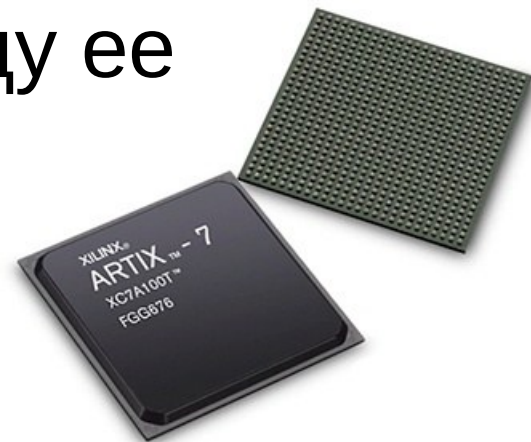


# Введение в ПЛИС

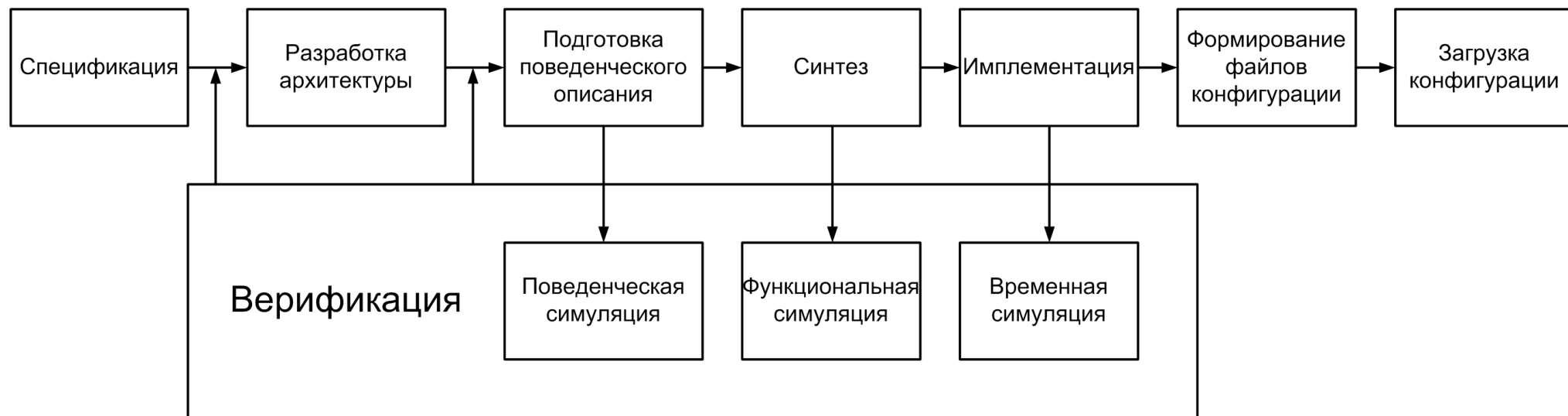
ПЛИС (программируемая логическая интегральная схема) – интегральная схема, у которой выполняемая логическая функция программируется уже после изготовления за счет задания соединений между ее структурными элементами



# Область применения ПЛИС

- Прототипирование ИС
- Цифровая обработка сигналов
- Реализация встраиваемых микроконтроллеров
- Системы с перестраиваемой архитектурой

# Этапы проектирования устройства на базе ПЛИС



# САПР для ПЛИС-разработки

САПР от вендора ПЛИС:

- Vivado (Xilinx, AMD)
- Quartus Prime (Altera, Intel)
- Lattice Diamond (Lattice Semiconductor)

Сторонние симуляторы:

- ModelSim
- QuestaSim
- IcarusVerilog + GTKWave



# Способы описания логики работы схемы

- Схемное описание
- Hardware description language (HDL):
  - SystemVerilog / Verilog
  - VHDL
- Смешанное описание

