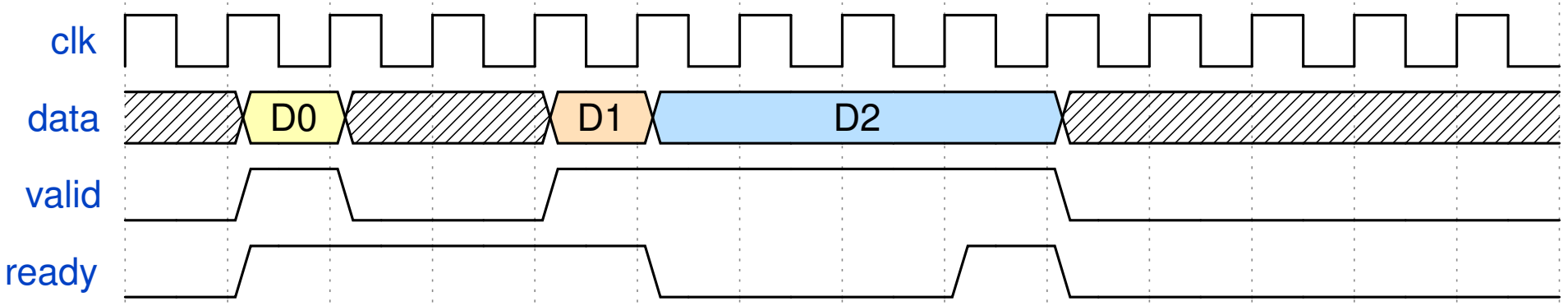
Интерфейс **valid/ready**
Двойные буфера
Счётчики кредитов

Дмитрий Смехов

Основные цели

- Проверка многопортовой памяти в различных режимах работы
- Применение транзакций в системе тестирования

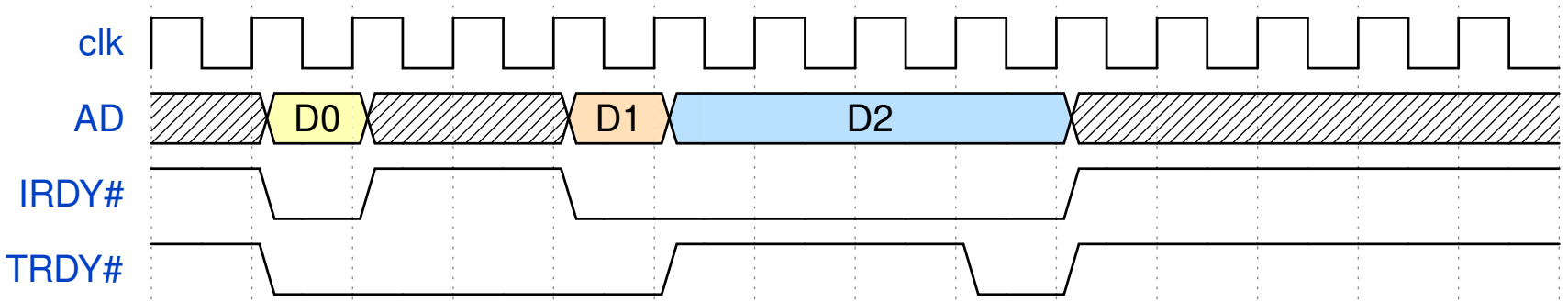
Интерфейс valid/ready



Слово данных считается переданным когда на фронте тактового сигнала установлены оба сигнала: **valid** и **ready**

Внешние шины

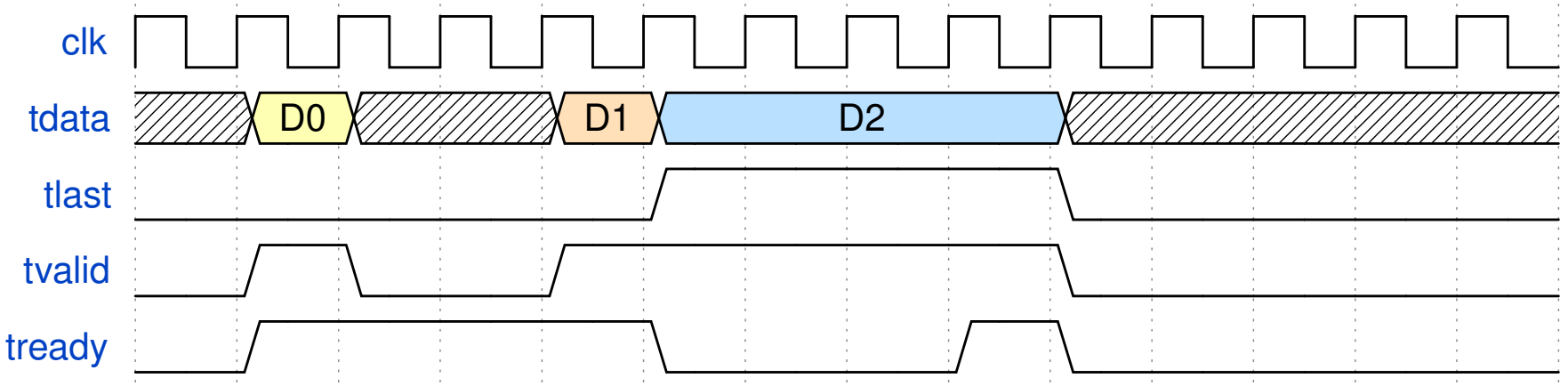
Подобный интерфейс используется в различных внешних шинах, таких как ISA, PCI, Compact Flash



На слайде представлен фрагмент временной диаграммы шины PCI. Данные передаются при IRDY#=0 и TRDY#=0

- **IRDY#** = NOT **VALID**
- **TDRY#** = NOT **READY**

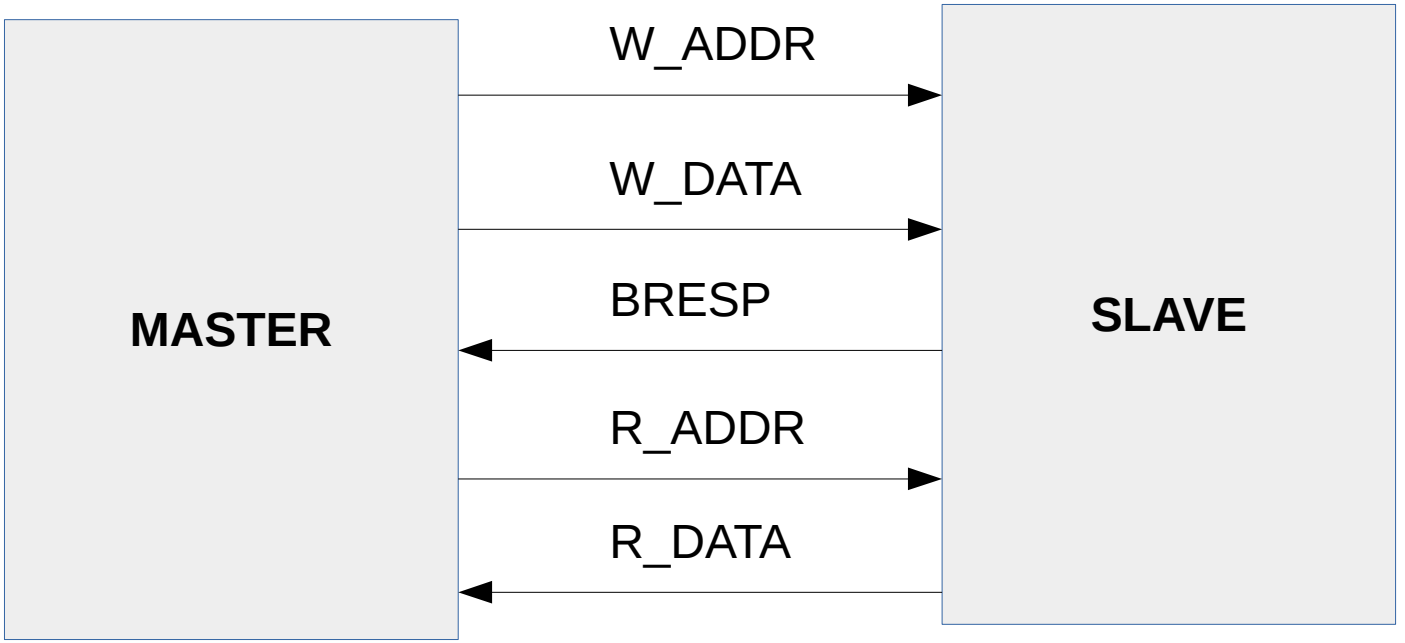
Внутренние шины - AXI STREAM



Дополнительные сигналы (**tlast**, **tuser**) передаются по тем же правилам что и **tdata**:

Передача происходит только при **tvalid**=1 и **tready**=1

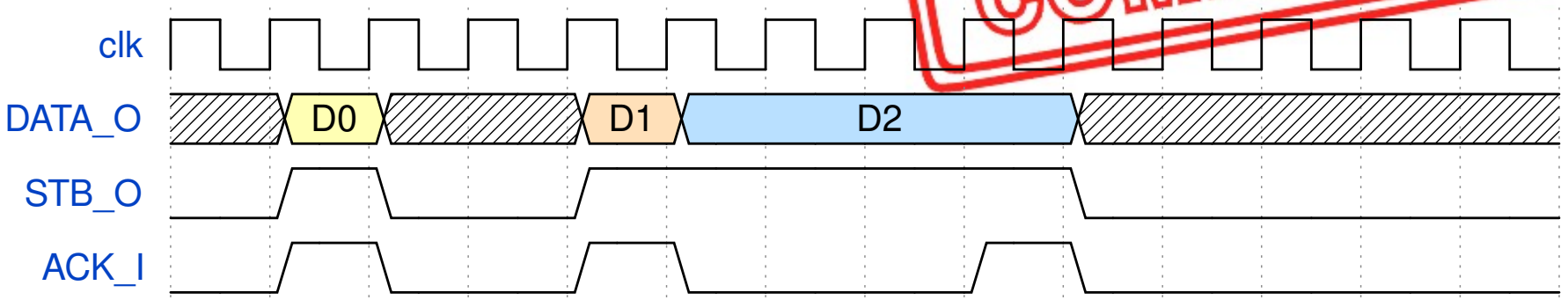
AXI MEMORY MAP - пять шин



Каждая шина содержит сигналы **valid** и **ready**

Передача происходит только при **valid**=1 и **ready**=1

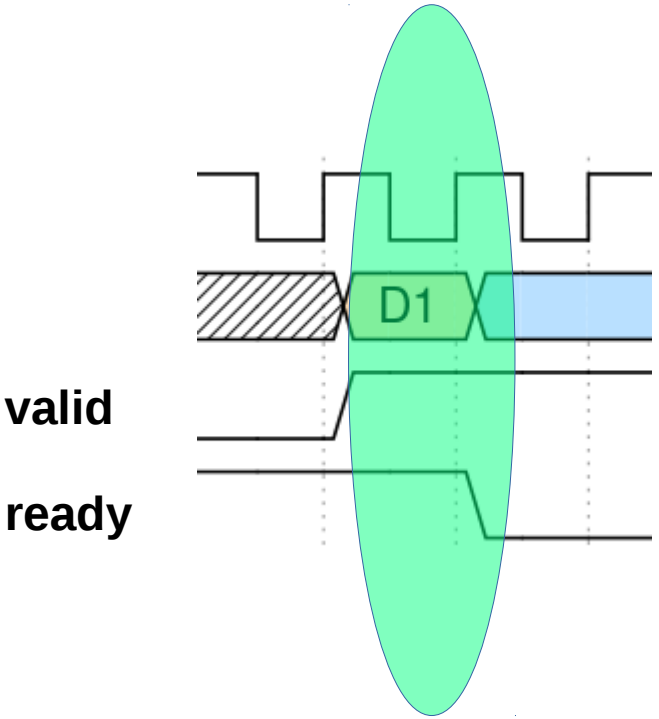
WISHBONE



В спецификации шины не указано что ACK_I может быть изначально в 1

Передача происходит только при **STB_O**=1 и **ACK_I**=1

Главная проблема valid/ready

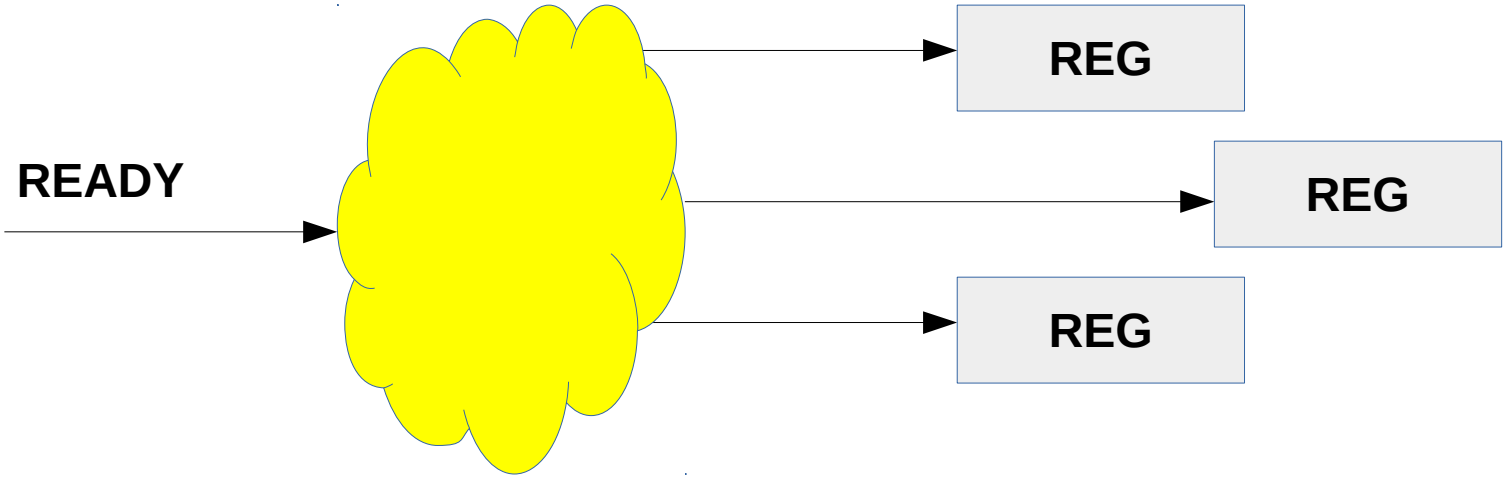


На передатчике сигнал **ready** должен быть обработан в том же самом такте.

Нет возможности подать сигнал на триггер.

Это достаточное основание для отказа от **valid/ready**

Проблемы для автомата передачи

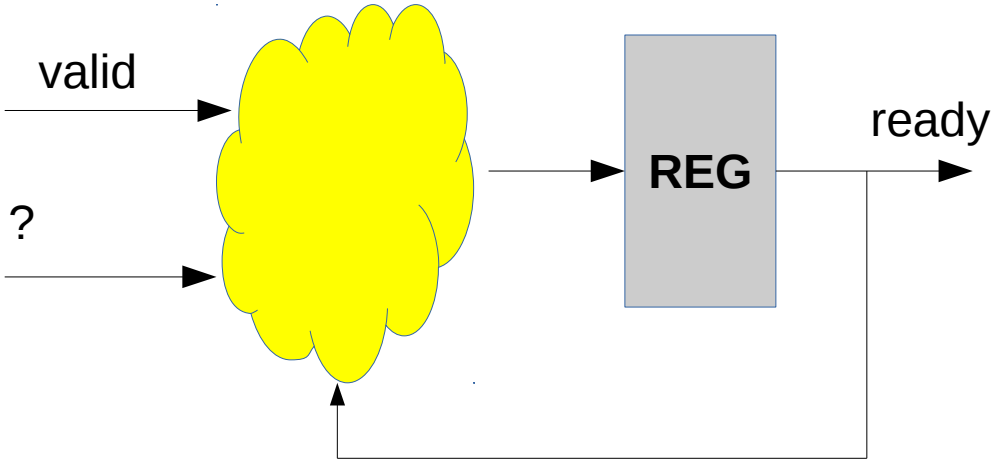


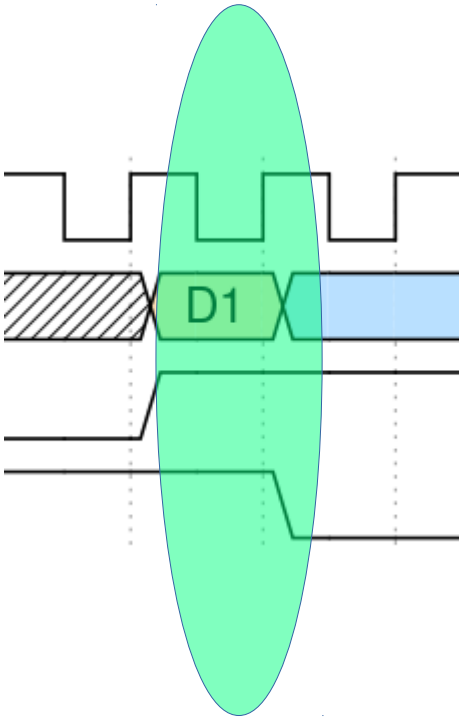
Как правило существует конечный автомат который передаёт данные на шину.

Сигнал ready должен через некую комбинационную логику попасть на все триггеры автомата.

