СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сегодняшние достижения в области электроники и компьютерных технологий открывают все новые и новые перспективы для создания и применения устройств, спроектированных на микроконтроллерах и микропроцессорах, в самых различных сферах человеческой деятельности. Одним из наиболее популярных микропроцессорных платформ в настоящее время является ARM. За последние 10 лет архитектура ARM развивалась особенно интенсивно, опережая все остальные процессорные технологии. В настоящее время 32-разрядные микроконтроллеры ARM используются в наиболее быстро развивающихся сегментах рынка, ориентированных на мобильные устройства и устройства коммуникации. Ежегодно рынок ARM микроконтроллеров захватывает все новые и новые типы устройств и технологических решений, а количество выпускаемых кристаллов ARM исчисляется десятками миллионов.

На сегодняшний день ARM микроконтроллеры применяются практически везде, начиная с мобильных телефонов и заканчивая устройствами электронного управления автомобилями и GPS навигаторами. Более того, ARM микропроцессоры вторгаются в отрасли, ранее считавшиеся недоступными для данной технологии – это настольные и переносные компьютеры и серверные решения. Так, например, уже следующее поколение операционных систем Windows, анонсированное как Windows 8, наряду с процессорами Intel и AMD будет работать и на платформе ARM.

Эта книга посвящена анализу практических методов программирования систем с микроконтроллерами ARM на языке C/C++ . Разработка программной части таких систем занимает львиную долю времени в процессе проектирования, причем в этот процесс вовлечена масса программистов, для которых эта книга и предназначена в первую очередь. Помимо анализа элементов "чистого" программирования на C/C++ в данной книге рассматривается и другой, не менее важный аспект разработки программного обеспечения – отладка и оптимизация программного обеспечения для микропроцессоров ARM.

Производительность приложений, написанных на языке высокого уровня, даже на таком мощном, как C++, будет существенно зависеть от того, насколько хорошо разработчик понимает программную архитектуру ARM и владеет основами программирования на нижнем уровне. По этой причине значительная

часть материала книги посвящена анализу программных интерфейсов языков C/C++ и ассемблера, что позволяет достичь высокой производительности C/C++ приложений.

Важнейшим этапом разработки программного обеспечения является отладка программного кода приложения. Если для самых простых приложений процесс отладки может занять несколько минут, то комплексные приложения могут потребовать от инженера-разработчика значительных усилий и времени, сопоставимого с временем разработки исходных текстов. Для успешной отладки приложения, написанного на языке высокого уровня, программист должен знать основы язык ассемблера микроконтроллеров ARM – в этой книге данная тема рассматривается достаточно широко, так же как и основные методы анализа ошибок и оптимизации программного кода.

Материал книги имеет практическую направленность, поэтому читатели найдут здесь много примеров исходных текстов программ, иллюстрирующих теоретические аспекты тех или иных проблем. Книга будет полезна разработчикам программного обеспечения для микропроцессоров ARM, инженерам и всем желающим ознакомиться с принципами программирования и отладки программного обеспечения для данной архитектуры. Книга содержит несколько глав, краткое описание каждой из которых приводится далее.

- **Глава 1**. ***Программная архитектура микроконтроллеров с ядром ARM7***. В данной главе рассматривается программная архитектура микроконтроллеров ARM и специфика разработки приложений для RISC-процессоров. Часть материала главы посвящена особенностям аппаратной реализации микроконтроллеров ARM и взаимодействия различных функциональных узлов на кристалле микропроцессора. Рассматривается структура программного обеспечения для систем с микроконтроллерами ARM, а также варианты выбора той или иной модели проектирования для реализации программной части.
- **Глава 2**. ***Инструменты программирования микроконтроллеров ARM***. В этой главе дается общая оценка инструментальным средствам программирования систем на базе микроконтроллеров ARM, а также рассматривается один из наиболее популярных пакетов разработки приложений в среде C/C++ для ARM – компилятор Keil C. Значительная часть материала главы посвящена методам создания приложений, положенных в основу инструментального пакета Keil C для ARM. На примере простейшего приложения рассматриваются все этапы разработки пользовательского программного обеспечения, включая компиляцию исходных текстов, генерацию объектных модулей и сборку приложения.
- **Глава 3**. ***Программирование периферийных устройств микроконтроллеров ARM на Keil C***. В этой главе рассматриваются принципы функционирования периферийных устройств (таймеров, портов ввода-вывода, аналого-цифровых и цифро-аналоговых преобразователей), входящих в состав кристаллов современных микроконтроллеров ARM и практические методы программирования таких устройств. На практических примерах рассмот-

рено функционирование системы прерываний микроконтроллеров ARM. Все исходные тексты программ сопровождаются детальным описанием.

