Введение в маршрут проектирования и упражнения с комбинационной логикой. Testbench для комбинационной логики.

школа синтеза цифровых схем

ПРИ ПАРТНЕРСТВЕ

YADRO · MP

Syntacore™

Александр Силантьев

Руководитель лаборатории ЦКП МИЭТ

Александр Силантьев

Руководитель лаборатории ЦКП МИЭТ

Окончил МИЭТ в 2014 году

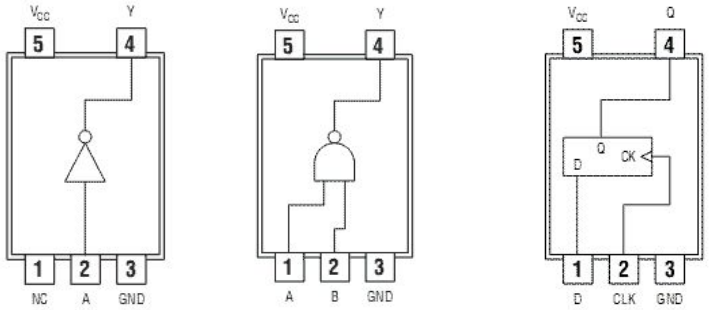
12 летний опыт инженерной деятельности

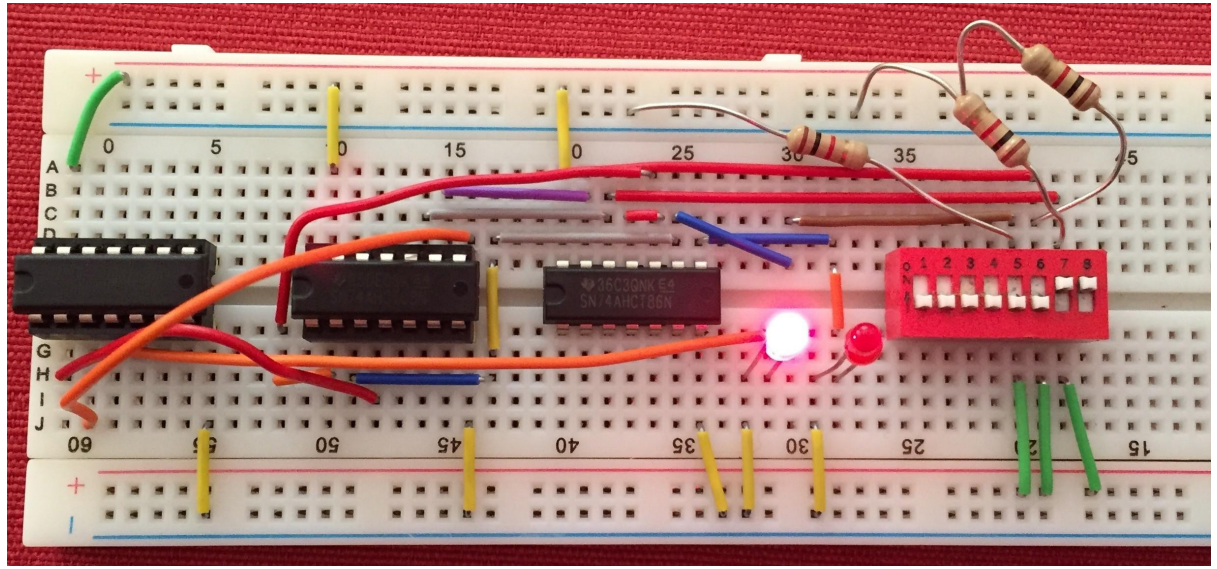
Старший преподаватель института МПСУ МИЭТ.

Организатор первого в России инженерного хакатона по микроэлектронике и системам на кристалле.

С 2014 года организатор семинаров, школ и олимпиад по популяризации электроники среди студентов и школьников

Микросхемы малой степени интеграции





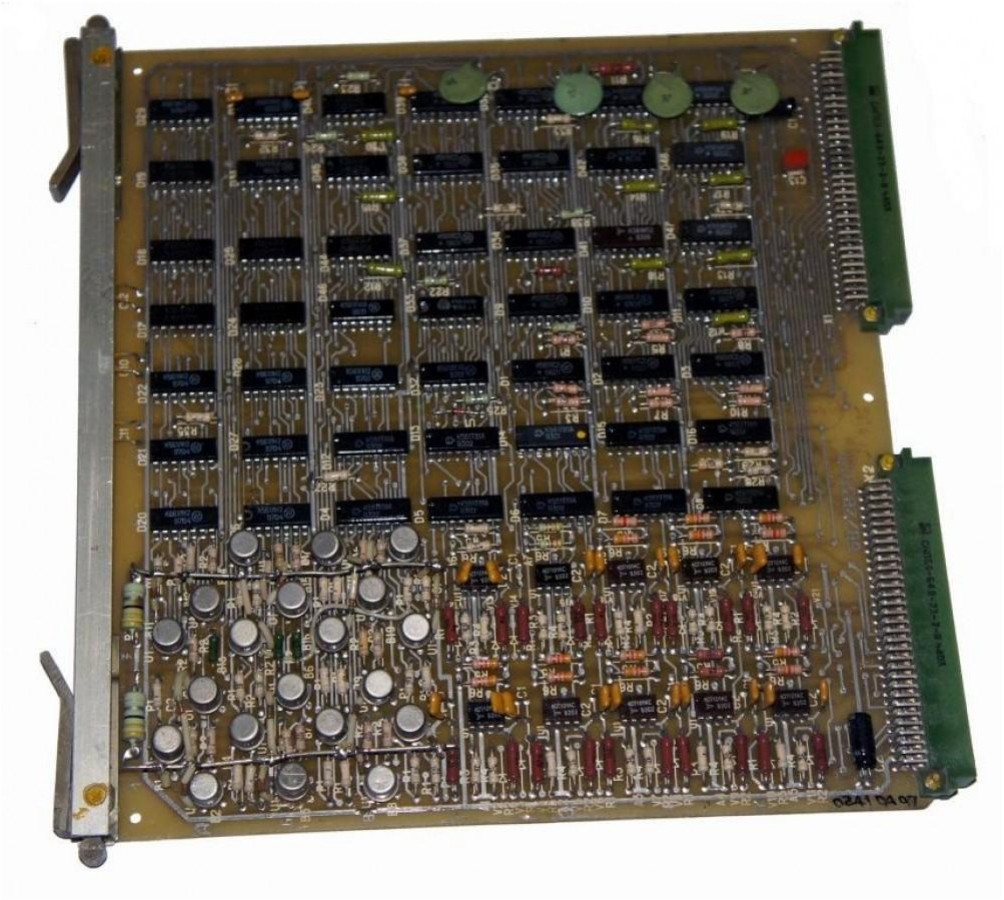


Схема intel 4004

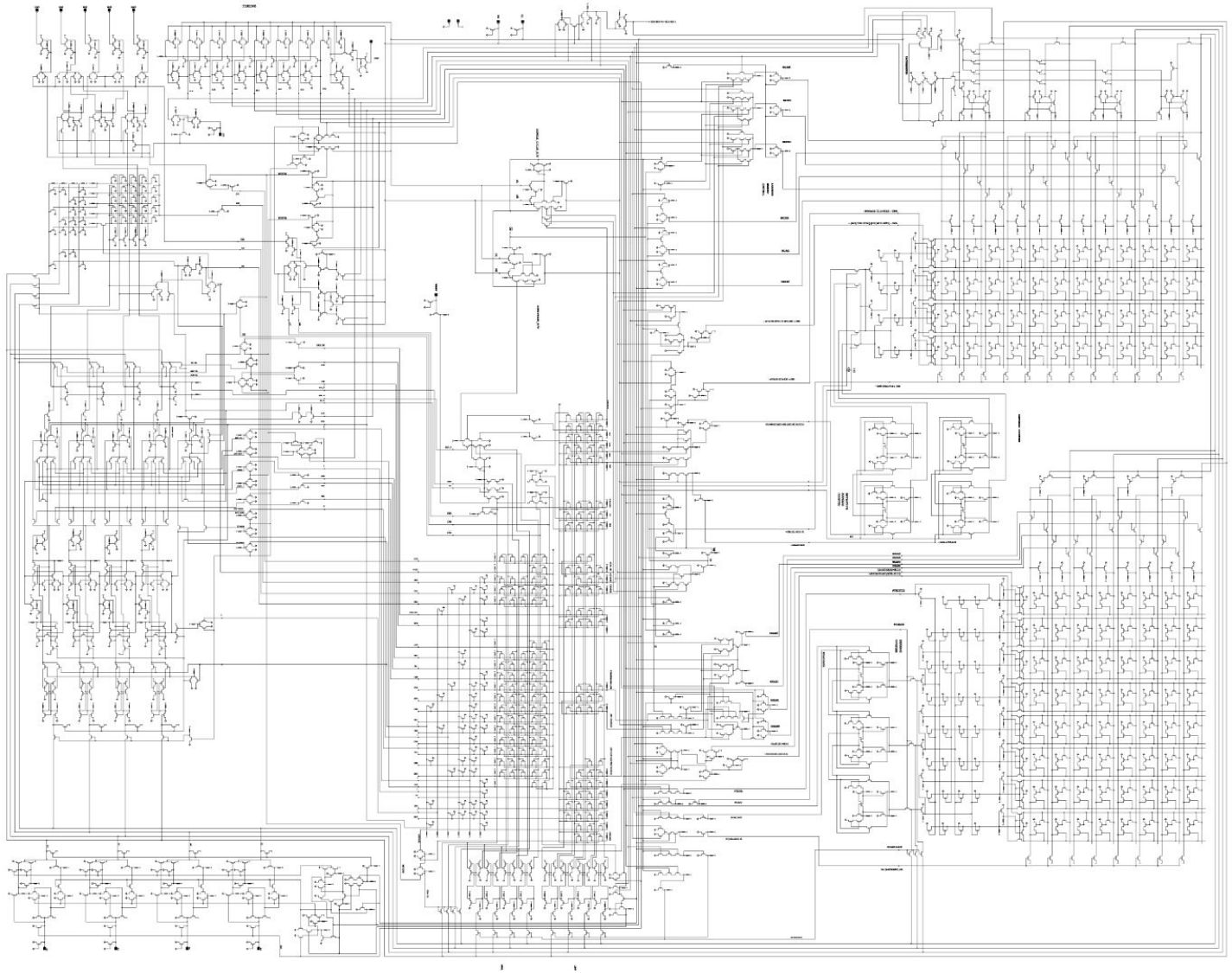


Схема intel 4004

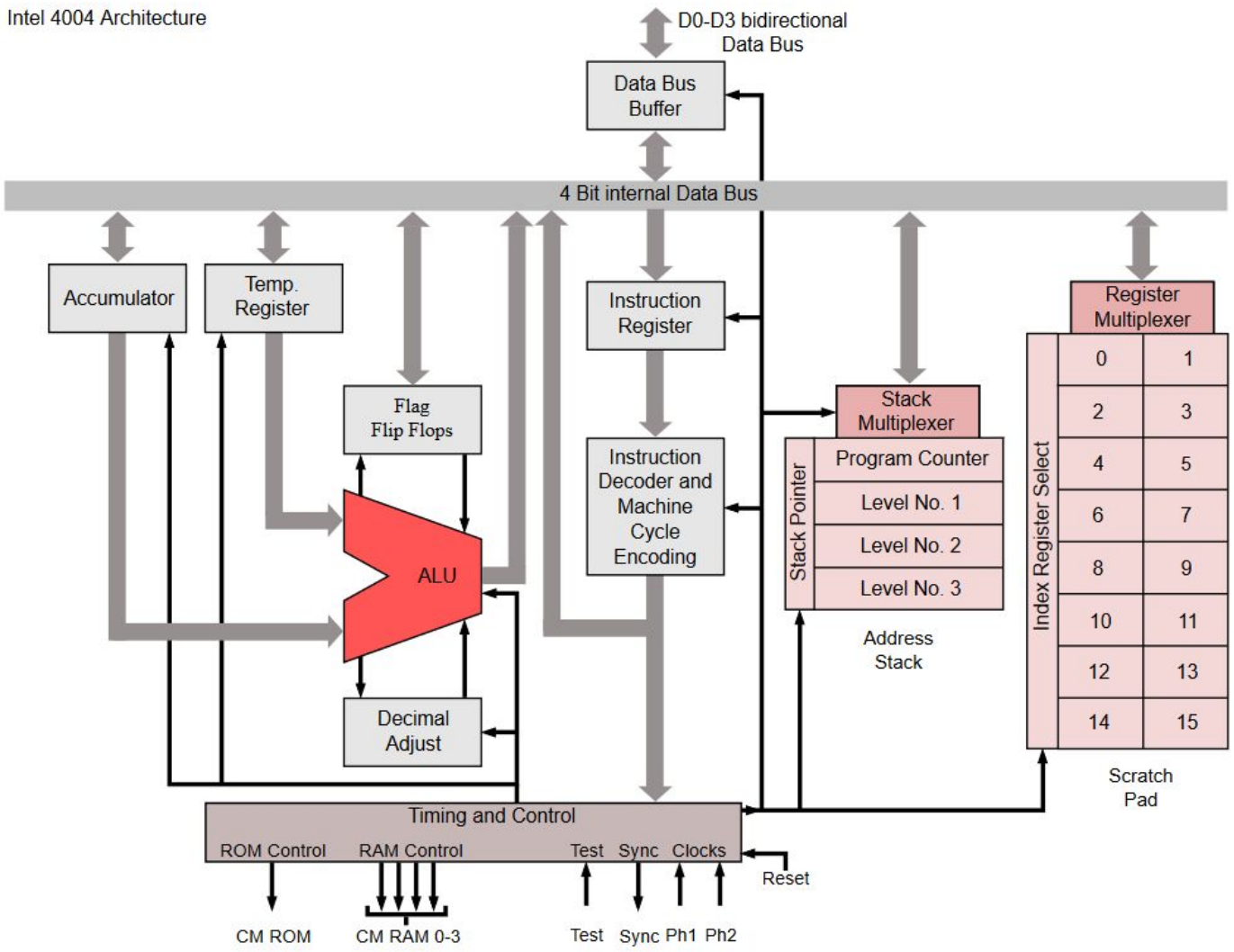
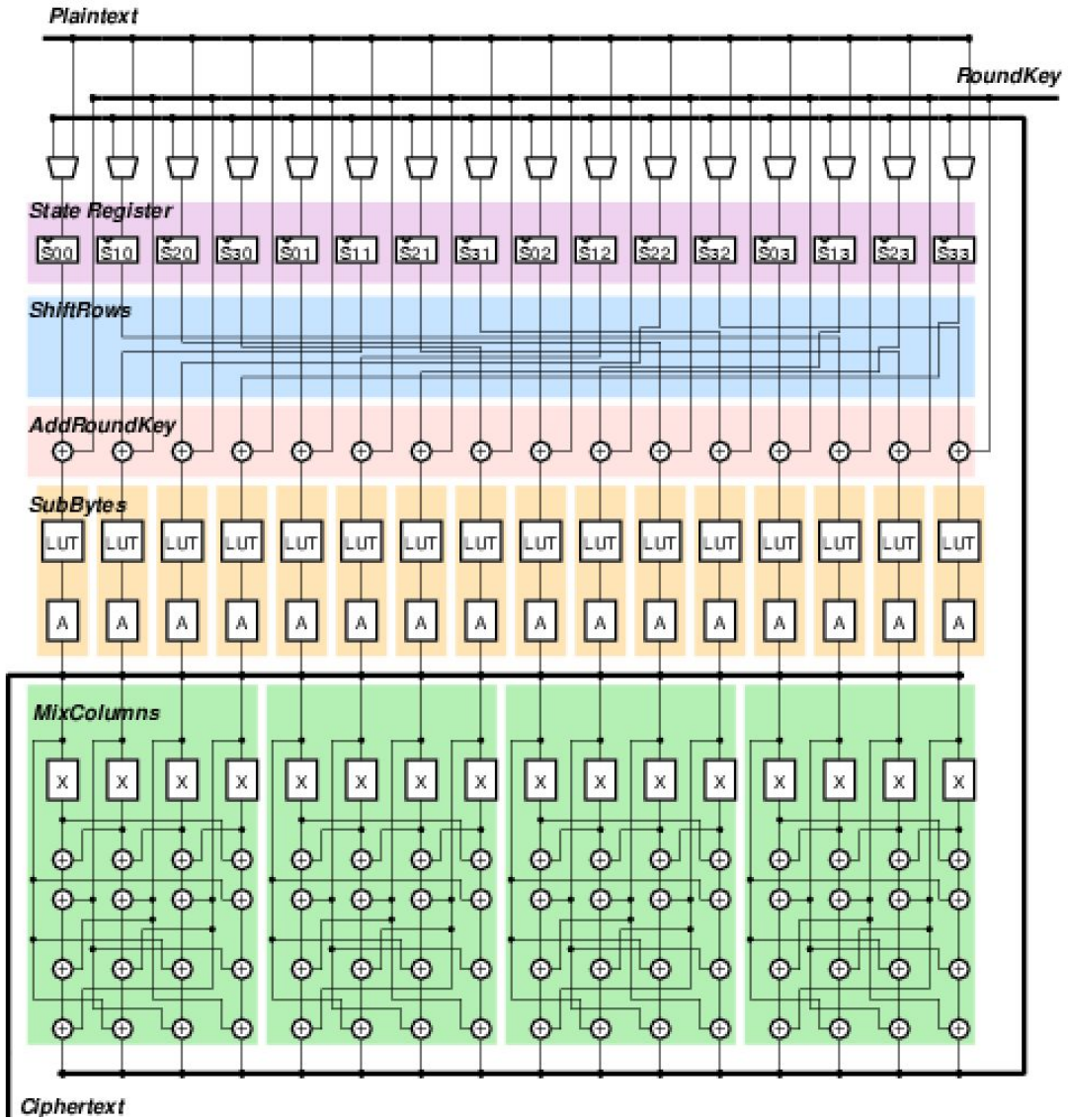


