

# Языки описания схем

(mk.cs.msu.ru → Лекционные курсы → Языки описания схем)

## Блок 11

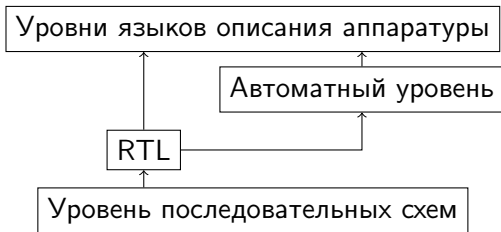
Verilog:  
от логических значений  
до комбинационных схем

Лектор:  
**Подымов Владислав Васильевич**

E-mail:

**valdus@yandex.ru**

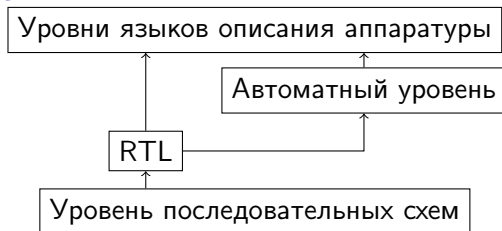
# Вступление



Разработка *последовательной схемы* и/или *RTL-описания схемы* — трудоёмкий и неустойчивый к ошибкам процесс:

- ▶ имеется *декларативное* описание поведения схемы
- ▶ из этого описания *методом пристального взгляда* извлекается основная масса триггеров/регистров
- ▶ схема *вручную* дополняется логическими вентилями/булевыми функциями, соединениями и вспомогательными триггерами/регистрами

# Вступление



В *языках описания аппаратуры* используются понятия и подходы, более близкие к декларативному описанию схем и позволяющие меньше задумываться о точной расстановке элементов схемы при её разработке

# Язык Verilog (V)

Verilog<sup>1</sup> — это один из двух самых популярных на данный момент языков описания цифровых микросхем<sup>2</sup>

Изначально этот язык создавался для **программной симуляции** схем:

- ▶ схема разрабатывается другими средствами
- ▶ на языке Verilog описывается **программная** модель, поведение которой приблизительно соответствует поведению схемы
- ▶ модель запускается (*как обычная программа*) и выдаёт информацию об изменении значений сигналов во времени и другие отладочные данные

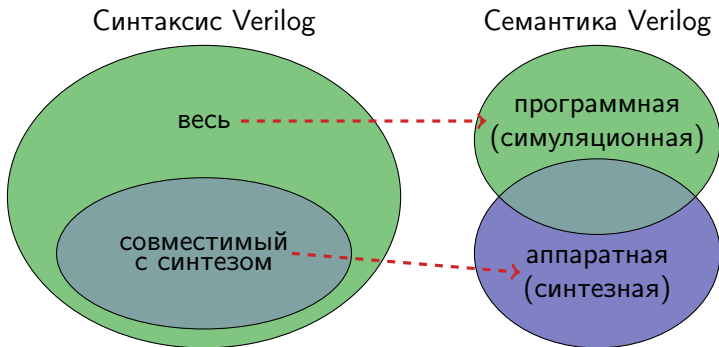
Язык оказался настолько удобным, что стал повсеместно применяться и для **синтеза** реальных схем

---

<sup>1</sup> Если считать его расширения и модификации, например, SystemVerilog

<sup>2</sup> Два самых популярных в мире языка описания аппаратуры — Verilog и VHDL. Эти языки похожи, при этом синтаксис Verilog проще, так что остановимся на нём

# Язык Verilog ( $\mathcal{V}$ )



Два главных документа, описывающих  $\mathcal{V}$  как

(1) средство симуляции и

(2) средство синтеза:

1. IEEE Standard for Verilog Hardware Description Language  
(в курсе обсуждается версия **2005**)
2. IEEE Standard for Verilog Register Transfer Level Synthesis  
(в курсе обсуждается версия **2002**)

# V и C/C++

Синтаксис Verilog местами очень похож на синтаксис C/C++<sup>1</sup>

Это сходство позволит избежать долгих объяснений “с нуля”, но следует иметь в виду, что оно поверхностно:

- ▶ Итог сборки кода
  - ▶ C/C++: машинный код, выполняемый процессором
  - ▶ Verilog: цифровая микросхема (*в том числе и сам процессор*)
- ▶ Трактровка переменных:
  - ▶ C/C++: последовательно изменяемые области памяти
  - ▶ Verilog: выделенные точки микросхемы
- ▶ Трактровка выражений и команд:
  - ▶ C/C++: команды машинного кода
  - ▶ Verilog: наборы логических вентилях и описание поведения подсхем
- ▶ ...

