Ambarella

SDK6 API: Network Support (AmbaLink)

Version 1.6

December 10, 2015

Confidentiality Notice:

The contents of this document are proprietary and confidential information of Ambarella Inc.

The material in this document is for information only. Ambarella assumes no responsibility for errors or omissions and reserves the right to change, without notice, product specifications, operating characteristics, packaging, ordering, etc. Ambarella assumes no liability for damage resulting from the use of information contained in this document. All brands, product names and company names are trademarks of their respective owners.

For full Ambarella contact information, please visit website: www.ambarella.com.

I Contents

II Preface

This document provides technical details using a set of consistent typographical conventions to help the user differentiate key concepts at a glance.

Conventions include:

Example	Description
AmbaGuiGen, **DirectUSB** **Save, File > Save** **Power, Reset, Home**	Software names GUI commands and command sequences Computer / Hardware buttons
Flash_IO_control **da**, **status**, **enable**	Register names and register fields. For example, **Flash_IO_control** is the register for global control of Flash I/O, and bit 17 (**da)** is used for DMA acknowledgement.
GPIO81, **CLK_AU**	Hardware external pins
VIL, VIH, VOL, VOH	Hardware pin parameters
INT_O, RXDATA_I	Hardware pin signals
amb_performance_t ***amb_operating_mode_t*** **amb_set_operating_mode()**	API details (e.g., functions, structures, and type definitions)
`/usr/local/bin` `success = amb_set_operating_mode (amb_XXX_base_address, & operating_mode)`	User entries into software dialogues and GUI windows File names and paths Command line scripting and Code

Table II-1. Typographical Conventions for Technical Documents.

Additional Ambarella typographical conventions include:

- Acronyms are given in UPPER CASE using the default font (e.g., AHB, ARM11 and DDRIO).
- Names of Ambarella documents and publicly available standards, specifications, and databooks appear in *italic* type.

1 Overview

1.1 Overview: Introduction

This document defines the Ambarella Network Support (AmbaLink) application programming interface (API) module for Ambarella SDK6 digital processing products, including A9, A9S, A12 and H1. Note that this list of supported chips is subject to change. Please contact an Ambarella representative with questions regarding chip compatibility.

AmbaLink consists of modularized APIs and third-party protocols that provide users with straightforward access to, and control over, key elements of the hardware registry. The AmbaLink SDK is used to enable network support for the A9 and A12 system-on-chip (SoC).

This section overviews AmbaLink architecture and software stack as follows:

- (Section 1.1.1) Introduction: AmbaLink SDK
- (Section 1.1.2) Introduction: AmbaLink Software Stack

1.1.1 Introduction: AmbaLink SDK

Ambarella uses the Cortex-A9 dual-core and single-core processor to provide high-performance functions and low-power design. In dual-OSes systems, resource allocation is always a big concern. Therefore, in order to accommodate different requests, AmbaLink has different architecture designs to utilize the maximum power of Cortex-A9. In the following sections, the ideas and concepts behind these architectures are introduced.

1.1.1.1 AmbaLink Architecture: Dual-Core AmbaLink Architecture

Linux provides abundant functionalities with respect to network and supports many kinds of device drivers. When users need to develop powerful applications, they prefer to let Linux have more resources. Therefore, Ambarella proposes the “AmbaLink Architecture” to achieve the needs of the users.

In AmbaLink Architecture, Linux can have its own CPU. Please refer to Figure 1-1. For example, in the Ambarella A9 and H1 SoCs:

- Cortex A9 Core0: Runs the ThreadX uniprocessor real-time operating system (RTOS) for camera functions.
- Cortex A9 Core1: Runs the Linux operating system (OS) for networking functions. Inter-Process Communication (IPC) for ThreadX RTOS and Linux.

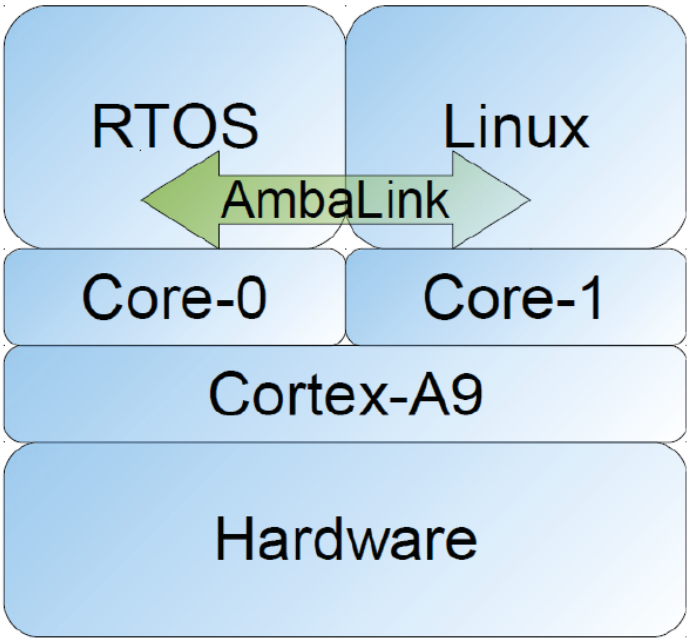

Figure 1-1. AmbaLink Running Within Cortex A9 on A9 SoC.

1.1.1.2 AmbaLink SDK: Single-Core BOSS Architecture

If most features are implemented in RTOS, Ambarella would suggest the customers to adopt “BOSS Architecture”. In this kind of architecture, Linux is a normal task running on RTOS in normal case. However, Linux can access hardware via AmbaBOSS but not via RTOS. For example, in Ambarella A12 SoC, ThreadX is running on a single core Cortex-A9 and Linux is a normal task running on ThreadX. Please refer to Figure 1-2.

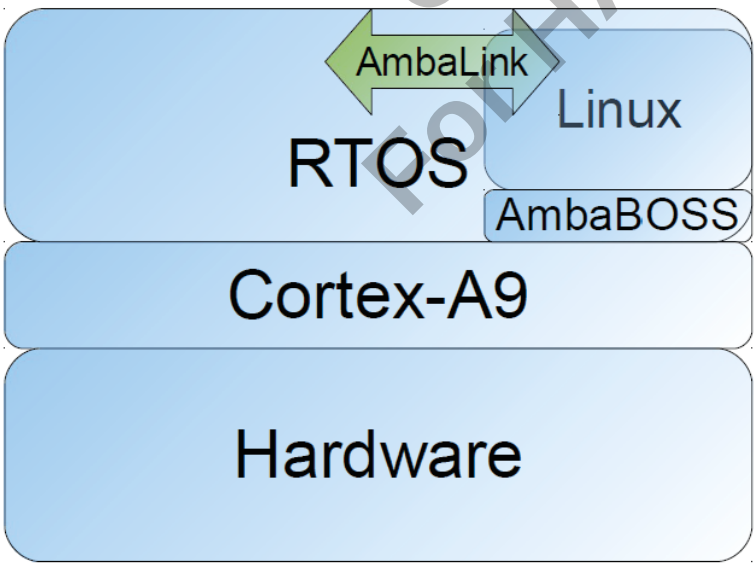

Figure 1-2. AmbaLink Running on A12 SoC.

1.1.1.3 AmbaLink SDK: SMP BOSS Architecture

Furthermore, Ambarella uses dual-core Cortex-A9 to develop the product to provide strong computing power. Therefore, “BOSS architecture” can be extended to “SMP BOSS Architecture”. However, only RTOS can run SMP mode, but Linux and AmbaBoss are bond on Core-0. For example, in the Ambarella A9S SoC, ThreadX runs on Core-0 and Core-1, but Linux only runs on Core-0. Please refer to Figure 1-3.

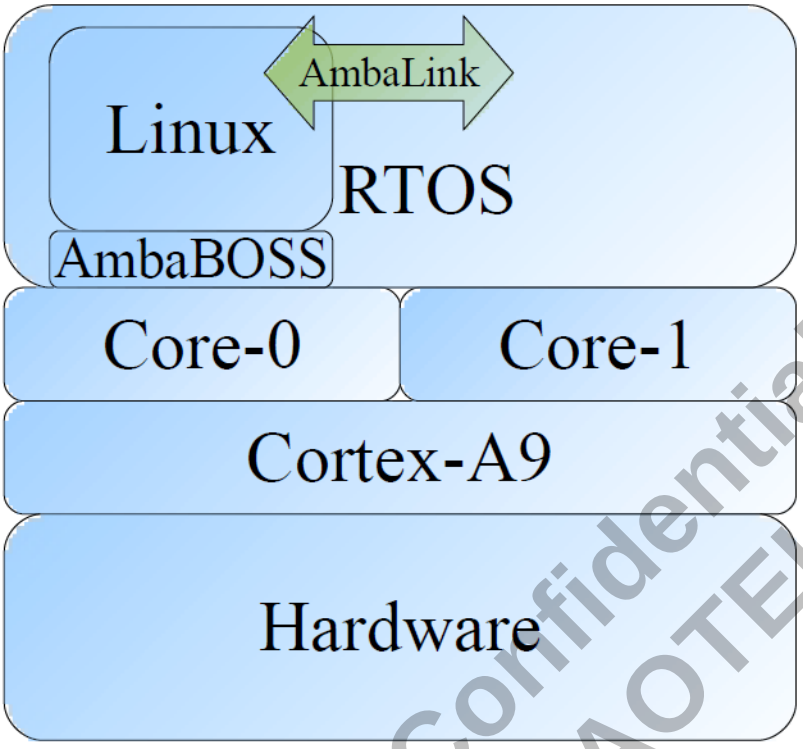

Figure 1-3. SMP BOSS Architecture.

1.1.2 Introduction: AmbaLink Software Stack

AmbaLink provides the capability to communicate between ThreadX and Linux, build RPC programs and provide-Linux boot control. AmbaLink consists of various modules shown in Figure 1-4.

Note that the Linux devices drivers, like WiFi, SDIO, etc. are parts of the AmbaLink SDK but they are not covered in this section.

1. AmbaLink

   AmbaLink provides control to boot Linux and RPMsg framework/devices initialization.

2. OS awareness driver and global lock

   Some hardware (HW) resources are shared by ThreadX and Linux. Hence, some critical sections are protected by global lock and the implementation of the global lock varies from driver to driver.

3. RPMsg framework and RPMsg devices

   RPMsg host is provided by Linux Kernel.
   RPMsg client is implemented in ThreadX.

4. Ambarella remote procedure call (RPC) framework

   Ambarella RPC framework is a client-server architecture. The functions contained within RPC are accessible by programs that communicate using the client-server methodology.

5. Virtual file systems

   AmbaFS allows Linux to access the ThreadX file system.
   RFS allows ThreadX to access the Linux file system.

6. Linux Hibernation support

   A hiber RPMsg channel is used by Linux to notify RTOS that the hibernation image is ready and then, RTOS saves the CPU state where Linux is running as well as hibernation images to a non-volatile storage device.

7. Linux Sleep support

   RTOS can notify Linux to enter the suspend-to-ram state. If the DRAM self-refresh mode is supported on the system, SOC can be powered off and boot from DRAM self-refresh mode, this is known as the warm boot. In such a mode, Linux is resumed from the suspend-to-ram state.

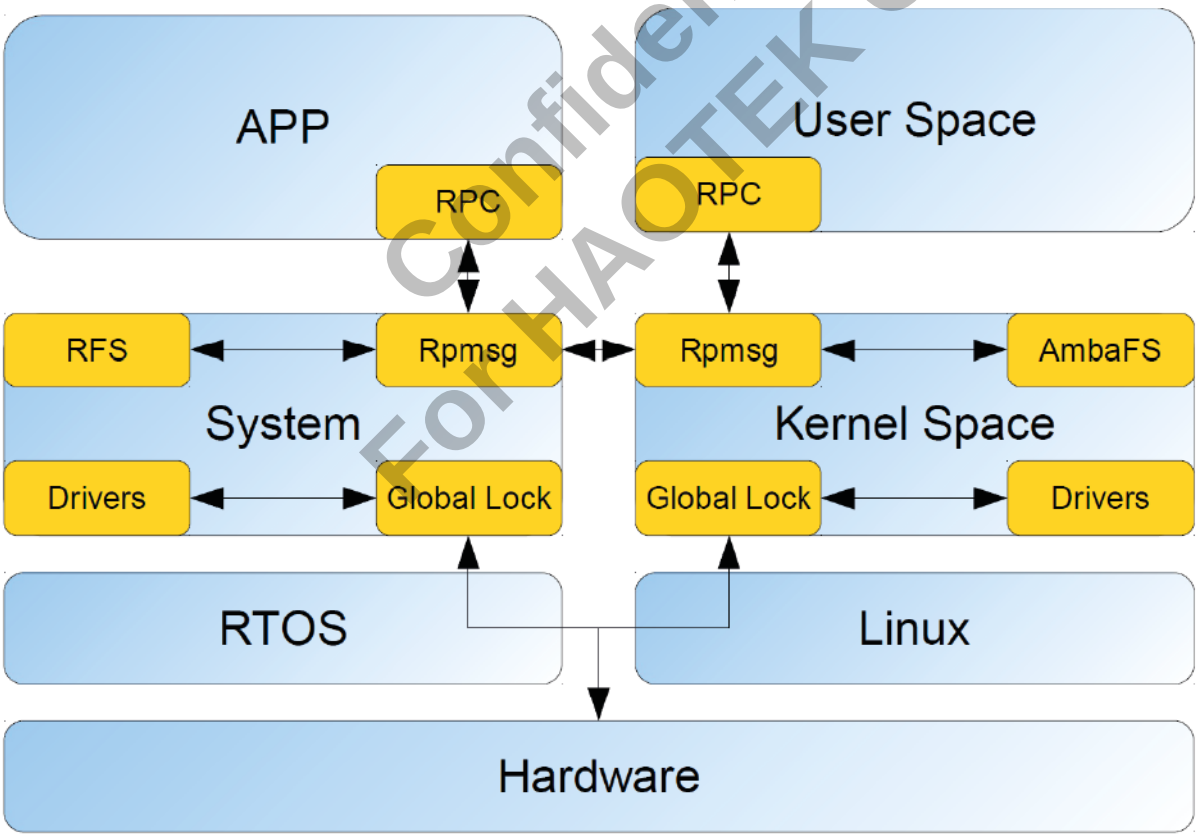

Figure 1-4. The Abstraction of AmbaLink Software Stack.

The document is organized as follows:

ThreadX:

- (Chapter 2) Spin Lock (ThreadX)
- (Chapter 3) Mutex (ThreadX)
- (Chapter 4) Remote Processor Messaging (ThreadX)
- (Chapter 5) Remote Procedure Calls (ThreadX)
- (Chapter 6) RTOS Virtual File System (ThreadX)

Linux:

- (Chapter 7) Spin Lock (Linux)
- (Chapter 8) Mutex (Linux)
- (Chapter 9) Remote Processor Messaging (Linux)
- (Chapter 10) Remote Procedure Calls (Linux)

Miscellaneous:

- (Chapter 11) AmbaLink Service Related APIs (ThreadX)

2 Spin Lock (ThreadX)

2.1 AmbaLink Spin Lock (ThreadX): Overview

This chapter discusses the AmbaLink APIs related to spin lock implementation on the ThreadX real-time operating system (RTOS).

An operating system (OS) can synchronize its internal execution with build-in synchronization primitives, such as spinlock, mutex, semaphore, event flags, message queue, completion, etc. However, if different operating systems need to share a common, system-wide resource, such as memory, I2C bus, or SD-card slot; a set of global OS-agnostic synchronization primitives are needed.

