УДК 681.3.06 (07)
Р 851

кафедра математического обеспечения и применения ЭВМ

ФАВТ

№ 3163

МИНИСТЕРСТВО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ТАГАНРОГСКИЙ ГОСУДАРСТВЕННЫЙ
РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Учебно-методическое пособие
по курсу
АРХИТЕКТУРА И ПРОГРАММИРОВАНИЕ
СИГНАЛЬНЫХ ПРОЦЕССОРОВ

РАБОТА С КОММУНИКАЦИОННЫМИ ПОРТАМИ SHARC-ПРОЦЕССОРОВ

Для студентов специальностей 22040, 351500

Таганрог 2001

УДК 681.3.06 (076.5) + 681.325.5. (076.5)

Составитель: Н.Ш. Хусаинов

Учебно-методическое пособие по курсу “Архитектура и программирование сигнальных процессоров". Работа с коммуникационными портами SHARC-процессоров. Таганрог: Изд-во ТРТУ, 2001. 50с.

Предназначено для студентов специальности 2204, изучающих курс "Архитектура и программирование сигнальных процессоров". Содержит описание архитектуры, основных возможностей и принципов программирования последовательных и линк-портов SHARC-процессоров фирмы Analog Devices Inc.

Ил.6. Библиогр.: 6 назв.

Рецензент П.П.Кравченко, д-р. техн. наук, профессор кафедры МОП ЭВМ ТРТУ.

Хусаинов Наиль Шавкятович

Учебно-методическое пособие
по курсу
АРХИТЕКТУРА И ПРОГРАММИРОВАНИЕ
СИГНАЛЬНЫХ ПРОЦЕССОРОВ

РАБОТА С КОММУНИКАЦИОННЫМИ ПОРТАМИ SHARC-ПРОЦЕССОРОВ

Для студентов специальности 220400, 351500

Ответственный за выпуск Хусаинов Н.Ш.
Редактор Белова Л.Ф.
Корректор Пономарева Н.В.

ЛР № 020565 от 23.06.1997г. Подписано к печати . .2001г.
Формат 60x84 1/16 Бумага офсетная. Гарнитура литературная.
Офсетная печать. Усл.п.л. — 3. Уч.-изд. л. — 3,1.
Заказ № Тираж 200 экз.

"С"

Издательство Таганрогского государственного
радиотехнического университета
ГСП 17А, Таганрог, 28, Некрасовский, 44
Типография Таганрогского государственного
радиотехнического университета
ГСП 17А, Таганрог, 28, Энгельса, 1

1. РАБОТА С ПОСЛЕДОВАТЕЛЬНЫМИ ПОРТАМИ SHARC-ПРОЦЕССОРОВ СЕМЕЙСТВА ADSP-2106x

В ADSP-21060 имеются два независимых последовательных порта (serial port) SPORT0 и SPORT1, обеспечивающих интерфейс процессора с различными периферийными устройствами ввода/вывода. Последовательные порты обеспечивают побитовый обмен данными с максимальной скоростью до *n* Мбит/с, где *n* - тактовая частота процессора, и характеризуются следующими возможностями:

- независимый прием и передача данных каждым портом;
- буферизация данных при передаче и двойная буферизация при приеме данных;
- аппаратное компандирование по A- и μ-законам при передаче и приеме[1];
- передача и прием слов данных размером от 3 до 32 битов, начиная со старших (Most Significant Bit, MSB) или с младших (Least Significant Bit, LSB) битов;
- использование независимых тактирующих сигналов для передачи и приема каждого бита (clock sync) и каждого слова (frame sync), причем синхронизация может быть либо внутренней (тактовые импульсы генерируются самим процессором), либо внешней (передача или прием с частотой тактовых импульсов, генерируемых внешним устройством);
- пересылка данных как по одному слову под управлением процессорного ядра, так и блоками данных с использованием DMA-контроллера;
- многоканальный режим приема и передачи данных с временным разделением битового потока.

1.1. Структура и принципы функционирования последовательных портов

Структура последовательного порта SPORTn приведена на рис.1. Передаваемые данные помещаются в регистр TXn, затем подвергаются компандированию (при необходимости) и автоматически сдвигаются в передающий регистр сдвига, из которого побитно, в соответствии с тактовыми импульсами TCLKn, выводятся на ножку DTn процессора. Если используется режим кадровой синхронизации, то начало передачи очередного слова данных инициируется сигналом TFSn.

При приеме данных выполняется обратная последовательность действий.

[1] Компандирование – процесс представления данных через логарифмическую шкалу квантования для сокращения числа битов, необходимых для передачи. Последовательные порты ADSP-2106x поддерживают два стандартных алгоритма компандирования, используемых для представления речевых сигналов в телефонных сетях: по A- и μ-законам.

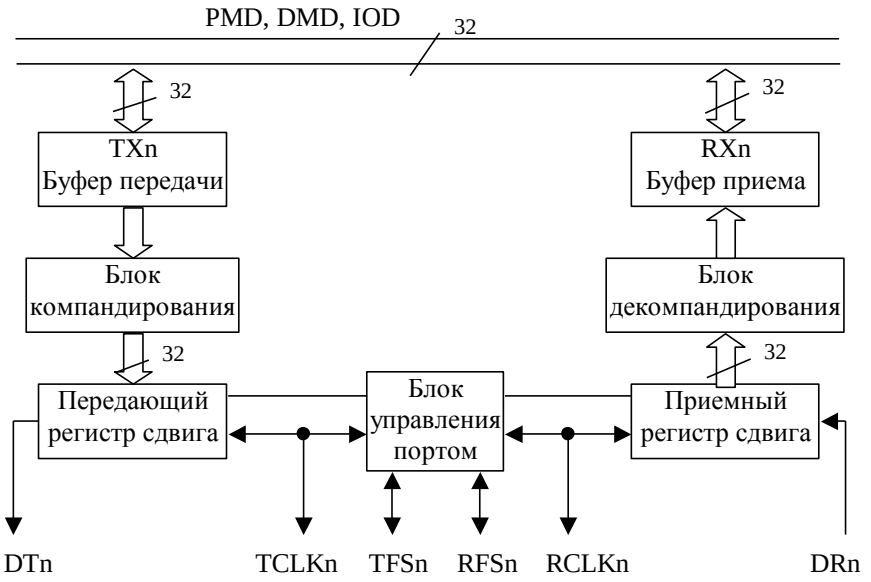

Рис. 1

1.2. Регистры управления и буферы данных последовательных портов

1.2.1. Регистры управления работой последовательных портов

Регистры конфигурирования и управления работой последовательных портов являются частью множества регистров IOP-процессора. Каждый последовательный порт имеет собственный набор 32-битных регистров управления и буферов данных.

Последовательный порт программируется путем записи нужных значений по соответствующим адресам памяти, в которых отображаются регистры последовательного порта. При модификации значений регистров управления последовательным портом изменения вступают в силу через один такт после операции записи. Таким образом, SPORTn будет готов начать передачу или прием данных через два цикла после того, как он будет разрешен установкой соответствующего бита в STCTLn или SRCTLn.

Регистр	Функция
STCTLn	регистр управления передачей через SPORTn
TXn	буфер данных при передаче
TDIVn	тактовый и кадровый делители частоты при передаче
MTCSn	выбор каналов при многоканальной передаче
MTCCSn	режим компандирования для каждого канала при многоканальной передаче
SRCTLn	регистр управления приемом
RXn	буфер данных при приеме через SPORTn
RDIVn	тактовый и кадровый делители частоты при приеме
MRCSn	выбор каналов при многоканальном приеме
MRCCSn	режим компандирования для каждого канала при многоканальном приеме
SPATHn	длина пути (для многопроцессорной системы). Разрядность – 16 бит
KEYWDn	режим сравнения (только для ADSP-21061)
KEYMASKn	маска для режима сравнения (только для ADSP-21061)

Основными регистрами управления последовательным портом являются его регистры STCTL и SRCTL.

Регистр STCTL

Биты	Поле	Функция
0	SPEN	разрешение передачи через последовательный порт
1-2	DTYPE	формат передаваемых данных
3	SENDN	направление передачи (=1 – с младших битов слова; =0 – со старших битов слова)
4-8	SLEN	= длина передаваемых слов – 1
9	PACK	распаковка данных (передавать одно 32-битное слово как два 16-битных)
10	ICLK	внутренняя генерация сигнала синхронизации данных на передачу
11	---	зарезервировано
12	CKRE	синхронизация по переднему/заднему фронту сигналов синхронизации данных и кадров при передаче
13	TFSR	режим кадровой синхронизации при передаче
14	ITFS	внутренняя генерация сигнала кадровой синхронизации при передаче
15	DITFS	зависимость сигнала кадровой синхронизации от данных при передаче
16	LTFS	активный уровень сигнала кадровой синхронизации (=0 – высокий, =1 – низкий) при передаче
17	LAFS	поздний сигнал кадровой синхронизации при передаче
18	SDEN	включение DMA-пересылки при передаче
19	SCHEN	разрешение цепочечной DMA при передаче
20-23	MFD	задержка кадра в многоканальном режиме при передаче
24-28	CHNL	статус текущего канала при передаче
29	TUVF	флаг потери значимости (underflow) в буфере передачи
30-31	TXS	статус буфера передачи (=11 – буфер полон, =00 – буфер пуст, =10 – буфер частично заполнен)

Регистр SRCTL

Биты	Поле	Функция
0	SPEN	разрешение приема через последовательный порт
1-2	DTYPE	формат принимаемых данных
3	SENDN	направление приема (=1 – с младших битов слова; =0 – со старших битов слова)
4-8	SLEN	= длина принимаемых слов – 1
9	PACK	упаковка данных (принимать два 16-битных слова как одно 32-битное)
10	ICLK	внутренняя генерация сигнала синхронизации данных при приеме
11	---	зарезервировано
12	CKRE	синхронизация по переднему/заднему фронту сигналов синхронизации данных и кадров при приеме
13	RFSR	режим кадровой синхронизации при приеме
14	IRFS	внутренняя генерация сигнала кадровой синхронизации при приеме
15	IMODE	режим сравнения при приеме данных (только для ADSP-21061)
16	LRFS	активный уровень сигнала кадровой синхронизации (=0 – высокий, =1 – низкий) при приеме
17	LAFS	поздний сигнал кадровой синхронизации при приеме
18	SDEN	включение DMA-пересылки при приеме
19	SCHEN	разрешение цепочечной DMA при приеме
20	IMAT	режим принятия значения в режиме со сравнением (=1 – принять слово, если результат сравнения ложный, =0 –если истинный)

21	D2DMA	разрешить режим 2-мерной DMA
22	SPL	подавать на вход значения с выхода того же порта (режим loopback, используется для тестирования)
23	MCE	разрешение многоканального режима
24-28	NCH	количество каналов – 1 (в многоканальном режиме)
29	ROVF	флаг переполнения (overflow) буфера приема
30-31	RXS	статус буфера приема (=11 – буфер полон, =00 – буфер пуст, =10 – буфер частично заполнен)

Для тестирования программ, использующих обмен данными через последовательный порт, может использоваться режим зацикливания порта, устанавливаемый битом SPL в регистре SRCTLn. В этом режиме сигналы на линиях DRx, RCLKx и RFSx порта всегда равны сигналам DTx, TCLKx и TFSx того же порта. Активными являются только сигналы передающей части последовательного порта, тогда как сигналы его приемной части игнорируются. Многоканальный режим при зацикливании не поддерживается.

1.2.2. Буферы приема и передачи данных

Регистры TX0 и TX1 являются буферами данных при передаче соответственно через последовательные порты SPORT0 и SPORT1. В эти 32-битные регистры загружаются подлежащие передаче значения либо DMA-контроллером, либо командой, выполняемой процессорным ядром, например:

```
dm(TX0) = R0; /* запись в регистр TX последовательного порта
                 SPORT0 значения из регистра R0 */
```

Каждый TX-регистр функционирует как двухуровневый FIFO-буфер, в котором одновременно могут находиться два слова данных. Когда завершается передача очередного слова из регистра сдвига и в TX-регистре уже находится следующее значение, содержимое буфера автоматически сдвигается в передающий регистр сдвига. При этом может генерироваться прерывание, которое показывает, что буфер TX готов к приему очередного значения от процессорного ядра (состояние "буфер не полон"). Однако прерывания не происходит, если, например, заполнение передающего регистра последовательного порта выполняется с использованием DMA-пересылки.

Возможна ситуация, когда в момент генерации сигнала кадровой синхронизации, означающего начало передачи нового слова, регистр TX не содержит нового слова. В этом случае устанавливается статусный бит потери значимости TUVF в регистре управления передачей STCTLn, соответствующем данному последовательному порту. Этот бит является "липким" и сбрасывается только при запрещении пересылок через SPORTn.

