■ USB 関連 PIC API 関数

1. 各クラス共用

以下は、usb_device.h の抜粋です。

関数名	機能
USBDeviceTasks( )	void USBDeviceTasks(void) USB デバイス側の状態を制御する重要な関数で、周期的にスタックと送受信データをやりとりする。この関数は列挙プロセス(デバイス検出)においては、願わくば 100μsec 毎に定期的に呼び出されるべきである。列挙プロセス後も定期的にこの関数を呼び出す必要がある。所要の呼出間隔は種々の状況によるが長くてもよい。この関数は PC からの送信データより速くに呼び出される必要がある。 This function is the main state machine of the USB device side stack.This function should be called periodically to receive and transmit. packets through the stack. This function should be called preferably once every 100us during the enumeration process. After the enumeration process this function still needs to be called periodically to respond to various situations on the bus but is more relaxed in its time requirements. This function should also be called at least as fast as the OUT data expected from the PC.
USBDeviceInit( )	void USBDeviceInit(void) この関数はデバイスをデフォルト状態に初期化する。USB モジュールは内部変数、レジスタ、フラグ等完全にリセットされる。 This function initializes the device stack it in the default state. The USB module will be completely

	reset including all of the internal variables, registers, and interrupt flags.
USBGetRemoteWakeupStatus( )	BOOL USBGetRemoteWakeupStatus(void) この関数によりホストによるリモートウェークアップが可能か否かをしることができる。 戻り値(BOOL): TRUE ー 可 FLASE ー 否 This function indicates if remote wakeup has been enabled by the host. Devices that support remote wakeup should use this function to determine if it should send a remote wakeup.
USBGetDeviceState( )	BYTE USBGetDeviceState(void) USB の状態を返す関数である。この関数はデバイスが通信可能な状態がどうか調べる関数である。アプリケーションはこの関数がCONFIGURED_STATE.を返すまで送受信をしてはならない。 戻り値: DETACHED_STATE - The device is not attached to the bus ATTACHED_STATE - The device is attached to the bus but POWERED_STATE - The device is not officially in the powered state DEFAULT_STATE - The device has received a RESET from the host ADR_PENDING_STATE - The device has received the SET_ADDRESS command buthasn't received the STATUS stage of the command soit is still

	operating on address 0. ADDRESS_STATE – The device has an address assigned but has not received a SET_CONFIGURATION command yet or has received a SET_CONFIGURATION with a configuration number of 0 (deconfigured) CONFIGURED_STATE – the device has received a non¥–zero SET_CONFIGURATION command is now ready for communication on the bus. This function returns the current state of the device on the USB. This function is used to determine when the device is ready to communicate on the bus. Applications should not try to send or receive data until this function returns CONFIGURED_STATE.
USBGetSuspendState( )	BOOL USBGetSuspendState(void) この関数はデバイスがサスペンド状態にあるかどうかを調べる関数である。デバイスがサスペンド状態の場合、デバイスはデータを返信することができない。 戻り値 TRUE – サスペンド状態. FALSE – 非サスペンド状態. This function indicates if this device is currently suspended. When a device is suspended it will not be able to transfer data over the bus. This function can be used by the application to skip over section of code that do not need to exectute if the device is unable to send data over the bus.
USBCBSuspend( )	void USBCBSuspend(void)

	USB のサスペンド状態が検出された時呼出しが行なわれるコールバック関数 Call back that is invoked when a USB suspend is detected.
USBCBWakeFromSuspend( )	void USBCBWakeFromSuspend(void) この関数はデバイスがサスペンド状態からウェークアップした時、呼び出しが行なわれるコールバック関数である。 This call back is invoked when a wakeup from USB suspend is detected.
USBCB_SOF_Handler( )	void USBCB_SOF_Handler(void) ホストからの SOF（Start of Frame)を受信した時に、呼び出しが行なわれるコールバック関数である。（オプション） This callback is called when a SOF packet is received by the host. (optional)
USBCBErrorHandler( )	void USBCBErrorHandler(void) USB エラーが発生した時に、呼び出しが行なわれるコールバック関数である。（オプション） This callback is called whenever a USB error occurs. (optional)
USBCBCheckOtherReq( )	void USBCBCheckOtherReq(void) スタックが処理できない内容のリクエストがエンドポイント0（コントロールエンドポイント）にきた時に、呼び出されるコールバック関数 This function is called whenever a request comes over endpoint 0 (thecontrol endpoint) that the stack does not know how to handle.
USBCBStdSetDscHandler( )	void USBCBStdSetDscHandler(void)

