**Потоковый NxM кроссбар с политикой Round-Robin**

Оглавление

[1. Описание 3](#_Toc164028642)

[1.1. Порты и параметры 3](#_Toc164028643)

[1.2. Протокол передачи данных 4](#_Toc164028644)

[1.3. Принцип коммутации 4](#_Toc164028645)

[2. Первые мысли 5](#_Toc164028646)

[3. Реализация 6](#_Toc164028647)

[3.1. Идея общими словами 6](#_Toc164028648)

[3.2. Подробнее о модулях 7](#_Toc164028649)

[3.2.1. stream\_xbar - верхний модуль 7](#_Toc164028650)

[3.2.2. arbiters\_unit – пре-арбитраж и арбитраж 8](#_Toc164028651)

[3.2.3. round\_robin\_arbiter – арбитр с политикой round-robin 9](#_Toc164028652)

[3.2.4. data\_communication\_net – непосредственная передача информации 13](#_Toc164028653)

[4. Пути оптимизации 14](#_Toc164028654)

# Описание

## Порты и параметры

**Параметры:**

* T\_DATA\_WIDTH = 8 – ширина регистра данных
* S\_DATA\_COUNT = 2 – количество мастер-устройств
* M\_DATA\_COUNT = 3 – количество слэйв-устройств
* T\_ID\_\_\_WIDTH = $clog2(S\_DATA\_COUNT) – ширина регистра для перечисления мастер-устройств
* T\_DEST\_WIDTH = $clog2(M\_DATA\_COUNT) – ширина регистра для перечисления слэйв-устройств

**Входные порты:**

* logic clk – сигнал тактирования
* logic rst\_n – инвертированный сигнал сброса
* logic [T\_DATA\_WIDTH-1:0] s\_data\_i [S\_DATA\_COUNT-1:0] – данные
* logic [T\_DEST\_WIDTH-1:0] s\_dest\_i [S\_DATA\_COUNT-1:0] – номер слэйв-устройства назначения
* logic [S\_DATA\_COUNT-1:0] s\_last\_i – сигнал, сигнализирующий об окончании транзакции
* logic [S\_DATA\_COUNT-1:0] s\_valid\_i – сигнал, сигнализирующий об корректности и неизменности входных данных до рукопожатия
* logic [M\_DATA\_COUNT-1:0] m\_ready\_i – сигнал от слэйв-устройства о готовности принять данные

**Выходные порты:**

* logic [T\_DATA\_WIDTH-1:0] m\_data\_o [M\_DATA\_COUNT-1:0] - данные
* logic [T\_ID\_\_\_WIDTH-1:0] m\_id\_o   [M\_DATA\_COUNT-1:0] - номер мастер-устройства источника
* logic [M\_DATA\_COUNT-1:0] m\_last\_o - сигнал окончания транзакции
* logic [M\_DATA\_COUNT-1:0] m\_valid\_o - сигнал валидности
* logic [S\_DATA\_COUNT-1:0] s\_ready\_o - сигнал мастер-устройству, сигнализирующий о готовности передать новые данные

## Протокол передачи данных

* 1. Данные передаются транзакциями.
  2. Одна транзакция может состоять из нескольких пакетов данных.
  3. Пакет считается переданным, если valid и ready установлены в единицу.
  4. Комбинационная зависимость сигнала valid от сигнала ready – запрещена.
  5. Сигнал ready может комбинационно зависеть от сигнала valid.
  6. Если сигнал valid установлен в 1, то сигналы, устанавливаемые master-устройством, не могут менять свое значение, пока не будет передан пакет данных.
  7. Для последнего из передаваемых в транзакции пакетов данных должен быть установлен сигнал last.

## Принцип коммутации

Потоковый кроссбар, обеспечивающий коммутацию S\_DATA\_COUNT master-устройств и M\_DATA\_COUNT slave-устройств. Выбор slave-устройства, на которое должен быть передан запрос, производится по значению сигнала s\_dest\_i. Если два master-устройства обращаются к одному slave-устройству, то выбор между ними производится в соответствии с политикой Round-Robin. Если два master-устройства обращаются к различным slave-устройствам, то транзакции должны проходить параллельно - не блокировать друг друга. Сигнал m\_id\_o принимает значение номера входного потока, с которого получена выходная транзакция. Арбитраж производится потранзакционно: переключение арбитра на следующий входной порт производится только после завершения передачи транзакции с текущего выбранного арбитром порта.

