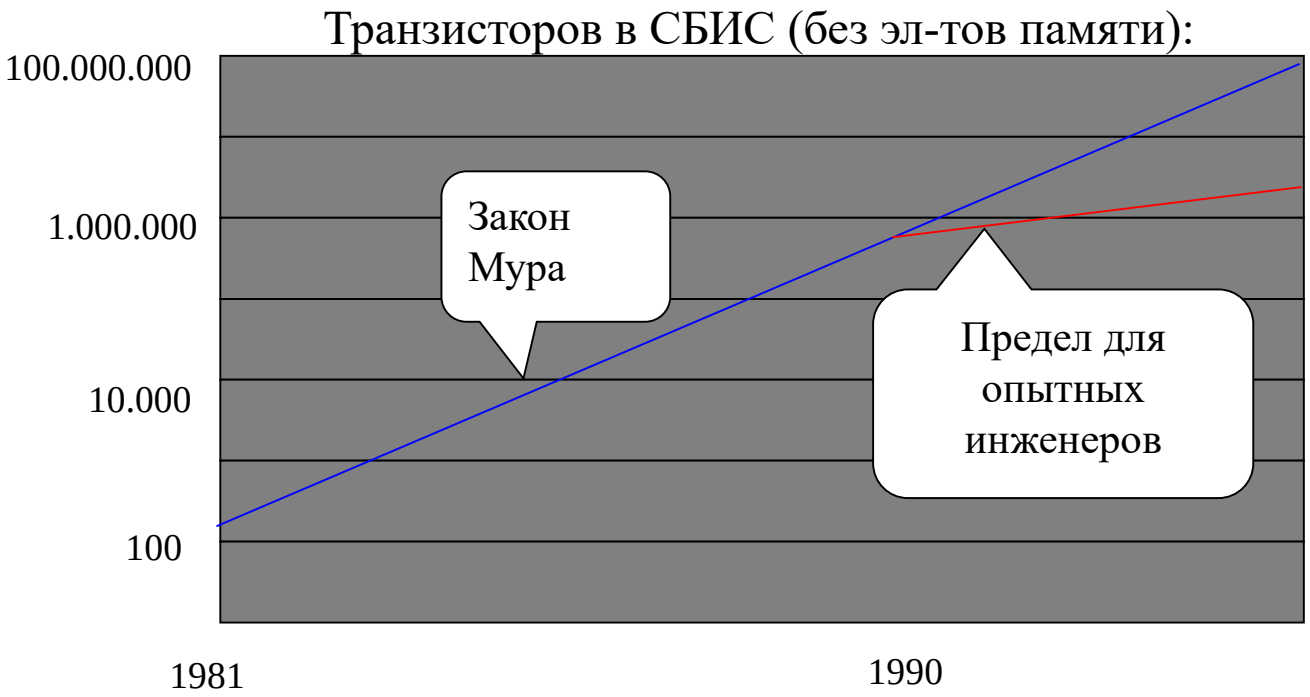
Основы VerilogHDL/SystemVerilog (синтез и моделирование)

alexander.antonov.ru@yandex.ru

Введение

Зачем нужны языки описания аппаратуры?

- Кризис производительности инженеров при использовании традиционных подходов к проектированию



Как борются с кризисом производительности?

- Переход к высокоуровневым средствам и методам проектирования:
  - ✓ Использование языков VHDL, **Verilog/SystemVerilog**
  - ✓ Использование высокоуровневого синтеза (HLS)
    - – Синтез описания на Си/C++ и на Си подобных языках (SystemC, CatapultC)
  - ✓ Использование библиотек параметризированных модулей
  - ✓ Использование библиотек готовых решений для алгоритмически сложных устройств – IP (Intellectual Property) модулей
- Совершенствование элементной базы:
  - ✓ Аппаратная реализация на кристаллах параметризуемых функционально сложных модулей (умножители, встроенные модули памяти, контроллеры внешней памяти, контроллеры Ethernet, контроллеры PCIe, интерфейсные модули…)

Что за язык VerilogHDL/System Verilog?

- Язык Verilog HDL / SystemVerilog - стандартизированный IEEE язык описания аппаратуры
  - ✓ Конкурент – язык VHDL
- Позволяет создавать:
  - ✓ Синтезируемые описания – на основе которых порождаются аппаратные средства.
  - ✓ Описание тестов – для проверки правильности работы синтезируемых описаний
    - – В дополнениях SystemVerilog много конструкций расширяющих возможности тестирования
- Похож на Си

История языка Verilog HDL / SystemVerilog

- Представлен в 1984 компанией Gateway Design Automation
- 1989 компания Cadence приобрела Gateway (Verilog-XL simulator)
- 1990 компания Cadence сделала язык Verilog общедоступным
- 1995 выпущен стандарт Verilog IEEE-1364-1995 (стандарт 95 года)
- 2001 выпущен стандарт Verilog IEEE-1364-2001 (стандарт 2001 года)
- 2005
  - ✓ Выпущен стандарт Verilog IEEE1364-2005 (стандарт Verilog2005)
  - ✓ SystemVerilog IEEE-1800-2005 Standard – надстройка над Verilog2005
- 2009 выпущен стандарт SystemVerilog – IEEE-1800-2009
  - ✓ Объединение – Verilog 2005 & System Verilog 2005
- 2012 выпущен стандарт SystemVerilog IEEE-1800-2012

Конструкции стандарта SystemVerilog IEEE-1800-2012

- Расширение набора языковых конструкций

SystemVerilog IEEE 1800

Literals	Arrays	Classes	Statements	Processes	Random	Clocking	Coverage	Interfaces
time	packed	new	unique	always_comb	rand/randc	##delay	covergroup	interface
string	unpacked	static	priority	always_latch	constraint		coverpoint	virtual
pattern	dynamic	this/super	do while	always_ff	randomize()	Assertions	cross	modport
	associative	extends	foreach	join_any	solve before	assert	wildcard	export
Types	queue	protected	return	join_none	dist with	assume	sequence	import
logic		cast	break	wait fork	rand_mode	cover	bins	
bit	Declarations	local	continue		constraint_mode	property	illegal_bins	DPI
byte	const		final	Subroutines	randcase	sequence		export
shortint	alias	Operators	iff	static	randsequence	intersect	Hierarchy	import
int	type	assignment		automatic		first_match	package	context
longint	var	wild equality		const ref	Synchronization	throughout	import	pure
shortreal		tagged		void	semaphore	within	nesting	
void		overloading			mailbox	bind		Program

Verilog-2001

ANSI C style ports	standard file I/O	`` (power operator)	multi dimensional arrays
generate	$value$plusargs	configurations	signed types
localparam	`ifndef `elsif `line	memory part selects	automatic
constant functions	@*	variable part select	

Verilog-1995

modules	$finish $fopen $fclose	initial	wire reg	parameters	+ = * /
parameters	$display $write	disable	integer real	function/tasks	%
function/tasks	$monitor	events	time	always @	>> <<
always @	`define `ifdef `else	wait # @	packed arrays	assign	
assign	`include `timescale	fork–join	2D memory		

Назначение конструкций SystemVerilog IEEE-1800-2012

C-Interfaces

Direct Programming Interface
C Modules

Object Oriented Verification Ext.

Classes, inheritance, data hiding and encapsulation, aggregation, polymorphism, parameterized classes

Primary Verification Extensions

Dynamic data and processes
SystemVerilog Assertions
Randomization and Constraints
Coverage

Increase Verification Efficiency!

Design Subset

Language Constructs
Static Data Types
Port Interfaces
Clocking Domains

Express more with less code!
Increase synthesis quality!

Verilog-2001

ANSI C style ports
generate
localparam
constant functions

standard file I/O
$value$plusargs
`ifndef `elsif `line
@*

(* attributes *)
configurations
memory part selects
variable part select

multi dimensional arrays
signed types
automatic
** (power operator)

Verilog-1995

modules
parameters
function/tasks
always @
assign

$finish $fopen $fclose
$display $write
$monitor
`define `ifdef `else
`include `timescale

initial
disable
events
wait # @
fork–join

wire reg
integer real
time
packed arrays
2D memory

modules
parameters
function/tasks
always @
assign

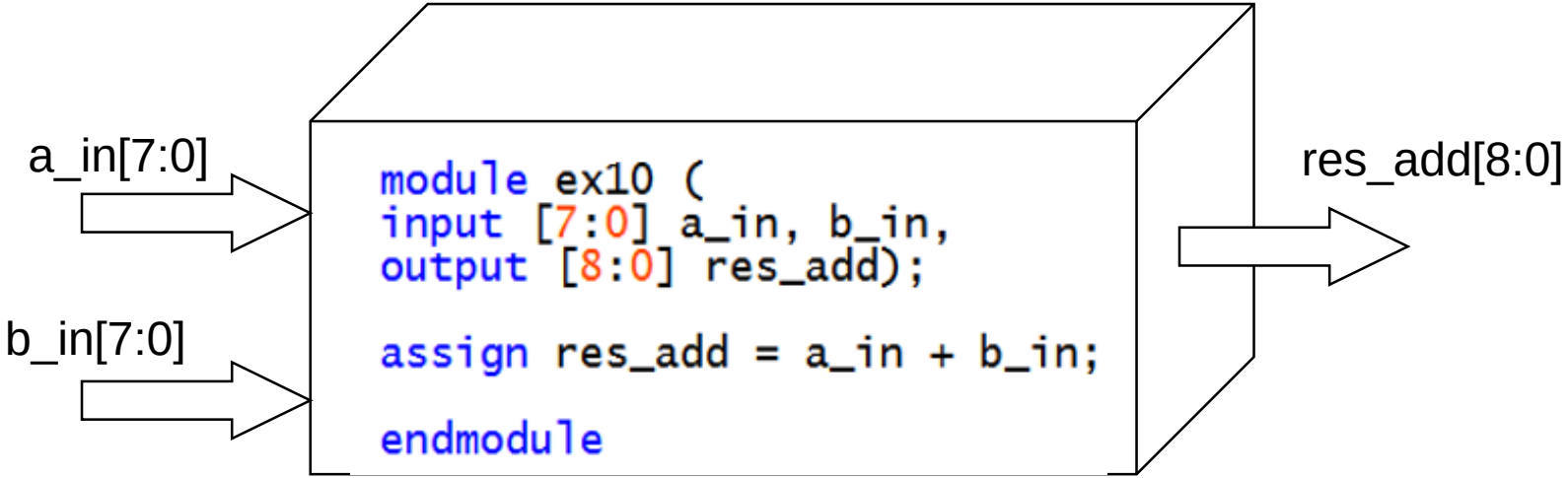
+ = * /
%
>> <<

Почему начинаем изучать Verilog’95 и Verilog 2001

- Verilog IEEE-1364-1995 (Verilog’95) и Verilog IEEE-1364-2001 (Verilog 2001) часть стандарта SystemVerilog IEEE-1800-2012
- Необходимо понимать базовые конструкции так как:
  - ✓ На Verilog’95/2001 написано огромное количество модулей
  - ✓ При описании IP чаще всего используются конструкции Verilog 2001
  - ✓ Создаваемые (средствами проектирования) списки соединений (netlist) используют конструкции Verilog’95/2001
- Verilog’95/2001 позволяет создавать достаточно сложные тесты
  - ✓ Поддерживается ModelSim ASE
- **С другой стороны** набор дополнительных конструкций стандарта SystemVerilog IEEE-1800-2012 (часто называемый SystemVerilog 1800) позволяет увеличить эффективность описания синтезируемых модулей и реализовать весьма эффективные новые подходы к моделированию.

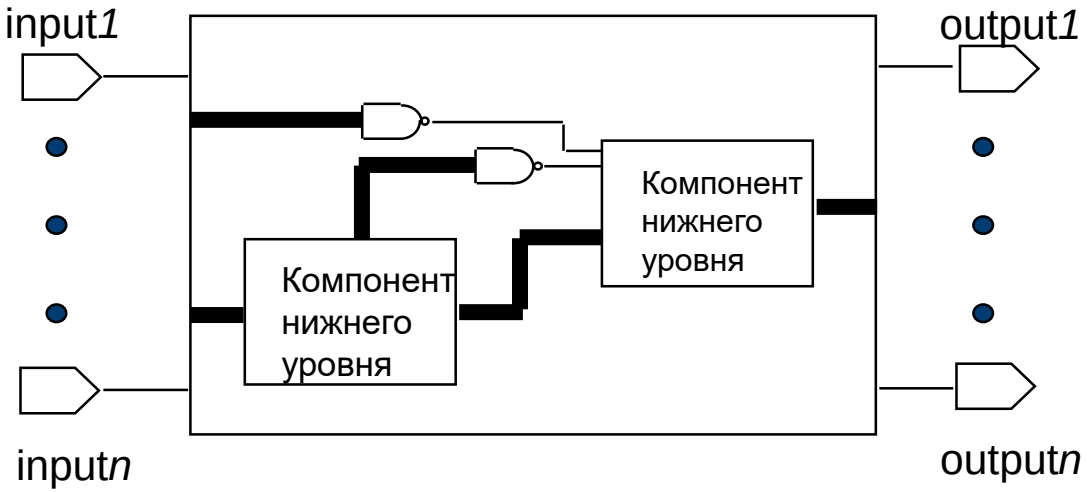
Термины (поведенческое описание)

- Поведенческое описание – описание алгоритма работы устройства
  - ✓ Описывается алгоритм работы, но не структура.
  - ✓ Используется для синтеза и моделирования



Термины (структурное описание)

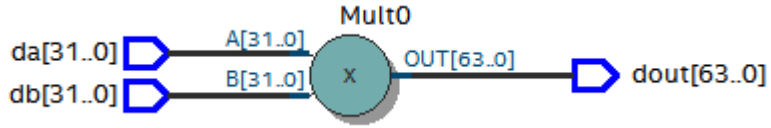
- Структурное описание – описание проектируемого модуля в виде взаимосвязанных компонентов более низкого уровня в иерархии описания.
  - ✓ Описывается структура, а не алгоритм работы
  - ✓ Используется для синтеза и моделирования

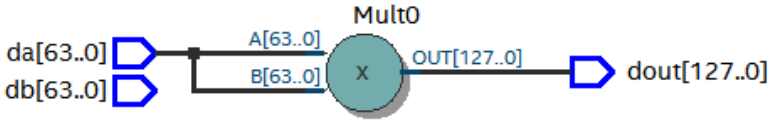


Термины (параметризируемое описание)

