

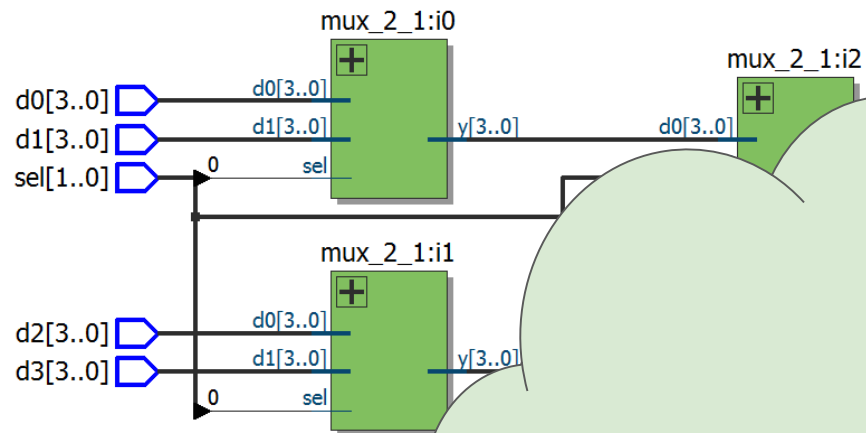
HDL, RTL и FPGA

Цифровая схемотехника. День 2

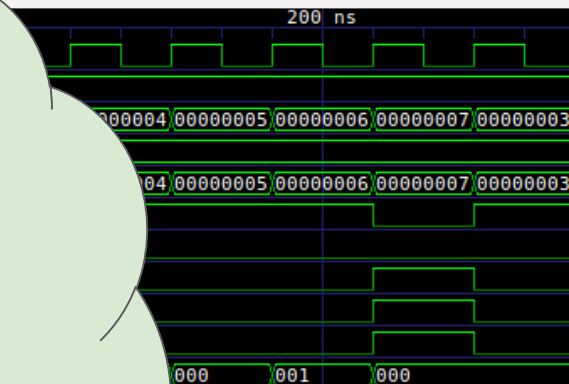
Yuri Panchul, Senior Hardware Design Engineer, MIPS
Stanislav Zhelnio, Hardware Design Engineer, IVA Technologies

MIPS

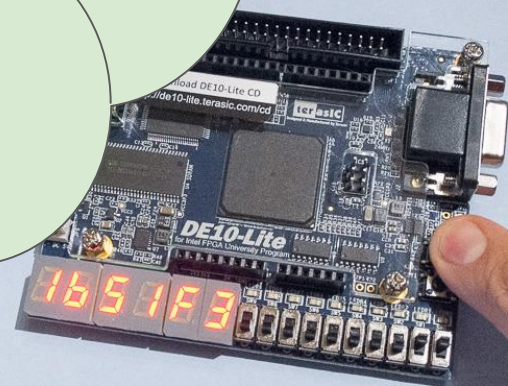
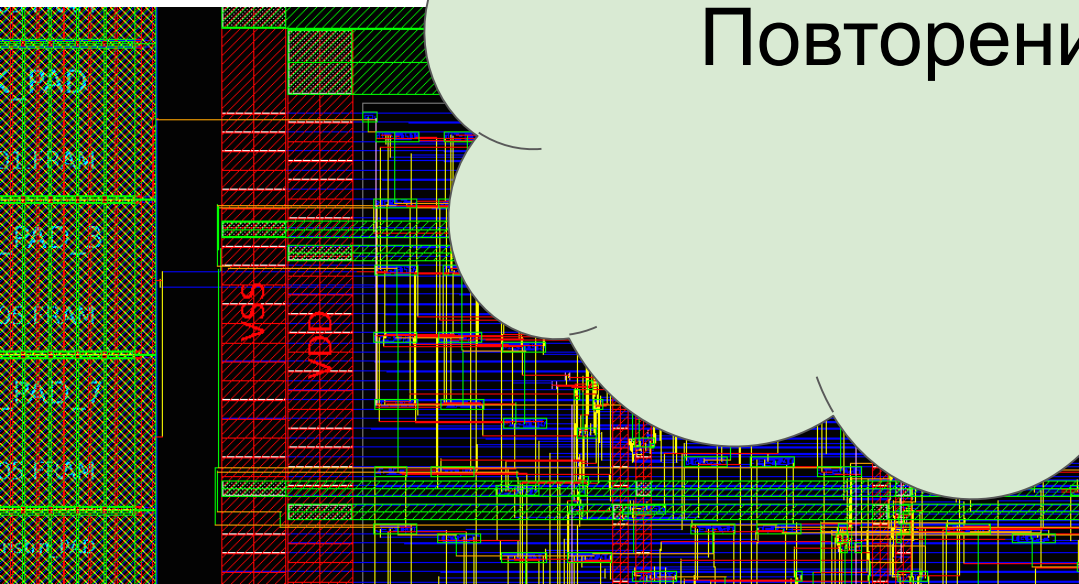




Marker: 670 ns | Cursor: 290500 ps



Повторение



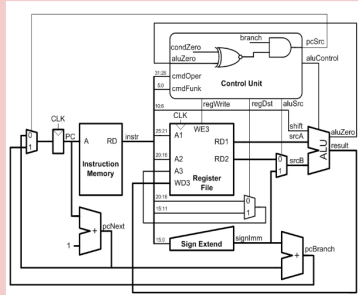
Пример двойственности аппаратной части и ПО 1/2

Микроконтроллер (embedded chip, ASIC, SoC)

Процессор

Написан на Verilog

Выполняет инструкции



Память

Содержит программу -
последовательность инструкций

Скомпилирована из исходных
кодов на C или ином языке

18800005
00001025
00451021
0044182a
5460ffff
00451021
03e00008
00000000

Программа: от кода на С к машинным кодам

C:

```
int f (int a, int b)
{
    int s = 0;

    while (s < a)
        s += b;

    return s;
}
```

Ассемблер:

```
sum:
    blez    $4, exit
    move    $2, $0

    addu    $2, $2, $5

loop:
    slt     $3, $2, $4
    bnel    $3, $0, loop
    addu    $2, $2, $5

exit:
    jr      $31
    nop
```

**Машинный
код**

```
18800005
00001025

00451021

0044182a
5460fffe
00451021

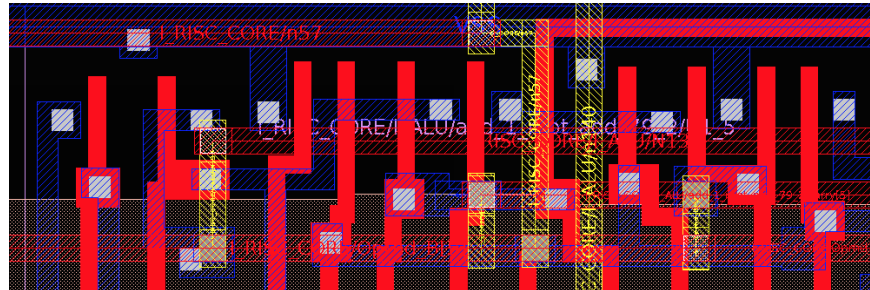
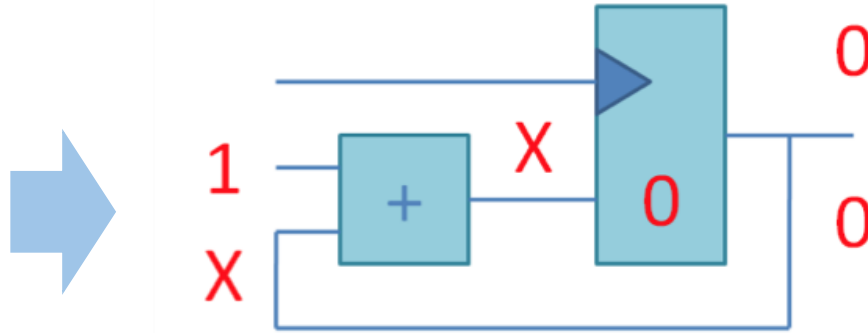
03e00008
00000000
```

Схемотехника: от Verilog к транзисторам (упрощенно)

```

module counter
(
    input  clock,
    input  reset,
    output logic [1:0] n
);
    always @(posedge clock)
    begin
        if (reset)
            n <= 0;
        else
            n <= n + 1;
    end
endmodule

```

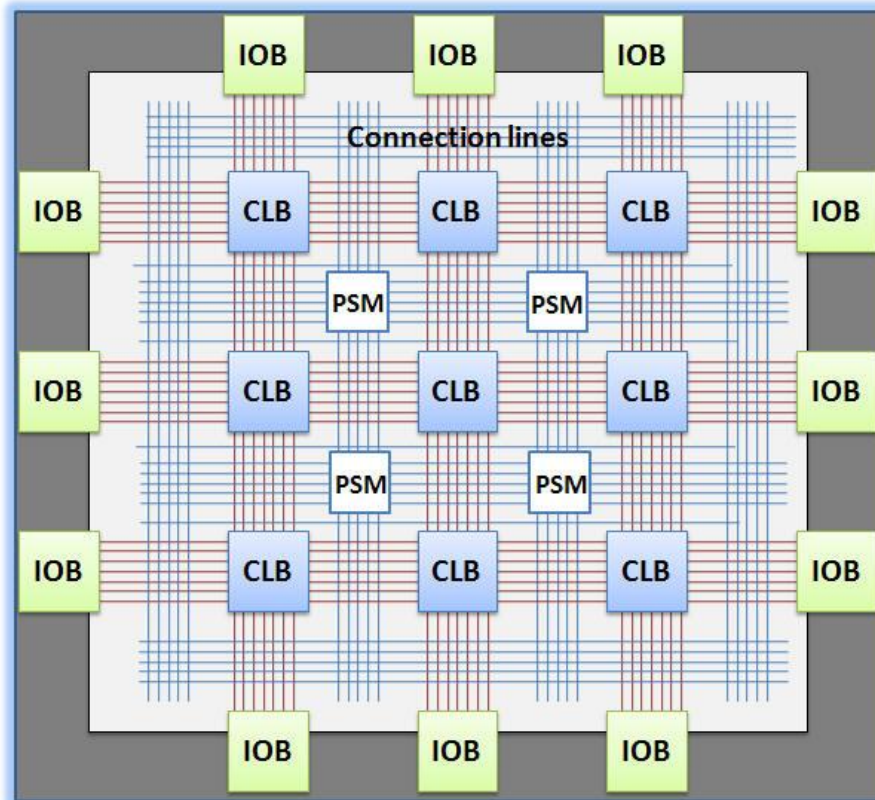


Что такое FPGA? Простыми словами

Матрица из логических ячеек

Одна ячейка может стать вентилем AND, другая OR, следующая — одним битом памяти

Внутри обычной FPGA нет процессора, но она может быть сконфигурирована так, чтобы работать как процессор

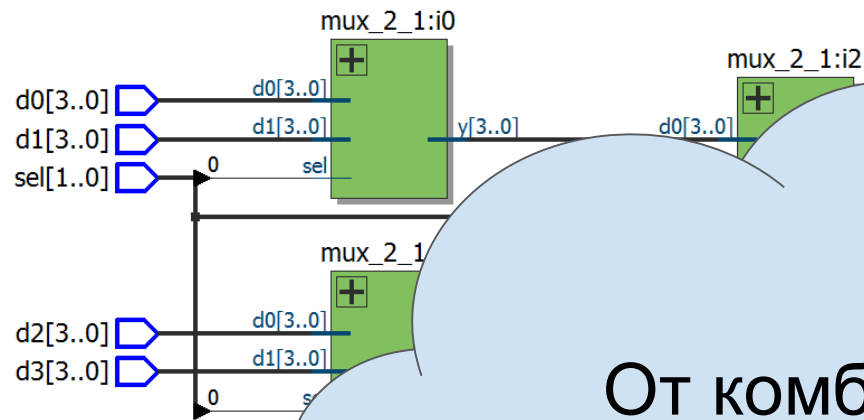


IOB
Input Output Block

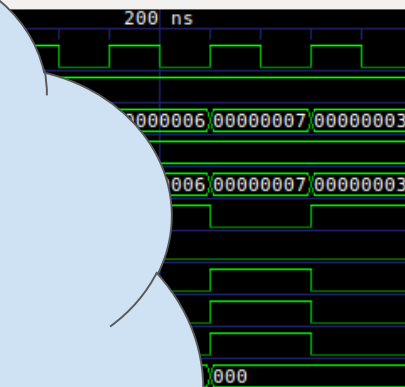
CLB
Configurable
Logic Block

PSM
Programable
Switch Matrix

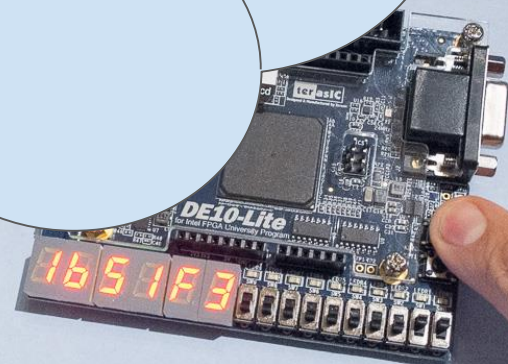
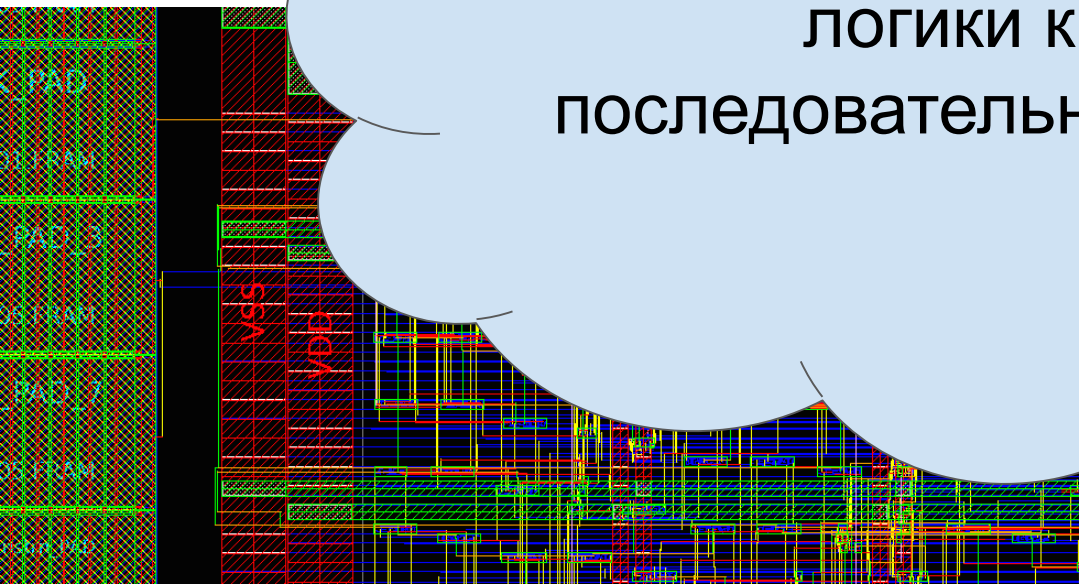
Connection lines
Single, Long
Double, Direct



Marker: 670 ns | Cursor: 290500 ps

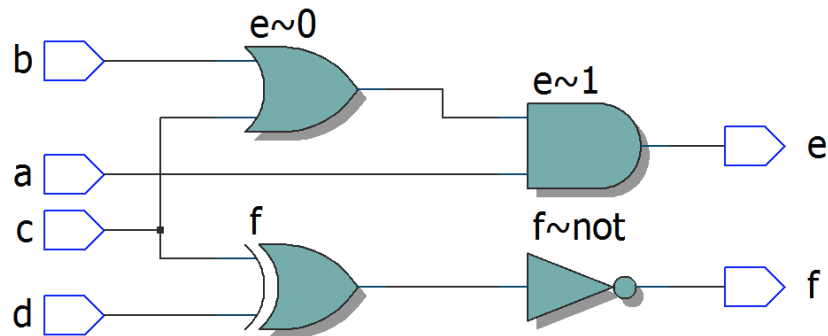


От комбинационной
ЛОГИКИ К
ПОСЛЕДОВАТЕЛЬНОСТНОЙ



Комбинационная логика

- Выходные сигналы группы логических вентилях зависят только от её входных сигналов
- Установление значений на выходах занимает некоторое время
- Такая группа вентилях называется «комбинационным облаком» (combinational cloud)
- Используется для вычисления логических и арифметических функций



```
module top
(
    input  a, b, c, d,
    output e, f

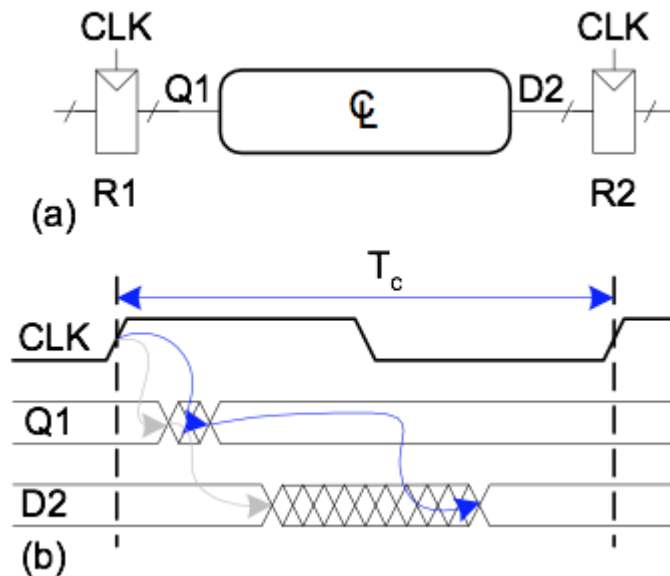
);

    assign e = a & (b | c);
    assign f = ~ (c ^ d);

endmodule
```


Комбинационная логика не работает мгновенно

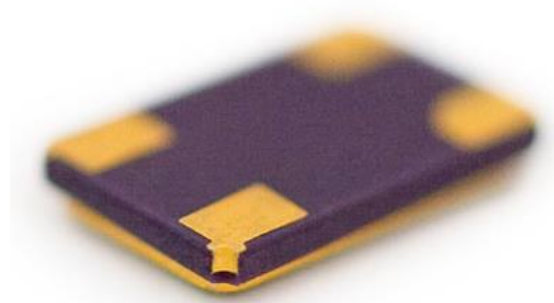
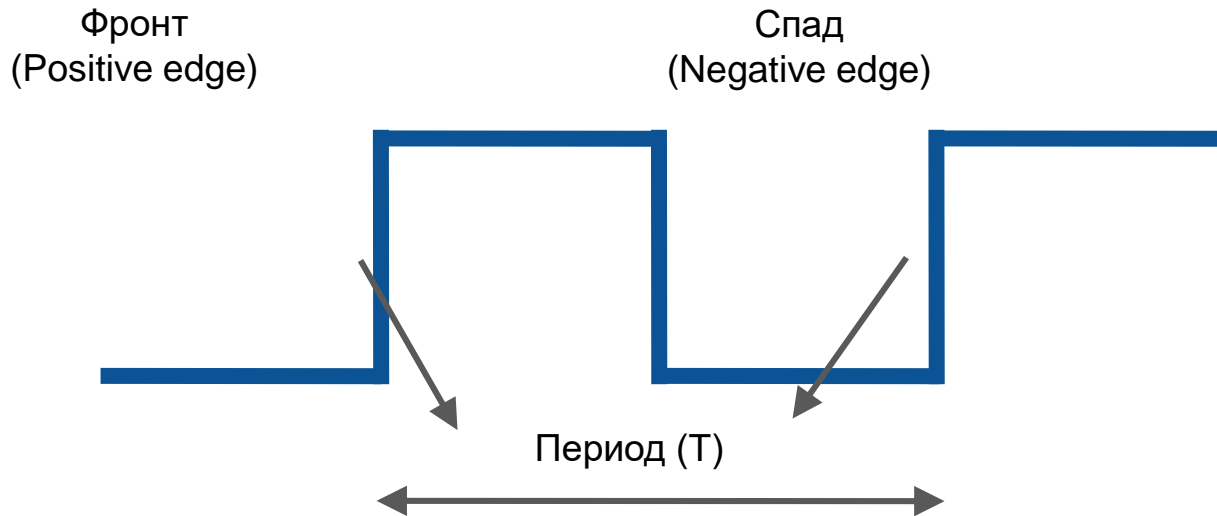
- Для распространения сигнала требуется время
- Перед тем, как сигналы примут окончательное значение на выходах блока могут появляться случайные значения
- Как определить момент, когда сигнал можно использовать?
- Мы можем синхронизировать вычисления с помощью специального тактового сигнала (clock).



The picture is from Digital Design and Computer Architecture
2nd Edition by David Harris and Sarah Harris. Elsevier, 2012

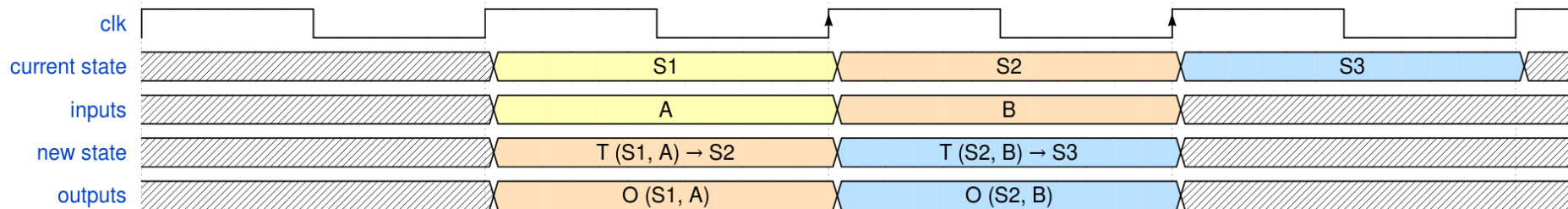
Тактовый сигнал (Clock)

- Периодический сигнал прямоугольной формы (меандр, square waveform)
- Его период (T) достаточен для того, чтобы завершились вычисления в любой из комбинационных схем
- Частота тактового сигнала: $F = 1 / T$.
- Тактовый сигнал обычно генерируется с помощью кварцевого генератора (найдите его на плате).

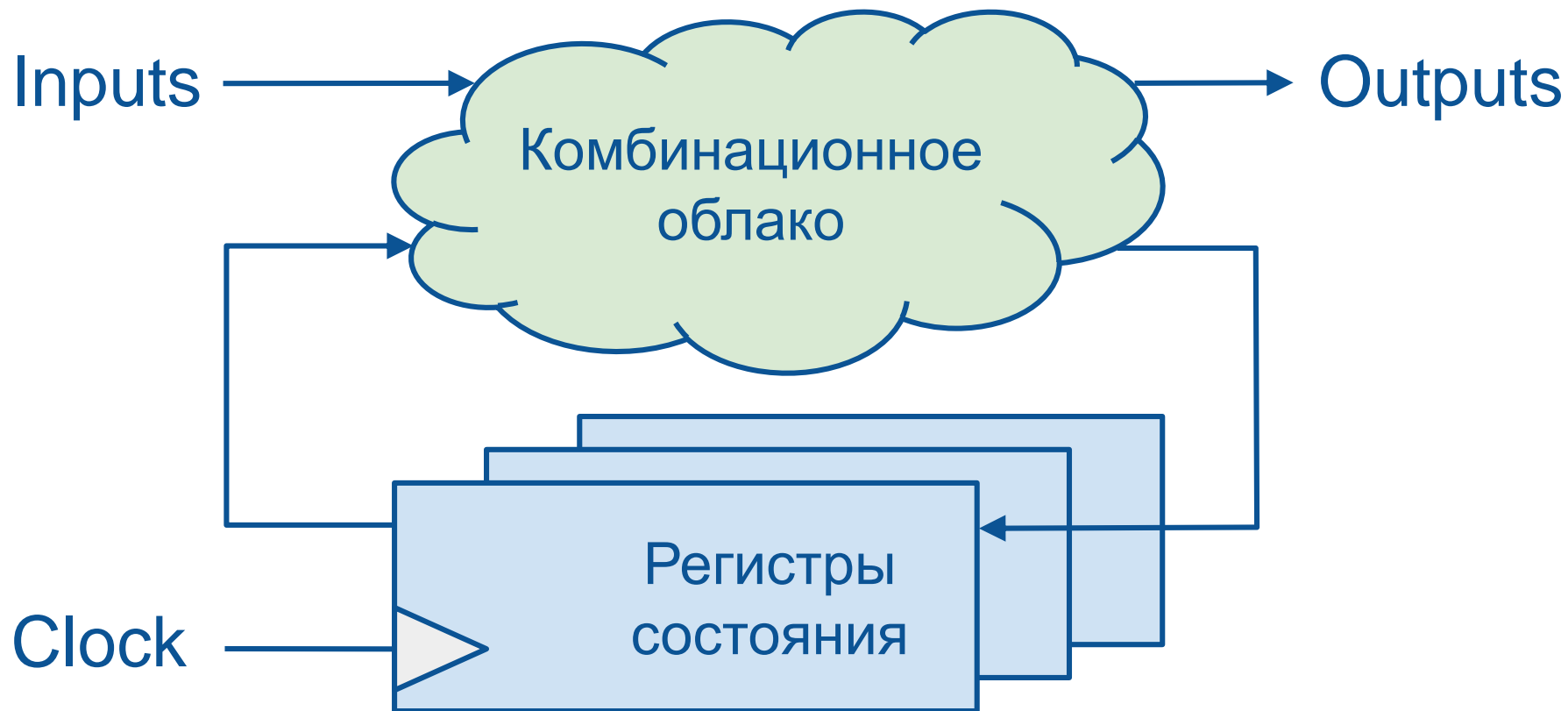


Последовательностная — схема с тактовым сигналом

- Последовательностная схема во время работы проходит через последовательность состояний
- Текущее состояние схемы сохраняется в D-триггерах (D-flip-flops)
- Следующее состояние вычисляется на основе текущего и сигналов на входе схемы
- Следующее состояние сохраняется в D-триггеры по фронту тактового сигнала
- Значение сигналов на выходах вычисляется на основе текущего состояния и сигналов на входе схемы

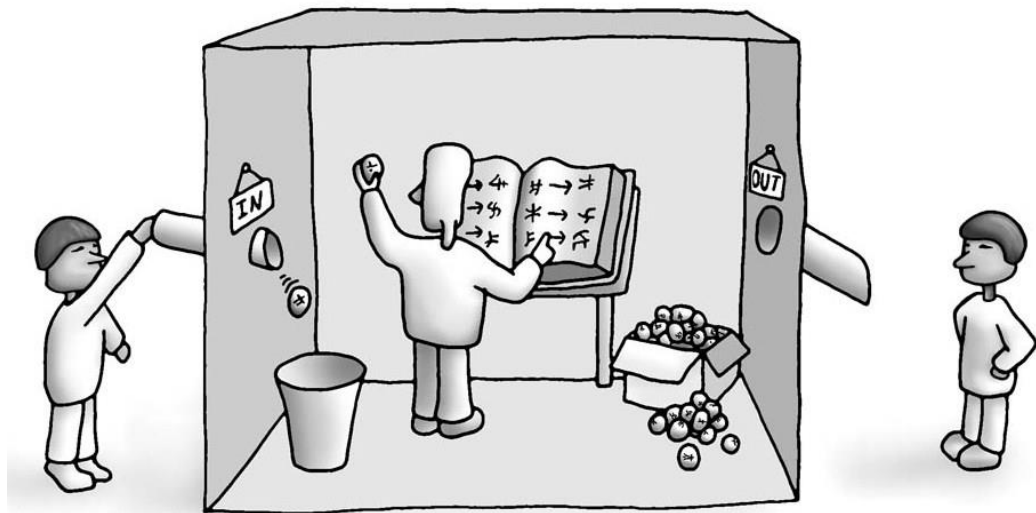


Модель Хаффмана для последовательных схем



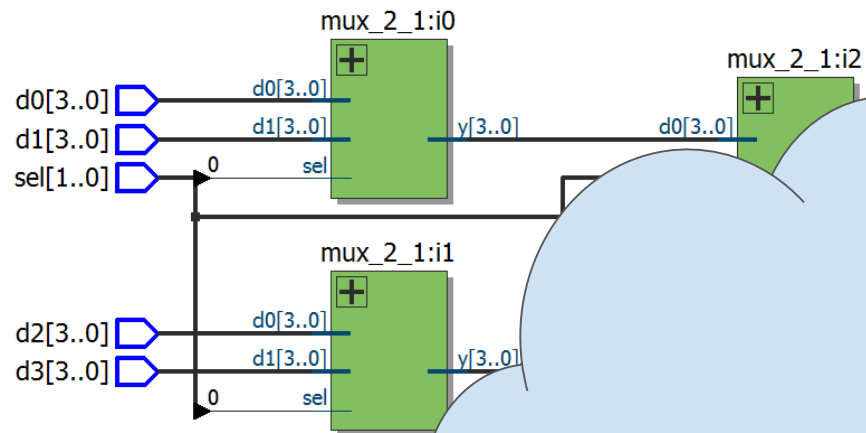
Что дает последовательная логика

- Счетчики
- Запоминание информации
- Добавление новых данных и повторение вычислений
- Ожидание события, поступающего извне
- Вкратце:
последовательная логика — это то, что заставляет компьютер делать интересные вещи

