Средства разработки Ember

для быстрой реализации проектов ZigBee

Данная статья посвящена описанию процесса разработки программного обеспечения для ZigBee-устройств, выполненных на базе микросхем Ember, с использованием новой версии библиотеки исходных кодов Application Framework V2 (AFV2) и программы Application Builder [1, 2].

Сергей Солодунов
sys@efo.ru

Получившая широкое распространение сетевая технология ZigBee продолжает свой активный рост на рынке беспроводных устройств. Недавно проведенные исследования альянса ZigBee показали, что, несмотря на мировой экономический кризис и связанные с ним последствия, продажи устройств, использующих ZigBee, неуклонно росли с 2007 г. в среднем на 62% в год [3]. В данный момент рынок насчитывает около 350 компаний, разрабатывающих или предлагающих готовую продукцию, использующую эту беспроводную технологию. Общий оборот этих компаний по данному направлению составляет около $1 трлн долларов. Во многом успех технологии связан с большой работой производителей микросхем и альянса по реализации и совершенствованию стека ZigBee.

Компания Ember, занимающая лидирующие позиции среди производителей микросхем, предлагает комплексное решение для разработки сетей ZigBee, включающее высокоинтегрированные микропотребляющие системы на кристалле EM250/256/EM351/EM357, профессиональное встраиваемое программное обеспечение, а также мощные аппаратные и программные средства разработки и отладки проектов ZigBee.

Кластеры и профили ZigBee

Для обеспечения возможности совместной работы устройств различных производителей в одной беспроводной сети альянс ZigBee разрабатывает стандартные профили приложения, которые регламентируют правила взаимодействия беспроводных устройств одного назначения. Достижение совместимости устройств на прикладном уровне оказалось возможным благодаря использованию стандартных типов сообщений, стандартных типов устройств, а также стандартных для этих устройств команд и атрибутов.

Набор связанных команд и атрибутов, определяющих свойства объекта и методы работы с ним, получил название кластер. Например, для стандартного устройства «лампочка» определен кластер **On/Off** («включить/выключить»), содержащий следующие атрибуты и команды:

- стандартный атрибут — состояние («горит/не горит»);
- стандартные команды — «включить», «выключить», «изменить состояние на противоположное».

Таким образом, кластер разделен на две части — серверную и клиентскую. Они находятся на разных устройствах: сервером является узел, который хранит в своей памяти значение атрибута (в нашем случае это лампочка), а клиентом — устройство, которое генерирует команду на изменение или считывание атрибута (в нашем примере это выключатель). Все ZigBee-кластеры перечислены и описаны в библиотеке кластеров ZigBee Cluster Library (ZCL) [4].

Стандартный профиль приложения ZigBee описывает стандартные устройства определенного назначения и указывает обязательные и опциональные кластеры, которые эти устройства реализуют.

Для разработки устройств, реализующих стандартные профили ZigBee, компания Ember предлагает программу — генератор конфигурационных файлов Application Builder [1, 2]. Эта программа, используя исходные коды каркаса приложений Application Framework из пакета EmberZNet, определяет, какие порции исходных кодов будут входить в проект для реализации требуемых стандартных операций, а какие нет. При помощи простого и удобного интерфейса можно задать тип устройства, профиль приложения, конфигурационные параметры стека, выбрать необходимые кластеры из библиотеки кластеров и сконфигурировать приемопередатчик Ember. Результатом работы программы Application Builder являются конфигурационные заголовочные файлы и файлы проекта для компилятора, соответствующего используемой аппаратной платформе. Программа поддерживает генерацию проектов для среды разработки xIDE (в случае использования кристаллов Em250) и для среды IAR (в случае использования кристаллов EM260/EM351/EM357).

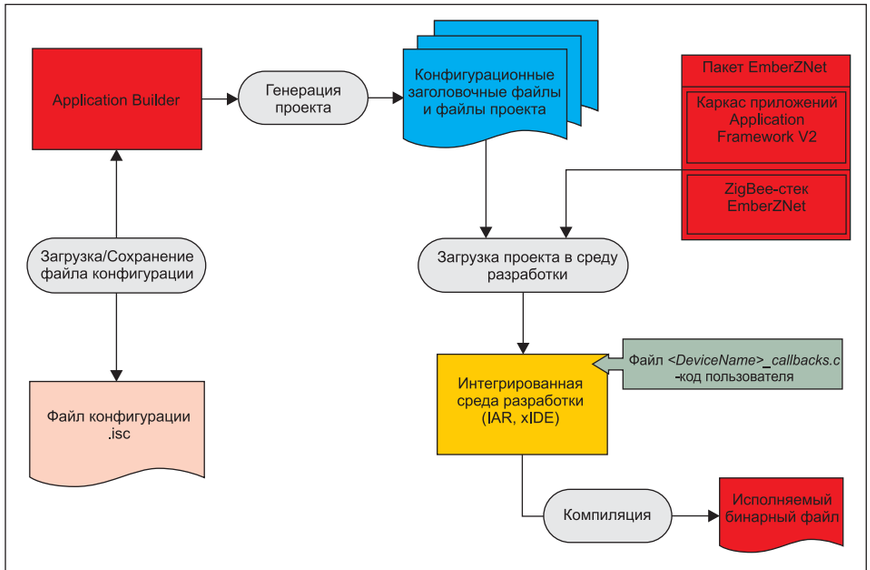

Рис. 1. Процесс создания ZigBee-приложения на базе каркаса приложения Application Framework V2

Процесс разработки приложения с использованием программы Application Builder

Процесс разработки приложения с использованием программы Application Builder на базе платформы Ember (стек ZigBee EmberZNet + каркас приложений Application Framework V2) показан на рис. 1.

