USB HID Host Library

2019-11-13

1 Content

2 Introduction

This is a library for making an **USB host** capable of reading and writing **packets** from/to **USB HID devices**. The library can handle mouses, keyboards and other (generic) HID devices.

At this moment only a version for PIC24 is available.

The library consists of 2 files:
"`USB_HOST_HID_Library`" and "`USB_HOST_Common_Library`". The first one, which has to be in a uses clause in your project, uses in turn the second one. Both should be entered in the project manager.

3 Usage in a program

3.1 Initialisation and test for readiness

The initialisation of the library and test for readiness of the USB HID device is done as follows:

```
uses USB_HOST_HID_Library, UartUtils;
...

procedure USB1Interrupt(); iv IVT_ADDR_USB1INTERRUPT;
begin
  USB_Interrupt;
end;

begin
  { Main program }

  InitMain;

  InitUsb;                          // <--- initialisation of the library

  repeat
  until USB_HID_Device_Ready or       // <--- test for readiness of the USB device
        (USB_Error > 0);

  if USB_HID_Device_Ready then        // the USB device is ready
  begin
    uart_write_line('');
    uart_write_line('HID device Ready');
    uart_write_line('');

    LatA.0 := 1;                    // signal readiness (example)
    uart_write_line('');
  end
  else
  begin                               // the USB device gives an error
    uart_write_line_word_hex(USB_Error);
    while true do; // stop all processing
  end;
```

The above example shows the error code if the device does produce an error.

As you can see the initialisation is done with the “InitUsb” routine, the readiness of the device is tested with “USB_HID_Device_Ready”. In case you do not want to block the software if “USB_HID_Device_Ready” stays false, you should also test “USB_Error”. If it becomes > 0 then you can stop waiting for “USB_HID_Device_Ready”, see the example above.
“InitUsb” has in principle to be called only once, all USB activity is handled under interrupt, also attachment and detachment of the USB device.

3.2 Reading and Writing packets

In this version of the library a packet length is defined by the USB device, it is not necessarely 64 bytes.

The length of a read (USB device to Host transfer) packet can be found as a result of "USB_HID_In_EndPoint_Size". A Size of zero indicates there is no IN endpoint.

The length of a write (Host to USB device transfer) packet as a result of "USB_HID_Out_EndPoint_Size". A size of zero indicates there is no OUT endpoint.

Reading and writing a packet is done as follows:

```
var Buff      : array[64] of byte;
      Success : boolean;
...
...
Success := USB_HID_Read(Buff); // a packet from the HID device is read into Buff
// the number if bytes read in is to be found in "USB_HID_In_EndPoint_Size".

if Success then // a packet was received from the HID device
begin
  // process the buffer content here
end;
...
...
// define the buffer content here
Success := USB_HID_Write(Buff); // the content of Buff is written to the device
// the number of bytes that are written can be found in "USB_HID_Out_EndPoint_Size"

if not success then // the transmission failed,
                    // the device was not ready to accept a packet
begin
  // perhaps retry at some later time
end;
...
...
```

Both routines above return "true" if success, "false" if failure.

4 Identification of the connected HID device.

The device connected to the USB host can be identified by in total 6 identifiers (3 numerical and 3 strings).
All identifiers are present in the so called USB Devicedescriptor or pointed to by it.
All identifiers are only valid if "**USB_HID_Device_Ready**" is true.

4.1 Numerical identifiers

The following 3 functions return each a word value:

```
function USB_Device_Vendor : word;
function USB_Device_Product: word;

function USB_Device_DeviceRelease: word; // BCD value
```

You can use these values to check which USB HID device is connected to the host.

4.2 String identifiers

These 3 identifiers are only exported in the library interface if the "`USB_HOST_STRINGS`" compiler directive in your project is defined. The best way to do this is adding the directive to a .pld file which is part of the project. Make sure the USB host library is also part of the project, it is compiled depending on that directive.

If the directive "`USB_HOST_STRINGS`" is defined in the project, then the program using the USB HID Host library must define 3 strings in which the identifiers will be presented:

```
{$IFDEF USB_HOST_STRINGS}
var  USB_Device_ManufacturerString,
     USB_Device_ProductString,
     USB_Device_SerialNrString: string[USBHostStringSize];
{$ENDIF}
```

In case a string identifier is not defined in the USB device an empty string will be the result.

