Буферизация (и handshaking)

школа синтеза цифровых схем

ПРИ ПАРТНЕРСТВЕ

YADRO · MP

Syntacore™

Занятие №9

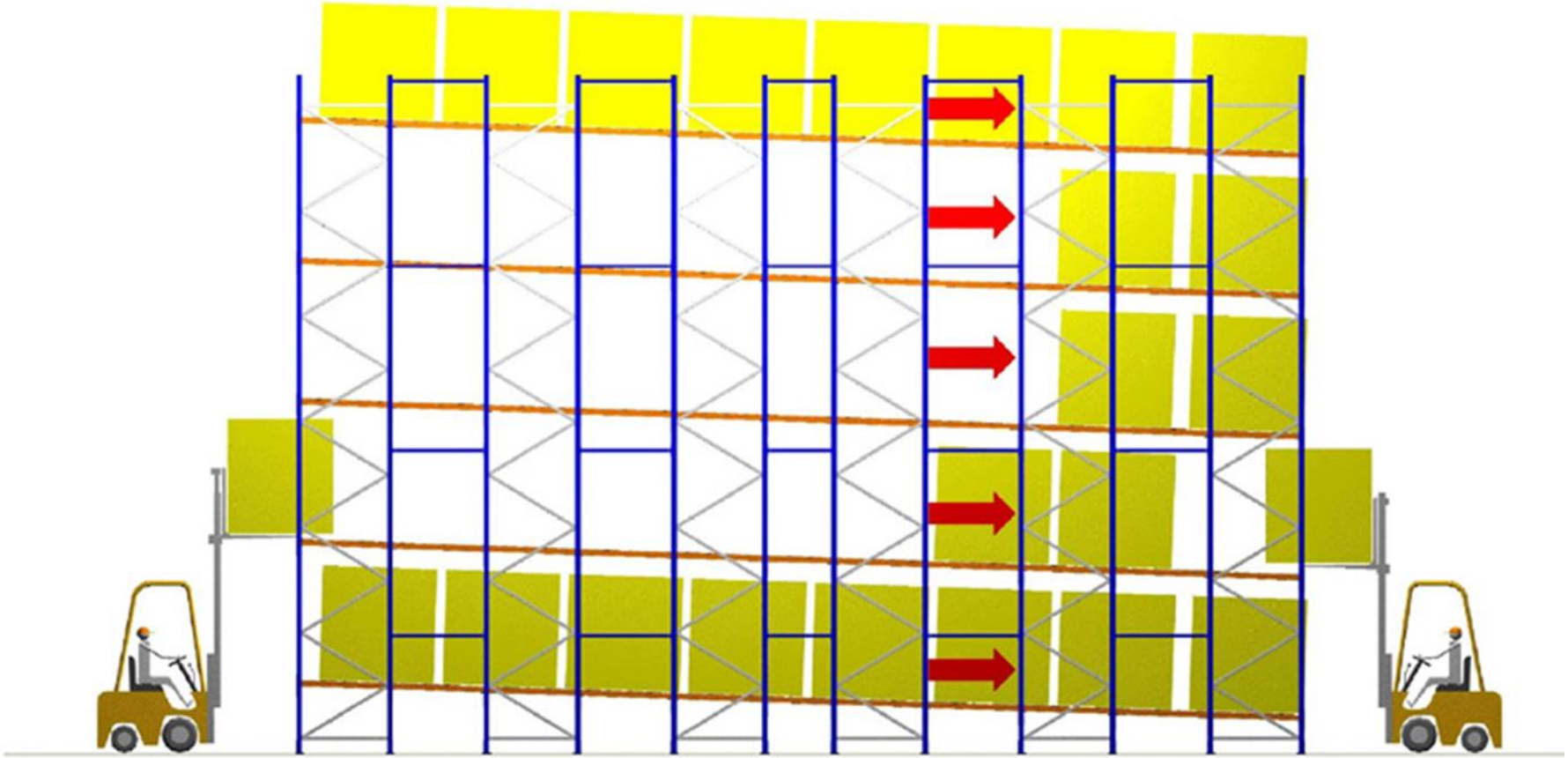
19 ноября 2022

Илья Кудрявцев

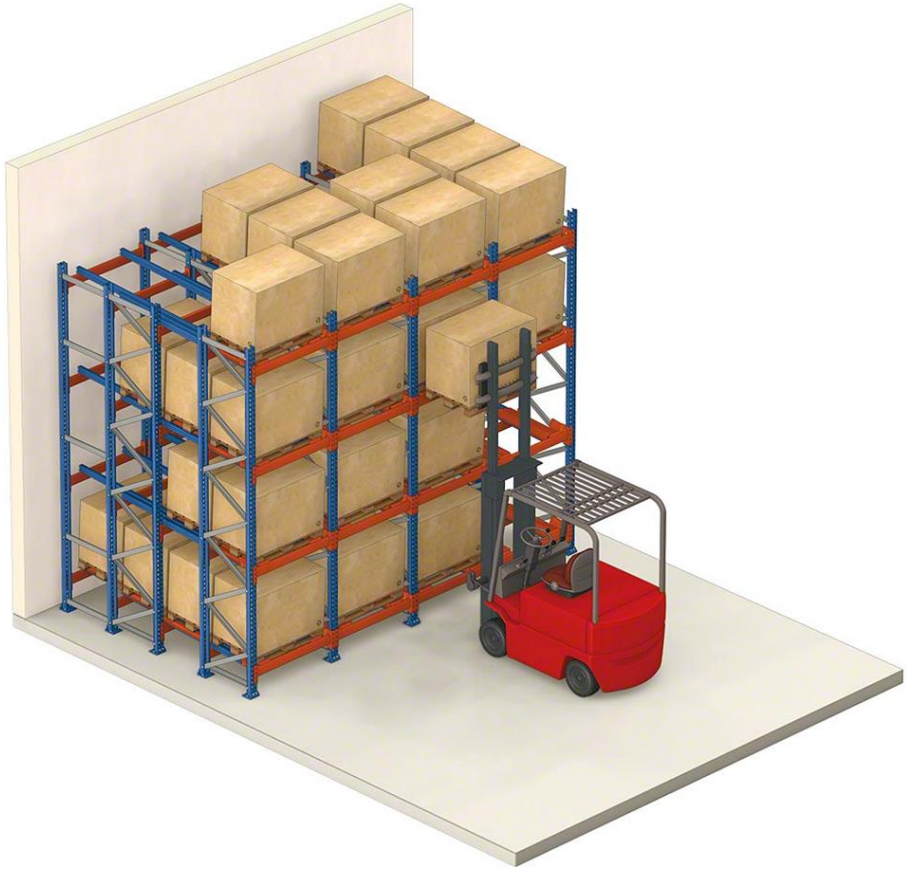
Преподаватель Самарского университета

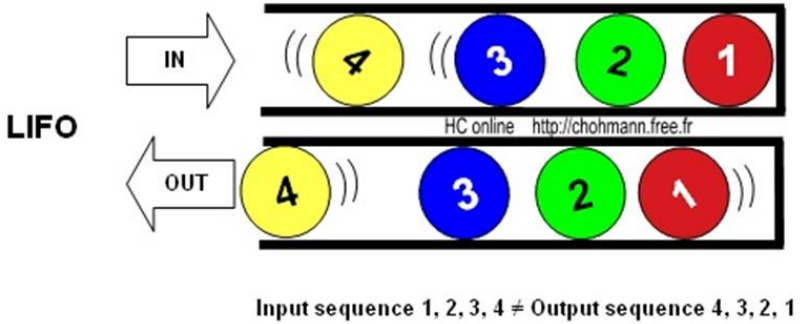
Концепция FIFO

Концепция FIFO



Концепция LIFO

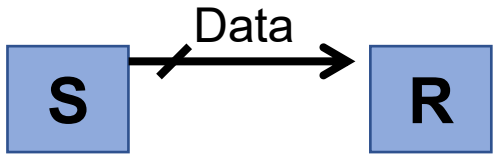




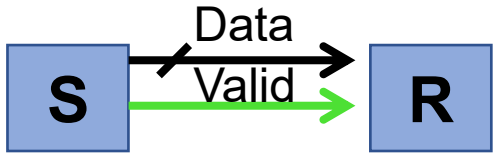
Input sequence 1, 2, 3, 4 ≠ Output sequence 4, 3, 2, 1

Передача данных

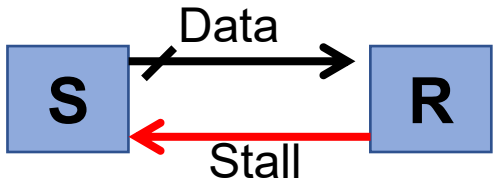
- Синхронная передача
  - Данные передаются в каждом такте



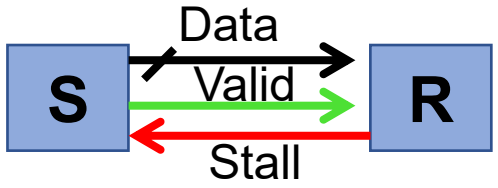
- С приостановкой передатчика
  - Сигнал Data valid



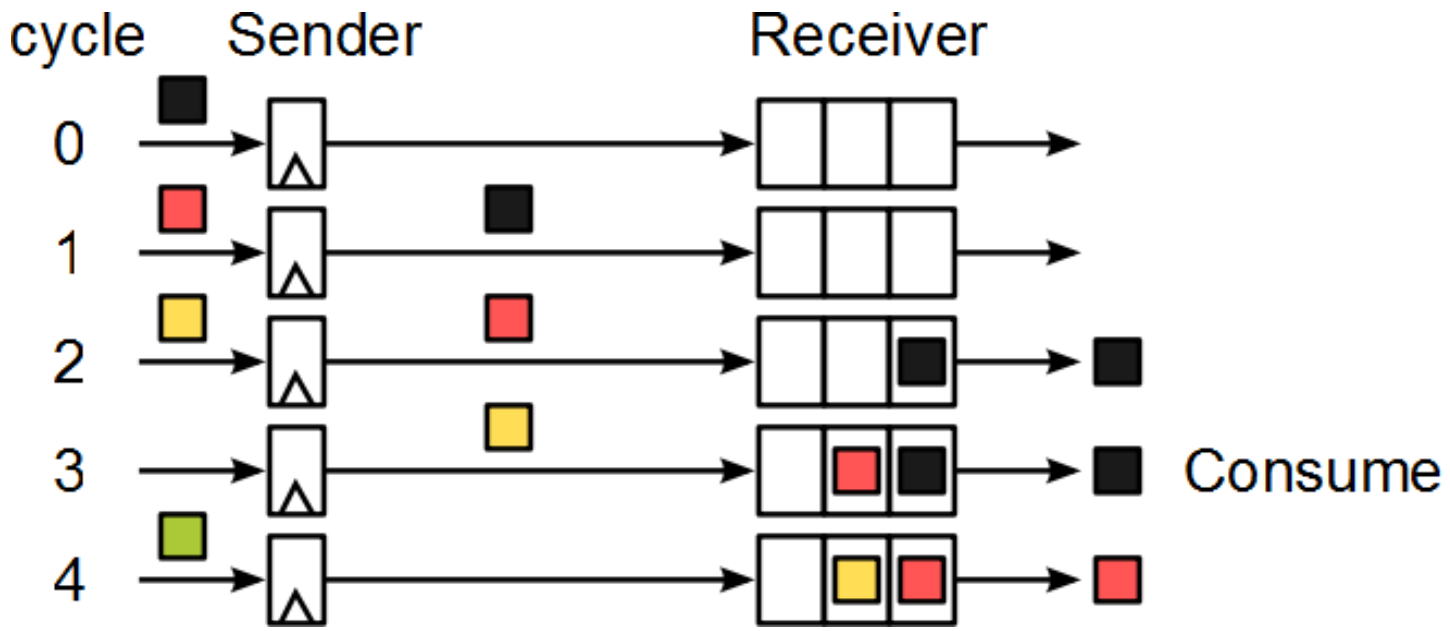
- С приостановкой приёмника
  - Сигнал Stall



- С двойной приостановкой
  - Сигналы Valid и Stall
  - Наличие буфера на стороне приёмника



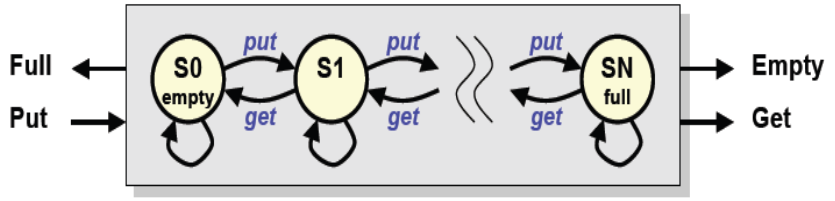
Развязка приёмника и передатчика



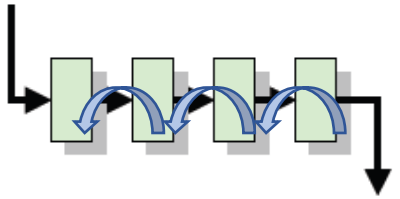
- Приёмник принимает данные, даже если конечный потребитель не готов их принимать
  - Когда остановиться? Как исключить переполнение?

Варианты построения буфера

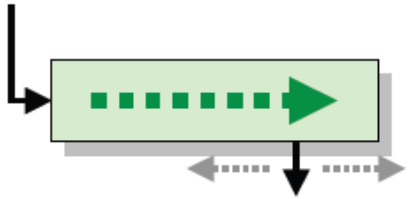
- Контейнер FIFO, сохраняющий порядок передачи
  - 4 сигнала (full, empty, put, get)



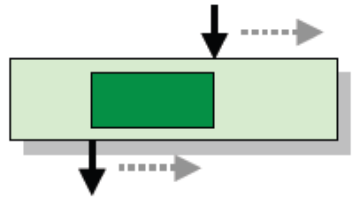
- Эластичный буфер
  - Каскад глубиной depth-1
  - Внутренние сигналы full/empty



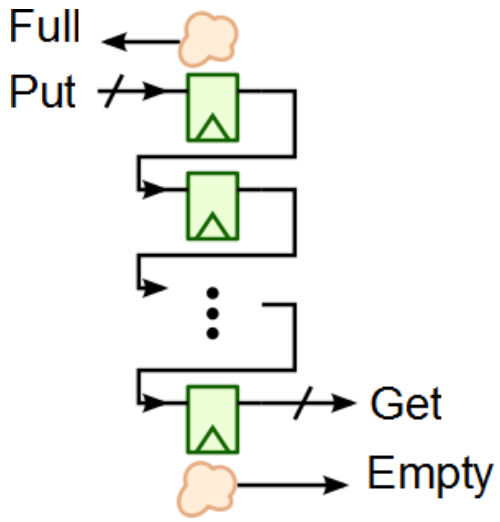
- Сдвиговый регистр с параллельным выходом
  - Put: последовательный ввод
  - Get: Указатель очереди чтения



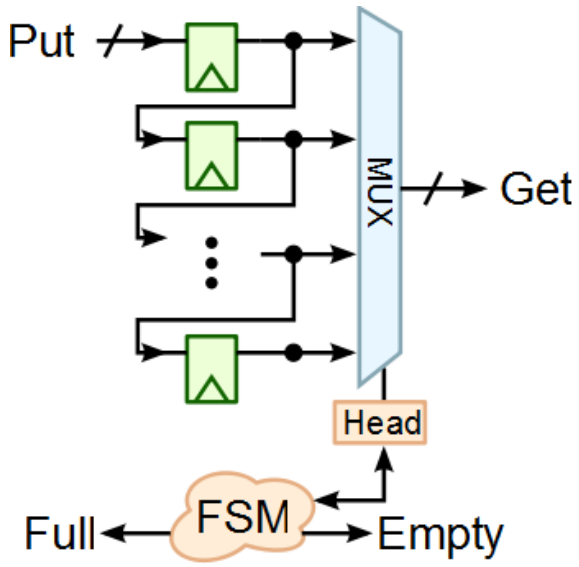
- Кольцевой буфер
  - Указатели чтения/записи
  - Логическая закольцовка
  - На основе RAM или регистров