Первым этапом разработки является задание конфигурации будущего проекта в программе Application Builder. Основное окно программы (рис. 2) содержит шесть вкладок, каждая из которых позволяет производить настройки определенного уровня:

- **ZCL cluster configuration** — настройки уровня приложения (тип ZigBee-устройства, кластеры, атрибуты);
- **Stack configuration** — конфигурация стека (параметры, определяющие политику безопасности устройства, настройки радиоинтерфейса физического уровня, установки конечной точки, временные интервалы энергосберегающих режимов для спящих устройств);

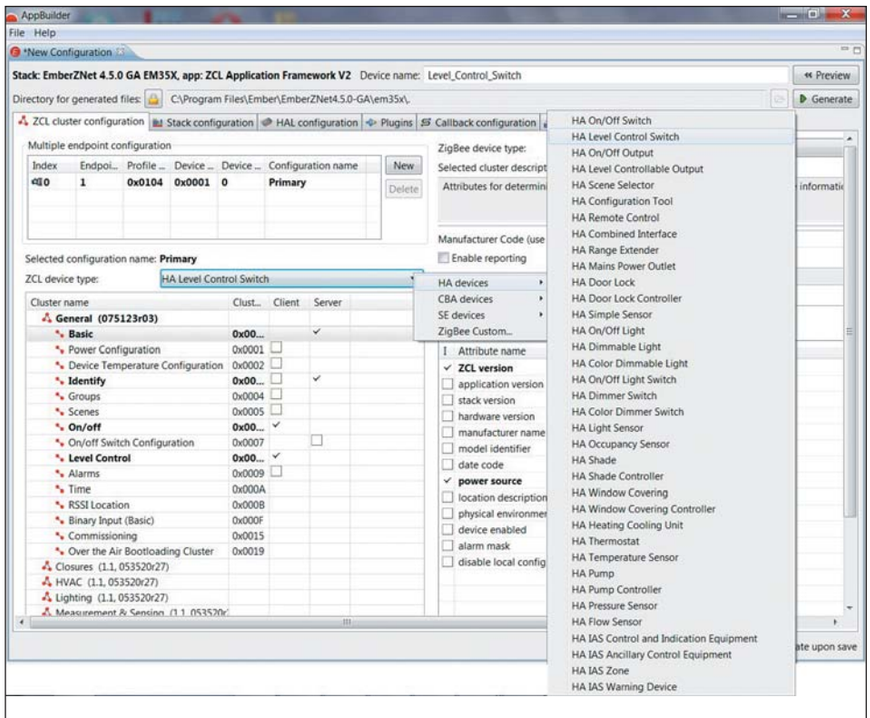

Рис. 2. Конфигурация проекта в программе Application Builder

- **HAL configuration** — настройки уровня аппаратной абстракции стека EmberZNet (выбор микросхемы приемопередатчика, для которой разрабатывается проект, конфигурирование входов и выходов, режим работы последовательного порта, выбор набора команд интерфейса командной строки и др.);
- **Plugins** — подключение/отключение дополнительных программных модулей, реализующих функциональность кластеров по умолчанию, и другие полезные программные операции;
- **Callback configuration** — подключение обратных функций каркаса Application Framework, функциональность которых пользователь желает определить в своем приложении;
- **Include configuration** — подключение к проекту дополнительных внешних с-файлов, token-файлов (описывают объекты, предназначенные для хранения в постоянной памяти микросхем Ember), задание макроопределений, событий.

Текущая конфигурация и настройки, заданные в программе Application Builder, могут быть сохранены в конфигурационном файле с расширением *.isc*. Таким образом, при желании их можно загрузить в Application Builder и вновь использовать. После того как все конфигурационные параметры определены, посредством нажатия кнопки **Generate** создаются следующие файлы проекта:

- *<DeviceName>_endpoint_configuration.h* — конфигурация статических структур данных AFV2. Позволяет разделять метаданные атрибутов между всеми конечными точками (устройствами, подключенными к данному узлу сети ZigBee), при этом каждая конечная точка может иметь свое собственное пространство памяти для хранения атрибутов. Определения (#defines) из этого файла используются файлом *app/framework/util/attribute-storage.c* для конфигурации всех данных приложения, относящихся к атрибутам.
- *<DeviceName>.h* — основной header-файл приложения. Содержит определения, которые подключают все выбранные во время конфигурации элементы и свойства AFV2.
- *<DeviceName>_callbacks.c* — содержит объявления и реализацию по умолчанию всех callback-функций, выбранных во время конфигурации. Сюда помещается пользовательский код верхнего уровня. Существует также возможность подключать дополнительные пользовательские header- и с-файлы.
- *<DeviceName>_board.h* — файл конфигурации уровня аппаратной абстракции для выбранной платформы. По умолчанию предполагает использование отладочных плат Ember. Конфигурируется в соответствии с настройками вкладки **HAL configuration tab** в программе Application Builder. При необходимости вместо данного файла можно подключать к проекту внешний пользовательский заголовочный файл с расширением *.h*, для этого необходимо выбрать опцию **Use custom board header** и указать путь к внешнему файлу конфигурации в поле **File.**
- *<DeviceName>_tokens.h* — хранит определения элементов данных («токенов»), хранящихся