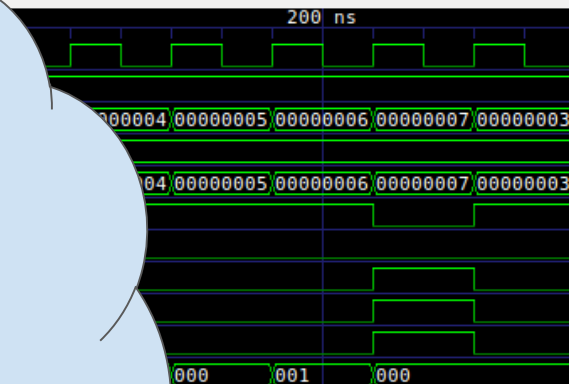


Изображение эксперимента «Китайская комната»:

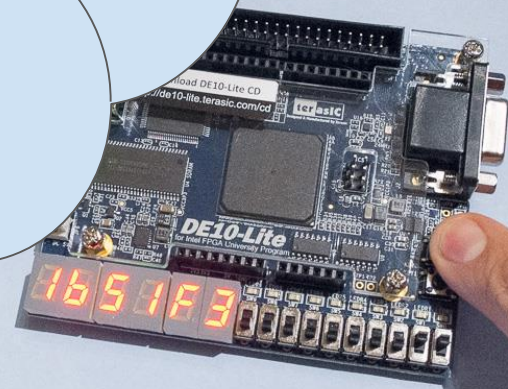
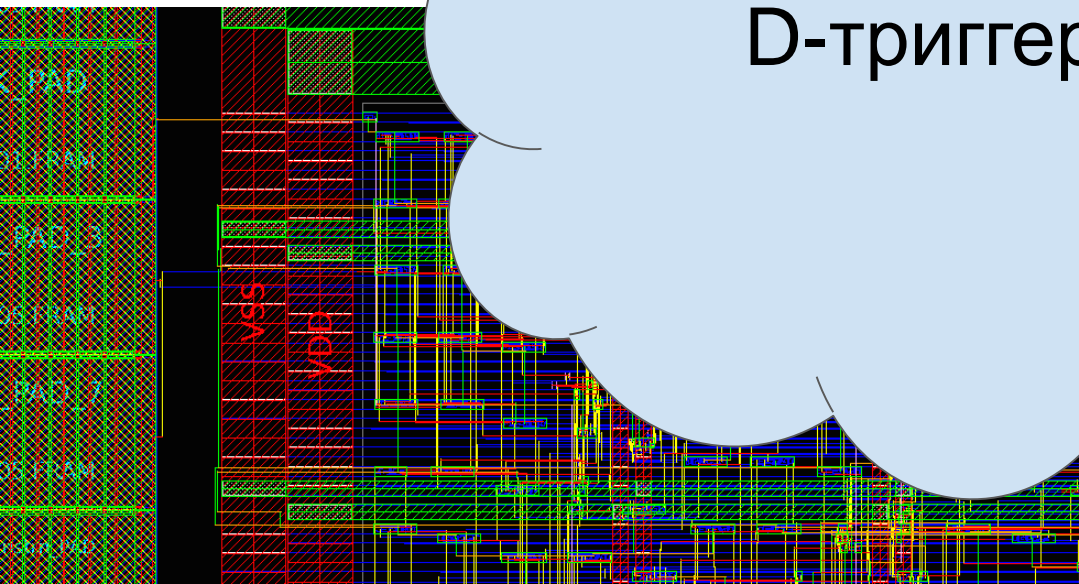
<http://deskarati.com/2014/07/01/john-searles-chinese-room-thought-experiment>



Marker: 670 ns | Cursor: 290500 ps



D-триггер

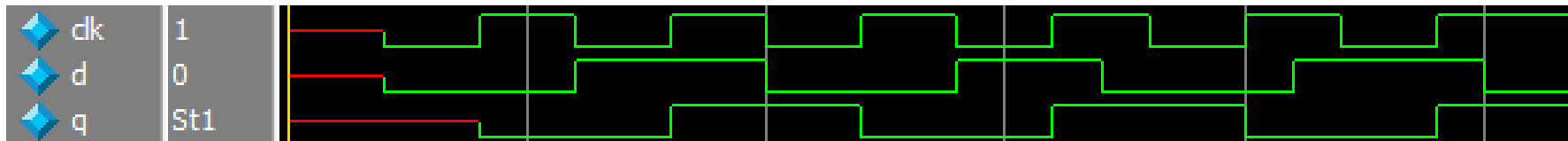


D-триггер – от же D-flip-flop, DFF

- “Каждый раз по фронту **clk** сохрани значение сигнала **d** в триггер сформированный для сигнала **q**. **q** также подключен к выходу.
- Блок **always** похож на **initial**, однако он вызывается каждым событием, перечисленным после **@**
- Присваивание **<=** называется **неблокирующим**, оно срабатывает сразу после обработки текущего блока **always**

```
module d_flip_flop  
(  
    input    clk,  
    input    d,  
    output   reg q  
);
```

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



Как выбрать тип присваивания

- Используйте **неблокирующее** присваивание (**`<=`**) внутри блока **`always`** **`@(posedge clk)`** при описании последовательностной логики:

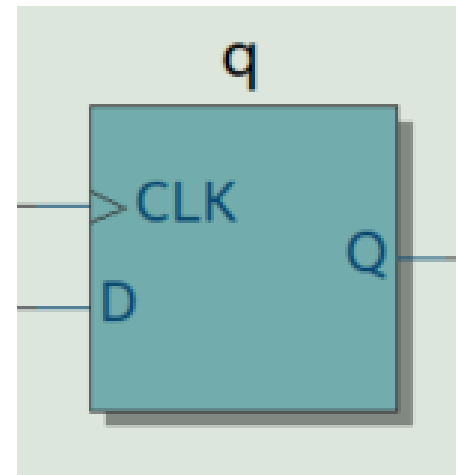
```
reg a;  
always @(posedge clk)  
    a <= b;
```
- Используйте **непрерывное** присваивание для описания простой комбинационной логики:

```
wire a;  
wire b = c;  
assign a = b;
```
- Используйте **блокирующее** присваивание (**`=`**) внутри блока **`always`** **`@(*)`** при описании более сложной комбинационной логики:

```
reg a;  
always @(*) begin  
    a = b;  
end
```

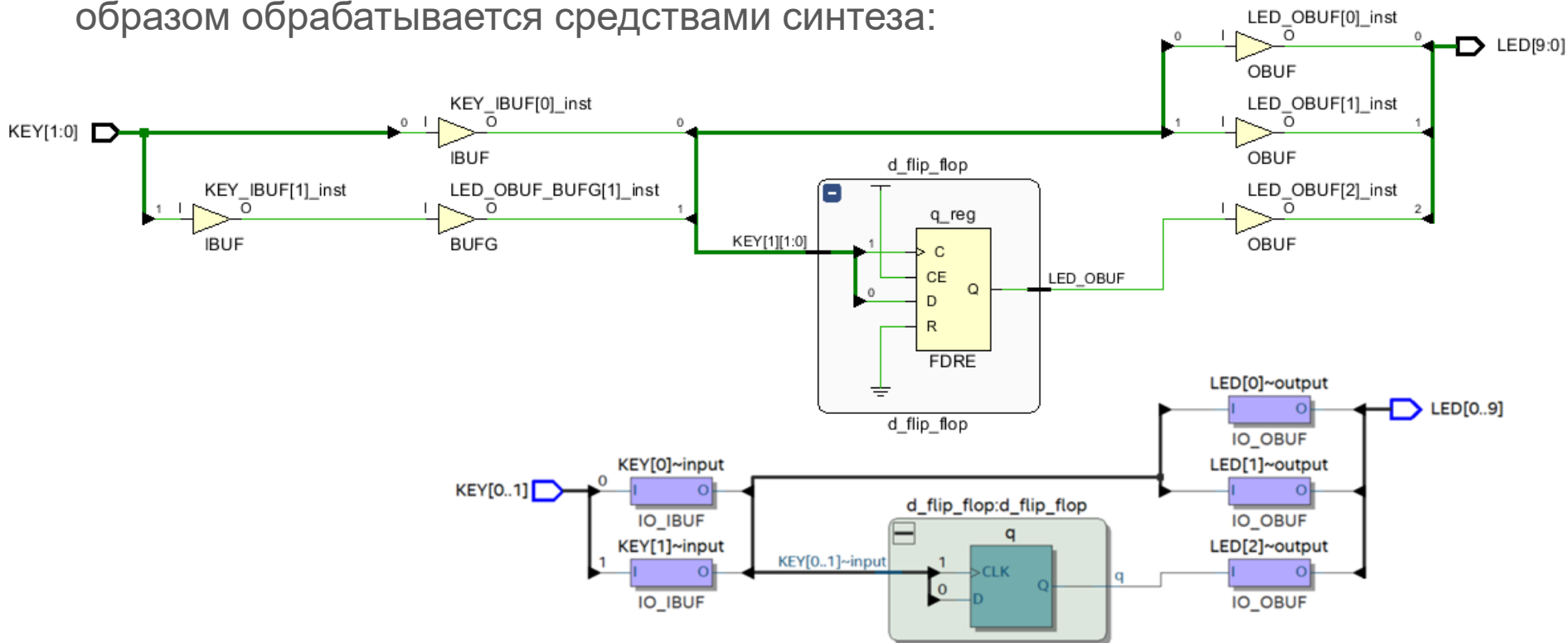
D-триггер - практика

- Откройте файл **ddec/lab/21_seq_dff/rtl/d_flip_flop.v**
- Откройте консоль в каталоге **ddec/lab/21_seq_dff**
- Выполните симуляцию примера:
make sim
- Выполните синтез и проверку работы примера на плате:
make synth
make load
make open
- Откройте синтезированный проект в режимах:
RTL-Viewer
Technology Map Viewer
Resource Property Editor

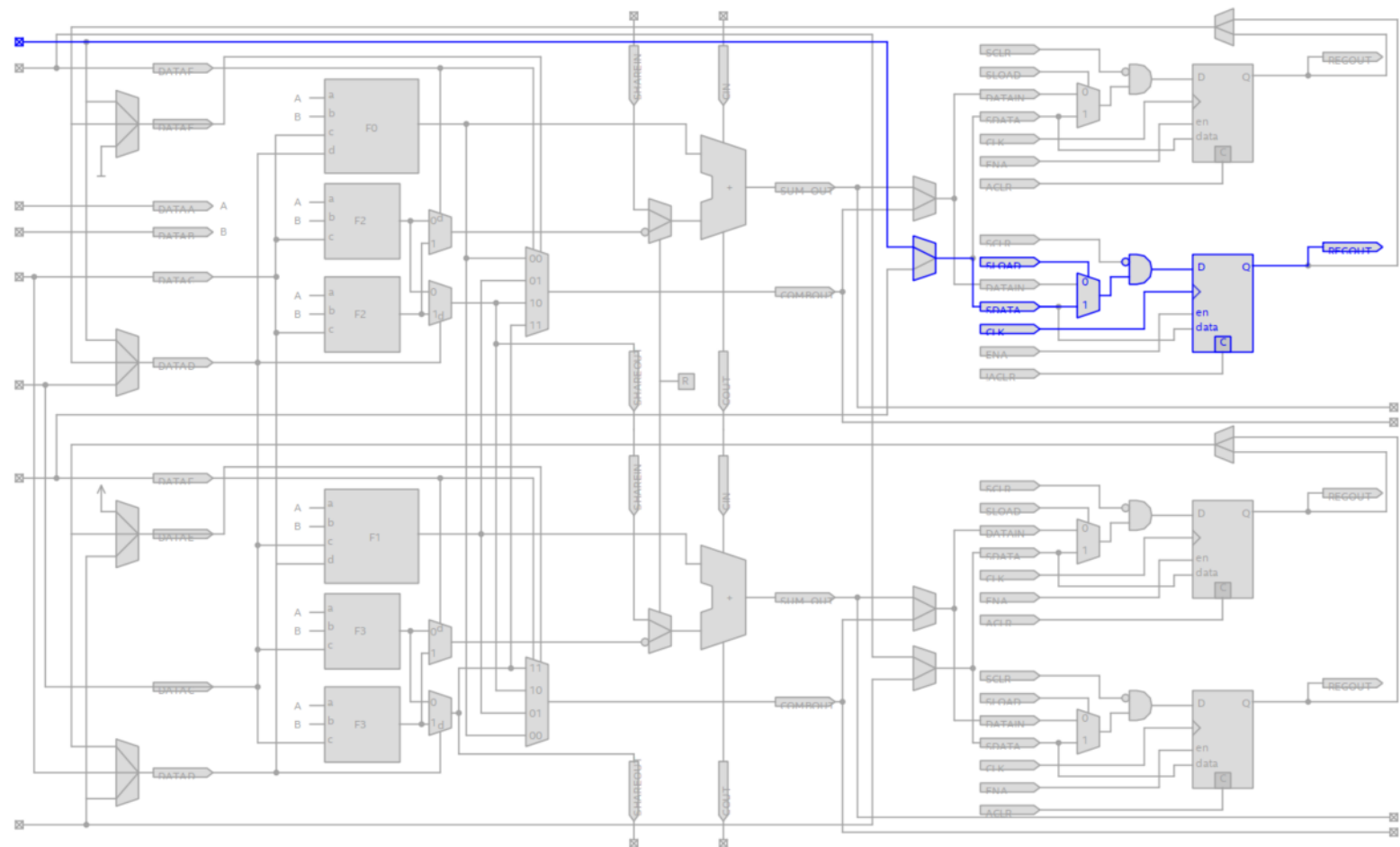


D-триггер и его синтез

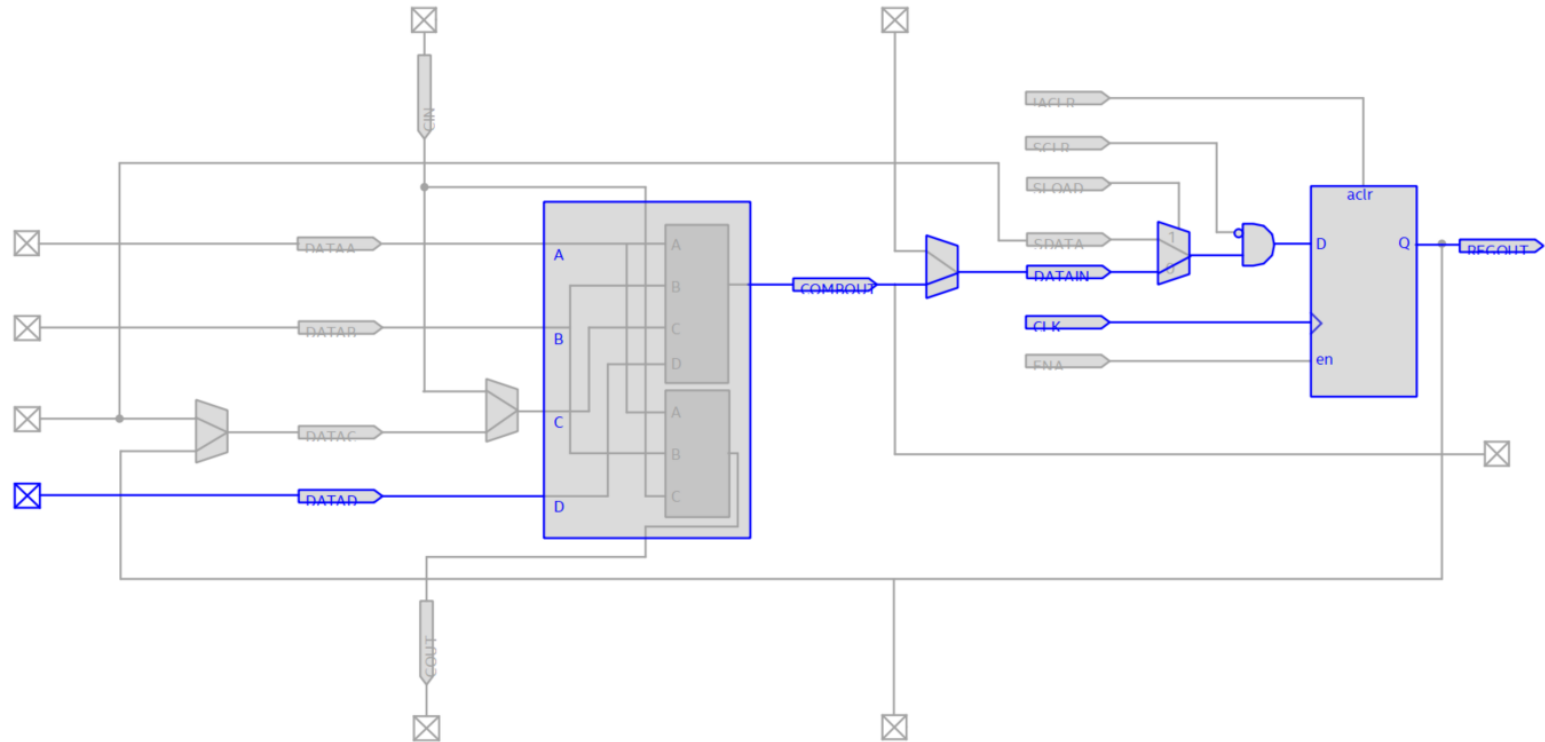
- Блок **always** @(posedge clk) распознается как D-триггер и соответствующим образом обрабатывается средствами синтеза:



D-триггер на базе логической ячейки FPGA Cyclone 5

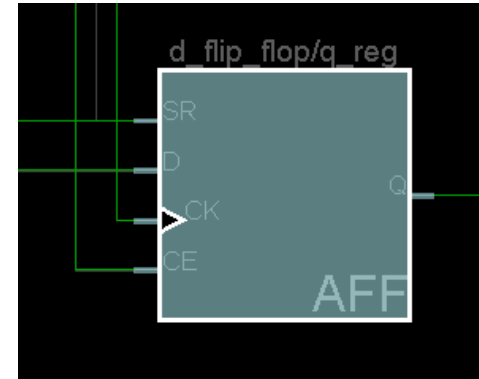
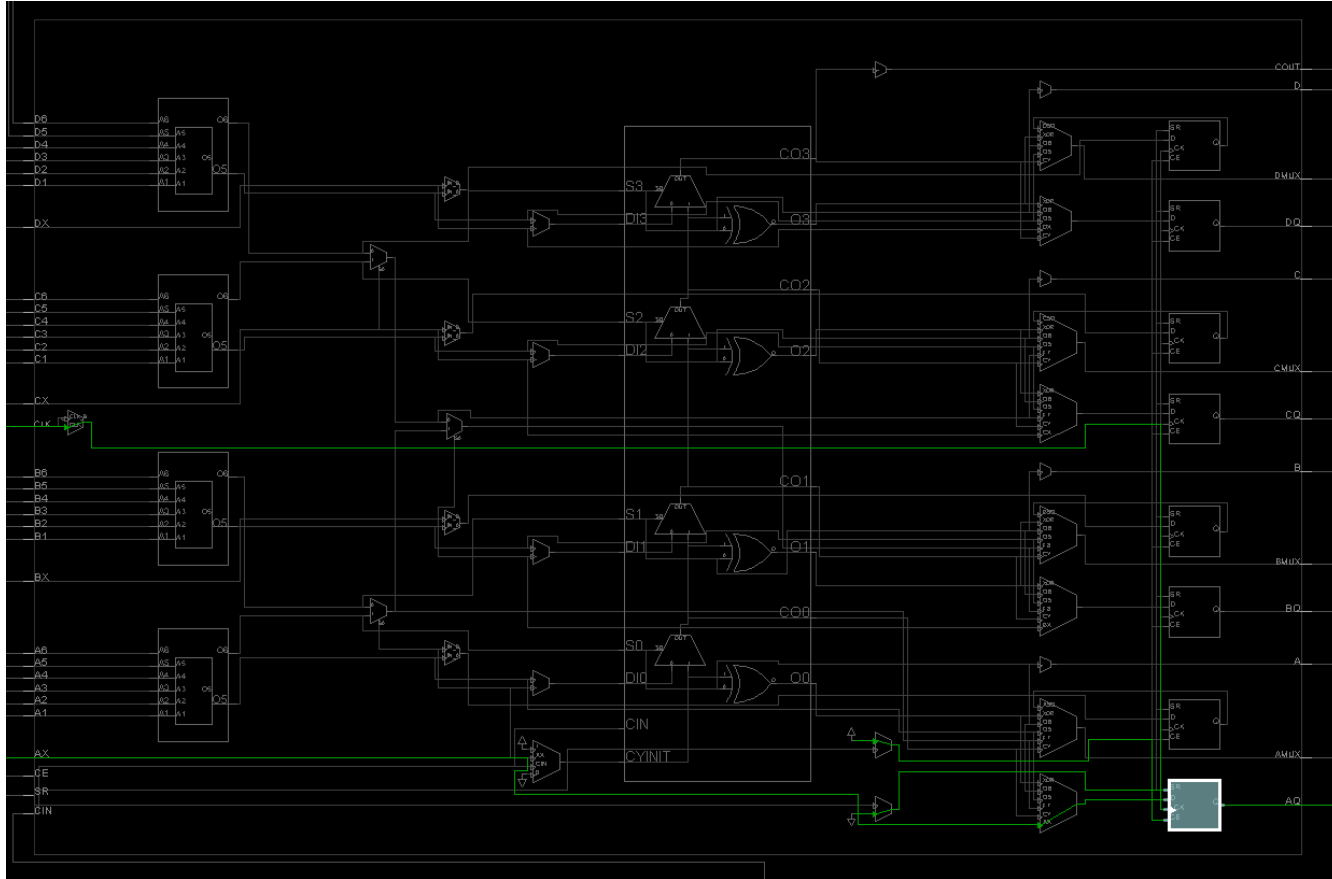


D-триггер на базе логической ячейки FPGA MAX10



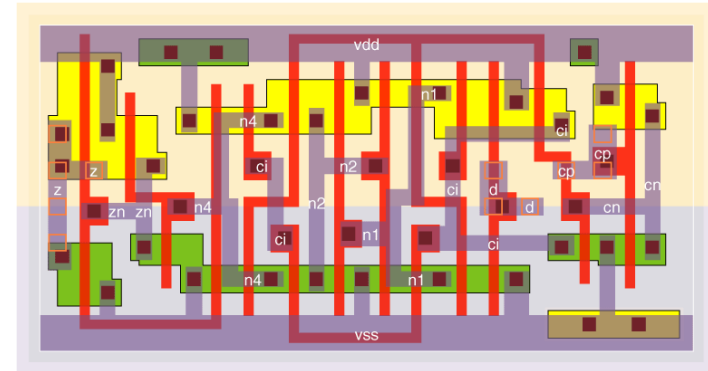
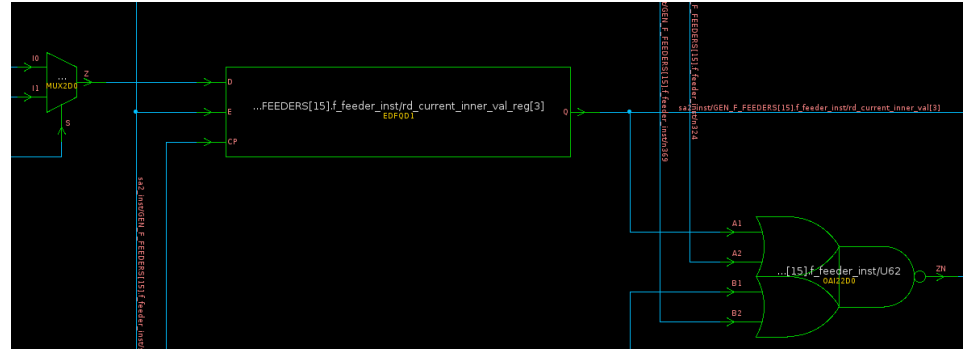
В данном случае Quartus провел входной сигнал через LUT – это не является признаком его «неоптимальной» работы. Если схема удовлетворяет всем временным требованиям, то оптимизация не имеет смысла

D-триггер на базе логической ячейки FPGA Artix-7



D-триггер в ASIC

- D-триггер входит в **библиотеку стандартных ячеек (standard cell library)** – типовых «кирпичиков» из которых строится микросхема
- Фрагмент схемы, полученной из RTL кода, в результате синтеза в **Synopsys Design Compiler**. Помимо D-триггера показаны фрагменты комбинационных схем на его входе и выходе
- Топология стандартной ячейки D-триггера (**D-flip-flop**)



The picture is from

<http://www.vlsitechnology.org/html/cells/vsclib013/dfnt1.html>

D-триггер с асинхронным сбросом

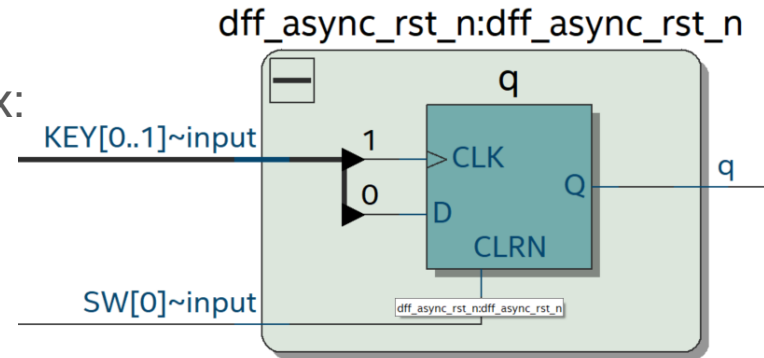
- Цепь сброса задает начальное состояние, в которое триггер переходит при включении питания
- В данном примере показан сброс активный в нуле (active low, “`negedge rst_n`”), в ряде схем используется сброс активный в единице (active high)
- Сброс необходим для сигналов управления, для чтобы устройство корректно работало после подачи питания

```
module dff_async_rst_n
(
    input        clk,
    input        rst_n,
    input        d,
    output reg    q
);
```

```
always @ (posedge clk or negedge rst_n)
    if (!rst_n)
        q <= 0;
    else
        q <= d;
```

D-триггер с асинхронным сбросом - практика

- Откройте файл **ddec/lab/22_seq_dff_arst/rtl/dff_async_rst_n.v**
- Откройте консоль в каталоге **ddec/lab/22_seq_dff_arst**
- Выполните симуляцию примера:
make sim
- Выполните синтез и проверку работы примера на плате:
make synth
make load
make open
- Откройте синтезированный проект в режимах:
RTL-Viewer
Technology Map Viewer
Resource Property Editor



D-триггер с синхронным сбросом

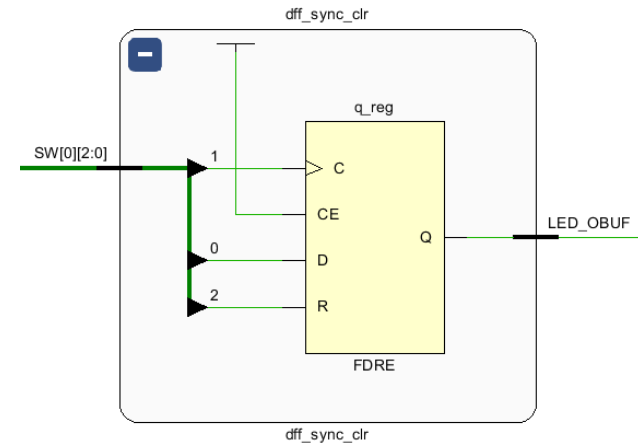
- Сигнал сброса может быть синхронным – в этом случае сброс триггера происходит по фронту тактовой частоты **clk**
- В зависимости от реализации сигнал сброса может быть активен в нуле, либо активен в единице
- У триггера могут присутствовать и использоваться оба входа сброса, к примеру:
 - асинхронный сброс **rst_n** используется для сброса всего чипа
 - значение сигнала синхронного сброса **clr** задается логикой того модуля, в состав которого входит D-триггер

```
module dff_sync_clr
(
    input      clk,
    input      clr,
    input      d,
    output reg q
);

always @ (posedge clk)
    if (clr)
        q <= 0;
    else
        q <= d;
```

