ИНСТИТУТ ТОЧНОЙ МЕХАНИКИ И
ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ
им. С.А. Лебедева
Отдел автономных беспроводных систем

Описание ZigZagAPI

Гекк Максим

Пыптев Сергей

Москва
2007 год

Оглавление

Перечень схем

Перечень таблиц

Листинги

1 Общее описание

Система ZigZag представляет собой коммуникационную среду для обмена сообщениями между прикладными объектами узлов сенсорной сети. На одном узле сети может находится несколько прикладных объектов. Каждый объект идентифицируется уникальным номером. Прикладные объекты предназначены, в частности, для управления датчиками (актуаторами) узла. Система ZigZag предоставляет объектам узла интерфейс прикладного программирования (API), который описывается в данном документе. ZigZag включает в себя операционную систему и стек протоколов ZigBee, а также диспетчер узла. На рисунке 1 представлена общая архитектура системы.

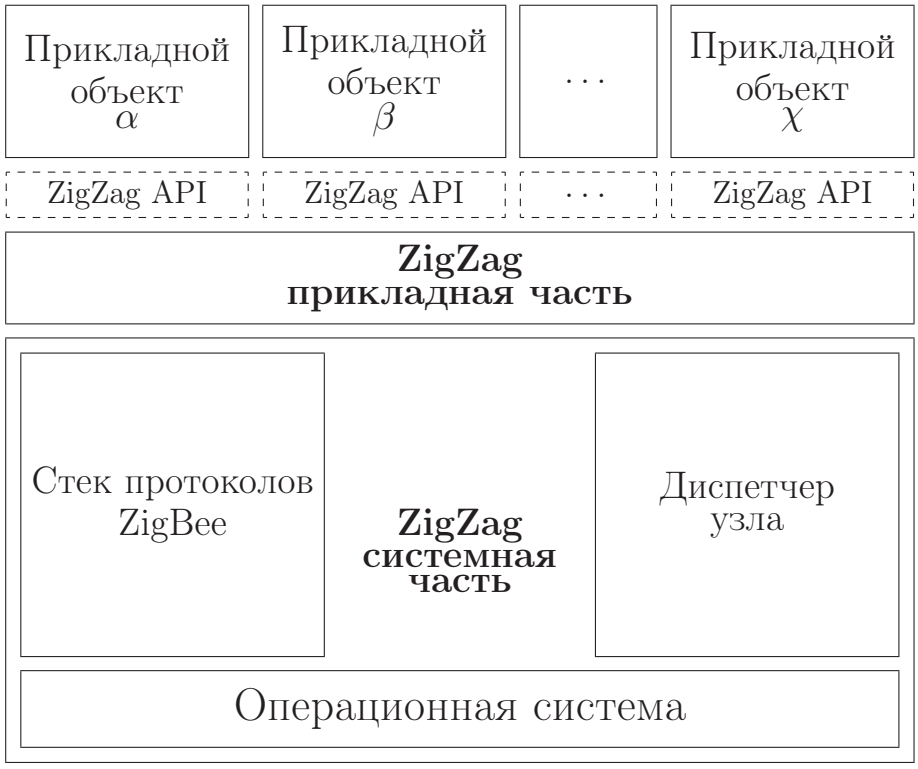

Рис. 1. Общая архитектура системы ZigZag.

Стек протоколов ZigBee позволяет связать узлы в единую сеть. В свою очередь, диспетчер узла предназначен для управления стеком ZigBee.

Поскольку, на одном узле может быть несколько прикладных объектов, то в системе ZigZag используется концепция портов. Каждый объект привязан к определённому порту. Все входящие сообщения перенаправляются прикладным объектам в соответствии с их портами.

Для назначения прикладному объекту номера и порта необходимо перед включением заголовочного файла zigzag.h задать два макроса **OBJ** и **PORT**. См. листинг 1.

Листинг 1. Назначение объекту номера и порта

```
1  #define  OBJ  19
2  #define  PORT  81
3  #include  <zigzag.h>
```

Заголовочный файл zigzag.h должен быть включён только единожды в одном прикладном объекте.

Из вне прикладной объект можно представить в виде набора атрибутов и связанных с ним типов сообщений, которые он может обрабатывать. Система ZigZag позволяет объектам запрашивать и устанавливать атрибуты друг друга.

2 Сервисы операционной среды

2.1 Интерфейс общего назначения

2.1.1 Инициализация прикладного объекта

Функция ***sys_init()*** вызывается системой ZigZag при запуске и предназначена для инициализации внутренних структур данных прикладного объекта. Сигнатура функции представлена в листинге 2.

Листинг 2. ***sys_init()*** - инициализация системы

```
1   void    sys_init() {}
```

Прикладной объект обязан предоставить системе ZigZag реализацию этой функции.

2.1.2 Запрос текущего времени

Функция ***sys_time()*** возвращает локальное время узла в миллисекундах с 1 января 1970 года. Все временные метки, отправляемые в сообщениях, должны быть получены с помощью этой функции. Сигнатура функции представлена в листинге 3.

