smxUSBH FAQ

by Yingbo Hu
R&D Embedded Software Engineer

1. What are the general steps to port smxUSBH to a different hardware or software platform?

smxUSBH defines three porting layer function sets: OS and compiler porting layer (uosport.c/h), hardware porting layer (uhdwport.c/h), and non-cacheable heap layer (uheap.c/h).

a. You need to implement all the macros and functions defined in uosport.h, according to your OS and compiler. If you are using the SMX® RTOS, this step has been already done for you.

b. Modify hardware-related functions in uhdwport.c for your hardware, if necessary. We support many standard evaluation boards, and these files have configuration sections for them. We recommend you use exactly the same hardware as the evaluation board unless modification is a must.

c. If your system is using an MMU or you enabled the data cache and you are using an OHCI/UHCI/EHCI compatible controller, make sure that the memory allocated by su_AllocNC() is non-cacheable. Otherwise you will have cache coherency problems. (See #4 for more information.)

d. Create a new project file for your compiler and put all of the .c and .s files within the XUSB directory into that project. Keep the original directory structure. Add search paths for XUSB\USBH, XUSB\HCD, etc to your project file. Remember, for most cases, the only files you should change are those files within XUSB root directory.

2. What I should do if smxUSBH does not run properly on my platform?

smxUSBH provides a function to output debug information to help you and us to diagnose the problem.

a. Set SU_DEBUG_LEVEL, which is defined in ucfg.h, to 6 and implement su_OutputString() in uhdwport.c to output the text information to a serial port, some reserved memory area, or the terminal window of your debugger.

b. Send the debug output information to us, if you are not able to diagnose the problem yourself.

3. Which USB host controller should I use?

See my whitepaper: How to Select a USB Host Controller.

4. If I enable cache or MMU on my processor, what should I do for OHCI, UHCI, and EHCI USB host controllers?

The memory used by the OHCI/UHCI/EHCI host controller needs to be non-cacheable, so if your system is using data cache or MMU to improve the performance, you may need to do one of the following:

a. Allocate a special memory area that is non-cacheable and modify the heap initialization function su_HeapInit() in uheap.c to allocate memory from that area instead of from the standard system heap. Your processor needs to set different cache properties for different memory areas. See the sample code in su_HeapInit() for Coldfire MCF5329.

b. Set the MMU to map the system heap area to a non-cacheable area. For example, physical memory 0x40000000 is mapped to virtual address 0x80000000 which is the cacheable memory area. Implement su_PhysicalToVirtualAddr() and su_VirtualToPhysicalAddr() in uhdwport.c to convert the memory address between physical address and virtual address, so all the memory pointers allocated by the USB are physical addresses instead of virtual addresses.

c. Use the internal SRAM for the USB heap. Internal SRAM is always non-cacheable. See the sample code in su_HeapInit() for Coldfire MCF5329.

d. Set SU_COPY_DATA in ucfg.h to 1 so that the application buffer passed to the USB class driver will be copied to a non-cacheable buffer. Then the USB controller can transfer it properly. In the normal case, the USB stack will use that memory directly. External USB controllers such as ISP116x, 1362, and 176x need to transfer data from the system's memory to the controller's internal FIFO via the external memory bus, so the memory used by the controller driver can be cacheable or bufferable.

5. Why is the performance of ISP116X/1362/176X not as good as OHCI/UHCI/EHCI?

OHCI/UHCI/EHCI USB controllers directly access the memory within the processor's memory space via DMA; there is no extra copy operation. But ISP116X, ISP1362, and ISP176X need to copy the data from the processor's memory to the internal FIFO of those USB controllers so they generate extra overhead doing this. The host controller driver also needs to copy some special data structures, the PTD, to meet the requirements of the host controller, which generates even more overhead.

6. How can I use DMA for the USB host controller driver of smxUSBH?

For OHCI, UHCI and EHCI, the DMA transfer is transparent to the host controller driver so you do not need any special code. For ISP116X, ISP1362, and ISP1761, you can use DMA to transfer data from processor's memory to the chip but we have not implemented it because DMA operation is processor-dependent. However you can implement it yourself in the porting layer functions, without the need to know the details of that host controller. For those host controller drivers, we define the data transfer function su_ISPxxxReadBuf() and su_ISPxxxWriteBuf() in

the porting layer. What you need to do is to implement those functions to use DMA instead of the default PIO approach.

7. What should I do if I want to use ISP176x and get high performance?

ISP176x itself cannot guarantee high performance. The USB stack and host controller driver also needs a high performance processor. From our experience, if you need 5MB/sec file system performance, you need at least a 200MHz ARM9 processor and 10MB/sec data transfer rate on the bus between your processor and ISP176x. Also, 32-bit data bus width is recommended.

8. What I should do if I want to use one USB port to support both Host and Device? Do I need OTG?

In most cases, you do not need OTG even if you need USB host and device support on the same port. The only case where you need OTG is when the connected device is also an OTG device and you need to switch the port's role without changing the USB cable. Port 1 on the NXP ISP1362 or ISP1761 can be used as a Host port or a Device port (but not at same time). What you need to do is:

a. Connect a mini-AB receptacle to Port 1.

b. Disable the OTG feature of Port 1.

c. Run smxUSBH and smxUSBD simultaneously on your system.

d. Use different USB cables when you connect your embedded device to a USB host or to a USB device.

When you want your system to work as USB device, plug in a mini-B connector to the mini-AB receptacle; when you want your system to work as USB host, plug in a mini-A connector to the mini-AB receptacle. The ID pin of the mini-AB receptacle will automatically route the USB port to the correct USB controller.

9. Are there any limitations for smxUSBH?

smxUSBH does not provide the following features for some simple USB host controllers:

a. It does not support external hub for CF522xx and MAX3421E. These processors or controllers all have severely limited memory, so supporting multiple USB devices by an external hub is not realistic.

b. It does not support ISOC transfer mode for CF522xx, LM3S, and MAX3421E. These processors or controllers are all low-end with limited memory, so supporting real time audio and video data transfer is not realistic.

10. Why are some flash disks slower than others? Is there any software problem?

A recent customer email:

"We have a particular USB Flash Disk that writes files 10 times slower than the others we have. We connected a USB protocol analyzer, and observed the following behavior on the slower device:

- Host sends data command to device
- Device ACKs
..
.. more data
..
- Host sends data command to device
- Device NAKs
..
.. device NAKs for a while
..
- Finally, the device ACKs, and the host continues sending more data

Is this normal behavior? Is this a flow control technique used when a USB Flash Disk cannot keep up? Is there possibly a stack problem?"

My reply: It is a normal case. NAK is a flow control flag used to notify the USB host that the device is not ready to receive data. NAK is generated by the device's firmware; it is not related to the USB host stack. Flash disks need to do garbage collection and wear leveling, so even a normal fast flash disk sometimes will become very slow. You can check that flash disk in Windows. It should behave the same way if you copy a big file to it.