User's Manual

RZ/G2 Group

Linux BSP Porting Guide

Renesas MPU
RZ Family
RZ/G Series

Renesas Electronics
www.renesas.com

Rev.1.10 May. 31, 2022

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.

How to Use This Manual

1. Purpose and Target Readers

This manual describes how users can port the Linux BSP for the RZ/G2 reference boards to boards (henceforth referred to as "custom boards") which they are developing with the use of the RZ/G2 Group MPU.

For more information on how to access the sources and how to build, see the manuals provided for each distribution method of Linux BSP.

- Via GitHub: "RZ/G2 Yocto recipe Start-Up Guide"
- Verified Linux Package for 64bit kernel (hereinafter called as VLP64): "Release Note" provided with VLP64

This manual is intended for users who intend to develop their own custom boards.

Statements in this manual apply to the following version of the BSP.

MPU	Reference Board	BSP Version
RZ/G2E	Silicon Linux EK874 Evaluation Kit (EK874) Revision E	BSP-1.0.7 or later (later version is recommended)
RZ/G2H	Hoperun Technology HiHope RZ/G2H platform	BSP-1.0.4 or later (later version is recommended)
RZ/G2M v1.3	Hoperun Technology HiHope RZ/G2M platform	BSP-1.0.1 or later (later version is recommended)
RZ/G2M v3.0	Hoperun Technology HiHope RZ/G2M platform	BSP-1.0.5 or later (later version is recommended)
RZ/G2N	Hoperun Technology HiHope RZ/G2N platform	BSP-1.0.2 or later (later version is recommended)
RZ/G2E	Silicon Linux EK874 Evaluation Kit (EK874)	BSP-1.0.0 or later (later version is recommended)

This manual includes references to the documents listed in the table below.

No.	Reference Content	Document Title	Number or Web Site for Obtaining the Document
[1]	Descriptions of how to handle the GitHub based development environment	RZ/G2 Yocto recipe Start-Up Guide	R01US0398EJ
[2]	Descriptions of how to handle the VLP based development environment, how to write TF-A to the reference boards	RZ/G Verified Linux Package for 64bit kernel Release Note	R01TU0277EJ
[3]	Specifications of the device tree	Devicetree Specification 0.2	https://github.com/devicetree-org/devicetree-specification/releases/download/v0.2/devicetree-specification-v0.2.pdf
[4]	Pin configurations	RZ/G Series, 2nd Generation User's Manual: Hardware	R01UH0808EJ
[5]	How to write TF-A to the reference boards	RZ/G2 Reference Boards Start-up Guide	R01TU0279EJ
[6]	How to port the U-Boot to the custom board	README of Das U-Boot v2021.10	https://github.com/renesas-rz/renesas-u-boot-cip/blob/v2021.10/rzg2/README

Particular attention should be paid to the precautionary notes when using the manual. These notes occur within the body of the text, at the end of each section.

The revision history summarizes the locations of revisions and additions. It does not list all revisions. Refer to the text of the manual for details.

2. List of Abbreviations, Acronyms and Keywords

Word	Full Form	Description
BSP	Board Support Package	A set of software components that allows a given OS to run on a specific hardware platform.
Linux BSP	Linux Board Support Package	Board support package to run the Linux environment
U-Boot	Das U-Boot	This is a boot loader, also referred to as the Universal Boot Loader, which is released under the GPL. “Universal” indicates that it can run on multiple types of platform.
VLP64	RZ/G Verified Linux Package for 64bit kernel	The software packages provided by Renesas for the users to evaluate/develop the GNU/Linux environment on the RZ/G2 Group processors.
TF-A	Trusted Firmware-A	-
ATF	Arm Trusted Firmware-A	-
ATF-A	Arm Trusted Firmware-A	-
EK874	-	Silicon Linux RZ/G2E evaluation kit
TFTP	Trivial File Transfer Protocol	-
PRR	Product Register	PRR indicates the product version and which of the Arm Cortex cores is present.
BL1	Boot Loader stage 1	-
BL2	Boot Loader stage 2	-
BL31	Boot Loader stage 3-1	-
BL33	Boot Loader stage 3-2	-
custom board	-	The board which will be based on RZ/G2 reference boards and includes the user’s customization

3. Notations Used in This Manual

`Constant width text` means the code fragment or the variable, function, environment variable or keyword in the body text.

`Constant width bold text` means the commands to be executed.

Green highlighter in the source code fragment shows the part to be focused on.

Characters with border in the source code fragment are the part to be referred from the body text.

The meaning of the other notation in the source code fragment is shown near each fragment.

Contents

1. Overview

This manual describes how users can port the Linux BSP from the RZ/G2 Linux platform that supports RZ/G2 reference boards to custom boards they are developing. Linux BSP for the RZ/G2 reference boards are provided as a part of the Yocto recipe on the GitHub repository or "RZ/G Verified Linux Package for 64bit kernel" (hereinafter, referred to as VLP64). When the users make their own custom boards based on the Linux BSP on the RZ/G2 reference boards, the Linux BSP are to be updated. Some parts in this manual describes how to port the Linux BSP for the Silicon Linux RZ/G2E evaluation kit (EK874) to a custom board as an example.

1.1 Porting Parts Explained in this Document

As shown in Figure 1.1, this document explains about following software components:

- Trusted Firmware-A (refer to the section 3. Porting Trusted Firmware-A)
- Das U-Boot (refer to the section 4. Porting U-Boot)
- Linux kernel (refer to the section 5. Porting the Linux Kernel)

- Also, how to add new machine configuration to Yocto recipe is explained in the section 0.

Adding new machine configuration to Yocto recipes. This enables the build configuration for the custom boards to both of Das U-boot and Linux kernel in Yocto build.

This document doesn't explain about Loader or user-land/root file system. To port user-land/root file system, please refer "Yocto Start-up Guide" document.
Also, this document assumes the boot process is QSPI flash boot.

This document assumes the boot procedure using SPI flash as a boot device.

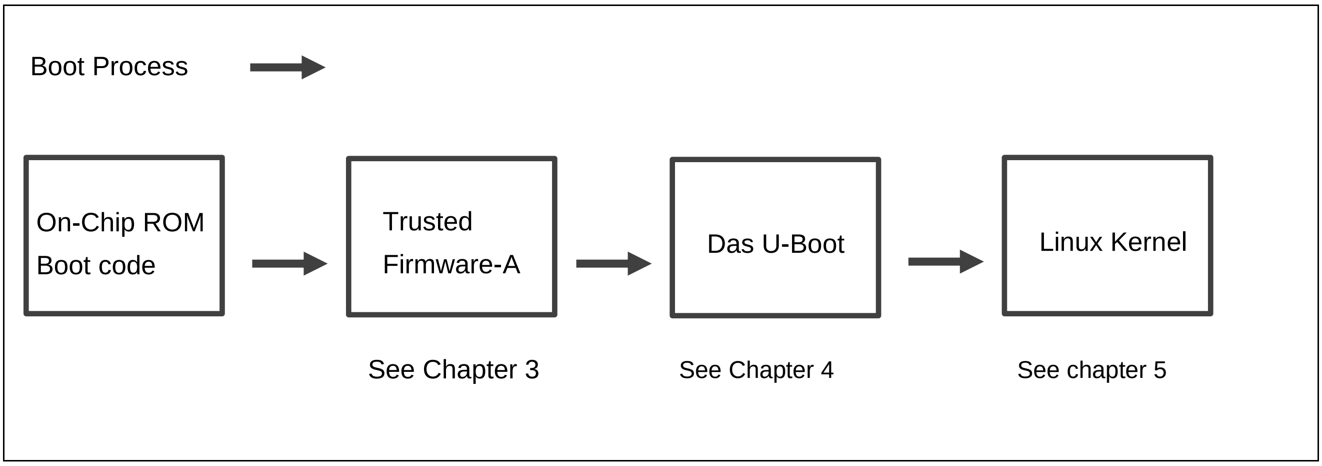

Figure 1.1 Porting Parts Explained in this Document

1.2 Prerequisites

This document assumes the prerequisites on Table 1.1.

Table 1.1 Prerequisites

Item	description
Linux BSP	Version 1.0.0 or later [VLP64 based development] tag: BSP-1.0.0 or later [Yocto recipe-based development]
Reference board	Hoperun Technology HiHope RZ/G2H platform [RZ/G2H] Hoperun Technology HiHope RZ/G2M platform [RZ/G2M v1.3, RZ/G2M v3.0] Hoperun Technology HiHope RZ/G2N platform [RZ/G2N] Silicon Linux RZ/G2E evaluation kit (EK874) [RZ/G2E]
Development environments	Listed on “2. Build environment” of VLP64’s release notes or “2. Environmental Requirement” of “RZ/G2 Yocto recipe Start-Up Guide” [1]

Supported boards are depend on the Linux BSP version which provided by Yocto recipe on GitHub or in VLP64 (Table 1.2). Later version is recommended when multiple versions support a board.

Table 1.2 Supported boards by each Linux BSP version

Version	Silicon Linux EK874 Evaluation Kit (EK874)	Hoperun Technology HiHope RZ/G2M platform (RZ/G2M v1.3)	Hoperun Technology HiHope RZ/G2M platform (RZ/G2M v3.0)	Hoperun Technology HiHope RZ/G2N platform	Hoperun Technology HiHope RZ/G2H platform
1.0.0	X				
1.0.1	X	X			
1.0.2	X	X		X	
1.0.3-RT	X	X		X	
1.0.4	X	X		X	X
1.0.5-RT 1.0.6 …	X	X	X	X	X

VLP64 are available on the site below. Please create an account to download the packages. Basic packages of VLP64 can be downloaded.

- Non-RT: https://www.renesas.com/products/microcontrollers-microprocessors/rz-cortex-a-mpus/rzg-linux-platform/rzg-marketplace/verified-linux-package/rzg2-vlp-eva
- RT: https://www.renesas.com/products/microcontrollers-microprocessors/rz-cortex-a-mpus/rzg-linux-platform/rzg-marketplace/verified-linux-package/rzg2-vlp-eva-rt

Yocto recipe are available on the following GitHub repository (also the related repositories described on README.md of this repository):

- https://github.com/renesas-rz/meta-rzg2

1.3 Overview of the boot sequence

The boot process of the Linux BSP for RZ/G2 are composed by the following sequences (shown on Figure 1.2).

1. Boot Loader stage 1 (BL1): Boot ROM Program loads Trusted Boot Firmware of Trusted Firmware-A and execute it.
2. Boot Loader stage 2 (BL2): Trusted Boot Firmware loads EL3 Runtime Software and U-Boot to the DRAM. Trusted Boot Firmware executes EL3 Runtime Software.
3. Boot Loader stage 3-1 (BL31): EL3 Runtime Software executes U-Boot.
4. Boot Loader stage 3-3 (BL33) U-Boot loads kernel image and devicetree and execute kernel image.

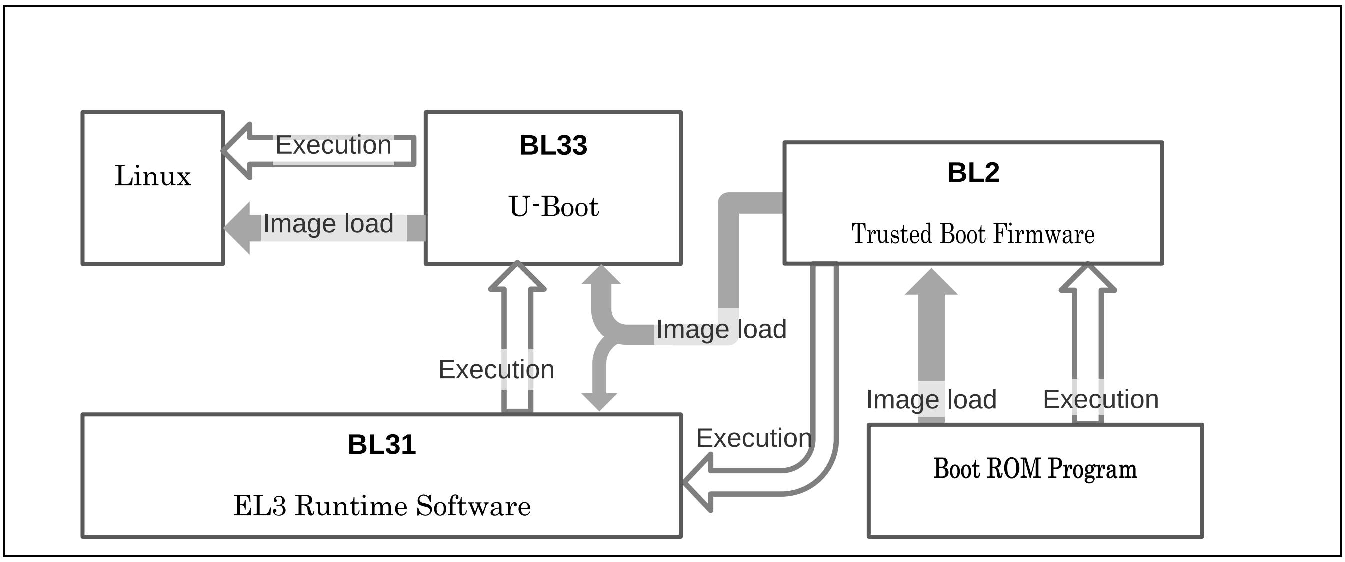

Figure 1.2 Overview of VLP64 boot sequence

2. Adding new machine configuration to Yocto recipes

Linux BSP for RZ/G2 are provided as a Yocto recipe and the machine configuration manages the MPU and the board information used by the U-boot, the Linux kernel and the other software in common. This chapter is optional for porting but recommended for easier board management.