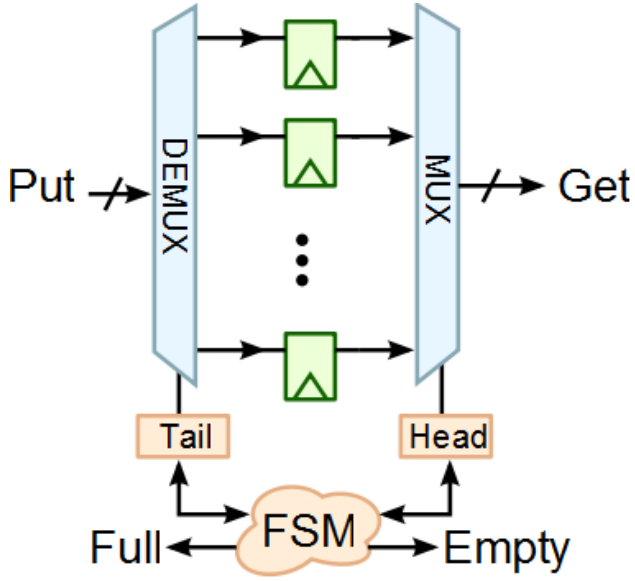
Реализация буферов



Эластичный



На регистрах сдвига



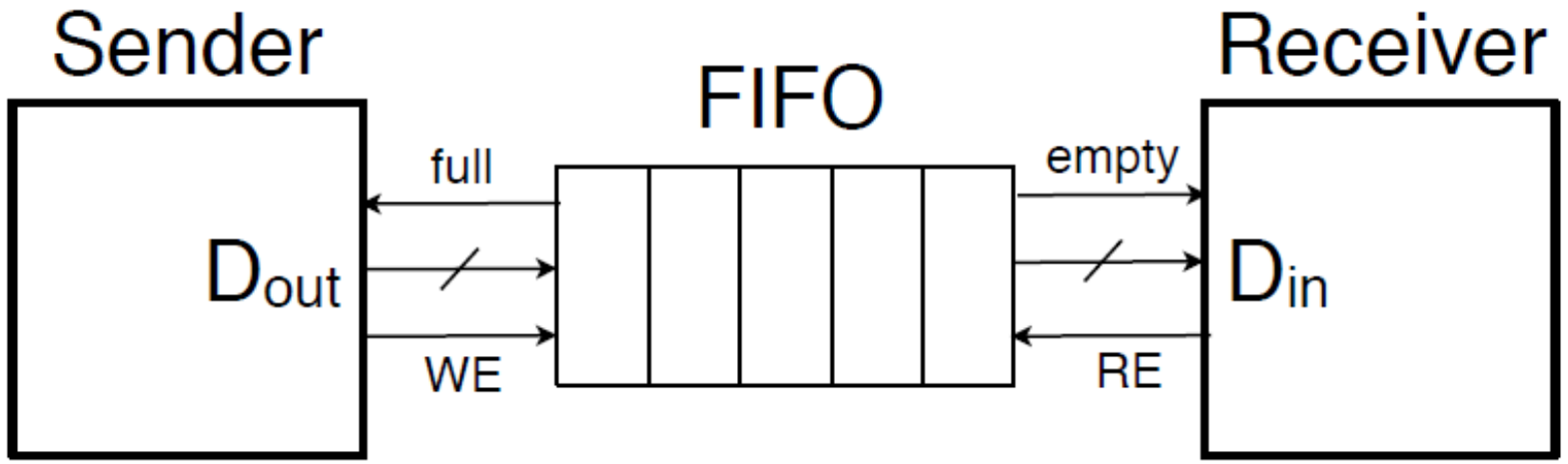
Циркулярный буфер

- Похожая базовая структура
- Мультиплексоры и указатели чтения/записи осуществляют контроль данных

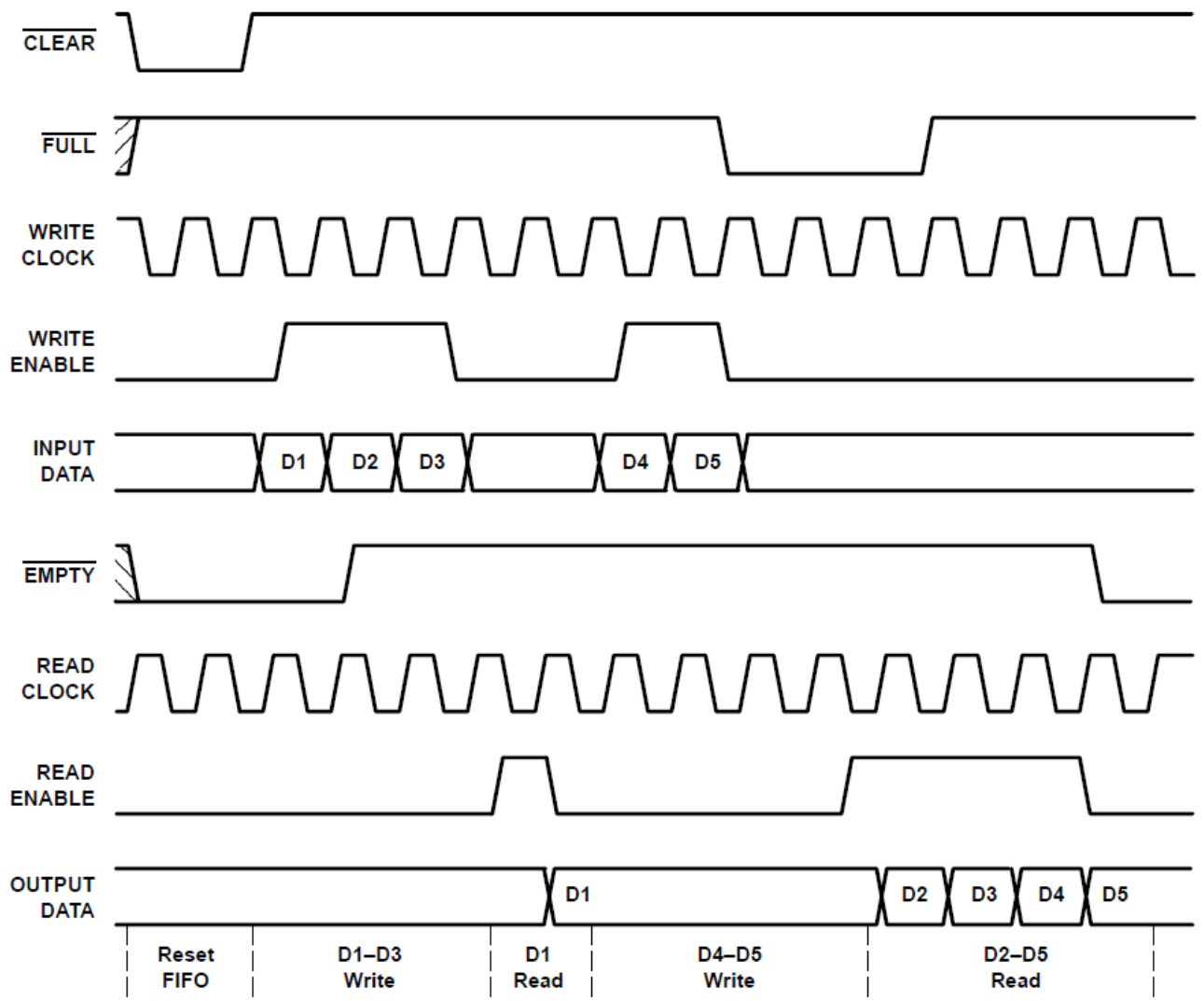
Классификация FIFO (с точки зрения режимов

1. С зависимыми операциями чтения и записи
2. С независимыми операциями чтения записи
   - Синхронные
   - Асинхронные

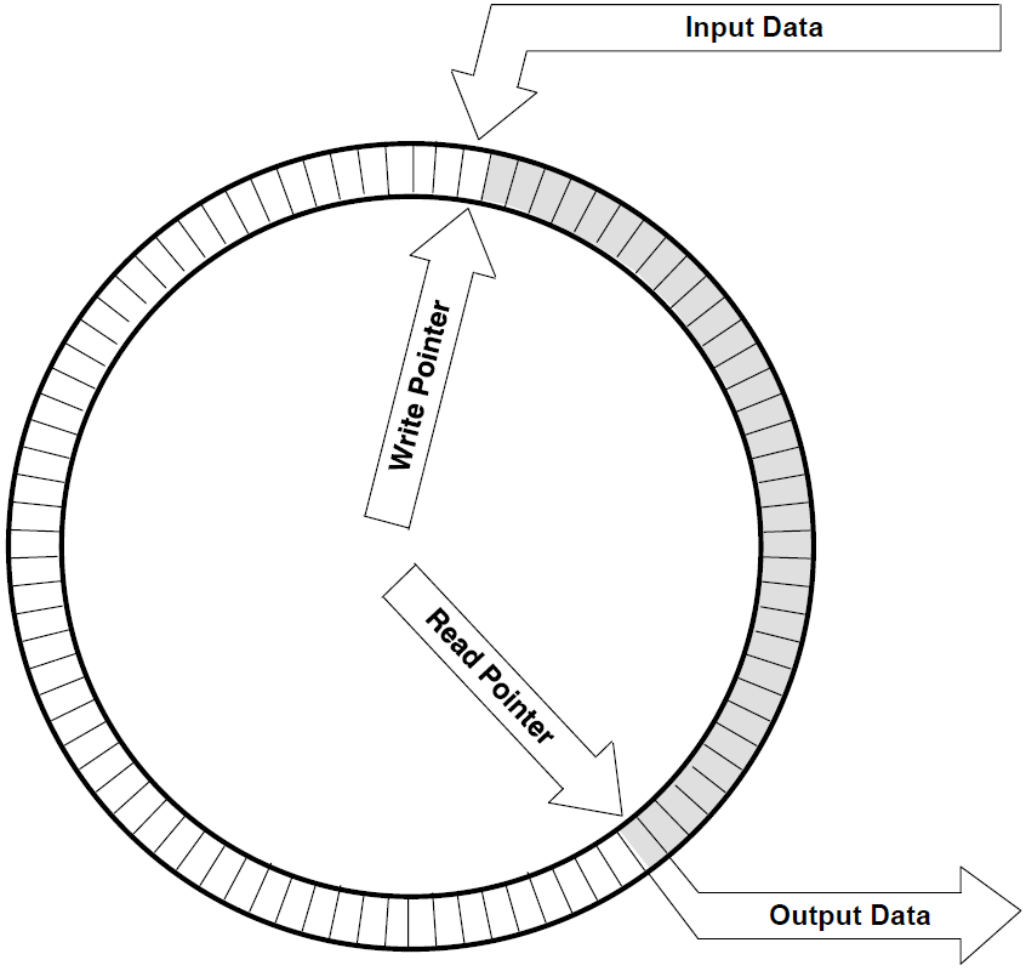
Синхронный FIFO



Синхронный FIFO (размер 4)



FIFO на основе RAM (циркулярный буфер)



Синхронный FIFO

```
module sync_fifo
#(
    parameter data_width    = 4,
    parameter address_width = 4,
    parameter ram_depth     = 16)
(
    output [data_width-1:0] data_out,
    output       full,
    output       empty,
    input [data_width-1:0]  data_in,
    input        clk,
    input        rst_a,
    input        wr_en,
    input        rd_en
);
```

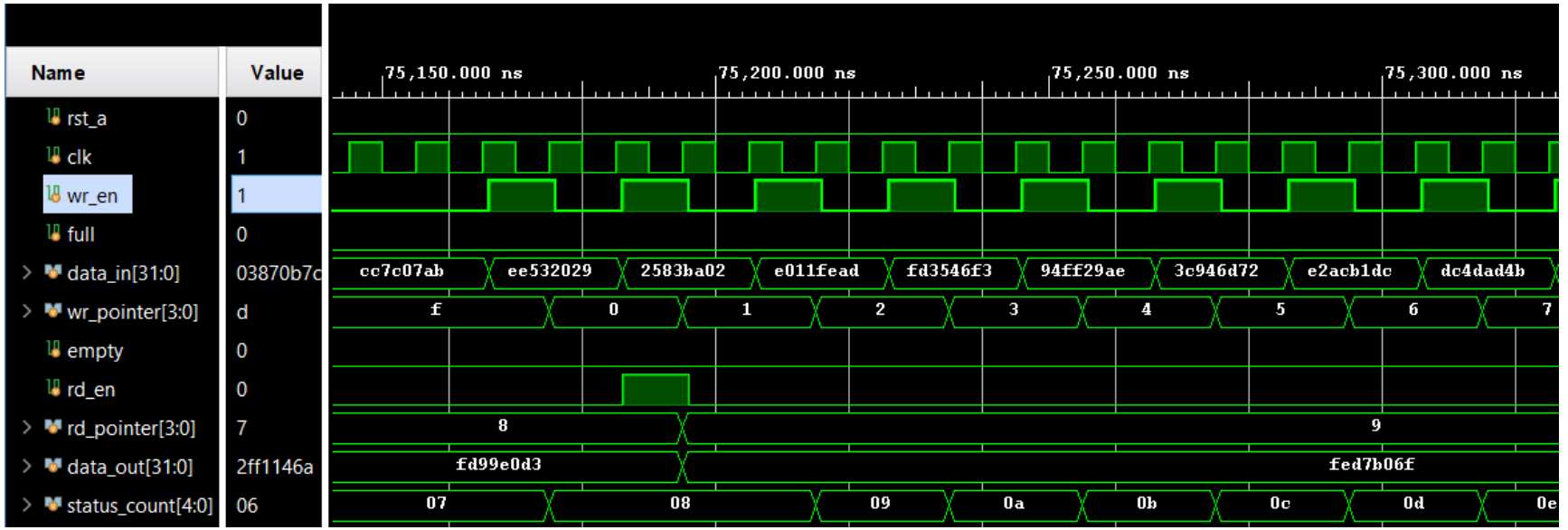
Синхронный FIFO

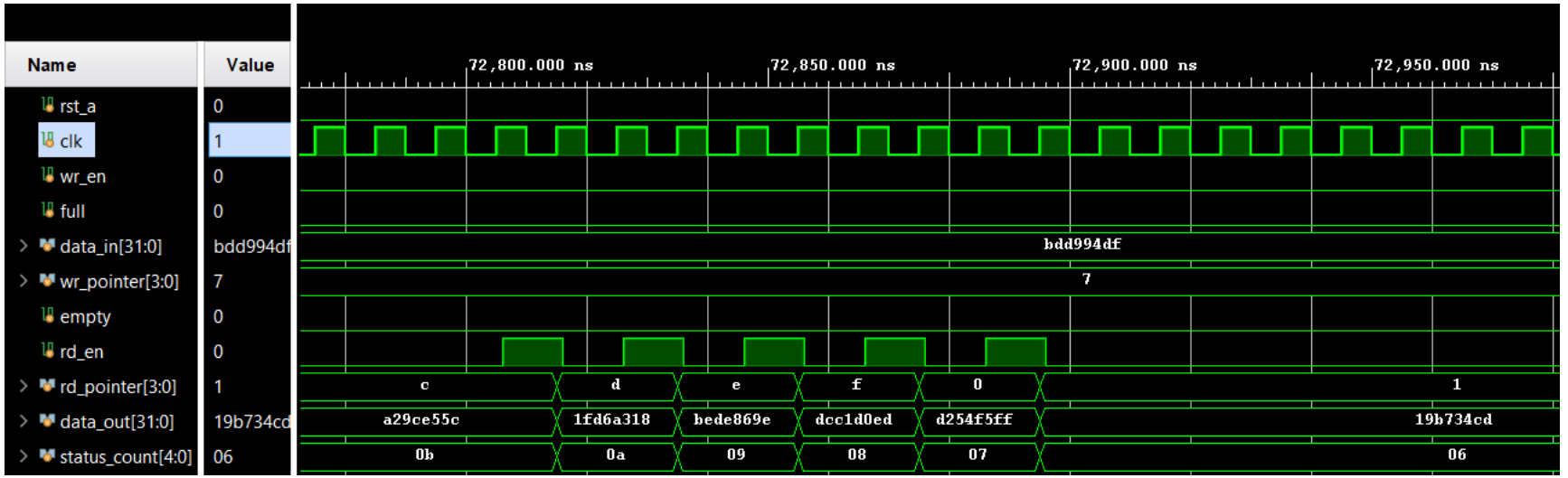
```
//--------------internal register declaration
   logic [address_width-1:0]                    wr_pointer;
   logic [address_width-1:0]                    rd_pointer;
   logic [address_width  :0]                    status_count;
   logic [data_width-1:0]                       data_out;
   logic [ram_depth - 1:0][data_width-1:0]      data_ram ;
//--------------wr_pointer pointing to write address
   always_ff @ (posedge clk, rst_a)
   begin
   if(rst_a) wr_pointer <= 0;
   else if(wr_en) begin
                  assert (full == 0) else $fatal(1,"%t : Overflow of a fifo",$time);
                  data_ram[wr_pointer] <= data_in;
                  wr_pointer <= wr_pointer+1;
                  end
   end
 //--------------read from FIFO
   always_ff @ (posedge clk, rst_a)
   begin
   if(rst_a) rd_pointer <= 0;
   else if(rd_en) begin
                  assert (empty == 0) else $fatal(1,"%t : Underflow of a fifo",$time);
                  data_out <= data_ram[rd_pointer];
                  rd_pointer <= rd_pointer+1;
                  end
   end
```

