

ОСНОВЫ

VerilogHDL/SystemVerilog

(синтез и моделирование)

Операторы языка

Операторы

□ Группы операторов:

- ✓ Арифметические - Arithmetic
- ✓ Побитовые – Bit-wise
- ✓ Свертки - Reduction
- ✓ Отношения - Relational
- ✓ Равенства - Equality
- ✓ Логические - Logical
- ✓ Сдвига - Shift

□ Отдельные операторы:

- ✓ Сцепления – Concatenation
- ✓ Повторения - Replication
- ✓ Условного выбора - Conditional

Операторы отношения - Relational

Символ	Функция	$\text{ain} = 3'b101 ; \text{bin} = 3'b110 ; \text{cin} = 3'b01x$	
$>$	Больше чем	$\text{ain} > \text{bin} \Rightarrow 1'b0$	$\text{bin} > \text{cin} \Rightarrow 1'bx$
$<$	Меньше чем	$\text{ain} < \text{bin} \Rightarrow 1'b1$	$\text{bin} < \text{cin} \Rightarrow 1'bx$
$>=$	Больше или равно	$\text{ain} >= \text{bin} \Rightarrow 1'b0$	$\text{bin} >= \text{cin} \Rightarrow 1'bx$
$<=$	Меньше или равно	$\text{ain} <= \text{bin} \Rightarrow 1'b1$	$\text{bin} <= \text{cin} \Rightarrow 1'bx$

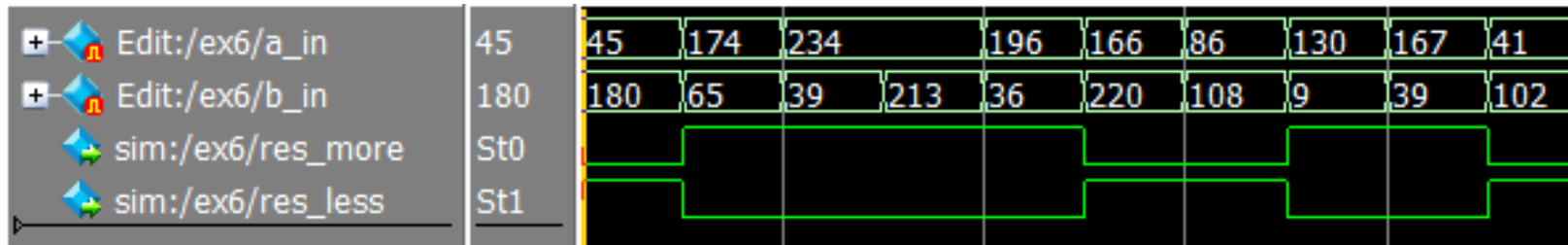
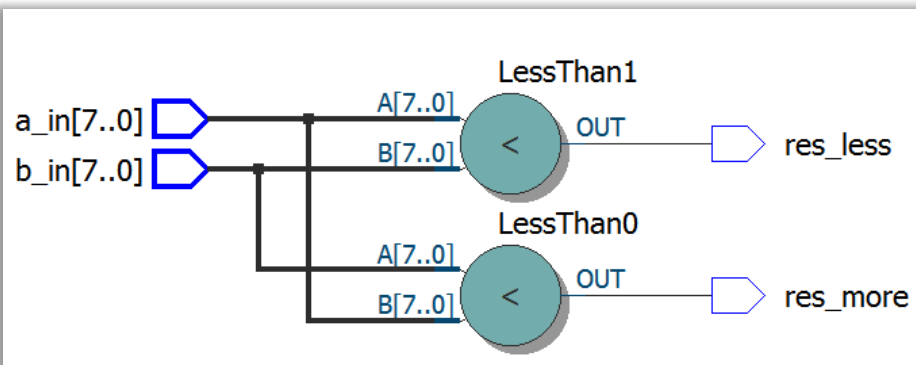
- ❑ Возвращают 1-битное значение: 1 (true) / 0 (false)
- ❑ Если один из операндов имеет меньший размер, то он дополняется слева необходимым числом разрядов:
 - ✓ нулями если операнд без знака
 - ✓ знаковым разрядом если операнд знаковый.
- ❑ X и Z рассматриваются как неизвестные значения

Пример (операторы отношения; аргументы без знака)

```
module ex6
(  input [7:0] a_in, b_in,
  output res_more, res_less);

assign res_more = a_in > b_in;
assign res_less = a_in < b_in;

endmodule
```

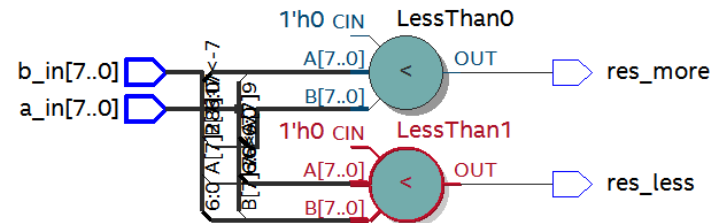
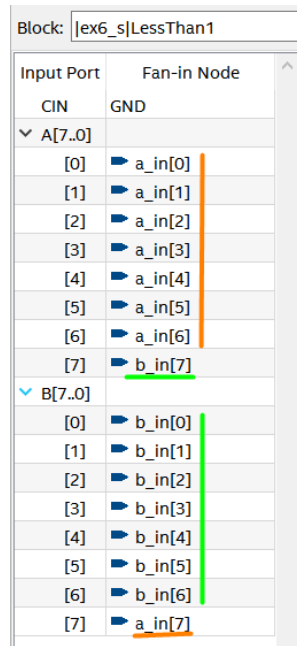


Пример (операторы отношения; аргументы со знаком)

```
module ex6_s  
(  input signed [7:0] a_in, b_in,  
  output res_more, res_less);
```

```
  assign res_more = a_in > b_in;  
  assign res_less = a_in < b_in;
```

```
endmodule
```



	/ex6s_tb/a_in	90	36	39	42	45	48	51	54	57	60	63	66
	/ex6s_tb/b_in	-46	84	91	98	105	112	119	126	-123	-116	-109	-102
	/ex6s_tb/res_m...	St1											
	/ex6s_tb/res_less	St0											

Операторы равенства - Equality

Символ	Функция	$ain = 3'b101 ; bin = 3'b110 ; cin = 3'b01x$	
==	Equality	$ain == bin \Rightarrow 1'b0$	$cin == cin \Rightarrow 1'bx$
!=	Inequality	$ain != bin \Rightarrow 1'b1$	$cin != cin \Rightarrow 1'bx$
===	Case equality	$ain === bin \Rightarrow 1'b0$	$cin === cin \Rightarrow 1'b1$
!==	Case inequality	$ain !== bin \Rightarrow 1'b1$	$cin !== cin \Rightarrow 1'b0$

- ❑ Возвращают 1-битное значение: true (1) / false (0)
- ❑ Если один из операндов имеет меньший размер, то он дополняется слева необходимым числом разрядов: нулями если операнд без знака; знаковым разрядом если операнд знаковый.
- ❑ Операторы == и != рассматривают X и Z как неизвестные \Rightarrow результат $\neg X$
- ❑ Операторы === и !== рассматривают X и Z как определенные значения. Для равенства положения X и Z в операндах должно полностью совпадать.

