1. Порядок выполнения задания

Задача выполняется поэтапно. Переход на последующий шаг может быть выполнен только после завершения работ на предыдущем шаге.

Этапы выполнения задания

1. Проектирование. Результатом этапа является документ, включающий:
    - аргументация принятых при проектировании решений;
    - микроархитектурные диаграммы устройства и его подмодулей;
    - описание работы спроектированных модулей;
    - (опционально) использованные математические и алгоритмические трюки.

      Постарайтесь отразить в документе ход ваших мыслей при проектировании блока. Документ должен исчерпывающе пояснять, почему именно такие решения были приняты при проектировании. Детализация должна быть достаточной, чтобы передать кодирование этого блока другому инженеру.

2. Кодирование и bring-up тест. Результатом данного этапа является:
    - код на Verilog или System Verilog
    - testbench и набор тестовых векторов для среды ModelSim
    - (опционально) скрипт для запуска симулятора (Makefile или bash)
3. Тестовый синтез. Результатом данного этапа является:
    - скрипт (Makefile или bash) или список шагов для запуска синтеза
    - отчеты о синтезе для произвольного FPGA device из поддерживаемых Quartus lite
    - (опционально) анализ критических путей в дизайне
    - (опционально) рекомендации по изменению дизайна, которые могут повысить частоту и/или пропускную способность

Результаты выполнения должны быть размещены в репозитории на github.com и передаваться в виде ссылки на репозиторий. Вопросы по заданию можно задавать по email: interns.rtl@syntacore.com. Рекомендуем также написать перед началом выполнения задания, а также сообщить примерные сроки ожидания результатов.

2. Описание базового интерфейса

Table 1. Сигналы интерфейса

Сигнал	Ширина	Направление	Описание
valid	1	master → slave	Запрос на передачу 1 пакета данных
ready	1	slave → master	Готовность принять 1 пакет данных
data	задается параметром	master → slave	Передаваемые данные

last	1	master → slave	Флаг завершения транзакции

В отдельных вариантах тестового задания часть сигналов может отсутствовать, либо могут быть добавлены дополнительные сигналы.

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.

3. Тестовые задания (варианты)

3.1. Задача 2. Потоковый арбитр с политикой Quolity-of-Service (QoS)

На вход модуля по `STREAM_COUNT` параллельным независимым потокам поступают транзакции. Из всех входных транзакций выбирается наиболее приоритетная, которая принимается и транслируется на выход модуля. Сигнал `m_id_o` принимает значение номера входного потока, с которого получена выходная транзакция. Приоритет входящий транзакции определяется сигналом `s_qos_i`. Приоритет транзакции является общим для всех пакетов транзакции (не меняется от пакета к пакету). Чем значение выше, тем более приоритетной является транзакция. Если на вход модуля подано несколько транзакций с совпадающим приоритетом, то выбор из них осуществляется по принципу Round-Robin. Нулевое значение сигнала `s_qos_i` не участвует в схеме сравнения приоритетов: транзакции с `s_qos_i` равным нулю рассматриваются по схеме Round-Robin наравне с наиболее приоритетными. Арбитраж производится потранзакционно: переключение арбитра на следующий входной порт производится только после завершения передачи транзакции с текущего выбранного арбитром порта.

Интерфейс модуля

```
module stream_arbiter #(
    parameter  T_DATA_WIDTH = 8,
               T_QOS__WIDTH = 4,
               STREAM_COUNT = 2,
    localparam T_ID___WIDTH = $clog2(STREAM_COUNT)
)(
    input  logic                    clk,
    input  logic                    rst_n,
    // input streams
    input  logic [T_DATA_WIDTH-1:0] s_data_i  [STREAM_COUNT-1:0],
    input  logic [T_QOS__WIDTH-1:0] s_qos_i   [STREAM_COUNT-1:0],
    input  logic [STREAM_COUNT-1:0] s_last_i ,
    input  logic [STREAM_COUNT-1:0] s_valid_i,
    output logic [STREAM_COUNT-1:0] s_ready_o,
    // output stream
    output logic [T_DATA_WIDTH-1:0] m_data_o,
    output logic [T_QOS__WIDTH-1:0] m_qos_o,
    output logic [T_ID___WIDTH-1:0] m_id_o,
    output logic                    m_last_o,
    output logic                    m_valid_o,
    input  logic                    m_ready_i
);
```

Ниже приведен пример работы модуля для конфигурации

```
T_DATA_WIDTH = 4
T_QOS__WIDTH = 2
STREAM_COUNT = 2
```

В данном примере предполагается, что между slave-интерфейсом и master-интерфейсом модуля данные не буферизируются (глубина конвейера равна 0). Для спроектированного Вами модуля это может быть не так, это решение должно быть принято и обосновано на этапе проектирования.