Листинг 3. ***sys_time()*** - текущее время

```
1   uint64_t    sys_time();
```

2.2 Интерфейс системных событий

С помощью данного интерфейса система ZigZag оповещает прикладной объект о различных событиях, происходящих в системе. Прикладному объекту также предоставляется возможность сгенерировать определённые события. Эта возможность должна использоваться прикладными объектами для завершения обработки прерывания. Самые необходимые действия по обработке прерывания объект может проводить в самой функции обработки прерывания. После этого объект должен сгенерировать системное событие и продолжить выполнение в обработчике события.

2.2.1 Обработка событий

Прикладной объект должен предоставить системе ZigZag реализацию функции ***event_handler()***, сигнатура которой представлена в листинге

4.

Листинг 4. Функция ***event_handler()*** - обработчик событий.

```
1  #include    <zigzag.h>
2  void  event_handler(
3      event_type_t     event_type,
4      unidata_t        unidata
5      );
```

Аргументы функции *event_handler()*:

1). ***event_type*** *- тип события. Системой ZigZag зарезервированы типы событий из диапазона от 0xe0 до 0xff. На данный момент используются события представленные в таблице 1.*

2). ***unidata*** *- аргумент содержит дополнительные данные о событии.*

Таблица 1. Системные типы событий.

Тип события	Номер события	Описание
EV_JOIN	0xff	Узел присоединился к сети
EV_LEAVE	0xfe	Узел покинул сеть
EV_SLEEP	0xfd	Узел переходит в состояние сна. Аргумент **unidata** содержит предполагаемую продолжительность сна в миллисекундах.

Тип события	Номер события	Описание
EV_SFREE	0xfc	Освобождена область памяти хранилища. Аргумент **unidata** содержит дескриптор этой области памяти.
EV_RUNLOCK	0xfb	Завершён доступ на чтение к области памяти хранилища. Аргумент **unidata** равен дескриптору области памяти.
EV_WUNLOCK	0xfa	Завершён доступ на запись к области памяти хранилища. Аргумент **unidata** равен дескриптору области памяти.
EV_AWRITTEN	0xf9	Записано новое значение атрибута. Аргумент **unidata** равен номеру атрибута.

2.2.2 Генерация событий

При помощи вызова функции ***event_emit()*** прикладной объект просит систему ZigZag сгенерировать событие и вызвать обработчик событий ***event_handler()***. При возникновении аппаратного прерывания и после его предварительной обработки объект должен вызвать функцию ***event_emit()*** для продолжения обработки прерывания уже в функции ***event_handler()***. Сигнатура функции ***event_emit()*** представлена в листинге 5.

Листинг 5. Функция ***event_emit()*** - генерация события.

```
1  #include <zigzag.h>
2  result_t event_emit(
3      priority_t      priority,
4      event_type_t    event_type,
5      unidata_t       unidata
```

```
6           );
```

Аргументы функции *event_emit()*:

1). ***priority*** *- приоритет сгенерированного события перед всеми остальными событиями. Чем больше значение аргумента, тем выше приоритет. Допустимые значения аргумента из диапазона от 0 до 127.*

2). ***event_type*** *- тип события. Разрешено использовать типы событий из диапазона от 0x00 до 0xdf.*

3). ***unidata*** *- значение, которое будет передано в соответствующий вызов функции* ***event_handler****.*

2.3 Интерфейс доступа к хранилищу

Хранилище предназначено для сохранения данных о каком-либо событии. Хранилище может использоваться для обмена данными как внутри прикладного объекта, так и между различными объектами, которые при этом могут находиться на разных узлах сети.

2.3.1 Запрос места в хранилище

Для запроса места в хранилище предназначена функция ***storage_alloc()***. Сигнатура этой функции представлена в листинге 6.

Листинг 6. Функция ***storage_alloc()*** - запрос места в хранилище

```
1   #include    <zigzag.h>
2   int32_t storage_alloc(
3       size_t  size
4       );
```

Аргументы функции *storage_alloc()*:

1). ***size*** *- размер запрашиваемой памяти в хранилище.*

Возвращаемое значение:

- *Функция возвращает отрицательное значение в случае ошибки. При этом допустимы следующие значения:*
  - *ENOMEM - недостаточно места в хранилище,*
  - *EINVAL - некорректное значение аргумента* ***size****,*
  - *ENOSYS - функция не поддерживается в текущей версии системы.*
- *Если место в хранилище успешно выделено, возвращается неотрицательный дескриптор. Этот дескриптор впоследствии может быть передан другим прикладным объектам в сети.*

2.3.2 Получение доступа к данным

К одной выделенной области памяти в хранилище могут получить доступ несколько прикладных объектов. При этом в любой момент времени либо несколько объектов могут иметь доступ на чтение, либо только один объект может иметь доступ на запись.

В листинге 7 представлена функция получения доступа.

