



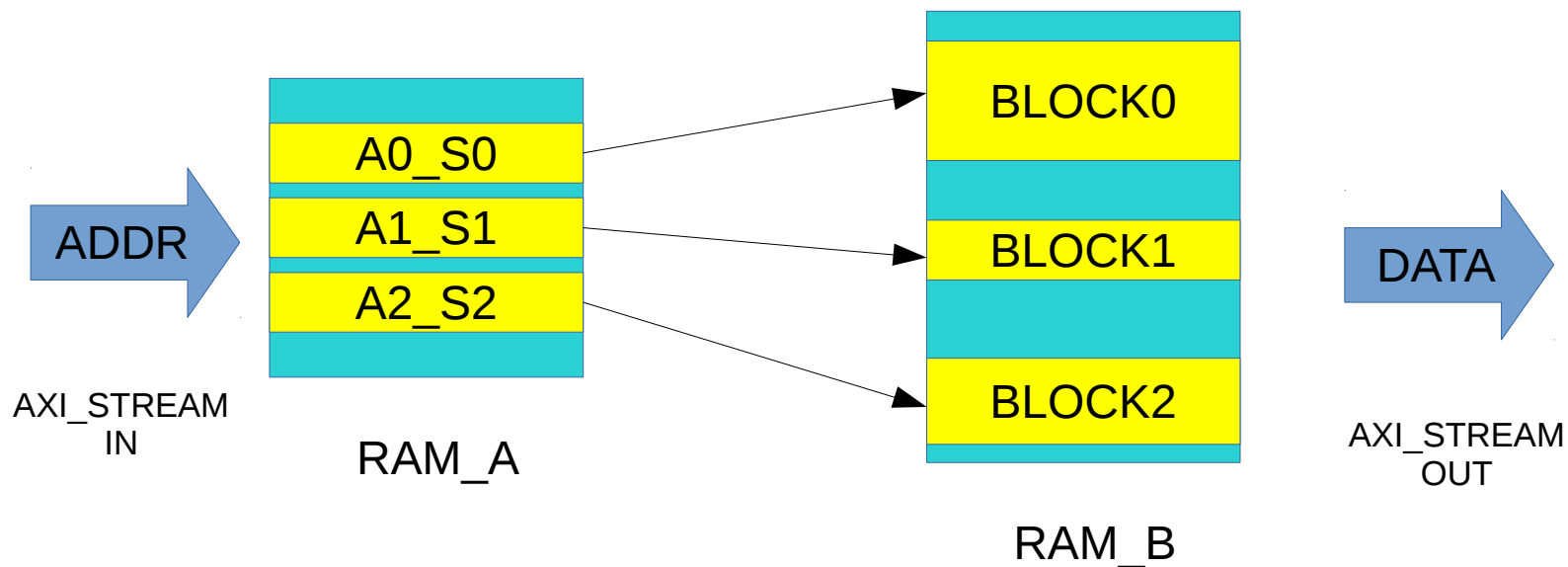
Пример 2.1.1.7



Основные цели

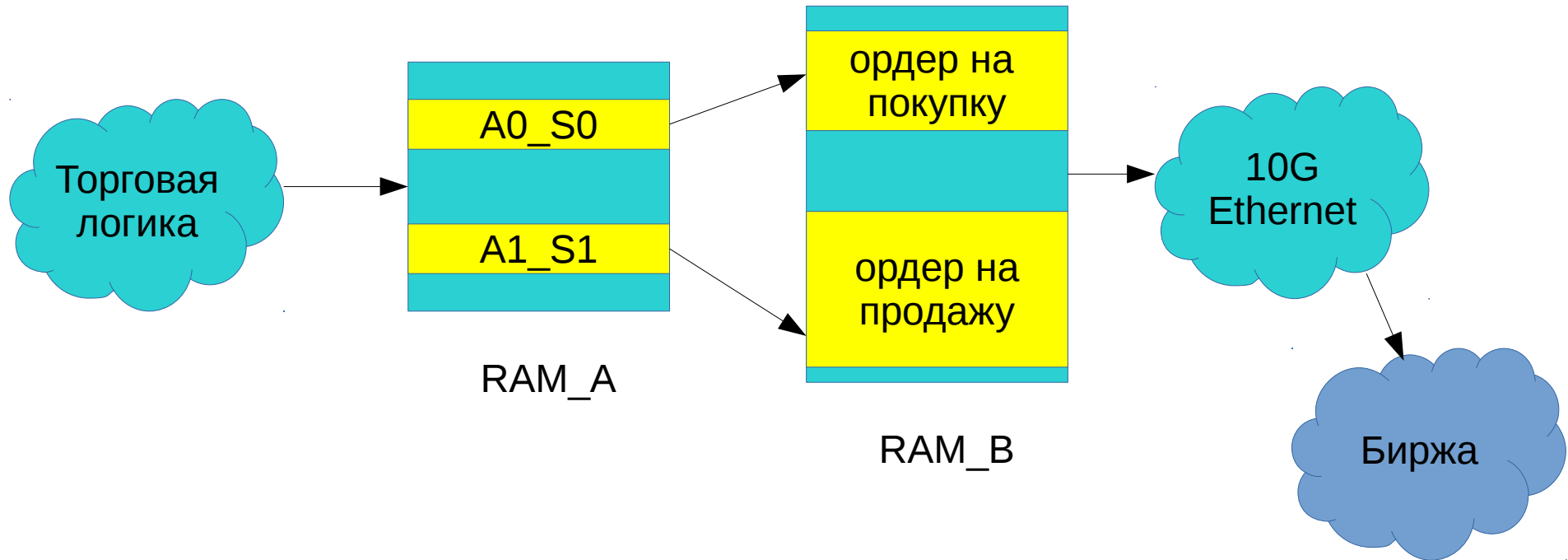
- Понятие счётчика кредитов
- Применение интерфейсов SystemVerilog
- Применение классов SystemVerilog
- Применение наследования классов