---

<sup>1</sup> И не просто так похож: создатели языка перенесли много синтаксических деталей из C, чтобы язык был *интуитивно понятнее*

## ∪: логические значения

В *C/C++* используются два логических значения:  
*истина* (**true**) и *ложь* (**false**)

В ∪ используются **четыре** логических значения:

- ▶ **1**: истина, единица, высокий уровень напряжения
- ▶ **0**: ложь, ноль, низкий уровень напряжения
- ▶ **X**: неопределённость
- ▶ **Z**: высокий импеданс

Константы, соответствующие этим логическим значениям, записываются так:<sup>1</sup>

**1'b1**

**1'b0**

**1'bx**

**1'bz**

---

<sup>1</sup> Это часть более широкого синтаксиса констант, но об этом позже

## $\mathcal{V}$ : логические значения $\mathcal{X}$ и $\mathcal{Z}$

Как понимать значение  $\mathcal{X}$ :

- ▶ Когда задаётся разработчиком:  
“мне неважно, что будет в этой точке схемы в этот момент”
- ▶ В программной семантике:  
полноценное значение в семантике выражений с приблизительной трактовкой “невозможно однозначно определить значение”
- ▶ В реальной схеме есть только конкретные напряжения, и средство синтеза доопределяет  $\mathcal{X}$  до 0 или 1 по своему усмотрению

Как понимать значение  $\mathcal{Z}$ :

- ▶ Коротко и огрублённо:  
“эта точка схемы изолирована от входных напряжений 0 и 1”
- ▶ Или так:  
“значение в точке не задаётся этой конструкцией языка”

В курсе значение  $\mathcal{Z}$  почти не обсуждается из-за сопутствующих технических нагромождений, не особо важных в начале изучения  $\mathcal{V}$

**Для простоты будем считать, что если значение  $\mathcal{Z}$  не упоминается, то его в языке не существует**



## ℳ: ОСНОВНЫЕ ТИПЫ ДАННЫХ

Типы данных ℳ делятся на две категории:

1. Типы соединений (*net data types*)
  - ▶ Пример такого типа: `wire` (провод)
2. Типы переменных (*variable data types*)
  - ▶ Пример такого типа: `reg`

Соединения и переменные будем называть **точками схемы**

(~ *переменные* в C/C++)

Объявления точек в ℳ устроены так же,  
как и объявления переменных в C/C++

- ▶ Объявление точек типа `wire` с именами `a`, `b`, `c`:

```
wire a, b, c;
```

- ▶ Объявление точек типа `reg` с именами `u`, `v`, `w`:

```
reg u, v, w;
```

## У: ОСНОВНЫЕ ТИПЫ ДАННЫХ

В простых случаях (wire, reg)

каждой точке сопоставляется некоторый *сигнал*,

устройство которого зависит от устройства сигналов в других точках

В аппаратной семантике сигнал определяется так,

как велит уровень абстракции описания схемы

В программной семантике сигнал

принимает *четыре логических значения* и имеет *мгновенные фронты*

Фронты сигнала, *положительные* ( $\uparrow$ ) и *отрицательные* ( $\downarrow$ ),

определяются так:

		после		
		0	$\mathcal{X}$	1
до	0		$\uparrow^1$	$\uparrow$
	$\mathcal{X}$	$\downarrow^1$		$\uparrow^1$
	1	$\downarrow$	$\downarrow^1$	

<sup>1</sup> При разработке схемы следует по возможности избегать фронтов, связанных со значением  $\mathcal{X}$

## У: основные типы данных

*Категория типа* каждой точки должна соответствовать тому, как точка **синтаксически** используется в коде схемы:

- ▶ некоторые конструкции гарантированно **не имеют памяти** (задают комбинационные схемы), и выходы таких конструкций обязаны быть соединениями
- ▶ некоторые конструкции в общем случае **имеют память** (способны задавать схемы с памятью), и выходы таких конструкций обязаны быть переменными

**Замечание:** название типа “reg” происходит от “register” (“регистр”), но это название не очень удачно: точками этого типа можно задавать сигналы **не только** на выходах регистров (и, в частности, триггеров)