2.1 Create machine config file

First create a machine config file in Renesas Yocto layer `meta-rzg2`, for example `meta-rzg2/conf/machine/custom-rzg2e.conf`. In this case, the machine name is `custom-rzg2e`. The typical machine name is composed only by lowercase letters: a-z, numbers: 0-9 and hyphen. Machine name example can be found in the conf/machine directories of the Yocto Metadata layers available on https://git.yoctoproject.org/cgit/cgit.cgi.

Refer to Renesas reference files at:

- ek874: meta-rzg2/conf/machine/ek874.conf
- hihope-rzg2m: meta-rzg2/conf/machine/hihope-rzg2m.conf
- hihope-rzg2n: meta-rzg2/conf/machine/hihope-rzg2n.conf
- hihope-rzg2h: meta-rzg2/conf/machine/hihope-rzg2h.conf

In this file, there are some noticeable settings as below example (taken from ek874.conf)

Symbol	Example value	Description
SOC_FAMILY	r8a774c0	Part name of RZ/G2E. Change this if another MPU is used. Note that list of available MPU can be found as .inc files under meta-rzg2/conf/machine/include/ (Ex: r8a774c0.inc, r8a774a1.inc…)
DEFAULTTUNE	cortexa53	Tune setting when building. In this case, choose this value because RZ/G2E r8a774c0 is based on ARM Cortex-A53. Change this corresponding to SOC_FAMILY.
SERIAL_CONSOLE	"115200 ttySC0"	Setting for using debug serial console (in this case EK874 use ttySC0 channel for debug serial, with baud rate 115200). Change this corresponding to hardware setting of custom board.
PREFERRED_PROVIDER_virtual/kernel	linux-renesas	Use package "linux-renesas" package for Linux kernel (shouldn't change)
KERNEL_DEVICETREE	"renesas/r8a774c0-ek874.dtb renesas/r8a774c0-cat874.dtb"	List of device tree in Linux kernel that will be used for this machine. Change this to the device tree of custom board created in 5.5
PREFERRED_VERSION_u-boot	v2021.10%	Version of u-boot to be build (shouldn't change)
UBOOT_CONFIG	ek874	Set defconfig for building u-boot. Change this to the defconfig of custom board created in 4.3.6
UBOOT_CONFIG[ek874]	silinux_ek874_defconfig	

2.2 Add compatible machine

Next, the newly created machine is needed to be added as compatible to some packages to allow building.

meta-rzg2/recipes-kernel/linux/linux-renesas_4.19.bb
recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_git.bb

```
COMPATIBLE_MACHINE = "ek874|hihope-rzg2m|hihope-rzg2n|hihope-rzg2h|custom-rzg2e"
```

If the multimedia features like OpenGL support or window system support is to be supported, need to add to some multimedia support packages as well.
(At default, the multimedia features are available in core-image-weston, core-image-bsp, core-image-hmi, and not available in core-image-minimal, core-image-bsp)

meta-rzg2/recipes-kernel/kernel-module-gles/kernel-module-gles.bb
meta-rzg2/recipes-graphics/wayland/wayland-kms_1.6.0.bb
meta-rzg2/recipes-graphics/wayland/libgbm.bb
meta-rzg2/recipes-graphics/gles-module/gles-user-module.bb

```
COMPATIBLE_MACHINE = "ek874|hihope-rzg2m|hihope-rzg2n|hihope-rzg2h|custom-rzg2e"
```

2.3 Update build configuration

Certain packages require specific build configurations depend on the MPU and the board. For MPU dependences, Yocto can configure automatically if SOC_FAMILY is the same with the reference boards, therefore following packages can work without any modification:

meta-rzg2/recipes-kernel/kernel-module-gles/kernel-module-gles.bb

```
SRC_URI_r8a774e1 = 'file://GSX_KM_H3.tar.bz2'
SRC_URI_r8a774b1 = 'file://GSX_KM_M3N.tar.bz2'
SRC_URI_r8a774a1 = 'file://GSX_KM_M3.tar.bz2'
SRC_URI_r8a774c0 = 'file://GSX_KM_E3.tar.bz2'
```

meta-rzg2/recipes-graphics/gles-module/gles-user-module.bb

```
SRC_URI_r8a774b1 = "file://r8a77965_linux_gsx_binaries_gles.tar.bz2"
SRC_URI_r8a774a1 = "file://r8a77960_linux_gsx_binaries_gles.tar.bz2"
SRC_URI_r8a774c0 = "file://r8a77990_linux_gsx_binaries_gles.tar.bz2"
SRC_URI_r8a774e1 = "file://r8a77951_linux_gsx_binaries_gles.tar.bz2"
```

But there is one package that depends on the board hardware and must be modified manually. *But it can be skipped if multimedia features are not used when the target image is core-image-minimal or core-image-bsp.*

meta-rzg2/recipes-kernel/kernel-module-mmngr/kernel-module-mmngr.bb

```
MMNGR_CFG_ek874 = "MMNGR_EBISU"
MMNGR_CFG_hihope-rzg2m = "MMNGR_SALVATORX"
MMNGR_CFG_hihope-rzg2n = "MMNGR_SALVATORX"
MMNGR_CFG_hihope-rzg2h = "MMNGR_SALVATORX"
```

MMNGR module depends on the RAM setting of the board. With different value of macro `MMNGR_CFG` it will have different set of definitions (for different address and memory size). If custom board has same memory setting with one reference boards from Renesas, set the same value.

```
MMNGR_CFG_custom-rzg2e = "MMNGR_EBISU"
```

If custom board has completely different RAM setting with all Renesas reference boards, need to create a new setting for it. This involves modifying source code of MMNGR. If it is too difficult, please contact to Renesas Sales.

3. Porting Trusted Firmware-A

Linux BSP for RZ/G2 uses Trusted Firmware-A (originally, known as Arm Trusted Firmware-A. hereinafter, referred to as TF-A) as a loader for BL2 and BL31 as shown on 0

Overview of the boot sequence.

3.1 Customization flow

Followings are one of example customization flow of TF-A provided with VLP64. This flow is based on one shown in https://wiki.yoctoproject.org/wiki/TipsAndTricks/Patching_the_source_for_a_recipe.

1. Setup the bitbake build environment. The directory `/home/user/work` is just an example and specify your existing Yocto build environment's work directory.
   - **`export WORK=/home/user/user_work`**
   - **`source poky/oe-init-build-env`**
2. (Optional) Making the layer for the customization. This step is required only when the modification for porting will be stored on the new Yocto layer (other than meta-rzg2). For the details of the following command, please refer to https://www.yoctoproject.org/docs/2.4.3/dev-manual/dev-manual.html#creating-a-general-layer-using-the-bitbake-layers-script.
   Note that the layer name "meta-userboard" is just an example and specify your custom boards' layer name.
   - **`bitbake-layers create-layer $WORK/meta-userboard`**
   - **`bitbake-layers add-layer $WORK/meta-userboard`**
3. Confirm the recipe or bbappend of the recipe has the appropriate `COMPATIBLE_MACHINE` (in this example "custom-rzg2e") as shown in 2.2 Add compatible machine.
4. Get TF-A source code. After the following commands, the source code will be available on the directory `$WORK/build/workspace/sources/arm-trusted-firmware` able.
   - **`devtool modify arm-trusted-firmware`**
5. Modify the source code with reference to “3.2 How to customize a functionality in TF-A”. **`bitbake arm-trusted-firmware -c devshell`** is easy to access the source code (not mandatory)
6. Modify the recipe `$WORK/meta-rzg2/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_git.bb` if requred. **`devtool edit-recipe arm-trusted-firmware -a`** is easy to access the recipe (not mandatory).
7. Build and test TF-A to confirm that the modification is correct and works well. Build results will be available on `$WORK/build/tmp/deploy/images/`*`[machine name]`*. Regarding how to write TF-A to the board, refer to the section “1.3Writing Bootloader” of “RZ/G2 Reference Boards Start-up Guide” [5] provided with VLP64 or the section “4. Writing of IPL/Secure” of “Yocto recipe Start-Up Guide” [1].
   - **`bitbake arm-trusted-firmware`**
8. Repeat from step 4 to get the TF-A for the custom board.
9. Commit the changes to the source code in the source code directory. The number of commits will be the same as the number of patch files created in the step 8.
   - **`cd $WORK/build/workspace/sources/arm-trusted-firmware`**
   - **`git add .`**
   - **`git commit`**
10. Convert the changes of the source code to the recipe. There are two ways:
    - **`devtool update-recipe arm-trusted firmware`** to update the original `arm-trusted-firmware`’s recipe on `$WORK/meta-rzg2/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_git.bb`
    - **`devtool update-recipe -a $WORK/meta-userboard arm-trusted-firmware`** to convert the changes to `.bbappend` file on the different layer like `$WORK/meta-userboard/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_git.bbappend` and patches on

$WORK/meta-userboard/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware/. Note that layer name "meta-userboard" are the same as the one used in step 2.

11. If the customization is finished, run the following command to remove the source code directory $WORK/build/workspace/sources/arm-trusted-firmware and recover the normal build method using $WORK/build/tmp/work/...
   — **devtool reset arm-trusted-firmware**

3.2 How to customize a functionality in TF-A

3.2.1 Board specific flag

From TF-A v2.5, code support for RZ/G2 will be divided into 2 types:

- Common code: code is used for both RZ/G2 and RCar Gen 3. Some built macros will have “RCAR” in their name and use “common” folder to store the source code.
- Specific code: code is used for only RZ/G2 and have “RZG” in name of build macros. Also code will be stored in “rzg” folder.

VLP64's TF-A can be applied the board specific configuration by using the build argument in ATFW_OPT defined in arm-trusted-firmware_git.bb.

$WORK/meta-rzg2/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_git.bb:

```
... (snip) ...
COMPATIBLE_MACHINE = "(ek874|hihope-rzg2m|hihope-rzg2n|hihope-rzg2h)"
PLATFORM = "rzg"
ATFW_OPT_LOSSY = "${@base_conditional("USE_MULTIMEDIA", "1", "RCAR_LOSSY_ENABLE=1", "",
d)}"
ATFW_OPT_r8a774c0 = "LSI=G2E RCAR_SA0_SIZE=0 RCAR_DRAM_DDR3L_MEMCONF=1
RCAR_DRAM_DDR3L_MEMDUAL=1 SPD="none""
ATFW_OPT_r8a774a1 = "LSI=G2M RCAR_DRAM_SPLIT=2 SPD="none""
ATFW_OPT_r8a774b1 = "LSI=G2N SPD="none""
ATFW_OPT_r8a774e1 = "LSI=G2H RCAR_DRAM_SPLIT=2 RCAR_DRAM_LPDDR4_MEMCONF=1
RCAR_DRAM_CHANNEL=5 SPD="none""
... (snip) ...
```

3.2.2 PFC/GPIO setting

VLP64's TF-A have PFC/GPIO settings for each device's reference board. The customization of PFC/GPIO settings based on the difference between the reference board and the custom board.

Following files are related to PFC/GPIO setting of RZ/G2 devices.

```
drivers/renesas/rzg/pfc
├── pfc_init.c
├── pfc.mk
├── pfc_regs.h
├── G2E
│   ├── pfc_init_g2e.c
│   └── pfc_init_g2e.h
├── G2H
│   ├── pfc_init_g2h.c
│   └── pfc_init_g2h.h
├── G2M
│   ├── pfc_init_g2m.c
│   └── pfc_init_g2m.h
├── G2N
│   ├── pfc_init_g2n.c
│   └── pfc_init_g2n.h
```

pfc_init() calls each device's PFC/GPIO initialization function depends on the value of Product Register (PRR) of RZ/G2 group. Table 3.1 shows each device's PFC/GPIO initialization function.

Table 3.1 File and Function list of PFC/GPIO setting

Product	Filename	Function
RZ/G2H	pfc_init_g2h.c	pfc_init_g2h()
RZ/G2M (v1.3, v3.0)	pfc_init_g2m.c	pfc_init_g2m()
RZ/G2N	pfc_init_g2n.c	pfc_init_g2n()
RZ/G2E	pfc_init_g2e.c	pfc_init_g2e()

3.2.2.1 Multiplexed pin functions

Some groups of pins of RZ/G2 Group devices have selectable two or more functions. "Module Select Register 0" (MOD_SEL0) to "Module Select Register n" (MOD_SELn) (n=2 [RZG2H], [RZG2M], [RZ/G2N]; n=1 [RZ/G2E]) can be configured as follows:

```
        /* initialize module select */
        pfc_reg_write(PFC_MOD_SEL0, MOD_SEL0_FOO
                                  | MOD_SEL0_BAR);
        pfc_reg_write(PFC_MOD_SEL1, MOD_SEL1_FOO
                                  | MOD_SEL1_BAR);
```

Macro definitions for MOD_SEL0 and MOD_SEL1 register values are defined in the files listed on Table 3.1 as follows:

```
...
#define MOD_SEL0_HSCIF0_A       ((uint32_t)0U << 24U)
#define MOD_SEL0_HSCIF0_B       ((uint32_t)1U << 24U)
#define MOD_SEL0_HSCIF1_A       ((uint32_t)0U << 23U)
#define MOD_SEL0_HSCIF1_B       ((uint32_t)1U << 23U)
#define MOD_SEL0_HSCIF2_A       ((uint32_t)0U << 22U)
#define MOD_SEL0_HSCIF2_B       ((uint32_t)1U << 22U)
...
```

Refer to "8.2.9 Module Select Register 0-2 (MOD_SEL0-2)" [RZG2H], [RZ/G2M] [RZ/G2N] or "9.2.9 Module Select Register 0-1 (MOD_SEL0-1)" [RZ/G2E] of "RZ/G Series, 2nd Generation User’s Manual: Hardware" [4] for the meaning of each bit.

