Руководство программиста MALT_SW

Выпуск 1.8

МГУ

авг. 05, 2019

Оглавление:

ГЛАВА 1

Краткое описание MALT_SW

Набор программных средств MALT_SW состоит из следующих компонентов:

1. Программный эмулятор MALTemu аппаратуры MALT (Multicore Architecture with Lightweight Threads).
2. Компоненты для установки компилятора GCC для MALT.
3. Системные библиотеки и наборы заголовочных файлов языков C/C++ для MALT.
4. Эмулятор сопроцессора (опционально).
5. Ассемблер и компилятор языка C для архитектуры сопроцессора (опционально).
6. Примеры программ и тесты для MALT и сопроцессора.
7. Инструментальное ПО для работы с MALT и сопроцессором.
8. Программы для взаимодействия с аппаратными платформами (ПЛИС/СБИС).

Компоненты MALT_sw предназначены для запуска тестовых и целевых программ в виртуальной среде. Виртуальная среда состоит из нескольких эмуляторов в которых все реальные аппаратные компоненты систем MALT и сопроцессора имитируются с помощью программных сущностей. Эмуляторы призваны обеспечить максимальное сходство основных характеристик эмулируемого оборудования с реальной аппаратурой, например, в эмуляторе MALTemu имитируются задержки доступа CPU в память и низкоуровневые аспекты аппаратной среды. Набор программных средств MALT_sw является одним из ключевых инструментов для создания, отладки и анализа производительности разрабатываемого прикладного и системного программного обеспечения для MALT.

В системе MALT поддерживается программная среда, основанная на программном обеспечении GNU и дополненная системными библиотеками, специфичными для среды MALT. Реализована поддержка архитектурой MALT компилятора GNU C/C++. Наличие компилятора GNU C/C++ позволяет перенести практически любую программу из огромного числа программ GNU в среду MALT. Компилятор поддерживается вместе со всеми необходимыми средствами сопровождения: компоновщиком, отладчиком, системой сборки make, утилитами из набора binutils (objdump, ar, as, nm и др.). Портированные в среду MALT библиотеки GMP (для работы с целыми числами, рациональными дробями и вещественными числами любой точности), MPC (поддерживает работу с комплексными числами произвольной точности) и MPFR (поддержка работы с вещественными числами произвольной точности и правильным округлением).

Разработана системная библиотека Libmalt, которая обеспечивает возможность использования аппаратных средств системы MALT. Адаптирована стандартная библиотека языка Си (Tiny_libc), которой достаточно для разработки базового системного ПО. Совместно с GNU C/C++ можно использовать библиотеку Libgloss, которая является адаптированной библиоткой newlib.

1.1 MALTemu

MALTemu – это эмулятор аппаратного обеспечения вычислительной системы MALT, относящейся к классам ManyCore и NoC (Network-on-Chip). Многоядерная система MALT на базе вычислительных ядер MB-Lite с 32-битной архитектурой MicroBlaze имеет иерархическую (двухуровневую) структуру, где сообщение между вычислительными элементами осуществляется с помощью высокоскоростной внутрисистемной шины передачи данных.

MALTemu – ключевой инструмент в цикле разработки и оптимизации программ для MALT, помогающий решать следующие основные задачи:

- разрабатывать и портировать программное обеспечение на систему MALT;
- получать оценки эффективности существующих вычислительных алгоритмов в случае их адаптации к оборудованию;
- производить низкоуровневую оптимизацию кода при адаптации к платформе MALT;
- проводить всестороннее тестирование программ на различных аппаратных конфигурациях;
- способствовать ознакомлению специалистов с программированием для системы MALT.

Кроме этого ряд дополнительно введенных в MALTemu механизмов контроля состояния эмулируемой аппаратуры помогает в процессе отладки решать следующие задачи:

- локализовывать некоторые классы труднообнаруживаемых программных ошибок;
- оперативно загружать и выгружать тестовые данные большого объема;
- использовать дополнительные «виртуальные» команды для вызова отладчика и отладочной печати в тех ситуациях, когда реальное оборудование этого не позволяет.

Более подробное описание архитекруры и внутреннего устройства эмулятора приведено в разделе Раздел 2.

1.2 Сопроцессор

Гибкость системы MALT позволяет подключать различные процессорные элементы в виде сопроцессоров. В качестве примера сопроцессора можно рассматривать множество ПЭ, объединенных в кластер по принципу SIMD-ускорителя. По умолчанию эмулятор сопроцессора подключается к MALTemu с интерфейсом, аналогичному аппаратной реализации. Более подробное описание архитектуры и внутреннего устройства эмулятора поставляемого сопроцессора (в зависимости от целевой задачи) приводится в отдельном документе.

ГЛАВА 2

Архитектура системы MALT

Многоядерная система MALT представляет собой процессор общего назначения, содержащий десятки или сотни простых управляющих ядер с возможностью подключения к кажому управляющему ядру сопроцессоров для вычислений. В зависимости от конфигурации системы в качестве оперативной памяти может использоваться память типа ZBT SRAM, DDR2 SDRAM, DDR3 SDRAM, RLDRAM 3. Управляющие ядра делятся на три типа. Первый тип управляющих ядер – это подчинённые ядра (Slave cores), которые объединяются в цепочки собственной пакетной шиной Simple Packet Bus (SPBUS). В начале каждой цепочки расположено управляющее ядро второго типа (Master core), задача которого – управление загрузкой подчиненных ядер и маршрутизация сообщений между различными цепочками. Управляющие ядра связаны между собой второй ветвью пакетной шины SPBUS и управляются управляющим ядром третьего типа (Super Master Core), задачей которого является распределение вычислительной нагрузки по подчинённым ядрам. К каждому подчиненному ядру подключается сопроцессор, тип которого определяется решаемой задачей. Однако, в зависимости от круга решаемых задач, вместо сопроцессоров для вычислений можно использовать подчиненные ядра. К ведущему ядру подключены основные периферийные устройства: таймер, последовательный порт UART, контроллер прерываний, контроллер флеш-памяти и др.

2.1 Основные компоненты

- *Supermaster*: основной задачей Supermaster является распределение вычислительной нагрузки по другим управляющим ядрам. Исполнение программы начинается с исполнения функции main() на Supermaster, которая в дальнейшем может задействовать подчинённые ядра системы. Supermaster автоматически запускает все ведущие ядра при инициализации. В отличие от других управляющих ядер Supermaster использует несколько периферийных устройств: контроллер прерываний, порт последовательной передачи данных для подключения к консоли и управления с помощью неё, а также таймер. Опционально можно подключать контроллер DMA, разъём для подключения платы PCIe и др. Supermaster использует сложный и гибкий механизм начальной загрузки (bootstrapping) с помощью стандартной библиотеки malt_ic_init() при начальной загрузке. Ряд функций и возможностей стандартной библиотеки предназначены только для Supermaster: работа с масками прерываний контроллера прерываний, установка функции, выполняемой при каждом прерывании. Supermaster по умолчанию не использует быструю локальную память для хранения стека, однако есть возможность переключения на использование быстрой памяти, а также вызова отдельных функций через использование быстрой памяти.

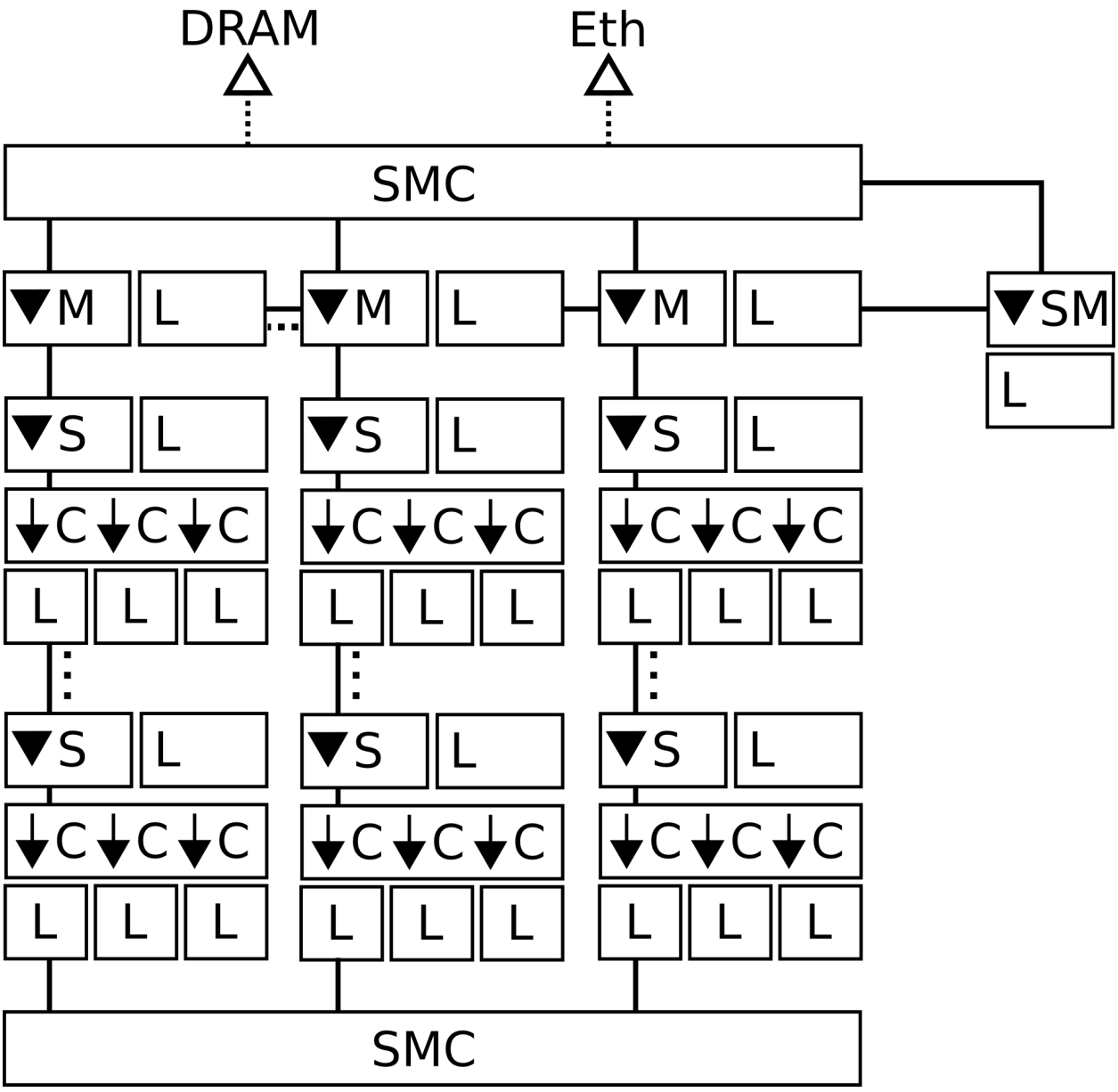

Рис. 2.1: Схема системы MALT

- *Master*: master-ядра располагаются в начале каждой цепочки. Основной задачей Master-ядер является управление загрузкой подчиненных ядер и маршрутизация сообщений между различными цепочками. Для начальной инициализации используется код в ПЗУ/ROM. Master-ядра обрабатывают пакеты данных, адресованные им. Первое слово таких пакетов они рассматривают как код команды для исполнения, а второе слово – как данные для этой команды. Среди команд для ядер типа Master есть команды на запуск подчинённых ядер (всех или одного), на получение состояния подчинённых ядер, на инвалидацию (сброс) процессорного кэша команд, на установку и запуск функции обратного вызова (callback), на передачу сообщения ядру типа Supermaster об освобожнении всех подчинённых ядер, на приём уведомления от подчинённого ядра о завершении его исполнения и др. Для прикладного программирования можно использовать callback-функции, по одной на каждое Master-ядро. Вызов такой функции аналогичен вызову стандартной C-функции с одним параметром-словом. Ядра типа Master используются в основном для маршрутизации данных при передаче пакетов по шине SPbus между ядрами на разных цепочках;
- *Slave*: являются по существу аппаратными аналогами программных тредов-нитей. Для начальной инициализации используется код в ПЗУ/ROM. Существующее ПО системы MALT не позволяет использовать с подчинёнными ядрами или ядрами типа Master ряд функций стандартной библиотеки, например, для стандартного ввода и вывода. Существующее ПО системы MALT также пока не позволяет при запуске тредов-функций передавать параметры. Если такой функции нужно получить параметры или передать результаты своей работы, то можно использовать либо общую память (это работает медленно, но реализуется очень просто), либо передачу данных через интерконнект. Подчиненные ядра и Master-ядра используют для хранения стека быструю локальную память, аналог TLS программных тредов. Часть этой быстрой памяти используется также для хранения буфера интерконнекта. Размер такой памяти – 8 КБ. Ядра типов Slave и Master имеют только один источник прерываний – приход пакета SPBUS. В одной цепочке допустимо наличие до 63 слейвов;
- *Сопроцессоры*: сопроцессоры подключаются к управляющим ядрам, тип сопроцессора определяется в зависимости от решаемой задачи;
- *Интерконнект или шина SPBUS*: шина SPBUS используется для быстрой передачи данных от одного ядра другому. Для программирования использования интерконнекта можно использовать вызов соответствующих системных функций. Аппаратно функции интерконнекта доступны через обращения к регистрам, отображаемым в локальную память каждого ядра. Передача данных осуществляется пакетами. Типовой размер пакета – 4 слова, из которых два слова несут служебную информацию и два слова – данные. Интеллектуальный контроллер памяти может передавать пакеты большей длины;
- *Кэш процессорных инструкций*: объем кэша инструкций зависит от конфигурации системы и составляет несколько КБ для каждого управляющего ядра. Сброс содержимого кэша инструкций осуществляется программно;
- *Общая память*: общая память подключена к общей шине и шине SPBUS через интеллектуальный контроллер, который может выполнять операции без обращения к управляющим ядрам системы. Операции:
  - копирование области общей памяти;
  - перенос данных из общей памяти в локальную память заданного ядра;
  - инкремент слова в памяти на заданную величину;
  - заполнение заданного региона общей памяти заданным словом;
  - косвенное чтение из общей памяти через список заданных адресов в общей памяти в локальную память заданного ядра;
  - операции с FE-битами общей памяти.