Листинг 7. Функция ***storage_lock()*** - доступ к хранилищу

```
1  #include    <zigzag.h>
2  void *  storage_lock(
3          int32_t storage_desc,
4          uint8_t  mode
5          );
```

Аргументы функции *storage_lock()*:

1). ***storage_desc*** *- дескриптор области памяти в хранилище.*

2). ***mode*** *- запрашиваемый режим доступа. Допустимы следующие значения:*

- *A_READ - доступ только на чтение,*
- *A_RW - доступ на чтение и запись.*

Возвращаемое значение:

Функция возвращает указатель на область памяти в случае успешного вызова или 0 в противном случае.

После завершения доступа к памяти хранилища должна быть вызвана функция ***storage_unlock()***. Сигнатура функции представлена в листинге 8.

Листинг 8. Функция ***storage_unlock()*** - завершение доступа

```
1  #include    <zigzag.h>
2  result_t  storage_unlock(
3      int32_t storage_desc
4      );
```

Аргументы функции *storage_unlock()*:

1). ***storage_desc*** *- дескриптор области памяти в хранилище.*

Возвращаемые значения:

- *ENOERR - успешно выполнено завершение доступа к хранилищу,*
- *EINVAL - некорректное значение аргумента* ***storage_desc****,*
- *ENOSYS - в текущей реализации функция не поддерживается.*

Заметим, что системы ZigZag извещает прикладные объекты узла о различных событиях, связанных с хранилищем, посредством интерфейса системных событий (см. раздел 2.2).

2.3.3 Высвобождение места в хранилище

Ранее выделенная область памяти может быть освобождена и возвращена в хранилище с помощью вызова функции ***storage_free()*** (см. листинг. 9).

Листинг 9. Функция ***storage_free()*** - возврат памяти в хранилище

```
1  #include    <zigzag.h>
2  result_t  storage_free(
3      int32_t storage_desc
4      );
```

Аргументы функции *storage_free()*:

1). ***storage_desc*** *- дескриптор области памяти в хранилище.*

Возвращаемые значения:

- *ENOERR - успешно выполнено освобождение области памяти в хранилище,*
- *EINVAL - некорректное значение аргумента* ***storage_desc****,*
- *ENOSYS - в текущей реализации функция не поддерживается.*

2.4 Интерфейс синхронных таймеров

Синхронные таймеры предоставляют возможность отсчитывать интервалы времени. В силу того, что функция извещения об истечении

интервала вызывается из задачи, система не может гарантировать определённое отклонение этого вызова от конца интервала. Число доступных синхронных таймеров в системе определяется константой SOFT_TIMER_TOTAL.

2.4.1 Установка таймера

Установить таймер на определённый интервал срабатывания позволяет функция ***stimer_set***. Если таймер уже был установлен, то он переустанавливается на новый интервал. Сигнатура этой функции представлена в листинге 10

Листинг 10. Функция ***stimer_set()*** - установка синхронного таймера.

```
1  #include    <zigzag.h>
2  result_t    stimer_set(
3      const uint8_t tnum,
4      const uint32_t milli_sec
5  );
```

Аргументы функции *stimer_set()*:

1). ***tnum*** *- номер синхронного таймера. Допустимы значения из диапазона от 0 до SOFT_TIMER_TOTAL.*

2). ***milli_sec*** *- длинна интервала в миллисекунда относительно текущего момента времени.*

Возвращаемые значения:

- *ENOERR - синхронный таймер успешно установлен,*
- *EINVAL - некорректное значение аргумента* ***tnum****,*
- *ENOSYS - в текущей реализации функция не поддерживается.*

2.4.2 Срабатывание таймер

По истечении заданного интервала времени система ZigZag вызывает функцию ***stimer_fired***. Сигнатура этой функции представлена в листинге 11.

Листинг 11. Функция ***stimer_fired()*** - срабатывание синхр. таймера.

```
1  #include    <zigzag.h>
2  void     stimer_fired(
3          const uint8_t tnum
4  );
```

Аргументы функции *stimer_fired()*:

1). ***tnum*** *- номер сработавшего синхронного таймера.*

3 Сервисы сетевого взаимодействия

3.1 Интерфейс доступа к атрибутам

3.1.1 Чтение атрибута

Получить текущее значение атрибута можно с помощью функции ***attr_read()***. Сигнатура этой функции представлена в листинге 12

Листинг 12. Функция ***attr_read()*** - чтение атрибута.

```
1  #include    <zigzag.h>
2  result_t attr_read(
3          uint8_t      anum,
4          void         *to
5          );
```

Аргументы функции *attr_read()*:

1). ***anum*** *- номер запрашиваемого атрибута.*

2). ***to*** *- указатель на область памяти, в которую будет записано текущее значение атрибута.*

Возвращаемое значение:

Функция ***attr_read()*** *в случае успешного вызова возвращает неотрицательное значение, равное размеру атрибута. Если вызов был неуспешен, то возвращается одно из следующих отрицательных значений:*

- *EINVAL - некорректный номер атрибута.*
- *EACCESS - нарушен режим доступа к атрибуту.*
- *ENOSYS - в текущей реализации атрибут не доступен.*

3.1.2 Запись атрибута

Запись нового значения атрибута прикладного объекта осуществляется посредством вызова функции ***attr_write()***. Сигнатура этой функции представленав листинге 13.