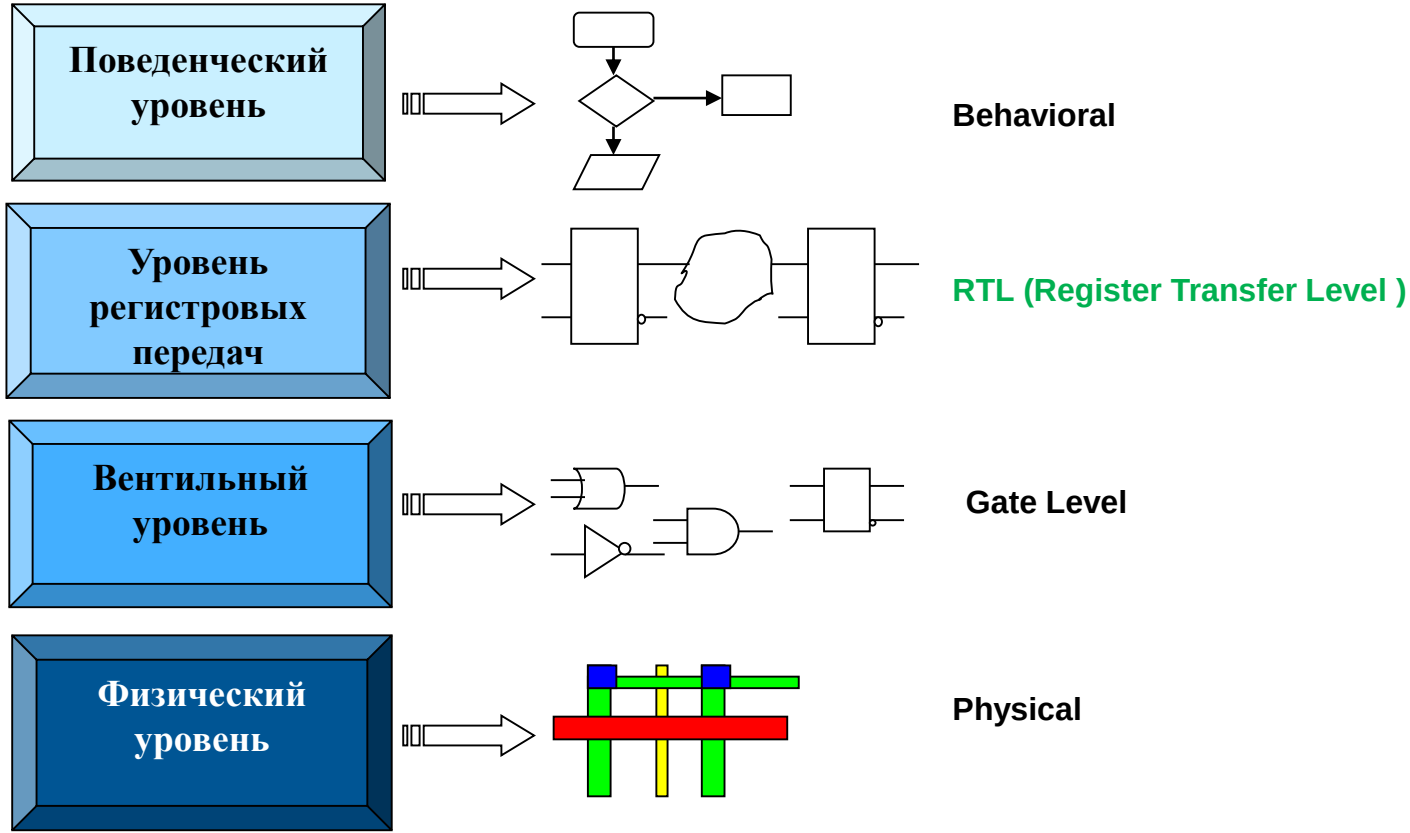
- Параметризироанное описание – описание, которое позволяет изменять (настраивать) структуру и характеристики описанных модулей
  - ✓ Может быть структурным и поведенческим
  - ✓ Используется для синтеза и моделирования

```
module param_mult
#(parameter WIDTH=64, TYPE="Power")
(  input [WIDTH-1:0] da,
   input [WIDTH-1:0] db,
   output [2*WIDTH-1:0] dout);

assign dout=(TYPE=="MULT")?(da*db):(da*da);

endmodule
```





Терминология (уровни описания проекта)



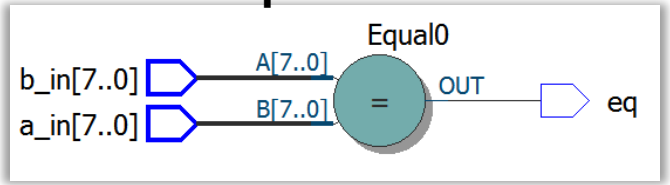
Терминология (синтез)

- **Синтез:** Преобразование описания проекта в схему на заданном элементном базисе (выбранной СБИС ПЛ).

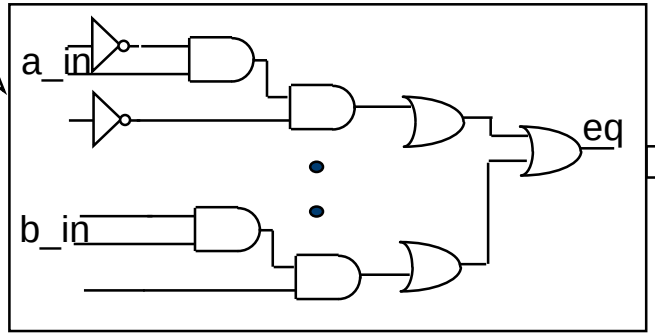
```
module ex6_c
(  input [7:0] a_in, b_in,
   output eq);
assign eq = a_in == b_in;
endmodule
```

Infer

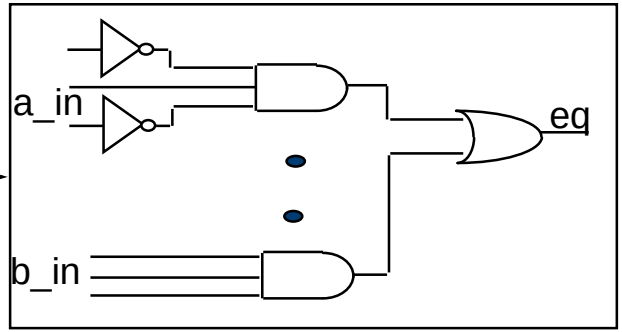
IP component



Translation



Optimization



Gates

Термины (моделирование)

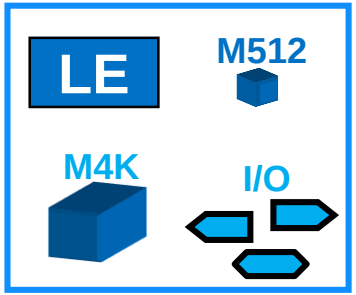
- Функциональное (поведенческое, RTL) моделирование – моделирование алгоритма работы:
  - ✓ с 0-ми задержками блоков и соединений блоков
- Временн**о**е моделирование – моделирование с учетом временных задержек (
  - ✓ моделируется работа на основе полученного списка соединений и SDF файла с временными задержками
    - – После синтеза (Post Synthesis Simulation)
      - – учитываются особенности синтеза, задержки соединений блоков либо не учитываются, либо используются усредненные.
    - – После трассировки (Post Place and Route Simulation)
      - – учитываются все задержки
- Тестовое воздействие, обычно, одно и тоже.

Процедура проектирования в Quartus Prime (1)

Техническое задание

Создание проекта
Ввод описания проекта: Verilog/SystemVerilog

Функциональное моделирование:
- **Средствами пакета Quartus**
- **Пакет ModelSim AE (ASE)**

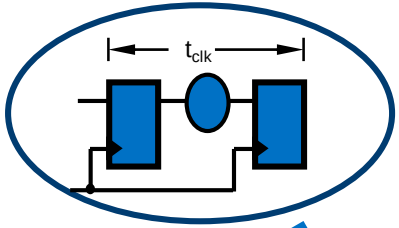


Синтез средствами пакета Quartus
- Выполняется оптимизация схемы с учетом ограничений по быстродействию и занимаемой площади СБИС

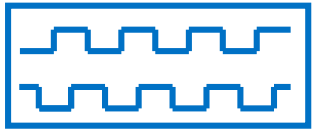
Quartus Prime
Design Software

Трассировка СБИС
Разводка и размещение внутренних ресурсов СБИС с учетом наложенных ограничений на быстродействие и занимаемые логические ресурсы

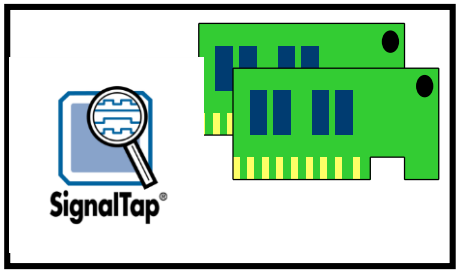
Процедура проектирования в Quartus Prime (2)



Временной анализ
- проверка соответствия созданной СБИС требованиям к быстродействию

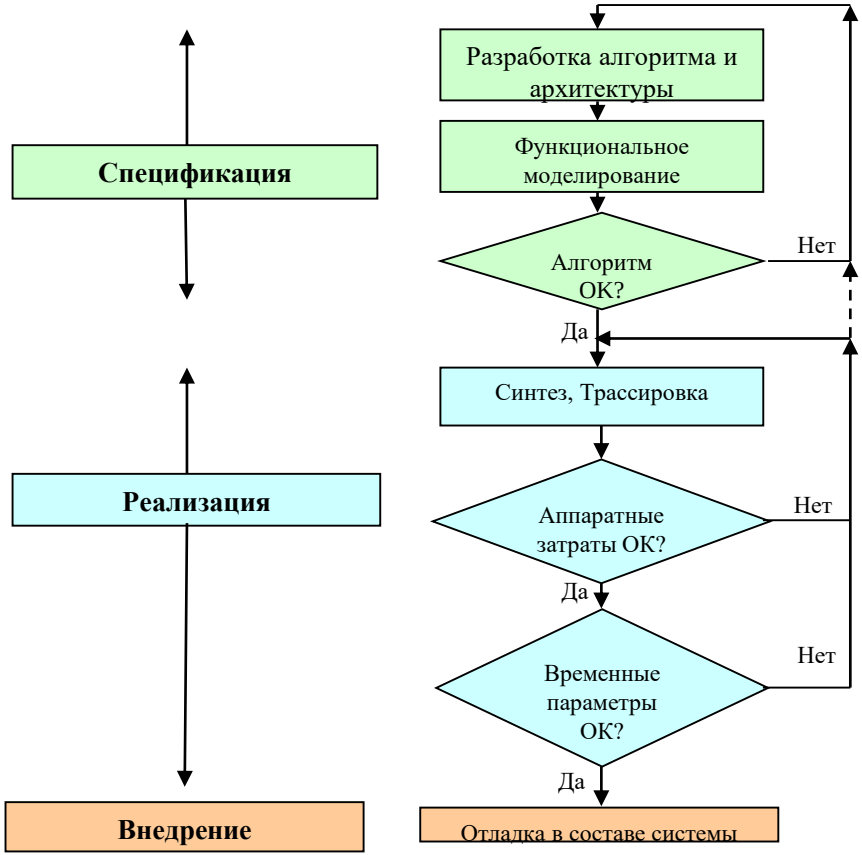


Временное моделирование: ОПЦИОНАЛЬНО
- **Средствами пакета Quartus** Чаще всего
- **Пакет ModelSim AE (ASE)** пропускается

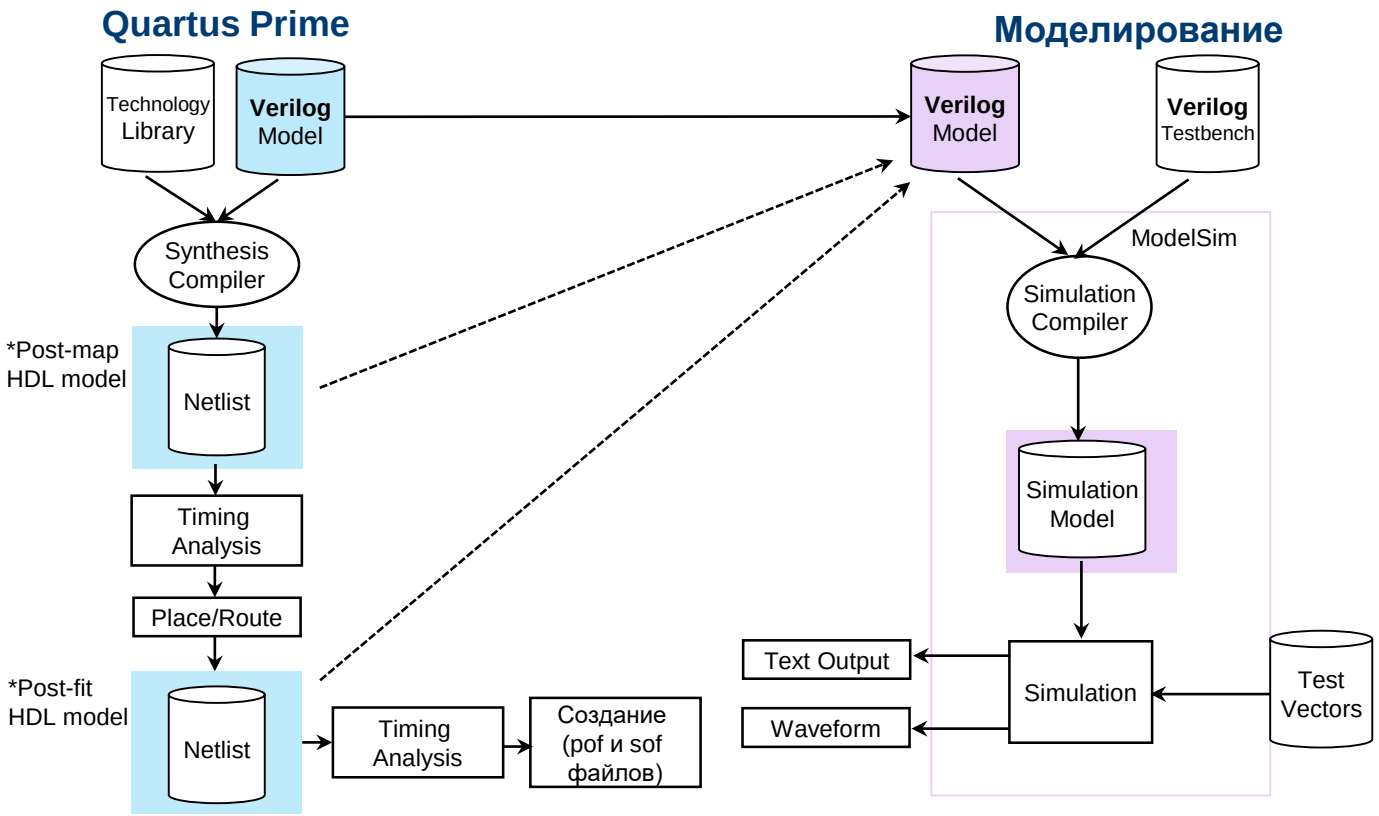


Программирование СБИС. Тестирование и отладка СБИС в составе системы
(ISP, SignalTap II)

