FreeRTOS — операционная система для микроконтроллеров

Это первая статья из цикла, посвященного операционной системе для микроконтроллеров FreeRTOS. Статья познакомит читателя с задачами, которые решают операционные системы (ОС) для микроконтроллеров (МК). Освещены вопросы целесообразности применения, преимущества и недостатки, присущие ОС для МК. Представлены возможности FreeRTOS, описаны ее особенности, а также приведена структура дистрибутива FreeRTOS с кратким описанием назначения входящих в него файлов и директорий.

Андрей КУРНИЦ
kurnits@stim.by

Что такое ОС для МК?

В нынешний век высоких технологий все профессионалы знакомы с термином «операционная система» (ОС). История ОС начинается с 1960-х годов. Первые ОС предназначались для больших ЭВМ, а впоследствии — и для персональных компьютеров. Назначением ОС стало заполнение ниши между низкоуровневой аппаратурой и высокоуровневыми программами, они предоставляют программам удобный интерфейс обращения к системным ресурсам, будь то процессорное время, память или устройства ввода/вывода. С тех пор технологии шагнули далеко вперед: целую вычислительную систему (процессор, память, устройства ввода/вывода) разместили на одном кристалле — появились микроконтроллеры (МК). В соответствии с древним изречением «Природа не любит пустоты» удачная концепция ОС не могла не быть применена и к микроконтроллерам. В настоящее время создано и развивается множество ОС, ориентированных на выполнение на МК [1, 6]. Однако МК как платформа для выполнения ОС имеет существенные отличия от современных компьютеров.

Прежде всего, МК работает в режиме реального времени, то есть время реакции микроконтроллерного устройства на внешнее событие должно быть строго меньше заданной величины и должно быть сопоставимо со скоростью протекания внешних процессов. Типичный пример: время реакции на срабатывание датчика давления в промышленной установке должно быть не более 5 мс, иначе произойдет авария. Таким образом, ОС для МК — это операционная система реального времени (ОСРВ). К ОСРВ предъявляются жесткие временные требования в отличие от распространенных ОС общего назначения (Windows, UNIX-подобные и др.).

Во-вторых, микроконтроллер, по сути, это однокристальный компьютер с сильно ограниченными аппаратными ресурсами, хотя диапазон выпускаемых МК по производительности и объемам памяти очень широк. Встречаются как «карлики», например 8-разрядный ATtiny10 с 6 выводами, 32 байт ОЗУ, 1 кбайт ПЗУ и производительностью 12×10^6 операций в секунду (12 MIPS), так и «гиганты», например 32-разрядный TMS320C28346 с 256 выводами, 512 кбайт ОЗУ и производительностью 600×10^6 операций с плавающей точкой в секунду (600 MFLOPS). Тем не менее все МК имеют существенные аппаратные ограничения, что предъявляет специфические требования к ОСРВ для МК.

Их основные особенности:

1. Низкая производительность.
2. Малый объем ОЗУ и ПЗУ.
3. Отсутствие блока управления памятью (Memory management unit, MMU), используемого большинством современных ОС, например Windows и UNIX-подобными.
4. Отсутствие аппаратных средств поддержки многозадачности (например, средств быстрого переключения контекста).

В-третьих, микроконтроллер сам по себе предназначен для выполнения низкоуровневых задач, будь то опрос состояния кнопок, передача команды по I^2C-интерфейсу или включение обмотки электромотора. Программа для МК, как правило, обращается к периферии напрямую, программист имеет полный контроль над аппаратной частью, нет необходимости в посредниках между аппаратурой и прикладной программой. Может показаться, что операционная система для МК вообще не нужна, что любую программу можно написать и без ОС. На самом деле так оно и есть! Но есть один нюанс: микроконтроллер редко используют только для опроса состояния кнопок, только для передачи команды по I^2C-интерфейсу или только для включения обмотки электромотора. Гораздо чаще из МК пытаются «выжать» все, на что он способен, а в микроконтроллерное устройство заложить все возможные функции. Количество функций-задач, одновременно выполняемых МК, может доходить до нескольких десятков. И вот тут-то и начинаются проблемы.

