Содержание

1 Введение

lusb представляет собой стек USB-устройства (device), предназначенный для микроконтроллеров. Стек написан на языке C и может быть перенесен на различные архитектуры. Ядро стека не содержит работу с аппаратурой, вся аппаратно-зависимая часть вынесена в порт. В настоящее время реализован только порт для контроллеров lpc17xx.

Стек поддерживает стандартные функции usb-устройства, за исключением пунктов, перечисленных ниже.

Стек НЕ поддерживает

1. Стек не поддерживает следующие стандартные запросы:
   `SET_INTERFACE`
   `SET_DESCRIPTOR`
   `SYNC_FRAME`
2. Стек пока не поддерживает скорость High Speed (и соответствующие дескрипторы)
3. В стеке не реализована поддержка нескольких конфигураций, альтернативные настройки интерфейсов
4. Не реализована поддержка изохронных контрольных точек

Порт для lpc17xx НЕ поддерживает:

1. Не реализована поддержка изохронных контрольных точек
2. Не реализована поддержка режима ATLE

2 Уровни программы

Программа, использующая стек lusb состоит из 3-х уровней.

1. Самый нижний уровень включает в себя аппаратно-зависимые функции работы с определенным контроллером USB. Этот уровень полностью реализует порт стека для данного контроллера.
2. Промежуточный уровень – уровень ядра стека. Здесь выполняются основные логические операции USB-стека. Этот уровень является связующим между портом и приложением (приложение вызывает функции ядра и не работает с функциями порта напрямую). Здесь реализована логика работы 0-ой контрольной точки, включая обработку стандартных управляющих запросов, а так же хранится состояние USB-стека.
3. Верхний уровень - уровень приложения – пользовательский код, использующий USB–стек.

Связь между уровнями существует в обоих направлениях. Связь сверху вниз осуществляется, как правило, посредством вызова верхним уровнем функций нижнего. Связь снизу вверх осуществляется с помощью callback-функций. Ядро реализует набор стандартных callback-функций, которые может вызывать порт при возникновении событий аппаратуры. Приложение может реализовывать так же свой набор callback-функций, которые будет вызывать ядро для оповещения о событиях (смотри пункт *Реализация callback-функций приложения*).

3 Список и описание используемых файлов

Для подключения стека необходимо добавить в проект все .c файлы из директории **lusb**, а также из **lusb/ports/<семейство контроллеров>**. Также необходимосоздать в проекте файлы (скопировать из примера) **typedefs.h**, **lusb_config_init.h** и **lusb_port_config.h**. Для работы со стеком из приложения необходимо в исходных кодах включить заголовочный файл **lusb.h**, который содержит определения всех необходимых функций и включает все необходимые заголовки.

Все файлы, относящиеся к стеку lusb, за исключением **typedefs.h**, **lusb_config_init.h** и **lusb_port_config.h**, содержатся в директории **lusb** и могут быть совместно использованы разными проектами. Файлы **lusb_config_init.h** и **lusb_port_config.h** содержат настройки стека, которые могут меняться от проекта к проекту, в зависимости от требований приложения, среды и т.д., поэтому каждый проект содержит собственную копию этих файлов (для некоторых приложений может понадобится так же ручное изменение дескрипторов, содержащихся в файле **lusb_descriptors.c** – в этом случае рекомендуется сделать его копию в проекте и использовать ее).

lusb_config_init.h содержит общие настройки стека. Здесь находятся макросы, которые определяют поля стандартных дескрипторов, размеры буферов для 0-ой конечной точки, какие точки используют DMA и т.д.

lusb_port_config.h содержит настройки, зависящие от порта стека на конкретный контроллер.

Файл **typedefs.h** содержит определение стандартных типов – `uint32_t`, `uint16_t`, `uint8_t`, `int32_t`, `int16_t` и `int8_t`, а так же определение `NULL`.

Файлы, относящиеся к портам под конкретные контроллеры, расположены в поддиректории **ports**. В ней находятся поддиректории с названиями, соответствующими архитектуре контроллера, для которой реализован порт (пока только **lpc17xx**). Проект должен включать файлы из одной из этих папок (состав файлов определяется портом).

Файлы, содержащиеся в корневой директории (**lusb**), содержат ядро стека и являются общими для всех портов. Все порты должны реализовывать определенный одинаковый набор функций. Прототипы этих функций находятся в файле **lusb_ll.h** (названия этих функций имеют вид **lusb_ll_xxx**).

Логика работы стека в большей части содержится в файле **lusb_core.c**. Здесь находятся callback-функции, вызываемые портом при возникновении различных аппаратных событий. В данном файле содержится реализация функции *`lusb_progress()`*, отвечающая за продвижение стека. Здесь также сосредоточена логика работы нулевой конечной точки, за исключением непосредственно обработки стандартных управляющий запросов, которая вынесена в отдельный файл – **lusb_ctrlreq.c**. Все стандартные дескрипторы usb содержатся в файле **descriptors.c**.

Если контроллер не поддерживает аппаратного DMA, то передача может осуществляться программно ядром стека. Логика программной передачи данных сосредоточена в файле **lusb_sio.c**. Реализация программной передачи включается, если определить `LUSB_USE_SIO` в **lusb_config_init.h**. Если `LUSB_USE_SIO` не определено, то файл **lusb_sio.c** можно исключить из проекта.

Определение констант и флагов стека сосредоточены в двух файлах: **lusb_usbdefs.h** и **lusb_const.h**. В первом находятся константы, определяемые спецификацией USB, а во втором константы, определяемые непосредственно стеком. Типы, определяемые стеком, находятся в **lusb_typedefs.h**.

lusb_appl_cb.h содержит прототипы callback-функций, которые могут быть использованы приложением.

В поддиректории **classes** находятся реализации стандартных классов устройств. В настоящее время реализован один класс — Mass Storage Device, реализация которого находится в поддиректории **msc**.

