

SDK6 AN: AmbaLink

Version 1.4

February 17, 2016



Confidentiality Notice:

Copyright © 2016 Ambarella, Inc.

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: <http://www.ambarella.com/>.

Confidential
For HAOTEK Only

I Contents

II	Preface	ii
1	Overview	1
1.1	Overview: Introduction	1
2	AmbaLink Customization	2
2.1	AmbaLink Customization: Overview	2
2.2	AmbaLink Customization: AmbaLink Memory Size Adjustment	2
2.3	AmbaLink Customization: Hardware Resources Management	8
2.4	AmbaLink Customization: USB Host Switch	9
2.5	AmbaLink Customization: Change Linux Root File-System Type	10
2.6	AmbaLink Customization: UART Console Customization	12
3	AmbaLink Development	14
3.1	AmbaLink Development: Overview	14
3.2	AmbaLink Development: Functionalities of AmbaLink	14
3.3	AmbaLink Development: Global Driver on BOSS	33
3.4	AmbaLink Development: AmbaLink RPC Program Development	34
4	AmbaLink Debugging	40
4.1	AmbaLink Debugging: Overview	40
4.2	AmbaLink Debugging: Hibernation Corruption	40
4.3	AmbaLink Debugging: FAQ for RPMsg and RPC	41
	Appendix 1 Additional Resources	A1
	Appendix 2 Important Notice	A2
	Appendix 3 Revision History	A3

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.

Abbreviations:

Acronym	Definition
CRC	Cyclic Redundancy Check (Error detecting code)
RPC	Remote Procedure Call
RPMsg	Remote Processor Messaging
VFSS	Virtual File System Support

1 Overview

1.1 Overview: Introduction

This application note (AN) describes the process of customization, development and debugging on AmbaLink. The purpose of this document is to provide developers with the information required to enable network support in Ambarella SDK6 systems.

The document provides details on AmbaLink. It includes the following chapters:

- [Chapter 2 “AmbaLink Customization”](#)
- [Chapter 3 “AmbaLink Development”](#)
- [Chapter 4 “AmbaLink Debugging”](#)

This document applies to multiple Ambarella product lines, 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.

Confidential
For HAOTEK Only

2 AmbaLink Customization

2.1 AmbaLink Customization: Overview

This chapter provides the following sections:

- [\(Section 2.2\) AmbaLink Customization: AmbaLink Memory Size Adjustment](#)
- [\(Section 2.3\) AmbaLink Customization: Hardware Resources Management](#)
- [\(Section 2.4\) AmbaLink Customization: USB Host Switch](#)
- [\(Section 2.5\) AmbaLink Customization: Change Linux Root File-System Type](#)
- [\(Section 2.6\) AmbaLink Customization: UART Console Customization](#)

2.2 AmbaLink Customization: AmbaLink Memory Size Adjustment

AmbaLink memory includes AmbaLink shared memory and Linux memory. AmbaLink shared memory is used to store the data structure and information related to dual-OSes such as RPMsg buffer, global mutex, hibernation, and so on. Ambarella places Linux memory behind the AmbaLink shared memory. Customers can adjust the AmbaLink memory based on their demands.

In this section, the customization of the AmbaLink memory is introduced and is divided into the following sections:

- [\(Section 2.2.1\) AmbaLink Memory Size Adjustment: AmbaLink Memory Layout Change](#)
- [\(Section 2.2.2\) AmbaLink Memory Size Adjustment: RPMsg/RPC/Vring Memory Size Adjustment](#)
- [\(Section 2.2.3\) AmbaLink Memory Size Adjustment: Linux Kernel/User Memory Size Adjustment](#)

2.2.1 AmbaLink Memory Size Adjustment: AmbaLink Memory Layout Change

In the RTOS view, Linux memory region is located at the end of the system memory map. AmbaLink memory layout is shown in [Figure 2-1](#). Customers can have some adjustments in AmbaLink memory. The related configurations are listed in [Table 2-1](#) and [Table 2-2](#). Ambarella provides two possible cases for reference.

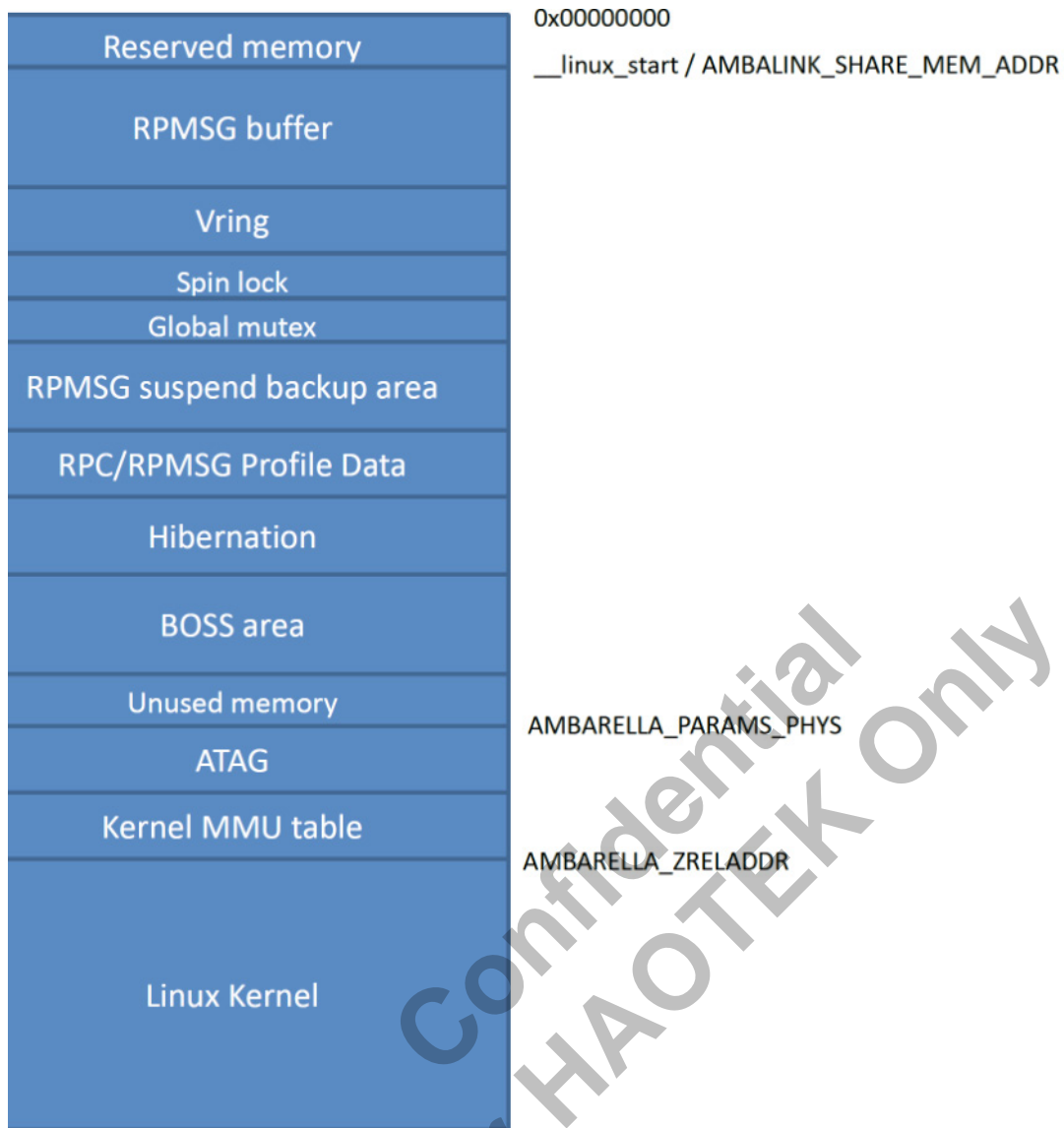


Figure 2-1. AmbaLink Memory Layout.

Related to the memory in RTOS.

RTOS Configuration	Value	Meaning
CONFIG_AMBALINK_MEM_SIZE	0x04000000	Total memory size used by Linux
CONFIG_AMBALINK_SHARED_MEM_SIZE	0x00600000	Shared memory size used by Linux and RTOS. This is included in the total Linux memory size.
CONFIG_SYS_LOADADDR	0xA0001000	RTOS load address. If Linux is put at the start of the DRAM and changes the memory size of Linux, the load address of RTOS should be changed.
CONFIG_LNX_LOADADDR	0x3C608000	Linux kernel load address. If Linux is put at the end of the DRAM and changes the memory size of Linux, the load address of Linux should be changed.

Table 2-1. The Related Configuration.

Linux Configuration	Value	Meaning	Calculation
CONFIG_AMBARELLA_PPM_SIZE	0x3C600000	Private physical memory size	RTOS_Linux_Memory_Region_Start + CONFIG_AMBALINK_SHARED_MEM_SIZE
CONFIG_AMBARELLA_TEXTOFS	0x00008000	The byte offset of the kernel image in RAM from the start of the Linux Kernel RAM	
CONFIG_AMBALINK_SHMADDR	0xDC000000	Virtual shared memory address which is common in RTOS and Linux.	RTOS_Virtual_RAM_Start + RAM_Size – Linux_Memory_Region_Size
CONFIG_AMBARELLA_ZRELADDR	0x3C608000	Physical address of decompressor code (of zImage). Not used in AmbaLink Linux.	
CONFIG_AMBARELLA_PARAMS_PHYS	0x3C600000	A tag address on which tagged parameters which have been placed by a boot loader. Not used in AmbaLink Linux.	
CONFIG_AMBARELLA_INITRD_PHYS	0x3C600000	Physical address to place the RAM disk. Not used in AmbaLink Linux.	

Table 2-2. The Configuration Related to the Memory in Linux.

Case 1: Increase in AmbaLink Memory Size

In this case, the size of AmbaLink memory is changed to 0x06000000. Therefore, AmbaLink_SHARE_MEM_ADDR is moved forward and so is AMBARELLA_PARAMS_PHYS. The corresponding changes can be found in [Table 2-3](#) and [Table 2-4](#).

RTOS Configuration	Value	Calculation
CONFIG_AMBALINK_MEM_SIZE	0x06000000	The size is increased to 0x06000000.
CONFIG_AMBALINK_SHARED_MEM_SIZE	0x00600000	Customers can also increase it. For simplifying this case, this size does not change.
CONFIG_SYS_LOADADDR	0xA0001000	No change
CONFIG_LNX_LOADADDR	0x3A608000	The loading address of Linux is also moved forward. $0x3C608000 - 0x02000000 = 0x3A608000$

Table 2-3. The Changes in RTOS when AmbaLink Memory is Moved Forward.

Linux Configuration	Value	Calculation
CONFIG_AMBARELLA_PPM_SIZE	0x3A600000	RTOS_Linux_Memory_Region_Start + CONFIG_AMBALINK_SHARED_MEM_SIZE 0x3A000000 + 0x00600000 = 0x3A600000
CONFIG_AMBARELLA_TEXTOFS	0x00008000	No change
CONFIG_AMBALINK_SHMADDR	0xDA000000	AmbaLink memory is move forward. 0XDC000000 – 0x02000000 = 0xDA000000
CONFIG_AMBARELLA_ZRE-LADDR	0x3A608000	It is also moved forward.
CONFIG_AMBARELLA_PARAMS_PHYS	0x3A600000	It is also moved forward.
CONFIG_AMBARELLA_INITRD_PHYS	0x3A600000	It is also moved forward.

Table 2-4. The Change in Linux Side when AmbaLink Memory is Moved Forward.

