KS Labs USB Demo Board
Описание программы

Версия документа: 1.3
Октябрь 2002г.

Это техническое описание включает в себя описание программы **USB Game Pad,** а также описывает стандартные запросы и запросы характерныые для устройств HID-класса.

Вся информация, представленная в этом документе, может быть использована только для ознакомительных целей. Ни одна из глав этого документа не может быть использована в какой либо форме без предварительного согласия с **KS Labs**.

Организация данного руководства

Глава 1 Разрабатываем USB GAME PAD

Глава 2 Процедуры инициализации и обслуживания USB контроллера

Глава 3 USB стандартные запросы

Глава 4 Процедуры касающиеся класса устройства HID

Глава 1. Разрабатываем USB GAME PAD

Перед тем как начать изучение этой главы ознакомтесь с главами 8 и 9 в USB Specification 1.1.

Итак, чтобы система увидела Ваше устройство, Вам необходимо пройти процедуру: Enumeration process (глава 9, стр.195). Вкратце процедуру "Enumeration" можно описать так: HUB, к которому подключено устройство уведомляет Хост о подключении. После истечения 100 мс (пока питание не станет стабильным) Хост инициирует команды разрешения порта и выдает "Reset" на протяжении 10 мс. После снятия сигнала ‘Сброс’ порт переходит в состояние "Default state" (с 0 адресом). Через "Default adress" Хост начинает читать дескриптор устройства ("Device descriptor"), для определения максимальной величины пакета данных ("Data payload"). После этого Хост вычитывает дескриптор конфигурации ("Configuration descriptor"). В нем устройство возвращает Хосту: Configuration Descriptor, Interface Descriptors, Endpoint Descriptors и дескриптор характерный для класа к которому принадлежит это устройство.

Итак есть две задачи: подключить USBN9604 и пройти "Enumeration process".

Глава 1.1 Подключение USBN9604 к AT9058515

Для подключения USB контроллера USBN9604 к микроконтроллеру AT9058515 выберем режим "Multiplexed Mode" Mode 1: (Mode 1 подсоединен к шине GND, Mode 0 подсоединен к +5В).

В этом режиме используется двунаправленная шины адреса/данных AD7...AD0, сигнал CS - выбора микросхемы, RD - чтение, WR - запись, ALE - сигнал записи адреса (активный уровень логическая "1"). Временные диаграмы работы интерфейса в этом режиме даны на стр. 15 в USBN9603.pdf.

Процедура записи байта USBN9604:

```
void USB_WR(char U_ADDR,char U_DATA)
{
 DDRA=0xFF;          /*Настраиваем PORTA на выход для выдачи адреса*/
 SETBIT(PORTC,ALE); /*Устанавливаем ALE в 1 */
 CLRBIT(PORTC,CS);  /*Устанавливаем CS в 1 Выбираем USBN9604*/
 PORTA=U_ADDR;       /*Выдаем адрес регистра*/
 CLRBIT(PORTC,ALE); /*Сбрасываем ALE в '0', защелкивая адрес*/
 CLRBIT(PORTC,WR);  /*Сбрасываем WR в '0', инициируя запись*/
 PORTA=U_DATA;       /*Выставляем на шину данные*/
 SETBIT(PORTC,WR);  /* Защелкиваем данные '1' на WR*/
 SETBIT(PORTC,CS);  /* Устанавливаем CS в '1'-отключаем микросхему*/
}
```

Процедура чтения байта из USBN9604:

```
unsigned char USB_RD(char U_ADDR)
{
 char U_DATA;

 DDRA=0xFF;          /*Настраиваем PORTA на выход для выдачи адреса*/
 SETBIT(PORTC,ALE); /*Устанавливаем ALE в 1 */
 CLRBIT(PORTC,CS);  /*Устанавливаем CS в 1 Выбираем USBN9604*/
 PORTA=U_ADDR;       /*Выдаем адрес регистра*/
 CLRBIT(PORTC,ALE); /*Сбрасываем ALE в '0', защелкивая адрес*/
 DDRA=0x00;          /*Настраиваем PORTA на вход для приема данных*/
 CLRBIT(PORTC,RD);  /*Сбрасываем RD в '0', инициируя чтение*/
 _NOP();
 U_DATA=PINA;        /*Считываем данные с шины*/
 SETBIT(PORTC,RD);  /* Защелкиваем данные '1' на RD*/
 SETBIT(PORTC,CS);  /* Устанавливаем CS в '1'- отключаем микросхему*/
 return(U_DATA);
}
```

Работа с USB контроллером USBN9604 основана на обработке прерывания, которое устанавливаются в ходе обмена с Хостом. На основе этого выход INTR {USBN9604} подсоединен на вход INT0 {AT9058515}. Причем используется "Active low push-pull Out" (INTOC=11).

Глава 1.2. Написание дескриптора устройства

Что же такое дескриптор? Это блок данных хранящихся в устройстве, который описывает как работает устройство и его взаимосвязь с системой (Хостом). Хост запрашивает дескрипторы устройства во время "Enumeration process" для того, чтобы выделить соответствующие данному устройству ресурсы и загрузить соответствующий драйвер. По запросу Get Descriptor Хост получает три типа дескрипторов: Device, Configuration and String. Тип запрашиваемого дескриптора Хост указывает в старшем байте поля wValue. При запросе дескриптора конфигурации, устройство возвращает дескриптор Конфигурации, дескриптор Интерфейса и дескриптор Конечных точек (Endpoin descriptors). Далее приведен **Device descriptor:**

```
flash char DEV_DESC[] =
{
 DEV_LENGTH,  /*Величина этого дескриптора в байтах */
 DEVICE,      /*Указываем на DEVICE descriptor */
 0x00,0x01,   /*Версия USB спецификации в BCD 1.0 */
 0x00,        /*Код класса устройства */
 0x00,        /*Код подкласса устройства */
 0x00,        /*Код протокола */
 0x08,        /*Размер пакета для Endpoint 0 */
 0x4B,0x53,   /*KS Labs Vendor ID */
 0x00,0x01,   /*KS Labs Product ID */
 0x30,0x01,   /*KS Labs Revision ID 1.3 */
 MFG_STR_OFS, /*Индекс Производителя в String Descriptor */
 PID_STR_OFS, /*Индекс Продукта в String Descriptor */
 NBR_STR,     /*Индекс Серийного номера в String Descriptor */
 0x01         /*Количество возможных конфигураций */
}
```

Configuration descriptor:

```
flash char CFG_DESC[] =
{
 CFG_LENGTH,    /*Величина этого дескриптора в байтах */
 CONFIGURATION, /*Указываем на CONFIGURATION descriptor */
 0x22,0x00,     /*Длина дескриптора включая Interface,Endpoint,HID */
 0x01,        /*Количество Интерфейсов поддерживаемых Конфигурацией*/
 0x01,          /*Номер этой Конфигурации */
 CFG_STR_OFS,   /*Индекс Описания Конфигурации в String Descriptor */
 0x80,          /*Указываем, что берем питание с USB шины */
 100,           /*Указываем, что Max потребление (200 mA) */
```

Interface descriptor:

```
 INT_LENGTH,    /*Величина этого дескриптора в байтах */
 INTERFACE,     /*Указываем на INTERFACE descriptor */
 0x00,          /*Номер Интерфеса в данной конфигурации */
 0x00,          /*Альтернативный Интерфейс. '0'- Нет другого */
 0x01,         /*Количество Endpoints используемых этим Интерфейсом*/
 HIDCLASS,      /*Код класса - HID устройство */
 NOSUBCLASS,    /*Код подкласса - нет подкласса */
 0x00,          /*Код протокола-устройство не использует протокола */
 INT_STR_OFS,   /*Индекс Описания Интерфейса в String Descriptor */
```