# Первые мысли

Поскольку кроссбар с политикой round-robin, то первым делом в голову пришло FIFO. С помощью указателей возможно:

* детектировать отсутствие мастеров в FIFO (флаг is\_empty)
* сэкономить на аппаратуре при сравнении и других комбинационных операциях
* уменьшить динамическое энергопотребление

Поскольку номер мастера считается по порядку от 0 и до S\_DATA\_COUNT, то для той же экономии аппаратуры, а также дальнейшего удобства работы на вход арбитра будет поступать маска, указывающая на то, хочет мастер общаться или нет.

Изначально предполагалось заносить в очередь номера мастеров, выставляющих запрос: [3, 1, 5, 4, …], но возникла проблема с тем, как за один такт записать в очередь несколько id мастеров, одновременно выставивших запрос. Так получилась реализация, при которой входящие маски обрабатываются и записываются в очередь. Добавляется ещё один указатель, движущийся по горизонтали, вдоль этих масок. Его значение – действующее мастер-устройство.

# Реализация

## Идея общими словами

Итога схема состоит из двух модулей: arbiters\_unit, содержащий в себе логику арбитража для каждого слэйв-устройства, и data\_communication\_net, соединяющий входные и выходные порты, исходя из информации арбитров.

Схема имеет буферизацию, задержку в 1 такт.

* 1-й такт. Поступили запросы на общение со слэйв-устройством. Они обрабатываются, и, в случае чего, меняется ведущий мастер. Выставляется сигнал о готовности выдать значение наружу. Эти сигналы поступают на выходные регистры data\_communication\_net.
* 2-й такт. Выходные регистры захлопывают информацию от ведущего устройства.

Необходимо, чтобы данные не менялись после выставления сигнала m\_valid\_o в логическую единицу. Возможно убрать задержку выхода данных, но тогда есть вероятность из-за задержек в прохождении сигнала, что данные изменятся после выставления сигнала m\_valid\_o. Также возникают сложности с записью в очередь нескольких id мастеров, выставивших запрос, в тот же такт, как выставляются сигналы s\_valid\_i.

## Подробнее о модулях

### stream\_xbar - верхний модуль

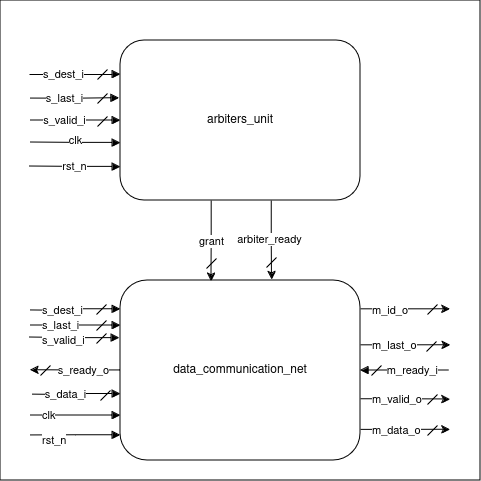


Рисунок 1 - Общая схема кроссбара

* logic [T\_ID\_\_\_WIDTH-1 : 0] **grant** [M\_DATA\_COUNT-1 : 0] – массив регистров, показывающий, какой мастер сейчас в работе. Регистров M\_DATA\_COUNT, как кол-во слэйв-устройств, а их разрядность позволяет пересчитать все возможные мастер-устройства.
* logic [M\_DATA\_COUNT-1 : 0] **arbiter\_ready** – в одном из тактов блок арбитров готовит наружу информацию о лидирующих мастерах. И к следующему фронту тактового сигнала выставляется данный флаг, подтверждающий валидность об id выставленных мастер-устройствах.