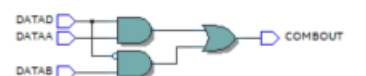
D-триггер с синхронным сбросом - практика

- Откройте файл **ddec/lab/23_seq_dff_clr/rtl/dff_sync_clr.v**
- Откройте консоль в каталоге **ddec/lab/23_seq_dff_clr**
- Выполните симуляцию примера: **make sim**
- Выполните синтез и проверку работы примера на плате:
make synth
make load
make open
- Откройте синтезированный проект в режимах:
RTL-Viewer
Technology Map Viewer
Resource Property Editor
- Добавьте к данной реализации сигнал асинхронного сброса **rst_n** и повторите описанные выше действия

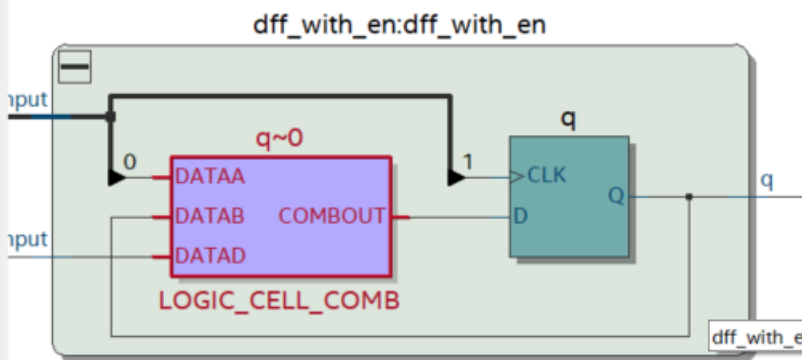


D-триггер с входом разрешения записи

- Запись нового значения происходит по фронту тактового сигнала **clk**, но только в тех случаях, когда сигнал разрешения записи **en** находится в высоком уровне



	DATAD	DATAB	DATAA	Q
q~0	0	0	0	0
	0	0	1	0
	0	1	0	1
	0	1	1	1
	1	0	0	0
	1	0	1	1
	1	1	0	0
	1	1	1	1



```
module dff_with_en
```

```
(
```

```
input      clk,
```

```
input      en,
```

```
input      d,
```

```
output reg q
```

```
);
```

```
always @ (posedge clk)
```

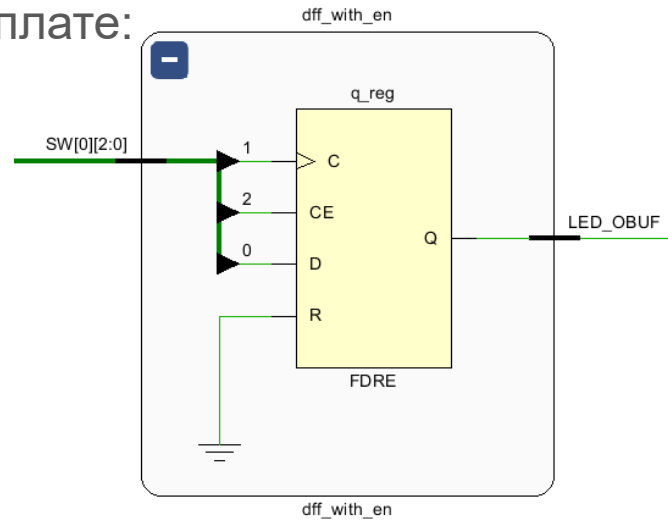
```
if (en)
```

```
q <= d;
```

```
endmodule
```

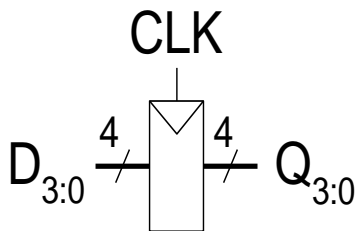
D-триггер с входом разрешения записи - практика

- Откройте файл **ddec/lab/24_seq_dff_en/rtl/dff_with_en.v**
- Откройте консоль в каталоге **ddec/lab/24_seq_dff_en**
- Выполните симуляцию примера: **make sim**
- Выполните синтез и проверку работы примера на плате:
make synth
make load
make open
- Откройте синтезированный проект в режимах:
RTL-Viewer
Technology Map Viewer
Resource Property Editor
- Добавьте порт разрешения записи к вашей реализации триггера с двумя сбросами и повторите все действия



Несколько D-триггеров образуют регистр

- Не путайте с ключевым словом **reg** или регистрами, которые доступны системному программисту в процессоре
- Количество D-триггеров, которые формируют регистр можно задавать с помощью параметра: **parameter**
- В данном примере параметром также задается начальное значение

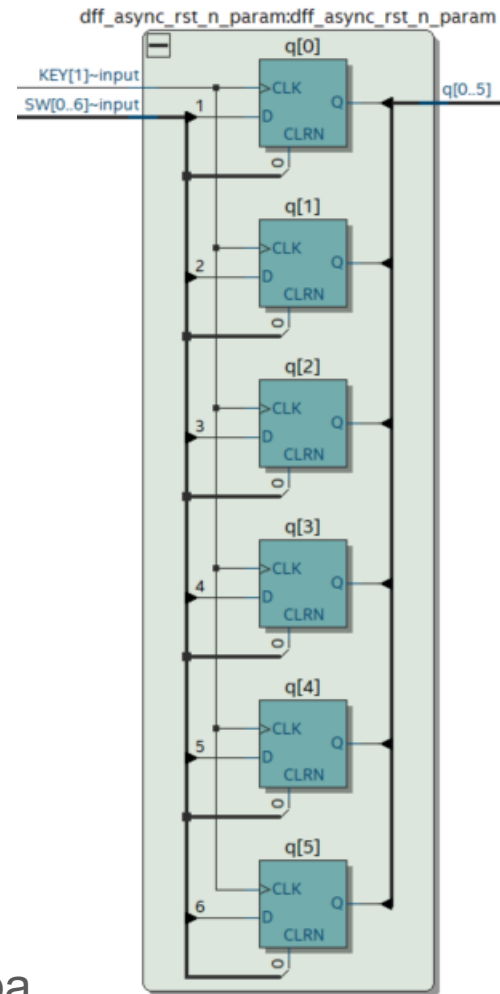


```
module dff_async_rst_n_param
#(
    parameter WIDTH = 8,
              RESET = 8'b0
)
(
    input                                clk,
    input                                rst_n,
    input    [WIDTH - 1 : 0] d,
    output reg [WIDTH - 1 : 0] q
);
```

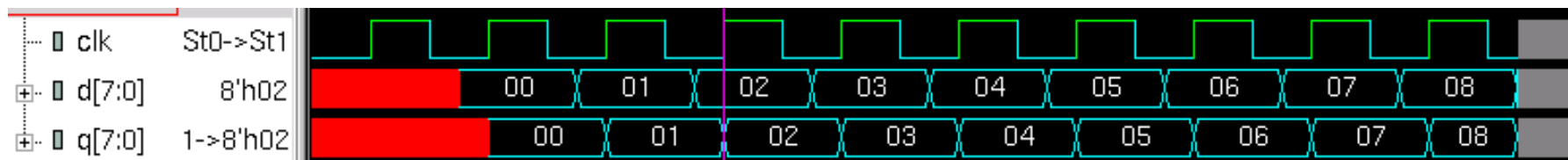
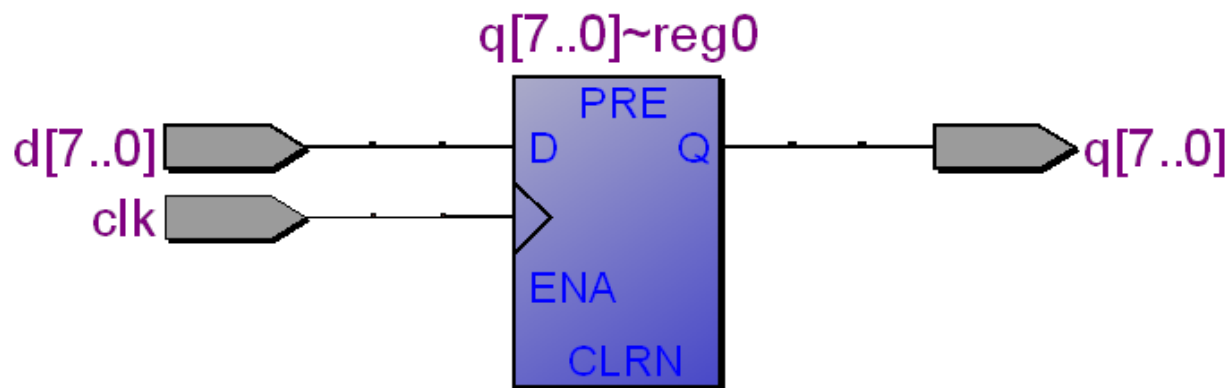
```
always @ (posedge clk or negedge rst_n)
    if (!rst_n)
        q <= RESET;
    else
        q <= d;
```


Регистр - практика

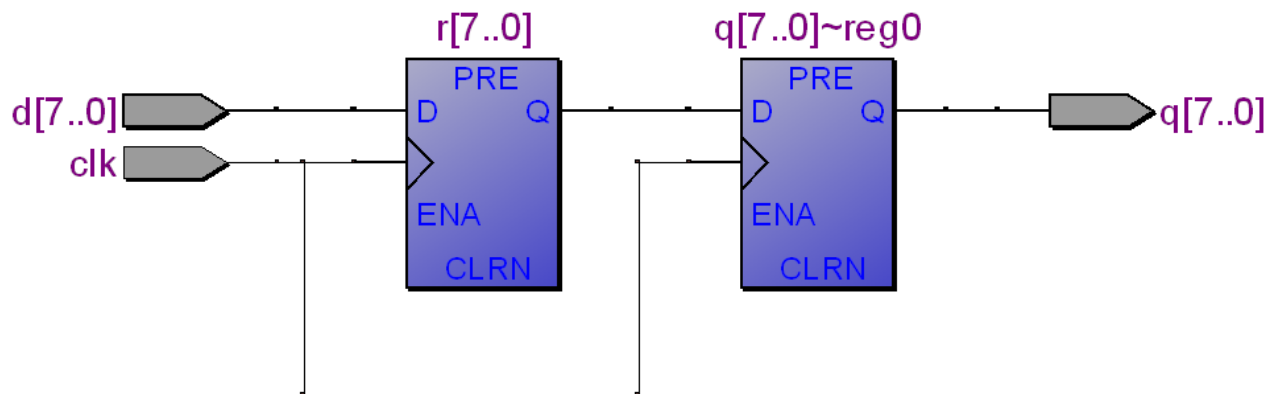
- Откройте файл **ddec/lab/25_seq_dff_param/rtl/dff_async_rst_n_param.v**
- Откройте консоль в каталоге **ddec/lab/25_seq_dff_param**
- Выполните симуляцию примера: **make sim**
- Выполните синтез и проверку работы примера на плате:
make synth
make load
make open
- Откройте синтезированный проект в режимах:
RTL-Viewer
Technology Map Viewer
Resource Property Editor
- Постройте регистр на базе ранее созданного вами триггера



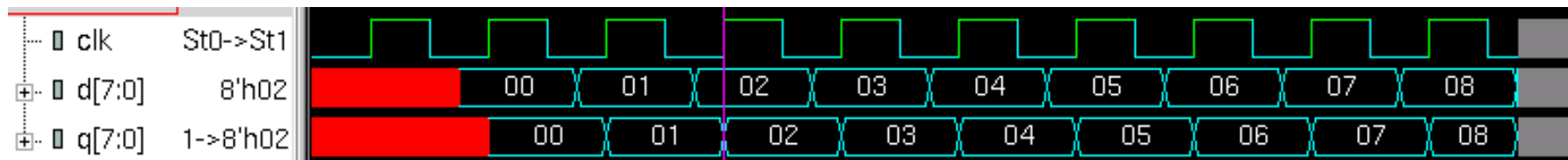
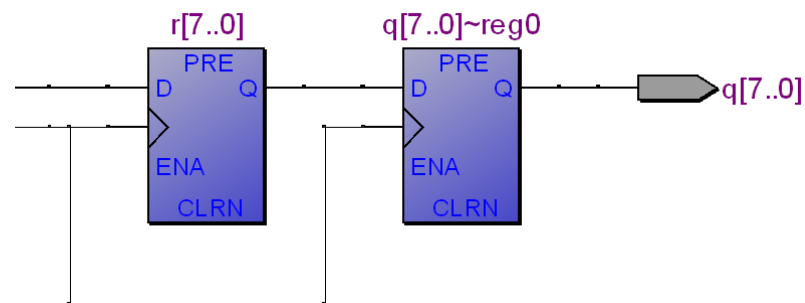
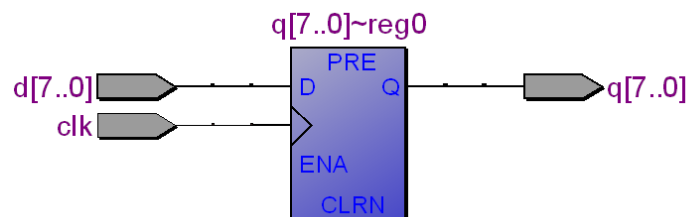
Регистр хранит значение в течении такта



Два последовательно соединенных регистра дают задержку в 2 такта



Сравните



Счетчик

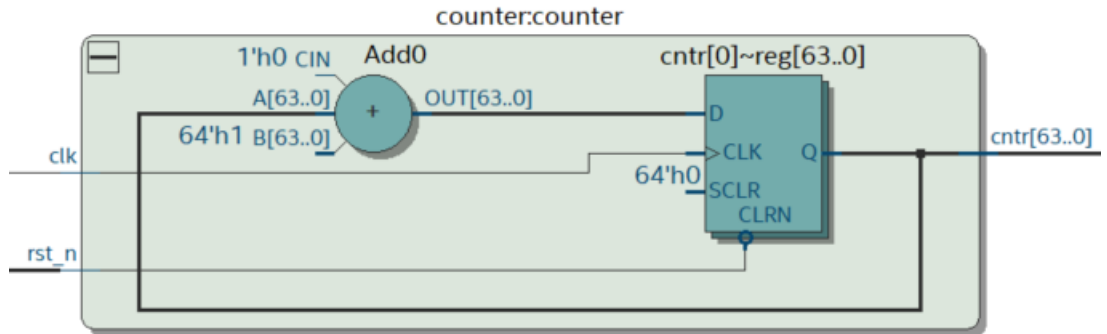
- Несколько D-триггеров образуют регистр
- Регистр + Сумматор = Счетчик
- Не путайте с ключевым словом `reg` или регистрами при программировании процессора.
- В данном примере результат сложения сохраняется для в регистре для использования в следующем такте

```
module counter
(
    input                clock,
    input                reset_n,
    output reg [31:0] count
);

always @(posedge clock or negedge reset_n)
begin
    if (!reset_n)
        count <= 32'b0;
    else
        count <= count + 32'b1;
end
```

Счетчик - практика

- Откройте файл **ddec/lab/26_seq_counter/rtl/counter.v**
- Откройте консоль в каталоге **ddec/lab/26_seq_counter**
- Выполните симуляцию примера: **make sim**
- Выполните синтез и проверку работы примера на плате:
make synth
make load
make open
- Откройте проект в режиме **RTL-Viewer**



Формирование и использование тактового сигнала в тестовом окружении

Тактовый сигнал можно использовать для того, чтобы своевременно подавать входные сигналы на порты (DUT = Design under Test).

```
initial
begin
    clk = 0;

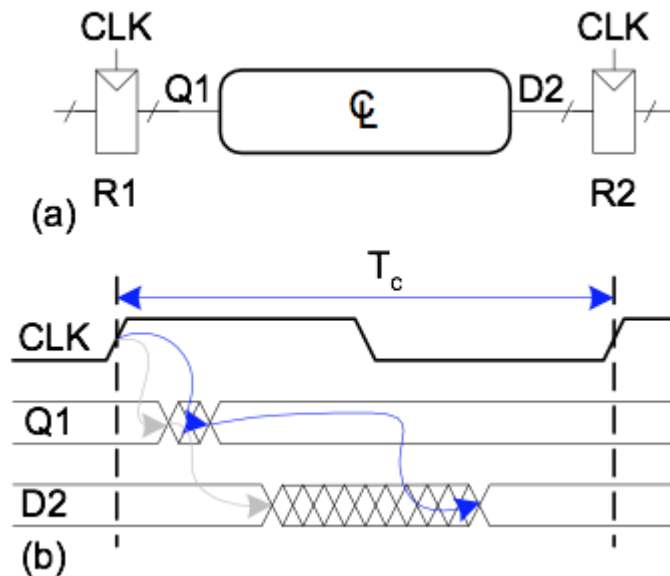
    forever
        # 10 clk = ! clk;
end
```

```
initial
begin
    clk_en  <= 1'b1;
    arg_vld <= 1'b0;

    repeat (2) @ (posedge clk);
    rst_n <= 0;
    repeat (2) @ (posedge clk);
    rst_n <= 1;
end
```

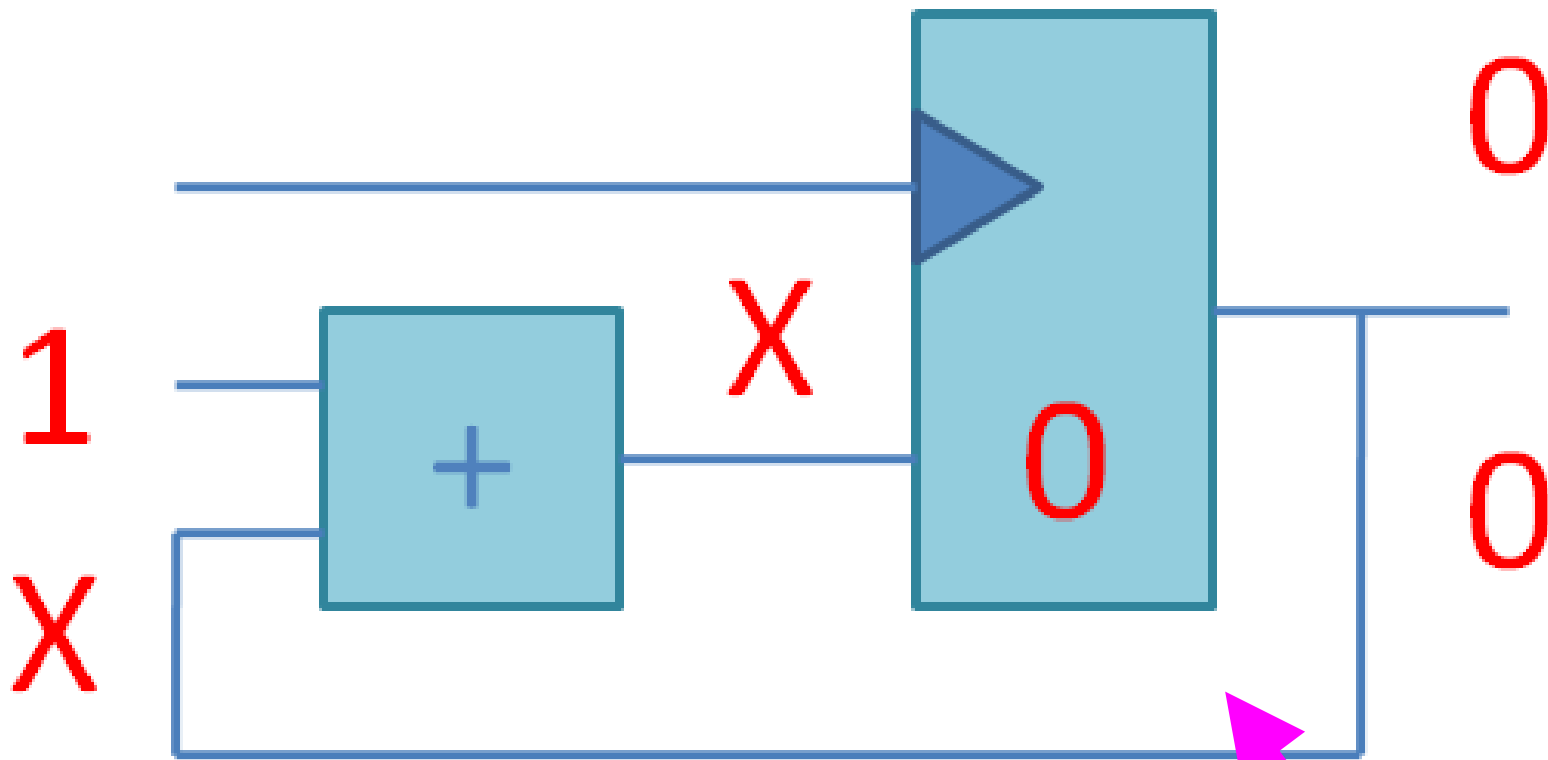

Сигнал распространяется не мгновенно

- Вспомним, что сигналу требуется время на распространение:
 - внутри регистра
 - от регистра к сумматору
 - внутри сумматора
 - от сумматора к регистру
- время требуется на сохранение значения внутри регистра
- Рассмотрим работу счетчика с учетом перечисленных задержек



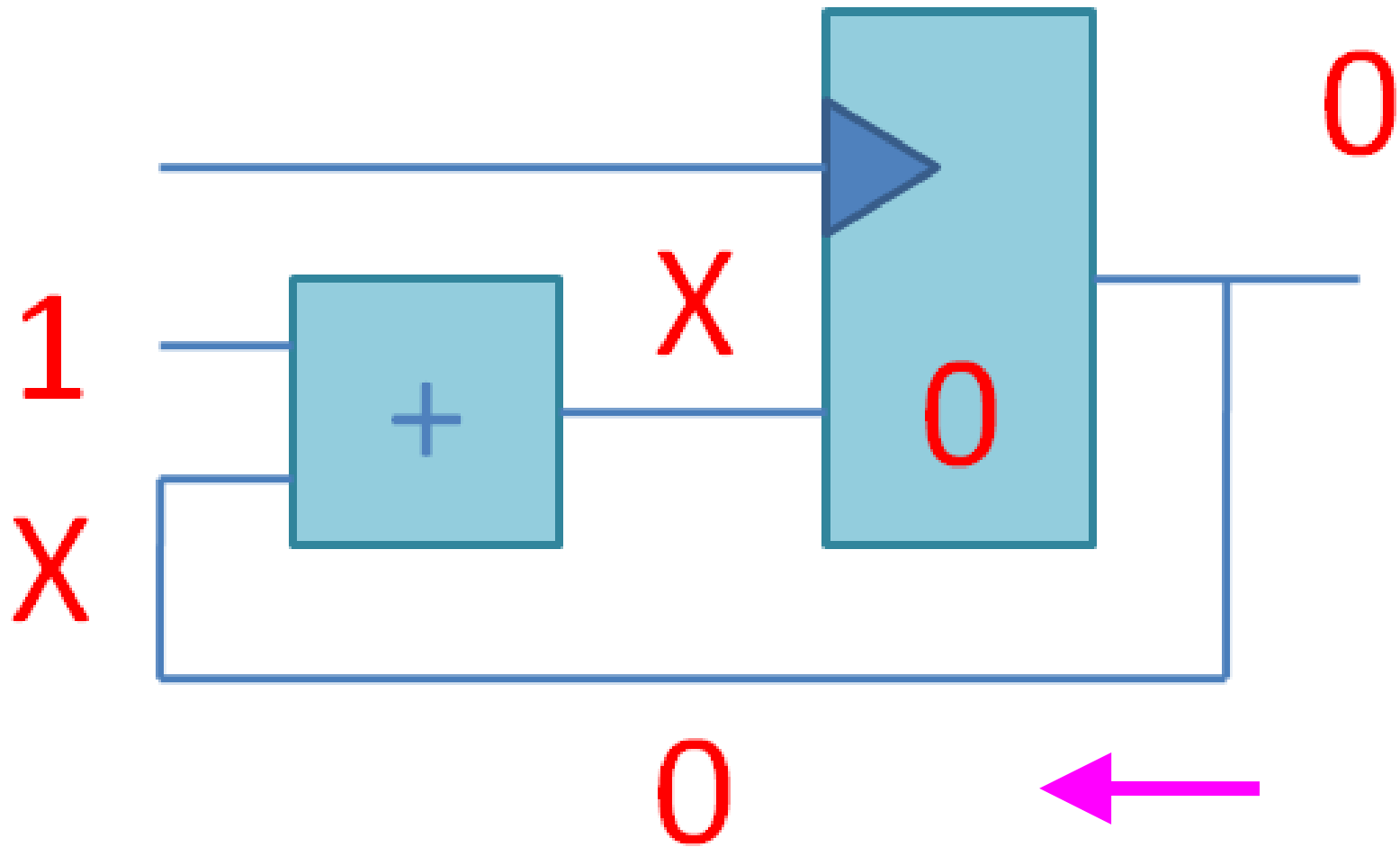
The picture is from Digital Design and Computer Architecture
2nd Edition by David Harris and Sarah Harris. Elsevier, 2012