Этапы проектирования



Процедура при использовании пакета ModelSim



Основные концепции языка

- Может использоваться:
  - ✓ для описания проектируемого модуля (для синтеза)
  - ✓ для описания тестов (для моделирования).
- Поддерживает способы описания:
  - ✓ поведенческое, структурное, смешанное
- Операторы выполняются параллельно (есть исключения)
- Позволяет создавать
  - ✓ Конфигурируемые описания
  - ✓ Иерархические описания

Вводные замечания

- Язык «похож» на язык Си
- Чувствителен к «регистру»: все ключевые слова – нижний регистр.
- Комментарии:
  - ✓ Одной строки - все от символов ***//*** до конца строки
  - ✓ Нескольких строк – начало /* …. окончание */
- Содержит зарезервированные ключевые слова:
  - ✓ input // a Verilog Keyword
  - ✓ wire // a Verilog Keyword
  - ✓ WIRE // a unique name (not a keyword)
  - ✓ Wire // a unique name (not a keyword)

Не следует использовать ключевые слова как имена, даже если они отличаются регистром

Вводные замечания

- Имена:
  - ✓ Должны начинаться с буквенного символа или подчеркивания (a-z A-Z _)
  - ✓ Могут содержать буквенные символы, числовые символы, подчеркивание, символ доллара (a-z A-Z _ 0-9 $)
  - ✓ Могут содержать до 1024 символов
- Язык допускает свободное использование пробелов, табуляции и переносов строк для улучшения читаемости текста.
- Операторы заканчиваются точкой с запятой (;)

Числа

Числа

- В языке определены
  - ✓ вещественные числа
  - ✓ целые числа
- Вещественные (не поддерживаются системой синтеза Quartus)
  - ✓ числа задаются по стандарту IEEE Std 754-1985 для чисел с плавающей запятой двойной точности
  - ✓ могут быть преобразованы в целые
  - ✓ могут быть представлены в:
    - – Десятичном виде: < value >.< value >
    - – Экспоненциальном виде: < mantissa >E< exponent >
  - ✓ округляются до ближайшего целого (когда присваиваются целому)

Целые числа

- Целые числа могут задаваться двумя способами, как:
  - ✓ **Simple decimal integer** – 32 битные знаковые числа. (Пример: 5)
  - ✓ **Based literal**
    - Формат <*Size*>’ <*Sign*> <*Base*> <*value*>
    - *Size* - значение определяет разрядность числа в битах
      - Если не задано – число 32 бита.
    - *Sign* **-** s (S), если задано, то число знаковое в дополнительном коде.
    - *Base* - формат числа (не чувствителен к регистру)
      - Decimal (’d or ’D) - десятичное
      - Hexadecimal (’h or ’H) - шестнадцатеричное
      - Binary (’b or ’B) - двоичное
      - Octal (’o or ’O) - восьмеричное

Целые числа (примеры)

Целое	Хранимое значение
1	00000000000000000000000000000001
8'hAA	10101010
6'b100011	100011
'hF	00000000000000000000000000001111

Дополнительные символы

- Дополнительные символы при задании чисел
  - ✓ ‘_’ (подчеркивание): используется для улучшения читаемости
    - – пример: 32’h21_65_bc_fe = 32-х битовое число в 16-ричном формате
  - ✓ ‘x’ или ‘X’ (неизвестное значение)
    - – Пример: 12’h12x = 12-х разрядное 16-ричное число; значение четырех младших разрядов – неизвестно
  - ✓ ‘z’ или ‘Z’ (high impedance)
    - – Пример: 1’bz = 1-разрядное число с high impedance
  - ✓ ‘?’ аналог ‘z’
    - – Пример: 1’bz = 1-разрядное число с high impedance

Отбрасывание старших разрядов

- Если количества бит недостаточно для представления числа, то старшие разряды числа отбрасываются:
  - ✓ 3’d8 отброшен старший разряд (1000 =>000) = 0
  - ✓ 4'sh86 отброшены 4 старших разряда (1000 0110 =>0110) =6
  - ✓ -4'h86 отброшены 4 старших разряда (1000 0110 =>0110) и получен дополнительный код числа 6 (1010) =-6
  - ✓ -4‘sh86 отброшены 4 старших разряда (1000 0110 =>0110) и получен дополнительный код числа 6 (1010) =-6
  - ✓ -4’h1a = -4’ha = 0110 – доп.код числа a
  - ✓ -4’sh1a = -(4’ha) = -(1010 – доп.код числа 6 = -6)=6

Заполнение старших разрядов

- Если старший разряд числа 0, x, z, то число заполняется 0, x, z, соответственно:
  - ✓ 3’b01 = 3’b001
  - ✓ 3’bx1 = 3’bxx1
  - ✓ 3’bz = 3’bzzz
- Если старший разряд числа 1, то число заполняется 0:
  - ✓ 3’b1 = 3’b001

Целые числа (примеры)

number	stored value	comment
5'b11010	11010	
5'b11_010	11010	_ ignored
5'o32	11010	
5'h1a	11010	
5'd26	11010	
5'b0	00000	0 extended
5'b1	00001	0 extended
5'bz	zzzzz	z extended
5'bx	xxxxx	x extended
5'bx01	xxx01	x extended

Знаковые и отрицательные числа

- Числа с признаком s(S) – знаковые числа, представленные в дополнительном коде
- Старший разряд – знаковый разряд:
  - ✓ 4’sb1111 – будет представлено как 1111 и = -1
  - ✓ 4’sb0111 – будет представлено как 0111 и = 7
- Отрицательные числа задаются знаком минус перед разделом <size>
  - ✓ -4’ b0001 = 4-х разрядное число, соответствующее дополнительному коду числа 0001 = -1
  - ✓ -4’ b1111 = 4-х разрядное число, соответствующее дополнительному коду числа 1111 = -15

Целые числа (примеры)

Целое	Хранимое значение	Значение
4’h4	0100	4
4’sh4	0100	4
4’ha	1010	10
4’sha	1010	-6
5’h9	01001	9
-5’h9	10111 (доп. код числа 9)	-9
5’sh9	01001	7
4’sh9	1001	-7
-4’sh9	0111 (-(4’sh9)=-(-7) =7)	7

Вопрос

- Равны ли числа -4’sb1111 и 4’b1111?

Ответ

- Равны ли числа -4’sb1111 и 4’b1111?

-4’sb1111 = -(-1) = 1

4’ b1111 = 15

Вопрос

- Равны ли числа 4’sb1111 и -4’b0001?

Ответ

- Равны ли числа 4’sb1111 и -4’b0001?

4’sb1111 = -1 (представлено как 1111)

-4’b0001 = -1 (представлено как 1111)

Вопрос

- Как будет храниться число 4’d17

(код в двоичной системе счисления) ?

Ответ

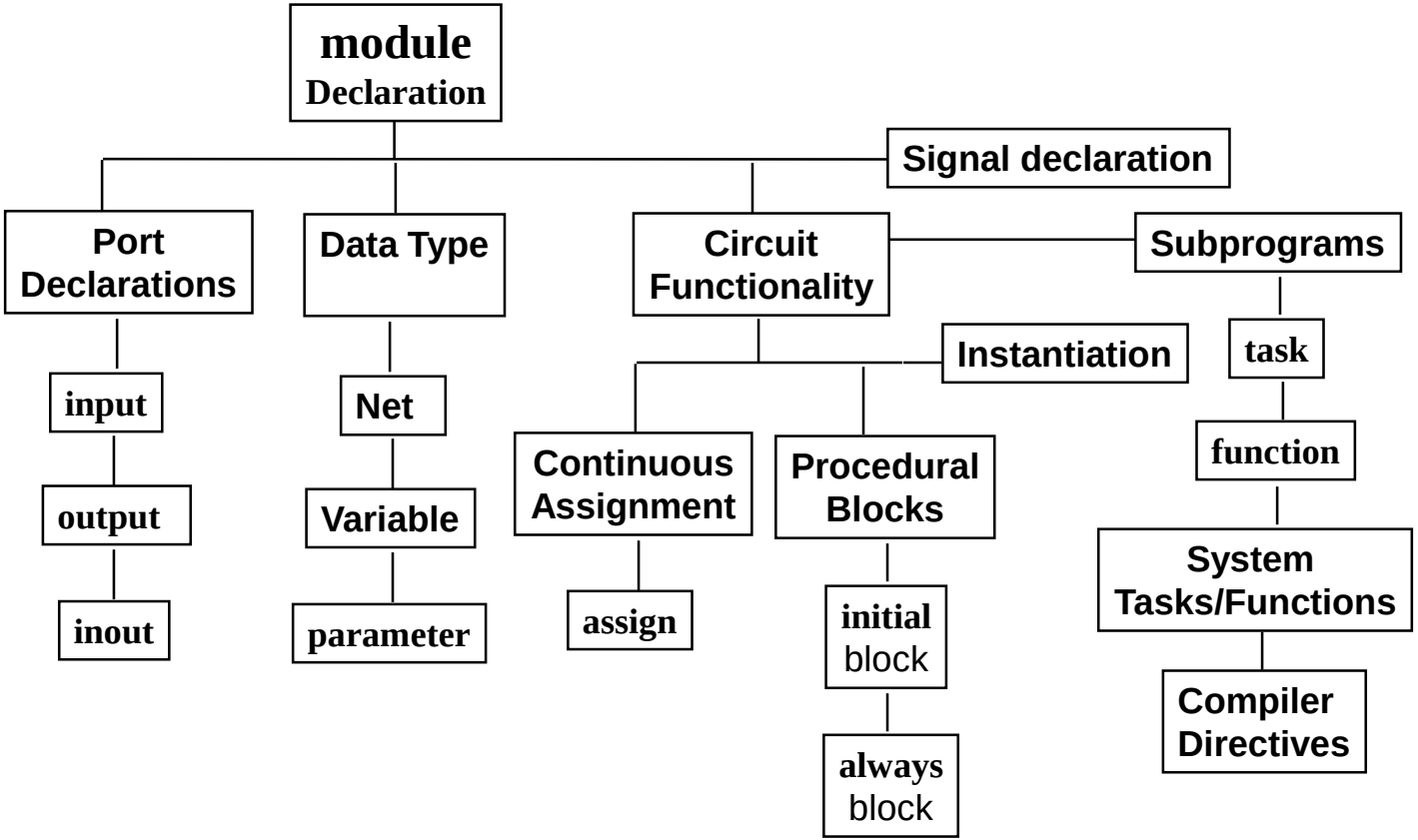
- Как будет храниться число 4’d17

(код в двоичной системе счисления) ?

MSB отброшен (10001 => 0001)

Основы языка

Что предстоит изучить



Модуль

- Модуль – основной элемент описания на языке Verilog\SystemVerilog
- Структура модуля:

Verilog 95

module *module_name (port_list);*

port declarations

signal declarations

circuit functionality

endmodule

Verilog 2001

module *module_name*
(port declarations);

signal declarations

circuit functionality

endmodule

Группы типов данных

- **Net** – типы данных группы представляют физическое соединение между структурными элементами
  - ✓ Используется для моделирования (описания) соединений в проекте
  - ✓ Тип wire– тип данных по умолчанию для всех input и output
  - ✓ Назначение сигналу с таким типом данных может быть реализовано только непрерывным назначением (**continuous assignment)**
- **Variable** – типы данных группы представляют элементы для временного хранения данных
  - ✓ Хранят значение при моделировании
  - ✓ Подобны переменным в обычных языках программирования
  - ✓ Не обязательно представляют триггеры (flip-flop; latch)
  - ✓ Назначение сигналу с таким типом данных может быть реализовано только в процедурном назначении **procedural assignment**

Набор значений

- В языке определено 4 значения:
  - ✓ 0 – логический ноль (false - ложь)
  - ✓ 1 – логическая единица (true - истина)
  - ✓ x – неизвестное значение
  - ✓ z – z состояние (плавающий уровень)

- Уровни сигнала (используются для разрешения конфликтов):

Название	Уровень силы
supply	высший
strong	
pull	
large	
weak	
medium	
small	
highz	низший

Типы данных класса Net

Тип данных	Для чего используется	Поддержка синтеза
wire	Используется для соединения модулей	Y
tri		Y
supply0	Представляют константные данные (питание и землю)	Y
supply1		Y
wand	Реализуют монтажную логику	Y
triand		Y
wor		Y
trior		Y
tri0	Цепь с z состоянием и резисторами pull-up/pull-down	Y
tri1		Y
trireg	Цепь, хранящая свое предыдущее значение	N

Объявление выводов (Port Declaration)

- Задается:
  - ✓ mode
    - **input** ⇒ input port
    - **output** ⇒ output port
    - **inout** ⇒ bidirectional port
  - ✓ data_type - тип данных.
    - Опционально.
    - По умолчанию – wire.
  - ✓ port_name - имя вывода
    - Одно или несколько имен через запятую
      - Если несколько, то все имеют одинаковый тип.

```
module [module_name]
    (
     [mode] [data_type] [port_names],
     [mode] [data_type] [port_names],
     . . .
     [mode] [data_type] [port_names]
    );
```

Примеры

- Стиль Verilog 2001

```
module eq1
    // I/O ports
    (
     input wire i0, i1,
     output wire eq
    );
```

- Стиль Verilog 2001
  - ✓ Тип данных wire задан по умолчанию

```
module eq1
    // I/O ports
    (
     input        i0, i1,
     output        eq
    );
```

- Стиль Verilog 95

```
module eq1 (i0, i1, eq);
    // declare mode
    input i0, i1;
    output eq;
```

Объявление сигнала (Signal declaration)

- Формат объявления внутреннего сигнала модуля

data_type <signed> <[range]> { signal_name <[array]>, …}

  - data_type – тип данных (из группы Net или Variable)
  - signed - <опционально> - признак знаковый
    - старший разряд – знак: 1-отрицательное число; 0-положительное
  - range - <опционально> - количество элементов вектора
    - если не задан, то сигнал – одноразрядная цепь
  - signal_name – имя цепи
  - array - <опционально> - количество элементов в массиве
    - массивы могут быть многомерными.

```
// signal declaration
wire p0, p1;
```

Побитовые логические операции (bitwise)

- В языке определено много типов операторов (будут изучаться позже).
  - ✓ один из них – bitwise – определяет логические операции, применяемые к каждому биту – побитовые логические операторы:

Символ оператора	Функция (применяется к каждому биту)
~	Инверсия
&	AND
|	OR
^	XOR
^~ or ~^	XNOR

Непрерывное назначение (continuous assignment)

- Синтаксис

```
assign [signal_name] = [expression];
```

  - ✓ assign – ключевое слово
  - ✓ signal_name – имя сигнала
    - – Тип сигнала – любой тип данных из группы Net
  - ✓ expression – присваиваемое выражение
    - – Может имить любой тип данных (из групп данных Net и Variable)
- Описывает поведение комбинационных схем
- Может быть сделано при объявлении сигнала (implicit continuous assignment)

```
wire  tmp;
assign tmp = d_in;
```

