Разработка программного обеспечения

для сети ZigBee на базе библиотеки EmberZNet. Часть II

Первые сети ZigBee в большинстве случаев проектировались на базе готовых радиомодулей со встроенным программным обеспечением. Такой подход существенно упрощает и ускоряет разработку. Вместе с тем для больших проектов, в которых количество устройств достигает десятков тысяч, экономически целесообразно разрабатывать собственные радиомодули и программное обеспечение.

Сергей Солодунов
sys@efo.ru

Компания Ember, производитель однокристальных решений для сетей ZigBee, поставляет для своих кристаллов библиотеку EmberZNet, позволяющую создавать полностью ZigBee-совместимые устройства. В первой части статьи рассматривался начальный этап разработки сети ZigBee — планирование, во время которого определяются основные параметры сети, выбираются оптимальные методы маршрутизации, типы используемых устройств, их количество и режимы работы [1]. Используя эти результаты, разработчик приступает к созданию программного кода приложения. Во второй части статьи будут описаны основные принципы разработки ZigBee-приложений на базе библиотеки EmberZNet.

Библиотека EmberZNet предоставляет два возможных варианта разработки приложений. Первый — это разработка на базе демонстрационных примеров, входящих в состав пакета EmberZNet (например, на базе приложений проекта Sensor/Sink). Второй — использование генератора конфигурационных файлов Application Builder. Наиболее удобным и универсальным является второй способ. В качестве основы при создании нового проекта программа Application Builder использует входящий в состав пакета EmberZNet набор исходных кодов Application Framework (каркас приложения), которые содержат реализацию всех стандартных устройств и кластеров ZigBee-профилей Home Automation (HA), Smart Energy (SE) и Construction & Building Automation (CBA) [1, 2].

Задачи стандартного EmberZNet-приложения

Вне зависимости от того, какой способ будет выбран разработчиком для создания собственного приложения, существует определенный

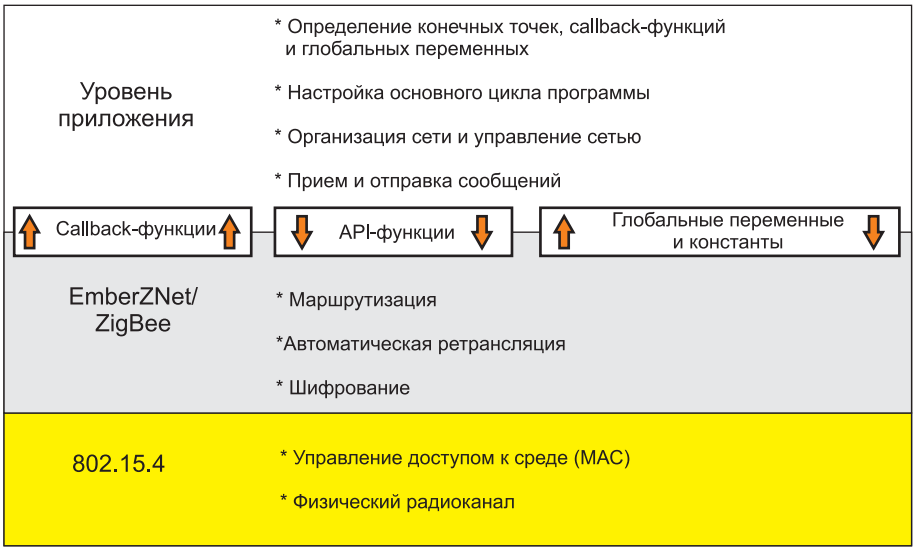

Рис. 1. Распределение функций программного обеспечения ZigBee-устройства

Таблица 1. Глобальные переменные, связанные с определением конечных точек

Переменная	Описание
int8u emberEndpointCount	Переменная; определяет, сколько конечных точек на этом узле.
EmberEndpointDescription PGM endpointDescription	Дескриптор конечной точки; определяет такие параметры, как идентификатор профиля, идентификатор устройства, версия устройства и др. Обратите внимание, что ключевое слово PGM используется только чтобы сообщить компилятору, что эта структура должна храниться во Flash-памяти, т. к. она не изменяется в течение жизни устройства.
EmberEndpoint endpoint	Структура конечная точка; содержит поля: идентификатор конечной точки, дескриптор конечной точки, поддерживаемые входные и выходные ZigBee-кластеры.
EmberEndpoint emberEndpoints[ ]	Массив, используемый стеком EmberZNet, в котором перечисляются все конечные точки.

Таблица 2. Основные константы

Константа	Описание
EMBER_STACK_PROFILE	Профиль стека: 1 (ZigBee) или 2 (ZigBee Pro).
EMBER_SECURITY_LEVEL	Уровень безопасности: 0 (no security) или 5 (шифрование и аутентификация).
EMBER_CHILD_TABLE_SIZE	Размер таблицы дочерних устройств (спящих + мобильных).
EMBER_RESERVED_MOBILE_CHILD_ENTRIES	Количество записей в таблице дочерних устройств, зарезервированных под мобильные узлы.
EMBER_NEIGHBOR_TABLE_SIZE	Размер таблицы соседних устройств.
TIME_BEFORE_SINK_ADVERTISE	Временной интервал между широковещательными запросами Many-to-one Route Discovery рассылаемых узлом — концентратором данных (Sink).

круг задач, которые выполняются любым приложением, реализованным на базе библиотеки EmberZNet.

На рис. 1 показано распределение функций программного обеспечения ZigBee-устройства по уровням их реализации. Видно, что первые два нижних уровня, 802.15.4 и EmberZNet, выполняются аппаратно и стеком EmberZNet соответственно. Реализация задач верхнего уровня (уровня приложения) остается за разработчиком. Взаимодействие уровня приложения с уровнем стека осуществляется посредством callback-функций, API-функций, глобальных констант и переменных.

Рассмотрим каждую из задач уровня приложения на примере демонстрационного проекта Sensor/Sink из библиотеки EmberZNet.

Определение конечных точек, callback-функций, глобальных констант и переменных

Для адресации нескольких устройств, доступных через один ZigBee-узел, используются специальные идентификаторы — конечные точки (например, это могут быть два устройства, выключатель света и диммер, подключенные к одному приемопередатчику ZigBee). Таким образом, любые устройства (за исключением простых ретрансляторов) должны иметь хотя бы одну конечную точку. В таблице 1 перечислены глобальные переменные, связанные с определением конечных точек, которые должны быть заданы в приложении.

Учитывая требования к системе, выработанные в результате планирования сети, программисту необходимо определить значения ряда констант, которые будут использоваться как стеком, так и самим приложением. В демонстрационных приложениях Ember основные константы «по умолчанию» определены с помощью директив ***#ifndef*** в файле ***ember-configuration-defaults.h.*** Чтобы изменить значения этих констант, необходимо переопределить их в файле **CONFIGURATION_HEADER**, путь к которому указывается в настройках препроцессора. В таблице 2 перечислены некоторые основные константы, которые должны быть заданы в приложении.

При определении указанного в таблице параметра EMBER_RESERVED_MOBILE_CHILD_ENTRIES необходимо учитывать, что количество поддерживаемых роутером мобильных устройств может быть больше значения данного параметра, так как несколько мобильных узлов могут совместно использовать одно зарезервированное место в таблице, отсоединяясь/присоединяясь к роутеру поочередно. Для этого нужно, в зависимости от требований системы, подобрать временные интервалы: время жизни мобильного узла в таблице родителя и интервал, с которым он будет выполнять переприсоединение к сети. Эксперимент показывает, что в демонстрационном приложении Ember одним зарезервированным местом могут без проблем пользоваться два мобильных узла.

Как уже упоминалось, одним из инструментов взаимодействия приложения и уровня стека Ember являются callback-функции. Они представляют собой процедуры, которые определяются в приложении, но вызываются стеком в известных ситуациях. Например, если стек принял из сети сообщение, которое адресовано данному узлу, то он вызывает callback-функцию emberIncomingMessageHandler (), содержание которой заранее задал программист конечного приложения. Перечень основных callback-функций, которые должен определить разработчик, приведен в таблице 3. Полное описание всех callback-функций можно найти в html-руководстве EmberZNet API Reference, входящем в состав пакета EmberZNet [3].

Настройка основного цикла программы

На рис. 2 изображен основной цикл программы.

Как видно на схеме, после включения питания перед входом в основной цикл выполняется блок инициализации. В этом блоке в первую очередь конфигурируются последовательные порты (SPI, UART). Далее вызывается функция инициализации стека emberInit (). Ее необходимо выполнить перед вызовом любой другой функции стека, за исключением инициализации последовательных портов (чтобы можно было выводить сообщения об ошибках). При использовании в программе каких-либо дополнительных библиотек в этом блоке необходимо вызвать соответствующие функции их инициализации.

Для возобновления работы узла в сети после перезагрузки или выключения питания вызывается функция emberNetworkInit (). Если узел прежде являлся членом сети, то она восстанавливает параметры этой сети (хранящиеся в энергонезависимой памяти кристалла) и ассоциирует с ней данный узел. Если ассоциация узла прошла успешно, то emberNetworkInit () возвращает значение EMBER_SUCCESS. В противном случае возвращается код, указывающий

Таблица 3. Перечень основных callback-функций

Callback-функция	Описание
emberMessageSentHandler( )	Вызывается после завершения передачи сообщения. Возвращает статус, указывающий, прошла ли передача успешно (было получено подтверждение в течение ожидаемого тайм-аута) или нет.
emberIncomingMessageHandler( )	Вызывается всякий раз, когда из сети приходит новое сообщение (не относится к сообщениям для автономного загрузчика или сообщениям, которые ретранслируются данным узлом).
emberStackStatusHandler( )	Вызывается при изменении статуса стека. Используется, как правило, для инициации адекватного реагирования на новое состояние стека. Приложение также может использовать функции emberStackIsUp () и emberNetworkState (), чтобы получить информацию о текущем состоянии в любой момент времени.
emberScanCompleteHandler( )	Вызывается после завершения операции сканирования сети.
emberNetworkFoundHandler( )	Вызывается, если в процессе сканирования была найдена сеть.

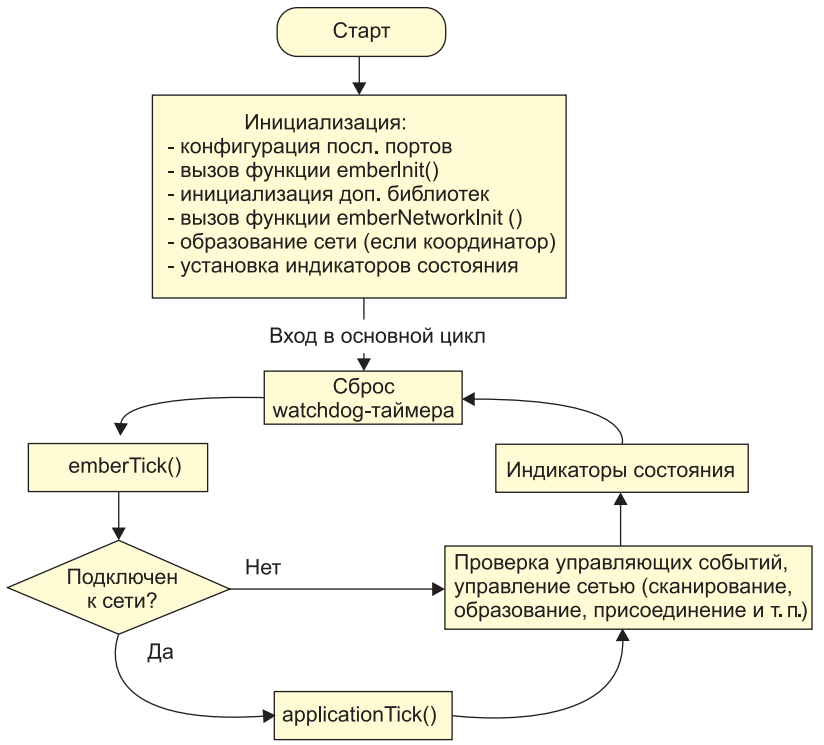

Рис. 2. Алгоритм работы стандартного EmberZNet-приложения

причину неудачи. Если устройство ранее не было подключено к сети, функция emberNetworkInit () возвращает значение EMBER_NOT_JOINED («не подключен»).