    Интеллектуальный контроллер подключен к интерконнекту через все Master-ядра, включая Supermaster. Обращение к общей памяти занимает порядка сотни тактов процессора, что на порядок медленнее интерконнекта.

2.1.1 Вспомогательные элементы

- *Контроллер прерываний*. Контроллер прерываний обслуживает до 4 устройств. В типовой конфигурации MALT к нему подключены три устройства: шина SPBUS, последовательный порт и таймер;
- *Таймер*. Таймер позволяет измерять точные интервалы времени. На всех ядрах доступны счетчики тактов, которые также позволяют измерять временные интервалы.

2.2 Управляющие ядра системы

Одно управляющее ядро системы MALT состоит из следующих компонентов:

- 32-битного процессорного RISC-ядра общего назначения MB-Lite;
- ПЗУ инструкций на шине инструкций MB-Lite;
- декодера адреса шины данных MB-Lite;
- кэша инструкций;
- блока локальной памяти;
- контроллера пакетной шины SPBUS;
- контроллера периферийных интерфейсов (Supermaster).

Структурная схема управляющего процессора приведена на Рис. 2.2.

Центральной частью одного управляющего ядра системы MALT является целочисленное процессорное ядро общего назначения MB-Lite. Ядро MB-Lite является целочисленным 32-битным RISC-процессором общего назначения, совместимым на уровне машинного кода с архитектурой MicroBlaze. Исполняемая программа может частично храниться в ПЗУ инструкций ядра либо подгружаться через используемую шину SPBUS с промежуточным хранением инструкций в кэше (24 кБайт). Данные для исполняемой программы могут быть загружены по шине SPBUS в локальную память (8 кБайт). Благодаря низколатентному доступу к локальной памяти данных достигается эффективная работа алгоритмов, требующих небольших объёмов данных по сравнению с алгоритмами, которые нуждаются в постоянной подгрузке данных из внешнего ресурса. Кеша данных не предусмотрено по причине значительных накладных расходов по обеспечению кэш-когерентности в высокопараллельных системах. На шине данных процессорного ядра присутствует декодер адреса, обеспечивающий подключение к шине данных как блоков памяти, так и других устройств с регистровым интерфейсом, отображаемым на шину данных.

Использование архитектуры RISC для управляющего ядра позволяет исполнять большинство его инструкций за один такт. Управляющее ядро предоставляет для использования 32 регистра общего назначения, счётчик-указатель исполняемых команд и несколько специальных регистров. Большинство команд работы с данными – трёхадресные, команды переходов – двухадресные. Инструкции процессора занимают одно слово – 4 байта. Однако, в системе команд есть команда-префикс для использования 32-разрядной константы, что фактически превращает эту и следующую команды в одну команду из двух машинных слов.

Архитектура MicroBlaze имеет ряд особенностей, отличающих её от процессоров Intel x86 или ARM. Используется порядок байт от старшего к младшему – Big Endian. Не используются биты-флажки результатов операций, вместо них используется сам результат операции. Например, если нужно совершить переход при отрицательном результате, то нужно делать переход с указанием регистра, куда помещён был результат. Другая особенность архитектуры MicroBlaze – использование так называемых пазов/слотов задержки при переходах по причине того, что переход нарушает нормальную работу конвейера инструкций и требует его перезагрузки, что задерживает ход исполнения программы на несколько тактов. Паз задержки – это инструкция, размещаемая сразу за инструкцией перехода с задержкой, которая исполняется после выполнения перехода. Таким образом, как бы нарушается естественный ход исполнения, но этим достигается экономия одного цикла процессора. Большинство команд переходов имеют варианты с пазами задержки и без, но некоторые команды переходов

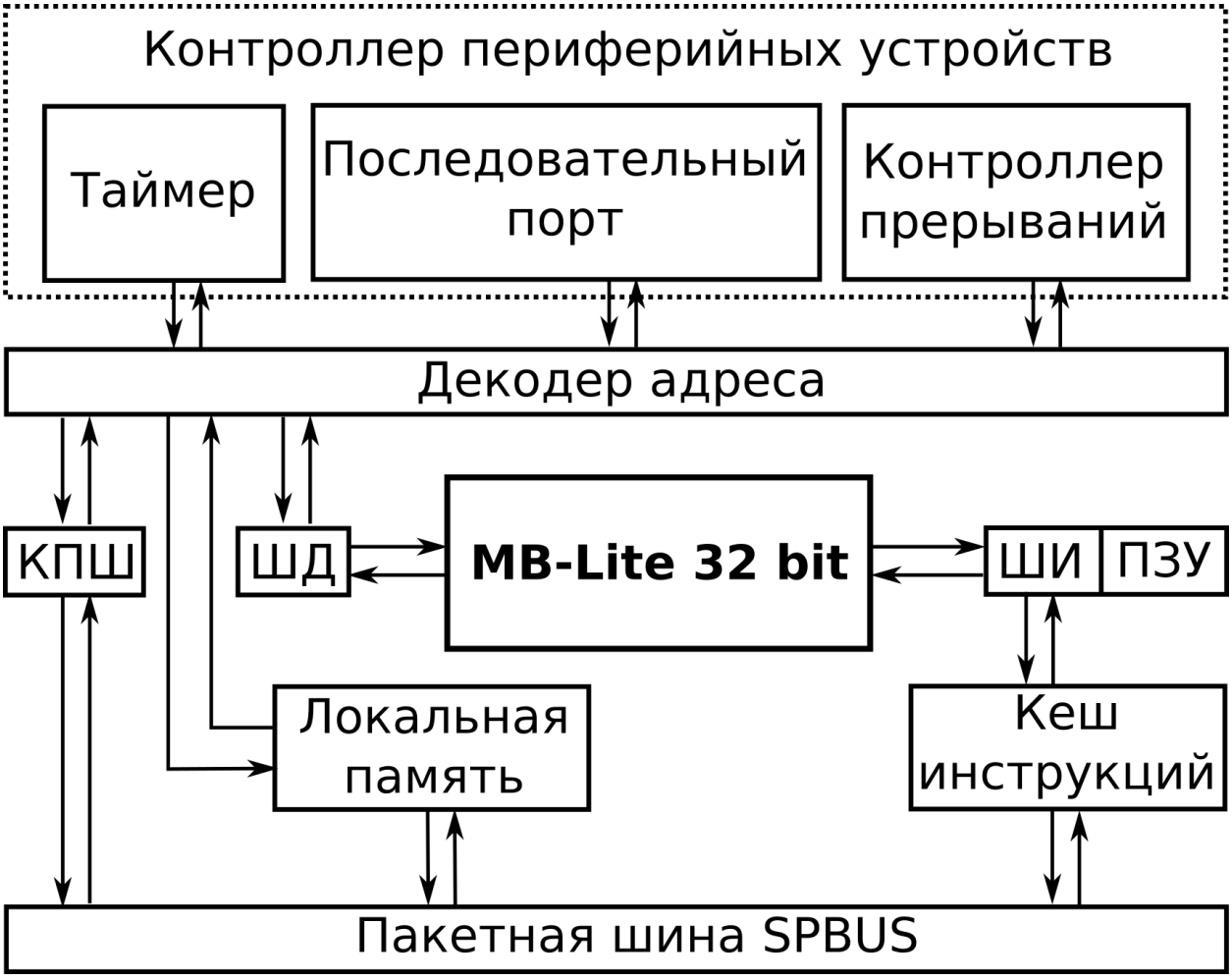

Рис. 2.2: Схема одного управляющего ядра (Supermaster), где ШД – шина данных, ШИ – шина инструкций и КПШ – контроллер пакетной шины SPBUS

всегда используют паз задержки. К особенностям архитектуры MicroBlaze можно отнести ещё факт того, что один из регистров общего назначения, R0, всегда равен 0 – любые операции с ним не меняют этого значения. Этот факт позволяет, в частности, использовать команду сложения для загрузки константы вместо специализированной для этой цели команды.

Часть возможностей процессора MicroBlaze опциональные: быстрый битовый сдвигатель (barrel shifter), умножитель, делитель, система защиты памяти, поддержка виртуальной памяти, работа с кэшами команд и инструкций, исключения, прерывания и т. п. В системе MALT используют только поддержку работы с кэшем инструкций, прерываниями и опционально с быстрым битовым сдвигателем и умножителем.

В архитектуре MALT используется только два флага регистра состояний процессора: флаг переноса и флаг разрешения прерываний. Флаг переноса используется только для арифметических команд и не служит признаком для переходов. Арифметические команды имеют вариации, позволяющие использовать или не использовать флаг переноса. Они также имеют вариации (это особенность ISA MicroBlaze), позволяющие устанавливать или сохранять неизменным флаг переноса. Флаг разрешения прерываний при его смене загрузкой регистра состояний актуализируется с задержкой на один такт.

Подробнее о системе команд процессора MicroBlaze и других его особенностях можно прочитать в разделе ref:*malt_rtl_mblite*.

2.3 Сопроцессор

Сопроцессор опционально подключается к Slave-ядру. К одному Slave-ядру можно подключить несколько сопроцессоров. Решаемая целевая задача определяет вид подключаемого сопроцессора. В стандартной системе malt_sw сопроцессоры отсутствуют. Описание сопроцессора для целевой задачи произведено в отдельном документе.

2.4 Внутрисистемная сеть SPBUS

2.4.1 Описание формата передаваемых данных

Базовой структурой, побайтно соответствующей поддерживаемому аппаратурой дескриптору пакета, является структура spbus_packet:

```
typedef struct __attribute__ ((__packed__)) spbus_packet
{
   u8_t  type;           // Тип пакета
   u8_t  len;            // Длина пакета
   u16_t target;         // Код ядра-получателя данных
   u16_t add_data;       // Поля для дополнительных (служебных) данных
   u16_t source;         // Код ядра, передающего данные
   u32_t data0;          // Первое слово данных
   u32_t data1;          // Второе слово данных
} spbus_packet;
```

Данная структура обеспечивает передачу как минимум двух 32-битных указателей, что достаточно для эффективной реализации взаимодействия в системе с общей памятью.

2.4.2 Описание топологии связей интерконнекта

Представление двумерного графа связности внутрисистемной сети SPBUS приведено на Рис. 2.3.

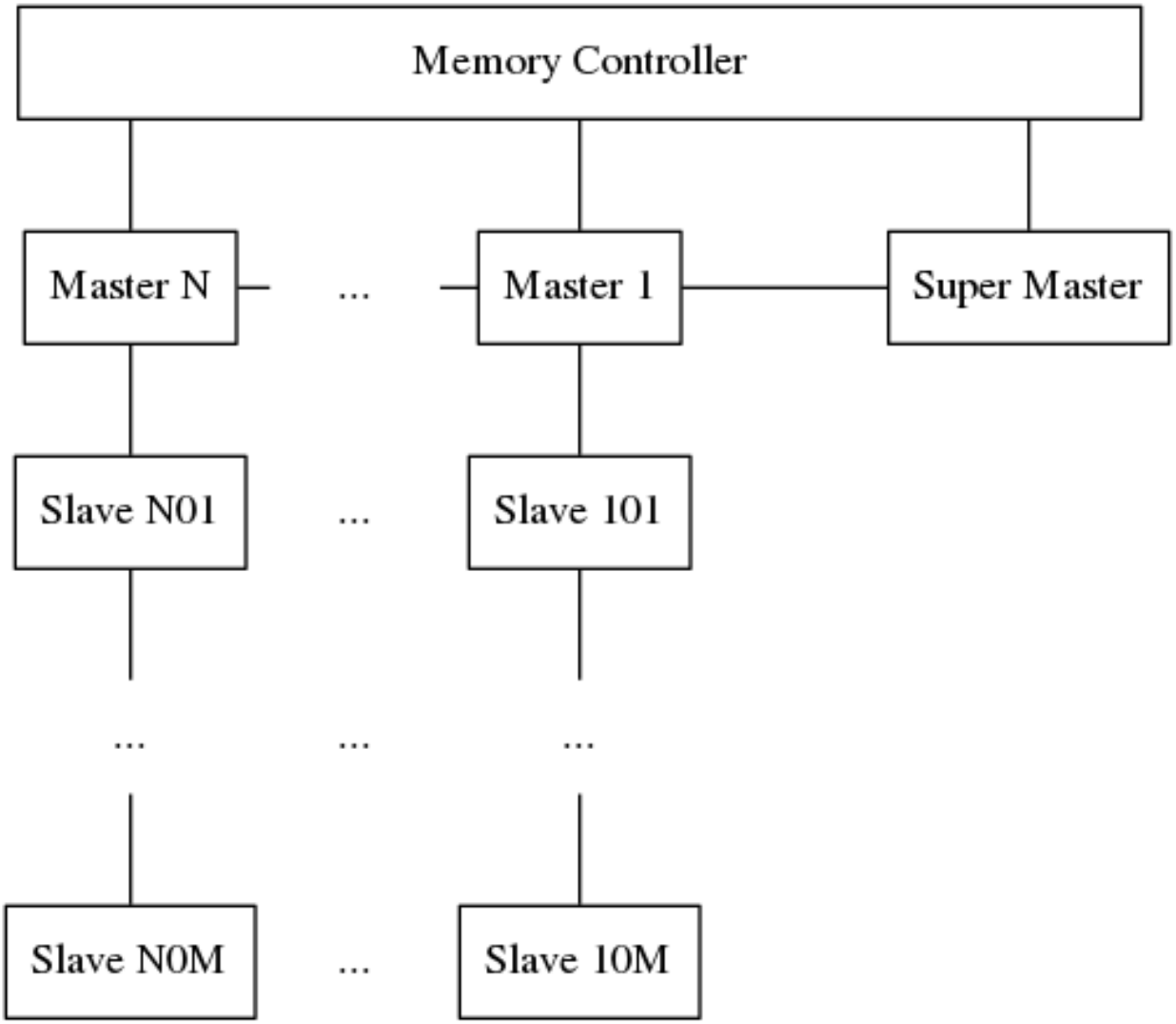

Рис. 2.3: Топология внутрисистемной сети SPBUS

Физические соединения организованы с помощью линий связи (цепочек), одна из которых (нулевая по номеру) служит для соединения других цепочек между собой.

