ИНТЕЛЛЕКТУАЛЬНОЕ УПРАВЛЕНИЕ В СНК НА БАЗЕ DSP И ПРОЦЕССОРА ОБЩЕГО НАЗНАЧЕНИЯ

ЛОК ТРУОНГ (LOC TRUONG), технический директор, Texas Instruments

Как распределить задачи по обработке ввода/вывода между процессором общего назначения и цифровым сигнальным процессором при создании встраиваемого приложения на базе системы на кристалле (СнК)? В статье на примере СнК на базе ARM- и DSP-ядер рассмотрены способы оптимизации управления вводом/выводом, а также особенности применения ОС Linux для распределения задач обработки сигнала в режиме реального времени. Статья представляет собой перевод [1].

DSP-ОРИЕНТИРОВАННАЯ МОДЕЛЬ

Цифровые сигнальные процессоры нацелены на решение специальных задач по обработке сигнала, в основном, представляющих собой комплекс взаимосвязанных DSP-алгоритмов. Возьмем, к примеру, модем стандарта V.32. Его основной задачей является сжатие данных ниже 3-кГц полосы частот в телефонной (коммутируемой) сети общего пользования. Он обеспечивает скорость передачи 9600 бод, что выглядит весьма внушительно по сравнению со скоростью передачи в 1200 бод, которая была доступна для пользователей двухпроводной невыделенной линии на протяжении ряда лет.

Программируемый DSP может выполнять основной алгоритм, QAM-модуляцию и многие другие задачи, в том числе коррекцию с помощью выравнивающего фильтра, исправление ошибок, подавление эхо-сигналов и т.д. Большинство этих алгоритмов последовательно связаны, представляя собой фазы конвейера обработки сигнала, во время которого DSP может обрабатывать блок данных в пределах заданного интервала времени.

Задачи по прерыванию передачи или переходу на пониженное энергопотребление могут быть простыми запросами, т.к. они подразумевают завершение работы конвейера. В то же время динамическое переключение скорости передачи может не быть таким же простым, т.к. такой запрос может включать переключение алгоритмов или коэффициентов в режиме реального времени, т.е. модификацию канала обработки сигнала.

В то время как разработчики встраиваемых систем пытаются увеличить скорость, используя возможности цифровой обработки сигнала, разработчики DSP-систем делают прямо противоположное, загружая процессор общего назначения задачами управления. Это имеет смысл делать, если целью является:

1) поддержка затрат, вложенных в создание DSP-кода, а также освоение инструментов разработки и структуры системы команд (ISA);

2) последовательное расширение возможностей управления, чтобы обеспечить растущие требования рынка;

3) последнее, но наиболее важное — не нарушать канал обработки сигнала, который был настроен на определенную скорость передачи данных и опробованные сценарии использования, например, в ходе полевых испытаний или сертификации кода.

Поскольку процессоры общего назначения главным образом предназначены для управления, было бы логично передать им выполнение задач по управлению событиями, таких как управление вводом/выводом. Благодаря процессору, предназначенному для этих задач, для разработчиков систем открываются новые возможности, теперь они могут:

- добавить в систему более сложные порты ввода/вывода, содержащие стековую память высокого уровня;
- создать специальные модули управления вводом/выводом для предварительной обработки данных;
- расширить интеллектуальные возможности управления вводом/выводом с помощью таких функций как кэширование данных, адаптивное управление скоростью передачи и буферизация данных.

Возникает вопрос: какие задачи мы собираемся возложить на процессор общего назначения? Чтобы ответить на него, нужно взвесить все «за» и «против» использования операционной системы для управления вводом/выводом.

ОБРАБОТКА ЗАДАЧ УПРАВЛЕНИЯ

Рассмотрим несложную современную СнК, содержащую DSP-ядро, процессорное ядро общего назначения, системный блок прямого доступа к памяти (DMA), несколько периферийных устройств ввода/вывода, таких как последовательные порты, порты ввода/вывода общего назначения (GPIO), USB и Ethernet.

Системы предназначена для выполнения последовательных функций обработки сигнала в потоке данных, поступающих через последовательный порт, и вывода результата в другой последовательный порт. Работой системы и параметрами обработки можно управлять локально с помощью панели ввода/вывода, подключенной к портам GPIO, или удаленно через Ethernet. Блок-схема гипотетической системы приведена на рисунке 1.