AmbaLink provides an access control scheme for all operating systems on the Ambarella platform. Once an operating system acquires a system resource through a lock, it has the exclusive access to that resource until it voluntarily releases the lock. Spin lock is one kind of lock provided and the related API is introduced in this Chapter.

A spin lock is a synchronization object used to ensure mutually exclusive access to a given thread. When a thread is enabled with a spin lock, any other thread attempting to acquire it will wait in a loop (“spinning”) while repeatedly polling whether the lock is available. The polling thread remains active and retains CPU control while spinning; however, no useful task can be performed under this condition. Therefore, it is critical that a spin lock is held for as little time as possible to minimize the unnecessary use of CPU resources.

2.2 AmbaLink Spin Lock (ThreadX): List of APIs

- AmbaIPC_SpinLock
- AmbaIPC_SpinUnlock
- AmbaIPC_SpinLockIrqSave
- AmbaIPC_SpinUnlockIrqRestore

2.2.1 AmbaIPC_SpinLock

API Syntax:

AmbaIPC_SpinLock (UINT32 SpinID)

Function Description:

- This function is used to assign a spin lock to a specified ID **@SpinID**.
- The **AmbaIPC_SpinLock** function does not disable interrupts. This function is used exclusively to lock a specified spin lock ID **@SpinID** across dual operating systems.
- Note that if an interrupt service routine (ISR) is anticipated to access a spin lock, the **AmbaIPC_SpinLockIrqSave** function should be used; otherwise, system deadlock may result.
- The following spin lock IDs are currently available:

```
typedef enum _AMBA_IPC_SPINLOCK_IDX_e_ {
    AMBA_IPC_SPINLOCK_GPIO,
    AMBA_IPC_NUM_SPINLOCK      /* Total number of global spin locks */
} AMBA_IPC_SPINLOCK_IDX_e;
```

Parameters:

Type	Parameter	Description
UINT32	**SpinID**	Spin lock ID to be locked

*Table 2-1. Parameters for Spin Lock API **AmbaIPC_SpinLock()**.*

Returns:

Return	Description
0	Completed successfully
- 1	Failed to lock the given ID

*Table 2-2. Returns for Spin Lock API **AmbaIPC_SpinLock()**.*

Example:

```
AmbaIPC_SpinUnlock(AMBA_IPC_SPINLOCK_GPIO);
```

See Also:

AmbaIPC_SpinUnlock()
AmbaIPC_SpinLockIrqSave()
AmbaIPC_SpinUnlockIrqRestore()

2.2.2 AmbaIPC_SpinUnlock

API Syntax:

AmbaIPC_SpinUnlock (UINT32 SpinID)

Function Description:

- This function is used to unlock a global spin lock with the specified ID **@SpinID**.
- The **AmbaIPC_SpinUnlock** function does not disable interrupts. This function is used exclusively to unlock a specified spin lock ID **@SpinID** across dual operating systems.
- Note that if an interrupt service routine (ISR) is anticipated to access a spin lock, the **AmbaIPC_SpinUnlockIrqRestore** function should be used; otherwise, system deadlock may result.
- The following spin lock IDs are currently available:

```
typedef enum _AMBA_IPC_SPINLOCK_IDX_e_ {
    AMBA_IPC_SPINLOCK_GPIO,
    AMBA_IPC_NUM_SPINLOCK      /* Total number of global spin lock */
} AMBA_IPC_SPINLOCK_IDX_e;
```

Parameters:

Type	Parameter	Description
UINT32	**SpinID**	Spin lock ID to be unlocked

*Table 2-3. Parameters for Spin Lock API **AmbaIPC_SpinUnlock()**.*

Returns:

Return	Description
0	Completed successfully
- 1	Failed to unlock the given ID

*Table 2-4. Returns for Spin Lock API **AmbaIPC_SpinUnlock()**.*

Example:

```
AmbaIPC_SpinUnlock(AMBA_IPC_SPINLOCK_GPIO);
```

See Also:

AmbaIPC_SpinUnlock()
AmbaIPC_SpinLockIrqSave()
AmbaIPC_SpinUnlockIrqRestore()

2.2.3 AmbaIPC_SpinLockIrqSave

API Syntax:

AmbaIPC_SpinLockIrqSave (UINT32 SpinID, UINT32 *pFlags)

Function Description:

- This function is used to assign a global spin lock to a specified ID **@SpinID**. The function is used when an interrupt service routine (ISR) is anticipated to access a spin lock.
- The **AmbaIPC_SpinLockIrqSave** function will disable interrupts and save the IRQ status to a variable **@pFlags**.
- The following spin lock IDs are currently available:

```
typedef enum _AMBA_IPC_SPINLOCK_IDX_e_ {
    AMBA_IPC_SPINLOCK_GPIO,
    AMBA_IPC_NUM_SPINLOCK      /* Total number of global spin lock */
} AMBA_IPC_SPINLOCK_IDX_e;
```

Parameters:

Type	Parameter	Description
UINT32	**SpinID**	Spin lock ID to be locked
UINT32 *	**pFlags**	Saved IRQ status

*Table 2-5. Parameters for Spin Lock API **AmbaIPC_SpinLockIrqSave()**.*

Returns:

Return	Description
0	Completed successfully
- 1	Failed to unlock the given ID

*Table 2-6. Returns for Spin Lock API **AmbaIPC_SpinLockIrqSave()**.*

Example:

```
UINT32 Flags;
AmbaIPC_SpinLockIrqSave(SpinID, &Flags);
```

See Also:

AmbaIPC_SpinUnlock()
AmbaIPC_SpinLockIrqSave()
AmbaIPC_SpinUnlockIrqRestore()

2.2.4 AmbaIPC_SpinUnlockIrqRestore

API Syntax:

AmbaIPC_SpinUnlockIrqRestore (UINT32 SpinID, UINT32 Flags)

Function Description:

- This function is used to unlock the global spin lock assigned to a specified ID **@SpinID**. The function is used when an interrupt service routine (ISR) is anticipated to access a spin lock.
- This function will restore the IRQ status from a variable **@Flags** and enable interrupts.
- The following spin lock IDs are currently available:

```
typedef enum _AMBA_IPC_SPINLOCK_IDX_e_ {
    AMBA_IPC_SPINLOCK_GPIO,
    AMBA_IPC_NUM_SPINLOCK      /* Total number of global spin lock */
} AMBA_IPC_SPINLOCK_IDX_e;
```

Parameters:

Type	Parameter	Description
UINT32	**SpinID**	Spin lock ID to be locked
UINT32	**Flags**	IRQ status to be restored

*Table 2-7. Parameters for Spin Lock API **AmbaIPC_SpinUnlockIrqRestore()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to unlock the given ID

*Table 2-8. Returns for Spin Lock API **AmbaIPC_SpinUnlockIrqRestore()**.*

Example:

```
UINT32 Flags;
AmbaIPC_SpinLockIrqSave(SpinID, &Flags);
AmbaIPC_SpinUnlockIrqRestore(SpinID, Flags);
```

See Also:

AmbaIPC_SpinUnlock()
AmbaIPC_SpinLockIrqSave()
AmbaIPC_SpinUnlockIrqRestore()

3 Mutex (ThreadX)

3.1 AmbaLink Mutex (ThreadX): Overview

This chapter discusses the AmbaLink APIs related to mutex implementation on the ThreadX real-time operating system (RTOS).

Similar in concept to a spin lock, a mutex (MUTual EXclusion) refers to a synchronization object used to ensure mutually exclusive access to a given thread. The fundamental difference between a mutex and a spin lock is that, while the mutex is locked by a thread, the other threads waiting on the lock can accomplish tasks without waiting.

3.2 AmbaLink Mutex (ThreadX): List of APIs

- AmbaIPC_MutexTake
- AmbaIPC_MutexGive

3.2.1 AmbaIPC_MutexTake

API Syntax:

AmbaIPC_MutexTake (UINT32 ID, UINT32 Ticks)

Function Description:

- This function is used to lock the global mutex **@ID** with a timeout value **@Ticks**.
- The unit of the timeout **@Ticks** is ms.
- The following mutex IDs are currently available:

```
typedef enum _AMBA_IPC_MUTEX_IDX_e_ {
AMBA_IPC_MUTEX_I2C_CHANNEL0 = 0,
AMBA_IPC_MUTEX_I2C_CHANNEL1,
AMBA_IPC_MUTEX_I2C_CHANNEL2,
AMBA_IPC_MUTEX_SPI_CHANNEL0,
AMBA_IPC_MUTEX_SPI_CHANNEL1,
AMBA_IPC_MUTEX_SPI_CHANNEL2,
AMBA_IPC_MUTEX_SD0,
AMBA_IPC_MUTEX_SD1,
AMBA_IPC_MUTEX_FIO,
AMBA_IPC_MUTEX_GPIO,
AMBA_IPC_MUTEX_PLL,
AMBA_IPC_NUM_MUTEX          /* Total number of global mutex */
} AMBA_IPC_MUTEX_IDX_e;
```

Parameters:

Type	Parameter	Description
UINT32	**ID**	Mutex ID to be locked
UINT32	**Ticks**	Timeout value (ms)

*Table 3-1. Parameters for Mutex API **AmbaIPC_MutexTake()**.*

Returns:

Return	Description
0x0	Completed successfully
0x01	Mutex was deleted while thread was suspended.
0x1D	Service was unable to acquire ownership of the mutex within the specified wait period.
0x1A	Suspension was aborted by another thread, timer, or interrupt service routine (ISR).
0x1C	Invalid mutex pointer
0x13	Invalid caller of this service

*Table 3-2. Returns for Mutex API **AmbaIPC_MutexTake()**.*

Example:

```
AmbaIPC_MutexTake(AMBA_IPC_MUTEX_SD1, AMBA_KAL_WAIT_FOREVER);
```

See Also:

AmbaIPC_Mutex()

3.2.2 AmbaIPC_MutexGive

API Syntax:

AmbaIPC_MutexGive (UINT32 ID)

Function Description:

- This function is used to unlock the global mutex **@ID**.
- The following mutex IDs are currently available:

```
typedef enum _AMBA_IPC_MUTEX_IDX_e_ {
AMBA_IPC_MUTEX_I2C_CHANNEL0 = 0,
AMBA_IPC_MUTEX_I2C_CHANNEL1,
AMBA_IPC_MUTEX_I2C_CHANNEL2,
AMBA_IPC_MUTEX_SPI_CHANNEL0,
AMBA_IPC_MUTEX_SPI_CHANNEL1,
AMBA_IPC_MUTEX_SPI_CHANNEL2,
AMBA_IPC_MUTEX_SD0,
AMBA_IPC_MUTEX_SD1,
AMBA_IPC_MUTEX_FIO,
AMBA_IPC_MUTEX_GPIO,
AMBA_IPC_MUTEX_PLL,
AMBA_IPC_NUM_MUTEX          /* Total number of global mutex */
} AMBA_IPC_MUTEX_IDX_e;
```

Parameters:

Type	Parameter	Description
UINT32	**ID**	Mutex ID to be locked

*Table 3-3. Parameters for Mutex API **AmbaIPC_MutexGive()**.*

Returns:

Return	Description
0x0	Completed successfully
0x01	Mutex was deleted while thread was suspended.
0x1D	Service was unable to acquire ownership of the mutex within the specified wait period.
0x1A	Suspension was aborted by another thread, timer, or interrupt service routine (ISR).
0x1C	Invalid mutex pointer
0x13	Invalid caller of this service

*Table 3-4. Returns for Mutex API **AmbaIPC_MutexGive()**.*

Example:

```
AmbaIPC_MutexGive(AMBA_IPC_MUTEX_SD1);
```

See Also:

AmbaIPC_MutexTake()

4 Remote Processor Messaging (ThreadX)

4.1 AmbaLink RPMsg (ThreadX): Overview

This chapter discusses the AmbaLink APIs related to Remote Processor Messaging (RPMsg) on the ThreadX real-time operating system (RTOS).

The overall structure of the AmbaLink framework is based on RPMsg operations. RPMsg is a VirtIO-based messaging bus that allows kernel drivers to communicate with remote system processors via inter-process communication (IPC). In turn, these kernel drivers expose the appropriate user-space interfaces as required. In the context of AmbaLink, RPMsg operations are used to send messages to, and receive messages from, processes running on a separate core.

Each RPMsg device serves as a communication channel to a remote processor; these devices are typically referred to as channels for this reason. RPMsg channels are identified by a textual name and have both a local (i.e., source) and a remote (i.e., destination) RPMsg address.

In ThreadX side, it needs to implement the RPMsg device and notify Linux of the registration of the RPMsg device. After the registration is done, the message can be transferred via the unique RPMsg channel. When inbound messages arrive, the RPMsg core dispatches them to the appropriate RPMsg device according to their destination address by invoking the device's receive handler with the payload of the inbound message.

Refer to `linux/Documentation/rpmsg.txt` for additional details.

4.2 AmbaLink RPMsg (ThreadX): List of APIs

- AmbaIPC_Alloc
- AmbaIPC_RegisterChannel
- AmbaIPC_TrySend
- AmbaIPC_Send
- AmbaIPC_UnregisterChannel

4.2.1 AmbaIPC_Alloc

API Syntax:

AmbaIPC_Alloc (const char *pName, AMBA_IPC_MSG_HANDLER_f MsgHandler)

Function Description:

- **AmbaIPC_Alloc** is used to allocate and initialize a RPMsg channel with an assigned name **@PName**. This function also registers a message handler **@MsgHandler**.
- The **AmbaIPC_Alloc** function returns a non-zero if an RPMsg channel is successfully allocated and initialized.

Parameters:

Type	Parameter	Description
const char *	**pName**	Channel name
AMBA_ IPC_MSG_ HANDLER_f	**MsgHandler**	Channel message handler. Please refer to Section 4.2.1.1 below for more details.

*Table 4-1. Parameters for RPMsg API **AmbaIPC_Alloc()**.*

Returns:

Return	Description
Nonzero	Newly created handle (RPMsg channel)
NULL	Allocation failed

*Table 4-2. Returns for RPMsg API **AmbaIPC_Alloc()**.*

Example:

Please refer to `rtos/ssp/unittest/AmbaiPC_Test.c`.

```
static AMBA_IPC_HANDLE Channel;
Channel = AmbaIPC_Alloc("echo_cortex", MsgHandler);
```

See Also:

None

4.2.1.1 AmbaIPC_Alloc > AMBA_IPC_MSG_HANDLER_f

Type	Description
AMBA_IPC_MSG_HANDLER_f	typedef int (*AMBA_IPC_MSG_HANDLER_f)(AMBA_IPC_HANDLE IpcHandle, AMBA_IPC_MSG_CTRL_s *pMsgCtrl). Please refer to Section 4.2.1.2 and Section 4.2.1.3 for details.

*Table 4-3. Definition of **AMBA_IPC_MSG_HANDLER_f** for RPMsg API **AmbaIPC_Alloc()**.*

4.2.1.2 AmbaIPC_Alloc > AMBA_IPC_HANDLE

Type	Description
AMBA_IPC_HANDLE	typedef void *AMBA_IPC_HANDLE

*Table 4-4. Definition of **AMBA_IPC_HANDLE** for RPMsg API **AmbaIPC_Alloc()**.*

4.2.1.3 AmbaIPC_Alloc > AMBA_IPC_MSG_CTRL_s

Type	Field	Description
UINT32	**Length**	Message length
void	***pMsgData**	Point to message

*Table 4-5. Definition of **AMBA_IPC_MSG_CTRL_s** for RPMsg API **AmbaIPC_Alloc()**.*

4.2.2 AmbaIPC_RegisterChannel

API Syntax:

AmbaIPC_RegisterChannel (AMBA_IPC_HANDLE Channel, const char *pRemote)

Function Description:

- This function registers a RPMsg channel **@Channel** with the remote host **@pRemote**.
- The remote host is fixed in Linux. **pRemote** uses “NULL” by default.

Parameters:

Type	Parameter	Description
AMBA_IPC_ HANDLE	**Channel**	RPMsg channel
const char *	**pRemote**	Remote host (Using “NULL” by default)

*Table 4-6. Parameters for RPMsg API **AmbaIPC_RegisterChannel()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to register a channel

*Table 4-7. Returns for RPMsg API **AmbaIPC_RegisterChannel()**.*

Example:

Refer to `rtos/ssp/unittest/AmbaIPC_Test.c`.

```
int Rval = 0;
static AMBA_IPC_HANDLE Channel;
Channel = AmbaIPC_Alloc(“echo_cortex”, MsgHandler);
Rval = AmbaIPC_RegisterChannel(Channel, NULL);
```

See Also:

AmbaIPC_Alloc()

4.2.3 AmbaIPC_TrySend

API Syntax:

AmbaIPC_TrySend (AMBA_IPC_HANDLE Channel, void *pData, int Length)

Function Description:

- This function is used to send a message **@pData** across a channel **@Channel**.
- If no buffers are available, this function will immediately return as failed (i.e., without waiting until a buffer becomes available).
- The **AmbaIPC_TrySend** function does not suspend operations to wait for a response after sending a message to Linux.

Parameters:

Type	Parameter	Description
AMBA_IPC_ HANDLE	**Channel**	RPMsg channel
void *	**pData**	Point to message buffer
int	**Length**	Message length

*Table 4-8. Parameters for RPMsg API **AmbaIPC_TrySend()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to send the message

*Table 4-9. Returns for RPMsg API **AmbaIPC_TrySend()**.*

Example:

Refer to `rtos/ssp/unittest/AmbaIPC_Test.c`.

```
int Rval = 0;
char *pMsg;
static AMBA_IPC_HANDLE Channel;
Channel = AmbaIPC_Alloc("echo_cortex", MsgHandler);
Rval = AmbaIPC_RegisterChannel(Channel, NULL);
pMsg = "Hello Linux! ";
AmbaIPC_TrySend(Channel, pMsg, strlen(pMsg) + 1);
```

See Also:

None

4.2.4 AmbaIPC_Send

API Syntax:

AmbaIPC_Send (AMBA_IPC_HANDLE Channel, void *pData, int Length)

Function Description:

- This function is used to send a message **@pData** across a channel **@Channel**.
- If no Transmit buffers are available, this function will be blocked until a buffer becomes available.
- The **AmbaIPC_Send** function does not suspend operations to wait for a response after sending a message to Linux.

Parameters:

Type	Parameter	Description
AMBA_IPC_ HANDLE	**Channel**	RPMsg channel
void *	**pData**	Message buffer
int	**Length**	Message length

*Table 4-10. Parameters for RPMsg API **AmbaIPC_Send()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to send the message

*Table 4-11. Returns for RPMsg API **AmbaIPC_Send()**.*

Example:

Refer to `rtos/ssp/unittest/AmbaIPC_Test.c`.

```
int Rval = 0;
char *pMsg;
static AMBA_IPC_HANDLE Channel;
Channel = AmbaIPC_Alloc("echo_cortex", MsgHandler);
Rval = AmbaIPC_RegisterChannel(Channel, NULL);
pMsg = "Hello Linux! ";
AmbaIPC_Send(Channel, pMsg, strlen(pMsg) + 1);
```

See Also:

None

4.2.5 AmbaIPC_UnregisterChannel

API Syntax:

AmbaIPC_UnregisterChannel (AMBA_IPC_HANDLE Channel)

Function Description:

- This function is used to de-register a channel **@Channel**.

Parameters:

Type	Parameter	Description
AMBA_IPC_ HANDLE	**Channel**	RPMsg channel

*Table 4-12. Parameters for RPMsg API **AmbaIPC_UnregisterChannel()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to de-register a channel

*Table 4-13. Returns for RPMsg API **AmbaIPC_UnregisterChannel()**.*

Example:

Refer to `rtos/ssp/unittest/AmbaIPC_Test.c`.

```
int Rval = 0;
static AMBA_IPC_HANDLE Channel;
Channel = AmbaIPC_Alloc("echo_cortex", MsgHandler);
Rval = AmbaIPC_RegisterChannel(Channel, NULL);
if (Rval == 0)
Rval = AmbaIPC_UnregisterChannel(Channel);
```

See Also:

AmbaIPC_RegisterChannel

5 Remote Procedure Calls (ThreadX)

5.1 AmbaLink RPC (ThreadX): Overview

This chapter discusses the AmbaLink APIs related to Remote Procedure Calls (RPCs) on the ThreadX real-time operating system (RTOS).

An RPC is a type of inter-process communication in which a program running on one processor executes a subroutine on another processor across a shared network. The AmbaLink RPC framework allows a software programmer—without knowing the details of the cross-processor interaction—to write code as though the subroutine will be executed locally.

5.2 AmbaLink RPC (ThreadX): List of APIs

- AmbaIPC_SvcRegister
- AmbaIPC_SvcUnregister
- AmbaIPC_ClientCreate
- AmbaIPC_ClientDestroy
- AmbaIPC_ClientCall
- AmbaIPC_RpcStrError

5.2.1 AmbaIPC_SvcRegister

API Syntax:

AmbaIPC_SvcRegister (INT32 Prog, INT32 Vers, char * const Name, UINT32 Priority, void *pStack, UINT32 StackSize, AMBA_IPC_PROG_INFO_s *info, INT32 new_thread)

Function Description:

- This function is used to register a service with a binder to Linux.
- The **AmbaIPC_SvcRegister** function also creates a local thread to handle incoming calls from the Linux client.

Parameters:

Type	Parameter	Description
INT32	**Prog**	Program ID
INT32	**Vers**	Program version number
char * const	**Name**	Thread name
UINT32	**Priority**	Thread priority
void *	**pStack**	Thread stack
UINT32	**StackSize**	Thread stack size
AMBA_IPC_ PROG_INFO_s *	**info**	IPC program information including communication modes, procedure functions, etc. Please refer to Section 5.2.1.1 below for definition.
INT32	**new_thread**	Specify if a new thread is to run this svc

*Table 5-1. Parameters for RPC API **AmbaIPC_SvcRegister()**.*

Returns:

Return	Description
0	Completed successfully
Nonzero	Failed to register

*Table 5-2. Returns for RPC API **AmbaIPC_SvcRegister()**.*

Example:

Program in Server (ThreadX). For more details, please refer to `rtos/ssp/unittest/AmbaIPC_Test.c.`

```
char *stack = AmbaLink_Malloc(0x1000);
prog_info->ProcNum = 2;
prog_info->pProcInfo = AmbaLink_Malloc(prog_info->ProcNum*sizeof(AMBA_IPC_PROC_s));
 prog_info->pProcInfo[0].Mode = AMBA_IPC_ASYNCHRONOUS;
 prog_info->pProcInfo[1].Mode = AMBA_IPC_SYNCHRONOUS;
 prog_info->pProcInfo[0].Proc = (AMBA_IPC_PROC_f) &AmbaRpcProg_R_Test_Print_Svc;
 prog_info->pProcInfo[1].Proc = (AMBA_IPC_PROC_f) &AmbaRpcProg_R_Test_Sum_Svc;
 AmbaIPC_SvcRegister(AMBA_RPC_PROG_R_TEST_PROG_ID, AMBA_RPC_PROG_R_TEST_VER, "test_rpc_svc",65, stack, 0x1000, prog_info, 1);
```

Program in client (Linux). For more details, please refer to `ambalink_sdk/pkg/ambaipc_test/clnt_test.c.`

```
int clnt;
clnt = ambaipc_clnt_create(host, AMBA_RPC_PROG_R_TEST_PROG_ID, AMBA_RPC_PROG_R_TEST_VER);
AmbaRpcProg_R_Test_Print_Clnt(msg, NULL, clnt);
```

See Also:

AmbaIPC_SvcUnregister()

5.2.1.1 AmbaIPC_SvcRegister > Amba_IPC_PROG_INFO_s

Type	Field	Description
INT32	**ProcNum**	Total number of procedures in the RPC program
AMBA_IPC_PROC_s *	**pProcInfo**	The information of the procedure in the RPC program

*Table 5-3. Definition of **Amba_IPC_PROG_INFO_s** for RPC API **AmbaIPC_SvcRegister()**.*

5.2.1.2 AmbaIPC_SvcRegister > Amba_IPC_PROG_INFO_s > AMBA_IPC_PROC_s

Type	Field	Description
AMBA_IPC_PROC_f	**Proc**	The callback function for the procedure
AMBA_IPC_COMMUNICATION_MODE_e	**Mode**	The communication mode for the procedure

*Table 5-4. Definition of **AMBA_IPC_PROC_s** for RPC API **AmbaIPC_SvcRegister()**.*

5.2.1.3 AmbaIPC_SvcRegister > Amba_IPC_PROG_INFO_s > AMBA_IPC_PROC_s > AMBA_ IPC_PROC_f

Field	Description
AMBA_IPC_PROC_f	typedef void (*AMBA_IPC_PROC_f)(void *, AMBA_IPC_SVC_ RESULT_s *) The function pointer prototype for RPC procedure. All the RPC procedures need to follow this type.

*Table 5-5. Definition of **AMBA_IPC_PROC_f** for RPC API **AmbaIPC_SvcRegister()**.*

5.2.1.4 AmbaIPC_SvcRegister > Amba_IPC_PROG_INFO_s > AMBA_IPC_PROC_s > AMBA_ IPC_COMMUNICATION_MODE_e

The supported communication modes are defined as follows:

```
typedef enum _AMBA_IPC_COMMUICATION_MODE_e_ {
      AMBA_IPC_SYNCHRONOUS = 0,
      AMBA_IPC_ASYNCHRONOUS,
      AMBA_IPC_MODE_MAX = 0xFFFFFFFF
} AMBA_IPC_COMMUICATION_MODE_e ;
```


5.2.1.5 AmbaIPC_SvcRegister > Amba_IPC_PROG_INFO_s > AMBA_IPC_PROC_s > AMBA_IPC_PROC_f > AMBA_IPC_RESULT_s

Type	Field	Description
INT32	**Length**	The size of the calculated result for the procedure
void*	**pResult**	The pointer to the calculated result
AMBA_IPC_ COMMUNICA-TION_MODE_e	**Mode**	The communication mode of the procedure. Please refer to Section 5.2.1.4.
AMBA_IPC_RE-PLY_STATUS_e	**Status**	The status of the procedure. Please refer to Section 5.2.1.6.

*Table 5-6. Definition of **AMBA_IPC_RESULT_s** for RPC API **AmbaIPC_SvcRegister()**.*

5.2.1.6 AmbaIPC_SvcRegister > Amba_IPC_PROG_INFO_s > AMBA_IPC_PROC_s > AMBA_ IPC_PROC_f > AMBA_IPC_SVC_RESULT_s > AMBA_IPC_REPLY_STATUS_s

The supported reply status of the procedure is defined as follows:

```
typedef enum _AMBA_IPC_REPLY_STATUS_e_ {
      AMBA_IPC_REPLY_SUCCESS = 0,
      AMBA_IPC_REPLY_PROG_UNAVAIL,
      AMBA_IPC_REPLY_PARA_INVALID,
      AMBA_IPC_REPLY_SYSTEM_ERROR,
      AMBA_IPC_REPLY_TIMEOUT,
      AMBA_IPC_REPLY_MAX = 0xFFFFFFFF
} AMBA_IPC_REPLY_STATUS_e;
```

5.2.2 AmbaIPC_SvcUnregister

API Syntax:

AmbaIPC_SvcUnregister (INT32 Program, INT32 Version)

Function Description:

- This function is used to de-register a service with a binder to Linux.
- The **AmbaIPC_SvcUnregister** function will also terminate the corresponding local service thread.

Parameters:

Type	Parameter	Description
INT32	**Program**	Program ID
INT32	**Version**	Program version number

*Table 5-7. Parameters for RPC API **AmbaIPC_SvcUnregister()**.*

Returns:

Return	Description
0	Completed successfully
Nonzero	Failed to de-register

*Table 5-8. Returns forRPC API **AmbaIPC_SvcUnregister()**.*

Example:

For more details, please refer to `rtos/ssp/unittest/AmbaIPC_Test.c`.

```
AmbaIPC_SvcUnregister(AMBA_RPC_PROG_R_TEST_PROG_ID,
AMBA_RPC_PROG_R_TEST_VER);
```

See Also:

AmbIPC_SvcRegister()

5.2.3 AmbaIPC_ClientCreate

API Syntax:

AmbaIPC_ClientCreate (INT32 Host, INT32 Program, INT32 Version)

Function Description:

- This function is used to create an inter-process communication (IPC) client.
- The client should be registered to the host.
- Supported hosts include:
  - **AMBA_IPC_HOST_LINUX**
  - **AMBA_IPC_HOST_THREADX**

Parameters:

Type	Parameter	Description
INT32	**Host**	Specifies whether the server is in Linux or RTOS
INT32	**Program**	RPC program ID
INT32	**Version**	RPC program version

*Table 5-9. Parameters for RPC API **AmbaIPC_ClientCreate()**.*

Returns:

Return	Description
Nonzero	Client created successfully
0	Creation failed

*Table 5-10. Returns for RPC API **AmbaIPC_ClientCreate()**.*

Example:

For more details, please refer to `rtos/ssp/unittest/AmbaIPC_Test.c`.

```
int Clnt = 0;

Clnt = AmbaIPC_ClientCreate(AMBA_IPC_HOST_LINUX,
AMBA_RPC_PROG_LU_TEST_PROG_ID, AMBA_RPC_PROG_LU_TEST_VER);
if (Clnt == 0) {
      AmbaShell_Print(env, "Cient creation failed\n");
      return 0;
}
```

See Also:

AmbaIPC_ClientDestroy()

5.2.4 AmbaIPC_ClientDestroy

API Syntax:

AmbaIPC_ClientDestroy (INT32 Clnt)

Function Description:

- This function is used to eliminate the IPC client using the **Clnt** parameter.

Parameters:

Type	Parameter	Description
INT32	**Clnt**	Client to be destroyed

*Table 5-11. Parameters for RPC API **AmbaIPC_ClientDestroy()**.*

Returns:

Return	Description
0	Completed successfully

*Table 5-12. Returns for RPC API **AmbaIPC_ClientDestroy()**.*

Example:

For more details, please refer to `rtos/ssp/unittest/AmbaIPC_Test.c`.

```
int clnt;

clnt = AmbaIPC_ClientCreate(AMBA_IPC_HOST_LINUX, AMBA_RPC_PROG_LU_TEST_PROG_
ID,
AMBA_RPC_PROG_LU_TEST_VER);
if (clnt == 0) {
      AmbaShell_Print(env, "Cient creation failed\n");
      return 0;
}
AmbaIPC_ClientDestroy(clnt);
```

See Also:

AmbaIPC_ClientCreate()

5.2.5 AmbaIPC_ClientCall

API Syntax:

AmbaIPC_ClientCall (INT32 Client, INT32 Proc, void *pIn, INT32 InLen, void *pOut, INT32 OutLen, INT32 Timeout)

Function Description:

- This function is used to execute the given function ID **@Proc** in the program **@Client** in the server.