5 Interface of the library

5.1 USB_HOST_HID_Library

```
unit USB_HOST_HID_Library;

// -----------------------------------------------------------------------------
// interface
// -----------------------------------------------------------------------------

uses USB_HOST_Common_Library;

const USB_HID_TYPE_UNKNOWN = 0;
      USB_HID_TYPE_KEYBOARD = 1;
      USB_HID_TYPE_MOUSE = 2;


procedure USB_Interrupt;
// To be called from the main interrupt routine (iv IVT_ADDR_USB1INTERRUPT)

procedure InitUsb;
// To be called once in the initialisation phase of the software (or when re-trying
initialisation after device detach)

function USB_HID_Device_Ready: boolean;
// Returns TRUE if an USB HID device is attached
```

```
function USB_HID_Read(var Buff: array[64] of byte): boolean;
// Returns true if some data has arrived, false means: no data arrived. Only valid if
USB_HID_Device_Ready
// The arguments are: //
//   Buff:  the user defined receive buffer (at least "USB_HID_In_EndPoint_Size" bytes in
size).
// Only to be used if there is an IN endpoint available.

function USB_HID_Write(var Buff: array[64] of byte): boolean;
// Returns success as true, false means: try later again (USB sendbuffer was still busy).
Only valid if USB_HID_Device_Ready
// The arguments are: //
//   Buff:  the user defined send buffer (at least "USB_HID_Out_EndPoint_Size" bytes in
size).
// Only to be used if there is an OUT endpoint available.

function USB_HID_SetReport(var Buff: array[64] of byte; Count:word): boolean;
// Sets the report of the HID device to the content of Buff. Only valid if
USB_HID_Device_Ready
// To be used if there is no OUT endpoint available.
// The problem here is that you should know the 'Count': the size of the report.
// The latter can be obtained from the Report Descriptor or the device type (see the USB
HID documentation)

var USB_Error: word;
// Returns the USB Error
// The USB_Error signals one error per bit (see "USB_HOST_Common_Library" for the
possible values)

function USB_HID_DeviceType: byte;
// Returns the type of HID device connected. Only valid if USB_HID_Device_Ready
// 0 = generic
// 1 = keyboard
// 2 = mouse

function USB_Device_Vendor : word; // <-- added on 2016-11-18
// Returns the USB Device Vendor numerical value. Only valid if USB_HID_Device_Ready

function USB_Device_Product: word; // <-- added on 2016-11-18
// Returns the USB Device Product numerical value. Only valid if USB_HID_Device_Ready

function USB_Device_DeviceRelease: word; // output is in BCD!   // <-- added on 2016-11-
18
// Returns the USB Device Release numerical value. Only valid if USB_HID_Device_Ready

function USB_HID_In_EndPoint_Size: byte;
// Returns the max size of the IN Endpoint (device -> host transfer). Only valid if
USB_HID_Device_Ready
// A result of zero means there is no IN endpoint

function USB_HID_Out_EndPoint_Size: byte;
// Returns the max size of the OUT Endpoint (host -> device transfer). Only valid if
USB_HID_Device_Ready
// A result of zero means that there is no OUT endpoint

function USB_HID_Report_Descriptor_size: word;
// Returns the size of the HID Report descriptor. Only valid if USB_HID_Device_Ready
```

```
function USB_HID_Get_Report_Descriptor(var Buff: array[64] of byte; MaxLen: byte):
boolean;
// Gets the devices's report descriptor. Buff is at least 64 bytes in size


{$IFDEF USB_HOST_STRINGS}
const USBHostStringSize = __USBHostStringSize;
{$ENDIF}

// -----------------------------------------------------------------------------
// -------------------- USB Error Constant Definitions -------------------------
// -----------------------------------------------------------------------------

const
      // USB_Error constants
      USB_DEVICE_DESCRIPTOR_ERROR                = $001; // bit 0
      USB_CONFIG_DESCRIPTOR_ERROR                = $002; // bit 1
      USB_INTERFACE_DESCRIPTOR_ERROR             = $004; // bit 2
      USB_INTERFACE_DESCRIPTOR_CLASS_ERROR       = $008; // bit 3
      USB_INTERFACE_DESCRIPTOR_SUBCLASS_ERROR    = $010; // bit 4
      USB_INTERFACE_DESCRIPTOR_PROTOCOL_ERROR    = $020; // bit 5
      USB_ENDPOINT_DESCRIPTOR_ERROR              = $040; // bit 6
      USB_ENDPOINT_DESCRIPTOR_ATTRIBUTES_ERROR = $080; // bit 7
      USB_ENDPOINT_DESCRIPTOR_PACKETSIZE_ERROR = $100; // bit 8


// -----------------------------------------------------------------------------
implementation
// -----------------------------------------------------------------------------
```