- Построение стенда тестирования

Постановка задачи



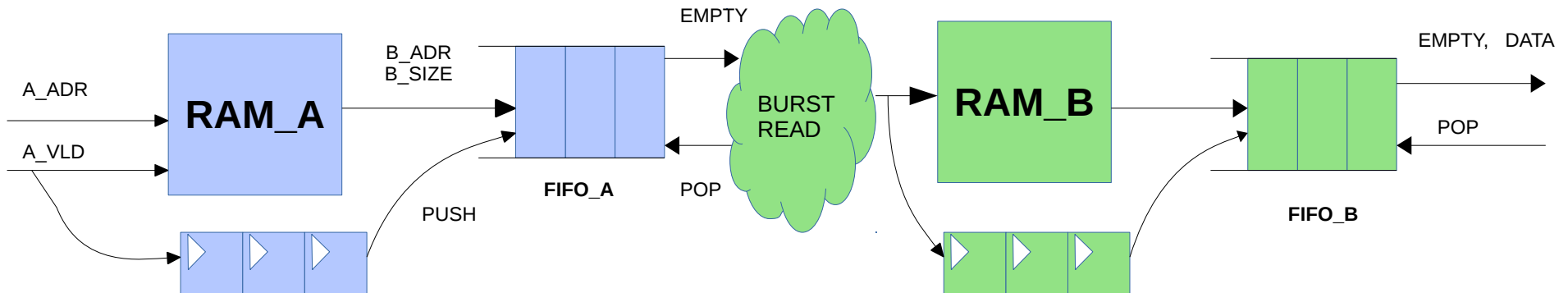
- Память RAM_A содержит адрес и размер блока
- Память RAM_B содержит блок данных
- Входной AXI-Stream — приходит адрес памяти RAM_A
- Выходной стрим — блок данных из памяти RAM_B