Счетчик - детально

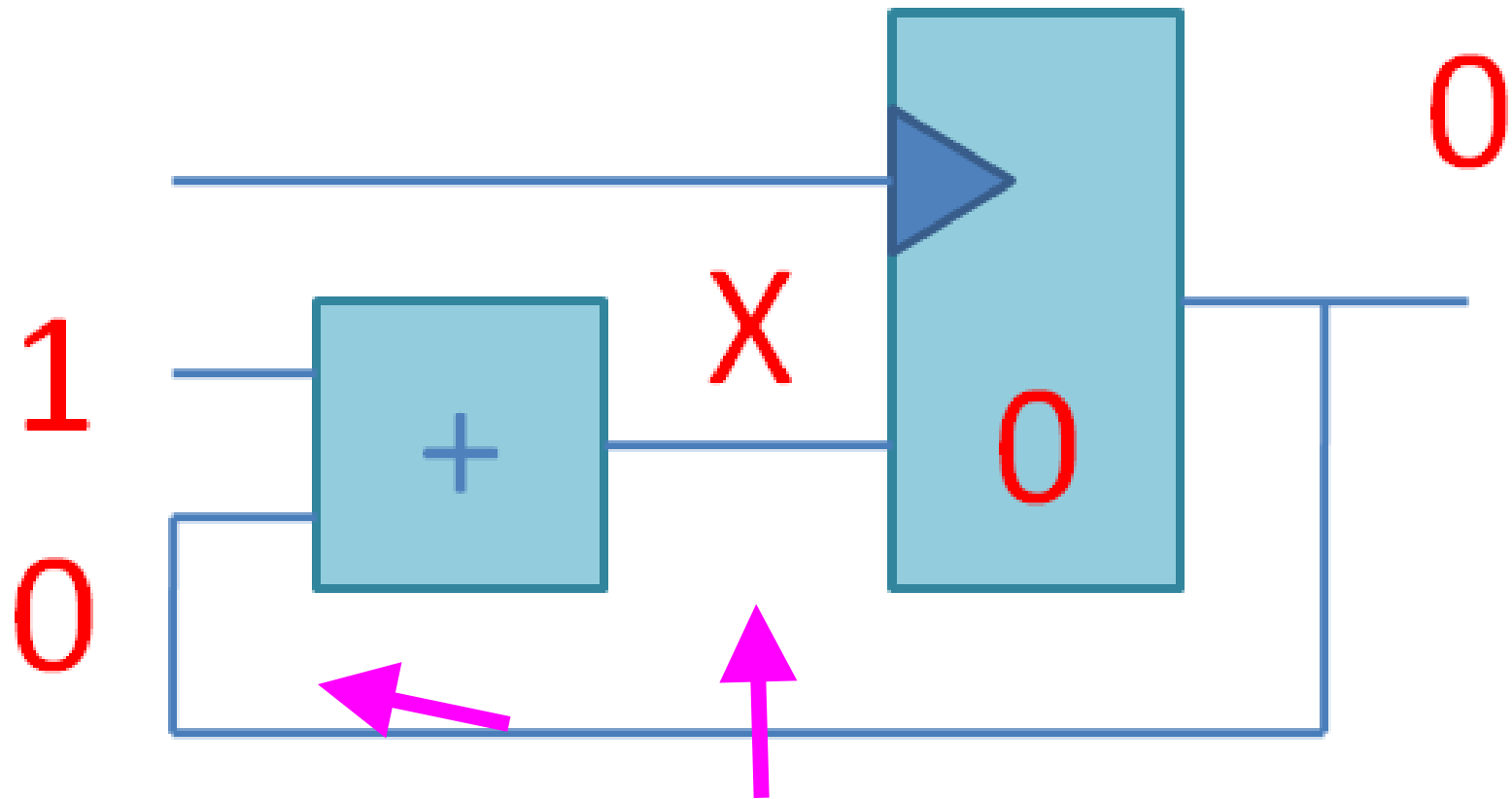


Регистр хранит 0, который распространяется к входу сумматора

Счетчик - детально

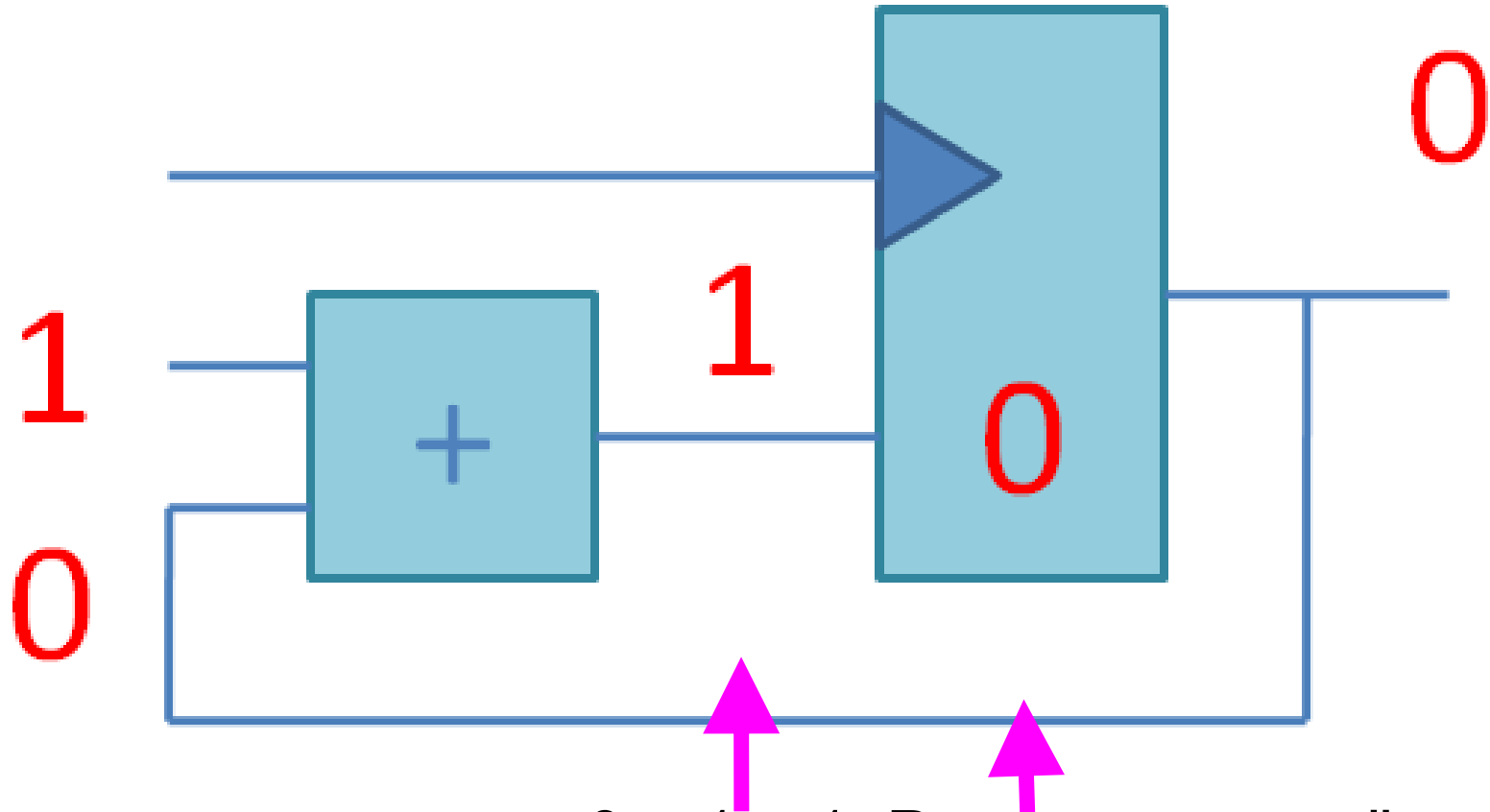


Счетчик - детально



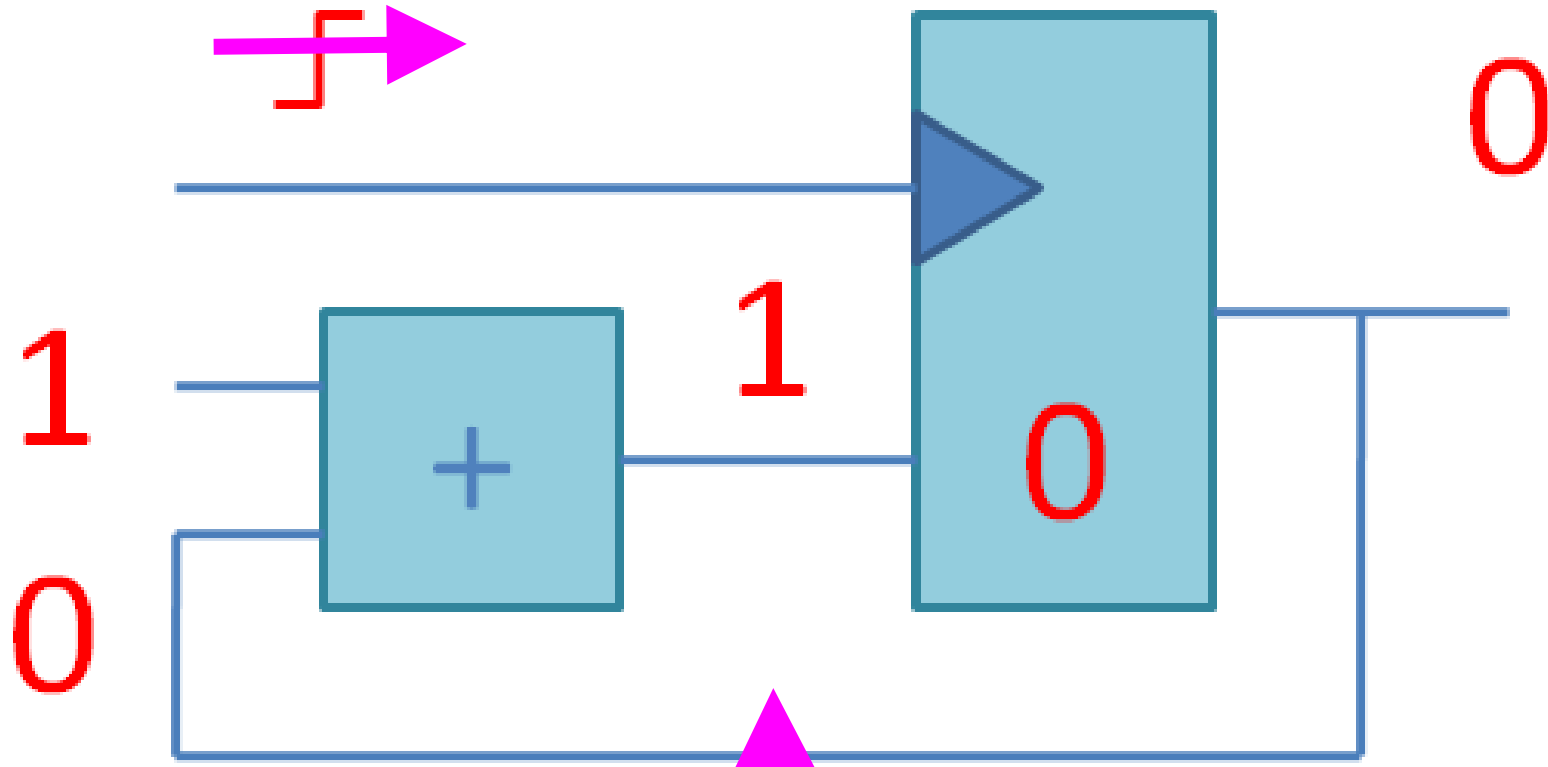
0 поступил в сумматор, но его выход ещё не установился

Счетчик - детально



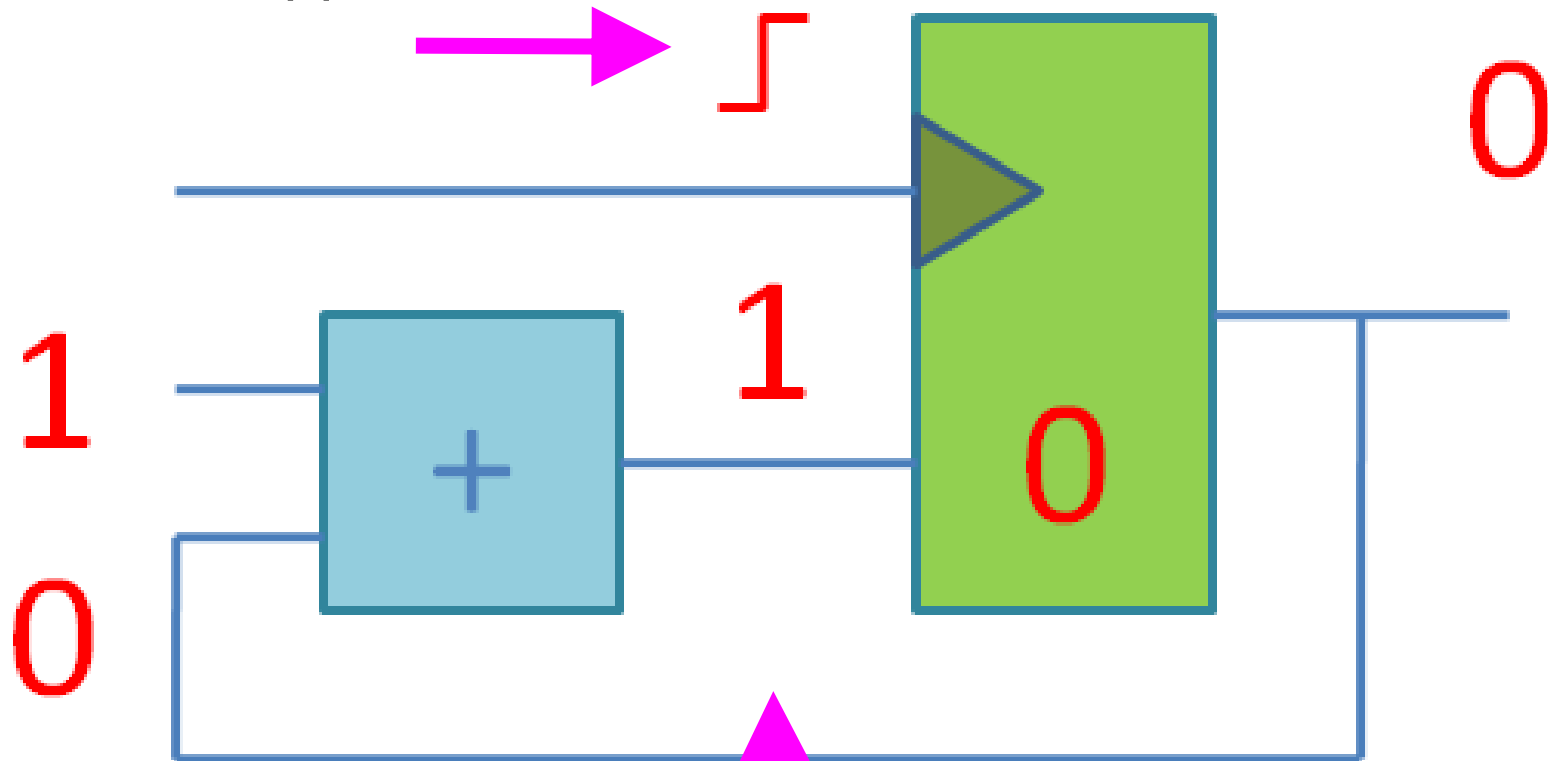
Сумматор вычислил $0 + 1 = 1$. Регистр все ещё хранит 0.

Счетчик - детально



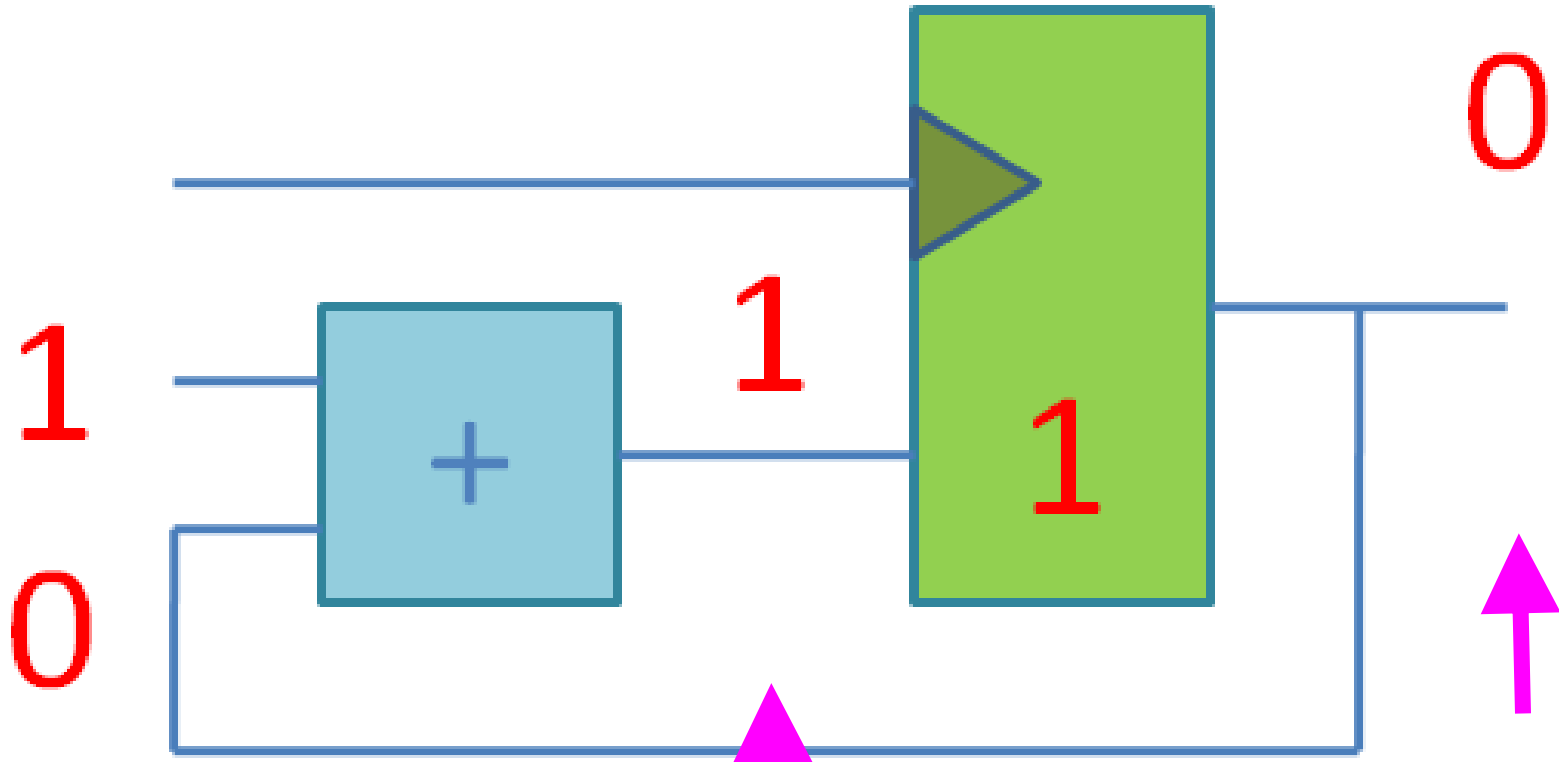
Наступает фронт тактового сигнала. Регистр хранит 0.

Счетчик - детально



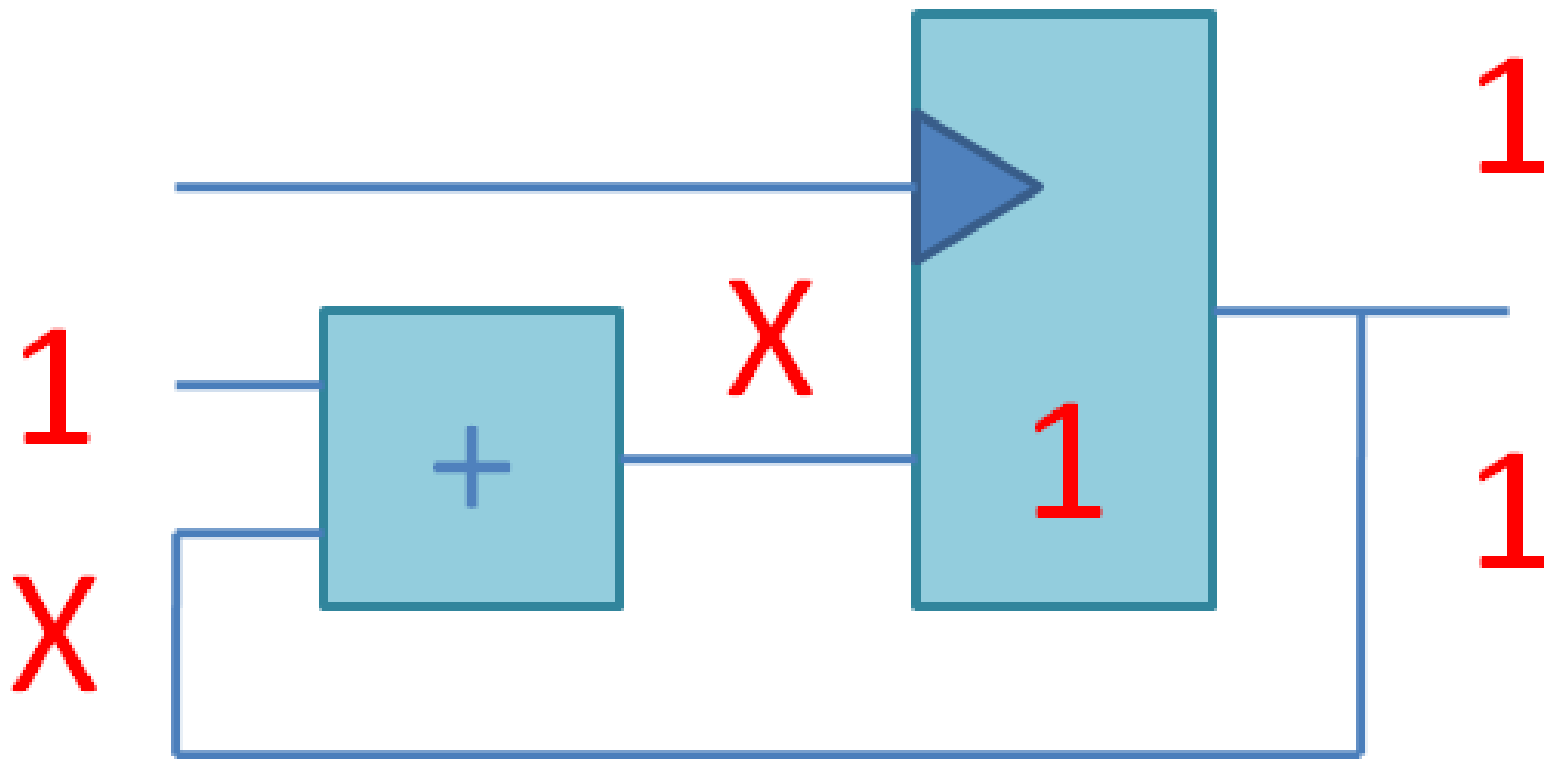
Апертурное время. В регистр сохраняется 1.

Счетчик - детально



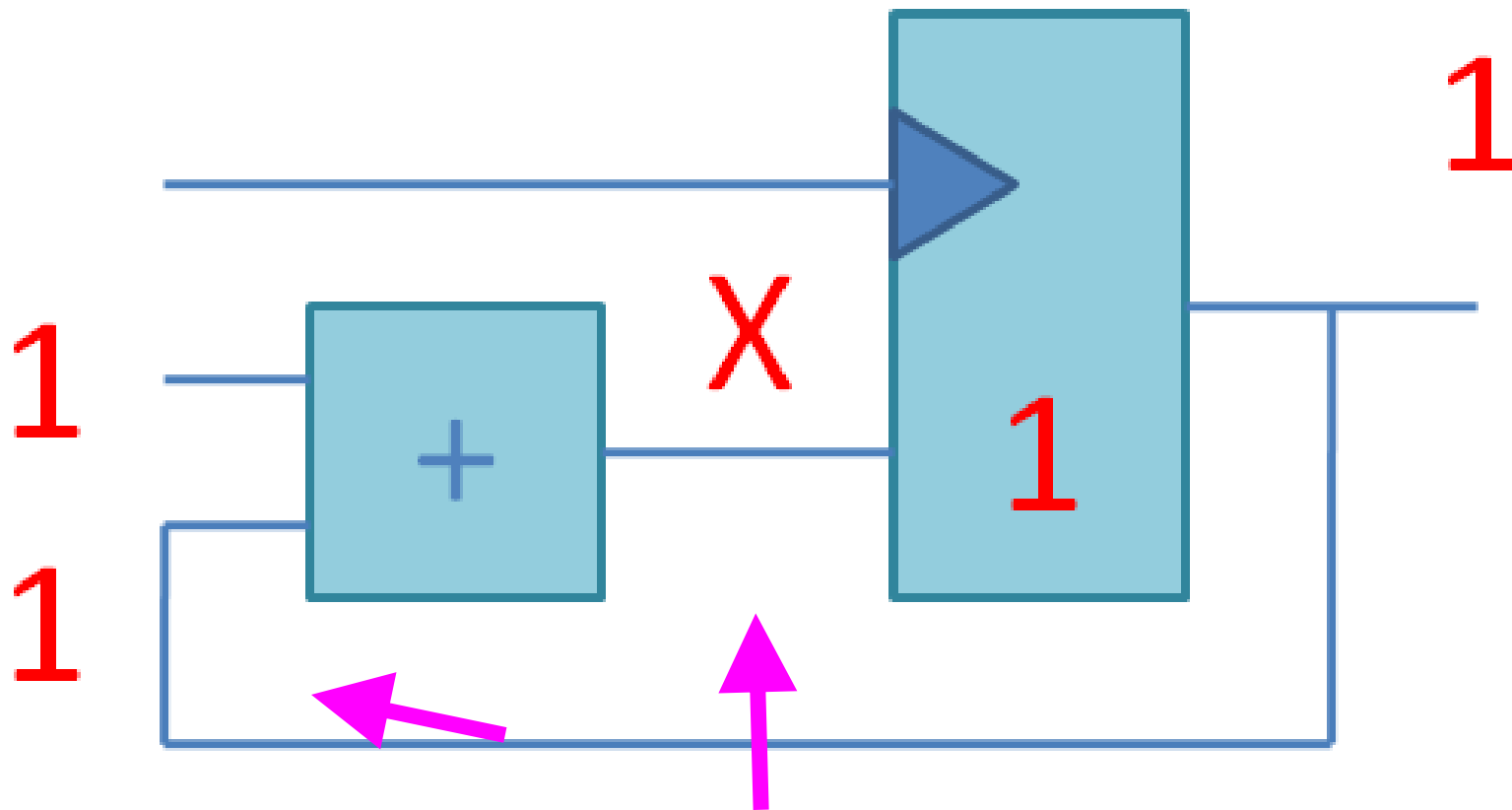
В регистр сохранена 1, это значение поступает на его выход

Счетчик - детально



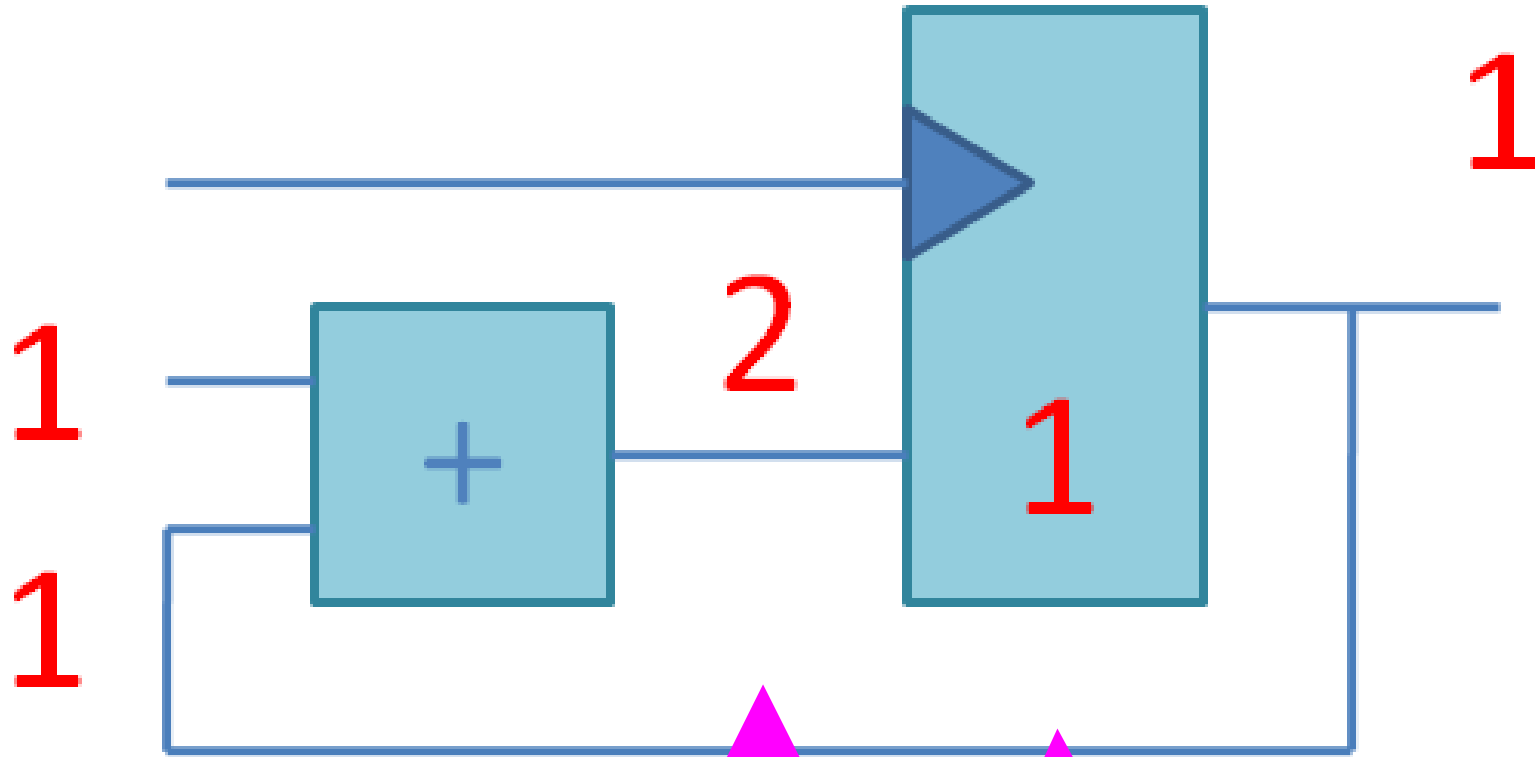
Текущее состояние 1, оно распространится к входу сумматора.

