AVR287: USB Host HID and Mass Storage Demonstration

8-bit AVR® Microcontrollers

Application Note

Features

- **Based on AVR® USB OTG Reduced Host**
- **Runs on AT90USB647/1287**
- **Support bootable/non-bootable standard USB mouse**
- **Support USB Hub feature (Mass Storage devices only).**
- **Mass Storage:**
  - **Reduced Block Commands (RBC)**
  - **SFF-8020i or MMC-2 (ATAPI) command blocks**
  - **UFI Typically, a floppy disk drive (FDD) device**
  - **SFF-8070i command blocks**
  - **SCSI transparent Command Set**

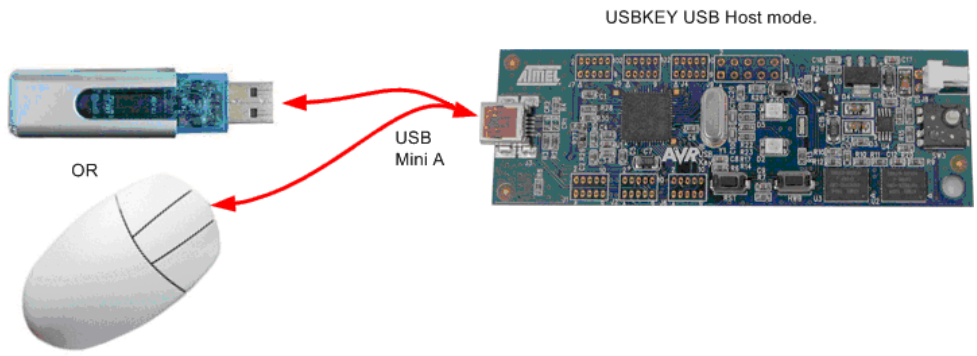
1 Introduction

More and more peripherals are using the USB interface nowadays. Having an USB host interface in the embedded applications which can support an USB standard device is a big benefit.

The aim of this document is to describe how to start and implement a USB host application based on the USB HID class (USB mouse) and Mass Storage class. And finally introduces a simple example of AT90USB (Series-7) managing both USB classes (HID and Mass Storage) and also a file system (FAT12/16/32).

A familiarity with the AVR USB firmware framework (http://www.atmel.com), the HID specification and Mass Storage knowledge (http://www.usb.org) are assumed.

Figure 1-1. Host HID and Mass Storage Application



Rev. 8229A-AVR-09/09

2 Theory of Operation

2.1 HID class

2.1.1 HID Configuration

The HID class requires 1 Ctrl endpoint (ep0), 1 Interrupt IN endpoint and 1 optional Interrupt OUT endpoint. Normally, the Ctrl endpoint (ep0) is only used to deal with the enumeration process. The Interrupt IN endpoint is used to transfer the IN report (like pressed key in mouse application) from the device to the host, and the Interrupt OUT endpoint is used to transfer the OUT report (like LED status in keyboard application) from the host to the device.

In conclusion, a standard USB mouse application requires 1 default control endpoint and 1 interrupt IN endpoint.

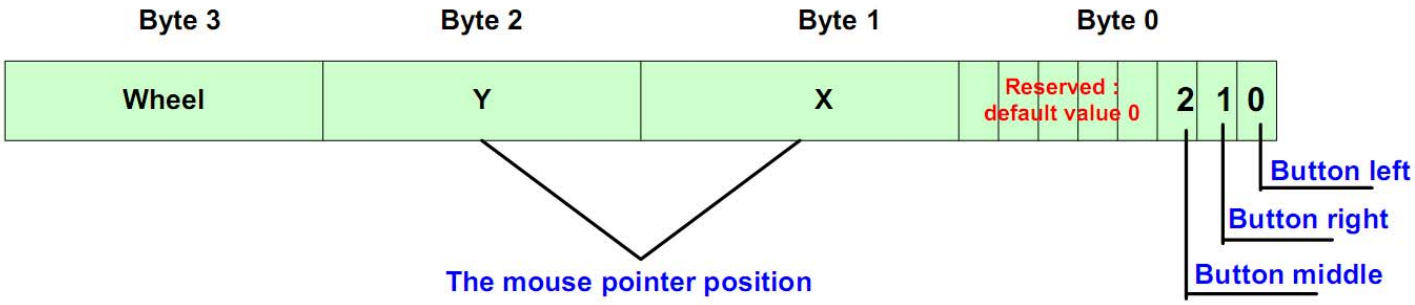
2.1.2 Data Transfer

The USB HID (USB mouse) application is a simple data exchange between the host and the mouse.

