smxUSBD™

USB Device Stack

smxUSBD is a robust USB device stack specifically designed and developed for embedded systems. It is written in C, and can run on any hardware platform. While optimized for SMX®, smxUSBD can be ported to another RTOS or operate in a stand-alone environment.

smxUSBD is a full-featured USB device stack. It offers a clean, modular design that enables embedded developers to easily add USB device capabilities to their products. Normally this is done to permit connection to a PC or laptop in order to upload or download data, tables, code, or audio, or to control or configure devices. smxUSBD device stack is offered separately from the smxUSBH host stack to reduce system cost and memory usage for projects not needing a host stack. It is compliant with the USB v2.0 specification (see www.usb.org).

For easy connectivity to a PC or laptop, smxUSBD includes a mouse function driver, and the following are available separately: Audio, Mass Storage, RNDIS (Ethernet over USB), and Serial. Each is compatible with the corresponding Windows, Mac OS, and Linux USB driver.[1] Thus, a device using smxUSBD does not require a custom W/M/L driver in order to connect to a PC or laptop. All that is needed is to decide on the device connection most appropriate for your device and to use the corresponding API for that device – see below.

Also available is a USB composite function driver framework, which allows a device to simultaneously look like two or more USB devices. The recently added USB multi-port serial function driver allows a single USB connection to function as multiple serial ports (up to 1/2 the number of controller endpoints). This function driver comes with a custom Windows driver that supports it.

[1]RNDIS is not supported by Mac OS

Features

- Supports all four USB data transfers (control, bulk, isochronous, and interrupt).
- Compliant with USB Specification 2.0.
- Function Drivers are available for audio, mass storage, mouse, RNDIS (Ethernet over USB), and serial. These are compatible with Windows, Mac OS, Linux drivers.[1]
- Minimum code footprint 7.5 KB for ARM Thumb and 14 KB for ColdFire, including controller driver.
- Minimum RAM footprint 5 KB.
- Multi-port serial using a custom Windows driver is available.
- Composite device support.
- Compatibility with ARM, Blackfin, ColdFire, PowerPC, x86, and other CPUs.
- Supports 16-bit addressing CPUs such as TI TMS320C55xx DSPs.
- NXP ISP1161, 1181, 1362, 158x, and 1761 device controller support
- On-chip device controller support for: Analog Devices Blackfin BF5xx, Atmel AT91, Freescale ColdFire 52211, 52223, 5227x, 525x, 532x/7x, 5445x, 548x/7x, i.MX1/L, NXP LPCxxxx, Sharp LH7A4xx, and STMicro STR7/9.
- Written entirely in ANSI-C.
- Driver template for new platform porting
- Optimized for SMX® RTOS.
- Easily portable to other RTOSs.
- Also runs stand-alone.

Layers

- **Function Driver Layer** provides USB functions to application such as mass storage, mouse, and serial.
- **Device Core Layer:** provides the common USB device framework.
- **Device Controller Driver Layer** provides the interface to the selected USB device controller. **Porting Layer** provides service functions related to the hardware, OS, and compiler.

Function Drivers

The following sections describe each function driver and its API. The USB host is a Windows, Mac OS, or Linux system, except for RNDIS, which supports Windows and Linux only.

Audio

The Audio function driver makes your device look like a sound card to the USB host. You can include a speaker and/or microphone in this audio device so you can playback and/or record sound. You can also integrate a MIDI port so your device can accept MIDI data. There is no need to install any driver or .inf file in Windows, Mac OS, or Linux to support this device but you may need to implement the sound device driver yourself, according to your system hardware and software environment.

sud_AudioIsConnected(port)
sud_AudioSendAudioData(port, pData, iLen)
sud_AudioGetAudioData(port, pData, iLen)
sud_AudioGetCurSpkSettings(port, pSettings)
sud_AudioGetCurMicSettings(port, *pSettings)
sud_AudioSendMIDIData(port, pData, iLen)
sud_AudioGetMIDIData(port, pData, iLen)
sud_AudioRegisterNotify(port, handler)
sud_AudioPackMIDIEvent(port, pData, pEvent)
sud_AudioUnpackMIDIEvent(port, pData, pEvent)

Mass Storage

The Mass Storage function driver makes your device look like a removable disk to the USB host. You can copy files to and from it.

sud_MSRegisterDisk(pdiskop, lun)

Mouse (Included)

The Mouse function driver makes your device look like an HID mouse to the USB host. It moves the mouse cursor on your PC.

sud_MouseInput(key, x, y, wheel)

RNDIS (Ethernet over USB)

The RNDIS function driver makes your device look like a Network Adapter to a Windows or Linux USB host. The host can communicate with this device via Ethernet data packets. Normally you need a TCP/IP stack on your device and use the APIs provide by this function driver to emulate an Ethernet device and add it to your network stack. This device has been integrated with smxNS, our TCP/IP stack. Then the host and your device can communicate with each other by TCP/IP with a USB cable instead of an Ethernet cable. One use of RNDIS is to allow configuring a device from the web browser on a host communicating with a web server on your device. This is especially useful if your processor has only a USB device controller and no Ethernet controller on chip.

