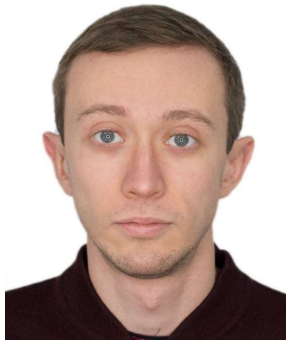
Введение в проектирование ПЛИС

Авторы курса:

Буссе Александр …
https://t.me/alex_busse



Покровский Андрей Дмитриевич
https://t.me/the_november_sun

Полезные ссылки:
Телеграмм чат: https://t.me/+b2MxHuWJN441MDky
GitHub: https://github.com/BusseAA/FPGA_introduction_course/wiki

План занятий

Даты планируемых занятий:

1. 04.09.2024
2. 18.09.2024
3. 02.10.2024
4. 16.10.2024
5. 30.10.2024 (выдача финального ДЗ)
6. 13.11.2024
7. 27.11.2024
8. 11.12.2024 (сдача всех ДЗ)

Оценка за курс:

- Всего будет три обычных ДЗ и один домашний проект
    1. 20 баллов
    2. 25 баллов
    3. 25 баллов
    4. 30 баллов (проект)
- Сдача ДЗ
    - Можно на следующее занятие или через одно
    - Сдача на третьем занятии после выдачи ДЗ **минус 10** баллов
    - Сдача на четвертом занятии после выдачи ДЗ **минус еще 10** баллов
    - Дальше без штрафов

CPU (Central Processing Unit)

Programming language

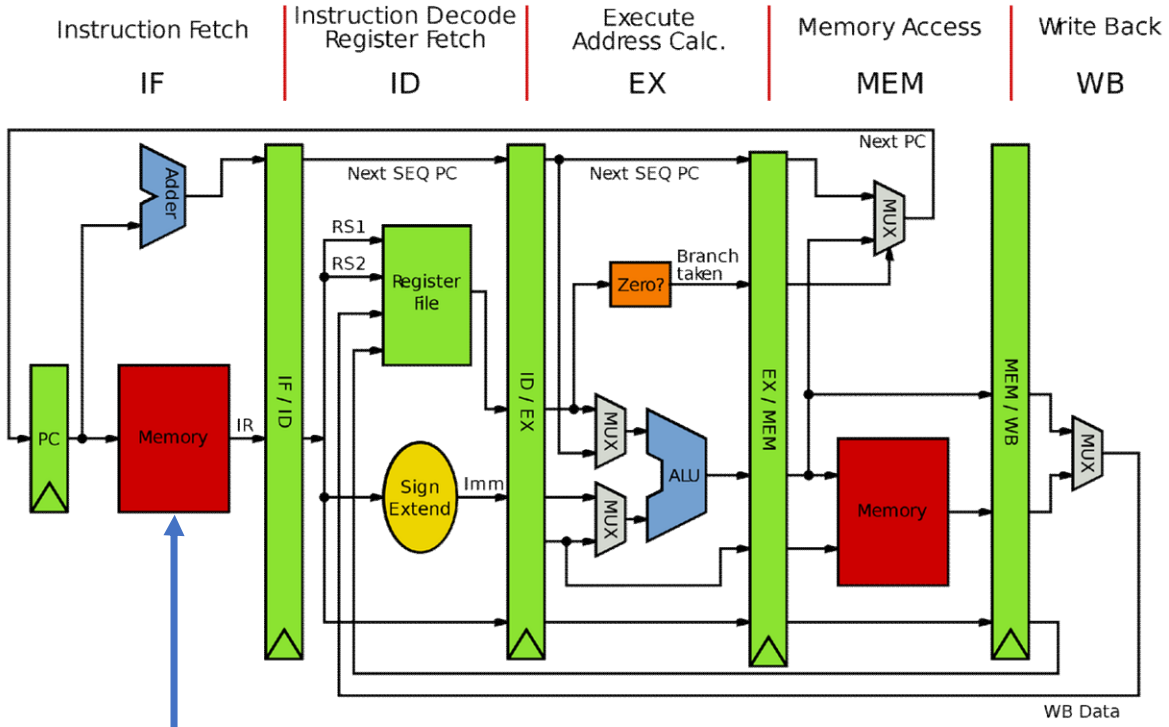
```
# include <iostream>

int main() {

     int a[10] = { .... };
     int b[10] = { .... };
     int c[10];

     for (int i = 0; i < 10; i++) {
          c[i] = a[i] + b[i];
     }
     return 0;
}
```

Compiler

Visual Studio

GCC

Opcodes + data



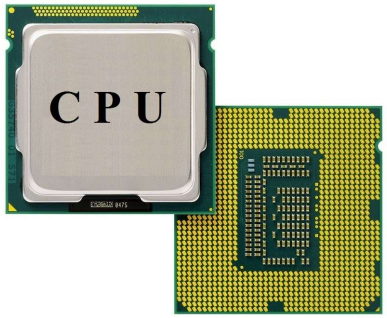
CPU (Central Processing Unit)

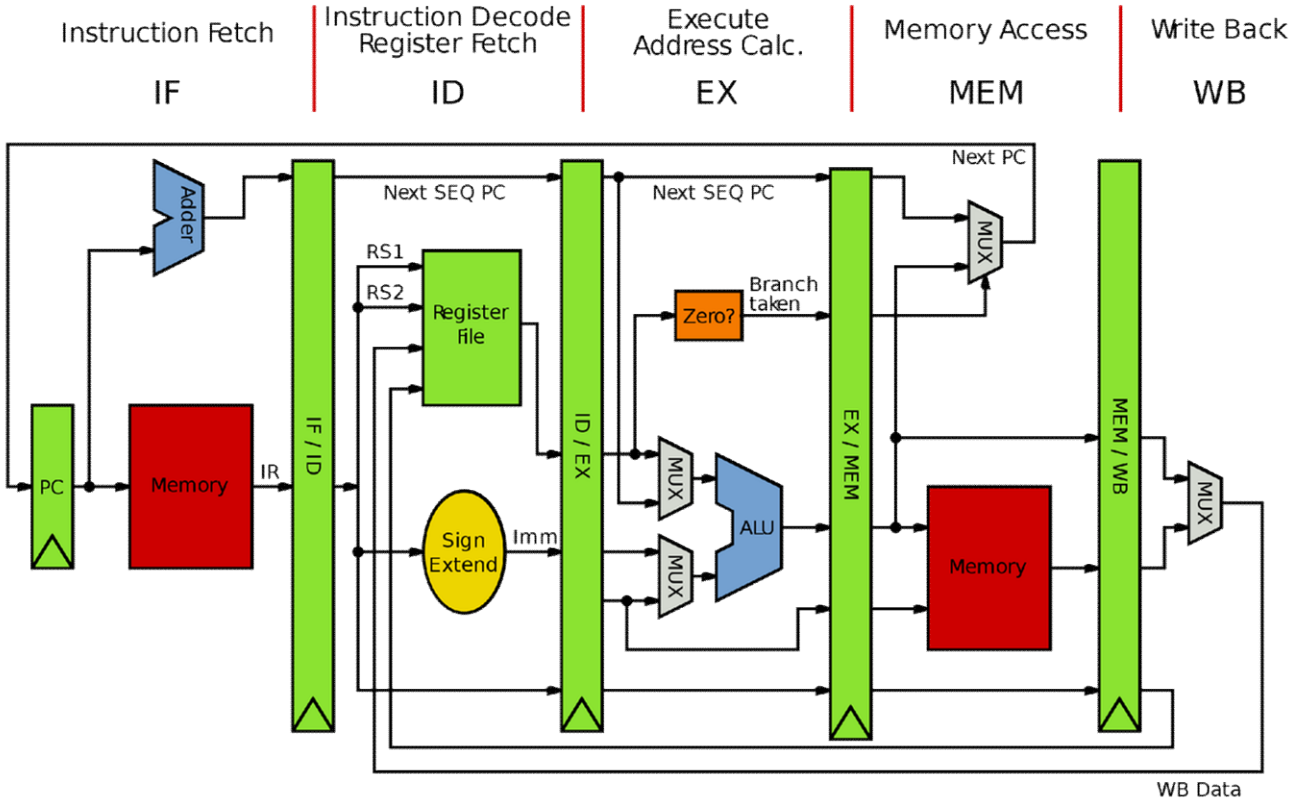
Плюсы:

- Универсальность
- Доступность
- Простота программирования

Минусы:

- Невысокая эффективность для специфичных задач

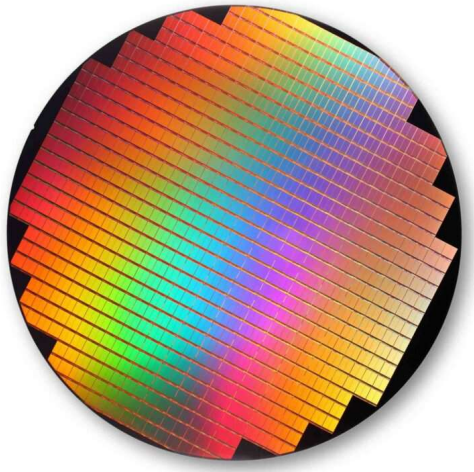




ASIC (application-specific integrated circuit)

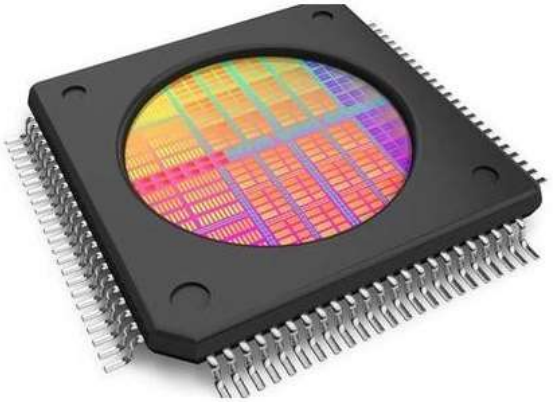
Примеры:

- Микросхема цифровой обработки звукового сигнала в мобильном телефоне
- Оборудование для майнинга криптовалют



Плюсы:

- Максимальная производительность и энергоэффективность для выполнения конкретной задачи



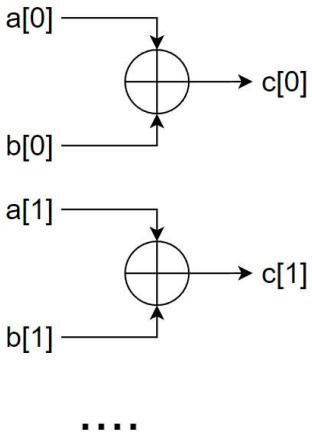
Минусы:

- Высокая стоимость выпуска больших партий микросхем
- Узкая специализация готовой микросхемы
- Большое время на разработку
- Сложность разработки

ASIC (application-specific integrated circuit)

HDL (Hardware Description Language)
(VHDL, AHDL, **Verilog**)

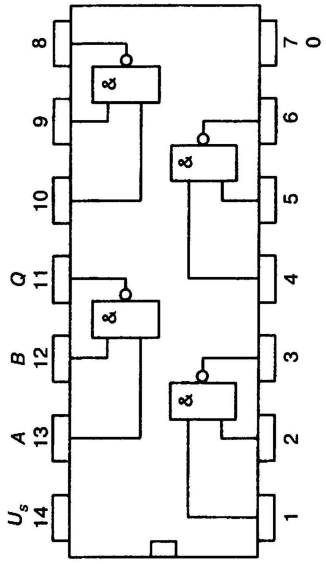
```
module AddVectors (
    a,
    b,
    c
);

    parameter N = 10;
    parameter W_DATA = 32;

    input a[N*W_DATA-1: 0];
    input b[N*W_DATA-1: 0];
    output c[N*W_DATA-1: 0];

    genvar i;
    generate
        for (i=0; i<N; i=i+1) begin
            assign c[(i+1)*W_DATA-1-: W_DATA] = \
                   a[(i+1)*W_DATA-1-: W_DATA] + \
                   b[(i+1)*W_DATA-1-: W_DATA];
        end
    endgenerate
```

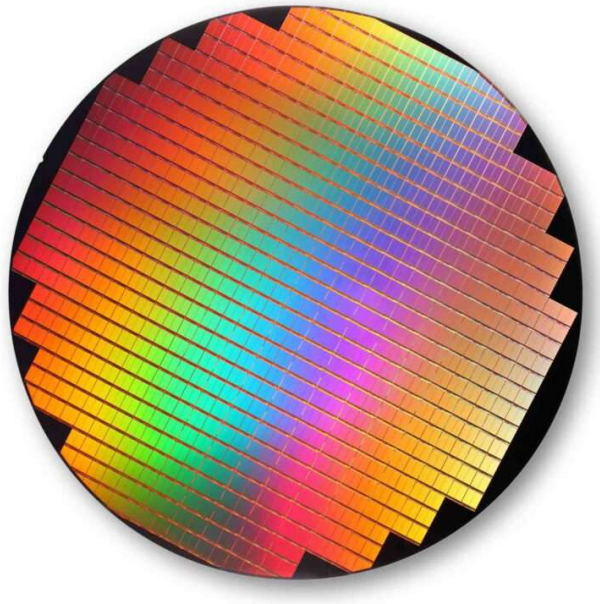


Design Compiler + elements libraries

SYNOPSYS® + tsmc

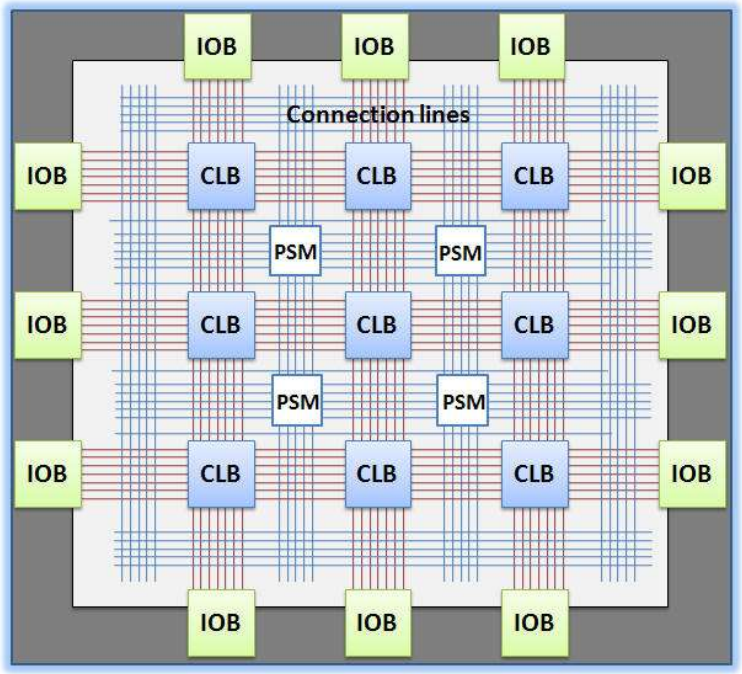
Elaborate
Map + optimize
Place and route
.....