Как организовать мультизадачность и поочередное выполнение каждой задачи? Как обеспечить запуск задачи через строго определенные интервалы времени? Как передать информацию от одной задачи другой? Обычно эти вопросы не встают перед программистом в начале разработки, а возникают где-то в середине, когда он запрограммировал большинство функций будущего устройства, используя изобретенные им самим средства «многозадачности». И тогда заказчик «просит» добавить еще несколько «маленьких» деталей в работу устройства, например сбор статистики работы и запись ее на какой-нибудь носитель… Знакомая ситуация?

Преимущества ОСРВ для МК

Здесь на помощь приходит ОСРВ. Рассмотрим преимущества, которые получил бы наш гипотетический программист, заложив в основу программного обеспечения своего устройства ОСРВ:

1. Многозадачность. ОСРВ предоставляет программисту готовый, отлаженный механизм многозадачности. Теперь каждую отдельную задачу можно программировать по отдельности так, как будто остальных задач не существует. Например, можно разработать архитектуру программы, то есть разбить ее на отдельные задачи и распределить их между командой программистов. Программисту не нужно заботиться о переключении между задачами: за него это сделает ОСРВ в соответствии с алгоритмом работы планировщика.

2. Временная база. Необходимо отмерять интервалы времени? Пожалуйста, любая ОСРВ имеет удобный программный интерфейс для отсчета интервалов времени и выполнения каких-либо действий в определенные моменты времени.
3. Обмен данными между задачами. Необходимо передать информацию от одной задачи к другой без потерь? Используйте очередь, которая гарантирует, что сообщения дойдут до адресата в том объеме и в той последовательности, в которой были отправлены.
4. Синхронизация. Разные задачи обращаются к одному и тому же аппаратному ресурсу? Используйте мьютексы или критические секции для организации совместного доступа к ресурсам. Необходимо выполнять задачи в строгой последовательности или по наступлении определенного события? Используйте семафоры или сигналы для синхронизации задач.

Кроме этого, одна и та же ОСРВ для МК может выполняться на множестве архитектур микроконтроллеров. Какое преимущество это дает? Часто приходится решать задачу не как разработать устройство с требуемой функциональностью, а как перенести имеющуюся разработку на новую аппаратную платформу. Это может быть связано с завершением производства того или иного МК (окончание Life cycle), с появлением на рынке МК, включающего в состав некоторые блоки, которые ранее были реализованы как отдельные микросхемы, и т. д. В случае использования ОСРВ затраты времени и сил на переход на другую платформу будут заметно ниже за счет того, что часть кода, связанная с работой ОСРВ, останется без изменений. Изменения коснутся только кода, отвечающего за обращение к встроенной периферии (таймеры, АЦП, последовательный приемопередатчик и т. д.).

Однако за все надо платить. Использование ОСРВ приводит к определенным накладным расходам. Это:

1. Дополнительный расход памяти программ для хранения ядра ОСРВ.
2. Дополнительный расход памяти данных для хранения стека каждой задачи, семафоров, очередей, мьютексов и других объектов ядра операционной системы.
3. Дополнительные затраты времени процессора на переключение между задачами.

Когда можно обойтись без ОСРВ для МК?

Конечно же, если вам необходимо разработать простейшее устройство, например индикатор температуры, который будет выполнять две функции: опрос датчика и индикацию на 7-сегментный светодиодный индикатор, то применение ОСРВ в таком устройстве будет непозволительным расточительством и приведет, в конечном счете, к удорожанию устройства.