- **Глава 4**. ***Программный интерфейс C/C++ и ассемблера для микроконтроллеров ARM***. Материал главы посвящен реализации программного интерфейса между программным кодом, написанным на языке C/C++ и процедурами, разработанными на языке макроассемблера среды Keil. Рассматриваются соглашения о вызовах процедур и передаче параметров процедурам, доступ к общим данным и функциям, использование стека и т.д. Материал главы сопровождается многочисленными примерами программ, иллюстрирующими различные подходы при реализации интерфейса между фрагментами кода на C++ и ассемблере.
- **Глава 5**. ***Отладка программного кода микроконтроллеров ARM***. Данная глава посвящена анализу основных методов отладки программного кода и методике обнаружения часто встречающихся ошибок при программировании микроконтроллеров ARM. Часть материала главы посвящена основам дизассемблирования и анализа программного кода.
- **Глава 6**. ***Анализ и оптимизация программного кода микроконтроллеров ARM***. В этой главе дается подробный анализ методов оптимизации приложений на уровне исходных текстов программ. Рассматриваются такие методы оптимизации кода, как разворачивание циклов, применение инструкций условного выполнения, выбор типов переменных и т.д. Теоретический материал сопровождается многочисленными примерами исходных текстов программ.

ГЛАВА 1

ПРОГРАММНАЯ АРХИТЕКТУРА МИКРОКОНТРОЛЛЕРОВ С ЯДРОМ ARM7

В этой главе будут рассмотрены основы архитектуры ядра ARM7TDMI, которое является базовым модулем для многочисленной линейки процессоров ARM.

Ядро ARM базируется на архитектуре RISC (**R**educed **I**nstruction **S**et **C**omputing, архитектура процессоров, построенная на основе сокращенного набора команд), особенностью которой является использование простых и эффективных инструкций процессора, которые могут выполняться за один цикл. Базовые концепции RISC предполагают перенос основного акцента в разработке приложений с аппаратной части на программную, поскольку производительность приложений поднять значительно проще программными методами, чем используя сложные аппаратные решения.

Вследствие этого программирование RISC-процессоров выдвигает более существенные требования к эффективности компиляторов, по сравнению с архитектурой CISC (**C**omplete **I**nstruction **S**et **C**omputing, архитектура процессоров с широким набором различных машинных команд переменной длины и разным временем их исполнения). Микропроцессоры с CISC архитектурой (например, Intel x86) не столь требовательны к программным средствам разработки – здесь основной упор делается на производительность аппаратной части. На Рис. 1.1 показаны эти отличия.

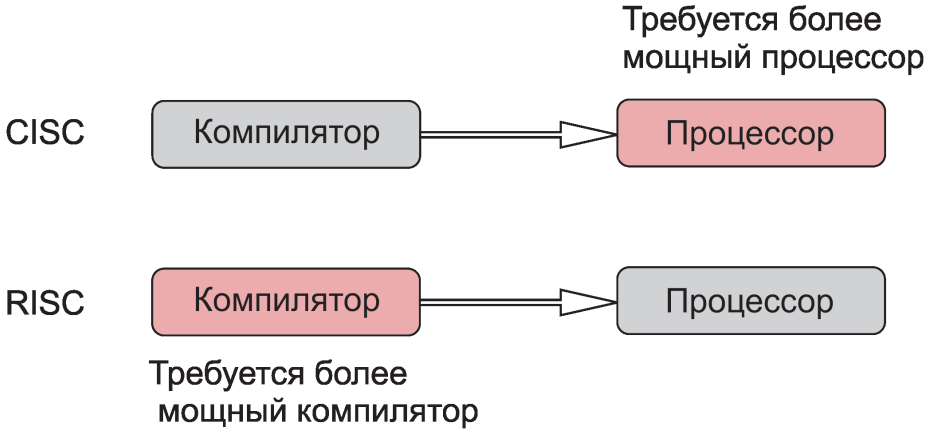

Рис. 1.1.

Если рассматривать архитектуру RISC более детально, то в ее основе лежат следующие базовые принципы, перечисленные далее.

1. **Система инструкций (команд) процессора.** Стандартный RISC-процессор имеет ограниченный набор типов инструкций, причем каждая из этих инструкций выполняется за один машинный цикл процессора. Разработка отдельных программных алгоритмов, например, деления, целиком возлагается на инструментальное средство разработки программ (компилятор) или самого разработчика. Каждая инструкция процессора имеет фиксированную длину, что позволяет успешно использовать принцип конвейера для выборки следующей инструкции в то время, пока предыдущая находится на стадии декодирования. В противоположность этому, в CISC-процессорах инструкции имеют различную длину и могут выполняться за несколько машинных циклов.

   Для иллюстрации этой концепции посмотрим, например, как выполняются операции сложения двух 32-разрядных целых чисел в процессорах CISC и RISC. Если, например, требуется сложить две переменные **i1** и **i2**, находящиеся в оперативной памяти, и сохранить результат в переменной **res**, также находящейся в памяти, то для Intel x86 процессора потребуются следующие команды:

```
MOV  EAX,  dword  ptr  a1
ADD  EAX,  dword  ptr  a2
MOV  dword  ptr  res,  EAX
```

   В эти вычисления вовлечен один регистр общего назначения **EAX**, а вычисление суммы (инструкция **ADD**) выполняется над операндами, расположенными в регистре и в памяти.