⟷

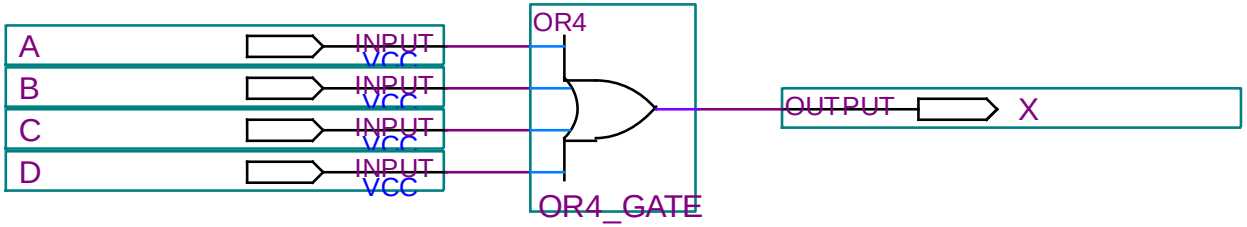
```
wire  tmp = d_in ;
```

- Все непрерывные назначения выполняются параллельно

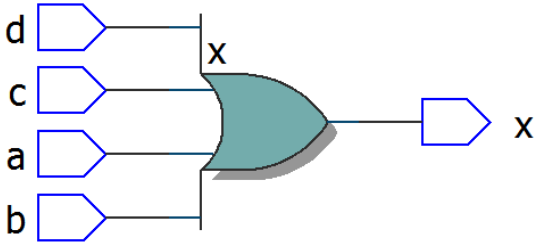
Особенности непрерывного назначения

- Левая часть оператора Left-Hand Side (LHS) должна иметь тип данных из группы Net
- При изменении любого операнда в правой части оператора - Right-Hand Side (RHS) оценивается значение выражения RHS и значение сигнала (LHS) обновляется
- RHS может быть выражением, содержащим тип net, тип variable, вызов функции (или их комбинации)
- В оператор может быть добавлена задержка (будет обсуждено позже)

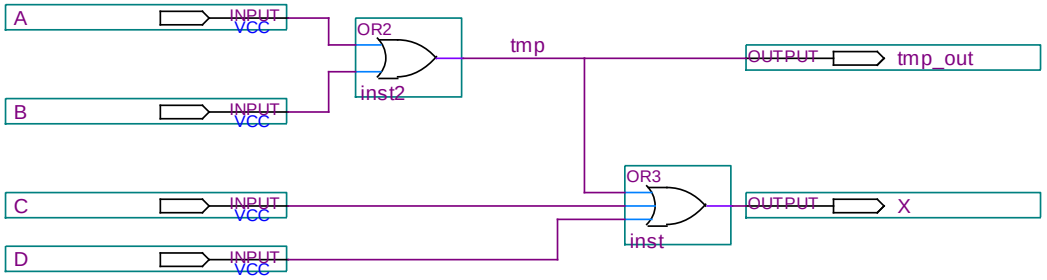
Пример описания модуля



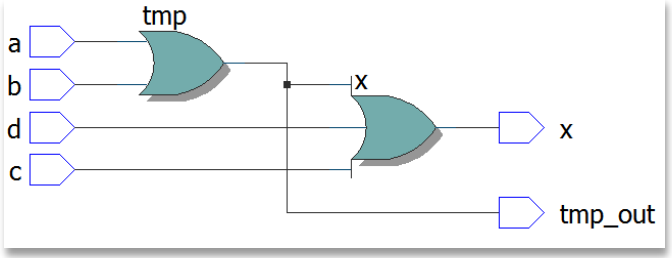

```
module or4_gate
(  input a, b, c, d,
   output x);

assign x=a | b | c | d;

endmodule
```



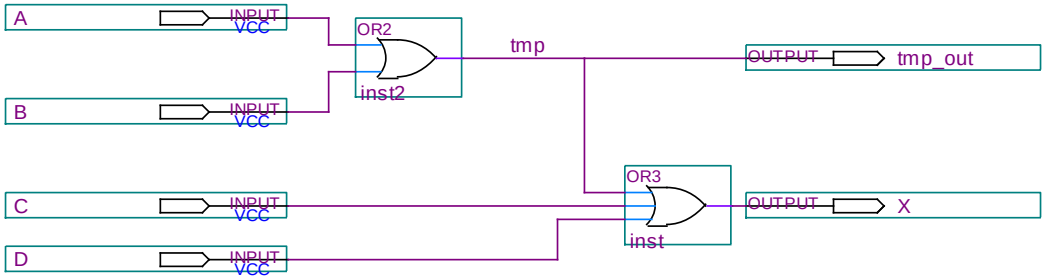
Явное объявление и использование сигнала



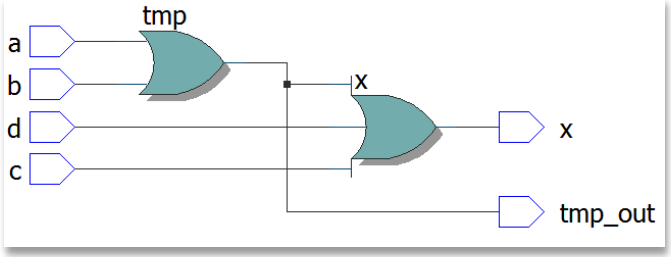

```
module or4_gate_s
(  input a, b, c, d,
   output x, tmp_out);

wire tmp;

assign tmp     = a|b;
assign tmp_out = tmp;
assign x       = tmp | c | d;

endmodule
```



Неявное объявление и использование сигнала




```
module or4_gate_si
(  input a, b, c, d,
   output x, tmp_out);

//wire tmp;

assign tmp     = a|b;
assign tmp_out = tmp;
assign x       = tmp | c | d;

endmodule
```

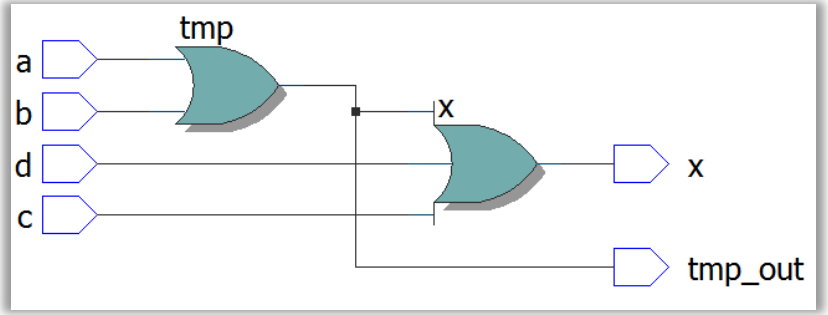


Type	ID	Message
⚠	10236	Verilog HDL Implicit Net warning at or4_gate_si.v(9): created implicit net for "tmp"

Результаты синтеза одинаковые или нет?

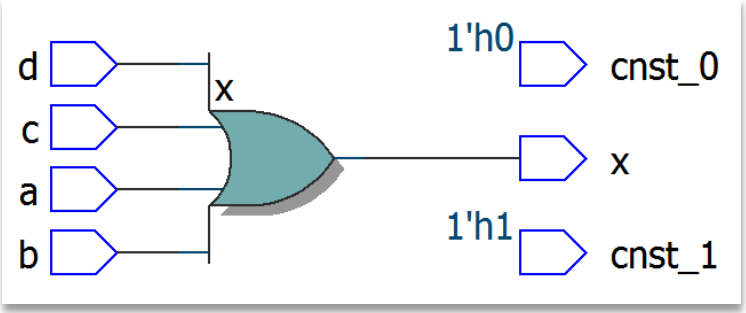
```
module or4_gate_s
(  input a, b, c, d,
   output x, tmp_out);

wire tmp;

assign tmp     = a|b;
assign tmp_out = tmp;
assign x       = tmp | c | d;

endmodule
```

```
module or4_gate_s
(  input a, b, c, d,
   output x, tmp_out);

wire tmp;

assign x       = tmp | c | d;
assign tmp_out = tmp;
assign tmp     = a|b;

endmodule
```



Задание константных значений

```
module or4_gate(
input a, b, c, d,
output x, cnst_0, cnst_1);

assign x=a | b | c | d;

assign cnst_1 = 1;
assign cnst_0 = 1'b0;

endmodule
```



Примеры объявления цепи

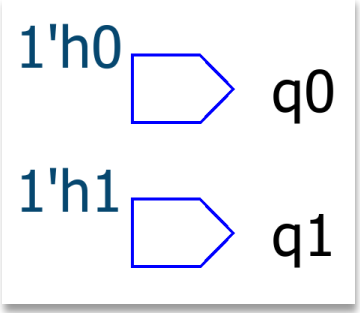
- Примеры:
  - wire clr_n;
  - wor temp;
  - supply0 cnst_0;
  - supply1 cnst_1;

- *Выводы* модуля по умолчанию имеют тип данных **wire,** но для *выходов* тип данных можно изменить:
  - output wor x;

Пример использования типа данных supply1, supply0:

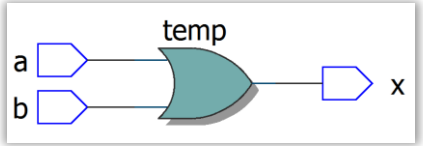
- *Выводы* модуля по умолчанию имеют тип данных **wire,** но для *выходов* тип данных можно изменить:
  - ✓ output wor x;
  - ✓ output supply1 y;

```
module ex2( q1, q0);

output  q1;
output supply0 q0;

supply1 tmp;

assign q1 = tmp;

endmodule
```



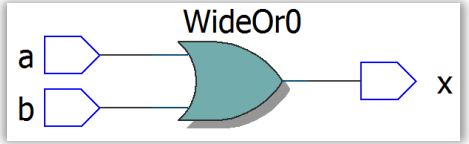
Пример использования типов данных wire и wor

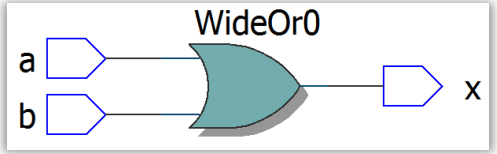
```
module ex2(a, b, x);
input a, b;
output   x;

wire temp;

assign temp = a | b;

assign x = temp;

endmodule
```




```
module ex2(a, b, x);
input a, b;
output   x;

wor temp;

assign temp = a;
assign temp = b;

assign x = temp;

endmodule
```

```
module ex2(a, b, x);
input a, b;
output  wor x;

assign x = a;
assign x = b;

endmodule
```





Пример неправильного назначения.

```
module ex2(a, b, x);
input a, b;
output   x;

wire temp;

assign temp = a;
assign temp = b;

assign x = temp;

endmodule
```

```
10031 Net "temp" at ex2.v(8) is already driven by input port "a", and cannot be driven by another signal
```

Вектор

Вектор

- Сигнал может быть объявлен как вектор (шина)

data_type <signed> <[range]> { signal_name, …}

- Две способа объявление вектора – диапазона индексов **range** :
  - ✓ [**high : low**] – убывающая последовательность индексов;
  - ✓ [**low : high**] – возрастающая последовательность индексов;
- Самый левый бит вектора всегда старший
  - ✓ Не зависимо от последовательности задания индексов
- Максимальный размер вектора – 2**32 разрядов
- Примеры:
  - ✓ wire [15:0] mult_out; // шина 16 бит; ; индекс старшего разряда - 15
  - ✓ wire [0:7] busA, busB; // две шина 8 бит; индекс старшего разряда - 0

Обращение к элементам вектора

- При одновременном обращении ко всем элементам вектора можно упустить квадратные скобки с индексами

```
wire [3:0] a, b;
assign a=b;
```

- При обращении к нескольким элементам вектора (**Constant Part select**) диапазон значений индексов указывают в квадратных скобках

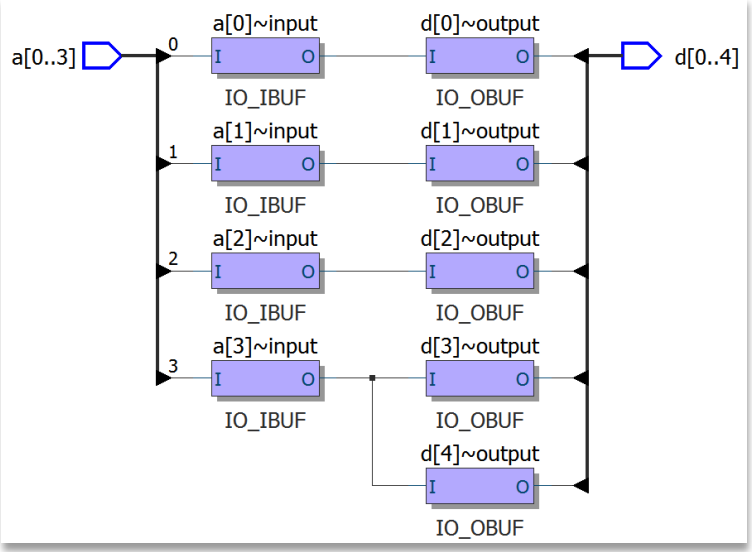
```
wire [3:0] a, b;
assign a[2:0]=b[3:1];
```

- При обращении к одному элементу вектора (**Bit select**) значение индекса указывают в квадратных скобках

```
wire [3:0] a, b;
assign a[3]=b[2];
```

Пример

```
module ex3(a, d);
input  [3:0] a;
output [4:0] d;

assign d[3:0] = a;
assign d[4] =a[3];

endmodule
```



Пример не правильного обращения к элементам вектора

- Нельзя менять порядок перечисления индексов вектора

```
module ex3(a, b);
input [3:0] a;
output [3:0] b;

assign b[0:3] = a;

endmodule
```

```
10198 Verilog HDL error at ex3.v(15): part-select direction is opposite from prefix index direction
```

Обращение к элементам вектора (Verilog2001)

- В Verilog2001 добавлен новый способ обращения к элементам вектора (**Variable Part Select**):

vector_name [starting_bit number + : width]