В этом случае можно применить один из «традиционных» для МК способов организации многозадачности. Прежде всего, это циклический алгоритм (round robin) [3], когда программист помещает все задачи в тело бесконечного цикла. При этом на подпрограммы, реализующие задачи, накладываются следующие ограничения:

1. Подпрограмма не должна содержать циклов ожидания наступления какого-либо события, например прерывания.
2. Подпрограмма должна лишь проверять, наступило ли событие, и как можно быстрее передавать управление следующей подпрограмме, то есть завершать свое выполнение.
3. Подпрограмма должна сохранять свое текущее состояние (например, в статической или глобальной переменной) до следующего вызова.

Таким образом, каждая задача представляется в виде конечного автомата. Дальнейшее развитие эта идея получила в SWITCH-технологии программирования [4, 5].

Резюме

Итак, применение ОСРВ оправдано в случае использования достаточно мощного МК при разработке сложного устройства с множеством функций, например:

1. Опрос датчиков.
2. Интерфейс с пользователем (простейшие клавиатура и дисплей).
3. Выдача управляющего воздействия.
4. Обмен информацией по нескольким внутрисхемным шинам I^2C, SPI, 1Wire и др.
5. Обмен информацией с внешними устройствами по интерфейсам RS-232C, RS-485, CAN, Ethernet, USB и др.
6. Реализация высокоуровневых протоколов, например TCP/IP, ProfiBus, ModBus, CANOpen и др.
7. Поддержка Flash-накопителей и, соответственно, файловой системы.

Обзор FreeRTOS

FreeRTOS — это многозадачная, мультиплатформенная, бесплатная операционная система жесткого реального времени с открытым исходным кодом. FreeRTOS была разработана компанией Real Time Engineers Ltd. специально для встраиваемых систем. На момент написания статьи (версия FreeRTOS 6.1.0) ОС официально поддерживает 23 архитектуры и 57 платформ (в подавляющем большинстве — микроконтроллеры) [7]. В течение 2008 и 2009 годов произошло более 77 500 загрузок FreeRTOS с официального сайта, что делает ее одной из самых популярных ОСРВ на сегодня. Бóльшая часть кода FreeRTOS написана на языке Си, ассемблерные вставки минимального объема применяются лишь там, где невозможно применить Си из-за специфики конкретной аппаратной платформы.

Существуют так называемые официально поддерживаемые аппаратные платформы — официальные порты и неофициальные, которые поставляются «как есть» и не поддерживаются напрямую. Кроме того, для одного и того же порта могут поддерживаться несколько средств разработки. Список официальных портов и средств разработки приведен в таблице 1.

Основные характеристики FreeRTOS:

1. Планировщик FreeRTOS поддерживает три типа многозадачности:
   – вытесняющую;
   – кооперативную;
   – гибридную.
2. Размер ядра FreeRTOS составляет всего 4–9 кбайт, в зависимости от типа платформы и настроек ядра.
3. FreeRTOS написана на языке Си (исходный код ядра представлен в виде всего лишь четырех Си-файлов).
4. Поддерживает задачи (tasks) и сопрограммы (co-routines). Сопрограммы специально созданы для МК с малым объемом ОЗУ.
5. Богатые возможности трассировки.
6. Возможность отслеживать факт переполнения стека.
7. Нет программных ограничений на количество одновременно выполняемых задач.
8. Нет программных ограничений на количество приоритетов задач.
9. Нет ограничений в использовании приоритетов: нескольким задачам может быть назначен одинаковый приоритет.
10. Развитые средства синхронизации «задача – задача» и «задача – прерывание»:
    – очереди;
    – двоичные семафоры;
    – счетные семафоры;
    – рекурсивные семафоры;
    – мьютексы.
11. Мьютексы с наследованием приоритета.
12. Поддержка модуля защиты памяти (Memory protection unit, MPU) в процессорах Cortex-M3.
13. Поставляется с отлаженными примерами проектов для каждого порта и для каждой среды разработки.
14. FreeRTOS полностью бесплатна, модифицированная лицензия GPL позволяет использовать FreeRTOS в проектах без раскрытия исходных кодов.
15. Документация в виде отдельного документа платная, но на официальном сайте [7] в режиме on-line доступно исчерпывающее техническое описание на английском языке.

