smxUSBO^{TM} User's Guide

USB OTG Add-on

Version 1.10
August 8, 2011

by Yingbo Hu

1. Overview

smxUSBO™ adds On-The-Go capability to smxUSBH and smxUSBD for the SMX® RTOS. It is written in C and can be easily ported to another RTOS (see Appendix A. Porting smxUSBO to Another OS). It allows embedded system developers to easily add USB OTG capabilities to their projects. Normally this is done to enable the device to be a dual-role device.

An OTG device is a dual-role device that can function both as a limited-capability host and as a USB peripheral. The targeted peripherals can be any combination of other OTG devices and peripheral-only devices. smxUSBO requires smxUSBH and/or smxUSBD; it only adds OTG capability to them.

Note: The purpose of OTG is to switch the role of not only the local device but also the connected (peer) device. You cannot switch the role of a PC, thumb drive, or other device. The only case where OTG is needed is if the peer device is OTG. There is a lot of confusion here because many processors have OTG controllers. In most cases customers' needs are satisfied using just the smxUSBH and smxUSBD stacks, and maybe a little cable detection logic if the port on the board is a mini-AB or micro-AB connector. Please discuss your needs with Micro Digital for advice.

The reader should be familiar with the USB 2.0 specification and OTG 1.0a supplement. All USB specifications can be found at http://www.usb.org/.

smxUSBO has three layers:

1. **OTG Core Layer**: Provides the common OTG SRP and HNP state machine and APIs used by the application to switch between Host and Device stack for the dual-role device.
2. **OTG Controller Driver (OCD) Layer**: Provides a unique interface for different USB OTG controllers such as ISP1362, ISP1761, and LPC3180.
3. **Porting Layer**: Provides functions related to the hardware, operating system, and compiler.

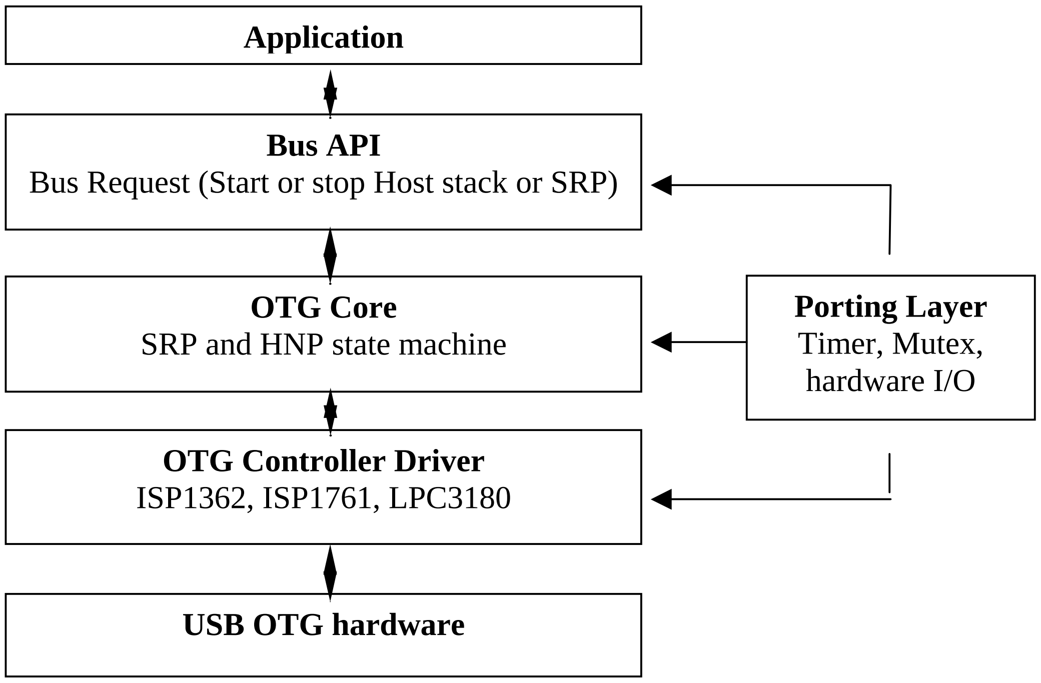
Application
Bus API
Bus Request (Start or stop Host stack or SRP)
OTG Core
SRP and HNP state machine
Porting Layer
Timer, Mutex, hardware I/O
OTG Controller Driver
ISP1362, ISP1761, LPC3180
USB OTG hardware

