AVR280: USB Host CDC Demonstration

1. Introduction

The RS232 interface has disappeared from the new generation of PCs replaced by the USB interface. To follow this change, applications based on UART interface have to migrate to USB. Migration to USB can mean heavy development both on the PC and on the device side. To avoid this development, Atmel offers you solutions based on the CDC class (Communication Device Class) :

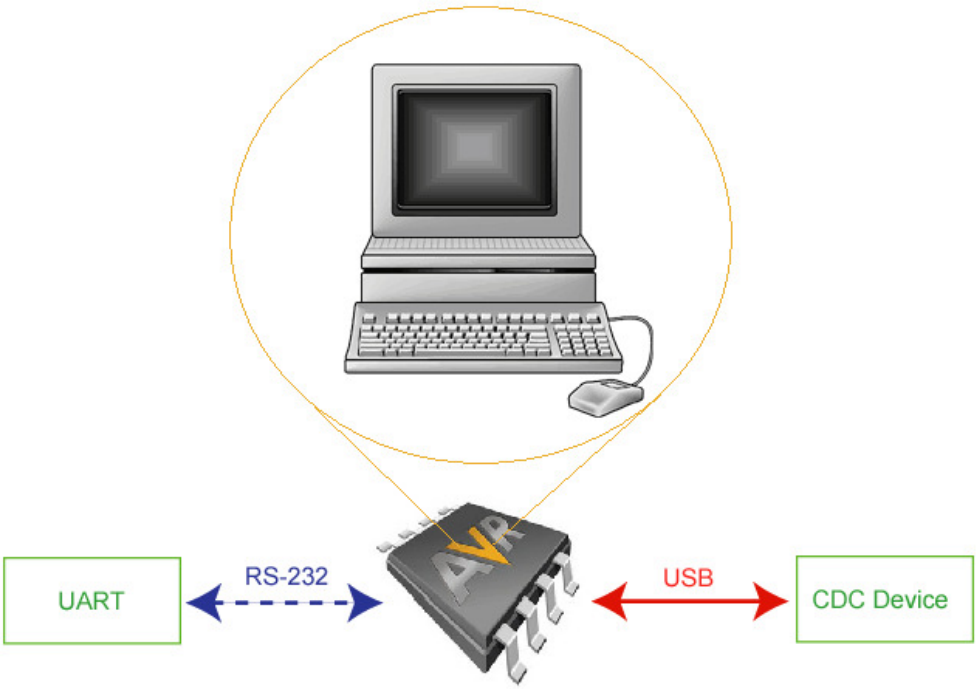
> CDC Device application (See Application Note AVR272) : such a USB device can be connected to a PC and appears as a Virtual Serial Port.

> CDC Host application : this application replaces a PC, and greets a CDC Device, allowing an easy and powerfull communication !

The aim of this document is to describe how to start and implement a Host CDC application using the STK525 or USBKEY starter kit, and finally introduces a simple example of dual USB-UART bridge between two PC.

A familiarity with the AVR USB software framework (http://www.atmel.com) and with the CDC specification (http://www.usb.org) is assumed.

Figure 1-1. Host CDC Application



8-bit AVR® Microcontrollers

Application Note

7727A–USB–09/07

2. Theory of Operation

2.1 CDC Configuration

2.1.0.1 USB

The CDC class configuration includes two interfaces :

- Data interface : consists of 2 pipes (bulk, typically), one per transfer direction, for data exchange
- Communication interface : used to transmit requests to manage the operational state of the device, and events notification. It is shared between two elements :
  - Management element, that consists of endpoint 0, and that configures and controls the device through standards and specific requests.
  - Optionnal notification element, that consists of an interrupt (may also be bulk) endpoint, used by the Device to notify events to the Host. Messages transfered are formatted as a 8-bytes header followed by a variable-length field.

In conclusion, a typical CDC application requires two pipes in addition to the default control endpoint. But note that a CDC Device is accepted by a standard PC only if it includes the optional notification element (using default driver).

2.1.0.2 Models

The CDC Class Specifications introduces various models of communication. Each model is characterized by the type of interface implemented, and the commands or protocol supported. This is specially used to communicate with USB modems that may use a specific protocol layer (for example V42bis), or even USB phones, etc.

