6 ПАМЯТЬ

Процессор поддерживает иерархическую модель памяти с различными параметрами размера и производительности, зависящими от уровня блока памяти в иерархии. Блоки памяти первого уровня (L1) расположены внутри кристалла и работают быстрее системы памяти второго уровня (L2). Блоки памяти второго уровня (L2) расположены вне кристалла и имеют бóльшие задержки доступа. Более быстрые блоки памяти L1, к которым обычно относятся небольшая сверхоперативная память или блоки кэш-памяти, расположены внутри ядра.

Архитектура памяти

Процессор имеет унифицированный 4-гигабайтный диапазон адресов, покрывающий комбинацию внутренней и внешней памяти, и ресурсов ввода/вывода, отображенных в карте памяти. Часть адресного пространства памяти в этом диапазоне выделена под внутренние (внутрикристальные) ресурсы. Области внутреннего пространства памяти занимают:

- блоки статических ОЗУ (SRAM) L1;
- набор регистров, отображенных в карте памяти (MMRs);
- загрузочное ПЗУ (ROM);
- определяемое пользователем ПЗУ команд (только в процессорах ADSP-BF532 и ADSP-BF531).

Часть внутренней SRAM L1 также может быть сконфигурирована для работы в качестве кэша. Процессор также обеспечивает поддержку пространства внешней памяти, которое включает пространство асинхронной памяти и пространство синхронной DRAM (SDRAM). Подробное описание каждой из этих областей памяти и поддерживающих их контроллеров см. в главе 17 «Устройство интерфейса внешней шины».

На рис. 6-1, 6-2 и 6-3 представлен обзор карт памяти системы процессоров ADSP-BF533, ADSP-BF532 и ADSP-BF531, соответственно. Следует отметить, что в архитектуре процессора не выделяется отдельное пространство ввода/вывода. Все ресурсы отображены в общем 32-разрядном адресном пространстве. Память процессора имеет байтовую адресацию.

Как показано в таблице 6-1 в процессорах ADSP-BF533, ADSP-BF532 и ADSP-BF531 реализованы различные конфигурации памяти данных и команд.

Память

Таблица 6-1. Конфигурация памяти.

Тип памяти	ADSP-BF531	ADSP-BF532	ADSP-BF533
SRAM команд /кэш команд, запираемый по входам или строкам	16 Кбайт	16 Кбайт	16 Кбайт
SRAM команд	16 Кбайт	32 Кбайт	64 Кбайт
ПЗУ команд	32 Кбайт	32 Кбайт	–
Кэш/SRAM данных	16 Кбайт	32 Кбайт	32 Кбайт
SRAM данных	–	-	32 Кбайт
Сверхоперативная SRAM данных	4 Кбайт	4 Кбайт	4 Кбайт
Общий размер	84 Кбайт	116К байт	148К байт

Старшая часть внутреннего адресного пространства выделена под регистры ядра и системы, отображенные в карте памяти. Доступ к этой области возможен только, когда процессор находится в режиме Супервизора или Эмуляции (см. главу 3 «Рабочие режимы и состояния»).

В процессорах ADSP-BF532 и ADSP-BF531 имеется ПЗУ команд объёмом 32 Кбайта. Это ПЗУ определяется пользователем, и, так как оно является частью памяти L1, обращение к нему производится с тактовой частотой ядра (`CCLK`).

В процессорах, в которых отсутствует определяемое пользователем ПЗУ команд, младший 1 Кбайт пространства внутренней памяти занят загрузочным ПЗУ. В зависимости от выбранной опции загрузки при сбросе процессора из этого пространства памяти выполняется соответствующая загрузочная программа (см. раздел «Методы загрузки» главы 3).

В карте внешней памяти имеются четыре банка пространства асинхронной памяти и один банк памяти SDRAM. Каждый банк асинхронной памяти имеет размер 1 Мбайт, банк SDRAM может иметь размер до 128 Мбайт.

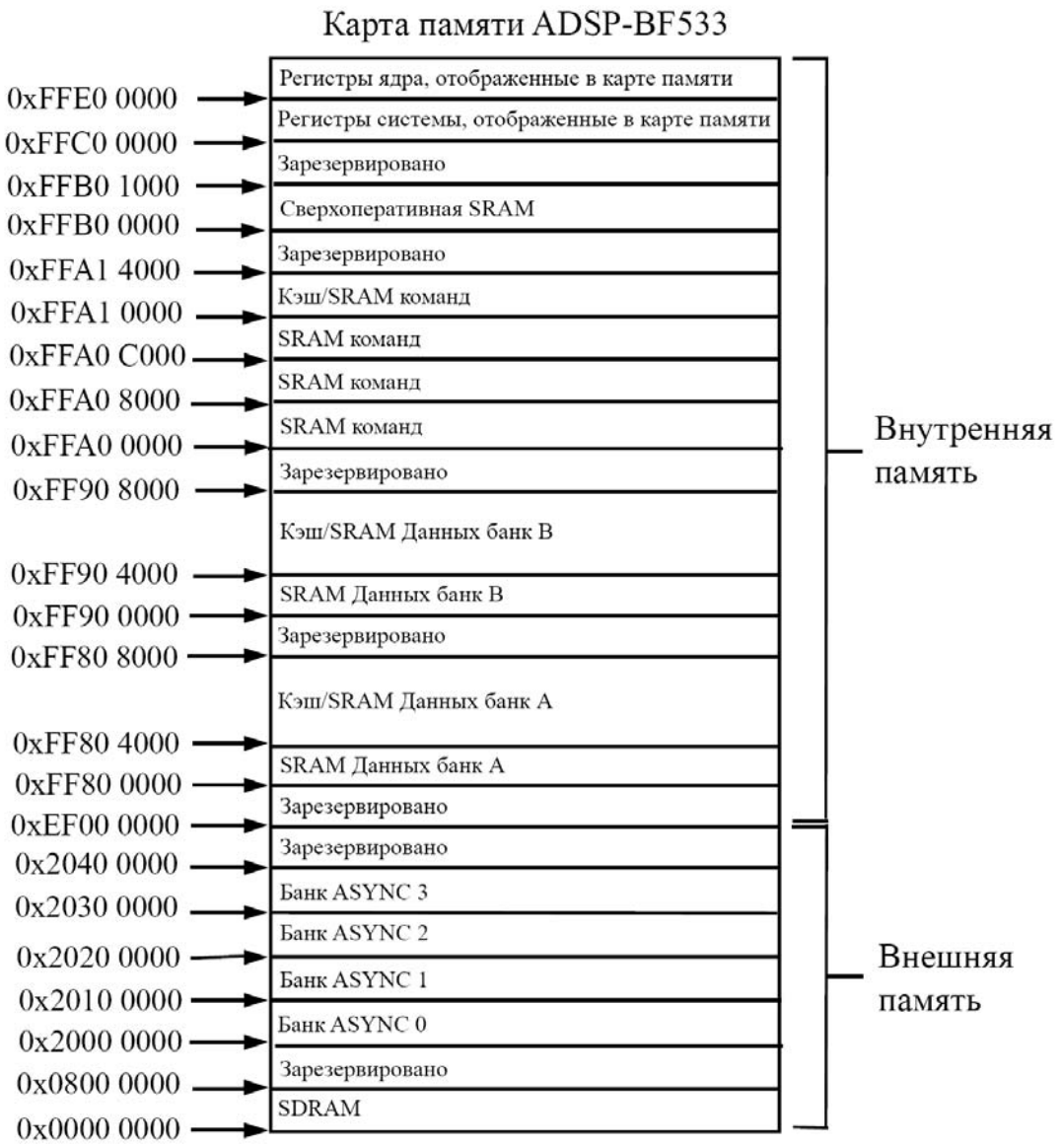


Рис. 6-1. Карта памяти ADSP-BF533

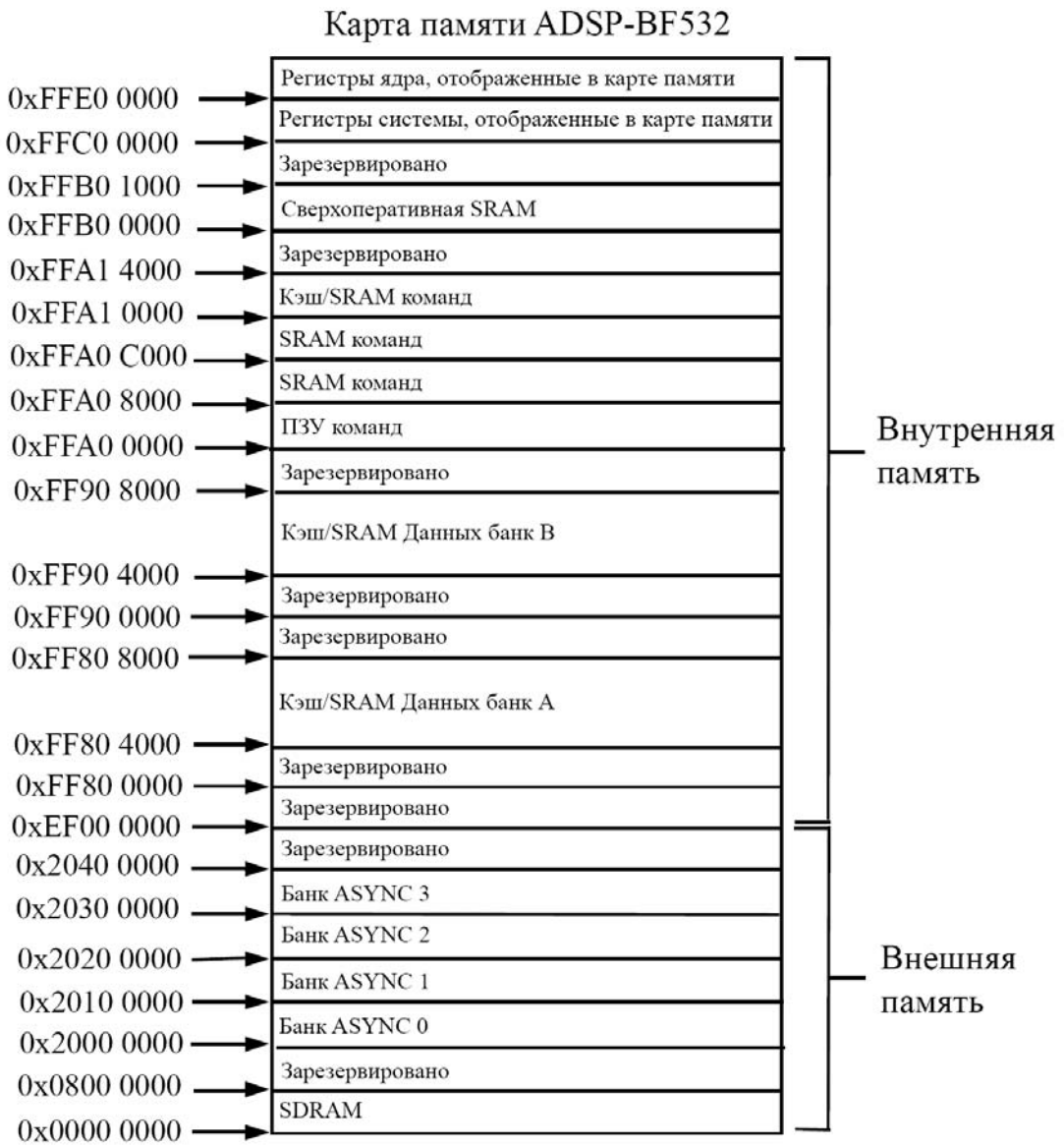


Рис. 6-2. Карта памяти ADSP-BF532.

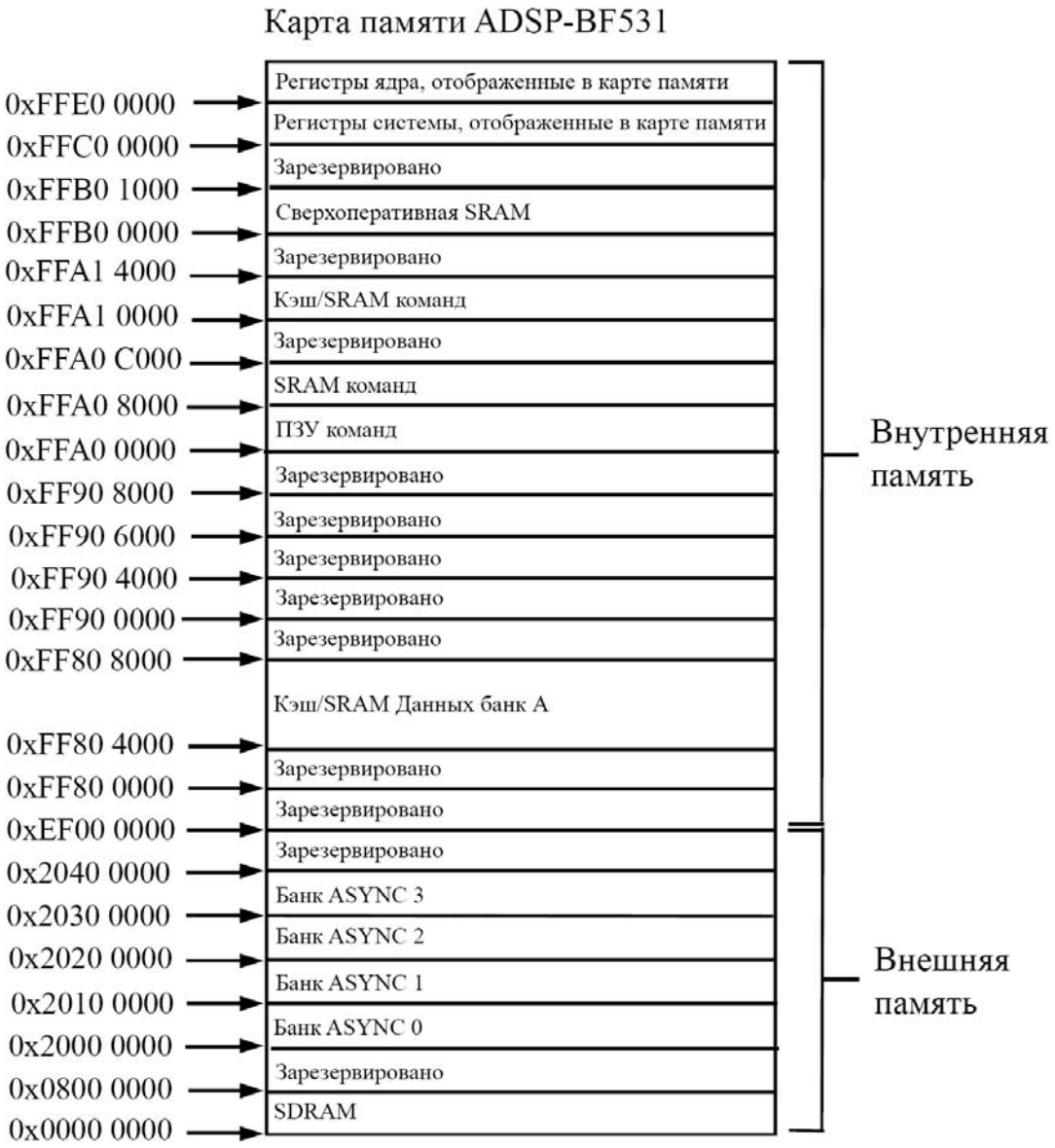


Рис. 6-3. Карта памяти ADSP-BF531.

Обзор внутренней памяти

Производительность системы памяти L1 обеспечивает высокую пропускную способность и низкую задержку. Так как память SRAM обеспечивает детерминированное время обращения и очень высокую пропускную способность, в системах ЦОС улучшение производительности обычно достигается включением быстрой SRAM в кристалл.

Добавление кэша данных и команд (SRAM с аппаратным управлением кэширования) одновременно обеспечивает высокую производительность и простую модель программирования. Применение кэша устраняет необходимость явного управления перемещением данных в память L1 и из нее. Возможны быстрая разработка кода и перенос кода с одного процессора на другой, не

требующая оптимизации производительности под конкретную организацию памяти.

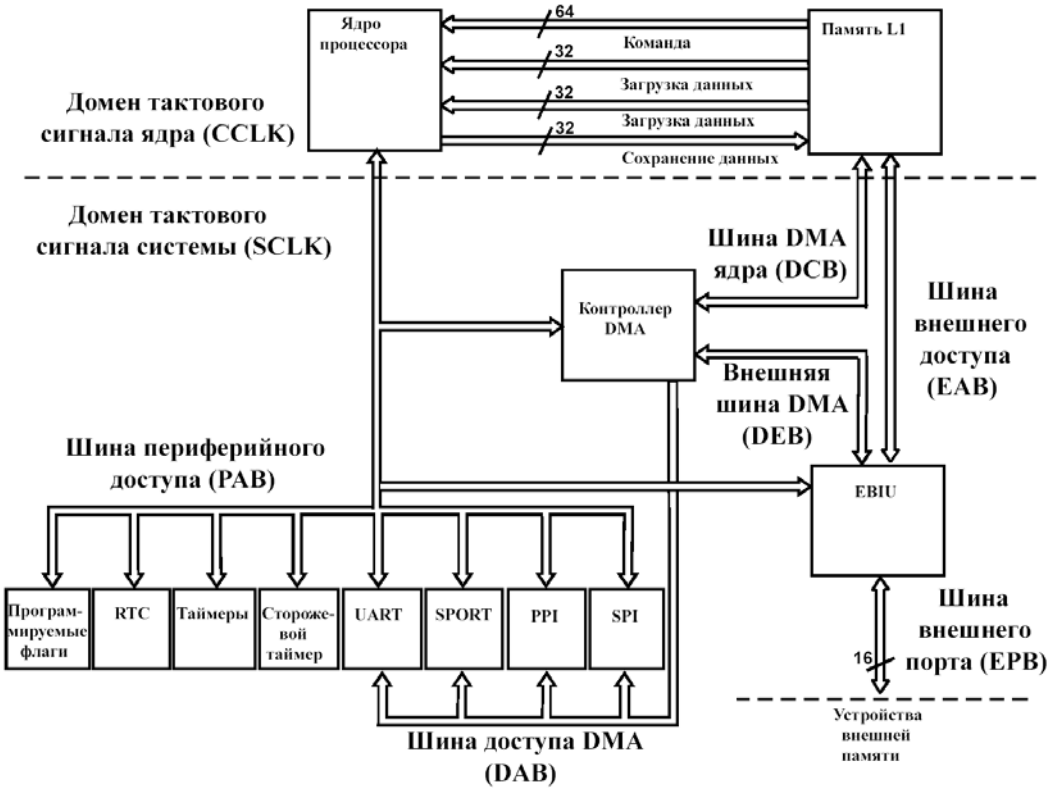


Рис. 6-4. Архитектура памяти процессора.

Память L1 имеет следующие особенности:

- Модифицированная гарвардская архитектура, позволяющая осуществлять до 4-х обращений к памяти ядра за один такт (одна 64-разрядная выборка команды, две 32-разрядных операции загрузки данных, одна 32-разрядная конвейерная операция сохранения данных).
- Одновременное выполнение DMA системы, поддержки кэша и обращений к памяти ядра.
- Доступ к SRAM с тактовой частотой процессора (`CCLK`) для реализации критичных алгоритмов ЦОС и быстрого переключения контекста.
- Опции кэша данных и кэша команд, характерные для кода микроконтроллеров, превосходная поддержка языков высокого уровня (HLL, High Level Language) и простота команд управления кэшем, таких как `PREFETCH` и `FLUSH`.
- Защита памяти.

Блоки памяти L1 работают с тактовой частотой ядра (`CCLK`).

Обзор сверхоперативной SRAM данных

В процессоре имеется выделенный банк сверхоперативной SRAM данных объемом 4 Кбайт. Сверхоперативная память не зависит от конфигурации других банков памяти L1 и не может конфигурироваться как кэш или использоваться в качестве объекта DMA. Обычно приложения используют сверхоперативную память данных при реализации операций, критичных ко времени выполнения. Например, для быстрого контекстного переключения при обработке прерывания пользовательский стек и стек супервизора следует размещать в сверхоперативной памяти.

Блоки памяти L1 работают с тактовой частотой ядра (CCLK).

