Многообразие реализаций очередей FIFO

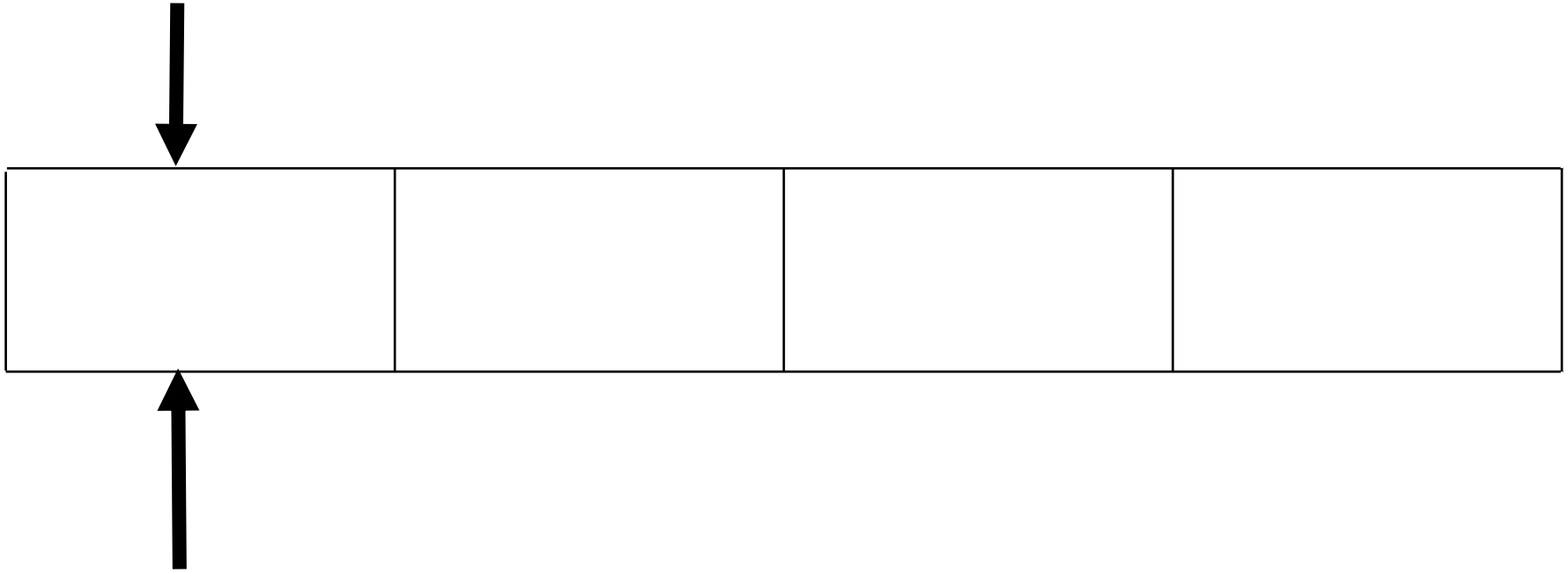
Сергей Иванец

FIFO – First In – First Out

- Модуль памяти типа «первый вошел – первый вышел».
- Циклический буфер.
- Для адресации используются внутренние указатели чтения и записи.
- Для управления используются сигналы чтения и записи.
- Для обозначения заполненности используются сигналы «Пустой» и «Полный».

FIFO пуст

Указатель записи: 00

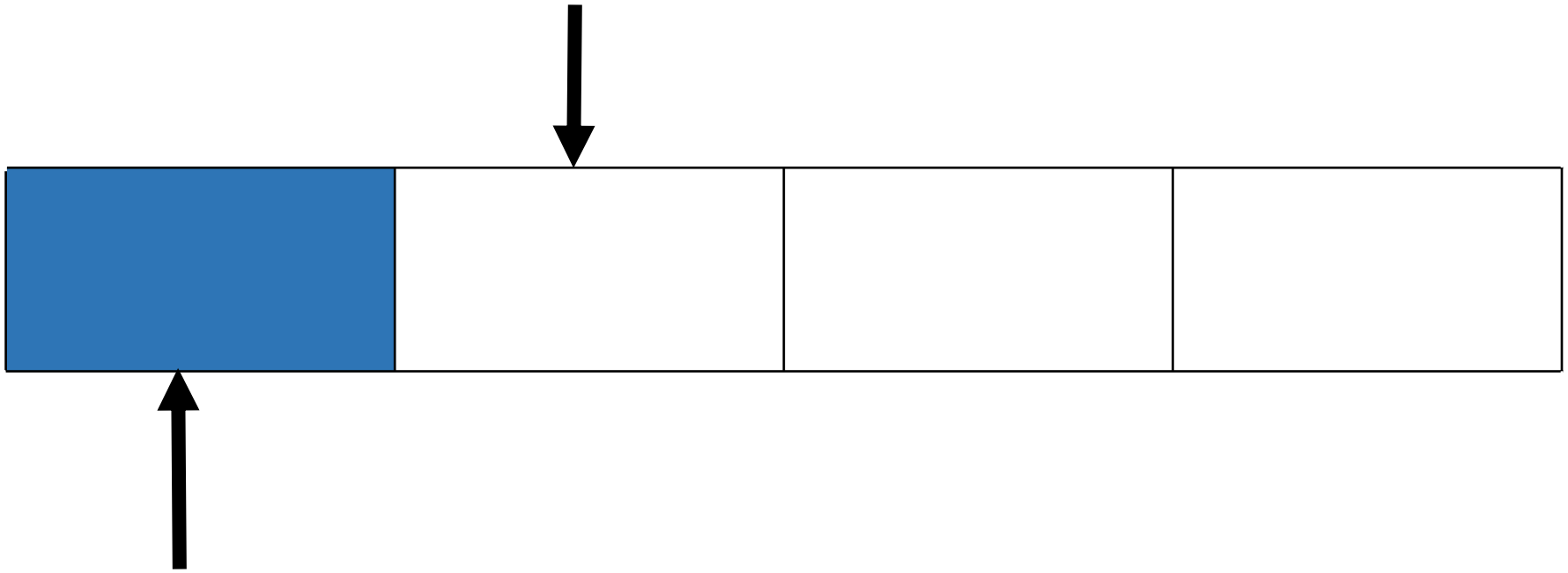


Указатель чтения: 00

Empty 1 Full 0

Запись первого байта

Указатель записи: 01

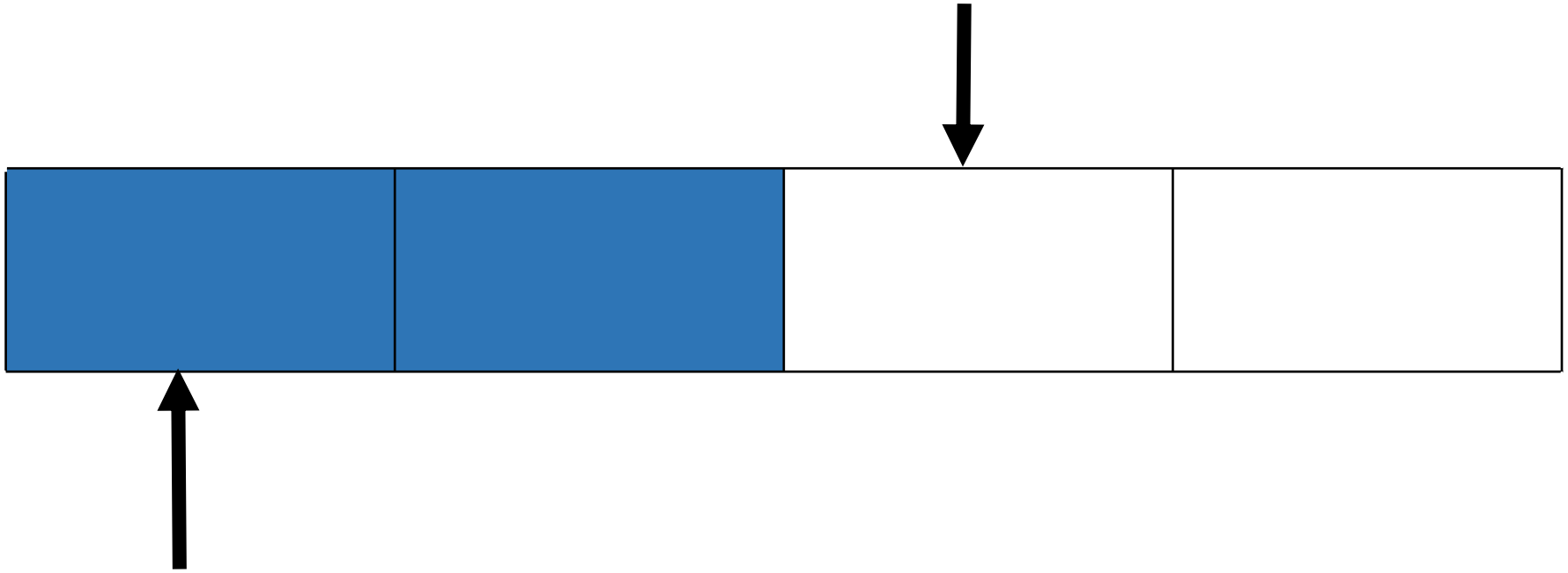


Указатель чтения: 00

Empty **0** Full 0

Запись второго байта

Указатель записи: 10

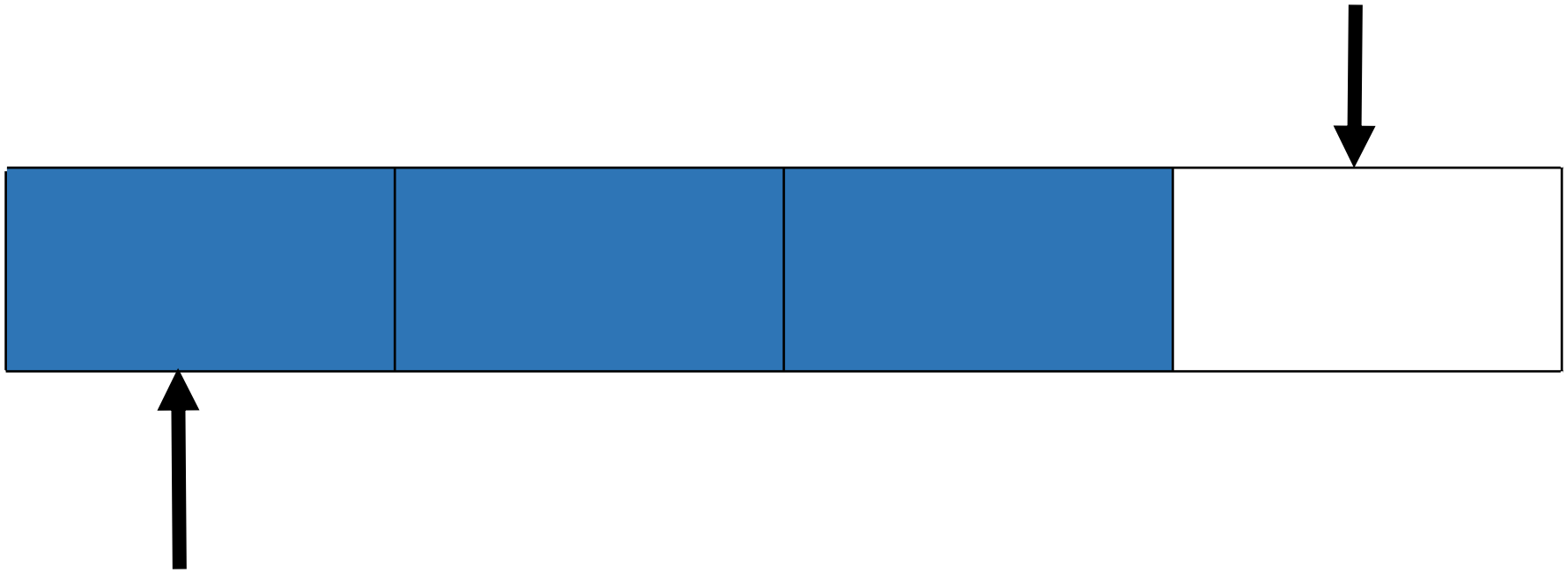


Указатель чтения: 00

Empty 0 Full 0

Запись третьего байта

Указатель записи: 11

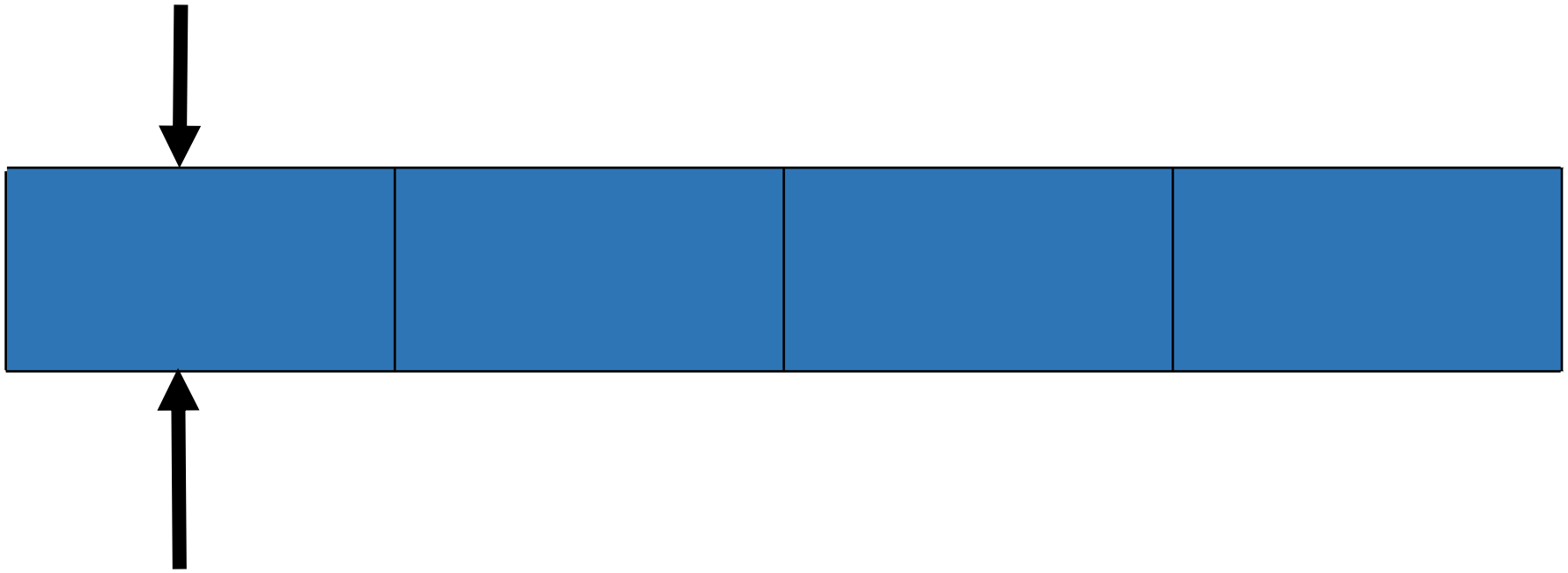


Указатель чтения: 00

Empty 0 Full 0

Запись четвертого байта. FIFO заполнен?