Это создаёт большие сложности при трассировке цепей

Регистровый ready на приёмнике

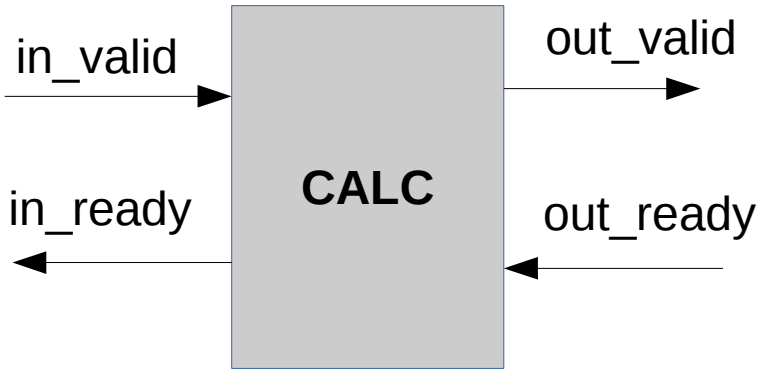


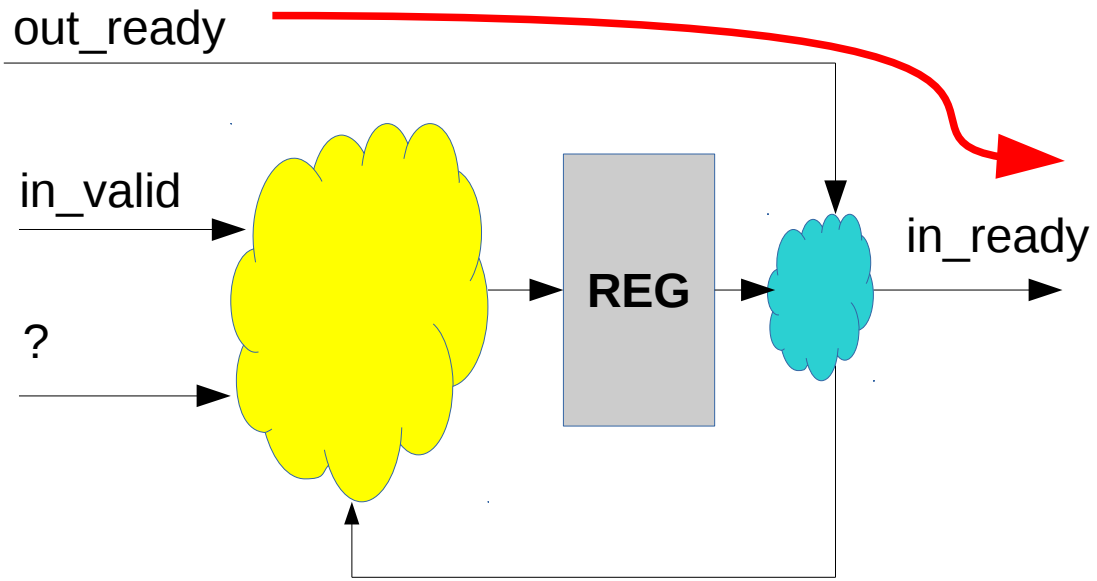


Сигнал **ready** может быть сформирован на регистре

Необходимо априорное знание о готовности последующей схемы

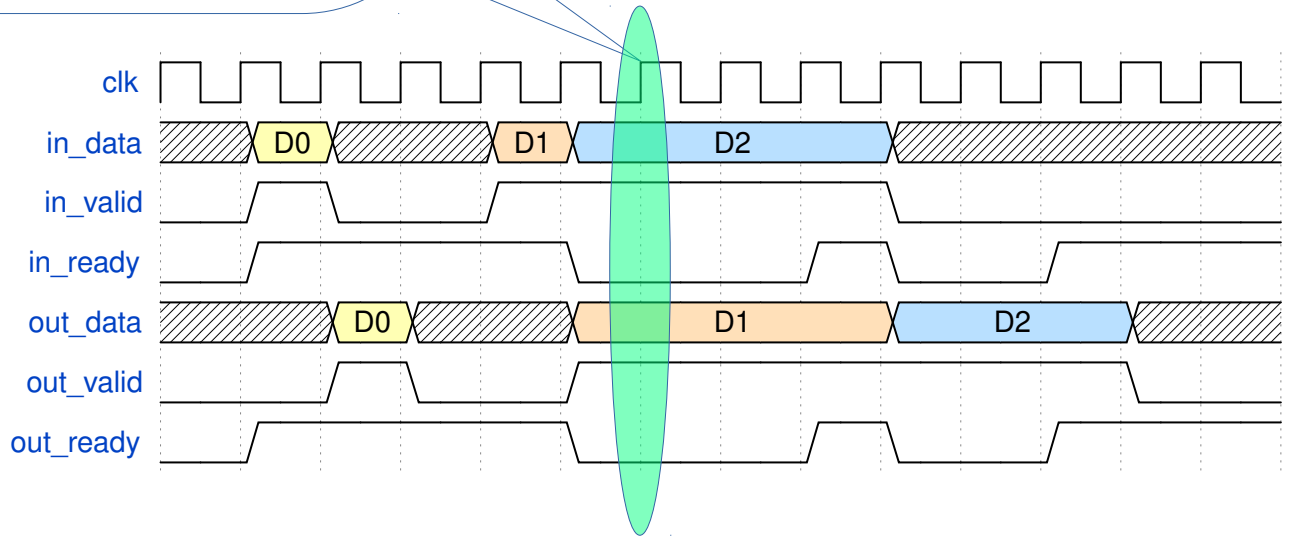
Комбинационное формирование ready



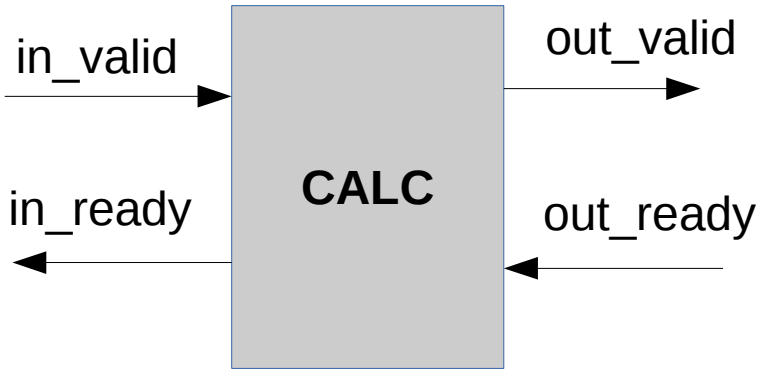


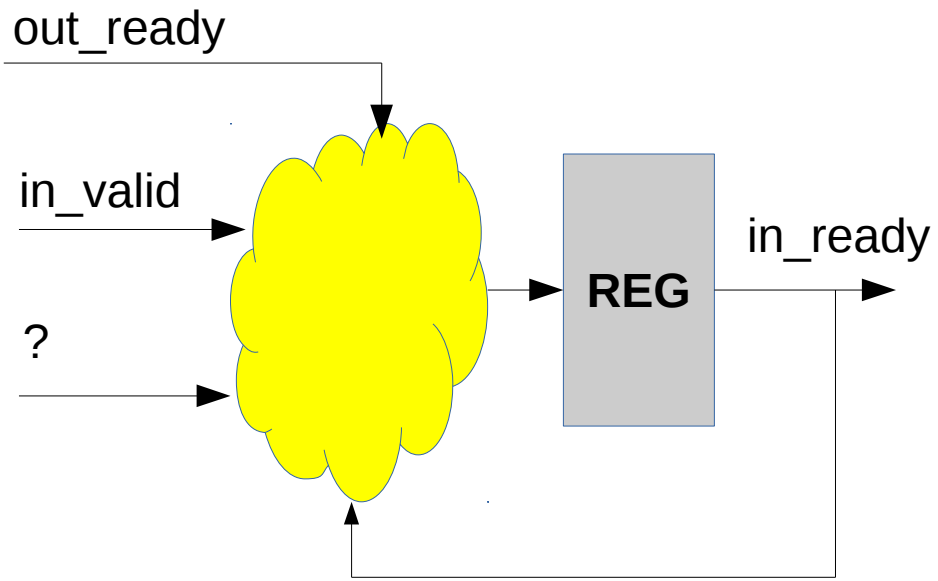
По каким то причинам приёмник не готов

Необходимо задержать передатчик

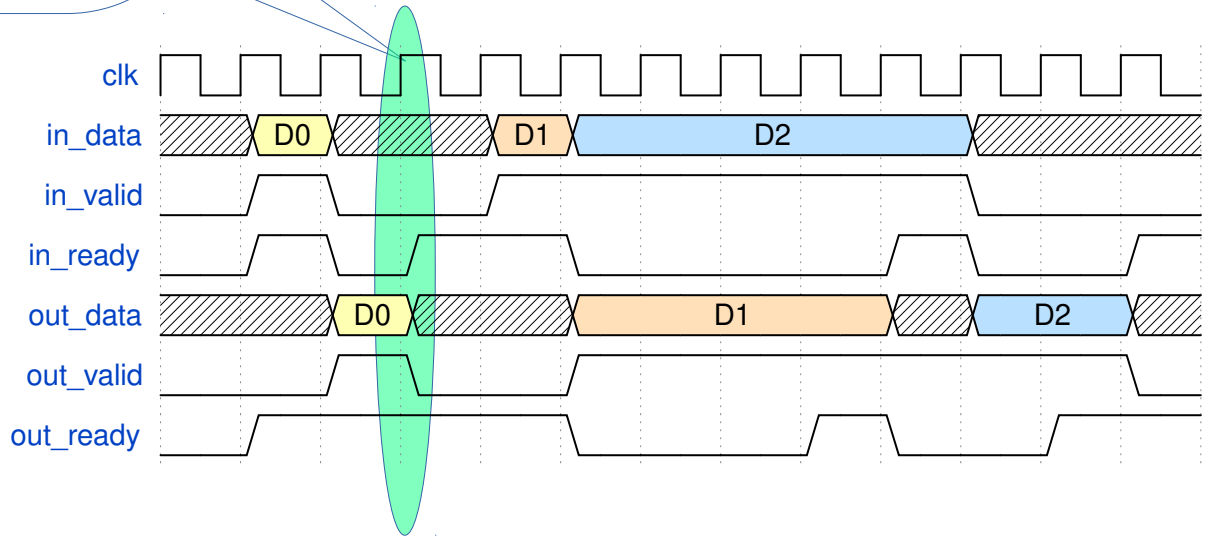


Формирование ready с паузой

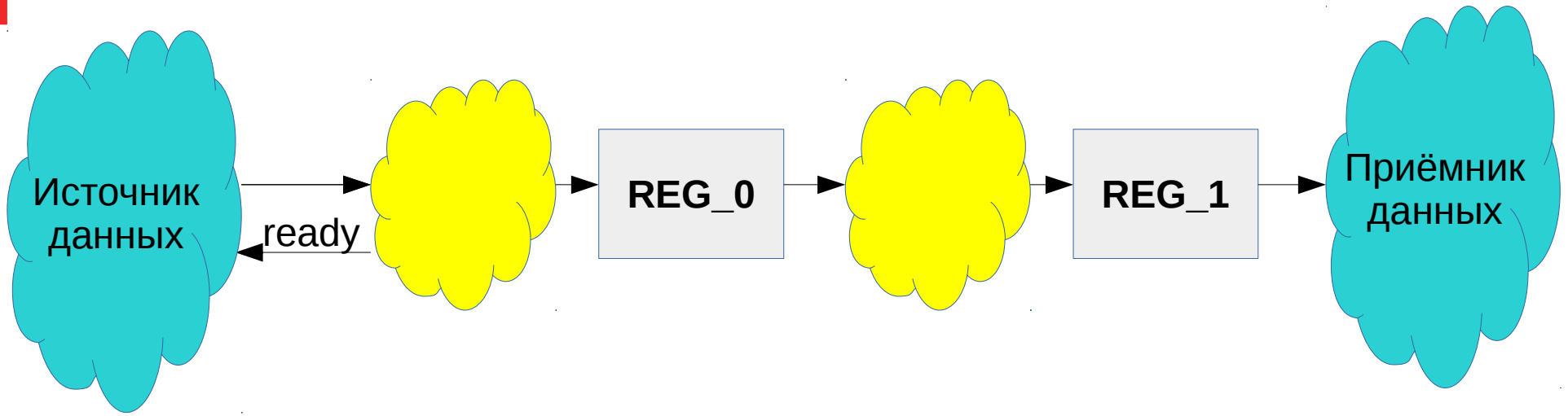




Обязательная пауза
до момента захвата
данных приёмником

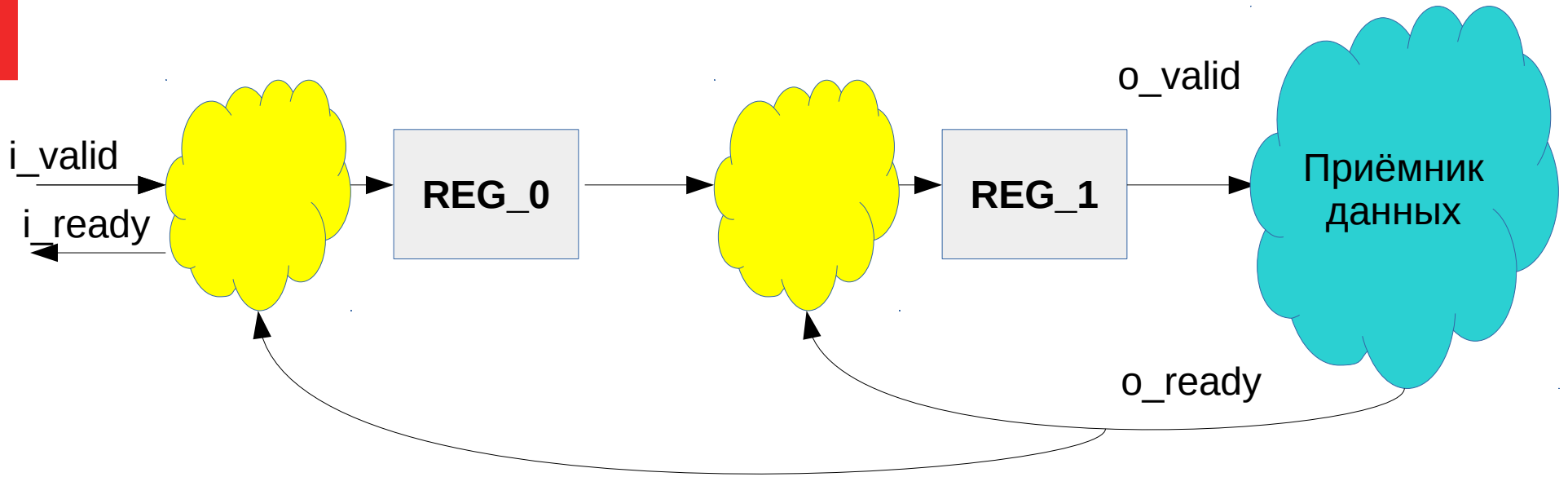


Типичный конвейер



Проблема — как сообщить источнику данных что приёмник готов или не готов к приёму данных

Вариант 1 — распространение o_ready

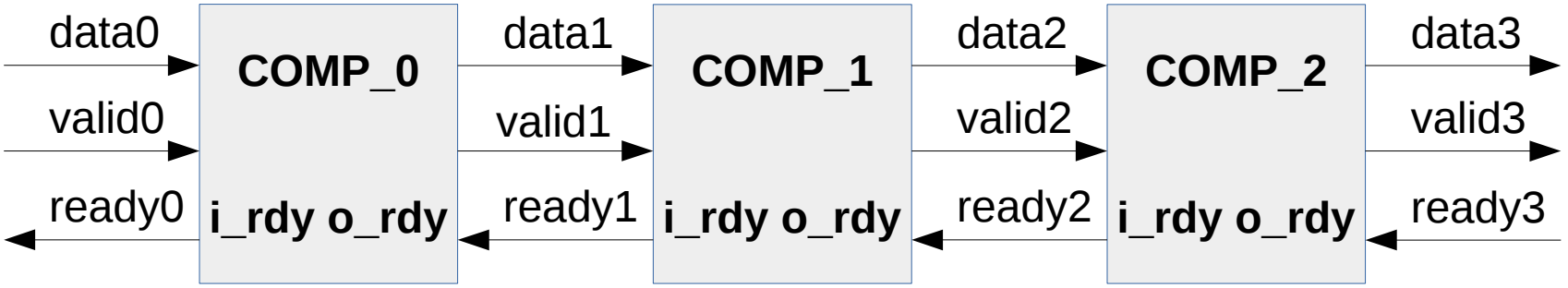


Как правило — приёмник формирует сигнал готовности к приёму данных

Сигнал **o_ready** поступает на все стадии конвейера.

Недостаток — проблемы с трассировкой

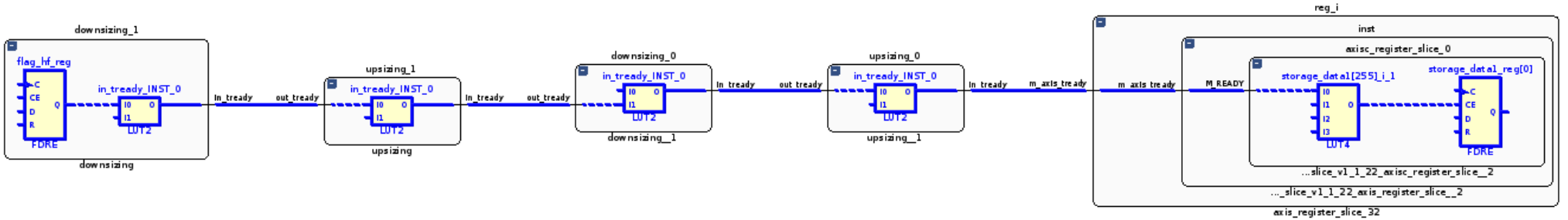
Пример каскадного соединения



Вполне возможна ситуация когда в каждом блоке сигнал **i_rdy** формируется через комбинационную схему с участием **o_rdy.**

В этом случае для сигнала **ready0** будет синтезирована очень большая комбинационная схема и будут проблемы при трассировке

