Basic USB Application

AT91 ARM® Thumb® Microcontrollers

Application Note

Scope

The Universal Serial Bus (USB) has been successfully implemented in most PC systems and is replacing the older parallel and serial ports. For a standard serial port, communications are performed directly by the application running on the computer. In order to be plug-and-play and hot-plug, the USB bus introduces a process that uniquely identifies a device to the Host computer in order for it to learn the capabilities of the device and to load the appropriate driver. This identification process is called enumeration and uses a standard set of commands described in Chapter 9 of the USB specification, “USB Device Framework”.

The USB is likely to replace other serial, parallel and game interfaces in PC and peripheral markets. The USB organization defines how devices and interfaces using the class or common capability are to be implemented and how developers of generic adaptative device drivers interact with compliant implementations. Several classes are approved and widely integrated in most popular operating systems: HID (Human Interface Device), MSD (Mass Storage Device), CDC (Communication Device Class), etc.

This application note describes how to make the USB bus appear as an RS-232 interface on the host PC. The source code corresponding to this application note is delivered in the AT91xxx_BasicUSB example in the AT91 library. It includes an application running on an AT91 demonstration board, PC drivers and a simple USB application running on the PC.

Applicable Documents

Table 1. Applicable Documents

Owner - Reference	Denomination
ARM Document Guidelines	Ref ORG00001 Version 1.0, 16-Oct-2002
USB Specification	Universal Serial Bus Revision 2.0 Specification
CDC Specification	Class Definition for Communication Devices 1.1
Common Class Specification	Common Class Base Specification 1.0

Reference Documents

Table 2. Reference Documents

Owner - Reference	Denomination
USB Device Port (UDP) Programmer Datasheet	Rev. 6083B-12/04
USB Device Port Summary	Rev. 6083AS-05/04
AT91SAM7S64-Datasheet	Rev.6070A-09/04
SAMBA™ User Guide	Rev. xxxx

USB Enumeration

USB Specification

When a USB device is attached to or removed from the USB, the host uses a process known as bus enumeration to identify and manage the device state changes necessary. When a USB device is attached to a powered port, the following actions are taken:

- The hub to which the USB device is now attached informs the host of the event via a reply on its status change pipe (refer to the Section 11.12.3 of the USB specification for more information). At this point, the USB device is in the powered state and the port to which it is attached is disabled.
- The host determines the exact nature of the change by querying the hub.
- Now that the host knows the port to which the new device has been attached, the host then waits for at least 100ms to allow completion of an insertion process and for power at the device to become stable. The host then issues a port enable and reset command to that port. Refer to Section 7.1.7.5 of the USB specification for sequence of events and timings of connection through device reset.
- The hub performs the required reset processing for that port (see Section 11.5.1.5 of the USB specification). When the reset signal is released, the port has been enabled. The USB device is now in default state and can draw no more than 100 mA from V_{BUS}. All of its registers and states have been reset and it answers to the default address.
- The host assigns a unique address to the USB device, moving the device to the address state.
- Before the USB device receives a unique address, its Default Control Pipe is still accessible via the default address. The host reads the device descriptor to determine what actual maximum data payload size this USB default pipe can use.

- The host reads the configuration information from the device by reading each configuration zero to n-1, where n is the number of configurations. This process may take several milliseconds to complete.
- Based on the configuration information and how the USB device will be used, the host assigns a configuration value to the device. The device is now in the configured state and all the endpoints in this configuration have taken their described characteristics. The USB device may now draw the amount of V_{BUS} power described in its descriptor for the selected configuration. From the device's point of view, it is now ready for use.

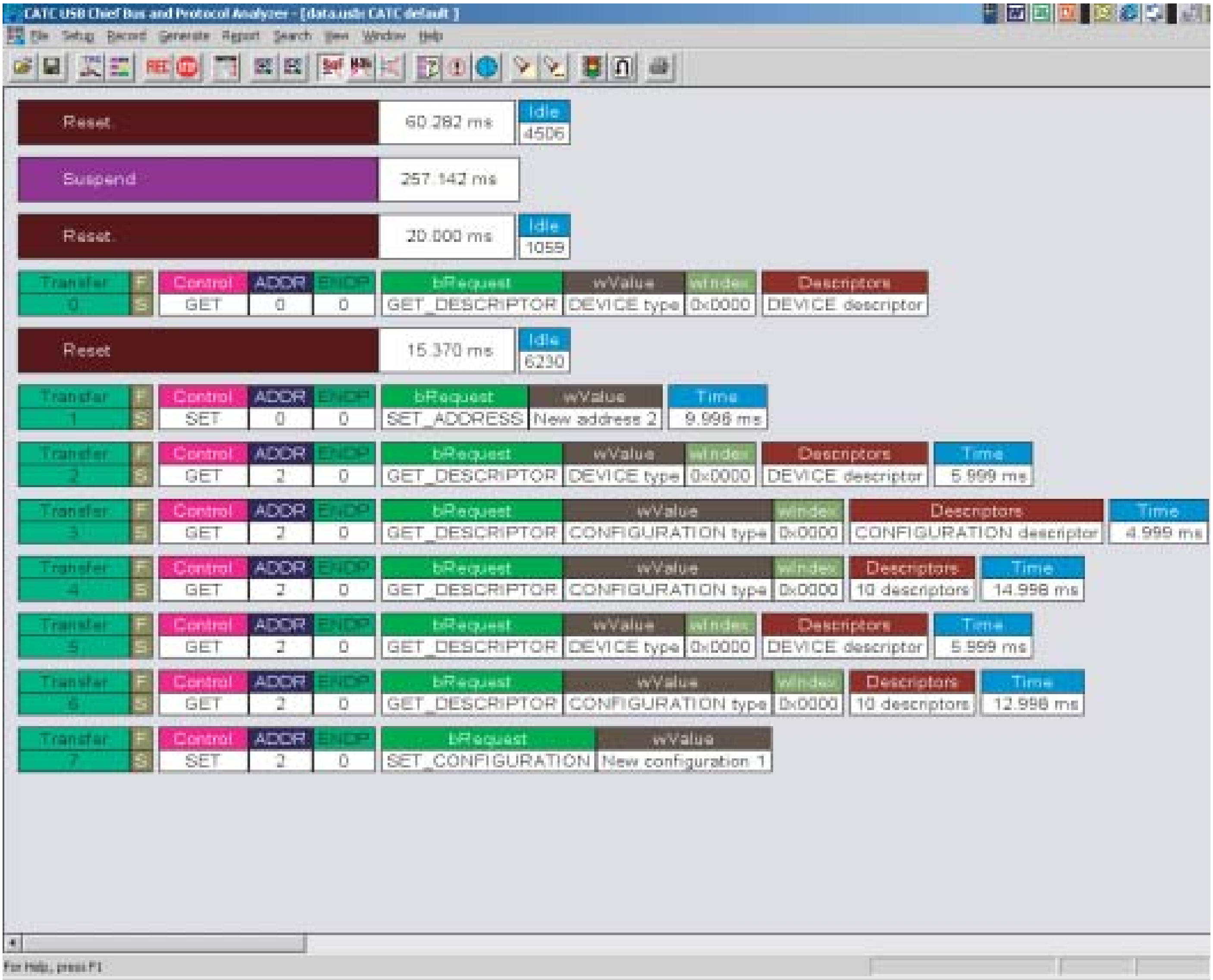
When the USB device is removed, the hub sends a notification to the host. Detaching a device disables the port to which it had been attached. Upon receiving the detach notification, the host will update its local topological information.

The enumeration process is used by the host when a device is attached to the USB bus. This process allows the host to identify and manage the device.

Device Identification

The host sends standard requests on the default control endpoint in order to identify the device and to load the appropriate driver (Refer to Chapter 9 of the USB Specification for more information). The device answers each request with the corresponding descriptor tables. The descriptor tables contain all the information relating to the device: characteristics of the device and number and characteristics of each configuration, interface and endpoint.

Figure 1. Enumeration Transfer


CATC USB Chief Bus and Protocol Analyzer - [data.usb: CATC default]
File Setup Record Generate Report Search View Window Help
For Help, press F1

```
Reset.        60.282 ms   Idle 4506
Suspend       257.142 ms
Reset.        20.000 ms   Idle 1059

Transfer F | Control | ADDR | ENDP | bRequest          | wValue             | wIndex | Descriptors
0        S | GET     | 0    | 0    | GET_DESCRIPTOR    | DEVICE type        | 0x0000 | DEVICE descriptor

Reset         15.370 ms   Idle 6230

Transfer F | Control | ADDR | ENDP | bRequest          | wValue             | Time
1        S | SET     | 0    | 0    | SET_ADDRESS       | New address 2      | 9.998 ms

Transfer F | Control | ADDR | ENDP | bRequest          | wValue             | wIndex | Descriptors              | Time
2        S | GET     | 2    | 0    | GET_DESCRIPTOR    | DEVICE type        | 0x0000 | DEVICE descriptor        | 5.999 ms

Transfer F | Control | ADDR | ENDP | bRequest          | wValue             | wIndex | Descriptors              | Time
3        S | GET     | 2    | 0    | GET_DESCRIPTOR    | CONFIGURATION type | 0x0000 | CONFIGURATION descriptor | 4.999 ms

Transfer F | Control | ADDR | ENDP | bRequest          | wValue             | wIndex | Descriptors              | Time
4        S | GET     | 2    | 0    | GET_DESCRIPTOR    | CONFIGURATION type | 0x0000 | 10 descriptors           | 14.998 ms

Transfer F | Control | ADDR | ENDP | bRequest          | wValue             | wIndex | Descriptors              | Time
5        S | GET     | 2    | 0    | GET_DESCRIPTOR    | DEVICE type        | 0x0000 | DEVICE descriptor        | 5.999 ms

Transfer F | Control | ADDR | ENDP | bRequest          | wValue             | wIndex | Descriptors              | Time
6        S | GET     | 2    | 0    | GET_DESCRIPTOR    | CONFIGURATION type | 0x0000 | 10 descriptors           | 12.998 ms

Transfer F | Control | ADDR | ENDP | bRequest          | wValue
7        S | SET     | 2    | 0    | SET_CONFIGURATION | New configuration 1
```

The standard descriptor types are:

- Device descriptor
- Configuration descriptor
- Interface descriptor
- Endpoint descriptor

Other descriptor types can be added corresponding to a specific USB class.

Each device is identified by a Vendor ID and a Device ID found in the device descriptor. Vendor IDs are delivered by the USB organization. Device IDs are managed by each device vendor. The device descriptor also contains information on whether the device belongs to a standard device class. The host then scans for an available driver corresponding to the Vendor ID and Product ID of the device or scans for a matching standard device class driver (e.g., mass storage driver, CDC driver, etc.) in its driver database. Microsoft stores driver information in .inf files.

For example, if the driver matches the mass storage class driver, then the device can be delivered without a specific host driver. When connected, the new device appears as a new disk mounted on the current file system.

If the device integrates extended features or does not match any class driver standard, then a host driver must be delivered.

Implementation in the AT91xx_BasicUSB Example

The AT91xx_BasicUSB example is the core of the SAMBA_Boot program. Refer to the SAMBA documentation for further information on SAMBA-Boot. This application is a very small monitor that is waiting for a command from the host, executes it and returns the result. This example can be used to build a powerful loader or to make the switch from RS-232 to USB.

The device enumerates as a CDC driver to take advantage of the installed PC RS-232 software to talk over the USB. The CDC class is implemented in all releases of Windows®, from Windows® 98SE to Windows® XP. The CDC document, available at www.usb.org, describes a way to implement devices such as ISDN modems and virtual COM ports.

The cdc_enumerate.c file in the AT91xx_BasicUSB example includes all USB elements. This file is designed to be easily integrated in another application. The interface to USP pipes is done through a read and a write method of a AT91S_CDC object described in the cdc_enumerate.h file.

Enumeration Process

The USB protocol is a master/slave protocol. This is the host that starts the enumeration by sending requests to the device through the control endpoint. The device is supposed to handle standard requests as defined in the USB Specification .

Request	Definition
GET_DESCRIPTOR	This request returns the current device configuration value.
SET_ADDRESS	This request sets the device address for all future device access.
SET_CONFIGURATION	This requests sets the device configuration.
GET_CONFIGURATION	This request returns the current device configuration value.
GET_STATUS	This request returns status for the specified recipient.
SET_FEATURE	This feature is used to set or enable a specific feature.
CLEAR_FEATURE	This request is used to clear or disable a specific feature.

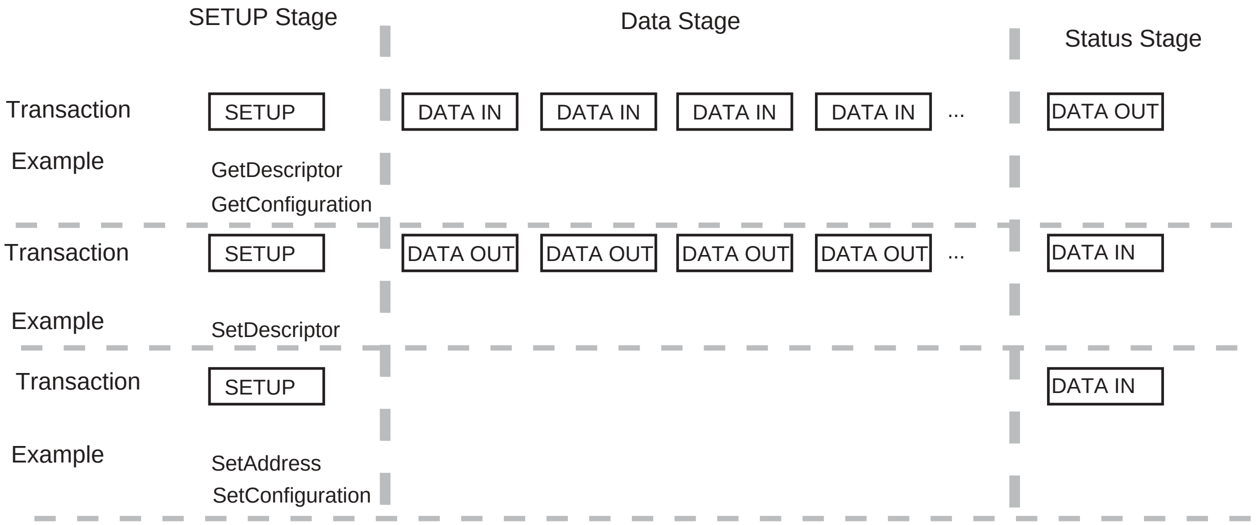
The device is also supposed to handle some class requests defined in the CDC class.

Request	Definition
SET_LINE_CODING	Configures DTE rate, stops bits, parity and number of character bits.
GET_LINE_CODING	Requests current DTE rate, stops bits, parity and number of character bits.
SET_CONTROL_LINE_STATE	RS-232 signal used to tell the DCE device the DTE device is now present..

Unhandled requests have to be STALLed.

A control transfer (request) is composed of three stages: the setup stage, the data stage and the status stage.

Figure 2. Different Control Transfer



Handling SETUP Transactions

Each time a new request is received by the device, the status flag corresponding to the endpoint 0 (control endpoint) is raised. The SETUP flag is set in the UDP_CSR[0] register. The application running on the device

- gets the setup packet
- acknowledges the packet clearing the SETUP flag
- decodes the request and sets the DIR flag if the next transaction is supposed to be a DATA IN transfer
- branches to the handling function.

If the request is not supported, the device must STALL the request.

Handling DATA IN Transactions

Data to be sent are written in the Control endpoint FIFO through the UDP_FDR[0] register. Once written, the host notifies the UDP that data have been written by setting TXPKTRDY flag in the UDP_CSR[0] register. Data have been sent once TXCOMPLETE flag has been set.

The function AT91F_USB_SendData() is used to handle the data IN stage of a control transaction. This function can be aborted if an anticipated DATA OUT packet has been sent by the host before all data payload has been sent to the host.

The function AT91F_USB_SendZlp() is used to send a Zero Length Packet to the host during a status stage.

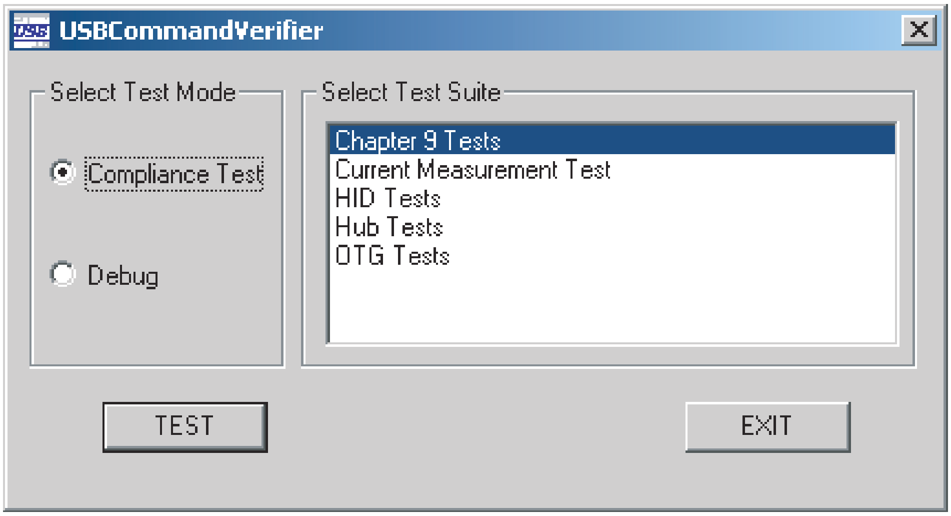
Handling Data Out Transactions

Data payload received is stored in the USB FIFO. The flag RCVDATABK0 is set. Once copied by the application from the UDP_FDR[0] register, the RCVDATABK0 flag is cleared by the application.

Validation

The http:\\www.usb.org web site distributes a free software to test Chapter 9 implementation. This software tests the standard transactions described in Chapter 9 of the USB specification.

Figure 3. USB Command Verifier



USBCommandVerifier

Select Test Mode

● Compliance Test

○ Debug

Select Test Suite

Chapter 9 Tests
Current Measurement Test
HID Tests
Hub Tests
OTG Tests

Descriptor Tables

The CDC document, available at www.usb.org, describes a way to implement devices such as ISDN modems and virtual COM ports.

In order to be considered a COM port, the USB device declares two interfaces:

- Abstract Control Model Communication through endpoint 3 (declared as interrupt IN)
- Abstract Control Model Data through endpoint 1 and endpoint 2 (declared as bulk endpoints)

```
const char devDescriptor[] = {
  /* Device descriptor */
  0x12,    // bLength
  0x01,    // bDescriptorType
  0x10,    // bcdUSBL
  0x01,    //
  0x02,    // bDeviceClass:    CDC class code
  0x00,    // bDeviceSubclass: CDC class sub code
  0x00,    // bDeviceProtocol: CDC Device protocol
  0x08,    // bMaxPacketSize0
  0xEB,    // idVendorL
  0x03,    //
  0x24,    // idProductL
  0x61,    //
  0x10,    // bcdDeviceL
  0x01,    //
  0x00,    // iManufacturer    // 0x01
  0x00,    // iProduct
  0x00,    // SerialNumber
  0x01     // bNumConfigs
};


const char cfgDescriptor[] = {
  /* ============== CONFIGURATION 1 =========== */
  /* Configuration 1 descriptor */
  0x09,    // CbLength
  0x02,    // CbDescriptorType
```

```
0x43,   // CwTotalLength 2 EP + Control
0x00,
0x02,   // CbNumInterfaces
0x01,   // CbConfigurationValue
0x00,   // CiConfiguration
0xC0,   // CbmAttributes 0xA0
0x00,   // CMaxPower
/* Communication Class Interface Descriptor Requirement */
0x09, // bLength
0x04, // bDescriptorType
0x00, // bInterfaceNumber
0x00, // bAlternateSetting
0x01, // bNumEndpoints
0x02, // bInterfaceClass
0x02, // bInterfaceSubclass
0x00, // bInterfaceProtocol
0x00, // iInterface
/* Header Functional Descriptor */
0x05, // bFunction Length
0x24, // bDescriptor type: CS_INTERFACE
0x00, // bDescriptor subtype: Header Func Desc
0x10, // bcdCDC:1.1
0x01,
/* ACM Functional Descriptor */
0x04, // bFunctionLength
0x24, // bDescriptor Type: CS_INTERFACE
0x02, // bDescriptor Subtype: ACM Func Desc
0x00, // bmCapabilities
/* Union Functional Descriptor */
0x05, // bFunctionLength
0x24, // bDescriptorType: CS_INTERFACE
0x06, // bDescriptor Subtype: Union Func Desc
0x00, // bMasterInterface: Communication Class Interface
0x01, // bSlaveInterface0: Data Class Interface
/* Call Management Functional Descriptor */
0x05, // bFunctionLength
0x24, // bDescriptor Type: CS_INTERFACE
0x01, // bDescriptor Subtype: Call Management Func Desc
0x00, // bmCapabilities: D1 + D0
0x01, // bDataInterface: Data Class Interface 1
/* Endpoint 1 descriptor */
0x07,   // bLength
0x05,   // bDescriptorType
0x83,   // bEndpointAddress, Endpoint 03 - IN
0x03,   // bmAttributes      INT
0x08,   // wMaxPacketSize
0x00,
0xFF,   // bInterval
/* Data Class Interface Descriptor Requirement */
0x09, // bLength
0x04, // bDescriptorType
0x01, // bInterfaceNumber
0x00, // bAlternateSetting
0x02, // bNumEndpoints
0x0A, // bInterfaceClass
0x00, // bInterfaceSubclass
```

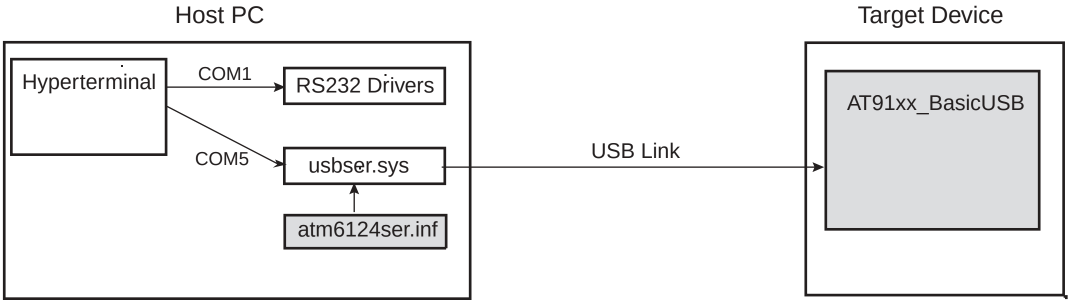
```
  0x00, // bInterfaceProtocol
  0x00, // iInterface
  /* First alternate setting */
  /* Endpoint 1 descriptor */
  0x07,   // bLength
  0x05,   // bDescriptorType
  0x01,   // bEndpointAddress, Endpoint 01 - OUT
  0x02,   // bmAttributes      BULK
  AT91C_EP_OUT_SIZE,   // wMaxPacketSize
  0x00,
  0x00,   // bInterval
  /* Endpoint 2 descriptor */
  0x07,   // bLength
  0x05,   // bDescriptorType
  0x82,   // bEndpointAddress, Endpoint 02 - IN
  0x02,   // bmAttributes      BULK
  AT91C_EP_IN_SIZE,   // wMaxPacketSize
  0x00,
  0x00    // bInterval
};
```

USB PC Application

The device enumerates as CDC (Communication Data Class) class compliant. It is then possible to use the USBSER.sys driver provided in the standard Windows distribution or the atm6124.sys driver provided by Atmel to speed up the communication in a particular case. The driver is selected depending on which INF file is selected the first time the device is connected to the host PC.

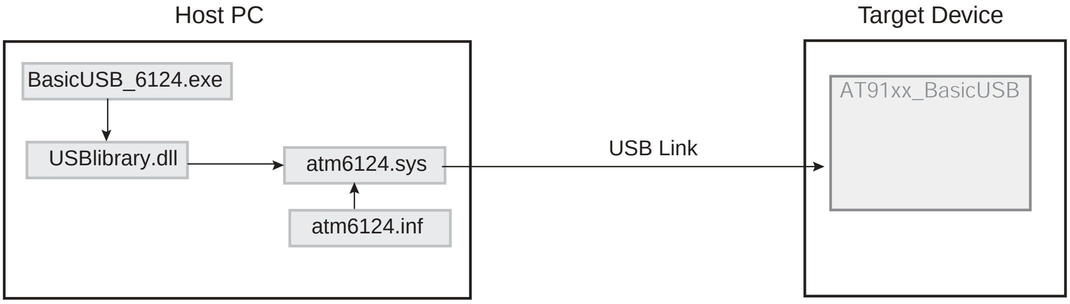
- If atm6124ser.inf is selected, communication to the device is done using the standard Windows usbser.sys driver. In this case, a new serial COM port is added to the existing ones. It is then possible to connect using Hyperterminal or other applications interfaced with serial drivers.

Figure 4. Application Using atm6124ser.inf: Terminal Application



- If atm6124.inf is selected, Windows asks for the atm6124.sys driver. The PC application must be designed to interface with this driver. To ease development, a DLL with a very simple interface is provided. A demo is delivered to illustrate driver performances. This solution is the basis of the SAMBA™ loading solution. It is delivered for demonstration purposes as is.

Figure 5. Application Using atm6124.inf: Loader Application



To interface with CDC devices, some commercial solutions also offer performance, support and certification.

Terminal Application Using Standard Windows CDC Driver

The goal is to connect a standard PC application (hyperterminal) to the AT91xx_BasicUSB application running on the target. All data sent to the device is echoed.

Installation

When plugging in a new device, Windows checks all its INF files to load the appropriate driver. The INF file contains the Vendor ID (VID), the Product ID (PID), or the USB class definition. If the VID/PID or the USB Class Definition of the USB device matches with an INF file, Windows loads the driver described in this file. However, Microsoft does not provide a standard INF file for a CDC driver, so Atmel provides an INF file that allows this driver to load under Windows 2000 and Windows XP. When plugged in for the first time, the user instructs the operating system which driver to use by selecting this INF file. The application manufacturer, using its own VID/PID, modifies these values in the embedded application and the INF file provided.

```
[Version]
DriverVer =10/06/1999,5.00.2157.0
LayoutFile=Layout.inf
Signature="$CHICAGO$"
Class=Modem
ClassGUID={4D36E96D-E325-11CE-BFC1-08002BE10318}
Provider=%Mfg%



[Manufacturer]
%Mfg% = Models

[ControlFlags]
ExcludeFromSelect=USB\VID_03EB&PID_6119

[DestinationDirs]
FakeModemCopyFileSection=12
DefaultDestDir=12

[Models]
%AT91MSG% = AT91SAM,USB\VID_03EB&PID_6124

[AT91SAM.NT]
CopyFiles=FakeModemCopyFileSection
AddReg=USB,ATMEL.resp,AT91SAM.AddReg

[AT91SAM.NT.Services]
AddService=usbser, 0x00000000, LowerFilter_Service_Inst

[AT91SAM.NT.HW]
AddReg=LowerFilterAddReg

[LowerFilterAddReg]
HKR,,"LowerFilters",0x00010000,"usbser"
```

```
[LowerFilter_Service_Inst]
DisplayName=%USBFilterString%
ServiceType= 1
StartType  = 3
ErrorControl = 0
ServiceBinary = %12%\usbser.sys

[FakeModemCopyFileSection]
usbser.sys,,,0x20

[Strings]
Mfg   = "ATMEL CORPORATION"
AT91MSG = "ATMEL AT91 USB serial emulation"
USBFilterString ="AT91 USB serial emulation"

[USB]
HKR,,FriendlyDriver,,Unimodem.vxd
HKR,,DevLoader,,*vcomm
HKR,,ConfigDialog,,serialui.dll
HKR,,AttachedTo,,COM5
HKR,,EnumPropPages,,"serialui.dll,EnumPropPages"
HKR,,DeviceType, 0, 01      ;
HKR,,PortSubClass,1,02

[AT91SAM.AddReg]  ;AT91SAM USB serial emulation
HKR,, Properties, 1, 00,00,00,00, 00,00,00,00, 00,00,00,00,
00,00,00,00, 00,00,00,00, 00,00,00,00, 00,c2,01,00, 00,C2,01,00
```

