Содержание

Автор

Даниэле Лакамера – технолог и разработчик программного обеспечения. Он является экспертом в области операционных систем и TCP/IP, имеет более 20 академических публикаций по оптимизации транспортных протоколов. Начал свою карьеру в качестве разработчика ядра Linux и свой первый вклад внес в версию Linux 2.6.

С 2012 года работает над архитектурами на основе микроконтроллеров, уделяя особое внимание проектированию, разработке и интеграции программного обеспечения для встраиваемых систем. Является активным участником многих проектов свободного программного обеспечения и соавтором стека TCP/IP и операционной системы (ОС) POSIX для устройств Cortex-M, которые распространяются под лицензией GPL. В настоящее время его деятельность сосредоточена на безопасности IoT, криптографии, безопасной загрузке и пользовательских транспортных протоколах.

Рецензент

Марко Оливерио получил докторскую степень в Университете Калабрии за диссертацию по защите ОС от атак по побочным каналам и аппаратных атак. Некоторые из его научных публикаций были представлены на значимых конференциях, посвященных ОС. После защиты докторской диссертации он начал работать разработчиком микропрограмм и встраиваемых систем, участвуя в нескольких проектах с открытым исходным кодом.

Научный редактор русского перевода

Романов Александр Юрьевич (https://www.hse.ru/staff/a.romanov) –научный редактор русского перевода данной книги, доцент Московского института электроники и математики им. А. Н. Тихонова Национального исследовательского университета «Высшая школа экономики» (МИЭМ НИУ ВШЭ). С 2014 г. работает в МИЭМ НИУ ВШЭ, где возглавляет лабораторию САПР (https://miem.hse.ru/edu/ce/cadsystem), специализирующуюся на проектной деятельности, а также разработке цифровых систем на ПЛИС/микроконтроллерах, робототехнических комплексов, аппаратных реализаций систем искусственного интеллекта, многопроцессорных систем, систем удаленного доступа к лабораторному оборудованию и т. д. В 2015 г. защитил диссертацию в Институте проблем проектирования в микроэлектронике РАН (г. Зеленоград), является автором более 200 научных статей, патентов и книг. Преподает встраиваемые системы с 2009 г.

Предисловие

За последние два десятилетия встраиваемые микроконтроллерные системы получили широкое распространение благодаря достижениям производителей микросхем и разработчиков микроэлектроники, стремящихся к увеличению вычислительной мощности и уменьшение размера логических узлов микропроцессоров и периферийных устройств.

Проектирование, реализация и интеграция программных компонентов для этих систем в большинстве случаев требуют прямого обращения к аппаратным функциям, когда приложения выполняются в одном потоке и нет операционной системы, обеспечивающей абстракции для доступа к функциям процессорного ядра и внешним периферийным устройствам. По этой причине разработка встраиваемых систем считается отдельной областью в индустрии разработки программного обеспечения, в которой подход разработчика и рабочий процесс должны быть соответствующим образом адаптированы к особенностям среды выполнения.

Эта книга кратко объясняет особенности аппаратной архитектуры типичной встраиваемой системы, знакомит с инструментами и методологиями, необходимыми для начала разработки архитектуры целевой системы, а затем проводит читателей через взаимодействие с системными функциями и периферийными устройствами. Некоторые области, такие как энергоэффективность и подключение к коммуникационным сетям, рассматриваются подробнее, чтобы дать более полное представление о методах, используемых для проектирования маломощных и подключенных систем. Далее в книге рассматривается более сложный проект, включающий упрощенную операционную систему реального времени, которая строится шаг за шагом, начиная с реализации отдельных системных компонентов. Наконец, во втором издании добавлен подробный анализ реализации TrustZone-M – технологии TEE, представленной ARM в новейшем семействе встраиваемых микроконтроллеров.

В этой книге основной акцент сделан на конкретных механизмах защиты и безопасности, включая практические примеры использования технологий, призванных повысить устойчивости системы к ошибкам программирования в коде приложения или даже к злонамеренным попыткам нарушить ее целостность.

Для кого эта книга

Если вы программист или инженер, который хочет освоить область разработки и программирования встраиваемых систем, эта книга для вас. Вы также найдете эту книгу полезной, если вы неопытный или начинающий про-

граммист встраиваемых систем, желающий расширить свои знания. Более опытным разработчикам встраиваемого программного обеспечения наша книга может пригодиться для обновления знаний о драйверах устройств, безопасности памяти, безопасной передаче данных, разделении привилегий и безопасных областях выполнения кода.

Краткое содержание книги

Глава 1 представляет собой введение во встраиваемые системы на базе микроконтроллеров. В ней также раскрывается тематический охват книги, от широкого определения встраиваемых систем до более конкретной архитектуры, на которой базируются все практические примеры – 32-битные микроконтроллеры с отображением физической памяти.

Глава 2 описывает используемые инструменты и рабочий процесс разработки. Это введение в наборы инструментов, отладчики и эмуляторы, которые можно использовать для создания кода в двоичном формате для последующей загрузки и запуска на целевой платформе.

Глава 3 посвящена стратегиям и методологиям командной разработки и тестирования. Эта глава содержит описание процессов, которые обычно применяются при разработке и тестировании программного обеспечения для встраиваемых систем.

В главе 4 детально рассмотрен этап загрузки (запуска, boot-up) встроенной системы, анализируются этапы загрузки и загрузчики. Она содержит подробное описание кода запуска и механизмов, используемых для разделения программного обеспечения на несколько этапов загрузки.

В главе 5 описаны некоторые оптимальные стратегии управления памятью. Показаны наиболее распространенные ловушки и дается объяснение, как избежать ошибок выделения памяти, которые могут привести к непредсказуемому поведению или полному выходу системы из строя.

В главе 6 рассматривается доступ к выводам GPIO и другим встроенным периферийным устройствам общего назначения. Это первое взаимодействие целевой платформы с внешним миром, использующее электрические сигналы для выполнения простых операций ввода/вывода.

Глава 7 проведет вас через интеграцию контроллеров последовательной шины (UART, SPI и I2C). Детальный анализ наиболее распространенных протоколов связи по шине сопровождается подробными примерами кода, необходимого для взаимодействия с приемопередатчиками, обычно доступными во встроенных системах.

В главе 8 рассматриваются методы снижения энергопотребления в энергосберегающих системах. Проектирование встраиваемых систем с низким и сверхнизким энергопотреблением требует выполнения определенных шагов для достижения требуемого уровня экономичности.

В главе 9 представлены протоколы и интерфейсы, необходимые для построения распределенных и подключенных систем. Системы IoT должны взаимодействовать с удаленными конечными точками, применяя стандартные сетевые протоколы, реализованные с использованием сторонних биб-

лиотек. Особое внимание уделяется защите связи между конечными точками с помощью защищенных сокетов.

В главе 10 объясняется принцип функционирования многозадачной операционной системы через реализацию планировщика задач в реальном времени. В этой главе предлагаются три подхода к реализации операционных систем для микроконтроллеров с нуля с использованием различных планировщиков.

