Ресурс памяти ПЛИС: RAM (random access memory)

- Ресурс для хранения данных
- Может быть реализована как распределенная память (Distributed RAM), как блочная память (BRAM) или как регистры
- Основные типы RAM: однопортовая (single port), простая двухпортовая (simple dual port), полная двухпортовая (true dual port)

Distributed RAM

- Реализуется на основе LUT в SliceM (например, 6-LUT может хранить 64 бита)
- Записывается синхронно
- Считывается асинхронно
- Подходит для реализации буферов памяти с быстрым доступом
- Не подходит для хранения большого количества данных

BRAM

- Специально выделенный ресурс ПЛИС
- Тот же объем памяти занимает меньше места на кристалле, чем distributed RAM
- Подходит для хранения больших объемов данных
- Обычно, чем больше и дороже ПЛИС, тем больше в ней блочной памяти

- САПР можно указать конкретную реализацию памяти с помощью атрибутов
- Атрибут – это специальная языковая конструкция, позволяющая влиять на синтез проекта

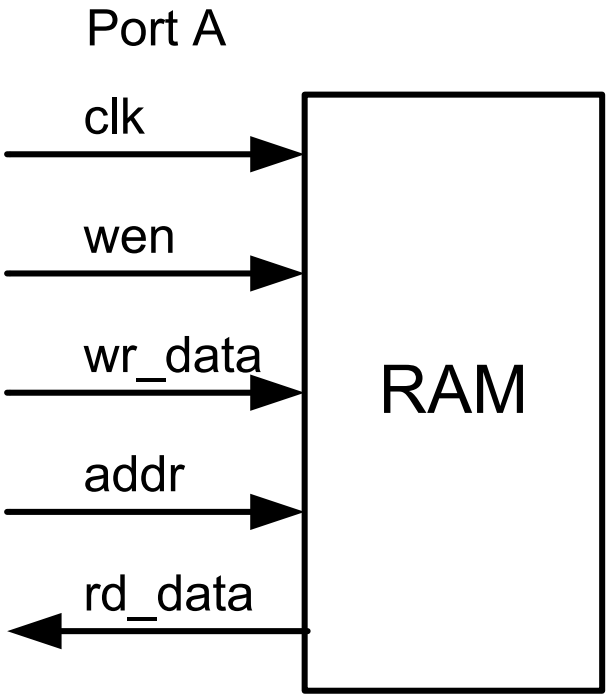
```
// used Block RAM
(*ram_style = "block"*)       logic [WIDTH-1:0] ram [DEPTH-1:0];
```

```
// used distributed LUT RAMs
(*ram_style = "distributed"*) logic [WIDTH-1:0] ram [DEPTH-1:0];
```

```
// used registers instead of RAM
(*ram_style = "registers"*)   logic [WIDTH-1:0] ram [DEPTH-1:0];
```

Single Port RAM

- Имеет один адресный вход
- Чтение и запись происходит по одному адресу

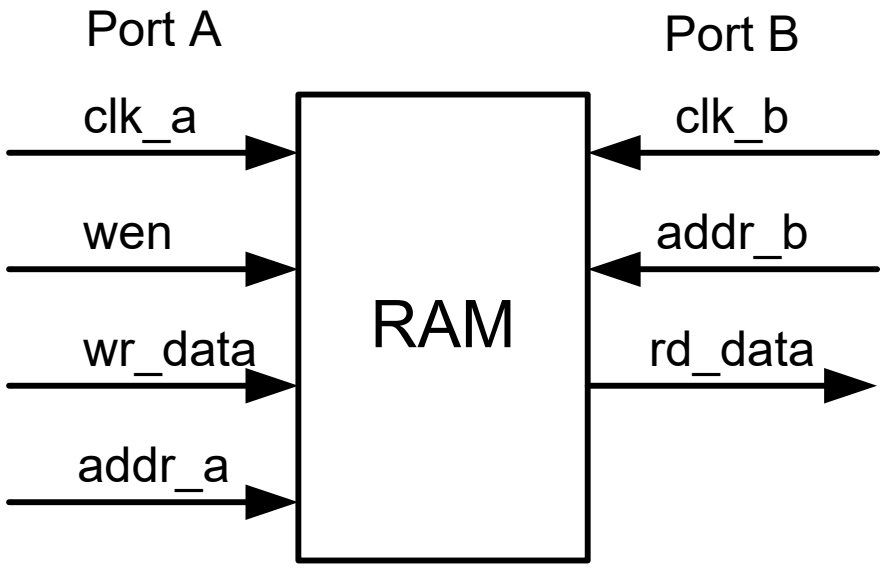


Single Port RAM

```
module single_port_ram #(
    parameter WIDTH = 8,
    parameter DEPTH = 64
)(
    input logic clk,
    input logic [WIDTH-1:0] wr_data,
    input logic [$clog2(DEPTH)-1:0] addr,
    input logic wen,
    output logic [WIDTH-1:0] rd_data
);
    logic [WIDTH-1:0] ram [DEPTH-1:0];

    always @(posedge clk) begin
       if(wen)
            ram[addr] <= wr_data; // запись
       rd_data <= ram[addr]; // чтение
    end

endmodule
```

Simple Dual Port RAM

- Один порт работает на чтение, другой на запись
- Каждый порт имеет свой адресный вход
- Порты тактируются разными сигналами
- На основе такой памяти строится FIFO