Пример — downsizing и upsizing



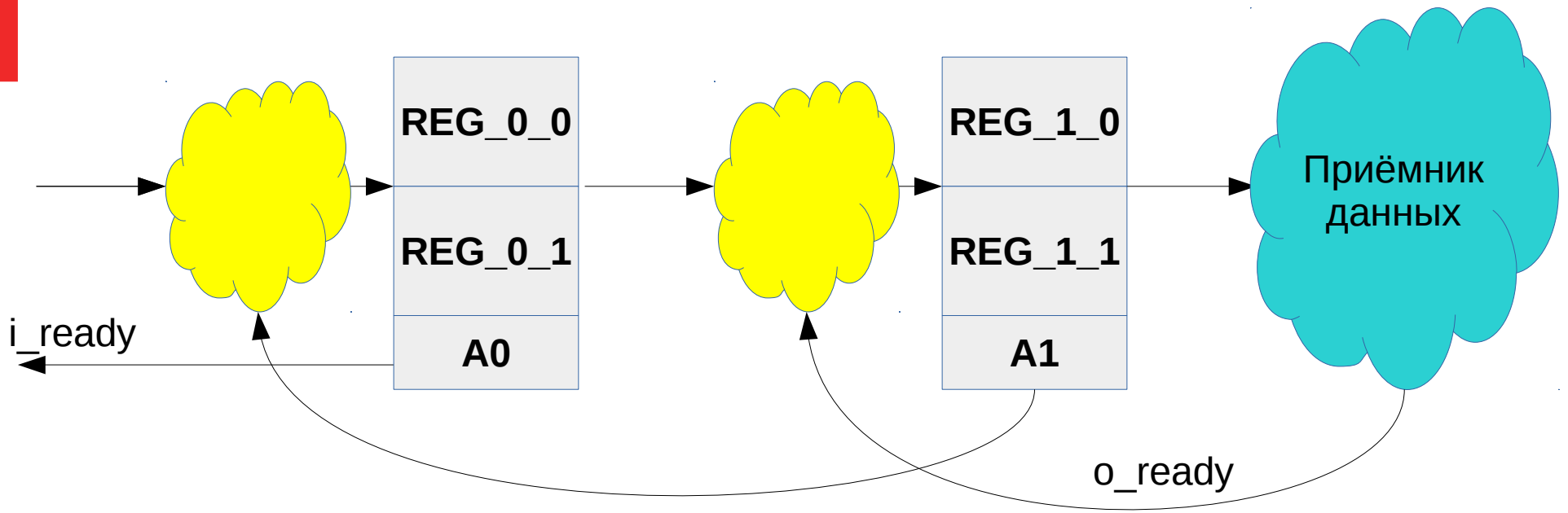
Комбинационная схема для сигнала out_ready проходит через все компоненты

Levels: 5

Fanout: 258

Slack: **-0.086** ns

Вариант 2 — двойной буфер

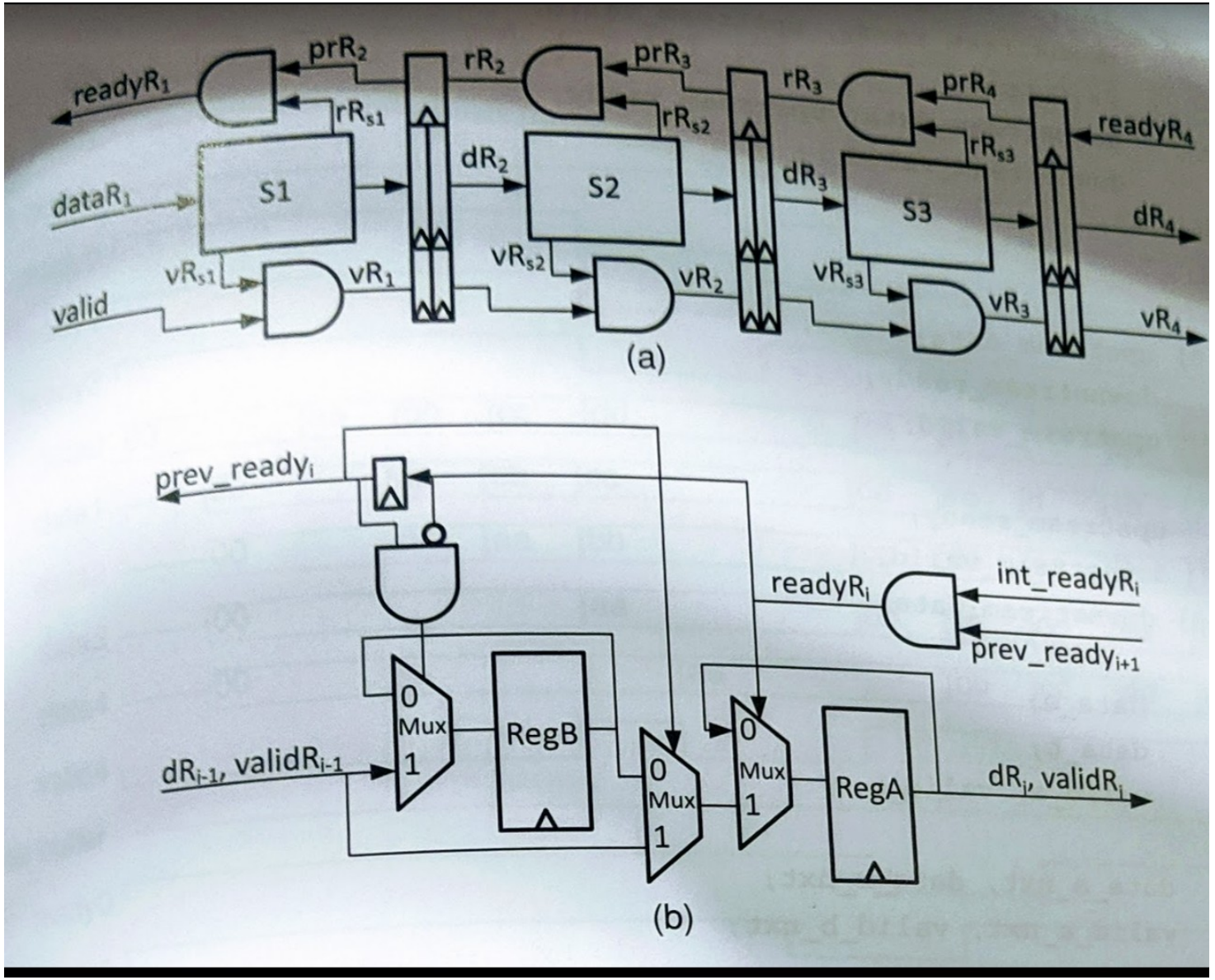


На каждой стадии конвейера используется двойной буфер.

Комбинационная схема анализирует только сигналы от соседних стадий конвейера.

Недостаток — усложнение логики вычислений

Классическая реализация

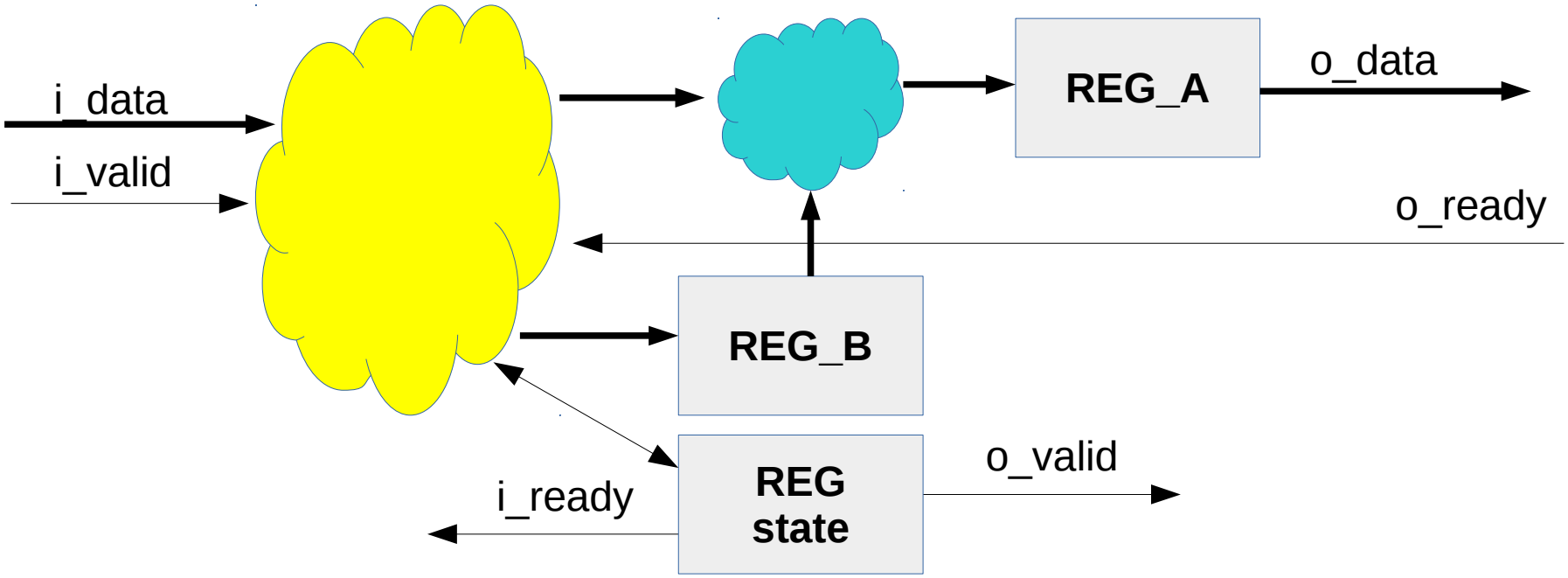


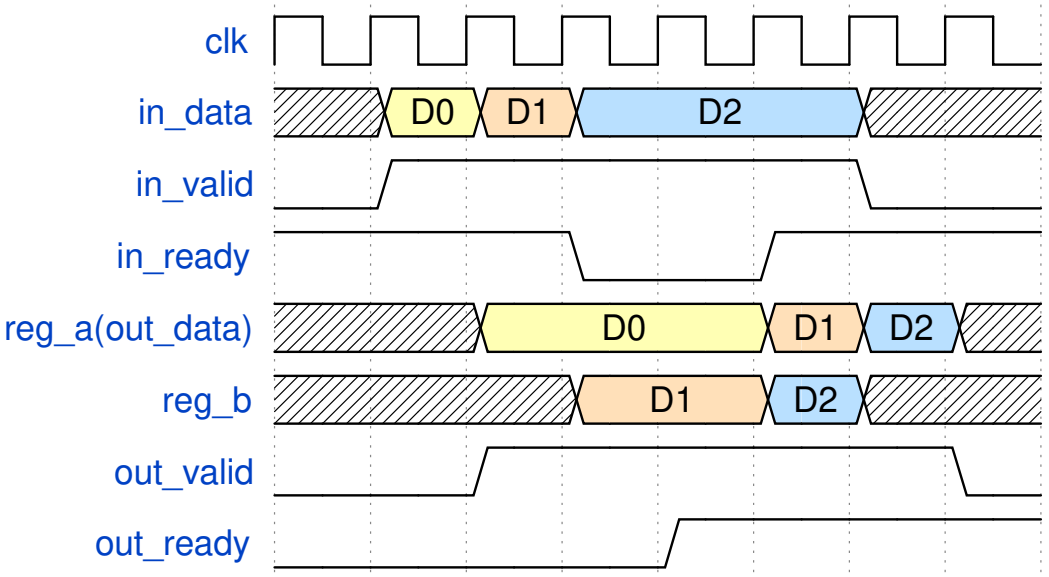
Особенности реализации

- Не используются входы CE на триггерах.
- Выход триггера заворачивается на вход через мультиплексор
- Синтезатор проводит оптимизацию с учётом доступности входа CE — получается более быстрая схема но совершенно не похожая на исходную
- gihub - пример **cascade_with_double**

- По своей сути это автомат с конечным числом состояний.
- Возможно применение общих стилей описания конечных автоматов