Обращение контроллера DMA к сверхоперативной SRAM данных невозможно.

Память команд L1

Память команд L1 состоит из комбинации выделенной SRAM и банков, которые могут конфигурироваться как SRAM или как кэш. Биты управления в регистре `IMEM_CONTROL` могут использоваться для организации четырех подуровней банков памяти L1 в 16 Кбайтном банке, конфигурируемом как кэш или SRAM, следующим образом:

- Простая SRAM.
- 4-входовый наборно-ассоциативный кэш команд.
- Кэш с возможностью запирания до 4-х входов.

Память команд L1 может использоваться только для хранения команд.

Регистр управления памятью команд (IMEM_CONTROL)

Регистр управления памятью команд (`IMEM_CONTROL`) содержит биты управления памятью команд L1. По умолчанию после сброса кэш и проверка адресов в ассоциативном буфере защиты и кэширования (CPLB) запрещены (см. раздел «Кэш команд L1»).

Когда установлен (равен 1) бит `LRUPRIORST`, сбрасываются все биты состояний кэширования `CPLB_LRUPRIO` (см. раздел «Регистры данных ICPLB (ICPLB_DATAx)). При этом также все кэшируемые строки принудительно переводятся в состояние равной (низкой) значимости. Политика замещения кэша основывается, в первую очередь, на значимости строки, указываемой битами `CPLB_LRUPRIO`, а затем на алгоритме LRU (поиск строки, которая дольше всех не использовалась, least recently used). Подробности см. в разделе «Запирание кэша команд по строкам». Для разрешения записи в биты `CPLB_LRUPRIO` при кэшировании новых строк бит `LRUPRIORST` должен иметь значение 0.

Биты ILOC[3:0] содержат полезную информацию только после того, как код программно загружен в кэш. См. раздел «Запирание кэша команд по входам». Эти биты определяют, какие входы не участвуют в политике замещения кэша. Их использование приводит к эффекту запирания кэша по входам, не участвующим в процессе замещения. При этом код, содержащийся в них, может быть удален из кэша при помощи команды IFLUSH. Если бит в поле ILOC[3:0] равен 0, соответствующий вход не запирается и участвует в политике замещения кэша. Если бит в поле ILOC[3:0] равен 1, соответствующий вход запирается и не участвует в политике замещения кэша.

При помощи бита IMC часть SRAM команд L1 резервируется для работы в качестве кэша. Следует отметить, что резервирование памяти в качестве кэша само по себе не разрешает кэширование обращений к памяти L2. Для этого также необходимо разрешить использование CPLB при помощи бита EN_ICPLB, и требуемые страницы памяти должны определяться как доступные для кэширования дескрипторами CPLB (регистры ICPLB_DATAx и ICPLB_ADDRx).

По умолчанию после сброса использование CPLB команд запрещено. В этом состоянии интерфейс памяти L1 выполняет только минимальную проверку адреса. При минимальной проверке адреса исключение генерируется всякий раз, когда процессор пытается произвести выборку команды из:

- зарезервированного (несуществующего) пространства памяти команд L1;
- пространства памяти данных L1;
- пространства регистров, отображенных в карте памяти.

При обновлении дескрипторов CPLB (регистров DCPLB_DATAx и DCPLB_ADDRx) использование CPLB должно запрещаться битом EN_ICPLB. Следует отметить, что, так как применяется слабое упорядочивание операций загрузки/сохранения (см. раздел “Упорядочивание операций загрузки регистров и сохранения в память»), запрещение CPLB должно сопровождаться командой CSYNC.

При разрешении/запрещении кэша или CPLB для корректной работы следующей командой после записи в регистр IMEM_CONTROL должна быть команда CSYNC.

Для гарантии корректной работы и совместимости с будущими проектами при каждой записи в этот регистр в позиции зарезервированных битов должно записываться значение 0.

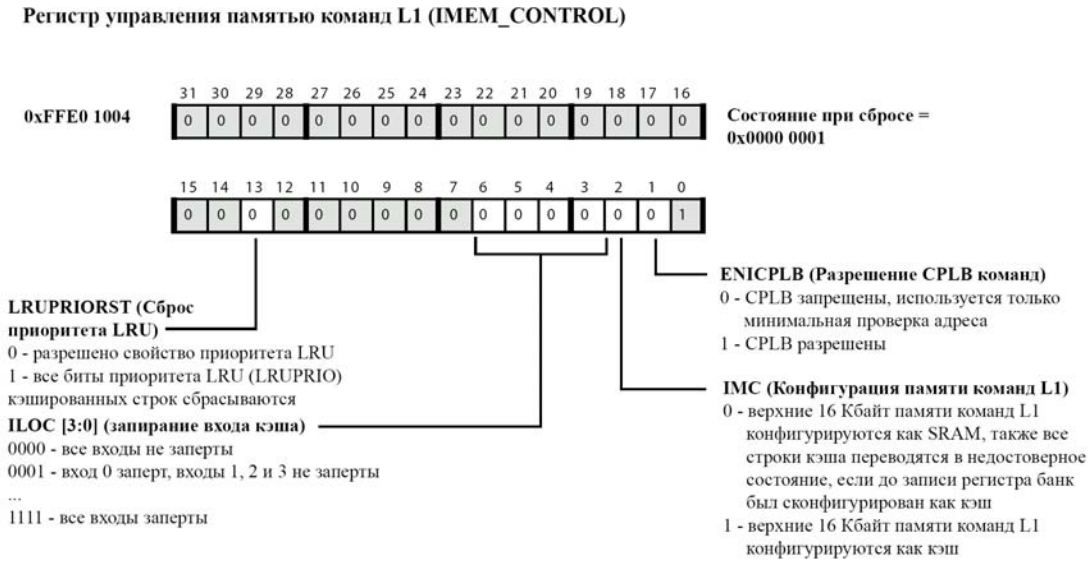


Рис. 6-5. Регистр управления памятью команд L1

SRAM команд L1

Ядро процессора производит чтение памяти команд по 64-разрядной шине выборки команд. Все адреса на этой шине выровнены по границе 64-х разрядов. В каждой выборке может возвращаться любая комбинация 16-, 32- или 64-разрядных команд (например, четыре 16-разрядные команды, две 16-разрядные и одна 32-разрядная команда или одна 64-разрядная команда).

Прямое обращение генераторов адреса данных, описываемых в главе 5, к памяти команд L1 невозможно. Обращение DAG к пространству памяти команд SRAM генерирует исключение (см. раздел «Исключения» в главе 4).

Доступы записи к памяти команд SRAM L1 должны выполняться с помощью 64-разрядного порта DMA системы. Так как SRAM реализована как набор однопортовых подуровней банка памяти, в действительности память команд работает как двухпортовая память.

В таблице 6-2 перечислены начальные адреса подуровней банков памяти команд L1.

Таблица 6-2. Подуровни банка памяти команд L1.

Подуровень банка памяти	Начальный адрес, ADSP-BF533	Начальный адрес, ADSP-BF532	Начальный адрес, ADSP-BF531
0	0xFFA0 0000	0xFFA0 8000	0xFFA0 8000
1	0xFFA0 1000	0xFFA0 9000	0xFFA0 9000
2	0xFFA0 2000	0xFFA0 A000	0xFFA0 A000
3	0xFFA0 3000	0xFFA0 B000	0xFFA0 B000
4	0xFFA0 4000	0xFFA0 C000	
5	0xFFA0 5000	0xFFA0 D000	
6	0xFFA0 6000	0xFFA0 E000	
7	0xFFA0 7000	0xFFA0 F000	
8	0xFFA0 8000		
9	0xFFA0 9000		
10	0xFFA0 A000		
11	0xFFA0 B000		
12	0xFFA0 C000		
13	0xFFA0 D000		
14	0xFFA0 E000		
15	0xFFA0 F000		

Архитектура банка памяти команд L1 представлена на рис. 6-6. Как показано на рисунке, каждый 16 Кбайтный банк состоит из четырех подуровней, объёмом по 4 Кбайта.

Кэш команд L1

Информацию о терминологии, касающейся кэша, см. в разделе «Терминология».

Память команд L1 также может быть сконфигурирована таким образом, чтобы содержать 4-входовый наборно-ассоциативный кэш объёмом 16 Кбайт. Каждый вход или строка кэша может независимо запираться, что позволяет улучшить среднее время задержки обращения для частей программы, критичных ко времени. Когда память сконфигурирована как кэш, прямое обращение к ней невозможно.

Когда кэш разрешен, возможно кэширование только тех страниц памяти, которые определяются как кэшируемые в CPLB. Когда разрешено использование CPLB, любой ячейке памяти, к которой выполняется обращение, должно соответствовать определение атрибутов в CPLB, в противном случае генерируется исключение CPLB. CPLB описывается в разделе «Свойства и защита памяти».

На рис. 6-6 показана общая организация кэша команд процессора Blackfin.

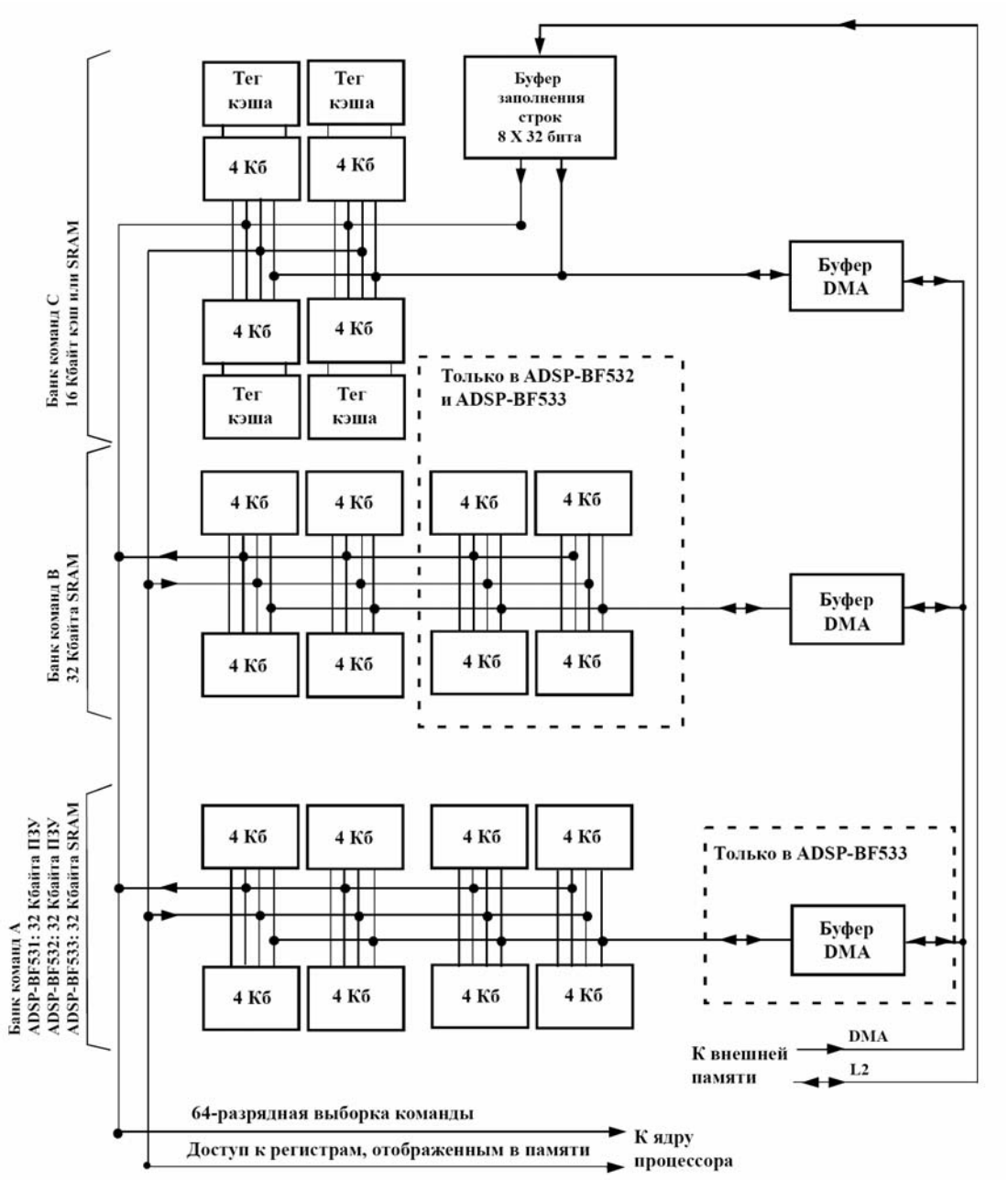


Рис. 6-6. Архитектура банка памяти команд L1

Строки кэша

Как показано на рис. 6-7, кэш состоит из набора строк. Каждая строка состоит из *компонента тега* и *компонента данных*.

- Компонент тега включает в себя 20-разрядный тег адреса, биты, указывающие состояние использования строки (LRU), бит достоверности и бит запирания строки.
- Компонент данных может содержать до четырех 64-разрядных слов данных команды.

Память

Компоненты тега и данных строк кэша хранятся в массивах памяти тегов и данных, соответственно.

Тег адреса состоит из старших 18 битов, а также битов 11 и 10 физического адреса. Биты 12 и 13 физического адреса не являются частью тега; они используются для идентификации 4 Кбайтного подуровня банка памяти, к которому осуществляется обращение.

Биты LRU являются частью алгоритма LRU, используемого для определения строки кэша, которую следует заменить при промахе в кэше.

Бит достоверности отображает состояние строки кэша. Строка кэша всегда является либо достоверной, либо недостоверной.

- Для недостоверных строк соответствующий бит достоверности сброшен. Это означает, что при операции сравнения адреса с тегом строка будет проигнорирована.
- Для достоверных строк соответствующий бит достоверности установлен. Это означает, что содержит достоверную команду/данные, совпадающие с источником в памяти.

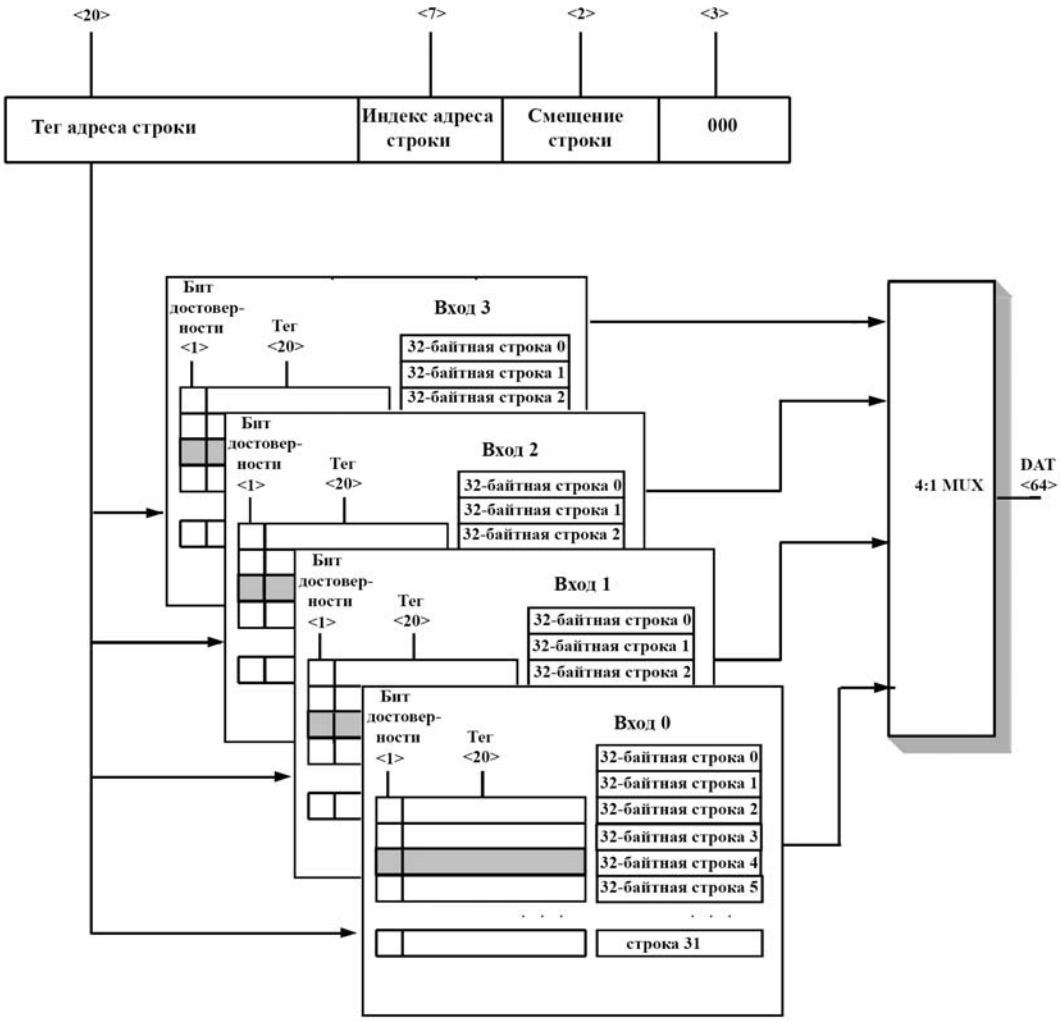


Рис. 6-7. Организация кэша команд процессора Blackfin.

Компоненты тега и данных строки кэша показаны на рис. 6-8.

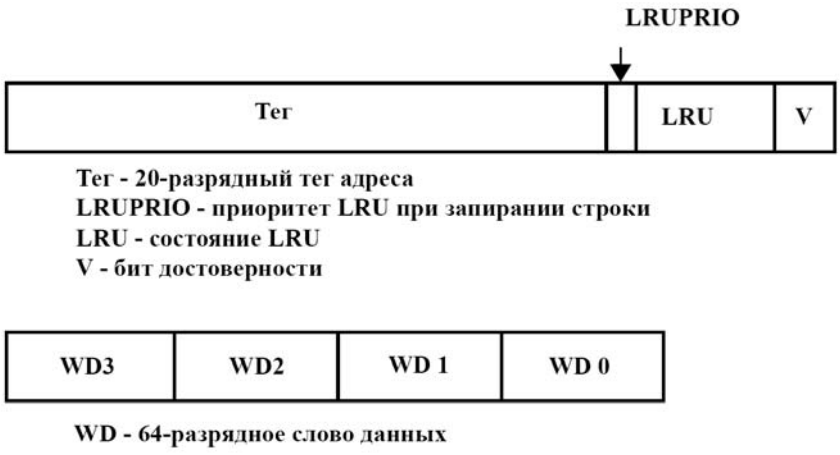


Рис. 6-8. Строка кэша – тег и данные

Совпадения и промахи в кэше

Совпадение в кэше происходит, когда адрес в запросе ядром выборки команды совпадает с достоверным элементом в кэше. Конкретнее, совпадение в кэше определяется сравнением старших 18-и битов и битов 10 и 11 адреса выбираемой команды с тегами адресов достоверных строк, содержащихся в наборе кэша. Набор кэша выбирается на основании битов 9÷5 адреса выбираемой команды. Совпадение в кэше происходит при совпадении в операции сравнения адреса с тегом, в противном случае происходит промах в кэше.

При промахе в кэше устройство памяти команд генерирует обращение типа заполнения строки, которое необходимо для получения недостающей строки из памяти, внешней по отношению к ядру. Адресом при обращении к внешней памяти является адрес требуемой команды. При промахе в кэше ядро останавливает выполнение программы до возврата требуемого слова команды из внешней памяти.

Заполнение строки кэша

Операция заполнения строки кэша включает выборку 32-х байтов данных из памяти. Эта операция начинается, когда устройство памяти команд запрашивает передачу данных типа чтения строки (передача пакета, состоящего из четырех 64-разрядных слов данных) по внешнему порту чтения данных. Адресом, используемым при чтении, является адрес требуемого слова команды. В ответ на запрос чтения строки, поступающий от устройства памяти команд, внешняя память сначала возвращает требуемое слово команды. После этого в последовательном порядке выбираются следующие три слова. Как показано в таблице 6-3 при необходимости при выборке может осуществляться циклическое обращение адреса.

Таблица 6-3. Порядок выборки слов при заполнении строки.