Работа планировщика FreeRTOS в режиме вытесняющей многозадачности имеет много общего с алгоритмом переключения потоков в современных ОС общего назначения. Вытесняющая многозадачность предполагает, что любая выполняющаяся задача с низким приоритетом прерывается готовой к выполнению задачей с более высоким приоритетом. Как только высокоприоритетная

Таблица 1. Список официальных портов FreeRTOS и средств разработки

Производитель	Поддерживаемые семейства (ядра)	Поддерживаемые средства разработки
Altera	Nios II	Nios II IDE, GCC
Atmel	SAM3 (Cortex-M3)	IAR, GCC, Keil, Rowley CrossWorks
	SAM7 (ARM7)	
	SAM9 (ARM9)	
	AT91	
	AVR32 UC3	
Cortus	APS3	Cortus IDE, GCC
Energy Micro	EFM32 (Cortex-M3)	IAR
Freescale	Coldfire V2	Codewarrior, GCC, Eclipse
	Coldfire V1	
	другие Coldfire	
	HCS12	
	PPC405 & PPC440	
Fujitsu	32 бит (например, MB91460)	Softune
	16 бит (например, MB96340 16FX)	
Luminary Micro / Texas Instruments	Все МК Stellaris на основе ядра Cortex-M3	Keil, IAR, Code Red, CodeSourcery GCC, Rowley CrossWorks
Microchip	PIC32	MPLAB C32, MPLAB C30, MPLAB C18, wizC
	PIC24	
	dsPIC	
	PIC18	
NEC	V850 (32 бит)	IAR
	78K0R (16 бит)	
NXP	LPC1700 (Cortex-M3)	GCC, Rowley CrossWorks, IAR, Keil, Red Suite, Eclipse
	LPC2000 (ARM7)	
Renesas	RX600/RX62N	GCC, HEW (High Performance Embedded Workbench), IAR
	SuperH	
	H8/S	
Silicon Labs (бывший Cygnal)	Сверхбыстрые i8051 совместимые МК	SDCC
ST	STM32 (Cortex-M3)	IAR, GCC, Keil, Rowley CrossWorks
	STR7 (ARM7)	
	STR9 (ARM9)	
Texas Instruments	MSP430	Rowley CrossWorks, IAR, GCC
Xilinx	PPC405, выполняющийся на Virtex4 FPGA	GCC
	PPC440, выполняющийся на Virtex5 FPGA	
	Microblaze	
i8086	Любой x86 совместимый процессор в реальном режиме (Real mode)	Open Watcom, Borland, Paradigm
	Win32 симулятор	Visual Studio

задача выполнила свои действия, она завершает свою работу или переходит в состояние ожидания, и управление снова получает задача с низким приоритетом. Переключение между задачами осуществляется через равные кванты времени работы планировщика, то есть высокоприоритетная задача, как только она стала готова к выполнению, ожидает окончания текущего кванта, после чего управление получает планировщик, который передает управление высокоприоритетной задаче.

Таким образом, время реакции FreeRTOS на внешние события в режиме вытесняющей многозадачности — не больше одного кванта времени планировщика, который можно задавать в настройках. По умолчанию он равен 1 мс.

Если готовы к выполнению несколько задач с одинаковым приоритетом, то в таком случае планировщик выделяет каждой из них по одному кванту времени, по истечении которого управление получает следующая задача с таким же приоритетом, и так далее по кругу.