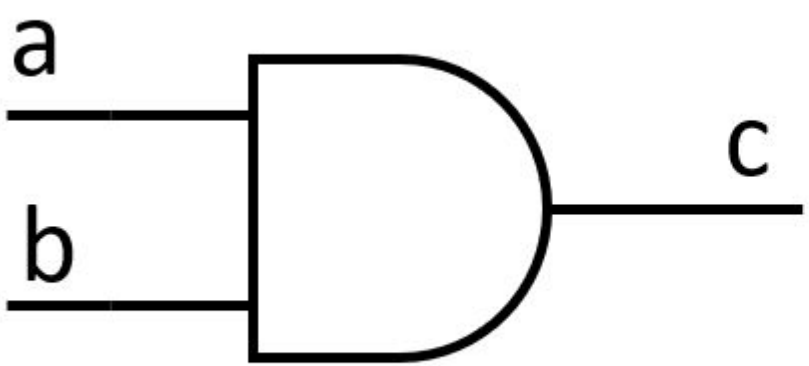
Схема блока аппаратного шифрования AES



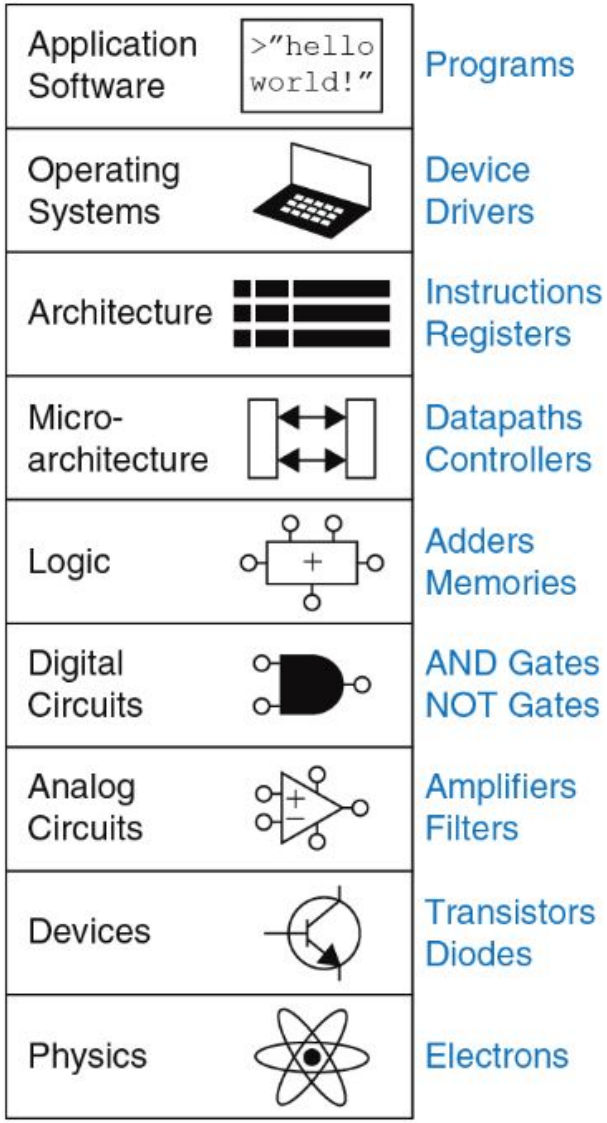
Управление сложностью в микроэлектронике

HDL

HDL – Hardware Description Language




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



HDL. Выбор

SystemVerilog Verilog VHDL

Verilog HDL. История

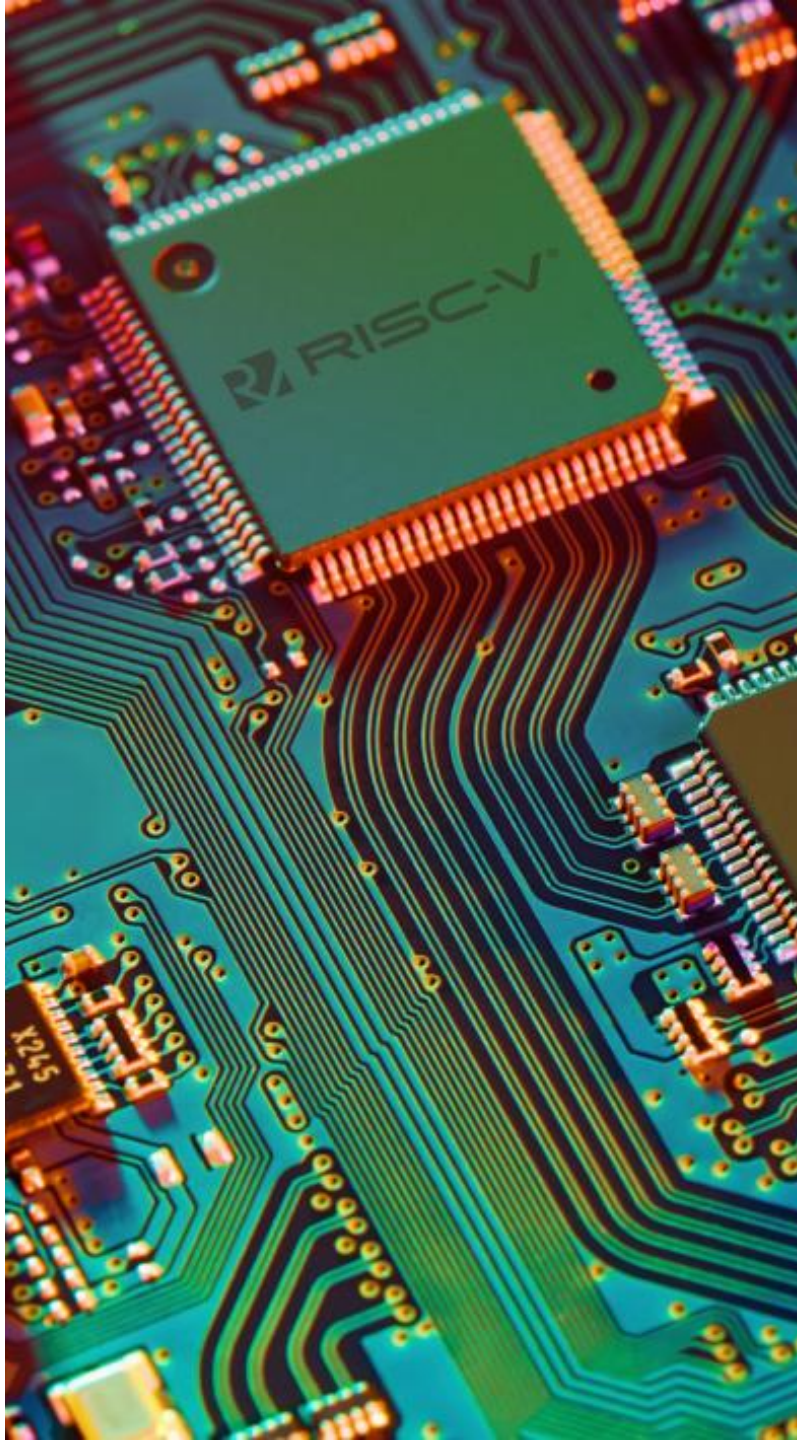
Verilog был разработан компанией Gateway Design Automation в 1984 году как фирменный язык для симуляции логических схем.

В 1989 году Gateway приобрела компания Cadence, и Verilog стал открытым стандартом в 1990 году под управлением сообщества Open Verilog International.

Язык стал стандартом IEEE в 1995 году.

В 2005 году язык был расширен для упорядочивания и лучшей поддержки моделирования и верификации систем.

Эти расширения были объединены в единый стандарт, который сейчас называется SystemVerilog(стандарт IEEE 1800-2009).

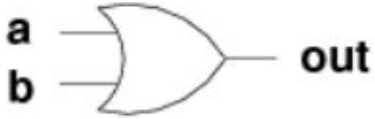


Комбинационная логика

AND

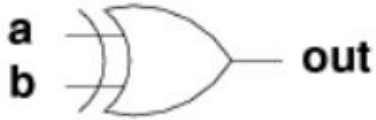
a	b	out
0	0	0
0	1	0
1	0	0
1	1	1

OR



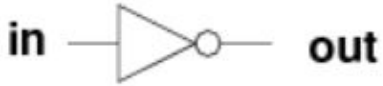
a	b	out
0	0	0
0	1	1
1	0	1
1	1	1

XOR



a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

NOT



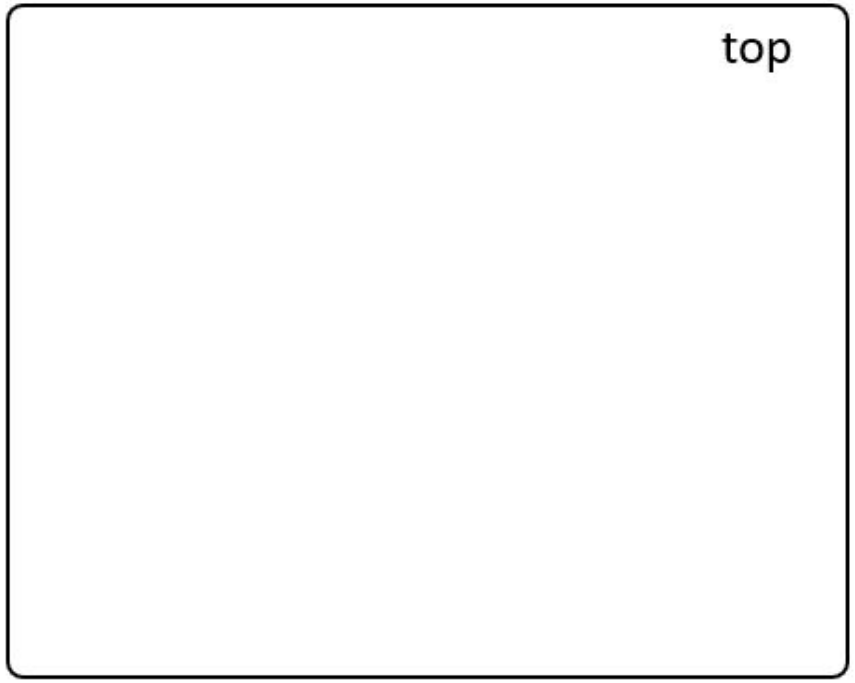
in	out
0	1
1	0

Verilog HDL

```
module

endmodule
```

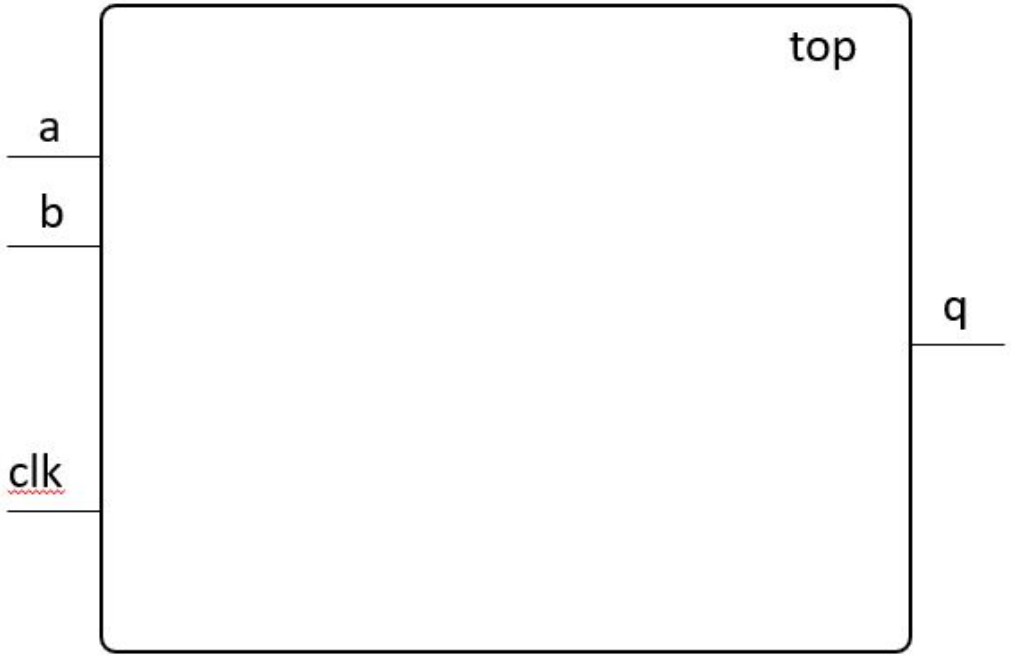
Verilog HDL

```
module top

endmodule
```