Some pins of RZ/G2 Group devices have selectable two or more functions. Peripheral Function Select Register 0 (IPSR0) to Peripheral Function Select Register 15 (IPSR15) can be configured as follows:

```
        /* initialize peripheral function select */
        pfc_reg_write(PFC_IPSR0, IPSR_28_FUNC(2) /* select 2nd function for IP0[31:28] */
                              ...
                              | IPSR_4_FUNC(1)); /* select 1st function for IP0[7:4] */
                              | IPSR_0_FUNC(0)); /* select 0th function for IP0[3:0] */
        pfc_reg_write(PFC_IPSR1, IPSR_28_FUNC(2) /* select 2nd function for IP1[31:28] */
                              ...
                              | IPSR_4_FUNC(1)); /* select 1st function for IP1[7:4] */
                              | IPSR_0_FUNC(0)); /* select 0th function for IP1[3:0] */
```

Macro definitions "IPSR_n_FUNC" are defined in the files listed on Table 3.1 as follows:

```
...

#define IPSR_28_FUNC(x)        ((uint32_t)(x) << 28U)
#define IPSR_24_FUNC(x)        ((uint32_t)(x) << 24U)
#define IPSR_20_FUNC(x)        ((uint32_t)(x) << 20U)
#define IPSR_16_FUNC(x)        ((uint32_t)(x) << 16U)
#define IPSR_12_FUNC(x)        ((uint32_t)(x) << 12U)
#define IPSR_8_FUNC(x)         ((uint32_t)(x) << 8U)
#define IPSR_4_FUNC(x)         ((uint32_t)(x) << 4U)
#define IPSR_0_FUNC(x)         ((uint32_t)(x) << 0U)

...
```

Refer to "9.2.3 Peripheral Function Select Register 0-18 (IPSR0-18)" [RZG2H], [RZ/G2M] [RZ/G2N] or "9.2.3 Peripheral Function Select Register 0-15 (IPSR0-15)" [RZ/G2E] of "RZ/G Series, 2nd Generation User's Manual: Hardware" [4] for the meaning of each bit.

3.2.2.2 GPIO/Peripheral Function Select

Some pins of RZ/G2 Group devices can be configured as GPIO or peripheral function. GPIO/Peripheral Function Select Register 0 (GPSR0) to GPIO/Peripheral Function Select Register 6 (GPSR6) can be configured as follows:

```
        /* initialize GPIO/peripheral function select */
        pfc_reg_write(PFC_GPSR0, GPSR0_FOO
                                | GPSR0_BAR);
```

Macro definitions for the registers GPSR0 to GPSR6 are defined in the files listed on Table 3.1 as follows:

```
...
#define GPSR0_SDA4            ((uint32_t)1U << 17U)
#define GPSR0_SCL4            ((uint32_t)1U << 16U)
#define GPSR0_D15             ((uint32_t)1U << 15U)
#define GPSR0_D14             ((uint32_t)1U << 14U)
#define GPSR0_D13             ((uint32_t)1U << 13U)
...
```

Refer to "8.2.2 GPIO/Peripheral Function Select Register 0-7 (GPSR0-7)" [RZG2H], [RZ/G2M] [RZ/G2N] or "9.2.2 GPIO/Peripheral Function Select Register 0-6 (GPSR0-6)" [RZ/G2E] of "RZ/G Series, 2nd Generation User's Manual: Hardware" [4] for the meaning of each bit.

3.2.2.3 Drive voltage select

SD Card/MMC Interfaces and EthernetAVB-IF have multiple IO voltage level.

POC Control Register 0 (POCCTRL0, named as POCCTRL0in the source code) controls IO voltage level of SD Card/MMC Interfaces. Note that the settings on this register must match the voltage of VDDQ_MMC.

```
        /* initialize POC control */
        reg = mmio_read_32(POCCTRL0_MASK);
        reg = ((reg & POCCTRL0_MASK) | POC_SD1_DAT3_33V
                                | POC_SD1_DAT2_33V
                                | POC_SD1_DAT1_33V
                                | POC_SD1_DAT0_33V
                                | POC_SD1_CMD_33V
                                | POC_SD1_CLK_33V
                                | POC_SD0_DAT3_33V
                                | POC_SD0_DAT2_33V
                                | POC_SD0_DAT1_33V
                                | POC_SD0_DAT0_33V
                                | POC_SD0_CMD_33V
                                | POC_SD0_CLK_33V);
        pfc_reg_write(PFC_POCCTRL0, reg);
```

Macro definition for POCCTRL0 register values are defined in the files listed on Table 3.1 as follows:

```
...
#define POC_SD3_DS_33V          ((uint32_t)1U << 29U)
#define POC_SD3_DAT7_33V        ((uint32_t)1U << 28U)
#define POC_SD3_DAT6_33V        ((uint32_t)1U << 27U)
#define POC_SD3_DAT5_33V        ((uint32_t)1U << 26U)
#define POC_SD3_DAT4_33V        ((uint32_t)1U << 25U)
...
```

Refer to "8.2.5 POC Control Register 0 (POCCTRL0)" [RZG2H], [RZ/G2M] [RZ/G2N] or "9.2.5 POC Control Register 0 (POCCTRL0)" [RZG2E] of "RZ/G Series, 2nd Generation User's Manual: Hardware" [4] for the meaning of each bit.

POC Control Register 2 (POCCTRL2, named as POCCTRL2 in the source code) controls IO voltage level of Ethernet AVB-IF. Note that the settings on this register must match the voltage of VDDQ25_AVB0.

```
        reg = mmio_read_32(POCCTRL2_MASK);
        reg = ((reg & POCCTRL2_MASK) & ~ POC2_VREF_33V);
        pfc_reg_write(PFC_POCCTRL2, reg);
```

Macro definition for POCCTRL2 register values are defined in the files listed on Table 3.1 as follows:

```
#define POC2_VREF_33V          ((uint32_t)1U << 0U)
```

Refer to "9.2.6 POC Control Register 2 (POCCTRL2)" [RZ/G2E] of "RZ/G Series, 2nd Generation User's Manual: Hardware" [4] for the meaning of each bit.

3.2.2.4 Pull-up/Pull-down

LSI pin pull-up/down control Register 0 (PUD0) to LSI pin pull-up/down control Register 5 (PUD5) control pull-up/pull-down of pins.

```
        /* initialize LSI pin pull-up/down control */
        pfc_reg_write(PFC_PUD0, 0x00080000U);
        pfc_reg_write(PFC_PUD1, 0xCE398464U);
        pfc_reg_write(PFC_PUD2, 0xA4C380F4U);
        pfc_reg_write(PFC_PUD3, 0x0000079FU);
        pfc_reg_write(PFC_PUD4, 0xFFF0FFFFU);
        pfc_reg_write(PFC_PUD5, 0x40000000U);
```

Refer to "8.2.8 LSI pin pull-up/down control Register 0-5 (PUD0-5)" [RZG2H], [RZ/G2M] [RZ/G2N] or "9.2.9 LSI pin pull-up/down control Register 0-5 (PUD0-5)" [RZG2E] of "RZ/G Series, 2nd Generation User's Manual: Hardware" [4] for the meaning of each bit.

LSI pin pull-enable register 0 (PUEN0) to LSI pin pull-enable register 5 (PUEN5) control on/off of pins pull-up/pull-down.

```
        /* initialize LSI pin pull-enable register */
        pfc_reg_write(PFC_PUEN0, 0x00000000U);
        pfc_reg_write(PFC_PUEN1, 0x00300000U);
        pfc_reg_write(PFC_PUEN2, 0x00400074U);
        pfc_reg_write(PFC_PUEN3, 0x00000000U);
        pfc_reg_write(PFC_PUEN4, 0x07900600U);
        pfc_reg_write(PFC_PUEN5, 0x00000000U);
```

Refer to "8.2.7 LSI pin pull-enable register 0-5 (PUEN0-5)" [RZG2H], [RZ/G2M] [RZ/G2N] or "9.2.8 LSI pin pull-enable register 0-5 (PUEN0-5)" [RZ/G2E] of "RZ/G Series, 2nd Generation User's Manual: Hardware" for the meaning of each bit.

Please be careful when set these bits. It can break your board if any wrong setting.

3.2.2.5 GPIO polarity

Positive/Negative Logic Select Register 0 (POSNEG0) to Positive/Negative Logic Select Register n (POSNEGn) (n = 7 [RZ/G2H], [RZ/G2M], [RZ/G2N]; n=6 [RZ/G2E]) select the polarity (positive or negative logic) of each port pin in general input mode, general output mode, or interrupt input mode.

```
        /* initialize positive/negative logic select */
        mmio_write_32(GPIO_POSNEG0, 0x00000000U);
        mmio_write_32(GPIO_POSNEG1, 0x00000000U);
        mmio_write_32(GPIO_POSNEG2, 0x00000000U);
        mmio_write_32(GPIO_POSNEG3, 0x00000000U);
        mmio_write_32(GPIO_POSNEG4, 0x00000000U);
        mmio_write_32(GPIO_POSNEG5, 0x00000000U);
        mmio_write_32(GPIO_POSNEG6, 0x00000000U);
```

Refer to "10.2.9 Positive/Negative Logic Select Register n (POSNEG0 to POSNEGn)" of "RZ/G Series, 2nd Generation User’s Manual: Hardware" [4] for the meaning of each bit.

3.2.2.6 General IO/Interrupt Switching

General IO/Interrupt Switching Register 0 (IOINTSEL0) to General IO/Interrupt Switching Register n (IOINTSELn) (n = 7 [RZ/G2H], [RZ/G2M], [RZ/G2N]; n=6 [RZ/G2E]) select either general input/output mode or interrupt input mode for each of the port pins 0 to 31 of the GPIO block.

```
        /* initialize general IO/interrupt switching */
        mmio_write_32(GPIO_IOINTSEL0, 0x00000000U);
        mmio_write_32(GPIO_IOINTSEL1, 0x00000000U);
        mmio_write_32(GPIO_IOINTSEL2, 0x00000000U);
        mmio_write_32(GPIO_IOINTSEL3, 0x00000000U);
        mmio_write_32(GPIO_IOINTSEL4, 0x00000000U);
        mmio_write_32(GPIO_IOINTSEL5, 0x00000000U);
        mmio_write_32(GPIO_IOINTSEL6, 0x00000000U);
```

Refer to " 10.2.1 General IO/Interrupt Switching Register n (IOINTSEL0 to IOINTSELn)" of "RZ/G Series, 2nd Generation User’s Manual: Hardware" [4] for the meaning of each bit.

3.2.2.7 GPIO output value settings

General Output Register 0 (OUTDT0) to General Output Register n (OUTDTn) (n = 7 [RZ/G2H], [RZ/G2M], [RZ/G2N]; n=6 [RZ/G2E]) select the values of GPIOs with General output mode.

```
        /* initialize general output register */
        mmio_write_32(GPIO_OUTDT0, 0x00000000U);
        mmio_write_32(GPIO_OUTDT1, 0x00000000U);
        mmio_write_32(GPIO_OUTDT2, 0x00000000U);
        mmio_write_32(GPIO_OUTDT3, 0x00006000U);
        mmio_write_32(GPIO_OUTDT5, 0x00000000U);
        mmio_write_32(GPIO_OUTDT6, 0x00000000U);
```

Refer to "10.2.3 General Output Register n (OUTDT0 to OUTDTn)" of "RZ/G Series, 2nd Generation User’s Manual: Hardware" [4] for the meaning of each bit.

Please be careful when set these bits. It can break your board if any wrong setting.

3.2.2.8 GPIO input/output

General Input/Output Switching Register 0 (INOUTSEL0) to General Input/Output Switching Register n (INOUTSELn) (n = 7 [RZ/G2H], [RZ/G2M], [RZ/G2N]; n=6 [RZ/G2E]) select selects either general input or general output mode for a port using the bit corresponding to the port number.

```
/* initialize general input/output switching */
mmio_write_32(GPIO_INOUTSEL0, 0x00020000U);
mmio_write_32(GPIO_INOUTSEL1, 0x00100000U);
mmio_write_32(GPIO_INOUTSEL2, 0x03000000U);
mmio_write_32(GPIO_INOUTSEL3, 0x0000E000U);
mmio_write_32(GPIO_INOUTSEL4, 0x00000440U);
mmio_write_32(GPIO_INOUTSEL5, 0x00080000U);
mmio_write_32(GPIO_INOUTSEL6, 0x00000010U);
```

Refer to "10.2.2 General Input/Output Switching Register n (INOUTSEL0 to INOUTSELn)" of "RZ/G Series, 2nd Generation User's Manual: Hardware" [4] for the meaning of each bit.

3.2.3 Mapping on SPI Flash

On VLP64 environment, U-Boot is stored the SPI flash area starting from 0x00300000 on SPI Flash. U-Boot's store address and size are defined in `tools/renesas/rzg_layout_create/sa6.c` . Defined values on Table 3.2 will specify the address and size of U-Boot.

