9. Cache and Tightly-Coupled Memory

NII52007-10.1.0

9. Кэш и прочно сопряжённая память

Введение

Ядро процессора Nios® II может иметь кэш инструкций и данных. В этой главе обсуждаются поведение, связанное с кэшем, которое вам нужно учитывать, чтобы гарантировать корректную работу вашей программы в процессоре Nios II. К счастью большинство программ, созданных на базе слоя аппаратной абстракции (HAL) Nios II, работают корректно без специальной организации места под кэш. Однако, некоторым программам нужно задавать кэш напрямую.

Для кода, который напрямую контролирует кэш, архитектура Nios II предоставляет средства для выполнения следующих действий:

- Инициализация строк в кэше инструкций и данных,
- Строки сброса (Flush lines) в кэше инструкций и данных,
- Пропуск кэша данных во время загрузки и сохранения инструкций,

Эта глава состоит из следующих секций:

Реализация кэша Nios II

В зависимости от реализации ядра Nios II, процессорная система Nios II может или не может иметь кэша инструкция или кэша данных. Вы можете написать программы общим методом, так чтобы функции программ корректно работали на любых процессорах Nios II, независимо от того, имеется или нет кэш память. Для ядра Nios II без одного или обоих кэшей, операции управления кэшем не имеют эффекта.

В существующих ядрах Nios II нет аппаратного механизма когерентности кэша. Однако если несколько мастеров имеют доступ к общей памяти, программа должна однозначно определять когерентность между всеми мастерами.

За подробной информацией о средствах каждой реализации ядра Nios II, обратитесь к главе "Подробности реализации ядра Nios II" в настольной книге по процессору Nios II.

Подробности конкретной процессорной системы Nios II определены в файле **system.h**. В примере 9-1 показан фрагмент из файла **system.h**, определяющий свойства кэша, такие как размер кэша и размер одной строки кэша.

Example 9–1. An Excerpt from system.h that Defines the Cache Structure

```
#define NIOS2_ICACHE_SIZE 4096
#define NIOS2_DCACHE_SIZE 0
#define NIOS2_ICACHE_LINE_SIZE 32
#define NIOS2_DCACHE_LINE_SIZE 0
```

Эта система имеет 4 килобайта (кб) кэша инструкций с 32 байтами в строке и не имеет кэша данных.

Функции HAL API управления кэшем

Интерфейс программного приложения (API) HAL предоставляет следующие функции для управления кэш памятью:

- alt_dcache_flush()
- alt_dcache_flush_all()
- alt_icache_flush()
- alt_icache_flush_all()
- alt_uncached_malloc()
- alt_uncached_free()
- alt_remap_uncached()
- alt_remap_cached()

За подробной информацией об API функциях, обратитесь к главе "Справка по HAL API" в настольной книге программиста Nios II.

Дополнительная информация

Эта глава охватывает только средства управления кэшем, которые используются программистами под Nios II. Здесь нет фундаментальных операций над кэшем. Обратитесь к книге "Кэш память" Jim Handy за описанием основных средств управления кэшем.

Инициализация кэша после сброса

После сброса содержимое кэша инструкций и кэша данных неизвестно. Они должны быть проинициализированы при запуске программы обработки сброса для последующей корректной работы.

Кэш Nios II не может быть запрещён программой; он всегда разрешён. Чтобы выполнить соответствующую операцию, сброс процессора принуждает кэш инструкций признать недействительной одну строку кэша инструкций, которая связана с адресом сброса процессора. Это вынуждает кэш инструкций забирать инструкции, относящиеся к этой строке кэша, из памяти. Адрес обработчика сброса должен быть выровнен по размеру строки кэша.

Обязанность первых восьми инструкций обработчика сброса – это инициализация остатка кэша инструкций. Nios II инструкция **initi** инициализирует одну строку кэша инструкций. Не используйте инструкцию **flushi,** поскольку она даст наилучший результат, когда будет использована для инициализации кэша инструкций в будущий версиях реализации Nios II.