PC BasicUSB Application Using Atmel atm6124.sys Driver

In some applications, devices are driven by the host and are not supposed to return asynchronous events to the host. This is the case in the AT91xxx_BasicUSB example: the device waits for data sent by the host and replies only when requested by the host.

Note: This example is the core of the SAMBA-Boot, a small monitor running on AT91 parts that waits for a command and then executes it.

The slot time reserved in the USB frame for an eventual asynchronous event from the device is useless. The communication with the device can be done only through two bulk endpoints plus the control endpoint. The host driver is easier and the bandwidth can be improved considerably to reach the maximum bandwith available through the USB link (~1 MByte/sec).

atm6124.sys Driver

Only the Abstract Control Model Interface is active and communication is done through the two bulk endpoints.

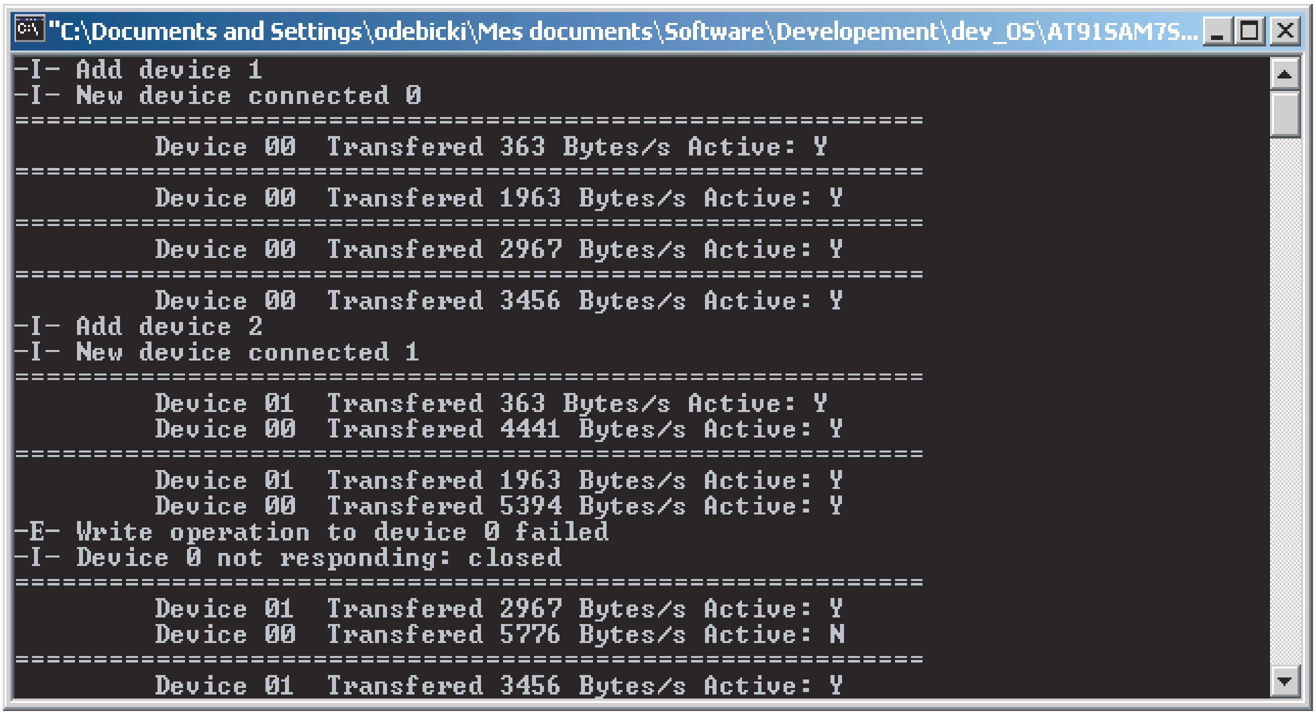
This driver has been compiled with Windows Driver Development Kit. More information concerning Driver Development Kits can be obtained from the the Microsoft web site: http://www.microsoft.com/whdc/ddk/winddk.mspx

When compiling user applications that interface with the driver, some libraries included in the Windows driver development Kit are required (ex. SetupDiEnumDeviceInterfaces() functions, etc.).

