ТЕХНИЧЕСКИЙ ОБЗОР ОСОБЕННОСТЕЙ ОПЕРАЦИОННОЙ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ TI-RTOS

А. Е. Чуфырев, В. А. Устюгов
Сыктывкарский государственный университет имени Питирима Сорокина
Россия, 167001, г. Сыктывкар, Октябрьский пр., 55

В статье приведен краткий технический обзор операционной системы реального времени для микроконтроллеров Texas Instruments Real-Time Operating System. Она упрощает разработку сложных проектов для встроенных систем, выступая посредником между прикладной программой и аппаратными средствами. Рассмотрены принципы функционирования, механика работы, основные возможности ОС. В заключительной части затронуты способы создания приложений. Многие описанные принципы справедливы и для других операционных систем реального времени для микроконтроллеров.

Ключевые слова: *операционная система реального времени, ОСРВ, микроконтроллер.*

TECHNICAL OVERVIEW OF FEATURES OF REAL-TIME OPERATING SYSTEM TI-RTOS

A. E. Chufyrev, V. A. Ustyugov
Pitirim Sorokin Syktyvkar State University
55 Oktyabrskiy Pr., 167001, Syktyvkar, Russia

The article presents a brief technical overview of real-time operating system for microcontrollers "Texas Instruments Real-Time Operating System". It simplifies development of complex projects for embedded systems and acts as an intermediary between the application and the hardware. The article describes the principles of operations, mechanics and the main features of the OS. In the final part there are some methods to create applications. A lot of information also describe the others real-time operating system for MCUs.

Keywords: *real-time operating systems, RTOS, TI-RTOS, microcontrollers, ARM.*

TI-RTOS Kernel (SYS/BIOS)

Компания Texas Instruments уже более 30 лет разрабатывает и поддерживает собственную open-source [3] операционную систему реального времени TI-RTOS. Она предоставляется бесплатно и включает в себя как непосредственно компоненты ОСРВ в частности, ядро TI-RTOS Kernel или SYS/BIOS, так и широкий спектр сопутствующих инструментов (документацию, аппаратные и сетевые драйверы, средства разработки, отладки и т.д.) (рис. 1[1]). Доступ к специфичным функциям ОС осуществляется с помощью предоставляемого API.

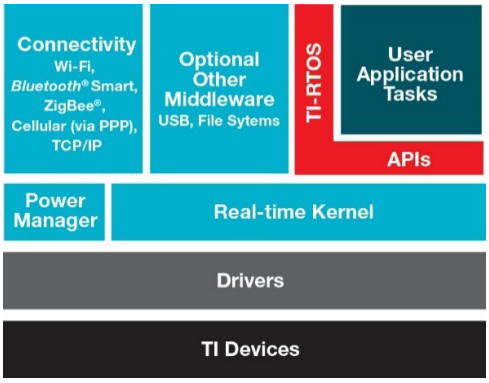


Рисунок 1 – Структурная схема TI-RTOS

TI-RTOS Kernel (англ. «ядро») по своей сути является библиотекой сервисов, которые пользователь может добавлять в свою систему чтобы организовать управление памятью, анализ в реальном времени, планирование потоков и синхронизацию между ними, передача данных от одного потока к другому. Таким образом, распространенные задачи уже решены, и пользователь может сосредоточиться на реализации конкретного программного алгоритма.

Составление программ, способных работать под ОС, производится на языке С (с возможными ассемблерными вставками). Сформулируем принципы построения и действия этой ОС [2]:

- Стремление к как можно меньшему потреблению ресурсов;
- Встроенный планировщик – приоритетный. Это означает, что процесс с наивысшим приоритетом в данный момент времени всегда выполняется в первую очередь;
- BIOS включается в работу программы только при его вызове специальными функциями. Представление о BIOS как о глобальном цикле, охватывающем все остальные команды, неверно;
- Методы BIOS оперируют объектами – структурами языка С. Когда вы создаете или изменяете какую-либо структуру, это не затрагивает остальные;
- BIOS – детерминирован. Любая команда выполняется строго определенное время, за исключением динамических операций с памятью (например, функция `malloc()`);
- Встроенные в BIOS средства журналирования выполнены прозрачными и потребляющими мало ресурсов, поэтому их рекомендуется оставлять даже в конечной «прошивке» устройства.

