数 字 电 路

Digital Circuits

09_PLD与Verilog HDL(1)

张俊霞
zjx@ustc.edu.cn

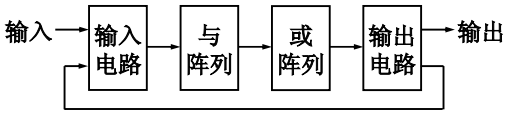
内容提纲

- **PLD**
  - 基本结构
  - 实现组合逻辑电路
- **Verilog HDL**
  - 基本语法规则
  - 变量数据类型
  - 程序基本结构
  - 描述组合逻辑电路

可编程逻辑器件

- **可编程逻辑器件（Programmable Logic Device，简称PLD）是一种可以由用户定义和设置逻辑功能的器件**
  - 与中小规模通用逻辑器件相比，具有集成度高、速度快、功耗低、可靠性高等优点
  - 与其他专用集成电路相比，具有产品开发周期短、用户投资风险小、小批量生产成本低等优势
- **按集成度PLD可分为**
  - 低密度PLD：PROM、PLA、PAL、GAL
  - 高密度PLD：CPLD、FPGA

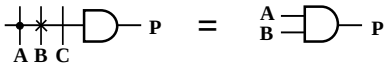
PLD基本结构

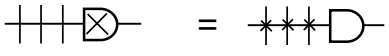


- **输入电路**
  - 提供互补输入变量(原变量和反变量)
- **与阵列**
  - 产生逻辑函数所需的乘积项
- **或阵列**
  - 选择乘积项，形成与或式，实现逻辑函数
- **输出电路**
  - 提供不同的输出方式

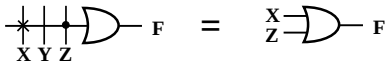
PLD中逻辑符号表示

- **与门：**





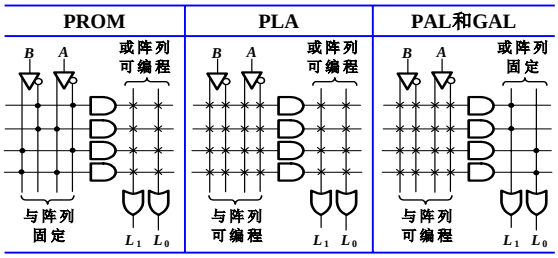
- **或门：**



- **互补输入缓冲器：**

标记	连接方式
●	固定连接
×	编程连接
空白	无连接

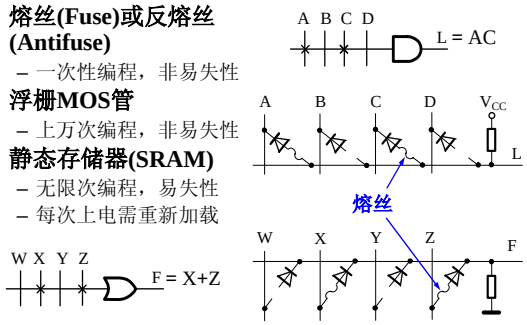
低密度PLD的与、或阵列结构



- PROM: Programmable Read Only Memory, **可编程只读存储器**
- PLA: Programmable Logic Array, **可编程逻辑阵列**
- PAL: Programmable Array Logic, **可编程阵列逻辑**
- GAL: Gate Array Logic, **门阵列逻辑**

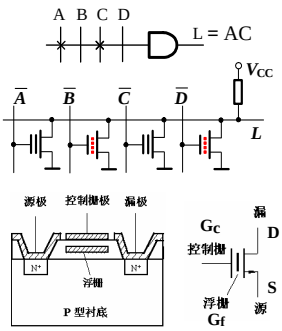
PLD中编程元件

- **熔丝(Fuse)或反熔丝(Antifuse)**
  - 一次性编程，非易失性
- **浮栅MOS管**
  - 上万次编程，非易失性
- **静态存储器(SRAM)**
  - 无限次编程，易失性
  - 每次上电需重新加载



