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. The 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 mouse and keyboard function drivers, and the following are available separately: Audio, Device Firmware Upgrade (DFU), Mass Storage, Media Transfer Protocol (MTP), RNDIS (Ethernet over USB), Serial, and Video. 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 connection type 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 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.

Features

- Supports all four USB data transfers (control, bulk, isochronous, and interrupt).
- Compliant with USB Specification 2.0.
- Function Drivers are available for audio, DFU, HID Comm, keyboard, mass storage, mouse, MTP, RNDIS (Ethernet over USB), serial, and video. These are compatible with Windows, Mac OS, Linux drivers.[1]
- Minimum code footprint 7 KB for ARM Cortex-M, including controller driver.
- Minimum RAM footprint 5 KB.
- Multi-port serial using a custom Windows driver is available.
- Composite device support.
- Compatibility with ARM/Cortex, Blackfin, ColdFire, PowerPC, x86, and other CPUs.
- Supports 16-bit addressing CPUs such as TI TMS320C55xx DSPs.
- NXP ISP1161, 1181, 1362, 158x, and 176x device controller support
- On-chip device controller support for: Analog Devices Blackfin BF5xx, Atmel AT91, Freescale ColdFire, Kinetis, and i.MX, Maxim, Netchip, NXP LPCxxxx, Renesas uPD720150, STMicro STR7/9 and STM32, Synopsys DWC, and TI AM1x/35x, LMxx.
- Written entirely in ANSI-C.
- Driver template for new platform porting
- Optimized for SMX® RTOS.
- Easily portable to other RTOSs.
- Also runs stand-alone.

[1]RNDIS is not supported by Mac OS

Layers

- **Function Driver Layer** provides USB functions to an application such as audio, mass storage, serial, and video.
- **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. See the smxUSBD User's Guide for any limitations and implementation you must do for these.

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)

DFU

The DFU function driver allows updating the firmware in your device. The Windows operating system does not have a built-in driver for it, so you need to use a driver, such as the one provided by MDI or others that are commonly available. The DFU runtime function may be used as part of a composite device with other functions such as serial or mass storage. This is not a full firmware update solution. See the smxUSBD User's Guide for details.

sud_DFUIsConnected()
sud_DFURegisterInterface (*pIF)
sud_DFUWriteDone (result, condition)
sud_DFUIsRuntimeMode()

HID Communication

The HID Communication function driver is useful for transferring small amounts of data. It is an alternative to the Serial function driver that does not require installing a driver or .inf file on Windows. It requires writing a special Windows application to communicate with it; a sample is provided.

sud_HIDIsConnected()
sud_HIDSendInput(*pDataBuf, size)
sud_HIDRegisterOutputNotify(handler)

Keyboard (Included)

The Keyboard function driver makes your device look like an HID keyboard to the USB host. It inputs key events to your PC.

sud_KBDInput(modifier, *key, count)

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(pDiskOper, lun)
```

Media Transfer Protocol (MTP)

The Media Transfer Protocol (MTP) function driver includes PTP support. It makes your device look like a digital still image device to the USB host. There is no need to install any driver or .inf file in Windows to support this device, but you may need to implement the file system interface (if not using smxFS) and the interface to get the properties of the files your device will support, such as the image width, height, etc.

MTP can also be used for general file transfer as an alternative to Mass Storage. Advantages are the ability to limit which files are accessed and that it is unnecessary to have two modes of operation to switch between local and USB access to the disk, to protect the file structure.

```
sud_MTPIsConnected(port)
sud_MTPRegisterInterface(pObjOper)
sud_MTPSendEvent(EventCode, iNumPar, *Par)
```

Mouse (Included)

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

```
sud_MouseInput(button, 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.

```
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_SerialIsPortConnected(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)
```