**Очевидное синтаксическое ограничение**, подразумевающееся во всех дальнейших описаниях: каждое логическое значение в каждой точке схемы должно задаваться **не более чем одной** языковой конструкцией<sup>1</sup>

---

<sup>1</sup> Это неверно в общем случае, но часто верно в синтезируемом коде, так что будем считать это правилом и отдельно обозначать все исключения

## ℳ: ШИНЫ

Объявление *шин* (в терминологии стандарта — **векторов**) в ℳ:

```
type [msb:lsb] id1, id2, ...;
```

- ▶ type — *невекторный* тип (wire, reg, ...)
- ▶ id1, id2, ... — имена точек
- ▶ msb и lsb — номерá старшего и младшего разрядов шины
- ▶ “type [msb:lsb]” — тип той же категории, что и type

### Примеры:

- ▶ Шина проводов x ширины 5:

```
wire [4:0] x;
```

- ▶ Шина reg-ов y ширины 3:

```
reg [2:0] y;
```

- ▶ То же, что и y, но разряды нумеруются с двойки:

```
reg [4:2] z;
```

## $\mathcal{V}$ : шины

В  $\mathcal{V}$  используются две арифметических трактовки значения в шине:

- ▶ **Беззнаковая**:

$$(\alpha_{n-1} \dots \alpha_0)_2 = \sum_{i=0}^{n-1} 2^i \cdot \alpha_i$$

- ▶ **Знаковая** (дополнительный код):

$$(\alpha_{n-1} \dots \alpha_0)_2^- = (\overline{\alpha_{n-1}} \alpha_{n-2} \dots \alpha_0)_2 - 2^{n-1}$$

По умолчанию шины полагаются беззнаковыми

Арифметическая трактовка корректна, если в шине содержатся **только** значения 0, 1 — иначе соответствующее число **не определено**

*Одноразрядная* точка обычно отождествляется с шиной точек ширины 1

## У: порты

**Портами** в  $\mathcal{U}$  и в целом в области схемотехники называются точки схемы, через которые она взаимодействует с окружением (ранее в лекциях это называлось *контактами* и *шинами контактов*)

Сейчас достаточно рассмотреть два вида портов:

- ▶ **Входные порты**, или просто **входы** —  
через эти порты сигналы посылаются в схему извне
  - ▶ обозначаются ключевым словом `input`
- ▶ **Выходные порты**, или просто **выходы** —  
через эти порты сигналы отправляются из схемы вовне
  - ▶ обозначаются ключевым словом `output`

## $\mathcal{V}$ : модули

Модуль в  $\mathcal{V}$  — это

- ▶ описание (под)схемы
- ▶ понятие, аналогичное *классу/функции/функтору* языка *C/C++*

В модуле содержатся, в числе прочего:

- ▶ **имя** (*~ имя класса/функции*)
- ▶ **объявление портов** (*~ объявление аргументов функции*)
- ▶ **тело** (*~ тело функции*)

В теле модуля описывается схема с портами (входами, выходами, ...), задаваемыми модулем

**Экземпляры** модуля (*~ объекты класса*)

можно вставлять в другие модули в качестве подсхем, задавая соединения портов подсхемы с точками схемы

## V: модули

Первый способ объявления модуля:

```
module <имя модуля>(<объявления портов через запятую>);  
    <тело модуля>  
endmodule
```

**Пример:** модуль M с входными проводами a, b, выходной шиной проводов u ширины 2 и выходным reg-ом v

```
module M(input wire a, b,  
          output wire [1:0] u, output reg v);  
    // тело модуля  
endmodule
```

(комментарии в V устроены так же, как и в C/C++)



## У: модули

### Второй способ объявления модуля:

```
module <имя модуля>(<имена портов через запятую>);  
    <объявления портов>  
    <тело модуля>  
endmodule
```

**Пример:** модуль M с входными проводами a, b, выходной шиной проводов u ширины 2 и выходным reg-ом v

```
module M(a, b, u, v);  
    input wire a, b;  
    output wire [1:0] u;  
    output reg v;  
    // тело модуля  
endmodule
```

## У: модули

Слово `wire` в объявлении портов можно опускать:

```
module M(input /* wire */ a, b,  
         output /* wire */ [1:0] u, output reg v);  
    // тело модуля  
endmodule
```

```
module M(a, b, u, v);  
    input /* wire */ a, b;  
    output /* wire */ [1:0] u;  
    output reg v;  
    // тело модуля  
endmodule
```