Case 2: No change in AmbaLink memory size, but decrease Linux memory to increase AmbaLink shared memory.

Most of the parts of AmbaLink shared memory are fixed except for the RPMsg buffer. Therefore, the only way to adjust the AmbaLink shared memory is to change the settings of the RPMsg buffer. There are two parameters: the entries of the RPMsg buffer and one RPMsg buffer size. The default settings are 2048 and 2048, respectively. Note that, when either one of these two settings are changed, the size of AmbaLink shared memory size which is given by customer is also changed. If the given size is too small, there will be errors during the compile time. Customers can query Ambarella about the accurate size. Furthermore, the loader address of the Linux Kernel may be affected when the AmbaLink shared memory is changed.

For example, if the customers do not change the AmbaLink memory size, but they want to increase the RPMsg buffer size to increase the AmbaLink shared memory size. As a result, they decide to decrease the Linux Memory to extend the AmbaLink shared memory. The corresponding changes are listed in [Table 2-5](#) and [Table 2-6](#). The details about changing the RPMsg settings are described in [Section 2.2.2](#).

RTOS Configuration	Value	Calculation
CONFIG_AMBALINK_MEM_SIZE	0x04000000	No change
CONFIG_AMBALINK_SHARED_MEM_SIZE	0x00A00000	When RPMsg buffer entry increases to 4096, the total AmbaLink shared memory size should be more than or equal to 0X00A00000.
CONFIG_SYS_LOADADDR	0xA0001000	No Change
CONFIG_LNX_LOADADDR	0x3CA08000	0x3C608000 + 0x00400000 (The increased AmbaLink shared memory) = 0x3CA08000

Table 2-5. The Changes in RTOS Side when AmbaLink Shared Memory Size is Enlarged.

Linux Configuration	Value	Calculation
CONFIG_AMBARELLA_PPM_SIZE	0x3CA00000	$0x3C600000 + 0x00400000$ (The increased AmbaLink shared memory) = 0x3CA00000
CONFIG_AMBARELLA_TEXTOFS	0x00008000	No change
CONFIG_AMBALINK_SHMADDR	0xDC000000	No change
CONFIG_AMBARELLA_ZRE-LADDR	0x3CA08000	$0x3C608000 + 0x00400000$ (The increased AmbaLink shared memory) = 0x3CA08000
CONFIG_AMBARELLA_PARAMS_PHYS	0x3CA00000	$0x3C600000 + 0x00400000$ (The increased AmbaLink shared memory) = 0x3CA00000
CONFIG_AMBARELLA_INITRD_PHYS	0x3CA00000	$0x3C600000 + 0x00400000$ (The increased AmbaLink shared memory) = 0x3CA00000

Table 2-6. The Changes in the Linux Side when AmbaLink Shared Memory Size is Enlarged.

2.2.2 AmbaLink Memory Size Adjustment: RPMsg/RPC/Vring Memory Size Adjustment

The change in RPMMSG size can also affect the size of RPC and Vring. This section describes the RPMsg, RPC and Vring memory size adjustment.

2.2.2.1 RPMsg/RPC/Vring Memory Size Adjustment: RPMMSG Memory Size Adjustment

Ambarella provides RPMsg adjustment for the cases listed below:

Case 1: Too many RPMsg are transferred concurrently so buffer space may be unavailable for other purposes.

Case 2: Carry more traffic for one RPMsg

Case 3: RPMsg usage is not much and customers want to decrease AmbaLink Memory.

However, the changes may also change the location of Linux Memory. Therefore, the Linux loading address should be changed as well. The corresponding settings can refer to [Section 2.2.1](#). The steps for RPMsg adjustment are listed below:

In RTOS side:

Step 1:

make menuconfig

Step 2:

>AmbaLink

>Ambalink Configuration

(2048) Size of an RPMsg

(2048) Number of RPMsg buffer

Customers can input the desired number and size for RPMsg.

Step 3: Check the related setting in [Section 2.2.1](#)

In the Linux side:

Step 1: Change the setting in ambalink_sdk/linux/arch/arm/mach-ambarella/include/plat/ambalink_cfg.h,

```
#define RPMMSG_NUM_BUFS      (2048)
```

```
#define RPMMSG_BUF_SIZE      (2048)
```

Step 2: Check the related settings in [Section 2.2.1](#) for the .config located in output.oem/ambalink/build/linux-custom/.

Step 3: Rebuild Linux Kernel
make linux-rebuild

2.2.2.2 RPMsg/RPC/Vring Memory Size Adjustment: RPC Size Adjustment

When RPMsg increases or decreases, the setting of the RPC packet payload size should also change. The RTOS size can be auto-adjusted when RPMsg changes. However, this auto adjustment needs to be manually setup in Linux.

In the Linux side:

Step 1: Input RPMsg size information in ambalink_sdk/pkg/ipcdef/aipc_cfg.c.

```
#define RPMSG_BUF_SIZE 2048
```

Step 2: Rebuild the library libaipc_cfg.so to provide the information of the RPC packet payload to other user space programs.
make ipcdef-rebuild

Note that if the customers want to use the value of the RPC payload in their RPC program, one can use the external variable "rpc_size" and link to libaipc_cfg.so.

2.2.2.3 RPMSG Memory Size Adjustment: Vring Size Adjustment

Vring is used to manage RPMSG buffer, so the total size varies when the number of RPMSG buffer entries varies. Customers do not take care of Vring size calculation since Ambarella provides the auto-calculation in SDK. If customers want to have a better understanding of Vring size, the details are described in the following paragraph.

RPMSG buffer is divided into sending and receiving buffer so there are two Vrings to manage them. Vring includes many structures such as struct vring_desc, vring_avail and so on.¹ The total size of Vring needs to sum up all the structures. However, the original equation is slightly complicated because each structure has different size. For easy calculation, Ambarella divides these structures into two groups and treats all the structures equally in the same group. Therefore, the equation can be simplified and easy to calculate in Ambarella SDK. The calculation is shown in the following files:

RTOS side: rtos/bsp/(target_platform)/AmbaLinkMemoryCfg.c

Linux side: ambalink_sdk/linux/arch/arm/mach-ambarella/include/plat/ambalink_cfg.h

2.2.3 AmbaLink Memory Size Adjustment: Linux Kernel/User Memory Size Adjustment

Take a12_dragonfly as an example. 2G/2G user/kernel split is chosen.

In Linux side:

1. Change the directory to build the Linux folder (ex: output.oem/a12_ambalink)
2. Type "make linux-menuconfig" to show the menuconfig window

¹ The related structures can be found in Linux source code and they are located in ambalink_sdk/linux/include/uapi/linux/virtio_ring.h.

3. Enable 2G/2G user/kernel split. Please find it as follows:

```
>Kernel Features
>Memory split
(X) 2G/2G user/kernel split
```

4. Type “make linux-rebuild && make” to rebuild Linux
5. Copy linux image to rtos/linux_image

In RTOS side:

1. Find AmbaLinkMemoryCfg.c in rtos/bsp/dragonfly
2. In AmbaLink_LinkCtrlCfg, modify “AmbaLinkCtrl.AmbaLinkKernelSpaceAddr=0x80000000”.
3. Type “make” to rebuild RTOS

2.3 AmbaLink Customization: Hardware Resources Management

- (Section 2.3.1) [Hardware Resources Management: Overview](#)

2.3.1 Hardware Resources Management: Overview

Hardware resource settings may be diverse for different chips. The customer can find the difference by looking up the configuration file. For example, in the Linux configuration, ambarella_a9_ambalink_defconfig and ambarella_a12_ambaboss_defconfig, the following hardware resources listed in [Table 2-7](#) and [Table 2-8](#) are used. Customers need to take care of these resources while developing the RTOS side.

Linux Only	Shared	Optional
AMBA_TIMER6	GIC	AMBA_TIMER5 (RPMsg/RPC Profiling)
AMBA_TIMER7	GPIO	Ethernet
SDIO	NAND	I2C
UART1 or UART2	PLL	USB

Table 2-7. ambarella_a9_ambalink_defconfig.

Linux Only	Shared	Optional
AMBA_TIMER5	VIC	AMBA_TIMER4 (RPMsg/RPC Profiling)
AMBA_TIMER6	GPIO	Ethernet
SDIO	NAND	I2C
UART1 or UART2	PLL	USB

Table 2-8. ambarella_a12_ambaboss_defconfig.

The “Linux Only” column:

- This means that the hardware resources are used in Linux only while in the AmbaLink environment. The Hardware resources are used to keep Linux running and maintain the WiFi/BT connection.
- RTOS development process does not need the hardware resources.

The “Shared” column:

- It means that the hardware resources are used by Linux and ThreadX.
- Shared hardware resources are handled by Ambarella.

The “Optional” column:

- It means that the hardware resources may not be in the final product and thus, they can be turned off.
- If they are enabled, RTOS development process needs to take care of it depending on whether it is a shared resource or a Linux Only resource.

Note that the Linux console is UART1 by default. If the users would like to change the console to UART2, please modify the “KernelCmdLine” in AmbaLinkMemCfg.c and enable UART2 in the Linux menuconfig.

2.4 AmbaLink Customization: USB Host Switch

The instructions below illustrate the customization of the AmbaLink.

- (Section 2.4.1) USB Host Switch: [USB Host Switch to Linux](#)
- (Section 2.4.2) USB Host Switch: [USB Host Switch to RTOS](#)

2.4.1 USB Host Switch: USB Host Switch to Linux

1. Preload

Ambarella releases the USB WIFI configuration to the users and this configuration will preload the USB modules for speeding up the resumption of hibernation.

If the the user does not want to load the USB modules during first boot, please use the following steps to uninstall the script.

For example, if the user does not want to load the USB WIFI driver RTL8188ETV, then use the following steps:

- Change to the user’s build folder
- Make **rtl8188_rtl8189-uninstall**
- Type **“make menuconfig”**
- De-select the USB wifi driver in Ambarella external packages -> Realtek SDK -> RTL8188ETV
- **“make”**

2. USB allocation

- Use the variable “AmbaLinkCtrl.AmbaLinkUsbOwner” to determine the USB owner
- Use the command **“t ipc usb_owner 1”** to set the variable to 1 and give the owner to linux

3. USB PHY Owner switch (ThreadX/SMP to Linux)
 - Use `enable_irq(udc->irq)` to switch the IRQ to Linux. Note that `disable_irq(udc->irq3)` should be called first.

2.4.2 USB Host Switch: USB Host Switch to RTOS

1. USB allocation
 - Use this command `"t ipc usb_owner 0"` and give the owner to RTOS.
2. USB PHY and IRQ Owner switch
 - The ISR owner of USB vbus detection should be set in RTOS side. Please use **AmbaUSB_System_SetUsbOwner** API. Then, call **AmbaUSBH_System_SetPhy0Owner** to set the USB PHY. The details can be found in *AMBARELLA SDK6 API USB* and *AMBARELLA SDK6 AN USB* documents.

