RZ/A1

RZ/A1 Linux BSP Porting Guide

EU_00xxx
Rev.1.02
Oct 24, 2016

Introduction

The purpose of this document is to explain how to modify the Linux-3.14 BSP for the RZ/A1 RSK board to work with your own board.

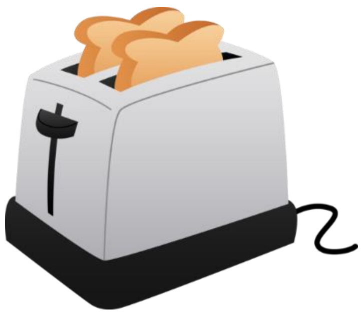
Target Device

RZ/A1L, RZ/A1M, RZ/A1H

Contents

1. The RZ Toaster

1.1 Naming



For all of the examples in this document, we will use a fake board name of "**rztoaster**" that we are porting to (because we had to pick some type of name). Therefore, anywhere you see "rztoaster" name, you should replace that with whatever your board name is.

When choosing a name, it is best to use a simple, single word name without hyphens '-' or underscores '_' in the name.

The text in blue will be text you need to add/modify.

The text in red will be the sample 'rztoaster' name from the company 'my_company' that you should choose for yourself.

Make sure you keep the case (UPPER and lower) the same as in the examples.

1.2 Git Cloning the Public Repositories

This is important.

Driver code is never perfect. As we are constantly finding/fixing bugs, or adding in new functionality and features, it is recommended that you base your project off the public github repositories using **git**. This way it will be much easier to merge in the latest code updates from Renesas. To do this, please perform the following steps in setting up your initial build environment on your host build machine. The "$" signifies the command line prompt in your Linux machine.

If you do not have "git" installed on your machine, please install it by entering the following commands in Ubuntu:

```
$ sudo apt-get install git
```

Now enter the following commands to "clone" and download the 3 main repositories for RZ/A1 Linux.

Note that the **u-boot** (u-boot-2015.01.git) and **kernel** (linux-3.14.git) repository clones are embedded under the BSP build environment (rskrza1_bsp.git) repository clone. Remember to replace the string "rztoaster" with what you want to call your bsp.

```
$ cd ~    (or...wherever you want to hold your custom bsp)

$ git clone https://github.com/renesas-rz/rskrza1_bsp.git  rztoaster_bsp
$ cd rztoaster_bsp
$ mkdir output
$ cd output
$ git clone https://github.com/renesas-rz/linux-3.14.git
$ git clone https://github.com/renesas-rz/u-boot-2015.01.git
$ cd ..
```

To pull in updates for your 3 repositories, you would execute the following commands from your custom BSP directory:

```
$ cd rztoaster_bsp           #  (or whatever you called it)
$ git pull                   # This updates your build environment scripts

$ cd output/linux-3.14       # The location of your git cloned kernel source
$ git pull                   # This updates your kernel source
$ cd ../..

$ cd output/u-boot-2015.01   # The location of your git cloned u-boot source
$ git pull                   # This updates your u-boot source
$ cd ../..
```

Note: If you plan on using these cloned git repositories to keep track of local modifications (ie, "git commit") it is recommended that you create local "branches" for both u-boot and kernel so that you can receive updates to the *master* branch (on github) and then merge in these updates to your custom BSP local branches.

2. u-boot: Converting RZ/A1 RSK BSP to your own platform

We suggest copying and renaming the RSK files before modifying them for your own specific board. This will allow you to work in parallel to the official Renesas RSK BSP.

2.1 Create a Machine (Board) Name and Number

Create a name for your board and pick a new number and add it to the file: **arch/arm/include/asm/mach-types.h**

```
 #define MACH_TYPE_COLIBRI_T30          4493
 #define MACH_TYPE_APALIS_T30           4513
 #define MACH_TYPE_RSKRZA1              4533
 #define MACH_TYPE_RZTOASTER            5000
```

Note about the ARM Machine Registry: The names and numbers within this file actually come from this ARM Linux Machine registry:

http://www.arm.linux.org.uk/developer/machines/

While not mandatory, you can register your board for an official name and number. New names and numbers are automatically added to the next Linux kernel version (file arch/arm/tools/mach-types). Doing this does not mean you have to submit your code to Linux.org. However, if you were planning on updating your BSP to a later version of u-boot or Linux kernel, then you would want to register a number so you could keep the name number between versions. Although, if you never submit your code upstream to the official Linux kernel, eventually they will delete your board name and number from the list.

Then also add these lines in blue(and red) as well to the end of the same file.

NOTE: machine_is_rza1rsk() is not really used anyway.

```
#ifdef CONFIG_MACH_RZTOASTER
# ifdef machine_arch_type
#  undef machine_arch_type
#  define machine_arch_type      __machine_arch_type
# else
#  define machine_arch_type      MACH_TYPE_RZTOASTER
# endif
# define machine_is_rztoaster()   (machine_arch_type == MACH_TYPE_RZTOASTER)
#else
# define machine_is_rztoaster()   (0)
#endif

/*
 * These have not yet been registered
 */

#ifndef machine_arch_type
#define machine_arch_type  __machine_arch_type
#endif

#endif
```

2.2 Create your own configuration header file for your board

Copy the RSKRZA1 header file, then modify it for your own board. This file will contain what drivers or u-boot utilities you would like to include.