Пример использования операторов равенства

```
module ex7
(  input  [7:0] a_in, b_in,
  output eq, case_eq);

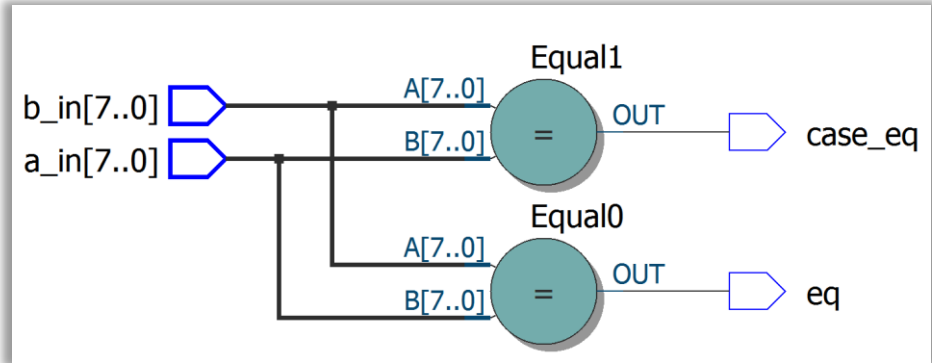
assign eq      = a_in == b_in;
assign case_eq = a_in === b_in;

endmodule

module ex7
(  input  signed [7:0] a_in, b_in,
  output eq, case_eq);

assign eq      = a_in == b_in;
assign case_eq = a_in === b_in;

endmodule
```



+	Edit:/ex7/a_in	11101101	11101101	11110011	111x11xx	01010010	111xx000	10001110
+	Edit:/ex7/b_in	11101101	11101101	01110110	11xx11xx	11100010	111xx000	10001110
	sim:/ex7/eq	St1						
	sim:/ex7/case_eq	St1						

Операторы условного выбора - Conditional

Символ	Функция	Формат и примеры
?:	Conditional	(condition) ? true_value : false_value

- ❑ Если условие (condition) :
 - ✓ Истинно, то результат - true_value
 - ✓ Ложно, то результат - false_value
- ❑ Примеры (*при sel=1 => q=a; при sel=0 => q=b*):
 - ✓ **assign q = (sel == 1'b1) ? a : b;**
 - ✓ **assign q = (sel) ? a : b;**
 - ✓ **assign q = (sel != 1'b0) ? a : b;**

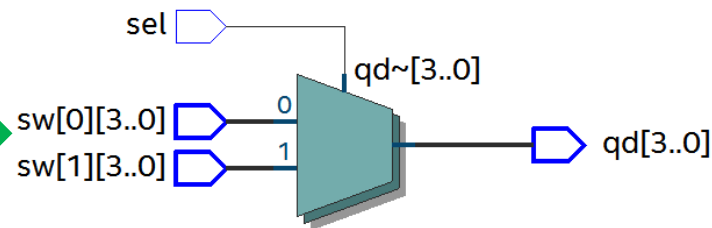
Пример

```
module mux_a  
(  input  [3:0] sw [1:0],  
  input  sel,  
  output [3:0] qd);
```

```
  assign qd = (sel) ? sw[1] : sw[0];
```

```
endmodule
```

Подстановка



❑ Сравните с приведенным ниже описанием

```
module mux_  
(  input  [3:0] sw [1:0],  
  input  sel,  
  output [3:0] qd);
```

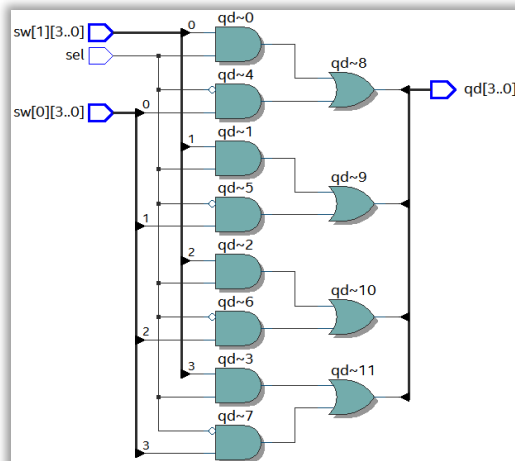
```
  wire [3:0] sel_v;
```

```
  assign sel_v[0] = sel;  
  assign sel_v[1] = sel;  
  assign sel_v[2] = sel;  
  assign sel_v[3] = sel;
```

```
  assign qd = (sw[1] & sel_v) | (sw[0] & ~sel_v);
```

```
endmodule
```

Синтез



Побитовые операторы - Bitwise Operators

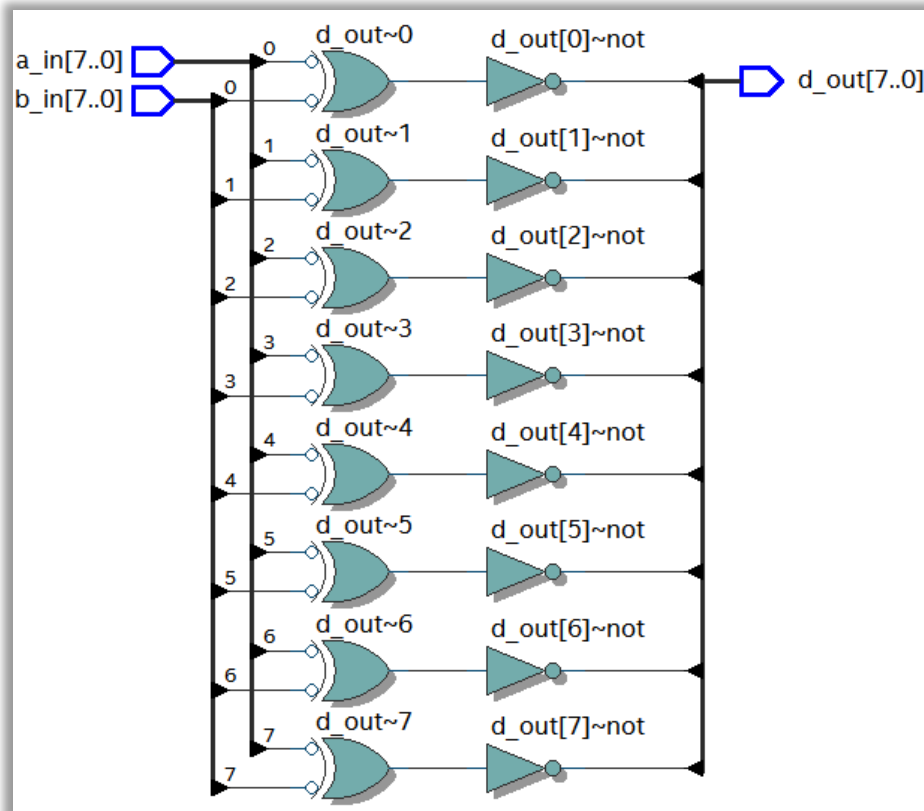
Символ	Функция	ain = 3'b101 ; bin = 3'b110 ; cin = 3'b01x	
~	Инверсия	$\sim \text{ain} \Rightarrow 3\text{b}'010$	$\sim \text{cin} \Rightarrow 3\text{b}'10\text{x}$
&	AND	$\text{ain} \& \text{bin} \Rightarrow 3\text{b}'100$	$\text{bin} \& \text{cin} \Rightarrow 3\text{b}'010$
	OR	$\text{ain} \text{bin} \Rightarrow 3\text{b}'111$	$\text{bin} \text{cin} \Rightarrow 3\text{b}'11\text{x}$
^	XOR	$\text{ain} \wedge \text{bin} \Rightarrow 3\text{b}'011$	$\text{bin} \wedge \text{cin} \Rightarrow 3\text{b}'10\text{x}$
$\wedge \sim$ or $\sim \wedge$	XNOR	$\text{ain} \wedge \sim \text{bin} \Rightarrow 3\text{b}'100$	$\text{bin} \sim \wedge \text{cin} \Rightarrow 3\text{b}'01\text{x}$