FPGA (field-programmable gate array)

Внутреннее строение ПЛИС



IOB
Input Output Block

CLB
Configurable
Logic Block

PSM
Programable
Switch Matrix

Connection lines
Single, Long
Double, Direct

Плюсы:

- Есть возможность поменять схему под конкретную задачу (возможность реконфигурации)

Минусы:

- Низкая энергоэффективность по сравнению с ASIC
- Высокая стоимость
- Долгая загрузка

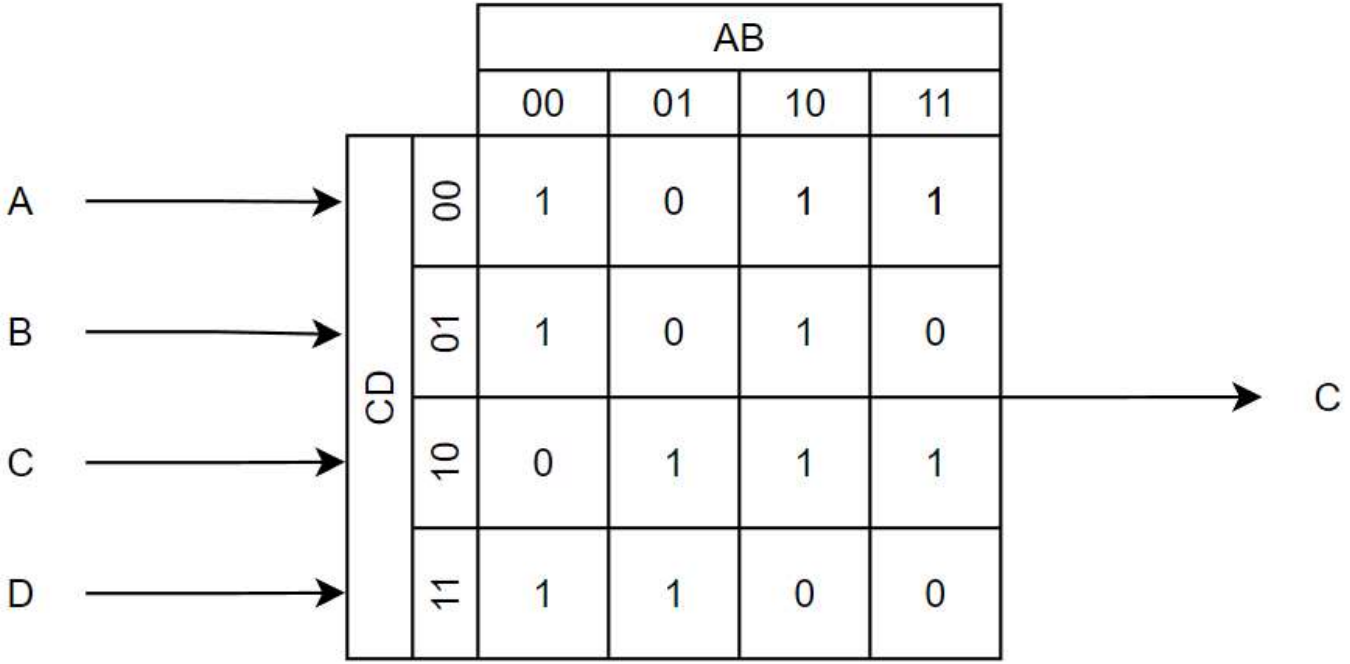
Применение:

- Прототипирование электронных схем
- Обработка сетевых пакетов
- Цифровая обработка сигналов (DSP)
- Computer Vision
- Нейронные сети

Комбинационные схемы (Combinational schemes).

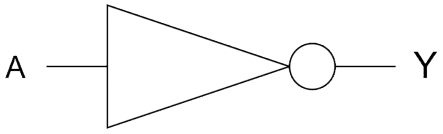
Логические схемы реализуют логические (Булевы) функции.

Основной принцип комбинационной схемы – выходное значение схемы зависит только от входных значений.



Примеры простейших комбинационных схем

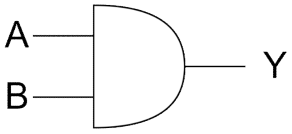
НЕ (NOT)



A	Y
0	1
1	0

Y=~A

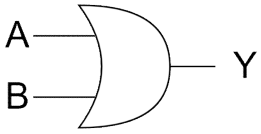
И (AND)



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Y=A&B

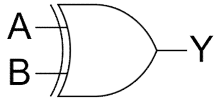
ИЛИ (OR)



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Y=A|B

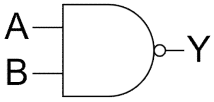
ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR)



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Y=A^B

И-НЕ (NAND)



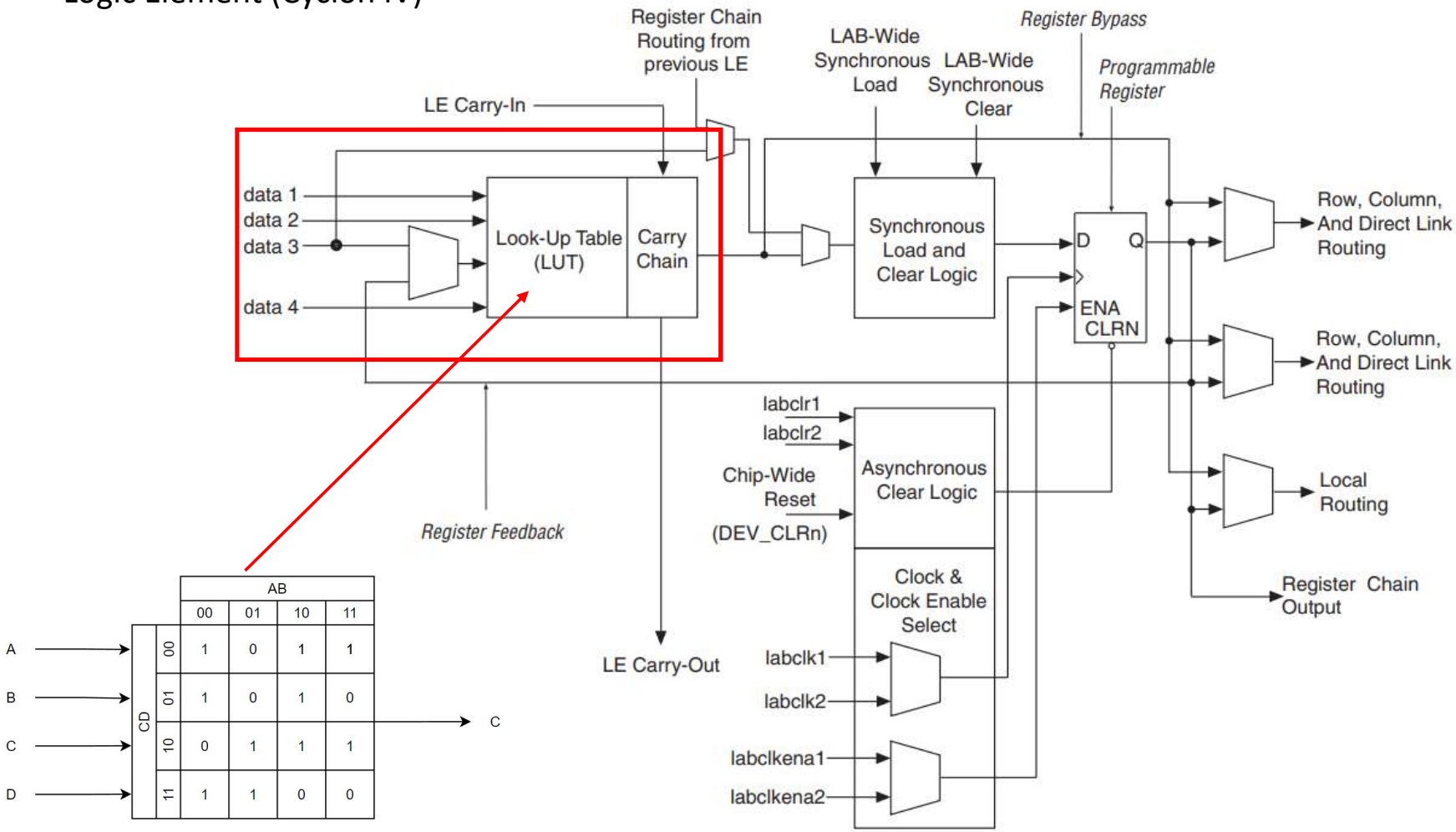
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Y=~(A&B)