Parameters:

Type	Parameter	Description
INT32	**Client**	Client instance
INT32	**Proc**	Function ID to be executed
void *	**pIn**	Pointer to input parameter
INT32	**InLen**	Length of input parameter
void *	**pOut**	Pointer to output parameter
INT32	**OutLen**	Length of output parameter
INT32	**Timeout**	Timeout value

*Table 5-13. Parameters for RPC API **AmbaIPC_ClientCall()**.*

Returns:

Return	Description
0	Completed successfully
1	Program is unavailable
2	Parameter is invalid
3	System error
4	Timeout

*Table 5-14. Returns for RPC API **AmbaIPC_ClientCall()**.*

Example:

Program in server (Linux). For more details, please refer to `ambalink_sdk/pkg/ambaipc_test/svc_test.c`

```
AMBA_IPC_PROG_INFO_s prog_info[1];
prog_info->ProcNum = 2;
prog_info->pProcInfo = malloc(prog_info->ProcNum*sizeof(AMBA_IPC_
PROC_s));
prog_info->pProcInfo[0].Mode = AMBA_IPC_ASYNCHRONOUS;
prog_info->pProcInfo[0].Proc =(AMBA_IPC_PROC_f)&
AmbaRpcProg_LU_Test_Print_Svc;
prog_info->pProcInfo[1].Mode = AMBA_IPC_SYNCHRONOUS;
prog_info->pProcInfo[1].Proc = (AMBA_IPC_PROC_f) &
AmbaRpcProg_LU_Test_Sum_Svc;
```

Program in client (ThreadX). For more details, please refer to
rtos/ssp/unittest/AmbaIPC_Test.c.

```
AMBA_IPC_REPLY_STATUS_e AmbaRpcProg_LU_Test_Print_Clnt(const char *pStr, int
*pResult, int Clnt)
{
      AMBA_IPC_REPLY_STATUS_e status;
      status = AmbaIPC_ClientCall(Clnt, AMBA_RPC_PROG_LU_TEST_PRINT, (void *)
pStr, strlen(pStr)+1, NULL, 0, 0);
      return status;
}
```

See Also:

None

5.2.6 AmbaIPC_RpcStrError

API Syntax:

AmbaIPC_RpcStrError (INT32 ErrNum)

Function Description:

- This function is used to get the description of the error code.

Parameters:

Type	Parameter	Description
INT32	**ErrNum**	The error code from the RPC related function

*Table 5-15. Parameters for RPC API **AmbaIPC_RpcStrError()**.*

Returns:

Return	Description
Const char *	The error description

*Table 5-16. Returns for RPC API **AmbaIPC_RpcStrError()**.*

Example:

```
AMBA IPC REPLY STATUS e status;
status= AmbaiPC ClientCall(Clnt, AMBA RPC PROG LU TEST PRINT, (void*)

pStr, strlen(pStr)+1, NULL, 0, 0);

      if( status != 0) {

            AmbaPrint("Client call failed cause %s", AmbaIPC_
RpcStrError(status));
}
```

See Also:

None

6 RTOS Virtual File System (ThreadX)

6.1 AmbaLink RFS (RTOS Virtual File System): Overview

This chapter discusses the AmbaLink APIs that are related to the RTOS Virtual File System (RFS) on the ThreadX real-time operating system (RTOS).

The RFS is a Linux kernel subsystem that provides file-related functionality to user-space programs, allowing ThreadX to access filesystems located in the Linux kernel space. The RFS enables calls such as **open()**, **read()**, and **write()** to be issued successfully regardless of the specific filesystem or the underlying physical medium, and this, eliminates the need to rewrite or recompile applications in the event that new filesystems or storage media types are introduced into Linux.

6.2 AmbaLink RFS (RTOS Virtual File System): List of APIs

- AmbaIPC_fopen
- AmbaIPC_fclose
- AmbaIPC_fread
- AmbaIPC_fwrite
- AmbaIPC_fseek
- AmbaIPC_ftell
- AmbaIPC_mkdir
- AmbaIPC_rmdir
- AmbaIPC_remove
- AmbaIPC_move
- AmbaIPC_chmod
- AmbaIPC_chdmod
- AmbaIPC_mount
- AmbaIPC_unmount
- AmbaIPC_sync
- AmbaIPC_fsync
- AmbaIPC_stat
- AmbaIPC_getdev
- AmbaIPC_feof
- AmbaIPC_opendir
- AmbaIPC_readdir
- AmbaIPC_closedir
- AmbaIPC_chdir

6.2.1 AmbaIPC_fopen

API Syntax:

AmbaIPC_fopen (const char *name, char *mode_str)

Function Description:

- This function is used to open a remote target file.

Parameters:

Type	Parameter	Description
const char*	**name**	The name of the target file
char*	**mode_str**	The file accessing mode **r** (read) or **w** (write)

*Table 6-1. Parameters for RFS API **AmbaIPC_fopen()**.*

Returns:

Return	Description
> 0	File pointer of the opened file
< = 0	Failed to open the target file

*Table 6-2. Returns for RFS API **AmbaIPC_fopen()**.*

Example:

For more details, please refer to `rtos/ssp/unittest/AmbaIPC_Test.c`

```
{
        void* fp;
        char* filename = "/tmp/test.c";

        fp = AmbaIPC_fopen(filename, "w");
          if (fp == 0 ) {
              AmbaShell_Print(env, "fopen failed (%d)\n", fp);
          }

}
```

See Also:

AmbaIPC_fclose ()

6.2.2 AmbaIPC_fclose

API Syntax:

AmbaIPC_fclose (void *fp)

Function Description:

- This function is used to close a remote file.

Parameters:

Type	Parameter	Description
void*	**fp**	File pointer returned by **AmbaIPC_fopen**

*Table 6-3. Parameters for RFS API **AmbaIPC_fclose()**.*

Returns:

Return	Description
0	The remote file was closed successfully.
- 1	Failed to close the remote file

*Table 6-4. Returns for RFS API **AmbaIPC_fclose()**.*

Example:

For more details, please refer to `rtos/ssp/unittest/AmbaIPC_Test.c`

```
int rval;
rval = AmbaIPC_fclose(fp);
if (rval < 0) {
          AmbaShell_Print(env, "fclose failed (%d)\n", rval);
}
```

See Also:

AmbaIPC_fopen()

6.2.3 AmbaIPC_fread

API Syntax:

AmbaIPC_fread (void *data, int size, void *fp)

Function Description:

- This function is used to read the remote target file.

Parameters:

Type	Parameter	Description
void*	**data**	Pointer to a block of memory in which the read bytes are stored
int	**size**	The size of read data
void*	**fp**	The file pointer

*Table 6-5. Parameters for RFS API **AmbaIPC_fread()**.*

Returns:

Return	Description
int	Total number of bytes successfully read

*Table 6-6. Returns for RFS API **AmbaIPC_fread()**.*

Example:

For more details, please refer to `rtos/ssp/unittest/AmbaIPC_Test.c`

```
int flen;
char pRdBuf[128];
flen = AmbaIPC_fread(pRdBuf, sizeof(pRdBuf), fp);
```

See Also:

AmbaIPC_fopen()
AmbaIPC_fclose()
AmbaIPC_fwrite()

6.2.4 AmbaIPC_fwrite

API Syntax:

AmbaIPC_fwrite (void *data, int size, void *fp)

Function Description:

- This function is used to write to a remote target file.

Parameters:

Type	Parameter	Description
void*	**data**	Pointer to a block of memory in which the written bytes are stored
int	**size**	The size of data written
void*	**fp**	The file pointer

*Table 6-7. Parameters for RFS API **AmbaIPC_fwrite()**.*

Returns:

Return	Description
int	Total number of bytes successfully written

*Table 6-8. Returns for RFS API **AmbaIPC_fwrite()**.*

Example:

For more details, please refer to `rtos/ssp/unittest/AmbaIPC_Test.c`

```
int ral;
char *msg = "hello world";
rval = AmbaIPC_fwrite(msg, strlen(msg), fp);
  if (rval < 0) {
 AmbaShell_Print(env, "fwrite failed (%d)\n", rval);
}
```

See Also:

AmbaIPC_fopen()
AmbaIPC_fclose()
AmbaIPC_fread()

6.2.5 AmbaIPC_fseek

API Syntax:

AmbaIPC_fseek (void *fp, INT64 offset, int origin)

Function Description:

- This function is used to set the position indicator associated with a file pointer to a new position.
- Origin can be specified as follows:
  - **AMBA_IPC_RFS_SEEK_SET**: Start of the file
  - **AMBA_IPC_RFS_SEEK_CUR**: Current position of the file pointer
  - **AMBA_IPC_RFS_SEEK_END**: End of the file

Parameters:

Type	Parameter	Description
void*	**fp**	The file pointer
INT64	**offset**	The number of bytes to offset from the origin
int	**origin**	Position used as reference for the offset

*Table 6-9. Parameters for RFS API **AmbaIPC_fseek()**.*

Returns:

Return	Description
0	The position indicator has been set successfully.
- 1	Failed to set the position indicator

*Table 6-10. Returns for RFS API **AmbaIPC_fseek()**.*

Example:

For more details, please refer to `rtos/ssp/unittest/AmbaIPC_Test.c`

```
int rval,
rval = AmbaIPC_fseek(fp, 0, AMBA_IPC_RFS_SEEK_SET);
if (rval < 0) {
                AmbaShell_Print(env, "fseek failed (%d)\n",
                        rval);
}
```

See Also:

AmbaIPC_ftell()

6.2.6 AmbaIPC_ftell

API Syntax:

AmbaIPC_ftell (void *fp)

Function Description:

- This function is used to retrieve the current position indicator for a given file pointer.

Parameters:

Type	Parameter	Description
void*	**fp**	The file pointer

*Table 6-11. Parameters for RFS API **AmbaIPC_ftell()**.*

Returns:

Return	Description
>= 0	The current position is returned.
- 1	Failed to retrieve the current position

*Table 6-12. Returns for RFS API **AmbaIPC_ftell()**.*

Example:

None

See Also:

AmbaIPC_fseek()

6.2.7 AmbaIPC_mkdir

API Syntax:

AmbaIPC_mkdir (const char *name)

Function Description:

- This function is used to create a remote directory.

Parameters:

Type	Parameter	Description
const char *	**name**	The name of the target directory

*Table 6-13. Parameters for RFS API **AmbaIPC_mkdir()**.*

Returns:

Return	Description
0	The target directory was created successfully.
- 1	Failed to create the target directory

*Table 6-14. Returns for RFS API **AmbaIPC_mkdir()**.*

Example:

```
char *path="/tmp/test.c"
AmbaIPC_mkdir(path);
```

See Also:

AmbaIPC_rmdir()

6.2.8 AmbaIPC_rmdir

API Syntax:

AmbaIPC_rmdir (const char *name)

Function Description:

- This function is used to remove a remote directory.

Parameters:

Type	Parameter	Description
const char *	**name**	The name of the target directory

*Table 6-15. Parameters for RFS API **AmbaIPC_rmdir()**.*

Returns:

Return	Description
0	The target directory was successfully removed.
- 1	Failed to remove the target directory

*Table 6-16. Returns for RFS API **AmbaIPC_rmdir()**.*

Example:

```
char *file = "/tmp/test.c";
AmbaIPC_rmdir(file);
```

See Also:

AmbaIPC_mkdir()

6.2.9 AmbaIPC_remove

API Syntax:

AmbaIPC_remove (const char *name)

Function Description:

- This function is used to remove a remote file.

Parameters:

Type	Parameter	Description
const char *	**name**	The name of the target file

*Table 6-17. Parameters for RFS API **AmbaIPC_remove()**.*

Returns:

Return	Description
0	The target file was removed successfully.
- 1	Failed to remove the target file

*Table 6-18. Returns for RFS API **AmbaIPC_remove()**.*

Example:

```
char *path="/tmp/test.c"
AmbaIPC_remove(path);
```

See Also:

None

6.2.10 AmbaIPC_move

API Syntax:

AmbaIPC_move (const char *old_name, const char *new_name)

Function Description:

- This function is used to rename a remote file.

Parameters:

Type	Parameter	Description
const char *	**old_name**	The original name of the target file
const char *	**new_name**	The new name of the target file

*Table 6-19. Parameters for RFS API **AmbaIPC_move()**.*

Returns:

Return	Description
0	The target file was renamed successfully.
- 1	Failed to rename the target file

*Table 6-20. Returns for RFS API **AmbaIPC_move()**.*

Example:

```
char *old_name="/tmp/old_test";
char *new_name="/tmp/new_test";

AmbaIPC_move(old_name, new_name);
```

See Also:

None

6.2.11 AmbaIPC_chmod

API Syntax:

AmbaIPC_chmod (const char *name, int mode)

Function Description:

- This function is used to change the access permission of a remote file.

Parameters:

Type	Parameter	Description
const char *	**name**	The name of the target file
int	**mode**	The access permission in octal digits. Please refer to the numeric mode in chmod of Linux.

*Table 6-21. Parameters for RFS API **AmbaIPC_chmod()**.*

Returns:

Return	Description
0	The access permission of the target file was changed successfully.
< 0	Failed to change the permission of the target file

*Table 6-22. Returns for RFS API **AmbaIPC_chmod()**.*

Example:

```
int mode = 0777;
char *name = "/tmp/test.c"

AmbaIPC_chmod(name, mode);
```

See Also:

AmbaIPC_chdmod()

6.2.12 AmbaIPC_chdmod

API Syntax:

AmbaIPC_chdmod (const char * dir_name, int mode)

Function Description:

- This function is used to change the access permission of a remote directory.

Parameters:

Type	Parameter	Description
const char *	**dir_name**	The name of the remote directory
int	**mode**	The access permission in octal digits. Please refer to the numeric mode in chmod in Linux.

*Table 6-23. Parameters for RFS API **AmbaIPC_chdmod()**.*

Returns:

Return	Description
0	The access permission of the target directory was changed successfully.
< 0	Failed to change the access permission

*Table 6-24. Returns for RFS API **AmbaIPC_chdmod()**.*

Example:

```
char *dir = /tmp/test;
int mode = 0777;

AmbaIPC_chdmod(dir, mode);
```

See Also:

AmbaIPC_chmod()

6.2.13 AmbaIPC_mount

API Syntax:

AmbaIPC_mount (const char * dev_name, const char *dir_name, const char* type)

Function Description:

- This function is used to attach the filesystem found on the device to the target directory.

Parameters:

Type	Parameter	Description
const char *	**dev_name**	The device which is required to be attached.
const char *	**dir_name**	The target directory to mount
const char *	**type**	The type of the file system

*Table 6-25. Parameters for RFS API **AmbaIPC_mount()**.*

Returns:

Return	Description
0	Mount the device to the target directory successfully
< 0	Failed to mount the device

*Table 6-26. Returns for RFS API **AmbaIPC_mount()**.*

Example:

```
char *dev_name = "/dev/sda1";
char *dir_name = "/boot";
char *type = "ext4";

AmbaIPC_mount(dev_name, dir_name, type);
```

See Also:

AmbaIPC_umount()

6.2.14 AmbaIPC_umount

API Syntax:

AmbaIPC_umount (const char * dir_name)

Function Description:

- This function is used to unmount the file system.

Parameters:

Type	Parameter	Description
const char *	**dir_name**	The name of the remote directory where the file system has been mounted.