Table 3.2 Defined value related to U-Boot in sa6.c

Define	Default value	Description
RZG_BL33SRC_ADDRESS *1	0x00300000	Offset address of U-Boot on SPI Flash
RZG_BL33DST_ADDRESS *1	0x50000000	Load destination address of U-Boot (Lower 32 bits of 64 bits address)
RZG_BL33DST_ADDRESSH *1	0x00000000	Load destination address of U-Boot (Higher 32 bits of 64 bits address)
RZG_BL33DST_SIZE	0x00040000	U-boot image size in a word unit

Note: 1.: `RZG_SA6_TYPE` is set to `RZG_SA6_TYPE_HYPERFLASH` on VLP64 so definitions surrounded by ifdef-else of `RZG_SA6_TYPE==RZG_SA6_TYPE_HYPERFLASH` will be activated.

tools/renesas/rzg_layout_create/sa6.c:

```
...

#define RZG_SA6_TYPE_HYPERFLASH       (0)
#define RZG_SA6_TYPE_EMMC            (1)

#if (RZG_SA6_TYPE == RZG_SA6_TYPE_HYPERFLASH)


...

/* Source address on flash for BL33 */
#define RZG_BL33SRC_ADDRESS          (0x00300000U)
...

/* Destination address for BL33 */
#define RZG_BL33DST_ADDRESS          (0x50000000U)
#define RZG_BL33DST_ADDRESSH         (0x00000000U)
/* Destination size for BL33 */
#define RZG_BL33DST_SIZE             (0x00040000U)

...
```

3.2.4 DRAM settings

Regarding how to port TF-A to the custom board with different DRAM capacity, number or speed from the reference boards, please contact to Renesas sales.

4. Porting U-Boot

This section describes how to port U-Boot to a custom board you are developing.

For details on the environment-specific procedure of U-boot porting, for example accessing source code, building or deploying, see the following environment-related documents:

- GitHub-based development environment: Refer to "RZ/G2 Yocto recipe Start-Up Guide" [1]
- VLP64: Refer to "Release Note" [2] and "Board Start-up Guide" [5] provided with VLP64

For details on the functions of U-Boot, see the “U-Boot User’s Manual: Software” for the RZ/G 2nd Generation Series.

4.1 Editing the U-Boot Source Code

To port U-Boot in the Linux BSP for use with the reference boards provided as the RZ/G Linux platform to a custom board you are developing, edit the U-Boot source code.

For details on how to edit the U-Boot source code, see the document for each development environment.

- GitHub-based development environment:
  A. From Bitbake tmp directory:
     After executing “Building Instructions” in “RZ/G2 Yocto recipe Start-Up Guide” [1], the U-boot sources will be on `$WORK/build/tmp/work/`*`[machine name]`*`-poky-linux/u-boot/...(snip).../git/`. You can edit these sources using editor application. Note that all source modification on this directory should be saved as Yocto recipes outside this directory.
  B. From GitHub:
     https://github.com/renesas-rz/renesas-u-boot-cip.git
     (branch v2018.09/rzg2,
     revision d2e3e367b6eb704f8c49537ca8e05577af6885e1 in the case of BSP-1.0.0,
     revision ca1cfbd21c01030027f476d34cd77d7aca3f5e82 in the case of BSP-1.0.1,
     revision d867a25a9e2b1a3e33ab3ae84c1a7512f51547c1 in the case of BSP-1.0.2,
     revision d9dbce52865ec3bf962769a02ce608e6e5c56bc0 in the case of BSP-1.0.3-RT,
     revision af0a5da1dd977a2ab1b00411d8e261796fb5fd5d in the case of BSP-1.0.4,
     revision 1e52f9518a85563f4752b7ca90ec34e0e000be25 in the case of BSP-1.0.5-RT,
     revision 19d7c74495e20b5d072e8843945e84364b0a0c1e in the case of BSP-1.0.6,
     revision a5d350acb9a0580a2bf53b9e07a9262257597eb6 in the case of BSP-1.0.7-RT)
     (branch v2020.10/rzg2,
     revision 3bc1267362123ce07acb6bdbcaeedec161bf3eef in the case of BSP-1.0.8,
     revision 3bc1267362123ce07acb6bdbcaeedec161bf3eef in the case of BSP-1.0.9-RT,
     revision c85fc78bf2de38d7580b571f29553cb1df236da0 in the case of BSP-1.0.10)
     (branch v2021.10/rzg2,
     revision 22ccd65ae1f645a3ed5bfd8e80e27a3af15f9243 in the case of BSP-1.0.12)

- VLP64: "After executing “Building images to run on the board” in “Release Note” provided with VLP64", the U-boot sources will be on `$WORK/build/tmp/work/`*`[machine name]`*`-poky-linux/u-boot/...(snip).../git/`. You can edit these sources using editor application. Note that all source modification on this directory should be saved as Yocto recipes outside this directory.

Table 4.1 The directories of U-Boot source

Version	Board	Directory of source codes
1.0.0	Silicon Linux EK874 Evaluation Kit (RZ/G2E)	$WORK/build/tmp/work/ek874-poky-linux/u-boot/…(snip)…/git/
1.0.1	Silicon Linux EK874 Evaluation Kit (RZ/G2E)	$WORK/build/tmp/work/ek874-poky-linux/u-boot/…(snip)…/git/
	Hoperun Technology HiHope RZ/G2M platform	$WORK/build/tmp/work/hihope_rzg2m-poky-linux/…(snip)…/git/
1.0.2	Silicon Linux EK874 Evaluation Kit (RZ/G2E)	$WORK/build/tmp/work/ek874-poky-linux/u-boot/…(snip)…/git/
	Hoperun Technology HiHope RZ/G2M platform	$WORK/build/tmp/work/hihope_rzg2m-poky-linux/…(snip)…/git/
	Hoperun Technology HiHope RZ/G2N platform	$WORK/build/tmp/work/hihope_rzg2n-poky-linux/…(snip)…/git/
1.0.3-RT	Silicon Linux EK874 Evaluation Kit (RZ/G2E)	$WORK/build/tmp/work/ek874-poky-linux/u-boot/…(snip)…/git/
	Hoperun Technology HiHope RZ/G2M platform	$WORK/build/tmp/work/hihope_rzg2m-poky-linux/…(snip)…/git/
	Hoperun Technology HiHope RZ/G2N platform	$WORK/build/tmp/work/hihope_rzg2n-poky-linux/…(snip)…/git/
1.0.4 1.0.5-RT …	Silicon Linux EK874 Evaluation Kit (RZ/G2E)	$WORK/build/tmp/work/ek874-poky-linux/u-boot/…(snip)…/git/
	Hoperun Technology HiHope RZ/G2M platform	$WORK/build/tmp/work/hihope_rzg2m-poky-linux/…(snip)…/git/
	Hoperun Technology HiHope RZ/G2N platform	$WORK/build/tmp/work/hihope_rzg2n-poky-linux/…(snip)…/git/
	Hoperun Technology HiHope RZ/G2H platform	$WORK/build/tmp/work/hihope_rzg2h-poky-linux/…(snip)…/git/

4.2 List of Board-specific files

Table 4.2 lists files which include board specific parts to the EK874 evaluation kit as an example of the board.

Table 4.2 List of Board-specific Files

Category	Filename	Remarks
Files related to board initialize	board/silinux/ek874/ek874.c	The names of directories and files depend on the reference board. For custom board support, the new directories and files are required based on those of the reference boards.
	board/silinux/ek874/Kconfig	
	board/silinux/ek874/Makefile	
	include/configs/silinux-ek874.h	

Note: Followings are the coresspondings between the boards name code and the actual board names:

- ek874: Silicon Linux EK874 Evaluation Kit (combine of CAT874 base-board and CAT875 sub-board)
- hihope-rzg2m: Hoperun Technology HiHope RZ/G2M platform
- hihope-rzg2n: Hoperun Technology HiHope RZ/G2N platform
- hihope-rzg2h: Hoperun Technology HiHope RZ/G2H platform

4.3 Procedure for Porting U-Boot

Follow the steps below to port the U-Boot.

Details on the steps of porting are given under “Building the Software”, “Testing of U-Boot Modifications, Ports to New Hardware, etc.”, and “U-Boot Porting Guide” in the `README` file [6] of the U-Boot source code.

Table 4.3 Procedure for Porting U-Boot

Section for Reference on Details	Description
4.3.1	Deciding the board name "custom-rzg2e"
4.3.2	Creating the Board Directory board/renesas/custom-rzg2e
4.3.3	Creating the board file board/renesas/custom-rzg2e/custom-rzg2e.c This file contains board_early_init_f(), board_init(), and board_late_init() functions, which perform init actions for the board.
4.3.4	Creating the board Kconfig file board/renesas/custom-rzg2e/Kconfig
4.3.5	Creating the Makefile board/renesas/custom-rzg2e/Makefile
4.3.6	Creating the defconfig file configs/r8a774c0_custom-rzg2e_defconfig
4.3.7	Creating the board header File include/configs/custom-rzg2e.h
4.3.8	Adding the entry to the board Kconfig file to architecture's Kconfig arch/arm/mach-rmobile/Kconfig.64
4.3.9	Creating the Device tree file arch/arm/dts/r8a774c0-custom-rzg2e.dts
4.3.10	Building U-Boot
4.3.11	Writing U-Boot to a Custom Board

4.3.1 Deciding the board name

U-Boot requires the board name to manage the settings for each board. It is recommended that the board name for U-Boot is the same as Yocto's machine configuration name set in 2.1 Create machine config file. Followings are the name of RZ/G2 reference boards:

- ek874: Silicon Linux EK874 Evaluation Kit (combine of CAT874 base-board and CAT875 sub-board)
- hihope-rzg2m: Hoperun Technology HiHope RZ/G2M platform
- hihope-rzg2n: Hoperun Technology HiHope RZ/G2N platform
- hihope-rzg2h: Hoperun Technology HiHope RZ/G2H platform

4.3.2 Creating the Board Directory

Users need to create board-specific versions of the files listed in the Table 4. for the custom board. Create the `custom-rzg2e` directory for the board under the directory `board`, and copy the files from the reference board's files:

- RZ/G2E: files in the "`board/silinux/ek874`"
- RZ/G2H, RZ/G2M (v1.3, v3.0), RZ/G2N: files in the "`board/hoperun/hihope-rzg2`"

Note: the actual naming rule is "`board/<vendor name>/<board name>`", so making your <vendor name> directory is recommended. In this document, the vender name is "`renesas`" for the ease of explanation.

Change the filenames of `ek874.c` to the desired names. In this case, they have been changed to `custom-rzg2e.c`. An example of the configuration of board-specific files that the users are to create is given in below table.

Table 4.4 Example of the File Configuration for the Custom Board

Directory	Filename
board/renesas/custom-rzg2e	Makefile
	custom-rzg2e.c
	Kconfig

4.3.3 Creating the board file

Create the file `board/renesas/custom-rzg2e/custom-rzg2e.c` based on the file of the reference boards:

- RZ/G2E: `board/silinux/ek874/ek874.c`
- RZ/G2H, RZ/G2M (v1.3, v3.0), RZ/G2N: `board/hoperun/hihope-rzg2/hihope-rzg2.c`

This file must implement function `board_init()` to perform the board initialize action. Also `board_early_init_f()` and `board_late_init()` functions can be used for early and late actions.

The board is initialized by the `board_init_f()` functions in the `common/board_f.c` file. The `board_init_f()` function handles processing defined by the `init_sequence_f[]` array of pointers to functions. The `init_sequence_f[]` array includes a pointer to the `board_early_init_f()` function.

```
static const init_fnc_t init_sequence_f[] = {
```

```
            setup_mon_len,
#ifdef CONFIG_OF_CONTROL
            fdtdec_setup,
#endif
            ...
            arch_cpu_init,        /* basic arch cpu dependent setup */
            mach_cpu_init,        /* SoC/machine dependent CPU setup */
            initf_dm,
            arch_cpu_init_dm,
#if defined(CONFIG_BOARD_EARLY_INIT_F)
            board_early_init_f,
#endif
#if defined(CONFIG_PPC) || defined(CONFIG_SYS_FSL_CLK) || defined(CONFIG_M68K)
            /* get CPU and bus clocks according to the environment variable */
            get_clocks,           /* get CPU and bus clocks (etc.) */
#endif
            ...
};
```

In addition, the `board_init()` and `board_late_init()` functions are called from the `board_init_r()` function in `common/board_r.c`. The `board_init_r()` function handles processing defined by the `init_sequence_r[]` array of pointers to functions, which includes `board_init()` and `board_late_init()`.

```
static init_fnc_t init_sequence_r[] = {
            initr_trace,
            initr_reloc,
            ...
#if defined(CONFIG_ARM) || defined(CONFIG_NDS32) || defined(CONFIG_RISCV) || \
            defined(CONFIG_SANDBOX)
            board_init,     /* Setup chipselects */
#endif
            ...
#ifdef CONFIG_BOARD_LATE_INIT
            board_late_init,
#endif
            ...
}
```

Use the functions in the `board/renesas/custom-rzg2e/custom-rzg2e.c` file shown in **Table 4.5** to initialize new boards.

Table 4.5 List of Board Initialization Functions

Function Name	**Description**
board_early_init_f(void)	This function is called if **CONFIG_BOARD_EARLY_INIT_F** is defined in the include/configs/custom-rzg2e.h file. For U-Boot for use with the EK874, this function does nothing.
board_init(void)	This function is called between the board_early_init_f() and board_late_init() functions. For U-Boot for use with the EK874, this function just set the boot parameters address.
board_late_init(void)	This function is called if **CONFIG_BOARD_LATE_INIT** is defined in the include/configs/custom-rzg2e.h file. U-Boot for use with the EK874 does not use this config.