Листинг 13. Функция ***attr_write()*** - запись атрибута.

```
1  #include    <zigzag.h>
2  result_t attr_write(
3      uint8_t      anum,
4      void         *from
5      );
```

Аргументы функции *attr_write()*:

1). ***anum*** *- номер записываемого атрибута.*

2). ***from*** *- указатель на область памяти, содержащей новое значение атрибута.*

Возвращаемое значение:

Функция ***attr_write()*** *должна возвратить одно из следующих значений:*

- *ENOERR - успешно установлено новое значение атрибута.*
- *EINVAL - некорректный номер атрибута.*
- *EACCESS - нарушен режим доступа к атрибуту. Запрещено изменение атрибута.*
- *ENOSYS - в текущей реализации не реализована установка атрибута.*

3.2 Интерфейс обмена сообщениями

3.2.1 Создание нового сообщения

Обмен данными между прикладными объектами осуществляется посредством передачи сообщений. В листинге 14 представлена сигнатура функции формирования нового сообщения.

Листинг 14. Функция ***msg_new()*** - создание сообщения.

```
1  #include    <zigzag.h>
2  msg_t   msg_new(
3      net_addr_t  dst_addr,
4      app_port_t  dst_port,
5      uint8_t     msg_type,
6      size_t      body_size
7      );
```

Аргументы функции *msg_new()*:

1). ***dst_addr*** *- сетевой адрес назначения,*

2). ***dst_ port*** *- прикладной порт назначения,*

3). ***msg_ type*** *- тип сообщения. Допустимые типы сообщений от 32 до 255.*

4). ***body_ size*** *- размер тела сообщения в октетах.*

Возвращаемое значение:

Функция возвращает идентификатор сообщения в случае успешного вызова, а именно значение ≥ 0*, либо отрицательное значение в случае ошибки. Допустимы следующие ошибочные значения:*

- *EINVAL - некорректное значение аргумента,*
- *ENOMEM - недостаточно памяти для создания нового сообщения.*
- *ENOSYS - в текущей реализации функция не поддерживается.*

После создания сообщение находится в заблокированном состоянии, то есть система ZigZag не может производить над этим сообщением какие-либо дейтсвия (например, отправку).

3.2.2 Получение информации о сообщении

Пока сообщение находится в заблокированном состоянии информация о нём может быть получена посредством вызова функции ***msg_ info()***. Сигнатура этой функции представлена в листинге 15.

Листинг 15. Функция ***msg_ info()*** - информация о сообщении

```
1  #include     <zigzag.h>
2  result_t msg_info(
3          msg_t    msg,
```

```
4           struct msginfo  *info
5           );
```

Аргументы функции *msg_info()*:

1). ***msg*** *- идентификатор сообщения.*

2). ***info*** *- указатель на структуру, в которую будет скопирована информация о сообщении. Структура определена в заголовочном файле zigzag.h. Определение структуры представлено в листинге 16.*

Листинг 16. Определение структуры **msginfo**.

```
1       struct msginfo {
2           net_addr_t  dst_addr;
3           app_port_t  dst_port;
4           net_addr_t  src_addr;
5           app_port_t  src_port;
6           uint8_t     msg_type;
7           size_t      body_size;
8           void        *body_ptr;
9       };
```

Описание полей структуры msginfo:

- ***dst_addr*** *- сетевой адрес назначения,*
- ***dst_port*** *- прикладной порт назначения,*
- ***src_addr*** *- сетевой адрес источника,*
- ***src_port*** *- прикладной порт источника,*
- ***msg_type*** *- тип сообщения,*

- ***body_size*** *- размер тела сообщения в октетах,*
- ***body_ptr*** *- указатель на тело сообщения.*

Возвращаемые функцией *msg_info()* значения:

- *ENOERR - информация о сообщении успешно получена,*
- *EINVAL - неправильный идентификатор сообщения или аргумент* ***info*** *равен нулю.*
- *ENOSYS - в текущей реализации вызов функции не поддерживается.*

3.2.3 Удаление сообщения

Удаление сообщения из системы осуществляется посредством функции ***msg_destroy***. Сигнатура функции представлена в листинге 17.

Листинг 17. Функция ***msg_destroy()*** - удаление сообщения.

```
1  #include    <zigzag.h>
2  result_t msg_destroy(
3      msg_t    msg
4      );
```

Аргументы функции *msg_destroy()*:

1). msg - идентификатор удаляемого сообщения.