## У: модули

Объявление типа и объявление порта **независимы**:

```
module M(input a, b,  
         output u, v);  
    // тело модуля  
    wire [1:0] u;  
    reg v;  
    // тело модуля  
endmodule
```

```
module M(a, b, u, v);  
    input a, b;  
    output u, v;  
    // тело модуля  
    wire [1:0] u;  
    reg v;  
    // тело модуля  
endmodule
```

**Синтаксическое ограничение:** все входы должны быть соединениями

**Можно:**

```
input wire [1:0] a;
```

**Нельзя:**

```
input reg [1:0] a;
```

## У: непрерывное присваивание (assign)

```
assign x = E;
```

- ▶  $x$  — соединение, не являющееся входом
- ▶  $E$  — **комбинационное выражение**: выражение, составленное из
  - ▶ точек,
  - ▶ констант и
  - ▶ специальных (**комбинационных**) операций

Содержательный смысл такой конструкции:

**в каждый момент времени**

**значение в точке  $x$  совпадает со значением выражения  $E$**

Такой смысл имеют все комбинационные схемы

### Пара простых примеров:

- ▶ сигнал из  $y$  без изменений направляется в соединение  $x$ :

```
assign x = y;
```

- ▶ значение провода  $x$  — сигнал с константным значением 1:

```
assign x = 1'b1;
```

## ∪: константы

Общий способ записи константных значений:

`<ширина>'<система счисления><значение>`

Системы счисления:

**b**: двоичная

**d**: десятичная

**o**: восьмеричная

**h**: шестнадцатеричная

**Пример**: шина ширины 5 со значением 29

`5'b11101`

`5'd29`

`5'o35`

`5'h1d`

**Упрощённая запись констант**:

`'<с.с.><значение> = N'<с.с.><значение>`,

где N — неспецифицированная ширина не менее 32

`<значение> = 'd<значение>`

## У: СПИСОК ОСНОВНЫХ КОМБИНАЦИОННЫХ ОПЕРАЦИЙ

	<code>x &amp;&amp; y</code>		<code>x    y</code>		<code>!x</code>	
<code>x + y</code>	<code>x - y</code>	<code>+x</code>	<code>-x</code>	<code>x * y</code>	<code>x / y</code>	<code>x % y</code>
<code>x == y</code>	<code>x != y</code>	<code>x &gt; y</code>	<code>x &gt;= y</code>	<code>x &lt; y</code>	<code>x &lt;= y</code>	
<code>x &lt;&lt; y</code>	<code>x &gt;&gt; y</code>	<code>x &lt;&lt;&lt; y</code>	<code>x &gt;&gt;&gt; y</code>			
<code>x &amp; y</code>	<code>x   y</code>	<code>x ^ y</code>	<code>~x</code>			
<code>&amp;x</code>	<code> x</code>	<code>~x</code>	<code>~&amp;x</code>	<code>~ x</code>	<code>~~x</code>	
		<code>x ? y : z</code>				
	<code>x[i]</code>		<code>x[i:j]</code>			
	<code>\$signed(x)</code>		<code>\$unsigned(x)</code>			

---

Очень похоже на операции языка C/C++ —  
это действительно так, но только содержательно и только отчасти

## $\mathcal{V}$ : аппаратная семантика assign

Аппаратная семантика присваивания “assign  $x = E$ ;” — произвольная *комбинационная схема*  $\Sigma$  следующего вида:

- ▶ Входы  $\Sigma$  — все точки, используемые в  $E$
- ▶ Выход  $\Sigma$  — шина  $x$
- ▶ Программной семантикой для каждого набора **булевых** значений  $(\alpha_1, \dots, \alpha_n)$  на входах задаётся результат выражения  $E$  — набор  $(\beta_1, \dots, \beta_k)$  **булевых** значений на выходе
- ▶ При *посылке* значений  $(\alpha_1, \dots, \alpha_n)$  на входы  $\Sigma$  на выходах *получаются* значения  $(\beta_1, \dots, \beta_k)$

## У: расширение и сужение шин

В семантику некоторых комбинационных операций включены механизмы **выравнивания** ширины аргументов: **расширения** слишком узких и **сужения** слишком широких

**Сужение** всегда устроено одинаково: отбрасываются старшие разряды

$$(x_{n-1}x_{n-2} \dots x_k x_{k-1} x_{k-2} \dots x_0) \rightsquigarrow (x_{k-1}x_{k-2} \dots x_0)$$

