

Функциональные модели шины и написание

Verification IP.

**Конвейерные интерфейсы с
внеочередным порядком
ответов.**

Занятие №20
1 апреля 2023



ШКОЛА СИНТЕЗА
ЦИФРОВЫХ СХЕМ

ПРИ ПАРТНЕРСТВЕ





Сергей Чусов
Инженер по верификации

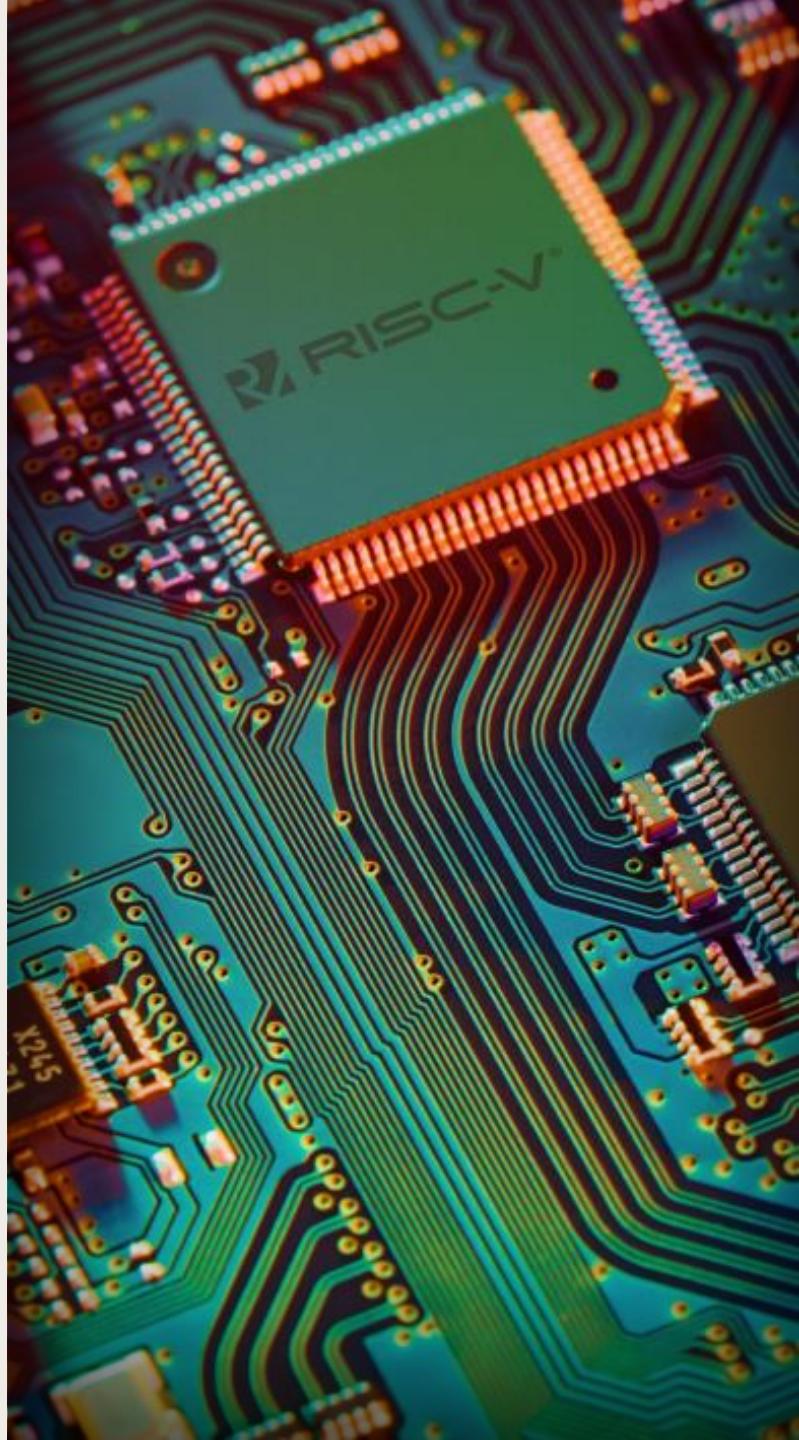
Специалист по верификации аппаратного
обеспечения

Сотрудник лаборатории НИЛ-ЭСК-МИЭТ

4 года опыта верификации СнК, процессорных
ядер, сетей на кристалле, гетерогенных
многопоточных систем

СТРУКТУРА ЗАНЯТИЯ

- Разбор домашнего задания предыдущей лекции
- Повторение предыдущих лекций
- Вступление
 - Верификация СнК
 - Все про Verification Intellectual Property (VIP)
 - VIP стандартных интерфейсов
- Основная часть
 - Введение в стандартную шину AXI4
 - Необходимая теория SystemVerilog
 - Разбор AXI4 VIP
- Задание

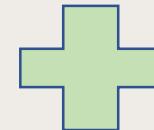
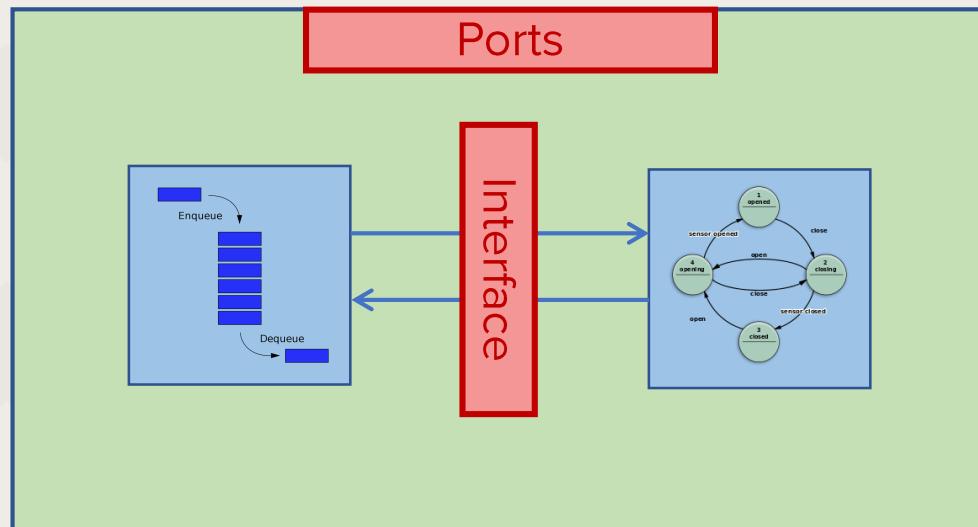


Разбор домашнего задания предыдущей лекции

Разбор домашнего задания предыдущей лекции

<gitflic.ru/project/yuri-panchul/valid-ready-etc>

boards/omdazz/07_assert_and_cover/30_fifo_assert_and_cover



Functional
Coverage
Model

fifo_coverage.sv

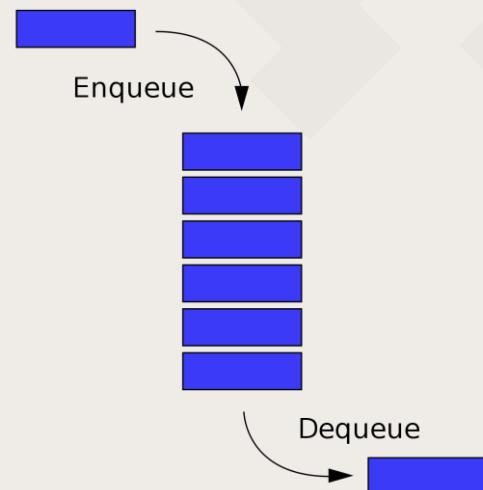
Разбор домашнего задания предыдущей лекции

- coverpoint для full и empty.

```
// TODO: add coverpoints for empty and full
```

```
coverpoint empty {
    bins empty      = { 1 };
    bins not_empty = { 0 };
}
coverpoint full   {
    bins full       = { 1 };
    bins not_full  = { 0 };
}
```

FIFO

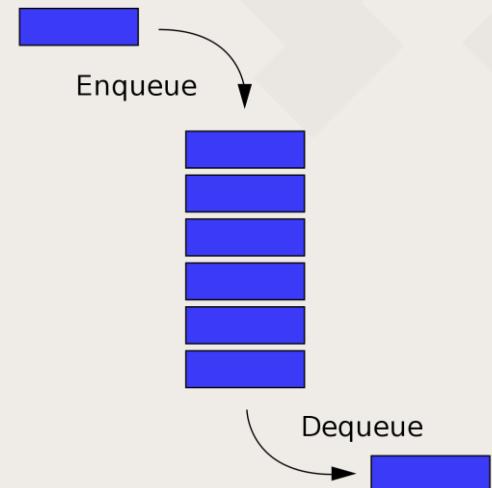


Разбор домашнего задания предыдущей лекции

- `illegal_bins` для `cross`.

```
cross push, pop, empty, full {  
    ...  
    // TODO: add other illegal cross bins  
  
    illegal_bins illegal_push_full  
    = binsof(push) intersect { 1 }  
    && binsof(full) intersect { 1 };  
  
    illegal_bins illegal_empty_full  
    = binsof(empty) intersect { 1 }  
    && binsof(full) intersect { 1 };  
}
```

FIFO

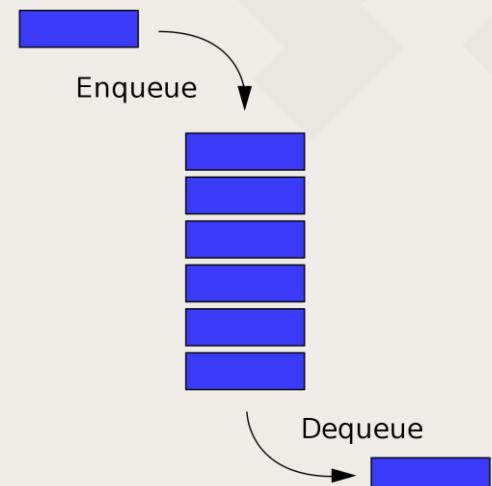


Разбор домашнего задания предыдущей лекции

- Изменения сигнала `push`.

```
push_transitions: coverpoint push {  
    ...  
    // TODO: add bins for push_101 and push_1001  
  
    bins push_101 = ( 1 => 0 => 1 );  
    bins push_1001 = ( 1 => 0 => 0 => 1 );  
  
    ...  
}
```

FIFO



Разбор домашнего задания предыдущей лекции

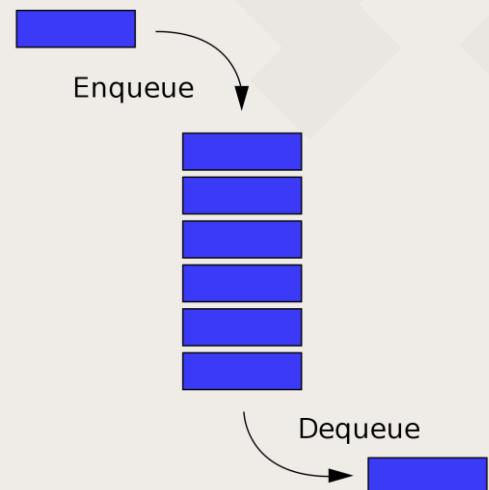
- **coverpoint** разности указателей на запись и чтение.

```
// TODO: implement wr_after_rd_pointer_distance

wr_after_rd_pointer_distance: coverpoint wr_ptr - rd_ptr
    iff (wr_ptr >= rd_ptr) {
        bins zero          = { 0 };
        bins one           = { 1 };
        bins max           = { depth - 1 };
        bins in_between [] = { [2 : depth - 2] };

        illegal_bins others = default;
    }
```

FIFO

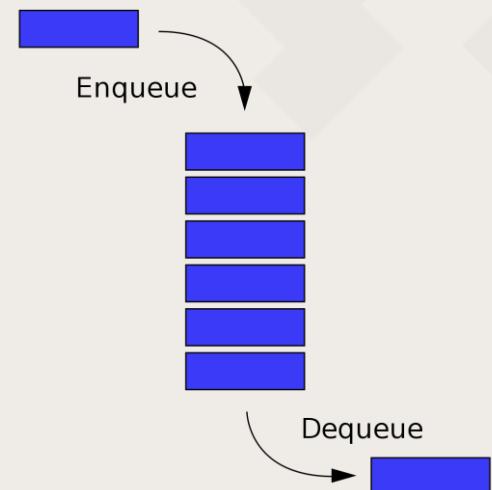


Разбор домашнего задания предыдущей лекции

- **coverpoint** разность счетчиков тактов **pop** и **push**.

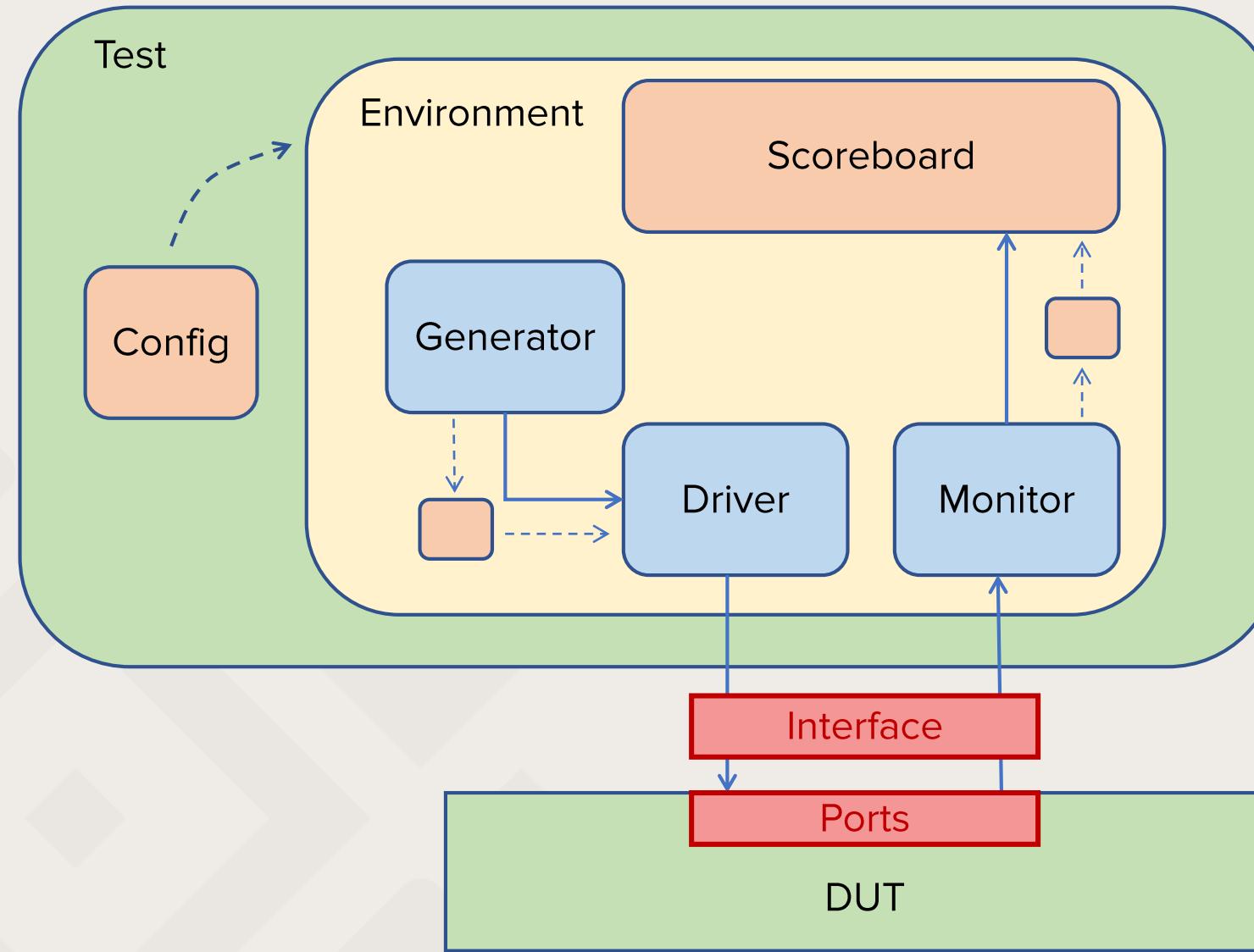
```
push_pop_distance: coverpoint cycle_pop - cycle_push {  
  
    // TODO: implement:  
    //  
    // pop_near_after_push  
    // push_far_from_pop  
    // pop_far_from_push  
    //  
    // A hint: use '$' for far distance  
  
    bins pop_near_after_push [] = { [ -5: -3] };  
    bins push_far_from_pop      = { [ 1: $] };  
    bins pop_far_from_push     = { [ $: -1] };  
}
```

FIFO

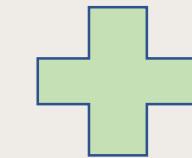


Повторение предыдущих лекций

Повторение предыдущих лекций: Constrained-random test

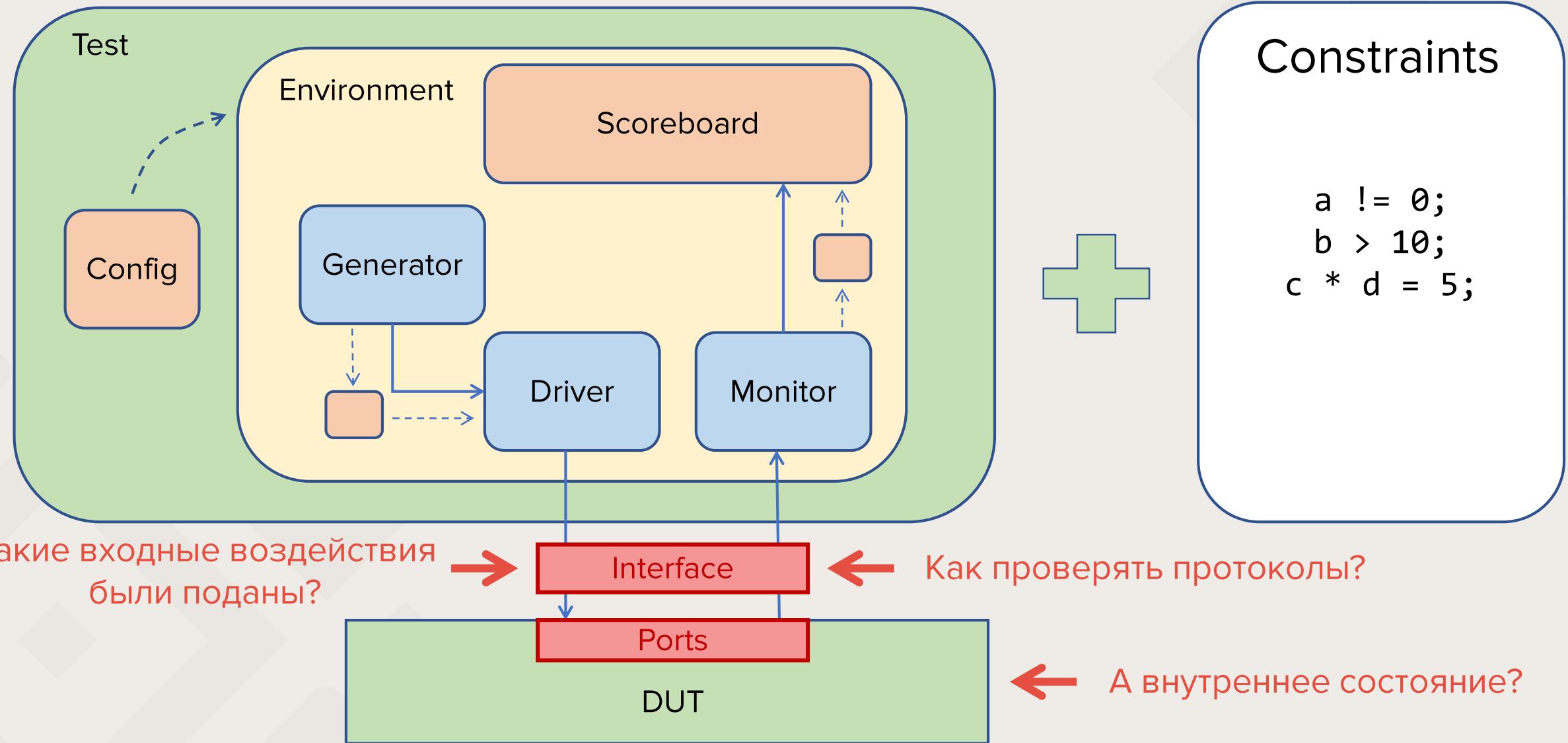


Constraints

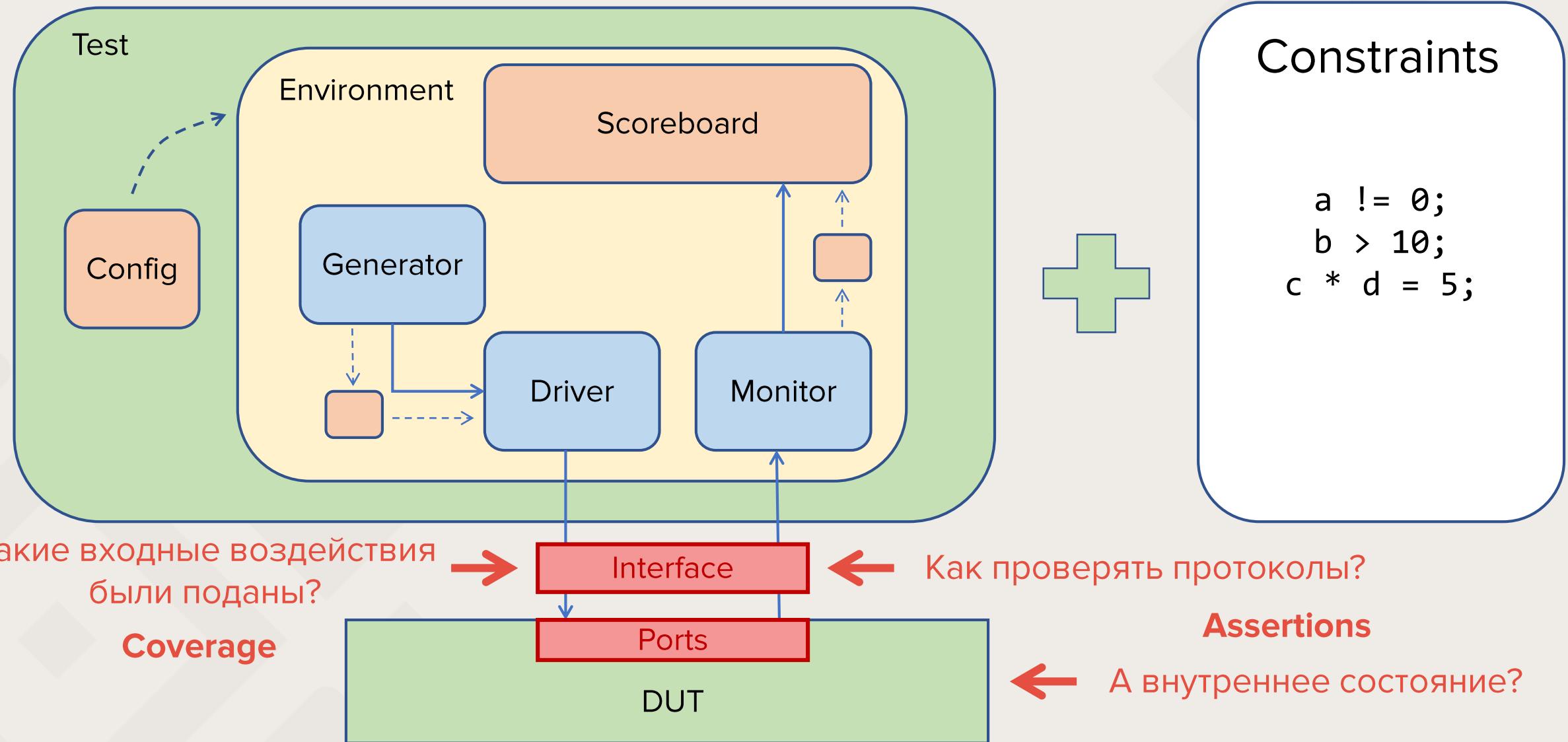


```
a != 0;  
b > 10;  
c * d = 5;
```

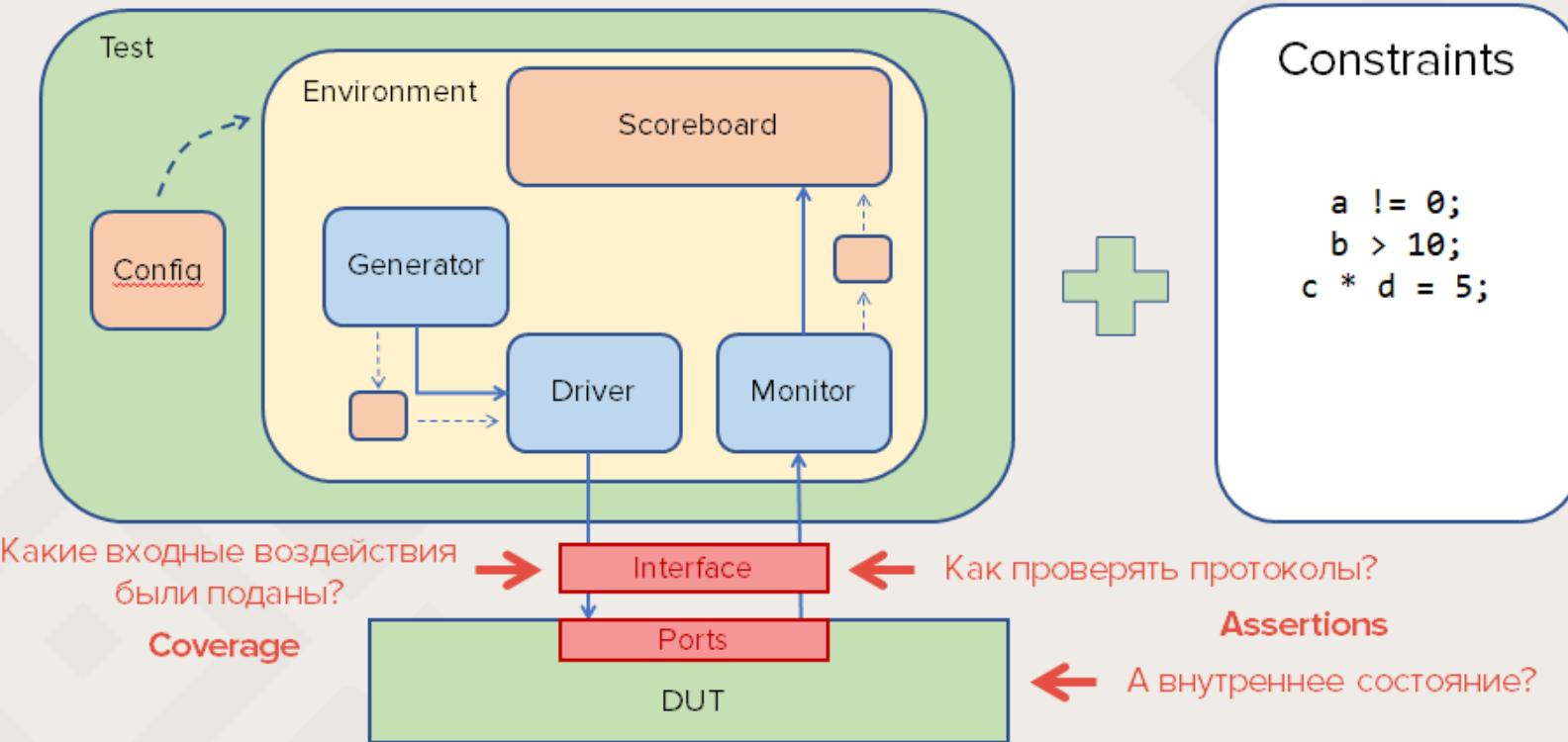
Повторение предыдущих лекций: Constrained-random test



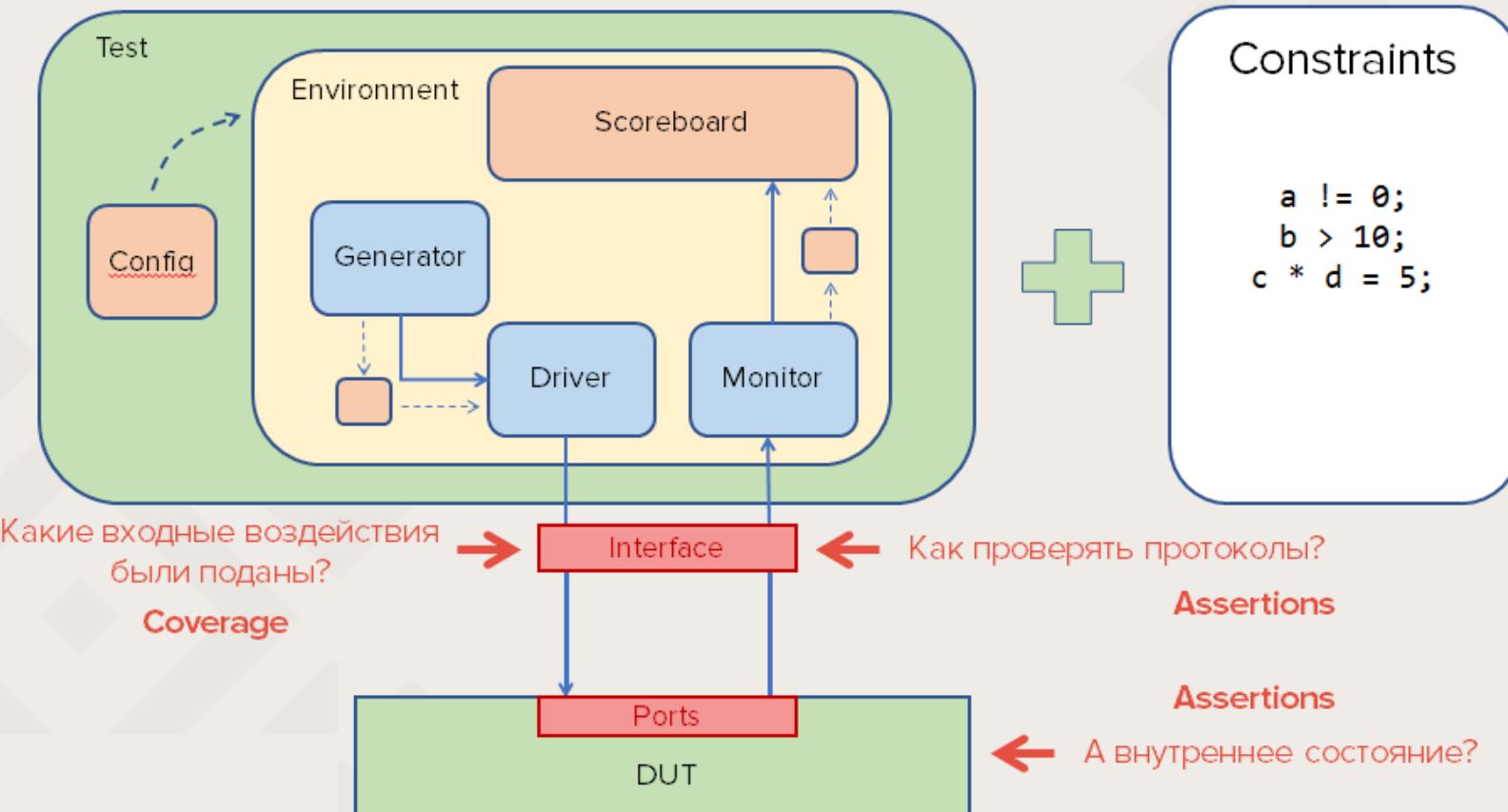
Повторение предыдущих лекций: Constrained-random test



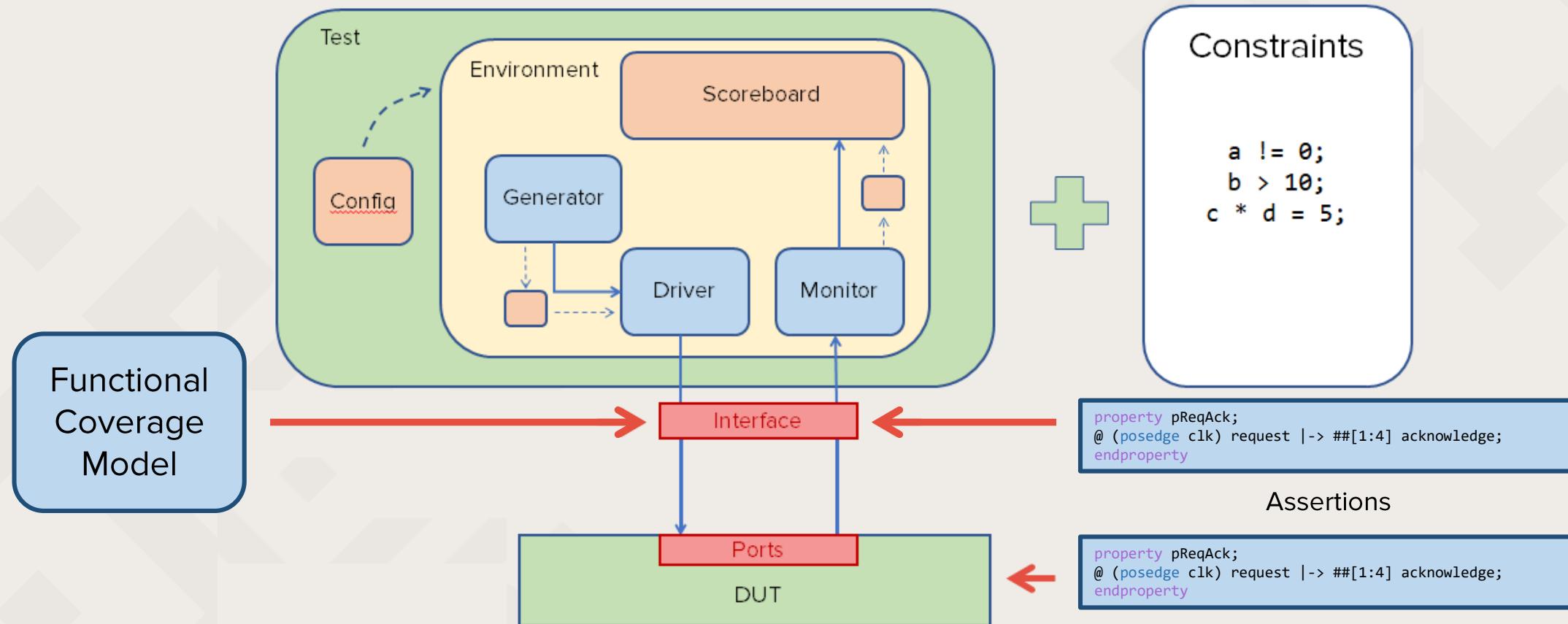
Повторение предыдущих лекций: Constrained-random test



Повторение предыдущих лекций: Constrained-random test

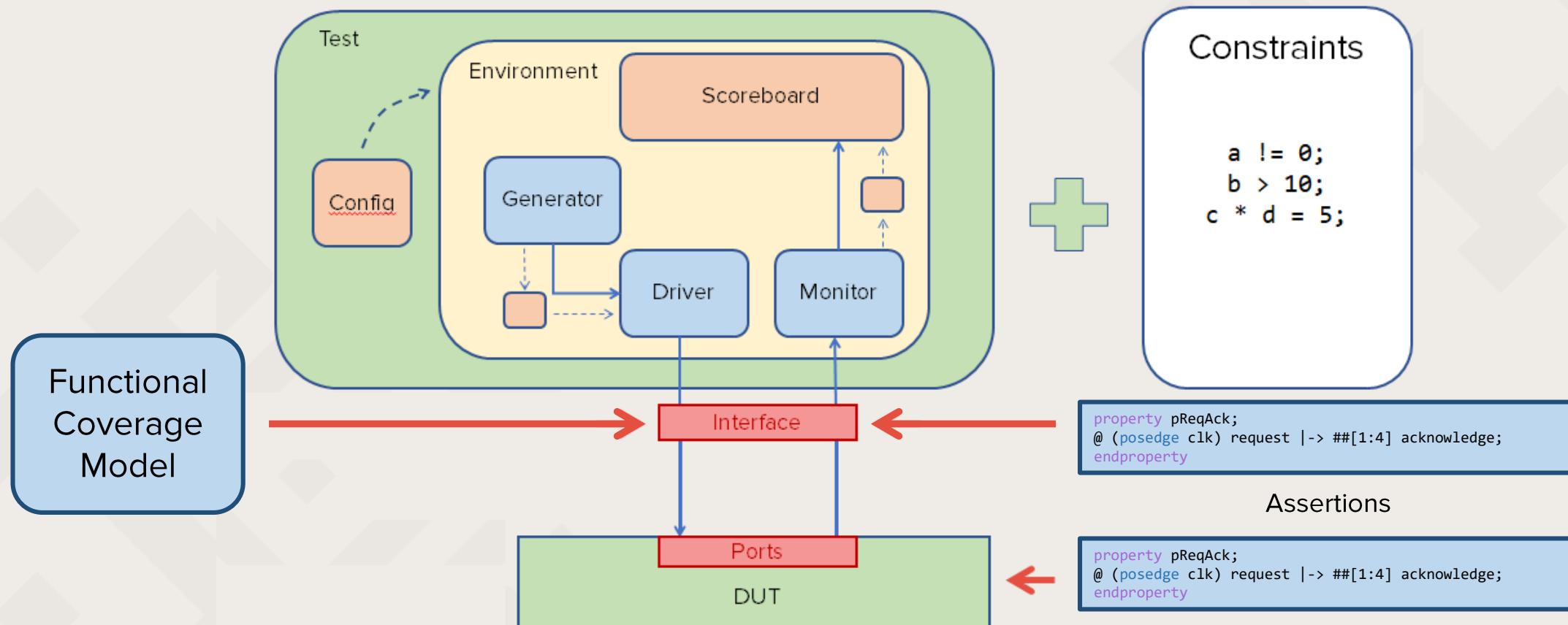


Повторение предыдущих лекций: ABV + Functional Coverage

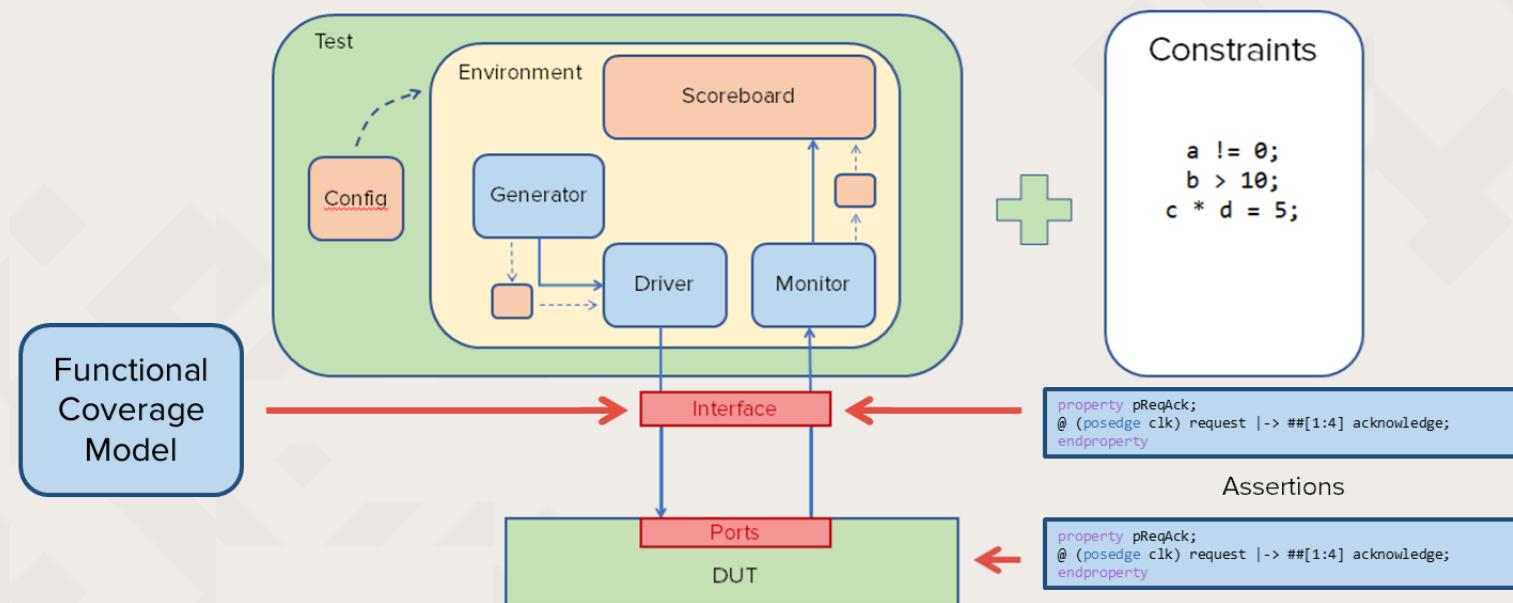


Вступление

ABV + Functional Coverage

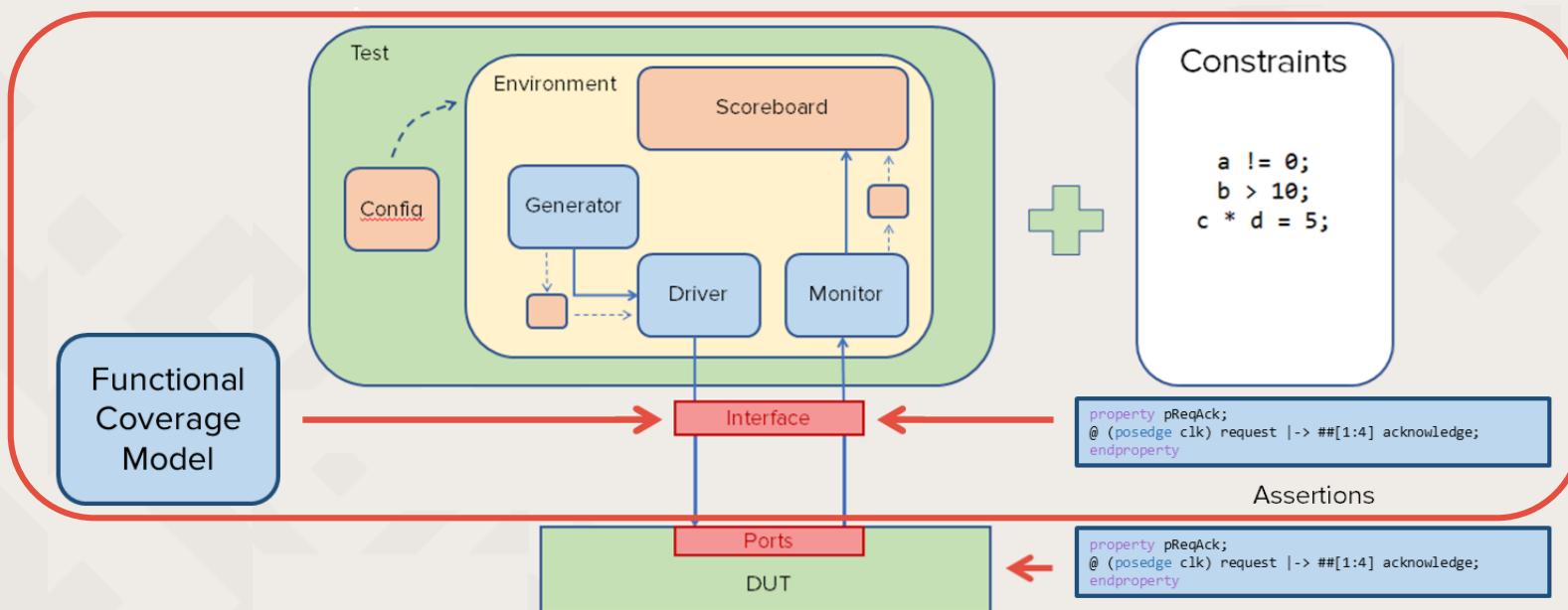


Verification Intellectual Property (VIP)



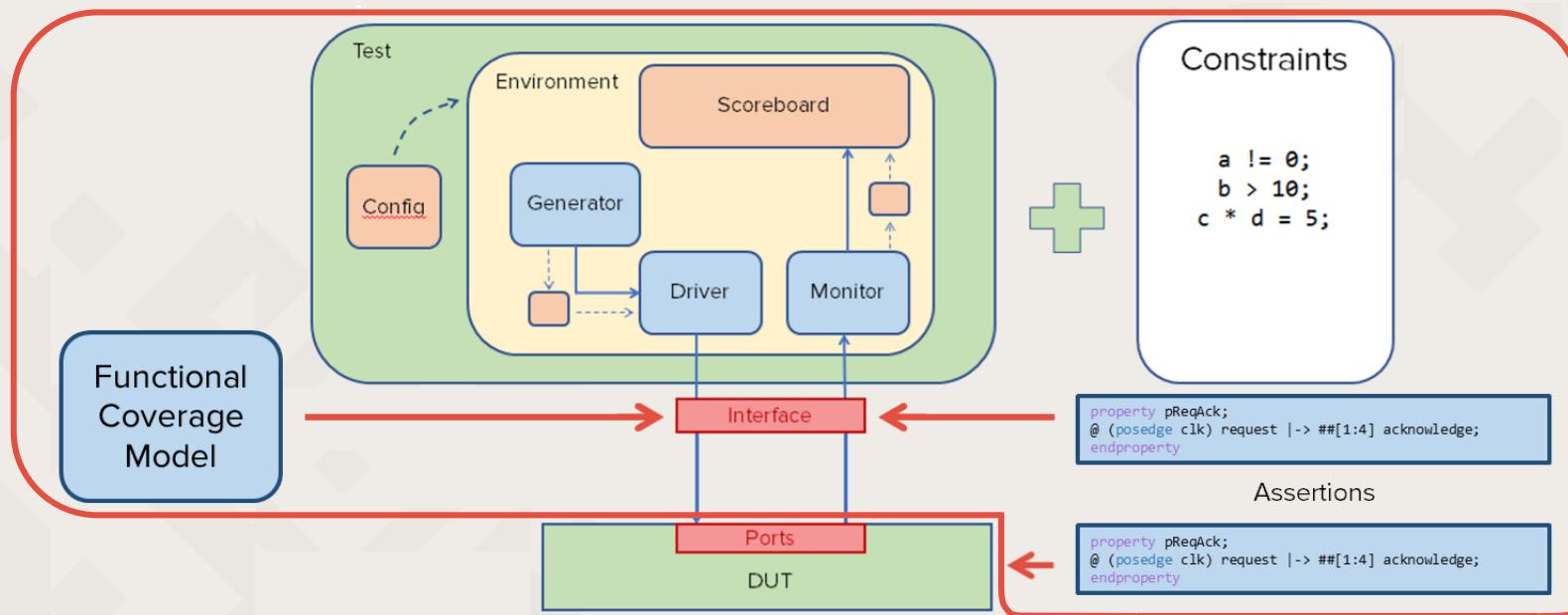
Verification Intellectual Property (VIP)

Verification Intellectual Property



Verification Intellectual Property (VIP)