Class specific descriptor (Hid Descriptor):

```
 HID_LENGTH,       /*Величина этого дескриптора в байтах */
 HID,              /*Указываем на HID descriptor - 21 */
 0x00,0x01,        /*Версия HID спецификации в BCD */
 0x00,         /*Страна для которой изготовлено изделие. 0-для всех*/
 1,            /*Количество HID дескрипторов. '1'-Report descriptor*/
 HIDREPORT,        /*Тип дескриптора класса '22'-HID REPORT */
 RPT_DESC_SIZE,0x0 /*Величина Report Descriptort в байтах */
```

Endpoint descriptor:

```
 END_LENGTH,       /*Величина этого дескриптора в байтах */
 ENDPOINT,         /*Указываем на ENDPOINT descriptor */
 0x85,             /*Указываем что это IN Endpoint с Адресом 5 */
 0x03,             /*Устанавливаем Тип передачи данных - Interrupt*/
 0x40,0x00,     /*Устанавливаем что Max выдаваемый размер пакета 64*/
 0xFF};            /*Время опроса Конечной точки - 255ms */
```

Константы используемые в дескрипторах:

```
#define DEV_LENGTH 18 /*Размер сегмента Device descriptor */
#define CFG_LENGTH 9  /*Размер сегмента Configuration descriptor */
#define INT_LENGTH 9  /*Размер сегмента Interface descriptor */
#define HID_LENGTH 9  /*Размер сегмента HID descriptor */
#define END_LENGTH 7  /*Размер сегмента Endpoint descriptor */
```

Коды классов

```
#define HIDCLASS 0x03
#define NOSUBCLASS 0x00
#define BOOTSUBCLASS 0x01
#define VENDORSPEC 0xFF
```

Типы Дескрипторов

```
#define DEVICE 0x01
#define CONFIGURATION 0x02
#define XSTRING 0x03
#define INTERFACE 0x04
#define ENDPOINT 0x05
#define HID 0x21
#define HIDREPORT 0x22
#define HIDPHYSICAL 0x23
```

Глава 1.3. Написание HID Report для USB GAME PAD

Минимально GAME PAD имеет 2 клавиши для изменения координат по оси X (влево/вправо), 2 клавиши для изменения координат по оси Y (вверх/вниз) и две клавиши: Fire/Jump и Trigger. Для кодирования положения по одной из осей координат достаточно 2 бита. Например: Вправо - "00", Влево - "11", в центре - "01". Соответственно, для кодирования положения в двух координатах необходимо 4 бита (2 бита - X ось, 2 бита - Y ось). Еще два бита необходимо для кодирования нажатия клавиш: Trigger и Fire ("1" - клавиша нажата). Значит HID Report от Game Pad-а будет состоять из 2 байт:

- 1 байт - значащих 4 бита кодируют положение.
- 2 байт - значащих 2 бита кодируют состояние кнопок.

Пример HID Report-а для Game Pad-а созданого при помощи утилиты **HID Descriptor Tool**:

```
flash char ReportDescriptor[52] =
{
 0x05, 0x01, // USAGE_PAGE (Generic Desktop)
 0x09, 0x05, // USAGE (Game Pad)
 0xa1, 0x01, // COLLECTION (Application)
 0x09, 0x01, // USAGE (Pointer)
 0xa1, 0x00, // COLLECTION (Physical)
 0x09, 0x30, // USAGE (X)
 0x09, 0x31, // USAGE (Y)
 0x15, 0x00, // LOGICAL_MINIMUM (0)
 0x25, 0x02, // LOGICAL_MAXIMUM (2)
 0x95, 0x02, // REPORT_COUNT (2)
 0x75, 0x02, // REPORT_SIZE (2)
 0x81, 0x02, // INPUT (Data,Var,Abs)
 0xc0,       // END_COLLECTION
 0x95, 0x04, // REPORT_COUNT (4)
 0x75, 0x01, // REPORT_SIZE (1)
 0x81, 0x03, // INPUT (Cnst,Var,Abs)
 0x05, 0x09, // USAGE_PAGE (Button)
 0x19, 0x01, // USAGE_MINIMUM (Button 1)
 0x29, 0x02, // USAGE_MAXIMUM (Button 2)
 0x15, 0x00, // LOGICAL_MINIMUM (0)
 0x25, 0x01, // LOGICAL_MAXIMUM (1)
 0x75, 0x01, // REPORT_SIZE (1)
 0x95, 0x02, // REPORT_COUNT (2)
 0x81, 0x02, // INPUT (Data,Var,Abs)
 0x95, 0x06, // REPORT_COUNT (6)
 0x81, 0x03, // INPUT (Cnst,Var,Abs)
 0xc0 // END_COLLECTION
};
```

Глава 2. Процедуры инициализации и обслуживания USB контроллера

Глава 2.1 Инициализация USBN9604

После включения питания USB контроллер необходимо проинициализировать. Ниже следует описание инициализации: Init_USBN9604().

Очищаем регистр статуса (устанавливаем что находимся не в режиме выдачи дескриптора, репорта и мультипакета). Сбрасываем регистр номера конфигурации. Инициируем програмный сброс, устанавливая бит "SRST" в "1" в регистре "MCNTRL". Устанавливаем что микросхема будет генерировать прерывание сигналом низкого уровня на выводе INTR (биты "INTOC" в регистре "MCNTRL"), а также включаем опорный источник на 3,3 В (бит "VGE" в регистре "MCNTRL"). Переводим USB контроллер в "Default adress state" устанавливая бит "AD_EN" в "1" и устанавливая нулевой адресс в регистре "FAR". Далее в регистр управления "0 Endpoint" EPC0 записываем 0, сигнализируя, что нулевая конечная точка не генерит "STALL handshake". Далее определяем условия при которых микросхема будет генерировать прерывание, путём установки соответствующих битов в регистрах-масках. Устанавливаем что будем обрабатывать "NAK" события на OUT пакет (когда произошла ошибка при транзакции ХОСТ - конечная точка). Устанавливаем, что будем обрабатывать "TX" события (когда данные из TXFIF0 и TXFIF03 были выданы в ХОСТ и IN транзакция завершена). Устанавливаем что будем обрабатывать "RX" событие (когда завершилось выполнение SETUP или OUT транзакции для RXFIFO0 и данные от ХОСТа пришли в конечную точку). А также в регистре ALTMSK устанавливаем маску, что обрабатываем события "Suspend 3 mc" (на шине установлен сигнал "IDLE" - устройство должно перейти в режим сна) и "Reset" (перейти в "Default Adress state"). В основном регистре масок MAMSK разрешаем соответствующие события. Запрещаем выдачу данных из TXFIFO0 и разрешаем прием в RXFIFO0. После этого в регистре управления состояния NFSR устанавливаем режим "Node Operational" - устройство сконфигурировано для работы на шину. И наконец, устанавливаем в регистре "MCNTRL" бит "NAT" в "1" - это указывает, что устройство готово к обнаружению при подключениии к USB шине. После подключения устройства ХОСТ найдет новый узел и перейдет к процедуре Энумерации "Enumeration procedure".