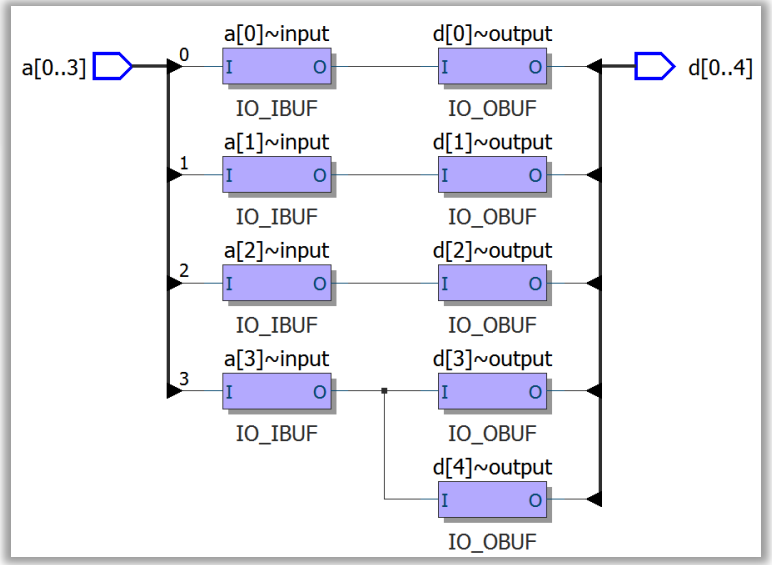
vector_name [starting_bit number - : width]

- ✓ starting_bit number – номер первого индекса
- ✓ +/- - увеличение/уменьшение индекса
- ✓ width – число элементов (включая элемент starting_bit number)

Пример

```
module ex3(a, d);
input  [3:0] a;
output [4:0] d;

assign d[3-:4] = a;
assign d[4-:1] =a[3+:1];

endmodule
```

```
module ex3(a, d);
input  [3:0] a;
output [4:0] d;

assign d[3:0] = a;
assign d[4] =a[3];

endmodule
```



Вопросы:

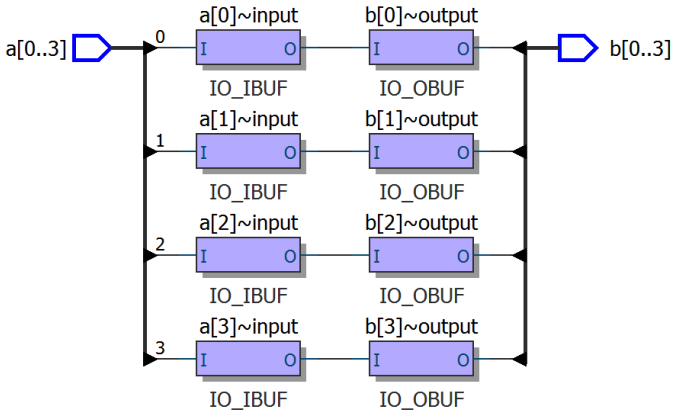
- Вопрос 1: Код правильный?

Да. Мы можем использовать любое направление счета индексов для **Variable Part Select**.

```
module ex3(a, b);
input [3:0] a;
output [3:0] b;

assign b[0+:4] = a;

endmodule
```

- Вопрос 2: Что будет подано на b[3] ?

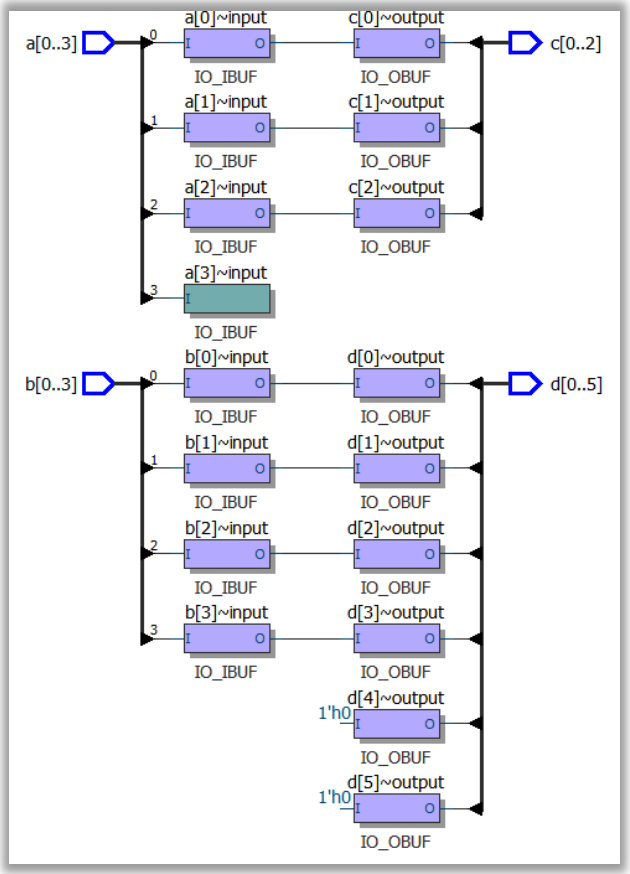
На b[3] будет подано a[3]; независимо от направления счета индексы будут использоваться в порядке объявления.



Обращение к элементам вектора (3)

- Если число разрядов в левой части выражения больше чем в правой, то недостающие разряды правой части дополняются нулями (старшие разряды).
- Если число разрядов в левой части выражения меньше чем в правой, то лишние разряды (старшие разряды) правой части отбрасываются

```
1  module ex4(a, b, c, d);
2  input [3:0] a, b;
3  output [2:0] c;
4  output [5:0] d;
5
6  assign c = a;
7  assign d = b;
8
9  endmodule
```



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 условного выбора - Conditional

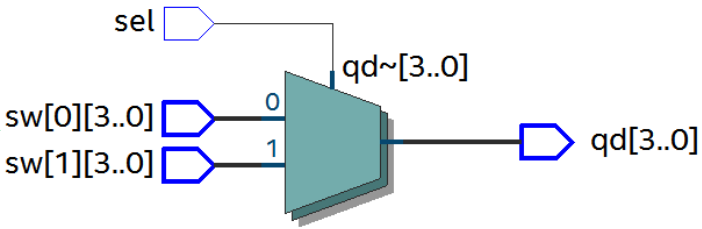
Символ	Функция	Формат и примеры
?:	Conditional	(condition) ? true_value : false_value

- 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
```

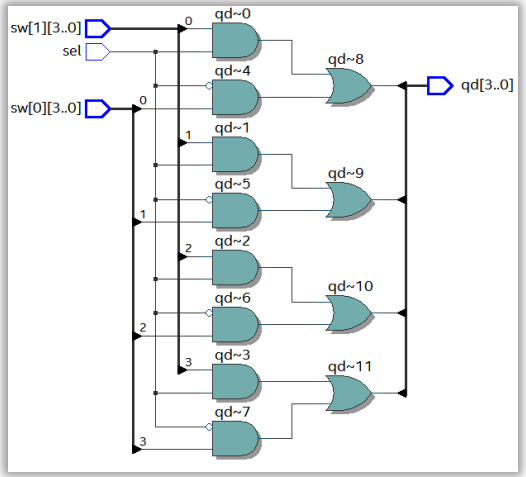
Подстановка



- 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



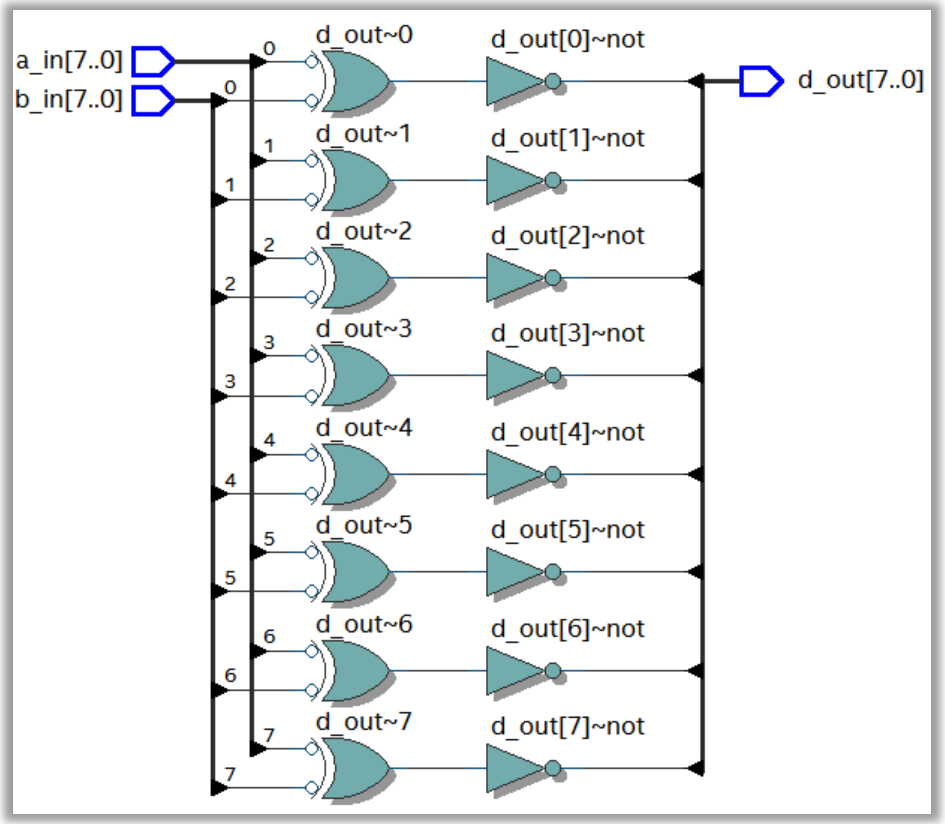
Побитовые операторы - Bitwise Operators

Символ	Функция	ain = 3’b101 ; bin = 3’b110 ; cin = 3’b01x	
~	Инверсия	~ain ⇒ 3b’010	~cin ⇒ 3’b10x
&	AND	ain & bin ⇒ 3’b100	bin & cin ⇒ 3’b010
|	OR	ain | bin ⇒ 3’b111	bin | cin ⇒ 3’b11x
^	XOR	ain ^ bin ⇒ 3’b011	bin ^ cin ⇒ 3’b10x
^~ or ~^	XNOR	ain ^~ bin ⇒ 3’b100	bin ~^ cin ⇒ 3’b01x

- 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
```



Приложение 1

Массивы

Массив

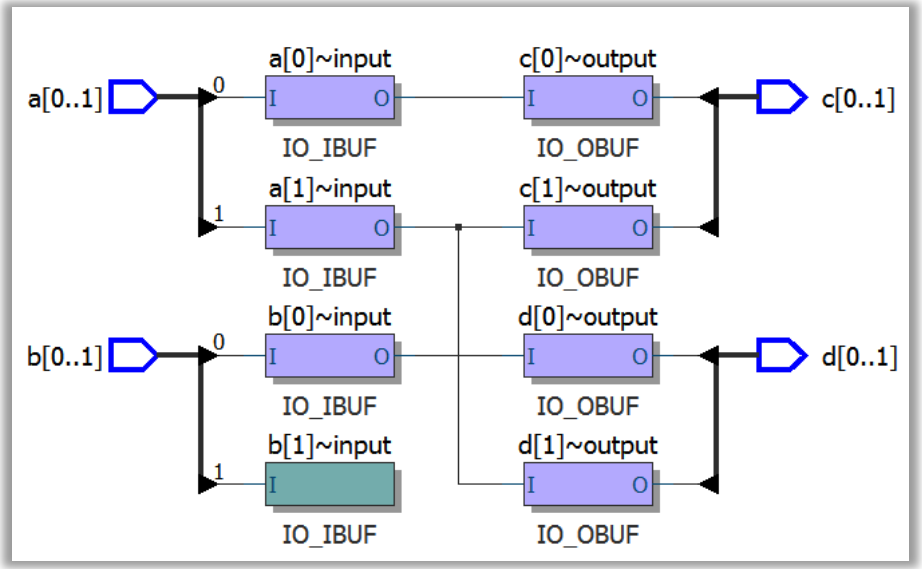
- В стандарте 2001 определен массив для класса Net.

data_type <signed> <[range]> { signal_name <[array]>, …}

- Размерность массива может быть любая
  - ✓ Через запятую указывается несколько наборов диапазонов индексов
- Диапазон индексов массива (для каждого измерения) может указываться в возрастающей и убывающей последовательности.
- Максимальный размер массива по каждому измерению
  - ✓ 2**24 элементов
- Если **range** (диапазон индексов вектора)
  - ✓ не задан – массив цепей (каждый элемент массива – сигнал),
  - ✓ задан – массив векторов (каждый элемент массива – вектор)

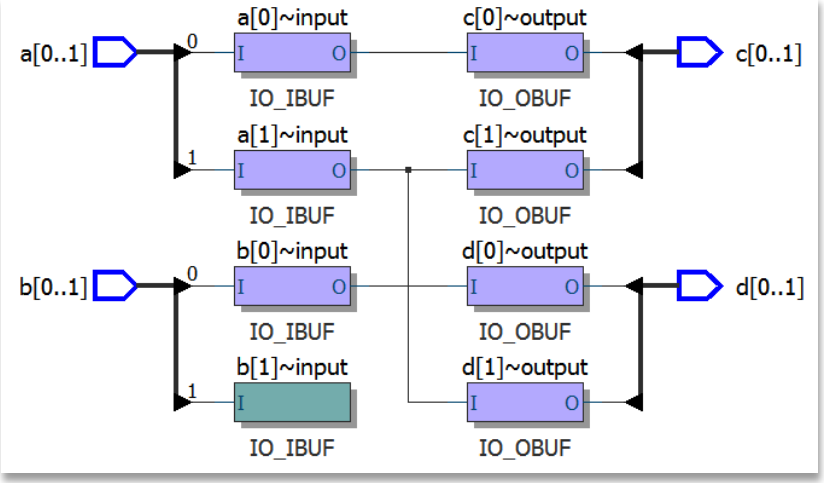
Пример объявления и использования массива векторов

```
1   module ex5(a, b, c, d);
2   input [1:0] a, b;
3   output [1:0] c;
4   output [1:0] d;
5
6   wire [1:0] mem [3:0];
7
8   assign mem[3] = a;
9
10  assign mem[2] = b;
11
12  assign d[1] = mem [3][1];
13
14  assign d[0] = mem [2][0];
15
16  assign c = mem[3];
17
18  endmodule
```