Требуемое слово	Порядок выборки следующих трех слов
WD0	WD0, WD1, WD2, WD3
WD1	WD1, WD2, WD3, WD0
WD2	WD2, WD3, WD0, WD1
WD3	WD3, WD0, WD1, WD2

Буфер заполнения строк

По мере получения из внешней памяти новой строки кэша, каждое 64-разрядное слово буферизируется в четырехэлементном буфере заполнения строк перед записью в 4-х Кбайтный банк памяти в L1. При использования буфера заполнения строк ядро может осуществлять обращение к данным, содержащимся в новой строке кэша, по мере ее получения из внешней памяти, а не ожидая записи её в кэш.

Замещение строки кэша

Когда устройство памяти команд сконфигурировано как кэш, биты 5÷9 адреса выборки команды используются в качестве индекса при выборе набора кэша, используемого в операции сравнения адреса с тегом. Если операция сравнения адреса с тегом приводит к промаху в кэше, устройство замещения строк кэша проверяет биты достоверности и LRU выбранного набора для определения элемента (во входе 0, 1, 2 или 3), в который будет помещена новая строка. См. рис. 6-7, «Организация кэша команд процессора Blackfin».

Устройство замещения строк кэша сначала проверяет наличие недостоверных элементов (элементов, соответствующий бит достоверности которых сброшен). Если найден хотя бы один недостоверный элемент, он выбирается в качестве приемника новой строки кэша. Если найдено несколько недостоверных элементов, выбор того из них, который будет замещён новой строкой, производится в порядке следующего приоритета:

- сначала выбирается вход 0;
- затем выбирается вход 1;
- затем выбирается вход 2;
- в последнюю очередь выбирается вход 3.

Например:

- Если вход 3 является недостоверным, а входы 0, 1 и 2 являются достоверными, для размещения новой строки кэша выбирается вход 3.
- Если входы 0 и 1 являются недостоверными, а входы 2 и 3 являются достоверными, для размещения новой строки кэша выбирается вход 0.
- Если входы 2 и 3 являются недостоверными, а входы 0 и 1 являются достоверными, для размещения новой строки кэша выбирается вход 2.

Если недостоверные элементы не найдены, логика замещения кэша использует LRU алгоритм.

Управление кэшем команд

Прямой доступ контроллера DMA системы и генераторов адреса данных ядра к кэшу команд невозможен. При использовании определенной комбинации команд и регистров ядра, отображенных в карте памяти, возможна косвенная инициализация тега команд и массивов данных, и обеспечивается механизм для проверки, инициализации и отладки кэша команд.

Необходимо явное управление когерентностью кэша команд. Для осуществления этого и гарантии того, что кэш команд выполняет выборку последней версии любого модифицированного пространства команд, следует, при необходимости, переводить элементы строк кэша команд в недостоверное состояние.

См. раздел “Перевод кэша команд в недостоверное состояние”.

Запирание кэша команд по строкам

Биты CPLB_LRUPRIO в регистрах ICPLB_DATAx (см. раздел «Свойства и защита памяти») используются для расширения возможностей управления тем, какая часть кода должна оставаться в кэше команд. При заполнении строки кэша состояние этого бита сохраняется вместе с тегом строки. Впоследствии он используется совместно с алгоритмом LRU для определения строки, которой необходимо пожертвовать в случае, когда при выборке новой кэшируемой строки все входы кэша заняты. Этот бит сигнализирует о том, что строка имеет “низкую” или “высокую” значимость. В модифицированной политике LRU строки с высокой значимостью могут замещать строки с низкой значимостью, а строки с низкой значимостью не могут замещать строки с высокой значимостью. Если все входы заняты строками с высокой значимостью, ядро выполнит выборку кэшируемой строки с низкой значимостью, но она не будет помещена в кэш. При выборке строки с высокой значимостью сначала выполняется поиск незанятых входов, затем поиск среди строк с низкой значимостью, которые дольше всех не использовались, и, наконец, среди других строк с высокой значимостью при помощи политики LRU. Строки с низкой значимостью могут замещать только незанятые входы или другие строки с низкой значимостью, при этом применяется политика LRU. Если *все* помещённые ранее в кэш строки с высокой значимостью становятся менее значимыми, они могут быть одновременно преобразованы в строки с низкой значимостью при помощи записи бита LRUPRIRST в регистре IMEM_CONTROL.

Запирание кэша команд по входам

Кэш команд имеет четыре независимых бита запирания (ILOC[3:0]), которые управляют каждым из его четырех входов. Когда кэш разрешён, в памяти команд L1 доступны 4 входа. Установление бита запирания для определенного входа предотвращает участие этого входа в политике замещения LRU. Т.о., команды, помещенные в кэш, соответствующий вход которых заперт, могут быть удалены из кэша только при помощи команды IFLUSH или манипуляции массивом тегов с применением “скрытых” регистров, отображенных в карте памяти.

Ниже представлен пример последовательности действий при запирании входа 0:

- Если требуемый код может уже находиться в кэше команд, в первую очередь необходимо перевести весь кэш в недостоверное состояние (пример см. в разделе «Управление достоверностью кэша команд»).
- При необходимости следует запретить прерывания, чтобы предотвратить возможное повреждение запертого кэша программами обслуживания прерываний.
- Заприте остальные входы, установив ILOC[3:1]. После этого только вход 0 кэша команд может замещаться новым кодом.
- Выполните требуемый код. Любые кэшируемые исключения, такие как выход из кода, возникающие при выполнении, также запираются в кэше команд.

- При выходе из кода, критичного ко времени, сбросьте ILOC[3:1] и установите ILOC[0]. После этого критичный код (и команда, установившая ILOC[0]) запирается во входе 0.
- При необходимости следует снова разрешить прерывания.

Если все четыре входа кэша заперты, дальнейшее помещение команд в кэш становится невозможным.

Перевод кэша в недостоверное состояние

Кэш команд может переводиться в недостоверное состояние по адресам, строкам кэша и полностью. Явный перевод строк кэша в недостоверное состояние по адресу строки возможен при помощи команды IFLUSH. Требуемый адрес команды формируется в P-регистрах. Так как кэш команд не должен содержать модифицированные данные, строки кэша просто переводятся в недостоверное состояние.

В следующем примере регистр P2 содержит адрес достоверной ячейки памяти. Если этот адрес передается в кэш, соответствующая строка кэша переводится в недостоверное состояние после выполнения команды.

Пример команды IFLUSH:

```
iflush [p2];  /* Строка кэша, содержащая адрес, на который
указывает P2, переводится в недостоверное состояние */
```

Так как команда IFLUSH используется для перевода в недостоверное состояние определенного адреса в карте памяти, использование ее для перевода в недостоверное состояние всего входа или банка кэша является непрактичным. Для прямого перевода больших частей кэша в недостоверное состояние может использоваться второй метод, заключающийся в установке битов достоверности для каждой строки в недостоверное состояние. Для реализации этого метода имеются дополнительные регистры, отображенные в карте памяти (ITEST_COMMAND и ITEST_DATA[1:0]), позволяющие выполнять чтение/запись всех элементов кэша напрямую. Этот метод обсуждается в следующем разделе.

Третий метод позволяет перевести весь кэш команд в недостоверное состояние. При сбросе бита IMC в регистре IMEM_CONTROL (см. рис. 6-5 «Регистр управления памятью команд L1») все биты достоверности в кэше команд устанавливаются в недостоверное состояние. Вторая запись в регистр IMEM_CONTROL с установкой бита IMC снова конфигурирует память команд как кэш. Перед переводом кэша в недостоверное состояние следует выполнить команду SSYNC, после каждой такой операции необходимо выполнить команду CSYNC.

Регистры проверки команд

Регистры проверки команд позволяют производить чтение/запись всех элементов кэша L1 напрямую. Их применение позволяет инициализировать массивы данных и тегов команд и обеспечивает механизм проверки, инициализации и отладки кэша.

При помощи регистра команд проверки команд (ITEST_COMMAND) осуществляется обращение к массивам данных или тегов кэша L1, через регистры данных проверки команд (ITEST_DATA[1:0]) выполняется передача данных. Регистры ITEST_DATAx содержат 64-разрядные записываемые или прочитанные данные. Младшие 32 бита помещаются в регистр ITEST_DATA[0], старшие 32 бита – в регистр ITEST_DATA[1]. При обращении к массивам тегов используется ITEST_DATA[0].

Следующие рисунки описывают регистры ITEST:
- рис. 6-9 “Регистр команд проверки команд”;
- рис. 6-10 “Регистр данных проверки команд 1”;
- рис. 6-11 “Регистр данных проверки команд 0”.

Доступ к этим регистрам возможен только в режимах Супервизора и Эмуляции. При записи в регистры ITEST, всегда следует сначала выполнять запись в регистры ITEST_DATAx, а затем в регистр ITEST_COMMAND. При чтении регистров ITEST необходимо выполнить эту последовательность в обратном порядке -– сначала выполняется чтение регистра ITEST_COMMAND, затем – регистров ITEST_DATAx.

Регистр команд проверки команд (ITEST_COMMAND)

При записи в регистр команд проверки команд (ITEST_COMMAND) выполняется обращение к массивам тегов или данных кэша L1 и передача данных через регистры данных проверки команд (ITEST_DATA[1:0]).

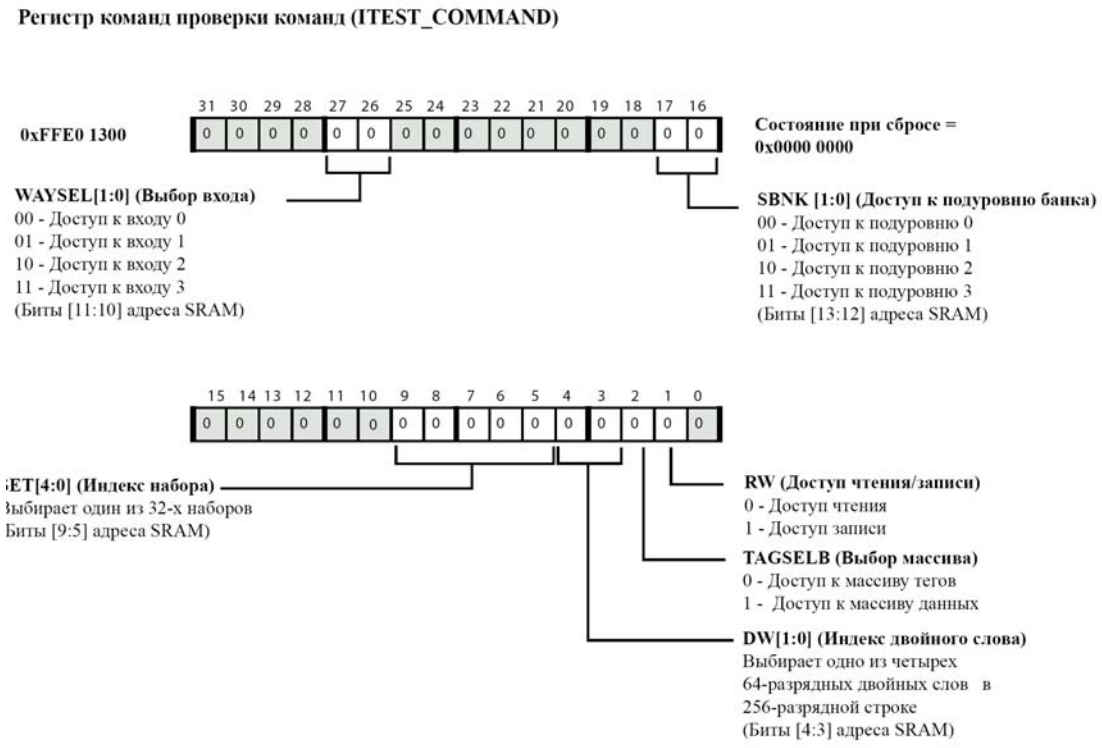


Рис. 6-9. Регистр команд проверки команд.

Регистр данных проверки команд (ITEST_DATA1)

Регистры данных проверки команд (`ITEST_DATA[1:0]`) используются для доступа к массивам данных кэша L1. Они содержат 64-разрядные данные, используемые при записи или получаемые при чтении. В регистр данных проверки команд 1 (`ITEST_DATA1`) помещаются старшие 32 бита.

Рис. 6-10. Регистр данных проверки команд 1.

Регистр данных проверки команд 0 (ITEST_DATA0)

Регистр данных проверки команд 0 (`ITEST_DATA0`) содержит младшие 32 бита 64-разрядных данных, используемых при записи или получаемых при чтении. Регистр `ITEST_DATA0` также используется для доступа к массивам тегов. Он также содержит биты достоверности и модификации, отображающие состояние строки кэша.

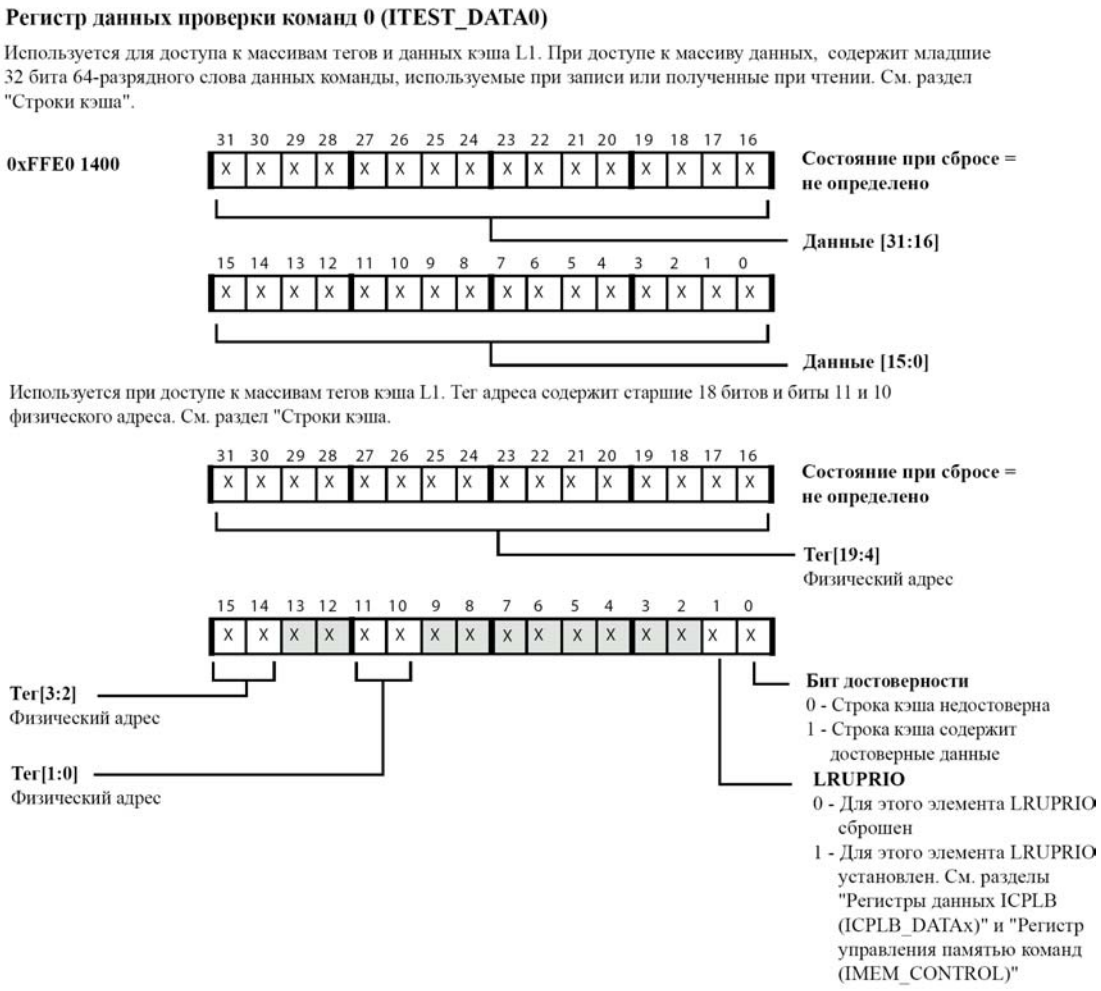


Рис. 6-11. Регистр данных проверки команд 0.

Память данных L1

SRAM/кэш данных L1 состоит из однопортовых подразделов, однако организован таким образом, чтобы снизить вероятность конфликтов доступа. Эта организация приводит к очевидному многопортовому характеру работы. В отсутствии конфликтов за один такт ядра могут выполняться следующие операции с данными в L1:

- две 32-разрядных загрузки с использованием DAG;
- одно конвейерное 32-разрядное сохранение в память с использованием DAG;
- одна 64-разрядная операция ввода/вывода с использованием DAG;
- один 64-разрядный доступ типа жертвования/заполнения строки кэша.

Память данных L1 может использоваться только для хранения данных.

Регистр управления памятью данных (DMEM_CONTROL)

Регистр управления памятью данных (DMEM_CONTROL) содержит биты управления памятью данных L1.

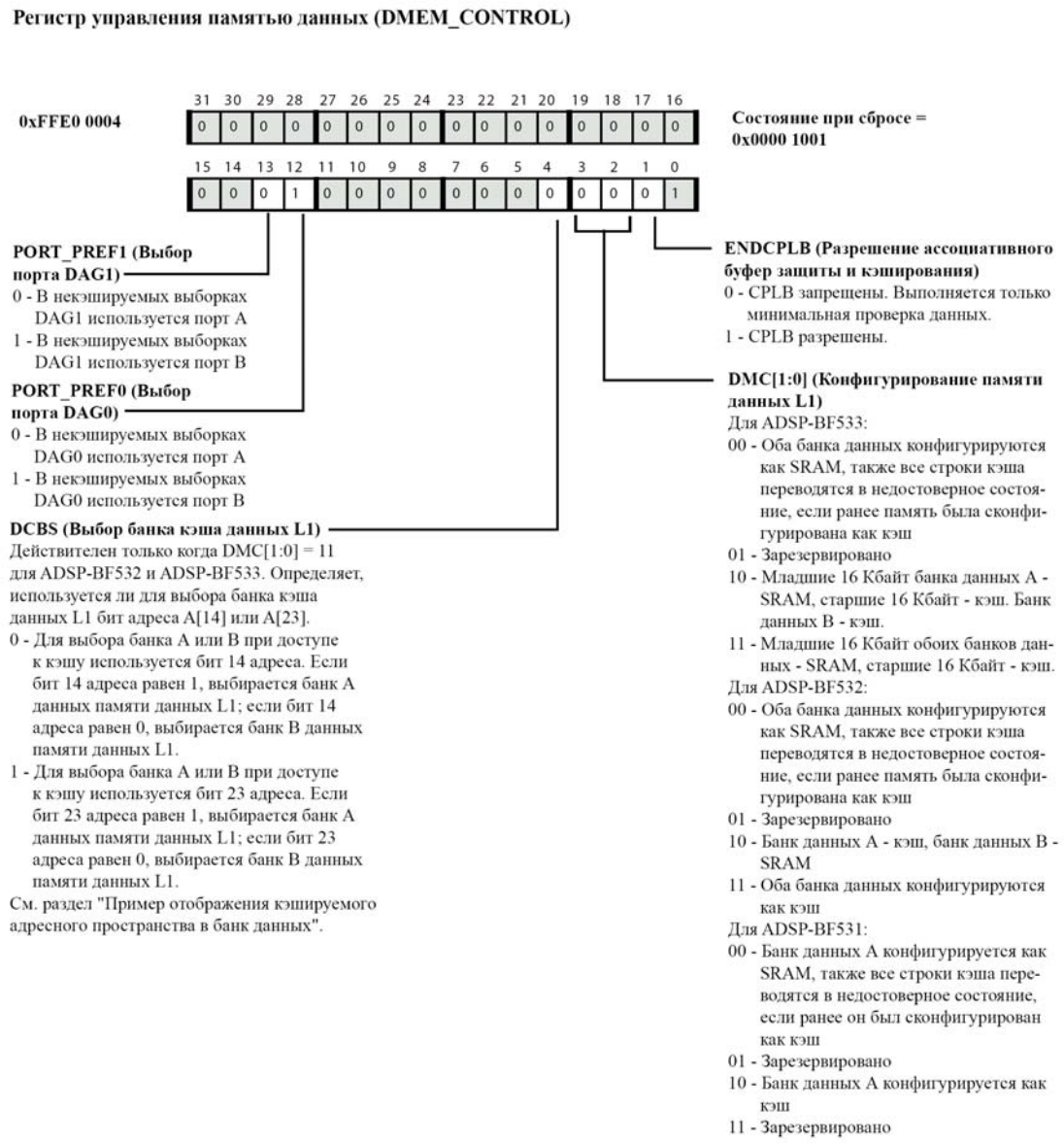


Рис. 6-12. Регистр управления памятью данных.

