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

Оглавление

[1. Описание 3](#_Toc165084368)

[1.1. Порты и параметры 3](#_Toc165084369)

[1.2. Протокол передачи данных 5](#_Toc165084370)

[1.3. Принцип коммутации 5](#_Toc165084371)

[2. Первые мысли 6](#_Toc165084372)

[3. Реализация 8](#_Toc165084373)

[3.1. Идея общими словами 8](#_Toc165084374)

[3.2. Подробнее о модулях 9](#_Toc165084375)

[3.2.1. stream\_xbar - верхний модуль 9](#_Toc165084376)

[3.2.2. arbiters\_unit – пре-арбитраж и арбитраж 10](#_Toc165084377)

[3.2.3. round\_robin\_arbiter – арбитр с политикой round-robin 11](#_Toc165084378)

[3.2.4. data\_communication\_net – непосредственная передача информации 18](#_Toc165084379)

[4. Пути оптимизации 20](#_Toc165084380)

# Описание

## Порты и параметры

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Разрядность** | **Описание** |
| 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:**

|  |  |  |
| --- | --- | --- |
| **Сигнал** | **Разрядность** | **Описание** |
| clk | 1 | сигнал тактирования |
| rst\_n | 1 | инвертированный сигнал сброса |
| s\_data\_i | [S\_DATA\_COUNT-1:0]  X  [T\_DATA\_WIDTH-1:0] | данные |
| s\_dest\_i | [S\_DATA\_COUNT-1:0]  X  [T\_DEST\_WIDTH-1:0] | номер слэйв-устройства назначения |
| s\_last\_i | [S\_DATA\_COUNT-1:0] | сигнал, сигнализирующий об окончании транзакции |
| s\_valid\_i | [S\_DATA\_COUNT-1:0] | сигнал от слэйв-устройства о готовности принять данные |
| m\_ready\_i | [M\_DATA\_COUNT-1:0]` | Сигнал от слэйв-устройства о готовности принять данные |

**Выходные порты:**

|  |  |  |
| --- | --- | --- |
| **Сигнал** | **Разрядность** | **Описание** |
| clk | 1 | сигнал тактирования |
| rst\_n | 1 | инвертированный сигнал сброса |
| m\_data\_o | [M\_DATA\_COUNT-1:0]  X  [T\_DATA\_WIDTH-1:0] | данные |
| m\_id\_o | [M\_DATA\_COUNT-1:0]  X  [T\_DEST\_WIDTH-1:0] | номер слэйв-устройства назначения |
| m\_last\_o | [M\_DATA\_COUNT-1:0] | сигнал, сигнализирующий об окончании транзакции |
| m\_valid\_o | [M\_DATA\_COUNT-1:0] | сигнал от слэйв-устройства о готовности принять данные |
| s\_ready\_o | [S\_DATA\_COUNT-1:0] | 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 что-то, связанное с методикой FIFO. Если использоваться указатели, то выйдет:

* сэкономить на аппаратуре при сравнении и других комбинационных операциях.
* уменьшить динамическое энергопотребление на отсутствии лишних переключениях регистров.
* Убрать задержки продвижения данных по сдвиговому регистру, который мог стать аналогом.

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 – ptr\_rd\_mask. Его значение – действующее мастер-устройство.

Изначально схема предполагалась буферизированной, с задержкой 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.

Позже было решено убрать задержку, сделав акцент на скорости выставления данных наружу.

Изначально, вместе с той же буферизацией, использовался на выходе арбитра сигнал ready\_o – такой сигнал, что если он в логической единице и приходит фронт тактового импульса, выходные id\_o корректен. Позже был убран за ненадобностью

Изначально, когда ptr\_rd\_mask указывал на бит в списке, то есть на текущий работающий мастер, то он, бит, уже равнялся нулю. Это было необходимо, чтобы смочь детектировать для следующего такта информацию о том, что текущая маска в списке уже пуста.   
Но позже встал вопрос с тем, что при выставлении в первый раз valid имеется задержка на выдачу данных наружу, равная 2-ум. Тогда было решено сделать ptr\_rd\_mask комбинационным. Но если бит, на который он должен указывать, будет ноль сразу же в начале работы, то указателю будет не за что ухватиться. Потому идея о том, чтобы обнулять в обработанной маске списка работающий мастер, была отвергнута.