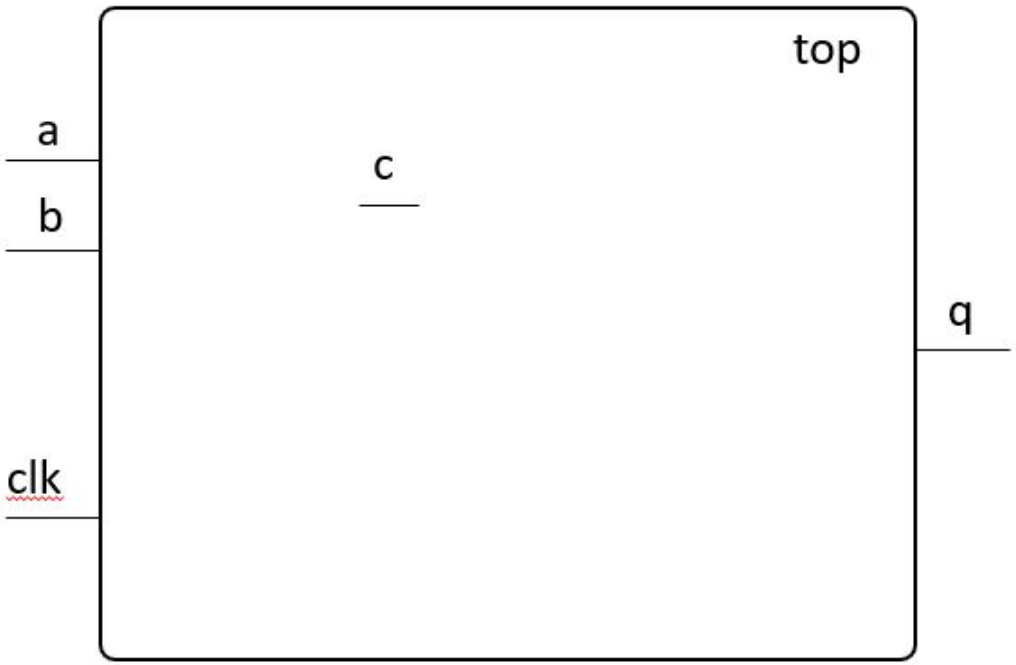
Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);



endmodule
```



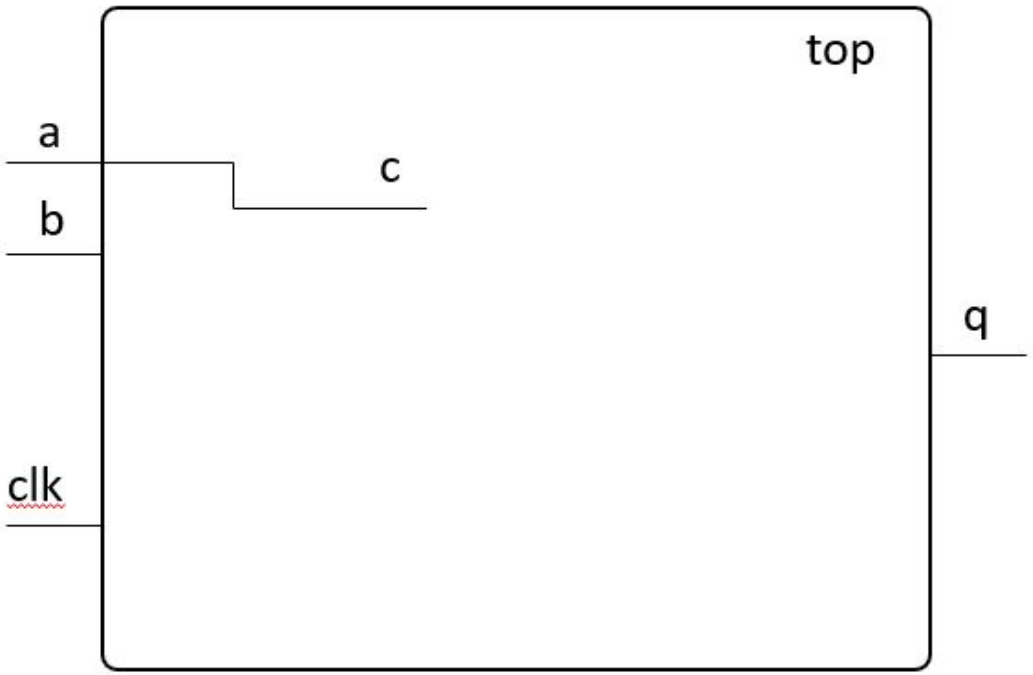
Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);

wire c;



endmodule
```



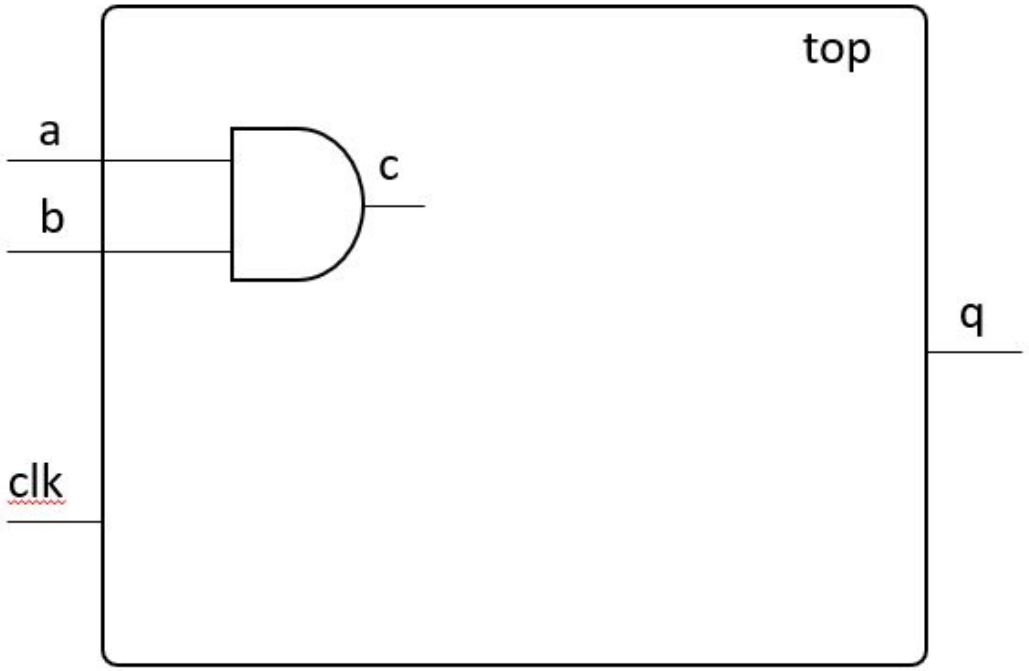
Verilog HDL

```
module top (
        input       a,
        input       b,
        input       clk,
        output      q
);

wire c;


assign c = a;

endmodule
```



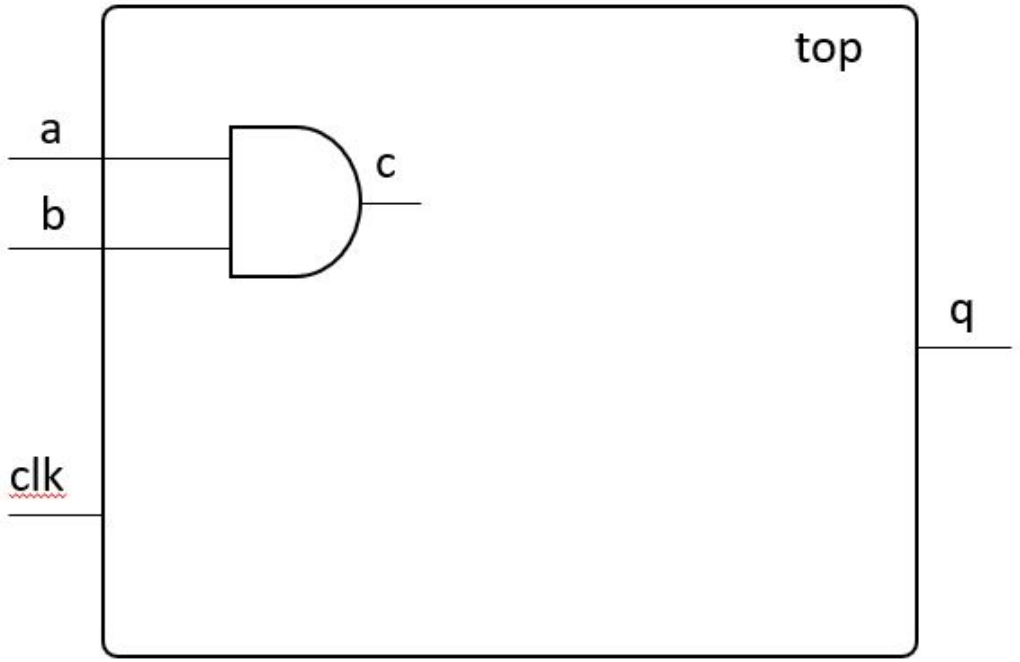
Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);

wire c;


assign c = a & b;

endmodule
```



Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);

wire c;

// Это комментарий, он ничего не изменяет
assign c = a & b;

endmodule
```



Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        inout       q    // двунаправленный сигнал, используется для внешних контактов микросхем
);

wire c;

// Это комментарий, он ничего не изменяет
assign c = a & b;

endmodule
```

Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);

wire [3:0] c; // 4-х проводная шина
wire [3:0] d;

assign c = d;

endmodule
```

Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);

wire [3:0] c, d; // объявление нескольких проводов одинаковой разрядности

assign c = d;

endmodule
```

Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);

wire [3:0] c;
wire [8:0] d;

assign q = d[3]; // можно присвоить нужный бит.

endmodule
```

Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);

wire [3:0] c;
wire [8:0] d;

assign q = d[3]; // можно присвоить нужный бит.

endmodule
```

Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);

wire [3:0] c;
wire [8:0] d;

assign c = d[7:4]; // можно назначить нужные биты другому проводу.

endmodule
```

Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);

wire [3:0] c;
wire [8:0] d;

assign q = d[c]; // можно назначить номер бита определяемой шиной.

endmodule
```

Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);

wire [3:0] c;
wire [7:0] d [0:24]; // массив из двадцатипяти 8-битных шин

assign c = d[14][7:4]; // подключение 4 бит из 14 шины массива шин.

endmodule
```

Verilog HDL

```
module top (
        input               clk,
        input     [8:0]     a, // входные и выходные порты модуля могут быть многоразрядными шинами.
        input               b,
        output    [3:0]     q
);

assign q = a[7:4];
endmodule
```

Формат описания чисел Verilog HDL

```
wire [10:0] a = 7; //32-х битное десятичное число, которое будет “обрезано” до 11 бит
wire [10:0] b = ’d7; //11-ти битное десятичное число
wire [10:0] b = 11’d7; //11-ти битное десятичное число
wire [3:0]   c  = 4'b0101; //4-х битное двоичное число
wire [3:0]   d  = 8’h7B;   //8-ми битное шестнадцатеричное число 7B
wire [47:0] e = 48’hEFCA7ED98F;  //48-ми битное шестнадцатеричное число

wire signed [10:0] b = -11’d7; //11-ти битное отрицательное десятичное число
```

Основные операции Verilog HDL

Символ	Назначение
{}	Конкатенация (concatenation)
+ - * /	Арифметические (arithmetic)
%	Модуль (modulus)
> >= < <=	Отношения (relational)
!	Логическое отрицание (logical NOT)
&&	Логическое И (logical AND)
||	Логическое ИЛИ (logical OR)

Основные операции Verilog HDL

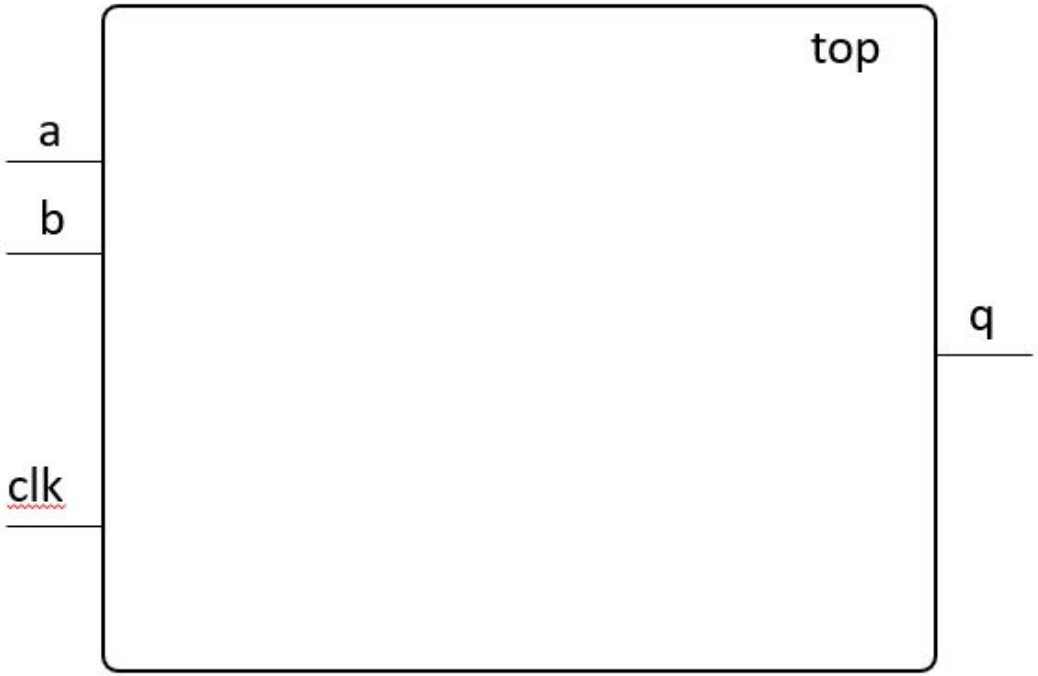
Символ	Назначение
==	Логическое равенство (logical equality)
!=	Логическое неравенство (logical inequality)
===	Идентичность (case equality)
!===	Не идентичность (case inecuality)
~	Побитовая инверсия (bit-wise NOT)
&	Побитовое И (bit-wise AND)
|	Побитовое ИЛИ (bit-wise OR)