Бит PORT_PREF1 используется для выбора порта данных, используемого для обработки некэшируемых выборок DAG1 из L2. Кэшируемые выборки всегда обрабатываются портом данных, физически связанным с кэш памятью, к которой выполняется обращение. Распределение трафика кэша, DAG0 и DAG1 по разным портам оптимизирует производительность, поддерживая заполненной очередь обращения к памяти L2.

Бит PORT_PREF0 используется для выбора порта данных, используемого для обработки некэшируемых выборок DAG0 из L2. Кэшируемые выборки всегда обрабатываются портом данных, физически связанным с кэш памятью, к которой

производится обращение. Распределение трафика кэша , DAG0 и DAG1 по разным портам оптимизирует производительность, поддерживая заполненной очередь в L2.

Для достижения оптимальной производительности при двойных операциях чтения DAG, DAG0 и DAG1 должны конфигурироваться на использование разных портов. Например, если при конфигурировании в PORT_PREF0 записывается 1, то в PORT_PREF1 должен записываться 0.

Бит DCBS обеспечивает определенный контроль над тем, какие из адресов будут попадать в один и тот же набор. Этот бит может определять, какие адреса будут оставаться в кэше, позволяя избегать жертвования циклически используемых наборов. Состояние этого бита не имеет значения до тех пор, пока оба банка данных (A и B) не используются как кэш (для этого биты DMC[1:0] этого регистра должны быть установлены в 11).

Бит ENDCPLB используется для разрешения/запрещения применения 16 ассоциативных буферов защиты и кэширования (CPLBs), используемых для данных (см. раздел «Кэш данных L1»). По умолчанию после сброса использование CPLB данных запрещено, при этом интерфейс памяти L1 выполняет только минимальную проверку адреса. При этой минимальной проверке исключение генерируется, когда процессор:

- адресует несуществующее (зарезервированное) пространство памяти L1;
- пытается выполнить невыровненный доступ к памяти;
- пытается выполнить доступ к пространству регистров, отображенных в карте памяти, при помощи DAG1 или при нахождении в Пользовательском режиме.

Использование CPLB должно быть запрещено при помощи этого бита при обновлении дескрипторов CPLB (регистров DCPLB_DATAx и DCPLB_ADDRx). Следует отметить, что вследствие применения “слабого” упорядочивания операций загрузки/сохранения (см. раздел “Упорядочивание операций загрузки регистров и сохранение в память”), запрещению CPLB должна предшествовать команда CSYNC.

При разрешении или запрещении кэша или CPLB для гарантии корректной работы после записи в регистр DMEM_CONTROL должна следовать команда SSYNC.

По умолчанию после сброса вся память данных L1 доступна как SRAM. Биты DMC[1:0] могут использоваться в целях резервирования частей этой памяти для работы в качестве кэша. Резервирование памяти для работы в качестве кэша само по себе не разрешает кэширование обращений к памяти L2. Для того, чтобы разрешить их кэширование, также должно быть разрешено использование CPLB (при помощи бита ENDCPLB), и дескрипторы CPLB (регистры DCPLB_DATAx и DCPLB_ADDRx) должны определять выбранные страницы памяти как доступные для кэширования.

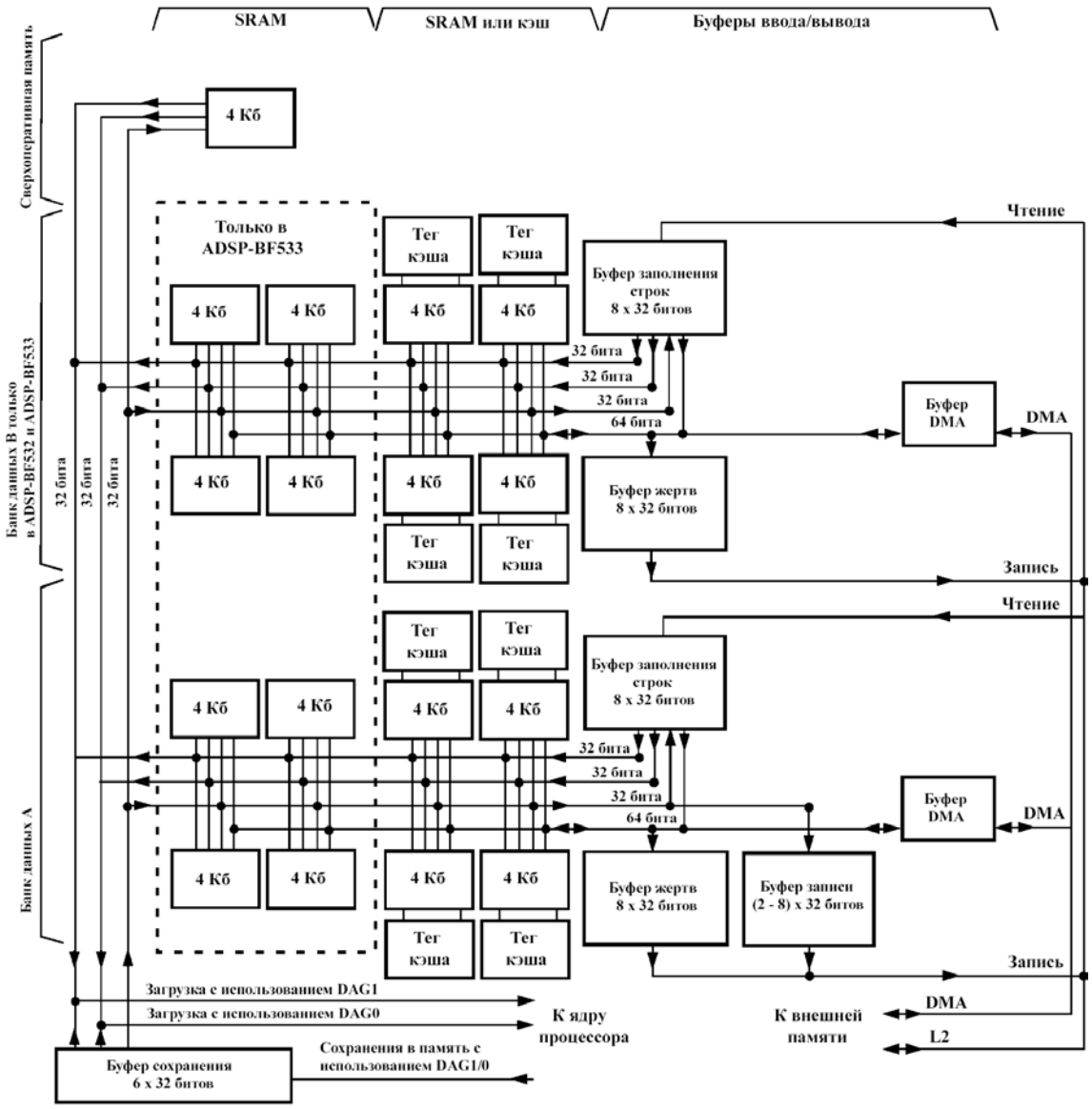
По умолчанию после сброса использование памяти в качестве кэша и проверка адресов с помощью CPLB запрещены.

Для гарантии корректной работы и совместимости с будущими проектами, при каждой записи в этот регистр зарезервированные биты должны устанавливаться в 0.

SRAM данных L1

Конфликты при обращениях к SRAM не возникают, пока не выполняется обращение к одному и тому же 32-разрядному слову (биты 2 адресов совпадают), одному подуровню банка объёмом 4 Кбайта (биты 12 и 13 адресов совпадают), одной 16 Кбайтной половине подуровня банка (биты 16 адресов совпадают) и к одному банку (биты 20 и 21 адресов совпадают). При обнаружении конфликта адреса, номинально в первую очередь доступ предоставляется генераторам адреса данных, затем буферу сохранения и, наконец, DMA и трафику жертвования/заполнения кэша. Чтобы гарантировать адекватную скорость выполнения передач DMA им назначается высший приоритет в случае, если они блокируются в течении 16 тактов ядра подряд или, если вторая операция ввода/вывода в режиме DMA ставится в очередь, до обработки первой операции ввода/вывода вывода в режиме DMA.

В таблице 6-4 показано размещение в карте памяти подуровней банков.



На рис. 6-13 показана архитектура памяти данных L1.

Таблица 6-4. Начальные адреса подуровней банков памяти данных SRAM.

Банк и подуровень банка памяти	ADSP-BF533	ADSP-BF532	ADSP-BF531
Банк данных А, подуровень 0	0xFF80 0000	-	-
Банк данных А, подуровень 1	0xFF80 1000	-	-
Банк данных А, подуровень 2	0xFF80 2000	-	-
Банк данных А, подуровень 3	0xFF80 3000	-	-
Банк данных А, подуровень 4	0xFF80 4000	0xFF80 4000	0xFF80 4000
Банк данных А, подуровень 5	0xFF80 5000	0xFF80 5000	0xFF80 5000
Банк данных А, подуровень 6	0xFF80 6000	0xFF80 6000	0xFF80 6000
Банк данных А,	0xFF80 7000	0xFF80 7000	0xFF80 7000

подуровень 7			
Банк данных B, подуровень 0	0xFF90 0000	-	-
Банк данных B, подуровень 1	0xFF90 1000	-	-
Банк данных B, подуровень 2	0xFF90 2000	-	-
Банк данных B, подуровень 3	0xFF90 3000	-	-
Банк данных B, подуровень 4	0xFF90 4000	0xFF90 4000	-
Банк данных B, подуровень 5	0xFF90 5000	0xFF90 5000	-
Банк данных B, подуровень 6	0xFF90 6000	0xFF90 6000	-
Банк данных B, подуровень 7	0xFF90 7000	0xFF90 7000	-

Кэш данных L1

Информацию о терминологии, касающейся кэша, см. в разделе «Терминология».

Когда кэш данных разрешен (управляется битами `DMC[1:0]` в регистре `DMEM_CONTROL`), либо банк данных A объёмом 16 Кбайт, либо оба банка данных (A и B), объёмом по 16 Кбайт каждый, могут быть настроены для использования в качестве кэша. В ADSP-BF533 используются старшие 16 Кбайт. В ADSP-BF531 доступен только банк данных A. В отличие от кэша команд, который является четырехвходовым наборно-ассоциативным, кэш данных является двухвходовым наборно-ассоциативным. Если для использования в качестве кэша доступны и разрешены два банка, вместо входов создаются дополнительные наборы. Когда и банк данных A, и банк данных B содержат память, работающую как кэш, для управления тем, каким из банков кэш-памяти обрабатываются обращения к определенной половине всего адресного пространства может использоваться бит `DCBS` в регистре `DMEM_CONTROL`. Бит `DCBS` используется для выбора бита адреса (14 или 23), управляющего распределением трафика между банками кэша. Он обеспечивает определенное управление над тем, какие из адресов попадают в один набор. Таким образом, этот бит может влиять на то, какие из адресов остаются в кэше, избегая жертвования циклически используемых наборов.

Конфликты при обращениях к кэшу не возникают, если они не производятся к одному подуровню банка объёмом 4 Кбайт, одной половине банка и к одному банку. Кэш обладает менее ярко выраженным многопортовым характером работы по сравнению со SRAM вследствие непроизводительных затрат на поддержку тегов. При конфликте адресов кэша, в первую очередь доступ обращениям с использованием регистра DTEST, затем буферу сохранения и, наконец, трафику жертвования/заполнения строки кэша.

Существует три различных режима работы кэша:
- сквозная запись с выделением строк кэша только в операциях записи;
- сквозная запись с выделением строк кэша и при чтении и при записи;
- отложенная запись с выделением строк кэша и при чтении и при записи.

Режим работы кэша задаётся дескрипторами CPLB (см. раздел “Свойства и защита памяти”). Так как для каждой страницы памяти режим работы кэша выбирается независимо, одновременно может использоваться любая комбинация этих режимов.

Если кэш разрешен (управляется битами DMC[1:0] в регистре DMEM_CONTROL), следует также разрешить использование CPLB данных (управляется битом ENDCPLB в регистре DMEM_CONTROL). Кэшируются только страницы памяти, определяемые как кэшируемые в CPLB данных. По умолчанию при запрещении CPLB данных кэширование не производится.

Конфигурирование в CPLB данных регистров, отображенных в карте памяти, или банков данных, служащих в качестве SRAM L1, как доступных для кэширования может привести к ошибкам в работе процессора.

Пример отображения кэшируемого адресного пространства в банках данных

Ниже приводится пример отображения кэшируемого адресного пространства в двух банках данных.

Когда оба банка в ADSP-BF533 и ADSP-BF532 сконфигурированы как кэш, они работают как два независимых двухвходовых наборно-ассоциативных кэша объёмом по 16 Кбайт каждый, которые могут независимо отображаться в адресном пространстве процессора Blackfin.

Если оба банка сконфигурированы как кэш, бит DCBS регистра DMEM_CONTROL назначает в качестве признака выбора банка кэша бит адреса A[14] или A[23]. Бит адреса A[14] или A[23] выбирает кэш, реализуемый банком данных A или банком данных B.

- Если DCBS = 0, то частью индекса адреса является A[14], и все адреса, в которых A[14] = 0, используют банк данных B. Все адреса, в которых A[14] = 1 используют банк данных A.

  В этом случае A[23] обрабатывается просто как один из битов адреса, который входит в состав тега кэша и используется в операциях сравнения для принятия решении о совпадении/промахе в кэше.

- Если DCBS = 1, то частью индекса адреса является A[23], и все адреса, в которых A[23] = 0, используют банк данных B. Все адреса, в которых A[23] = 1 используют банк данных A.

  В этом случае A[14] обрабатывается просто как один из битов адреса, который входит в состав тега кэша и используется в операциях сравнения для принятия решении о совпадении/промахе в кэше.

  В результате выбора DCBS = 0 или DCBS = 1:

- Если DCBS = 0, для выбора между банками данных A и B используется A[14].

  В двух областях кэш-памяти объёмом по 16 Кбайт, реализуемых двумя банками данных, отображаются чередующиеся страницы памяти объёмом 16 Кбайт.
  Следовательно:
  Любые данные из первых 16 Кбайт памяти могут помещаться только в банк данных B.
  Любые данные из следующего диапазона адресов (16 Кбайт ÷ 32 Кбайта – 1) могут помещаться только в банк данных A.
  Любые данные из следующего диапазона адресов (32 Кбайта ÷ 48 Кбайт – 1) могут помещаться только в банк данных B.
  И так далее.
  В результате кэш работает как один неразрывный двухвходовый наборно-ассоциативный кэш объёмом 32 Кбайта. Каждый вход занимает 16 Кбайт. Все элементы данных с совпадающими первыми 14 битами адреса указывают на отдельный набор, в котором может храниться до двух элементов (по одному в каждом входе).

- Если DCBS = 0, для выбора между банками данных A и B используется A[23].
  Когда DCBS=1, работа системы более похожа на работу двух независимых блоков кэш-памяти, каждый из которых представляет собой двухвходовый наборно-ассоциативный кэш объёмом 16 Кбайт. Каждый банк обслуживает чередующиеся через один наборы блоков памяти объёмом 8 Мбайт.
  Например, банк данных B кэширует все обращения к данным в первых 8 Мбайтах диапазона адресов памяти. Таким образом, данные из каждой области объёмом 8 Мбайт (а не из чередующихся блоков по 16 Кбайт) конкурируют за две строки. Аналогично, банк данных A кэширует данные, расположенные в диапазоне адресов от 8 до 16 Мбайт.
  Например, если приложение работает с одним набором данных размером 1 Мбайт, целиком расположенной в первых 8 Мбайтах памяти, он полностью обслуживается одной половиной кэша, то есть банком данных B (двухвходовым наборно-ассоциативным кэшем объёмом 16 Кбайт). В этом случае приложение не получает преимуществ от использования банка A.

Для большинства приложений предпочтительнее работать с DCBS=0.

Однако, если приложение работает с двумя наборами данных, расположенных в двух пространствах памяти, разнесённых, по меньшей мере, на 8 Мбайт, возможно более детальное управление над помещением данных в кэш. Например, если программа выполняет ряд двойных операций умножения-накопления, при которых оба генератора адреса данных выполняют доступ к данным на каждом такте, при отображении наборов данных, используемых генераторами адреса данных, в разные банки данных гарантируется следующее:

- DAG0 при всех доступах получает данные из банка данных A,
- DAG1 получает данные из банка данных B.

При таком упорядочивании ядро использует для передач строк кэша обе шины данных, и достигается максимальная разрядность передач данных между кэшем и ядром.

На рис. 6-14 показан пример отображения памяти в кэше данных при `DCBS = 1`.

Значение `DCBS` может изменяться динамически; но для того, чтобы гарантировать отсутствие потерь данных, необходимо предварительно сбросить весь кэш и перевести его в недостоверное состояние.

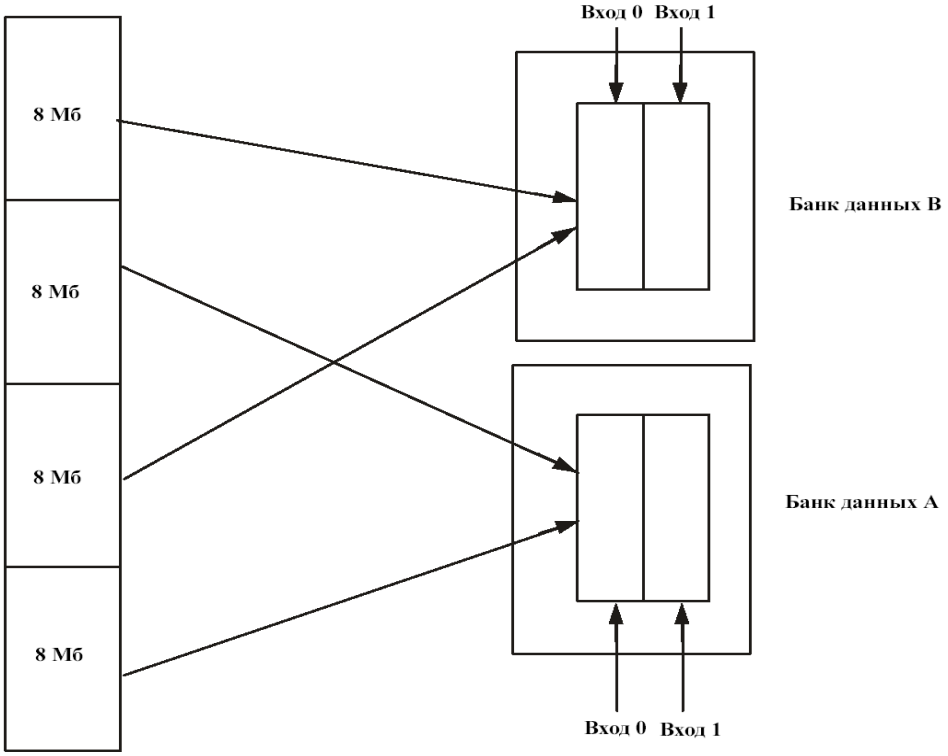


Рис. 6-14. Отображение в кэше данных при `DCBS = 1`.

Доступ к кэшу данных

Контроллер кэша сравнивает адрес, формируемый генератором адреса данных, с битами тега. Если логический адрес присутствует в кэше L1, происходит совпадение в кэше, и осуществляется доступ к данным в L1. Если логический адрес в кэше L1 отсутствует, происходит промах в кэше, и транзакция передается на следующий уровень памяти при помощи интерфейса системы. Индекс строки и политика замещения строк контроллером кэша определяют тег кэша и пространство памяти, выделяемые под данные, возвращаемые внешней памятью.