Если узел определен как координатор сети (демонстрационное приложение Sink из библиотеки EmberZNet) и не является членом сети (emberNetworkInit () возвращает значение, отличное от EMBER_SUCCESS), то программа может, в зависимости от требований конкретного проекта, образовать сеть с заранее заданными параметрами с помощью функции emberFormNetwork () или просканировать эфир на наличие свободных каналов с наименьшим уровнем помех (функция emberScanForUnusedPanId ()) и, используя результаты поиска, образовать сеть.

В конце блока инициализации устанавливаются значения индикаторов, отображающих состояние приложения.

В начале основного цикла выполняется сброс сторожевого таймера. Далее вызывается процедура EmberTick (), которая выполняет обязательные периодические операции стека EmberZNet и должна вызываться как минимум один раз в течение основного цикла. Затем с помощью функции emberNetworkState () программа проверяет статус узла в сети. Если его статус EMBER_JOINED_NETWORK («присоединен»), то выполняется основная функция приложения applicationTick (). В противном случае выполнение программы переходит в блок проверки управляющих событий. В этом блоке в зависимости от управляющих событий и логики работы программы осуществляется соответствующее управление сетью (сканирование, образование сети, присоединение и др.). В демонстрационных приложениях Ember управляющими событиями являются нажатия кнопок, подключенных ко входам микросхемы, и команды, поступающие через последовательный интерфейс. В конечном приложении эти события могут быть встроены в само приложение.

Функция ApplicationTick () отвечает за выполнение основных задач приложения: переключение режимов работы, опрос состояния датчиков, обработка событий, отправка сообщений и т. п.

В конце основного цикла устанавливаются значения индикаторов состояния приложения (светодиоды, буквенно-цифровой дисплей).

Организация сети и управление сетью

Библиотека EmberZNet располагает большим набором API-функций, предоставляющих широкие возможности по организации и управлению ZigBee-сетью [3]. В их число входят функции, выполняющие такие основные задачи, как:

- обнаружение сети;
- присоединение к сети;
- формирование новой сети.

Соответствующие этим операциям функции стека EmberZNet приведены в таблице 4.

Все перечисленные функции возвращают значение типа EmberStatus, это код длинной в один байт, указывающий на успех или причину неудачи вызова той или иной функции. EmberStatus является самым распространенным возвращаемым значением API-функций стека EmberZNet, поэтому все его возможные значения перечислены и описаны в разделе Status Codes html-руководства EmberZNet API Reference.

Прием и отправка сообщений

Прием сообщений в приложении осуществляется с помощью callback-функции emberIncomingMessageHandler() (табл. 3). Использовать данную функцию нужно с осторожностью, так как ее длительное выполнение может привести к задержке получения последующих сообщений. Поэтому в callback-функции программа должна только извлекать принятые данные, а их дальнейшую обработку выполнять в основном цикле. Вообще говоря, этого правила следует придерживаться и при работе с любыми другими callback-функциям, так как это гарантирует своевременное реагирование программы на события, происходящие в стеке.

Для отправки сообщений используются специальные буферы стека, которые необходимо выделить в памяти и заполнить передаваемыми данными с помощью функции emberFillLinkedBuffers () (табл. 5). В качестве аргументов в эту функцию передаются указатель на отсылаемые данные и переменная, указывающая размер этих данных в байтах. Функция emberFillLinkedBuffers () возвращает указатель на первый буфер в цепочке выделенных буферов стека. Этот указатель программист должен передать в качестве аргумента в функцию отправки сообщений (аргумент ***message*** в функциях emberSendUnicast, emberSendBroadcast и emberSendMulticast). Статус отправки, указывающий, прошла передача успешно или нет, возвращается callback-функцией emberMessageSentHandler() (табл. 3).