2.5 AmbaLink Customization: Change Linux Root File-System Type

This section introduces the changes to the Linux Root File-System type due to AmbaLink customization. It includes the following sections:

- (Section 2.5.1) [Change Linux Root File-System Type: Change to UBIFS](#)
- (Section 2.5.2) [Change Linux Root File-System Type: Change to SquashFS](#)

2.5.1 Change Linux Root File-System Type: Change to UBIFS

Linux side:

1. Change the directory to build the Linux folder (ex: `output.oem/a9s_ambalink`)
2. Type `"make linux-menuconfig"` to show the menuconfig window
3. Enable caching block device access to the MTD devices. Please find it as follows:

```
Check
> Device Drivers
    >Memory Technology Device (MTD) support
        <*> Caching block device access to MTD device
```

RTOS side:

1. Copy `Image` and `rootfs.ubi` from the Linux image generated folder (ex: `output.oem/a9s_ambalink/images`) to the Linux image folder in the RTOS side (ex: `rtos/linux_image/boss`)
2. Change the directory to the RTOS build folder and type `"make menuconfig"` to modify the rootfs settings as follows:
 - > Ambalink Configuration
 - > Command line of AmbaLink
 - > `root=/dev/mtdblock5 rootfstype=squashfs console=ttyS1 nr_cpus=1 maxcpus=0`
 - ↓
 - `"ubi.mtd=5 root=ubi0:linux rootfstype=ubifs console=ttyS1 nr_cpu=1 maxcpus=0"`

```

>Firmware Programmer
  > Linux Root FS Partition
    > "${srctree}/linux_image/boss/rootfs.squashfs"
      ↓
    "${srctree}/linux_image/boss/rootfs.ubi"

```

2.5.2 Change Linux Root File-System Type: Change to SquashFS

Linux side:

1. Change the directory to build the Linux folder (ex: output.oem/a9s_ambalink)
2. Type "make linux-menuconfig" to show the menuconfig window
3. Disable the caching block device access to the MTD device and enable Ambarella Read only block device access to MTD devices. Please find them as follows:

```

Check
> Device Drivers
  >Memory Technology Device (MTD) support
    <> Caching block device access to MTD device
    <*> Ambarella Read only block device access to MTD devices

```

4. Double check the following options:

```

> File System
  >Miscellaneous filesystems
    <*> SquashFS 4.0 – Squashed file system support
      ☐ Squashfs XATTR support
      ☐ Include support for ZLIB compressed file systems
      [*] Include support for LZO compressed file systems
      ☐ Include support for XZ compressed file systems
      [*]Use 4K device block size?
      [*]Additional option for memory-constrained systems
      (3)Number of fragments cached

```

RTOS side:

1. Copy Image and rootfs.squashfs from the Linux image generated folder (ex: output.oem/a9s_ambalink/images) to the Linux image folder in the RTOS side (ex: rtos/linux_image/boss)
2. Change the directory to the RTOS build folder and type "make menuconfig" to modify the rootfs settings as follows:

```

> Ambalink Configuration
  > Command line of AmbaLink
    > "ubi.mtd=5 root=ubi0:linux rootfstype=ubifs console=ttyS1 nr_cpu=1 maxcpus=0"
      ↓
    "root=/dev/mtdblock5 rootfstype=squashfs console=ttyS1 nr_cpus=1 maxcpus=0"

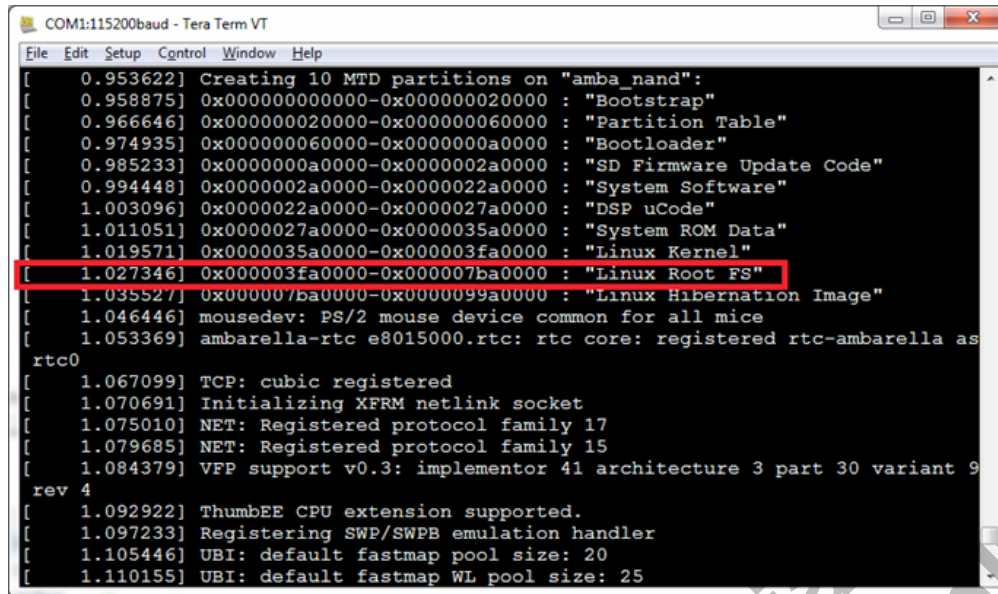
```

```

>Firmware Programmer
  > Linux Root FS Partition
    > "${srctree}/linux_image/boss/rootfs.ubifs"
      ↓
    "${srctree}/linux_image/boss/rootfs.squashfs"

```


Note that rootfs is not always in mtdblock5. Please check it in the Linux log first. For example, the Linux log for A12 is shown in Figure 2-2. LINUX_RFS is located in the ninth mtdblock, so root=/dev/mtdblock8.



```
COM1:115200baud - Tera Term VT
File Edit Setup Control Window Help
[ 0.953622] Creating 10 MTD partitions on "amba_nand":
[ 0.958875] 0x000000000000-0x000000020000 : "Bootstrap"
[ 0.966646] 0x000000020000-0x000000060000 : "Partition Table"
[ 0.974935] 0x000000060000-0x0000000a0000 : "Bootloader"
[ 0.985233] 0x0000000a0000-0x0000002a0000 : "SD Firmware Update Code"
[ 0.994448] 0x0000002a0000-0x00000022a0000 : "System Software"
[ 1.003096] 0x00000022a0000-0x00000027a0000 : "DSP uCode"
[ 1.011051] 0x00000027a0000-0x00000035a0000 : "System ROM Data"
[ 1.019571] 0x00000035a0000-0x0000003fa0000 : "Linux Kernel"
[ 1.027346] 0x0000003fa0000-0x000007ba0000 : "Linux Root FS"
[ 1.035527] 0x000007ba0000-0x0000099a0000 : "Linux Hibernation Image"
[ 1.046446] mousedev: PS/2 mouse device common for all mice
[ 1.053369] ambarella-rtc e8015000.rtc: rtc core: registered rtc-ambarella as
rtc0
[ 1.067099] TCP: cubic registered
[ 1.070691] Initializing XFRM netlink socket
[ 1.075010] NET: Registered protocol family 17
[ 1.079685] NET: Registered protocol family 15
[ 1.084379] VFP support v0.3: implementor 41 architecture 3 part 30 variant 9
rev 4
[ 1.092922] ThumbEE CPU extension supported.
[ 1.097233] Registering SWP/SWPB emulation handler
[ 1.105446] UBI: default fastmap pool size: 20
[ 1.110155] UBI: default fastmap WL pool size: 25
```

Figure 2-2. Linux Log for A12.

2.6 AmbaLink Customization: UART Console Customization

In some devices, there is only one physical UART port and it is used by RTOS in default. To change the UART for Linux, please follow the steps listed below:

In the Linux side:

1. Modify uart0 settings in the dbt files. The dbt files are located in ambarella/bootloader/boards. Take A12 as an example, the file location is ambarella/bootloader/boards/a12evk/bsp/a12evk_nand.dts. The updated content is as follows:

```
Uart0: uart@e8005000 {
    status = "disabled"
};
↓
Uart0: uart@e8005000 {
    status = "ok"
};
```

2. Follow the instructions in the document *Ambarella SDK6 AN Build Environment (Ubuntu 14.04)* to rebuild the dtb.

In the RTOS side:

1. Update the dbt_debug.bin.
2. Change the directory to the RTOS build folder and type “make menuconfig” to modify the rootfs settings as follows:
 - > Ambalink Configuration
 - > Command line of AmbaLink
 - > “ubi.mtd=5 root=ubi0:linux rootfstype=ubifs console=ttyS0 nr_cpu=1 maxcpus=0”

Note:

1. After the Linux is booted up, ttyS0 is taken by Linux and IRQs are directed to Linux. Therefore, the users cannot type at AmbaShell in RTOS.
2. No one handles UART if the Linux shell is disabled. Please enable the Linux shell to enable the configuration BR2_AMBARELLA_LINUX_RESPAWN_SH=y.

Confidential
For HAOTEK Only

3 AmbaLink Development

3.1 AmbaLink Development: Overview

This chapter guides the customers how to develop their modules on AmbaLink. First, the functionalities of AmbaLink are introduced. Then, the development of the RPC program is described for communication between the dual operating systems via AmbaLink. The following sections are listed as follows:

- (Section 3.2) AmbaLink Development: Functionalities of AmbaLink
- (Section 3.3) AmbaLink Development: Global Driver on BOSS
- (Section 3.4) AmbaLink Development: AmbaLink RPC Program Development

3.2 AmbaLink Development: Functionalities of AmbaLink

This section describes the functionalities of AmbaLink. Furthermore, the commands used for the verification of AmbaLink functionality are also introduced. The section is organized as follows:

- (Section 3.2.1) Functionalities of AmbaLink: RPMsg (Remote Processor Messaging)
- (Section 3.2.2) Functionalities of AmbaLink: RPC (Remote Procedure Call)
- (Section 3.2.3) Functionalities of AmbaLink: RPC Utility-Execution
- (Section 3.2.4) Functionalities of AmbaLink: VFSS (Virtual File System Support) Test for AmbaFS
- (Section 3.2.5) Functionalities of AmbaLink: Suspend Mode
- (Section 3.2.6) Functionalities of AmbaLink: Wi-Fi Support
- (Section 3.2.7) Functionalities of AmbaLink: USB Wi-Fi
- (Section 3.2.8) Functionalities of AmbaLink: Ethernet Support
- (Section 3.2.9) Functionalities of AmbaLink: IPC Spin Lock (GPIO)
- (Section 3.2.10) Functionalities of AmbaLink: IPC Mutex (I2C)
- (Section 3.2.11) Functionalities of AmbaLink: IPC Mutex (NAND)

3.2.1 Functionalities of AmbaLink: RPMsg (Remote Processor Messaging)

This section describes the procedure for testing the Remote Processor Messaging (RPMsg) functionality. The section is organized as follows:

- (Section 3.2.1.1) RPMsg: Requirements
- (Section 3.2.1.2) RPMsg: Test Purpose
- (Section 3.2.1.3) RPMsg: Test Flow and Results

3.2.1.1 RPSMsg: Requirements

In order for the RPSMsg module to function properly, the following system requirements must be met:

- Firmware that includes AmbaLink functionality must have been correctly enabled.
- The system must boot successfully (both the ThreadX and Linux consoles must be enabled).

3.2.1.2 RPSMsg: Test Purpose

The purpose of this test is to verify RPSMsg communications between the Linux Kernel and the ThreadX RTOS. The test is structured according to a **string echo** call-and-response concept. During this test, the ThreadX RTOS transmits a string to Linux, and Linux is required to echo the identical string back to ThreadX.