   Для RISC-процессора, например, ARM7 LPC2148, та же операция потребует следующей последовательности инструкций:

```
LDR    r0,  =i1
LDR    r1,  =i2
LDR    r3,  =res
LDR    r0,  [r0]
LDR    r1,  [r1]
ADD    r0,  r0,  r1
STR    r0,  [r3]
```

   Для выполнения любой операции в RISC-процессоре (кроме загрузки и сохранения данных, **LDR** и **STR** соответственно) оба операнда должны находиться в регистрах – это видно из мнемоники инструкции **ADD**, где переменные **i1** и **i2** находятся в регистрах **r0 r1** соответственно. Сравнивая два фрагмента кода для CISC и RISC процессоров, мы видим, что для RISC-процессора требуется больше инструкций. С другой стороны, большое количество регистров в ARM процессорах позволяет очень эффективно выполнять вычислительные операции с несколькими переменными, поскольку промежуточные результаты вычислений можно разместить в регистрах. Кроме того, в ARM процессорах можно использовать инструкции множественного доступа к нескольким ячейкам памяти, что повышает производительность операций чтения/записи данных.

Дополнительные возможности повышения производительности ARM микропроцессоров достигаются и при использовании инструкций условного выполнения, когда следующая инструкция выполняется только в том случае, если предыдущая инструкция установила определенные флаги в регистре состояния программы. Эти и другие возможности оптимизации быстродействия программного кода ARM мы рассмотрим далее в этой книге.

2. **Конвейер инструкций.** Процесс обработки каждой инструкции процессора разбивается на несколько этапов, которые выполняются одновременно. В идеальном варианте, для достижения максимальной производительности конвейер инструкций должен продвигаться на один шаг в каждом машинном цикле, при этом декодирование команды может осуществляться на одном шаге конвейера. Этот подход отличается от принятого для CISC-архитектур, в которых декодирование инструкций требует выполнения специальных микропрограмм.
3. **Использование регистров процессора.** В RISC-процессорах имеется набор многочисленных регистров общего применения. Каждый из регистров может содержать данные или адрес данных в памяти, поэтому регистры являются локальными хранилищами данных при выполнении всех операций в процессоре. Для сравнения: в CISC-процессорах имеется ограниченный набор регистров, каждый из которых имеет отдельное функциональное назначение, поэтому многие инструкции CISC в качестве одного из операндов используют ячейку памяти. В примере сложения двух чисел для CISC процессоров Intel x86 использовалась, например, инструкция **`ADD`**, одним из операндов которой являлась ячейка памяти. Такая инструкция требует достаточно много машинных циклов, что снижает производительность приложения, особенно если подобные инструкции используются при циклических вычислениях. Несмотря на существенные различия в архитектурах, в последнее время осуществляется постепенное сближение архитектур RISC и CISC. Так, например, CISC-микропроцессоры на уровне микропрограмм реализованы по принципам RISC, что увеличивает скорость выполнения микроинструкций.

Здесь нужно отметить еще один важный момент: базовое ядро ARM микроконтроллеров не вполне наследует концепции RISC по той причине, что специфика приложений для встроенных и мобильных систем требует большей гибкости и производительности, чем может обеспечить "чистая" RISC-архитектура. Кроме того, одним из основных требований к ARM-процессорам является низкое энергопотребление, поскольку мобильные и переносные устройства работают, как правило, с батарейным питанием.

Программная архитектура современных ARM-микроконтроллеров отличается от той, что реализована в традиционных RISC-устройствах, прежде всего в плане более расширенных возможностей инструкций процессора. Эти отличия обеспечивают более высокую производительность встроенных приложений – они описаны в следующем разделе.

1.1. Особенности выполнения инструкций микроконтроллеров ARM

Некоторые инструкции ARM требуют для выполнения не одного, как классические RISC команды, а нескольких машинных циклов. Так, например, инструкции множественной загрузки/сохранения (load–store–multiple instructions), в зависимости от количества задействованых регистров, используют разное количество машинных циклов. Обмен данными с памятью для таких инструкций осуществляется по последовательным адресам, что повышает производительность системы по сравнению с операциями с произвольным доступом.

Характерной особенностью ARM-процессоров является наличие встроенной многорегистровой схемы циклического сдвига (barrel shifter), которая позволяет выполнять сложные арифметические и логические операции в одной инструкции. Циклический сдвигатель реализован как аппаратное устройство на кристалле процессора и позволяет выполнять арифметические и логические сдвиги содержимого операнда-источника перед выполнением инструкции. С помощью схемы сдвига можно реализовать операции быстрого умножения на степень двух, а также комбинированные операции сложения/вычитания/умножение целых чисел.

Все инструкции процессора ARM могут выполняться в так называемом "условном" режиме (conditional execution). Это означает, что инструкция будет выполнена только в том случае, если условие, указанное в мнемонике инструкции, истинно. Например, команда сложения содержимого двух регистров **r0** и **r1** и результатом в регистре **r0** с безусловным выполнением имеет мнемонику

```
ADD  r0,  r0,  r1
```

Если эта же команда должна выполняться, только если предыдущие инструкции установили флаг **Z** регистре состояния, то мнемоника инструкции сложения будет следующей:

```
ADDEQ  r0,  r0,  r1
```

Инструкция будет пропущена, если требование Z=1 выполняться не будет. Условно выполняемые инструкции позволяют оптимизировать программный код за счет минимизации количества ветвлений, что очень важно при разработке приложений реального времени с ARM-процессорами.

В набор инструкций современных микропроцессоров ARM включена также группа инструкций, позволяющая программировать алгоритмы цифровой обработки сигналов. К этой группе относятся инструкции быстрого умножения 16 г 16 бит, а также инструкции умножения с насыщением. С помощью этой группы инструкций можно создавать эффективные программные алгоритмы без использования специального процессора цифровой обработки сигналов.