5.2 USB_HOST_Common_Library

```
unit USB_HOST_Common_Library;

// interface

{$IFDEF USB_HOST_STRINGS} // <-- added 2016-11-18
const __USBHostStringSize = 30; // string size to be used for the following 3 strings
var  USB_Device_ManufacturerString,
    USB_Device_ProductString,
    USB_Device_SerialNrString: string[__USBHostStringSize]; external;
     // content of the strings is only valid if USB_MSD_Device_Ready is true
{$ENDIF}

procedure __SendToken(Token, EndPoint: byte);
function   __Enumerate: boolean;
procedure __Reset_Usb_Device;
procedure __SendSetupCommand;
function   __SendRxCommand(var Buf: array[64] of byte; NrToReceive: byte): word;
procedure __SendTxCommand(var Buf: array[64] of byte; Len_: byte);
function   __USB_Device_Vendor : word; // <-- added on 2016-11-18
function   __USB_Device_Product: word; // <-- added on 2016-11-18
function   __USB_Device_DeviceRelease: word; // output is in BCD!
```

6 Keyboard and mouse data format

```
This is the content meaning of data read from keyboard and mouse:
```

6.1 USB Keyboard Data Report Format

Byte	b7	b6	b5	b4	b3	b2	b1	b0
0 Modifiers	Right GUI	Right Alt	Right Shift	Right Ctrl	Left GUI	Left Alt	Left Shift	Left Ctrl
1	Reserved							
2	Key 1 scan code							
3	key 2 scan code							
4	Key 3 scan code							
5	Key 4 scan code							
6	Key 5 scan code							
7	Key 6 scan code							

6.2 USB Mouse Data Report Format

Byte	b7	b6	b5	b4	b3	b2	b1	b0
Left Alt	Left Shift	Left Button						
1	Horizontal (X) Displacement							
2	Vertical (Y) Displacement							

This is the output report format for the keyboard leds:

6.3 USB Keyboard Set_Report Data Format for LED

Byte	b7	b6	b5	b4	b3	b2	b1	b0
0 LED Report	Con stant	Con stant	Con stant	Kana	Com pose	Scroll Lock	Caps Lo	