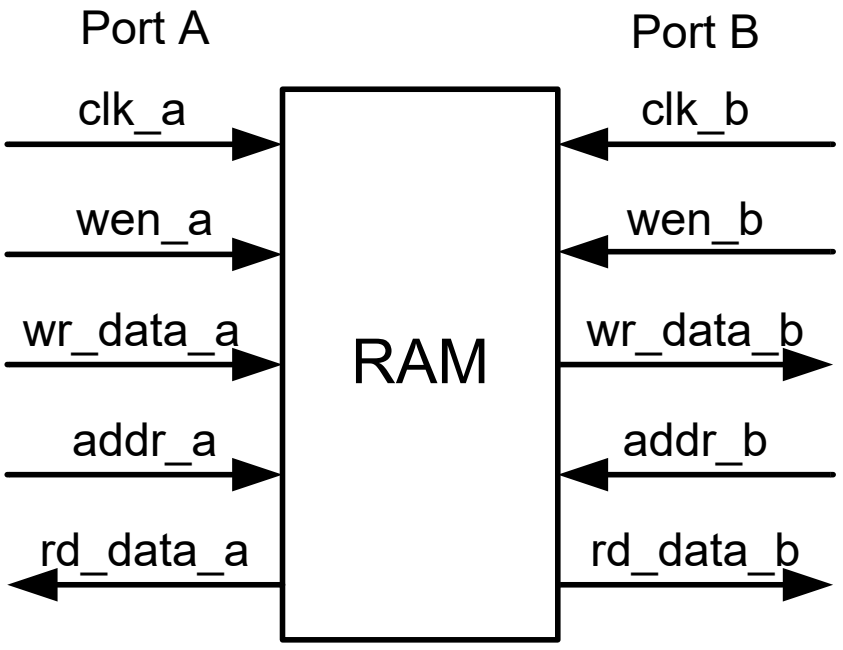


```
module simple_dual_port_ram #(
    parameter WIDTH = 8,
    parameter DEPTH = 64
)(
    input logic clk_a, // тактовая частота записи
    input logic clk_b, // тактовая частота чтения
    input logic [WIDTH-1:0] wr_data,
    input logic [$clog2(DEPTH)-1:0] addr_a,
    input logic wen,
    input logic [$clog2(DEPTH)-1:0] addr_b,
    output logic [WIDTH-1:0] rd_data
);
    logic [WIDTH-1:0] ram [DEPTH-1:0];

    always @(posedge clk_a) begin // запись
       if(wen)
            ram[addr_a] <= wr_data;
    end

    always @(posedge clk_b) begin  // чтение
       rd_data <= ram[addr_b];
    end

endmodule
```

Simple Dual Port RAM

True Dual Port RAM

- Оба порта работают на чтение и запись
- Каждый порт имеет свой адресный вход
- В пределах одного порта чтение и запись происходит по одному адресу
- Порты тактируются разными сигналами
- На основе такой памяти строится FIFO



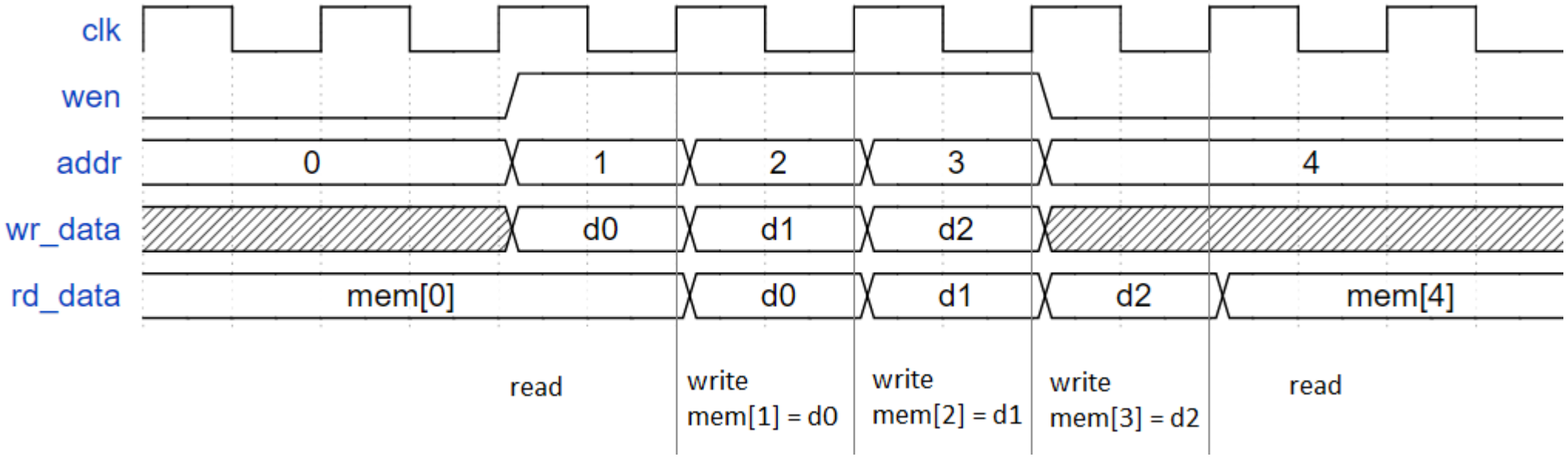
```
module true_dual_port_ram #(
    parameter WIDTH = 8,
    parameter DEPTH = 64
)(
    input logic clk_a, // тактовая частота записи
    input logic clk_b, // тактовая частота чтения
    input logic [WIDTH-1:0] wr_data_a,
    input logic [$clog2(DEPTH)-1:0] addr_a,
    input logic wen_a,
    output logic [WIDTH-1:0] rd_data_a,
    input logic [WIDTH-1:0] wr_data_b,
    input logic [$clog2(DEPTH)-1:0] addr_b,
    input logic wen_b,
    output logic [WIDTH-1:0] rd_data_b
);
    logic [WIDTH-1:0] ram [DEPTH-1:0];

    always @(posedge clk_a) begin
       if(wen_a)
            ram[addr_a] <= wr_data_a; // запись с порта A
       rd_data_a <= ram[addr_a]; // чтение по порту A
    end

    always @(posedge clk_b) begin
       if(wen_b)
            ram[addr_b] <= wr_data_b; // запись с порта B
       rd_data_b <= ram[addr_b]; // чтение по порту B
    end

endmodule
```