1.2. Основы аппаратной архитектуры микроконтроллеров ARM

Встроенные и мобильные устройства и системы широко используются для управления и измерения в самых различных отраслях науки, техники и производства. ARM микропроцессоры как нельзя лучше подходят для реализации таких систем. Кроме ядра, выполняющего функции вычислительного интеллектуального модуля, на кристалле ARM микроконтроллера имеется ряд устройств, которые позволяют строить интерфейс с внешним миром (датчиками, реле и двигателями, компьютерами и т. д.).

Ядро ARM, помимо реализации вычислительных функций, выполняет функции управления периферийными устройствами кристалла, такими, как порты ввода-вывода, таймеры, устройства оцифровки сигналов (аналого-цифровые и цифро-аналоговые преобразователи). Аппаратную часть современного ARM микропроцессора можно представить так, как показано на рис. 1.2.

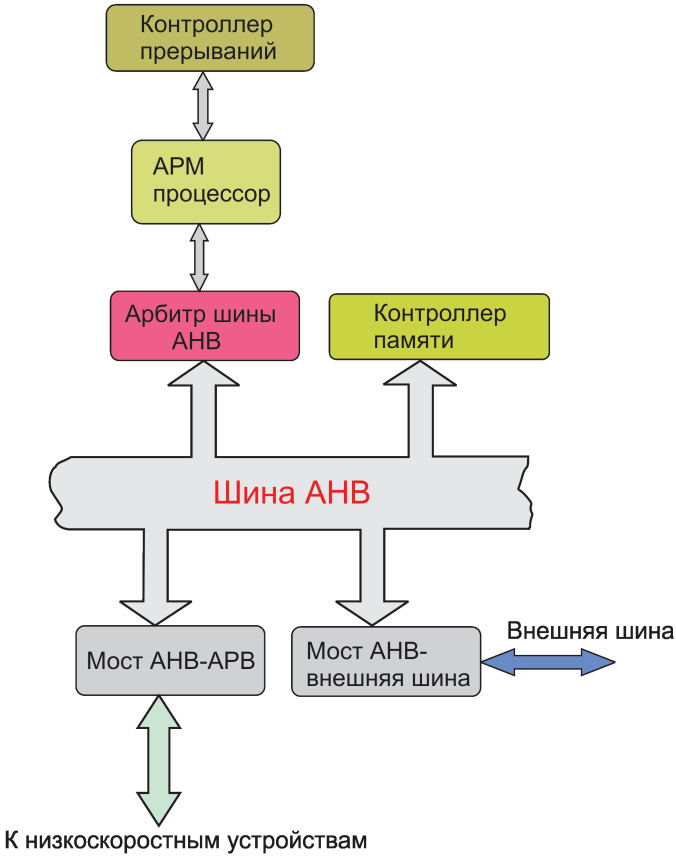

Рис. 1.2.

Рассмотрим эту схему детально, поскольку она отображает основные функциональные узлы и связи между ними, характерные для всех микроконтроллеров ARM. Важнейшим узлом в архитектуре микроконтроллера является контроллер прерываний, который осуществляет захват и приоритезацию внешних сигналов (событий), поступающих от периферийных устройств. Именно наличие этого функционального блока позволяет создавать системы реального времени, хотя существуют и другие подходы к построению таких систем.

Когда какое-либо устройство требует вмешательства со стороны процессора, оно выставляет сигнал прерывания. Контроллер прерывания, в свою очередь, управляет доступом к источнику прерывания со стороны программного обеспечения посредством установки соответствующих бит в регистрах прерываний. В микроконтроллерах ARM применяются два типа контроллеров прерываний – стандартный и векторизованный (Vector Interrupt Controller, VIC).

Стандартный контроллер прерывания просто посылает сигнал прерывания микроконтроллеру, когда какое-либо устройство требует немедленного обслуживания, и устанавливает сигнал прерывания на соответствующей линии. Такой контроллер можно запрограммировать так, чтобы он игнорировал или маскировал запросы на обслуживание от какого-либо устройства или целой группы устройств. Программа-обработчик прерывания определяет, какое устройство выставило запрос на обслуживание, посредством чтения регистра карты прерываний в контроллере прерываний.

Векторизованный контроллер прерываний выполняет более гибкую обработку запроса на обслуживание. Такой контроллер позволяет задавать приоритеты прерываний и упрощает процесс определения устройства, которое вызвало прерывание. После определения источника прерывания и его приоритета VIC-контроллер выставляет сигнал прерывания для микроконтроллера только в том случае, если приоритет только-что инициированного запроса выше, чем у прерывания, которое обслуживается в данный момент. В зависимости от способа реализации, векторизованный контроллер прерываний может либо вызвать стандартный обработчик прерывания, который, в свою очередь, может использовать адрес обработчика прерывания из VIC-контроллера, либо передать обработку прерывания микроконтроллеру. Во втором случае микроконтроллер вызывает обработчик прерывания напрямую.

Важнейшей функцией микроконтроллера является управление ресурсами памяти системы. Современные устройства ARM включают блоки управления памятью, архитектура которых может отличаться в зависимости функций, заложенных в устройство фирмами-производителями, однако принципы их построения одинаковы. В большинстве современных микроконтроллеров для хранения программного кода используется флэш-память, а данные хранятся в статической памяти.