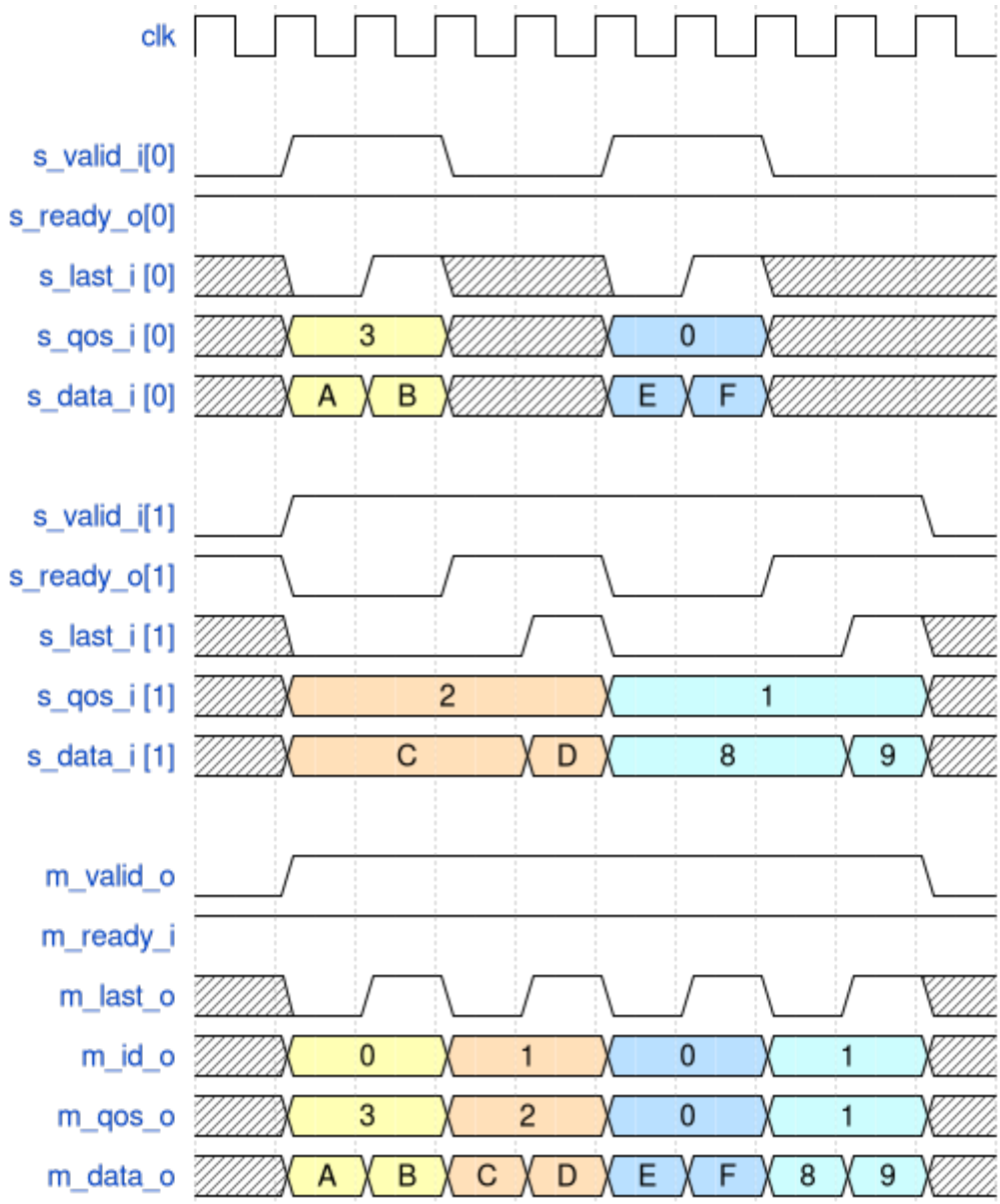

Figure 1. Пример работы арбитра