В главе 11 описаны механизмы TEE, обычно доступные во встраиваемых системах, и приводится пример запуска кода в защищенных и незащищенных областях с использованием ARM TrustZone-M. На современных микроконтроллерах TEE предоставляет возможность защитить определенные области памяти или периферийных устройств, ограничив их доступ из незащищенной области выполнения.

Как получить максимальную пользу от этой книги

Ожидается, что вы владеете языком C и понимаете, как работают компьютерные системы. Для запуска кода практических примеров требуется рабочий компьютер с установленной платформой GNU/Linux. Для полного понимания реализованных механизмов иногда потребуется пошаговое изучение примеров кода. Вам предлагается изменять, улучшать и повторно использовать предоставленные примеры, применяя полученные знания.

ПО и оборудование, рассмотренные в этой книге	**Необходимая операционная система**
Набор STM32F4DISCOVERY	
Плата STM32L5 Nucleo-144	
Инструменты разработки GNU ARM	Windows, macOS, Linux
GNU Make	
OpenOCD	

Дополнительные инструкции по использованию необходимых инструментов разработки приведены в главе 2.

Если вы используете цифровую версию этой книги, вы можете скачать файловый архив книги с сайта издательства. Это поможет вам избежать возможных ошибок, связанных с копированием и вставкой кода.

Используемые соглашения

В этой книге используется ряд соглашений по оформлению текста.

`Код в тексте`: обозначает кодовые слова в тексте, имена таблиц базы данных, имена папок, имена файлов, расширения файлов, пути, фиктивные URL-адреса, пользовательский ввод и дескрипторы. Пример: «При вызове из

командной строки должен быть указан один файл конфигурации с несколькими платформами и конфигурациями платы разработки, расположенными в каталоге /scripts».

Блок кода оформлен следующим образом:

```
/* Переход к небезопасной области кода */
asm volatile("mov r12, %0" ::"r"
     ((uint32_t)app_entry - 1));
asm volatile("blxns r12" );
```

Когда необходимо привлечь внимание читателя к определенной части блока кода, соответствующие строки или элементы выделяются жирным шрифтом:

```
Secure Area 1:
  SECWM1_PSTRT : 0x0 (0x8000000)
  SECWM1_PEND : 0x39 (0x8039000)
```

Ввод или вывод командной строки записывается следующим образом:

```
$ renode /opt/renode/scripts/single-node/stm32f4_discovery.resc
```

Команды для консоли отладчика записываются следующим образом:

```
> add-symbol-file app.elf 0x1000
> bt full
```

Советы и важные примечания

Оформлены, как показано здесь.

Предисловие от научного редактора русского перевода

Дорогие друзья!
Книга, которую вы держите в руках, продолжает серию книг «Книжная полка истового инженера», которую выпускает издательство «ДМК Пресс» совместно с Московским институтом электроники и математики им. А. Н. Тихонова НИУ ВШЭ при поддержке группы компаний YADRO (yadro.com).

Если вы интересуетесь цифровой электроникой, разработкой на ПЛИС, проектированием на языках описания аппаратуры Verilog или VHDL, то вы, скорее всего, уже знакомы с книгами по цифровому синтезу, такими как: Д. Харрис и С. Л. Харрис «Цифровая схемотехника и архитектура компьютера», «Цифровая схемотехника и архитектура компьютера: RISC-V», «Цифровая схемотехника и архитектура компьютера. Дополнение по архитектуре ARM»; «Цифровой синтез: практический курс» (под ред. А. Ю. Романова и Ю. В. Панчула), Ф. Бруно «Программирование FPGA для начинающих» и др. (воспользуйтесь QR-кодами, чтобы узнать об упомянутых книгах).

Эта книга несколько отличается от упомянутых выше и посвящена программированию на классическом языке C++, продолжая тренд, заданный еще одной книгой из этой серии – «C++20 в деталях» (автор: Р. Гримм). Только, если в той книге анализируются последние новшества и высокоуровневые возможности C++, то в этой, наоборот, рассмотрены особенности программирования на самом низком уровне встраиваемых систем.

Прочитав эту книгу вы узнаете, как устроены встраиваемые системы, как организовано программирование, верификация и отладка для таких систем, как происходит загрузка программного кода, управление памятью, взаимодействие с периферией, как устроены стандартные шинные интерфейсы, как управлять питанием и энергосбережением, как устроена многопоточность на уровне операционной системы, как реализовано управление доверен-

ной средой выполнения, и, наконец, как строятся распределенные системы и системы Интернета вещей. То есть все то, что еще называют системным программированием, когда разработчик своей программой взаимодействует с устройством непосредственно без абстрагирования и безопасного кода.

Интересно и подробно излагаемый материал дополнен хорошими и рабочими примерами программного кода, доступными для скачивания из репозитория.

Эта книга будет интересна тем, кому уже надоели проекты на Arduino и хочется перейти к чему-то более мощному, как микропроцессоры от STM, но нет желания отдавать львиную долю производительности на надстройки в виде полноценной операционной системы реального времени на Raspberry PI или Jetson Nano. Эта книга для тех, кто хочет понять, как устроена операционная система изнутри, как работают привычные механизмы распараллеливания задач и разделение памяти. Она для тех, кто хочет разрабатывать компактный, энергоэффективный, производительный и безопасный код на самом низком уровне.

С другой стороны, эта книга будет интересна и программистам кода, выполняемого на высоком уровне. Она позволит лучше понять то системное программное и аппаратное окружение, в котором выполняется высокоуровневая программа, как устроены драйвера устройств и системные библиотеки, и расширить свой кругозор.

Таким образом, аудитория этой книги: широкий круг читателей от студентов технических вузов, до разработчиков аппаратуры и программистов различных профилей. Она требует наличия базовых знаний синтаксиса, основных концепций и принципов программирования на C++. Также для лучшего освоения материала и апробации приведенных примеров может потребоваться отладочная плата с STM32.

В добрый путь!

Александр Юрьевич Романов,
к. т. н., доцент ДКИ МИЭМ НИУ ВШЭ,
преподаватель курсов «Проектирование систем на кристалле»,
«Системное проектирование цифровых устройств»
и «Системы искусственного интеллекта»
г. Москва, Россия

Часть I

ВВЕДЕНИЕ В РАЗРАБОТКУ ВСТРАИВАЕМЫХ СИСТЕМ

В первой главе этой книги представлен общий обзор встраиваемых систем и показано, чем разработка таких систем отличается от других направлений инженерной деятельности, с которыми вы можете быть знакомы. Вторая глава помогает превратить рабочий компьютер инженера в настоящую лабораторию для разработки аппаратного и программного обеспечения и оптимизировать действия, необходимые для создания, тестирования, отладки и развертывания встроенного программного обеспечения.

Первая часть состоит из следующих глав:

- главы 1 «Встроенные системы с практической точки зрения»;
- главы 2 «Рабочая среда и оптимизация рабочего процесса».

Глава 1

Встраиваемые системы с практической точки зрения

Проектирование и разработка программного обеспечения для встраиваемых систем сопряжены с иным набором задач, чем традиционная разработка приложений высокого уровня.

Первая глава знакомит читателей с основными понятиями, компонентами и платформами, которые будут встречаться на протяжении всей книги.