4 Стандартная последовательность вызовов

Перед началом работы со стеком необходимо вызвать функцию *lusb_init()*, которая выполняет все необходимые начальные аппаратные настройки usb-контроллера (**Внимание!!** для порта под **lpc17xx** установка PLL1 для usb не выполняется, так как предполагается, что она выполнена в startup) и переводит модуль в неподключенное состояние. В зависимости от реализации порта, функция так же может протестировать настройки порта и в случае, если тест не пройден, вернуть код ошибки.

Далее необходимо вызвать *lusb_connect()* для подключения модуля к шине, передав ей в качестве параметра единицу.

Для продвижения usb-стека необходимо периодически вызывать функцию *lusb_progress()*, в которой выполняются фоновые задачи стека.

Чтобы начать передачу данных, необходимо дождаться, пока устройство будет сконфигурировано. Для того, чтобы определить, что устройство сконфигурировано, можно воспользоваться функцией *lusb_dev_is_configured()* или реализовать пользовательскую callback-функцию *lusb_appl_cb_devstate_ch()*.

Таким образом, последовательность вызовов может иметь следующий вид:

```
//инициализация usb
int res = lusb_init();
if (res!=LUSB_ERR_SUCCESS)
{
	//ошибка инициализации usb-модуля...
}
else
{
	//подключение модуля к шине
	lusb_connect(1);
	//ждем, пока usb-модуль будет сконфигурирован...
	while (!lusb_dev_is_configured())
		lusb_progress();

	while (lusb_dev_is_configured())
	{
		//тут идет работа с конечными точками...
		//...
		//...

		//продвижение стека
		lusb_progress();
	}
}
```

О том, как осуществляется передача данных по конечным точкам, смотри главу *«Передача данных»*.

5 Реализация callback-функций приложения

Приложение может реализовать callback-функции, которые будет вызывать ядро lusb при возникновении определенных событий. Для того чтобы подключить callback-функцию, надо определить соответствующий макрос в **lusb_config_init.h**. Объявление прототипов пользовательских callback-функций находятся в файле **lusb_appl_cb.h**.

Следует учитывать, что в зависимости от настроек стека определенные функции могут вызываться как из прерывания, так и из *lusb_progress()*. Какая часть стека выполняется в обработчике прерывания, а какая внутри *lusb_progress()* определяется макросами LUSB_INTERRUPT и LUSB_CTLREQ_PROC_IN_INT из **lusb_config_init.h**. Если LUSB_INTERRUPT не определен, то прерывания не используются вообще, и вся работа стека сосредоточена внутри функции *lusb_progress()*. При определении LUSB_INTERRUPT обработка основных событий USB-контроллера и передача данных осуществляется внутри обработчиков прерываний. При включении дополнительно макроса LUSB_CTLREQ_PROC_IN_INT в прерывании выполняется и обработка всех управляющих запросов.

Все callback-функции, которые может реализовать приложение, приведены в разделе *Callback-функции приложения*, в которой дано описание всех callback-функций с указанием макроса, который необходимо определить в **lusb_config_init.h** для их включения. Также там указывается, откуда производится вызов этой функции – из прерывания или из *lusb_progress()*.

6 Состояние usb-устройства

USB-устройство может находится в одном из состояний, определяемых спецификацией USB. В lusb состояние устройства определяется набором флагов, определенных в таблице:

Таблица 6.1. Флаги состояния USB-устройства

Флаг	Описание
LUSB_DEVSTATE_ATTACHED [1)]	Устройство подключено к usb-шине
LUSB_DEVSTATE_POWERED [1)]	На устройство подано питание
LUSB_DEVSTATE_DEFAULT	После подачи питания устройство было сброшено (выполнен bus reset)
LUSB_DEVSTATE_ADDRESS	Устройству был назначен адрес, отличный от 0 (запрос SET_ADDRESS)
LUSB_DEVSTATE_CONFIGURED	Устройство сконфигурировано (установлена ненулевая конфигурация с помощью запроса SET_CONFIGURATION)
LUSB_DEVSTATE_SUSPENDED	Работа устройства приостановлена (из-за отсутствия активности на шине, перехода в спящий режим и т.д.)

[1] Как правило, программная часть стека не различает эти два состояния и изменяет сразу оба флага в зависимости от наличия питания USB-кабеля.

Для получения состояния устройства можно использовать функцию *lusb_dev_get_state()*. Наиболее актуально для приложения знать состояние флагов LUSB_DEVSTATE_CONFIGURED и LUSB_DEVSTATE_SUSPENDED. Пока устройство не сконфигурировано, все конечные точки usb-устройства, кроме управляющей, отключены. В этом состоянии нельзя вызывать функции для работы с конечными точками (**lusb_ep_xxx**), так как в общем случае состав и количество конечных точек не известны до установления конфигурации, и все они запрещены. При приостановке устройства обмен так же приостанавливается, кроме того перевод в это состояние можно использовать для перехода в энергосберегающий режим. Для проверки данных двух состояний можно использовать функции *lusb_dev_is_configured()* и *lusb_dev_is_suspended()* соответственно.

Кроме того, в случае, если необходимо провести какое-либо действие в ответ на изменение состояния устройство, приложение может реализовать callback-функцию *lusb_appl_cb_devstate_ch()*. В функцию передается старое и новое состояние устройства (как правило, функция вызывается в контексте прерывания, однако это может зависеть от реализации порта). Так же приложение может зарегистрировать функции, вызываемые при сбросе шины (*lusb_appl_cb_bus_reset()*), по которому устройство переходит в состояние по-умолчанию (флаг LUSB_DEVSTATE_DEFAULT), при изменении подключения (*lusb_appl_cb_conch()*) или при изменении состояния suspend (*lusb_appl_cb_suspch()*).