- ❑ Функция применяется побитно (между соответствующими битами векторов)
- ❑ Если один из операндов имеет меньший размер, то он дополняется слева необходимым числом нулей.
- ❑ Значения X и Z рассматриваются как неизвестные – результат X
 - ✓ результат может иметь другое значение:
 - 0 в случае $0 \& x = 0$
 - 1 в случае $1 | x = 1$

Пример выполнения побитовых операций

```
module ex_bw
(  input [7:0] a_in,b_in,
  output [7:0] d_out);

assign d_out = (~a_in ^ ~b_in );
endmodule
```



Оператор объединения Concatenate

Символ	Функция	Пример
{ }	Concatenate	$\text{ain} = 3'b010 ; \text{bin} = 3'b110$ $\{\text{ain}, \text{bin}\} \Rightarrow 6'b010_110$

- ❑ Оператор позволяет объединить сигналы и вектора
- ❑ Создается, на время, новый вектор, содержащий элементы объединяемых сигналов/векторов

Оператор повторения Replicate

Символ	Функция	Пример
<code>{n {}}</code>	Replicate	<code>{3 {3'b101}} ⇒ 9'b101_101_101</code>

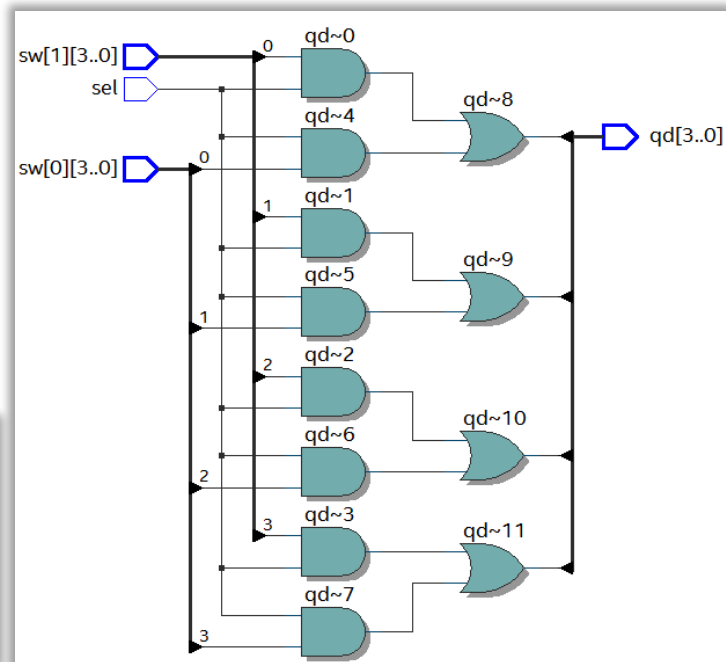
- ❑ Оператор `n` раз повторяет содержимое фигурных скобок

Пример (операторы объединения и повторения)

```
module mux_  
(  input  [3:0] sw [1:0],  
  input  sel,  
  output [3:0] qd);  
  
wire [3:0] sel_v;  
  
assign sel_v[0] = sel;  
assign sel_v[1] = sel;  
assign sel_v[2] = sel;  
assign sel_v[3] = sel;  
  
assign qd = (sw[1] & sel_v) | (sw[0] & ~sel_v);  
endmodule
```



```
module mux_r  
(  input  [3:0] sw [1:0],  
  input  sel,  
  output [3:0] qd);  
  
  assign qd = (sw[1] & {4{sel}}) | (sw[0] & {sel, sel, sel, sel});  
endmodule
```



Операторы свертки - Reduction

Символ	Функция	Пример ain=4'b1010; bin=4'b10xz; cin=4'b111z		
&	AND all bits	&ain \Rightarrow 1'b0	&bin \Rightarrow 1'b0	&cin \Rightarrow 1'bx
~&	NAND all bits	~&ain \Rightarrow 1'b1	~&bin \Rightarrow 1'b1	~&cin \Rightarrow 1'bx
	OR all bits	ain \Rightarrow 1'b1	bin \Rightarrow 1'b1	cin \Rightarrow 1'b1
~	NOR all bits	~ ain \Rightarrow 1'b0	~ bin \Rightarrow 1'b0	~ cin \Rightarrow 1'b0
^	XOR all bits	^ain \Rightarrow 1'b0	^bin \Rightarrow 1'bx	^cin \Rightarrow 1'bx
^~ or ~^	XNOR all bits	~^ain \Rightarrow 1'b1	~^bin \Rightarrow 1'bx	~^cin \Rightarrow 1'bx

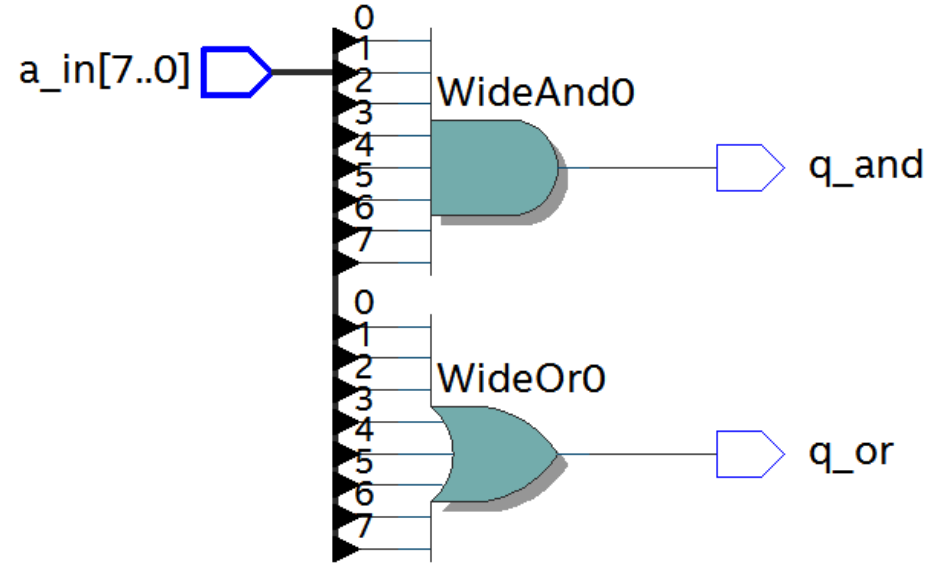
- ❑ Сворачивает вектор в однобитовое значение применяя выбранную функцию
- ❑ Значения X и Z рассматриваются как неизвестные – результат X
 - ✓ результат может иметь другое значение:
 - 0 в случае $0 \& x \dots \& x = 0$
 - 1 в случае $1 | x \dots | x = 1$

