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 projects. Normally this is done to permit connection to a PC or laptop in order to upload or download data, tables, code, etc. *smxUSBD* device stack is offered separately from the s*mxUSBH* 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, s*mxUSBD* includes three function drivers: serial, mouse, and mass storage. Each is compatible with the corresponding Windows USB function driver. Thus, a device using *smxUSBD* does not require a custom Windows 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.

Features

- Function Drivers are available for serial, mouse, and mass storage. These are compatible with Windows class drivers.
- Composite Device support.
- Compatibility with ARM, ColdFire, PowerPC, x86, and other CPU's.
- Compliant with USB Specification 2.0.
- Philips ISP1161, 1181, 1362, 158x, and 1761 device controller support
- Atmel AT91, Freescale ColdFire, Sharp LH7A4xx, and STMicro STR7 on-chip device controller support. Others being developed.
- Written entirely in ANSI-C.
- Porting, integration, and development services are available.
- Typical code footprint is less than 22 KB for a CISC processor.
- Optimized for SMX® RTOS.
- Easily portable to other RTOSs.
- Also runs stand-alone.

Layers

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

Driver Key to Following Tables

ISP1181 also supports ISP1161 and ISP1362
ISP158x also supports ISP1761

Code Size

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

Component	ARM IAR (KB)	CF CW (KB)
Core	9.5	9.5
Mass Storage Emulator	5	5
Mouse Emulator	1	1
Serial Emulator	2.5	2.5
Composite Driver	1	1
Philips ISP1181 Driver	N/A	4
Philips ISP158x Driver	N/A	N/A
Atmel AT91 Driver	3	—
STMicro STR7 Driver	4	—

Data Size

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

Device Controller	Core + Mass Storage	Core + Mouse	Core + Serial
ISP1181	6	2	4
ISP158x	7	2.5	5
AT91	6	2	4
STR7	6	2	4

Performance

Serial

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

Device Controller	Packet Size (Bytes)	Rate (KB/sec)
ISP1181, AT91, SR7	64	125
ISP1181, AT91, SR7	256	240
ISP1181, AT91, SR7	512	289
ISP158x	1024	645

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)
ISP1181	1071	1071
ISP158x	5300	3890

Function Driver API

Mass Storage

sud_MSRegisterDisk(pdiskop, lun)

Mouse

sud_MouseInput(key, x, y, wheel)

Serial

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 mouse and mass storage. See the smxUSBD User's Guide for more discussion.

Writing New Drivers

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

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

Porting

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 particular target. 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 particular RTOS.

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