В этой главе обсуждаются следующие темы:

- определение предметной области;
- универсальный ввод/вывод;
- интерфейсы и периферия;
- подключенные системы;
- введение в механизмы изоляции;
- эталонная платформа.

1.1. Определение предметной области

Встраиваемые системы – это вычислительные устройства, которые выполняют определенные задачи без прямого или постоянного взаимодействия с пользователем. Из-за разнообразия рынков и технологий они имеют разные формы и размеры, но преимущественно это устройства небольшого размера с ограниченным количеством ресурсов.

В этой книге основные понятия встраиваемых систем будут проанализированы с точки зрения разработки программных компонентов, взаимодействующих с ресурсами и периферийными узлами устройства. Начнем с определения области применения методов и архитектурных шаблонов, описанных в нашей книге, в рамках более широкого определения встраиваемых систем.

1.1.1. Встраиваемые Linux-системы

Значительную часть рынка встраиваемых систем занимают устройства, аппаратных ресурсов которых достаточно для запуска разновидностей операционной системы GNU/Linux. Рассмотрение таких систем выходит за рамки этой книги, поскольку их разработка в значительной степени опирается на различные стратегии комбинирования и интеграции компонентов. Типичная аппаратная платформа, на которой может работать система на основе ядра Linux, оснащена достаточно большим объемом оперативной памяти (до нескольких гигабайт) и достаточным объемом энергонезависимой памяти для хранения всех программных компонентов, предоставляемых в дистрибутиве GNU/Linux.

Кроме того чтобы менеджер памяти Linux предоставлял отдельные виртуальные адресные пространства каждому процессу в системе, устройство должно быть оснащено *блоком управления памятью* (memory management unit, MMU) – аппаратным компонентом, помогающим операционной системе (ОС) преобразовывать физические адреса в виртуальные, и наоборот, во время выполнения прикладной программы.

Устройства с такими характеристиками нужны далеко не всегда, поэтому для снижения производственных затрат на практике стараются использовать системы с более простой конструкцией.

Производители оборудования и разработчики микросхем постоянно ищут новые методы повышения производительности систем на основе микроконтроллеров. За последнее десятилетие появились новые поколения платформ, позволяющие снизить стоимость оборудования, сложность встроенного ПО, размер и энергопотребление, сохраняя при этом набор функций, наиболее интересных для рынка встраиваемых систем.

В некоторых случаях встраиваемые вычислительные устройства должны выполнять ряд задач в течение короткого, измеримого и предсказуемого промежутка времени. Такие устройства представляют собой *системы реального времени* (real-time system, RTS), принципиально отличающиеся от привычных нам многозадачных систем, таких как настольные компьютеры, серверы и мобильные телефоны.

Обработка в реальном времени – это цель, которую очень трудно, если вообще возможно, достичь на встраиваемых платформах Linux. Ядро Linux не предназначено для работы в жестком режиме реального времени, и даже если применяются патчи планировщика ядра, помогающие удовлетворить требования к работе в реальном времени, результаты несопоставимы со специализированными системами.

Встраиваемые устройства с батарейным и извлекаемым из среды питанием[1] по определению должны обладать низким энергопотреблением и энергосберегающими технологиями беспроводной связи. Большое количество ресурсов и повышенная аппаратная сложность систем на базе Linux не всегда

---

[1] Energy-harvesting devices – устройства со сверхнизким энергопотреблением, способные извлекать и накапливать электрическую энергию из внешних источников (свет, тепло, электромагнитные поля).

позволяют достичь необходимого уровня энергопотребления по сравнению с более простыми специализированными устройствами.

В этой книге рассматриваются встраиваемые системы на основе 32-разрядных микроконтроллеров, которые способны выполнять однопоточные приложения без ОС (bare-metal, на «голом железе»), а также поддерживают минимальные ОС реального времени. Такие системы широко применяются в промышленном производстве изделий, предназначенных для конкретных целей. В последнее время они все чаще служат аппаратной основой для более общих, многоцелевых платформ разработки.

1.1.2. 8-разрядные микроконтроллеры

В прошлом на рынке встраиваемых систем доминировали 8-битные микроконтроллеры. Их несложная конструкция позволяет разрабатывать небольшие приложения, которые могут выполнять набор предопределенных задач. Но они слишком просты и обладают небольшим количеством ресурсов, недостаточным для реализации даже примитивной операционной системы. Вдобавок характеристики 32-разрядных микроконтроллеров значительно эволюционировали, и теперь они охватывают все варианты использования 8-разрядных микроконтроллеров в том же диапазоне цены, размеров и энергопотребления.

В настоящее время 8-разрядные микроконтроллеры в основном занимают нишу на рынке обучающих платформ, предназначенных для ознакомления любителей и новичков с основами программирования для электронных устройств. В этой книге мы не рассматриваем 8-разрядные платформы, поскольку им не хватает характеристик, позволяющих использовать системное программирование, многопоточность и расширенные функции для создания профессиональных встраиваемых систем.

В контексте данной книги термин «встраиваемые системы» используется для обозначения класса устройств, которые основаны на микроконтроллерах, обладающих ограниченными ресурсами, но позволяют создавать системы реального времени с помощью функций, предоставляемых аппаратной архитектурой и системным программированием.

1.1.3. Аппаратная архитектура

Архитектура встраиваемой системы сосредоточена вокруг ее *микроконтроллера*, также иногда называемого *блоком микроконтроллера* (microcontroller unit, MCU). Обычно это одна интегральная схема, содержащая процессор, ОЗУ, флеш-память, последовательные приемники и передатчики и другие базовые компоненты. Рынок предлагает множество различных вариантов архитектур микроконтроллеров, поставщиков, ценовых диапазонов, функций и интегрированных ресурсов. Обычно стараются разрабатывать недорогие автономные системы с низким потреблением энергии и ресурсов на

одной интегральной схеме. Поэтому их часто называют *системой на чипе* (System-on-Chip, SoC), или *однокристалльной системой*.

Из-за разнообразия процессоров, памяти и интерфейсов, которые можно интегрировать в чип, не существует единой эталонной архитектуры микроконтроллеров. Тем не менее некоторые архитектурные компоненты являются общими для широкого спектра моделей и брендов и даже для различных архитектур процессоров.

Одни микроконтроллеры предназначены для специализированных приложений и оснащены различными интерфейсами для связи с периферийными устройствами и внешним миром. Другие ориентированы на решения с минимальными затратами на оборудование или с жестко ограниченным потреблением энергии.

Тем не менее практически каждый современный микроконтроллер имеет следующий минимальный набор компонентов:

- микропроцессор;
- оперативная память (ОЗУ);
- флеш-память;
- последовательные приемопередатчики.

Кроме того, все больше устройств нуждаются во встроенном доступе к сети для связи с другими устройствами и шлюзами. Некоторые микроконтроллеры поддерживают либо хорошо зарекомендовавшие себя стандарты, такие как интерфейсы Ethernet или Wi-Fi, либо специальные протоколы, разработанные в соответствии с требованиями к встраиваемым системам, такие как радиоинтерфейсы с частотой до 1 ГГц или шина *локальной сети контроллеров* (controller area network, CAN), и частично или полностью реализованные в рамках интегральной схемы микроконтроллера.