Verification Intellectual Property

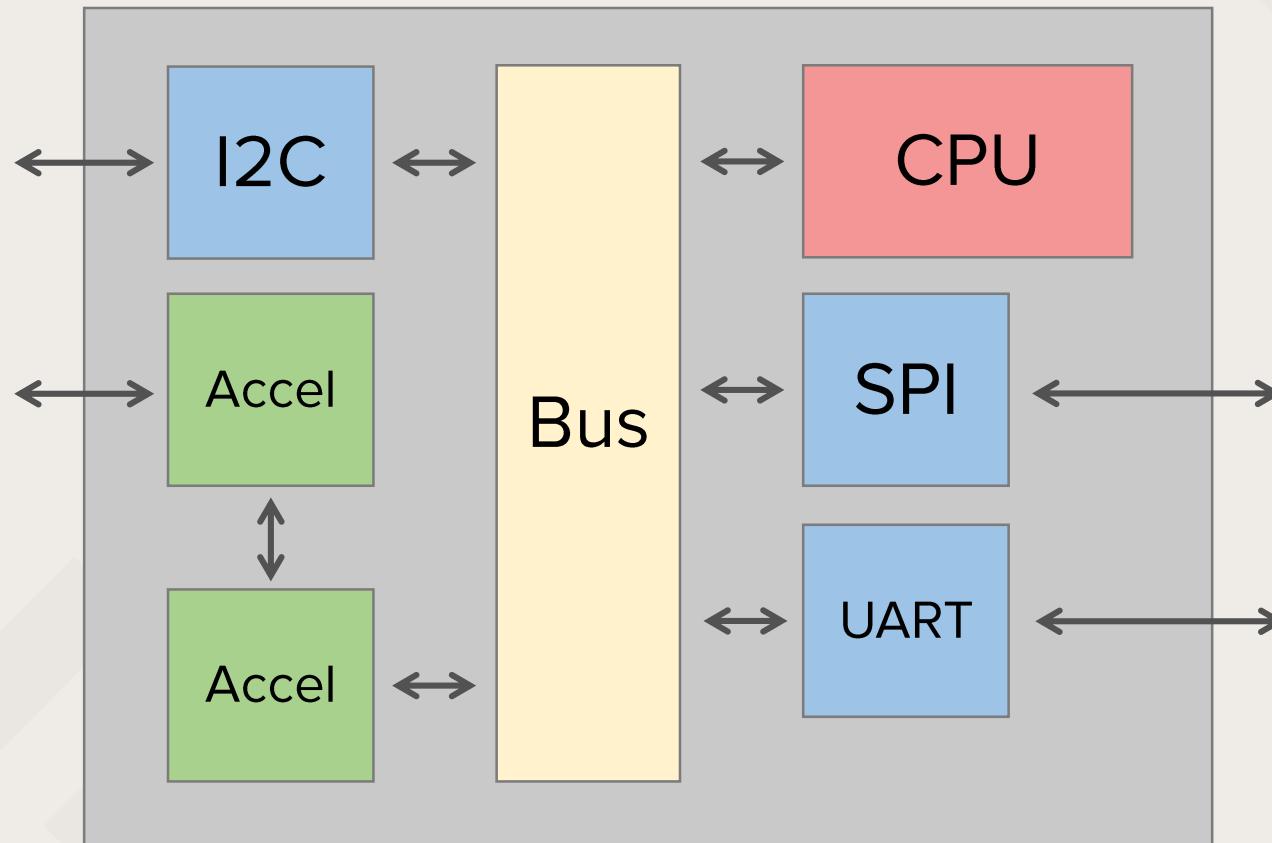


Использование VIP

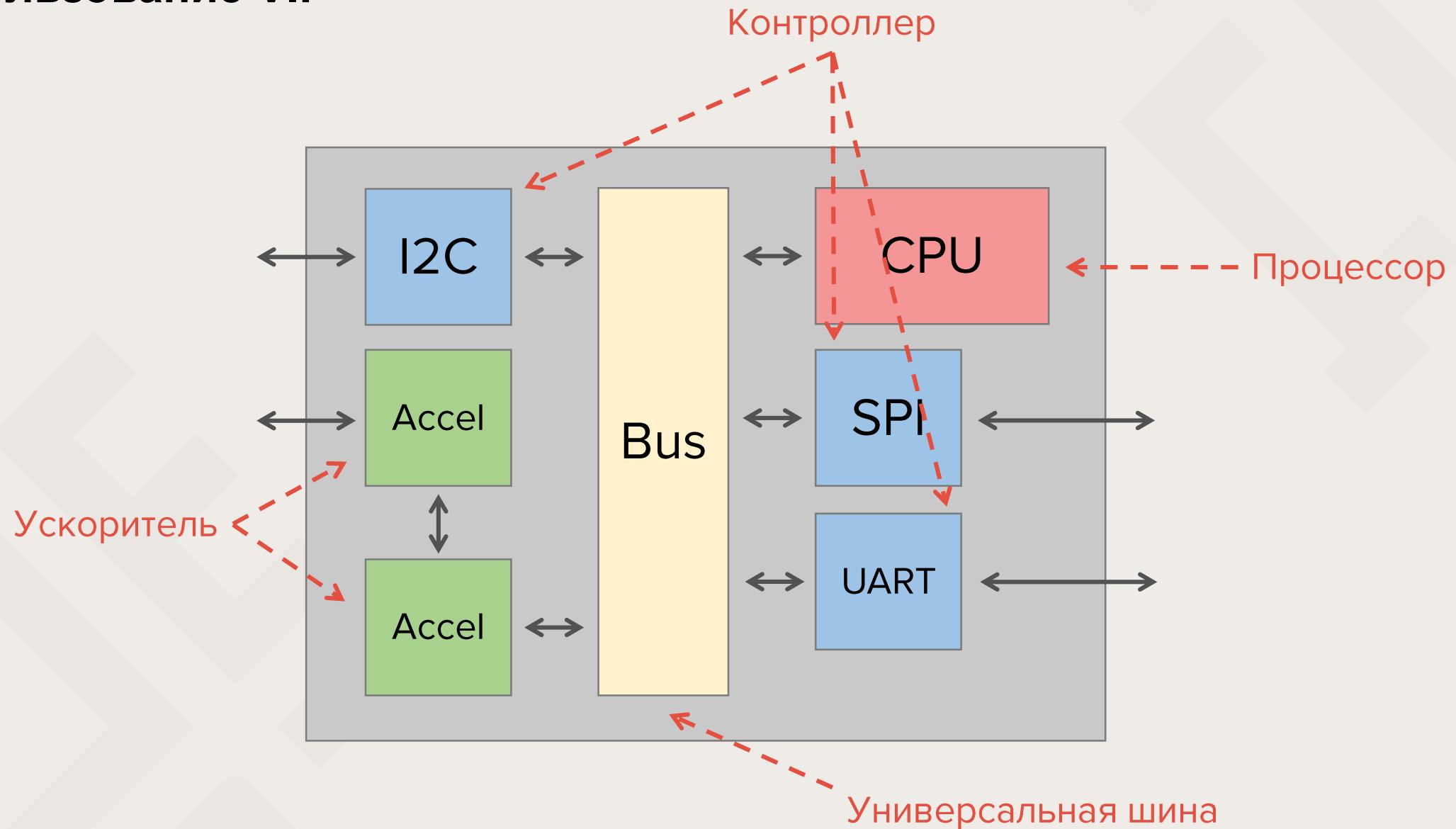
Использование VIP



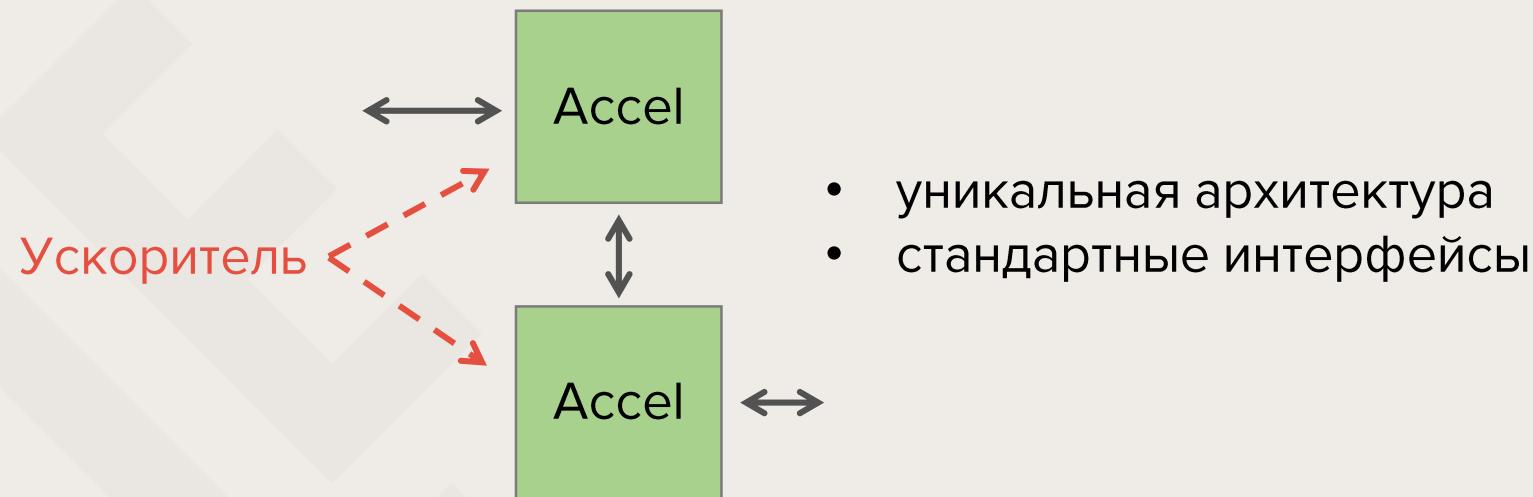
Использование VIP



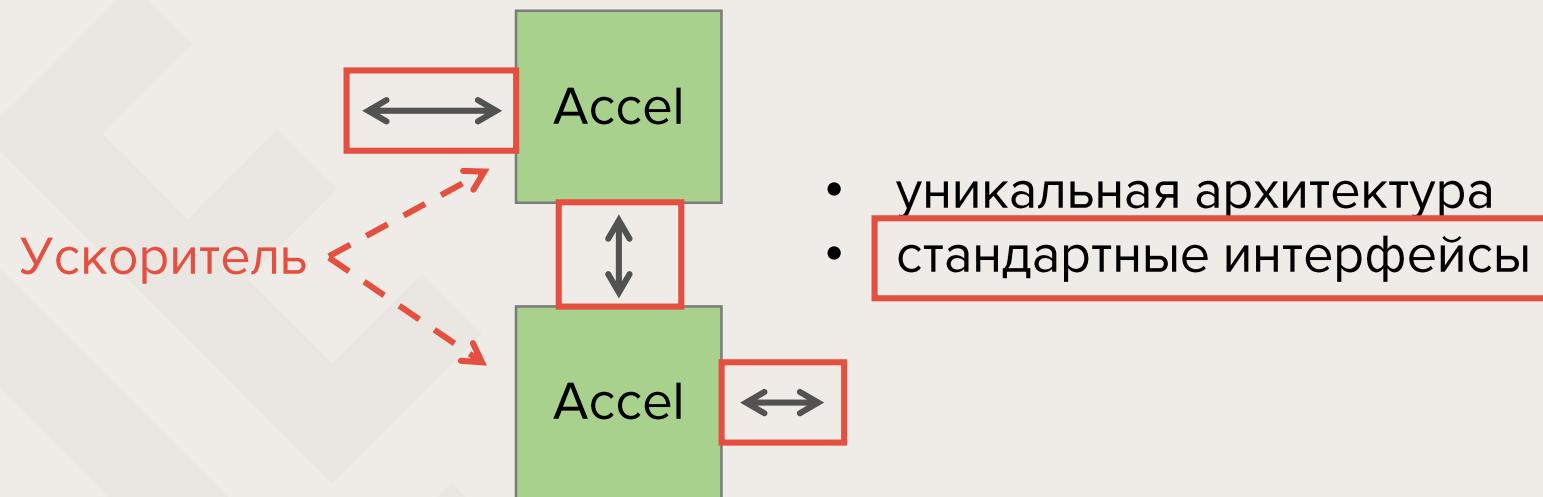
Использование VIP



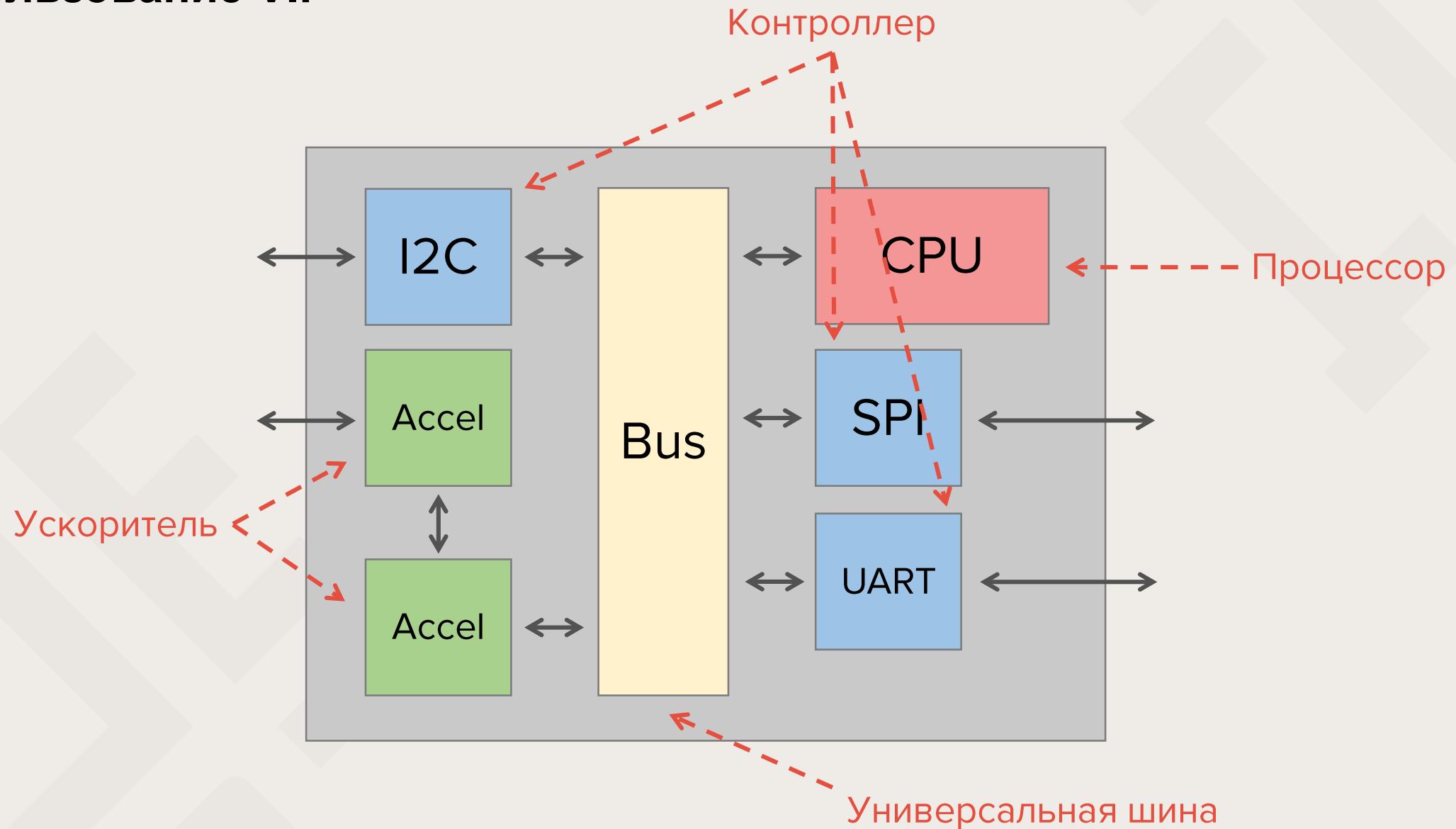
Использование VIP



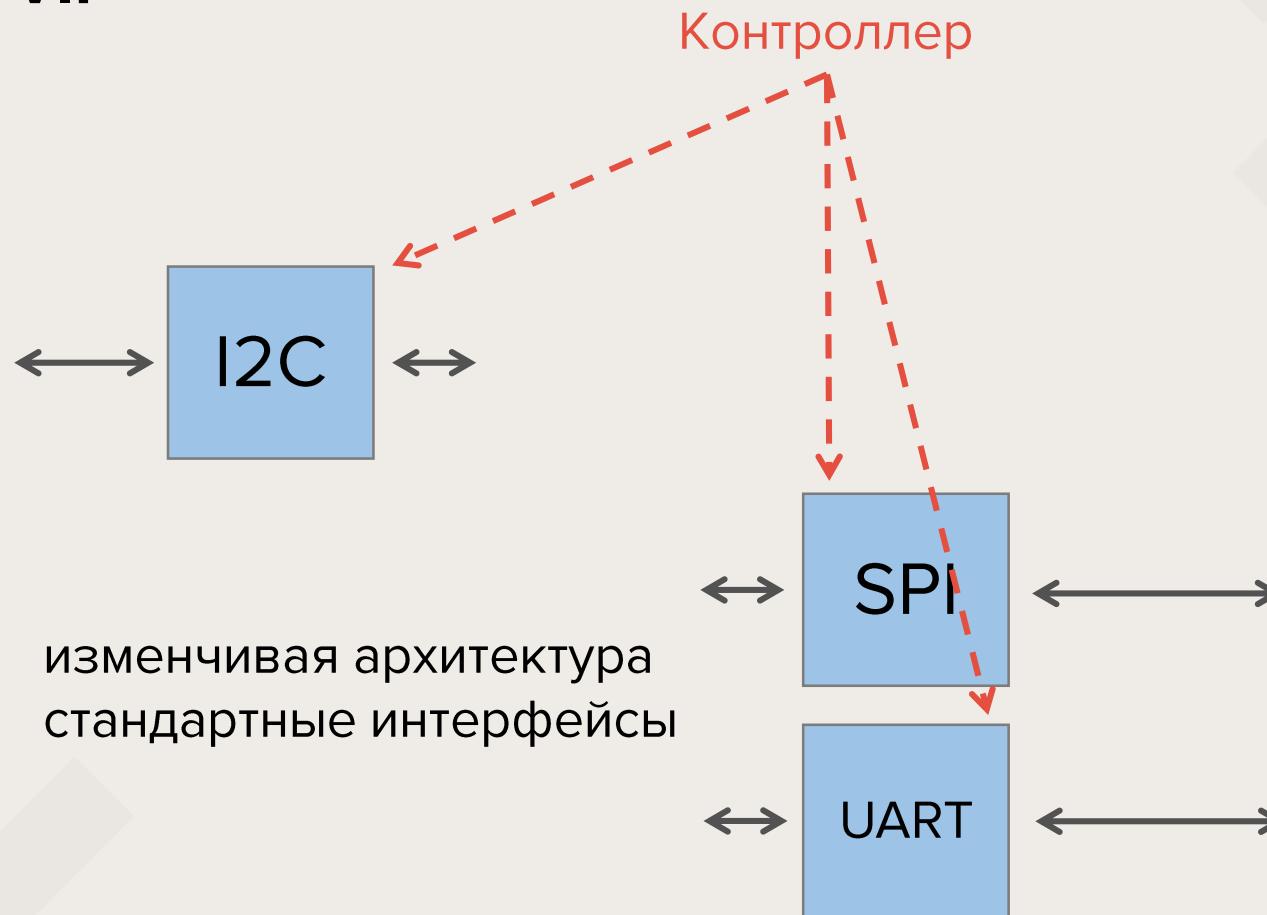
Использование VIP



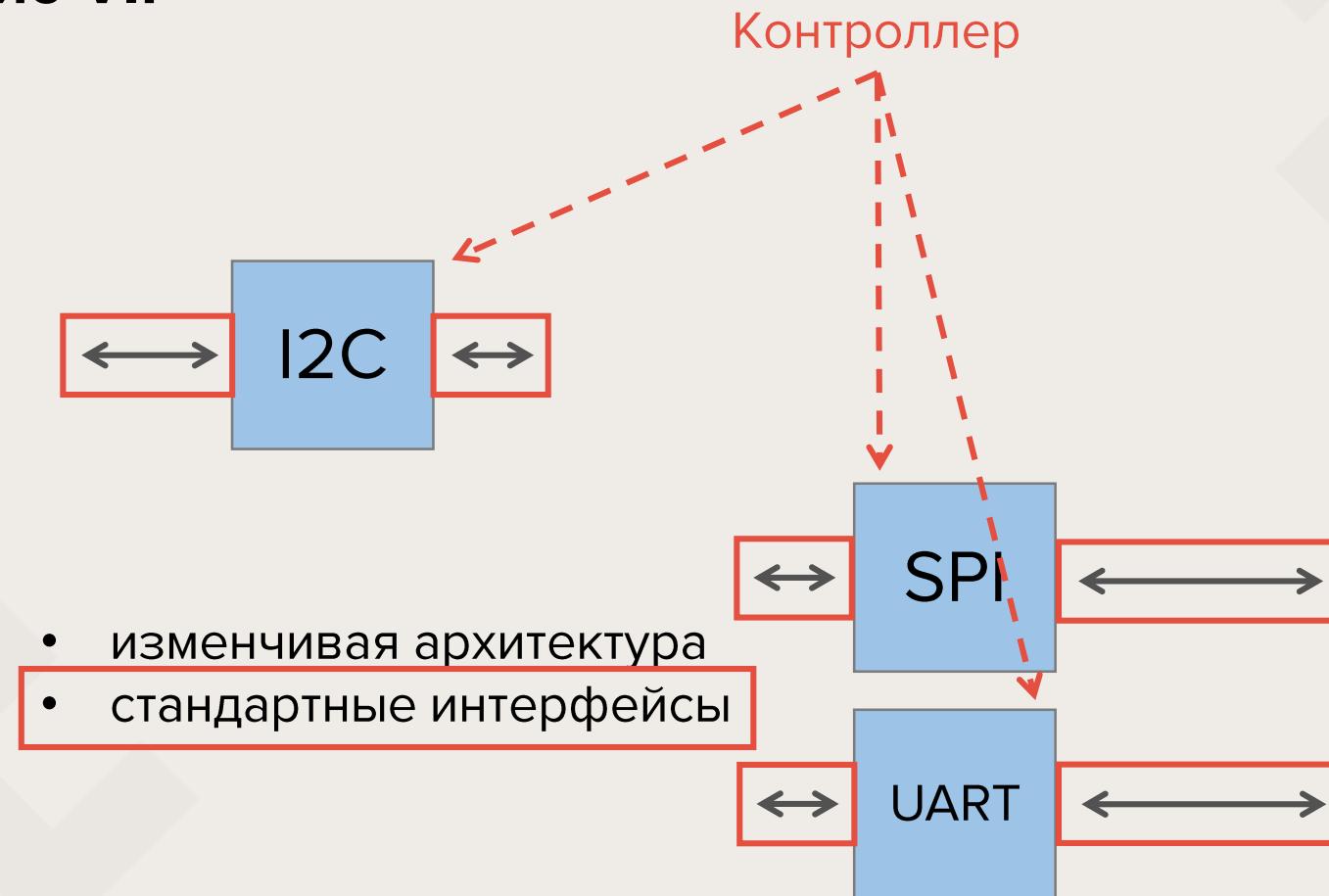
Использование VIP



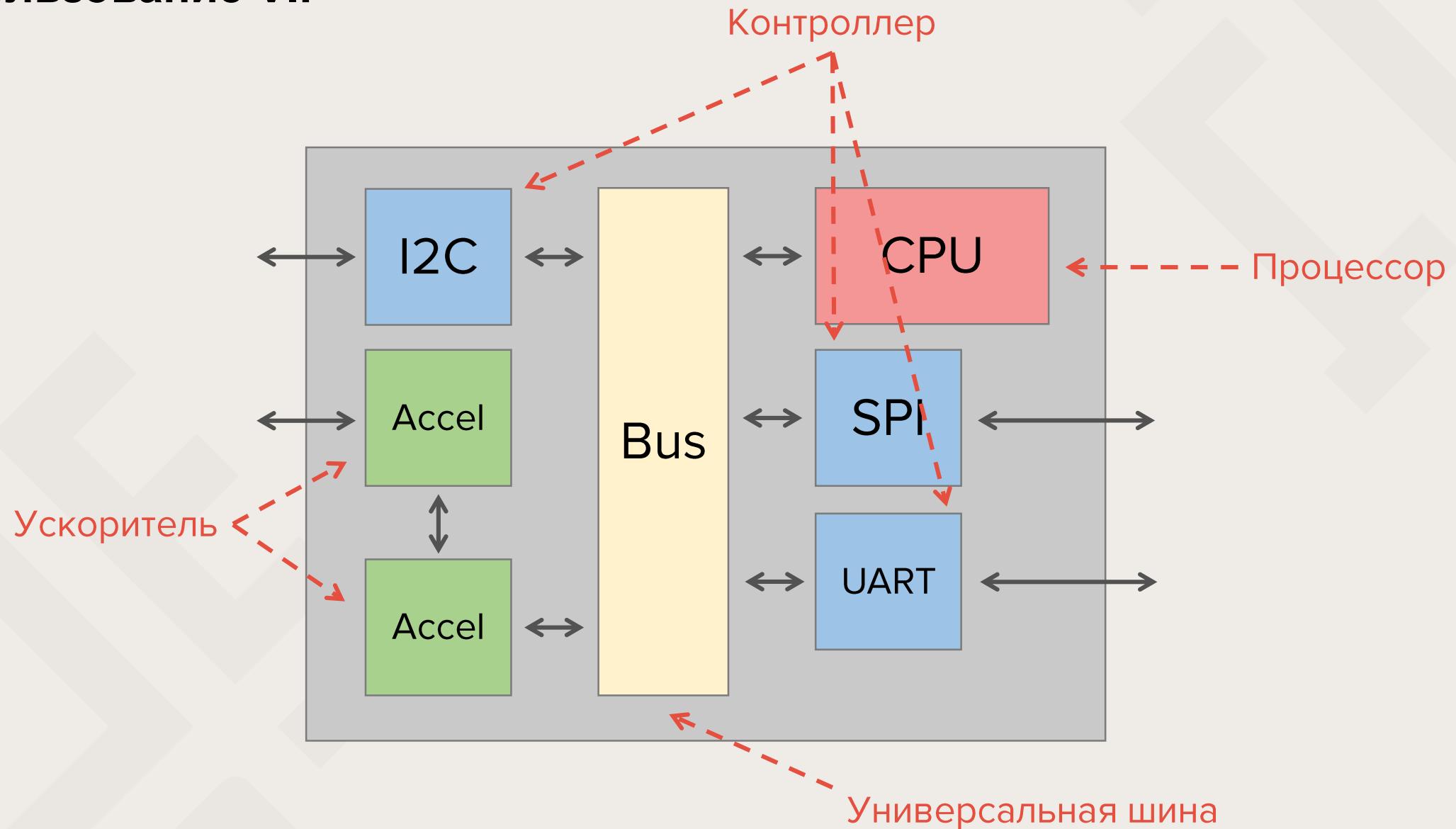
Использование VIP



Использование VIP

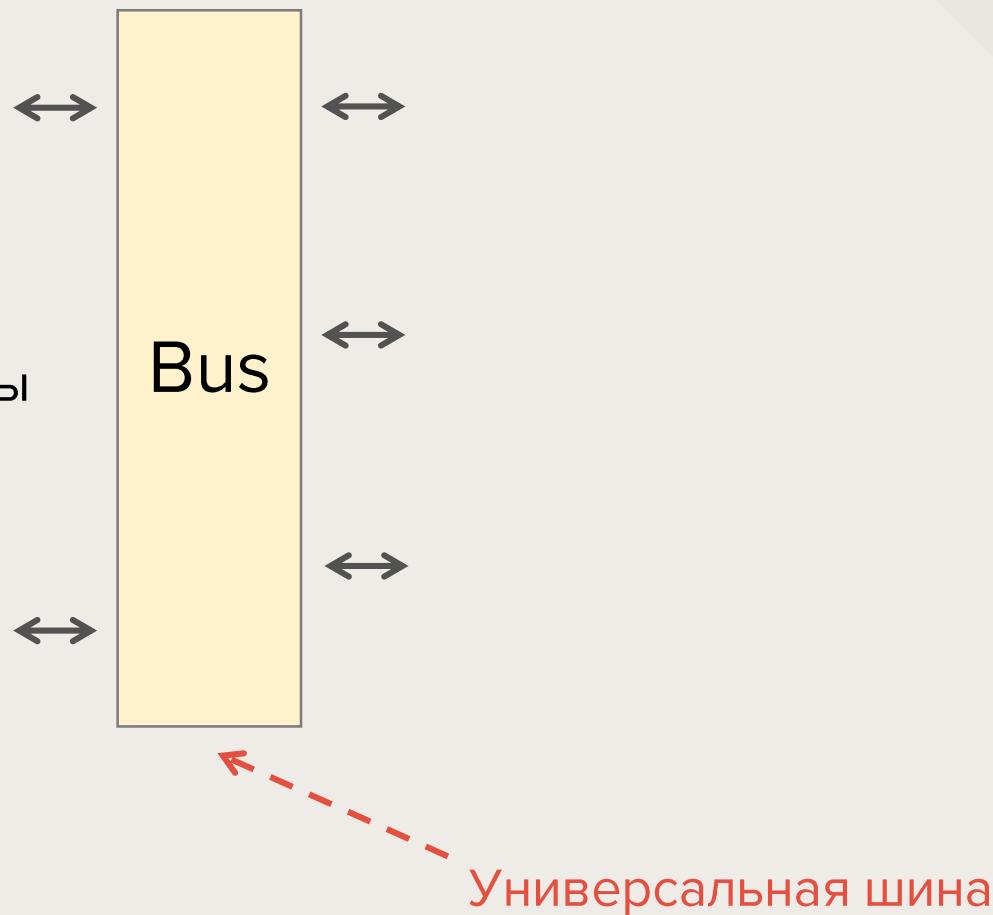


Использование VIP



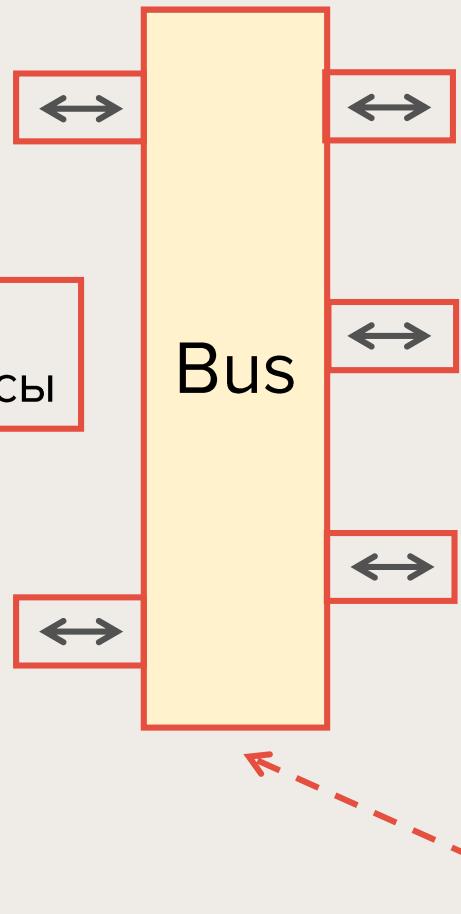
Использование VIP

- известная архитектура
- стандартные интерфейсы

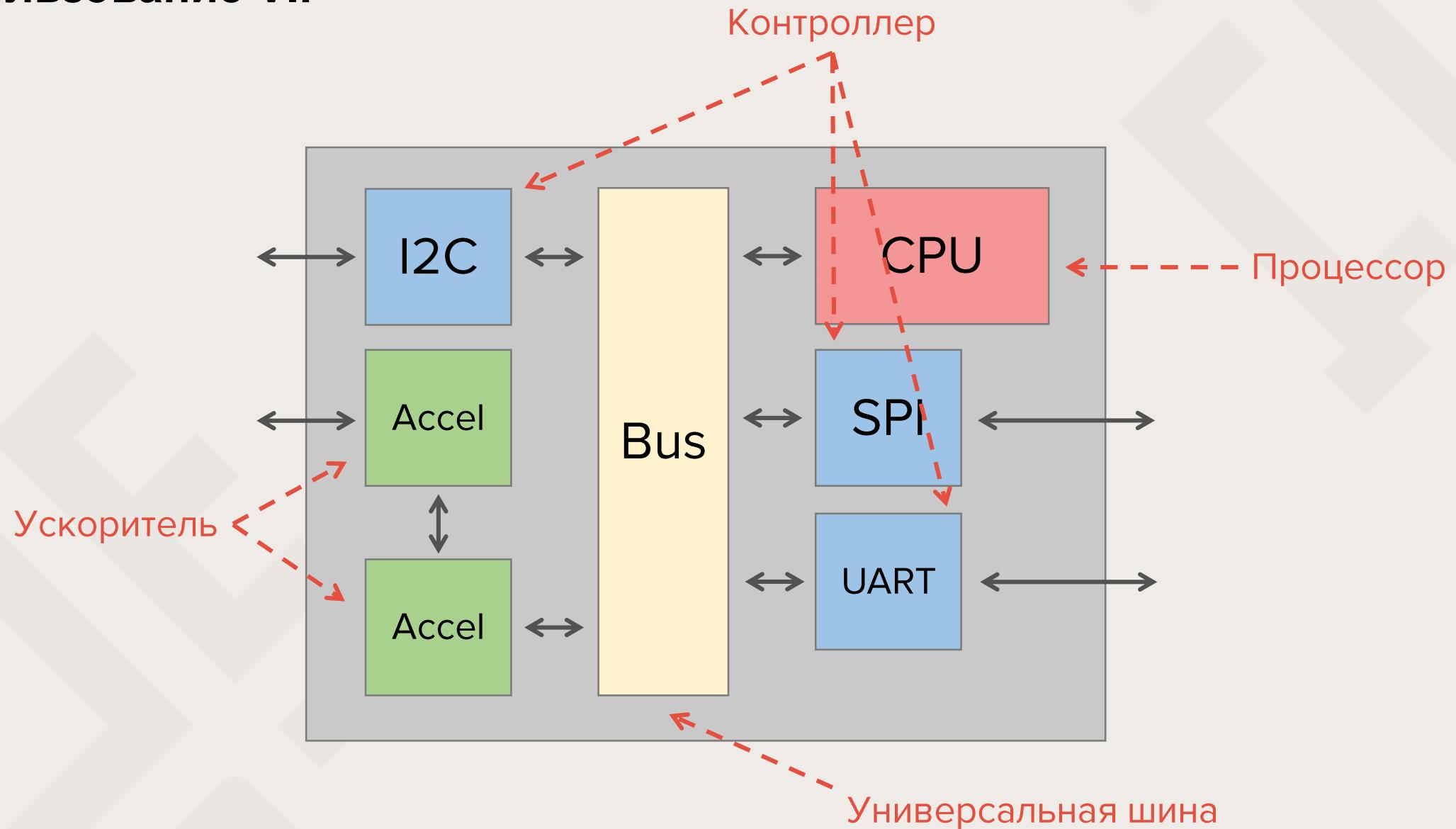


Использование VIP

- известная архитектура
- стандартные интерфейсы

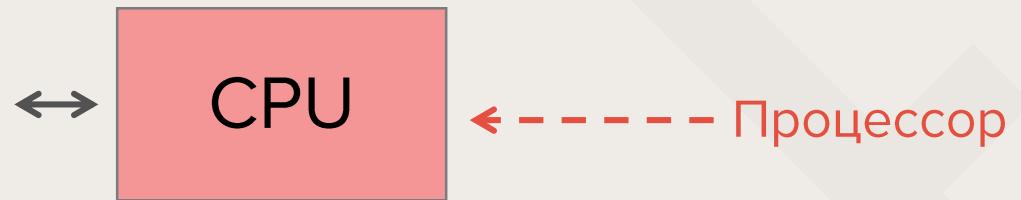


Использование VIP



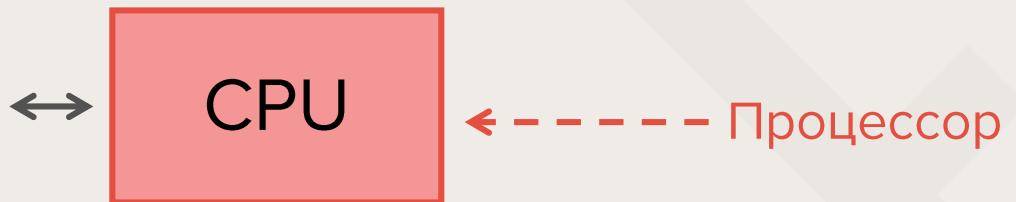
Использование VIP

- известная архитектура
- изменчивые интерфейсы

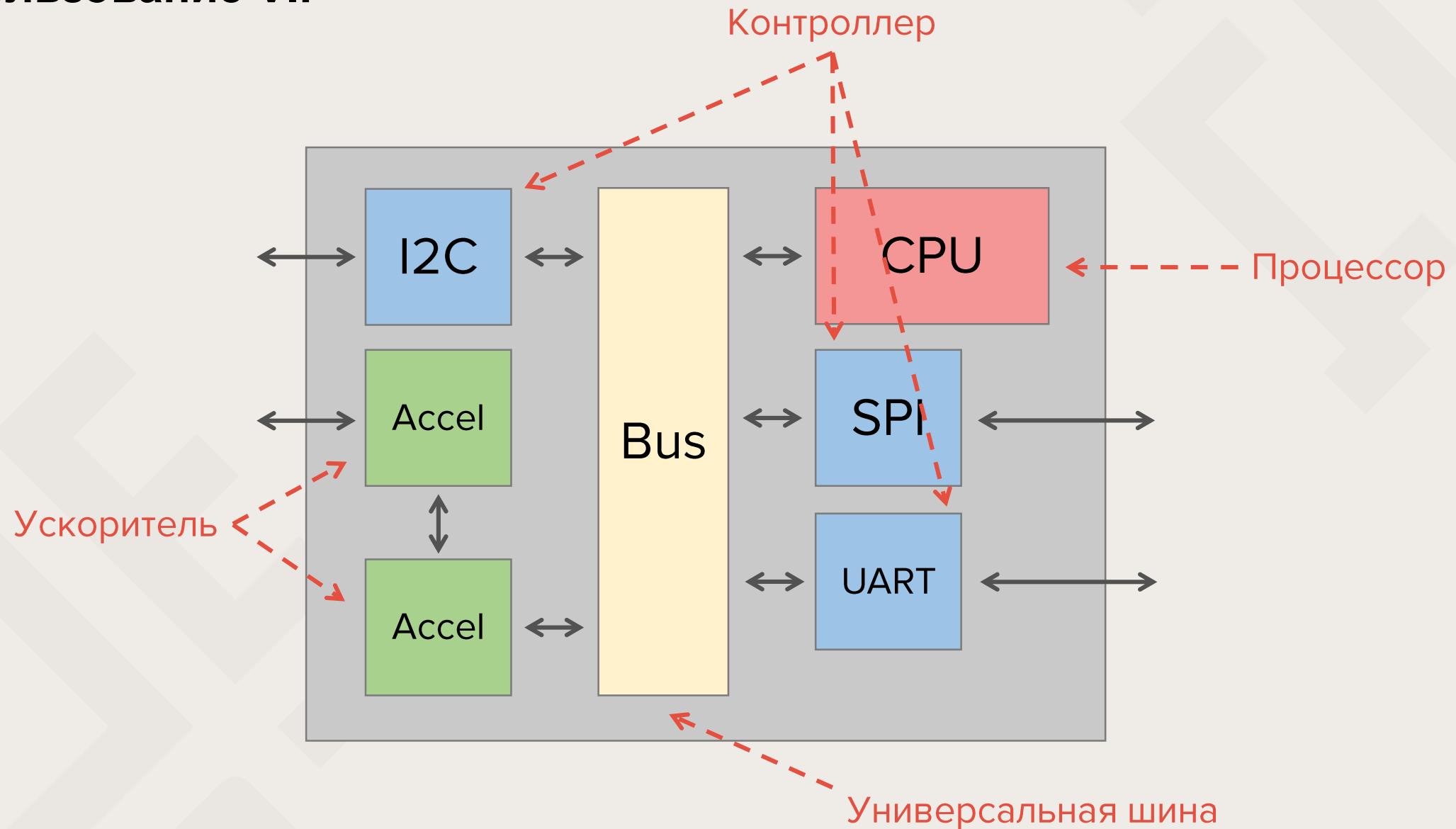


Использование VIP

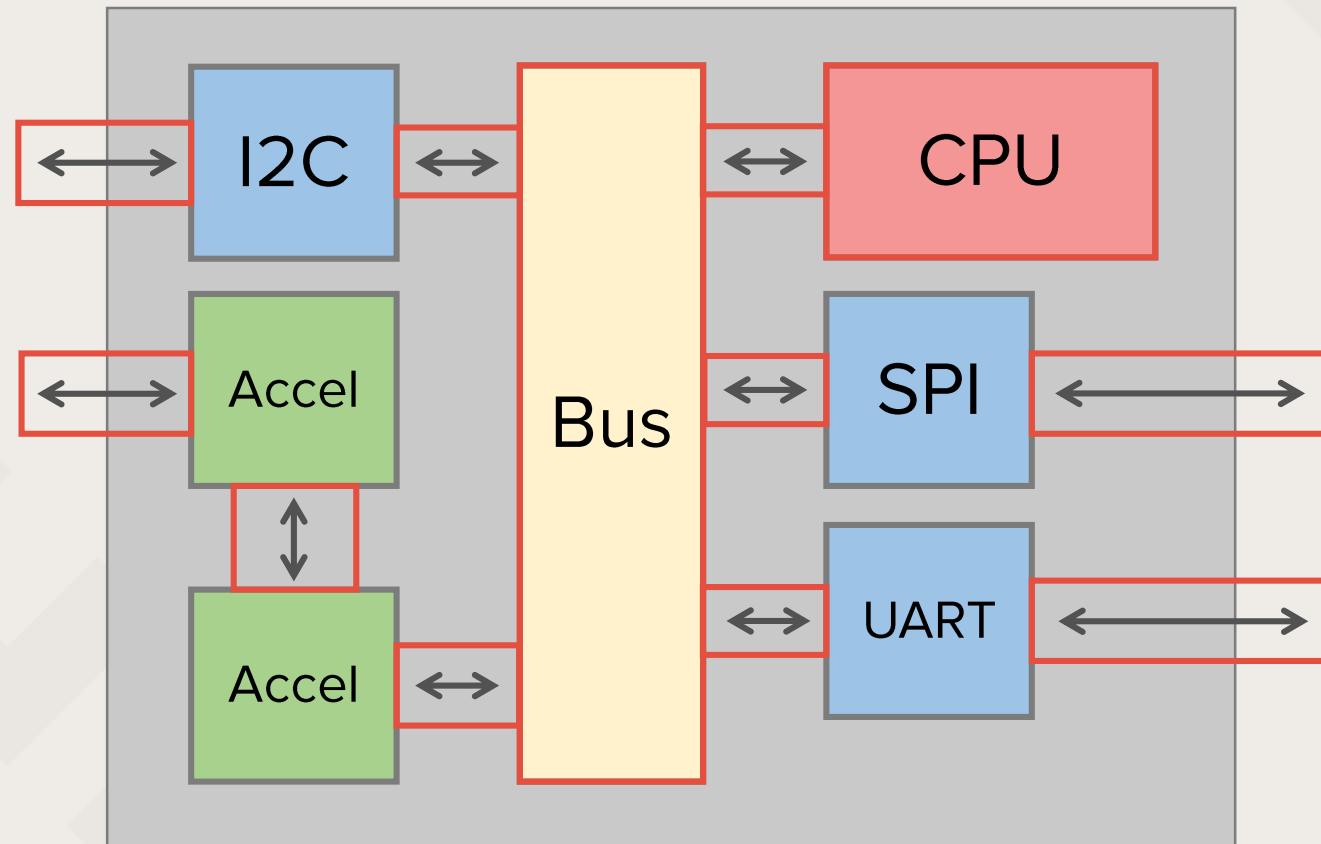
- известная архитектура
- изменчивые интерфейсы



Использование VIP



Использование VIP



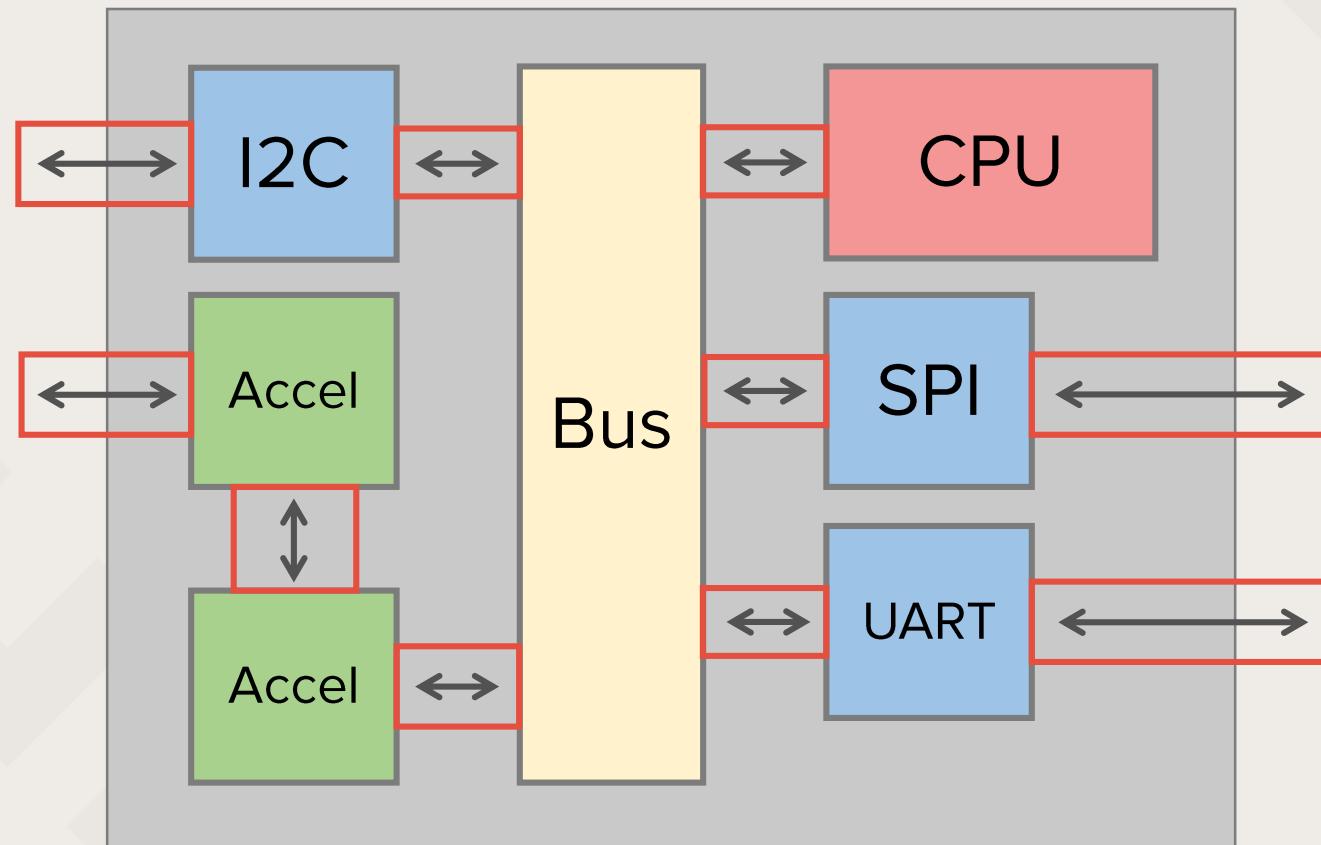
VIP!