Video

The Video function driver makes your device look like a web camera to a Windows, Mac OS, or Linux USB host. There is no need to install any driver or .inf file in Windows to support this device but you may need to implement the camera sensor driver, and you may also need to customize the configuration of the driver for your real hardware features, such as the set of features your camera sensor will support.

```
sud_VideoIsConnected(port)
sud_VideoSendVideoData(port, pData, len)
sud_VideoGetVideoDataLen(port, len)
sud_VideoRegisterNotify(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 the 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 provides a USB device controller driver template and a section in the manual, to help you write a new driver if it does not support yours.

Porting

Little or no porting is necessary when smxUSBD is used with SMX®. It is designed to also work with other RTOSs and to run standalone, but it works best in a multitasking environment. The RTOS porting layer is handled by smxBase.

The hardware porting layer consists of two files, udport.h and udport.c. These files contain definitions, macros, and functions to port 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.

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 Thm2** IAR (KB)	**ARM** IAR (KB)	**BF** VDSP (KB)	**CF** CW (KB)
Core	5	4	8	12	9
Audio drv	3	3	6	3.5	6.5
DFU drv			2		
HID Cm drv	0.5	0.5	1	1	1
Keybrd drv	0.5	0.5	1	1	1
Mass St. drv	3.1	3	5	5.5	5
Mouse drv	0.5	0.5	1	1	1
MTP drv			8		
RNDIS drv	2.5	2	3.5	2.5	4.7
Serial drv	1.5	1.5	2.5	2.5	2.7
Video drv		11	16		
Composite	0.5	0.5	1	1	1
Analog Dev BF5xx	—	—	—	3.3	—
Atmel AT91	2	—	3	—	—
Atmel AT91 HS			4	—	—
Freescale CF5225x/1x /2x, Kxx	—	4	—	—	5
Freescale CF532x/7x, 525x,5445x, i.MX31			3.5	—	4
Maxim MAX342x			3.5		
NXP ISP176x					
NXP LPC31xx			3.5	—	—
NXP LPCxxxx	2.6	2	4	—	—
PLX Net2272				4.5	
STMicro STR7/9, STM32F103	2.5		4	—	—
Synopsys, STMicro STM32F107 STM32F20x		2.5			
TI AM1x, AM35x, LM3S		2	3	—	—

IAR = IAR EWARM; CW = CodeWarrior; VDSP = VisualDSP

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
Audio driver	2
DFU driver	1
HID Communication driver	0.5
Keyboard driver	0.5
Mass Storage driver	2
Mouse driver	0.5
MTP driver	6+
RNDIS driver	2
Serial driver (each port)	1
Video driver (full speed)	4
Video driver (high speed)	7
Composite driver	0.5
Analog Devices BF5xx	0.5
Atmel AT91	0.5
Atmel AT91 high speed	0.5
Freescale CF5225x/1x/2x, Kxx	1
Freescale CF532x/7x, 525x, 5445x, i.MX31	1
Maxim MAX3421	0.5
NXP ISP176x	1
NXP LPC31xx	1
NXP LPCxxxx	0.5
PLX Net2272	0.5
STMicro STR7/9, STM32F101/2/3	0.5
Synopsys, STMicro STM32F105/7, STM32F20x	0.5
TI AM1x, AM35x, LM3S	0.5

MTP data size is 6KB + ObjectsNum*64

Stack Size

smxUSBD has one internal task in the device controller driver that uses about 1KB stack (or 2KB for MTP). Application tasks typically use 0.5 to 1.5KB depending on the function driver.

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 (HS)	5000	5000
ISP1181 (FS)	1071	1071
ISP158x (HS)	5300	3890

RNDIS

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

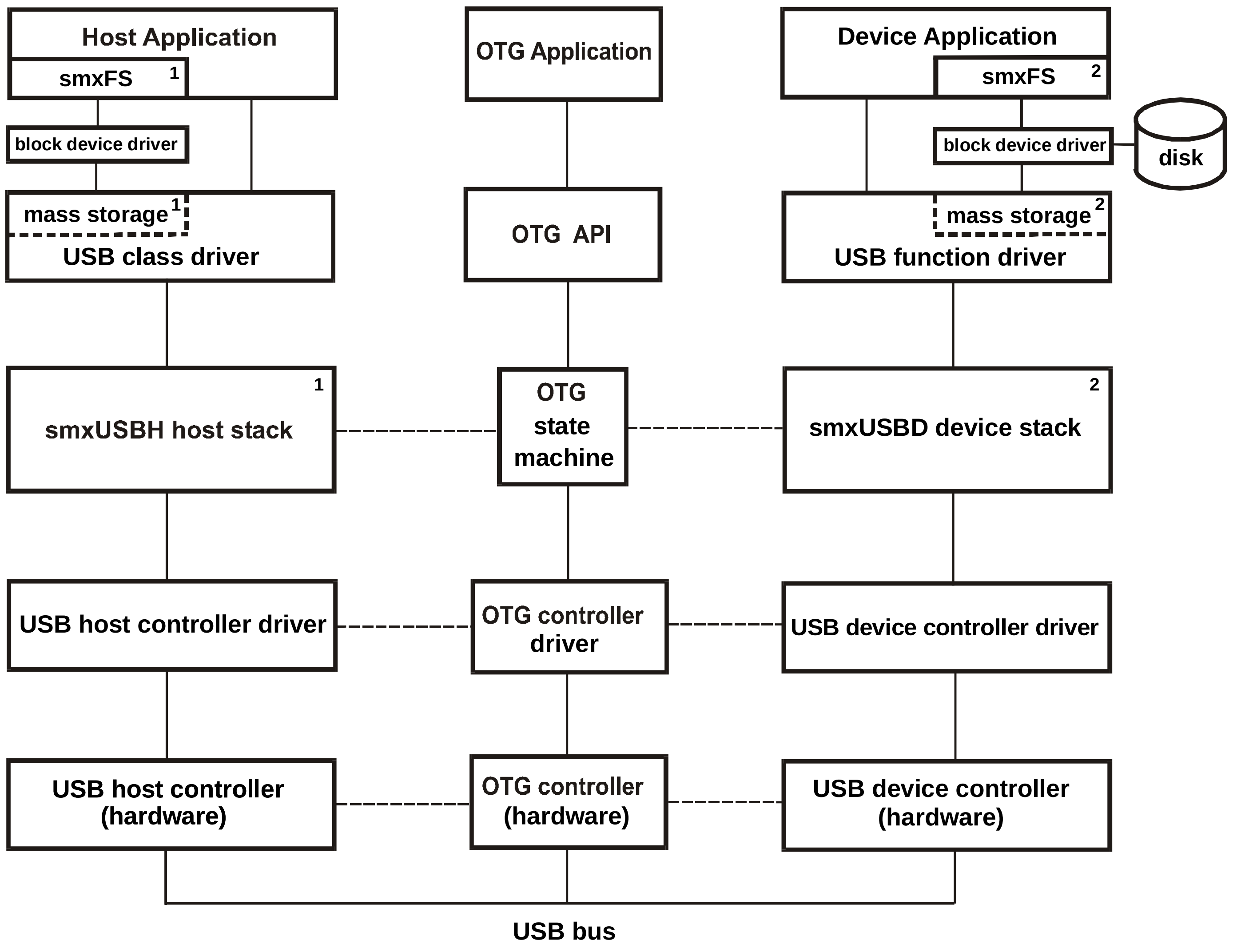
Device Controller	Packet Size (Bytes)	Send/Receive (KB/sec)
CF532x/7x (HS)	512	256

Serial

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

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 Illustration



1 Included in USB Thumb Drive Bundle

2 Included in USB Disk Emulator Bundle