Кооперативная многозадачность отличается от вытесняющей тем, что планировщик самостоятельно не может прервать выполнение текущей задачи, даже если появилась готовая к выполнению задача с более высоким приоритетом. Каждая задача должна самостоятельно передать управление планировщику. Таким образом, высокоприоритетная задача будет ожидать, пока низкоприоритетная завершит свою работу и отдаст управление планировщику. Время реакции системы на внешнее событие становится неопределенным и зависит от того, как долго текущая задача будет выполняться до передачи управления. Кооперативная многозадачность применялась в семействе ОС Windows 3.x.

Вытесняющая и кооперативная концепции многозадачности объединяются вместе в гибридной многозадачности, когда вызов планировщика происходит каждый квант времени, но, в отличие от вытесняющей многозадачности, программист имеет возможность сделать это принудительно в теле задачи. Особенно полезен этот режим, когда необходимо сократить время реакции системы на прерывание. Допустим, в текущий момент выполняется низкоприоритетная задача, а высокоприоритетная ожидает наступления некоторого прерывания. Далее происходит прерывание, но по окончании работы обработчика прерываний выполнение возвращается к текущей низкоприоритетной задаче, а высокоприоритетная ожидает, пока закончится текущий квант времени. Однако если после выполнения обработчика прерывания передать управление планировщику, то он передаст управление высокоприоритетной задаче, что позволяет значительно сократить время реакции системы на прерывание, связанное с внешним событием.

Для оценки затрат времени, вносимых планировщиком FreeRTOS, можно сравнить два распространенных семейства МК: PIC и AVR. Затраты времени складываются из времени переключения контекста, когда планировщик определяет задачу для выполнения в следующем кванте времени, и времени сохранения/восстановления контекста, когда текущее состояние задачи (регистры процессора) сохраняется/извлекается из стека (таблица 2). Замеры приведены для компиляторов MPLAB PIC18 compiler и WinAVR соответственно, уровень оптимизации — максимальный по скорости.

Для того чтобы оценить объем ОЗУ, требуемый для работы FreeRTOS, достаточно привести расчет расхода ОЗУ для следующей конфигурации:

1. Порт для процессоров ARM7, среда разработки IAR STR71x.
2. Полная оптимизация (Full optimization) включена.
3. Все компоненты FreeRTOS, кроме сопрограмм и трассировки, включены.
4. 4 приоритета задач.

Объемы расхода ОЗУ для такой конфигурации приведены в таблице 3.

Расход ОЗУ будет существенно ниже при работе FreeRTOS на 8- и 16-битных архитектурах.

Таблица 2. Расход времени на переключение между задачами

Микроконтроллер	Частота тактирования, МГц	Время переключения контекста, мкс	Время сохранения / восстановления контекста, мкс
ATMega323	8	41,8	~8
PIC18F452	20	66,2	~10

Таблица 3. Объемы ОЗУ, требуемые для работы FreeRTOS

Объект	Расход ОЗУ, байт
Планировщик (sheduler)	236
Каждая дополнительная очередь (queue)	76 + память для хранения всех элементов очереди (зависит от размера очереди)
Каждая дополнительная задача (task)	64 + стек задачи

Кроме самой FreeRTOS, существуют также ее коммерческие версии: SafeRTOS и OpenRTOS. SafeRTOS — это ОСРВ, соответствующая уровню функциональной безопасности SIL3, имеющая такую же функциональную модель, что и FreeRTOS, и ориентированная на применение в системах с высокими требованиями к безопасности, например в медицинской и аэрокосмической отраслях. OpenRTOS отличается от FreeRTOS

лишь тем, что поставляется под коммерческой лицензией, с гарантией производителя и отменяет некоторые несущественные ограничения, присущие FreeRTOS. Подробно с особенностями SafeRTOS и OpenRTOS можно ознакомиться в [8].

Конечно, FreeRTOS — это далеко не единственный выбор для разработчика. В настоящее время существует множество других ОСРВ для МК, среди которых можно назвать uC/OS-II, µClinux, Salvo, jacOS и др. [6]. Однако обсуждение достоинств и недостатков этих ОС выходит за рамки данной статьи.