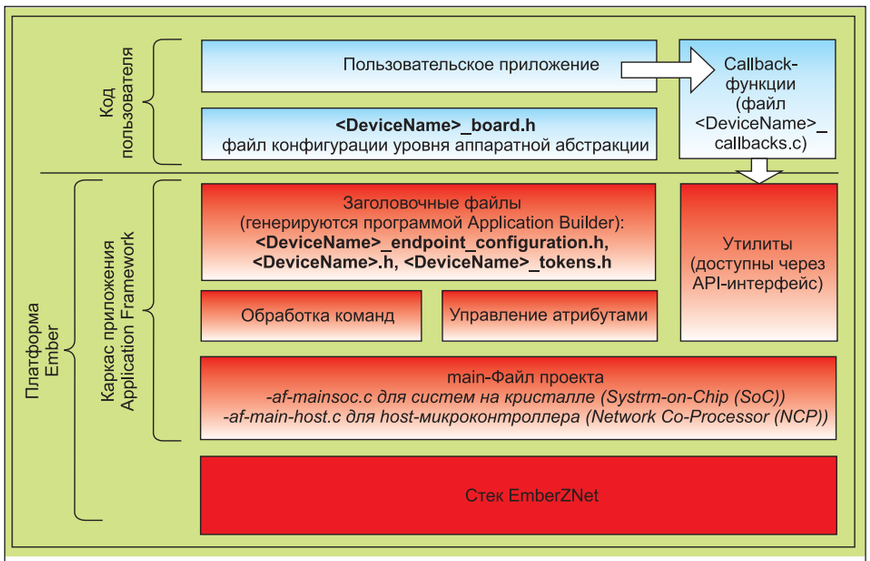

Рис. 3. Структура проекта, выполненного на базе платформы Ember

в постоянной памяти чипов EM250 или EM35x.

- *<DeviceName> .ewp, eww, .xip, .xiw, .mak* — файлы проекта для сред разработки xIDE или IAR, в зависимости от выбранной платформы.

После генерации данных файлов проект можно открыть в соответствующей среде разработки и приступить к добавлению пользовательского кода, относящегося к функциональности разрабатываемого приложения. Далее проект компилируется в исполняемый бинарный файл, который уже можно загружать непосредственно в память программ используемой микросхемы Ember.

Каркас приложения Application Framework V2

Каркас приложения Application Framework версии 2 (AFV2) представляет собой существенное обновление предыдущей версии набора исходных кодов Application Framework V1 (AFV1) [1, 2]. Среди множества прочих преимуществ AFV2 по сравнению AFV1 можно выделить следующие основные:

- Возможность конфигурировать части кода, относящиеся как к реализации стандартных кластеров и профилей, так и к функционированию приложения в целом.
- Высокий уровень абстракции — весь код пользовательского приложения располагается в отдельном файле (*<DeviceName>_callbacks.c*), при этом взаимодействие с каркасом приложения осуществляется через callback-функции, что освобождает разработчика от необходимости разбираться в коде самого каркаса и одновременно защищает последний от непреднамеренного повреждения.
- Поддержка до 239 конечных точек (устройств, подключенных к одному радиоузлу сети ZigBee), возможность создания пользовательских кластеров с помощью XML-файлов, поддержка tokenised-атрибутов (хранение ZCL-атрибутов в энергонезависимой памяти), улучшенный интерфейс командной строки.

Структура и состав Application Framework V2

На рис. 3 показана структура проекта, выполненного на базе платформы Ember. В иерархии проекта каркас приложения AFV2 располагается над стеком EmberZNet, взаимодействуя с ним посредством API-функций и handler-функций стека. При этом на уровень приложения AFV2 предоставляет пользователю свой более абстрактный и специализированный интерфейс. В то время как стек EmberZNet реализует все основные операции в сети ZigBee PRO, такие как образование сети, присоединение к сети, ретрансляция данных и прочее, AFV2 выполняет все операции, связанные с функционированием ZigBee-кластеров: формирование, обработка стандартных кластерных команд (или ZCL-команд), запись, изменение и чтение атрибутов.

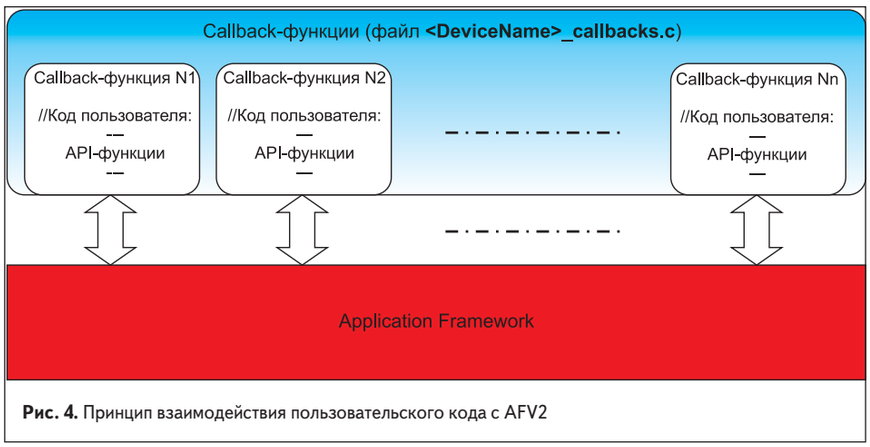

Рис. 4. Принцип взаимодействия пользовательского кода с AFV2

Принципы разработки пользовательского кода. Интерфейс пользователя — API- и Callback-функции

Одна из главных идей создания AFV2 — это разделение пользовательского кода и кода, предоставляемого Ember в качестве каркаса для разрабатываемого проекта. Согласно этому, весь пользовательский код должен располагаться вне каркаса приложения, взаимодействуя при этом с ним с помощью callback-функций, определенных в отдельном файле (*<DeviceName>_callbacks.c*), из которых программист может вызывать API-функции утилит, реализованных в AFV2. Такой принцип взаимодействия пользовательского кода с AFV2 иллюстрирует рис. 4.

Как уже упоминалось ранее, набор реализуемых пользователем callback-функций выбирается в процессе конфигурации проекта в программе Application Builder (вкладка **Callback configuration**).