### arbiters\_unit – пре-арбитраж и арбитраж

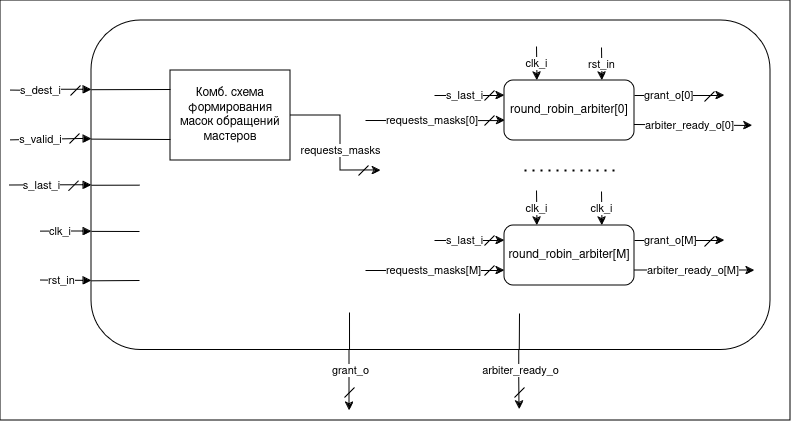


Рисунок 2 - Блок арбитража

Блок содержит в себе M\_DATA\_COUNT арбитров, отвечающих за выбор мастер-устройства для соответствующего слэйв-устройств. Арбитры принимают на вход маску запросов на общение от мастер-устройств, а также сигналы s\_last\_i[S\_DATA\_COUNT-1:0] для отслеживания окончания транзакции. На выходе grant\_o регистр об выбранном мастере и сигнал о готовности данных arbiter\_ready\_o.   
Также важно заметить, что арбитры заменяемы и для определенного слэйв-устройства есть возможность поставить арбитр с другой политикой.

* logic [S\_DATA\_COUNT-1 : 0] **requests\_masks**[M\_DATA\_COUNT-1 : 0] – массив регистров, масок мастеров, обозначающих информацию об их готовности общаться с данным слэйв-устройством. Каждая из масок массива, которых столько, сколько слэйв-устройств, попадает на арбитр соответствующего слэйв-устройства. Для их формирования необходим массив регистров  
  [T\_DEST\_WIDTH-1:0] **s\_dest\_i** [S\_DATA\_COUNT-1:0] с информацией от мастер-устройств о выбранном слэйв-устройстве и сигналы s\_valid\_i [S\_DATA\_COUNT-1:0], подтверждающие валидность передаваемых данных.

### round\_robin\_arbiter – арбитр с политикой round-robin

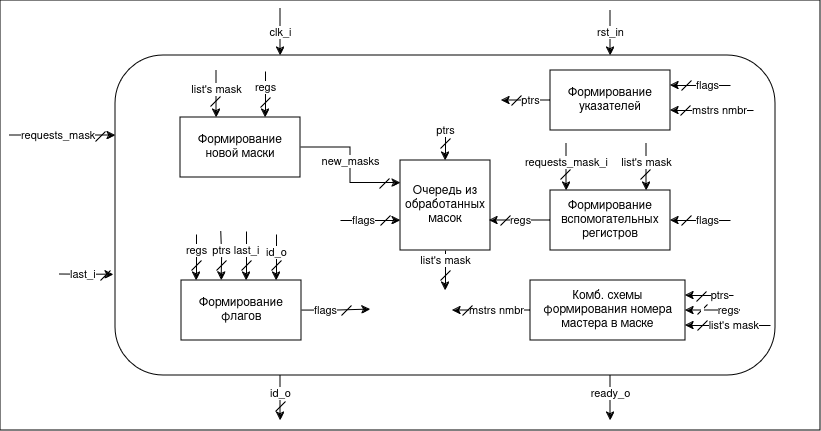


Рисунок 3 - Структура round-robin арбитра

**Round-Robin политика:**

* Мастер-устройство изменяется, если завершается транзакция: приходит last\_i сигнал от мастера, работающего в данный момент.
* Мастер-устройства “встают в очередь”.
* Если несколько мастер-устройств обращаются к данном слэйв-устройству в один момент времени, то помещаться в очередь они будут в приоритетном порядке. Приоритет выстраивается в зависимости от расположения в requests\_mask: от младшего бита к старшему.

***Алгоритм работы (на примере 5-ти мастеров):***

Создаётся массив, очередь из регистров, обработанных входящих масок.  
Запись в эту очередь и считывание из неё информации об следующем мастере производится тремя указателями.

**Сама очередь:**

* logic [S\_DATA\_COUNT-1 : 0] **list\_of\_mask** [S\_DATA\_COUNT-1:0] - размер маски равен S\_DATA\_COUNT - количество мастеров. Количество ячеек в списке также будет равно S\_DATA\_COUNT (в нашем случае S\_DATA\_COUNT = 5), потому что даже если все мастера делают запрос в разное время и, таким образом, выдерживая сигнал last\_i, равным логической единице, то будет занято ровно столько ячеек, сколько всего мастеров.