Ниже приведен текст процедуры инициализации USBN9604.

```
void Init_USBN9604(void)
 {
  ClrBitR(status,GETDESC);   //Выходим из режима выдачи Дескриптора
  ClrBitR(status,MULTIPAK);  //Устанавливаем что не выдаем Multipaket
  usb_cfg = 0
  USB_WR(MCNTRL,SRST);    //Инициируем програмный сброс USBN9604
  USB_WR(MCNTRL,VGE+INT_L_P);//Выбираем режим для INTR-Low Push Pull,
                             //включаем 3.3B Voltage Regulator
  USB_WR(VREGCTL,0x40);      //Устанавливаем 3.3B
  USB_WR(FAR,AD_EN+0);       //Устанавливаем  Default address
  USB_WR(EPC0, 0x00);        //Endpoint 0 Normal operation
  USB_WR(NAKMSK,NAK_O0+NAK_I3);//Разрешаем прерывания по NAK событиям
  USB_WR(TXMSK,TXFIFO0+TXFIFO3);//Разрешаем прерывания по TX событиям
  USB_WR(RXMSK,RXFIFO0);        //Разрешаем прерывания по RX событиям
  USB_WR(ALTMSK,SD3+RESET_A);  //Разрешаем прерывания по ALT событиям
  USB_WR(MAMSK,(INTR_E+RX_EV+NAK+TX_EV+ALT)); //Разрешаем прерывания
  FLUSHTX0();     //Очищаем TXFIFO0 и запрещаем выдачу из него данных
  USB_WR(RXC0,RX_EN);           //Разрешаем прием данных в Endpoint0
  USB_WR(NFSR,OPR_ST);           //Переходим в Operational State
  USB_WR(MCNTRL,VGE+INT_L_P+NAT);//Устанавливаем NODE ATTACH
 }
```

Глава 2.2. Процедура обработки прерывания от USBN9604

Все прцедуры обмена с ХОСТом происходят на очень большой скорости, поэтому работа с USBN9604 представляет собой разветвленную сруктуру обработки прерываний. Если на шине происходит какое-то событие, то оно отображается в регистре событий "MAEV". Прерывание возникает только лишь в том случае, если оно разрешено (устанавлен соответсвующий бит в регистре масок). Задача процедуры обработки прерывания состоит в чтении регистра событий, определения какое из них вызвало прерывание и вызов соответствующей функции обработке этого события.

Процедура обработки прерывания называется USB_ISR() и выполняется по приходу уровня логического "0" на вывод INT0 микроконтроллера AT9058515.

В первую очередь считываем регистр состояния событий MAEV и переписываем его в регистр Event. После этого анализируем что вызвало прерывание. Если это было RX событие, то процедура вызывает функцию обработки RX события. Если это было не RX событие, то процедура анализирует было ли вызвано прерывание одним из TX событий. Если это TX событие то в регистр EVENT считываем состояние TX событий из TXEV, и анализируем какое из TX событий произошло: либо TX0 - выдача данных из TXFIFO0, либо TX3 - выдача данных из TXFIFO3 (endpoint 5). Далее вызываем соответствующие функции обработки этих событий. Если же прерывание вызвано не TX событием, то анализируем не было ли на шине альтернативных событий (Reset, Suspend или Resume). Если возможно одно из этих событий то вызываем функцию USB_alt() обработки альтернативных событий. В самую последнюю очередь начинаем проверку NAK IN и NAK OUT событий - неподтверждения принятия данных соответственно ХОСТом и устройством.

В регистр Event считываем содержимое регистра NAKEV и анализируем к какой конечной точке относится это событие. В зависимости от результата анализа вызываем функцию onak0() - обработки NAK OUT для Endpoint 0, либо inak3() - обработки события NAK IN для Endpoint 5. Листинг программа обработки событий следует ниже:

```
interrupt [INT0_vect] void usb_isr(void)
{
 unsigned char Event;
 Event=USB_RD(MAEV);   //Из регистра событий читаем байт состояния и
                        //определяем, что вызвало прерывание
 if (Event & RX_EV)    //Если это RX событие то
  {
   Event=USB_RD(RXEV); //Из регистра событий приема считываем байт
                       //состояния
   if(Event & RXFIFO0) //Если пакет данных пришел в RXFIFO0
    {
     rx_0(); //Вызываем процедуру обработки пришедшего пакета
     LCD_COM(128+64+6);//Переходим на вторую строку LCD
     LCD_STR(RX0);     //Выводим надпись RX0
    }
   }
 else                  //Иначе
  if (Event & TX_EV)   //Если это TX событие
   { //Начинаем проверку с какой конечной точкой оно связано
    Event=USB_RD(TXEV);//Из регистра событий считываем байт состояния
    if(Event & TXFIFO0) //Если пакет данных был выдан из TXFIFO0
     {
      tx_0();//Вызываем процедуру обработки события выдачи данных
      LCD_COM(128+64+6); //Переходим на вторую строку LCD
      LCD_STR(TX0);      //Выводим надпись TX0
     }
    else
     if (Event & TXFIFO3)//Если пакет данных был выдан из TXFIFO3
      {
       tx_3();//Вызываем процедуру обработки выдачи данных
       LCD_COM(128+64+6);//Переходим на вторую строку LCD
       LCD_STR(TX3);     //Выводим надпись TX0
      }
   }
 else                    //Иначе
  if(Event & ALT) //Если это событие (Reset,Suspend,Resume)
   {
    usb_alt(); //Вызываем процедуру обработки альтернативных событий
    LCD_COM(128+64+6);   //Переходим на вторую строку LCD
    LCD_STR(ALTI);       //Выводим надпись ALT
   }
  else                   //Иначе
   if (Event & NAK)      //Если это NAK событие
    {
     Event=USB_RD(NAKEV);//Начинаем проверку с какой конечной точкой
                         //оно связано
     if (Event & NAK_O0) //Если это генерация NAK OUT для Endpoint 0
      {
       onak0(); //Вызываем процедуру обработки события NAK OUT
       LCD_COM(128+64+6);//Переходим на вторую строку LCD
       LCD_STR(NAK0);    //Выводим надпись NAK0
      }
```

```
      else
       if(Event & NAK_I3) //Если это генерациея NAK IN для Endpoint 5
        {
         inak3(); //Вызываем процедуру обработки NAK OUT для Endpoint0
         LCD_COM(128+64+6);//Переходим на вторую строку LCD
         LCD_STR(NAK3);    //Выводим надпись NAK3
        }
  }
}
```

Глава 2.3. Процедура обработки альтернативных событий USB_alt ()

Процедура считывает в прмежуточный регистр Event из регистра ALTEV состояние флагов указывающих какое из альтернативных событий произошло на шине. Далее обрабатываются 3 альтернативных события Reset, Suspend, Resume.

Если прерывание было вызвано событием Reset, то процедура устанавливает в регистре управления состоянием NFSR состояние "Node Reset". В регистре адреса функции FAR устанавливает бит AD-EN и сбрасывает адресс в "0" (переводим функцию в "Default adress state"). Разрешаем только 0 конечную точку, сбрасывая все биты регистра EPC0 в 0. Очищаем TXFIFO0 и разрешаем прием данных в Endpoint0. После этого переводим USB контроллер в "Node operational state" устанавливая биты NFS в регистре NFSR в "10". Устройство успешно прошло Reset и готово к конфигурированию.