Использование VIP

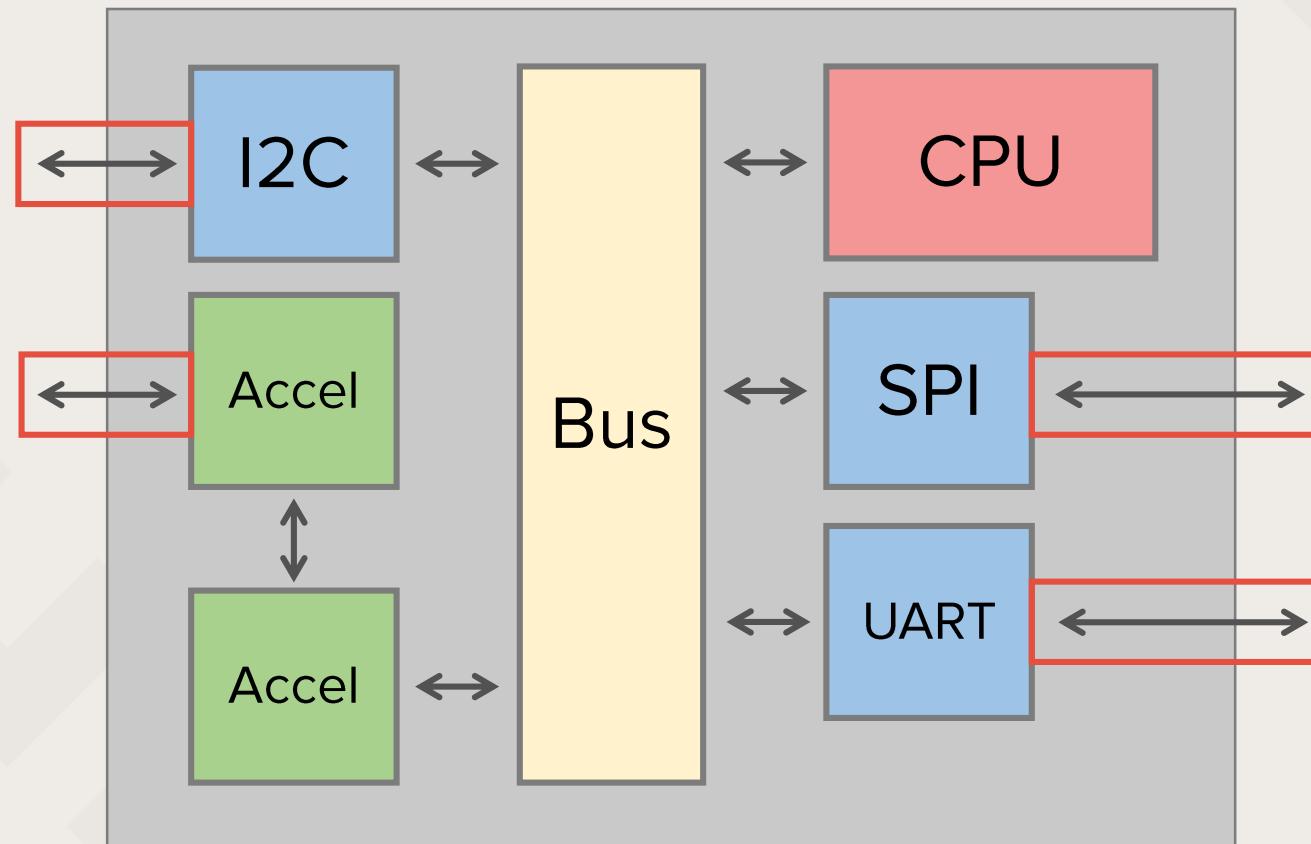
- В современных СнК **основные области использования VIP:**
 - Универсальные интерфейсы
 - Процессорные ядра
- Ограничение в **распространенности использования.**

Стандартные интерфейсы СнК

Интерфейсы



Интерфейсы: Внешние

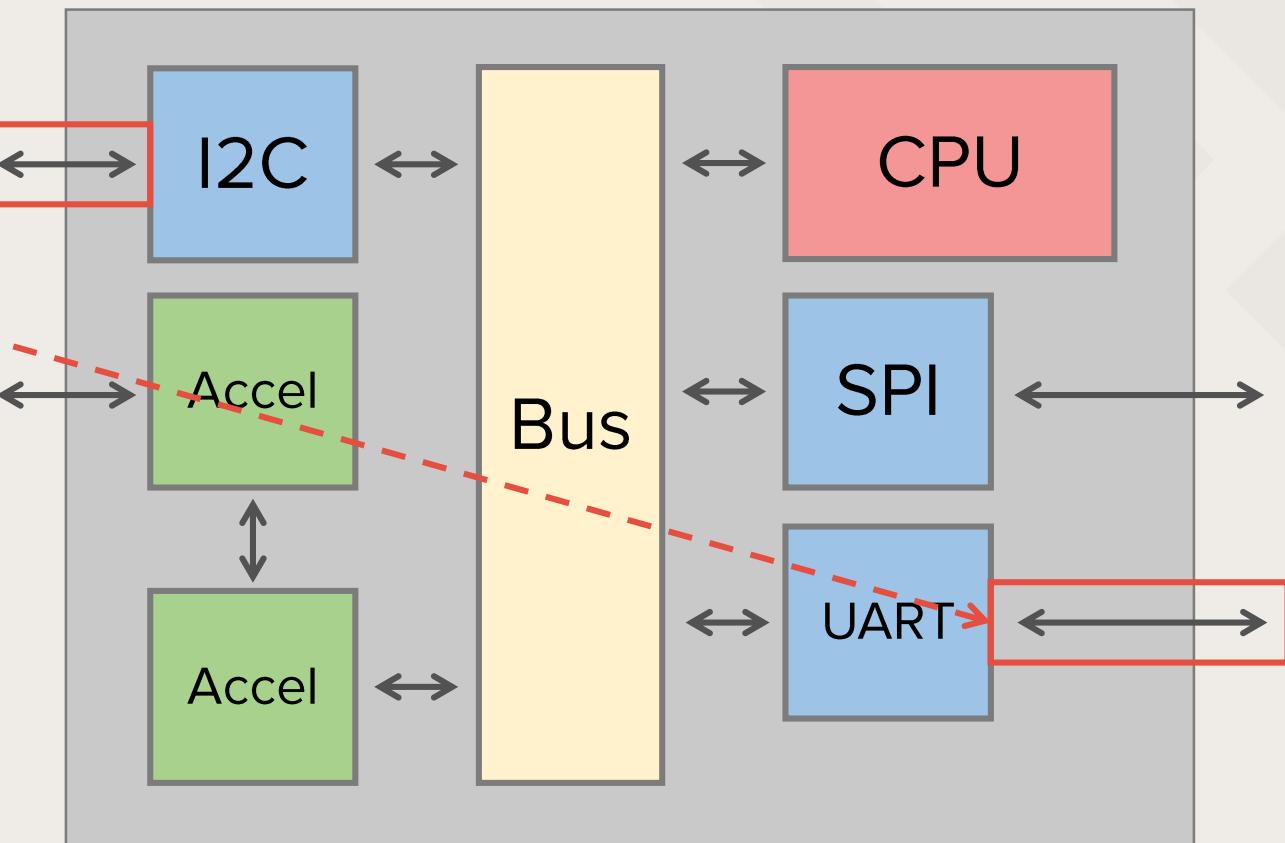


Интерфейсы: Внешние

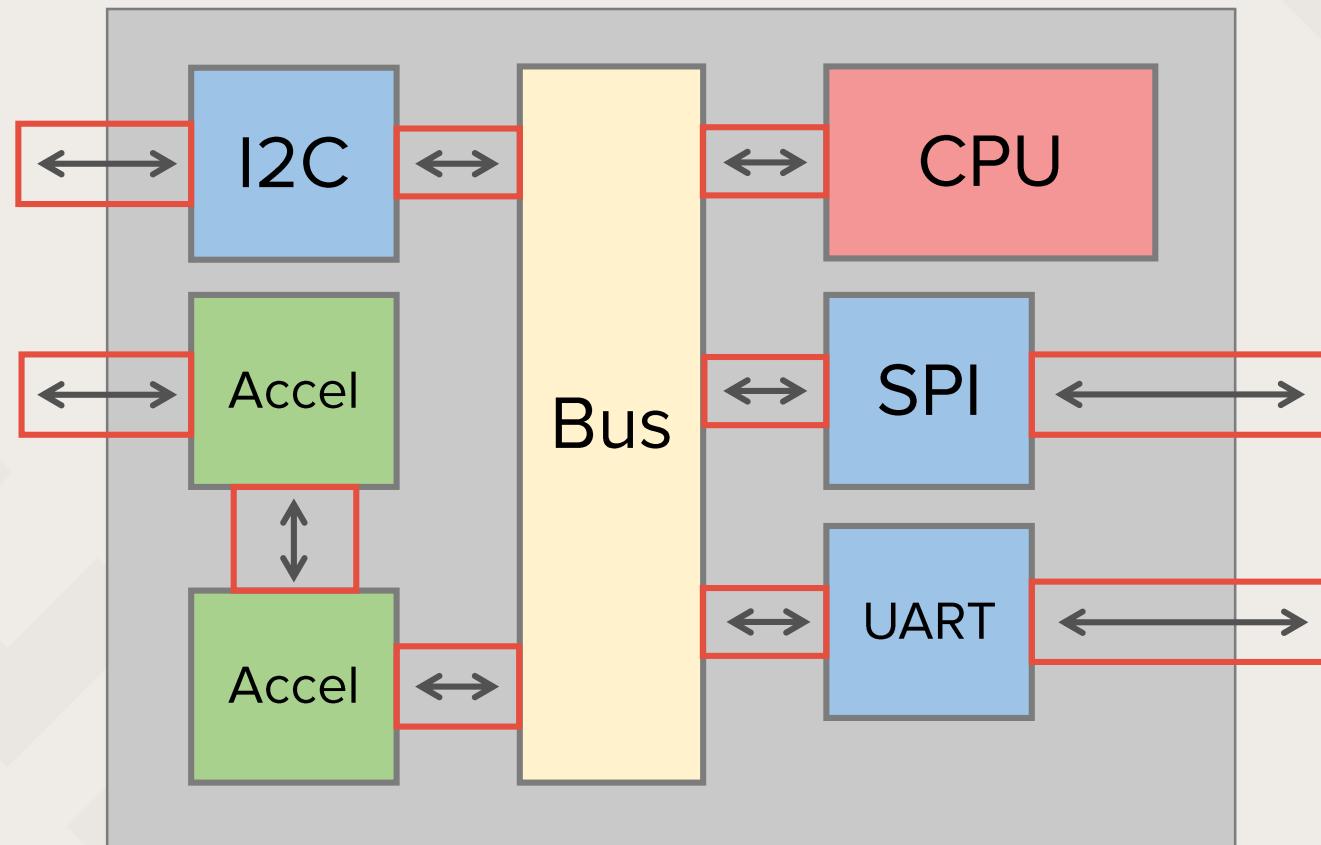
- Последовательные:
 - UART
 - SPI
 - I2C
 - ...
- Параллельные:
 - SATA
 - PCI Express
 - ...

Интерфейсы: Внешние

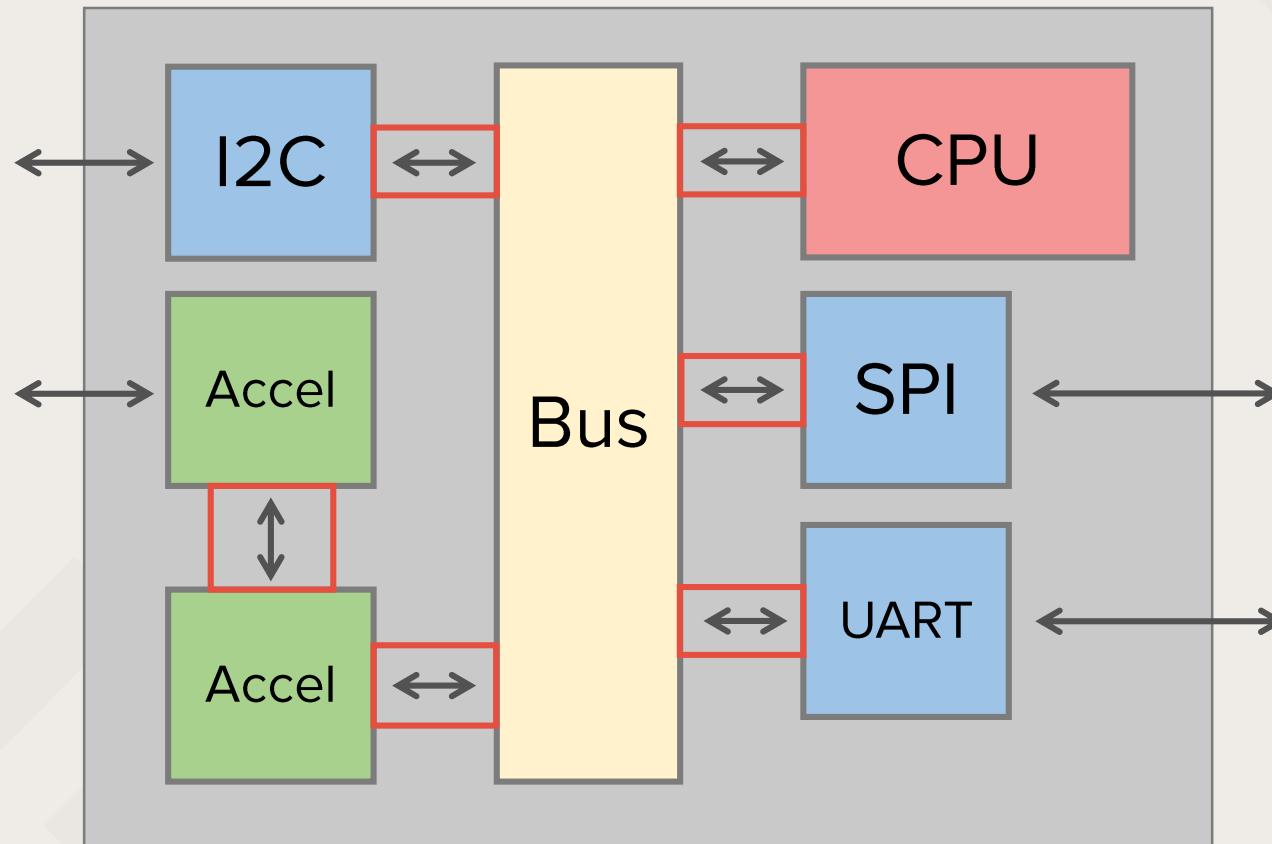
- Последовательные:
 - **UART**
 - SPI
 - **I2C**
 - ...
- Параллельные:
 - SATA
 - PIC Express
 - ...



Интерфейсы

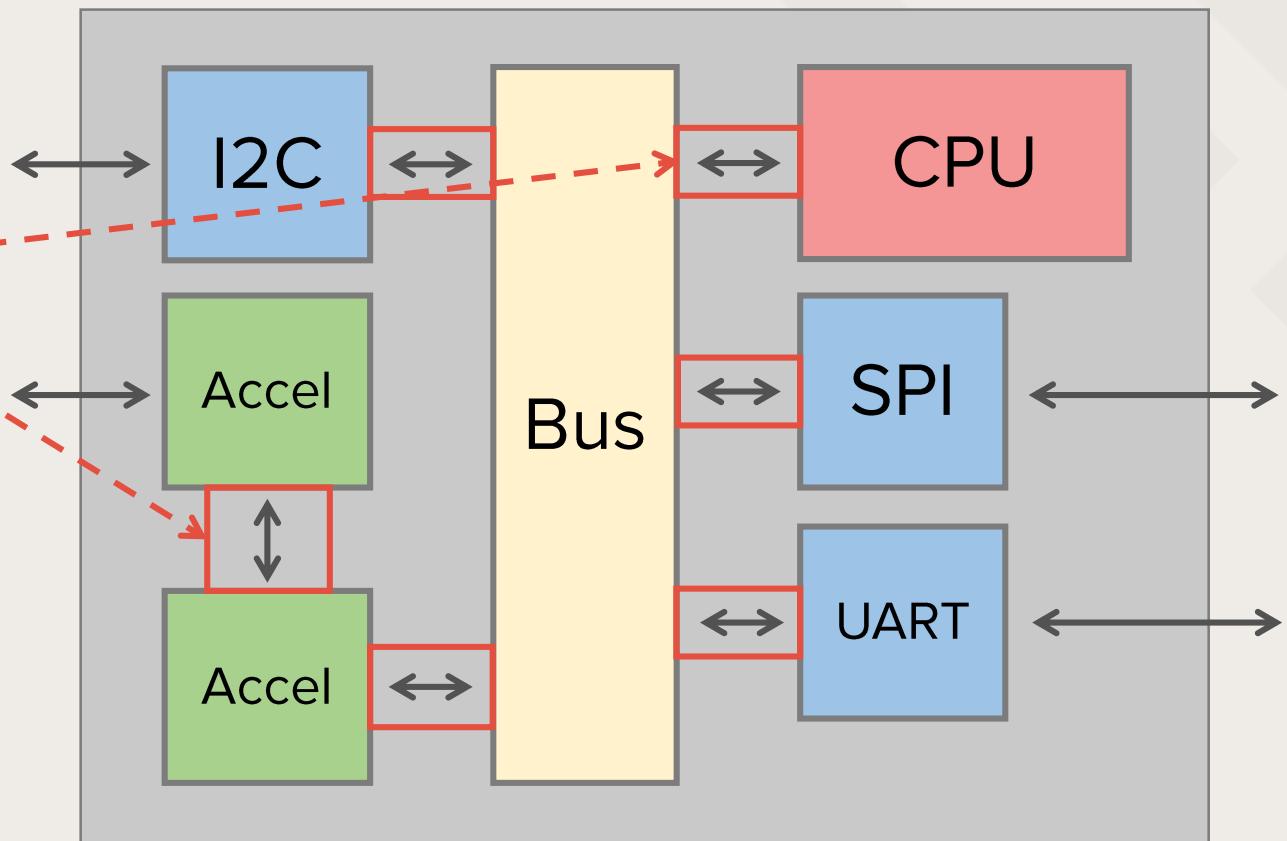


Интерфейсы: Внутренние



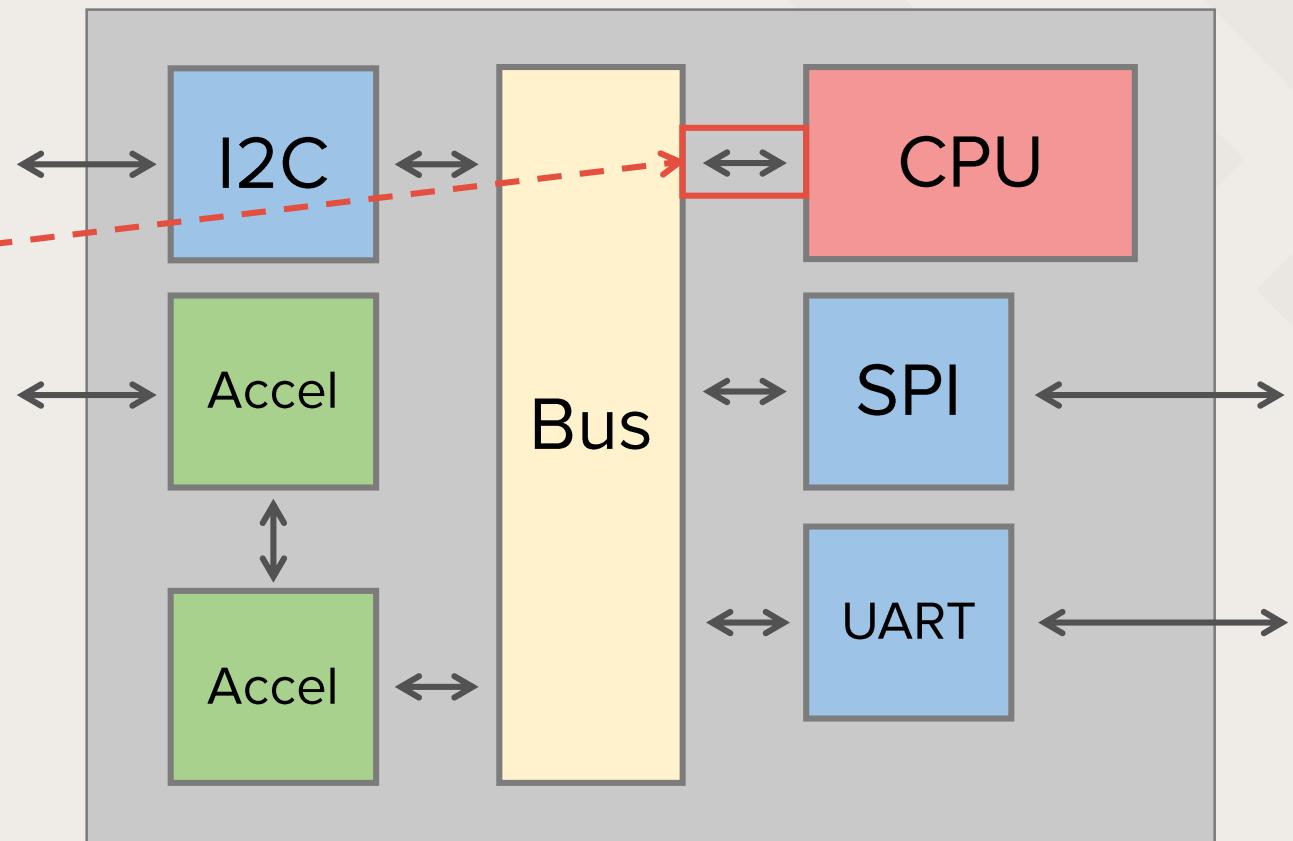
Интерфейсы: Внутренние (стандартные шины)

- AMBA APB
- AMBA AXI-Stream
- AMBA AXI4
- AMBA ...
- Wishbone
- CoreFrame
- CoreConnect
- ...



Интерфейсы: Внутренние (стандартные шины)

- AMBA APB
- AMBA AXI-Stream
- **AMBA AXI4**
- AMBA ...
- Wishbone
- CoreFrame
- CoreConnect
- ...



AMBA AXI4

Стандартные шины: AXI4

- Основные **области применения**: ?

Стандартные шины: AXI4

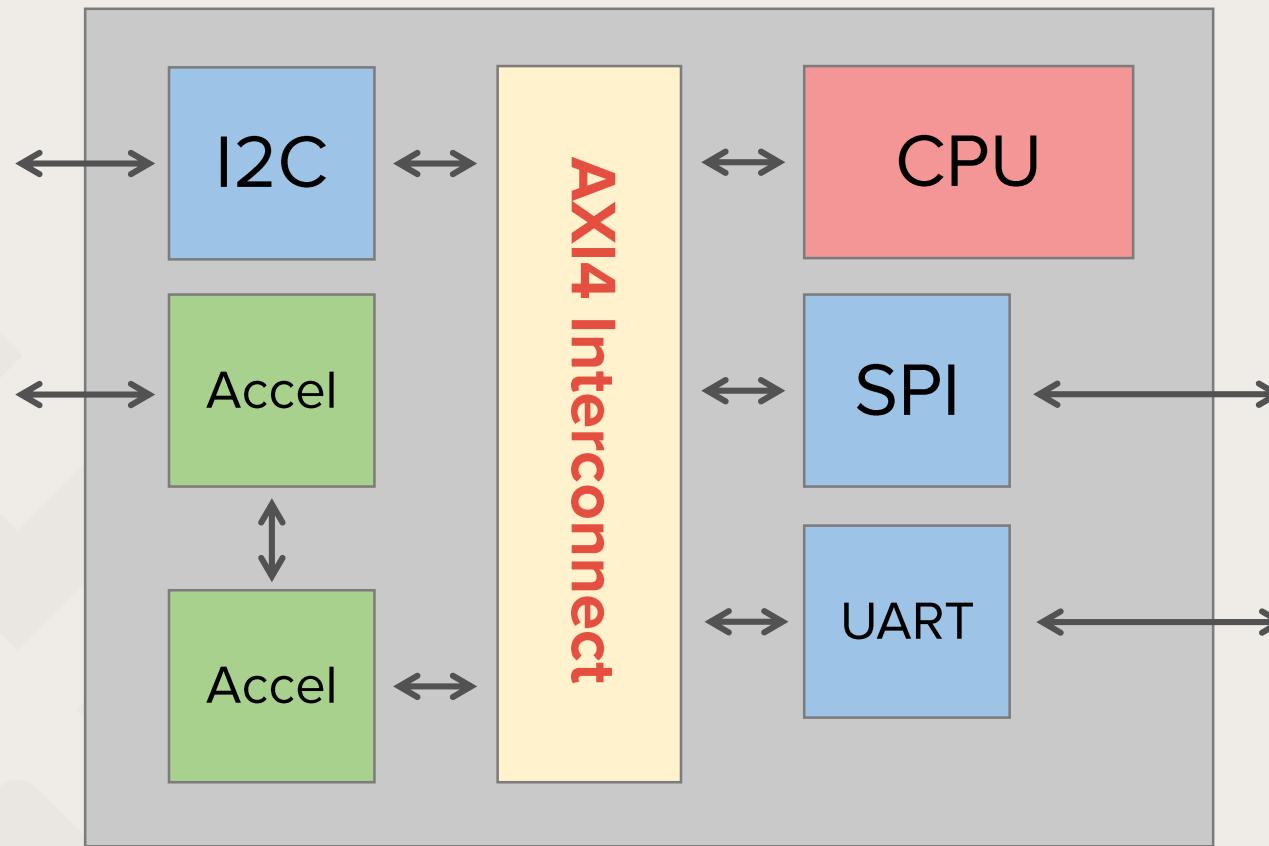
- Из **спецификации**:
- Протокол AXI:
- подходит для проектов с **высокой пропускной способностью и малой задержкой**;
- обеспечивает **высокую частоту** без использования сложных межсоединений;
- подходит для **контроллеров памяти с высокой начальной задержкой доступа**;
- обеспечивает **гибкость** при реализации архитектур межсоединений.

Стандартные шины: AXI4

- Основные **области применения**:
 - высокочастотные и высокопроизводительные дизайны;
 - контроллеры памяти;
 - процессорные ядра;
 - кэш-память;
 - связанные системы ускорителей.

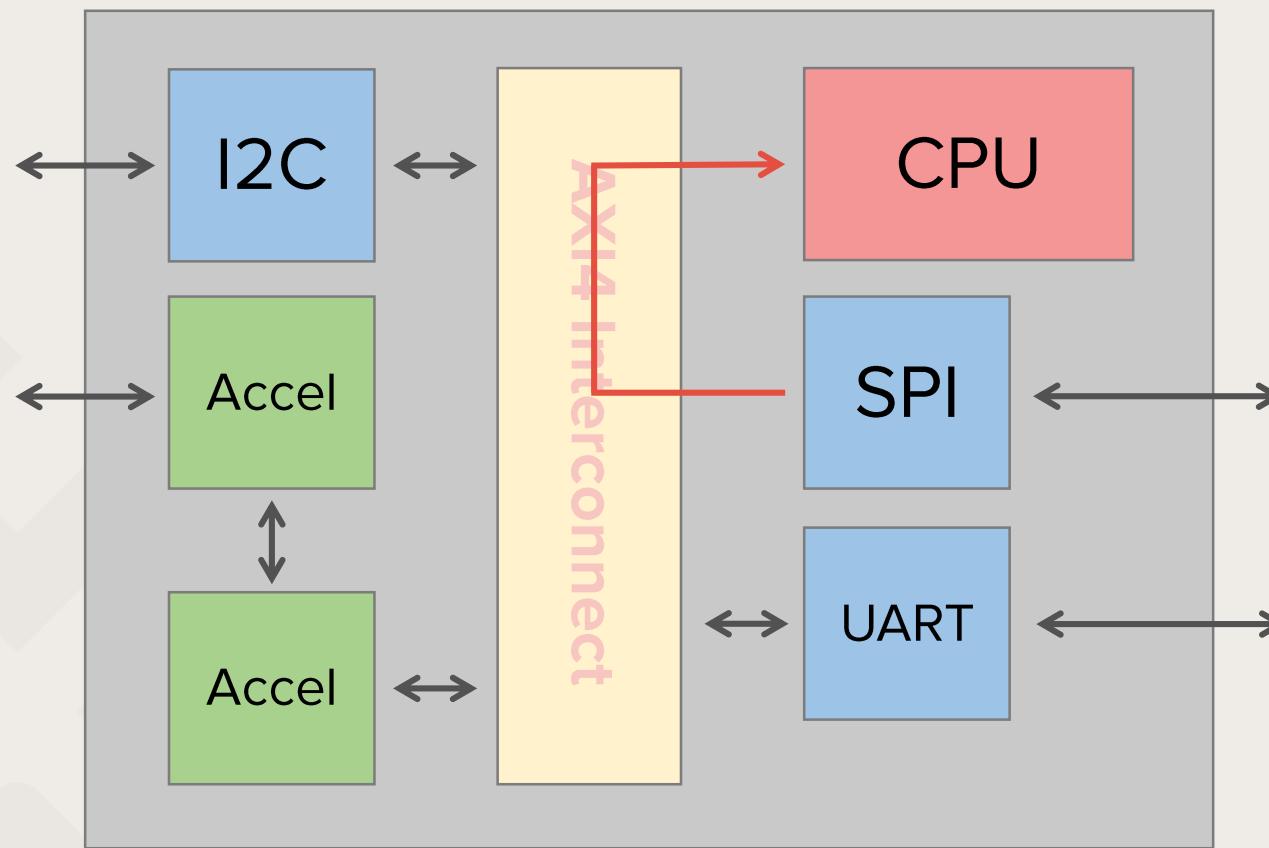
AXI4: Межсоединения

- AXI4 **зачастую применяется для связи множества элементов** с рамках СнК.



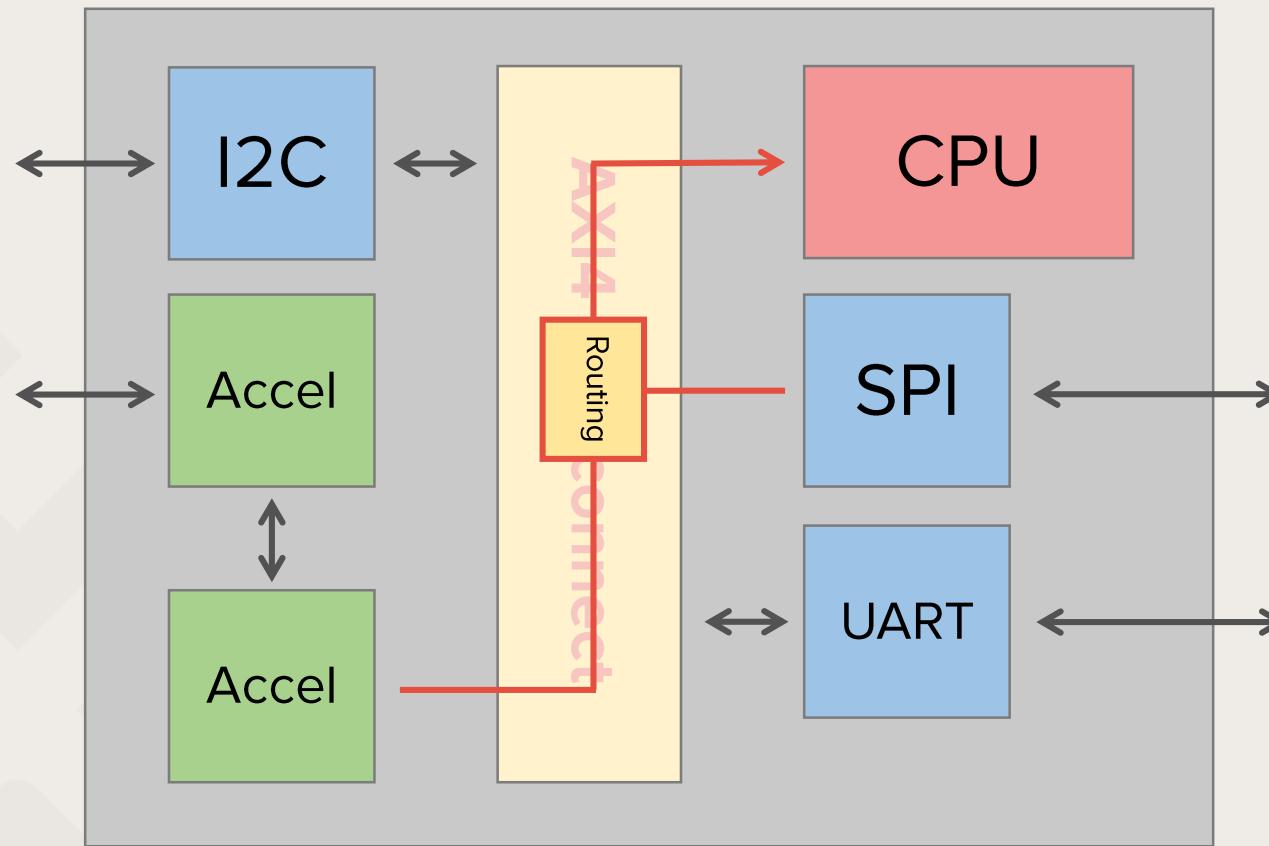
AXI4: Межсоединения

- AXI4 **зачастую применяется для связи множества элементов** с рамках СнК.

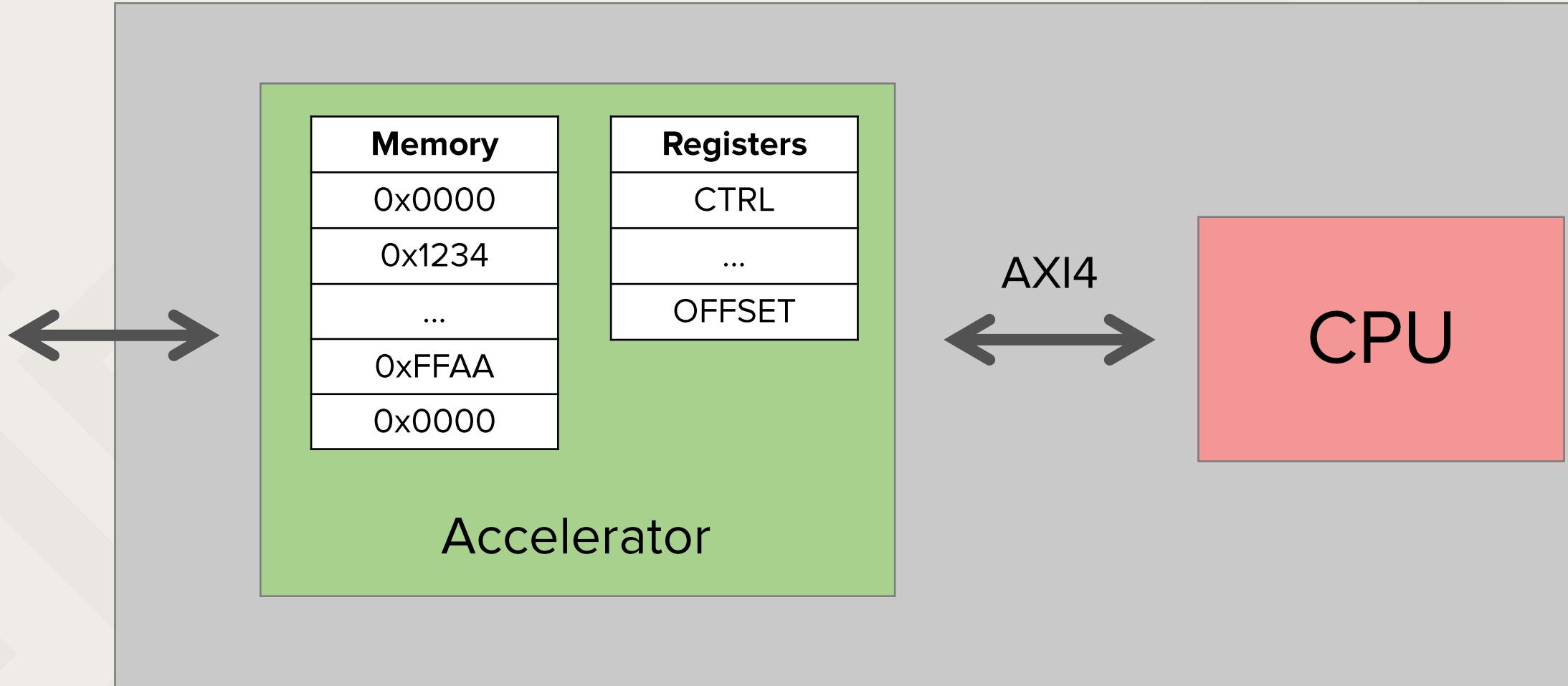


AXI4: Межсоединения

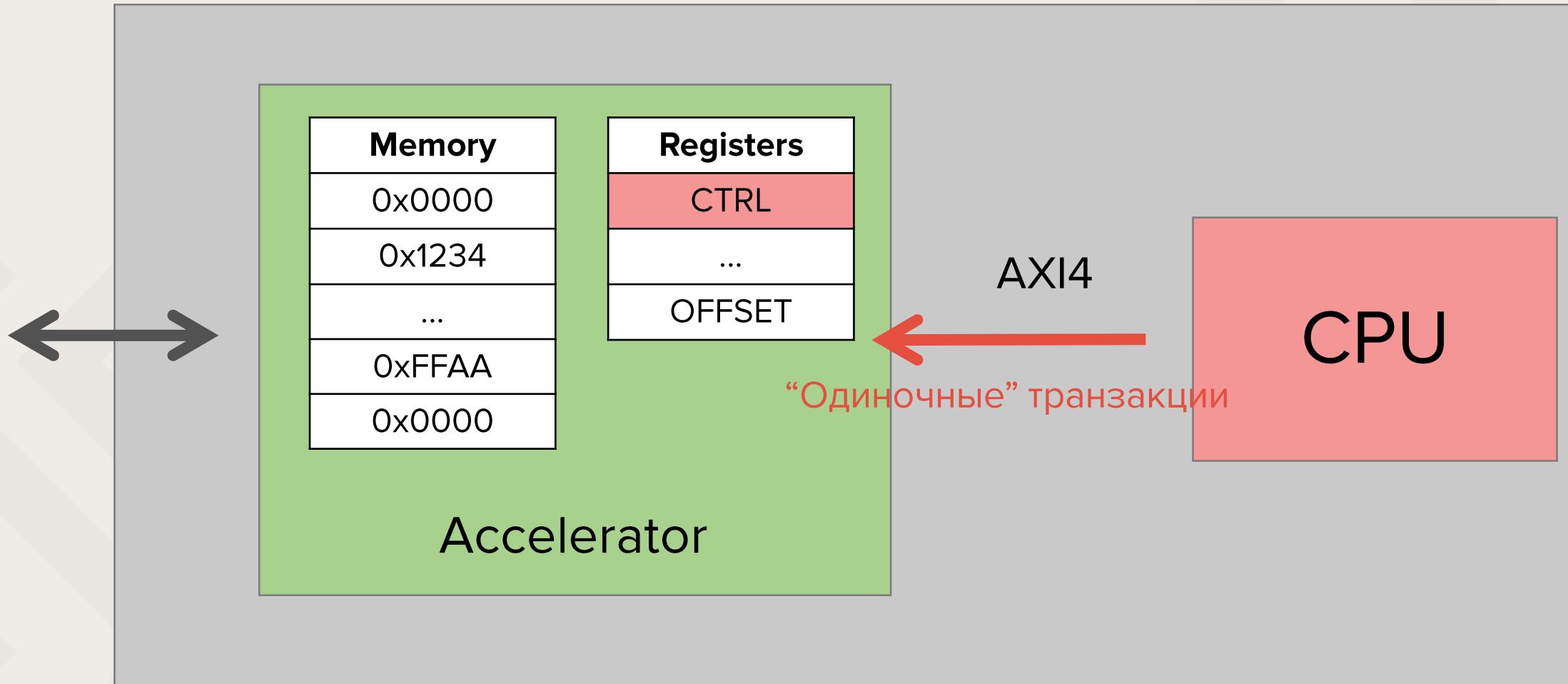
- AXI4 **зачастую применяется для связи множества элементов** с рамках СнК.



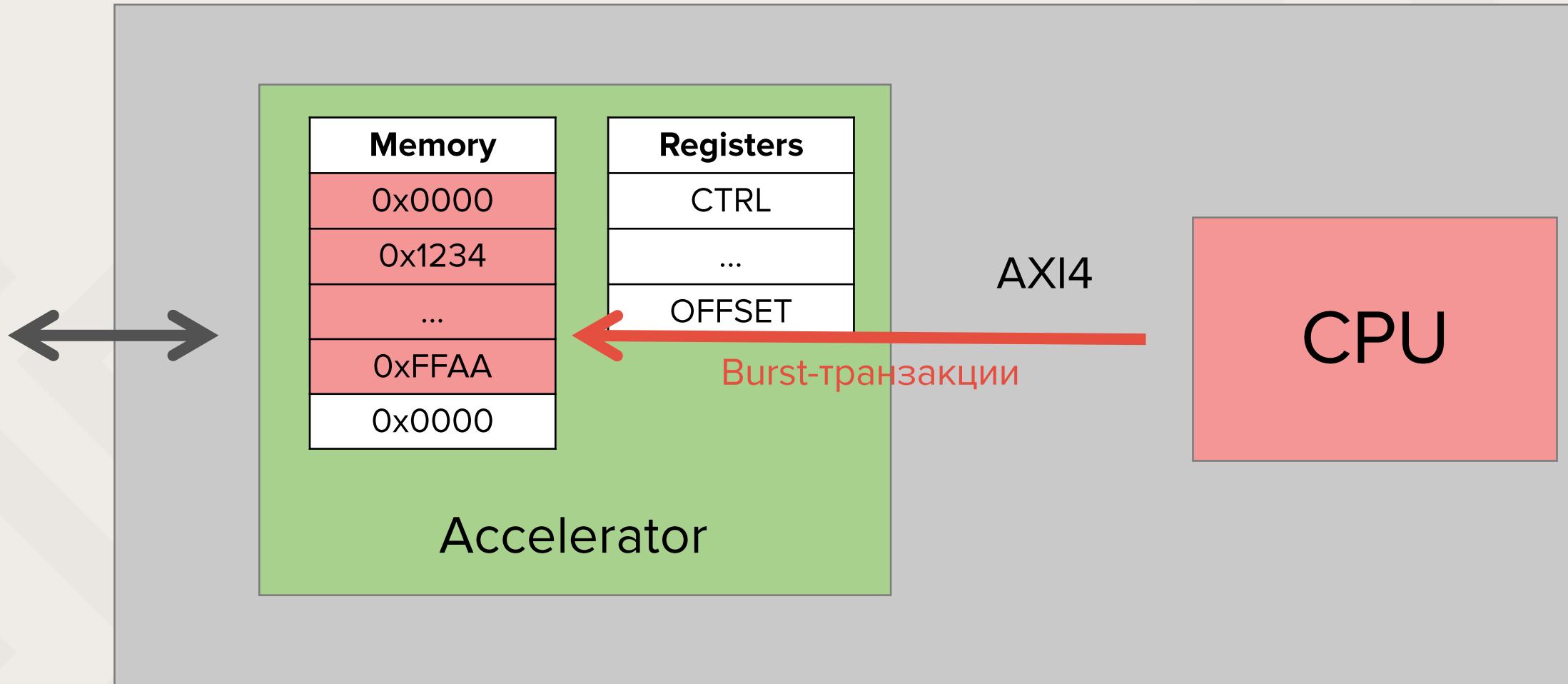
AXI4: Разделение памяти



AXI4: Разделение памяти

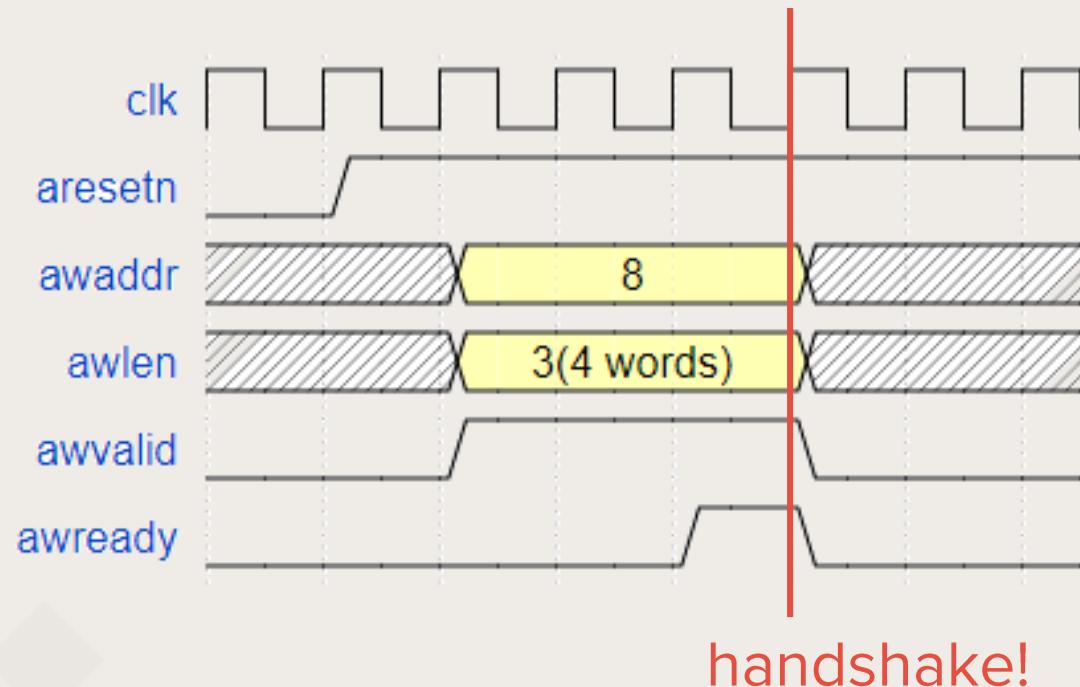


AXI4: Разделение памяти



AXI4: Handshake (рукопожатие)

- В **AXI4** передача информации происходит при помощи “рукопожатия” (**handshake**) сигналов `valid` и `ready`.

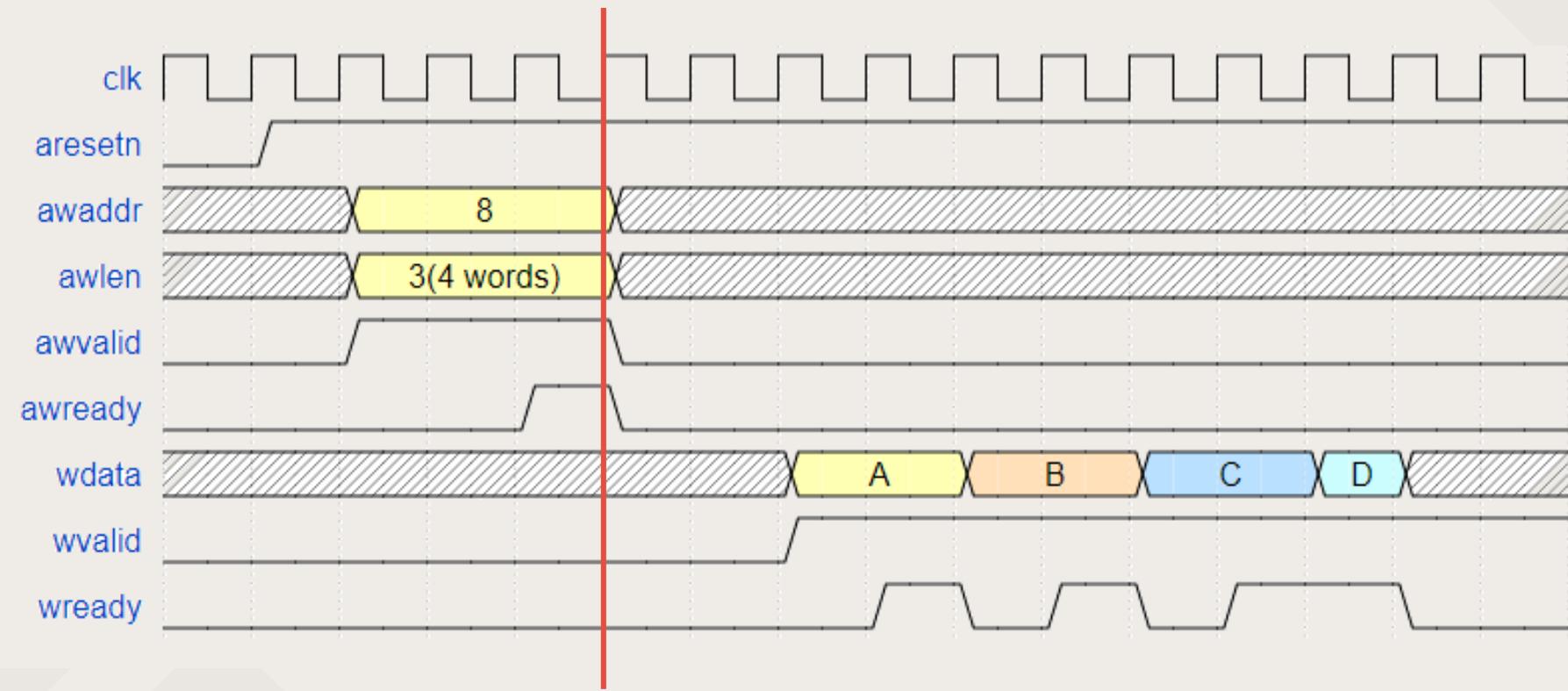


AXI4: Burst

- Протокол **AXI4** является **burst-based**.
- То есть:
 - Информация **передается пакетами**;
 - Ведущее устройство начинает каждый пакет, передавая **управляющую информацию** и адрес первой порции данных;
 - По мере получения пакета ведомое устройство должно **вычислять адреса последующих передач** в пакете.