Host asks the mouse if there is new data available each P time (polling interval time), the mouse will send the data if it is available, otherwise it will send a NAK (No Acknowledge) to tell the host that there is no data available.

The data sent to the host is called ‘report’. This report has the structure below:

Figure 2-1. USB Report Structure



Each time a button is pushed or the mouse is moved, this report will be sent to the host. These bytes will be read in this order: byte0 -> byte3.

2.2 Mass Storage class

2.2.1 Mass Storage Configuration

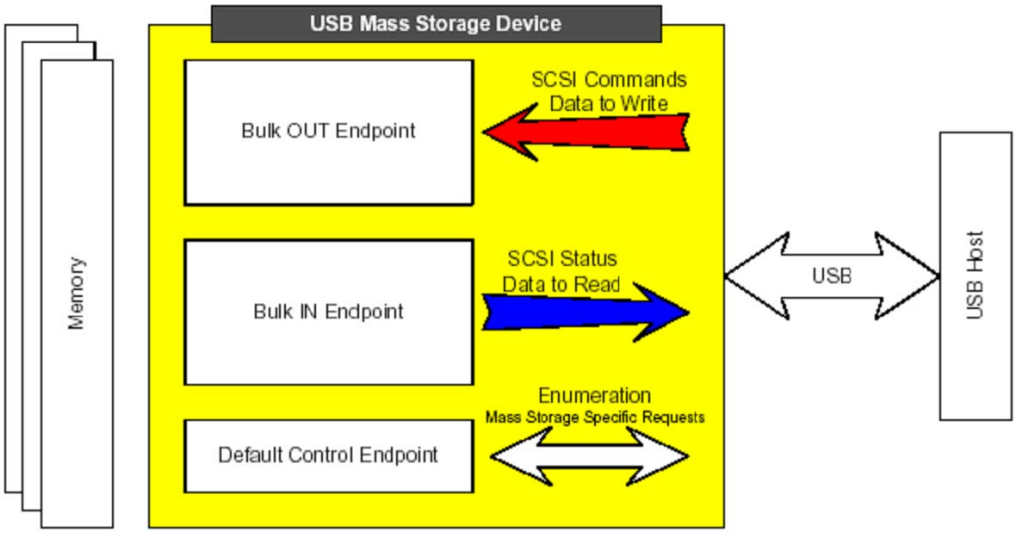
The Mass Storage application uses two bulk endpoints (one IN and one OUT) to perform the status and data transfer. The endpoint 0 (control endpoint) is used only to perform the enumeration process, the errors management and to determine the LUN (Logic Unit Number) value.

2.2.2 Data Transfer

The USB data exchange for the Mass Storage application is based on the SCSI (Small Computer System Interface) commands. In other words, the Mass Storage application is a set of SCSI commands send by the host to manage the file transfer.

The Mass Storage class allows one device to manage several storage units at the same time thanks to the LUN.