Пример реализации действий на изменение состояния устройства.

```
//callback на изменение состояния устройства
void lusb_appl_cb_devstate_ch(uint8_t old_state, uint8_t new_state)
{
  if ((new_state & LUSB_DEVSTATE_CONFIGURED)
        && !(old_state & LUSB_DEVSTATE_CONFIGURED))
  {
        //перешли в сконфигурированное состояние

  }
  else if (!(new_state & LUSB_DEVSTATE_CONFIGURED)
        && (old_state & LUSB_DEVSTATE_CONFIGURED))
  {
        //вышли из сконфигурированного состояния
  }
}
```

7 Обработка пользовательских управляющих запросов

Все стандартные управляющие запросы USB обрабатывает стек lusb. Однако, пользователь может реализовать обработку собственных запросов в приложении. Для этого приложение может реализовать callback-функции *lusb_appl_cb_custom_ctrlreq_rx()* и/или *lusb_appl_cb_custom_ctrlreq_tx()*.

Все управляющие запросы по направлению передачи данных делятся на два типа.

В первом случае данные передаются от хоста к устройству (сюда же относятся запросы без данных). В этом случае ядро обрабатывает запрос после приема данных. Если запрос стандартный, ядро обрабатывает его само, иначе оно вызывает функцию *lusb_appl_cb_custom_ctrlreq_rx()* (если включен макрос LUSB_APPL_CB_CUSTOM_CTRLREQ_RX). Все параметры запроса USB передаются в первом параметре в виде указателя на структуру *t_lusb_req*. Указатель на буфер, в котором сохранены данные запроса, передаются в качестве второго параметра. Данные запроса сохраняются ядром в стандартном буфере 0-ой конечной точки, размер которого определяется параметром LUSB_EP0_RX_BUF_SIZE. Если приложение успешно обработало заданный запрос, оно должно вернуть LUSB_ERR_SUCCESS, иначе — LUSB_ERR_UNSUPPORTED_REQ.

В случае, если данные управляющего запроса передаются от устройства к хосту, по приему запроса данные должны быть подготовлены для передачи. Если запрос не стандартный, ядро вызывает функцию *lusb_appl_cb_custom_ctrlreq_tx()* (если включен макрос LUSB_APPL_CB_CUSTOM_CTRLREQ_TX). Параметры запроса также передаются в первом параметре. Второй параметр представляет собой указатель на переменную, содержащую длину буфера на передачу. Если приложение поддерживает этот запрос, оно должно записать в эту переменную значение длины большее или равное нулю. При этом в качестве возвращаемого значения функция должна вернуть указатель на буфер для передачи, что позволяет приложению использовать свой буфер и избежать лишнего копирования данных в стандартный буфер. При желании, можно так же использовать стандартный буфер на передачу **lusb_ctrl_tx_buf** размер которого определяется параметром LUSB_EP0_TX_BUF_SIZE. Если приложение не поддерживает запрос, то не должно изменять значение длины, и возвращаемое значение функции не используется.

Таблица 7.1. Поля структуры `t_lusb_req`

<table>
<tr><th>Поле</th><th>Описание</th></tr>
<tr><td>req_type</td><td>Тип запроса (из спецификации USB). Имеет следующий формат:
<table>
<tr><th>7</th><th>6 5</th><th>4 0</th></tr>
<tr><td>Dir</td><td>Type</td><td>Recipient</td></tr>
</table>
Dir – направление передачи данных в запросе
USB_REQ_REQTYPE_DIR_IN (0x80) – от устройства к хосту
USB_REQ_REQTYPE_DIR_OUT (0) – от хоста к устройству
Type – тип запроса:
USB_REQ_REQTYPE_TYPE_STD – стандартный запрос
USB_REQ_REQTYPE_TYPE_CLASS – запрос, определяемый классом
USB_REQ_REQTYPE_TYPE_VENDOR – пользовательский запрос
Recipient – определяет, кому предназначен запрос
USB_REQ_REQTYPE_RECIPIENT_DEVICE – устройству
USB_REQ_REQTYPE_RECIPIENT_INTERFACE – интерфейсу
USB_REQ_REQTYPE_RECIPIENT_ENDPOINT – конечной точке</td></tr>
<tr><td>request</td><td>Код запроса</td></tr>
<tr><td>val_l</td><td>Младшая половина поля wValue (зависит от запроса)</td></tr>
<tr><td>val_h</td><td>Старшая половина поля wValue (зависит от запроса)</td></tr>
<tr><td>index</td><td>Индекс или смещение (зависит от запроса)</td></tr>
<tr><td>length</td><td>Количество байт данных, запрашиваемых или передаваемых хостом</td></tr>
</table>

8 Обработка пользовательских запросов по пакетам («на лету»)

Описанный предыдущей главе способ обработки пользовательских управляющих запросов предполагает, что все данные управляющего запроса передаются целиком и не превышают заданного размера временного буфера конечной точки. В этом режиме передача данных скрыта от пользователя. Приложение обрабатывает уже полностью принятый массив данных в случае передачи от хоста к устройству, или подготавливает этот массив один раз целиком перед отправкой.

Однако может возникнуть ситуация, когда по управляющей конечной точки передается массив данных и требуется обрабатывать или подготавливать данные по пакетам (пока другие передаются) а не целиком сразу.

Этот режим включается для управляющих запросов на прием при определении макроса `LUSB_EP0_RX_PARTIAL`, а для управляющих запросов на передачу — `LUSB_EP0_TX_PARTIAL`. При этом данный режим включается для всех пользовательских запросов (Vendor Request) указанного направления, то есть описанный в предыдущей главе механизм не может быть использован — обработка части пользовательских запросов одним способом, а части другим невозможна (при этом стандартные запросы обрабатываются всегда целиком).

Прием в пакетном режиме осуществляется следующим образом. После приема управляющего запроса, когда приходит каждый новый пакет, стек считывает его во временный буфер и вызывает функцию приложения *`lusb_appl_cb_custom_ctrlreq_partial_rx()`*. В функцию помимо указателя на структуру с описанием запроса, передается указатель на буфер с принятыми данными, количество принятых данных в буфера и смещения этих данных относительно начала.

То есть, если размер пакета равен 64, то *`lusb_appl_cb_custom_ctrlreq_partial_rx()`* будет вызвана со смещением 0 для первого пакета, 64 для второго, 128 для третьего и так далее. Размер будет всегда равен 64.