Пример (операторы свертки)

```
module ex5
(  input [7:0] a_in,
  output q_and, q_or );

assign q_or  = | a_in;
assign q_and = & a_in;

endmodule
```



Логические операторы - Logical

Символ	Функция	Пример $ain=3'b101$; $bin=3'b000$; $cin=3'b01x$		
!	Expression not true	$!ain \Rightarrow 1'b0$	$!bin \Rightarrow 1'b1$	$!cin \Rightarrow 1'bx$
&&	AND of two expressions	$ain \&\& bin \Rightarrow 1'b0$	$bin \&\& cin \Rightarrow 1'bx$	
	OR of two expressions	$ain bin \Rightarrow 1'b1$	$bin cin \Rightarrow 1'bx$	

❑ Алгоритм

- ✓ Каждый операнд рассматривается как отдельное выражение
- ✓ Выражение с нулевым значением заменяется на false (0)
- ✓ Выражение с ненулевым значением заменяется на true (1)
- ✓ Применяется соответствующая функция
 - Возвращает 1-битное значение true (1) / false (0)

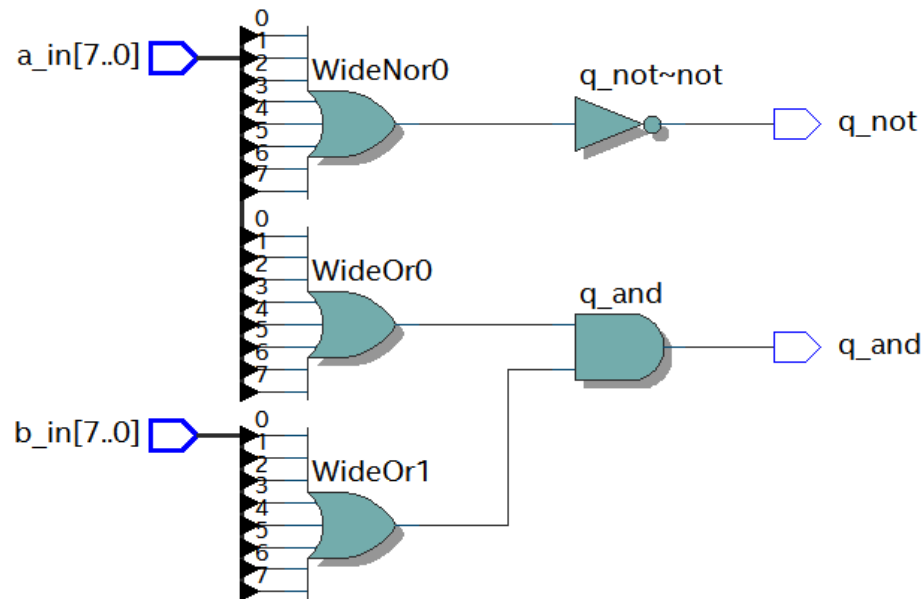
❑ X и Z рассматриваются как неизвестные и результат – X

Пример (логические операторы)

```
module ex8_s
(  input  [7:0] a_in, b_in,
  output q_and, q_not);

assign q_not = ! a_in;
assign q_and = a_in && b_in;

endmodule
```



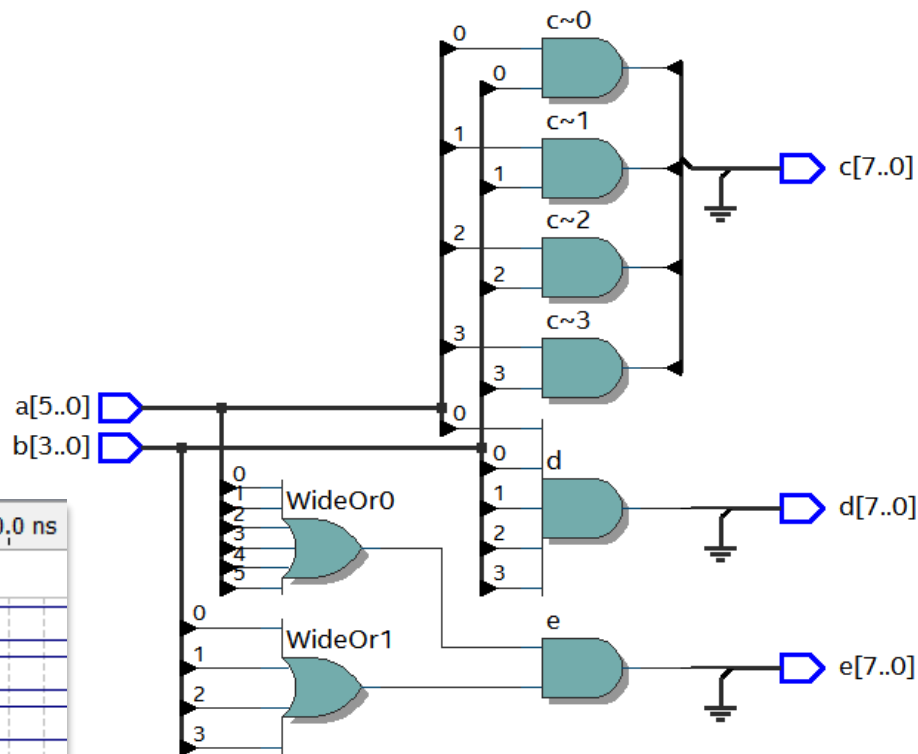
Пример (смесь операторов)

```
module ex8_and2
(  input  [5:0] a,
  input  [3:0] b,
  output [7:0] c, d, e);
```

```
  assign c = a & b;
  assign d = a & &b;
  assign e = a && b;
```

```
endmodule
```

	Name	Value at 0 ps	0 ps	160,0 ns	320,0 ns	480,0 ns	640,0 ns
			0 ps				
in	a	B 100101				100101	
in	b	B 1111				1111	
out	c	B 00000101				00000101	
out	d	B 00000001				00000001	
out	e	B 00000001				00000001	



Чему будет равно значение c, d, e ?

```
module ex8_and1  
(  output [7:0] c, d, e);
```

```
wire  [5:0] a = 6'b100101;  
wire  [3:0] b = 4'b1111;
```

```
assign c = a & b;  
assign d = a & &b;  
assign e = a && b;
```

□ C = 100101 & 001111 = 00000101

□ D = 100101 & &(1111) = 100101 & 000001 = 00000001

```
endmodule
```

□ E = (100101) && (1111) = 1 & 1 = 00000001

8'h5  c[7..0]