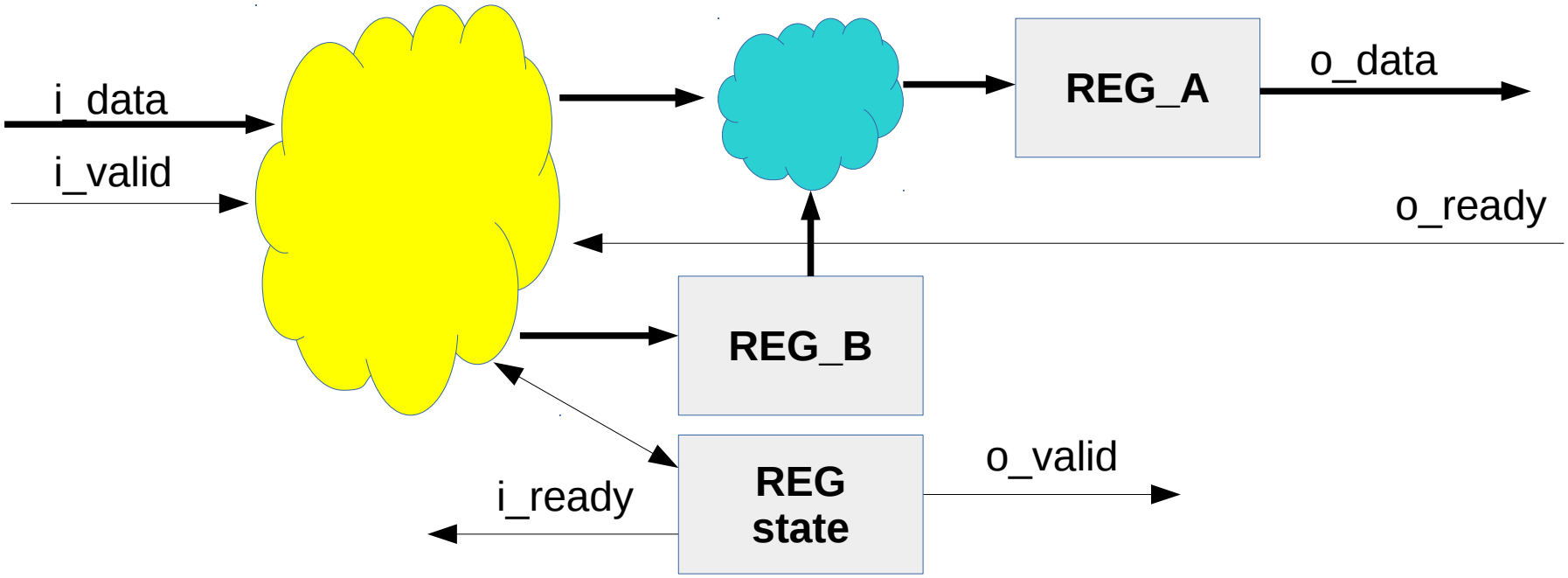
Цикл работы двойного буфера

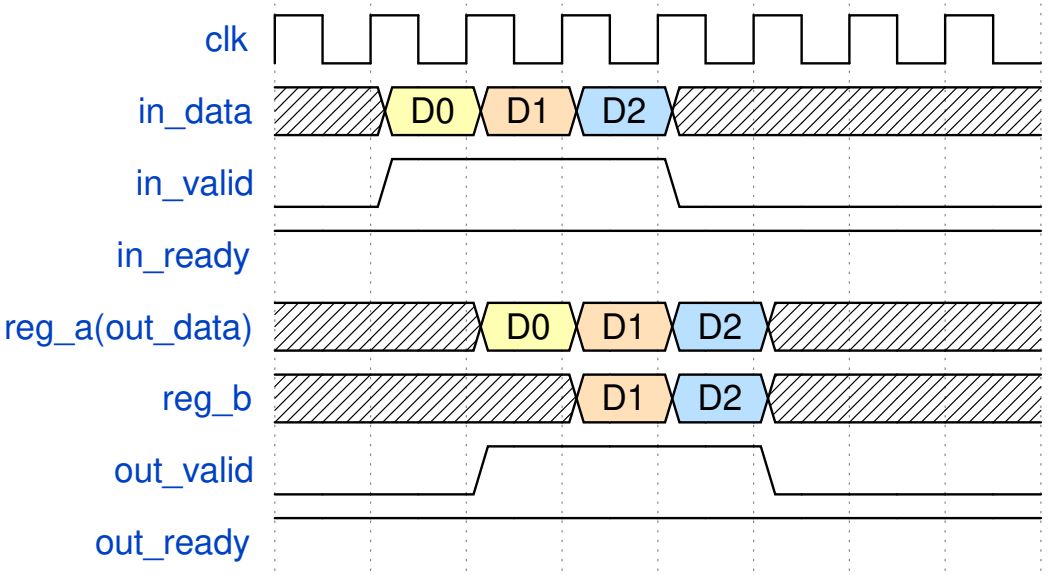




В момент приёма **D0** нет информации о том будет ли передача по шине **out_data**

Цикл работы двойного буфера





Непрерывная передача при постоянной готовности приёмника

Двойной буфер как конечный автомат

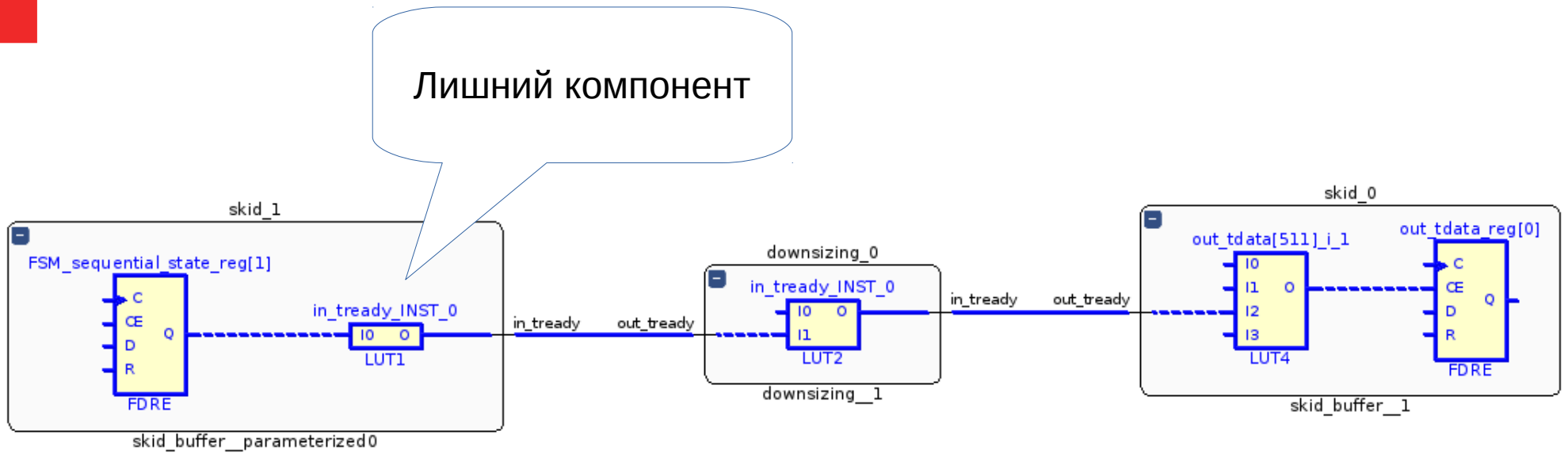
Сигналы **in_tready** и **out_tvalid** являются состоянием автомата

```
assign  is_write    = in_tvalid & in_tready_i;
assign  is_read     = out_tvalid & out_tready;

assign in_tready    = state[1];
assign out_tvalid   = state[0];
```

github — пример **skid_buffer**

```
always_ff @(posedge aclk) begin

    if( is_write )
        buf_tdata <= #1 in_tdata;

    case( state )
        2'b10: begin // empty: in_tready=1, out_tvalid=0
            out_tdata <= #1 in_tdata;
            if( is_write ) begin
                state <= #1 2'b11;
            end
            in_ready_i <= #1 '1;
        end

        2'b11: begin // half: in_tready=1, out_tvalid=1
            case( {is_write, is_read })
                2'b01: begin  // read
                        state <= #1 2'b10;
                end

                2'b10: begin   // write
                        state <= #1 2'b01;
                        in_ready_i <= #1 '0;
                end

                2'b11: begin   // write & read
                    out_tdata <= #1 in_tdata;
                end
            endcase

        end

        2'b01: begin    // full: in_tready=0, out_tvalid=1
               if( is_read ) begin
                  state <= #1 2'b11;
                  out_tdata <= #1 buf_tdata;
                  in_ready_i <= #1 '1;
               end
        end

    endcase
```

Пример — downsizing, upsizing и skid_buffer



Комбинационная логика ограничена

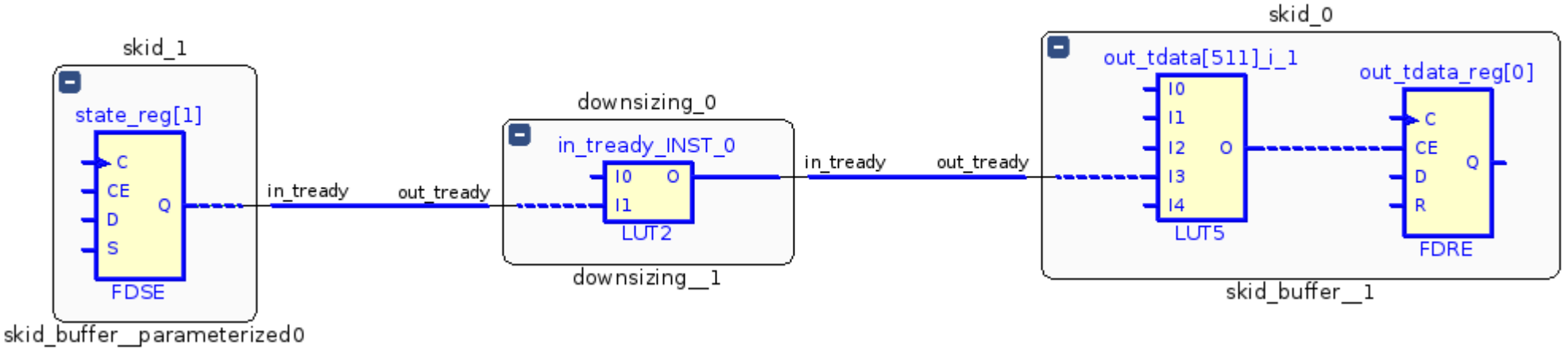
Levels: 2

Fanout: 512

Slack: **0.322** ns

Очень полезно изучать результаты синтеза !

Оптимизация примера skid_buffer



Уменьшено количество комбинационных уровней

Levels: **2**

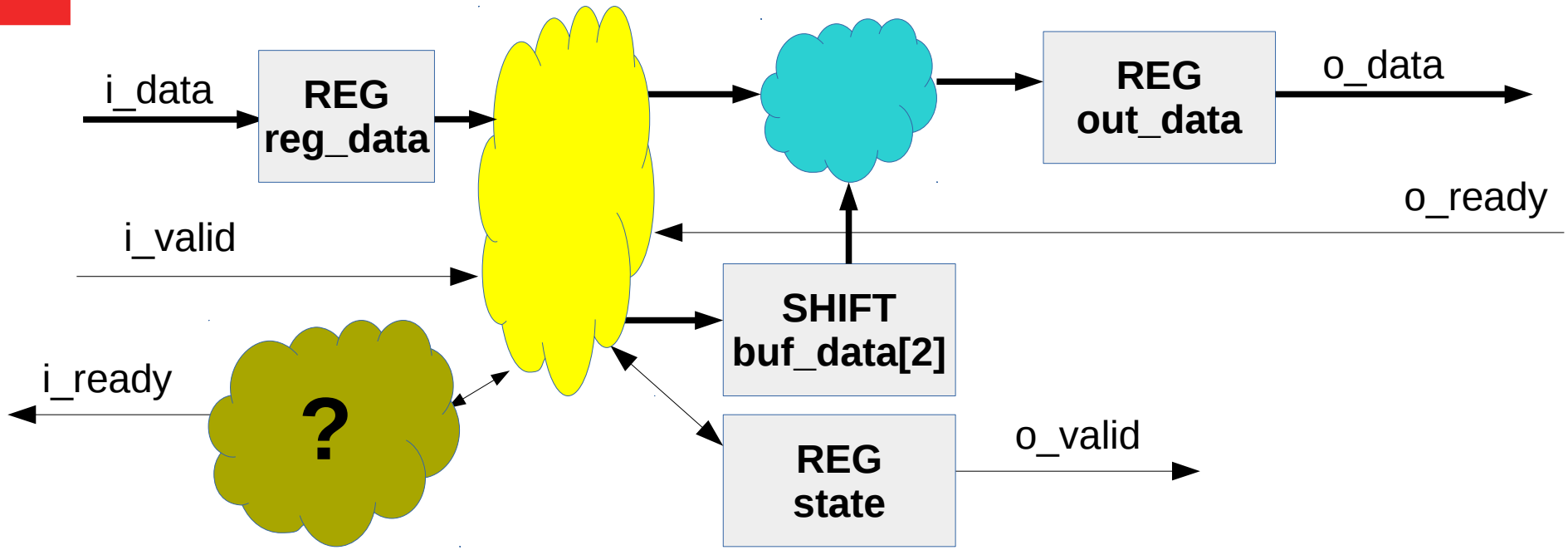
Fanout: 512

Slack: **0.547** ns

Сигнал **in_tready** разделён на два — внутренний и внешний

Добавлены атрибуты **keep**

Как отделить вход ?