Входной буфер может быть обработан сразу внутри *`lusb_appl_cb_custom_ctrlreq_partial_rx()`* или где либо в другом месте программы. В обоих случаях, когда содержимое буфера больше не нужно, должна быть вызвана функция *`lusb_ctrl_rx_packet_done()`*, что укажет, что стек может принимать следующий пакет. Повторный вызов *`lusb_appl_cb_custom_ctrlreq_partial_rx()`* может произойти только после того, как предыдущий был обработан. Следует не забывать вызывать *`lusb_ctrl_rx_packet_done()`* на каждый принятый пакет, в противном случае новый пакет не будет принят.

При передаче данных, после того как стек принял управляющий запрос, он вызывает пользовательскую функцию *`lusb_appl_cb_custom_ctrlreq_partial_tx()`*, сообщая приложению, что оно

должно подготовить буфер на передачу. После того, как приложение подготовит буфер, оно вызывает *lusb_ctrl_tx_packet()*, сообщая стеку буфер на передачу и его размер. По завершению передачи стек снова вызывает *lusb_appl_cb_custom_ctrlreq_partial_tx()*, в которую помимо указателя на структуру с параметрами запроса передается так же смещение от начала управляющего запроса. Передача завершается в одном из случаев:

- были переданы все запрашиваемые данные (поле `length` из *t_lusb_req*).
- *usb_ctrl_tx_packet()* была вызвана с размером, не кратным размеру конечной точки (по завершению передачи некратного пакета)
- *usb_ctrl_tx_packet()* была вызвана с нулевым размером

9 Передача данных

Для передачи данных используются конечные точки, за исключением нулевой, которая используется для передачи управляющих запросов. Каждой ненулевой конечной точке должен соответствовать свой дескриптор, в котором содержится ее тип, адрес и размер пакета. Дескрипторы конечных точек находятся в общем массиве, задающем конфигурацию, **lusb_cfgs** (включает дескриптор конфигурации, интерфейсов, конечных точек), который определен в **lusb_descriptors.c**. Задаваемые параметры определяются макросами из **lusb_config_init.h**.

При работе с конечными точками, приложение, при вызове функций lusb, использует не адрес конечной точки, а ее индекс. Индекс конечной точки соответствует ее номеру в массиве дескрипторов. При этом номер начинается с 0 (что соответствует первой точки для передачи данных, не путать с нулевой конечной точкой!).

В базовых настройках стека используются две конечные точки типа bulk – на передачу и на прием, при этом им соответствуют индексы `LUSB_TX_EP_IND` и `LUSB_RX_EP_IND`.

Для работы с конечными точками предназначен ряд функций **lusb_ep_xxx**. Эти функции можно вызвать, только когда устройство находится в сконфигурированном состоянии (для проверки этого может использоваться функция *`lusb_dev_is_configured()`*), иначе они вернут ошибку `LUSB_ERR_DEV_UNCONFIGURED`.

Передача данных может быть осуществлена как с использованием аппаратного DMA (если поддерживается аппаратурой и портом), так и по прерываниям (логика передачи реализована в ядре). Указать, по каким конечным точкам данные передаются с использованием аппаратного DMA можно с помощью определения `LUSB_DMA_EP_MSK` из **lusb_config_init.h**, в виде объединённых через «или» макросов `LUSB_EPFLAG(адрес кт)`, где `адрес кт` – стандартный для usb адрес конечной точки (из дескриптора). При этом интерфейс для передачи остается одинаковым со стороны приложения, в независимости от механизма передачи.

Для осуществления приема/передачи приложение должно добавить задание на обмен (дескриптор для передачи данных – data descriptor (dd)), вызвав функцию *`lusb_ep_add_dd()`*. При этом приложение указывает буфер, содержащий подготовленные данные на передачу или подготовленный для приема данных, и его размер. Направление передачи определяется конечной точкой (ее дескриптором).

Для каждой конечной точки можно добавлять несколько заданий (буферов), так как для каждой конечной точки существует свой связанный список заданий, и новое задание добавляется в конец этого списка. Размер буфера должен быть кратен размеру конечной точки (за исключением случая передачи последнего пакета меньшей длины для указания окончания передачи).

Количество обслуживаемых дескрипторов для конечной точки можно определить с помощью функции *`lusb_ep_get_dd_in_progress()`* (это количество увеличивается при вызове *`lusb_ep_add_dd()`*, а уменьшается при завершении передачи буфера данных, определяемого заданием). С помощью этой функции можно отслеживать момент,

когда обработка предыдущего дескриптора завершена (как только кол-во дескрипторов стало меньше числа добавленных).

Состояние задания по передачи данных описывается структурой *t_lusb_dd_status*, которая включает количество переданных байт, текущий адрес записи/чтения и состояние задания. Состояние последнего обработанного дескриптора сохраняется в поле dd информации о контрольной точке, указатель на которую можно получить, вызвав *lusb_ep_get_info()*. Состояние обрабатываемого в данный момент задания можно получить, вызвав *lusb_ep_get_dd_status()*.

При приеме данных, в случае, если был принят пакет меньшей длины чем размер конечной точки, то по завершению приема данного пакета статус текущего задания становится равным LUSB_DDSTATUS_CPL_UNDERRUN и следующий пакет будет уже сохранен в буфер, соответствующий следующему заданию (дескриптору).

Если требуется принудительно завершить передачу данных по конечной точке при условии, что не все задание по передачи данных выполнены, можно вызвать функцию *lusb_ep_clear_dma()* в результате чего будет очищен буфер конечной точки, а так же весь список дескрипторов.