Возвращаемые функцией *msg_destroy()* значения:

- *ENOERR - сообщение успешно удалено из системы,*
- *EINVAL - некорретный идентификатор сообщения,*

- *EBUSY - системы производит над сообщением какие-то операции. Удалено может быть только заблокированное сообщение.*
- *ENOSYS - в текущей версии системы функция не поддерживается.*

3.2.4 Передача сообщений

Для передачи сообщения предназначена функция ***msg_send()***. После вызова этой функции сообщение разблокируется и система ZigZag начинает процедуру отправки этого сообщения по назначению. Сигнатура функции ***msg_send()*** представлена в листинге 18.

Листинг 18. Функция ***msg_send()*** - отправка сообщения.

```
1  #include    <zigzag.h>
2  result_t msg_send(
3      msg_t    msg
4      );
```

Аргументы функции *msg_send()*:

1). msg - идентификатор отправляемого сообщения.

Возвращаемые функцией *msg_send()* значения:

- *ENOERR - успешно начата процедура отправки сообщения,*
- *EINVAL - некорректное значение аргумента **msg**,*
- *ENOSYS - функция не поддерживается системой ZigZag .*

После окончания процедуры отправки система ZigZag вызывает функцию ***msg_send_done()***, в свою очередь прикладной объект обязан предоставить реализацию этой функции. В листинге 19 представлена сигнатура функции ***msg_send_done()***.

Листинг 19. Функция ***msg_send_done()*** - окончание отправки

```
1  #include    <zigzag.h>
2  void msg_send_done(
3      msg_t    msg
4      send_status_t    status
5      );
```

Аргументы функции *msg_send_done()*:

1). ***msg*** *- идентификатор сообщения,*

2). ***status*** *- статус завершения процедуры отправки сообщения. Допустимы следующие значения:*

- *STATUS_SUCCESS - сообщение успешно отправлено по назначению,*
- *STATUS_TIMEOUT - продолжительность процедуры отправки превысила допустимый предел. Сообщение не было отправлено,*
- *STATUS_MAX_ATTEMPTS - превышено число попыток отправки сообщения. Сообщение не было отправлено по назначению.*

3.2.5 Приём сообщений

Приняв сообщение предназначенное данному прикладному объекту система ZigZag вызывает функцию ***msg_recv()***. В свою очередь прикладной объект обязан предоставить реализацию этой функции. Сигнатура функции представлена в листинге 20.

Листинг 20. Функция ***msg_recv()*** - приём сообщения.

```
1  #include    <zigzag.h>
```

```
2       size_t msg_recv(
3           msg_t    msg
4           ) {}
```

Аргументы функции *msg_recv()*:

1). ***msg*** *- идентификатор принятого сообщения.*

Возвращаемое значение:

Функция обязана возвратить размер всего сообщения (включая заголовок сообщения) или 0, если размер сообщения определить не удалось.

Перед вызовом функции ***msg_recv()*** сообщение переводится в заблокированное состояние. Поэтому прикладной объект может воспользоваться функцией ***msg_info()*** для получения информации о сообщении.

Поскольку размер тела сообщения системе не известен, то поле **body_size** структуры **msginfo** равно 0. При этом поле **body_ptr** указывает на первый октет тела сообщения. По этой же причине прикладной объект обязан сообщить системе размер сообщения, возвратив его в функции ***msg_recv()***. Если ***msg_recv()*** возвратит некорректное значение, то оставшиеся сообщения в пакете будут обработаны неправильно.

4 Сервисы абстракции от аппаратуры

4.1 Интерфейс аппаратных прерываний

4.1.1 Обработка прерываний

Обработка аппаратных прерываний осуществляется в функции ***irq_handler()***. Прикладной объект обязан предоставить реализацию этой функции. Сигнатура функции представлена в листинге 21.

Листинг 21. Функция ***irq_handler()*** - обработчик прерываний.

```
1  #include    <zigzag.h>
2  void  irq_handler(
3        irq_t   irq
4        ) {}
```

Аргументы функции *irq_handler()*:

1). ***irq*** *- номер прерывания. Зависит от используемого микроконтроллера. Так для TI msp430 определены следующие номера прерываний (в скобках указан адрес вектора прерывания):*

- *0 - DAC/DMA (0xffe0),*
- *2 - PORT2 (0xffe2),*
- *8 - PORT1 (0xffe8),*
- *14 - ADC 12 (0xffee)*
- *22 - COMPARATOR A (0xfff6)*

Из функции ***irq_handler*** разрешено вызывать следующие функции системы ZigZag :

- ***event_emit()***;
- ***storage_alloc()***, ***storage_destroy()***;
- ***storage_lock()***, ***storage_unlock()***;
- ***sys_time()***;
- ***timer_set()***, ***timer_info()***, ***timer_stop()***.

4.2 Интерфейс асинхронных таймеров

Данный интерфейс позволяет воспользоваться возможностями существующего в системе аппаратного таймера. Число доступных таймеров зависит от аппаратной конфигурации узла.

4.2.1 Значение счётчика таймера

Функция ***timer_counter()*** возвращает текущее значение счётчика таймера. Частота обновления счётчика зависит от аппаратной конфигурации узла. В листинге 22 представлена сигнатура функции ***timer_counter()***.