*Table 6-27. Parameters for RFS API **AmbaIPC_umount()**.*

Returns:

Return	Description
0	The file system is unmounted successfully.
< 0	Failed to unmount the file system

*Table 6-28. Returns for RFS API **AmbaIPC_umount()**.*

Example:

None

See Also:

AmbaIPC_mount()

6.2.15 AmbaIPC_sync

API Syntax:

AmbaIPC_sync (void)

Function Description:

- This function is used to flush the file system buffers.

Parameters:

None

Returns:

Return	Description
0	Successfully flush the file system buffers
< 0	Failed to flush the file system buffers

*Table 6-29. Returns for RFS API **AmbaIPC_sync()**.*

Example:

None

See Also:

AmbaIPC_fsync()

6.2.16 AmbaIPC_fsync

API Syntax:

AmbaIPC_fsync (void *fp)

Function Description:

- This function is used to synchronize all the modified buffer cache pages for the file.

Parameters:

Type	Parameter	Description
void *	**fp**	The file pointer to the file desired to be synchronized

*Table 6-30. Parameters for RFS API **AmbaIPC_fsync()**.*

Returns:

Return	Description
0	Successfully synchronize the status of the file
< 0	Failed to synchronize the file

*Table 6-31. Returns for RFS API **AmbaIPC_fsync()**.*

Example:

None

See Also:

AmbaIPC_sync()

6.2.17 AmbaIPC_stat

API Syntax:

AmbaIPC_stat (const char * name, AMBA_IPC_RFS_STAT_s *stat)

Function Description:

- This function is used to get information regarding a file or directory.
- The data structure storing the file information is defined as follows:

```
typedef struct {
    UINT64  ino;
    UINT32  dev;
    UINT16  mode;
    UINT32  nlink;
    UINT32  uid;
    UINT32  gid;
    UINT32  rdev;
    INT64   size;
    INT32   atime_sec;
    INT32   atime_nsec;
    INT32   mtime_sec;
    INT32   mtime_nsec;
    INT32   ctime_sec;
    INT32   ctime_nsec;
    UINT32  blksize;
    UINT64  blocks;
} AMBA_IPC_RFS_STAT_s
```


Parameters:

Type	Parameter	Description
const char *	**name**	The name of the file or directory
AMBA_IPC_ RFS_STAT_s *	**stat**	The data structure is used to store the file information.

*Table 6-32. Parameters for RFS API **AmbaIPC_stat()**.*

Returns:

Return	Description
0	Successful in getting the file information
< 0	Failed to get the file information

*Table 6-33. Returns for RFS API **AmbaIPC_stat()**.*

Example:

```
char *path="/tmp/test.c";
AMBA_IPC_RFS_STAT_s stat;
if(AmbaIPC_stat(path, &stat) == 0)
       AmbaShell_Print(env, "%s \t", dirent->name);
```

See Also:

None

6.2.18 AmbaIPC_getdev

API Syntax:

AmbaIPC_getdev (const char * path, AMBA_IPC_RFS_DEVINF_s *devinf)

Function Description:

- This function is used to get the device information.
- The data structure storing the device information is defined as follows:

```
typedef struct _AMBA_IPC_RFS_DEVINF_s_ {
    UINT32 cls;
    UINT32 ecl;
    UINT32 bps;
    UINT32 spc;
    UINT32 cpg;
    UINT32 ecg;
    AMBA_IPC_RFS_FMT_TYPE_e fmt;
} AMBA_IPC_RFS_DEVINF_s;
```

- The supported format types are listed:

```
typedef enum _AMBA_IPC_RFS_FMT_TYPE_e_
    { AMBA_IPC_RFS_FMT_FAT12 = 0,
      AMBA_IPC_RFS_FMT_FAT16 = 1,
      AMBA_IPC_RFS_FMT_FAT32 = 2,
      AMBA_IPC_RFS_FMT_EXFAT = 3
} AMBA_IPC_RFS_FMT_TYPE_e;
```


Parameters:

Type	Parameter	Description
const char *	**path**	The path where the device is located.
AMBA_IPC_RFS_DEVINF_s *	**devif**	The data structure is used to store the device information.

*Table 6-34. Parameters for RFS API **AmbaIPC_getdev()**.*

Returns:

Return	Description
0	Successful in getting the device information
< 0	Failed to get the device information

*Table 6-35. Returns for RFS API **AmbaIPC_getdev()**.*

Example:

```
int rval, n;
AMBA_IPC_RFS_DEVINF_s dev_inf;
char *path = "/tmp/test"
rval = AmbaIPC_getdev(path, &dev_inf);
     if (rval == -1) {
         AmbaShell_Print(env, "getdev error (%d)\n", rval);
         return;
      }
      n = (((UINT64)(dev_inf.ecl * dev_inf.spc)) * dev_inf.bps) / tsize;
```

See Also:

None

6.2.19 AmbaIPC_feof

API Syntax:

AmbaIPC_feof (void *fp)

Function Description:

- This function is used to check whether the file pointer points to the end-of-file.

Parameters:

Type	Parameter	Description
void *	**fp**	File pointer

*Table 6-36. Parameters for RFS API **AmbaIPC_feof()**.*

Returns:

Return	Description
0	Reaches the end-of-file
< 0	Does not reach the end-of-file

*Table 6-37. Returns for RFS API **AmbaIPC_feof()**.*

Example:

```
if( !AmbaIPC_feof(fp) ) {
      AmbaPrint("Reach the end-of-file");
}
```

See Also:

AmbaIPC_ftell()
AmbaIPC_fseek()

6.2.20 AmbaIPC_opendir

API Syntax:

AmbaIPC_opendir (const char *name)

Function Description:

- This function is used to open a remote directory.

Parameters:

Type	Parameter	Description
const char *	**name**	The name of the remote directory

*Table 6-38. Parameters for RFS API **AmbaIPC_opendir()**.*

Returns:

Return	Description
void *	The pointer to point to the directory structure storing the directory information

*Table 6-39. Returns for RFS API **AmbaIPC_opendir()**.*

Example:

```
AMBA_IPC_RFS_DIRENT_s *dirent;
void *dirp;
dirp = AmbaIPC_opendir("/tmp/test");
```

See Also:

AmbaIPC_readdir()
AmbaIPC_closedir()

6.2.21 AmbaIPC_readdir

API Syntax:

AmbaIPC_readdir (void *dirp)

Function Description:

- This function is used to read from the remote directory.
- The data structure storing the information for the file read from the remote directory is defined as follows:

```
typedef struct {
    UINT64  ino;
    UINT64  off;
    UINT16  reclen;
    UINT8   type;
    char    name[0];
} AMBA_IPC_RFS_DIRENT_s;
```

Parameters:

Type	Parameter	Description
void *	**dirp**	The pointer to the structure storing the information for a remote directory.

*Table 6-40. Parameters for RFS API **AmbaIPC_readdir()**.*

Returns:

Return	Description
AMBA_IPC_RFS_DIRENT_s*	Pointer to the directory structure storing the information about the file read from the remote directory

*Table 6-41. Returns for RFS API **AmbaIPC_readdir()**.*

Example:

```
AMBA_IPC_RFS_DIRENT_s *dirent;
char *slot=”/tmp”
dirent = AmbaIPC_readdir(dirp);
    while (dirent != NULL){
      memset(path, 0x0, 100);
      strcpy(path, slot);
      strcat(path, “/”);
      strcat(path, dirent->name);
      dirent = AmbaIPC_readdir(dirp);
    }
```

See Also:

AmbaIPC_opendir()
AmbaIPC_closedir()

6.2.22 AmbaIPC_closedir

API Syntax:

AmbaIPC_closedir (void *dirp)

Function Description:

- This function is used to close a remote directory.

Parameters:

Type	Parameter	Description
void *	**dirp**	Pointer to the structure storing the information for the remote directory.

*Table 6-42. Parameters for RFS API **AmbaIPC_closedir()**.*

Returns:

Return	Description
0	Close the remote directory successfully
< 0	Failed to close the remote directory

*Table 6-43. Returns for RFS API **AmbaIPC_closedir()**.*

Example:

```
AmbaIPC_closedir(dirp);
```

See Also:

AmbaIPC_opendir()
AmbaIPC_readdir()

6.2.23 AmbaIPC_chdir

API Syntax:

AmbaIPC_chdir (const char* path)

Function Description:

- This function is used to change the current working directory to the directory specified in the path.

Parameters:

Type	Parameter	Description
const char *	**path**	The target path

*Table 6-44. Parameters for RFS API **AmbaIPC_chdir()**.*

Returns:

Return	Description
0	Successfully changed the working directory
< 0	Failed to change the working directory

*Table 6-45. Returns for RFS API **AmbaIPC_chdir()**.*

Example:

```
char *path = "/tmp/test";
AmbaIPC_chdir(path);
```

See Also:

None

7 Spin Lock (Linux)

7.1 AmbaLink Spin Lock (Linux): Overview

This chapter discusses the AmbaLink APIs related to the spin lock implementation on the Linux operating system.

An operating system (OS) can synchronize its internal execution with built-in synchronization primitives, such as spinlock, mutex, semaphore, event flags, message queue, completion, etc. However, if different operating systems need to share a common, system-wide resource, such as memory, I2C bus, or SD-card slot; a set of global OS-agnostic synchronization primitives are needed.

AmbaLink provides an access control scheme for all operating systems on the Ambarella platform. Once an operating system acquires a system resource through a lock, it has the exclusive access to that resource until it voluntarily releases the lock. Spin lock is one kind of lock provided and the related APIs are introduced in this Chapter.

A spin lock is a synchronization object used to ensure mutually exclusive access to a given thread. When a thread is enabled with a spin lock, any other thread attempting to acquire it will wait in a loop (“spinning”) while repeatedly polling to check if the lock is available. The polling thread remains active and retains CPU control while spinning; however, no useful task can be performed under this condition. Therefore, it is critical that a spin lock is held for as little time as possible to minimize the unnecessary use of CPU resources.

7.2 AmbaLink Spin Lock (Linux): List of APIs

- aipc_spin_lock
- aipc_spin_unlock
- aipc_spin_lock_irqsave
- aipc_spin_unlock_irqrestore

7.2.1 aipc_spin_lock

API Syntax:

aipc_spin_lock (int id)

Function Description:

- This function is used to assign a spin lock to a specified ID **@id**.
- The **aipc_spin_lock** function does not disable interrupts. This function is used exclusively to lock a specified spin lock ID **@id** across dual operating systems.
- Note that if an interrupt service routine (ISR) is anticipated to access a spin lock, the **aipc_spin_lock_irqsave** function should be used; otherwise, system deadlock may result.
- The following spin lock IDs are currently available:

```
typedef enum _AMBA_IPC_SPINLOCK_IDX_e_ {
            AMBA_IPC_SPINLOCK_GPIO,
            AMBA_IPC_NUM_SPINLOCK      /* Total number of global spin lock */
} AMBA_IPC_SPINLOCK_IDX_e;
```

Parameters:

Type	Parameter	Description
int	**id**	Spin lock ID to be locked

*Table 7-1. Parameters for Spin Lock API **aipc_spin_lock()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to lock the given ID

*Table 7-2. Returns for Spin Lock API **aipc_spin_lock()**.*

Example:

```
aipc_spin_lock(AMBA_IPC_SPINLOCK_GPIO);
```

See Also:

AmbaIPC_SpinLock()

7.2.2 aipc_spin_unlock

API Syntax:

aipc_spin_unlock (int id)

Function Description:

- This function is used to unlock a spin lock with the specified ID **@id**.
- The **aipc_spin_unlock** function does not disable interrupts. This function is used exclusively to unlock a specified spin lock ID **@id** across dual operating systems.
- Note that if an interrupt service routine (ISR) is anticipated to access a spin lock, the **aipc_spin_unlock_irqrestore** function should be used; otherwise, it may result in a system deadlock.
- The following spin lock IDs are currently available:

```
typedef enum _AMBA_IPC_SPINLOCK_IDX_e_ {
            AMBA_IPC_SPINLOCK_GPIO,
            AMBA_IPC_NUM_SPINLOCK      /* Total number of global spin lock */
} AMBA_IPC_SPINLOCK_IDX_e;
```

Parameters:

Type	Parameter	Description
int	**id**	Spin lock ID to be unlocked

*Table 7-3. Parameters for Spin Lock API **aipc_spin_unlock()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to unlock the given ID

*Table 7-4. Returns for Spin Lock API **aipc_spin_unlock()**.*

Example:

```
aipc_spin_unlock(AMBA_IPC_SPINLOCK_GPIO);
```

See Also:

AmbaIPC_SpinUnlock()

7.2.3 aipc_spin_lock_irqsave

API Syntax:

aipc_spin_lock_irqsave (int id, unsigned long *flags)

Function Description:

- This function is used to assign a global spin lock to a specified ID **@id**. The function is used when an interrupt service routine (ISR) is anticipated to access a spin lock.
- The **aipc_spin_lock_irqsave** function will disable interrupts and save the IRQ status to a variable **@ flags**.
- The following spin lock IDs are currently available:

```
typedef enum _AMBA_IPC_SPINLOCK_IDX_e_ {
            AMBA_IPC_SPINLOCK_GPIO,
            AMBA_IPC_NUM_SPINLOCK      /* Total number of global spin lock */
} AMBA_IPC_SPINLOCK_IDX_e;
```

Parameters:

Type	Parameter	Description
int	**id**	Spin lock ID to be locked
unsigned long *	**flags**	Saved IRQ status

*Table 7-5. Parameters for Spin Lock API **aipc_spin_lock_irqsave()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to unlock the given ID

*Table 7-6. Returns for Spin Lock API **aipc_spin_lock_irqsave()**.*

Example:

AmbaIPC_SpinLockIrqSave()

See Also:

```
unsigned long Flags;
aipc_spin_lock_irqsave(SpinID, &Flags);
```

7.2.4 aipc_spin_unlock_irqrestore

API Syntax:

aipc_spin_unlock_irqrestore (int id, unsigned long flags)

Function Description:

- This function is used to unlock the global spin lock assigned to a specified ID **@id**. The function is used when an interrupt service routine (ISR) is anticipated to access a spin lock.
- This function will restore the IRQ status from a variable **@flags** and enable interrupts.
- The following spin lock IDs are currently available:

```
typedef enum _AMBA_IPC_SPINLOCK_IDX_e_ {
            AMBA_IPC_SPINLOCK_GPIO,
            AMBA_IPC_NUM_SPINLOCK      /* Total number of global spin lock */
} AMBA_IPC_SPINLOCK_IDX_e;
```

Parameters:

Type	Parameter	Description
int	**id**	Spin lock ID to be locked
unsigned long	**flags**	IRQ status to be restored

*Table 7-7. Parameters for Spin Lock API **aipc_spin_unlock_irqrestore ()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to unlock the given ID

*Table 7-8. Returns for Spin Lock API **aipc_spin_unlock_irqrestore()**.*

Example:

```
unsigned long Flags;
aipc_spin_lock_irqsave(SpinID, &Flags);
aipc_spin_unlock_irqrestore(SpinID, Flags);
```

See Also:

AmbaIPC_SPinUnlockIrqRestore()

8 Mutex (Linux)

8.1 AmbaLink Mutex (Linux): Overview

This chapter discusses the AmbaLink APIs related to mutex implementation on the Linux operating system.

Similar in concept to a spin lock, a mutex (MUTual EXclusion) refers to a synchronization object used to ensure mutually exclusive access to a given thread. The fundamental difference between a mutex and a spin lock is that, while the mutex is locked by a thread, the other threads waiting on the lock can accomplish tasks without waiting.