Для управления несколькими схемами буферизации ввода/вывода и простыми задачами ввода/вывода, такими как командные кнопки сканирования, можно использовать простую цепь управления на процессоре общего назначения. Но когда возникает необходимость организовать более сложные процессы, например, управление сетевым протоколом TCP/IP для поддержки хотя бы простого HTTP-сервера, то появляются трудности в поддержке управляющей цепи. Потребуется компактная программа-планировщик. Работающий на базе алгоритма циклического обслуживания планировщик помогает распределить задания между системными ресурсами на основе стандартного соглашения о начале выполнения задачи и ее завершении.

Такая система называется схемой совместной многозадачности, в которой можно установить порядок выполнения нескольких заданий. Однако, когда число заданий и модулей растет, их синхронизация на основе того, кто и когда начинает исполнение, становится проблемой. Заданиям все труднее рабо-

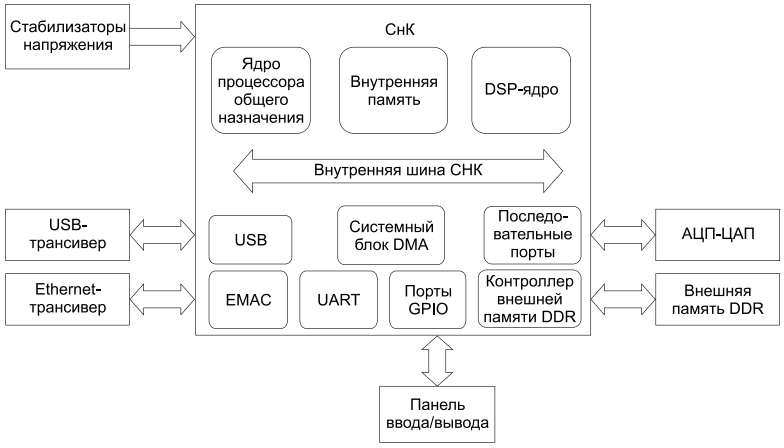


Рис. 1. **Блок-схема гипотетической системы на базе DSP и процессора общего назначения**

тать совместно, и появляется слишком много событий, которым нужно назначить приоритет. Классическим решением проблемы является прерывание обслуживания событий, когда для каждого задания задается определенный интервал времени выполнения.

На основе рабочего цикла таймера планировщик может теперь контролировать время исполнения задания, что позволяет максимально использовать ресурсы центрального процессора. В связи с этим возникает необходимость создания специальных правил доступа к ресурсам системы. Разработка и поддержка этой части приложения может стать полномасштабным процессом, в случае если увеличится объем кода и количество сценариев его использования.

К счастью, имеется большой выбор операционных систем, работающих в многозадачном режиме с приоритетами, которые подходят для встраиваемых систем. Некоторые из них даже сертифицированы для ответственных приложений, например, для систем обеспечения жизнедеятельности или бортового навигационного оборудования. Пожалуй, наиболее мощной системой, для которой доступно множество пакетов прикладных программ, стеков протоколов высокого уровня, и у которой имеется достаточно большой резерв опытных программистов, является Linux.

После своего создания в 1991 г., ОС Linux была внедрена на настольных компьютерах, а затем быстро проникла на рынок серверов и встраиваемых систем. Сегодня, когда происходит динамичное развитие компактных устройств, много внимания уделяется проектам для встраиваемых систем, что увеличивает потребность во встраиваемой версии Linux и пакетах прикладных программ, которые могут быть портированы на различные типы процессоров общего назначения, используемых в СнК.

С точки зрения управления вводом/выводом в Linux содержится широкий набор драйверов, стеков протоколов и несколько планировщиков ввода/вывода. Из почти 13,5 млн. строк исходного кода в ядре Linux версии 2.6.35 примерно половина строк связаны с драйверами.

Устройства поддерживаются с помощью нескольких категорий драйверов: символьных, блочных, сетевых и шинных. Сложные интерфейсы ввода/вывода, такие как USB, Wi-Fi и Ethernet, обеспечены полными стеками протоколов поддержки. В настоящее время в ядре 2.6 имеется возможность выбора из четырех планировщиков: с холостыми командами, с упреждением, планировщика по сроку завершения задания и CFQ-планировщика.