Все компоненты должны иметь общую шину с процессором, который отвечает за координацию работы периферийной логики. ОЗУ, флеш-память и управляющие регистры приемопередатчиков отображаются в одном и том же физическом адресном пространстве (рис. 1.1).

Адреса, по которым отображаются ОЗУ и флеш-память, зависят от конкретной модели микроконтроллера и обычно указаны в техническом описании. Микроконтроллер выполняет код на своем родном машинном языке – последовательности инструкций, переданных в виде двоичного исполняемого файла. Формат команд и файла зависит от архитектуры микроконтроллера. Исполняемый файл обычно получается в результате компиляции и сборки исходного кода программы с последующим преобразованием выходного файла компилятора в машинный код целевого микропроцессора.

Часть процессора предназначена для выполнения инструкций, которые хранятся в двоичном формате непосредственно в ОЗУ, а также во внутренней флеш-памяти. Обычно выполнение кода начинается с нулевого адреса в памяти или со специального адреса, указанного в техническом описании микроконтроллера. Микропроцессор быстрее всего извлекает и выполняет код из ОЗУ, но постоянная прошивка хранится во флеш-памяти, емкость которой обычно больше, чем ОЗУ почти у всех микроконтроллеров, и позволяет сохранять данные при выключении питания и перезагрузках.

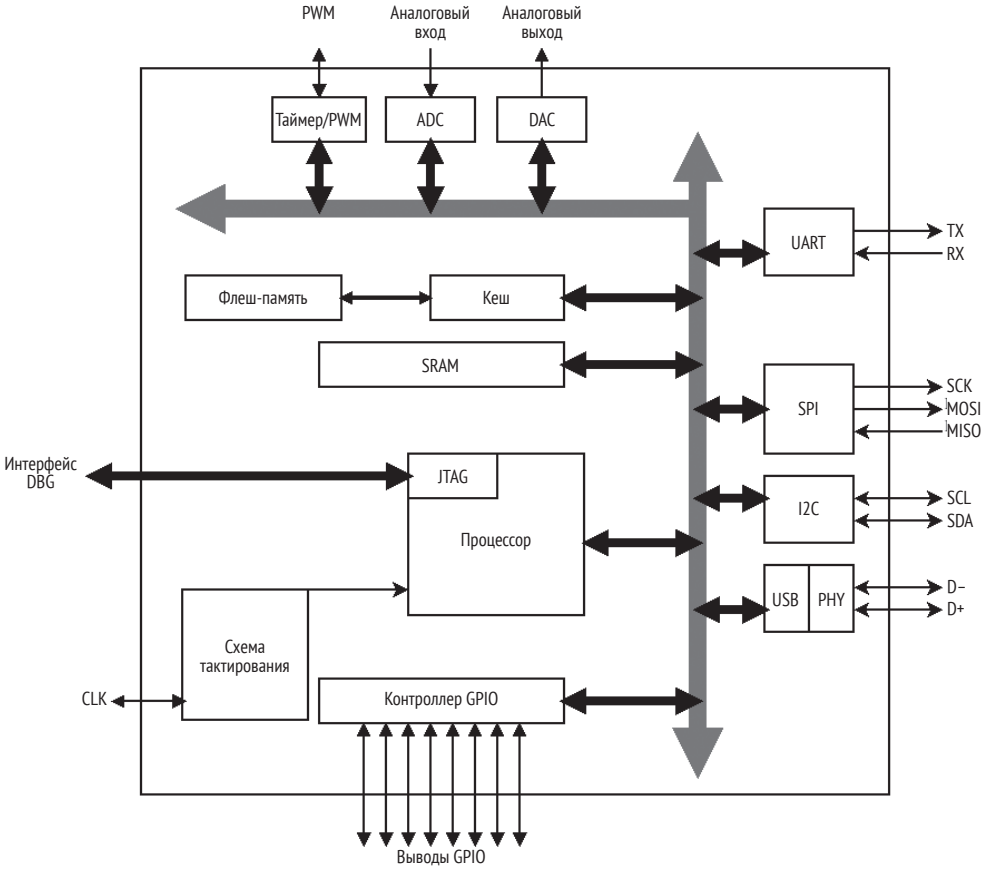

Рис. 1.1 ❖ Упрощенная блок-схема компонентов универсального микроконтроллера

Для компиляции исполняемого файла и загрузки его во флеш-память микроконтроллера требуется компьютер (хост-машина), оснащенный необходимыми аппаратными и программными средствами. Для создания правильного машинного кода из текста программы на языке высокого уровня необходимо указать компилятору характеристики и требования целевого микроконтроллера. По многим уважительным причинам наиболее популярным языком программирования для разработки программного обеспечения встраиваемых систем является C, хотя это не единственный доступный вариант. Для программирования встраиваемых систем можно использовать языки и более высокого уровня, такие как Rust или C++.

Примечание

В этой книге основное внимание будет уделено программированию на C, потому что он менее абстрактный, чем любой другой язык высокого уровня, что упрощает понимание поведения базового оборудования при просмотре кода.

Все современные платформы встроенных систем также имеют по крайней мере один механизм (например, JTAG) для отладки и загрузки программного обеспечения во флеш-память. При доступе к интерфейсу отладки с хост-машины отладчик может взаимодействовать с блоком точек останова (breakpoint) в процессоре, прерывая и возобновляя выполнение, а также выполнять чтение и запись любого адреса в памяти.

Значительную часть программирования встраиваемых систем составляет реализация обмена данными с периферийными устройствами через собственные интерфейсы микроконтроллера. Для разработки встраиваемого программного обеспечения требуются базовые знания в области электроники, способность разбираться в схемах и технических описаниях, а также уверенное обращение с измерительными приборами, такими как логические анализаторы или осциллографы.

1.1.4. Типичные затруднения

Разработка встраиваемых систем требует постоянного внимания к спецификациям и аппаратным ограничениям. Это постоянный поиск компромисса между наиболее эффективным способом выполнения конкретных задач и использованием очень ограниченных ресурсов. Разработчики встраиваемых систем постоянно встречаются с типичными затруднениями, которые редко встречаются в других средах. Вот несколько примеров таких затруднений:

- во флеш-памяти не хватает места для реализации новой функции;
- недостаточно оперативной памяти для хранения сложных структур или копирования больших буферов данных;
- не хватает быстродействия процессора для своевременного выполнения всех необходимых вычислений и обработки данных;
- устройство с питанием от электрической батареи потребляет слишком много электроэнергии.

Кроме того, операционные системы персональных компьютеров и смартфонов широко используют MMU – компонент процессора, который позволяет выполнять преобразования между физическими и виртуальными адресами на аппаратном уровне.

MMU является необходимой абстракцией для реализации разделения адресного пространства между задачами, а также между задачами и самим ядром. Микроконтроллеры не имеют MMU, и обычно у них нет достаточного объема энергонезависимой памяти для хранения ядер, приложений и библиотек. По этой причине встраиваемые системы часто работают в рамках одной задачи, при этом основной цикл программы выполняет всю обработку данных и обмен данными в определенном порядке. Некоторые устройства могут запускать встроенные ОС, которые намного проще, чем их аналоги для компьютеров или смартфонов.

