Guardian Hardware Monitor: опыт разработки

И.Е. Тарасов (ilya_e_tarasov@mail.ru), П.Н. Советов (peter@sovietov.com)
Декабрь, 2014

О продукте

Guardian Hardware Monitor[1] (GHM) это аксессуар для компьютерных корпусов. От обычного контроллера вентиляторов он отличается, в первую очередь, функцией создания достаточно сложных и программируемых световых эффектов. Такие эффекты отображаются на адресуемых лентах специального типа внутри вентиляторов, а также на светодиодных лентах других видов. GHM имеет сенсорный LCD-экран и динамик. Обмен данными с компьютером происходит с помощью USB-кабеля. При всех дополнительных возможностях планируемая цена на GHM сравнима с существующими ценами на рынке контроллеров вентиляторов. Целевой аудиторией продукта авторы видят сборщиков и покупателей так называемых бутиковых компьютерных корпусов.

Первоначальный вариант реализации

Работа над устройством началась вполне традиционным способом, с попытки создания макетной платы на базе популярного микроконтроллера STM32. Любая из подсистем GHM, взятая по отдельности, не представляет особенного труда в реализации. Дело осложняется необходимостью одновременной работы большого числа функций. Например, обновление данных для многочисленных адресуемых лент и прямое управление LCD-экраном осуществляются в рамках достаточно жестких временных ограничений. Использование отдельных чипов для различной периферии привело к чрезмерному усложнению и удорожанию макетной платы. Особенно ярко недостатки проявились при реализации эффекта стробоскопа. В реализации этого эффекта участвовали: контроллер вентиляторов, МК STM32, светодиодный контроллер и ненадежная аналоговая схема на MOSFET-транзисторах. Необходимость быстрого прототипирования в условиях постоянно изменяющихся требований к продукту окончательно поставила крест на этом варианте реализации.

[1] http://www.computerpoweruser.com/article/16281/giving-new-life-to-lighting

Реализация на ПЛИС

Удачным решением оказался переход на ПЛИС. Использование этой технологии позволило избавиться от большинства микросхем на макетной плате. Существенно упростила работу библиотека И.Е. Тарасова для создания гетерогенных вычислительных систем. Структурная схема проекта естественным образом отобразилась на ПЛИС. Элементам схемы были сопоставлены конфигурируемые софт-процессоры с настраиваемой разрядностью, а также блоки периферии. Типичные нюансы программирования микроконтроллера, такие, как: учет таймеров и аппаратных интерфейсов, расположение пинов, работа с прерываниями и DMA, использование ОС реального времени — перестали иметь значение. Возможность горячей замены кода на отдельных ядрах системы оказалась полезной при отладке и в работе программируемых пользователем эффектов.

Адаптация к архитектуре Xilinx

Софт-процессор для ПЛИС во многих случаях является естественным решением, облегчающим и разработку, и использование системы. В части разработки важным является то, что большинство изменений при отладке вносятся не в аппаратную часть системы (что требует постоянного перезапуска САПР ПЛИС), а в программу софт-процессора. Можно заметить, что ПЛИС большого объема способны загрузить мощную рабочую станцию на 10-20 часов для каждой итерации проектирования (т.е. даже при минимальных изменениях проекта). Архитектура софт-процессора, с одной стороны, может быть и произвольной, поскольку разработчик имеет дело с программируемой микросхемой, но с другой, желательно реализовать процессор, адаптированный и к задаче, и к архитектуре самой ПЛИС, и к имеющимся программным инструментам.

ПЛИС семейства Spartan-6 представляет собой недорогую аппаратную платформу, которая может конкурировать по цене с микроконтроллерами среднего уровня. Использованная в проекте микросхема XC6LX9 имеет 9 тыс. эквивалентных логических ячеек (5700 LUT и 11400 триггеров), 32 блока синхронной двупортовой памяти с организацией 1024 x18 и 16 аппаратных умножителей (менее актуальных для описываемого проекта). Логические генераторы (LUT) имеют 6 входов.

В ходе разработки форт-процессоров (с 2000 года) было выявлено, что стековая архитектура достаточно удобна для реализации на базе ПЛИС Xilinx. С точки зрения общей схемотехники, использование вершины стека в качестве обязательных операндов позволяет отказаться от отдельного такта декодирования команды. Это связано с тем, что регистровая архитектура предполагает, что информация об операндах содержится в команде, а следовательно, индексы операндов станут известны только после чтения команды из памяти. Для стековой архитектуры чтение команды и чтение вершины стека может выполняться параллельно. Таким образом, становится доступной двухтактная модель конвейера “выборка-исполнение”. Для регистрового процессора базовой является трехтактная модель - “выборка-декодирование-исполнение”. Сокращение конвейера приводит не только к упрощению управляющих схем процессора, но и к уменьшению штрафов на перезаполнение конвейера. В форт-процессоре переход к новому адресу вызывает простой конвейера на 1 такт.