Поместите инструкцию **initi** в цикле, который исполняет **initi** для каждого адреса строки кэша инструкций. В примере 9-2 показан пример ассемблерного кода инициализации кэша инструкций.

Example 9–2. Assembly Code to Initialize the Instruction Cache

```
    mov     r4, r0
    movhi   r5, %hi(NIOS2_ICACHE_SIZE)
    ori     r5, r5, %lo(NIOS2_ICACHE_SIZE)
icache_init_loop:
    initi   r4
    addi    r4, r4, NIOS2_ICACHE_LINE_SIZE
    bltu    r4, r5, icache_init_loop
```

После завершения инициализации кэша инструкций, необходимо проинициализировать кэш данных. Nios II инструкция **initd** инициализирует одну строку кэша данных. Не используйте инструкцию **flushd,** поскольку она записывает испорченные строки обратно в память. Кэш данных неопределённый после сброса, включая дескрипторы строк кэша. Использование **flushd** может вызвать неожиданную запись случайных данных по случайным адресам. Инструкция **initd** не записывает обратно в память испорченные данные.

Поместите инструкцию **initd** в цикле, который исполняет **initd** для каждого адреса строки кэша данных. В примере 9-3 показан пример ассемблерного кода инициализации кэша данных.

Example 9–3. Assembly Code to Initialize the Data Cache

```
    mov     r4, r0
    movhi   r5, %hi(NIOS2_DCACHE_SIZE)
    ori     r5, r5, %lo(NIOS2_DCACHE_SIZE)
dcache_init_loop:
    initd   0(r4)
    addi    r4, r4, NIOS2_DCACHE_LINE_SIZE
    bltu    r4, r5, dcache_init_loop
```

Можно исполнять код инициализации кэша инструкций и кэша данных на ядрах Nios II, которые не имеют одного или обоих кэшей. Инструкции **initi** и **initd** просто замещаются инструкциями **nop**, если кэш не представлен в соответствующем типе процессорной системы.

Для пользователей HAL

Программам, написанным для HAL, не требуется управлять инициализацией кэш памяти. Си код HAL на стадии прогона (run-time) (crt0.S) предоставляет обработчик сброса по умолчанию, который выполняет инициализацию кэша перед вызовом alt_main() или main().

Написание драйверов устройств

Драйверы устройств обычно получают доступ к регистрам, ассоциированным с этим устройством. Эти регистры размещаются в адресном пространстве процессора Nios II. Когда устанавливается доступ к регистрам устройства, кэш данных должен пропускаться, чтобы доступ не был потерян или отложен из-за доступа к кэшу данных.

Когда пишется драйвер устройства, пропуск кэша данных осуществляется набором инструкций **ldio/stio**. В ядрах Nios II без кэша данных, эти инструкции похожи на соответствующие инструкции **ld/st** и поэтому они не опасны.

Декларирование указателя Си **volatile** не делает указателю доступ к пропуску кэша данных. Ключевое слово **volatile** только препятствует компилятору оптимизировать доступ использования указателя. Такое поведение **volatile** отличается от поведения в первом поколении процессора Nios II.

Для пользователей HAL

HAL предлагает макросы языка Си IORD и IOWR, которые распространяются на соответствующие инструкции ассемблера для пропуска кэша данных. Макрос IORD распространяется на инструкцию **ldwio**, а макрос IOWR распространяется на инструкцию **stwio**. Эти макросы позволяют HAL драйверам устройств доступ к регистрам устройства.

В табл. 9-1 показаны доступные макросы. Все эти макросы пропускают кэш данных во время выполнения этих операций. В основном ваша программа пропускает значения, определённые в файле **system.h** в качестве параметров BASE и REGNUM. Эти макросы определены в файле <*Nios II EDS install path*>**/components/ altera_nios2/ HAL/inc/io.h**.