Разработчики компьютерных приложений часто рассматривают базовую систему просто как готовую среду выполнения с определенными свойствами,

в то время как при разработке встраиваемых систем нередко приходится создавать с нуля все компоненты, от процедуры загрузки до логики приложения. Во встраиваемой среде различные программные компоненты теснее связаны друг с другом из-за отсутствия более сложных абстракций, таких как разделение памяти между процессами и ядром ОС.

Разработчики, которые впервые обращаются к встраиваемым системам, обычно обнаруживают, что отладка некоторых систем представляет собой более сложный процесс, чем просто запуск программы и наблюдение за результатами. Это особенно верно для систем, которые имеют ограниченные возможности взаимодействия с человеком или полностью их лишены.

Для успешного программирования встраиваемых систем нужен правильно организованный рабочий процесс, включающий в себя четко определенные тестовые примеры, список ключевых показателей эффективности системы, полученных в результате анализа спецификаций для выявления возможностей компромиссов, инструменты и процедуры для выполнения всех необходимых измерений и хорошо отлаженный этап прототипирования.

В этом контексте особого внимания заслуживает безопасность. Как обычно, при разработке кода, выполняемого на системном уровне, разумно помнить о глобальных последствиях возможных ошибок. Код большинства встроенных приложений выполняется с расширенными привилегиями, и неправильное поведение одной задачи может повлиять на стабильность и целостность всей микропрограммы. Некоторые платформы предлагают специальные механизмы защиты памяти и встроенного разделения привилегий, которые полезны для построения отказоустойчивых систем даже при отсутствии полноценной ОС, основанной на разделении адресных пространств процессов.

1.1.5. Многопоточность

Одним из преимуществ использования микроконтроллеров, предназначенных для построения встраиваемых систем, является возможность запуска логически разделенных задач в отдельных исполнительных устройствах за счет разделения ресурсов по времени.

Самая распространенная разновидность встраиваемых программ представляет собой последовательно выполняемый код на основе главного цикла, где модули и компоненты образуют интерфейсы обратного вызова. Но современные микроконтроллеры предлагают разработчикам систем специальные средства для создания многозадачной среды и запуска логически разделенных приложений.

Эти средства особенно удобны при разработке более сложных систем реального времени; знакомство с ними помогает понять модель безопасности, основанную на изоляции процессов и сегментации памяти.

1.2. ОЗУ

«640 Кбайт памяти должно хватить всем».

– Билл Гейтс
(основатель и бывший директор Microsoft)

За последние три десятилетия это известное утверждение неоднократно цитировали, чтобы подчеркнуть технологический прогресс и выдающиеся достижения индустрии ПК. Сегодня многие программисты воспринимают эту фразу исключительно как шутку, но именно на такие объемы памяти приходится ориентироваться разработчикам встраиваемых систем спустя более 30 лет после первоначального выпуска MS-DOS.

Хотя сегодня большинство встраиваемых систем оснащены более объемной памятью, особенно благодаря наличию интерфейсов для подключения внешней памяти DRAM, самые простые устройства, которые можно запрограммировать на C, имеют всего 4 Кбайта ОЗУ для реализации всей системной логики. При проектировании встраиваемой системы необходимо тщательно оценивать объем памяти, потенциально необходимой для операций, выполняемых системой, и выделяемой под буферы для связи с периферийными модулями и внешними устройствами.

Модель памяти встраиваемой системы проще, чем у ПК и мобильных устройств. Доступ к памяти обычно осуществляется на физическом уровне, поэтому все указатели в коде сообщают физическое расположение данных, на которые они указывают. В современных ПК за преобразование физических адресов в виртуальное представление запущенных задач отвечает операционная система.

Преимущество прямого адресного доступа к физической памяти в системах без MMU заключается в более простой трансляции адресов при кодировании и отладке. С другой стороны, некоторые базовые функции, реализованные в любой современной ОС, такие как своп (process swapping) и динамическое изменение размера адресных пространств, во встраиваемых системах становятся громоздкими, а иногда и невозможными.

Встраиваемые системы требуют очень кропотливой работы с памятью. Программисты, которые привыкли разрабатывать код для персональных компьютеров, ожидают, что среда выполнения обеспечит определенный уровень защиты. Виртуальное адресное пространство не допускает перекрытия областей памяти, а ОС легко обнаруживает несанкционированный доступ к памяти и нарушения сегментации, поэтому оперативно завершает процесс и предотвращает компрометацию всей системы.

Во встраиваемых системах, особенно при разработке кода для «голого железа», границы каждого пула адресов необходимо проверять вручную. Случайное изменение нескольких битов в неправильной области памяти или даже простое обращение к другой области памяти может привести к фаталь-

ной и необратимой ошибке. Система может зависнуть или, в худшем случае, начать вести себя непредсказуемо. При работе с памятью во встраиваемых системах необходимо соблюдать требования безопасности, особенно при работе с жизненно важными устройствами. Выявить ошибки работы с памятью в уже готовой программе обычно намного труднее, чем заставить себя разрабатывать безопасный код и защищать систему от ошибок программиста.

Правильные приемы работы с памятью будут показаны в главе 5.

1.3. Флеш-память

Серверы и персональные компьютеры хранят исполняемые приложения и библиотеки на внешних запоминающих устройствах. Перед выполнением код приложений считывают, преобразуют, возможно, распаковывают и сохраняют в ОЗУ до начала выполнения.

Микропрограмма (прошивка, firmware) встраиваемого устройства представляет собой, как правило, единственный файл двоичного кода, содержащий все программные компоненты и сохраняемый во внутренней флеш-памяти микроконтроллера. Поскольку флеш-память напрямую отображается на фиксированный сегмент адресов в пространстве памяти, процессор может последовательно извлекать и выполнять из нее отдельные инструкции без промежуточных шагов. Этот механизм называется *выполнением на месте* (execute in place, XIP).

Все неизменяемые разделы прошивки не требуют загрузки в оперативную память и доступны через прямую адресацию в пространстве памяти. Сюда входят не только исполняемые инструкции, но и все переменные, помеченные компилятором как константы. С другой стороны, при использовании XIP требуется несколько дополнительных шагов при подготовке образа микропрограммы для сохранения во флеш-памяти, а компоновщик должен быть проинструктирован о различных отображаемых в памяти областях целевого микроконтроллера.

Внутренняя флеш-память, отображаемая в адресное пространство микроконтроллера, недоступна для произвольной записи. Изменение содержимого внутренней флеш-памяти возможно только с использованием блочного доступа из-за аппаратных особенностей устройств флеш-памяти. Чтобы изменить значение одного байта во флеш-памяти, весь блок, содержащий этот байт, необходимо стереть и записать заново. Механизм доступа к блочной флеш-памяти для записи известен как *программирование в приложении* (in-application programming, IAP). Некоторые реализации файловой системы абстрагируют операции записи на блочном флеш-устройстве, создавая временную копию блока, в котором выполняется операция записи.