# 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.   
Очередь мастеров отслеживается путём квадратной матрицы (в тексте также сплывает слово очередь или список, list - всё об одном), размер которой равняется кол-ву мастеров. Также используются три указателей для работы с этой матрицей. Два указателя на чтение: один выбирает ячейку, другой мастера внутри этой ячейки, – третий выбирает куда записывать новые запросы на общение. Эти самые запросы на входе в арбитр конвертируются в маску запросов. Если запросы мастеров разделены во времени, то информация о них будет записываться в новую ячейку этого списка. Если же одновременно, то в одну ячейку запишутся они все, но перебираться внутри этой ячейки будут от младшего бита к старшему.

.

## Подробнее о модулях

### stream\_xbar - верхний модуль

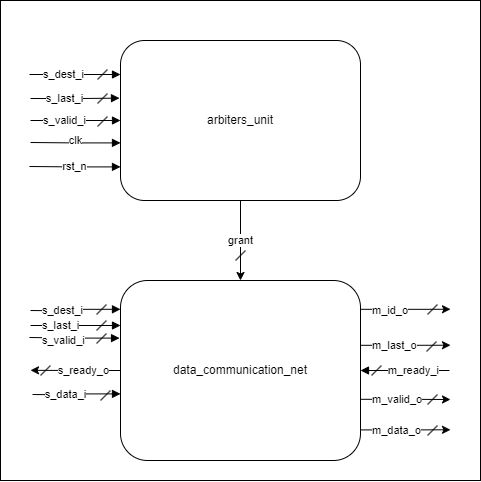


Рисунок 1 - Общая схема кроссбара

* 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.

### 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 сигнал grant\_o об выбранных мастерах.  
*Также стоит 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, то будет занято ровно столько ячеек, сколько всего мастеров, так как мастер, вставший в очередь, не может встать в неё дважды.

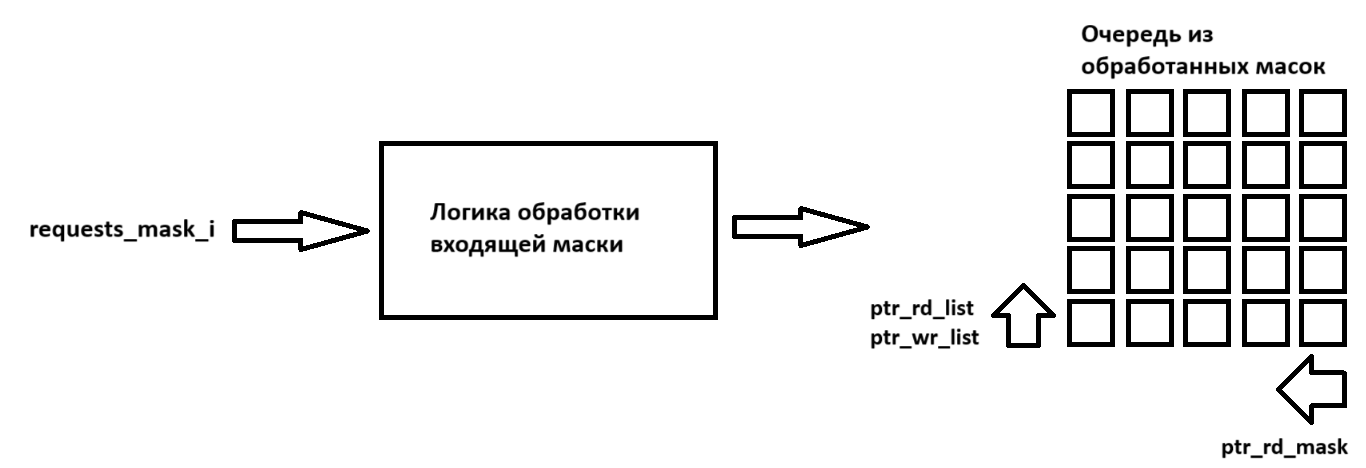


Рисунок 4 - От маски до загрузки в очередь

**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.

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

|  |  |  |
| --- | --- | --- |
| **Сигнал** | **Разрядность** | **Описание** |
| clk\_i | 1 | Сигнал тактирования |
| rst\_in | 1 | Сигнал сброса |
| requests\_mask\_i | [S\_DATA\_COUNT-1:0] | входящая маска запросов мастер-устройств |
| last\_i | [S\_DATA\_COUNT-1:0] | s\_last\_i сигналы от мастер-устройств |

Выходные порты

|  |  |  |
| --- | --- | --- |
| **Сигнал** | **Разрядность** | **Описание** |
| id\_o | [T\_ID\_\_\_WIDTH-1:0] | номер выбранного мастер-устройства. Является ptr\_rd\_mask |

Указатели:

|  |  |  |
| --- | --- | --- |
| **Сигнал** | **Разрядность** | **Описание** |
| ptr\_wr\_list | [T\_ID\_\_\_WIDTH-1 : 0] | указывает на строку очереди, которая будет перезаписана. |
| ptr\_rd\_list | [T\_ID\_\_\_WIDTH-1 : 0] | на строку очереди, обработанную маску, которая на данный момент считывается. |
| ptr\_rd\_mask | [T\_ID\_\_\_WIDTH-1 : 0] | 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:

|  |  |
| --- | --- |
| **Сигнал** | **Описание** |
| is\_empty | list\_of\_masks пустой. Никого нет в очереди. |
| Is\_valid | Кто-то из мастеров установил valid сигнал. |
| Is\_start | Очередь была пустой, но запись в список масок вот-вот начнется. Необходимо, чтобы при первом же появлении valid смочь установить выходного мастера, а не ждать записи в list. |
| Is\_new\_mask | Пришёл сигнал valid от мастера, который ещё не встал в очередь. |
| Is\_last | Работающий мастер послал сигнал last. |
| Is\_almost\_empty\_mask | Последний мастер обработанной маски списка отправил last сигнал. Дальше смена ptd\_rd\_list указателя на следующую обработанную маску списка. |
| Is\_almost\_empty | Последняя обработанная маска списка. Необходимо для отслеживания окончания очереди. |
| Is\_end\_of\_master | Мастер отправил не только last, но также выставил valid сигнал. |
| Is\_jump\_to\_next\_mask |  |
| Is\_almost\_end | Последний мастер выставил всё необходимое для окончания транзакции, и нет новых масок |
| Is\_was\_almost\_end | Отслеживания процесса окончания очереди. Используется, если остался один мастер, но после рукопожатия он продолжает держать valid сигнал. По идее его запрос должен обработаться, как новая маска, а значит будет запись в новые ячейки списка. Но зачем тратить энергию на перезаписывание, если можно оставаться в такой же позиции, как и при окончании предыдущей транзакции? Этот флаг помогает в работе с этим. |
| Is\_dfntl\_end | Все транзакции отправлены. Никто не оставил свой valid. Никто не встал в очередь. |
| Is\_time\_to\_write | Когда записать новую ячейку. В основном это при появлении новой маски. Но есть исключения, как в случае с последним мастером. |
| Is\_time\_for\_shifting | Когда сдвигать указатель на другую ячейку. |
| Is\_time\_to\_update\_mask | В одной ячейке может быть несколько запросов (мастеры выставили запросы в один момент). При отработке мастера нужно обнулить его позицию в маске ячейки списка. |

Основные регистры:

|  |  |  |
| --- | --- | --- |
| Сигналы | Разрядность | Описание |
| crnt\_mask | [S\_DATA\_COUNT-1 : 0] | Текущая маска запросов от мастеров. |
| used\_mask | [S\_DATA\_COUNT-1 : 0] | Текущая использующаяся маска списка. На неё указывает ptr\_rd\_list. |

Для формирования новой маски:

|  |  |  |
| --- | --- | --- |
| Сигналы | Разрядность | Описание |
| masters\_in\_line | [S\_DATA\_COUNT-1 : 0] | слепок из всех мастеров, находящихся в данный момент в очереди. |
| updated\_mask | [S\_DATA\_COUNT-1 : 0] | При отработке мастера необходимо обнулить его позицию в маске списка. |
| new\_mask | [S\_DATA\_COUNT-1 : 0] | Обработанная маска запросов от мастеров, не находящихся в очереди. |

Для формирования номера старшего мастера:

|  |  |  |
| --- | --- | --- |
| Сигналы | Разрядность | Описание |
| used\_mask\_low\_bit | [T\_ID\_\_\_WIDTH-1 : 0] | Поддерживает комбинационный выход арбитра на Id того мастера, который сейчас выбран для работы. В основном используется этот сигнал |
| crnt\_mask\_low\_bit | [T\_ID\_\_\_WIDTH-1 : 0] | Указывает на мастера, который должен работать, если бы мы выбирали его из текущей маски запросов. Используется с краевых случаях, когда список пустой или при старте. |

Для формирования битовой маски из номера старшего мастера:

|  |  |  |
| --- | --- | --- |
| Сигналы | Разрядность | Описание |
|  |  | Текущий работающий мастер по указателю ptr\_rd\_list. Используется при расчетах значений масок для записи в список. |