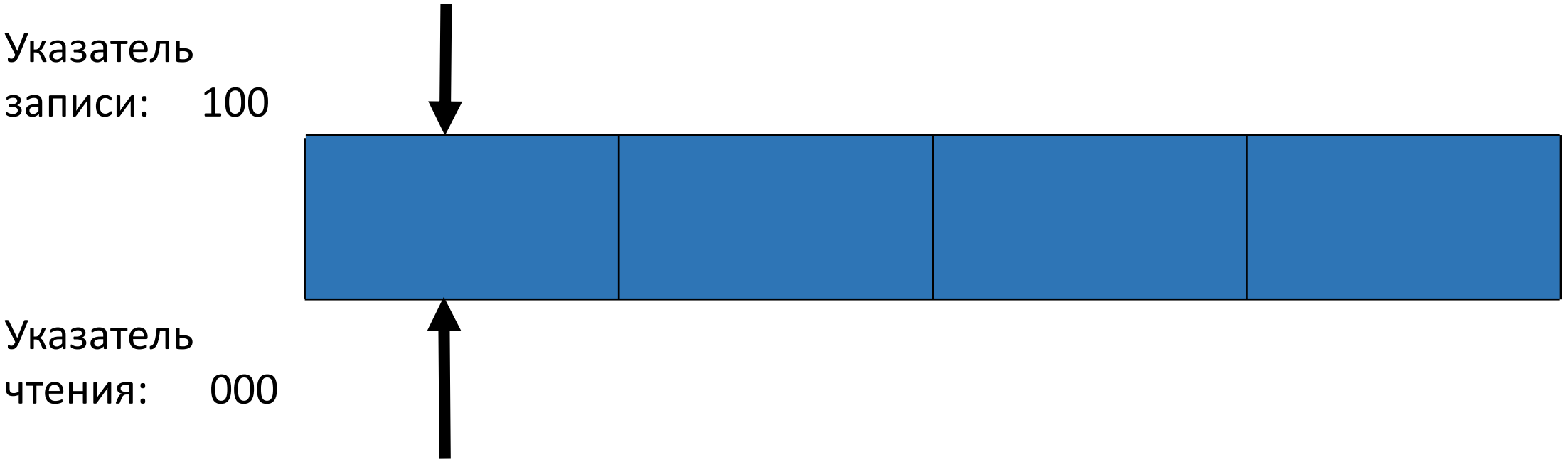
Указатель записи: 00



Указатель чтения: 00

Empty 0 **Full** **1 или 0?**

Запись четвертого байта. Дополнительный бит в указателе

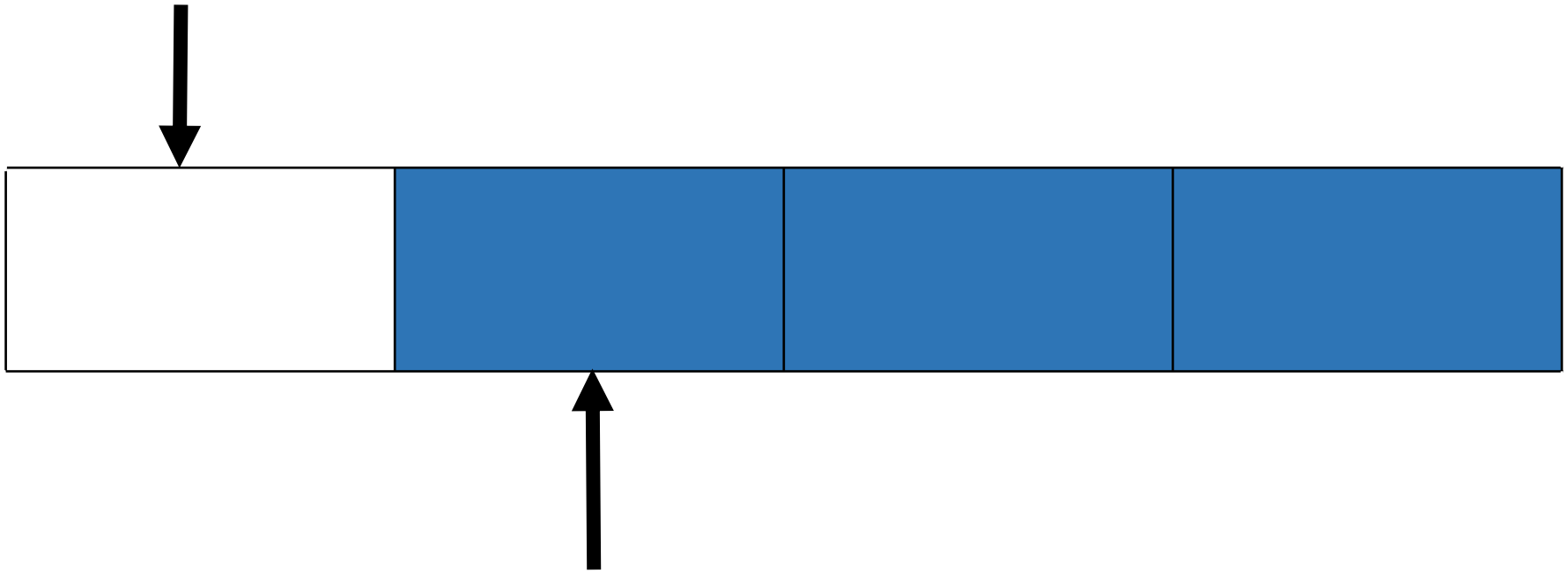


Empty 0 **Full 1**

```
assign full = (wr_ptr[2] ^ rd_ptr[2]) & (wr_ptr[1:0] == rd_ptr[1:0]);
```