При выборе компонентов для устройства на базе микроконтроллера очень важно, чтобы размер флеш-памяти соответствовал пространству, занимаемому микропрограммой. Флеш-память остается одним из самых дорогих компонентов микроконтроллера, поэтому для крупномасштабного производства стараются выбирать чипы с памятью минимального размера, в ко-

торую помещается прошивка. Разработка программного обеспечения с жесткими ограничениями на размер машинного кода в настоящее время редко встречается в других областях, но при попытке втиснуть несколько функций в небольшое хранилище без этого не обойтись. Для некоторых архитектур существуют оптимизирующие компиляторы, способствующие уменьшению размера кода при сборке и компоновке встроенного ПО.

Доступ к дополнительной энергонезависимой памяти, находящейся за пределами микросхемы микроконтроллера, обычно можно получить с помощью специальных интерфейсов, таких как *последовательный периферийный интерфейс* (serial peripheral interface, SPI). Технология изготовления внутренней и внешней флеш-памяти различается. Хотя внешняя флеш-память обычно более плотная и менее дорогая, она не допускает прямого отображения памяти в физическом адресном пространстве, что делает ее непригодной для хранения образов микропрограмм. Это связано с тем, что невозможно последовательно выполнить код, извлекающий инструкции, если не использовать механизм загрузки исполняемых символов в ОЗУ – доступ для чтения на устройствах такого типа выполняется по одному блоку за раз. С другой стороны, доступ для записи может быть быстрее по сравнению с IAP, что делает внешние устройства энергонезависимой памяти идеальным местом для длительного хранения данных, которые получаются во время выполнения микропрограмм.

1.4. Универсальный ввод/вывод (GPIO)

Самая главная функция, которую можно реализовать с помощью любого микроконтроллера и благодаря которой они приобрели такую популярность, – это возможность управлять сигналами на заданных выводах интегральной схемы. Микроконтроллер может управлять *цифровым выходом*, устанавливая на нем логический уровень 0 или 1. Аналогичным образом вывод микросхемы можно использовать для считывания логического уровня, когда он настроен как *цифровой вход*. Программа считывает цифровое значение 1, когда приложенное к нему напряжение превышает определенный порог, и 0 в остальных случаях.

1.4.1. АЦП и ЦАП

Некоторые чипы имеют встроенные модули *аналого-цифрового преобразователя* (АЦП), которые способны определять напряжение, подаваемое на вывод, и делать его *выборку* (получать двоичное значение измеряемой величины). АЦП часто используют для измерения мгновенных значений переменного напряжения, вырабатываемого каким-либо внешним устройством. Встроенное программное обеспечение считывает напряжение на *аналоговом входе* микроконтроллера с точностью, зависящей от разрядности АЦП.

Модуль *цифроаналогового преобразователя* (ЦАП) выполняет противоположную работу, преобразуя значение в регистре микроконтроллера в соответствующее напряжение на *аналоговом выходе* чипа.

1.4.2. Таймеры и ШИМ

Микроконтроллеры могут использовать различные способы измерения времени. Обычно они имеют как минимум один интерфейс, основанный на таймере обратного отсчета, который может запускать прерывание и автоматически сбрасываться по истечении заданного периода времени.

Выводы GPIO, сконфигурированные как линии выхода, могут быть запрограммированы для вывода прямоугольного периодического сигнала с предварительно настроенной частотой и длительностью рабочего цикла. Такой способ управления сигналом называется *широтно-импульсной модуляцией* (ШИМ) и имеет несколько применений: от управления выходными периферийными исполнительными устройствами до регулировки яркости светодиодов или даже воспроизведения звукового сигнала через динамик.

Более подробная информация о GPIO, таймерах прерываний и сторожевых таймерах будет рассмотрена в главе 6.

1.5. Интерфейсы и периферийные устройства

Для связи с периферийными устройствами и другими микроконтроллерами в мире встраиваемых систем хорошо зарекомендовали себя несколько стандартов. Некоторые из внешних выводов микроконтроллера можно запрограммировать для связи с внешними периферийными устройствами с использованием определенных протоколов. Вот несколько общих интерфейсов, доступных на большинстве архитектур:

- *асинхронная последовательная связь* на основе UART;
- *шина последовательного периферийного интерфейса* (SPI);
- *шина связи между интегральными схемами* (inter-integrated circuit, I^2C);
- *универсальная последовательная шина* (universal serial bus, USB).

Рассмотрим каждую из них подробно.

1.5.1. Асинхронная последовательная связь на основе UART

Асинхронная связь обеспечивается *универсальным асинхронным приемником-передатчиком* (universal asynchronous receiver-transmitter, UART). Интерфейсы такого типа, широко известные как *последовательные порты*, называются асинхронными, потому что им не нужно использовать общий тактовый

сигнал для синхронизации отправителя и получателя, а вместо этого они работают с предопределенными тактовыми сигналами, которые могут быть согласованы во время обмена данными. Микроконтроллеры могут содержать несколько UART, которые подключаются к определенным выводам при настройке (или подключены постоянно). Асинхронная связь обеспечивается UART как полнодуплексный канал по двум независимым линиям, соединяющим вывод RX каждой конечной точки с выводом TX на противоположной стороне.

Для успешного обмена данными системы на двух конечных точках должны настроить UART, используя одни и те же параметры. К ним относятся кадрирование байтов на линии и частота кадров данных. Все параметры должны быть известны обеим конечным точкам заранее, чтобы правильно установить канал связи. Несмотря на то что последовательная связь на основе UART проще, чем другие типы связи, она по-прежнему широко используется в электронных устройствах, особенно в качестве интерфейса для модемов и приемников GPS. Кроме того, используя последовательные преобразователи TTL-USB, можно легко подключить UART к консоли на хост-компьютере, что часто удобно для обмена данными с устройством (например, при отладке).

1.5.2. SPI

Другим классическим протоколом на основе последовательного канала является SPI. Эта технология, разработанная в конце 1980-х годов, была разработана, чтобы заменить асинхронную последовательную связь с периферийными устройствами путем внесения нескольких улучшений:

- отдельная линия тактовых импульсов для синхронизации конечных точек;
- протокол типа «ведущий–ведомый»;
- связь «один ко многим» по одной трехпроводной шине.

Ведущее устройство, обычно микроконтроллер, делит шину с одним или несколькими ведомыми устройствами. Для запуска связи используется отдельный сигнал *выбора ведомого устройства* (slave select, SS), предназначенный для адресации каждого ведомого устройства, подключенного к шине. Шина использует две независимые линии для передачи данных, по одной в каждом направлении, и общую тактовую линию, которая синхронизирует два конца линии связи. Благодаря тактовым импульсам, которые генерирует ведущее устройство, передача данных более надежна, что позволяет достичь более высоких скоростей обмена данными, чем в обычном UART. Одним из секретов неизменной популярности SPI в нескольких поколениях микроконтроллеров является низкая сложность, необходимая для разработки ведомых устройств, которые могут быть очень простыми и состоять буквально из одного сдвигового регистра. SPI обычно используется в сенсорных устройствах, ЖК-дисплеях, контроллерах флеш-памяти и сетевых интерфейсах.