But, for users that do not care about such models, and just want to exchange data between two application, these models are irrelevant.

This application note will shortly introduces the reader to these models and to the associated commands and/or requests. However, its primary goal is to explain the Host CDC firmware low level architecture and to demonstrate the simplest configuration with an evaluation example.

2.2 Transfers

2.2.1 Data Transfer

2.2.1.1 Raw Data

Once the Device has been enumerated by the Host, data can easily be sent between the applications.

- The device can fill in its IN Endpoint with any amount of data. The host is assumed to poll as often as possible the IN endpoint and get back any possible data in its IN pipe.
- The host can fill in its OUT Pipe with any amount of data. The device will receive the packet in its OUT Endpoint as soon as the Host completes the transmission.

That is the simplest way for two basic applications to exchange data with the USB performance, but with the easiness of an UART :

- uses a Bulk transport way that can theorically reach up to 1.2Mbytes / sec at Full Speed.
- USB reduced size (and low-cost) connectivity

This is the simplest use case of an UART-to-USB bridge. In such a configuration, once the device has been enumerated and configured, the Managemenent Interface can be left aside, and all the data is exchanged through the Data Interface.

2.2.1.2 Encapsulated Data

For higher level application, the CDC class defines a format of data encapsulation to handle the packet with a protocol data wrapper.

This mode will not be discussed here since it goes beyond the objectives of a simple demonstration. See Table 1 at page 11 of the “CDC Class Specification v1.1” for more information.

2.2.2 Communication Management

This part of the class is not mandatory for simple data exchange applications.

2.2.2.1 Management element

Through the default control endpoint, the Host can send bidirectional requests to the Device. The format of these requests follows the layout defined in the USB Specification 2.0 :

Table 2-1. Management request packet format

bmRequestType	bRequest	wValue	wIndex	wLength	Data

Two types of requests can be considered :

- Requests that set the Device into a specific state or configuration, or that send information :
  - SET_LINE_CODING : this request configures the Device with parity, baudrate, and some other parameters
  - SET_HOOK_STATE : this request puts the Device line into a specific state (on-hook, off-hook, snooping)
  - SEND_ENCAPSULATED_COMMAND : this requests sends a packet within a specific encapsulation protocol
  - etc...*(see Section 6.1 of CDC Class Specification)*
- Requests that get the Device state or configuration, or any other information :
  - GET_LINE_CODING : this request gets the Device configuration
  - GET_ENCAPSULATED_RESPONSE : this request gets a packet within a specific encapsulation protocol
  - etc...*(see Section 6.1 of CDC Class Specification)*

2.2.2.2 Notification element

Through this optional pipe (interrupt IN transfer), the Device can transfer special events notifications to the Host.

Table 2-2. Notification packet format

bmRequestType	bNotification	wValue	wIndex	wLength	Optional Data

Some examples :

- NETWORK_CONNECTION : the device informs the host about the connection state (after a change, for example)

– RESPONSE_AVAILABLE : the device informs the host that a response is available. Host will need to send a GET_ENCAPSULATED_RESPONSE request through the management element to get this response.

