Применение и оптимизация стороннего бесплатного программного обеспечения на процессорах семейства BlackFin

Лосев Г. И.

Лосев Герман Игоревич / Losev German Igorevich – инженер, студент,
кафедра систем автоматического управления и контроля,
Национальный исследовательский университет
Московский институт электронной техники, г. Москва

Аннотация: *основная цель статьи - это показать применение стороннего бесплатного программного обеспечения вместе с процессорами фирмы Analog Device семейства Blackfin. Статья подробно описывает все особенности портирования различных приложений по обработке цифровых данных на данное семейство процессоров и различные методы оптимизации производительности. Статья рассматривает как методы, специфичные для конкретного семейства процессоров, так и методы, которые можно применить.*

Ключевые слова: *устройства цифровой обработки информации, процессоры семейства Blackfin, оптимизация, портирование.*

В современном мире для разработчиков устройств цифровой обработки данных существует большое количество готового бесплатного программного обеспечения, которое может значительно сократить время разработки устройства без увеличения его стоимости. Существует целое сообщество разработчиков, которые посвящают свое время цели создания открытых стандартов и приложений для цифровых медиа устройств. Одной из таких групп, например, является фонд Xiph.Org, некоммерческая организация, чья цель - поддержка и разработка бесплатных, доступных протоколов и программного обеспечения для нужд людей, разработчиков и коммерческих предприятий. Эта головная организация курирует управление такими технологиями, как видео (Therora), музыкальных (кодирование с потерями Vorbis и без потерь Flac) и речевых (Speex) кодеков. Но, несмотря на то, что данная фирма и подобные им предлагают доступные решения для реализации различных функций на устройствах обработки цифровых данных, эти приложения разрабатываются и тестируются на оборудовании, редко совпадающем с необходимым для конкретного разработчика. Поэтому им необходимо уметь приспосабливать эти приложения под свои нужды и правильно выбирать элементы обработки цифровых данных.

Прежде чем использовать конкретное программное обеспечение для реализации устройства цифровой обработки данных, важно проанализировать типы обработки, вовлеченные в данном типе устройств. Как и многие другие алгоритмы цифровой обработки данных, существует два основных этапа: внешний и внутренний интерфейс. В течении этапа внешнего интерфейса основные операции: распаковка пакетов и заголовков, табличный поиск, декодирование Хафмана и т. д. Этот тип операций включает большое количество условного кода и имеет тенденцию занимать относительно большой объем программного кода. Следовательно, разработчики встраиваемых систем обычно используют микроконтроллеры для этого этапа. Обработка, связанная с внутренним интерфейсом, определяется функциями фильтрования, обратного преобразования, и стандартными векторными операциями. В отличие от фазы внешнего интерфейса, этап внутреннего интерфейса включает в себя больше цикловых конструкций и операций доступа к памяти и достаточно часто меньше в масштабе кода. По этим причинам для обработки внутреннего интерфейса во встраиваемых системах используются ЦСП (цифровые сигнальные процессоры). Архитектура процессоров Blackfin объединяет функциональности микроконтроллеров и ЦСП, поэтому больше нет необходимости в двух различных устройствах.

Производительность кода имеет первостепенное значение, когда существующие приложения, такие, как различные декодеры аудио или драйверы последовательных портов, портируются на новый процессор. Тем не менее существует множество техник, доступных для оптимизации производительности в целом, некоторые требуют только минимальных дополнительных усилий. Разработчики программного обеспечения могут получить большую выгоду, ознакомившись с этими процедурами.

Первый этап портирования любой части программного обеспечения на встраиваемый процессор как Blackfin, это настройка процедур нижнего уровня ввода-вывода. Многие из кодеков для декодирования аудио файлов или другие подобные программы предполагают, что данные приходят из файла, и результат обработки сохраняется в файле. Это главным образом потому, что многие из них были изначально разработаны для работы и легкого тестирования на Unix\Linux системах, где процедуры ввода/вывода доступны в операционной системе. Во встраиваемых медийных системах, однако, входы и/или выходы часто исполнены в виде соединённых преобразователей данных, которые транслируют между цифровыми и реальными аналоговыми приемниками (доменами). Рисунок 1 показывает

концептуальный обзор возможной реализации базирующегося на процессоре Blackfin 504 устройства измерения гидроакустического давления. Входной поток битов принимается из АЦП, а выход процессора через приём-передатчик интерфейса RS-485 передаёт данные на компьютер.