Синхронный FIFO

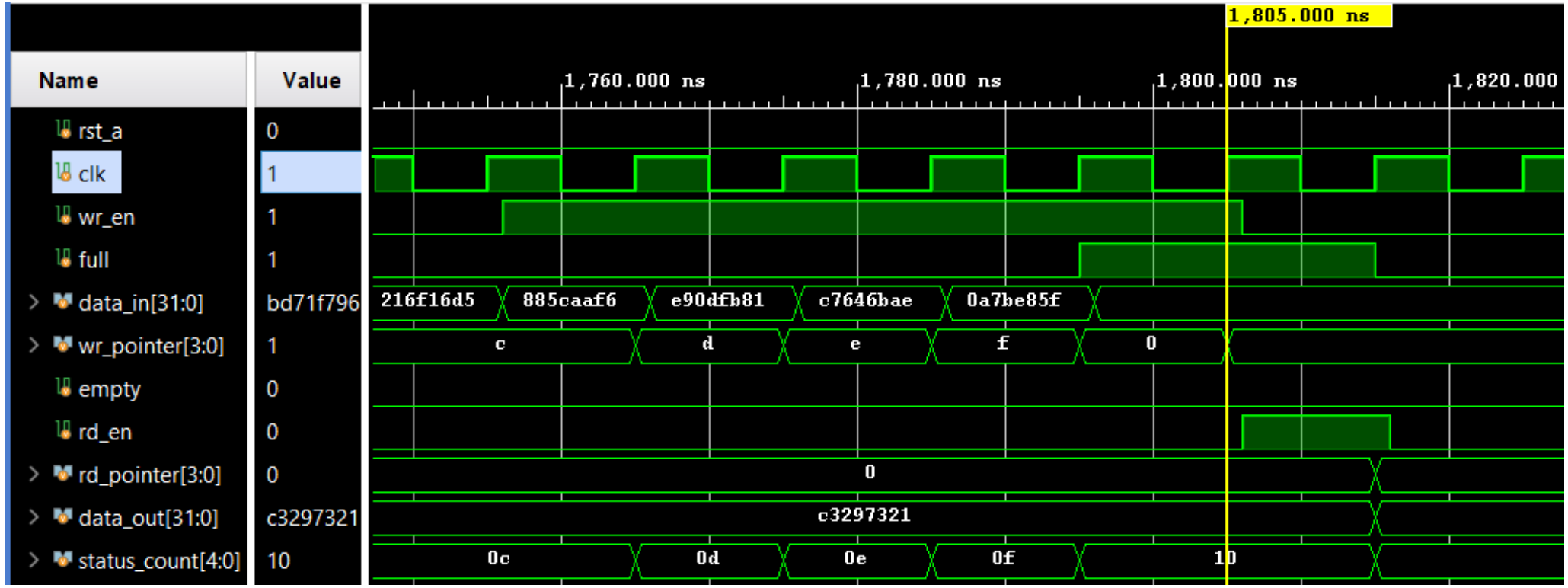
```
//--------------Status pointer for full and empty checking
   always_ff @ (posedge clk,rst_a)
   begin
   if(rst_a) status_count <= 0;
   else begin
   if(wr_en && !rd_en && status_count != ram_depth)
         status_count <= status_count + 1;
   else if(rd_en && !wr_en && (status_count != 0))
   status_count <= status_count - 1;
        end
   end // always @ (posedge clk,posedge rst_a)

   assign full = (status_count == (ram_depth));
   assign empty = (status_count == 0);

endmodule // sync_fifo
```

Синхронный FIFO (с пустым тактом)

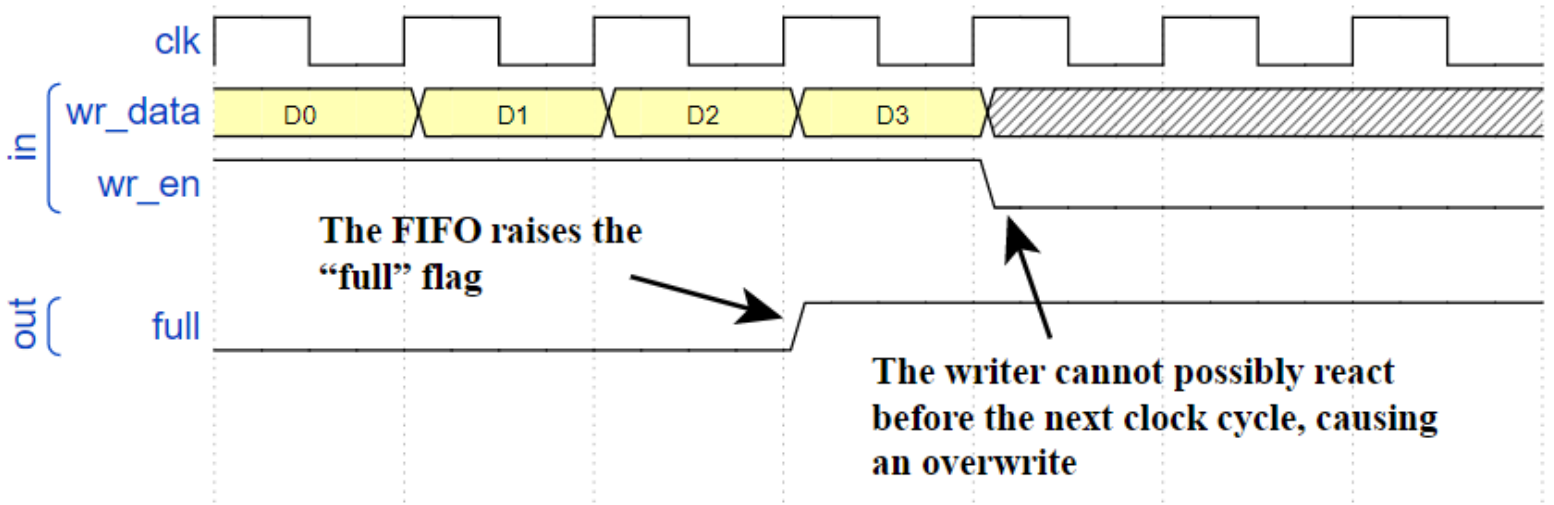




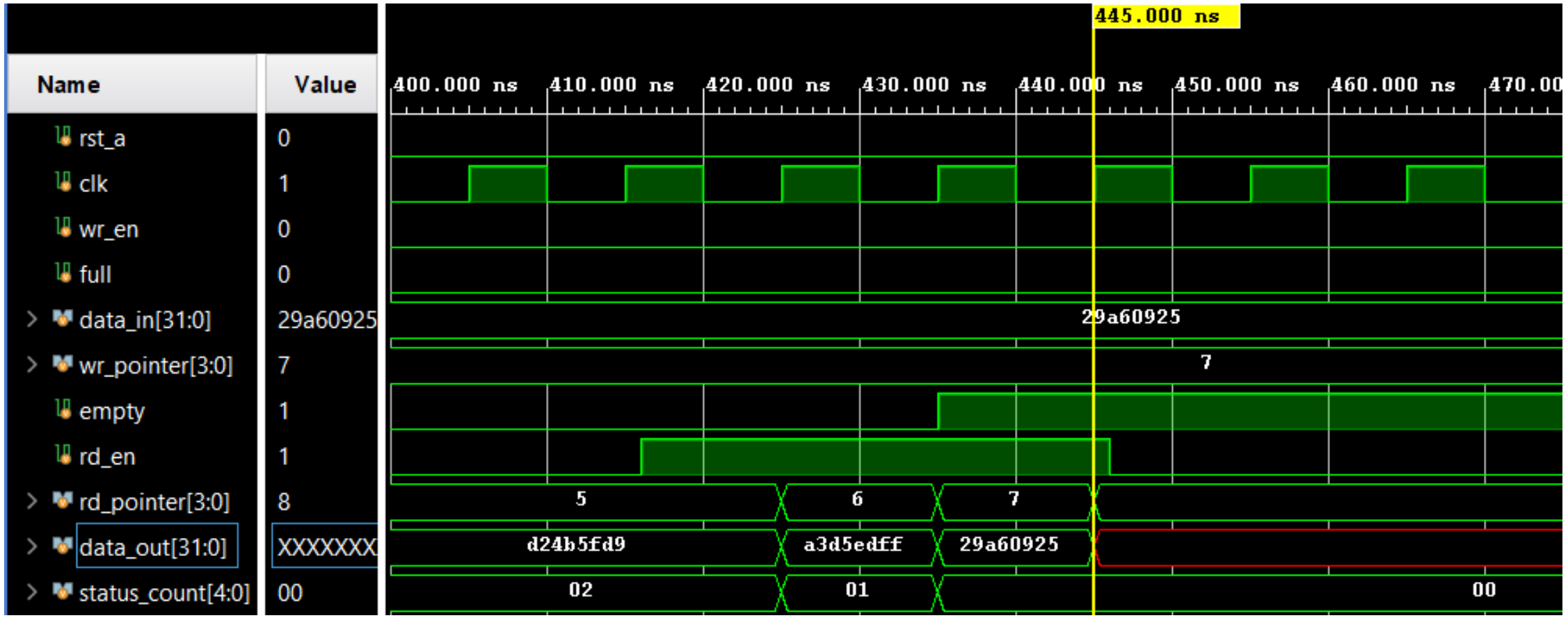
Синхронный FIFO (переполнение)



Синхронный FIFO (верификация)



Синхронный FIFO (чтение пустого буфера)



Синхронный FIFO (модификация)

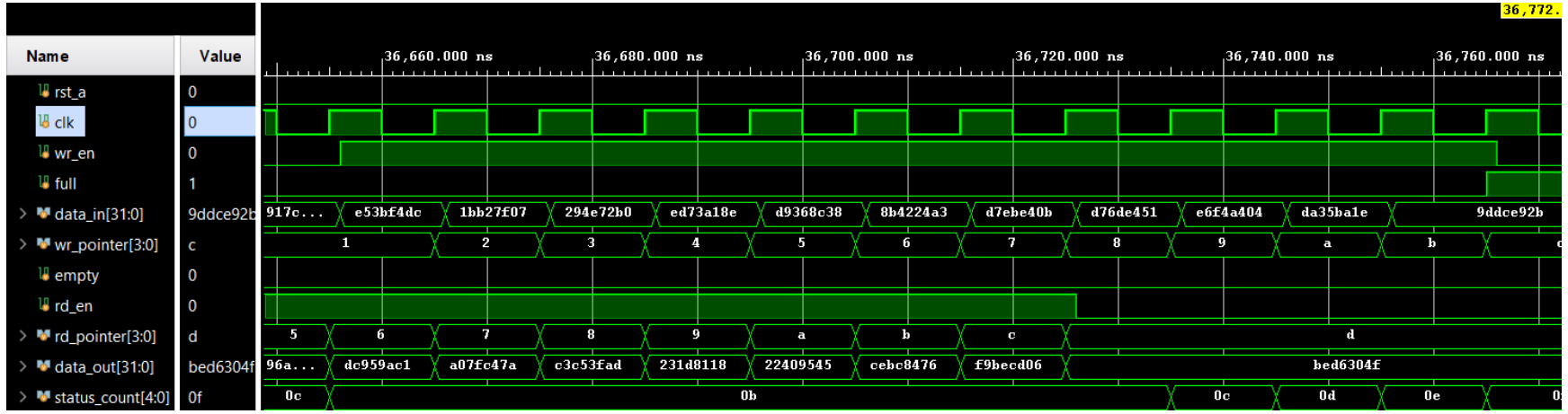
```
assign full = (status_count == (ram_depth));
assign empty = (status_count == 0);
```

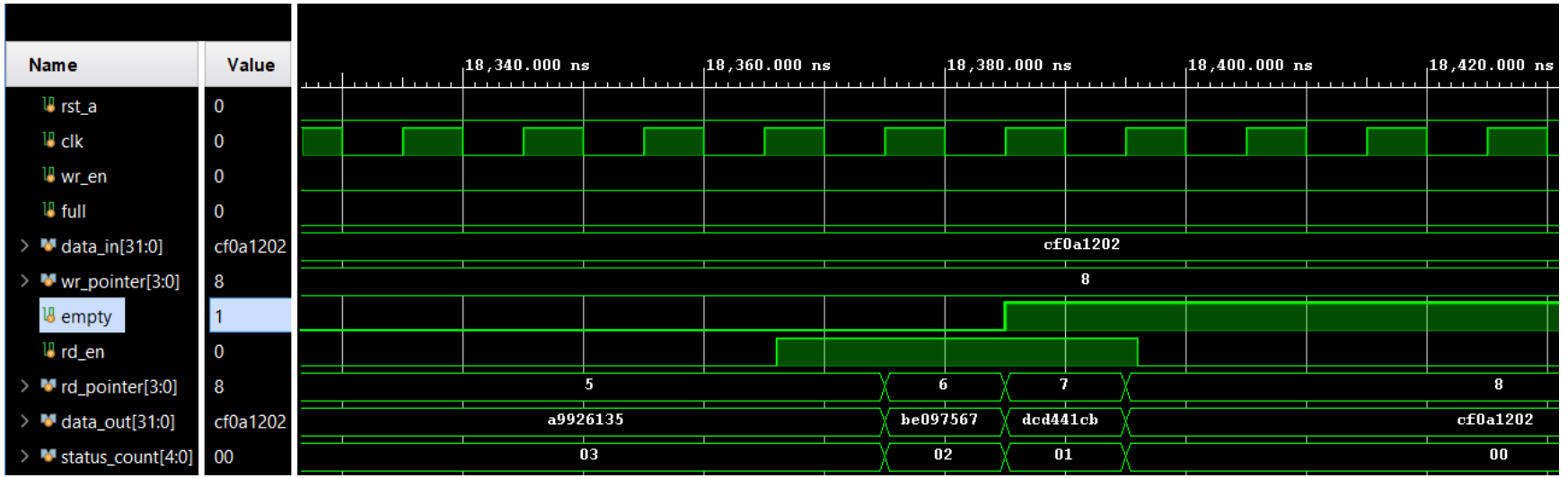
```
assign full = (status_count >= ram_depth-1);
assign empty = (status_count <= 1);
```

```
 if(rst_a) wr_pointer <= 0;
 else if(wr_en) begin
                assert (full == 0) else $fatal(1,"%t : Overflow of a fifo",$time);
```

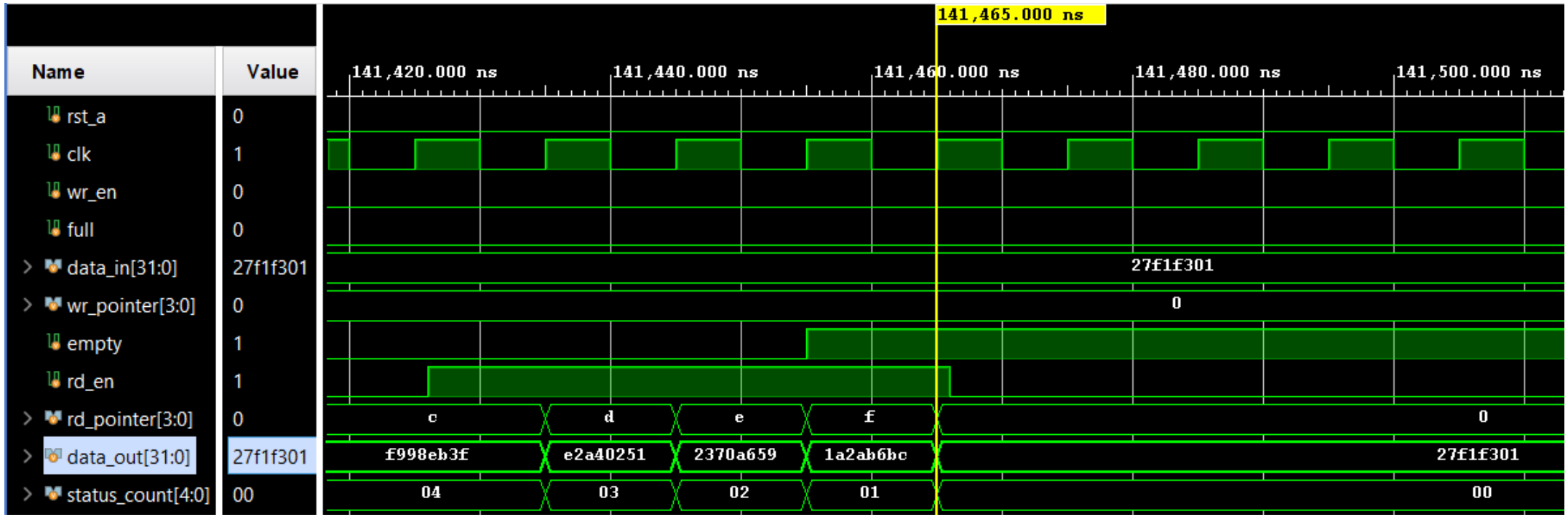
```
if(rst_a) wr_pointer <= 0;
else if(wr_en) begin
               assert (status_count <= ram_depth) else $fatal(1,"Overflow of a fifo");
```