Если USB контроллер детектировал режим сна "Suspend Event", то в регистре маски альтернативных событий устанавливаем, что теперь будем отслеживать события Reset и Resume. В регистре управления состоянием NFSR устонавливаем состояние "Node Suspend", переводим USB контроллер в режим низкого энергопотребления.

Если на шине была продетектирована какая-то активность, то USBN9604 вырабатывает прерывание "Resume Event". Процедура обработки альтернативных событий устанавливает в регистре маски альтернативных событий, что теперь будут отслеживаться события Reset и Suspend, а в регистре управления состоянием NFSR устанавливает режим "Node Operetional" - нормальный режим работы.

Глава 2.4. Процедура приема данных в Endpoint 0: RX0 ()

Процедура RX0 обрабатывает стандартные USB запросы к устройству, запросы типичные для класа устройства (HID запросы), а также обеспечивает прием данных по Control Transfers.

С самого начала считываем состояние принятого пакета из регистра статуса приема RXS0. Далее анализируем принятый пакет ("Setup" или "OUT" пакет). Если это "Setup" пакет то его длинна равна 8 байтам. Считываем 8 байт в буфер приема. Запрещаем прием данных в RXFIFO0 и запрещаем выдачу данных из TXFIFO0. Запрещаем выдачу "STALL".

Далее в поле bmRequestType (USB_BUF [0]) оставляем только 6 и 5 биты, определяющие тип запроса. Где D6, D5: 0 - стандартный тип запроса, 1 - запрос характерный для класса, 2 - запрос определяемый производными.

Если это был стандартный запрос то далее по полю bRequest (USB_BUF[1]) оопределяем какой запрос подал ХОСТ. В зависимости от от полученного запроса процедура вызывает соответствующую функцию, обрабатывающую этот стандартный запрос. Если же принятый запрос не поддерживается, то мы отвечаем STALL-ом. Если же это был запрос характерный для класса устройства "Class Request" то в зависимости от запроса (USB_BUF[1]) вызываем функцию его обработки.

Каждая из функций обработки запрса от ХОСТа может загрузить до 8 байт ответа на установленый запрос. После обработки полученного запроса мы разрешаем выдачу данных из TXFIFO0 с DATA1 PID, а также в регистре DATA PID устанавливаем, что следующий пакет в ХОСТ будет идти с DATA0 PID.

Если же принятый пакет от ХОСТа не является "Setup" пакетом, то это наверное "OUT" пакет. Если это данные которые идут в фазе данных команды Set Report, то выводим их на индикатор. После этого переходим в "Status" фазу и на "IN Token" от ХОСТа отвечаем пакетом 0 длины с DATA1 PID. Если же это не фаза данных команды Set Report, то запрещаем выдачу данных из TXFIFO0, выходим из режима выдачи дескриптора и разрешаем прием данных в Endpoint 0.

```
void rx_0 (void)
 {
  unsigned char rxstat; //Состояние принятого пакета
  unsigned char i;

 rxstat=USB_RD(RXS0); //Считываем состояние принятого пакета
  if(rxstat&SETUP_R) //Приняли Setup пакет?
   {
    for(i=0;i<8;i++)  //Считываем принятый 8-ми байтный пакет в буфер
      usb_buf[i]=USB_RD(RXD0);

    FLUSHRX0(); //Запрещаем прием данных в FIFO0 на время обработки
    FLUSHTX0(); //Запрещаем выдачу данных из FIFO0
    UsbBitClr(EPC0,STALL);   //Запрещаем выдачу STALL Endpoint 0

    switch(usb_buf[0]&0x60)  //В поле bmRequestType оставляем только
                             //биты D6,D5
      {
       case 0x00:            //Был тип запроса Standart? D6.D5=00
        switch (usb_buf[1])  //Определяем тип запроса
         {
          case CLEAR_FEATURE: //Запрос CLEAR_FEATURE
            ClrFeature();
            break;

          case  GET_CONFIGURATION:  //Запрос GET_CONFIGURATION
            USB_WR(TXD0,usb_cfg);
            break;

          case  GET_DESCRIPTOR:     //Запрос GET_DESCRIPTOR
            GetDescriptor();
            break;
```

```
    case GET_INTERFACE:       //Запрос GET_INTERFACE
      USB_WR(TXD0,0);
      break;

     case GET_STATUS:         //Запрос GET_STATUS
      GetStatus();
      break;

     case SET_ADDRESS:        //Запрос SET_ADDRESS
       address=(usb_buf[2]|AD_EN);
       break;

     case SET_CONFIGURATION:  //Запрос SET_CONFIGURATION
       SetConfiguration();
       break;

     case SET_FEATURE:        //Запрос SET_FEATURE
       SetFeature();
       break;

     case SET_INTERFACE:
     if(usb_buf[2]);//Если  интерфейс “0” то выдаем STALL
      UsbBitSet(EPC0,STALL); //у данной Конфигурации один
      break;                 //Интерфейс

     default:
       UsbBitSet(EPC0,STALL);//Принятый запрос не
       break;                //поддерживается
    }
    break;

case 0x20:    //Был тип запроса Class? D6.D5=01
 switch(usb_buf[1])    //Определяем тип запроса
   {
    case GET_REPORT:            //Запрос GET_REPORT
     GetReport();
     break;

    case SET_REPORT:           //Запрос SET_REPORT
     SetReport();
     break;

    case GET_IDLE:             //Запрос GET_IDLE
     GetIdle();
     break;

    case SET_IDLE:             //Запрос SET_IDLE
      SetIdle();
      break;

    case GET_PROTOCOL:        //Запрос GET_PROTOCOL
      GetProtocol();
      break;

    case SET_PROTOCOL:        //Запрос SET_PROTOCOL
      SetProtocol();
      break;
```

```
            default:
             UsbBitSet(EPC0,STALL);  //Принятый запрос не
             break;                  // поддерживается

          }
          break;

          default:
           UsbBitSet(EPC0,STALL); //Принятый тип запроса не
            break;                //поддерживается

  }

   if(status&OutPack)//Если принимаем данные по Set Report то просто
    { //разрешаем прием в RXFIFO0, без выдачи данных
     USB_WR(RXC0,RX_EN);
    }
   else //Иначе отвечаем на Setup пакет данными загруженными в RXFIFO
    {   //функциями ответа на Запрос
     USB_WR(TXC0,TX_TOGL+TX_EN);  //Разрешаем выдачу данных DATA1 PID
     ClrBitR(DataPID,TGL0PID);    //следующий будет DATA0 PID
    }
 }

/*Если это был не SETUP пакет значит это должен быть OUT пакет*/
/*Выходим из выдачи Multipacket и разрешаем прием данных в RXFIFO0*/

else
  {
  if((status&OutPack)&&(rxstat&RX_LAST))//Если это данные идущие по
              //команде Set Report и пакет принят успешно
    {         //(Пока не осуществляем проверку Типа и ID репорта)
     ClrBitR(status,OutPack);//Выходим из фазы данных команды Set
                             //Report
     LCD_COM(128+4);         //Переходим на 1 строку в 5 знакоместо
     for(i=0;i<Rpt_Cnt;i++)  //Выводим принятые по команде Set Report
                             //данные на индикатор
     LCD_CHR(USB_RD(TXD0)+0x30);//то количество, что было указано в
                               //wLenght
     USB_WR(TXC0,TX_TOGL+TX_EN);//Переходим в фазу Статуса
     ClrBitR(DataPID,TGL0PID);  //устанавливая при этом DATA1 PID
    }

  else
   if(status&MULTIPAK)   //Находимся в состоянии выдачи Multipacket
    {
     FLUSHTX0();           //Запрещаем выдачу данных из TXFIFO0
     USB_WR(RXC0,RX_EN);   //Разрешаем прием данных в Endpoint 0
     ClrBitR(status,GETDESC);//Выходим из режима выдачи Дескриптора
     ClrBitR(status,MULTIPAK);//Устанавливаем что не выдаем
                              //Multipacket
    }
   }
 }
```

Глава 2.5. Процедура выдачи данных из Endpoint 0: TX0 ().