Use this command to make a copy of the file:

```
$ cp include/configs/rskrza1.h include/configs/rztoaster.h
```

Edit the new configuration file and change the line:

```
#define CONFIG_MACH_TYPE MACH_TYPE_RSKRZA1
```

to

```
#define CONFIG_MACH_TYPE MACH_TYPE_RZTOASTER
```

2.3 Make a copy of the RSKRZA1 board directory

Use the RZ/A RSK board files as a starting point for your specific board. You will want to make your modification outside of the rskrza1 directory so if you get BSP updates, the rskrza1 files will not conflict.

Also note that you should create a new directory for your company name under the board directory. In the example below

Use this command to copy the directory:

```
$ mkdir -p board/my_company/rztoaster
$ cp -a board/renesas/rskrza1/* board/my_company/rztoaster
```

Use this command to rename the rskrza1.c file to your board name:

```
$ mv board/my_company/rztoaster/rskrza1.c board/my_company/rztoaster/rztoaster.c
```

Use this command to modify the Makefile to use our renamed file:

```
$ sed -i 's/rskrza1.o/rztoaster.o/g' board/my_company/rztoaster/Makefile
```

Use this command to change machine type define:

```
$ sed -i 's/MACH_TYPE_RSKRZA1/MACH_TYPE_RZTOASTER/g' board/my_company/rztoaster/rztoaster.c
```

2.4 Create a defconfig file

Copy the current defconfig file with this command:

```
$ cp configs/rskrza1_defconfig configs/rztoaster_defconfig
```

In the file, change RSKRZA1 to RZTOASTER

```
$ sed -i 's/RSKRZA1/RZTOASTER/g' configs/rztoaster_defconfig
```

2.5 Change the Kconfig files

Use these commands to replace ‘RSKRZ1’ and ‘rskrza1’ with your board name inside your Kconfig file. You also need to change ‘renesas’ to ‘my_company’ to match your new company directory name.

```
$ sed -i 's/RSKRZA1/RZTOASTER/g' board/my_company/rztoaster/Kconfig
$ sed -i 's/rskrza1/rztoaster/g' board/my_company/rztoaster/Kconfig
$ sed -i 's/renesas/my_company/g' board/my_company/rztoaster/Kconfig
```

Edit the file:

`arch/arm/cpu/armv7/rza1/Kconfig`

and add in your board (example below in blue/red)

```
$ gedit arch/arm/cpu/armv7/rza1/Kconfig
```

Add the text in blue/red

```
choice
        prompt "Renesas RZ/A1H ARM SoCs board select"

config TARGET_RSKRZA1
        bool "RSK-RZA/1 board"
config TARGET_RZTOASTER
        bool "RZ Toaster board"

endchoice

config SYS_SOC
        default "rza1"

source "board/renesas/rskrza1/Kconfig"
source "board/my_company/rztoaster/Kconfig"

endif
```

2.6 Build your new system

Now you should be able to test building your system. Since you didn't really change any code, it should build OK.

NOTE: Before you can use 'make', you will need to set up the environment variables such as PATH, ARCH and CROSS_COMPILE. If you are using the Renesas BSP, you can use the setup_env.sh file. Make sure you 'source' the file, not just run it. You only have to source it once per terminal window you open.

```
$ cd ../..      # cd back to base of rskrza1 bsp
$ export ROOTDIR=$(pwd) ; source ./setup_env.sh
$ cd -          # cd back to where you were
```

```
$ make distclean
$ make rztoaster_config
$ make
```

3. u-boot: I/O Pin setup

All the I/O pin mux setup is in board/my_company/rztoaster/rztoaster.c.

Most of the pin setup is done in function board_early_init_f( ). This function is call very early in boot so you can set up all your pins before you try to do things like set up external SDRAM or SPI flash. The "_f" in the function name is there to tell you that this function will run from flash, which makes sense because u-boot can't relocated itself to RAM until the RAM is set up, and the RAM could be external (although by default, we just use internal RAM because its faster and some systems might not even have external SDRAM). With that said, there is also another function board_early_init_r( ) that is still only run once during boot, but later after u-boot has been relocated to RAM.

If you are booting with the MODE pins set to boot from serial flash (QSPI), then some of the QSPI pin will automatically set for you. Basically, only the ones for doing normal single wire SPI. If you want to do the full Quad SPI, or dual Quad SPI, you will need to set those pins up in the board_early_init_f( ) function.

Please make note, the port for the QSPI pins are different between the RZ/A1L and RZ/A1H. So please adjust the code accordingly.

4. u-boot: Change Internal RAM Size

The RZ/A1 RSK board has a RZ/A1H part with 10MB internal RAM. For RZ/A1L (3MB) and RZ/A1M (5MB) users, make sure you remember to change the size of the internal RAM to 3MB (or 5MB) if you are using a RZ/A1L or RZ/A1M part on the CONFIG_SYS_SDRAM_SIZE line

For example:

```
#ifdef USE_INTERNAL_RAM
#define CONFIG_SYS_SDRAM_BASE           0x20000000
#define CONFIG_SYS_SDRAM_SIZE           (3 * 1024 * 1024)
#define CONFIG_SYS_INIT_SP_ADDR        (CONFIG_SYS_SDRAM_BASE +
CONFIG_SYS_SDRAM_SIZE - 1*1024*1024)
#else
#define CONFIG_SYS_SDRAM_BASE           0x08000000
#define CONFIG_SYS_SDRAM_SIZE           (32 * 1024 * 1024)
#define CONFIG_SYS_INIT_SP_ADDR        (0x09F00000)
#endif
```

NOTE: We suggest you always use USE_INTERNAL_RAM setting, even if plan on having external SDRAM on your system. There is really no benefit to running u-boot from external SDRAM instead of internal RAM. The option of selecting external SDRAM for u-boot is in the configuration file simply for reference in case someone ever has some reason why they need to do this.