Таблица 9.1.Поля структуры `t_lusb_dd_status`

<table>
<tr><th>Поле</th><th>Описание</th></tr>
<tr><td>status</td><td>Состояние задания. Одна из констант:
<table>
<tr><th>Константа</th><th>Описание</th></tr>
<tr><td>LUSB_DDSTATUS_NOTSERVICED</td><td>Выполнение данного задания еще не начиналось</td></tr>
<tr><td>LUSB_DDSTATUS_IN_PROGRESS</td><td>Задание в настоящее время обрабатывается</td></tr>
<tr><td>LUSB_DDSTATUS_CPL_SUCCESS</td><td>Задание было успешно завершено (все данные переданы)</td></tr>
<tr><td>LUSB_DDSTATUS_CPL_UNDERRUN</td><td>Задание завершено по причине приема пакета меньшей длины</td></tr>
<tr><td>LUSB_DDSTATUS_CPL_OVERRUN</td><td>Задание завершено с ошибкой - последний пакет оказался не выравнен по границе буфера</td></tr>
<tr><td>LUSB_DDSTATUS_SYSTEM_ERR</td><td>При обработке задания произошла системная ошибка</td></tr>
</table></td></tr>
<tr><td>flags</td><td>Флаги, в настоящее время зарезервированное поле</td></tr>
<tr><td>last_addr</td><td>Адрес, по которому будет произведена запись/чтение следующего байта при обмене. При завершении задания, указывает на байт, следующий за последним принятым/переданным байтом</td></tr>
<tr><td>trans_cnt</td><td>Количество реально переданных байт</td></tr>
<tr><td>length</td><td>Длина буфера, который необходимо передать в соответствии с заданием</td></tr>
</table>

Таблица 9.2.Поля структуры `t_lusb_ep_info`

<table>
<tr><th>Поле</th><th>Описание</th></tr>
<tr><td>descr</td><td>Указатель на дескриптор, описывающий данную конечную точку</td></tr>
<tr><td>flags</td><td>Флаги, описывающие состояние конечной точки (резерв)</td></tr>
<tr><td>dd_flags</td><td>Флаги, описывающие события, произошедшие при обмене данными:
<table>
<tr><th>Константа</th><th>Описание</th></tr>
<tr><td>LUSB_EPFLAGS_DD_NOTFOUND</td><td>Произошло событие, что не был найден нужный дескриптор</td></tr>
<tr><td>LUSB_EPFLAGS_UNDERRUN_OCCUR</td><td>Был принят пакет меньшего размера</td></tr>
<tr><td>LUSB_EPFLAGS_ERR_OCCUR</td><td>Произошла ошибка</td></tr>
<tr><td></td><td></td></tr>
</table></td></tr>
<tr><td>dd_req_cnt[2)]</td><td>Количество запрошенных заданий на обмен данными</td></tr>
<tr><td>dd_cpl_cnt[2)]</td><td>Количество выполненных заданий на обмен данными</td></tr>
<tr><td>pkt_rdy_cnt[1) 2)]</td><td>Количество пакетов, которые можно принять/передать</td></tr>
<tr><td>pkt_cpl_cnt[1) 2)]</td><td>Количество пакетов, которые были приняты/переданы</td></tr>
<tr><td>cur_dd_addr[1) 2)]</td><td>Указатель на текущий дескриптор задания на обмен данными</td></tr>
<tr><td>dd</td><td>Состояние последнего выполненного задания на обмен данными</td></tr>
</table>

[1)] поля используются только при реализации обмена софтом, а не при аппаратном dma

[2)] поля предназначены для использования стеком, а не приложением

10 Список функций

10.1 Функции инициализации и продвижения стека

10.1.1 Инициализация стека

Формат: `int lusb_init(void)`
Назначение: Функция инициализирует все аппаратные настройки usb-контроллера и переводит устройство в неподключенное состояние. Функция должна быть вызвана перед вызовом остальных функций стека lusb. Вызывается как правило один раз.
Возвращаемое значение: Код ошибки `LUSB_ERR_SUCCESS` – инициализация прошла успешно `LUSB_ERR_MODULE_TST` – ошибка прохождения теста USB-модуля … - возможны другие коды (в зависимости от порта)

10.1.2 Подключение к шине

Формат: `void lusb_connect(uint8_t con)`
Назначение: Функция выполняет подключение (отключение) устройства к шине USB. Вызов этой функции приводит к подключению подтягивающего резистора соответствующей к линии шины USB, что сообщает хосту, что подключено новое устройство.
Передаваемые параметры: `con` – 0 – устройство должно быть отключен от шины USB 1 – устройство должено быть подключен к шине USB

10.1.3 Продвижение стека

Формат: `void lusb_progress(void)`
Назначение: Функция выполняет фоновые задачи стека. Должна вызываться периодически после вызова *`lusb_init()`* для продвижения стека.

10.2 Функции для отслеживания состояния устройства

10.2.1 Получение состояния устройства

Формат:	`uint8_t lusb_dev_get_state(void)`
Назначение: Получить состояние usb-устройства. Состояние описывается набором флагов. Подробнее смотри главу *Состояние usb-устройства*.	
Возвращаемое значение: Состояние устройства в виде флагов.	

10.2.2 Проверка, сконфигурировано ли устройство

Формат:	`uint8_t lusb_dev_is_configured(void)`
Назначение: Проверка, сконфигурировано ли устройство.	
Возвращаемое значение: > 0 – устройство сконфигурировано 0 – устройство не сконфигурировано.	

10.2.3 Проверка, приостановлена ли работа на шине USB

Формат:	`uint8_t lusb_dev_is_suspended(void)`
Назначение: Проверка, переведено ли устройство в приостановленное состояние.	
Возвращаемое значение: 1 – работа устройства приостановлена. 0 – нормальный режим	

10.3 Функции для работы с конечными точками

10.3.1 Добавление задания на прием или передачу данных

Формат: `int lusb_ep_add_dd(uint8_t ep_ind, void* buf, uint32_t length, uint32_t flags);`

Назначение: Добавление для конечной точки задания (дескриптора) на прием/передачу данных. Задание помещается в конец очереди (если есть добавленные, но еще не обработанные задания). Направление передачи определяется дескриптором конечной точки с соответствующим индексом.

При стандартных настройках индекс `LUSB_TX_EP_IND` соответствует конечной точке на передачу данных от устройства к хосту, а индекс `LUSB_RX_EP_IND` — конечной точки на прием данных от хоста устройством

Передаваемые параметры:

`ep_ind` – индекс конечной точки (начиная с 0)

`buf` – указатель на буфер, либо содержащий подготовленные данные для передачи, либо подготовленных для приема данных.

