Verilog & 数字逻辑基础概念全面汇总

一、数制与编码

1. 常用数制

- 二进制、八进制、十六进制
- BCD码（8421码、余3码等）
- 格雷码（循环码）：相邻码只有一位变化

2. 带符号数表示

- **原码**：最高位为符号位，0正1负
- **反码**：正数同原码，负数除符号位取反
- **补码**（最常用）：
  - 正数：同原码
  - 负数：反码+1
  - n位补码范围：$[-2^{n-1}, 2^{n-1}-1]$
  - 运算方便，0的表示唯一

3. 溢出与进位

- **溢出（Overflow）**：有符号数运算结果超出表示范围
  - 判断：$C_{n-1} \oplus C_n$（次高位与最高位进位异或）
  - 或：两正数得负，两负数得正
- **进位（Carry）**：无符号数运算超出范围
- 两者独立，可同时发生

二、逻辑代数基础

1. 三大规则

- **代入规则**：等式中的变量可用函数代替
- **反演规则**：求反函数，运算符互换，变量取反

$$F = A \cdot B + C \rightarrow \overline{F} = (\overline{A} + \overline{B}) \cdot \overline{C}$$

- **对偶规则**：运算符互换，0/1互换，变量不变
  - 若等式成立，则对偶式也成立

2. 逻辑门类型

- 基本门：NOT、AND、OR
- 通用门：NAND、NOR（可单独实现任何逻辑）
- 异或门：

$$A \oplus B = A\overline{B} + \overline{A}B$$

- 同或门：

$$A \odot B = AB + \overline{A}\overline{B}$$

3. 逻辑函数化简

- 代数化简法
- **卡诺图法**（≤4变量）
- 奎因-麦克拉斯基法（Q-M法，多变量系统）

4. 与或式和或与式

(1) 与或式（Sum of Products, SOP）

定义

- **形式**：若干个**乘积项（与项）** 的**逻辑或**
- **表达式**：

$$F = AB + \overline{A}C + BC$$

- **特点**：
  - 先与（AND）后或（OR）
  - 每个乘积项是若干个原变量或反变量的与
  - 对应卡诺图中**圈1**得到的表达式

标准形式

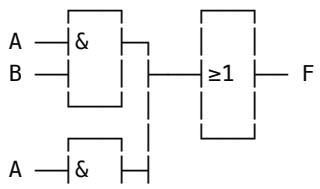
- **最小项之和（标准SOP）**：
  - 每个乘积项包含所有变量（原变量或反变量）
  - 例：3变量函数

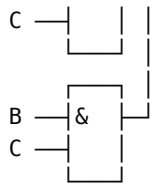
$$F = \overline{A}BC + A\overline{B}C + ABC$$

  - 用最小项表示：

$$F = \sum m(3,5,7)$$

硬件实现





- 第一级：与门（实现乘积项）
- 第二级：或门（实现和）

(2) 或与式（Product of Sums, POS）

定义

- **形式**：若干个**和项（或项）** 的**逻辑与**
- **表达式**：

$$F = (A + B)(\overline{A} + C)(B + C)$$

- **特点**：
  - 先或（OR）后与（AND）
  - 每个和项是若干个原变量或反变量的或
  - 对应卡诺图中**圈0**得到的表达式（再取反）

标准形式

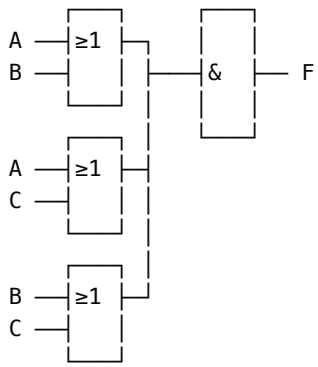
- **最大项之积（标准POS）**：
  - 每个和项包含所有变量（原变量或反变量）
  - 例：3变量函数

$$F = (A + B + C)(\overline{A} + B + C)(A + \overline{B} + C)$$

  - 用最大项表示：

$$F = \prod M(0, 1, 2)$$

硬件实现



- 第一级：或门（实现和项）
- 第二级：与门（实现积）

(3) SOP与POS的关系

对偶关系

- 若 F 的SOP式为：

$$F = AB + \overline{A}C$$

- 则 F 的POS式可通过**对偶+取反**得到：
  i. 求对偶：

$$F_d = (A + B)(\overline{A} + C)$$

  ii. 再取反得POS：实际上直接圈0更简单

转换方法

1. **代数法**：
   - SOP→POS：使用德摩根定律

$$F = AB + \overline{A}C \Rightarrow \overline{F} = \overline{AB} \cdot \overline{\overline{A}C} = (\overline{A} + \overline{B})(A + \overline{C})$$