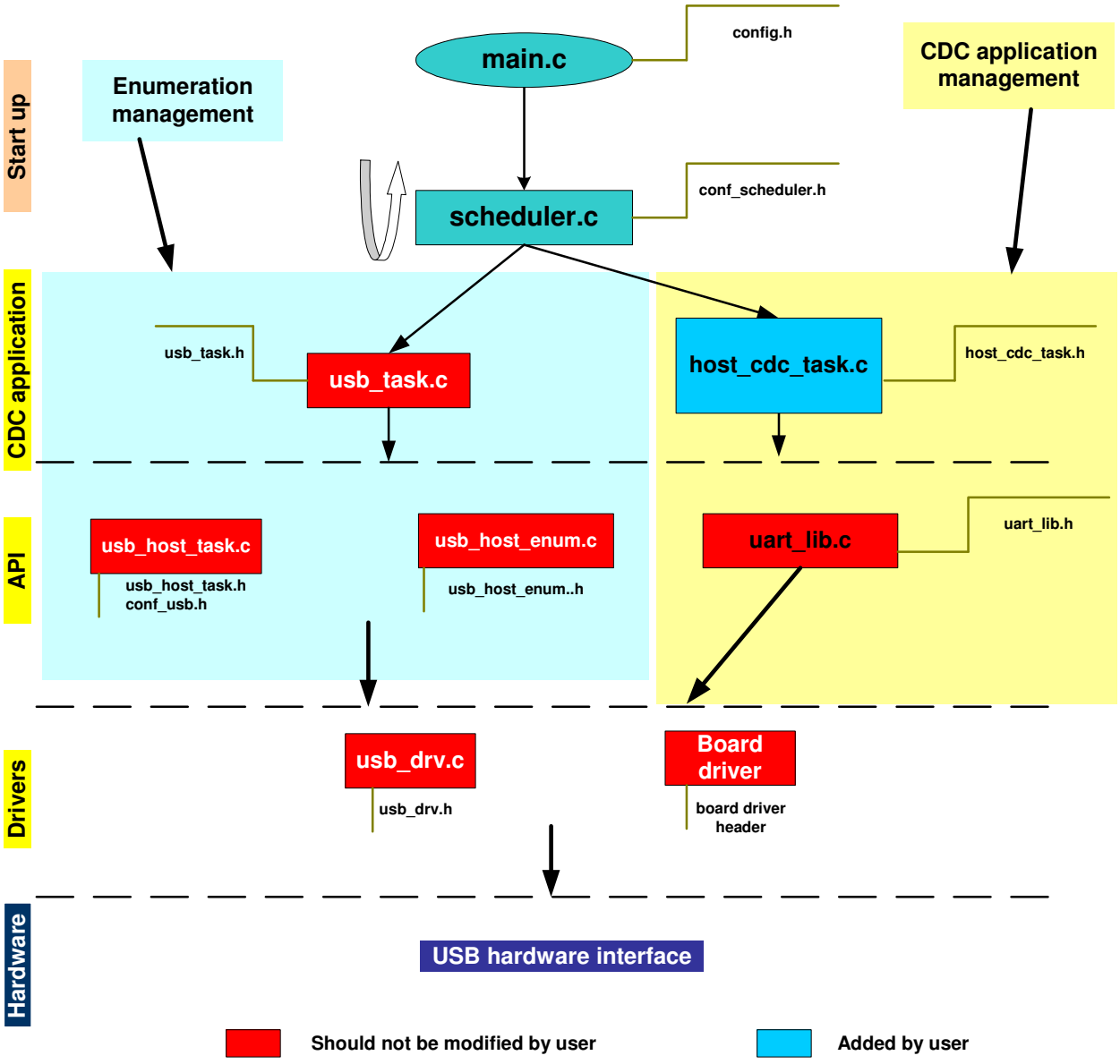
– etc...*(see Section 6.3 of CDC Class Specification)*

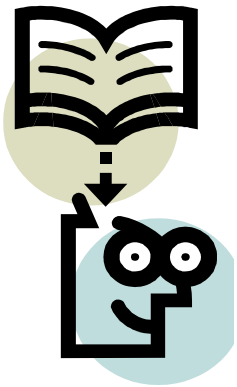
For more details and information about communication management, please read section 6 of the “CDC Class specification”.

3. ATMEL Software Architecture

Below is an overview of the architecture of the Host CDC firmware, where appear all files required for the operation.

Figure 3-1. Host CDC Firmware architecture





The management of the CDC is made in the “host_cdc_task.c” file. The main function, periodically called by the scheduler, is *host_cdc_task(void)*, that handles three main operations :

- Connection (and disconnection) detection
  - Accept or reject any new device according to its interface descriptors
  - Associate the physical pipes with the CDC software channels
- Data transfer
  - Check for incoming data stream
  - Send data if required by the user application
- Management transfer
  - Check if an event notification has been received
  - User can also send request through the default control endpoint

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_cdc_task()* sees a connection notification (*Is_new_device_connection_event()* macro returns TRUE).

The number of currently accepted interfaces is returned by *Get_nb_supported_interface()* function. For each interface number, 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.

First, the program checks if the connected device has a CDC Data interface. If the device interface is accepted, the connection function searches which pipe is IN, and which pipe is OUT, to associate them with more common names : *pipe_cdc_data_bulkin* and *pipe_cdc_data_bulkout*. After that, the IN pipe is unfreezed to enable data reception in the pipe.