RX-регистр функционирует подобно трехуровневому FIFO-буферу, состоящему из двух регистров данных и приемного регистра сдвига размерностью по 32 бита каждый. При получении всех битов слова в приемном регистре сдвига оно переносится из регистра сдвига во второй регистр данных. При чтении из RX-регистра, выполняемом в очередной инструкции или при DMA-пересылке, слово из второго регистра данных сдвигается в первый регистр данных. Если при завершении приема третьего слова первое слово еще не прочитано из RX-регистра DMA-контроллером или процессорным ядром, третье слово будет записано поверх второго, т.е. произойдет потеря данных. Для отслеживания такой ситуации при приеме последнего бита третьего (последнего) слова, когда оба регистра данных заняты, генерируется "липкий" статусный бит переполнения ROVF в соответствующем регистре управления приемом SRCTLn, который может быть сброшен только запрещением пересылок через последовательный порт.

Прерывание приема данных генерируется в момент, когда в RX-буфер заносится полученное значение (состояние "буфер не пуст"). Прерывания не происходит, если пересылка данных из последовательного порта во внутреннюю память выполняется DMA-контроллером.

Внимание! При попытке записи процессорным ядром слова данных в полный TX-буфер (или чтения ядром пустого RX-буфера) выполнение программы приостанавливается до освобождения TX-регистра (или заполнения RX-регистра). Чтобы избежать этого, перед обращением к буферу необходимо проанализировать значение его статуса ("полон", "пуст", "частично заполнен"). Например:

```
...
R1 = 30;          // для записи в TX проверить младший бит статуса
R0 = DM(STCTL0);
BTST R0 BY R1;
IF NOT SZ JUMP(PC, -2); // если =1, значит TX-буфер еще полон
dm(TX0) = R2;           // иначе – записать значение
...
```

Статусные биты в регистрах STCTLn и SRCTLn изменяются каждый раз при записи или чтении буфера процессорным ядром, даже если последовательный порт не включен. При записи в RX-буфер или чтении из TX-буфера без передачи данных через порт (для выполнения операций аппаратного компандирования или декомпандирования) соответствующий последовательный порт должен быть запрещен.

1.2.3. Режимы синхронизации

1.2.3.1. Тактовая синхронизация

Сигналы тактовой синхронизации конфигурируются битами ICLK в реги-

страх управления STCTLn и SRCTLn и значениями регистров TDIVn и RDIVn.

Если ICLK=0, то внешний тактовый сигнал принимается на линии TCLKn или RCLKn, а значения регистров TDIVn и RDIVn игнорируются. Внешний тактовый сигнал не обязательно должен быть синхронизирован с внутренней тактовой частотой самого ADSP.

Если ICLK=1, то ножки TCLKn и RCLKn рассматриваются как выходы, а частоты генерации внутренних тактовых импульсов на передачу и прием определяются младшими 16 битами регистров TDIVn (поле TCLKDIV) и RDIVn (поле RCLKDIV) соответственно. Значение в поле xCLCKDIV позволяет определить частоту передачи или приема битов через последовательный порт относительно частоты ядра процессора (CLKIN) по следующей формуле:

```
Тактовая частота передачи/приема = CLKIN / (xCLKDIV+1).
```

Максимальная частота передачи/приема данных через SPORTn равна частоте ядра ADSP-21060 и достигается в случае, когда значение поля xCLKDIV равно 0.

Например, необходимо рассчитать значение RCLKDIV для приема 32-разрядных значений с частотой 1000 Гц через последовательный порт.

В этом случае количество битов, принимаемых в секунду через порт (частота приема), равно 1000 x 32 = 32 КГц = 32000 битов.

Тактовая частота процессора: 40 МГц.

Делитель тактовой частоты RCLKDIV = 40000000 / 32000 – 1 = 1249.

Для правильной передачи данных между двумя процессорами передающий последовательный порт обычно конфигурируется как синхронизируемый внутренним тактовым сигналом, а принимающий – как синхронизируемый внешним тактовым сигналом (сигналом передающего порта). Это связано также с ограничениями ADSP-21060 при передаче данных по внешним синхроимпульсам на частоте, приближающейся к внутренней частоте ядра процессора[2].

1.2.3.2. Кадровая синхронизация

Сигналы кадровой синхронизации TFSn и RFSn отмечают начало передачи и приема каждого слова через последовательный порт. Необходимость режима кадровой синхронизации задается битами TFSR и RFSR в регистрах STCTLn и SRCTLn соответственно.

Если бит TFSR или RFSR установлен, то передача или прием очередного слова начинается только после сигнала кадровой синхронизации. Для обеспечения непрерывной передачи данных следующее слово должно быть записано в TX-регистр до окончания передачи текущего слова из регистра сдвига. Если TFSR=0 или RFSR=0, соответствующий сигнал кадровой синхронизации для начала передачи каждого слова не требуется. Единственный сигнал кадровой синхронизации требуется для инициирования процесса передачи, после чего он игнорируется (рис. 2).

Как и в случае тактовой синхронизации, каждый из сигналов кадровой синхронизации независимо друг от друга может либо генерироваться самим процессором с заданной частотой (при ITFS=1 или IRFS=1) и выводиться на линии TFSn или RFSn, либо приниматься от внешнего устройства на линии TFSn или RFSn (соответственно при ITFS=0 или IRFS=0). В первом случае для определения частоты кадровой синхронизации для передачи и приема используются старшие 16 битов регистров TDIVn (поле TFSDIV) и RDIVn (поле RFSDIV).

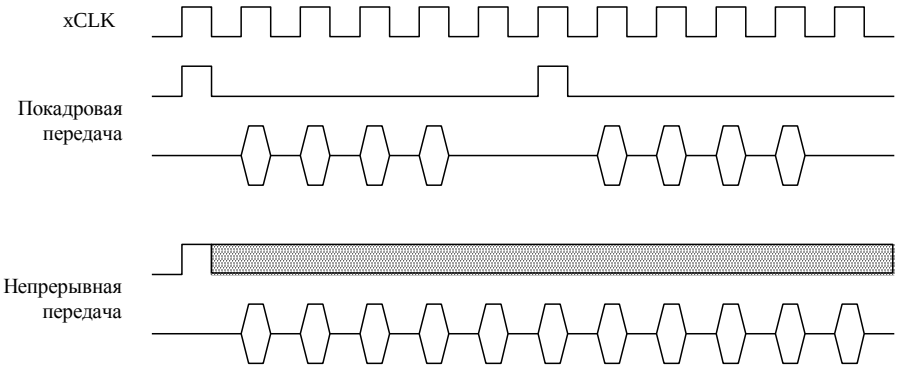

Рис.2

Поля TFSDIV и RFSDIV задают количество сигналов тактовой синхронизации порта (сгенерированных процессором или полученных извне) между генерациями внутренних сигналов кадровой синхронизации TFSn и RFSn. Количество сигналов тактовой синхронизации между генерацией сигналов кадровой синхронизации равно xFSDIV+1. Значение, записанное в поле xFSDIV, не должно быть меньше значения поля SLEN в регистре STCTLn или SRCTLn.

```
xFSDIV = тактовая частота порта / частота кадровой синхр. – 1
```

Сигнал кадровой синхронизации может генерироваться или тестироваться

---

[2] Рекомендуется использовать частоту обмена данными, совпадающую с частотой самого ADSP-2106x только при приеме данных с внешними источниками генерации тактовых импульсов (ICLK=0, IRFS=0).

Также следует проявлять осторожность при использовании цепочечной DMA для передачи/приема через последовательный порт слов длиной меньше 7 бит, поскольку во время загрузки TCB-блока внутренняя I/O-шина занята в течение нескольких тактов. Передаваемые или принимаемые в этот период данные могут быть потеряны.

либо в цикле тактовой синхронизации порта, предшествующем передаче/приему первого бита слова ("ранняя кадровая синхронизация" – обычный режим работы, бит LAFS=0), либо в том же цикле тактовой синхронизации, в котором происходит передача/прием первого бита слова ("поздняя кадровая синхронизация" – альтернативный режим работы, бит LAFS=1).

В режиме "ранней синхронизации" работы первый бит передается (или защелкивается при приеме) в цикле тактовой синхронизации следующим после цикла, в котором сигнал кадровой синхронизации имел активное состояние. Сигнал кадровой синхронизации тестируется в следующий раз только для начала передачи/приема следующего слова. При непрерывной передаче/приеме данных (первый бит следующего слова передается/принимается сразу после последнего бита предыдущего слова) и установленном режиме "ранней синхронизации" сигнал кадровой синхронизации проверяется при передаче/приеме последнего бита каждого слова и не приводит к задержкам. Продолжительность внутреннего сигнала кадровой синхронизации – 1 такт.

Если бит LAFS=1, первый бит слова передается (или защелкивается при приеме) в том же цикле тактовой синхронизации порта, в котором выставлен сигнал кадровой синхронизации. В отличие от ранней синхронизации, в альтернативном режиме внутренний сигнал кадровой синхронизации остается активным при передаче/приеме всего слова, а внешний проверяется только при передаче/приеме первого бита.

По умолчанию, внутренний сгенерированный сигнал кадровой синхронизации передачи TFSn выставляется, только когда TX-буфер содержит следующее слово для передачи. Когда очередное слово записывается в TX-регистр, оно начнет передаваться только после генерации сигнала TFSn.

Для генерации сигнала кадровой синхронизации с частотой, указанной в поле xFSDIV, независимо от наличия слова в TX-буфере (например, для синхронизации внешнего устройства без передачи ему данных) необходимо установить бит DITFS в регистре STCTLn. При этом содержимое TX будет передаваться вновь с каждой новой генерацией сигнала xFSDIV.

При выставлении TFSn сигнала (внутреннего или внешнего) и отсутствия новых данных в TX-регистре устанавливается "липкий" флаг TUVF (регистр STCTLn).

1.3. Форматы данных, передаваемых через последовательные порты

Форматы слов данных, передаваемых через последовательный порт, определяются битами DTYPE, SENDN, SLEN и PACK в регистрах STCTLn и SRCTLn.

Длина слова, передаваемого или принимаемого через последовательный порт, задается 5-битовым полем SLEN, в которое заносится количество разрядов в слове, уменьшенное на 1. Поле SLEN не может быть меньше 2 (длина слова не менее 3 бит). Слова длиной менее 32-х разрядов выравниваются по правому краю в регистрах RX и TX.

Принимаемые данные размерностью 16 битов и менее могут упаковываться в 32-битные слова, а передаваемые 32-битные слова могут выводиться как два 16-битных слова. Если бит PACK=1 в регистре SRCTLn, два последовательно принятых слова упаковываются в одно, причем первое принятое слово будет находиться в младших 16 битах, а второе – в старших 16 битах. Оба слова будут выровнены к правым границам своих полей. При передаче данных распаковка выполняется аналогично (если установлен бит PACK в регистре STCTLn). В этом случае прерывания по приему или передаче слова генерируются для 32-битного слова, а не для каждого 16-битного слова.

Внимание! Режим с упаковкой данных целесообразно использовать для снижения загрузки I/O-шины, поскольку в этом случае за одну пересылку передается два 16-разрядных слова. При этом каждая половинка упакованного 32-битного слова, полученного через последовательный порт и записанного в пространство памяти нормальных адресов, может быть доступна через пространство памяти коротких адресов.

Поле DTYPE задает один из четырех форматов данных, которые могут быть переданы через последовательный порт.

Значение DTYPE	**Формат данных**
в одноканальном режиме	
00	выравнивание вправо, неиспользуемые биты равны 0
01	выравнивание вправо, знаковое расширение
10	компандирование по μ-закону
11	компандирование по A-закону
в многоканальном режиме	
x0	выравнивание вправо, неиспользуемые биты равны 0
x1	выравнивание вправо, знаковое расширение
0x	компандирование по μ-закону
1x	компандирование по A-закону

Эти форматы применяются к данным, находящимся в TX- и RX-регистрах.

Примечание. Для многоканального режима выбор типа компандирования и заполнение старших битов устанавливается независимо друг от друга. В этом случае передача значений в линейном формате осуществляется, когда канал активен и режим компандирования для данного канала в регистре MTCCSx или MRCCSx выключен (см. ниже). Режим знакового расширения (нулем или знаковым битом) одинаков для всех каналов при приеме и для всех каналов при передаче и определяется младшим битом поля DTYPE. Если включен режим знакового расширения, то оно выполняется только для тех каналов, для которых не включено компандирование. В противном случае старшие биты слов будут заполнены нулями.

Запись в TX-регистры 32-битного значения приводит к его компрессии до 8-битового значения, расширенному со знаком до длины слова, подлежащего пе-

редаче. Если исходное 32-битное значение больше максимально допустимого при компандировании по A- или μ-закону (соответственно максимального 14-разрядного и 13-разрядного значения), оно преобразуется в максимально возможное значение с учетом знака.