`length` – длина буфера в байтах. При приеме, должна быть кратна размеру пакета конечной точки. При передаче так же должна быть кратна, за исключением случая передачи пакета меньшей длины, указывающего конец передачи.

`flags` – флаги (резерв).

Возвращаемое значение:

`LUSB_ERR_SUCCESS` – дескриптор успешно добавлен

`LUSB_ERR_DEV_UNCONFIGURED` – устройство находится не в сконфигурированном состоянии. Работа с конечными точками невозможна.

`LUSB_ERR_EP_INDEX` – задан неверный индекс конечной точки.

`LUSB_ERR_NO_FREE_DESCR` – нет свободного места для добавления дескриптора (количество обрабатываемых дескрипторов уже равно максимальному).

`LUSB_ERR_EP_UNSUP_DMA` – для конечной точки с помощью `LUSB_DMA_EP_MSK` задан режим аппаратного DMA, однако данный режим исключен из стека или не поддерживается портом.

`LUSB_ERR_EP_UNSUP_SIO` – для конечной точки задан режим передачи софтом, однако данный режим выключен в настройках стека.

10.3.2 Получение количества заданий находящихся в процессе обслуживания

Формат: `int lusb_ep_get_dd_in_progress(uint8_t ep_ind)`
Назначение: Данная функция возвращает количество заданий (дескрипторов) для заданной конечной точки, которые были добавлены в очередь, но обработка (передача данных в соответствии с заданием) которых еще не завершена. Может использоваться для определения момента, когда закончилась передача данных в соответствии с заданием и соответствующий буфер может быть использован для чтения принятых данных или записи следующего блока данных на передачу.
Передаваемые параметры: `ep_ind` – индекс конечной точки (начиная с 0)
Возвращаемое значение: Если ≥ 0 – количеств дескрипторов в очереди Если < 0 – код ошибки: `LUSB_ERR_DEV_UNCONFIGURED` – устройство находится не в сконфигурированном состоянии. Работа с конечными точками невозможна. `LUSB_ERR_EP_INDEX` – задан неверный индекс конечной точки.

10.3.3 Получение состояния текущего задания на передачу

Формат: `int lusb_ep_get_dd_status(uint8_t ep_ind, t_lusb_dd_status* dd_st)`
Назначение: Данная функция возвращает информацию о состоянии текущего задания (дескриптора) по передаче/приему данных.
Передаваемые параметры: `ep_ind` – индекс конечной точки (начиная с 0) `dd_st` – указатель на структуру *`t_lusb_dd_status`*, в которой будет сохранено состояние текущего задания.
Возвращаемое значение: `LUSB_ERR_SUCCESS` – функция выполнена успешно `LUSB_ERR_DEV_UNCONFIGURED` – устройство находится не в сконфигурированном состоянии. Работа с конечными точками невозможна. `LUSB_ERR_EP_INDEX` – задан неверный индекс конечной точки. `LUSB_ERR_DESCR_NOT_FOUND` – соответствующий дескриптор не найден – в случае, если передача данных сейчас не идет по заданной конечной точке

10.3.4 Получение информации о состоянии конечной точки

Формат: **t_lusb_ep_info lusb_ep_get_info(uint8_t ep_in)**

Назначение: Данная функция возвращает указатель на структуру с информацией о состоянии конечной точки, которая так же включает состояние последнего завершенного задания по передаче данных.

Внимание!: функция сейчас реализована в виде макроса без проверки индекса конечной точки

Внимание!: функция возвращает указатель на структуру, используемую ядром! Приложение не должно изменять поля в данной структуре. При этом ядро может изменять поля данной структуры. При необходимости приложение может скопировать данные структуры.

Передаваемые параметры:

`ep_ind` – индекс конечной точки (начиная с 0)

Возвращаемое значение:

Если устройство не сконфигурировано – возвращает `NULL`, иначе – указатель на структуру *`t_lusb_ep_info`*, содержащую информацию о конечной точке.

10.3.5 Очистка списка заданий для конечной точки

Формат: int lusb_ep_clear_dma(void)

Назначение: Данная функция очищает буфер конечной точки и весь список заданий на прием/передачу для нее. Точка на это время может быть временно запрещена. Используется для того, чтобы прервать передачу данных не дожидаясь завершения текущих заданий.

Возвращаемое значение:

`LUSB_ERR_SUCCESS` – дескриптор успешно добавлен

`LUSB_ERR_DEV_UNCONFIGURED` – устройство находится не в сконфигурированном состоянии. Работа с конечными точками невозможна.

`LUSB_ERR_EP_INDEX` – задан неверный индекс конечной точки.

`LUSB_ERR_EP_UNSUP_DMA` – для конечной точки с помощью `LUSB_DMA_EP_MSK` задан режим аппаратного DMA, однако данный режим исключен из стека или не поддерживается портом.

`LUSB_ERR_EP_UNSUP_SIO` – для конечной точки задан режим передачи софтом, однако данный режим выключен в настройках стека.

10.4 Вспомогательные функции для обработки управляющих запросов по пакетам

10.4.1 Признак завершения обработки принятого пакета по управляющей контрольной точке

Формат: `int lusb_ctrl_rx_packet_done(void)`

Назначение: Данная функция сообщает стеку, что принятый пакет данных пользовательского запроса был обработан приложением и стек может считывать следующий принимаемый пакет.

Должна вызываться на каждый вызов стеком функции приложения *lusb_appl_cb_custom_ctrlreq_partial_rx()*, для которой приложение вернуло LUSB_ERR_SUCCESS.

Возвращаемое значение:

LUSB_ERR_SUCCESS – выполнено успешно

10.4.2 Передача следующего пакета по управляющей контрольной точке

Формат: `int lusb_ctrl_tx_packet(const uint8_t* buff, uint16_t size)`

Назначение: Данная функция сообщает стеку, что приложение подготовило пакет для передачи по управляющей конечной точки. Вызывается после вызова стеком *lusb_appl_cb_custom_ctrlreq_partial_tx()*. Функция указывает какой буфер и какого размера должен быть передан. Содержимое буфера не должно изменяться до следующего управляющего запроса или следующего вызова .