8.2 AmbaLink Mutex (Linux): List of APIs

- aipc_mutex_lock
- aipc_mutex_unlock

8.2.1 aipc_mutex_lock

API Syntax:

aipc_mutex_lock (int id)

Function Description:

- This function is used to lock the global mutex **@id**.
- This service attempts to obtain exclusive ownership of the specified mutex. If another execution unit (EU) currently owns the mutex, the calling thread is suspended until the mutex is released by the remote EU. Otherwise, it behaves exactly the same as a native Linux mutex.
- The following mutex IDs are currently available:

```
typedef enum _AMBA_IPC_MUTEX_IDX_e_ {
            AMBA_IPC_MUTEX_I2C_CHANNEL0 = 0,
            AMBA_IPC_MUTEX_I2C_CHANNEL1,
            AMBA_IPC_MUTEX_I2C_CHANNEL2,
            AMBA_IPC_MUTEX_SPI_CHANNEL0,
            AMBA_IPC_MUTEX_SPI_CHANNEL1,
            AMBA_IPC_MUTEX_SPI_CHANNEL2,
            AMBA_IPC_MUTEX_SD0,
            AMBA_IPC_MUTEX_SD1,
            AMBA_IPC_MUTEX_FIO,
            AMBA_IPC_MUTEX_GPIO,
            AMBA_IPC_MUTEX_PLL,
            AMBA_IPC_NUM_MUTEX      /* Total number of global mutex */
} AMBA_IPC_MUTEX_IDX_e;
```

Parameters:

Type	Parameter	Description
int	**id**	Mutex ID to be locked

*Table 8-1. Parameters for Mutex API **aipc_mutex_lock()**.*

Returns:

None

Example:

```
aipc_mutex_lock(AMBA_IPC_MUTEX_SD1);
```

See Also:

AmbaIPC_MutexTake()

8.2.2 aipc_mutex_unlock

API Syntax:

aipc_mutex_unlock (int id)

Function Description:

- This function is used to unlock the global mutex **@id**.
- The following mutex IDs are currently available:

```
typedef enum _AMBA_IPC_MUTEX_IDX_e_ {
            AMBA_IPC_MUTEX_I2C_CHANNEL0 = 0,
            AMBA_IPC_MUTEX_I2C_CHANNEL1,
            AMBA_IPC_MUTEX_I2C_CHANNEL2,
            AMBA_IPC_MUTEX_SPI_CHANNEL0,
            AMBA_IPC_MUTEX_SPI_CHANNEL1,
            AMBA_IPC_MUTEX_SPI_CHANNEL2,
            AMBA_IPC_MUTEX_SD0,
            AMBA_IPC_MUTEX_SD1,
            AMBA_IPC_MUTEX_FIO,
            AMBA_IPC_MUTEX_GPIO,
            AMBA_IPC_MUTEX_PLL,
            AMBA_IPC_NUM_MUTEX          /* Total number of global mutex */
} AMBA_IPC_MUTEX_IDX_e;
```

Parameters:

Type	Parameter	Description
int	**id**	Mutex ID to be unlocked

*Table 8-2. Parameters for Mutex API **aipc_mutex_unlock()**.*

Returns:

None

Example:

```
aipc_mutex_unlock(AMBA_IPC_MUTEX_SD1);
```

See Also:

AmbaIPC_MutexTake()

9 Remote Processor Messaging (Linux)

9.1 AmbaLink RPMsg (Linux): Overview

This chapter discusses the AmbaLink APIs related to Remote Processor Messaging (RPMsg) on the Linux operating system.

The overall structure of the AmbaLink framework is based on RPMsg operations. RPMsg is a VirtIO-based messaging bus that allows kernel drivers to communicate with remote system processors via inter-process communication (IPC). In turn, these kernel drivers expose the appropriate user-space interfaces as required. In the context of AmbaLink, RPMsg operations are used to send messages to, and receive messages from, processes running on a separate core.

Each RPMsg device serves as a communication channel to a remote processor; these devices are typically referred to as channels for this reason. RPMsg channels are identified by a textual name and have both a local (i.e., source) and a remote (i.e., destination) RPMsg address. The Linux kernel implements the RPMsg host driver and ThreadX implements the RPMsg device, which is also called the RPMsg channel.

When a kernel driver begins listening on a channel, its Receive callback is bound with a unique RPMsg local address (32-bit integer). Therefore, when inbound messages arrive, the RPMsg core dispatches them to the appropriate driver according to their destination address by invoking the driver's Receive handler with the payload of the inbound message.

Refer to `linux/Documentation/rpmsg.txt` for additional detail.

9.2 AmbaLink RPMsg (Linux): List of APIs

- register_rpmsg_driver
- unregister_rpmsg_driver
- rpmsg_trysend
- rpmsg_send

9.2.1 register_rpmsg_driver

API Syntax:

register_rpmsg_driver (struct rpmsg_driver *rpdrv)

Function Description:

- The **register_rpmsg_driver** function is used to register an RPMsg driver with the RPMsg bus.
- When using this function, the following information must be provided:
  - A pointer to an RPMsg driver `struct` record, which contains the **probe()** and **remove()** driver functions
  - A Receive (Rx) callback
  - An **id_table** specifying the names of relevant channels.

Parameters:

Type	Parameter	Description
struct rpmsg_ driver *	**rpdrv**	Pointer to a **rpmsg_driver** `struct`. Plese refer to Section 9.2.1.1 below for definition.

*Table 9-1. Parameters for RPMsg API **register_rpmsg_driver()**.*

Returns:

Return	Description
0	Completed successfully
Nonzero	Register failed

*Table 9-2. Returns for RPMsg API **register_rpmsg_driver()**.*

Example:

```
static struct rpmsg_driver rpmsg_echo_driver = {
      .drv.name   = KBUILD_MODNAME,
      .drv.owner  = THIS_MODULE,
      .id_table   = rpmsg_echo_id_table,
      .probe            = rpmsg_echo_probe,
      .callback   = rpmsg_echo_cb,
      .remove           = rpmsg_echo_remove,
};

static int __init rpmsg_echo_init(void)
{
      return register_rpmsg_driver(&rpmsg_echo_driver);
}
```

See Also:

None

9.2.1.1 register_rpmsg_driver > rpmsg_driver

Type	Field	Description
device_driver	**drv**	Underlying device driver
rpmsg_device_id	***id_table**	RPMsg IDs serviced by this driver
int	**(*probe)**	Invoked when a matching RPMsg channel (i.e. device) is found
void	**(*remove)**	Invoked when the RPMsg channel is removed
void	**(*callback)**	Invoked when an inbound message is received on the channel

*Table 9-3. Definition of **rpmsg_driver** for RPMsg API **register_rpmsg_driver()**.*

9.2.2 unregister_rpmsg_driver

API Syntax:

unregister_rpmsg_driver (struct rpmsg_driver *rpdrv)

Function Description:

- This function is used to de-register an RPMsg driver from the RPMsg bus.
- A pointer to a previously-registered **rpmsg_driver** `struct` must be provided.

Parameters:

Type	Parameter	Description
struct rpmsg_ driver *	**rpdrv**	Pointer to a **rpmsg_driver** `struct`.

*Table 9-4. Parameters for RPMsg API **unregister_rpmsg_driver()**.*

Returns:

Return	Description
0	Completed successfully
Nonzero	Failed to de-register a channel

*Table 9-5. Returns for RPMsg API **unregister_rpmsg_driver()**.*

Example:

```
static struct rpmsg_driver rpmsg_echo_driver = {
      .drv.name   = KBUILD_MODNAME,
      .drv.owner  = THIS_MODULE,
      .id_table   = rpmsg_echo_id_table,
      .probe      = rpmsg_echo_probe,
      .callback   = rpmsg_echo_cb,
      .remove     = rpmsg_echo_remove,
};

static int __init rpmsg_echo_init(void)
{
      return register_rpmsg_driver(&rpmsg_echo_driver);
}

static void __exit rpmsg_echo_fini(void)
{
      unregister_rpmsg_driver(&rpmsg_echo_driver);
}
```

See Also:

register_rpmsg_driver

9.2.3 rpmsg_trysend

API Syntax:

rpmsg_trysend (struct rpmsg_channel *rpdev, void *data, int len)

Function Description:

- This function is used to send a message to the remote processor on a specified channel.
- The caller should specify the channel, the data to be sent, and the data length (in bytes).
- The message will be sent across the specified channel (i.e., its source and destination address fields will be set to those of the channel).
- If no Transmit (Tx) buffers are available, the function will immediately return **-ENOMEM** (i.e., without waiting until a buffer becomes available.)
- Currently, this function can only be called from a process context.

Parameters:

Type	Parameter	Description
struct rpmsg_ channel *	**rpdev**	The RPMsg channel
void *	**data**	Payload of message
int	**len**	Length of payload

*Table 9-6. Parameters for RPMsg API **rpmsg_trysend()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to send the message

*Table 9-7. Returns for RPMsg API **rpmsg_trysend()**.*

Example:

```
static void rpmsg_echo_cb(struct rpmsg_channel *rpdev, void *data, int len,
void *priv, u32 src)
{
      printk("[ %20s ] recv msg: [%s] from 0x%x and len %d\n",
             __func__, (const char*)data, src, len);

      /* Echo the recved message back */
      rpmsg_trysend(rpdev, data, len);
}
```

See Also:

None

9.2.4 rpmsg_send

API Syntax:

rpmsg_send (struct rpmsg_channel *rpdev, void *data, int len)

Function Description:

- This function is used to send a message to the remote processor on a specified channel.
- The caller should specify the channel, the data to be sent, and the data length (in bytes).
- The message will be sent across the specified channel (i.e., its source and destination address fields will be set to those of the channel).
- If no Transmit (Tx) buffers are available, the function will block until a buffer becomes available (i.e., until the remote processor consumes a Tx buffer and returns it to the VirtIO used descriptor ring), or until a timeout period of 15 seconds lapses. When the latter occurs, a **-ERESTARTSYS** message is returned.
- Currently, this function can only be called from a process context.

Parameters:

Type	Parameter	Description
struct rpmsg_ channel *	**rpdev**	The RPMsg channel
void *	**data**	Payload of message
int	**len**	Length of payload

*Table 9-8. Parameters for RPMsg API **rpmsg_send()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to send the message

*Table 9-9. Returns for RPMsg API **rpmsg_send()**.*

Example:

```
static void rpmsg_echo_cb(struct rpmsg_channel *rpdev, void *data, int len,
void *priv, u32 src)
{
      printk("[ %20s ] recv msg: [%s] from 0x%x and len %d\n",
             __func__, (const char*)data, src, len);

      /* Echo the recved message back */
      rpmsg_send(rpdev, data, len);
}
```

See Also:

rpmsg_trysend()

10 Remote Procedure Calls (Linux)

10.1 AmbaLink RPC (Linux): Overview

This chapter discusses the AmbaLink APIs related to Remote Procedure Calls (RPCs) on the Linux operating system.

An RPC is a type of inter-process communication in which a program running on one processor executes a subroutine on another processor across a shared network. The AmbaLink RPC framework allows a software programmer—without knowing the details of the cross-processor interaction—to write code as though the subroutine will be executed locally.

10.2 AmbaLink RPC (Linux): List of APIs

- ambaipc_svc_register
- ambaipc_svc_unregister
- ambaipc_clnt_create
- ambaipc_clnt_destroy
- ambaipc_clnt_call
- ambaipc_strerror

10.2.1 ambaipc_svc_register

API Syntax:

ambaipc_svc_register (int prog, int vers, char *name, AMBA_IPC_PROG_INFO_s *info, int new_thread)

Function Description:

- This function registers an RPC service with program number **@prog** and version number **@vers**. After this function is called, the local host can service RPC requests with a (`prog, vers, xxx`) signature.

Parameters:

Type	Parameter	Description
int	**prog**	Specifies the program number
int	**vers**	Specifies the version number
char *	**name**	Specifies the name of the service
AMBA_IPC_ PROG_INFO_s *	**info**	Specifies the program information including communication modes, procedure functions and etc. Please refer to Section 5.2.1.1 for more details.
int	**new_thread**	Specifies if a new thread runs this svc or not

*Table 10-1. Parameters for RPC API **ambaipc_svc_register()**.*

Returns:

Return	Description
0	Completed successfully
Nonzero	Failed to register

*Table 10-2. Returns for RPC API **ambaipc_svc_register()**.*

Example:

For more details, please refer to `ambalink_sdk/pkg/ambaipc_test/svc_test.c`

```
{
            AMBA_IPC_PROG_INFO_s prog_info[1];
            prog_info->ProcNum = 2;
            prog_info->pProcInfo = malloc(
                  prog_info->ProcNum*sizeof(AMBA_IPC_PROC_s));
            prog_info->pProcInfo[0].Mode = AMBA_IPC_ASYNCHRONOUS;
            prog_info->pProcInfo[0].Proc =
                  (AMBA_IPC_PROC_f) &AmbaRpcProg_LU_Test_Print_Svc;
            prog_info->pProcInfo[1].Mode = AMBA_IPC_SYNCHRONOUS;
            prog_info->pProcInfo[1].Proc = (AMBA_IPC_PROC_f) &AmbaRpcProg_
              AmbaRpcProg_LU_Test_Sum_Svc;
            ambaipc_svc_register(AMBA_RPC_PROG_LU_TEST_PROG_ID,
              AMBA_RPC_PROG_LU_TEST_VER, "linux_svc_test",
              prog_info, 1);

}
```

See Also:

AmbaIPC_SvcRegister()
AmbaIPC_Svc_unRegister()

10.2.2 ambaipc_svc_unregister

API Syntax:

ambaipc_svc_unregister (int prog, int vers)

Function Description:

- This function de-registers an RPC service with program number **@prog** and version number **@vers**. After this function is called, the local host will reject any RPC request with a (`prog, vers, xxx`) signature.

Parameters:

Type	Parameter	Description
int	**prog**	Specifies the program number
int	**vers**	Specifies the version number

*Table 10-3. Parameters for RPC API **ambaipc_svc_unregister()**.*

Returns:

Return	Description
0	Completed successfully
Nonzero	Failed to de-register

*Table 10-4. Returns for RPC API **ambaipc_svc_unregister()**.*

Example:

For more details, please refer to `ambalink_sdk/pkg/ambaipc_test/svc_test.c`

```
{
      ambaipc_svc_register(AMBA_RPC_PROG_LU_TEST_PROG_ID,
AMBA_RPC_PROG_LU_TEST_VER, "linux_svc_test", prog_info, 1);
      ambaipc_svc_unregister(AMBA_RPC_PROG_LU_TEST_PROG_ID,
AMBA_RPC_PROG_LU_TEST_VER);

}
```

See Also:

AmbaIPC_SvcUnregister()
AmbaIPC_Svc_register()

10.2.3 ambaipc_clnt_create

API Syntax:

ambaipc_clnt_create (int host, int prog, int vers)

Function Description:

- This function is used to create an RPC client that can be used to make RPC calls with a (`prog, vers, xxx`) signature to the remote host.