1.5.3. I²C

Протокол I²C немного сложнее, потому что он разработан с другой целью: соединение нескольких микроконтроллеров, а также нескольких ведомых устройств на одной и той же двухпроводной шине. По двум линиям шины передаются *последовательные тактовые импульсы* (serial clock, SCL) и *последовательные данные* (serial data, SDA). В отличие от SPI или UART, шина I²C является полудуплексной, поскольку два направления потока данных используют поочередно одну и ту же линию. Благодаря 7-битному механизму адресации ведомых устройств, включенному в протокол, он не требует дополнительных сигналов, предназначенных для выбора ведомых устройств. Допускается использование нескольких ведущих устройств на одной линии, при условии что все они следуют логике арбитража в случае конфликтов на шине.

1.5.4. USB

Протокол USB, изначально предназначенный для замены UART и включающий в себя множество протоколов в одном аппаратном разъеме, очень популярен в персональных компьютерах, портативных устройствах и огромном количестве периферийных устройств.

Этот протокол работает в режиме «хост–устройство». Хост-контроллер предоставляет услуги, которые может использовать подключаемое устройство. USB-трансиверы, присутствующие во многих микроконтроллерах, иногда могут работать в обоих режимах. Реализуя верхний уровень стандартов USB, микроконтроллер может эмулировать различные типы устройств, такие как последовательные порты, запоминающие устройства и двухточечные интерфейсы Ethernet. Таким образом получаются различные USB-устройства на базе микроконтроллера, которые можно подключать к хосту.

Если трансивер поддерживает режим хоста, встраиваемая система может работать как USB-хост и к ней можно подключать внешние устройства. В этом случае во встраиваемой системе должны быть реализованы драйверы устройств и приложения для доступа к функциям, предоставляемым устройством.

Когда оба режима реализованы на одном USB-контроллере, трансивер работает в режиме «на ходу» (on-the-go, OTG), а выбор и настройка нужного режима могут быть выполнены во время работы.

Более подробное введение в некоторые из наиболее распространенных протоколов, используемых для связи с периферийными устройствами и другими системами, представлены в главе 7.

1.6. Подключенные системы

Все большее количество встраиваемых устройств, предназначенных для различного применения, теперь способны устанавливать сетевую связь со

своими одноранговыми узлами в ближайшем окружении или со шлюзами, направляющими их трафик в более широкую сеть или интернет. Термин *интернет вещей* (Internet of Things, IoT) принято использовать для описания сетей, в которых эти встраиваемые устройства могут обмениваться данными с использованием интернет-протоколов.

Это означает, что к устройствам IoT можно обращаться в сети так же, как к более сложным системам, таким как ПК или мобильные устройства, и, что наиболее важно, они используют для обмена данными стандартные протоколы транспортного уровня, типичные для интернет-коммуникаций. TCP/IP – это набор протоколов, стандартизированных организацией IETF; он представляет собой детальное описание инфраструктуры для интернета и других автономных локальных сетей.

Интернет-протокол (internet protocol, IP) обеспечивает сетевое подключение, но при условии, что базовый канал обеспечивает связь на основе пакетов и механизмы для управления и регулирования доступа к физической среде. Многие сетевые интерфейсы удовлетворяют этим требованиям. Семейства альтернативных протоколов, несовместимых с TCP/IP, все еще используются в некоторых распределенных встраиваемых системах, но явным преимуществом использования стандарта TCP/IP во встраиваемом устройстве является то, что в случае связи с невстраиваемыми системами нет необходимости в механизме трансляции для отправки кадров данных за пределы локальной сети.

Помимо протоколов связи, которые широко используются в обычных системах, таких как Ethernet или беспроводная локальная сеть, встраиваемые системы используют технологии, специально разработанные для требований, предъявляемых IoT. Были разработаны и введены в действие новые стандарты, обеспечивающие эффективную связь для устройств, обладающих ограниченными аппаратными ресурсами и действующих в условиях строгого энергосбережения.

В последнее время были разработаны новые технологии связи, ориентированные на более низкую скорость передачи данных и высокое энергосбережение. Эти протоколы предназначены для обеспечения узкополосной связи на больших расстояниях. Кадр таких протоколов слишком мал для размещения IP-пакетов, поэтому эти технологии в основном используются для передачи небольших полезных данных, таких как периодические показания датчиков или параметры конфигурации устройства, если доступен двунаправленный канал, и им требуется шлюз-транслятор для преобразования пакетов, потому что на определенном этапе данные могут путешествовать по интернету.

Но взаимодействие с облачными сервисами в большинстве случаев требует подключения всех узлов к интернету и внедрения тех же протоколов связи, которые используются серверами и ИТ-инфраструктурой. Реализовать протокол TCP/IP на встраиваемом устройстве не всегда легко. Несмотря на то что существует несколько доступных открытых библиотек, системный код TCP/IP является сложным, большим по размеру и часто имеет требования к памяти, которых трудно придерживаться.

Вышесказанное относится и к библиотеке Secure Socket Layer (SSL)/Transport Layer Security (TLS), которая обеспечивает конфиденциальность и аутен-

тификацию между двумя конечными точками связи. Выбор правильного микроконтроллера для проекта в данном случае имеет ключевое значение. Если встраиваемая система должна быть подключена к интернету и поддерживать защищенную связь через сокеты, то требования к флеш-памяти и ОЗУ следует тщательно продумать на этапе проектирования, чтобы обеспечить возможность использования сторонних библиотек.

1.6.1. Особенности распределенных систем

Разработка распределенных встраиваемых систем, особенно тех, которые основаны на технологиях беспроводной связи, добавляет ряд интересных проблем.

Некоторые из этих проблем связаны со следующими аспектами:

- выбор подходящих технологий и протоколов;
- ограничения на скорость передачи данных, размер пакета и доступ к среде;
- доступность узлов;
- единичные точки отказа в топологии;
- настройка маршрутов;
- аутентификация задействованных хостов;
- конфиденциальность коммуникаций в среде;
- влияние буферизации на скорость сети, задержку и использование ОЗУ;
- сложность реализации стеков протоколов.

В главе 9 анализируются некоторые технологии канального уровня, применяемые во встраиваемых системах для обеспечения удаленной связи, где поддержка протокола TCP/IP интегрирована в структуру распределенных систем, интегрированных со службами интернета вещей.

1.7. Механизмы изоляции

Некоторые современные микроконтроллеры поддерживают механизм изоляции между доверенным и недоверенным программным обеспечением, работающим в устройстве. Этот механизм основан на расширении процессора, доступном только в некоторых архитектурах. Это расширение обычно основано на своего рода физическом разделении внутри самого процессора между двумя режимами выполнения. Весь код, работающий из недоверенной области в системе, будет иметь ограниченный доступ к ОЗУ, встроенным модулям на чипе и периферийным устройствам, которые должны быть заранее динамически настроены доверенным ПО.

Программное обеспечение, работающее из доверенной области, также может предоставлять возможности, недоступные напрямую из ненадежной области, с помощью вызовов специальных функций, пересекающих границу между областями.

В главе 11 рассмотрена технология, лежащая в основе *доверенной среды выполнения* (trust execution environment, TEE), а также программные компоненты, используемые в реальных встроенных системах для формирования безопасной среды выполнения недоверенных модулей и компонентов.