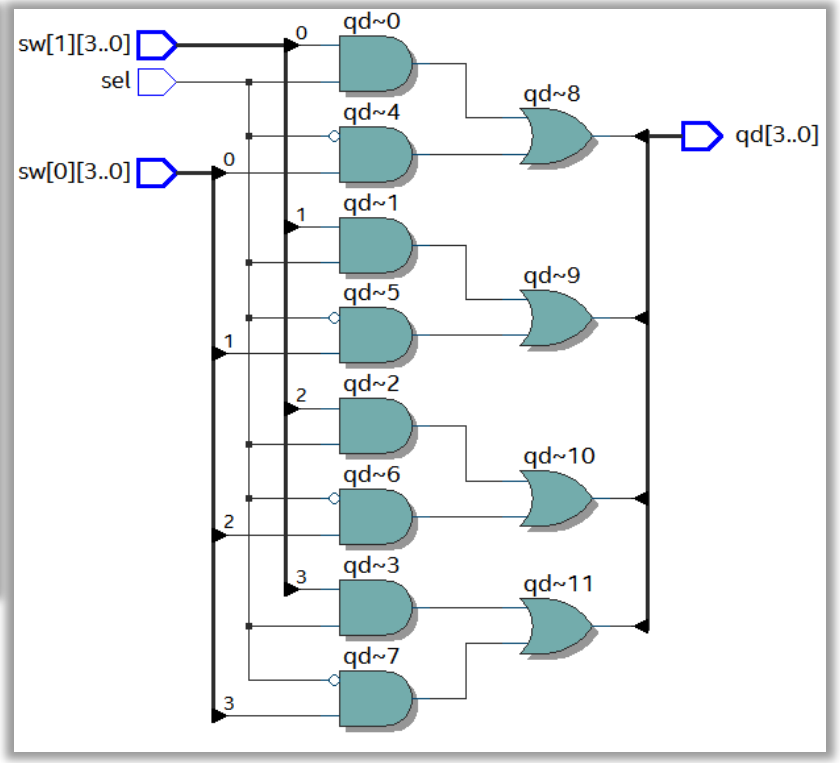
Пример многомерного массива векторов

```
1   module ex6(a, b, c, d);
2   input [1:0] a, b;
3   output [1:0] c;
4   output [1:0] d;
5
6   wire [1:0] mem [3:0] [5 :0];
7
8   assign mem[3] [5] = a;
9
10  assign mem[2] [4] = b;
11
12  assign d[1] = mem [3][5] [1];
13
14  assign d[0] = mem [2][4] [0];
15
16  assign c = mem[3] [5];
17
18  endmodule
```



Пример описания мультиплексора 2(4)=>1(4)

```
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
```



Приложение 2

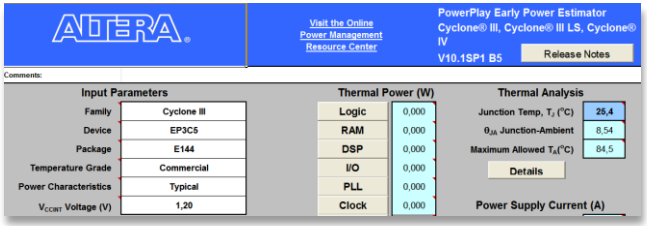
Встроенная система моделирования

Верификация проекта

- Значительная часть периода разработки тратится на верификацию проекта
- Верификация включает
  - ✓ Моделирование
    - – Поведенческое
    - – Временное
      - – После синтеза
      - – После трассировки СБИС
  - ✓ Временной анализ
  - ✓ Анализ энергопотребления
  - ✓ Анализ целостности сигналов
  - ✓ Тестирование в системе

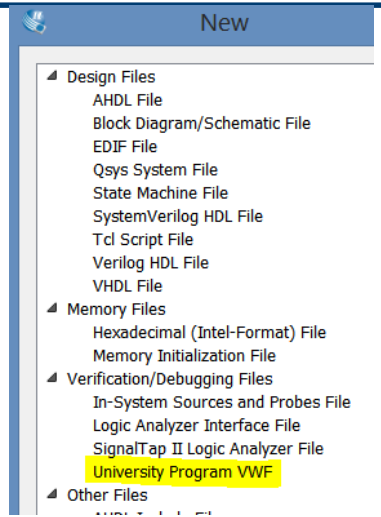
Приложения для верификации проекта в пакете Quartus

- Пакет Quartus® включает следующие приложения для верификации проектов:
  - ✓ Встроенная система моделирования
  - ✓ Modelsim - система моделирования Mentor Graphics
  - ✓ TimeQuest – временной анализатор
  - ✓ PowerPlay – анализатор энергопотребления
  - ✓ SignalTapII - Средство отладки в составе системы
- Дополнительные средства анализа энергопотребления
  - ✓ Таблицы xls: PowerPlay Early Power Estimator

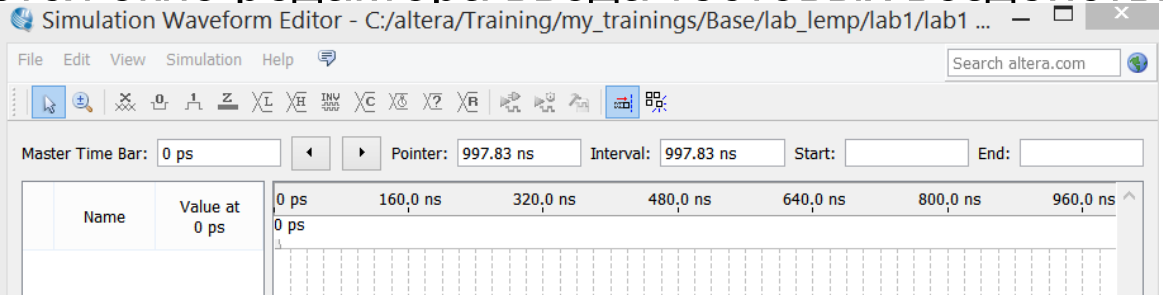


Создание файла с заготовкой тестовых воздействий

- Команда File>=New => University Program VWF

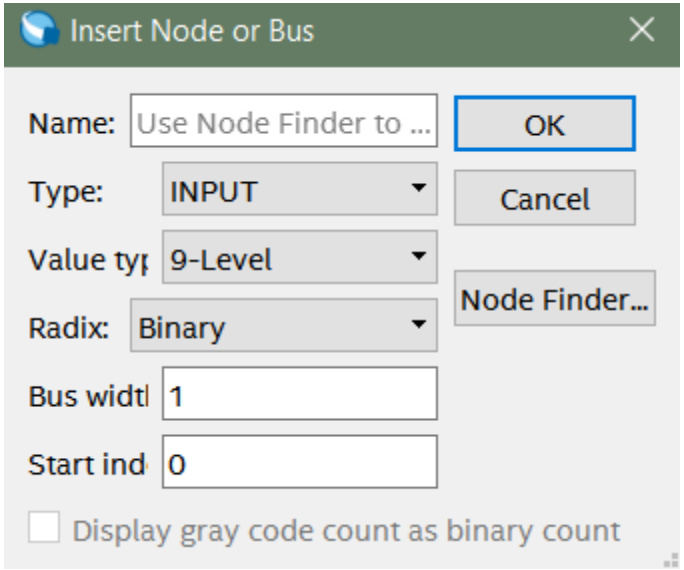


- Открывается окно редактора ввода тестовых воздействий



Задание входов для ввода тестовых воздействий

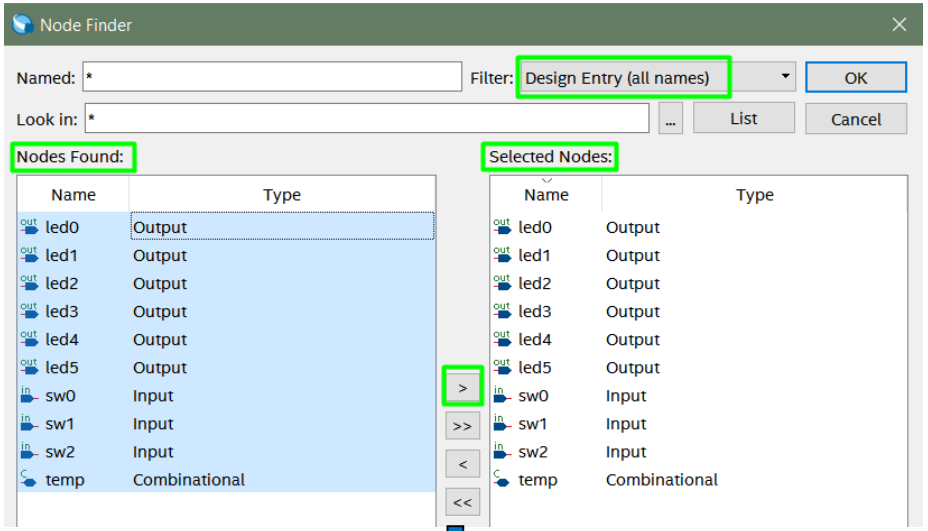
- Команда Edit=>Insert=> Insert Node or Bus

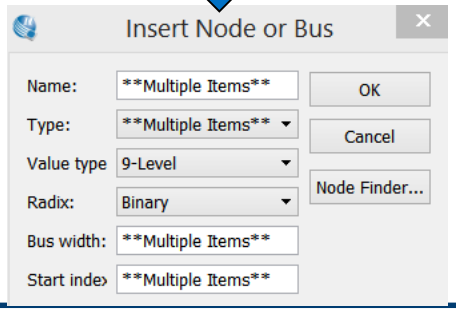


- Далее запустить Node Finder

Задание входов для ввода тестовых воздействий

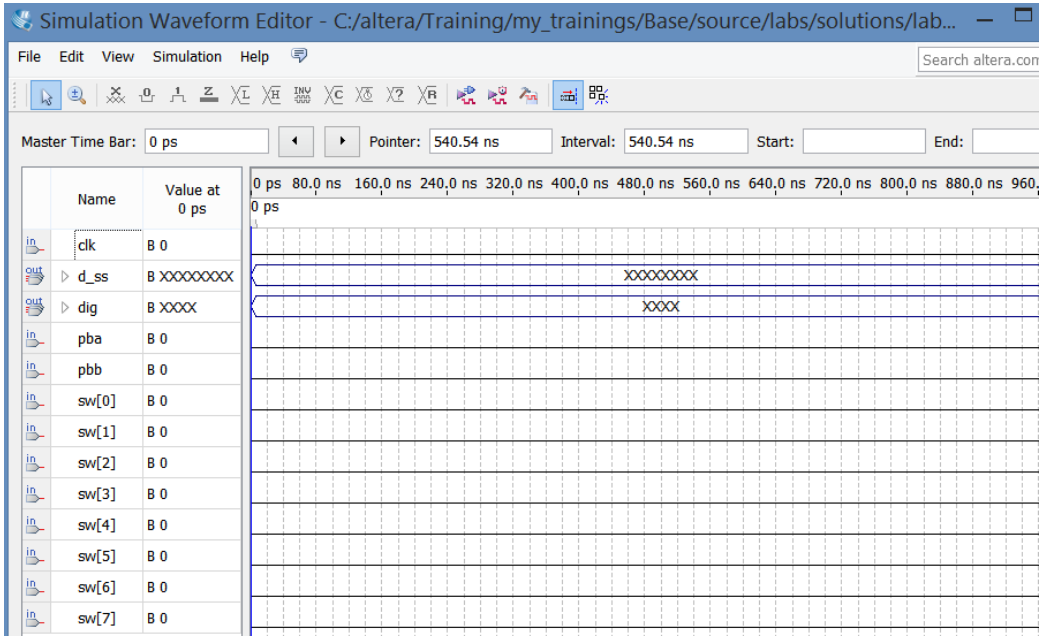
- В окне Node Finder нажать кнопку List
  - ✓ В разделе Nodes Found выделить интересующие сигналы и шины
  - ✓ Нажать символ > - перенести выбранные сигналы в окно Selected Nodes
  - ✓ Нажать кнопку OK, затем нажать OK еще раз.





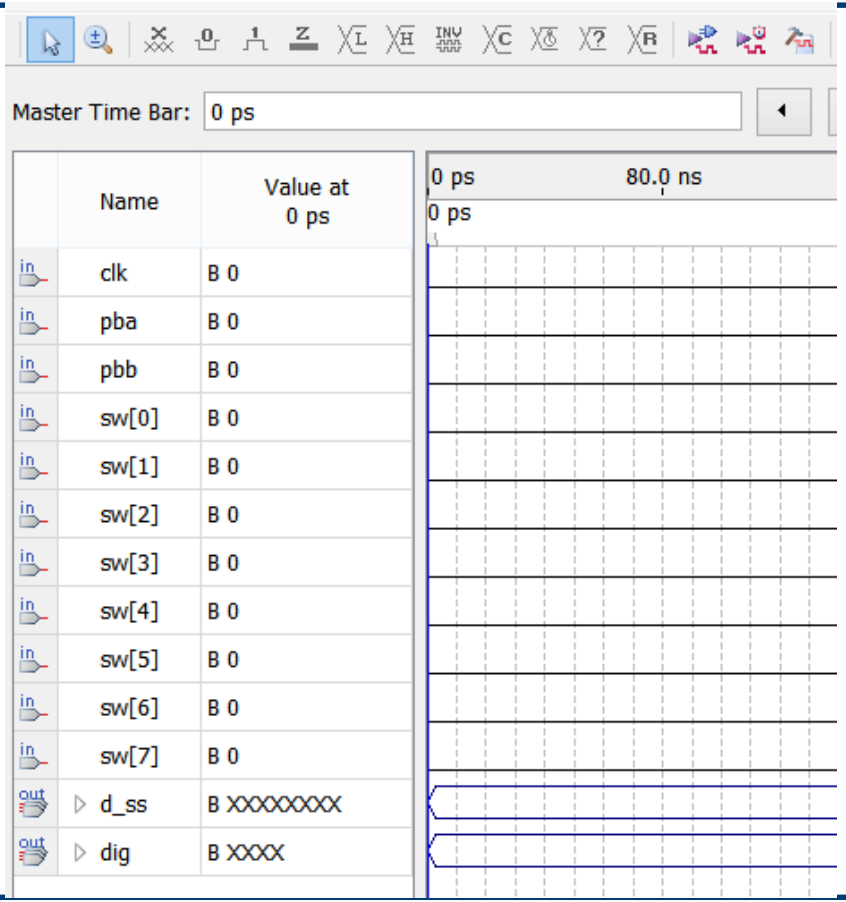
Окно редактора тестовых воздействий

- В окне редактора тестовых воздействий появятся выбранные входы (in) и выходы (out) (цепи и шины)



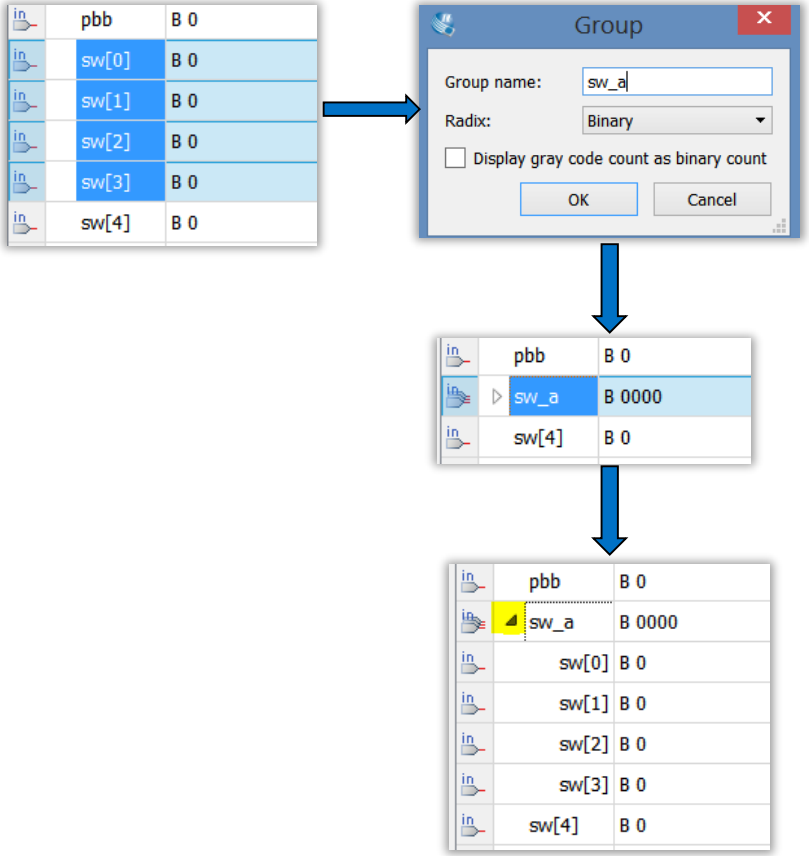
Перемещение выводов

- Выводы можно переместить в удобном для анализа порядке
  - ✓ выделить, нажав левую кнопку мыши, и не отпуская кнопку переместить вывод в нужное положение