С точки зрения процесса передачи данных, каждая линия является полнодуплексной, то есть одновременная передача пакетов возможна в обоих направлениях. Инициатором посылки и получателем являются ядра CPU, а адресация производится указанием их кодов.

Код ядра каждого процессора может быть вычислен по следующим формулам:

1. Код управляющего процессора (Supermaster) равен 0.
2. Коды вспомогательных процессоров-маршрутизаторов равны 0x100 * L, где L – номер линии (линии нумеруются, начиная с единицы). Эти процессоры организуют транзит между ядрами, не находящимися в пределах одной линии.
3. Код контроллеров ускорителей равен 0x100 * L + N, где L – номер линии, а N – порядковый номер ядра внутри линии (начиная с единицы).

Число 0x100 – это 16-е представление числа 256 в так называемом C-формате. Таким образом, ядро 0x207 – это 7-е подчинённое ядро (slave), подключённое ко второму ядру типа Master.

Специальные устройства получают адреса специальным образом, например, контроллер памяти имеет адрес 0xff00.

2.4.3 Описание алгоритма маршрутизации передачи данных

Граф связности системной сети SPBUS представляет собой неориентированное дерево, поэтому между каждыми двумя ядрами существует естественный кратчайший маршрут, который закодирован в процедурах, реализующих используемый алгоритм маршрутизации.

Данные процедуры, выполняющиеся на ядрах (источнике, получателе и промежуточных ядрах, осуществляющих транзит) охарактеризованы следующими правилами:

1. В случае совпадения кода ядра и адреса получателя пакет считается принятым.
2. В случае если ядро-получатель находится в текущей цепочке, осуществляется передача непосредственно получателю.
3. В случае если ядро-получатель находится в другой цепочке, происходит транзит через одно или два вспомогательных ядра, которые производят соответствующие пересылки в пределах проходящих через них линий связи.

Задержка в аппаратной среде связана со свойствами организованного в пределах кристалла физического канала с длительностью десятки циклов CPU. В эмуляторе аппаратная задержка моделируется путем останова ядра-получателя на количество циклов шины, рассчитываемое по формуле, аппроксимирующей реальную задержку в аппаратуре.

Темп передачи данных в аппаратуре определяется физическими свойствами канала и ограничен скоростью работы процедур посылки, которая тратит больше времени на подготовку и обработку пакетов, чем реальная аппаратная среда.

Наибольшие задержки происходят при транзите через вспомогательные Master-ядра, так как маршрутизация осуществляется ими программно. Master-ядро должно принять при помощи вызова соответствующей системной функции требующий транзита пакет и затем при помощи другой системной функции переслать его дальше.

2.4.4 Структурные компоненты интерконнекта

Компоненты интерконнекта:

- Аппаратные связи между ядрами;

- Аппаратный круговой буфер на каждом ядре объемом 32 пакета с доступом из прикладных программ;
- Программный дополнительный буфер пакетов (используется, начиная с аппаратной версии 1.8) – сюда прерыванием копируются пакеты данных и уведомлений из аппаратного кругового буфера, которые здесь ожидают извлечения при помощи специальных функций системной библиотеки;
- Спецификация формата пакетов;
- Функции системной библиотеки для приёма и отправки пакетов;
- Три аппаратных указателя-счётчика, которые связаны с аппаратурой и практически недоступны для прикладных программ:
  - для приёма пакетов;
  - для указания на пакеты в буфере, ожидающие обработки через прерывание;
  - для указания на пакеты в буфере, ожидающие обработки прикладной программой (не используется, начиная с аппаратной версии 1.8).

Счётчики-указатели на пакеты, ожидающие обработки, автоматически увеличиваются на 1 при обращении.

Пакеты, обрабатываемые через прерывание, не могут маршрутизироваться Master-ядрами. Если необходимо передавать пакет от Supermaster-ядра к Slave-ядру, то следует на соответствующем мастере установить callback-функцию, которая своим вызовом передаст требуемый пакет на Slave-ядро.

Пакет – это последовательность из 4 слов (16 байт), из которых 2 слова – это пересылаемые данные. Оставшиеся два слова содержат адрес ядра-отправителя (полуслово – 2 байта), адрес ядра-получателя (полуслово), тип пакета (байт), длина (байт, обычно фиксирована в 2), дополнительных данных (полуслово).

Существует два типа пакетов для обработки программно: пакеты данных – тип 16 и пакеты уведомлений – тип 20. Пакеты уведомлений не несут данных, а только сигнализируют о своем появлении и служат для синхронизации. На ядрах типа Master пакеты данных могут использоваться для специальных целей, такие пакеты в первом поле данных определяют свой подтип, который задаёт способ их использования Master-ядром.

Начиная с аппаратной версии 1.8 для пакетов данных и уведомлений, используется дополнительный буфер, размещенный в быстрой статической памяти каждого ядра. Дополнительный буфер размещается после области TLS на дне стека и растет в сторону вершины стека. Для управления буфером в области TLS хранятся два указателя на начало и конец очереди принимаемых программно пакетов. Приходящие пакеты размещаются в конец очереди, что создаёт возможность столкновения с содержимым стека. Имеется возможность проверки наличия ситуации для помещения очередного пакета в данные стека и сброс такого пакета для избежания коррумпирования содержимого стека. Теоретически такой сброшенный пакет можно в дальнейшем достать из аппаратного кольцевого буфера по сохраняемому при таком сбросе указателю. Рассматриваемая возможность носит опциональный характер, так как вероятность столкновения со стеком при нормальной работе чрезвычайно мала, а при чрезвычайно интенсивном росте стека вероятность аварийного завершения работы очень велика. По умолчанию эта опция включена.

Для извлечения пакетов из дополнительного буфера сделана базовая функция malt_spbus_recv_ext(), которая позволяет принимать пакеты как в естественном порядке, так и с фильтрацией по типу и/или адресу отправителя пакета. При использовании фильтрации работа по извлечению пакета несколько замедляется. Если искомый пакет расположен в начале очереди, то после копирования этого пакета из буфера счетчик-указатель принимаемых пакетов инкрементируется. Если же искомый пакет оказывается не в начале очереди, то он после копирования из буфера инвалидируется, а счетчик-указатель принимаемых пакетов не меняется. При переходе счетчика-указателя размещаемых пакетов за так называемую «мягкую» границу и при небольшом размере очереди она сдвигается к началу дополнительного буфера. А при переходе за так называемую «жёсткую» границу сдвиг происходит безусловно. Названные границы можно менять специальными системными функциями. Сдвиги буфера производятся названной выше функцией системной библиотеки. Для удобства пользователей на основе этой базовой функции сделаны функции или макросы с уменьшенным числом аргументов malt_spbus_recv() – для приёма пакетов без учёта адреса отправителя пакета, malt_rec_ic_packet_from() – для

приёма пакетов без учёта типа пакета, malt_rec_ic_notification_from() – для приёма пакетов-уведомлений от заданного адресом отправителя пакета.

Таким образом, в процессе приёма и передачи пакетов начало и конец дополнительного буфера постоянно сдвигаются, что отчасти напоминает движение гусеницы. Для использования интерконнекта с буферизацией необходимо резервирование 5 слов (20 байт) в памяти TLS (Thread Local Storage) каждого ядра:

- Первое слово – это указатель на конец *гусеницы*, в который кладутся приходящие пакеты. После приёма пакета этот указатель увеличивается. Приём пакетов в буфер производится через прерывание.
- Второе слово – это указатель на начало *гусеницы* в буфере, с которого идет приём пакетов функцией системной библиотеки для прикладных программ. При получении пакета, если нужный пакет был в начале «гусеницы», происходит увеличение указателя, а если пакет был не в начале, то значение указателя не меняется, но место в буфере, занимаемое принятым пакетом инвалидируется.
- Третье слово – это маркер столкновения со стеком. Если это слово равно нулю, то столкновения *буфера-гусеницы* со с стеком не зафиксировано. Если такое столкновение было зафиксировано, то в этом слове хранится указатель на не принятый из-за опасности повредить содержимое стека пакет в аппаратном буфере.
- Четвертое слово хранит *жёсткую границу* для *гусеницы*. Если при получении пакета системной функцией фиксируется заползание за эту границу, то происходит безусловный сдвиг *гусеницы* к началу буфера.
- Пятое слово хранит *мягкую границу* для гусеницы. Если при получении пакета системной функции фиксируется заползание за эту границу, то сдвиг *гусеницы* к началу буфера происходит только, если размер «гусеницы» меньше заданного (сейчас это константа 6). При сдвиге все инвалидированные промежутки ликвидируются.

Дополнительный буфер интерконнекта и системный стек ядра разделяют общую область памяти. Они расширяются навстречу друг другу.

Интерконнект с дополнительным буфером повышает надежность операций получения пакетов и позволяет проводить фильтрацию пакетов, как было описано выше.

2.5 «Умный» контроллер памяти

В системе MALT на аппаратном уровне реализован Smart Memory Controller с поддержкой типовых операции с памятью, которые обычно отсутствуют в аппаратуре компьютерных систем либо присутствуют в урезанной форме контроллера DMA/ПДП. Контроллер является активной сущностью, которая получает управление и обрабатывает поступившие запросы на каждом такте работы системы.

Простейшие операции контроллера памяти: READ, чтения из памяти с учетом значение FE-бита, и WRITE записи в память данных. Если бит находится в состоянии empty, то операция чтения приостанавливается до тех пор, пока бит не перейдет в состояние full. Если операция чтения была заблокирована, то после её разблокировки читаются данные, получаемые в момент разблокировки. Операция записи значения FE-бита переводит его в полное состояние, не освобождая заблокированные чтения, в связи с чем не следует применять операцию WRITE к заблокированной ячейке памяти. Снятие блокировки производится операцией WRITE-UNBLOCK, которая помимо записи данных в ячейку памяти может правильно разблокировать ожидающие операции чтения.