Чтение первого байта

Указатель записи: 100

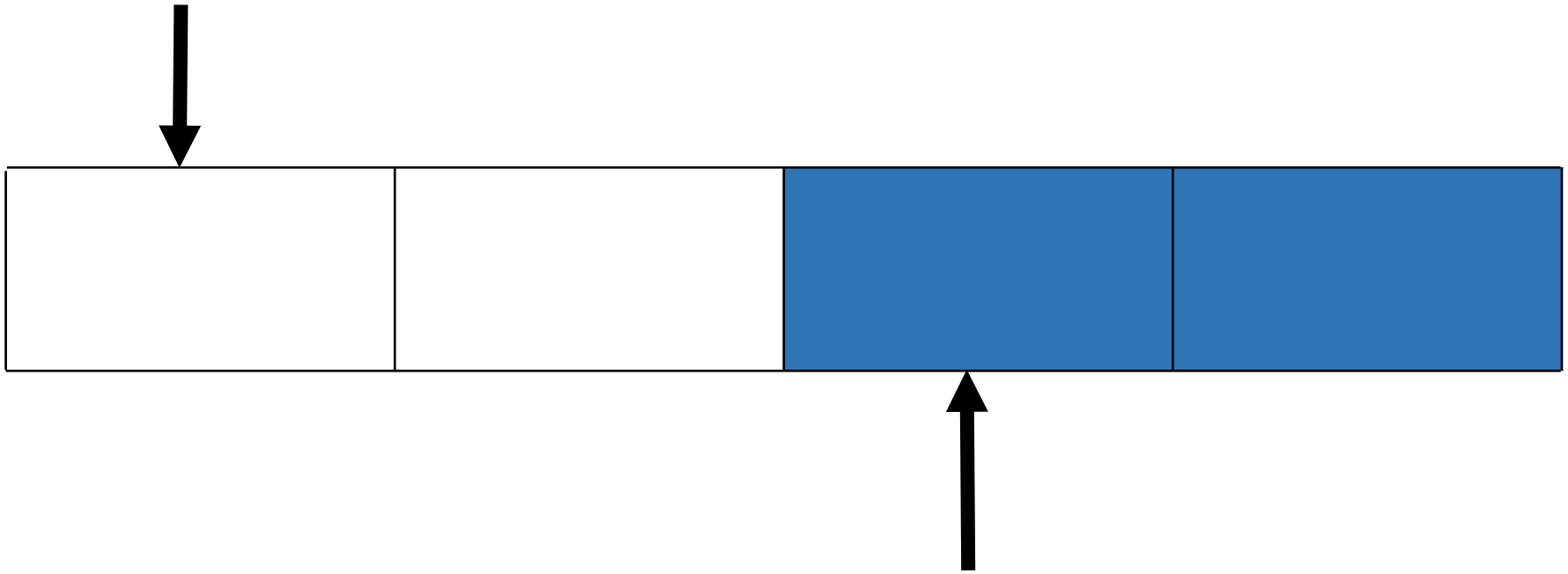


Указатель чтения: 001

Empty 0 **Full** **0**

Чтение второго байта

Указатель записи: 100

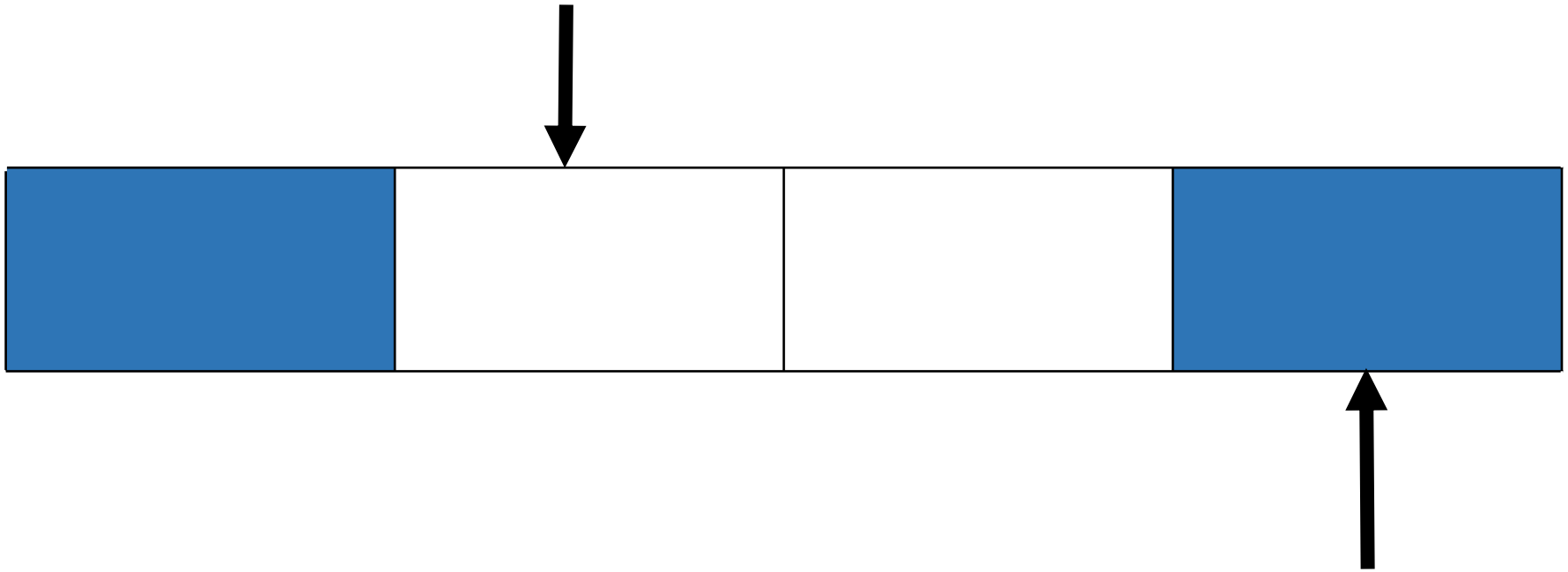


Указатель чтения: 010

Empty 0 Full 0

Чтение и запись одновременно

Указатель записи: 101

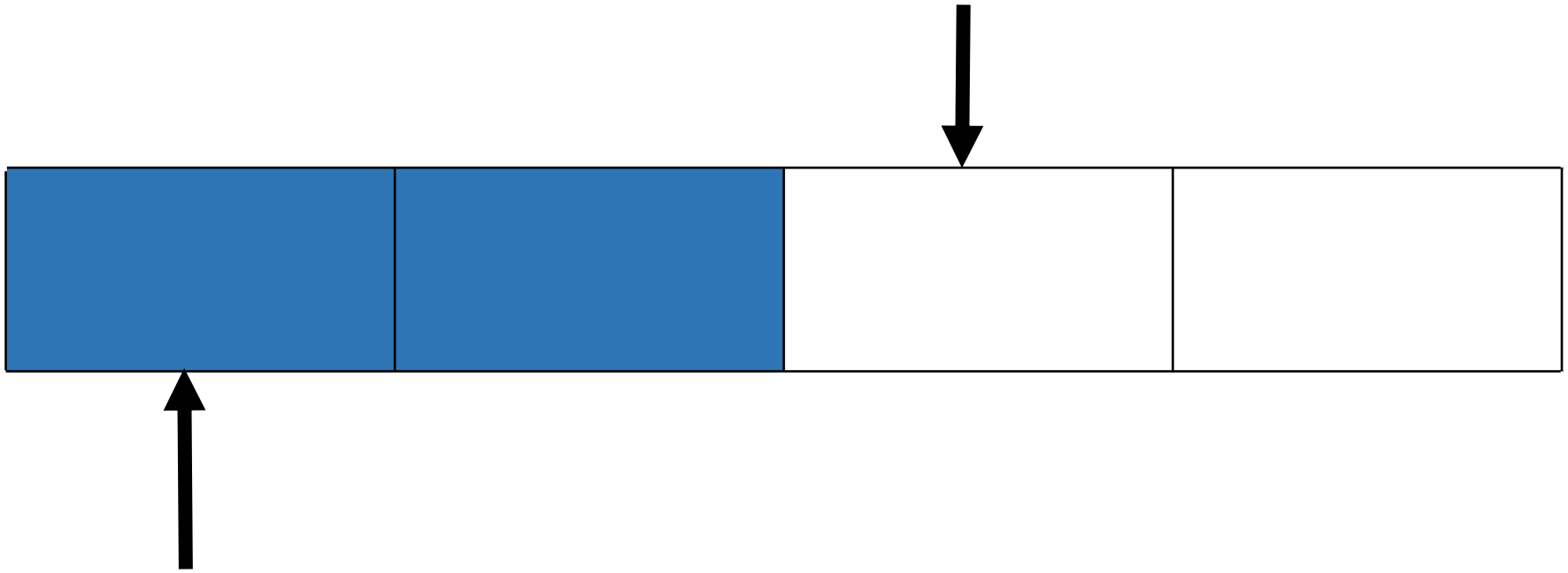


Указатель чтения: 011

Empty 0 Full 0

Чтение и запись одновременно

Указатель записи: 110

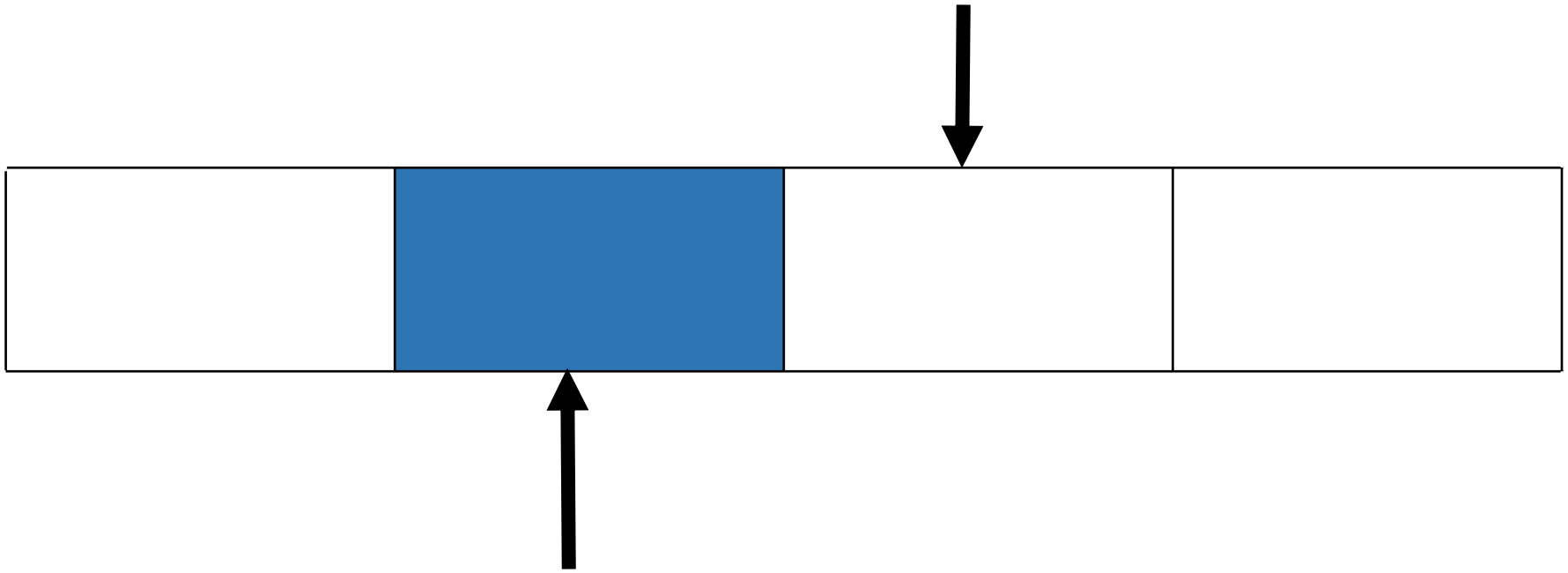


Указатель чтения: 100

Empty 0 Full 0

Чтение

Указатель записи: 110

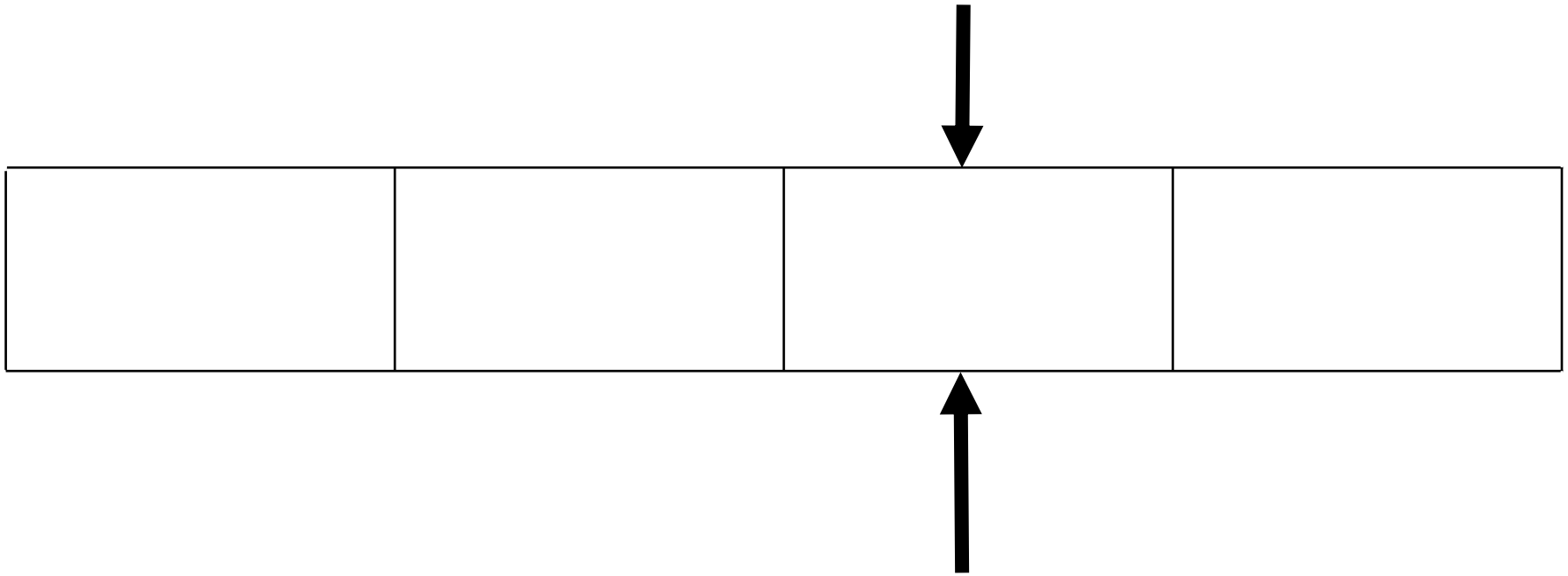


Указатель чтения: 101

Empty 0 Full 0

Чтение. FIFO пуст

Указатель записи: 110



Указатель чтения: 110

Empty **1** Full 0

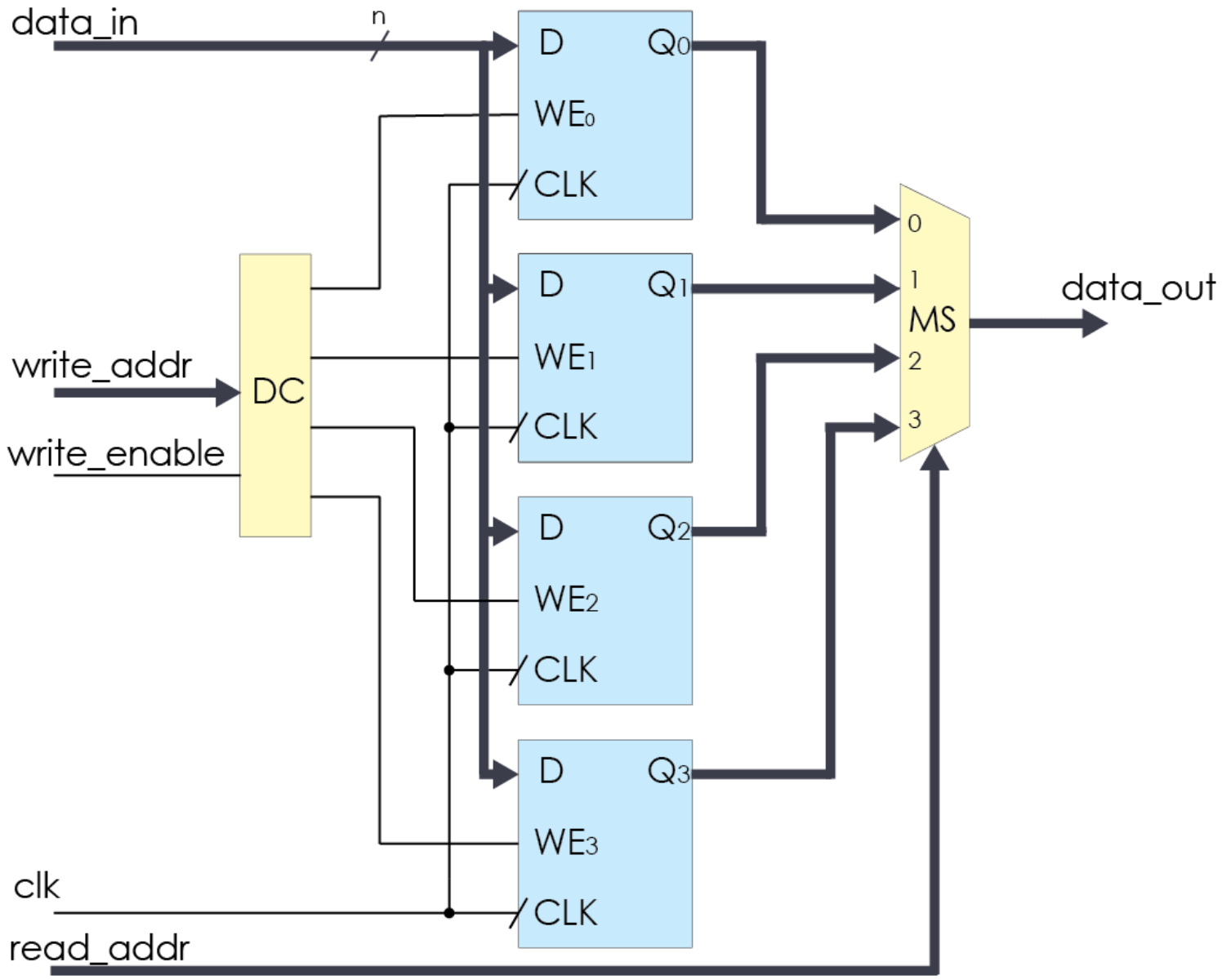
```
assign empty = (wr_ptr == rd_ptr);
```

Блок памяти для FIFO

- Регистровый файл
- Внутренняя память ПЛИС
- Внешняя память или память на кристалле

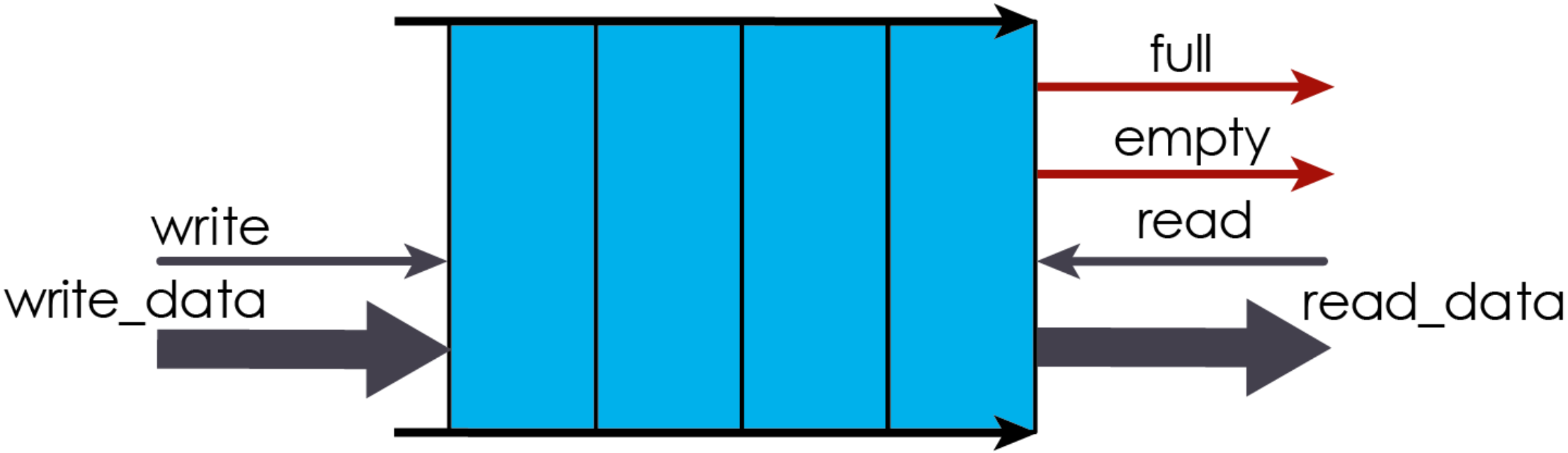
Регистровый файл

- Два порта:
  - Порт записи
  - Порт чтения
- Возможность чтения и записи различных регистров одновременно
- В данном случае реализован на триггерах логических элементов (Logic Element) ПЛИС



Пример реализации простейшего FIFO

Порты простейшего FIFO




```
#(
   parameter    FIFO_DEPTH      = 4,
                FIFO_DATA_WIDTH = 8)
```

Порты

Порт	Разрядность	Направление	Описание
clk	1	Input	Тактовый сигнал
reset	1	input	Сброс. Активная 1
write	1	input	Запись. Активная 1
read	1	input	Чтение. Активная 1
write_data	[FIFO_DATA_WIDTH-1:0]	input	Записываемые данные
read_data	[FIFO_DATA_WIDTH-1:0]	output	Читаемые данные
empty	1	output	FIFO пуст
full	1	output	FIFO полон

Параметры

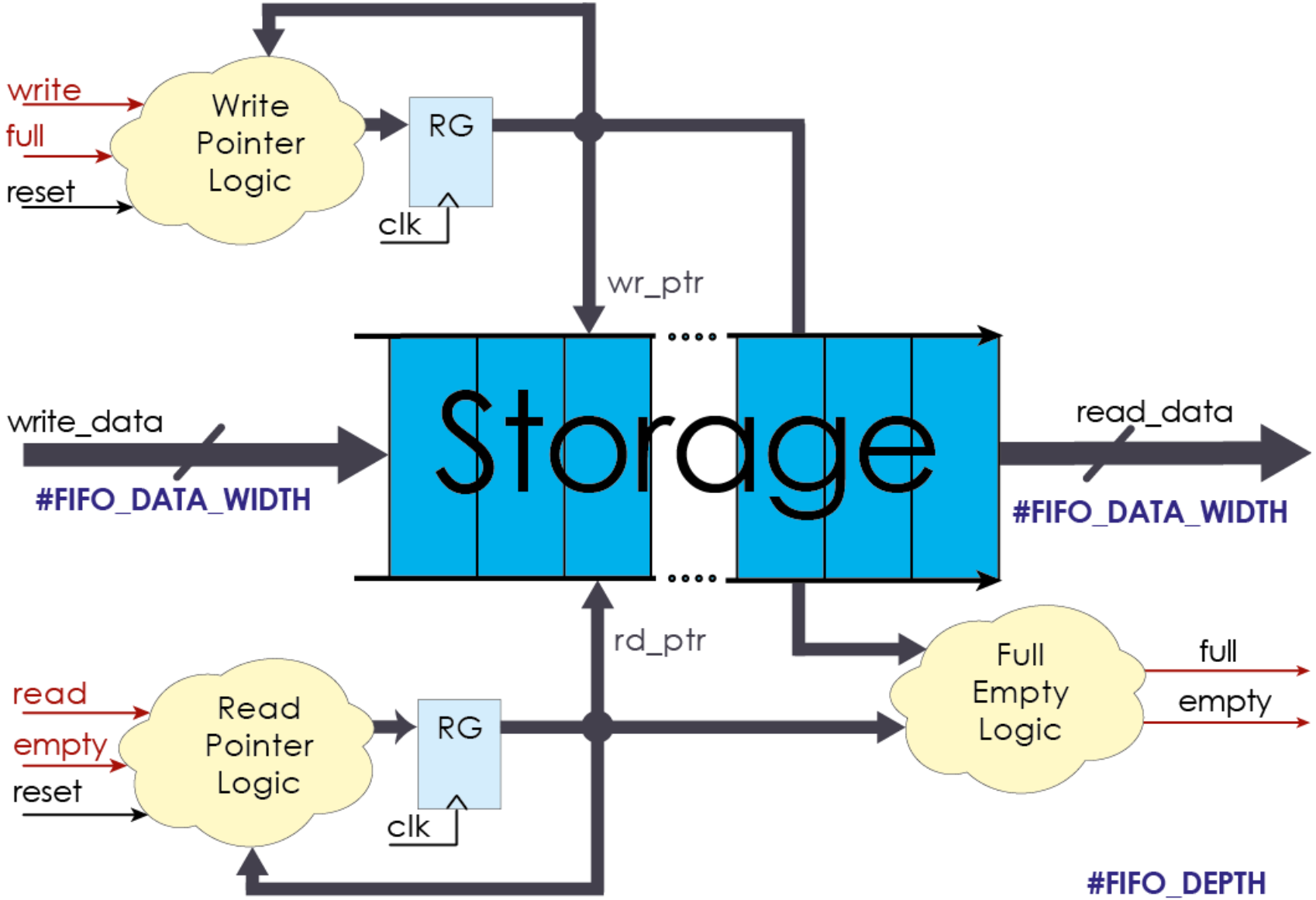
Параметр	Назначение
FIFO_DEPTH	Глубина FIFO – количество ячеек памяти в модуле
FIFO_DATA_WIDTH	Разрядность данных

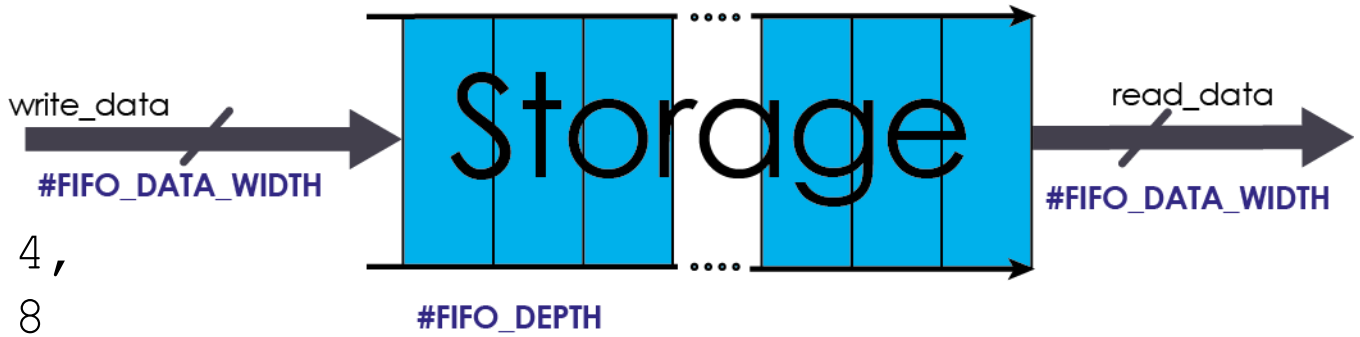
Структура простейшего FIFO

Параметры:

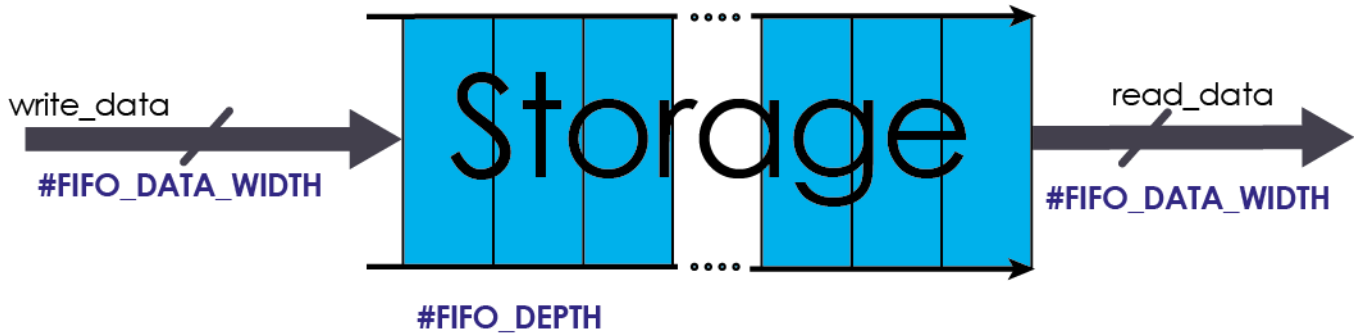
FIFO_DEPTH

FIFO_DATA_WIDTH



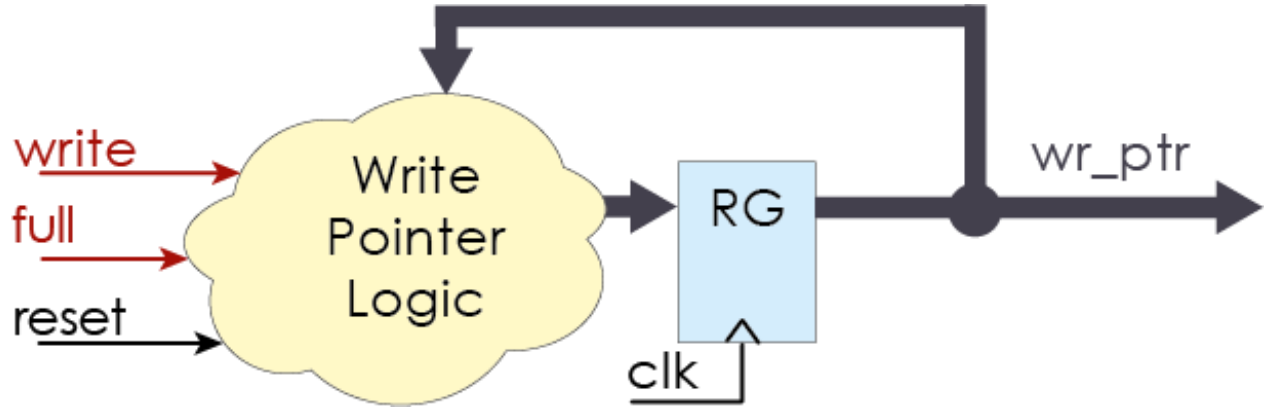


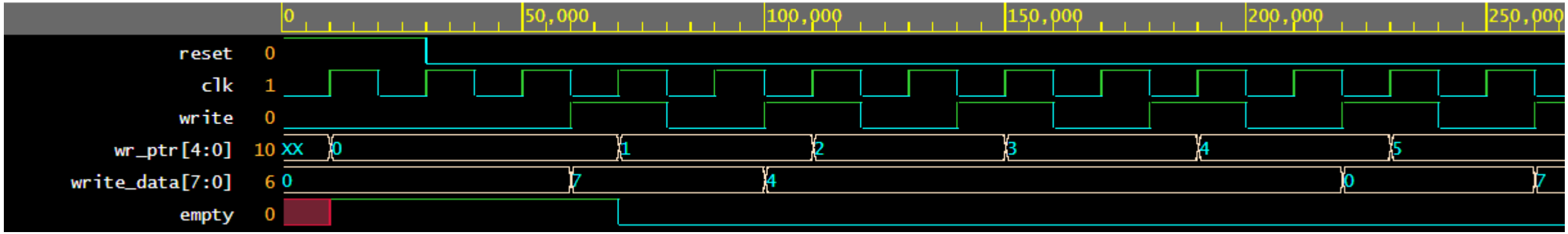

```
module fifo_simple
#(
  parameter    FIFO_DEPTH      = 4,
               FIFO_DATA_WIDTH = 8
)
(
  input                                clk,
  input                                reset,

  input                                write,
  input                                read,

  input       [FIFO_DATA_WIDTH-1:0] write_data,
  output reg  [FIFO_DATA_WIDTH-1:0] read_data,

  output                               empty,
  output                               full
);
```

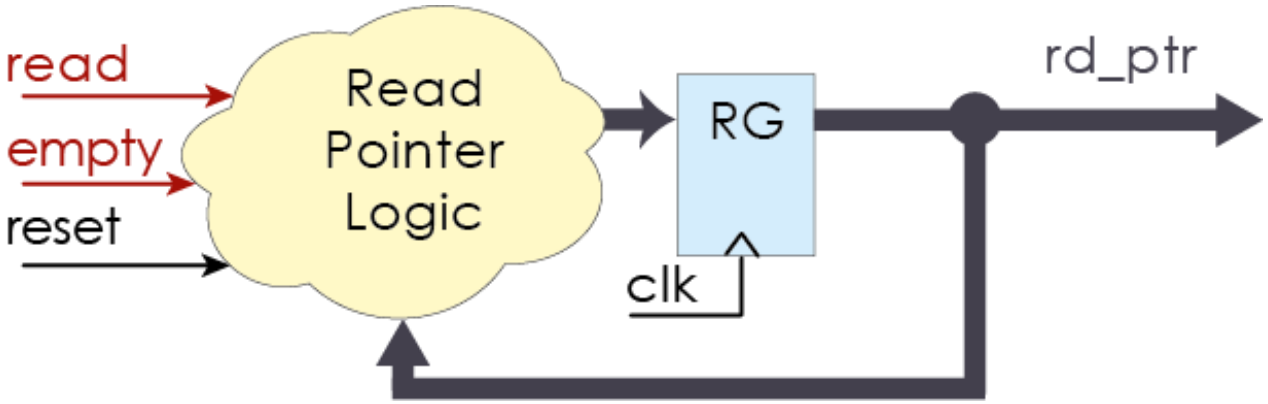



```
localparam  FIFO_PTR_WIDTH  = $clog2(FIFO_DEPTH) + 1;

reg [FIFO_DATA_WIDTH-1:0] fifo_array [FIFO_DEPTH-1:0];

reg [FIFO_PTR_WIDTH-1:0]  rd_ptr;
reg [FIFO_PTR_WIDTH-1:0]  wr_ptr;
```

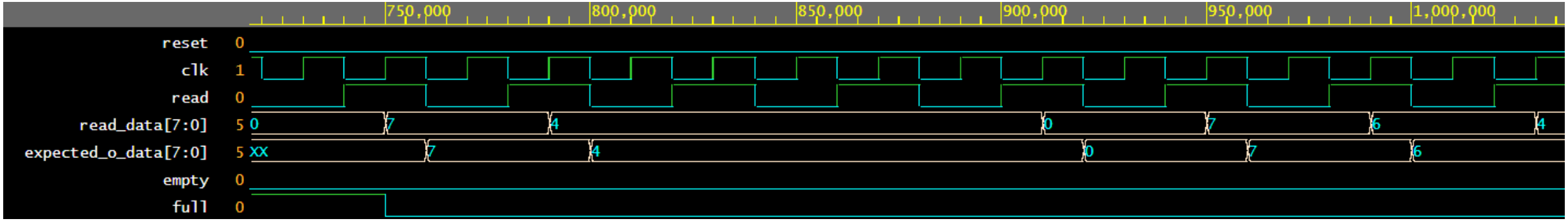
```
// Write Pointer Logic
always @ (posedge clk)
begin
   if (reset)
      wr_ptr <= {FIFO_PTR_WIDTH{1'b0}};
   else if (write & !full)
        wr_ptr <= wr_ptr + 1'b1;
end
```

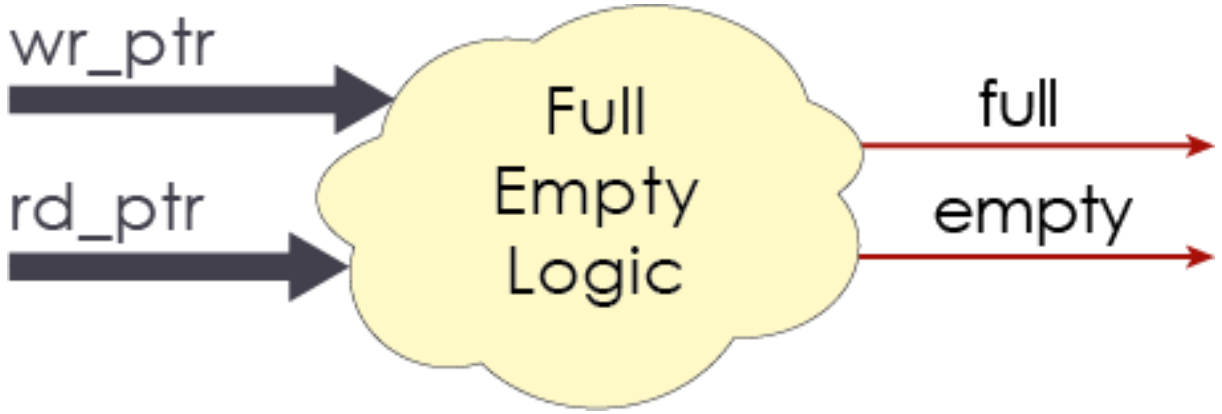







```
// Read Pointer Logic
 always @ (posedge clk)
 begin
   if (reset)
     rd_ptr <= {FIFO_PTR_WIDTH{1'b0}};
   else if (read & !empty)
       rd_ptr <= rd_ptr + 1'b1;
 end
```






```
// Full and Empty flags
assign full =
     (wr_ptr[FIFO_PTR_WIDTH-1] ^ rd_ptr[FIFO_PTR_WIDTH-1])
   & (wr_ptr[FIFO_PTR_WIDTH-2:0] == rd_ptr[FIFO_PTR_WIDTH-2:0]);
assign empty = (wr_ptr == rd_ptr);
```

```
// FIFO Write
 always @ (posedge clk)
 begin
   if (reset)
     fifo_array[wr_ptr] <= {FIFO_DATA_WIDTH{1'b0}};
   else if (write & !full)
       fifo_array[wr_ptr[FIFO_PTR_WIDTH-2:0]] <= write_data;
 end

 // FIFO Read
 always @ (posedge clk)  begin
   if (reset)
     read_data <= {FIFO_DATA_WIDTH{1'b0}};
   else if (read & !empty)
       read_data <= fifo_array[rd_ptr[FIFO_PTR_WIDTH-2:0]];
 end
```

Тестбенч. Direct тест

- Таски:
  - Сброс reset_task ()
  - Чтение read_fifo ()
  - Запись write_fifo ([7:0] data)
- Запись FIFO до заполнения
- Чтение FIFO до пустого
- Запись FIFO до заполнения
- Чтение FIFO до пустого

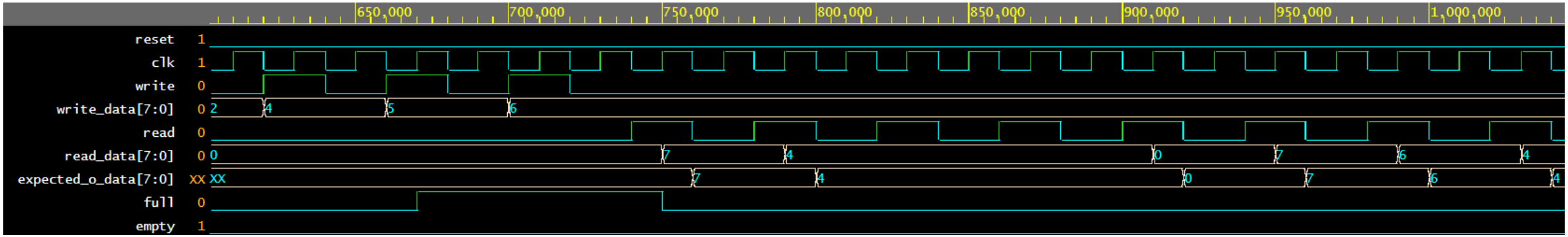
Сброс

```
task reset_task ();
  begin
    reset      = 1'b1;
    write      = 1'b0;
    read       = 1'b0;
    write_data = 0;
    repeat (2)  @ (posedge clk);
    reset = 1'b0;
  end
endtask
```

Чтение

```
task read_fifo ();
  begin
    read = 1'b1;
    # clock_period;
    read = 1'b0;
  end
endtask
```

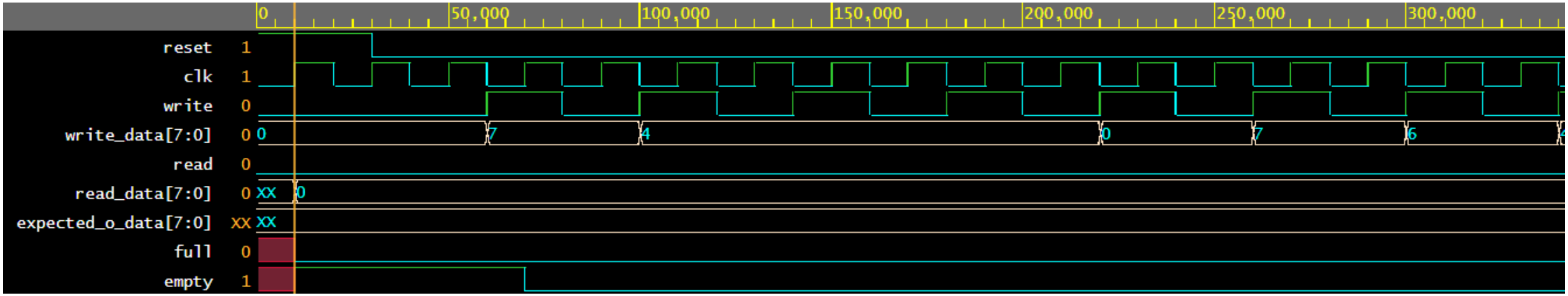
Чтение FIFO


650,000
700,000
750,000
800,000
850,000
900,000
950,000
1,000,000
reset 1
clk 1
write 0
write_data[7:0] 0 2 4 5 6
read 0
read_data[7:0] 0 0 7 4 0 7 6 4
expected_o_data[7:0] XX XX 7 4 0 7 6 4
full 0
empty 1

Запись

```
task write_fifo ([7:0] data);
  begin
    write = 1'b1;
    write_data = data;
    # clock_period write = 1'b0;
  end
endtask
```