Счетчик - детально



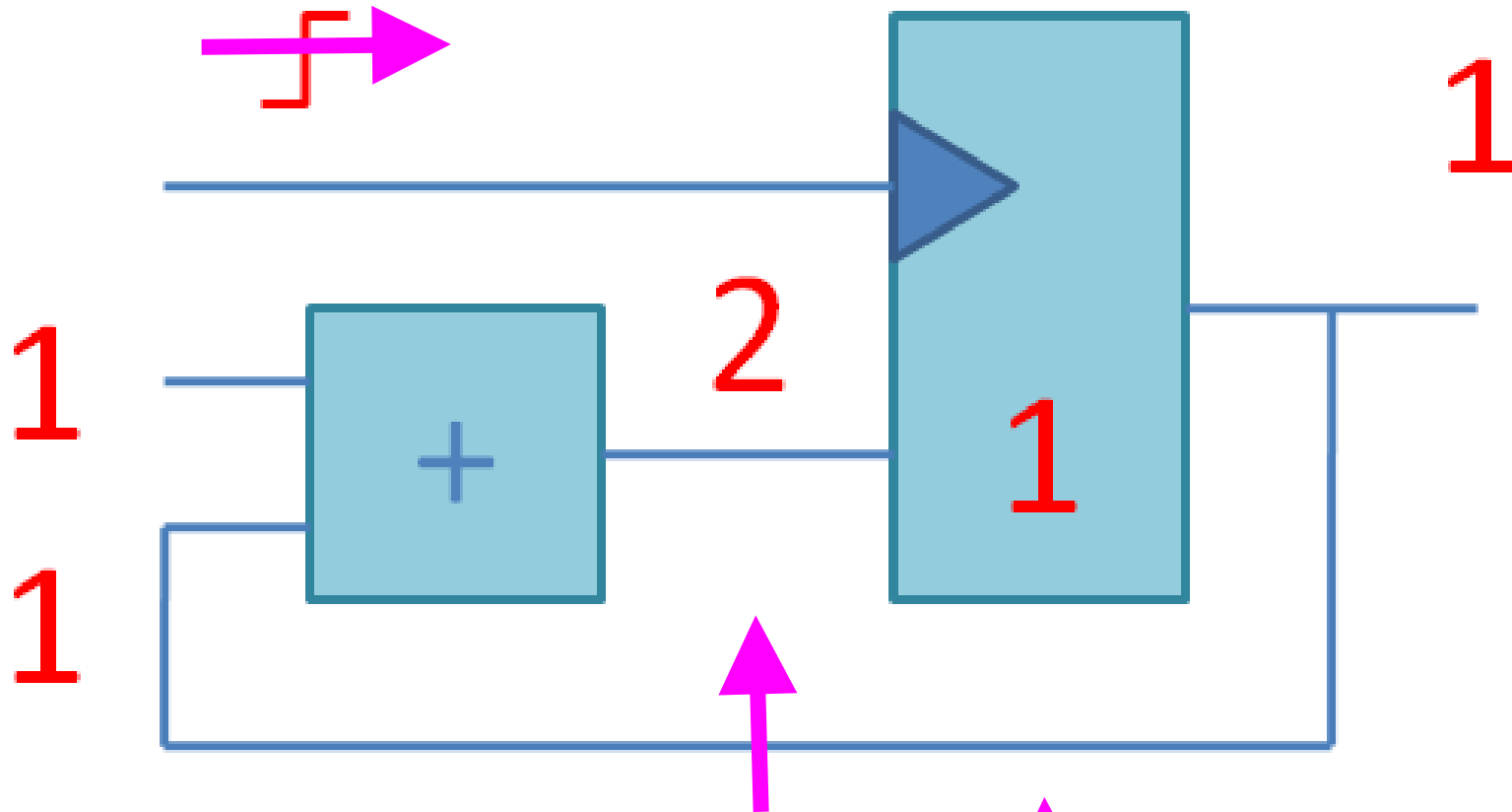
1 поступила на вход сумматора, но его выход еще не стабилен

Счетчик - детально



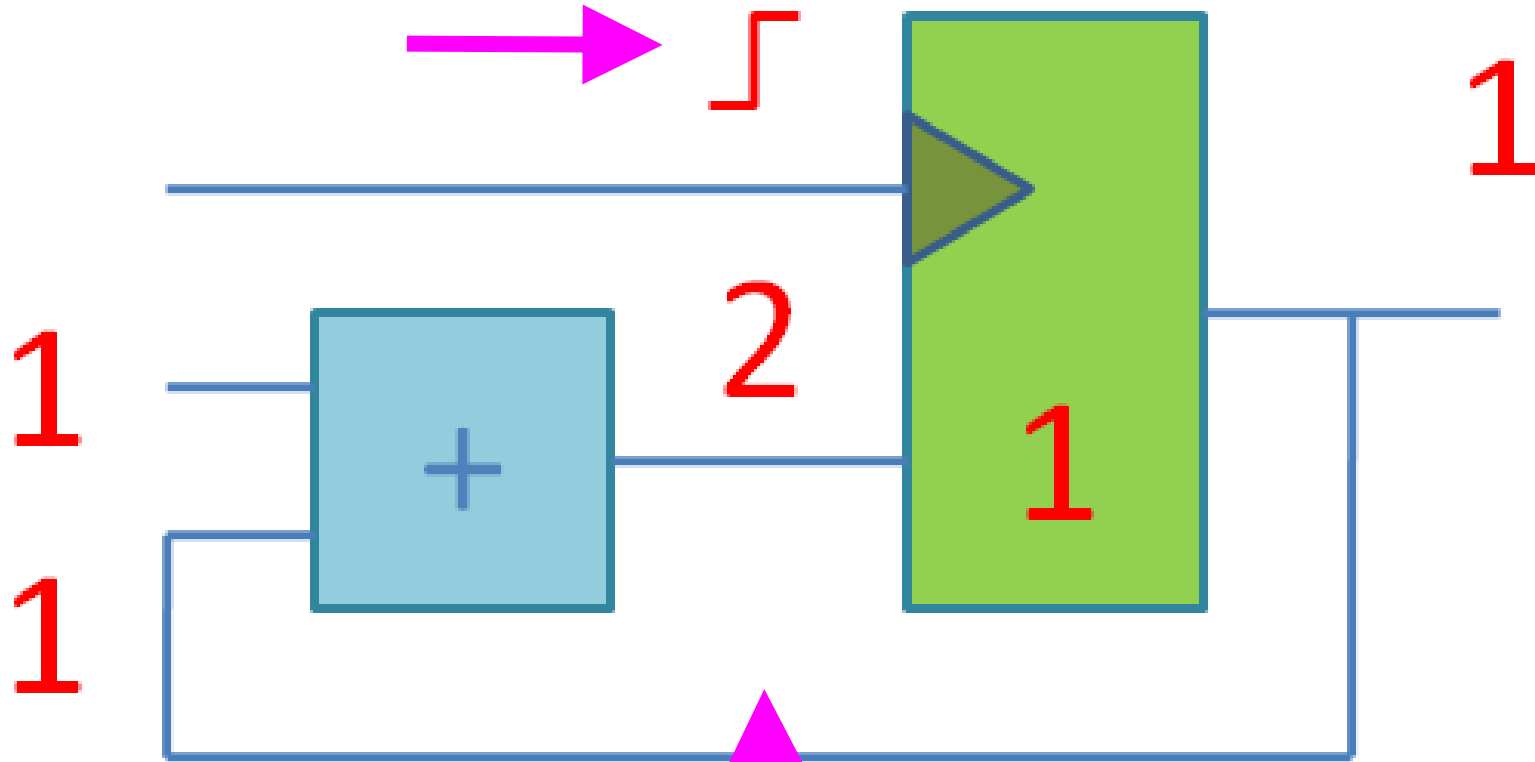
Сумматор вычислил $1 + 1 = 2$. В регистре все еще 1

Счетчик - детально



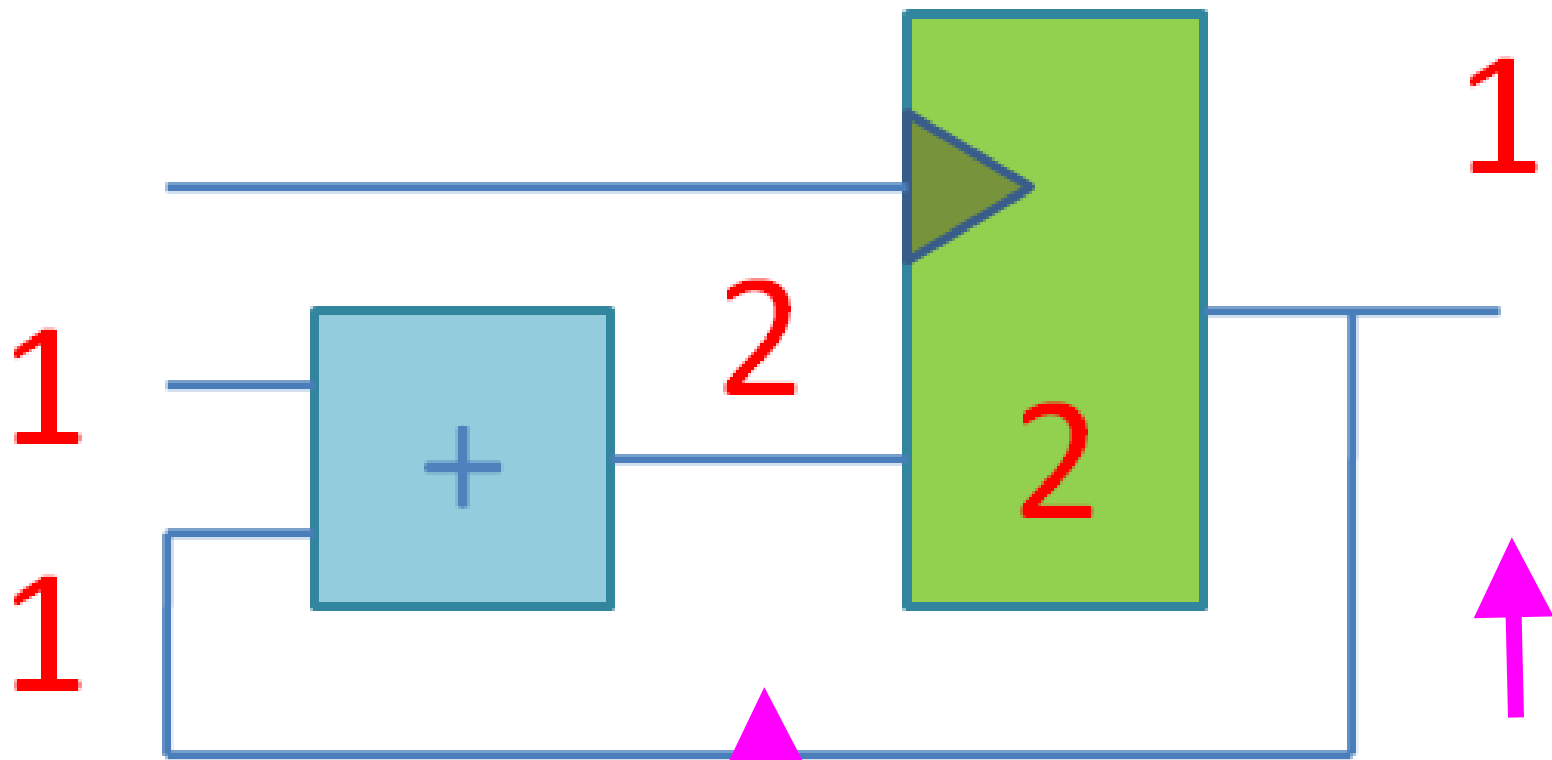
Наступает фронт тактового сигнала. В регистре все еще 1.

Счетчик - детально



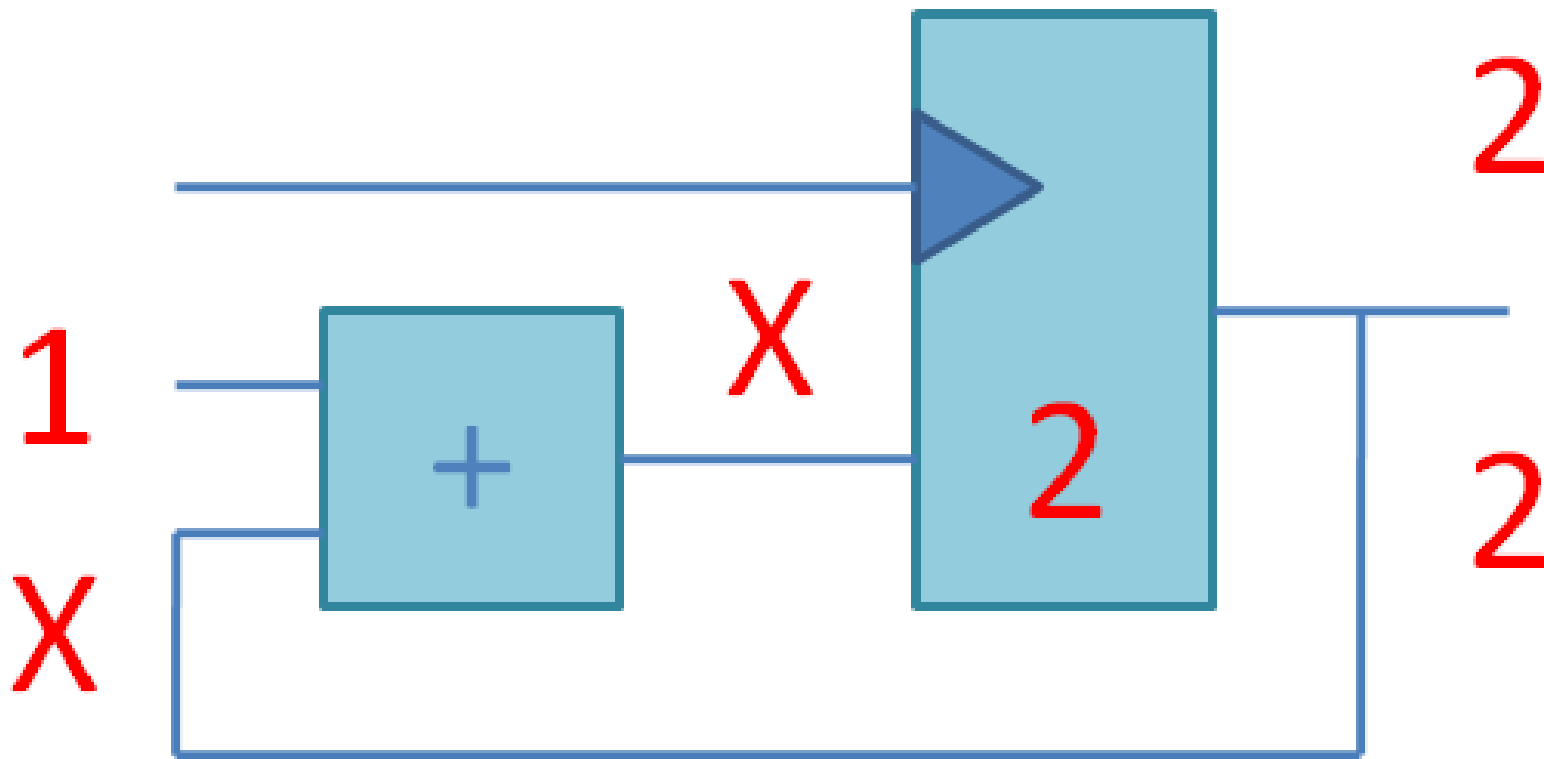
Апертурное время. В регистр сохраняется 2.

Счетчик - детально



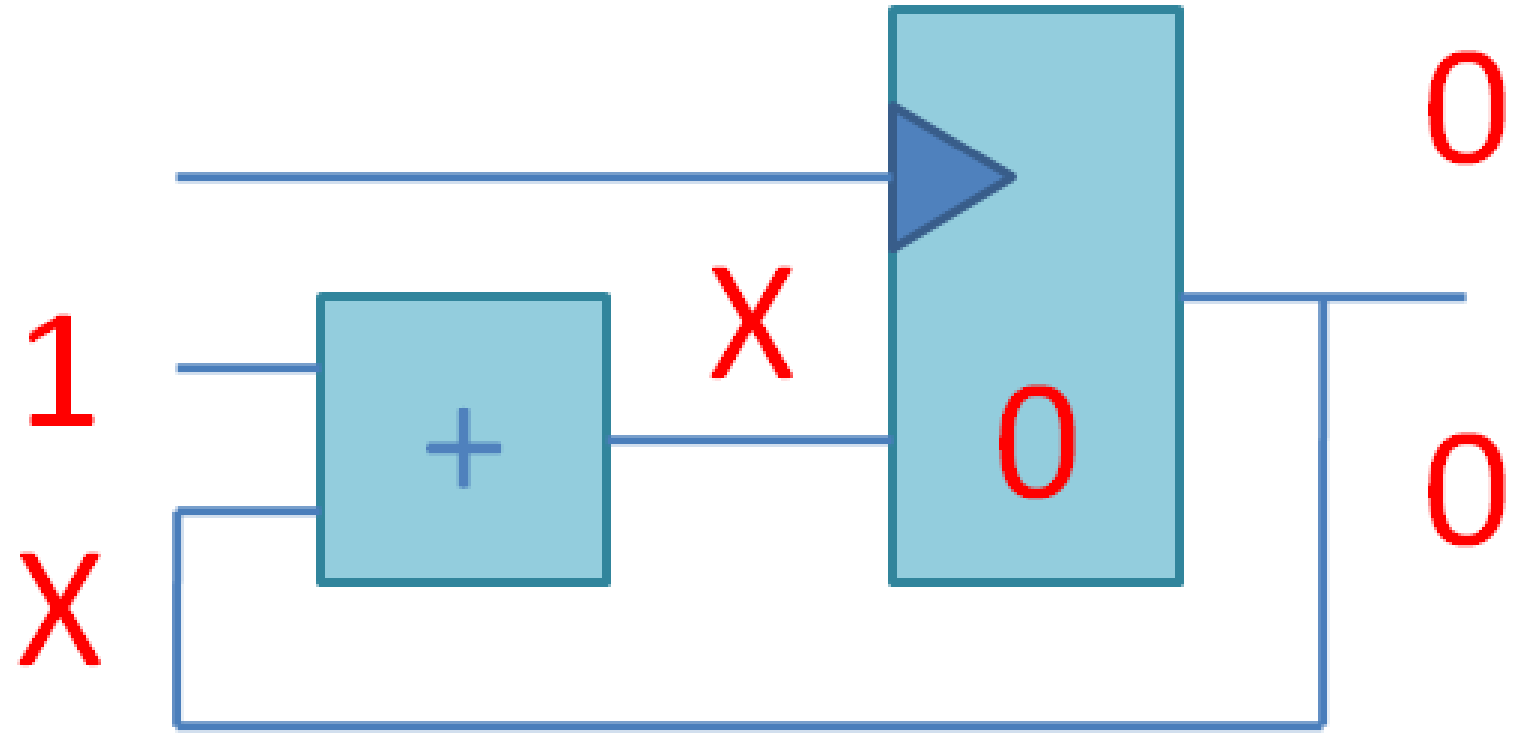
В регистр сохранена 2 и она поступает на его выход.

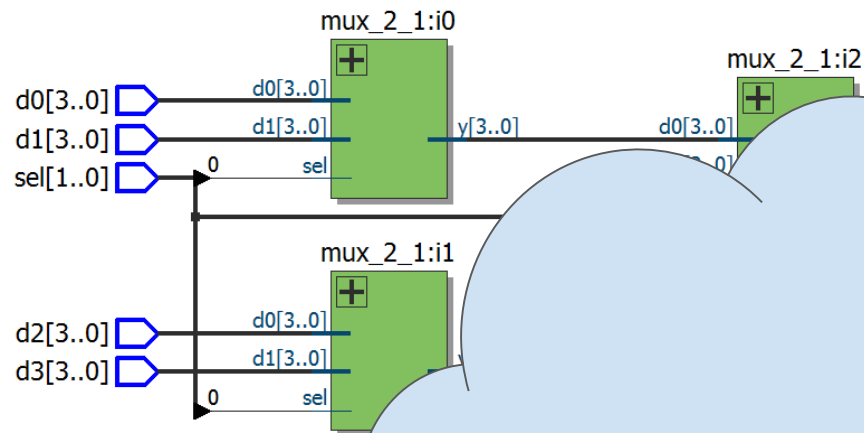
Счетчик - детально



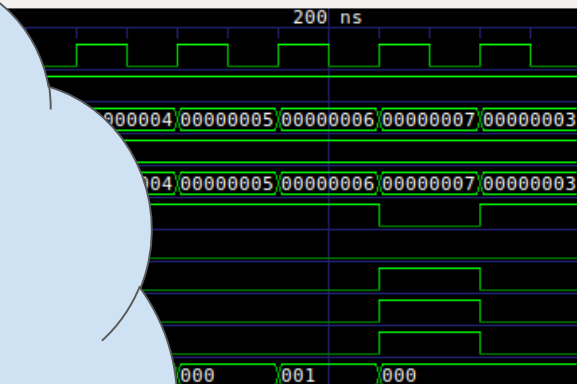
Текущее состояние 2, и оно распространяется к входу сумматора

Счетчик - детально

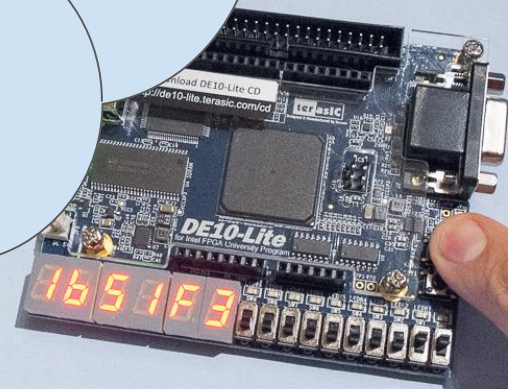
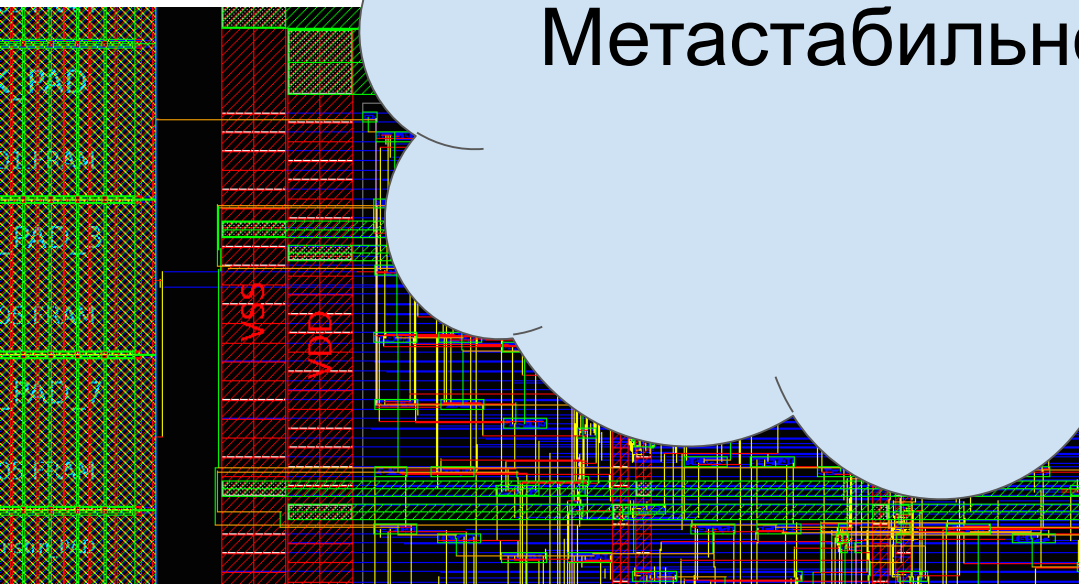




Marker: 670 ns | Cursor: 290500 ps



Метастабильность

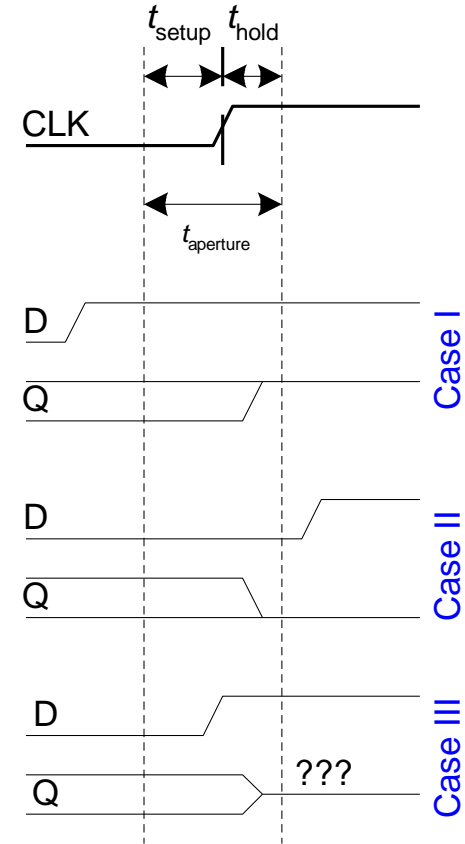


Динамическая дисциплина

- Для того, чтобы триггер смог сохранить значение по фронту тактового сигнала данные на его входе должны быть неизменны:
- в течении **времени установки t_{setup}** до фронта тактового сигнала
- В течении **времени удержания t_{hold}** после фронта тактового сигнала
- **Апертурное время** – все то время, когда сигнал на входе должен быть неизменен

$$t_{\text{aperture}} = t_{\text{setup}} + t_{\text{hold}}$$

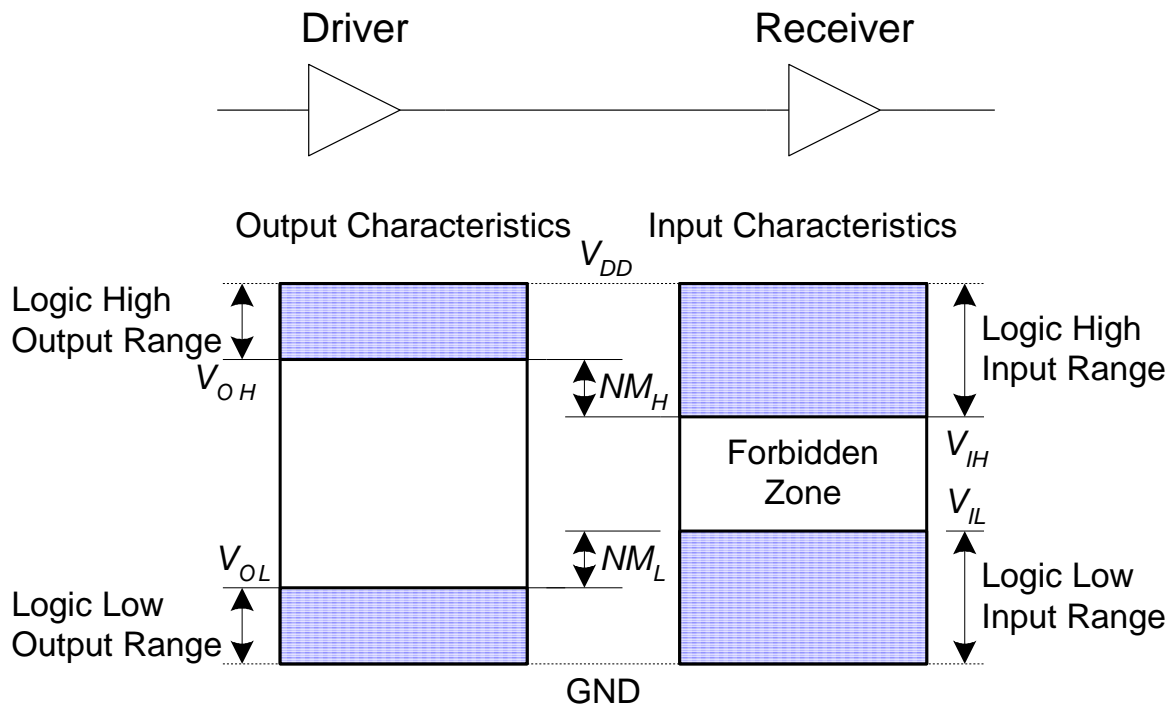
- Если входной сигнал изменился в апертурное время, то выход триггера может быть **нестабильным**