Работа с FE-битами производится с помощью операций READ, WRITE-UNBLOCK и READ-MODIFY. Последняя операция при обращении к полной ячейке работает как чтение с последующим переводом ячейки в пустое состояние. Если READ-MODIFY обращается к пустой ячейке, то она блокируется. Операция WRITE-UNBLOCK переводит ячейку в полное состояние и освобождает все заблокированные операции чтения (READ) до первой операции чтения с модификацией, которую также освобождает. Если заблокированных операций чтения с модификацией нет, то происходит разблокировка всех чтений. Если же операция чтения с

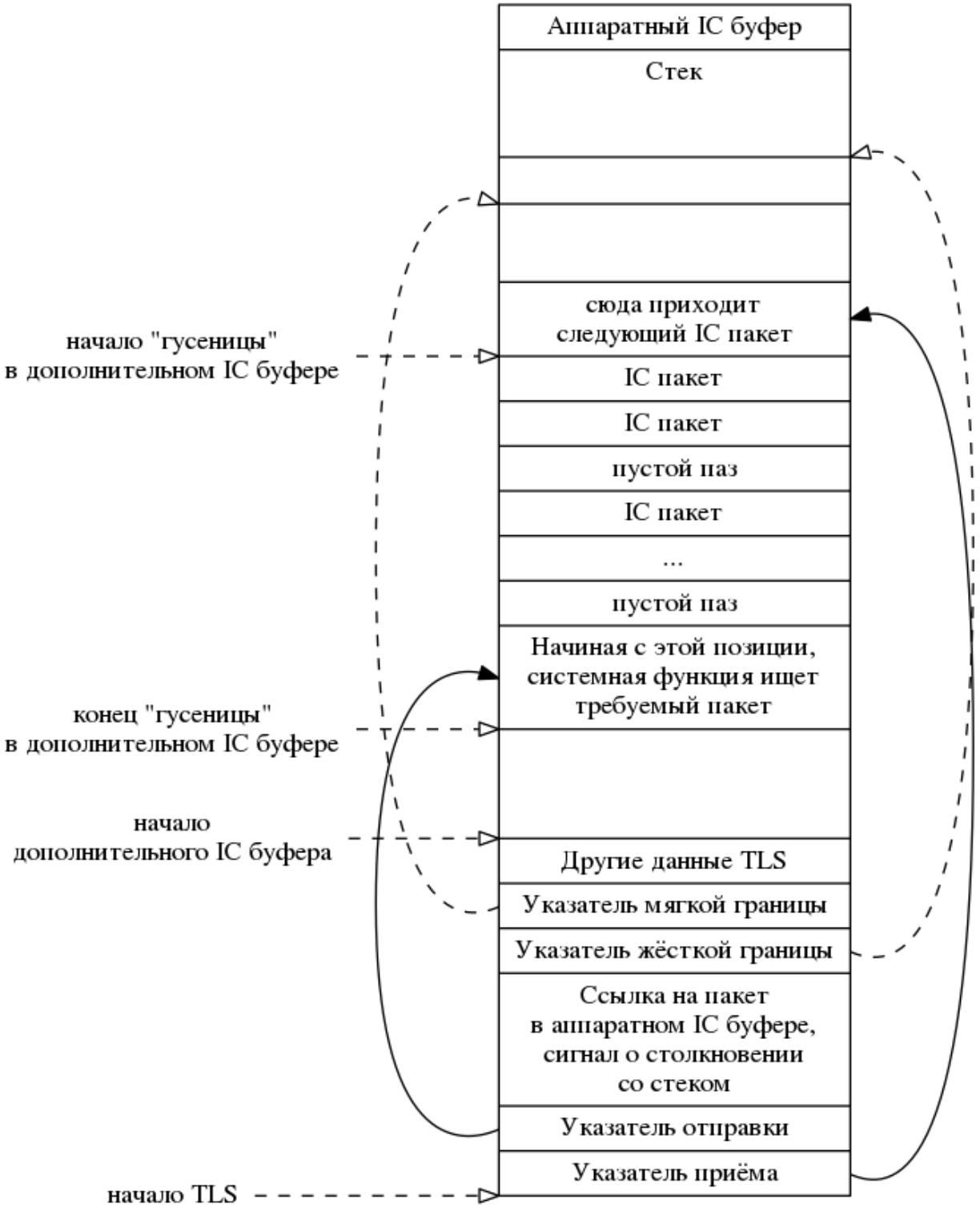

Рис. 2.4: Структурная схема дополнительного буфера и смежных областей памяти

модификацией была разблокирована, то она при своём выполнении, считав данные, переводит FE-бит в пустое состояние, что блокирует все последующие чтения.

Рис. 2.5: Схема работы операции READ для FE-битов

Операция атомарного инкремента позволяет изменить значение выбранной ячейки памяти на знаковое расширение заданного 16-разрядного числа. Результат операции – это изменённое значение ячейки памяти, которое возвращается по заданному адресу в локальную память ядра. Операции атомарного инкремента, READ, WRITE-UNBLOCK и READ-MODIFY применяются к словам, 4-байтным величинам.

Контроллер памяти может выполнять блочные операции, которые применяются к данным, размер и адрес которых в общей памяти кратен 16 байтам (четырём словам). Контроллер памяти может переносить блоки данных из общей памяти в локальную, а также копировать блоки данных из одной области общей памяти в другую. Заданную область общей памяти можно заполнять заданной величиной, 4-байтным словом. Последняя операция – это косвенное чтение данных из общей памяти в локальную. Для выполнения этой операции необходимо сформировать в общей памяти массив адресов. При выполнении операции данные, слова, указанные по этим адресам будут, перенесены в локальную память по заданному адресу.

Для ряда операций с контроллером памяти сделаны удобные функции-обёртки malt_read_modify(), malt_fill_write() – это операция WRITE-UNBLOCK, malt_ext_memop_inc() – инкремент.

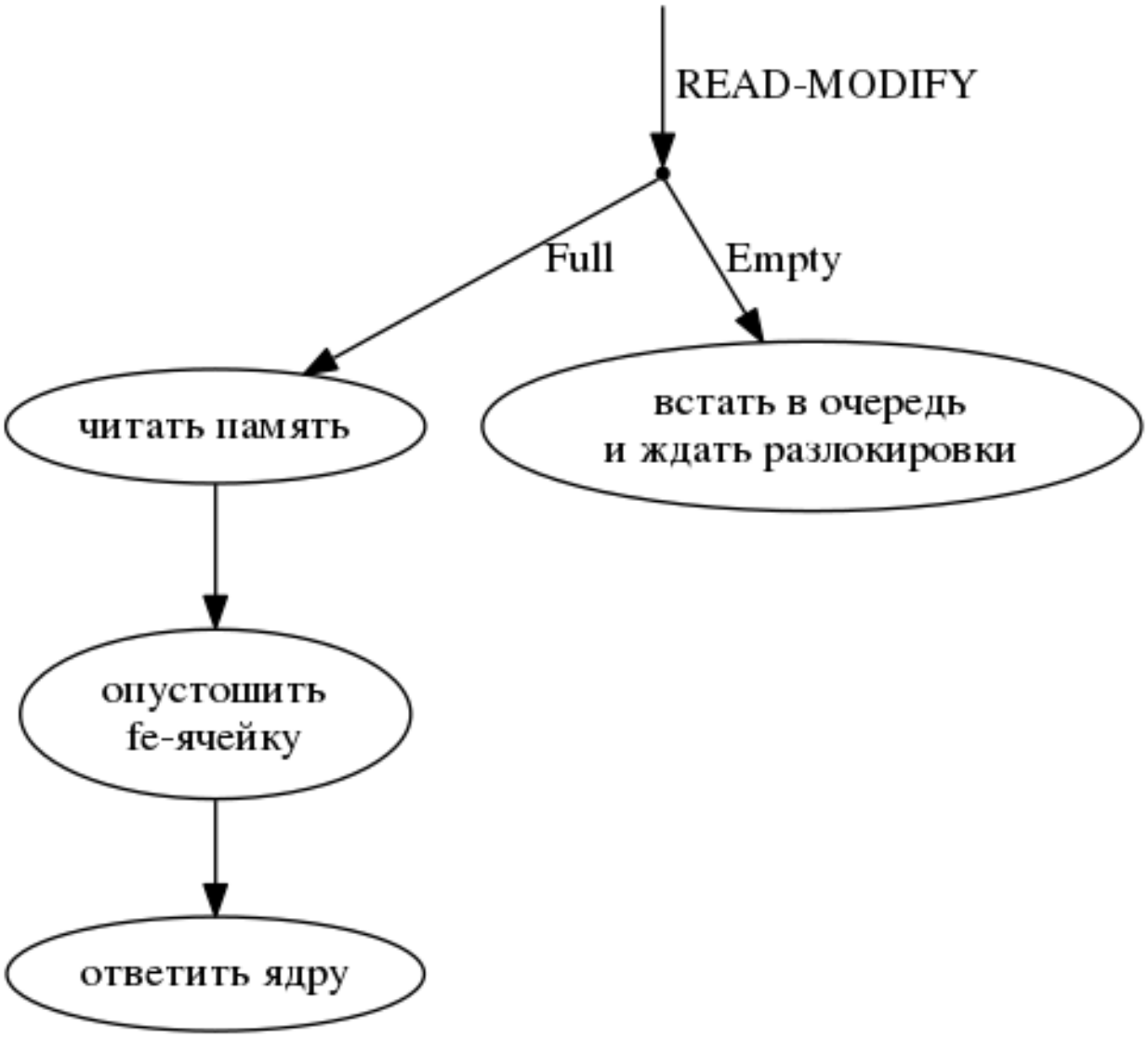

Рис. 2.6: Схема работы операции READ-MODIFY для FE-битов

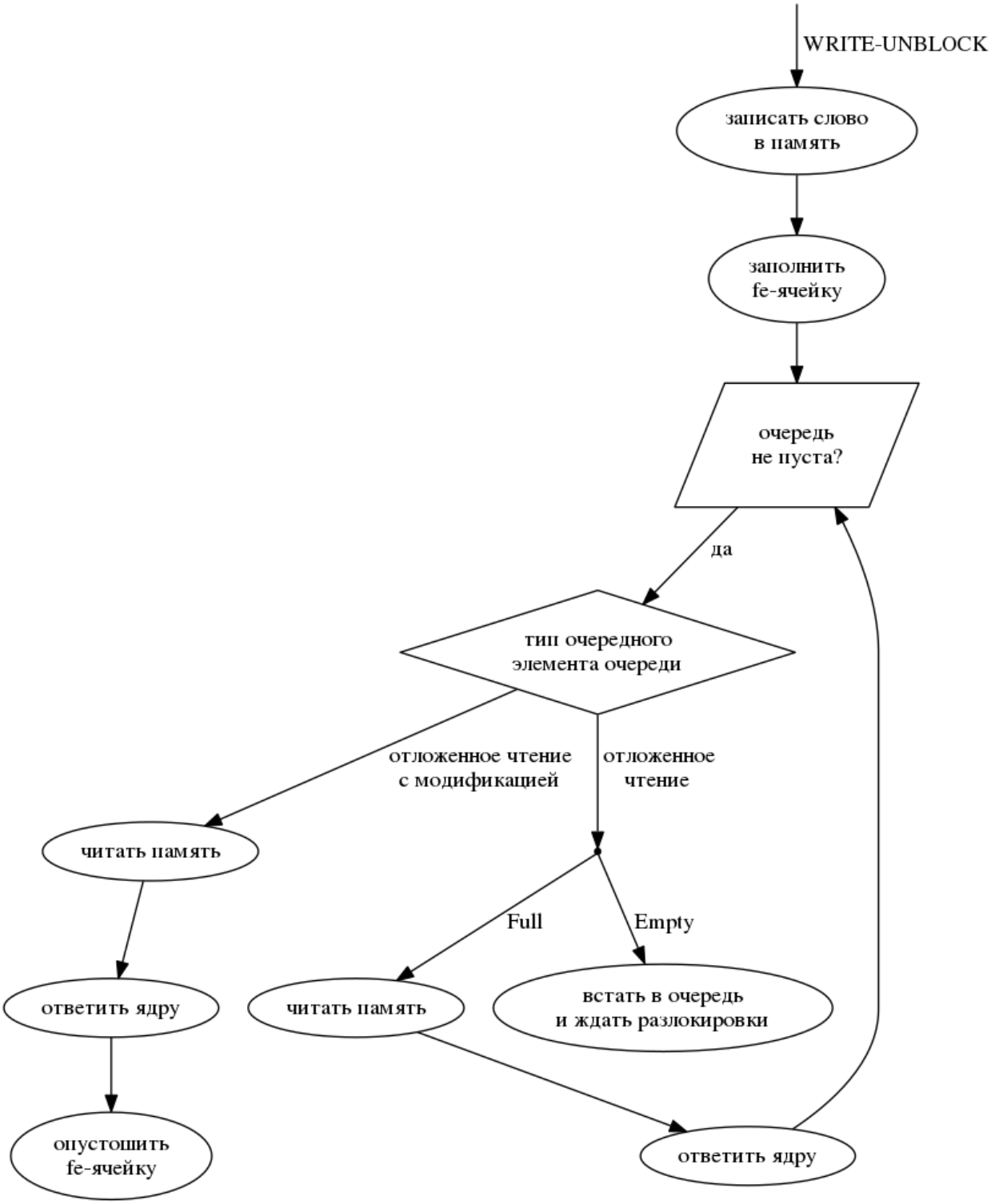

Рис. 2.7: Схема работы операции WRITE-UNBLOCK для FE-битов

ГЛАВА 3

Библиотека Libmalt

Libmalt – библиотека, содержащая функции, специфичные для архитектуры MALT. Для использования функций в C-коде необходимо подключить заголовочный файл libmalt.h, а при линковке – статическую библиотеку libmalt.a. Для использования функций в C++-коде необходимо подключить заголовочный файл libmalt.h, а при линковке – статическую библиотеку libmalt_plus.a совместно с библиотекой libmalt.a. Исходные коды библиотек находятся в *sw/lib/libmalt*, а заголовочные файлы в *sw/include/malt*.

3.1 Использование в программах

Библиотеки при сборке помещаются по адресу *sw/lib/libmalt.a* и *sw/lib/libmalt_plus.a* и автоматически прилинковываются к программе (при использовании Makefile из примеров). Для использования функций библиотеки в программе необходимо подключить заголовочный файл **libmalt.h**.

При старте ядро-супермастер использует стек в общей памяти, а ведомые ядра – в своей локальной памяти.

