Librerías sobre el USB en C18

Las librerías que aquí aparecen son un ejemplo, dependiendo de la clase (HID, MSD, GEN, BOOT o CDC) hay algunas variaciones.

En rojo aparecen los datos que hay que cambiar en cada firmware. En azul las líneas que dependen de la clase. Para más información visitar la página de microchip, en la que hay ejemplos de cada clase.

usb.h

Esta librería proporciona la forma de incluir todos los archivos necesarios del firmware del USB de Microchip.

El orden de inclusión es importante, ya que se resuelven los conflictos de dependencia con el orden correcto:

```
#include "autofiles\usbcfg.h"
#include "system\usb\usbdefs\usbdefs_std_dsc.h"
#include "autofiles\usbdsc.h"

#include "system\usb\usbdefs\usbdefs_ep0_buff.h"
#include "system\usb\usbmmap.h"

#include "system\usb\usbdrv\usbdrv.h"
#include "system\usb\usbctrltrf\usbctrltrf.h"
#include "system\usb\usb9\usb9.h"
```

Si USB_USE_HID está definida se incluye la librería hid.h

```
#if defined(USB_USE_HID)        Ver autofiles\usbcfg.h
#include "system\usb\class\hid\hid.h"
#endif
```

Si USB_USE_MSD está definida se incluye la librería msd.h

```
#if defined(USB_USE_MSD)        Ver autofiles\usbcfg.h
#include "system\usb\class\msd\msd.h"
#endif
```

Si USB_USE_CDC está definida se incluye la librería cdc.h

```
#if defined(USB_USE_CDC)        Ver autofiles\usbcfg.h
#include "system\usb\class\cdc\cdc.h"
#endif
```

usbcfg.h: Configuración

Definiciones

Tamaño del buffer del Endpoint 0 (8, 16 32 ó 64):

```
#define EP0_BUFF_SIZE 8
```

Número máximo de interrupciones:

```
#define MAX_NUM_INT 1
```

Definición de parámetros descritos en usbdrv.h:

Modo Ping-Pong:

```
#define MODE_PP         _PPBM0
```

Valor de configuración:

```
#define UCFG_VAL        _PUEN|_TRINT|_FS|MODE_PP
```

E/S auto-alimentadas:

```
#define USE_SELF_POWER_SENSE_IO
```

E/S dependientes del bus USB:

```
#define USE_USB_BUS_SENSE_IO
```

Las dos definiciones anteriores se realizarán cuando sea necesario, siendo totalmente independiente una de la otra.

Uso de la clase del dispositivo

Dependiendo de la aplicación hay que definir la clase que vamos a utilizar.

- ✓ Uso de la interfaz humana del dispositivo USB: `#define USB_USE_HID`
- ✓ Uso del CDC del dispositivo: `#define USB_USE_CDC`
- ✓ Uso de la clase Almacenamiento masivo: `#define USB_USE_MSD`

MUID = Microchip USB Clase ID

Se utiliza para identificar la clase del USB de la sesión actual de control de la transferencia del EP0:

`#define MUID_NULL`	0	Ninguna
`#define MUID_USB9`	1	USB9
`#define MUID_HID`	2	Interfaz humana
`#define MUID_CDC`	3	Clase de comunicación del dispositivo

Distribución de los Endpoint

`#define HID_INTF_ID`	0x00	Identificación de la interfaz
`#define HID_UEP`	UEP1	Endpoints que se utlizan
`#define HID_BD_OUT`	ep1Bo	Buffer descriptor de salida
`#define HID_INT_OUT_EP_SIZE`	3	Tamaño de la interrupción del Endpoint de salida
`#define HID_BD_IN`	ep1Bi	Buffer descriptor de entrada
`#define HID_INT_IN_EP_SIZE`	3	Tamaño de la interrupción del Endpoint de entrada
`#define HID_NUM_OF_DSC`	1	Número de descriptores
`#define HID_RPT01_SIZE`	50	Tamaño del informe

Macros HID

Ver la dirección del descriptor HID:

```
#define mUSBGetHIDDscAdr(ptr)
```

```
{
if(usb_active_cfg == 1)
ptr = (rom byte*)&cfg01.hid_i00a00;
}
```

Ver la dirección del informe del descriptor HID:

```
#define mUSBGetHIDRptDscAdr(ptr)
{
if(usb_active_cfg == 1)
ptr = (rom byte*)&hid_rpt01;
}
```

Ver el tamaño del informe del descriptor HID:

```
#define mUSBGetHIDRptDscSize(count)
{
if(usb_active_cfg == 1)
count = sizeof(hid_rpt01);
}
```

Número máximo de Endpoints:

```
#define MAX_EP_NUMBER      1              //UEP1
```

usbdefs_std_dsc.h: Definiciones estándar de los descriptores:

Incluye:

#include "system\typedefs.h"	Librería en la que se definen tipos de datos

Definiciones

Tipos de descriptores:

#define DSC_DEV	0x01	Descriptor del dispositivo
#define DSC_CFG	0x02	Descriptor de configuración
#define DSC_STR	0x03	Descriptor de la secuencia
#define DSC_INTF	0x04	Descriptor de la interfaz
#define DSC_EP	0x05	Descriptor del Endpoint

Definición de los Endpoint (sólo usarse con los descriptores, para cualquier otro uso utilizar los definidos en usbdrv.h):

#define _EP01_OUT	0x01
#define _EP01_IN	0x81
#define _EP02_OUT	0x02
#define _EP02_IN	0x82
#define _EP03_OUT	0x03
#define _EP03_IN	0x83
#define _EP04_OUT	0x04
#define _EP04_IN	0x84
#define _EP05_OUT	0x05
#define _EP05_IN	0x85
#define _EP06_OUT	0x06
#define _EP06_IN	0x86
#define _EP07_OUT	0x07
#define _EP07_IN	0x87
#define _EP08_OUT	0x08
#define _EP08_IN	0x88
#define _EP09_OUT	0x09
#define _EP09_IN	0x89
#define _EP10_OUT	0x0A
#define _EP10_IN	0x8A
#define _EP11_OUT	0x0B
#define _EP11_IN	0x8B
#define _EP12_OUT	0x0C
#define _EP12_IN	0x8C
#define _EP13_OUT	0x0D
#define _EP13_IN	0x8D
#define _EP14_OUT	0x0E
#define _EP14_IN	0x8E
#define _EP15_OUT	0x0F
#define _EP15_IN	0x8F

Atributos de configuración:

#define _DEFAULT	0x01<<7	Valor por defecto (El Bit 7 se activa)
#define _SELF	0x01<<6	Auto-alimentado (Mantener si está activo)

#define _RWU	0x01<<5	Reinicio remoto (Mantener si está activo)

Tipo de transferencia del Endpoint:

#define _CTRL	0x00	Transferencia de control
#define _ISO	0x01	Transferencia síncrona
#define _BULK	0x02	Transferencia Bulk
#define _INT	0x03	Transferencia interrupción

Tipo de sincronización del Endpoint de transferencia síncrona:

#define _NS	0x00<<2	Sin sincronización
#define _AS	0x01<<2	Asíncrona
#define _AD	0x02<<2	Adaptivo
#define _SY	0x03<<2	Síncrona

Utilización del Endpoint tipo síncrono:

#define _DE	0x00<<4	Endpoint de datos
#define _FE	0x01<<4	Endpoint retroalimentado
#define _IE	0x02<<4	Endpoint de datos y retroalimentado

Estructuras

Estructura de los descriptores del dispositivo USB (ver usbdsc.c):

```
typedef struct _USB_DEV_DSC
{
  Longitud;                    Tipo de descriptor;          Bcd del USB;
  byte bLength;                byte bDscType;               word bcdUSB;
  Clase del dispositivo;       Subclase del dispositivo;    Protocolo del dispositivo;
  byte bDevCls;                byte bDevSubCls;             byte bDevProtocol;
  Tamaño máximo del paquete;   Identificador del fabricante; Identificación del
producto;
  byte bMaxPktSize0;           word idVendor;               word idProduct;
  Bcd del dsipositivo;         Información de la manufactura; Información del producto;
  word bcdDevice;              byte iMFR;                   byte iProduct;
  Número de serie;             Número de configuración;
  byte iSerialNum;             byte bNumCfg;
} USB_DEV_DSC;
```