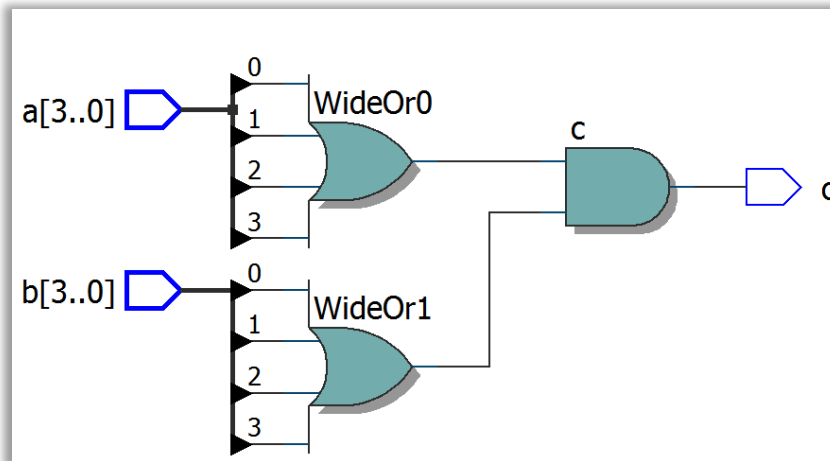
8'h1  d[7..0]

8'h1  e[7..0]

Результат синтеза одинаковый или нет?

```
module ex8_q  
(  input  [3:0] a, b,  
  output  c);  
  
assign c = a && b;  
  
endmodule
```

```
module ex8_qa  
(  input  [3:0] a, b,  
  output  c);  
  
assign c = |a & |b;  
  
endmodule
```



Операторы сдвига - Shift

Символ	Функция	Пример $ain = 3'b101$; $bin = 3'b01x$	
<<	Logical shift left	$ain << 2 \Rightarrow 3'b100$	$bin << 2 \Rightarrow 3'bx00$
>>	Logical shift right	$ain >> 2 \Rightarrow 3'b001$	$bin >> 2 \Rightarrow 3'b000$
<<<	Arithmetic shift left	$ain <<< 2 \Rightarrow 3'b100$	$bin <<< 2 \Rightarrow 3'bx00$
>>>	Arithmetic shift right	$ain >>> 2 \Rightarrow 3'b111$ (signed)	$bin >>> 2 \Rightarrow 3'b000$ (signed)

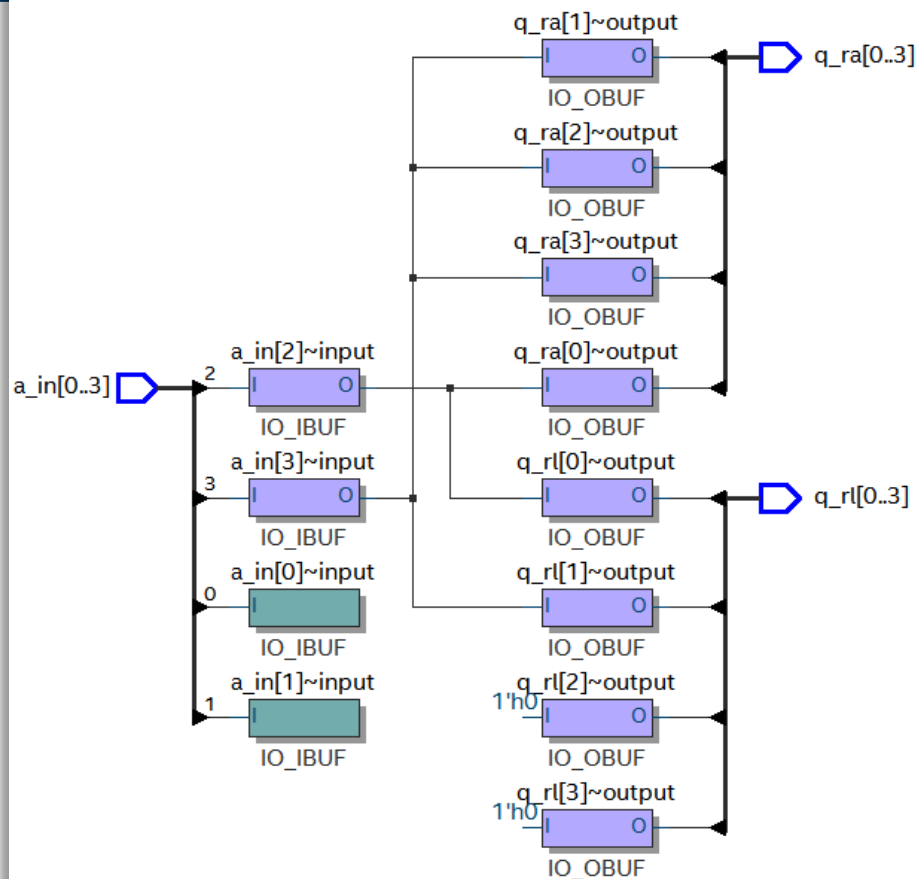
- ❑ Сдвигает вектор влево или вправо на заданное число бит
- ❑ Сдвиг влево (логический или арифметический):
 - ✓ освобождающиеся разряды заполняются нулями
- ❑ Сдвиг вправо
 - ✓ Логический: освобождающиеся разряды заполняются нулями
 - ✓ Арифметический (без знаковый): освобождающиеся разряды заполняются нулями
 - ✓ Арифметический (знаковый): освобождающиеся разряды заполняются знаковым разрядом
- ❑ Сдвигаемые биты теряются
- ❑ Сдвиг на значение X или Z (правый операнд) дает результат - X

Пример (операторы сдвига)

```
module ex9
(  input  signed [3:0] a_in,
  output [3:0]  q_ra, q_rl);

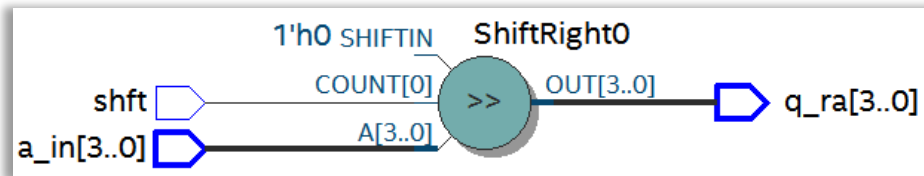
assign q_ra = a_in >>> 2;
assign q_rl = a_in >> 2;

endmodule
```



Пример (сдвиг на произвольное число разрядов)

```
module ex9_  
  ( input [3:0] a_in,  
    input shft,  
    output [3:0] q_ra);  
  assign q_ra = a_in >> shft;  
endmodule
```



Арифметические операторы Arithmetic

Символ	Функция	Пример ain=5; bin=10; cin=2'b01; din=2'b0z		
+	Add, Positive	$\text{bin} + \text{cin} \Rightarrow 11$	$+\text{bin} \Rightarrow 10$	$\text{ain} + \text{din} \Rightarrow x$
-	Subtract, Negate	$\text{bin} - \text{cin} \Rightarrow 9$	$-\text{bin} \Rightarrow -10$	$\text{ain} - \text{din} \Rightarrow x$
*	Multiply	$\text{ain} * \text{bin} \Rightarrow 50$		
/	Divide	$\text{bin} / \text{ain} \Rightarrow 2$		
%	Modulus	$\text{bin} \% \text{ain} \Rightarrow 0$		
**	Exponent	$\text{ain} ** 2 \Rightarrow 25$		