Координация доступа к регионам памяти, занимаемых программным кодом и данными на аппаратном уровне, как раз и осуществляется посредством блока управления памятью. На уровне программы пользователя управление ресурсами памяти осуществляется посредством стандартных библиотечных функций

C (**malloc**, **realloc**, **free** и т. д.), которые оперируют с регионами памяти, находящимися в куче (heap). Эффективность использования и распределения памяти для работающего приложения существенно влияет на производительность системы, поэтому все современные компиляторы (Keil, IAR и др.) имеют целый ряд опций для настройки распределения ресурсов памяти.

Для управления периферийными устройствами и контроллером памяти в микроконтроллерах ARM реализована шинная архитектура. Эта архитектура кардинально отличается от той, что используется, например, в персональных компьютерах на базе процессоров x86 (Intel). В персональных компьютерах все внешние устройства подсоединяются к процессору через одну их шин PCI и/или PCI-Express, которые являются внешними по отношению к процессору. В противоположность этому, все шины микроконтроллеров ARM реализованы на кристалле процессора.

Все устройства, присоединеные к внутренней шине микроконтроллера ARM, могут работать в одном из двух режимов: ведущего (master) или ведомого (slave). Сам ARM микроконтроллер всегда работает в режиме ведущего – это означает, что он инициирует запросы на обмен данными для устройств, находящихся на шине. Периферийные устройства могут работать только в режиме ведомого и осуществлять обмен данными по запросу ведущего.

Архитектуру шины микроконтроллера можно представить в виде двух уровневой модели. Первый уровень реализует непосредственные физические соединения посредством электрических сигналов и характеризуется разрядностью шины, которая может варьироваться от 16 до 64. Второй уровень характеризуется тем, что здесь используются определенные протоколы – система логических правил, в соответствии с которыми выполняется обмен данными между процессором и периферийными устройствами.

В микроконтроллерах ARM применяется архитектура шин под названием AMBA (Advanced Microcontroller Bus Architecture), разработанная около двадцати лет тому назад. Эта архитектура была адартирована для применения с ARM микроконтроллерами. Первыми такими адаптированными шинами были ASB (ARM System Bus) и APB (ARM Peripheral Bus). Позже была разработана шина AHB (ARM High Performance Bus), которая в настоящее время применяется в большинстве микроконтроллеров.

Преимуществом использования шины AMBA является то, что разработчики периферийных устройств микроконтроллеров могут использовать одно и то же устройство во многих проектах без каких-либо изменений аппаратно-архитектурного решения. Если сравнивать различные модификации шин, то AHB способна обеспечить более высокую пропускную способность по сравнению с ASB. Это объясняется тем, что AHB является мультиплексированной шиной с централизованным управлением, в то время как ASB базируется на принципе двунаправленного потока данных. По этой причине шина AHB может функционировать при высоких тактовых частотах, а разрядность данных на этой шине может достигать 64 или 128 бит.

На Рис. 1.2 показана конфигурация кристалла ARM микроконтроллера с

тремя шинами: AHB – для подсоединения высокопроизводительных периферийных устройств, APB – для работы с низкоскоростными устройствами и внешняя шина для подсоединения внешних устройств. Обратите внимание на то, что для устройств на внешней шине требуется мост для перехода на шину AHB.

1.3. Программное обеспечение для систем с ARM микроконтроллерами

Программное обеспечение встроенных систем на микроконтроллерах в самом общем случае может состоять из четырех основных компонентов, показанных на Рис. 1.3. Каждый программный компонент в этом стеке использует определенный уровнь абстракции для разделения программного кода и аппаратного устройства, управляемого этим кодом. Код инициализации выполняется первым при сбросе или перезагрузке системы и определяется специфическими характеристиками данного типа процессора, архитектурой прерываний и системой управления памятью. Код инициализации обычно очень короткий, поскольку его основная функция – сконфигурировать базовые компоненты системы и передать управление загрузчику операционной системы.

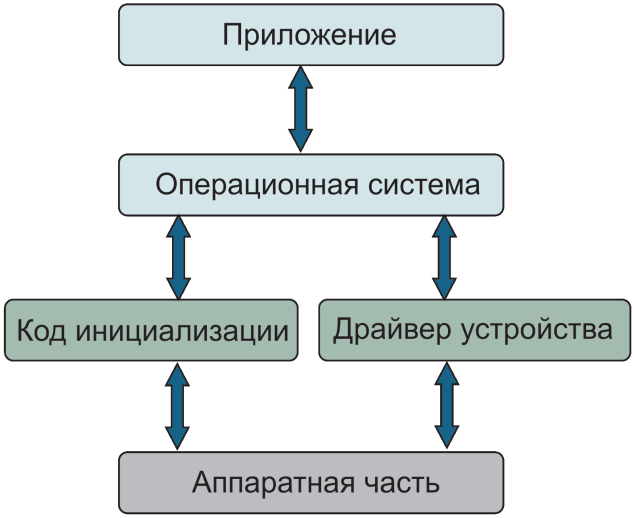

Рис. 1.3.

Операционная система выстраивает необходимую инфраструктуру для управления приложениями и аппаратными ресурсами. Для многих встроенных и переносных систем вообще не требуется комплексная операционная система – в этих случаях можно обойтись каким-либо простым менеджером заданий. Общая структура такого менеджера (Рис. 1.4), управляемого событиями, характерна для многих простых операционных систем реального времени (RTOS).