Строка кэша данных может принимать одно из трех состояний: недостоверное, эксклюзивное (достоверная и чистая строка) и модифицированное (достоверная и измененная строка). Если выделяемая строка уже занята достоверными данными,

и кэш сконфигурирован для работы в режиме с отложенной записью, контроллер проверяет состояние строки кэша и поступает соответствующим образом:

- если состояние строки эксклюзивное (строка чистая), новые тег и данные записываются поверх старой строки;
- если состояние строки модифицированное (строка содержит измененные данные), то кэш содержит единственную достоверную копию данных.

Если строка содержит измененные данные, текущее содержимое кэша копируется во внешнюю память перед записью новых данных в кэш.

В процессоре имеются буферы “жертв” и буферы заполнения строк. Эти буферы используются, если при промахе в кэше, вызванном операцией загрузки, генерируется жертвуемая строка кэша, которую следует заменить. Операция заполнения строки переходит к внешней памяти. Кэш данных выполняет запрос системе на заполнение строки (как запрос критичного слова) и, затем, направляет эти данные в ожидающий DAG по мере обновления им строки кэша. Другими словами, кэш выполняет перенаправление критичных слов.

Кэш данных поддерживает режимы совпадений при промахе в операции сохранения (hit-under-a-store-miss) и совпадений при промахе в операции предвыборки (hit-under-a-prefetch-miss). Другими словами, при промахе в операции записи или при выполнении команды `PREFETCH`, вызывающей промах в кэше (при обращении к области, доступной для кэширования), в конвейере команд добавляется, по меньшей мере, 4 такта останова. Более того, последующая команда загрузки или сохранения может вызвать совпадение в кэше L1 по завершению заполнения строки.

Прерывания достаточного приоритета (по отношению к текущему контексту) отменяют остановленную команду загрузки. Следовательно, если при выполнении операции загрузки регистра происходит промах при обращении к кэшу памяти данных L1 и в интерфейсе системы генерируется операция заполнения строки, имеющая высокую задержку, можно вызвать прерывание ядра, в результате чего ядро начинает обработку другого контекста. Доступ системы с заполнением строки кэша не отменяется, и кэш данных обновляется новыми данными до того, как будут обслужены любые дальнейшие операции, способные вызвать промах в кэше в соответствующем банке данных. Дополнительную информацию см. в разделе «Исключения».

Метод записи кэша

Операции записи кэш-памяти могут реализовываться с использованием метода сквозной записи или метода отложенной записи:

- При каждой операции сохранения кэш со сквозной записью инициирует запись во внешнюю память одновременно с записью в кэш.

  Если строка кэша замещается или явно очищается программным способом, вместо записи во внешнюю память содержимое строки кэша переводится в недостоверное состояние.

- Кэш с отложенной записью не выполняет запись во внешнюю память до тех пор, пока строка не будет замещена при выполнении операции загрузки, требующей использования этой строки.

В каждом банке памяти данных L1 реализован буфер копирования, разрядность которого равна полной разрядности строки кэша. Кроме того, двухэлементный буфер записи в памяти данных L1 предназначен для обслуживания всех операций сохранения с запрещением кэширования или защитой от сквозного сохранения. Буфер записи очищается командой `SSYNC`.

Регистр приоритета прерываний (IPRIO) и глубина буфера записи.

Регистр приоритета прерываний (`IPRIO`) может использоваться для управления буфером записи порта A (см. рис. 6-13 “Архитектура памяти данных L1”).

Биты `IPRIO[3:0]` могут применяться для программного задания отметки уровня прерывания низкого приоритета. При возникновении прерывания, вызывающего переход процессора от программы обслуживания прерывания с низким приоритетом программе обслуживания прерывания с высоким приоритетом, размер буфера записи увеличивается от двух до восьми слов. Это свойство позволяет программе обслуживания прерывания осуществлять работу и выполнять запись без добавления начальных тактов останова в случае, когда буфер записи уже заполнен в результате работы программы обслуживания предыдущего прерывания. Оно наиболее полезно при осуществлении записей в медленную внешнюю память. При возврате из программы обслуживания прерывания с высоким приоритетом в программу обслуживания прерывания с низким приоритетом или в Пользовательский режим, ядро добавляет циклы останова до тех пор, пока буфер записи не завершит операции записи, необходимые для обратного перехода в двухэлементное состояние. По умолчанию буфер записи является буфером FIFO с фиксированной глубиной в два элемента.

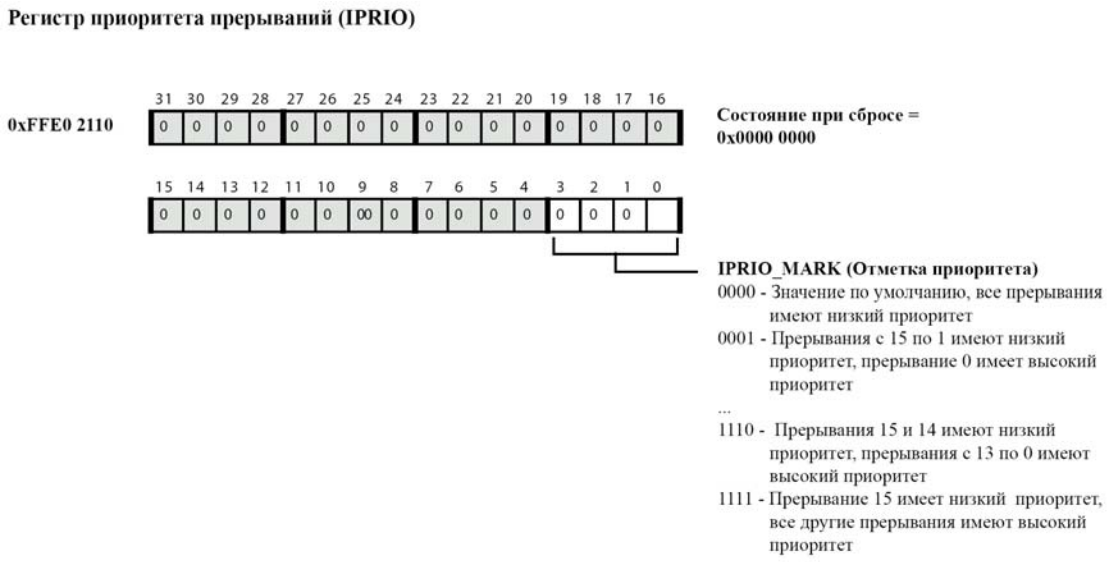


Рис. 6-15. Регистр приоритета прерываний.

Команды управления кэшем данных

В процессоре определены три команды управления кэшем данных, доступные в Пользовательском режиме и режиме Супервизора. Этими командами являются PREFETCH, FLUSH и FLUSHINV.

- Команда PREFETCH (предварительная выборка) пытается выделить строку в кэше L1. Если при выполнении предварительной выборки возникает совпадение в кэше, генерируется исключение или адресуется область, недоступная для кэширования, команда PREFETCH работает аналогично команде NOP.
- Команда FLUSH (сброс кэша данных) вызывает синхронизацию определенной строки кэша данных с внешней памятью. Если кэшированная строка является модифицированной, по этой команде она записывается из кэша во внешнюю память и помечается как чистая. Если указанная строка кэша данных является чистой или не существует, команда FLUSH работает аналогично команде NOP.
- Команда FLUSHINV (сброс и перевод в недостоверное состояние строки кэша данных) вызывает выполнение кэшем данных той же процедуры, что и при команде FLUSH, с последующим переводом определенной строки кэша в недостоверное состояние.

Если программное обеспечение требует синхронизации с аппаратными средствами системы, для гарантии завершения операции сброса кэша команда FLUSH должна сопровождаться командой SSYNC. Если упорядочение необходимо для гарантии того, что все предыдущие операции сохранения в память завершены (проведены через все очереди), перед командой FLUSH следует выполнить команду SSYNC.

Перевод кэша данных в недостоверное состояние

Помимо команды FLUSHINV, описанной в предыдущем разделе, существуют два дополнительных метода перевода кэша данных в недостоверное состояние, которые не требуют сброса кэша. Первый метод заключается в простой установке бита достоверности каждой строки кэша в недостоверное состояние. Для реализации этого метода существуют дополнительные регистры, отображенные в карте памяти, (DTEST_COMMAND и DTEST_DATA[1:0]), позволяющие выполнять произвольные операции чтения/записи всех элементов кэша напрямую. Этот метод обсуждается в следующих разделах.

Второй метод реализует перевод в недостоверное состояние всего кэша данных. При сбросе битов DMC[1:0] в регистре DMEM_CONTROL (см. рис. 6-12, “Регистр управления памятью данных L1”) все биты достоверности кэша данных устанавливаются в недостоверное состояние. Повторная запись в регистр DMEM_CONTROL с целью установки битов DMC[1:0] в их предыдущее состояние приводит к возврату предыдущей конфигурации памяти данных как кэш/SRAM. Перед переводом кэша в недостоверное состояние следует выполнить команду SSYNC, после каждой из этих операций необходимо добавить команду CSYNC.

Регистры проверки данных

Как и память команд L1, память данных L1 содержит дополнительные регистры, отображенные в карте памяти, которые позволяют выполнять произвольные операции чтения/записи всех элементов кэша напрямую. Эти регистры обеспечивают механизм проверки, инициализации и отладки кэша данных.

При записи в регистр команд проверки данных (DTEST_COMMAND) осуществляется доступ к массивам тегов или данных кэша L1; данные передаются через регистры данных проверки данных (DTEST_DATA[1:0]). Регистры DTEST_DATA[1:0] содержат записываемые 64-разрядные данные или являются приемниками при чтении 64-разрядных данных. В регистре DTEST_DATA[0] содержатся младшие 32 бита, в регистре DTEST_DATA[1] содержатся старшие 32 бита. Для доступа к массивам тегов используется регистр DTEST_DATA[0].

После записи в регистр DTEST_COMMAND требуется выполнение команды CSYNC.

Регистры DTEST представлены на следующих рисунках:
- рис. 6-16 «Регистр команд проверки данных»;
- рис. 6-17 «Регистр данных проверки данных 1»;
- рис. 6-18 «Регистр данных проверки данных 0».

Доступ к этим регистрам возможен только в Пользовательском режиме или режиме Супервизора. При записи в регистры DTEST необходимо всегда выполнять сначала запись в регистры DTEST_DATA, а затем в регистр DTEST_COMMAND.

Регистр команд проверки данных (DTEST_COMMAND)

При записи в регистр команд проверки данных (DTEST_COMMAND) осуществляется доступ к массивам тегов или данных кэша L1, и данные передаются через регистры данных проверки данных (DTEST_DATA).

Бит доступа к командам/данным позволяет осуществлять прямое обращение к SRAM команд L1 при помощи регистра DTEST_COMMAND.

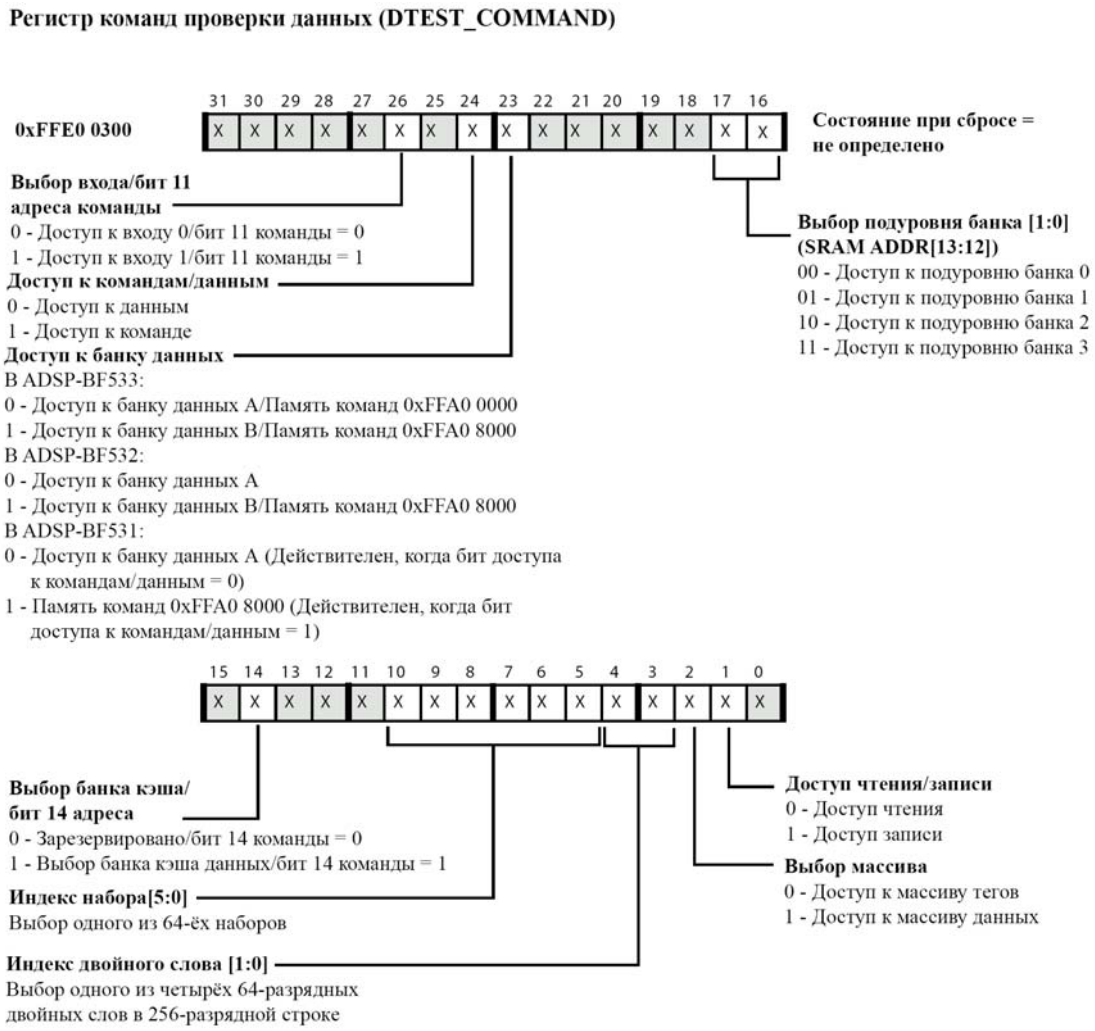


Рис. 6-16. Регистр команд проверки данных.

Регистр данных проверки данных 1 (DTEST_DATA1)

Регистры данных проверки данных (DTEST_DATA[1:0]) содержат 64-разрядные записываемые данные или являются приемниками 64-разрядных данных при чтении. В регистре данных проверки данных 1 (DTEST_DATA1) хранятся старшие 32 бита.

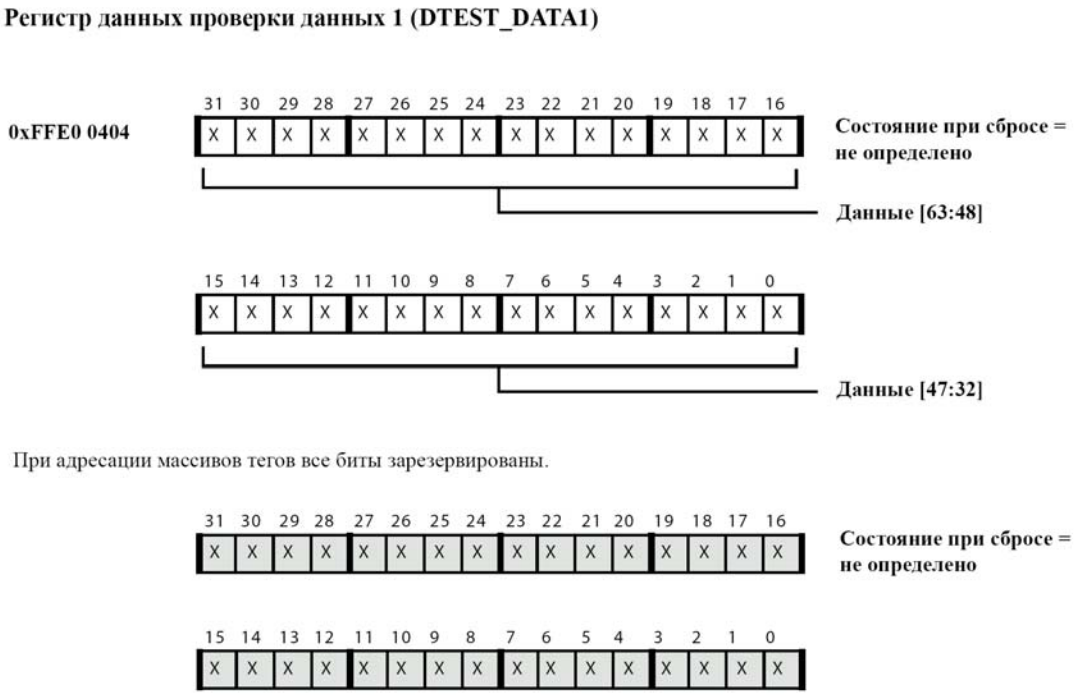


Рис. 6-17. Регистр данных проверки данных 1.

Регистр данных проверки данных 0 (DTEST_DATA0)

Регистр данных проверки данных 0 (`DTEST_DATA0`) содержит младшие 32 бита записываемых 64-разрядных данных или является приёмником младших 32 битов 64-разрядных данных при чтении. Регистр `DTEST_DATA0` также используется для доступа к массиву тегов и содержит биты достоверности и модификации, отображающие состояние строки кэша.

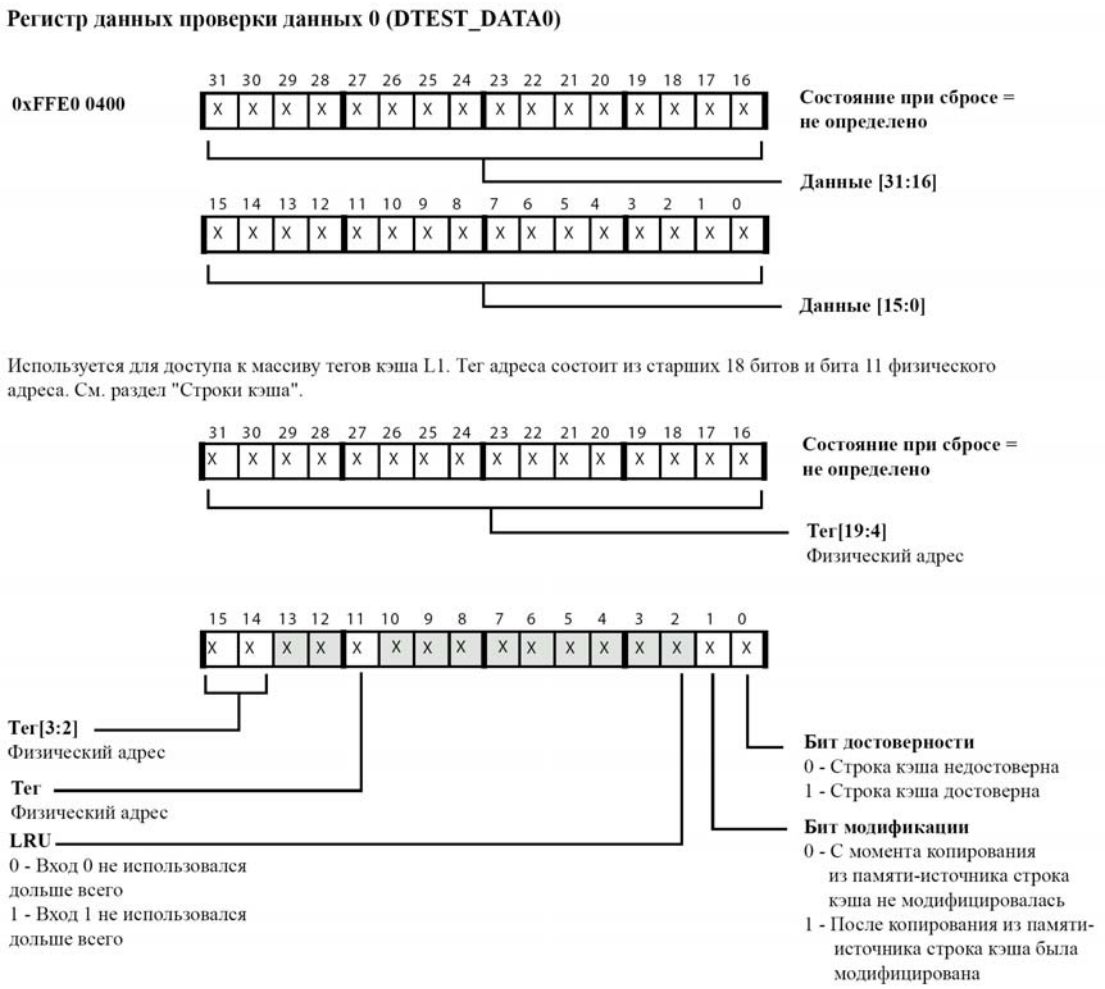


Рис. 6-18. Регистр данных проверки данных 0.