Потоки

Все ОСРВ помимо классических функций оперируют понятием *поток*. Поток – это функция, выполняющаяся в определенном контексте, который задает тип потока, его приоритет и другие параметры. Все потоки находятся в определенный момент времени в одном из трех состояний: выполняющемся,

[1] *Здесь и далее иллюстрации приведены из официальной документации Texas Instruments*

готовым к выполнению и заблокированном. В TI-RTOS существуют 4 вида [7] потоков: HWI, SWI, Task и Idle (рис. 2).

HWI

HWI (HardWare Interrupt) – аппаратное прерывание. BIOS автоматически выполняет рутинные операции (сохранение текущего состояния процессора, очистка флага прерываний и т.д.) и кроме того привносит технологию «умного возвращения» (smart return). Данный алгоритм после завершения обработки прерывания возвращается к потоку с наивысшим приоритетом, если таковой был включен в ходе выполнения HWI, а не к предыдущему состоянию процессора. Для особо критичных ко времени реакции ситуаций оставлена возможность зарегистрировать прерывание вне TI-RTOS.

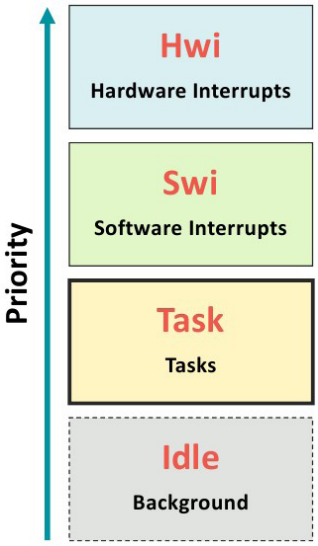


Рисунок 2 – Распределение приоритетов потоков (выше – больше)

Согласно концепции построения BIOS-программ, обработчик прерывания должен завершать свою работу как можно быстрее, выполняя лишь самые срочные этапы реагирования (как правило, это операции с аппаратным оборудованием) [4]. Этим достигается отсутствие конфликтов и зацикливаний при большом количестве прерываний и можно обойтись без их вложения друг в друга (nesting). С этой целью в TI-RTOS предусмотрена функция `Swi_post()`, передающая остаточную активность некоторому потоку типа SWI, завершающему начатую HWI работу.

Приоритет HWI, как аппаратного прерывания, физически заложен в кристалле и не может быть изменен. Отсюда следует, что по шкале приоритетов планировщика SYS/BIOS все HWI располагаются выше остальных типов потоков.

SWI

SWI (SoftWare Interrupt) – программное прерывание. Служит, в первую очередь, как «продолжение» соответствующего HWI, вызвавшего функцию `Swi_post()`. Важно понимать, что SWI не существуют для планировщика до тех пор, пока они не будут вызваны. В этом прослеживается аналогия с аппаратными прерываниями.

BIOS позволяет назначить SWI-потокам до 32 уровней приоритета – чем выше номер, тем быстрее выполнится код. Возможно назначение одного уровня нескольким SWI, в таком случае они будут располагаться в очереди планировщика в порядке времени их вызова – более ранний вызов гарантирует и более раннее выполнение. Преимуществом такого назначения является значительное освобождение общего стека, т.к. все одноприоритетные SWI обладают единым стеком. Существует возможность повысить (но не понизить) приоритет какого-либо SWI в процессе работы программы. Ограничение вызвано тем, что все SWI используют системный стек [7] и при понижении приоритета становятся возможными такие ситуации, которые приведут к потерям данных и сбоям. По шкале приоритетов SWI располагаются ниже HWI.

Task и Idle. Семафоры

Последний тип потока в TI-RTOS – Task (англ. «задача»). Данный вид потока сконструирован для непрерывного и параллельного выполнения вместе с другими объектами типа Task (рис. 3).

