Декомпозиция задачи

1. Slave + Логика формирования записи буфера
2. Master + Логика формирования выходного пакета
3. Внутренний буфер
    a. Блок памяти
    b. Логика адресации и выбора выходного пакета
    c. Логика приостановки получения

О выборе аппаратной реализации

В проекте должен быть сделан упор на быстродействие => необходимо минимизировать число тактов, в которых s_ready_o=0 и m_valid_o=0.

Реализация внутреннего буфера необходима (как минимум, на 1 входной пакет), так как интерфейс предусматривает возможность, что следующий слейв на линии выставит сигнал ready=0, поэтому реализуемый модуль должен сохранить этот пакет. Больший размер усложнит реализацию, но увеличит быстродействие, позволив потоковую передачу (без установки m_valid_o в 0).

Интерфейс синхронный, но поскольку синхросигнал не включен в шину интерфейса, за него следует принять сигнал clk. Т. е. приём пакета должен делаться за 1 такт.

Принято решение не удалять незначащие элементы из потока, так как невозможно удалить их за один такт. Исключением являются пустые выходные пакеты: они должны быть пропущены.

Если M_KEEP_WIDTH> S_KEEP_WIDTH, то пакеты должны пропускаться как есть, а для свободных линий m_keep_o[i] должен устанавливаться в 0.

Модуль должен быть сделан по принципу конвейера с длительностью стадии 1 такт:

1. Формирование записи буфера из входного пакета
2. Отправка записи в буфер
3. Чтение записи из буфера (и пропуск, если выходной пакет полностью состоит из нулей); формирование выходного пакета
4. Отправка выходного пакета

Описание подмодулей

Slave

```
module slave #(
      parameter   S_KEEP_WIDTH = 3,
                  T_DATA_WIDTH = 1,
                  BUF_IN_ENTRY_SZ = (2 + T_DATA_WIDTH)* S_KEEP_WIDTH
)(
      input clk,
      input s_valid_i,
      input s_last_i,
      input [S_KEEP_WIDTH-1:0] s_keep_i,
      input [T_DATA_WIDTH-1:0] s_data_i [S_KEEP_WIDTH],
      output wire s_ready_o,
      input en,
      output reg slave_entry_valid,
      output reg [BUF_IN_ENTRY_SZ-1:0] slave_entry
```

```
);
```

Первые 6 сигналов реализуют базовый интерфейс; slave_entry – это готовая запись буфера. Важно: строка буфера может содержать несколько таких записей (в строке буфера их помещается целое число). slave_entry_valid – бит готовности данных в slave_entry.

Параметр BUF_IN_ENTRY_SZ определяется структурой буфера. Его значение равно размеру входной записи буфера, то есть размер пакета плюс набор флагов keep и last.

Буфер

Буфер представляет собой блок памяти длины BUF_IN_ENTRY_SZ*BUF_OUT_ENTRY_SZ*M, где M – множитель, регулирующий длину буфера (задаётся извне).

На вход поступает массив из записей, число которых равно S_KEEP_WIDTH, а их структура имеет вид:

Флаг keep	Флаг last	data[i]
1 бит	1 бит	T_DATA_WIDTH

Таким образом, BUF_IN_ENTRY_SZ = (2 + T_DATA_WIDTH)* S_KEEP_WIDTH.

Флаг last выставляется в 1 только для последнего блока в последнем пакете транзакции.

Адресация

Адресация кольцевая, по принципу FIFO; при переполнении адреса он обнуляется, возвращаясь к началу блока памяти.

Существует два адреса: для чтения и записи. Адрес записи может быть увеличен только на максимальное из значений {BUF_IN_ENTRY_SZ, BUF_OUT_ENTRY_SZ}, чтобы обеспечить единообразие для любых соотношений этих величин.

Адрес чтения инкрементируется на BUF_OUT_ENTRY_SZ, в том числе, если пакет последний. Такие случаи обрабатываются в подмодуле master.

Управление адресом происходит внутри буфера: адреса чтения и записи инкрементируются по фронту clk если соответственно slave_entry_valid или master_entry_ready равно 1.

Случай overflow

Переполнение наступает, если следующая запись в буфер перепишет валидные данные. В этом случае необходимо остановить получение, подняв сигнал overflow.

Случай underflow

Сигнал underflow равен 1, когда следующее чтение пересечёт точку записи, и будут прочитаны невалидные данные. Для случая S_KEEP_WIDTH>= M_KEEP_WIDTH этот сигнал всегда равен 1.

Интерфейс

```
module buffer #(
      parameter   S_KEEP_WIDTH = 3,
                  T_DATA_WIDTH = 1,
                  M_KEEP_WIDTH = 7
)(
      input clk,
      input slave_entry_valid,
      input [BUF_IN_ENTRY_SZ-1:0] slave_entry,
      input master_entry_ready,
      output reg [BUF_OUT_ENTRY_SZ-1:0] master_entry,
      output wire overflow,
      output wire underflow
```

```
);
```

Сигнал overflow равен 1, когда адрес чтения больше адреса записи на BUF_IN_ENTRY_SZ.

Сигнал master_entry_valid становится равным 0, если адрес чтения равен адресу записи.

Допустима реализация этого блока на базе мегафункции Quartus.

Master

```
module master#(
      parameter   S_KEEP_WIDTH = 3,
                  T_DATA_WIDTH = 1,
                  M_KEEP_WIDTH = 7
)(

      input clk,
      output m_valid_o,
      input m_ready_i,
      output m_last_o,
      output [M_KEEP_WIDTH-1:0] m_keep_o,
      output [T_DATA_WIDTH-1:0] m_data_o [M_KEEP_WIDTH],
      output wire master_entry_ready,
      input [BUF_OUT_ENTRY_SZ-1:0] master_entry
);
```

Аналогично, первые 6 сигналов – базовый интерфейс.

Об отслеживании последнего пакета транзакции

Если последний блок последнего пакета транзакции не приходится на конец выходного пакета, а сигнал underflow равен 0, то master должен выполнить следующие шаги:

1. Приостановить получение следующего пакета из буфера
2. Сформировать последний пакет транзакции, установив в 0 биты keep для тех блоков, которые идут после блока с поднятым битом last; выполнить отправку
3. Инвертировать биты keep этого же пакета; отправить
4. Возобновить получение

Если underflow равен 1, master должен выполнить только шаг 2.

Схема устройства