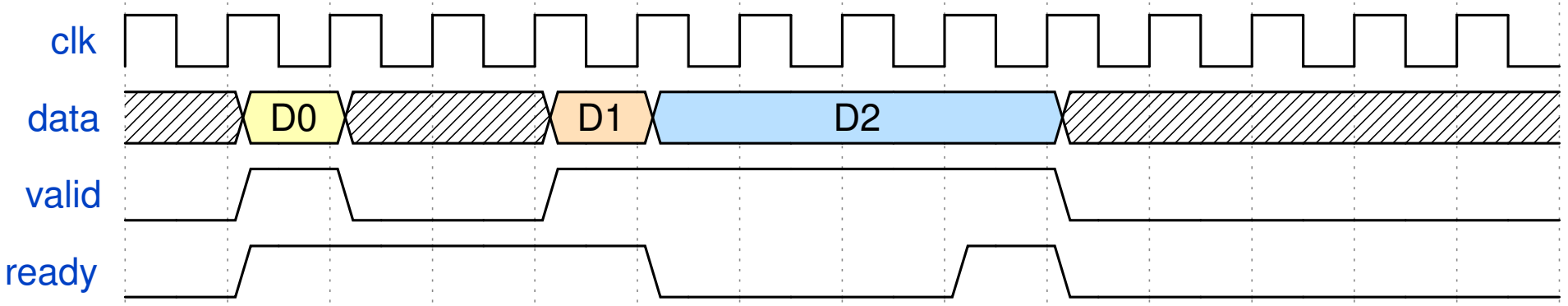
Интерфейс valid/ready
Двойные буфера
Счётчики кредитов

Основные цели

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

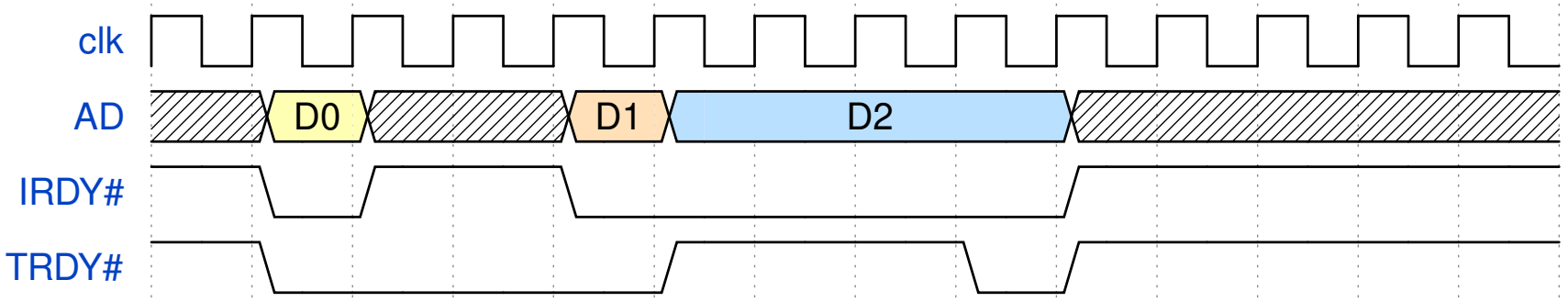
Интерфейс valid/ready



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

Внешние шины

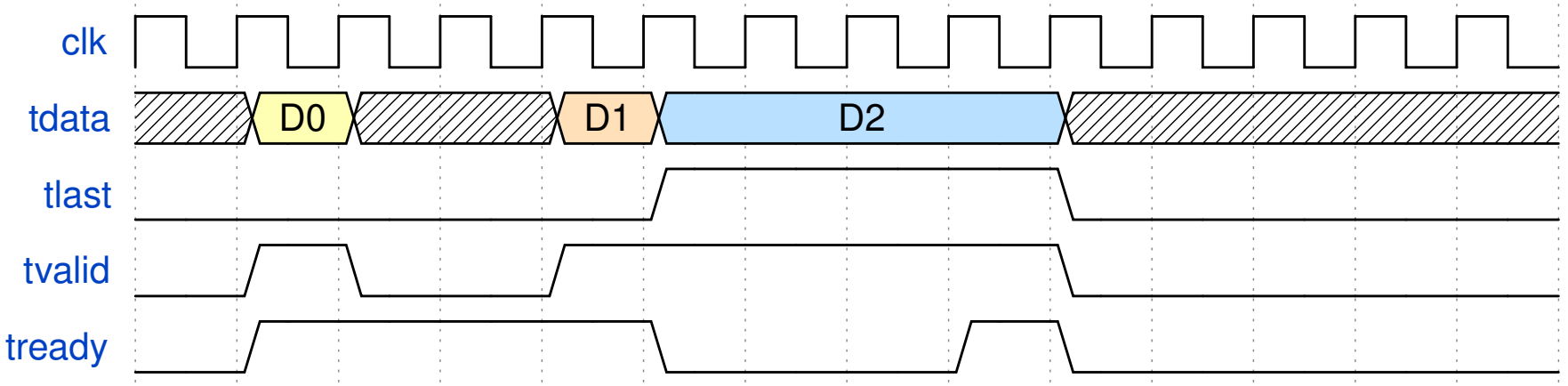
Подобный интерфейс используется в различных внешних шинах, таких как ISA, PCI, Compact Flash