Внешняя память

Пространство внешней памяти показано на рис. 6-1. Одна из областей памяти предназначена для поддержки SDRAM. Размер банка SDRAM может изменяться программно от 16 до 128 Мбайт. Начальный адрес банка – 0x0000 0000.

Каждый из следующих четырех банков имеет размер 1 Мбайт и предназначен для поддержки асинхронной памяти. Начальный адрес банка асинхронной памяти – 0x2000 0000.

Свойства и защита памяти

В данном разделе описывается устройство управления памятью (MMU, Memory Management Unit), страницы памяти, управление CPLB, управление MMU и регистры CPLB.

Устройство управления памятью

В состав процессора Blackfin входит устройство управления памятью (MMU), в котором реализован страничный подход. Этот механизм обеспечивает управление кэшированием областей памяти, а также управление атрибутами защиты на уровне страниц. MMU обеспечивает большую гибкость при распределении памяти и ресурсов ввода/вывода между задачами и полный контроль прав доступа и параметров кэша.

MMU реализован в виде двух блоков 16-элементной ассоциативной памяти (CAM, Content Addressable Memory). Каждый элемент называется дескриптором ассоциативного буфера защиты и кэширования (CPLB). Когда работа MMU разрешена, каждый достоверный элемент в нем проверяется при выполнении любой операции выборки, загрузки или сохранения в память, на предмет соответствия между запрашиваемыми адресами и страницей, описываемой элементом CPLB. При совпадении в транзакции памяти используются атрибуты защиты и кэширования, указанные в дескрипторе, при этом при выполнении команды не добавляются дополнительные такты.

Так как память L1 разделена на память данных и память команд, элементы CPLB также разделяются на CPLB данных и CPLB команд. Шестнадцать элементов CPLB используются в запросах выборки команд; они называются *ICPLB*. Другие шестнадцать элементов CPLB используются в страницах данных; они называются *DCPLB*. Применение ICPLB и DCPLB разрешается установкой соответствующих битов в регистрах управления памятью команд L1 (`IMEM_CONTROL`) и памятью данных L1 (`DMEM_CONTROL`), соответственно. Эти регистры показаны на рис. 6-5 и 6-12, соответственно.

Каждый элемент CPLB состоит из пары 32-разрядных значений. В операциях выборки команд:

- `ICPLB_ADDR[n]` определяет начальный адрес страницы, описываемой дескриптором CPLB.
- `ICPLB_DATA[n]` определяет свойства страницы, описываемой дескриптором CPLB.

В операциях с данными:

- `DCPLB_ADDR[m]` определяет начальный адрес страницы, описываемой дескриптором CPLB.
- `DCPLB_DATA[m]` определяет свойства страницы, описываемой дескриптором CPLB.

Существуют два дескриптора CPLB, используемые по умолчанию для сверхоперативной памяти данных и пространства регистров ядра и системы, отображенных в карте памяти. Эти дескрипторы определяют указанное пространство памяти как недоступное для кэширования. Таким образом, для этих областей памяти нет необходимости настраивать дополнительные CPLB.

Если для этого пространства настроены достоверные CPLB, то CPLB, используемые по умолчанию, игнорируются.

Страницы памяти

4-х Гбайтное адресное пространство процессора может быть разделено на диапазоны памяти или ввода/вывода меньшего размера, называемые страницами памяти. Каждому адресу в пределах страницы соответствуют атрибуты, определенные для этой страницы. Архитектура поддерживает четыре различных размера страницы:

- 1 Кбайт;
- 4 Кбайта;
- 1 Мбайт;
- 4 Мбайта.

Использование различных размеров страниц обеспечивает гибкий механизм назначения атрибутов различным типам памяти или устройствам ввода/вывода.

Атрибуты страниц памяти

Каждая страница описывается дескриптором, состоящим из двух слов – слова дескриптора адреса, `xCPLB_ADDR[n]`, и слова дескриптора свойств, `xCPLB_DATA[n]`. Слово дескриптора адреса указывает базовый адрес страницы в памяти. Страницы должны выравниваться по адресам, значение которых делятся на размер страницы без остатка. Например, страница размером 4 Мбайта должна начинаться с адреса, делящегося на 4 Мбайта; в то же время, страница размером 1 Кбайт может начинаться с любой килобайтной границы. Второе слово дескриптора определяет другие свойства или атрибуты страницы. К ним относятся:

- Размер страницы
  1 Кбайт, 4 Кбайта, 1 Мбайт, 4 Мбайта

- Кэшируемая/некэшируемая
  Доступ к этой странице осуществляется с использованием кэша L1 или в обход кэша.

- Если страница кэшируемая: кэширование со сквозной/с отложенной записью
  Запись данных в кэш напрямую отображается в памяти или откладывается до нового выделения строки кэша. При сквозной записи строка может выделяться только при чтении или и при чтении и при записи.

- Модифицированная
  С момента последней загрузки CPLB данные в странице памяти были изменены.

- Разрешение доступа записи в режиме Супервизора.
  - Разрешает или запрещает запись в эту страницу в режиме Супервизора.
  - Используется только для страниц данных.

- Разрешение доступа записи в Пользовательском режиме.
  - Разрешает или запрещает запись в эту страницу в Пользовательском режиме.
  - Используется только для страниц данных.

- Разрешение доступа чтения в Пользовательском режиме.
  Разрешает или запрещает чтение этой страницы в Пользовательском режиме.

- Достоверность.
  Этот бит можно использовать для проверки достоверности данных CPLB.

- Запирание.
  Это свойство следует сохранить для регистров, отображенных в карте памяти; оно не участвует в политике замещения CPLB.

Таблица дескрипторов страниц

Для того, чтобы при обращениях к памяти использовался кэш, при разрешении CPLB для доступа к командам, доступа к данным или для обоих типов доступов, необходимо, чтобы пара регистров, отображенных в карте памяти, содержала достоверный элемент CPLB. Количество регистров, выделенных под элементы CPLB, ограничено 16 дескрипторами для выборок команд и 16 дескрипторами для операций загрузки и сохранения данных в память.

Для небольших и/или простых моделей памяти можно определить набор дескрипторов CPLB, вписывающихся в эти 32 элемента и покрывающих все адресуемое пространство. При этом в дальнейшем изменение дескрипторов не потребуется. Этот тип определения называется *статической* моделью управления памятью.

Однако в операционных средах зачастую определяется большее число дескрипторов CPLB, необходимое для покрытия адресуемой памяти и пространств ввода/вывода, чем количество доступных внутренних регистров CPLB, отображаемых в карте памяти. В этом случае применяется структура данных, размещаемая в памяти, которая называется таблицей дескрипторов страниц. В архитектуре процессора Blackfin не задан определённый формат таблицы дескрипторов страниц. В различных операционных системах, имеющих различные модели управления памятью, могут реализовываться структуры таблиц дескрипторов страниц, соответствующие требованиям ОС. Это свойство позволяет реализовать достигать компромисс между предпочтительным уровнем защиты и атрибутами производительности программ поддержки управления памятью.

Управление CPLB

Когда процессор Blackfin вызывает операцию с памятью, для которой отсутствует достоверный дескриптор CPLB, происходит исключение, которое переводит

процессор в режим Супервизора, для выполнения программы обработчика исключения MMU (доп. информацию см. в разделе “Исключения” глава 4). Обычно обработчик является частью ядра операционной системы (ОС), реализующей политику замещения CPLB.

Перед разрешением CPLB необходимо определить достоверные дескрипторы CPLB и для таблицы дескрипторов страниц, и для обработчика исключений MMU. Обычно в этих дескрипторах CPLB устанавливается бит `LOCK`, для запрещения их непреднамеренного замещения программой.

Адрес, вызвавший сбой, используется обработчиком в качестве индекса структуры таблицы дескрипторов страниц для поиска в ней достоверного дескриптора CPLB, который может быть загружен в одну из пар внутренних регистров CPLB. Если все внутренние регистры содержат достоверные элементы CPLB, обработчик заменяет один из дескрипторов, загружая в него информацию нового дескриптора. Перед загрузкой данных нового дескриптора в любой из PCLB необходимо запретить использование соответствующей группы из шестнадцати CPLB при помощи:

- бита разрешения DCPLB (`ENDCPLB`) в регистре `DMEM_CONTROL` для дескрипторов данных или
- бита разрешения ICPLB (`ENICPLB`) в регистре `IMEM_CONTROL` для дескрипторов команд.

Ответственность за политику и используемый алгоритм замещения CPLB возлагается на системный обработчик исключения MMU. Эта политика, диктуемая характеристиками операционной системы, обычно реализуется в виде модифицированной политики LRU, планирования с круговым замещением или псевдослучайного замещения.

После загрузки нового дескриптора CPLB выполняется возврат из обработчика исключения и повторно выполняется операция с памятью, вызвавшая сбой. При этом, для запрошенного в операции адреса будет найден достоверный дескриптор CPLB, и операция будет выполнена нормально.

При помощи одной команды может генерироваться выборка команды и один или два доступа к данным. Возможная ситуация, при которой более чем в одной из этих операций с памятью будет выполняться доступ к данным, для которых отсутствует достоверный дескриптор CPLB в паре регистров, отображенных в карте памяти. В этом случае исключения обслуживаются в соответствии с приоритетом в следующем порядке:

- Несовпадение страницы при выборке команды.
- Несовпадение страницы при операции DAG0.
- Несовпадение страницы при операции DAG1.

Применение MMU

Управление памятью является необязательным свойством архитектуры процессора Blackfin. Его применение обуславливается системными требованиями отдельного приложения. При сбросе все CPLB запрещены, и устройство управления памятью не используется.

Если вся память L1 сконфигурирована как SRAM, функции MMU для данных и команд являются необязательными и зависят от требований приложения по распределению параметров защиты пространств памяти между различными задачами или между Пользовательским режимом и режимом Супервизора. Для распределения параметров защиты памяти между задачами операционная система может поддерживать раздельные страницы памяти команд и/или данных для каждой задачи и делать определённые страницы видимыми только при выполнении конкретной задачи. При переключении между задачами операционная система должна гарантировать перевод в недостоверное состояние любых внутренних дескрипторов CPLB, которые будут недоступны в новой задаче. Операционная система также может производить предварительную загрузку дескрипторов, соответствующих новой задаче.

Во многих операционных системах прикладная программа выполняется в Пользовательском режиме, в то время как операционная система и ее службы выполняются в режиме Супервизора. Желательно защищать код и структуры данных, используемые операционной системой, от непреднамеренной модификации выполняемым в Пользовательском режиме приложением. Эта защита может быть достигнута определением дескрипторов CPLB для защищенных диапазонов памяти, разрешающих доступ записи только в режиме Супервизора. При попытках записи в защищенную область памяти в Пользовательском режиме, исключение генерируется до модификации памяти. Приложениям, работающим в Пользовательском режиме, может, по желанию, разрешаться доступ к тем структурам данных, которые могут понадобиться в приложении. Даже для функций в режиме Супервизора может блокироваться возможность записи некоторых страниц памяти, содержащих код, который не должен изменяться. Так как элементы CPLB являются регистрами, отображенными в карте памяти, запись которых возможна только в режиме Супервизора, пользовательские программы не могут получать доступ к ресурсам, защищенным подобным образом.

Если память команд L1 или память данных L1 частично или полностью сконфигурирована как кэш, необходимо разрешить использование соответствующих CPLB. Когда при разрешённом кэше команда генерирует запрос доступа к памяти, процессор сначала проверяет ICPLB, определяя, принадлежит ли запрошенный адрес диапазону адресов, доступному для кэширования. Если пара регистров, соответствующих запрошенному адресу, не содержит достоверный элемент CPLB, для получения достоверного дескриптора ICPLB и определения возможности кэширования генерируется исключение MMU. В результате, если разрешена работа памяти команд L1 в качестве кэша, для любой области памяти, содержащей команды, должен быть определен достоверный дескриптор ICPLB. Эти дескрипторы должны либо постоянно присутствовать в регистрах, отображенных в карте памяти, либо располагаться в таблице

дескрипторов страниц, управляемой обработчиком исключений MMU. Аналогично, если один или оба банка данных L1 сконфигурированы как кэш, все возможные диапазоны памяти данных должны поддерживаться дескрипторами DCPLB.

Перед разрешением кэша следует настроить и разрешить использование MMU и соответствующих структур данных.

Примеры защищенных областей памяти

На рис. 6-19 приведен пример распределения CPLB данных и команд. Следует отметить, что некоторые ICPLB и DCPLB имеют общие дескрипторы, соответствующие одному адресному пространству.

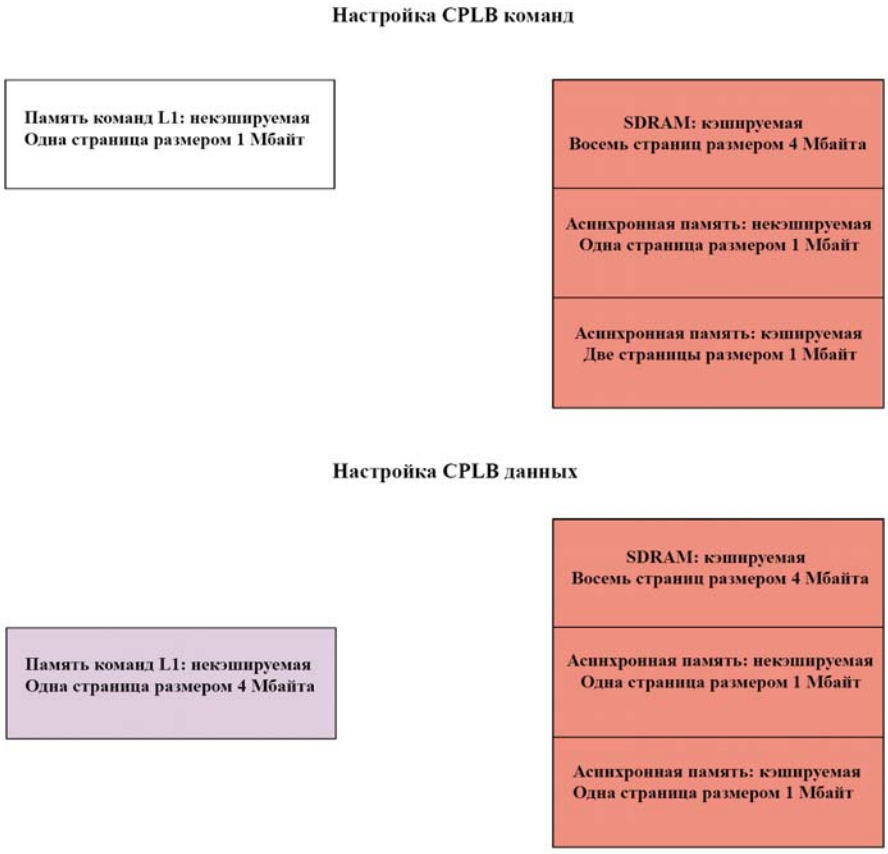


Рис. 6-19. Примеры защищенных областей памяти.

Регистры данных ICPLB (ICPLB_DATAx)

На рис. 6-20 показаны регистры данных ICPLB.

Для гарантии корректной работы и совместимости с будущими проектами при каждой записи этих регистров все зарезервированные биты должны устанавливаться в 0.

Регистры данных ICPLB (ICPLB_DATAx)

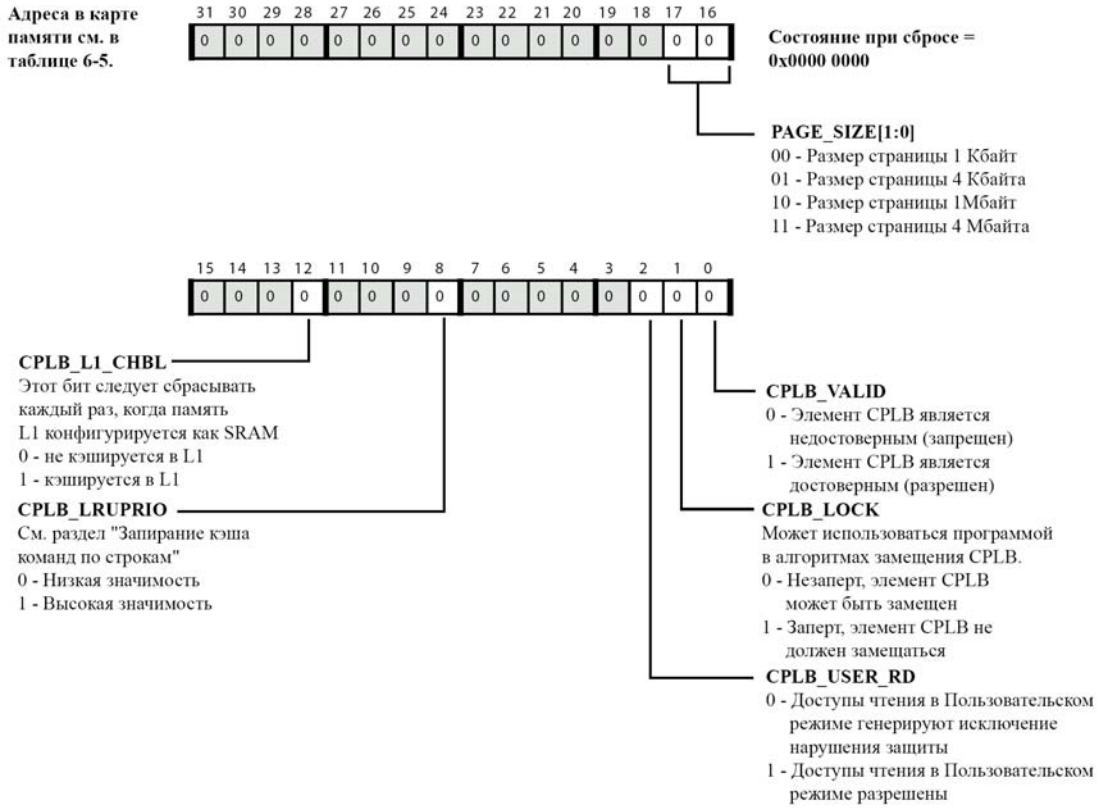


Рис. 6-20. Регистры данных ICPLB.

Таблица 6-5. Адреса регистров данных ICPLB в карте памяти.

Название регистра	Адрес в карте памяти
ICPLB_DATA0	0xFFE0 1200
ICPLB_DATA1	0xFFE0 1204
ICPLB_DATA2	0xFFE0 1208
ICPLB_DATA3	0xFFE0 120C
ICPLB_DATA4	0xFFE0 1210
ICPLB_DATA5	0xFFE0 1214
ICPLB_DATA6	0xFFE0 1218
ICPLB_DATA7	0xFFE0 121C
ICPLB_DATA8	0xFFE0 1220
ICPLB_DATA9	0xFFE0 1224
ICPLB_DATA10	0xFFE0 1228
ICPLB_DATA11	0xFFE0 122C
ICPLB_DATA12	0xFFE0 1230
ICPLB_DATA13	0xFFE0 1234
ICPLB_DATA14	0xFFE0 1238
ICPLB_DATA15	0xFFE0 123C

Регистры данных DCPLB (DCPLB_DATAx)

На рис. 6-21 показаны регистры данных DCPLB.

Для гарантии корректной работы и совместимости с будущими проектами при каждой записи этих регистров все зарезервированные биты должны устанавливаться в 0.

