//

PART 1: USB_vnd.c

//

```
/*****************************************************************************
* File Name: USB_vnd.c
* Version 2.50
*
* Description:
*  USB vendor request handler.
*
* Note:
*
*****************************************************************************
* Copyright 2008-2012, Cypress Semiconductor Corporation.  All rights reserved.
* You may use this file only in accordance with the license, terms, conditions,
* disclaimers, and limitations in the end user license agreement accompanying
```

```
* the software package with which this file was provided.
********************************************************************************/

#include "USB.h"

#if(USB_EXTERN_VND == USB_FALSE)
/***************************************
* Vendor Specific Declarations
***************************************/

/* `#START VENDOR_SPECIFIC_DECLARATIONS` Place your declaration here */
#include <Print.h>
static int k;
uint8  rq, ss;
////////////////////////////////////////////////////////////////////////////////
typedef struct        // vendor request table
```

```
{    byte typ;         // reqest type
     byte req;         // reqest code
     word val;         // value
     word idx;         // index
     word len;         // length
     byte *pRQ;        // data pointer
} VendorTableTyp;
////////////////////////////////////////////////////////////////////////////
VendorTableTyp VendorTable;
static byte rec[16];
static byte buf[2];
static word len, siz;
////////////////////////////////////////////////////////////////////////////
typedef byte (*VendorCallBack)( uint8, VendorTableTyp* );
extern volatile VendorCallBack VendorRequestProc;    // user call back pointer
////////////////////////////////////////////////////////////////////////////
```

```
/* `#END` */


/***************************************
* External References
***************************************/

uint8 USB_InitControlRead(void) ;
uint8 USB_InitControlWrite(void) ;


extern uint8 CYCODE USB_MSOS_CONFIGURATION_DESCR[];

extern volatile T_USB_TD USB_currentTD;
```

```
/*****************************************************************************
* Function Name: USB_HandleVendorRqst
*****************************************************************************
*
* Summary:
*  This routine provide users with a method to implement vendor specifc
*  requests.
*
*  To implement vendor specific requests, add your code in this function to
*  decode and disposition the request.  If the request is handled, your code
*  must set the variable "requestHandled" to TRUE, indicating that the
*  request has been handled.
*
* Parameters:
*  None.
*
```

```
* Return:
*  requestHandled.
*
* Reentrant:
*  No.
*
*******************************************************************************/
uint8 USB_HandleVendorRqst(void)
{
    uint8 requestHandled = USB_FALSE;

    if ((CY_GET_REG8(USB_bmRequestType) & USB_RQST_DIR_MASK) == USB_RQST_DIR_D2H)
    {
        /* Control Read */
        switch (CY_GET_REG8(USB_bRequest))
        {
```

```
            case USB_GET_EXTENDED_CONFIG_DESCRIPTOR:
                #if defined(USB_ENABLE_MSOS_STRING)
                    USB_currentTD.pData = &USB_MSOS_CONFIGURATION_DESCR[0u];
                    USB_currentTD.count = USB_MSOS_CONFIGURATION_DESCR[0u];
                    requestHandled  = USB_InitControlRead();
                #endif /* End USB_ENABLE_MSOS_STRING */
                break;
            default:
                break;
        }
    }

    /* `#START VENDOR_SPECIFIC_CODE` Place your vendor specific request here */
    if ( CY_GET_REG8(USB_bmRequestType) & USB_RQST_TYPE_VND )
    {    // call vendor process //
         if( VendorRequestProc )                // exist user call back, call it
```

```
{    len= CY_GET_REG16(USB_length); // set length
     if( len )
     {    USB_currentTD.count= 0; //len+10;  // size is zero
          USB_currentTD.pData= &buf[0];      // set buffer
          USB_ControlWriteDataStage();       // dummy read from host
          //siz= USB_ReadOutEP( 0, buf, 32 );
     }
     VendorTable.typ= CY_GET_REG8(USB_bmRequestType);// set request type
     VendorTable.req= CY_GET_REG8(USB_bRequest); // set request code
     VendorTable.val= CY_GET_REG16(USB_wValue);  // set value
     VendorTable.idx= CY_GET_REG16(USB_wValue);  // set index
     VendorTable.len= CY_GET_REG16(USB_length);  // set length
     VendorTable.pRQ= &buf[0];
     ss = VendorRequestProc( VendorTable.req, &VendorTable ); // user call
back
     return(USB_TRUE);
```

```
    }   }
    /* `#END` */

    return(requestHandled);
}


/***************************************************************************
* Additional user functions supporting Vendor Specific Requests
***************************************************************************/

/* `#START VENDOR_SPECIFIC_FUNCTIONS` Place any additional functions here */

/* `#END` */

#endif /* USB_EXTERN_VND */
```

```
/* [] END OF FILE */
```

```
//////////////////////////////////////////////////////////////////////////////
          PART 2: Users Vendor function process
//////////////////////////////////////////////////////////////////////////////
#include <USB_Blk.h>
//////////////////////////////////////////////////////////////////////////////
#define EP_INP      (1)
#define EP_OUT      (2)
#define EP_BUF_SIZE (64)
//////////////////////////////////////////////////////////////////////////////
static byte buf[EP_BUF_SIZE];
static byte len;
//////////////////////////////////////////////////////////////////////////////
// Useful mapping:
// ===============
```

```
// EP0_DR0 (bmRequestType)      = 0x40|0xC0, a vendor specific command (enabled)
// EP0_DR1 (bRequest)           =
// EP0_DR2 (wValueL)            = 16-bit field, varies according to bRequest.
