|  |
| --- |
| Verilog HDL Operators |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Verilog Operator** | **Name** | **Functional Group** | | [ ] | bit-select or part-select |  | | ( ) | parenthesis |  | | !  ~  &  |  ~&  ~|  ^  ~^ or ^~ | logical negation  negation  reduction AND  reduction OR  reduction NAND  reduction NOR  reduction XOR  reduction XNOR | logical  bit-wise  reduction  reduction  reduction  reduction  reduction  reduction | | +  - | unary (sign) plus  unary (sign) minus | arithmetic  arithmetic | | { } | concatenation | concatenation | | {{ }} | replication | replication | | \*  /  % | multiply  divide  modulus | arithmetic  arithmetic  arithmetic | | +  - | binary plus  binary minus | arithmetic  arithmetic | | <<  >> | shift left  shift right | shift  shift | | >  >=  <  <= | greater than  greater than or equal to  less than  less than or equal to | relational relational relational relational | | ==  != | case equality  case inequality | equality equality | | &  ^  | | bit-wise AND  bit-wise XOR  bit-wise OR | bit-wise  bit-wise  bit-wise | | &&  || | logical AND  logical OR | logical  logical | | ?: | conditional | conditional |     **1. Arithmetic**   There are five arithmetic operators in Verilog.  **module** Arithmetic (A, B, Y1, Y2, Y3, Y4, Y5);  **input** [2:0] A, B;  **output** [3:0] Y1;  **output** [4:0] Y3;  **output** [2:0] Y2, Y4, Y5;  **reg** [3:0] Y1;  **reg** [4:0] Y3;  **reg** [2:0] Y2, Y4, Y5;    **always** @(A **or** B)  **begin**                                    Y1=A+B;//addition                                    Y2=A-B;//subtraction                                    Y3=A\*B;//multiplication                                    Y4=A/B;//division                                    Y5=A%B;//modulus of A divided by B  **end**  **endmodule**    **2. Sign**    These operators simply assign a positive "+" or negative "-" sign to a singular operand. Usually no sign operators is defined, in which case the default "+" is assumed.    **module** Sign (A, B, Y1, Y2, Y3);    **input** [2:0] A, B;  **output** [3:0] Y1, Y2, Y3;  **reg** [3:0] Y1, Y2, Y3;    **always** @(A or B)  **begin**                                    Y1=+A/-B;                                    Y2=-A+-B;                                    Y3=A\*-B;  **end**  **endmodule**    **3. Relational**    Relational operators compare two operands and returns an indication of whether the compared relationship is true or false. The result of a comparison is either **0** or **1**. It is **0** if the comparison is false and **1** is the comparison is true.    **module** Relational (A, B, Y1, Y2, Y3, Y4);    **input** [2:0] A, B;  **output** Y1, Y2, Y3, Y4;  **reg** Y1, Y2, Y3, Y4;    **always** @(A **or** B)  **begin**                                    Y1=A<B;//less than                                    Y2=A<=B;//less than or equal to                                    Y3=A>B;//greater than  **if** (A>B)                                                     Y4=1;  **else**                                                     Y4=0;  **end**  **endmodule**    **4. Equality and inequality**    Equality and inequality operators are used in exactly the same way as relational operators and return a true or false indication depending on whether any two operands are equivalent or not.    **module** Equality (A, B, Y1, Y2, Y3);    **input** [2:0] A, B;  **output** Y1, Y2;  **output** [2:0] Y3;  **reg** Y1, Y2;  **reg** [2:0] Y3;  **always** @(A **or** B)  **begin**                                    Y1=A==B;//Y1=1 if A equivalent to B                                    Y2=A!=B;//Y2=1 if A not equivalent to B  **if** (A==B)//parenthesis needed                                                     Y3=A;  **else**                                                     Y3=B;  **end**  **endmodule**    **5. Logical**    Logical comparison operators are used in conjuction with relational and equality operators as described in the relational operators section and equality and inequality operators section. They provide a means to perform multiple comparisons within a a single expression.    **module** Logical (A, B, C, D, E, F, Y);    **input** [2:0] A, B, C, D, E, F;  **output** Y;  **reg** Y;    **always** @(A **or** B **or** C **or** D **or** E **or** F)  **begin**  **if** ((A==B) && ((C>D) || !(E<F)))                                                     Y=1;  **else**                                                     Y=0;  **end**  **endmodule**    **6. Bit-wise**    Logical bit-wise operators take two single or multiple operands on either side of the operator and return a single bit result. The only exception is the **NOT** operator, which negates the single operand that follows. Verilog does not have the equivalent of **NAND** or **NOR** operator, their funstion is implemented by negating the **AND** and **OR** operators.    **module** Bitwise (A, B, Y);    **input** [6:0] A;  **input** [5:0] B;  **output** [6:0] Y;  **reg** [6:0] Y;    **always** @(A **or** B)  **begin**                                    Y(0)=A(0)&B(0); //binary AND                                    Y(1)=A(1)|B(1); //binary OR                                    Y(2)=!(A(2)&B(2)); //negated AND                                    Y(3)=!(A(3)|B(3)); //negated OR                                    Y(4)=A(4)^B(4); //binary XOR                                    Y(5)=A(5)~^B(5); //binary XNOR                                    Y(6)=!A(6); //unary negation  **end**  **endmodule**    **7. Shift**    Shift operators require two operands. The operand before the operator contains data to be shifted and the operand after the operator contains the number of single bit shift operations to be performed. **0** is being used to fill the blank positions.    **module** Shift (A, Y1, Y2);    **input** [7:0] A;  **output** [7:0] Y1, Y2;  **parameter** B=3; reg [7:0] Y1, Y2;    **always** @(A)  **begin**                                    Y1=A<<B; //logical shift left                                    Y2=A>>B; //logical shift right  **end**  **endmodule**    **8. Concatenation and Replication**    The concatenation operator "{ , }" combines (concatenates) the bits of two or more data objects. The objects may be scalar (single bit) or vectored (muliple bit). Mutiple concatenations may be performed with a constant prefix and is known as replication.    **module** Concatenation (A, B, Y);    **input** [2:0] A, B;  **output** [14:0] Y;  **parameter** C=3'b011;  **reg** [14:0] Y;    **always** @(A or B)  **begin**                                    Y={A, B, (2{C}}, 3'b110};  **end**  **endmodule**    **9. Reduction**    Verilog has six reduction operators, these operators accept a single vectored (multiple bit) operand, performs the appropriate bit-wise reduction on all bits of the operand, and returns a single bit result. For example, the four bits of A are **AND**ed together to produce Y1.    **module** Reduction (A, Y1, Y2, Y3, Y4, Y5, Y6);    **input** [3:0] A;  **output** Y1, Y2, Y3, Y4, Y5, Y6;  **reg** Y1, Y2, Y3, Y4, Y5, Y6;    **always** @(A)  **begin**                                    Y1=&A; //reduction AND                                    Y2=|A; //reduction OR                                    Y3=~&A; //reduction NAND                                    Y4=~|A; //reduction NOR                                    Y5=^A; //reduction XOR                                    Y6=~^A; //reduction XNOR  **end**  **endmodule**    **10. Conditional**    An expression using conditional operator evaluates the logical expression before the "?". If the expression is true then the expression before the colon (:) is evaluated and assigned to the output. If the logical expression is false then the expression after the colon is evaluated and assigned to the output.    **module** Conditional (Time, Y);    **input** [2:0] Time;  **output** [2:0] Y;  **reg** [2:0] Y;  **parameter** Zero =3b'000;  **parameter** TimeOut = 3b'110;    **always** @(Time)  **begin**                                    Y=(Time!=TimeOut) ? Time +1 : Zero;  **end**  **endmodule** |