Figure 2-2. USB Mass Storage Application Overview

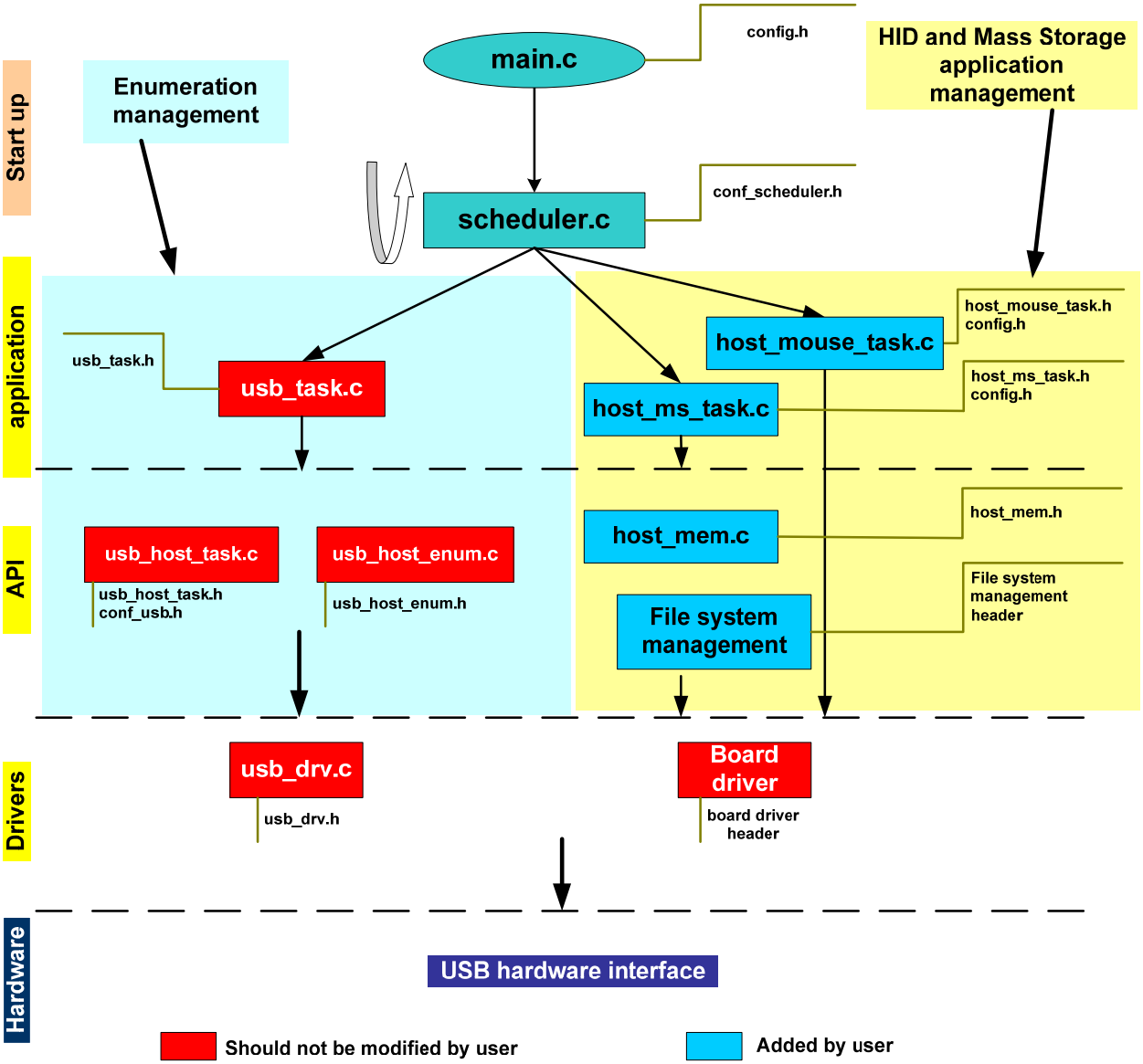


For more detailed information, please read the “USB Mass Storage Class Bulk-Only Transport specification (http://www.usb.org)”.

3 Atmel® Software Architecture

Below is an overview of the architecture of the Host HID and Mass Storage firmware, where appear all files required for the operation.

Figure 3-1. Host HID and Mass Storage architecture



The management of HID class (mouse) is implemented in the “host_mouse_task.c” file, while the Mass Storage class is implemented in the “host_ms_task.c” file. The scheduler periodically calls the usb_task(void), the host_mouse_task(void) and the host_ms_task(void) function. The operations of these three functions are described below:

- usb_task(void)
  - USB mode detection

  - USB host/device enumeration
- host_mouse_task(void)
  - Check if a mouse has plugged in.
  - Get data from mouse, and perform user application
  - Check if the mouse has removed from the host.
- host_ms_task(void)
  - Check if a Mass Storage device has plugged in.
  - Perform user application
  - Check if a Mass Storage device has removed from the host.

3.1 Enumeration

When a device connects to the host, enumeration starts. If the device interface is accepted by the USB host firmware low-level task, that compares the device descriptors with the list of supported interfaces (defined in “conf_usb.h” file), then the host_mouse_task() and host_ms_task() sees a connection notification (Is_new_device_connection_event() macro returns TRUE).

The number of currently accepted device is returned by Get_nb_device() function if enabled the USB_HUB_SUPPORT in conf_usb.h file, and the number of currently accepted interfaces is returned by Get_nb_supported_interface() function. For each interface of each device, program access to the class, subclass and protocol codes of the device thanks to the Get_class(i), Get_subclass(i) and Get_protocol(i) macros.

The program checks if the connected device has a supported class interface, if the device interface is accepted, the program configured the pipes by “host_auto_configure_endpoint” function or “User_configure_endpoint()” function dependant on whether the “HOST_AUTO_CFG_ENDPOINT” is enabled in the conf_usb.h file or not.

When the configuration is finished, host sends the Set_configuration() request to the device, and turn to DEVICE_READY status.