4.3.4 Creating the board Kconfig file

Create the file `board/renesas/custom-rzg2e/Kconfig` base on reference:

- RZ/G2E: `board/silinux/ek874/Kconfig`
- RZ/G2H, RZ/G2M (v1.3, v3.0), RZ/G2N: `board/hoperun/hihope-rzg2/Kconfig`

Following is the `Kconfig` example for the custom board:

```
if TARGET_CUSTOM_RZG2E

config SYS_SOC
       default "rmobile"

config SYS_BOARD
       default "custom-rzg2e"

config SYS_VENDOR
       default "renesas"

config SYS_CONFIG_NAME
       default "custom-rzg2e"

endif
```

Settings here will indicate the path to the `Makefile` to include the board file (related to 4.3.2 Creating the Board Directory and 4.3.3 Creating the board file) and the path to the board header file (related to 4.3.7 Creating the board header File):

```
board/SYS_VENDOR/SYS_BOARD/Makefile -> board/renesas/custom-rzg2e/Makefile
include/configs/SYS_CONFIG_NAME.h -> include/configs/custom-rzg2e.h
```

Note that `TARGET_CUSTOM_RZG2E` should be defined in the defconfig file which created in 4.3.6 Creating the defconfig file.

4.3.5 Creating the Makefile

Create file `board/renesas/custom-rzg2e/Makefile` file with below content. The name must be matched with the board file created in 4.3.3 Creating the board file, but change `.c` extension to `.o`

```
obj-y   := custom-rzg2e.o
```

4.3.6 Creating the defconfig file

The defconfig file contains the user config-able settings for the board. A new file should be created for custom board, for example `configs/r8a774c0_custom_rzg2e_defconfig`, based on below reference:

- RZ/G2E: `configs/silinux_ek874_defconfig`
- RZ/G2H, RZ/G2M (v1.3, v3.0), RZ/G2N: `configs/ hihope-rzg2_defconfig`

In this defconfig, the default device tree must point to the file created above, but change `.dts` extension to `.dtb` (`.dtb` is the file extension of binary device tree after build from `.dts`). This device tree name is related to **5.5 Creating the Device Tree and Modifying the Makefile**.

```
CONFIG_DEFAULT_FDT_FILE="r8a774c0-custom-rzg2e.dtb"
```

Also, the settings to specify MPU and boards shall be specified in this defconfig:

```
CONFIG_ARM=y
CONFIG_ARCH_RMOBILE=y
...
CONFIG_RCAR_GEN3=y
CONFIG_R8A774C0=y
CONFIG_TARGET_CUSTOM_RZG2E=y
...
```

Note that `CONFIG_R8A774C0` is specified depends on the MPU on the custom board, and `CONFIG_TARGET CUSTOM_RZG2E` is specified as defined in the board Kconfig file (refer to **4.3.4 Creating the board Kconfig file**).

Identify the devicetree for U-boot configuration is also required. This setting must be the same as the devicetree creating in 4.3.9 Creating the Device tree file.

```
...
CONFIG_DEFAULT_DEVICE_TREE="r8a774c0-custom-rzg2e-u-boot"
...
```

Other configs can be chosen freely. For example, below are list of configs to enable driver support for micro-SD and USB in EK874:

```
...
CONFIG_DM_MMC=y
CONFIG_MMC_IO_VOLTAGE=y
CONFIG_MMC_UHS_SUPPORT=y
CONFIG_MMC_HS200_SUPPORT=y
CONFIG_RENESAS_SDHI=y
...
CONFIG_USB=y
CONFIG_DM_USB=y
CONFIG_USB_XHCI_HCD=y
CONFIG_USB_EHCI_HCD=y
CONFIG_USB_EHCI_GENERIC=y
CONFIG_USB_STORAGE=y
```

Note that each configuration items must be described on Kconfig files in the U-Boot source. For example, `CONFIG_USB` is specified in `drivers/usb/Kconfig`.

4.3.7 Creating the board header File

Copy the boad header file of your MPU's reference board and change its filename as desired. In this case, it has been changed to `include/configs/custom-rzg2e.h`.

- RZ/G2E: `include/configs/silinux-ek874.h`
- RZ/G2H, RZ/G2M (v1.3, v3.0), RZ/G2N: `include/configs/hihope-rzg2.h`

Modify the macro definitions in the `include/configs/custom-rzg2e`.h file for the custom board. This section only describes some of items to be set.

Note that the U-boot configuration by the macros in board header file is gradually migrated to defconfig, Kconfig and devicetree approach. Already many configurations are done by defconfig and device tree.

For details on the other items, refer to "Configuration Options" in `README` for the U-Boot source code.

Below table shows some of settings in the `include/configs/silinux-ek874.h file`.

Table 4.6 Some of Settings in the include/configs/silinux-ek874.h File

Symbol	Example value	Description
#include "rcar-gen3-common.h"		Includes the commons configs in file rcar-gen3-common.h, include some configs for console, memory, …
CONFIG_BITBANGMII_MULTI		Define config to enable Ethernet features

Below table shows some of settings in the `include/configs/rcar-gen3-common.h` file.

Table 4.7 Some of Settings in the include/configs/rcar-gen3-common.h File

Symbol	Example value	Description
DRAM_RSV_SIZE	0x08000000	Size of the reserved DRAM area from the top of DRAM area. Note that this macro is only used in rcar-gen3-common.h
CONFIG_SYS_SDRAM_BASE	(0x40000000 + DRAM_RSV_SIZE)	Start address of DRAM area for U-Boot
CONFIG_MAX_MEM_MAPPED	(0x80000000u - DRAM_RSV_SIZE)	DRAM size
CONFIG_SYS_LOAD_ADDR	0x58000000	
CONFIG_LOADADDR	CONFIG_SYS_LOAD_ADDR	Default address to be set to environment variable "loadaddr"
CONFIG_EXTRA_ENV_SETTINGS	"usb_pgood_delay=2000\0" \ "fdt_high=0xffffffffffffffff\0" \ "initrd_high=0xffffffffffffffff\0"	Additional default environment settings (note that some default environment settings) fdt_high: If set this restricts the maximum address that the flattened device tree will be copied into upon boot. If this is set to the special value 0xFFFFFFFFFFFFFFFF then the fdt will not be copied. initrd_high: If this variable is not set, initrd images will be copied to the highest possible address in RAM. If this is set to the special value 0xFFFFFFFFFFFFFFFF then the initrd images will not be copied. usb_pgood_delay: Wait for power to become stable (msec). If not defined, the default wait time is 100 ms. If it is set to a value less than 100, it will be processed with a waiting time of 100 ms.
CONFIG_BOOTCOMMAND	"tftp 0x48080000 Image; tftp 0x48000000 Image-"CONFIG_DEFAULT_FDT_FILE"; booti 0x48080000 - 0x48000000"	Command line to boot Linux Kernel. This command will be executed after bootdelay seconds. If no definitions are held on this variable, u-boot use console mode.

4.3.8 Adding the entry to the board Kconfig file to architecture's Kconfig

The board Kconfig files created **in 4.3.4 Creating the board Kconfig file** are to be included the architecture's Kconfig file. Add config entry for the board Kconfig file and source that file in `arch/arm/mach-rmobile/Kconfig.64` as follows:

```
if RCAR_GEN3

...

choice

...

config TARGET_CUSTOM_RZG2E
      bool "RZ/G2 Custom board"
      help
        Support for RZ/G2 Custom board

endchoice

config SYS_SOC
      default "rmobile"

...
source "board/renesas/custom-rzg2e/Kconfig"
...

endif
```

4.3.9 Creating the Device tree file

U-boot uses device tree in the same manner as Linux kernel. For detail explanation of device tree, refer to 5.5. Creating the Device Tree and Modifying the Makefile

Create the device tree for custom board, for example `arch/arm/dts/r8a774c0-custom-rzg2e.dts` with example:

- RZ/G2E: `arch/arm/dts/r8a774c0-ek874.dts`
- RZ/G2M (v1.3, v3.0): `arch/arm/dts/r8a774a1-hihope-rzg2m.dts`
- RZ/G2N: `arch/arm/dts/r8a774a1-hihope-rzg2n.dts`
- RZ/G2H: `arch/arm/dts/r8a774b1-hihope-rzg2h.dts`

This file must be modified corresponding to the real hardware on the custom board.

For example, if memory (DRAM) of the custom board is different, memory setting must be changed:

```
        memory@48000000 {
                device_type = "memory";
                /* first 128MB is reserved for secure area. */
                reg = <0x0 0x48000000 0x0 0x38000000>;
        };
```

Or if the custom board does not use the Ethernet over AVB module, remove the AVB node:

```
&avb {
        pinctrl-0 = <&avb_pins>;
        pinctrl-names = "default";
        renesas,no-ether-link;
        phy-handle = <&phy0>;
        phy-mode = "rgmii-txid";
        status = "okay";

        phy0: ethernet-phy@0 {
                rxc-skew-ps = <1500>;
                reg = <0>;
                interrupt-parent = <&gpio2>;
                interrupts = <21 IRQ_TYPE_LEVEL_LOW>;
                reset-gpios = <&gpio1 20 GPIO_ACTIVE_LOW>;
        };
};
```

4.3.10 Building U-Boot

For details on how to build U-Boot, see the documents for each development environment:

- GitHub-based development environment/downloaded VLP: Execute **`bitbake -C compile u-boot`**. Symbolic links u-boot.bin and u-boot.srec on `$WORK/build/tmp/deploy/images/*/` will be updated.
  - — Note that all modification to the source code under `$WORK/build/tmp/work` should be saved as a Yocto recipe. This is because the directories under the `$WORK/build/tmp/work` will be removed by the Bitbake commands like **`bitbake -c cleanall u-boot`**.

4.3.11 Writing U-Boot to a Custom Board

- GitHub-based development environment: Refer to "Writing of IPL/Secure" of "RZ/G2 Yocto recipe Start-Up Guide" [1]
- VLP64: Refer to the section "Writing Bootloader" in "Board Start-up Guide" [5] provided with VLP64.

5. Porting the Linux Kernel

This section describes how to port the Linux kernel to a custom board you are developing.

For details on the procedure for using the RZ/G Linux platform tools to customize the Linux kernel, see the documents for each development environments:

- GitHub-based development environment: Refer to "RZ/G2 Yocto recipe Start-Up Guide" [1]
- VLP64: Refer to "Release Note" [2] and "Board Start-up Guide" [5] provided with VLP64

5.1 Editing the Source Code of the Linux Kernel

To port the Linux kernel in the Linux BSP for use with the RZ/G2E Silicon Linux EK874 Evaluation kit provided as the RZ/G Linux platform to a custom board you are developing, edit the source code of the Linux kernel as required.

For details on how to edit the source code of the Linux kernel, see the document for each development environment:

- GitHub-based development environment: After executing “Building Instructions” in “RZ/G2 Yocto recipe Start-Up Guide” [1], the kernel source will be on `$WORK/build/tmp/work-shared/`*[machine name]*`/kernel-source/`. You can edit sources using editor. Note that all source modification in this directory should be saved as Yocto recipe outside of this directory.
- Downloaded VLP: After executing “Building images to run on the board” in “Release Note” [2] provided with VLP64", the kernel source will be on $`WORK/build/tmp/work-shared/`*[machine name]*`/kernel-source/`. You can edit sources using editor. Note that all source modification in this directory should be saved as Yocto recipe outside of this directory.

5.2 Procedure for Porting the Linux Kernel

Table 5.1 shows the procedure for porting the Linux kernel.

Table 5.1 Procedure for Porting the Linux Kernel

Step	Description	Section for Reference on Details
1	Adding the device drivers	5.4
2	Creating the device tree and editing the Makefile	5.5
3	Amending the kernel configuration	5.6
4	Building the Linux Kernel	5.7

5.3 List of Board-specific Files

Table 5.2 lists files specific to the Silicon Linux EK874 Evaluation kit for RZ/G2E as an example of board-specific files.

Table 5.2 List of Board-specific Files

Classification	Filename	Remarks
Kernel configs	arch/arm64/configs/defconfig	—
Files related to device tree	arch/arm64/boot/dts/renesas/Makefile	
	arch/arm64/boot/dts/renesas/r8a774c0.dtsi	The filenames are depend on the reference board and these are the examples of RZ/G2E and EK874.
	arch/arm64/boot/dts/renesas/r8a774c0-cat874.dts	
	arch/arm64/boot/dts/renesas/cat875.dtsi	
	arch/arm64/boot/dts/renesas/r8a774c0-ek874.dts	
	arch/arm64/boot/dts/renesas/r8a774c0-ek874-idk-2121wr.dts	
	arch/arm64/boot/dts/renesas/r8a774c0-ek874-mipi-2.1.dts or arch/arm64/boot/dts/renesas/r8a774c0-ek874-mipi-2.4.dts	
	arch/arm64/boot/dts/renesas/aistarvision-mipi-adapter-2.1.dtsi	

5.4 Adding, Modifying, and Deleting Device Drivers

To add the device drivers for new device on the custom board, take the following steps:

1. Add the source files of the drivers: Add the source files of the drivers to be added to the corresponding directories under `drivers/`.
2. Add the settings for compiling the added source files to the `Makefile` under the directories to which the source files have been added.
3. Add a configuration option to the `Kconfig` under the directories to which the source files have been added. This allow user to select enabling the drivers during kernel configuration.
4. Enable the configuration option during kernel configuration.

To change the device drivers to be used, change the settings to enable or disable the given drivers in the kernel configuration (see section **5.6 Changing the Kernel Configuration**).

To delete a device driver, disable the device driver to be deleted in the kernel configuration. In most cases, removing the source files of the drivers is not required.

Followings are how the pin function controller driver are implemented and built on the Linux kernel. Most of the drivers can be introduced in the same manner.