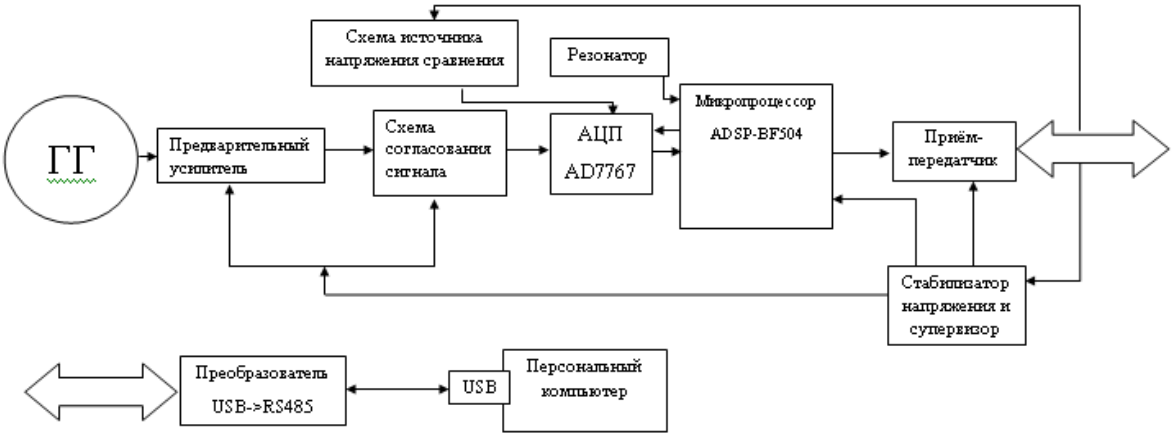

Рис. 1. Реализация измерителя акустического давления

Когда необходимо оптимизировать систему, чтобы она работала эффективно, необходимо продумать возможную последовательность действий оптимизации для достижения наибольшей эффективности. Одна из возможностей — это сфокусироваться сперва на оптимизации алгоритма изнутри программного кода на языке Си, к примеру, затем перейти к упорядочиванию потока данных системы и, наконец, настроить отдельные части кода на уровне сборки. Для того чтобы показать эффективность этого метода, рисунок 2 показывает соответствующее падение загрузки процессора при успешных оптимизационных шагах.

Скорее всего, наиболее полезным инструментом для оптимизации кода является часть программного обеспечения VisualDSP++, поставляемого вместе с отладочными платами процессоров, называемая Statistical Profiler (профайлер статистики). Использование Statistical Profiler позволяет программисту быстро сконцентрироваться на блоках кода, которые во время работы программы используют наибольшую часть производительности процессора. Фокусирование на этих критических областях дает самую высокую предельную отдачу. Выяснилось, что циклы являются главными кандидатами для оптимизации в алгоритмах устройств цифровой обработки данных. В этом есть смысл, учитывая, что DSP-ориентированная обработка чисел обычно происходит внутри циклов [2, с. 32].

Также существует глобальный подход к оптимизации кода. Во-первых, компилятор может настроить на сохранение памяти или на скорость работы программы. Кроме того, можно воспользоваться функцией, когда некоторые блоки программы заменяются автоматически на блоки, написанные на ассемблере, внутри программы на Си. Это снова создает баланс между скоростью и размером. Наконец, компиляторы, наподобие тех, что доступны для Blackfin, могут использовать двухфазную обработку, чтобы устанавливать отношения между различными исходными файлами внутри одного проекта для дальнейшего ускорения выполнения программы (межпроцедурный анализ).