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

- IRDY# = VALID
- TDRY# = READY

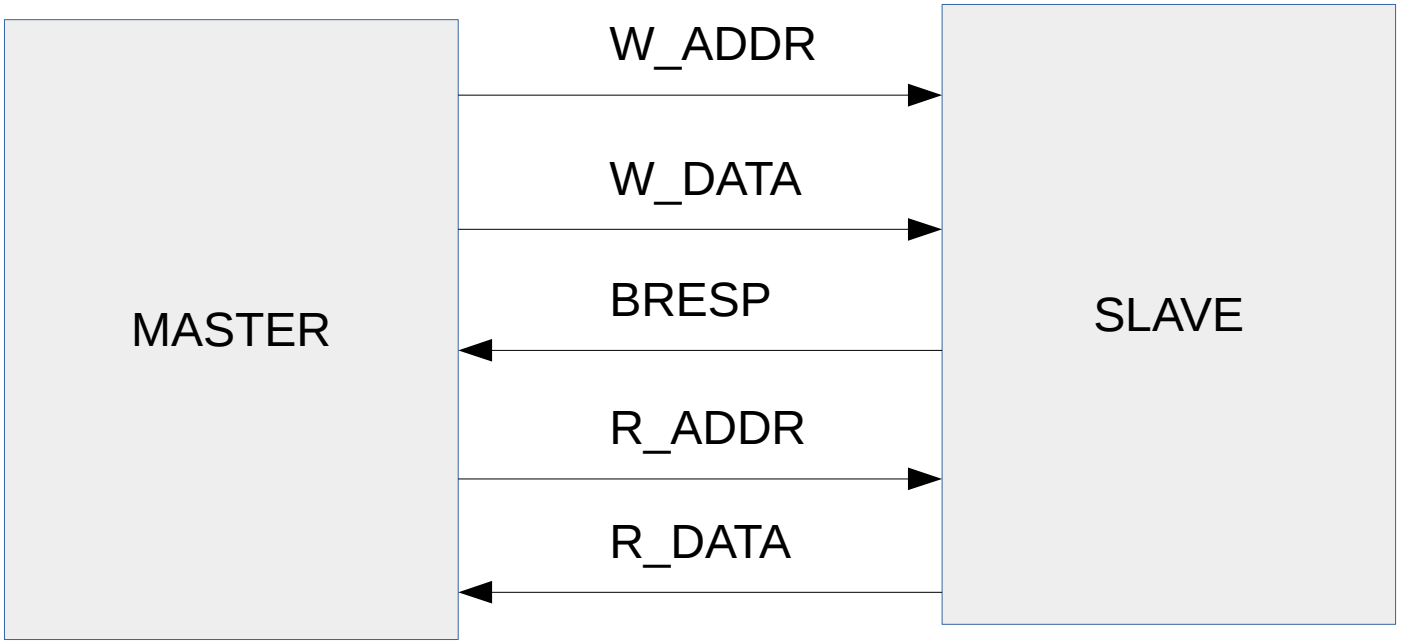
Внутренние шины - AXI STREAM



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

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

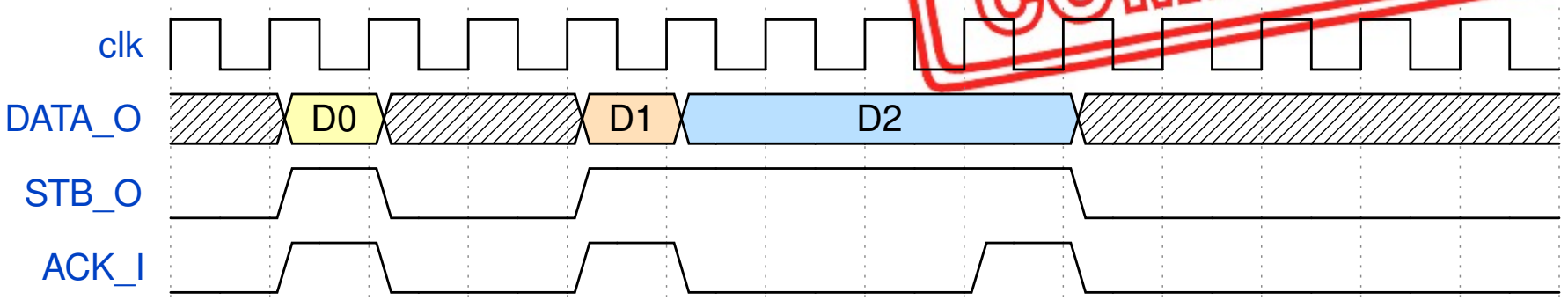
AXI MEMORY MAP - пять шин



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

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

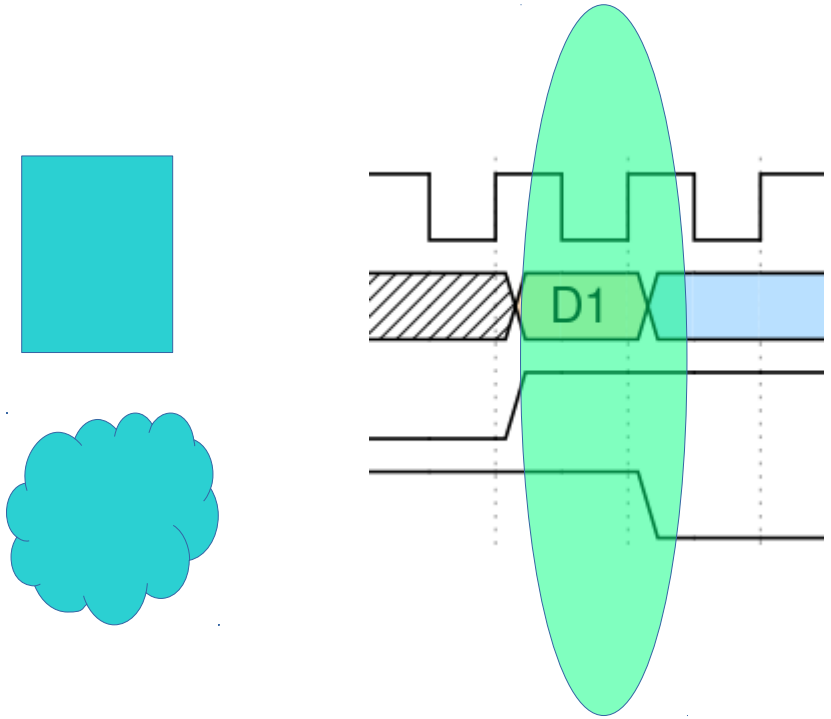
WISHBONE



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

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

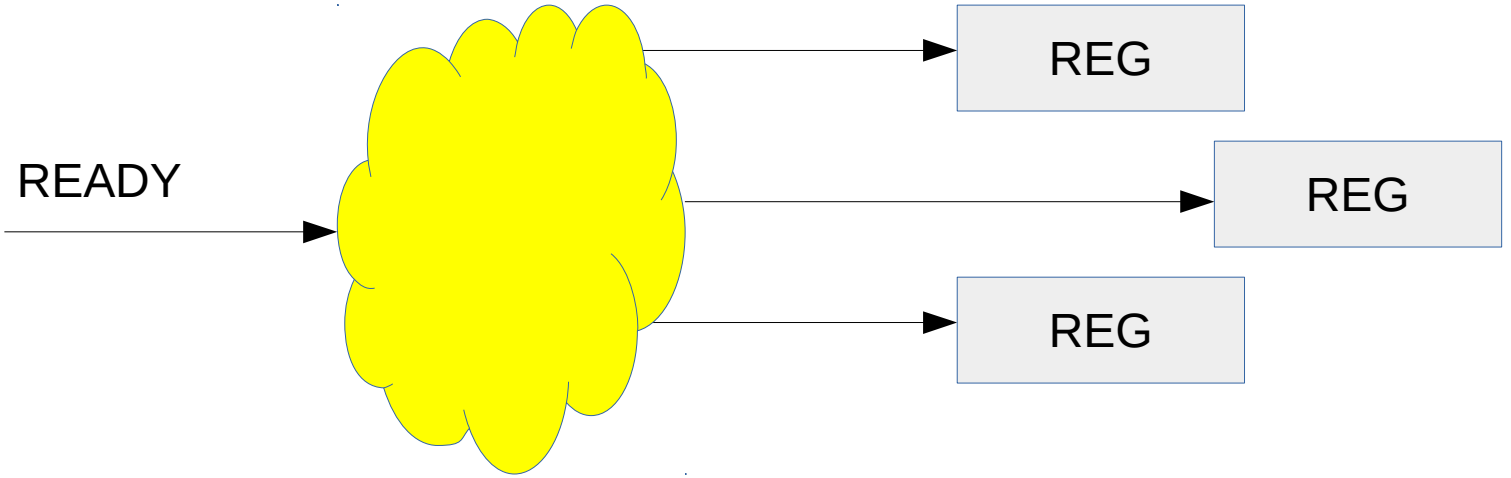
Главная проблема valid/ready



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

Нет возможности подать сигнал на триггер.

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.