Estructura del descriptor de configuración del USB:

```
typedef struct _USB_CFG_DSC
{
  Longitud;                    Tipo de descriptor;          Longitud total;
  byte bLength;                byte bDscType;               word wTotalLength;
  Número de interfaz;          Valor de configuración;      Información de
configuración;
```

```
  byte bNumIntf;          byte bCfgValue;          byte iCfg;
  Atributos;              Máxima energía;
  byte bmAttributes;      byte bMaxPower;
} USB_CFG_DSC;
```

Estructura de la interfaz del dispositivo USB:

```
typedef struct _USB_INTF_DSC
{
  Longitud;               Tipo de descriptor;      Número de interfaz;
  byte bLength;           byte bDscType;           byte bIntfNum;
  Configuración alterna;  Número de Endpoints;     Clase de interfaz;
  byte bAltSetting;       byte bNumEPs;            byte bIntfCls;
  Subclase de interfaz;   Protocolo de la interfaz; Información de la interfaz;
  byte bIntfSubCls;       byte bIntfProtocol;      byte iIntf;
} USB_INTF_DSC;
```

Estructura del Descriptor del Endpoint del USB:

```
typedef struct _USB_EP_DSC
{
  Longitud;               Tipo de descriptor;      Dirección del Endpoint;
  byte bLength;           byte bDscType;           byte bEPAdr;
  Atributos;              Tamaño máximo del paquete; Intervalo;
  byte bmAttributes;      word wMaxPktSize;        byte bInterval;
} USB_EP_DSC;
```

usbdsc.h: descriptores

Librerías que incluye

```
#include "system\typedefs.h"
#include "autofiles\usbcfg.h"
#include "system\usb\usb.h"
```

Definiciones

```
#define CFG01 rom struct                    Configuración 01
{
  USB_CFG_DSC       cd01;                   Descriptor de configuración
  USB_INTF_DSC      i00a00;                 Descriptor de la interfaz
  USB_EP_DSC        ep01o_i00a00;           Endpoint descriptor de salida
  USB_EP_DSC        ep01i_i00a00;           Endpoint descriptor de entrada
} cfg01
```

Externas

```
extern rom USB_DEV_DSC device_dsc;
extern CFG01;
extern rom struct{byte bLength;byte bDscType;word string[1];}sd000;
```

usbdefs_ep0_buff.h: Descripciones del buffer del Endpoint 0

Incluye

```
#include "system\typedefs.h"
#include "autofiles\usbcfg.h"
```

Control de transferencias setup

Cada paquete setup tiene 8bytes. Sin embargo, el tamaño del buffer del Endpoint 0 es el especificado en la librería usbcfg.h.

El tamaño del buffer puede ser de 8, 16, 32 ó 64.

Los primeros 8bytes se definen para direccionarse directamente para mejorar la velocidad y reducir el tamaño del código. El resto de bytes se direccionan indirectamente.

```
typedef union _CTRL_TRF_SETUP
{
  Matriz para el direccionamiento indirecto
  struct
  {
  byte _byte[EP0_BUFF_SIZE];
  };

  Respuestas estándar del dispositivo
  struct
  {
  byte bmRequestType;          Tipo de respuesta
  byte bRequest;               Respuesta
  word wValue;                 Valor
  word wIndex;                 Índice
  word wLength;                Longitud
  };
  struct
  {
  unsigned :8;
  unsigned :8;
  WORD W_Value;                Valor
  WORD W_Index;                Índice
  WORD W_Length;               Longitud
  };
  struct
  {
  unsigned Recipient:5;        Dispositivo, Interfaz, Endpoint, Otro
  unsigned RequestType:2;      Estándar, Clase, Fabricante, Reservado
  unsigned DataDir:1;          Host-al-dispositivo, Dispositivo-al-host
  unsigned :8;
  byte bFeature;               Reinicio remoto, Paro del Endpoint
  unsigned :8;
  unsigned :8;
```

```
unsigned :8;
unsigned :8;
unsigned :8;
};
struct
{
unsigned :8;
unsigned :8;
byte bDscIndex;          Sólo para configuración y String del descriptor
byte bDscType;           Dispositivo, Configuración, String
word wLangID;            Identificación de idioma
unsigned :8;
unsigned :8;
};
struct
{
unsigned :8;
unsigned :8;
BYTE bDevADR;            Dirección del dispositivo 0-127
byte bDevADRH;           Tiene que ser cero
unsigned :8;
unsigned :8;
unsigned :8;
unsigned :8;
};
struct
{
unsigned :8;
unsigned :8;
byte bCfgValue;          Valor de configuración 0-255
byte bCfgRSD;            Tiene que ser cero (Reservado)
unsigned :8;
unsigned :8;
unsigned :8;
unsigned :8;
};
struct
{
unsigned :8;
unsigned :8;
byte bAltID;             Valor alterno de configuración 0-255
byte bAltID_H;           Tiene que ser cero
byte bIntfID;            Valor del número de interfaz 0-255
byte bIntfID_H;          Tiene que ser cero
unsigned :8;
unsigned :8;
};
struct
{
unsigned :8;
```

```
  unsigned :8;
  unsigned :8;
  unsigned :8;
  byte bEPID;                    Identificación del Endpoint ID (Número y Dirección)
  byte bEPID_H;                  Tiene que ser cero
  unsigned :8;
  unsigned :8;
  };
  struct
  {
  unsigned :8;
  unsigned :8;
  unsigned :8;
  unsigned :8;
  unsigned EPNum:4;              Número del Endpoint 0-15
  unsigned :3;
  unsigned EPDir:1;              Dirección del Endpoint: 0-OUT, 1-IN
  unsigned :8;
  unsigned :8;
  unsigned :8;
  };
} CTRL_TRF_SETUP;
```

Control de transferencia de datos

```
typedef union _CTRL_TRF_DATA
{
```

Matriz para el direccionamiento indirecto:

```
  struct
  {
  byte _byte[EP0_BUFF_SIZE];
  };
```

Los primeros 8bytes direccionables directamente:

```
  struct
  {
  byte _byte0;
  byte _byte1;
  byte _byte2;
  byte _byte3;
  byte _byte4;
  byte _byte5;
  byte _byte6;
  byte _byte7;
  };
  struct
  {
  word _word0;
  word _word1;
  word _word2;
  word _word3;
```

```
    };
} CTRL_TRF_DATA;
```

usbmmap.h

Incluye

```
#include "system\typedefs.h"
```

Definiciones

Parámetros de inicialización del descriptor del registro estado:

```
#define _BSTALL     0x04    Parada del buffer activa
#define _DTSEN      0x08    Dato de sincronización activo
#define _INCDIS     0x10    Incremento de dirección desactivado
#define _KEN        0x20    Guardado del buffer descriptor por el SIE
  activo
#define _DAT0       0x00    Paquete DATA0 esperando el siguiente
#define _DAT1       0x40    Paquete DATA1 esperando el siguiente
#define _DTSMASK    0x40    Máscara DTS
#define _USIE       0x80    El SIE controla el buffer
#define _UCPU       0x00    La CPU controla el buffer
```

Estados del dispositivo USB. Para utilizarlos con [byte usb_device_state]:

```
#define DETACHED_STATE      0   Sin conexión
#define ATTACHED_STATE      1   Conectado
#define POWERED_STATE       2   Alimentado
#define DEFAULT_STATE       3   Por defecto
#define ADR_PENDING_STATE   4   Pendiente de dirección
#define ADDRESS_STATE       5   Direccionado
#define CONFIGURED_STATE    6   Configurado
```