Табл. 9-1. HAL I/O макросы для пропуска кэша данных

Макрос	Использование
IORD(BASE, REGNUM)	Читает значение регистра по офсету REGNUM в устройстве с базовым адресом BASE. Регистрам подразумевается офсет по ширине адресной шины.
IOWR(BASE, REGNUM, DATA)	Пишет значение DATA в регистр по офсету REGNUM в устройство с базовым адресом BASE. Регистрам подразумевается офсет по ширине адресной шины.
IORD_32DIRECT(BASE, OFFSET)	Делает 32-битный доступ для чтения по месту с адресом BASE+OFFSET.
IORD_16DIRECT(BASE, OFFSET)	Делает 16-битный доступ для чтения по месту с адресом BASE+OFFSET.
IORD_8DIRECT(BASE, OFFSET)	Делает 8-битный доступ для чтения по месту с адресом BASE+OFFSET.
IOWR_32DIRECT(BASE, OFFSET, DATA)	Делает 32-битный доступ для записи значения DATA по месту с адресом BASE+OFFSET.
IOWR_16DIRECT(BASE, OFFSET, DATA)	Делает 16-битный доступ для записи значения DATA по месту с адресом BASE+OFFSET.
IOWR_8DIRECT(BASE, OFFSET, DATA)	Делает 8-битный доступ для записи значения DATA по месту с адресом BASE+OFFSET.

Написание программ-загрузчиков или самоизменяемого кода

Программе, которая записывает инструкции в память, такая как программа-загрузчик или самоизменяемый код, необходимо следить за тем, чтобы старые инструкции сбрасывались из кэша инструкций и конвейера процессора. Такой сброс совершается инструкциями **flushi** и **flushp** соответственно. Дополнительно, если новая инструкция(-ии) записывается в память с использованием инструкций сохранения, которые не могут пропускать кэш данных, то вы должны использовать инструкцию **flushd** для переноса новой инструкции (-ий) из кэша данных в память.

В примере 9-4 показан ассемблерный код, который пишет новую инструкцию в память.

Example 9–4. Assembly Code That Writes a New Instruction to Memory

```
/*
 * Assume new instruction in r4 and
 * instruction address already in r5.
 */
stw    r4, 0(r5)
flushd    0(r5)
flushi    r5
flushp
```

Инструкция **stw** записывает новую инструкцию в r4 для адреса инструкций, заданного в r5. Если имеется кэш данных, инструкция будет записана прямо в кэш данных и соответствующая строка будет испорчена. Инструкция **flushd** записывает строку кэша данных, ассоциированную с адресом в r5 в память и признаёт недействительной эту строку в кэше данных. Инструкция **flushi** признаёт недействительной строку кэша инструкций, ассоциированную с адресом в r5. В завершении, инструкция **flushp** следит за тем, чтобы конвейер процессора не взял старую инструкцию по адресу, определённому в r5.

Обратите внимание, что в примере 9-4 используется пара **stw/flushd** вместо инструкции **stwio**. Инструкция **stwio** не может сбросить кэш данных, и поэтому должна оставить устаревшие данные в кэше данных.

Эта кодовая последовательность корректна для всех реализаций процессора Nios II. Если ядро Nios II не имеет определённого кэша, то соответствующая инструкция сброса (**flushd** или **flushi**) исполняется как **nop**.

Для пользователей HAL

HAL API не имеет функций для подобного управления кэшем.

Управление кэшем в системах мультимастер и мультипроцессор

Архитектура Nios II не предоставляет аппаратной когерентности кэша. Вместо этого должна предоставляться программа когерентности кэша при обмене через память общего доступа. Содержимое кэша данных всех процессоров, имеющих доступ к общей памяти, должно управляться программой, чтобы следить за тем, что все мастера читают самое последнее значение и не перезаписывают новые данные устаревшими данными. Такое управление получается при использовании сброса кэша данных и пропуском для перемещения данных между общей памятью и кэшем данных.