Листинг 22. Функция ***timer_counter()*** - значение счётчика.

```
1  #include    <zigzag.h>
2  uint32_t    timer_counter()
```

4.2.2 Установка таймера

Установить таймер на определённое время срабатывания позволяет функция ***timer_set()***. Сигнатура этой функции представлена в листинге 23.

Листинг 23. Функция ***timer_set()*** - установка таймера.

```
1  #include    <zigzag.h>
2  result_t    timer_set(
3      uint8_t     tnum,
4      uint32_t    tpoint
5      );
```

Аргументы функции *timer_set()*:

1). ***tnum*** *- номер аппаратного таймера. Число доступных таймеров зависит от реализации.*

2). ***tpoint*** *- момент времени срабатывания таймера. Когда* ***tpoint*** *будет равен счётчику таймера произойдёт срабатывание. При повторном вызове* ***timer_set()*** *таймер устанавливается на новое значение* ***tpoint****.*

Возвращаемые значения:

- *ENOERR - таймер успешно установлен,*
- *EINVAL - некорректное значение аргумента. Возможно неправильно указан номер таймера.*
- *ENOSYS - функция не поддреживается системой.*

4.2.3 Срабатывание таймера

Если таймер был установлен посредством функции ***timer_set()*** и **tpoint**≥***timer_count()*** происходит срабатывание таймера и система вызывает функцию ***timer_fired()***. Прикладной объект обязан предоставить реализацию этой функции. Сигнатура функции ***timer_fired()*** представлена в листинге 24.

Листинг 24. Функция ***timer_fired()*** - срабатывание таймера.

```
1  #include    <zigzag.h>
2  void    timer_fired(
3      uint8_t     tnum
4      );
```

Аргументы функции *timer_fired()*:

1). ***tnum*** *- номер сработавшего аппаратного таймера.*

Из функции ***timer_fired()*** разрешено вызывать те же функции, что и из ***irq_handler()***.

4.2.4 Получение информации о таймере

Узнать установлен ли таймер и на какое значение можно с помощью функции ***timer_info()***, сигнатура которой представлена в листинге 25.

Листинг 25. ***timer_info()*** - информация о таймере

```
1  #include    <zigzag.h>
2
3  result_t    timer_info(
4      uint8_t     tnum,
5      struct timerinfo    *tinfo
6      );
```

Аргументы функции *timer_info()*:

1). ***tnum*** - *номер таймера, информацию о котором требуется получить.*

2). ***tinfo*** - *указатель на структуру, в которую записывается информация о таймере. Структура определена в заголовочном файле zigzag.h. Определение структуры представлено в листинге 26.*

Листинг 26. Определение структуры **timerinfo**

```
1  struct timerinfo {
2      uint8_t     is_set;
3      uint32_t    tpoint;
4  };
```

Описание полей структуры:

a). ***is_set*** *Если поле равно 0, то таймер не установлен, иначе если* ***is_set*** *> 0, то таймер установлен.*

b). ***tpoint*** *- значение счётчика, при достижении которого должно произойти срабатывание таймера в случае если он установлен.*

4.2.5 Остановка таймера

Остановить установленный таймер позволяет функция ***timer_stop()***. В листинге 27 представлена сигнатура этой функции.

Листинг 27. ***timer_stop()*** - остановка таймера

```
1  #include    <zigzag.h>
2
3  result_t    timer_stop(
4      uint8_t     tnum
5      );
```

Аргументы функции *timer_stop()*:

1). ***tnum*** *- номер таймера, который требуется остановить.*

Возвращаемые значения:

- *ENOERR - таймер успешно остановлен,*
- *EINVAL - некорректное значение аргумента* ***tnum***,
- *ENOSYS - в текущей реализации функция не поддерживается.*

4.3 Интерфейс цифровых входов/выходов

Данный интерфейс позволяет работать с портами цифровых входов/выходов. Число доступных в системе портов задаётся константой

PORT_TOTAL. Нумерация портов начинается с 1.

4.3.1 Запись в порт

Функция ***port_write()*** позволяет задать значения битов порта, которые будут выведены на соответствующие ножки ввода/вывода (сконфигурированные на функцию ввода/вывода и имеющие направление на вывод). Сигнатура функции представлена в листинге 28.

Листинг 28. ***port_write()*** - запись в порт

```
1  #include    <zigzag.h>
2  result_t    port_write(
3      const uint8_t  port_num,
4      const port_t mask,
5      const port_t value
6  );
```

Аргументы функции *port_write()*:

1). ***port_num*** *- номер порта.*

2). ***mask*** *- маска, задающая выходы порта, которые будут записаны.*

3). ***value*** *- значение, которое будет записано в порт в соответствии с маской.*

Возвращаемые значения:

- *ENOERR - все биты, определённые маской, успешно записаны,*
- *EINVAL - некорректное значение аргумента* ***port_num****,*
- *EPERM - запись в порт запрещена*
- *ENOSYS - в текущей реализации функция не поддерживается.*

4.3.2 Чтение из порта

Если биты порта скофигурированы на функцию ввода/вывода и имеют направление на ввод, то с помощью функции ***port_read()*** могут быть получены уровни входных сигналов соответсвующих входов порта. Сигнатура функции представлена в листинге ***portreadfunc***.