Аналогично, при приеме через последовательный порт в режиме компандирования получаемые 8-битные данные декомпандируются и записываются в 32-разрядный RX-регистр со знаковым расширением.

Аппаратные схемы компандирования и декомпандирования могут быть использованы и без передачи или получения данных через последовательный порт, например в целях тестирования или отладки программы.

```
...              // передача через порт должна быть запрещена
R0 = 0x0000000;  // разрешить компандирование
dm(STCTL0) = R0; // записать значение в регистр управления SPORT0
dm(TX0) = R1;    // записать исходное 32-битное значение в TX0
nop;     // 1 такт чтобы новое значение перезаписалось в TX0
R1 = dm(TX0);    // прочитать 8-битовое компандированное значение
...
```

Аналогичная последовательность действий применяется для выполнения аппаратного декомпандирования без передачи через последовательный порт, но в этом случае необходимо использовать RXn-регистр.

1.4. Организация многоканального режима работы последовательного порта

1.4.1. Конфигурирование последовательного порта для работы в многоканальном режиме

Многоканальный режим работы последовательного порта позволяет организовывать обмен данными с последовательными системами передачи данных с временным мультиплексированием. В многоканальном режиме каждое слово данных битового потока соответствует отдельному каналу таким образом, что, например, блок из 24 слов данных содержит по одному слову для каждого из 24 каналов.

Последовательный порт может автоматически выбирать слова, соответствующие указанным каналам (из 32-х возможных каналов), применяя при необходимости схему компандирования или декомпандирования. Только при передаче данных в многоканальном режиме возможно нахождение вывода DTn в "третьем" состоянии в интервале времени, соответствующем неиспользуемому каналу. При приеме данных биты слова, соответствующие неиспользуемому каналу передачи, игнорируются.

Многоканальный режим разрешается установкой бита MCE в регистре SRCTLn. Многоканальный режим разрешается или запрещается одновременно и для приемника, и для передатчика последовательного порта.

Количество каналов, используемых в режиме многоканального обмена через последовательный порт, задается 5-битным полем NCH в регистре управления портом SRCTLn. В NCH записывается значение, на 1 меньшее действительного количества каналов.

5-битное статусное поле CHNL в регистре STCTLn содержит номер канала, активного в настоящий момент времени при многоканальной передаче/приеме через SPORTn. При переходе к каждому следующем слову поле CHNL увеличивается на единицу по модулю NCH.

Разрешение и запрещение каждого канала для приема и передачи выполняется записью 0 или 1 в бит с номером, соответствующим номеру канала в 32-битовых регистрах MTCSn (каналы передачи) и MRCSn (каналы приема).

Режим компандирования в многоканальном режиме работы последовательного порта задается для каждого канала отдельно путем установки или сброса соответствующего бита в регистрах MTCCSn и MRCCSn. Вид компандирования определяется старшим битом поля DTYPE в регистрах STCTLn и SRCTLn.

1.4.2. Синхронизация в многоканальном режиме

Все приемные и передающие устройства в многоканальной системе должны иметь единую синхронизацию. Для индикации начала блока в многоканальном режиме используется сигнал RFSn, который используется и приемником, и передатчиком последовательного порта как сигнал кадровой синхронизации. Таким образом, активное значение RFSn означает начало передачи/приема нового блока многоканальной последовательности.

Сигнал TFSn в данном режиме является признаком подтверждения передачи данных (активности канала) и указывает на то, управляет ли процессор ножкой DTn в данном цикле тактовой синхронизации порта SPORTn. На рис.3 показан пример работы последовательного порта в многоканальном режиме: передача ведется по каналам № 1 и 2, а прием – по каналам № 0 и 2. Сигнал RFS означает начало блока "кадров", сигнал TFS активен только для каналов, "включенных" на передачу, и может использоваться внешним устройством как индикатор передачи данных в интервале времени, соответствующем тому или иному каналу.

4-битное поле MFD в регистре управления STCTLn задает задержку в циклах тактовой синхронизации порта между приходом сигнала кадровой синхронизации и передачей/приемом первого бита в многоканальном режиме передачи. Нулевое значение MFD приводит к одновременному (в одном цикле тактовой синхронизации) обнаружению сигнала кадровой синхронизации RFSn и передаче/приему первого бита данных.

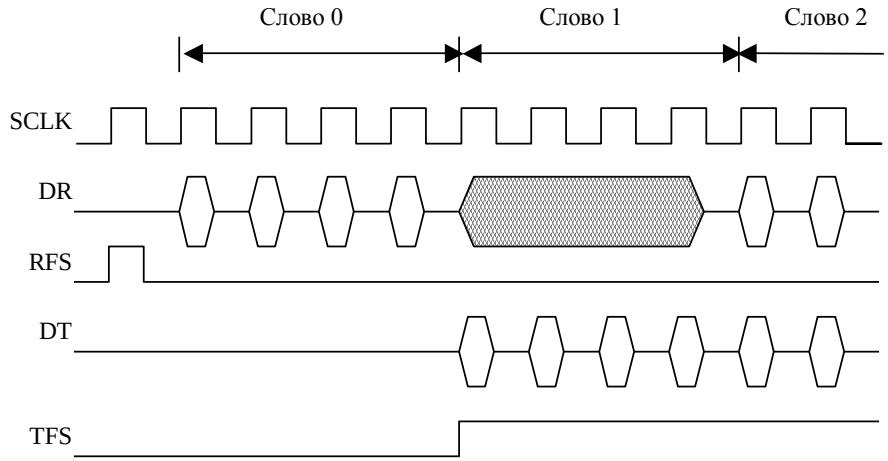

Рис.3

1.4.3. Регистры сравнения (только для ADSP-21061)

В многоканальном режиме имеется возможность маскирования принимаемых значений. Регистры KEYWDXn содержат шаблоны для сравнения с поступившими данными, а KEYMASKn – номера битов, подлежащих сравнению. Установка в KEYMASKn бита запрещает сравнение друг с другом соответствующих битов в принимаемом слове и в шаблоне KEYWDn. По результатам выполненного сравнения принятое слово либо отвергается, либо записывается в RXn-регистр с последующей генерацией прерывания или DMA-пересылкой во внутреннюю память. В каком случае принимать данные – когда сравнение дало результат "истина" или "ложь" – определяется значением бита IMAT в регистре управления портом SRCTLn.

Режим приема со сравнением включается установкой бита IMODE в регистре SRCTLn.

В режиме приема данных со сравнением не могут использоваться схемы компандирования и декомпандирования. В режиме приема данных со сравнением биты регистров MRCCSx используются не для указания необходимости декомпандирования для каждого канала, а для определения, необходимо ли выполнять сравнение с использованием KEYWDn и KEYMASKn для слова данных, полученного по данному каналу. Если соответствующий бит равен 0, то сравнение не выполняются и процессор принимает все поступающие слова.

Режим приема данных со сравнением позволяет освободить процессорное ядро от необходимых в некоторых случаях операций сравнения полученного значения с каким-либо шаблоном и осуществить ввод только тех значений, которые, например, больше нуля.

1.5. Обмен данными между последовательным портом и памятью

Обмен передаваемыми и принимаемыми данными между внутренней памятью ADSP-21060 и последовательными портами может осуществляться одним из двух способов: пословный обмен под управлением процессорного ядра или обмен блоками данных с использованием DMA-пересылок. В обоих способах управление обменом данными осуществляется обычно по сигналам прерывания: если DMA-пересылка для последовательного порта не включена, SPORT генерирует прерывание каждый раз после получения и при начале передачи очередного слова данных. Механизм DMA для последовательного порта реализует механизм генерации прерываний только после получения или передачи целого блока данных, что значительно сокращает вычислительные затраты и повышает эффективность использования процессорного ядра.

Как и любые другие прерывания, прерывания последовательного порта могут быть разрешены или замаскированы битами с 10 по 13 в регистре IMASK.

Биты в IRPTL и IMASK	Адрес обработчика	Имя прерывания	Функция
10	0x28	SPR0I	прерывание SPORT0 при приеме или DMA-пересылке по каналу № 0
11	0x2C	SPR1I	прерывание SPORT1 при приеме или DMA-пересылке по каналу № 1 (или прерывание линк-буфера 0)
12	0x30	SPT0I	прерывание SPORT0 при передаче или DMA-пересылке по каналу № 2
13	0x34	SPT1I	прерывание SPORT1 при передаче или DMA-пересылке по каналу № 3 (или прерывание линк-буфера 0)

1.5.1. Обмен данными с использованием процессорного ядра

При непосредственном обмене данными между регистрами процессорного ядра и буферами последовательных портов крайне неэффективно используются возможности ADSP-2106x, поскольку процессорное ядро большую часть времени проводит в ожидании, когда очередное слово будет передано или принято через SPORTn.

Прерывание по передаче/приему слова данных, генерируемое, если DMA-пересылки через последовательный порт запрещены, может быть использовано для реализации процедуры переноса значений между RXn- или TXn-регистрами и внутренней памятью на фоне выполняемой процессорным ядром задачи.

Ниже приведен фрагмент программного кода, иллюстрирующий такой способ организации передачи данных через последовательный порт.

```
    #define N 8
    #include "def21060.h"
    /*************************************************************
***/
    /*                          Сегмент данных во внутренней памяти
*/
    /*************************************************************
**/
    .segment/dm dm32_b1;
    .var  source[N]=  0x11111111,  0x22222222,  0x33333333,  0x44444444,
0x55555555, 0x66666666, 0x77777777, 0x88888888;
    .endseg;
    /*************************************************************
**/
    /*                                  Сегмент прерывания по сбросу
*/
    /*************************************************************
**/
    .segment/pm rst_svc;
       nop;
       jump start;          // Переход на начало обработки
    .endseg;
    /*************************************************************
**/
    /*           Вызов обработчика прерывания от SPORT0 при передаче
*/
    /*************************************************************
**/
    .segment/pm spt0_svc;
           jump s0tx;       // Чтобы обработчик мог быть >4 инстр.
    .endseg;
    /*************************************************************
**/
    /*  Программный код для инициализации последовательного порта и
*/
    /*  установки режима пересылки через SPORT0 под управлением ядра
*/
    /*************************************************************
**/
    .segment/pm pm48_1b0;           // Инициализация и запуск пересылки
    start:
       r0=0x00270007;       // В TDIV0-регистр: TCLKDIV=7,TFSDIV=39
       dm(TDIV0)=r0;        // что соответствует :
                                 // sclock=CLKIN/8, framerate=sclock/40
            r0=0x000064f1;  // В STCTL0-регистре: SPEN=1
                            // (передача через SPORT0)
       dm(STCTL0)=r0;       // SLEN=15 (16-битные слова);
       // ICLK=1 (внутренняя генерация такт.синхр.для передачи);
       // TFSR=1, (используется кадровая синхронизация для передачи);
       // ITFS=1, (внутренний сигнал кадровой синхронизации);
       // DITFS=0,(кадровая синхронизация независимая от данных);
       // все остальные биты равны 0
          b0=source;            //  Указатель  на  источник  данных
       (i0=b0)
       l0=@source;          // Длина буфера
          bit set imask SPT0I;     // Разрешить прерывания от регистра
       TX0
          bit set mode1 IRPTEN;    // Разрешить глобальные прерывания
          r0=dm(i0,1);             // Записать первое значение
                           // в TX0 для начала передачи
       dm(TX0)=r0;
    wait: idle;                    // Ожидать прерывания от TX0
       jump wait;
    /*************************************************************
***/
    /* Обработчик прерывания от буфера TX0. Прерывание генерируется ко-
гда */
    /*    в    TX0    освободилось    место    для    очередного    слова
*/
    /*************************************************************
***/
s0tx:
          rti (db);
          r0=dm(i0,1);             // взять очередное слово из буфера
          dm(TX0)=r0;              // и записать его в TX0-буфер
    .endseg;
```

Листинг 1.1.

1.5.2. Организация DMA-пересылок через последовательный порт

Каждый из DMA-каналов, работающих с последовательными портами, имеет свой бит разрешения SDEN в соответствующих регистрах управления портами STCTLx и SRCTLx. DMA-канал для последовательного порта конфигурируется путем загрузки необходимых значений в соответствующие регистры управления DMA-пересылкой.

Регистр	Разрядность	Функция
IIx	17	индексный регистр (адрес текущего элемента в буфере данных во внутренней памяти)
IMx	16	модификатор индекса (значение со знаком)
Cx	16	счетчик (уменьшается при пересылке каждого слова, а при достижении 0 вызывает генерацию прерывания DMA-пересылки)
CPx	18	указатель цепочки – адрес набора параметров для следующей DMA-пересылки (старший бит определяет, необходимо ли генерировать прерывание после завершения DMA-пересылки каждого блока данных)

GPx	17	регистр общего назначения или регистр для 2D DMA
DBx	16	регистр общего назначения или регистр для 2D DMA
DAx	16	регистр общего назначения или регистр для 2D DMA

После установки параметров DMA-пересылки, разрешения DMA-канала и получения слова данных в RX-буфер, автоматически будет сгенерирован запрос на использование IOD-шины и по ней полученное слово будет передано в буфер внутренней памяти. Аналогично, как только при передаче данных в TX-буфере освободится место для следующего слова, оно будет автоматически передано в него из внутренней памяти. Эти пересылки продолжаются до приема или передачи всего буфера данных.

