



MYD-YT113X RISC-V

Processor Application Note

File status: [] Draft [✓] Release	FILE ID:	MYIR-MYD-YT113X-SW-AN02-EN-L5.4.61
	VERSION:	V1.0[Doc]
	AUTHOR:	MSW0307
	RELEASE:	2024-08-09
	UPDATED:	2024-07-29

Revision History

VERSION	AUTHOR	PARTICIPANT	DATE	DESCRIPTION
V1.0[Doc]	MSW0307	MSW0041	2024-08-09	Official Release

CONTENT

Revision History	- 1 -
CONTENT	- 2 -
1. Overview	- 4 -
2. Environment Preparation	- 5 -
2.1. Hardware Resources	- 5 -
2.1.1. Hardware Connection	- 5 -
2.2. Software Resources	- 6 -
3. Cortex-A-Cortex-R communication	- 8 -
4. rtos catalog introduction	- 12 -
4.1. lichee/rtos directory	- 13 -
4.1.1. arch directory	- 14 -
4.1.2. Components directory	- 15 -
4.1.3. Drivers directory	- 15 -
4.1.4. Include directory	- 16 -
4.1.5. Kernel directory	- 16 -
4.1.6. Projects directory	- 16 -
4.2. lichee/rtos-hal directory	- 18 -
4.2.1. hal directory	- 18 -
4.2.2. include directory	- 19 -
5. Rtos Image Compile and Enable Cortex-R	- 20 -
5.1. buildroot builds overall	- 20 -
5.1.1. Selection of overall platform options	- 20 -
5.1.2. Compiling an rtos image	- 21 -
5.1.3. Packaging Cortex-R images	- 23 -
5.1.4. enable Cortex-R	- 24 -
5.1.5. Cortex-A communicates with Cortex-R	- 24 -
5.2. Compile rtos separately	- 25 -

- 5.2.1. Loading Compilation Environment..... - 25 -
- 5.2.2. Select Configuration..... - 25 -
- 5.2.3. Compiling the rtos image..... - 27 -
- 5.2.4. Update the rtos image file..... - 28 -
- 5.2.5. enable Cortex-R..... - 28 -
- 5.2.6. Cortex-A communicates with Cortex-R..... - 29 -

- 6. Cortex-A and Cortex-R source code..... - 30 -
 - 6.1. Cortex-A Source Code (ARM)..... - 30 -
 - 6.1.1. Obtaining Source Code..... - 30 -
 - 6.1.2. Compile source code..... - 30 -
 - 6.1.3. Key Code Description..... - 30 -
 - 6.2. Cortex-R source code (RISC-V)..... - 32 -
 - 6.2.1. Get source code..... - 32 -
 - 6.2.2. Compile source code..... - 32 -
 - 6.2.3. Key Code Description..... - 33 -

- Appendix A..... - 36 -

1. Overview

RISC-V is an open-source Instruction Set Architecture (ISA) based on the Reduced Instruction Set Computing (RISC) principles. It was first introduced by researchers at the University of California, Berkeley in 2010. The aim was to provide a simple, efficient, and open architecture for use by academia, industry, and individual developers. The design philosophy of RISC-V is to simplify the instruction set to achieve high performance and flexibility while maintaining sufficient extensibility to adapt to various application scenarios.

The main advantage of RISC-V is its open source nature, which means that anyone is free to use, modify, and implement the architecture without paying licensing fees. This openness fosters technological innovation and broad community participation. RISC-V provides a standard base instruction set and allows developers to add custom extensions to meet specific needs. It supports applications ranging from embedded devices to high-performance computing, including but not limited to Internet of Things (IoT) devices, artificial intelligence accelerators, and server processors.

In recent years, the adoption of RISC-V has gradually increased, supported by numerous technology companies such as SiFive and WesternDigital. This trend is driving not only the growth of the RISC-V ecosystem, but also the maturation of the associated toolchains and software, making RISC-V an increasingly competitive choice, especially in applications that require customization and high performance.

This paper mainly introduces how to use Cortex-R (RISC-V) to communicate with Cortex-A (ARM) on MYD-YT113-I model development board.

Note: RISC-V is abbreviated as Cortex-R and ARM is abbreviated as Cortex-A

2. Environment Preparation

2.1. Hardware Resources

- MYD-YT113-I development board
- TTL serial port cable two
- 12V power supply

2.1.1. Hardware Connection

J4 debug (UART5) is the Cortex-A TTL serial port of the development board (red box 1), and UART4 is the Cortex-R TTL serial port of the development board (red box 2). When using the Cortex-A to communicate with the Cortex-R, you need to access these two serial ports, the connection diagram is as follows:

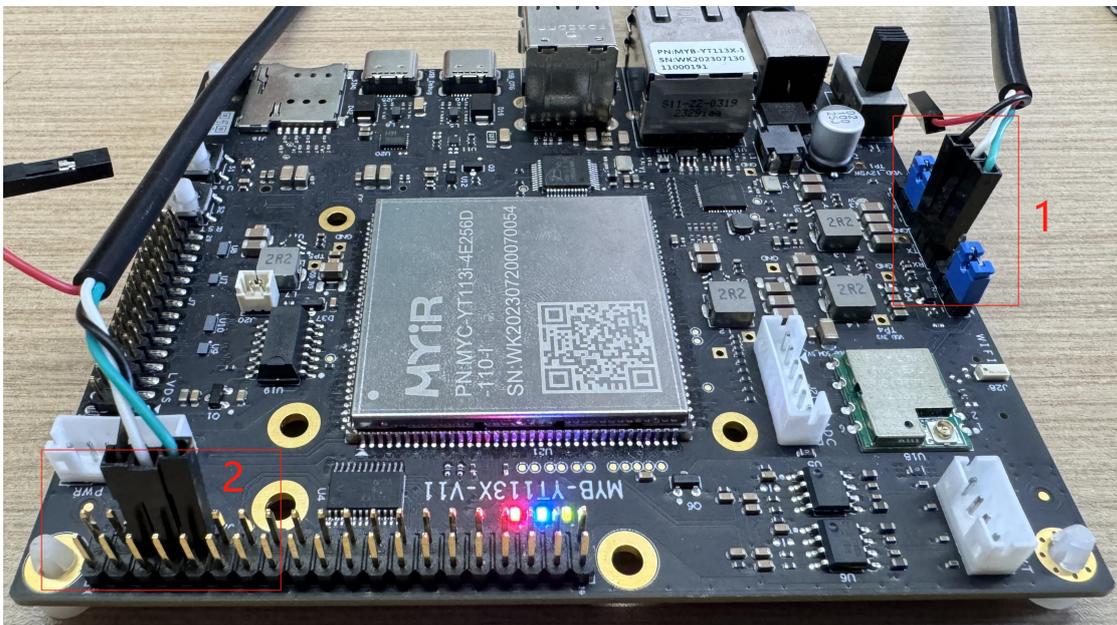


Figure 2-1. Physical diagram of Cortex-R and Cortex-A connection

Users can check the following schematic diagram to connect the serial ports of the Cortex-A and the Cortex-R.

The Cortex-A TTL serial port schematic is as follows:

J4 (UART5): pin 1 (RX) pin 2 (TX) pin 3 (GND)

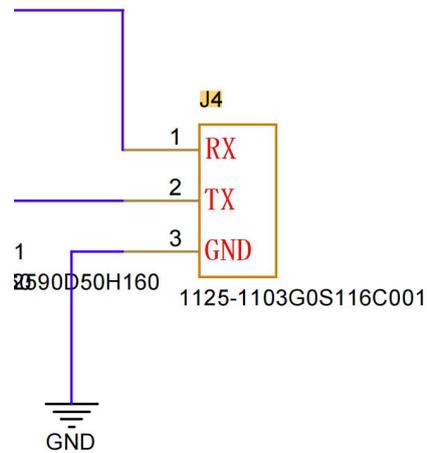


Figure 2-2. Cortex-A TTL schematic diagram

The Cortex-R TTL serial port schematic is shown below:

J2 (UART4): pin 6 (GND) pin 8 (TX) pin 10 (RX)

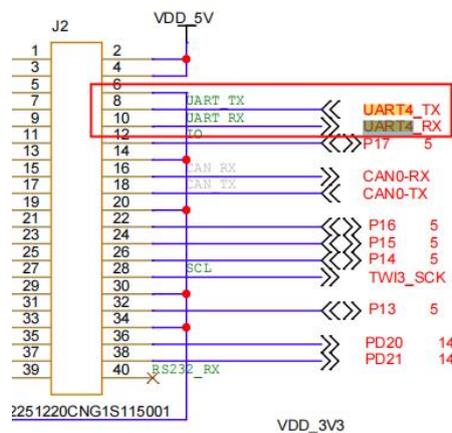


Figure 2-3. Cortex-R TTL schematic diagram

2.2. Software Resources

- MYD-YT113X-Distribution-L5.4.61-V2.0.0.tar.gz
- Ubuntu20.04 64bit

Instead of explaining how to set up the virtual machine environment and unzip the SDK, please refer to Chapter 2 in the "*MYD-YT113X Linux Software Development Guide*" for details on setting up the host environment.

3. Cortex-A-Cortex-R communication

MYiR has built a good image to support Cortex-A and Cortex-R communication, users can get the image from the "02-Images/MYD-YT113-I/eMMC" directory, all the current images are supported, choose one of the images to burn to the development board, please refer to Chapter 5 of "MYD-YT113X Linux Software Development Guide" for the specific burning steps.

For hardware connection, please refer to section 2.2.1.

- Enable Cortex-R

Execute the following command on the development board to turn on the Cortex-R:

```
root@myd-yt113-i:~# echo start > /sys/class/remoteproc/remoteproc1/state
```

The following printout appears in the Cortex-R serial port (UART4) after executing this command:

```
rt: irq for uart4 already enabled

*****
** Welcome to T113_I_C906 FreeRTOS V1.6.0 **
** Copyright (C) 2019-2021 AllwinnerTech **
**                                     **
**   starting riscv FreeRTOS           **
*****

Date:Jul 24 2024, Time:19:26:09
[RV] [AMP_INFO][openamp_platform_init:112]rproc0 init
[RV] [AMP_INFO][rproc_common_mmap:128]remoteproc mmap pa: 0x4233c430,
da: 0xffffffff, size = 0x94
```

```
[RV] [AMP_INFO][rproc_common_mmap:199]map pa(0x4233c430) to va(0x4233c430)
[RV] [AMP_INFO][rproc_common_mmap:128]remoteproc mmap pa: 0x42900000, da: 0x42900000, size = 0x40000
[RV] [AMP_INFO][rproc_common_mmap:199]map pa(0x42900000) to va(0x42900000)
[RV] [AMP_INFO][rproc_common_mmap:128]remoteproc mmap pa: 0xffffffff, da: 0x42940000, size = 0xd46
[RV] [AMP_INFO][rproc_common_mmap:199]map pa(0x42940000) to va(0x42940000)
[RV] [AMP_INFO][rproc_common_mmap:128]remoteproc mmap pa: 0xffffffff, da: 0x42942000, size = 0xd46
[RV] [AMP_INFO][rproc_common_mmap:199]map pa(0x42942000) to va(0x42942000)
[RV] [AMP_INFO][openamp_sunxi_create_rpmsg_vdev:365]Wait connected to remote master
[RV] [AMP_INFO][openamp_sunxi_create_rpmsg_vdev:371]Connecte to remote master Succeeded
[RV] [AMP_INFO][openamp_platform_init:157]rproc0 init done
[RV] [AMP_INFO][openamp_platform_init:112]rproc0 init
[RV] [AMP_INFO][openamp_platform_init:122]rproc0 already init.
[RV] [AMP_INFO][openamp_ept_open:321]Waiting for ept('sunxi,rpmsg_ctrl') ready
[RV] [AMP_INFO][openamp_ept_open:326]ept('sunxi,rpmsg_ctrl') ready! src: 0x400, dst: 0x400
rpmsg ctrldev: Start Running...
Wating message!
```