Дополнительно к этому, ПЛИС Xilinx имеют удобную для реализации стека особенность. Поскольку LUT представляют собой фрагменты памяти, хранящие полную таблицу истинности для 64 комбинаций входных значений, их альтернативным режимом является режим двупортовой распределенной памяти. Таким образом, для ПЛИС Xilinx достаточно привлекательна реализация небольших структур памяти, распределенных по кристаллу. С каждым триггером связана LUT, обеспечивающая дополнительные 32 бита в глубину на каждый бит в триггере. Поэтому оказывается возможной реализация нескольких стековых структур с приемлемыми затратами ресурсов ПЛИС.

Софт-процессор kf532 представляет собой 5-е поколение софт-процессоров с аппаратной поддержкой Форта. Это процессор с двухтактным конвейером, раздельными стеками данных и возвратов и аппаратной поддержкой базовых команд Форта. Ввиду того, что команды Форта не требуют указания операндов, разрядность команд процессора составляет 6 бит, что существенно упрощает их декодирование, поскольку сравнение команды с константой может быть выполнено всего в одной LUT современных ПЛИС.

В процессоре есть возможность добавления встроенных модулей: аппаратного стека циклов, модуля деления независимых операндов, системного таймера и набора системных регистров. Разрядность данных параметризуется, т.е. может быть установлена на верхнем уровне проекта без необходимости коррекции исходных текстов процессора. При изменении разрядности сохраняется совместимость кода как на уровне исходных текстов, так и на двоичном уровне. Это объясняется тем, что единственная команда загрузки литерала работает по принципу сдвигового регистра. Если

процессор имеет меньшую разрядность, лишние биты будут потеряны, однако это не вызывает аварийной ситуации.

По сравнению с софт-процессором Microblaze, официально предлагаемым Xilinx, характеристики kf532 выглядят достаточно привлекательными для проектов начального уровня. Если Microblaze занимает от 600 до 3500 LUT (типично 2000) в зависимости от конфигурации, то kf532 в 32-разрядной версии, со стеком циклов и аппаратной поддержкой деления занимает 1600 LUT. Кроме того, в kf уже встроен загрузчик, работающий по UART и шинный контроллер. Процессор может работать на частоте до 100 МГц в ПЛИС Spartan-6 минимальной градации скорости. Это несколько меньше, чем для Microblaze в типичной конфигурации, однако для kf532 не используются топологические проектные ограничения, оптимизирующие взаимное расположение компонентов процессора на кристалле ПЛИС. Эта работа достаточно трудоемка, и аналогичного прироста производительности (порядка 10-30%) можно добиться и другими способами, прежде всего - добавлением в систему аппаратных ускорителей для наиболее важных операций.

Эксплуатация процессора подтвердила предположения о высокой плотности форт-кода. Типичные задачи, программируемые на Форте, занимают существенно меньший объем памяти по сравнению с аналогичными программами на Си для Microblaze. Это особенно важно для ПЛИС начального уровня, где эффективная с точки зрения электроники блочная память представляет собой достаточно дефицитный ресурс.

В текущей реализации проекта софт-процессоры обмениваются между собой информацией с помощью механизма разделяемой памяти. Ко всем ядрам имеется прямой доступ через виртуальный COM-порт, а встроенный загрузчик позволяет обновлять программный код произвольного ядра в любой момент без остановки всей системы. Видеоконтроллер для LCD с разрешением 480x272x16bpp реализован в виде периферийного модуля. Он использует внешнюю статическую память объемом 4 Мбит, которая может хранить две видеостраницы. Видеоконтроллер поддерживает быстрое переключение страниц, а также ускоряет операции по выводу отдельных пикселов и горизонтальных линий.

Создание инструментального ПО

При использовании форт-процессора в составе многоядерной системы возникает вопрос по поводу его программирования. Имеющийся компилятор Форта не слишком подходил для проекта. Одной из особенностей GHM

является возможность создания пользовательских эффектов на простом языке GuardianScript. Графический интерфейс и другие подсистемы также предпочтительнее было совместно разрабатывать на понятном всей команде языке с Си-подобным синтаксисом, не отвлекаясь на особенности Форта. Забегая вперед, можно отметить, что компилятор GuardianScript был создан П.Н. Советовым за неделю и использовался как для создания световых эффектов, так и для прочих, уже системных задач. Этот результат имел важное значение в условиях сжатых сроков разработки и постоянно меняющихся требований.