Синхронный FIFO (после модификации)



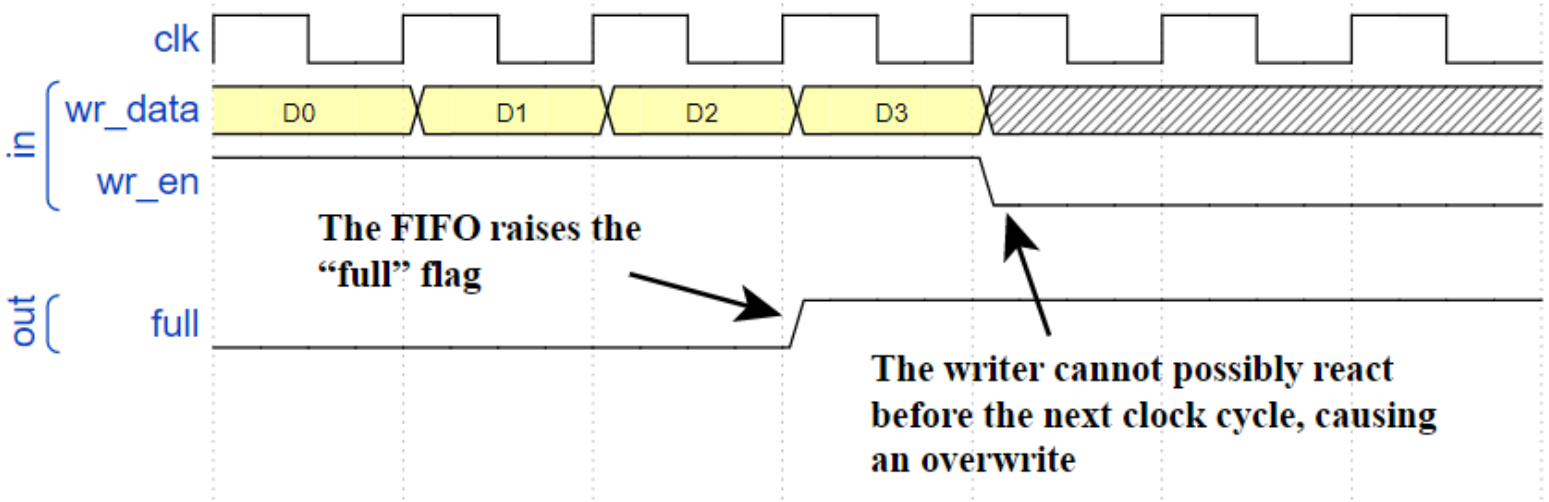


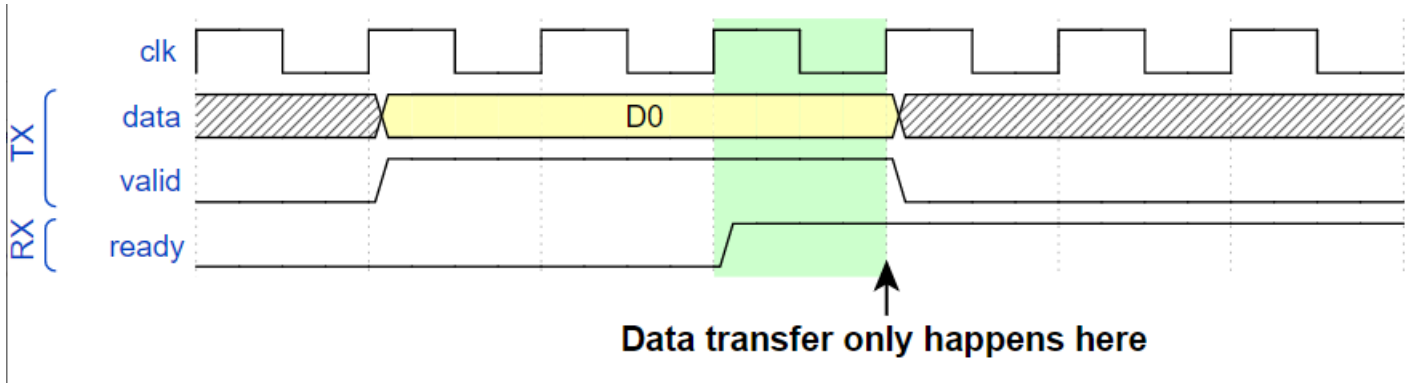
Синхронный FIFO (после модификации)



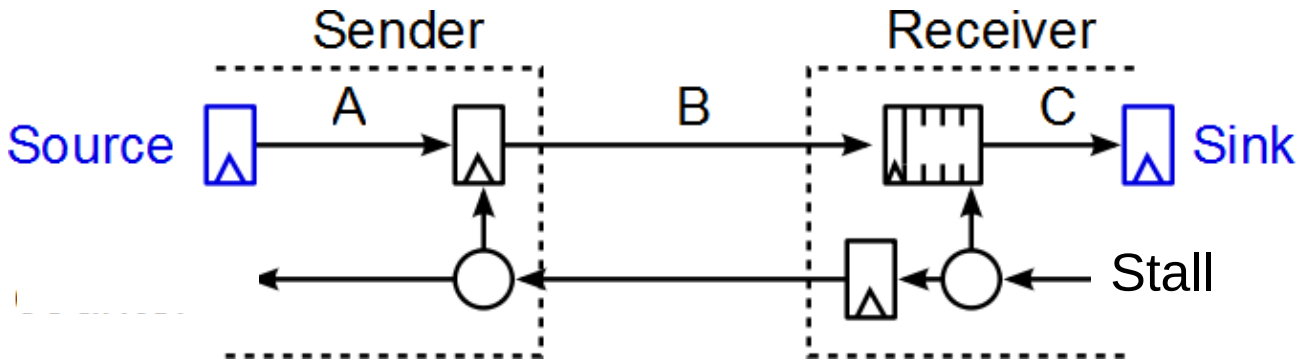
Проблема : последний записанный элемент не считывается !

Handshaking



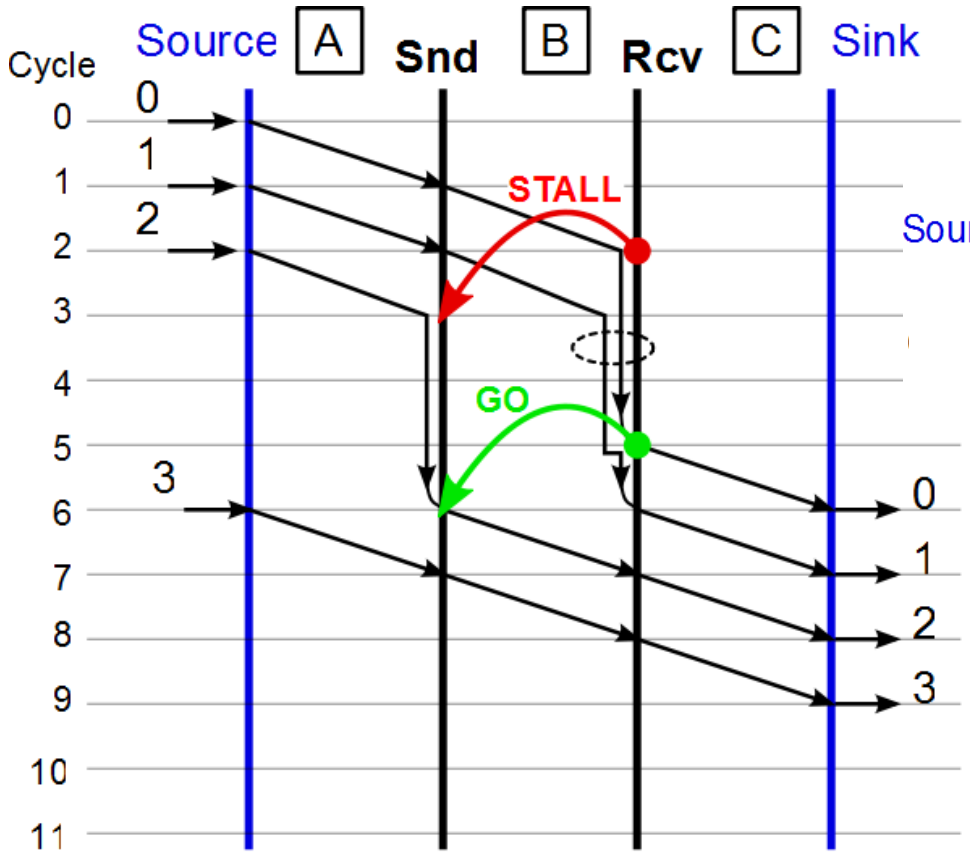


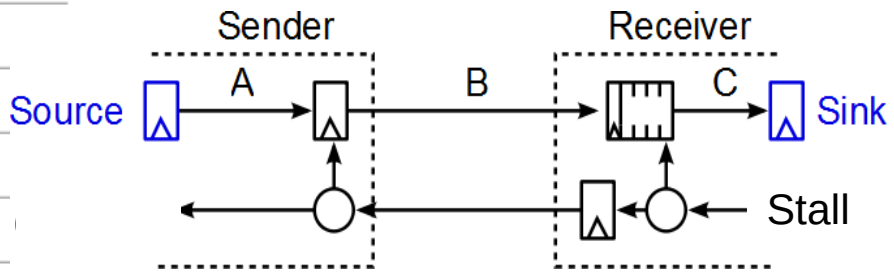
Вариант с приостановкой приёма



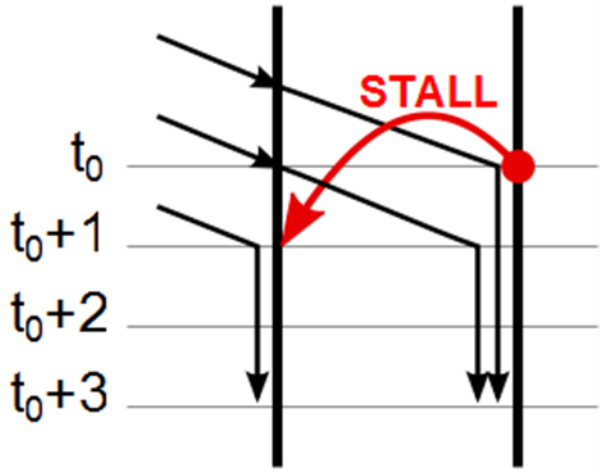
- Один сигнал STALL/GO отправляется передатчику
  - STALL=0 (G0) передатчик может передавать
  - STALL=1 (STALL) передатчик должен ждать
  - Передатчик реагирует на изменения в сигнале приёмника
- Data valid (не показано) активируется при наличии передаваемых данных

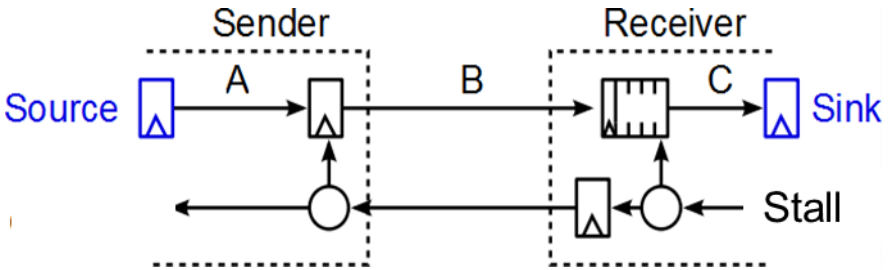
Вариант с приостановкой приёма





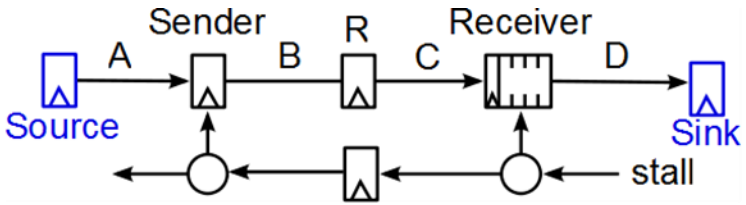
Вариант с приостановкой приёма

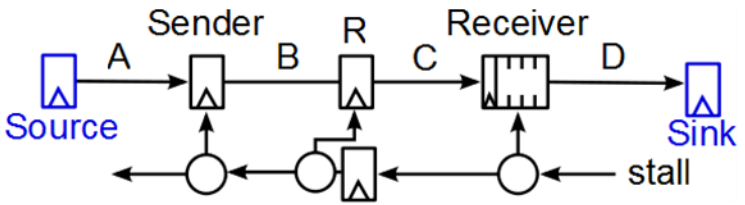


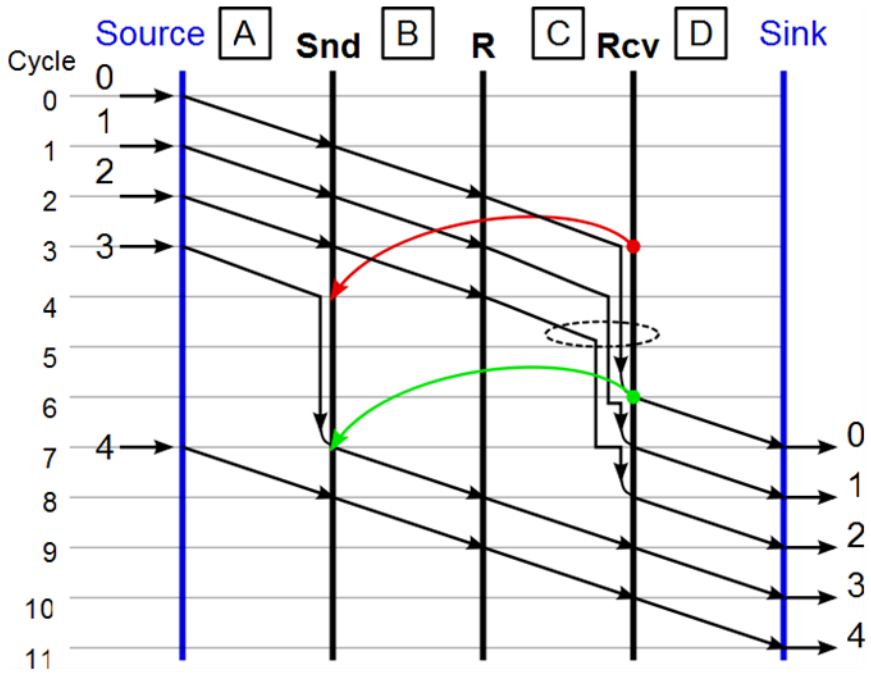


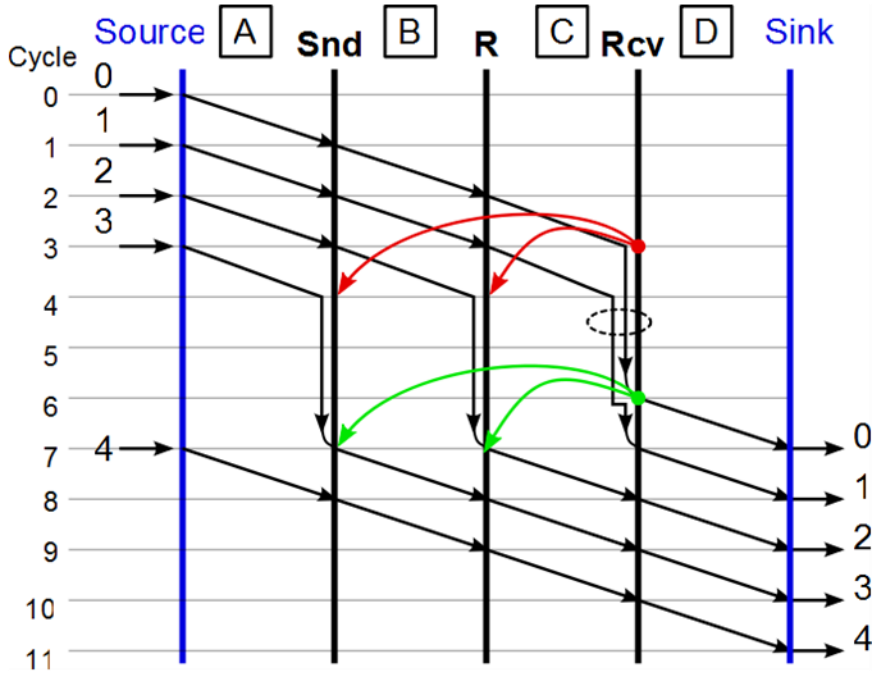
- STALL должен быть выставлен достаточно рано, чтобы:
  - Не потерять передаваемую информацию
- GO должен быть выставлен достаточно поздно, чтобы:
  - Успеть прочитать ранее принятые данные
- Минимальный объём буфера для обеспечения обмена без потерь данных должен покрывать временные затраты на передачу STALL&GO

