

1. LIBRERÍA DEL USB: MPUSBAPI.DLL

1.1. INTRODUCCIÓN

Para una mayor facilidad de desarrollo de aplicaciones basadas en el bus USB, Microchip ha creado un archivo dll en el que proporciona las funciones de acceso al puerto USB con un microcontrolador de la familia PIC18Fxx5x.

Para un funcionamiento correcto, se necesita el driver mchpusb.sys.

Este archivo sirve tanto para Visual Basic como para Visual C, entre otros.

1.2. FUNCIONES

1.2.1. MPUSBGETDLLVERSION(VOID)

Lee el nivel de revisión del MPUSAPI.dll. Es un nivel de revisión de 32bits.

Esta función no devuelve la versión del código, no realiza nada con el USB.

Devuelve la versión de la dll en formato hexadecimal de 32bits.

`MPUSBGetDLLVersion()`

1.2.2. MPUSBGETDEVICECOUNT(PVID_PID)

Devuelve el número de dispositivo con VID_PID asignado.

pVID_PID: Input: cadena de caracteres del número de identificación asignado.

`MPUSBGetDeviceCount(vid_pid)`

1.2.3. **MPUSBOPEN(INSTANCE, PVID_PID, PEP, DWDIR, DWRESERVED)**

Devuelve el acceso al pipe del Endpoint con el VID_PID asignado.

Todas las pipes se abren con el atributo FILE_FLAG_OVERLAPPED. Esto permite que MPUSBRead, MPUSBWrite y MPUSBReadInt tengan un valor de time-out.

Nota: el valor del time-out no tiene sentido en una pipe síncrona.

instance: Input: Un número de dispositivo para abrir. Normalmente, se utiliza primero la llamada de MPUSBGetDeviceCount para saber cuantos dispositivos hay.

Es importante entender que el driver lo comparten distintos dispositivos. El número devuelto por el MPUSBGetDeviceCount tiene que ser igual o menor que el número de todos los dispositivos actualmente conectados y usando el driver genérico.

Ejemplo:

Si hay tres dispositivos con los siguientes PID_VID conectados:

- ◆ Dispositivo tipo 0, VID 0x04d8, PID 0x0001
- ◆ Dispositivo tipo 1, VID 0x04d8, PID 0x0002
- ◆ Dispositivo tipo 2, VID 0x04d8, PID 0x0003

Si el dispositivo que nos interesa tiene VID=0x04d8 y PID=0x0002 el MPUSBGetDeviceCount devolverá un '1'.

Al llamar la función tiene que haber un mecanismo que intente llamar MPUSOpen() desde 0 hasta MAX_NUM_MPUSB_DEV. Se tiene que contar el número de llamadas exitosas. Cuando este número sea igual al número devuelto por MPUSBGetDeviceCount, hay que dejar de hacer las llamadas porque no puede haber más dispositivos con el mismo VID_PID.

pVID_PID: Input: String que contiene el PID&VID del dispositivo objetivo. El formato es "vid_xxxx&pid_yyyy". Donde xxxx es el valor del VID y el yyyy el del PID, los dos en hexadecimal.

Ejemplo:

Si un dispositivo tiene un VID=0x04d8 y un PID=0x000b, el string de entrada es: "vid_0x04d8&pid_0x000b".

pEP: Input: String con el número del Endpoint que se va a abrir. El formato es "\\MCHP_EPz" o "\\MCHP_EPz" dependiendo del lenguaje de programación. Donde z es el número del Endpoint en decimal.

Ejemplo:

"\\MCHP_EP1" o "\\MCHP_EP1"

Este argumento puede ser NULL (nulo) para crear lazos con Endpoints de funciones no específicas.

Las funciones específicas son: MPUSBRead, MPUSBWrite, MPUSBReadInt.

Nota: Para utilizar MPUSBReadInt(), el formato de pEP tiene que ser “\\MCHP_EPz_ASYNC”. Esta opción sólo está disponible para un Endpoint interrupción IN. La pipe de datos abierta con “_ASYNC” debe almacenar datos con el intervalo especificado en el Endpoint descriptor con un máximo de 100 recepciones. Cualquier otro dato recibido después de llenar el buffer del driver se ignora.