Then, the program checks if the connected device has a CDC Communication interface (notification element). If yes, the pipe address is also stored in a common name : *pipe_cdc_comm_int*. User enable the management interface detection by defining the label CDC_USE_MANAGEMENT_INTERFACE. The interface number is also stored in the variable *cdc_interface_comm*, because it will be needed if management requests are used.

Once the connection is accepted, the *cdc_connected* flag is set to “1” and the pipes can be used by the user application.

Below is the corresponding function code :

Code 3-1. CDC device connection detection

```
if(Is_new_device_connection_event())   //Device connection
    {
      for(i=0;i<Get_nb_supported_interface();i++)
      {
        // Data Interface
        if((Get_class(i)==CDC_DATA_CLASS) && (Get_protocol(i)==CDC_DATA_PROTOCOL))
        {
          cdc_connected=1;
          Host_enable_sof_interrupt();
          LOG_STR_CODE(log_cdc_connect);
          if(Is_ep_addr_in(Get_ep_addr(i,0)))
          {  // Yes associate it to the CDC Data IN pipe
            pipe_cdc_data_bulkin = host_get_hwd_pipe_nb(Get_ep_addr(i,0));
            pipe_cdc_data_bulkout = host_get_hwd_pipe_nb(Get_ep_addr(i,1));
          }
          else
          {  // No, invert...
            pipe_cdc_data_bulkin = host_get_hwd_pipe_nb(Get_ep_addr(i,1));
            pipe_cdc_data_bulkout = host_get_hwd_pipe_nb(Get_ep_addr(i,0));
          }
          Host_select_pipe(PIPE_CDC_DATA_IN);
          Host_continuous_in_mode();
          Host_unfreeze_pipe();
          break;
        }
        // Management Interface
#ifdef   CDC_USE_MANAGEMENT_INTERFACE
        if(Get_class(i)==CDC_COMM_CLASS && Get_protocol(i)==CDC_COMM_PROTOCOL)
        {
          cdc_interface_comm = i;      // store interface number
          pipe_cdc_comm_int = host_get_hwd_pipe_nb(Get_ep_addr(i,0));
          Host_select_pipe(PIPE_CDC_COMM);
          Host_continuous_in_mode();
          Host_unfreeze_pipe();
        }
#endif
      }
    }
```

3.2 Data Transfer

Data transfer is very easy to implement.

3.2.1 Receiving data

If the CDC device is connected, the program checks as often as possible (every time the function is entered) if the pipe has received new data.

The current firmware allows two operations with the pipe data :

- Data is stored in an array :

  - the array *cdc_stream_in_array[CDC_STREAM_IN_SIZE]* is filled in by firmware with incoming data
  - the variable *rx_counter* indicates the position of the next byte to be written in the array (initialized to 0 ; the buffer is full when *rx_counter* equals CDC_STREAM_IN_SIZE*)*. Consequently, it indicates for the user the number of written bytes.
  - when user firmware reads the data from the array, it must either use another index variable to accede to the data, or read all the array at one go and clear *rx_counter*. If user decrements *rx_counter* for each byte read from the array, and exits its functions before reading the entire array, problems will occur (for example data corruption on next pipe reception)
- Data is sent to the UART (USB-UART bridge configuration) :
  - all the bytes are sent to the UART
  - this is a blocking function, that is waiting for each byte to be transmitted before trying to accede to the next data (this limitation can be easily enhanced)
  - this mode is enabled by defining the CDC_USE_UART label

Code 3-2. Reading data from the device

```
Host_select_pipe(PIPE_CDC_DATA_IN);
      if (Is_host_in_received() && (Is_host_stall()==FALSE))
      {
#ifdef CDC_USE_UART
            while (Host_data_length_U8() != 0)
            {
              uart_putchar(Host_read_byte());
            }
            Host_ack_in_received();    // pipe is empty
            Host_send_in();            // ready to receive more data
#else
            while ((rx_counter != CDC_STREAM_IN_SIZE) && (Host_data_length_U8() != 0))
            {
              cdc_stream_in_array[rx_counter] = Host_read_byte();
              rx_counter++;
            }
            if (Host_data_length_U8() == 0)
            {
               Host_ack_in_received();    // pipe is empty
               Host_send_in();            // ready to receive more data
            }
#endif
      }
```

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 (or packet wrapper for example), acceding directly to the pipe.