Прерывание которое вознивает по выдаче пакета означает, что пакет данных, которые мы положили в TXFIFO0 был выдан в ХОСТ успешно, либо при передаче возникли ошибки. По-этому с самого начала процедура считывает регистр статуса передатчика, чтобы определить не возникло ли при передаче данных ошибок. Если передача произошла успешно и мы находимся в режиме выдачи Мультипакета (Выдачи Дескриптора устройства , Конфигурации или HID репорта), тогда загружаем очередную порцию данных в TXFIFO, выбираем DATA PID и в регистре DataPid меняем флаг PID на противоположный. Если мы находимся в режиме установки адреса устройства, то назначенный Хост-ом адрес записываем в регистр FAR и выходим из режима установки адреса устройства, разрешая при этом прием данных в RXFIFO0.

Если же при передаче пакета возникли ошибки, т.е. ХОСТ ответил NAK, то мы выходим из режима выдачи Дескриптора и разрешаем прием данных в RXFIFO0.

```
void tx_0(void)
{
 unsigned char txstat,i;
 txstat=USB_RD(TXS0); //Считываем состояние после отсылки пакета
 if(txstat & TX_DONE) //Если передача данных выполнена
 {
  FLUSHTX0();         //Очищаем TXFIFO0 и запрещаем выдачу данных
  if(txstat & ACK_STAT) //Если было получено подтверждение (ACK)
   {
    if(status & MULTIPAK) //Если мы выдаем Мультипакет
    {
     for(i=0;i<8;i++) //Загружаем очередную порцию данных в TXFIFO0
      mlti_pkt();
     if(DataPID & TGL0PID) //Устанавлтваем DATA PID с которым должны
      USB_WR(TXC0,TX_TOGL+TX_EN); выдаватся данные
     else
      USB_WR(TXC0,TX_EN);
      DataPID ^= TGL0PID; //Меняем флаг DataPID на противоположный
    }
    else //Если находимся не в режиме выдачи Мультипакета
    {
     if(address) //А находимся в режиме Set Address
      {
       USB_WR(FAR,address); //Записываем адрес назначенный Хостом
       address=0; //Выходим из режима установки адреса устройства
      }
    USB_WR(RXC0,RX_EN); //Разрешаем прием данных в RXFIFO0
  }
}
else //Если же Хост ответил NAK: Произошли ошибки при передаче
 {
  ClrBitR(status,GETDESC); //Выходим из режима выдачи Дескриптора
  ClrBitR(status,MULTIPAK); //Устанавливаем что не выдаем Multipacket
  USB_WR(RXC0,RX_EN); //Разрешаем прием данных в RXFIFO0
 }
}
}
```

Глава 2.6. Процедура обслуживания NAK ответа для Endpoint0.

Событие NAK - это один из путей сказать Хосту, что устройство не готово выполнить передачу данных.

В данном случае процедура NAK0() отлавливает NAK ответ на OUT транзакциии разрешает прием данных в RXFIFO0. Какая же ситуация может вызвать NAK ответ? Например, устройство находится в фазе выдачи дескриптора в ХОСТ. После выдачи устройством пакета Device Descriptor - длинной в 8 байт, ХОСТ прерывает прием остальных данных и инициирует OUT пакет. У устройства в этот момент разрешен только передатчик, т.к. оно выдает дескриптор, а прием данных запрещен. Соответственно ответом на OUT транзакцию является "NAK Handshake". Чтобы постоянно не остаться в мертвом цикле генерации NAK нам необходимо очистить очередь в TXFIF0 и запретить выдачу данных из Endpoint 0, а также выйти из режима выдачи дескриптора. Эту последовательность действий и выполняет функция NAK0().

```
void onak0(void)
{
 if(status & MULTIPAK) //Если находимся в режиме выдачи Мультипакета
 {
  ClrBitR(status,GETDESC); //Выходим из режима выдачи Дескриптора
  ClrBitR(status,MULTIPAK);//Устанавливаем что не выдаем Multipacket
  FLUSHTX0();              //Очищаем TXFIFO0 и запрещаем выдачу
                           //данных
  USB_WR(RXC0,RX_EN);      //Разрешаем прием данных в RXFIFO0
 }
}
```

Глава 3. USB стандартные запросы

Глава 3.1. Процедура Get descriptor()

Как уже было указано выше, Дескриптор устройства это блок данных описывающий режим работы устройства (количество конечных точек, тип передачи данных, время опроса устройства и т.д.). ХОСТ запрашивает дескриптор устройства во время процесса энумерации для правильной инициализации (выбора соответствующего драйвера) и выделения необходимых ресурсов. Запрос "Get_Descriptor" представляет собой 8-ми байтовый пакет. Тип запрашиваемого у устройства дескриптора указывается в старшем байте wValue (USB_BUF [3]). В поле wLength ХОСТ указывает размер желаемого ответа от устройства в байтах (USB_BUF[6...7]). Если дескриптор имеет размер больше, чем запросил ХОСТ, то возвращаются только первые данные. Если же дескриптор имеет размер меньше запрашиваемого, то мы должны ответить либо нулевым пакетом, либо пакетом длинной не более чем "payload size".