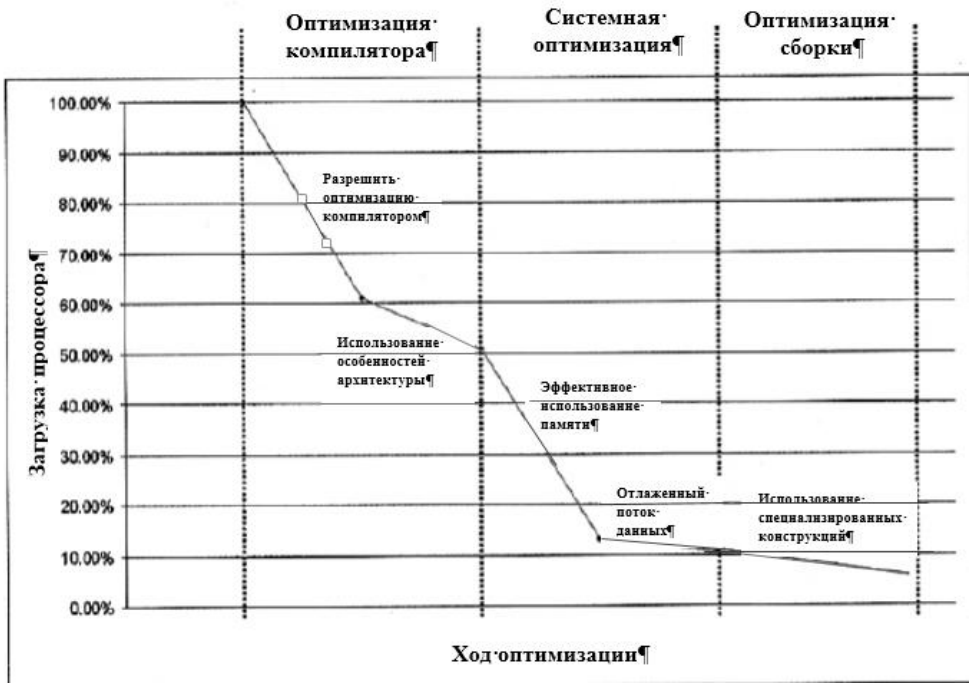

Рис. 2. График уменьшения загрузки процессора при использовании различных способов оптимизации

Как отмечено выше, большинство описанных программ для медийных алгоритмов используют арифметику с плавающей точкой. Те машины, которые написаны с дробной фиксированной точкой, на самом деле до сих пор имеют существенный недостаток. Язык, который выбирают для основной части алгоритмов кодеков, это Си, но язык Си не поддерживает «изначально» использование данных с дробной фиксированной точкой. По этой причине многие алгоритмы, использующие числа с дробной фиксированной точкой, эмулируются с помощью целочисленной математики. В то время как это делает код высокопортативным, этот подход не достигает производительности, получаемой при замене некоторых математических функций с помощью конструкций компилятора, определенных для конкретного процессора, для получения высочайшей вычислительной эффективности.

Оптимизация системы начинается с корректного распределения памяти. В лучшем случае, весь код и все данные поместятся внутрь процессорной памяти L1. К сожалению, это не всегда возможно, особенно когда применяется большой код на языке Си внутри сетевого устройства. Настоящая дилемма в том, что процессоры оптимизированы на перемещение памяти независимо от ядра посредством прямого доступа к памяти (DMA), но программисты обычно работают сиспользованием кэша вместо этого. В то время как разгон процессоров является неизбежной реальностью, использование DMA или кэша для больших передач является обязательным для сохранения производительности.

Для начала давайте рассмотрим несколько свойств, поддерживаемых архитектурой шин Blackfin, по существу. Первое, это способность обрабатывать запросы без вмешательства ядра. Поскольку внутренняя память, обычно сконструированная как вспомогательный банк данных, имеет одновременный доступ к контроллеру DMA и ядру и может оперировать с данными в одном вспомогательном банке, в то время как DMA наполняет новый буфер во втором вспомогательном банке данных. При определенных условиях одновременный доступ к тому же банку данных тоже возможен. Когда осуществляется доступ к внешней памяти, обычно доступна только одна физическая шина. Как результат, функция арбитража становится более критичной. Когда рассматривается то, что в любой возможный цикл внешняя память может давать доступ на заполнение линий кэша инструкций, в то же время она является источником и пунктом назначения для входящих и исходящих данных - задача становится явной.

Процессоры Blackfin имеют иерархическую архитектуру памяти, которая стремится к балансированию нескольких уровней памяти с разным размером и уровнем производительности. Чиповая память L1, которая ближе всех к ядру процессора, работает с полной частотой процессора. Эта память может быть настроена как SRAM и/или как кэш. Приложения, которые требуют наибольшего детерминизма, могут получить доступ к SRAM за один цикл работы процессора. Для систем, которые требуют большого объема кода, дополнительная память доступна как на чипе, так и вне его, но такая память имеет большую задержку. SDRAM медленнее, чем L1 SRAM, но она необходима для хранения больших программ и буферов данных. Тем не менее есть несколько способов для программистов

воспользоваться преимуществами быстрой памяти L1. Если целевое приложение помещается напрямую в память L1, не нужно никаких дополнительных действий, кроме как программисту поместить код приложения напрямую в это адресное пространство [3].

Когда код приложения слишком большой, как в случае, когда добавляем, скажем, сетевой компонент к аудио кодеку, можно воспользоваться механизмом кэширования, что даст доступ программисту к большой внешней памяти. Этот служит как способ автоматически вносить код в L1 память по необходимости. Как только код окажется в L1, он может быть выполнен за один цикл ядра так же, как если бы он хранился на чипе изначально. Основное достоинство этого процесса в том, что программные документы не должны регулировать движение кода из и в кэш. Использовать кэш лучше всего, когда алгоритм программы имеет линейный вид. Кэш инструкций на самом деле выполняет две роли. Во-первых, он помогает сделать предварительную выборку инструкций из внешней памяти в более эффективной манере. Также, поскольку кэш обычно работает с некоторым типом «недавно использованных» алгоритмов, инструкции, которые используются чаще, как правило, сохраняются в кэше.