Основные операции Verilog HDL

Символ	Назначение
<<	Сдвиг влево (left shift)
>>	Сдвиг вправо (right shift)
<<<	Циклический сдвиг влево (arithm. left shift)
>>>	Циклический сдвиг вправо (arithm. right shift)
?:	Тернарный оператор (ternary)

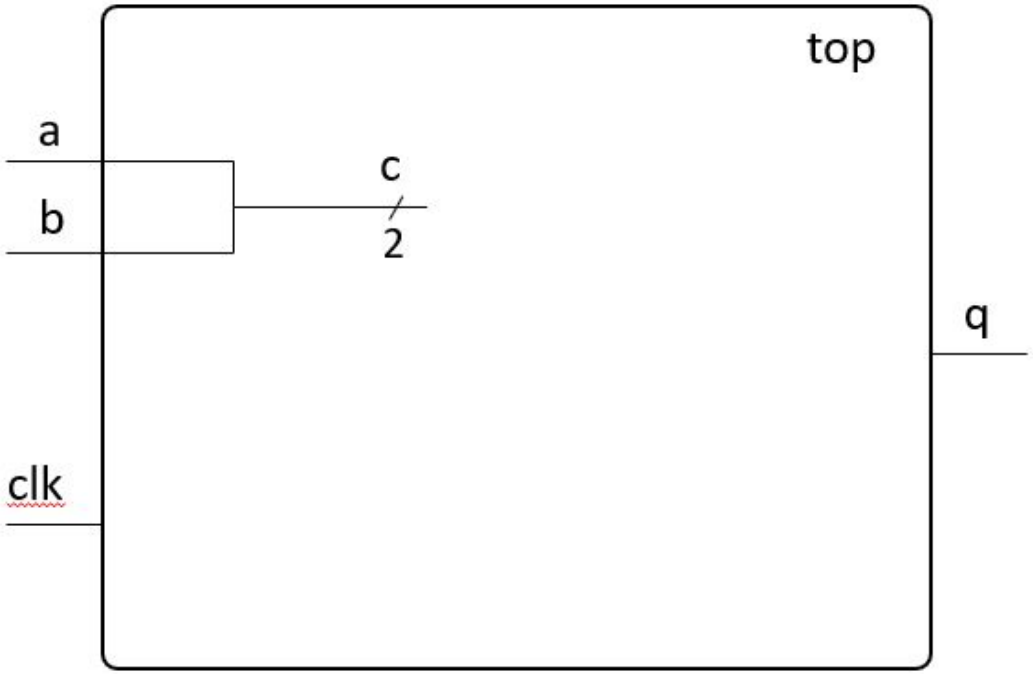
Манипуляции с битами Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);



endmodule
```



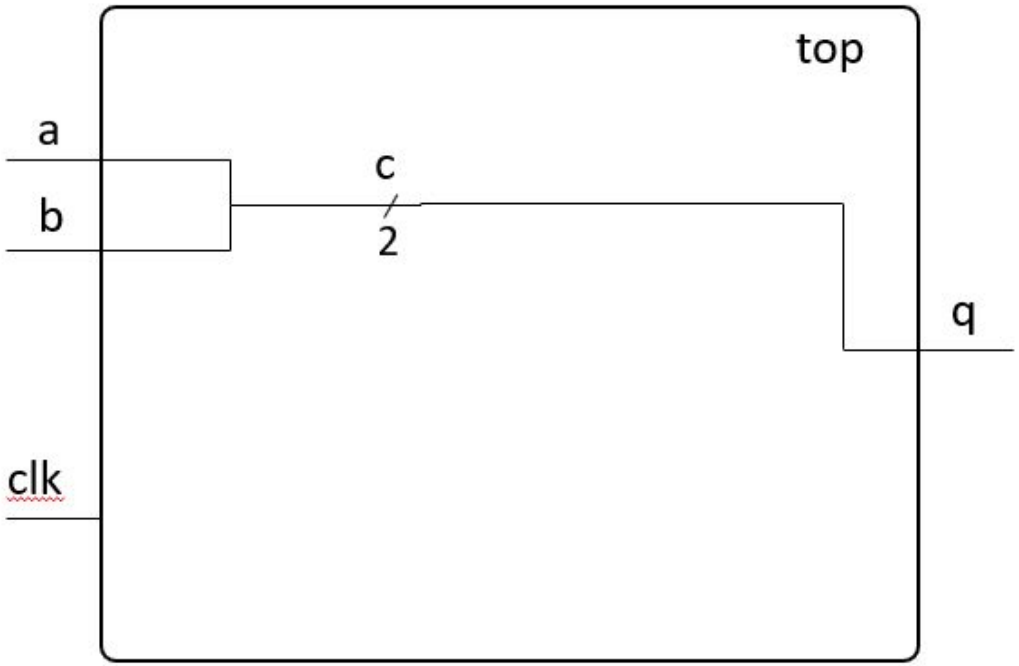
Манипуляции с битами Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);

wire [1:0] c; // Многобитный сигнал (шина)

assign c = {a,b}; // Конкатенация


endmodule
```



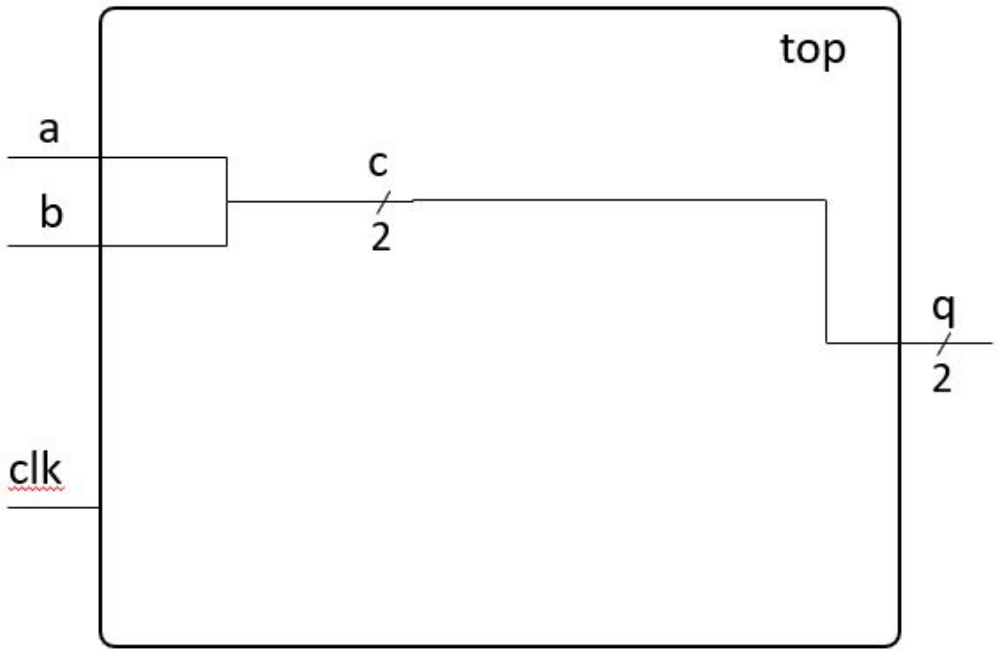
Манипуляции с битами Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);

wire [1:0] c; // Многобитный сигнал (шина)

assign c = {a,b}; // Конкатенация

assign q = c[0]; // Обращение к биту

endmodule
```



Манипуляции с битами Verilog HDL

```
module top (
        input       clk,
        input       a,
        input       b,
        output      q
);

wire [1:0] c; // Многобитный сигнал (шина)

assign c = {a,b}; // Конкатенация

assign q = c;

endmodule
```



Verilog HDL. Сложение и вычитание

```
module simple_add_sub (
        input    [7:0]    operandA, operandB //два входных 8-ми битных операнда
        output [8:0]    out_sum, out_dif        // Выходы для арифметических операций имеют дополнительный 9-й бит переполнения
);
//Максимальное 8-ми битное безнаковое число 255. При сложении 255 + 255 получится 510 которое можно представить минимум 9-ю битами.
assign out_sum = operandA + operandB;
assign out_dif = operandA - operandB;
//Если сделать выход сумматора 8-ми битным то случится переполнение и результат сложения 255 + 3 будет равен 2.
endmodule
```

Verilog HDL. Операции над знаковыми данными

```
module simple_add_sub (
        input   signed  [7:0]    operandA, operandB //два входных знаковых 8-ми битных операнда
        output signed [8:0]    out_sum, out_dif       // Выходы для арифметических операций имеют
дополнительный 9-й бит переполнения
// для корректного выполнения знаковых операция. Операнды и результат должны быть объявлены как signed
);
//Минимальное 8-ми битное знаковое число -128. Максимальное 8-ми битное знаковое число 127. При
сложении -128 + 127 получится -1.
assign out_sum = operandA + operandB;
assign out_dif = operandA - operandB;
endmodule
```

Verilog HDL. Умножение и деление

```
module simple_add_sub (
    input   [7:0]   operandA, operandB //два входных 8-ми битных операнда
    output [15:0]   out_mul, out_div, out_rem      // Для умножения чтобы избежать переполнения выходная разрядность должна быть не меньше чем сумму разрядностей операндов для беззнакового случая и сумма разрядностей операндов -1 для знакового. Для данного случая это 16 бит и 15 бит. Для знакового случая разрядность меньше из за наличия старшего бита знака.
);

assign out_mul = operandA * operandB; // Умножение 8 * 8 = 64
assign out_div  = operandA / operandB; // Деление 16 / 3 = 5 Округление до целого числа вниз
assign out_rem = operandA % operandB; // Остаток от деления 16 % 3 = 1
//ВАЖНО. Использование операций деления и остатка от деления не используется в синтезируемом коде только для симуляции. Так как дает очень сложную реализацию по площади и очень плохими таймингами. Например в процессорах деление реализуется итерационными алгоритмами.
endmodule
```

Verilog HDL. Логические и арифметические сдвиги

```
module simple_add_sub (
        input    [7:0]    operandA, operandB //два входных 8-ми битных операнда
        output [7:0]    out_sll, out_slr, out_sar //Выходы для операций сдвига
);

//логический сдвиг влево на значение в  operandB.
assign   out_sll = operandA << operandB;

// пример: на сколько сдвигать определяется 3-мя битами второго операнда. Например operandA =
8'b1010_1110 operandB = 8'b0000_0011 тогда out_slr = 8'b0001_0101
assign   out_slr = operandA >> operandB[2:0];
//арифметический сдвиг вправо (сохранение знака числа) Например operandA = 8'b1111_1100 тогда out_sar =
8'b1111_1111
assign   out_sar = operandA >>> 3;

endmodule
```

Verilog HDL. Битовые логические операции

```
module simple_add_sub (
        input   [7:0]   operandA, operandB //два входных 8-ми битных операнда
        output [7:0]    out_bit_and , out_bit_or, out_bit_or, out_bit_not
);

assign  out_bit_and = operandA & operandB; //8'b0011_1101 & 8'b1010_0110 = 8'b0010_0100
assign  out_bit_or   = operandA | operandB;   //8'b0011_1101 | 8'b1010_0110 = 8'b1011_1111
assign  out_bit_or   = operandA ^ operandB;  //8'b0011_1101 ^ 8'b1010_0110 = 8'b1001_1011
assign  out_bit_not  = ~operandA;                // ~8'b0011_1101 = 8'b1100_0010

endmodule
```

Verilog HDL. Булевые логические операции

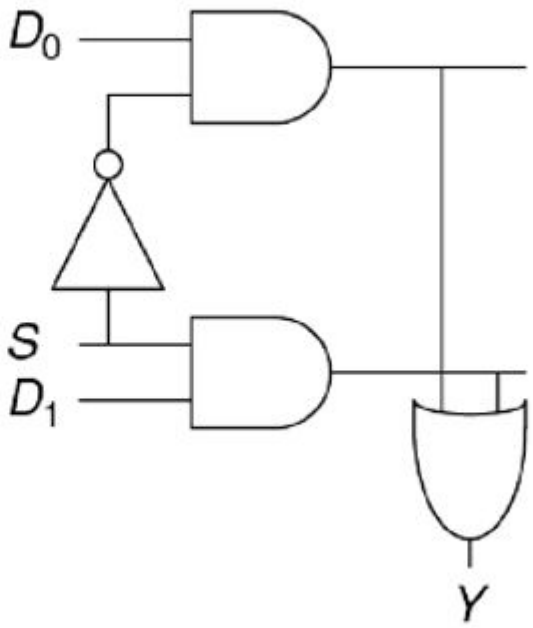
```
module simple_add_sub (
        input   [7:0]   operandA, operandB //два входных 8-ми битных операнда
        output          out_bool_and , out_bool_or  , out_bool_not
);

assign  out_bool_and = operandA && operandB; //8'b0011_1101 && 8'b1010_0110 = 1'b1
assign  out_bool_or   = operandA || operandB;   //8'b0011_1101 || 8'b1010_0110 = 1'b1
assign  out_bool_not  = !operandA;              // !8'b0011_1101 = 1'b0

endmodule
```

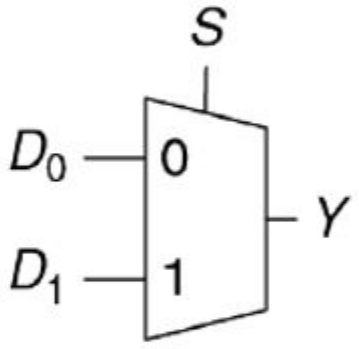
Verilog HDL. Операции свертки

```
module simple_add_sub (
        input    [7:0]    operandA,
        output            out_reduction_and, out_reduction_or, out_redution_xor
);

//Операции выполняются между битами внутри одной шины
assign   out_reduction_and = &operandA;  //&8'b0011_1101 = 1'b0
assign   out_reduction_or   = |operandA;  //|8'b0011_1101  = 1'b1
assign   out_redution_xor   = ^operandA;  // ^8'b0011_1101  = 1'b1

endmodule
```

Verilog HDL. Мультиплексор

```
module mul (
        input       D0,
        input       D1,
        input       S,
        output      Y
);

assign Y = S ? D1 : D0;

endmodule
```





Verilog HDL. Операции сравнения

```
module simple_add_sub (
        input    [7:0]    operandA, operandB
        output            out_eq, out_ne, out_gt, out_lt, out_ge, out_le
);

assign   out_eq = operandA == operandB;
assign   out_ne = operandA != operandB;
assign   out_ge = operandA >= operandB;
assign   out_le  = operandA <= operandB;
assign   out_gt  = operandA > operandB;
assign   out_lt   = operandA < operandB;


endmodule
```

Verilog HDL. Тип reg

```
module top (
        input           clk,
        input   [8:0]   a,
        input           b,
        output  [3:0]   q
);
```

// reg используют при поведенческом (behavioral) описании схемы. Если регистру постоянно присваивается значение комбинаторной (логической) функции, то он ведет себя точно как провод (wire). Если же регистру присваивается значение в синхронной логике, например по фронту сигнала тактовой частоты, то ему, в конечном счете, будет соответствовать физический D-триггер или группа D-триггеров. D-триггер – это логический элемент способный запоминать один бит информации.

