SystemVerilog - Оператори

[Оператори](#_s37gsgt8mxgp)

[SystemVerilog assignment operators](#_dpqsq9l2kczc)

[Пример - логически оператори](#_1t3h5sf)

[Пример - аритметични оператори](#_xriq3a296t7l)

[Пример - условен оператор](#_4d34og8)

[Оператор if-else-if](#_i73pcpwqeh8)

[Оператор case](#_yvihuiksh6tl)

[Пример alu.sv](#_f9suac5jjz82)

### Оператори

| **побитови**~ НЕ& И| ИЛИ^ XOR~^ XNOR | **логически**! логическо НЕ&& логическо И|| логическо ИЛИ | **релации**< по-малко<= по-малко или равно> по-голямо>= по-голямо или равно== равно!= неравно=== равно (вкл. X и Z)!== неравно (вкл. X и Z) |
| --- | --- | --- |
| **аритметични**+ събиране- изваждане\* умножение/ деление% остатък | **други**{ } конкатенация<< преместване в ляво>> преместване в дясно?: условен оператор |  |

#### SystemVerilog assignment operators

**Continuous assignment (nets and variables)**

 **assign** net\_lvalue = expression

 example

assign mynet = enable;

**Procedural assignments (variables)**

**Blocking**

variable\_lvalue assignment\_operator expression

assignment\_operator: = | += | -= | \*= | /= | %= | &= | |= | ^= | <<= | >>= | <<<= | >>>=

**=** assign right-hand side to left-hand side

**+=** add right-hand side to left-hand side and assign

-= subtract right-hand side from left-hand side and assign

\*= multiply left-hand side by right-hand side and assign

/= divide left-hand side by right-hand side and assign

%= divide left-hand side by right-hand side and assign the remainder

&= bitwise AND right-hand side with left-hand side and assign

|= bitwise OR right-hand side with left-hand side and assign

^= bitwise exclusive OR right-hand side with left-hand side and assign

<<= bitwise left-shift the left-hand side by the number of times indicated by the right-hand side and assign

>>= bitwise right-shift the left-hand side by the number of times indicated by the right-hand side and assign

<<<= arithmetic left-shift the left-hand side by the number of times indicated by the right-hand side and assign

>>>= arithmetic right-shift the left-hand side by the number of times indicated by the right-hand side and assign

**Nonblocking**

variable\_lvalue **<=** expression

| **Пример - побитови оператори** | **логически оператори** |
| --- | --- |
| module bitwise\_operators( input [3:0] A, B, output [3:0] A\_or\_B, A\_and\_B, not\_A );      assign A\_or\_B = A | B; assign A\_and\_B = A & B; assign not\_A = ~A; endmodule | module logical\_operators( input [3:0] A, B, output [3:0] A\_or\_B, A\_and\_B, not\_A );  assign A\_or\_B = A || B; assign A\_and\_B = A && B; assign not\_A = !A; endmodule |
|  A B -> A\_or\_B A\_and\_B not\_A0000 0000 -> 0000 0000 11111111 0000 -> 1111 0000 00000001 0001 -> 0001 0001 11101010 0101 -> 1111 0000 0101 | A B -> A\_or\_B A\_and\_B not\_A0000 0000 -> 0000 0000 00011111 0000 -> 0001 0000 00000001 0001 -> 0001 0001 00001010 0101 -> 0001 0001 0000 |
|  |  |

### Пример - релации

module comparator (

 output A\_lt\_B, A\_eq\_B, A\_gt\_B,

 input [3:0] A, B

 );

 assign A\_lt\_B = A < B;

 assign A\_gt\_B = A > B;

 assign A\_eq\_B = A == B;

endmodule

### Пример - аритметични оператори

module adder\_4b(

 output [3:0] Sum,

 output C\_out,

 input [3:0] A, B,

 input C\_in

 );

 assign {C\_out, Sum} = A + B + C\_in;

endmodule

### Пример - условен оператор

examples/simulation/mux\_2x1.sv

module mux\_2x1\_cont(

 output logic [7:0] y,

 input [7:0] a,b,

 input select);

 assign y = select ? a : b;

endmodule

### Оператор if-else-if

if (израз 1) оператор 1

else if (израз 2) оператор 2

… още else if …

else оператор 3

Пример

examples/simulation/mux\_2x1.sv

module mux\_2x1\_proc(

 output logic [7:0] y,

 input [7:0] a,b,

 input select);

 always\_comb

 if (select) y = a;

 else y = b;

endmodule

Пример

if (alu\_func == 2’b00) aluout = a + b;

else if (alu\_func == 2’b01) aluout = a - b;

else if (alu\_func == 2’b10) aluout = a & b;

else aluout = a | b;

### Оператор case

case (израз)

избор 1: оператор 1

избор 2: оператор 2

… още case …

default: оператор 3

endcase

### Пример alu.sv

examples/simulation/alu\_definitions.sv

package definitions;

 typedef enum logic [2:0] {ADD,SUB,MULT,DIV,SL,SR} opcode\_t;

 typedef enum logic {UNSIGNED, SIGNED} operand\_type\_t;

 typedef union packed {

 logic [23:0] u\_data;

 logic signed [23:0] s\_data;

 } data\_t;

 typedef struct packed {

 opcode\_t opc;

 operand\_type\_t op\_type;

 data\_t op\_a;

 data\_t op\_b;

 } instruction\_t;

endpackage

examples/simulation/alu.sv

import definitions::\*; // import package into $unit space

module alu (input instruction\_t instr, output data\_t alu\_out);

 always\_comb begin

 if (instr.op\_type == SIGNED) begin

 alu\_out.s\_data = instr.op\_a.s\_data;

 unique case (instr.opc)

 ADD : alu\_out.s\_data += instr.op\_b.s\_data;

 SUB : alu\_out.s\_data -= instr.op\_b.s\_data;

 MULT : alu\_out.s\_data \*= instr.op\_b.s\_data;

 DIV : alu\_out.s\_data /= instr.op\_b.s\_data;

 SL : alu\_out.s\_data <<<= 2;

 SR : alu\_out.s\_data >>>= 2;

 endcase

 end

 else begin

 alu\_out.u\_data = instr.op\_a.u\_data;

 unique case (instr.opc)

 ADD : alu\_out.u\_data += instr.op\_b.u\_data;

 SUB : alu\_out.u\_data -= instr.op\_b.u\_data;

 MULT : alu\_out.u\_data \*= instr.op\_b.u\_data;

 DIV : alu\_out.u\_data /= instr.op\_b.u\_data;

 SL : alu\_out.u\_data <<= 2;

 SR : alu\_out.u\_data >>= 2;

 endcase

 end

 end

endmodule

examples/simulation/alu\_test.sv

import definitions::\*; // import package into $unit space

module alu\_test;

 instruction\_t instr;

 data\_t alu\_out;

 alu uut (.\*);

 initial begin

 instr.opc = ADD;

 instr.op\_type = UNSIGNED;

 instr.op\_a = 24'h0005;

 instr.op\_b = 24'h0003;

 #10;

 instr.opc = SUB;

 #10;

 instr = {MULT,UNSIGNED,24'd12,24'd3};

 #10;

 $finish;

 end

 initial begin

 $monitor(alu\_out);

 end

endmodule