The picture is from Digital Design and Computer Architecture, 2nd Edition by David Harris and Sarah Harris. Elsevier, 2012

Статическая дисциплина

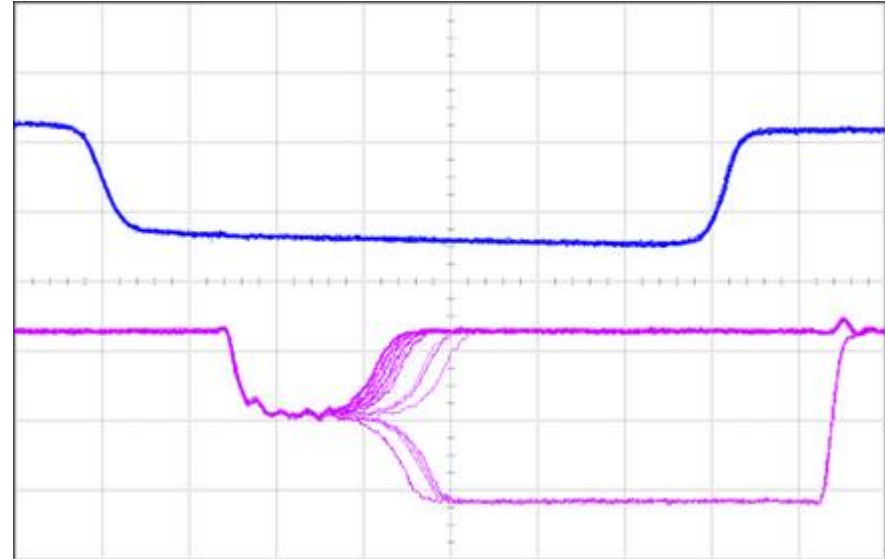
- Напряжение сигнала на входе цифровой схемы должно быть выше V_{IH} или ниже V_{IL} .
- Промежуток между V_{IL} и V_{IH} называется запрещенной зоной
- Если в апертурное время сигнал на входе триггера находится в запрещенной зоне, то выход триггера будет нестабильным



The picture is from Digital Design and Computer Architecture, 2nd Edition by David Harris and Sarah Harris. Elsevier, 2012

Метастабильность

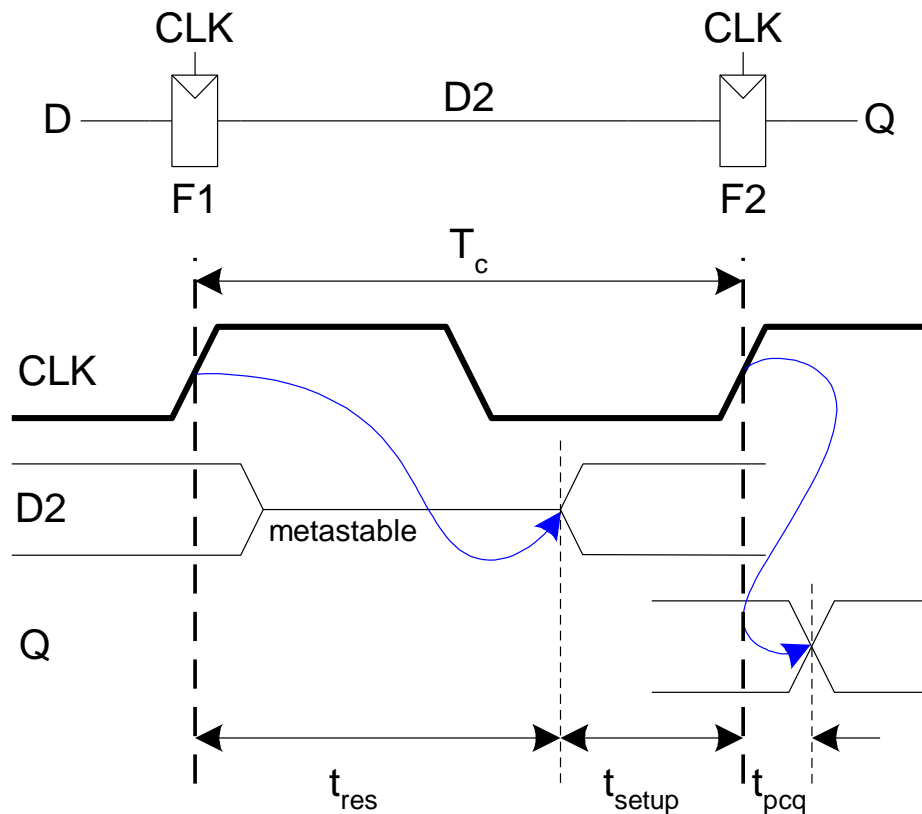
- Если вход комбинационной схемы находится запрещенной зоне, то ее выходы также могут быть в запрещенной зоне
- Если вход триггера в апертурное время находится в запрещенной зоне, то и выход триггера *некоторое время* может находиться в запрещенной зоне
- Такое состояние схемы называется **метастабильностью**,
- к нему приводит нарушение статической или динамической дисциплины
- может распространиться дальше по схеме



The picture is from <http://www.pvsm.ru>

Синхронизатор

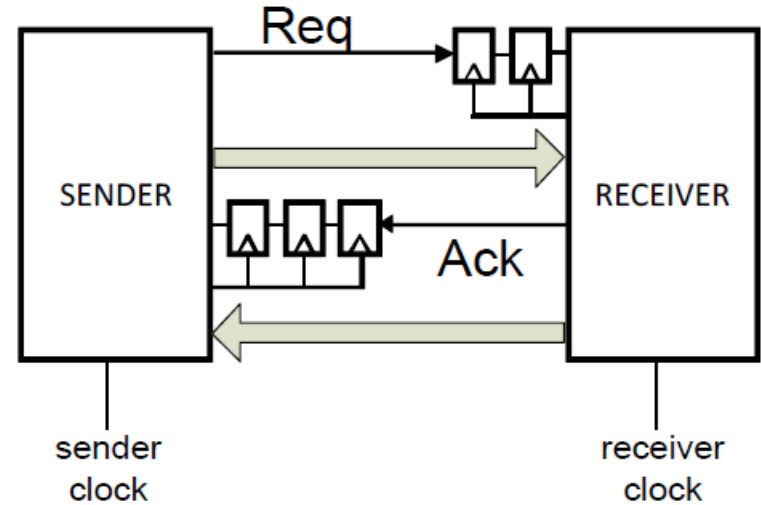
- Наличие асинхронных входов чипа неизбежно (кнопки, сигналы от других устройств и т.д.)
- Для уменьшения вероятности сбоя из-за метастабильности используют синхронизаторы
- Синхронизатор представляет собой 2-3 последовательно соединенных регистра
- С рассчитываемой вероятностью спасает от нарушений *динамической* дисциплины
- Может использоваться только для одиночных/независимых сигналов



The picture is from Digital Design and Computer Architecture, 2nd Edition by David Harris and Sarah Harris. Elsevier, 2012

Методики синхронизации

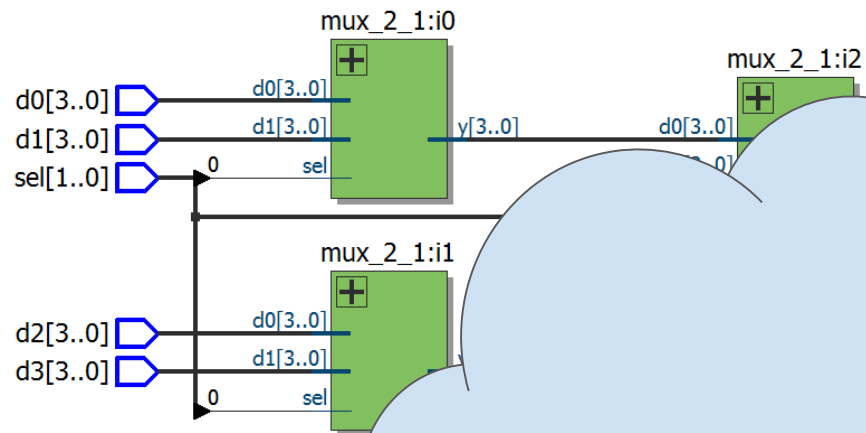
- В ряде случаев синхронизация необходима не только при считывании информации из внешнего мира, но и при обмене данными внутри микросхемы. Например, когда блоки одного чипа работают на разных частотах
- Для этого используется набор методик под названием **Clock Domain Crossing**
- Рассмотренный ранее одиночный синхронизатор на 2х регистрах не может быть использован для синхронизации обмена данными по шинам шириной более 1 бита
- В таких случаях используют более сложные схемы
- А также применяют особые способы кодирования счетчиков: **код Грея, код Джонсона**



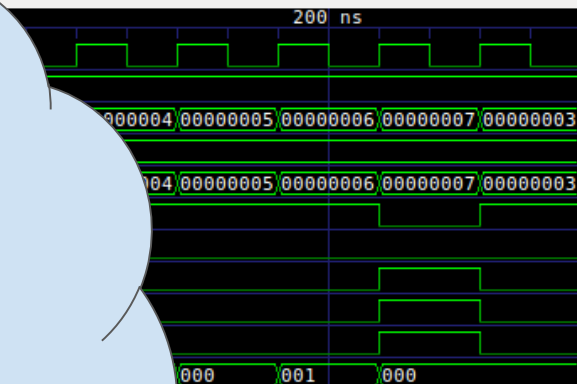
The picture is from Metastability and Synchronizers
A Tutorial, by Ran Ginosar, IEEE 2011

На что похожа метастабильность

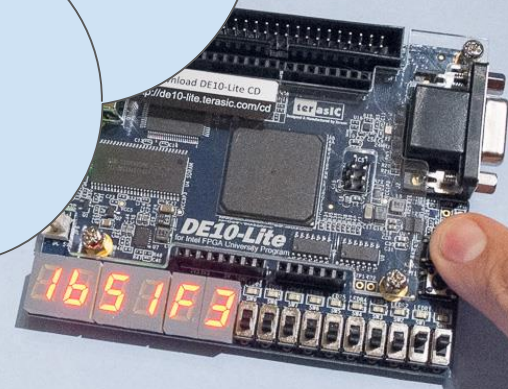
- Откройте файл **ddec/lab/27_seq_synchronizers/rtl/synchronizer.v**
- Откройте консоль в каталоге **ddec/lab/27_seq_synchronizers**
- Выполните симуляцию примера: **make sim**
- Выполните синтез и загрузку примера на плату:
make synth
make load
make open
- Переключение **SW[1]** приводит к переключению **LED[1:0]**
- Подключите провод к контакту **GPIO[0]**, второй конец провода оставьте неподключенным - «висящим» в воздухе. Наводки, возникающие в этой импровизированной антенне, достаточны для изменения поведения схемы
- Подобные симптомы очень похожи на наблюдаемые при метастабильности: «мусор» на выходах схемы, сбой на индикаторах



Marker: 670 ns | Cursor: 290500 ps

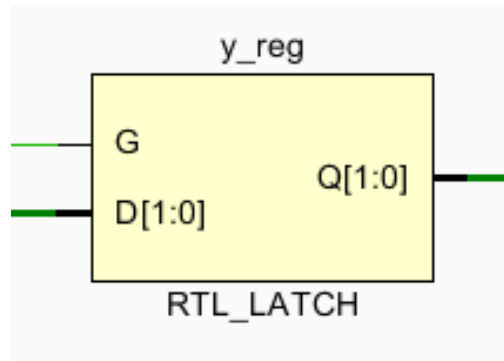


Защелка



Защелка - Latch

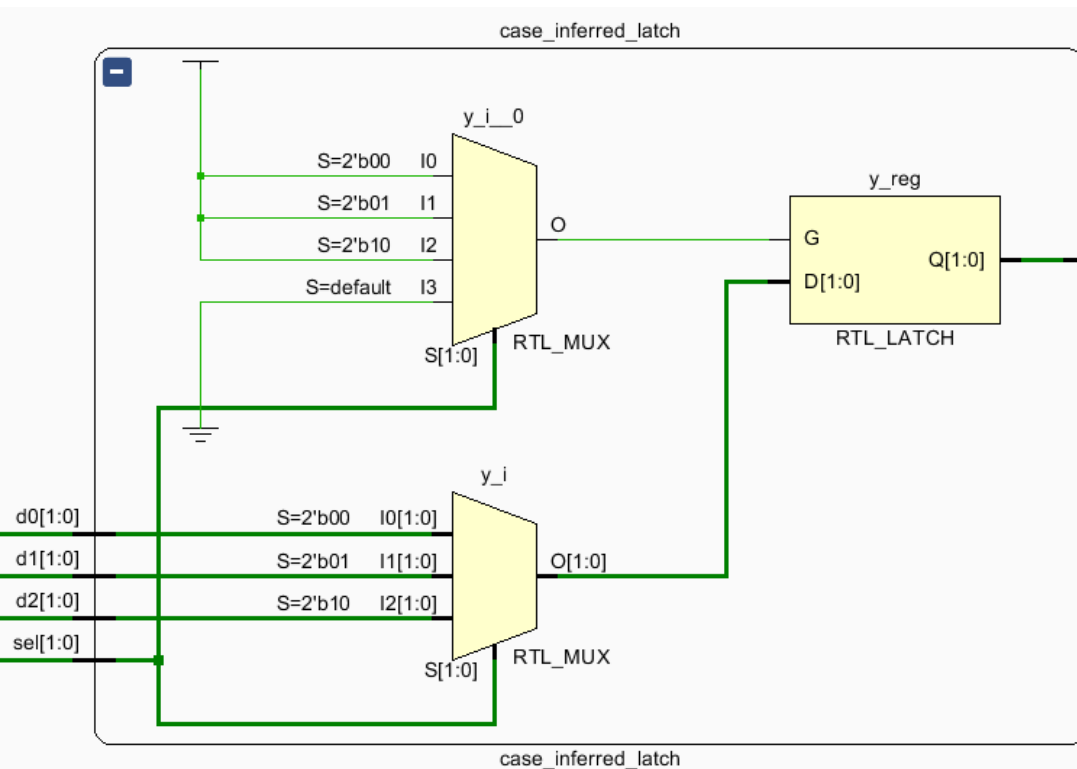
- Если на разрешающем входе **G** установлен высокий уровень сигнала, то на выходе **Q** установлен такой же уровень, как на входе **D**
- Если **G** находится в низком уровне, то значение на выходе **Q** не изменяется
- Защелки автоматически добавляются в дизайн в тех случаях, когда в комбинационной схеме не для всех сочетаний входных сигналов, предусмотрены соответствующие выходные значения (например, используется блок **case** без **default**)
- Автоматически добавленные в дизайн защелки оказывают негативное влияние на максимальную частоту, с которой может работать схема



G	Q
1	D
0	Qprev

Защелка – может быть добавлена автоматически

- Откройте файл [ddec/lab/28_latch/rtl/inferred_latch.v](#)
- Значение **Y** при **sel=2'b11** не предусмотрено, это приводит к синтезу защелки

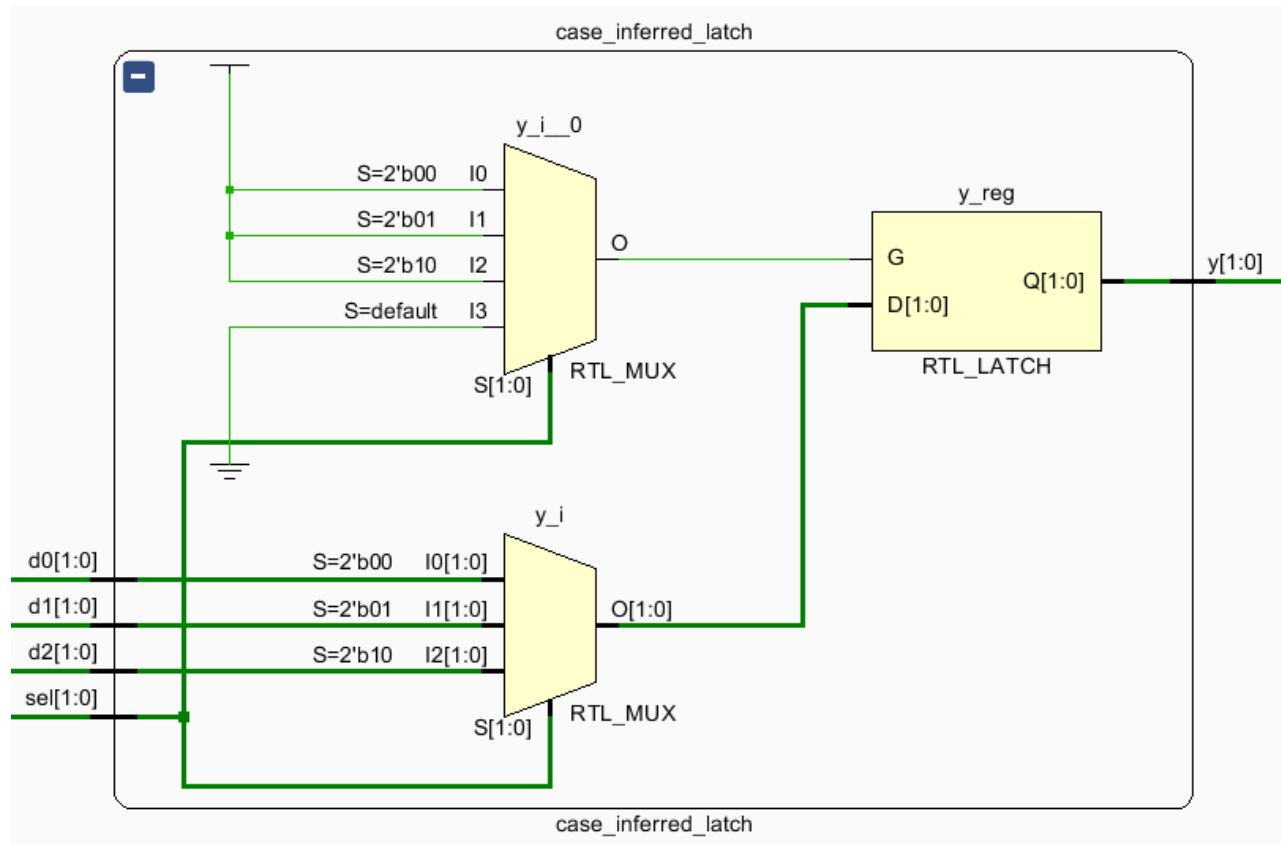


```
module case_inferred_latch
(
    input    [1:0] d0, d1, d2,
    input    [1:0] sel,
    output reg [1:0] y
);

// the same result can be obtained
// when 'if' is used without 'else'
always @(*)
    case (sel)
        2'b00: y = d0;
        2'b01: y = d1;
        2'b10: y = d2;
    endcase
```

Защелка – возможны ошибки

- Данная схема является комбинационной - на вход защелки не поступает тактовый сигнал
- Возможна ситуация, когда запрет записи (**G=0**) поступит на вход защелки позже, чем начнут меняться данные на входе **D**
- В этом случае в защелке будет сохранено некорректное значение

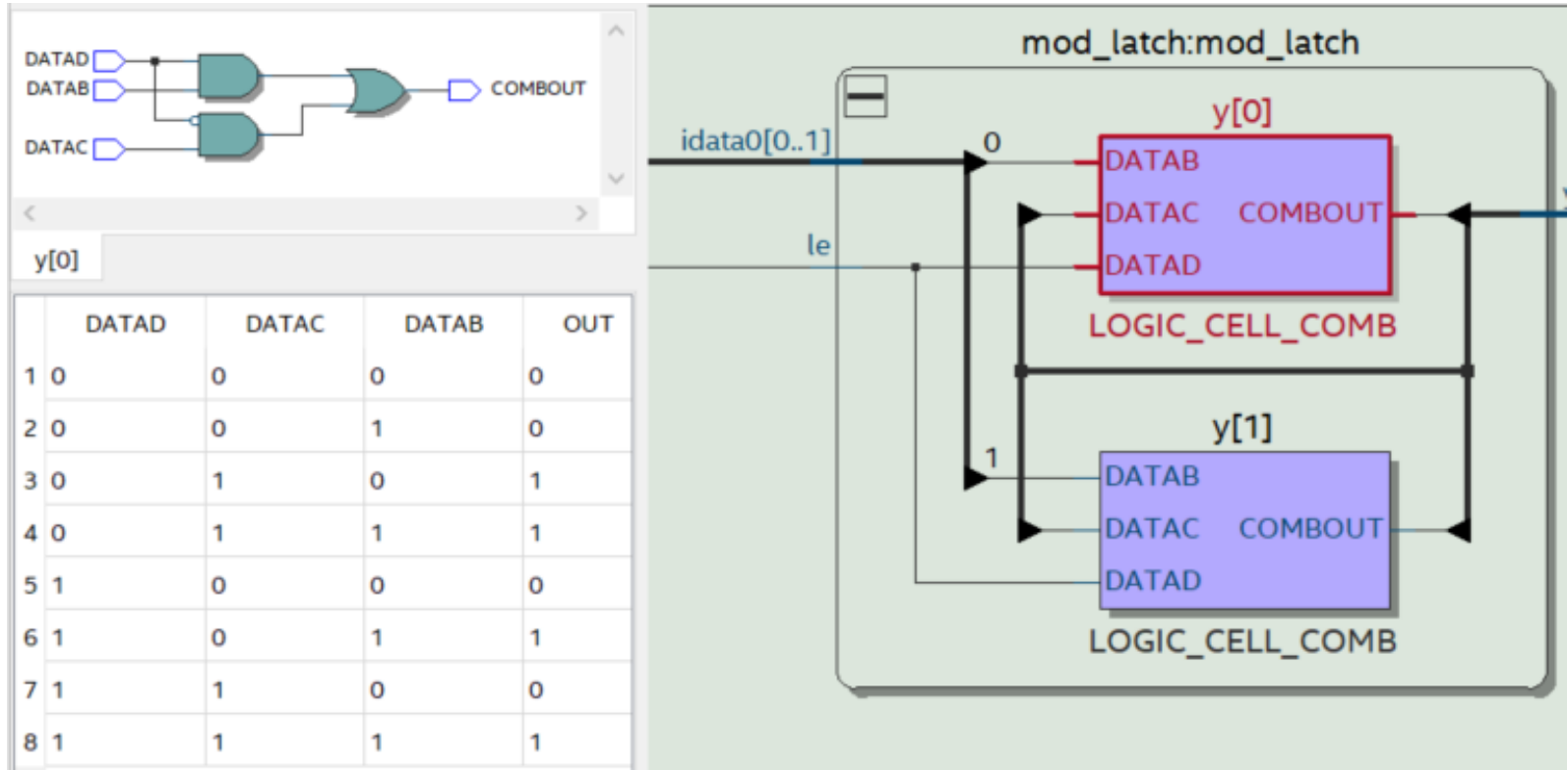


Защелка – практика (1 / 2)

- Откройте файл **ddec/lab/28_latch/rtl/inferred_latch.v**
- Откройте консоль в каталоге **ddec/lab/28_latch**
- Выполните симуляцию примера: **make sim**
- Выполните синтез проекта для плат **Nexys 4 DDR, DE10-Lite, DE0-CV**
- Выполните программирование отладочных плат
- Сравните работы защелки на разных платах – везде ли она ведет себя одинаково?
- Откройте проект в режимах:
RTL-Viewer и **Technology Map Viewer** для Intel Quartus
Elaborated Design и **Implemented Design – Schematic** для Xilinx Vivado

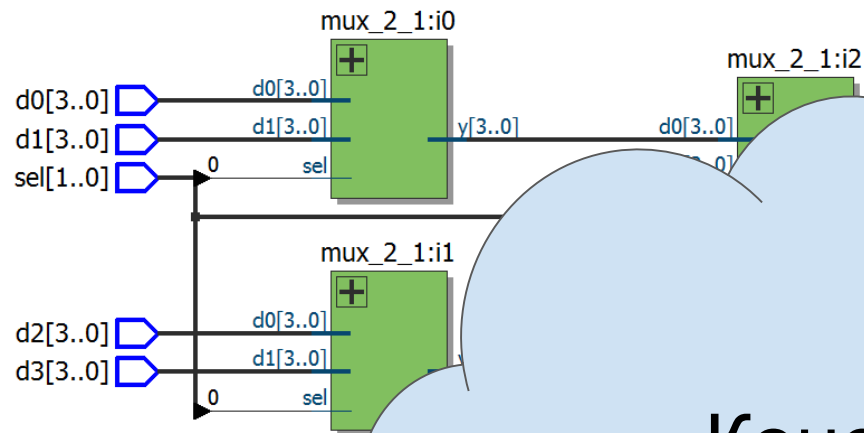
Защелка – практика (2 / 2)

- Обратите внимание на имплементацию защелки в Quartus: реализованный на LUT мультиплексор, выход которого подключен к одному из его входов

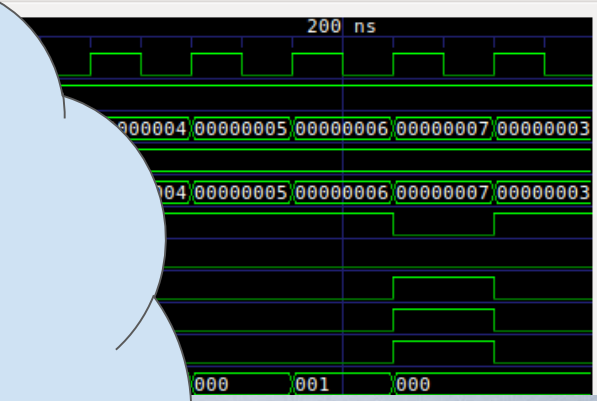


Защелка - итоги

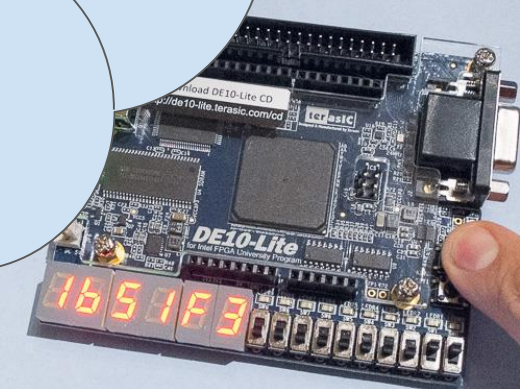
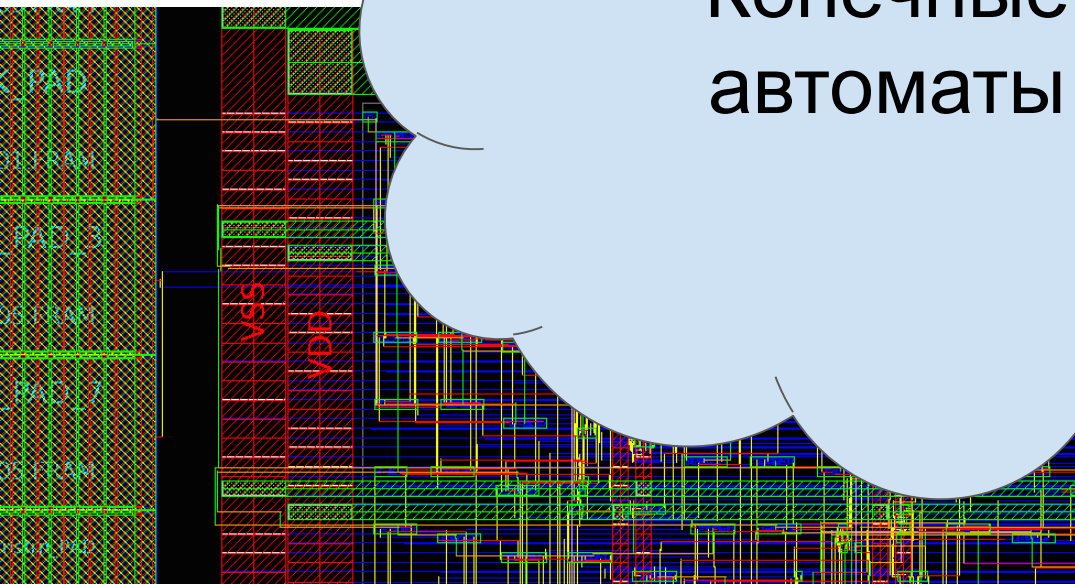
- При описании комбинационных схем для каждого из возможных сочетаний входных сигналов должны быть предусмотрены соответствующие значения выходных сигналов
- Не используйте защелки в проектах на FPGA
- Примеры использования «полезных» защелок (ASIC):
 - **Latch based clock gating**
 - **Muller C-element**