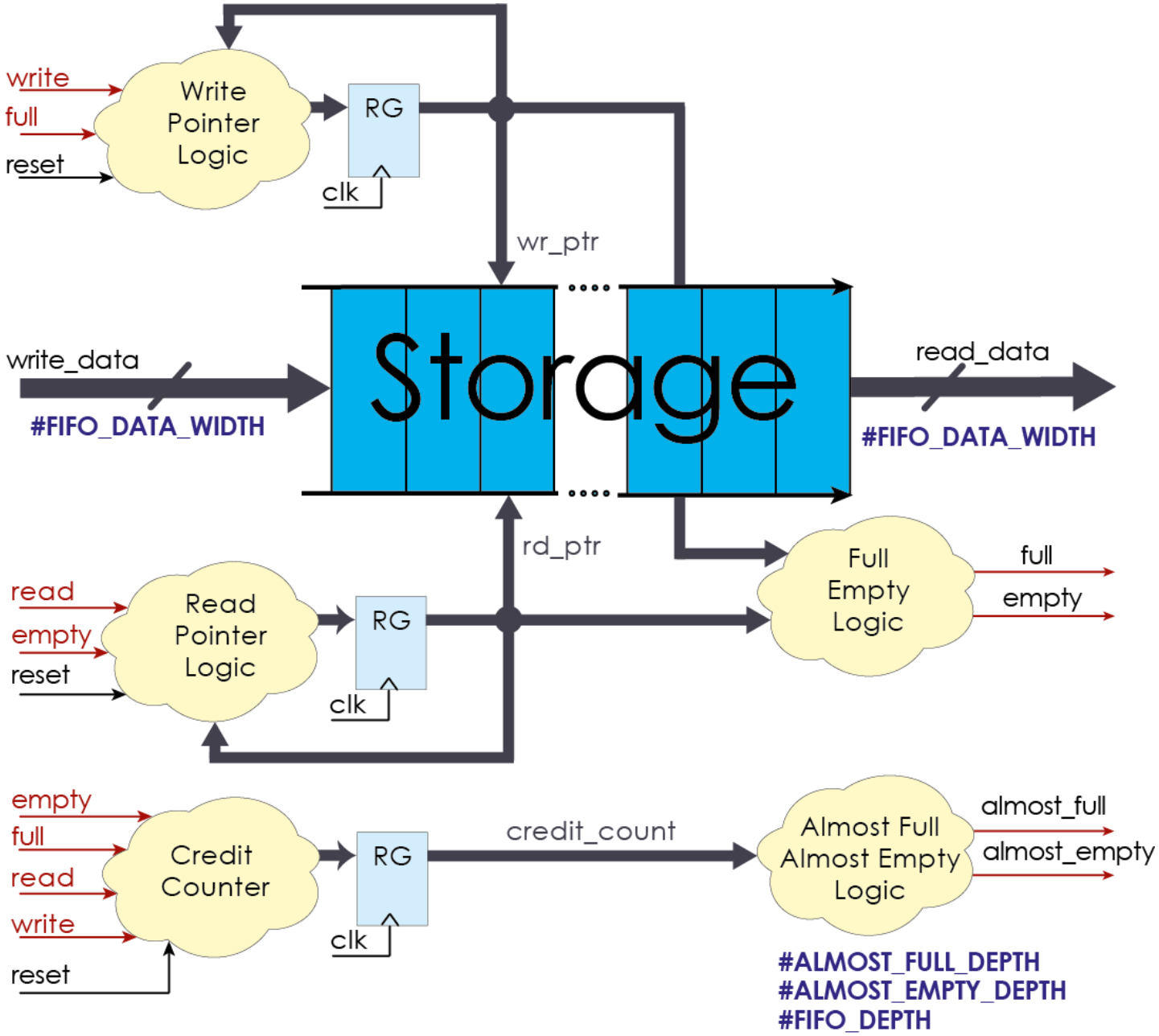
Запись в FIFO



```
initial
  begin
    reset_task ();
    # (clock_period * 1.5);
    // Write to FIFO
    for (i = 0; i <10; i = i + 1)
    begin
      write_fifo (i);
      # clock_period;
    end
    // Read from FIFO
    repeat (10)
      begin
        read_fifo ();
        # clock_period;
      end
```

```
  // Write to FIFO
    for (i = 0; i <10; i = i + 1)
    begin
      write_fifo (i+16);
      # clock_period;
    end
    // Read from FIFO
    repeat (10)
      begin
        read_fifo ();
        # clock_period;
      end
    # clock_period;
    $finish;
end
```

Добавим дополнительные сигналы



Порты

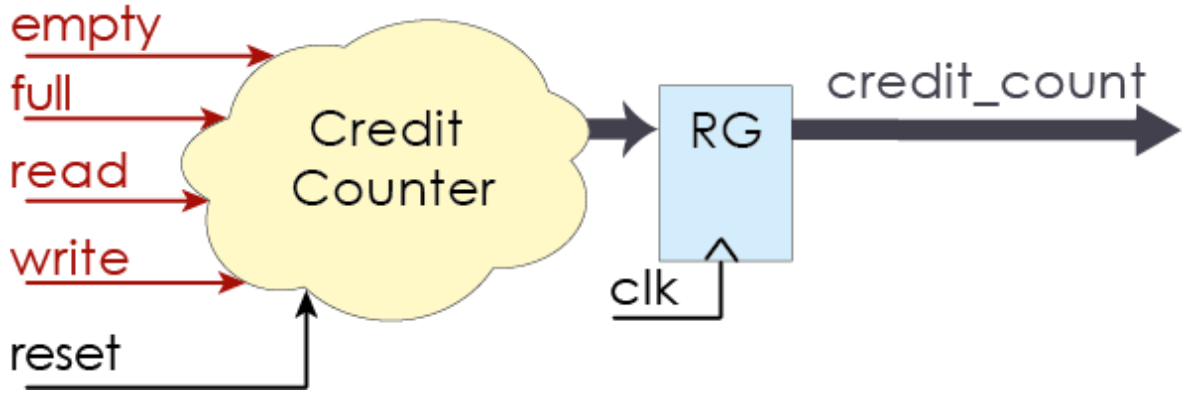
Порт	Разрядность	Направление	Описание
almost_full	1	output	FIFO почти полон
almost_empty	1	output	FIFO почти пуст

Параметры

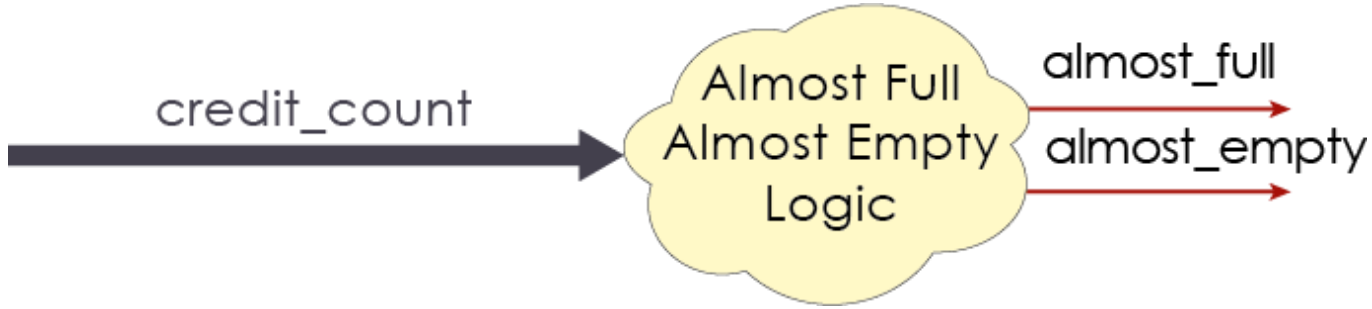
Параметр	Назначение
ALMOST_FULL_DEPTH	Количество ячеек до заполнения FIFO
ALMOST_EMPTY_DEPTH	Количество ячеек до опустошения FIFO

```
module fifo_generic
#(
  parameter   FIFO_DEPTH         = 8,
              FIFO_DATA_WIDTH    = 8,
              ALMOST_FULL_DEPTH  = 2,
              ALMOST_EMPTY_DEPTH = 2)
( input                                clk,
  input                                reset,
  input                                write,
  input                                read,
  input       [FIFO_DATA_WIDTH-1:0]    write_data,
  output reg  [FIFO_DATA_WIDTH-1:0]    read_data,
  output                               empty,
  output                               full,
  output                               almost_empty,
  output                               almost_full);
```

```
localparam FIFO_PTR_WIDTH   = $clog2(FIFO_DEPTH) + 1;
localparam ALMOST_FULL_VALUE = FIFO_DEPTH - ALMOST_FULL_DEPTH;

reg [FIFO_DATA_WIDTH-1:0] fifo_array [FIFO_DEPTH-1:0];

reg [FIFO_PTR_WIDTH-1:0]  rd_ptr;
reg [FIFO_PTR_WIDTH-1:0]  wr_ptr;
reg [FIFO_PTR_WIDTH-1:0]  operation_count;
```

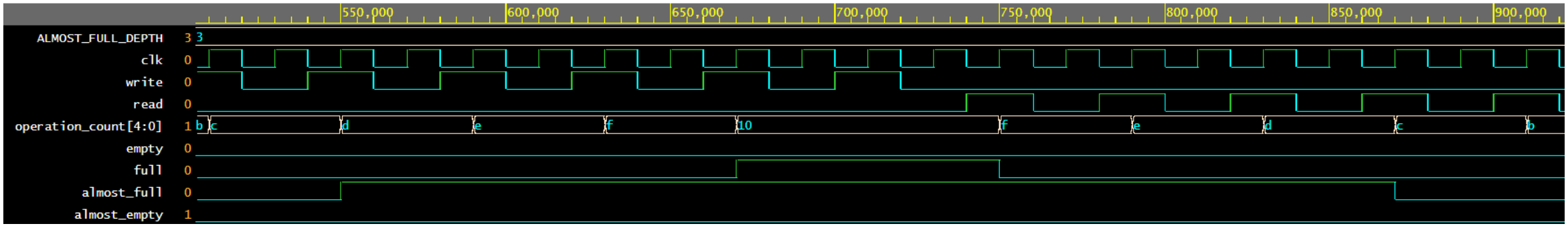



```
// Credit Counter
always @ (posedge clk)
begin
  if (reset)
    operation_count <= {FIFO_PTR_WIDTH{1'b0}};
  else if (write & read & empty)
    operation_count <= operation_count;
    else if (write & !full)
      operation_count <= operation_count + 1'b1;
      else if (read & !empty)
        operation_count <= operation_count - 1'b1;
end
```

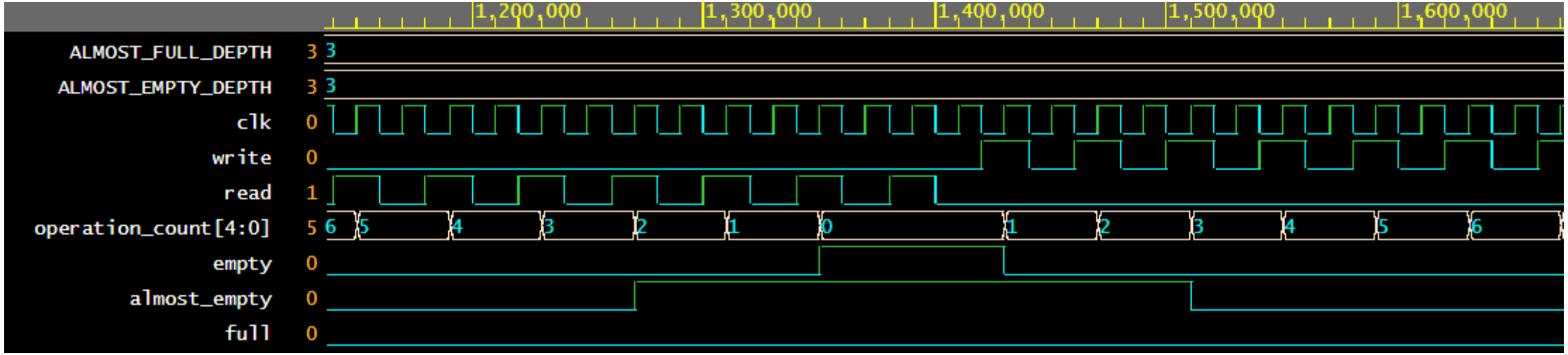



```
parameter  ALMOST_FULL_DEPTH  = 2;
parameter ALMOST_EMPTY_DEPTH = 2;

// Almost Full and Almost Empty flags
assign almost_full  = (credit_count < (FIFO_DEPTH - ALMOST_FULL_DEPTH))
                        ? 1'b0 : 1'b1;
assign almost_empty = (credit_count < ALMOST_EMPTY_DEPTH) ? 1'b1 : 1'b0;
```

Формирование флагов full, almost_full



Формирование флагов empty, almost_empty



Testbench. Queue (1)

- // Queue for storage FIFO data
- logic [FIFO_DATA_WIDTH-1:0] queue_FIFO [$];
- logic [FIFO_DATA_WIDTH-1:0] expected_o_data;

Testbench. Queue (2)

```
always @ (posedge clk)
begin
  if (write & !full)
     queue_FIFO.push_back (write_data);
end
always @ (negedge clk)
begin
  if (read & !empty)
  begin
    expected_o_data = repeat(1) @(negedge clk) queue_FIFO [0];
    if (i_fifo.read_data != expected_o_data)
      $error ("Data mismatch: read_data %h != expected_o_data %h",
        i_fifo.read_data, expected_o_data);
  end
end
```

Testbench. Queue (3)

- always @ (posedge clk)
- begin
- if (read & !empty)
- void' (queue_FIFO.pop_front ());
- end

Testbench. Assertion (1)

```
// Read pointer empty check
property p_rd_stable;
  @(posedge clk) disable iff(reset)
  read && empty |=> (i_fifo.rd_ptr == $past(i_fifo.rd_ptr));
endproperty:p_rd_stable

a_p_rd_stable: assert property (p_rd_stable)
    $warning("Read when empty: ", $time);
  else $error("Incorrect read when empty : ", $time);
```

Testbench. Assertion (2)

```
// Empty set
property p_empty_set;
  @(posedge clk) read && (!write) && (i_fifo.operation_count == 1) |=> empty;
endproperty: p_empty_set

a_p_empty_set: assert property (p_empty_set)
  else $error("Empty set failed ", $time);

// Empty reset
property p_empty_reset;
  @(posedge clk) empty && write |=> !empty;
endproperty: p_empty_reset
```

Testbench. Coverage (1)

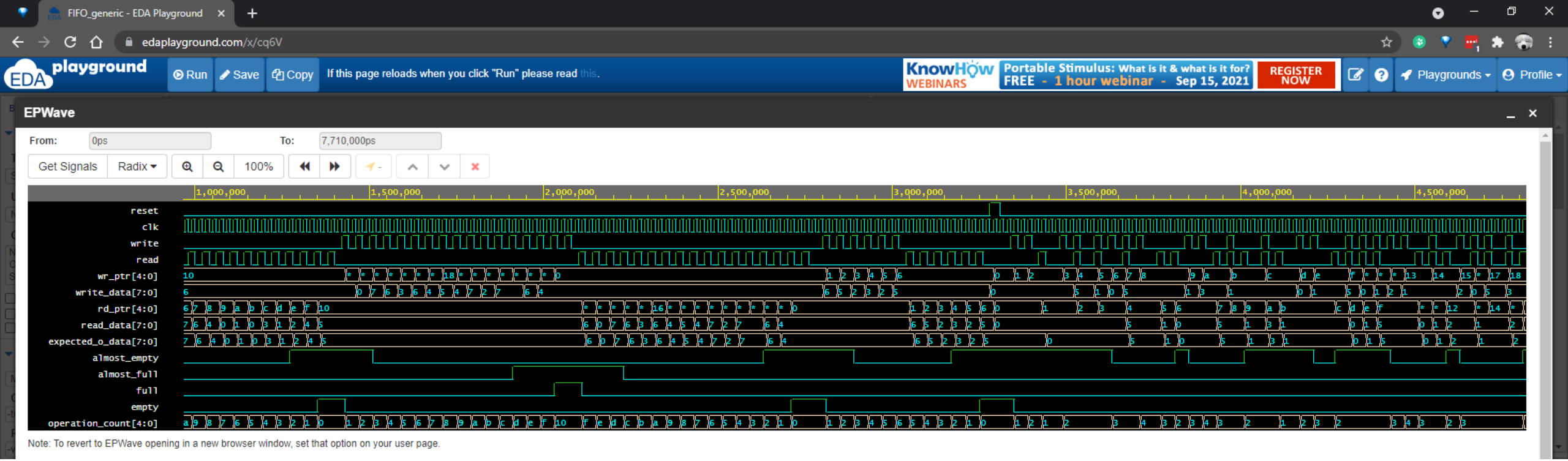
```
logic [1:0] mode;                          // covergroup
  covergroup cg_rw @(posedge clk);         // mode - FIFO mode
    option.at_least = FIFO_DEPTH * 2;      // mode = {read, write}
    coverpoint mode {
      bins mode_none   = {2'b00};
      bins mode_wr     = {2'b01};
      bins mode_rd     = {2'b10};
      bins mode_wr_rd  = {2'b11};
    }
  endgroup
  cg_rw cg = new;
```