Конвейер vs эластичный буфер





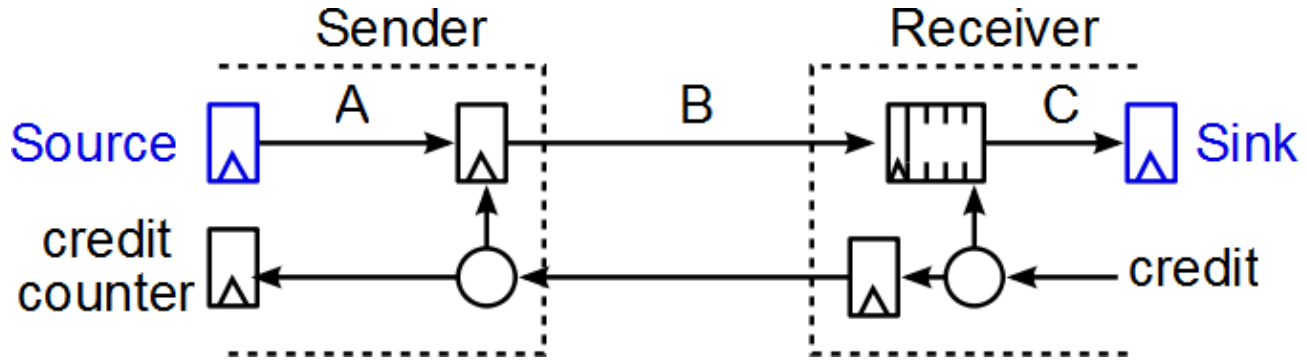




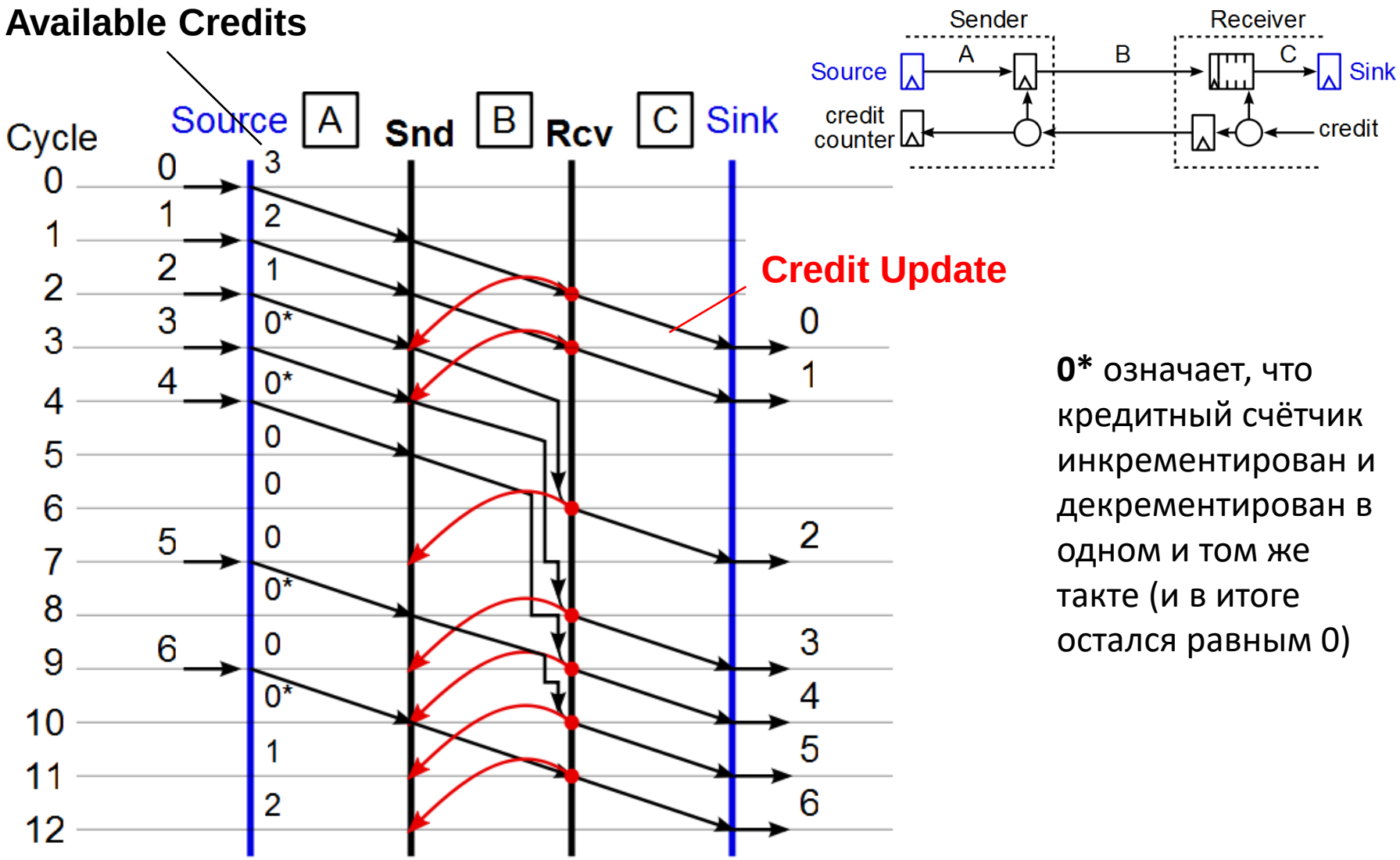
В течение Round-trip Time (RTT) передатчик «слеп». Передатчик получает информацию от приёмника после начала передачи

Кредитный счётчик

- Отправитель следит за наличием мест в буфере приёмника
  - Число свободных мест называется кредитами
  - Кредитный счётчик считает кредиты
- Если число кредитов > 0 отправитель может посылать новое слово
  - Число кредитов декрементируется с каждой передачей
- Когда в буфере приёмника освобождается место, отправителя уведомляют для инкремента счётчика

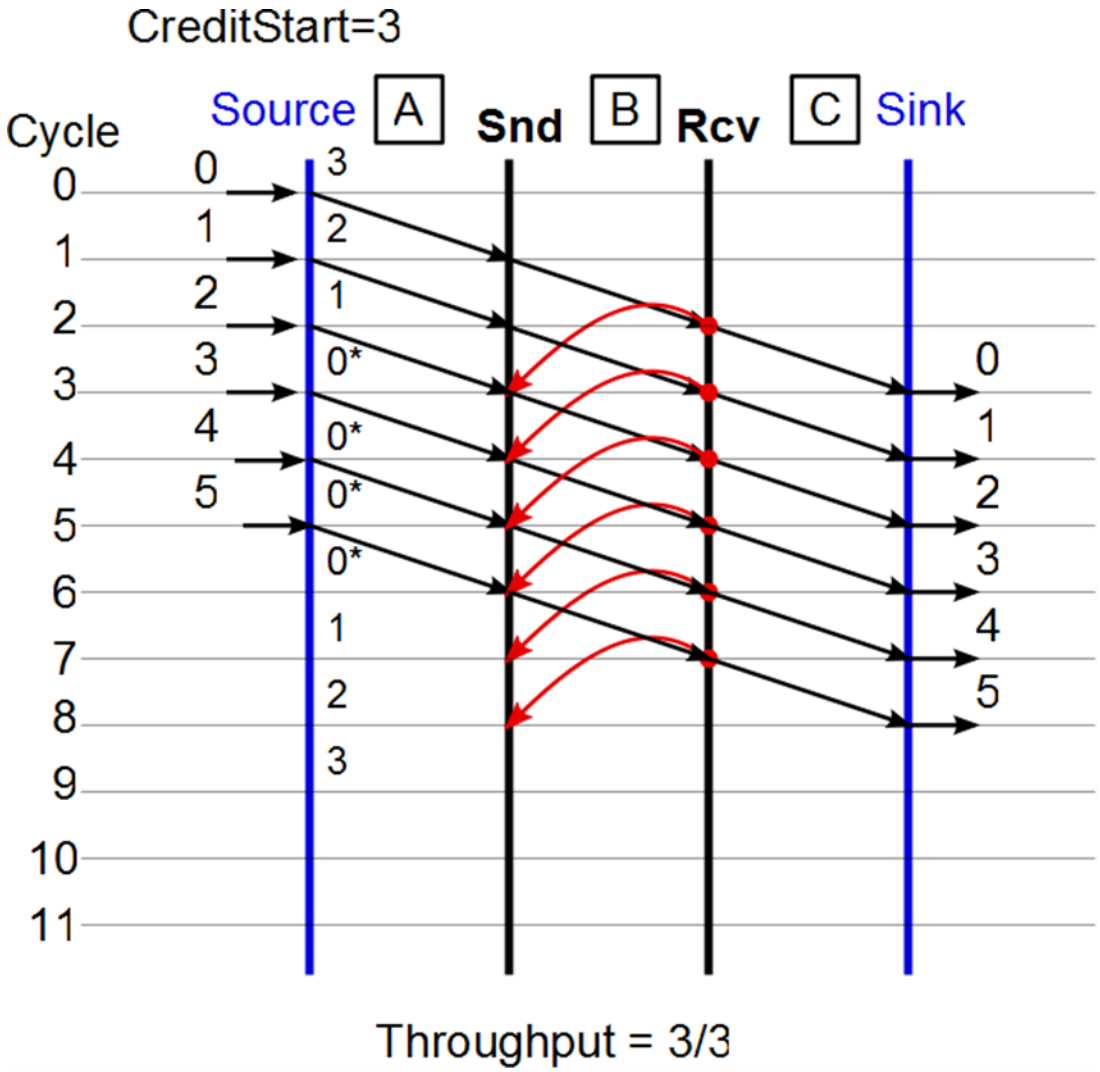


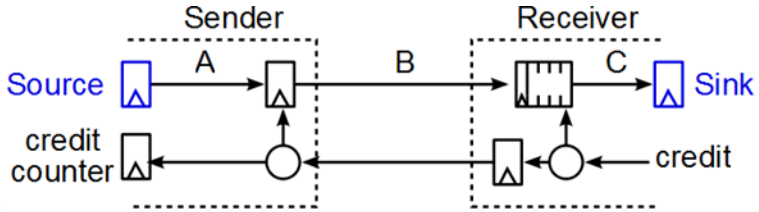
Кредитный счётчик



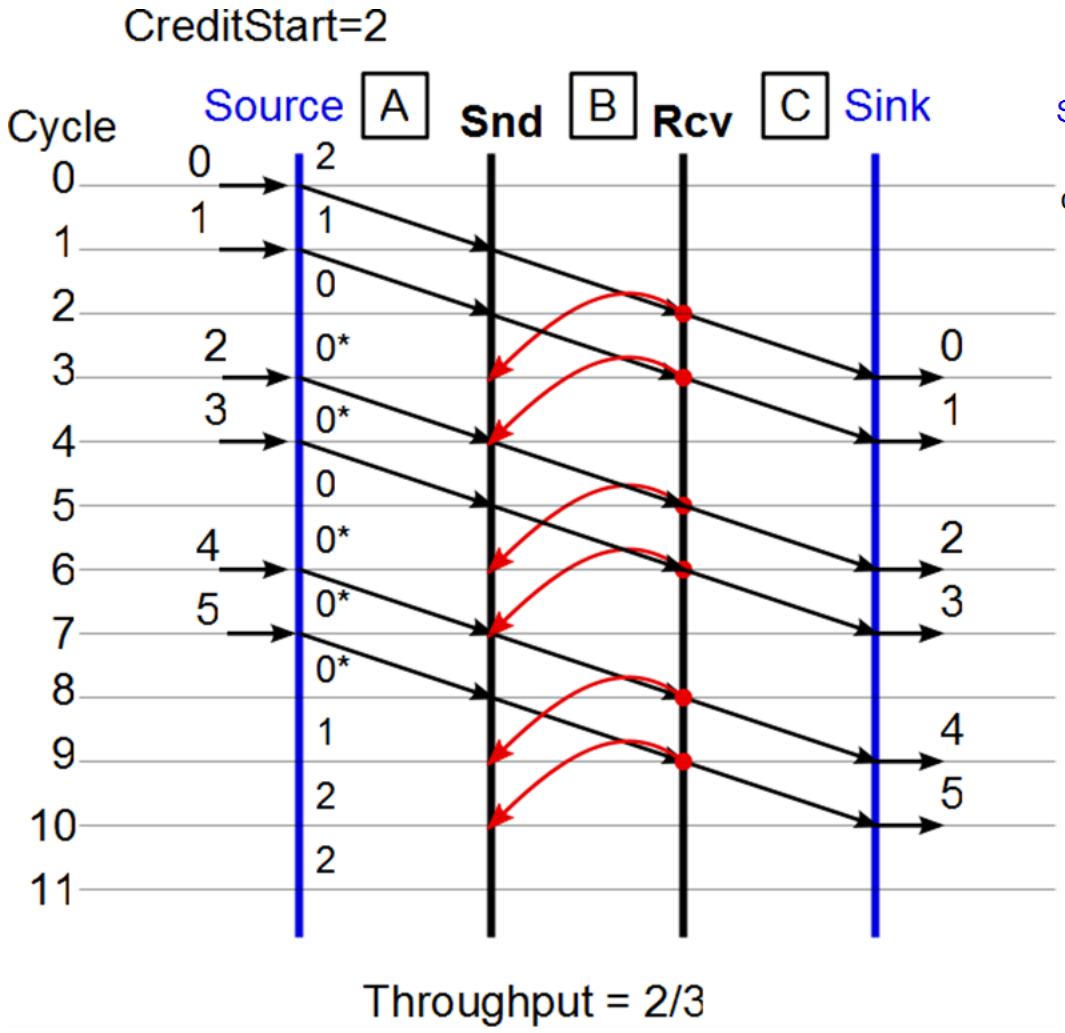
0* означает, что кредитный счётчик инкрементирован и декрементирован в одном и том же такте (и в итоге остался равным 0)

