Реализация стандартного профиля ZigBee “Home Automation”

на базе платформы Ember. Часть 2

Напомним, что в первой части данной статьи, которая была опубликована в предыдущем выпуске журнала «Беспроводные технологии» [1], был сделан краткий обзор механизмов технологии ZigBee, обеспечивающих возможность совместной работы в одной беспроводной сети устройств различных производителей. Были также описаны возможности платформы Ember и программы Application Builder для разработки устройств стандартного ZigBee-профиля Home Automation. Во второй части приводится пример создания проектов с использованием этих программных средств.

Сергей Солодунов
sys@efo.ru

Диммер и пульт управления в стандарте ZigBee

Предположим, нам необходимо разработать на базе платформы EM250 два устройства стандартного профиля Home Automation: источник света с регулируемой яркостью (Dimmable Light) и удаленный пульт управления для этого устройства (Dimmer Switch).

Воспользуемся для этого стандартными аппаратными отладочными средствами Ember. Для каждого устройства нам понадобится набор, состоящий из отладочной платы EM250 Breakout Board и радиомодуля EM250 Radio Communication Module. В качестве инструментов для разработки проектов будем использовать комплект программных средств Ember, включающий в себя библиотеку EmberZNet, программу Application Builder и компилятор xIDE [1]. Для каждого проектируемого устройства будет создаваться отдельный проект.

Таблица 1. Перечень изменений, вносимых в конфигурацию

Опция	Light	Switch
Вкладка ZCL Cluster Configuration		
ZCL Device Type (набор предопределенных ZCL ZigBee-устройств. AppBuilder автоматически заполняет таблицу кластеров в соответствии со спецификацией)	HA Dimmable Light	HA Dimmer Switch
Power Configuration (опциональный кластер)	Откл.	Откл.
Device temperature configuration (опциональный кластер)	Откл.	Откл.
Occupancy sensing (опциональный кластер)	Откл.	Откл.
ZigBee device type (тип устройства в ZigBee-сети)	Coordinator or Router	Router
Вкладка Stack Configuration		
Smart Energy (специфические настройки для устройств, имеющих профиль Smart)	Все значения установить равными нулю	Все значения установить равными нулю
Вкладка HAL Configuration		
Platform	EM250	EM250
Bootloader	Standalone	Standalone
Debug Level	Normal	Normal
Application Serial Port	Port 1	Port 1
Baud-rate	115200	115200
Command Line Interface	Full	Full

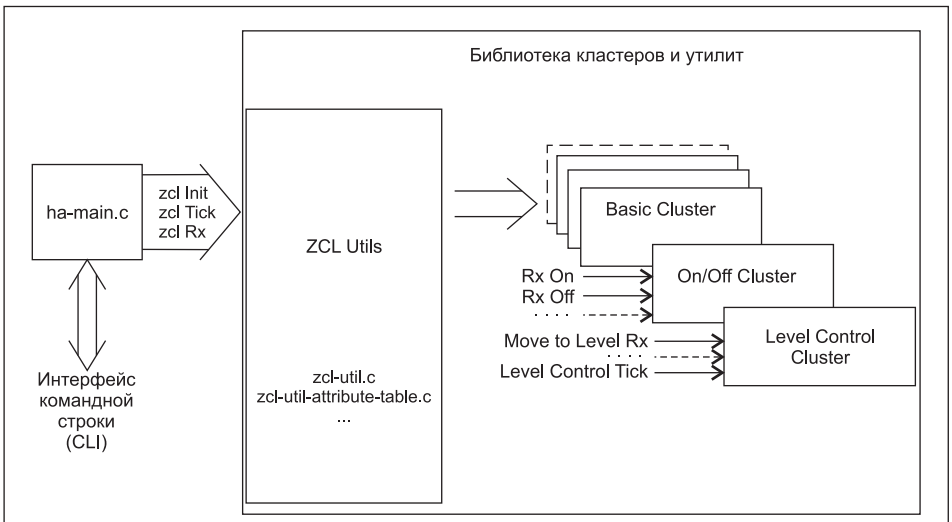

Рис. 1.