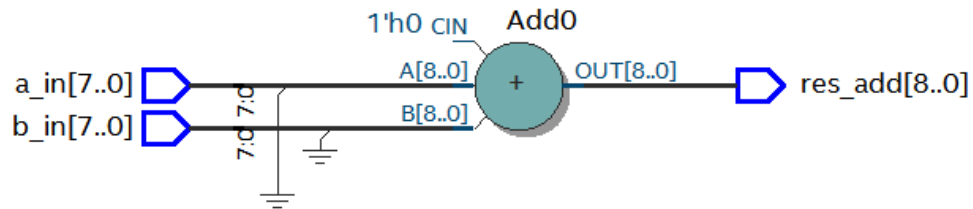
- ❑ Если результат имеет большую разрядность чем операнды, то перенос реализуется автоматически
- ❑ Переносы теряются если операнды и результат имеют одинаковую разрядность

Пример ADD

```
module ex1
(  input  [7:0] a_in, b_in,
  output [8:0] res_add);

assign res_add = a_in + b_in;

endmodule
```



		Time				
		0	1	2	3	4
+	Edit:/ex1/a_in	0				
+	Edit:/ex1/b_in	255				
+	sim:/ex1/res_add	255	256	257	258	259

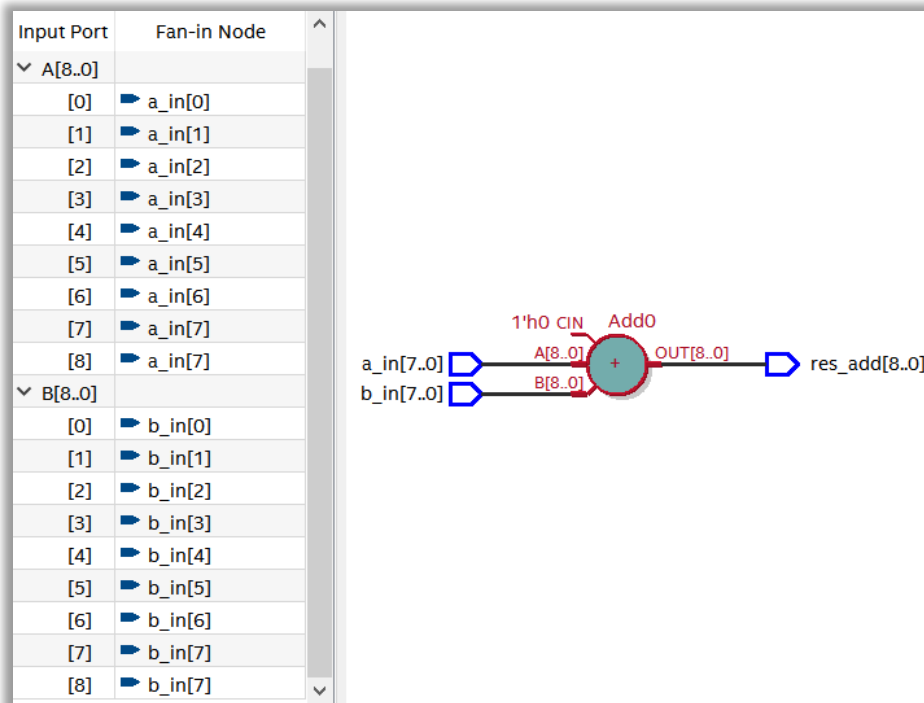
Пример знакового сумматора

```

module ex1s
(  input  signed [7:0] a_in, b_in,
  output [8:0] res_add);

assign res_add = a_in + b_in;

endmodule
    
```



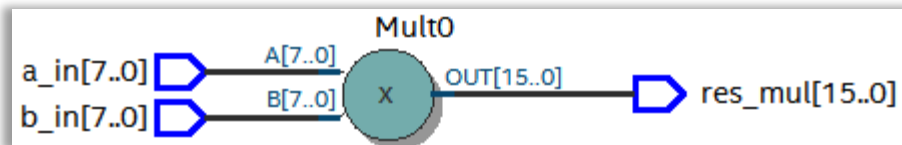
/ex1s_tb/a_in	104	119	120	121	122	123	124	125	126	127	-128	-127	-126	-125	-124	-123	-122
/ex1s_tb/b_in	104	119	120	121	122	123	124	125	126	127	-128	-127	-126	-125	-124	-123	-122
/ex1s_tb/res_add	208	238	240	242	244	246	248	250	252	254	-256	-254	-252	-250	-248	-246	-244

Пример умножителя

```
module ex2
(  input [7:0] a_in, b_in,
  output [15:0] res_mul);

assign res_mul = a_in * b_in;

endmodule
```



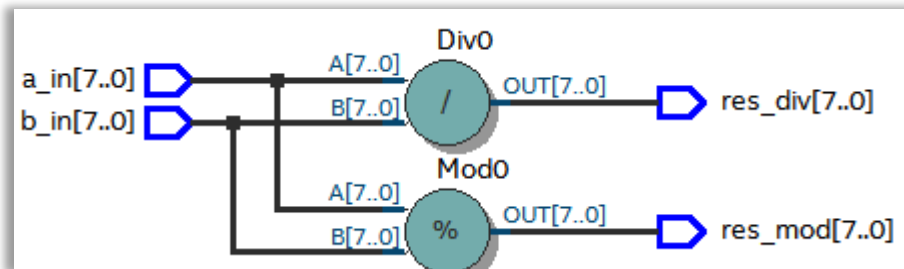
Edit:/ex2/a_in	10	0	1	2	3	4	5	6	7	8	9
Edit:/ex2/b_in	245	255	254	253	252	251	250	249	248	247	246
sim:/ex2/res_mul	2450	0	254	506	756	1004	1250	1494	1736	1976	2214

Пример делителя и модуля числа

```
module ex3
(  input  [7:0] a_in, b_in,
  output [7:0] res_div, res_mod);

assign res_div = a_in / b_in;
assign res_mod = a_in % b_in;

endmodule
```

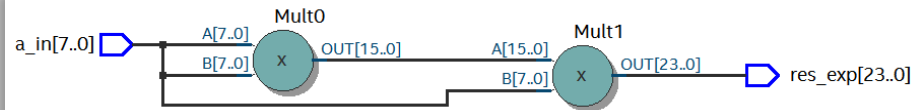


+ Edit:/ex3/a_in	255	255											
+ Edit:/ex3/b_in	0	255	254	253	252	251	0	1	2	3	4	5	6
+ sim:/ex3/res_div	x	1						255	127	85	63	51	42
+ sim:/ex3/res_mod	x	0	1	2	3	4		0	1	0	3	0	3