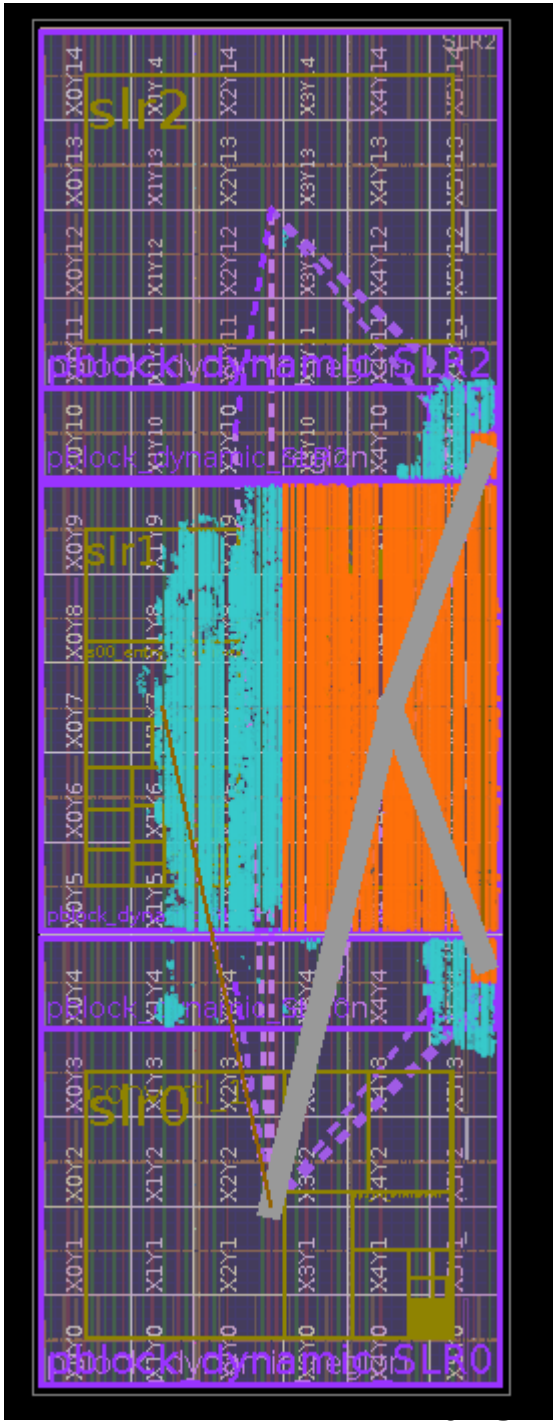
Это создаёт большие сложности при трассировке цепей

Проблемы длинных передач

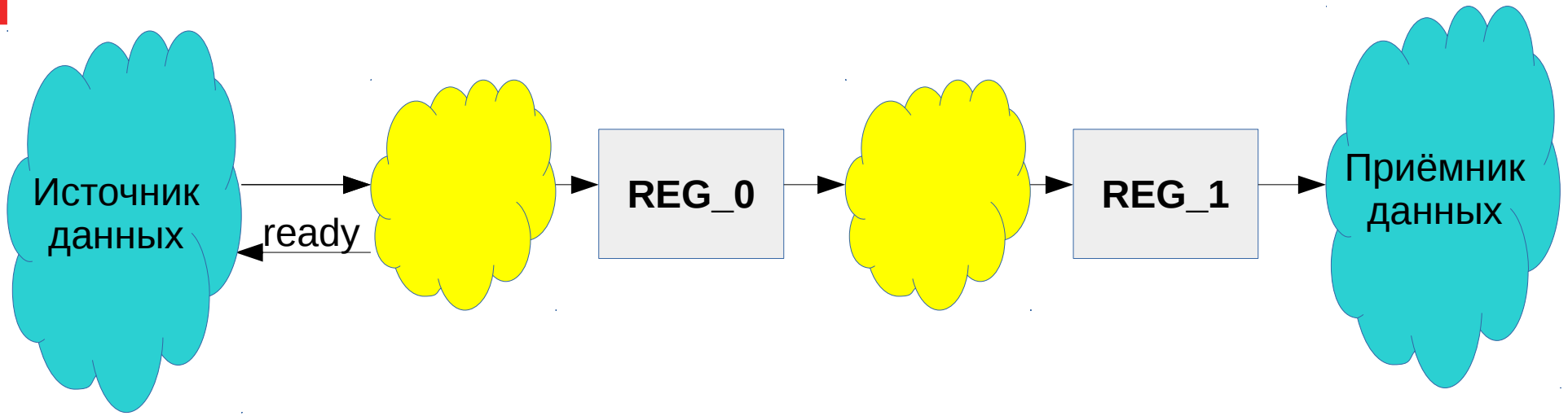
Современные кристаллы FPGA и ASIC очень большие.