Листинг 1

```
while(TRUE) {
halResetWatchdog();      // сброс watchdog таймера
emberTick();             // обработка задач стека
zclTick();               // передача управления
                            кластерам для выполнения
                            периодических задач
appTick();               // выполнение основных
                            задач приложения
                            (переключение режимов работы,
                            опрос состояния датчиков,
                            обработка событий и т. п.)
cliProcessSerialInput(); // обмен данными через
                            последовательный порт
                            (интерфейс командной строки)
}
```

Поскольку оба проекта будут использовать файлы пакета EmberZNet, скопируем для удобства папку, в которую установлен стек, в две заранее созданные папки, например, light и switch.

Для каждого из двух проектируемых устройств следуем по шагам, описанным в первой части данной статьи [1]. Запускаем программу Application Builder. При этом загружается конфигурация нового устройства по умолчанию (New Configuration). В строке ввода имени устройства (device name) изменяем имя ZigBee соответственно на Light или Switch. Указываем путь к папке, в которую будут генерироваться заголовочные файлы и файлы xIDE проекта (Directory of generated files): …light или …switch. В таблице 1 перечислены все изменения, которые необходимо внести в конфигурацию по умолчанию для каждого устройства перед запуском генерации проекта, т. е. перед нажатием кнопки Generate.

После генерации открываем проект каждого устройства в среде xIDE, запуская файлы Light и Switch с расширением .xiw или .xip, сгенерированные Application Builder.

Проект содержит множество связанных между собой c-файлов, заголовочных h-файлов (в их число входят и те файлы, которые были сгенерированы программой AppBuilder) и откомпилированных библиотек. Головным файлом проекта является файл ha-main.c.

Каждому ZigBee-кластеру соответствует свой c-файл, а обращение к функциям, определенным в этих файлах, из верхнего уровня приложения (т. е. из файла ha-mail.c) выполняется по схеме, приведенной на рис. 1.

Как показано на рисунке, доступ к кластерам из приложения можно получить через функции библиотеки утилит (ZCL Utils). Основные функции для работы с кластерами — zclInit, zclTick и zclRx — определены для каждого кластера в отдельности. Но обращение к ним предлагается производить через функции-надстройки, которые определены и находятся в файле zcl-util.c. Доступ к значениям атрибутов всех кластеров осуществляется с помощью функций, определенных в файле zcl-util-attribute-table.c.

С помощью функции zclInit производится инициализация кластера, т. е. установка начальных значений его атрибутов.

Функция zclTick предназначена для выполнения периодических задач кластера, например изменения значения атрибута с определенным интервалом времени.

Функция zclRx представляет собой обработчик определенной zcl-команды кластера, который выполняет операции над атрибутами в соответствии с принятой командой. Т.е. для каждой zcl-команды в кластере определена своя функция zclRx. Например, для кластера «On/Off» их три: «On/Off Rx On», «On/Off Rx Off» и «On/Off Rx Toggle». Для кластера Level Control таких функций больше: «Move to Level Rx», «Move Rx», «Step Rx», «Stop Rx» и др. Подробное описание всех команд для этих и других кластеров находится в библиотеке ZigBee-кластеров (ZCL— ZigBee Cluster Library)[2].

Обработка всех принятых по радиоканалу zcl-команд производится с помощью функции zclUtilProcessMessage(), которая определена в файле zcl-util.c. Она распознает команду и кластер, которому адресована данная команда, и вызывает соответствующую zclRx-функцию. Вызов функции zclUtilProcessMessage() производится из обработчика входящих сообщений emberIncomingMessageHandler(), который, в свою очередь, автоматически вызывается стеком при получении устройством нового ZigBee-пакета по радиоканалу.