Figure1 smxUSBO Structure

2. Files

Like other SMX® RTOS products, all source code for smxUSBO is stored in its own directory, named “XUSBO”, under the main SMX directory. Here is a summary of the directory structure.

2.1 Directory Structure

```
SMX
    APP
        DEMO      usbodemo.c (for SMX)
        NORTOS    Build directory for standalone (non-SMX) releases. Has demo too.
    XUSBO         Configuration and porting layer files
        XX.YY     Build directory for SMX releases
        Core      OTG core layer support
        OCD       All OTG controller drivers
```

2.2 Files

2.2.1 Main Files

FILE	DESCRIPTION
smxusbo.h	smxUSBO API header file. Use in application files.
uocfg.h	smxUSBO configuration file. Allows enabling/disabling main components of smxUSBO.
uoinit.h uoinit.c	Initialization of smxUSBO, including the selected OTG controller.
uointern.h	Main internal header file. Included by smxUSBO files rather than including individual header files, to ensure files are included in the proper order.

2.2.2 Porting Layer

FILE	DESCRIPTION
uoport.h uoport.c	Porting functions. OS- and compiler-related functions are based on smxBase.

2.2.3 OTG Core

FILE	DESCRIPTION
uoapi.h uoapi.c	APIs used by the application for bus requests of Host and Device stacks.
uodrv.h uodrv.c	USB OTG state machine.

2.2.4 OTG Controller Driver

FILE	DESCRIPTION
uo1362.h uo1362.c	NXP/Philips ISP1362 OTG controller driver.
uo1761.h uo1761.c	NXP/Philips ISP1761 OTG controller driver.
uolpc2.h uolpc2.c	NXP/Philips LPC2468 built-in OTG controller driver.
uolpc3.h uolpc3.c	NXP/Philips LPC3180 built-in OTG controller driver.

3. smxUSBO Library and Demos

This section documents details of configuring and building the library and demos.

3.1 smxUSBO Configuration

3.1.1 uocfg.h

smxUSBO can be configured so that it includes support for specific USB OTG controllers, thus saving code space. The following sections describe the settings.

SUO_ISP1362

Set to “1” to include support for an NXP/Philips ISP1362 OTG controller. Set to “0” to exclude support.

SUO_ISP1761

Set to “1” to include support for an NXP/Philips ISP1761 OTG controller. Set to “0” to exclude support.

SUO_LPC3180

Set to “1” to include support for an NXP/Philips LPC3180 built-in OTG controller. Set to “0” to exclude support.

SUO_LPC2468

Set to “1” to include support for an NXP/Philips LPC2468 built-in OTG controller. Set to “0” to exclude support.

Note: Please only set ONE OTG controller macro to “1”. smxUSBO does not currently support the use of multiple types of OTG controller at the same time.

SUO_ONCHIP_CP

Set to “1” to use on-chip charge pump. Set to “0” to disable. On-chip charge pump can only provide up to 8-mA current.

SUO_SUPPORT_HNP

Set to “1” if your OTG device support dual-role mode. Set to “0” if it only works as peripheral.

SUO_DEBUG_LEVEL

Specifies the debug level. The following values are supported:

0 disables all debug output and debug statements are null macros

1	only output fatal error information
2	output additional warning information
3	output additional status information
4	output additional device change information
5	output additional data transfer information
6	output interrupt information

3.1.2 uoport.h

The general interrupt-related porting interface is defined in smxBase. Please see the smxBase User's Guide for detailed information. The following are smxUSBO-specific hardware-related configuration settings. In addition to these, it may be necessary to re-implement the functions in uoport.c for your hardware.