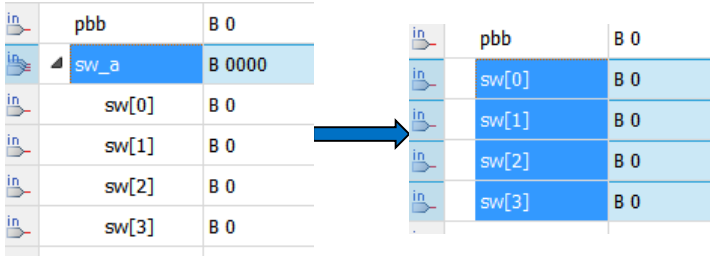
Группировка выводов

- Для того, чтобы сгруппировать выводы в шину следует:
  - ✓ выделить сигналы
  - ✓ выполнить команду Edit=>Grouping=>Group
  - ✓ Задать имя группы,
  - ✓ выбрать систему счисления для представления данных в группе
  - ✓ нажать ОК
- Чтобы посмотреть содержимое группы следует нажать на символ группы



Разбиение группы выводов

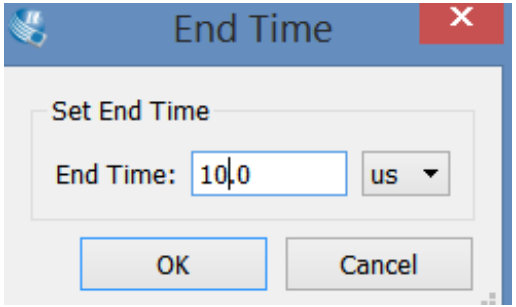
- Для того, чтобы разбить группу выводов следует:
  - ✓ выделить группу
  - ✓ выполнить команду Edit=>Grouping=>UnGroup



- По умолчанию, при создании группы старшинство разрядов устанавливается сверху вниз: вывод лежащий выше всех будет MSB (в примере, приведенном выше: sw[0] – MSB в группе).
- Для изменения весов разрядов используется команда: Edit=>Reverse Group or Bus bit Order

Задание длины теста

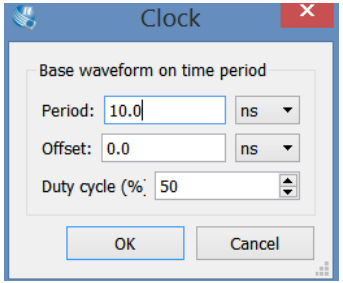
- Для задание длины теста следует выполнить команду:
  - ✓ Edit=>Set End Time
- В появившемся окне:
  - ✓ выбрать масштаб времени (мкс или нс)
  - ✓ установить длину теста

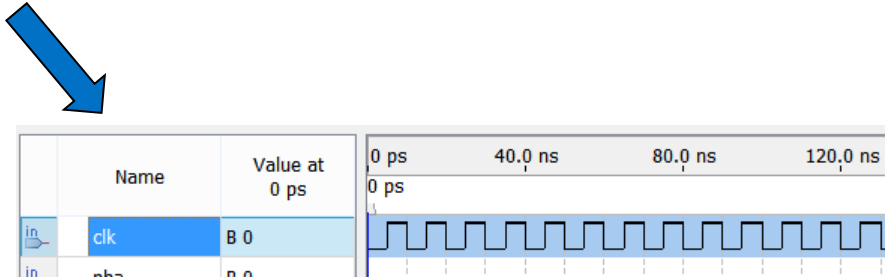


Чем длиннее тест, тем больше будет время моделирования

Задание тактового сигнала

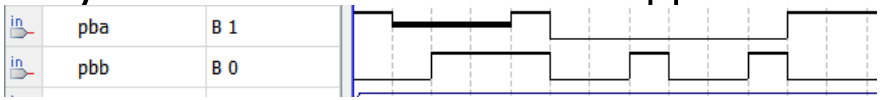
- Для задания тактового сигнала следует: выделить тактовый вход; выполнить команду Edit=>Value=>Overwrite Clock
- В появившемся окне задать период, сдвиг фазы фронта тактового сигнала, скважность





Задание значения одноразрядному входу

- Для задания значения одноразрядному входу на всю длину теста следует: выделить одноразрядный вход а затем выполнить команду Edit=>Value=>:
  - ✓ Forcing Low (0) – задание 0
  - ✓ Forcing High (1) – задание 1
  - ✓ Forcing High Impedance (Z) – задание Z состояния
- Для инвертирования значения сигнала следует выполнить команду Edit=>Value=>Invert
- Для изменения/задания значения сигнала на определенном промежутке времени следует: выделить промежуток времени и выполнить одну из указанных выше команд



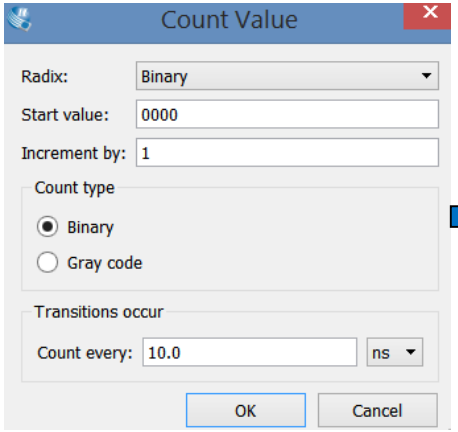
Задание значения группе выводов

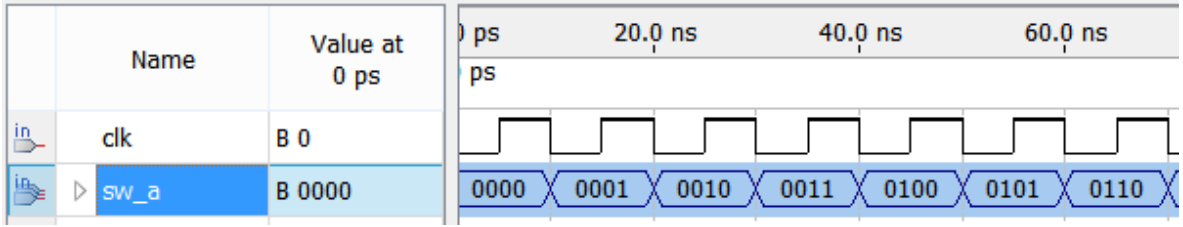
- Для задания значения группе выводов следует:
  - ✓ выделить группу выводов;
  - ✓ выполнить команду Edit=>Value=>:
    - – Count Value
    - – Arbitrary Value
    - – Random Value

- Если выделена не группа а набор выводов, то команда Count value недоступна.

Задание Edit=>Value=>Count Value

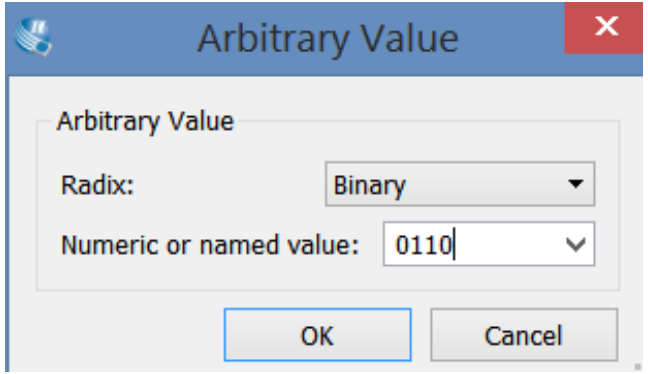
- В окне Count Value следует задать:
  - ✓ Систему счисления для отображения данных (Radix)
  - ✓ Начальное значение (Start Value)
  - ✓ Приращение (Increment)
  - ✓ Тип счетчика (binary – двоичный; Gray code – код Грея)
  - ✓ Период изменения данных и единицу измерения времени

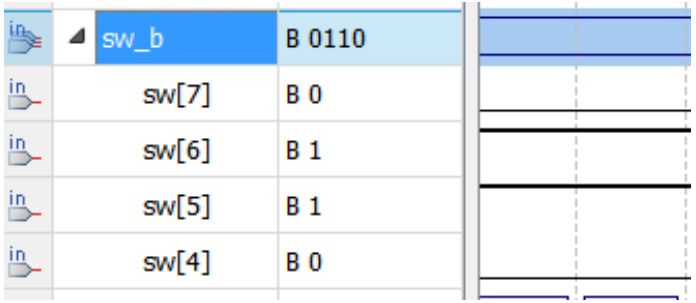




Задание Edit=>Value=>Arbitrary Value

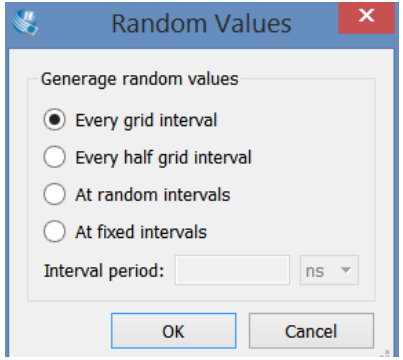
- В окне Arbitrary Value следует задать:
  - ✓ Систему счисления для отображения данных (Radix)
  - ✓ Произвольное число (разрядность определяется разрядностью шины)

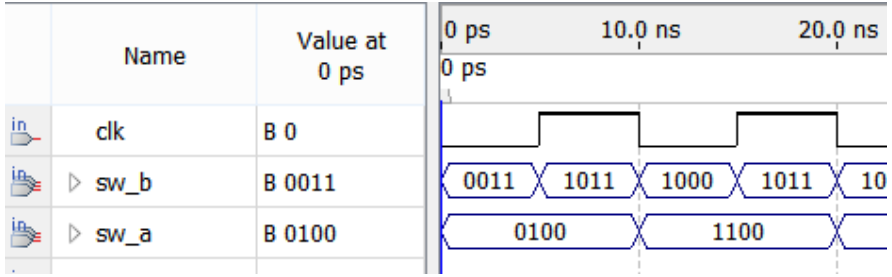




Задание Edit=>Value=>Random Value

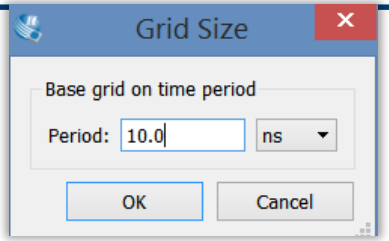
- В окне Random Value следует задать:
  - ✓ Систему счисления для отображения данных (Radix)
  - ✓ Частоту формирования случайного значения:
    - – На каждом временном интервале
    - – На половине каждого временного интервала
    - – Произвольный интервал формирования
    - – Фиксированный интервал (следует задать этот интервал)



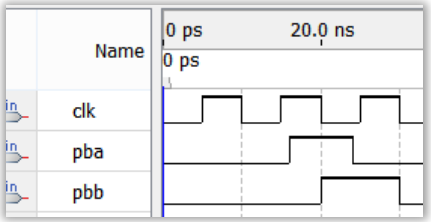


Команды настройки

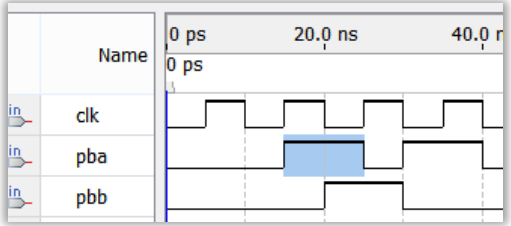
- Edit=>Grid Size – позволяет задать временной интервал



- Edit=>Snap To Grid – включает/выключает привязку редактирования к границам временных интервалов



- Edit=>Snap To transition – включает/выключает привязку редактирования к моментам изменения сигнала

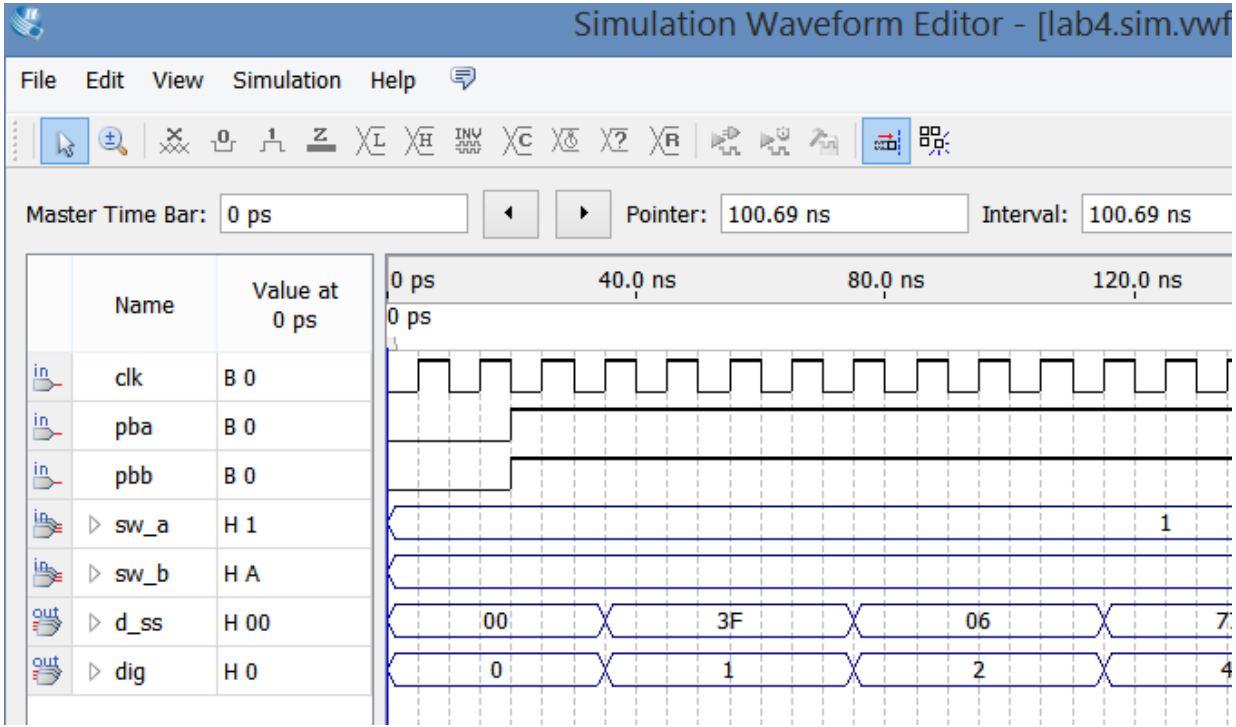


Процедура функционального моделирования

- Создать схему и выполнить команду **Processing=> Start => Start Analysis and Synthesis**
- Создать и настроить файл UWF:
  - ✓ подключить выводы и сигналы,
  - ✓ ввести временные диаграммы входных сигналов
- Выполнить команду Simulation=>Simulation Settings=>Restore defaults
  - ✓ Один раз перед первым запуском функционального моделирования
- Сохранить файл с тестами (файл UWF)
- Запустить функциональное моделирование
  - ✓ выполнить команду Simulation => Run Functional Simulation