```
reg [3:0] c;

endmodule
```

Verilog HDL. Тип reg

```
module top (
        input                   clk,
        input       [8:0]       a,
        input                   b,
        output reg [3:0]        q // выходные сигналы и шины можно объявлять как reg
);

endmodule
```

Verilog HDL. Блок always

```
module top (
        input                   clk,
        input       [8:0]       a,
        input                   b,
        output reg  [3:0]       q
);
//Присвоение в блоке always возможно только для сигналов типа reg!!!!
always @(*) begin
        q = !a[7:4];
end

endmodule
```

Verilog HDL. Блок always

```
//Связанные между собой выражения превратятся в общую цепь комбинационной логики
wire  [3:0] a, b, c, d, e;
reg   [3:0] f, g, h, j;
always@(*) begin
f = a + b;
g = f & c;
h = g | d;
j = h - e;
end
```

Verilog HDL. Блок always

```
//Связанные между собой выражения превратятся в общую цепь комбинационной логики
wire  [3:0] a, b, c, d, e;
reg   [3:0] f, g, h, j;
always@(*) begin
j = (((a + b) & c) | d) - e;
end
```

Verilog HDL. If-else

```
//можно описывать мультиплексоры с помощью if-else а не тернарного оператора
reg  [3:0] c;
always @(a or b or d) begin
      if (d)  begin
          c = a & b;
      end  else begin
          c = a + b;
      end
end
```

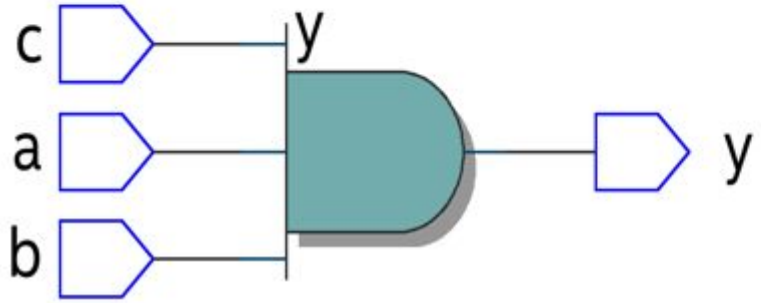
Verilog HDL. Case

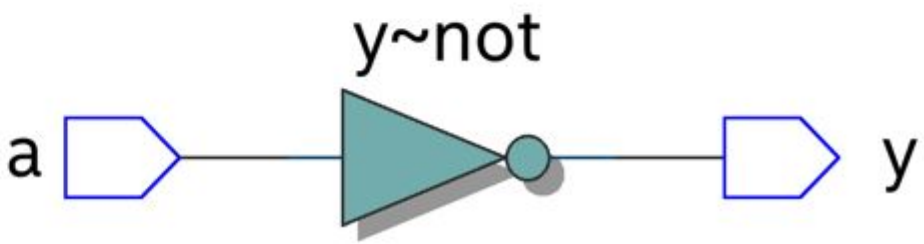
```
//многовходовые мультиплексоры и дешифраторы можно описывать через case
reg  [3:0] c;
wire  [1:0] option;
wire  [7:0] a, b, c, d;
reg    [7:0] e;
always @(a or b or c or d or option) begin
case (option)
  0: e = a;
  1: e = b;
  2: e = c;
  3: e = d;
endcase
end
```

Verilog HDL. Иерархия модулей

```
module and_3 (
        input     a, b, c,
        output    y
);

assign y = a & b & c;

endmodule
```

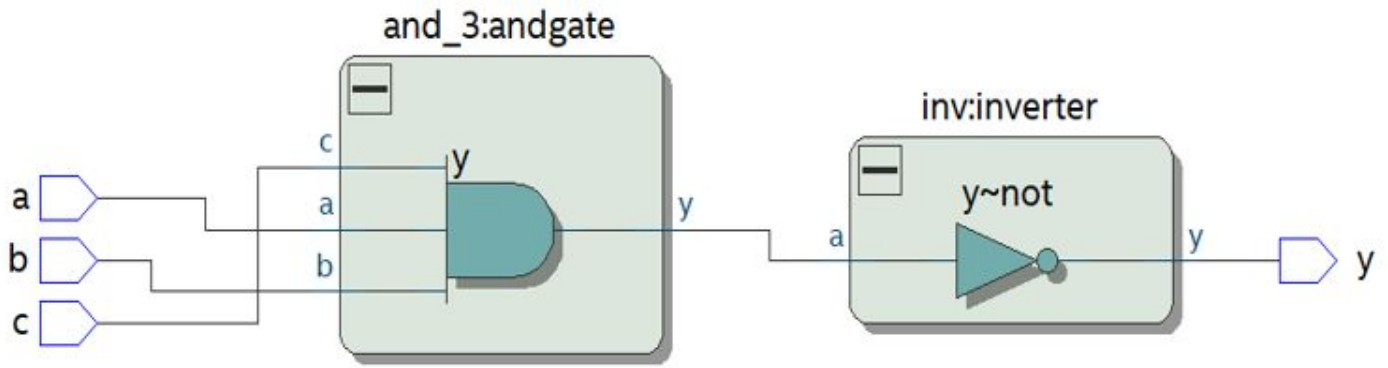
```
module inv (
        input     a
        output    y
);

assign y = ~a;

endmodule
```





Verilog HDL. Иерархия модулей

```
module top (
        input     a, b, c,
        output    y
);

wire n1;

and_3 andgate (
        .a(a), .b(b),
        .c(c), .y(n1)
);

inv inverter (
        .a(n1), .y(y)
);