7 Full example

The progam below shows (via Uart1) the data of an USB HID device when it is attached.
Also the report descriptor (describing the IN data format) is shown. No decoding of it is provided however.
Additionally data is read from the HID device and shown via Uart1 (hex output).
One byte (with changing value) is also written to the HID device. The leds from a keyboard are set with the "Set Report" USB HID command.
This example needs, besides the USB Host units already mentioned, a unit named "UartUtils". It can be found here: https://libstock.mikroe.com/projects/view/39/utilities.

```
program Test_of_Own_HID_HOST_Lib;

{ Declarations section }

uses USB_HOST_HID_Library ,UartUtils;

var SendBuffer: array[64] of byte; volatile;
```

```
    RecBuffer:  array[64] of byte; volatile;

{$IFDEF USB_HOST_STRINGS}
var  USB_Device_ManufacturerString,
    USB_Device_ProductString,
    USB_Device_SerialNrString: string[USBHostStringSize];
{$ENDIF}

var Leds: byte;
    ReportDescriptor: array[64] of byte;
    PrevReady: boolean;
    Tmp: word;

procedure InitMain;
begin
  //----------------------------------------
  // Disable interrupts
  //----------------------------------------
  USB1IE_bit := 0;        // disable usb interrupts
  ADPCFG := 0xFFFF;       // Configure AN pins as digital I/O

  //----------------------------------------
  // Ports Configuration
  //----------------------------------------
  LATA  := 0;
  TRISA := 0xfffe;  // 1 bit for the led

  Leds := 0; // report for a keyboard
  PrevReady := false;

  //----------------------------------------
  // Uart Configuration
  //----------------------------------------
  // Uart1
  PPS_Mapping (8, _INPUT,  _U1RX);  // RP7 is uart1 input  pin 17
  PPS_Mapping (7, _OUTPUT, _U1TX);  // RP6 is uart1 output pin 16

  Uart1_Init(115200);
  delay_ms(100);
  Uart1_Write_Text(#13 +#10 +#13 +#10);
  Uart1_Write_Text('Start' +#13 +#10);
  Uart1_Write_Text(#13 +#10 +#13 +#10);

end;

procedure USB1Interrupt(); iv IVT_ADDR_USB1INTERRUPT;
begin
  USB_Interrupt;
end;

begin
  { Main program }

  InitMain;
  InitUsb;

  while true do
  begin
    if USB_HID_Device_Ready <> PrevReady then // something changed
    begin
      PrevReady := Usb_HID_Device_Ready;

      if PrevReady then // usb device became ready
      begin
        LatA.0 := 1;

        Uart1_Write_Text(#13 + #10 + 'Device ready' +#13 +#10);

        Uart1_write_text('Vendor: ');
        uart_write_line_word_hex(USB_Device_Vendor);

        Uart1_write_text('Product: ');
        uart_write_line_word_hex(USB_Device_Product);

        Uart1_write_text('Release: ');
        Tmp := bcd2Dec16(USB_Device_DeviceRelease);
        uart_write_line_word(Tmp);

        Uart1_write_text('DeviceType: ');
        if USB_HID_DeviceType = USB_HID_TYPE_UNKNOWN then Uart1_write_text('Unknown');
        if USB_HID_DeviceType = USB_HID_TYPE_MOUSE then Uart1_write_text('Mouse');
        if USB_HID_DeviceType = USB_HID_TYPE_KEYBOARD then Uart1_write_text('Keyboard');
        Uart1_write_text(#13 + #10);

        Uart1_write_text('In Endpoint size: ');
        Uart_Write_Line_byte(USB_HID_In_Endpoint_size);

        Uart1_write_text('Out Endpoint size: ');
        Uart_Write_Line_byte(USB_HID_Out_Endpoint_size);

        Uart1_write_text('HID_Report Descriptor size: ');
        Uart_Write_Line_word(USB_HID_Report_Descriptor_Size);


{$IFDEF USB_HOST_STRINGS}
        Uart1_write_text('Manufacturer: ');
        Uart_write_line(USB_Device_ManufacturerString);

        Uart1_write_text('Product: ');
        Uart_write_line(USB_Device_ProductString);

        Uart1_write_text('Serial Number: ');
        Uart_write_line(USB_Device_SerialNrString);
{$ENDIF}

        Uart_write_Line('Report Descriptor:');
        USB_HID_Get_Report_Descriptor(ReportDescriptor, USB_HID_Report_Descriptor_size);
        Uart_DumpBuffer(ReportDescriptor, USB_HID_Report_Descriptor_size);

        Uart1_Write_Text(#13 +#10);
      end
      else
      begin // usb device became absent
        LatA.0 := 0;
        Uart_write_Line('Device Disconnected');
        uart_write_text('USB error: ');
        uart_write_line_word_hex(USB_Error);
        Uart1_Write_Text(#13 +#10);
      end;
    end;

    if USB_HID_Device_Ready then
    begin

      if USB_HID_DeviceType <> USB_HID_TYPE_MOUSE then delay_ms(500); // every 1/2 second
for non mouse devices

      if USB_HID_Device_Ready
      then
      begin
        LatA.0 := 1;

        // get some data from the HID device and send it to the uart
        memset(@Recbuffer, 0, 64); // clear buffer, for test purpose only
        if USB_HID_Read(RecBuffer) then Uart_DumpBuffer(RecBuffer,
USB_HID_In_EndPoint_Size);

        SendBuffer[0] := Leds;
        inc(Leds);
        Leds := leds mod 8; // values 0..7

        if USB_HID_Out_EndPoint_size > 0  // the device has an 'out' endpoint (see from
the host)
        then USB_HID_Write(SendBuffer) // simply send the outbuffer
        else
        begin // the device has no 'out' endpoint, but perhaps a report can be set in it
(e.g. for keyboards)
          if (USB_HID_DeviceType = USB_HID_TYPE_KEYBOARD) then // keyboard
          begin
            if USB_HID_SetReport(Sendbuffer,1)
            then Uart_write_Line('Set Report Ok')
            else Uart_write_Line('Set Report Failed')
          end;
        end;
      end;

    end;

  end;

end.
```

[end of document]