5. u-boot: Clock setup

The RSK board uses a 13.333MHx XTAL for a 400MHz CPU Instruction clock (Iφ) and 66MHz Peripheral clock 1 (P1φ) and a 33MHz Peripheral clock 0 (P0φ).

If you are not using a 13.33 MHz XTAL, please adjust the following settings in your **include/configs/rztoaster.h**

```
/* Board Clock */
#define CONFIG_SYS_CLK_FREQ    66666666 /* P1 clock. */
#define CONFIG_SYS_HZ          1000
```

```
/* Set clocks based on 13.3333MHz xtal */
#define FRQCR_D          0x1035      /* CPU= 300-400 MHz */
```

As well as the timer calculations that are based off of the P0 clock in file **arch/arm/cpu/armv7/rza1/timer.c**

```
static unsigned long get_usec (void)
{
      ...
      ...
      /* Timer source clock (P0) is 33.33 Mhz */
      return (unsigned long)(ost0_timer / 33);
}
```

```
ulong get_timer(ulong base)
{
      const ulong timecnt = OST_TIMER_RESET / (33 * 1000); /*130150*/
```

6. u-boot: Serial Console Port Select

The RSK using serial channel 2 (SCIF2) for serial console. If your board uses something different, please make the correct change in file **include/configs/rztoaster.h** file (remember, your filename will be whatever you choose from the first step).

```
#define CONFIG_BAUDRATE        115200
```

```
/* Serial */
#define CONFIG_SCIF_CONSOLE
#define CONFIG_CONS_SCIF2
#define SCIF2_BASE                   0xE8008000
#define CONFIG_SH_SCIF_CLK_FREQ CONFIG_SYS_CLK_FREQ
```

7. u-boot: Initial Debugging using RAM Only (not Flash)

Before you attempt to program SPI Flash (or NOR Flash), you should consider starting out with just trying to load u-boot into RAM with the JTAG. Then running it from there and getting to a serial prompt (no Flash involved). Then, you could start to confirm pin setup and figure out the SPI flash programming and such.

These instructions walk you through using a Segger J-Link.

7.1 Verify I/O pin Setup

The first step would be to first make sure the pin mux settings make sense (see section I/O Pin setup).

7.2 Change Base Address

Next you will need to change the base address of where your code will run from (from the default address to the RAM address). You would do that in your **include/configs/rztoaster.h** file (remember, your filename will be whatever you choose from the first step). You will want to change the **CONFIG_SYS_TEXT_BASE** setting from 0x18000000 to **0x20020000** and rebuild.

NOTE: In the latest BSP code, there is now a SPI_FLASH_LOADER option that will do this for you. Simply uncomment the line:

```
 /* #define SPI_FLASH_LOADER */
```

If you are wondering why the RAM address 0x20020000 instead of 0x20000000, look at System Control Register 3 (SYSCR3) which says that the Data Retention RAM areas are read-only after reset until you enable them in code (which we do in the file lowlevel_init.S immediately after RESET). However, when you are trying to download your code using the J-Link, that code has not run yet so the RAM at address 0x20000000 is Read-Only, meaning you can't download to it. Therefore, we'll use address 0x20020000.

HELPFUL HINT If you want to simply comment out the CONFIG_SYS_TEXT_BASE line and make a copy of it, or any other line in this .h file, you cannot use the "//" C++ style comment markers, otherwise you will get a strange build error that might take you a while to figure out what is cause it. Therefore, just use the standard /*   */ comment markers for everything in that file.

7.3 Downloading (to RAM) and Running using J-LinkExe

With your board powered and your J-Link plugged in, start up JLink Commander (JLinkExe if using Linux or JLink.exe if using Windows) in the directory that has your u-boot.bin binary.

For the following examples, we'll use the Linux version….because we'll probably be bouncing back and forth between building and testing.

```
$ cd u-boot-2015.01
$ JLinkExe -speed 12000 -if JTAG -device R7S721001
```

NOTE: If using J-Link 5.10+, add **-jtagconf -1,-1** to the command line.

Next, from within the JLink window, issue commands to reset the device (rx 100), download your code (loadbin), set the PC to your code (setpc), then start it running (g), then exit Jlink (exit).

```
J-Link> rx 100
J-Link> loadbin u-boot.bin,0x20020000
J-Link> setpc 0x20020000
J-Link> g
J-Link> exit
```

Hopefully you'll get a serial prompt. If not, could use GDB.

7.4 Downloading (to RAM) and Running using J-Link and GDB

These instructions are for the RAM-based u-boot.

If you need to debug u-boot at this stage, you can use GDB. This sections assumes you have already read the application note "GDB Debugging for RZA1 Linux with J-Link v1.00".