# SystemVerilog

- Семантика языка описывает изменение сигналов цифровой схемы с течением времени
- Содержит синтезируемое подмножество, т.е. некоторые (не все) конструкции языка могут быть преобразованы синтезаторами “реальную” схему

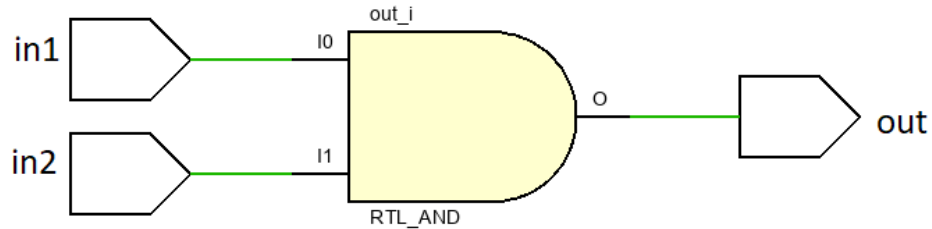
# Модуль

- Основной строительный блок SystemVerilog
- Описание содержится между ключевыми словами `module` и `endmodule`

```
module basic_and(  
    input  in1,  
    input  in2,  
    output out  
);
```

```
    assign out = in1 & in2; // непрерывное присваивание
```

```
endmodule
```



# Непрерывное присваивание

- Ключевое слово “assign”
- При непрерывном присваивании любое изменение в правой части выражения вызывает изменение в левой части
- Непрерывное присваивание является параллельным оператором



- Внутри описания модуля можно инициализировать и назначать промежуточные сигналы

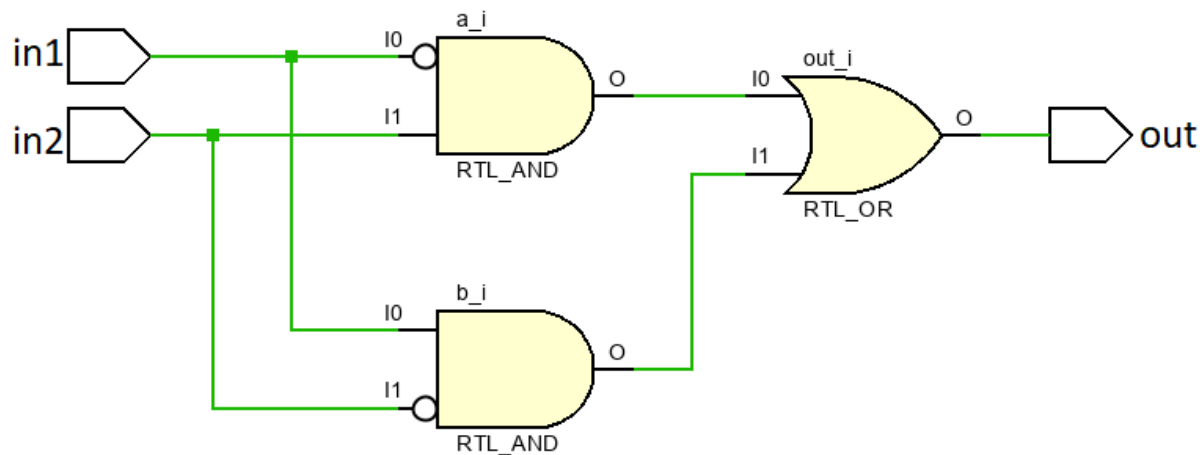
```
module basic_xor(  
    input  in1,  
    input  in2,  
    output out  
);
```

```
    logic a;  
    logic b;
```

```
    assign a = ~in1 & in2;  
    assign b = in1 & ~in2;
```

```
    assign out = a | b;
```

```
endmodule
```



# Тип данных logic

- Основной тип данных SystemVerilog – logic. Является типом данных по умолчанию для всех сигналов
- Тип logic – это одноразрядный тип; может иметь 4 значения: 0, 1, x (значение неизвестно), z (высокий импеданс)
- Значения x и z важны только при симуляции и помогают в выявлении ошибок схемы
- Чаще всего, если сигнал принимает значение x, значит он неинициализирован
- Значение z означает, что сигнал не подключен