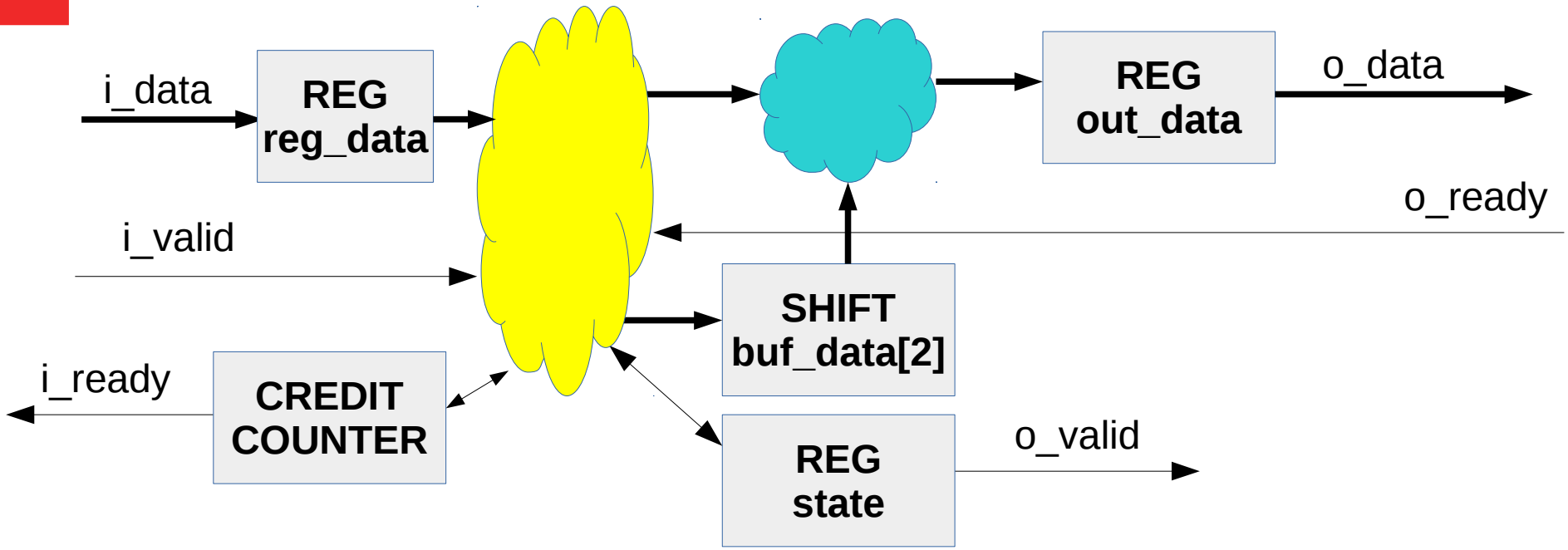


Цель — поставить на входе **i_data** простой регистр без управления.

Проблема — как сформировать **i_ready** ?

Априорные данные — сколько слов может быть запомнено если нет чтения

skid_crd — используем счётчик кредитов



Новый компонент **CREDIT_COUNTER,** размерность 3 бита

- Начальное значение: **3`b110**
- При записи слова — счётчик уменьшается на 1
- При чтении слова — счётчик увеличивается на 1
- **i_ready = CREDIT_COUNTER[2]**

Работа кредитного счётчика

Чтение запрещено

Op_w	crd_cnt	i_ready
WRITE	110	1
WRITE	101	1
WRITE	100	1
PAUSE	**011**	**0**

o_valid	o_ready
0	0
0	0
1	0
1	0

Передача останавливается после запоминания трёх слов во внутренних регистрах

Работа кредитного счётчика

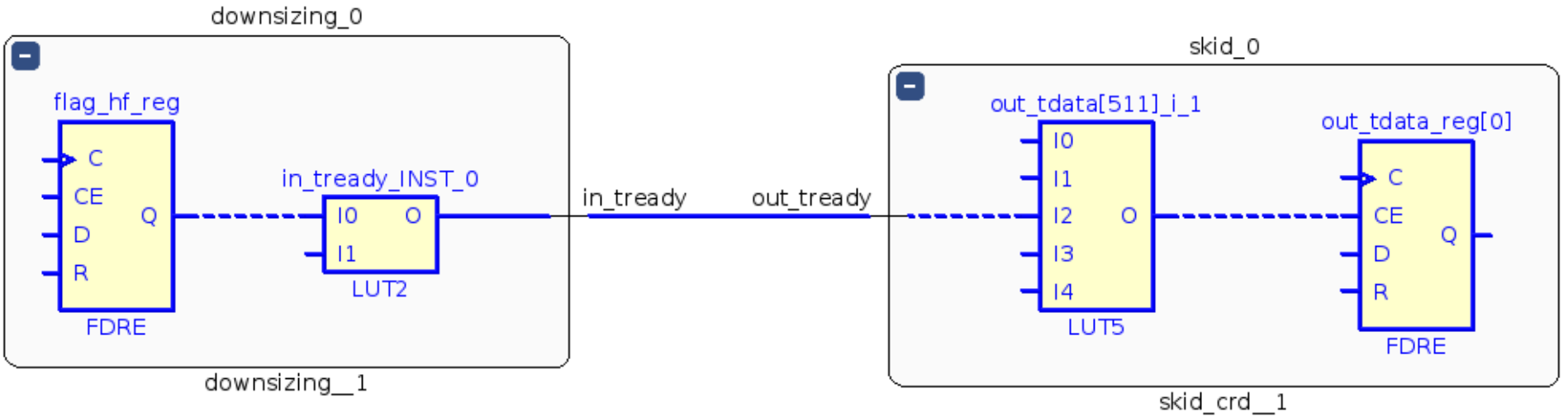
Чтение разрешено

Op_w	crd_cnt	i_ready
WRITE	110	1
WRITE	101	1
WRITE	100	1
WRITE	**100**	1
WRITE	**100**	1

o_valid	o_ready
0	1
0	1
1	1
1	1
1	1

Достигнута непрерывная передача данных с задержкой на два такта

Пример — downsizing, upsizing и skid_crd



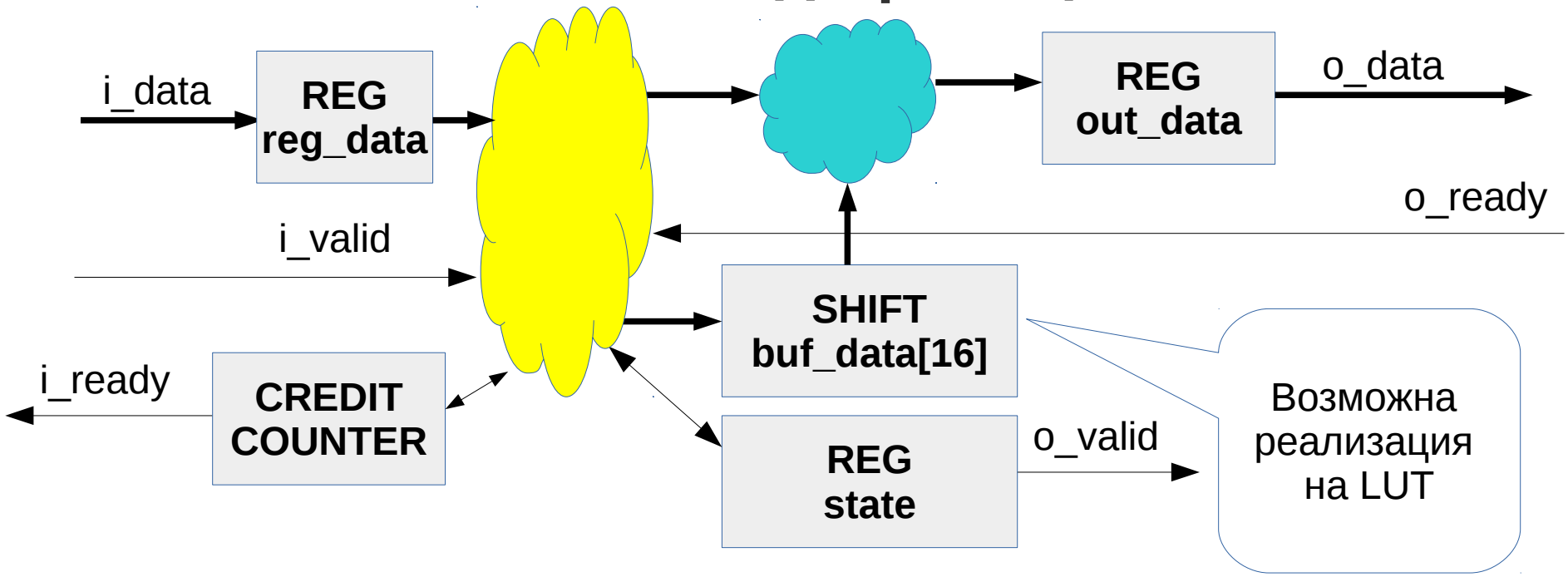
Комбинационная логика ограничена

Levels: 2

Fanout: 512

Slack: **0.581** ns

skid_crd — возможная модификация



SHIFT — это сдвиговый регистр, для ПЛИС Xilinx может быть реализован на одном LUT. Размерность до 16 бит

По своей сути это уже обычное FIFO

Сравнение примеров

Название	Slack	LUT	FF	NET
cascade	**-0.086**	1058	1552	5446
+ **skid_buffer**	0.322	2376	4132	11925
+ double_buffer	0.459	2362	4133	11135
+ **skid_crd**	0.581	2380	6702	14502

Общий вывод — буферизация позволяет увеличить быстродействие схемы но за счёт увеличения занятых ресурсов.

Исключение — skid_buffer и double_buffer

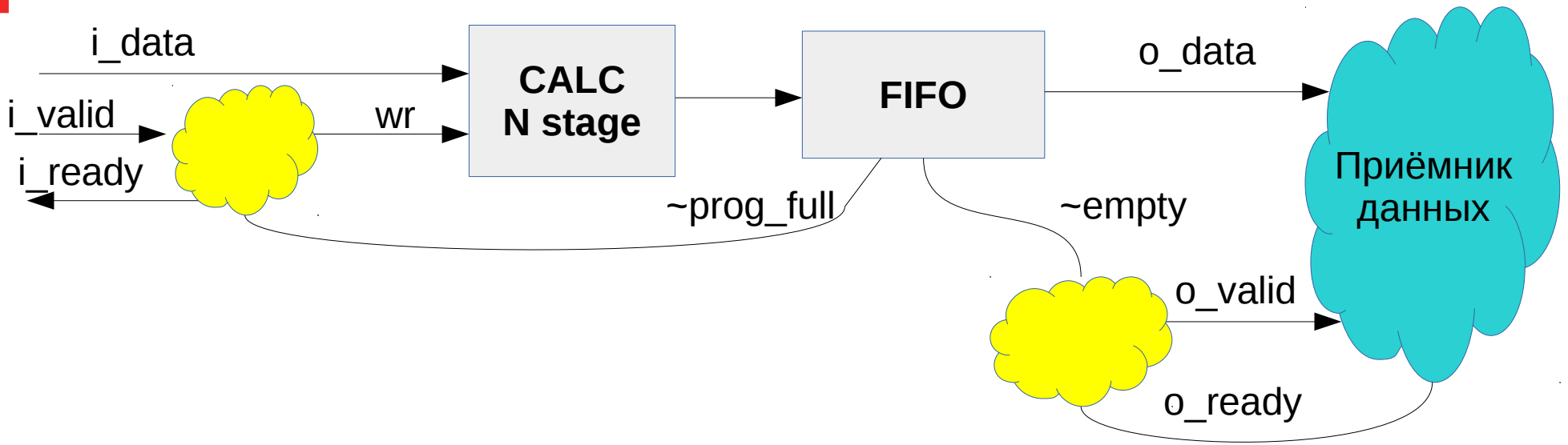
Сравнение примеров - модификация

Название	Slack	LUT	FF	NET
cascade	**-0.086**	1058	1552	5446
+ double_buffer	0.459	2362	4133	11135
+ skid_buffer	0.541	2359	4121	11900
+ **skid_crd**	0.581	2380	6702	14502

skid_buffer — небольшая оптимизация, добавлены атрибуты **keep** на некоторые цепи. Результат синтеза изменился.