- Execute Cortex-A application program

After enabling the Cortex-R, the user can execute the pre compiled Cortex-A application of MYiR to communicate with the Cortex-R. In the Cortex-A serial port (UART5), execute the following command:

```

root@myd-yt113-i:~# ./rpmsg_echo -r c906_rproc@0 -m myir_test -n 5
Sending message #0:myir_test
Receiving message #0:myir_test
Sending message #1:myir_test
Receiving message #1:myir_test
Sending message #2:myir_test
Receiving message #2:myir_test
Sending message #3:myir_test
Receiving message #3:myir_test
Sending message #4:myir_test
Receiving message #4:myir_test
Deleted rpmsg endpoint file: /dev/rpmsg1
At this point, the Cortex-R prints as follows:
[RV] [AMP_INFO][openamp_ept_open:321]Waiting for ept('sunxi,rpmsg_client') ready
[RV] [AMP_INFO][openamp_ept_open:326]ept('sunxi,rpmsg_client') ready! src: 0x401, dst: 0x401
rpmsg1: binding
Receiving message #0: myir_test
Sending message #0: myir_test
Receiving message #1: myir_test
Sending message #1: myir_test
Receiving message #2: myir_test
Sending message #2: myir_test
Receiving message #3: myir_test
Sending message #3: myir_test
Receiving message #4: myir_test
Sending message #4: myir_test
rpmsg1: unbinding
rpmsg1 echo thread exit...

```

From the printout of the Cortex-A and Cortex-R, it can be seen that the Cortex-A sends five sets of *"myir_test"* data to the Cortex-R, and the Cortex-R receives it

and returns it to the Cortex-A, which communicates successfully with the Cortex-R at this time.

4. rtos catalog introduction

After setting up the hosting environment, unpack the "*MYD-YT113X-Distribution-L5.4.61-V2.0.0.tar.gz*" SDK archive to get the following directory structure:

```
PC$ $HOME/T113$ tree -L 1
├── brandy
├── build
├── buildroot
├── build.sh -> build/top_build.sh
├── device
├── kernel
├── openwrt
├── platform
├── prebuilt
├── rtos
└── tools

10 directories, 1 file
```

This chapter mainly introduces the rtos directory, which has the following structure:

```
PC$ $HOME/T113/rtos$ tree -L 2
.
├── board
│   ├── t113_i_c906           t113_i_c906 Board level configuration directory
│   ├── t113_s3p_c906
│   ├── t113_s4_c906
│   └── t113_s4p_c906
├── envsetup.sh -> tools/scripts/source_envsetup.sh
└── lichee
```

```

| └─ dsp                                DSPFreeRTOS System
| └─ rtos                                C906/E906 FreeRTOS System
| └─ rtos-components                    FreeRTOS PublicComponents
| └─ rtos-hal                            BSP Device
└─ tools
    └─ image-file
    └─ scripts
    └─ tool
    └─ win-tools
    └─ xradio-tools

```

16 directories, 1 file

- board : Contains the configuration directory for each SoC board
- envsetup.sh : SDK environment initialization scripts
- lichee : FreeRTOS system main directory
- tools : A directory of packaging-related tools and scripts.

4.1. lichee/rtos directory

The beginning of Chapter 4 introduced the rtos directory structure under the top-level directory of the source code, and this subsection talks about the "*lichee/rtos*" directory under that directory, which stores the important source code and configuration files for compiling and building RISC-V. The following is the "*lichee/rtos*" directory structure:

```
PC$ $HOME/T113/rtos/lichee/rtos$ tree -L 1
```

```

└─ arch
└─ components
└─ drivers
└─ include

```

```
├─ Kconfig
├─ kernel
├─ LICENSE
├─ Makefile
├─ projects
├─ scripts
└─ tools
```

8 directories, 3 files

- arch : Processor architecture related
- build : Compile temporary file output directory
- components : Components
- drivers : Drivers
- include : header files
- kernel : FreeRTOS kernel
- Projects : Scheme Engineering
- tools : Toolchain

Directories under "*lichee/rtos*" are explained separately in the following sections

4.1.1. arch directory

The arch directory is mainly for SoC architecture related content, each SoC is managed in a separate directory, which mainly includes the implementation of ARCH initialization, interrupt handling, exception handling, and memory mapping functions related to the risc-v architecture.

```
PC$ $HOME/T113/rtos/lichee/rtos/arch$ tree -L 1
```

```
.
├─ common
├─ Kconfig
├─ Makefile
├─ objects.mk
└─ risc-v
```

2 directories, 3 files

4.1.2. Components directory

The "*Components*" directory contains "*allwinner*" and third-party components.

```
PC$ $HOME/T113/rtos/lichee/rtos/components$ tree -L 1
.
├── aw
├── common -> ../../rtos-components
├── Kconfig
├── Makefile
├── objects.mk
└── thirdparty
```

3 directories, 3 files

4.1.3. Drivers directory

The "*drivers*" directory contains the required peripheral drivers, mainly the specific implementation of each peripheral controller driver (hal soft link) and the OSAL layer interface (osal).

```
PC$ $HOME/T113/rtos/lichee/rtos/drivers$ tree -L 1
.
├── drv
├── Kconfig
├── Makefile
├── objects.mk
├── osal
└── rtos-hal -> ../../rtos-hal/
```

3 directories, 3 files

4.1.4. Include directory

The "*include*" directory centrally manages the data structure definitions and function declarations provided by each module

```
PC$ $HOME/T113/rtos/lichee/rtos/include$ tree -L 1
├── arch
├── FreeRTOS_POSIX
├── .....
└── vsprintf.h
```