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

На слайде ALVEO U200

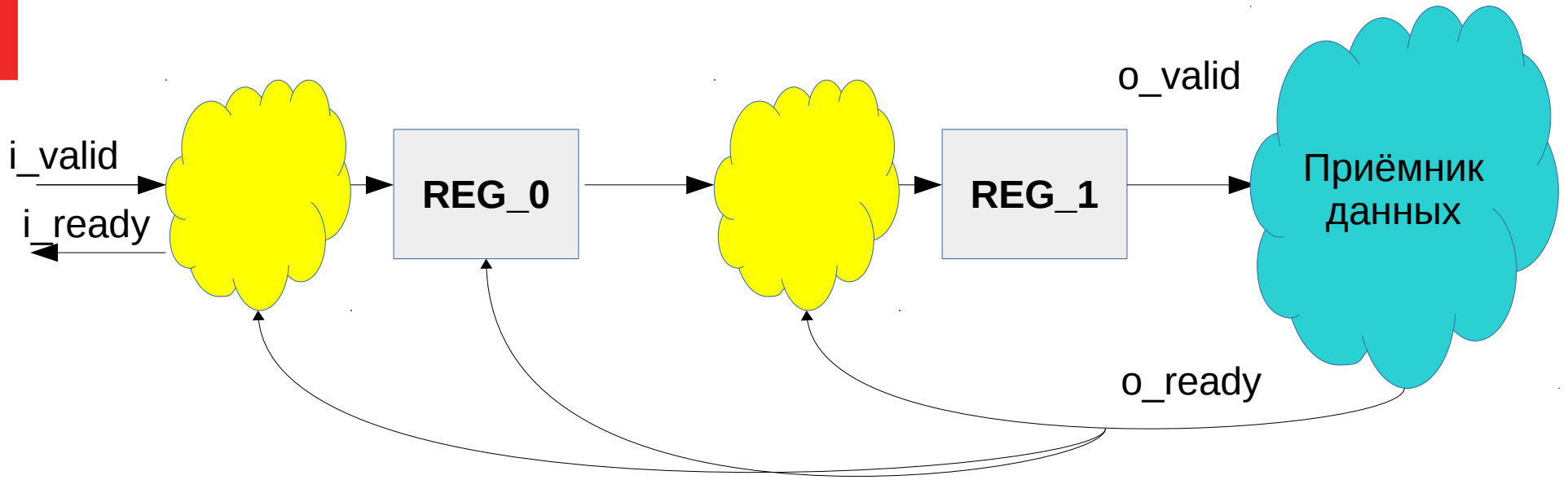


Типичный конвейер



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

Вариант 1 — распространение o_ready

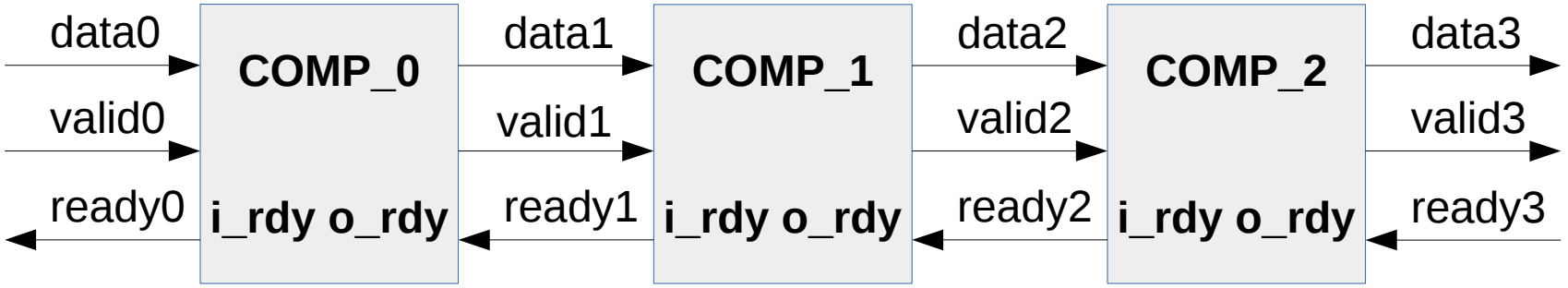


Как правило — приёмник формирует сигнал готовности к приёму данных

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

Недостаток — проблемы с трассировкой

Проблемы каскадного соединения

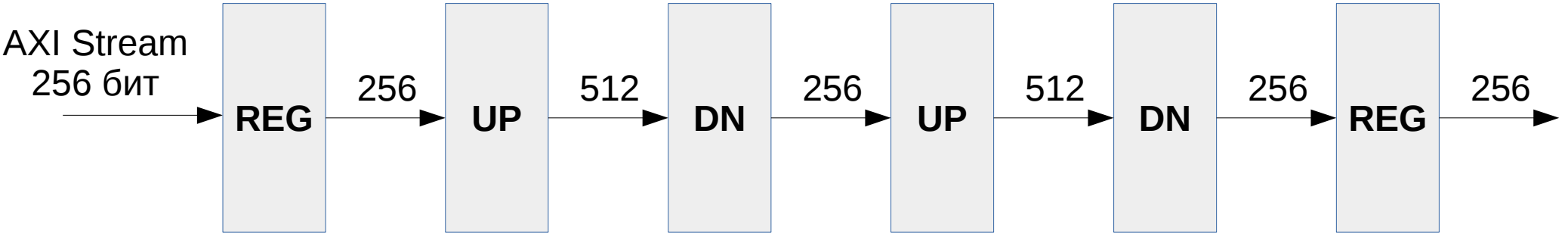


Вполне возможна ситуация когда в каждом блоке сигнал **i_rdy** формируется через комбинационную схему с участием **o_rdy.**

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

- github — пример cascade

Пример — downsizing и upsizing



Трассировка примера в системе Vivado 2020.2

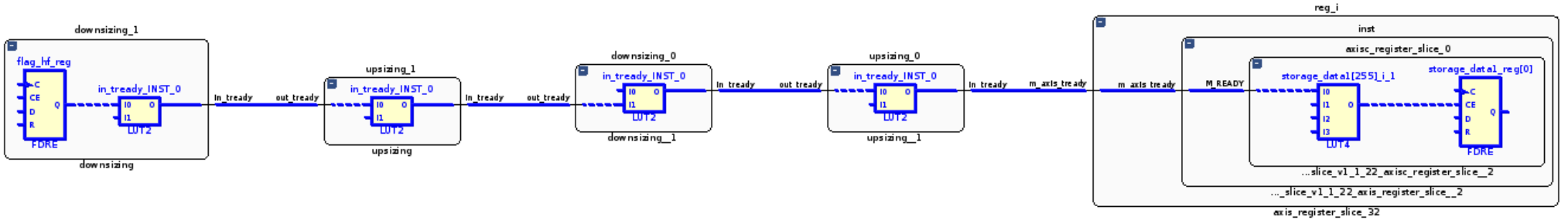
REG — IP Core из каталога Vivado: AXI4 Stream Reg Slice

Синтез для Kintex Ultrascale+ KU3P

Тактовая частота 666 МГц

- github — пример **cascade**

Пример — downsizing и upsizing



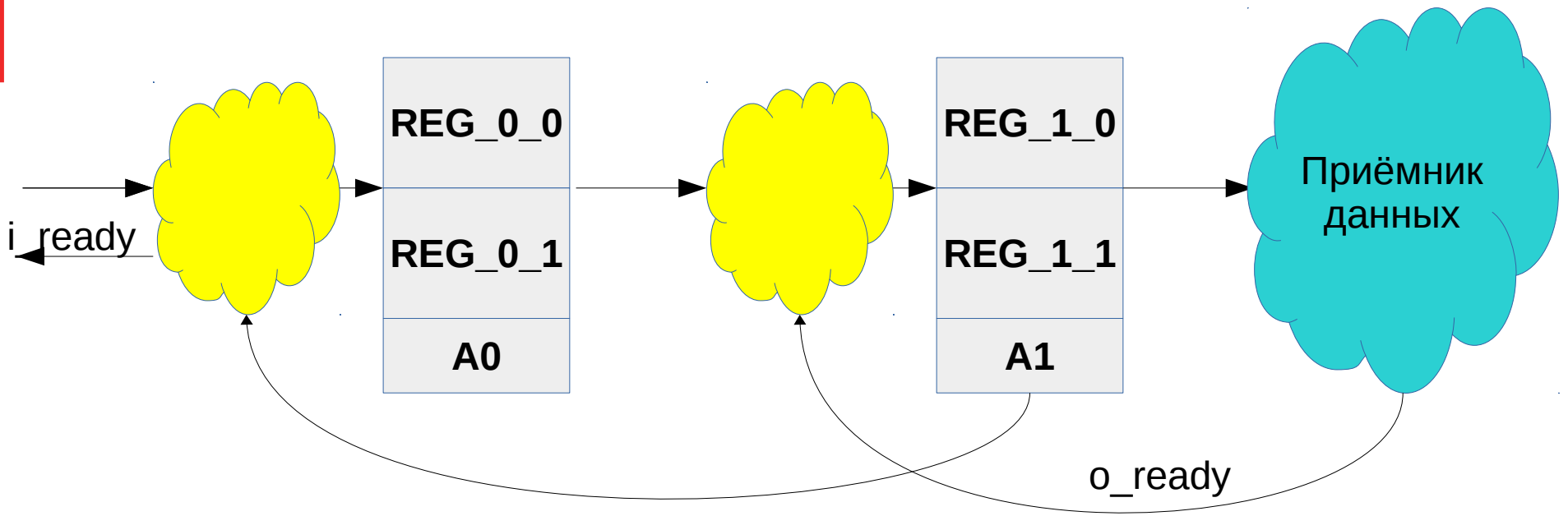
Комбинационная схема для сигнала out_ready проходит через все компоненты

Levels: 5

Fanout: 258

Slack: **-0.086** ns

Вариант 2 — двойной буфер

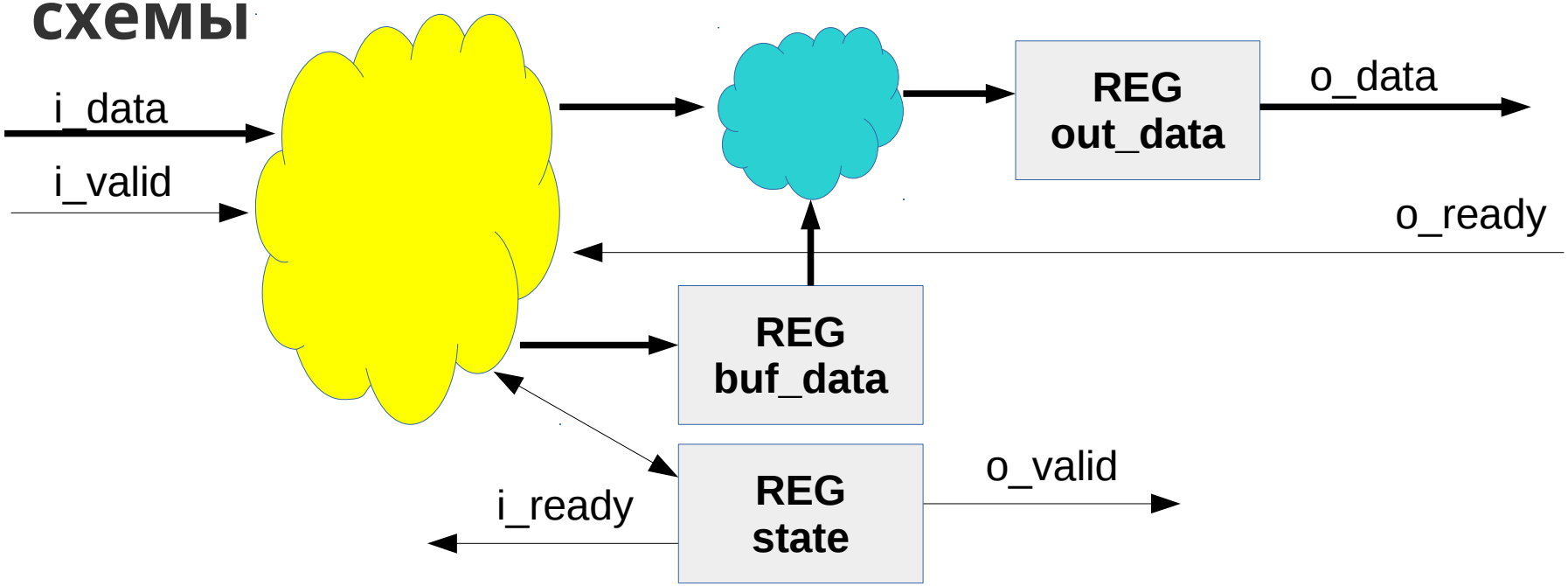


На каждой стадии конвейера используется двойной буфер.

