**1. Проектирование.**

*1.1. Аргументация принятых при проектировании решений*.

Алгоритм работы разрабатываемого устройства был задуман следующим:

1. Арбитр принимает на вход STREAM\_COUNT параллельных независимых потоков, после чего сортирует приоритеты потоков в порядке убывания по принципу Round-Robin (нулевой приоритет рассматривается наравне с наивысшим приоритетом).

2. После окончания сортировки арбитр выводит входные потоки в отсортированном порядке

3. После того, как последний поток был выведен, арбитр принимает новые потоки.

Ограничения данного алгоритма основываются на приведенном в задании примере:

1. Новые данные на уже освобожденные (прочитанные) потоки не приходят, пока не будут прочитаны все текущие потоки;

2. Приоритеты текущих потоков не меняются.

При проектировании после учета данных ограничений было принято решение использовать логику конечного однопроцессорного автомата Мили со следующими состояниями:

Состояние 1: сортировка входных приоритетов текущих потоков в порядке убывания по принципу Round\_Robin;

Состояние 2: вывод текущих потоков в отсортированном порядке.

В дальнейшем было принято решение добавить третье состояние, при котором арбитр готов к последующей обработке входящих потоков, но master-интерфейс пребывает в состоянии неготовности (m\_ready\_i = 0).

*1.2. Микроархитектурные диаграммы устройства и его подмодулей.*

Устройство состоит из одного цельного модуля, однако содержит логику сортировки вставками, которую также стоит упомянуть.

Диаграмма работы сортировки вставками приведена на рис. 1:

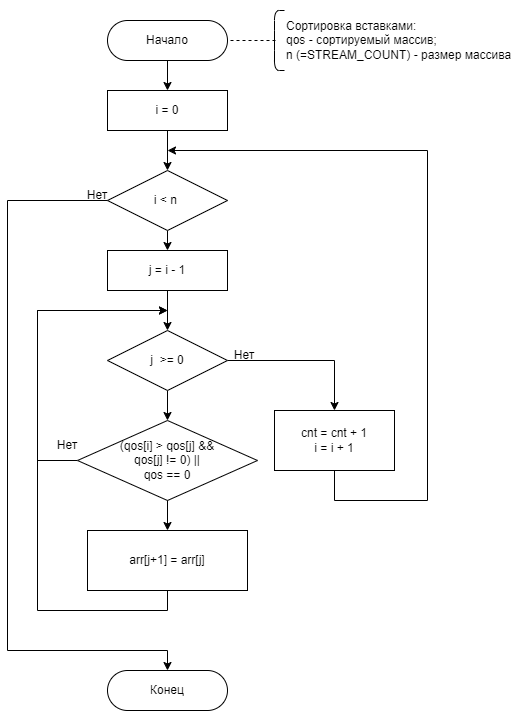


Рисунок . Диаграмма сортировки вставками

Сортировка вставками модифицирована таким образом, что потоки с приоритетом 0 сортировались наравне с наивысшими приоритетами. Алгоритм сортировки следующий:

1. Алгоритм проходит через каждый входной поток i от 0 до STREAM\_COUNT

2. Для каждого потока i он проверяет, установлен ли сигнал s\_valid\_i[i], что указывает на наличие данных для обработки.

3. Если s\_valid\_i[i] установлен в true, алгоритм начинает сортировку потоков на основе их приоритетов. Приоритеты определяются значениями в массиве s\_qos\_i.

4. Для сортировки используется вложенный цикл. Он перебирает ранее отсортированные потоки j, начиная с последнего отсортированного потока и двигаясь к первому.

5. Внутри внутреннего цикла алгоритм сравнивает приоритет текущего потока i (s\_qos\_i[i]) с приоритетом потока на позиции j в массиве отсортированных приоритетов (s\_qos\_i[sorted\_priorities[j]]).

6. Если приоритет потока i выше, чем приоритет потока на позиции j, и при этом приоритет на этой позиции не равен 0, то алгоритм меняет местами позиции потока i и потока на позиции j в массивах sorted\_priorities и sorted\_streams.

7. Переменная sorted\_count увеличивается для отслеживания количества отсортированных потоков (если не все потоки валидны).

8. Если приоритеты не упорядочены в необходимом порядке, внутренний цикл завершается, и алгоритм переходит к следующему потоку i.