Testbench. Coverage (2)

```
while (cg.get_coverage () < 100.0)
  begin
    mode = $urandom_range (3, 0);
    casex (mode)
      2'b00  : none_fifo ();
      2'b01  : begin
                 write_fifo ($urandom_range (FIFO_DATA_WIDTH-1, 0));
                 # clock_period;
               end
```

Testbench. Coverage (3)

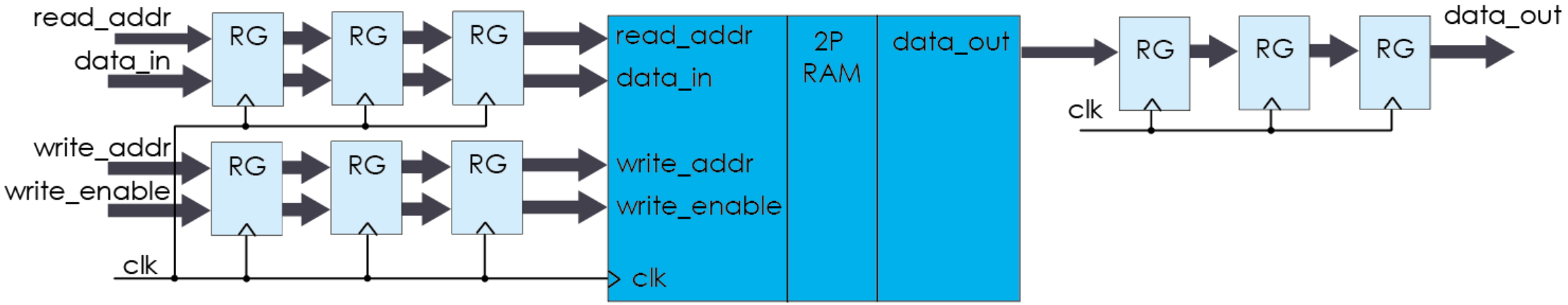
```
2'b10  : begin
                read_fifo ();
                # clock_period;
              end
      2'b11  : begin
                read_write_fifo ($urandom_range (FIFO_DATA_WIDTH-1,
0));
                # clock_period;
              end
      default: none_fifo ();
    endcase
  end;
  $display("Coverage = %0.2f %%", cg.get_inst_coverage ());
```



- https://www.edaplayground.com/x/cq6V

FIFO на памяти с латентностью

Память с латентностью

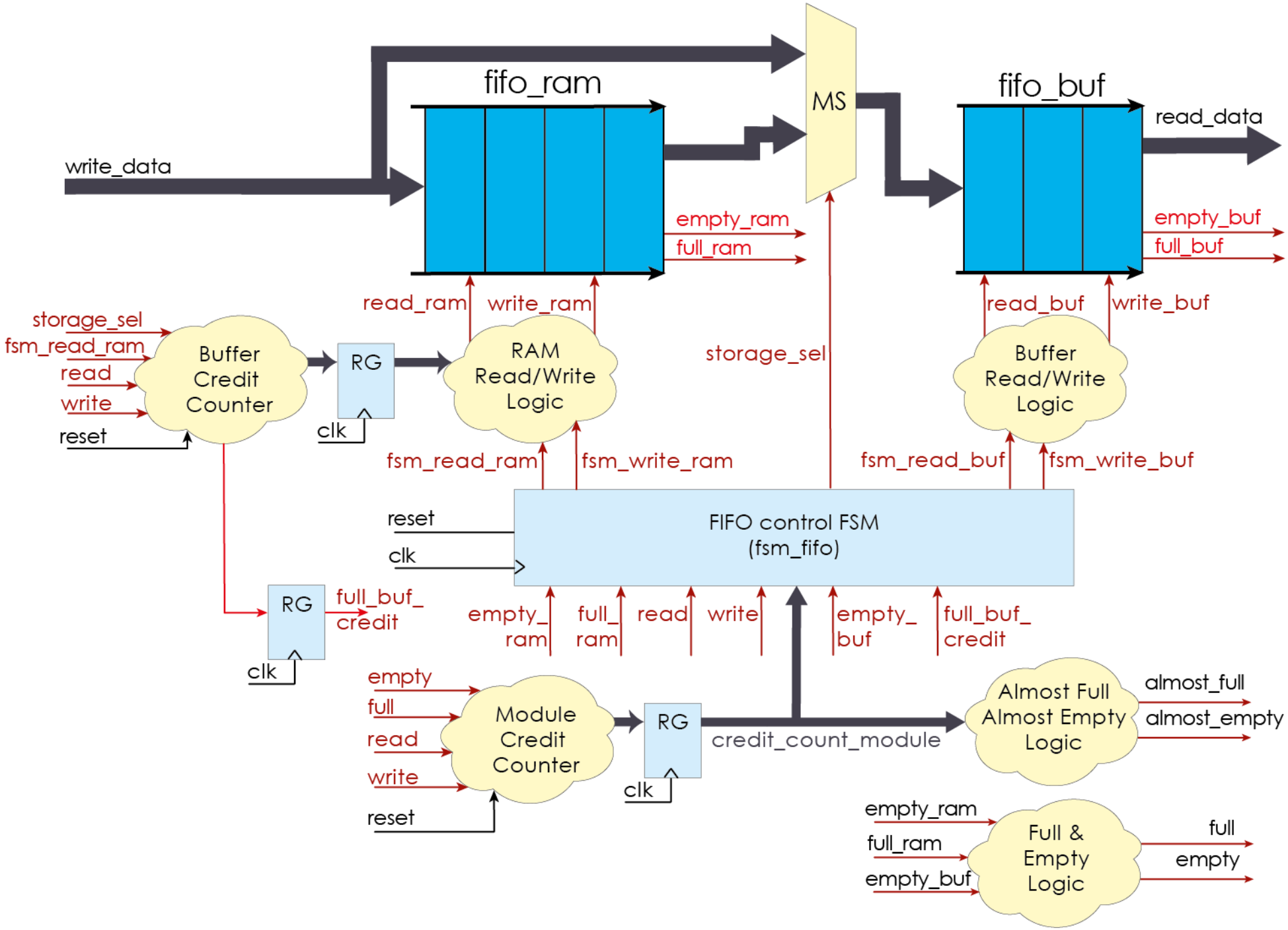


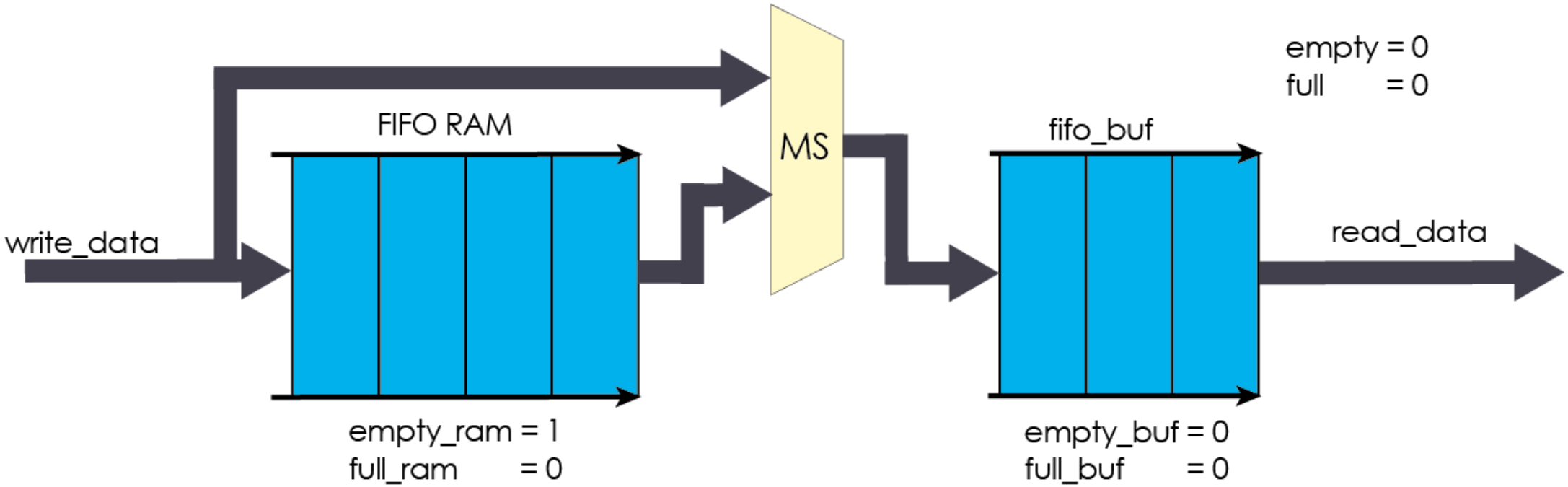
FIFO на модуле с латентностью

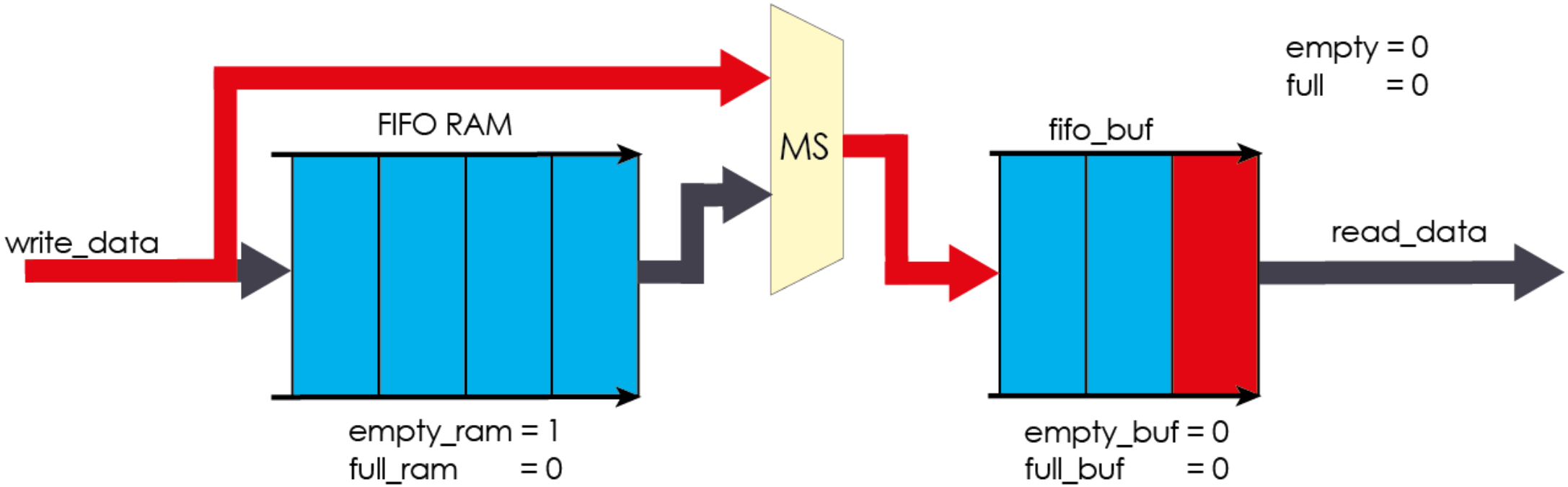
- Задержка от чтения или записи равна латентности
- Проблемы при интенсивном обмене
- Проблемы при работе с полным модулем (full = 1).

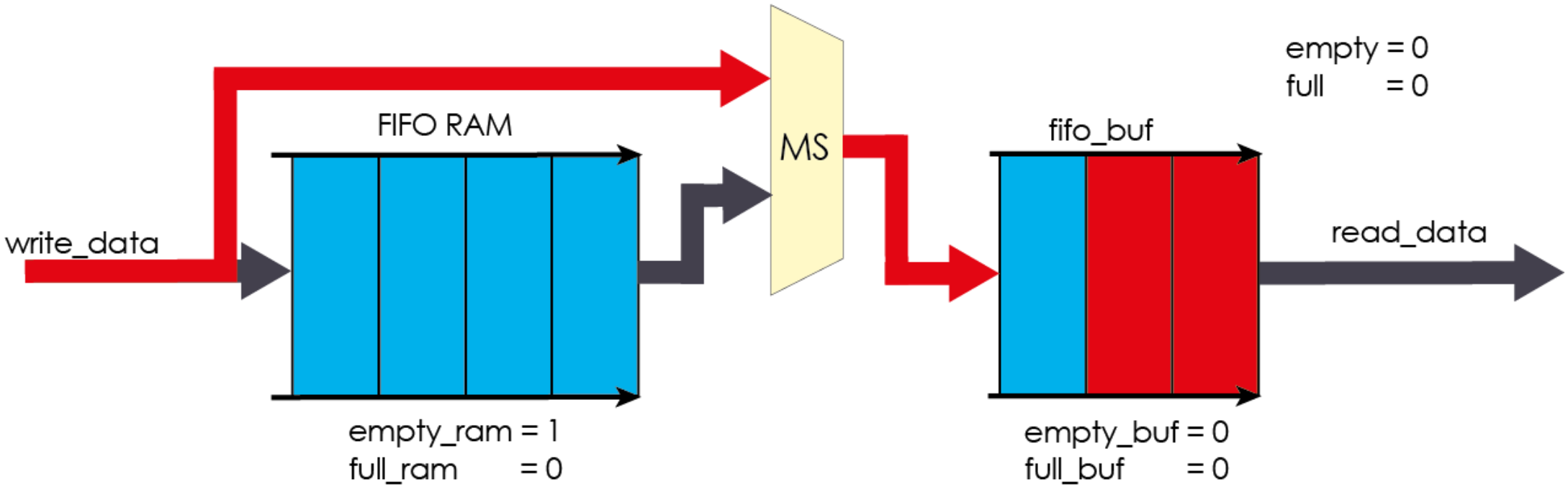
FIFO с буфером для устранения латентности

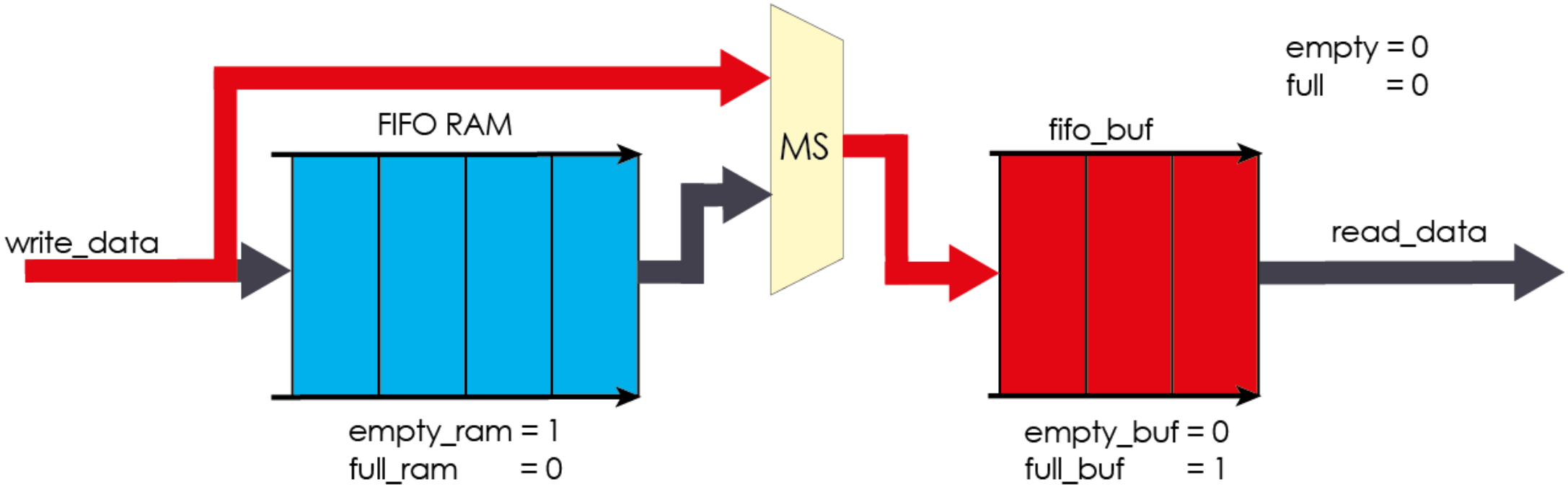
- Два модуля FIFO:
  - FIFO основанное на модуле памяти с латентностью.
  - FIFO основанное на регистровом файле. Длина должна быть больше латентности памяти + 2. В нашем примере мы берем число равное степени 2 от латентности + 1

