

### 1、用自己的话总结 **always** 语句和 **assign** 语句的区别

**assign** 语句用于描述组合逻辑。它是一种连续赋值语句，意味着它会在模块被实例化时立即执行，并且当其右侧表达式中的任何信号发生变化时，它会自动更新左侧的信号。这种语句不依赖于时钟信号，因此它描述的是信号之间的即时关系。

**always** 语句用于描述组合逻辑和时序逻辑。它是一个过程赋值语句，包含一个敏感列表（sensitivity list）和一系列的行为语句。敏感列表定义了哪些信号的变化会触发 **always** 块内的代码执行。对于组合逻辑，通常会使用 **always @(\*)**，这样任何输入信号的变化都会触发块内的代码。而对于时序逻辑，敏感列表通常包含时钟信号，如 **always @(posedge clk)**，表示在时钟信号的上升沿触发。

总结来说，**assign** 语句适用于描述不依赖于时钟的即时信号关系，而 **always** 语句则用于描述更复杂的逻辑，包括那些需要在特定条件下（如时钟边沿或信号变化）执行的行为。

### 2、用自己的话总结 **reg** 类型变量和 **wire** 类型变量的区别

**reg** 类型变量是一种可以存储值的变量，它需要被明确地赋值。一旦被赋值，**reg** 变量会保持这个值，直到它被重新赋值为止。这意味着 **reg** 变量能够跟踪和存储其状态，这使得它非常适合用于描述时序逻辑，如触发器和寄存器，它们需要在特定的时钟边沿或条件触发时保持或更新其状态。

**wire** 类型变量主要用于表示连接不同模块或逻辑门的信号线。**wire** 变量不能直接赋值，它只能通过连续赋值语句（如 **assign** 语句）或模块实例化中的连续赋值来获得值。**wire** 变量没有状态或存储的概念，它的值完全取决于其驱动表达式的当前值。一旦输入信号发生变化，**wire** 变量的值会立即随之改变，这使得它非常适合用于描述组合逻辑，其中输出是输入信号的直接函数。

简而言之，**reg** 变量用于存储和跟踪状态，适用于时序逻辑；而 **wire** 变量用于传递信号，适用于组合逻辑，且不存储任何状态信息。

### 3、完成第三页 PPT 中的真值表

a[0]	a[1]	b[0]	b[1]	equal
0	0	0	0	1
0	0	0	1	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

### 4、请使用 **vivado** 依次编写半加器、全加器、8 位加法器并验证正确性

```

23 module half_adder(
24     input A,
25     input B,
26     output SUM,
27     output CARRY
28 );
29     assign SUM = A ^ B; // 异或实现 SUM
30     assign CARRY = A & B; // 与实现 CARRY
31 endmodule

```

半加器

```

23 module full_adder(
24     input A,
25     input B,
26     input CIN,
27     output SUM,
28     output COUT
29 );
30     wire sum1, carry1, carry2;
31
32     half_adder ha1(.A(A), .B(B), .SUM(sum1), .CARRY(carry1));
33     half_adder ha2(.A(sum1), .B(CIN), .SUM(SUM), .CARRY(carry2));
34
35     assign COUT = carry1 | carry2; // 或实现进位输出
36 endmodule

```

全加器

```

23 module adder_8bit(
24     input [7:0] A,
25     input [7:0] B,
26     input CIN,
27     output [7:0] SUM,
28     output COUT
29 );
30     wire [7:0] carry; // 每个位的进位信号
31
32     full_adder fa0(.A(A[0]), .B(B[0]), .CIN(CIN), .SUM(SUM[0]), .COUT(carry[0]));
33     full_adder fa1(.A(A[1]), .B(B[1]), .CIN(carry[0]), .SUM(SUM[1]), .COUT(carry[1]));
34     full_adder fa2(.A(A[2]), .B(B[2]), .CIN(carry[1]), .SUM(SUM[2]), .COUT(carry[2]));
35     full_adder fa3(.A(A[3]), .B(B[3]), .CIN(carry[2]), .SUM(SUM[3]), .COUT(carry[3]));
36     full_adder fa4(.A(A[4]), .B(B[4]), .CIN(carry[3]), .SUM(SUM[4]), .COUT(carry[4]));
37     full_adder fa5(.A(A[5]), .B(B[5]), .CIN(carry[4]), .SUM(SUM[5]), .COUT(carry[5]));
38     full_adder fa6(.A(A[6]), .B(B[6]), .CIN(carry[5]), .SUM(SUM[6]), .COUT(carry[6]));
39     full_adder fa7(.A(A[7]), .B(B[7]), .CIN(carry[6]), .SUM(SUM[7]), .COUT(COUT));
40 endmodule

```

8位加法器

Name	Value	
A	1	半加器
B	1	
SUM	0	
CARRY	1	

  

Name	Value	
A	1	全加器
B	1	
CIN	1	
SUM	1	
COUT	1	

  

Name	Value	
A[7:0]	b1	8位加法器
B[7:0]	55	
CIN	1	
SUM[7:0]	07	
COUT	1	

显然，半加器、全加器、8位加法器经过测试，都是正确的。