Tipos de memoria para el control de transferencias, utilizado en USB_DEVICE_STATUS:

```
#define _RAM 0
#define _ROM 1
```

Tipos

```
typedef union _USB_DEVICE_STATUS            Estado del dispositivo
{
  byte _byte;
  struct
  {
  unsigned RemoteWakeup:1;      [0]Desactivado [1]Activado: Ver usbdrv.h, usb9.h
  unsigned ctrl_trf_mem:1;      [0]RAM [1]ROM
  };
} USB_DEVICE_STATUS;



typedef union _BD_STAT          Estado del buffer descriptor
{
```

```
  byte _byte;
  struct{
  unsigned BC8:1;
  unsigned BC9:1;
  unsigned BSTALL:1;          Parada del buffer activa
  unsigned DTSEN:1;           Dato de sincronización activo
  unsigned INCDIS:1;          Incremento de dirección desactivado
  unsigned KEN:1;             Guardado del buffer descriptor por el SIE activo
  unsigned DTS:1;             Valor del dato de sincronización
  unsigned UOWN:1;            Propiedad del USB
  };
  struct{
  unsigned BC8:1;
  unsigned BC9:1;
  unsigned PID0:1;
  unsigned PID1:1;
  unsigned PID2:1;
  unsigned PID3:1;
  unsigned :1;
  unsigned UOWN:1;
  };
  struct{
  unsigned :2;
  unsigned PID:4;             Paquete de identificación
  unsigned :2;
  };
} BD_STAT;

typedef union _BDT            Tabla del buffer descriptor
{
  struct
  {
  BD_STAT Stat;
  byte Cnt;
  byte ADRL;                  Dirección del buffer baja
  byte ADRH;                  Dirección del buffer alta
  };
  struct
  {
  unsigned :8;
  unsigned :8;
  byte* ADR;                  Dirección del Buffer
  };
} BDT;
```

Externas

```
extern byte usb_device_state;
extern USB_DEVICE_STATUS usb_stat;
extern byte usb_active_cfg;
extern byte usb_alt_intf[MAX_NUM_INT];
```

```
extern volatile far BDT ep0Bo;      Buffer descriptor del Endpoint #0 Out
extern volatile far BDT ep0Bi;      Buffer descriptor del Endpoint #0 In
extern volatile far BDT ep1Bo;      Buffer descriptor del Endpoint #1 Out
extern volatile far BDT ep1Bi;      Buffer descriptor del Endpoint #1 In
extern volatile far BDT ep2Bo;      Buffer descriptor del Endpoint #2 Out
extern volatile far BDT ep2Bi;      Buffer descriptor del Endpoint #2 In
extern volatile far BDT ep3Bo;      Buffer descriptor del Endpoint #3 Out
extern volatile far BDT ep3Bi;      Buffer descriptor del Endpoint #3 In
extern volatile far BDT ep4Bo;      Buffer descriptor del Endpoint #4 Out
extern volatile far BDT ep4Bi;      Buffer descriptor del Endpoint #4 In
extern volatile far BDT ep5Bo;      Buffer descriptor del Endpoint #5 Out
extern volatile far BDT ep5Bi;      Buffer descriptor del Endpoint #5 In
extern volatile far BDT ep6Bo;      Buffer descriptor del Endpoint #6 Out
extern volatile far BDT ep6Bi;      Buffer descriptor del Endpoint #6 In
extern volatile far BDT ep7Bo;      Buffer descriptor del Endpoint #7 Out
extern volatile far BDT ep7Bi;      Buffer descriptor del Endpoint #7 In
extern volatile far BDT ep8Bo;      Buffer descriptor del Endpoint #8 Out
extern volatile far BDT ep8Bi;      Buffer descriptor del Endpoint #8 In
extern volatile far BDT ep9Bo;      Buffer descriptor del Endpoint #9 Out
extern volatile far BDT ep9Bi;      Buffer descriptor del Endpoint #9 In
extern volatile far BDT ep10Bo;     Buffer descriptor del Endpoint #10 Out
extern volatile far BDT ep10Bi;     Buffer descriptor del Endpoint #10 In
extern volatile far BDT ep11Bo;     Buffer descriptor del Endpoint #11 Out
extern volatile far BDT ep11Bi;     Buffer descriptor del Endpoint #11 In
extern volatile far BDT ep12Bo;     Buffer descriptor del Endpoint #12 Out
extern volatile far BDT ep12Bi;     Buffer descriptor del Endpoint #12 In
extern volatile far BDT ep13Bo;     Buffer descriptor del Endpoint #13 Out
extern volatile far BDT ep13Bi;     Buffer descriptor del Endpoint #13 In
extern volatile far BDT ep14Bo;     Buffer descriptor del Endpoint #14 Out
extern volatile far BDT ep14Bi;     Buffer descriptor del Endpoint #14 In
extern volatile far BDT ep15Bo;     Buffer descriptor del Endpoint #15 Out
extern volatile far BDT ep15Bi;     Buffer descriptor del Endpoint #15 In

extern volatile far CTRL_TRF_SETUP SetupPkt;
extern volatile far CTRL_TRF_DATA CtrlTrfData;

#if defined(USB_USE_HID)        Si está definido USB_USE_HID
extern volatile far unsigned char hid_report_out[HID_INT_OUT_EP_SIZE];
extern volatile far unsigned char hid_report_in[HID_INT_IN_EP_SIZE];
#endif
```

usbdrv.h: Driver del USB

Incluye

```
#include "system\typedefs.h"
#include "system\usb\usb.h"
```

Definiciones

Parámetros de configuración iniciales:

#define _PPBM0	0x00	Buffer Ping-pong Modo 0
#define _PPBM1	0x01	Buffer Ping-pong Modo 1
#define _PPBM2	0x02	Buffer Ping-pong Modo 2
#define _LS	0x00	Modo USB Low-Speed
#define _FS	0x04	Modo USB Full-Speed
#define _TRINT	0x00	Transmisor-receptor interno
#define _TREXT	0x08	Transmisor-receptor externo
#define _PUEN	0x10	Usar resistencias pull-up internas
#define _OEMON	0x40	Usar el indicador de salida SIE
#define _UTEYE	0x80	Usar el test “Patrón de ojo”

Parámetros de los Endpoint iniciales:

#define EP_CTRL	0x06	Pipe de control
#define EP_OUT	0x0C	Pipe de salida
#define EP_IN	0x0A	Pipe de entrada
#define EP_OUT_IN	0x0E	Pipe de entrada y salida
#define HSHK_EN	0x10	Activar paquetes de protocolo

Los paquetes de protocolo se tienen que desactivar en la síncronas

Definiciones de los Endpoints PICmicro

El formato de la dirección de los EP PICmicro: X:EP3:EP2:EP1:EP0:DIR:PPBI:X

Esto se utiliza cuando se comprueba el valor leído de la USTAT

NOTA: estas definiciones no se usan en los descriptores porque tienen distinto formato. Se definen en : "system\usb\usbdefs\usbdefs_std_dsc.h"

#define OUT	0	
#define IN	1	
#define PIC_EP_NUM_MASK	0b01111000	Número de máscara del Endpoint
#define PIC_EP_DIR_MASK	0b00000100	Dirección de la máscara del Endpoint