При организации цепочечных DMA-пересылок после приема или передачи очередного буфера данных ADSP-21060 загружает в регистры управления DMA-пересылкой находящиеся в памяти по адресу, указанному в регистре CPx, параметры следующей DMA-пересылки и автоматически запускает ее. Режим разрешения цепочечных DMA-пересылок устанавливается битом SCHEN в регистрах управления портами STCTLn и SRCTLn.

```
    #define N 8
    #include "def21060.h"
    /*************************************************************
**/
    /*                              Сегмент данных во внутренней памяти
*/
    /*************************************************************
**/
    .segment/dm dm32_b1;             // Сегмент с данными
    .var  source[N]=  0x11111111,  0x22222222,  0x33333333,  0x44444444,
0x55555555, 0x66666666, 0x77777777, 0x88888888;
    .var destination[N];             // Приемник получаемых данных
    .endseg;
    /*************************************************************
***/
    /*                                   Сегмент прерывания по сбросу
*/
    /*************************************************************
***/
    .segment/pm rst_svc;
       nop;
       jump start;
    .endseg;
    /*************************************************************
***/
    /*              Вызов обработчика прерывания от SPORT0 при приеме
*/
    /*************************************************************
***/
    .segment/pm spr1_svc;
       jump s1rx;
    .endseg;

    /*************************************************************
***/
    /* Инициализация и запуск передачи и приема данных через SPORT1
*/
    /* Для пересылки данных из внутренней памяти в TX1 при передаче
*/
    /* используется DMA-канал № 3, для пересылки данных из RX1 во
*/
    /*    внутреннюю    память    при    приеме    -    DMA-канал    №    1
*/
    /*************************************************************
***/
    .segment/pm pm48_1b0;
    start:
       r0=source;
            dm(II3)=r0;              // Источник передаваемых данных
       r0=destination;
       dm(II1)=r0;           // Буфер для принимаемых данных
       r0=1;
       dm(IM3)=r0;           // Модификатор индекса для передачи
       dm(IM1)=r0;           // Аналогично для DMA-канала 1
       r0=@source;
       dm(C3)=r0;            // В счетчик - число слов
       dm(C1)=r0;            // Аналогично для приема
       r0=0x004421f1;        // В SRCTL1-регистре:
       dm(SRCTL1)=r0;        // SPEN=1 (прием через SPORT1);
       // SLEN=31 (32-битные слова);
          // RFSR=1 (использование кадровой синхронизации при приеме);
       // SDEN=1 (разрешить DMA-пересылку принимаемых в RX1 данных);
       // SPL=1 (зациклить порт DT->DR и TFS->RFS)
       r0=0x00270007;        // TDIV0-регистр: TCLKDIV=7, TFSDIV=39
       dm(TDIV1)=r0;         // sclock=CLKIN/8, framerate=sclock/40
       r0=0x000465f1;        // STCTL1-регистр:
       dm(STCTL1)=r0; // SPEN=1 (разрешить передачу через SPORT1);
       // SLEN=31 (32-битные слова);
       // ICLK=1 (внутренняя тактовая синхронизация при передаче);
       // TFSR=1 (режим передачи с кадровой синхронизацией);
       // ITFS=1 (внутренняя кадровая синхронизация);
       // DITFS=0 (независимая от данных кадровая синхронизация при
       // передаче);
       // SDEN=1 (разрешить DMA-пересылку при передачу через TX1;
       // остальные биты равны 0
       bit set imask SPR1I;  // Разрешить прерывание приема
                             // блока данных через SPORT1
       bit set mode1 IRPTEN; // разрешение прерываний
    wait: idle;
       jump wait;            // Ждать прерывания по завершению приема

    /*************************************************************
***/
    /* Обработчик прерывания по завершению приема данных через SPORT1
*/
    /*************************************************************
***/
    s1rx:
       rti;                  // Вызывается только один раз
    .endseg;
```

Листинг 1.2

Контрольные вопросы

1. Каковы назначение, структура и принцип функционирования последовательных портов?
2. Что такое тактовая и кадровая синхронизация?
3. Каковы особенности работы последовательных портов в многоканальном режиме?
4. Какими способами можно организовать обмен данными между последовательными портами и внутренней памятью?
5. Как можно организовать обмен данными между последовательными портами и внешней памятью?
6. В чем особенность режима приема со сравнением?
7. Какие форматы могут использоваться для передачи данных через последовательные порты?
8. Могут ли иметь место потери данных при передаче и приеме через последовательный порт?

Варианты заданий

Написать программный код для реализации согласованного с преподавателем алгоритма поблочной обработки сигнала. Ввод и вывод отсчетов сигнала – через последовательные порты. Для обмена данными между последовательными портами и памятью использовать цепочечные DMA-пересылки или пословную передачу под управлением процессорного ядра.

1. Входные данные - значения отсчетов, компандированные по А-закону. Выходные данные - значения отсчетов, компандированные по μ-закону.
2. Входные данные - компандированные по А-закону. Выходные данные - разности между текущим и предыдущим линейными значениями входных отсчетов.
3. Входные данные - разности между текущим и предыдущим линейными значениями отсчетов. Выходные данные - компандированные по А-закону значения отсчетов.
4. Входные данные - многоканальные компандированные значения отсчетов по каналам № 0, 12, 16. Выходные данные - линейные значения отсчетов по каналам № 2, 11, 17.
5. Реализовать прием только положительных значений, не выходящих за границы 8-разрядной сетки (для ADSP-21061). После окончания приема выдать полученные значения в обратном порядке.
6. Организовать DMA-передачу и прием данных с зацикливанием последовательного порта.

2. Работа с линк-портами SHARC-процессоров

SHARC-процессоры семейства ADSP-2106x и старше (кроме ADSP-21061) имеют дополнительные возможности организации обмена данными через шесть 4-битовых линк-портов (звеньевых портов). Каждый линк-порт может тактироваться частотой до двух раз превышающей внутреннюю частоту процессора, что позволяет передавать до 8 битов за один такт. Линк-порты широко используются для организации взаимодействия процессоров в многопроцессорных системах различной структуры.

Линк-порты характеризуются следующими возможностями:

- могут функционировать одновременно и независимо друг от друга;
- позволяют передавать/принимать 32- и 48-разрядные слова;
- доступны как для DMA-контроллера (DMA-пересылок), так и для ядра процессора;
- доступны для внешнего хост-процессора при использовании режимов прямого чтения и записи (Direct Reads and Direct Writes);
- имеют двухуровневые FIFO-буферы для передачи и приема данных;
- поддерживают обмен данными с использованием сигналов тактовой синхронизации и подтверждения.

2.1. Архитектура и принципы функционирования линк-портов и линк-буферов

Каждый линк-порт состоит из четырех двунаправленных линий данных $LxDAT_{3-0}$ и двух линий взаимодействия: тактирования (LxCLK) и подтверждения (LxACK). Линии LxCLK и LxACK позволяют осуществлять асинхронную передачу данных. Когда порт сконфигурирован на передачу, он управляет линиями данных и линией LxCLK. Если порт настроен на прием, он управляет только линией LxACK (рис. 4).

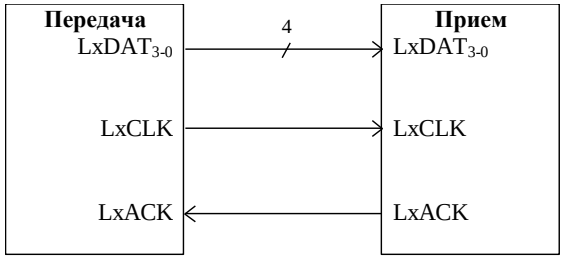

Рис.4

Посредством регистра LAR с каждым из шести линк-портов может быть связан один из шести линк-буферов (LBUF0-LBUF5)[3]. Линк-буфер включает в себя два регистра (рис. 5).

При передаче данных внутренний регистр используется для получения из внутренней памяти слова данных либо при выполнении DMA-пересылки, либо при непосредственной записи слова в буферный регистр инструкцией процессорного ядра. Внешний регистр выполняет передачу слов через линк-порт группами по 4 бита, начиная со старших битов. Таким образом, эти два регистра образуют двухуровневую FIFO-структуру. Когда передача слова завершена, выполняется сдвиг следующего слова из внутреннего регистра во внешний, устанавливается состояние "буфер неполон" в поле статуса линк-буфера LxSTAT в регистре LCOM и генерируется либо запрос на DMA-пересылку очередного слова (если DMA-пересылка для данного линк-буфера включена), либо маскируемое прерывание от линк-буфера. Если передающий регистр пуст (следующее слово не готово для передачи), то линк-порт снимает свой сигнал LxCLK и приостанавливает передачу данных.

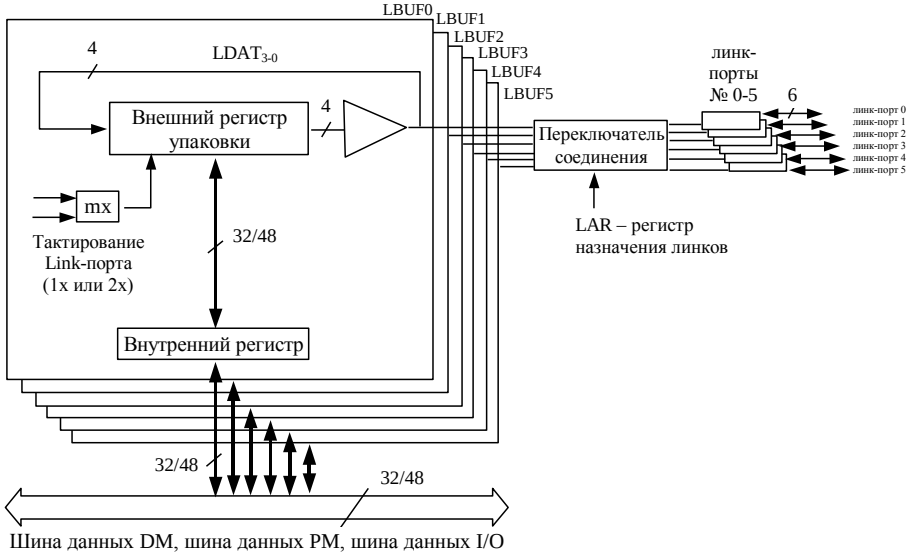

Рис. 5

Во время приема внешний регистр используется для упаковки получаемых данных в 32- или 48-разрядные слова, начиная со старшей 4-битовой группы (разрядность передаваемых/принимаемых слов определяется полем LEXT в регистре управления LCTL). По окончании формирования слово передается во внутренний регистр для передачи его во внутреннюю память посредством DMA-пересылки или чтения процессорным ядром. Если данные не были вовремя прочитаны и двухуровневый FIFO-буфер оказался заполненным, линк-порт снимает свой сигнал подтверждения приема LxACK приостанавливает прием данных.

Передаваемое/принимаемое линк-портом слово состоит из 8 или 12 полубайтов (соответственно для 32- и 48-разрядных слов). Передающий линк-порт устанавливает активный уровень сигнала тактовой синхронизации LxCLK при передаче каждого полубайта. Задний фронт сигнала LxCLK используется приемником для защелкивания полученного полубайта. Приемник выставляет активный уровень на линию LxACK, когда он готов для приема очередного слова (т.е. после приема каждых 8 или 12 полубайтов). Передающий порт проверяет состояние линии LxACK только в начале передачи каждого слова данных. Если в этот момент сигнал LxACK имеет неактивный уровень, передающий порт не начинает передачу следующего слова, а продолжает удерживать высокий уровень на линии LxCLK и первую 4-битовую группу данных на линии данных. После получения сигнала подтверждения LxACK процессор периодически чередует на линии LxCLK низкий и высокий уровень и выполняет передачу очередного слова. Если передающий буфер пуст, сигнал LxCLK имеет низкий уровень независимо от состояния линии LxACK.

Приемный буфер может заполниться вследствие того, что DMA-каналы, связанные с линк-буферами, могут иметь более низкий приоритет по сравнению, например, с DMA-каналами последовательных портов или операцией цепочечной загрузки DMA. Поэтому для предотвращения переполнения буфера приемный линк-порт может снять свой сигнал LxACK. Естественно, что как только линк-буфер получает право использования шины ввода/вывода, происходит пересылка принятого слова внутреннюю память (и, таким образом, освобождение места в приемном буфере), и приемник вновь выставляет высокий уровень на линию LxACK.