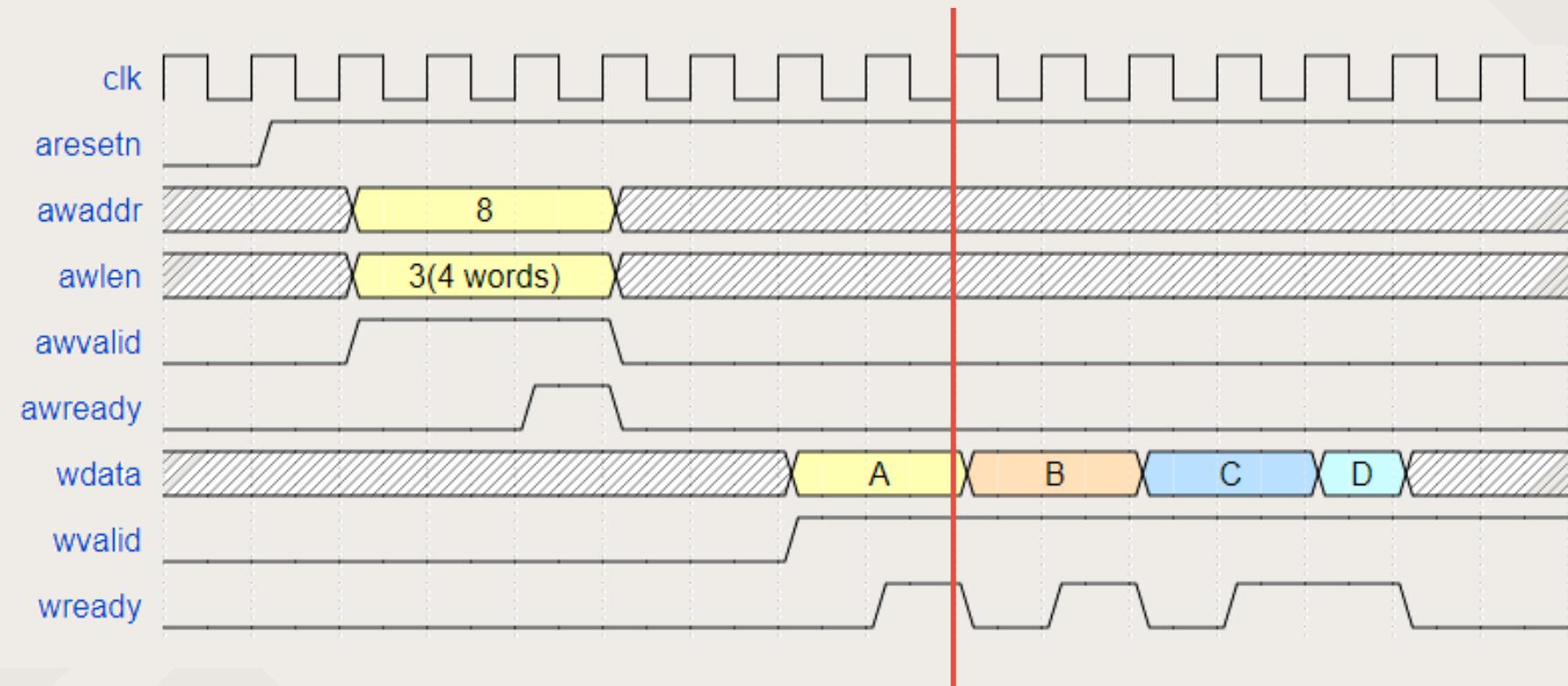
AXI4: Burst

- Передается **пакетами**;
- **Сначала – управляющая** информация;
- **Вычисление адресов** последующих передач;



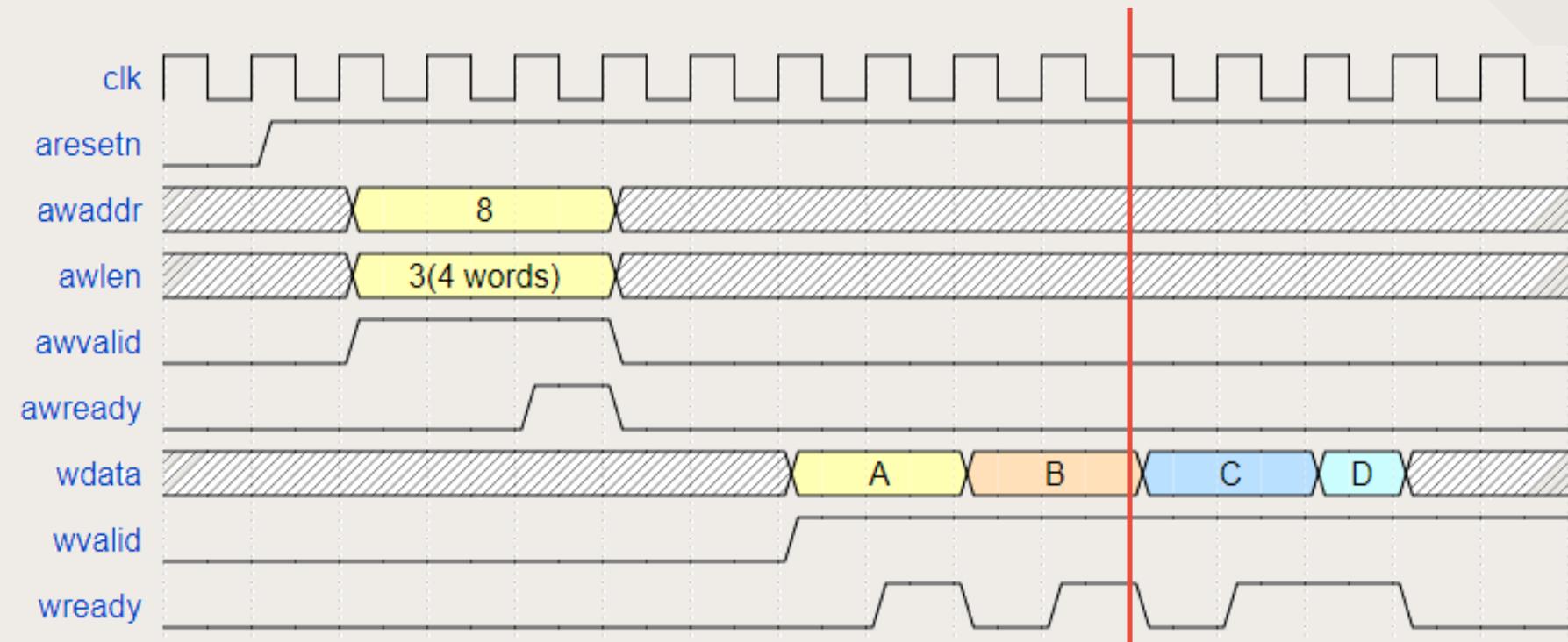
AXI4: Burst

- Передается **пакетами**;
- **Сначала – управляющая** информация;
- **Вычисление адресов** последующих передач;



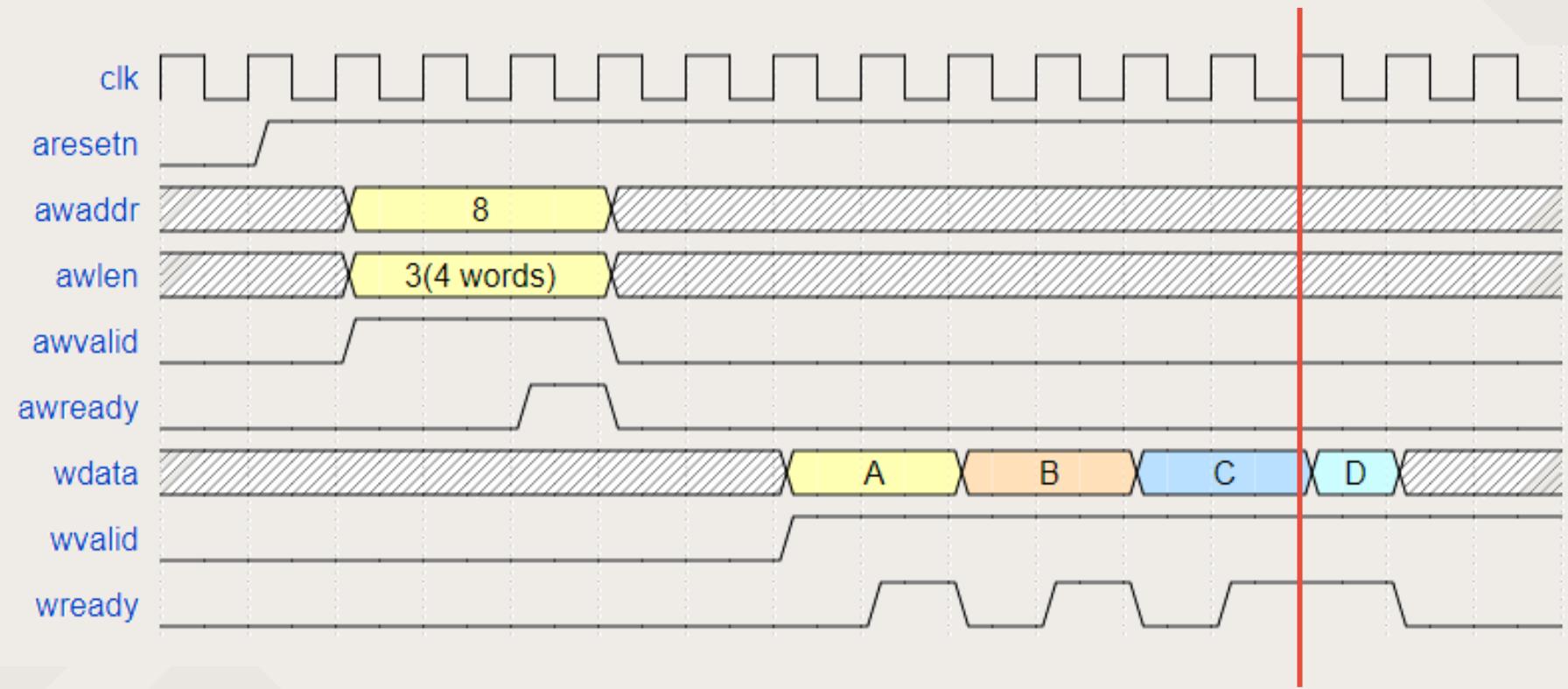
AXI4: Burst

- Передается **пакетами**;
- **Сначала – управляющая** информация;
- **Вычисление адресов** последующих передач;



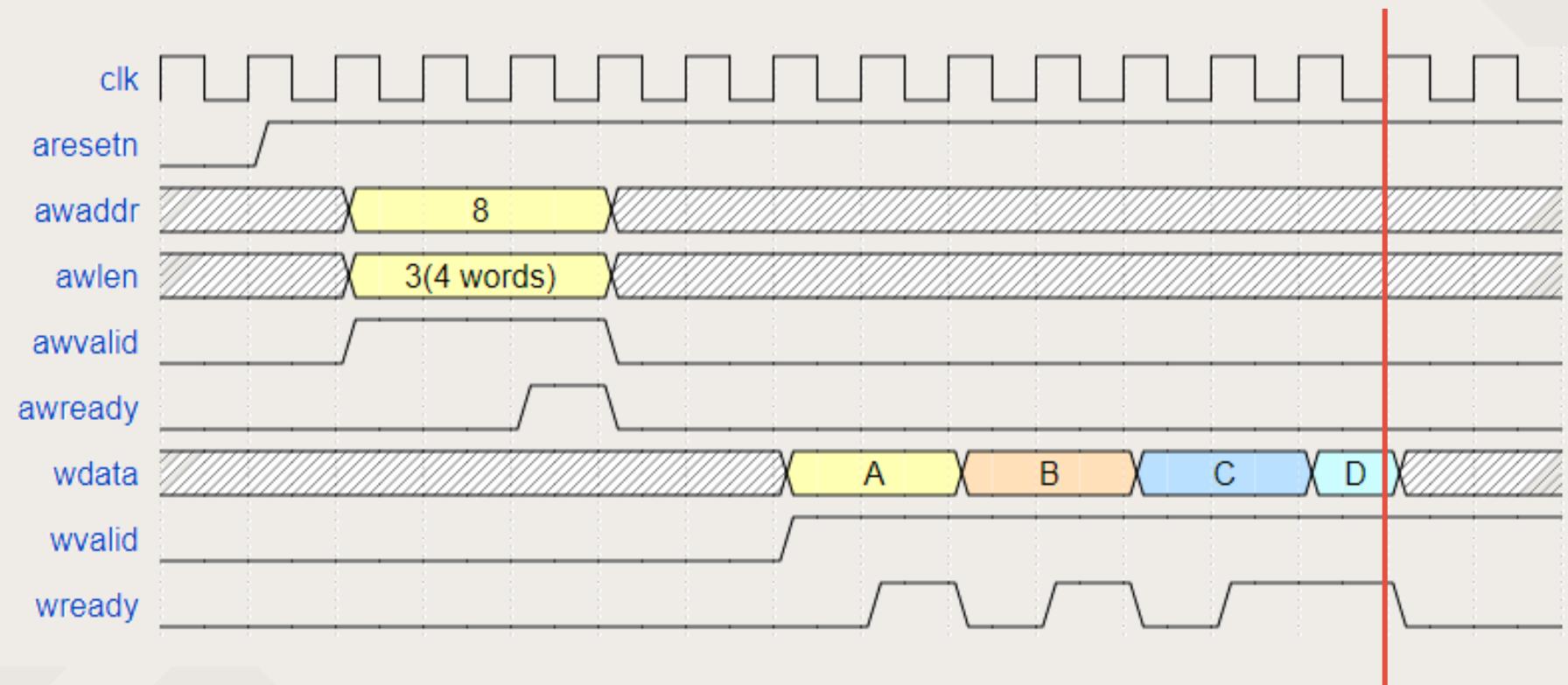
AXI4: Burst

- Передается **пакетами**;
- Сначала – управляющая** информация;
- Вычисление адресов** последующих передач;



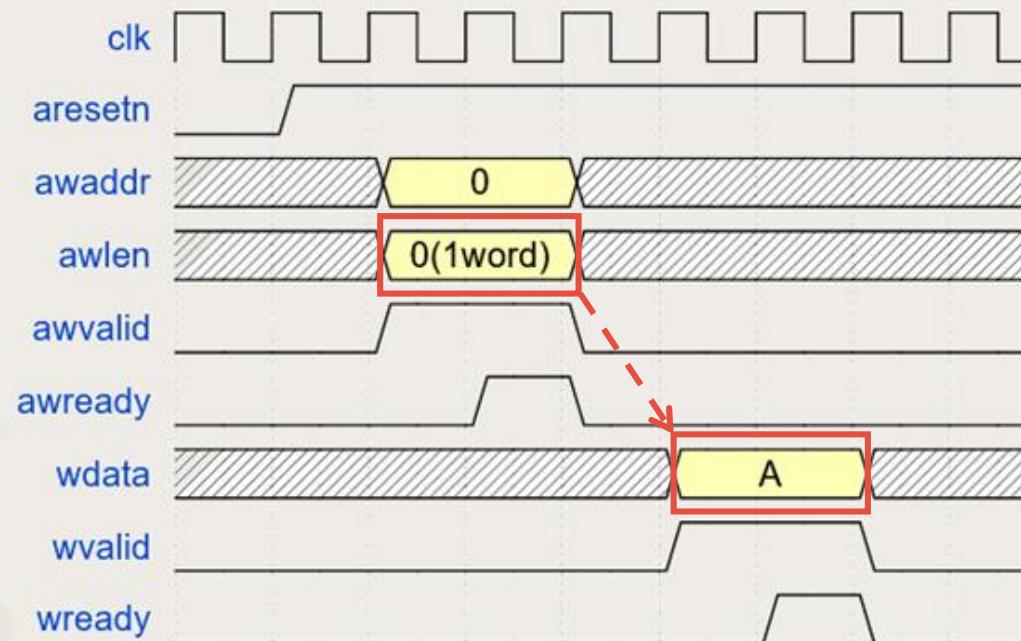
AXI4: Burst

- Передается **пакетами**;
- **Сначала – управляющая** информация;
- **Вычисление адресов** последующих передач;



AXI4: Burst

- В данной лекции все **транзакции будут одиночными**.
- То есть **размер burst'a будет равен 1**.



АХ4: 5 независимых каналов

- Канал адреса записи
- Канал данных записи
- Канал ответа на запись

3 канала для записи

+

- Канал адреса чтения
- Канал данных чтения

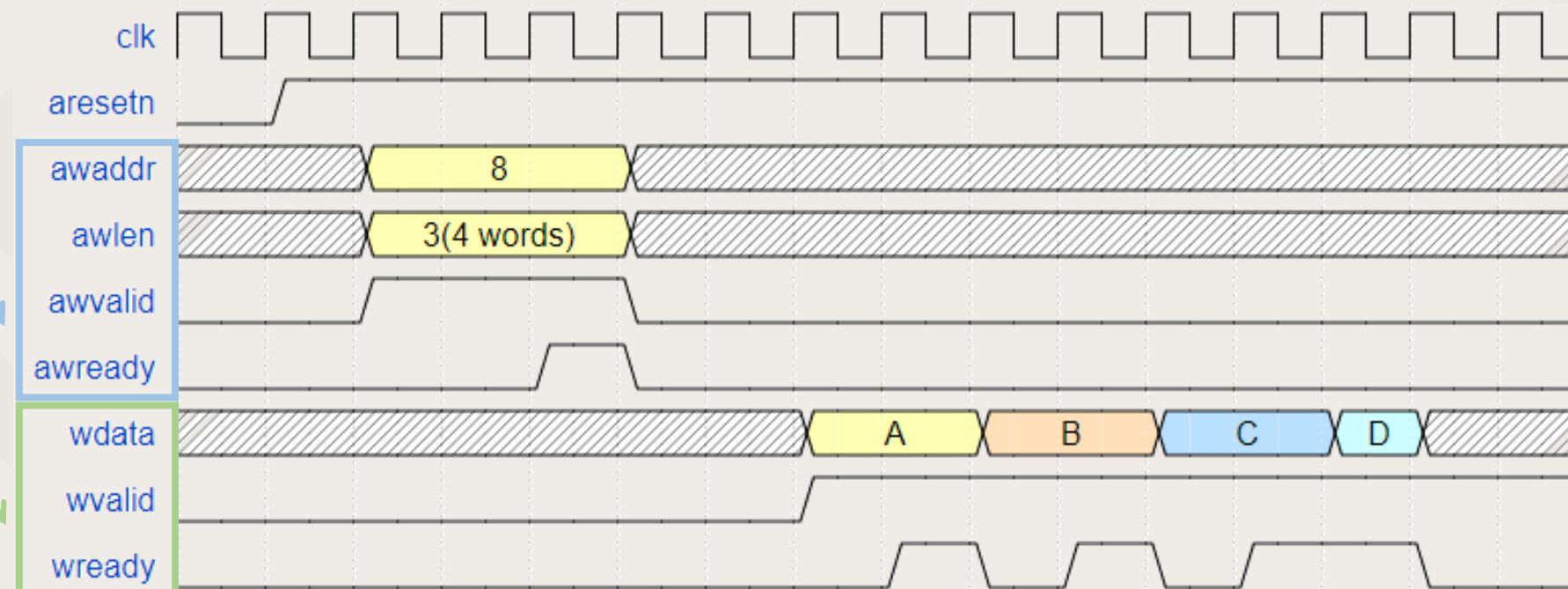
2 канала для чтения

- Каналы **являются независимыми.**
- В **данной лекции** будут рассмотрены **минимальные наборы сигналов** каждого канала.