NOTA: la librería tiene una errata en la definición de los Endpoints, lo correcto es:

```
#define EP00_OUT        ((0x00<<3)|(OUT<<2))
#define EP00_IN         ((0x00<<3)|(IN<<2))
#define EP01_OUT        ((0x01<<3)|(OUT<<2))
#define EP01_IN         ((0x01<<3)|(IN<<2))
#define EP02_OUT        ((0x02<<3)|(OUT<<2))
#define EP02_IN         ((0x02<<3)|(IN<<2))
#define EP03_OUT        ((0x03<<3)|(OUT<<2))
#define EP03_IN         ((0x03<<3)|(IN<<2))
#define EP04_OUT        ((0x04<<3)|(OUT<<2))
#define EP04_IN         ((0x04<<3)|(IN<<2))
#define EP05_OUT        ((0x05<<3)|(OUT<<2))
#define EP05_IN         ((0x05<<3)|(IN<<2))
#define EP06_OUT        ((0x06<<3)|(OUT<<2))
#define EP06_IN         ((0x06<<3)|(IN<<2))
#define EP07_OUT        ((0x07<<3)|(OUT<<2))
#define EP07_IN         ((0x07<<3)|(IN<<2))
#define EP08_OUT        ((0x08<<3)|(OUT<<2))
#define EP08_IN         ((0x08<<3)|(IN<<2))
#define EP09_OUT        ((0x09<<3)|(OUT<<2))
#define EP09_IN         ((0x09<<3)|(IN<<2))
#define EP10_OUT        ((0x0A<<3)|(OUT<<2))
#define EP10_IN         ((0x0A<<3)|(IN<<2))
#define EP11_OUT        ((0x0B<<3)|(OUT<<2))
#define EP11_IN         ((0x0B<<3)|(IN<<2))
#define EP12_OUT        ((0x0C<<3)|(OUT<<2))
#define EP12_IN         ((0x0C<<3)|(IN<<2))
#define EP13_OUT        ((0x0D<<3)|(OUT<<2))
#define EP13_IN         ((0x0D<<3)|(IN<<2))
#define EP14_OUT        ((0x0E<<3)|(OUT<<2))
#define EP14_IN         ((0x0E<<3)|(IN<<2))
#define EP15_OUT        ((0x0F<<3)|(OUT<<2))
#define EP15_IN         ((0x0F<<3)|(IN<<2))
```

mInitializeUSBDriver()

Configura el modulo USB.
La definición de UCFG_VAL está en autofiles\usbcfg.h

Este registro determina: velocidad del USB Speed, selección de las resistencias pull-up del chip, selección del transmisor-receptor del chip, modo de chequeo "patrón de ojo", buffer modo Ping-pong

```
#define mInitializeUSBDriver()      {UCFG = UCFG_VAL;
  usb_device_state = DETACHED_STATE;
  usb_stat._byte = 0x00;
  usb_active_cfg = 0x00;
  }
```

void mDisableEP1to15()

Esta macro desactiva todos los Endpoints menos el 0.

Hay que invocar esta macro cada vez que el host envíe una señal de RESET o una respuesta a SET_CONFIGURATION

```
#define mDisableEP1to15()           ClearArray((byte*)&UEP1,15);
```

O lo que es lo mismo:

```
#define mDisableEP1to15()           UEP1=0x00;UEP2=0x00;UEP3=0x00;
  UEP4=0x00;UEP5=0x00;UEP6=0x00;UEP7=0x00;
  UEP8=0x00;UEP9=0x00;UEP10=0x00;UEP11=0x00;
  UEP12=0x00;UEP13=0x00;UEP14=0x00;UEP15=0x00;
```

mUSBBufferReady(buffer_dsc)

Precondición:

Endpoint IN: El buffer está cargado y listo para enviar.

Endpoint OUT: El buffer puede escribir al SIE.

Entrada: *byte buffer_dsc:* Nombre del grupo del buffer descriptor (e.j. ep0Bo, ep1Bi) declarado en usbmmap.h. Los nombres se pueden cambiar por legibilidad; ver los ejemplos en usbcfg.h (#define HID_BD_OUT ep1Bo)

Esta macro se tiene que llamar cada vez que ocurra:

1. Que se llene un buffer de un Endpoint, que no sea el EP0, con datos.
2. Que se lea un buffer de un Endpoint, que no sea el EP0.

Esta macro convierte la propiedad del buffer al SIE para dar servicio; además, cambia el bit DTS para sincronización.

```
#define mUSBBufferReady(buffer_dsc)
{
  buffer_dsc.Stat._byte &= _DTSMASK;              Guarda sólo el bit DTS
  buffer_dsc.Stat.DTS = !buffer_dsc.Stat.DTS;     Cambia el bit DTS
  buffer_dsc.Stat._byte |= _USIE|_DTSEN;          Cambia la propiedad al SIE
}
```

Prototipos públicos

```
void USBCheckBusStatus(void);
void USBDriverService(void);
void USBRemoteWakeup(void);
void USBSoftDetach(void);
```

```
void ClearArray(byte* startAdr,byte count);
```

usbctrltrf.h: Control de transferencias del USB

Incluye

```
#include "system\typedefs.h"
```

Definiciones

Estado de las transferencias de control

#define WAIT_SETUP	0	Espera Setup
#define CTRL_TRF_TX	1	Transf. de control de transmisión
#define CTRL_TRF_RX	2	Transf. de control de recepción

Tipos de Tokens:

#define SETUP_TOKEN	0b00001101	Token setup
#define OUT_TOKEN	0b00000001	Token de salida
#define IN_TOKEN	0b00001001	Token de entrada

Definición de los tipos de respuesta:

#define HOST_TO_DEV	0	Host-al-Dispositivo
#define DEV_TO_HOST	1	Dispositivo-al-Host
#define STANDARD	0x00	
#define CLASS	0x01	Clase
#define VENDOR	0x02	Fabricante
#define RCPT_DEV	0	Dispositivo destinatario
#define RCPT_INTF	1	Destinatario de la interfaz
#define RCPT_EP	2	Destinatario del Endpoint
#define RCPT_OTH	3	

Externas

```
extern byte ctrl_trf_session_owner;

extern POINTER pSrc;
extern POINTER pDst;
extern WORD wCount;
```

Prototipos públicos

```
void USBCtrlEPService(void);
void USBCtrlTrfTxService(void);
void USBCtrlTrfRxService(void);
void USBCtrlEPServiceComplete(void);
void USBPrepareForNextSetupTrf(void);
```

usb9.h

Incluye

```
#include "system\typedefs.h"
```

Definiciones

Códigos de respuesta estándar:

#define GET_STATUS	0	Obtiene estado
#define CLR_FEATURE	1	Borra característica
#define SET_FEATURE	3	Fija característica
#define SET_ADR	5	Fija dirección
#define GET_DSC	6	Obtiene descriptor
#define SET_DSC	7	Fija descriptor
#define GET_CFG	8	Obtiene configuración
#define SET_CFG	9	Fija configuración
#define GET_INTF	10	Obtiene interfaz
#define SET_INTF	11	Fija interfaz
#define SYNCH_FRAME	12	Marco de sincronismo

Características de los selectores estándar:

#define DEVICE_REMOTE_WAKEUP	0x01	Reinicio remoto del dispositivo
#define ENDPOINT_HALT	0x00	Paro del Endpoint

mUSBCheckAdrPendingState()

Rutina de chequeo especializado, comprueba si el dispositivo está en el estado "Pendiente de dirección" y le da servicio si está.

```
#define mUSBCheckAdrPendingState()
if(usb_device_state==ADR_PENDING_STATE)
  {
  UADDR = SetupPkt.bDevADR._byte;
  if(UADDR > 0)
  usb_device_state=ADDRESS_STATE;
  else
  usb_device_state=DEFAULT_STATE;
}
```

Prototipos públicos

```
void USBCheckStdRequest(void);
```

usbgen.h: USB genérico

Incluye

```
#include "system\typedefs.h"
```

Definiciones

(bit) mUSBGenRxIsBusy(void)

Esta macro se utiliza para comprobar que el Endpoint de salida está ocupado (lo controla el SIE) o no.

Uso típico: if(mUSBGenRxIsBusy())

```
#define mUSBGenRxIsBusy()		USBGEN_BD_OUT.Stat.UOWN
```

(bit) mUSBGenTxIsBusy(void)

Esta macro se utiliza para comprobar que el Endpoint de entrada está ocupado (lo controla el SIE) o no.