# Иерархия модулей

- Внутри описания модуля можно определять другие модули, т.е. определять экземпляр (instance) модуля
- Когда модуль содержит экземпляр другого модуля, создается новый уровень иерархии проекта Vivado

```

module basic_xor(
    input  in1,
    input  in2,
    output out
);

    logic a;
    logic b;

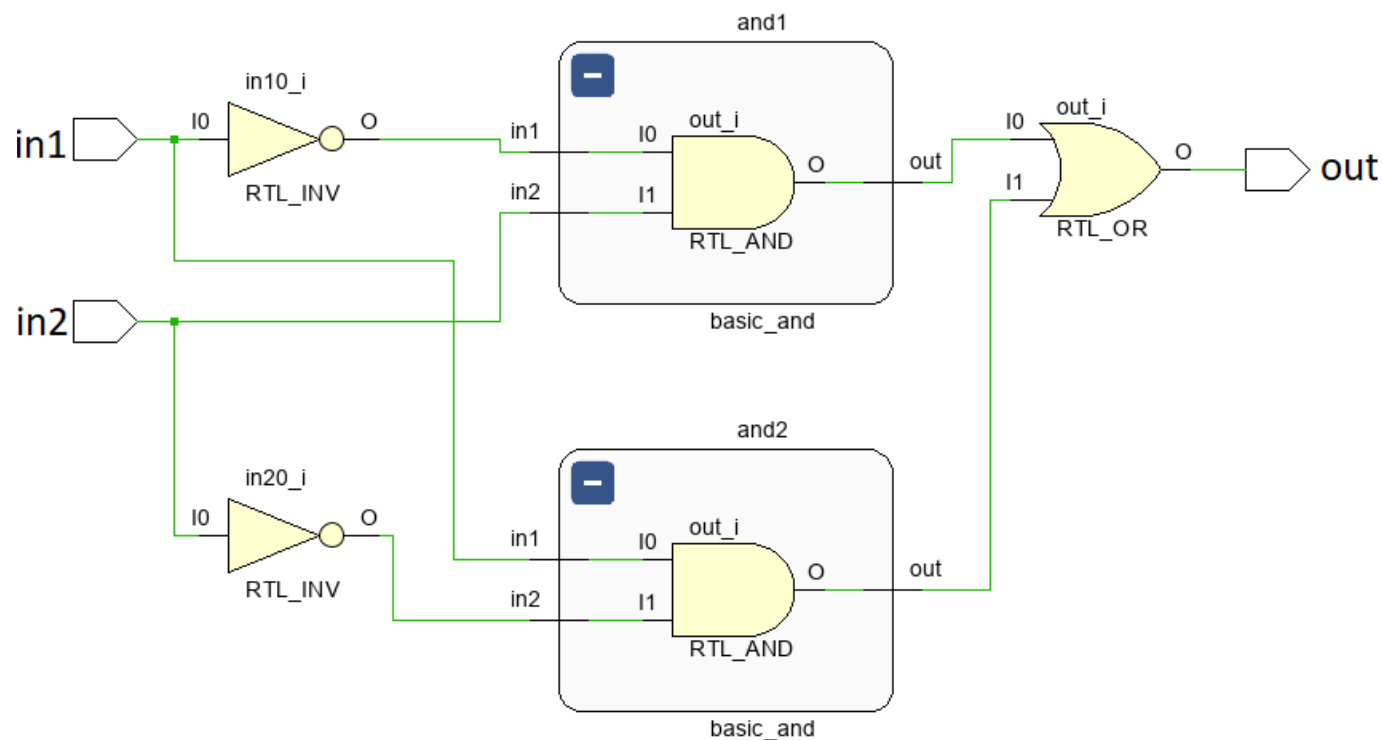
    basic_and and1(
        .in1(~in1),
        .in2(in2),
        .out(a)
    );

    basic_and and2(
        .in1(in1),
        .in2(~in2),
        .out(b)
    );

    assign out = a | b;

endmodule