3.2.2 Sending data

The principe of operation is very similar to the data reception stage. Data to be transmitted is first stored in an array, *cdc_stream_out_array[CDC_STREAM_OUT_SIZE]*. A global variable, *tx_counter*, indicates the number of bytes currently stored, and consequently the position of the next data byte to be stored.

There are two possible sources of data for the array :

- User program : user specific firmware can load data into the array, incrementing *tx_counter* for each byte written (initialized to 0 when no data stored)
- UART : if the label CDC_USE_UART is defined, the array will be filled in with each new byte received on the UART.

Moreover, there are two possible conditions for the array data to be transferred to the OUT pipe :

- Buffer full : as soon as the buffer is full (*tx_counter* = CDC_STREAM_OUT_SIZE), all the array data is transferred to the pipe.
- Time-out : a time counter variable, *cdc_cpt_sof*, is incremented at each USB Start Of Frame (every 1ms) interrupt in the *sof_action()* function. When this counter reaches or exceeds the user defined value CDC_NB_MS_BEFORE_FLUSH, and if the array is not empty, all the buffer data is transferred to the pipe.

The pipe transmission on the USB is ensured by the *cdc_pipe_out_usb_flush()* function.

Code 3-3. Sending data to the device

```
#ifdef CDC_USE_UART
        // Check if new byte in USART, to be stored for USB
        if (uart_test_hit() && (tx_counter != CDC_STREAM_OUT_SIZE))
        {
          cdc_stream_out_array[tx_counter] = uart_getchar();
          tx_counter++;
        }
#endif


      // Check if pipe flush is needed (buffer full or time-out period elapsed)
      if(((cdc_cpt_sof>=CDC_NB_MS_BEFORE_FLUSH) && (tx_counter!=0)) || (tx_counter == CDC_STREAM_OUT_SIZE))
      {
        cdc_cpt_sof=0;
        cdc_pipe_out_usb_flush();
      }
```

Code 3-4. Additionnal functions

```
void cdc_pipe_out_usb_flush (void)
{
  Host_select_pipe(PIPE_CDC_DATA_IN);  // BULK IN must be frozen else BULK OUT may not be sent
  Host_freeze_pipe();
  if (PIPE_GOOD == host_send_data(PIPE_CDC_DATA_OUT, tx_counter, cdc_stream_out_array))
  {
    tx_counter = 0;    // if frame not sent, will try again next time (no data loss)
  }
  Host_select_pipe(PIPE_CDC_DATA_IN);
  Host_unfreeze_pipe();
}

void sof_action(void)
{
  cdc_cpt_sof++;
}
```

3.3 Communication Management

3.3.1 Management element

Through the endpoint 0 are transferred CDC specific requests, defined in the CDC Class Specification. These requests can be defined in the following form :

Code 3-5. Management request layout

```
#define host_cdc_get_line_coding()      (usb_request.bmRequestType = BM_REQUEST_TYPE,\
                                         usb_request.bRequest     = B_REQUEST,\
                                         usb_request.wValue       = W_VALUE,\
                                         usb_request.wIndex       = W_INDEX,\
                                         usb_request.wLength      = W_LENGTH,\
                                         usb_request.uncomplete_read = FALSE,\
                                         host_send_control(data_stage))
```

Where the parameters in ***BOLD & ITALIC*** must be replaced by the parameters matching with the specific request.

The W_INDEX field is typically equal to "*cdc_interface_comm*", that is assignated to the management interface (if enabled) of the device.

The W_LENGTH field contains the number of data bytes to be transmitted in the request.

Data bytes (to be sent, or received) are stored in the "*data_stage*" array.

User can easily add other request by editing the "host_cdc_task.h" file. However, some of them are already integrated in the Atmel Host CDC Firmware :

3.3.1.1 Encapsulated requests

These requests are used to transmit specific requests that are encapsulated according to a specific protocol.

Table 3-1. SEND_ENCAPSULATED_COMMAND request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001b	0x00	0x00	Interface	Amount of data associated with this recipient	Control protocol-based command

Table 3-2. GET_ENCAPSULATED_RESPONSE request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001	0x01	0x00	Interface	Amount of data associated with this recipient	Protocol dependent data

3.3.1.2 Communication parameters requests

These requests may be used to set (or to get) a configuration to (or from) the CDC Device for its UART communication.