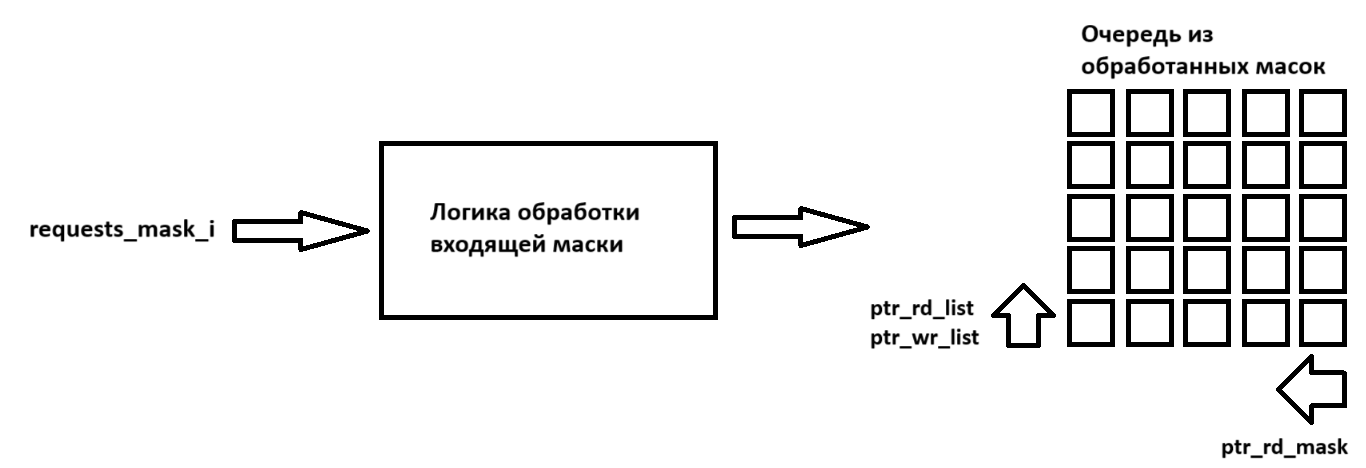


Рисунок 4 - От маски до загрузки в очередь

**Таким образом:**

1. Очередь из мастер-устройств выстраивается в  
    list\_of\_masks [0] [0]  
    list\_of\_masks [0] [1]  
    …  
    list\_of\_masks [S\_DATA\_COUNT-1] [S\_DATA\_COUNT-2]  
    list\_of\_masks [S\_DATA\_COUNT-1] [S\_DATA\_COUNT-1]
2. Новая маска обрабатывается, чтобы убрать запросы от мастер-устройств, висевших до неё, и оставляет только новоприбывшие запросы.
3. Когда ptr\_rd\_mask указывается на бит в списке, то есть на текущий работающий мастер, то он уже равняется нулю. Это необходимо, чтобы смочь детектировать для следующего такта информацию о том, что текущая маска в списке уже пуста.

**Информация об остальных сигналах:**  
Входные выходные порты:

* + input  logic [S\_DATA\_COUNT-1:0] **requests\_mask\_i** – входящая маска запросов мастер-устройств
  + input  logic [S\_DATA\_COUNT-1:0] **last\_i** – s\_last\_i сигналы от мастер-устройств
  + output logic [T\_ID\_\_\_WIDTH-1:0] **id\_o** – номер выбранного мастер-устройства. Является ptr\_rd\_mask.
  + output logic **ready\_o** – сигнал, подтверждающий, что если он в логической единице и приходит фронт тактового импульса, выходные id\_o корректен.

Указатели:

* logic [T\_ID\_\_\_WIDTH-1 : 0] **ptr\_wr\_list** – указывает на строку очереди, которая будет перезаписана.
* logic [T\_ID\_\_\_WIDTH-1 : 0] **ptr\_rd\_list** – на строку очереди, обработанную маску, которая на данный момент считывается.
* logic [T\_ID\_\_\_WIDTH-1 : 0] **ptr\_rd\_mask** – указывает на столбец (происходит смещение внутри маски, на которую указывает ptr\_rd\_list) очереди, который на данный момент используется. В действительности стоит говорить об совместном использовании ptr\_rd\_mask и ptr\_rd\_list, поскольку их комбинация позволяет определить, кто на данный момент их мастер-устройств будет работать с слэйв-устройством.