```



- Сигналы можно объединять в шину

```
module bitwise_operations(
```

```
    input          in1,
```

```
    input          in2,
```

```
    output [3 : 0] out
```

```
);
```

```
    assign out[0] = ~in1;          // Побитовая инверсия
```

```
    assign out[1] = in1 & in2;     // Побитовое И
```

```
    assign out[2] = in1 | in2;     // Побитовое ИЛИ
```

```
    /*
```

```
    Побитовое
```

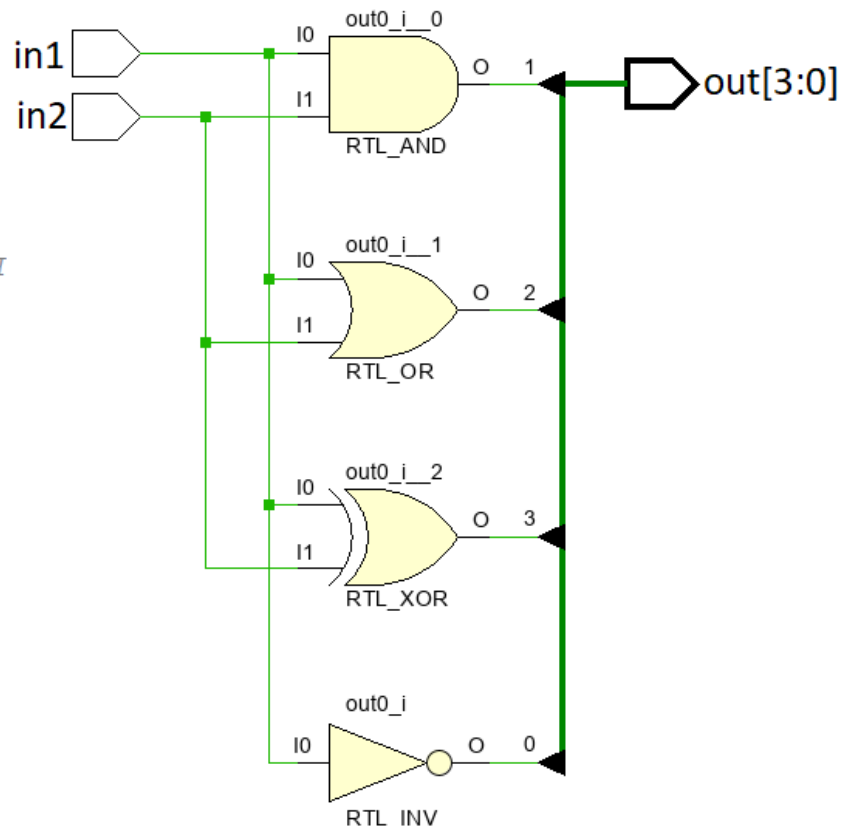
```
    исключающее
```

```
    ИЛИ
```

```
    */
```

```
    assign out[3] = in1 ^ in2;
```

```
endmodule
```



# Литералы

- SystemVerilog поддерживает целочисленные литералы. Они могут быть выражены разными способами. Для моделирования или синтеза компилятор SystemVerilog должен знать ряд характеристик каждого целочисленного литерала, включая его размер, знак или базу.
- База числа указывается с помощью апострофа, за которым следует символ, определяющий основание: d для десятичного числа, h для шестнадцатеричного, b для двоичного и o для восьмеричного ('b100 -32 битное значение без знака)
- Литералы с основанием могут иметь значения x или z ('hx)

- По умолчанию целочисленный литерал с основанием является беззнаковым. После апострофа можно добавить букву s, чтобы сделать значение знаковым
- Подпись значения изменит его поведение для определенных операций
- Разрядность целочисленного литерала также можно указать, добавив число перед апострофом

