

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

Описание ZigZagAPI

Гекк Максим

Пышtev Сергей

Москва

2007 год

# Оглавление

<b>1</b>	<b>Общее описание</b>	<b>5</b>
<b>2</b>	<b>Сервисы операционной среды</b>	<b>7</b>
2.1	Интерфейс общего назначения . . . . .	7
2.1.1	Инициализация прикладного объекта . . . . .	7
2.1.2	Запрос текущего времени . . . . .	7
2.2	Интерфейс системных событий . . . . .	7
2.2.1	Обработка событий . . . . .	8
2.2.2	Генерация событий . . . . .	9
2.3	Интерфейс доступа к хранилищу . . . . .	10
2.3.1	Запрос места в хранилище . . . . .	10
2.3.2	Получение доступа к данным . . . . .	11
2.3.3	Высвобождение места в хранилище . . . . .	13
2.4	Интерфейс синхронных таймеров . . . . .	14
<b>3</b>	<b>Сервисы сетевого взаимодействия</b>	<b>14</b>
3.1	Интерфейс доступа к атрибутам . . . . .	14
3.1.1	Чтение атрибута . . . . .	14
3.1.2	Запись атрибута . . . . .	15
3.2	Интерфейс обмена сообщениями . . . . .	16
3.2.1	Создание нового сообщения . . . . .	16
3.2.2	Получение информации о сообщении . . . . .	17
3.2.3	Удаление сообщения . . . . .	19
3.2.4	Передача сообщений . . . . .	19
3.2.5	Приём сообщений . . . . .	21
<b>4</b>	<b>Сервисы абстракции от аппаратуры</b>	<b>22</b>
4.1	Интерфейс аппаратных прерываний . . . . .	22

4.1.1	Обработка прерываний . . . . .	22
4.2	Интерфейс асинхронных таймеров . . . . .	23
4.2.1	Значение счётчика таймера . . . . .	23
4.2.2	Установка таймера . . . . .	24
4.2.3	Срабатывание таймера . . . . .	25
4.2.4	Получение информации о таймере . . . . .	25
4.2.5	Остановка таймера . . . . .	26
4.3	Интерфейс цифровых входов/выходов . . . . .	27
4.4	Интерфейс аналого-цифрового преобразователя . . . . .	27
<b>Библиография</b>		<b>28</b>

## Перечень схем

1	Общая архитектура системы ZigZag. . . . .	5
---	---	---

## Перечень таблиц

1	Системные типы событий. . . . .	8
---	---------------------------------	---

## Листинги

1	Сигнатура макроса BIND . . . . .	6
2	<b>sys_init()</b> - инициализация системы . . . . .	7
3	<b>sys_time()</b> - текущее время . . . . .	7
4	Функция <b>event_handler()</b> - обработчик событий. . . . .	8
5	Функция <b>event_emit()</b> - генерация события. . . . .	9
6	Функция <b>storage_alloc()</b> - запрос места в хранилище . .	11
7	Функция <b>storage_lock()</b> - доступ к хранилищу . . . . .	12
8	Функция <b>storage_unlock()</b> - завершение доступа . . . . .	12
9	Функция <b>storage_free()</b> - возврат памяти в хранилище .	13
10	Функция <b>attr_read()</b> - чтение атрибута. . . . .	14
11	Функция <b>attr_write()</b> - запись атрибута. . . . .	15
12	Функция <b>msg_new()</b> - создание сообщения. . . . .	16
13	Функция <b>msg_info()</b> - информация о сообщении . . . . .	17
14	Определение структуры <b>msginfo</b> . . . . .	17
15	Функция <b>msg_destroy()</b> - удаление сообщения. . . . .	19
16	Функция <b>msg_send()</b> - отправка сообщения. . . . .	20
17	Функция <b>msg_send_done()</b> - окончание отправки . . . .	20
18	Функция <b>msg_recv()</b> - приём сообщения. . . . .	21
19	Функция <b>irq_handler()</b> - обработчик прерываний. . . . .	22

20	Функция <b><i>timer_counter()</i></b> - значение счётчика. . . . .	24
21	Функция <b><i>timer_set()</i></b> - установка таймера. . . . .	24
22	Функция <b><i>timer_fired()</i></b> - срабатывание таймера. . . . .	25
23	<b><i>timer_info()</i></b> - информация о таймере . . . . .	25
24	Определение структуры <b>timerinfo</b> . . . . .	26
25	<b><i>timer_stop()</i></b> - остановка таймера . . . . .	27

# 1 Общее описание

Система ZigZag представляет собой коммуникационную среду для обмена сообщениями между прикладными объектами узлов сенсорной сети. На одном узле сети может находиться 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.