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  01101 | 0 | 0 | 1 |
| 3 | 11111 | 0 | 00000  00000  00000  00010  01100 | 2 | 0 | 2 |
| 4 | 11111 | 1 | 00000  00000  10001  00010  01100 | 2 | 0 | 3 |
| 5 | 11011 | 0 | 00000  00000  10001  00010  01000 | 3 | 0 | 3 |

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

Комбинационная логика, конфигурирующая на выход входные порты в зависимости grant\_i сигнала, он же id\_o арбитров.   
Стоит отметить, что запись m\_valid\_o происходит следующий образом:

**m\_valid\_o[s\_idx]   
=   
s\_dest\_i [grant\_i[s\_idx]] == s\_idx  
&&   
s\_valid\_i [grant\_i[s\_idx]];**

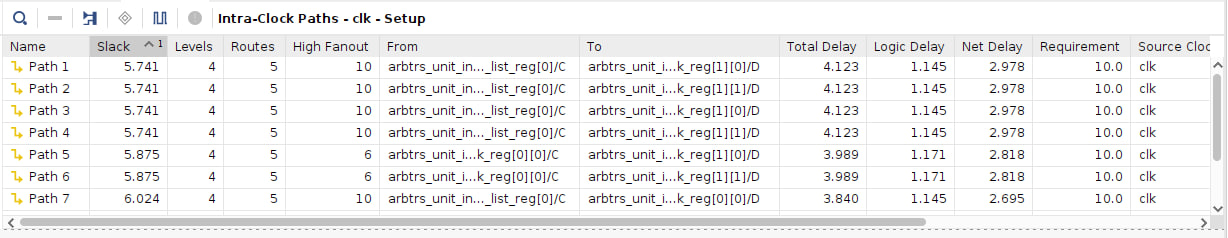
Помимо входного valid сигнала ещё проверяется по значению s\_dest\_i мастера, а действительно ли он в данный слэйв пишет. Это необходимо для разрешения проблем, возникающих, когда из-за устройства арбитра для non-valid мастера будет выставляться на выход id\_o = 0. На слэйва в действительности никто не подаёт, но из-за нулевого по умолчанию выхода слэйва будет проверяться valid сигнал нулевого мастера. А он, в свою очередь, может куда-то писать, а следовательно, иметь выставленным в единицу valid сигнал. Для этого дополнительно и проверяется назначение мастера.

# Анализ критических путей в дизайне

Наибольший критический путь лежит внутри арбитра.

Также основа поиска, очевидно, лежит в логике, определяющей какую-либо работу: смещение, выборка - со списком масок. Тогда можно обратить внимание на флаги. Наибольшие затраты по времени должны пройти через сигнал is\_almost\_empty, поскольку в его логике используется компаратор, чего нет в других флагах. Но и это ещё не всё: данный флаг является частью других: is\_jump\_to\_next\_mask, is\_almost\_end. Предположительно, критический путь будет связан с этими флагами.

**Обращаясь к САПР**

Частота при синтезе в файлах ограничений была установлена в 100 МГц

Исходя и slack на наихудший по времени критический путь, можно сказать, что приблизительная максимальная частота ~ 234797 МГц.

# Пути оптимизации

**Для повышения частоты**

1. Перебрать комбинационную логику. Высока вероятность, что такой большой массив сигналов можно пересобрать, сконфигурировать так, чтобы итоговая схема была оптимальнее в плане критических путей.
2. Внедрить в схему регистры-стадии, равномерно распределив их таким образом, чтобы путь прохождения сигнала по каждому из участков был примерно одинаковый. Таким образом путь от регистра до регистра уменьшится, и частота поднимется. К примеру, регистры можно поставить у приоритетных шифраторов для сигналов used\_mask\_low\_bit и crnt\_mask\_low\_bit, а также рядом с компараторами, образующих флаги.
3. На частоту влияет то, какой плотной является итоговая схема, поскольку чем плотнее схема, тем меньше время прохождения сигнала, тем выше частота. Добиться этого можно, если мы говорим о ПЛИС, непосредственно оперируя примитивами платы.
4. Поднять немного частоты, если будет использоваться конвейер и FIFO на выход, можно, используя кредитный счётчик, тем самым не протягивая сигнал ready от FIFO через конвейер.

**Для пропускной способности**

1. Для повышения пропускной способности можно поставить на выход схемы FIFO. Тогда мастер, не дожидаясь m\_ready, будет продолжать посылать пакеты, а слэйв, если он долго не принимал, быстро заберёт те, что успели накопиться в FIFO.