Пример возведения в степень

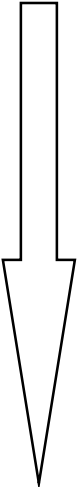
```
module ex4
(
  input  [7:0] a_in,
  output [23:0] res_exp);

  assign res_exp = a_in ** 3;
endmodule
```



+	Edit:/ex4/a_in	12	0	1	2	3	4	5	6	7	8	9	10	11
+	sim:/ex4/res_exp	1728	0	1	8	27	64	125	216	343	512	729	1000	1331

Приоритет операторов

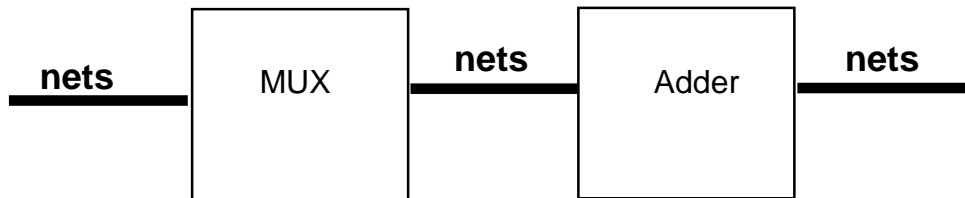
Операторы	Приоритет
+ - ! ~ & ~& etc. (unary* operators)	<div>Высокий</div>  <div>Низкий</div>
**	
* / %	
+ - (binary operators)	
<< >> <<< >>>	
< > <= >=	
== != === !==	
& (binary operator)	
^ ~^ ^~ (binary operators)	
(binary operator)	
&&	
?:	
{ } { }	

Рекомендуется использовать () для изменения порядка выполнения операторов

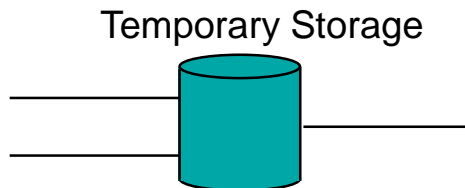
Группа типов данных Variable

Типы данных (напоминание)

- Группа типов данных Net – представляет физическую связь между элементами структуры



- Группа типов данных Variable – представляет элементы для временного хранения данных



*Тип **reg** не соответствует физическому триггеру*

Типы данных группы Variable

Типы данных	Для чего используется	Поддержка синтеза
reg	Переменная. Для знакового представления используйте reg signed.	Y
integer	Знаковая переменная (обычно 32 бита)	Y
time	Без знаковое целое (обычно 64 бита) используется для хранения времени при моделировании	N
real	Переменная с плавающей запятой, двойной точности	N
realtime	Переменная с плавающей запятой, двойной точности, используемая с time	N

Два типа процедурных блоков

❑ **initial**

- ✓ Используется для инициализации начальных значений

❑ **always**

- ✓ Используется для задания алгоритма работы устройства

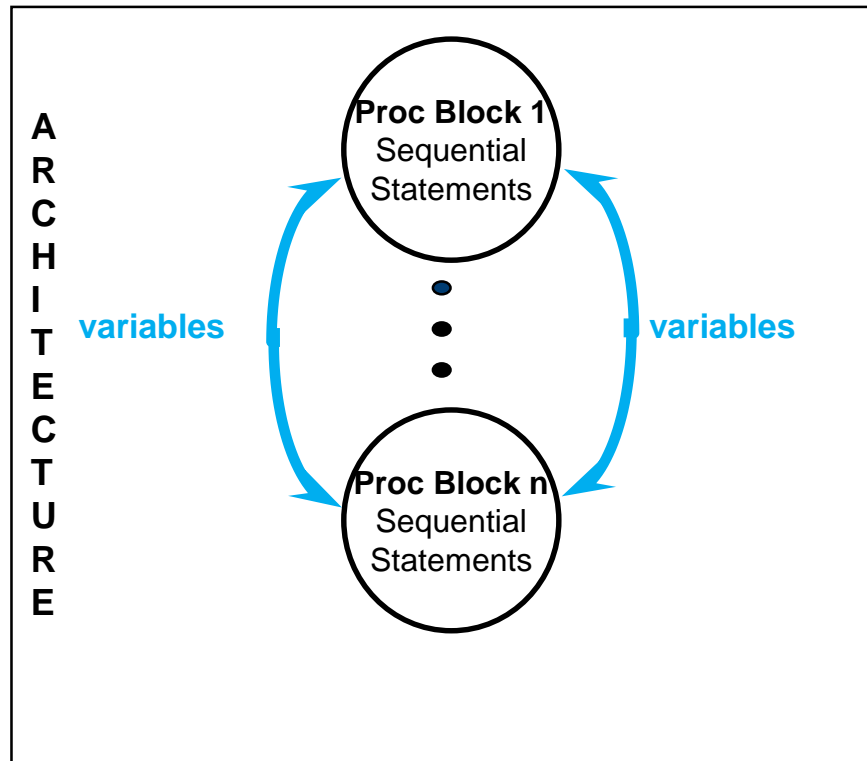
❑ Каждый always и initial блок - отдельный процесс

❑ always и initial не могут использоваться рекурсивно

❑ Особенности назначений

- ✓ LHS - тип данных из группы variable (reg, integer, real, time)
- ✓ RHS - типы данных групп net и variable или вызовы функций.

Процедурные блоки



- ❑ Каждый процедурный блок выполняется параллельно со всеми остальными блоками и непрерывными назначениями
 - ✓ Последовательность использования процедурных блоков `always/initial` не имеет значения
- ❑ Внутри процедурного блока операторы выполняются последовательно (`begin ... end`)
 - ✓ Последовательность использования операторов внутри процедурного блока имеет значение.

Процедурный блок initial

- ❑ Содержит поведенческое описание
- ❑ Каждый initial выполняется начиная с момента времени 0
 - ✓ выполняется только один раз и больше не выполняется
- ❑ Необходимо использовать ключевые слова begin и end (fork join) для группировки операторов (если блок содержит более одного оператора)
- ❑ Примеры использования
 - ✓ Инициализация сигналов при синтезе; Моделирование;
 - Любая функциональность, которая должна быть активирована один раз
- ❑ Обратите внимание:
 - ✓ хотя блок initial выполняется только один раз, но операторы внутри блока могут продолжать работать все время моделирования

Процедурный блок `always`

- ❑ Содержит поведенческое описание
- ❑ Каждый блок `always` выполняется начиная с момента времени 0
 - ✓ выполняется постоянно (циклически)
- ❑ Необходимо использовать ключевые слова `begin` и `end` (`fork join`) для группировки операторов (если блок содержит более одного оператора)
- ❑ Примеры использования
 - ✓ Описание для синтеза и моделирования
 - Любой процесс или функциональность, выполнение которых должно повторяться циклически