PC BasicUSB Application Demo

The goal of this demo it to automatically detect any new boards connected and to start a new task for each board that generates a USB transfer.

Figure 6. Application Snapshot


"C:\Documents and Settings\odebicki\Mes documents\Software\Developement\dev_OS\AT91SAM7S...

```
-I- Add device 1
-I- New device connected 0
==========================================================
         Device 00  Transfered 363 Bytes/s Active: Y
==========================================================
         Device 00  Transfered 1963 Bytes/s Active: Y
==========================================================
         Device 00  Transfered 2967 Bytes/s Active: Y
==========================================================
         Device 00  Transfered 3456 Bytes/s Active: Y
-I- Add device 2
-I- New device connected 1
==========================================================
         Device 01  Transfered 363 Bytes/s Active: Y
         Device 00  Transfered 4441 Bytes/s Active: Y
==========================================================
         Device 01  Transfered 1963 Bytes/s Active: Y
         Device 00  Transfered 5394 Bytes/s Active: Y
-E- Write operation to device 0 failed
-I- Device 0 not responding: closed
==========================================================
         Device 01  Transfered 2967 Bytes/s Active: Y
         Device 00  Transfered 5776 Bytes/s Active: N
==========================================================
         Device 01  Transfered 3456 Bytes/s Active: Y
```

This is a multi-thread application developped under Microsoft Visual C++® V6.0. The main thread scans all new devices connected, calling the GetUsbDeviceListName() function. This function belongs to the USBlibrary.dll. Each time a new device is detected, a new child thread (DeviceThread()) is started. This thread sends a random buffer to the device and reads it back to compare what has been sent and received. If the read or write funtion abort, the thread is stopped. If no connection is active, the application stops.

To ease the demonstration, the random buffer generated by the GenerateBuffer() function is supposed to be non-null and data must not be a multiple of a data payload.

Installation

To use this example, copy atmusb6124.inf in the Windows C:\WINNT\inf directory and atmusb6124.sys in the Windows C:\WINNT\system32\drivers directory. The first time the USB device is connected, the host PC asks for an driver installation file (.inf file). The atmusb6124.sys driver is associated with this device. The PC application is then ready to communicate. This is the same driver used by the SAMBA application.

Interface to USBlibrary.dll

To ease PC application development, a communication DLL is provided. The final application just has to be linked with this DLL and the Windows DDK build utility is not required.

The interface of the USBLibrary.dll is defined in the USBLibrary.h file:

```
int GetUsbDeviceListName(char** deviceList); // Get list of connected
devices
class  CFCPipeUSB {
    HANDLE m_hPipeIn;     // Handel of the input file
    HANDLE m_hPipeOut;  // Handel of the output file
public :
    CFCPipeUSB();
    short Open(char *sDeviceName); // Open the communication
    short Close();                                 // Close the
communication
    short ReadPipe(LPVOID pBuffer, ULONG ulBufferSize); // Read a
buffer of data
    short WritePipe(LPVOID pBuffer, ULONG ulBufferSize); // Write a
buffer of data
};
```

The library *USBLibrairy.dll* provides one function *GetUsbDeviceListName* and one class *CFCPipeUSB.*

The function *GetUsbDeviceListName* returns the number of connected devices and completes the list of device names in the *deviceList* buffer. The *deviceList* buffer is allocated by the DLL. The structure returned is the following:

```
(char**) deviceList
  -> (char*) Name of Device 1 ‘\0‘
  -> (char*) Name of Device 2 ‘\0’
  -> (char*) Name of Device 3 ‘\0’
  -> (char*) ...
```

It is then possible to associate a *CFCPipeUSB* instance with each connected device. Through each pipe it is possible to perform read and write operations.

If pipe operations are aborted because the device does not answer, a timeout occurs and error code is returned.

Document Details

Title	Basic USB Application
Literature Number	6123

Revision History

Version A	**Publication Date:** 16-Sep-04

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

Disclaimer: Atmel Corporation makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in Atmel's Terms and Conditions located on the Company's web site. The Company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of Atmel are granted by the Company in connection with the sale of Atmel products, expressly or by implication. Atmel's products are not authorized for use as critical components in life support devices or systems.

Printed on recycled paper.