Пример применения в HFT



- Торговая логика формирует команду на покупку или продажу
- Память RAM_A содержит адрес и размер пакета для отправки в сеть Ethernet
- Память RAM_B содержит сам пакет

Микроархитектура step1

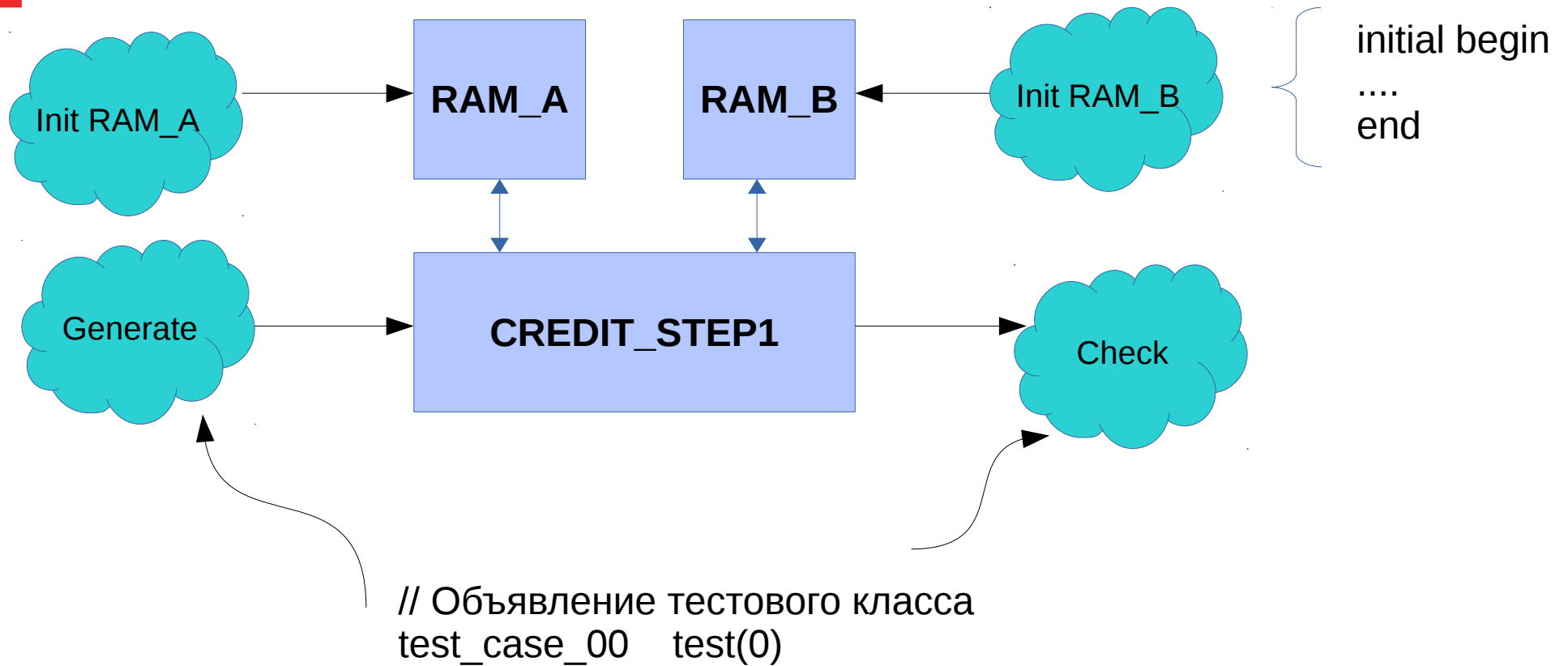


Проблемы:

- FIFO_A, FIFO_B — возможно переполнение
- Какой должен быть размер FIFO ?