Следует обратить внимание на то, что сигнал LxACK влияет только на то, когда передающая сторона начнет передавать следующее слово. Полубайты текущего слова будут продолжать защелкиваться приемной стороной независимо от уровня сигнала LxACK.

Поскольку данные защелкиваются приемником по заднему фронту сигнала LxCLK, то операция приема является по сути своей асинхронной и может происходить на любой частоте вплоть до удвоенной частоты процессора. Если частота приема меньше тактовой частоты процессора CLKIN, то бит LCLKX2x в регистре управления LCOM для соответствующего линк-буфера должен быть сброшен. Если же частота приема лежит в интервале от CLKIN до 2*CLKIN, то бит LCLKX2x должен быть установлен. Это вызовет изменение поля состояния буфера (и генерацию соответствующего DMA-запроса на предоставление шины) не после 8-

[3] То есть, линк-буфер и линк-порт - это не одно и то же. Например, один и тот же линк-порт может быть назначен одновременно двум линк-буферам (такой режим называется "loopback" и используется при отладке программного кода).

го/12-го, а после приема 6-го/10-го полубайта. Переключение бита LCLKX2x во время передачи слова может привести к непредсказуемым последствиям.

Примечание. Когда линк-порт запрещен, линии $LxDAT_{3-0}$, LxCLK, LxACK находятся в неопределенном состоянии. Когда разрешена передача через порт, то линк-порт управляет линиями данных $LxDAT_{3-0}$, на линии LxCLK удерживается высокий уровень, а LxACK находится в третьем состоянии. Если порт разрешен для приема данных, то линии данных и LxCLK находятся в третьем состоянии, а на линии LxACK удерживается высокий уровень.

2.2. Регистры управления линк-портами

Для управления работой линк-портов используются три регистра: регистр управления линк-буферами (LCTL), общий регистр управления линками (LCOM) и регистр назначения линков (LAR). Для конфигурирования линк-порта значения в эти регистры должны быть записаны в следующем порядке: LAR, LCOM и LCTL. Перед переназначением линк-порта через регистр LAR соответствующий ему линк-буфер должен быть запрещен соответствующим битом в регистре LCTL.

Линк-порт является запрещенным, если он не имеет назначенного ему буфера или если назначенный ему буфер запрещен.

С помощью назначения одному линк-порту двух линк-буферов (режим "loopback") могут выполняться одновременно до трех пересылок типа "память-память". В таком режиме линк-порт не управляет линиями $LxDAT_{3-0}$, LxCLK и LxACK.

Регистр LAR

Биты	Поле	Функция
0-2	A0LB	номер назначенного линк-порта для линк-буфера № 0 (000-линк-порт № 0, 001-линк-порт № 1, 010-линк-порт № 2, 011-линк-порт № 3, 100-линк-порт № 4, 101-линк-порт № 5, 111-буфер неактивен)
3-5	A1LB	аналогично для линк-буфера № 1
6-8	A2LB	аналогично для линк-буфера № 2
9-11	A3LB	аналогично для линк-буфера № 3
12-14	A4LB	аналогично для линк-буфера № 4
15-17	A5LB	аналогично для линк-буфера № 5
18-31		зарезервировано

Регистр LCTL

Биты	ПОЛЕ	Функция
0-3	-	биты управления линк-буфером № 0: бит № 0 (L0EN) – разрешение линк-буфера. При сбросе этого бита очищаются биты LxSTAT и LPERR в регистре LCOM и связанный линк-порт прекращает прием данных (управление линией LxACK) или передачу данных (управление линией LxCLK) бит № 1 (L0DEN) – разрешение DMA-пересылок через линк-буфер № 0 бит № 2 (L0CHEN) – разрешение цепочечных DMA-пересылок через линк-буфер № 0 бит № 3 (L0TRAN) – направление работы линк-буфера: 1-передача, 0-прием
4-7	-	аналогичные биты управления линк-буфером № 1
8-11	-	аналогичные биты управления линк-буфером № 2
12-15	-	аналогичные биты управления линк-буфером № 3
16-19	-	аналогичные биты управления линк-буфером № 4
20-23	-	аналогичные биты управления линк-буфером № 5
24	LEXT0	разрядность слов, передаваемых/принимаемых через линк-буфер № 0: 1 - 48 бит, 0 - 32 бита.
25	LEXT1	разрядность слов, передаваемых/принимаемых через линк-буфер № 1: 1 - 48 бит, 0 - 32 бита
26	LEXT2	разрядность слов, передаваемых/принимаемых через линк-буфер № 2: 1 - 48 бит, 0 - 32 бита
27	LEXT3	разрядность слов, передаваемых/принимаемых через линк-буфер № 3: 1 - 48 бит, 0 - 32 бита
28	LEXT4	разрядность слов, передаваемых/принимаемых через линк-буфер № 4: 1 - 48 бит, 0 - 32 бита
29	LEXT5	разрядность слов, передаваемых/принимаемых через линк-буфер № 5: 1 - 48 бит, 0 - 32 бита
30-31		зарезервировано

Поле LEXTx имеет приоритет над полем длины слова внутренней памяти IMDW в регистре SYSCON. Если LEXTx=1, то обращение к памяти происходит в пространство памяти 48-рязрядных слов независимо от значения бита IMDW.

Регистр LCOM

Биты	Поле	Функция
0-1	L0STAT	состояние линк-буфера № 0: 11-полон, 00-пуст, 10-частично полон
2-3	L1STAT	состояние линк-буфера № 1: 11-полон, 00-пуст, 10-частично полон
4-5	L2STAT	состояние линк-буфера № 2: 11-полон, 00-пуст, 10-частично полон
6-7	L3STAT	состояние линк-буфера № 3: 11-полон, 00-пуст, 10-частично полон
8-9	L4STAT	состояние линк-буфера № 4: 11-полон, 00-пуст, 10-частично полон
10-11	L5STAT	состояние линк-буфера № 5: 11-полон, 00-пуст, 10-частично полон
12	LCLKX20	передача данных на удвоенной частоте через линк-буфер № 0
13	LCLKX21	передача данных на удвоенной частоте через линк-буфер № 1
14	LCLKX22	передача данных на удвоенной частоте через линк-буфер № 2
15	LCLKX23	передача данных на удвоенной частоте через линк-буфер № 3
16	LCLKX24	передача данных на удвоенной частоте через линк-буфер № 4
17	LCLKX25	передача данных на удвоенной частоте через линк-буфер № 5
18	L2DDMA	разрешение 2D-DMA (общее поле для всех портов). Линк-буферы № 4 и 5 на DMA-каналах № 6 и 7 не поддерживают режим двумерной DMA
19	LPDRD	запретить использование внутреннего согласующего резистора для линий LxCLK и LACK (общее поле для всех портов)
20	LMSP	=1 для разрешения режима mesh multiprocessing (общее поле для всех портов)
21-22	LPATHD	поле задержки при переключении LPATH в режиме mesh multiprocessing (общее поле для всех портов)
23-25	---	зарезервировано
26	LPERR0	состояние приема слова через линк-буфер № 0 (1 - незавершен, 0 - завершен)
27	LPERR1	состояние приема слова через линк-буфер № 1 (1 - незавершен, 0 - завершен)
28	LPERR2	состояние приема слова через линк-буфер № 2 (1 - незавершен, 0 - завершен)
29	LPERR3	состояние приема слова через линк-буфер № 3 (1 - незавершен, 0 - завершен)
30	LPERR4	состояние приема слова через линк-буфер № 4 (1 - незавершен, 0 - завершен)
31	LPERR5	состояние приема слова через линк-буфер № 5 (1 - незавершен, 0 - завершен)

2.3. Доступ к линк-буферам

2.3.1. Доступ процессорного ядра

Для ряда приложений, где задержки при DMA-доступе к внутренней памяти могут оказаться слишком велики, предпочтительным является непосредственный доступ процессорного ядра к линк-буферам как к отображаемым в память регистрам процессора ввода/вывода. При этом бит разрешения DMA-пересылок через данный линк-буфер LxDEN должен быть сброшен. Ниже приведен пример программного кода для пересылок данных через линк-порт под управлением процессорного ядра.

```
    #include "def21060.h"
    /*************************************************************
***/
    /*                                    Обработчик прерывания по сбросу
*/
    /*************************************************************
***/
    .segment/pm rst_svc;
       nop;                    // зарезервировано для загрузчика
       jump start;
    .endseg;
    /*************************************************************
*/
    /*                                  Инициализация и передача данных
*/
    /*************************************************************
*/
    .segment/pm pm48_1b0;
    start:
       r0=0x0003f03f;  // LAR-регистр: LBUF2->порт № 0, LBUF3->порт № 0
       dm(LAR)=r0;     // зациклить порт № 0
       r0=0x0000c000;  // LCOM-регистр:
       dm(LCOM)=r0;    // обмен на 2x- частоте
            r0=0x00009100;    //  LCTL-регистр:  32-битные  данные,  LBUF2-
       прием
       dm(LCTL)=r0;    // LBUF3-передача. Всегда писать LCTL после LAR
       r0=0x12345678;  // тестовое слово для передачи
       dm(LBUF3)=r0;   // записать в LBUF3 вручную (ядром)
       r1=dm(LBUF2);   // "висит" пока слово не получено
    wait:
       jump wait;
    .endseg;
```

Листинг 2.1

При попытке прочитать пустой буфер или записать в полный буфер процессорное ядро "зависнет" до окончания приема или передачи очередного слова. До зависания может быть передано 4 слова (двухуровневые FIFO на передающей и приемной стороне).

2.3.2. Доступ внешнего хост-процессора

Внешний хост-процессор может получить доступ к линк-буферам, используя операции прямого чтения и записи (Direct Reads, Direct Writes). В данном режиме разрядность слова данных для соответствующего линк-буфера определяется не его битом LEXTx, а выбирается в соответствии со значением бита HPM в регистре управления SYSCON.

2.3.3. DMA-пересылки через линк-порты

Линк-буферы № 0-5 позволяют организовать DMA-обмен с внутренней памятью посредством DMA-каналов № 1, 3, 4, 5, 6, 7 соответственно. Для разрешения DMA-пересылок через соответствующий линк-буфер необходимо установить регистры DMA-параметров связанного с ним DMA-канала и установить бит разрешения DMA-пересылок через данный линк-буфер в регистре LCTL.

DMA-канал для линк-порта конфигурируется путем загрузки необходимых значений в соответствующие регистры управления DMA-пересылкой.

Регистр	Разрядность	Функция
IIx	17	индексный регистр (адрес текущего элемента в буфере данных во внутренней памяти)
IMx	16	модификатор индекса (значение со знаком)
Cx	16	счетчик (уменьшается при пересылке каждого слова, а при достижении 0 вызывает генерацию прерывания DMA-пересылки)
CPx	18	указатель цепочки – адрес набора параметров для следующей DMA-пересылки (старший бит определяет, необходимо ли генерировать прерывание после завершения DMA-пересылки каждого блока данных)
GPx	17	регистр общего назначения или регистр для 2D DMA
DBx	16	регистр общего назначения или регистр для 2D DMA
DAx	16	регистр общего назначения или регистр для 2D DMA

В режиме цепочечной DMA (бит LxCHEN равен 1) ADSP-2106x автоматически инициализирует и запускает следующую DMA-пересылку по окончании текущей DMA-пересылки.

```
    #define N 8
    #include "def21060.h"

    /*************************************************************
***/
    /*                              Сегмент данных во внутренней памяти
*/
    /*************************************************************
***/
    .segment/dm dm32_b1;
    .var source[N] = 0x11111111, 0x22222222, 0x33333333, 0x44444444,
0x55555555, 0x66666666, 0x77777777, 0x88888888;
    .var destination[N];
    .endseg;
    /*************************************************************
**/
    /*                                     Сегмент прерывания по сбросу
*/
    /*************************************************************
**/
    .segment/pm rst_svc;
       nop;
       jump start;
    .endseg;
    /*************************************************************
**/
    /*                                   Обработчик прерывания от LBUF2
*/
    /*************************************************************
**/
    .segment/pm lp2_svc;
       jump lp2rx;
    .endseg;
    /*************************************************************
**/
    /*                       Установка параметров и запуск DMA-пересылки
*/
    /*************************************************************
**/
    .segment/pm pm48_1b0;
    start: r0=source;
       dm(II5)=r0;            // установить DMA-адрес источника
       r0=destination;
       dm(II4)=r0;            // установить DMA-адрес приемника
       r0=1;
       dm(IM5)=r0;            // модификаторы адреса =1
       dm(IM4)=r0;
       r0=@source;
       dm(C5)=r0;             // количество слов в DMA-пересылке
       dm(C4)=r0;
       r0=0x0000c000;         // LCOM-регистр: 2x-частота
       dm(LCOM)=r0;
       r0=0x0003f03f; // LAR-регистр: LBUF2->порт № 0, LBUF3->порт № 0
       dm(LAR)=r0;            // зациклить через линк-порт № 0
       r0=0x0000b300;         // LCTL-регистр: 32-битные слова
                              // LBUF2-прием, LBUF3=tx
       dm(LCTL)=r0;           // DMA через LBUF2 и LBUF3
              /* DMA-пересылка запущена */
```

```
      bit set imask LP2I;   // разрешить прерывание от линк-буфера № 2
      bit set mode1 IRPTEN; // разрешение прерываний
   wait:
      idle;                 // ждем окончания DMA-пересылки
      jump wait;
   /*****************************************************************
**/
   /*                      Необходимый обработчик прерывания от LBUF2
*/
   /*****************************************************************
**/
   lp2rx:
      ...
      rti;
   .endseg;
```

Листинг 2.2

2.4. Прерывания при работе с линк-портами

Как и любые другие прерывания, прерывания линк-портов могут быть разрешены или замаскированы установкой или сбросом соответствующих битов в регистре IMASK.