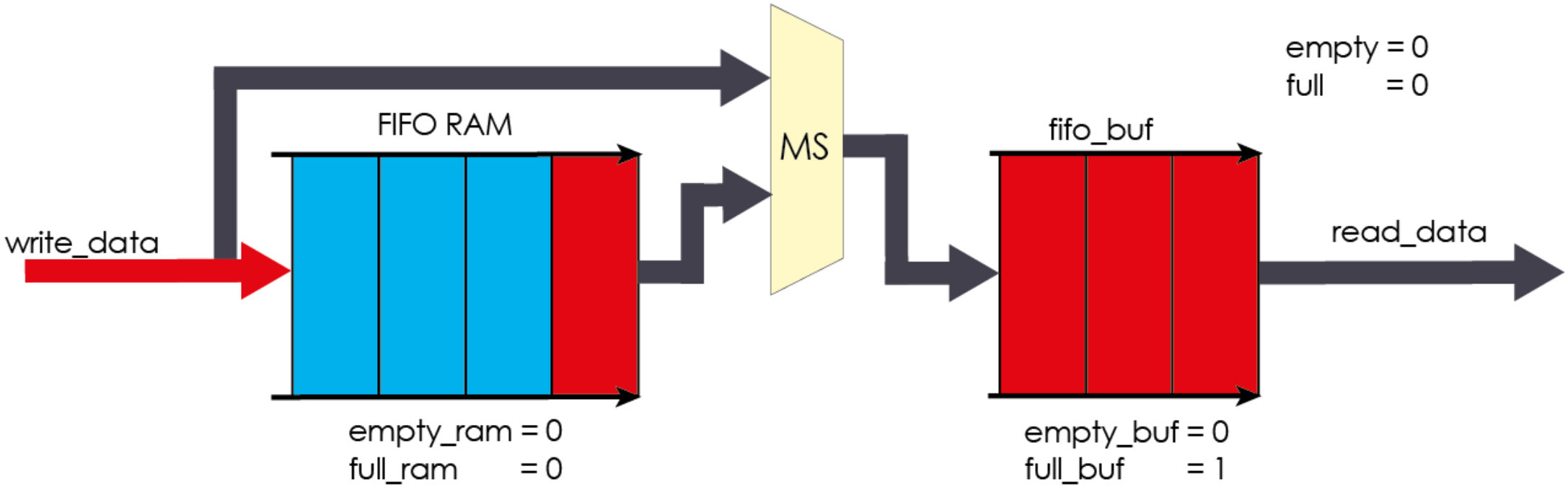

fifo_ram
MS
fifo_buf
write_data
read_data
empty_ram
full_ram
empty_buf
full_buf
read_ram
write_ram
read_buf
write_buf
storage_sel
fsm_read_ram
read
write
reset
Buffer Credit Counter
RG
clk
RAM Read/Write Logic
storage_sel
Buffer Read/Write Logic
fsm_read_ram
fsm_write_ram
fsm_read_buf
fsm_write_buf
reset
clk
FIFO control FSM (fsm_fifo)
RG
full_buf_ credit
clk
empty_ ram
full_ ram
read
write
empty_ buf
full_buf_ credit
empty
full
read
write
reset
Module Credit Counter
RG
clk
credit_count_module
Almost Full Almost Empty Logic
almost_full
almost_empty
empty_ram
full_ram
empty_buf
Full & Empty Logic
full
empty

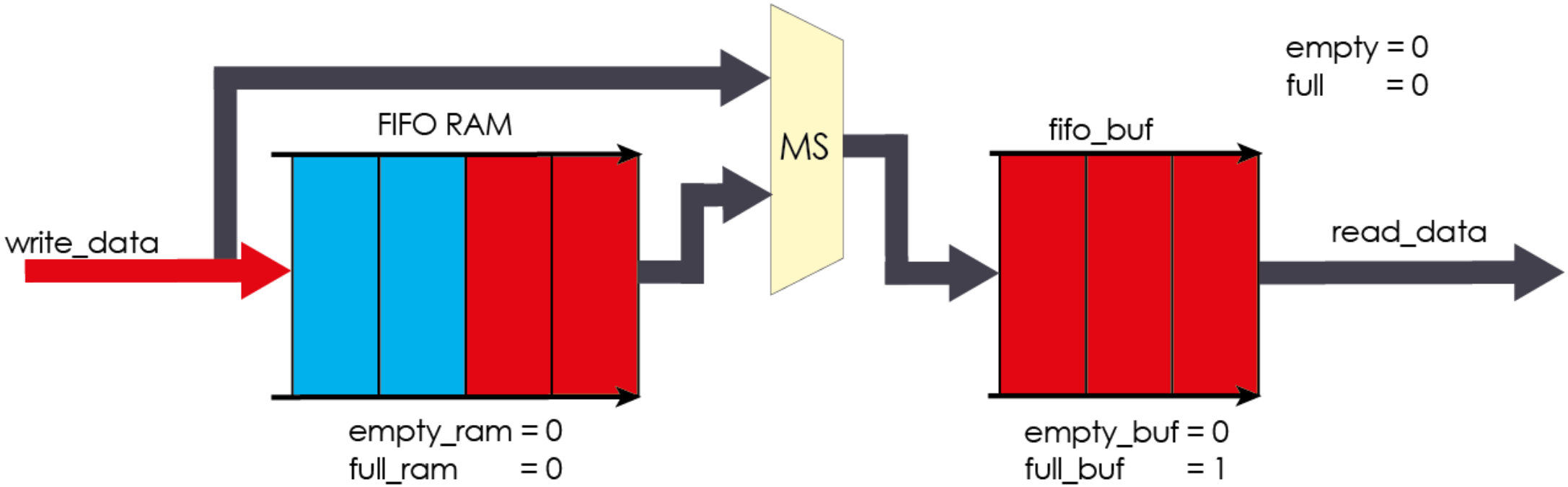

empty = 0
full = 0
FIFO RAM
MS
fifo_buf
write_data
read_data
empty_ram = 1
full_ram = 0
empty_buf = 0
full_buf = 0

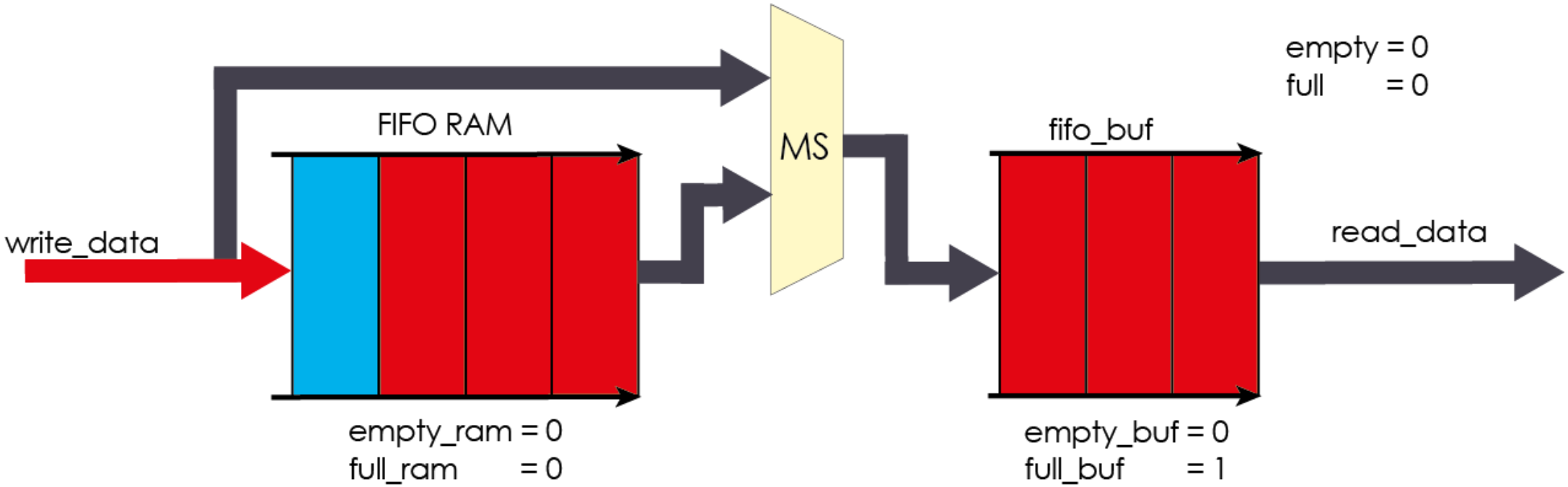

empty = 0
full = 0
FIFO RAM
MS
fifo_buf
write_data
read_data
empty_ram = 1
full_ram = 0
empty_buf = 0
full_buf = 0

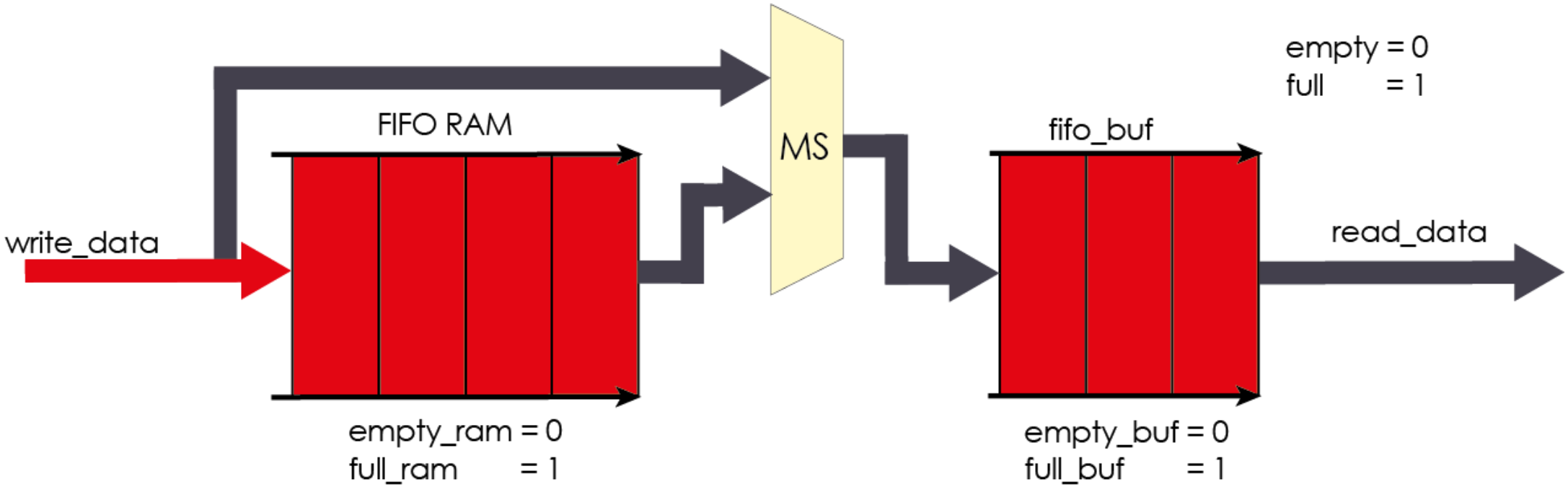

empty = 0
full = 0
FIFO RAM
MS
fifo_buf
write_data
read_data
empty_ram = 1
full_ram = 0
empty_buf = 0
full_buf = 0

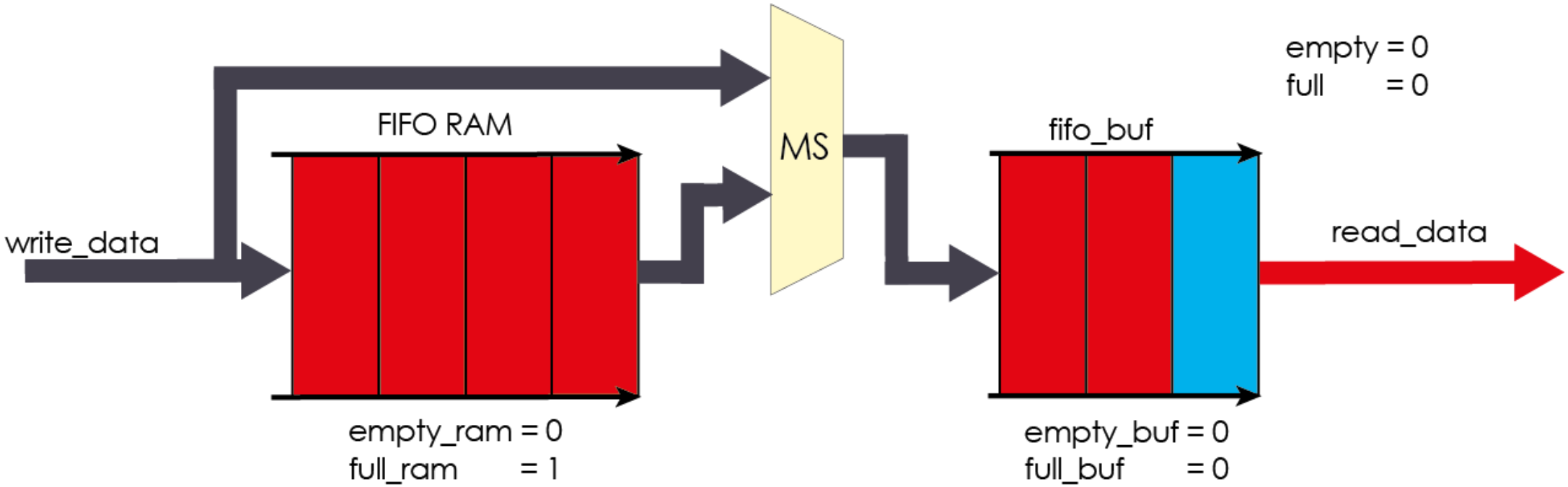

empty = 0
full = 0
FIFO RAM
MS
fifo_buf
write_data
read_data
empty_ram = 1
full_ram = 0
empty_buf = 0
full_buf = 1

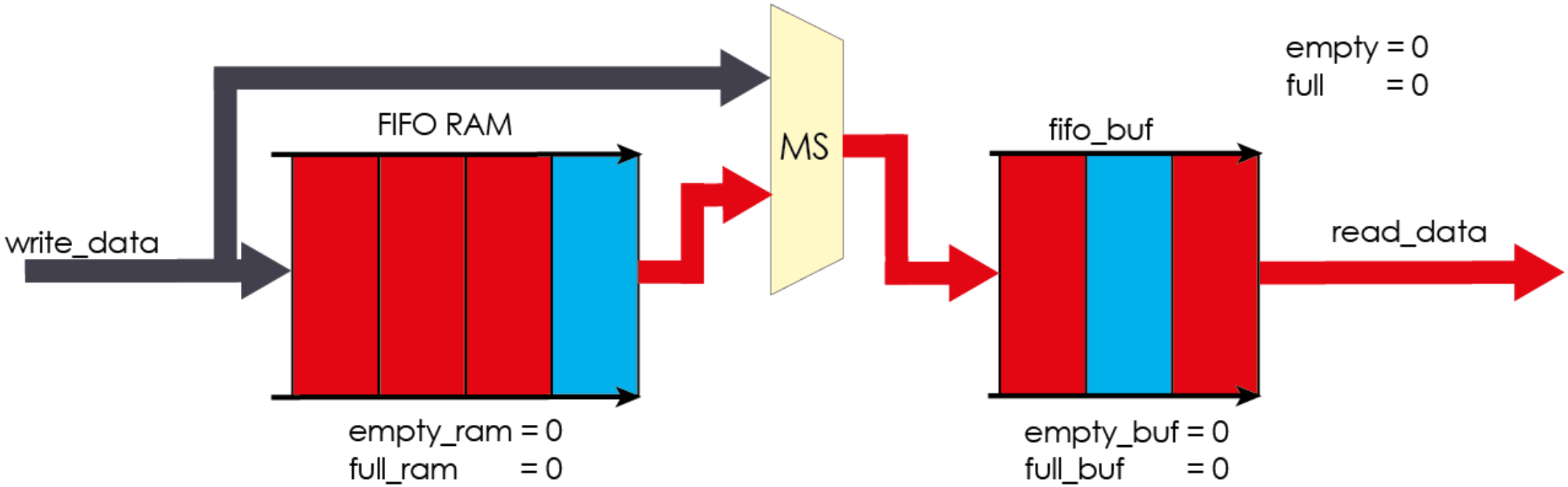

empty = 0
full = 0
FIFO RAM
MS
fifo_buf
write_data
read_data
empty_ram = 0
full_ram = 0
empty_buf = 0
full_buf = 1