Теперь, когда мы обсудили, как наилучшим образом управлять кодом для улучшения производительности приложений, давайте рассмотрим возможности по перемещению данных. Как альтернатива кэшу данные могут перемещаться в и из памяти L1, используя DMA, которая независима от ядра. В то время как ядро оперирует одной секцией памяти, DMA доставляет следующий буфер данных, который должен быть обработан.

Архитектура памяти данных Blackfin так же важна для общей производительности системы, как и скорость выполнения инструкций. Поскольку часто множество операций передачи данных происходит одновременно в мультимедийных приложениях, структура шины должна поддерживать одновременно доступ ядра и DMA ко всем областям внутренней и внешней памяти. Критически важно, чтобы арбитраж контроллера DMA и ядра выполнялся автоматически, или производительность упадет колоссально. Взаимодействие ядро-к-DMA должно быть использовано только для установки контроллера DMA и затем вновь для реагирования на прерывания, когда данные готовы к обработке. В дополнении кэш данных может тоже улучшить общую производительность системы.

В стандартном режиме, Blackfin выполняет выборку данных как часть основного функционала ядра. В то время как это наименее эффективный механизм для передачи данных, он наиболее прост с точки зрения программного модуля. Быстрая сверхоперативная память обычно доступна как часть L1 памяти, но для больших, вне-чиповых буферов время доступа пострадает, если ядро будет выбирать все. Это не только займет множество циклов на выборку данных, но также ядро процессора будет занято выборкой данных. Процессоры Blackfin имеют возможности по передаче данных между периферией и памятью, а также между различными сегментами памяти.

Для подобных приложений двойная схема буфера данных используется для согласования движком DMA. Как только одна половина циклического буфера очистится последовательным портом DMA, другая половина заполняется данными. Для контроля скорости, с которой обрабатываются сжатые данные, DMA ISR (прерывание) изменяет сигнальный флаг, который декодер может считать для того, чтобы убедиться, что запись в определенную половину двойного буфера безопасна. В системе без операционной системы подача сигналов флагом равносильна трате одного цикла ЦПУ, однако под управлением ОС планировщик может переключаться на другую цель, чтобы процессор продолжал выполнять реальную работу. Использование DMA, однако, может привести к некорректным результатам, если когерентность данных не будет учтена.

Последняя стадия оптимизации включает в себя перезапись изолированных сегментов открытого Си кода на язык ассемблер. Наилучшими кандидатами для перезаписи на ассемблер обычно являются подпрограммы обслуживания прерываний и многократно используемые модули сигнальной обработки. Мотивацией для написания обработчиков прерываний в ассемблере заключается в том, что неэффективный ISR замедлит отзыв обработчика прерываний. Как пример: некоторые аудио решения должны форматировать AC97 данные, предназначенные для аудио- цифро- аналогового преобразователя внутри аудио ISR. Поскольку это происходит на периодической основе, длинные аудио прерывания могут замедлить отзыв других событий. Перезапись этого прерывания на ассемблере лучший способ уменьшения количества циклов.

Сегодня готовы решения для встраивания различных кодеков, программ обработки, и другие инструменты по обработке цифровых данных существуют и доступны при необходимости. Также многие из этих инструментов поставляются вместе с отладочными платами процессоров серии Blackfin. Вместе они предоставляют широкий диапазон возможностей по интегрированию возможностей по обработке музыки, видео, речи и других видов цифровой информации, при этом сохраняют достаточно свободной процессорной мощи для дополнительных возможностей и функционала. Как пример, добавления встроенного Ethernet на новых 537/536 открывают дверь для новых дешёвых сетевых аудио и голосовых приложений. Очевидно, безвозмездный исходный код предвещает революцию в мире встраиваемого

оборудования обработки цифровых данных, и процессоры семейства Blackfin готовы воспользоваться всеми преимуществами этой ситуации.

Литература

1. ADSP-BF537 Blackfin Processor Hardware Reference, Revision 3.4, February 2013; http://www.analog.com/media/en/dsp-documentation/processor-manuals/ADSP-BF537_hwr_rev3.4.pdf (дата обращения: 13.12.2015).
2. Getting started with ADSP-BF537 EZ-KIT Lite, Revision 1.0, Jenuary 2005.
3. *Сотников А*. Особенности архитектуры и программирования двухъядерных процессоров семейства Blackfin ADSP-BF561// Компоненты и технологии 2007, № 6; http://www.kit-e.ru/articles/dsp/2007_6_16.php (дата обращения: 13.12.2015).