5.3 组合逻辑

组合逻辑一般可以分成三类：门级逻辑、选择器和比较器、运算逻辑等。

5.2.1 组合逻辑 verilog 代码

Verilog 的程序代码，组合逻辑的代码结构如下：

always@(*) begin

代码语句；

end

或者是 assign 代码语句；

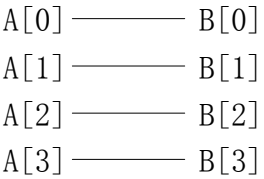
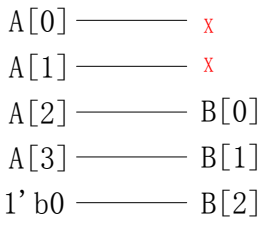
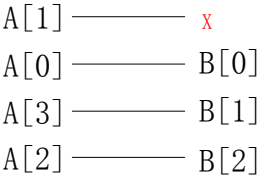
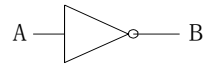
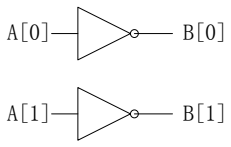
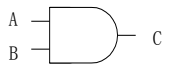
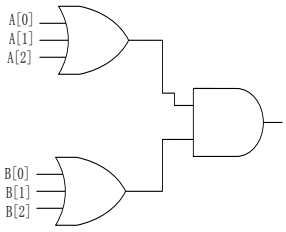
代码中@后是敏感列表，当括号内的条件满足时，就执行一次。括号内的“*”，表示代码语句中所有信号有变化。结合起来就是：当代码语句中的信号有变化时，就执行一次。这与组合逻辑电路的功能是对应的。组合逻辑也是信号变化，输出立刻变化。

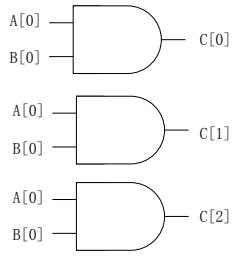
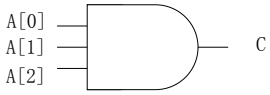
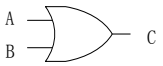
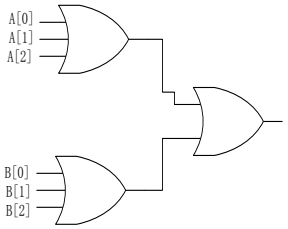
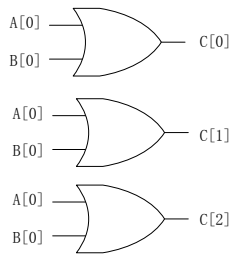
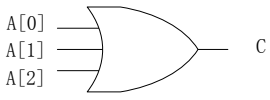
有很多代码的敏感列表里列出所有信号，如 always@(a or b or c)，建议不要这样写，容易把信号给遗漏，从而导致不希望的电路出现。

明德扬建议初学者前面章节中不要用 assign 语句，这样全部代码都是 always 结构。

5.2.2 门级逻辑

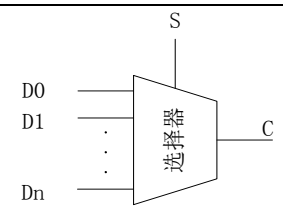
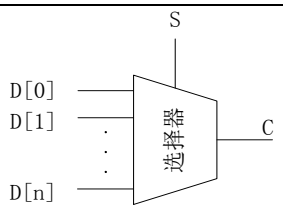
分类	情况	功能	代码	电路示意图	备注