Комбинационная схема анализирует только сигналы от соседних стадий конвейера.

Недостаток — усложнение логики вычислений

Двойной буфер - разрыв комбинационной схемы



Компонент содержит два регистра данных: **out_tdata, buf_tdata**

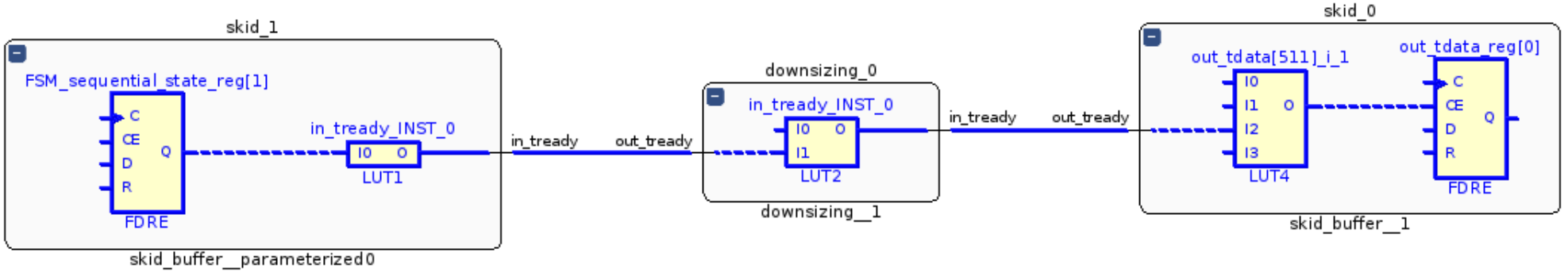
Сигналы **i_ready, o_valid** - выходы с регистра

Сигналы **i_valid, o_ready** влияют на каждый бит регистра данных

Плохая буферизация со стороны входа

- github — пример **skid_buffer**

Пример — downsizing, upsizing и skid_buffer



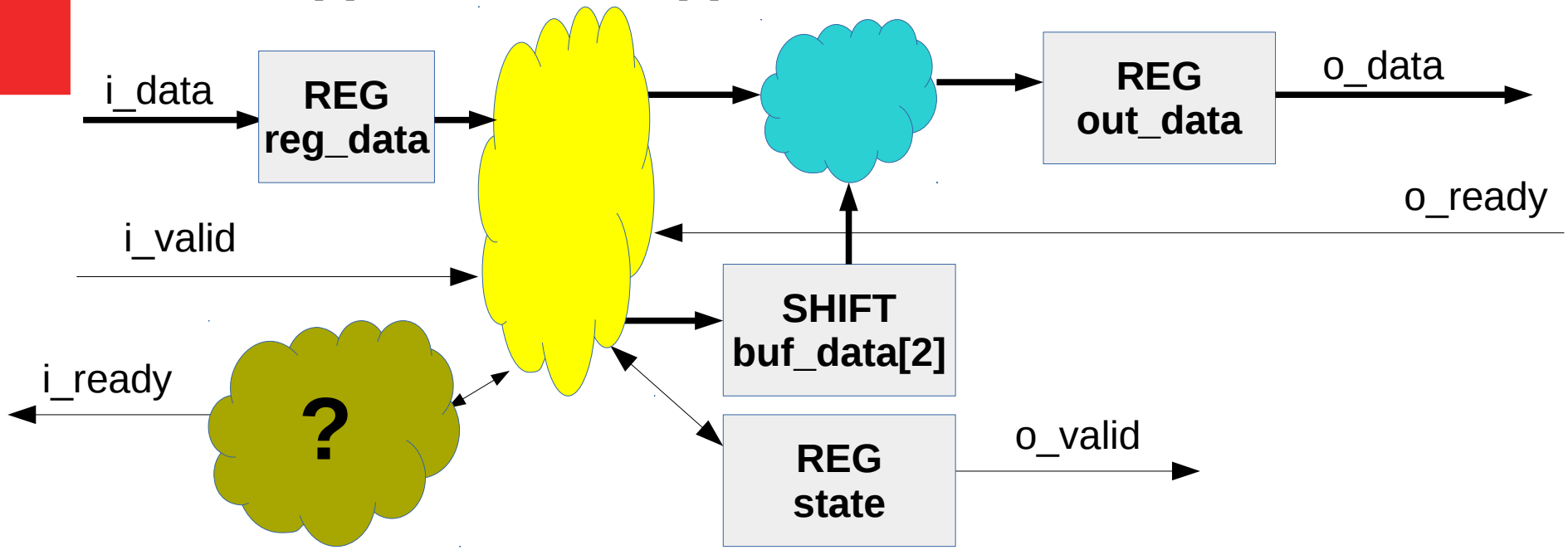
Комбинационная логика ограничена

Levels: 3

Fanout: 512

Slack: 0.322 ns

Как отделить вход ?

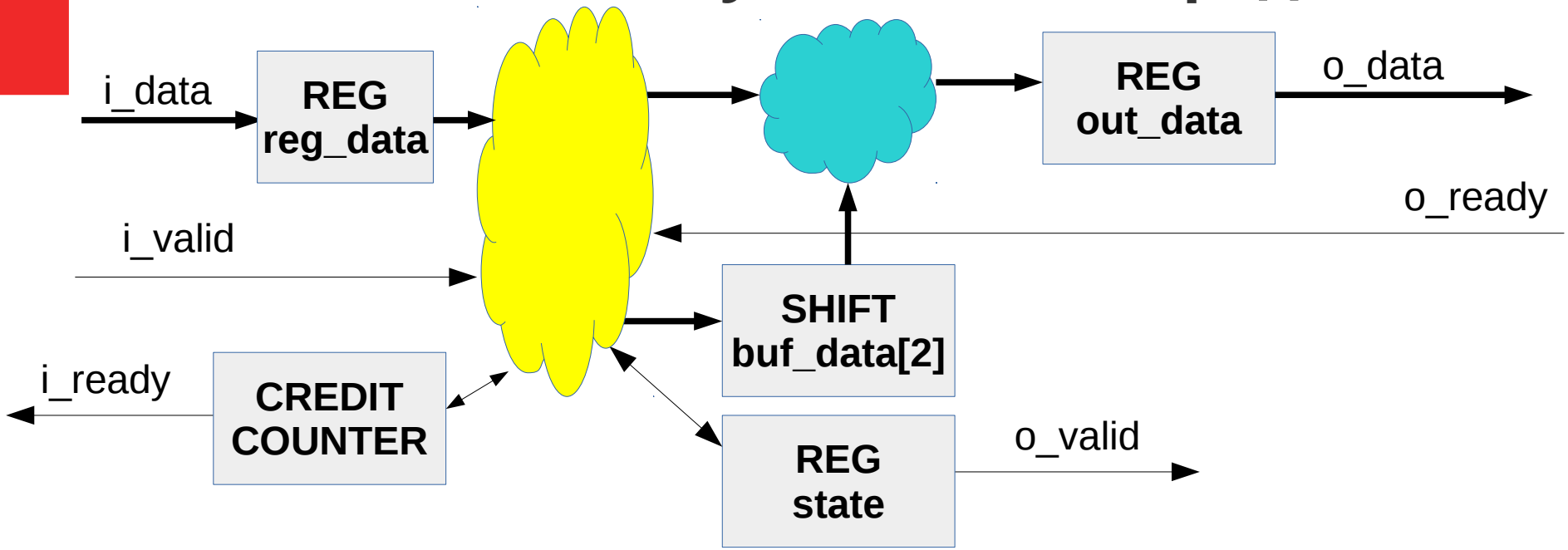


Цель — поставить на входе **i_data** простой регистр без управления.