   - POS→SOP：同样用德摩根
2. **卡诺图法**：
   - SOP：圈**1**，写乘积项
   - POS：圈**0**，写和项，然后对整个表达式取反（或直接写原函数的POS）

一般原则：

1. **输出多为1时**：用POS更简（圈0的圈少）
2. **输出多为0时**：用SOP更简（圈1的圈少）
3. **PLD实现**：传统PLD基于与或阵列，适合SOP
4. **FPGA实现**：基于LUT，两者无差异

三、组合逻辑电路

1. 常见模块

- **编码器**（Encoder）：$2^n \to n$
  - 优先编码器：输入可多路有效
- **译码器**（Decoder）：$n \to 2^n$
  - 可用于实现逻辑函数
- **数据选择器（MUX）**：2^n选1
  - 实现逻辑函数：函数输入接地址端，真值接数据端
- **数据分配器（DEMUX）**：$1 \to 2^n$
- **比较器**：数值比较
- **加法器**：

- 半加器：不考虑进位
- 全加器：考虑进位
- 超前进位加法器：提高速度

2. 竞争与冒险

竞争（Race）

- 定义：信号经不同路径到达同一点的时间差
- **逻辑竞争**：可能不影响功能
- **功能竞争**：导致错误状态变化

冒险（Hazard）

- **静态冒险**：输出应有不变，但出现毛刺
  - 静态0冒险：应从0→0，出现0→1→0
  - 静态1冒险：应从1→1，出现1→0→1
- **动态冒险**：输出变化时出现多次跳变（0→1→0→1）

消除方法

- 增加冗余项（卡诺图相邻圈相切时）
- 输出加滤波电容（硬件）
- 插入寄存器（同步设计）
- 格雷码计数避免多bit同时变化

四、时序逻辑电路

1. 存储元件

锁存器（Latch）

- 电平敏感
- **RS锁存器**：
  - NOR型：高电平有效，S=1,R=1禁止
  - $$Q_{n+1} = S + \overline{R}Q_n$$
  - 约束条件：$RS = 0$
  - NAND型：低电平有效，S'=0,R'=0禁止
  - $$Q_{n+1} = \overline{S} + RQ_n$$
  - 约束条件：$R + S = 1$
- **D锁存器**：使能期间透明

触发器（Flip-Flop）

- 边沿敏感（时钟端有三角符号）
- **D触发器**：

$$Q_{n+1} = D$$

- **JK触发器**：

$$Q_{n+1} = J\overline{Q_n} + \overline{K}Q_n$$

  - J=K=0：保持；J=K=1：翻转
  - J=K=T：T触发器：

$$Q_{n+1} = T \oplus Q_n$$

- **T触发器**：翻转控制
- **SR触发器**：避免S=R=1

2. 时序电路类型

- **同步时序电路**：统一时钟控制
- **异步时序电路**：无统一时钟或时钟不同步

3. 有限状态机（FSM）

Moore型

- 输出仅取决于当前状态
- 输出写在状态圈内
- 延迟较大，抗干扰好

Mealy型

- 输出取决于当前状态和输入
- 输出写在转移边上
- 响应快，状态数少

设计步骤

1. 状态图/表
2. 状态编码（二进制、格雷码、独热码）
3. 求激励方程（使用触发器特性方程）
4. 画逻辑图

五、Verilog HDL基础

1. 模块结构

```
module 模块名(
    input  wire/reg ...,
    output wire/reg ...,
    inout  wire ...
);
    // 内部声明
    // 功能描述
endmodule
```

2. 数据类型

- **wire**：连线型，用于连续赋值和模块互连
- **reg**：寄存器型，用于过程赋值
  - 不一定是硬件寄存器！取决于使用方式
- **parameter**：参数，常量
- **integer**：整数，常用于循环

3. 赋值方式

连续赋值（组合逻辑）

```
assign out = a & b;  // wire类型
```

- 实时响应输入变化
- 左侧必须是wire

过程赋值（时序/组合逻辑）

```
always @(posedge clk)  // 时序逻辑
always @(*)            // 组合逻辑
```