Этот алгоритм сортирует входные потоки по их приоритетам в порядке убывания, при этом обрабатывая 0 по принципу Round-Robin с наивысшими приоритетами. Используются два внутренних массива: sorted\_priorities для отслеживания отсортированных приоритетов и sorted\_streams для отслеживания соответствующих индексов потоков.

Диаграмма работы устройства приведена на рис. 2:

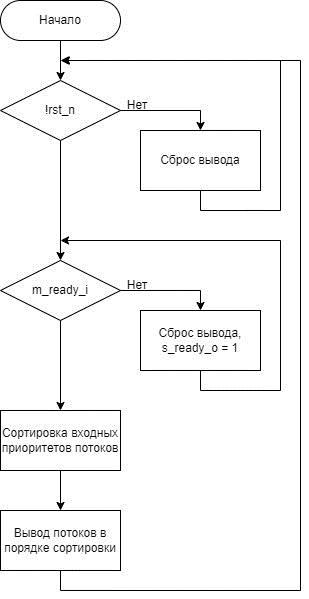


Рисунок . Диаграмма работы устройства

Устройство не имеет конечного состояния. Алгоритм работы устройства соответствует принятым при проектировании решений.

Единственным математическим приемом, использованным в данном задании, было использование сортировки вставками. Однако работу модуля можно улучшить, если заменить алгоритм сортировки на более быстрый (например, сортировку слиянием или быструю сортировку).

**2. Кодирование и bring-up тест.**

2.1. Код System Verilog представлен в файле **stream\_arbiter.sv**

2.2. Testbench с набором тестовых векторов для среды ModelSim представлен в файле **tb\_stream\_arbiter.v.** Результат запуска testbench представлен на рис. 3:

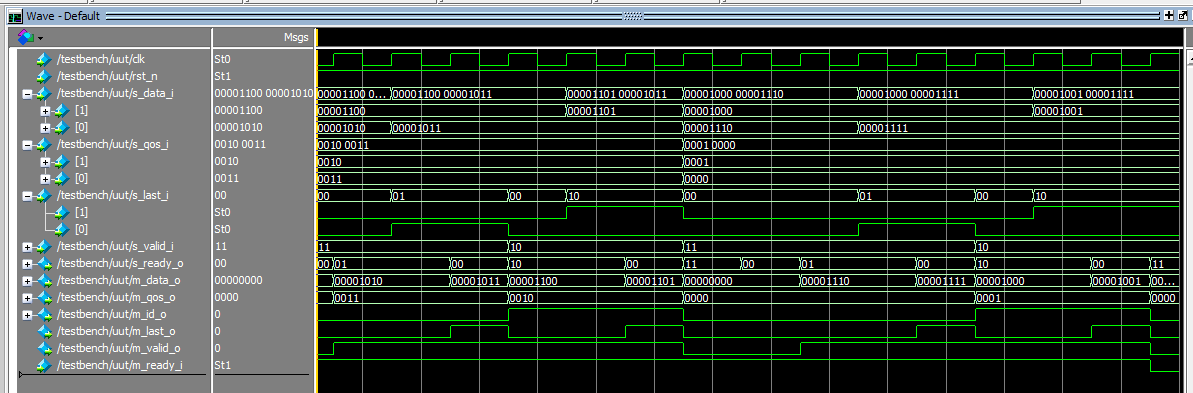


Рисунок . Testbench разработанного устройства

Как видно из представленной диаграммы, в отличие от представленного в задании примера, устройство выдает задержку в 1 такт после принятия всех потоков и перед выводом данных. Данная задержка объясняется тем, что устройству необходимо время для формирования отсортированного массива.

**3. Тестовый синтез.**

3.1. Для запуска синтеза необходимо создать проект в среде Quartus, добавить файл **stream\_arbiter.sv** в качестве основного модуля и выполнить компиляцию.