Литерал	Основание	Знак	Размер
16'd5	Десятичное	Беззнаковое	16 бит
4'sb0101	Двоичное	Знаковое	4 бит
12'h2ff	Шестнадцатичное	Беззнаковое	12 бит

- При назначении сигналу литерала можно указать большую или меньшую разрядность, чем количество бит, необходимое для представления значения. В таком случае старшие биты будут либо расширены нулями, либо усечены

```
logic [2:0] a, b, c;  
logic [3:0] d;  
logic [1:0] e;  
assign a = 3'b111;  
assign b = 'd7;  
assign c = '1;      // все 3 бита установятся в 1  
assign d = 3'h7;     // d = 4'b0111  
assign e = 7;        // e = 2'b11
```



# Параметры


- Параметры позволяют определить константы для модулей, вычисляемые на этапе компиляции
- Существует два типа параметров: те, значение которых можно задавать при создании экземпляра модуля (ключевое слово “parameter”) и параметры для внутреннего использования (ключевое слово “localparam”)
- Область видимости параметров ограничена экземпляром модуля, в котором он был объявлен
- При помощи parameter можно конфигурировать модуль, инстанцируя его несколько раз с разными значениями параметров
- localparam обычно используется внутри модуля для создания именованных констант
- parameter можно объявлять как внутри модуля наравне с localparam, так и в списке параметров модуля (ANSI-style)

```
localparam N0 = 4;  
localparam N1 = 8;
```

```
logic [N0-1:0] in0;  
logic [N0-1:0] out0;  
logic [N1-1:0] in1;  
logic [N1-1:0] out1;
```

```
not_gate #(N0) not_gate_4(  
    .in(in0),  
    .out(out0)  
);
```

```
not_gate #(N1) not_gate_8(  
    .in(in1),  
    .out(out1)  
);
```



```
module not_gate #(  
    parameter N = 4  
) (  
    input  [N-1:0]    in,  
    output [N-1:0]    out  
);  
    localparam logic [N-1:0] a = '1;  
    assign out = in ^ a;  
  
endmodule
```

# Основные операторы SystemVerilog

- Побитовые операторы

Оператор	Функция
$a \& b$	Побитовое И
$a   b$	Побитовое ИЛИ
$a \wedge b$	Побитовое исключающее ИЛИ
$\sim a$	Побитовое отрицание (инверсия)

```
logic [1:0] a;  
logic [1:0] b;  
logic [1:0] c;
```

=

```
logic [1:0] a;  
logic [1:0] b;  
logic [1:0] c;
```

```
assign c[0] = a[0] & b[0];  
assign c[1] = a[1] & b[1];
```

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

- Логические операторы

Оператор	Функция
<code>a &amp;&amp; b</code>	Логическое И
<code>a    b</code>	Логическое ИЛИ
<code>!a</code>	Логическое отрицание

- Операторы редукции (`&a`, `|a`, `^a`)

```
logic [2:0] a;
```

```
logic b;
```

=

```
logic [2:0] a;
```

```
logic b;
```

```
assign b = a[0] & a[1] & a[2];
```

```
assign b = &a;
```

- Арифметические операторы

Оператор	Функция
$a+b$	Суммирование
$a-b$	Вычитание
$-a$	Унарный минус
$a*b$	Умножение
$a/b$	Деление
$a\%b$	Модуль
$a**b$	Степень

- Операторы сдвига

Оператор	Функция
$a<<b$	Логический сдвиг влево
$a>>b$	Логический сдвиг вправо
$a<<<b$	Арифметический сдвиг влево
$a>>>b$	Арифметический сдвиг вправо

- Операторы сравнения

Оператор	Функция
a==b	Равно
a!=b	Не равно
a<b	Меньше
a<=b	Меньше или равно
a>b	Больше
a>=b	Больше или равно