Разработка теста

step1



Базовый класс base_test

```
typedef struct
{
    logic [7:0]  addr;  //!< value for address
    int         delay; //!< delay after send
    int         last;  //!< 1 - last data
} type_test_input;
```

```
type_test_input test_input[128];
int              max_test_input;
int              current_index;
```

```
logic [15:0] ram0_data[256];
logic [15:0] ram1_data[256];
```

```
class base_test;
```

```
function new(); // Инициализация
```

```
function type_test_input get_input();
```

```
function logic[15:0] get_ram0( int addr );
```

```
function type_test_expect get_expect()
```

```
function void create_expect_data();
```

Класс test_case_00 - формирование данных

```
class test_case_00 extends base_test;
```

```
function init_mode0();
```

```
function init_mode1();
```

```
function init_mode1();
```

```
function new ( int mode );
```

Класс формирует массивы
ram0_data, ram1_data, test_input

По этим массивам формируются
данные для подачи на вход и для
сравнения с выходом

```
function init_mode0;
```

```
int ii=0;
```

```
test_input[ii].addr = 8'h02; test_input[ii].delay = 0; ii++;
```

```
test_input[ii].addr = 8'h08; test_input[ii].delay = 0; ii++;
```

```
test_input[ii].addr = 8'h0A; test_input[ii].delay = 0; ii++;
```

```
test_input[ii].addr = 8'h0D; test_input[ii].delay = 0; ii++;
```

```
test_input[ii].addr = 8'h02; test_input[ii].delay = 0; ii++;
```

```
test_input[ii].addr = 8'h08; test_input[ii].delay = 0; ii++;
```

```
test_input[ii].addr = 8'h0A; test_input[ii].delay = 0; ii++;
```

```
test_input[ii].addr = 8'h0D; test_input[ii].delay = 16; ii++;
```

```
test_input[ii].addr = 8'h08; test_input[ii].delay = 16; ii++;
```

```
test_input[ii].addr = 8'h0F; test_input[ii].delay = 16; ii++;
```

```
test_input[ii].addr = 8'h10; test_input[ii].delay = 16; ii++;
```

```
max_test_input=ii;
```

```
endfunction;
```


Результат тестирования

В общем случае недостаточно иметь один тест. Требуется группа тестов которая позволит проверить функционирование в разных условиях.

Возможно применение разных технологий.
Например скрипт на TCL или Google Test.

Требование — запуск одиночного теста и запуск выделенной группы тестов.
При запуске группы тестов для каждого теста необходимо сохранять лог работы.

Пример вывода:

```
TEST 0  PASSED
TEST 1  PASSED
TEST 2  FAILED
```

```
-----
Total test:  3
Passed:     2
Failed:     1
```

Интерфейсы в SystemVerilog

```
interface type_i_axis #(parameter WIDTH = 8);  
    logic [WIDTH-1:0]  tdata;  
    logic              tvalid;  
    logic              tlast;  
    logic              tready;  
  
    modport master (  
        output tdata,  
        output tvalid,  
        output tlast,  
        input  tready  
    );  
  
    modport slave (  
        input  tdata,  
        input  tvalid,  
        input  tlast,  
        output tready  
    );  
  
endinterface
```

Интерфейсы позволяют резко снизить количество сигналов для связи между компонентами.

Достаточно иметь только один сигнал.

modport позволяет определить состав и направление сигналов для каждого варианта использования интерфейса

Компонент credit_07_step1

```
module credit_07_step1
(
    input wire      reset_p,           //! 1 - reset
    input wire      clk,              //! clock

    type_i_axis.slave    stream_addr,  //! Input stream with addr for RAM0

    type_i_mem_rd.master ram0_rd,      //! Interface for read from RAM0

    type_i_mem_rd.master ram1_rd,      //! Interface for read from RAM1

    type_i_axis.master   stream_data  //! Output stream with data from RAM1
);
```