AXI4: 5 независимых каналов

- Канал адреса записи
- Канал данных записи

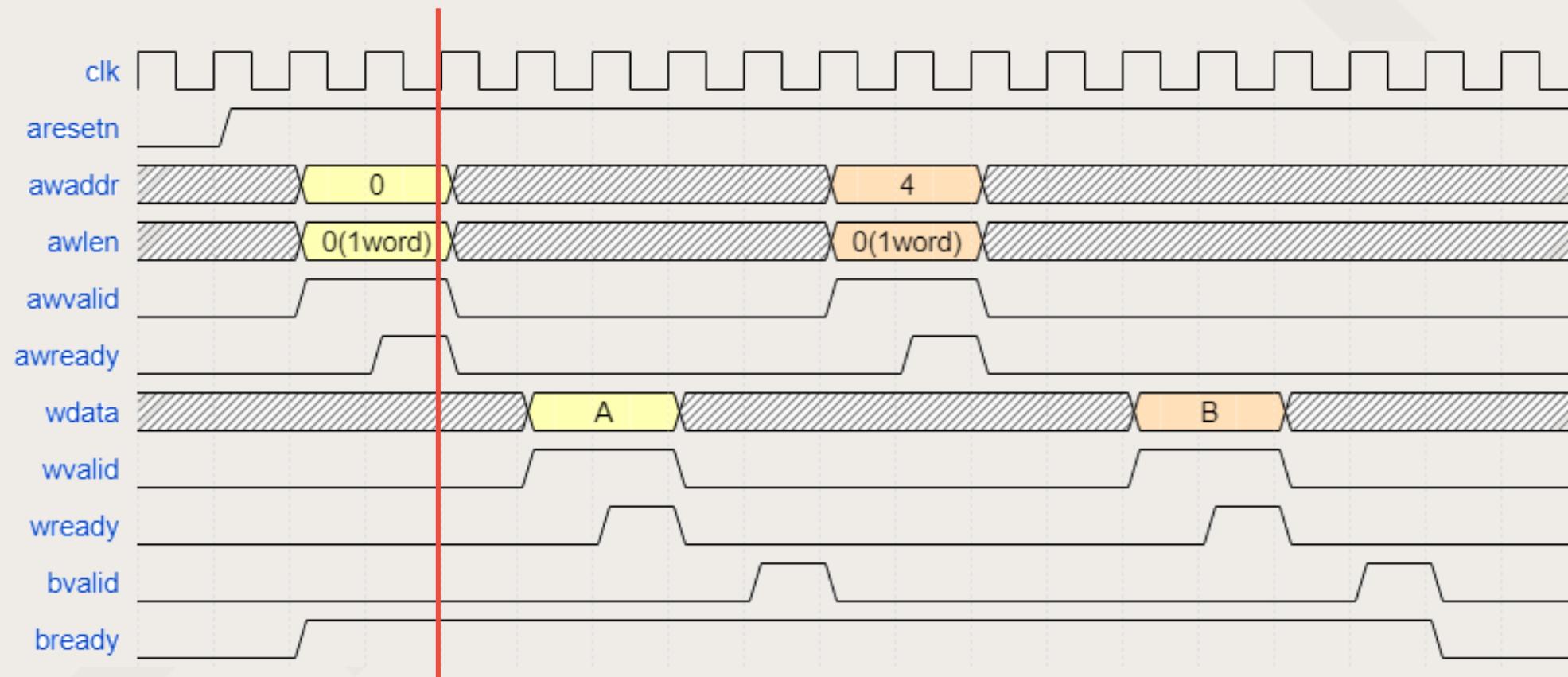
Пример 2 каналов записи



AXI4: Одиночные транзакции: запись

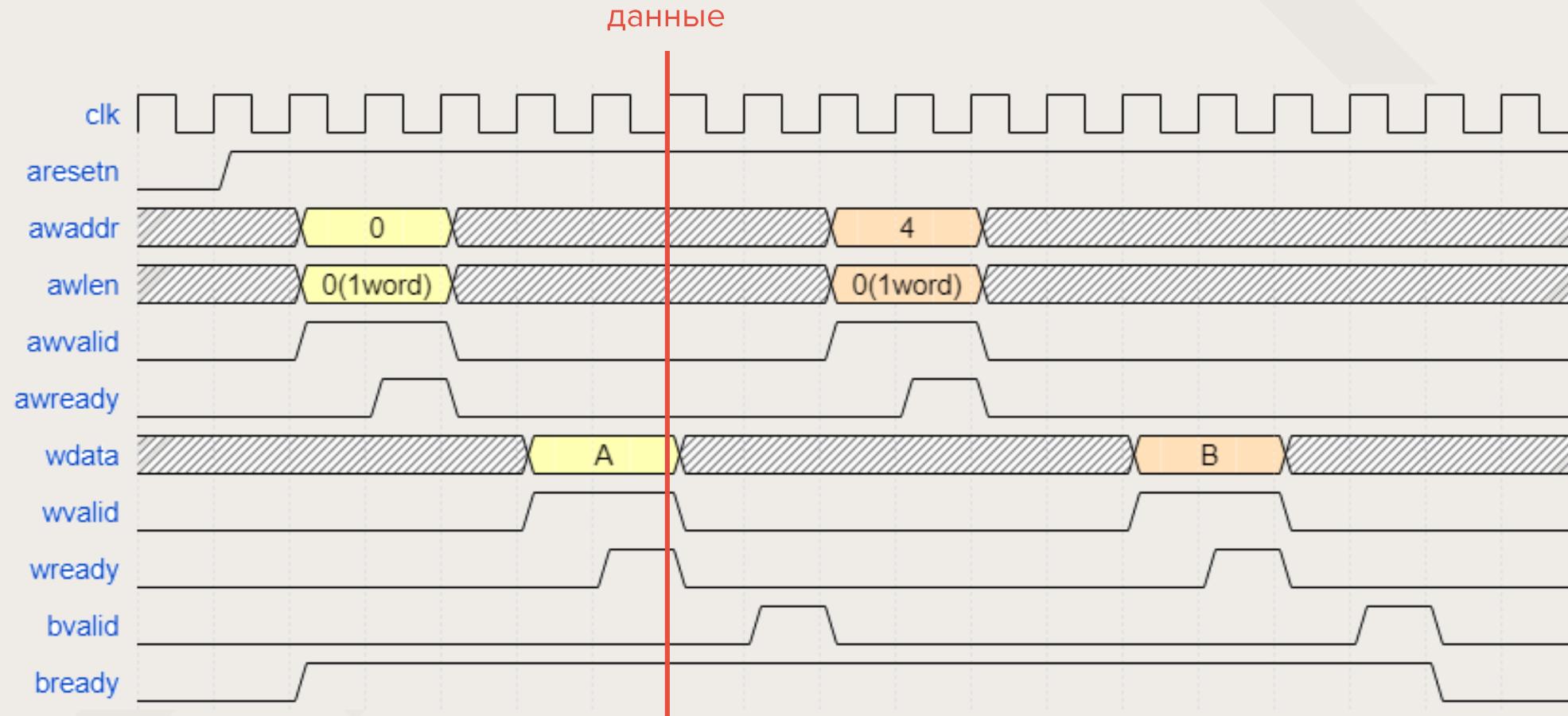
- Транзакция 1

управляющая информация



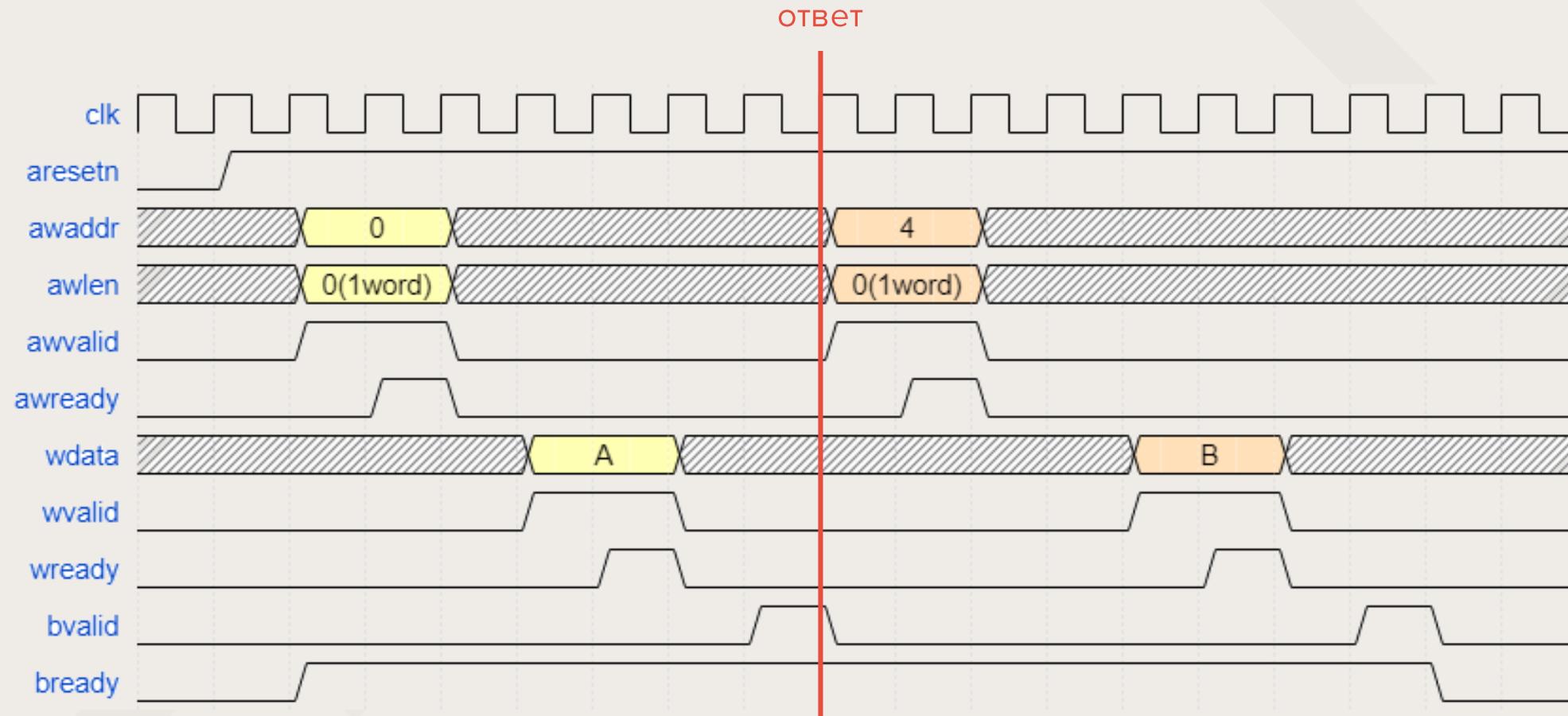
AXI4: Одиночные транзакции: запись

- Транзакция 1



AXI4: Одиночные транзакции: запись

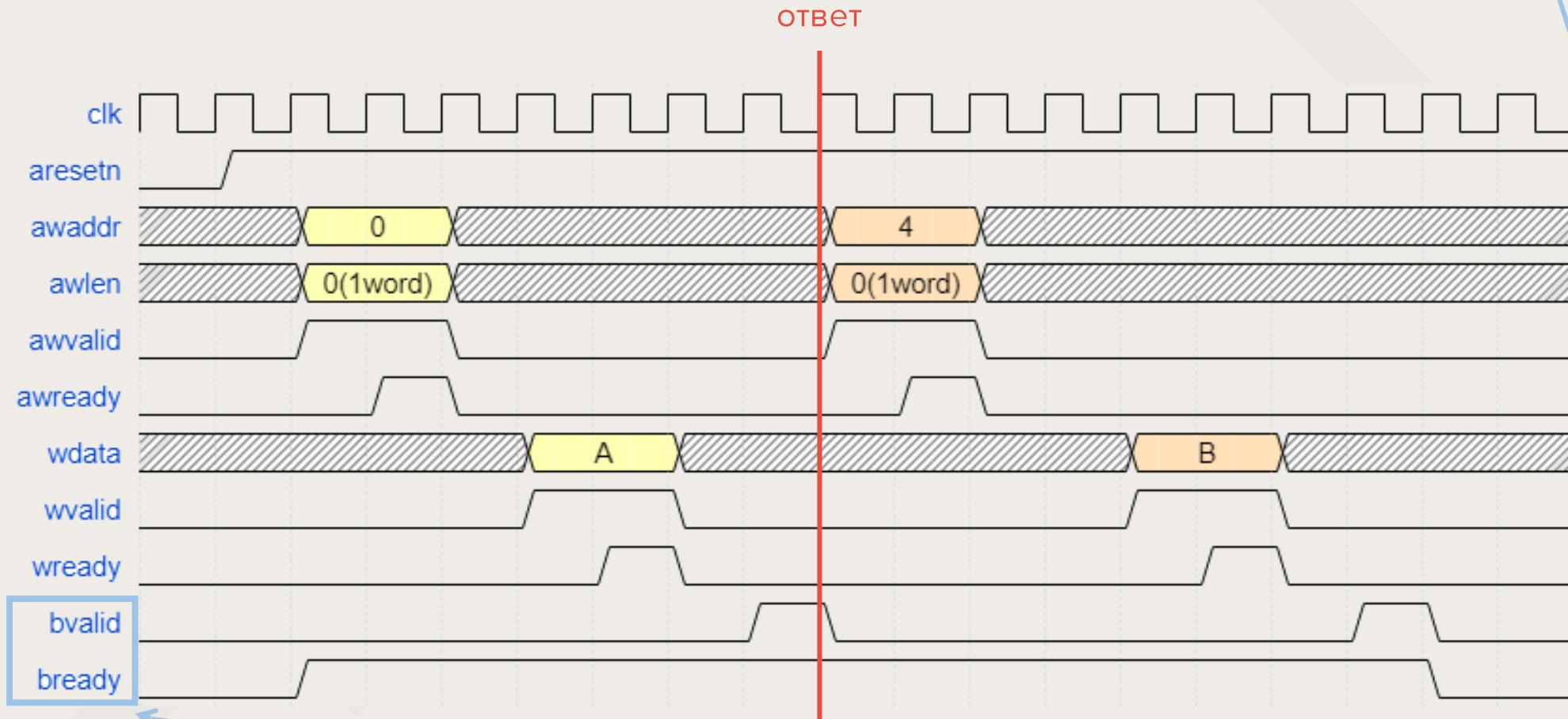
- Транзакция 1



AXI4: Одиночные транзакции: запись

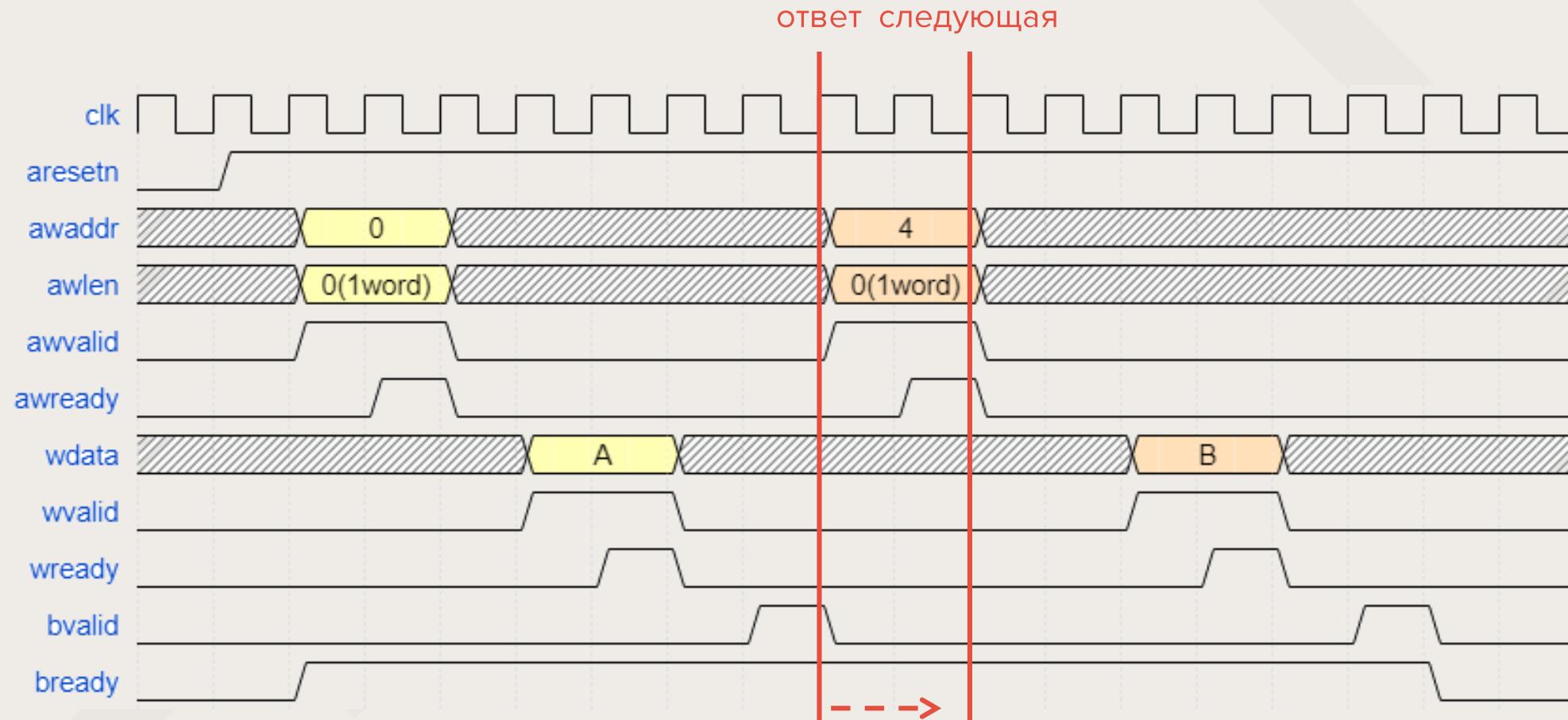
- Транзакция 1

Канал ответа на запись



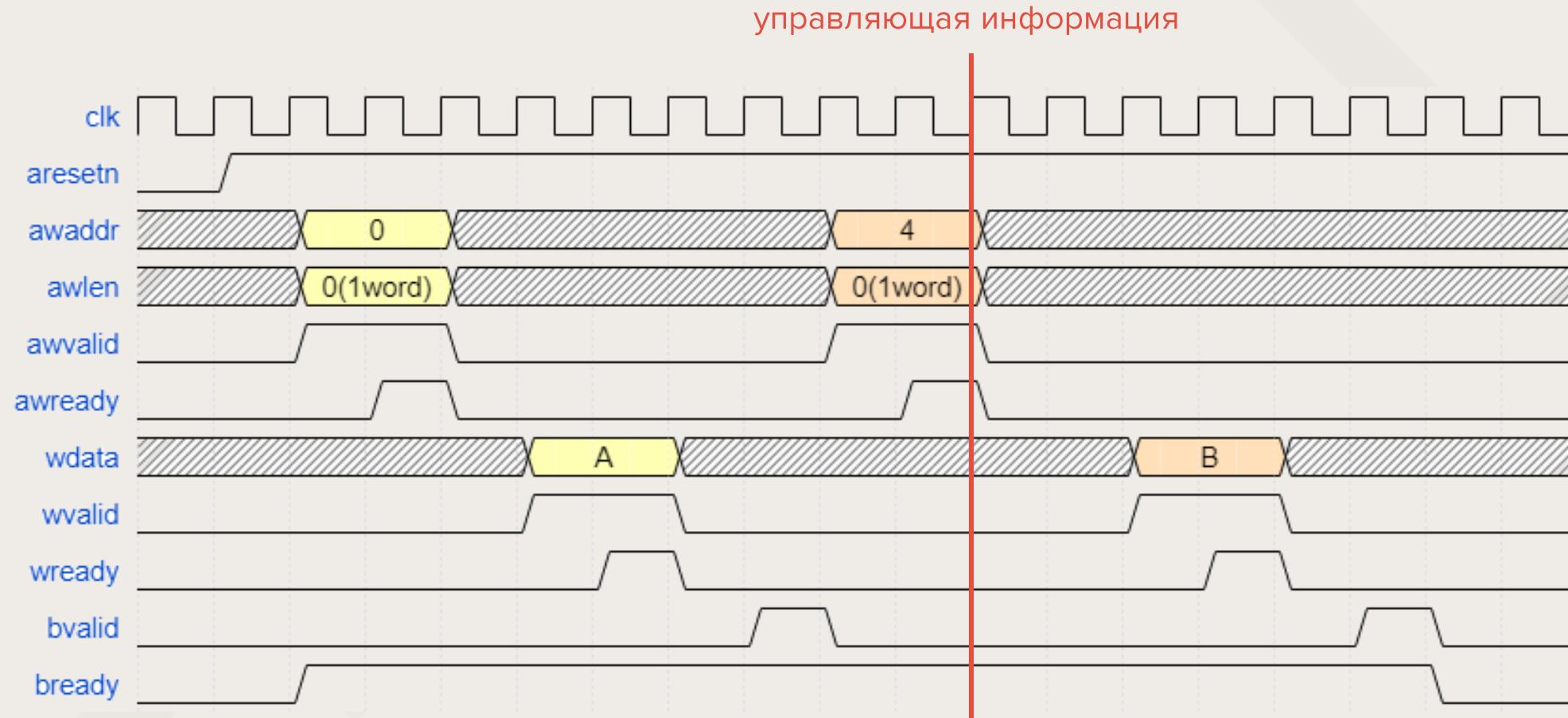
AXI4: Одиночные транзакции: запись

- Транзакция 1



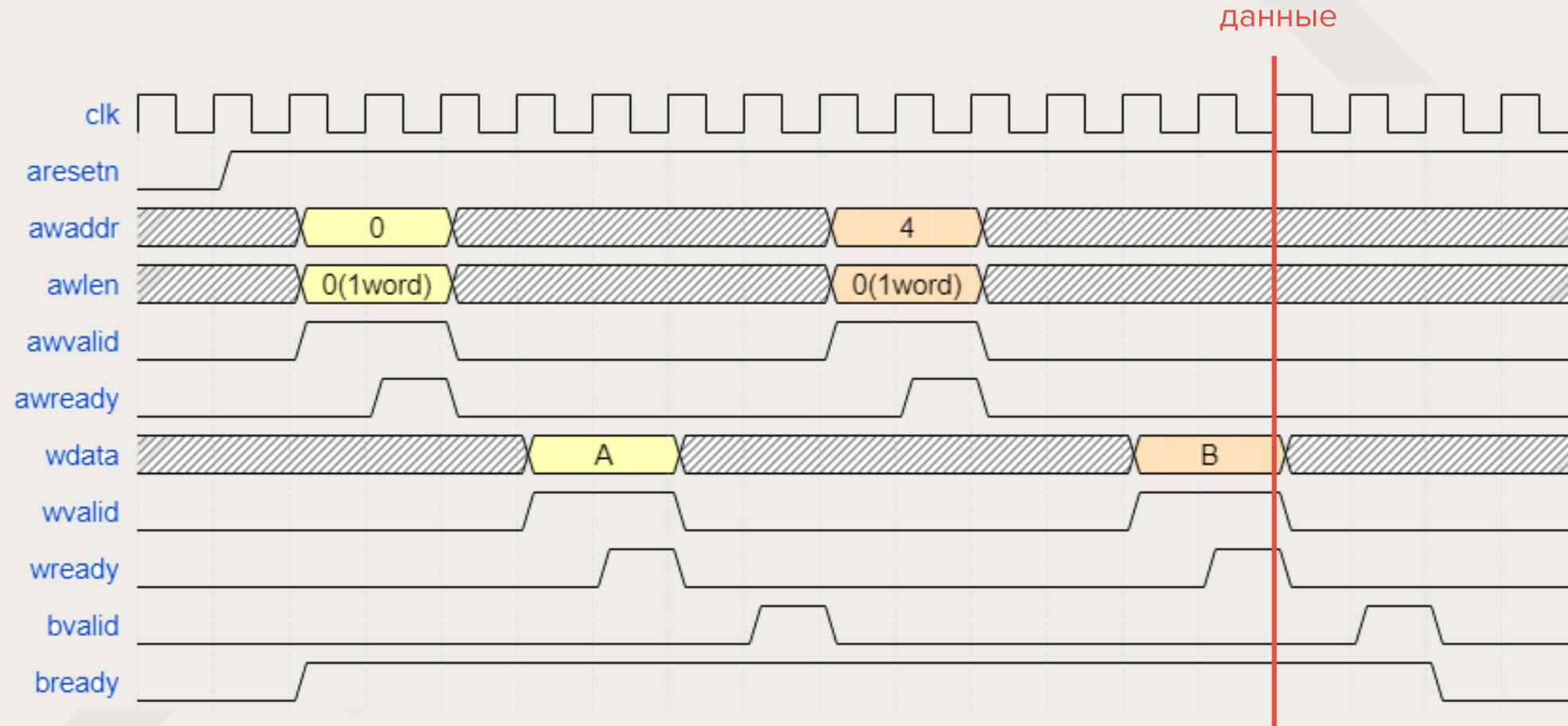
AXI4: Одиночные транзакции: запись

- Транзакция 2



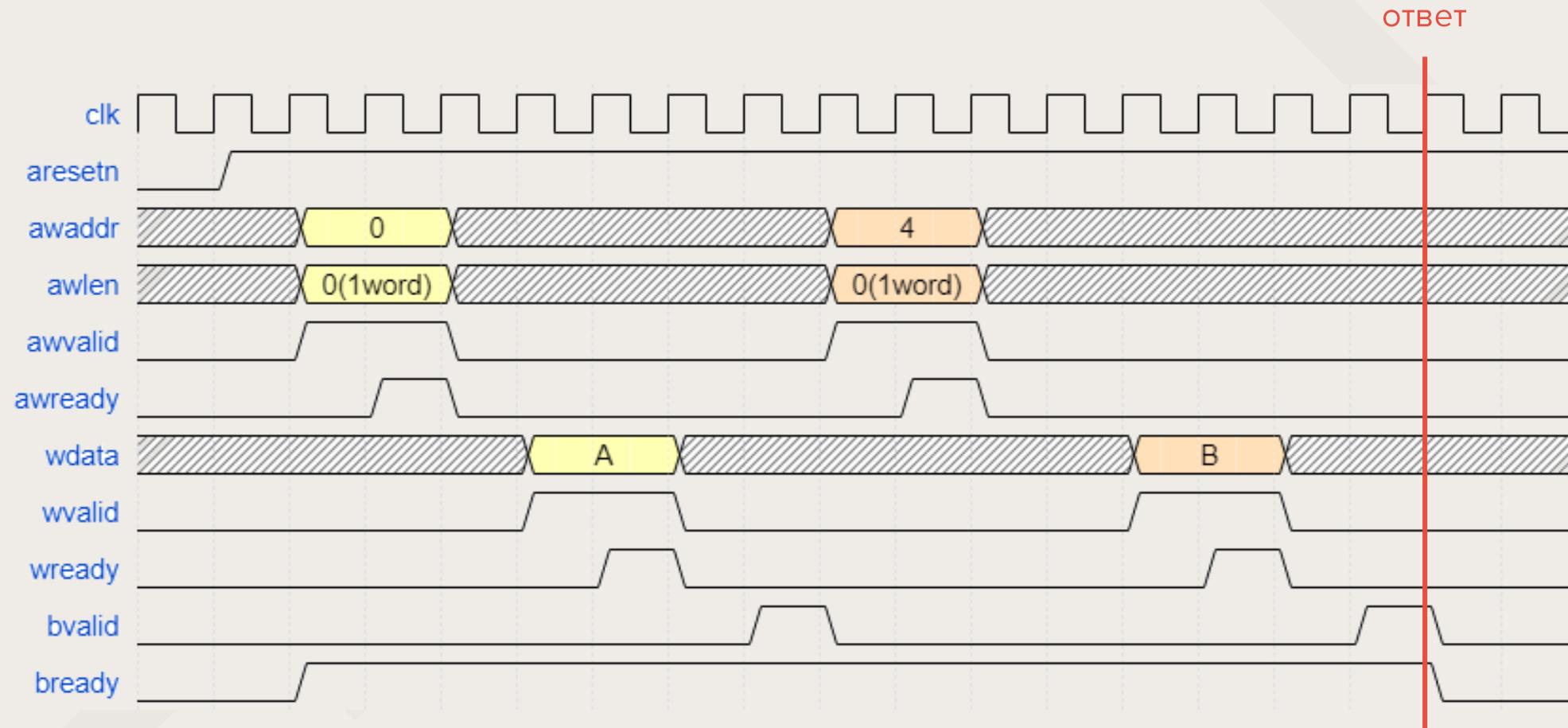
AXI4: Одиночные транзакции: запись

- Транзакция 2

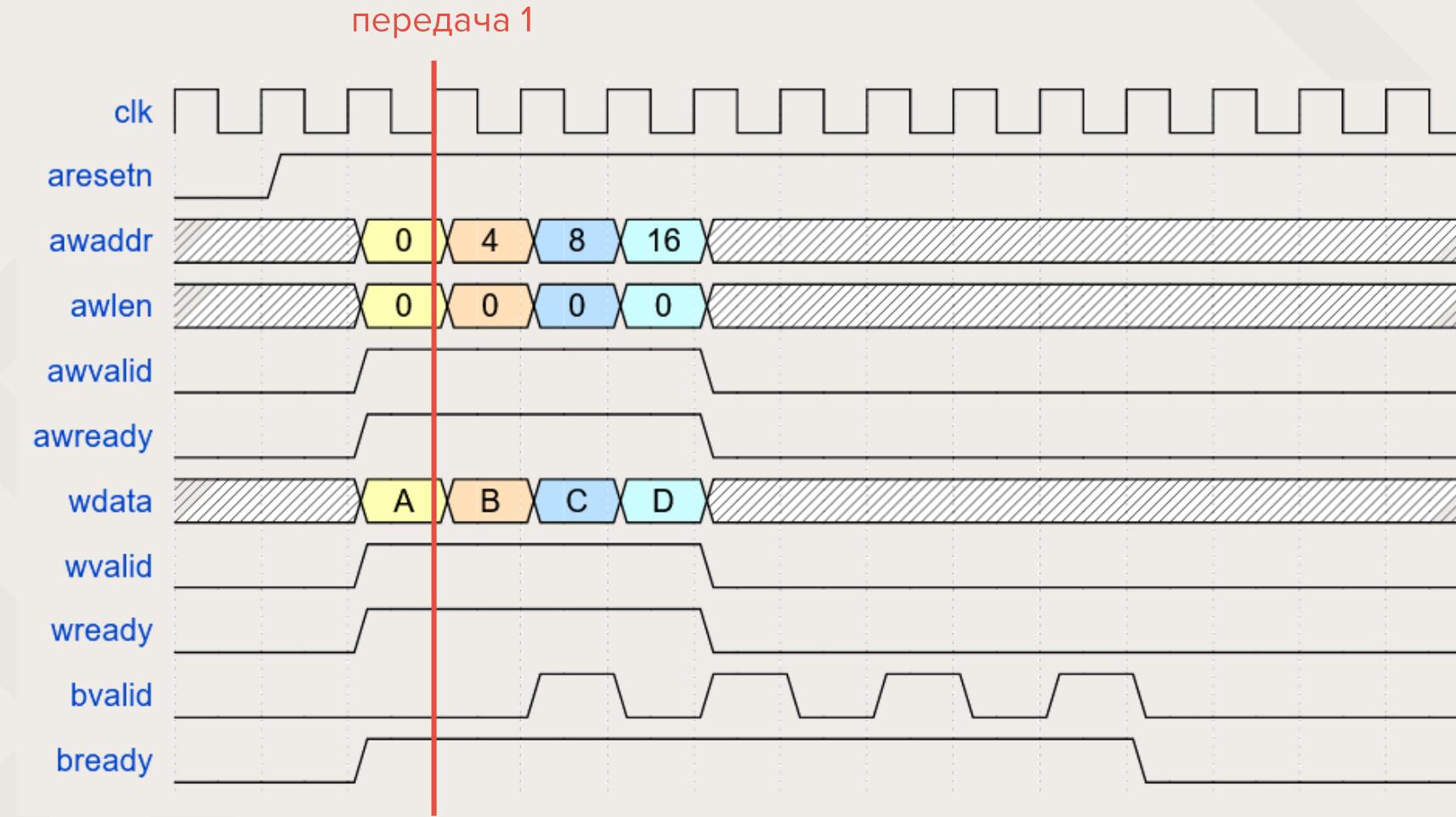


AXI4: Одиночные транзакции: запись

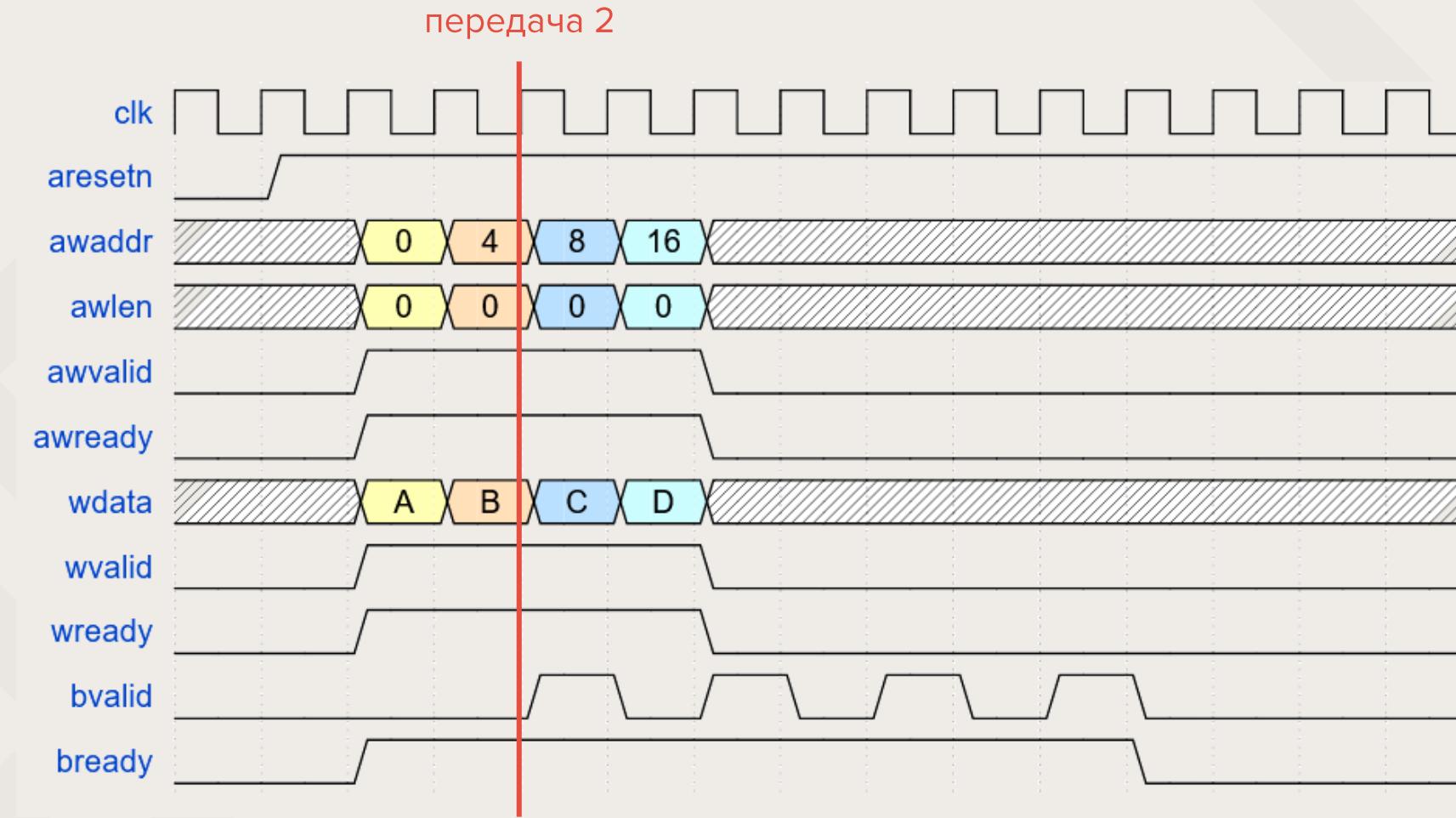
- Транзакция 2



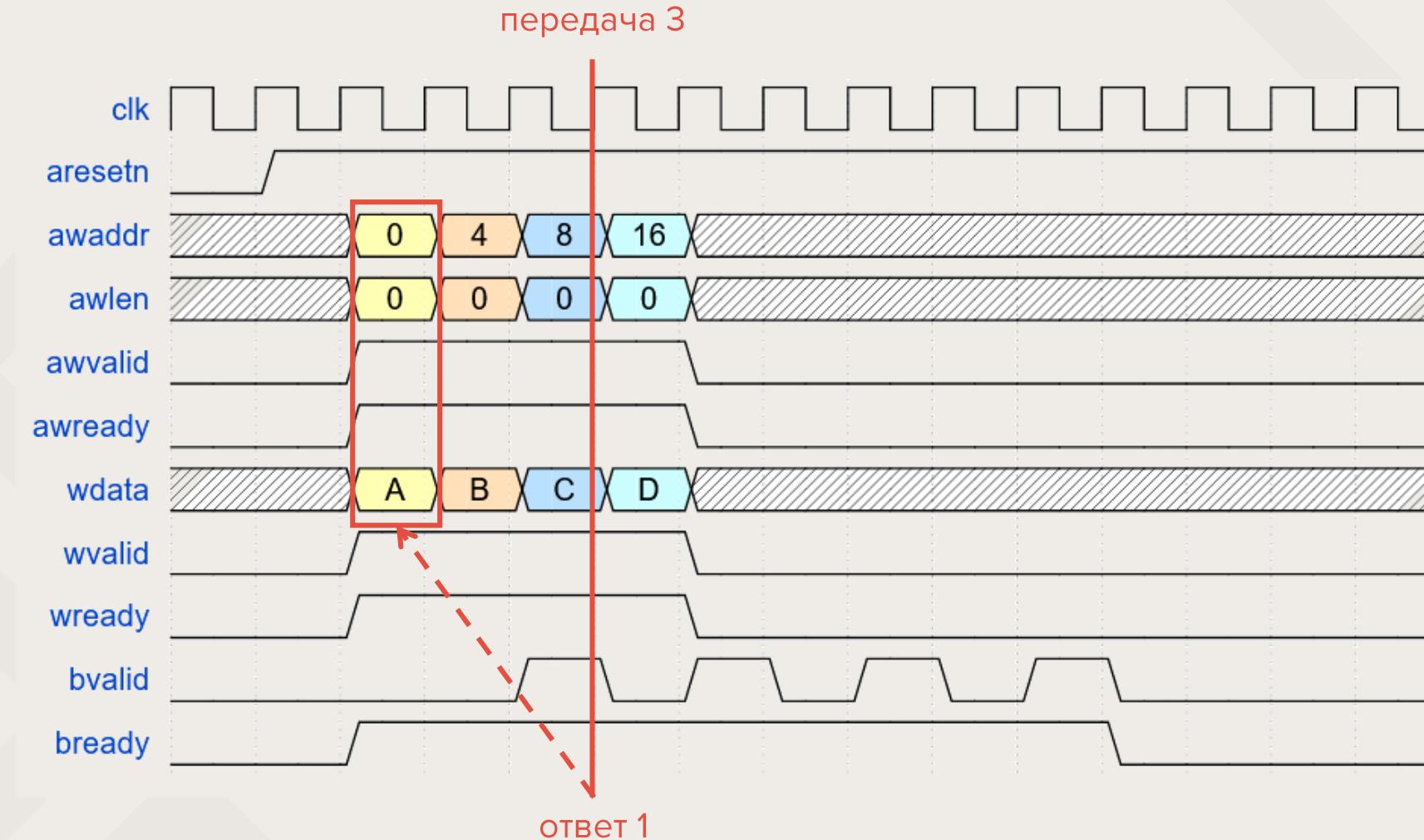
AXI4: Конвейерные транзакции: запись



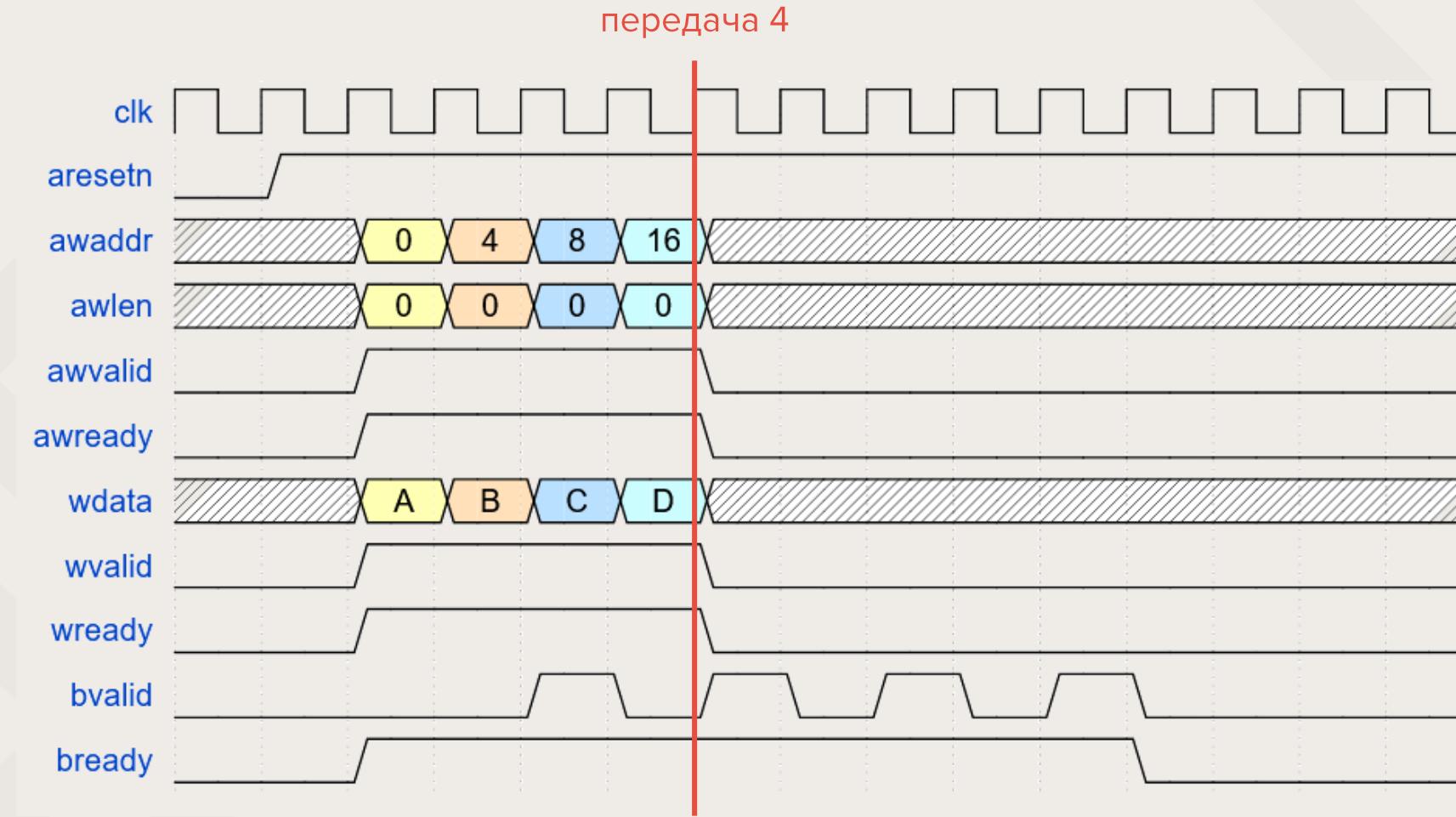
AXI4: Конвейерные транзакции: запись



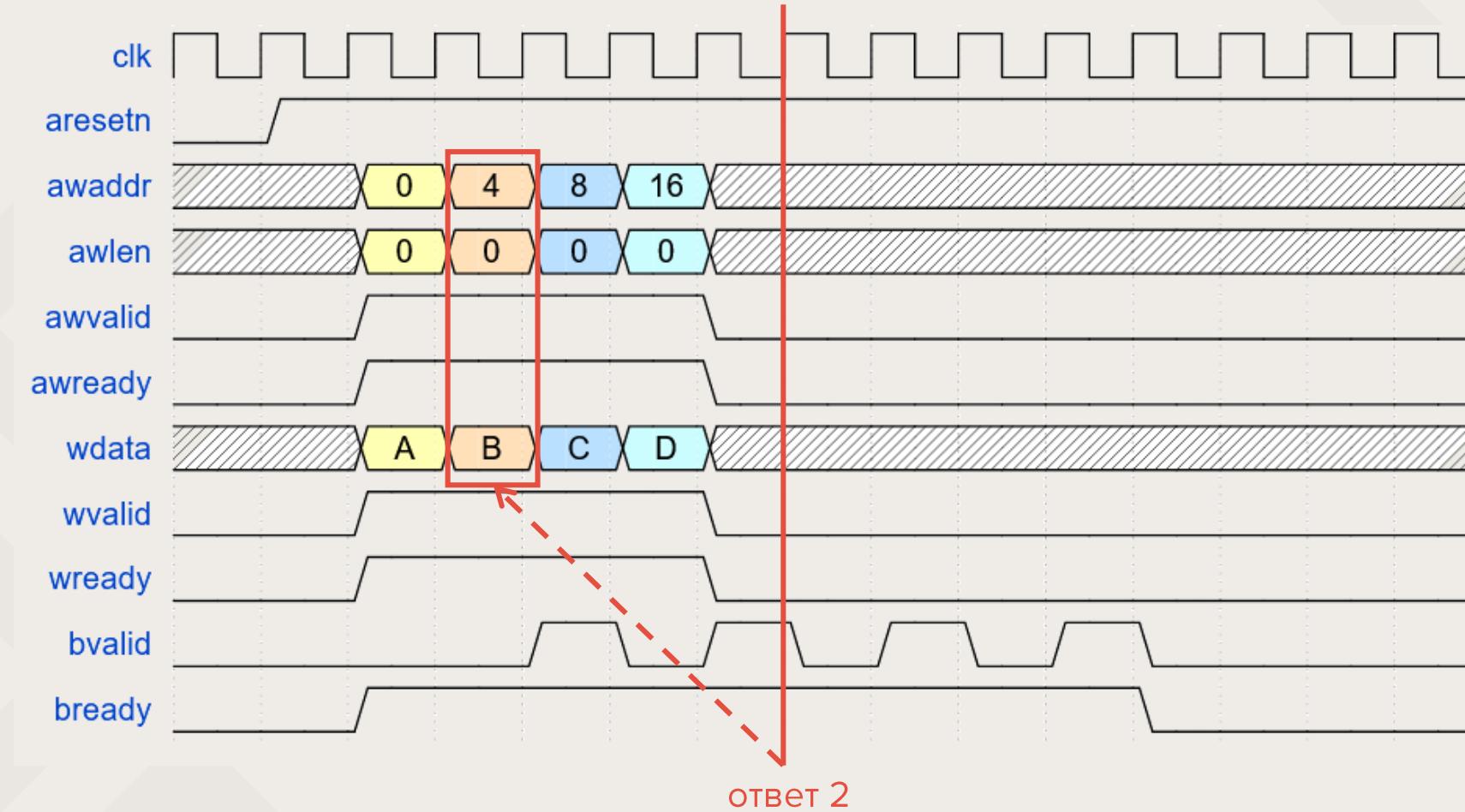
AXI4: Конвейерные транзакции: запись



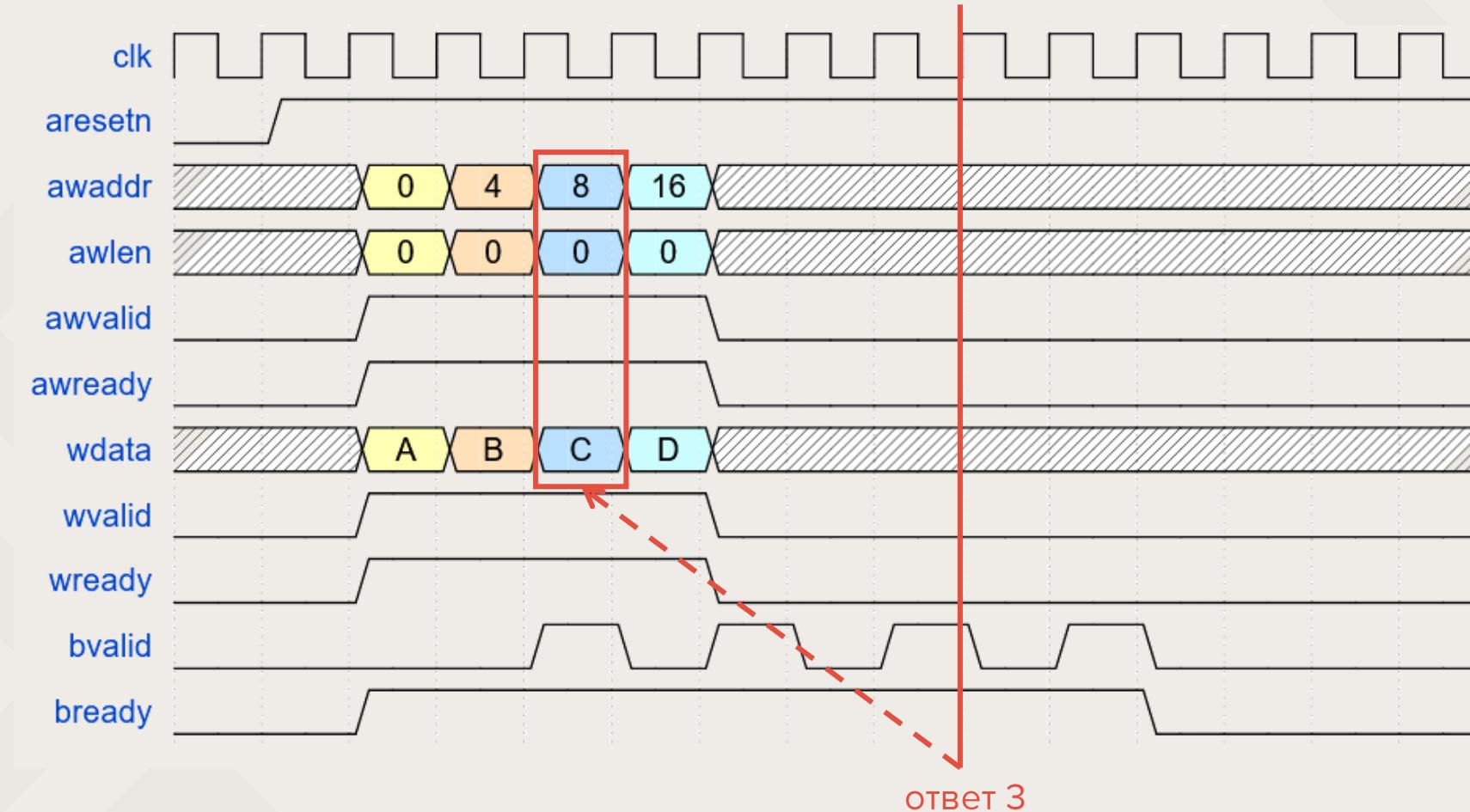
AXI4: Конвейерные транзакции: запись



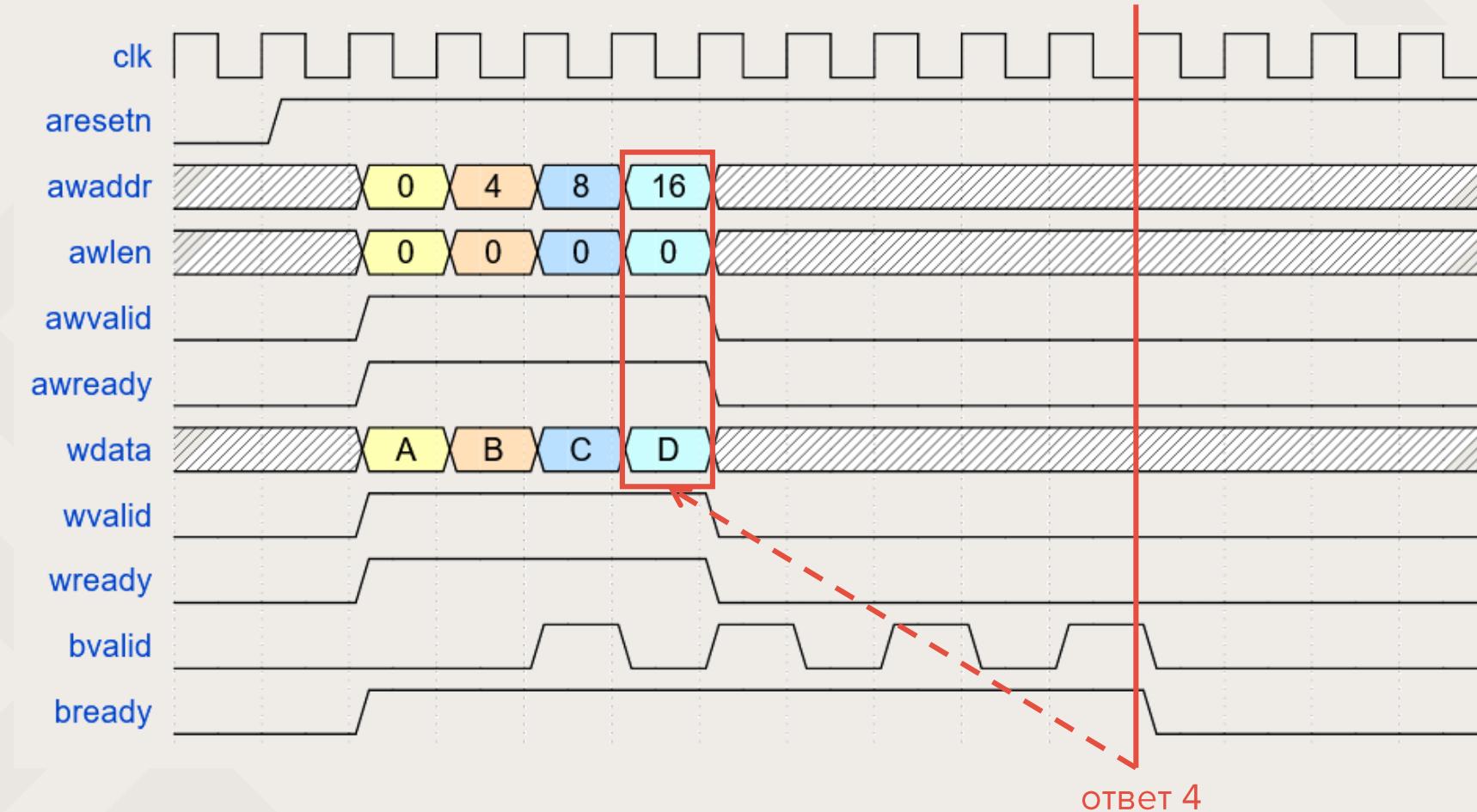
AXI4: Конвейерные транзакции: запись



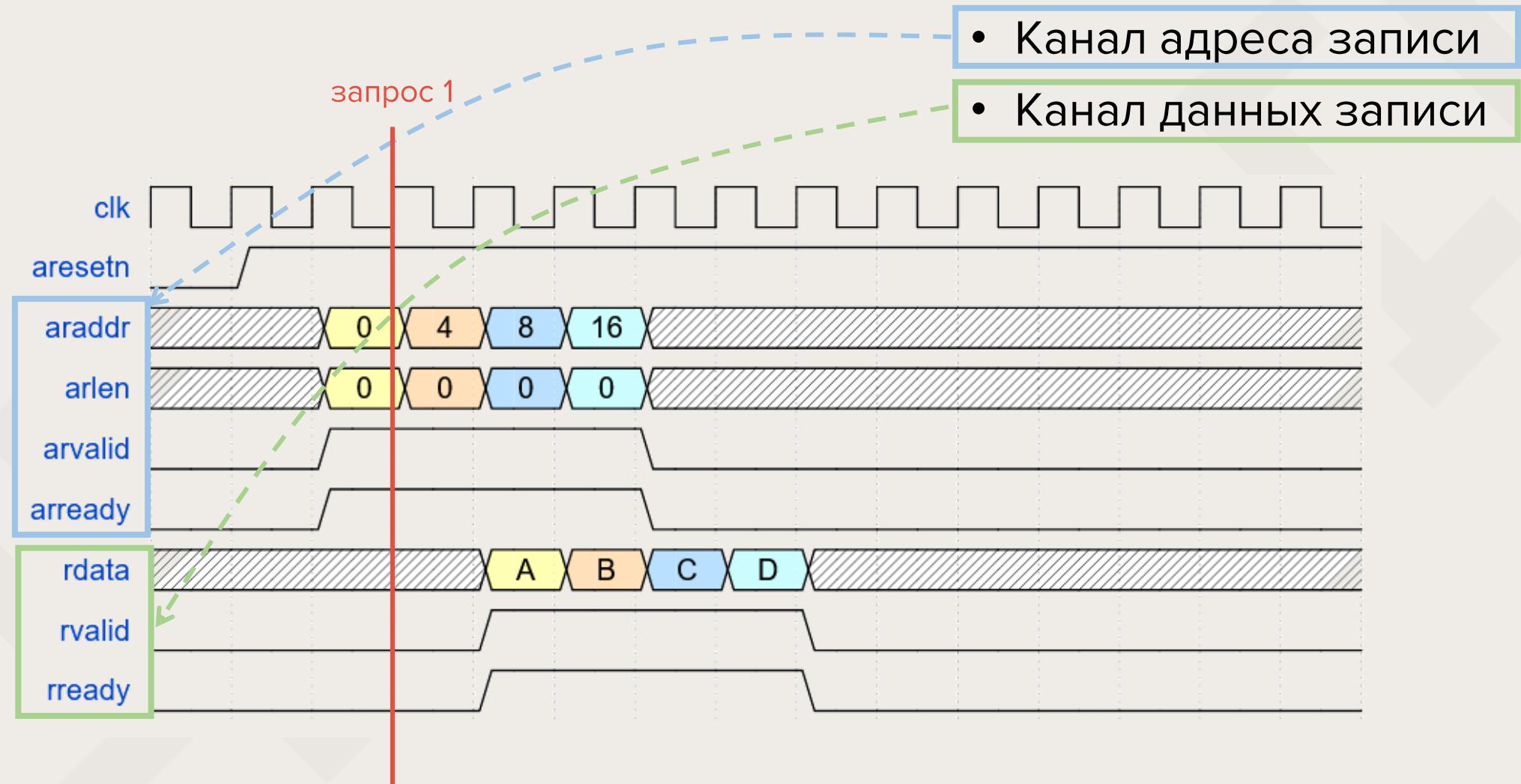
AXI4: Конвейерные транзакции: запись



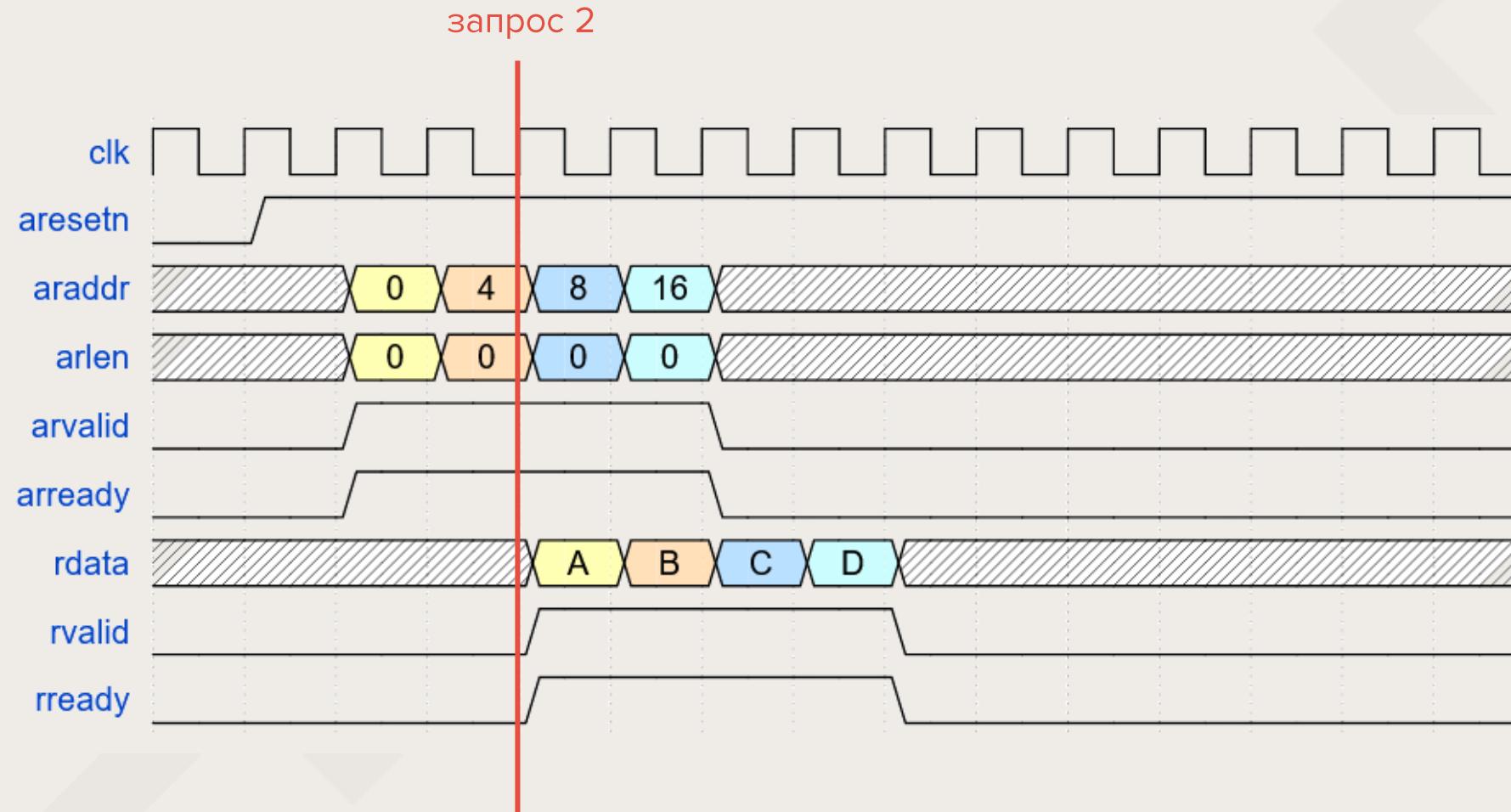
АХ14: Конвейерные транзакции: запись



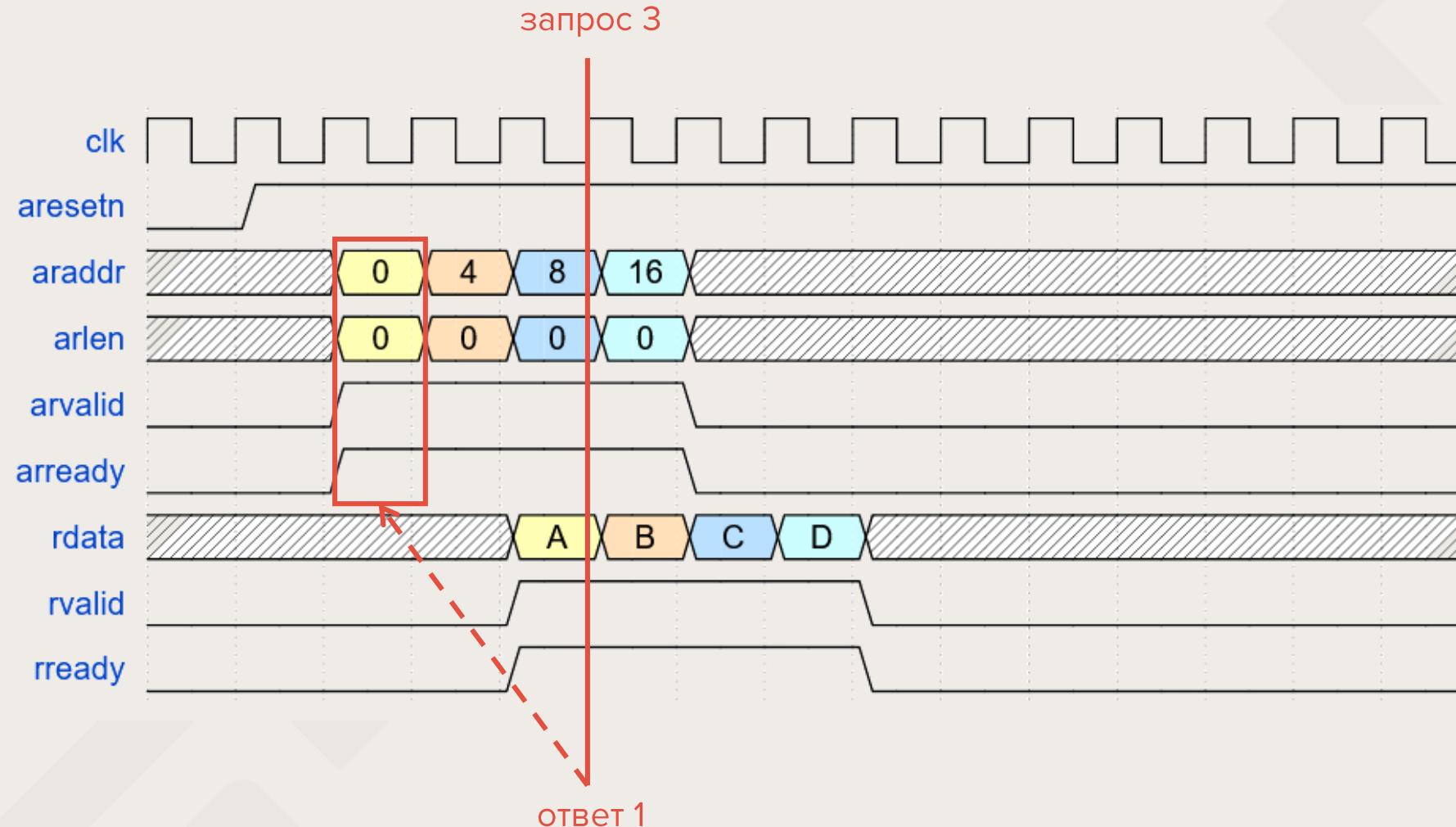
AXI4: Конвейерные транзакции: чтение



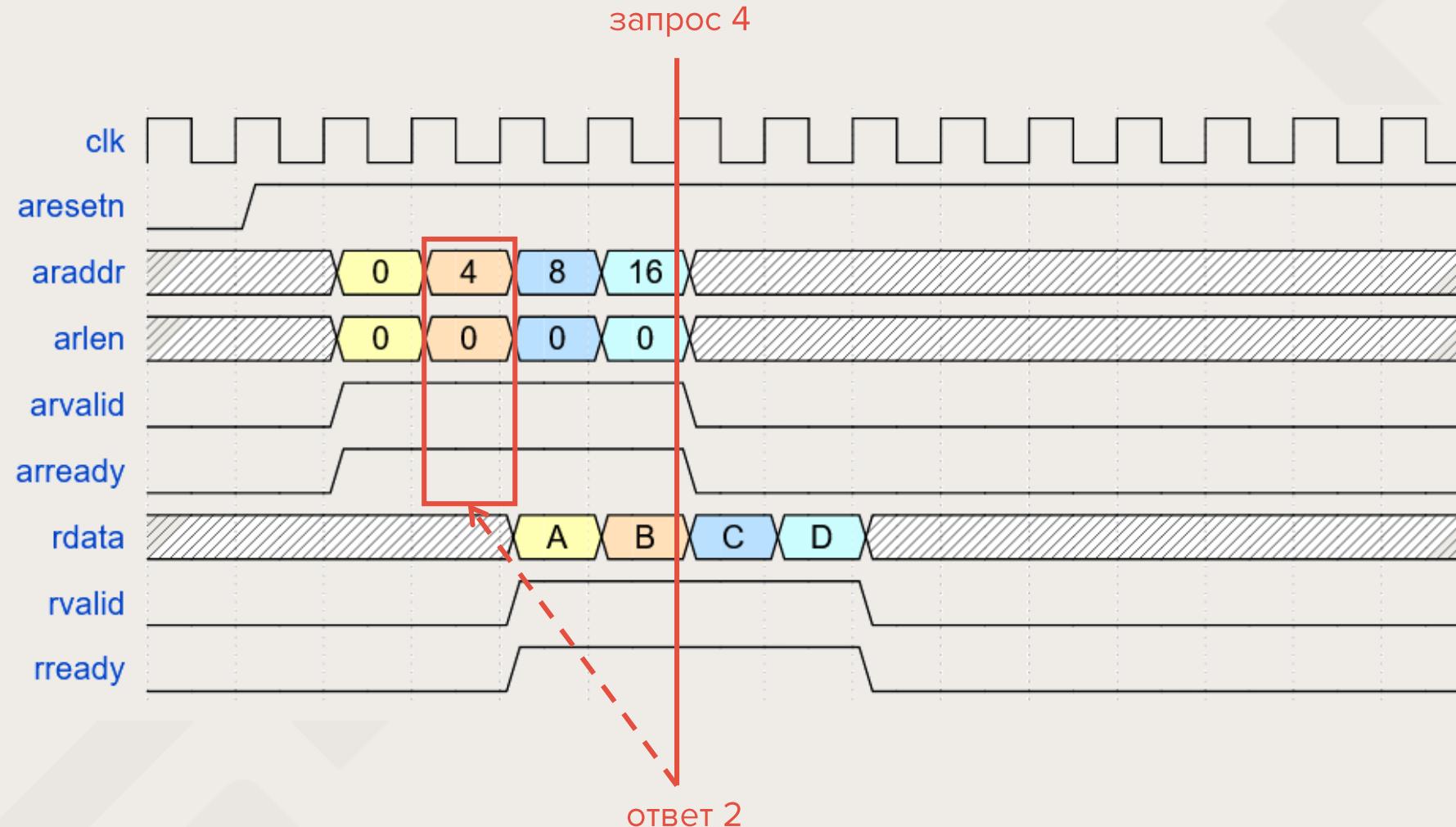
AXI4: Конвейерные транзакции: чтение



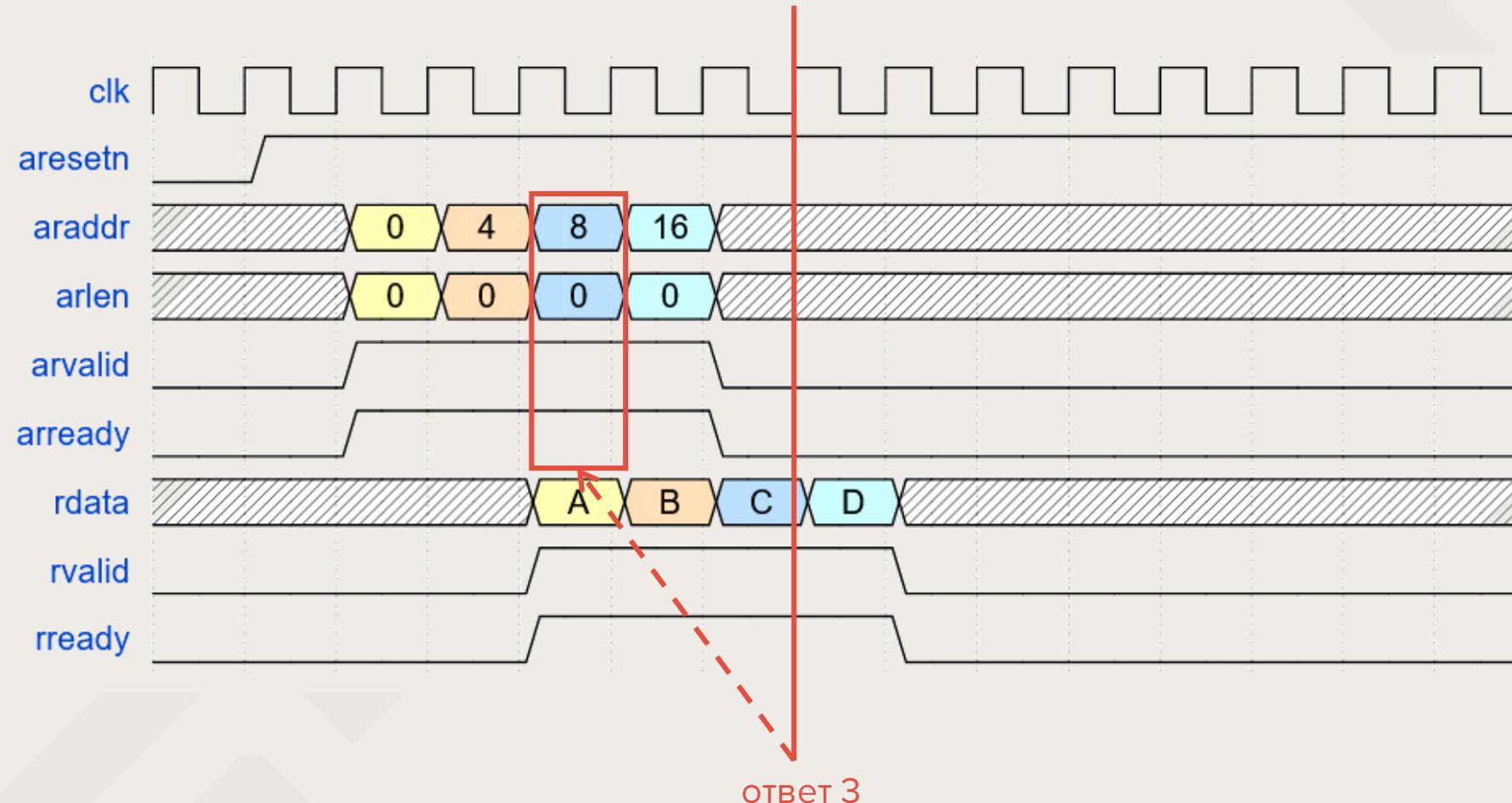
AXI4: Конвейерные транзакции: чтение



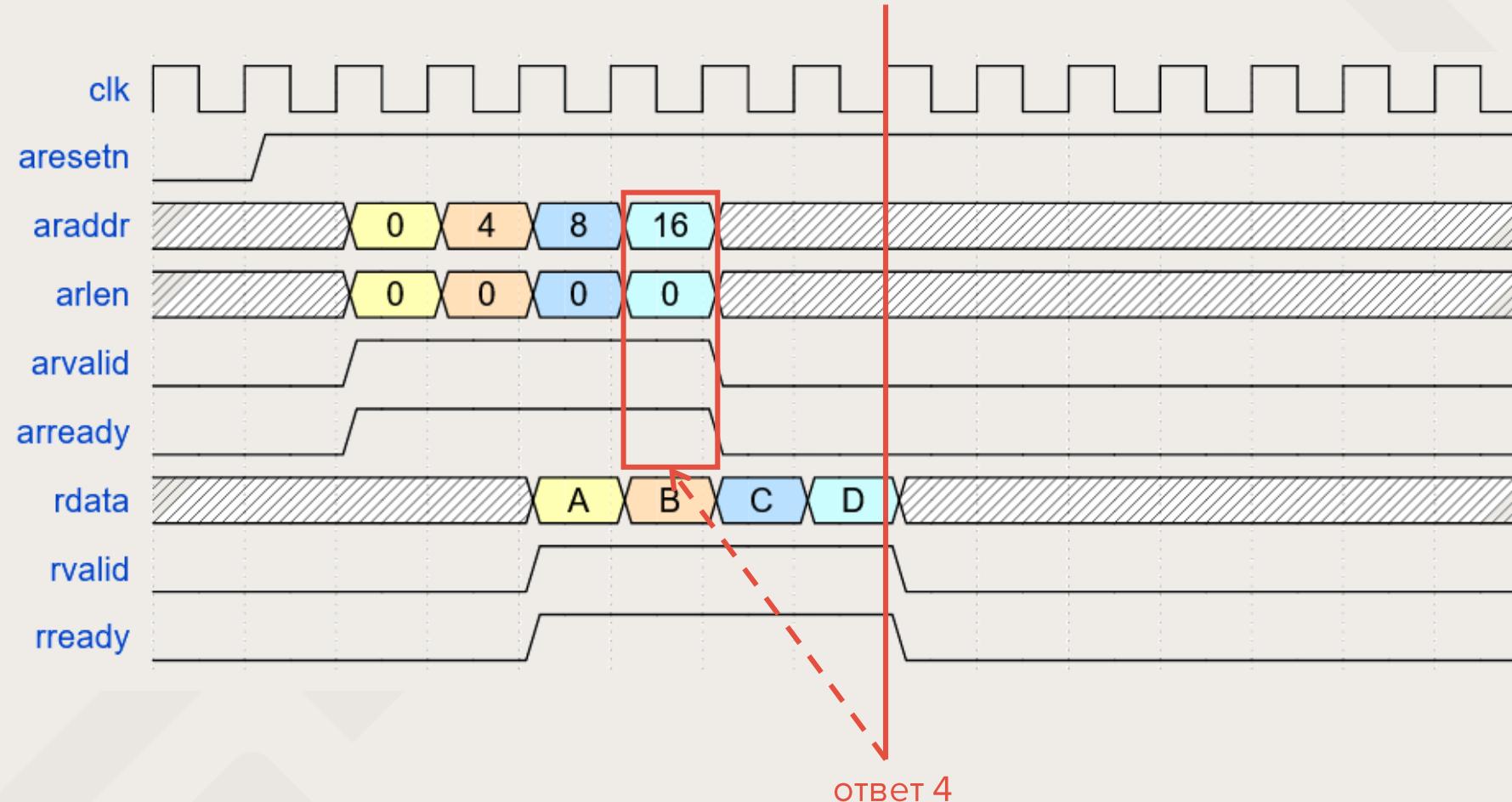
AXI4: Конвейерные транзакции: чтение



AXI4: Конвейерные транзакции: чтение



AXI4: Конвейерные транзакции: чтение

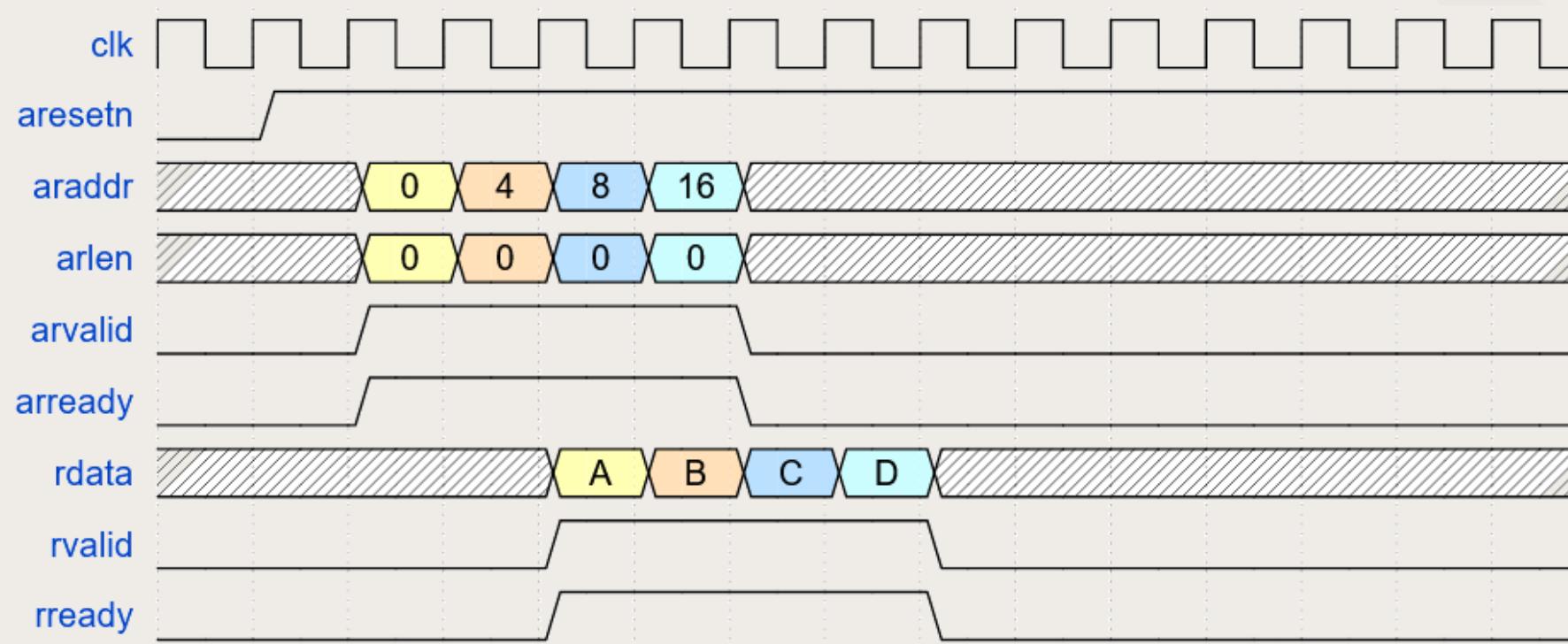


AXI4: Конвейерные транзакции: чтение out-of-order

- В AXI4 **данные чтения не всегда могут быть возвращены в том порядке, в котором были запрошены.**

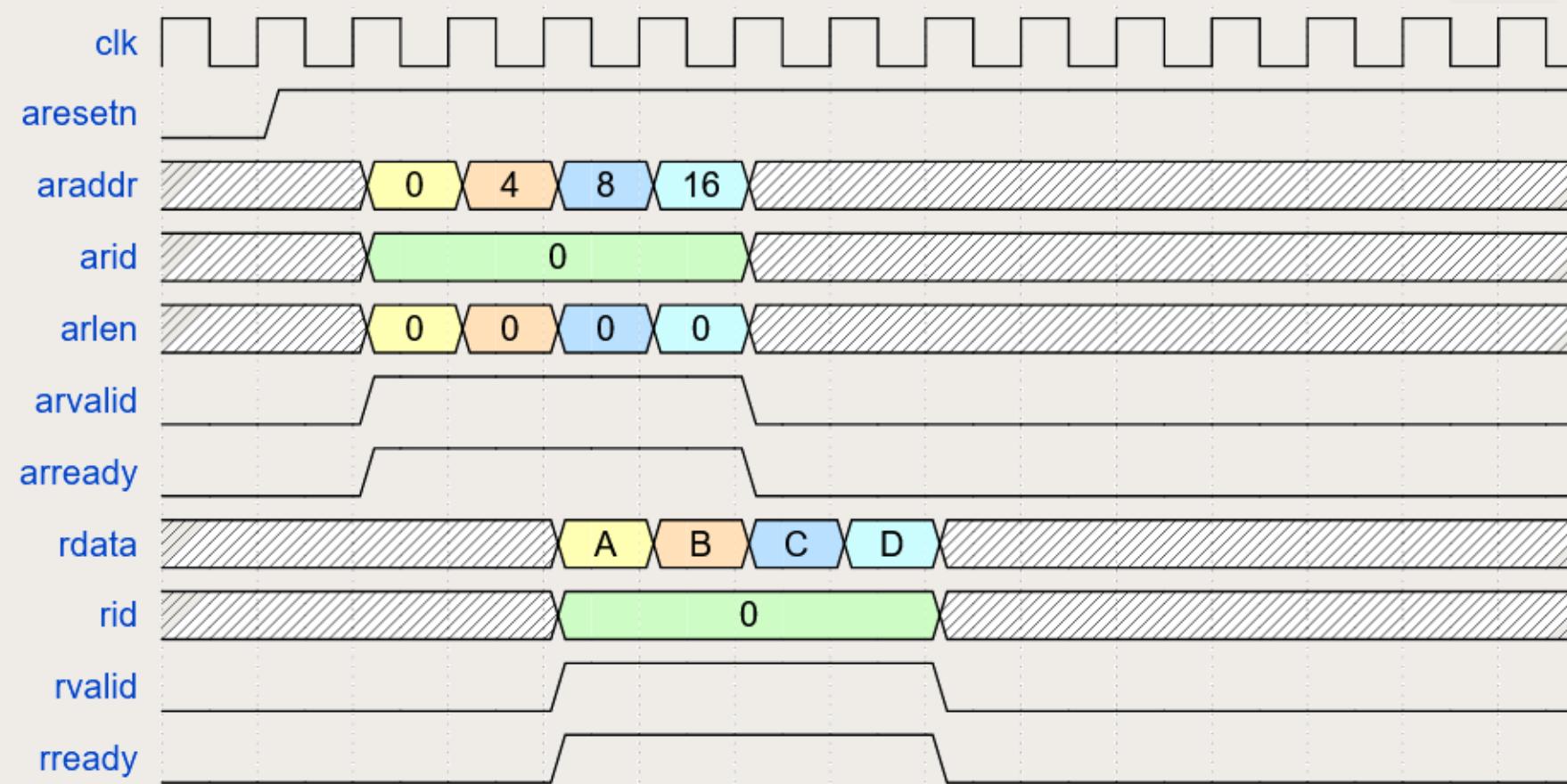
AXI4: Конвейерные транзакции: чтение out-of-order

- В AXI4 **данные чтения не всегда могут быть возвращены в том порядке, в котором были запрошены.**



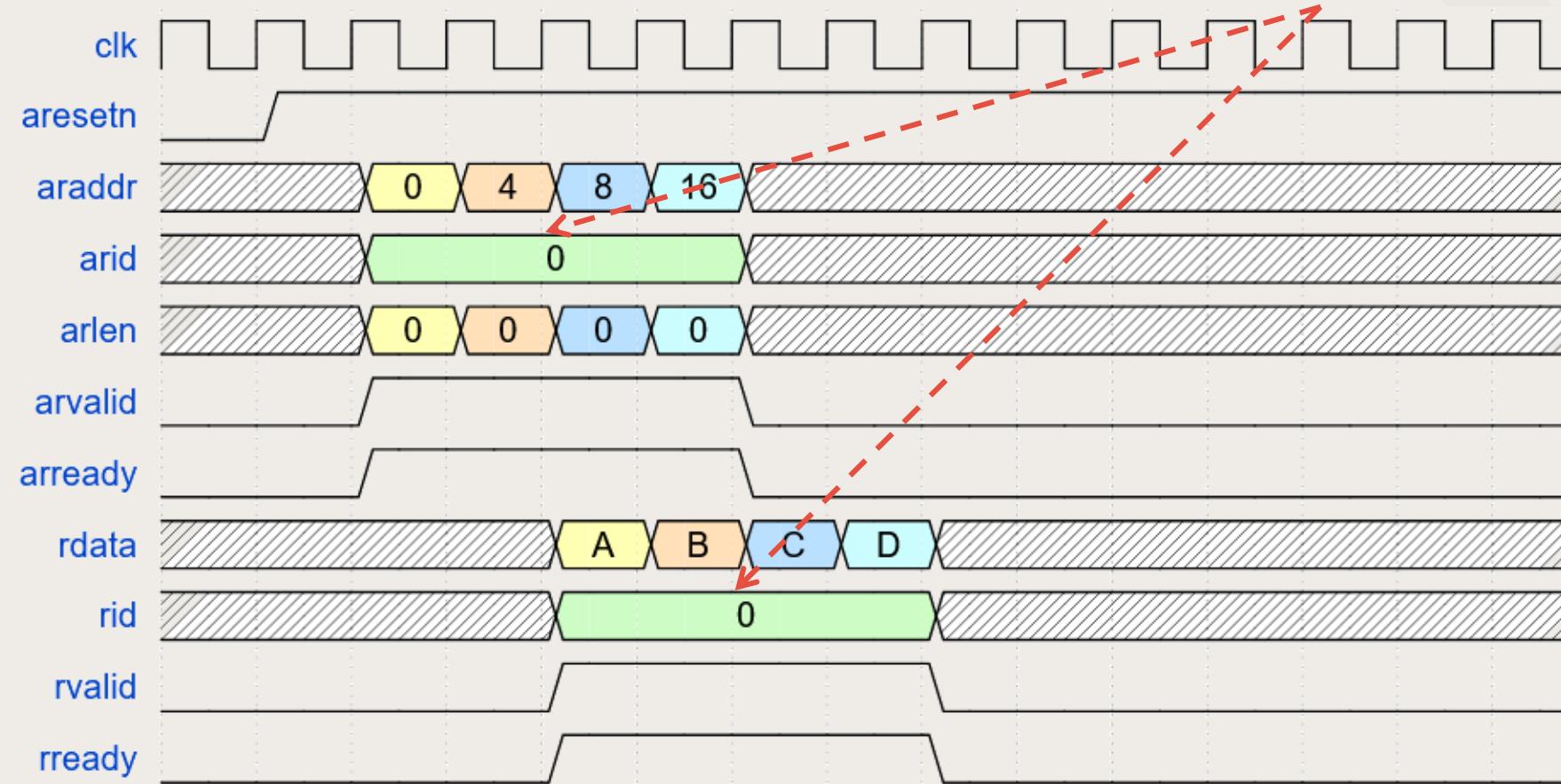
AXI4: Конвейерные транзакции: чтение out-of-order

- В AXI4 данные чтения должны быть возвращены **в запрошенном порядке только для конкретного идентификатора (ID)**.