Биты в IRPTL и IMASK	Адрес обработчика	Имя прерывания	Функция
11	0x2C	SPR1I	прерывание линк-буфера 0 при DMA-пересылке по каналу № 1 (или прерывание SPORT1 при приеме)
13	0x34	SPT1I	прерывание линк-буфера 1 при DMA-пересылке по каналу № 3 (или прерывание SPORT1 при передаче)
14	0x38	LP2I	прерывание линк-буфера 2 при DMA-пересылке по каналу № 4
15	0x3C	LP3I	прерывание линк-буфера 3 при DMA-пересылке по каналу № 5
16	0x40	EP0I	прерывание линк-буфера 4 при DMA-пересылке по каналу № 6 (или прерывание буфера внешнего порта EPB0)
17	0x44	EP1I	прерывание линк-буфера 5 при DMA-пересылке по каналу № 7 (или прерывание буфера внешнего порта EPB1)
20	0x50	LSRQ	запрос на обслуживание линк-порта

Линк-порты позволяют использовать три типа прерываний.

1. Прерывания, вызванные приемом или получением данных при доступе к линк-буферам процессорного ядра без использования DMA-пересылок. В этом режиме маскируемое прерывание генерируется, когда приемный буфер не пуст или когда передающий буфер неполон.

2. Прерывания, генерируемые по окончании DMA-пересылки. В этом режиме прерывание генерируется, когда регистр счетчика DMA сбрасывается в 0. Поскольку линк-буфер по окончании приема блока данных оказывается пустым, то передающая сторона имеет возможность записать еще одно или два дополнительных (управляющих) слова в линк-буфер. В случае использования такого протокола обмена процедура обработки прерывания по завершению DMA-пересылки на приемной стороне должна прочитать значения из линк-буфера и в зависимости от их значения выбрать дальнейший ход выполнения программы.

3. Запрос на обслуживание линк-порта позволяет запрещенному линк-порту сгенерировать прерывание при попытке доступа к нему внешним устройством. Это позволяет двум процессорам взаимодействовать между собой без строгого предварительного согласования порядка и направления передачи данных через линк-порты.

Запрос на обслуживание линк-порта (прерывание LSRQ) генерируется в случае, когда линк-порт запрещен, но на линиях LxACK или LxCLK внешним устройством установлен активный уровень. При этом активный уровень сигнала LxACK (при запрещенном порте, т.е. LxEN=0) сигнализирует о запросе на передачу данных из запрещенного порта, а активный уровень сигнала LxCLK – о запросе на прием данных в запрещенный порт.

Для определения, какой из линк-портов требует выполнения процедуры обработки, используется регистр LSRQ.

Регистр LSRQ

Биты	Поле	Функция
0-3	---	зарезервировано
4	L0TM	маска передачи через линк-порт № 0
5	L0RM	маска приема через линк-порт № 0
6	L1TM	маска передачи через линк-порт № 1
7	L1RM	маска приема через линк-порт № 1
8	L2TM	маска передачи через линк-порт № 2
9	L2RM	маска приема через линк-порт № 2
10	L3TM	маска передачи через линк-порт № 3
11	L3R	маска приема через линк-порт № 3

	M	
12	L4TM	маска передачи через линк-порт № 4
13	L4RM	маска приема через линк-порт № 4
14	L5TM	маска передачи через линк-порт № 5
15	L5RM	маска приема через линк-порт № 5
16-19	---	зарезервировано
20	L0TRQ	состояние запроса на передачу через линк-порт № 0
21	L0RRQ	состояние запроса на прием через линк-порт № 0
22	L1TRQ	состояние запроса на передачу через линк-порт № 1
23	L1RRQ	состояние запроса на прием через линк-порт № 1
24	L2TRQ	состояние запроса на передачу через линк-порт № 2
25	L2RRQ	состояние запроса на прием через линк-порт № 2
26	L3TRQ	состояние запроса на передачу через линк-порт № 3
27	L3RRQ	состояние запроса на прием через линк-порт № 3
28	L4TRQ	состояние запроса на передачу через линк-порт № 4
29	L4RRQ	состояние запроса на прием через линк-порт № 4
30	L5TRQ	состояние запроса на передачу через линк-порт № 5
31	L5RRQ	состояние запроса на прием через линк-порт № 5

Запрос на обслуживание как для приема, так и для передачи данных от каждого линк-порта может быть замаскирован соответствующими битами LxTM и LxRM в регистре LAR. Все текущие состояния LxTRQ и LxRRQ объединяются по логическому "ИЛИ" для генерации маскируемого запроса на обслуживание линк-порта.

Использование прерывания LSRQ позволяет синхронизировать процессоры перед обменом данными, чтобы избежать потерь, которые могут произойти при попытке одновременной передачи данных двумя линк-портами друг другу. Ниже приведен пример программного кода, реализующий ожидание master-процессора (линк-порт № 0, линк-буфер № 3) до момента получения запроса на передачу от линка slave-процессора (линк-порт №0, линк-буфер № 2) и последующую передачу слова от master-процессора в slave-процессор.

```
    /*************************************************************
***/
    /*          фрагмент кода, загружаемый в master-процессор
*/
    /*************************************************************
***/
    ...
          /* ожидать от slave запроса на передачу ему маркера */
    bit clr imask LSRQI;     // запретить прерывание через LSRQ
    r0=0x10;
    dm(LSRQ)=r0;      // отслеживать запрос на передачу через порт 0
    r0=0x00000000;
    dm(LCTL)=r0;             // запретить все LBUF
       /* дождаться, пока slave перестанет выставлять запрос */
    disabled:
    r0=dm(LSRQ);             // проверить запрос по линии LxACK
        r0=FEXT r0 BY 20:1;  // на передачу через линк-порт 0
    r0=pass r0;
    if NE jump disabled;
       /* выход из цикла ожидания когда slave "замолчал"         */
       /* теперь активный уровень на линии LxACK будет означать, */
       /* что slave ждет от master передачи слова данных         */
    r0=0x00000000;
    dm(LCTL)=r0;             // запретить все LBUF
    bit set imask LSRQI;     // разрешить прерывание по LSRQ
    r0=0x10;
    dm(LSRQ)=r0;      // разрешить запрос на передачу через порт 0
    rti;              // выход из прерывания. Сразу обработка LSRQ
    ...
       /* обработчик прерывания LSRQ по запросу на передачу */
    lsrq_service:
    r0=0x00009000;           // LCTL-регистр:32-битные слова,
                             // LBUF3-передача
    dm(LCTL)=r0;             // запретить DMA через LBUF3
    r0=0x12345678;           // записать ключевое слово 0x12345678
    dm(LBUF3)=r0;            // послать слово в slave
    token_read:
    r1=0x40;                 // проверить, прочитал ли slave слово
    r0=dm(LCOM);             // путем проверки полноты буфера
    r0=r0 AND r1;            // пока буфер полон -
    if NE jump token_read;   // слово не прочитано - ждать
       /* слово прочитано – далее по алгоритму */
    ...

    /*************************************************************
*/
    /*                фрагмент  кода,  загружаемый  в  slave-процессор
*/
    /*************************************************************
*/
    ...
    bit clr imask LSRQI;     // запретить прерывания по LSRQ
    r0=0x10;                 // отслеживать запрос на передачу
    dm(LSRQ)=r0;             // через линк-порт № 0
    r0=0x00000000;
```

```
dm(LCTL)=r0;              // LCTL-регистр: запретить все LBUF
   /* дождаться, пока master перестанет выставлять запрос */
disabled2:
r0=dm(LSRQ);              // ждать сброса сигнала LxACK
r0=FEXT r0 BY 20:1;       // таким образом синхронизируются
r0=pass r0;               // master и slave процессоры
if NE jump disabled2;     // дождаться пассивного уровня LxACK
   /* теперь выставить активный уровень на линии LxACK */
   /* чтобы master передал в slave слово данных        */
r0=0x3fe3f;
dm(LAR)=r0;               // LAR-регистр: LBUF2->линк-порт № 0
r0=0x00000100;            // все остальные – неактивные
dm(LCTL)=r0;              // Прием через LBUF2 (slave) без DMA
   /* генерируется запрос к мастеру на передачу по LSRQ */
bit clr mode1 NESTM;      // запретить вложенные прерывания
r0=dm(LBUF2);             // прочитать слово от master
   /* далее – по алгоритму */
...
```

Листинг 2.3

2.5. Принципы взаимодействия через линк-порты в многопроцессорной системе

2.5.1. Способы распознавания ошибок при передаче

Ошибки при передаче через линк-порты могут быть обнаружены путем чтения бита LPERRx в регистре LCOM. Бит LPERRx отражает состояние счетчика полубайтов и сбрасывается в 0, когда последний достигает значения нуля (т.е. очередные 8 или 12 полубайт переданы/получены). Если значение LPERRx не равно нулю после окончания приема данных, то при передаче данных произошла ошибка. В то же время процедура обнаружения ошибок передачи обычно запускается только по окончании DMA-пересылки, а счетчик DMA-пересылок всегда гарантирует точное количество полученных слов. Поэтому для обнаружения ошибок при передаче данных передающая и приемная стороны могут придерживаться следующего протокола:

- передающая сторона посылает одно дополнительное слово в конце передачи блока данных. После этого передатчик меняет направление пересылки через свой линк-порт на "прием" и ждет соответствующего сообщения от приемника;
- приемная сторона по окончании получения блока данных (вызвана процедура обработки прерывания для соответствующего DMA-канала) проверяет, получено ли дополнительное слоо в линк-буфере (статус линк-буфера) , и затем читает значение бита LPERRx. После этого приемник посылает соответствующее сообщение передатчику через тот же самый или другой линк-порт.

2.5.2. Протокол передачи маркера (token passing)

При необходимости передать данные через линк-порты двух процессоров необходимо знать, какой из них в настоящий момент является мастером (master), т.е. передатчиком, а какой – подчиненным (slave), т.е. приемником, чтобы избежать попытки начать передачу данных одновременно с двух сторон. Для этой цели может использоваться протокол передачи маркера, диаграмма которого приведена на рис. 6.

Маркер представляет собой программный флаг, сходный с семафором, который передается между процессорами. После сброса (по умолчанию) маркер соответствует линк-порту одного из процессоров, делая его мастером и, соответственно, передатчиком. Когда линк-порт slave-процессора хочет стать мастером, он выставляет активный уровень на свою линию LxACK, чтобы "привлечь внимание" мастера. Мастер в соответствии с принятым протоколом интерпретирует этот сигнал либо как запрос на передачу данных, либо как запрос на передачу маркера.

Если мастер согласен отдать маркер, он посылает slave-процессору определенное "слово освобождения маркера" (token release word, TRW) и сбрасывает свой флаг маркера. Приемник проверяет полученное слово и если это слово является TRW, то устанавливает свой флаг маркера, становится мастером и может передавать данные. Если же полученное слово не является словом освобождения маркера, то приемник ожидает начала поступления новых данных от мастера.

В соответствии с протоколом мастер сам может инициировать передачу маркера путем посылки ключевого слова TRW, не дожидаясь сигнала запроса (LxACK) от приемника.

В приложении приведен программный код, загружаемый в оба процессора для реализации одного из вариантов протокола передачи маркера. Процессор с ID1 является по умолчанию мастером (передатчиком), а с ID2 – приемником (slave). Мастер передает с использованием DMA буфер данных через линк-порт № 0 и линк-буфер LBUF3, а slave-процессор принимает в режиме DMA буфер данных через свои линк-порт № 0 и линк-буфер LBUF2. После этого приемник запрашивает у мастера маркер путем генерации LSRQ-прерывания, поступающего в запрещенный к тому моменту времени линк-порт № 0 передатчика. В ответ на этот запрос мастер посылает слово освобождения маркера TRW и ожидает, будет ли оно принято slave-процессором. В свою очередь приемник проверяет, является ли полученное слово словом освобождения процессора и подтверждает получение маркера очисткой линк-буфера мастера за определенное заранее число тактов. Если маркер принят, то slave-процессор становится мастером (передатчиком) и передает свой буфер данных новому приемнику (slave). В противном случае (маркер не принят) мастер начинает передачу второго буфера данных. По завершении обмена данными master-процессор настраивает линк-буфер LBUF2 на прием без использования DMA, а slave-процессор устанавливает свой линк-буфер LBUF3 на передачу без использования DMA.