Анализ результатов функционального моделирования

- По окончанию моделирования откроется отдельное окно с временными диаграммами результатов моделирования



Приложение 3

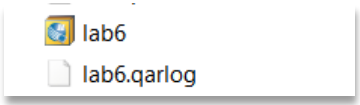
Архивирование проекта

Создание копии проекта

Очистка проекта

Архивирование проекта

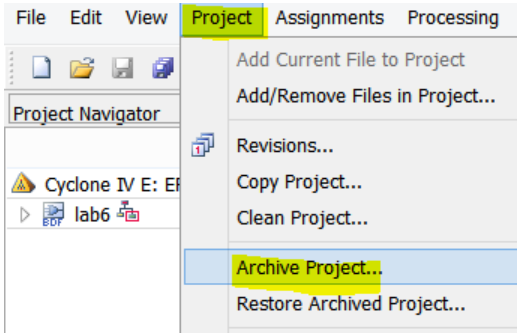
- Создается 2 файла
  - ✓ Файл архива (.QAR)
    - – Включает логические файлы, QPF, QSF файлы
    - – Позволяет добавить базу данных (папку db из папки проекта)
      - – Если база данных не включена, то после восстановления потребуется перекомпиляция проекта
    - – Включает созданный QDF для архива
  - ✓ Отчет о архивации (.QARLOG)

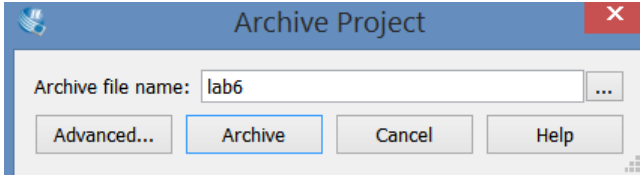


- Примеры использования
  - ✓ Хранение (например для контроля версий)
  - ✓ Передача проекта (Project handoff)
    - – Например в Altera support
- Файлы из пользовательских библиотек, использованные в проекте, копируются в архив

Запуск архивации проекта

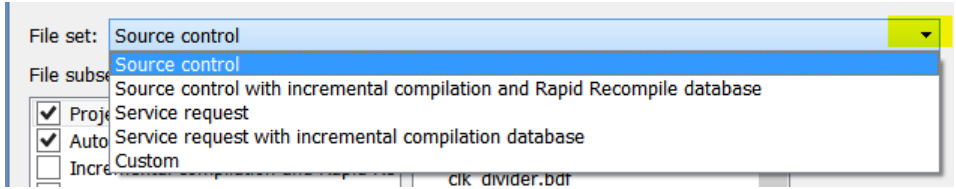
- Команда:
- Project=> Archive Project
- Открывает окно архивации проекта, в котором можно задать:
  - ✓ Задать имя архива
  - ✓ Задать папку для хранения архива
  - ✓ Запустить процедуру архивации
  - ✓ Открыть окно дополнительных настроек





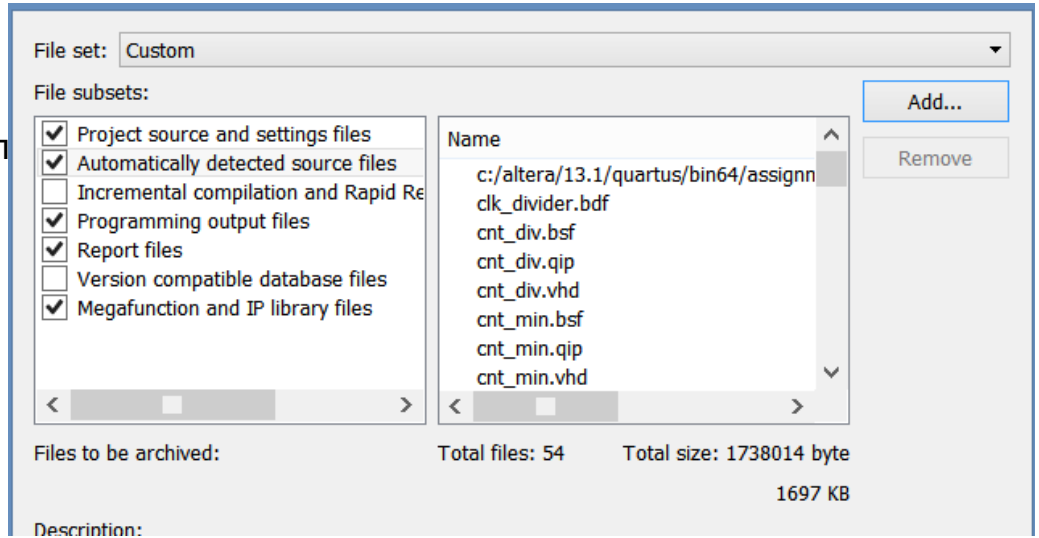
Окно дополнительных настроек

- Окно дополнительных настроек позволяет задать предустановленные опции указав назначение архива
  - ✓ Контроль версий (Source Control)
  - ✓ Контроль версий при разрешенной поэтапной компиляции
  - ✓ Запрос поддержки (service request)
  - ✓ Запрос поддержки при разрешенной поэтапной компиляции
  - ✓ Создать собственный набор настроек (Custom)



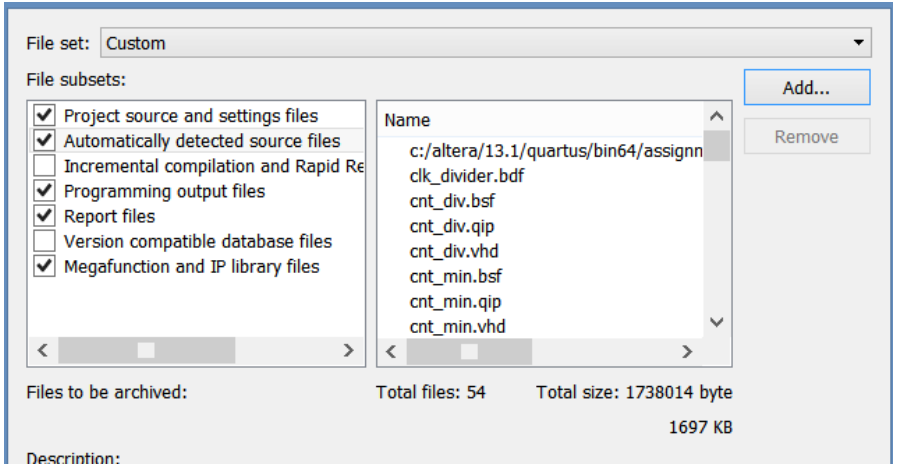
Окно дополнительных настроек (Набор настроек)

- Позволяет включить в архив:
  - ✓ Исходные файлы и файлы с установками
  - ✓ Автоматически найденные исходные файл
  - ✓ Результаты поэтапной компиляции
  - ✓ Файлы для программирования СБИС
  - ✓ Файлы с отчетами
  - ✓ Базу данных в формате, совместимом с другими версиями пакета
  - ✓ Библиотечные файлы мегафункций и IP модулей



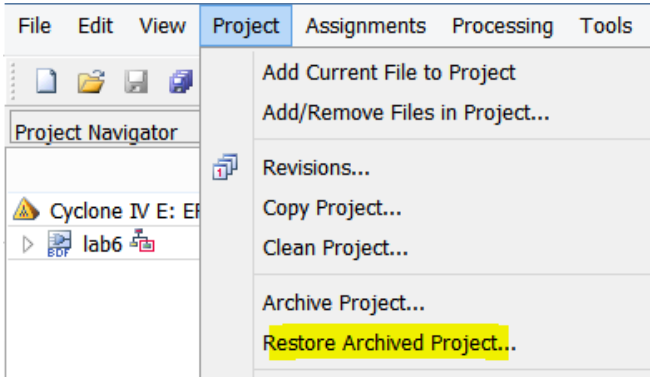
Окно дополнительных настроек (Набор настроек)

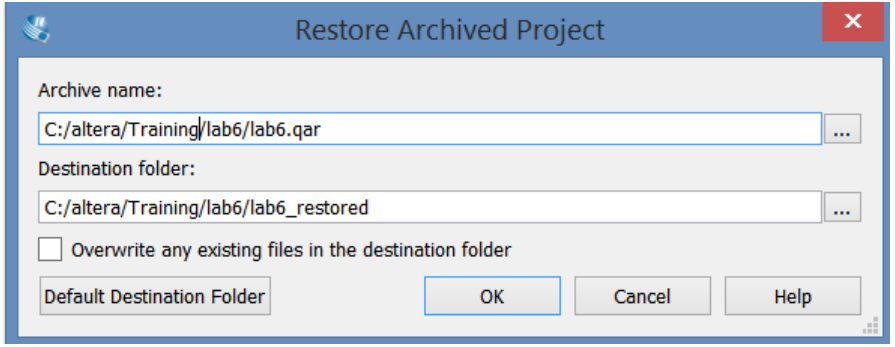
- Включаемые в архив файлы отображаются в колонке Name
- Строка Files to be archived показывает сколько файлов попадет в архив и их общий объем (до архивирования)
- Кнопка ADD позволяет добавить к архиву любые файлы (папки)



Восстановление проекта из архива

- Команда: Project=> Restore Archived Project открывает окно восстановления проекта из архива
- В окне можно выбрать архив и папку восстановления архива:
  - ✓ По умолчанию папка будет иметь имя xxx_restored



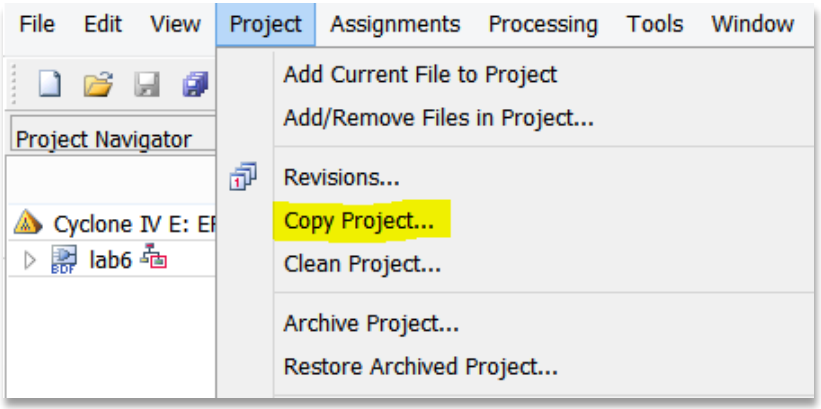


Копия проекта

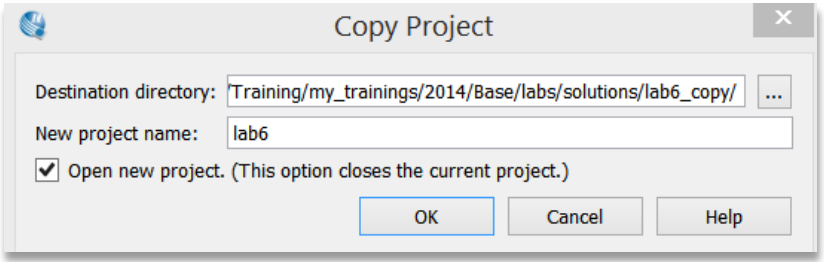
- При создании копии проекта создается дубликат проекта в новой папке
  - ✓ Project file (.QPF)
  - ✓ Design files
  - ✓ Settings files
- Пример использования
  - ✓ Создание дубликата проекта до редактирования логических файлов (схем и\или HDL файлов)
- Пользовательские библиотеки не копируются
- Новый QDF не создается; если QDF уже существует, то он копируется

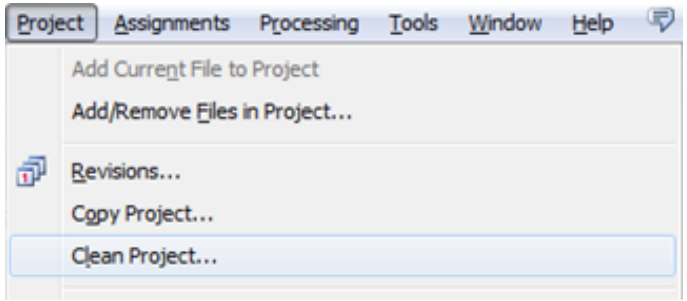
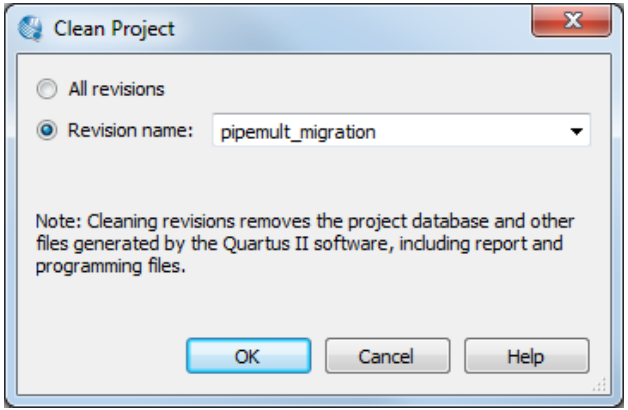
Создание копии проекта

- Команда: Project =>Copy Project открывает окно настроек копирования проекта



- В окне настроек задается:
  - ✓ Папка, в которой будет создана копия проекта
  - ✓ Имя копии проекта
  - ✓ Опция:
    - – Открыть новую копию проекта



Очистка проекта (Clean Project)

- Команда Project=>Clean Project:
  - ✓ Удаляет отчеты, файл для программирования, базу данных, и другие созданные компилятором файлы
  - ✓ Позволяет очистить все версии проекта или только выбранную версию (создание версий проекта будет рассмотрено позже)