3.2 Заголовочные файлы

Таблица 3.1: Заголовочные файлы

libmalt.h	Заголовочный файл верхнего уровня, в программы необходимо включать только его, в нём подключаются все остальные. Для добавления новой функции необходимо редактировать заголовочный файл той подсистемы, к которой она относится
malt_addr	Основные системные константы – адреса устройств, регистров и основные константы. Этот файл необходимо менять при каждом изменении memory_map любого ядра
malt_periph.h	Функции и макросы для работы с периферийными устройствами и контроллером прерываний ведущего ядра, функции и макросы для получения некоторых системных величин
malt_ic.h	Функции и макросы интерконнекта – запуск потоков и взаимодействие между ядрами
malt_spbus.h	Функции и определения для взаимодействия с системной шиной SPbus. Описание структуры пакета шины spbus_packet
malt_fe.h	Управление памятью с дополнительными признаками
malt_proc.h	Функции, связанные с процессором
malt_asm.h	Функции и макросы для поддержки использования языка ассемблера
malt_cdefs.h	Удобные макросы для программирования на C
malt_debug.h	Средства для поддержки отладки
malt_mutex.h	Мьютексы и операции для них
types.h	Базовые типы, используемые в системной библиотеке

3.3 Функции для работы с периферией

3.3.1 malt_read_reg

Определение:

uint32_t **malt_read_reg**(uint32_t **a*)

Описание: Функция чтения регистра по адресу **a**.

Возвращаемое значение: Возвращает беззнаковое целое 32-х битное значение из регистра.

3.3.2 malt_write_reg

Определение:

void **malt_write_reg**(uint32_t **a*, uint32_t *v*)

Описание: Функция записи значения **v** в регистр по адресу **a**.

Возвращаемое значение: Нет.

3.3.3 malt_read_gpio

Определение:

void **malt_read_gpio**(uint32_t **a*)

Описание: Функция чтения регистра GPIO (адрес **a** игнорируется).

Возвращаемое значение: Функция возвращает целое 32-х битное значение – содержимое регистра GPIO. Доступна только на ядре-супермастере.

3.3.4 malt_write_gpio

Определение:

void **malt_write_gpio**(uint32_t **a*, uint32_t *v*)

Описание: Функция записи значения **v** в регистр GPIO (адрес **a** игнорируется). Доступна только на ядре-супермастере.

Возвращаемое значение: Нет.

3.3.5 malt_get_time_us

Определение:

uint32_t **malt_get_time_us**()

Описание: Получение значения счётчика микросекунд с момента запуска системы.

Возвращаемое значение: Функция возвращает 32-х битное значение в микросекундах, которое переполняется примерно каждые 1.2 часа.

3.3.6 malt_get_time_us_64

Определение:

u64_t **malt_get_time_us_64**()

Описание: Получение значения счётчика микросекунд с момента запуска системы.

Возвращаемое значение: Функция возвращает 64-х битное значение в микросекундах.

3.3.7 malt_get_int_num

Определение:

uint32_t **malt_get_int_num**()

Описание: Получение номера текущего прерывания (только внутри обработчика прерывания на супермастере). Доступна только на ядре-супермастере.

Возвращаемое значение: 32-х битное значение - номер прерывания.

3.3.8 malt_set_interrupt_mask

Определение:

void **malt_set_interrupt_mask**(uint32_t *m*)

Описание: Установка маски прерываний **m** на супермастере. Доступна только на ядре-супермастере.

Возвращаемое значение: Нет.

3.3.9 malt_get_interrupt_mask

Определение:

uint32_t **malt_get_interrupt_mask**()

Описание: Получение маски прерываний на супермастере. Доступна только на ядре-супермастере.

Возможные маски прерываний:

- **INTMSK_INTERCON** – маска прерывания интерконнекта;
- **INTMSK_TIMER** – маска прерывания от таймера;
- **INTMSK_UART** – маска прерывания от UART;
- **INTMSK_ETHERNET** – маска прерывания от ETHERNET.

Возвращаемое значение: 32-х битное значение маски.

3.3.10 malt_enable_interrupts

Определение:

void **malt_enable_interrupts**()

Описание: Функция включения прерываний на текущем ядре.

Возвращаемое значение: Нет.

3.3.11 malt_set_interrupt_timer_period

Определение:

void **malt_set_interrupt_timer_period**(uint32_t *period*)

Описание: Установка таймера прерываний значением **period** в мс. Доступна только на ядре-супермастере.

Возвращаемое значение: Нет.

3.3.12 malt_disable_interrupts

Определение:

void **malt_disable_interrupts**()

Описание: Функция отключения прерываний на текущем ядре.

Возвращаемое значение: Нет.

3.3.13 malt_save_interrupt_state

Определение:

void **malt_save_interrupt_state**(uint32_t *ADDR*)

Описание: Функция сохранения текущего статуса процессорного флага разрешения прерываний в переменной **ADDR** типа слово (4 байта). Эту переменную нужно создать до вызова функции.

Возвращаемое значение: Нет.

3.3.14 malt_restore_interrupt_state

Определение:

void **malt_restore_interrupt_state**(uint32_t *ADDR*)

Описание: Функция восстановления процессорного флага разрешения прерываний из переменной **ADDR** типа слово (4 байта).

Возвращаемое значение: Нет.

3.3.15 malt_invalidate_icache

Определение:

void **malt_invalidate_icache**()

Описание: Функция сброса кеша инструкций.

Возвращаемое значение: Нет.

3.3.16 malt_set_smst_inthandler

Определение:

void **malt_set_smst_inthandler**(f)

Описание: Установка функции **f**, которая будет вызываться после каждого вызова прерывания. При f = 0 происходит сброс текущей функции-прерывания. Доступна только на супермастере.

Возвращаемое значение: Нет.

3.3.17 malt_uart_rx_waiting

Определение:

int **malt_uart_rx_waiting**()

Описание: Функция проверяет наличие данных в приемном FIFO UART. Доступна только на ядре-супермастере.

Возвращаемое значение: Функция возвращает целое значение равное *0*, если FIFO пуст и отличное от нуля если в FIFO есть данные.

3.4 Функции - задержки

3.4.1 malt_sleep_us

Определение:

void **malt_sleep_us**(unsigned int *us*)

Описание: Задержка на us микросекунд.

Возвращаемое значение: Нет.

3.4.2 malt_sleep_ms

Определение:

void **malt_sleep_ms**(unsigned int *ms*)

Описание: Задержка на ms милисекунд.

Возвращаемое значение: Нет.

3.4.3 malt_sleep_pcnt_us

Определение:

void **malt_sleep_pcnt_us**(u64_t *us*)

Описание: Задержка на us микросекунд. Функция запускает счетчик производительности.

Возвращаемое значение: Нет.

3.5 Функции расширенного контроллера памяти

3.5.1 malt_ext_memop_blockread

Определение:

void **malt_ext_memop_blockread**(void *loc_addr*, void *glb_addr*, uint32_t *wrd_cnt*)

Описание: Чтение блока памяти размером **wrd_cnt** 32-битных слов из глобальной памяти по адресу **glb_addr** в локальную по адресу **loc_addr**. Глобальный и локальный адреса должны быть выровнены по границе слова. Вызов является неблокирующим; чтобы убедиться, что все чтения завершены, рекомендуется использовать макрос **malt_wait_breads_ready()**.

Возвращаемое значение: Нет.

3.5.2 malt_ext_memop_memcpy

Определение:

void **malt_ext_memop_memcpy**(uint32_t *dst*, uint32_t *src*, uint32_t *bytes*)

Описание: Копирование блока памяти размером **bytes** байт с адреса **src** на адрес **dst** внутри РКП. Адреса должны быть выровнены по границе слова,размер блока не должен превышать 2^16 байт. Для пользовательских задач рекомендуется использовать обёртку **malt_memcpy**.

Возвращаемое значение: Нет.

3.5.3 malt_ext_memop_memset

Определение:

void **malt_ext_memop_memset**(uint32_t *dst*, uint32_t *val*, uint32_t *bytes*)

Описание: Запись **bytes** байт, начиная с адреса **dst**, повторяющимся 32-битным словом **val**.

Возвращаемое значение: Нет.

3.5.4 malt_ext_memop_inc

Определение:

i32_t **malt_ext_memop_inc**(void* *addr*, i16_t *data*)

Описание: Атомарное прибавление к ячейке памяти по адресу **addr** значения **data** (16 бит).

Возвращаемое значение: Возвращает результат выполнения.

3.5.5 malt_breads_pending

Определение:

uint32_t **malt_breads_pending**()

Описание: Получение незавершенных блочных чтений на данном ядре.

Возвращаемое значение: Возвращает 32-х битное целое значение.

3.5.6 malt_all_breads_ready

Определение:

int **malt_all_breads_ready**()

Описание: Получение статуса блочных чтений.

Возвращаемое значение: Возвращает истину, если нет незавершённых блочных чтений, иначе возвращает ложь.

3.5.7 malt_wait_breads_ready

Определение:

void **malt_wait_breads_ready**()

Описание: Функция, останавливающая выполнение до завершения всех блочных чтений на данном ядре.

Возвращаемое значение: Нет.

3.5.8 malt_memset

Определение:

void ***malt_memset**(void *s, int *c*, size_t *n*)

Описание: Функция записывает первые **n** байт по адресу **s** константными байтами **c**. Аналогична *memset*, но для ускорения использует **malt_ext_memop_memset**.

Возвращаемое значение: Функция возвращает указатель на записанную память **s**.

3.5.9 malt_memcpy

Определение:

void ***malt_memcpy**(void **s1*, const void **s2*, size_t *n*)

Описание: Функция копирует **n** байт с адреса **s2** на адрес **s1**. Аналогична *memcpy*, но для ускорения использует **malt_ext_memop_memcpy**. Единственным ограничением является нахождение обоих адресов в глобальной памяти.

Возвращаемое значение: Функция возвращает указатель на **s1**.

3.5.10 malt_read_modify

Определение:

uint32_t **malt_read_modify**(void* *addr*)

Описание: Выполняет опреацию контроллера памяти *READ-MODIFY*. Читает слово по адресу **addr** в глобальной памяти с переключением FE-бита согласно схеме операции *читать и модифицировать* для FE-битов.

Возвращаемое значение: Возвращает 32-х битное целое значение прочитанное по адресу **addr**.

3.5.11 malt_fill_write

Определение:

void **malt_fill_write**(volatile void* *addr*, volatile uint32_t *data*)

Описание: Выполняет операцию контроллера памяти *WRITE-UNBLOCK*. Освобождает все заблокированные операции чтения, записывает данные **data** по указанному адресу **addr** в глобальной памяти, согласно схеме операции *писать и разблокировать* для FE-битов.

Возвращаемое значение: Нет.

3.5.12 malt_putchar

Определение:

void **malt_putchar**(int *c*)

Описание: Вывод значения **c** в UART. Отправляется младший байт.

Возвращаемое значение: Нет.

3.5.13 malt_getchar

Определение:

int **malt_getchar**()

Описание: Получение байта из приемного буффера FIFO UART по готовности.

Возвращаемое значение: Функция возвращает целое значение содержащее байт из приемного буффера UART.

3.6 Функции MALT-интерконнекта

3.6.1 malt_core_line_to_num

Определение:

int **malt_core_line_to_num**(int *l*, int *c*)

Описание: Получение интерконнект-номера ядра по номеру линии **l** и номеру ядра **c**.

Возвращаемое значение: Функция возвращает целое значение – интерконнект-номер ядра.

3.6.2 malt_cur_core_num

Определение:

uint32_t **malt_cur_core_num**()

Описание: Получение интерконнект-номера (адрес, например, 0x201) текущего ядра.

Возвращаемое значение: Функция возвращает 32-х битное целое значение - интерконнект-номер.

3.6.3 malt_cur_core_line

Определение:

int32_t **malt_cur_core_line**()

Описание: Получение номера интерконнект-цепочки текущего ядра.

Возвращаемое значение: Функция возвращает 32-х битное целое значение - номер интерконнект-цепочки текущего ядра, например, для ядра с адресом 0x201 должно возвращаться 2.

3.6.4 malt_seq_core_num

Определение:

int32_t **malt_seq_core_num**(int32_t *core*)

Описание: Получение порядкового номера ядра.

Возвращаемое значение: Функция возвращает 32-х битное целое значение - порядковый номер ядра **core**. Например, в системе с 4-я мастерами и 2 подчинёнными ядрами в цепочке для ядра 0x201 должно возвращаться 5, для ядра 0x202 - 6, для ядра 0x300 - 7 и т. д.

3.6.5 malt_seq_slv_num

Определение:

int32_t **malt_seq_slv_num**(uin32_t *core*)

Описание: Получение порядкового номера подчиненного ядра.

Возвращаемое значение: Функция возвращает 32-х битное целое значение - порядковый номер подчинённого ядра *core*, считая только подчинённые. В системе с 4-я мастерами и 2 подчинёнными ядрами в цепочке для ядра 0x201 должно возвращаться 2, для ядра 0x202 - 3, для ядра 0x301 - 4 и т. д.

3.6.6 malt_slv_num

Определение:

int32_t **malt_slv_num**(int32_t *core*)

Описание: Получение номера подчиненного ядра на цепочке **core**.

Возвращаемое значение: Возвращает 32-х битное целое значение - номер ядра, например, для ядра с адресом 0x201 должно возвращаться 1.

3.6.7 malt_seq_cur_slv_num

Определение:

int32_t **malt_seq_cur_slv_num**()

Описание: Получение порядкового номера подчиненного ядра на цепочке текущего ядра.

Возвращаемое значение: Возвращает 32-х битное целое значение - номер подчиненного ядра,

3.6.8 malt_core_line_num

Определение:

int32_t **malt_core_line_num**(int32_t *core*)

Описание: Получение номера цепочки ядра.