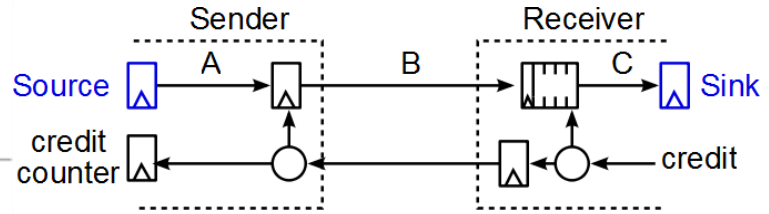
Кредитный счётчик



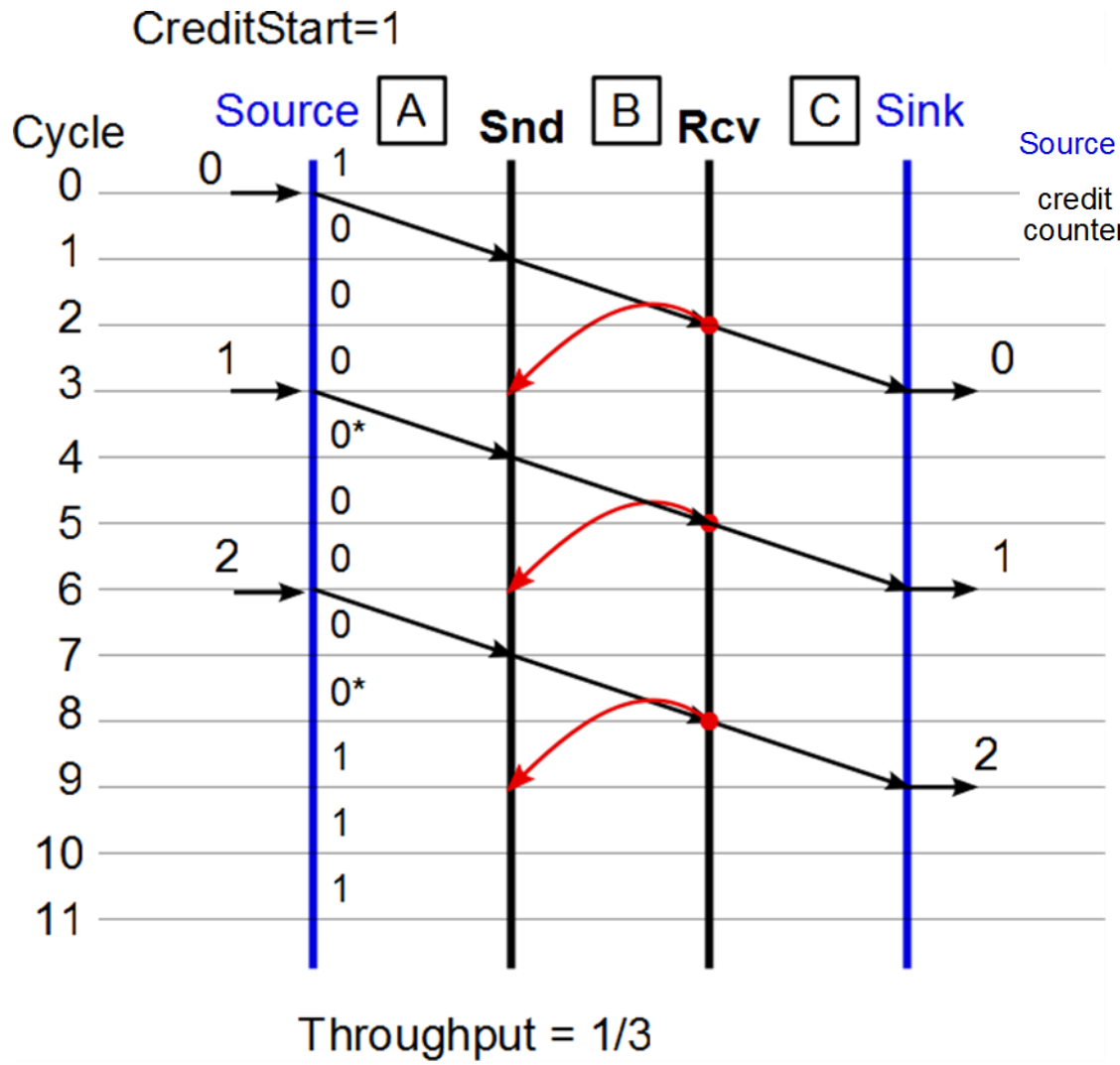


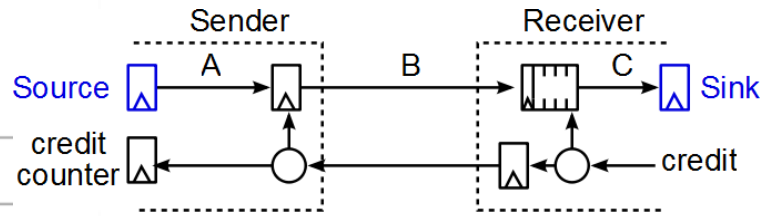
Кредитный счётчик



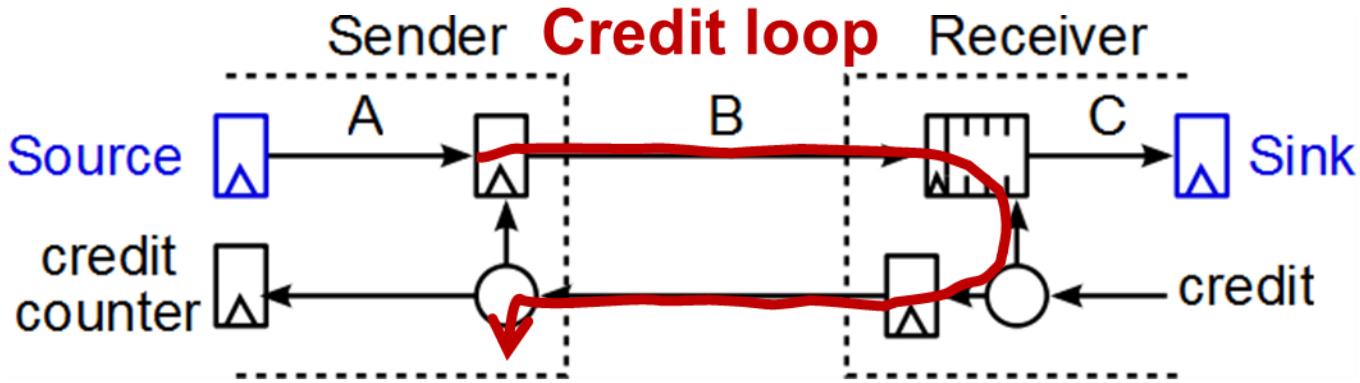


Кредитный счётчик



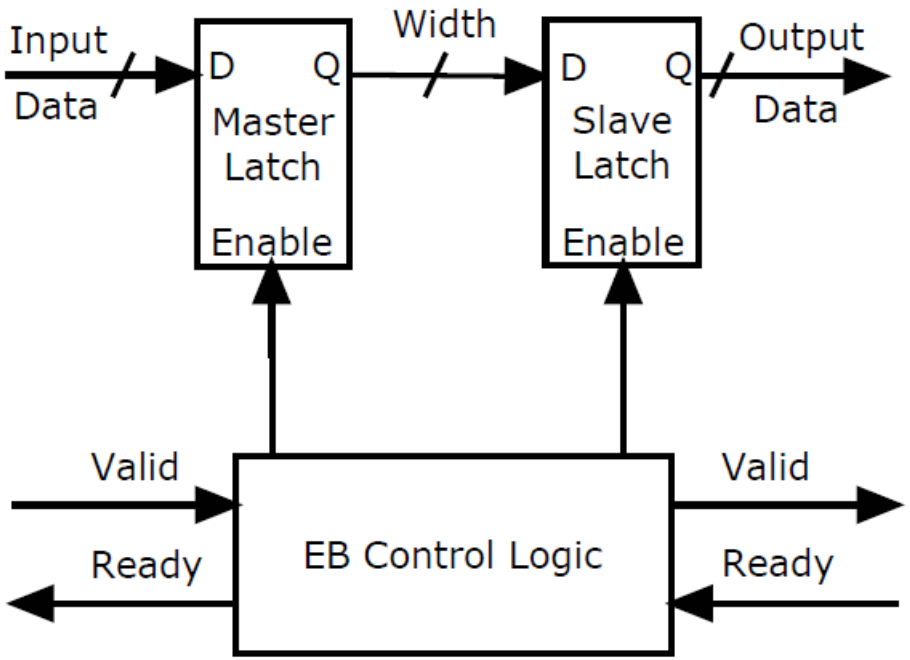


Кредитный счётчик

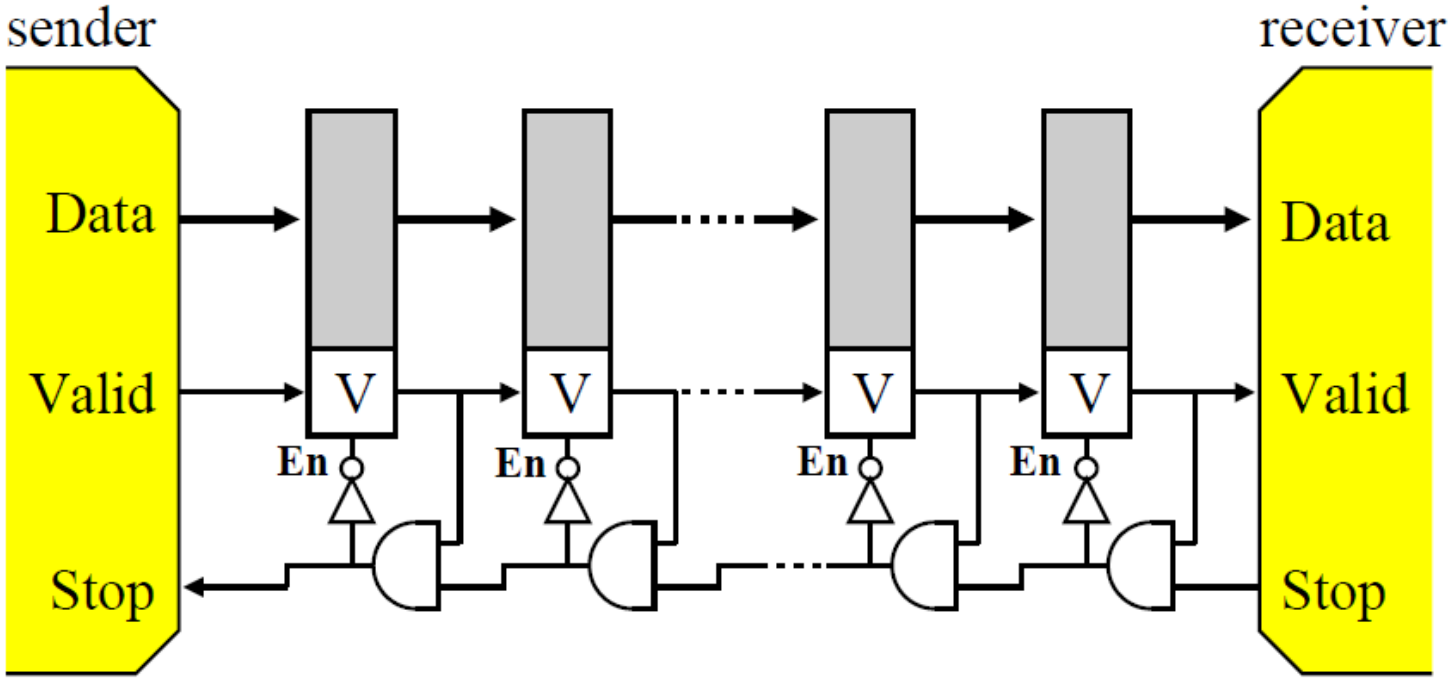


- Число регистров, через которые проходят данные и кредиты определяют длину кредитного контура
  - 100% пропускная способность гарантируется, если число ячеек в буфере равно числу регистров в кредитной петле.
- Изменение числа кредитов может изменить пропускную способность «на лету»
  - Потери отсутствуют при любом размере буфера > 0.
  - Схема Stall and Go требует, как минимум на одну ячейку больше