Планировщики выполняют интеллектуальные операции: объединение запросов, чтение с упреждением, установление срока исполнения задания и т.д. Нацеленные, главным образом, на обслуживание доступа к жесткому диску, планировщики оптимизированы для использования в серверах, но они имеют также большой потенциал использования во встраиваемых приложениях.

РЕЖИМ РЕАЛЬНОГО ВРЕМЕНИ В LINUX

Одной из наиболее широко обсуждаемых тем, связанных со встраиваемыми системами, является возможность устойчивой работы Linux в режиме реального времени. Речь идет не просто о скорости отклика системы, т.е. способности центрального процессора во время выполнения задания быстро реагировать на процессы и потоки с высоким приоритетом. (Эта возможность была значительно улучшена, начиная с версии 2.6, благодаря использованию в ядре ОС вытесняющей многозадачности (pre-emptive multitasking) и применению усовершенствованного планировщика процессов). Мы сосредоточимся на том, как приложения могут выдерживать установленный срок выполнения задания, и какой механизм для этого предусмотрен в Linux.

Сроки исполнения задания можно разделить на мягкие и жесткие или, говоря проще, — на нефатальные и фатальные. Однако определение того, какой срок считать фатальным, является субъективным. Например, в результате нарушения конечного срока доставки аудиопакета в передатчик может возникнуть слышимый щелчок или треск.

Когда это происходит в MP3-плеере, то он может быть еле заметным для пользователя, поэтому срок исполнения задания может рассматриваться как нефатальный. Однако для системы оповещения на футбольном стадионе это может привести к выходу из строя динамиков, что будет фатальным результатом.

Конечный срок исполнения во встраиваемой системе обычно устанавливают исходя из возможности системы реагировать на конкретное событие в течение определенного интервала времени. Если отклик системы за пределами этого интервала все же обеспечивает приемлемый результат, то мы имеем дело с системой мягкого реального времени. В противном случае речь идет о системе жесткого реального времени.

Ключевым параметром при определении того, отвечает ли система требованиям жесткого реального времени, является то, насколько точно можно спрогнозировать конечный срок исполнения задания, генерируемый событием. Смысл этого иллюстрирует рисунок 2. В результате генерации аппаратного прерывания возникает событие и фиксируется срок завершения определенного задания. Для обслуживания события будет выполнена программа обработки прерывания, но до этого момента работа процессора должна быть прервана.

В большинстве систем программа-диспетчер прерываний запускается после организации вложения прерываний. Общее время задержки определяется суммой времени ожидания аппаратного прерывания и времени ожидания программы-обработчика прерывания. Время задержки аппаратного прерывания может меняться в зависимости от того, было ли глобальное прерывание заблокировано процессором из-за наличия критической секции программы, например, элементарной команды, в его конвейере. Эти изменения определяют первый джиттер, показанный на рисунке 2.

Ввиду влияния работы кэш-памяти, управления доступом к внутренней шине и состязания за доступ к внешней памяти реальное число циклов

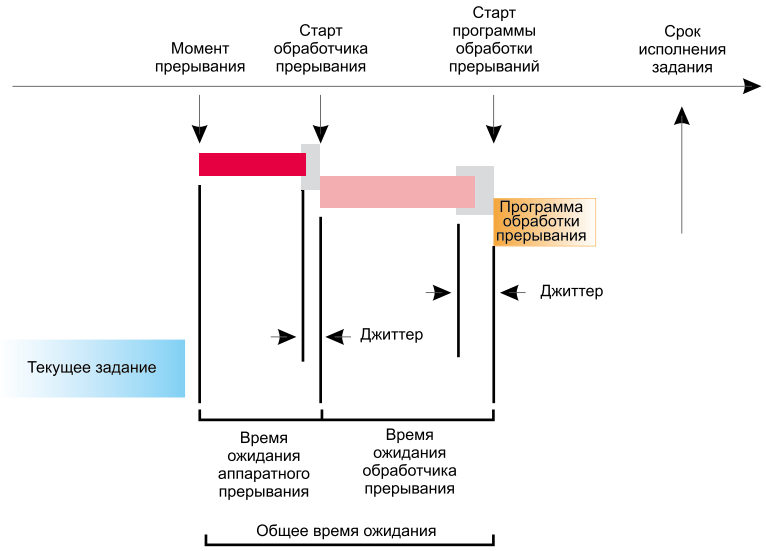


Рис. 2. **При прогнозировании времени ожидания системы необходимо учитывать максимальную величину джиттера для наихудшего случая**