Table 3-3. SET_LINE_CODING request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001b	0x20	0x00	Interface	0x07	Line coding structure

Table 3-4. GET_LINE_CODING request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001b	0x21	0x00	Interface	0x07	Line coding structure

Table 3-5. Line coding structure

Offset	Field	Size	Value	Description
0	dwDTERate	4	Number	Data terminal rate, in bits per second
4	bCharFormat	1	Number	Stop bits : – 0 - 1 stop bit – 1 - 1.5 stop bits – 2 - 2 stop bits
5	bParityType	1	Number	Parity – 0 - None – 1 - Odd – 2 - Even – 3 - Mark – 4 - Space
6	bDataBits	1	Number	Data bits number (5, 6, 7, 8, or 16)

3.3.2 Notification element

The notification element, that consists of the additional IN endpoint, receives events notification from the device.

The current Atmel Host CDC software does not include a packet wrapper for this pipe. But a location is ready to greet user's notification handler.

Please read the CDC Class Specification for more information about features offered by this interface.

Code 3-5. Notification pipe handler template

```
Host_select_pipe(PIPE_CDC_COMM);
    if (Is_host_in_received())
    {
      // Handle here notification messages sent by device
      // Notifications messages have the following structure :
      // bmRequestType - bNotification - wValue - wIndex - wLength - Data    (wLength is the number of bytes of the Data field)

      //   - NETWORK_CONNECTION : indicates that device has connected to network
      //   - RESPONSE_AVAILABLE : indicates that device has a ready encapsulated response (wait for host request)
      //   - SERIAL_STATE : indicates state of device' UART (errors, carriers and misc. signals)
      //   - etc...

      // ...and now...just coding...
      Host_ack_in_received();
      Host_send_in();
    }
```

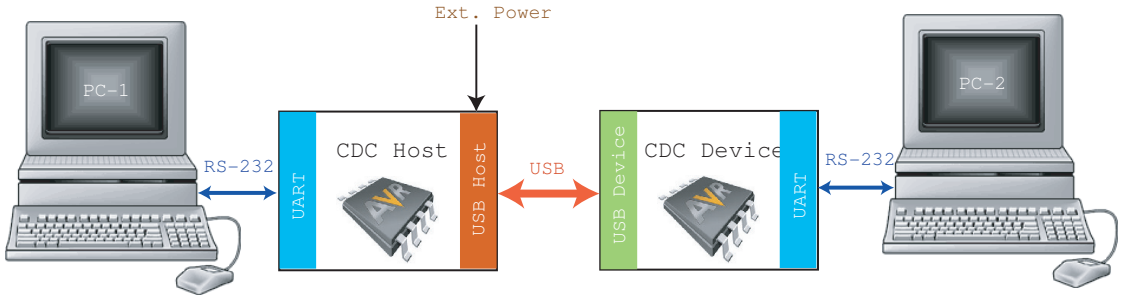
4. Example

4.1 Overview

All this theory may appear complex, so here is a simple example to allow a quick evaluation and cheer users about CDC Class implementation, with an evaluation purpose : a dual USB-to-UART bridge.

In this configuration, the CDC Host application has an UART connected to the serial port of a PC-1. A CDC Device has also an UART that is connected to another serial port of the same PC-1, or to the serial port of another PC-2. Both CDC applications are connected together through a USB connection. This application, totally useless as an industrial or consumer application, simply shows the mechanisms of CDC data transfers.

Figure 4-1. Dual USB-UART bridge evaluation example



Note: Additional CDC device boards can be used to provide RS232 ports on the PCs if they only have USB ports available.

The CDC Device can be implemented on any AVR USB using a software package available on Atmel website (http://www.atmel.com). The CDC Host uses the corresponding software package that is also available on the Internet.

4.2 Hardware

Both software packages can be run on available starter kits. At the time of writing, the Host CDC package can be run on STK525 or USBKEY package (featuring AT90USB647/1287), and the

Device CDC package can be run on STK525, USBKEY (AT90USB64x/128x) or STK526 (AT90USB82/162).