True Dual Port RAM

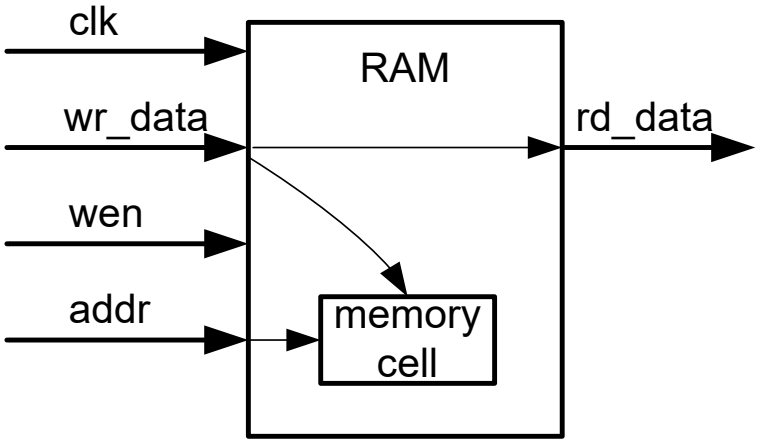
Режимы записи RAM

- Реализация RAM отличается по режиму записи (при реализации через IP или макрос Vivado - параметр write_mode)
- Существует три типа реализации: запись перед чтением (write_first), чтение перед записью (read_first), без чтения при записи (no_change)

Запись перед чтением (write_first)

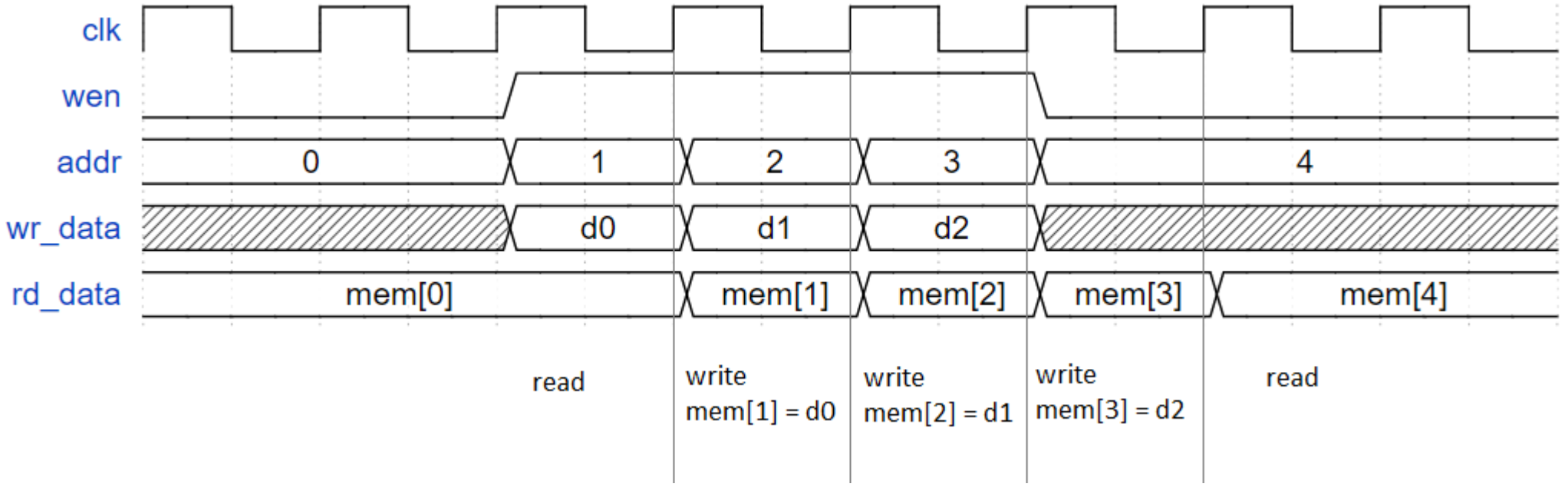


Запись перед чтением (write_first)