Возвращаемое значение: Возвращает 32-х битное целое значение.

3.6.9 malt_core_master_num

Определение:

int32_t **malt_core_master_num**(int32_t *core*)

Описание: Получение номера мастер-ядра, к которому подключено ядро **core**.

Возвращаемое значение: Функция возвращает 32-х битное целое значение - номер мастер-ядра.

3.6.10 malt_is_same_line

Определение:

int **malt_is_same_line**(int32_t *core*)

Описание: Проверка расположения ядра core на одной цепочке с ядром.

Возвращаемое значение: Функция возвращает истину, если **core** находится на одной цепочке с текущим ядром.

3.6.11 malt_mst_on_line

Определение:

int32_t **malt_mst_on_line**(line)

Описание: Получение номера мастер-ядра на цепочке **line**.

Возвращаемое значение: Функция возвращает 32-х битное целое значение - номер мастер-ядра.

3.6.12 malt_is_smaster

Определение:

int **malt_is_smaster**()

Описание: Функция-макрос проверяет на каком ядре происходит выполнение.

Возвращаемое значение: Возвращает истину, если она исполняется на супермастере.

3.6.13 malt_is_master

Определение:

int **malt_is_master**()

Описание: Функция-макрос проверяет на каком ядре происходит выполнение.

Возвращаемое значение: Функция возвращает истину, если она исполняется на мастер-ядре.

3.6.14 malt_core_is_smaster

Определение:

int **malt_core_is_smaster**(int32_t *core*)

Описание: Функция-макрос проверяет тип ядра core

Возвращаемое значение: Возвращает истину, если core супермастер-ядро.

3.6.15 malt_core_is_master

Определение:

int **malt_core_is_master**(int32_t *core*)

Описание: Функция-макрос проверяет тип ядра core

Возвращаемое значение: Возвращает истину, если core является мастер-ядром.

3.6.16 malt_core_is_slave

Определение:

int **malt_core_is_slave**(int32_t *core*)

Описание: Функция-макрос проверяет тип ядра core

Возвращаемое значение: Возвращает истину, если core *slave* ядро.

3.6.17 malt_chk_ic_oflow

Определение:

uint32_t **malt_chk_ic_oflow**()

Описание: Функция проверяет наличие ошибок интерконнекта. После чтения статуса сбрасывает регистр ошибок.

Возвращаемое значение: Функция возвращает 32 разрядное беззнаковое целое значение. Если ошибки нет возвращает *0*. В случае ошибки возвращает ненулевое значение - код из регистра ошибок.

3.6.18 malt_slv_end_thread

Определение:

void **malt_slv_end_thread**()

Описание: Функция при запуске на подчинённом ядре завершает текущий поток.

Возвращаемое значение: Нет.

3.6.19 malt_all_slaves_free

Определение:

int **malt_all_slaves_free**()

Описание: Проверка состояния подчинённых ядер.

Возвращаемое значение: Функция возвращает истину, если все подчинённые ядра данного мастера свободны.

3.6.20 malt_all_slaves_busy

Определение:

int **malt_all_slaves_busy**()

Описание: Проверка состояния подчинённых ядер.

Возвращаемое значение: Функция возвращает истину, если все подчинённые ядра данного мастера заняты.

3.6.21 malt_send_ic

Определение:

int **malt_send_ic**(ic_packet **icp*, const u16_t *trg*)

Описание: Отправка интерконнект-пакета *ic_packet*, содержащего два слова *data0* и *data1* ядру **trg** по адресу **icp**.

Возвращаемое значение: В случае успеха возвращает *0*, в случае ошибки возвращает отрицательное значение.

3.6.22 malt_rec_ic

Определение:

int **malt_send_ic**(ic_packet *_icp_)

Описание: Функция для получения интерконнект-пакета *ic_packet*, содержащего два слова *data0* и *data1* и по адресу **icp**.

Возвращаемое значение: Функция возвращает номер ядра-отправителя пакетов при успешном завершении и отрицательное значение при неуспехе, а именно -1 – при отсутствии пакетов или -2 – при сбросе пакета.

3.6.23 malt_rec_ic_lock

Определение:

int **malt_rec_ic_lock**(ic_packet *_icp_)

Описание: Функция для получение интерконнект-пакета *ic_packet*, содержащего два слова *data0* и *data1* и по адресу **icp** с блокировкой. Аналогична *malt_rec_ic*, но возврат из функции не произойдет до получения пакета.

Возвращаемое значение: Функция возвращает номер ядра-отправителя пакетов при успешном завершении и отрицательное значение при неуспехе, а именно -1 – при отсутствии пакетов или -2 – при сбросе пакета.

3.6.24 malt_rec_ic_from

Определение:

int **malt_send_ic_from**(ic_packet *_icp_, u16_t *src*)

Описание: Функция для получения интерконнект-пакета *ic_packet*, содержащего два слова *data0* и *data1* по адресу **icp** от ядра **src**.

Возвращаемое значение: Функция возвращает номер ядра-отправителя пакетов при успешном завершении и отрицательное значение при неуспехе, а именно -1 – при отсутствии пакетов или -2 – при сбросе пакета.

3.6.25 malt_rec_ic_from_lock

Определение:

int **malt_rec_ic_lock**(ic_packet *_icp_, u16_t *src*)

Описание: Функция для получение интерконнект-пакета *ic_packet*, содержащего два слова *data0* и *data1* по адресу **icp** от ядра **src** с блокировкой. Аналогична *malt_rec_ic*, но возврат из функции не произойдет до получения пакета.

Возвращаемое значение: Функция возвращает номер ядра-отправителя пакетов при успешном завершении и отрицательное значение при неуспехе, а именно -1 – при отсутствии пакетов или -2 – при сбросе пакета.

3.6.26 malt_send_ic_data

Определение:

int **malt_send_ic_data**(int *core*, uint32_t *data0*, uint32_t *data1*)

Описание: Отправка интерконнект-сообщения, содержащего два слова **data0** и **data1** ядру **core**,

Возвращаемое значение: В случае успеха возвращает *0*, в случае ошибки возвращает отрицательное значение.

3.6.27 malt_rec_ic_notification_from

Определение:

int **malt_rec_ic_notification_from**(u16_t *core_addr*)

Описание: Функция позволяет получать пакеты-уведомления только от заданного ядра-отправителя **core_addr**. Может использоваться только с MALT, начиная с аппаратной версии 1.8.

Возвращаемое значение: Возвращает целое значение – адрес отправителя.

3.6.28 malt_rec_ic_notification

Определение:

int **malt_rec_ic_notification**()

Описание: Упрощенный вариант функции malt_rec_ic_notification_from() позволяет получать пакеты-уведомления от любого ядра-отправителя. Может использоваться только с MALT, начиная с аппаратной версии 1.8.

Возвращаемое значение: Возвращает целое значение – адрес отправителя.

3.6.29 malt_ic_notify

Определение:

void **malt_ic_notify**(core)

Описание: Отправка интерконнект-сообщения ядру **core** типа IC_SPB_STYPE_NOTIFY.

Возвращаемое значение: Нет.

3.6.30 malt_all_lines_free

Определение:

int **malt_all_lines_free**()

Описание: Проверка статуса slave-ядер. Используется на ядре-супермастере

Возвращаемое значение: Функция возвращает *1*, если все slave-ядра системы свободны, иначе возвращает 0.

3.6.31 malt_wait_all_lines_free

Определение:

int **malt_wait_all_lines_free**()

Описание: Функция, ожидающая освобождения всех подчиненных ядер. Используется только на супермастере.

Возвращаемое значение: Функция сегда возвращает 0.

3.6.32 malt_sm_wait_all_lines_free

Определение:

int **malt_sm_wait_all_lines_free**()

Описание: Более быстрый вариант функции malt_wait_all_lines_free. Использует malt_local_stack_call() для вызова malt_wait_all_lines_free().

Возвращаемое значение: Возвращает целое значение.

3.6.33 malt_sm_putchar

Определение:

void **malt_sm_putchar**(int *c*)

Описание: Вывод байта данных в UART гарантировано через супермастер ядро

Возвращаемое значение: Нет.

3.6.34 malt_start_thread_this_line

Определение:

int **malt_start_thread_this_line**(int(**fp*)(void), uint32_t *iseg*, uint32_t *dseg*)

Описание: Запуск потока на одном из свободных подчинённых ядер текущего семейства (доступна только на мастерах). Поток начнёт выполняться с функции fp с сегментными регистрами iseg и dseg.

Возвращаемое значение: Функция в случае успешного запуска возвращает номер ядра, в случае ошибки отрицательное значение.

3.6.35 malt_start_thread_line

Определение:

int **malt_start_thread_line**(int *line*, int(**fp*)(void), uint32_t *iseg*, uint32_t *dseg*)

Описание: Запуск потока на одном свободном подчинённом ядре цепочки line.

Возвращаемое значение: Функция возвращает номер запущенного ядра или *-1* при неуспехе запуска.

3.6.36 malt_start_thread

Определение:

int **malt_start_thread**(int(**fp*)(void), uint32_t *iseg*, uint32_t *dseg*)

Описание: Запуск потока на одном из свободных подчинённых ядер системы.

Возвращаемое значение: Функция возвращает номер запущенного ядра или *-1* при неуспехе запуска.

3.6.37 malt_start_thr

Определение:

int **malt_start_thr**(void* *fp*, uint32_t *n*, ...)

Описание: Запуск потока на одном из свободных подчинённых ядер системы, Параметры, начиная со второго – это параметры вызова функции, задаваемой указателем **fp**. Все параметры передаются как целые типа *uint32_t*. Число параметров должно быть не большим 8.

Возвращаемое значение: Функция возвращает номер запущенного ядра или *-1* при неуспехе запуска.

3.6.38 malt_start_full_line

Определение:

int **malt_start_full_line**(int *line*, int(**fp*)(void))

Описание: Запуск одинаковой функции **fp** на всех свободных подчинённых ядрах цепочки line.

Возвращаемое значение: Функция возвращает количество запущенных ядер.

3.6.39 malt_start_all_slaves

Определение:

int **malt_start_all_slaves**(int(**fp*)(void))

Описание: Запуск одинаковой функции **fp** на всех свободных подчинённых ядрах системы.

Возвращаемое значение: Функция возвращает количество запущенных ядер.

3.6.40 malt_mst_end_thread

Определение:

int **malt_mst_end_thread**(uint32_t *core*)

Описание: Завершения работы потока на slave-ядре **core**. Выполняется на master-ядре цепочки, которой принадлежит slave-ядро

Возвращаемое значение: Функция возвращает *0* в случае успеха, в случае ошибки возвращает *-1*

3.6.41 malt_ic_set_buffer_size_sm

Определение:

int **malt_ic_set_buffer_size_sm**(uint32_t *size*)

Описание: Изменение размера буфера интерконнекта на ядре типа Supermaster. Размер задаётся в пакетах и должен быть числом-степенью 2. Размер буфера по умолчанию равен 512 байтам или 32 пакетам. Предельный размер – 4096 байт, 256 пакетов. Не следует делать размер слишком маленьким. Размер буфера интерконнекта рекомендуется устанавливать в самом начале работы. При необходимости перенесения стека в быструю локальную память его следует делать уже после изменения размера буфера. Размер буфера и локальной памяти связаны: чем больше буфер, тем меньше локальная память. Эта функция доступна только с аппаратной версии 1.8.

Возвращаемое значение: Функция возвращает отрицательное значение при ошибочном завершении и *1* при успешном.

3.6.42 malt_ic_set_buffer_size_masters

Определение:

int **malt_ic_set_buffer_size_masters**(uint32_t *size*)

Описание: Изменить размер буфера интерконнекта на всех ядрах типа Master. Размер задаётся в пакетах и должен быть числом-степенью 2. Размер буфера по умолчанию равен 512 байтам или 32 пакетам. Предельный размер – 4096 байт, 256 пакетов. Не следует делать размер слишком маленьким. Размер буфера интерконнекта рекомендуется устанавливать в самом начале работы при отсутствии загруженных Slave-ядер, так как при изменении размера буфера выполняется перезапуск всех Master-ядер. Размер буфера и стека связаны: чем больше буфер, тем меньше локальная память. Доступна только с аппаратной версии 1.8.

Возвращаемое значение: Функция возвращает отрицательное значение при ошибочном завершении и *1* при успешном.

3.6.43 malt_ic_set_buffer_size_slaves

Определение:

int **malt_ic_set_buffer_size_slaves**(uint32_t *size*)

Описание: Изменить размер буфера интерконнекта на всех ядрах типа Slave. Размер задаётся в пакетах и должен быть числом-степенью 2. Размер буфера по умолчанию равен 512 байтам или 32 пакетам. Предельный размер – 4096 байт, 256 пакетов. Не следует делать размер слишком маленьким. Размер буфера интерконнекта рекомендуется устанавливать в самом начале работы, так как при изменении размера буфера выполняется перезапуск всех Slave-ядер. Размер буфера и стека связаны: чем больше буфер, тем меньше локальная память. Доступна только с аппаратной версии 1.8.

Возвращаемое значение: Функция возвращает отрицательное значение при ошибочном завершении и *1* при успешном.

3.6.44 malt_get_ic_buf_size

Определение:

int **malt_get_ic_buf_size**(uint32_t *core*)

Описание: Получить размер буфера интерконнекта на ядре *core* в пакетах. Система MALT не позволяет узнать размер этого буфера на супермастере с других ядер.