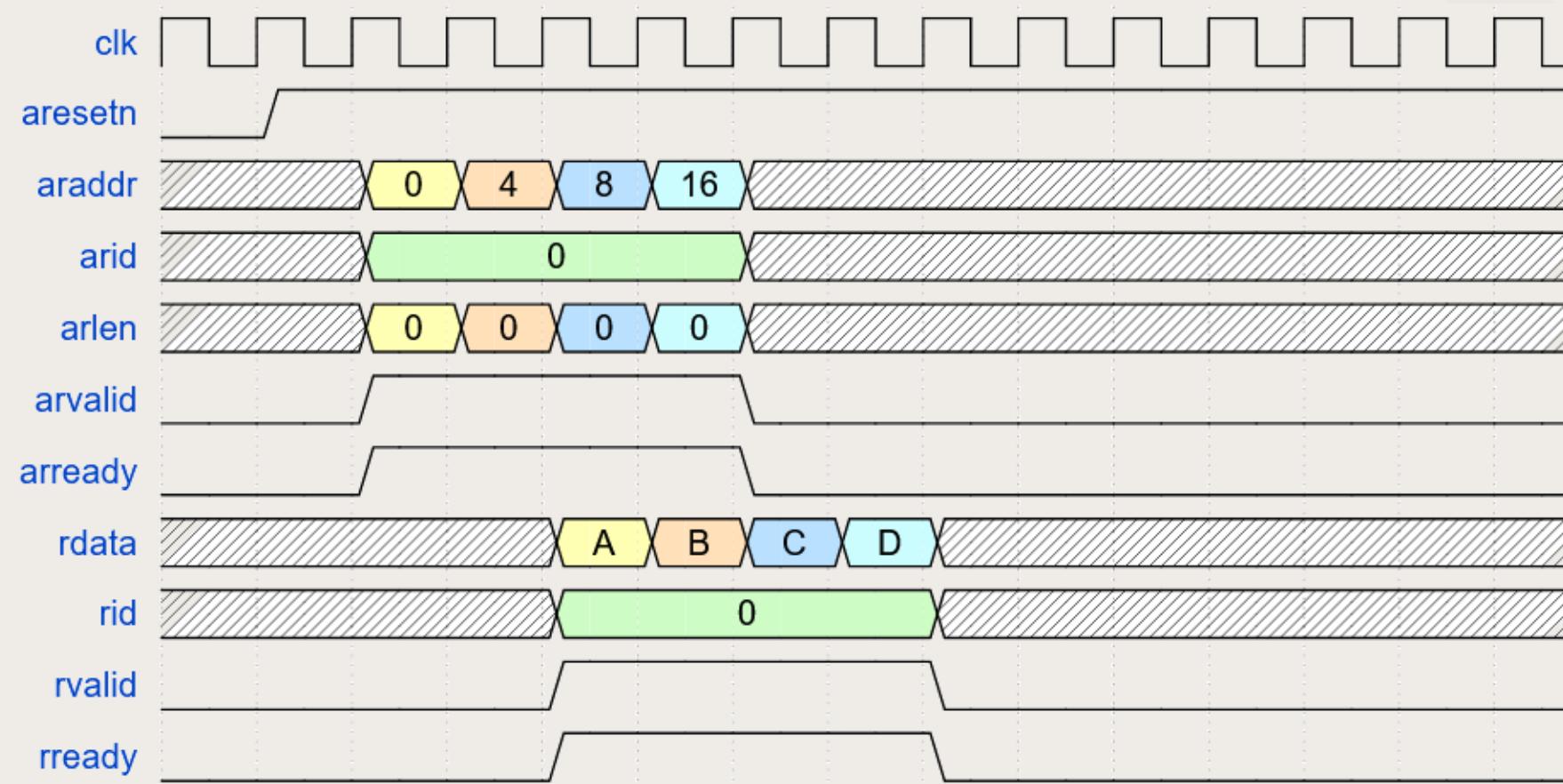
AXI4: Конвейерные транзакции: чтение out-of-order

- В AXI4 данные чтения должны быть возвращены **в запрошенном порядке только для конкретного идентификатора (ID)**.



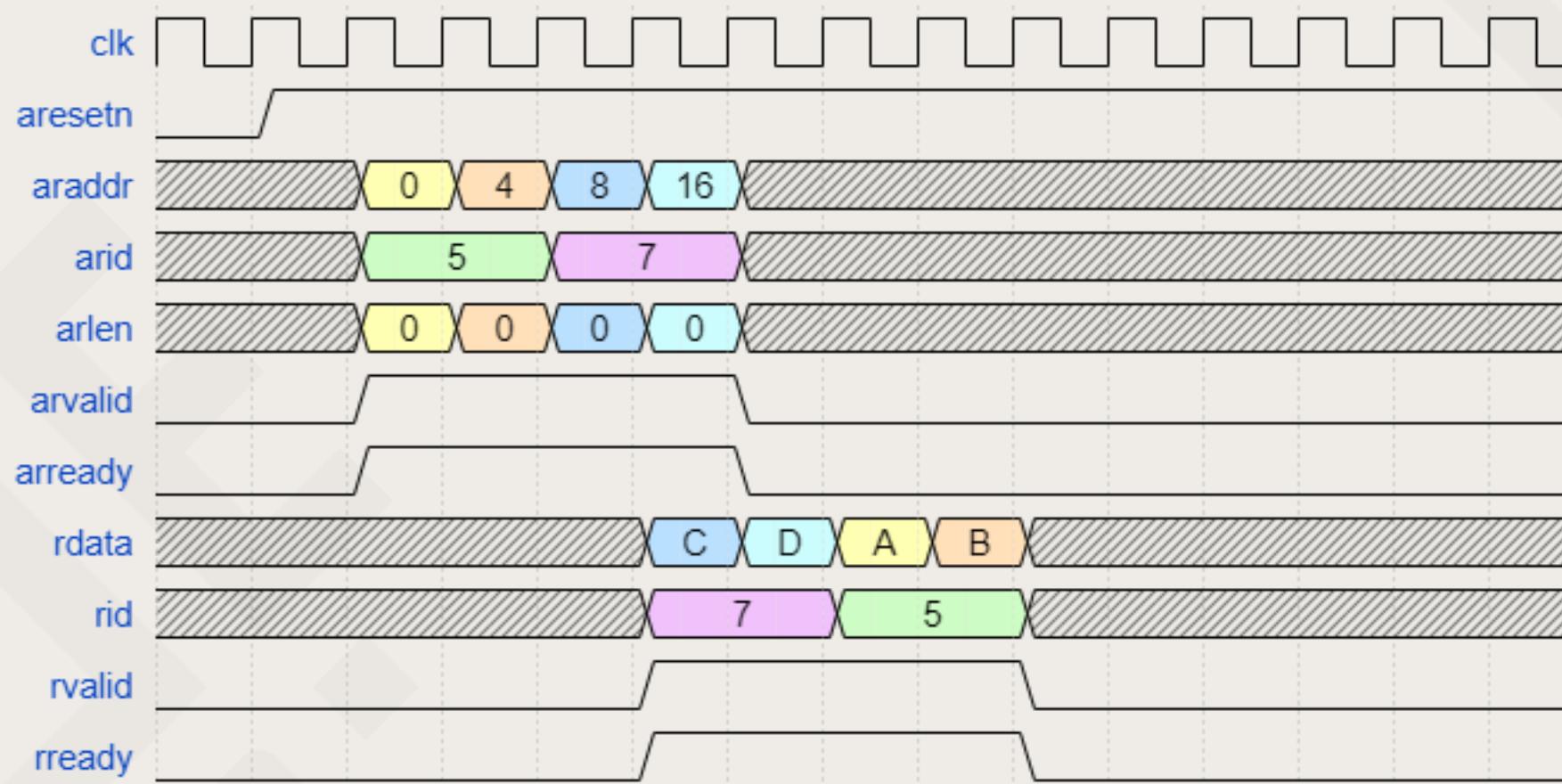
AXI4: Конвейерные транзакции: чтение out-of-order

- В AXI4 данные чтения должны быть возвращены **в запрошенном порядке только для конкретного идентификатора (ID)**.



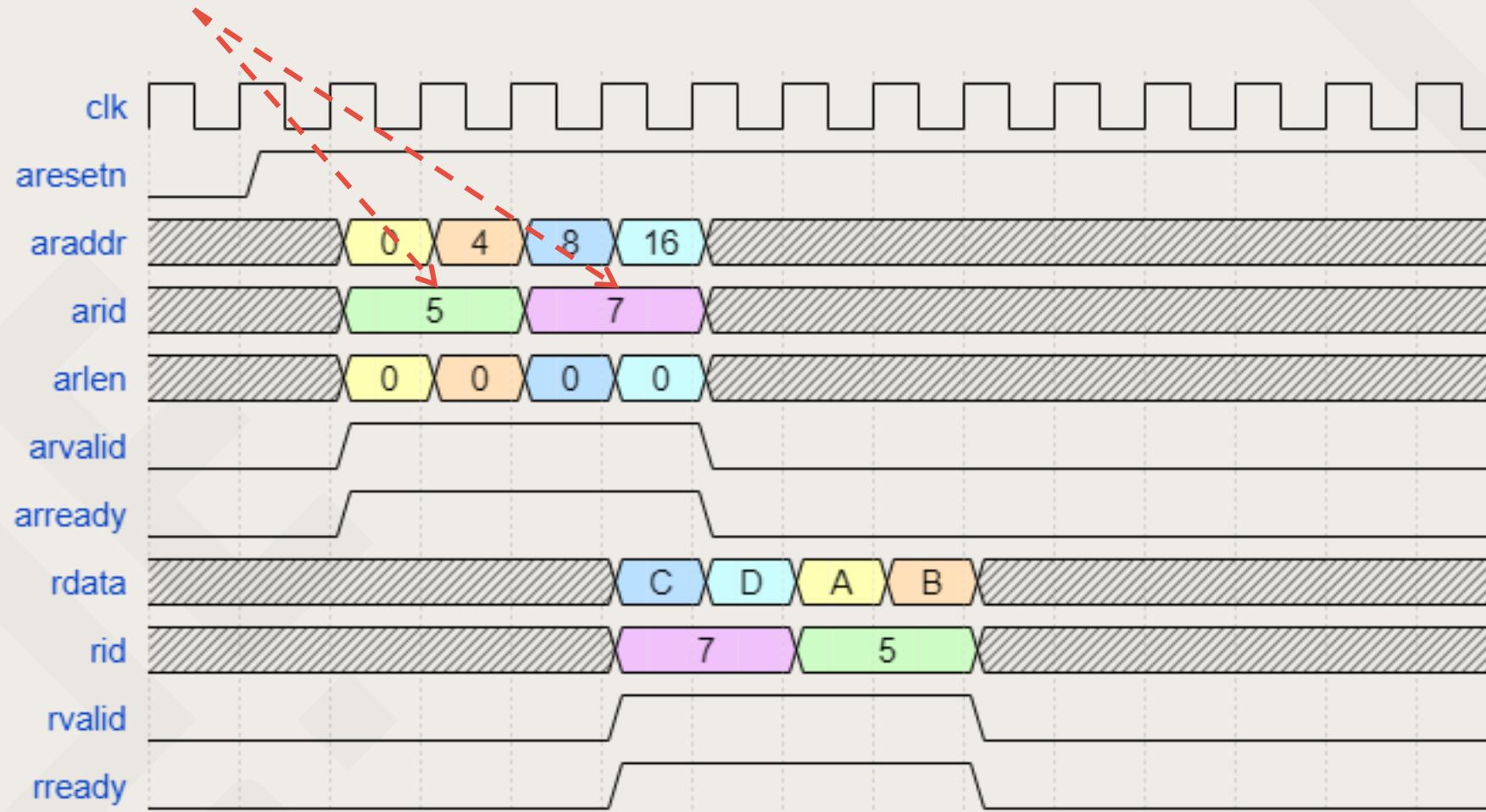
AXI4: Конвейерные транзакции: чтение out-of-order

- Для различных ID последовательность возврата не определена.



AXI4: Конвейерные транзакции: чтение out-of-order

- Для различных ID последовательность возврата **не определена**.



Основная часть

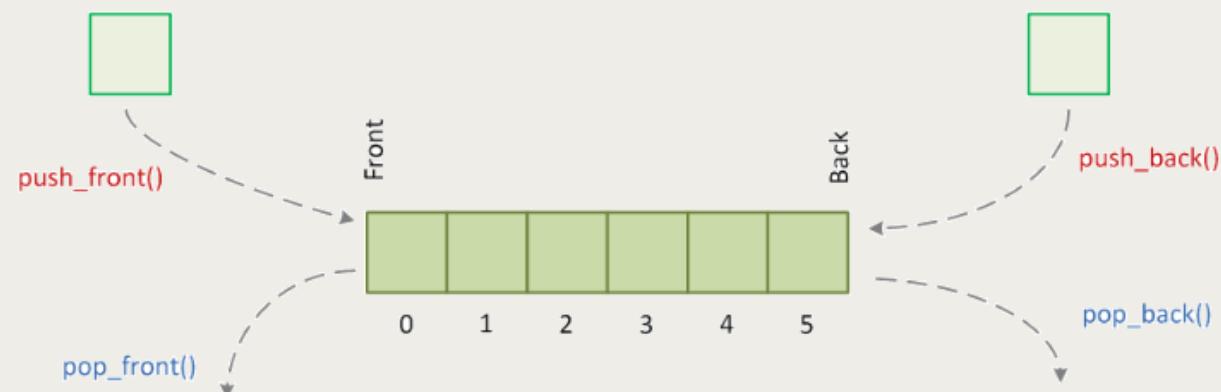
Конструкции SystemVerilog: `typedef`

- Ключевое слово `typedef` используется **для объявления пользовательских типов.**

```
typedef bit [31:0] word_t;  
  
word_t my_word; // bit [31:0] my_word
```

Типы данных SystemVerilog: queue

- Очередь (`queue`) — это **упорядоченный набор однородных элементов** переменной длины.
- Очередь поддерживает константное время доступа к элементам, а также вставку и удаление оных из начала/конца. Каждый элемент очереди определяется порядковым номером, обозначающим его позицию внутри очереди начиная с нуля.



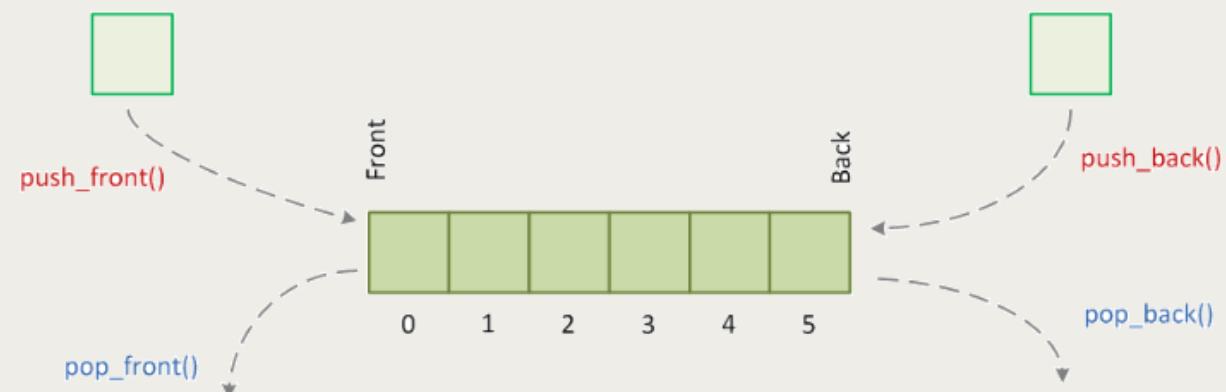
Типы данных SystemVerilog: queue

- Очереди объявляются тем же синтаксисом, что и распакованные массивы, но указывая \$ в качестве размера массива.

```
logic [31:0] my_queue [$]; // Очередь из элементов типа logic  
  
my_queue = {};           // Инициализация пустой очереди  
  
my_queue = {32'd1, 32'd2}; // Инициализация очереди значениями  
                         // Теперь в очереди данные: {1, 2}  
  
logic [31:0] a;  
a = my_queue[0];         // Обращение к нулевому элементу очереди, a = 1
```

Типы данных SystemVerilog: queue

- Для взаимодействия с очередями существуют **специальные методы**, перечислим лишь необходимые в рамках занятий:
 - `size()` – возвращает размер очереди;
 - `push_back(a)` – записывает в конец очереди значение `a`;
 - `pop_front()` – возвращает из начала очереди значение;
 - `delete(i)` – удаляет элемент с позицией `i` в очереди.



Типы данных SystemVerilog: enum

- Тип enum определяет набор именованных значений.
- По умолчанию значения нумеруются с нуля. Можно задавать и пользовательскую нумерацию

```
enum { red, green, blue, yellow } colors;  
// red = 0, green = 1, blue = 2 ...
```

Типы данных SystemVerilog: enum

- Чаще всего **enum** используется в связке с **typedef**.
- Тогда можно создавать переменные объявленного типа.

```
typedef enum { red, green, blue, yellow } colors_e;  
  
colors_e my_color;  
my_color = red;
```

Типы данных SystemVerilog: Ассоциативный массив

- **Ассоциативный массив** — словарь, хранящий пару **«ключ: значение»**.
- Для элемента ассоциативного массива **не выделяется память** до тех пор, пока он не будет “создан”.
- Ключ ассоциативного массива **может быть любого типа**.
- Темплейт:

```
<тип данных> <имя массива> [<тип ключа>]
```

Ключ	Значение
“One”	1
“Two”	2
...	...
“Inf”	-1

Типы данных SystemVerilog: Ассоциативный массив

- **Инициализация** происходит при помощи **присваиваний**:
- Пример:

```
int ass_arr [string];
```

Ключ	Значение

Типы данных SystemVerilog: Ассоциативный массив

- Инициализация происходит при помощи присваиваний:
- Пример:

```
int ass_arr [string];  
  
ass_arr[ "One" ] = 1;
```

Ключ	Значение
“One”	1

Типы данных SystemVerilog: Ассоциативный массив

- **Инициализация** происходит при помощи **присваиваний**:
- Пример:

```
int ass_arr [string];  
  
ass_arr[ "One" ] = 1;  
ass_arr[ "Two" ] = 2;
```

Ключ	Значение
“One”	1
“Two”	2

Типы данных SystemVerilog: Ассоциативный массив

- **Обращение** происходит **по ключу**:
- Пример:

```
int ass_arr [string];  
  
ass_arr[ "One" ] = 1;  
ass_arr[ "Two" ] = 2;  
  
int val = ass_arr[ "One" ]; // Returns 1
```

Ключ	Значение
“One”	1
“Two”	2

Типы данных SystemVerilog: Ассоциативный массив

- Удаление происходит при помощи метода `delete()`:
- Пример:

```
int ass_arr [string];  
  
ass_arr[ "One" ] = 1;  
ass_arr[ "Two" ] = 2;  
  
int val = ass_arr[ "One" ]; // Returns 1  
  
ass_arr[ "Two" ].delete(); // Deletes 2
```

Ключ	Значение
“One”	1

Конструкции SystemVerilog: fork-join

- Для запуска нескольких процессов параллельно используется конструкция **fork-join**.
- Пример:

```
initial begin
    #10 $display("%0t Process 0", $time());
    fork
        #10 $display("%0t Process 1", $time());
        #15 $display("%0t Process 2", $time());
    join
    #3 $display("%0t Process 3", $time());
end
```

Конструкции SystemVerilog: fork-join

- Пример:

```
initial begin
    #10 $display("%0t Process 0", $time());
    fork
        #10 $display("%0t Process 1", $time());
        #15 $display("%0t Process 2", $time());
    join
    #3 $display("%0t Process 3", $time());
end
```

- Результат:

```
# 10 Process 0
# 20 Process 1
# 25 Process 2
# 28 Process 3
```

Statement - 1

Statement - 2

fork

Process - 1

Process - 2

Process - 3

join

Statement - 3

Statement - 4

Конструкции SystemVerilog: fork-join_any

- Выполнение кода после **fork-join** начинается после того, как **все параллельные процессы будут завершены**. Для **fork-join_any** выполнение продолжается после того, как **хотя бы один процесс будет завершен**.
- Пример:

```
initial begin
    #10 $display("%0t Process 0", $time());
    fork
        #10 $display("%0t Process 1", $time());
        #15 $display("%0t Process 2", $time());
    join_any
    #3 $display("%0t Process 3", $time());
end
```

Конструкции SystemVerilog: fork-join_any

- Пример:

```
initial begin
    #10 $display("%0t Process 0", $time());
    fork
        #10 $display("%0t Process 1", $time());
        #15 $display("%0t Process 2", $time());
    join_any
    #3 $display("%0t Process 3", $time());
end
```

- Результат:

```
# 10 Process 0
# 20 Process 1
# 23 Process 3
# 25 Process 2
```

Statement - 1

Statement - 2

fork

Process - 1

Process - 2

Process - 3

join_any

Statement - 3

Statement - 4

Конструкции SystemVerilog: fork-join_none

- Выполнение кода после **fork-join** начинается после того, как **все параллельные процессы будут завершены**. Для **fork-join_none** выполнение продолжается **сразу после запуска этого блока**.
- Пример:

```
initial begin
    #10 $display("%0t Process 0", $time());
    fork
        #10 $display("%0t Process 1", $time());
        #15 $display("%0t Process 2", $time());
    join_none
    #3 $display("%0t Process 3", $time());
end
```

Конструкции SystemVerilog: fork-join_none

- Пример:

```
initial begin
    #10 $display("%0t Process 0", $time());
    fork
        #10 $display("%0t Process 1", $time());
        #15 $display("%0t Process 2", $time());
    join_none
    #3 $display("%0t Process 3", $time());
end
```

- Результат:

```
# 10 Process 0
# 13 Process 3
# 20 Process 1
# 25 Process 2
```

Statement - 1

Statement - 2

fork

Process - 1

Process - 2

Process - 3

join_none

Statement - 3

Statement - 4

SystemVerilog: package

- **package** является своего рода "коллекцией", содержащей объявленные пользователем элементы. Это могут быть объявления типов, задач функций, свойств.
- Доступ к элементам **package** осуществляется при помощи оператора **::** и при помощи ключевого слова **import**.

SystemVerilog: package

- Доступ к элементам **package** осуществляется при помощи оператора **::** или при помощи ключевого слова **import**.

```
package my_pkg;  
    typedef enum { red, green, blue, yellow} colors_e;  
endpackage
```

```
module my_module;  
    import my_pkg::colors_e;  
    colors_e color;  
endmodule
```

SystemVerilog: package

- Доступ к элементам **package** осуществляется при помощи оператора **::** или при помощи ключевого слова **import**.

```
package my_pkg;  
    typedef enum { red, green, blue, yellow} colors_e;  
endpackage
```

```
module my_module;  
    import my_pkg::*;  
    colors_e color;  
endmodule
```

SystemVerilog: package

- Доступ к элементам **package** осуществляется при помощи оператора **::** или при помощи ключевого слова **import**.

```
package my_pkg;  
    typedef enum { red, green, blue, yellow} colors_e;  
endpackage  
  
module my_module;  
  
    my_pkg::colors_e color;  
endmodule
```

Разбор AXI4 VIP

AXI4 VIP

- <https://gitflic.ru/project/yuri-panchul/valid-ready-etc>
- valid-ready-etc/boards/omdazz/08_axi_master_slave_monitor

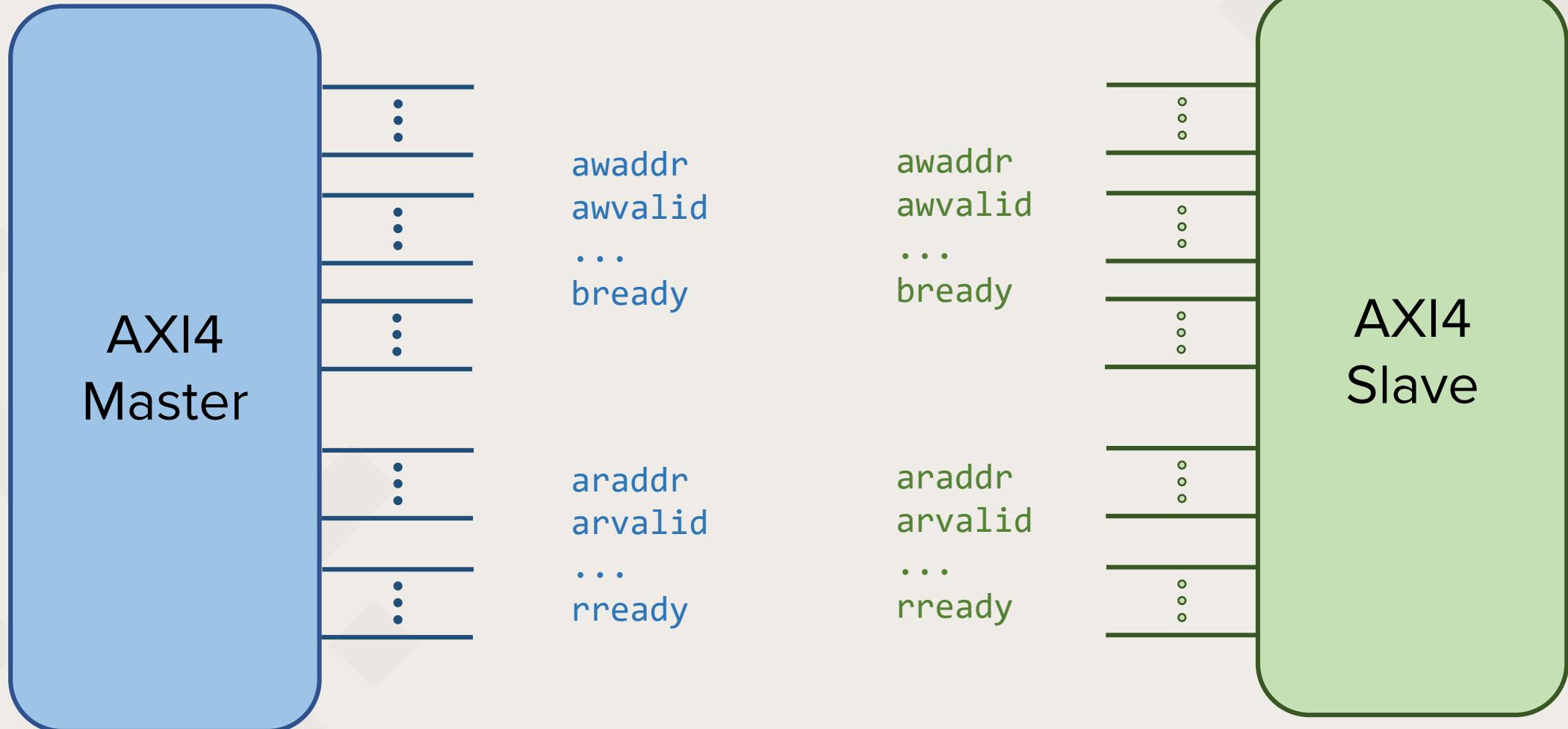
AXI4 VIP: Структурная схема



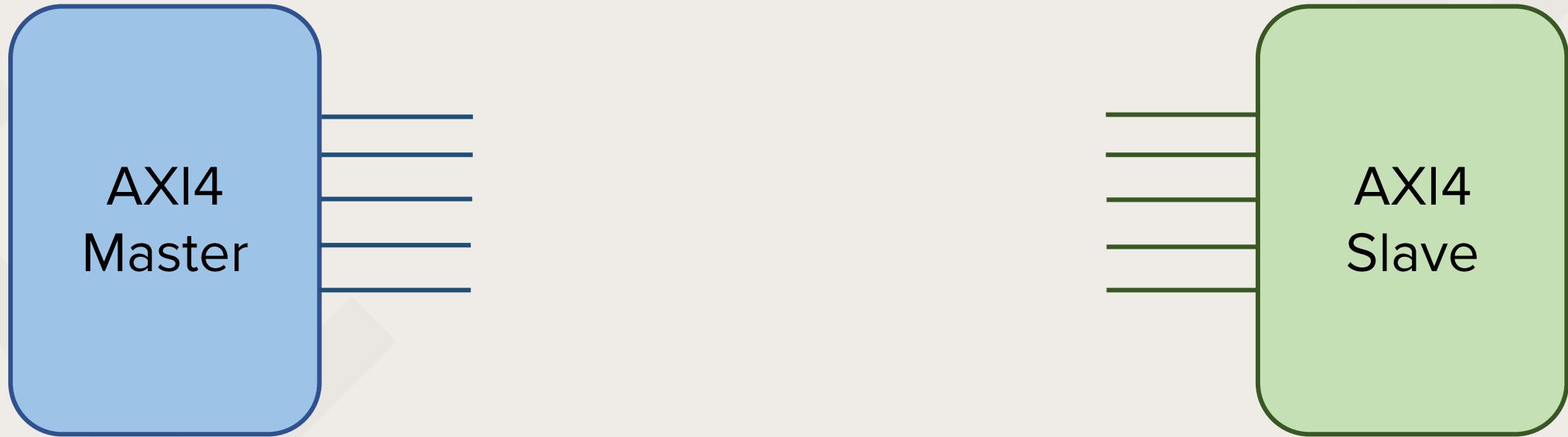
Запись

Чтение

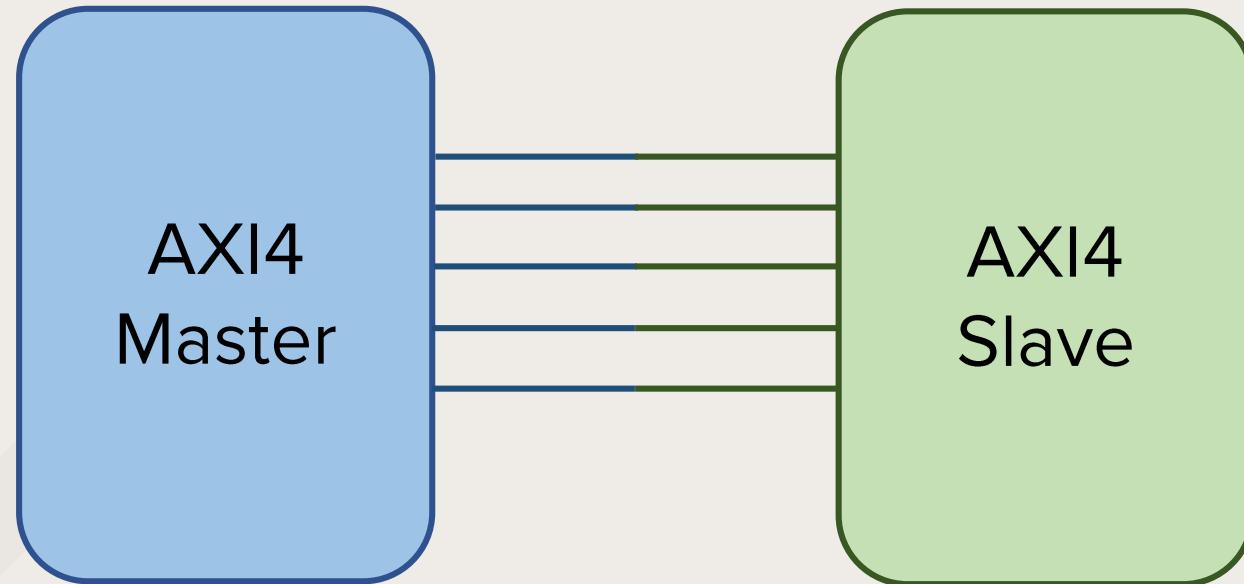
AXI4 VIP: Структурная схема



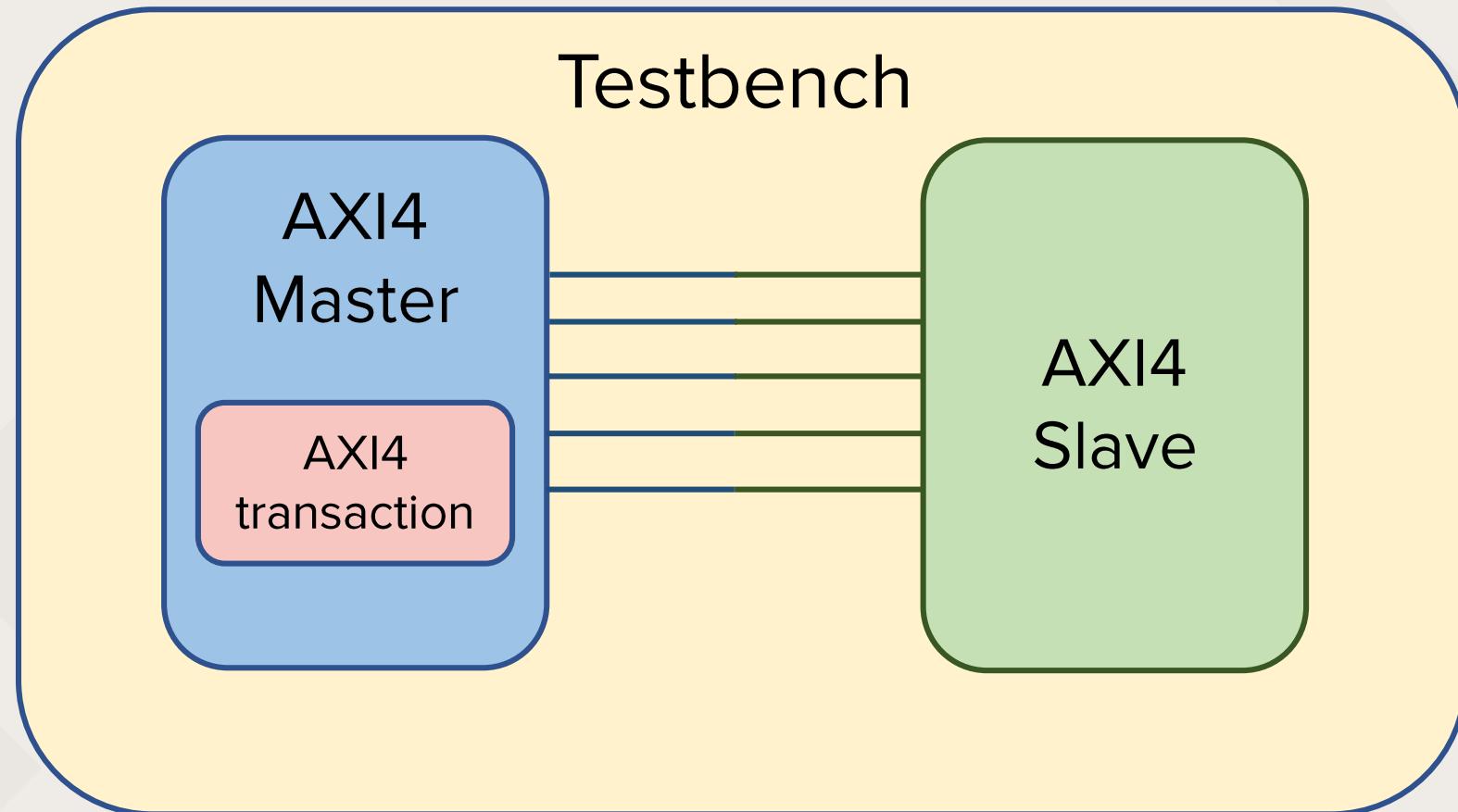
AXI4 VIP: Структурная схема



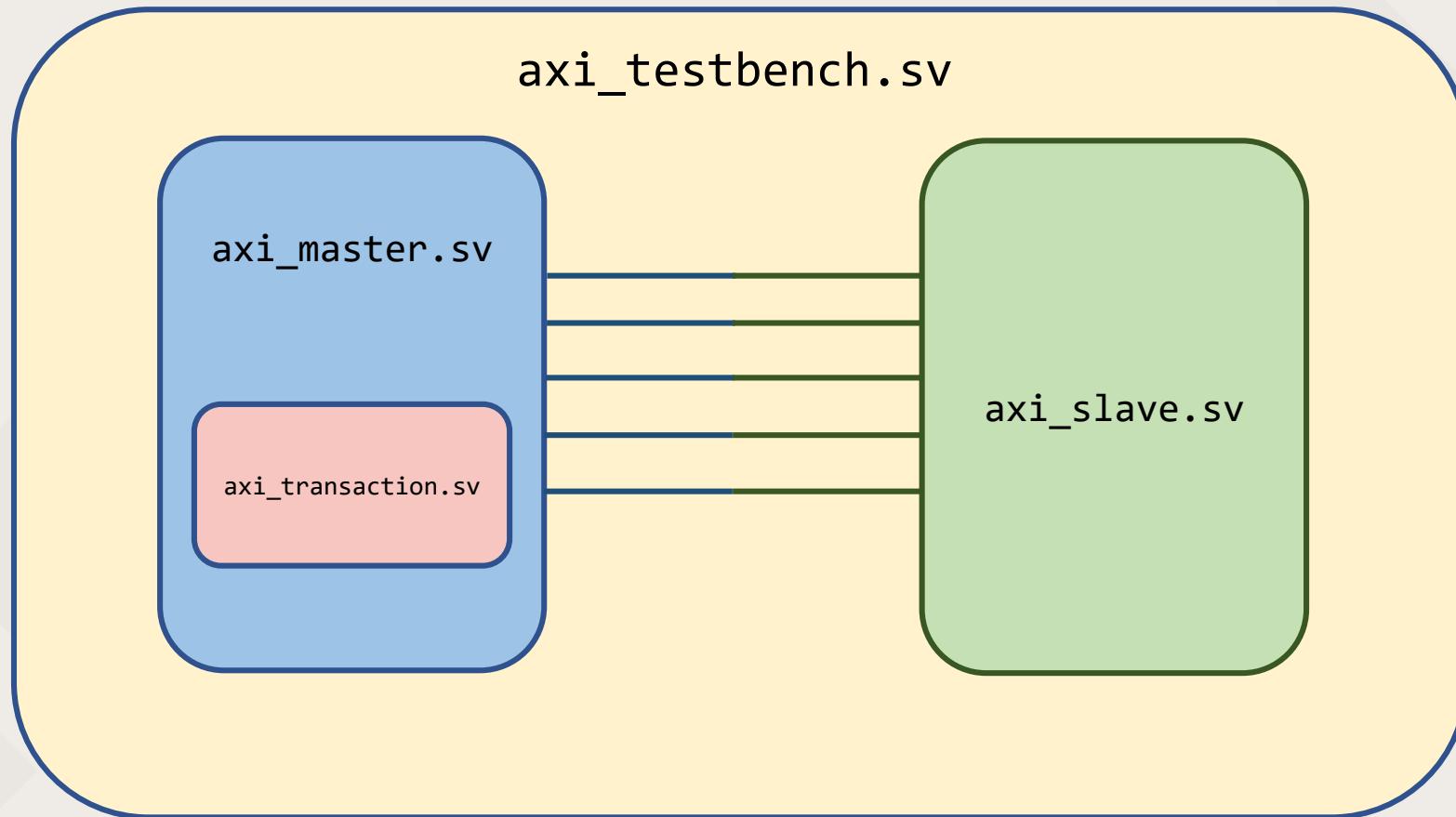
AXI4 VIP: Структурная схема



AXI4 VIP: Структурная схема



AXI4 VIP: Структурная схема



AXI4 VIP: axi_transaction.sv

- 08_axi_master_slave_monitor/51_axi_transaction

AXI4 VIP: axi_transaction.sv

```
package axi_transaction;  
  
endpackage
```

- **Package**

AXI4 VIP: axi_transaction.sv

```
package axi_transaction;

parameter addr_width  = 32,
          data_width   = 32,
          id_width     = 4,
          max_delay    = 100;

parameter n_ids        = 1 << id_width,
          delay_width  = $clog2 (max_delay + 1);

endpackage
```

- **Параметры**

- ширины сигналов;
- максимальная задержка данных и адресов;
- максимальное количество ID.

AXI4 VIP: axi_transaction.sv

```
package axi_transaction;  
...  
typedef enum { read, write } op_t;  
  
typedef logic [addr_width - 1:0] addr_t;  
typedef logic [data_width - 1:0] data_t;  
typedef logic [id_width - 1:0] id_t;  
typedef logic [delay_width - 1:0] delay_t;  
  
endpackage
```

- **Определение пользовательских типов**
- **op_t** отвечает за тип операции: запись или чтение.

AXI4 VIP: axi_transaction.sv

```
package axi_transaction;  
...  
  
class axi_transaction;  
...  
  
endclass  
  
endpackage
```

- **Основной класс AXI4 транзакции**

AXI4 VIP: axi_transaction.sv

```
...
class axi_transaction;

rand op_t      op;
rand addr_t    addr;

rand data_t    data;

rand id_t      id;

rand delay_t   addr_delay;
rand delay_t   data_delay;

endclass
...
```

- **Основная конфигурация**

- Адрес
- Данные
- ID
- Задержки

AXI4 VIP: axi_transaction.sv

```
...
class axi_transaction;
  ...
  bit data_is_set;
  bit addr_is_sent;
  bit data_is_sent;
endclass
...
```

- **Вспомогательные поля**
- Информация о состоянии обработки транзакции
- Эти поля будут использоваться в VIP

AXI4 VIP: axi_transaction.sv

```
...
class axi_transaction;
...
constraint addr_c {
    addr dist
    {
        [ 0      : 3                  ] := 10,
        [ 4      : 9                  ] :/ 50,
        [ 32'ha : 32'hffffffffff ] :/ 10
    };
}
endclass
...
```

- **Ограничения**
- Адрес
- **dist** – распределение

AXI4 VIP: axi_transaction.sv

```
...
class axi_transaction;
    ...
constraint id_c {
    id < n_ids;
    op == write -> id == 0;
}
endclass
...
```

- **Ограничения**
 - ID
 - Ограничиваем количество ID
 - Фиксируем ID для записи

AXI4 VIP: axi_transaction.sv

```
...
class axi_transaction;
  ...
  constraint addr_data_delay_c {
    addr_delay <= max_delay;
    if (op == read)
      data_delay == 0;
    else
      data_delay <= max_delay;
    ...
  }
endclass
...
```

- **Ограничения**
 - Задержки для данных и адреса
 - Через сколько тактов выставлять адрес и данные на линию

AXI4 VIP: axi_transaction.sv

```
...
class axi_transaction;
  ...
constraint addr_data_delay_c {
  ...
    signed#(data_delay) -> signed#(addr_delay) dist {
      0      := 30,
      [-1:1] := 30,
      [-3:3] := 35,
      [-5:5] := 5
    };
}
endclass
...
```

- **Ограничения**
 - Задержки для данных и адреса
 - "Расстояние" между адресом и данными

AXI4 VIP: axi_transaction.sv

```
...
class axi_transaction;
    ...
function string str ();
    string s;
    ...
    if (op == read & id != 0)
        s = { s, $sformatf (" id=%0d", id) };
endfunction
endclass
...
```

- Вывод полей класса

AXI4 VIP: axi_transaction.sv

- **Практика.**
- 08_axi_master_slave_monitor/51_axi_transaction/axi_testbench.sv

AXI4 VIP: axi_master.sv и axi_slave.sv

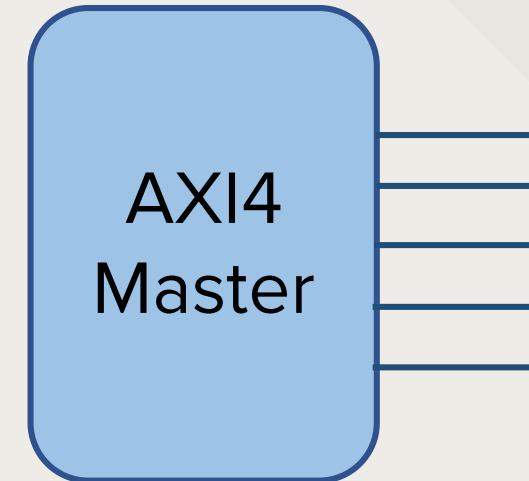
- 08_axi_master_slave_monitor/52_axi_master_no_resp

AXI4 VIP: axi_master.sv

```
module axi_master
(
    input          clk,
    input          rst,
    output addr_t araddr,
    output id_t   arid,
    output logic  arvalid,
    input          arready,
    ...
    output addr_t awaddr,
    output logic  bready
);

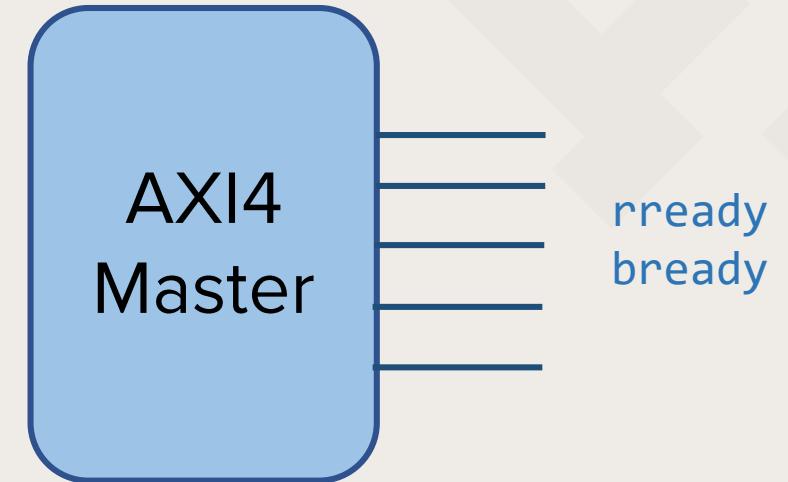
import axi_transaction::*;

...
```



AXI4 VIP: axi_master.sv

```
...  
  
    always @ (posedge clk) begin  
        rready <= 0; // No read support ...  
        bready <= 0; // No write response ...  
    end  
  
...  
...
```



AXI4 VIP: axi_master.sv

```
...
initial reset_probabilities();

//-----  
// Queues and scoreboards

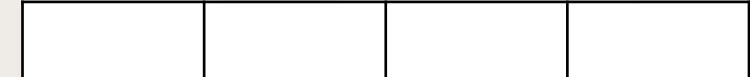
axi_transaction send_queue [$];
axi_transaction receive_queue [$];

axi_transaction wr_addr_queue [$];
axi_transaction wr_data_queue [$];

axi_transaction wr_resp_queue [$];

`define complete_everything wait fork;
...
```

send_queue



recieve_queue



wr_addr_queue



wr_data_queue



wr_resp_queue



AXI4 VIP: axi_master.sv

```
...
task automatic start_write
(
...
);

fork
begin
    ...
    tr = new();
    assert (tr.randomize () with
    {
        ...
    });
    send_queue.push_back(tr);
...