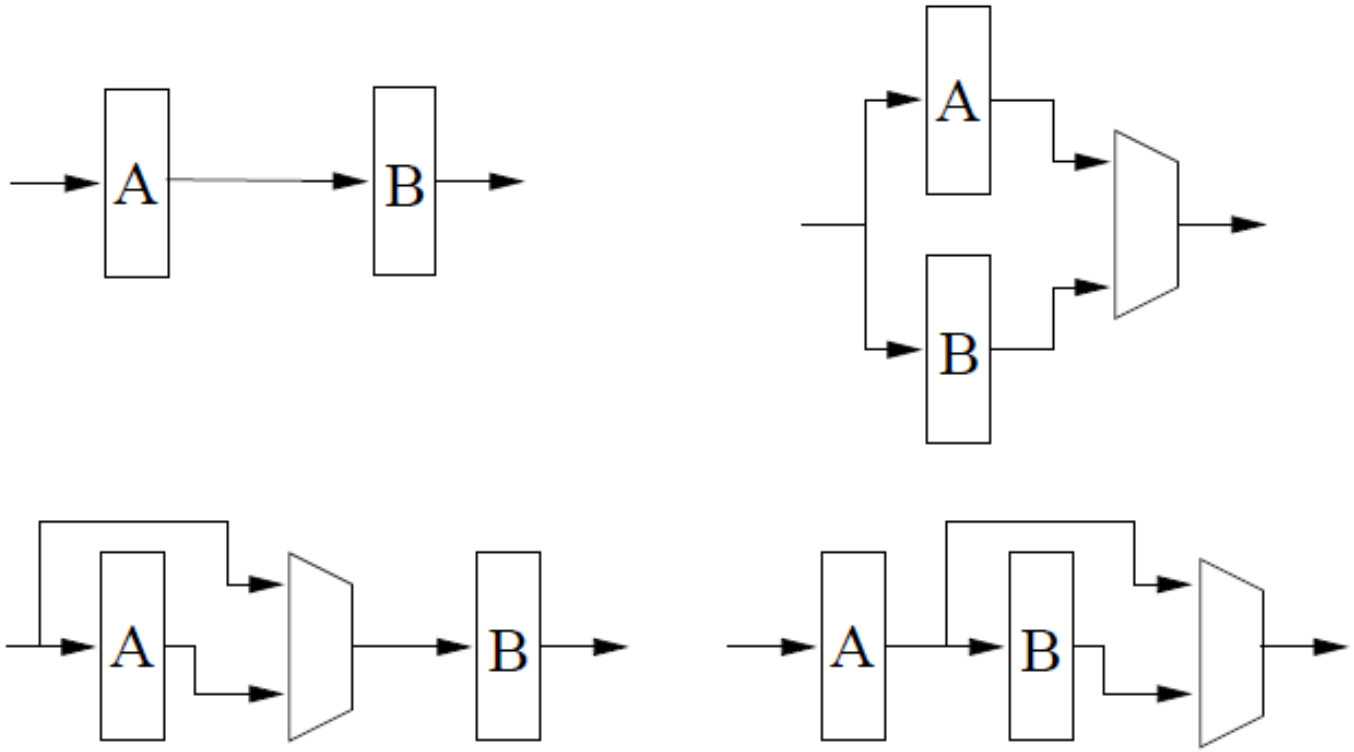
Эластичный буфер



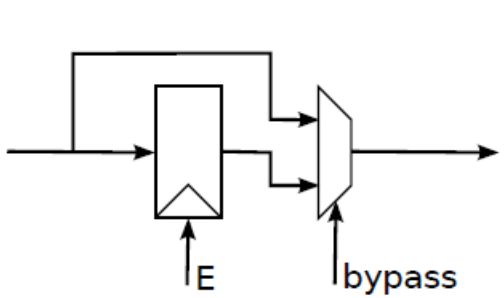
Эластичный буфер

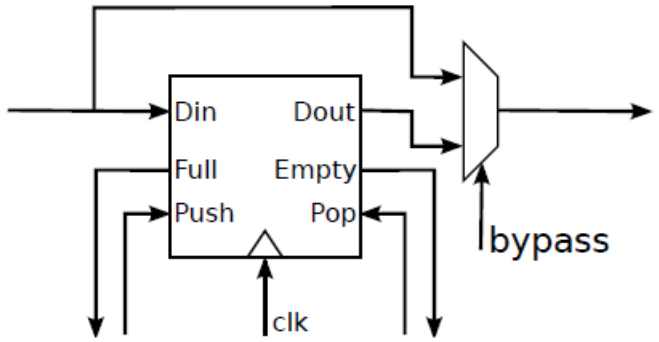


Эластичный буфер



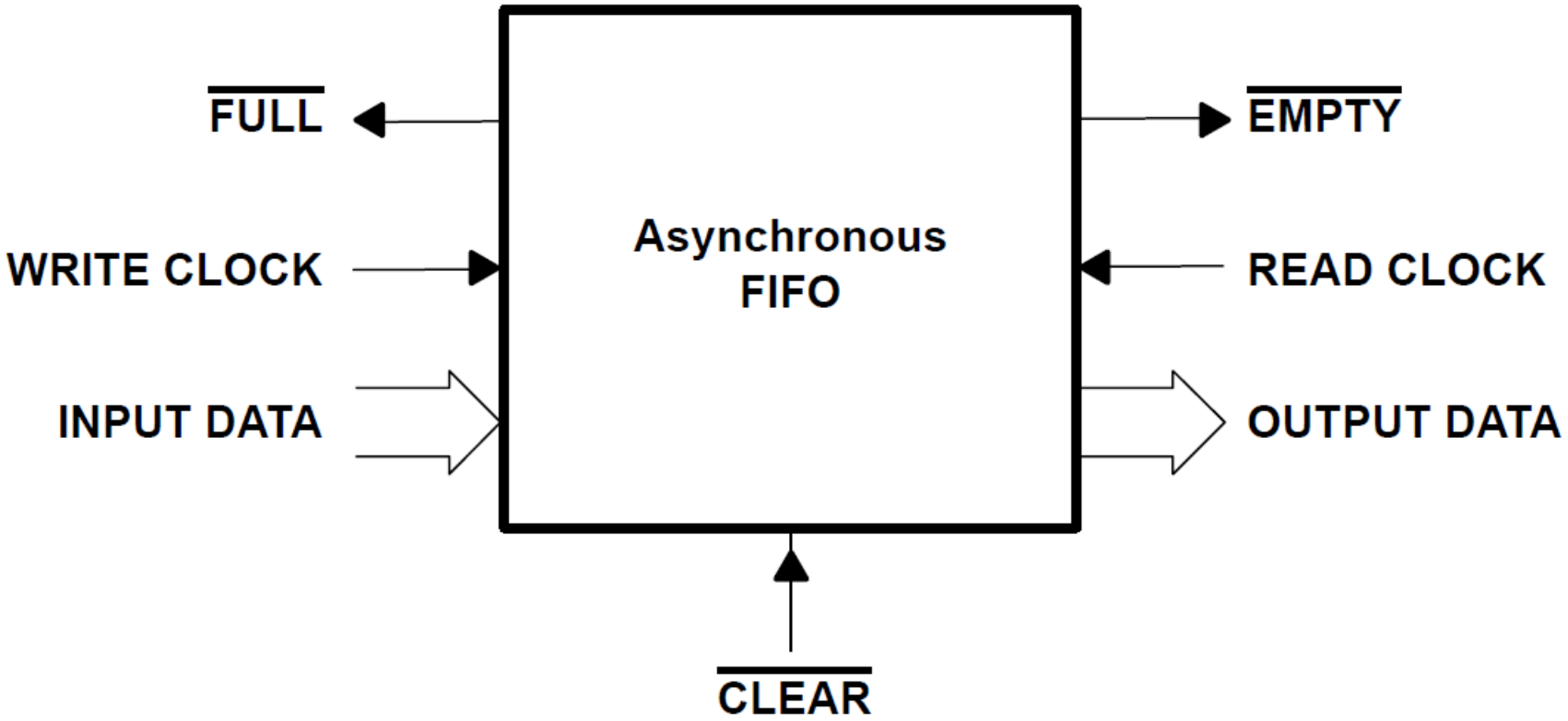
Skid - buffer



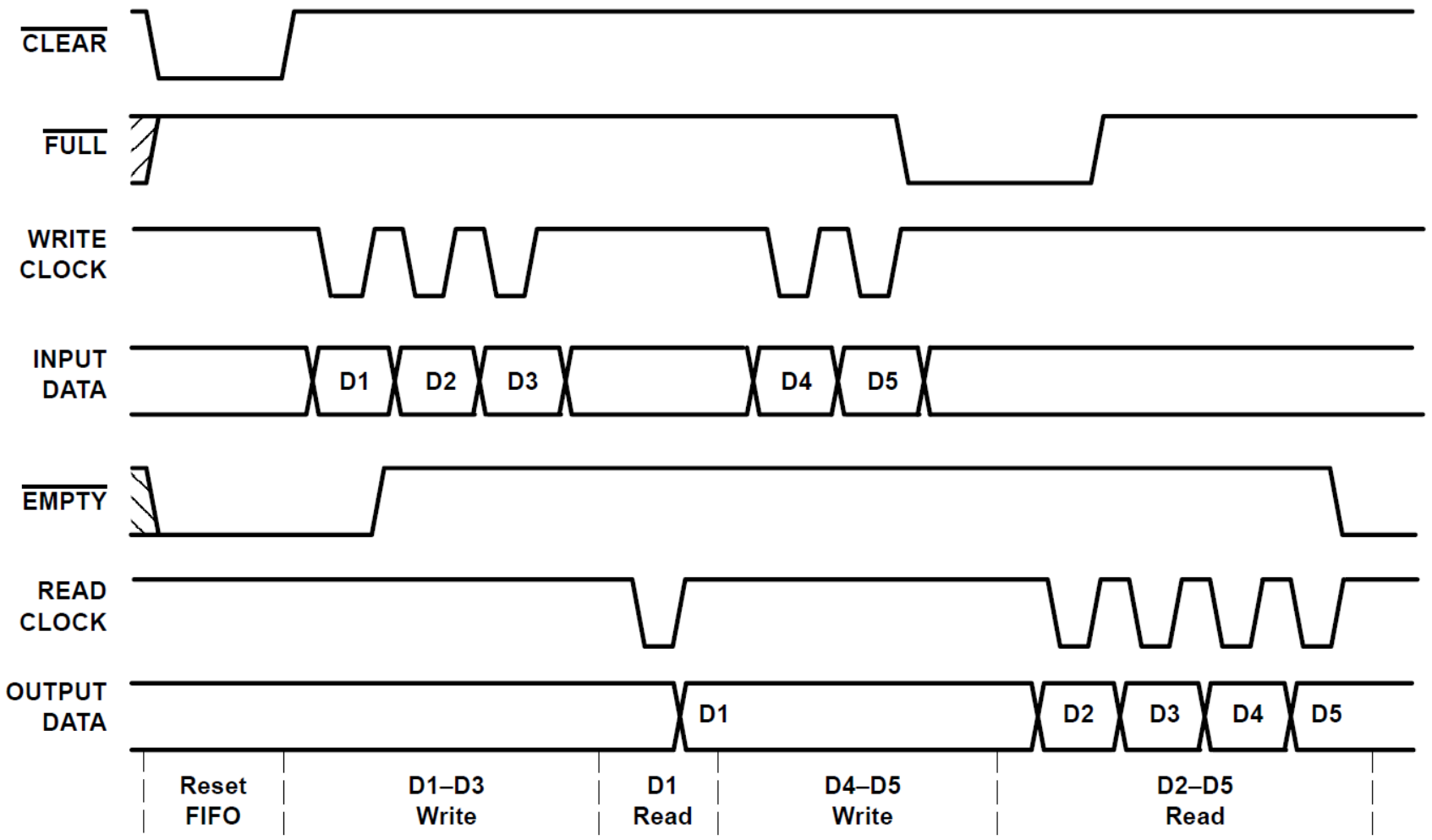


Буферы с нулевой задержкой

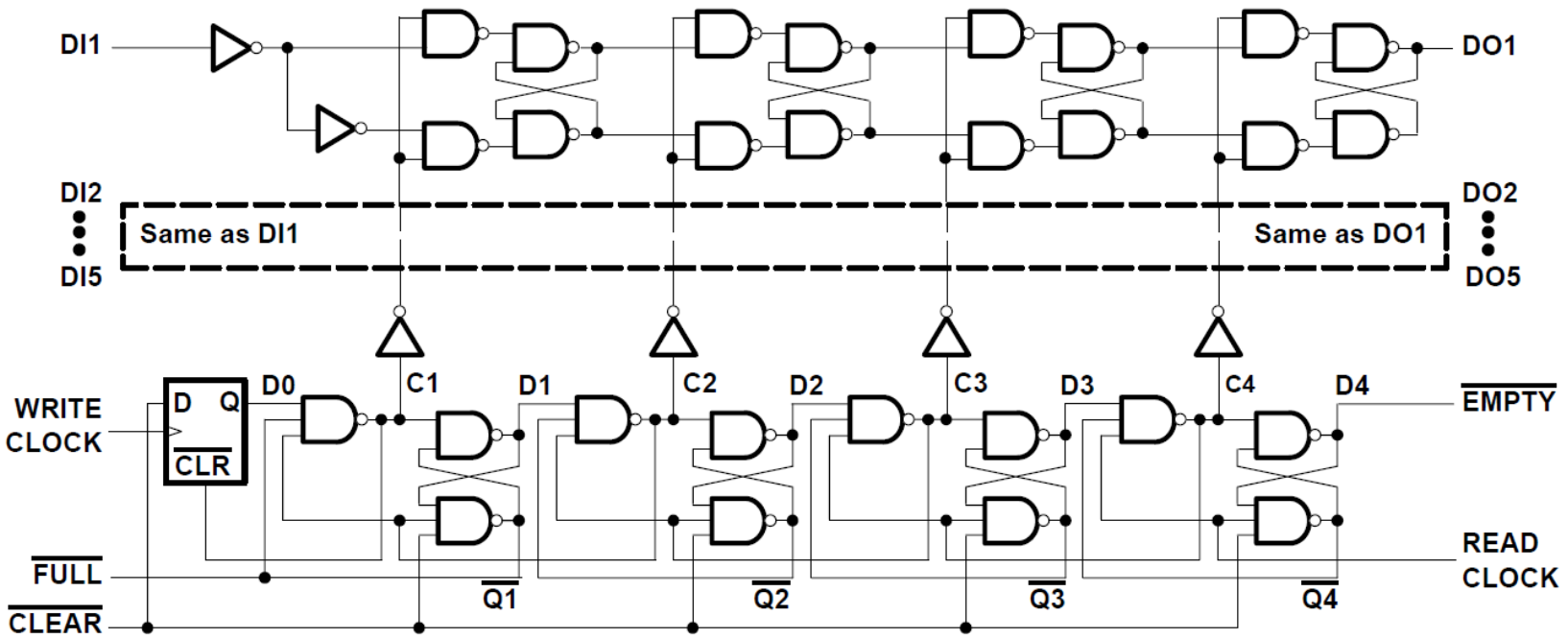
Асинхронный FIFO



Асинхронный FIFO (размер 4)