Подключение компонента

```
assign fifo_0_overflow = credit_07_dut.fifo_w4_overflow;
assign fifo_1_overflow = credit_07_dut.fifo_out_overflow;
```

credit_07_step1

credit_07_dut

```
(
    .reset_p      ( reset_p      ),
    .clk          ( clk          ),

    .stream_addr  ( stream_addr  ),
    .ram0_rd      ( ram0_rd      ),
    .ram1_rd      ( ram1_rd      ),

    .stream_data  ( stream_data  )

);
```

cr_ram256x16 ram_a

```
(
    .reset_p      ( reset_p      ),
    .clk          ( clk          ),
    .mem_wr       ( ram0_wr      ),
    .mem_rd       ( ram0_rd      )

);
```

```
type_i_axis      stream_addr();
type_i_axis #(WIDTH(16)) stream_data();
```

```
type_i_mem_rd ram0_rd();
type_i_mem_rd ram1_rd();
```

```
// Формирование результата теста
initial begin
```

```
.....
```

```
$display("Time: %0t", $time);
if( cnt_ok==test.get_max_expect &&
    0==cnt_error &&
    0==fifo_0_overflow &&
    0==fifo_1_overflow)
begin
    $display( "TEST PASSED");
end else begin
    $display( "TEST FAILED");
end
end
end
```

Тестирование credit_07_step1

TEST 0 PASSED
TEST 1 FAILED
TEST 2 FAILED

Total test: 3
Passed: 1
Failed: 2

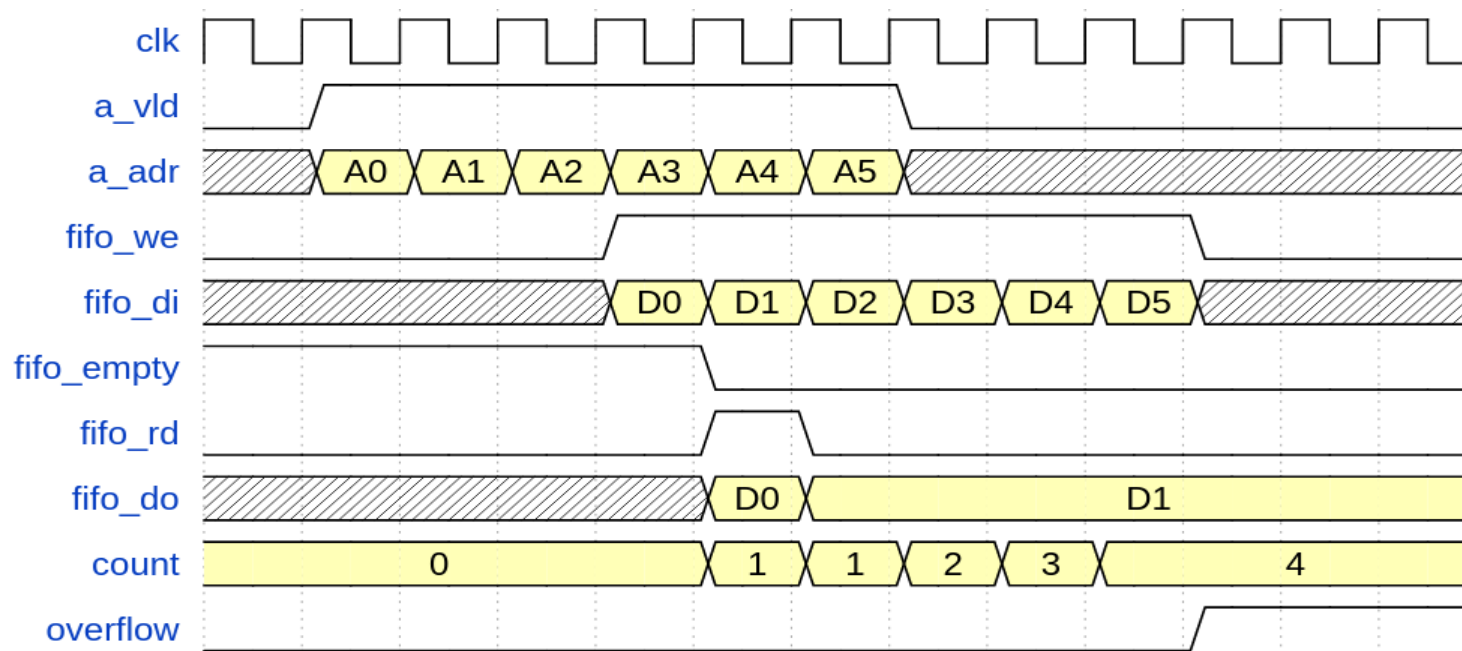
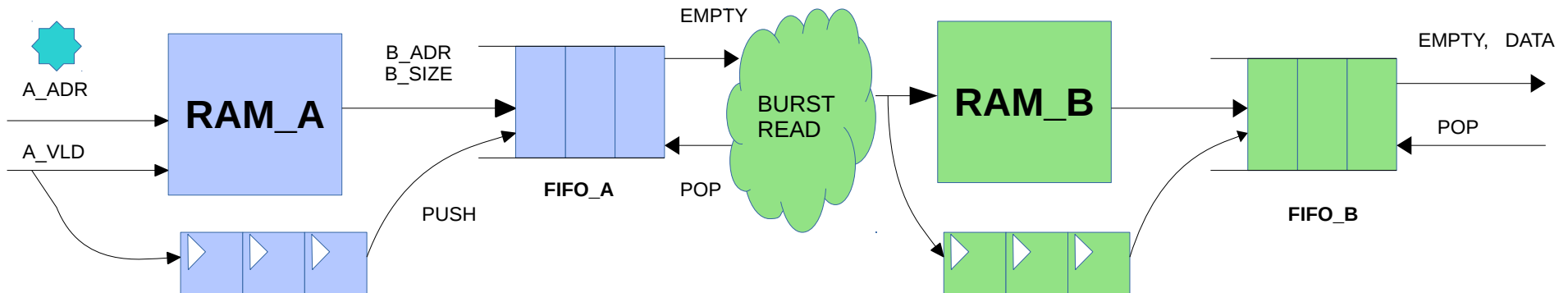
TEST 0 - редкие команды. FIFO не переполняется и тест проходит успешно.