Функция обрабатывающая запрос "Get_Descriptor" первоначально устанавливает флаги GetDesc и MULTIPAK, что соответствует режиму выдачи Дескриптора. Т.к. дескриптор может представлять собой структуру размер которой больше 8 байт, и за одну IN транзакцию невозможно выдать весь дескриптор, то мы входим в режим мультипакета - выдачи данных за несколько обращений. После этого функция анализирует какой же дескриптор запросил ХОСТ (старший байт wValue). В зависимости от запроса устанавливаем размер и индекс соответствующего дескриптора. В конце процедуры данные загружаются в RXFIFO0 для выдачи в ХОСТ.

```
void GetDescriptor(void)
{
 unsigned char i;
 SetBitR(status,GETDESC); //Устанавливаем,что выдаем escriptor
 SetBitR(status,MULTIPAK);//Устанавливаем,что выдаем Multipacket
 des_typ = usb_buf[3];    //Сохраняем тип запрашиваемого дескриптора

 switch (des_typ)        //Анализируем какой дескриптор запросил Хост
  {
   case DEVICE:          //Если это дескриптор устройства
    des_idx = 0;         //Начнем выдачу с 1 байта из таблицы
                         //дескриптора у-ва
    des_sze = DEV_DESC_SIZE; //Устанавливаем длину выдаваемого
    break;                   //дескриптора

   case CONFIGURATION:   //Если это дескриптор конфигурации
    des_idx = 0; //Начнем выдачу с 1 байта из таблицы дескриптора
                 //конфигурации
    des_sze = CFG_DESC_SIZE; //Устанавливаем длину выдаваемого
    break;                   //дескриптора

   case XSTRING:             //Если это строковый дескриптор
    des_idx = usb_buf[2];    //Загружаем индекс запрашиваемого
                             //сторкового дескриптора LOWBYTE(wValue)
```

```
   switch(des_idx) //В зависимости от запрашиваемого дескриптора
    {
     case 0:       //Если Хост запрашивает String Descriptor
     des_sze=4;    //Устанавливаем размер String Descriptor (4 байта)
     break;

     case MFG_STR:      //Если Хост запрашивает Manufacturer String
     des_sze=MFG_STR_L; //Устанавливаем величину Manufacturer String
     //(количество сиволов + bDescriptorTyp(3) + bLength)
     des_idx=4;         //Устанавливаем индекс (где находится первый
     break;          //байт) Manufacturer String в массиве STR_DATA[]

     case PID_STR:      //Если Хост запрашивает Product Ident String
     des_sze=PID_STR_L; //Устанавливаем величину Product Ident String
     des_idx=4+MFG_STR_L; //Устанавливаем индекс (где находится
     break;  //первый байт) Product Ident String в массиве STR_DATA[]

     case SNBR_STR: //Если Хост запрашивает Serial Number String
     des_sze=SNBR_STR_L; //Устанавливаем величину Serial Number
     des_idx=4+MFG_STR_L+PID_STR_L; //Устанавливаем индекс Serial
     break;                         //Number в массиве STR_DATA[]

     case CFG_STR:      //Если Хост запрашивает Configuration String
     des_sze=CFG_STR_L; //Устанавливаем величину Configuration String
     des_idx=4+MFG_STR_L+PID_STR_L+SNBR_STR_L;//Устанавливаем индекс
     break;             //Configuration String в массиве STR_DATA[]

     case INT_STR:      //Если Хост запрашивает Interface String
     des_sze=INT_STR_L; //Устанавливаем величину Interface String
     des_idx=4+MFG_STR_L+PID_STR_L+SNBR_STR_L+CFG_STR_L;
      break;            //Устанавливаем индекс Interface String в
                        //массиве STR_DATA[]
    }
   break;

   case HID:            //Если Хост запрашивает HID Descriptor
   des_idx = CFG_LENGTH+INT_LENGTH;//Устанавливаем индекс HID
                                   //Descriptor в массиве CFG_DESC[]
   des_sze = CFG_DESC[(CFG_LENGTH+INT_LENGTH)];//Устанавливаем
   break; //величину HID Descriptor(первый байт в HID дескрипторе)

   case HIDREPORT: //Если Хост запрашивает HID Report
   des_idx = 0;    //Устанавливаем, что будем выдавать Report из
                   //массива ReportDescriptor[]
   des_sze = RPT_DESC_SIZE; //Устанавливаем величину HID Report
   break;          //Descriptor

   default: //Если мы не поддерживаем запрашиваемый дескриптор то:
    des_idx = 0;
    des_sze = 0;
    break;
  }
if(usb_buf[7]==0) //Хост запросил размер дескриптора меньше 256 байт
if(des_sze>usb_buf[6]) //Смотрим длина нашего дескриптора не
                       //превышает запршенной длины?
des_sze = usb_buf[6];  //Превышает, значит устанавливаем длину
                       //дескриптора переданную ХОСТом
```

```
//Выдаем 8 байт из таблицы дескриптора в TXFIFO0
for(i=0;i<8;i++)
mlti_pkt();
}
```

Глава 3.2. Процедуры Setfeature() и Clrfeature()

Запрос Set feature используется для установки конечной точки в состояние генерации STALL, если обращение идет к endpoint (wValue), либо разрешение опции "Remote wakeup", если обращение идет к Device, либо установку каких-либо других возможностей, когда обращение идет к интерфейсу. Для данного примера "Remote WakeUP" не используется, не используется также "Feature interface", соответственно процедура Set feature() игнорирует обращение к Endpoint. Если обращение было к конечной точке, то функция считывает из wIndex (USB_BUF[4]), к какой именно конечной точке было обращение, затем в регистре управления Endpoint EPC устанавливает бит STALL, тем самым инициируя генерацию "STALL" на любое обращение к этой Endpoint. В програмном регистре "STALLD" устанавливается соответствующий бит в "1" (необходимо для выдаачи состояния конечных точек по команде Get Status()).
Т.к. управляющий канал (Endpoint 0) не может находится в STALL-е, то в регистре управления EPC0 мы не устанавливаем STALL бит, а лишь фиксируем его в регистре STALLD.

```
void SetFeature(void)
{
 switch (usb_buf[0]&0x03) //Оставляем в bmRequestType биты D1.D0 -
                          //Recipient(к кому обращение)
  {
   case 0: //Обращение к DEVICE
    break;

   case 1: //Обращение к INTERFACE
    break;

   case 2: //Обращение к ENDPOINT
    switch (usb_buf[4]&0x0F)//Убираем Direction bit D7, оставляем
                            //только биты D3...D0 - указывающие
     {            //у какой Endpoint необходимо установить FEATURE
      case 0:
     //Управляющий канал не может находится в STALLE
     SetBitR(stalld,BIT0); //Поэтому фиксируем эту установку только в
     break;                //регистре состояния Endpoint

     case 1:
      UsbBitSet(EPC1,STALL); //Устанавливаем HALT Condition у EP1
      SetBitR(stalld,BIT1);  //Записывем в регистр состояния Endpoint
      break;

     case 2:
      UsbBitSet(EPC2,STALL); //Устанавливаем HALT Condition у EP2
      SetBitR(stalld,BIT2);  //Записывем в регистр состояния Endpoint
      break;
```

```
     case 3:
      UsbBitSet(EPC3,STALL); //Устанавливаем HALT Condition у EP3
      SetBitR(stalld,BIT3);  //Записывем в регистр состояния Endpoint
      break;

     case 4:
      UsbBitSet(EPC4,STALL); //Устанавливаем HALT Condition у EP4
      SetBitR(stalld,BIT4);  //Записывем в регистр состояния Endpoint
      break;

     case 5:
      UsbBitSet(EPC5,STALL); //Устанавливаем HALT Condition у EP5
      SetBitR(stalld,BIT5);  //Записывем в регистр состояния Endpoint
      break;

    case 6:
     UsbBitSet(EPC6,STALL); //Устанавливаем HALT Condition у EP6
     SetBitR(stalld,BIT6);  //Записывем в регистр состояния Endpoint
     break;

    default:
     break;
   }
   break;

default:
 break;
 }
}
```