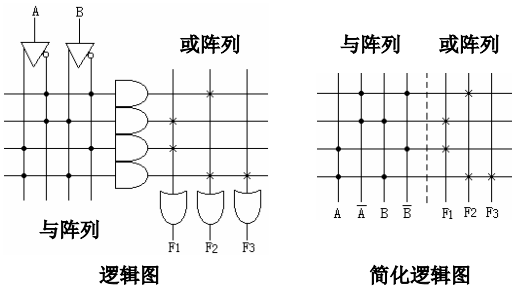
浮栅MOS管

- **利用浮栅上有/无电荷(编程)，实现无/有连接**
  - 浮栅上无负电荷，控制栅加正常逻辑电压，可控制 MOS管导通和截止，相当于建立了连接
  - 浮栅上有负电荷，开启电压升高，控制栅加正常逻辑电压，MOS管均不导通，相当于无连接
- **浮栅MOS管的编程方法与其工艺结构有关**
  - 电写入和紫外线擦除
  - 电写入和电擦除

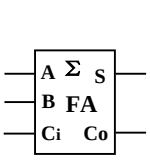


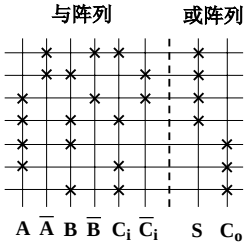
示例—PROM实现组合逻辑

F1 = $\overline{A}$·B +A·$\overline{B}$　F2 = $\overline{A}$·$\overline{B}$ +A·B　F3 = A·B



示例—PLA实现一位全加器





$$C_O = AB + (A \oplus B)C_i$$
$$= AB + AC_i + BC_i$$

$$S = A \oplus B \oplus C_i = \overline{A}\overline{B}C_i + \overline{A}B\overline{C_i} + A\overline{B}\overline{C_i} + ABC_i$$

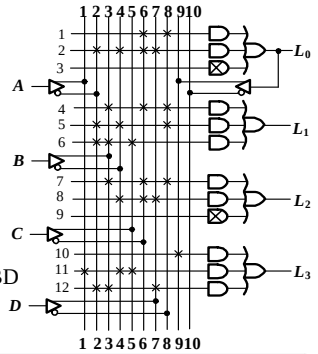
示例—PAL实现组合逻辑

$$L_0 = \overline{C}\,\overline{D} + \overline{A}\,B\,C\,D$$

$$L_1 = B\overline{C}\,\overline{D} + \overline{A}B\overline{D} + \overline{A}BC$$

$$L_2 = B\overline{C}\,\overline{D} + \overline{B}C\overline{D}$$

$$L_3 = L_0 + A\overline{B}C + \overline{A}BD$$
$$= \overline{C}\,\overline{D} + \overline{A}BCD + A\overline{B}C + \overline{A}BD$$



硬件描述语言概述

- **HDL (Hardware Description Languag)**
  - 是一种以文本形式来描述数字系统硬件的结构和行为的语言
  - 可以从多种抽象层次对数字系统建模
- **两种常用HDL**
  - VHDL：1981年由美国国防部组织开发，1987年成为IEEE标准
  - Verilog HDL：1983年由Gateway Design Automation公司(后被Cadence收购)开发，1995年成为IEEE标准

HDL主要特征

- **HDL语言既包含一些高级程序设计语言的结构形式，同时也兼顾描述硬件线路连接的具体构件**
- **通过使用结构级或行为级描述可以在不同的抽象层次描述设计**
  - 五个抽象层次：系统级、算法级、寄存器传输级、逻辑（门）级、电路（开关）级
- **HDL语言是并发的，即具有在同一时刻执行多个任务的能力**
- **HDL语言有时序的概念**

HDL抽象层次

抽象层次	行为	结构	物理
系统级	性能描述	部件定义及其之间的逻辑连接形式	芯片、模块、电路板、子系统
算法级	接口应答算法	硬件模块、数据结构	部件、电路板之间的连接
寄存器传输级	寄存器操作状态表	部件定义及连接	芯片、宏单元
逻辑级	布尔方程描述	门电路、触发器、锁存器	标准单元布局图
电路级	微分方程表达	晶体管、电阻、电容、电感元件	晶体管布局图

Verilog HDL和VHDL的比较

- **Verilog HDL和VHDL均为IEEE标准，在大多数情况下，两者基本相同**
- **Verilog HDL和C语言的风格相似，如果有C语言的基础，比较容易入门，而VHDL来源于Ada语言，需要一定的专业培训**
- **一般认为 Verilog HDL在系统级抽象方面要比 VHDL 略差一些，而在门级和电路级描述方面要强得多**