SUO_ISP1362_BASE, SUO_ISP1362_IRQ

Set to the base address and IRQ number for your device controller. These do not need to be set for x86 PCI systems because this information is retrieved from the PCI BIOS.

3.2 Building the Library

After configuring uocfg.h (see 3.1.1 uocfg.h), build the library with the makefile or project file supplied in the build directory (e.g. MC.P3). It is built just like all other SMX module libraries, as documented in the SMX Quick Start. If a makefile is provided, run the mak.bat file to invoke it. Run it without arguments for syntax.

3.3 Building and Running the Demos

For SMX releases, the demos are stored in \SMX\APP\DEMO. For this product it is:

usbodemo.c	main demo file

The demo file is integrated with the smx Protosystem. It is enabled just like all other SMX module demos, as documented in the SMX Quick Start. For makefile builds, simply uncomment the macros **susbo** in pro.mak and **susbodm** in demodefs.mki. You also need to enable the smxUSBH and smxUSBD libraries and demos by uncommenting the macros **susb** and **susbd** in pro.mak and **susbdm** and **susbddm** in demodefs.mki.

You need two USB OTG (dual-role) devices to run the demo to show how a dual-role device can work as a Host and then later switch to a Peripheral, without needing to change the cable. You also need a special OTG cable. For this cable, one plug is mini-A and the other is mini-B.

The following is a summary of what the demo does:

1. Connect the two devices with an OTG cable. A-Device or B-Device is determined by the plug type for the OTG cable. If mini-A is plugged into a Device then it is the A-Device by default, otherwise, it is the B-Device.
2. After the demo runs, both devices are in idle mode. On the A-Device, the demo shows status “A-Device Host Idle” and on the B-Device, it shows status “B-Device Peripheral Idle”. On the A-Device, press the 'R' key to make it work as USB Host, and B-Device will become USB Peripheral. At that time the demo on A-Device shows “A-Device Host Busy” and B-Device it shows “B-Device Peripheral Busy”. Press 'D' key on A-Device to stop it from working as USB Host, and both A-Device and B-Device will go to idle mode.
3. On the B-Device, in idle mode, press the 'R' key to make B-Device work as the USB Host (the demo shows “B-Device Host Busy”), and A-Device will be switched to USB Peripheral (the demo shows “A-Device Peripheral Busy”). Press the 'D' key to switch the B-Device back to Peripheral (the demo will show “B-Device Peripheral Busy”), and A-Device will return back to Host (the demo shows “A-Device Host Busy”).

3.4 Initialization

smxUSBO is automatically initialized by an SMX application, if SMXUSBO is defined by the application makefile or project file. This is done by smxusbo_init() which is called by smx_modules_init(), called by ainit(). For non-SMX applications, call suo_Initialize() from your initialization code.

4. OTG Application Interface

This section describes the APIs are called by application to enable and switch the role of USB OTG port.

The application interface is defined in **uoapi.h**. This interface includes:

```
int     suo_ARequestHost();
int     suo_AGoIdle();
int     suo_BRequestHost();
int     suo_BReturnDevice();

int     suo_GetDeviceType();
int     suo_GetStackType();
int     suo_GetIdleType();
```

int **suo_ARequestHost** ()

Summary Switch the A-Device from a_idle state to a_host state.

Details The A-Device is initialized to a_idle state by default. This function starts the OTG port as a USB host port, and then smxUSBH will begin its work to enumerate the USB device on the B-Device plug of the OTG cable. Call this function only when it is an A-Device and the bus is idle.

Parameters none

Returns

1	A-Device is working as a Host.
0	Failed.

See Also suo_AGoIdle(), suo_GetDeviceType(), suo_GetIdleType()

int **suo_AGoIdle** ()

Summary Switch the A-Device back to a_idle state.