Процедура Clr Feature() игнорирует все запрсы к устройству и интерфейсу. Если запрос был к конечной точке, то в регистре EPC соответствующей Endpoint сбрасывается бит STALL, и конечная точка из HALT mode переходит в режим нормальной работы "Normal operation". Также в регистре STALLD сбрасывается флаг, соответствующий конечной точке к которой было обращение.

```
void ClrFeature(void)
{
 switch (usb_buf[0]&0x03) //Оставляем в bmRequestType биты D1.D0 -
                          //Recipient(к кому обращение)
  {
    case 0:   //Обращение к DEVICE
     break;
    case 1:   //Обращение к INTERFACE
     break;
    case 2:   //Обращение к ENDPOINT
     switch(usb_buf[4]&0x0F) //Убираем Direction bit D7, оставляем
                             //только биты  D3...D0 - указывающие
     {//у какой Endpoint необходимо очистить FEATURE
      case 0:
       UsbBitClr(EPC0,STALL);//Убираем HALT Condition у EP0
       ClrBitR(stalld,BIT0); //Записывем в регистр состояния Endpoint
       break;
```

```
      case 1:
       UsbBitClr(EPC1,STALL); //Убираем HALT Condition у EP1
       ClrBitR(stalld,BIT1);  //Записывем в регистр состояния Endpoint
       break;
      case 2:
       UsbBitClr(EPC2,STALL); //Убираем HALT Condition у EP2
       ClrBitR(stalld,BIT2);  //Записывем в регистр состояния Endpoint
       break;
      case 3:
       UsbBitClr(EPC3,STALL); //Убираем HALT Condition у EP3
       ClrBitR(stalld,BIT3);  //Записывем в регистр состояния Endpoint
       break;
      case 4:
       UsbBitClr(EPC4,STALL); //Убираем HALT Condition у EP4
       ClrBitR(stalld,BIT4);  //Записывем в регистр состояния Endpoint
       break;
      case 5:
       UsbBitClr(EPC5,STALL); //Убираем HALT Condition у EP5
       ClrBitR(stalld,BIT5);  //Записывем в регистр состояния Endpoint
       break;
      case 6:
       UsbBitClr(EPC6,STALL); //Убираем HALT Condition у EP6
       ClrBitR(stalld,BIT6);  //Записывем в регистр состояния Endpoint
       break;
      default:
       break;
     }
     break;

     default:
      break;
   }
}
```

Глава 3.3. Процедура Get Status()

Запрос Gert Status возвращает 2 байта ответа в Хост. Первый байт ответа в зависимости от того, к чему идет обращение (Device, Interface, Endpoint) имеет определенное значение. Если обращение к Device (BmRequest Type D1,D0=00) то Хост желает получить информацию о возможности запроса устройством Remote WakeUp - это бит D1, и информацию о питании устройства Self Powered - это бит D0. Если бит D0 - установлен в "1", то устройство питается от внешнего источника питания, если установлен в "0", то питание берется с шины V+. Этот бит не может быть изменен запросами Set Feature и Clear Feature. Бит D1 указывает на возможность установки устройством на USB шине сигнала Remote WakeUp, если он взведен в "1". Битовое поле Remote WakeUp может быть модифицировано командами Set Feature и Clear Feature. По умолчанию бит сброшен в "0" - запрещен Remote Wake Up.

Если обращение запроса Get Status направлено к Interface, то байт ответа по спецификации зарезервирован и равен "0".

Если обращение направлено к Endpoint, то в ответе бит D0 обозначает текущее состояние Endpoint. Если конечнаая точка находится в состоянии "Halt Condition", то бит D0 необходимоо установить в "1". Опционально очистить состояние Halt можно при помощи команды Clear Feature. Также состояние Halt очищается после приема команд Set Configuration и Set Interface.

Второй байт ответа на команду Get Status зарезервирован и равен "0".

```
void GetStatus(void)
{
 switch (usb_buf[0]&0x03)//Оставляем в bmRequestType биты D1.D0 -
                         //Recipient(к кому обращение)
 {
  case 0:                //Обращение к DEVICE
   USB_WR(TXD0,0);       //Первый байт ответа равен 0 т.к.
   break;                //RemouteWakeUp=0 и SelfPowered=0

  case 1:                //Обращение к INTERFACE
   USB_WR(TXD0,0);      //Первый байт ответа зарезервирован и равен 0
   break;
  case 2:                //Обращение к ENDPOINT
   switch (usb_buf[4]&0x0F) //Убираем Direction bit D7, оставляем
                            //только биты D3...D0 - указывающие
   { //у какой EP узнаем Status:(состояние HALT или Normal Operation)
    case 0: EPSTATUS(BIT0);
    case 1: EPSTATUS(BIT1);
    case 2: EPSTATUS(BIT2);
    case 3: EPSTATUS(BIT3);
    case 4: EPSTATUS(BIT4);
    case 5: EPSTATUS(BIT5);
    case 6: EPSTATUS(BIT6);
    default:
     break;
   }
   break;

  default:
   USB_WR(TXD0,0);
    break;
 }
USB_WR(TXD0,0);//Выдаем второй байт STATUS-а он зарезервирован и
               //равен '0'
}
```

Глава 3.4. Процедура Set Configuration() и Get Configuration()

По включению питания или после сброса, USB устройство находится в несконфигурированом состоянии. Это означает, что у него разрешена только 0 конечная точка, при этом устройство занимает ограниченную полосу на шине и находится в состоянии Low Power. После процесса "Enumeration" Хост считывает дескрипторы устройства и определяет может ли оно быть установлено в системе. Некоторые устройства поддерживают несколько конфигураций. Цель этого состоит в том чтобы для определенной работы устройство занимало наименьшее количество ресурсов (полосы пропускания шины, потребление питания).

Запрос Set Configuration используется для выбора конфигурации (если их несколько в устройстве) или для выключения устройства (если Хост передает конфигурационное значение равное 0). Наше устройство имеет только одну конфигурацию (номер "1"). При приходе запроса Set Configuration процедура анализирует младший байт wValue - какую конфигурацию хочет установить Хост. Если это значение отлично от 0, то процедура разрешает In Endpoint по адресу 5, сбрасывает Data Toggle Pid в Data 0 и очищает TXFIFO3. Номер установленной конфигурации заносится в регистр usb_cfg. Если же Хост устанавливает "0" конфигурацию, то процедура просто запрещает все конечные точки, кроме Endpoint 0.

```
void SetConfiguration(void)
{
 usb_cfg = usb_buf[2];//Загружаем номер Configuration из wValue
 if (usb_buf[2]!=0)   //У нас пока только одна конфигурация =1
 {
  DataPID = 0;        //Устанавливаем DataPID в DATA0
  stalld = 0;         //Устанавливаем, что все находится не в STALL-е
  FLUSHTX3();         //Очищаем TXFIFO3 и запрещаем передачу
  USB_WR(EPC5,EP_EN+05);//Разрешаем EP5 по адресу 5
 }
 else
 {
  USB_WR(EPC1,0); //Запрещаем EP1
  USB_WR(EPC2,0); //Запрещаем EP2
  USB_WR(EPC5,0); //Запрещаем EP5
  USB_WR(EPC6,0); //Запрещаем EP6
 }
}
```

Процедура Get Configuration внедрена в rx0 процедуру. Она просто возвращает номер конфигурации хранящийся в регистре usb_cfg.

Глава 3.5 Процедура Set Address()

Запрос Set Address является одним из самых первых в процессе "Enumeration". Установка переданного "Device Address" (младший байт wValue) должна произойтти после успешного завершения "Status stage" этого запроса. Для этого в процедуре rx0() при декодировании запроса Set Address в переменную address записывается переданный Хостом адрес и устанавливается бит ‘Address Enable’. В фазе статуса, т.е. в процедуре tx_0(), данные из переменной address перезаписываются в регистр адресса функции FAR, а в переменную address заносится 0 - выход из режима установки адреса.

Глава 4. Процедуры касающиеся класса устройства HID

Запросы касающиеся класса устройства позволяют Хосту узнать возможности данного устройства, а также установить "Output" и "Feature" items. Данные транзакции выполняются через Default pipe (конечную точку 0).