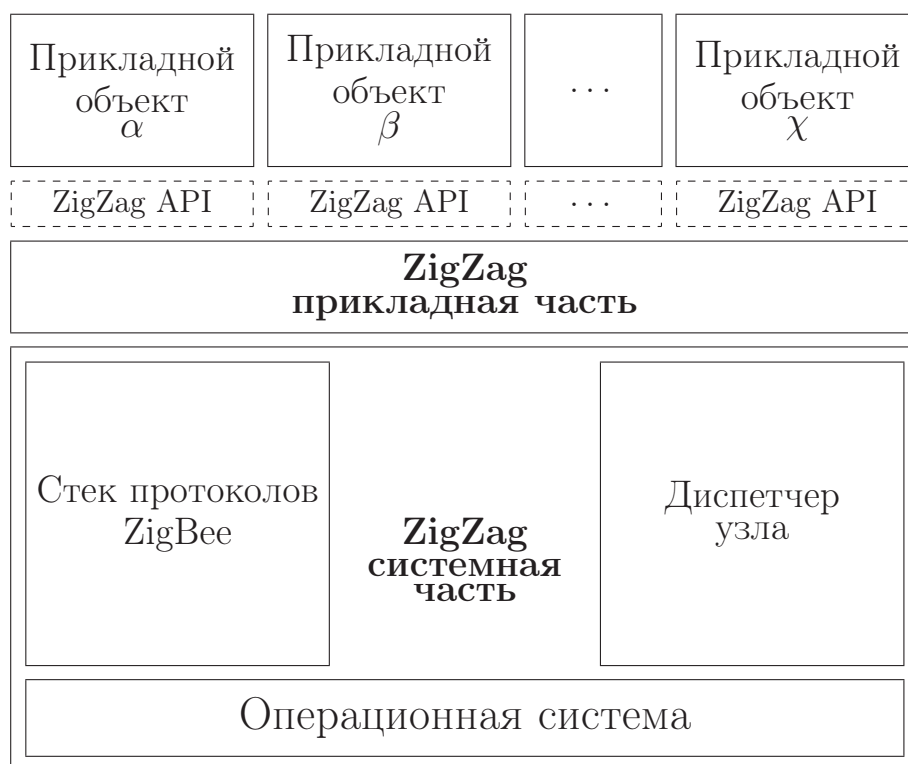


Рис. 1. Общая архитектура системы ZigZag.

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 связывания прикладного объекта с определённым портом и назначения ему имени предназначен макрос ***BIND***, сигнатура которого представлена в листинге 1.

Листинг 1. Сигнатура макроса BIND

```
1  BIND(  
2      obj_number ,  
3      port_number ,  
4      obj_name  
5  )
```

#### Описание аргументов макроса **BIND**:

- 1). ***obj\_number*** - уникальный номер прикладного объекта,
- 2). ***port\_number*** - номер порта, к которому привязывается объект,
- 3). ***obj\_name*** - имя прикладного объекта.

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 необ-



ходимые 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 0x00 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 <b>unidata</b> содержит предполагаемую продолжительность сна в миллисекундах.
EV_SFREE	0xfc	Освобождена область памяти хранилища. Аргумент <b>unidata</b> содержит дескриптор этой области памяти.
EV_RUNLOCK	0xfb	Завершён доступ на чтение к области памяти хранилища. Аргумент <b>unidata</b> равен дескриптору области памяти.
EV_WUNLOCK	0xfa	Завершён доступ на запись к области памяти хранилища. Аргумент <b>unidata</b> равен дескриптору области памяти.
EV_AWRITED	0xf9	Записано новое значение атрибута. Аргумент <b>unidata</b> 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

# 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 10

Листинг 10. 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 представлена в листинге 11.

Листинг 11. 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 вернуть 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 12 представлена сигнатура функции формирования нового сообщения.

Листинг 12. 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  $\geq 0$ , 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 действия ( 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 13.

Листинг 13. 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 14.

Листинг 14. 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 15.

Листинг 15. 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* - некорректный 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 16.

Листинг 16. 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 17 представлена сигнатура функции ***msg\_send\_done()***.

Листинг 17. 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 18.

Листинг 18. 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 вернуть размер всего сообщения ( включая заголовки 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 19.

Листинг 19. 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 ( 0xffff6 )

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 конфи-



гурации узла. В листинге 20 представлена сигнатура функции ***timer\_counter()***.

Листинг 20. 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 21.

Листинг 21. 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 поддерживается системой.

### 4.2.3 Срабатывание таймера

Если таймер был установлен посредством функции ***timer\_set()*** и ***tpoint***  $\geq$  ***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 22.

Листинг 22. 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 работавшего 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 23.

Листинг 23. ***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 24.

#### Листинг 24. 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 25 представлена сигнатура этой функции.

Листинг 25. *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

## 4.4 Интерфейс аналого-цифрового преобразователя

# 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.