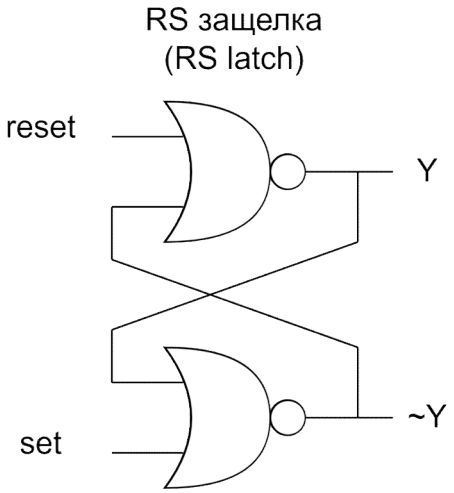
Обозначение элементов на Verilog

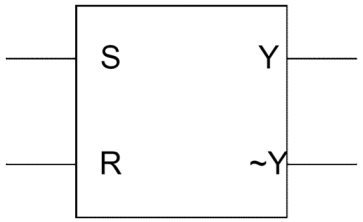
Реализация Комбинационной логики в FPGA

Logic Element (Cyclon IV)



Запоминающие устройства. RS защелка (RS latch).

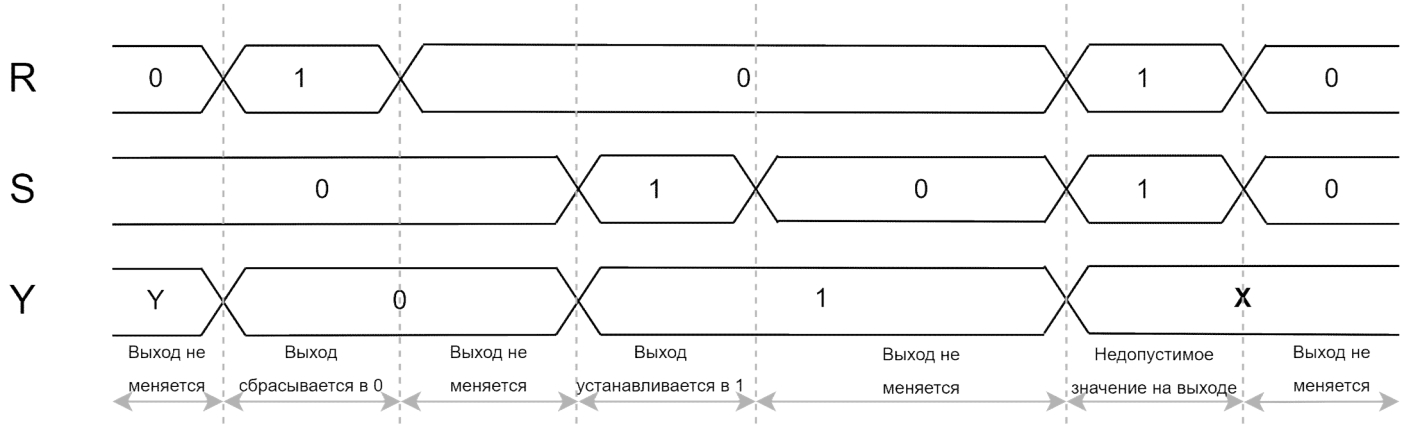




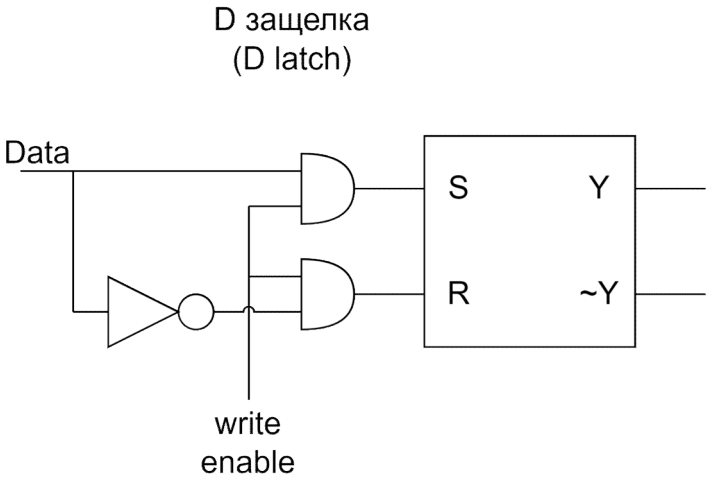
S	R	Yt-1	Yt
0	0	Y	Y
0	1	Y	0
1	0	Y	1
1	1	Y	**X**

Не допустимо на вход RS защелки одновременно подавать R и S равные единице. Иначе на его выходе будет неопределенное значение **X**. На самом деле, выход примет значение 0 или 1, но невозможно предугадать, какое из них.

Состояния на выходе RS защелки определяется уровнями входных сигналов и значением выходного сигнала.

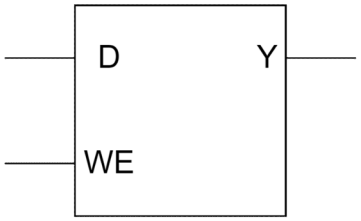


Запоминающие устройства.
D защелка (D latch).

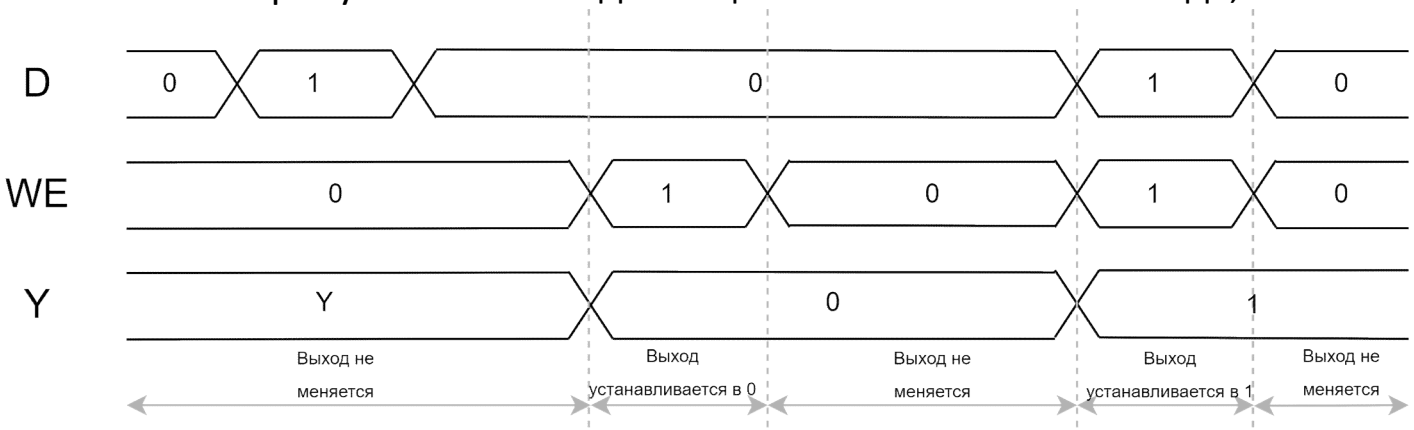


D	WE	Yt-1	Yt
D	0	Y	Y
0	1	Y	0
1	1	Y	1

D защелка содержит в себе RS защелку, но благодаря наличию сигнала разрешения записи WE на входе RS защелки одновременно не может быть единиц на обоих портах. Таким образом D защелка решает проблему RS защелки.