TEST 1 - несколько подряд идущих команд.
Переполняется FIFO_0

TEST 2 - несколько редких команд с передачей
максимального пакета. Переполняется FIFO_1

Тестовые наборы данных созданы с учётом знания архитектуры компонента

Переполнение FIFO_A



Приостановка записи в FIFO

Вариант 1

Использование флагов FIFO

Контролируется флаг почти заполненного FIFO или флаг заполнения половины FIFO

Уровень формирования флага FIFO должен соответствовать размеру конвейера.

После остановки записи по флагу в FIFO ещё будет записано N слов.

Разрешение записи тоже будет с задержкой

Вариант 2

Использование счётчика кредитов

Счётчик кредитов устанавливается перед узлом с конвейером

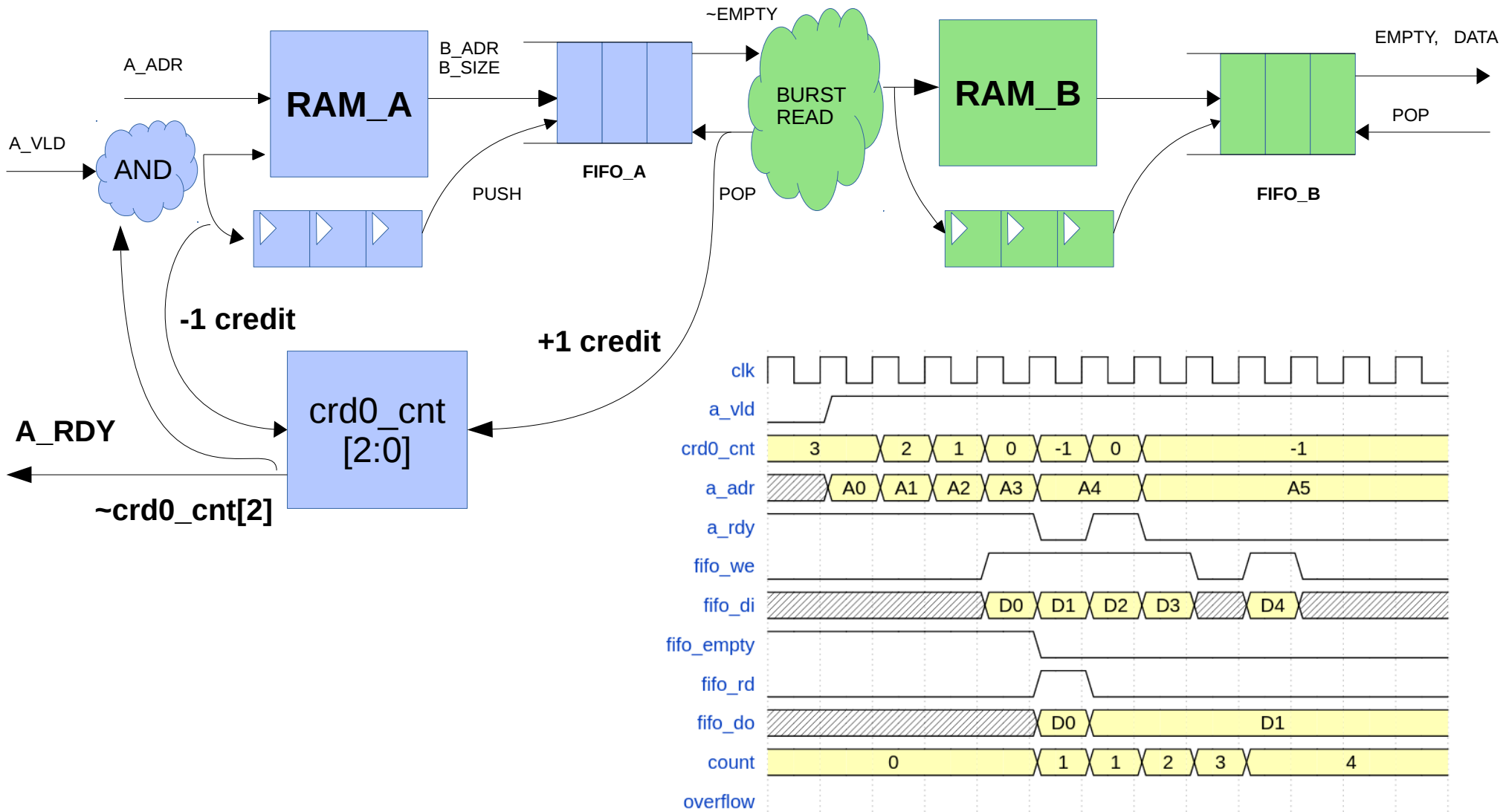
Запись останавливается сразу как только в конвейер будет записано число заданное число слов

Запись начинается сразу как только будет вычитано слово из FIFO

Два преимущества:

- Производительность
- Требуется FIFO меньшего размера

Step2 — Добавление счётчика кредитов 1



Тестирование credit_07_step2

TEST 0 PASSED
TEST 1 **PASSED**
TEST 2 FAILED

Total test: 3
Passed: 2
Failed: 1

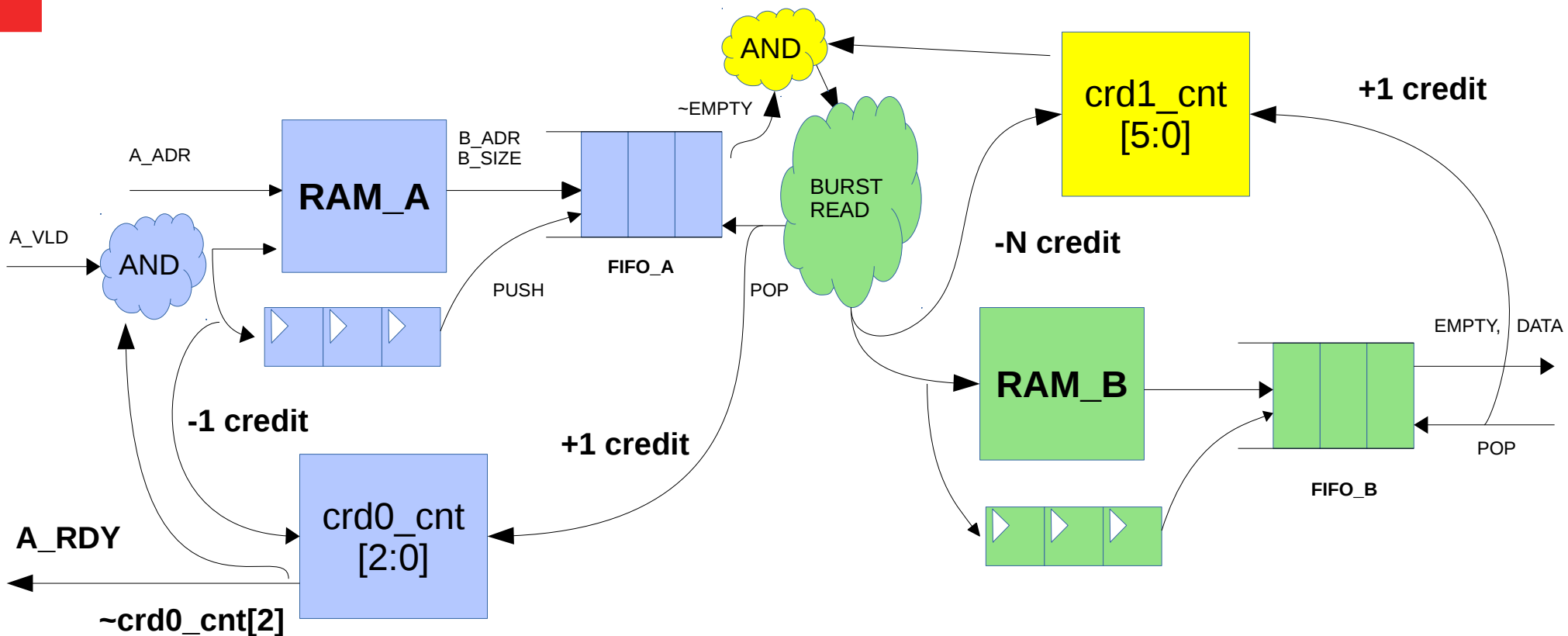
TEST 0 - редкие команды. FIFO не переполняется и тест проходит успешно.

TEST 1 - несколько подряд идущих команд.
Переполняется FIFO_0

TEST 2 - несколько редких команд с передачей
максимального пакета. Переполняется FIFO_1

Тестовые наборы данных созданы с учётом знания архитектуры компонента

Step3 — Добавление счётчика кредитов 2



Особенность - Счётчик crd1_cnt уменьшается сразу на N кредитов

N — это число слов которое надо прочитать из RAM_B

Тестирование credit_07_step3

TEST 0 PASSED
TEST 1 PASSED
TEST 2 **PASSED**

Total test: 3
Passed: 3
Failed: 0

TEST 0 - редкие команды. FIFO не переполняется и тест проходит успешно.

TEST 1 - несколько подряд идущих команд.
Переполняется FIFO_0

TEST 3 - несколько редких команд с передачей
максимального пакета. Переполняется FIFO_1

Тестовые наборы данных созданы с учётом знания архитектуры компонента

Все примеры доступны на GitHub