Проблема — как сформировать **i_ready** ?

Априорные данные — сколько слов может быть запомнено если нет чтения

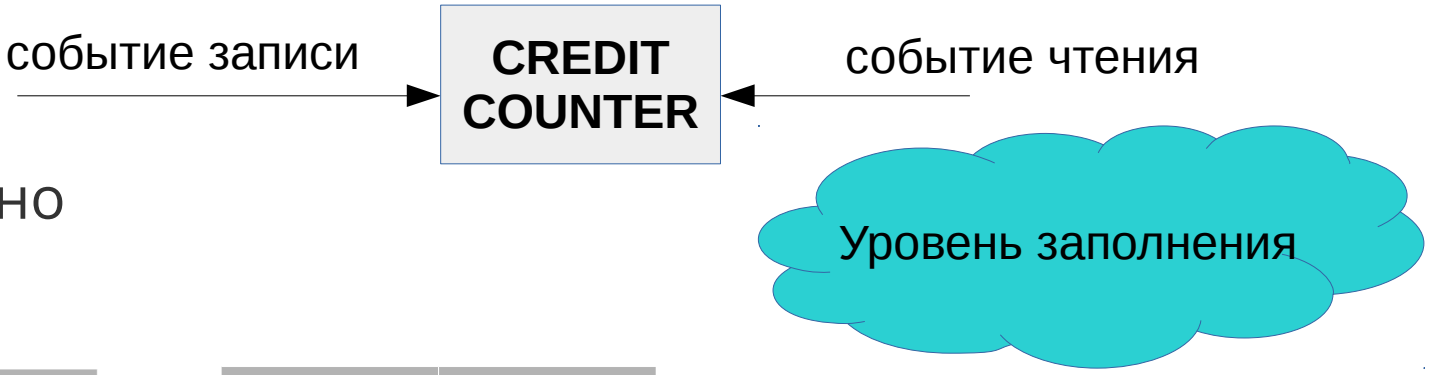
skid_crd — используем счётчик кредитов



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

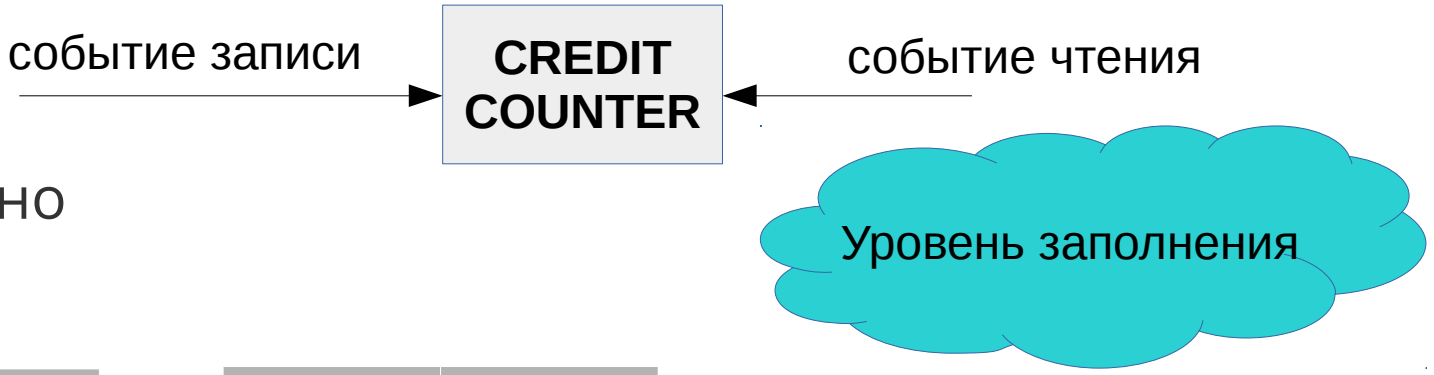


Чтение запрещено

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

Работа кредитного счётчика



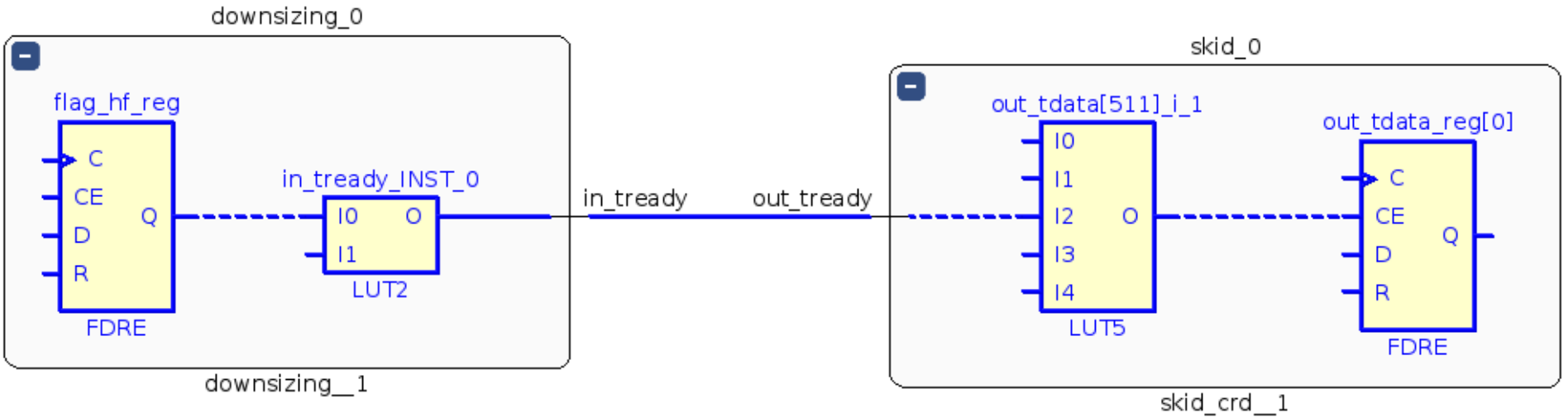
Чтение разрешено

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

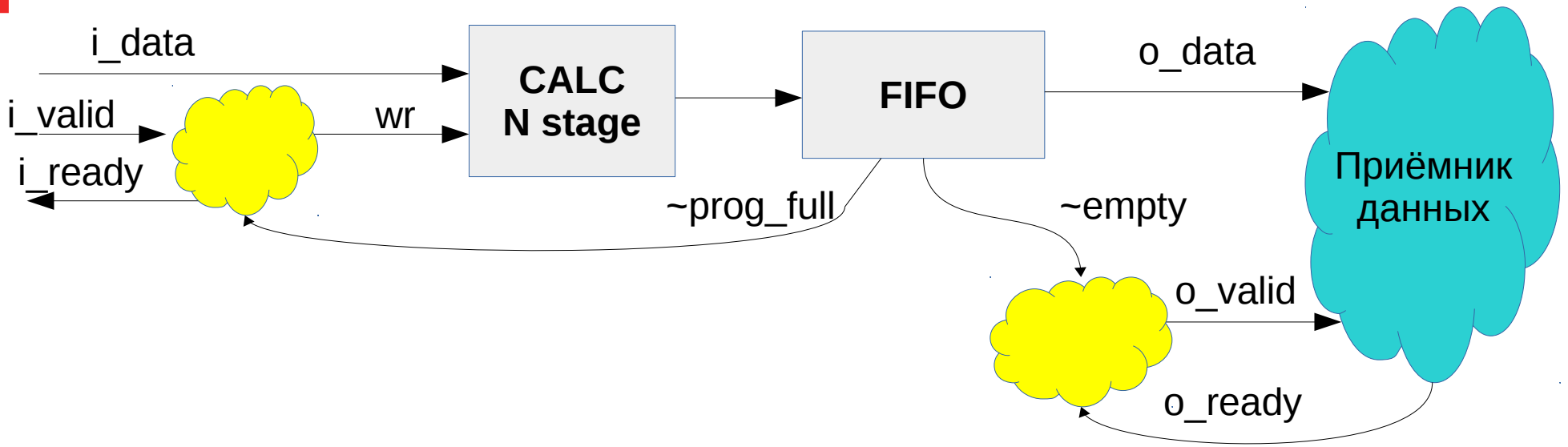
Сравнение примеров

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

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

А можно ли увеличить быстродействие и не увеличивать занимаемые ресурсы ?