Details When the A-Device is working as a USB Host (a_host), call this function to make it switch back to idle mode (a_idle). This device will not use USB and suspends the bus. Call this function only when it is an A-Device and the bus is busy. Also do not call any smxUSBH APIs after this function call.

Parameters none

Returns

1	A-Device has switched back to idle mode.
0	Failed.

See Also suo_ARequestHost(), suo_GetDeviceType(), suo_GetIdleType()

int **suo_BRequestHost** ()

Summary Switch the B-Device from b_idle state to b_host state.

Details The B-Device is initialized to b_idle state by default. This function starts the OTG port as the USB host port, and then smxUSBH will begin its work on the B-Device to enumerate the USB device on the A-Device plug of the OTG cable. Call this function only when it is a B-Device and the bus is idle.

Parameters none.

Returns

1	B-Device has switched to Host mode.
0	Failed.

See Also suo_BReturnDevice(), suo_GetDeviceType(), suo_GetIdleType()

int **suo_BReturnDevice** ()

Summary	Switch the B-Device back to b_peripheral state.
Details	When the B-Device is working as a USB Host (b_host), call this function to make it switch back to Peripheral mode (b_peripheral). At the same time, the A-Device will return back to Host mode and continues using the USB. The A-Device must call suo_AGoIdle() to suspend the USB completely. Call this function only when it is a B-Device and the bus is busy.
Parameters	none
Returns	1 B-Device has switched back to Peripheral mode. 0 Failed.
See Also	suo_BRequestHost(), suo_GetDeviceType(), suo_GetIdleType()

int **suo_GetDeviceType** ()

Summary	Get current device type, A or B Device.
Details	Call this function to get the current device type. Device type is determined by the plug types of the OTG cable. If mini-A is plugged into the device then it is an A-Device by default; otherwise, it is a B-Device.
Parameters	none
Returns	SUO_A_DEVICE A-Device. SUO_B_DEVICE B-Device.
See Also	suo_GetStackType(), suo_GetIdleType()

int **suo_GetStackType** ()

Summary	Get current stack type (Host or Device) that is controlling this port.
Details	Call this function to determine which stack is controlling this port. Because an OTG device is a dual-role device, either a Host stack or a Device stack can control the OTG port, based on the application request.
Parameters	none
Returns	SUO_HOST — USB Host stack is controlling the OTG port. SUO_DEVICE — USB Device stack is controlling the OTG port.
See Also	suo_GetIdleType(), suo_GetDeviceType()

int **suo_GetIdleType** ()

Summary	Get current bus status, idle (bus suspended) or busy.
Details	Call this function to get the current bus status. If a Host stack or a Device stack is using the bus then it returns SUO_BUSY. Otherwise, if the host stack suspends the bus, it returns SUO_IDLE.
Parameters	none
Returns	SUO_IDLE — USB Host stack suspended the bus. SUO_BUSY — USB Host and Device stack are using the USB bus.
See Also	suo_GetDeviceType(), suo_GetStackType()

5. Writing a New OTG Controller Driver

This section describes how to write a new smxUSBO OTG controller driver.

The USB OTG controller drivers are a sub-module of smxUSBO. You do not need to know the details about the other parts such as the smxUSBO core layer.

To add a new USB OTG driver, the only thing you need to do is implement an OTG Controller operation interface and then register it with the smxUSBO core layer in uoinit.c, by calling suo_RegisterOCD(). This function should be called after any other smxUSBO initialization function calls.

Micro Digital does not recommend that you write an OTG controller driver yourself. Please discuss this with Micro Digital before you decide to do it.