3.2.1.3 RPSMsg: Test Flow and Results

- Reference Code:
 - ThreadX: `rtos/ssp/unittest/AmbaIPC_Test.c`
 - Linux: `drivers/rpsmsg/ambarella/host/echo/rpsmsg_echo.c`

- Test Flow:
 - In the ThreadX console, run the RPSMsg test tool:

```
a:\> t ipc rpsmsg test
```

- Test Results:

- Linux:

```
[ 354.782299] [ rpsmsg_echo_cb ] recv msg: [Hello Linux! 0] from 0x401 and len 16
[ 354.790454] [ rpsmsg_echo_cb ] recv msg: [Hello Linux! 1] from 0x401 and len 16
[ 354.798413] [ rpsmsg_echo_cb ] recv msg: [Hello Linux! 2] from 0x401 and len 16
[ 354.806364] [ rpsmsg_echo_cb ] recv msg: [Hello Linux! 3] from 0x401 and len 16
[ 354.814312] [ rpsmsg_echo_cb ] recv msg: [Hello Linux! 4] from 0x401 and len 16
[ 354.822266] [ rpsmsg_echo_cb ] recv msg: [Hello Linux! 5] from 0x401 and len 16
[ 354.830216] [ rpsmsg_echo_cb ] recv msg: [Hello Linux! 6] from 0x401 and len 16
[ 354.838159] [ rpsmsg_echo_cb ] recv msg: [Hello Linux! 7] from 0x401 and len 16
```

- ThreadX:

```
send message: [Hello Linux! 0]
send message: [Hello Linux! 1]
send message: [Hello Linux! 2]
send message: [Hello Linux! 3]
send message: [Hello Linux! 4]
send message: [Hello Linux! 5]
send message: [Hello Linux! 6]
send message: [Hello Linux! 7]
a:\> [00356342][CA9_0] recv message: [Hello Linux! 0]
[00356348][CA9_0] recv message: [Hello Linux! 1]
[00356356][CA9_0] recv message: [Hello Linux! 2]
[00356364][CA9_0] recv message: [Hello Linux! 3]
[00356372][CA9_0] recv message: [Hello Linux! 4]
[00356381][CA9_0] recv message: [Hello Linux! 5]
[00356388][CA9_0] recv message: [Hello Linux! 6]
[00356396][CA9_0] recv message: [Hello Linux! 7]
```

3.2.2 Functionalities of AmbaLink: RPC (Remote Procedure Call)

This section describes the procedure for testing the Remote Procedure Call (RPC) functionality. The section is organized as follows:

- [\(Section 3.2.2.1\) RPC: Requirements](#)
- [\(Section 3.2.2.2\) RPC: Test Purpose](#)
- [\(Section 3.2.2.3\) RPC: Test Flow and Results](#)

3.2.2.1 RPC: Requirements

In order for the RPC module to function properly, the following system requirements must be met:

- Firmware that includes AmbaLink functionality must have been correctly enabled.
- The system must boot successfully (both the ThreadX and Linux consoles must be enabled).

3.2.2.2 RPC: Test Purpose

The purpose of this test is to verify that bi-directional RPC communications exist between the Linux user-space and the ThreadX RTOS.

3.2.2.3 RPC: Test Flow and Results

3.2.2.3.1 Test Flow and Results: Testing RPC Communications from the Linux User-Space to ThreadX

- Reference Code:

- ThreadX: `rtos/ssp/unittest/AmbaIPC_Test.c`
- Linux: `AmbaLink_sdk/pkg/ambaipc_test/clnt_test.c`

- Test Flow:

- At the ThreadX console, enable the RPC server. The command below registers a test program to the server.

```
a:\> t ipc rpc svc test +
```

- At the Linux console, run the RPC test client. The command below runs the client with host ID 1 for the test program. Host ID 1 indicates that the host is on the ThreadX side.

```
root@a9 / $ clnt_test 1
```

- Test Results:

- Linux:

```
clnt[899679336]: status is 0, sum is 0
clnt[901776488]: status is 0, sum is 0
...
clnt[899679336]: status is 0, sum is 62
clnt[901776488]: status is 0, sum is 62
clnt[899679336]: status is 2
clnt[899679336]: status is 3
clnt[901776488]: status is 2
clnt[901776488]: status is 3
```

- ThreadX:

```
[00051997][CA9_0] [svc echo] hello server!!
[00052019][CA9_0] [svc echo] hello server!!
```

3.2.2.3.2 Test Flow and Results: Testing RPC Communications from ThreadX to the Linux User-Space

- Reference Code:

- ThreadX: `rtos/ssp/unittest/AmbaIPC_Test.c`
- Linux: `AmbaLink_sdk/pkg/ambaipc_test/svc_test.c`

- Test Flow:

1. At the Linux console, run the **RPC test server**. The following command registers a test program to the server.

```
root@a9 / $ svc_test &
```

2. At the ThreadX console, run the **RPC sum-up** test client. The following command runs **procedure 2** with the test program, and the argument is 4 for the sum-up operation.

```
a:\> t ipc rpc clnt test 2 4
```

3. At the ThreadX console, run the **RPC echo** test client. The following command runs **procedure 1** with the test program, and the argument is **hello** for the print operation.

```
a:\> t ipc rpc clnt test 1 hello
```

- Test Results:

- ThreadX (After Step 3 Above):

```
Return value is 8
```

- Linux (After Step 4 Above):

```
service echo: [hello]
```

3.2.3 Functionalities of AmbaLink: RPC Utility-Execution

This section describes the procedure for testing the RPC execution command. The section is organized as follows:

- [\(Section 3.2.3.1\) Execution: Requirements](#)
- [\(Section 3.2.3.2\) Execution: Test Purpose](#)
- [\(Section 3.2.3.3\) Execution: Test Flow and Results](#)

3.2.3.1 Execution: Requirements

In order to test the execution command properly, the following system requirements must be met:

- Firmware that includes the AmbaLink functionality must have been correctly enabled.
- The system must boot successfully (both the ThreadX and Linux consoles must be enabled).

3.2.3.2 Execution: Test Purpose

RTOS can execute the Linux command through the RPC utility tool. There are two execution commands: **exec1** and

exec2. The execution results run by **exec1** and **exec2** will be shown in RTOS and Linux, respectively.

3.2.3.3 Execution: Test Flow and Results

- Reference Code:

- ThreadX: `rtos/ssp/unittest/AmbaIPC_Test.c`
- Linux: `AmbaLink_sdk/pkg/ambaipc_util/util_svc.c`

- Test Flow:

1. At the RTOS console, the `exec1` command executes “**ls -la**” in Linux to see the file list. As a result, it will return the execution result to RTOS.

- `a:\> t ipc rpc clnt exec1 ls -la`

2. At the RTOS console, the `exec2` command executes “**ls -la**” in Linux to see the file list. The result only is only seen on the Linux side and no result is returned to RTOS.

- `a:\> t ipc rpc clnt exec2 ls -la`

Note:

- The user can replace other commands with “**ls -la**”. However, some special characters such as `|`, and `“”` cannot be recognized by AmbaShell. Please use a simple command to test the execution command.
- Currently, the message size of the execution result is limited to 2048 bytes. If the output size exceeds 2048 bytes, the output message will not be completely shown through the `exec1` command.

- Test Result:

The file list should be the same in Step 1 and Step 2.

- (Step 1) On the RTOS side

total 4

```
drwxr-xrwx 18 root root 1376 Dec 13 2013 .
drwxr-xrwx 18 root root 1376 Dec 13 2013 ..
drwxr-xrwx 2 root root 5832 Dec 13 2013 bin
drwxr-xr-x 5 root root 2920 Jan 1 00:00 dev
drwxr-xr-x 8 root root 1992 Dec 13 2013 etc
drwxr-xrwx 4 root root 288 Dec 12 2013 home
-rwxr-x--- 1 root root 242 Oct 30 2013 init

drwxr-xr-x 3 root root 2464 Dec 13 2013 lib
lrwxrwxrwx 1 root root 11 Dec 12 2013 linuxrc -> bin/busybox
drwxr-xrwx 2 root root 160 Oct 30 2013 media
drwxr-xrwx 2 root root 160 Oct 30 2013 mnt
drwxr-xrwx 2 root root 160 Oct 30 2013 opt
drwxr-xr-x 2 root root 60 Jan 1 00:00 pref
dr-xr-xr-x 42 root root 0 Jan 1 00:00 proc
```

```
drwxr-xrwx 2 root root 376 Oct 30 2013 root
lrwxrwxrwx 1 root root 3 Nov 11 2013 run -> tmp
drwxr-xr-x 2 root root 5040 Dec 13 2013 sbin
dr-xr-xr-x 12 root root 0 Jan 1 00:00 sys
drwxrwxrwt 11 root root 320 Jan 1 00:00 tmp
drwxr-xrwx 7 root root 480 Dec 12 2013 usr
drwxr-xrwx 3 root root 736 Dec 13 2013 var
```

- (Step 2) On the Linux side, total 4

```
drwxr-xrwx 18 root root 1376 Dec 13 2013 .
drwxr-xrwx 18 root root 1376 Dec 13 2013 ..
drwxr-xrwx 2 root root 5832 Dec 13 2013 bin
drwxr-xr-x 5 root root 2920 Jan 1 00:00 dev
drwxr-xr-x 8 root root 1992 Dec 13 2013 etc
drwxr-xrwx 4 root root 288 Dec 12 2013 home
-rwxr-x--- 1 root root 242 Oct 30 2013 init
drwxr-xr-x 3 root root 2464 Dec 13 2013 lib
lrwxrwxrwx 1 root root 11 Dec 12 2013 linuxrc -> bin/busybox
drwxr-xrwx 2 root root 160 Oct 30 2013 media
drwxr-xrwx 2 root root 160 Oct 30 2013 mnt
drwxr-xrwx 2 root root 160 Oct 30 2013 opt
drwxr-xr-x 2 root root 60 Jan 1 00:00 pref

dr-xr-xr-x 42 root root 0 Jan 1 00:00 proc
drwxr-xrwx 2 root root 376 Oct 30 2013 root
lrwxrwxrwx 1 root root 3 Nov 11 2013 run -> tmp
drwxr-xr- 2 root root 5040 Dec 13 2013 sbin
dr-xr-xr-x 12 root root 0 Jan 1 00:00 sys
drwxrwxrwt 11 root root 320 Jan 1 00:00 tmp
drwxr-xrwx 7 root root 480 Dec 12 2013 usr
drwxr-xrwx 3 root root 736 Dec 13 2013 var
```


3.2.4 Functionalities of AmbaLink: VFSS (Virtual File System Support) Test for AmbaFS

This section describes the procedure for testing the Virtual File System Support (VFSS) functionality. The section is organized as follows:

- (Section 3.2.4.1) VFSS: Requirements
- (Section 3.2.4.2) VFSS: Test Purpose
- (Section 3.2.4.3) VFSS: Test Flow and Results

3.2.4.1 VFSS: Requirements

In order for the VFSS module to function properly, the following requirements must be met:

- The firmware that includes the AmbaLink functionality must have been correctly enabled.
- The system must boot successfully (both the ThreadX and Linux consoles must be enabled).
- An SD card must be inserted in the SD slot on the Main Board.
- The `ls` command must be used on both ThreadX and Linux to verify the file list.

3.2.4.2 VFSS: Test Purpose

The purpose of this test is to confirm that VFS support exists by executing a virtual file system transfer operation on Linux.

3.2.4.3 VFSS: Test Flow and Results

- Test Flow:

1. On the Linux side, mount **AmbaFS** to a node:

```
root@a9 / $ mkdir /tmp/rtos
root@a9 / $ mount -t ambafs c: /tmp/rtos
root@a9 / $ cd /tmp/rtos
root@a9 /tmp/rtos $ ls -l
```

2. On the ThreadX side, use an `ls` command to view the file list.

```
c:\> cd c:
c:\> ls
```

- Test Results:
 - The file list should be the same on both the ThreadX and Linux sides.
 - ThreadX:

```
d---- Jan  1 2008 18:31:00      131072 [DCIM]
d---- Jan  1 2008 18:31:08      131072 [MISC]
f-a-- Jan  1 2008 03:27:26        3708 vin_info_parse.log
f-a-- Jan  1 2008 03:27:28        940 vin_info_dump.log
f-a-- Jan  1 2008 03:27:28      131072 drv_dump.log
f-a-- Jan  1 2008 03:27:30        116 vin_irq_1.log
f-a-- Jan  1 2008 03:27:30    3069434 dsp1.log
f-a-- Jan  2 2008 16:56:44      346401 dsp.log
f-a-- Jan  2 2008 16:56:44      131072 drv.log
```

- Linux:

```
total 0
drw-r--r--  1 root    root    131072 Jan  1 2008 DCIM
drw-r--r--  1 root    root    131072 Jan  1 2008 MISC
-rw-r--r--  1 root    root    131072 Jan  2 2008 drv.log
-rw-r--r--  1 root    root    131072 Jan  1 2008 drv_dump.log
-rw-r--r--  1 root    root    346401 Jan  2 2008 dsp.log
-rw-r--r--  1 root    root    3069434 Jan  1 2008 dsp1.log
-rw-r--r--  1 root    root        940 Jan  1 2008 vin_info_dump.log
-rw-r--r--  1 root    root    3708 Jan  1 2008 vin_info_parse.log
-rw-r--r--  1 root    root        116 Jan  1 2008 vin_irq_1.log
```

3.2.5 Functionalities of AmbaLink: Suspend Mode

This section describes the procedure for testing the hibernation functionality. The section is organized as follows:

- [\(Section 3.2.5.1\) Suspend Mode: Requirements](#)
- [\(Section 3.2.5.2\) Suspend Mode: Test Purpose](#)
- [\(Section 3.2.5.3\) Suspend Mode: Test Flow and Results](#)

3.2.5.1 Suspend Mode: Requirements

In order for the suspend mode feature to function correctly, the following system requirements must be met:

- The Firmware that includes the AmbaLink functionality must have been correctly enabled.
- The system must boot successfully (both the ThreadX and Linux consoles must be enabled).

3.2.5.2 Suspend Mode: Test Purpose

The purpose of this test is to verify the hibernation function.

3.2.5.3 Suspend Mode: Test Flow and Results

a. Hibernation

- Test Flow:

In the RTOS side, enter the command as listed in the following command:

a:> t hiber wipeout

Or, program firmware to flash for the first time, and then reboot the system.

- Test Result

Check below log to make sure hibernation is saved to NAND.

```
[00004155][CA9_0] Linux is ColdBoot and IPC is ready!  
[00011149][CA9_0] Linux suspend AMBA_LINK_HIBER_TO_DISK is done
```

b. Sleep to RAM test

- Test Flow:

In RTOS side, input the command:

a:> t hiber stress 3 10

- Test Result

The log of the last loop in RTOS.

```
[00263377][CA9_0] ===== stress loop 9 =====  
[00263377][CA9_0] AmbaIPC_LinkCtrlSuspendLinux(3)  
[00263476][CA9_0] AmbaIPC_LinkCtrlWaitSuspendLinux(3) done.  
[00263476][CA9_0] AmbaIPC_LinkCtrlResumeLinux(3)  
[00263780][CA9_0] Linux is WarmBoot and IPC is ready!  
[00263780][CA9_0] Linux resume AMBA_LINK_SLEEP_TO_RAM is done  
[00263780][CA9_0] AmbaIPC_LinkCtrlWaitResumeLinux(3) done.
```

3.2.6 Functionalities of AmbaLink: Wi-Fi Support

This section describes the procedure for testing the Wi-Fi functionality. The section is organized as follows:

- [\(Section 3.2.6.1\) Wi-Fi: Requirements](#)
- [\(Section 3.2.6.2\) Wi-Fi: Test Purpose](#)
- [\(Section 3.2.6.3\) Wi-Fi: Test Flow and Results](#)

3.2.6.1 Wi-Fi: Requirements

In order for the Wi-Fi connection to function correctly, the following requirements must be met:

- The firmware that includes the AmbaLink functionality must have been correctly enabled.
- The system must boot successfully (both the ThreadX and Linux consoles must be enabled).
- An Atheros SDIO Wi-Fi card must be inserted into the uppermost SD slot on the Main Board.
- A Windows or Linux PC must be associated with a camera SSID.
- A Windows PC with Wi-Fi capability and the **Iperf** tool installed.
 - The user can download **Iperf** from <http://iperf.fr/>.

3.2.6.2 Wi-Fi: Test Purpose

The purpose of this test is to verify the Wi-Fi connection.

3.2.6.3 Wi-Fi: Test Flow and Results

- Test Flow:
 1. On the Windows PC, use Wi-Fi to connect to the Ambarella platform:
 - The default AP name is **amba_boss**.
 2. Run the **Iperf** server on the Windows PC:
 - Open the command window.
 - Change to the **Iperf** folder and type `iperf -s`.
 3. At the Linux console, initiate transmit operations and perform a throughput test for 100 seconds:
 - Assuming that the IP address of the Windows PC is 192.168.42.2:

```
iperf -c 192.168.42.2 -i 1 -t 100
```

- Test Results:

```

root@a9 / $ iperf -c 192.168.42.2 -i 1 -t 100
-----
Client connecting to 192.168.42.2, TCP port 5001
TCP window size: 20.2 KByte (default)
-----
[  3] local 192.168.42.1 port 57322 connected with 192.168.42.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[  3]  0.0- 1.0 sec  4.75 MBytes 39.8 Mbits/sec
[  3]  1.0- 2.0 sec  4.62 MBytes 38.8 Mbits/sec
[  3]  2.0- 3.0 sec  4.50 MBytes 37.7 Mbits/sec
[  3]  3.0- 4.0 sec  4.62 MBytes 38.8 Mbits/sec
...
[  3] 99.0-100.0 sec 4.62 MBytes 38.8 Mbits/sec
[  3]  0.0-100.0 sec 413 MBytes 34.6 Mbits/sec

```

3.2.7 Functionalities of AmbaLink: USB Wi-Fi

This section describes the procedure for testing the USB Wi-Fi functionality. The section is organized as follows:

- [\(Section 3.2.7.1\) USB Wi-Fi: Requirements](#)
- [\(Section 3.2.7.2\) USB Wi-Fi: Test Purpose](#)
- [\(Section 3.2.7.3\) USB Wi-Fi: Test Flow and Results](#)

3.2.7.1 USB Wi-Fi: Requirements

In order for the Wi-Fi connection to function correctly, the following requirements must be met:

- Firmware that includes the AmbaLink functionality must have been correctly enabled.
 - **Note that the Linux firmware with the USB Wi-Fi is different from SDIO.**
- The system must boot successfully (both the ThreadX and Linux consoles must be enabled).
- A USB Wi-Fi card must be inserted into the USB slot.
- A Windows or Linux PC must be associated with Camera SSID (e.g., amba_boss).

3.2.7.2 USB Wi-Fi: Test Purpose

The purpose of this test is to verify the USB Wi-Fi connection and test the Wi-Fi throughput.

3.2.7.3 USB Wi-Fi: Test Flow and Results

- Test Flow:
 1. At the Linux console, use the following:
 - `root@a9 / $ echo host > /proc/ambarella/uport`
 2. On the Windows PC, use Wi-Fi to connect to the Ambarella platform:
 - Default AP name is **amba_boss**.
 3. Run the **Iperf** server on the Windows PC:
 - Open the command window.
 - Change to the **Iperf** folder and type `iperf -s`.
 4. At the Linux console, initiate transmit operations and perform a throughput test for 100 seconds:
 - Assuming that the IP address of the Windows PC is 192.168.42.2:

```
iperf -c 192.168.42.2 -i 1 -t 100
```

- Test Results:

```
root@a9 / $ iperf -c 192.168.42.2 -i 1 -t 100
-----
Client connecting to 192.168.42.2, TCP port 5001
TCP window size: 20.2 KByte (default)
-----
[ 3] local 192.168.42.1 port 57322 connected with 192.168.42.2 port 5001
[ ID] Interval Transfer Bandwidth
[ 3] 0.0- 1.0 sec 4.75 MBytes 39.8 Mbits/sec
[ 3] 1.0- 2.0 sec 4.62 MBytes 38.8 Mbits/sec
[ 3] 2.0- 3.0 sec 4.50 MBytes 37.7 Mbits/sec
[ 3] 3.0- 4.0 sec 4.62 MBytes 38.8 Mbits/sec
[ 3] 4.0- 5.0 sec 1.25 MBytes 10.5 Mbits/sec
...
[ 3] 99.0-100.0 sec 4.62 MBytes 38.8 Mbits/sec
[ 3] 0.0-100.0 sec 413 MBytes 34.6 Mbits/sec
```

3.2.8 Functionalities of AmbaLink: Ethernet Support

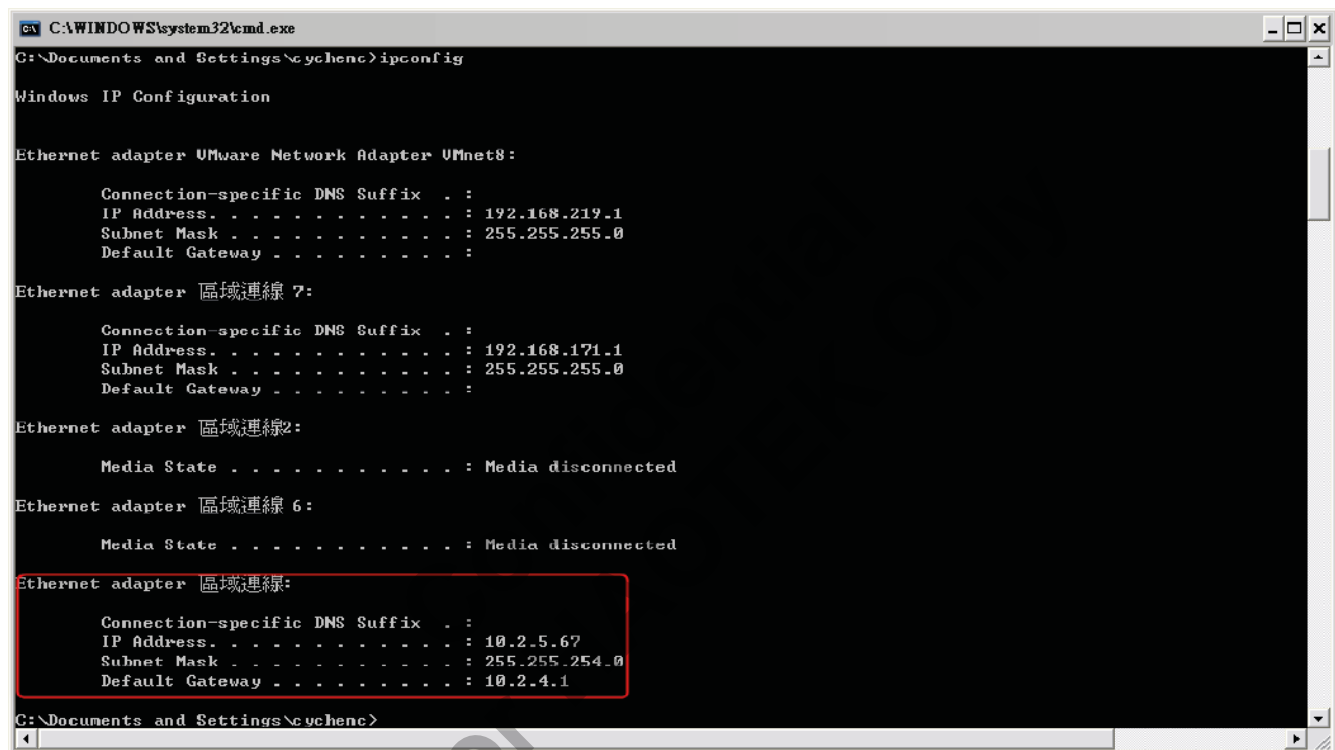
This section describes the procedure for testing the Ethernet functionality. The section is organized as follows:

- [\(Section 3.2.8.1\) Ethernet: Requirements](#)
- [\(Section 3.2.8.2\) Ethernet: Test Purpose](#)
- [\(Section 3.2.8.3\) Ethernet: Test Flow and Results](#)

3.2.8.1 Ethernet: Requirements

In order for the Ethernet connection to function correctly, the following requirements must be met:

- Firmware that includes AmbaLink functionality must have been correctly enabled.
- The system must boot successfully (both the ThreadX and Linux consoles must be enabled).
- An Ethernet cable must be used to connect the PC with the Main Board.
- The PC's IP address must be verified through the command window.



```
CAWINDOWS\system32\cmd.exe
C:\Documents and Settings\cychenc>ipconfig

Windows IP Configuration

Ethernet adapter VMware Network Adapter VMnet8:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . . : 192.168.219.1
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 

Ethernet adapter 區域連線 7:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . . : 192.168.171.1
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 

Ethernet adapter 區域連線2:

    Media State . . . . . : Media disconnected

Ethernet adapter 區域連線 6:

    Media State . . . . . : Media disconnected

Ethernet adapter 區域連線:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . . : 10.2.5.67
    Subnet Mask . . . . . : 255.255.254.0
    Default Gateway . . . . . : 10.2.4.1

C:\Documents and Settings\cychenc>
```

Figure 3-1. IP of the PC in the command window.

3.2.8.2 Ethernet: Test Purpose

The purpose of this test is to verify the Ethernet function.

3.2.8.3 Ethernet: Test Flow and Results

- Test Flow:
 1. At the Linux console, set an IP address (in the same domain as the PC) to eth0.
 - root@a9 / \$ ifconfig eth0 10.2.5.66
 - root@a9 / \$ ping -c 10 10.2.5.67

- **Test Result:**

```
root@a9 / $ ping -c 10 10.2.5.67
PING 10.2.5.67 (10.2.5.67): 56 data bytes
64 bytes from 10.2.5.67: seq=0 ttl=128 time=0.561 ms
64 bytes from 10.2.5.67: seq=1 ttl=128 time=0.426 ms
64 bytes from 10.2.5.67: seq=2 ttl=128 time=0.477 ms
64 bytes from 10.2.5.67: seq=3 ttl=128 time=0.463 ms
64 bytes from 10.2.5.67: seq=4 ttl=128 time=0.427 ms
64 bytes from 10.2.5.67: seq=5 ttl=128 time=0.418 ms
64 bytes from 10.2.5.67: seq=6 ttl=128 time=0.411 ms
64 bytes from 10.2.5.67: seq=7 ttl=128 time=0.536 ms
64 bytes from 10.2.5.67: seq=8 ttl=128 time=0.406 ms
64 bytes from 10.2.5.67: seq=9 ttl=128 time=0.395 ms
--- 10.2.5.67 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 0.395/0.452/0.561 ms
```

3.2.9 Functionalities of AmbaLink: IPC Spin Lock (GPIO)

IPC spin lock (GPIO) is one of the functionalities of AmbaLink. It includes the following sections:

- [\(Section 3.2.9.1\) IPC Spin Lock: Requirements](#)
- [\(Section 3.2.9.2\) IPC Spin Lock: Test Purpose](#)
- [\(Section 3.2.9.3\) IPC Spin Lock: Test Flow and Results](#)

3.2.9.1 IPC Spin Lock: Requirements

1. Firmware with `amba_link` function enabled
2. System boot successfully. (both RTOS and Linux console work)

3.2.9.2 IPC Spin Lock: Test Purpose

To verify the IPC spinlock function for GPIO driver.

3.2.9.3 IPC Spin Lock: Test Flow and Results

- Test Flow:

On the Linux side, check the boot log

- Test Result:

Pass:

No GPIO error message

- Fail:

GPIO error message

3.2.10 Functionalities of AmbaLink: IPC Mutex (I2C)

IPC Mutex is another functionalities of AmbaLink. It includes the following sections:

- [\(Section 3.2.10.1\) IPC Mutex: Requirements](#)
- [\(Section 3.2.10.2\) IPC Mutex: Test Purpose](#)
- [\(Section 3.2.10.3\) IPC Mutex: Test Flow and Results](#)

3.2.10.1 IPC Mutex: Requirements

1. Firmware with AmbaLink function enabled
2. System boots successfully. (both RTOS and Linux console work)

3.2.10.2 IPC Mutex: Test Purpose

To verify the IPC mutex function for I2C driver.

3.2.10.3 IPC Mutex: Test Flow and Results

3.2.10.3.1 Test Flow:

1. At Linux, check the boot log
2. At Linux, check the ethernet
 - root@a9 / \$ ifconfig -a

3.2.10.3.2 Test Result:

- Pass:

1. After step 1, no I2C error message and the LED (D5) blinked
2. After step 2, there should be an eth0 device

```
eth0 Link encap:Ethernet HWaddr BE:37:75:7B:6C:A5  
BROADCAST MULTICAST MTU:1500 Metric:1  
RX packets:0 errors:0 dropped:0 overruns:0 frame:0  
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  
collisions:0 txqueuelen:1000  
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)  
Interrupt:59
```

```
lo Link encap:Local Loopback  
inet addr:127.0.0.1 Mask:255.0.0.0  
UP LOOPBACK RUNNING MTU:65536 Metric:1  
RX packets:0 errors:0 dropped:0 overruns:0 frame:0  
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  
collisions:0 txqueuelen:0  
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

- Fail:

1. Any I2C error message
2. The LED (D5) does not blink.
3. The eth0 does not exist.

3.2.11 Functionalities of AmbaLink: IPC Mutex (NAND)

IPC Mutex (NAND) is also a functionality of AmbaLink. It includes the following sections:

- (Section 3.2.11.1) IPC Mutex: Requirements
- (Section 3.2.11.2) IPC Mutex: Test Purpose
- (Section 3.2.11.3) IPC Mutex: Test Flow and Results

3.2.11.1 IPC Mutex: Requirements

1. Firmware with `amba_link` function enabled
2. System boots successfully. (both RTOS and Linux console work)

3.2.11.2 IPC Mutex: Test Purpose

To verify the IPC mutex function for NAND driver.

3.2.11.3 IPC Mutex: Test Flow and Results

- Test Flow:

Note that there is a NAND known issue, you can only run this test under capture mode.

1. In the Linux operating system, read the partitions information

```
root@a9 / $ cat /proc/mtd
dev: size erasesize name
mtd0: 00020000 00020000 "Bootstrap"
mtd1: 00040000 00020000 "Partition Table"
mtd2: 00040000 00020000 "Bootloader"
mtd3: 00100000 00020000 "SD Firmware Update Code"
mtd4: 02000000 00020000 "System Software"
mtd5: 00500000 00020000 "DSP uCode"
mtd6: 00a00000 00020000 "System ROM Data"
mtd7: 00a00000 00020000 "Linux Kernel"
mtd8: 03c00000 00020000 "Linux Root FS"
mtd9: 01e00000 00020000 "Linux Hibernation Image"
mtd10: 05200000 00020000 "Storage1"
mtd11: 00800000 00020000 "Storage2"
mtd12: 00f20000 00020000 "Index for Video Recording"
mtd13: 000e0000 00020000 "User Settings"
mtd14: 00120000 00020000 "Calibration Data"
```

2. On the Linux side, insert mtd speed test module and set the device to “Linux Hibernation Image”, here is “dev=9”

- root@a9 / \$ modprobe mtd_speedtest dev=9

3. At RTOS, run the throughput test

- a:\> t fioprf_randvrfy a 0x2000 all

- Test Result:

- Pass:

1. After Step 1, Linux console will look like:

```
[ 339.298610]
[ 339.300149] =====
[ 339.313363] mtd_speedtest: MTD device: 9
[ 339.321364] mtd_speedtest: MTD device size 66060288, eraseblock size 131072,
page size 2048, count of eraseblocks 504, pages per eraseblock 64, OOB size 64
[ 339.360753] mtd_speedtest: scanning for bad eraseblocks
[ 339.366099] mtd_speedtest: scanned 504 eraseblocks, 0 are bad
[ 339.943076] mtd_speedtest: testing eraseblock write speed
[ 352.852367] mtd_speedtest: eraseblock write speed is 4999 KiB/s
[ 352.858315] mtd_speedtest: testing eraseblock read speed
[ 358.915261] mtd_speedtest: eraseblock read speed is 10661 KiB/s
[ 359.626098] mtd_speedtest: testing page write speed
[ 373.639806] mtd_speedtest: page write speed is 4605 KiB/s
[ 373.645233] mtd_speedtest: testing page read speed
[ 379.838619] mtd_speedtest: page read speed is 10425 KiB/s
[ 380.548324] mtd_speedtest: testing 2 page write speed
[ 394.040083] mtd_speedtest: 2 page write speed is 4783 KiB/s
[ 394.045683] mtd_speedtest: testing 2 page read speed
[ 400.200789] mtd_speedtest: 2 page read speed is 10489 KiB/s
[ 400.206389] mtd_speedtest: Testing erase speed
[ 400.916348] mtd_speedtest: erase speed is 91376 KiB/s
[ 400.921430] mtd_speedtest: Testing 2x multi-block erase speed
[ 401.298275] mtd_speedtest: 2x multi-block erase speed is 173886 KiB/s
[ 401.304744] mtd_speedtest: Testing 4x multi-block erase speed
[ 401.677135] mtd_speedtest: 4x multi-block erase speed is 175782 KiB/s
[ 401.683605] mtd_speedtest: Testing 8x multi-block erase speed
[ 402.053971] mtd_speedtest: 8x multi-block erase speed is 177230 KiB/s
[ 402.060440] mtd_speedtest: Testing 16x multi-block erase speed
[ 402.429487] mtd_speedtest: 16x multi-block erase speed is 177230 KiB/s
[ 402.436041] mtd_speedtest: Testing 32x multi-block erase speed
[ 402.804055] mtd_speedtest: 32x multi-block erase speed is 178209 KiB/s
[ 402.810610] mtd_speedtest: Testing 64x multi-block erase speed
[ 403.178568] mtd_speedtest: 64x multi-block erase speed is 178209 KiB/s
[ 403.185115] mtd_speedtest: finished
[ 403.188627] =====
```

2. After Step 1, the RTOS console will look like:

- Fail:

1. There is no error message.

3.3 AmbaLink Development: Global Driver on BOSS

In BOSS architecture, RTOS and Linux are running together on a single processor. Accessing the shared resources between dual OSes needs special attention such as the device controller. Driver programming on the dual OSes environment is very tricky, and thus Ambarella does not suggest customers to do it unless customers understand the driver well and is capable to debug it. Ambarella only provides a concept.

A device driver setups device registers and IRQ handler. The ideas related to handling the shared resources are listed below:

1. Critical sections like to setup device registers need to be protected by the global mutex or the spinlock. To use mutex or spinlock depends on how long the critical section is running.
2. IRQ needs to be handled in the right OS. That is, the OS requests device IO or the OS serves the device IO needs to own the IRQ. In short, set IRQ owner to the right OS.
3. Spinlock is used for critical section with very short time of execution. In the region of spinlock protection, OS system calls to trigger scheduling cannot be used. The example driver using the spinlock is the GPIO driver in dual OSes environment.

In the next section, Ambarella provides the examples to show how to write the global drivers.

3.3.1 Global Driver on BOSS: Examples

Type 1: Request-Receive-IRQ Driver

The typical driver of this type is NAND driver. For this driver, the IRQ would happen only after the controller's registers be programmed. Thus, before NAND controller runs the request, the IRQ owner needs to set for the OS running NAND driver. Use global mutex to protect the register setup is needed. Pseudo code is shown as below. The usage of global mutex can refer to Chapter 3 and 8 as well as the APIs of the set IRQ owner is in Section 11.2.13 and 11.2.14 of *AMBARELLA SDK6 API Ambalink*. Furthermore, spinlock APIs can be found in Chapter 2 and 7.