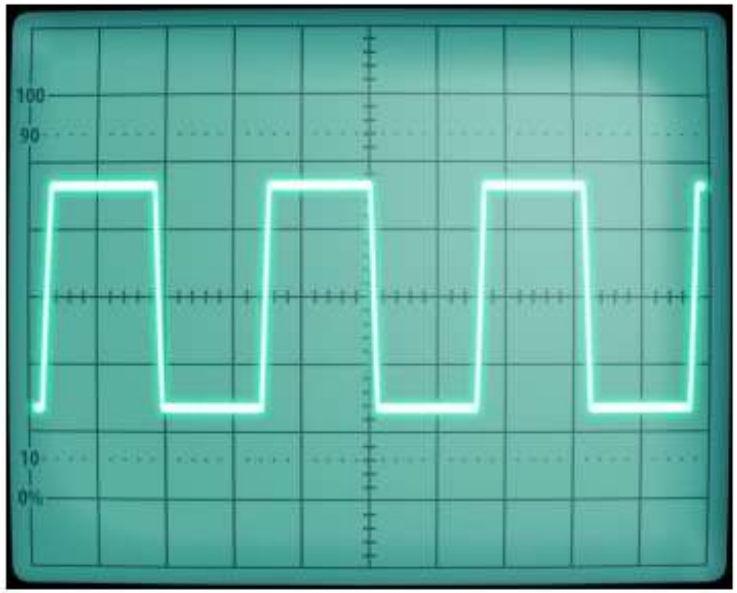


Сигнал WE пропускает на выход D защелки значение на его входе, если WE = 1.



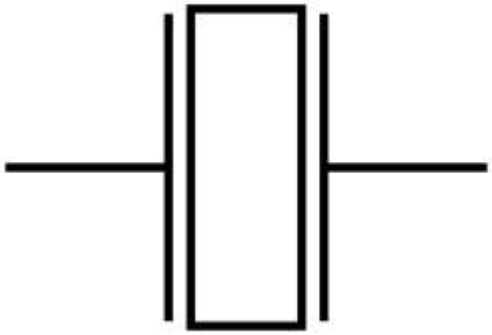
Тактирование

- Тактовый сигнал — clock
- Фронт — positive edge
- Спад — negative edge

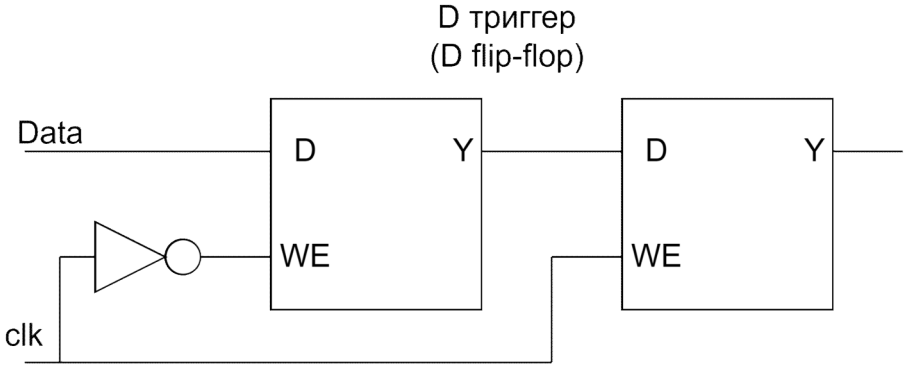


Тактирование

- Комбинационная логика не работает мгновенно, поэтому вычисления нужно *синхронизировать*
- Кварцевый генератор генерирует *тактовый сигнал* — clock
- Все комбинационные схемы должны успеть закончить вычисления за период тактового сигнала

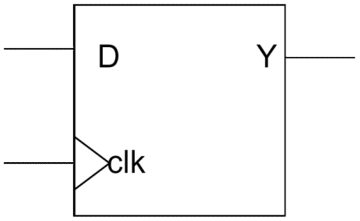


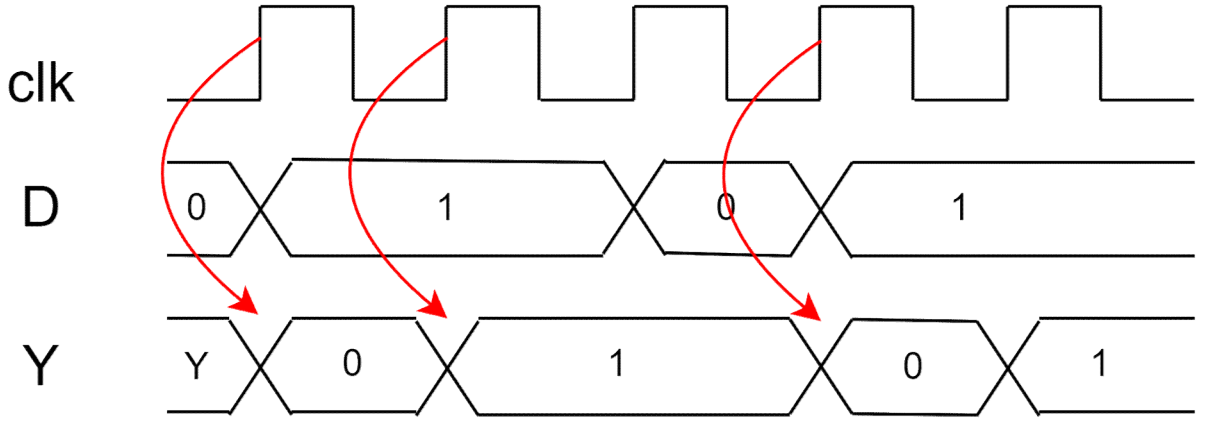
Запоминающие устройства.
D триггер (D flip-flop).



D	clk	Yt-1	Yt
D	0	Y	Y
D	1	Y	Y
D	0->1	Y	D
D	1->0	Y	Y

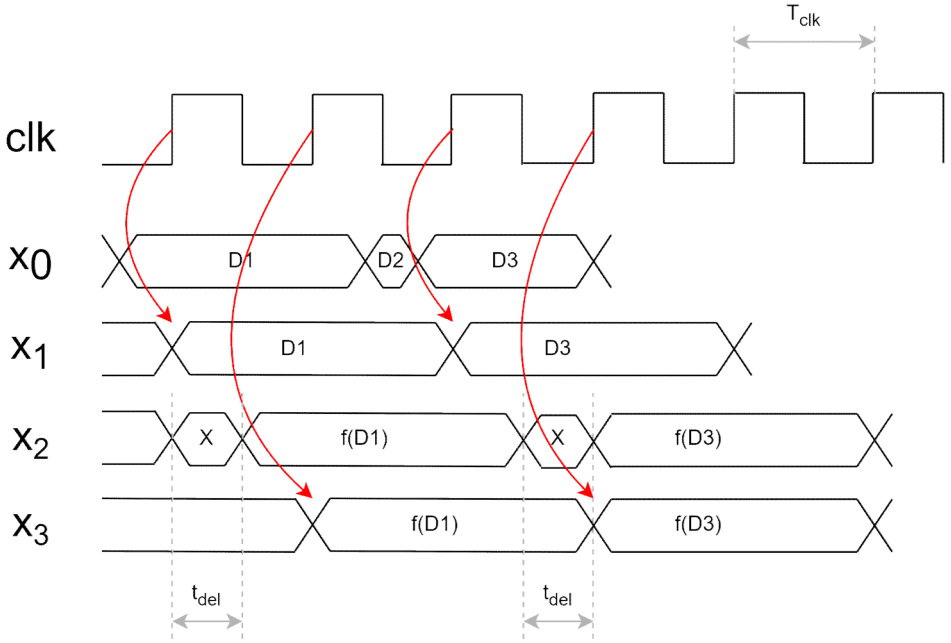
У D триггера изменение состояния происходит только при изменении тактового сигнала clk из 0 в 1 (при другой структуре возможно из 1 в 0).