// EP0_DR3 (wValueH)            =
// EP0_DR4 (wIndexL)            = 16-bit field, varies according to bRequest.
// EP0_DR5 (wIndexH)            =
// EP0_DR6 (wLengthL)          = Number of bytes to transfer if there is a data phase.
// EP0_DR7 {wLengthH}           =
// ////////////////////////////////////////////////////////////////
/***
// ////////////////////////////////////////////////////////////////
// Vendor Request Basic Command        ////////////////////////////
// ////////////////////////////////////////////////////////////////
    VR_CHK=  0xA0,    // Check Device
    VR_INQ=  0xA1,    // Vendor Request
    VR_STS=  0xA2,    // Get Device Status
```

```
    VR_TRX=  0xAA,    // Transfer mode
    VR_LBK=  0xAB,    // Loop Back mode
    VR_WDT=  0xAD,    // Start transfer
    VR_DIS=  0xAF // Disconnect

//////////////////////////////////////////////////////////////////////////////
typedef struct        // vendor request table
{   byte typ;         // reqest type
    byte req;         // reqest code
    word val;         // value
    word idx;         // index
    word len;         // length
    byte *pRQ;        // data pointer
} VendorTableTyp;
***/
```

```
//////////////////////////////////////////////////////////////////////////////
volatile byte UsbControlRec[32];
volatile VendorCallBack VendorRequestProc;  // user call back pointer
static byte DeviceName[32];
//////////////////////////////////////////////////////////////////////////////
byte OnVendorRequest( byte req, VendorTableTyp *tab )
{   static int k;  byte buf[32];
    LCD_Printf(0,0, " Rq=%02x   ", req );
    LCD_Printf(0,8, " Ln=%d  ", tab->len );
    LCD_Printf(1,8, " Vp=%d  ", USB_VBusPresent() );

// ////////////////////////////////////////////////////////////
//   VR_CHK=  0xA0,    // Check Device
//   VR_INQ=  0xA1,    // Vendor Request
//   VR_STS=  0xA2,    // Get Device Status
// ////////////////////////////////////////////////////////////
```

```
LCD_Printf(1,0, " K=%d ", ++k );


switch( req )
{ case VR_CHK:        // check device ready
     USB_currentTD.count = 1;
     UsbControlRec[0] = ACK;        // ACK
     USB_currentTD.pData = &UsbControlRec[0];
     USB_InitControlRead();         // send response to host
     break;

  case VR_INQ:        // Inquialy device name
     strcpy( UsbControlRec, DeviceName );
     USB_currentTD.count = strlen(UsbControlRec);
     USB_currentTD.pData = &UsbControlRec[0];
     USB_InitControlRead();         // send device name to host
     break;
```

```
      default:
         break;
    }
    return 0;
}

// ////////////////////////////////////////////////////////////////
void USBULK_Start( byte dev_name[] )
{   YLed_ON;
    VendorRequestProc= & OnVendorRequest;
    strncpy( DeviceName, dev_name, sizeof(DeviceName)-1 );
    USB_Start(0, USB_3V_OPERATION);         // Start USBFS Operation with 3V operation
    while(!USB_GetConfiguration());     // Wait for Device to enumerate
    USB_EnableOutEP(EP_OUT);
    YLed_OFF;
```

```
}

// ////////////////////////////////////////////////////////////////
int USBULK_Read( byte rec[], int tmout )
{ int len=0;
    TimeOut= tmout;   // tmout<0: wait forever, tmout=0: no wait //
    do
    {   while( USB_GetEPState(EP_OUT)!=USB_OUT_BUFFER_FULL && ( TimeOut>0 ||
tmout<0 ) )
             {  /* Wait for data received HERE */  };
        if( USB_GetEPState(EP_OUT)==USB_OUT_BUFFER_FULL )
        {   len = USB_GetEPCount(EP_OUT);      // Read received bytes count
            if(len>EP_BUF_SIZE ) len= EP_BUF_SIZE;
            len= USB_ReadOutEP(EP_OUT, &rec[0], len);   // read from host
            // LCD_Printf( 1,0, " Tx len=%d ", length );
    }   } while( TimeOut );
```

```
    return len;
}

// ////////////////////////////////////////////////////////////////
void USBULK_Write( byte rec[], int len )
{   while(USB_GetEPState(EP_INP) != USB_IN_BUFFER_EMPTY)
          {  /* Check for IN buffer is empty */ };
    USB_LoadInEP(EP_INP, rec, len );       // send to host
}

////////////////////////////////////////////////////////////////////////////
```