**Расширение**, как правило, относится к одному из двух видов:

1. **Беззнаковое**: в старшие разряды дописывается 0

$$(x_{k-1}x_{k-2} \dots x_0) \rightsquigarrow (00 \dots 0x_{k-1}x_{k-2} \dots x_0)$$

2. **Знаковое**: старший разряд “размножается” нужное число раз

$$(x_{k-1}x_{k-2} \dots x_0) \rightsquigarrow (x_{k-1}x_{k-1} \dots x_{k-1}x_{k-1}x_{k-2} \dots x_0),$$

то есть

$$(0x_{k-2} \dots x_0) \rightsquigarrow (00 \dots 00x_{k-2} \dots x_0)$$

$$(1x_{k-2} \dots x_0) \rightsquigarrow (11 \dots 11x_{k-2} \dots x_0)$$

$$(Xx_{k-2} \dots x_0) \rightsquigarrow (XX \dots XXx_{k-2} \dots x_0)$$

$$(Zx_{k-2} \dots x_0) \rightsquigarrow (ZZ \dots ZZx_{k-2} \dots x_0)$$



## $\mathcal{V}$ : знаковость результата выражения

Каждое значение в комбинационном выражении трактуется либо как *знаковое*, либо как *беззнаковое*

Этим определяются *арифметическая трактовка* значения и способ *расширения* шины по умолчанию

Язык  $\mathcal{V}$  “тяготеет” к беззнаковой трактовке: согласно **стандартной** схеме учёта знаковости аргументов,

- ▶ результат выполнения операции объявляется беззнаковым  $\Leftrightarrow$  хотя бы один из аргументов беззнаковый
- ▶ перед вычислением результата знаковость аргументов приравнивается знаковости результата
  - ▶ Расширение и сужение шин выполняется после смены знаковости
  - ▶ При изменении знаковости аргументов их значения не меняются — меняется **только** арифметическая трактовка этих значений

# $\mathcal{V}$ : комбинационные операции

## Логические операции

$x \ \&\& \ y$

$x \ || \ y$

$!x$

**Программная семантика:** результат — беззнаковая шина ширины 1, значение которой определяется согласно таблицам

		x && y		
		y	0	$\mathcal{X}$
x		0	$\mathcal{X}$	1
0		0	0	0
$\mathcal{X}$		0	$\mathcal{X}$	$\mathcal{X}$
1		0	$\mathcal{X}$	1

		x    y		
		y	0	$\mathcal{X}$
x		0	$\mathcal{X}$	1
0		0	$\mathcal{X}$	1
$\mathcal{X}$		$\mathcal{X}$	$\mathcal{X}$	1
1		1	1	1

		!x
x		!x
0		1
$\mathcal{X}$		$\mathcal{X}$
1		0

При вычислении результата аргументы сужаются до ширины 1

# У: комбинационные операции

## Арифметические операции

$x + y$      $x - y$      $+x$      $-x$      $x * y$      $x / y$      $x \% y$

### Программная семантика:

- ▶ Результат — естественный для арифметики *с переполнением*
  - ▶ Если хотя бы один из аргументов *не определён*, то результат —  $(\mathcal{X}\mathcal{X}\dots\mathcal{X})$
- ▶ Знаковость аргументов учитывается *стандартно*
- ▶ **Ширина выражения** — это максимум ширины аргументов и
  - ▶ если операция является внешней, то ширины левой части присваивания
  - ▶ иначе — ширины “объемлющего” выражения, если эта ширина задана
- ▶ **Ширина результата** равна ширине выражения
  - ▶ При вычислении результата аргументы расширяются до ширины выражения

# ∨: комбинационные операции

## Арифметические отношения:

$x == y$     $x != y$     $x > y$     $x >= y$     $x < y$     $x <= y$

## Программная семантика:

- ▶ Результат — беззнаковая шина ширины 1 со следующим значением:
  - ▶ если оба числа-аргумента *определены*, то
    - ▶ 1, если числа входят в отношение, и
    - ▶ 0, если не входят
  - ▶ иначе результат —  $\mathcal{X}$
- ▶ Знаковость аргументов учитывается *стандартно*
  - ▶ Но результат всё равно беззнаковый
- ▶ При вычислении результата узкий аргумент расширяется до широкого