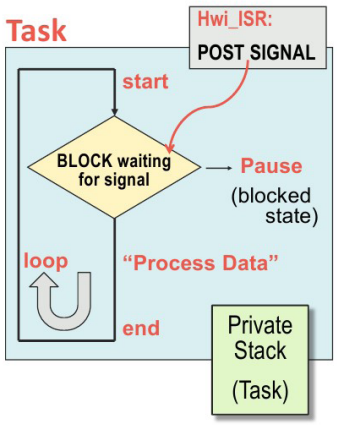


Рисунок 3 – Схема работы потока типа Task

SYS/BIOS обеспечивает особый сигнальный механизм для таких потоков – семафоры (semaphore) [6]. Когда где-либо в ходе исполнения программы встречается функция `Sempahore_post()`, с Task снимается блокировка, и он ожидает своей очереди к выполнению.

Обычно организацию Task-функций выполняют по следующему алгоритму:

1. Вступительный код, выполняющий подготовительную работу – устанавливающий окружение (environment) Task. Он выполняется однократно;
2. Цикл (чаще всего бесконечный), содержащий функцию `Sempahore_pend()`, которая блокирует исполнение дальнейших команд до сигнала – вызова функции `Sempahore_post()`. Если же Task разблокирован, то выполняется дальнейший код цикла и процесс повторяется;
3. Конечный код, очищающий окружение Task перед выходом. Этот участок потока выполняется единожды и только в случае выхода из цикла.

Каждый Task обладает собственным стеком, что позволяет ему безопасно блокироваться или становиться на паузу без опасений потери данных. Размер стека задается пользователем. Окружение Task сохраняется в системе до тех пор, пока он не будет закрыт. Task готов к выполнению как только он был создан. Так, статический[2] Task начинает выполняться при вызове функции `BIOS_start()`.

Task обладают самым низким приоритетом в операционной системе. Возможно назначение до 32 уровней приоритета [2]. При этом, благодаря наличию собственного стека становится возможным динамическое изменение приоритета Task в любом направлении.

Одной из важных разновидностей Task является ждущий процесс (Idle). Это поток с самым низким приоритетом в очереди планировщика из всех. Он выполняется тогда, когда процессор закончил выполнение всех остальных потоков. Пользователь может назначить определенную функцию в качестве Idle. Однако, даже если в системе не зарегистрирован Idle-поток в явном виде, он тем не менее существует для планировщика и содержит в себе (в простейшем случае) пустой бесконечный цикл, либо некоторые операции самодиагностики (например, проверку стека на переполнение).

[2] *См. раздел «Статическое и динамическое создание объектов».*

Семафоры представляют собой служебные сигнальные объекты операционной системы. Они также создаются пользователем и назначаются каждый своему Task. Одним из атрибутов семафора является счетчик, автоматически инкрементируемый при вызове функции `Sempahore_post()` и декрементируемый при вызове `Sempahore_pend()` (рис. 4). Программист может назначить, как долго поток будет пребывать в блокированном состоянии – от нескольких системных тактов[3], до бесконечного ожидания. Не рекомендуется использовать бесконечное блокирование Task, т.к. в этом случае процессор может навсегда «застрять» в ожидании семафора, который никогда не будет отмечен по каким-либо причинам.

Механика работы семафоров не ограничивается перечисленными выше принципами и предоставляет множество различных вариантов и комбинаций для максимально гибкой и эффективной настройки работы приложения. Так, например, в SYS/BIOS предусмотрен сервис Событий (Events). Это более сложный способ организации Task-потоков, когда с процесса снимается блокировка только при определенной логической комбинации (логические операции AND и OR) нескольких семафоров (до 32).

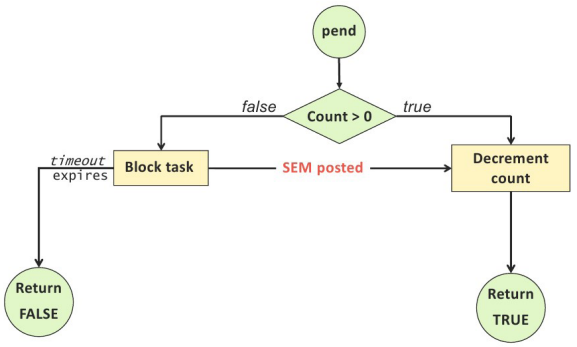
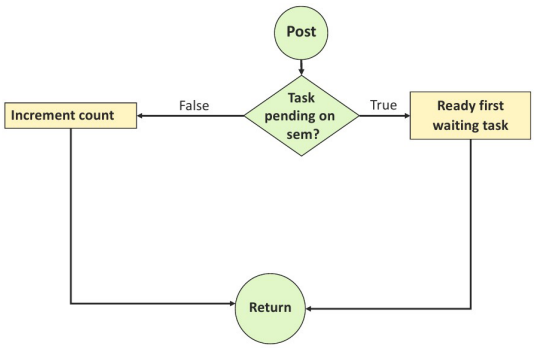