Verilog HDL与 C语言的比较

- **虽然Verilog的某些语法与C语言接近，但存在本质上的区别**
  - Verilog是一种硬件语言，最终是为了产生实际的硬件电路或对硬件电路进行仿真
  - C语言是一种软件语言，是控制硬件来实现某些功能
- **利用Verilog编程时，要时刻记着：Verilog是硬件语言，要将其与硬件电路对应起来**

C	Verilog
procedures	modules
variables	wires/regs
parameters	ports
control (if,case,?:)	control (if,case,?:)

Verilog HDL基本语法

- **空白符（间隔符）**
  - 主要起分隔文本的作用，可以使文本错落有致，便于阅读与修改
  - 包括空格、制表符 、换行符及换页符
- **注释符**
  - 改善程序的可读性，在编译时不起作用
  - 多行注释符：以/*开始到*/结束
  - 单行注释符：以//开始到行尾结束

Verilog HDL基本语法 (续1)

- **关键字**
  - Verilog语言内部已经使用的词，例如，module、endmodule、input、output、wire、reg、and等
  - **字母都是小写**，例如Input不是关键字
- **标识符**
  - 用于对象（如模块名、电路的输入与输出端口、变量等）命名
  - 以字母或下划线“_”开始，字母、数字、符号“$”和下划线的组合
  - **字母大小敏感**，如Input，iNput可以是不同的标识符
  - 不能与关键词相同

Verilog HDL基本语法 (续2)

- **逻辑值集合**
  - 0：低电平、逻辑0或“假”
  - 1：高电平、逻辑1或“真”
  - x/X：不确定的值（未知状态）
  - z/Z：高阻态
- **常量与符号常量**

常量

- **整数型**
  - 十进制数形式表示，例如，30、－2
  - 带基数形式表示，格式为：
    <+/－> <位宽>’ <基数符号> <数值>
  - 基数符号：十进制 D/d，二进制 B/b，八进制 O/o，十六进制 H/h
  - 例如，–8’d101、5’o37、8’HeD，8’b1001_001x
- **实数型常量**
  - 十进制记数法，例如，0.1、2.0、5.67
  - 科学记数法，例如，23_5.1e2、5E－4

符号常量

- **用参数定义语句定义一个标识符来代表一个常量**
  - **常用来定义变量的位宽及延时等**
- **定义格式**

  parameter 参数名1＝常量表达式1，参数名2＝常量表达式2，……；

  例如：parameter BIT=1, BYTE=8, PI=3.14;

变量数据类型

- **线网(net)型 ：表示元件之间的物理连线**
  - 输出值紧随输入值的变化而变化
  - 最常用类型是wire
- **寄存器(register)型 ：表示抽象存储元件**
  - 在赋新值以前保持原值
  - 只能在initial或always语句中被赋值
  - 最常用类型是reg
- **定义格式**

  wire/reg [MSB:LSB] 变量名1，…，变量名n;

  例如： wire a, b; reg[3:0] state;

Verilog HDL程序基本结构

- **由实现特定功能的模块构成**

module 模块名 (端口名1, 端口名2, …);

端口类型说明(input, outout, inout);
参数定义(可选);
数据类型定义(wire, reg等);
} **说明部分**

实例化低层模块和基本门级元件;
连续赋值语句（assign）;
过程块结构（initial和always）
　行为描述语句;
} **功能描述部分 顺序是任意的**

endmodule

VerilogHDL描述组合逻辑电路

- **组合逻辑电路的门级描述**
  - 使用内置的基本门级元件描述
- **组合逻辑电路的数据流描述**
  - 使用连续赋值assign语句描述
- **组合逻辑电路的行为级描述**
  - 使用always结构描述

基本门级元件

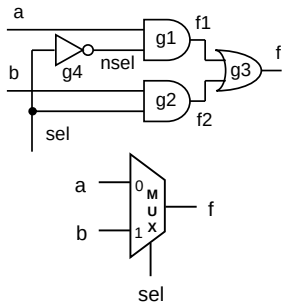
元件符号	功能说明	元件符号	功能说明
and	多输入端与门	nand	多输入端与非门
or	多输入端或门	nor	多输入端或非门
xor	多输入端异或门	xnor	多输入端异或非门
buf	多输出端缓冲器	not	多输出端反相器
bufif1	高电平有效三态缓冲器	notif1	高电平有效的三态反相器
bufif0	低电平有效三态缓冲器	notif0	低电平有效的三态反相器