# ∪: комбинационные операции

## Сдвиговые операции:

$x \ll y$

$x \gg y$

$x \lll y$

$x \ggg y$

## Программная семантика:

Результат имеет ту же ширину и знаковость, что и  $x$

“ $\ll$ ” и “ $\lll$ ”:

сдвиг  $x$  влево на  $y$  разрядов с заполнением нолями

“ $\gg$ ”:

*(логический сдвиг)*

результат — сдвиг  $x$  вправо

на  $y$  разрядов с заполнением нолями

“ $\ggg$ ”:

*(арифметический сдвиг)*

результат — сдвиг  $x$  вправо на  $y$  разрядов

с заполнением при помощи расширения шины

Если число  $y$  не определено, то результат —  $(xx \dots x)$

# У: комбинационные операции

## Многобитовые логические операции:

$x \& y$

$x | y$

$x \wedge y$

$\sim x$

## Программная семантика:

- ▶ **Результат:**  
каждый разряд получается из соответствующих разрядов  $x$  и  $y$  применением соответствующей операции: ( $\&$ ,  $|$ ,  $\wedge$ ,  $!$ )
- ▶ **Ширина результата** равна ширине самого широкого аргумента
  - ▶ Узкий аргумент расширяется до широкого беззнаково
- ▶ Знаковость аргументов учитывается стандартно

# У: комбинационные операции

## Операции редукции:

$\&x$

$|x$

$\sim x$

$\sim\&x$

$\sim|x$

$\sim\sim x$

## Программная семантика:

Результат — беззнаковая шина ширины 1 со следующим значением:

- ▶ Для операции без “ $\sim$ ”:  
соответствующая логическая операция ( $\&\&$ ,  $||$ ,  $!=$ )  
применяется к паре младших разрядов, и затем,  
итеративно до конца шины,  
к результату предыдущего шага и следующему разряду
- ▶ Для операции с “ $\sim$ ”:  
отрицание (!) результата  
соответствующей операции без “ $\sim$ ”

# ∪: комбинационные операции

## Тернарный оператор:

$x ? y : z$

## Программная семантика:

- ▶ **Ширина результата** — это максимум ширин  $y$  и  $z$ 
  - ▶ Узкий аргумент  $y/z$  расширяется до широкого беззнаково
- ▶ **Результат** зависит от значения **условия** — выражения “ $x == 0$ ”:
  - ▶  $1 \Rightarrow$  результат совпадает с  $z$
  - ▶  $0 \Rightarrow$  результат совпадает с  $y$
  - ▶  $\mathcal{X} \Rightarrow i$ -й разряд результата — это
    - ▶  $i$ -й разряд  $y$ , если  $i$ -е разряды  $y$  и  $z$  равны
    - ▶  $\mathcal{X}$  иначе
- ▶ Знаковость аргументов  $y$  и  $z$  учитывается стандартно



# ∪: комбинационные операции

## Операции индексации

$x[i]$

$x[i:j]$

**Программная семантика:** если  $x$  — шина, то

- ▶  $x[i]$  — соединение шины  $x$  с индексом  $i$
- ▶  $x[i:j]$  — шина ( $x[i] \dots x[j]$ )

## Операции конкатенции (слева) и репликации (справа)

$\{x_1, x_2, \dots, x_n\}$

$\{N\{x_1, x_2, \dots, x_n\}\}$

**Программная семантика:** если  $x_i$  — шина ширины  $k_i$ , то

- ▶  $\{x_1, x_2, \dots, x_n\}$  — беззнаковая шина  
 $(x_1[k_1-1] \dots x_1[0] \ x_2[k_2-1] \dots x_2[0] \ \dots \ x_n[k_n-1] \dots x_n[0])$
- ▶  $\{N\{x_1, x_2, \dots, x_n\}\}$  равносильно конкатенации  $N$  копий шины  $\{x_1, x_2, \dots, x_n\}$

**Замечание:** индексация и конкатенация

могут использоваться в левых частях присваиваний

# У: особенности работы со знаковостью

## Операции изменения знака

`$signed(x)`

`$unsigned(x)`

## Программная семантика:

- ▶ ширина и значение результата совпадают с шириной и значением аргумента
- ▶ результат трактуется как знаковый (`$signed`) или беззнаковый (`$unsigned`)

## У: особенности работы со знаковостью

Для всех точек по умолчанию применяется беззнаковая трактовка

Чтобы это изменить, следует добавить слово `signed` в объявление:

```
input signed x;  
wire signed y;  
reg signed [1:0] z;
```