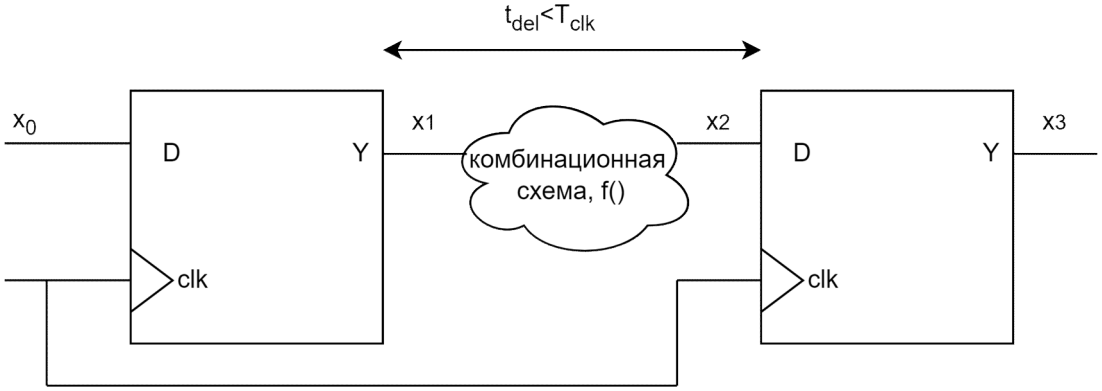




Синхронизация вычислений

- Комбинационная логика не работает мгновенно и каждый элемент имеет свою задержку по времени.
- Все вычисления в комбинационной схеме должны быть выполнены за период тактового сигнала, генерируемого кварцевым генератором.
- В противном случае в триггерах зафиксируются значения, соответствующие значениям промежуточных вычислений.





Verilog: literals

<size>’<base><number>

- **Size** – количество бит в представлении числа.
- **Base** – основание представления (b/B - binary), (d/D – decimal), (h/H – hex)
- **Number** – число в соответствии с выбранным основанием.

Примеры (12 с разными основаниями):

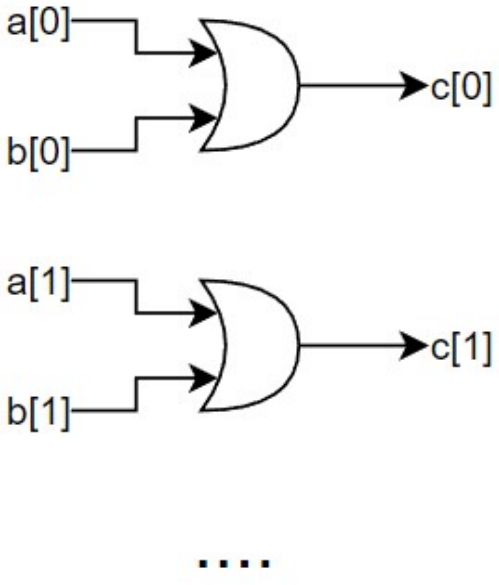
- 4’b1100
- 4’d12
- 4’hC

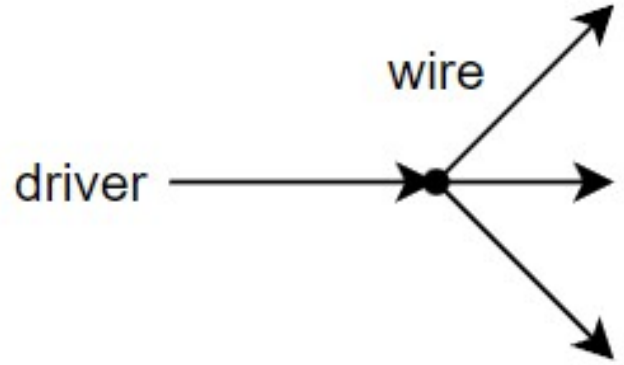
Так же сущевствует состояния X (value unknown/don’t care) Z (high impedance).

Verilog: комбинационная логика

Переменные типа wire используются для описания комбинационной логики.
Синтаксис: wire [older_bit_index: younger_bit_index] variable_name;
Для описания логики используется непрерывное присваивание assign (continuous assignment).

```
wire [15:0] a;
wire [15:0] b;
wire [15:0] c;
wire d;

assign c = b&a;
```

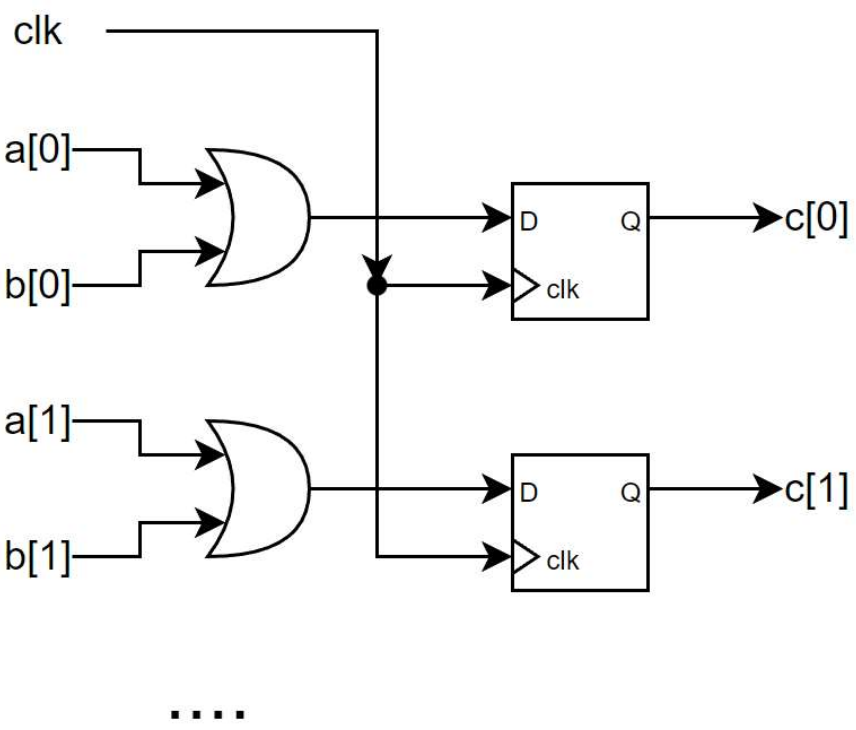




Verilog: Последовательностные схемы

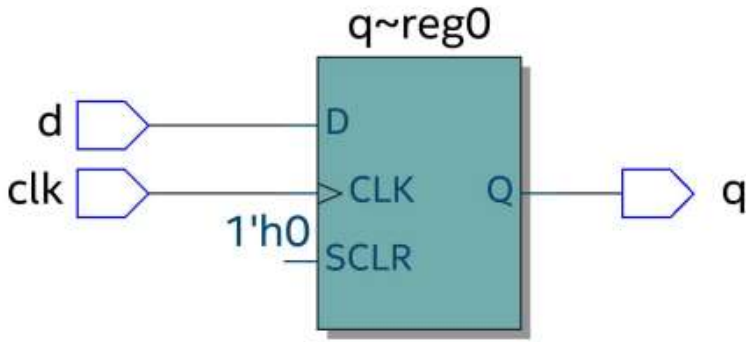
Переменные reg в большинстве случаев представляют собой регистр.
Последовательностная логика описывается с помощью блоков always и неблокирующих присваиваний <=

```
wire [15:0] a;
wire [15:0] b;
reg [15:0] c;

always @(posedge clk) begin
    c <= a&b;
end
```