- arch : Architecture-related header files
- FreeRTOS_POSIX : POSIX header files

4.1.5. Kernel directory

The "*kernel*" directory mainly contains the "*kernel*" source code of "*FreeRTOS*" and the system function related code implemented by "*allwinner*".

```
PC$ $HOME/T113/rtos/lichee/rtos/kernel$ tree -L 1
.
├── FreeRTOS
├── FreeRTOS-orig
├── Kconfig
├── Makefile
├── objects.mk
└── Posix
```

3 directories, 3 files

4.1.6. Projects directory

Each subdirectory in the projects directory represents a project, which implements the main entry. bin compiled by different project has different functions, and each project has an independent FreeRTOSConfig configuration. For example, the

MYD-YT113-I model development board corresponds to the "*t113_i_c906*" subdirectory.

```
PC$ $HOME/T113/rtos/lichee/rtos/projects$ tree -L 1
```

```
.  
├── config.h  
├── Kconfig  
├── Makefile  
├── objects.mk  
├── t113_i_c906  
├── t113_s3p_c906  
├── t113_s4_c906  
└── t113_s4p_c906
```

```
4 directories, 4 files
```

4.2. lichee/rtos-hal directory

The beginning of chapter 4 introduces the rtos directory structure under the top-level directory of the source code, and this section talks about the "*lichee/rtos-hal*" directory under the directory, which is the BSP driver directory for storing various driver codes. The rtos-hal subdirectory in the "*lichee/rtos/drivers*" directory is softlinked to this directory. The following describes this directory.

```
PC$ $HOME/T113/rtos/lichee/rtos-hal$ tree -L 1
```

```
.
├── hal
└── include
```

```
2 directories, 0 files
```

The "*lichee/rtos hal*" directory mainly includes directories such as hal (BSP driver code) and include (driver related header files), which will be introduced separately below.

4.2.1. hal directory

The hal directory mainly contains the peripheral driver code and driver test code, the "*source*" subdirectory is for the driver code and the test subdirectory is for the driver "*test*" code.

```
PC$ $HOME/T113/rtos/lichee/rtos-hal/hal$ tree -L 2
```

```
├── Makefile
├── source
│   ├── ccmu
│   ├── gpio
│   ├── .....
│   ├── uart
│   └── watchdog
```

```
└─ test
├─ ccmu
├─ gpio
├─ .....
├─ uart
└─ watchdog
```

4.2.2. include directory

The include directory contains driver-related header files and system-related interface header files.

```
PC$ $HOME/T113/rtos/lichee/rtos-hal/include$ tree -L 2
├─ hal
│   ├── aw-alsa-lib
│   ├── aw_common.h
│   ├── .....
│   ├── sunxi_hal_usb.h
│   ├── sunxi_hal_watchdog.h
│   └─ video
└─ osal
├─ hal_atomic.h
├─ hal_cache.h
├─ .....
├─ hal_waitqueue.h
└─ hal_workqueue.h
```

5. rtos Image Compile and Enable

Cortex-R

There are two ways to compile an rtos image:

1. By "buildroot" overall compilation, the generated "amp_rv0.bin" is directly packaged and generated in the image.
2. Compile "amp_rv0.bin" separately in the rtos directory, and then manually place the file into the file system.

amp_rv0.bin is an rtos image file.

This section describes the two compilation methods and their application scenarios.

5.1. buildroot builds overall

This method is suitable for making mass production cards for mass production, which can save production time. This method can be used when users have modified their Cortex-R image.

Note: Before using this method, it is necessary to build an image according to Chapter 4 of the "MYD-YT113X Linux Software Development Guide". This section takes the myir-image-yt113i-full image as an example to explain, and users can choose according to their actual situation.

5.1.1. Selection of overall platform options

First, enter the top-level directory of the source code and execute `./build.sh config`.

```
PC$ $HOME/T113$ ./build.sh config
```

```

=====ACTION List: mk_config ;=====
options :
All available platform:
  0. android
  1. linux
Choice [linux]: 1
All available linux_dev:
  0. bsp
  1. buildroot
  2. openwrt
Choice [buildroot]: 1
All available ic:
  0. t113_i
Choice [t113_i]: 0
All available board:
  0. myir-image-yt113i-core
  1. myir-image-yt113i-full
Choice [myir-image-yt113i-full]: 1
All available flash:
  0. default
  1. nor
Choice [default]: 0

```

5.1.2. Compiling an rtos image

Execute in the top-level directory of the source code "*./build.sh rtos*".

```

PC$ $HOME/T113$ ./build.sh rtos
=====ACTION List: build_rtos ;=====
options :
INFO: build rtos ...
Setup env done!
Run lunch_rtos to select project
last=t113_i_c906_evb1_auto

```

```

select=t113_i_c906_evb1_auto...
t113_i_c906/evb1_auto
'/media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/rtos/lichee/rtos/pr
jects/t113_i_c906/evb1_auto/defconfig' -> '/media/home/nico/MYIR/MYD-YT11
3X/TINA/20240622/T113/rtos/lichee/rtos/.config'
=====
RTOS_BUILD_TOP=/media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/
rtos
RTOS_TARGET_ARCH=riscv
RTOS_TARGET_CHIP=sun8iw20p1
RTOS_TARGET_DEVICE=t113_i_c906
RTOS_PROJECT_NAME=t113_i_c906_evb1_auto
=====
Run mrtos_menuconfig to config rtos
Run m or mrtos to build rtos
build rtos ...
Dark Builder
Version (1.6.0 - BiCEP2 (Gravitational Waves))
*[CC] [SCRIPT] build/t113_i_c906_evb1_auto/img/sys_config.fex
[CONF] [Tina-RT-Builder] .dbuild/./include/generated/t113_i_c906_evb1_auto/
autoconf.h
[LDS] [Linker] projects/t113_i_c906/evb1_auto/freertos.lds
CC build/t113_i_c906_evb1_auto/arch/common/version.o
LD build/t113_i_c906_evb1_auto/arch/common/obj-in.o
LD build/t113_i_c906_evb1_auto/arch/obj-in.o
[LD] [Tina-RT-Builder] build/t113_i_c906_evb1_auto/img/rt_system.elf
if [ -n /media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/rtos/lichee/rt
os/build/t113_i_c906_evb1_auto/img/ ]; then mkdir -p /media/home/nico/MYIR/
MYD-YT113X/TINA/20240622/T113/rtos/lichee/rtos/build/t113_i_c906_evb1_auto
/img; fi
Memory region      Used Size  Region Size  %age Used
RAM:      259736 B    6 MB    4.13%