- 阻塞赋值（ = ）：顺序执行，用于组合逻辑
- 非阻塞赋值（ <= ）：并行执行，用于时序逻辑

4. always块规则

组合逻辑

```
always @(*) begin
    // 所有分支必须赋值，否则产生锁存器!
    if(cond) out = a;
    else out = b;
end
```

- 敏感列表用 `@(*)` 或 `@(a, b, ...)`
- 所有条件分支必须赋值

时序逻辑

```
always @(posedge clk or negedge rst_n) begin
    if(!rst_n) q <= 0;
    else q <= d;
end
```

- 边沿敏感列表
- 常用非阻塞赋值

5. 按位vs逻辑运算符

运算符	类型	操作数	结果位宽	功能
&	按位与	多位	操作数位宽	逐位进行与运算
&&	逻辑与	标量	1位	布尔逻辑与运算

示例对比

```
wire [3:0] a = 4'b1101;
wire [3:0] b = 4'b1011;
wire [3:0] c;
wire       d;

// 按位与：逐位运算
assign c = a & b;  // c = 4'b1001 (1101 & 1011 = 1001)

// 逻辑与：转换为布尔值后运算
assign d = a && b; // d = 1'b1 (a≠0 且 b≠0 → true)
```

特殊情况处理

```
wire [3:0] x = 4'b0x01;  // 包含x
wire [3:0] y = 4'b1011;

assign bit_and = x & y;   // 结果：4'b0x01 (按位：0&1=0, x&0=0, 0&1=0, 1&1=1)
assign log_and = x && y;  // 结果：1'bx  (x包含未知→布尔值不确定)
```

6. 赋值vs相等运算符

运算符	类型	使用场景	功能
=	阻塞赋值	组合逻辑/initial块	立即赋值，顺序执行
<=	非阻塞赋值	时序逻辑always块	同步赋值，并行执行
==	逻辑相等	条件判断	比较，结果1位（0/1/x）

运算符	类型	使用场景	功能
===	全等比较	条件判断	比较，包括x/z状态

赋值运算符详解

```
// = 阻塞赋值（顺序执行）
always @(*) begin
    a = b;  // 立即执行
    c = a;  // 使用更新后的a值
end
// 等效于：c = b;

// <= 非阻塞赋值（并行执行）
always @(posedge clk) begin
    a <= b;  // 记录b的当前值
    c <= a;  // 记录a的旧值（不是b！）
end
// 结果：c得到a上一个时钟周期的值
```

相等运算符详解

```
// == 逻辑相等（4态逻辑：0,1,x,z）
wire [1:0] p = 2'b1x;
wire [1:0] q = 2'b10;

wire eq1 = (p == q);  // 结果：1'bx（因为x与0比较不确定）

// === 全等比较（匹配所有状态，包括x,z）
wire eq2 = (p === q); // 结果：1'b0（1x不全等于10）
```

== 的真值表（以1位为例）

a	b	a==b
0	0	1
0	1	0
1	0	0
1	1	1
0	x	x
x	x	x
0	z	x
z	z	x

=== 的真值表

a	b	a===b
0	0	1
0	1	0
1	0	0
1	1	1
0	x	0
x	x	1
0	z	0
z	z	1
x	z	0

常见错误示例

```
// 错误1：组合逻辑中错误使用非阻塞赋值
always @(*) begin
    a <= b;  // 应使用 =，否则可能产生锁存器
    c <= a;
end

// 错误2：在时序逻辑中混合使用赋值类型
always @(posedge clk) begin
    a = b;     // 阻塞赋值，不推荐
    c <= a;    // 非阻塞赋值
end

// 错误3：将 == 写成 =
always @(*) begin
    if (a = 1'b1)  // 这是赋值！应改为 ==
        out = b;
end

// 错误4：对多位数据使用逻辑运算符
wire [3:0] data = 4'b0011;
wire result = data && 1'b1;  // 意图不明，应该用按位&或直接判断
```

黄金规则

1. **组合逻辑**：用 `=`（阻塞赋值）
2. **时序逻辑**：用 `<=`（非阻塞赋值）
3. **位运算**：用 `&`、`|`、`^`
4. **条件判断**：用 `&&`、`||`、`==`
5. **测试平台**：可用 `===` 检查x/z状态
6. **always块类型**：
   - `always @(*)` → 用 `=`
   - `always @(posedge clk)` → 用 `<=`