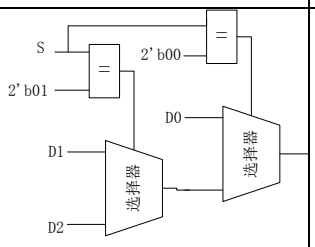
连线	1 位相连	将两根线连接	reg A,B; always@(*)begin B=A; end		
	多位相连	将两端的线一对一连接	reg[3:0] A,B; always@(*)begin B=A; end		当 A 和 B 是 n 位时，实际就有 n 根线。但为看图方便，我们通常只画出一根线表示
	移位 (右移：>> 左移：<<)	A 向右或向左移动后，赋值给 B	reg[3:0] A; reg[2:0] B; always@(*)begin B=A>>2; end		移位操作，实际上是选哪线相连。
	拼接 (符号：{})	将大括号内的内，按位置一对一连接	reg[3:0] A; reg[2:0] B; always@(*)begin B={A[2],A[3],A[0]}; end		拼接，实际上是选哪线相连。
反相器	1 位反相器 （符号：~）	将值取反	reg A,B; always@(*)begin B=~A; end		
	多位反相器 （符号：~）	将值取反	reg[1:0] A,B; always@(*)begin B=~A; end		如果 A 和 B 都是 n 位，实际电路就是有 n 个反相器。画电路图时可画一个来简化。
与门	1 位逻辑与 （符号： &&）	A 和 B 都为 1，C 为 1；否则 C 为 0。	reg A,B; always@(*)begin C=A&&B; end		注意：FPGA 支持多输入的与门，例如四输入与门，输入可为 ABCD，输出为 E，当 ABCD 同时为 1 时，E 为 1
	多位逻辑与 （符号： &&）	A 或 B 都不为 0 时，C 为 1，否则为 0。	reg[2:0] A，B，C; always@(*)begin C=A&&B; end		多位信号之间的逻辑与，很容易引起歧义，设计最好不要用多位数的逻辑与。如果要实现上面功能，建议代码改为如下： always@(*)begin C=(A!=0)&&(B!=0); end

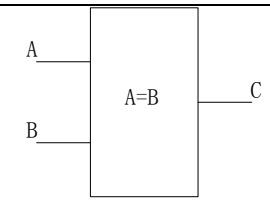
	按位与（符号：&）常用 1	A 和 B 对应的比特分别相与。	reg[2:0] A，B，C; always@(*)begin C=A&B; end		
	按位与（符号：&）常用 2	A 的各位之间相与。	reg[2:0] A; always@(*)begin C=&A; end		
或门	1 位逻辑或（符号：||）	A 和 B 其中 1 个为 1，C 为 1；否则 C 为 0。	reg A,B; always@(*)begin C=A||B; end		注意：FPGA 支持多输入的与门，例如四输入与门，输入可为 ABCD，输出为 E，当 ABCD 同时为 1 时，E 为 1
	多位逻辑或（符号：||）	A 和 B 其中 1 个非 0，C 为 1；否则 C 为 0。	reg[2:0] A，B，C; always@(*)begin C=A||B; end		多位信号之间的逻辑或，很容易引起歧义。最好不要用多位的逻辑或。如果要实现相同功能，建议改为如下： always@(*)begin C=(A!=0)&&(B!=0); end
	按位或（符号：|）常用 1	A 和 B 对应的比特相或。	reg[2:0] A，B，C; always@(*)begin C=A&B; end		
	按位或（符号：||）常用 2	A 的各位之间相或	reg[2:0] A; always@(*)begin C=&A; end		

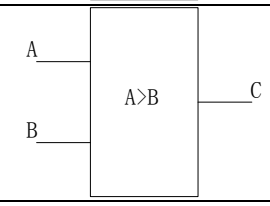
此外还有同或门、与非门、或非门等等，但画电路图时比较少用。同学们需要记清楚上述电路示意图，这是后面画图练习的基础。

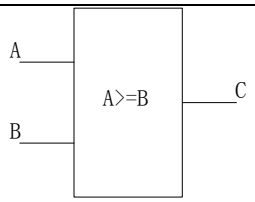
5.2.3 选择器和比较器

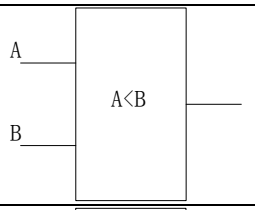
分类	情况	功能	代码	电路示意图	备注
选择器	常见形式 1	通过 S，选择输入给输出 C。	always@(*)begin case(S) 2'b00 : C=D0; 2'b01 : C=D1; 2'b10 : C=D2; default : C= D3; endcase		
	常见形式 2	通过 S，选择输入给输出 C。	always@(*)begin C = D[S]; endcase		此时代码亦非常常用，其本质也是选择器。