Parameters:

Type	Parameter	Description
int	**host**	Specifies the name of remote host
int	**prog**	Specifies the program number
int	**vers**	Specifies the version number

*Table 10-5. Parameters for RPC API **ambaipc_clnt_create()**.*

Returns:

Return	Description
Nonzero	Newly created client
0	Creation failed

*Table 10-6. Returns for RPC API **ambaipc_clnt_create()**.*

Example:

For more details, please refer to `ambalink_sdk/pkg/ambaipc_test/clnt_test.c`.

```
      static void* clnt_thread_func(void *arg)
{
      int i, clnt, status, result;
      AMBA_RPC_PROG_TEST_SUM_ARG_s pSum[1];
      clnt = ambaipc_clnt_create(host, AMBA_RPC_PROG_R_TEST_PROG_ID,
      AMBA_RPC_PROG_R_TEST_VER);
      if (!clnt)
            goto done;

      /* test proc=1, batching mode */
      AmbaRpcProg_R_Test_Print_Clnt(msg, NULL, clnt);
```

```
        /* test proc=2, returns sume of array[0] and array[1] */
        for (i = 0; i < 32; i++) {
              pSum->a = pSum->b = i;
              status = AmbaRpcProg_R_Test_Sum_Clnt(pSum, &result, clnt);
              printf("clnt[%d]: status is %d, sum is %d\n", clnt, status, re-
sult);       }
        /* test IPC_REPLY_PARA_INVALID */
        pSum->a = pSum->b = 1024;
        status = AmbaRpcProg_R_Test_Sum_Clnt(pSum, &result, clnt);
        printf("clnt[%d]: status is %d\n", clnt, status);
        ambaipc_clnt_destroy(clnt);

        /* test clnt_call after destroy */
        status = AmbaRpcProg_R_Test_Sum_Clnt(pSum, &result, clnt);
        printf("clnt[%d]: status is %d\n", clnt, status);
        done:
        pthread_exit(&result);
        }
```

See Also:

AmbaIPC_ClientCreate()

10.2.4 ambaipc_clnt_destroy

API Syntax:

ambaipc_clnt_destroy (int clnt_id)

Function Description:

- This function releases all the resources reserved by the RPC client **@clnt**.

Parameters:

Type	Parameter	Description
int	**clnt_id**	Pointer to the structure of the client handle

*Table 10-7. Parameters for RPC API **ambaipc_clnt_destroy()**.*

Returns:

Return	Description
0	Completed successfully

*Table 10-8. Returns for RPC API **ambaipc_clnt_destroy()**.*

Example:

For more details, please refer to *ambalink_sdk/pkg/ambaipc_test/clnt_test.c*

```
static void* clnt_thread_func(void *arg)
{
        int i, clnt, status, result;
        AMBA_RPC_PROG_TEST_SUM_ARG_s pSum[1];
        clnt = ambaipc_clnt_create(host, AMBA_RPC_PROG_R_TEST_PROG_ID,
        AMBA_RPC_PROG_R_TEST_VER);
        if (!clnt)
           goto done;

        /* test proc=1, batching mode */

        AmbaRpcProg_R_Test_Print_Clnt(msg, NULL, clnt);
```

```
        /* test proc=2, returns sume of array[0] and array[1] */
        for (i = 0; i < 32; i++) {
              pSum->a = pSum->b = i;
              status = AmbaRpcProg_R_Test_Sum_Clnt(pSum, &result, clnt);
              printf("clnt[%d]: status is %d, sum is %d\n", clnt, status,
              result);          }
        /* test IPC_REPLY_PARA_INVALID */
        pSum->a = pSum->b = 1024;
        status = AmbaRpcProg_R_Test_Sum_Clnt(pSum, &result, clnt);
        printf("clnt[%d]: status is %d\n", clnt, status);
        ambaipc_clnt_destroy(clnt);

        /* test clnt_call after destroy */
        status = AmbaRpcProg_R_Test_Sum_Clnt(pSum, &result, clnt);
        printf("clnt[%d]: status is %d\n", clnt, status);
        done:
              pthread_exit(&result);
  }
```

See Also:

AmbaIPC_ClientDestroy()

10.2.5 ambaipc_clnt_call

API Syntax:

ambaipc_clnt_call(int clnt_id, int proc, void *in, int in_len, void *out, int out_len, int timeout)

Function Description:

- This function attempts to make an RPC call with a (`prog, vers, xxx`) signature, where **@prog** and **@vers** come from client **@clnt**.
- The input parameter is located at address **in**, and the return parameter will be stored at address **out**.

Parameters:

Type	Parameter	Description
int	**clnt_id**	Pointer to the client handle that results from **clnt_create**
int	**proc**	Specifies the remote procedure number
void*	**in**	Pointer to the address of the procedure's arguments
int	**in_len**	Specifies the length of the argument in bytes
void*	**out**	Pointer to the address where the results are placed
int	**out_len**	Specifies the maximum length of the results buffer in bytes
int	**timeout**	Sets the time allowed for the results to return

*Table 10-9. Parameters for RPC API **ambaipc_clnt_call()**.*

Returns:

Return	Description
0	Completed successfully
1	Target service is not available
2	Input parameter is invalid
3	This signals a general system error, such as an out-of-memory error.
4	Timeout

*Table 10-10. Returns for RPC API **ambaipc_clnt_call()**.*

Example:

Program in server (ThreadX). For more details, please refer to `rtos/ssp/unittest/AmbaIPC_Test.c`.

```
{
      AMBA_IPC_PROG_INFO_s prog_info[1];
      prog_info->ProcNum = 2;
      prog_info->pProcInfo = AmbaLink_Malloc(
            prog_info->ProcNum*sizeof(AMBA_IPC_PROC_s));
      prog_info->pProcInfo[0].Mode = AMBA_IPC_ASYNCHRONOUS;
      prog_info->pProcInfo[1].Mode = AMBA_IPC_SYNCHRONOUS;
      prog_info->pProcInfo[0].Proc = (AMBA_IPC_PROC_f)
        &AmbaRpcProg_R_Test_Print_Svc;;
      prog_info->pProcInfo[1].Proc = (AMBA_IPC_PROC_f)
        &AmbaRpcProg_R_Test_Sum_Svc;
      AmbaIPC_SvcRegister(AMBA_RPC_PROG_R_TEST_PROG_ID,
      AMBA_RPC_PROG_R_TEST_VER, "test_svc",15, stack, 0x1000, prog_info, 1);
      }
```

Program in client (Linux). For more details, please refer to `ambalink_sdk/pkg/ambaipc_test/clnt_test.c`

```
      static void* clnt_thread_func(void *arg)
{
      int i, clnt, status, result;
      AMBA_RPC_PROG_TEST_SUM_ARG_s pSum[1];
      clnt = ambaipc_clnt_create(host, AMBA_RPC_PROG_R_TEST_PROG_ID,
      AMBA_RPC_PROG_R_TEST_VER);
      if (!clnt)
            goto done;
      /* test proc=1, batching mode */
      AmbaRpcProg_R_Test_Print_Clnt(msg, NULL, clnt);

      /* test proc=2, returns sume of array[0] and array[1] */
      for (i = 0; i < 32; i++) {
            pSum->a = pSum->b = i;
            status = AmbaRpcProg_R_Test_Sum_Clnt(pSum, &result, clnt);
            printf("clnt[%d]: status is %d, sum is %d\n", clnt, status, re-
sult);      }

      /* test IPC_REPLY_PARA_INVALID */
      pSum->a = pSum->b = 1024;
      status = AmbaRpcProg_R_Test_Sum_Clnt(pSum, &result, clnt);
      printf("clnt[%d]: status is %d\n", clnt, status);

      ambaipc_clnt_destroy(clnt);

      /* test clnt_call after destroy */
      status = AmbaRpcProg_R_Test_Sum_Clnt(pSum, &result, clnt);
      printf("clnt[%d]: status is %d\n", clnt, status);

      done:
      pthread_exit(&result);
}
```

See Also:

AmbaIPC_ClientCall()

10.2.6 ambaipc_strerror

API Syntax:

ambaipc_strerror (int error)

Function Description:

- This function is used to get the description of error codes.

Parameters:

Type	Parameter	Description
int	**error**	The error code

*Table 10-11. Parameters for RPC API **ambaipc_strerror()**.*

Returns:

Return	Description
const char *	The error description

*Table 10-12. Returns for RPC API **ambaipc_strerror()**.*

Example:

```
int status;
status = AmbaRpcProg_R_Test_Sum_Clnt(pSum, &result, clnt);
printf("clnt[%d]:  status is %d\n", clnt, ambaipc_strerror (status));
```

See Also:

None

11 AmbaLink Service Related APIs (ThreadX)

11.1 AmbaLink Service Related APIs (ThreadX): Overview

This chapter provides information on the AmbaLink service related APIs on the ThreadX real-time operating system (RTOS).

Ambarella provides blocking APIs and callback functions for users. The callback function is used to notify the system that an important event is complete. Users can hook up their function to the API to perform other initializations. Note that users should hook up their APIs before **AmbaLink_Init()**.

11.2 AmbaLink Service Related APIs (ThreadX): List of APIs

- AmbaLink_Init
- AmbaLink_Load
- AmbaLink_Boot
- AmbaLink_BootType
- AmbaIPC_LinkCtrlSuspendLinux
- AmbaIPC_LinkCtrlWaitSuspendLinux
- AmbaIPC_LinkCtrlResumeLinux
- AmbaIPC_LinkCtrlWaitResumeLinux
- AmbaLink_UserIpcInitCallBack
- AmbaIPC_LinkCtrlSuspendDoneCallBack
- AmbaIPC_LinkCtrlResumeLinuxDoneCallBack
- AmbaHiber_InitCallBack
- AmbaLink_SetIrqOwner
- AmbaLink_GetIrqOwner
- AmbaLink_ChangeBossPriority
- AmbaLink_AdjustBossSchedulePeriod
- AmbaLink_ForceScheduleEnable

11.2.1 AmbaLink_Init

API Syntax:

AmbaLink_Init (void)

Function Description:

- This function is the entry point of AmbaLink and is used to initialize AmbaLink.

Parameters:

None

Returns:

None

Example:

None

See Also:

AmbaLink_Load()
AmbaLink_Boot()

11.2.2 AmbaLink_Load

API Syntax:

AmbaLink_Load (void)

Function Description:

- This function is used to load Linux. It is usually called after **AmbaLink_Init()**.

Parameters:

None

Returns:

None

Example:

None

See Also:

AmbaLink_Load()
AmbaLink_Boot()

11.2.3 AmbaLink_Boot

API Syntax:

AmbaLink_Boot (UINT32 TimeOut)

Function Description:

- After loading Linux, this function is used to boot Linux.

Parameters:

Type	Parameter	Description
UINT32	**TimeOut**	The value of time-out in miniseconds

*Table 11-1. Parameters for AmbaLink Service Related API **AmbaLink_Boot()**.*

Returns:

Return	Description
0	Success
None zero	Boot failed

*Table 11-2. Returns for AmbaLink Service Related API **AmbaLink_Boot()**.*

Example:

None

See Also:

AmbaLink_Init()
AmbaLink_Load()

11.2.4 AmbaLink_BootType

API Syntax:

AmbaLink_BootType (UINT32 TimeOutMs)

Function Description:

- This function is used to return the boot type of Linux.
- This is a blocking API and is blocked before AmbaLink IPC is ready.

Parameters:

Type	Parameter	Description
UINT32	**TimeOutMs**	The value of time-out in miniseconds

*Table 11-3. Parameters for AmbaLink Service Related API **AmbaLink_BootType()**.*

Returns:

Return	Description
0	Cold boot
1	Warm boot
2	Hibernation boot

*Table 11-4. Returns for AmbaLink Service Related API **AmbaLink_BootType()**.*

Example:

```
{
        int BootType;

        BootType = AmbaLink_BootType(5000);

        AmbaPrint("Linux is %s and IPC is ready!",
        (BootType == 0) ? "ColdBoot" :
        (BootType == 1) ? "WarmBoot" : "Hibernation Boot");

}
```

See Also:

None

11.2.5 AmbaIPC_LinkCtrlSuspendLinux

API Syntax:

AmbaIPC_LinkCtrlSuspendLinux (UINT32 SuspendMode)

Function Description:

- This function is used to suspend Linux to disk or RAM according to SuspendMode.
- ```
  typedef enum _AMBA_LINK_SUSPEND_MODE_e_ {
     AMBA_LINK_HIBER_TO_DISK = 0,
     AMBA_LINK_HIBER_TO_RAM,
     AMBA_LINK_STANDBY_TO_RAM,
     AMBA_LINK_SLEEP_TO_RAM,
    } AMBA_LINK_SUSPEND_MODE_e;
  ```

Parameters:

Type	Parameter	Description
UINT32	**SuspendMode**	The mode to suspend. Please refer to **AMBA_LINK_SUSPEND_MODE_e**.

*Table 11-5. Parameters for AmbaLink Service Related API **AmbaIPC_LinkCtrlSuspendLinux()**.*

Returns:

Return	Description
0	Success in sending suspend command to Linux
- 1	Failure in sending suspend command to Linux

*Table 11-6. Returns for AmbaLink Service Related API **AmbaIPC_LinkCtrlSuspendLinux()**.*

Example:

```
AmbaPrint("AmbaIPC_LinkCtrlSuspendLinux(%d)\n");
AmbaIPC_LinkCtrlSuspendLinux(Mode);

if (AmbaIPC_LinkCtrlWaitSuspendLinux(5000) == OK)
    AmbaPrint("AmbaIPC_LinkCtrlWaitSuspendLinux(%d) done.\n", Mode);
```

See Also:

None
```

11.2.6 AmbaIPC_LinkCtrlWaitSuspendLinux

API Syntax:

AmbaIPC_LinkCtrlWaitSuspendLinux (UINT32 TimeOutMs)

Function Description:

- This function is used to suspend Linux to disk or RAM according to SuspendMode.
-
```
typedef enum _AMBA_LINK_SUSPEND_MODE_e_ {
   AMBA_LINK_HIBER_TO_DISK = 0,
   AMBA_LINK_HIBER_TO_RAM,
   AMBA_LINK_STANDBY_TO_RAM,
   AMBA_LINK_SLEEP_TO_RAM,
  } AMBA_LINK_SUSPEND_MODE_e;
```

Parameters:

Type	Parameter	Description
UINT32	**TimeOutMs**	The value of time-out in miniseconds

*Table 11-7. Parameters for AmbaLink Service Related API **AmbaIPC_LinkCtrlWaitSuspendLinux()**.*

Returns:

Return	Description
0	Success in waiting for Linux suspend state to be done
Nonzero	Failure in waiting for acknowledgement of the completion of Linux suspend state

*Table 11-8. Returns for AmbaLink Service Related API **AmbaIPC_LinkCtrlWaitSuspendLinux()**.*

Example:

```
AmbaPrint("AmbaIPC_LinkCtrlSuspendLinux(%d)\n");
AmbaIPC_LinkCtrlSuspendLinux(Mode);

if (AmbaIPC_LinkCtrlWaitSuspendLinux(5000) == OK)
    AmbaPrint("AmbaIPC_LinkCtrlWaitSuspendLinux(%d) done.\n", Mode);
```

See Also:

AmbaIPC_LinkCtrlSuspendLinux()

11.2.7 AmbaIPC_LinkCtrlResumeLinux

API Syntax:

AmbaIPC_LinkCtrlResumeLinux (UINT32 SuspendMode)

Function Description:

- This function is used to resume Linux from suspend state according to SuspendMode.
-
```
typedef enum _AMBA_LINK_SUSPEND_MODE_e_ {
      AMBA_LINK_HIBER_TO_DISK = 0,
      AMBA_LINK_HIBER_TO_RAM,
      AMBA_LINK_STANDBY_TO_RAM,
      AMBA_LINK_SLEEP_TO_RAM,
  } AMBA_LINK_SUSPEND_MODE_e;
```

Parameters:

Type	Parameter	Description
UINT32	**SuspendMode**	The suspend mode to resume from. Please refer to **AMBA_LINK_SUSPEND_MODE_e**.

*Table 11-9. Parameters for AmbaLink Service Related API **AmbaIPC_LinkCtrlWaitSuspendLinux**().*

Returns:

Return	Description
0	Success in resuming for Linux and Linux start to run
Nonzero	Failure in resuming Linux

*Table 11-10. Returns for AmbaLink Service Related API **AmbaIPC_LinkCtrlWaitSuspendLinux()**.*

Example:

```
AmbaPrint("AmbaIPC_LinkCtrlResumeLinux(%d)\n", Mode);
AmbaIPC_LinkCtrlResumeLinux(Mode);

if (AmbaIPC_LinkCtrlWaitResumeLinux(5000) == OK)
    AmbaPrint("AmbaIPC_LinkCtrlWaitResumeLinux(%d) done.\n", Mode);
```

See Also:

AmbaIPC_LinkCtrlWaitResumeLinux()

11.2.8 AmbaIPC_LinkCtrlWaitResumeLinux

API Syntax:

AmbaIPC_LinkCtrlWaitResumeLinux (UINT32 TimeOutMs)

Function Description:

- This function is used to wait for resume Linux to be completed.

Parameters:

Type	Parameter	Description
UINT32	**TimeOutMs**	The value of time-out in miniseconds

*Table 11-11. Parameters for AmbaLink Service Related API **AmbaIPC_LinkCtrlWaitResumeLinux**().*

Returns:

Return	Description
0	Success in resuming the resume Linux state. At this time, the IPC is ready.
Nonzero	Failure in waiting for Linux resume state to be completed

*Table 11-12. Returns for AmbaLink Service Related API **AmbaIPC_LinkCtrlWaitSuspendLinux()**.*

Example:

```
AmbaPrint("AmbaIPC_LinkCtrlResumeLinux(%d)\n", Mode);
AmbaIPC_LinkCtrlResumeLinux(Mode);

if (AmbaIPC_LinkCtrlWaitResumeLinux(5000) == OK)
    AmbaPrint("AmbaIPC_LinkCtrlWaitResumeLinux(%d) done.\n", Mode);
```

See Also:

AmbaIPC_LinkCtrlResumeLinux()

11.2.9 AmbaLink_UserIpcInitCallBack

API Syntax:

(*__AmbaLink_UserIpcInitCallBack__)(void)

Function Description:

- This function is called after the initialization of the default IPC channels.
- This function is called before restoring the RPMSG information while hibernation resumes.
- The initialization of the IPC channels created by the users should be called in this callback function.

Parameters:

None

Returns:

None

Example:

Program in ThreadX. For more details, please refer to `rtos/ssp/unittest/AmbaUserSysCtrl.c`.

```
{
      AmbaLink_UserIpcInitCallBack = AmbaIPC_TestInit;
}

int AmbaIPC_TestInit(void)
{
    Channel = AmbaIPC_Alloc("echo_cortex", MsgHandler);
      if (Channel == NULL) {
            AmbaPrint("%s: AmbaIPC_Alloc failed!", __func__);
            return -1;
      }
    AmbaIPC_RegisterChannel(Channel, NULL);

    AmbaTest_Init();
    AmbaTest_RegisterCommand("ipc", IpcTestEntry);

    AmbaLink_RPCInit();

    return 0;
}
```

See Also:

None

11.2.10 AmbaIPC_LinkCtrlSuspendDoneCallBack

API Syntax:

AmbaIPC_LinkCtrlSuspendDoneCallBack (UINT32 SuspendMode)

Function Description:

- This function is called after the suspend is completed for all suspend modes. While this function is called, the linux suspend is completed.

Parameters:

Type	Parameter	Description
UINT32	**SuspendMode**	The mode to suspend

*Table 11-13. Parameters for AmbaLink Service Related API **AmbaIPC_LinkCtrlSuspendDoneCallBack()**.*

Returns:

None

Example:

```
void AmbaLink_SuspendDoneCallBackImpl(UINT32 SuspendMode)
{
  switch (SuspendMode) {
  case AMBA_LINK_HIBER_TO_DISK:
     AmbaPrint("Linux suspend AMBA_LINK_HIBER_TO_DISK is done");
        break;
  case AMBA_LINK_HIBER_TO_RAM:
     AmbaPrint("Linux suspend AMBA_LINK_HIBER_TO_RAM is done");
        break;
  case AMBA_LINK_STANDBY_TO_RAM:
     AmbaPrint("Linux suspend AMBA_LINK_STANDBY_TO_RAM is done");
        break;
  case AMBA_LINK_SLEEP_TO_RAM:
     AmbaPrint("Linux suspend AMBA_LINK_SLEEP_TO_RAM is done");
     break;
    default:
      break;
    }
}
```

See Also:

AmbaIPC_LinkCtrlResumeLinuxDoneCallBack()

11.2.11 AmbaIPC_LinkCtrlResumeLinuxDoneCallBack

API Syntax:

(*__AmbaIPC_LinkCtrlResumeLinuxDoneCallBack__)(UINT32)

Function Description:

- This function is called after the IPC is ready without considering if it is a cold, warm, and hibernation boot. While this function is called, the linux resume is done.

Parameters:

Type	Parameter	Description
UINT32	**SuspendMode**	The suspend mode to resume from. Please refer to **AMBA_LINK_SUSPEND_MODE_e**.

*Table 11-14. Parameters for AmbaLink Service Related API **AmbaIPC_LinkCtrlResumeLinuxDoneCallBack()**.*

Returns:

None

Example:

Program in ThreadX. For more details, please refer to `rtos/ssp/unittest/AmbaUserSysCtrl.c`.

```
void AmbaLink_ResumeDoneCallBackImpl(UINT32 SuspendMode)
{
    int BootType;

    BootType = AmbaLink_BootType(3000);

    AmbaPrint("Linux is %s and IPC is ready!",
        (BootType == AMBALINK_COLD_BOOT) ? "ColdBoot" :
        (BootType == AMBALINK_WARM_BOOT) ? "WarmBoot" : "Hibernation Boot");

    if (BootType == AMBALINK_COLD_BOOT) {
        return;
    }
```

```
        switch (SuspendMode) {
        case AMBA_LINK_HIBER_TO_DISK:
            AmbaPrint("Linux resume AMBA_LINK_HIBER_TO_DISK is done");
            break;
        case AMBA_LINK_HIBER_TO_RAM:
            AmbaPrint("Linux resume AMBA_LINK_HIBER_TO_RAM is done");
            break;
        case AMBA_LINK_STANDBY_TO_RAM:
            AmbaPrint("Linux  resume AMBA_LINK_STANDBY_TO_RAM is done");
            break;
        case AMBA_LINK_SLEEP_TO_RAM:
            AmbaPrint("Linux resume AMBA_LINK_SLEEP_TO_RAM is done");
            break;
        default:
            break;
        }

    }
```

See Also:

AmbaIPC_LinkCtrlSuspendLinuxDoneCallBack()

11.2.12 AmbaHiber_InitCallBack

API Syntax:

(*__AmbaHiber_InitCallBack__)(UINT32)

Function Description:

- This function is called after the initialization of the hibernation service of ThreadX.

Parameters:

None

Returns:

None

Example:

Program in ThreadX. For more details, please refer to `rtos/ssp/unittest/AmbaUserSysCtrl.c`.

```
{
      AmbaHiber_InitCallBack = AmbaHiber_TestInit;
}


int AmbaHiber_TestInit(void)
{
      AmbaTest_Init();
      AmbaTest_RegisterCommand("hiber", HiberTestEntry);

      return 0;
}
```

See Also:

None

11.2.13 AmbaLink_SetIrqOwner

API Syntax:

AmbaLink_SetIrqOwner (int Irq, int Update)

Function Description:

- This function is used to set the owner of a BOSS IRQ.
- The value of update parameter can be
  - 0 – Set IRQ to RTOS and do not update the VIC
  - 1 – Set IRQ to RTOS and update the VIC
  - 2 – Set IRQ to Linux and do not update the VIC
  - 3 – Set IRQ to Linux and update the VIC

Parameters:

Type	Parameter	Description
int	**Irq**	The interrupt ID
int	**Update**	Update to interrupt controller or not

*Table 11-15. Parameters for AmbaLink Service Related API **AmbaLink_SetIrqOwner()**.*

Returns:

None

Example:

None

See Also:

None

11.2.14 AmbaLink_GetIrqOwner

API Syntax:

AmbaLink_GetIrqOwner (int Irq)

Function Description:

- This function is used to get the owner of a BOSS IRQ.

Parameters:

Type	Parameter	Description
int	**Irq**	The interrupt ID

*Table 11-16. Parameters for AmbaLink Service Related API **AmbaLink_GetIrqOwner()**.*

Returns:

Return	Description
0	RTOS
1	Linux

*Table 11-17. Returns for AmbaLink Service Related API **AmbaLink_GetIrqOwner()**.*

Example:

None

See Also:

None

11.2.15 AmbaLink_ChangeBossPriority

API Syntax:

AmbaLink_ChangeBossPriority (int NewPriority)

Function Description:

- This function is used to change the priority of BOSS.

Parameters:

Type	Parameter	Description
int	**NewPriority**	The interrupt ID

*Table 11-18. Parameters for AmbaLink Service Related API **AmbaLink_ChangeBossPriority()**.*

Returns:

Return	Description
0	Success

*Table 11-19. Returns for AmbaLink Service Related API **AmbaLink_ChangeBossPriority()**.*

Example:

None

See Also:

None

11.2.16 AmbaLink_AdjustBossSchedulePeriod

API Syntax:

AmbaLink_AdjustBossSchedulePeriod (UINT32 Period)

Function Description:

- This function is used to adjust the BOSS schedule period.

Parameters:

Type	Parameter	Description
UINT32	**Period**	New schedule period

*Table 11-20. Parameters for AmbaLink Service Related API **AmbaLink_AdjustBossSchedulePeriod()**.*

Returns:

None

Example:

None

See Also:

None

11.2.17 AmbaLink_ForceScheduleEnable

API Syntax:

AmbaLink_ForceScheduleEnable (int Enable)

Function Description:

- This function is used to enable or disable force scheduling BOSS task feature. Note that the tasks that have lower priority than BOSS task will be blocked.

Parameters:

Type	Parameter	Description
int	**Enable**	Enable this feature or not

*Table 11-21. Parameters for AmbaLink Service Related API **AmbaLink_ForceScheduleEnable()**.*

Returns:

None

Example:

None

See Also:

None

Appendix 1 Additional Resources

The most important references for this document are listed below.

- AMBARELLA_SDK6_AN_AmbaLink
- AMBARELLA_SDK6_AN_Build_Environment
- AMBARELLA_SDK6_AN_AmbaLink_Migration

For other documents of potential interest, please contact an Ambarella representative.

Appendix 2 Important Notice

All Ambarella design specifications, datasheets, drawings, files, and other documents (together and separately, “materials”) are provided on an “*as is*” basis, and Ambarella makes no warranties, expressed, implied, statutory, or otherwise with respect to the materials, and expressly disclaims all implied warranties of noninfringement, merchantability, and fitness for a particular purpose. The information contained herein is believed to be accurate and reliable. However, Ambarella assumes no responsibility for the consequences of use of such information.

Ambarella Incorporated reserves the right to correct, modify, enhance, improve, and otherwise change its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

All products are sold subject to Ambarella’s terms and conditions of sale supplied at the time of order acknowledgment. Ambarella warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with its standard warranty. Testing and other quality control techniques are used to the extent Ambarella deems necessary to support this warranty.

Ambarella assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using Ambarella components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

Ambarella does not warrant or represent that any license, either expressed or implied, is granted under any Ambarella patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which Ambarella products or services are used. Information published by Ambarella regarding third-party products or services does not constitute a license from Ambarella to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from Ambarella under the patents or other intellectual property of Ambarella.

Reproduction of information from Ambarella documents is not permissible without prior approval from Ambarella.

Ambarella products are not authorized for use in safety-critical applications (such as life support) where a failure of the product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Customers acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of Ambarella products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by Ambarella. Further, Customers must fully indemnify Ambarella and its representatives against any damages arising out of the use of Ambarella products in such safety-critical applications.

Ambarella products are neither designed nor intended for use in automotive and military/aerospace applications or environments. Customers acknowledge and agree that any such use of Ambarella products is solely at the Customer’s risk, and they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

Appendix 3 Revision History

NOTE: Page numbers for previous drafts may differ from page numbers in the current version.

Version	Date	Comments
0.1	6 November 2013	Formatting
0.5	7 November 2013	Refine all chapter descriptions and formatting
0.6	03 December 2013	Update the examples in Chapter 5 RPC (ThreadX), Chapter 9 RPC (Linux).
0.7	23 December 2013	Update in RPC (ThreadX), RPMsg (Linux) and RPC (Linux); Add chapter LKVFS (ThreadX).
0.8	24 December 2013	Update descriptions and formatting
0.9	24 January 2014	Update in RPC (ThreadX) and RPC (Linux).
1.0	14 February 2014	Update examples in Remote Processor Messaging (ThreadX), RPC (ThreadX) and Remote Procedure Calls (Linux).
1.1	8 May 2014	Add Chapter 11, AmbaLink Service related APId (Thread X).
1.2	13 May 2014	Update in Section 5.2.1, add Section 6.2.10 - 6.2.23.
	14 May 2014	Update in syntax of Chapter 11, AmbaLink Service Related APIs.
1.3	12 September 2014	Formatted into SDK6. Chapter 3: Spin Lock and Chapter 4: Mutex are moved to be before Chapter 2: Remote Processor Messaging. Chapter 8: Spin Lock and Chapter 9: Mutex are moved to be before Chapter 7: Remote Processor Messaging. Update in Chapter 1: Overview, Chapter 2: Spin Lock (ThreadX), Section 2.1, Section 2.2.1 - 2.2.4. Update in Section 3.2.1 - 3.2.2. Update in Section 5.2.5. Update in Overview of Section 6.1, Section 6.2.1 - 6.2.23. Update in Section 7.2.1 - 7.2.4. Update in Section 10.1. and 10.2.5. Update in Section 11.2.4.
1.4	12 March 2015	Update in Sections 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.5, 5.2.1, 5.2.2, 5.2.3, 5.2.4, 5.2.5, 6.2.1, 6.2.2, 6.2.3, 6.2.4, 6.2.5, 10.2.3, 10.2.5, 11.2.6, 11.2.8, 11.2.9, Added Sections 5.2.6 **AmbaIPC_RpcStrError**, 10.2.6 **AmbaIPC_StrError**, 11.2.1 **AmbaLink_Init**, 11.2.2 **AmbaLink_Load**, 11.2.3 **AmbaLink_Boot**, 11.2.10 **AmbaLink_Boot AmbaLink_SetIrqOwner**, 11.2.11 **AmbaLink_GetIrqOwner**, 11.2.12 **AmbaLink_ChangeBossPriority**,11.2.13 **AmbaLink_AdjustBossSchedulePeriod**, 11.2.14 **AmbaLink_ForceScheduleEnable**.
1.5	10 September 2015	Update in Sections 1.1.1 Introduction: AmbaLink SDK and 1.1.1.1 AmbaLink SDK: Dual-Core AmbaLink Architecture, 6.1, 6.2, 6.2.17, 6.2.18, 6.2.20 and 6.2.21.
1.6	10 December 2015	Update in Sections 1.1 Overview: Introduction, 1.1.1.1 AmbaLink Architecture: Dual-Core AmbaLink Architecture, 1.1.1.3 AmbaLink SDK: SMP BOSS Architecture and Appendix 1 Additional Resources.

Table A3-1. Revision History.