Uso típico: if(mUSBGenTxIsBusy())

```
#define mUSBGenTxIsBusy()		USBGEN_BD_IN.Stat.UOWN
```

byte mUSBGenGetRxLength(void)

Salida: mUSBGenGetRxLength devuelve usbgen_rx_len (longitude de Rx).

mUSBGenGetRxLength se utiliza para recuperar el número de bytes copiados al buffer del usuario en la última llamada a la función USBGenRead.

```
#define mUSBGenGetRxLength()		usbgen_rx_len
```

Externas

```
extern byte usbgen_rx_len;
```

Prototipos Públicos

```
void USBGenInitEP(void);
void USBGenWrite(byte *buffer, byte len);
byte USBGenRead(byte *buffer, byte len);
```

msd.h: Almacenamiento masivo

```
#ifndef MSD_H
#define MSD_H
```

Incluye

```
#include "system\typedefs.h"
#include "io_cfg.h"                    Mapa de pines E/S
```

Definiciones

Código de la clase de interfaz MSD:

```
#define MSD_INTF                       0x08
```

Código de la subclase de la interfas de clase MSD:

```
#define MSD_INTF_SUBCLASS              0x06
```

Código de protocolo de la clase de la interfaz MSD:

```
#define MSD_PROTOCOL                   0x50
```

Comandos de la clase:

```
#define MSD_RESET                      0xff
#define GET_MAX_LUN                    0xfe

#define BLOCKLEN_512                   0x0200

#define STMSDTRIS                      TRISD0
#define STRUNTRIS                      TRISD1
#define STMSDLED                       LATDbits.LATD0
#define STRUNLED                       LATDbits.LATD1
#define ToggleRUNLED()                 STRUNLED = !STRUNLED;
```

Set de commandos de código de la subclase transparente SCSI:

```
#define INQUIRY                                0x12
#define READ_FORMAT_CAPACITY                   0x23
#define READ_CAPACITY                          0x25
#define READ_10                                0x28
#define WRITE_10                               0x2A
#define REQUEST_SENSE                          0x03
#define MODE_SENSE                             0x1A
#define PREVENT_ALLOW_MEDIUM_REMOVAL           0x1E
#define TEST_UNIT_READY                        0x00
#define VERIFY                                 0x2F
#define STOP_START                             0x1B
```

Varios estados del Firmware de almacenamiento masivo:

```
#define MSD_WAIT        0          Esperando para un CBW válido
#define MSD_DATA_IN     2          Estado de datos IN (Dispositivo-> Host)
#define MSD_DATA_OUT    3          Estado de datos OUT (Host -> Device)
```

```
#define MSD_CSW_SIZE  0x0d      Datos CSW de 10 bytes CSW
#define MSD_CBW_SIZE  0x1f      Datos CSW de 31 bytes CBW

#define INVALID_CBW   1
#define VALID_CBW     !INVALID_CBW
#define MAX_LUN       0
```

Clave de los códigos de error Sense

```
#define S_NO_SENSE            0x0
#define S_RECOVERED_ERROR     0x1
#define S_NOT_READY           0x2
#define S_MEDIUM_ERROR        0x3
#define S_HARDWARE_ERROR      0X4
#define S_ILLEGAL_REQUEST     0x5
#define S_UNIT_ATTENTION      0x6
#define S_DATA_PROTECT        0x7
#define S_BLANK_CHECK         0x8
#define S_VENDOR_SPECIFIC     0x9
#define S_COPY_ABORTED        0xA
#define S_ABORTED_COMMAND     0xB
#define S_OBSOLETE            0xC
#define S_VOLUME_OVERFLOW     0xD
#define S_MISCOMPARE          0xE

#define S_CURRENT             0x70
#define S_DEFERRED            0x71
```

Códigos ASC ASCQ para datos (sólo el que vamos a utilizar)

Con una respuesta de clave sense ilegal de un comando no soportado:

```
#define ASC_INVALID_COMMAND_OPCODE 0x20
#define ASCQ_INVALID_COMMAND_OPCODE 0x00
```

De SPC-3 Tabla 185

Con una respuesta de clave sense ilegal para probar si la unidad está disponible:

```
#define ASC_LOGICAL_UNIT_NOT_SUPPORTED 0x25
#define ASCQ_LOGICAL_UNIT_NOT_SUPPORTED 0x00
```

Con una clave sense Not ready

```
#define ASC_LOGICAL_UNIT_DOES_NOT_RESPOND 0x05
#define ASCQ_LOGICAL_UNIT_DOES_NOT_RESPOND 0x00

#define ASC_MEDIUM_NOT_PRESENT 0x3a
#define ASCQ_MEDIUM_NOT_PRESENT 0x00

#define ASC_LOGICAL_UNIT_NOT_READY_CAUSE_NOT_REPORTABLE 0x04
#define ASCQ_LOGICAL_UNIT_NOT_READY_CAUSE_NOT_REPORTABLE 0x00

#define ASC_LOGICAL_UNIT_IN_PROCESS 0x04
```

```
#define ASCQ_LOGICAL_UNIT_IN_PROCESS 0x01

#define ASC_LOGICAL_UNIT_NOT_READY_INIT_REQD 0x04
#define ASCQ_LOGICAL_UNIT_NOT_READY_INIT_REQD 0x02

#define ASC_LOGICAL_UNIT_NOT_READY_INTERVENTION_REQD 0x04
#define ASCQ_LOGICAL_UNIT_NOT_READY_INTERVENTION_REQD 0x03

#define ASC_LOGICAL_UNIT_NOT_READY_FORMATTING 0x04
#define ASCQ_LOGICAL_UNIT_NOT_READY_FORMATTING 0x04

#define ASC_LOGICAL_BLOCK_ADDRESS_OUT_OF_RANGE 0x21
#define ASCQ_LOGICAL_BLOCK_ADDRESS_OUT_OF_RANGE 0x00

#define ASC_WRITE_PROTECTED 0x27
#define ASCQ_WRITE_PROTECTED 0x00
```

(bit) mMSDRxIsBusy(void)

Esta macro se utiliza para comprobar si el Endpoint MSD OUT está ocupado (controlado por el SIE) o no.

Uso típico: if(mMSDRxIsBusy())

```
#define mMSDRxIsBusy()          MSD_BD_OUT.Stat.UOWN
```

(bit) mMSDTxIsBusy(void)

Esta macro se utiliza para comprobar si el Endpoint MSD IN está ocupado (controlado por el SIE) o no.

Uso típico: if(mMSDTxIsBusy())

```
#define mMSDTxIsBusy()          MSD_BD_IN.Stat.UOWN
```

(bit) mMin(void)

Esta macro se utiliza para encontrar el menor de dos argumentos.