Все callback-функции в Application Builder разделены на две группы:

- ***Cluster-Specific Command Handling Callbacks*** (кластерные) — callback-функции, связанные с функционированием кластеров. Определяют реакцию кластеров на полученные устройством ZCL-команды, операции, выполняемые кластерами циклически, при инициализации, при изменении значения того или иного атрибута. В большинстве случаев существует строгое соответствие между ZCL-командами и кластерными callback-функциями. Примеры кластерных callback-функций:
  - ***void emberAfOnOffClusterOnCallback (void)*** — вызывается AFV2 при получении команды **On** кластера **On/Off**;
  - ***void emberAfOnOffClusterServerAttributeChangedCallback(int8u endpoint, EmberAfAttributeId attributeId)*** — вызывается AFV2 при изменении значения серверного атрибута ***attributeId*** кластера **On/Off** конечной точки ***endpoint***.
- ***Non Cluster-Related Callbacks*** (общего назначения) — callback-функции, которые вызываются из Application Framework в тех местах, где пользователь может пожелать получить какую-либо информацию о текущих операциях, выполняемых Application Framework, или добавить свой код. Все функции этой группы описаны в файле *tool/appbuilder/callbacks.xml*. Пример callback-функции общего назначения:
  - ***boolean emberAfPreCommandReceivedCallback(EmberAfClusterCommand* cmd,***

***boolean** isInterpan)* — вызывается каркасом AFV2 при получении ZCL-команды, до того как она будет соответствующим образом обработана и выполнена кодом AFV2. Функция возвращает логическое значение, указывающее, была ли данная команда обработана приложением пользователя, или нет. Если сообщение не было обработано пользователем, то оно подлежит дальнейшей обработке каркасом AFV2.

В таблице 1 приведен список наиболее важных callback-функций, которые пользователь может определить в своем приложении.

Контроль и управление операциями AFV2 со стороны пользовательского кода осуществляется с помощью API-функций. Определения всех API-функций AFV2 содержатся в файле *app/framework/include/af.h*. Множество из этих функций повторяют или расширяют API-функции стека EmberZNet, делая их более абстрактными и удобными для использования конечным разработчиком. В то же время AFV2 предлагает пользователю большое количество функций, выполняющих более специфические операции верхнего уровня, такие как формирование, прием и отправка по сети ZigBee стандартных ZCL-команд, операции над атрибутами кластеров, а также многие другие полезные утилиты. Все API-функции AFV2, предназначенные для использования разработчиком, имеют префикс 'emberAf'.

Необходимо отметить, что в связи с тем, что в новой версии Application Framework реализована поддержка до 239 конечных точек, большинство функций по работе с кластерами теперь, в качестве дополнительного аргумента, принимают параметр, указывающий идентификатор конечной точки (endpointId), которой адресовано действие данной функции. Это, в частности, относится к функциям инициализации и управления кластерами, циклически выполняемым функциям кластеров, а также функциям управления атрибутами.

Для получения информации о конечных точках и атрибутах приложения предлагаются функции, определения которых содержатся в файле *app/framework/util/attribute-storage.h*. Например, чтобы узнать, содержит ли конечная точка определенный атрибут, можно использовать функцию ***boolean emberAfContainsAttribute**(**int8u** endpoint, **ClusterId** clusterId, **AttributeId** attributeId)*. Она возвращает булеву переменную, которая указывает, реализован ли запрашиваемый кластер и атрибут для данной конечной точки.

Отправка и прием ZCL-команд

Пожалуй, наиболее интересными с точки зрения разработки ZigBee-приложения, реализующего стандартные профили приложения, являются API- и callback-функции AFV2, предназначенные для отправки, приема и обработки стандартных ZCL-команд.

Для отправки по сети ZigBee стандартной ZCL-команды необходимо выполнить следующие действия:

- Сформировать ZCL-команду, используя функцию заполнения буфера команд, соответствующую отсылаемой команде, из набора готовых макрофункций, определенных в файле *app/framework/gen/client-command-macro.h*. Данные макрофункции заполняют буфер команд значениями, соответствующими отправляемой команде. Например, для формирования команды ***Identify*** кластера ***Identify***, представляющей собой запрос на идентификацию удаленного узла сети ZigBee, необходимо вызвать функцию ***emberAfFillCommandIdentifyClusterIdentify(identifyTime)***. В качества аргумента в эту функцию передается параметр ***identifyTime*** команды ***Identify***, который указывает время в секундах, в течение которого, согласно описанию команды, узел-получатель должен идентифицировать себя. Сам процесс идентификации и соответствующий ему код определяются пользователем (например, мигание светодиодом или звуковой сигнал).
- Получить с помощью API-функции ***EmberApsFrame* emberAfGetCommandApsFrame(void)*** указатель на структуру ***EmberApsFrame***, отображающую APS-заголовок ZigBee-сообщения, и заполнить поля **source endpoint** и **destination endpoints** соответственно значениями конечной точки отправителя и конечной точки получателя для отправляемой команды. Остальные поля в структуре ***EmberApsFrame*** (такие как порядковый номер) автоматически заполняются кодом каркаса AFV2.
- Отправить сформированную команду в ZigBee-сообщении с помощью одной из API-функций отправки. Команда может быть отправлена в адресном, широковещательном или групповом ZigBee-сообщении с помощью соответствующих функций:
  - ***EmberStatus emberAfSendCommandMulticast(int16u** multicastId)**;***
  - ***EmberStatus emberAfSendCommandUnicast(EmberOutgoingMessageType** type**, int16u** indexOrDestination)**;***
  - ***EmberStatus emberAfSendCommandBroadcast(int16u** destination)**;***

Здесь необходимо отметить, что помимо стандартных ZCL-команд каркас AFV2 также позволяет отправлять и произвольные ZigBee-сообщения, содержащие данные любого формата. Для этого применяются API-функции отправки, определенные в файле *app/framework/include/af.h*. Такая возможность может быть особенно интересна разработчикам, не использующим стандартные профили приложения.