- Конкатенация

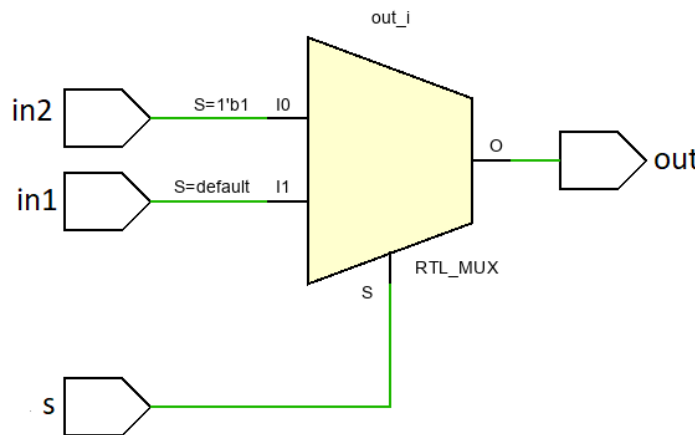
Оператор	Функция
{a,b}	Объединение a и b
{n{a,b}}	Объединение a и b n раз

```
logic [7 : 0] s;  
// s == 8'b10101011  
assign s = {{3{1'b1, 1'b0}}, 2'b11};
```

# Условный оператор

- В комбинации с непрерывным присваиванием описывает операцию мультиплексирования
- Имеет синтаксис и поведение аналогичное тернарному оператору языка C

```
module mux (  
    input  in1,  
    input  in2,  
    input  s,  
    output out  
);  
  
    assign out = s ? in2 : in1;  
  
endmodule
```



s	in2	in1	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

# Процедурный блок `always_comb`

- Используется для описания комбинационной логики
- Позволяет описывать комбинационные схемы
- Допускает использование в своем теле условных операций `if-else`, `case`, циклов `for`, `while` и `do-while`
- Команды внутри блока выполняются последовательно
- Ключевые слова `begin` и `end` являются необязательными, если блок содержит одну инструкцию



```

module mux (
    input logic in1,
    input logic in2,
    input logic s,
    output logic out
);
    always_comb begin
        if(s)
            out = in2;
        else
            out = in1;
        end
    end
endmodule

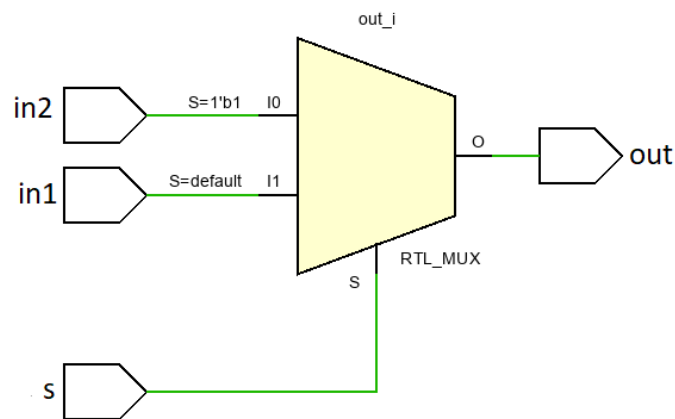
```

=

```

module mux (
    input logic in1,
    input logic in2,
    input logic s,
    output logic out
);
    always_comb begin
        case(s)
            1'b0: out = in1;
            1'b1: out = in2;
        endcase
    end
endmodule

```



# Тестбенч

- Тестбенч - модуль, задающий тестовое окружение для тестируемого модуля
- Обычно не имеет портов, содержит внутри себя инстанцированные тестируемые модули и их тестовое окружение
- В качестве тестового окружения могут выступать средства (модули, классы, процессы и т.п.), формирующие входные воздействия для тестируемых модулей, а так же средства проверки результатов тестируемых модулей

# Литература

- Дэвид М. Хэррис, Сара Л. Хэррис:  
“Цифровая схемотехника и архитектура  
компьютера”
- IEEE Std 1800-2017 Standard for  
SystemVerilog - Unified Hardware Design,  
Specification, and Verification Language