А можно ли увеличить быстродействие и не увеличивать занимаемые ресурсы ?

Вариант 3 — FIFO



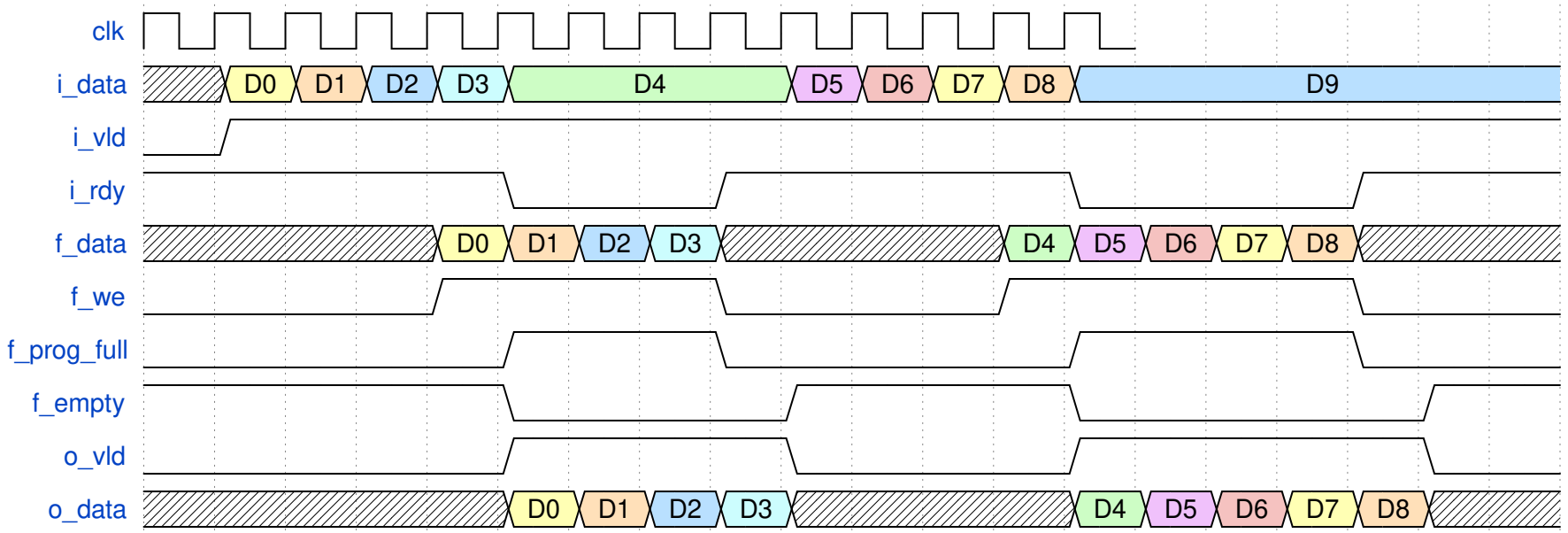
Главная идея — отказ от **valid/ready** внутри конвейера

N — число стадий конвейера

Какой оптимальный размер FIFO ?

prog_full — флаг почти полного FIFO, как минимум равен N

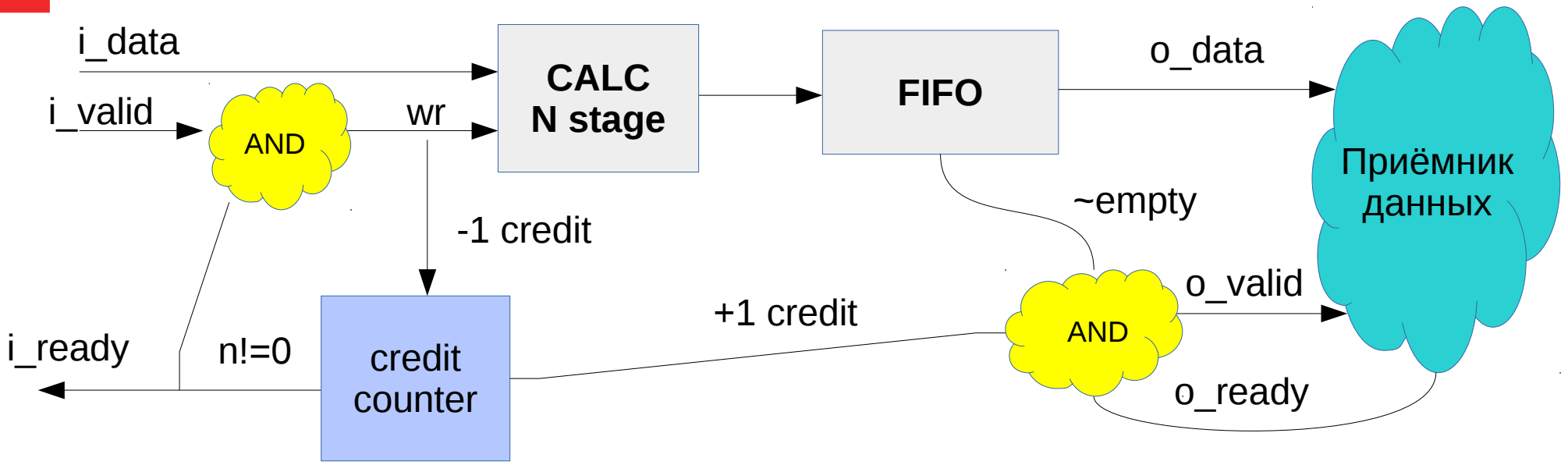
Размер FIFO 4 слова, конвейер 3 такта



При постоянной записи и при постоянном разрешении чтения возникают паузы при работе конвейера.

Оптимальный размер FIFO равен 2N

Вариант 4 — счётчик кредитов



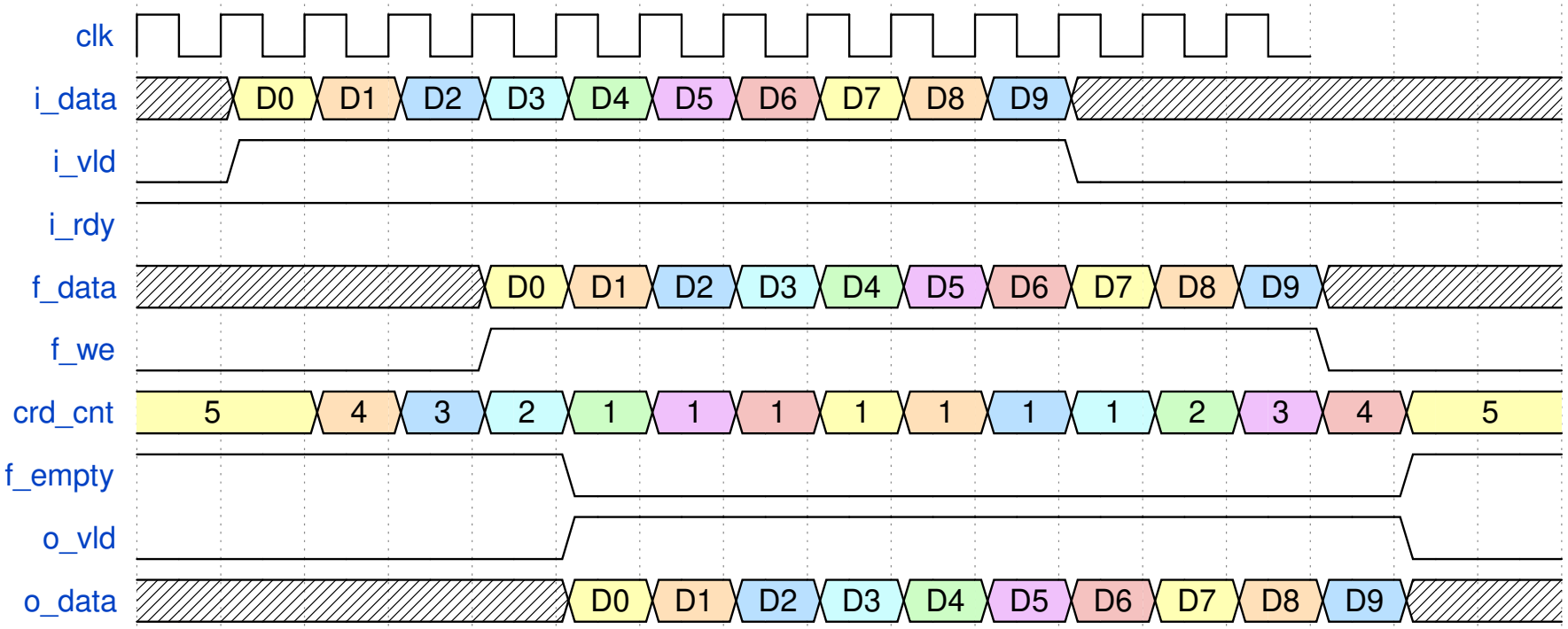
Начальное значение счётчика кредитов равно размеру FIFO

Оптимальный размер FIFO равен N+2

На вход узла CALC подаётся столько данных, сколько есть места в FIFO

github — пример **credit**

Размер FIFO 5 слов, конвейер 3 такта



Начальное значение счётчика кредитов равно размеру FIFO

Оптимальный размер FIFO равен N+2

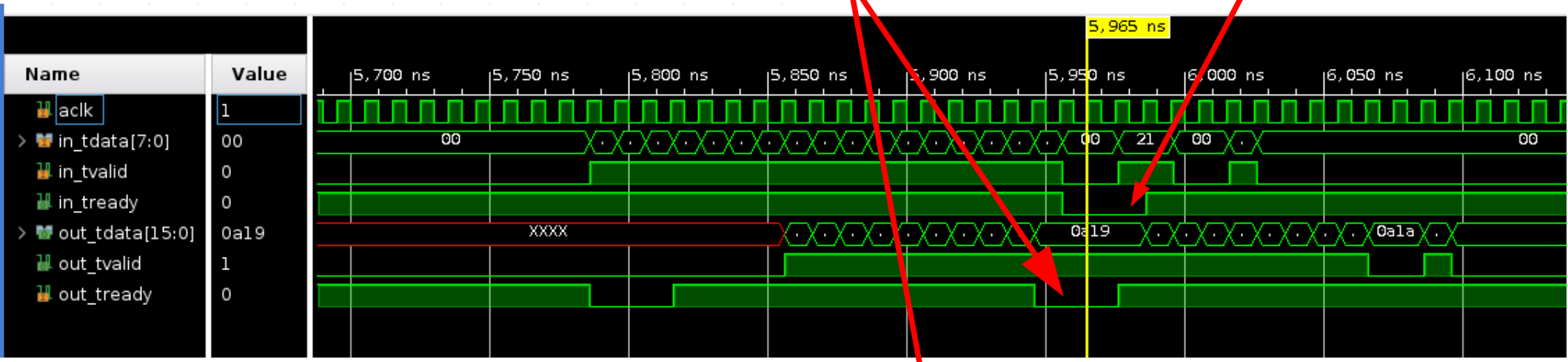
На вход узла CALC подаётся столько данных, сколько есть места в FIFO