For the host mouse task, when it gets a new device connection notification, it compares the class and protocol to check out if it is a mouse connected. If it is, the task unfreezes the IN pipe and prepared for the data transfer.

Below is the corresponding function code:

Code 3-1. Host HID mouse device detection

```
if(Is_new_device_connection_event())   //Device connection
{
   mouse_connected=0;
   for(i=0;i<Get_nb_supported_interface();i++)
   {
      if(Get_class(i)==HID_CLASS
           && Get_protocol(i)==HID_PROTOCOL_MOUSE)
      {
         mouse_connected=1;
         host_hid_set_idle();
         host_get_hid_report_descriptor();
         LOG_STR_CODE(log_mouse_connect);
```

```
         PIPE_MOUSE_IN=host_get_hwd_pipe_nb(Get_ep_addr(i,0));
         Host_select_pipe(PIPE_MOUSE_IN);
         Host_continuous_in_mode();
         Host_unfreeze_pipe();
         break;
      }
   }
}
```

For the mass storage task, when it sees the new device connection event, to support “multiple mass storage devices” feature, it searches each interface of each device. If the current interface is a mass storage class, it stores the current device index in the device mass storage array dms[n] and records the max mass storage device number in the variable dms_connected. After then, the task configures the IN pipe and OUT pipe and initializes all USB drives (Mass Storage device).

Below is the corresponding function code:

Code 3-2. Host Mass Storage device detection

```
if(Is_new_device_connection_event())
{
   for(k=0;k<=Get_nb_device()-1;k++)
   {
      Host_select_device(k);
      new_dms=TRUE;
      for(i=0;i<Get_nb_supported_interface();i++)
      {
         if(Get_class(i)==MS_CLASS)
         {
            LOG_STR_CODE(log_ms_connect);

            if (dms_connected!=0) // Other DMS connected ?
            {
               for(n=0;n<USB_MAX_DMS_NUMBER;n++)
               {
                  if(dms[n].device_index==k)
                  {
                     new_dms=FALSE;
                  }
               }
            }
            if(new_dms)
            {
               dms_connected++;  // TODO check USB_MAX_DMS_NUMBER
               dms[dms_connected-1].device_index=k;
```

```
//Get correct physical pipes associated to IN/OUT Mass Storage
Endpoints
               if(Is_ep_addr_in(Get_ep_addr(i,0)))
               {  //Yes associate it to the MassStorage IN pipe
                  dms[dms_connected-
1].pipe_in=usb_tree.device[k].interface[i].ep[0].pipe_number;
                  dms[dms_connected-
1].pipe_out=usb_tree.device[k].interface[i].ep[1].pipe_number;
               }
               else
               {  //No, invert...
                  dms[dms_connected-
1].pipe_in=usb_tree.device[k].interface[i].ep[1].pipe_number;
                  dms[dms_connected-
1].pipe_out=usb_tree.device[k].interface[i].ep[0].pipe_number;
               }
//Get number of lun in the Device mass storage connected
               Select_dms(dms_connected-1);
               dms[dms_connected-1].nb_lun=host_get_lun();
               //Compute USB lun number
               lun=0;
               for(n=0;n<dms_connected-1;n++)
               {
                  lun +=dms[n].nb_lun;
               }
               //Initialize all usb drives
               for(n = 0; n < dms[dms_connected-1].nb_lun; n++)
               {
                  host_ms_inquiry();
                  //host_read_format_capacity(lun); //Some devices
require this command beforethe read capacity command
                  host_read_capacity(lun, &capacity);
                  host_ms_request_sense();
                  while (CTRL_GOOD != host_test_unit_ready(lun));
                  host_read_capacity( lun, &capacity );
                  lun++;
               }
               break;
            }
         }
      }
   }
}
```

3.2 Data Transfer

3.2.1 Host HID mouse task

If the mouse is connected, the task checks every time it entered to see if there is a valid IN package received. When the data is available, program read the data by Host_read_byte() function, and perform the user function. After then, host send a IN command by Host_send_in() function and is ready for the next data package.