记忆口诀

> **"单字按位，双字逻辑"**
> & 、 | 、 ^ → 按位运算
> && 、 || → 逻辑运算
>
> **"等号家族"**
> = → 立即赋值
> <= → 同步赋值
> == → 模糊比较（可处理x）
> === → 精确比较（仿真专用）

7. 其他常见错误

产生意外锁存器

```
// 错误：缺少else分支
always @(*) begin
    if(en) out = a;
    // 无else → 产生锁存器!
end

// 正确：默认赋值
always @(*) begin
    out = 1'b0;  // 默认值
    if(en) out = a;
end
```

多驱动冲突

```
// 错误：两个always块驱动同一信号
always @(*) out = a;
always @(*) out = b;  // 冲突!
```

六、时序分析关键概念

1. 关键路径（Critical Path）

- 定义：寄存器到寄存器之间延迟最长的路径
- 决定系统最高时钟频率
- 优化方法：流水线、逻辑重构、调整扇出

2. 时序参数

- **建立时间（Tsu）**：数据在时钟沿前必须稳定的时间
- **保持时间（Th）**：数据在时钟沿后必须保持的时间

- **时钟到输出时间（Tco）**：时钟沿后输出有效的时间
- **时钟偏移（Skew）**：时钟到达不同触发器的时间差

3. 时序约束方程

最小时钟周期

$$T_{clk} \geq T_{co} + T_{logic\_max} + T_{su} - T_{skew}$$

保持时间检查

$$T_{co\_min} + T_{logic\_min} \geq T_h + T_{skew}$$

七、存储器与可编程逻辑

1. 存储器类型

- **ROM**：只读存储器
  - 容量表示：字数×位数
  - 地址位数 = $\log_2$(字数)
- **RAM**：随机存取存储器
  - SRAM：静态，速度快
  - DRAM：动态，容量大
- **FIFO**：先进先出队列
- **LIFO/Stack**：后进先出堆栈

2. 可编程逻辑器件

- **PLD**：可编程逻辑器件
- **CPLD**：复杂PLD，与或阵列结构
- **FPGA**：现场可编程门阵列
  - 基于LUT（查找表）
  - 可重构逻辑

八、综合与验证概念

1. 综合（Synthesis）

- 将HDL转换为门级网表
- 优化目标：面积、速度、功耗

2. 仿真（Simulation）

- **功能仿真**：验证逻辑正确性

- **时序仿真**：包含延迟信息
- **测试平台（Testbench）**：

```
initial begin
    clk = 0;
    rst_n = 0;
    #10 rst_n = 1;
    #100 $finish;
end
always #5 clk = ~clk;
```

3. 形式验证

- 等价性检查
- 模型检查

九、设计原则与技巧

1. 同步设计原则

- 使用全局时钟
- 避免异步反馈
- 时钟域交叉用同步器（两级触发器）

2. 复位策略

- **同步复位**：在时钟控制下复位
- **异步复位**：立即复位，需考虑复位恢复时间
- **推荐**：异步复位，同步释放

3. 代码风格

- 模块化设计
- 参数化设计
- 注释充分
- 统一命名规范

4. 避免亚稳态

- 跨时钟域用同步器
- 满足建立/保持时间
- 异步信号打两拍

5. timescale

(1) 基本语法

```
`timescale <time_unit> / <time_precision>
```

- **时间单位（time_unit）**：仿真中时间和延迟的基本单位
- **时间精度（time_precision）**：仿真器能处理的最小时间单位
- 必须在模块外部声明（通常在所有模块之前）

(2) 常用值

```
`timescale 1ns / 1ps    // 1纳秒单位，1皮秒精度
`timescale 100ps / 10ps  // 100皮秒单位，10皮秒精度
`timescale 1us / 1ns     // 1微秒单位，1纳秒精度
```

(3) 作用与规则

- **延迟控制**：

```
  #5;          // 延迟5个时间单位（如5ns）
  #3.14159;    // 延迟3.14159个时间单位，精度舍入
```

- **精度舍入**：
  - 如果 `#3.14159` 在 `1ns/1ps` 下，实际延迟=3.142ns（四舍五入到1ps精度）
  - 如果 `#3.14159` 在 `1ns/100ps` 下，实际延迟=3.1ns（四舍五入到100ps）
- **波形显示精度**：影响波形查看器显示的时间分辨率