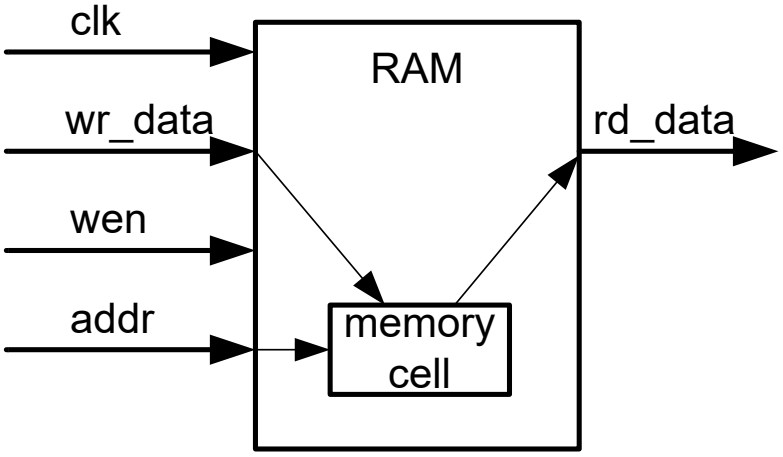



```
always @(posedge clk) begin
    if(wen) begin
          ram[addr] <= wr_data;
          rd_data <= wr_data;
    end
    else
          rd_data <= ram[addr];
end
```

Чтение перед записью (read_first)

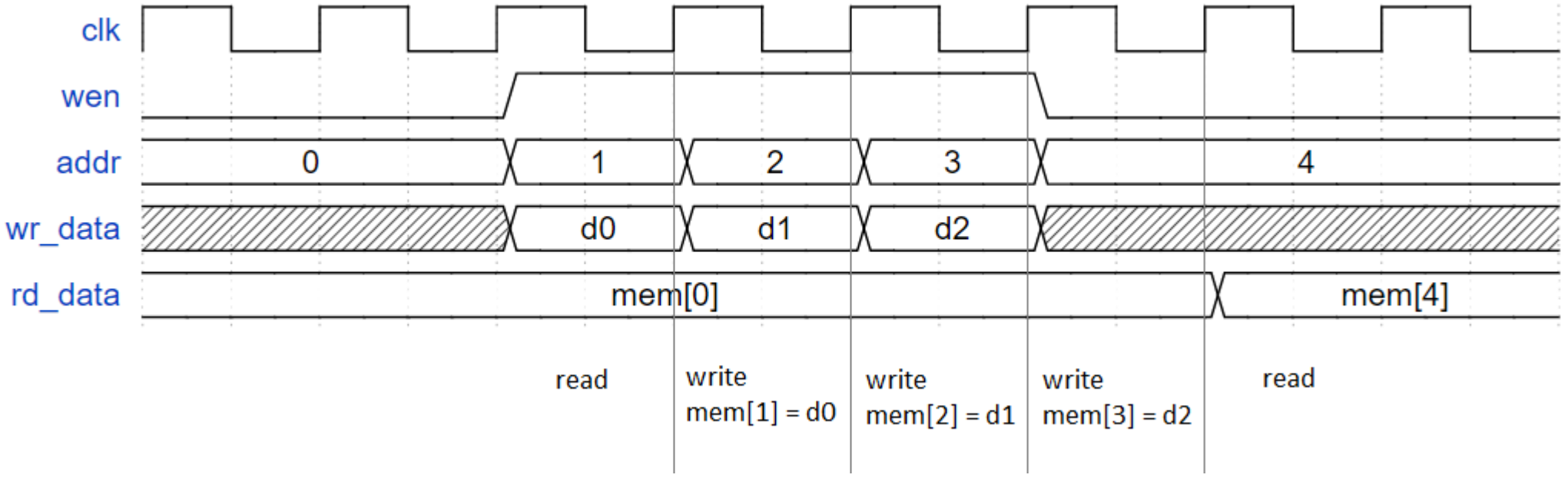


Чтение перед записью (read_first)

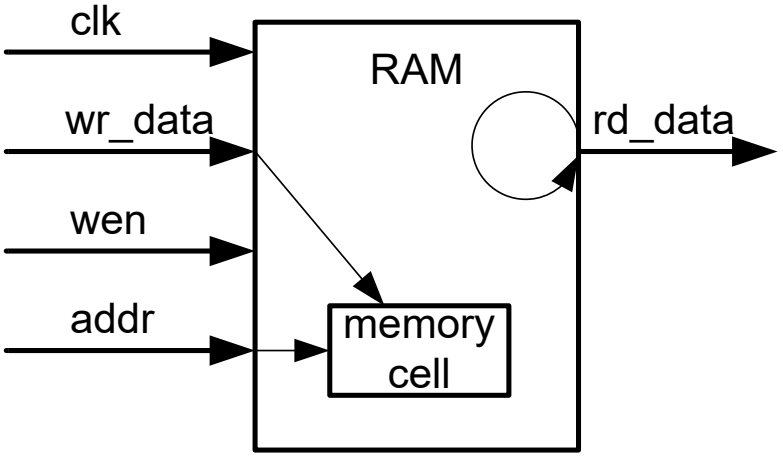



```
always @(posedge clk) begin
    if(wen) begin
        ram[addr] <= wr_data;
    end
    rd_data <= ram[addr];
end
```

Без чтения при записи (no_change)



Без чтения при записи (no_change)




```
always @(posedge clk) begin
    if(wen)
        ram[addr] <= wr_data;
    else
        rd_data <= ram[addr];
end
```