Закономерно возникла необходимость быстрого построения компилятора GuardianScript для форт-процессора. При этом, системный диалект этого языка, в отличие от варианта для создания эффектов, должен был обладать дополнительными возможностями, включая синтаксис для межпроцессного взаимодействия и средства встраивания кода на Форте. Кроме того, компилятор нужен был и для ПК, для отладки и моделирования (на деле он практически не использовался), а также в составе клиентской части веб-приложения, для пользователей. Для решения этих задач был выбран подход на основе языка META II и его более поздних вариантов.

META II[2] (Val Schorre, 1963) это язык для написания компиляторов. В нем используется набор правил, синтаксис которых напоминает BNF. В правой части правил применяются вставки кода на целевом языке. Синтаксический анализ основан на методе рекурсивного спуска. Более интересно, что META II может произвести новую версию генератора компиляторов по собственному описанию. Это свойство называется метакомпиляцией. С ним легко создавать варианты META II для разнообразных платформ. Кроме того, несложно специализировать и сам генератор компиляторов. Это может иметь смысл, например, для отказа от метода отката при синтаксическом разборе, что сделает компиляцию более эффективной и предсказуемой для узкого класса разбираемых языков. META II — необычайно компактная система, поэтому ее удобно использовать для создания компиляторов и генераторов компиляторов, встраиваемых в приложение. Программа метакомпилятора занимает всего 22 строки, а в качестве результата генерируется код для виртуальной стековой машины, поддерживающей 19 простых инструкций. META-подход широко использует сопоставление с образцом. Все стадии построения компилятора или интерпретатора несложно реализовать в рамках единой META-программы. Нет концептуального деления на лексический и синтаксический разбор. Генерирование и обработка AST (абстрактного

[2] http://www.hcs64.com/files/pd1-3-schorre.pdf

синтаксического дерева) тоже не требует специальных средств вне META-языка (TREE-META[3], 1968).

Реализация знаменитой системы NLS (графический интерфейс, гипертекст, средства коллективной работы, 1968) Дугласа Энгельбарта стала возможной благодаря использованию множества специализированных языков, созданных с помощью системы TREE-META. Более современным примером использования META-подхода является система OMeta[4] (Alessandro Warth, 2007) — ключевой элемент проекта STEPS Алана Кэя по сокращению сложности ПО для ПК, в рамках которого создано множество специализированных языков. С помощью OMeta реализован стек TCP/IP в 200 строк с автоматическим разбором ASCII-диаграмм в RFC-документах. Еще один пример из проекта STEPS: векторная 2d библиотека на декларативном, потоковом языке, объемом в 450 строк. OMeta обладает мощным арсеналом средств для генерирования компиляторов: запоминание промежуточных результатов разбора (Packrat), поддержка левой рекурсии, использование правил высшего порядка. Возможности META-подхода не ограничиваются только построением компиляторов. В том же ключе легко реализуется разбор разнообразных форматов, от текстовых JSON-данных, до графического PNG.

Для проекта GHM на OMeta/JS реализован встроенный в веб-приложение вариант компилятора языка GuardianScript. Язык этот является, во многом, подмножеством Си и JavaScript. Из синтаксиса последних, в частности, были убраны операторы ++ и --, а использование фигурных скобок в циклах и ветвлениях возведено в ранг требования. Реализация компилятора на META-языке заняла менее 200 строк. Трудно переоценить возможность в любой момент внести изменения в синтаксис GuardianScript. Во время разработки каждый запуск программы начинался с перестроения компилятора, а заканчивался передачей целевого кода в устройство, однако весь процесс занимал менее секунды. В версии компилятора для форт-процессора к аппаратному стеку параметров, куда попадают данные перед вызовом функции, был добавлен стек[5] в области памяти данных, где размещаются аргументы, локальные переменные и массивы на время вызова функции. Таким образом появилась возможность программировать на двух уровнях: уровень Форта для считанного числа участков кода, требующих особенного быстродействия и уровень GuardianScript, на котором написана большая часть программы. С помощью META-подхода были также созданы

---

3 http://bitsavers.informatik.uni-stuttgart.de/pdf/sri/arc/rulifson/A_Tree_Meta_For_The_XDS_940_Appendix_D_Apr68.pdf

4 http://tinlizzie.org/ometa/

5 http://users.ece.cmu.edu/~koopman/forth/rochester_90b.pdf

дизассемблер кода форт-процессора и преобразователь SVF-файлов для интерфейса JTAG.

Выводы

Основная разработка GHM заняла менее полугода. Были созданы универсальная плата на базе ПЛИС Xilinix и набор инструментального ПО, которые можно рассматривать как возможный фундамент для целого ряда проектов, в том числе и учебных Arduino-подобных. Успешно опробован подход одновременного аппаратного и программного проектирования и разработки. Это подход с использованием инструментария для создания гетерогенных вычислительных систем на ПЛИС и быстрой разработки компиляторов специализированных языков по META-технологии.