```
void nand_request (void *request)
{
    AmbaIPC_MutexTake (AMBA_IPC_MUTEX_NAND, AMBA_KAL_WAIT_FOREVER);
    AmbaLink_SetIrqOwner(nand_irq, current_os);
    /* registers setup for this request start */
    ...
    /* registers setup for this request end */
    AmbaIPC_MutexGive (AMBA_IPC_MUTEX_NAND);
}
```

Type 2: Ready-Receive-IRQ Driver

The typical driver of this type is SDIO driver for SDIO WiFi operation. The SDIO WiFi interrupt is generated when SDIO device got the packet. Thus the IRQ owner has to set for OS running SDIO driver all the time. Pseudo code is show as below:

```
void sdio_init (void)
{
    .....
    AmbaLink_SetIrqOwner(sdio_irq, current_os);
    .....
}

void sdio_request (void *request)
{
    AmbaIPC_MutexTake(AMBA_IPC_MUTEX_SD1, AMBA_KAL_WAIT_FOREVER);
    /* registers setup for this request start */
    .....
    /* registers setup for this request end */
    AmbaIPC_MutexGive(AMBA_IPC_MUTEX_SD1);
}
```

3.4 AmbaLink Development: AmbaLink RPC Program Development

This section describes the process of the AmbaLink RPC program development. The section is organized as follows:

- [\(Section 3.4.1\) AmbaLink RPC Program Development: Create a Common Header File](#)
- [\(Section 3.4.2\) AmbaLink RPC Program Development: How RPC Works](#)
- [\(Section 3.4.3\) AmbaLink RPC Program Development: Create the Server and Client Programs](#)
- [\(Section 3.4.4\) AmbaLink RPC Program Development: Build RPC Program](#)

3.4.1 AmbaLink RPC Program Development: Create a Common Header File

This section introduces the process to create a common header file as follows:

- [\(Section 3.4.1.1\) Create a Common Header file: Including Header File](#)
- [\(Section 3.4.1.2\) Create a Common Header file: The Necessary Definition](#)
- [\(Section 3.4.1.3\) Create a Common Header file: The RPC Function Prototype](#)
- [\(Section 3.4.1.4\) Create a Common Header file: File Placement](#)

3.4.1.1 Create a Common Header file: Including Header File

- `AmbaIPC_Rpc_Def.h`

The common RPC data structures are defined in the above file, so it is necessary to be included as part of the RPC program development.

3.4.1.2 Create a Common Header file: The Necessary Definition

- `Program id`

Program ID must be unique and fixed. The format should be restricted to `ModuleName_PROG_ID`¹. (Module-Name is programmer-defined, but `PROG_ID` is fixed)

To manage the IPCs easily, Ambarella defines the program ID range². Program ID should be within `0x10000001~0x1FFFFFFF`, if the server is in RTOS; Program ID should be within `0x20000001~0x2FFFFFFF`, if the server is in the Linux user space.

Note that the program ID should be recorded in `ambalink_sdk/pkg/ipcdef/PROGID_SPACE`.

- `Version number`

The naming rule is `ModuleName_VER`.

- `RPC procedure id`

The naming rule is `ModuleName_ProcedureName`, and examples are **AMBA_RPC_PROG_TEST_PRINT** and **AMBA_RPC_PROG_TEST_SUM** (Please refer to `AmbaIPC_RpcProg_Test.h`).

3.4.1.3 Create a Common Header file: The RPC Function Prototype

For ease of use, Ambarella simplifies the RPC program interface. Here, we use the following test codes to describe.

- **Example (client side):**

```
AMBA_IPC_REPLY_STATUS_e AmbaRpcProg_Test_Print_Clnr(  
    const char *pStr, //Client request  
    int *pResult, //The responded results through RPC call  
    int Clnr //Client ID  
);
```

- The format of return value has to be `AMBA_IPC_REPLY_STATUS_e`³.
- The data structure of Request is programmer-defined depending on what kind of data structure the server can handle. In **AmbaRpcProg_Test_Sum** function, a structure **AMBA_RPC_PROG_TEST_SUM_ARG_s** is defined to store the inputs for summation (Please refer to `AmbaIPC_RpcProg_Test.h`).

¹ AmbaLink will scan all the RPC header files and calculate the CRC values. Program ID is used to index CRC values and hence, the format of the program ID should be fixed.

² The host can be specified by assigning the host in the **ambaipc_clnt_create** function. However, this is just for testing purposes.

³ The detailed definition can be looked up in `AmbaIPC_Rpc_Def.h`.

- **Example (server side):**

```
void AmbaRpcProg_Test_Print_Svc(
    const char *pStr, //Client request
    AMBA_IPC_SVC_RESULT_s *pRet //The responded result
);
```

Note:

- The first parameter is exactly the same as the first parameter in the RPC client.
- The responded results includes the calculated results, status, and communication mode.

3.4.1.4 Create a Common Header file: File Placement

The common header files for the RPC program should be placed in the predefined directories:

- ThreadX: `rtos/vendors/ambarella/inc/ssp/link/rpcprog.`
- Linux: `ambalink_sdk/pkg/ipcdef.`

Note that the common header files should be exactly the same in a dual OS or the RPC will fail due to the CRC checking error.

3.4.2 AmbaLink RPC Program Development: How RPC Works

AmbaLink has provided APIs for client and server communication. The mechanism of the RPC call is simply represented in [Figure 3-2](#). On the server side, the server is needed to be registered by the **ambaipc_svc_register**⁴ function. Then, the server will be on standby for client requests. On the client side, the client should be created by calling the **ambaipc_clnt_create**⁵ function. When the client has a request to the server, **ambaipc_clnt_call**⁶ will be invoked to bring the client requests to the server. Finally, the server and the client need to be cleared by invoking **ambaipc_svc_unregister** and **ambaipc_clnt_destroy**, respectively.

Note that the detailed explanations of these APIs are defined in *AMBARELLA SDK6 API AmbaLink* document.

⁴ When registering a service, the program ID will be treated as the server ID.

⁵ When creating a client, the program ID is needed to bind the server.

⁶ In **ambaipc_clnt_call**, the client needs to specify the server and the corresponding procedure.

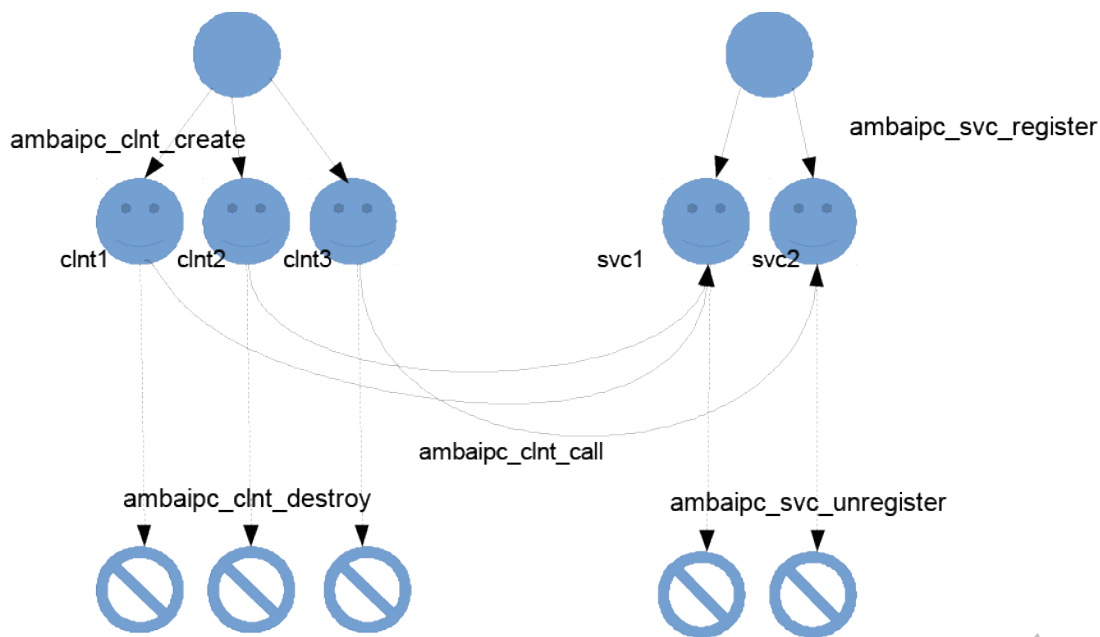


Figure 3-2. RPC call Mechanism.

3.4.3 AmbaLink RPC Program Development: Create the Server and Client Programs

The prototypes are defined in the header file. The user needs to construct the server and client programs. For instance, Linux passes a string to ThreadX and expects that ThreadX will print the string. The implementation of the RPC client is shown below:

Example (client side):

```
AMBA_IPC_REPLY_STATUS_e AmbaRpcProg_Test_Print_Clnr(const char *pStr, int
*pResult, int Clnr)
{
    AMBA_IPC_REPLY_STATUS_e status;
    status = ambaipc_clnt_call(Clnr, AMBA_RPC_PROG_TEST_PRINT, (void *)
pStr, strlen(pStr)+1, 0, 0, 100);
    return status;
}
```

- The procedure number should be specified while calling **ambaipc_clnt_call**.
- The status of the RPC call has to be recorded.

Correspondingly, ThreadX should have the implementation of the server codes. Please refer to the following example:

Example (server side):

```
void AmbaRpcProg_Test_Print_Svc(const char *msg, AMBA_IPC_SVC_RESULT_s *pRet)
{
    AmbaPrint("\t[svc echo] %s\n", msg); pRet->Mode = AMBA_IPC_ASYNCHRONOUS; pRet->Status = AMBA_IPC_REPLY_SUCCESS;
}
```

- The RPC status and communication mode should be specified.

For the server to handle client calls appropriately, information about the RPC program (i.e., communication modes and procedure function) should be specified while registering the service. Please refer to the following example:

Example:

```
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_Test_Print_Svc;
prog_info->pProcInfo[1].Proc = (AMBA_IPC_PROC_f) &AmbaRpcProg_Test_Sum_Svc;
AmbaIPC_SvcRegister(prog_id, AMBA_RPC_PROG_TEST_VER, "test_svc",
15, stack, 0x1000, prog_info, 0);
```

The example codes can be discovered in `ambalink_sdk/pkg/ambaipc_test/clnt_test.c` and `rtos/ssp/unittest/AmbaIPC Test.c`.

3.4.4 AmbaLink RPC Program Development: Build RPC Program

After completing the implementation of the RPC client and server, AmbaLink should be rebuilt. The following uses the A9 chip as an example.

On the Linux side:

1. Change to the correct directory.