FIFO

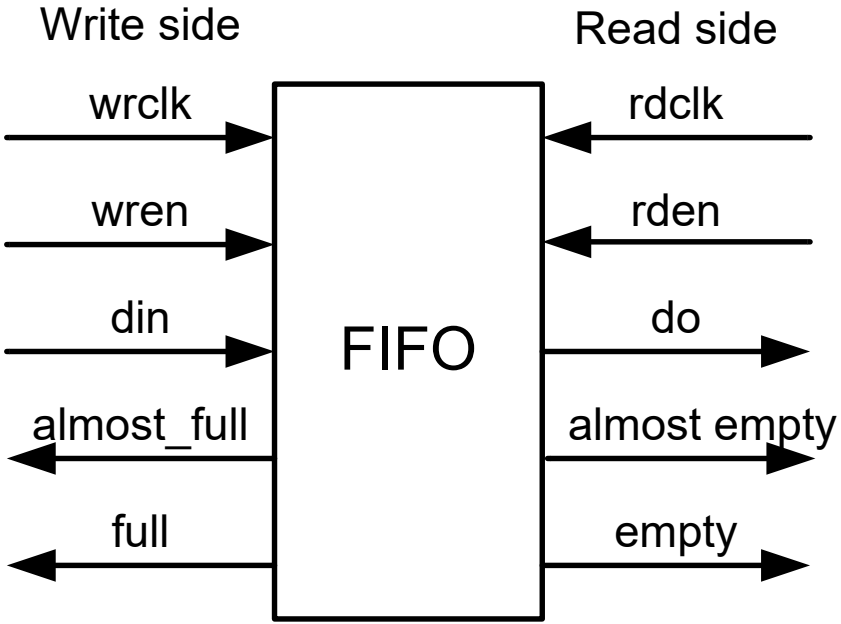
- Блоки RAM используются для формирования FIFO
- Основные применения: пересечение тактовых доменов, буферизация данных при их отправке за пределы ПЛИС, промежуточное хранения данных для их дальнейшей обработки
- В отличие от “чистого” RAM у FIFO порядок записи и чтения фиксирован

FIFO

- Может иметь один тактовый сигнал (Single Clock FIFO) или независимые тактовые сигналы чтения и записи (Dual Clock FIFO)
- Single Clock FIFO используется для буферизации
- Dual Clock FIFO используется для передачи данных между разными тактовыми доменами
- В Dual Clock FIFO все сигналы, относящиеся к чтению, синхронизируются с тактовым сигналом чтения и все сигналы для записи синхронизируются с тактовым сигналом записи

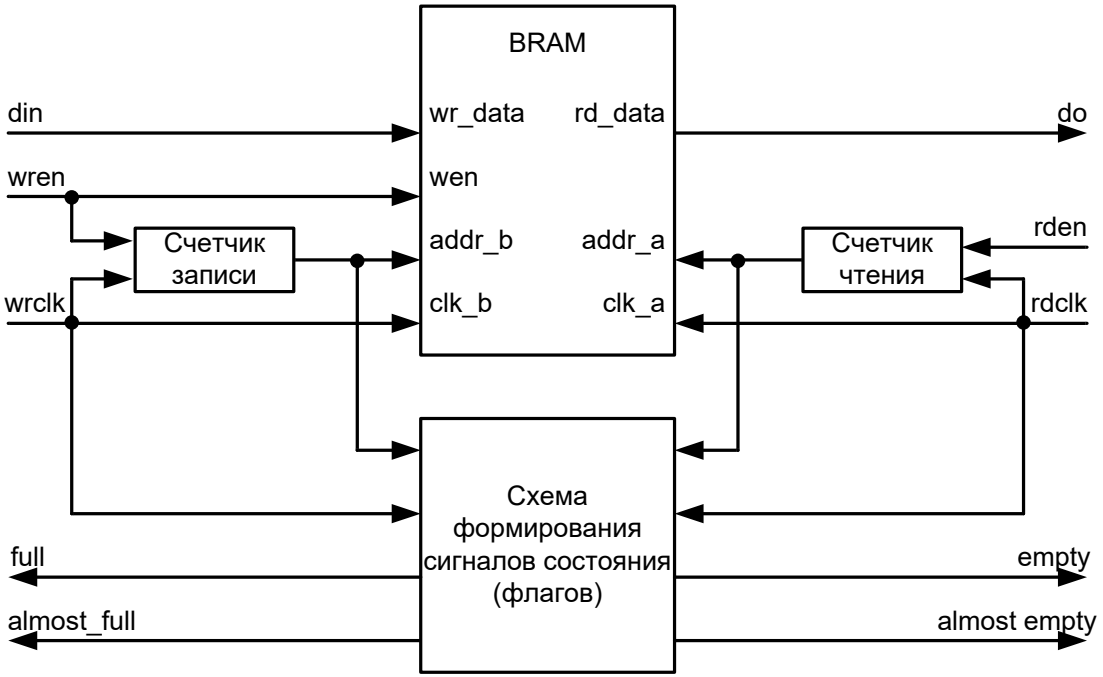
FIFO

- Не имеет адресов чтения/записи (производит последовательное считывание/запись)
- Запись в FIFO осуществляется синхронно по сигналу wen
- Считывание из FIFO осуществляется синхронно по сигналу ren
- Флаги full/empty указывают на заполненный/пустой FIFO
- Флаги almost _full/almost_empty используются для раннего предупреждения о заполненном или пустом FIFO (опциональны)