Use the following commands to start debugging with the RAM based u-boot.

The 'monitor' command in GDB basically lets you send commands directly to the J-Link, so we can use the loadbin J-Link custom command from within GDB. Note that you have to use the full path to your u-boot.bin.

```
(gdb) target remote localhost:2331
(gdb) monitor reset
(gdb) file u-boot
(gdb) monitor loadbin ~/rskrza1_bsp/output/u-boot-2015.01/u-boot.bin,0x20020000
(gdb) set $pc = 0x20020000
(gdb) si
(gdb) si

    etc...
```

If you are doing a lot of rebuilding of u-boot and running, you could make a macro that you put inside your local .gdbinit file and will take care of loading u-boot and setting up all the addresses and such. For example, you could put the following in your .gdbinit file, and then if you build a new u-boot, you simply just type "ramload" in GDB and it will reload it for you.

```
define ramload
        #Get rid of any existing breakpoints
        delete

        #Load in our file/binary
        #NOTE: that you need the full path of your file!!
        monitor reset
        file u-boot
        monitor loadbin /home/renesas/u-boot-2015.01/u-boot.bin,0x20020000

        #Run the code till right before the relocation
        set $pc = 0x20020000
        tb relocate_code
        c
        shell sleep 1

        #Figure out where it will be copying to, then reload symbol file
        set $i = gd->relocaddr
        print $i
        symbol-file
        add-symbol-file u-boot $i

        #Set a 1-time breakpoint at board_init_r()
        tb board_init_r
        c

        #At the end, you should be in the board_init_r function.
        #Now you can set breakpoint or whatever you want, or continue to run
end
```

8. u-boot: Adding QSPI Support

8.1 Add Support for your SPI Flash

When you first try to connect to your SPI flash, you might get a message like this:

```
=> sf probe 0
SF: Unsupported flash IDs: manuf 20, jedec ba20, ext_jedec 1000
Failed to initialize SPI flash at 0:0
```

This means you need to add the manufacture of your SPI flash device.

You can look up what the SPI flash devices are supported and match up yours by looking at file:

```
      u-boot-2015.01/drivers/mtd/spi/sf_params.c
```

For the example above you will see that line

```
        {"N25Q512",       0x20ba20, 0x0,       64 * 1024,  1024, RD_FULL, WR_QPP | E_FSR | SECT_4K},
```

is only included if CONFIG_SPI_FLASH_STMICRO is defined.

Therefore, you simply add that Flash manufacture to your **include/configs/rztoaster.h** file

You can check file

u-boot-2015.01/drivers/mtd/spi/sf_params.c

For example:

```
/* Spi-Flash configuration */
#define CONFIG_RZ_SPI
#define CONFIG_SPI_FLASH
#define CONFIG_SPI_FLASH_SPANSION
#define CONFIG_SPI_FLASH_STMICRO
#define CONFIG_RZA1_BASE_QSPI0            0x3FEFA000
#define CONFIG_SPI_FLASH_BAR        /* For SPI Flash bigger than 16MB */
```

Now, after you rebuild and download again, you should see something like this:

```
=> sf probe 0
SF: Detected N25Q512 with page size 256 Bytes, erase size 4 KiB, total 64 MiB
```

8.2 u-boot: QSPI XIP Support

If plan on using the QSPI in XIP mode for Linux, then you will need to make sure the commands and device settings are correct. In your board/my_company/rztoaster/rztoaster.c file, you will see a function **do_qspi**( ). Please review that function and confirm the setup will work for the SPI flash device you have selected.

In the BSP, a Spansion and Micron device has been tested. If you are not using one of those device, hopefully they are similar enough where you only need to make minor modifications. Many device manufactures follow the same conventions as the Spansion devices on the RSK board. Micron devices tend to be different, so they have been included in the RSK BSP as an example.

9. u-boot: Programming QSPI Flash

Please refer to section **u-boot: Initial Debugging using RAM Only (not Flash)** on what a RAM-based u-boot is.

9.1 Using 'RAM-based' u-boot to program QSPI flash

If you have successfully confirmed that your RAM-based u-boot can erase and program SPI flash memory, then you can use it to program the real Flash-based u-boot into SPI flash.

First, save your working RAM based u-boot.bin

```
$ cp u-boot.bin u-boot-ram.bin
```

Now edit your configuration file and put your CONFIG_SYS_TEXT_BASE setting back to 0x18200000 and re-build.

Once again, connect with J-Link and download your RAM-based u-boot and get it up and running. However, this time, do not exit the JLinkExe program, but instead download the Flash based u-boot.bin to RAM as well. It's OK to use the low RAM address (0x20000000) because by this point, u-boot has already relocated itself into the upper part of your memory so there will not be any conflict. Also, the u-boot will have already enabled the RAM below address 0x20020000.

Then you can use the serial flash commands to erase the write the Flash-based u-boot into SPI flash.

For example:

```
$ cd u-boot-2015.01
$ JLinkExe -speed 12000 -if JTAG -device R7S721001
```

NOTE: If using J-Link 5.10+, add **-jtagconf -1,-1** to the command line.