5.1 OTG Controller Operation Interface

The device controller operation interface is defined in uodrv.h as:

```
typedef struct
{
   void (*OCInit) (void);
   void (*OCRelease) (void);
   void (*OCPullDown) (uint bTrue);
   void (*OCPullUp) (uint bTrue);
   void (*OCSOF) (uint bTrue);
   void (*OCVbusPowerUp) (uint bTrue);
   void (*OCVbusSessionReq) (uint bTrue);
   void (*OCVbusDischarge) (uint bTrue);
   void (*OCHCDCSelect) (uint bTrue);
   void (*OCEnableDetSRP) (uint bTrue);
   void (*OCEnableRemoteDisconnect) (uint bTrue);
   void (*OCEnableRemoteSE0) (uint bTrue);
   void (*OCStartTimer) (u32 uTimerout);
   void (*OCStopTimer) (void);
   void (*OCStatus) (SUO_Otg_Hangle_T *);
   void (*OCClearIRQ) (u16 wIRQ);
   void (*OCSelectVbus) (uint iOnChip);
   void (*OCEnableDCIRQ) (uint bEnable);
   void (*OCIntHandler) (SUO_Otg_Hangle_T *);
}SUO_OC_OPERATION_T;
```

5.1.1 OCInit()

The smxUSBO core layer calls this function first to do any necessary OTG Controller initialization.

5.1.2 OCRelease()

The smxUSBO core layer calls this function before it shuts down, to do any necessary OTG Controller cleanup.

5.1.3 OCPullDown()

The smxUSBO core layer calls this function to enable/disable the pull down resistor of the DM pin.

5.1.4 OCPullUp()

The smxUSBO core layer calls this function to enable/disable the pull up resistor of the DP pin.

5.1.5 OCSOF()

The smxUSBO core layer calls this function to enable/disable host Start of Frame of this port. Disabling the SOF will suspend the bus after 3 ms.

5.1.6 OCVbusPowerUp()

The smxUSBO core layer calls this function to enable/disable the Vbus power.

5.1.7 OCVbusSessionReq()

The smxUSBO core layer calls this function to start/stop charging the Vbus.

5.1.8 OCVbusDischarge()

The smxUSBO core layer calls this function to start/stop discharging the Vbus.

5.1.9 OCHCDCSelect()

The smxUSBO core layer calls this function to select HC or DC to control this OTG port.

5.1.10 OCEnableDetSRP()

The smxUSBO core layer calls this function to enable/disable SRP detection for A-Device.

5.1.11 OCEnableRemoteDisconnect()

The smxUSBO core layer calls this function enable/disable remote disconnect detection.

5.1.12 OCEnableRemoteSE0()

The smxUSBO core layer calls this function to enable/disable remote SE0 status detection.

5.1.13 OCStartTimer()

The smxUSBO core layer calls this function to start a timer.

5.1.14 OCStopTimer()

The smxUSBO core layer calls this function to stop a timer.

5.1.15 OCStatus()

The smxUSBO core layer calls this function to get the current OTG status. You may need to translate the register bitmap to the OTG internal data structure.

5.1.16 OCClearIRQ()

The smxUSBO core layer calls this function to clear the OTG interrupt.

5.1.17 OCSelectVbus()

The smxUSBO core layer calls this function to set whether to use on-chip or external charge pump.

5.1.18 OCEnableDCIRQ()

The smxUSBO core layer calls this function to enable/disable the device controller interrupt.

5.1.19 OCIntHandler()

The smxUSBO core layer calls this function when an OTG interrupt occurs to handle that interrupt.

5.2 Set State Machine Status and Input Value

In the ISR of your OTG driver, it is necessary to set some OTG state machine status and input values. They are defined in SUO_OTG_HANDLE in uodrv.h, and are documented below:

```
u8  id:         1;
```

The ID pin status of the USB OTG port. Set it by reading the OTG status register in OCStatus(). You may also need to update it in the ISR if there is an ID pin interrupt.