```

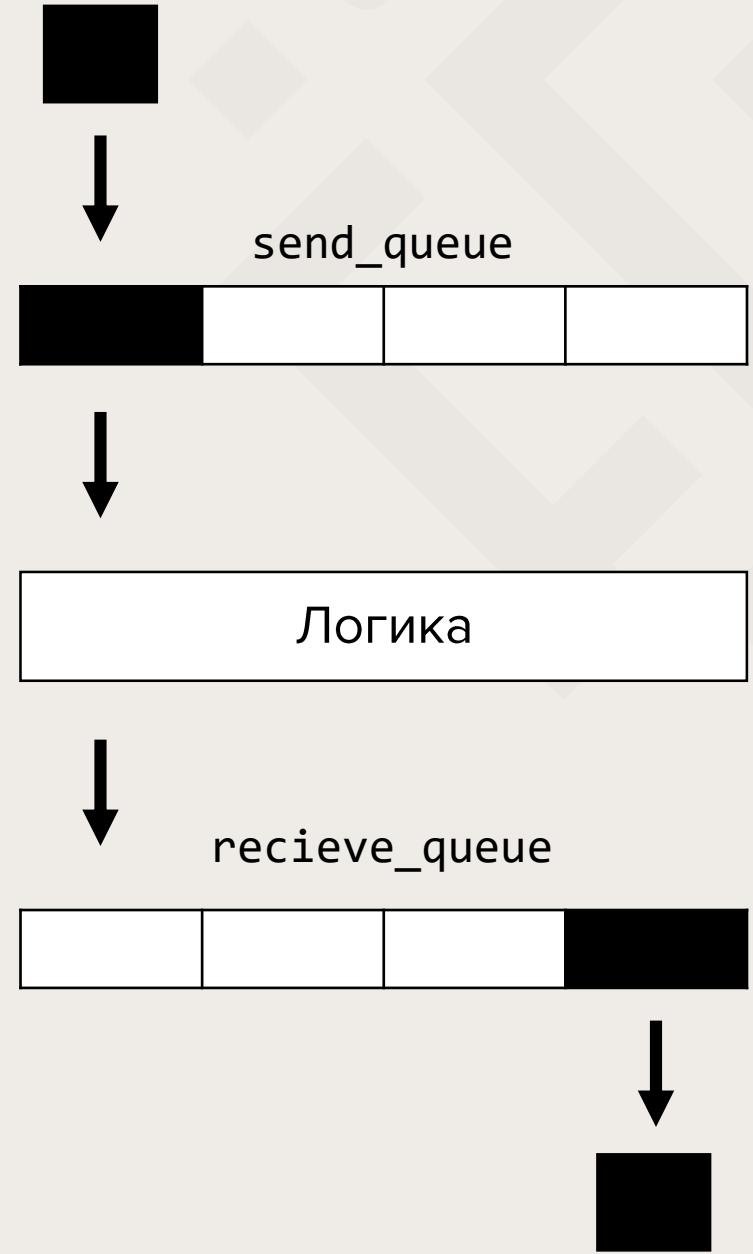


AXI4 VIP: axi_master.sv

```
...
do
begin
  @ (posedge clk);
  # 1 ;
end
while ( receive_queue.size () == 0
      || receive_queue [0] != tr);

  void' (receive_queue.pop_front ());
end
join_none // This will end simultaneously

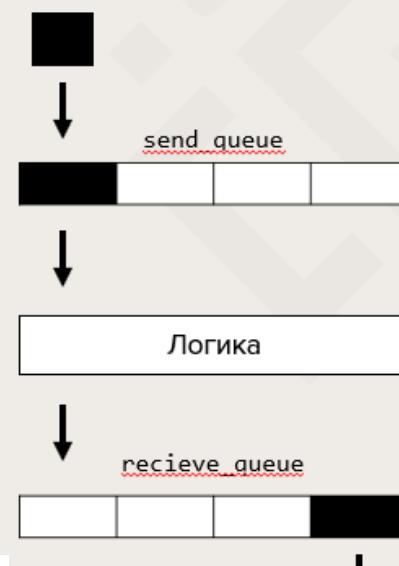
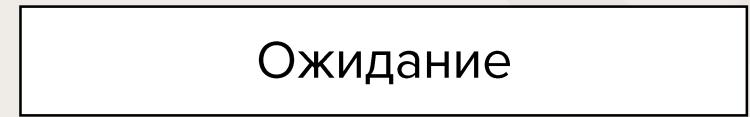
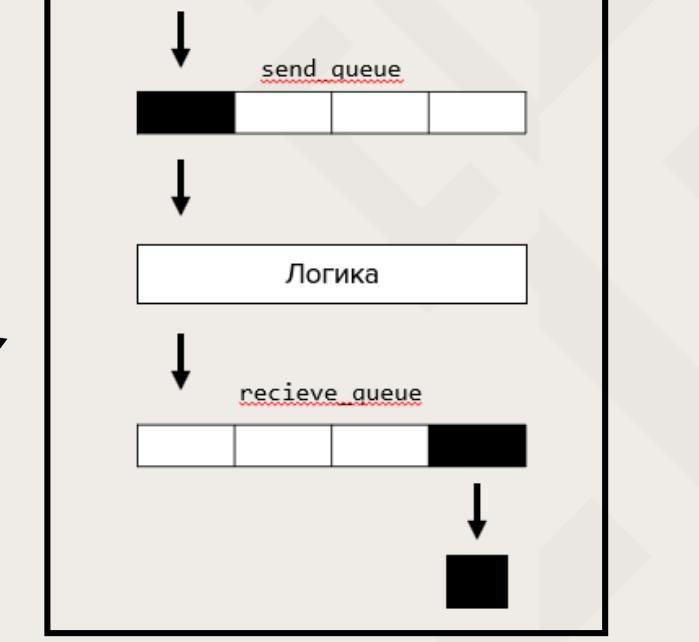
endtask
...
```



Основная часть

AXI4 VIP: axi_master.sv

```
...
task automatic run_write
(
...
);
start_write
(
...
);
`complete_everything
endtask
...
```



AXI4 VIP: axi_master.sv

```
...
//-----
// Main processing

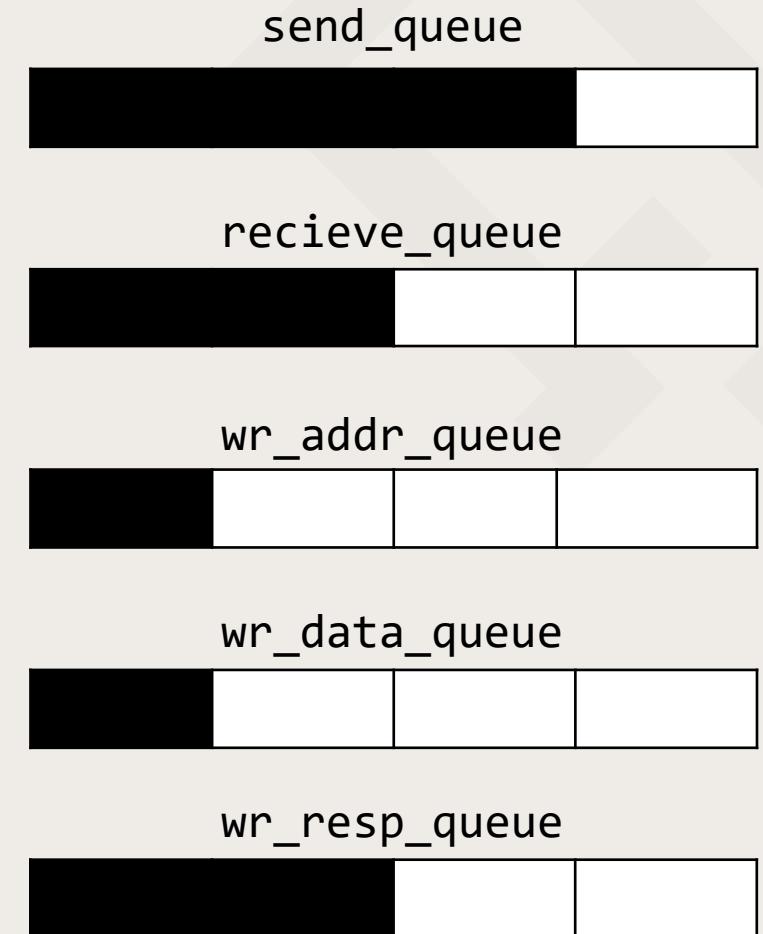
int unsigned cycle;
axi_transaction tr;

always @ (posedge clk)
  if (rst)
    begin
      cycle = 0; // Blocking assignment here ...
      ...
    end
  end

//-----
// Clearing the queues
...
...
```

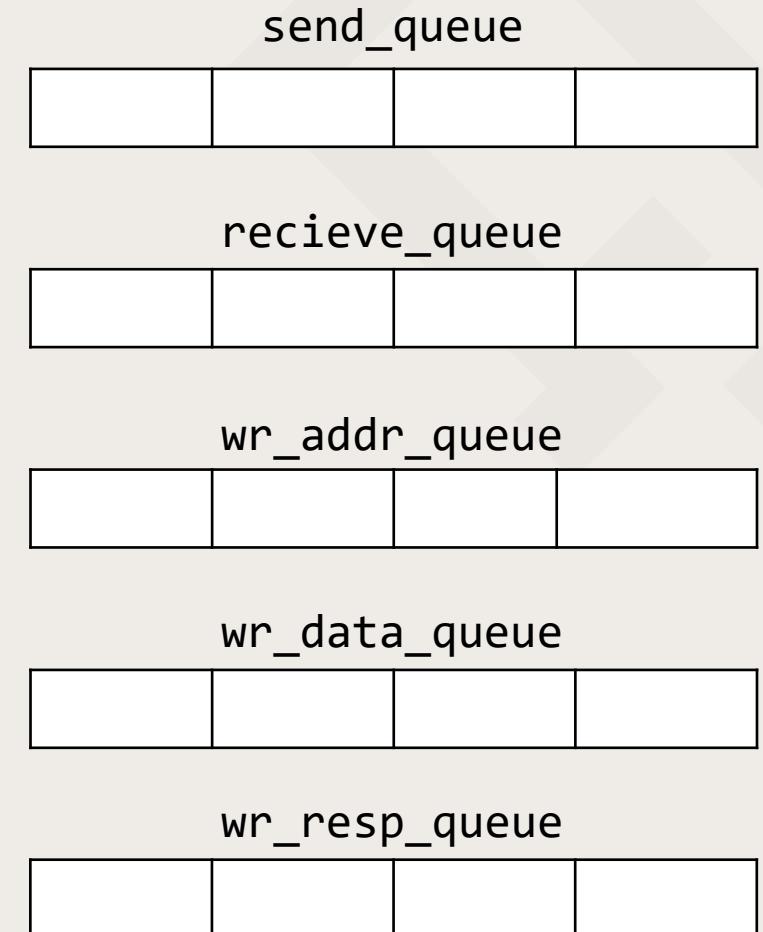
AXI4 VIP: axi_master.sv

```
...
    send_queue .delete ();
    receive_queue .delete ();
...
wr_resp_queue .delete ();
...
arvalid <= '0;
awvalid <= '0;
wvalid  <= '0;
end
...
```



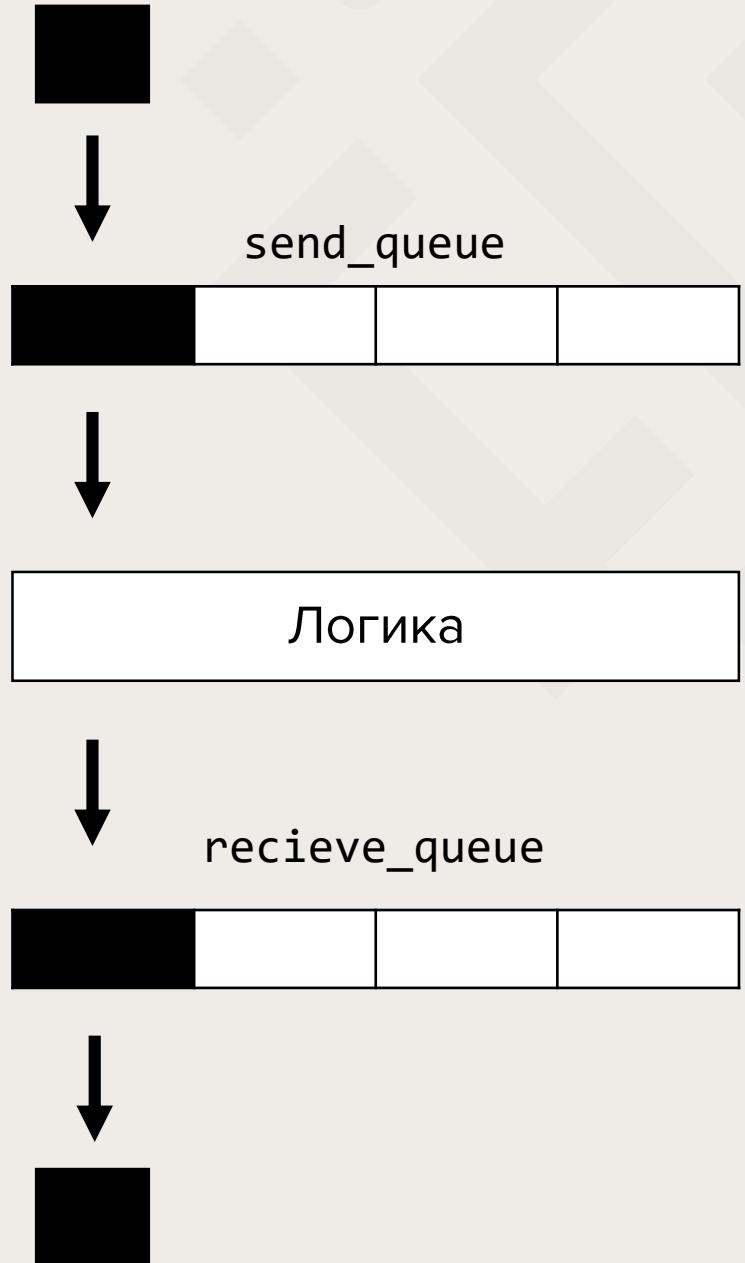
AXI4 VIP: axi_master.sv

```
...
    send_queue .delete ();
    receive_queue .delete ();
...
wr_resp_queue .delete ();
...
arvalid <= '0;
awvalid <= '0;
wvalid  <= '0;
end
...
```



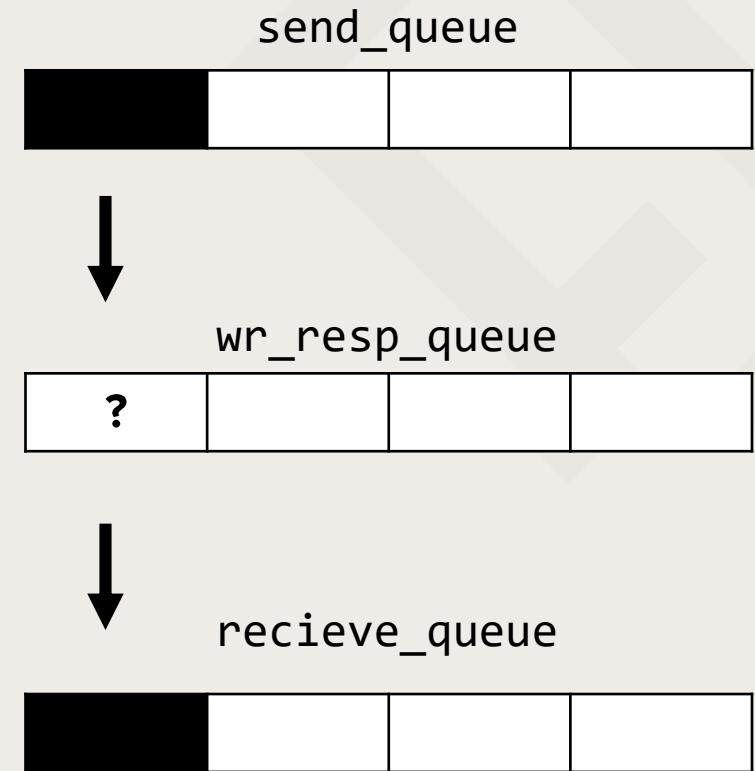
AXI4 VIP: axi_master.sv

```
...
else
begin
...
if (wr_resp_queue.size () > 0)
begin
  if (wr_resp_queue[0].addr_is_sent &&
      wr_resp_queue[0].data_is_sent)
    receive_queue.push_back(
      wr_resp_queue.pop_front ())
);
end
...
...
```



AXI4 VIP: axi_master.sv

```
...
else
begin
  ...
  if (wr_resp_queue.size () > 0)
begin
  if (wr_resp_queue[0].addr_is_sent &&
      wr_resp_queue[0].data_is_sent)
    receive_queue.push_back(
      wr_resp_queue.pop_front ())
  );
end
...
...
```



AXI4 VIP: axi_master.sv

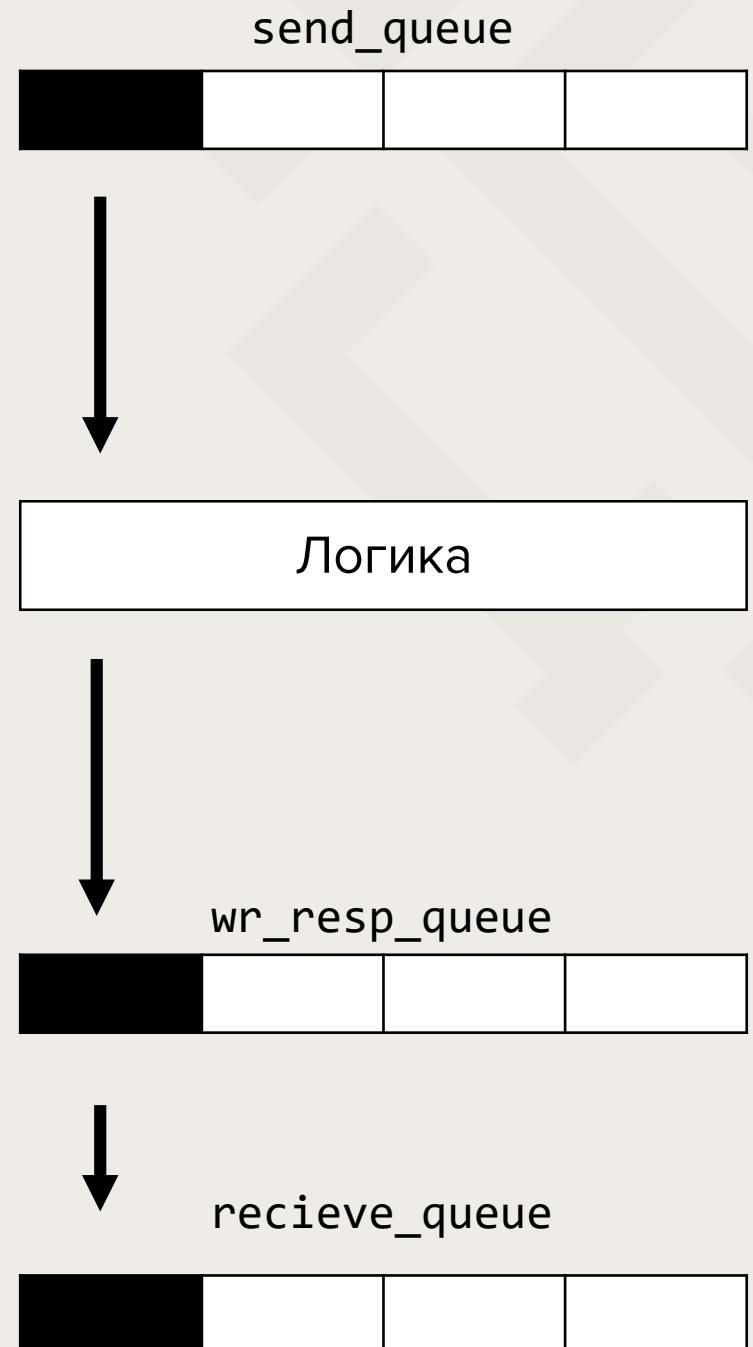
```
...
    if (awvalid & awready)
begin
    assert (wr_addr_queue.size () > 0);

    tr = wr_addr_queue.pop_front ();
    tr.addr_is_sent = 1;

    if (~ tr.data_is_sent)
        wr_resp_queue.push_back (tr);

    $display ("%0d master: write address ..."
end
...

```



AXI4 VIP: axi_master.sv

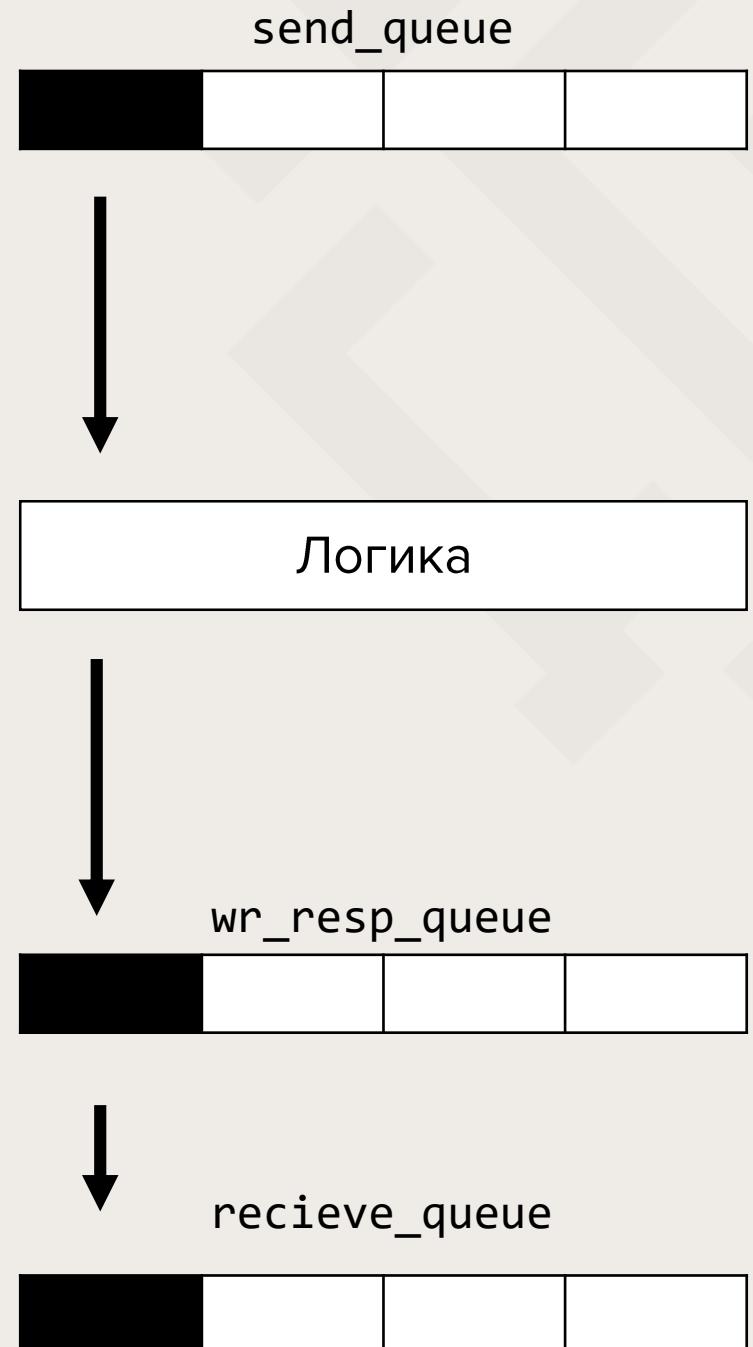
```
...
if (wvalid & wready)
begin
    assert (wr_data_queue.size () > 0);

    tr = wr_data_queue.pop_front ();
    tr.data_is_sent = 1;

    if (~ tr.addr_is_sent)
        wr_resp_queue.push_back (tr);

    $display ("%0d master: write data ..."
end
...

```



AXI4 VIP: axi_master.sv

```

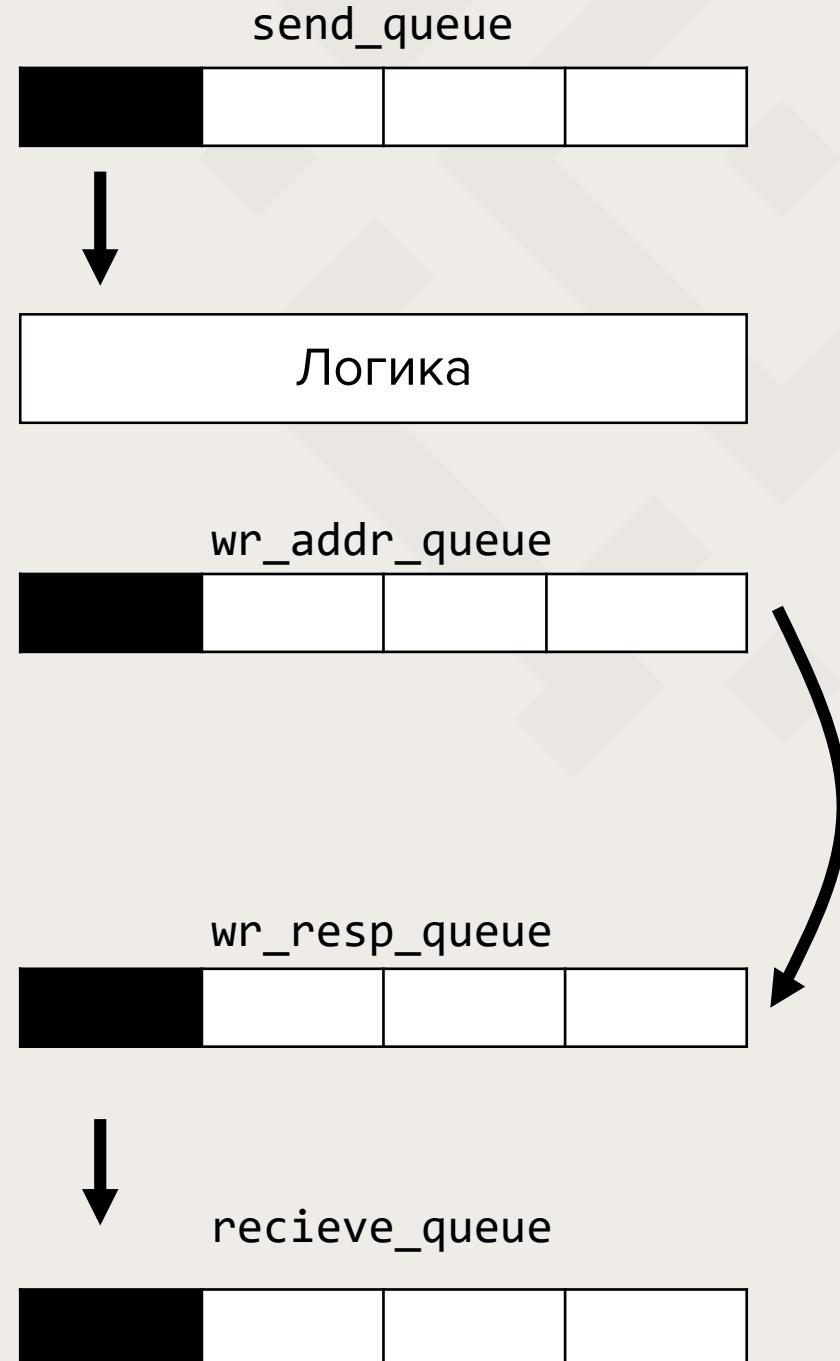
...
    if (awvalid & awready)
begin
    assert (wr_addr_queue.size () > 0);

    tr = wr_addr_queue.pop_front ();
    tr.addr_is_sent = 1;

    if (~ tr.data_is_sent)
        wr_resp_queue.push_back (tr);

    $display ("%0d master: write address ..."
end
...

```



Основная часть

AXI4 VIP: axi_master.sv

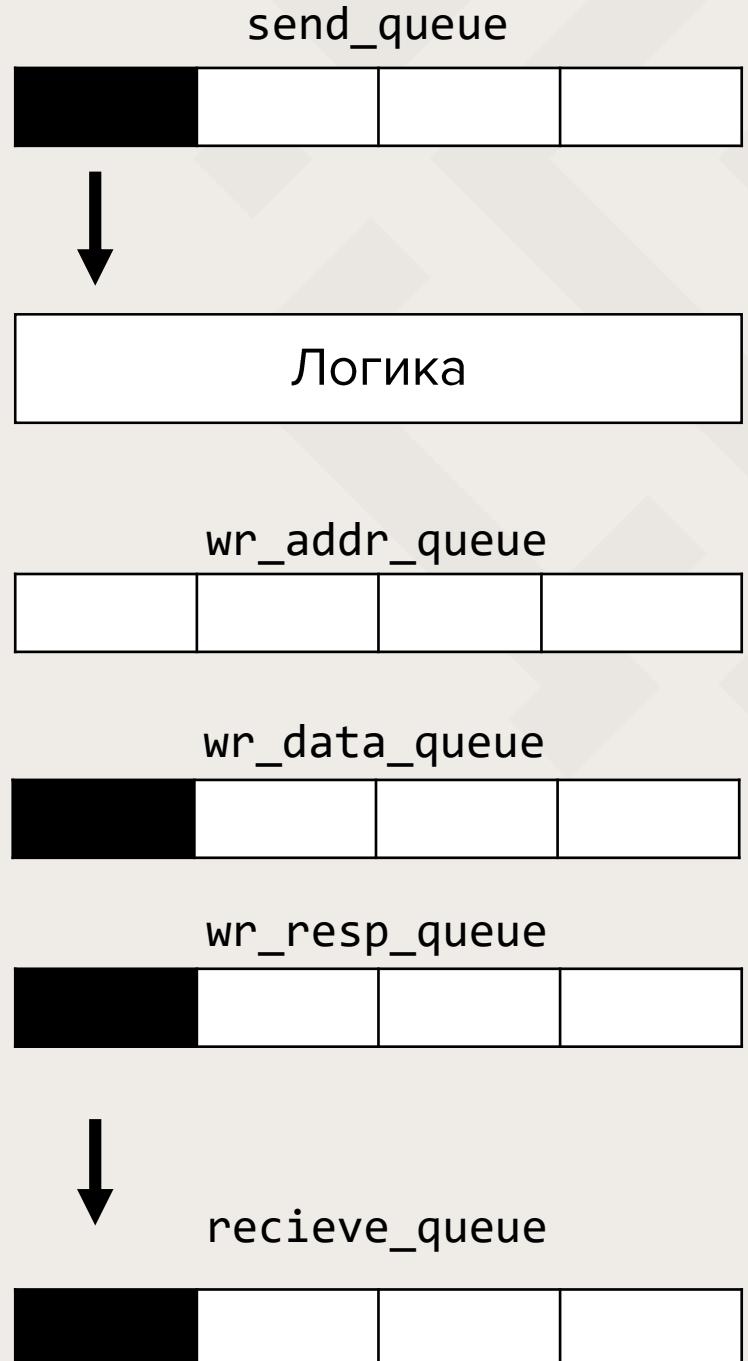
```
...
    if (wvalid & wready)
begin
    assert (wr_data_queue.size () > 0);

    tr = wr_data_queue.pop_front ();
    tr.data_is_sent = 1;

    if (~ tr.addr_is_sent)
        wr_resp_queue.push_back (tr);

    $display ("%0d master: write data ..."
end

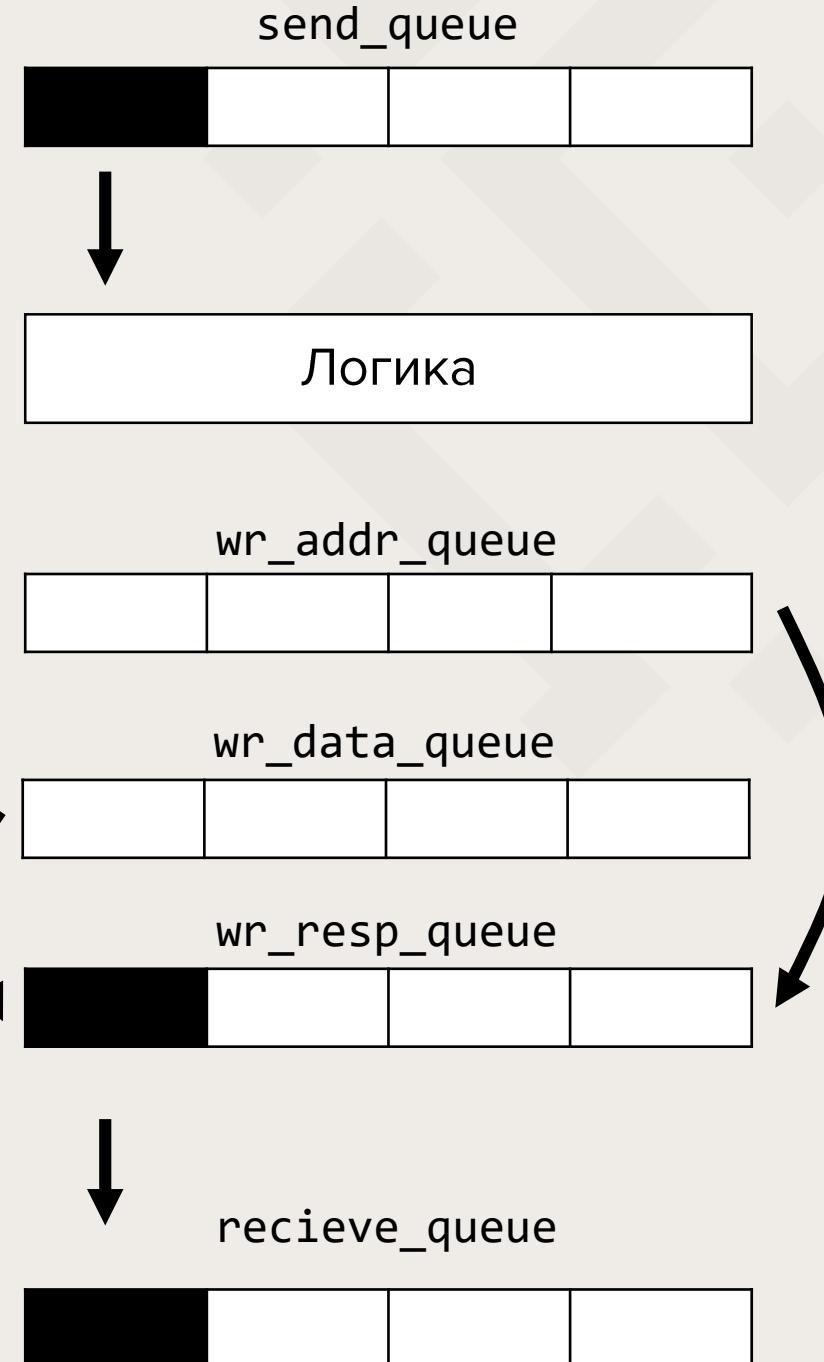
...
```



Основная часть

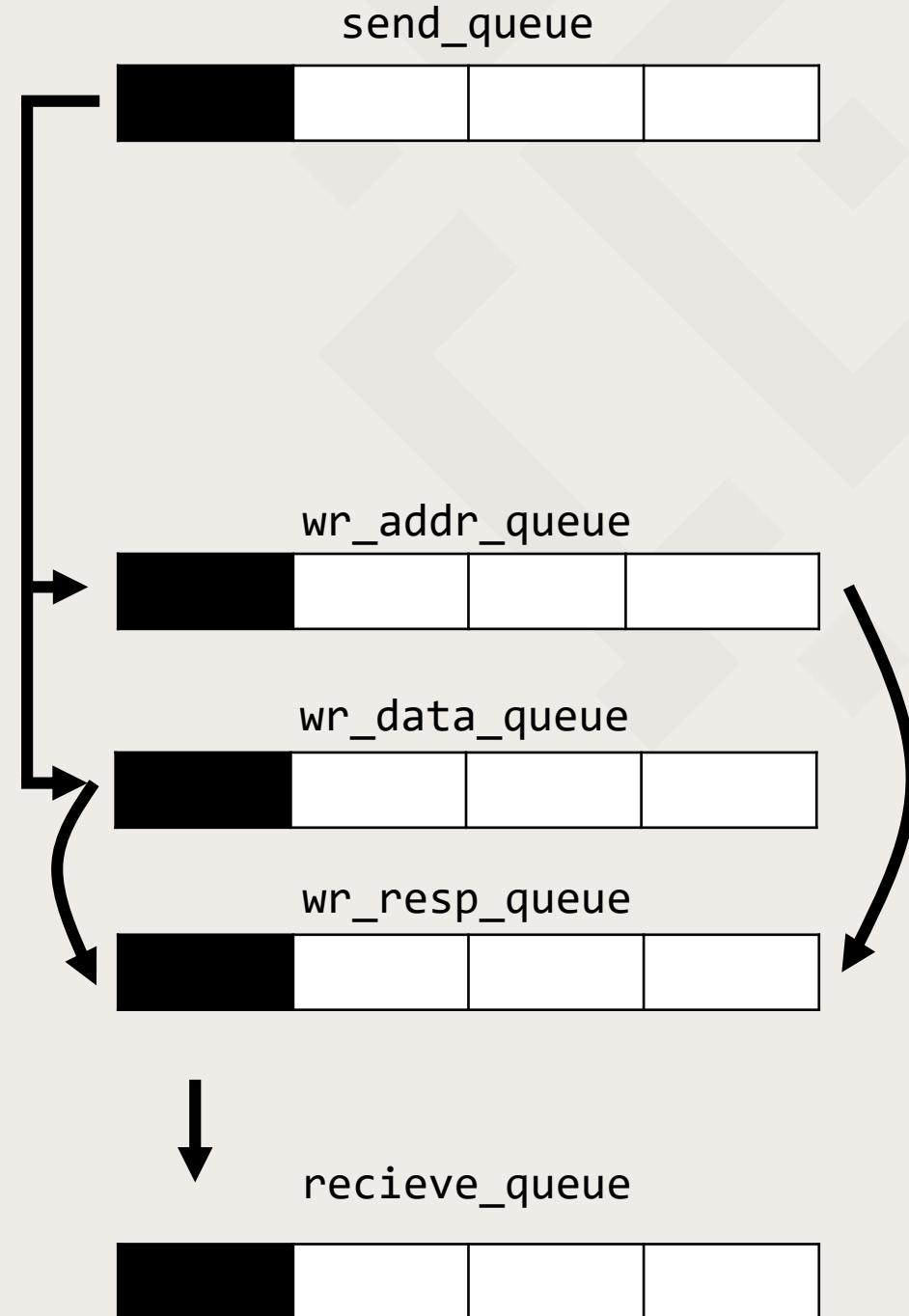
AXI4 VIP: axi_master.sv

```
...
    while (send_queue.size () > 0)
begin
    tr = send_queue.pop_front ();
    wr_addr_queue.push_back (tr);
    wr_data_queue.push_back (tr);
end
...
...
```



AXI4 VIP: axi_master.sv

```
...
  while (send_queue.size () > 0)
begin
    tr = send_queue.pop_front ();
    wr_addr_queue.push_back (tr);
    wr_data_queue.push_back (tr);
end
...
...
```



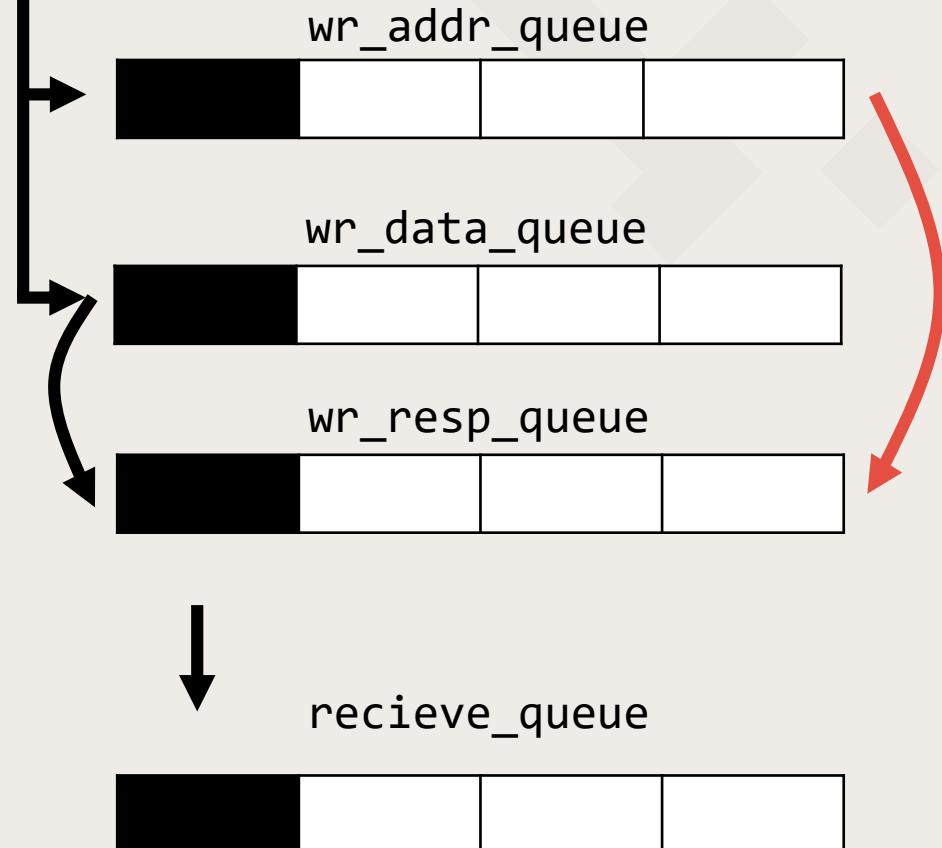
AXI4 VIP: axi_master.sv

```

...
  if (wr_addr_queue.size () > 0) begin
    tr = wr_addr_queue [0];
    if (tr.addr_delay > 0) begin
      tr.addr_delay --;
    end
    else begin
      awvalid <= '1;
      awaddr  <= tr.addr;
      ...
    end
  end
...

```

Зависимость от задержки



AXI4 VIP: axi_master.sv

```

...
    if (awvalid & awready)
begin
    assert (wr_addr_queue.size () > 0);

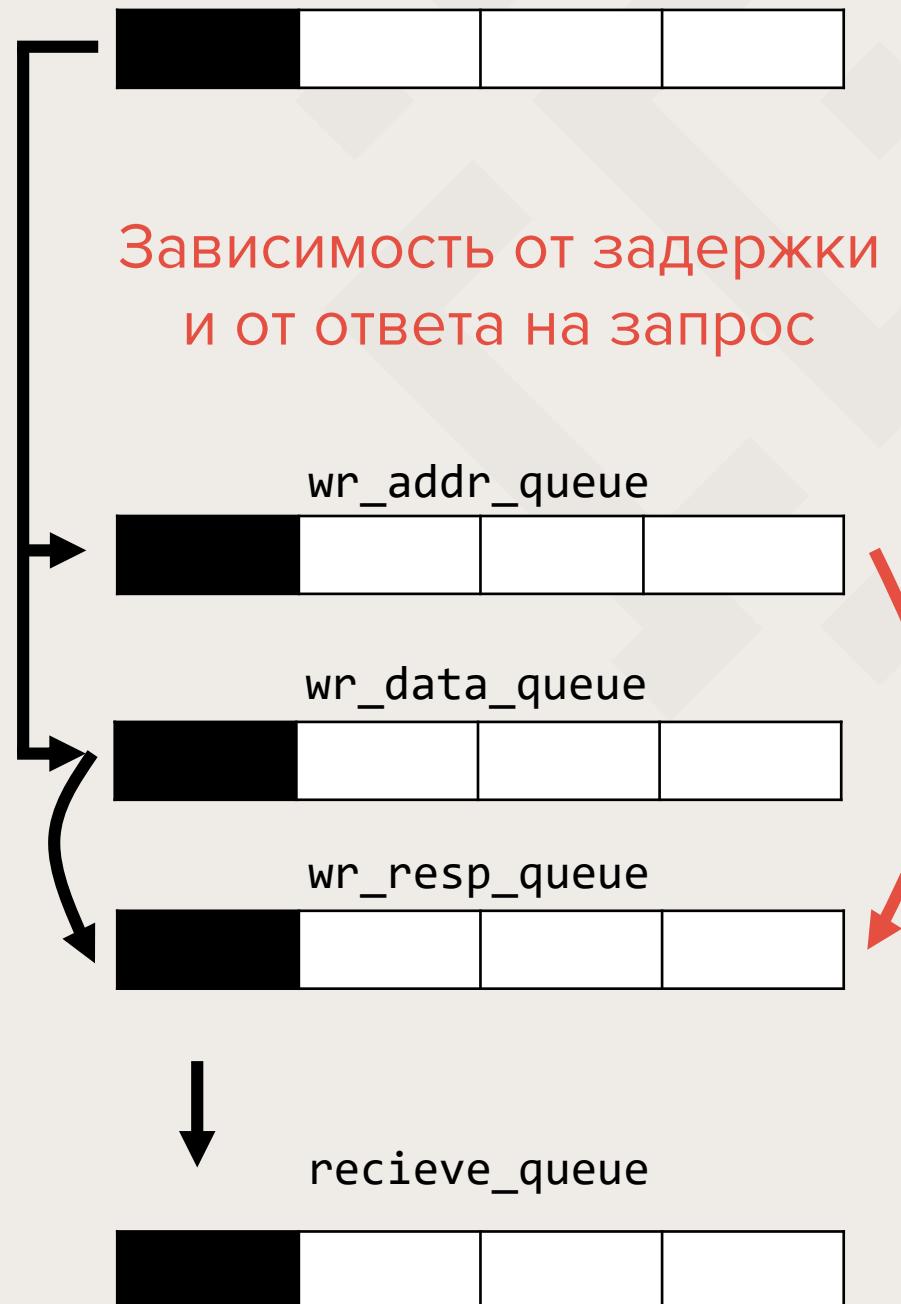
    tr = wr_addr_queue.pop_front ();
    tr.addr_is_sent = 1;

    if (~ tr.data_is_sent)
        wr_resp_queue.push_back (tr);

    $display ("%0d master: write address ..."
end
...

```

Зависимость от задержки
и от ответа на запрос



AXI4 VIP: axi_master.sv

```
...
if (wr_data_queue.size () > 0) begin
    tr = wr_data_queue [0];
    ...
    if (tr.data_delay > 0) begin
        tr.data_delay --;
    end
    else begin
        wvalid <= '1;
        wdata  <= tr.data;
        ...
    end
end
...
```



AXI4 VIP: axi_master.sv

```

...
    if (wvalid & wready)
begin
    assert (wr_data_queue.size () > 0);

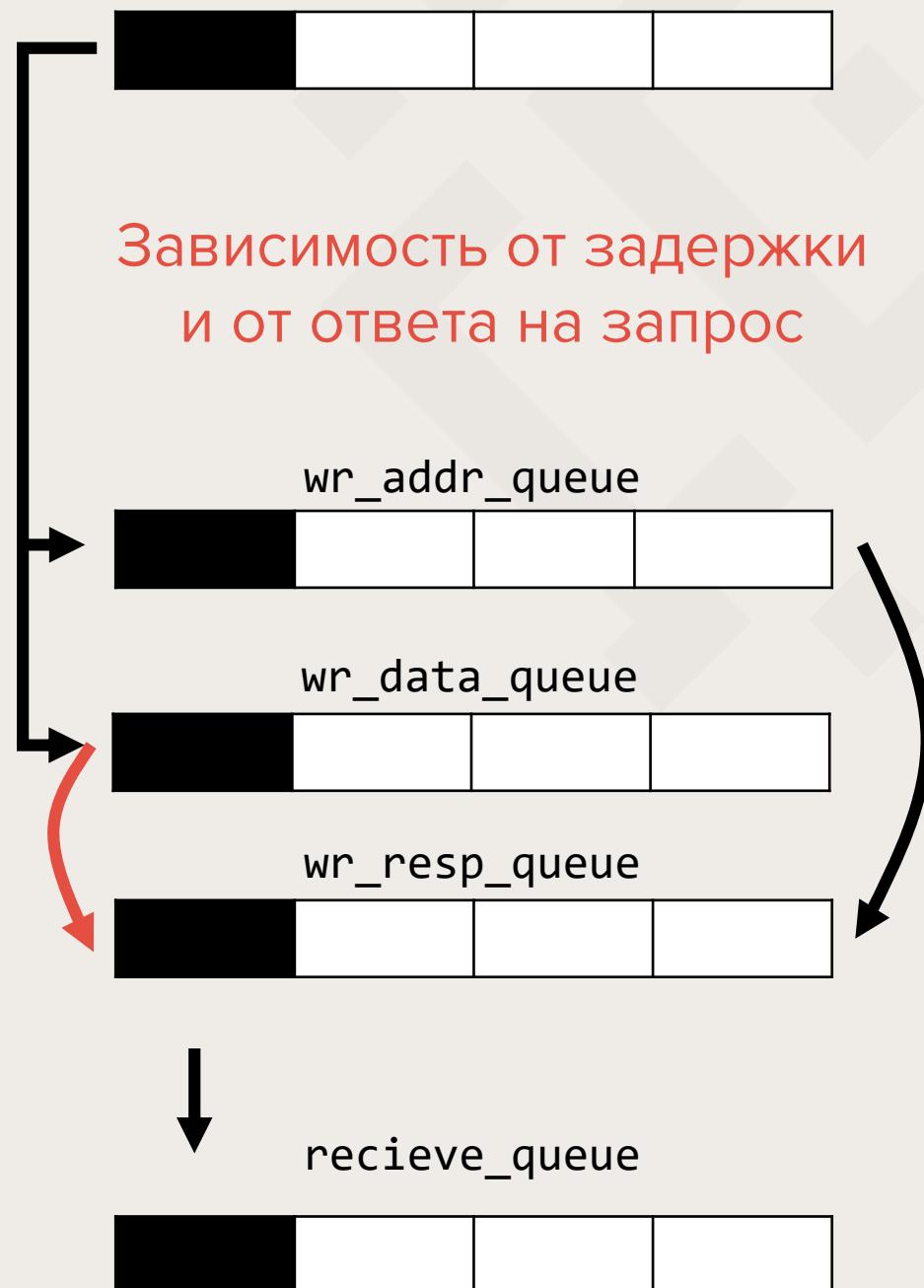
    tr = wr_data_queue.pop_front ();
    tr.data_is_sent = 1;

    if (~ tr.addr_is_sent)
        wr_resp_queue.push_back (tr);

    $display ("%0d master: write data ..."
end
...