D-триггер

```
reg q;

always @(posedge clk) begin
     q <= d;
end
```

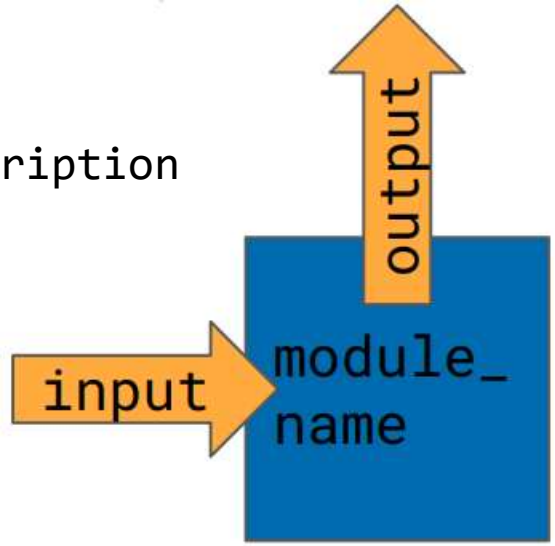


D	CLK	Q_n
0	↑	0
1	↑	1
*	0	Q_{n-1}
*	1	Q_{n-1}

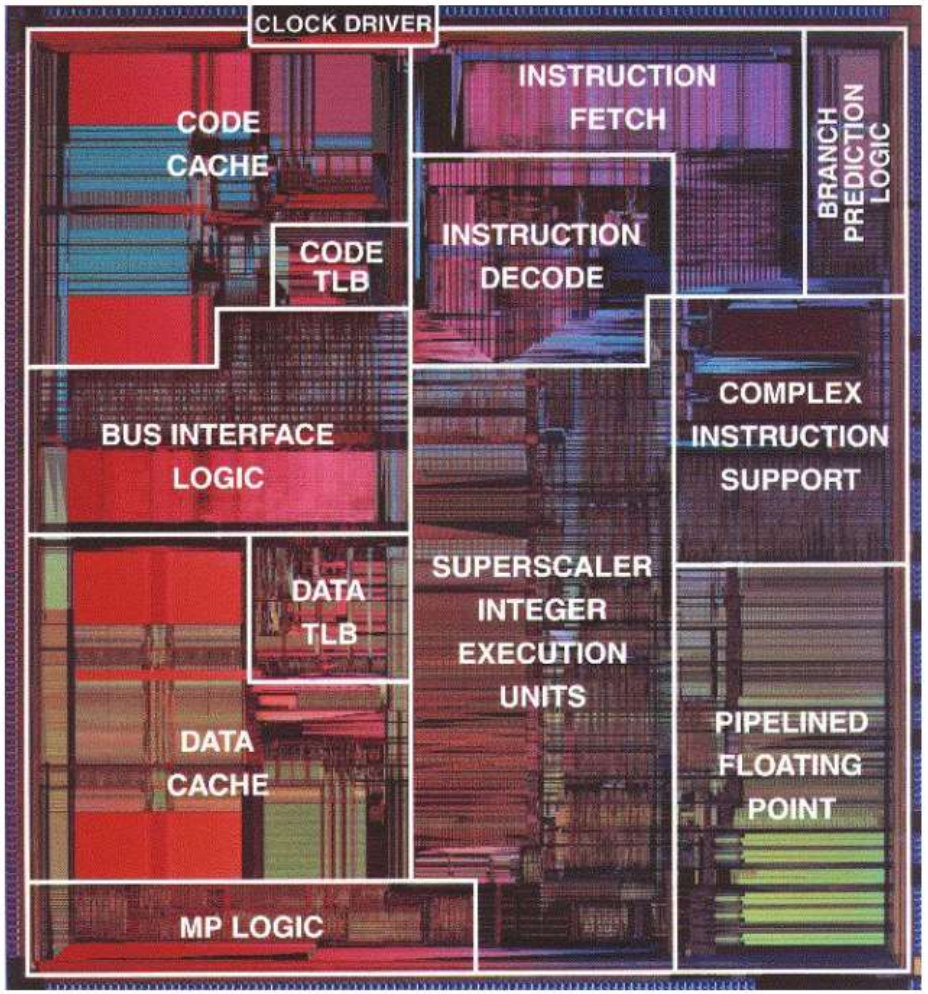
Verilog: Module

- Модуль – основная конструкция языка
- Представляет схему или часть схемы

```
module module_name(
    port_direction port_name,
    port_direction port_name,
    . . .
);

    // module description

endmodule
```

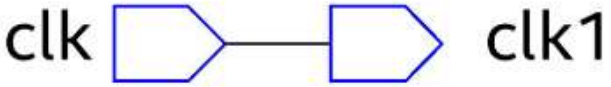


Примеры модулей ЦПУ



Verilog: Module

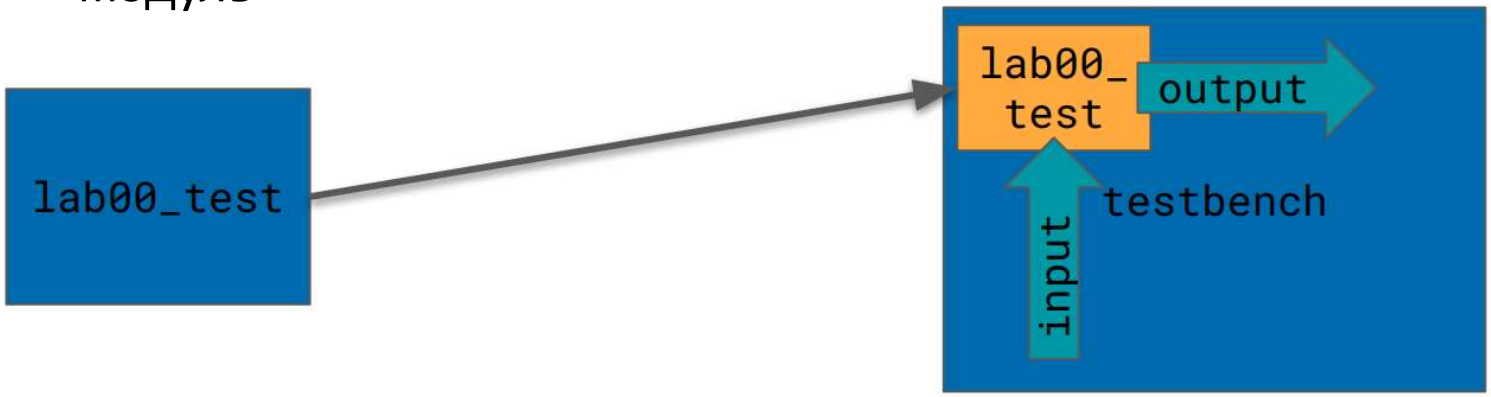
```
module lab00_test(
    input clk,
    output clk1
);

    assign clk1 = clk;

endmodule
```