Рисунок 4 – Алгоритм работы функции Sempahore_pend() (вверху) и Sempahore_post() (внизу)



Статическое и динамическое создание объектов

Когда мы говорим про «статический» объект, например, Task, это означает, что тело и параметры этого Task помещаются в оперативную память при старте SYS/BIOS и не могут быть удалены из нее в дальнейшем. Ресурсы статического объекта доступны все время, пока работает система. Такой тип размещения прост в использовании и обеспечивает полностью детерминированный доступ. Тем не менее, вы не можете повторно использовать занятую оперативную память во время работы программы. Такая модель оптимальна для ресурсов, доступ к которым необходим постоянно [9].

В SYS/BIOS для некоторых объектов, в том числе для HWI, SWI, Task, Semaphore есть возможность их динамического создания. Объявленные таким образом объекты хранятся во Flash-памяти в составе программного кода, однако после включения контроллера и загрузки приложения они не сразу размечаются в оперативной памяти. Только после соответствующих команд инициализации в определенном месте программы, они помещаются в RAM и могут выполнять свои функции. В дальнейшем мы можем в любой момент времени закрыть такие динамически созданные объекты и выгрузить их из оперативной памяти. Для хранения динамических объектов программист должен выделить в оперативной памяти место под специальный буфер, называемый *heap* (англ. «куча»), и определить его размер. Однако следует учитывать, что сам процесс распределения памяти не детерминирован во времени, т.к. BIOS каждый раз вынужден искать требуемые блоки памяти, что занимает различное число тактов [1].

Временной модуль

Для функций, работающих по заданным временным интервалам, в SYS/BIOS предусмотрен специальный модуль – Clock Module. TI-RTOS «отбирает» у пользователя один аппаратный таймер для генерации «тиков» соответствующей частоты.

Предположим, что необходимо обслужить 3 потока с частотами 10, 1500 и 6000 Гц. В таком случае необходимо выбрать частоту следования импульсов таймера BIOS равной частоте третьего потока – наибольшей. Пользователю следует указать это в конфигурационном файле ОС, а создание прерывания и обработчика система берет на себя. Таймер будет отсчитывать промежутки времени, равные 1/6000 секунды, и как только значение внутреннего счетчика совпадет с одним из чисел 600, 4 и 1 соответственно, BIOS выполнит заданные потоки.

Межпоточная коммуникация

В большинстве ситуаций потоки так или иначе должны сообщаться друг с другом и почти всегда встает вопрос о безопасности их взаимной коммуникации. Самым простым решением проблемы можно назвать установку для потоков, требующих обмена общим «сообщением», одинакового приоритета. Обладающие одним приоритетом процессы не в состоянии вступить в спор за данные. Такой метод, конечно, не является строгим решением, т.к. мы далеко не всегда можем установить один приоритет для потоков (например, для разнородных, таких как HWI и Task), и кроме того, сводятся на нет все преимущества системы приоритетов. Поэтому в SYS/BIOS предусмотрены несколько механизмов безопасной межпоточной коммуникации (inter-thread communications). Их можно разделить на две большие группы: модель «производитель-потребитель» и модель «параллельный доступ» [7].

Модель «производитель-потребитель»

При взаимодействии такого типа потоки согласуются друг с другом формальным протоколом – определенным набором правил обмена информацией – что исключает конфликты. Обмен информацией осуществляется только в одну сторону – от производителя к потребителю.