```

Зависимость от задержки
и от ответа на запрос

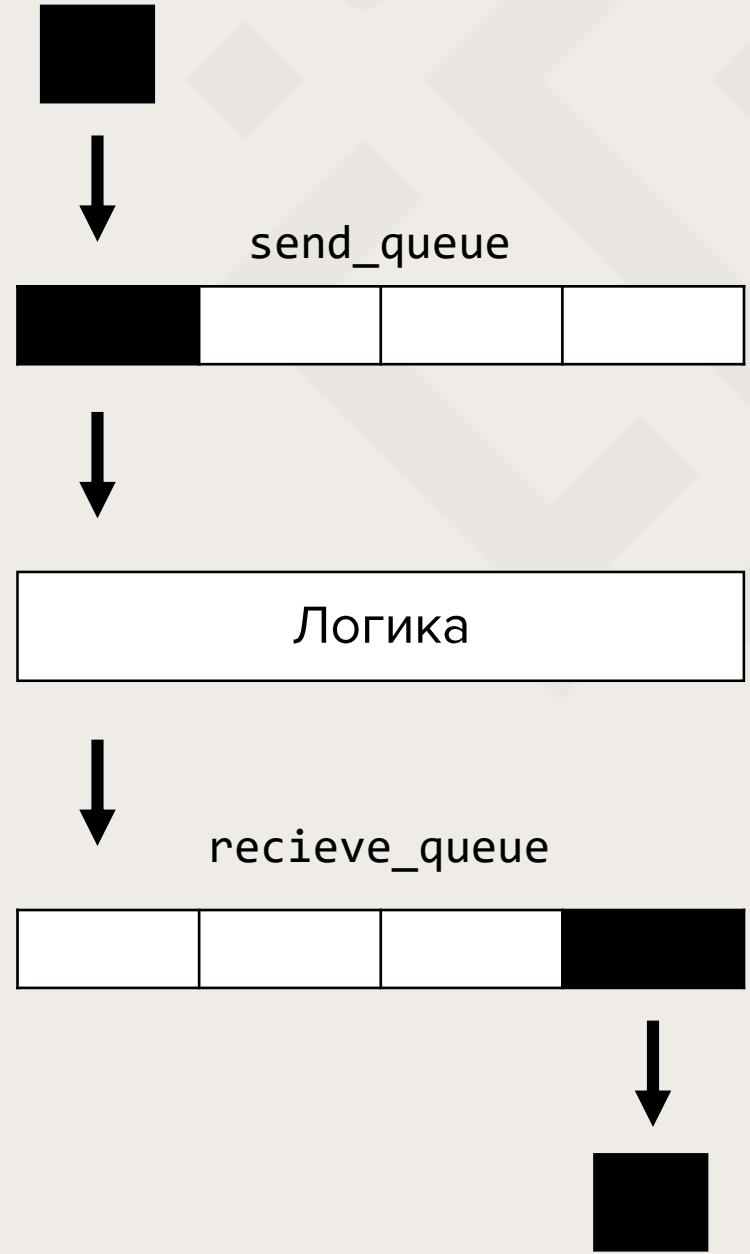


AXI4 VIP: axi_master.sv

```
...
do
begin
    @ (posedge clk);
    # 1 ;
end
while ( receive_queue.size () == 0
        || receive_queue [0] != tr);

    void' (receive_queue.pop_front ());
end
join_none // This will end simultaneously

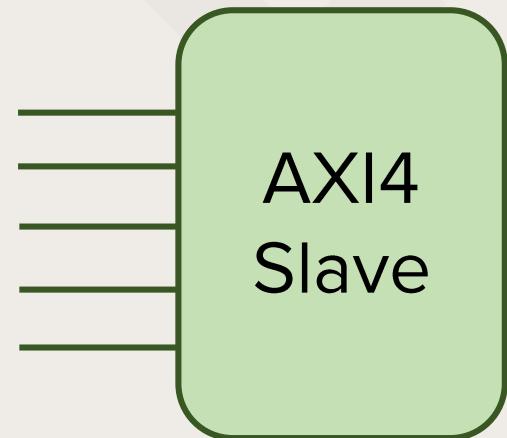
endtask
...
```



AXI4 VIP: axi_slave.sv

```
module axi_slave
(
    input          clk,
    input          rst,
    ...
    output id_t    rid,
    output logic   rvalid,
    input          rready,
    output logic   bvalid,
    input          bready
);
...

```



AXI4 VIP: axi_slave.sv

```
...
import axi_transaction::*;

//-----
// Ready signal and response cycle randomization

logic [6:0] write_address_ready_probability,
            write_data_ready_probability;

always @ (posedge clk)
begin
    awready <= ( $urandom_range (0, 99) < ...
    wready   <= ( $urandom_range (0, 99) < ...
end

...
...
```



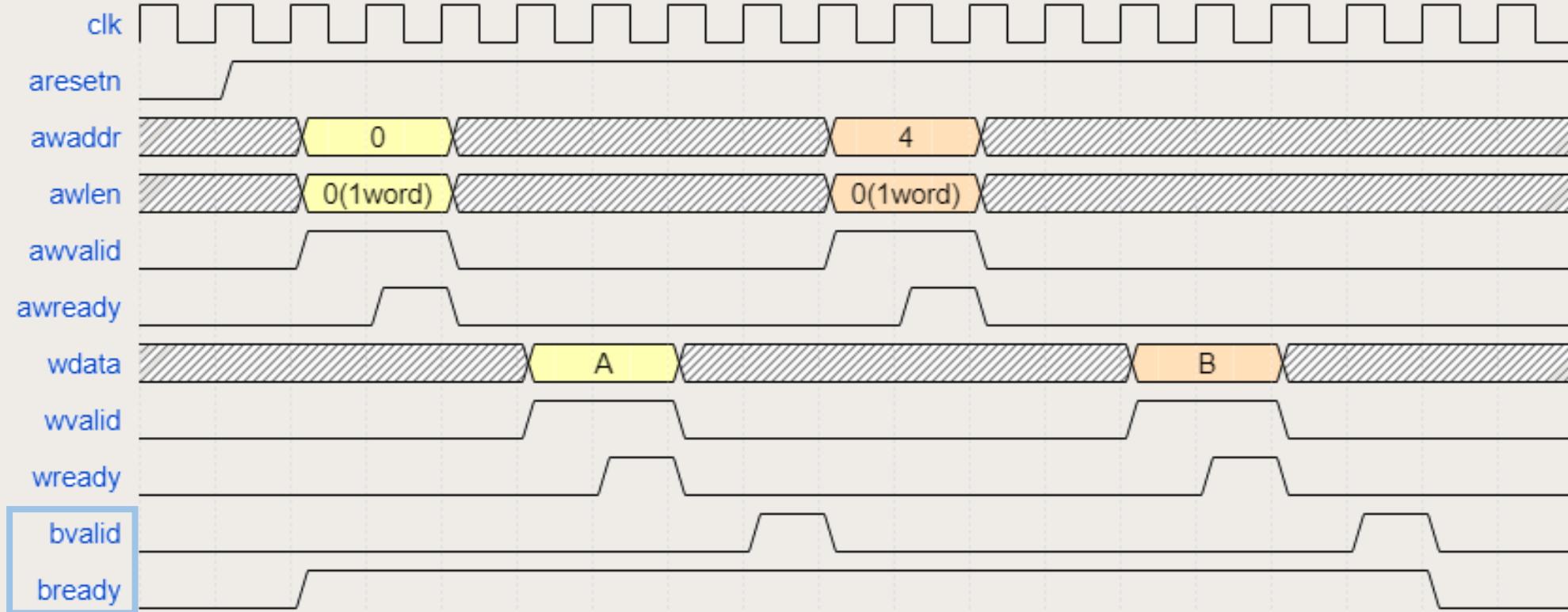
AXI4 VIP: axi_master.sv и axi_slave.sv

- **Практика.**
- 08_axi_master_slave_monitor/52_axi_master_no_resp/axi_testbench.sv

AXI4 VIP: axi_master.sv и axi_slave.sv

- А если добавить **ответ на запись?**

Канал ответа на запись



AXI4 VIP: axi_master.sv и axi_slave.sv

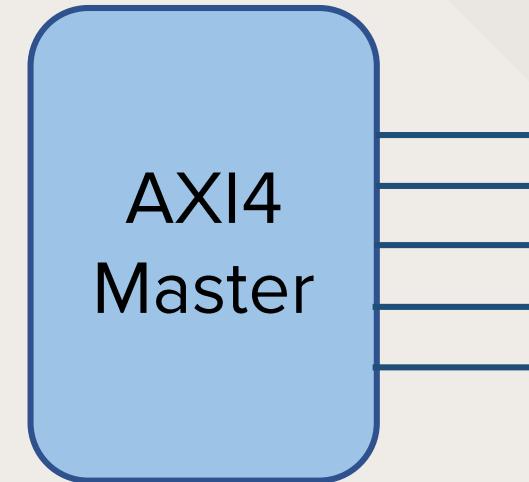
- 08_axi_master_slave_monitor/53_axi_master_resp

AXI4 VIP: axi_master.sv

```
module axi_master
(
    input          clk,
    input          rst,
    output addr_t araddr,
    output id_t   arid,
    output logic  arvalid,
    input          arready,
    ...
    output addr_t awaddr,
    output logic  bready
);

import axi_transaction::*;

...
```



AXI4 VIP: axi_master.sv

```
...
//-----
// Ready signal randomization

logic [6:0] write_response_ready_probability;

always @ (posedge clk) begin
    rready <= 0; // No read support ...
    bready <= ($urandom_range (0, 99) < ...
end

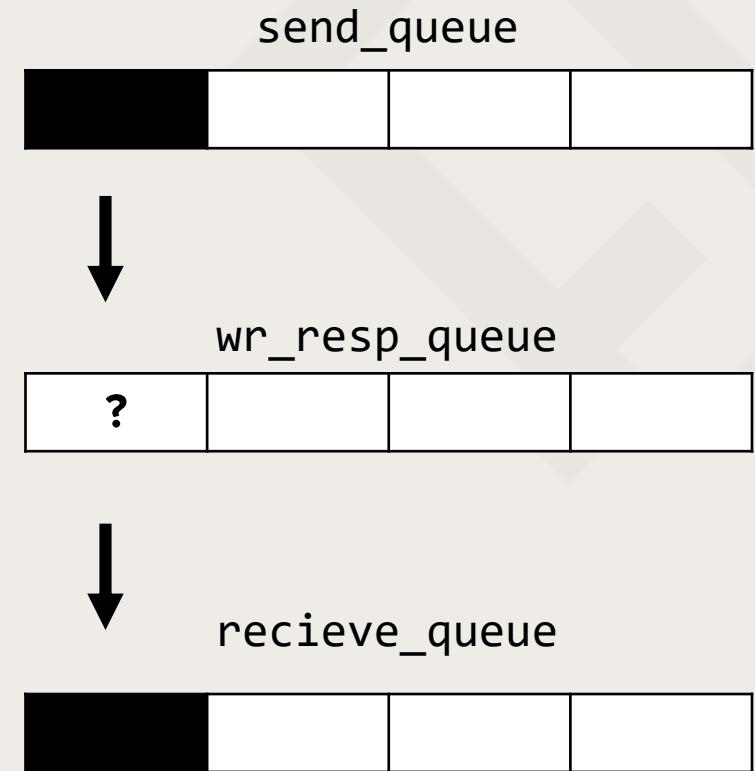
task reset_probabilities ();
    write_response_ready_probability = 100;
endtask

...
```



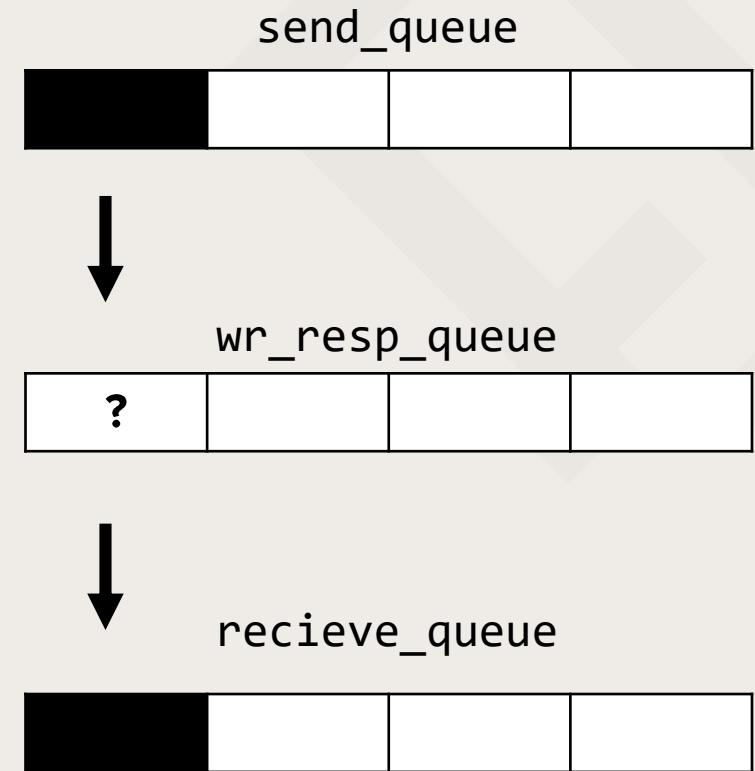
AXI4 VIP: axi_master.sv

```
...
else
begin
...
if (wr_resp_queue.size () > 0)
begin
  if (wr_resp_queue[0].addr_is_sent &&
      wr_resp_queue[0].data_is_sent)
    receive_queue.push_back(
      wr_resp_queue.pop_front ())
);
end
...
...
```



AXI4 VIP: axi_master.sv

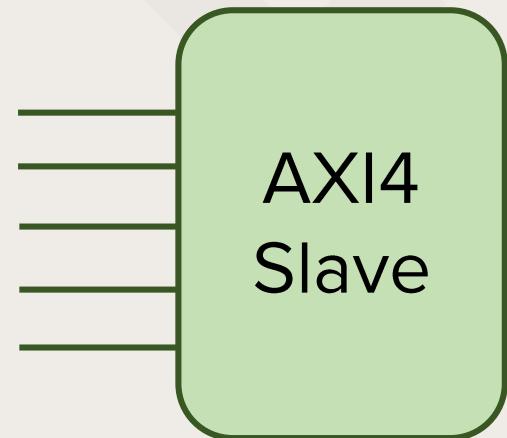
```
if (bvalid & bready) begin
    if (wr_resp_queue.size () == 0) begin
        $fatal ("Unexpected write response");
    end
    else begin
        tr = wr_resp_queue.pop_front ();
        ...
        if (~ tr.addr_is_sent)
            $fatal ("Unexpected write response ...");
        if (~ tr.data_is_sent)
            $fatal ("Unexpected write response ...");
        receive_queue.push_back (tr);
    ...
}
```



AXI4 VIP: axi_slave.sv

```
module axi_slave
(
    input          clk,
    input          rst,
    ...
    output id_t    rid,
    output logic   rvalid,
    input          rready,
    output logic   bvalid,
    input          bready
);
...

```



AXI4 VIP: axi_slave.sv

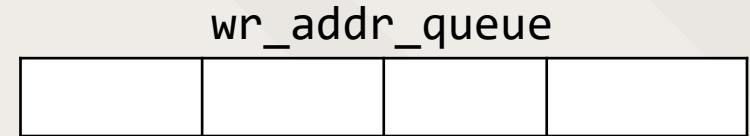
```
...
initial reset_probabilities();

//-----  
// Queues, counters and memories

addr_t wr_addr_queue [$];
data_t wr_data_queue [$];

int unsigned wr_resp_counter;

...
```



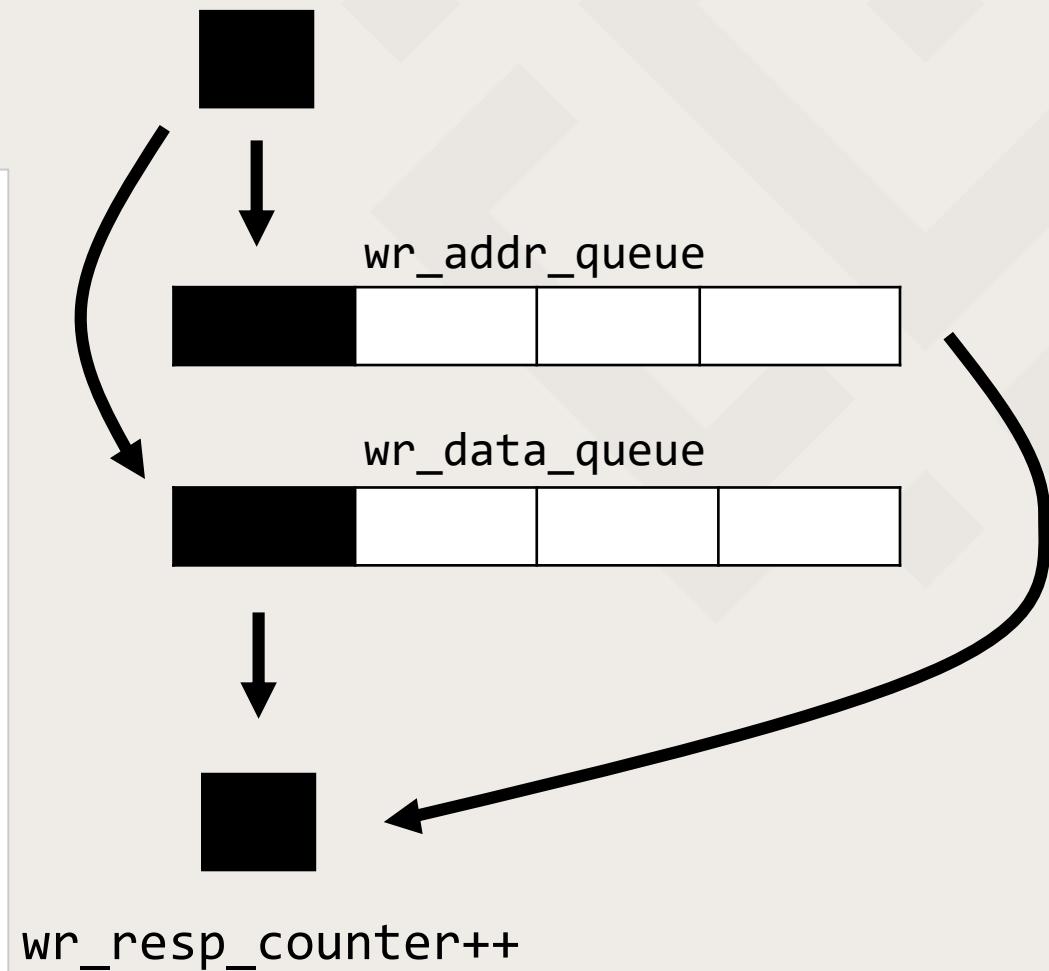
AXI4 VIP: axi_slave.sv

```
...
if (awvalid & awready)
    wr_addr_queue.push_back (awaddr);

if (wvalid & wready)
    wr_data_queue.push_back (wdata);

if ( wr_addr_queue.size () > 0
    & wr_data_queue.size () > 0)
begin
    void'(wr_addr_queue.pop_front ());
    void'(wr_data_queue.pop_front ());
    wr_resp_counter++;
end
...

```



AXI4 VIP: axi_slave.sv

```
...
if (~ bvalid | bready)
begin
    bvalid <= '0;

    if ( wr_resp_counter > 0
        && $urandom_range (0, 99) < ...
begin
    bvalid <= '1;
    wr_resp_counter--;
end
end

...

```



AXI4 VIP: axi_master.sv и axi_slave.sv

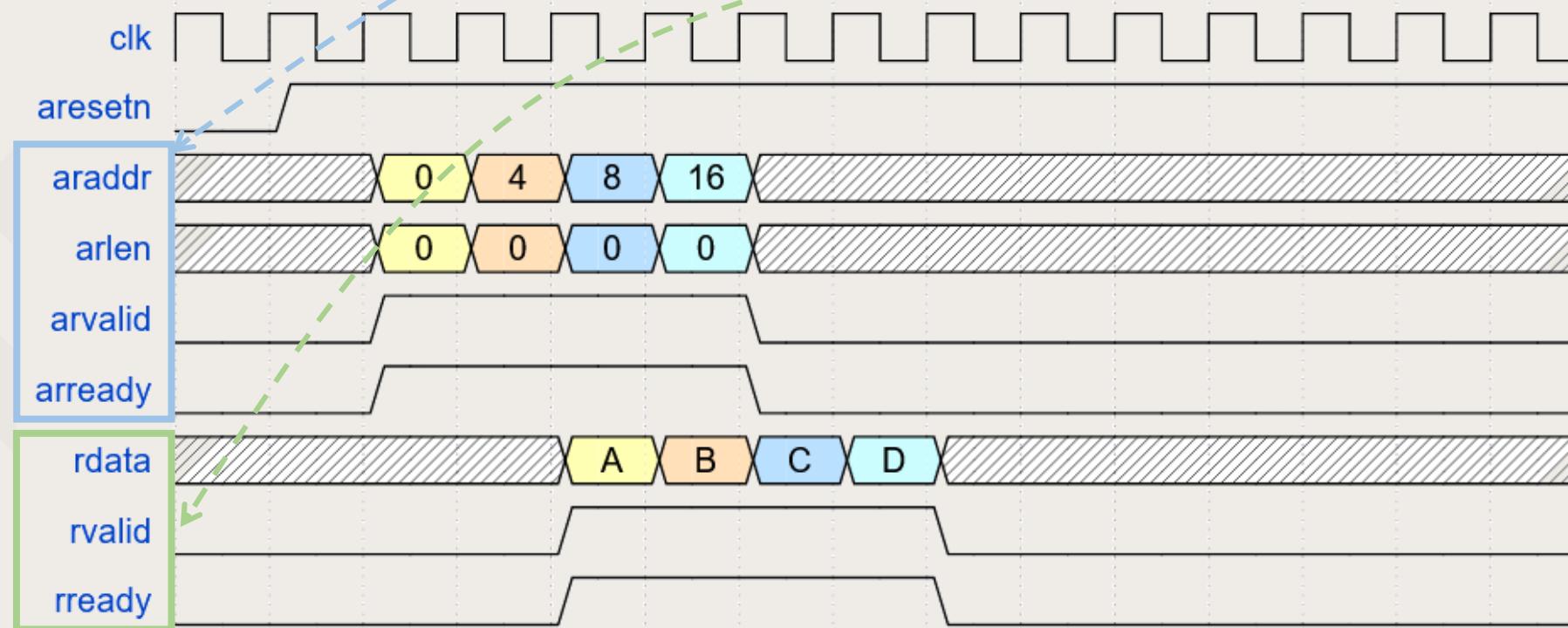
- **Практика.**
- 08_axi_master_slave_monitor/53_axi_master_resp/axi_testbench.sv

AXI4 VIP: axi_master.sv и axi_slave.sv

- А если добавить **чтение?**

- Канал адреса чтения

- Канал данных чтения



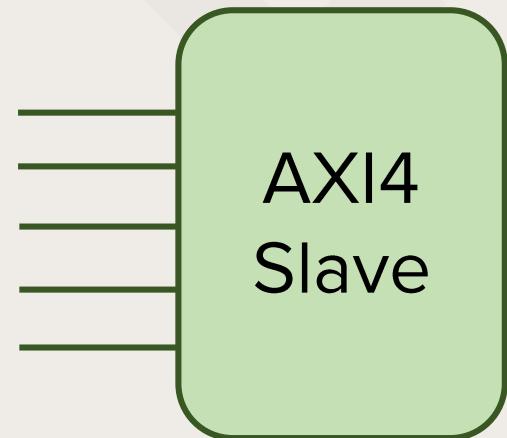
AXI4 VIP: axi_master.sv и axi_slave.sv

- 08_axi_master_slave_monitor/53_axi_master_resp/50_axi_pipelined_wr_out_of_order_rd

AXI4 VIP: axi_slave.sv

```
module axi_slave
(
    input          clk,
    input          rst,
    ...
    output id_t    rid,
    output logic   rvalid,
    input          rready,
    output logic   bvalid,
    input          bready
);
...

```



AXI4 VIP: axi_slave.sv

```
...
initial reset_probabilities();

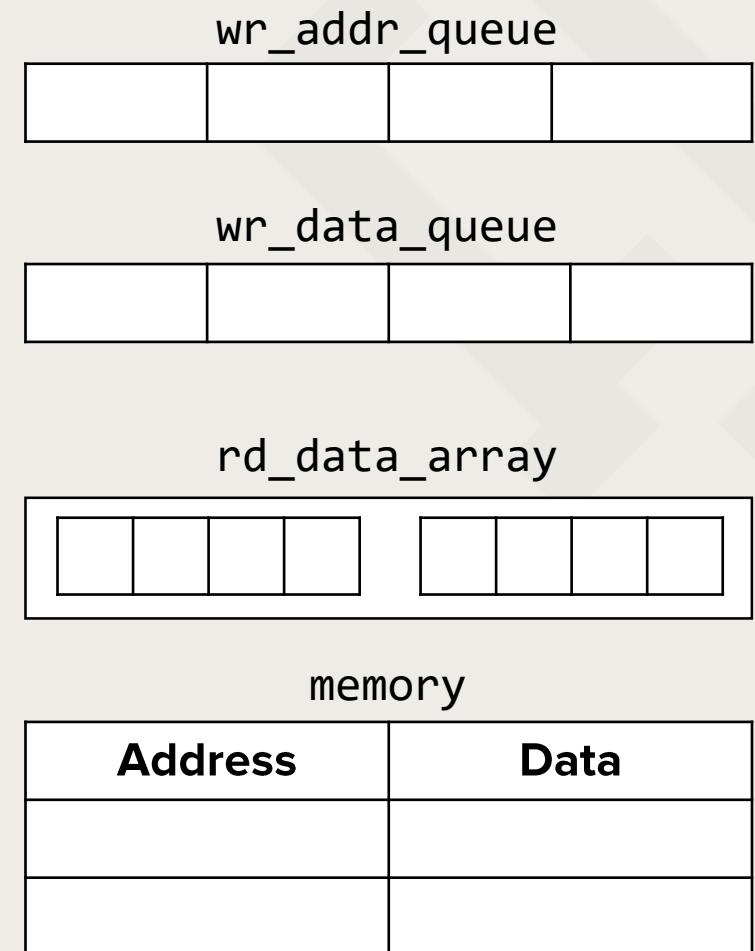
//-----  
// Queues, counters and memories

addr_t wr_addr_queue[$];
data_t wr_data_queue[$];
data_t rd_data_array[n_ids][$];

int unsigned wr_resp_counter;

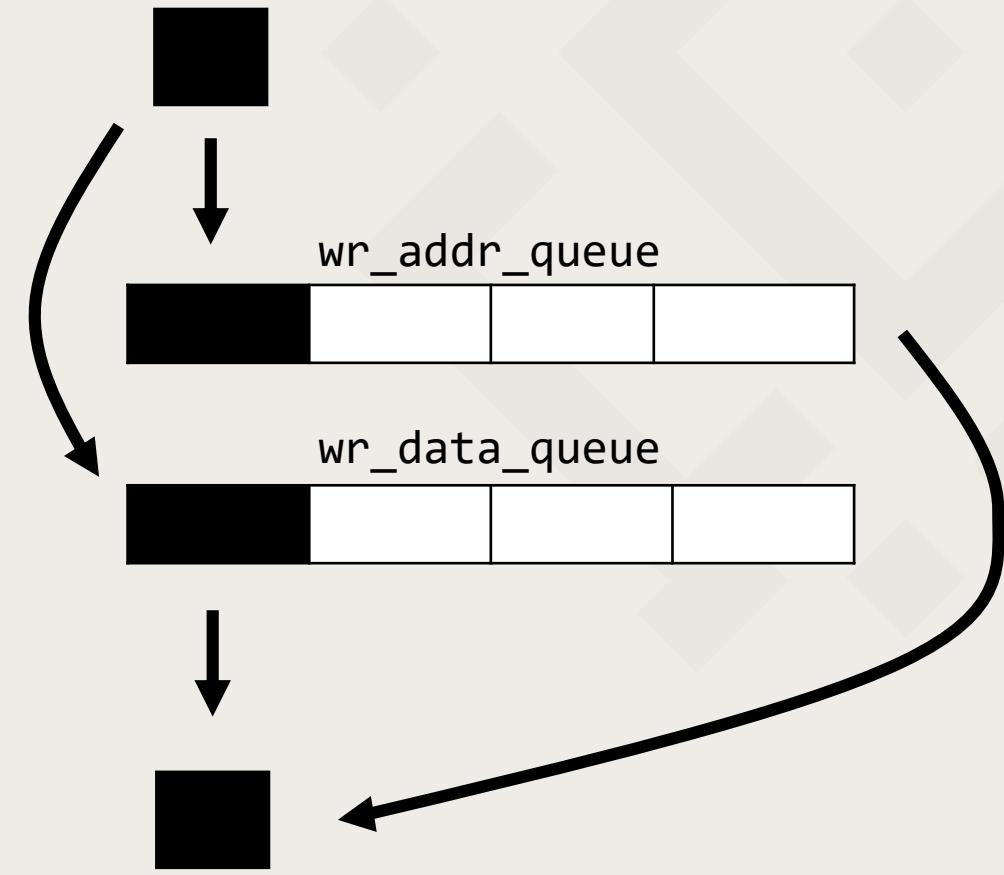
data_t memory [addr_t]; // A sparse array

...
```



AXI4 VIP: axi_slave.sv

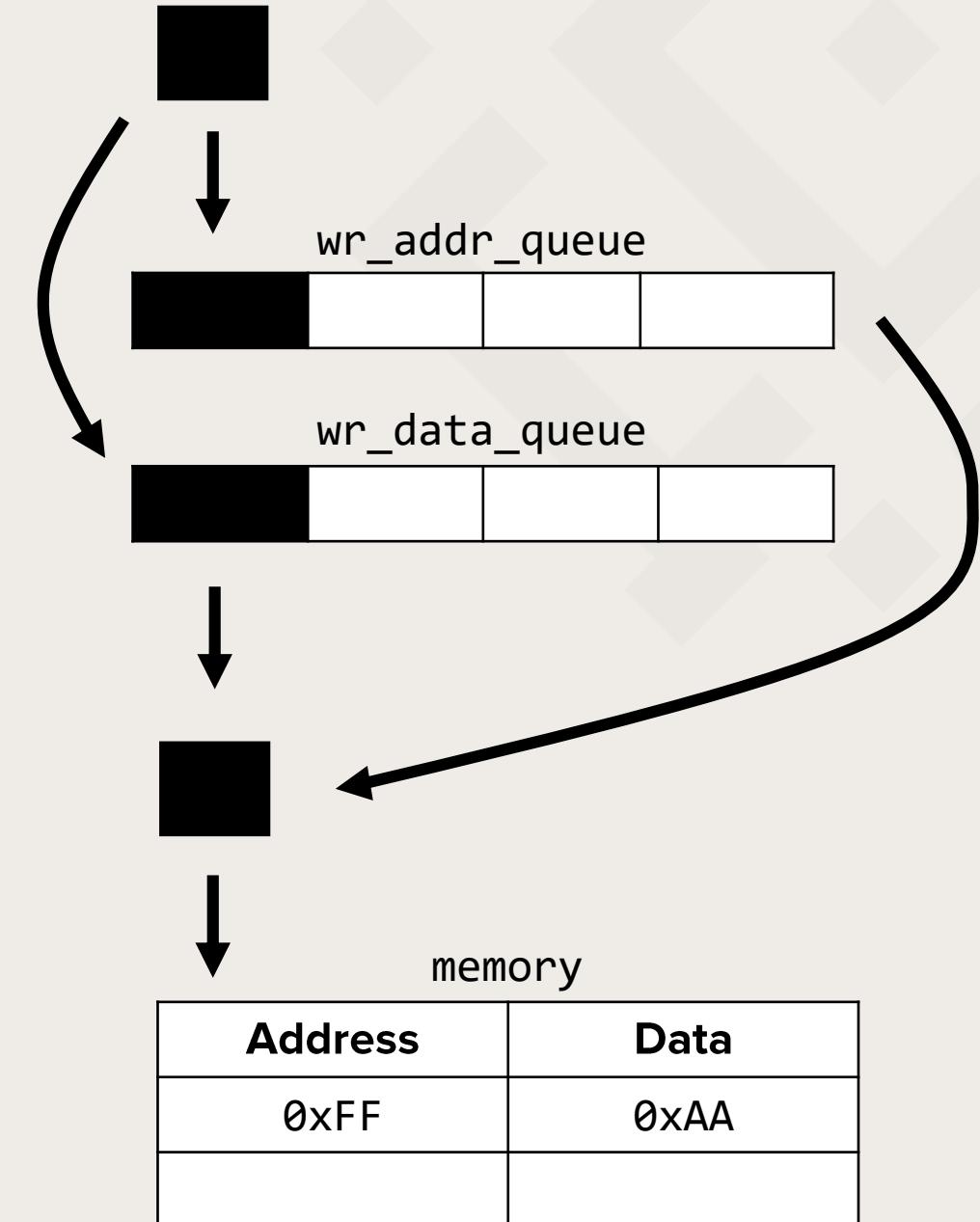
```
...
...
if ( wr_addr_queue.size () > 0
    & wr_data_queue.size () > 0)
begin
    void'(wr_addr_queue.pop_front ());
    void'(wr_data_queue.pop_front ());
    wr_resp_counter++;
end
...
...
```



AXI4 VIP: axi_slave.sv

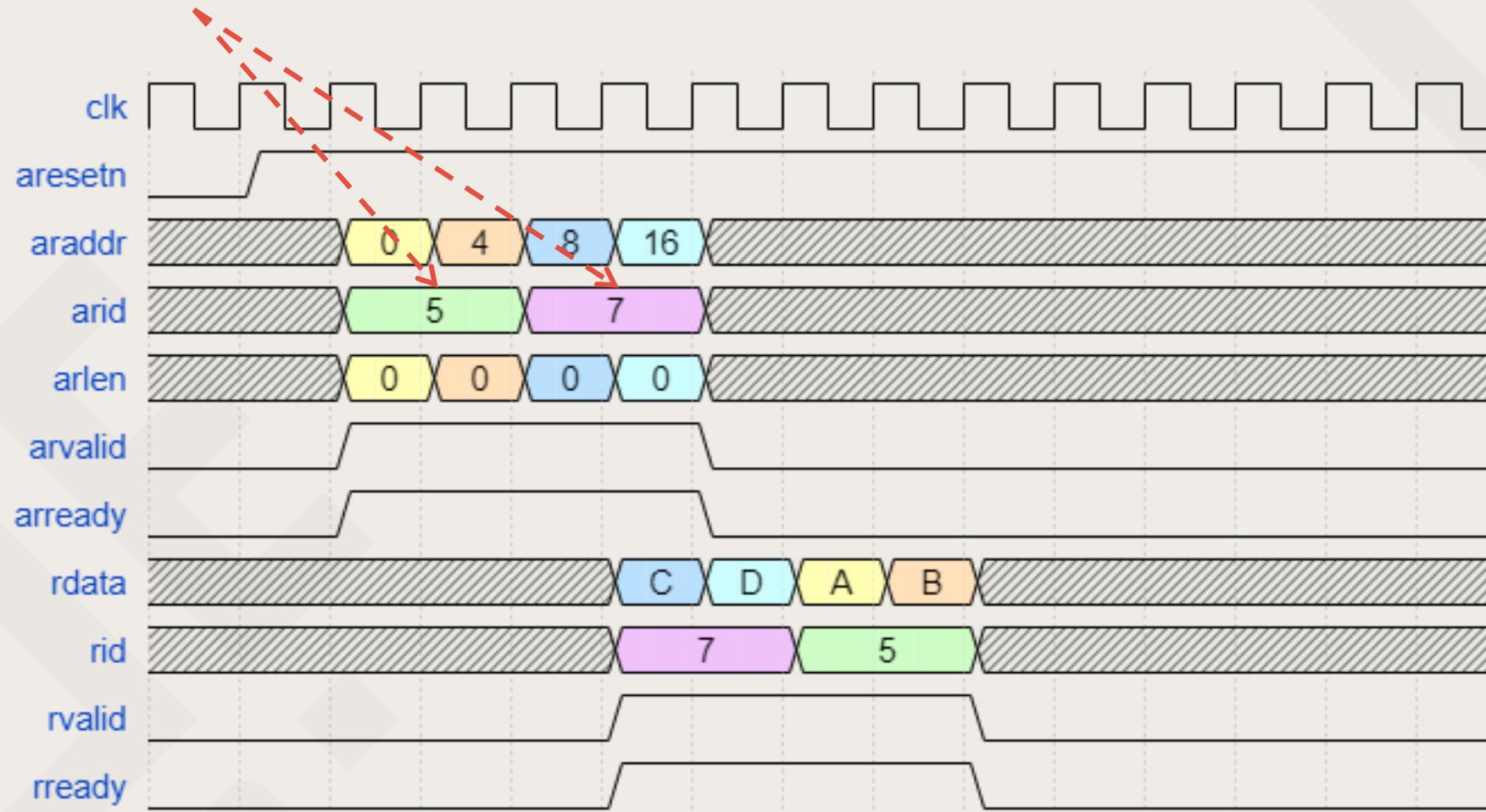
```
...
...
if ( wr_addr_queue.size () > 0
    & wr_data_queue.size () > 0)
begin
    $display ("slave: write memory [%h]" ...
memory [wr_addr_queue.pop_front ()]
        = wr_data_queue.pop_front ();

    wr_resp_counter++;
end
...
```



AXI4: Конвейерные транзакции: чтение out-of-order

- Для различных ID последовательность возврата **не определена**.



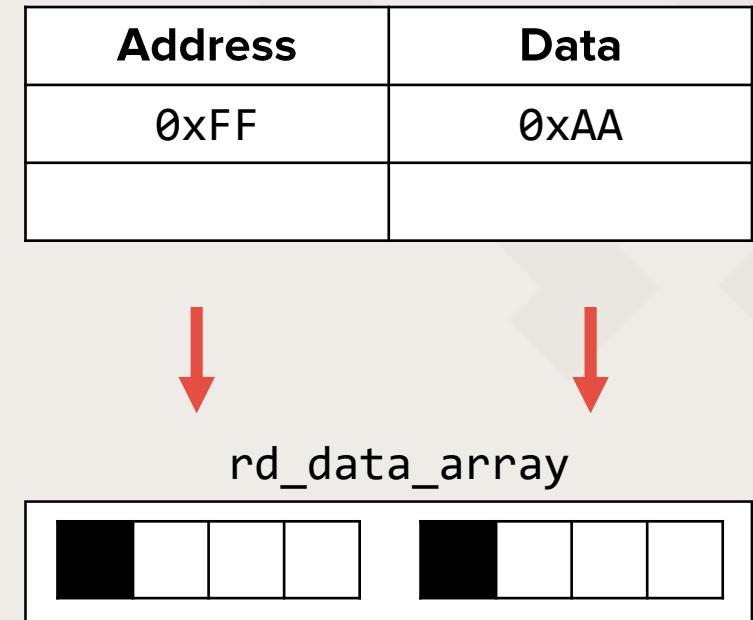
AXI4 VIP: axi_slave.sv

```
...
if (arvalid & arready) begin
  if (! memory.exists(araddr)) begin
    $display ("slave: attempt to read ...");

    rd_data_array [arid].push_back ('x);
  end
  else begin
    $display ("slave: read memory [%h] ...");

    rd_data_array [arid].push_back (
      memory [araddr]
    );
  end
end
...

```



Зависимость от ID

AXI4 VIP: axi_slave.sv

```
...
if (~ rvalid | rready) begin
...
random_offset = $urandom_range (0, n_ids - 1);
for (int i = 0; i < n_ids; i++) begin
...
id = (i + random_offset) % n_ids;
if (rd_data_array [id].size () > 0) begin
rdata <= rd_data_array [id].pop_front ();
rid <= id;
rvalid <= '1;
end
...

```



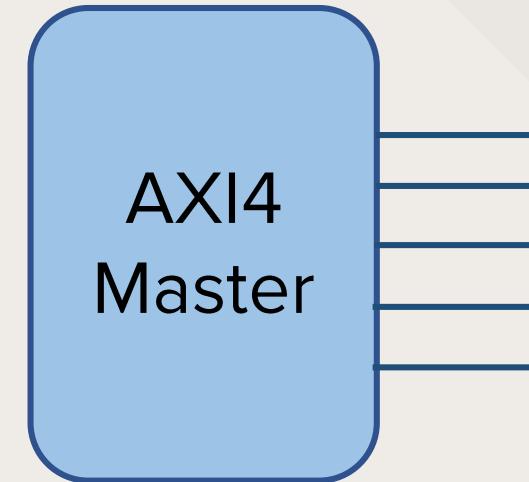
Случайный выбор

AXI4 VIP: axi_master.sv

```
module axi_master
(
    input          clk,
    input          rst,
    output addr_t araddr,
    output id_t   arid,
    output logic  arvalid,
    input          arready,
    ...
    output addr_t awaddr,
    output logic  bready
);

import axi_transaction::*;

...
```



AXI4 VIP: axi_master.sv

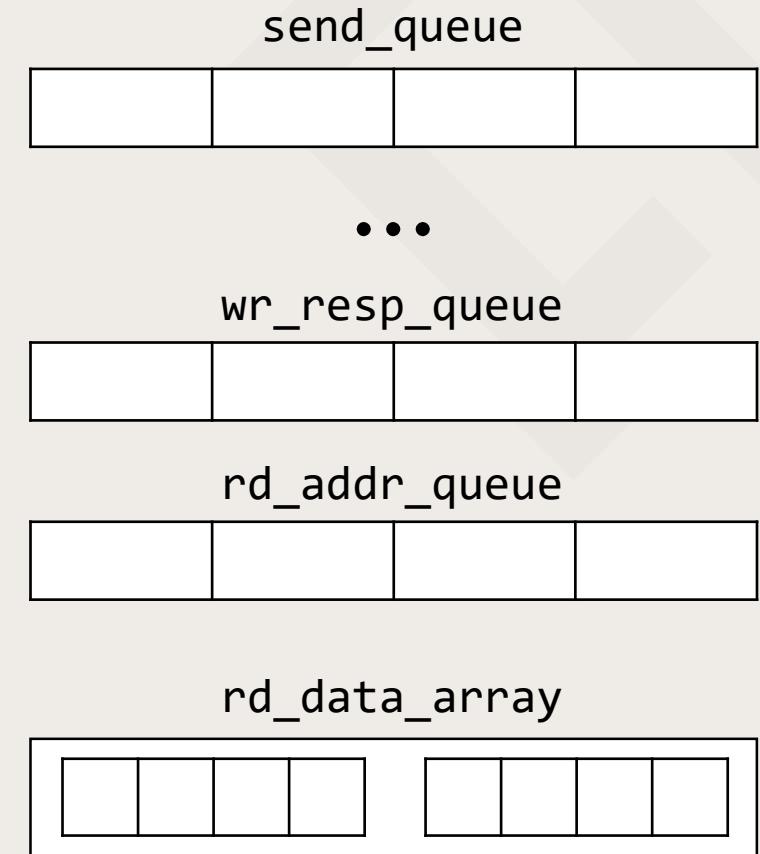
```
...
initial reset_probabilities ();

//-----  
// Queues and scoreboards

axi_transaction send_queue   [$];
axi_transaction rd_addr_queue[$];
...
axi_transaction rd_data_array [n_ids][$];
axi_transaction wr_resp_queue[$];

`define complete_everything wait fork;

...
```

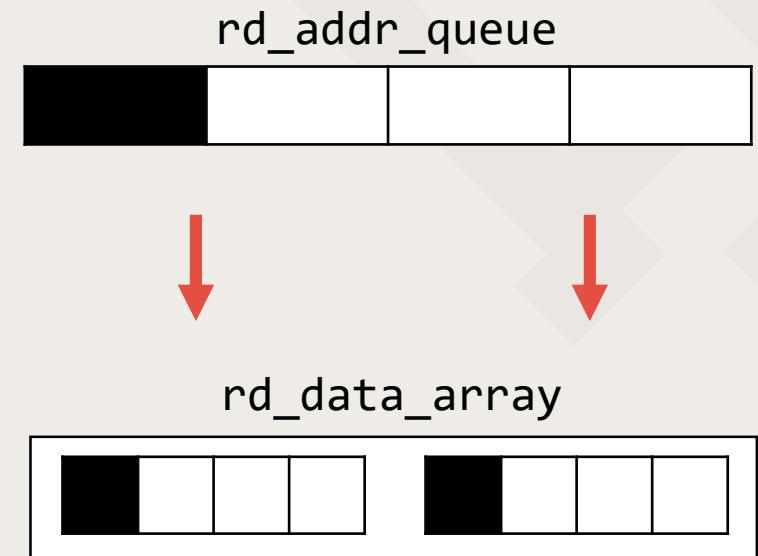


AXI4 VIP: axi_master.sv

```
...
if (arvalid & arready)
begin
    assert (rd_addr_queue.size () > 0);
    tr = rd_addr_queue.pop_front ();

    rd_data_array [tr.id].push_back (tr);

    $display ("%0d master: read address ...
end
...
...
```



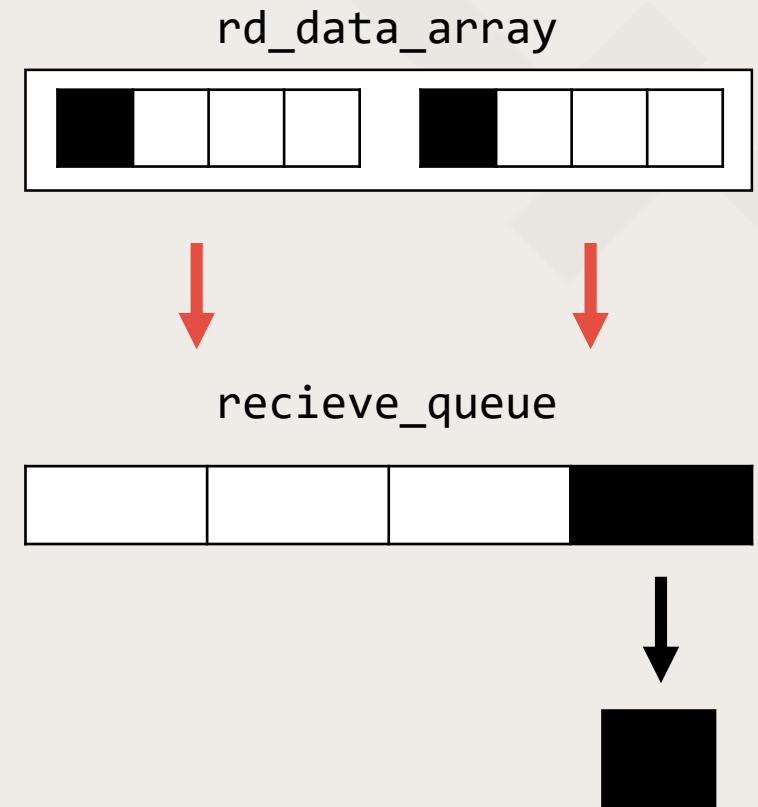
Зависимость от ID

AXI4 VIP: axi_master.sv

```
...
if (rvalid & rready) begin
    if (rd_data_array [rid].size () == 0)
        $fatal ("Unexpected read data");
    else
        begin
            tr = rd_data_array [rid].pop_front ();
            tr.data      = rdata;
            tr.data_is_set = 1;
            $display ("%0d master: received read ...");
            receive_queue.push_back (tr);
        end
end
...

```

Зависимость от ID



AXI4 VIP: axi_master.sv и axi_slave.sv

- **Практика**
- gitflic.ru/project/yuri-panchul/valid-ready-etc
- 08_axi_master_slave_monitor/53_axi_master_resp