endmodule
```

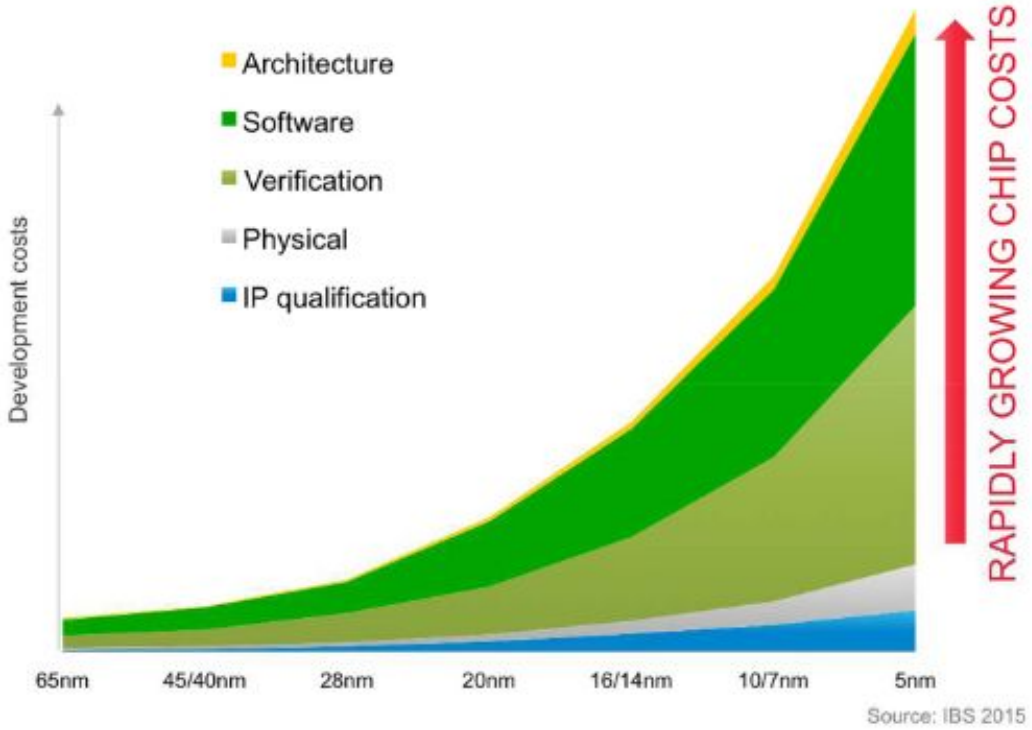


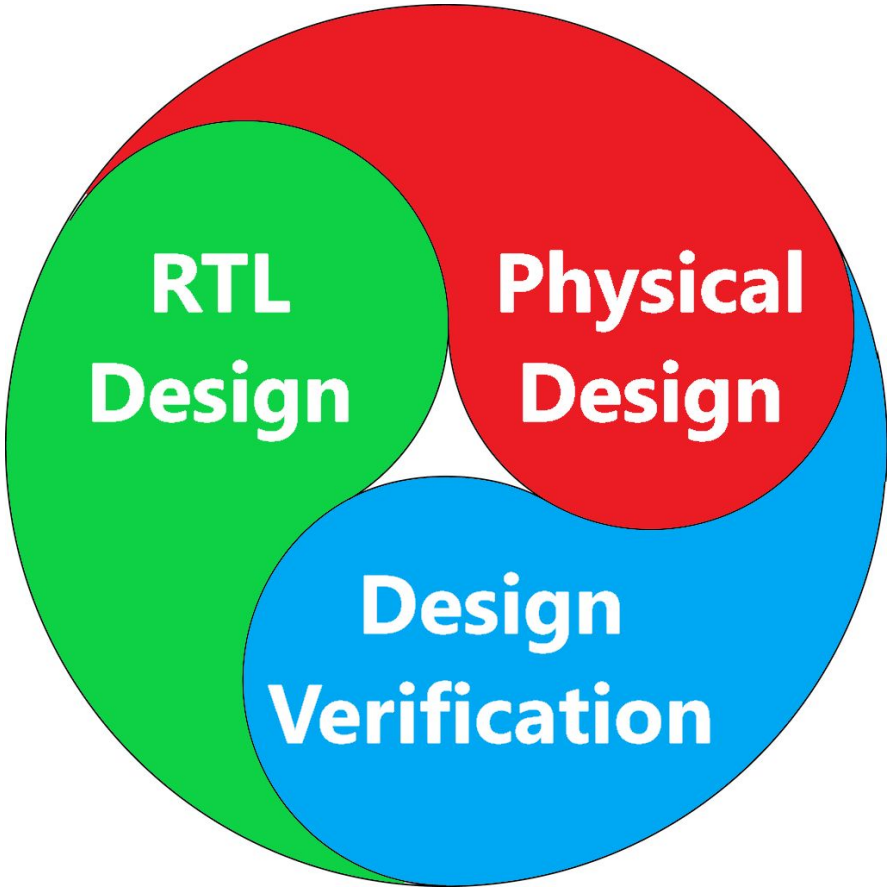
Имя подключаемого **модуля** (and_3, inv)

Название примитива. Например, нам может понадобиться 3 копии модуля and_3. Тогда мы сможем подключить 3 экземпляра модуля **and_3**, используя различные наименования для прототипов (andgate_1, andgate_2 ...)

Символ **точка**, перед наименованием порта отсылает к реальному порту подключаемого модуля. **В скобках** обозначается , куда будут подключаться сигналы в top-модуле

Верификация и симуляция

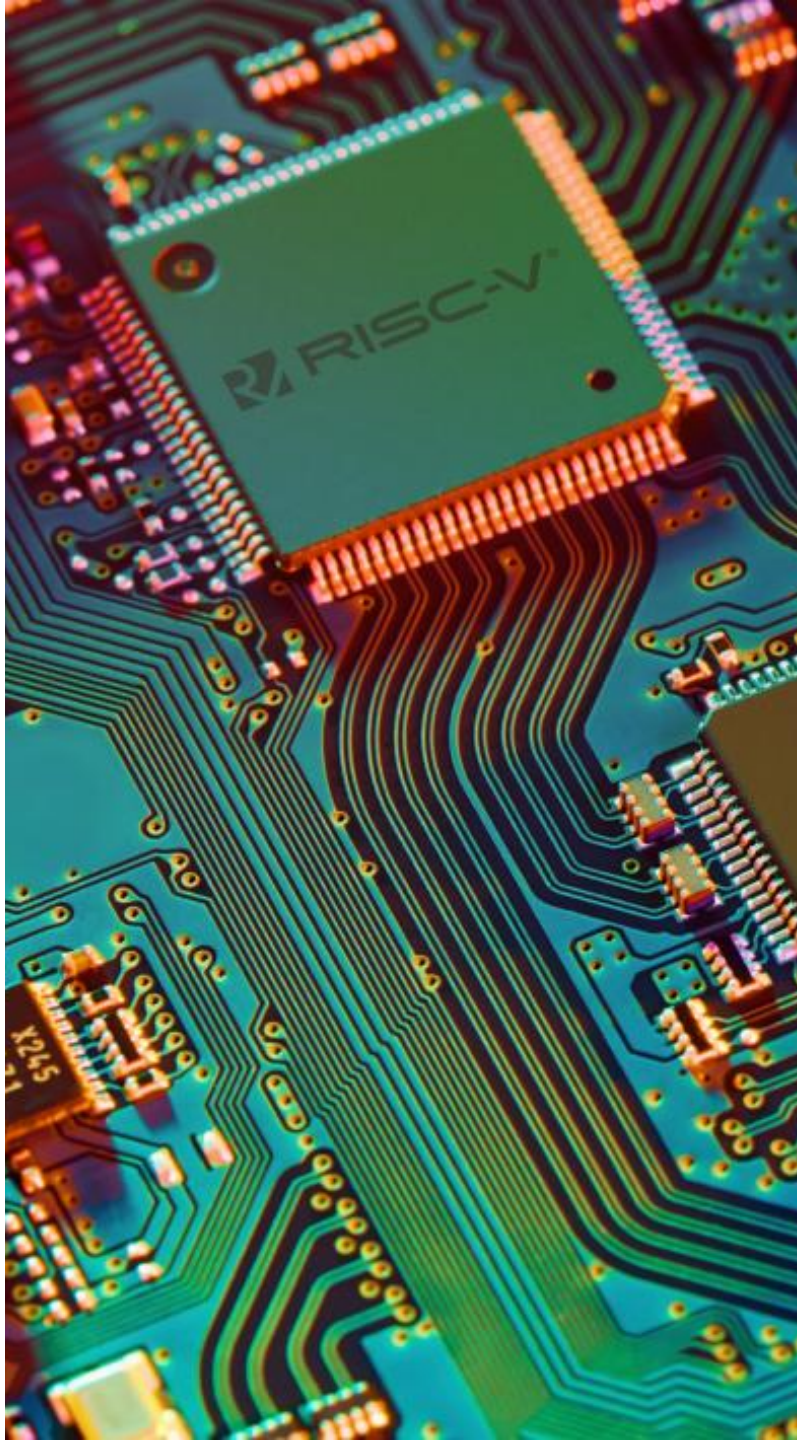




Симуляция. Инструменты для задания

Бесплатный симулятор Icarus Verilog, который хотя и не поддерживает весь SystemVerilog, но поддерживает Verilog 2005 с некоторыми элементами SystemVerilog, достаточных для решения наших задач.

GTKWave, программой для работы с временными диаграммами. Для первых десяти задач GTKWave нам не понадобится, но его стоит установить вместе с Icarus Verilog на будущее.

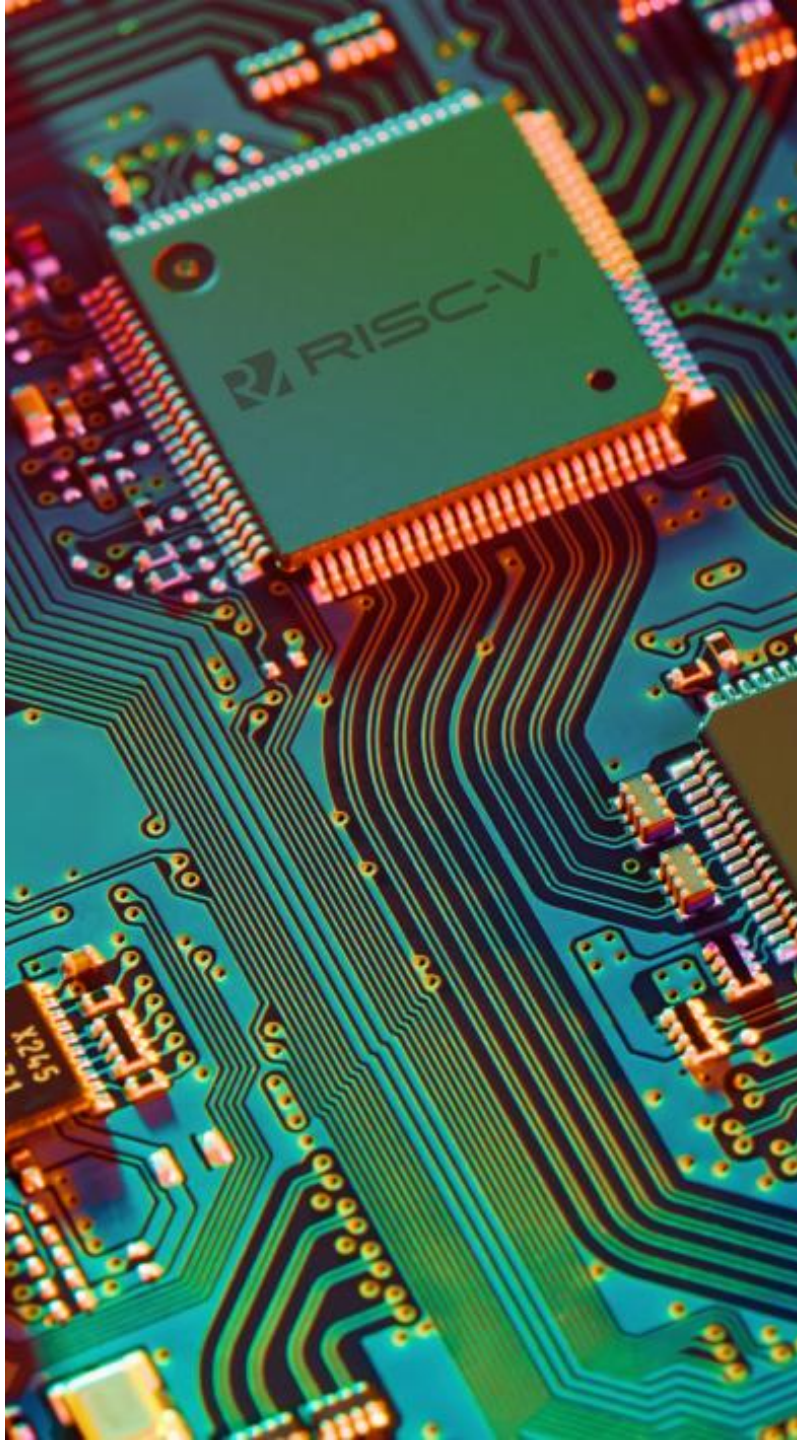


Icarus Verilog

Icarus Verilog – среда для поведенческого моделирования цифровых схем, описанных на языке Verilog;

Icarus Verilog – свободное (открытое) и кроссплатформенное ПО;

Icarus Verilog – инструмент командной строки (не имеет графического интерфейса пользователя).



Icarus Verilog. Установка

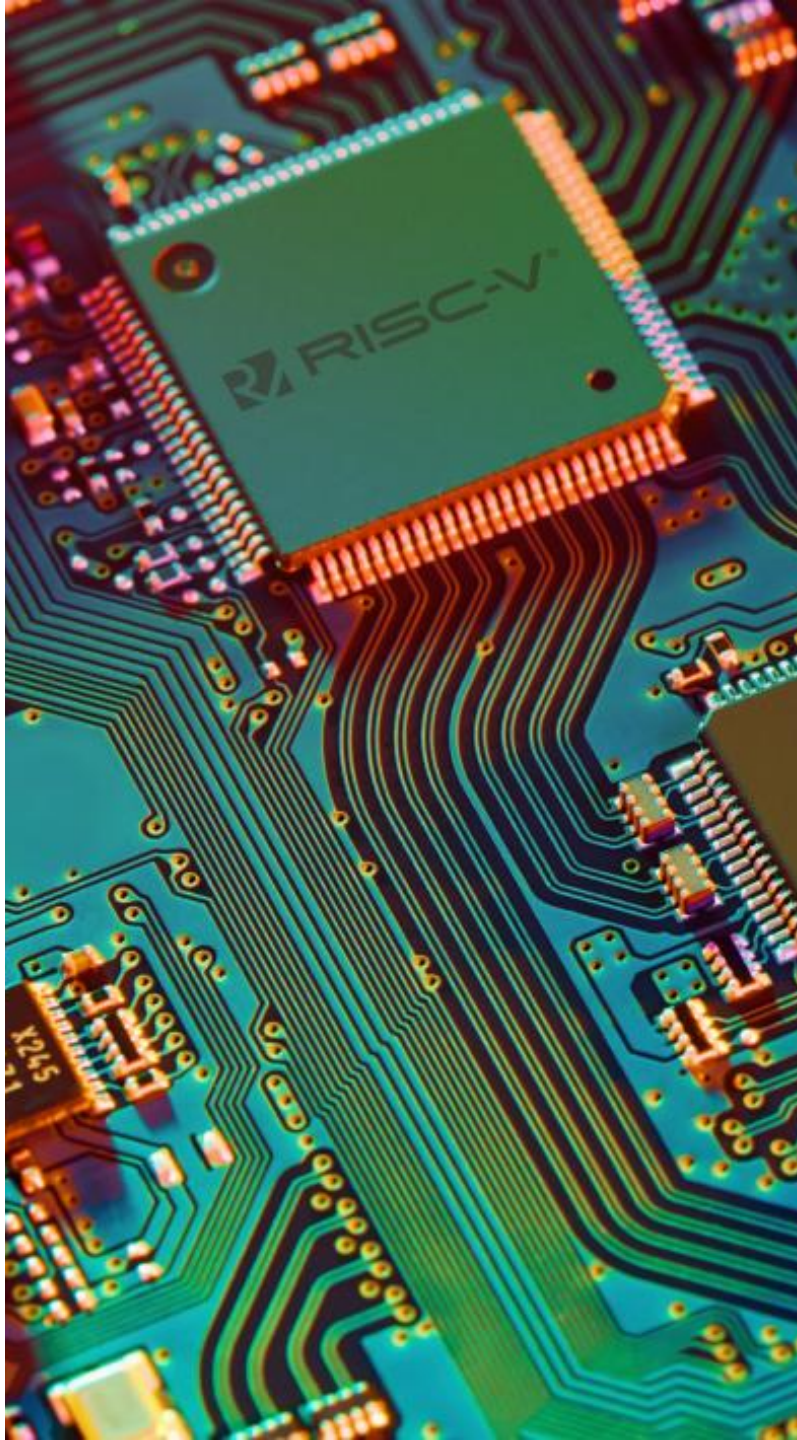
Под Linux Icarus Verilog и GTKWave ставится "sudo apt-get install verilog gtkwave".

Версия Icarus Verilog и GTKWave для Windows здесь.

Важно! При установке выставить флаг о добавлении Icarus в path Windows.

Если у вас под Windows что-то не работает, проверьте, какие директории стоят у вас в path.

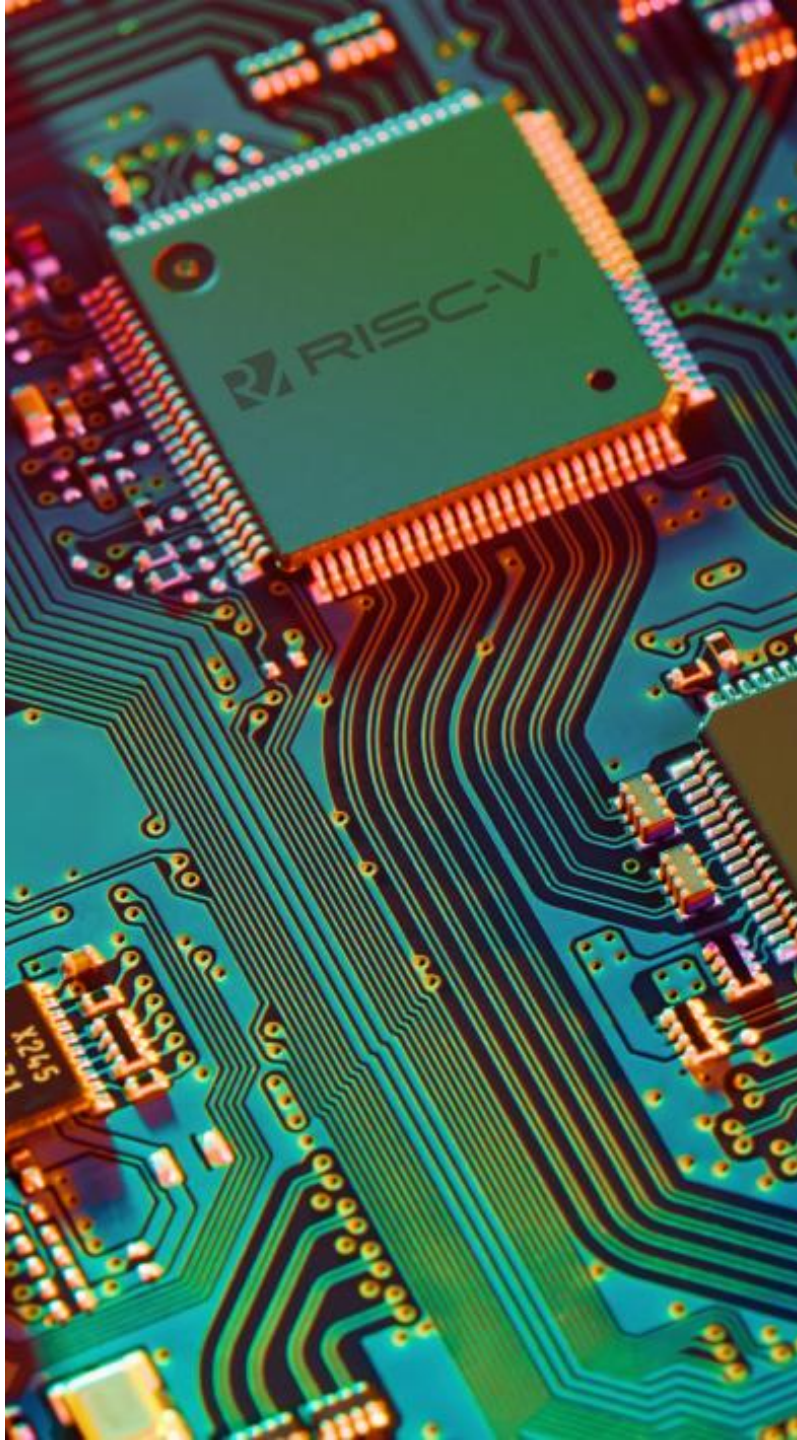
Установка в консоли с помощью программы brew: brew install icarus-verilog.



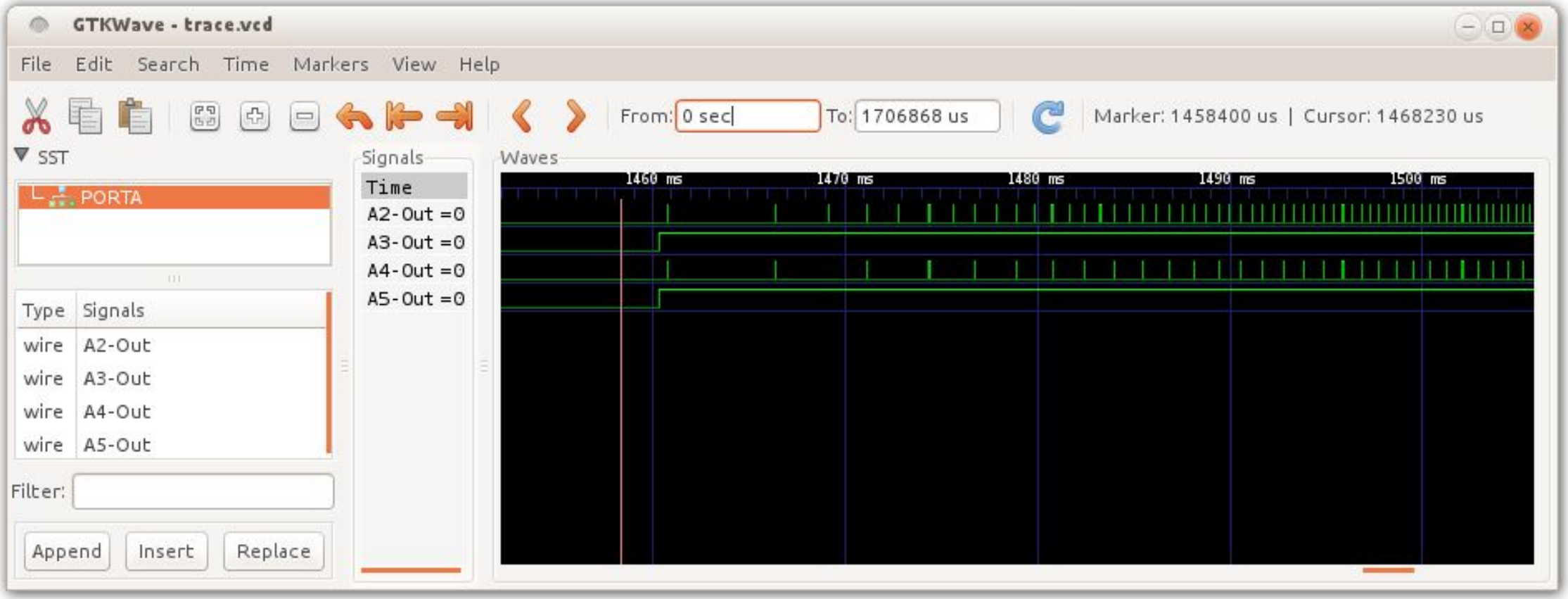
Инструменты для задания. Запуск упражнений.

В зависимости от операционной системы запускаете скрипт:

- run_all_using_iverilog_under_linux_or_macos_brew.sh
- run_all_using_iverilog_under_windows.bat



Симулятор. GTKwave.

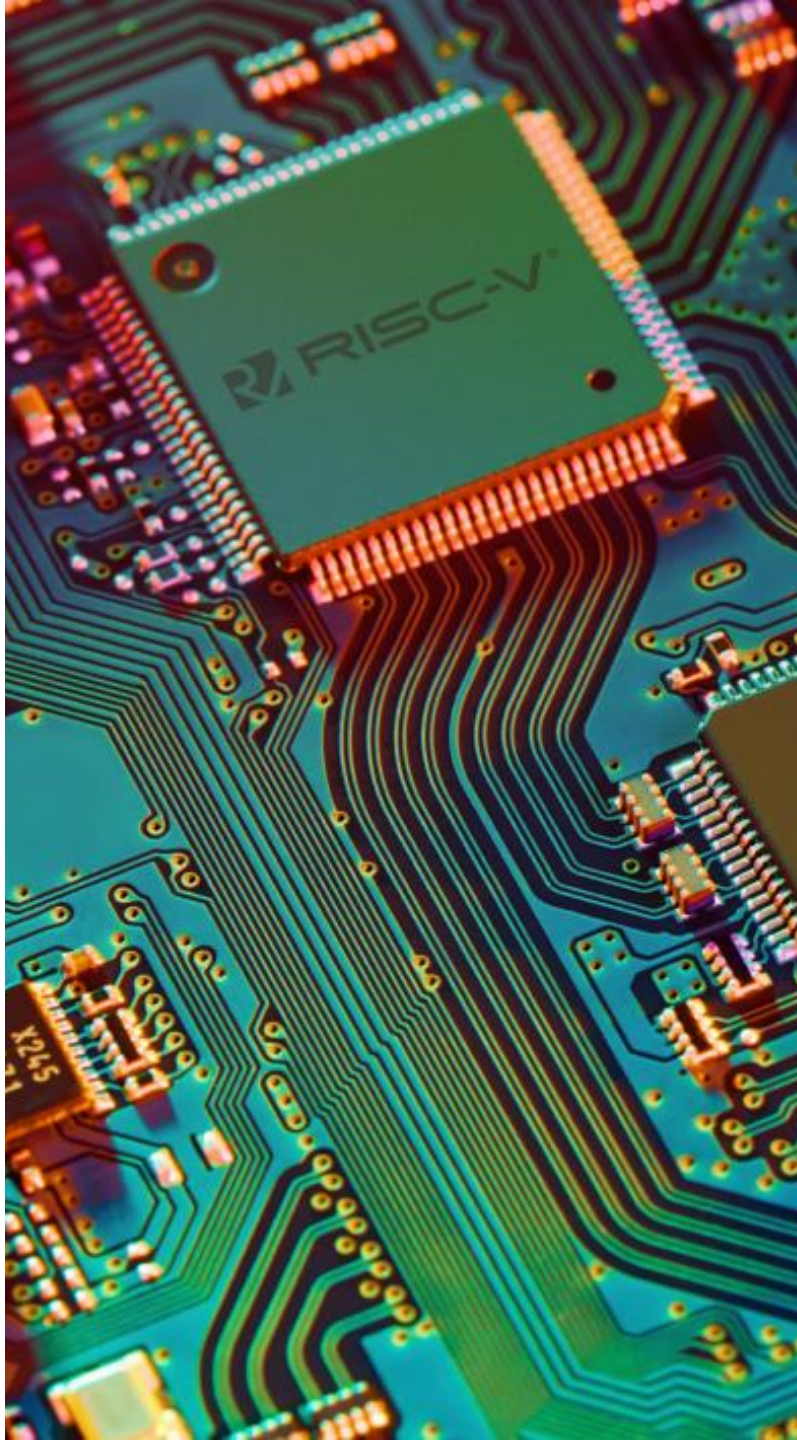


Среда тестирования

Среда тестирования (testbench) – это модуль на HDL, который используется для тестирования другого модуля, называемого тестируемое устройство (device under test, DUT). (Некоторые программы разработки называют тестируемый модуль unit under test, UUT.)

Среда тестирования содержит операторы для генерации значений, подаваемых на входы DUT и, в идеале, также и для проверки, что на выходе получаются правильные значения.

Наборы входных и желаемых выходных значений называются тестовыми векторами.



Пример testbench

Директива timescale устанавливает квант времени моделирования;

Шапка модуля testbench не содержит сигналов;

Тестируемый модуль подключается внутрь testbench;

Символ «#» позволяет задержать сигнал на некоторое количество времени, указанное в директиве timescale.