	SET_DESCRIPTOR リクエストがきた時、呼び出されるコールバック関数 This callback is called when a SET_DESCRIPTOR request is received (optional)
USBCBInitEP( )	void USBCBInitEP(void) SET_CONFIGURATION がきた時に、呼び出されるコールバック関数. This function is called whenever the device receives a SET_CONFIGURATION request.
USBCBSendResume( )	void USBCBSendResume(void) リモートウェークアップを初期化した時、呼び出されるコールバック関数 This function should be called to initiate a remote wakeup. (optional)
USBCBEP0DataReceived( )	void USBCBEP0DataReceived(void) EP0 データパケットを受信した時、呼び出されるコールバック関数 This function is called whenever a EP0 data packet is received. (optional)
USBHandleBusy( )	BOOL USBHandleBusy(USB_HANDLE handle) 戻り値I TRUE ー ビジー状態 FALSE ー 非ビジー状態 インプットハンドルがビジーかどうか調べる関数。ビジーでないなら前回の送信は完了したことになる。 Checks to see if the input handle is busy. if the handle is no longer busy then the last transmission is complete

USBHandleGetLength( )	WORD USBHandleGetLength(USB_HANDLE handle) インプットハンドルのデスティネーションバッファーの長さを修復する関数 Retrieves the length of the destination buffer of the input handle
USBHandleGetAddr( )	WORD USBHandleGetAddr(USB_HANDLE) インプットハンドルのデスティネーションバッファーのアドレスを修復する関数 Retrieves the address of the destination buffer of the input handle
USBEP0SetSourceRAM( )	void USBEP0SetSourceRAM(BYTE* src) コントロールエンドポイントにデータのアドレスをセットする関数 Sets the address of the data to send over the control endpoint
USBEP0SetSourceROM( )	void USBEP0SetSourceROM(BYTE* src) コントロールエンドポイントにデータのアドレスをセットする関数 Sets the address of the data to send over the control endpoint
USBEP0Transmit( )	void USBEP0Transmit(BYTE options) コントロールエンドポイントにデータのアドレスをセットする関数 Sets the address of the data to send over the control endpoint
void USBEP0SetSize( )	void USBEP0SetSize(WORD size) コントロールエンドポイントにデータのサイズをセッ

	トする関数 Sets the size of the data to send over the control endpoint
USBEP0SendRAMPtr( )	void USBEP0SendRAMPtr(BYTE* src, WORD size, BYTE Options) RAM から送信したいデータのサイズやオプションをセットする関数 Sets the source, size, and options of the data you wish to send from a RAM source
USBEP0SendROMPtr( )	void USBEP0SendROMPtr(BYTE* src, WORD size, BYTE Options) ROM から送信したいデータのサイズやオプションをセットする関数 Sets the source, size, and options of the data you wish to send from a ROM source
USBTxOnePacket( )	USB_HANDLE USBTxOnePacket(BYTE ep, BYTE* data, WORD len) 明確化されたされたデータを明確化されたエンドポイントに送信する関数 Sends the specified data out the specified endpoint
USBRxOnePacket( )	void USBRxOnePacket(BYTE ep, BYTE* data, WORD len) 明確化されたされたデータを明確化されたエンドポイントに受信する関数 Receives the specified data out the specified endpoint
USBClearInterruptFlag( )	void USBClearInterruptFlag(WORD reg, BYTE flag) 明確化された割り込みフラグをクリアする関数 Clears the specified interrupt flag