The USB Device board should be configured in Bus Powered mode for the simplest operation. The USB Host board must be Self Powered (external power supply) and configured in order to provide power to the USB Device board.

4.3 Software

4.3.1 Microcontroller

4.3.1.1 Operation description

This example does not exchange data through the management interface. However, this interface may be implemented, in order to ensure compatibility with other CDC applications.

Once enumerated, every byte received from PC-1 will be transferred from UART to the OUT pipe of the CDC Host, then received in the OUT endpoint of the CDC Device, and finally tranferred to PC-2 through the UART. Bytes sent by PC-2 follow the opposite direction and arrive into the PC-1 serial port buffer.

4.3.1.2 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.

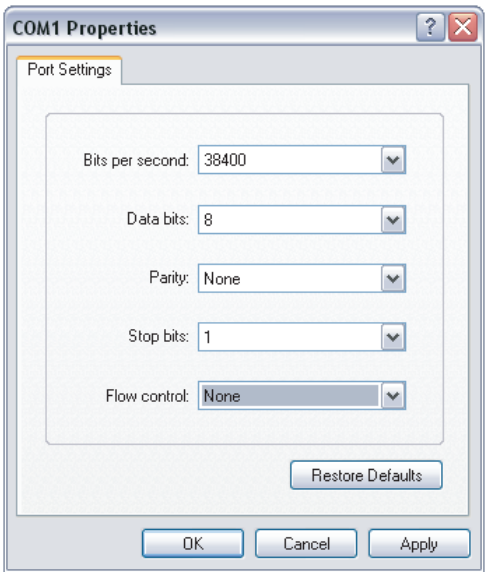
- UART Baudrate : in “config.h” file, defines the label BAUDRATE to the desired baudrate. Packages come with a default value of 38,4 kbps (BAUDRATE = 38400).
- USB configuration, in the ”conf_usb.h” file :
  - the CLASS_SUBCLASS_PROTOCOL array must include the CDC Data Interface and CDC Communication Interface values
  - the Host_sof_action() must be linked to the sof_action() function (whose prototype must be declared too)
- Host CDC configuration : in “config.h” file or any header file accessible by “host_cdc_task.c” file must be defined the following values or labels :
  - CDC_USE_MANAGEMENT_INTERFACE must be defined (even if no data is exchanged through this interface)
  - CDC_USE_UART must be defined (to enable the USB-UART bridge feature)
  - CDC_NB_MS_BEFORE_FLUSH is not very important for a human controlled application, so it can be set to 0x05 by default (the influence of this value may depend on the application)
  - CDC_STREAM_OUT_SIZE and CDC_STREAM_IN_SIZE depend on the expected data rate : higher the data rate expected (number of character that may be sent per second in reality), higher these two values. But don’t exceed the data pipe or endpoint size ! (to prevent loss of data) Default value is 16 (0x10).

4.3.2 PC serial port

To exploit easily any serial port of a PC, you may run a terminal. Under Windows, you can launch Hyper Terminal (Accessories => Communications). First you will have to select the COM port on which is connected the application (host or device). Then, the detailled configuration of

the port must be correctly entered. Using our package unmodified, you must set the following configuration :

Figure 4-2. Hyper Terminal configuration



Then appears the terminal dialog screen, on which you can enter character to be sent on open serial port, or see characters that have been received on the port.

Every character that you will enter in the screen of PC-1 will appear on PC-2, and vice-versa.

5. Conclusion

In conclusion, the CDC Class is a wide specification that covers many and many different configurations, and match with several communication standards, to support devices such as modem, mobile phones, LAN interfaces...

Although the work load required for the implementation of such devices can be very significant, a basic implementation of data transmitter is easily accessible. This application note has been created to help people who want to use the Host capability of Atmel AVR USB products to set up a powerfull communication mean, easy to use and reliable, using today's technologies. Applications such an USB-to-UART bridge (or similar) can be easily created.

6. Related Documentation

- AVR USB products Datasheet
- USB Sotware Framework
- CDC Device application note (AVR272)
- USB CDC class specification

Available on :

- http://www.atmel.com
- http://www.usb.com

Atmel Corporation

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

Regional Headquarters

Europe

Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia

Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

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

Atmel Operations

Memory

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards

Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive

Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom

Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature

7727A–USB–09/07