Пусть поток А – «производитель» некоторых данных или, в терминах TI-RTOS, сообщения. Поток А помещает их в особый «контейнер» BIOS и сигнализирует потоку В о том, что данные готовы, и он может их прочитать. Поток В, являющийся потребителем, был в заблокированном состоянии и ожидал готовности данных от потока А. После «пробуждения» (т.е. получения сигнала) поток В считывает данные и выполняет над ними заданные операции. Исключение конфликтов здесь достигается строгим порядком доступа к данным. Потоки с таким подходом повышают свою модульность, что поло-

[3] *См. раздел «Временной модуль».*

жительно сказывается на возможности их повторного использования [2]. Сервисами, предоставляющими такое «общение» в SYS/BIOS служат *Queues* (англ. «очереди») и *Mailboxes* (англ. «почтовые ящики»).

Queues

Queue внешне представляет собой структуру языка C, определенную программистом. Она может содержать указатели, буферы, переменные и т.д. Поток A помещает сообщение (message) (функция `Queue_put()`) в такой контейнер, сигнализирует Потоку B, что данные готовы, и затем Поток B забирает сообщение (функция `Queue_get()`). Как следует из названия сервиса – «очередь» – сообщения поступают в контейнер один за другим и извлекаются из него в том же порядке (принцип FIFO). Размер Queue ограничен только объемом памяти, поэтому он может быть любого желаемого размера и вам не нужно указывать его заранее [7]. Поэтому Queue стоит использовать в тех случаях, когда вы не знаете наперед объем пересылаемых данных – очередь можно расширить или сократить в процессе исполнения кода. Однако в сервисе Queue нет встроенного сигнального механизма, оповещающего поток B, и его придется написать вручную. В качестве такого механизма может выступать, например, семафор.

Mailboxes

Почтовый ящик – структура со статичным размером и длиной – в отличие от очередей. При создании Mailbox необходимо указать число сообщений и размер каждого из них. Для отправки и приема сообщений используются также две функции:

- `Mailbox_post()`: помещает сообщение в ящик или блокирует выполнение, если он полон, ожидая, пока потребитель прочитает сообщение;
- `Mailbox_pend()`: забирает сообщение из ящика или блокирует выполнение, если он пуст, ожидая поступления сообщения от производителя.

Таким образом, Mailbox имеет встроенный механизм сигнализации. Недостатком почтовых ящиков является хранение его копии каждым из потоков A и B.

Модель параллельного доступа. Мьютексы

Как и в предыдущем пункте, рассмотрим два процесса A и B. Если между ними реализована модель постоянного (параллельного) доступа, то у каждого потока, по определению, есть доступ к общим данным в любой момент времени. Естественно, что для предотвращения конфликтов должна применяться какая-либо защита [4].

Одна из (но не единственная) реализаций согласованного доступа в BIOS – сервис *мьютексов* (MutEx, англ. *mutual exclusion* – «взаимное исключение»). Идея мьютексов состоит в том, что только один поток в данный момент времени обладает «ключом» (объектом типа мьютекс) для доступа к общим данным. После того, как этот процесс выполнит свои задачи по обработке данных, он возвращает ключ BIOS, который в свою очередь передает его следующему потоку, запросившему мьютекс. Таким образом, ОС распределяет мьютексы между потоками согласно задумке программиста. Мьютексы можно представить как простейшие двоичные семафоры.

Отрицательными явлениями при использовании мьютексов являются инверсия приоритета, deadlock и livelock (по своей сути виды рекурсий, вызывающие взаимные блокировки потоков). В BIOS предусмотрены инструменты для устранения подобных проблем.

TI-RTOS-приложения

Создание

Вести разработку приложений под TI-RTOS возможно с применением различных инструментов, однако полную функциональность и удобство способна предоставить только фирменная интегрированная среда разработки (ИСР) Texas Instruments *Code Composer Studio* (сокращенно CCS), поэтому программирование всегда ведется с ее участием. Продукт распространяется бесплатно с ограничением на объем создаваемого кода; доступен для операционных систем Windows, Linux, OS X [6].