Параметры:

buf – указатель буфер с данными, которые должны быть переданы

size – размер данных в буфере

Возвращаемое значение:

LUSB_ERR_SUCCESS – выполнено успешно

10.5 Callback-функции приложения

10.5.1 Обработка управляющих запросов с приемом данных

Формат: `int lusb_appl_cb_custom_ctrlreq_rx(t_lusb_req* req, uint8_t* buf)`
Назначение: Данная функция вызывается ядром, если принят нестандартный (не обрабатываемый ядром) управляющий запрос, в котором данные передаются от хоста к устройству (или запросы без данных). Функция вызывается, когда все данные уже успешно приняты, и может использоваться для обработки приложением пользовательских управляющих запросов. Если запрос обработан, то функция должна вернуть `LUSB_ERR_SUCCESS`, в результате чего ядро пошлет завершающий пакет. Если приложение не поддерживает данный запрос или запрос не может быть завершен, то функция должна вернуть `LUSB_ERR_UNSUPPORTED_REQ`, что будет указанием ядру послать пакет STALL на данный запрос.
Макрос для подключения функции: `LUSB_APPL_CB_CUSTOM_CTRLREQ_RX`
Контекст вызова: Из *`lusb_progress()`*, если не определены одновременно `LUSB_INTERRUPT` и `LUSB_CTLREQ_PROC_IN_INT`, иначе – из прерывания
Передаваемые параметры: `req` – указатель на стандартные параметры USB-запроса в виде структуры *`t_lusb_req`* `buf` – принятый массив с данными (длиной `req->length`)
Возвращаемое значение: `LUSB_ERR_SUCCESS` – запрос обработан успешно `LUSB_ERR_UNSUPPORTED_REQ` – запрос не поддерживается (или ошибка обработки).

10.5.2 Обработка управляющих запросов с передачей данных

Формат: `const void* lusb_appl_cb_custom_ctrlreq_tx (t_lusb_req* req, int* length)`

Назначение: Данная функция вызывается ядром, если принят нестандартный (не обрабатываемый ядром) управляющий запрос, в котором данные передаются от устройства к хосту. В данной функции приложение может подготовить данные для передачи, в ответ на пользовательский запрос.

Если приложение обрабатывает запрос, то оно должно записать в переменную `length` длину буфера (больше или равна 0) и вернуть указатель на буфер для передачи.

В качестве буфера можно использовать стандартный определенный в ядер буфер **lusb_ctrl_tx_buf** размера `LUSB_EP0_TX_BUF_SIZE`.

Если запрос не поддерживается, то переменная `length` не изменяется. Перед вызовом данной функции ядро записывает в `length` отрицательное значение, и если оно осталось отрицательным после вызова функции, то ядро посылает пакет STALL, указывающий, что запрос не поддерживается. Если длина ≥ 0, то ядро передает подготовленные данные. При этом длина переданных данных будет минимальным значением из `length` и `req->length`.

Нулевая длина означает, что будет передан пакет нулевой длины, при этом указатель на буфер не имеет значения.

Макрос для подключения функции: `LUSB_APPL_CB_CUSTOM_CTRLREQ_TX`

Контекст вызова: Из *`lusb_progress()`*, если не определены одновременно `LUSB_INTERRUPT` и `LUSB_CTLREQ_PROC_IN_INT`, иначе – из прерывания

Передаваемые параметры:

`req` – указатель на стандартные параметры USB запроса в виде структуры *`t_lusb_req`*

`length` – по данному адресу функция возвращает длину (≥ 0) буфера для передачи в байтах. Если длина остается без изменений или передается значение меньше 0, это означает, что запрос не поддерживается.

Возвращаемое значение:

Указатель на буфер с данными для передачи, если `length` > 0, иначе не имеет значения.

10.5.3 Обработка принятого пакета данных управляющего запроса

Формат: `int lusb_appl_cb_custom_ctrlreq_partial_rx(t_lusb_req* req, uint8_t* buf, uint16_t offset, uint16_t length)`

Назначение: Данная функция вызывается ядром, если после приема пользовательского запроса (Vendor Request) был принят и прочитан пакет данных и разрешен режим обработки управляющих запросов на прием по пакетам.

Если приложение поддерживает запрос, то оно должно вернуть `E124_ERR_SUCCESS`. После того, как данные из буфера будут обработаны, приложение должно вызвать функцию *`lusb_ctrl_rx_packet_done()`*, что будет означать, что стек может принимать следующий пакет. *`lusb_ctrl_rx_packet_done()`* может быть вызвана из любого места программы, но главное, чтобы на каждый вызов стеком *`lusb_appl_cb_custom_ctrlreq_partial_rx()`*, приложение вызывало в ответ *`lusb_ctrl_rx_packet_done()`*. В противном случае буфер управляющей конечной точки не будет освобожден и следующие пакеты не будут приниматься.

Если приложение не поддерживает запрос, то оно должно вернуть `LUSB_ERR_UNSUPPORTED_REQ`. В этом случае вызывать *`lusb_ctrl_rx_packet_done()`* не обязательно.

Макрос для подключения функции: `LUSB_EP0_RX_PARTIAL`

Контекст вызова: Из *`lusb_progress()`*

Передаваемые параметры:

`req` – указатель на стандартные параметры USB запроса в виде структуры *`t_lusb_req`*

`buf` – указатель на буфер, содержащий прочитанные данные

`offset` – смещение принятого пакета относительно начала управляющего запроса

`length` – размер принятого пакет

Возвращаемое значение:

`LUSB_ERR_SUCCESS` – приложение приступило к обработке запроса

`LUSB_ERR_UNSUPPORTED_REQ` – запрос не поддерживается (или ошибка обработки).