С чего начать?

Начать разработку микроконтроллерного устройства, работающего под управлением FreeRTOS, можно с загрузки ее последней версии по адресу [9]. Дистрибутив FreeRTOS доступен в виде обычного или самораспаковывающегося ZIP-архива. Дистрибутив содержит непосредственно код ядра (в виде нескольких заголовочных файлов и файлов с исходным кодом) и демонстрационные проекты (по одному проекту на каждую среду разработки для каждого порта). Далее следует распаковать архив в любое подходящее место на станции разработки.

Несмотря на достаточно большое количество файлов в архиве (5062 файла для версии 6.1.0), структура директорий на самом деле проста. Если планируется проектировать устройства на 2–3 архитектурах в 1–2 средах разработки, то бóльшая часть файлов, относящихся к демонстрационным проектам и различным средам разработки, не понадобится.

Подробная структура директорий приведена на рисунке.

Весь исходный код ядра находится в директории ***/Source***. Его составляют следующие файлы:

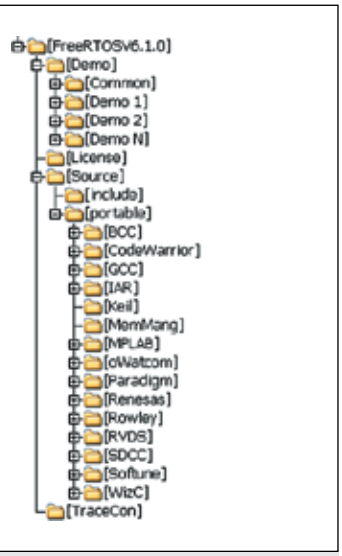

Рисунок. Структура директорий FreeRTOS после установки на станцию разработки

1. ***tasks.c*** — планировщик, реализация механизма задач.
2. ***queue.c*** — реализация очередей.
3. ***list.c*** — внутренние нужды планировщика, однако функции могут использоваться и в прикладных программах.
4. ***croutine.c*** — реализация сопрограмм (может отсутствовать в случае, если сопрограммы не используются).

Заголовочные файлы, которые находятся в директории ***Source/Include***:

1. ***tasks.h***, ***queue.h***, ***list.h***, ***croutine.h*** — заголовочные файлы соответственно для одноименных файлов с кодом.
2. ***FreeRTOS.h*** — содержит препроцессорные директивы для настройки компиляции.
3. ***mpu_wrappers.h*** — содержит переопределения функций программного интерфейса (API-функций) FreeRTOS для поддержки модуля защиты памяти (MPU).
4. ***portable.h*** — платформенно-зависимые настройки.
5. ***projdefs.h*** — некоторые системные определения.
6. ***semphr.h*** — определяет API-функции для работы с семафорами, которые реализованы на основе очередей.
7. ***StackMacros.h*** — содержит макросы для контроля переполнения стека.

Каждая аппаратная платформа требует небольшой части кода ядра, которая реализует взаимодействие FreeRTOS с этой платформой. Весь платформенно-зависимый код находится в поддиректории ***/Source/Portable***, где он систематизирован по средам разработки (IAR, GCC и т. д.) и аппаратным платформам (например, AtmelSAM7S64, MSP430F449). К примеру, поддиректория ***/Source/Portable/GCC/ATMega323*** содержит файлы port.c и portmacro.h, реализующие сохранение/восстановление контекста задачи, инициализацию таймера для создания временной базы, инициализацию стека каждой задачи и другие аппаратно-зависимые функции для микроконтроллеров семейства mega AVR и компилятора WinAVR (GCC).

Отдельно следует выделить поддиректорию ***/Source/Portable/MemMang***, в которой содержатся файлы ***heap_1.c***, ***heap_2.c***, ***heap_3.c***, реализующие 3 различных механизма выделения памяти для нужд FreeRTOS, которые будут подробно описаны позже.