1.8. Базовая платформа

Если говорить о разработке микроконтроллеров для встраиваемых систем, то в настоящее время предпочтительной стратегией при проектировании встроенного процессорного ядра является использование архитектуры *вычислителя с сокращенным набором команд* (reduced instruction set computer, RISC). Производители интегральных схем рекомендуют придерживаться нескольких базовых архитектур в качестве рекомендаций по выбору состава логики для интеграции в микроконтроллер. Каждая архитектура отличается от других несколькими характеристиками реализации процессорного ядра и включает в себя одно или несколько семейств микропроцессоров, характеризуемых следующими параметрами:

- размер слова, используемый для регистров и адресов (8-, 16-, 32- или 64-битный);
- набор инструкций;
- набор конфигураций;
- порядок следования байтов;
- расширенные функции процессора (контроллер прерываний, FPU, MMU);
- стратегии кеширования;
- устройство конвейера.

Выбор базовой платформы для встраиваемой системы зависит от потребностей вашего проекта. Маломощные и менее функциональные процессоры, как правило, больше подходят для устройств с низким энергопотреблением, имеют меньший корпус микроконтроллера и дешевле. Системы более высокого класса поставляются с большим набором ресурсов, а некоторые из них имеют специальное оборудование для выполнения сложных вычислений (например, модуль с плавающей запятой или аппаратный модуль симметричного шифрования Advanced Encryption Standard (AES) для разгрузки процессорного ядра). Широко распространенные в прошлом 8- и 16-битные ядра постепенно уступают место 32-битным архитектурам, но некоторые успешные проекты остаются относительно популярными на некоторых нишевых рынках и среди радиолюбителей.

1.8.1. Базовая архитектура ARM

ARM является самым распространенным поставщиком базовых платформ на рынке встраиваемых систем: для встраиваемых приложений произведено

более 10 млрд микроконтроллеров на базе ARM. Одним из наиболее интересных проектов ядер в индустрии встраиваемых систем является семейство ARM Cortex-M, в модельный ряд которого входит множество вариантов ядер от экономичных и энергоэффективных до высокопроизводительных ядер, специально разработанных для мультимедийных микроконтроллеров. Несмотря на использование трех различных наборов инструкций (ARMv6, ARMv7 и ARMv8), все процессоры семейства Cortex-M используют один и тот же программный интерфейс, что повышает переносимость программ между микроконтроллерами одного семейства.

Большинство примеров в этой книге основаны на семействе процессоров Cortex-M. Хотя большинство рассматриваемых концепций применимы и к другим семействам, такой выбор базовой платформы открывает возможность для наиболее полного анализа взаимодействия кода программы с аппаратной частью. В частности, в некоторых примерах в этой книге используются специальные инструкции по сборке из набора инструкций ARMv7, который реализован в некоторых ядрах Cortex-M.

1.8.2. Микропроцессор Cortex-M

Основные характеристики 32-битных ядер семейства Cortex-M следующие:

- 16 регистров ЦП общего назначения;
- 16-битные инструкции для оптимизации плотности кода;
- встроенный *контроллер приоритетных векторных прерываний* (nested vector interrupt controller, NVIC) с 8...16 уровнями приоритета;
- архитектура ARMv6-M (M0, M0+), ARMv7-M (M3, M4, M7) или ARMv8-M (M23, M33);
- дополнительный *блок защиты памяти* (memory protection unit, MPU) с 8 регионами;
- дополнительный механизм изоляции TEE (ARM TrustZone-M).

Общее адресное пространство памяти составляет 4 ГБ. Начало внутренней оперативной памяти обычно отображается по фиксированному адресу `0x20000000`. Отображение внутренней флеш-памяти, а также других периферийных устройств зависит от производителя чипа. Но самые высокие адреса размером 512 МБ (от `0xE0000000` до `0xFFFFFFFF`) зарезервированы для *блока управления системой* (system control block, SCB), представляющего собой набор параметров конфигурации и диагностики, к которым программное обеспечение может получить доступ в любое время для прямого взаимодействия с ядром.

Синхронная связь с периферийными устройствами и другими аппаратными компонентами может быть запущена через линии прерывания. Процессор может принимать и распознавать несколько различных цифровых входных сигналов и быстро реагировать на них, прерывая выполнение основной программы и временно переходя к определенному месту в памяти. Cortex-M поддерживает до 240 линий прерывания на высокопроизводительных ядрах семейства.

Вектор прерывания, расположенный в начале образа программы во флеш-памяти, содержит адреса подпрограмм прерывания, которые будут автоматически выполняться при определенных событиях. Благодаря NVIC линиям прерывания можно назначать приоритеты, так что когда прерывание с более высоким приоритетом происходит во время выполнения процедуры для более низкого прерывания, текущая процедура прерывания временно приостанавливается, чтобы позволить обслужить линию прерывания с более высоким приоритетом. Это обеспечивает минимальную задержку прерывания для критически важных событий системы.

Программное обеспечение микроконтроллера может работать в двух режимах: непривилегированном или привилегированном. ЦП имеет встроенную поддержку разделения привилегий между системным и прикладным программным обеспечением и даже предоставляет два разных регистра для двух независимых указателей стека. В главе 10 более подробно рассмотрено, как правильно реализовать разделение привилегий, а также как обеспечить разделение памяти при выполнении ненадежного кода. Технология разделения памяти, например, используется для сокрытия секретов, таких как закрытые ключи, от прямого доступа из внешнего мира. В главе 11 показано, как правильно реализовать разделение привилегий, а также как обеспечить разделение памяти внутри ОС при запуске кода приложения с другим уровнем доверия.

Ядро Cortex-M присутствует во многих микроконтроллерах от разных поставщиков микросхем. Программные инструменты одинаковы для всех платформ, но каждый вариант микроконтроллера имеет свою конфигурацию, которую необходимо учитывать при разработке системы. Доступны так называемые библиотеки конвергенции, помогающие абстрагироваться от деталей реализации конкретного производителя и улучшить переносимость между разными моделями и брендами. Производители предоставляют обучающие наборы (оценочные платы) и всю документацию, необходимую для начала работы и оценки на этапе проектирования, а также полезную для разработки прототипов на более позднем этапе. Некоторые из оценочных плат оснащены датчиками, мультимедийной электроникой или другими периферийными устройствами, расширяющими функциональные возможности микроконтроллера. Отдельные платы даже поставляются в комплекте с предварительно сконфигурированными сторонними библиотеками промежуточного программного обеспечения, включая коммуникационные стеки TCP/IP, библиотеки TLS и криптографии, простые файловые системы и другие вспомогательные компоненты, а также модули, которые можно быстро и легко добавить в программный проект.

1.9. Заключение

При составлении требований к встроенному программному обеспечению прежде всего вы должны иметь хорошее представление об аппаратной платформе и ее компонентах. В этой главе сделан краткий обзор архитектур

современных микроконтроллеров, показаны некоторые особенности встраиваемых устройств и сделали акцент на том, что разработчикам следует эффективно переосмыслить свой подход к удовлетворению требований и решению задач, в то же время принимая во внимание особенности и ограничения целевой платформы.

В следующей главе будут проанализированы инструменты и процедуры, обычно используемые при разработке встраиваемых систем, включая наборы инструментов командной строки и *интегрированные среды разработки* (integrated development environment, IDE). В ней будет показано, как организовать рабочий процесс и как эффективно предотвращать, находить и исправлять ошибки.