(4) 重要规则

1. **编译单位唯一性**：同一个编译单元（通常是整个文件）只能有一个 `timescale`
2. **精度决定舍入**：所有延迟都按精度进行舍入
3. **影响仿真速度**：精度越高，仿真越慢
4. **默认值**：如未指定，仿真器使用默认设置（通常为 `1ns/1ns` ）

(5) 示例

```
`timescale 1ns / 10ps  // 1ns单位，10ps精度

module test;
  reg clk;

  initial begin
    clk = 0;
    forever #5 clk = ~clk;  // 每5ns翻转一次
  end

  initial begin
    #3.14159;  // 实际延迟=3.14ns（舍入到10ps）
    $display("Time is: %t", $time);
  end
endmodule
```

(6) 常见问题

- **精度不足**： `timescale 1ns/1ns` 时， `#0.5` 会被舍入为 `#0` 或 `#1`
- **多文件冲突**：不同文件有不同 `timescale` 时，以最后一个编译的为准
- **测试平台精度**：测试平台精度应≥设计精度，否则可能错过边沿

(7) 扩展： `$timeformat` 函数

与 `timescale` 配合使用，控制时间显示格式：

```
// -9表示纳秒，3表示小数点后3位，'ns'是单位后缀
$timeformat(-9, 3, " ns", 10);

// 显示：Time = 123.456 ns
$display("Time = %t", $realtime);
```

十、常见问题与解决

1. 毛刺消除

毛刺产生原因

- **竞争**：信号经不同路径到达同一点的时间差
- **逻辑冒险**：输入变化时，输出短暂出现非期望脉冲
- **函数冒险**：多输入同时变化导致

消除方法（按优先级）

（1） 同步设计（最有效）

```
// 毛刺信号加寄存器同步
always @(posedge clk) begin
    glitchy_signal_reg <= glitchy_comb;  // 毛刺被时钟过滤
    clean_signal <= glitchy_signal_reg;  // 两级更稳
end
```

（2）增加冗余项（卡诺图法）

- **原理**：覆盖相邻但不相切的1圈
- **例**：$F = AB + \overline{A}C$ 有毛刺（B=C=1，A变化时）
- **加冗余项**：$F = AB + \overline{A}C + BC$ 消除毛刺

（3） 格雷码计数

```
// 普通二进制计数：多bit同时变化 → 毛刺
// 格雷码计数：每次只有1bit变化 → 无毛刺
assign gray_code = binary_code ^ (binary_code >> 1);
```

输出滤波（硬件）

- 加小电容（低频电路）
- 施密特触发器整形

时序约束

- 用 `set_false_path` 忽略异步路径
- 多周期路径约束

2. 时序违例

产生原因

电路不满足建立时间（Setup Time） 或保持时间（Hold Time） 要求，导致寄存器采样错误。

（1） 建立时间违例（Setup Time Violation）

- **现象**：数据在时钟沿前**未稳定足够时间**
- **公式**：

$$T_{clk} \geq T_{co} + T_{logic} + T_{su} - T_{skew}$$

- **影响**：采到错误数据（前一个值或亚稳态）
- **解决方案**：

```
// 1. 降低时钟频率
// 2. 插入流水线
always @(posedge clk) begin
    stage1 <= input_data;   // 一级
    stage2 <= stage1;       // 二级
end
// 3. 优化组合逻辑（减少T_logic）
```

（2） 保持时间违例（Hold Time Violation）

- **现象**：数据在时钟沿后**未能保持足够时间**
- **公式**：

$$T_{co\_min} + T_{logic\_min} \geq T_h + T_{skew}$$

- **影响**：采到新数据（本该采旧值）
- **解决方案**：

```
// 1. 增加缓冲延迟（插入buffer）
// 2. 调整时钟树（控制skew）
// 3. 使用慢速寄存器（增加T_co_min）
```

解决方法

- 插入流水线

- 降低时钟频率
- 逻辑优化
- 调整约束

3. 面积优化

- 资源共享
- 状态编码优化（如用独热码）
- 消除冗余逻辑

总结表：触发器与锁存器对比

特性	触发器（Flip-Flop）	锁存器（Latch）
触发方式	边沿触发	电平触发
时钟符号	有三角▶	无三角
数据透明期	时钟边沿瞬间	使能有效期间
抗干扰性	好	差
时序控制	容易	困难
Verilog代码	`always @(posedge clk)`	`always @(*)` 且分支不全
用途	同步时序电路	异步接口、门控时钟