Вариант 3 — FIFO



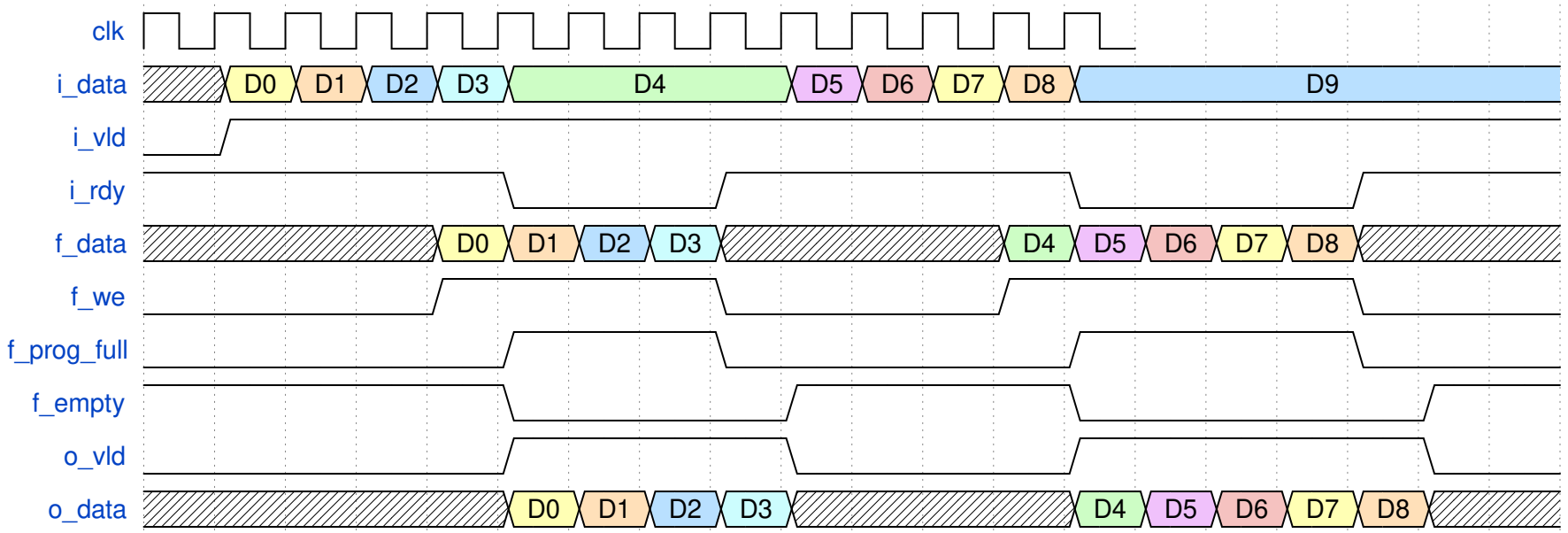
Главная идея — отказ от **valid/ready** внутри конвейера

N — число стадий конвейера

Какой оптимальный размер FIFO ?