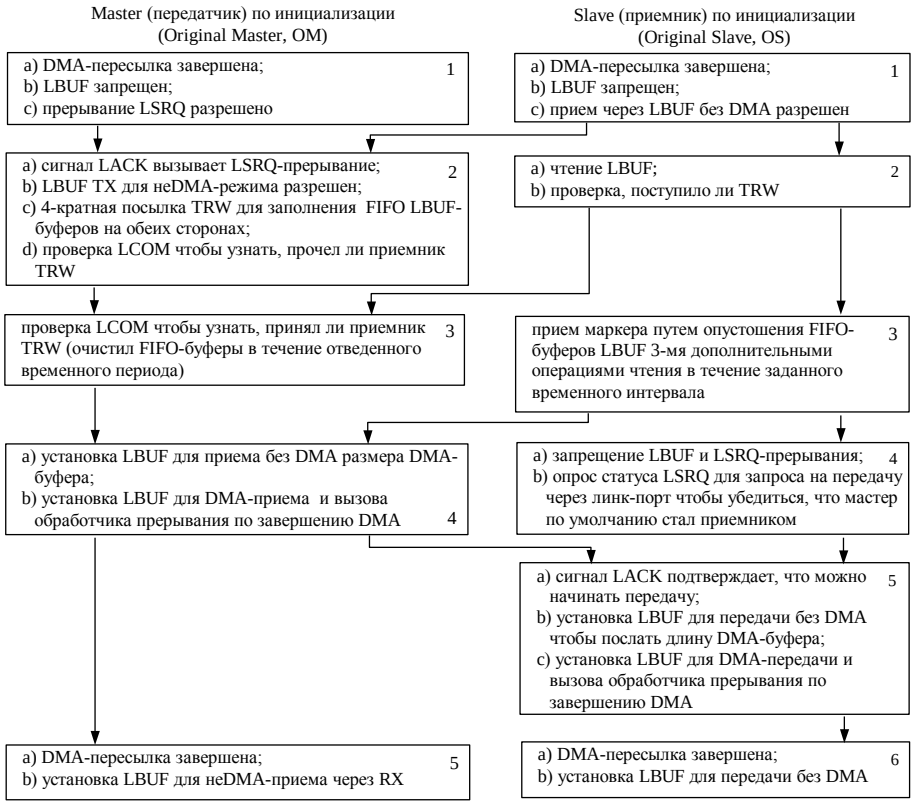

Рис. 6

При реализации протокола обмена данными с передачей маркера следует обратить внимание на следующее:

- буферы обоих линк-портов не должны быть одновременно разрешены на передачу. В противном случае данные могут быть переданы и потеряны вследствие того, что ни один из линк-портов не будет выставлять сигнал подтверждения LxACK. Поэтому в приведенном выше примере сначала исходный master-процессор становится slave-процессором и только после этого первоначальный slave-процессор становится master-процессором, чтобы избежать возможности одновременной передачи данных (для этого проверяются биты статуса в регистре LSRQ);

- временная синхронизация процессоров (в критической секции) не должна нарушаться. Для этого в приведенном примере запрещены вложенные прерывания.

Контрольные вопросы

1. В чем различие между линк-портом и линк-буфером?
2. Какие сигналы (линии) используются для обмена данными между процессорами посредством линков?
3. Какие управляющие регистры используются при работе с линк-портами?
4. Какие существуют режимы приема и передачи данных через линк-порт?
5. Как отслеживаются ошибки при передаче через линк-порты?
6. Когда необходимо использование протокола передачи маркера?

Варианты заданий

1. Реализовать обмен буферами данными между двумя процессорами через линк-порты с использованием DMA-пересылок, причем направления передачи должны чередоваться. Алгоритм обработки данных выбрать по согласованию с преподавателем.
2. Написать фрагменты программного кода для реализации обмена данными в многопроцессорной системе с линейной конфигурацией.
3. Написать фрагменты программного кода для реализации обмена данными в многопроцессорной системе с 2D-конфигурацией.
4. Написать фрагменты программного кода для реализации обмена данными в многопроцессорной системе с 3D-конфигурацией.

Литература

1. ADSP-21020/21010 User's Manual. Second Edition. - Norwood: Analog Devices Inc., 1994. - 396 p.
2. ADSP-21060 SHARC Preliminary User's Manual. Second Edition 3/94. - Norwood: Analog Devices Inc., 1994. - 186 p.
3. ADSP-21000 Family Assembler Tools & Simulator Manual. Second Edition. - Norwood: Analog Devices Inc., 1994. - 236 p.
4. ADSP-21000 Family Application Handbook Volume 1. First Edition. - Norwood: Analog Devices Inc., 1994. - 352 p.
5. Хусаинов Н.Ш., Калачев Д.П. Руководство к циклу лабораторных работ по курсу "Программирование сигнальных процессоров". – Таганрог: Изд-во ТРТУ, 2000. – 56с.
6. Хусаинов Н.Ш. Руководство к лабораторным работам по курсу "Архитектура и программирование сигнальных процессоров" "Программирование DMA-контроллера процессоров семейства ADSP-2106x". – Таганрог: Изд-во ТРТУ.

– 24с.

Содержание

Приложение

```
#include "def21060.h"
#define N 8                             /* размер буфера */
#define trw 0x0                  /* кодовое слово - маркер */
#define orig_master_id 1         /* ID процессора, который master по умолчанию */
#define orig_slave_id 2          /* ID процессора, который slave по умолчанию */

.SEGMENT/DM dm_data;
.var source_1[N]= 0x11111111, 0x22222222, 0x33333333, 0x44444444, 0x11111111, 0x22222222,
0x33333333, 0x44444444;
.var source_2[N]= 0x55555555, 0x66666666, 0x77777777, 0x88888888, 0x55555555, 0x66666666,
0x77777777, 0x88888888;
.var source_3[N]= 0x11111111, 0x22222222, 0x33333333, 0x44444444, 0x55555555, 0x66666666,
0x77777777, 0x88888888;
.var destination_1[N];
.var destination_2[N];
.var destination_3;
.ENDSEG;

.SEGMENT/PM isr_tabl;                        /* таблица обработчиков прерываний */
          NOP; NOP;NOP;NOP;             /* зарезервированные прерывания */
rst_svc:  nop; jump start; nop; nop;    /* прерывание по сбросу */
          NOP; NOP; NOP; NOP;
sovfi_svc:       RTI; RTI; RTI; RTI;
tmzhi_svc:       RTI; RTI; RTI; RTI;
vrpti_svc:       RTI; RTI; RTI; RTI;
irq2_svc: RTI; RTI; RTI; RTI;
irq1_svc: RTI; RTI; RTI; RTI;
irq0_svc: RTI; RTI; RTI; RTI;
          NOP; NOP; NOP; NOP;
spr0_svc: RTI; RTI; RTI; RTI;
spr1_svc: RTI; RTI; RTI; RTI;
spt0_svc: RTI; RTI; RTI; RTI;
spt1_svc: RTI; RTI; RTI; RTI;
lp2_svc:  nop; jump lp2; nop; nop;      /* прерывание от LBUF2 */
lp3_svc:  nop; jump lp3; nop; nop;      /* прерывание от LBUF3 */
ep0_svc:  RTI; RTI; RTI; RTI;
ep1_svc:  RTI; RTI; RTI; RTI;
ep2_svc:  RTI; RTI; RTI; RTI;
ep3_svc:  RTI; RTI; RTI; RTI;
lsrq_svc: nop; jump lsrq; nop; nop;     /* прерывание на обслуживание л
cb7_svc:  RTI; RTI; RTI; RTI;
cb15_svc: RTI; RTI; RTI; RTI;
tmzl_svc: RTI; RTI; RTI; RTI;
.ENDSEG;
/*------------------------------------------------------------------------
*/
/*                              инициализационный фрагмент
/*------------------------------------------------------------------------
*/
.SEGMENT/PM pm_code;
start:
   bit set mode2 FLG0O;          /* сконфигурировать Flag0 как выход */
   bit clr astat FLG0;           /* сбросить Flag0 в ASTAT чтобы был пр
         /* проверка, процессор – master или slave */
   r0=dm(SYSTAT);
   r0=FEXT r0 BY 8:3;            /* взять поле ID из регистра SYSTAT */
   r1=orig_master_id;
   r1=r0-r1;
   if eq jump start_as_master;   /* если master – на нужный код */
   r1=orig_slave_id;
   r1=r0-r1;
   if eq jump start_as_slave;    /* если slave – на нужный код */
   idle;                         /* ни то ни другое – ничего не делать
   nop;

/*------------------------------------------------------------------------
*/
/*                                запуск в режиме master
/*------------------------------------------------------------------------
*/
start_as_master:
   bit set astat FLG0;   /* установить Flag0 чтобы показать, что master
         /* установить параметры DMA-пересылки (передача из source_1)
   r0=source_1;
   dm(II5)=r0;
   r0=1;
   dm(IM5)=r0;
   r0=@source_1;
   dm(C5)=r0;
   r0=0xc000;                    /* LCOM-регистр: 2x-скорость через LBU
   dm(LCOM)=r0;
   r0=0x3f1ff;                   /* LAR-регистр: LBUF3->линк-порт № 0 *
   dm(LAR)=r0;
   bit set imask LP3I;           /* разрешить прерывания от LBUF3 */
   bit set mode1 IRPTEN;         /* глобальное разрешение прерываний */
   r0=0x0000b000;                /* LCTL-регистр: 32-битные слова, LBUF
   dm(LCTL)=r0;                  /* разрешить DMA через LBUF3 */
         /* DMA-пересылка запущена */
wait_1: idle;                          /* ждать окончания DMA-пересылк
   jump wait_1;                  /* вернуться сюда после любого прерыва
   nop;
   nop;

/*------------------------------------------------------------------------
-*/
/*                                                              обработчик
*/
```

```
   /*-------------------------------------------------------------------------------------------
-*/
   lp3:                                                                    3
      if NOT FLAG0_IN jump lp3_orig_slave;        /* проверка на master */ OM
      nop;                                        /* и соответствующий переход */
      nop;                                        /* если slave */
            /* если master - то завершение обработчика имеет вид */
   lp3_orig_master:
            /* синхронизироваться со slave-процессором */
      bit clr imask LSRQI;          /* запретить прерывание через LSRQ */
      r0=0x10;                      /* отслеживать запрос на передачу */
      dm(LSRQ)=r0;                  /* через линк-порт № 0 */
      r0=0x00000000;
      dm(LCTL)=r0;                  /* запретить все LBUF */
   disabled1:
      r0=dm(LSRQ);                  /* проверить LSRQ: есть ли запрос по линии LxACK*/
      r0=FEXT r0 BY 20:1;           /* (пассивное состояние). Синхронизация */
      r0=pass r0;                   /* master и slave - процессоров */
      if NE jump disabled1;         /* ждать пассивного уровня */
                                    /* для синхронизации master и slave */
      r0=0x00000000;
      dm(LCTL)=r0;                  /* запретить все LBUF */
      bit set imask LSRQI;          /* разрешить прерывание по LSRQ */
      r0=0x10;
      dm(LSRQ)=r0;                  /* разрешить в LSRQ запрос на передачу через порт №r0=0x3fe3f;
      rti;                          /* теперь ждать от slave запроса на передачу маркера*/ dm(LAR)=r0;

            /* если slave - то завершение обработчика имеет вид */
   lp3_orig_slave:
            /* в конце установить режим передачи без DMA */
      bit clr imask LP3I;           /* запретить прерывание от LBUF3 */
      r0=0x3f1ff;
      dm(LAR)=r0;                   /* LAR-регистр: LBUF3->линк-порт № 0 */
      r0=0x00009000;
      dm(LCTL)=r0;                  /* разрешить передачу через LBUF3 без DMA */
      rti;

   /*-------------------------------------------------------------------------------------------
*/                                                                         4
   /*                                                обработчик прерывания rti; LSRQ
*/
   /*-------------------------------------------------------------------------------------------
*/
   lsrq:
      bit clr imask LP3I;           /* запретить прерывание от LBUF3 */
      r0=0x00009000;                /* LCTL-регистр:32-битные слова, LBUF3-передача */
      dm(LCTL)=r0;                  /* без использования DMA */
      r0=trw;                       /* записать слово освобождения маркера в регистр*/
      dm(LBUF3)=r0;                 /* послать слово в slave */
      dm(LBUF3)=r0;                 /* заполнить FIFO-буферы линков в master и slave*/
      dm(LBUF3)=r0;                 /* для этого записать 4 раза (2 FIFO x 2 уровня)*/
      dm(LBUF3)=r0;
   token_read:
      r1=0x40;                      /* проверить, прочитал ли slave маркер */
      r0=dm(LCOM);                  /* путем проверки заполненности буфера */
      r0=r0 AND r1;                 /* пока буфер полон - */
      if NE jump token_read;        /* маркер не прочитан - ждать */
```

```
                                    /* дать slave время на ответ: принимает он маркер или нет */
      LCNTR=10, DO wait_for_slave UNTIL LCE;
   wait_for_slave: nop;
      r1=0xc0;                      /* проверить реакцию slave-процессора:
      r0=dm(LCOM);                  /* очистил ли он FIFO-буферы */
      r0=r0 AND r1;                 /* за время ожидания master-процессора
      if NE jump second_master_mode; /* если буфер не пуст - снова master
                                    /* иначе (slave очистил FIFO-буферы) - master становится sla
   slave_mode:
            /* чтобы избежать одновременной передачи через линки */
      bit clr imask LSRQI;          /* запретить LSRQ-прерывания */
      r0=0x10;
      dm(LSRQ)=r0;                  /* разрешить в LSRQ запрос на передачу
      r0=0x00000000;
      dm(LCTL)=r0;                  /* LCTL-регистр: запретить все LBUF */
            /* дождаться, пока бывший slave перестанет генерировать */
            /* запрос передачи от бывшего master через LSRQ         */
   slave_disabled:
      r0=dm(LSRQ);                  /* проверить, что slave сейчас запреще
      r0=FEXT r0 BY 20:1;           /* если slave запрещен, то он снял сиг
      r0=pass r0;                   /* и прекратил генерацию запроса LxTRQ
      if NE jump slave_disabled;    /* дождаться запрещения slave-процессо
            /* получить параметры DMA-пересылки от бывшего slave   */
            /* и запустить пересылку но уже на прием и через LBUF2 */
                                                                           OM
      r0=0x3fe3f;                                                          4
      dm(LAR)=r0;                   /* LAR-регистр: LBUF2->линк-порт № 0 *
      r0=0x00000100;
      dm(LCTL)=r0;                  /* LCTL-регистр: разрешить прием через
      r0=dm(LBUF2);                 /* прочитать кол-во DMA-слов */
      dm(C4)=r0;                    /* установить счетчик DMA-слов (длину
      r0=destination_3;
      dm(II4)=r0;                   /* установить адрес DMA-приемника */
      r0=1;                                                                OM 4b
      dm(IM4)=r0;                   /* размер шага */
      r0=0x00000300;
      dm(LCTL)=r0;                  /* разрешить DMA-прием через LBUF2 */
      bit clr irptl LP2I;           /* сбросить бит прерывания от LBUF2 в
      bit set imask LP2I;           /* разрешить прерывания от LBUF2 */
      bit set mode1 IRPTEN;         /* глобальное разрешение прерываний */
        /* повторная работа в режиме master (маркер не перешел) но теперь
        /* master идет через LBUF2, а slave должен принимать DMA-данные че
   second_master_mode:
   token_read2:
      r1=0xC0;                      /* ждать пока slave прочитает маркеры
      r0=dm(LCOM);                  /* путем проверки состояния FIFO-буфер
      r0=r0 AND r1;                 /* чтобы убедиться в синхронности рабо
      if NE jump token_read2;       /* для второй DMA-передачи */
      r0=0x3fe3f;
      dm(LAR)=r0;                   /* LAR-регистр: LBUF2->порт № 0 */
      r0=0x00000900;
      dm(LCTL)=r0;                  /* передача через LBUF2 без использова
      r0=@source_2;
      dm(LBUF2)=r0;                 /* передать в slave кол-во DMA-слов */
      r0=source_2;
      dm(II4)=r0;                   /* установить параметры DMA-передачи:
      r0=1;                         /* теперь передача - из source_2 */
      dm(IM4)=r0;                   /* модификатор = 1 */
      r0=@source_2;
```