Такой централизованный подход снижает избыточность кода, в то же время повышая его читаемость. Проекты, сгенерированные программой Application Builder, содержат поддержку интерфейса командной строки (CLI), который позволяет вводить zcl-команды и получать на них ответы через последовательный канал. Функции, реализующие этот интерфейс, определенны в файле ha-zcl-cli.c.

Основной цикл программы в общем случае выглядит так, как показано в листинге 1.

Если оставить проект без изменений, т. е. без добавления пользовательского кода, и откомпилировать, то устройство, в которое он будет загружен, сможет принимать по радиоканалу стандартные zcl-команды и выполнять соответствующие операции с атрибутами. С помощью интерфейса командной строки можно будет отправлять любые zcl-команды на удаленный ZigBee-узел. Воспользуемся этой возможностью: откомпилируем проект Switch, не внося в него никаких изменений, и загрузим его в один из радиомодулей.

Второй проект Light доработаем таким образом, чтобы значение атрибута Current Level кластера Level Control отображалось яркостью свечения светодиода на отладочной плате. При этом пользовательский код разместим внутри функции appTick.

В качестве источника света для устройства «Dimmable Light» можно использовать светодиод LED2 на отладочной плате Ember. Подавая на него ШИМ-сигнал с различной длительностью импульса, будем изменять яркость его свечения. Длительность импульса в этом случае должна соответствовать текущему значению атрибута Current Level, которое можно прочитать из памяти с помощью функции zclUtilReadAttribute (определена в файле zcl-util-attribute-table.c).

Чтобы мерцание светодиода было незаметно для человеческого глаза, частота ШИМ-сигнала должна быть больше 50Гц.

Итак, добавляем следующие переменные и константы в функцию appTick() (листинг 2):

Листинг 2

```
int16u pwm_period = 15;   // период ШИМ-сигнала
                             (15 мс — 67 Гц)
int8u pwm_width;          // длительность импульса
static int16u last_pulse = 0;//переменная, хранящая время
                             начала текущего периода
int8u currentValue = 0, dataType;//переменная, хранящая
                             текущее значение атрибута
boolean status;
```

Для задач, требующих измерения интервалов времени внутри функции AppTick(), каждый раз вызывается функция библиотеки EmberZNet halCommonGetInt16uMillisecondTick(), которая возвращает текущее значение внутреннего таймера в миллисекундах (листинг 3):

Листинг 3

```
time = halCommonGetInt16uMillisecondTick();
```

После вызова функции halCommonGetInt16 uMillisecondTick() добавляем следующий код (листинг 4):

Листинг 4

```
// Читаем из памяти значение атрибута,
    отображающего значение уровня яркости
status = zclUtilReadAttribute (
            ZCL_LEVEL_CONTROL_CLUSTER_ID,
            ZCL_CURRENT_LEVEL_ATTRIBUTE_ID,
            (int8u*) &currentValue,
            1, &dataType);
// Если произошла ошибка чтения,
    выводим сообщение в последовательный порт
if (status!= ZCL_SUCCESS) {
ZCL_UTIL_DEBUG («Err: can't read current_level_attribute»);
return;
}
// Новое значение длительности импульса равно
    новому значению атрибута
pwm_width = currentValue;
// Зажигаем светодиод в начале периода ШИМ-сигнала
if ((time - last_pulse > = pwm_period) && (pwm_width > 0)){
last_pulse = time;
halSetLed(BOARDLED2);
}
// Гасим светодиод, если длительность импульса истекла
if (time - last_pulse > = pwm_width){
halClearLed(BOARDLED2);
}
```

Компилируем проект и загружаем во второй радиомодуль.

Теперь оба модуля готовы к работе. Подключаем отладочные платы к последовательному порту компьютера. Для каждого модуля отдельно запускаем программу Hyper Terminal и устанавливаем связь. Настройки com-порта устанавливаем следующие:

- скорость (бит/с) — 115 200;
- биты данных — 8;
- четность — нет;
- стоповые биты — 1;
- управление потоком — нет.