Структура

Основное отличие TI-RTOS-приложения заключается в наличии особого файла проекта – файла конфигурации `.cfg`. Он хранит информацию о специфичных для ОС определений: общие настройки системы (размеры стека и кучи, частота тактирования), потоки (их параметры и поведение, размеры стеков), сервисы логирования и т.д. В CCS предусмотрен графический интерфейс доступа к `.cfg` -файлу. Продвинутые пользователи могут править этот файл в ручном режиме.

В основном коде программы необходимо указать как используемые классические, так и специальные библиотеки TI-RTOS. Во время компиляции и загрузки программы на МК CCS объединяет файлы проекта в единственный исполняемый файл .out, связывая между собой команды исходного кода и

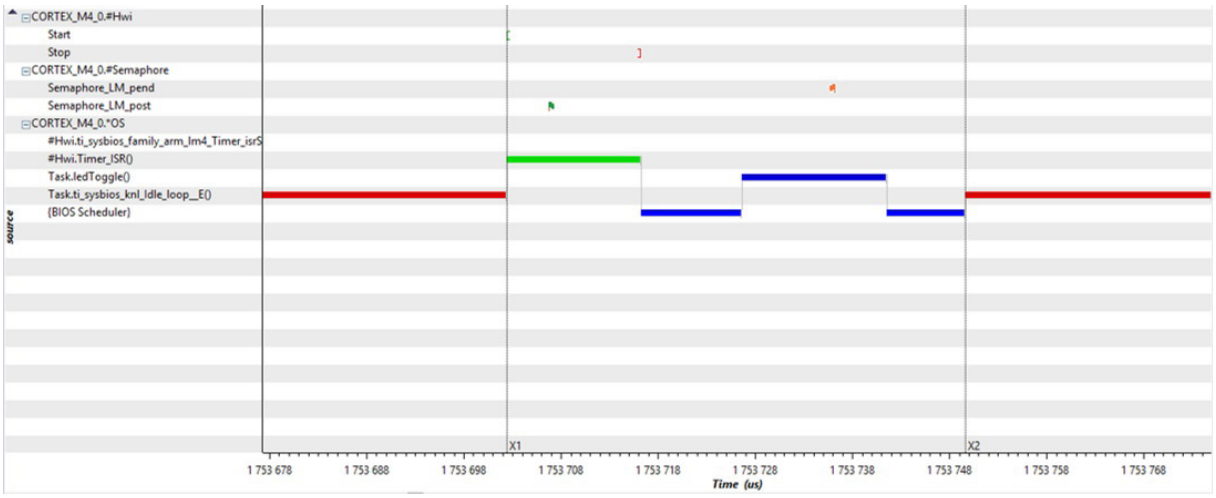


Рисунок 5 – Графическое отображение работы потоков BIOS на временной линии в CCS

`.cfg` -файла.

Отладка и сбор информации

Для устранения проблем и оптимизации приложений предусмотрен инструментарий отладки с большим выбором сервисов. Среди них:

- **UIA** (Unified Instrumentation Architecture) – набор методов создания и сбора отладочной информации. В частности, UIA предоставляет облегченный аналог функции `printf() – Log_Info()`. Оригинальный вариант этой функции может потреблять до 10 кБ памяти и 10 тысяч циклов [5], в то время как `Log_Info()` тратит всего лишь 40 байт и 40 итераций.
- **ROV** (RTOS Object Viewer) – сервис, позволяющий посмотреть состояние всех ресурсов ОС в режиме паузы программы: системные сообщения, параметры потоков и т.д. Отдельно стоит выделить возможность строить временные диаграммы потоков и событий, графически изображать загруженность ЦПУ и памяти (рис. 5).

Timestamp (англ. «метка времени») – небольшой сервис, позволяющий измерять время выполнения той или иной операции. Специальные функции помещаются в начало и конец исследуемого участка и на выходе получаем значение в тактах процессора;

- **Традиционные методы отладки** – пошаговое выполнение, точки останова, просмотр содержимого регистров и памяти налету, отслеживание переменных и выражений [8] и т.д..