1. The source file of the GPIO driver is stored under the `drivers/gpio` directory. The source file `gpio-rcar.c` of the driver is placed in the above directory.

```
drivers/gpio/
          Makefile
          Kconfig
          ...
          gpio-rcar.c
          ...
```

2. The line corresponding to each driver is included in the `Makefile` in the individual directories so that each source file becomes a target for compilation. The driver for the Renesas GPIO in the statement of the file.

drivers/gpio/Makefile:

```
...
obj-$(CONFIG_GPIO_RCAR)        += gpio-rcar.o
...
```

3. The entry corresponding to `drivers/gpio/Kconfig` is included so as to use **`make menuconfig`** or another method to set `CONFIG_GPIO_RCAR`. This config is set as the tristate type. This indicates that both embedding the driver in the kernel and handling the driver as a loadable kernel module are supported for this driver. In addition, it being placed between `if` and `endif` indicates that `CONFIG_GPIO_RCAR` is only specifiable when `GPIOLIB` is set. Take this into account.

drivers/gpio/Kconfig:

```
#
# GPIO infrastructure and drivers
#

config ARCH_HAVE_CUSTOM_GPIO_H
	bool
	help
	  Selecting this config option from the architecture Kconfig allows
	  the architecture to provide a custom asm/gpio.h implementation
	  overriding the default implementations.  New uses of this are
	  strongly discouraged.

menuconfig GPIOLIB
	bool "GPIO Support"
	select ANON_INODES
	help
	  This enables GPIO support through the generic GPIO library.
	  You only need to enable this, if you also want to enable
	  one or more of the GPIO drivers below.

	  If unsure, say N.

if GPIOLIB
...
config GPIO_RCAR
	tristate "Renesas R-Car GPIO"
	depends on ARCH_RENESAS || COMPILE_TEST
	select GPIOLIB_IRQCHIP
	help
	  Say yes here to support GPIO on Renesas R-Car SoCs.
...
endif
```

4. To compile the Renesas GPIO driver as part of building the kernel, **`CONFIG_GPIO_RCAR = y`** (for embedding the driver in the kernel) or **`CONFIG_GPIO_RCAR = m`** (for handling the driver as a loadable kernel module) is set in the kernel configuration. For details on how to change the kernel configuration, see section **5.6 Changing the Kernel Configuration**.

5. After enable the config, user need to enable driver in device tree as well. Refer to next section 5.5 for the detail explanation.

5.5 Creating the Device Tree and Modifying the Makefile

Copy the device tree source file and rename and edit it for your custom board. This section describes how to customize the devicetree for the custom board based on RZ/G2E as an example, but the custom board based on the other RZ/G2 Group devices can be handled in the same manner. When the custom board is based on the main board of RZ/G2E EK874 Kit, copy the `r8a774c0-cat874.dts` file and rename it respectively for your custom board and edit it. In this case, it is assumed that the copied file is `r8a774c0-custom-rzg2e.dts`. Refer to Table 5.1 for the other boards.

Table 5.1 Device Tree Files to be copied

Note: All above files are placed in the directory `arch/arm64/boot/dts/renesas/`.

Processor	LSI Version	Board	Board Revision	Device Tree	Included Files
G2H	3.0	HiHope-G2H	4	• r8a774e1-hihope-rzg2h.dts • r8a774e1-hihope-rzg2h-ex.dts • r8a774e1-hihope-rzg2h-ex-idk-1110wr.dts • r8a774e1-hihope-rzg2h-ex-mipi-2.1.dts	• r8a774e1.dtsi • hihope-common.dtsi • hihope-rev4.dtsi • hihope-rzg2-ex.dtsi • hihope-rzg2-ex-lvds.dtsi • rzg2-advantech-idk-1110wr-panel.dtsi • aistarvision-mipi-adapter-2.1.dtsi
G2M	1.2	HiHope-G2M	2	• r8a774a1-hihope-rzg2m-rev2.dts • r8a774a1-hihope-rzg2m-rev2-ex.dts • r8a774a1-hihope-rzg2m-rev2-ex-idk-1110wr.dts • r8a774a1-hihope-rzg2m-rev2-ex-mipi-2.1.dts	• r8a774a1.dtsi • hihope-common.dtsi • hihope-rev2.dtsi • hihope-rzg2-ex-lvds.dtsi • hihope-rzg2-ex.dtsi • rzg2-advantech-idk-1110wr-panel.dtsi • aistarvision-mipi-adapter-2.1.dtsi
	1.3		4	• r8a774a1-hihope-rzg2m.dts • r8a774a1-hihope-rzg2m-ex.dts • r8a774a1-hihope-rzg2m-ex-idk-1110wr.dts • r8a774a1-hihope-rzg2m-ex-mipi-2.1.dts	• r8a774a1.dtsi • hihope-common.dtsi • hihope-rev4.dtsi • hihope-rzg2-ex.dtsi • hihope-rzg2-ex-lvds.dtsi • rzg2-advantech-idk-1110wr-panel.dtsi • aistarvision-mipi-adapter-2.1.dtsi
	3.0		4	• r8a774a3-hihope-rzg2m.dts • r8a774a3-hihope-rzg2m-ex.dts • r8a774a3-hihope-rzg2m-ex-idk-1110wr.dts • r8a774a3-hihope-rzg2m-ex-mipi-2.1.dts	• r8a774a3.dtsi • hihope-common.dtsi • hihope-rev4.dtsi • hihope-rzg2-ex.dtsi • hihope-rzg2-ex-lvds.dtsi • rzg2-advantech-idk-1110wr-panel.dtsi • aistarvision-mipi-adapter-2.1.dtsi
G2N	1.1	HiHope-G2N	2	• r8a774b1-hihope-rzg2n-rev2.dts • r8a774b1-hihope-rzg2n-rev2-ex.dts • r8a774b1-hihope-rzg2n-rev2-ex-idk-1110wr.dts • r8a774b1-hihope-rzg2n-rev2-ex-mipi-2.1.dts	• r8a774b1.dtsi • hihope-common.dtsi • hihope-rev2.dtsi • hihope-rzg2-ex-lvds.dtsi • hihope-rzg2-ex.dtsi • rzg2-advantech-idk-1110wr-panel.dtsi • aistarvision-mipi-adapter-2.1.dtsi
			4	• r8a774b1-hihope-rzg2n.dts	• r8a774b1.dtsi

				• r8a774b1-hihope-rzg2n-ex.dts • r8a774b1-hihope-rzg2n-ex-idk-1110wr.dts • r8a774b1-hihope-rzg2n-ex-mipi-2.1.dts	• hihope-common.dtsi • hihope-rzg2-ex.dtsi • rzg2-advantech-idk-1110wr-panel.dtsi • aistarvision-mipi-adapter-2.1.dtsi
G2E	1.0	EK874	B	• r8a774c0-es10-cat874.dts • r8a774c0-es10-ek874.dts • r8a774c0-es10-ek874-idk-2121wr.dts • r8a774c0-es10-ek874-mipi-2.1.dts	• r8a774c0-es10.dtsi • cat875.dtsi • cat874-common.dtsi • aistarvision-mipi-adapter-2.1.dtsi
	1.1		C	• r8a774c0-cat874-revc.dts • r8a774c0-ek874-revc.dts • r8a774c0-ek874-revc-idk-2121wr.dts • r8a774c0-ek874-revc-mipi-2.1.dts	• r8a774c0.dtsi • cat875.dtsi • aistarvision-mipi-adapter-2.1.dtsi • cat874-common.dtsi
			E	• r8a774c0-cat874.dts • r8a774c0-ek874.dts • r8a774c0-ek874-idk-2121wr.dts • r8a774c0-ek874-mipi-2.1.dts	• r8a774c0.dtsi • cat875.dtsi • aistarvision-mipi-adapter-2.1.dtsi • cat874-common.dtsi

Structure of device tree files for all platforms:

- **Main board only:**
  - G2H: r8a774e1-hihope-rzg2h.dts
  - G2M v1.3: r8a774a1-hihope-rzg2m.dts, r8a774a1-hihope-rzg2m-rev2.dts.
  - G2M v3.0: r8a774a3-hihope-rzg2m.dts
  - G2N: r8a774b1-hihope-rzg2n.dts, r8a774b1-hihope-rzg2n-rev2.dts
  - G2E: r8a774c0-cat874.dts, r8a774c0-cat874-revc.dts, r8a774c0-es10-cat874.dts
- **Main board + Sub board:**
  - G2H: r8a774e1-hihope-rzg2h-ex.dts
  - G2M v1.3: r8a774a1-hihope-rzg2m-ex.dts, r8a774a1-hihope-rzg2m-rev2-ex.dts.
  - G2M v3.0: r8a774a3-hihope-rzg2m-ex.dts
  - G2N: r8a774b1-hihope-rzg2n-ex.dts, r8a774b1-hihope-rzg2n-rev2-ex.dts
  - G2E: r8a774c0-ek874.dts, r8a774c0-ek874-revc.dts, r8a774c0-es10-ek874.dts
- **Main board + Sub board + LVDS Panel:**
  - G2H: r8a774e1-hihope-rzg2h-ex-idk-1110wr.dts
  - G2M v1.3: r8a774a1-hihope-rzg2m-ex-idk-1110wr.dts, r8a774a1-hihope-rzg2m-rev2-ex-idk-1110wr.dts.
  - G2M v3.0: r8a774a3-hihope-rzg2m-ex-idk-1110wr.dts
  - G2N: r8a774b1-hihope-rzg2n-ex-idk-1110wr.dts, r8a774b1-hihope-rzg2n-rev2-ex-idk-1110wr.dts
  - G2E: r8a774c0-ek874-idk-2121wr.dts, r8a774c0-ek874-revc-idk-2121wr.dts, r8a774c0-es10-ek874-idk-2121wr.dts
- **Main board + Sub board + MIPI Mezzanine Camera Adapter v2.1:**
  - G2H: r8a774e1-hihope-rzg2h-ex-mipi-2.1.dts
  - G2M v1.3: r8a774a1-hihope-rzg2m-ex-mipi-2.1.dts, r8a774a1-hihope-rzg2m-rev2-ex.dts.
  - G2M v3.0: r8a774a3-hihope-rzg2m-ex-mipi-2.1.dts

- G2N: r8a774b1-hihope-rzg2n-ex-mipi-2.1.dts, r8a774b1-hihope-rzg2n-rev2-ex-mipi-2.1.dts
- G2E: r8a774c0-ek874-mipi-2.1.dts, r8a774c0-ek874-revc-idk-2121wr.dts, r8a774c0-es10-ek874-mipi-2.1.dts

Note: All device tree files support HDMI by default.

To support MIPI Mezzanine Camera Adapter v2.1 along with LVDS panel, please update in device tree files:
Example:

- **Main board + Sub board + MIPI Mezzanine Camera Adapter v2.1:** r8a774e1-hihope-rzg2h-ex-mipi-2.1.dts

```
-   #include "r8a774e1-hihope-rzg2h-ex.dts"
+  #include " r8a774e1-hihope-rzg2h-ex-idk-1110wr.dts "
```

- **Main board + Sub board + LVDS Panel:** r8a774e1-hihope-rzg2h-ex-idk-1110wr.dts

```
-   #include "r8a774e1-hihope-rzg2h-ex.dts"
+  #include " r8a774e1-hihope-rzg2h-ex-mipi-2.1.dts"
```

5.5.1 Device Trees of the RZ/G2E EK874 Kit

This subsection briefly describes the configuration of the device trees of the Silicon Linux EK874 kit. For the fundamentals of device trees, see the documentation on the Web page at http://elinux.org/Device_Tree_Reference or the Devicetree Specification 0.2 [3].

“Device Tree” means “Device tree source files” or “Device tree blob”. “Device tree source files (DTS)” are text files, included in the kernel source tree. “Device tree blob (DTB)” is a binary file, which will be read from the kernel image when booting. As shown in Figure 5.1, the device tree compiler builds the device tree blob from the device tree source files and related header files.

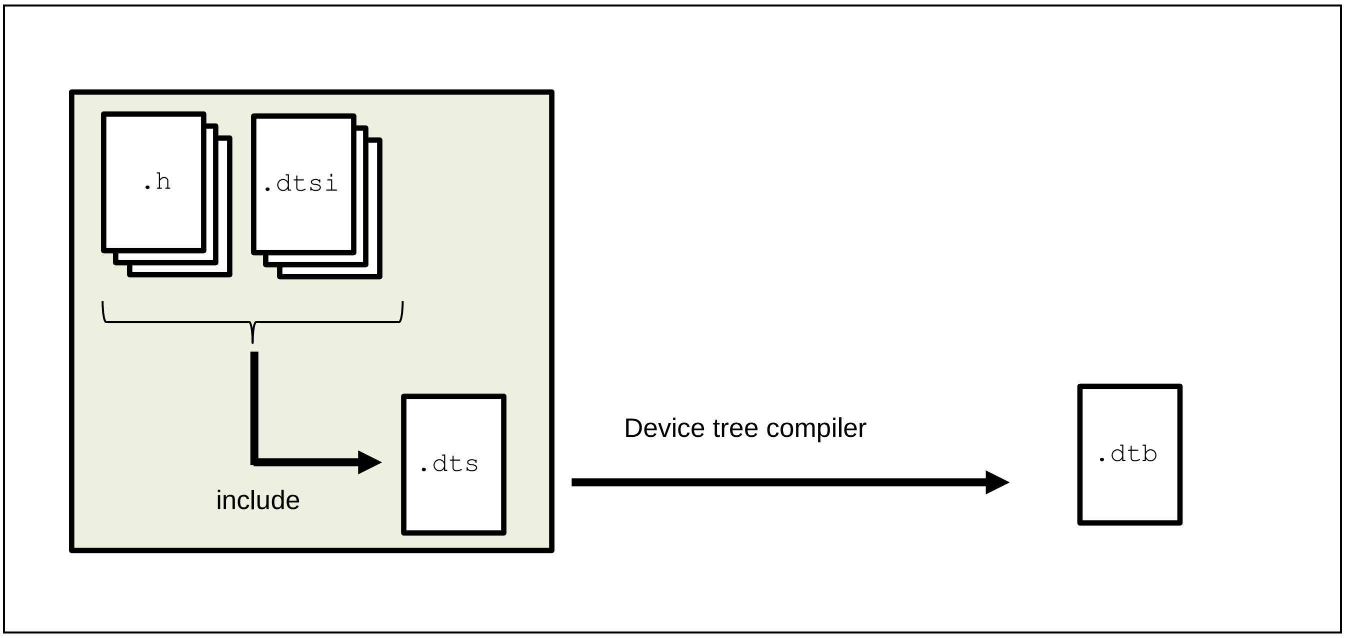

Figure 5.1 A flow from device tree source files to a device tree blob

5.5.2 Creating the Device Tree and Modifying the Makefile

Copy `arch/arm64/boot/dts/renesas/r8a774c0-cat874.dts` and rename it for your custom board. In this case, it is assumed that the copied file is `r8a774c0-custom-rzg2e.dts`.