3.2. Отчеты о синтезе для произвольного FPGA device из поддерживаемых Quartus II представлены на рис. 4-5:

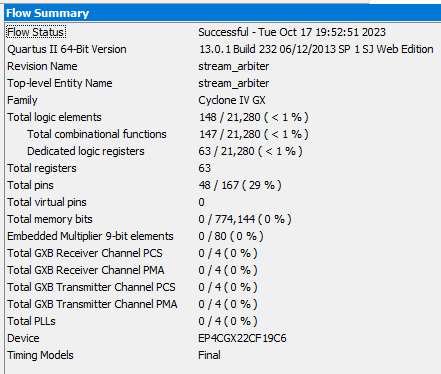


Рисунок . Результаты компиляции

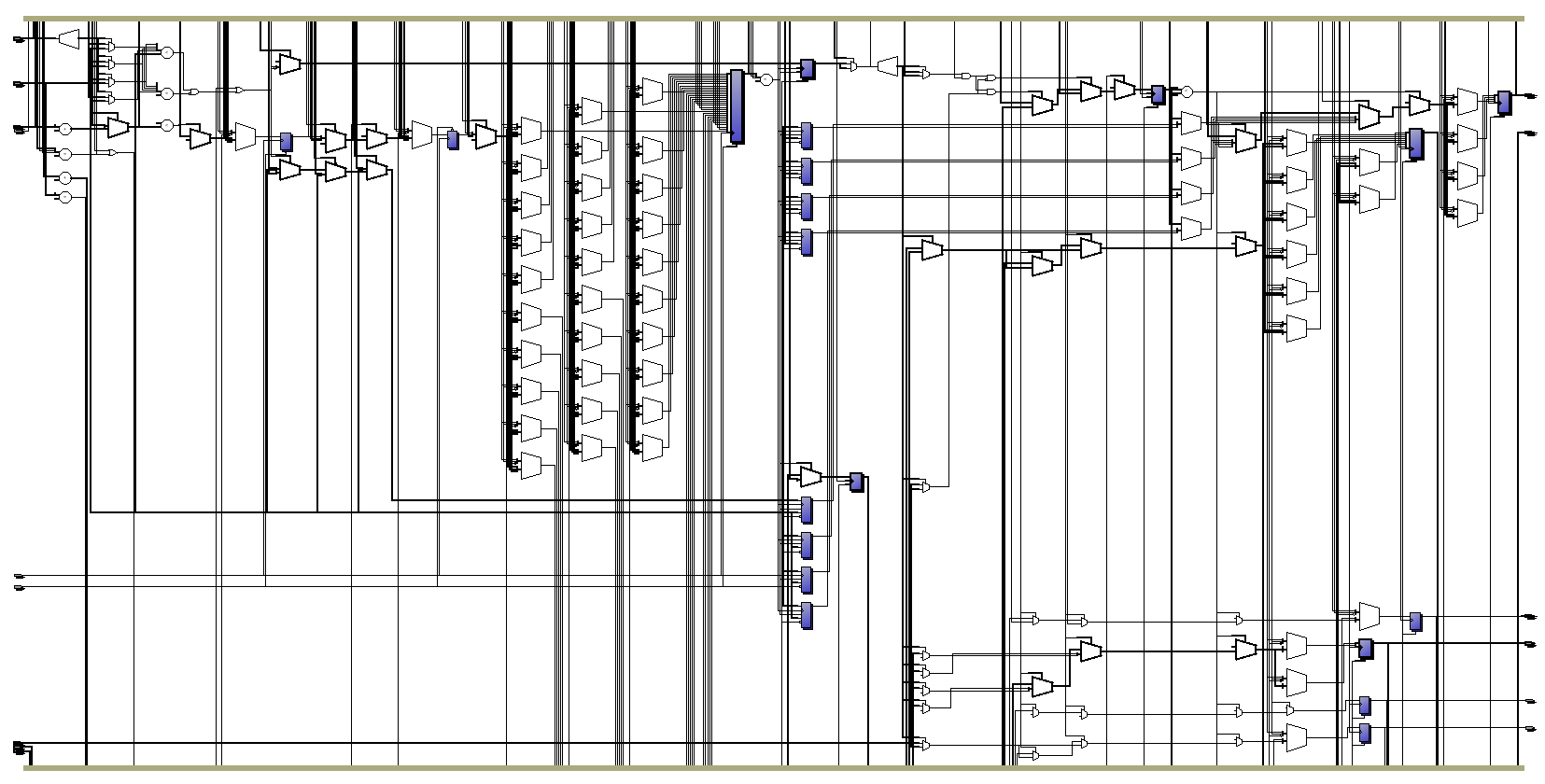


Рисунок . RTL-диаграмма