USBClearInterruptRegister( )	void USBClearInterruptRegister(WORD reg) 明確化された割り込みレジスタをクリアする関数 Clears the specified interrupt register
USBStallEndpoint( )	void USBStallEndpoint(BYTE ep, BYTE dir) 明確化されたエンドポイントを停止する関数 STALLs the specified endpoint
USBStallEndpoint( );	void USBStallEndpoint(BYTE ep, BYTE dir);
USBSoftDetach( );	void USBSoftDetach(void);
USBCtrlEPService( );	void USBCtrlEPService(void);
USBCtrlTrfSetupHandler( );	void USBCtrlTrfSetupHandler(void);
USBCtrlTrfInHandler( );	void USBCtrlTrfInHandler(void);
USBCheckStdRequest( );	void USBCheckStdRequest(void);
USBStdGetDscHandler( );	void USBStdGetDscHandler(void);
USBCtrlEPServiceComplete( );	void USBCtrlEPServiceComplete(void);
USBCtrlTrfTxService( );	void USBCtrlTrfTxService(void);
USBPrepareForNextSetupTrf( );	void USBPrepareForNextSetupTrf(void);
USBCtrlTrfRxService( );	void USBCtrlTrfRxService(void);
USBStdSetCfgHandler( );	void USBStdSetCfgHandler(void);
USBStdGetStatusHandler( );	void USBStdGetStatusHandler(void);
USBStdFeatureReqHandler( );	void USBStdFeatureReqHandler(void);
USBCtrlTrfOutHandler(void);	void USBCtrlTrfOutHandler(void);
USBIsTxBusy( );	BOOL USBIsTxBusy(BYTE EPNumber);
USBPut( );	void USBPut(BYTE EPNum, BYTE Data);
USBEPService( );	void USBEPService(void);
USBConfigureEndpoint( );	void USBConfigureEndpoint(BYTE EPNum, BYTE direction);
USBProtocolResetHandler( );	void USBProtocolResetHandler(void);
USBWakeFromSuspend( );	void USBWakeFromSuspend(void);

USBSuspend( );	void USBSuspend(void);
USBStallHandler( );	void USBStallHandler(void);
USBTransferOnePacket( );	volatile USB_HANDLE USBTransferOnePacket(BYTE ep, BYTE dir, BYTE* data, BYTE len);
USBEnableEndpoint( );	void USBEnableEndpoint(BYTE ep,BYTE options);
USBInitEP( );	void USBInitEP(BYTE ROM* pConfig);
USBCBEP0DataReceived( );	void USBCBEP0DataReceived(void);

2. CDC クラス用

関数名	機能
getsUSBUSART( )	USB CDC Bulk OUT エンドポイントから指定されたバイト数の文字列を取得する関数 BYTE getsUSBUSART(char *buffer, //コピーを取得するデータ列のポインタ BYTE len //取得する文字列の長さ(バイト数)); 戻り値: 取得した文字数(バイト数) Summary: getsUSBUSART copies a string of BYTEs received through USB CDC Bulk OUT endpoint to a user's specified location. It is a non-blocking function. It does not wait for data if there is no data available. Instead it returns '0' to notify the caller that there is no data available.
putUSBUSART( )	データ列を指定された長さだけ USB に書き込む関数。 0x00 もデータとしてデータ列に書き込める。

	void putUSBUSART(
	  char *data, //書き込むデータ列のポインタ
	  BYTE length //書き込むデータの長さ(バイト数)
	)
	
	戻り値: なし
	
	Summary:
	putUSBUSART writes an array of data to the USB. Use this version, is capable of transfering 0x00 (what is typically a NULL character in any of the string transfer functions).

putsUSBUSART( )	RAM にある文字列を USB に書き込む関数。終端の 0x00 も USB に書き込まれる void putsUSBUSART(char *data //書き込む文字列のポインタ) 戻り値: なし Summary: putsUSBUSART writes a string of data to the USB including the null character. Use this version, 'puts', to transfer data from a RAM buffer.
putrsUSBUSART( )	ROM にある文字列を USB に書き込む関数。終端の 0x00 も USB に書き込まれる。 void putrsUSBUSART(const ROM char *data //書き込む文字列のポインタ) Summary: putrsUSBUSART writes a string of data to the USB including the null character. Use this version, 'putrs', to transfer data literals and data located in program memory.