```

```
*[IMAGE] [Tina-RT-Builder] build/t113_i_c906_evb1_auto/img/rt_system.bin
*[SYMS] [Tina-RT-Builder] build/t113_i_c906_evb1_auto/img/rt_system.syms
text data bss dec hex filename
178528 70176 11032 259736 3f698 build/t113_i_c906_evb1_auto/img/rt_syst
em.elf
copying /media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/rtos/lichee
/rtos/build/t113_i_c906_evb1_auto/img/rt_system.bin to /media/home/nico/MYIR
/MYD-YT113X/TINA/20240622/T113/rtos/board/t113_i_c906/evb1_auto/bin/freert
os.fex
#### make completed successfully
'/media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/rtos/lichee/rtos/bui
ld/t113_i_c906_evb1_auto/img/rt_system.bin' -> '/media/home/nico/MYIR/MYD-Y
T113X/TINA/20240622/T113/rtos/board/t113_i_c906/evb1_auto/bin/rtos_riscv_su
n8iw20p1.fex'
'/media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/rtos/lichee/rtos/bui
ld/t113_i_c906_evb1_auto/img/rt_system.elf' -> '/media/home/nico/MYIR/MYD-Y
T113X/TINA/20240622/T113/device/config/chips/t113_i/configs/myir-image-yt11
3i-full/bin/amp_rv0.bin'
```

Note: If you modify the source code or configuration, it is recommended to run `./build.shrtosclean` to clear the generated file of the last compilation, and then recompile it again

5.1.3. Packaging Cortex-R images

Finally, executing `./build.sh pack` in the top directory of the source code will pack the `amp_rv0.bin` (R kernel image) file into the image.

The user only needs to follow chapter 5 of *"MYD-YT113X Linux Software Development Guide"* to burn the image into the development board.

```
PC$ $HOME/T113$ ./build.sh pack
=====ACTION List: mk_pack ;=====
options :
INFO: packing firmware ...
```

```
INFO: /media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/out/t113_i/co
mmon/keys
copying tools file
copying configs file
copying product configs file
linux copying boardt&linux_kernel_version configs file
ls: cannot access '/media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/d
evice/config/chips/t113_i/configs/myir-image-yt113i-full/linux-5.4/env*': No such
file or directory
Use u-boot env file:

=====Partial packaging process omitted=====

Dragon execute image.cfg SUCCESS !
-----image is at-----

537M  /media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/out/myir-i
mage-yt113i-full.img
```

5.1.4. enable Cortex-R

Burn the successfully compiled image onto the development board and execute the following command in the Cortex-A serial port (UART5) to enable on the Cortex-R:

```
root@myd-yt113-i:~# echo start > /sys/class/remoteproc/remoteproc1/state
```

No printing information is described here, it is the same as the printing information in Chapter 3.

5.1.5. Cortex-A communicates with Cortex-R

After Cortex-R is enabled, the user can execute the MYiR compiled Cortex-A application to communicate with Cortex-R, and execute the following command in the Cortex-A serial port (UART5) :

```
root@myd-yt113-i:~# ./rpmsg_echo -r c906_rproc@0 -m myir_test -n 5
Sending message #0:myir_test
Receiving message #0:myir_test
Sending message #1:myir_test
Receiving message #1:myir_test
Sending message #2:myir_test
Receiving message #2:myir_test
Sending message #3:myir_test
Receiving message #3:myir_test
Sending message #4:myir_test
Receiving message #4:myir_test
Deleted rpmsg endpoint file: /dev/rpmsg1
```

5.2. Compile rtos separately

This method is applicable to the debugging stage, when the user modifies the source code or configuration of the Cortex-R, compile and generate the "*amp_rv0.bin*" image file separately, and then use the U disk or SCP and other methods to move the "*amp_rv0.bin*" to the development board to update the Cortex-R image separately, which can save the debugging time.

5.2.1. Loading Compilation Environment

First, enter the top-level directory of the rtos source code, and then execute the "*source envsetup.sh*" command.

```
PC$ $HOME/T113/rtos$ source envsetup.sh
Setup env done!
Run lunch_rtos to select project
```

5.2.2. Select Configuration

After loading the compilation environment, execute the "lunch_rtos" command and select the configuration for the corresponding model. For the MYD-YT113-I model development board, choose the corresponding configuration below.

```
PC$ $HOME/T113/rtos$ lunch_rtos
last=t113_i_c906_evb1_auto
```

You're building on Linux

Lunch menu... pick a combo:

1. t113_c906_evb1
2. t113_c906_evb1_auto
3. **t113_i_c906_evb1_auto**
4. t113_i_c906_evb1_auto_fastboot_video
5. t113_s3p_c906_evb1_auto
6. t113_s3p_c906_evb1_auto_fastboot
7. t113_s3p_c906_evb1_auto_fastboot_video
8. t113_s3p_c906_evb1_auto_non_os
9. t113_s3p_c906_example_demo
10. t113_s4_c906_evb1_auto
11. t113_s4_c906_evb1_auto_fastboot_video
12. t113_s4p_c906_evb1_auto

Which would you like? [Default t113_i_c906_evb1_auto]: **3**

select=3...