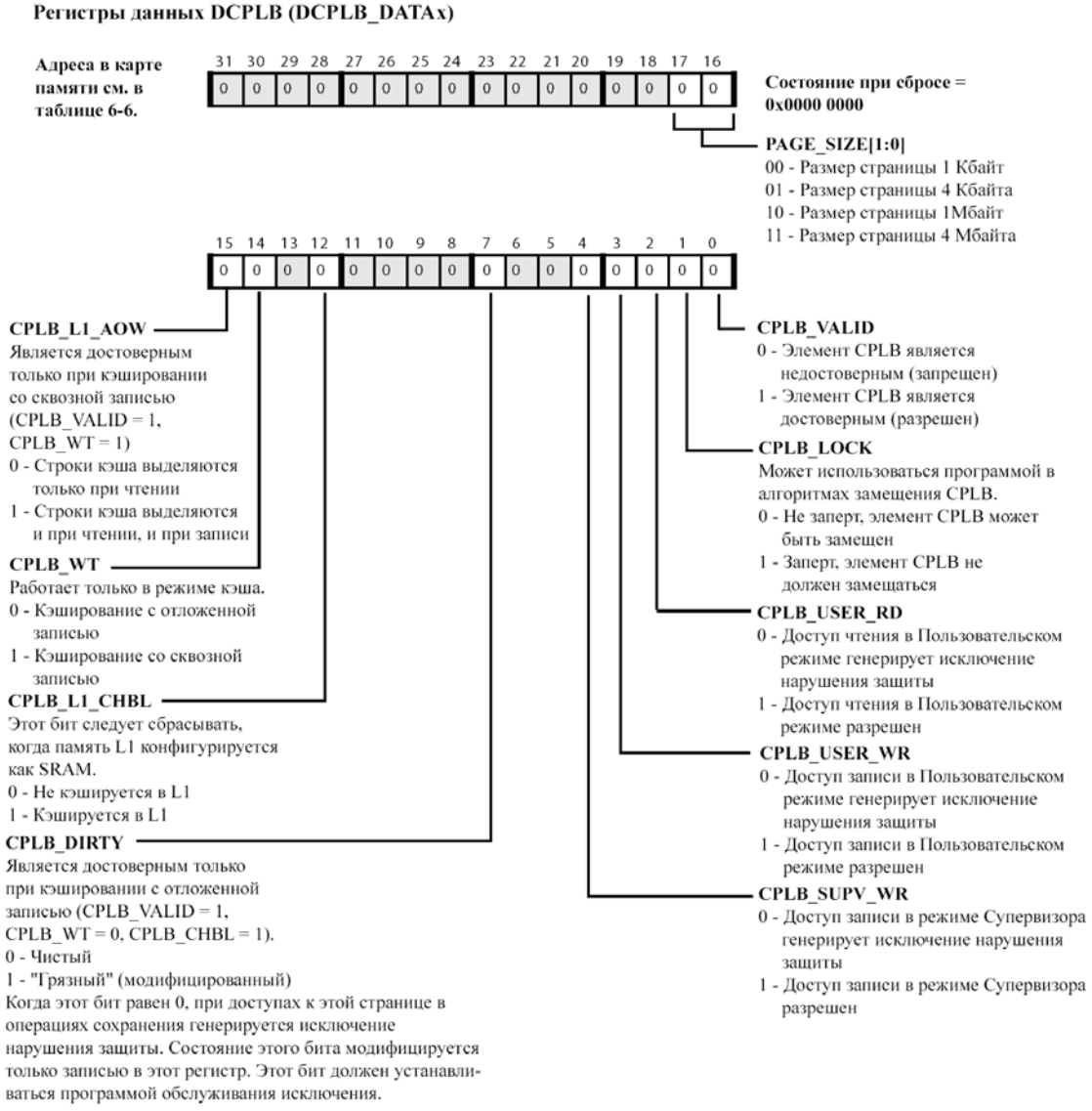


Рис. 6-21. Регистры данных DCPLB.

Таблица 6-6. Адреса регистров данных DCPLB в карте памяти.

Название регистра	Адрес в карте памяти
DCPLB_DATA0	0xFFE0 0200
DCPLB_DATA1	0xFFE0 0204
DCPLB_DATA2	0xFFE0 0208
DCPLB_DATA3	0xFFE0 020C
DCPLB_DATA4	0xFFE0 0210
DCPLB_DATA5	0xFFE0 0214
DCPLB_DATA6	0xFFE0 0218

DCPLB_DATA7	0xFFE0 021C
DCPLB_DATA8	0xFFE0 0220
DCPLB_DATA9	0xFFE0 0224
DCPLB_DATA10	0xFFE0 0228
DCPLB_DATA11	0xFFE0 022C
DCPLB_DATA12	0xFFE0 0230
DCPLB_DATA13	0xFFE0 0234
DCPLB_DATA14	0xFFE0 0238
DCPLB_DATA15	0xFFE0 023C

Регистры адреса DCPLB (DCPLB_ADDRx)

На рис. 6-22 показаны регистры адреса DCPLB.

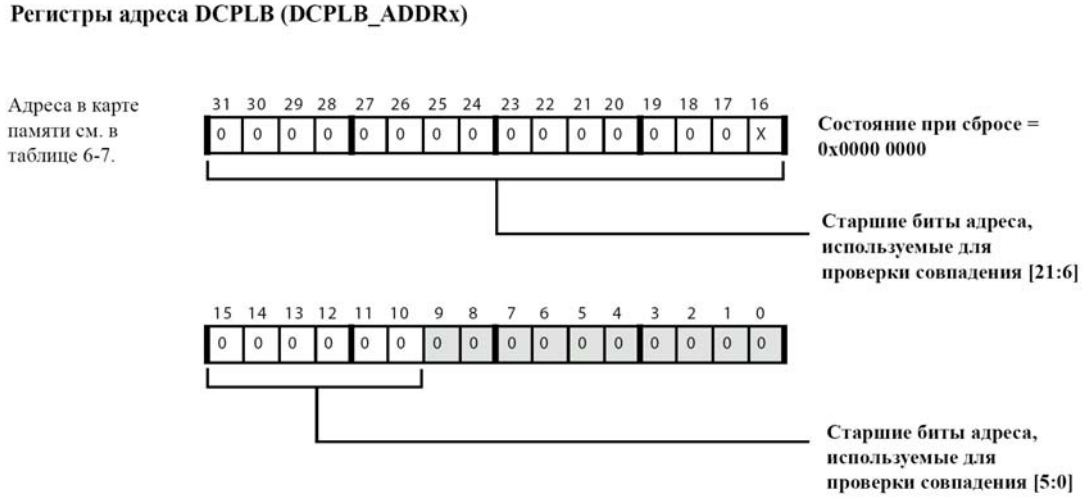


Рис. 6-22. Регистры адреса DCPLB.

Таблица 6-7. Адреса регистров адреса DCPLB в карте памяти.

Название регистра	Адрес в карте памяти
DCPLB_ADDR0	0xFFE0 0100
DCPLB_ADDR1	0xFFE0 0104
DCPLB_ADDR2	0xFFE0 0108
DCPLB_ADDR3	0xFFE0 010C
DCPLB_ADDR4	0xFFE0 0110
DCPLB_ADDR5	0xFFE0 0114
DCPLB_ADDR6	0xFFE0 0118
DCPLB_ADDR7	0xFFE0 011C
DCPLB_ADDR8	0xFFE0 0120
DCPLB_ADDR9	0xFFE0 0124
DCPLB_ADDR10	0xFFE0 0128
DCPLB_ADDR11	0xFFE0 012C
DCPLB_ADDR12	0xFFE0 0130
DCPLB_ADDR13	0xFFE0 0134
DCPLB_ADDR14	0xFFE0 0138
DCPLB_ADDR15	0xFFE0 013C

Регистры адреса ICPLB (ICPLB_ADDRx)

На рис. 6-23 показаны регистры адреса ICPLB.

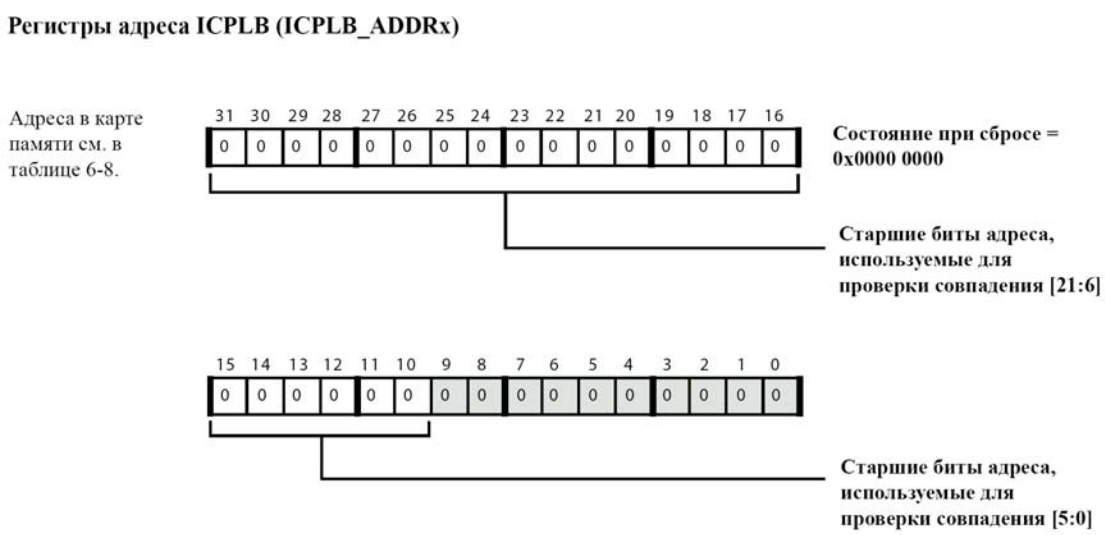


Рис. 6-23. Регистры адреса ICPLB.

Таблица 6-8. Адреса регистров адреса ICPLB в карте памяти.

Название регистра	Адрес в карте памяти
ICPLB_ADDR	0xFFE0 1100
ICPLB_ADDR	0xFFE0 1104
ICPLB_ADDR	0xFFE0 1108
ICPLB_ADDR	0xFFE0 110C
ICPLB_ADDR	0xFFE0 1110
ICPLB_ADDR	0xFFE0 1114
ICPLB_ADDR	0xFFE0 1118
ICPLB_ADDR	0xFFE0 111C
ICPLB_ADDR	0xFFE0 1120
ICPLB_ADDR	0xFFE0 1124
ICPLB_ADDR	0xFFE0 1128
ICPLB_ADDR	0xFFE0 112C
ICPLB_ADDR	0xFFE0 1130
ICPLB_ADDR	0xFFE0 1134
ICPLB_ADDR	0xFFE0 1138
ICPLB_ADDR	0xFFE0 113C

Регистры состояния ICPLB и DCPLB (DCPLB_STATUS, ICPLB_STATUS)

Биты в регистре состояния DCPLB (`DCPLB_STATUS`) и регистре состояния ICPLB (`ICPLB_STATUS`) идентифицируют элемент CPLB, вызвавший исключение CPLB. Программа обслуживания исключения может определить источник сбоя, исследуя элементы CPLB.

Регистры `DCPLB_STATUS` и `ICPLB_STATUS` являются достоверными только при выполнении программы обслуживания исключения, вызванного сбоем.

Биты FAULT_DAG, FAULT_USERSUPV и FAULT_RW регистра состояния DCPLB (DCPLB_STATUS) используются для идентификации элемента CPLB, вызвавшего исключение CPLB (см. рис. 6-24).

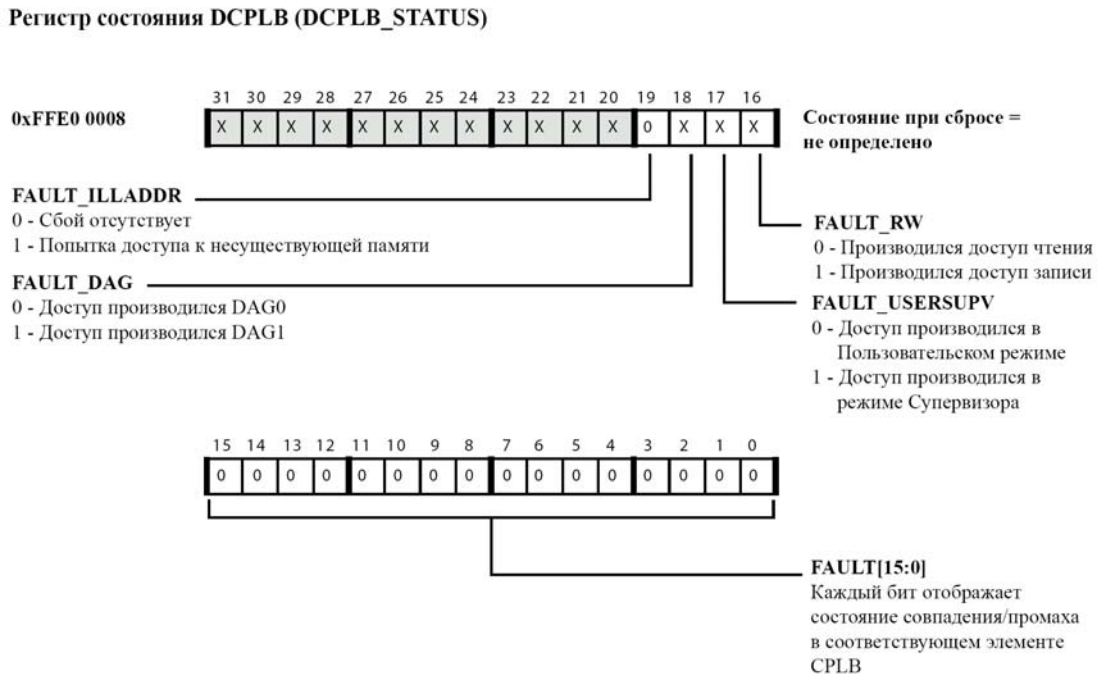


Рис. 6-24. Регистр состояния DCPLB.

Бит FAULT_USERSUPV регистра состояния ICPLB (ICPLB_STATUS) используется для идентификации элемента CPLB, вызвавшего исключение CPLB (см. рис. 6-25).

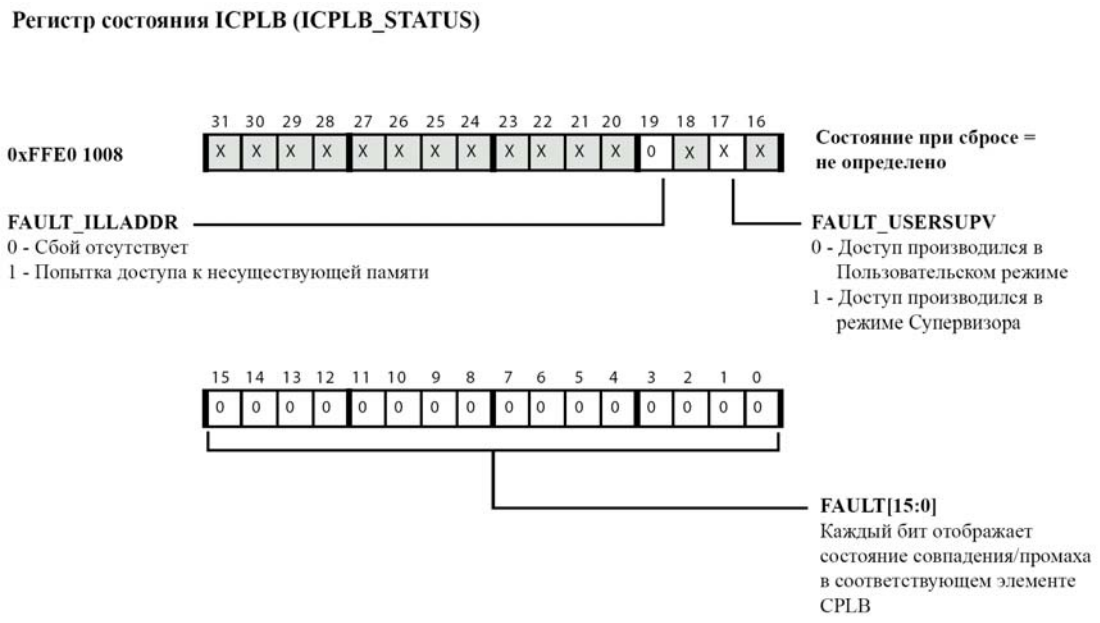


Рис. 6-25. Регистр состояния ICPLB.

Регистры ошибочного адреса DCPLB и ICPLB (DCPLB_FAULT_ADDR, ICPLB_FAULT_ADDR)

Регистр ошибочного адреса DCPLB (`DCPLB_FAULT_ADDR`) и регистр ошибочного адреса ICPLB (`ICPLB_FAULT_ADDR`) содержат адрес, вызвавший сбой в памяти данных L1 и памяти команд L1, соответственно. См. рис. 6-26 и рис. 6-27.

Регистры `DCPLB_FAULT_ADDR` и `ICPLB_FAULT_ADDR` достоверны только при выполнении программы обслуживания исключения, вызванного сбоем.

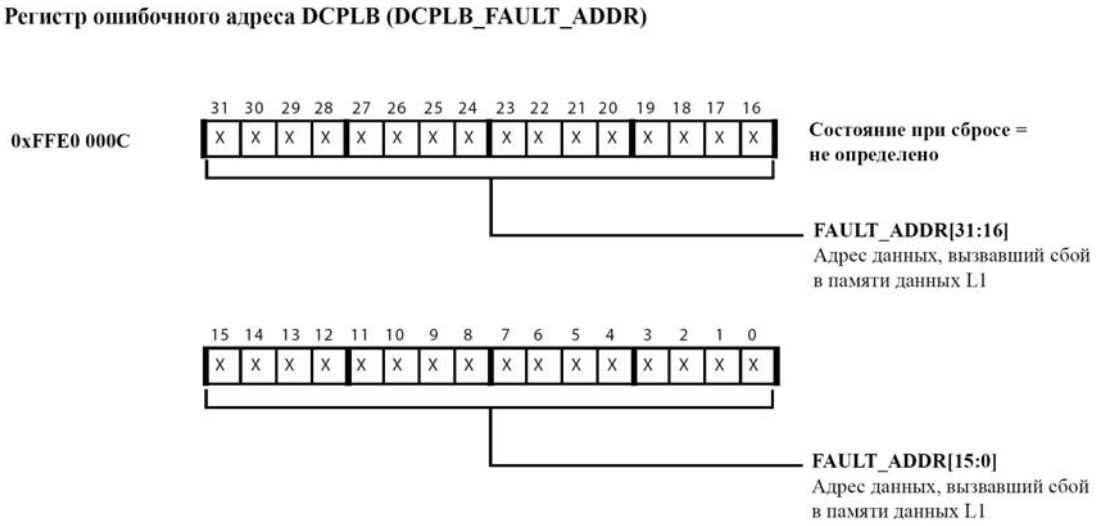


Рис. 6-26. Регистр ошибочного адреса DCPLB.

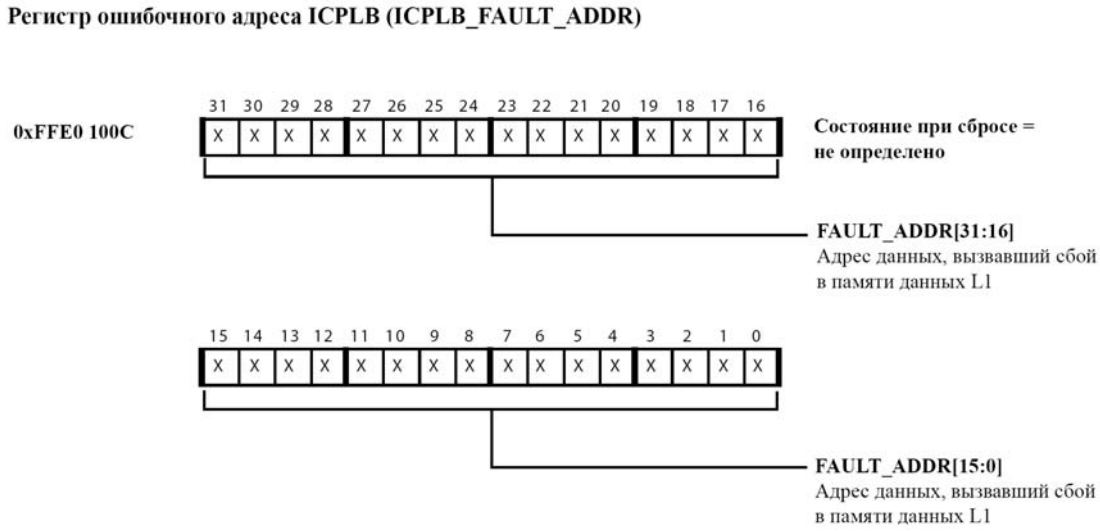


Рис. 6-27. Регистр ошибочного адреса ICPLB.