программы обработчика не всегда одинаково, а меняется в определенных пределах, что вызывает второй джиттер. Нам нужно определить реальную величину этих джиттеров, чтобы узнать время задержки для наихудшего случая. Чтобы получить действительно точный результат, джиттер следует измерять на реальной аппаратуре, где запущен реальный сценарий приложения в течение некоторого интервала времени. В случае простых сценариев, где нужна только программа обслуживания прерываний (если общее время ожидания плюс время реакции программы обслуживания прерываний меньше времени, оставшегося до срока исполнения задания), систему можно определить как систему жесткого реального времени.

Иногда время реакции программы обслуживания прерываний является только частью общего времени ожидания. Ее роль может заключаться только в инициации более крупной функции, встроенной в поток, задание или процесс. Определение времени начала этого задания выполняется планировщиком.

Планировщик проверяет ждущие запросы и их приоритеты и определяет, когда задание может начать работу. Когда наступает это время, он выполняет контекстное переключение и передает ресурсы центрального процессора потоку процессов или заданию. Имеется несколько мест в системе, где время ожидания может изменяться, как и в данном случае.

Если в ОС не реализовано планирование с вытеснением задач, ожидание может занять некоторое время, т.к. текущий процесс ядра должен завершиться до того, как начнет выполняться следующее задание. Вытесняющее ядро ОС (ядро с приоритетами) может сократить время ожидания, поскольку для каждого задания имеется только конечный интервал времени исполнения. Кроме того, структуры ключевых данных могут быть защищены с помощью семафоров или взаимного исключения (объектов-мьютексов). ОС и драйверы манипулируют этими структурами, используя критические секции программы, чтобы не допустить случайного доступа.

Обычно во время работы этих секций программы для выполнения элементарных операций на несколько интервалов блокируются прерывания. Поэтому если ОС выполняет задачу, нужно измерить задержку с момента, когда программа обслуживания прерываний отправляет событие до контекстного переключения, и включить ее в суммарное время задержки.

Для оптимизации Linux и обработки процессов в режиме реального времени, необходимо понять механизм распределения времени в ядре. Чтобы снизить задержку при распределении времени, ядро Linux должно быть построено с помощью функций, реализующих вытеснение задач (CONFIG_PREEMPT). С целью дальнейшего сокращения времени задержки можно использовать патч для режима реального времени Linux Real-Time (RT). Он добавляет опцию для режима реального времени CONFIG_PREEMPT_RT при построении ядра. В коде Linux RT это реализуется с помощью снижения числа критических секций в ядре или путем замены взаимоблокировки (spinlock) на объект rtmutexe, чтобы операция стала выгружаемой.

ОБРАБОТКА ВВОДА/ВЫВОДА НА РЕАЛЬНОЙ АППАРАТУРЕ

Как разработчику встраиваемой системы общего назначения, так и разработчику DSP-системы, лучше всего использовать СнК с гибкими моделями программирования. Это может быть модель, ориентированная на процессор общего назначения, где он является управляющим устройством (master), и модель, ориентированная на DSP, где процессор общего назначения выступает в роли сопроцессора. На рисунке 3 показана блок-схема прикладного процессора OMAP-L138 C6-Integra с двумя ядрами — DSP и ARM.

Вся периферия, к которой раньше осуществлялся доступ со стороны DSP, в данном процессоре может также управляться ARM-ядром. В СнК предусмотрена работа внешней унифицированной карты памяти, которая совместно используется двумя процессорами в пределах 4-Гбайт адресного пространства. Поскольку в прикладном процессоре OMAP-L138 C6-Integra имеются два ядра, между ними должна быть обеспечена связь с помощью аппаратного механизма межпроцессорной коммуникации (IPC).