Next, from within JLinkEXE window:

```
J-Link> rx 100
J-Link> loadbin u-boot-ram.bin,0x20020000
J-Link> setpc 0x20020000
J-Link> g
```

Confirm that u-boot comes up in your serial terminal. Then, go back to your JLinkExe session and download you Flash-based u-boot to RAM. You will need to GO ('g') command because JLink automatically halts the CPU when a loadbin command is used.

```
J-Link> loadbin u-boot.bin,0x20000000
J-Link> g
J-Link> exit
```

Now, in your u-boot console, program in your u-boot.

```
=> sf probe 0
SF: Detected N25Q512 with page size 256 Bytes, erase size 4 KiB, total 64 MiB
=> sf erase 0 80000
SF: 393216 bytes @ 0x0 Erased: OK
=> sf write 20000000 0 80000
SF: 393216 bytes @ 0x0 Written: OK
=> md 18000000
18000000: ea0000be e59ff014 e59ff014 e59ff014    ................
18000010: e59ff014 e59ff014 e59ff014 e59ff014    ................
18000020: 18000060 180000c0 18000120 18000180    `....... .......
18000030: 180001e0 18000240 180002a0 deadbeef    ....@...........
18000040: 0badc0de e320f000 e320f000 e320f000    ...... ... ... .
18000050: e320f000 e320f000 e320f000 e320f000    .. ... ... ... .
18000060: e51fd028 e58de000 e14fe000 e58de004    (.........O.....
18000070: e3a0d013 e169f00d e1a0e00f e1b0f00e    ......i.........
18000080: e24dd048 e88d1fff e51f2050 e892000c    H.M.....P ......
18000090: e28d0048 e28d5034 e1a0100e e885000f    H...4P..........
180000a0: e1a0000d eb0002c7 e320f000 e320f000    .......... ... .
180000b0: e320f000 e320f000 e320f000 e320f000    .. ... ... ... .
180000c0: e51fd088 e58de000 e14fe000 e58de004    ..........O.....
180000d0: e3a0d013 e169f00d e1a0e00f e1b0f00e    ......i.........
180000e0: e24dd048 e88d1fff e51f20b0 e892000c    H.M...... ......
180000f0: e28d0048 e28d5034 e1a0100e e885000f    H...4P..........
=>
```

HINT: If you are using external SDRAM, then after the RAM-u-boot runs, your SDRAM has been setup. Therefore, you could use that address to download you binary file to. This is helpful when you start to program in bigger binaries like the Linux kernel and root file system.

9.2 Automating Programing using a JLink Script

You can automate the process of programming the SPI flash by creating a script file that you pass to JLinkEXE.

First create a script file called **load_spi_uboot.txt** in the directory where your u-boot.bin file is located.

```
$ cd u-boot-2015.01
$ gedit load_spi_uboot.txt
```

Contents of load_spi_uboot.txt

```
rx 100
loadbin u-boot-ram.bin,0x20020000
setpc 0x20020000
g
Sleep 2000
loadbin u-boot.bin,0x20000000
g
exit
```

Now run your script:

```
$ JLinkExe -speed 12000 -if JTAG -device R7S721001 -CommanderScript load_spi_uboot.txt
```

NOTE: If using J-Link 5.10+, add **-jtagconf -1,-1** to the command line.

After your script is complete, your RAM-u-boot should be running and your new image to be programmed should be at the beginning of RAM. Now, you just need to program it in:

```
=> sf probe 0 ; sf erase 0 80000 ; sf write 20000000 0 80000
SF: Detected N25Q512 with page size 256 Bytes, erase size 4 KiB, total 64 MiB
SF: 524288 bytes @ 0x0 Erased: OK
SF: 524288 bytes @ 0x0 Written: OK
=>
```

10. u-boot: Some notes about your board specific file

Since you copied the rskrza1.c file to make your product specific rztoaster.c file (or whatever you called it), that means there will be a lot of extra BSP board only stuff in there that you can go ahead and delete. Here are some notes about what is in that file:

11. Kernel: Converting RZ/A1 RSK BSP to your own platform

We suggest copying and renaming the RSK files before modifying them for your own specific board. This will allow you to work in parallel to the official Renesas RSK BSP.

For all of these examples we'll assume a board name of "**rztoaster**". Therefore, anywhere you see that name, you can replace it with whatever your real board name is.

The text in blue will be text you need to add/modify.

The text in red will be the sample 'rztoaster' name that you can choose for yourself.

11.1 Create a Machine (Board) Name

Use the same name and number for your board that you created for u-boot in section **2.1 Create a Machine (Board) Name and Number** and add a line to the file:

arch/arm/tools/mach-types

They are in numerical order, so you can just add it to the bottom of the file, for example:

arch/arm/include/asm/mach-types.h

```
. . .
eukrea_cpuimx28sd MACH_EUKREA_CPUIMX28SD  EUKREA_CPUIMX28SD       4573
domotab           MACH_DOMOTAB            DOMOTAB                 4574
pfla03            MACH_PFLA03             PFLA03                  4575
rztoaser          MACH_RZTOASTER          RZTOASER                5000
```

11.2 Create a KConfig Entry

Now that you will have a new 'platform', you will need to have some way of selecting it in the kernel build system. To do that, you will need to add a new entry.