В директории ***/Demo*** находятся готовые к компиляции и сборке демонстрационные проекты (***Demo 1***, ***Demo 2***, …, ***Demo N*** на рисунке). Общая часть кода для всех демонстрационных проектов выделена в поддиректорию ***/Demo/Common***.

Чтобы использовать FreeRTOS в своем проекте, необходимо включить в него файлы исходного кода ядра и сопутствующие заголовочные файлы. Нет необходимости модифицировать их или понимать их реализацию.

Например, если планируется использовать порт для микроконтроллеров MSP430 и GCC-компилятор, то для создания проекта «с нуля» понадобятся поддиректории ***/Source/Portable/GCC/MSP430F449*** и ***/Source/Portable/MemMang***. Все остальные поддиректории из директории ***/Source/Portable*** не нужны и могут быть удалены.

Если же планируется модифицировать существующий демонстрационный проект (что, собственно, и рекомендуется сделать в начале изучения FreeRTOS), то понадобятся также поддиректории ***/Demo/msp430_GCC*** и ***/Demo/Common***. Остальные поддиректории, находящиеся в ***/Demo***, не нужны и могут быть удалены.

При создании приложения рекомендуется использовать ***makefile*** (или файл проекта среды разработки) от соответствующего демонстрационного проекта как отправную точку. Целесообразно исключить из сборки (***build***) файлы из директории ***/Demo***, заменив их своими, а файлы из директории ***/Source*** оставить нетронутыми. Это гарантия того, что все исходные файлы ядра FreeRTOS будут включены в сборку и настройки компилятора останутся корректными.

Следует упомянуть также о заголовочном файле ***FreeRTOSConfig.h***, который находится в каждом демонстрационном проекте. ***FreeRTOSConfig.h*** содержит определения (***#define***), позволяющие произвести настройку ядра FreeRTOS:

1. Набор системных функций.
2. Использование сопрограмм.
3. Количество приоритетов задач и сопрограмм.
4. Размеры памяти (стека и кучи).
5. Тактовая частота МК.
6. Период работы планировщика — квант времени, выделяемый каждой задаче для выполнения, который обычно равен 1 мс.

Отключение некоторых системных функций и уменьшение количества приоритетов позволяет уменьшить расход памяти программ и данных.

В дистрибутив FreeRTOS включены также средства для конвертирования трассировочной информации, полученной от планировщика, в текстовую форму (директория ***/TraceCon***) и текст лицензии (директория ***/License***).

Выводы

С помощью первой статьи цикла читатель мог познакомиться с операционной системой для микроконтроллеров FreeRTOS. Показаны ее основные особенности. Описано содержимое дистрибутива FreeRTOS. Приведены основные шаги, с которых следует начинать разработку устройства, работающего под управлением FreeRTOS.

В следующих публикациях внимание будет уделено механизму многозадачности, а именно задачам и сопрограммам. Будет приведен образец работы планировщика на примере микроконтроллеров AVR фирмы Atmel и компилятора WinAVR (GCC).

Литература

1. Сорокин С. Как много ОСРВ хороших… // Современные технологии автоматизации. 1997. № 2.
2. Борисов-Смирнов А. Операционные системы реального времени для микроконтроллеров // Chip news. 2008. № 5.
3. Сорокин С. Системы реального времени // Современные технологии автоматизации. 1997. № 2.
4. http://ru.wikipedia.org/wiki/Switch-технология
5. Татарчевский В. Применение SWITCH- технологии при разработке прикладного программного обеспечения для микроконтроллеров // Компоненты и технологии. 2006. № 11.
6. http://ru.wikipedia.org/wiki/Список_операционных_систем
7. http://www.freertos.org
8. http://www.freertos.org/index.html? http://www.freertos.org/a00114.html
9. http://sourceforge.net/projects/freertos/files/FreeRTOS/