Листинг 29. ***port_read()*** - чтение из порта

```
1  #include    <zigzag.h>
2  result_t    port_read(
3      const uint8_t port_num,
4      const port_t mask,
5      port_t *const value_ptr
6  );
```

Аргументы функции *port_read()*:

1). ***port_num*** *- номер порта.*

2). ***mask*** *- маска, задающая входы порта, которые будут прочитаны.*

3). ***value_ptr*** *- указатель на область памяти, в которую будут записаны текущие значения входов порта в соответсвии с маской.*

Возвращаемые значения:

- *ENOERR - все биты, определённые маской, успешно прочитаны,*
- *EINVAL - некорректное значение аргумента* ***port_num****,*
- *EPERM - чтение из порта запрещена*
- *ENOSYS - в текущей реализации функция не поддерживается.*

4.3.3 Установка атрибутов порта

Функция ***port_set_attr*** позволяет сконфигурировать порт, задав соответствующие атрибуты порта. В качестве параметра в функцию ***port_set_attr*** передаётся указатель на структуру с атрибутами порта. В листинге 30 представлено определение структуры из заголовочного файла zigzag.h, содержащей атрибуты порта.

Листинг 30. Определение структуры **port_attr_t**

```
1    typedef struct {
2        port_t  dir;
3        port_t  sel;
4        port_t  ie;
5        port_t  ies;
6} port_attr_t;
```

Поля структуры *port_attr_t*:

1). ***dir*** *- направление порта. Если соответствующий бит установлен в 1, то ножка порта переключается на вывод. В противном случае на ввод.*

2). ***sel*** *- поле выбора функции. Ножки порта могут быть мультиплексированы с другими функциями периферийных модулей. Каждый бит этого поля определяет, как будет использована ножка - в качестве порта ввода/вывода или в качестве функции периферийного модуля. Если соответствующий бит равен 0, то для ножки выбирается функция ввода/вывода, иначе функция периферийного модуля.*

3). ***ie*** *- разрешение прерываний. Каждый бит этого поля разрешает прерывание для соответствующей ножки порта (не для всех*

портов). Если установлен бит в 0, то прерывание запрещено, иначе разрешено.

4). ***ies*** *- выбор фронта прерывания. Каждый бит данного поля позволяет выбрать, по какому фронту сигнала будет происходить прерывание для соответствующей ножки ввода/вывода (не для всех портов). Если соответствующий бит поля равен 0, то флаг прерывания устанавливается при изменении уровня сигнала с низкого на высокий. Есл бит равен 1, то флаг прерывания устанавливается при изменении уровня сигнала с высокого на низкий.*

В листинге 31 представлена сигнатура функции установки атрибутов порта.

Листинг 31. **port_set_attr()** - установка атрибутов порта

```
1  #include    <zigzag.h>
2  result_t    port_set_attr(
3      const uint8_t port_num,
4      const port_t mask,
5      const port_attr_t *const port_attr_ptr
6  );
```

Аргументы функции *port_set_attr()*:

1). ***port_num*** *- номер порта.*

2). ***mask*** *- маска, задающая входы/выходы порта, атрибуты которых будут установлены.*

3). ***port_attr_ptr*** *- указатель на структуру с новыми значениями атрибутов.*

Возвращаемые значения:

- *ENOERR - атрибуты всех входов/выходов успешно установлены в соответствии с маской,*
- *EINVAL - некорректное значение аргумента* ***port_num****,*
- *EPERM - установка атрибутов какого-либо из входов/выходов порта из маски запрещена,*
- *ENOSYS - в текущей реализации функция не поддерживается.*

4.3.4 Получение атрибутов порта

Получить текущие значения атрибутов порта позволяет функция ***port_get _attr***, сигнатура которой представлена в листинге 32.

Листинг 32. **port_get_attr()** - получение атрибутов порта

```
1  #include    <zigzag.h>
2  result_t    port_get_attr(
3      const uint8_t port_num,
4      const port_t mask,
5      port_attr_t *const port_attr_ptr
6  );
```

Аргументы функции *port_get_attr()*:

1). ***port_num*** *- номер порта.*

2). ***mask*** *- маска, задающая входы/выходы порта, атрибуты которых должны быть получены.*

3). ***port_attr_ptr*** *- указатель на структуру, в которую будут записаны текущие значения атрибутов порта.*

Возвращаемые значения:

- *ENOERR - атрибуты всех входов/выходов успешно получены в соответствии с маской,*
- *EINVAL - некорректное значение аргумента **port_num**,*
- *EPERM - получение атрибутов какого-либо из входов/выходов порта из маски запрещена,*
- *ENOSYS - в текущей реализации функция не поддерживается.*

4.3.5 Получение флагов прерываний

Ножки определённых портов имеют возможность вызова прерываний. Для получения текущих значений флагов прерываний порта предназначена функци ***port_get_iflag()***. Сигнатура этой функции представлена в листинге 33.