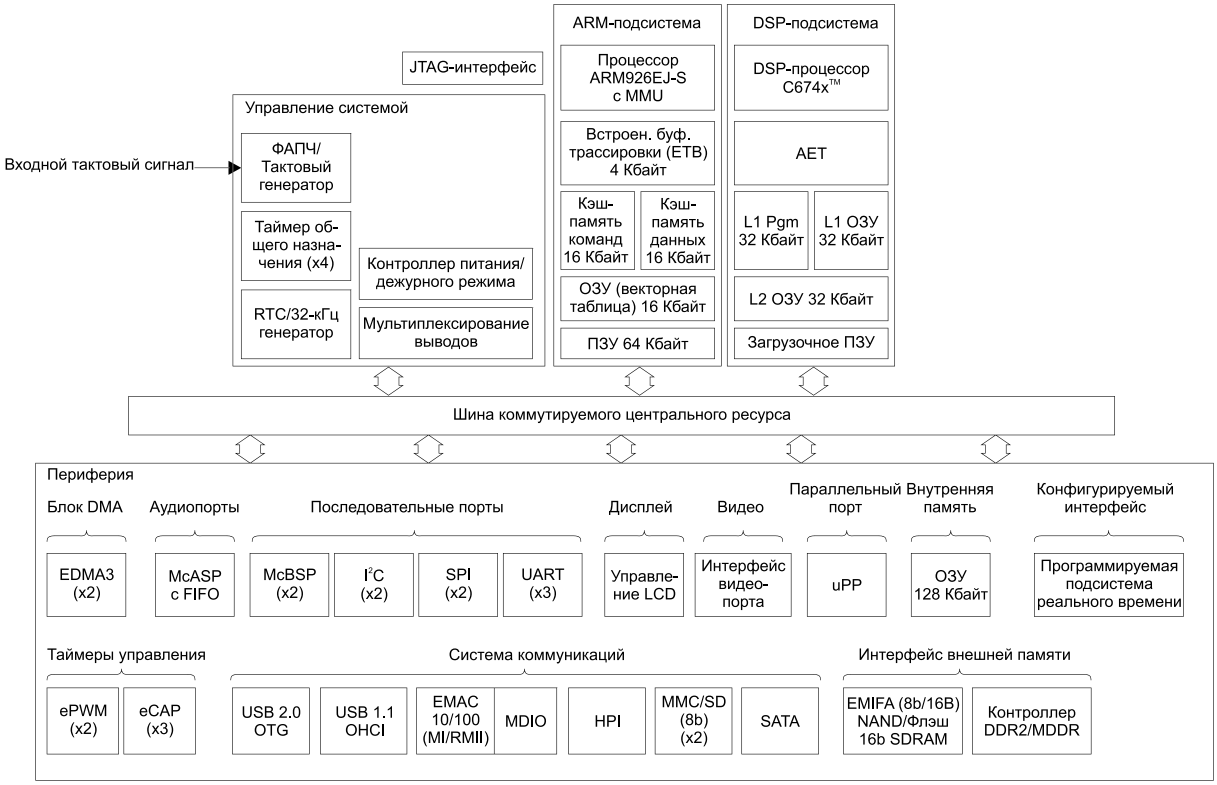


Рис. 3. **Блок-схема прикладного процессора OMAP-L138 C6-Integra с DSP+ARM-ядрами**