La aplicación del usuario tiene que llamar MPUSBReadInt() a menudo sin superar el máximo de 100.

dwDir: Especifica la dirección del Endpoint:

- ◆ MP_READ: para MPUSBRead y MPUSBReadInt
- ◆ MP_Write: para MPUSBWrite

dwReserved: por ahora nada.

`MPUSBOpen(0, vid_pid, out_pipe, MP_WRITE, 0)`

1.2.4. MPUSBREAD(HANDLE, PDATA, DWLEN, PLENGTH, DWMILLISECONDS)

handle: Input: Identifica la pipe del Endpoint que se va a leer. La pipe unida tiene que crearse con el atributo de acceso MP_READ.

pData: Output: Puntero al buffer que recibe el dato leído de la pipe.

dwLen: Input: Especifica el número de bytes que hay que leer de la pipe.

pLenght: Output: Puntero al número de bytes leídos. MPUSBRead pone este valor a cero antes de cualquier lectura o de chequear un error.

dwMilliseconds: Input: Especifica el intervalo de time-out en milisegundos. La función vuelve si transcurre el intervalo aunque no se complete la operación. Si dwMilliseconds=0, la función comprueba los datos de la pipe y vuelve inmediatamente. Si dwMilliseconds es infinito, el intervalo de time-out nunca termina.

`MPUSBRead(myInPipe, VarPtr(s(0)), DatosDeseados, Datos, 1000)`

1.2.5. MPUSBWRITE(HANDLE, PDATA, DWLEN, PLENGTH, DWMILLISECONDS)

handle: Input: Identifica la pipe del Endpoint que se va a escribir. La pipe unida tiene que crearse con el atributo de acceso MP_WRITE.

pData: Output: Puntero al buffer que contiene los datos que se van a escribir en la pipe.

dwLen: Input: Especifica el número de bytes que se van a escribir en la pipe.

pLenght: Output: Puntero al número de bytes que se escriben al llamar esta función. MPUSBWrite pone este valor a cero antes de cualquier lectura o de chequear un error.

dwMilliseconds: Input: Especifica el intervalo de time-out en milisegundos. La función vuelve si transcurre el intervalo aunque no se complete la operación. Si dwMilliseconds=0, la función comprueba los datos de la pipe y vuelve inmediatamente. Si dwMilliseconds es infinito, el intervalo de time-out nunca termina.

```
MPUSBWrite(myOutPipe, VarPtr(SendData(0)), bytes, VarPtr(bytes), 1000)
```

1.2.6. MPUSBREADINT(HANDLE, PDATA, DWLEN, PLENGTH, DWMILLISECONDS)

handle: Input: Identifica la pipe del Endpoint que se va a leer. La pipe unida tiene que crearse con el atributo de acceso MP_READ.

pData: Output: Puntero al buffer que recibe el dato leído de la pipe.

dwLen: Input: Especifica el número de bytes que hay que leer de la pipe.

pLenght: Output: Puntero al número de bytes leídos. MPUSBRead pone este valor a cero antes de cualquier lectura o de chequear un error.

dwMilliseconds: Input: Especifica el intervalo de time-out en milisegundos. La función vuelve si transcurre el intervalo aunque no se complete la operación. Si dwMilliseconds=0, la función comprueba los datos de la pipe y vuelve inmediatamente. Si dwMilliseconds es infinito, el intervalo de time-out nunca termina.

```
MPUSBReadInt(myOutPipe, VarPtr(SendData(0)), bytes, VarPtr(bytes), 1000)
```

1.2.7. MPUSBCLOSE(HANDLE)

Cierra una determinada unión.

handle: Input: Identifica la pipe del Endpoint que se va a cerrar.

```
MPUSBClose (myOutPipe)
```

1.3. TIPOS DE TRANSFERENCIAS

En este apartado se recomienda que función utilizar dependiendo del tipo de transferencia.