Сравнение примеров credit и conv_with_fifo

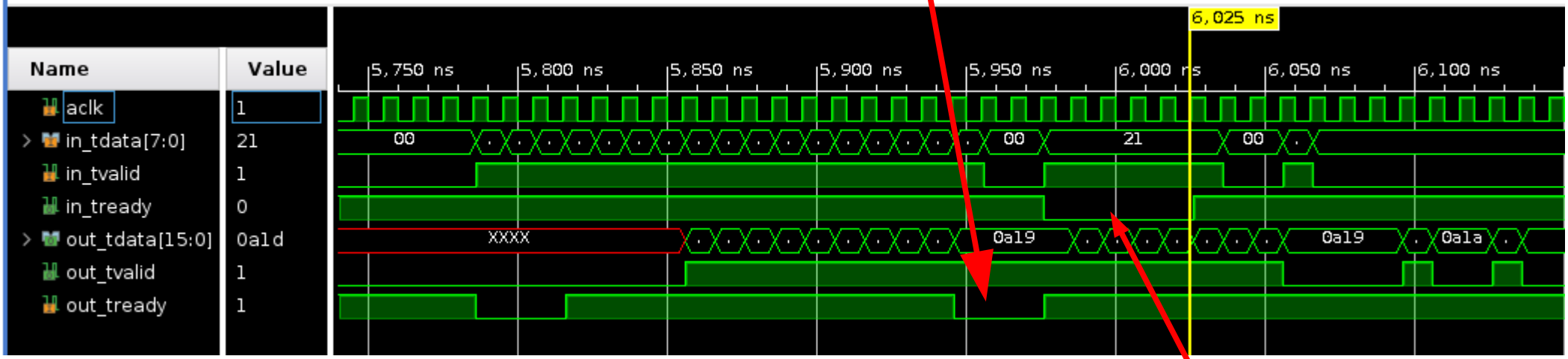
Пример **credit**

Пауза приёма в три такта

Пауза в три такта



Пример **conv_with_fifo**



Пауза в пять тактов

Задержка памяти 6 тактов. FIFO размером 8 слов

Статистика примеров credit и conv_with_fifo

Пример **credit**

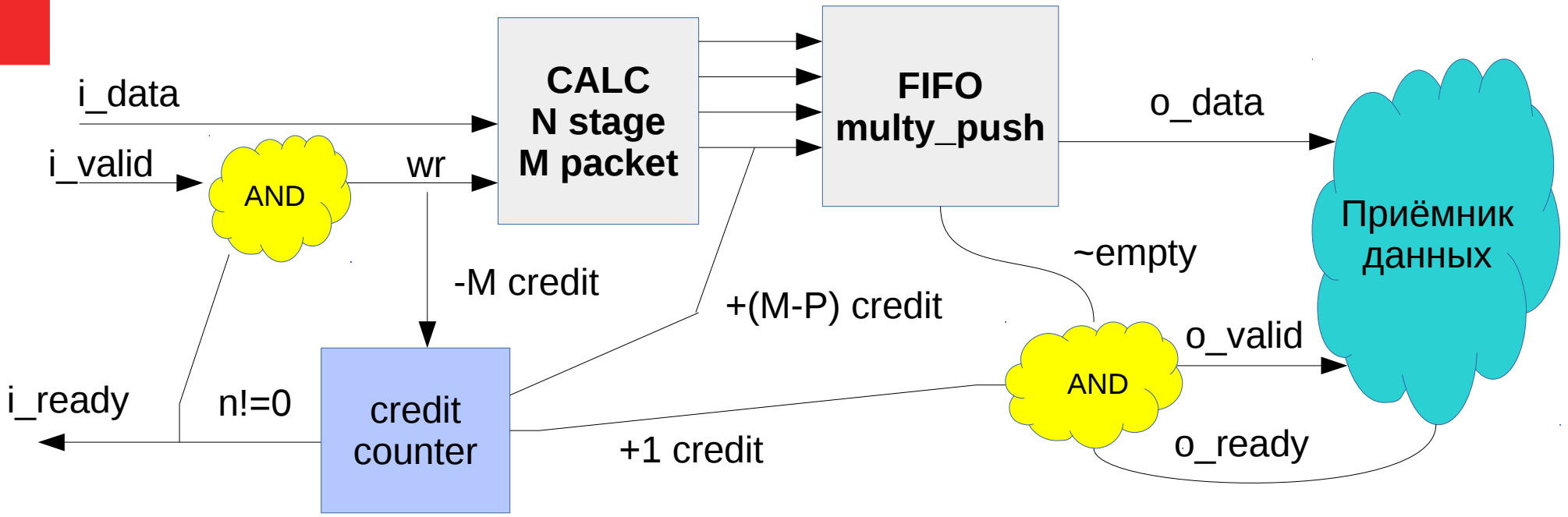
```
Statistic:  min_delay:  8   max_delay: 81   avr_delay: 21.032193   velocity: 0.195746

test_id=    1 test_name:       randomize         TEST_PASSED
```

Пример **conv_with_fifo**

```
Statistic:  min_delay:  8   max_delay: 62   avr_delay: 18.008032   velocity: 0.234574

test_id=    1 test_name:       randomize         TEST_PASSED
```

Увеличение скорости ~19 % при одинаковом размере FIFO

Счётчик с возвратом кредитов



Узел CALC может формировать пакеты для записи в FIFO

Максимальный размер пакета равен **M**

Размер пакета **P** известен только через **N** тактов

FIFO может одновременно принять M слов

Чтение из FIFO всегда по одному слову

Работа кредитного счётчика

Поступают данные на вход.

- Кредитный счётчик уменьшается на максимально возможный размер. (пессимистическая оценка)
- Если счётчик меньше либо равен нулю то приём данных останавливается

Через N тактов узел CALC сообщает реальное число P которое будет записано в FIFO

- Кредитный счётчик увеличивается на число (M-P) (реальная оценка)

Приёмник данных вычитывает слово из FIFO

- Кредитный счётчик увеличивается на единицу

Приём данных разрешён при значении кредитного счётчика больше нуля

github — пример **credit_return**

Сравнение FIFO и кредитного счётчика

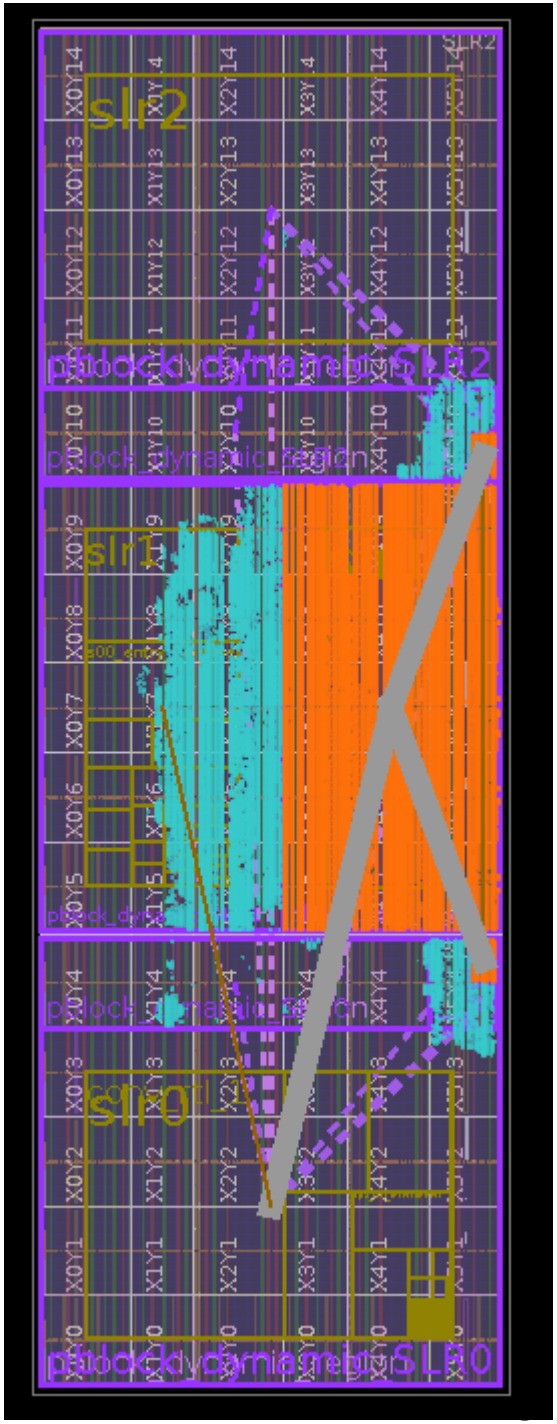
Реализация через FIFO	Реализация черз кредитный счётчик
Приостановка записи по флагу FIFO prog_full	Приостановка при отрицательном значении кредитного счётчика
Неизвестно сколько данных уже в пути	Есть информация о количестве данных. Первая оценка пессимистическая, вторая реальная
FIFO должно иметь место для приёма N пакетов размера M	FIFO должно иметь место для приёма M+2 слова

Проблемы длинных передач

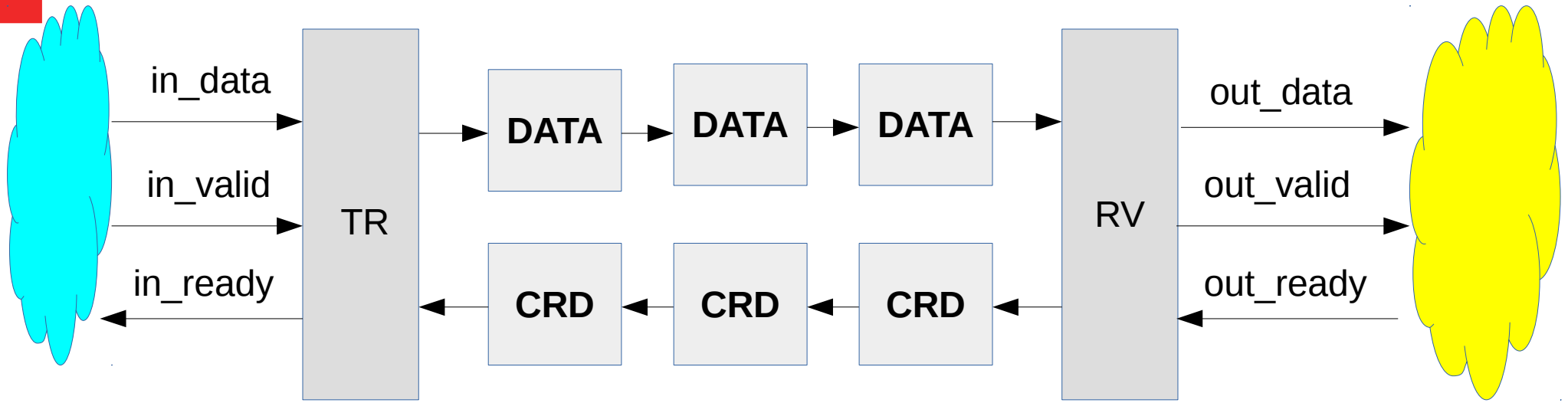
Современные кристаллы FPGA и ASIC очень большие.

Связать два узла которые находятся в разных концах кристалла без дополнительной буферизации невозможно.

На слайде ALVEO U200



Вариант шины на основе кредитов



Приёмник передаёт передатчику число данных которое он может принять.

Передатчик передаёт только разрешённое число данных

Простая буферизация на регистрах

Аналоги:

- протокол TCP
- система NoC (Network on Chip) для Xilinx Versal ACAP

Примеры на github

skid_buffer - Двойной буфер

skid_crd — буфер со счётчиком кредитов

cascade — каскадное соединение компонентов

cascade_with_double — каскадное соединение с буферизацией на double_buffer

cascade_with_skid — каскадное соединение с буферизацией на skid_buffer

conv_with_fifo — конвейер с FIFO на выходе

credit - Кредитный счётчик

credit_return - Счётчик с возвратом кредитов

Заключение

Не существует идеального решения подходящего для всех проектов.

- Конвейеризация — один из способов повышения быстродействия
- Двойная буферизация — один из способов отслеживания сигнала готовности приёмника
- Буфер со счётчиком кредитов позволяет изолировать вход компонента
- FIFO после конвейерного вычислителя позволяет упростить архитектуру вычислителя и уменьшить занимаемые ресурсы
- Кредитные счётчики позволяют уменьшить размер FIFO