Стратегии применения планировщика

Благодаря гибкости планировщика в TI-RTOS организацию работы потоков можно составить различными способами. Рассмотрим 4 самых распространенных из них[7]:

- **Deadline Monotonic**. Хорошая, но нечасто используемая стратегия. Идея проста – самым важным задачам – самые высокие приоритеты;
- **Rate Monotonic**. Наиболее широко распространенная стратегия. Максимальный приоритет отдается процессу с самой большой частотой. Доказано, что если загрузка процессорного ядра не превышает 69%, то такой подход гарантирует работу в режиме реального времени без пропусков и запаздываний;
- **Stu Monotonic**. Подход, противоположный предыдущему. Изначально все потоки обладают одним приоритетом. Если процессор слабо загружен, то мы динамически повышаем приоритеты важных потоков. Таким образом, выравнивается нагрузка на ядро и все потоки успевают завершиться в режиме реального времени. Сопутствующий плюс – уменьшение размера стека (стек не растет при использовании процессов одного приоритета).
- **Dynamic Priorities**. Динамически повышаем приоритеты потоков, не успевающих выполнить свою работу в срок.

Дополнительные модули TI-RTOS

Подключаемые расширения ОС распространяются вместе с общим пакетом TI-RTOS [5] и включают в себя такие модули как:

- **Драйверы периферии.** Создателями TI-RTOS написаны драйверы для упрощения доступа к периферии в рамках работы с ОСРВ. Доступны следующие драйверы:
  - EMAC (реализация Ethernet-протокола);
  - UART, I2C, SPI;
  - GPIO;
  - SDSPI (взаимодействие с внешними SD-картами по интерфейсу SPI);
  - USBMSCHFatFs (реализация USB-интерфейса для подключения внешних Flash-дисков);
  - Wi-Fi;
  - Watchdog.
- **FatFs** – сторонняя свободная файловая система типа FAT (File Allocation Table), разработанная специально для использования во встроенных системах. Она позволяет удобно хранить данные на внешнем носителе (Flash-диск либо карта памяти SD).
- **NDK** (Network Developer's Kit) – группа API для организации разнообразных сетевых функций. Стек протоколов, предоставляемый платформой NDK, позволяет добавить коммуникационные возможности для уже существующих приложений. Доступны реализации следующих уровней (по модели OSI) и протоколов в них:
  - Application layer: DHCP-клиент и -сервер, DNS, HTTP, Telnet;
  - Transport layer: TCP, UDP, NAT;
  - Network layer: IP, ICMP, route;
  - Data Link layer: PPP, PPPoE.

ЛИТЕРАТУРА

1. ARM Optimizing C/C++ Compiler v15.9.0.STS User's Guide / авт. Texas Instruments Inc.. – Texas Instruments, Post Office Box 655303, Dallas, Texas 75265 : [б.н.], Сентябрь 2015 г.. – SPNU151K.
2. Intro to the TI-RTOS Kernel Workshop: Student Guide [pdf] / авт. Mindshare Advantage в сотрудничестве с Texas Instruments Inc.. – 2015 г..
3. Open Source RTOS [В Интернете] // OSRTOS. – http://www.osrtos.com/.
4. Programming Embedded Systems in C and C++ [Книга] / авт. Barr Michael. – [б.м.] : O'Reilly Media, 1999.
5. The Definitive Guide to ARM® Cortex®-M3 and Cortex-M4 Processors [Book] / auth. Yiu Joseph. – The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, UK; 225 Wyman Street, Waltham, MA 02451, USA : Elsevier Inc., 2014. – Third Edition.
6. TI-RTOS – Texas Instruments Wiki [В Интернете] // http://wiki.ti.com/. – Texas Instruments, 2015 г.. – http://processors.wiki.ti.com/index.php/TI-RTOS.
7. TI-RTOS 1.21 User's Guide [pdf] / авт. Texas Instruments Inc.. – Post Office Box 655303 Dallas, Texas 75265 : [б.н.], Январь 2014 г.. – SPRUHD4E.
8. Ядро Cortex-M3 компании ARM. Полное руководство [Книга] / авт. Ю Джозеф / перев. Евстифеева А. В.. – Москва : Додэка-XXI, 2012.
9. Язык программирования Си [Книга] / авт. Керниган Брайан и Ритчи Деннис. – Москва : Вильямс, 2006.

Поступила в редакцию 14.01.2016