В такой системе менеджер заданий представляет собой программно реализованный "суперцикл" – это может быть обычный цикл с оператором **`while`** в C++. В нем каждому отдельному заданию для выполнения выделяется определенный промежуток (квант) времени, который контролируется посредством аппаратного таймера, входящего в состав встроенного периферийного оборудования микроконтроллера ARM.

По завершению кванта аремени менеджер заданий передает управление следующему в цепочке заданию и получает статус только-что выполненной операции (завершена, приостановлена и т. д.). Алгоритм управления заданиями может включать и дополнительный программный код, например, для возобновления приостановленного задания или продолжения выполнения незавершившегося задания в следующем кванте времени и т. д.

Рис. 1.4.

Вернемся к Рис. 1.3. Одним из программных компонентов, который управляет непосредственно периферийным устройством, является драйвер устройства. Понятие "драйвер устройства" для встроенных систем может интерпретироваться в широких пределах, в зависимости от структуры программного обеспечения системы в целом.

В простейших случаях, когда приложение пользователя управляет всеми периферийными устройствами системы, драйвер устройства – это фрагмент программного кода в самом приложении. Если приложения пользователя управляются комплексной операционной системой, то драйвер устройства разрабатывается как отдельное приложение с соответствующим программным интерфейсом для взаимодействия с другими уровнями программного обеспечения.

В любом случае конфигурация драйвера устройства во многом определяется требованиями к встроенной системе, а также тем, управляется ли система обычным приложением или же с помощью операционной системы. В следующих главах будут рассмотрены практические аспекты программирования встроенных систем на базе микроконтроллеров ARM, а также методы отладки и оптимизации программного кода.

ГЛАВА 2

ИНСТРУМЕНТЫ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ ARM

Качество программ, разработанных для любой современной аппаратной платформы, жизненный цикл их разработки, а также возможности расширения и модернизации уже существующих приложений во многом определяются качеством применяемых инструментальных средств разработки. Для каждой платформы имеется множество инструментов разработки приложений на языках высокого уровня, каждое из которых обладает определенными преимуществами и недостатками. Для разработчиков C/C++ приложений созданы многочисленные инструментальные средства, позволяющие не только быстро создавать приложения, но и выполнять многие трудоемкие этапы создания приложений, такие, как отладка и оптимизация в сжатые сроки.

Среди множества инструментальных средств для разработки C/C++ приложений на определенной аппаратной платформе всегда можно выделить наиболее развитые, и по этой причине наиболее популярные инструментальные средства. Например, наиболее популярным инструментом разработки C/C++ приложений в операционных системах Windows на платформе x86 де-факто является Microsoft Visual C++. Многие принципы, используемые в этой среде, так или иначе реализованы в других средствах разработки третьих фирм, поэтому подходы, заложенные в Visual C++, стали фактически стандартными для разработчиков, использующих другие инструментальные средства.

Для платформы ARM7 и последующих в этой линейке фактическим стандартом стала среда разработки Keil C для ARM, работающая в операционных системах Windows. Эта глава посвящена базовым принципам разработки приложений для ARM микропроцессоров в этой среде.

2.1. Среда разработки Keil C и интерфейс пользователя µVision IDE

Средства разработки, входящие в состав пакета Keil C, интегрированы в графический интерфейс пользователя, известный под названием µVision IDE (мы будем работать с версией 4.0). Среда µVision IDE представляет собой многооконный интерфейс, как и большинство подобных пакетов, а также включает текстовый редактор для набора и редактирования исходных текстов программ и мастер проектов, который позволяет легко создавать шаблоны приложения

для конкретного типа ARM процессора. Мастер проектов выполняет создание приложения в несколько шагов, интуитивно понятных разработчику, оставляя, впрочем, возможности для ручной настройки параметров проекта уже после создания шаблона.

Средства разработки C/C++ приложений включают компилятор исходных текстов программ на C/C++, продвинутый макроассемблер, компоновщик (линкер) и генератор файлов в формате HEX (HEX-файлов). Среда µVision IDE включает интегрированную утилиту Make, позволяющую выполнить сборку, компиляцию и компоновку приложения в автоматическом режиме. В большинстве случаев использование Make вполне оправдано, однако разработчики при необходимости могут использовать и свои собственные скрипты, в которых могут быть заложены какие-то спефические требования. Кроме того, разработчик может настраивать параметры компилятора и линкера, вызывая определенные диалоговые окна и устанавливая параметры вручную. Во всех проектах из этой книги мы будем использовать утилиту Make для сборки приложений по умолчанию.

Будучи продвинутой средой разработки приложений, µVision включает отладчик/симулятор, а также специальный интерфейс для загрузчика ULINK Debug Adapter. Здесь я сделаю одно важное замечание. Для демонстрации работы программного кода мы будет использовать симуляторы аппаратных средств, что исключает необходимость приобретения платы (модуля) разработки с микроконтроллером ARM. Конечно, в этом случае вы не сможете записать код программы во флэш-память и запустить его на выполнение на железе. Тем не менее, симулятор Keil позволяет с высокой достоверностью моделировать аппаратный интерфейс микроконтроллера, не говоря уже о выполнении задач вычислительного характера, поэтому программный код, протестированный в симуляторе, будет работать и на реальном железе.