t113_i_c906/evb1_auto

```
'/media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/rtos/lichee/rtos/projects/t113_i_c906/evb1_auto/defconfig' -> '/media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/rtos/lichee/rtos/.config'
```

=====

```
RTOS_BUILD_TOP=/media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/rtos
```

```
RTOS_TARGET_ARCH=riscv
```

```
RTOS_TARGET_CHIP=sun8iw20p1
RTOS_TARGET_DEVICE=t113_i_c906
RTOS_PROJECT_NAME=t113_i_c906_evb1_auto
=====
Run mrtos_menuconfig to config rtos
Run m or mrtos to build rtos
```

5.2.3. Compiling the rtos image

After selecting the configuration, execute "*mrtos*" to start compiling rtos.

```
PC$ $HOME/T113/rtos$ mrtos
build rtos ...
Dark Builder
Version (1.6.0 - BiCEP2 (Gravitational Waves))
*[CC] [SCRIPT] build/t113_i_c906_evb1_auto/img/sys_config.fex
[CONF] [Tina-RT-Builder] .dbuild/./include/generated/t113_i_c906_evb1_auto/
autoconf.h
[LDS] [Linker] projects/t113_i_c906/evb1_auto/freertos.lds
CC build/t113_i_c906_evb1_auto/arch/common/version.o
LD build/t113_i_c906_evb1_auto/arch/common/obj-in.o
LD build/t113_i_c906_evb1_auto/arch/obj-in.o
[LD] [Tina-RT-Builder] build/t113_i_c906_evb1_auto/img/rt_system.elf
if [ -n /media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/rtos/lichee/rt
os/build/t113_i_c906_evb1_auto/img/ ]; then mkdir -p /media/home/nico/MYIR/
MYD-YT113X/TINA/20240622/T113/rtos/lichee/rtos/build/t113_i_c906_evb1_auto
/img; fi
Memory region      Used Size  Region Size  %age Used
      RAM:    260040 B    6 MB    4.13%
*[IMAGE] [Tina-RT-Builder] build/t113_i_c906_evb1_auto/img/rt_system.bin
*[SYMS] [Tina-RT-Builder] build/t113_i_c906_evb1_auto/img/rt_system.syms
text data bss dec hex filename
178832 70176 11032 260040 3f7c8 build/t113_i_c906_evb1_auto/img/rt_syst
em.elf
```

```

copying /media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/rtos/lichee
/rtos/build/t113_i_c906_evb1_auto/img/rt_system.bin to /media/home/nico/MYIR
/MYD-YT113X/TINA/20240622/T113/rtos/board/t113_i_c906/evb1_auto/bin/freert
os.fex
#### make completed successfully
myir-image-yt113i-full
'/media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/rtos/lichee/rtos/bui
ld/t113_i_c906_evb1_auto/img/rt_system.elf' -> '/media/home/nico/MYIR/MYD-Y
T113X/TINA/20240622/T113/rtos/./device/config/chips/t113_i/configs/myir-imag
e-yt113i-full/bin/amp_rv0.bin'
'/media/home/nico/MYIR/MYD-YT113X/TINA/20240622/T113/rtos/lichee/rtos/bui
ld/t113_i_c906_evb1_auto/img/rt_system.bin' -> '/media/home/nico/MYIR/MYD-Y
T113X/TINA/20240622/T113/rtos/board/t113_i_c906/evb1_auto/bin/rtos_riscv_su
n8iw20p1.fex'

```

5.2.4. Update the rtos image file

The "*amp_rv0.bin*" file compiled separately in the previous section needs to be replaced in the "*/lib/firmware*" directory on the development board, overwriting the old one. Users can upload to the development board through a USB flash drive or SCP, and the upload process is not described here.

Before updating an rtos image, if Cortex-R has been enabled, you need to disable it by running the following command:

```
root@myd-yt113-i:~# echo stop > /sys/class/remoteproc/remoteproc1/state
```

After closing, run the following command to update the rtos image.

```
root@myd-yt113-i:/lib/firmware# echo amp_rv0.bin > /sys/class/remoteproc/rem
oteproc1/firmware
```

Note: The *amp_rv0.bin* file must be placed in the "*/lib/firmware*" directory to update the rtos image, not in other directories.

5.2.5. enable Cortex-R

After updating the rtos image separately, run the following command in the Cortex-A serial port (UART5) to enable the Cortex-R:

```
root@myd-yt113-i:~# echo start > /sys/class/remoteproc/remoteproc1/state
```

No printing information is described here, it is the same as the printing information in Chapter 3.

5.2.6. Cortex-A communicates with Cortex-R

After Cortex-R is enabled, the user can execute the MYiR compiled Cortex-A application to communicate with Cortex-R, and execute the following command in the Cortex-A serial port (UART5) :

```
root@myd-yt113-i:~# ./rpmsg_echo -r c906_rproc@0 -m myir_test -n 5
Sending message #0:myir_test
Receiving message #0:myir_test
Sending message #1:myir_test
Receiving message #1:myir_test
Sending message #2:myir_test
Receiving message #2:myir_test
Sending message #3:myir_test
Receiving message #3:myir_test
Sending message #4:myir_test
Receiving message #4:myir_test
Deleted rpmsg endpoint file: /dev/rpmsg1
```

6. Cortex-A and Cortex-R source code

6.1. Cortex-A Source Code (ARM)

6.1.1. Obtaining Source Code

MYiR will provide the Cortex-A source code program, customers can get it from "04-Sources/RISC-V" directory. The Cortex-A program source code is as follows:

```
PC$ $HOME/RISC-V$ ls  
librmsg.c librmsg.h main.c Makefile rmsg.h
```

6.1.2. Compile source code

After obtaining the Cortex-A source code, customers need to put the source code into their own host environment, then load the compilation chain according to chapter 2.2.1 in the "MYD-YT113X Linux Software Development Guide", and then compile the program:

```
PC$ $HOME/RISC-V$ make  
arm-linux-gnueabi-gcc -D_GNU_SOURCE -I. -c main.c -o main.o  
arm-linux-gnueabi-gcc -D_GNU_SOURCE -I. -c librmsg.c -o librmsg.o  
arm-linux-gnueabi-gcc -D_GNU_SOURCE -I. -o rmsg_echo main.o librmsg.o
```