Листинг 33. **port_get_iflag()** - получение флагов прерывний

```
1  #include    <zigzag.h>
2  result_t    port_get_iflag(
3      const uint8_t port_num,
4      const port_t mask,
5      port_t *const iflag_ptr
6  );
```

Аргументы функции *port_get_iflag()*:

1). ***port_num*** *- номер порта.*

2). ***mask*** *- маска, задающая входы/выходы порта, флаги прерываний которых должны быть получены.*

3). ***iflag_ptr*** *- указатель на область памяти, в которую будут записаны текущие значения флагов прерываний порта в соответствии с маской.*

Возвращаемые значения:

- *ENOERR - успешно получены все флаги прерываний порта,*
- *EINVAL - некорректное значение аргумента* ***port_num****,*
- *EPERM - получение флага прерывания какого-либо из входов/выходов порта из маски запрещена,*
- *ENOSYS - в текущей реализации функция не поддерживается.*

4.3.6 Сброс флагов прерываний

Для сброса флагов прерываний ножек порта предназначена функци ***port_reset_iflag()***. Сигнатура этой функции представлена в листинге 34.

Листинг 34. **port_reset_iflag()** - сброс флагов прерывний

```
1  #include    <zigzag.h>
2  result_t    port_reset_iflag(
3      const uint8_t port_num,
4      const port_t mask
5  );
```

Аргументы функции *port_reset_iflag()*:

1). ***port_num*** *- номер порта.*

2). ***mask*** *- маска, задающая входы/выходы порта, флаги прерываний которых должны быть сброшены.*

Возвращаемые значения:

- *ENOERR - успешно сброшены все флаги прерываний порта в соответствии с маской,*
- *EINVAL - некорректное значение аргумента* ***port_num****,*
- *EPERM - сброс флага прерывания какого-либо из входов/выходов порта из маски запрещена,*
- *ENOSYS - в текущей реализации функция не поддерживается.*

4.4 Интерфейс АЦП

4.4.1 Инициализация АЦП

Перед использованием аналого-цифрового преобразователя его необходимо проинициализировать, задав режим работы. Сигнатура функции инициализации представлена в листинге 35.

Листинг 35. **adc_init()** - инициализация АЦП

```
1  #include    <zigzag.h>
2  result_t    adc_init( adc_mode_t mode, ... );
```

Аргументы функции *adc_init()*:

1). ***mode*** *- режим работы АЦП. Допустимы следующие режимы:*

- *SINGLE_CONVERSION - одна выборка-преобразование с канала 0.*

2). ***...*** *- дополнительные аргументы, зависящие от режима.*

Возвращаемые значения:

- *ENOERR - АЦП успешно проинициализировано,*

- *EINVAL - некорректное значение аргумента функции,*

4.4.2 Запуск АЦП

Подготовка и запуск выборки-преобразования аналого-цифрового преобразователя.

Листинг 36. **adc_start()** - запуск АЦП

```
1  #include    <zigzag.h>
2  result_t    adc_start( bool_t only_prepare );
```

Аргументы функции *adc_start()*:

1). ***only_prepare*** *- Если значение параметра равно TRUE, то функция должна только подготовить АЦП к процедуре выборки и преобразования.*

Возвращаемые значения:

- *ENOERR - выборка-преобразование запущены,*

4.4.3 Получение данных с АЦП

Функция позволяет получить результат преобразования АЦП. Результаты преобразования становятся доступны после получения события EV_ADC.

Листинг 37. **adc_read()** - вычитывание с АЦП

```
1  #include    <zigzag.h>
2  int32_t     adc_read( uint16_t channel );
```

Аргументы функции *adc_read()*:

1). ***channel*** *- номер канала.*

Возвращаемые значения: Возвращается очередное значение преобразования или значение < 0 в случае ошибки.

4.4.4 Останов АЦП

Функция ***adc_stop()*** предназначена для остановки работы АЦП. Для повторного запуска АЦП нужно вызвать функцию ***adc_start()***.

Листинг 38. **adc_stop()** - останов АЦП

```
1  #include    <zigzag.h>
2  result_t    adc_stop( bool_t force );
```

Аргументы функции *adc_stop()*:

1). ***force*** *- если значение аргумента равно TRUE, то принудительно остановить АЦП.*

Возвращаемые значения:

- *ENOERR - возвращается в случае успешной остановки АЦП,*
- *EBUSY - если АЦП занят выборкой-преобразованием и значение аргумента* ***force*** *равно FALSE.*

4.5 Интерфейс ЦАП

Библиография

[1] IEEE 802.15.4 *Standard Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)* // IEEE Standard for Information Technology, IEEE-SA Standards Board, 2006.

[2] ZigBee Alliance *ZigBee specification* // December, 2006.