Именование процедурных блоков

- ❑ Процедурному блоку может быть присвоено имя
 - ✓ Необходимо использовать `begin ... end` (fork ... join) даже с одним оператором
 - ✓ после `begin (fork)` добавляется имя блока
- ❑ Преимущества
 - ✓ Позволяет ссылаться на процедурные блоки по именам
 - ✓ Позволяет декларировать локальные объекты для процедурного блока
 - ✓ Позволяет в системе моделирования осуществлять мониторинг процедурного блока по имени

initial

begin : `clock_init`

`clk = 1'b0;`

end

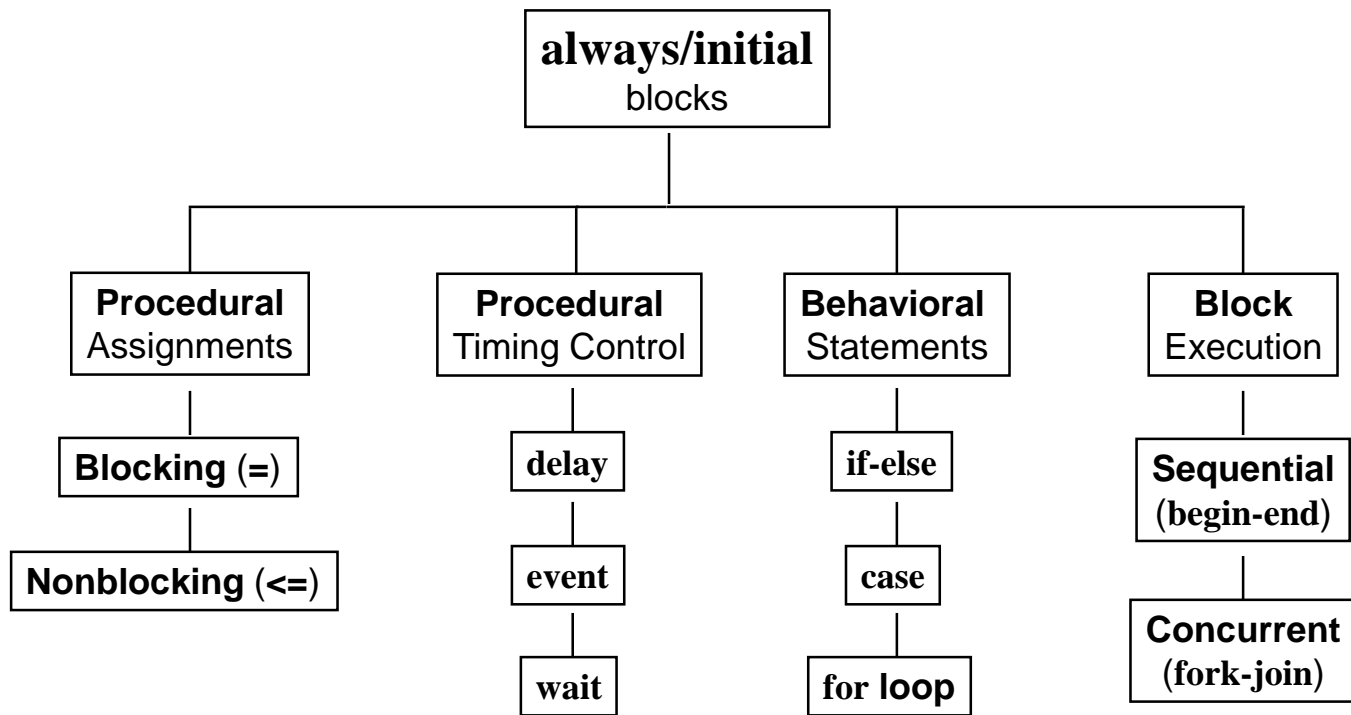
always

begin : `clock_proc`

`clk = ~clk;`

end

Особенности блоков always/initial



Выполнение процедурных блоков

- ❑ Все процедурные блоки (процессы) во всех модулях проекта выполняются вместе
 - ✓ Выполнение начинается в момент времени 0
 - ✓ После завершения всеми процессами выполнения текущего цикла (на данном временном шаге), модельное время увеличивается (переход к следующему временному шагу)
 - ✓ Текущий цикл выполнения заканчивается когда процедурный блок достигает одного из условий
 - Выполнен последний оператор блока initial
 - Выполняется блокирующее (Blocking) назначение с задержкой
 - Достигнут оператор с управлением событиями Event control
 - Достигнут оператор Wait statement
- ❑ Текущий цикл выполнения (текущий временной шаг) используется для определения поведения модели (описания модуля)
 - ✓ Simulation tool: Текущий временной шаг выполнения процесса соответствует времени моделирования
 - ✓ Synthesis tool: Порождает функциональность соответствующую физической аппаратуре.

Управление событиями Event Control

- ❑ Задается выражением `@(<event>)`
 - ✓ Выражение приостанавливает выполнение процедурных операторов до наступления события (до изменения значения выражения) у переменной из списка `<event>`
- ❑ Выражение `@(<event>)` может использоваться только в процедурных блоках `initial` или `always`
- ❑ Обеспечивает управление чувствительное к изменениям выражения:
 - ✓ `@(clk)` – выполнение приостанавливается до любого изменения `clk`
 - ✓ `@(posedge clk)` – выполнение приостанавливается до появления фронта `clk`
 - Фронт определяется как один из переходов : `0=>1`, `0=>x`, `0=>z`, `x=>1`, `z=>1`
 - ✓ `@(negedge clk)` – выполнение приостанавливается до появления спада `clk`
 - Фронт определяется как один из переходов : `1=>0`, `1=>x`, `x=>0`, `z=>0`
- ❑ Для проверки нескольких событий используют логическое OR:
 - ✓ Запятая (,) в Verilog '2001; or в Verilog 95.

Управление событиями и список чувствительности

- ❑ Использование `@(<event>)` в начале блока `always` позволяет контролировать момент начала выполнения блока
 - ✓ Для начала выполнения блока `always` требуется наступление события
 - ✓ Блок `always` становится “чувствительным” к переменным в списке `<event>`
- ❑ Список `<event>` называют – списком чувствительности (Sensitivity List)
- ❑ Использование управлением событиями поддержано средствами синтеза

Формат

```
always @(sensitivity_list) begin  
    -- Statement_1  
    -- .....  
    -- Statement_N  
end
```

Пример

```
// Процесс запускается если  
// значение любого из входов  
// а, b, c или d изменяется  
always @(a, b, c, d) begin  
    y = (a ^ b) & (c ~ | d);  
end
```

Два типа RTL процессов

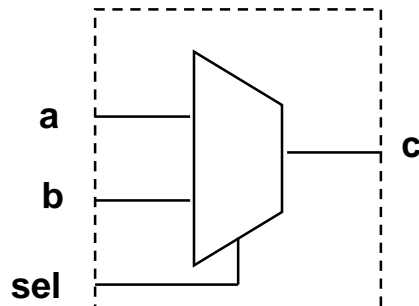
- Комбинационный процесс

- Чувствителен ко всем сигналам в процессе

```
always @ (a, b, sel)  
always @ *
```

** - добавить все входы*

Список чувствительности включает все входы комбинационной цепи



- Тактовый (регистровый) процесс

- Чувствителен к тактовым сигналам и сигналам управления

```
always @(posedge clk, negedge clr_n)
```

Список чувствительности не включает d вход, а только тактовый сигнал и сигнал асинхронного сброса