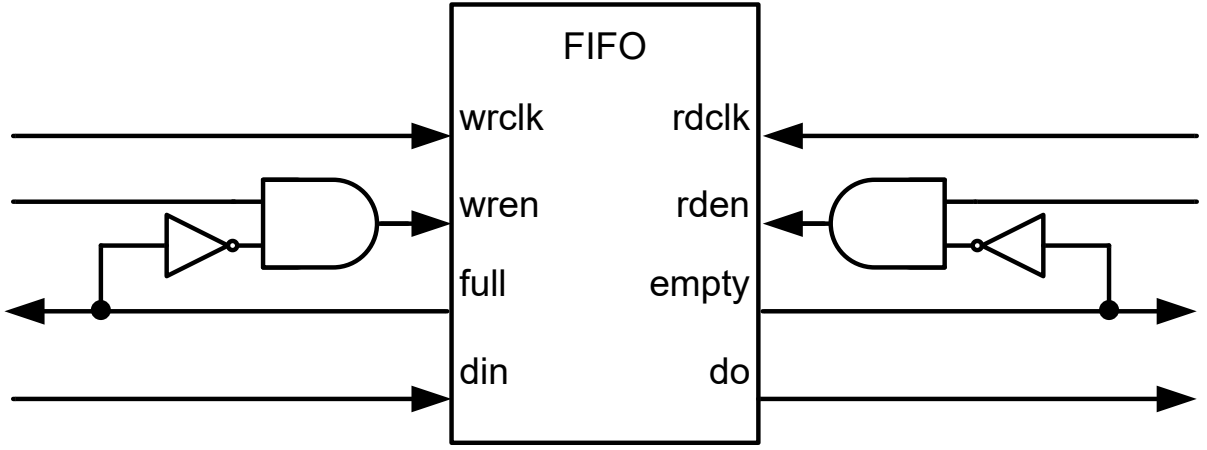
FIFO

- В общем случае представляет собой Dual port RAM, два счетчика адреса и логику формирования сигналов состояния full/empty (almost_full/almost_empty)



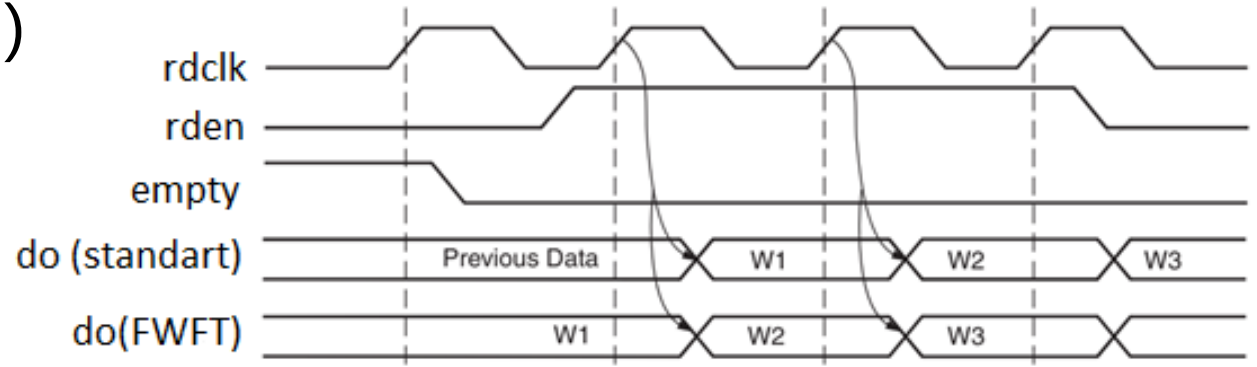
FIFO

- Нельзя писать в заполненное FIFO (overflow) и считывать из пустого (underflow)
- При записи и чтении необходимо анализировать флаги full и empty



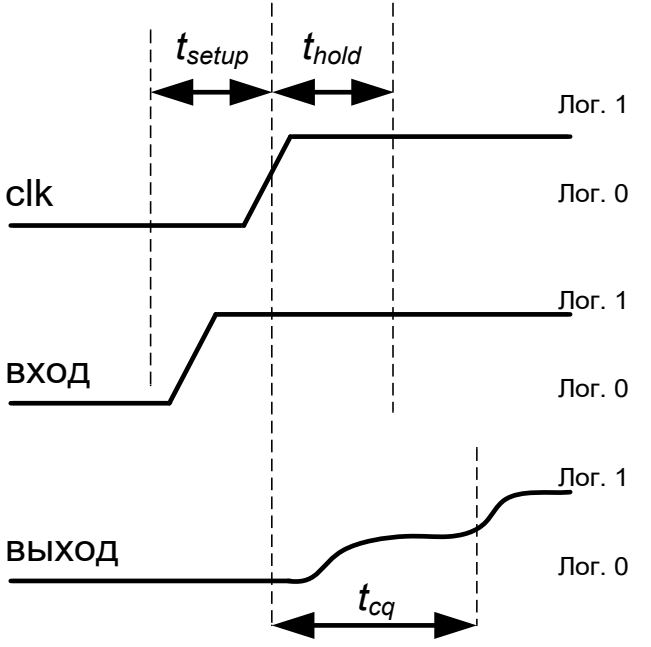
FIFO

- Может работать в двух режимах: стандартный режим (Standard) и First Word Fall Through (FWFT)
- В стандартном режиме первое записанное слово на выходе появится после установки rden по фронту следующего тактового сигнала чтения
- В FWFT режиме первое слово, записанное в пустой FIFO, автоматически появится на выходе вне зависимости от rden (последующие данные считывается как обычно – при установленном rden)