USBCheckCDCRequest( )	データパケットをルーチンで監視する関数。 usb_function_cdc.c の実際に実行される処理が記述されている。 void USBCheckCDCRequest(void //引数なし) Summary: This routine checks the setup data packet to see if it knows how to handle it
CDCInitEP( )	CDC ドライバーを初期化する関数。この関数により通信条件やエンドポイントが初期化される。 void CDCInitEP(void //引数なし) Summary: This function initializes the CDC function driver. This function should be called after the SET_CONFIGURATION command. Description: This function initializes the CDC function driver. This function sets the default line coding (baud rate, bit parity, number of data bits,and format). This function also enables the endpoints and prepares for the first transfer from the host.This function should be called after the SET_CONFIGURATION command. This is most simply done by calling this function from theUSBCBInitEP() function.
CDCTxService( )	ホストとデバイスに係る処理関数。この関数はデバイスのコンフィグレーション確定に際し1度だけ呼ばれる。 void CDCTxService(void //引数なし)

	Summary: CDCTxService handles device-to-host transaction(s). This function should be called once per Main Program loop after the device reaches the configured state.

備考　各関数の技術情報は下記に詳しく記載されています。

① usb_function_cdc.c

② AN956 Migrating Applications to USB from RS-232 UART with Minimal Impact on PC Software

3. 汎用(Geric)クラス用

以下は、usb_function_generic.h の抜粋です。

関数名	機能
mUSBGenRxIsBusy( )	(bit) mUSBGenRxIsBusy(void) 戻り値(bit)： ビジーの場合： 1 ビジーでない場合： 0 SIE の OUT エンドポイントがビジーかどうかの判別をおこなうマクロ This macro is used to check if the OUT endpoint is busy (owned by SIE) or not. Typical Usage: if(mUSBGenRxIsBusy())
mUSBGenTxIsBusy( )	(bit) mUSBGenTxIsBusy(void) 戻り値(bit)： ビジーの場合： 1 ビジーでない場合： 0 SIE の IN エンドポイントがビジーかどうかの判別をおこなう

	マクロ This macro is used to check if the IN endpoint is busy (owned by SIE) or not. Typical Usage: if(mUSBGenTxIsBusy())
mUSBGenGetRxLength( )	byte mUSBGenGetRxLength(void) 戻り値(byte): 直近の USBGenRead ( )関数により、読み込んだバイト数 直近の USBGenRead ( )関数により、読み込んだバイト数を返す mUSBGenGetRxLength is used to retrieve the number of bytes copied to user's buffer by the most recent call to USBGenRead function. Return Values: mUSBGenGetRxLength returns usbgen_rx_len
USBGenWrite( )	USB_HANDLE USBGenWrite(BYTE ep, //エンドポイント名 BYTE* data, //送信データ WORD len //送信データの長さ) 戻り値: 送信ハンドル (注) 送信データをエンドポイントに書き込む Sends the specified data out the specified endpoint This function sends the specified data out the specified endpoint and returns a handle to the transfer information. ep – the endpoint you want to send the data out of

	data – pointer to the data that you wish to send len – the length of the data that you wish to send
USBGenRead( )	USB_HANDLE USBGenRead(BYTE ep, //エンドポイント名 BYTE* data, //受信データ WORD len //受信するデータの長さ) 戻り値： 送信ハンドル(注) 受信データをエンドポイントから len バイト読み込む Receives the specified data out the specified endpoint Receives the specified data out the specified endpoint. ep – the endpoint you want to receive the data into data – pointer to where the data will go when it arrives len – the length of the data that you wish to receive USB_HANDLE – a handle for the transfer. This information should be kept to track the status of the transfer

(注)　USB_HANDLE –

　USB_HANDLE は BDT（バッファーデスクリプターテーブル）へのデータのポインタである。このポインタは最後に送信したデータの長さ、状態等いろ
いろな情報を読み込むのに使われる。 この USB_HANDLE は最初に確実に NULL に初期化しておくと既知の状態でつかうことができる。
usb_device.h のなかでは、以下のように定義されている。
#define USB_HANDLE volatile BDT_ENTRY*

USB_HANDLE is a pointer to an entry in the BDT(Buffer Descriptor Table). This pointer can be used to read the length of the last transfer, the status of the last transfer, and various other information. Insure to initialize USB_HANDLE objects to NULL so that they are in a known state during their first usage.
#define USB_HANDLE volatile BDT_ENTRY*