OM1b

OM 1a

OM 2(a-c)

OM 3

OM 4a

OM 4b

OM 4c

```
   dm(C4)=r0;                  /* количество передаваемых слов */
   r0=0x00000b00;              /* LCTL-регистр: 32-битные данные, LBUF2-передача*/
   dm(LCTL)=r0;                /* разрешить DMA через LBUF2 */
          /* DMA-пересылка начата */
   bit clr irptl LP2I;         /* сбросить флаг последнего прерывания от LBUF2 */
   bit set imask LP2I;         /* разрешить прерывания от LBUF2 */
   bit set mode1 IRPTEN;       /* глобальное разрешение прерываний */
   rti;

/*-------------------------------------------------------------------------------
*/
/*                                                обработчик прерывания от LBUF2
*/
/*-------------------------------------------------------------------------------
*/
lp2:
   if NOT FLAG0_IN jump lp2_orig_slave;        /* проверить на master/slave */
   nop;                                        /* и перейти куда надо */
   nop;
          /* если master - то завершение обработчика имеет вид  */
lp2_orig_master:
          /* установить LBUF2 на прием без DMA */
   r0=0x3fe3f;
   dm(LAR)=r0;                 /* LAR-регистр: LBUF2->линк-порт № 0 */
   r0=0x00000100;
   dm(LCTL)=r0;                /* LBUF2: режим приема без DMA */
   rti;                        /* ждать прерывания */
          /* если slave - то завершение обработчика имеет вид  */
lp2_orig_slave:
          /* синхронизироваться с master-процессором */
   bit clr imask LSRQI;        /* запретить прерывания по LSRQ */
   r0=0x10;                    /* отслеживать запрос на передачу */
   dm(LSRQ)=r0;                /* через линк-порт № 0 */
   r0=0x00000000;
   dm(LCTL)=r0;                /* LCTL-регистр: запретить все LBUF */
disabled2:
   r0=dm(LSRQ);                /* ждать сброса сигнала по линии LxACK */
   r0=FEXT r0 BY 20:1;         /* таким образом выполняется синхронизация */
   r0=pass r0;                 /* master и slave процессоров */
   if NE jump disabled2;       /* дождаться пассивного уровня LxACK */
          /* теперь slave выставляет активный уровень на LxACK */
          /* чтобы показать master-процессору, чтобы он передал ему маркер */
   bit clr imask LP2I;         /* запретить прерывания от LBUF2
   r0=0x3fe3f;                 /* LAR-регистр: LBUF2->линк-порт № 0 */
   dm(LAR)=r0;                 /* все остальные - неактивные */
   r0=0x00000100;
   dm(LCTL)=r0;                /* Прием через LBUF2 без DMA */
   bit clr mode1 NESTM;        /* запретить вложенные прерывания */
          /* во избежание потери синхронизации */
   r0=dm(LBUF2);               /* прочитать, отдает ли master маркер */

      /* в следующем фрагменте кода slave проверяет, является ли первое после окончания */
      /* DMA-пересылки слово - TRW. Если нет, то slave очищает FIFO-буферы и продолжает */
      /* работать как slave. В противном случае slave решает, принимать ли ему маркер  */
   r1=trw;                     /* прочитать кодовое слово маркера */
   r0 = r0 - r1;               /* сравнить с тем, что получено в LBUF2 */
   if NE jump second_slave_mode; /* master не отдал маркер  */
   nop;
   nop;
          /* если следующие 3 строки закомментировать, то slave */
          /* станет master. В противном случае slave останется slave *
   LCNTR=20, DO reject_token UNTIL LCE;
   reject_token: nop;
   jump second_slave_mode;
   nop;                                /* задержка при чтении входного сообще
   nop;
          /* переход slave в режим master */
master_mode:
   r0=dm(LBUF2);                       /* читать 3 раза для очистки своего FI
   r0=dm(LBUF2);                       /* и FIFO master-процессора. То есть m
   r0=dm(LBUF2);                       /* передал маркер */
          /* чтобы не передавали одновременно два линк-порта */
   bit clr imask LSRQI;                /* запретить прерывания по LSRQ */
   r0=0x10;                            /* отслеживать только запрос на переда
   dm(LSRQ)=r0;                        /* через линк-порт № 0 в LSRQ*/
   r0=0x00000000;
   dm(LCTL)=r0;                        /* LCTL-регистр: запретить все LBUF */
disabled3:
   r0=dm(LSRQ);                        /* ждать сброса сигнала на линии LxACK
   r0=FEXT r0 BY 20:1;                 /* синхронизируя таким образом master
   r0=pass r0;
   if NE jump disabled3;               /* еще есть сигнал LxACK - ждать сброс
          /* следующий активный сигнал на LxACK будет означать, что ma
slave_enabled:
   r0=dm(LSRQ);                        /* проверить, что master стал slave-пр
   r0=FEXT r0 BY 20:1;                 /* путем ожидания активного уровня LxA
   r0=pass r0;                         /* по LxTRQ */
   if EQ jump slave_enabled;           /* сигнал еще не активен - ждать */
   r0=0x3f1ff;
   dm(LAR)=r0;                         /* LAR-регистр: LBUF3->линк-порт № 0 *
   r0=0x00009000;
   dm(LCTL)=r0;                        /* передача через LBUF3 без DMA */
   r0=@source_3;                       /* размер DMA-пересылки */
   dm(LBUF3)=r0;
   r0=0xc0;
wait: r1=dm(LCOM);                             /* проверить, прочитано ли пере
   r0=r0 AND r1;
   if NE jump wait;                    /* ждать пока не будет прочитано */
   nop;
   r0=source_3;
   dm(II5)=r0;                         /* установка параметров DMA на передач
   r0=1;                               */
   dm(IM5)=r0;                         /* модификатор = 1 */
   r0=@source_3;
   dm(C5)=r0;                          /* счетчик DMA = размеру буфера */
   r0=0x0000b000;                      /*LCTL-регистр: 32-битные данные, пере
   dm(LCTL)=r0;                        /* разрешить DMA через LBUF3 */
          /* началась DMA-пересылка */
   bit clr irptl LP3I;                 /* сбросить признак прерывания от LBUF
   bit set imask LP3I;                 /* разрешить прерывания от LBUF3 */
   rti;                                */ /* повторение работы в режиме slave */
second_slave_mode:
   r0=dm(LBUF2);                       /* освободить свой FIFO-буфер и буфер
   r0=dm(LBUF2);                       /* master-процессора */
   r0=dm(LBUF2);
   r0=0x3f1ff;
   dm(LAR)=r0;                         /* LAR-регистр: LBUF3->линк-порт № 0 *
```

OS 1b

OS 1c

OS 2a

OS 2b

OS 3

OS 4a

OS 4b

OS 5b

OS 5c

```
   r0=0x00001000;
   dm(LCTL)=r0;                  /* разрешить прием через LBUF3 без DMA */
   r1=dm(LBUF3);                 /* прочитать число слов в новой DMA-пересылке */
   r0=destination_2;
   dm(II5)=r0;                   /* установить параметры DMA для приема в destination */
   r0=1;
   dm(IM5)=r0;                   /* модификатор = 1 */
   r0=@destination_2;
   dm(C5)=r1;                    /* кол-во слов в DMA получено от мастера */
   r0=0x00003000;                /*LCTL-регистр:32-битные слова, прием через LBUF3*/
   dm(LCTL)=r0;                  /* разрешить DMA через LBUF3 */
          /* DMA-пересылка началась */
   bit clr irptl LP3I;           /* сбросить флаг прерывания от LBUF3 */
   bit set imask LP3I;           /* разрешить прерывания от LBUF3 */
   rti;
/*-------------------------------------------------------------------------------------------
*/
/*                                                              начало  работы  в  режиме  slave
*/
/*-------------------------------------------------------------------------------------------
*/
start_as_slave:
   r0=destination_1;
   dm(II4)=r0;                   /* параметры DMA - на прием через LBUF2 */
   r0=1;
   dm(IM4)=r0;                   /* модификатор = 1*/
   r0=@destination_1;            /* размер DMA-последовательности */
   dm(C4)=r0;
   r0=0xc000;                    /* LCOM-регистр: двойная скорость */
   dm(LCOM)=r0;
   r0=0x3fe3f;                   /* LAR-регистр: LBUF2->линк-порт № 0 */
   dm(LAR)=r0;
   bit set imask LP2I;           /* разрешить прерывания от LBUF2 */
   bit set mode1 IRPTEN;         /* глобальное разрешение прерываний */
   r0=0x00000300;                /*LCTL-регистр:32-битные слова, прием через LBUF2*/
   dm(LCTL)=r0;                  /* разрешить DMA через LBUF2 */
          /* DMA-пересылка началась */
wait_2: idle;                          /* ждать прерывания от LBUF2 */
   jump wait_2;                  /*вернуться сюда  после любого прерывания в slave*/
   nop;
   nop;
.ENDSEG;
```

OS 1a