При удачном установлении связи нажатие клавиши Enter приводит к появлению указателя на начало строки для ввода Light> (или Switch>, соответственно). Меню печатается в результате выполнения команды help. Оно отображает список всех команд интерфейса CLI, предназначенных для работы с модулем через последовательный порт.

Сначала следует образовать ZigBee-сеть. Координатором сети в нашем случае будет устройство Light. Следующая команда формирует сеть на 11 канале, с мощностью передатчика 2 дБм и идентификатором сети (PAN ID) 00aa (рис. 2):

```
Light> network form 11 2 00aa
```

Теперь с помощью команды pjoin разрешим присоединение к этой сети других ZigBee-устройств на 60 с:

```
Light> network pjoin 60
```

Следующей командой, вводимой на стороне пульта управления, подключаем устройство Switch к созданной сети:

```
Switch> network join 11 2 00aa
```

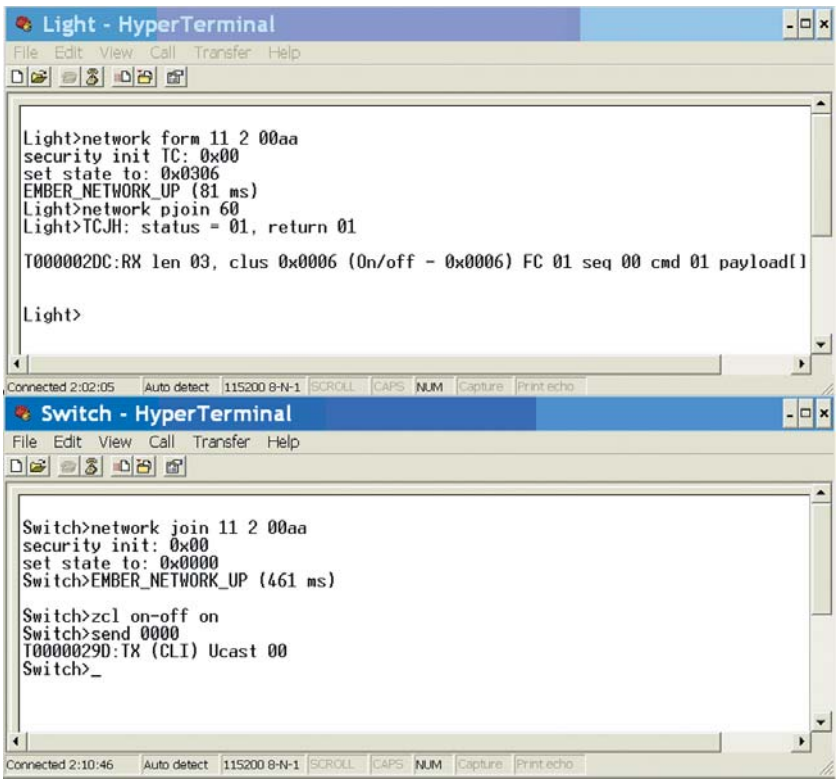

Рис. 2. Управление устройствами Light и Switch по последовательному каналу

Формируем сообщение с командой «включить» кластера вкл/выкл (on-off):

```
Switch> zcl on-off on
```

Передаем сообщение с этой командой на координатор, т. е. на Light:

```
Switch> send 0000
```

В результате выполнения последней команды включается светодиод LED2 на отладочной плате. Аналогично выключаем светодиод:

```
Switch> zcl on-off off
Switch> send 0000
```

Попробуем выполнить более сложную команду, например «плавно зажечь свет» («плавно погасить свет») кластера Level Control:

```
Switch> zcl level-control move 00 01
```

Первый параметр команды move определяет направление изменения уровня (увеличение — 00, уменьшение 01), второй задает шаг изменения уровня.

Передаем сообщение с этой командой на Light:

```
Switch> send 0000
```

В результате выполнения этой команды яркость светодиода плавно увеличивается до максимального значения.

Подробное описание работы с интерфейсом командной строки (CLI) и всех доступных команд содержится в документации к пакету EmberZNet [3].