Таблица 4. Основные API-функции для организации и управления сетью

API-функция	Описание
EmberStatus emberStartScan (EmberNetworkScanType scanType, int32u channelMask, int8u duration)	Функция сканирует все ZigBee-каналы, заданные маской channelMask. В зависимости от параметра scanType сканирование производится либо по уровню RSSI принимаемого сигнала, либо по доступным для присоединения сетям. Результаты сканирования доступны через callback-функцию emberNetworkFoundHandler ().
EmberStatus emberJoinNetwork (EmberNodeType nodeType, EmberNetworkParameters * parameters)	Функция присоединения к сети с заданными параметрами. Эта процедура может занять несколько секунд. Отправлять сообщения можно только после вызова стеком callback-функции emberStackStatusHandler () со значением аргумента EMBER_NETWORK_UP.
EmberStatus emberFormNetwork (EmberNetworkParameters * parameters)	Функция образования сети с заданными параметрами (parameters).

Таблица 5. Основные API-функции, используемые при отправке сообщений

API-функция	Описание
EmberStatus emberSendUnicast (EmberOutgoingMessageType type, int16u indexOrDestination, EmberApsFrame * apsFrame, EmberMessageBuffer message)	Отправка адресного сообщения в соответствии со спецификацией ZigBee PRO.
EmberStatus emberSendBroadcast (EmberNodeId destination, EmberApsFrame * apsFrame, int8u radius, EmberMessageBuffer message)	Отправка широковещательного сообщения в соответствии со спецификацией ZigBee PRO. Сообщение доставляется всем узлам, находящимся в пределах заданного максимального количества ретрансляций (хопов) от отправителя.
EmberStatus emberSendMulticast (EmberApsFrame * apsFrame, int8u radius, int8u nonmemberRadius, EmberMessageBuffer message)	Отправка группового сообщения в соответствии со спецификацией ZigBee PRO. Сообщение доставляется всем узлам, имеющим заданный идентификатор группы и находящимся в пределах заданного максимального количества ретрансляций (хопов) от отправителя.
EmberMessageBuffer emberFillLinkedBuffers (int8u * contents, int8u length)	Выделяет цепочку буферов стека, достаточную для размещения данных длиной length байтов, и заполняет их данными, переданными с помощью указателя contents. Возвращает указатель на первый буфер в цепочке выделенных буферов стека.

Отладка программы

После создания текста программы код приложения компилируется совместно с библиотекой EmberZNet в среде разработки IAR и загружается в кристаллы EM351/EM357 при помощи устройства InSight Adapter [1].

Следует обратить внимание, что при проектировании ZigBee-сети ведется одновременная разработка кода для всех используемых типов узлов: координатора, роутеров и конечных устройств. Разумеется, и отладка этих приложений должна выполняться совместно. Для этого удобно использовать отладочные средства Ember, InSight Adapter и InSight Desktop, о которых говорилось в первой части статьи [1]. В качестве отладочных плат можно использовать платы отладочного комплекта компании Telegesis ETRX3DVKA (рис. 3), в состав которого входят: 8 съемных радиомодулей, выполненных на базе кристаллов EM351/EM357, с различным типом используемой антенны и максимальной выходной мощностью, три отладочные платы с разъемом для присоединения съемных радиомодулей, а также ZigBee-шлюз ETRX2USB. Как съемные радиомодули, так и шлюз имеют разъемы для внутрисхемного программирования.

Рис. 3. Отладочный комплект для реализации и отладки беспроводной сети ZigBee на базе встраиваемых радиомодулей серии ETRX3 компании Telegesis

Компания Telegesis является партнером Ember по разработке недорогих отладочных комплектов и радиомодулей ETRX351/357.

В заключение хочется отметить, что на базе платформы Ember уже выполнены сотни успешных проектов по всему миру, что позволило компании в августе этого года занять 16-е место в списке наиболее активно развивающихся предприятий США в категории разработки аппаратного обеспечения.

Литература

1. Солодунов С. Разработка программного обеспечения для сетей ZigBee на базе библиотеки EmberZNet. Часть 1 // Беспроводные технологии. 2010. № 3.
2. EmberZNet4.3.0/em35x/documentation/afv2-index.htm
3. EmberZNet4.3.0/em35x/documentation/120-3022-000_API_EM35x/index.htm