sud_RNDISIsPortConnected(port)
sud_RNDISWriteData(port, pBuf, len)
sud_RNDISRegisterPortNotify(port, handler)
sud_RNDISSetEthernetAddr(port, MACaddr)

Serial

The Serial function driver makes your device look like one or more COM ports to a Windows, Mac OS, or Linux USB host. You can use standard Win32 functions to communicate with

the device, just like if it were connected to a real RS232 port. For the multi-port option, we provide a custom Windows USB serial driver, since the built-in Windows driver supports only one port. Our driver also allows using only 1/2 the number of endpoints saving them for other uses.

```
sud_SerialIsPort Connected(port)
sud_SerialWriteData(port, pBuf, len)
sud_SerialDataLen(port)
sud_SerialReadData(port, pBuf, len)
sud_SerialSetLineState(port, iState)
sud_SerialGetLineState(port, piState)
sud_SerialGetLineCoding(port, pdwDTERate,
        pbParityType, pbDataBits, pbStopBits)
sud_SerialRegisterPortNotify(port, handler)
```

Composite Devices

smxUSBD allows creating a composite device. Such a device has multiple interfaces that are active at the same time using a single controller chip. For example, a composite device might combine serial and mass storage. See the smxUSBD User's Guide for more discussion of this.

Writing New Drivers

Contact us first to make sure we are not already working on a driver you need.

smxUSBD provides a function driver template and a section in the manual to help you write a new function driver, if needed.

smxUSBD also provides a USB device controller driver template and a section in the manual, to help you write a new driver in case it does not support your USB device controller.

Porting

Due to its extensive processor support, little or no porting is necessary when smxUSBD is used with SMX. However, smxUSBD is designed to work with other RTOSs and to run standalone.

The hardware porting layer consists of two files, udhdw.h and udhdw.c. These files contain definitions, macros, and functions to port smxUSBD to a new processor. In addition, if the USB device controller is not among those already supported, a new driver will need to be written.

smxUSBD was developed for use with SMX®, but it can be ported to any RTOS. The RTOS porting layer consists of two files, udosport.h and udosport.c. These files contain definitions, macros, and functions to port to a new RTOS.

smxUSBD works best in a multitasking environment. However, it can also be ported to a non-multitasking stand-alone environment.

16-Bit Addressing Support

smxUSBD supports processors that can only do 16-bit memory addressing (not byte addressing) such as the TI TMS320C55xx DSPs. These processors are difficult to support for typical communication protocols because of byte data and byte fields in standard protocol data structures. This support is enabled by a configuration option in smxUSBD.

Testing

We test smxUSBD with USBCheck v5.1 on a Windows PC to verify that it passes the Chapter 9 USB compliance tests for full speed and high speed. We also test with USBCV v1.3 and it passes the Chapter 9, HID, and MSC tests.

Code Size

Code size can vary greatly depending upon the processor, compiler, and optimization level.

Component	**ARM Thm** IAR (KB)	**ARM** IAR (KB)	**BF** VDSP (KB)	**CF** CW (KB)
Core	5	8	12	9
Analog Dev BF5xx	—	—	3.3	—
Atmel AT91	2	3	—	—
Freescale CF532x	—	—	—	4
Freescale CF548x	—	—	—	9
Freescale MX1			—	—
NXP ISP1181				4
NXP ISP158x				—
NXP LPCxxxx	2.6	4	—	—
Sharp LH7A40x	2.5	4	—	—
STMicro STR7/9	2.5	4	—	—
Audio drv	3	6	3.5	6.5
Mass Storage drv	3.1	5	5.5	5
Mouse drv	0.5	1	1	1
Serial drv	1.5	2.5	2.5	2.7
RNDIS drv	2.5	3.5	2.5	4.7
Composite drv	0.5	1	1	1

CF532x also supports 5227x, 525x, 537x, 5445x
ISP1181 also supports ISP1161 and ISP1362
ISP158x also supports ISP1761
IAR = IAR EWARM; CW = CodeWarrior

Data Size

All RAM used by smxUSBD for data is pre-allocated from the heap during initialization. Following is a table of RAM usage:

Component	**Size** (KB)
Core	1.5
Analog Devices BF5xx	0.5
Atmel AT91	0.5
Freescale CF5329, 525x	1
Freescale CF548x	1
Freescale MX1	0.5
NXP ISP1181	0.5
NXP ISP158x	1
NXP LPCxxxx	0.5
Sharp LH7A400/4	0.5
STMicro STR7/9	0.5
Audio driver	2
Mass Storage driver	2
Mouse driver	0.5
Serial driver (each port)	1
RNDIS driver	2
Composite driver	0.5

Performance

Mass Storage

The following table shows mass storage performance using a RAM disk in the device.