A-Device

```
u8    a_vbus_vld:       1;
```

The VBUS Valid status of the A-device. Set it by reading the OTG status register in OCStatus(). You may also need to update it in the ISR if there is a VBUS Valid interrupt. 1 if VBUS is higher than VA_VBUS_VLD.

```
u8    a_sess_vld:       1;
```

The SESSION Valid status of the A-device. Set it by reading the OTG status register in OCStatus(). You may also need to update it in the ISR if there is Session Valid interrupt. 1 if VBUS is higher than VA_SESS_VLD.

```
u8    a_srp_det:        1;
```

Flag showing if A-device has detected SRP. You may need to update it in the ISR if there is SRP detect interrupt, or set it according to the session valid status.

```
u8    b_conn:           1;
```

Remote B-device connection status. See the OTG specification section 6.6.1.12 b_conn to understand how to set it.

```
u8    b_bus_suspend:    1;
```

Flag showing if the bus is suspended for the remote B-device. Normally there is an interrupt in the device controller or OTG controller for this. Set it in the ISR or by reading the OTG status register.

```
u8    b_bus_resume:     1;
```

Flag showing if the bus is resumed for the remote B-device. Normally there is an interrupt in the device controller or OTG controller for this. Set it in the ISR or by reading the OTG status register.

```
u8    b_hnp_support:    1;
```

Flag for remote B-device. It is an external input. The host stack will check the OTG descriptor to find if the remote B-device supports HNP, and then it calls suo_SetBDevAttributes() to set this flag. This is not related the OTG controller so you don't need to set it in the OCD.

u8   b_srp_support: 1;

Flag for remote B-device. It is an external input. The host stack will check the OTG descriptor to find if the remote B-device supports SRP, and then it calls suo_SetBDevAttributes() to set this flag. This is not related the OTG controller so you don't need to set it in the OCD.

u8   a_set_b_hnp_en: 1;

Flag for A-device. The host stack will set this bit in suo_AEnableBHost() to enable the remote B-device to start HNP after the A-device finishes its role. It is not related to the OTG controller so you don't need to set it in the OCD.

u8   a_suspend_req: 1;

Flag for A-device. The application requests to suspend the bus. Not used yet.

u8   a_host_done: 1;

Flag for A-device. The A-Device is still using the bus. It is not related to the OTG controller so you don't need to set it in the OCD.

B-Device

u8   b_sess_end: 1;

The SESSION End status of the B-device. Set it by reading the OTG register in OCStatus(). You may also need to update it in the ISR if there is a Session End interrupt. 1 if VBUS is less than VA_SESS_END.

u8   b_sess_vld: 1;

The SESSION Valid status of the B-device. Set it by reading the OTG register in OCStatus(). You may also need to update it in the ISR if there is a Session Valid interrupt. Set it to 1 if VBUS is higher than VB_SESS_VLD.

u8   b_se0_srp: 1;

Flag bit is set to 1, when the B-device is in the SUO_B_IDLE state, and the line has been at SE0 for more than the minimum time (2ms) before generating SRP (TB_SE0_SRP).

u8   a_conn: 1;

Remote A-device connection status. See the OTG specification section 6.6.1.5 a_conn to understand how to set it.

```
u8    a_bus_suspend:    1;
```

Flag showing if the bus is suspended by the remote A-device. Normally there is an interrupt in the device controller or OTG controller for this. Set it in the ISR or by reading the OTG status register.

```
u8    a_bus_resume:     1;
```

Flag showing if the bus is resumed by the remote A-device. Normally there is an interrupt in the device controller or OTG controller for this. Set it in the ISR or by reading the OTG status register.

```
u8    a_bus_reset:      1;
```

Flag showing if the bus is reset by the remote A-device. Normally there is an interrupt in the device controller or OTG controller for this. Set it in the ISR or by reading the OTG status register.

```
u8    b_srp_done:       1;
```

Flag showing if the B-device has finished its SRP procedure.

```
u8    b_hnp_en:         1;
```

Flag showing if the remote A-device used SET_FEATURE request to enable HNP. It is not related to the OTG controller so you don't need to set it in the OCD.

```
u8    a_hnp_support:    1;
```

Not used.

```
u8    a_alt_hnp_support:1;
```

Not used.