At this time to generate "rmsg_echo" executable program, the program is the application of the Cortex-A, the user can be used through the U disk or SCP and other methods of copying to the development board can be used.

```
PC$ $HOME/RISC-V$ ls  
librmsg.c librmsg.h librmsg.o main.c main.o Makefile rmsg_echo rmsg.  
h
```

6.1.3. Key Code Description

The main file in the Cortex-A source code is main.c. The key code of the program is explained below:

- Create rpmsg endpoint

```
ret = rpmsg_alloc_ept(ctrl_name, ept_name);
if (ret < 0) {
    printf("rpmsg_alloc_ept for ept %s (%s) failed\n", ept_name, ctrl_name);
    return -1;
}
```

- Open rpmsg endpoint

```
ept_id = ret;
snprintf(ept_file_path, sizeof(ept_file_path), "/dev/rpmsg%d", ept_id);
ept_fd = open(ept_file_path, O_RDWR);
if (ept_fd < 0) {
    printf("open %s failed\n", ept_file_path);
    return -1;
}
```

- Sending data

```
ret = write(ept_fd, send_buf, strlen((char *)send_buf) + 1);
if (ret < 0) {
    printf("Sending message failed: %s\n", strerror(errno));
    close(ept_fd);
    return -1;
}
else {
    printf("Sending message #%d:%s\n", i, send_buf);
}
fds[0].fd = ept_fd;
fds[0].events = POLLIN;
```

- Receive data

```
ret = poll(fds, 1, 1000);
if (ret < 0) {
```

```

break;
} else if (ret > 0 && (fds[0].revents & POLLIN)) {
    uint8_t rx_buf[RPMSG_DATA_MAX_LEN];
    ssize_t nread = read(ept_fd, rx_buf, sizeof(rx_buf));
    if (nread < 0) {
        printf("Receiving file error: %s\n", strerror(errno));
        break;
    } else if (nread > 0) {
        printf("Receiving message #d:%s\n", i, rx_buf);
    }
}
}

```

- Close the rpmsg node

```

ret = rpmsg_free_ept(ctrl_name, ept_id);
if (ret) {
    printf("rpmsg_free_ept for /dev/rpmsg%d (%s) failed\n", ept_id, ctrl_name);
    return -1;
} else {
    printf("Deleted rpmsg endpoint file: /dev/rpmsg%d\n", ept_id);
}
}

```

6.2. Cortex-R source code (RISC-V)

6.2.1. Get source code

MYiR will provide the Cortex-R source code program, the customer obtains the "04-Sources/MYD-YT113X-Distribution-L5.4.61-V2.0.0.tar.gz" SDK resource package, and then unpacks it into their own hosting environment. The directory of the Cortex-R source code program is as follows:

```

PC$ $HOME/T113/rtos/lichee/rtos/projects/t113_i_c906/evb1_auto/src$ ls
alsa_config.c assert.c card_default.c FreeRTOSConfig.h hooks.c main.c

```

6.2.2. Compile source code

After updating or modifying the source code, customers can refer to Chapter 5 to compile the rtos source code and generate "*amp_rv0.bin*" file.

6.2.3. Key Code Description

- rpmsg controller creation

```
void cpu0_app_entry(void *param)
{
    const char *name = "echo_demo";
    struct ept_parm *parm = NULL;
    /* initialize openamp */
    openamp_init();                openamp initialization
    /* Create ctrldev, Linux will generate the device /dev/rpmsg_ctrl-xxxxxx */
    rpmsg_ctrldev_create();        rpmsg controller creation
    printf("Wating message!\n");
    /* Create ept */
    rpmsg_client_bind(name, rpmsg_ept_callback, rpmsg_bind_cb,
        rpmsg_unbind_cb, 1, parm);    endpoint establishment

    vTaskDelete(NULL);
}
```

- rpmsg_ept_callback function

```
static int rpmsg_ept_callback(struct rpmsg_endpoint *ept, void *data,
    size_t len, uint32_t src, void *priv)
```

```
{
    struct ept_test_entry *eptdev = ept->priv;
```

Copy data to message queue

```
if (eptdev->cnt < RPMSG_TEST_RX_QUEUE) {
    memcpy(eptdev->rx_queue[eptdev->tail], data, len);
    eptdev->rx_len[eptdev->tail] = len;
    eptdev->tail = (eptdev->tail + 1) % RPMSG_TEST_RX_QUEUE;
```

```
eptdev->cnt++;
} else {
    printf("rpsmsg%d: rx queue is full\n", (int)eptdev->client->id);
}

return 0;
}
```

- The rpsmsg.find_cb function

```
static int rpsmsg_bind_cb(struct rpsmsg_ept_client *client)
{
    int ret;
    struct ept_test_entry *eptdev;

    printf("rpsmsg%d: binding\n", client->id);
```

Requesting Memory

```
eptdev = hal_malloc(sizeof(*eptdev));
if (!eptdev) {
    openamp_err("failed to alloc client entry\n");
    return -ENOMEM;
}
```

```
memset(eptdev, 0, sizeof(*eptdev));
eptdev->client = client;
client->ept->priv = eptdev;
```

Create and enable the rpsmsg_echo_thread thread

```
eptdev->echo_task = hal_thread_create(rpsmsg_echo_thread, eptdev, client->name,
                                     1024, configMAX_PRIORITIES - 4);
```

```

if (!eptdev->echo_task) {
    printf("Failed to create %s echo_task\n", client->name);
    ret = -ENOMEM;
    goto free_eptdev;
}
hal_thread_start(eptdev->echo_task);
return 0;

free_eptdev:
    hal_free(eptdev);
return ret;
}

```

- **rpmsg_unbind_cb function**

```

static int rpmsg_unbind_cb(struct rpmsg_ept_client *client)
{
    struct ept_test_entry *eptdev = client->ept->priv;

    printf("rpmsg%d: unbinding\n", client->id);
    eptdev->stop = 1;
    hal_free(eptdev);
    return 0;
}

```

Appendix A

Warranty & Technical Support Services

MYIR Electronics Limited is a global provider of ARM hardware and software tools, design solutions for embedded applications. We support our customers in a wide range of services to accelerate your time to market.