Глава 4.1. Процедура обработки запроса Get Report()

Запрос Get Report позволяет Хосту получить репорт череез Control pipe. Этот запрос необходим для считывания абсолютных значений из устройства, а также определения состояния Feature items. Процедура просто сохраняет номер (ID) и тип запрашиваемого репорта, если Хост запросил IN-репорт, то в RXFIFO0 загружаются данные текущего состояния кнопок Game Pad.

```
void GetReport(void)
{
 Rpt_ID = usb_buf[2];  //Считываем Report ID
 Rpt_typ = usb_buf[3]; //Сохраняем тип запрашиваемого репорта (1-
                       //IN,2-OUT,3-Feature)
 if(Rpt_typ==1)        //Если Хост запрашивает IN report
  queue_rpt(TXD0);     //Выдаем REPORT через Control Pipe
}
```

Глава 4.2 Процедура обработки запроса Set Report()

Запрос Set Report позволяет Хосту посылать репорты устройству, что является основной возможностью устанавливать состояние различных органов управления (например сбросить их в первоначальное состояние). Процедура обработки этого запроса первоначально считывает Report ID и Report Type, также считывает из младшего байта wLength количество байт, передаваемых Хостом и устанавливает бит Out Pack в регистре status, что означает что в процедуре rx_0() необходимо будет принять данные репорта следующие в фазе данных.

```
void SetReport(void)
{
 Rpt_ID = usb_buf[2];  //Считываем Report ID
 Rpt_Cnt = usb_buf[6]; //Считываем из wLength количество принимаемых
                       //данных в фазе данных (LowByte wLength)
 Rpt_typ = usb_buf[3]; //Сохраняем тип запрашиваемого репорта (1-
                       //IN,2-OUT,3-Feature)
 SetBitR(status,OutPack);//Устанавливаем что далее следует фаза
                         //данных команды Set Report
}
```

Глава 4.3. Процедуры обработки запросов Set Idle() и Get Idle()

Запрос Set Idle задает время через которое Хост ожидает увидеть следующий репорт по Interrupt In pipe. Этот запрос используется для уменьшения частоты выдачи репортов в Interrupt In pipe. Этот запрс заставит In Endpoint выдавать NAK пока репорт остается без изменений. Время через которое необходимо выдавать репорт (Duration) задается Хостом в старшем байте wValue. Если это значение равно 0, то конечная точка отвечает данным без задержки, как только было изменение в данных репорта.
Если значение Duration не равно 0, то используется фиксированая задержка. Младший бит значения Duration соответствует задержке в 4 мс. Если Хост задал Duration меньше чем частота опроса конечной точки (задается в Endpoint descriptor), то репорты генерируются с частотой опроса конечной точки. В младшем байте wValue находится Report ID - номер репорта к которому прикрепляется переданое значение Duration. Если Report ID равен 0, то данное значение Duration Хост накладывает на все доступные Report-ы усройства. Процедура обработки запроса Set Idle просто сохраняет значение Duration в переменной USB_idl. Процедура обработки Get Idle считывает значение установленой Duration из переменной USB_idl и выдает его обратно в Хост.

Глава 4.4. Процедуры обработки запросов Set Protocol() и GetProtocol()

Запрос Set Protocol поддерживается устройствами Boot подкласса. Значение старшего байта wValue указывает какой протокол должен быть использован. Когда устройство проинициализировано оно выдает report протокол. Однако Хост не знает в каком первоначальном состоянии находится устройство и поэтому запросом Set Protocol он устанавливает выдачу устройством желаемый протокол. Процедура Set Protocol() считывает значение из старшего байта wValue (тип выдаваемого ответа Хосту: 0 - Boot Protocol, 1 - Report Protocol), и сохраняет его в переменной Prt_Typ.

Процедура Get Protocol() просто выдает сохраненное значение из переменной Prt_Typ обратно в Хост.

Глава 4.5. Процедура опроса клавиш и генерации Report: Update()

Данная процедура считывает из PINC состояние клавиш передвижения по осям X и Y, а также из порта PIND состояния нажатых кнопок Button 1 и Button 2. Потом в зависимости от нажатых клавиш процедура генерирует Report размером в 2 байта: первый байт представляет собой данные положения курсора по осям координат X и Y (D0, D1 - X ось; D2, D3 - Y ось). Второй байт представляет собой данные указывающие какая из кнопок нажата (D0 - Button 1, D1 - Button 2). "1" - указывает, что клавиша нажата.

Глава 4.6. Процедура выдачи Report в Хост из Endpoint 5: tx_3()

Хост опрашивает конечную точку (Endpoint5) с частотой указаной в Endpoint Descriptor в значении Interval. Эти запросы обрабатываются процедурой tx_3().
Первое, что делает эта процедура - чтение регистра статуса передатчика TXS3. После этого очищаем и запрещаем выдачу данных из TXFIFO3. Если предыдущий пакет был выдан успешно из FIFO (установлены биты TXDONE и ACK_STAT в регистре статуса TXS3), то вызываем процедуру Update(), которая опросит клавиши и создаст новый 2-х байтовый Report. После этого загружаем новый Report в TXFIFO3 и выбрав соответствующий DATA PID разрешая выдачу данных из TXFIFO3.

```
void tx_3(void)
{
 unsigned char txstat;

 txstat=USB_RD(TXS3); //Считываем состояние передачи
 if((txstat & ACK_STAT)&&(txstat & TX_DONE)) //Если выдача данных
                                             //произошла успешно
 {
  FLUSHTX3();           //Очищаем TXFIFO3 и запрещаеп передачу
                        // Выбираем соответствующий DataPID
  if(DataPID & TGL3PID) //Если до этого был DATA1 PID
   ClrBitR(DataPID,TGL3PID); //То устанавливаем DATA0 PID
  else
   SetBitR(DataPID,TGL3PID); //Иначе устанавливаем DATA1 PID
   update();                //Вызываем процедуру опроса клавиш Game
                            //Pad и генерируем Report
   queue_rpt(TXD3);     //Загружаем сгенерированный Report в TXFIFO3

  //Далее выбираем DATA PID и разрешаем выдачу данных из FIFO
   if(DataPID & TGL3PID)     //Если необходимо установить DATA1 PID
    USB_WR(TXC3,TX_TOGL+TX_LAST+TX_EN); //Устанавливаем DATA1 PID и
                            //разрешаем выдачу данных из TXFIFO
   else                     //Если необходимо установить DATA0 PID
    USB_WR(TXC3,TX_LAST+TX_EN);//Устанавливаем DATA0 PID и разрешаем
                            //выдачу данных из TXFIFO
 }
}
```

Глава 4.7. Процедура обработки события NAK для Endpoint 3: inak 3 ()

Если при выдаче репорта из Endpoint 3 в Хост произошел сбой и Хост ответил NAK, то нам необходимо повторить снова выдачу текущего Report. Для этого необходимо взвести бит RFF в регистре контроля передачи TXC3 - что означает перезагрузку TXFIFO3 теми же данными (восстанавливается указатель TXRP). После этого выбирается соответствующий данной транзакции Data PID и разрешается выдача данных из FIFO.

```
void inak3(void)
{
 //FLUSHTX3(); //Очищаем TXFIFO и запрещаем вдачу данных
 //queue_rpt(TXD3); //Загружаем сгенерированный Report в TXFIFO
 //Разрешаем передачу с выбором соответствующего DATA PID

 if(DataPID & TGL3PID) //Если текущий DATA PID=1
  USB_WR(TXC3,RFF+TX_TOGL+TX_LAST+TX_EN);//Устанавливаем DATA1 PID и
                            //перегружаем FIFO предыдущими данными
 else                           //Если текущий DATA PID=0
 USB_WR(TXC3,RFF+TX_LAST+TX_EN);//Устанавливаем DATA0 PID и
                            //перегружаем FIFO предыдущими данными
}
```