Marker: 670 ns | Cursor: 290500 ps

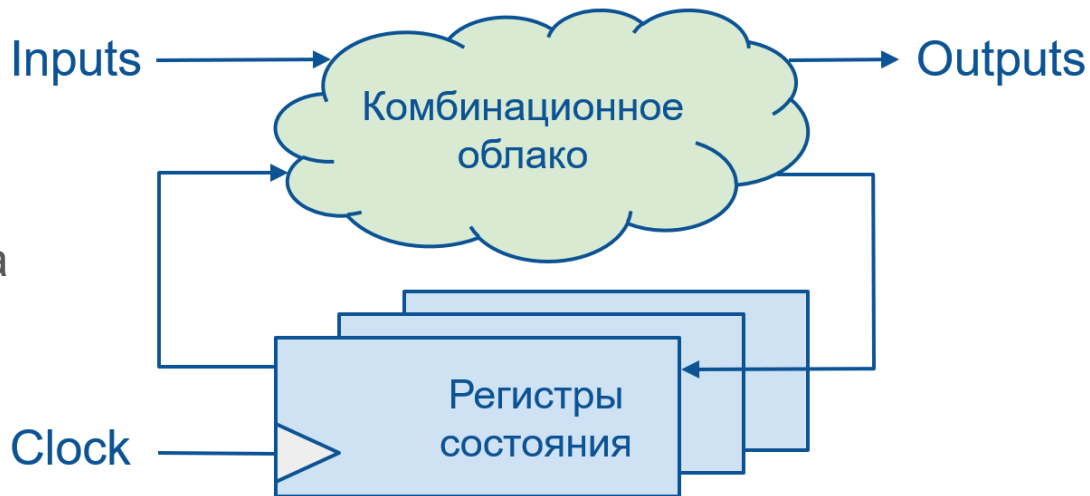


Конечные
автоматы



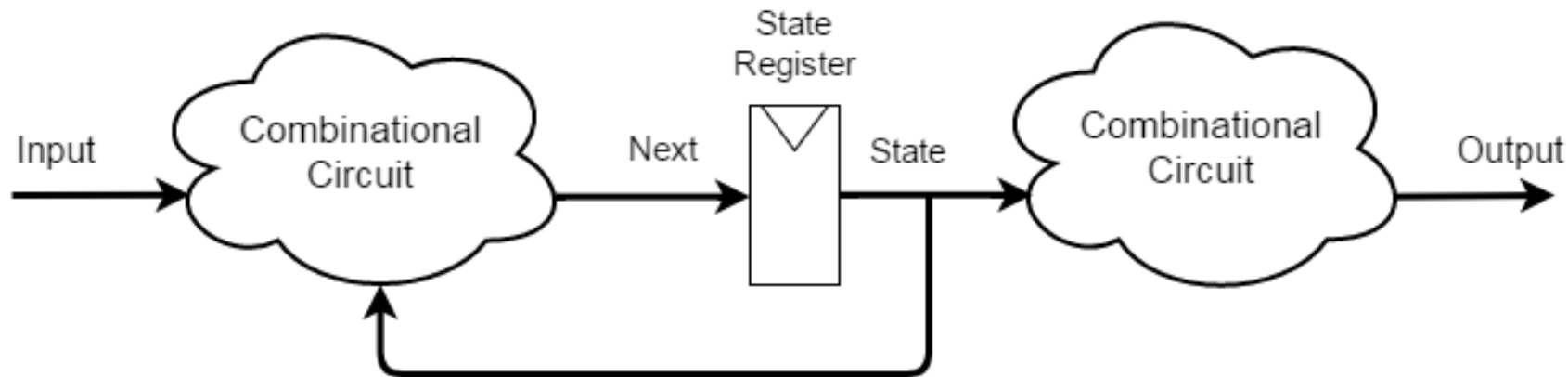
Конечный автомат, принятие решений

- **Автомат, FSM - Finite State Machine**
- Мощная абстракция, помогающая управлять сложностью проектируемых систем
- У каждого автомата есть конечный набор состояний
- Текущее состояние автомата хранится в регистре(ах) состояния
- Следующее состояние автомата зависит от его текущего состояния и сигналов на его входе
- Сигналы на выходе автомата зависят от текущего состояния и, в ряде случаев, от входных сигналов



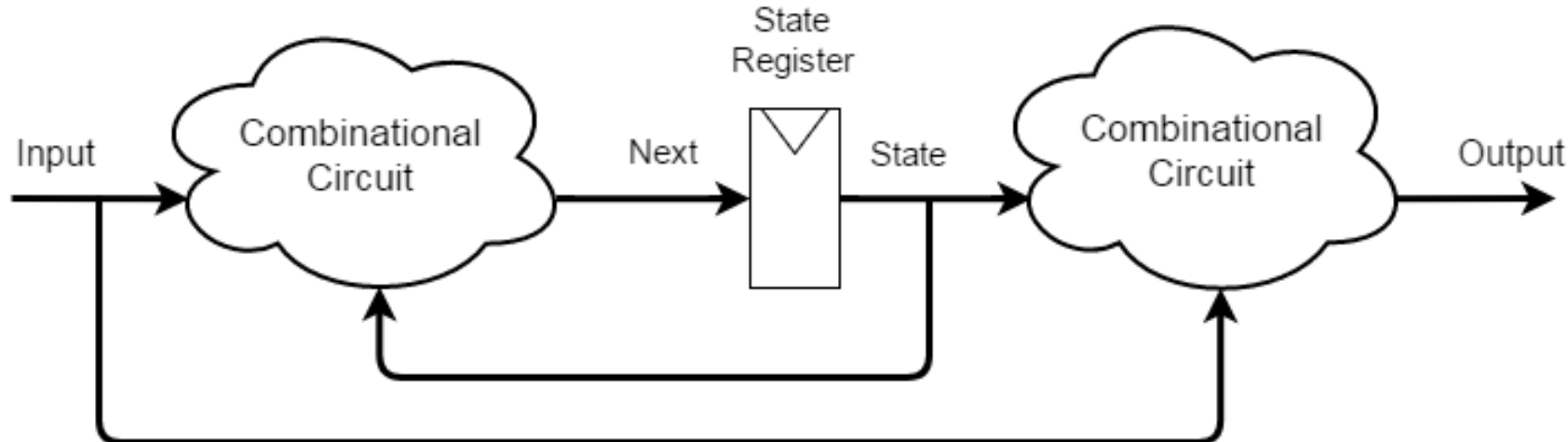
Автомат Мура

- Текущее состояние (**State**) хранится в регистре состояния (**State Register**)
- Следующее состояние (**Next**) зависит от текущего, а также от сигналов на входе автомата (**Input**)
- Сигналы на выходе конечного автомата (**Output**) зависят **только** от текущего состояния



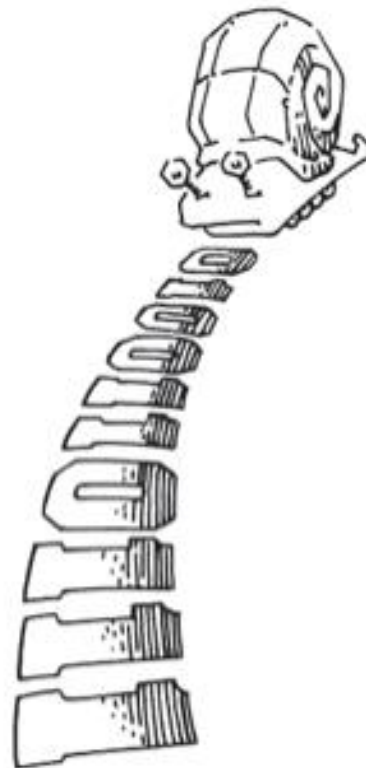
Автомат Мили

- Текущее состояние (**State**) хранится в регистре состояния (**State Register**)
- Следующее состояние (**Next**) зависит от текущего, а также от сигналов на входе автомата (**Input**)
- Сигналы на выходе конечного автомата (**Output**) зависят от текущего состояния, **а также от сигналов на его входе**



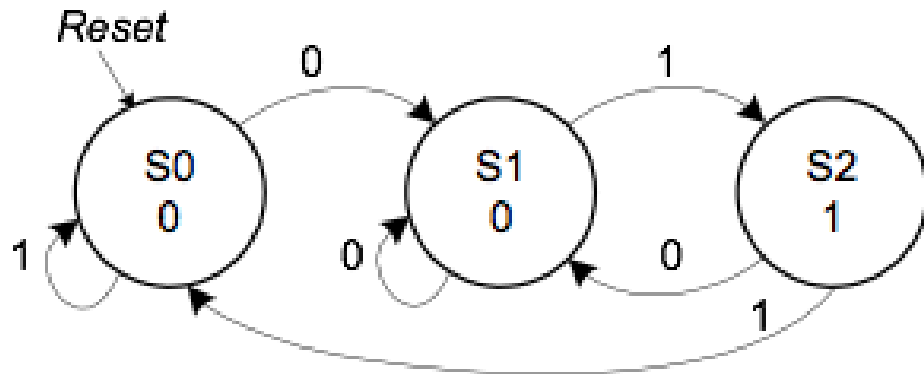
Конечный автомат, принятие решений

- Рассмотрим пример автомата, распознающего последовательности
- Этот пример приведён в книге "Цифровая схемотехника и архитектура компьютера", Дэвид М. Хэррис и Сара Л. Хэррис, 2013
- "Улитка ползёт по полоске бумаги, на которую нанесены нули и единицы. Улитка улыбается каждый раз, когда последние 2 знака, по которым она проползла, это 01. Сконструируйте конечный автомат, который в голове у улитки»

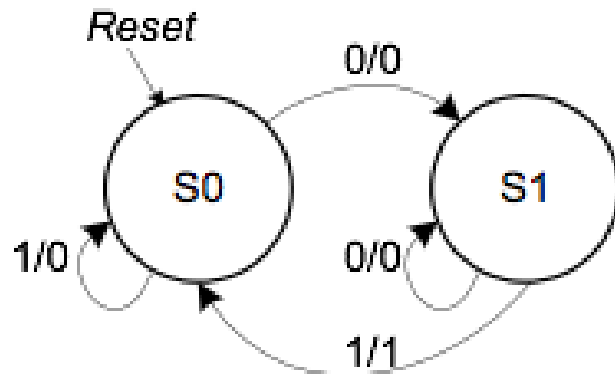


Диаграммы конечных автоматов

Автомат Мура



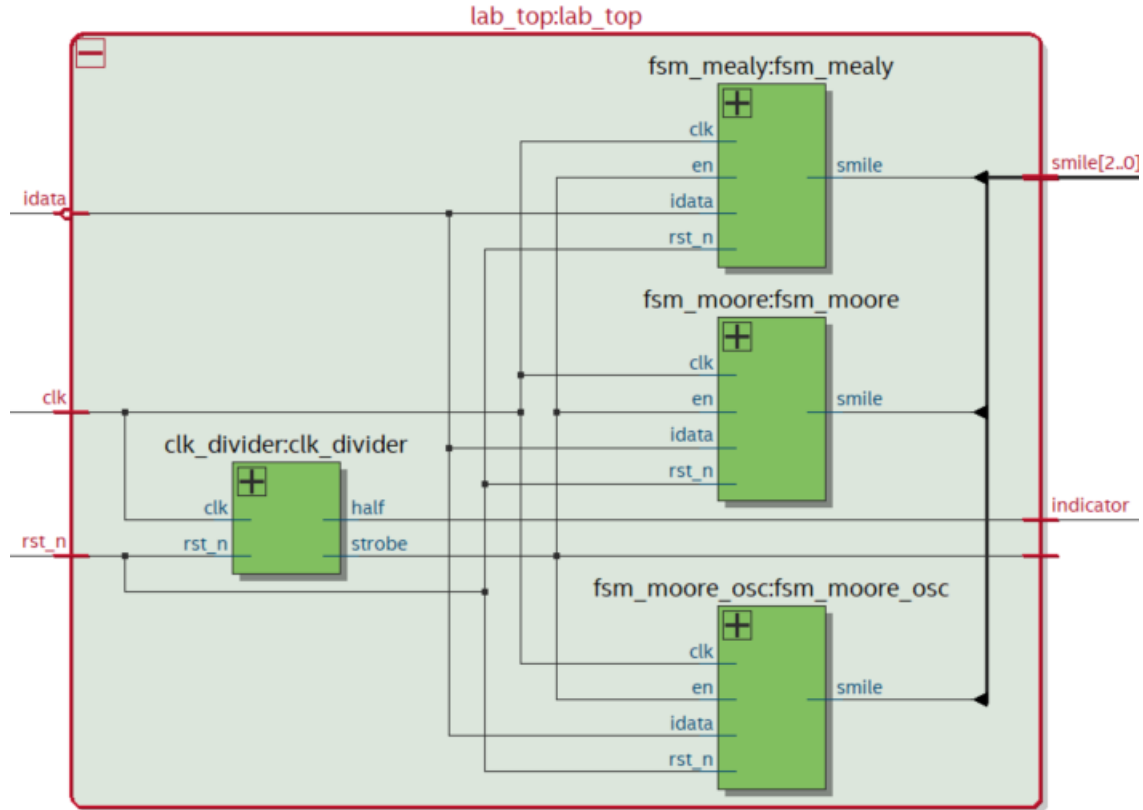
Автомат Мили



- Кругом обозначают состояние
- Стрелка обозначает переход между состояниями в зависимости от входа
- Для автомата Мили стрелки обозначены как **вход** / **выход**

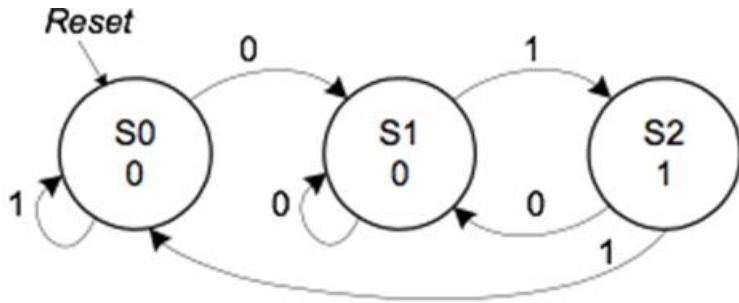
Автомат на Verilog

- Откройте файлы **ddec/lab/31_fsm_basic/rtl/lab_top.v**
- В модуле **lab_top** находятся модули трех конечных автоматов, каждый из которых решает задачу улитки
- Основанный на обычном счетчике частоты **clk_divider** подает на вход автоматов сигнал **en**, который разрешает изменение состояния автомата на следующее



Автомат Мура на Verilog (1 / 2)

- Файл **fsm_moore.v**
- Регистр состояния
- Входной сигнал **en** разрешает изменение состояния автомата на следующее
- Состояния **S0-S2** являются константами
- После сброса автомат переходит в состояние **S0**



```
module fsm_moore
(
    input  clk, input  rst_n,
    input  en,  input  idata, output smile
);

parameter [1:0] S0 = 0, S1 = 1, S2 = 2;
reg [1:0] state, next_state;

// State register
always @ (posedge clk or negedge rst_n)
    if (! rst_n)
        state <= S0;
    else if (en)
        state <= next_state;
```

Автомат Мура на Verilog (2 / 2)

- Определение следующего состояния
- Определение значения выходного сигнала

```
// Next state logic
```

```
always @*
```

```
    case (state)
```

```
        S0      : if (idata) next_state = S0; else next_state = S1;
```

```
        S1      : if (idata) next_state = S2; else next_state = S1;
```

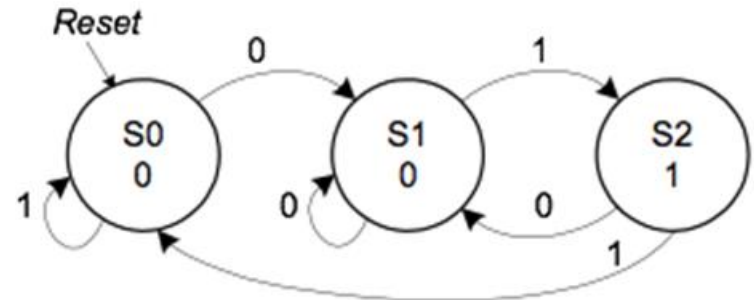
```
        S2      : if (idata) next_state = S0; else next_state = S1;
```

```
        default: next_state = S0;
```

```
    endcase
```

```
// Output logic based on current state
```

```
assign smile = (state == S2);
```



Автомат Мили на Verilog

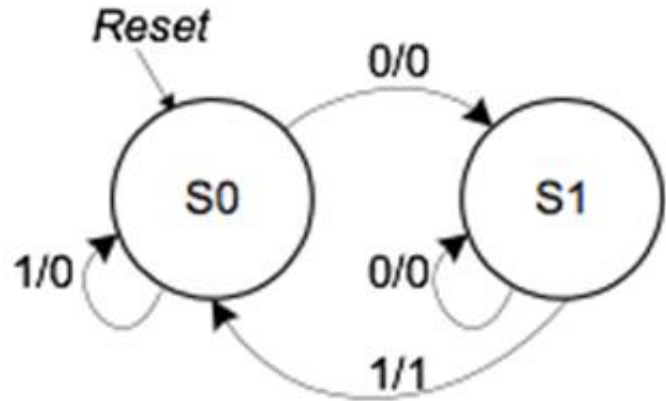
- Файл **fsm_mealy.v**
- Регистр состояния реализован точно также, как и для автомата Мура
- Определены два состояния: S0 и S1
- Значение выходного сигнала зависит не только от текущего состояния, но также и от сигнала на выходе

```
// Next state logic
```

```
assign next_state = idata ? S0 : S1;
```

```
// Output logic based on current state
```

```
assign smile = (idata & state == S1);
```



Автомат Мура v2

- Файл **fsm_moore_ose.v**
- Правильно закодированный код состояния упрощает реализацию
- В данном случае регистр статуса представляет собой **сдвиговый регистр**: на каждом такте он сдвигается влево, а в младший разряд помещается значение входного сигнала
- Таким образом код статуса **S_SMILE=2'b01** – это искомая последовательность
- В этом автомате есть два состояния (**2'b10** и **2'b11**) для которых не введены отдельные именные константы

```
parameter [1:0] S_RESET = 2'b00,  
               S_SMILE = 2'b01;
```

```
reg [1:0] state;  
wire [1:0] next_state;
```

```
// State register  
always @ (posedge clk or negedge rst_n)  
    if (! rst_n)  
        state <= S_RESET;  
    else if (en)  
        state <= next_state;
```

```
// Next state logic (shift register!)  
assign next_state = { state[0], idata };
```

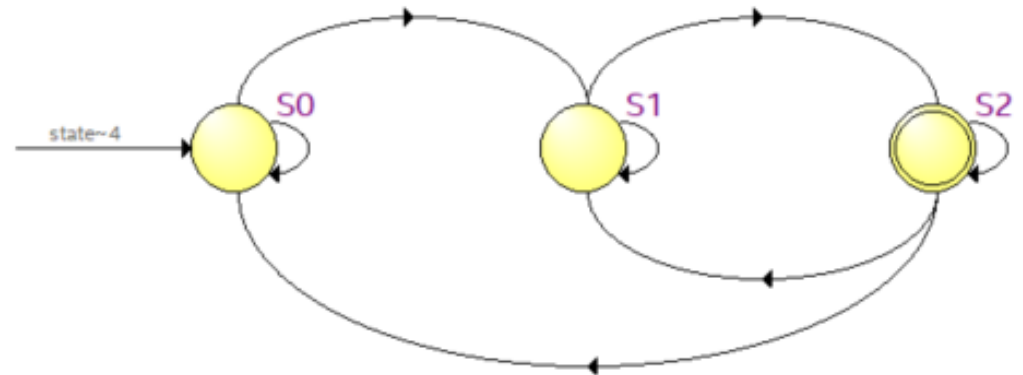
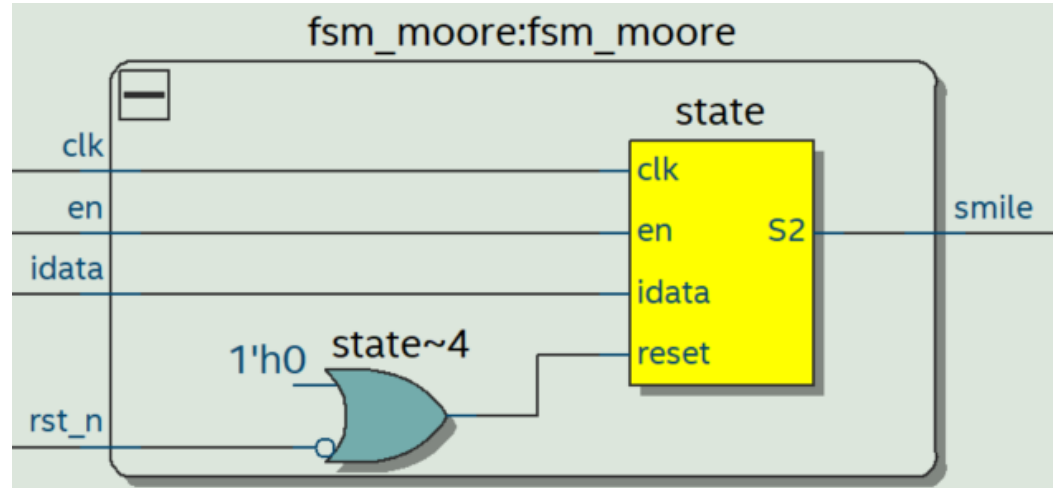
```
// Output logic based on current state  
assign smile = (state == S_SMILE);
```

Простой автомат - практика

- Откройте консоль в каталоге **ddec/lab/ 31_fsm_basic**
- Выполните симуляцию примера: **make sim**
- Выполните синтез и проверку работы примера на плате:
make synth
make load
make open
- Какие отличия есть в поведении разных реализаций конечных автоматов?
- Откройте проект в режиме
RTL-Viewer

Автомат и его синтез

- Средства синтеза умеют в ряде случаев распознавать конечные автоматы
- И применять к ним оптимизацию
- Плохо спроектированный автомат может свести на нет любую оптимизацию



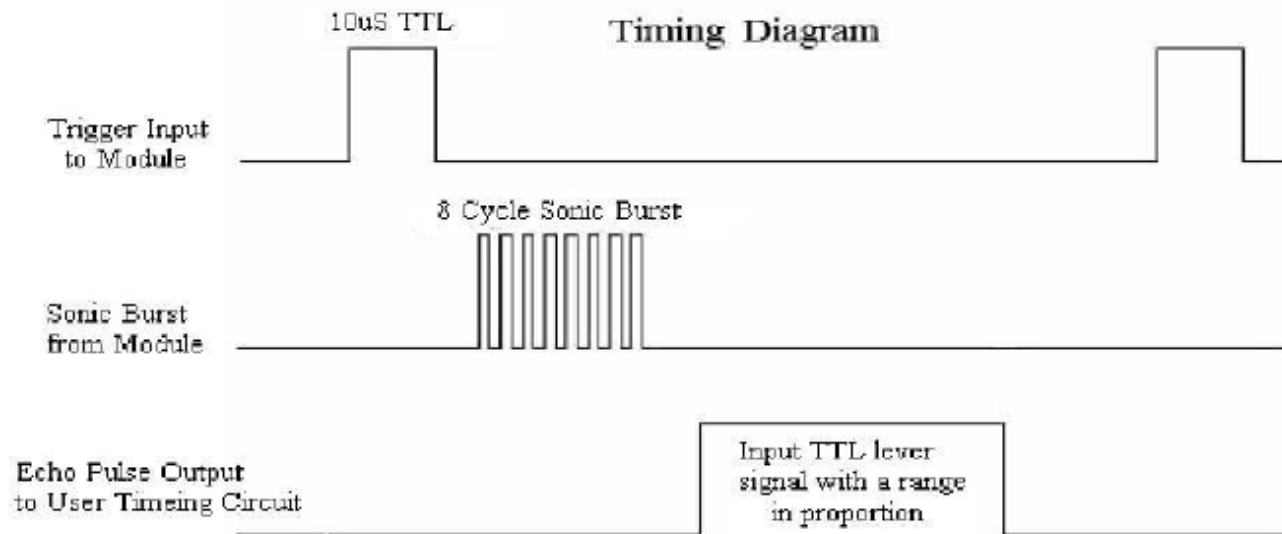
Проектирование сложных автоматов

- На примере ультразвукового датчика расстояния **SR04**
- Общий подход к проектированию:
- Анализ технического задания и документации – необходимо понять, что именно должен делать разрабатываемый модуль
- Декомпозиция задачи – спроектировать и отладить несколько простых автоматов проще, чем один сложный
- Анализ временных диаграмм, выявление состояний и правил перехода между ними
- Реализация конечного автомата на Verilog
- Симуляция с использованием модели устройства
- Отладка и проверка работоспособности на отладочной плате



Анализ требований и документации

- Для того, чтобы модуль начал измерение, на его вход **trigger** необходимо подать импульс длиной **10 us**
- Ответный импульс **echo** пропорционален расстоянию до преграды:
58 us = 1 см
- Максимально измеряемое расстояние: **4 м**
- Измерения проводить раз в **0.5 секунды**

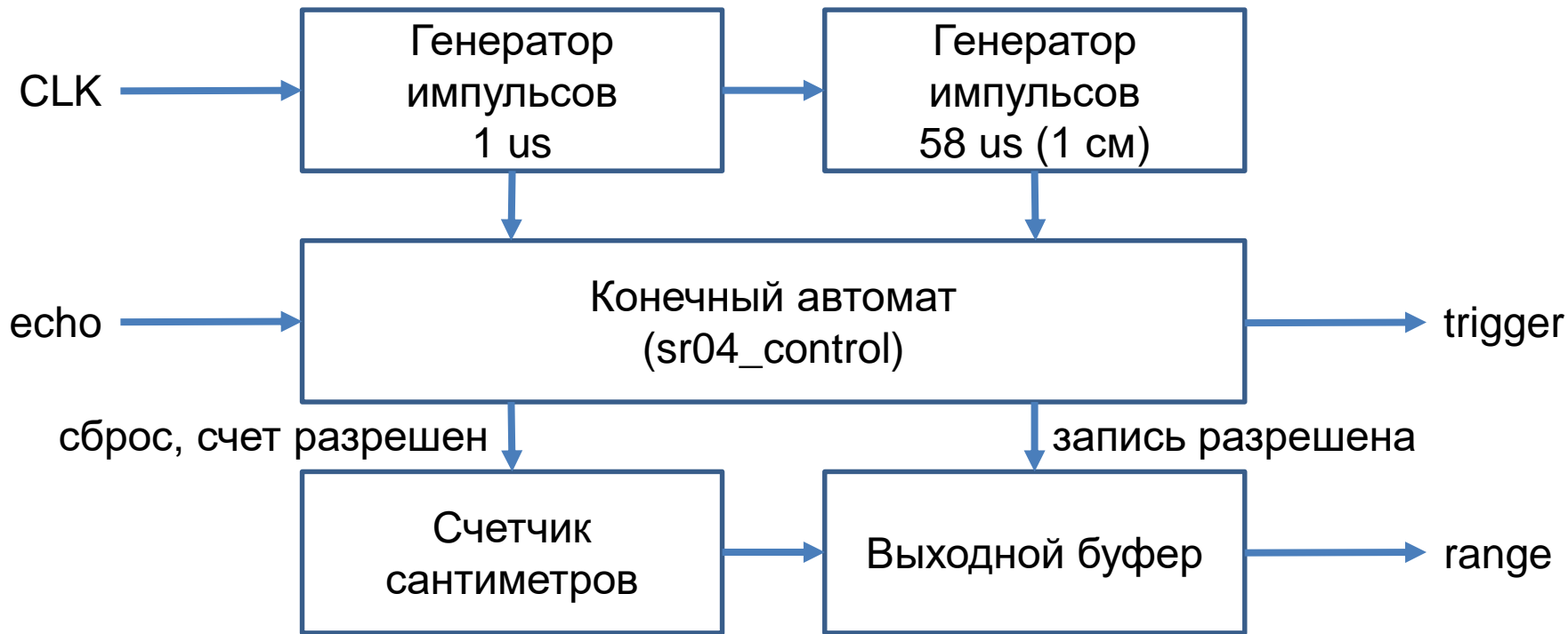


Анализ задачи

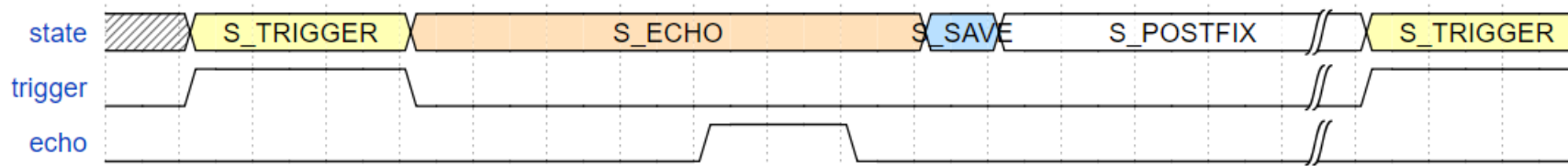


- Тактовый сигнал **CLK** на плате **DE10-Lite**: **50 МГц**
- Для вычисления длительности выходного сигнала-триггера, а также пауз между измерениями подойдет периодичность импульсов **1 us**, т.е. каждые **50** тактов **CLK**
- Расстояние измеряется в сантиметрах, **1 см** соответствует **58 us** длительности сигнала echo
- **Вывод 1**: необходимы 2 модуля, формирующих импульсы каждые **1 us** и **58 us**
- Расстояние до препятствия будет отображаться на 7-сегментных индикаторах
- В момент измерения на экране должно отображаться ранее измеренное значение
- **Вывод 2**: на выходе счетчика, подсчитывающего «сантиметровые» импульсы (**58us**) должен стоять буферный регистр, подключенный к 7-сегментным индикаторам

