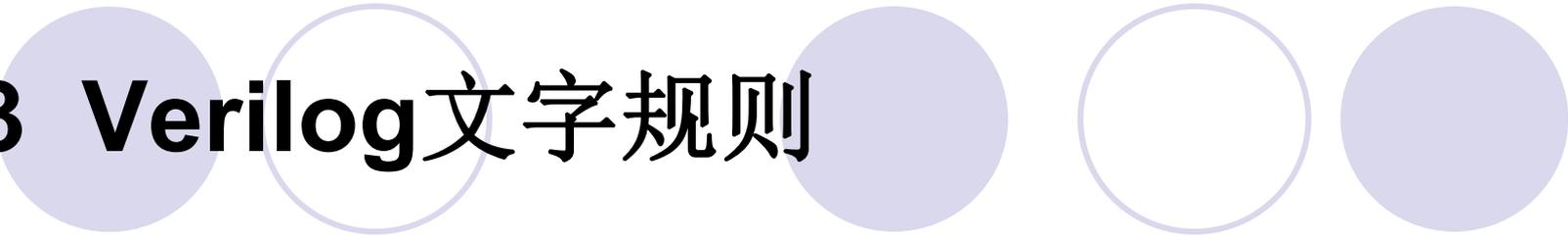
```
parameter width=8, msize=1024;
reg[width-1:0] MEM87[msize-1:0];

reg[7:0] mem87[128:0];
mem87[16]=8'b11001001; //mem87 存储器的第 16 单元被赋值为二进制数 11001001
mem87[122]=76;        //mem87 存储器的第 122 单元被赋值为十进制数 76。

reg [15:0] A;          // 定义了一个 16 位的寄存器
reg MEM[15:0];        // 定义了一个字长为 1, 即 1 位的, 容量深度为 16 的存储器

A[5] = 1'b0;          // 允许对寄存器 A 的第 5 位赋值 0
MEM[7] = 1'b1;        // 允许对存储器 MEM 的第 7 个单元赋值 1
A = 16'hABCD ;       // 允许对寄存器 A 整体赋值
MEM = 16'hABCD;      // 错误! 不允许对存储器多个或者所有单元同时赋值
```

2.3 Verilog文字规则



2.3.1 Verilog的4种逻辑状态

2.3.2 Verilog的数字表达形式

<位宽> '<进制> <数字>

2.3.3 数据类型表示方式

2.3 Verilog基本要素与文字规则

2.3.4 常量

1. 整数

```
reg [3:0] A ; reg [5:0] B ; reg [31:0] C ;
```

```
A<= 6'B11_0110 ; // A 实际获得赋值 4'B0110，高 2 位被截去。进制符号 b 或 B 大小写都可。  
A<= 'o466;      // 'o466 = 'H136，A 实际获得低 4 位：4'B0110。高位被截去。  
A<= 123;        // 123=32'h0000_007B，转换为 32 位二进制数，A 实际获得赋值 4'B1011。  
A<= 8'hAC;      // A 实际获得赋值 4'h1100，高 4 位被截去。  
C <= -5;        // -5=32'hFFFFFFFB ， A 即获得赋值 32'hFFFFFFFB。  
B <= -7'd30;    // -7'd30 = 7'H62，A 实际获得赋值=6'H22，高 1 位被截去。
```

2.3 Verilog基本要素与文字规则

2.3.4 常量

2. 实数

```
1.335,          88_670_551.453_909 (=88670551.453909),    1.0,  
44.99e-2 (=0.4499),    0.1,          3E-4 (=0.0003)
```

3. 字符串

```
"ERROR" ,    "Both S and Q equal to 1" ,    "X" ,    "BB$CC"
```

```
reg [8*5:1] ALM; initial begin ALM = "ERROR" ; end
```

2.3 Verilog基本要素与文字规则

2.3.5 标识符、关键字及其它文字规则

1. 标识符

```
Decoder_1, FFT, Sig_N, Not_Ack, State0, _Decoder_, REG

2FFT // 起始为数字
Sig_#N // 符号“#”不能成为标识符的构成
Not-Ack // 符号“-”不能成为标识符的构成
data_ _BUS // 标识符中不能有双下划线
reg // 关键词
ADDER* // 标识符中不允许包含字符*
```

2.3 Verilog基本要素与文字规则

2.3.5 标识符、关键字及其它文字规则

2. 关键字

3. 注释符号

4. 规范的程序书写格式

5. 文件取名和存盘

2.3 Verilog基本要素与文字规则

2.3.6 参数定义关键词parameter和localparam的用法

parameter 标识符名 1 = 表达式或数值 1, 标识符名 2 = 表达式或数值 2, ... ;

```
parameter A=15 , B=4'b1011, C=8'hAC ;
```

```
parameter d=8'b1001_0011 , e=8'sb10101101 ;
```