10.5.4 Запрос на подготовку пакета данных управляющего запроса на передачу

Формат: `int lusb_appl_cb_custom_ctrlreq_partial_tx(t_lusb_req* req, uint16_t offset)`
Назначение: Данная функция вызывается ядром, для указания того, что приложение должно подготовить пакет данных на передачу по нулевой конечной точки. Вызывается, срезу после приема управляющего запроса с направлением передачи от устройства в хост, а так же после завершения передачи предыдущего пакета, если новый пакет принят. Когда пакет будет подготовлен, приложение вызывает функцию *lusb_ctrl_tx_packet()*. Переданный буфер не должен изменяться до следующего управляющего запроса. Если приложение поддерживает запрос, то оно должно вернуть E124_ERR_SUCCESS. Иначе возвращается LUSB_ERR_UNSUPPORTED_REQ.
Макрос для подключения функции: LUSB_EP0_RX_PARTIAL
Контекст вызова: Из *lusb_progress()*
Передаваемые параметры: req – указатель на стандартные параметры USB запроса в виде структуры *t_lusb_req* offset – смещение принятого пакета относительно начала управляющего запроса
Возвращаемое значение: LUSB_ERR_SUCCESS – приложение приступило к обработке запроса LUSB_ERR_UNSUPPORTED_REQ – запрос не поддерживается (или ошибка обработки).

10.5.5 Обработка события сброса шины

Формат: `void lusb_appl_cb_bus_reset(void)`
Назначение: Данная функция вызывается ядром, если по USB-шине пришел сигнал сброса шины (bus reset). По этому сигналу устройство переводится ядром в состояние LUSB_DEVSTATE_DEFAULT.
Макрос для подключения функции: LUSB_APPL_CB_BUS_RESET
Контекст вызова: Из прерывания, если определен LUSB_INTERRUPT. **Примечание**: может зависеть от реализации порта.

10.5.6 Обработка события подключения или отключения от шины

Формат: `void lusb_appl_cb_conch (uint8_t con)`
Назначение: Данная функция вызывается ядром при изменении состояния подключения к шине. Как правило, подключено ли устройство, определяется по наличию питания на USB-шине.
Макрос для подключения функции: `LUSB_APPL_CB_CONNECT_CHANGED`
Контекст вызова: Из прерывания, если определен `LUSB_INTERRUPT`. **Примечание**: может зависеть от реализации порта – в lpc17xx вызывается из *`lusb_progress()`*.
Передаваемые параметры: `con` – 1 – произошло подключение к шине USB. 0 – произошло отключение от шины.

10.5.7 Обработка события перехода в приостановленное состояние

Формат: `void lusb_appl_cb_suspch (uint8_t susp)`
Назначение: Данная функция вызывается ядром при переходе USB-устройства в приостановленное (suspend) состояние или при выходе из него (пробуждении).
Макрос для подключения функции: `LUSB_APPL_CB_SUSPEND_CHANGED`
Контекст вызова: Из прерывания, если определен `LUSB_INTERRUPT`. **Примечание**: может зависеть от реализации порта.
Передаваемые параметры: `susp` – 1 – произошел переход в приостановленное состояние. 0 – пробуждение из приостановленного состояния.

10.5.8 Изменение состояния USB-устройства

Формат: `void lusb_appl_cb_devstate_ch (uint8_t old_state, uint8_t new_state)`
Назначение: Данная функция вызывается ядром при любом изменении состояния usb-устройства. Подробнее см. главу *Состояние usb-устройства*.
Макрос для подключения функции: `LUSB_APPL_CB_DEVSTATE_CHANGED`
Контекст вызова: Из прерывания, если определен `LUSB_INTERRUPT`. **Примечание**: может зависеть от реализации порта.
Передаваемые параметры: `old_state` – флаги, описывающие состояние устройства до изменения. `new_state` – флаги, описывающие состояние устройства после изменения.

10.5.9 Обработка событий передачи данных

Формат: `void lusb_appl_cb_dma_event (uint8_t ep_ind, uint8_t event, t_lusb_dd_status* pDD)`

Назначение: Данная функция вызывается ядром при возникновении события при передаче данных по конечной точке (не управляющей). Данная функция вызывается как в случае, если контроллер поддерживает аппаратный DMA, так и при передаче данных софтом.

Возможны следующие варианты событий:

- `LUSB_DMA_EVENT_EOT` – завершена передача данных в соответствии с заданием. Состояние завершенного задания передается через параметр `pDD`.
- `LUSB_DMA_EVENT_NODD` – не найден следующее задание. Данное событие возникает в случае, если хост запросил данные на прием или передачу, но стек не нашел дескриптора, чтобы обработать данных запрос. В результате этого, устройство ответило пакетом NACK на запрос. Происходит как правило, если закончилась цепочка заданий и пришел новый запрос до добавления следующего задания
- `LUSB_DMA_EVENT_ERR` – произошла системная ошибка при передачи данных по данной конечной точке

Макрос для подключения функции: `LUSB_APPL_CB_DMA_EVENT`

Контекст вызова: Из прерывания, если определен `LUSB_INTERRUPT`.

Передаваемые параметры:

`ep_ind` – индекс конечной точки, для которой произошло событие

`event` – код произошедшего события

`pDD` – если `event` равно `LUSB_DMA_EVENT_EOT`, то указатель на структуру *`t_lusb_dd_status`*, содержащую информацию о выполненном задании на передачу.

10.5.10 Получение серийного номера устройства

Формат: `char* lusb_app_cb_get_serial(void);`

Назначение: Данная функция вызывается ядром при обработке управляющего запроса на получение строкового дескриптора, содержащего серийный номер устройства. Данная функция позволяет пользователю самостоятельно сформировать строку с серийным номером (например, прочитать и ПЗУ).

Функция должна вернуть указатель на строку, содержащую серийный номер в ASCII кодировке (только английский язык). Стек сам формирует на основе этой строки дескриптор. UTF-16 символы формируются просто добавлением нуля в старший байт.

Макрос для подключения функции: `LUSB_USE_MAN_GEN_SERIAL`

Контекст вызова: Из *`lusb_progress()`*.

Возвращаемое значение:

указатель на строку с серийным номером устройства