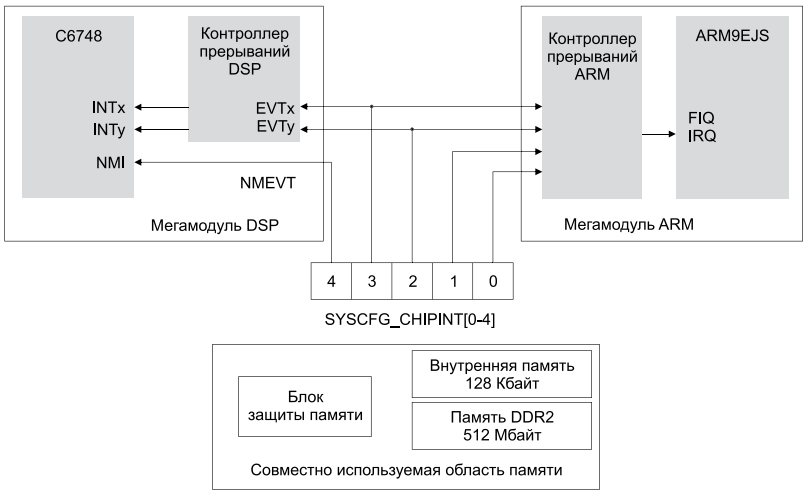


Рис. 4. **Механизм коммуникации между процессорами DSP и ARM**

Пример организации такого механизма для процессора OMAP-L138 C6-Integra показан на рисунке 4. На уровне СнК в регистре распределения памяти CHIPSIG, который расположен в модуле системной конфигурации SYSCFG, выделяется пять бит CHIPINT для передачи сигналов между DSP- и ARM-ядрами.

Для передачи сообщений в ARM-ядро можно выделить до четырех сигналов для DSP и два сигнала для ARM, чтобы передавать их в DSP, а также дополнительный сигнал для немаскируемого прерывания DSP. Заметим, что два бита являются общими как для DSP, так и для ARM с тем, чтобы можно было выполнять их одновременное прерывание. Эта полезная функция может быть использована для задач отладки. Запись единицы в этот бит генерирует сигнал или событие. Эти события попадают на соответствующий контроллер прерываний и распределяются по входам прерываний ядра. Для передачи данных между ядрами в качестве разделяемой памяти можно использовать любую память: внутреннюю (128 Кбайт) или внешнюю (512 Мбайт).

Взаимные права могут управляться с помощью механизмов взаимного исключения или семафоров, которые предусмотрены в ОС. В СнК выделен специальный блок защиты памяти системного уровня, который защищает область памяти от записи ведущими устройствами шины, такими как ARM- или DSP-ядро или блоками прямого доступа к памяти (DMA). Эта функция может быть полезной в процессе отладки программного механизма межпроцессорной коммуникации, а также детектирования некорректно функционирующих программ или утечки памяти (захвата памяти без ее последующего освобождения).

ПРИНЦИПЫ РЕАЛИЗАЦИИ

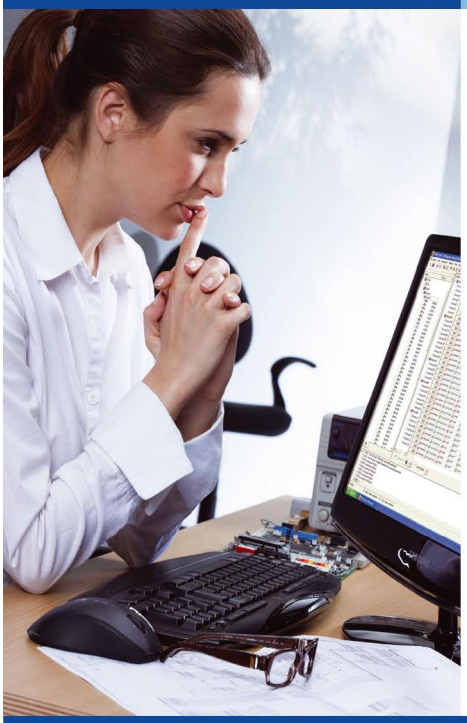
Поскольку в интегрированном DSP+ARM-процессоре вся периферия ввода/вывода на кристалле доступна со стороны ARM или DSP, то имеется возможность выбора, какая периферия каким процессором должна управляться, чтобы обеспечить режим реального времени в системе. Например, такая архитектура будет поддерживать следующие сценарии:

- обработка сигналов на DSP и вся обработка ввода/вывода на ARM;
- обработка сигналов и обработка выборочных задач ввода/вывода в режиме жесткого реального времени на DSP, а других задач ввода/вывода в режиме мягкого реального времени и фоновых задач на ARM.

Для DSP-подсистемы программное ядро DSP/BIOS можно использовать как многозадачную ОС с вытеснением или совсем не использовать ОС, чтобы максимально увеличить производительность DSP для каналов обработки сигнала.