MYIR is an ARM Connected Community Member and work closely with ARM and many semiconductor vendors. We sell products ranging from board level products such as development boards, single board computers and CPU modules to help with your evaluation, prototype, and system integration or creating your own applications. Our products are used widely in industrial control, medical devices, consumer electronic, telecommunication systems, Human Machine Interface (HMI) and more other embedded applications. MYIR has an experienced team and provides custom design services based on ARM processors to help customers make your idea a reality.

The contents below introduce to customers the warranty and technical support services provided by MYIR as well as the matters needing attention in using MYIR's products.

Service Guarantee

MYIR regards the product quality as the life of an enterprise. We strictly check and control the core board design, the procurement of components, production control, product testing, packaging, shipping and other aspects and strive to provide products with best quality to customers. We believe that only quality products and excellent services can ensure the long-term cooperation and mutual benefit.

Price

MYIR insists on providing customers with the most valuable products. We do not pursue excess profits which we think only for short-time cooperation. Instead, we hope to establish long-term cooperation and win-win business with customers. So we will offer reasonable prices in the hope of making the business greater with the customers together hand in hand.

Delivery Time

MYIR will always keep a certain stock for its regular products. If your order quantity is less than the amount of inventory, the delivery time would be within three days; if your order quantity is greater than the number of inventory, the delivery time would be always four to six weeks. If for any urgent delivery, we can negotiate with customer and try to supply the goods in advance.

Technical Support

MYIR has a professional technical support team. Customer can contact us by email (support@myirtech.com), we will try to reply you within 48 hours. For mass production and customized products, we will specify person to follow the case and ensure the smooth production.

After-sale Service

MYIR offers one year free technical support and after-sales maintenance service from the purchase date.

The service covers:

Technical support service

MYIR offers technical support for the hardware and software materials which have provided to customers;

- To help customers compile and run the source code we offer;
- To help customers solve problems occurred during operations if users follow the user manual documents;
- To judge whether the failure exists;
- To provide free software upgrading service.

However, the following situations are not included in the scope of our free technical support service:

- Hardware or software problems occurred during customers' own development;
- Problems occurred when customers compile or run the OS which is tailored by themselves;
- Problems occurred during customers' own applications development;
- Problems occurred during the modification of MYIR's software source code.

After-sales maintenance service

The products except LCD, which are not used properly, will take the twelve months free maintenance service since the purchase date. But following situations are not included in the scope of our free maintenance service:

- The warranty period is expired;
- The customer cannot provide proof-of-purchase or the product has no serial number;
- The customer has not followed the instruction of the manual which has caused the damage the product;
- Due to the natural disasters (unexpected matters), or natural attrition of the components, or unexpected matters leads the defects of appearance/function;
- Due to the power supply, bump, leaking of the roof, pets, moist, impurities into the boards, all those reasons which have caused the damage of the products or defects of appearance;
- Due to unauthorized weld or dismantle parts or repair the products which has caused the damage of the products or defects of appearance;

- Due to unauthorized installation of the software, system or incorrect configuration or computer virus which has caused the damage of products.

Warm tips

1. MYIR does not supply maintenance service to LCD. We suggest the customer first check the LCD when receiving the goods. In case the LCD cannot run or no display, customer should contact MYIR within 7 business days from the moment get the goods.
2. Please do not use finger nails or hard sharp object to touch the surface of the LCD.
3. MYIR suggests user purchasing a piece of special wiper to wipe the LCD after long time use, please avoid clean the surface with fingers or hands to leave fingerprint.
4. Do not clean the surface of the screen with chemicals.
5. Please read through the product user manual before you using MYIR's products.
6. For any maintenance service, customers should communicate with MYIR to confirm the issue first. MYIR's support team will judge the failure to see if the goods need to be returned for repair service, we will issue you RMA number for return maintenance service after confirmation.

Maintenance period and charges

- MYIR will test the products within three days after receipt of the returned goods and inform customer the testing result. Then we will arrange shipment within one week for the repaired goods to the customer. For any special failure, we will negotiate with customers to confirm the maintenance period.
- For products within warranty period and caused by quality problem, MYIR offers free maintenance service; for products within warranty period but out of free maintenance service scope, MYIR provides maintenance service but shall charge some basic material cost; for products out of warranty period, MYIR provides maintenance service but shall charge some basic material cost and handling fee.

Shipping cost

During the warranty period, the shipping cost which delivered to MYIR should be responsible by user; MYIR will pay for the return shipping cost to users when the product is repaired. If the warranty period is expired, all the shipping cost will be responsible by users.

Products Life Cycle

MYIR will always select mainstream chips for our design, thus to ensure at least ten years continuous supply; if meeting some main chip stopping production, we will inform customers in time and assist customers with products updating and upgrading.

Value-added Services

1. MYIR provides services of driver development base on MYIR's products, like serial port, USB, Ethernet, LCD, etc.
2. MYIR provides the services of OS porting, BSP drivers' development, API software development, etc.
3. MYIR provides other products supporting services like power adapter, LCD panel, etc.
4. ODM/OEM services.

MYIR Electronics Limited

Room 04, 6th Floor, Building No.2, Fada Road,
Yunli Intelligent Park, Bantian, Longgang District.

Support Email: support@myirtech.com

Sales Email: sales.en@myirtech.com

Phone: +86-755-22984836

Fax: +86-755-25532724

Website: www.myirtech.com