Uso típico: mMin(A, B)

```
#define mMin(A,B) (A<B)?A:B
```

Estructuras

```
typedef struct _USB_MSD_CBW                 31 bytes totales CBW
{
  dword dCBWSignature;                      55 53 42 43h
  dword dCBWTag;                            Enviado por el host, el dispositivo se hace eco
con el
                                            valor en CSW (asociado a CSW con CBW)
  dword dCBWDataTransferLength;             Número de bytes de datos que el host espera
transferir
  byte bCBWFlags;                           Flags CBW, bit 7 = 0 salida de datos del host-
                                            dispositivo; bit 7=1 dispositivo-host, el resto de
bits 0
  byte bCBWLUN;                             MS1bits son siempre cero, 0 en nuestro caso es
una
```

```
                                        sóla unidad lógica
  byte bCBWCBLength;                    MS3bits son cero
  byte CBWCB[16];                       Bloque de comando que ejecuta el dispositivo
} USB_MSD_CBW;

typedef struct {                        /Bloque de comando para leer 10 (0x28) y
escribir 10
                                        (0x2a) comandos
  byte Opcode;
  byte Flags;                           b7-b5 lectura protegida, b4 DPO, b3 FUA,
                                        b2 Reservado, b1 FUA_NV, b0 Obsoleto
  DWORD LBA;
byte GroupNumber;                       b4-b0 es el número de grupo el resto reservados
  WORD TransferLength;
  byte Control;
} ReadWriteCB;

typedef struct {                        Formato del comando Inquiry
  byte Opcode;
  byte EVPD;                            sólo b0  enable vital product data
  byte PageCode;
  word AllocationLength;
  byte Control;
} InquiryCB;

typedef struct {                        capacidad de lectura 10
  byte Opcode;
  byte Reserved1;
  dword LBA;                            Bloque de dirección lógico
  word Reserved2;
  byte PMI;                             Partial medium Indicator sólo b0
  byte Control;
} ReadCapacityCB;

typedef struct {                        Respuesta Sense 0x03
  byte Opcode;
  byte Desc;
  word Reserved;
  byte AllocationLength;
  byte Control;
} RequestSenseCB;

typedef struct {                        Modo Sense 0x1A
  byte Opcode;
  byte DBD;                             Actualmente sólo se utiliza b3 como bloque
                                        descriptor desactivado
  byte PageCode;                        b7,b6 PC=página de control, b5-b0 página de
código
                                        Página de Control bits 00=> valor actual,
                                        01=>valores modificables,10=>valor por defecto,
                                        11=>valores guardados
```

```
  byte SubPageCode;
  byte AllocationLength;
  byte Control;
} ModeSenseCB;

typedef struct {                          Prevenir el permiso de retirada del medio 0x1E
  byte Opcode;
  byte Reserved[3];
  byte Prevent;                           Sólo se previenen b1-b0, el resto reservados
  byte Control;
} PreventAllowMediumRemovalCB;

typedef struct {                          Unidad de prueba disponible 0x00
  byte Opcode;
  dword Reserved;
  byte Control;
} TestUnitReadyCB;

typedef struct {                          Verificar 10 Comando 0x2F
  byte Opcode;
  byte VRProtect;                         b7-b5 VRProtect, b4 DPO, b3-b2,Reservado,
                                          b1 BYTCHK, b0 Obsoleto
  dword LBA;
  byte GroupNumber;                       Número del grupo b4-b0, el resto reservado
  word VerificationLength;
  byte Control;
} VerifyCB;

typedef struct {                          STOP_START 0x1B
  byte Opcode;
  byte Immed;
  word Reserved;
  byte Start;                             b7-b4 Condición de energía, b3-b2 reservedo,
                                          b1 LOEJ, b0 Start
  byte Control;
} StopStartCB;

typedef struct _USB_MSD_CSW               CSW
{
  dword dCSWSignature;                    55 53 42 53h firma del paquete de CSW
  dword dCSWTag;                          eco dCBWTag del paquete CBW
  dword dCSWDataResidue;                  diferencia en los datos esperados
                                          (dCBWDataTransferLength)
                                          y la cantidad actual procesada/enviada
  byte bCSWStatus;                        00h Comando aprobado, 01h Comando Fallido,
                                          02h Error de fase, el resto obsoleto/reservado
} USB_MSD_CSW;

typedef struct
{
```

```
  byte Peripheral;                  Clasificador del periférico:3;
Periférico_DevType:5;
  byte Removble;                    medio removible bit7 = 0 medio no removible,
                                    resto reservado
  byte Version;                     versión
  byte Response_Data_Format;        b7,b6 Obsoleto, b5 Acceso de control
coordinado,
                                    b4 direccionamiento jerárquico soportado
                                    b3:0 respuesta al formato de datos 2 indica que la
                                    respuesta esta en el formato definido por spec
  byte AdditionalLength;            longitud en bytes de los datos que quedan de la
                                    indagación estándar
  byte Sccstp;                      b7 SCCS, b6 ACC, b5-b4 TGPS, b3 3PC,
                                    b2-b1 Reservado, b0 Protegido
  byte bqueetc;                     b7 bque, b6- EncServ, b5-VS, b4-MultiP,
                                    b3-MChngr, b2-b1 Obsoleto, b0-Addr16
  byte CmdQue;                      b7-b6 Obsoleto, b5-WBUS, b4-Sync, b3-Linked,
                                    b2 Obsoleto, b1 Cmdque, b0-VS
  char vendorID[8];
  char productID[16];
  char productRev[4];
} InquiryResponse;

typedef struct {
  byte ModeDataLen;
  byte MediumType;
  unsigned Resv:4;
  unsigned DPOFUA:1;                0 indica que no soporta los bits DPO y FUA
  unsigned notused:2;
  unsigned WP:1;                    0 indica que no protege la escritura
  byte BlockDscLen;                 Longitud del Bloque Descriptor
} tModeParamHdr;
```

Modo corto del bloque descriptor LBA (ver Página 1009, SBC-2)

```
typedef struct {
  byte NumBlocks[4];
  byte Resv;                        reservado
  byte BlockLen[3];
} tBlockDescriptor;

/* Page_0 mode page format */
typedef struct {

  unsigned PageCode:6;              SPC-3 7.4.5
  unsigned SPF:1;                   SubPageFormat=0 medio Page_0 formato
  unsigned PS:1;                    Parámetros salvables

  byte PageLength;                  si 2..n bytes del modo parámetro PageLength = n-
1
  byte ModeParam[];                 modo parámetros
} tModePage;
```

```
typedef struct {
  tModeParamHdr Header;
  tBlockDescriptor BlockDsc;
  tModePage modePage;
} ModeSenseResponse;
```

Formato fijado si el bit Desc de la respuesta sense CBW es 0

```
typedef union {
  struct
  {
    byte _byte[18];
  };
  struct {
  unsigned ResponseCode:7;          b6-b0 es el código de respuesta fijado o el formato
                                    del descriptor
  unsigned VALID:1;                 Poner a 1 indica que el campo de información tiene
                                    un valor válido
  byte Obsolete;
  unsigned SenseKey:4;              Referencia SPC-3 Sección 4.5.6
  unsigned Resv:1;
  unsigned ILI:1;                   Indicador de la longitud incorrecta
  unsigned EOM:1;                   Fin del medio
  unsigned FILEMARK:1;              para los comandos READ y SPACE

  DWORD Information;                Tipo de dispositivo o comando específico
                                    (SPC-33.1.18)
  byte AddSenseLen;                 Número de bytes sense adicionales que siguen
<=244
  DWORD CmdSpecificInfo;            depende del comando de la excepción ocurrida
  byte ASC;                         código sense adicional
  byte ASCQ;                        código sense adicional sección clasificada
                                    4.5.2.1 SPC-3
  byte FRUC;                        Campo reemplazable código de unidad 4.5.2.5
SPC-3

  byte SenseKeySpecific[3];         SKSV son los msb de la clave sense específica de
                                    campo válido fijado=>SKS válido. Los 18-n
bytes
                                    sense adicionales se pueden definir más tarde
18bytes
                                    de respuesta sense de formato fijado
  };
} RequestSenseResponse;
```

Externas

```
extern CSD gblCSDReg;
```

declarado en sdcard.c

Prototipos públicos

```
void USBCheckMSDRequest(void);
```

```
void ProcessIO(void);
void SDCardInit(void);
void MSDInitEP(void);
```

cdc.h: Dispositivos de comunicación

```
#ifndef CDC_H
#define CDC_H
```

Incluye

```
#include "system\typedefs.h"
```

Definiciones

Repuesta de clase específica:

```
#define SEND_ENCAPSULATED_COMMAND       0x00
#define GET_ENCAPSULATED_RESPONSE       0x01
#define SET_COMM_FEATURE                0x02
#define GET_COMM_FEATURE                0x03
#define CLEAR_COMM_FEATURE              0x04
#define SET_LINE_CODING                 0x20
#define GET_LINE_CODING                 0x21
#define SET_CONTROL_LINE_STATE          0x22
#define SEND_BREAK                      0x23
```