Метастабильность

- Это состояние, в котором выходные данные триггера в FPGA неизвестны или недетерминированы в связи с нарушением времени установки или удержания триггера
- В метастабильном состоянии выходной сигнал колеблется на значении между высоким и низким уровнем в течение некоторого периода времени, превышающего t_{cq} триггера, после чего сигнал стабилизируется либо в 0, либо в 1



Метастабильность

- Нельзя точно сказать, какое значение примет выходной сигнал после выхода из метастабильного состояния
- Такое поведение порождает неизвестный результат в последующих элементах
- Если выходной сигнал триггера, попавший в метастабильное состояние, не успеет стабилизироваться до момента времени, когда следующий триггер захватит этот сигнал, то следующий триггер также может попасть в метастабильное состояние
- Внутри ПЛИС нарушение временных характеристик триггера в случае, например, слишком длинных путей распространения сигнала, контролируется САПР при временном анализе

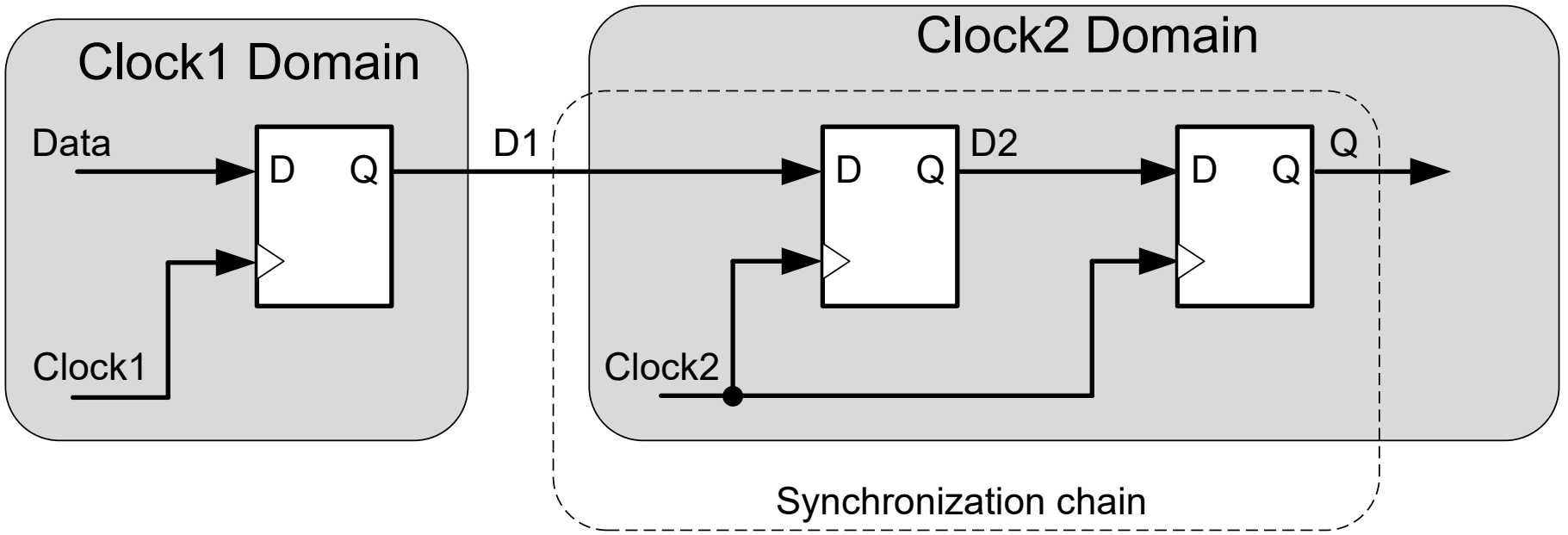
Метастабильность

- Проблемы метастабильности обычно возникают, когда сигнал передается между схемами в несвязанных или асинхронных тактовых доменах
- В этом случае разработчик не может гарантировать, что сигнал не будет нарушать временные характеристики триггеров, поскольку сигнал может поступить в любое время относительно тактового сигнала
- Не каждое изменение сигнала, нарушающее временные характеристики триггера, приводит к метастабильному выходному сигналу
- Вероятность перехода триггера в метастабильное состояние и время, необходимое для возврата в стабильное состояние, варьируются в зависимости от технологического процесса изготовления устройства и условий эксплуатации