```
$ cd ambalink_sdk/output/a9_ambalink
```

2. Rebuild package **ipcdef** to let AmbaLink know a new RPC header file is added and calculate the corresponding CRC value.

```
$ make ipcdef-rebuild; make
```

3. Rebuild the package that includes the RPC program. Here, the user uses package **ambaipc-test** as an example.

```
$ make ambaipc_test-rebuild; make
```

On the ThreadX side:

1. Change to the directory

```
$ cd system_a9_rtos/rtos/
```

2. Build the desired target according to the chosen config

```
$ make
```

Confidential
For HAOTEK Only

4 AmbaLink Debugging

4.1 AmbaLink Debugging: Overview

This chapter provides the debugging guide for users to diagnose the problems of AmbaLink. The organization of this chapter is listed as follows:

- [\(Section 4.2\) AmbaLink Debugging: Hibernation Corruption](#)
- [\(Section 4.3\) AmbaLink Debugging: FAQ for RPMsg and RPC](#)

4.2 AmbaLink Debugging: Hibernation Corruption

4.2.1 Hibernation Corruption: Overview

To speed up the boot time of Linux, Linux supports hibernation technology. Hibernation takes a snapshot of the current system status, including the registers of devices, and restores the image when the system returns from hibernation. When this happens, there is a possibility that the registers of the shared resources are corrupted by Linux in a dual-OSes system. Ambarella provides the following methods to prevent the corruption of the shared hardware resources.

This chapter provides information on the following:

- [\(Section 4.2.2\) Hibernation, GPIO and IRQ Corruption: GPIO](#)

4.2.2 Hibernation, GPIO and IRQ Corruption: GPIO

Ambarella provides a touch list named “**char OnlyResumeGpios[]**” in `AmbaLinkMemCfg.c`. Linux only restores the registers of these GPIOs while resuming.

Ambarella recommends the user to list all the GPIOs used by Linux.

4.3 AmbaLink Debugging: FAQ for RPMsg and RPC

4.3.1 FAQ for RPMsg and RPC: Overview

This chapter provides the following sections:

- (Section 4.3.2) FAQ for RPMsg and RPC: RPMsg Related Questions
- (Section 4.3.3) FAQ for RPMsg and RPC: RPC Related Questions

4.3.2 FAQ for RPMsg and RPC: RPMsg Related Questions

(Related to Ambarella RPMsg implementation)

- **How is the memory allocated for the RPMsg channel?**

The memory for the RPMsg channel is from the AmbaLink shared memory. For details, please refer to `AmbaLinkMemoryCfg.c`. Details such as the buffer size and entry number can be also found there.

- **Is there a queuing mechanism to hold messages or is a channel defined as one message queue?**

As long as the RPMsg buffers are available, the channel can send out the message. If there is no buffer, `AmbaIPC_TrySend` will return failed.

- **How's memory protected? Is there a single spinlock for all RPMsg shared memory or is there a spinlock per channel?**

There is a shared structure between ThreadX and Linux to maintain the RPMsg buffer usage. No spinlock is needed.

(Related to QoS)

- **What is the expected throughput?**

It depends on the system load. The average time for one RPMsg transferred is about 100 μ s in preview mode. However, the statistics may change in different cases. Ambarella provides AmbaIPC profiling tool to get the throughput result. For more details, please contact an Ambarella representative.

- **Is there a degradation in performance as more channels are used?**

As long as the RPMsg buffers are available, the channel can send message continuously. Messages are held in the RPMsg buffer and the other OS will process it when an interrupt is received.

- **Asymmetric allocation: Is it possible to allocate more memory or bandwidth per channel over the other channel?**

No, Ambarella does not support asymmetric allocation for the RPMsg channel.

(Related to RPMsg programming)

- **When `rpmmsg_send` returns an error code, does it mean that message has not reached the far end, or is it that it just can't queue the message?**

If `AmbalPC_TrySend()` returns failure, it means there is no available buffer for the RPMsg transmission. Therefore, the message does not need to be sent.

- **What is the MTU size per RPMsg?**

`RPMsg_BUF_SIZE` – 16 (RPMsg header size)

`RPMsg_BUG_SIZE` can be found in `AmbaLinkMemoryCfg.c`.

- **What happens when Linux tries to send messages over a channel which is not initialized yet?**

It cannot happen cause ThreadX will run `rpmmsg_init()` first, and it would wait until Linux RPMsg framework is ready.

- **Does the queue get empty when the system is suspended? Does the memory get initialized when the system resumes?**

RPMsg information would be saved/restored when the system suspends/resumes.

- **Why does RTOS block but Linux print RPMsg successful information after `AmbalPC_RegisterChannel` is called?**

Sometimes, after the RPMsg registration in RTOS, response from Linux is blocked. Please check the corresponding RPMsg channel in Linux, it may not have found the corresponding RPMsg driver or the channel name may not match.

- **What the limitation is when implementing RPMMSG callback function?**

Please do not let event flag and mutex wait forever in the callback function. It may lead to deadlock.

4.3.3 FAQ for RPMsg and RPC: RPC Related Questions

Before creating a RPC program, please read [Section 3.4 “AmbaLink Development: AmbaLink RPC Program Development”](#) in *AMBARELLA SDK6 AN AmbaLink*.

(1) Possible error messages:

“binding error! Client needs to bind again next time”

“Binding svc program timeout!”

This error means that the client cannot connect the binder correctly.

- Binder is not enabled.

Please check if the binder is running. Customers can learn how to run the binder by referring to [Section 2.3](#) in the document *AMBARELLA SDK6 AN AmbaLink*. If the binding is not successful while the client is created, the binding will connect again during the following client calls. Therefore, the client call should work as long as the binder is enabled later even if the first binding fails.

(2) Possible error messages:

“ERROR!! Program id is not found”

- The corresponding service is not registered.
Please check if the server is enabled or not.
- The format of the program ID is wrong.
Please refer to [Section 3.2.3](#) in the document *AMBARELLA SDK6 AN AmbaLink*.

(3) Possible error messages:

“The CRC values are not matching”

- The RPC header file is in the wrong folder.
Please refer to [Section 3.4.2](#) in the document *AMBARELLA SDK6 AN AmbaLink*. Customers can have their own RPC header files elsewhere on the ThreadX side. Take **ssp_unittest** as an example, the additional RPC file directory can be appended at parameter **RPC_CRC_HEADERS** as follows:

```
RPC_CRC_HEADERS="${src-tree}/vendors/ambarella/inc/ssp/link/rpcprog ${src-tree}/vendors/ambarella/inc/ssp/example/rpcprog"
```

- The RPC header files for dual OSES are not consistent.
Please ensure that these two RPC header files are exactly the same.
- Add or modify the RPC header file without rebuilding CRC.
Please refer to [Section 3.4.4](#) in the *AMBARELLA SDK6 AN AmbaLink* document.

(4) Possible error messages:

“svc_thread_func receiving error:No buffer space available”

- Send rate from RTOS to Linux is too fast.
The send rate is limited to the restriction of the socket buffer in Linux. After the evaluation in the preview mode, it is concluded that the maximum sending rate is about 200 (packet/μs).

(5) RPC deadlock issue:

It may lead to deadlock when all the following conditions are satisfied:

- Server in RTOS and client in Linux
- The number of RPC procedures is large than 1.
- It calls a synchronous IPC (RPC/RPMSG) in one of the RPC svc procedures.

The resource allocation graph is displayed below:

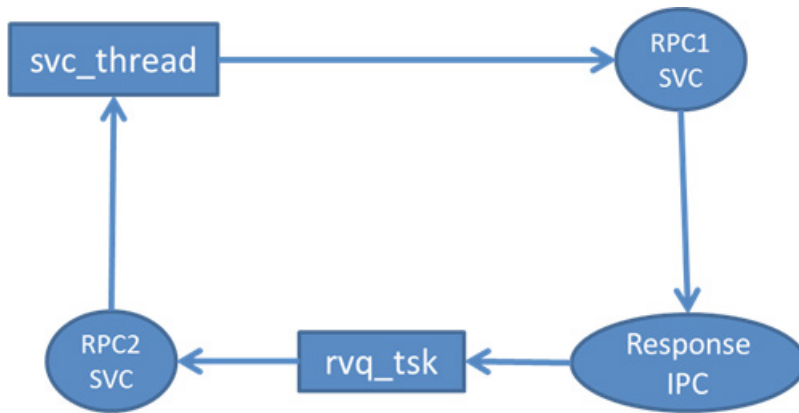


Figure 4-1. Deadlock Resource Allocation Graph.

When RPC 1 svc occupies the svc thread, RPC 2 svc should wait until the RPC 1 svc is done. However, if RPC 1 is waiting for the response of another IPC, but this IPC is blocked because the rvq_tsk is occupied by RPC 2 svc, a deadlock is induced.

Solution 1: Let one svc_thread only focus on RPC1 svc, so RPC 2 svc does not need to wait for RPC 1 svc.

Solution 2: Let svc_thread to be in context switch to RPC 2 svc when RPC 1 waits for the response of IPC.

Appendix 1 Additional Resources

The most important references for this document are listed below.

- AMBARELLA_SDK6_API_AmbaLink
- AMBARELLA_SDK6_AN_Build_Environment
- AMBARELLA_SDK6_AN_AmbaLink_Migration

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

Confidential
For HAOTEK Only

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
1.0	17 September 2014	Preliminary release
1.1	11 March 2015	Update in Sections 2.2.1, 3.2.1.3, 3.2.2.3.1, 3.2.2.3.2, 3.2.3.3, 3.3.1.4, 3.3.3 , 3.3.4. & 4.3.3. Added Sections 2.4, 2.4.1, 3.2.9, 3.2.9.1, 3.2.9.2, 3.2.9.3, 3.2.10, 3.2.10.1, 3.2.10.2, 3.2.10.3, 3.2.11, 3.2.11.1, 3.2.11.2 & 3.2.11.3.
1.2	27 August 2015	Add Section 2.2.1 AmbaLink Memory Size Adjustment: AmbaLink Memory Layout Change, 2.2.2 AmbaLink Memory Size Adjustment: RPMsg/RPC Memory Size Adjustment, 2.5 AmbaLink Customization: Change Linux Root File-System Type, 2.6 AmbaLink Customization: UART Console customization, 2.2.2.2 RPMsg/RPC Memory Size Adjustment: RPMsg Memroy Size Adjustmen. Update in Section 2.2, 2.2.1, 4.3.3. Remove 4.4 AmbaLink Debugging: Debug Linux.
1.3	10 December 2015	Update in Sections 1.1 Overview: Introduction, 2.2.2 AmbaLink Memory Adjustment: AmbaLink Memory Layout Change, 2.3.1 Hardware Resources Management: Overview, 3.4.4 AmbaLink RPC Program Development: Build RPC Program, Appendix 1 Additional Resources, 3.2.7.3 USB Wi-Fi: Test Flow Results, and 4.3.2 FAQ for RPMsg and RPC: RPMsg Related Questions. Delete Section 1.2 Introduction: Scope of Document and 4.2.3 Hibernation, GPIO and TRQ Corruption: GIC.
1.4	17 February 2016	Update in Sections 2.2.2, 2.2.2.1 and 2.2.2.2. Add Section 2.2.2.3 RPMSG Memory Size Adjustment: Vring Size Adjustment.

Table A3-1. Revision History.