Code 3-3. Host HID mouse data transfer

```
Host_select_pipe(PIPE_MOUSE_IN);
if(Is_host_in_received())
{
   if(Is_host_stall()==FALSE)
   {
      i=Host_read_byte();
      new_x=(S8)Host_read_byte();
      new_y=(S8)Host_read_byte();
      if(new_x==0)
      { Led0_off(); Led1_off();}
      else if(new_x>0)
      {Led0_on(); Led1_off();}
      else
      { Led0_off(); Led1_on();}

      if(new_y==0)
      { Led2_off(); Led3_off();}
      else if(new_y>0)
      { Led2_on(); Led3_off();}
      else
      { Led2_off(); Led3_on();}
   }
   Host_ack_in_received();
   Host_send_in();
}
```

These operations are implemented for an evaluation purpose. User can use the current function “as is”, but he is also free to implement its own data handler.

3.2.2 Host Mass Storage task

As mentioned before, the Mass Storage application is a set of SCSI commands send by the host to manage the file transfer. The basic data transfer is perform by host_get_data() function and host_send_data() function.

The user application should implement the SCSI encoder, file system decoder (refer to the application note AVR144).

4 Example

4.1 Overview

All this theory may appear complex, so here is a simple example to allow a quick evaluation about USB Host HID Class and Host Mass Storage Class implementation, with an evaluation purpose.

In this configuration, the demo board first is connected to a PC by USB interface, working as a USB Mass Storage device, preparing the demo data. Then, the demo board disconnected from PC and connected to a Mass Storage device to show the Mass Storage data transfer. We can also connect a standard USB mouse to the demo board to show the USB Host HID mouse application.

The USB Host HID and Mass Storage application can be implemented on any AVR USB having the host capability by using a software package available on Atmel website (http://www.atmel.com).

4.2 Hardware

Both software packages can be run on available starter kits. At the time of writing, the Host HID mouse and Mass Storage package can be run on STK®525 or USBKEY package (featuring AT90USB647/1287), and the Mass Storage device which will be connected to the demo board can be a USB flash disk or one of the starter kits running the Mass Storage device firmware.

The host board should be connecting in host mode (miniA plug inserted). An external power supply is needed and configured in order to provide power to the USB Device board.

4.3 Software

4.3.1 Operation description

For USB Host HID mouse task, if you connected a standard USB mouse to the demo board, the mouse movements are visible with the LEDs.

Once enumerated, IN event will be trigged every time the mouse moved or buttons are clicked. The host will read the data by the USB interface. Movement along X axis will lit the LED0 and LED1 (X>0, LED0 on, X<0, LED1 on), while Y direction will lit the LED2 and LED3 (Y>0, LED2 on, Y<0, LED3 on).

For USB Host Mass Storage task, once enumerated, files can be exchanged between on board data flash disk and the Mass Storage Device. The right direction of the joystick allows to read the content of a Mass Storage Device “OUT” directory and to write it to an on board data flash “IN” directory and vice versa for the “left” from data flash to Mass Storage Device

4.3.2 Configuration

Some parameters must be defined on each microcontroller to ensure correct operation. The software package does not need to be modified, it is working “as is” and is configured with the values given below.

- USB configuration, in the “conf_usb.h” file:
  - USB_HOST_FEATURE must be enabled to support USB Host function.

- USB_DEVICE_FEATURE must be enabled to support the connection to the PC.
- The VID_PID_TABLE array must include the supported device's VID and PID if the HOST_STRICT_VID_PID_TABLE is enabled.
- The CLASS_SUBCLASS_PROTOCOL array must include the HID class Mouse protocol (both bootable and non-bootable subclass should be included to support all standard USB mouse) and Mass Storage interface (Mass Storage class, SCSI subclass, bulk only protocol). All these class/subclass/protocol things need to be added by the user for the HUB if needed.
- USB_HUB_SUPPORT should be enabled if you want to use several Mass Storage devices.
- HOST_STRICT_VID_PID_TABLE can be either enabled or disabled. To support different products from different vendors, this Macro is recommended to be disabled.

• USB Mass Storage Configuration:
- HOST_SYNC_MODE in the "config.h" file must be enabled to run the USB Host Mass Storage example program.

5 Conclusion

Atmel USB solutions can offer robust and simplified system for wide range of applications which can cover almost all the market demands and at the same time shorten the time-to-market for client with integrated software frame occupied with necessary application note.

6 Related Documentation

- AVR USB products Datasheet
- USB Software Library for AT90USBxxx Microcontrollers (AVR276)
- USB Mouse Demonstration (AVR270)
- USB Mass Storage Implementation (AVR273)
- USB HID Class specification
- USB Mass Storage Class specification
- Using the ATMEL File System management for AT32UC3x, AT90USBx and ATmega32U4 (AVR114).

Available on

http://www.atmel.com

http://www.usb.org

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
http://www.atmel.com/

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature