USB 1.1 VCOM Device Application Note V1.0

Publication Release Date: Aug. 2008

Support Chips:

NUC710A

NUC745A

Support Platforms:

All

The information in this document is subject to change without notice.

The Nuvoton Technology Corp. shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

Table of Contents

1. Introduction 4

1.1. USBD 1.1 Introduction 4

2. Solution 5

2.1. VCOM Device 5

2.2. Get Descriptor Request 5

Standard Device Descriptor 5
Standard Configuration Descriptor 5
Standard String Descriptor 6

2.3. Configure Endpoint 6

Endpoint A – Bulk IN 6
Endpoint B – Bulk OUT 6

2.4. Trigger Endpoint DMA 6

Endpoint A – Bulk IN 6
Endpoint B – Bulk OUT 7

2.5. Parse USB Standard Request 7

2.6. Interrupt Service Routine 7

3. Example 8

3.1. Configure Endpoint 8

3.2. Parse USB Standard Request 10

3.3. Trigger Endpoint DMA 11

3.4. Interrupt Service Routine 13

3.5. Main Routine 15

4. Revision History 17

1. Introduction

1.1. USBD 1.1 Introduction

NUC710A/NUC745A has a USB 1.1 Device Controller built-in. The device controller can be configured to be a mass storage device or a virtual COM port, etc. This controller consists of four endpoints, designated EP0, EPA, EPB and EPC.

EP0 : The default endpoint uses control transfer (In / Out) to handle configuration and control functions required by the USB specification. Maximum packet size is 16 bytes.

EPA : Designed as a general endpoint. This endpoint could be programmed to be an Interrupt IN endpoint or an Isochronous IN endpoint or a Bulk IN endpoint or Bulk OUT endpoint.

EPB : Designed as a general endpoint. This endpoint could be programmed to be an Interrupt IN endpoint or an Isochronous IN endpoint or a Bulk IN endpoint or Bulk OUT endpoint.

EPC : Designed as a general endpoint. This endpoint could be programmed to be an Interrupt IN endpoint or an Isochronous IN endpoint or a Bulk IN endpoint or Bulk OUT endpoint.

The USB controller has built-in hard-wired state machine to automatically respond to USB standard device request. It also supports to detect the class and vendor requests. For GetDescriptor request and Class or Vendor command, the firmware will control these procedures.

This document will help user to configure the NUCP710A/NUC745A USB to be a virtual COM.

2. Solution

2.1. VCOM Device

Use VCOM device, the host PC should install a corresponding Windows device driver. The driver can be found at Linux BSP.

The PC driver will send a vendor command (0x12) to device to tell the bulk IN length first. The device should prepare the bulk IN data after get the transfer length. Use this flow, device will easy to do bulk IN transfer.

After finish transfer, the PC driver will send a vendor command (0x13) to device to reset the device FIFO and then close the transfer port.

2.2. Get Descriptor Request

Before send the descriptor to host, user should enable the USB string and interface register first. In virtual COM sample, user should set the 0x30 to **REG_USB_IFSTR** register to enable string descriptor-1 and string descriptor-2 control.

Standard Device Descriptor

The standard device descriptor should be set as
DevDesc[5] = {0x01100112, 0x10000000, 0x70210416, 0x02010100, 0x00000100};
bcdUSB = 0x0110;
DeviceClass = 0x00;
DeviceSubClass = 0x00;
DeviceProtocol = 0x00;
MaxPacketSize0 = 0x10 (16 bytes);
NumConfigurations = 0x01;

Standard Configuration Descriptor

The standard configuration descriptor should be set as
CfgDesc[8] = {0x00200209, 0xC0000101, 0x00040932, 0x00000200 , 0x05070000, 0x00400281, 0x02050701, 0x01004002};
NumInsterfaces = 0x01;
Attributes = 0xC0 (self-powered);
NumEndpoints = 0x02;

InterfaceClass = 0x00;
InterfaceSubClass = 0x00;
InterfaceProtocol = 0x00;
Endpoint MaxPacketSize = 0x40;

Standard String Descriptor

The string descriptor should be set as
StrDesc_lang[1] = { 0x04090304};
StrDesc_Vendor[6] = { 0x00550316, 0x00420053, 0x00440020, 0x00760065, 0x00630069 ,0x00000065};
StrDesc_Dev[12] = { 0x0057032E, 0x00390039, 0x00300037, 0x00200032, 0x00530055, 0x00200042, 0x00690056, 0x00740072, 0x00610075, 0x0020006C, 0x004F0043, 0x0000004D};
Vendor string = “USB Device”;
Device string = “W99702 USB Virtual COM”;

2.3. Configure Endpoint

Endpoint A – Bulk IN

Set 0x30400011 to **REG_USB_EPA_INFO**;
Set 0x00000001 to **REG_USB_EPA_CTL** to enable endpoint A.
Set 0x00000008 to **REG_USB_EPA_IE** to enable endpoint A DMA interrupt.

Endpoint B – Bulk OUT

Set 0x20400012 to **REG_USB_EPB_INFO**;
Set 0x00000001 to **REG_USB_EPB_CTL** to enable endpoint A.
Set 0x00000008 to **REG_USB_EPB_IE** to enable endpoint A DMA interrupt.

2.4. Trigger Endpoint DMA

Endpoint A – Bulk IN

If transfer size is zero, set 0x40 to **REG_USB_EPA_CTL** to send zero packet. Then wait for the EPA DMA interrupt complete.
If transfer size is not zero, set 0x04 to **REG_USB_EPA_CTL** to send the packet. Then wait for the EPA DMA interrupt complete.

Endpoint B – Bulk OUT

If transfer size is zero, set 0x40 to **REG_USB_EPB_CTL** to send zero packet. Then wait for the EPB DMA interrupt complete.
If transfer size is not zero, set 0x04 to **REG_USB_EPB_CTL** to send the packet. Then wait for the EPB DMA interrupt complete.

2.5. Parse USB Standard Request

Get the data from **REG_USB_ODATA0**, **REG_USB_ODATA1**, **REG_USB_ODATA2** and **REG_USB_ODATA3**.
Request Type, Request, and Value are in **REG_USB_ODATA0** register.
Index and length are in **REG_USB_ODATA1** register.

2.6. Interrupt Service Routine

Set 0x47f7 to **REG_USB_IE** register when initialing the device controller. It will enable the “Reset end”, “Class or Vendor command data in”, “Class or Vendor command data out”, “USB Vendor command”, “USB Class command”, “USB Get_String_Descriptor Command”, “USB Get_Configuration_Descriptor Command”, “USB Get_Device_Descriptor Command”, “USB Resume Detect”, “USB Suspend Detect”, and “USB Reset Command Detect” interrupts.

When interrupt occurred, check the **REG_USB_IS**, **REG_EPA_IS**, and **REG_EPB_IS** registers to service it. After service it, write the **REG_USB_IC**, **REG_EPA_IC**, and **REG_EPB_IC** registers to clear interrupt.

3. Example

3.1. Configure Endpoint

```
USB_Setup_Endpoint(EP_A,1,Ep_In,Ep_Bulk,1,0,64);
USB_Setup_Endpoint(EP_B,2,Ep_Out,Ep_Bulk,1,0,64);

UINT8 USB_Setup_Endpoint(UINT8 epname,UINT8 epnum,UINT8 epdir,UINT8 eptype,
                    UINT8 epcon,UINT8 epinf,UINT16 epmaxps )
{
   UINT32 tmp;

   if (epname == EP_A)
   {
    if ((epdir==Ep_In)&&(eptype==Ep_Bulk)&&(epcon==1))
        outpw(REG_USB_EPA_INFO,0x30000010);
    else if ((epdir==Ep_In)&&(eptype==Ep_Int)&&(epcon==1))
        outpw(REG_USB_EPA_INFO,0x50000010);
    else if ((epdir==Ep_Out)&&(eptype==Ep_Bulk)&&(epcon==1))
        outpw(REG_USB_EPA_INFO,0x20000010);
    else
         return 0;
    tmp = epinf;
    tmp = tmp << 8;
    outpw(REG_USB_EPA_INFO,(inpw(REG_USB_EPA_INFO)|tmp));
    tmp = epmaxps;
    tmp = tmp << 16;
    outpw(REG_USB_EPA_INFO,(inpw(REG_USB_EPA_INFO)|tmp));
    outpw(REG_USB_EPA_INFO,(inpw(REG_USB_EPA_INFO)|(epnum&0x0f)));
       USB_EP_Inf_Array[EP_A].EP_Num = epnum;
       USB_EP_Inf_Array[EP_A].EP_Dir = epdir;
       USB_EP_Inf_Array[EP_A].EP_Type = eptype;
```

```
    outpw(REG_USB_EPA_CTL,0x00000001);
   }
   else if (epname == EP_B)
   {
    if ((epdir==Ep_In)&&(eptype==Ep_Bulk)&&(epcon==1))
       outpw(REG_USB_EPB_INFO,0x30000010);
    else if ((epdir==Ep_In)&&(eptype==Ep_Int)&&(epcon==1))
       outpw(REG_USB_EPB_INFO,0x50000010);
    else if ((epdir==Ep_Out)&&(eptype==Ep_Bulk)&&(epcon==1))
       outpw(REG_USB_EPB_INFO,0x20000010);
    else
        return 0;
    tmp = epinf;
    tmp = tmp << 8;
    outpw(REG_USB_EPB_INFO,(inpw(REG_USB_EPB_INFO)|tmp));
    tmp = epmaxps;
    tmp = tmp << 16;
    outpw(REG_USB_EPB_INFO,(inpw(REG_USB_EPB_INFO)|tmp));
    outpw(REG_USB_EPB_INFO,(inpw(REG_USB_EPB_INFO)|(epnum&0x0f)));
      USB_EP_Inf_Array[EP_B].EP_Num = epnum;
      USB_EP_Inf_Array[EP_B].EP_Dir = epdir;
      USB_EP_Inf_Array[EP_B].EP_Type = eptype;
    outpw(REG_USB_EPB_CTL,0x00000001);
   }
   else if (epname == EP_C)
   {
    if ((epdir==Ep_In)&&(eptype==Ep_Bulk)&&(epcon==1))
       outpw(REG_USB_EPC_INFO,0x30000010);
    else if ((epdir==Ep_In)&&(eptype==Ep_Int)&&(epcon==1))
       outpw(REG_USB_EPC_INFO,0x50000010);
    else if ((epdir==Ep_Out)&&(eptype==Ep_Bulk)&&(epcon==1))
       outpw(REG_USB_EPC_INFO,0x20000010);
    else
        return 0;
    tmp = epinf;
    tmp = tmp << 8;
    outpw(REG_USB_EPC_INFO,(inpw(REG_USB_EPC_INFO)|tmp));
    tmp = epmaxps;
```

```
    tmp = tmp << 16;
    outpw(REG_USB_EPC_INFO,(inpw(REG_USB_EPC_INFO)|tmp));
    outpw(REG_USB_EPC_INFO,(inpw(REG_USB_EPC_INFO)|(epnum&0x0f)));
       USB_EP_Inf_Array[EP_C].EP_Num = epnum;
       USB_EP_Inf_Array[EP_C].EP_Dir = epdir;
       USB_EP_Inf_Array[EP_C].EP_Type = eptype;
    outpw(REG_USB_EPC_CTL,0x00000001);
   }
   else
       return 0;
   return 1;
}
```

3.2. Parse USB Standard Request

```
typedef struct
{UINT8 Req_Type;
 UINT8 Req;
 UINT16 Value;
 UINT16 Index;
 UINT16 Length;
}USB_Vender_Cmd_Format_T;
USB_Vender_Cmd_Format_T volatile USB_CtlOut_Format;

void USB_Cmd_Parsing(void)
{
    USB_CtlOut_Format.Req_Type = inpb(USB_BA+0x18);//A_U_CTOD0;
    USB_CtlOut_Format.Req = inpb(USB_BA+0x19);//A_U_CTOD1;

    USB_CtlOut_Format.Value = inphw(USB_BA+0x1a);//2,3

    USB_CtlOut_Format.Index = inphw(USB_BA+0x1c);//4,5

    USB_CtlOut_Format.Length = inphw(USB_BA+0x1e);//6,7

    if ((USB_CtlOut_Format.Req == 0x04)||(USB_CtlOut_Format.Req == 0x05))
```

```
        USB_MEM_START =
(USB_CtlOut_Format.Value<<16)+USB_CtlOut_Format.Index;


    if (GET_STR_Flag == 1)
        USB_CtlOut_Format.Index = 0;
}
```

3.3. Trigger Endpoint DMA

```
void SDRAM_USB_Transfer(UINT8 epname,UINT32 DRAM_Addr ,UINT32 Tran_Size)
{
   if (epname == EP_A)
   {
      if(Tran_Size==0)
      {
         DMA_CInt_A_Flag=0;
         outpw(REG_USB_EPA_CTL,inpw(REG_USB_EPA_CTL)|0x00000040);

while((USBModeFlag)&&((inpw(REG_USB_EPA_CTL)&0x00000040)==0x00000040));
      }
      else
      {
         outpw(REG_USB_EPA_ADDR,DRAM_Addr);
         outpw(REG_USB_EPA_LENTH,Tran_Size);
         DMA_CInt_A_Flag=0;
         outpw(REG_USB_EPA_CTL,inpw(REG_USB_EPA_CTL)|0x00000004);

while((USBModeFlag)&&((inpw(REG_USB_EPA_CTL)&0x00000004)==0x00000004));
      }


   }
   else if (epname == EP_B)
   {
      if(Tran_Size==0)
      {
         DMA_CInt_B_Flag=0;
```

```
            outpw(REG_USB_EPB_CTL,inpw(REG_USB_EPB_CTL)|0x00000040);

while((USBModeFlag)&&((inpw(REG_USB_EPB_CTL)&0x00000040)==0x00000040));
        }
        else
        {
            outpw(REG_USB_EPB_ADDR,DRAM_Addr);
            outpw(REG_USB_EPB_LENTH,Tran_Size);
            DMA_CInt_B_Flag=0;
            outpw(REG_USB_EPB_CTL,inpw(REG_USB_EPB_CTL)|0x00000004);

while((USBModeFlag)&&((inpw(REG_USB_EPB_CTL)&0x00000004)==0x00000004));
        }
    }
    else if (epname == EP_C)
    {
        if(Tran_Size==0)
        {
            DMA_CInt_C_Flag=0;
            outpw(REG_USB_EPC_CTL,inpw(REG_USB_EPC_CTL)|0x00000040);

while((USBModeFlag)&&((inpw(REG_USB_EPC_CTL)&0x00000040)==0x00000040));
        }
        else
        {
            outpw(REG_USB_EPC_ADDR,DRAM_Addr);
            outpw(REG_USB_EPC_LENTH,Tran_Size);
            DMA_CInt_C_Flag=0;
            outpw(REG_USB_EPC_CTL,inpw(REG_USB_EPC_CTL)|0x00000004);

while((USBModeFlag)&&((inpw(REG_USB_EPC_CTL)&0x00000004)==0x00000004));
        }
    }
}
```

