Technical feedback on the USB Host library – Multiple device enumeration through cascaded hubs

Dear Suwa,

First of all, I would like to thank you for your valuable work on extending the Microchip USB Host library to support multiple USB devices and cascaded USB hubs.

I am currently evaluating your library as the foundation for a USB HID interface for a Z80-based system. During development and testing, I encountered a problem related to the enumeration of multiple USB devices already connected to the hubs when the microcontroller is powered on or reset.

After a detailed analysis of the library, I introduced two modifications that I would like to share with you, together with their technical motivations and experimental results.

1. Hardware and development environment

The test platform consists of:

- Microcontroller: PIC24FJ256GB606.
- External oscillator: 8 MHz.
- Development environment: MPLAB X IDE 6.15.
- Compiler: MPLAB XC16 v2.10.
- UART diagnostic output through an FT232 USB-to-serial interface.
- One seven-port USB hub connected directly to the USB Host.
- One four-port USB hub connected to a downstream port of the first hub.
- Multiple USB HID devices connected to both hubs, including keyboards and mice.

A USB Mass Storage device was also introduced during the final tests to verify the correct handling of unsupported USB device classes.

2. Description of the original problem

The library generally operated correctly when USB devices were connected individually after the initialization of the USB Host.

However, when several devices were already connected to the hubs before powering on or resetting the microcontroller, the enumeration process frequently failed.

Two different behaviours were observed:

- With the original heap size of 4096 bytes, the microcontroller repeatedly reset, reporting an RCON value of `0x4000`.

- Increasing the heap size to 8192 bytes eliminated these resets during the initial tests, but device enumeration still failed after several communication attempts, eventually entering `STATE_HOLDING`.

The diagnostic output suggested that the enumeration of a device could be affected by events generated by other hub ports while the enumeration process was still in progress.

A detailed examination of the USB Host and Hub state machines revealed two potential problems.

3. First modification: preservation of the selected endpoint in the transaction scheduler

3.1 Problem identified

The following functions in `usb_host.c` were examined:

- `_USB_FrameFirstTransaction()`
- `_USB_FindService()`
- `_USB_FindServiceEndpoint()`
- `_USB_FrameNextTransaction()`
- `_USB_AdvanceFindNextToken()`

The function `_USB_FrameFirstTransaction()` builds a list of endpoints requiring service by calling `_USB_FindService()` for the different USB transfer types.

During this process, `_USB_FindServiceEndpoint()` selects an endpoint and stores its pointer in:

```
pCurrentEndpoint[deviceAddress]
```

However, the original `USB_SERVICE_ENDPOINT` structure contains only the device address and the transfer type.

Consequently, if the same USB device has pending transactions belonging to different transfer types, a subsequent call to `_USB_FindServiceEndpoint()` may overwrite the endpoint pointer previously selected for that device.

When the scheduler subsequently processes an entry in the `ServiceEndpoint[]` array, the current endpoint pointer may therefore no longer correspond to the transfer type associated with that entry.

3.2 Modification of USB_SERVICE_ENDPOINT

An endpoint pointer was added to the structure:

```
typedef struct
{
    BYTE deviceAddress;
    BYTE transferType;
    USB_ENDPOINT_INFO *pEndpoint;
} USB_SERVICE_ENDPOINT;
```

Each scheduler entry now contains all three elements required to identify the selected transaction:

- Device address.
- Transfer type.
- Endpoint pointer.

3.3 Modification of _USB_FindService()

When an endpoint is added to the service list, its pointer is stored together with the device address and transfer type.

The following statement was added before incrementing `bnumberOfSE`:

```
ServiceEndpoint[bnumberOfSE].pEndpoint =
    pCurrentEndpoint[deviceAddress];
```

3.4 Modification of _USB_FrameNextTransaction()

At the beginning of the transaction selection loop, the endpoint pointer associated with the current scheduler entry is restored:

```
pCurrentEndpoint[ServiceEndpoint[nextSE].deviceAddress] =
    ServiceEndpoint[nextSE].pEndpoint;
```

This ensures that the endpoint selected during the construction of the service list is the same endpoint used when the corresponding transaction is processed.

3.5 Experimental result

Although this modification corrects an inconsistency in the transaction scheduling mechanism, it did not, by itself, resolve the original enumeration failure.

Further analysis was therefore necessary.

4. Second modification: serialization of USB default address usage

4.1 Problem identified

The second problem concerns the management of USB address zero, also known as the default USB address.

According to the USB protocol, a newly reset USB device responds to address zero until a successful `SET_ADDRESS` request assigns a unique address to it.

When several USB devices are connected to one or more hubs, multiple port connection events may be generated within a short period of time.

The original implementation allows the Hub state machine to initiate a port reset while another device may still be undergoing enumeration at address zero.

Furthermore, `USBHostTasks()` processes the Hub state machine and subsequently iterates through the state machines associated with the individual USB device addresses.

The USB transaction scheduler prevents simultaneous transmission of multiple USB transactions, but this does not guarantee exclusive use of address zero throughout the initial enumeration sequence.

If two downstream devices are reset and enabled before either of them has completed its address assignment, both devices may respond to transactions directed to address zero.

This represents a potential addressing conflict, regardless of the correct operation of the transaction scheduler.

4.2 Introduction of the enumeration ownership concept

A new static variable was introduced in `usb_host.c`:

```
static BYTE enumerationOwner = 0;
```

This variable identifies the device that currently has exclusive access to the USB default address.

Its meaning is:

- `enumerationOwner == 0`: the default USB address is available.
- `enumerationOwner != 0`: the default USB address is reserved for the device associated with the specified logical device address.

The reservation begins when a Hub port reset request is successfully initiated and ends when the corresponding device successfully completes its `SET_ADDRESS` transaction.

The purpose is not to serialize the entire enumeration process, but only the portion during which a device requires exclusive access to address zero.

4.3 Modification of HUB_SET_PORT_FEATURE

Within `_USB_HubTasks()`, the following condition was added at the beginning of `HUB_SET_PORT_FEATURE`:

```
if (selector[n] == PORT_RESET &&
    enumerationOwner != 0 &&
    enumerationOwner != HubPortAddress[n][port[n]-1])
{
    break;
}
```

This prevents the reset of another downstream port while the default USB address is reserved by a different device.

The comparison with the current device address is necessary to allow the owner itself to repeat a failed port reset request.

After `USBHostIssueDeviceRequest()` returns `USB_SUCCESS`, the following code was added before entering `HUB_SET_PORT_FEATURE_WAIT`:

```
if (selector[n] == PORT_RESET)
{
    enumerationOwner = HubPortAddress[n][port[n]-1];
}
```

The default USB address is therefore reserved as soon as the port reset request has been accepted for processing.

Other Hub port feature requests are not directly affected by this condition.

4.4 Modification of SUBSUBSTATE_SET_DEVICE_ADDRESS_COMPLETE

Within `USBHostTasks()`, in the state `SUBSUBSTATE_SET_DEVICE_ADDRESS_COMPLETE`, the original code updates the address used by the USB hardware:

```
usbDeviceInfo[deviceAddress].deviceAddressAndSpeed =
    (usbDeviceInfo[deviceAddress].flags.bfIsLowSpeed << 7) |
    usbDeviceInfo[deviceAddress].deviceAddress;
```

Immediately after this statement, the following code was added:

```
if (enumerationOwner == deviceAddress)
{
    enumerationOwner = 0;
}
```

At this point, the `SET_ADDRESS` control transfer has been successfully completed and the device no longer requires the default USB address.

Another downstream port may therefore be reset, even while the previous device is still completing the remaining enumeration stages, such as:

- Configuration descriptor acquisition.
- Configuration descriptor parsing.
- Configuration selection.
- Client driver initialization.

This preserves the possibility of overlapping enumeration processes once the devices have been assigned unique USB addresses.

4.5 Modification of hub_detached()

The following code was added to `hub_detached()`, immediately after setting the device state to `STATE_DETACHED`:

```
if (enumerationOwner == address)
{
    enumerationOwner = 0;
}
```

This releases the reservation when the device owning the default USB address is disconnected and its disconnection is processed.

Since `hub_detached()` recursively processes the devices connected to a cascaded hub, the same mechanism also applies when an upstream hub is disconnected.

4.6 Handling of failed port reset requests

The original implementation of `HUB_SET_PORT_FEATURE_WAIT` returns to `HUB_SET_PORT_FEATURE` when a port feature request fails.

No modification to this state was necessary.

The ownership check introduced in `HUB_SET_PORT_FEATURE` explicitly allows the current owner to repeat its own reset request without releasing the reservation.

This prevents another device from acquiring the default USB address between successive reset attempts.

5. Experimental results

After introducing the enumeration ownership mechanism, the original enumeration problem was no longer observed in the tests performed.

The following operating conditions were successfully tested:

- Power-on with multiple USB devices already connected.
- Hardware reset of the microcontroller with all USB devices connected.
- Repeated power-off and power-on cycles.
- Enumeration of devices connected through cascaded USB hubs.
- Enumeration of multiple keyboards.
- Enumeration of keyboards and mice within the same USB topology.
- Connection and rejection of an unsupported USB Mass Storage device.
- Enumeration of an additional HID device while the unsupported device remained connected.
- Hardware reset with both supported and unsupported USB devices already connected.

In one of the final tests, the following USB devices were successfully processed:

USB address	Device	Result
0x01	Seven-port USB hub	Successfully initialized
0x03	Four-port USB hub	Successfully initialized
0x04	Keyboard	Successfully initialized
0x09	Keyboard	Successfully initialized
0x0A	USB Mass Storage device	Correctly rejected
0x0C	Mouse	Successfully initialized

The unsupported USB Mass Storage device correctly entered `STATE_HOLDING` after receiving its unique USB address and after the configuration descriptor parser determined that no compatible client driver was available.

This did not prevent the subsequent enumeration and initialization of the mouse.

The diagnostic output also confirmed that the enumeration of a second device could begin while the first device was still completing its configuration and client driver initialization, after its unique USB address had been assigned.

6. Heap size considerations

During the initial investigation, the heap size was increased from 4096 to 8192 bytes.

However, after introducing both code modifications, the original heap size of 4096 bytes was restored and the complete USB topology described above was successfully initialized.

The experimental results can be summarized as follows:

Software configuration	Heap size	Result
Original library	4096 bytes	Repeated microcontroller resets
Original library	8192 bytes	Enumeration failures
Modified endpoint scheduler	8192 bytes	Enumeration failures
Modified scheduler and enumeration ownership	8192 bytes	Successful enumeration
Modified scheduler and enumeration ownership	4096 bytes	Successful enumeration

These results demonstrate that increasing the heap size is not necessary to resolve the enumeration problem under the tested operating conditions.

The precise cause of the original microcontroller resets has not been conclusively established.

For my own Z80 USB interface application, I intend to retain a heap size of 8192 bytes as a precautionary memory allocation margin, rather than as a requirement of the proposed corrections.

7. Remaining considerations

The modifications have demonstrated satisfactory operation under the tested conditions, but some exceptional conditions require further investigation.

In particular:

- If enumeration fails permanently before the successful completion of `SET_ADDRESS`, the ownership reservation remains active until the corresponding disconnection is processed.
- Releasing the reservation unconditionally in this situation would be incorrect, since the failed device might still respond to address zero. A possible solution would involve disabling the corresponding Hub port before releasing the reservation.
- Deferring a port reset by remaining in `HUB_SET_PORT_FEATURE` may, under certain event sequences, prevent the same Hub state machine from processing a pending reset-completion or disconnection event belonging to the current owner. This potential blocking condition has not been observed during the successful tests but deserves further investigation.
- The maximum heap requirement for a fully populated USB topology has not been determined.

These limitations should be considered if the proposed modifications are integrated into a general-purpose release of the library.

8. Conclusions

The two modifications introduce two distinct concepts into the USB Host library:

1. **Preservation of the endpoint context within each transaction scheduler entry**, ensuring consistency between endpoint selection and transaction execution.
2. **Exclusive ownership of the USB default address**, preventing another downstream port from being reset before the device currently using address zero has successfully completed its address assignment.

The second modification produced the decisive improvement in the observed enumeration behaviour, with the first modification already in place.

The modified library has reached a satisfactory level of reliability for me to proceed with the development of my Z80 USB HID interface.

All my source code additions and modifications are identified by dated comments in the following format:

```
// Aggiunto Andrea DD/MM/YYYY
```

I would be pleased to make the modified source code available to you for examination and possible integration into your USB Host library.

I hope that this contribution will be useful to you and to other developers using your work.

Thank you again for making your USB Host implementation available to the developer community.

Best regards,

Andrea Vavassori
Italy