Модель транзакций памяти

Доступ к ячейкам внутренней и внешней памяти осуществляется в формате с порядком следования байтов, начиная с младшего. На рис. 6-28 показано слово данных, хранящееся в регистре `R0` и в ячейке памяти по адресу *addr*. B0 обозначает младший байт 32-разрядного слова.

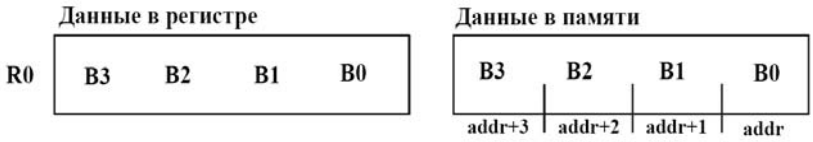


Рис. 6-28. Хранение данных в формате с порядком следования байтов, начиная с младшего.

На рис. 6-29 показаны 16- и 32-разрядные команды, хранящиеся в памяти. Слева на рисунке показана 16-разрядная команда, хранящаяся в памяти. Старший байт команды хранится по старшему адресу (байт B1 по адресу *addr+1*), а младший байт – по младшему адресу (байт B0 по адресу *addr*).

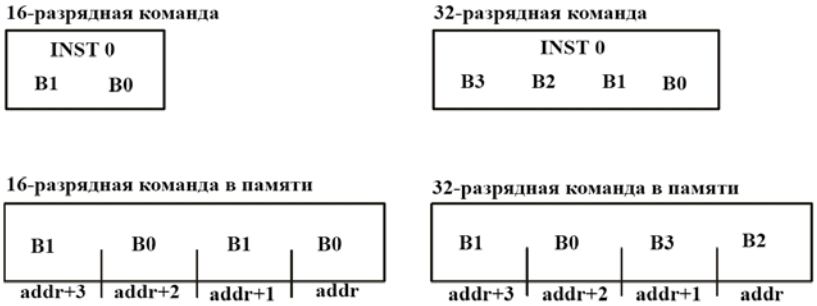


Рис. 6-29. Хранение команд в формате с порядком следования байтов, начиная с младшего.

Справа на рисунке показана 32-разрядная команда, хранящаяся в памяти. Следует отметить, что старшее 16-разрядное полуслово команды (байты B3 и B2) хранятся по младшим адресам (*addr+1* и *addr*), а младшее полуслово (байты B1 и B0) хранятся по старшим адресам (*addr+3* и *addr+2*).

Выполнение операций загрузки/сохранения

Архитектура процессора Blackfin поддерживает RISC-концепцию механизма загрузки/сохранения (load/store). Этот механизм является характеристикой RISC-архитектуры, в которой операции с памятью (операции загрузки и сохранения) намеренно отделены от арифметических функций, использующих продукты операций с памятью. Это разделение происходит вследствие того, что операции с памятью, в частности команды доступа к внешней памяти или устройствам ввода/вывода, обычно занимают много тактов и привели бы к остановам

процессора, что не позволило бы достичь скорости вычисления, равной одной команде за такт.

Отделение операций загрузки от связанных с ними арифметических функций позволяет компиляторам или программистам, пишущим программы на языке ассемблера, помещать между командой загрузки и зависящими от неё командами команды, не связанные с результатами загрузки. Команды, не связанные с результатами загрузки, выполняются параллельно с ожиданием возврата данных системой памяти. Если значение возвращается до того, как зависящая от загрузки операция достигает уровня исполнения (EX) конвейера, операция, не связанная с загрузкой, выполняется за один такт.

При выполнении операций записи команда сохранения считается завершенной сразу после выполнения, несмотря на то, что до действительного момента записи данных во внешнюю память или в устройство ввода/вывода может пройти много тактов. Это упорядочение позволяет процессору выполнять одну команду за такт, при этом синхронизация между завершением записей и выполнением последующих команд не гарантируется. Более того, в контексте большинства операций с памятью эта синхронизация полагается несущественной.

Замкнутый конвейер

В архитектуре процессора Blackfin реализован замкнутый конвейер выполнения команд. При выполнении команды загрузки, регистр-приемник операций чтения помечается как занятый до момента, пока система памяти не возвратит значение. Если последующая команда пытается обратиться к регистру до того, как становится доступно новое значение, будет произведен останов конвейера до момента завершения операции с памятью. Этим гарантируется, что команды, которые требуют использования данных, являющихся результатом загрузки, не будут использовать предыдущие или недостоверные данные, несмотря на то, что возможно начало выполнения команд до завершения чтения памяти.

Этот механизм позволяет выполнять независимые команды между загрузкой и командами, использующими результат чтения; при этом от программиста или компилятора не требуется знание действительного количества тактов, необходимого для завершения операции чтения из памяти. Если команда, следующая непосредственно за командой загрузки, использует тот же регистр, просто добавляются остановы до возврата значения. Таким образом, поведение команды совпадает с ожиданиями программиста. Однако, если после команды загрузки, но до команды, использующей тот же регистр, помещаются другие четыре команды, они будут выполнены, и повысится общая производительность процессора.

Упорядочение операций загрузки и сохранения

Снижение требований к синхронизации между командами доступа к памяти и смежными с ними командами называется слабым упорядочением операций загрузки и сохранения. Слабое упорядочение подразумевает, что действительные

времена завершения операций с памятью, – и даже порядок возникновения этих событий, – может не совпадать с порядком их следования в коде исходной программы. Гарантируется только следующее:

- Операции загрузки будут завершены до того, как возвращаемые данные будут использованы в последующей команде.
- В операциях загрузки, использующих ранее записанные данные, будут использованы обновлённые значения.
- Операции сохранения в конечном итоге распространятся на приёмник данных.

При применении слабого упорядочения допускается приоритетное выполнение операций чтения перед выполнением операций записи системой памяти. В данном случае, выполнение операции записи, находящейся на любом уровне конвейера, может быть отложено последующей операцией чтения, допуская завершение чтения до завершения записи. Операции чтения имеют приоритет перед операциями записи, так как от результата операции чтения может зависеть другая операция, которая ожидает её завершения. В то же время, если выполнение операции записи занимает больше циклов, чем требуется, процессор полагает запись завершённой и не выполняет останов конвейера. Такое поведение может привести к ситуации, при которой в действительности в процессе выполнения программы операция чтения, следующая в исходном коде после операции записи, возвратит значение раньше, чем завершится запись. Это упорядочение обеспечивает значительный выигрыш в производительности при выполнении большинства команд работы с памятью. Однако оно может вызывать побочные эффекты, которые должны учитываться программистом во избежание некорректной работы системы.

При выполнении записи в область, не являющуюся физической памятью, (например, в регистры устройства ввода/вывода) или чтении из неё, порядок завершения операций чтения и записи зачастую имеет значение. Например, чтение регистра состояния может зависеть от записи в регистр управления. Если адреса совпадают, при чтении может возвращаться значение из буфера записи, а не из регистра устройства ввода/вывода; при этом возможно обращение порядка операций чтения и записи в регистр. Оба эти эффекта могут вызывать нежелательные побочные явления во взаимодействии программы с периферийными устройствами. Для того, чтобы гарантировать отсутствие подобных эффектов в программе, требующей точного (жёсткого) упорядочения операций загрузки и сохранения, следует использовать команды синхронизации (`CSYNC` или `SSYNC`).

Команды синхронизации

При требовании жёсткого упорядочения операций загрузки и сохранения, которое может возникать в случае последовательных записей в устройство ввода/вывода с целью его настройки и управления, используйте команды синхронизации ядра или системы, `CSYNC` или `SSYNC`, соответственно.

Команда `CSYNC` гарантирует завершение всех ожидающих завершения операций ядра и очистку буфера ядра (между ядром процессора и блоками памяти L1) до

перехода к выполнению следующей команды. Ожидающие завершения операции ядра могут включать любые прерывания, ожидающие обслуживания, спекулятивные состояния (такие, как предсказания переходов) или исключения.

Рассмотрим следующий фрагмент программы:

```
IF CC JUMP away_from_here
csync;
r0 = [p0];
away_from_here:
```

В этом примере применение команды `CSYNC` гарантирует следующее:

- Вычисляется условный переход (`IF CC JUMP away_from_here`), при этом в конвейере команд добавляются циклы останова до тех пор, пока не будет оценено условие и не будут сброшены все элементы в буфере сохранения процессора.
- Завершается обработка всех прерываний и исключений, которые обслуживались до вызова `CSYNC`.
- Спекулятивная (опережающая) операция загрузки из памяти не выполняется.

Команда `SSYNC` гарантирует, завершение всех предыдущих операций в интерфейсе между блоками памяти L1 и остальной частью кристалла. В дополнение к выполнению функций синхронизации ядра командой `CSYNC`, команда `SSYNC` сбрасывает любые буферы записи между памятью L1 и доменом системы и генерирует запрос синхронизации, требующий подтверждения для завершения выполнения `SSYNC`.

Спекулятивное выполнение загрузки

Операции загрузки из памяти не изменяют состояния значения в памяти. Следовательно, вызов спекулятивной (опережающей) операции чтения из памяти для последующей команды загрузки обычно не имеет нежелательных побочных эффектов. В некоторых случаях, например, при загрузке, следующей за командой условного перехода, можно повысить производительность, вызывая запрос чтения перед оценкой условного перехода. Например:

```
IF CC JUMP away_from_here
R0 = [ P2 ] ;
…
away_from_here:
```

Если переход выполняется, команда загрузки удаляется из конвейера, и любые, возвращаемые результаты могут игнорироваться. И, наоборот, если переход не выполняется, возврат корректного значения из памяти произойдет раньше, чем, если бы был произведен останов до выполнения оценки перехода.

Однако, при работе с устройствами ввода/вывода, это может вызвать нежелательные побочные эффекты для периферии, возвращающей последовательности данных из FIFO или из регистра, изменяющего значение в

зависимости от числа запрошенных операций чтения. Для устранения этих эффектов и гарантии корректной работы между операциями чтения следует использовать команды синхронизации (`CSYNC` или `SSYNC`).

В операциях сохранения спекулятивный доступ к памяти никогда не выполняется, так как это может повлечь модификацию значения в памяти до того, как будет принято решение о необходимости выполнения команды.

Условное выполнение загрузки

Команды синхронизации вызывают принудительное завершение всех спекулятивных операций перед инициацией обращения к памяти командой загрузки. Однако, сама команда загрузки может генерировать несколько операций чтения из памяти, так как она является прерываемой. Если между завершением выполнения команды синхронизации и завершением выполнения команды загрузки возникает прерывание достаточного приоритета, программный автомат отменяет выполнение команды загрузки. После выполнения прерывания прерванная загрузка выполняется заново. Этот подход минимизирует задержку обслуживания прерывания. Однако, возможна ситуация, когда до отмены загрузки был инициирован цикл чтения из памяти; при этом после возобновления выполнения загрузки последует вторая операция чтения. В большинстве обращений к памяти многократное чтение из одного адреса памяти не влечет побочных эффектов. Но для некоторых устройств, отображаемых в карте памяти (например, периферийных FIFO данных), операции чтения являются деструктивными. При каждом чтении из такого устройства данные в FIFO продвигаются и не могут быть восстановлены и прочитаны повторно.

При доступе к отображаемым в карте памяти устройствам, состояние которых зависит от числа операций записи или чтения по определенному адресу, перед выполнением операции загрузки или сохранения следует запретить прерывания.

Работа с памятью

Этот раздел содержит информацию о выравнивании данных в памяти и операциях с памятью, поддерживающих взаимодействие задач с использованием семафоров. Он также включает краткое обслуживание регистров, отображенных в карте памяти, и пример программирования регистров ядра, отображенных в карте памяти.

Выравнивание

Операции с невыровненной памятью явно не поддерживаются. Невыровненные обращения к памяти генерируют исключение невыровненного доступа (см. раздел “Исключения” в главе 4). Однако, так как некоторые потоки данных (например, 8-разрядные видеоданные) могут намеренно быть не выровнены в памяти, возможно

запрещение исключений выравнивания при помощи команды DISALGNEXPCT. Более того, некоторые команды, работающие с группами, которые состоят из четырёх 8-разрядных данных, автоматически запрещают исключения выравнивания.

Когерентность кэша

При необходимости программа должна обеспечивать поддержку когерентности кэша для разделяемых данных. Для этого используется команда FLUSH (см. раздел «Команды управления кэшем команд») и/или явный перевод строк кэша в недостоверное состояние при помощи регистров ядра, отображенных в карте памяти (см. раздел «Регистры проверки данных»).

Элементарные операции

Процессор поддерживает одну элементарную операцию – TESTSET. Элементарные операции используются для организации непрерываемых операций с памятью в целях поддержки взаимодействия между задачами при помощи семафоров. Команда TESTSET загружает половину слова из косвенно адресуемой памяти, проверяет, равен ли младший байт нулю, и затем устанавливает старший бит младшего байта, не воздействуя на остальные биты. Если байт изначально равен нулю, устанавливается бит CC. Если байт не равен нулю, бит CC сбрасывается. Последовательность этой транзакции памяти является элементарной (неделимой) – аппаратный захват шины гарантирует, что между операциями проверки и установки этой команды не возникнут другие операции с памятью. Команда TESTSET может прерываться ядром. В этом случае при возврате из прерывания команда TESTSET выполняется заново.

Команда TESTSET может адресовать все 4-х гигабайтное адресное пространство, но не должна обращаться к памяти ядра (L1 или пространству регистров, отображенных в карте памяти), так как элементарные доступы к этой области памяти не поддерживаются.

Архитектура памяти всегда обрабатывает элементарные операции как доступы с запретом кэширования, даже если дескриптор CPLB, соответствующий адресу, указывает на доступ с возможностью кэширования. Выполнение операции TESTSET над областями памяти, доступными для кэширования, не рекомендуется, так как архитектура процессора не может гарантировать когерентность кэшируемой ячейки памяти при выполнении команды TESTSET.

Регистры, отображенные в карте памяти

Зарезервированное пространство регистров, отображенных в карте памяти, расположено в старшей части пространства памяти (0xFFC0 0000). Эта область определена как недоступная для кэширования и разделяется регистрами системы

(0xFFC0 0000 – 0xFFE0 0000) и ядра (0xFFE0 0000 – 0xFFFF FFFF), отображенными в карте памяти.

Если требуется жесткое упорядочение, после сохранения в регистр, отображенный в памяти, следует добавить команду синхронизации. Дополнительную информацию см. в разделе “Операции загрузки/сохранения в память”.

Все регистры, отображенные в карте памяти, доступны только в режиме Супервизора. Доступ к ним в Пользовательском режиме генерирует исключение нарушения защиты.

Запись и чтение всех регистров ядра, отображенных в карте памяти, осуществляется при помощи 32-разрядных доступов. Однако в некоторых регистрах может быть задействовано менее 32-х битов. В этом случае незадействованные биты зарезервированы. Регистры системы, отображенные в карте памяти, могут быть 16-разрядными.

Доступ к несуществующим регистрам генерирует исключение некорректного доступа. Система игнорирует записи в регистры, доступные только для чтения.

Перечень всех регистров ядра, отображенных в карте памяти, представлен в приложении A. В приложении B представлен перечень регистров системы, отображенных в карте памяти.

Пример программирования регистров ядра, отображенных в карте памяти

Доступ к регистрам ядра, отображенным в карте памяти, возможен только как к выровненным 32-разрядным словам. Невыровненные доступы генерируют исключение. В листинге 6-1 представлены команды, необходимые для манипуляции регистрами ядра, отображенными в карте памяти.

Листинг 6-1. Программирование регистров ядра, отображенных в карте памяти.

```
CLI R0;   /* Запрещение прерываний и сохранение IMASK */
P0 = MMR_BASE;   /* 32-разрядная команда загрузки базового
адреса регистров, отображенных в карте памяти */
R1 = [P0 + TIMER_CONTROL_REG];    /* Получение значения
регистра управления */
BITSET R1, #N;   /* Установка бита N */
[P0 + TIMER_CONTROL_REG] = R1;        /* Сохранение в
управляющий регистр */
CSYNC;     /* Гарантирует выполнение записи в регистр */
STI R0;     /* Разрешение прерываний */
```


Команда `CLI` сохраняет содержимое регистра `IMASK` и сбрасывает его, запрещая прерывания. Команда `STI` восстанавливает содержимое

регистра `IMASK`, разрешая прерывания. Выполнение команд, расположенных между `CLI` и `STI`, не должно прерываться.

Терминология

Для описания памяти используется следующая терминология.

блок кэша. Наименьшая единица памяти, передаваемая на следующий уровень/со следующего уровня памяти из кэша/в кэш в результате промаха в кэше.

совпадение в кэше. Обращение к памяти, которому соответствует достоверный элемент в кэше.

строка кэша. То же, что и блок кэша. В этой главе для обозначения блоков кэша используется термин “строка кэша”.

промах в кэше. Обращение к памяти, которое не совпадает ни с одним из достоверных элементов в кэше.

кэш с прямым отображением. Архитектура кэша, в которой каждая строка может присутствовать в кэше только в одной позиции. Также называется одновходовым ассоциативным кэшем.

“грязный” или модифицированный. Бит состояния, хранящийся совместно с тегом. Указывает, изменялись ли данные в строке кэша данных с момента их копирования из памяти-источника (требуется ли их обновление в памяти-источнике).

исключительная, чистая. Состояние строки кэша, указывающее на её достоверность и совпадение данных, содержащихся в ней, с данными в памяти-источнике. При замещении данных, содержащихся в чистой строке кэша, их можно не копировать в память-источник.

полностью ассоциативный. Архитектура кэша, в которой каждая строка может размещаться в любой позиции.

индекс. Часть адреса, используемая для выбора элемента массива (например, индекс строки).

алгоритм LRU (замещения строки, не использованной дольше всех). Алгоритм замещения, используемый кэшем, в котором, в первую очередь, замещается строка, которая не использовалась дольше остальных.

память 1-го уровня (L1). Память, доступная ядру напрямую без использования промежуточных подсистем памяти.

little endian (начиная с младшего). Родной формат хранения данных в процессоре Blackfin. Младший байт слова содержится в ячейке памяти с младшим адресом, старший байт – в ячейке со старшим адресом.

политика замещения. Функция, используемая процессором для определния строки, которую следует заменить при промахе в кэше. Она зачастую реализует алгоритм LRU.

набор. Группа N-строковых элементов во входах N-входового кэша, выбираемая полем `INDEX` адреса (см. рис. 6-7).

наборно-ассоциативный. Архитектура кэша, в которой размещение строк ограничено определённым количеством наборов (или входов).

тег. Старшие биты адреса, хранящиеся совместно с кэшированной строкой данных и используемые для идентификации адреса определённого источника в памяти, представленного этой строкой.

бит достоверности. Бит состояния, хранящийся совместно с тегом и указывающий на то, что соответствующие тег и данные действительны и правильны и могут использоваться для удовлетворения обращений к памяти.

жертва. Модифицированная строка кэша, которая должна быть записана в память до её замещения в целях высвобождения памяти при выделении строки.

вход. Массив строковых элементов в N-входовом кэше. (см. рис. 6-7)

отложенная запись. Политика записи кэша, также известная как отложенное копирование (*copyback*). При записи данные записываются только в строку кэша. Модифицированная строка кэша записывается в исходную память только при её замещении. Выделение строк кэша осуществляется и при чтении и при записи.

сквозная запись. Политика записи кэша, также известная как сквозное сохранение. При записи данные записываются и в строку кэша, и в память-источник. Модифицированная строка *не* записывается в память-источник при замещении. Строки кэша должны выделяться при операциях чтения и могут выделяться при операциях записи.