Возвращаемое значение: Функция при неудаче возвращает отрицательное значение, а при успехе – размер буфера.

3.7 Функции для работы с мьютексами

3.7.1 malt_inc_mutex_lock

Определение:

void **malt_inc_mutex_lock**(malt_mutex* *mx*)

Описание: Операции захвата мьютекса **mx** (без поддержки рекурсии) на базе атомарной операции инкремент.

Возвращаемое значение: Нет.

3.7.2 malt_inc_mutex_unlock

Определение:

void **malt_inc_mutex_unlock**(malt_mutex* *mx*)

Описание: Операции освобождения мьютекса **mx** (без поддержки рекурсии) на базе атомарной операции инкремент.

Возвращаемое значение: Нет.

3.7.3 malt_fe_mutex_lock

Определение:

void **malt_fe_mutex_lock**(malt_mutex* *mx*)

Описание: Операции захвата мьютекса **mx** (без поддержки рекурсии) на базе операций READ-MODIFY и WRITE-UNBLOCK с FE-битами.

Возвращаемое значение: Нет.

3.7.4 malt_fe_mutex_unlock

Определение:

void **malt_fe_mutex_unlock**(malt_mutex* *mx*)

Описание: Операции освобождения мьютекса **mx** (без поддержки рекурсии) на базе операций READ-MODIFY и WRITE-UNBLOCK с FE-битами.

Возвращаемое значение: Нет.

3.7.5 malt_nohw_mutex_lock

Определение:

void **malt_nohw_mutex_lock**(malt_mutex* *mx*)

Описание: Операции захвата мьютекса **mx** (без поддержки рекурсии), реализованного без использования специальных аппаратных средств.

Возвращаемое значение:

3.7.6 malt_nohw_mutex_unlock

Определение:

void **malt_nohw_mutex_unlock**(malt_mutex* *mx*)

Описание: Операции освобождения мьютекса **mx** (без поддержки рекурсии), реализованного без использования специальных аппаратных средств.

Возвращаемое значение: Нет.

3.7.7 malt_inc_recursive_mutex_lock

Определение:

void **malt_inc_recursive_mutex_lock**(malt_mutex* *mx*)

Описание: Операции захвата мьютекса **mx** с поддержкой рекурсии на базе атомарной операции инкремент.

Возвращаемое значение: Нет.

3.7.8 malt_inc_recursive_mutex_unlock

Определение:

void **malt_inc_recursive_mutex_unlock**(malt_mutex* *mx*)

Описание: Операции освобождения мьютекса **mx** с поддержкой рекурсии на базе атомарной операции инкремент.

Возвращаемое значение: Нет.

3.7.9 malt_nohw_recursive_mutex_lock

Определение:

void **malt_nohw_recursive_mutex_lock**(malt_mutex* *mx*)

Описание: Операции захвата мьютекса **mx** с поддержкой рекурсии, реализованные без использования специальных аппаратных средств.

Возвращаемое значение: Нет.

3.7.10 malt_nohw_recursive_mutex_unlock

Определение:

void **malt_nohw_recursive_mutex_unlock**(malt_mutex* *mx*)

Описание: Операции освобождения мьютекса **mx** с поддержкой рекурсии, реализованные без использования специальных аппаратных средств.

Возвращаемое значение: Нет.

3.8 Информация о системе

3.8.1 malt_print_system_config

Определение:

void **malt_print_system_config**()

Описание: Функция печатает в терминал максимально подробную информацию о системе, на которой она была запущена.

Возвращаемое значение: Нет.

3.8.2 malt_get_slv_busy_reg

Определение:

uint64_t **malt_get_slv_busy_reg**()

Описание: Функция на мастер-ядре возвращает статус занятости всех подчинённых ядер этой цепочки, по биту на ядро.

Возвращаемое значение: Функция возвращает целое 64-битное значение, в котором каждый бит соответствует состоянию подчиненных ядер: *0* - ядро свободно, *1* - занято.

3.8.3 malt_get_slaves_count

Определение:

uint32_t **malt_get_slaves_count**()

Описание: Получение числа подчинённых ядер в одной цепочке (без учёта мастера).

Возвращаемое значение: Функция возвращает целое значение – число ядер.

3.8.4 malt_get_lines_count

Определение:

uint32_t **malt_get_lines_count**()

Описание: Получение числа цепочек в системе.

Возвращаемое значение: Функция возвращает целое значение – количество цепочек в системе.

3.8.5 malt_get_total_slaves

Определение:

uint32_t **malt_get_total_slaves**()

Описание: Получение полного количества подчинённых ядер в системе.

Возвращаемое значение: Функция возвращает целое значение – полное количество подчинённых ядер в системе.

3.8.6 malt_get_shared_mem_size

Определение:

uint32_t **malt_get_shared_mem_size**()

Описание: Получение размера памяти разделяемой между всеми ядрами системы в байтах. Иначе говоря, размер глобальной памяти.

Возвращаемое значение: Функция возвращает целое значение – размер памяти.

3.8.7 malt_get_local_mem_size

Определение:

uint32_t **malt_get_local_mem_size**()

Описание: Получение полного размера локальной памяти на ядре за вычетом приёмного буфера шины.

Возвращаемое значение: Функция возвращает целое значение – размер памяти.

3.8.8 malt_get_version

Определение:

uint32_t **malt_get_version**()

Описание: Получение полного кода аппаратной ревизии системы MALT.

Возвращаемое значение: Функция возвращает 32-х разрядное целое значение – полный код аппаратной ревизии системы MALT.

3.8.9 malt_get_major_version

Определение:

uint32_t **malt_get_major_version**()

Описание: Получение старшего байта версии аппаратной ревизии системы MALT.

Возвращаемое значение: Возвращает целое значение, в младшем байте значение старшего байта аппаратной ревизии системы MALT.

3.8.10 malt_get_minor_version

Определение:

uint32_t **malt_get_minor_version**()

Описание: Получение младшего байта версии версии аппаратной ревизии системы MALT.

Возвращаемое значение: Возвращает целое значение, в младшем байте значение младшего байта аппаратной ревизии системы MALT.

3.8.11 malt_get_revision_version

Определение:

uint32_t **malt_get_revision_version**()

Описание: Получение версии ревизии MALT

Возвращаемое значение: Возвращает целое значение, в младшем байте значение версии аппаратной ревизии системы MALT.

3.8.12 malt_get_index_version

Определение:

uint32_t **malt_get_index_version**()

Описание: Получение индекса аппаратной ревизии системы MALT.

Возвращаемое значение: Возвращает целое значение, в младшем байте значение индекса аппаратной ревизии системы MALT.

3.8.13 malt_get_rom_version

Определение:

uint32_t **malt_get_rom_version**()

Описание: Получение полного кода версии ПЗУ инструкций ядер MALT.

Возвращаемое значение: Функция возвращает целое 32-х разрядное значение – полный код версии ПЗУ инструкций ядер MALT.

3.8.14 malt_get_major_rom_version

Определение:

uint32_t **malt_get_major_rom_version**()

Описание: Получение старшего байта версии ПЗУ инструкций ядер MALT.

Возвращаемое значение: Функция возвращает целое 32-х разрядное значение, в младшем байте старший байт версии ПЗУ.

3.8.15 malt_get_minor_rom_version

Определение:

uint32_t **malt_get_minor_rom_version**()

Описание: Получение младшего байта версии ПЗУ инструкций ядер MALT.

Возвращаемое значение: Функция возвращает целое 32-х разрядное значение, в младшем байте младший байт версии ПЗУ.

3.8.16 malt_get_sdk_version

Определение:

uint32_t **malt_get_sdk_version**()

Описание: Получение полного кода версии среды разработки (SDK) MALT_sw.

Возвращаемое значение: Функция возвращает целое 32-х разрядное значение – полный код версии среды разработки (SDK) MALT_sw.

3.8.17 malt_get_major_sdk_version

Определение:

uint32_t **malt_get_major_sdk_version**()

Описание: Получение двух старших байт кода версии среды разработки (SDK) MALT_sw.

Возвращаемое значение: Функция возвращает целое 32-х разрядное значение, содержащее 2 старших байта кода версии среды разработки (SDK) MALT_sw.

3.8.18 malt_get_minor_sdk_version

Определение:

uint32_t **malt_get_minor_sdk_version**()

Описание: Получение двух младших байт кода версии среды разработки (SDK) MALT_sw.

Возвращаемое значение: Функция возвращает целое 32-х разрядное значение, содержащее 2 младших байта кода версии среды разработки (SDK) MALT_sw.

3.8.19 malt_get_core_freq

Определение:

uint32_t **malt_get_core_freq**()

Описание: Получение частоты ядер общего назначения в кГц.

Возвращаемое значение: Функция возвращает целое 32-х разрядное значение – частота ядер общего назначения в кГц.

3.8.20 malt_get_bus_freq

Определение:

uint32_t **malt_get_bus_freq**()

Описание: Получение частоты шины SPBUS в кГц.

Возвращаемое значение: Функция возвращает целое 32-х разрядное значение – частота шины SPBUS в кГц.

3.8.21 malt_get_mem_freq

Определение:

uint32_t **malt_get_mem_freq**()

Описание: Получение значения частоты контроллера памяти в кГц.

Возвращаемое значение: Функция возвращает целое 32-х разрядное значение – частота контроллера памяти в кГц.

3.8.22 malt_get_coproc_freq

Определение:

uint32_t **malt_get_coproc_freq**()

Описание: Получение значения частоты сопроцессора в кГц.

Возвращаемое значение: Функция возвращает целое 32-х разрядное значение – частота сопроцессора в кГц.

3.8.23 malt_is_emu

Определение:

uint32_t **malt_is_emu**()

Описание: Проверка исполнения на эмуляторе.

Возвращаемое значение: Функция возвращает целое значение отличное от нуля в случае исполнения на эмуляторе и *0* в противном случае.

3.8.24 malt_is_rtlsim

Определение:

uint32_t **malt_is_rtlsim**()

Описание: Проверка исполнения на симуляторе.

Возвращаемое значение: Функция возвращает целое значение отличное от нуля в случае исполнения на симуляторе и *0* в противном случае.

3.8.25 malt_is_fpga

Определение:

uint32_t **malt_is_fpga**()

Описание: Проверка исполнения на FPGA.

Возвращаемое значение: Функция возвращает целое значение отличное от нуля в случае исполнения на FPGA и *0* в противном случае.

3.8.26 malt_is_pasic

Определение:

uint32_t **malt_is_pasic**()

Описание: Проверка исполнения на ASIC.

Возвращаемое значение: Функция возвращает целое значение отличное от нуля в случае исполнения на ASIC и *0* в противном случае.

3.8.27 malt_coproc_present

Определение:

uint32_t **malt_coproc_present**()

Описание: Проверка наличия в системе сопроцессоров.

Возвращаемое значение: Функция возвращает целое значение отличное от нуля в случае наличия в системе сопроцессоров и *0* в противном случае.

3.8.28 malt_get_coproc_num

Определение:

uint32_t **malt_get_coproc_num**()

Описание: Получение числа сопроцессоров в системе.

Возвращаемое значение: Функция возвращает 32-х разрядное значение, в младшем байте которого число сопроцессоров в системе.

3.8.29 malt_has_gepard_coproc

Определение:

uint32_t **malt_has_gepard_coproc**()

Описание: Проверка наличия в системе сопроцессоров Гепард.

Возвращаемое значение: Функция возвращает целое значение отличное от нуля в случае наличия в системе сопроцессоров Гепард и *0* в противном случае.

3.8.30 malt_has_progressor_coproc

Определение:

uint32_t **malt_has_progressor_coproc**()

Описание: Проверка наличия в системе сопроцессоров Прогрессор.

Возвращаемое значение: Функция возвращает целое значение отличное от нуля в случае наличия в системе сопроцессоров Прогрессор и *0* в противном случае.

3.8.31 malt_has_fe

Определение:

int **malt_has_fe**()

Описание: Проверка наличия в контроллере памяти поддержки FE-битов.

Возвращаемое значение: Функция возвращает целое значение отличное от нуля в случае наличия в контроллере памяти поддержки FE-битов и *0* в противном случае.

3.8.32 malt_has_secmemctl

Определение:

int **malt_has_secmemctl**()

Описание: Проверка наличия в системе второго контроллера памяти.

Возвращаемое значение: Функция возвращает истину в случае присутствия в системе двух контроллеров памяти с разных сторон цепочек.

3.8.33 malt_has_ethernet

Определение:

int **malt_has_ethernet**()

Описание: Проверка наличия в системе контроллера Ethernet.

Возвращаемое значение: Функция возвращает истину в случае присутствия в системе контроллера интерфейса Ethernet.

3.8.34 malt_has_spbuart

Определение:

int **malt_has_spbuart**()

Описание: Проверка наличия в системе контроллера UART на шине SPBUS.

Возвращаемое значение: Функция возвращает истину в случае присутствия в системе контроллера UART доступного всем ядрам через шину SPBUS.

3.8.35 malt_has_updatablerom

Определение:

int **malt_has_updatablerom**()

Описание: Проверка наличия в системе функциональности обновления ПЗУ инструкций (обновление сохраняется до сброса системы).

Возвращаемое значение: Функция возвращает истину в случае возможности замены кода, расположенного в ПЗУ инструкций.