Теперь рассмотрим процесс обработки ZCL-команд на принимающем узле (рис. 5). Все входящие ZigBee-сообщения (в том числе сообщения произвольного формата, не соответствующие стандартным ZCL-командам) автоматически передаются стеком в handler-функцию ***emberIncomingMessageHandler()***, которая определена в main-файле проекта. В случае получения стандартной ZCL-команды ***emberIncomingMessageHandler()*** вызывает обработчик команд AFV2 ***emberAfProcessMessage()***, который находится в файле *app/framework/util/util.c*. Последний заполняет структуру типа ***EmberAfClusterCommand***, которая хранит все данные, относящиеся к полученной по радиоканалу команде. Теперь глобальный указатель на эту структуру, ***emAfCurrentCommand***, доступен из любой функции в процессе обработки команды (для получения этого указателя используется специальная функция ***emberAfCurrentCommand()***). Далее обработчик команд вызывает callback-функцию

Таблица 1. Список наиболее важных callback-функций AFV2-приложения

Callback-функция	Описание
void emberAfMainInitCallback(void) {}	Вызывается в начале выполнения программы из главного файла проекта. Дает приложению пользователя возможность выполнить какие-либо необходимые инициализирующие операции.
void emberAfClusterInitCallback (int8u endpoint, int16u clusterId) {}	Вызывается единожды для каждого кластера, реализованного в устройстве, в начале выполнения программы. Позволяет при необходимости выполнить какие-либо инициализационные операции для каждого кластера.
void mainTickCallback(void) {}	Циклически вызывается из главного цикла программы. Весь код, который в предыдущей версии AFV размещался в функции appTick(), теперь размещается в этой функции. Эта callback-функция может быть использована для обслуживания конечного автомата. Например, если приложение оперирует состояниями joined или commissioned, они должны отслеживаться и устанавливаться в этой функции.
boolean emberAfPreGoToSleepCallback (int32u sleepDurationAttempt) {return FALSE;}	Вызывается перед переходом устройства в режим пониженного энергопотребления (сна). Функция позволяет приложению контролировать, может ли устройство перейти в режим сна на промежуток времени *sleepDurationAttempt*. Возвращаемое значение TRUE сигнализирует, что попытка перехода в сон была отменена. Значение FALSE сигнализирует о том, что устройство может переходить в сон.
void emberAfPostWakeUpCallback(int32u sleepDuration) {}	Вызывается сразу после выхода устройства из режима сна. Аргумент *sleepDuration* указывает время, в течение которого устройство находилось в спящем режиме.

emberAfPreCommandReceivedCallback(), которая предоставляет пользователю возможность самостоятельно обработать принятую команду. Эта функция возвращает логическое значение, указывающее, была ли данная команда обработана приложением пользователя или нет. По умолчанию она возвращает значение FALSE, что говорит о том, что сообщение не было обработано пользователем и подлежит дальнейшей обработке каркасом AFV2. В последнем случае, если принятая команда является глобальной (чтение или запись атрибута), обработчик команд передает ее в функцию обработки глобальных команд ***emAfProcessGlobalCommand()*** (определена в файле *process-global-message.c*). В противном случае он передает ее в обработчик специфических кластерных команд ***emAfProcessClusterSpecificCommand()*** (определена в файле *process-cluster-message.c*). Далее получивший управление обработчик, в зависимости от того, как были определены настройки **Plugins** и **Callback_configuration** в процессе конфигурации проекта, вызывает либо непосредственно функцию реализации принятой команды по умолчанию, либо соответствующую определяемую пользователем callback-функцию.

Значительная часть кода обработки команд генерируется программой AppBuilder из XML-файлов, хранящихся в папке *tool/appbuilder*. Весь код, генерируемый из XML-файлов, помещается в папку *app/framework/gen*. Руководство по созданию пользовательских кластеров и генерации кода обработки соответствующих команд можно найти в справочнике программы Application Builder (меню **Help–>Help Contents**).

Согласно спецификации ZCL, в случае адресного сообщения после выполнения ZCL-команды узел должен отправить соответствующий отклик отправителю. Для команд, обработанных по умолчанию, такой отклик отправляется каркасом AFV2 автоматически. Если же команда обрабатывается через callback-функцию, то для нее пользователь должен самостоятельно отправить отклик. Так как отклик представляет собой стандартную ZCL-команду, то и отправляется он тем же самым способом, который был описан ранее. Для формирования буфера команд для откликов на стандартные ZCL-команды предлагается набор макрофункций, определенных в файле *app/framework/gen/*