Интерфейс командной строки включен в проект для отладочных целей. Реальный пульт управления и диммер могут не иметь связи с компьютером. Пользователь в зависимости от задач своего приложения может реализовать собственный алгоритм образования сети и передачи zcl-команд, встроив его в код приложения.

К примеру, нам необходимо, чтобы при нажатии какой-либо кнопки на пульте управления на диммер отправлялась zcl-команда «On» кластера «On/Off».

Функцию, формирующую эту zcl-команду (листинг 5), для удобства разместим в файле ha-zcl-cli.c. Если в будущем понадобится куда-

Листинг 5

```
/* ***********************************************
Данная функция формирует командный (ZCL) буфер
и APS-заголовок ZigBee-сообщения
В случае, если эта функция будет размещаться вне файла
ha-zcl-cli.c, требуется предопределить следующие
переменные и функции:
int8u appZclBuffer[APP_ZCL_BUFFER_SIZE];
int8u* pAppZclBuffer = appZclBuffer;
int8u appZclBufferLen;
EmberApsFrame globalApsFrame;
void zclCliBufferSetup(void);
*********************************************** */
void zclUtilCmdOn(void) {
    //APS-заголовок
    zclCliBufferSetup();
    globalApsFrame.clusterId = ZCL_ON_OFF_CLUSTER_ID;

    //ZCL-буфер
    appZclBuffer[0] = ZCL_CLUSTER_SPECIFIC_COMMAND;
    appZclBuffer[1] = appGetNextZclSeqNum();
    appZclBuffer[2] = ZCL_ON_COMMAND_ID;
    //Размер ZCL-буфера
    appZclBufferLen=3;
}
```

либо ее переместить, то необходимо будет предопределить переменные и функции, указанные в комментариях.

Пусть при нажатии кнопки устанавливается флаг buttonPressed. Тогда будем периодически проверять этот флаг и в случае, если он установлен, отправлять ZigBee-сообщение с zcl-командой «On» на диммер, т. е. на узел с адресом 0x0000. Таким образом, в функцию appTick добавляем следующий код (листинг 6):

Листинг 6

```
If (buttonPressed) {
      zclUtilCmdOn(); // формируем zcl-команду "On"
      //отправляем команду на диммер
      zaAppSendUnicast(EMBER_OUTGOING_DIRECT,
                       0,
                       &globalApsFrame,
                       appZclBufferLen,
                       pAppZclBuffer);
}
```

В результате мы имеем два полностью ZigBee-совместимых стандартных устройства профиля Home Automation. При их создании нам потребовалось добавить код, обрабатывающий текущее значение атрибута Current Level серверной части кластера Level Control в устройстве диммер. Используя это значение, мы управляли яркостью свечения светодиода на отладочной плате Ember. При этом мы не вникали в тонкости реализации этого кластера. Выяснилось также, что формирование и отправка zcl-команд могут быть легко встроены в код приложения, при этом можно воспользоваться готовыми функциями из файла ha-zcl-cli.c.

Заключение

Приведенный пример показал, что платформа Ember позволяет достаточно просто реализовать беспроводную систему устройств домашней автоматизации, поддерживающую стандартный профиль Home Automation. При этом разработчик может, не вдаваясь в подробности реализации задач, связанных с функционированием сети ZigBee и работой ZigBee-кластеров, вести разработку основного алгоритма и программ, реализующих функциональные особенности приложения. Использование в качестве основного инструмента разработки генератора конфигурационных файлов Application Builder не только ускоряет процесс разработки, но также позволяет создавать проекты, которые могут быть быстро сертифицированы на соответствие спецификации ZigBee Pro Feature Set.

Литература

1. Солодунов С. Реализация стандартного профиля ZigBee "Home Automation" на базе платформы Ember. Часть 1 // Беспроводные технологии. 2009. № 3.
2. ZigBee Cluster Library Specification, October 19, 2007, www.zigbee.org
3. EmberZNet3.4.1/em250/app/sampleApps.html