To build the copied device tree for the custom board, it is required to edit `arch/arm/boot/dts/Makefile`. Note that the filename which should be listed on Makefile is not `.dts` but `.dtb`. Following is the example edit result:

arch/arm64/boot/dts/renesas/Makefile:

```
dtb-$(CONFIG_ARCH_R8A774C0) += r8a774c0-cat874.dtb r8a774c0-ek874.dtb \
                           r8a774c0-custom-rzg2e.dtb
dtb-$(CONFIG_ARCH_R8A7795) += r8a7795-salvator-x.dtb r8a7795-h3ulcb.dtb
dtb-$(CONFIG_ARCH_R8A7795) += r8a7795-h3ulcb-kf.dtb
```

5.5.3 Customizing CMA

There are three types of CMA regions defined in device tree.

1) Default CMA region: It is for kernel, general drivers and multimedia package
   e.g. `arch/arm64/boot/dts/renesas/r8a774c0-cat874.dts`

```
/* global autoconfigured region for contiguous allocations */
             linux,cma@58000000 {
                    compatible = "shared-dma-pool";
                    reusable;
                    reg = <0x00000000 0x58000000 0x0 0x10000000>;
                    linux,cma-default;
```

```
                };
```

0x58000000 is start address of CMA region
0x10000000 is size of CMA region

Notes)

- 128 MB in this CMA is reserved for kernel and general drivers, and the remaining is reserved for multimedia package.
- This CMA region can be adjusted by changing the start address and the size.
- Should take care of the lack of memory allocated by kernel and general drivers when reducing the region size.

2) CMA region for MMP: it is for multimedia package (specific H/Ws)
e.g. `arch/arm64/boot/dts/renesas/r8a774c0-cat874.dts`

```
/* device specific region for contiguous allocations */
                mmp_reserved: linux,multimedia {
                        compatible = "shared-dma-pool";
                        reusable;
                        reg = <0x00000000 0x68000000 0x0 0x08000000>;
                };
```

0x68000000 is start address of CMA region
0x08000000 is size of CMA region

Notes)

- This CMA is reserved for multimedia package.
- This CMA region can be adjusted by changing the start address and the size.
- Should take care of the lack of memory allocated by kernel and general drivers when reducing the region size.

3) CMA region for lossy compress/decompress (specific H/Ws)
e.g. `arch/arm64/boot/dts/renesas/r8a774a1-hihope-rzg2m.dts`

```
/* device specific region for Lossy Decompression */
                lossy_decompress: linux,lossy_decompress@54000000 {
                        no-map;
                        reg = <0x00000000 0x54000000 0x0 0x03000000>;
                }
```

0x54000000 is start address of CMA region
0x03000000 is size of CMA region

Notes)

- This CMA is reserved for lossy compress/decompress feature supported by multimedia package (RZG2E does not support lossy compress/decompress)
- This CMA region can NOT be adjusted, start address and the size is fixed.

5.5.4 Enabling or Disabling Existing Devices

Most devices in `arch/arm64/boot/dts/renesas/r8a774c0.dtsi` are disabled by default. Set the states of individual devices through the properties as shown below.

- Enabling a device: Modify status property as `status = "okay";`.
- Disabling a device: Modify status property as `status = "disabled";`.

If not set the status property, it will be `"okay"` as default.

For example, the SCIF1 and SCIF2 of RZ/G2E are disabled in `r8a774c0.dtsi`.

arch/arm64/boot/dts/renesas/r8a774c0.dtsi:

```
        scif1: serial@e6e68000 {
                compatible = "renesas,scif-r8a774c0",
                             "renesas,rcar-gen3-scif", "renesas,scif";
                reg = <0 0xe6e68000 0 64>;
                interrupts = <GIC_SPI 153 IRQ_TYPE_LEVEL_HIGH>;
                clocks = <&cpg CPG_MOD 206>,
                         <&cpg CPG_CORE R8A774C0_CLK_S3D1C>,
                         <&scif_clk>;
                clock-names = "fck", "brg_int", "scif_clk";
                dmas = <&dmac1 0x53>, <&dmac1 0x52>,
                       <&dmac2 0x53>, <&dmac2 0x52>;
                dma-names = "tx", "rx", "tx", "rx";
                power-domains = <&sysc R8A774C0_PD_ALWAYS_ON>;
                resets = <&cpg 206>;
                status = "disabled";
        };

        scif2: serial@e6e88000 {
                compatible = "renesas,scif-r8a774c0",
                             "renesas,rcar-gen3-scif", "renesas,scif";
                reg = <0 0xe6e88000 0 64>;
                interrupts = <GIC_SPI 164 IRQ_TYPE_LEVEL_HIGH>;
                clocks = <&cpg CPG_MOD 310>,
                         <&cpg CPG_CORE R8A774C0_CLK_S3D1C>,
                         <&scif_clk>;
                clock-names = "fck", "brg_int", "scif_clk";
                power-domains = <&sysc R8A774C0_PD_ALWAYS_ON>;
                resets = <&cpg 310>;
                status = "disabled";
        };
```

For the Silicon Linux EK874 Kit, of scif1 and scif2, only scif2 is enabled in r8a774c0-ek874.dts.

arch/arm64/boot/dts/renesas/r8a774c0-ek874.dts

```
&scif2 {
        pinctrl-0 = <&scif2_pins>;
        pinctrl-names = "default";

        status = "okay";
};
```

5.5.5 Customizing the pin multiplex

This subsection describes how to handle the device tree when changing the pin configuration.

For the general statements related to the pin configuration in the device tree, see the Documentation/devicetree/bindings/pinctrl/pinctrl-bindings.txt. For statements specific to the Renesas pin function controller, which is a driver from Renesas for controlling the pin configuration, see the Documentation/devicetree/bindings/pinctrl/renesas,pfc-pinctrl.txt.

Setting the pin configuration requires the following items.

pinctrl-0 and pinctrl-names properties of the device for which the pin configuration is to be set
Pin configuration node, which is a child node of the pin function controller

For example, the pin configuration of the HSCIF3 interface is set as follows: the node &hscif3 for HSCIF3 configuration has the pinctrl-0 property, which refers to the hscif3_pins node ((1) below). After that, the hscif3_pins node (1) is added as a child node of the pin function controller, pfc, and in the hscif3_pins node, hscif3 (2) is specified as the "function" property and hscif3_data_c (3) and hscif3_ctrl_c (4) are specified as the “groups” properties.

arch/arm64/boot/dts/renesas/r8a774c0-cat874.dts:

```
...
&pfc {
        ...
             (1)
        hscif3_pins: hscif3 {                 (3)              (4)
                groups = "hscif3_data_c", "hscif3_ctrl_c";
                function = "hscif3";
        };                          (2)

        ...
};
...
&hscif3 {                         (1)
        pinctrl-0 = <&hscif3_pins>;
        pinctrl-names = "default";
```

```
        ...
};
```

The values for “function” and “groups” properties are defined in `drivers/pinctrl/sh-pfc/pfc-r8a77990.c`

Notice	*The file name is different here (should be pfc-r8a774c0.c for RZ/G2E). This happens because r8a77990 pin is compatible with r8a774c0, so RZ/G2E re-uses pin define from r8a77990 instead of creating a new file. But remember that r8a77990 has more hardware than RZ/G2E, and in some hardware r8a77990 has more channels than RZ/G2E. Should refer to hardware manual of RZ/G2E to confirm.*

This is decided by below configs in

arch/arm64/boot/dts/renesas/r8a774c0.dtsi

```
pfc: pin-controller@e6060000 {                        (5)
        compatible = "renesas,pfc-r8a774c0";
        reg = <0 0xe6060000 0 0x50c>;
};
```

and

drivers/pinctrl/sh-pfc/core.c

```
#ifdef CONFIG_PINCTRL_PFC_R8A774C0
{                                                     (5)
        compatible = "renesas,pfc-r8a774c0",
        .data = &r8a774c0_pinmux_info,
},                                              (6)
#endif
```

and

drivers/pinctrl/sh-pfc/pfc-r8a77990.c

```
#ifdef CONFIG_PINCTRL_PFC_R8A774C0                         (6)
const struct sh_pfc_soc_info r8a774c0_pinmux_info = {
        .name = "r8a774c0_pfc",
        .ops = &r8a77990_pinmux_ops,
        .unlock_reg = 0xe6060000, /* PMMR */

        .function = { PINMUX_FUNCTION_BEGIN, PINMUX_FUNCTION_END },

        .pins = pinmux_pins,
        .nr_pins = ARRAY_SIZE(pinmux_pins),
        ...
#ifdef CONFIG_PINCTRL_PFC_R8A77990
const struct sh_pfc_soc_info r8a77990_pinmux_info = {
        .name = "r8a77990_pfc",
```

```
        .ops = &r8a77990_pinmux_ops,
        .unlock_reg = 0xe6060000, /* PMMR */

        .function = { PINMUX_FUNCTION_BEGIN, PINMUX_FUNCTION_END },

        .pins = pinmux_pins,
        .nr_pins = ARRAY_SIZE(pinmux_pins),
        ...
```

The values specifiable as the "function" properties are arranged in an array as shown below. For example, `hscif3` (2) is specifiable as the "function" property in the pin configuration node.

drivers/pinctrl/sh-pfc/pfc-r8a77990.c:

```
static const struct {
        struct sh_pfc_function common[41];
        struct sh_pfc_function r8a77990[0];
} pinmux_functions = {
        .common = {
                SH_PFC_FUNCTION(audio_clk),
                SH_PFC_FUNCTION(avb),
                ...            (2)
                SH_PFC_FUNCTION(hscif3),
                ...
                SH_PFC_FUNCTION(du),
                SH_PFC_FUNCTION(ssi),
};
```

The values specifiable as the "`groups`" properties are defined in the array below in response to the "function" properties being specified. For example, if `hscif3` (2) is specified as the "`function`" property, the values specifiable as the "`groups`" properties are defined in the `hscif3_groups[]` array.

drivers/pinctrl/sh-pfc/pfc-r8a77990.c:

```
static const char * const hscif3_groups[] = {
        "hscif3_data_a",      (2)
        "hscif3_data_b",
        "hscif3_clk_b",
        "hscif3_data_c", (3)
        "hscif3_clk_c",
        "hscif3_ctrl_c", (4)
        "hscif3_data_d",
        "hscif3_data_e",
        "hscif3_ctrl_e",
};
```

The meanings of the values specifiable as individual "`groups`" properties can be judged from the following structures. The code below is an example, showing the case where `hscif3_data_c` (3) is specified as a "`groups`" property. `hscif3_data_c_pins[]` is a group of the numbers of the GPIO pins corresponding to pins assigned when `hscif3_data_c` (3) is specified as a "`groups`" property. `hscif3_data_c_mux[]` represents the pin names assigned when `hscif3_data_c` (3) is specified as a "`groups`" property. The pin names, such as `HRX3_C` (7) and `HTX3_C` (8), are described in the hardware manual.

drivers/pinctrl/sh-pfc/pfc-r8a77990.c:

```
static const unsigned int hscif3_data_c_pins[] = {
          /* RX, TX */     (3)
          RCAR_GP_PIN(2, 10), RCAR_GP_PIN(2, 9),
};
                                (3)
static const unsigned int hscif3_data_c_mux[] = {
          HRX3_C_MARK, HTX3_C_MARK,
};             (7)           (8)
```

5.5.6 Newly Creating a Pin Group

```
If no pin groups satisfy the combination of pins to be used in section 0
```

Customizing the pin multiplex, adding a new pin group is required. Add a new pin group to the pinctrl drivers of your MPU. If you use RZ/G2E (R8A774C0), add a new pin group to drivers/pinctrl/sh-pfc/pfc-r8a77990.c.

Followings are how to add a new pin group "`module1_data_c`" corresponding to "`module1`" to pinctrl driver. Find entries of pins you want to add in the pin multiplexing table in the hardware manual of the MPU. And add lines using "Pin Name" of pins and corresponding GPIO in the entry as follows.

drivers/pinctrl/sh-pfc/pfc-r8a77990.c:

```
static const unsigned int module1_data_c_pins[] = {
	RCAR_GP_PIN(m, n), RCAR_GP_PIN(m, n),
};

static const unsigned int module1_data_c_mux[] = {
	MODULE1_SIG1_C_MARK, MODULE1_SIG2_C_MARK,
};

static const struct sh_pfc_pin_group pinmux_groups[] = {
	...
	SH_PFC_PIN_GROUP(module1_data_c),
	...
};

static const char * const module1_groups[] = {
	"module1_data",
	"module1_data_b",
	"module1_data_c",
};
```

If `MODULE1_SIG1_C_MARK` or `MODULE1_SIG2_C_MARK` is not defined in `drivers/pinctrl/sh-pfc/pfc-r8a77990.c`, please contact to Renesas sales.