Edit the file **arch/arm/mach-shmobile/Kconfig** as follows:

```
config MACH_GENMAI
      bool "Genmai board"
      depends on ARCH_R7S72100
      select USE_OF

config MACH_RSKRZA1
      bool "RZ/A1 RSK board"
      depends on ARCH_R7S72100
      select USE_OF

config MACH_RZTOASTER
      bool "RZ/A1 Toaster board"
      depends on ARCH_R7S72100
      select USE_OF
```

Now, in *menuconfig*, will be able to select your device. But, you don't have to do that now…later we'll create a new default config file that will select your board for you.

11.3 Create a Device Tree for your board

In the kernel, devices are moving away from defining your devices in code (usually in your *board* file) and instead defining what devices are in your system in what is called a Device Tree file that is passed the kernel at boot time. The idea is that you could select from multiple different board definitions without having to rebuild the kernel.

The Renesas BSP kernel that is based off of Linux-3.14 is a mix of device tree definitions and legacy board (source code) definitions. So, you will need to make a Device Tree for your board, but you will still need to do modifications in your board file.

Make a copy of the **file arch/arm/boot/dts/r7s72100-rskrza1.dts** and name it according to your board name.

```
$ cp arch/arm/boot/dts/r7s72100-rskrza1.dts arch/arm/boot/dts/r7s72100-rztoaster.dts
```

You will also need to add that new file to the Makefile in charge of building the device trees.

arch/arm/boot/dts/Makefile

```
dtb-$(CONFIG_ARCH_SHMOBILE_LEGACY) += r7s72100-genmai.dtb \
      r7s72100-rskrza1.dtb \
      r7s72100-rztoaster.dtb \
      r8a7740-armadillo800eva.dtb \
```

Now in that new file, change all the instances of "rskrza1" and "RSKRZA1" to "rztoaster" and "RZTOASTER"

For example, you can use 'sed' to do that replace for you on the command line

```
$ sed -i 's/rskrza1/rztoaster/g' arch/arm/boot/dts/r7s72100-rztoaster.dts

$ sed -i 's/RSKRZA1/RZTOASTER/g' arch/arm/boot/dts/r7s72100-rztoaster.dts
```

11.4 Make a Copy of the RSKRZA1 "Board" file

The board file is where most of your board specific configurations are going to go. Use the RZ/A1 RSK board files as a starting point. You'll end up modifying this file a lot, so it's good to do that all in a new file. Also, if you have cloned the Renesas BSP from the git repository, you can pull new updates (without creating conflicts) and then compare them against your version.

```
$ cp arch/arm/mach-shmobile/board-rskrza1.c arch/arm/mach-shmobile/board-rztoaster.c
```

Now, you will need to edit the Makefile in order to build that file when your platform is selected.

arch/arm/mach-shmobile/Makefile

```
obj-$(CONFIG_MACH_GENMAI)      += board-genmai.o
obj-$(CONFIG_MACH_RSKRZA1)     += board-rskrza1.o pfc-rza1.o
obj-$(CONFIG_MACH_RZTOASTER)   += board-rztoaster.o pfc-rza1.o
obj-$(CONFIG_MACH_MARZEN)      += board-marzen.o
```

11.5 Modify Your Board File to use Your New Machine Name

Since you just copied the board file from the RSKRZA1, you now have to go in and change all the references to the machine name ("rskrza1") to your new machine name. We will also go and change some function name as well.

The "rskrza1" string shows up in many places in this file, but the list below is the most important instances need to change.

Current Function Name	New Function Name
rskrza1_add_standard_devices	rztoaster_add_standard_devices
rskrza1_init_late	rztoaster_init_late

At the very end of the file, modify the compatibility string and DT_MACHINE definition by replacing rskrza1 with your machine name. Make sure you replace everything below in red.

```
static const char * const rskrza1_boards_compat_dt[] __initconst = {
        "renesas,rztoaster",
        NULL,
};
DT_MACHINE_START(RZTOASTER_DT, "rztoaster")
        .init_early         = r7s72100_init_early,
        .init_machine       = rztoaster_add_standard_devices,
        .init_late          = rztoaster_init_late,
        .dt_compat          = rztoaster_boards_compat_dt,
        .map_io             = r7s72100_map_io,
        .restart            = r7s72100_restart,
MACHINE_END
```

11.6 Create Default Configuration Files

To configure the kernel properly, you'll need to know what you need to set…out of the hundreds of options. Therefore, it's best to just start with the BSP configuration and modify them from there.

Copy them as follows:

```
$ cp arch/arm/configs/rskrza1_defconfig  arch/arm/configs/rztoaster_defconfig

$ cp arch/arm/configs/rskrza1_xip_defconfig arch/arm/configs/rztoaster_xip_defconfig
```

Now, you need to change the "CONFIG_MACH_RSKRZA1" string to "CONFIG_MACH_RZTOASTER" in those file. For example:

```
$ sed -i 's/CONFIG_MACH_RSKRZA1/CONFIG_MACH_RZTOASTER/g' arch/arm/configs/rztoaster_defconfig

$ sed -i 's/CONFIG_MACH_RSKRZA1/CONFIG_MACH_RZTOASTER/g' arch/arm/configs/rztoaster_xip_defconfig
```

NOTE: The string "RSKRZA1" will appear other places in those files, but we don't want to change them.