Декомпозиция задачи

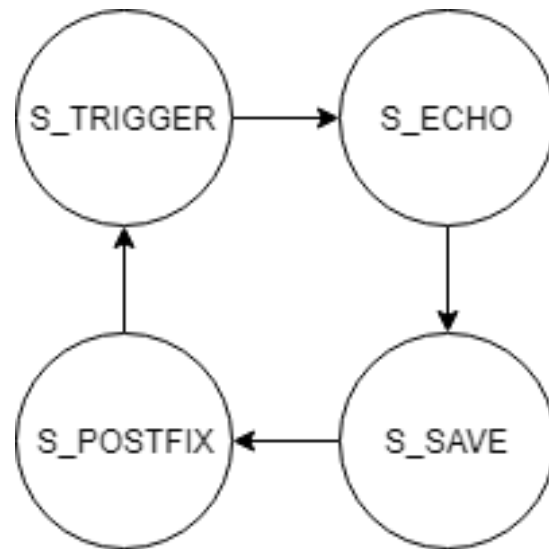


Состояния и переходы между ними



Проанализировав временную диаграмму обмена с датчиком выделим 4 состояния конечного автомата **sr04_control**, смена состояний осуществляется по таймеру:

- **S_TRIGGER** – 10 us
- **S_ECHO** – $400 \cdot 58 \text{ us} = 400 \text{ см}$
- **S_SAVE** – 1 такт
- **S_POSTFIX** – 500 000 us



Реализация на Verilog – регистр состояния

- Откройте файл [dddec/lab/32_fsm_ultrasonic/rtl/sr04_receiver.v](#)

```
localparam S_TRIGGER = 0, // trigger input to module
           S_ECHO     = 1, // wait for echo signal
           S_SAVE     = 2, // save the result
           S_POSTFIX  = 3; // measurement cycle final delay

// State variables
wire [1:0] state;
reg  [1:0] state_nx;
prn_register #(2) state_r (clk, rst_n, state_nx, state);

wire [DELAY_WIDTH-1:0] delay;
wire [DELAY_WIDTH-1:0] delay_nx = (state_nx != state) ? 0 : delay + 1;
wire                  delay_we = strobe_us;
prn_register_we #(DELAY_WIDTH) delay_r (clk, rst_n, delay_we, delay_nx, delay);
```

Реализация на Verilog – следующее состояние

```
// next state variables value  
always @(*) begin  
    state_nx = state;  
    case (state)  
        S_TRIGGER : if (delay == DELAY_TRIGGER)  
                        state_nx = S_ECHO;  
        S_ECHO     : if (delay == DELAY_ECHO)  
                        state_nx = S_SAVE;  
        S_SAVE      : state_nx = S_POSTFIX;  
        S_POSTFIX   : if (delay == DELAY_POSTFIX)  
                        state_nx = S_TRIGGER;  
    endcase  
end
```

Реализация на Verilog – выходные сигналы

```
// fsm output
always @(*) begin
    trigger      = 1'b0;
    measure_cm   = 1'b0;
    measure_end  = 1'b0;
    case (state)
        S_TRIGGER : trigger      = 1'b1;
        S_ECHO     : measure_cm   = strobe_cm & echo;
        S_SAVE     : measure_end  = 1'b1;
    endcase
end
```

Симуляционная модель устройства

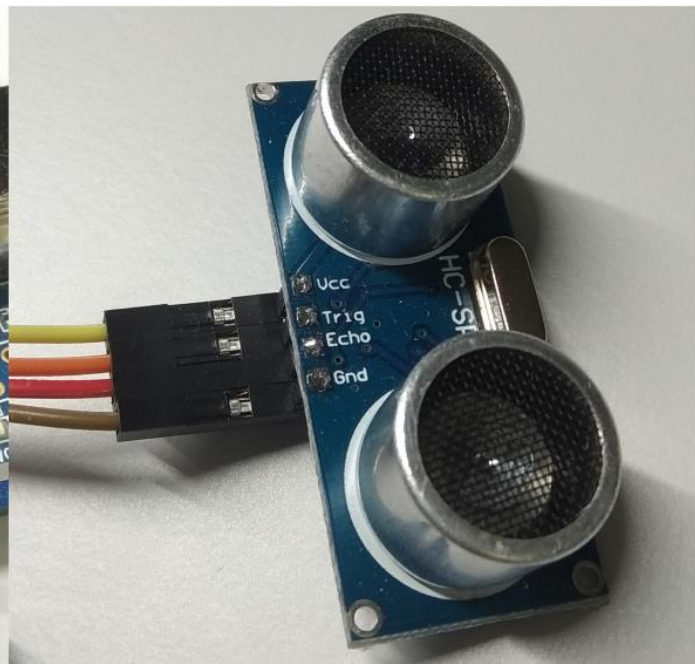
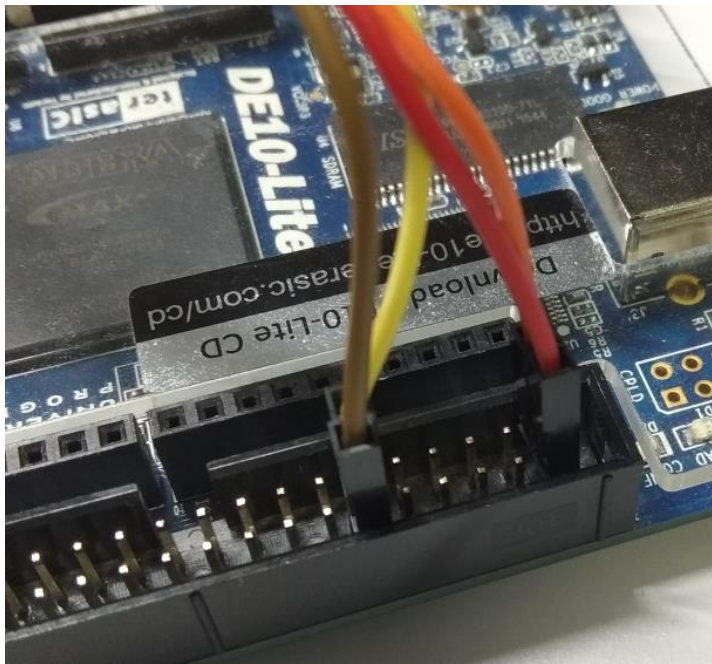
- Программа симулятор не умеет работать с реальными устройствами
- При отладке в симуляторе используют **симуляционные модели устройств**
- Для сложных блоков (к примеру, чипы оперативной памяти) симуляционные модели обычно предоставляются производителем
- Простую модель можно написать самому

```
module sr04_simulation_model #(
    parameter Twait = 1000, Techo = 3000
)()
    input      clk,      input      trigger,
    input      enable, output reg echo
);

initial begin
    echo = 1'b0;
    forever begin
        @(posedge clk);
        @(negedge trigger);
        if(enable) begin
            repeat(Twait) @(posedge clk); echo = 1'b1;
            repeat(Techo) @(posedge clk); echo = 1'b0;
        end
    end
end
```

Подключение датчика к плате

- Датчик работает от 5В, максимально поддерживаемое напряжение на входе MAX10 - 3.3В. В таких случаях требуется схема преобразования уровней
- То, что чип MAX10 на Terasic DE10-Lite не сгорает при работе с этим датчиком, было выяснено случайным образом. Будьте осторожны при работе с другими FPGA платами!



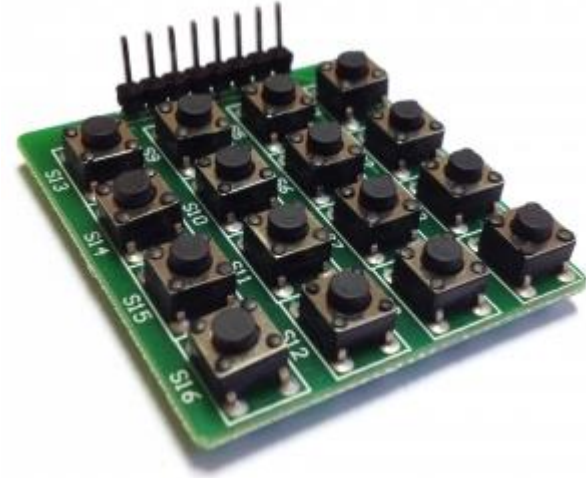
Отладка и проверка работоспособности

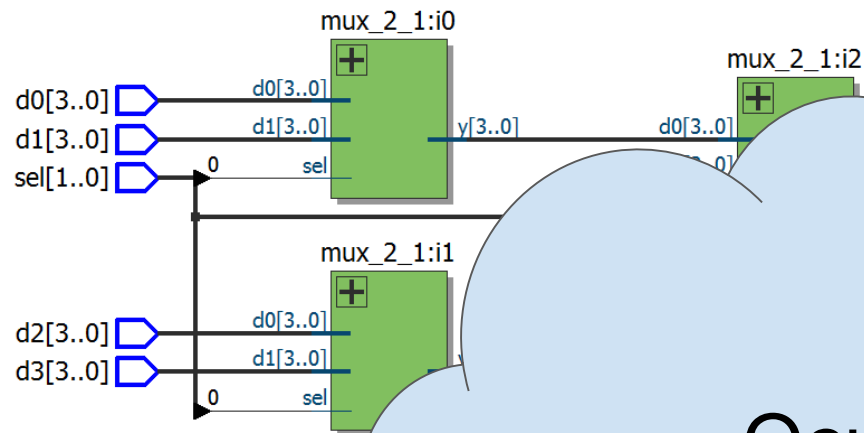
- Откройте консоль в каталоге [dddec/lab/32_fsm_ultrasonic](#)
- Выполните симуляцию примера:
make sim
- Выполните синтез и проверку работы примера на плате:
make synth
make load
make open
- Результаты измерений отображаются в 16-ричном виде. Что необходимо сделать для того, чтобы результат отображался в более привычном 10-чном виде. Какие варианты решения есть у этой задачи?
- Спроектируйте конечный автомат для опроса 16-клавишной клавиатуры. При нажатии на кнопку на 7-сегментном индикаторе должен отображаться код нажатой клавиши. Ранее введенные значения при этом сдвигаются влево

Задание для самостоятельной работы

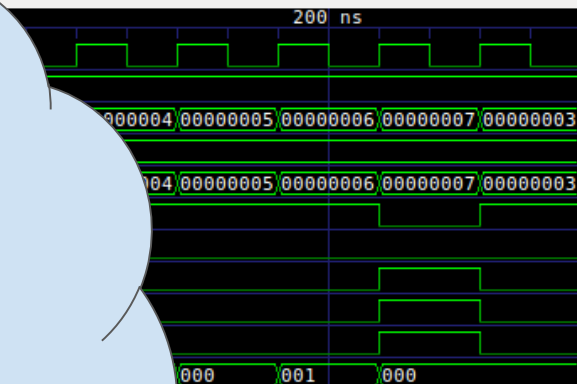
- Спроектируйте конечный автомат для опроса 16-кнопочной клавиатуры
- На 7-сегментном индикаторе должен отображаться код нажатой клавиши. Ранее введенные значения при этом сдвигаются влево (как у калькулятора)
- Для того, чтобы опрос клавиш выполнялся корректно вам необходимо будет использовать подтягивающий резистор на портах FPGA
- Его можно включить в файле **system.qsf** или **script_quartus.tcl**, пример:

set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO[7]

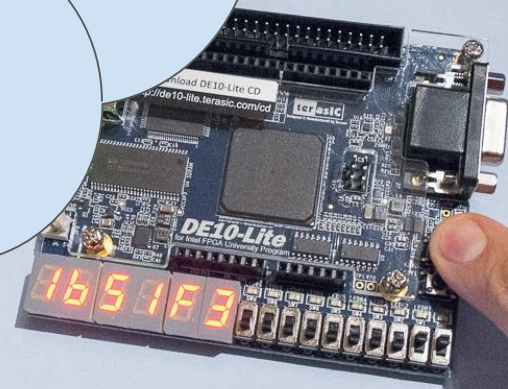
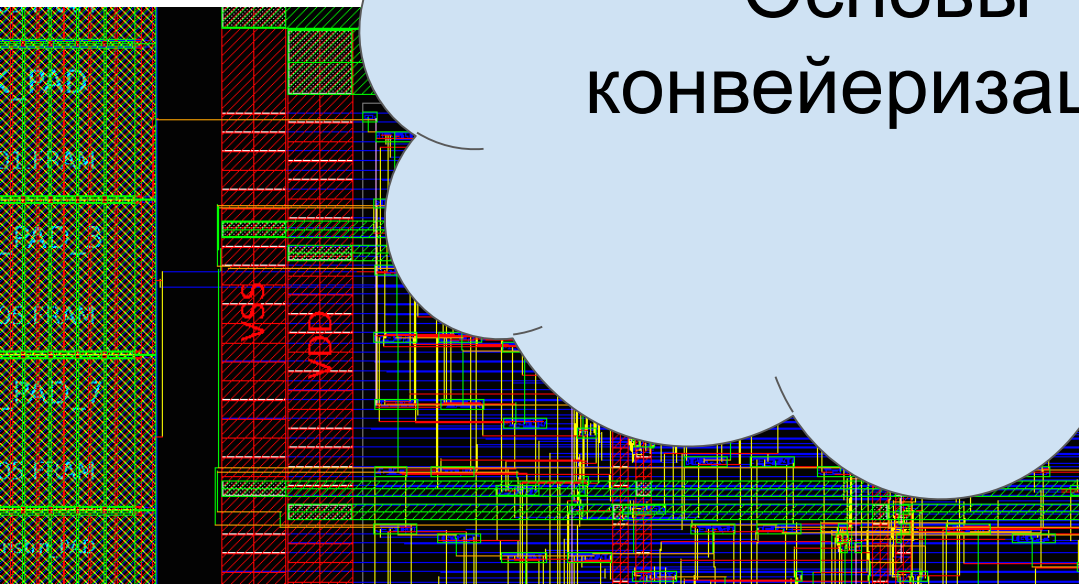




Marker: 670 ns | Cursor: 290500 ps

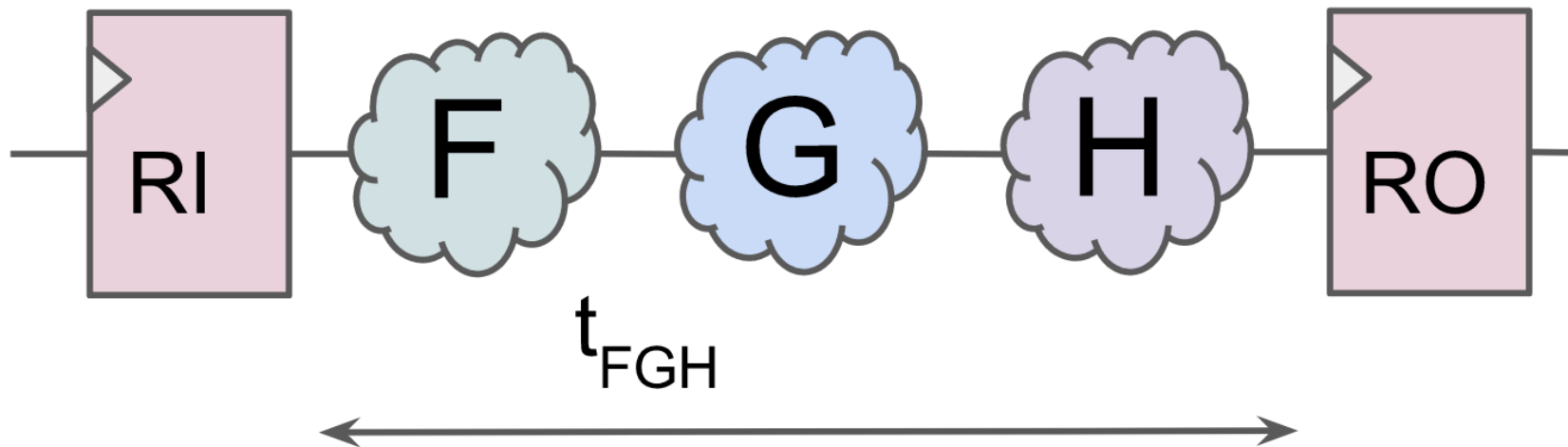


Основы конвейеризации



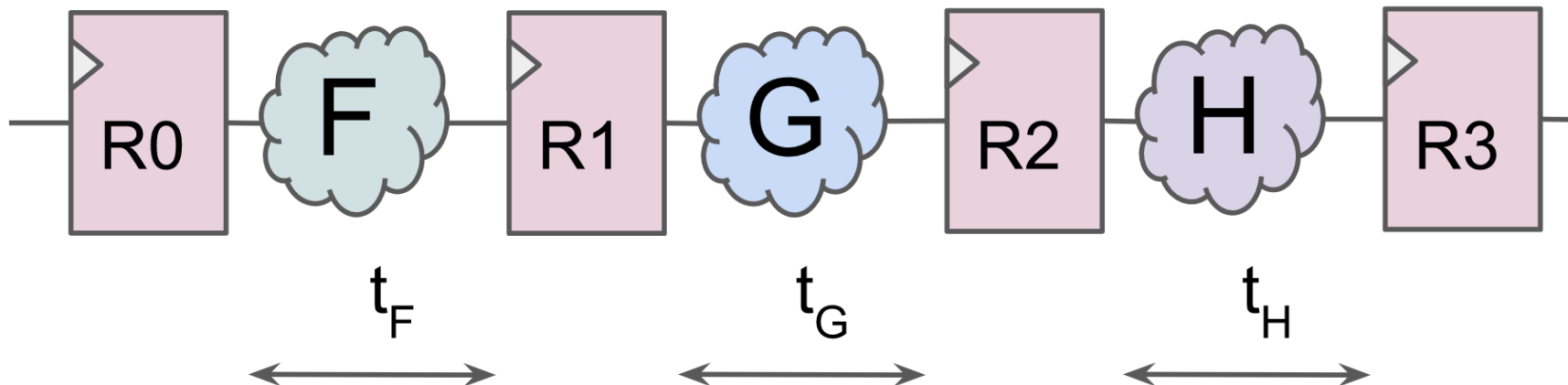
Концепция конвейеризации - 1

- Пусть наше вычисление выполняется за несколько шагов
- Мы можем получить один результат вычислений за такт
- Период тактового сигнала равен t_{FGH} .



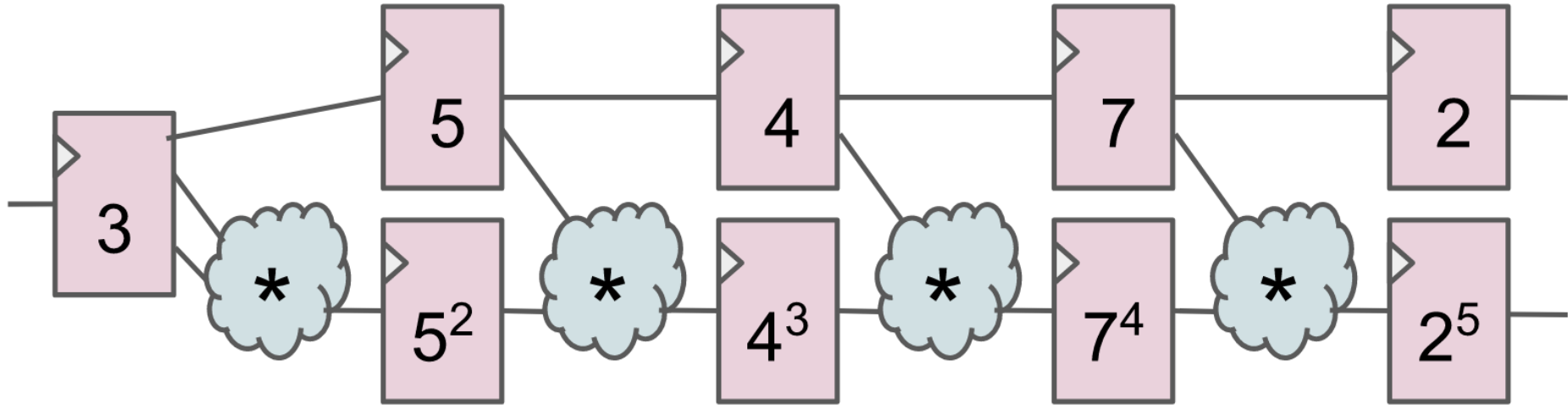
Концепция конвейеризации - 2

- Добавим регистры между шагами
- Это уменьшает период тактового сигнала с $(t_F + t_G + t_H)$ до $\max(t_F, t_G, t_H)$.
- Теперь мы можем подавать данные на вход схемы не дожидаясь, пока результаты предыдущего вычисления появятся на ее выходе: $F \rightarrow G \rightarrow H$
- Это позволяет увеличить пропускную способность схемы



Концепция конвейеризации - 3

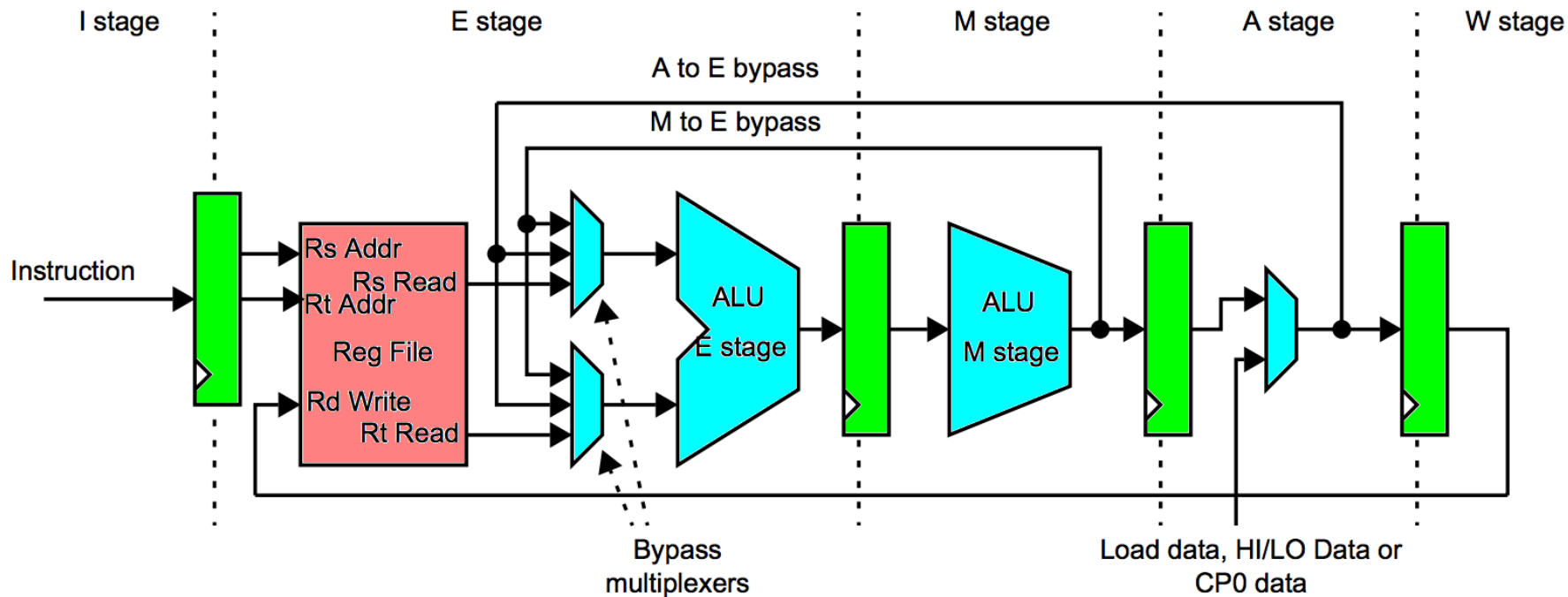
- Пример схемы, которая вычисляет $F(n) = n^5$.
- На каждом такте на вход схемы подаётся новое число
- При этом результат доступен на выходе схемы спустя 4 такта



Конвейер процессора

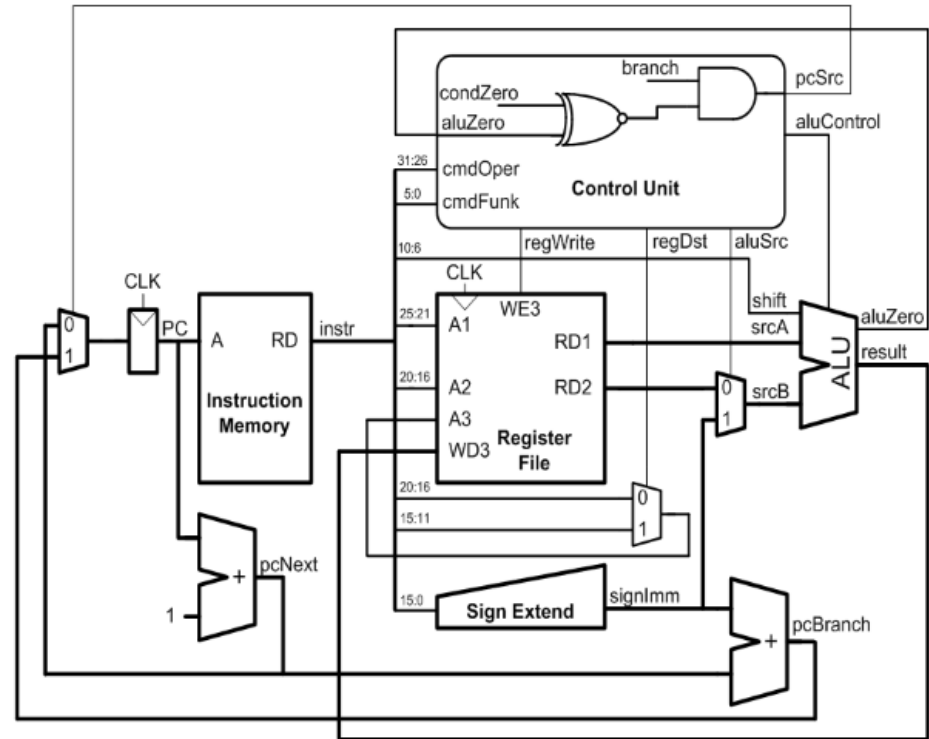
— наиболее известный пример конвейеризации

Исполнительное устройство процессора MIPS M5150 обрабатывает поток инструкций



Изучение процессоров с помощью schoolMIPS и MIPSfpga

- schoolMIPS -
простейшее RISC
ядро созданное для
обучение основам
- MIPSfpga -
промышленное ядро,
один из вариантов
MIPS M5150 с
предыдущего слайда



Спасибо!