Флаги:

* logic **is\_empty**       - list\_of\_masks пустой. Никого нет в очереди.
* logic **is\_new\_mask** - ещё один мастер выставил запрос об общении.
* logic **is\_empty\_mask** - текущая маска (used\_mask) пуста. Необходимо передвинуть ptr\_rd\_list.
* logic **is\_last**       - пришёл сигнал last\_i от работающего в данный момент мастера.
* logic **is\_was\_last** - сохранения сигнала is\_last на следующий такт. Непосредственно используется в логике арбитра.

Основные регистры:

* logic [S\_DATA\_COUNT-1 : 0] **crnt\_mask** – текущая маска.
* logic [S\_DATA\_COUNT-1 : 0] **last\_mask** – пришедшая маска с предыдущего такта.
* logic [S\_DATA\_COUNT-1 : 0] **used\_mask** – является текущей использующейся маской списка.

Для формирования новой маски:

* logic [S\_DATA\_COUNT-1 : 0] **masters\_in\_line** - слепок из всех мастеров, находящихся в данный момент в очереди.
* logic [S\_DATA\_COUNT-1 : 0] **updated\_mask** - если пришёл сигнал last\_i, но маска, на которую указывает ptr\_rd\_list, не нулевая, то в ptr\_rd\_list будет записана данная маска.
* logic [S\_DATA\_COUNT-1 : 0] **next\_mask**  - если пришёл сигнал last\_i, и маска, на которую указывает ptr\_rd\_list, нулевая, то в ptr\_rd\_list + 1 будет записана данная маска.
* logic [S\_DATA\_COUNT-1 : 0] **new\_mask** - если is\_new\_mask, то новую маску необходимо записать в список по указателю ptr\_wr\_list.

Для формирования номера старшего мастера:

* logic [T\_ID\_\_\_WIDTH-1 : 0] **next\_used\_mask\_low\_bit** - из следующей в очереди обработанной маски. Используется, если от мастера пришёл сигнал last\_i, и маска, на которую указывает ptr\_rd\_list, нулевая.
* logic [T\_ID\_\_\_WIDTH-1 : 0] **used\_mask\_low\_bit** - из использующейся на данной момент маски списка. Используется, если от мастера пришёл сигнал last\_i, но маска, на которую указывает ptr\_rd\_list, ещё содержит в себе информацию о следующем мастере.
* logic [T\_ID\_\_\_WIDTH-1 : 0] **crnt\_mask\_low\_bit** - из только пришедшей requests\_mask\_i. Используется, если очередь пуста.

Для формирования битовой маски из номера старшего мастера:

* + logic [S\_DATA\_COUNT-1 : 0] **lead\_master\_of\_used\_mask** - текущей маски по указателю ptr\_rd\_list
  + logic [S\_DATA\_COUNT-1 : 0] **lead\_master\_of\_next\_mask** - маски по указателю   
    ptr\_rd\_list + 1
  + logic [S\_DATA\_COUNT-1 : 0] **lead\_master\_of\_new\_mask** - маски по указателю ptr\_wr\_list

Таблица 1 - Пример заполнения списка обработанными масками

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Такт | Входная маска | Is\_last | list\_of\_mask | ptr\_rd\_mask | ptr\_rd\_list | ptr\_wr\_list |
| 1 | 01101 | 0 | 00000  00000  00000  00000  00000 | 0 | 0 | 0 |
| 2 | 01111 | 1 | 00000  00000  00000  00000  01100 | 0 | 0 | 1 |
| 3 | 11111 | 0 | 00000  00000  00000  00010  01000 | 2 | 0 | 2 |
| 4 | 11111 | 1 | 00000  00000  10001  00010  01000 | 2 | 0 | 3 |
| 5 | 11011 | 0 | 00000  00000  10001  00010  00000 | 3 | 0 | 3 |

### data\_communication\_net – непосредственная передача информации

Содержит регистры на выходе, разрешающие “захлопывать” информацию по сигналу arbiter\_ready\_i для соответствующего слэйв-устройства.

# Пути оптимизации