Tipo	Función	¿Aplicable time-out?
Interrupt IN	MPUSRead, MPUSReadInt	si
Interrupt OUT	MPUSBWrite	si
Bulk IN	MPUSBRead	si
Bulk OUT	MPUSWrite	si
Isochronous IN	MPUSBRead	no
Isochronous OUT	MPUSBWrite	no

Interrupt: tipo interrupción

Isochronous: tipo síncrono

Nota: “Input” y “output” se refiere a los parámetros designados en las llamadas a estas funciones, que son lo opuesto a los sentidos comunes desde la perspectiva de una aplicación haciendo llamadas.

1.4. DECLARACIÓN DE CONSTANTES Y VARIABLES

Aquí aparecen las constantes y variables que el fabricante recomienda usar. Todas son optativas, dejando la elección al programador.

También, se comentan las pequeñas variaciones que existen al declarar estas variables en los distintos lenguajes.

```
MPUS_FAIL=0
MPUSB_SUCCESS=1
MP_WRITE=0
MP_READ=1
MAX_NUM_MPUSB_DEV=127
vid_pid= "vid_04d8&pid_0011"
```

En Visual Basic:

```
out_pipe= "\MCHP_EPx"
in_pipe= "\MCHP_EPy"
```

En C y Delphi:

```
out_pipe= "\\MCHP_EPx"
in_pipe= "\\MCHP_EPy"
```

Siendo x e y números del Endpoint por los que se van a realizar las transmisiones.

1.5. DECLARACIÓN DE LAS FUNCIONES

En el último punto de la librería, se comenta como incluir las funciones en varios lenguajes de programación.

1.5.1. C

Se declara con #include "_mpusbapi.h". Los datos devueltos son:

```
DWORD      _MPUSBGetDLLVersion(void)
DWORD      _MPUSBGetDeviceCount(PCHAR pVID_PID)
HANDLE      _MPUSBOpen(DWORD instance, PCHAR pVID_PID, PCHAR pEP, DWORD
dwDir, DWORD dwReserved);
BOOLEAN     _MPUSBClose(HANDLE handle);
DWORD       _MPUSBRead(HANDLE handle, PVOID pData, DWORD dwLen, PDWORD
pLength, DWORD dwMilliseconds);
DWORD       _MPUSBWrite(HANDLE handle, PVOID pData, DWORD dwLen, PDWORD
pLength, DWORD dwMilliseconds);
DWORD       _MPUSBReadInt(HANDLE handle, PVOID pData, DWORD dwLen, PDWORD
pLength, DWORD dwMilliseconds);
```

1.5.2. VB

```
Public Declare Function MPUSBGetDLLVersion Lib "mpusbapi.dll" () As Long
Public Declare Function MPUSBGetDeviceCount Lib "mpusbapi.dll" (ByVal pVID_PID As
String) As Long
Public Declare Function MPUSBOpen Lib "mpusbapi.dll" (ByVal instance As Long, ByVal
pVID_PID As String, ByVal pEP As String, ByVal dwDir As Long, ByVal dwReserved As
Long) As Long
Public Declare Function MPUSBClose Lib "mpusbapi.dll" (ByVal handle As Long) As Long
Public Declare Function MPUSBRead Lib "mpusbapi.dll" (ByVal handle As Long, ByVal pData
As Long, ByVal dwLen As Long, ByRef pLength As Long, ByVal dwMilliseconds As Long) As
Long
Public Declare Function MPUSBWrite Lib "mpusbapi.dll" (ByVal handle As Long, ByVal
pData As Long, ByVal dwLen As Long, ByRef pLength As Long, ByVal dwMilliseconds As
Long) As Long
Public Declare Function MPUSBReadInt Lib "mpusbapi.dll" (ByVal handle As Long, ByVal
pData As Long, ByVal dwLen As Long, ByRef pLength As Long, ByVal dwMilliseconds As
Long) As Long
```