Device Controller	File Read (KB/sec)	File Write (KB/sec)
BF5xx	5000	5000
ISP1181	1071	1071
ISP158x	5300	3890

RNDIS

The following table shows Ethernet over USB performance for the indicated packet size.

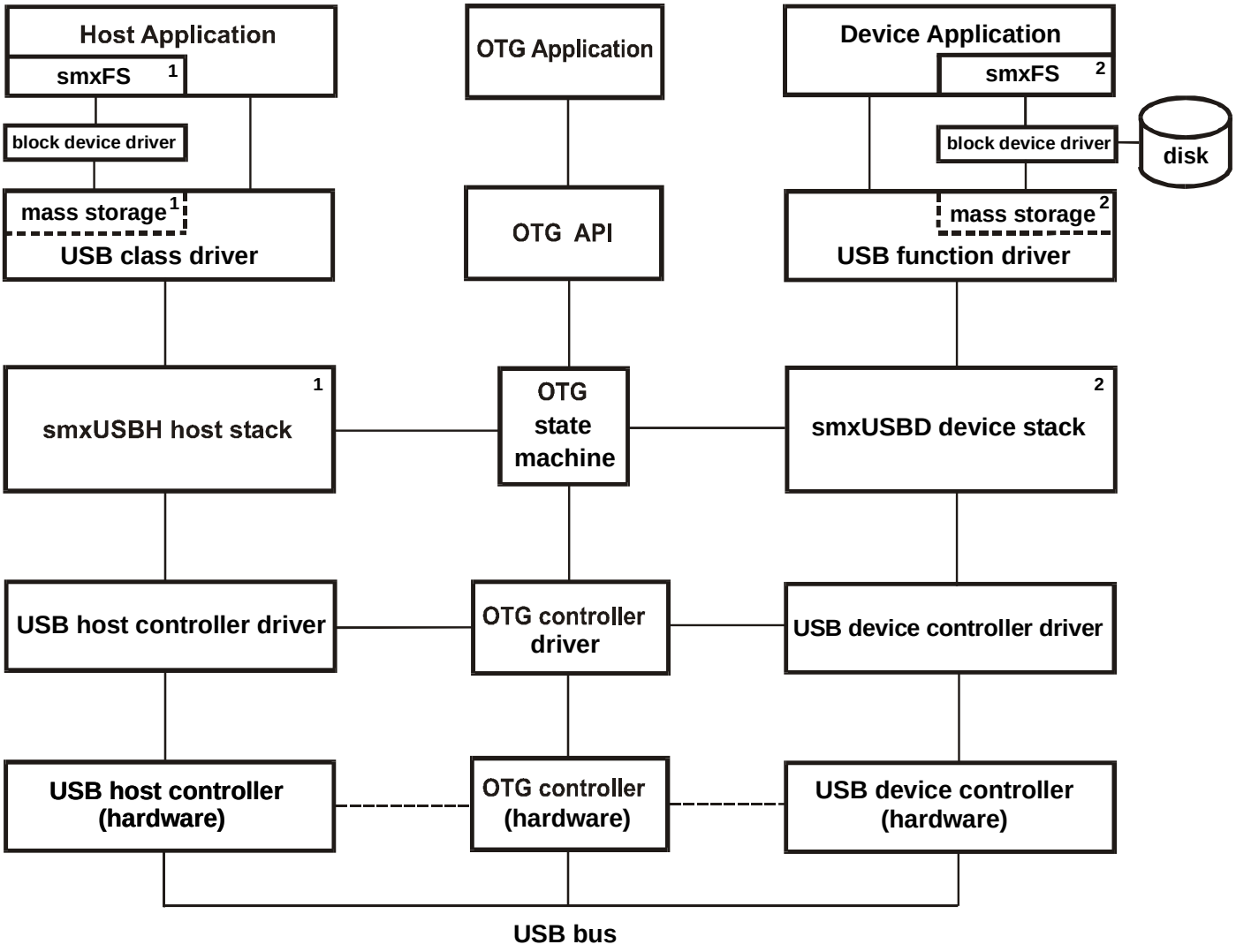
Device Controller	Packet Size (Bytes)	Send/Receive (KB/sec)
CF532x/7x	512	256

Serial

The following table shows the transfer rates for sending and receiving serial data for different application packet sizes.

Device Controller	Packet Size (Bytes)	Rate (KB/sec)
BF5xx (HS)	256	800
BF5xx (HS)	1024	2500
ISP1362 (FS)	64	140
ISP1362 (FS)	256	460
ISP1362 (FS)	512	804
ISP1362 (FS)	1024	887
ISP1761 (FS)	512	1830
ISP1761 (HS)	1024	2870
MCF54455 (HS)	16K	8000

smxUSB Product Overview



1 Included in USB Thumb Drive Bundle

2 Included in USB Disk Emulator Bundle