Константы вида “`<значение>`” по умолчанию трактуются как знаковые

Остальные константы

(`'<с.с.><значение>`, `<ширина>'<с.с.><значение>`)

по умолчанию трактуются как беззнаковые

**Знаковые константы** с указанием системы счисления выглядят так:

`'s<с.с.><значение>` `<ширина>'s<с.с.><значение>`

## ∪: примеры (модуль + assign)

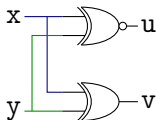
```
module M(input x, y, output u, v);  
    assign u = x && y || !x && !y;  
    assign v = !u;  
endmodule
```

Программная семантика модуля:

		u		
		0	X	1
x \ y	0	1	X	0
	X	X	X	X
	1	0	X	1

		v		
		0	X	1
x \ y	0	0	X	1
	X	X	X	X
	1	1	X	0

Одна из аппаратных семантик модуля:



## ∪: примеры (модуль + assign)

```
module M(input x, y, output u, v);  
    assign u = x && y || !x && !y;  
    assign v = !u;  
endmodule
```

Объявление соединения и assign можно “совместить”:

```
wire x;  
assign x = E;    = wire x = E;
```

```
module M(input x, y, output u, v);  
    wire tmp1 = x && y;  
    wire tmp2 = !x && !y;  
    assign u = tmp1 || tmp2;  
    assign v = !u;  
endmodule
```

## $\mathcal{V}$ : примеры (модуль + assign)

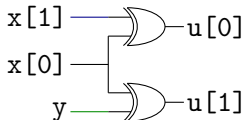
```
module M(input [1:0] x, input y, output [1:0] u);  
  assign {u[0], u[1]} = x ^ {x[0], y};  
endmodule
```

Программная семантика модуля:

		u[0]		
		x[1]	0	$\mathcal{X}$
x[0]	0	0	$\mathcal{X}$	1
	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$
	1	1	$\mathcal{X}$	0

		u[1]		
		y	0	$\mathcal{X}$
x[0]	0	0	$\mathcal{X}$	1
	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{X}$
	1	1	$\mathcal{X}$	0

Одна из аппаратных семантик модуля:



## У: использование подсхем

В теле модуля можно использовать *экземпляры* других модулей в качестве подсхем

Синтаксис вставки экземпляра:

```
<имя модуля> <имя экземпляра>  
  (<назначения портов через запятую>);
```

Рекомендуемый синтаксис назначения порта:

```
.<имя порта>(<выражение>)
```

Смысл такого назначения порта:

- ▶ входной порт: выполняется непрерывное присваивание произвольного <выражения> в порт подсхемы
- ▶ выходной порт:
  - ▶ выполняется непрерывное присваивание порта подсхемы в <выражение>
  - ▶ допускаются только <выражения>, которые разрешено располагать в левой части непрерывного присваивания (“правильно собранные” из переменных, индексаций и конкатенаций)

## У: использование подсхем

Порядок назначений портов экземпляра и порядок объявлений портов соответствующего модуля **не обязаны совпадать** при использовании рекомендуемого синтаксиса

Если входной порт экземпляра не назначен, то на этот вход посылается значение  $Z^1$

Если выходной порт экземпляра не назначен, то *ничего страшного не происходит*

---

<sup>1</sup> Если не уверены, действительно ли так нужно, то старайтесь этого избегать



## ∪: использование подсхем

**Пример напоследок:** реализация сумматора трёхразрядных чисел<sup>1</sup>

```
module adder_cell(input x, y, cin, output sum, cout);
    assign {cout, sum} = x + y + cin;
endmodule

module adder(input [2:0] x, y, output [3:0] z);
    adder_cell cell1(.x(x[0]), .y(y[0]), .cin(1'b0),
                    .sum(z[0]), .cout(c1)); // 2
    adder_cell cell2(.x(x[1]), .y(y[1]), .cin(c1),
                    .sum(z[1]), .cout(c2));
    adder_cell cell3(.x(x[2]), .y(y[2]), .cin(c2),
                    .sum(z[2]), .cout(z[3]));
endmodule
```

---

<sup>1</sup> Не реализуйте сумматор так!

Это просто демонстрация возможностей языка

<sup>2</sup> **ВНИМАНИЕ!** Каждая необъявленная точка *по определению* имеет тип `wire` — в том числе точки `c1` и `c2`