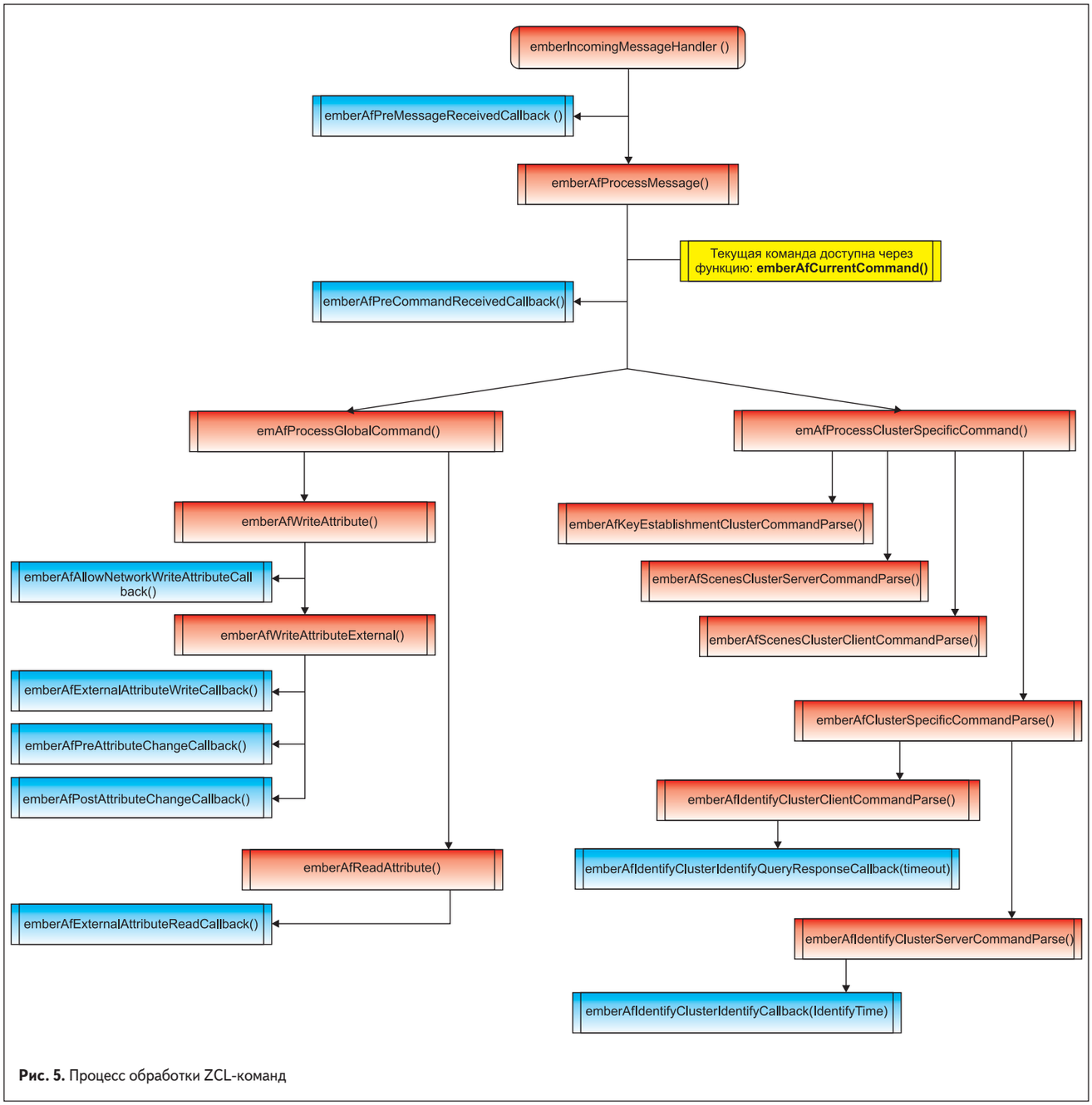

Рис. 5. Процесс обработки ZCL-команд

clientcommand-macro.h. Например, отклик для команды ***Identify*** кластера **Identify** формируется с помощью функции ***emberAfFillCommandIdentifyClusterIdentifyQueryResponse (timeout)***, где аргумент ***timeout*** — это параметр команды-отклика, который указывает время в секундах, в течение которого узел будет себя идентифицировать.

Полное описание всех API- и callback-функций каркаса приложений Application Framework V2 можно найти в html-руководстве Ember Application Framework API reference, входящем в состав документации к пакету EmberZNet [5].

Плагины

Как уже упоминалось ранее, AFV2 содержит широкий набор программных модулей, или плагинов, которые реализуют функциональность всех стандартных ZigBee-кластеров в соответствии со спецификацией ZCL и описаниями публичных профилей приложения. У разработчика есть возможность подключать/отключать использование того или иного плагина в своем приложении во время конфигурации проекта в программе Application Builder (вкладка **Plugins**).

Если плагин подключен, то это означает, что он реализует определенный набор callback-функций, относящихся к задачам, выполняемым этим плагином. Описание каждого плагина и список выполняемых им callback-функций приводятся во вкладке **Plugins** программы Application Bulder. В списке всех доступных callback-функций (в левом окошке вкладки **Callback_configuration**) callback-функции, реализованные тем или иным подключенным плагином, отмечаются специальным отличительным значком  , расположенным слева от имени функции. Назначение каждой callback-функции кратко описано в комментариях перед объявлением функции в правом окошке вкладки **Callback_configuration.**

Таким образом, подключив плагин, реализующий функциональность определенного кластера, пользователь может не заботиться об обработке команд и выполнении соответствующих этим командам операций над атрибутами данного кластера. Разработчику конечного приложения остается только лишь анализировать значения этих атрибутов, их изменение, при этом фокусируясь на более специфических задачах приложения. Например, если разрабатываемое устройство включает в себя серверную часть кластера **Level Control** («управление уровнем»), то при подключении плагина **Level control cluster** все команды, принятые от клиентской части данного кластера, будут обрабатываться им автоматически. Для кластера **Level Control** это такие команды, как ***Move to Level*** (непрерывно увеличивать значение атрибута «уровень» до установленного значения в течение заданного времени), ***Move ()*** (уменьшать или увеличивать значение атрибута «уровень» до его минимального/максимального предельного значения с заданным шагом изменения) и др. То есть, в данном случае задача кода пользователя заключается только в мониторинге значения переменной-атрибута **Level** и ее программной или физической интерпретации в соответствии с функциональным назначением устройства (например, это может быть код управления генератором ШИМ-сигнала в соответствии с текущим значением атрибута **Level**).