示例— Mux with Primitives

```
module mux(f, a, b, sel);
output f;
input a, b, sel;

and   g1(f1, a, nsel),
      g2(f2, b, sel);
or    g3(f, f1, f2);
not   g4(nsel, sel);

endmodule
```



赋值语句

- 连续赋值语句

  assign 变量名= 赋值表达式

  – 只能对线网型变量进行赋值，不能对寄存器型变量进行赋值
  – 仅用于描述组合逻辑
- 过程赋值语句

  变量名= 赋值表达式

  – 只能对寄存器数据类型的变量赋值
  – 在always和initial语句内的赋值
  – 可用于描述组合和时序逻辑

Verilog HDL运算符

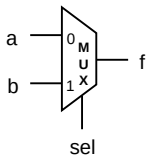
类型	符号	功能说明	类型	符号	功能说明
算术运算符	+	二进制加	关系运算符	>	大于
	-	二进制减		<	小于
	*	二进制乘		>=	大于或等于
	/	二进制除		<=	小于或等于
	%	求模		==	等于
				!=	不等于
位运算符	~	按位取反	缩位运算符	&	缩位与
	&	按位与		~&	缩位与非
	|	按位或		|	缩位或
	^	按位异或		~|	缩位或非
	^~ 或 ~^	按位同或		^	缩位异或
				^~ 或 ~^	缩位同或
逻辑运算符	!	逻辑非	移位运算符	>>	右移
	&&	逻辑与		<<	左移
	||	逻辑或			

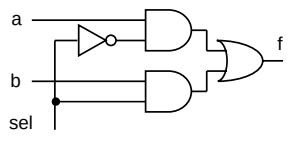
示例— Mux with Assign

```
module mux(f, a, b, sel);
output f;
input a, b, sel;

assign f = (a & ~sel) | (b & sel);

endmodule
```





if条件语句

- 表达式一般为逻辑表达式或关系表达式
  – 系统对表达式的值进行判断，若为0，x，z，按“假”处理；若为1，则按“真”处理，执行指定语句
- if和else后可包含单个或多个语句，多句时用begin-end块语句括起来
- if语句嵌套使用时，注意if与else的配对关系

```
if（表达式） 语句1;
if（表达式） 语句1;
   else  语句2;
if（表达式1） 语句1;
   else if（表达式2）语句2;
   else if（表达式3）语句3;
        ……
   else if（表达式n）语句n;
   else  语句n+1;
```

Case条件语句

- **case （敏感表达式）**
  值1：语句1；
  值2：语句2；
  ……
  值n：语句n；
  default： 语句n+1；
  endcase

条件语句使用要点

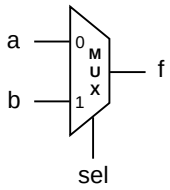
- **描述组合电路时，应注意列出所有条件分支，否则编译器认为条件不满足时，会引进一个锁存器保持原值，产生时序电路而非组合电路**
- **每个变量至少有4种取值，为包含所有分支，可在if语句后加上 else；在 case语句后加上 default**

示例— Mux with Always（If)

```
module mux(f, a, b, sel);
output f;
input a, b, sel;
reg f;

always @(a or b or sel)
 if (sel) f = b;
 else f = a;

endmodule
```

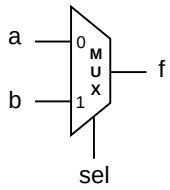


示例— Mux with Always（Case)

```
module mux(f, a, b, sel);
output f;
input a, b, sel;
reg f;

always @(a or b or sel)
  case(sel)
    1'b0 : f = a;
    1'b1 : f = b;
  endcase

endmodule
```



示例— BCD-七段译码

```
module   decode4_7(abcdefg，bcd);
output [6:0] abcdefg;
input [3:0]   bcd;
reg [6:0] abcdefg;
always  @  (bcd)  begin
   case (bcd)
     4'd0: abcdefg=7'b1111110;
     4'd1: abcdefg=7'b0110000;
     4'd2: abcdefg=7'b1101101;
     4'd3: abcdefg=7'b1111001;
```


示例— BCD-七段译码(续)

```
     4'd4: abcdefg=7'b0110011;
     4'd5: abcdefg=7'b1011011;
     4'd6: abcdefg=7'b1011111;
     4'd7: abcdefg=7'b1110000;
     4'd8: abcdefg=7'b1111111;
     4'd9: abcdefg=7'b1111011;
     default : abcdefg=7'bx;
   endcase
end
endmodule
```