Инструкция **flushd** следит за тем, чтобы кэш данных и память содержали одно и то же значение на одной строке. Если строка содержит испорченные данные, она пишется в память. Эта строка признаётся недействительной в кэше данных.

Следовательно, пропуск кэша данных очень важен. Процессор не может проверять, есть ли адрес в кэше данных во время пропуска кэша данных. Если программа не может гарантировать, что соответствующий адрес находится в кэше данных, она должна сбросить адрес из кэша данных перед пропуском его для загрузки или сохранения. Это действие гарантирует, что процессор не пропустит новые (повреждённые) данные в кэш и ошибочно запишет доступные старые данные в память.

Бит 31 пропуск кэша

Набор инструкций **ldio/stio** явно пропускает кэш данных. Бит 31 предоставляет другой способ пропуска кэша данных. Используя бит 31, обычный набор инструкций **ld/st** может быть использован для пропуска кэша данных, если последний используемый бит адреса (бит 31) установлен в единицу. Значение бита 31 используется только внутренне процессором; бит 31 принудительно обнуляется в фактическом адресе доступа. Это ограничивает максимальное адресного пространство до 31 бита.

Использование бита 31 для пропуска кэша данных – это удобный механизм для программы, поскольку контроль кэширования соответствующего адреса уже содержится в адресе. Такое использование позволяет адресу передаваться коду, использующему обычный набор инструкций **ld/st**, и гарантирует, что любой доступ к этому адресу пропустит кэш данных.

Бит 31 пропуска кэша предоставляется только с ядром Nios II/f и не должен использоваться с другими ядрами Nios II. Другие ядра Nios II сохраняют максимальную адресацию до 31 бит, чтобы упростить миграцию кода с одной реализации ядра на другую. Они эффективно игнорируют значение бита 31, что позволяет коду, написанному для ядра Nios II/f, использовать бит 31 для пропуска кэша, и корректно запускаться на других текущих реализациях Nios II. Таким образом это средство зависит от реализации ядра Nios II.

За подробной информацией о средствах каждой реализации ядра Nios II, обратитесь к главе "Подробности реализации ядра Nios II" в настольной книге по процессору Nios II.

Для пользователей HAL

HAL предлагает макрос языка Си IORD_*DIRECT, который распространяется на набор инструкций **ldio** и макрос IOWR_*DIRECT, который распространяется на набор инструкций **stio**. Обратитесь к табл. 9-1 на стр. 9-4. Эти макросы предлагают доступ к некэшируемым регионам памяти.

HAL предлагает процедуры alt_uncached_malloc(), alt_uncached_free(), alt_remap_uncached() и alt_remap_cached() для размещения и манипуляции регионами в некэшируемой памяти. Эти процедуры доступны в ядрах Nios II с и без кэша данных – код, написанный для ядра Nios II с кэшем данных, полностью совместим с ядром Nios II без кэша данных.

Процедуры alt_uncached_malloc() и alt_remap_uncached() гарантируют то, что размещённый регион памяти не является кэшем данных и весь дальнейший доступ к размещённому региону памяти пропускает кэш данных.

Прочно сопряжённая память

Если вы хотите иметь постоянные характеристике кэша, разместите ваш код или данные в прочно сопряжённую память. Прочно сопряжённая память – это быстрая внутри чиповая память, которая пропускает кэш и имеет гарантированную низкую задержку. Прочно сопряжённая память даёт наилучшие характеристики доступа к памяти. Вы можете назначить код и данные разделам внутри чиповой памяти тем же способом, как и другим секциям памяти.

Кэш инструкций не влияет на прочно сопряжённую память. Однако инструкции управления кэшем становятся NOPs, что может стать результатом непредвиденного переполнения.

За дополнительной информацией обратитесь к секции "Использование памяти" в главе "Разработка программ с использованием слоя аппаратной абстракции" в настольной книге программиста Nios II.

Содержание