Хочется добавить, что набор предлагаемых плагинов постоянно совершенствуется и пополняется. Так, с выходом пакета EmberZNet версии 4.3 в AFV2 в этот набор были добавлены плагины, реализующие функциональность кластеров нового профиля ZigBee Smart Energy 1.1. Пожалуй, наиболее интересные из них — **Over-the-Air Upgrade (OTA)** и **Tunneling.** Первый плагин предоставляет все необходимое программное обеспечение, позволяющее стандартным образом, по радиоканалу, производить обновление программного обеспечения ZigBee-устройств на объекте. Он представляет собой набор программных модулей, обеспечивающих выполнение операции как клиентской, так и серверной части кластера **Over the Air Bootloading Cluster** для всех аппаратных платформ Ember (EM250,EM260,EM351,EM357). Плагин **Tunneling** реализует функциональность одноименного кластера, позволяющего организовывать прозрачный канал между любыми двумя устройствами в сети ZigBee, обеспечивая тем самым возможность инкапсуляции различных, в том числе сложных, протоколов обмена данных со счетчиками в стандартные ZigBee-пакеты.

Управление событиями

Еще одно важное нововведение в AFV2 — это появление системы управления событиями, с помощью которой централизованно управляются и контролируются все периодические задачи, выполняемые стеком, каркасом AFV2 и пользовательским кодом. Механизм управления событиями позволяет сократить объем кода и расходуемой RAM, а также обеспечивает более эффективное использование временных ресурсов спящих устройств ZigBee, тем самым снижая энергопотребление последних.

С помощью данного механизма все периодические операции (события), выполняемые устройством, могут быть запущены (активированы) или остановлены (деактивированы) на уровне приложения, например на основании вводимых пользователем данных, полученных по радиоканалу команд или в процессе инициализации устройства.

Различаются два типа событий AFV2: пользовательские и кластерные. Пользовательские события создаются пользователем и могут быть использованы для любых целей в пределах приложения. Кластерные события непосредственно связаны с реализацией функциональности кластеров в плагинах AFV2.

Пользовательские события

Определение пользовательского события предполагает создание двух элементов: функции, которая вызывается менеджером событий по наступлении времени выполнения события, и структуры ***EmberEventControl***, используемой в качестве указателя на данное событие при его планировании (задание периода выполнения, активации/деактивации и т. п.).

Программа Application Builder предоставляет удобный интерфейс пользователя для создания пользовательских событий. Чтобы добавить в приложение новое событие, необходимо открыть вкладку **Includes** конфигурации проекта и рядом с таблицей **Custom Events** нажать на кнопку **New**. В результате в таблице появится новая запись, в которой необходимо задать имя события и соответствующей ему функции. В процессе генерации проекта это событие будет автоматически добавлено в список (таблицу) событий AFV2, а в файл *<DeviceName>_callbacks.c* будет сгенерировано объявление самой функции и пустая заготовка для нее.

Необходимо отметить, что таблица событий AFV2 создается во время компиляции и является статической. То есть события не могут быть добавлены во время выполнения программы, они должны быть заранее определены, а пользовательский код в свою очередь имеет возможность управлять расписанием их выполнения. Таким образом, в любой момент времени событие может быть либо активированным, т. е. ожидающим вызова по заданному расписанию, либо деактивированным, т. е. ожидающим задания периода выполнения и активации. Имеется возможность создавать пользовательские таблицы событий, а также объединять события в задачи [5].

Для управления пользовательскими событиями AFV2 предлагает специальный набор API-функций, наиболее важные из которых перечислены в таблице 2.

Таблица 2. API-функции управления событиями AFV2

API-функция	Описание
void emberEventControlSetActive(EmberEventControl someEvent);	Активирует событие *someEvent*
void emberEventControlSetInactive(EmberEventControl someEvent);	Деактивирует событие *someEvent*
void emberEventControlSetDelayMS(EmberEventControl someEvent, int16u delay)	Задает период *delay* выполнения события *someEvent* в миллисекундах
void emberEventControlSetDelayQS(EmberEventControl someEvent, int16u delay)	Задает период *delay* выполнения события *someEvent* в четвертьсекундах
void emberEventControlSetDelayMinutes(EmberEventControl someEvent, int16u delay)	Задает период *delay* выполнения события *someEvent* в минутах
int32u emberAfMsToNextEvent (int32u maxMs)	Возвращает время в миллисекундах, оставшееся до выполнения следующего запланированного события из списка событий AFV2 в течение ближайших *maxMs*. Если в течение этого времени нет запланированных событий, то возвращается значение *maxMs*

Кластерные события

Для каждого реализуемого в устройстве кластера в приложении AFV2 создается клиентская и серверная циклическая callback-функция, или кластерное событие, которое предназначено для выполнения каких-либо периодических задач клиентской и серверной части кластера соответственно. Программа Application Builder генерирует таблицу таких событий для всех используемых кластеров всех конечных точек устройства. Эта таблица генерируется в заголовочный файл *<DeviceName>_endpoint_config.h*, которым в свою очередь оперирует менеджер событий Application Framework. Исходный код последнего содержится в файле *app/framework/util/af-event.c*.

Управление кластерными событиями может осуществляться из соответствующих плагинов либо из пользовательского кода при помощи двух предназначенных для этого API-функций:

- ***EmberStatus emberAfScheduleClusterTick(int8u** endpoint, **int16u** clusterId, **boolean** isClient, **int32u** timeMs, **EmberAfEventSleepControl** sleepControl)**;**
- ***EmberStatus emberAfDeactivateClusterTick (int8u** endpoint, **int16u** clusterId, **boolean** isClient)**;**

Первая функция используется для планирования кластерного события внутри механизма событий AFV2. Она предоставляет коду реализации функциональности кластеров доступ ко всем кластерным событиям по идентификатору конечной точки и кластера. С помощью бинарного аргумента *isClient* указывается, к какой части кластера данное событие относится — к клиентской (TRUE) или серверной (FALSE). Аргумент *timeMs* задает период выполнения этого события в миллисекундах. Аргумент *sleepControl* позволяет устанавливать приоритет планируемого кластерного события, определяющий, в каком режиме энергопотребления устройство может находиться, если данное событие активно (т. е. ожидает своего выполнения менеджером событий). Значение этого аргумента относится только к устройствам, поддерживающим режимы пониженного энергопотребления (спящим устройствам), и для устройств других типов не имеет никакого эффекта. Аргумент имеет перечисляемый тип ***EmberAfEventSleepControl***, возможные значения которого перечислены в файле *app/framework/include/af-types.h*:

- EMBER_AF_OK_TO_HIBERNATE — устройство может быть переведено каркасом AFV2 в режим длительного глубокого сна (с самым низким энергопотреблением), пока не наступит время следующего вызова данного события.
- EMBER_AF_OK_TO_NAP — устройство может переводиться каркасом AFV2 в режим кратковременного сна, а в промежутках между такими периодами оно должно пробуждаться (т. е. переходить в обычный рабочий режим) и выполнять опрос своего родительского узла на наличие сообщений для него. Такой приоритет применим, например, к событию, которое представляет собой таймер, срабатывающий через заданное время, в течение которого устройство ожидает какое-либо ответное сообщение от другого устройства в сети.
- EMBER_AF_STAY_AWAKE — каркасу AFV2 запрещено переводить устройство в режимы пониженного энергопотребления, пока данное событие не будет выполнено. Такой приоритет, в частности, имеет смысл применять к событию, которое будет выполняться очень часто и нет необходимости перехода в сон между выполнениями данного события. Дело в том, что логика работы каркаса приложения AFV2 построена таким образом, что после каждого пробуждения из режима сна спящее устройство производит опрос родительского узла, и если это будет происходить слишком часто, то энергопотребление устройства может значительно возрасти. Если же спящее устройство все время удерживается в обычном рабочем режиме, то оно будет выполнять опрос «родителя» каждую секунду.

Возвращаемое значение EMBER_SUCCESS функции ***emberAfScheduleClusterTick()*** информирует приложение о том, что событие было найдено в таблице событий AFV2 и спланировано согласно заданным в функции значениям. В противном случае функция возвращает значение EMBER_BAD_ARGUMENT.

Вторая API-функция, ***emberAfDeactivate ClusterTick()***, позволяет деактивировать любое кластерное событие, указав с помощью передаваемых аргументов идентификатор конечной точки, кластера и к какой части кластера это событие относится (к клиентской или серверной). Данная функция может вызываться из функции самого события, чтобы механизм управления событиями не запустил это событие снова. Эта функция может быть также вызвана в любом другом месте приложения, где требуется деактивировать то или иное событие до его выполнения каркасом AFV2.

Более подробную информацию о механизме управления событиями AFV2 можно найти в документе AFV2 Developer Guide, входящем в состав документации к пакету EmberZNet [5].

Интерфейс командной строки (CLI)

Каркас приложения AFV2 содержит поддержку интерфейса командной строки Command Line Interface (CLI), который обеспечивает возможность управления ZigBee-устройством через последовательный порт. Интерфейс CLI реализует множество команд как общего, так и специфического кластерного назначения. Команды общего назначения позволяют выполнять такие операции, как формирование сети с заданными параметрами, присоединение к уже существующей сети, чтение и запись значений атрибутов кластеров и многие другие. Специфические кластерные команды позволяют формировать и отправлять по сети стандартные ZCL-команды для большинства стандартных кластеров ZigBee.

В отличие от первой версии AFV, где формат вводимых аргументов для каждой команды был строго задан, вторая версия каркаса поддерживает ввод аргументов как в десятичном, так и в шестнадцатеричном формате. Если аргумент имеет префикс ‘0x’, то он будет принят обработчиком интерфейса CLI как значение, переданное в шестнадцатеричном формате, иначе — как значение в десятичном формате.

Кроме того, в новой версии AFV появилась возможность расширять набор команд интерфейса CLI, т. е. дополнять его собственными пользовательскими командами. Для этого используется callback-функция ***ember AfCustomCommandLineCallback ()***, которая вызывается каркасом AFV2 при получении по последовательному каналу команды, начинающейся со слова «custom». Аргументы, переданные с пользовательской командой, могут быть получены с помощью API-функций интерпретатора команд, описание и пример использования которых можно найти в файле *app/util/serial/command-interpreter2.h*.

Заключение

В данной статье были рассмотрены основные возможности, которые предлагает программное обеспечение Ember для быстрой разработки устройств, поддерживающих стандартные профили ZigBee. Было показано, что использование встраиваемой библиотеки EmberZNet и каркаса приложений Application Framework V2 в комплекте с профессиональным программным обеспечением Ember значительно упрощает процесс разработки и отладки устройств ZigBee. Кроме того, значительно упрощается и ускоряется процесс сертификации конечных устройств на соответствие спецификации ZigBee Pro и стандартным профилям приложений ZigBee, поскольку платформа Ember имеет статус Golden Suit (используется сертифицирующими органами в качестве эталона для тестирования устройств на соответствие спецификации ZigBee Pro), а каркас приложений Application Framework V2, в свою очередь, гарантирует формирование и обработку стандартных команд и данных в строгом соответствии с требованиями спецификаций профилей приложений. ■

Литература

1. Солодунов С. Реализация стандартного профиля ZigBee Home Automation на базе платформы Ember. Часть 1. // Беспроводные технологии. 2009. № 3.
2. Солодунов С. Реализация стандартного профиля ZigBee Home Automation на базе платформы Ember. Часть 2. // Беспроводные технологии. 2009. № 4.
3. Heile B. ZigBee Overview Wireless Sensor Networking // Presentation on European ZigBee Developers' Conference. Munich, Germany. May, 2011. http://www.zigbee.org/imwp/download.asp?ContentID=20240
4. ZigBee Cluster Library Specification. May, 2008. www.zigbee.org.
5. EmberZNet4.5.3-GA/em35x/documentation/afv2-index.htm