5.5.7 Adding, Deleting, and Modifying Devices

Adding, deleting, or modifying devices correspond to adding, deleting, or modifying nodes in the device tree, respectively.

In addition, what the node of a device should be depends on the specifications of the driver for the given device. For details on the items to be included in the node, see the documentation for the given driver under `Documentation/devicetree/bindings`. For example, the specifications of the device tree of the HSCIF module are given in the following file.

- `Documentation/devicetree/bindings/serial/renesas,sci-serial.txt`

Here, a change to the HSCIF channels to be used is described as an example. In this example, the changes shown in Table 5.3 are made.

Table 5.3 Example of Changes to the HSCIF Channels to be used

	RZ/G2E Evaluation Kit EK874	Custom Board
HSCIF2	Used with HRX2_A pin, HTX2_A pin, HCTS2#_A and HRTS2#_A	Unused
HSCIF3	Unused	Used with HRX3_C pin, HTX3_C pin, HCTS3#_C and HRTS3#_C

In this case, the changes described below are required. This is because HSCIF2 and HSCIF3 are handled as different devices in the device tree although both are channels of the HSCIF module. Specifically, the deletion of HSCIF2 and the addition of HSCIF3 are required.

1. Disabling HSCIF2 and deleting the corresponding node from the pin configuration
2. Enabling HSCIF3 and adding the corresponding node to the pin configuration

1. Disabling HSCIF2 and deleting the corresponding node from the pin configuration

arch/arm/boot/dts/r8a774c0-custom-rzg2e.dts:

(Note that this code fragment is in unified diff format. Add the lines starting with + and delete those starting with – and make sure the character '+' or '-' will not be included in your code.)

```
 &pfc {
-		hscif2_pins: hscif2 {
-			groups = "hscif2_data_a", "hscif2_ctrl_a";
-			function = "hscif2";
-		};

		...
 };

 ...

-&hscif2 {
-		pinctrl-0 = <&hscif2_pins>;
-		pinctrl-names = "default";
-
-		uart-has-rtscts;
-		status = "okay";
-};
```

2. Enabling SCIF0 and adding the corresponding node to the pin configuration

arch/arm/boot/dts/r8a774c0-custom-rzg2e.dts:

(Note that this code fragment is in unified diff format. Add the lines starting with + and delete those starting with – and make sure the character '+' or '-' will not be included in your code.)

```
 &pfc {
+		hscif3_pins: hscif3 {
+			groups = "hscif3_data_c", "hscif3_ctrl_c";
+			function = "hscif3";
+		};

		...
 };

 ...

+&hscif3 {
+		pinctrl-0 = <&hscif3_pins>;
+		pinctrl-names = "default";
+
+		uart-has-rtscts;
+		status = "okay";
+};
```

5.6 Changing the Kernel Configuration

For details on how to change the kernel configuration, see the document for each development environments:

- GitHub-based development environment/VLP64:
    1. Making kernel configuration fragment (`.cfg`). Refer to https://www.yoctoproject.org/docs/2.4.3/kernel-dev/kernel-dev.html#creating-config-fragments for the details.
        A. By executing **`bitbake linux-renesas -c menuconfig`**, you can edit the kernel configuration with `menuconfig`. After that, the command **`bitbake linux-renesas -c diffconfig`** will makes the fragment.cfg under the `$WORK/build/tmp/work/[machine name]-poky-linux/linux-renesas/4.19...(snip).../`.
        B. Writing kernel configuration fragment manually. For example, a command **`echo "CONFIG_USB_VIDEO_CLASS=y" >> fragment.cfg`** makes the simple kernel configuration fragment.
    2. Copy the fragment under the directory `$WORK/meta-rzg2/recipes-kernel/linux/linux-renesas/` . And rename it for the ease of configurations management.
    3. Add following lines to `$WORK/meta-rzg2/recipes-kernel/linux/linux-renesas_4.19.bb` to include the fragment to the sources of kernel configuration:

```
SRC_URI_append = " \
    file://fragment.cfg \
"
```

5.7 Building the Linux Kernel

For details on how to build the Linux kernel, see the document for each development environments:

- GitHub-based development environments/downloaded VLP: After building the kernel with the command **`bitbake -C compile linux-renesas`**, the symbolic links to kernel image and device tree are available as shown on Table 5.2.
    — Note that all modifications to the kernel sources and configurations should be saved as Yocto recipe.

Table 5.2 Symbolic link path for images and devicetrees

Processor	LSI Version	Board	Board Revision	Symbolic link path after **`$WORK/build/tmp/deploy/images/`**
G2H	3.0	HiHope-G2H	4	• hihope-rzg2h/r8a774e1-hihope-rzg2h.dts • hihope-rzg2h/r8a774e1-hihope-rzg2h-ex.dts • hihope-rzg2h/r8a774e1-hihope-rzg2h-ex-idk-1110wr.dts • hihope-rzg2h/r8a774e1-hihope-rzg2h-ex-mipi-2.1.dts • hihope-rzg2h/r8a774e1-hihope-rzg2h-ex-mipi-2.4.dts
G2M	1.2	HiHope-G2M	2	• hihope-rzg2m/r8a774a1-hihope-rzg2m-rev2.dts • hihope-rzg2m/r8a774a1-hihope-rzg2m-rev2-ex.dts

				• hihope-rzg2m/r8a774a1-hihope-rzg2m-rev2-ex-idk-1110wr.dts • hihope-rzg2m/r8a774a1-hihope-rzg2m-rev2-ex-mipi-2.1.dts • hihope-rzg2m/r8a774a1-hihope-rzg2m-rev2-ex-mipi-2.4.dts
	1.3		4	• hihope-rzg2m/r8a774a1-hihope-rzg2m.dts • hihope-rzg2m/r8a774a1-hihope-rzg2m-ex.dts • hihope-rzg2m/r8a774a1-hihope-rzg2m-ex-idk-1110wr.dts • hihope-rzg2m/r8a774a1-hihope-rzg2m-ex-mipi-2.1.dts • hihope-rzg2m/r8a774a1-hihope-rzg2m-ex-mipi-2.4.dts
	3.0		4	• hihope-rzg2m/r8a774a3-hihope-rzg2m.dts • hihope-rzg2m/r8a774a3-hihope-rzg2m-ex.dts • hihope-rzg2m/r8a774a3-hihope-rzg2m-ex-idk-1110wr.dts • hihope-rzg2m/r8a774a3-hihope-rzg2m-ex-mipi-2.1.dts • hihope-rzg2m/r8a774a3-hihope-rzg2m-ex-mipi-2.4.dts
G2N	1.1	HiHope-G2N	2	• hihope-rzg2n/r8a774b1-hihope-rzg2n-rev2.dts • hihope-rzg2n/r8a774b1-hihope-rzg2n-rev2-ex.dts • hihope-rzg2n/r8a774b1-hihope-rzg2n-rev2-ex-idk-1110wr.dts • hihope-rzg2n/r8a774b1-hihope-rzg2n-rev2-ex-mipi-2.1.dts • hihope-rzg2n/r8a774b1-hihope-rzg2n-rev2-ex-mipi-2.4.dts
			4	• hihope-rzg2n/r8a774b1-hihope-rzg2n.dts • hihope-rzg2n/r8a774b1-hihope-rzg2n-ex.dts • hihope-rzg2n/r8a774b1-hihope-rzg2n-ex-idk-1110wr.dts • hihope-rzg2n/r8a774b1-hihope-rzg2n-ex-mipi-2.1.dts • hihope-rzg2n/r8a774b1-hihope-rzg2n-ex-mipi-2.4.dts
G2E	1.0	EK874	B	• ek874/r8a774c0-es10-cat874.dts • ek874/r8a774c0-es10-ek874.dts • ek874/r8a774c0-es10-ek874-idk-2121wr.dts • ek874/r8a774c0-es10-ek874-mipi-2.1.dts • ek874/r8a774c0-es10-ek874-mipi-2.4.dts
	1.1		C	• ek874/r8a774c0-cat874-revc.dts • ek874/r8a774c0-ek874-revc.dts • ek874/r8a774c0-ek874-revc-idk-2121wr.dts • ek874/r8a774c0-ek874-revc-mipi-2.1.dts • ek874/r8a774c0-ek874-revc-mipi-2.4.dts
			E	• ek874/r8a774c0-cat874.dts • ek874/r8a774c0-ek874.dts • ek874/r8a774c0-ek874-idk-2121wr.dts • ek874/r8a774c0-ek874-mipi-2.1.dts • ek874/r8a774c0-ek874-mipi-2.4.dts

5.8 Examples of Adding Devices/Kernel functions

This section provides examples of adding several peripheral devices.

5.8.1 Adding an I2C Device

If you are to connect an I2C device to the I2C or IIC module in the RZ/G2 on the custom board, proceed with the following steps.

- Settings of the I2C or IIC module in the RZ/G2
    - Enable the I2C or IIC module to be used (see section 5.5.4 Enabling or Disabling Existing Devices).

```
Change the pin configuration if you wish to use a configuration which differs
from that for use with the reference board (see section 0
```

- Customizing the pin multiplex).

- Settings of the I2C device connected to the RZ/G2
    - If the kernel does not include a driver for the I2C device, add the driver (see section 5.5.7 Adding, Deleting, and Modifying Devices).
    - Add the I2C device to the device tree (see section 5.5.7 Adding, Deleting, and Modifying Devices).
    - Change the kernel configuration so as to enable the driver for the I2C device (see section 5.6 Changing the Kernel Configuration).

Examples of the connection of I2C devices are given in sections 5.8.1.1 Using the I2C Connection to Add an RTC and 5.8.1.2 Using the I2C Connection to Add an EEPROM Device.

5.8.1.1 Using the I2C Connection to Add an RTC

Using the I2C connection to add a realtime clock (RTC) is described here. In this example, RTC chip BQ32000 Series will be connected to I2C channel 2.

Add the following node to the device tree. For details on how to enter a node in the device tree, see the `Documentation/devicetree/bindings/rtc/ti,bq32k.txt`, which is an explanatory note on the device tree for the BQ32000 Series. In this example, the i2c2 interface is used as the I2C module for connection.

```
&i2c2{ /* Select the appropriate master for connection */
          ...
          rtc@68 {
                         compatible = "bq32000";
                         reg = <0x68>;
          };
          ...
};
```

To enable the driver for the BQ32000 Series, set `CONFIG_RTC_DRV_BQ32K = y` (for embedding the driver in the kernel) or `CONFIG_RTC_DRV_BQ32K = m` (for handling the driver as a loadable kernel module) in the kernel configuration.

5.8.1.2 Using the I2C Connection to Add an EEPROM Device

Using the I2C connection to add an EEPROM device is described here. The file `drivers/misc/eeprom/at24.c` is used as the driver for a general EEPROM with I2C interface. For details on how to enter a node in the device tree, see the `Documentation/devicetree/bindings/eeprom/eeprom.txt`. Followings is the example of using Renesas I2C interface EEPROM R1EX24002ATAS0I.

Add the following node to the device tree. In this example, the i2c2 interface is used as the I2C module for connection. Take care with the following points.

- In this example, the EEPROM is connected to the i2c2 interface. Select the actual destination for connection of the given device.
- 0x50 represents the address of the device. Select the address to suit the setting on the device side.
- In this example, the page size is set to 16 bytes (not essential). Set the size to suit the specifications of the given device.
- In this example, read-only is set as the type of read or write operation (not essential). Select the operation to suit the intended use of the given device.

```
&i2c2 { /* Select the appropriate master for connection */
          ...
          eeprom@50 { /* Select the address */
                        compatible = “renesas,24c02”;
                        reg = <0x50>; /* Select the address */
                        pagesize = <16>; /* Optional: Default is 1. */
                        read-only; /* optional */
          };
          ...
};
```

To enable the driver `drivers/misc/eeprom/at24.c`, set `CONFIG_EEPROM_AT24 = y` (for embedding the driver in the kernel) or `CONFIG_EEPROM_AT24 = m` (for handling the driver as a loadable kernel module) in the kernel configuration.

Revision History	RZ/G2 Group Linux BSP Porting Guide

Rev.	Date	Description	
		Page	Summary
1.00	Oct. 29, 2019	—	Initial release
1.01	Jan. 10, 2020	38,39	Add section 5.5.3 Customizing CMA
1.02	Jun. 15, 2020	—	Add support G2H
1.03	Aug. 17, 2020	—	Add support G2M v3.0
1.04	Nov. 16, 2020	—	Change version to keep consistent with other documents.
1.05	Feb. 26, 2021	8, 9, 10, 16, 17	Update change of TF-A Yocto recipe and source code. Update Renesas webpage URL to download VPL64 package.
1.06	May. 31, 2021	—	Add support EK874 Revision E Board (RZ/G2E)
1.07	Aug. 31, 2021	8	Add information about RZ/G2's TF-A source code from v2.5
1.08	Nov. 15, 2021	—	Change version to keep consistent with other documents.
1.09	Feb. 28, 2022	46, 47	Add new devicetree files for MIPI Adapter v2.4
1.10	May. 31, 2022	—	Update information about new U-boot version v2021.10

RZ/G2 Group Linux BSP Porting Guide

Publication Date: Rev.1.10 May. 31, 2022

Published by: Renesas Electronics Corporation

RZ/G2 Group Linux BSP Porting Guide

Renesas Electronics Corporation