Notificaciones

Nota: las notificaciones se obtienen de la interface de comunicación (Endpoint Interrupción)

```
#define NETWORK_CONNECTION              0x00
#define RESPONSE_AVAILABLE              0x01
#define SERIAL_STATE                    0x20
```

Código de la clase del dispositivo:

```
#define CDC_DEVICE                      0x02
```

Código de la clase de la interfaz de comunicación

```
#define COMM_INTF                       0x02
```

Código de la subclase de la interfaz de comunicación

```
#define ABSTRACT_CONTROL_MODEL          0x02
```

Código del protocolo de control de la clase de la interfaz de comunicación

```
#define V25TER                  0x01    Comandos comunes AT ("Hayes(TM)")
```

Código de la clase de la interfaz de datos

```
#define DATA_INTF                       0x0A
```

Código del protocolo de la clase de la interfaz de datos

```
#define NO_PROTOCOL             0x00    No necesita un protrocolo de clase específico
```

Código selector de las características de la comunicación

```
#define ABSTRACT_STATE                  0x01
#define COUNTRY_SETTING                 0x02
```

Descriptores funcionales

Tipos de valor del campo bDscType

```
#define CS_INTERFACE                    0x24
#define CS_ENDPOINT                     0x25
```

bDscSubType en descriptores funcionales

```
#define DSC_FN_HEADER                   0x00
#define DSC_FN_CALL_MGT                 0x01
#define DSC_FN_ACM                      0x02    ACM – Administración de control
abstracta
#define DSC_FN_DLM                      0x03    DLM – Dirección de línea directa
#define DSC_FN_TELEPHONE_RINGER         0x04
#define DSC_FN_RPT_CAPABILITIES         0x05
#define DSC_FN_UNION                    0x06
#define DSC_FN_COUNTRY_SELECTION        0x07
#define DSC_FN_TEL_OP_MODES             0x08
#define DSC_FN_USB_TERMINAL             0x09
```

Estados de tranferencia CDC Bulk IN

```
#define CDC_TX_READY                    0
#define CDC_TX_BUSY                     1
#define CDC_TX_BUSY_ZLP                 2       ZLP: Paquete de longitud cero
#define CDC_TX_COMPLETING               3
```

BOOL mUSBUSARTIsTxTrfReady(void)

Esta macro se utiliza para comprobar si la clase CDC está disponible para enviar mas datos.

Uso típico: if(mUSBUSARTIsTxTrfReady())

```
#define mUSBUSARTIsTxTrfReady()     (cdc_trf_state == CDC_TX_READY)
```

(bit) mCDCUsartRxIsBusy(void)

Esta macro se utiliza para comprobar si el Endpoint CDC Bulk OUT está ocupado (controlado por el SIE) o no.

Uso típico: if(mCDCUsartRxIsBusy())

```
#define mCDCUsartRxIsBusy()         CDC_BULK_BD_OUT.Stat.UOWN
```

(bit) mCDCUsartTxIsBusy(void)

Esta macro se utiliza para comprobar si el Endpoint CDC Bulk IN está ocupado (controlado por el SIE) o no.

Uso típico: if(mCDCUsartTxIsBusy())

```
#define mCDCUsartTxIsBusy()         CDC_BULK_BD_IN.Stat.UOWN
```

byte mCDCGetRxLength(void)

Salida: devuelve cdc_rx_len

mCDCGetRxLength se utiliza para recuperar el número de bytes que se han copiado al buffer del usuario en la última llamada a la función getsUSBUSART.

```
#define mCDCGetRxLength()           cdc_rx_len
```

void mUSBUSARTTxRam(byte *pData, byte len)

Precondición: cdc_trf_state tiene que estar en el estado CDC_TX_READY.
El valor de 'len' tiene que se igual o menor de 255bytes.

Entrada: *pDdata:* Puntero al comienzo de la localización de los bytes de datos.
len: número de bytes que se van a transferir.

Esta macro se utiliza para transferir datos localizados en la memoria de datos.

Utilizar esta macro cuando:

1. La longitud de la transferencia se conoce
2. Los datos no terminan con uno nulo

Nota: Esta macro sólo manipula la transferencia setup. La transferencia actual la manipula CDCTxService().

```
#define mUSBUSARTTxRam(pData,len)
{
  pCDCSrc.bRam = pData;
  cdc_tx_len = len;
  cdc_mem_type = _RAM;
  cdc_trf_state = CDC_TX_BUSY;
}
```

void mUSBUSARTTxRom(rom byte *pData, byte len)

Precondición: cdc_trf_state tiene que estar en el estado CDC_TX_READY.
El valor de 'len' tiene que se igual o menor de 255bytes.

Entrada: *pDdata:* Puntero al comienzo de la localización de los bytes de datos.
len: número de bytes que se van a transferir.

Esta macro se utiliza para transferir datos localizados en la memoria de programa.

Utilizar esta macro cuando:

3. La longitud de la transferencia se conoce
4. Los datos no terminan con uno nulo

Nota: Esta macro sólo manipula la transferencia setup. La transferencia actual la manipula CDCTxService().

```
#define mUSBUSARTTxRom(pData,len)
{
  pCDCSrc.bRom = pData;
  cdc_tx_len = len;
  cdc_mem_type = _ROM;
  cdc_trf_state = CDC_TX_BUSY;
}
```

Estructuras

Estructura de la línea de codificación

```
#define LINE_CODING_LENGTH              0x07

typedef union _LINE_CODING
{
  struct
  {
  byte _byte[LINE_CODING_LENGTH];
```

```
  };
  struct
  {
  DWORD                                 dwDTERate;     Estructura de datos compleja
  byte                                  bCharFormat;
  byte                                  bParityType;
  byte                                  bDataBits;
  };
} LINE_CODING;

typedef union _CONTROL_SIGNAL_BITMAP
{
  byte _byte;
  struct
  {
  unsigned DTE_PRESENT;                 [0] No Presente [1] Presente
  unsigned CARRIER_CONTROL;             [0] Desactiva [1] Activa
  };
} CONTROL_SIGNAL_BITMAP;
```

Descriptor de cabecera funcional

```
typedef struct _USB_CDC_HEADER_FN_DSC
{
  byte bFNLength;
  byte bDscType;
  byte bDscSubType;
  word bcdCDC;
} USB_CDC_HEADER_FN_DSC;
```

Descriptor funcional de dirección de control abstracto

```
typedef struct _USB_CDC_ACM_FN_DSC
{
  byte bFNLength;
  byte bDscType;
  byte bDscSubType;
  byte bmCapabilities;
} USB_CDC_ACM_FN_DSC;
```

Descriptor funcional de unión

```
typedef struct _USB_CDC_UNION_FN_DSC
{
  byte bFNLength;
  byte bDscType;
  byte bDscSubType;
  byte bMasterIntf;
  byte bSaveIntf0;
} USB_CDC_UNION_FN_DSC;
```

Descriptor funcional de control de llamadas

```
typedef struct _USB_CDC_CALL_MGT_FN_DSC
{
```

```
    byte bFNLength;
    byte bDscType;
    byte bDscSubType;
    byte bmCapabilities;
    byte bDataInterface;
} USB_CDC_CALL_MGT_FN_DSC;
```

Externas

```
extern byte cdc_rx_len;

extern byte cdc_trf_state;
extern POINTER pCDCSrc;
extern byte cdc_tx_len;
extern byte cdc_mem_type;
```

Prototipos publicos

```
void USBCheckCDCRequest(void);
void CDCInitEP(void);
byte getsUSBUSART(char *buffer, byte len);
void putrsUSBUSART(const rom char *data);
void putsUSBUSART(char *data);
void CDCTxService(void);
#endif              CDC_H
```

hid.h: Dispositivo interfaz con humanos

Incluye

```
#include "system\typedefs.h"
```

Definiciones

Repuestas de clase específicas:

```
#define GET_REPORT          0x01    Obtener informe
#define GET_IDLE            0x02    Obtener reposo
#define GET_PROTOCOL        0x03    Obtener protocolo
#define SET_REPORT          0x09    Fijar informe
#define SET_IDLE            0x0A    Fijar reposo
#define SET_PROTOCOL        0x0B    Fijar protocolo
```