3.4. Interrupt Service Routine

```
void USB_Int_ISR(void)
{
    UINT32 id;

    id = inpw(REG_USB_IS);
    if (id&0x00000001)
        USB_ISR_Reset_Start();
    if (id&0x00000002)
        USB_ISR_Suspend();
    if (id&0x00000004)
        USB_ISR_Resume();
    if (id&0x00000008)
        USB_ISR_Error();
     … … … …
    if (id&0x00004000)
        USB_ISR_Reset_End();

    id = inpw(REG_USB_EPA_IS);
    if (id&0x00000001)
        USB_ISR_EpA_Stall();
    if (id&0x00000002)
        USB_ISR_EpA_Token_Input();
    if (id&0x00000008)
        USB_ISR_EpA_DMA_Complete();
    if (id&0x00000010)
        USB_ISR_EpA_Bus_Err();

    id = inpw(REG_USB_EPB_IS);
    if (id&0x00000001)
        USB_ISR_EpB_Stall();
    if (id&0x00000002)
        USB_ISR_EpB_Token_Input();
    if (id&0x00000008)
        USB_ISR_EpB_DMA_Complete();
    if (id&0x00000010)
```

```
        USB_ISR_EpB_Bus_Err();

    id = inpw(REG_USB_EPC_IS);
    if (id&0x00000001)
        USB_ISR_EpC_Stall();
    if (id&0x00000002)
        USB_ISR_EpC_Token_Input();
    if (id&0x00000008)
        USB_ISR_EpC_DMA_Complete();
    if (id&0x00000010)
        USB_ISR_EpC_Bus_Err();
}

void USB_ISR_Reset_Start(void)
{
    outpw(REG_USB_EPA_CTL,inpw(REG_USB_EPA_CTL)|0x00000002);
    outpw(REG_USB_EPB_CTL,inpw(REG_USB_EPB_CTL)|0x00000002);
    outpw(REG_USB_EPA_CTL,inpw(REG_USB_EPA_CTL)&0xfffffffd);
    outpw(REG_USB_EPB_CTL,inpw(REG_USB_EPB_CTL)&0xfffffffd);
    outpw(REG_USB_IC,0x00000001);
}

void USB_ISR_Reset_End(void)
{
    if ((inpw(REG_USB_EPA_CTL)&0x00000004)==0x00000004)
    {
        outpw(REG_USB_EPA_CTL,inpw(REG_USB_EPA_CTL)&0xfffffffb);
        outpw(REG_USB_EPA_CTL,inpw(REG_USB_EPA_CTL)|0x00000004);
    }
    if ((inpw(REG_USB_EPB_CTL)&0x00000004)==0x00000004)
    {
        outpw(REG_USB_EPB_CTL,inpw(REG_USB_EPB_CTL)&0xfffffffb);
        outpw(REG_USB_EPB_CTL,inpw(REG_USB_EPB_CTL)|0x00000004);
    }
    ResetFlag=1;
    outpw(REG_USB_IC,0x00004000);
}
void USB_ISR_Suspend(void)
```

```
{
    if (DEV_RES_Flag == 0)
    {
     DEV_RES_Flag = ~DEV_RES_Flag;
    }
    outpw(REG_USB_IC,0x00000002);
}

void USB_ISR_Resume(void)
{
    outpw(REG_USB_IC,0x00000004);
}

void USB_ISR_Error(void)
{
    outpw(REG_USB_IC,0x00000008);
}

… … … …
void USB_ISR_EpA_DMA_Complete(void)
{
     DMA_CInt_A_Flag=1;
     outpw(REG_USB_EPA_IC,0x00000008);
}

void USB_ISR_EpB_DMA_Complete(void)
{
     outpw(REG_USB_EPB_IC,0x00000008);
}
```

3.5. Main Routine

The following sample code will tell user how to program an “usbprintf()” function.

```
void usbprintf(PINT8 pcStr,...)
{
```

```
    INT8 *argP;

    if (!_usb_bIsUSBInit)
    {
        outpw(REG_SYS_CFG, inpw(REG_SYS_CFG) & 0xffffffbf);
        USB_Init();
        GPIO_Int_Init();
        USB_Int_Init();
        _usb_bIsUSBInit = TRUE;
    }

    usbvaStart(argP, pcStr);      /* point at the end of the format string */
    while (*pcStr)
    {                   /* this works because args are all ints */
        if (*pcStr == '%')
            pcStr = usbFormatItem(pcStr + 1, usbvaArg(argP, INT));
        else
            usbPutChar(*pcStr++);
    }

   if (USBModeFlag)
   {
        while(1)
        {
            SDRAM_USB_Transfer(EP_A, Virtual_Com_Addr_In,
(_usb_uBulkInTail-_usb_uBulkInHead));
            _usb_uBulkInTail = 0;
            _usb_uBulkInHead = 0;
            break;
       }
    }
}
```

4. Revision History

Version	Date	Description
V1.0	Sep. 2008	● Created

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in systems or equipment intended for surgical implantation, atomic energy control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications wherein failure of Nuvoton products could result or lead to a situation wherein personal injury, death or severe property or environmental damage could occur. Nuvoton customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from such improper use or sales.