Метастабильность

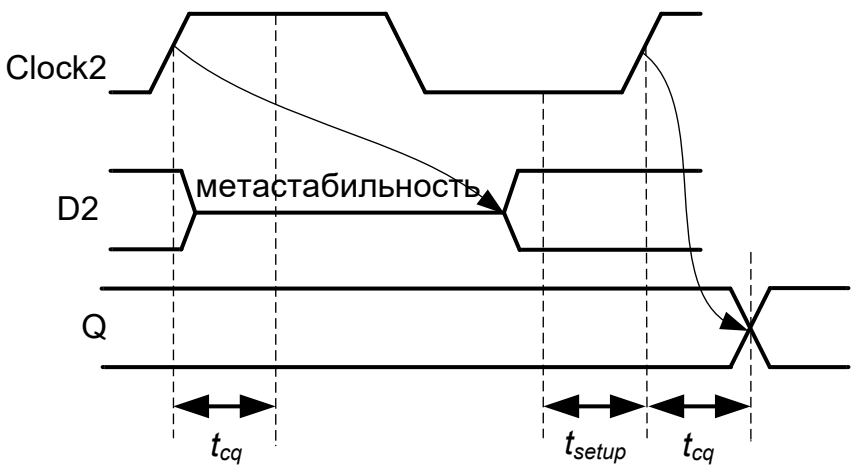
- Когда сигнал передается между тактовыми доменами, необходимо синхронизировать этот сигнал с новым тактовым доменом, прежде чем его можно будет использовать
- Чтобы свести к минимуму сбои из-за метастабильности, разработчики обычно используют последовательность триггеров (цепь синхронизации, синхронизатор) для синхронизации сигнала с новой тактовой областью.
- Эти триггеры дают дополнительное время потенциально метастабильному сигналу для установки стабильного значения, прежде чем сигнал будет использоваться в остальной части проекта

Цепь синхронизации



Цепь синхронизации

- Если период тактового сигнала достаточно большой, то к концу периода выход первого триггера стабилизируется, второй триггер зафиксирует стабильное значение и сформирует корректный выходной сигнал



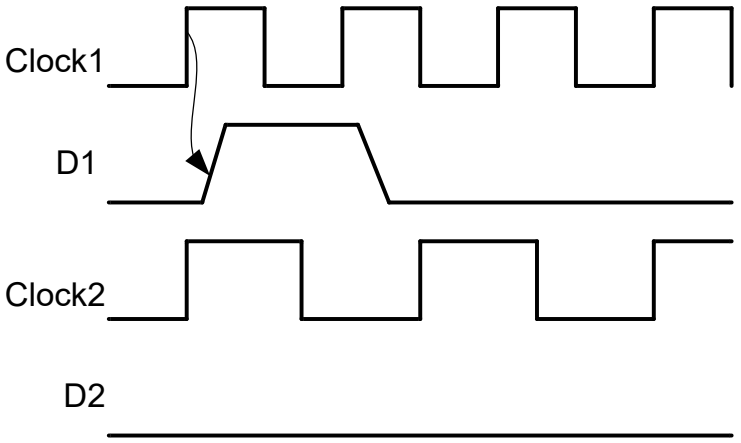
Цепь синхронизации

Последовательность триггеров будет выполнять роль цепи синхронизации, если:

- все триггеры цепочки тактируются одним тактовым сигналом
- на первый триггер цепочки приходит асинхронный сигнал
- каждый триггер соединен только с одним триггером (за исключением последнего триггера цепочки)

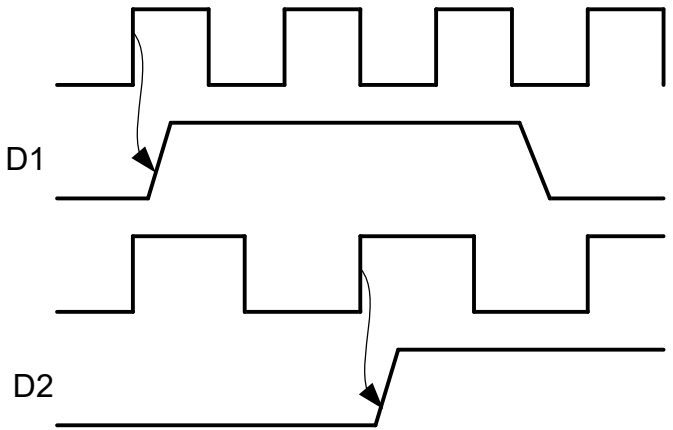
Цепь синхронизации

- Если тактовый домен 1 работает на частоте, большей чем тактовый домен 2, может возникнуть ситуация когда сигнал D1 устанавливается и сбрасывается между фронтами тактового сигнала домена 2
- Такой сигнал не захватится синхронизатором



Цепь синхронизации

- Если частота тактового домена 1, больше частоты тактового домена 2, то сигнал D1 необходимо “растянуть” для его захвата в медленном тактовом домене 2
- Чтобы должным образом гарантировать соблюдение времени установки и удержания, рекомендуется растягивать сигнал так, чтобы он занимал как минимум 2 тактовых цикла в медленной тактовой области



Метастабильность

- В случае многоразрядной шины нельзя поставить по синхронизатору на каждый бит шины: из-за небольших перекосов сигналы шины могут быть захвачены на разных нарастающих фронтах тактового сигнала приемного тактового домена (например, разные параметры триггеров разрядов шины)
- Кроме того значение сигнала триггера вышедшего из метастабильного состояния не определено (т.е. может быть либо 0, либо 1) и результирующее значение шины может быть некорректным
- В этом случае удобно использовать FIFO с независимыми тактовыми сигналами чтения и записи (Dual Clock FIFO)

Литература

- Дэвид М. Хэррис, Сара Л. Хэррис: “Цифровая схемотехника и архитектура компьютера”
- www.01signal.com/constraints/timing
- AMD Xilinx: “Vivado Design Suite User Guide: Synthesis (UG901)”
- AMD Xilinx: “7 Series FPGAs Memory Resources User Guide (UG473)”
- Clifford E. Cummings: “Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog”

Дополнительно:

- Clifford E. Cummings: “Simulation and Synthesis Techniques for Asynchronous FIFO Design”