always\_comb begin

    no\_overflow = ~|mac\_out[N-1][QM + QN + WM + WN + N - 1 : QM + QN + WN - 1]; //all integer bit bitwise not(or)

    no\_underflow = &mac\_out[N-1][QM + QN + WM + WN + N - 1 : QM + QN + WN - 1]; //all integer bit bitwise and

    if (no\_overflow | no\_underflow) begin

        mac\_final = $signed(mac\_out[N-1][QM + QN + WN - 1 : WN]); //Truncate

    end else begin

      if (mac\_out[N-1][QM + QN + WM + WN + N - 1]) //msb==1 means nagative value

        mac\_final = -(2\*\*(QM + QN - 1));  //minimum negetive

      else

        mac\_final = 2\*\*(QM + QN - 1) - 1;  //maximum positive

    end

end

**没有上溢出=所选择的位都是0**

**没有下溢=所选择的位都是1**

**这个是检测是否溢出的，如果溢出了，比如overflow了，最高位就会有1，那么no\_overflow就会是false；**

**比如1000是-8，然后再加一个1111，如果是符号位拓展了，就是11111000+11111111 = 11110111，所以就溢出了，**

**如果不溢出，原本位长的符号位应该和扩展位一致**

下溢（Underflow）指的是在进行数值计算时，结果的绝对值小于能够用于表示这个数的最小值。在计算机中，数值通常以有限位数来表示，如果某个计算结果的绝对值小于能够表示的最小值，就会发生下溢。

下溢可能会导致结果丢失精度，因为小于能够被表示的最小值的数会被截断成零或者某个特殊的小数。

**Implicit nets**

wire [2:0] a, c; // Two vectors

assign a = 3'b101; // a = 101

assign b = a; // b = 1 implicitly-created wire

assign c = b; // c = 001 <-- bug

my\_module i1 (d,e); // d and e are implicitly one-bit wide if not declared.  
// This could be a bug if the port was intended to be a vector.

Adding `default\_nettype none would make the second line of code an error, which makes the bug more visible.

**Unpacked vs. Packed Arrays**

reg [7:0] mem [255:0]; // 256 unpacked elements, each of which is a 8-bit packed vector of reg.

reg mem2 [28:0]; // 29 unpacked elements, each of which is a 1-bit reg.

**Bitwise vs. Logical Operators**

A bitwise operation between two N-bit vectors replicates the operation for each bit of the vector and produces a N-bit output,

while a logical operation treats the entire vector as a boolean value (true = non-zero, false = zero) and produces a 1-bit output.

负数看真值，去掉符号位，先-1再按位取反

For synthesizing hardware, two types of always blocks are relevant:

Combinational: always @(\*)

Clocked: always @(posedge clk)

A note on wire vs. reg:

The left-hand-side of an assign statement must be a net type (e.g., wire),

while the left-hand-side of a procedural assignment (in an always block) must be a variable type (e.g., reg).

**Blocking vs. Non-Blocking Assignment**

There are three types of assignments in Verilog:

·Continuous assignments (assign x = y;). Can only be used when not inside a procedure ("always block").

·Procedural blocking assignment: (x = y;). Can only be used inside a procedure.

·Procedural non-blocking assignment: (x <= y;). Can only be used inside a procedure.

In a **combinational** always block, use **blocking** assignments. In a **clocked** always block, use **non-blocking** assignments.