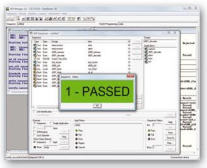
www.jtag-technologies.ru

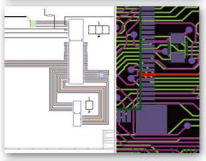
JTAG TECHNOLOGIES



Широчайший выбор средств периферийного сканирования

- Трансляция схем из любых систем проектирования
- Автоматическая генерация большинства видов тестов
- Точная диагностика дефектов и их визуализация на печатной плате
- Единые программно-аппаратные комплексы для тестирования и программирования





от мирового лидера в области тестирования и программирования собранных печатных плат

С 1993 года JTAG Technologies занимается созданием программно-аппаратных комплексов для тестирования и программирования собранных печатных плат. Сегодня инструменты JTAG Technologies позволяют генерировать большинство тестов и приложений полностью автоматически. В то же время средства ручного создания алгоритмов и тестовых воздействий дают непревзойденную гибкость и неограниченные возможности.

Представительство JTAG Technologies в России
Телефон: (812) 313-9159
Факс: (812) 313-9100
E-mail: russia@jtag.com

Эксклюзивный дистрибьютор: ЗАО «Предприятие Остек»
Телефон: (495) 788-4444
Факс: (495) 788-4442
E-mail: info@ostec-group.ru

Со стороны ARM-подсистемы также имеется возможность выбора, использовать ли ОС реального времени, например Linux, для встраиваемых систем VxWorks от Wind River, Integrity Secure Virtualization (ISV) от Green Hills, или не использовать ОС вовсе.

Возможно, главным преимуществом метода управления вводом/выводом без использования ОС является то, что разработчик в этом случае может оптимально сконфигурировать архитектуру системы, чтобы получить максимальную производительность каналов передачи данных. Это означает, что в некоторых случаях можно вплотную приблизиться к теоретическому уровню пропускной способности, которую может обеспечить устройство или система.

Многие поставщики предлагают библиотеки поддержки печатных плат (BSL), которые разработчик может с успехом использовать для оптимизации работы системы ввода/вывода. Если планируется использовать облегченную ОС, то имеется множество таких библиотек, доступных для ядра ARM9.

При реализации в приложении задач обработки ввода/вывода на ARM разработчику понадобится BSL от производителя плат. Библиотека поставляется в виде исходного кода и обеспечивает весь функционал для установки и управления различными встроенными в кристалл подсистемами ввода/вывода. Эти библиотеки поставляются на CD вместе с платами разработки и оценочными модулями, а также могут быть загружены с сайта производителя. Например, прикладной DSP+ARM-процессор OMAP-L138 C6-Integra, о котором шла речь выше, снаряжен интегрированной средой разработки Code Composer Studio компании TI и способен работать с низкоуровневым кодом при реализации приложения.

Оптимальным способом ускорения процесса разработки приложения на базе Linux для встраиваемых систем с применением DSP+ARM-процессоров является использование пакета разработки программ (SDK) для Linux, который содержит, как правило, ядро Linux, драйверы, специализированные программные компоненты от поставщика, а также дополнительные комплекты исходного кода. Поставщики полупроводников обычно предлагают такие библиотеки вместе с мультимедийным микропрограммным обеспечением и инструментарием разработки ПО с графическим интерфейсом пользователя Qt для реализации сложных приложений.

ЗАКЛЮЧЕНИЕ

Имеется несколько способов распределения заданий между ARM- и DSP-ядрами, чтобы достичь оптимальной производительности СнК. В некоторых случаях ARM демонстрирует более высокую производительность, если его использовать в качестве основного ядра, а в системах и приложениях с жесткими сроками исполнения задания, которые требуют интенсивной цифровой обработки сигнала в режиме реального времени, в качестве главного процессора лучше использовать DSP.

Интегрированные СнК обеспечивают наращиваемость и гибкость при определении задачи для каждого процессора, обеспечивая связь между двумя процессорами и позволяя каждому из них получить доступ к периферии.

Разработчики могут ускорить создание прототипа и эффективно программировать каждое ядро системы с помощью доступных инструментов разработки и программной поддержки. Имеется широкий выбор поддерживаемых решений: от реализации приложения без ОС до построения системы на базе ОС реального времени для встраиваемых систем, подобных Linux.

ЛИТЕРАТУРА

1. Loc Truong. Managing intelligent I/O processing on DSP + GPP SoCs//www.eetimes.com.