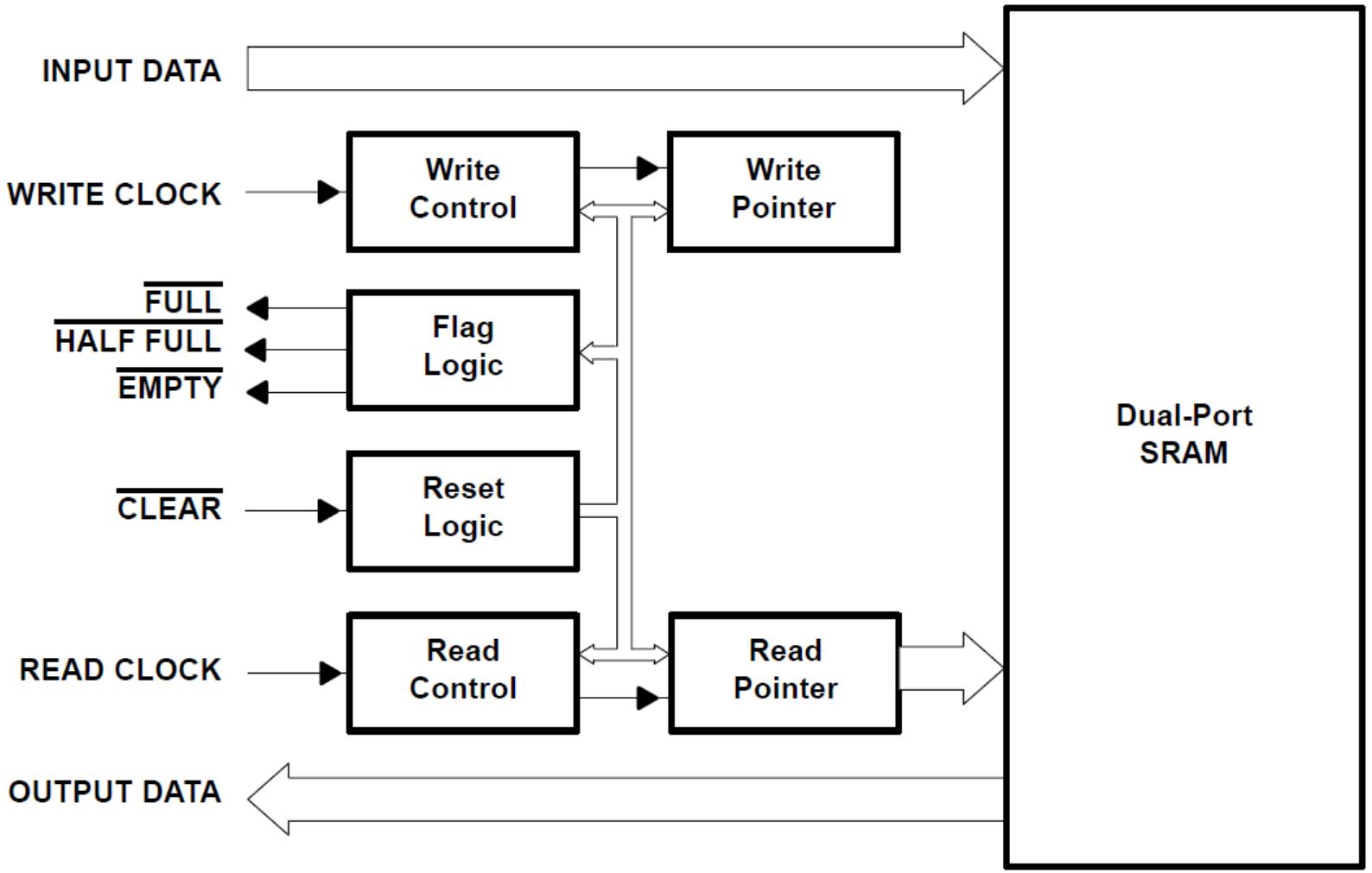


Fall-Through FIFO

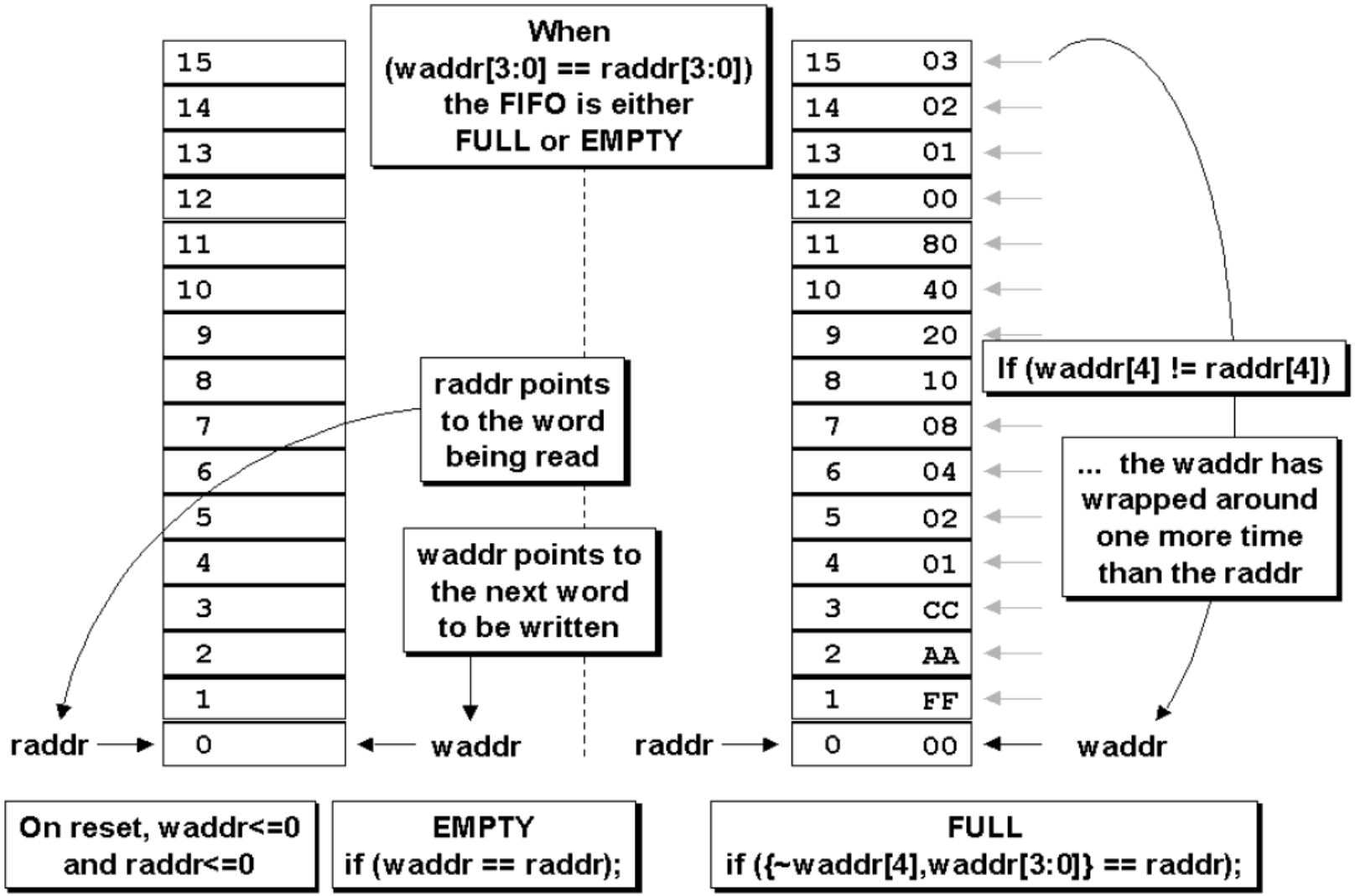


Circuitry of 4 × 5 Fall-Through FIFO

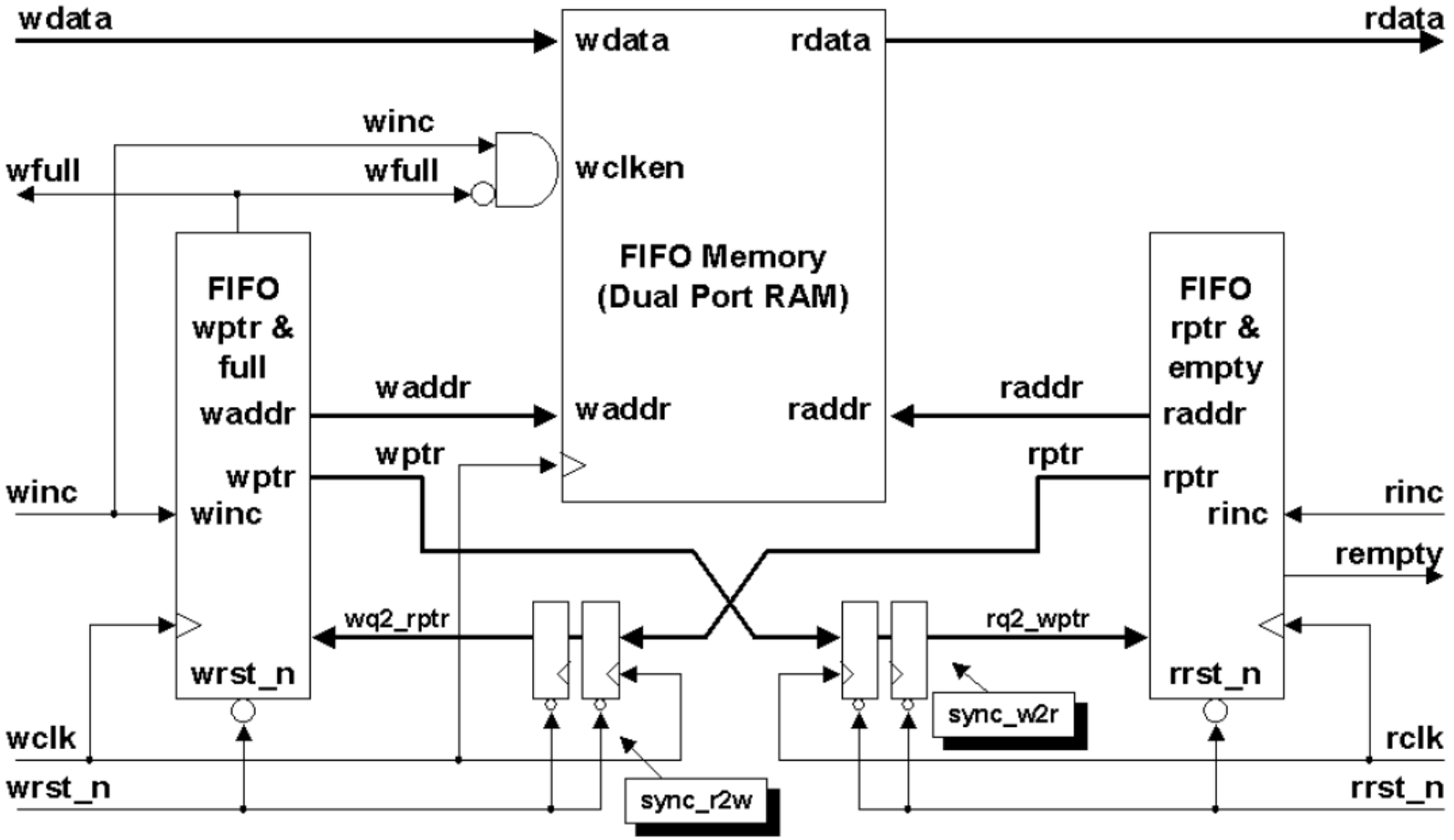
Асинхронный FIFO на основе RAM



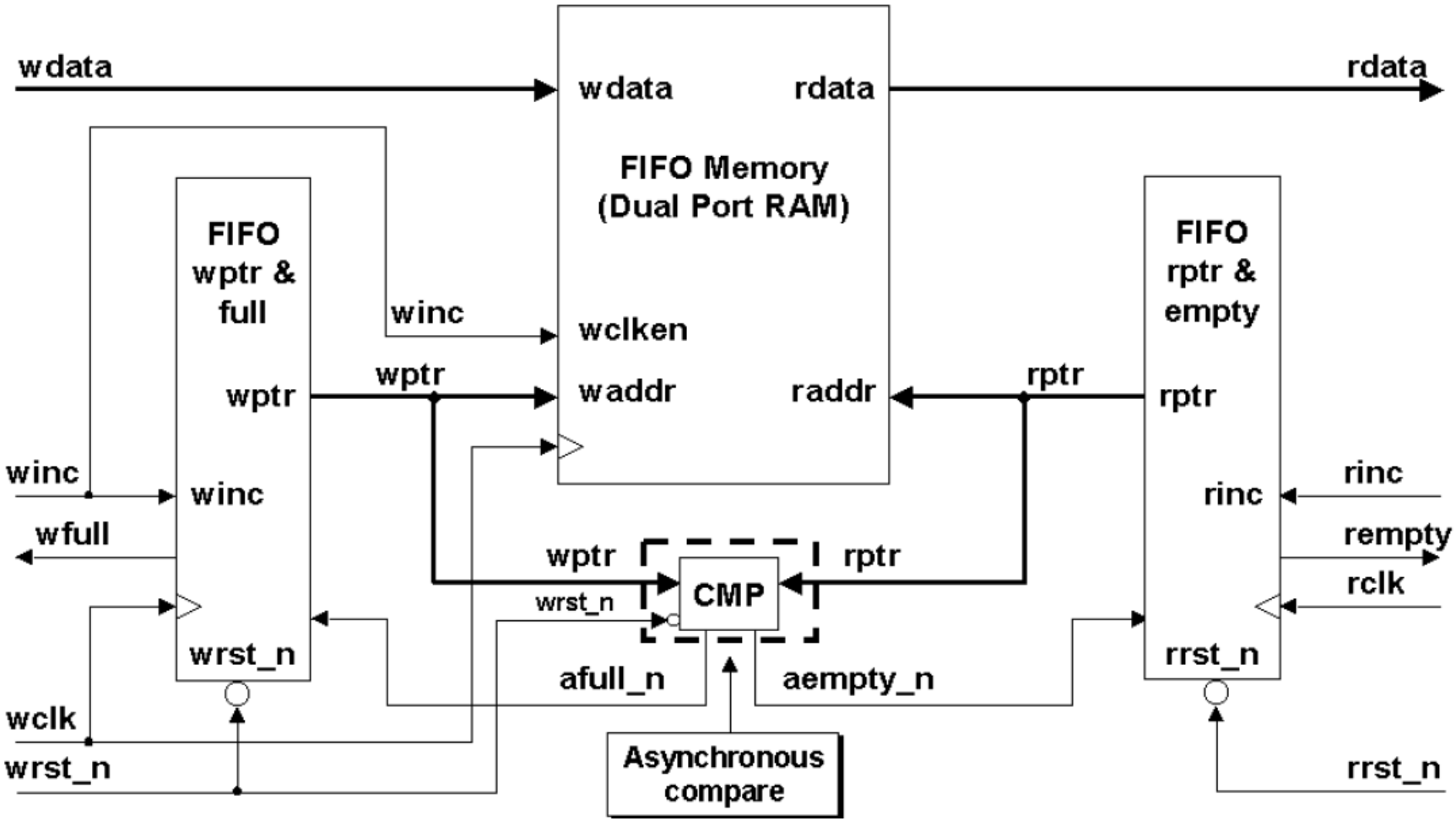
Асинхронный FIFO на основе RAM



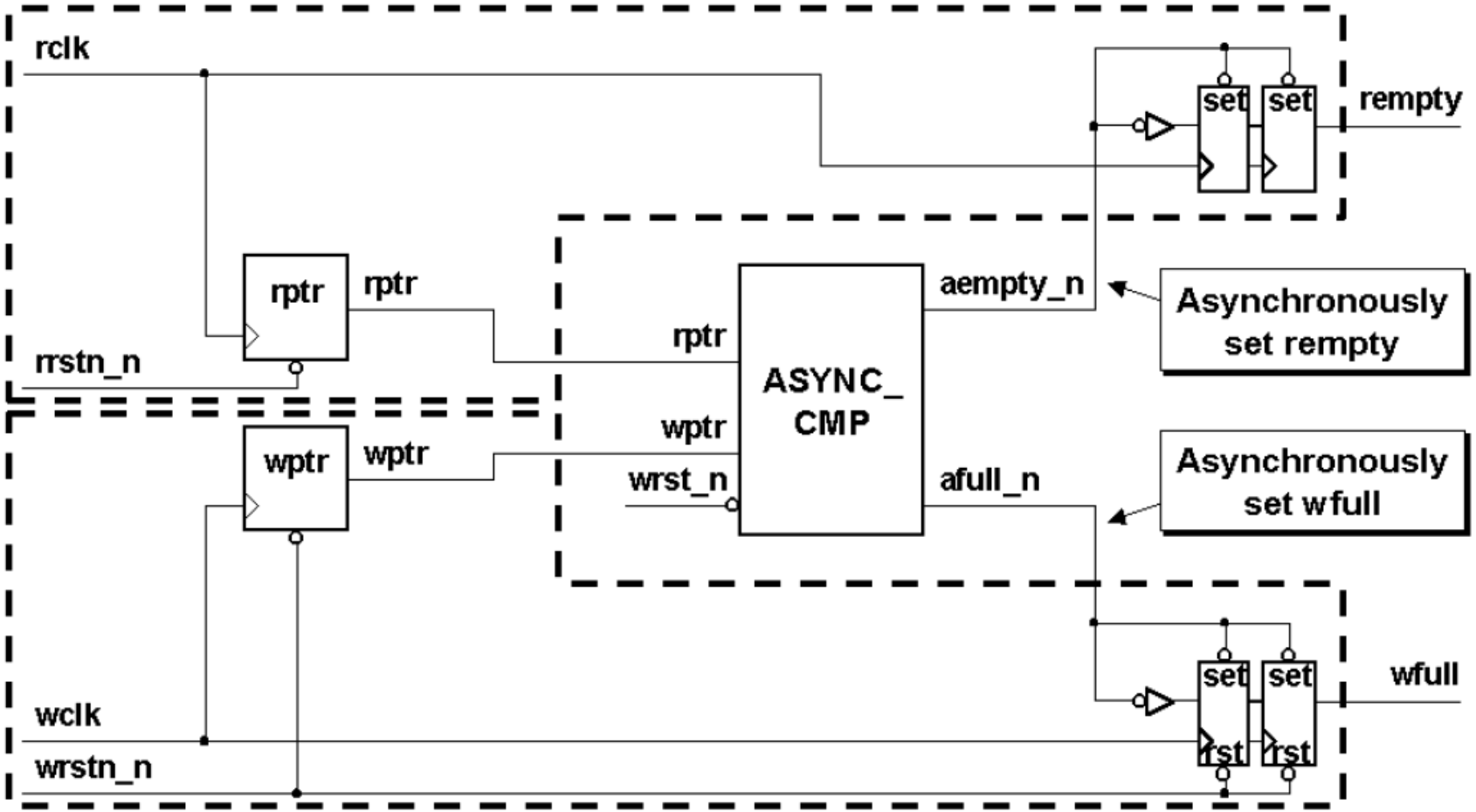
Асинхронный FIFO на основе RAM



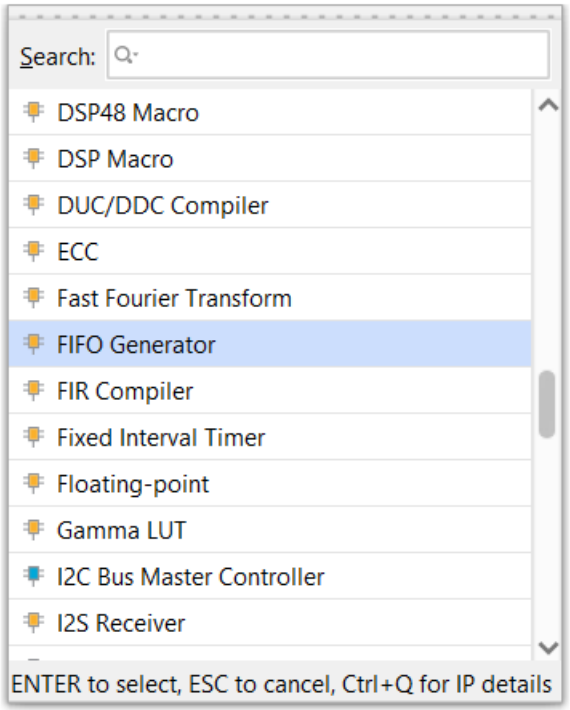
Асинхронный FIFO на основе RAM

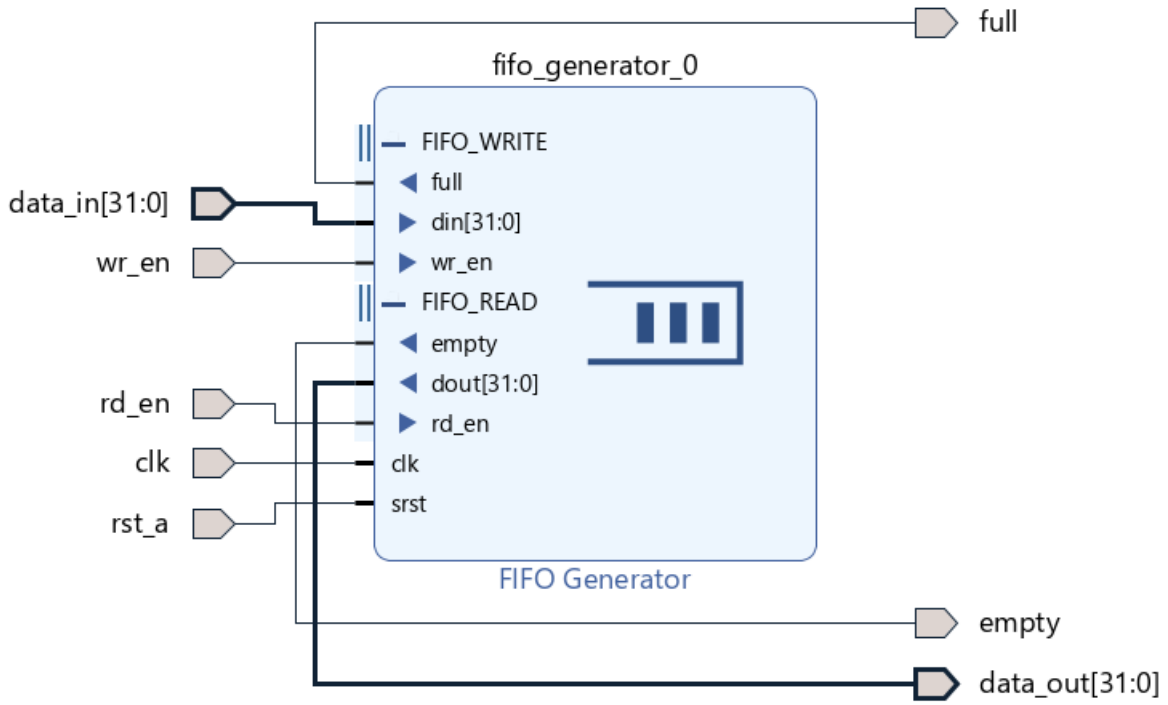


Асинхронный FIFO на основе RAM



Генератор FIFO (Vivado)





Генератор FIFO (Vivado)

Basic | **Native Ports** | **Status Flags** | **Data Counts** | **Summary**

Interface Type

(•) Native ( ) AXI Memory Mapped ( ) AXI Stream

Fifo Implementation: Common Clock Block RAM

FIFO Implementation Options

Supported Features

	Memory Type	(1)	(2)	(3)	(4)	(5)
Common Clock (CLK)	**Block RAM**	✓	✓		✓	✓
Common Clock (CLK)	Distributed RAM		✓			
Common Clock (CLK)	Shift Register					
Common Clock (CLK)	Built-in FIFO		✓	✓	✓	✓
Independent Clocks (RD_CLK, WR_CLK)	Block RAM	✓	✓		✓	✓
Independent Clocks (RD_CLK, WR_CLK)	Distributed RAM		✓			
Independent Clocks (RD_CLK, WR_CLK)	Built-in FIFO		✓	✓	✓	✓

(1) Non-symmetric aspect ratios (different read and write data widths)
(2) First-Word Fall-Through
(3) Uses Built-in FIFO primitives
(4) ECC support
(5) Dynamic Error Injection