Обычно для адаптации программного кода для конкретного типа микроконтроллера требуются минимальные средства, поскольку среда µVision IDE позволяет генерировать программный код для конкретного типа устройства (как это делается, мы увидим на последующих примерах). Конечно, использование симулятора не может на все 100% заменить реальный физический микроконтроллер, однако большинство задач создания программного кода и его отладки могут быть успешно решены и в отсутствие реальных аппаратных средств.

Для построения всех примеров в среде Keil мы будем использовать демонстрационную версию Keil C с ограничением по размеру исполняемого кода 32K (версия 4.21). Несмотря на ограничение по размеру исполняемого кода, даже в этом случае можно научиться разрабатывать довольно сложные приложения. Вначале мы посмотрим, как можно быстро создать простейшее приложение наподобие “Hello, World” в среде Keil C для ARM, затем в деталях проанализируем те этапы которые требуется выполнить для превращения исходных текстов написанных на языке C/C++ в исполняемый двоичный файл (приложение).

2.2. Программа “Hello, World!” в среде Keil

Вначале посмотрим, как создется программа “Hello, World” в Keil C. Для этого нужно иметь установленный пакет Keil RealView MDK для ARM на вашем ПК (как минимум, демонстрационную версию).

На первом шаге создаем новый проект в среде Keil. В этом и других пакетах программирования проект будет включать несколько файлов (модулей), которые участвуют в построении данного приложения. Итак, приступим к созданию нового проекта (назовем его Hello). Для этого в основном меню выбираем опцию **New µVision Projec**t (Рис. 2.1):

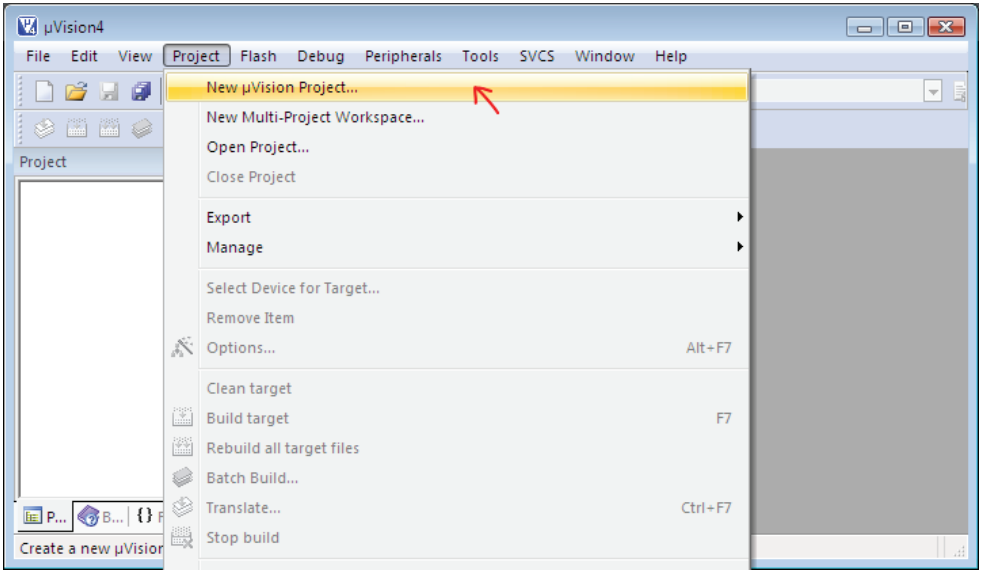

Рис. 2.1.

Данная опция позволяет создать проект, в который в процессе разработки будут добавлены необходимые файлы с исходными текстами программ, а также, при необходимости файлы заголовков и библиотеки функций. Проект можно интерпретировать как контейнер, в который или помещаются какие-то объекты или из него извлекаются (удаляются) ненужные объекты. Структура проекта полностью описывается файлом проекта с расширением .uvproj.

В открывшемся диалоговом окне выберем или создадим каталог, в котором будет сохранен наш проект (пусть это будет каталог Hello) и сохраним проект под именем Hello (Рис. 2.2).

После нажатия кнопки **Save** будет создан файл проекта Hello.uvproj. Файл проекта описывает все объекты проекта и связи между ними. На практике файл проекта является не чем иным, как XML-файлом, который, в принципе, может редактироваться вручную, если вдруг исходный файл по каким-то причинам был потерян.

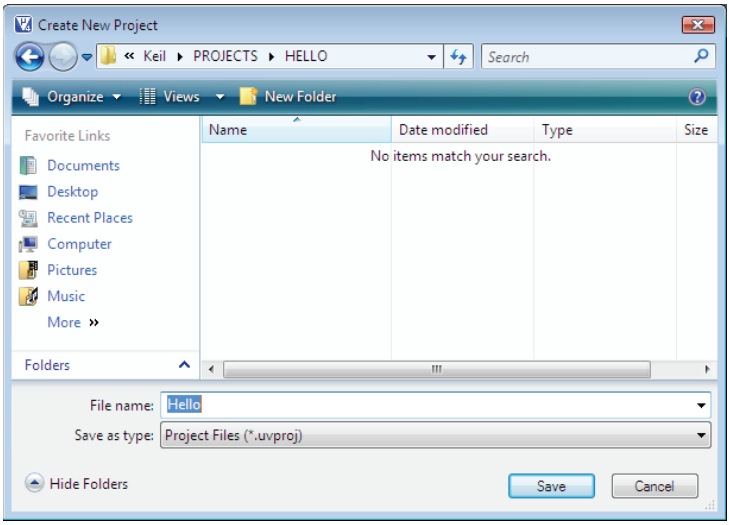

Рис. 2.2.

