July 2007 | Clifton Labs Inc. |


Revision 0.2

# USB Mass Storage & CDC Interfaces

## Abstract

It is convenient to be able to have a mass storage device functioning, and also have two way communication to and from the same device over the same physical connection. As the mass storage device takes 2 endpoints (in addition to the required control endpoint 0) and it is possible to have uart over usb through the cdc interface (which requires 3 endpoints). We use a standard development kit and existing code examples to show the feasibility of creating such a device. The following tutorial describes the integration of the Atmel Mass Storage and CDC examples as well as the driver modification which must take place in order to have a device which successfully functions as a composite device simultaneously allowing a mass storage device and com communication.

## Contents



## Pre-requisites

**Hardware**
- AT90USB device with atleast 6 USB endpoints
- Hardware to program device

**Software**
- Atmel Mass Storage Code Example
- Atmel CDC Example
- AVR studio with avr-gcc compiler*

**Operating System**
- Microsoft Windows 2000 or Windows XP
- Standard Mass Storage Driver
- Full Administrator Access*

Mass Storage Demo CDC Demo (Insure both work separately on your device)
*Helpful but can be done without depending on circumstances.

# Environment Setup

After downloading both the Mass Storage and CDC demos from Atmel insure that both work on you device as expected (the CDC demo will come with the needed driver and assuming you have Win 2K or XP you'll have a Mass Storage driver as well).

Create a new project by copying the Atmel Mass Storage Project to a new location.

Your must then modify the *.aps project file located in the gcc directory.

You must replace all instances of "C:\Atmel\at90usb128-…" to your current path location.

Now using AVR studio open the aps file and compile it. It should produce no errors.

Now examine both demo folders (CDC and Mass Storage) and incorporate all files that are unique to the CDC demo into the Mass Storage demo. (A file listing of the merged demos can be found at the end of this tutorial.

*"There's always another way"*

## Adding the CDC task to the Makefile and scheduler

Edit the **makefile** (located in the gcc directory) to include

**On the OBJECTS = line**

Uart_usb_lib.o uart_lib.o and cdc_task.o to

**In the ##Build section**

```
uart_drv.o: ../../../../common/lib_mcu/uart_drv.c
   $(CC) $(INCLUDES) $(CFLAGS) –c $<
uart_lib.o: ../../../../common/lib_mcu/uart_lib.c
   $(CC) $(INCLUDES) $(CFLAGS) –c $<
uart_usb_lib.o: ../uart_usb_lib.c
   $(CC) $(INCLUDES) $(CFLAGS) –c $<
cdc_task.o:  ../cdc_task.c
   $(CC) $(INCLUDES) $(CFLAGS) –c $<
```

Now the easy part, open **conf_scheduler.h** and add the following lines:

```
#define Scheduler_task_3_init cdc_task_init
#define Scheduler_task_3 cdc_task
```

Make sure everything compiles correctly

# The USB Device, Interface, and Endpoint configuration.

Before we integrate the cdc task into the usb specific requests, we should first define and understand the configuration of the device (at its many levels). Below are depictions of the two **standard** Mass Storage and CDC configurations.

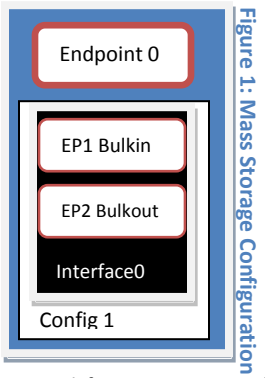



Figure 1: Mass Storage Configuration

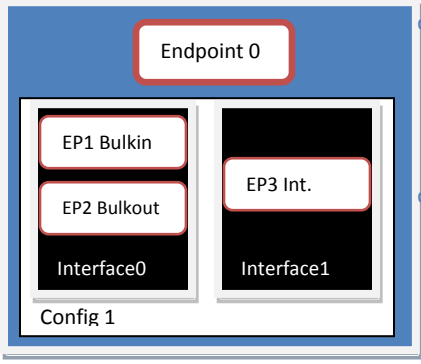



Figure 2: CDC Configuration

First and foremost, Microsoft cannot access multiple configurations simultaneously. Therefore we must keep 1 configuration. The first and seemly straightforward solution for integration is to have a configuration with three interfaces. Unfortunately, unless you want to write custom drivers it doesn't work (several claims have been made online that you can use a feature in XP though no substantiation of the claims has been seen). The solution is depicted in figure 3 (The code for defining this is found on the following pages).

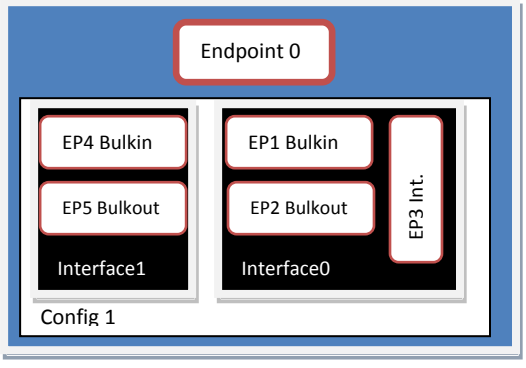



Figure 3: MS and CDC Interfaces

# USB Configuration (code)

(changes | ***additions***)

config.h

```
#define BAUDRATE 38400
#define UART_U2
#define NB_MS_BEFORE_FLUSH 50
#define REPEATE_KEY_PRESSED 100
```

conf_usb.h

```
#define NB_ENDPOINTS   6
#define EP_MS_IN    4   //Mass Storage In
#define EP_MS_OUT 5  //Mass Storage Out
#define TX_EP          0x01 //CDC IN
#define RX_EP          0x02   //CDC OUT
#define INT_EP         0x03  //Interrupt
#define Usb_suspend_action suspend_action();
Extern void suspend  action(void);
```

usb_descriptors.h (configuration parameters)

```
// USB Device descriptor
#define USB_SPECIFICATION     0x0200
#define DEVICE_CLASS          0x00
#define DEVICE_SUB_CLASS      0
#define DEVICE_PROTOCOL       0
#define EP_CONTROL_LENGTH 64
#define VENDOR_ID      0xXXXX // CHANGE
#define PRODUCT_ID   0xYYYY // CHANGE
#define RELEASE_NUMBER       0x1000
#define MAN_INDEX           0x01
#define PROD_INDEX          0x02
#define SN_INDEX           0x03
#define NB_CONFIGURATION     1

//  CONFIGURATION
#define NB_INTERFACE      2
#define CONF_NB          1
#define CONF_INDEX       0
#define CONF_ATTRIBUTES
USB_CONFIG_BUSPOWERED
#define MAX_POWER         50

// START OF CDC INTERFACE //
//Interface 1 descriptor
#define INTERFACE0_NB       0
#define ALTERNATE0          0
#define NB_ENDPOINT0        3
#define INTERFACE0_CLASS     0x02
#define INTERFACE0_SUB_CLASS 0x02
#define INTERFACE0_PROTOCOL  0x00
#define INTERFACE0_INDEX     0

//USB ENDPOINT 3 descriptor
//INTERUPT IN

#define TX_EP_SIZE       0x20
#define ENDPOINT_NB_3    (0x80  | INT_EP)
#define EP_ATTRIBUTES_3   0x03
#define EP_SIZE_3        0x08
#define EP_INTERVAL_3     0x10
```

```
//USB ENDPOINT 1 descriptor
//BULK OUT
#define ENDPOINT_NB_1     RX_EP
#define EP_ATTRIBUTES_1   0x02
#define EP_SIZE_1         0x40
#define EP_INTERVAL_1     0x00

//USB ENDPOINT 2 descriptor
//BULK IN
#define ENDPOINT_NB_2     (0x80  | TX_EP)
#define EP_ATTRIBUTES_2   0x02
#define EP_SIZE_2         0x40
#define EP_INTERVAL_2     0x00

// END OF  CDC INTERFACE //
// START OF  USB MS INTERFACE
// USB Interface descriptor
#define INTERFACE1_NB    1
#define ALTERNATE1       0
#define NB_ENDPOINT1     2
#define INTERFACE1_CLASS     0x08

// Mass Storage Class
#define INTERFACE1_SUB_CLASS 0x06
#define INTERFACE1_PROTOCOL  0x50
#define INTERFACE1_INDEX     0

 // USB Endpoint 4 descriptor FS
#define ENDPOINT_NB_4      EP_MS_OUT
#define EP_ATTRIBUTES_4    0x02
 #define EP_IN_LENGTH       64
#define EP_SIZE_4          EP_IN_LENGTH
#define EP_INTERVAL_4      0x00

 // USB Endpoint 5 descriptor FS
#define ENDPOINT_NB_5      (EP_MS_IN | 0x80)
#define EP_ATTRIBUTES_5    0x02
#define EP_IN_LENGTH       64
#define EP_SIZE_5          EP_IN_LENGTH
#define EP_INTERVAL_5      0x00
```



Clifton Labs, Inc.

7450 Montgomery Rd, Suite 300
Cincinnati, Ohio 45236
Mike Borowczak
borowcm [at] cliftonlabs [dot] com

*Computer Engineering Services & Solutions*

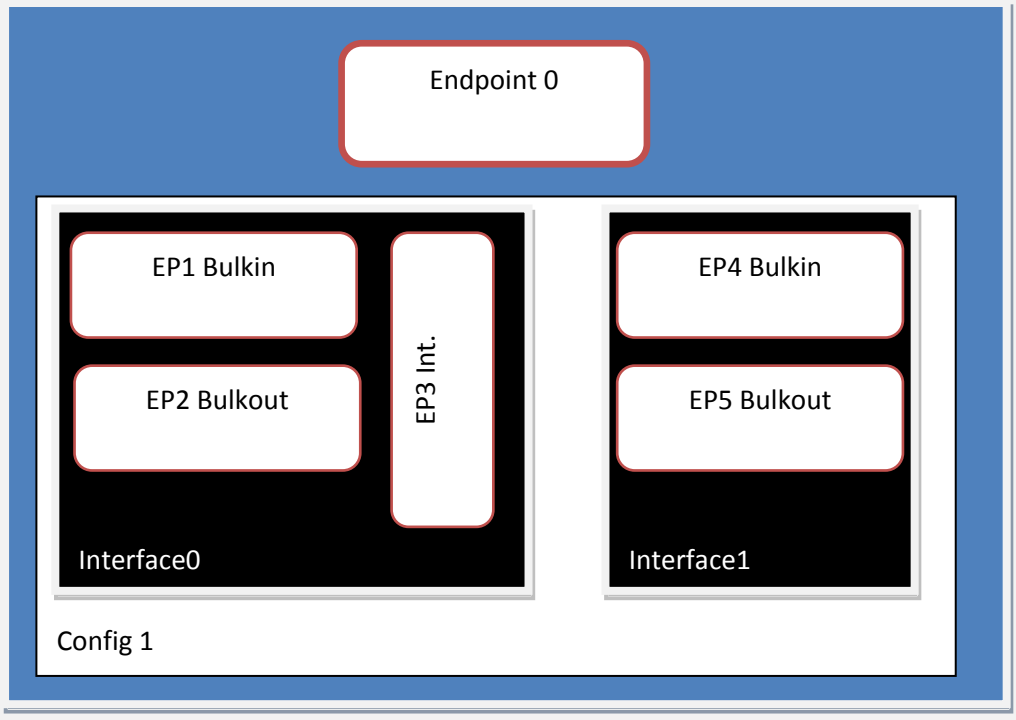
Online at www.cliftonlabs.com

```
typedef struct
{
  S_usb_configuration_descriptor cfg;
  S_usb_interface_descriptor    ifc0;
  U8 CS_INTERFACE[19];
  S_usb_endpoint_descriptor     ep3;
  S_usb_endpoint_descriptor     ep1;
  S_usb_endpoint_descriptor     ep2;
  S_usb_interface_descriptor    ifc1;
  S_usb_endpoint_descriptor     ep4;
  S_usb_endpoint_descriptor     ep5;
} S_usb_user_configuration_descriptor;
```

# USB Configuration (code) cont.

**Usb_descriptors.c <exerpt>**

```
// usb_user_configuration_descriptor FS
code S_usb_user_configuration_descriptor
usb_conf_desc = {
 { sizeof(S_usb_configuration_descriptor)
 , CONFIGURATION_DESCRIPTOR
 ,
Usb_write_word_enum_struc(sizeof(usb_conf
_desc))
 , NB_INTERFACE
 , CONF_NB
 , CONF_INDEX
 , CONF_ATTRIBUTES
 , MAX_POWER
 }
 ,
  { sizeof(S_usb_interface_descriptor)
 , INTERFACE_DESCRIPTOR
 , INTERFACE0_NB
 , ALTERNATE0
 , NB_ENDPOINT0
 , INTERFACE0_CLASS
 , INTERFACE0_SUB_CLASS
 , INTERFACE0_PROTOCOL
 , INTERFACE0_INDEX
 }
 ,
{ 0x05, 0x24, 0x00, 0x10, 0x01, 0x05, 0x24,
0x01, 0x03, 0x01, 0x04, 0x24, 0x02, 0x06,0x05,
0x24, 0x06, 0x00, 0x01 }
 ,
 { sizeof(S_usb_endpoint_descriptor)
 , ENDPOINT_DESCRIPTOR
 , ENDPOINT_NB_3
 , EP_ATTRIBUTES_3
 , Usb_write_word_enum_struc(EP_SIZE_3)
 , EP_INTERVAL_3
 }
 ,
 { sizeof(S_usb_endpoint_descriptor)
 , ENDPOINT_DESCRIPTOR
 , ENDPOINT_NB_1
 , EP_ATTRIBUTES_1
 , Usb_write_word_enum_struc(EP_SIZE_1)
 , EP_INTERVAL_1
 }
 ,
 { sizeof(S_usb_endpoint_descriptor)
 , ENDPOINT_DESCRIPTOR
 , ENDPOINT_NB_2
 , EP_ATTRIBUTES_2
 , Usb_write_word_enum_struc(EP_SIZE_2)
 , EP_INTERVAL_2
 }
  ,
 { sizeof(S_usb_interface_descriptor)
 , INTERFACE_DESCRIPTOR
 , INTERFACE1_NB
 , ALTERNATE1
 , NB_ENDPOINT1
 , INTERFACE1_CLASS
 , INTERFACE1_SUB_CLASS
 , INTERFACE1_PROTOCOL
 , INTERFACE1_INDEX
 }
,
 { sizeof(S_usb_endpoint_descriptor)
 , ENDPOINT_DESCRIPTOR
 , ENDPOINT_NB_4
 , EP_ATTRIBUTES_4
 , Usb_write_word_enum_struc(EP_SIZE_4)
 , EP_INTERVAL_4
 }
 ,
 { sizeof(S_usb_endpoint_descriptor)
 , ENDPOINT_DESCRIPTOR
 , ENDPOINT_NB_5
 , EP_ATTRIBUTES_5
 , Usb_write_word_enum_struc(EP_SIZE_5)
 , EP_INTERVAL_5
 }
};
```



# USB Configuration (code) cont.

## Driver Setup:

Step 1: Follow directions in CDC example code to install the custom driver normally (this should have been done earlier when you tested the example prior to messing around with all of this :)

Step 2: Edit the .inf file and add one line:

%ATMEL_CDC%=Reader, USB\VID_XXXX&PID_YYYY&MI_00

Step 3: Find and backup your usbstore.inf file now add the following line under the Microsoft section

%GenericBulkOnly.DeviceDesc%=USBSTOR_BULK, USB\VID_XXXX&PID_YYYY&MI_01

XXXX and YYYY are the values specifies in your usb_descriptor.h file

Step 4: Backup your registry, use regedt32 goto:
HKEY_LOCAL_MACHINE
→System
→ControlSet00X (all of them)
+→Enum
++→USB (delete all references to your device)
++→USBSTOR (delete all references to your device)

[Changing the security may be needed but is easy to do]

Step 5: Program and Start you device as normal, you should be asked to install software (always do it manually and it should work fine (i.e. don't let the online software tell you which driver to use).

**WARNING:**
Before disconnecting your device YOU MUST uninstall the AT90USBxxx CDC USB to UART MGM (COM x) otherwise you'll have to manually delete registry entries.

Usb_specific_request.c

```
#include "lib_mcu\uart\uart_lib.h"
extern S_line_coding   line_coding;

switch(request)
  {
...
  case GET_LINE_CODING:
        cdc_get_line_coding();
        return TRUE;
        break;
  case SET_LINE_CODING:
        cdc_set_line_coding();
        return TRUE;
        break;
   case SET_CONTROL_LINE_STATE:
        cdc_set_control_line_state();
        return TRUE;
        break;
     default:
       return FALSE;
       break;
  }

void usb_user_endpoint_init(U8 conf_nb)
{
 usb_configure_endpoint(EP_MS_IN, \
                  TYPE_BULK,    \
                  DIRECTION_IN, \
                  ep_size,      \
                  TWO_BANKS,    \
                  NYET_ENABLED);
 usb_configure_endpoint(EP_MS_OUT, \
                  TYPE_BULK,    \
                  DIRECTION_OUT, \
                  ep_size,      \
                  TWO_BANKS,    \
                  NYET_ENABLED);
  usb_configure_endpoint(INT_EP,   \
                  TYPE_INTERRUPT, \
                  DIRECTION_IN, \
                  SIZE_32,      \
                  ONE_BANK,     \
                  NYET_ENABLED);
 usb_configure_endpoint(TX_EP, \
                  TYPE_BULK, \
                  DIRECTION_IN, \
                  SIZE_32,    \
                  ONE_BANK,    \
                  NYET_ENABLED);
 usb_configure_endpoint(RX_EP,  \
                  TYPE_BULK,    \
                  DIRECTION_OUT, \
                  SIZE_32,      \
                  TWO_BANKS,    \
                  NYET_ENABLED);

 Usb_reset_endpoint(EP_MS_IN);
 Usb_reset_endpoint(EP_MS_OUT);
 Usb_reset_endpoint(INT_EP);
 Usb_reset_endpoint(TX_EP);
 Usb_reset_endpoint(RX_EP);

}
```

**Usb_specific_request.c (cont)**

```
void cdc_get_line_coding(void)
{
  Usb_ack_receive_setup();
Usb_write_byte(LSB0(line_coding.dwDTERate));
Usb_write_byte(LSB1(line_coding.dwDTERate));
Usb_write_byte(LSB2(line_coding.dwDTERate));
Usb_write_byte(LSB3(line_coding.dwDTERate));
Usb_write_byte(line_coding.bCharFormat);
Usb_write_byte(line_coding.bParityType);
Usb_write_byte(line_coding.bDataBits);
Usb_send_control_in();
while(!(Is_usb_read_control_enabled()));
//Usb_clear_tx_complete();
 while(!Is_usb_receive_out());
  Usb_ack_receive_out();
}

void cdc_set_line_coding (void)
{
   Usb_ack_receive_setup();
   while (!(Is_usb_receive_out()));
LSB0(line_coding.dwDTERate) =
Usb_read_byte();
LSB1(line_coding.dwDTERate) =
Usb_read_byte();
LSB2(line_coding.dwDTERate) =
Usb_read_byte();
LSB3(line_coding.dwDTERate) =
Usb_read_byte();
line_coding.bCharFormat = Usb_read_byte();
line_coding.bParityType = Usb_read_byte();
line_coding.bDataBits = Usb_read_byte();
Usb_ack_receive_out();
Usb_send_control_in();
 while(!(Is_usb_read_control_enabled()));
#ifdef UART_U2

Uart_set_baudrate((line_coding.dwDTERate)/2);
#else
  Uart_set_baudrate(line_coding.dwDTERate);
#endif
}


void cdc_set_control_line_state (void)
{
```

Usb_specific_request.h

```
#define GET_LINE_CODING         0x21
#define SET_LINE_CODING         0x20
#define SET_CONTROL_LINE_STATE 0x22

void  cdc_get_line_coding();
void  cdc_set_line_coding();
void  cdc_set_control_line_state (void);

typedef struct
{
        U32 dwDTERate;
        U8 bCharFormat;
        U8 bParityType;
        U8 bDataBits;
}S_line_coding;
```