11.7 Test Build

That this point, you should be able to do a test build of your system. Since you really didn't change any source file yet, it should compile

For a XIP kernel:

```
$ make rztoaster_xip_defconfig
$ make xipImage
$ make dtbs
```

For a non-XIP kernel:

```
$ make rztoaster_defconfig
$ make uImage LOADADDR=0x08008000
$ make dtbs
```

12. Kernel: Enable (Early) Low Level Debugging for Linux Booting

If your kernel is crashing before it is finish booting up, and before the serial console driver is initialized (which is about ½ way through boot), then you will not get any messages out of the serial port. Therefore, it's helpful to enable the early print feature of the kernel.

In the kernel config, you'll want to enable **CONFIG_DEBUG_LL** and **CONFIG_EARLY_PRINTK.**

The default serial port is SCIF2 (same as the RSK). If you are not using that serial port, you'll have to manually add your SCIF to the build system, then go back and select it (under "Kernel Hacking >> Kernel low-level debugging port".

For example, here is to add SCIF3:

In file **arch/arm/Kconfig.debug**

```
	config DEBUG_R7S72100_SCIF2
		bool "Kernel low-level debugging messages via SCIF2 on R7S72100"
		depends on ARCH_R7S72100
		help
		  Say Y here if you want kernel low-level debugging support
		  via SCIF2 on Renesas RZ/A1H (R7S72100).

	config DEBUG_R7S72100_SCIF3
		bool "Kernel low-level debugging messages via SCIF3 on R7S72100"
		depends on ARCH_R7S72100
		help
		  Say Y here if you want kernel low-level debugging support
		  via SCIF3 on Renesas RZ/A1H (R7S72100).
```

```
        default "debug/omap2plus.S" if DEBUG_OMAP2PLUS_UART
        default "debug/renesas-scif.S" if DEBUG_R7S72100_SCIF2
        default "debug/renesas-scif.S" if DEBUG_R7S72100_SCIF3
```

```
        default 0xe0000000 if ARCH_SPEAR13XX
        default 0xe8008000 if DEBUG_R7S72100_SCIF2
        default 0xe8008800 if DEBUG_R7S72100_SCIF3
```

```
        depends on DEBUG_LL_UART_8250 || DEBUG_LL_UART_PL01X || \
              DEBUG_LL_UART_EFM32 || \
              DEBUG_UART_8250 || DEBUG_UART_PL01X || DEBUG_R7S72100_SCIF2 \
              || DEBUG_R7S72100_SCIF3
```

Next you will need to add **earlyprintk** to the kernel command line boot arguments in order to enable it. You can do that by simply overriding the current default arguments that were prepared for the BSP. For example:

```
=> print xargs
xargs=console=ttySC2,115200 console=tty0 ignore_loglevel root=/dev/mtdblock0
=> set xargs 'console=ttySC2,115200 console=tty0 ignore_loglevel root=/dev/mtdblock0 earlyprintk'
=> print xargs
xargs=console=ttySC2,115200 console=tty0 ignore_loglevel root=/dev/mtdblock0 earlyprintk
=> run x_boot
. . .
```

13. Kernel: Editing Your Board File

The board file is where most of your board specific configurations are going to go. Since you copied the BSP file, you should now have the file:

```
    arch/arm/mach-shmobile/board-rztoaster.c
```

Should go through this file line by line and either remove devices, setup or functionality that is not relevant for your board.

Note that in this file are many structures that define the address location of registers and interrupt numbers for all the peripherals in the RZ/A1. These should not be changed as they are chip specific, not board specific. Since Linux drivers are written to be used by many different devices, the board file needs to specify this information when ‘registering’ the devices and drivers it would like to enable.

To make BSP porting easier, the comment “**BOARD:**” will appear throughout the file and will highlight portions of code that are board specific and should be reviewed. Some of these definitions are only relevant to the RSK board and can be removed, others may need to be modified if for example you are using a different pinout of a peripheral.

Please do a text search for “BOARD:” and review each comment.

Below are some additional comments to keep in mind when porting.

13.1 LCD Video

13.1.1 LCD Frame Buffer Address on RZ/A1L when using SDRAM

Make sure if you are using an RZ/A1L with external SDRAM, that you set the frame buffer address in the board file to something that is within the range of the internal 3MB RAM.

13.1.2 General Settings

- Set VDC5_BPP and VDC5_FBSIZE

13.1.3 Parallel LCD Output

- The BSP used a panel label “gwp0700cnwv04”. If you are not using that exact same panel, you should just do a text search-and-replace of that string with one that better matches your panel.
- If your board has a LCD panel, check the pins setup in the structure **lcd0_common[ ]** and **lcd0_tcon[ ]** .

13.1.4 LVDS LCD Output

- The BSP contains an LVDS example using a panel label “hsd070pww1”. If you are not using that exact same panel, you should just do a text search-and-replace of that string with one that better matches your panel.
- The LVDS pins only come out on Port 5, so you can use function vdc5fb_pinmux_lvds( ) without modification.
- The LVDS can be used with either VDC5 ch0 or ch1. LVDS is selected by setting “.use_lvds” to ‘1’ in your “struct vdc5fb_pdata”.

13.2 Pin Setup

While some pins were already setup in u-boot, those were only for functionality that would be used in u-boot. Basically all pin setup is done in the board file. Therefore, you should search for and review all the calls to function **r7s72100_pfc_pin_assign( )** and determine if they match your board.