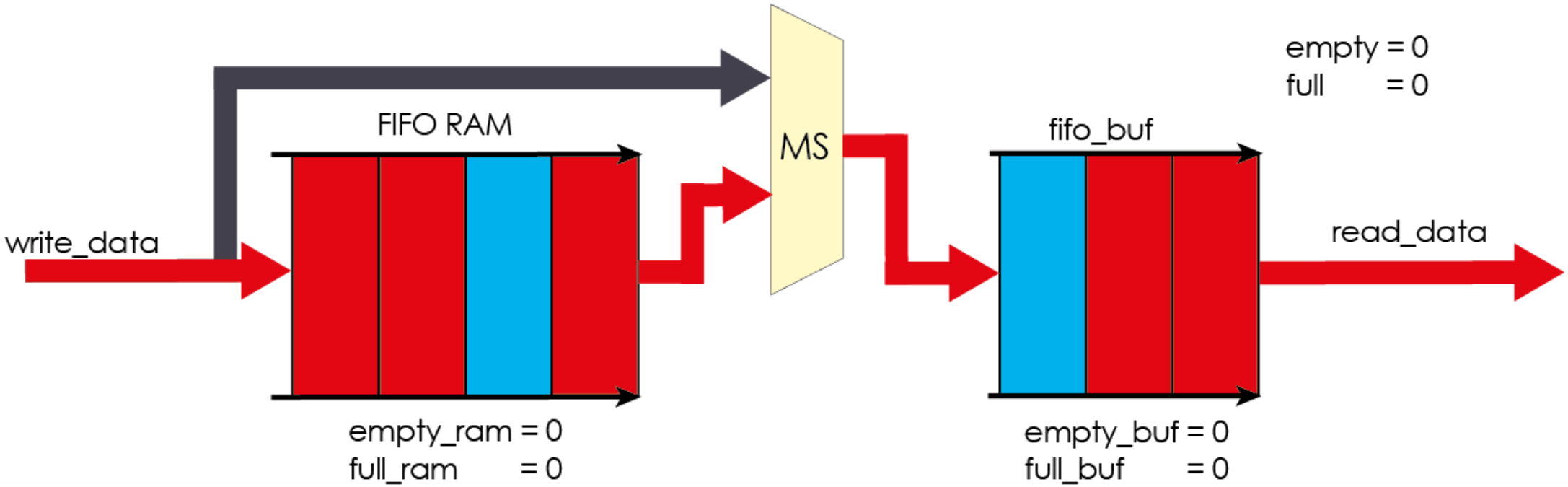

empty = 0
full = 0
FIFO RAM
MS
fifo_buf
write_data
read_data
empty_ram = 0
full_ram = 0
empty_buf = 0
full_buf = 1


empty = 0
full = 0
FIFO RAM
MS
fifo_buf
write_data
read_data
empty_ram = 0
full_ram = 0
empty_buf = 0
full_buf = 1

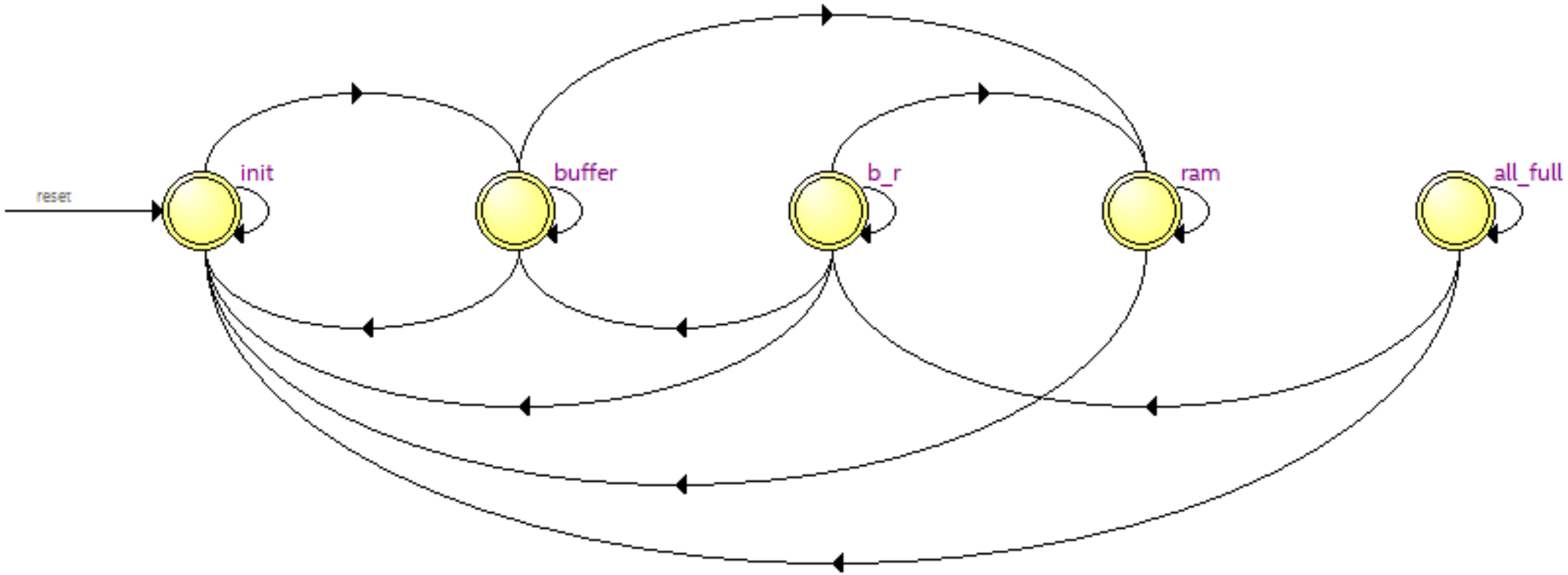

empty = 0
full = 1
FIFO RAM
MS
fifo_buf
write_data
read_data
empty_ram = 0
full_ram = 1
empty_buf = 0
full_buf = 1


empty = 0
full = 0
FIFO RAM
MS
fifo_buf
write_data
read_data
empty_ram = 0
full_ram = 1
empty_buf = 0
full_buf = 0


write_data
FIFO RAM
MS
fifo_buf
empty = 0
full = 0
read_data
empty_ram = 0
full_ram = 0
empty_buf = 0
full_buf = 0


empty = 0
full = 0
FIFO RAM
MS
fifo_buf
write_data
read_data
empty_ram = 0
full_ram = 0
empty_buf = 0
full_buf = 0

Конечный автомат управления режимами



Состояния конечного автомата

- init – состояние после сброса или полностью пустого FIFO
- buffer – запись входных данных и чтение выходных данных только из буферного FIFO.
- ram – запись входных данных в FIFO на памяти и чтение выходных данных из буферного FIFO. Записи в буферное FIFO нет.
- b_r – работа обоих FIFO:
  - Входные данные пишутся в FIFO на памяти.
  - Выходные данные читаются из буферного FIFO.
  - Чтение FIFO на памяти и запись этих данных в буферное FIFO
- all_full – полностью заполенный модуль. Нет записи. Чтение выходных данных из буферного FIFO

Кредитные счетчики

- Кредитный счетчик модуля (credit_count_module) – расчет заполненности всего модуля
- Кредитный счетчик буфера (credit_count_buffer) – расчет заполенности буфера и управление записью в память

Кредитный счетчик для модуля

read	write	full	empty	Credit_counter_module
1	0	X	0	- 1
0	1	0	X	+ 1
1	1	0	1	+ 1
1	1	1	0	- 1
1	1	0	0	nop
0	0	X	X	nop

Кредитный счетчик буфера

- Учитывает сигналы записи и чтения всего модуля, сигнал чтения FIFO на памяти, состояние конечного автомата.
- Режимы работы:
  - Буфер пуст
  - Буфер полон
  - Буфер не пуст и не полон

FIFO_buffer - EDA Playground

edaplayground.com/x/ZPTn

EDA playground | Run | Save | Copy | If this page reloads when you click "Run" please read this.

Playgrounds | Profile

Brought to you by DOULOS

Languages & Libraries

Testbench + Design

SystemVerilog/Verilog

UVM / OVM

None

Other Libraries

None
OVL 2.8.1
SVUnit 2.11

- [ ] Enable TL-Verilog
- [ ] Enable Easier UVM
- [ ] Enable VUnit

Tools & Simulators

Mentor Questa 2020.1

Compile Options

-timescale 1ns/1ns

Run Options

-voptargs=+acc=npr

Run Time: 10 ms

- [ ] Use **run.do** Tcl file
- [ ] Use **run.bash** shell script
- [x] Open **EPWave** after run
- [ ] Download files after run

Examples

Community

Collaborate

Forum

Follow @edaplayground

testbench.sv

SV/Verilog Testbench

```
module tb
  #( parameter test_id = 1
   )
  ();


  string    test_name[3:0]=
  {
   "randomize",
   "direct_low_write_high_read",
   "direct_high_write_low_read",
   "direct_base"
  };


  localparam FIFO_DEPTH         = 32;
  localparam FIFO_DATA_WIDTH    = 8;
  localparam ALMOST_FULL_DEPTH  = 3;
  localparam ALMOST_EMPTY_DEPTH = 3;
  localparam LATENCY            = 3;// min value 2
  localparam BUFFER_DEPTH       = 8;


logic                       reset_p;    //! 1 - reset
logic                       clk=0;        //! clock
logic [FIFO_DATA_WIDTH-1:0] data_i=0;
logic                       data_we=0;
logic [FIFO_DATA_WIDTH-1:0] data_o;
logic                       data_rd=0;
logic                       full;
logic                       empty;
logic                       almost_full;
logic                       almost_empty;

int                     cnt_wr=0;
int                     cnt_rd=0;
int                     cnt_ok=0;
int                     cnt_error=0;

int                     show_ok=0;
int                     show_error=0;
```

design.sv | fifo_reg_file.sv | FIFO_simple_DP_RAM.sv | fsm_fifo.sv | simple_dual_port_RAM.sv | fifo_2p_ram_buffer.sv

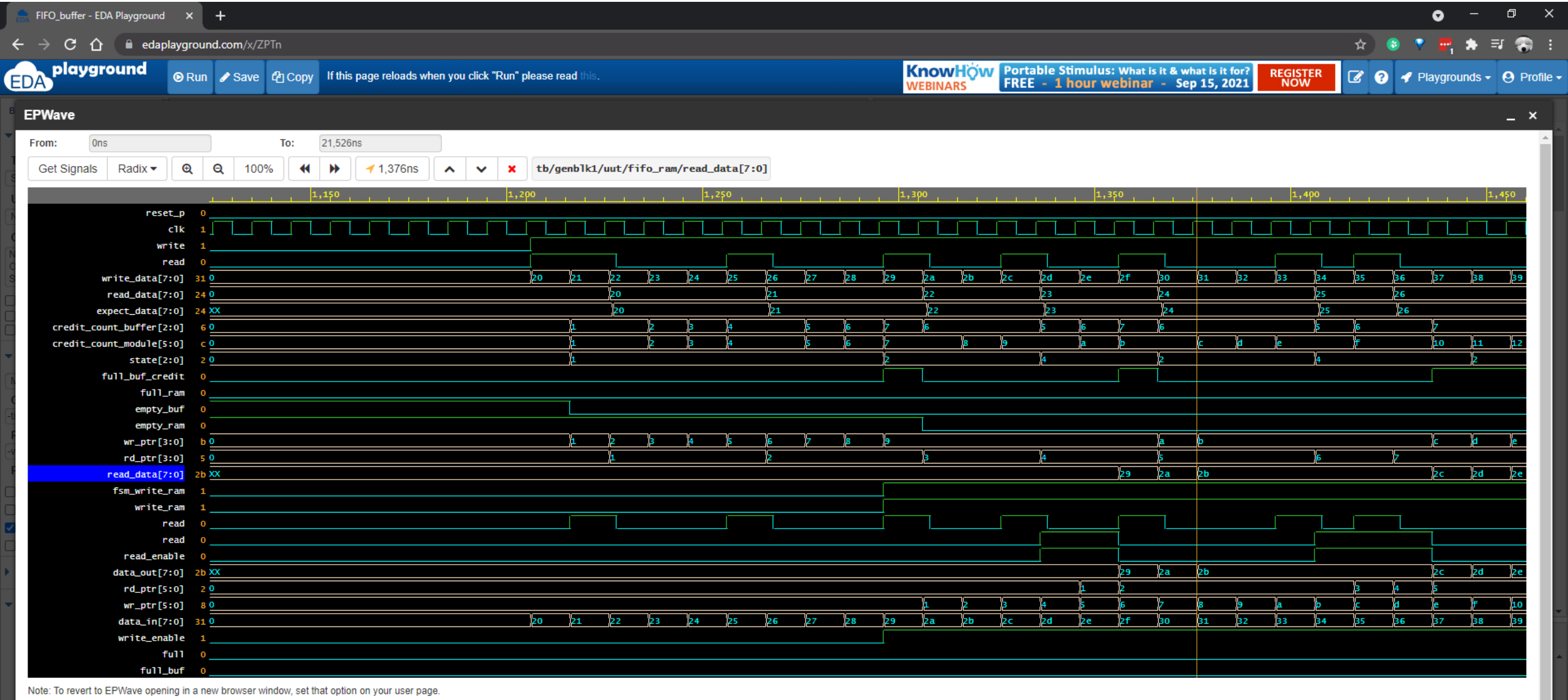
SV/Verilog Design

```
`include "FIFO_simple_DP_RAM.sv"
`include "fsm_fifo.sv"
`include "fifo_reg_file.sv"

`timescale 1 ns / 1 ns
`default_nettype none

module fifo_2p_ram_buffer
#(
  parameter   FIFO_RAM_DEPTH      = 256, // depth for RAM FIFO
              FIFO_DATA_WIDTH     = 8,
              ALMOST_FULL_DEPTH   = 2,
              ALMOST_EMPTY_DEPTH  = 2,
              LATENCY             = 3, // min value 2
              BUFFER_DEPTH        = 8  // min LATENCY + 3
)

(
  input wire                            clk,
  input wire                            reset,

  input wire                            write,
  input wire                            read,

  input wire     [FIFO_DATA_WIDTH-1:0] write_data,
  output wire    [FIFO_DATA_WIDTH-1:0] read_data,

  output wire                           empty,
  output wire                           full,
  output wire                           almost_empty,
  output wire                           almost_full
);

  localparam FIFO_PTR_WIDTH    = $clog2(FIFO_RAM_DEPTH) + 1;
  localparam BUF_CREDIT_COUNT  = $clog2(BUFFER_DEPTH);
  logic [FIFO_PTR_WIDTH-1:0]       credit_count_module;
  logic [BUF_CREDIT_COUNT-1:0]     credit_count_buffer;

  //------------------------------------------------
  // Modules
  //------------------------------------------------
```

Log | Share

```
#
#
# test_id=1     test_name: direct_high_write_low_read         TEST_PASSED
#
#
# ** Note: $finish    : testbench.sv(91)
#    Time: 42065 ns  Iteration: 1  Instance: /tb
# End time: 16:59:44 on Sep 10,2021, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
Finding VCD file...
./dump.vcd
[2021-09-10 16:59:44 EDT] Opening EPWave...
Done
```

EPWave



- https://www.edaplayground.com/x/ZPTn

Другие типы FIFO

- FIFO с несколькими входами записи и их арбитражом
- FIFO с однопортовой, а не двухпортовой памятью.
- Асинхронные FIFO

Спасибо за внимание

Sergey.Ivanets@gmail.com

https://t.me/DigitalDesignSchool

https://www.youtube.com/channel/UCdvZsFLSdbPbHJMOWDsO8Cw