Tipos de clase de descriptor:

```
#define DSC_HID             0x21    Descriptor HID
#define DSC_RPT             0x22    Descriptor informe
#define DSC_PHY             0x23
```

Selección de protocolo:

```
#define BOOT_PROTOCOL       0x00    Protocolo de inicio
#define RPT_PROTOCOL        0x01    Informe de protocolo
```

Código de clase de interfaz HID:

```
#define HID_INTF            0x03    Interfaz HID
```

Código de subclase de interfaz HID:

```
#define BOOT_INTF_SUBCLASS  0x01
```

Código de protocolo de clase de interfaz:

```
#define HID_PROTOCOL_NONE       0x00    Ninguno
#define HID_PROTOCOL_KEYBOAD    0x01    Teclado
#define HID_PROTOCOL_MOUSE      0x02    Ratón
```

(bit) mHIDRxIsBusy()

Esta macro comprueba si el Endpoint de salida del HID está ocupado (controlado por el SIE) o no.

Aplicación típica: `if(mHIDRxIsBusy())`

```
#define mHIDRxIsBusy()      HID_BD_OUT.Stat.UOWN
```

(bit) mHIDTxIsBusy()

Esta macro comprueba si el Endpoint de entrada del HID está coupado (controlado por el SIE) o no.

Aplicación típica: `if(mHIDTxIsBusy())`

```
#define mHIDTxIsBusy()      HID_BD_IN.Stat.UOWN
```

byte mHIDGetRptRxLength()

Salida: mHIDGetRptRxLength devuelve un informe de la longitud del receptor HID (hid_rpt_rx_len).

La mHIDGetRptRxLength se utiliza para recuperar el número de bytes copiándolos al buffer de usuario en orden de la última llamada a la función HIDRxReport.

```
#define mHIDGetRptRxLength()    hid_rpt_rx_len
```

Estructuras

```
typedef struct _USB_HID_DSC_HEADER          Cabecera del descriptor HID
{
  byte bDscType;                Tipo de descriptor
  word wDscLength;              Longitud del descriptor
} USB_HID_DSC_HEADER;

typedef struct _USB_HID_DSC                 Descriptor HID
{
  Longitud                      Tipo de descriptor          Bcd del HID
  byte bLength;                 byte bDscType;              word bcdHID;
  Código de territorio          Número del descriptor
  byte bCountryCode;            byte bNumDsc;
  USB_HID_DSC_HEADER            hid_dsc_header[HID_NUM_OF_DSC];
                                HID_NUM_OF_DSC se define en autofiles\usbcfg.h
} USB_HID_DSC;
```

Externas

```
extern byte hid_rpt_rx_len;
```

Prototipos públicos

```
void HIDInitEP(void);
void USBCheckHIDRequest(void);
void HIDTxReport(char *buffer, byte len);
byte HIDRxReport(char *buffer, byte len);
```

io_cfg.h

Incluye

```
#include "autofiles\usbcfg.h"
```

Tris

```
#define INPUT_PIN               1
#define OUTPUT_PIN              0
```

USB

```
#define tris_usb_bus_sense  TRISAbits.TRISA1        entrada

#if defined(USE_USB_BUS_SENSE_IO)
#define usb_bus_sense           PORTAbits.RA1
#else
#define usb_bus_sense           1
#endif

#define tris_self_power         TRISAbits.TRISA2    entrada

#if defined(USE_SELF_POWER_SENSE_IO)
#define self_power              PORTAbits.RA2
#else
#define self_power              1
#endif
```

Interfaz del transmisor externo

```
#define tris_usb_vpo            TRISBbits.TRISB3    Salida
#define tris_usb_vmo            TRISBbits.TRISB2    Salida
#define tris_usb_rcv            TRISAbits.TRISA4    Entrada
#define tris_usb_vp             TRISCbits.TRISC5    Entrada
#define tris_usb_vm             TRISCbits.TRISC4    Entrada
#define tris_usb_oe             TRISCbits.TRISC1    Salida

#define tris_usb_suspnd         TRISAbits.TRISA3    Salida
```

LED

```
#define mInitAllLEDs()          LATD &= 0xF0; TRISD &= 0xF0;

#define mLED_1                  LATDbits.LATD0
#define mLED_2                  LATDbits.LATD1
#define mLED_3                  LATDbits.LATD2
#define mLED_4                  LATDbits.LATD3

#define mLED_1_On()             mLED_1 = 1;
#define mLED_2_On()             mLED_2 = 1;
#define mLED_3_On()             mLED_3 = 1;
#define mLED_4_On()             mLED_4 = 1;
```

```
#define mLED_1_Off()            mLED_1 = 0;
#define mLED_2_Off()            mLED_2 = 0;
#define mLED_3_Off()            mLED_3 = 0;
#define mLED_4_Off()            mLED_4 = 0;

#define mLED_1_Toggle()         mLED_1 = !mLED_1;
#define mLED_2_Toggle()         mLED_2 = !mLED_2;
#define mLED_3_Toggle()         mLED_3 = !mLED_3;
#define mLED_4_Toggle()         mLED_4 = !mLED_4;
```

Interruptores

```
#define mInitAllSwitches()      TRISBbits.TRISB4=1;TRISBbits.TRISB5=1;
#define mInitSwitch2()          TRISBbits.TRISB4=1;
#define mInitSwitch3()          TRISBbits.TRISB5=1;
//#define sw2                   PORTBbits.RB4
#define sw3                     PORTBbits.RB5
```

Potenciómetro

```
#define mInitPOT()
TRISAbits.TRISA0=1;ADCON0=0x01;ADCON2=0x3C;
```

SPI: Líneas de Chip Select

```
#define tris_cs_temp_sensor TRISBbits.TRISB2        Salida
#define cs_temp_sensor          LATBbits.LATB2

#define tris_cs_sdmmc           TRISBbits.TRISB3    Salida
#define cs_sdmmc                LATBbits.LATB3
```

SDMMC

```
#define TRIS_CARD_DETECT        TRISBbits.TRISB4    Entrada
#define CARD_DETECT             PORTBbits.RB4

#define TRIS_WRITE_DETECT       TRISAbits.TRISA4    Entrada
#define WRITE_DETECT            PORTAbits.RA4
```

interrupt.h

Incluye

```
#include "system\typedefs.h"
```

Definiciones

```
#define mEnableInterrupt()   INTCONbits.GIE = 1;
```

Prototipos

```
void low_isr(void);
void high_isr(void);
```

usb_compile_time_validation.h: validación del tiempo de compilado

Incluye

```
#include "system\typedefs.h"
#include "system\usb\usb.h"
```

Validación del USB

```
#if   (EP0_BUFF_SIZE != 8) && (EP0_BUFF_SIZE != 16) &&
      (EP0_BUFF_SIZE != 32) && (EP0_BUFF_SIZE != 64)
#error(Tamaño del buffer del Endpoint 0 incorrecto, comprueba "autofiles\usbcfg.h")
#endif

#if defined(HID_INT_OUT_EP_SIZE)
  #if (HID_INT_OUT_EP_SIZE > 64)
      #error(El tamaño del Endpoint de salida de HID no puede ser mayor de 64,
comprueba                                                    "autofiles\usbcfg.h")
  #endif
#endif

#ifdef HID_INT_IN_EP_SIZE
  #if (HID_INT_IN_EP_SIZE > 64)
      #error(El tamaño del Endpoint de entrada de HID no puede ser mayor de 64,
comprueba                                                    "autofiles\usbcfg.h")
  #endif
#endif
```