```
1  `timescale 1ns/1ps
2
3  module testbench ();
4
5    reg clk;
6    reg rst;
7    reg enable;
8    wire [7:0] counter;
9
10
11   counter UUT
12   (
13   .clk     (clk),
14   .en      (enable),
15   .rst     (rst),
16   .counter (counter)
17   );
18
19
20   always
21     #5 clk = ~clk;
22   end
23
```

Пример testbench

Директива timescale устанавливает квант времени моделирования;

Шапка модуля testbench не содержит сигналов;

Тестируемый модуль подключается внутрь testbench;

Символ «#» позволяет задержать сигнал на некоторое количество времени, указанное в директиве timescale.

```
1  `timescale 1ns/1ps
2
3  module testbench ();
4
5    reg clk;
6    reg rst;
7    reg enable;
8    wire [7:0] counter;
9
10
11   counter UUT
12   (
13   .clk     (clk),
14   .en      (enable),
15   .rst     (rst),
16   .counter (counter)
17   );
18
19
20   always
21     #5 clk = ~clk;
22   end
23
```

Пример testbench

Директива timescale устанавливает квант времени моделирования;

Шапка модуля testbench не содержит сигналов;

Тестируемый модуль подключается внутрь testbench;

Символ «#» позволяет задержать сигнал на некоторое количество времени, указанное в директиве timescale.

```
1  `timescale 1ns/1ps
2
3  module testbench ();
4
5    reg clk;
6    reg rst;
7    reg enable;
8    wire [7:0] counter;
9
10
11   counter UUT
12   (
13   .clk     (clk),
14   .en      (enable),
15   .rst     (rst),
16   .counter (counter)
17   );
18
19
20   always
21     #5 clk = ~clk;
22   end
23
```

Пример testbench

Директива timescale устанавливает квант времени моделирования;

Шапка модуля testbench не содержит сигналов;

Тестируемый модуль подключается внутрь testbench;

Символ «#» позволяет задержать сигнал на некоторое количество времени, указанное в директиве timescale.

```
1  `timescale 1ns/1ps
2
3  module testbench ();
4
5    reg clk;
6    reg rst;
7    reg enable;
8    wire [7:0] counter;
9
10
11   counter UUT
12   (
13   .clk     (clk),
14   .en      (enable),
15   .rst     (rst),
16   .counter (counter)
17   );
18
19
20   always
21     #5 clk = ~clk;
22   end
23
```

Пример testbench

Директива timescale устанавливает квант времени моделирования;

Шапка модуля testbench не содержит сигналов;

Тестируемый модуль подключается внутрь testbench;

Символ «#» позволяет задержать сигнал на некоторое количество времени, указанное в директиве timescale.

```
1   `timescale 1ns/1ps
2
3   module testbench ();
4
5     reg clk;
6     reg rst;
7     reg enable;
8     wire [7:0] counter;
9
10
11    counter UUT
12    (
13    .clk     (clk),
14    .en      (enable),
15    .rst     (rst),
16    .counter (counter)
17    );
18
19
20    always
21      #5 clk = ~clk;
22    end
23
```

Пример testbench

С помощью блока «initial» и «#» можно задавать сигналам типа reg меняющиеся во времени значения;

Конструкция «$finish;» указывает симулятору на то, что пора завершить моделирование.

```
25  initial begin
26      // Initial values
27      clk    = 0;
28      rst    = 0;
29      enable = 0;
30
31      // Reset
32      #20
33      rst = 1;
34      #20
35      rst = 0;
36      #20
37
38      // Set enable
39      enable = 1;
40      #10
41      enable = 0;
42
43      #30
44      enable = 1;
45      #10
46      enable = 0;
47      #40
48      enable = 1;
49      #10
50      enable = 0;
51
52      // Reset again
53      #20
54      rst = 1;
55      #20
56      rst = 0;
57      #20
58
59      $finish;
60  end
```

Пример testbench

```
63  initial
64    $monitor($stime,, counter);
65
66
67  initial begin
68    $dumpfile("out.vcd");
69    $dumpvars;
70  end
71
72  endmodule
```

Для вывода результатов моделирования «наружу» предлагается 2 способа:

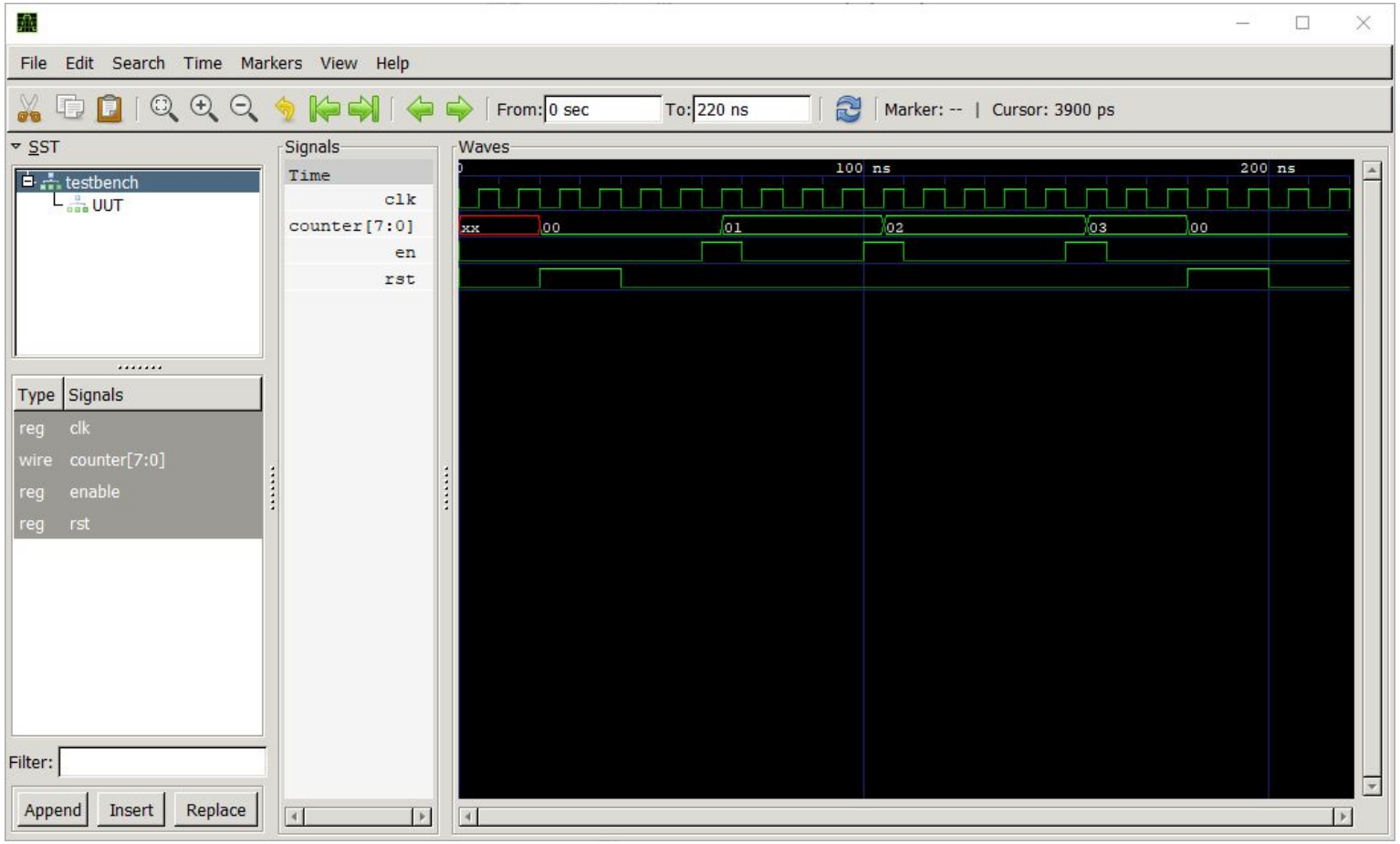
Конструкция «$monitor();» позволяет выводить в текстовую консоль сообщение при изменении указанных в ней сигналов;

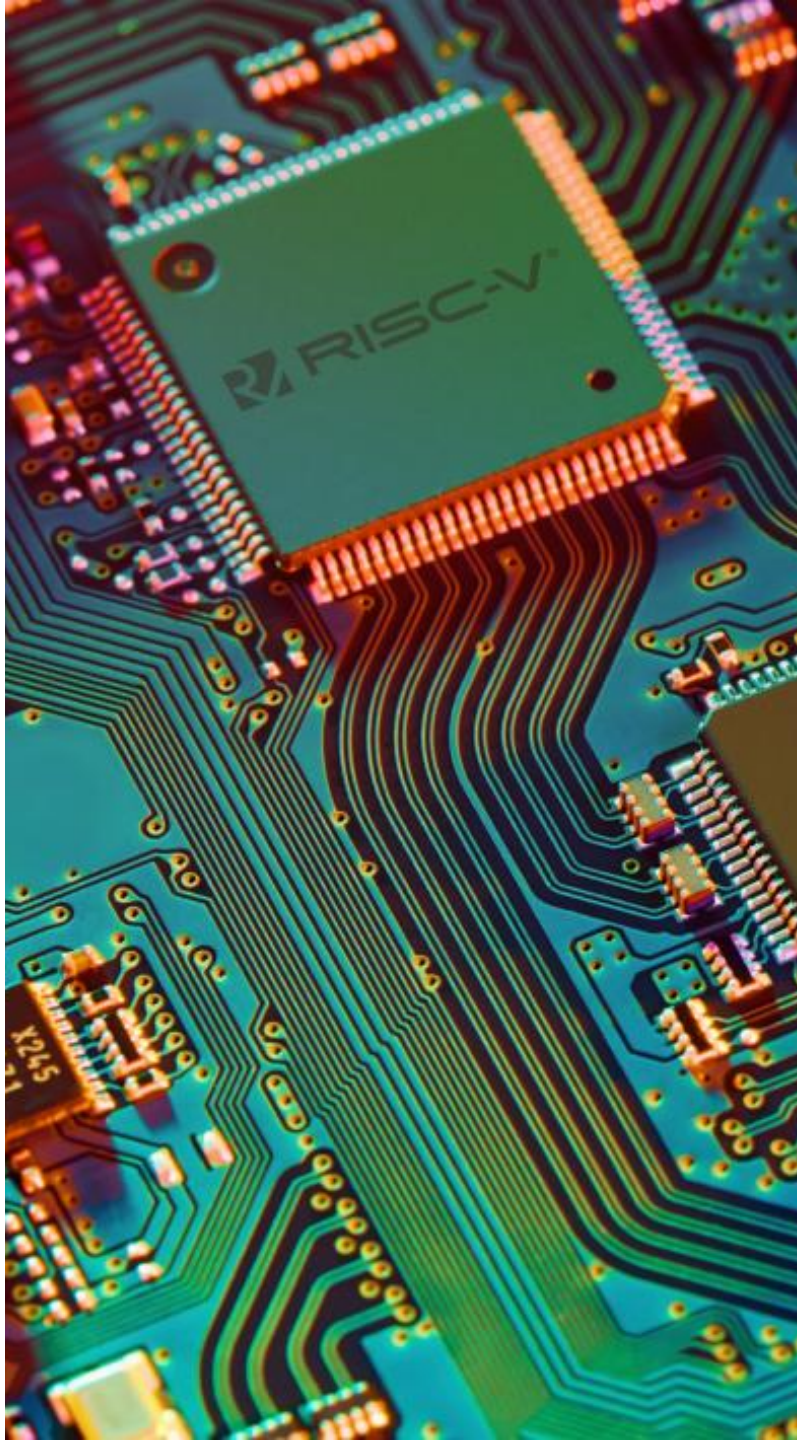
Конструкции «$dumpfile();» и «$dumpvars();» позволяют записывать лог моделирования в файл .vcd.