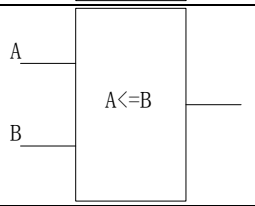
	用 if else	通过判断 s 来选择。	always@(*)begin
			if(S==0)
			C=D0;
			else if(S==2'b01)
			C=D1;
			else
			C=D2;
			end
					请注意此处用到了相等比较器和选择器。请掌握这种 if else 代码的画法。

比较器	相等	相等则为 1	always@(*)begin
			if(A==B)
			C=1;
			else
			C=0;
			end
					

	大于	大于则为 1	always@(*)begin
			if(A>B)
			C=1;
			else
			C=0;
			end
					

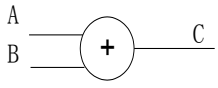
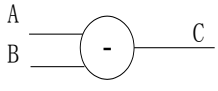
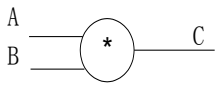
	大于等于	大于等于则为 1	always@(*)begin
			if(A>=B)
			C=1;
			else
			C=0;
			end
					

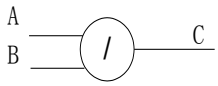
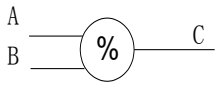
	小于	小于则为 1	always@(*)begin
			if(A<B)
			C=1;
			else
			C=0;
			end
					

	小于等于	小于等于则为 1	always@(*)begin
			if(A<=B)
			C=1;
			else
			C=0;
			end
					

5.2.4 运算逻辑

日常生活中，最常用的运算逻辑是加、减、乘、除、求余。FPGA 也有对应的加(+)、减(-)、乘(*)、除(/)、求余(%)的运算符。

分类	功能	代码	电路示意图	备注
加法器	两数相加	always@(*)begin C =A+B; endcase		本质上，运算逻辑都是由与门、或门等门逻辑搭建起来的电路，例如 1 位的加法就是 S= A ^B;Cout= A&&B。
减法器	两数相减	always@(*)begin C =A-B; endcase		
乘法器	两数相乘	always@(*)begin C =A*B; endcase		在二进制运算中，乘法运算实质上就是加法运算，例如 1111*111 = （1111） + （11110） +（111100）。所以乘法器会比加法器消耗的资源多。

除法器	两数相除	always@(*)begin C =A/B; endcase		二进制运算中，除法和求余涉及到加法、减法和移位等运算，所以除法和求余电路资源都非常大，在设计时要尽力避免除法和求余。如果一定要用到除法，尽量让除数为 2 的 n 次方，如 2，4，8，16 等。因为 a/2 实质就是 a 向右移 1 位；a/4 实质就是 a 向右移 2 位。移位运算是不消耗资源的。
求余器	两数求余	always@(*)begin C =A%B; endcase		

更加复杂的运算逻辑，如 LOG、SIN、COS 等，是没有直接电路对应的，一般是调 IP 核，或者自己设计。

总结

1. 电路示意图是画电路的基础，同学们务必能根据代码画出电路示意图。

2. 注意如果信号是多位的，但示意图一般画 1 位，但我们应该清楚示意图背后表示的位数。位数越多，其资源肯定越大。

3. 代码和电路是一一对应的。我们编写的代码，一定要求有电路对应，没有电路对应的代码，即使编写出来也没用。

4. 以上是列出的常用电路，是可以直接综合的。我们就是以上面电路为基础，像搭积木一样，搭建起整个项目。