Both A-Device and B-Device

```
u8    bus_req:          1;
```

Flag showing if the device wants to use the bus. It is not related to the OTG controller so you don't need to set it in the OCD.

```
u8    bus_drop:         1;
```

Flag showing if the device wants to drop the bus (i.e. finish using the bus). It is not related to the OTG controller so you don't need to set it in the OCD.

6. Hardware Porting Notes

The general interrupt-related hardware porting interface is done through smxBase. The hardware porting layer which is smxUSBO-specific consists of uoport.h and uoport.c. These files contain definitions, macros, and functions to port smxUSBO to particular target hardware.

6.1 uoport.h

smxBase takes care of defining most of the hardware porting defines.

A. SUO_BUS_INVERTED: Set this to 1 if you have a big endian CPU and you inverted the connection of the device controller to the data bus. (If you are designing your own hardware, it is recommended to do this for better performance, so there is no need to invert data i/o to OTG controller registers in smxUSBO.)

B. Driver BASE and IRQ settings: Set these to the proper addresses for your OTG controller.

6.2 uoport.c

Some of the functions in uoport.c may need to be adapted for your target.

suo_HdwInit()

This function is called first when initializing smxUSBO. It does the following:

Initializes the hardware platform's OTG subsystem. For example, it enables the USB OTG controller, sets up the clock, and finds the PCI BIOS.

Determines the OTG Controller's I/O base, memory base, and IRQ number.

Initializes other hardware required by smxUSBO. For example, it opens a serial port and sets up the parameters for the suo_DebugL() function to output debug information.

suo_HdwRelease()

Disables the USB OTG subsystem of the hardware.

suo_ISP1362Read32(), suo_ISP1362Read16(), suo_ISP1362Write32(), suo_ISP1362Write16()

These functions are used when the smxUSBO NXP/Philips ISP1362 OTG controller driver accesses I/O registers. **You may need to tune the implementation of these functions to meet the timing requirement of ISP1362 according to your hardware implementation.**

Appendix A. Porting smxUSBO to Another OS

smxUSBO’s porting layer maps onto smxBase services. Please see the smxBase User’s Guide for detailed information to port it to another OS.

Appendix B. Memory Usage

B.1 Code Size

Code size will vary widely depending upon CPU, compiler, and optimization level. The figures below are intended as an example.

Component	**ARM9** **IAR v4.31**	**CF** **CodeWarrior v5.0**	**X86** **VC++ v6.0**
Core	5 KB	5 KB	4.5 KB
NXP/Philips ISP1362	2 KB	2 KB	1.7 KB
NXP/Philips ISP1761	2 KB	2 KB	1.8 KB
NXP/Philips LPC2468	3 KB	N/A	N/A
NXP/Philips LPC3180	3 KB	N/A	N/A

B.2 Data Size (RAM Requirement)

smxUSBO needs less than 128 bytes for global data. It does not need to allocate dynamic memory.

Appendix C. Specification Reference

smxUSBO is based on the following specifications. USB related documents are available at www.usb.org

C.1 USB Specifications

Universal Serial Bus Specification, Revision 2.0
On-The-Go Supplement to the USB 2.0 Specification, Revision 1.0a

C.2 OTG Controller Specifications

ISP1362 Single-chip Universal Serial Bus On-The-Go controller, Rev. 04
ISP1761 Hi-Speed Universal Serial Bus On-The-Go controller, Rev. 02
LPC3180 User Manual, Rev. 01, 1 June 2006
ISP1306 Universal Serial Bus On-The-Go Transceiver, Rev. 02, 04 January 2005
LPC2468 User Manual, Rev. 01, 18 December 2006

C.3 PCI Specification

PCI Local Bus Specification, Revision 2.1