3.8.36 malt_get_interrupts_state

Определение:

uint32_t **malt_get_interrupts_state**()

Описание: Функция проверяет запрет/разрешение прерываний.

Возвращаемое значение: Функция возвращает целое значение равное *0*, если прерывания запрещены, и отличное от нуля, если разрешены.

3.8.37 malt_soft_ic_buffer_show

Определение:

void **malt_soft_ic_buffer_show**(char **debugger_buffer*, int *extent*)

Описание: Распечатка содержимого дополнительного буфера интерконнекта вместе со значениями обоих связанных с этим буфером указателей в текстовый буфер **debugger_buffer**. Дополнительно об особенностях использования буфера можно прочитать в описании функции **malt_hard_ic_buffer_show()**. Параметр **extent** – это число дополнительно печатаемых пакетов в буфере. Если печатать с **extent** = *0*, то будут распечатаны данные только по пакетам, ещё неизвлечённым из дополнительного буфера. Если использовать **extent** > *0*, то будут ещё печататься данные по **extent** позициям в дополнительном буфере, где могли быть извлечённые недавно пакеты. Эта функция может использоваться только с MALT, начиная с аппаратной версии 1.8.

Возвращаемое значение: Нет.

3.8.38 malt_hard_ic_buffer_show

Определение:

void **malt_hard_ic_buffer_show**(char **debugger_buffer*)

Описание: Распечатка текущего содержимого аппаратного буфера интерконнекта в переменную-буфер **debugger_buffer**. Для корректного вызова этой функции нужно предварительное наличие такого буфера достаточного объема. Распечатка производится добавлением содержимого буфера к переменной-буферу. Таким образом, если нужно обеспечить печать «с чистого листа», то нужно занулить первый байт в буфере. Предлагается использовать размеры буфера от 10 килобайт. Вместе с содержимым буфера распечатывается также значение счётчика принимаемых пакетов, следующий пакет будет размещен по этому адресу в буфере.

Возвращаемое значение: Нет.

3.9 Отладочные функции и макросы в эмуляторе

3.9.1 malt_trace_watch

Определение:

void **malt_trace_watch**(uint32_t *addr*)

Описание: Функция включает *watch* по адресу глобальной памяти **addr**, а все обращения по этому адресу будут печататься в консоль эмулятора. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Нет.

3.9.2 malt_trace_opcode

Определение:

void **malt_trace_opcode**(e)

Описание: Функция при истинном e включает вывод дизассемблерного листинга исполняемых на данном ядре инструкций в файл maltemu.XXXX.log , при ложном – отключает. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Нет.

3.9.3 malt_trace_mcreadcnt

Определение:

uint32_t **malt_trace_mcreadcnt**()

Описание: Функция выполняет чтение счетчика в контроллере памяти. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Функция возвращает целое 32-х разрядное значение счетчика контроллера памяти.

3.9.4 malt_trace_mcwritecnt

Определение:

uint32_t **malt_trace_mcwritecnt**()

Описание: Функция выполняет чтение значения счетчика записи в контроллере памяти. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Функция возвращает целое 32-х разрядное значение записи счетчика контроллера памяти.

3.9.5 malt_trace_memstat

Определение:

void **malt_trace_memstat**()

Описание: Функция записывает в регистр вывода статуса контроллера памяти 0. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Нет.

3.9.6 malt_trace_rescnt

Определение:

void **malt_trace_rescnt**()

Описание: Функция записывает в регистр вывода статуса контроллера памяти 1. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Нет.

3.9.7 malt_trace_finish

Определение:

void **malt_trace_finish**()

Описание: Функция устанавливает статус завершения программы равным *0*. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Нет.

3.9.8 malt_finish

Определение:

void **malt_finish**(i32_t *c*)

Описание: Функция записывает значение **c** (код завершения программы) в регистр GPIO.

Возвращаемое значение: Нет.

3.9.9 malt_trace_rclose

Определение:

void **malt_trace_rclose**()

Описание: Функция закрывает файл, открытый для чтение. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Нет.

3.9.10 malt_trace_wclose

Определение:

malt_trace_wclose()

Описание: Функция закрывает файл, открытый для записи. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Нет.

3.9.11 malt_trace_ropen

Определение:

int **malt_trace_ropen**(char **filename*)

Описание: Открытие файла на чтение, путь в ФС хоста должен указываться относительно текущей рабочей директории. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Функция возвращает целое значение - *0* в случае успеха, в случае неудачи код ошибки.

3.9.12 malt_trace_wopen

Определение:

int **malt_trace_wopen**(char **filename*)

Описание: Открытие файла на запись, путь в ФС хоста должен указываться относительно текущей рабочей директории. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Функция возвращает целое значение - *0* в случае успеха, в случае неудачи код ошибки.

3.9.13 malt_trace_write

Определение:

int **malt_trace_write**(void **p*, int *s*)

Описание: Запись в файл **s** байт взятых по адресу **p**. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Функция возвращает целое значение - число записанных байт.

3.9.14 malt_trace_read

Определение:

int **malt_trace_read**(void **p*, int *s*)

Описание: Последовательное чтение из файла **s** байт в память по адресу **p**. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Функция возвращает целое значение - число прочитанных байт.

3.9.15 malt_trace

Определение:

void **malt_trace**(char **s*)

Описание: Функция выводит отладочное сообщение. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Нет.

3.9.16 malt_trace_func

Определение:

void **malt_trace_func**()

Описание: Печать в консоль эмулятора названия текущей функции. Доступна только при исполнении на эмуляторе.

Возвращаемое значение: Нет.

3.9.17 malt_trace_float

Определение:

void **malt_trace_float**(float *f*)

Описание: Печать в консоль эмулятора дробного числа одинарной точности.

Возвращаемое значение: Нет.

3.9.18 malt_trace_print

Определение:

void **malt_trace_print**(char **fmt*, ...)

Описание: Печать в консоль эмулятора строки форматированного вывода со стандартным printf-синтаксисом (без float). Вызывает задержки в исполнении программы.

Возвращаемое значение: Нет.

3.9.19 malt_trace_fatal

Определение:

void **malt_trace_fatal**(char **fmt*, ...)

Описание: Печать в консоль эмулятора строки форматированного вывода со стандартным printf-синтаксисом (без float). После выполнения завершает эмулятор как после аварии с установкой кода возврата в 1.

Возвращаемое значение: Нет.

3.10 Профилировочные функции

3.10.1 malt_start_pcnt

Определение:

void **malt_start_pcnt**()

Описание: Запуск счётчиков измерения эффективности интерфейса к памяти и счетчика общего числа прошедших тактов на ядре.

Возвращаемое значение: Нет.

3.10.2 malt_stop_pcnt

Определение:

void **malt_stop_pcnt**()

Описание: Остановка счётчиков измерения эффективности интерфейса к памяти и счетчика общего числа прошедших тактов на ядре.

Возвращаемое значение: Нет.

3.10.3 malt_restart_pcnt

Определение:

void **malt_restart_pcnt**()

Описание: Перезапуск счётчиков измерения эффективности интерфейса к памяти и счетчика общего числа прошедших тактов на ядре.

Возвращаемое значение: Нет.

3.10.4 malt_read_pcnt

Определение:

void malt_read_pcnt(u64_t* imem, u64_t* dmem, u64_t* total);

Описание: Чтение счётчиков измерения эффективности интерфейса к памяти и счетчика общего числа прошедших тактов на ядре. Счётчик total даёт полное число прошедших тактов, а счётчики imem и dmem число тактов простоя из-за латентности инструкций и данных соответственно. Данная функция даёт возможность измерять продолжительные интервалы при помощи 64-разрядных счётчиков, но несёт существенные накладные расходы.

Возвращаемое значение: Нет.

3.10.5 malt_total_pcnt_word

Определение:

uint32_t malt_total_pcnt_word();

Описание: Чтение 32-битного счётчика прошедших тактов.

Возвращаемое значение: Нет.

3.10.6 malt_instr_pcnt_word

Определение:

uint32_t malt_instr_pcnt_word();

Описание: Чтение 32-битного счётчика простоя по причине недоступности инструкций (латентности получения кода из общей памяти).

Возвращаемое значение: Нет.

3.10.7 malt_data_pcnt_word

Определение:

uint32_t malt_data_pcnt_word();

Описание: Чтение 32-битного счётчика простоя по причине недоступности данных (латентности чтения/записи данных).

Возвращаемое значение: Нет.

3.11 Функции для управления стеком

3.11.1 malt_local_stack_call

Определение:

int **malt_local_stack_call**(int (**fp*)(void))

Описание: Вызов функции **fp** без параметров с использованием локального стека; полезно на CM, где по умолчанию стек в глобальной памяти.

Возвращаемое значение: Функция возвращает значение соответствующее возврату функции **fp**.

3.11.2 malt_global_stack_call

Определение:

int **malt_global_stack_call**(int (**fp*)(void), int *size_kb*)

Описание: Вызов функции **fp** без параметров с выделением глобального стека размера size_kb кБайт; полезно на слейвах, где по умолчанию стек в локальной памяти.

Возвращаемое значение: Функция возвращает значение соответствующее возврату функции **fp**.

3.11.3 malt_start_thr_gl_stack

Определение:

int **malt_start_thr_gl_stack**(void* *fp*, unsigned *stack_sz*)

Описание: Функция на первом доступном slave-ядре создает поток и вызывает функцию через указатель **fp** с использованием стека в глобальной памяти . Параметр stack_sz задает размер стека в байтах. Если этот параметр равен 0, то для стека берётся стандартный размер, 16 КБ.

Возвращаемое значение: Функция возвращает целое значение – номер ядра.

3.11.4 malt_stack_get_pointer

Определение:

uint32_t **malt_stack_get_pointer**()

Описание: Функция возвращает текущий указатель стека.

Возвращаемое значение: Функция возвращает беззнаковое целое значение – указатель стека.

3.11.5 malt_stack_is_local

Определение:

int **malt_stack_is_local**()

Описание: Проверка расположения текущего стека.

Возвращаемое значение: Функция возвращает истину при использовании локального стека.

ГЛАВА 4

Библиотека newlib

Библиотека *newlib* – это реализация стандартной библиотеки языка Си, объединяющая несколько библиотек. Предназначена для использования во встраиваемых системах. В библиотеке *newlib* реализована платформонезависимая часть стандартной C библиотеки.

Системная библиотека *newlib* используется в инструментальных средствах системы MALT. Исходный код находится в каталоге MALT/tools-vx.x/build/newlib-x.x.x. Включает две основные библиотеки libc и математическую библиотеку libm.

Описание *libc* библиотеки *newlib* https://sourceware.org/newlib/libc.html

Описание математической библиотеки *libm* из *newlib* https://sourceware.org/newlib/libm.html

4.1 Библиотека libGloss

Системная библиотка *libgloss* включает реализацию необходимых для *newlib* функции, специфичных для аппаратной платформы MALT. Исходные коды библиотеки находятся в директории sw/lib/libgloss.

Библиотека предоставляет часть стандартных функций среды Unix. Предоставляются, в частности, write (только для консоли), exit, sbrk, упрощенные варианты fstat и isatty. Библиотека поддерживает работу со стандартой файловой системой ROM FS через стандартные системные функции close, read, lseek и open. Она также содержит функции-пустышки для unlink, mkdir, getcwd, stat для возможности линковки использующих их программ.

Эта библиотека является частью стандартного процесса сборки программы в системе MALT при использовании системной библиотеки *newlib*. Эта библиотека легко может быть расширена, в том числе и пользователями.

4.2 Использование библиотеки newlib

Подключение библиотеки *newlib* выполняется с помощью установки переменной системы сборки make **MALT_LIBC** в значение **newlib**. Переменную следует устанавливать глобально через export. Для этого добавить в Makefile строку:

```
export MALT_LIBC = newlib
```

4.3 Файловая система ROM FS

Любой каталог файловой системы можно, используя утилиты типа genromfs, упаковать в файл-образ этой системы, который можно компоновать, используя *newlib*, с объектными модулями для системы MALT. Это позволяет использовать в программах для MALT стандартные средства для работы с файлами стандартной библиотеки C и соответствующие системные вызовы ядра операционной системы. Поддерживаются файлы, каталоги, символические и жесткие ссылки. Символические ссылки, указывающие на объекты, используя абсолютный адрес относительно корневого каталога, в файл-образе будут указывать на адрес относительно корневого каталога этого образа. Все объекты в такой файловой системе доступны только для чтения.

Рассмотрим небольшой пример, иллюстрирующий этапы работы по подключению и использованию ROM FS. Пусть в каталоге файловой системы DATA содержатся нужные для обработки данные. Утилита genromfs доступна, в составе MALT_sw.

Командой

```
genromfs -f ИМЯ-ФАЙЛА-ОБРАЗА.romfs -d DATA
```

получаем файл-образ выбранного каталога. Расширение имени файла romfs для файл-образа фиксировано в системе сборки MALT. Далее в Makefile нужно установить переменную ROMFS, равной имени файл-образа без расширения (ROMFS = ИМЯ-ФАЙЛА-ОБРАЗА). После чего запуск в командной строке make emu должен приводить к сборке целевой программы и её запуску в эмуляторе.

Среди стандартных тестов для системы MALT в каталоге sw/app/internal_tests/romfs_test находится тест для проверки базовой функциональности работы с ROM FS.

M