prog_full — флаг почти полного FIFO, как минимум равен N

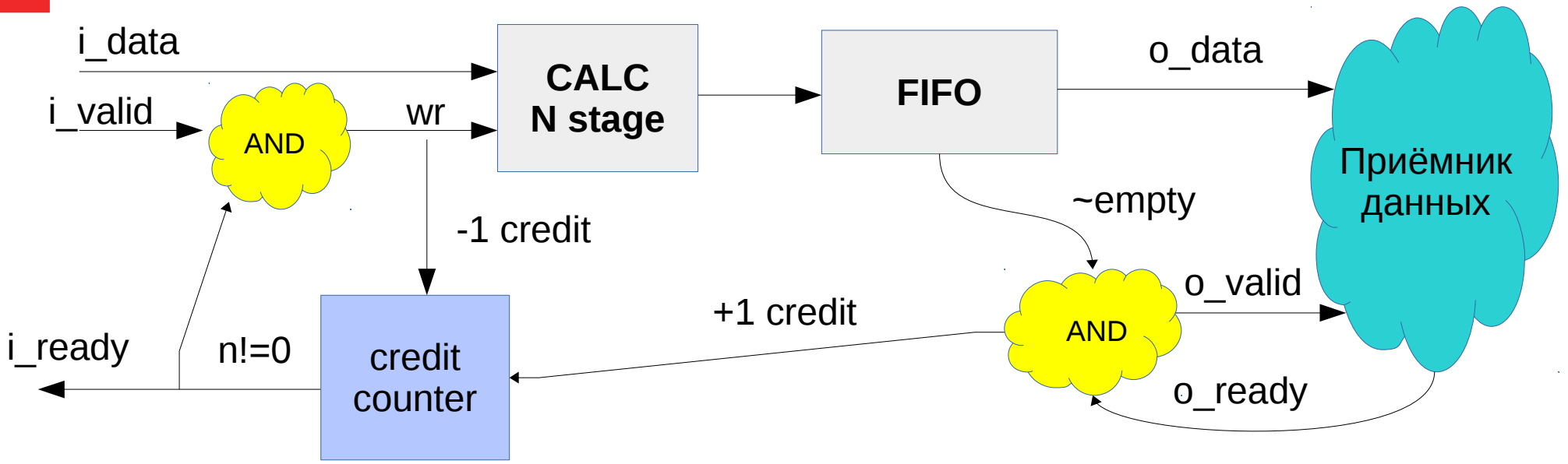
Размер FIFO 4 слова, конвейер 3 такта



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

Оптимальный размер FIFO равен 2N

Вариант 4 — счётчик кредитов

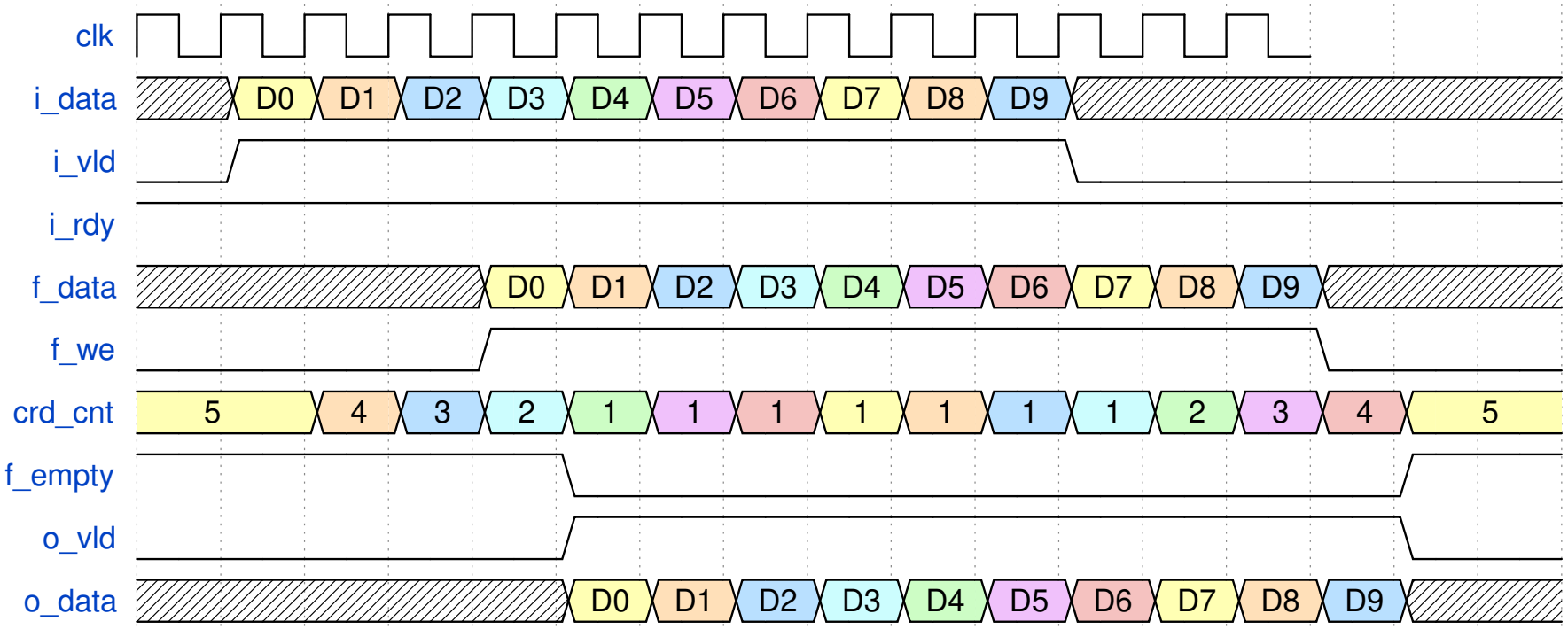


Начальное значение счётчика кредитов равно размеру FIFO

Оптимальный размер FIFO равен N+2

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

Размер FIFO 5 слов, конвейер 3 такта

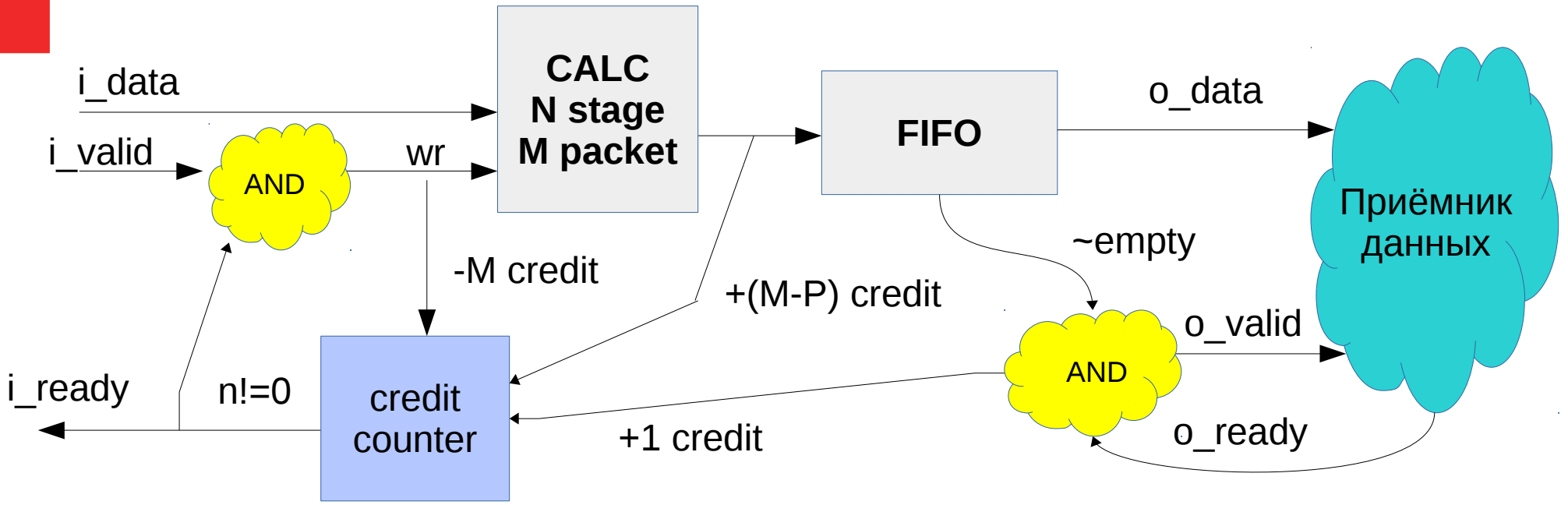


Начальное значение счётчика кредитов равно размеру FIFO

Оптимальный размер FIFO равен N+2

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

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

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

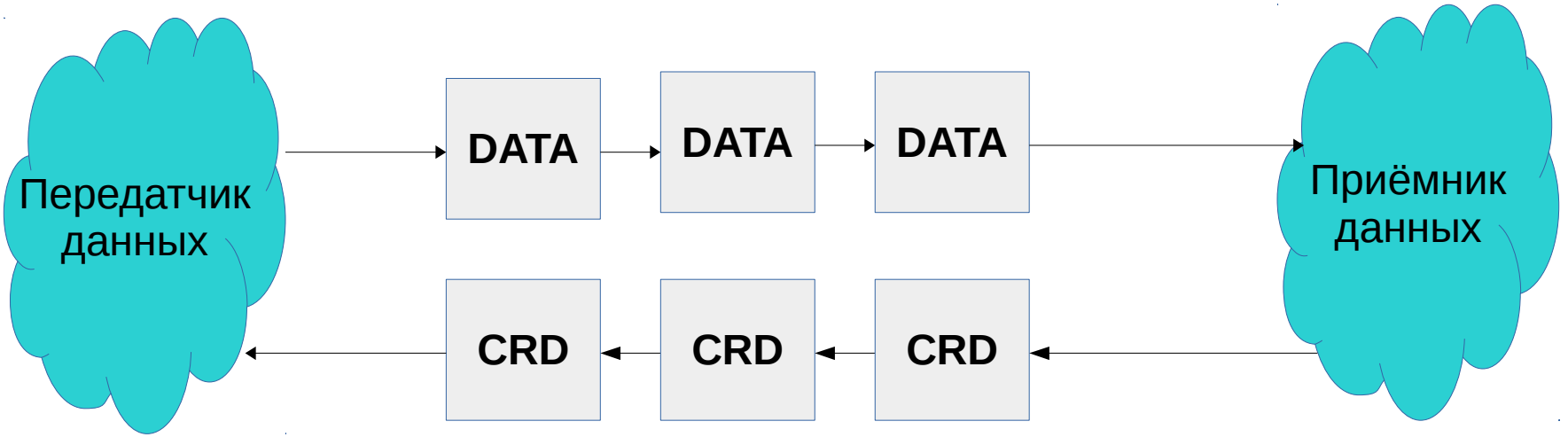
- Кредитный счётчик увеличивается на единицу

Приём данных разрешён при значении кредитного счётчика больше нуля

Сравнение FIFO и кредитного счётчика

FIFO	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 N+2 слова

Вариант шины на основе кредитов



Приёмник передаёт передатчику число данных которое он может принять.

Передатчик передаёт только разрешённое число данных

Аналог — протокол TCP

Примеры на github

skid_buffer - Двойной буфер

skid_crd — буфер со счётчиком кредитов

cascade — каскадное соединение компонентов

cascade_with_skid — каскадное соединение с буферизацией

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