```
PS C:\Users\Conf\Seafile\Teaching\ПЦУ\iverilog> vvp test
VCD info: dumpfile out.vcd opened for output.
         0   x
        20   0
        65   1
       105   2
       155   3
       180   0
```

Запуск Работа с файлами .vcd

Для просмотра файлов .vcd существует открытая утилита GTKWave



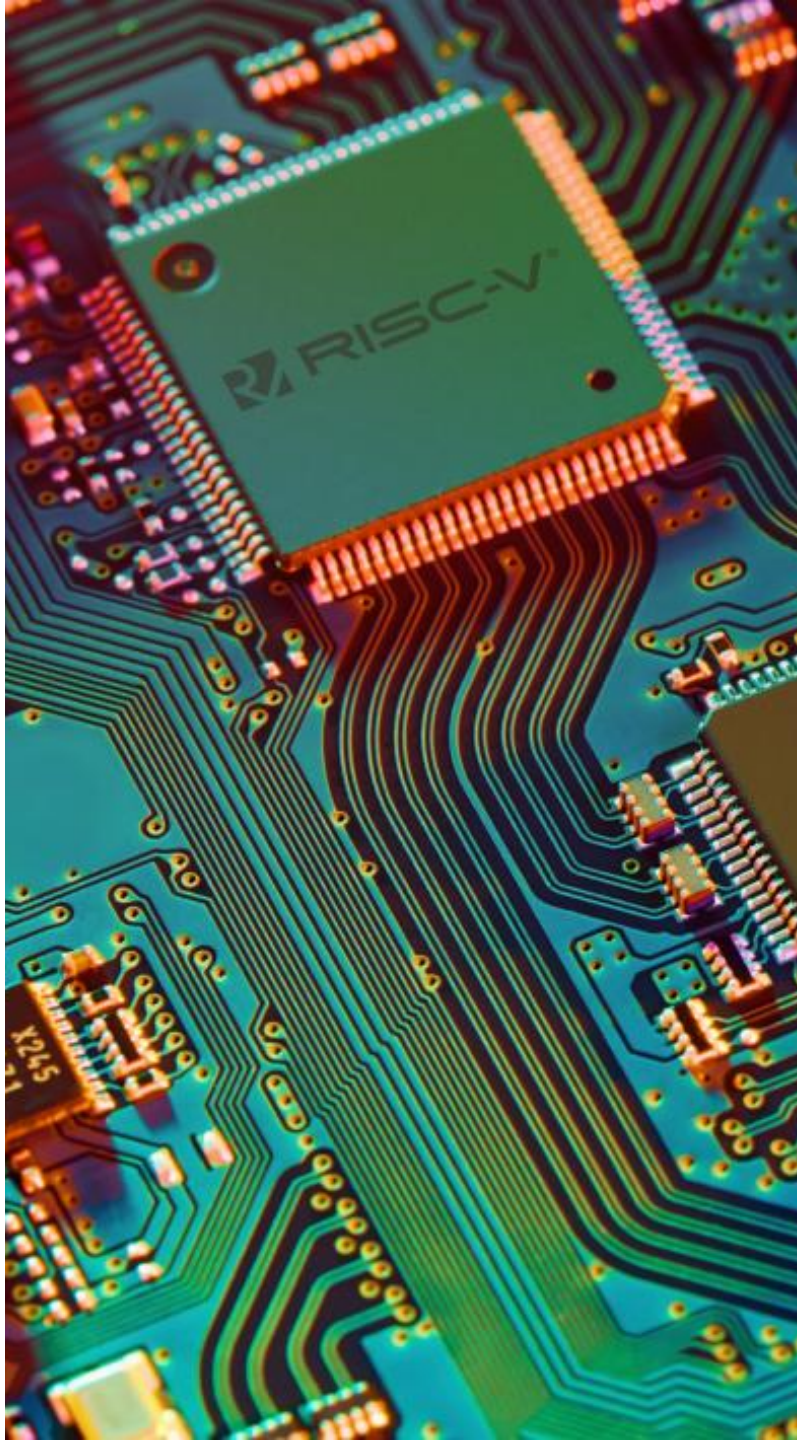


Упражнения. Testbench

```
module testbench();

  logic a, b, o;
  int i, j;

  xor_gate_using_mux inst (a, b, o);

  initial
    begin
      for (i = 0; i <= 1; i++)
      for (j = 0; j <= 1; j++)
      begin
        a = i;
        b = j;

        # 1;
        $display ("TEST %b ^ %b = %b", a, b, o);

        if (o !== (a ^ b))
          begin
            $display ("%s FAIL: %h EXPECTED", `__FILE__, a ^ b);
            $finish;
          end
      end

      $display ("%s PASS", `__FILE__);
      $finish;
    end

endmodule
```

Упражнения. Скрипт

```
#!/bin/sh

rm -rf log.txt

for f in *.v
do
        iverilog -g2005-sv $f >> log.txt 2>&1  \
    && vvp a.out              >> log.txt 2>&1

    # gtkwave dump.vcd
done

rm -rf a.out

grep -e PASS -e FAIL -e error log.txt
```



Упражнения. Лог

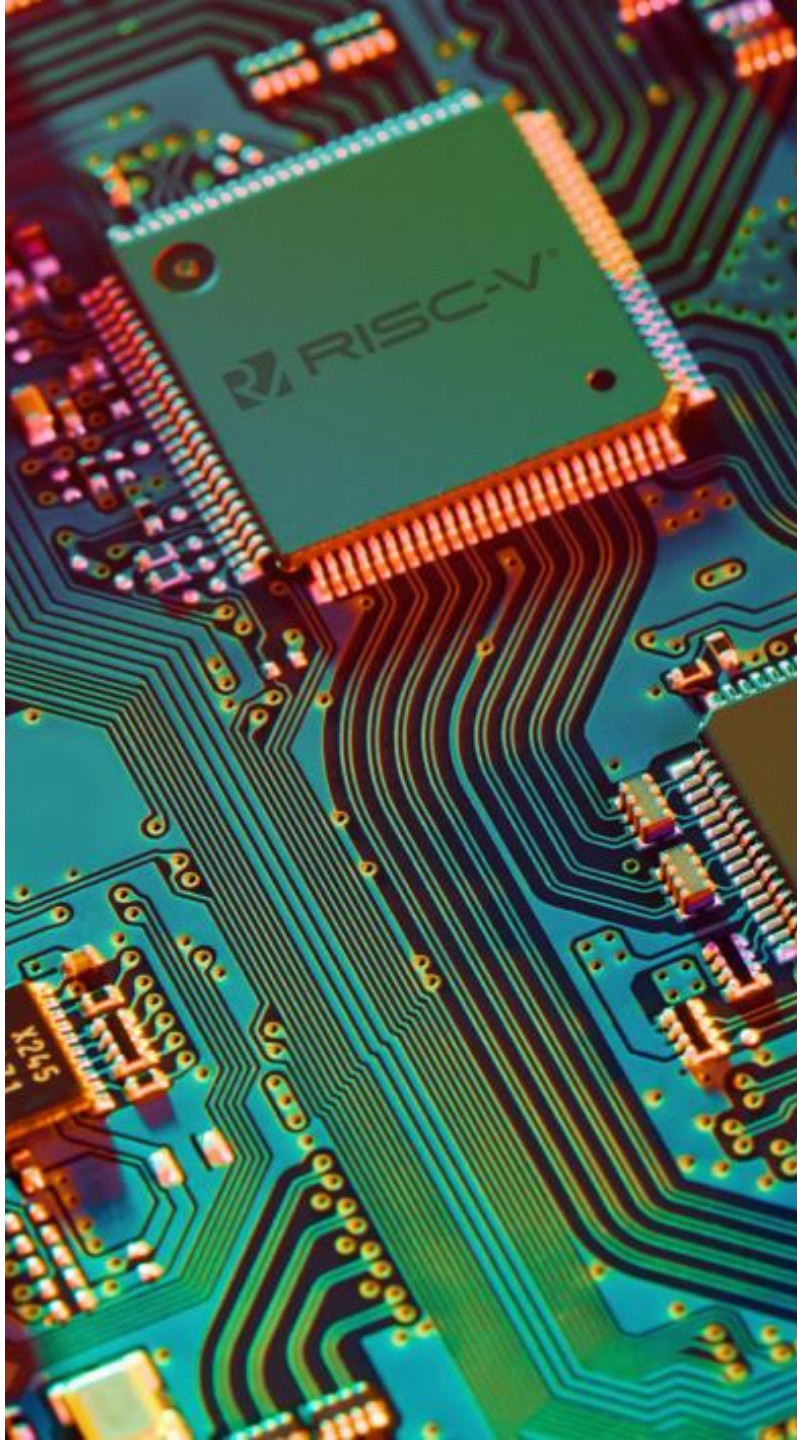
```
p005_mux_gates.v PASS
p005_mux_gates.v:127: $finish called at 8 (1s)
TEST d { a b c d } sel 0 y a
TEST d { a b c d } sel 1 y b
TEST d { a b c d } sel 2 y c
TEST d { a b c d } sel 3 y d
TEST d { 7 a 3 x } sel 0 y 7
TEST d { 7 a 3 x } sel 1 y a
TEST d { 7 a 3 x } sel 2 y 3
TEST d { 7 a 3 x } sel 3 y x
p006_mux_2n_using_muxes_n.v PASS
p006_mux_2n_using_muxes_n.v:103: $finish called at 8 (1s)
TEST d { a b c d } sel 0 y a
TEST d { a b c d } sel 1 y b
TEST d { a b c d } sel 2 y c
TEST d { a b c d } sel 3 y d
TEST d { 7 a 3 x } sel 0 y 7
TEST d { 7 a 3 x } sel 1 y a
TEST d { 7 a 3 x } sel 2 y 3
TEST d { 7 a 3 x } sel 3 y x
p007_mux_using_narrow_data_muxes.v PASS
p007_mux_using_narrow_data_muxes.v:97: $finish called at 8 (1s)
```

```
TEST ~ 0 = 1
TEST ~ 1 = 0
p008_not_gate_using_mux.v PASS
p008_not_gate_using_mux.v:60: $finish called at 2 (1s)
TEST 0 & 0 = 0
TEST 0 & 1 = 0
TEST 1 & 0 = 0
TEST 1 & 1 = 1
p009_and_gate_using_mux.v PASS
p009_and_gate_using_mux.v:63: $finish called at 4 (1s)
TEST 0 | 0 = 0
TEST 0 | 1 = 1
TEST 1 | 0 = 1
TEST 1 | 1 = 1
p010_or_gate_using_mux.v PASS
p010_or_gate_using_mux.v:64: $finish called at 4 (1s)
TEST 0 ^ 0 = 0
TEST 0 ^ 1 = 1
TEST 1 ^ 0 = 1
TEST 1 ^ 1 = 0
p011_xor_gate_using_mux.v PASS
p011_xor_gate_using_mux.v:72: $finish called at 4 (1s)
```

Упражнения

Нужно выполнить 17 упражнений из директории day_1

Все упражнения проверяются автоматически. Смотрите лог симуляции.

В зависимости от операционной системы запускаете скрипт:

- run_all_using_iverilog_under_linux_or_macos_brew.sh
- run_all_using_iverilog_under_windows.bat