Содержимое части этого файла, открытого с помощью редактора MS Word, показано на Рис. 2.3. Если разрабатывается сложный проект, содержащий много настроек и много объектов, то полезно иметь сохраненную копию файла проекта, особенно на какой-то определенной стадии. Мастер проектов Keil также создает резервные копии файла проекта после изменений, но вам может понадобиться какая-либо промежуточная версия, которая к этому моменту времени уже могла быть перезаписана!

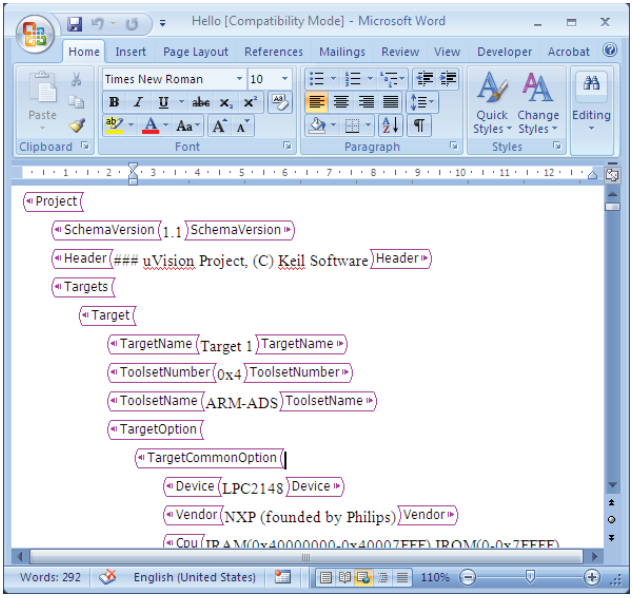

Рис. 2.3.

В меню **Project** (см. Рис. 2.1) имеются еще две опции, которые в данный момент нам не нужны, но которые могут понадобится в дальнейшей работе над проектами. Опция **Open Project...,** как следует из названия, позволяет открыть один из уже существующих проектов, хотя если вы постоянно работаете над каким-то проектом, то быстрее будет сделать то же самое, выбрав один из последних проектов в списке недавно открываемых проектов в нижней части меню.

Еще одна опция, **New Multi-Project Workspace**, предназначена для расширения возможностей по управлению проектами путем создания "рабочей области". Здесь нужно объяснить концепцию "рабочей области" или, что то же самое, "рабочего пространства" для нескольких проектов. В нашем случае при создании проекта Hello новый рабочий проект связан по умолчанию с одним рабочим пространством. Если бы возникла необходимость разработать, скажем, два проекта, в каждом из которых нужно было бы использовать файлы, общие для обоих проектов, то оба проекта можно было бы отладить в одном рабочем пространстве, что сэкономило бы время.

Еще одна ситуация, когда удобно использовать рабочее пространство, возникает, когда разработчик создает приложение для двух или более разнотипных ARM микропроцессоров, используя одни и те же исходные тексты программ. В этом случае можно создать несколько проектов для разных устройств и включить их в одно рабочее пространство.

Вернемся к нашему проекту. В следующем диалоговом окне мастер проектов предложит выбрать тип микроконтроллера от конкретного производителя (Рис. 2.4):

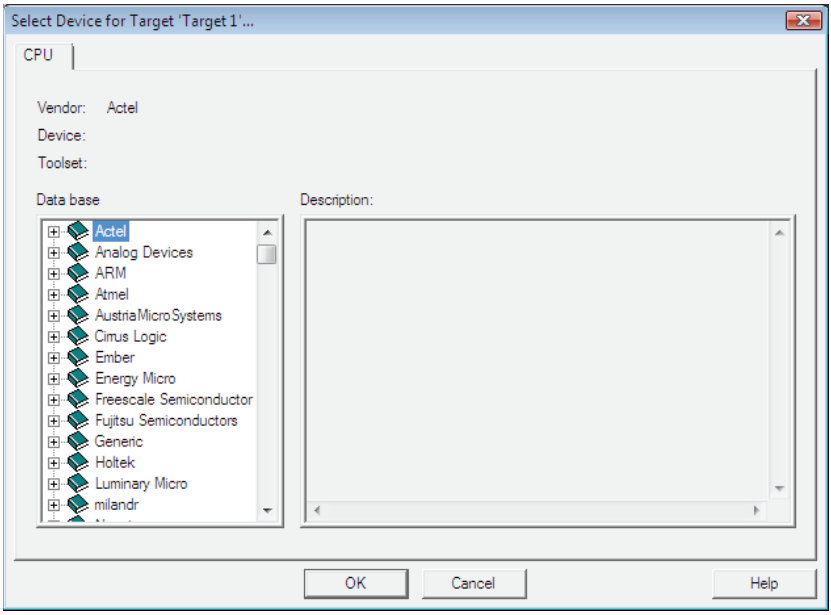

Рис. 2.4.