Проверка работоспособности схемы

Тестируемый **синтезируемый** модуль

Тестирующий **несинтезируемый** модуль

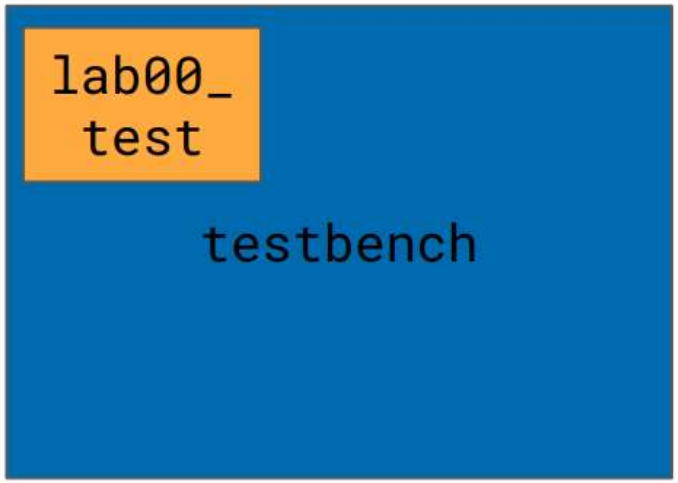


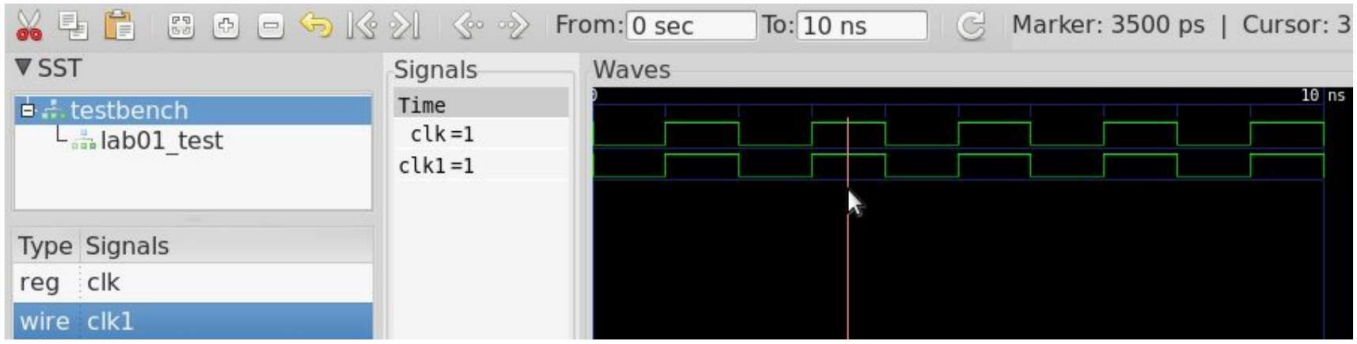
lab00_test.v

testbench.v

Тестовое окружение

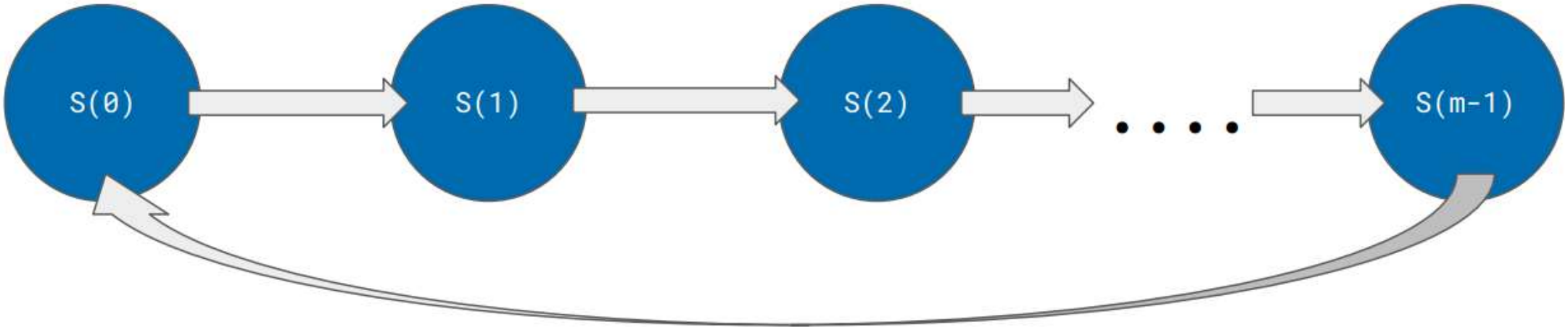
```
`timescale 1 ns /100 ps

module testbench();

    reg clk = 1'b0;
    always begin
        #1 clk = ~clk;
    end
    wire clk1;

    lab00_test lab00_test(.clk(clk), .clk1(clk1));

    initial begin
        $dumpvars;
        $display("Test started...");
        #10 $finish;
    end

endmodule
```





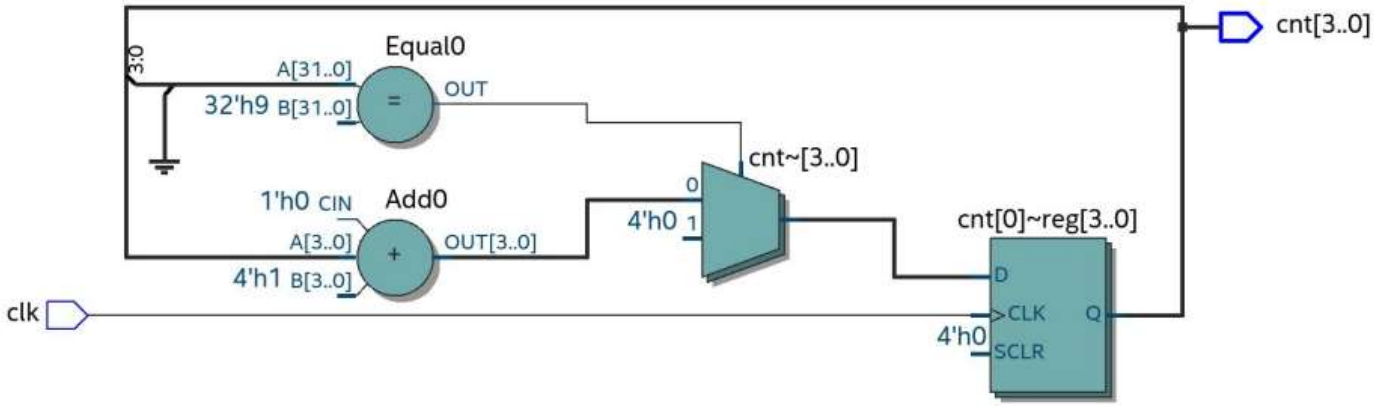
Счетчик

- Тактируемая последовательностная схема
- Диаграмма состояний — единственное кольцо
- Счетчик от 0 до m-1 имеет m состояний
- Счетчик от 0 до 2^n-1 требует n бит и называется двоичным nразрядным счетчиком
- Может использоваться как делитель частоты на m



Счетчик

```
module counter(
    input clk
);

reg [3:0]cnt = 0;

always @(posedge clk) begin
    if (cnt == 9)
        cnt <= 0;
    else
        cnt <= cnt + 1;
end

endmodule
```



Счетчик

```
module counter(
    input clk
);

reg [3:0]cnt = 0;

always @(posedge clk) begin
    if (cnt == 9)
        cnt <= 0;
    else
        cnt <= cnt + 1;
end

endmodule
```