14. Kernel: Clock settings

If you are not using a 13.33MHz crystal, here are some locations that you should change the clock definitions:

Platform Specific Device Tree File: arch/arm/boot/dts/r7s72100-rztoaster.dts

```
&extal_clk {
      clock-frequency = <13330000>;
};

&usb_x1_clk {
      clock-frequency = <48000000>;
};
```

RZ/A1 Common Clock File: arch/arm/mach-shmobile/clock-r7s2100.c

Since all RZ/A1 platforms use the same file, you use #ifdef in order to not break others.

```
static struct clk extal_clk = {
#ifdef CONFIG_RZTOASTER
      .rate       = 12000000, /* 12MHz  crystal */
#else
      .rate       = 13330000,
#endif
      .mapping    = &cpg_mapping,
};
```

Platform Specific Board File: arch/arm/mach-shmobile/board-rztoaster.c

```
#define     P1CLK             ((13330000 * 30) / 6)
#define     PIXCLOCK(hz, div)  \
      (u32)(1000000000000 / ((double)(hz) / (double)(div)))
```

15. Kernel: SDRAM on Chip Select 3

NOTE: This change has already been made in the Renesas Linux-3.14 BSP, so now this information is just for reference.

The RZ-RSK board uses SDRAM on CS2, so memory starts at 0x08000000. If you use CS3, memory will start at 0x0C000000.

However, if you create a uImage and try to boot that on CS3, it will crash. The reason is the uImage will always decompress to 0x08008000.

If you have CONFIG_AUTO_ZRELADDR=y defined in your kernel config, then the code below will run and given that mask value of 0xf8000000, you can never end up with a 0x0C000000…it will always be 0x08008000…and hence it will crash.

```
#ifdef CONFIG_AUTO_ZRELADDR
           @ determine final kernel image address
           mov  r4, pc
           and  r4, r4, #0xf8000000
           add  r4, r4, #TEXT_OFFSET
#else
           ldr  r4, =zreladdr
#endif
```

arch/arm/boot/compressed/head.S

Therefore, what you want to do is remove **CONFIG_AUTO_ZRELADDR** from your kernel config so it will set zreladd manually at build time.

The value of zreladd is set by editing the following file:

```
arch/arm/mach-shmobile/Makefile.boot
```

```
# per-board load address for uImage
loadaddr-y  :=
loadaddr-$(CONFIG_MACH_APE6EVM) += 0x40008000
loadaddr-$(CONFIG_MACH_APE6EVM_REFERENCE) += 0x40008000
loadaddr-$(CONFIG_MACH_ARMADILLO800EVA) += 0x40008000
loadaddr-$(CONFIG_MACH_ARMADILLO800EVA_REFERENCE) += 0x40008000
loadaddr-$(CONFIG_MACH_BOCKW) += 0x60008000
loadaddr-$(CONFIG_MACH_BOCKW_REFERENCE) += 0x60008000
loadaddr-$(CONFIG_MACH_GENMAI) += 0x08008000
loadaddr-$(CONFIG_MACH_RSKRZA1) += 0x08008000   << change to 0x0C008000
loadaddr-$(CONFIG_MACH_KOELSCH) += 0x40008000
loadaddr-$(CONFIG_MACH_KZM9G) += 0x41008000
loadaddr-$(CONFIG_MACH_KZM9G_REFERENCE) += 0x41008000
loadaddr-$(CONFIG_MACH_LAGER) += 0x40008000
loadaddr-$(CONFIG_MACH_MACKEREL) += 0x40008000
loadaddr-$(CONFIG_MACH_MARZEN) += 0x60008000

__ZRELADDR  := $(sort $(loadaddr-y))
   zreladdr-y   += $(__ZRELADDR)

# Unsupported legacy stuff
#
#params_phys-y (Instead: Pass atags pointer in r2)
#initrd_phys-y (Instead: Use compiled-in initramfs)
```

arch/arm/mach-shmobile/Makefile.boot

Of course the proper way is to make your own BSP and define your own machine id like CONFIG_MACH_MYBOARD, then simple add a new line like such:

```
loadaddr-$(CONFIG_MACH_MYBOARD) += 0x0C008000
```

NOTE: After removing CONFIG_AUTO_ZRELADDR, sometimes the file autoconf.h will not be regenerated. Therefore, make sure you do a full re-build of the kernel to forced a regeneration.

16. Misc: Installing J-Link software in Linux

To install the Segger Jlink drivers and utilities for Linux do the following. This example was for a 32-bit Ubuntu (Debian) Virtual Machine.

Download "Software and documentation pack for Linux V5.00i, DEB Installer 32-bit version" from

https://www.segger.com/jlink-software.html

In a command window, install the downloaded file:

```
$ sudo dpkg -i jlink_5.0.9_i386.deb
```

After the install, on a command line you can type "JLink" then hit the 'tab' key on your keyboard to see all the utilities it installed.

17. FAQ: Why the name SH-Mobile?

- Why the name SH-Mobile?

You may have noticed that the directory that this device is under the ARM directory is called SH-Mobile. This is because over the years, Renesas original supported SH4A and SH-Mobile devices in the kernel under the arch/sh directory. However, Renesas started making devices which had both SH4A and ARM cores in them, so support for those boards needed to be under the ARM tree when you wanted to build a kernel for the ARM core. Eventually, the SH4A cores were removed and only the ARM core remained, but almost all the peripheral drivers were still the same as the SH4A and SH-Mobile devices, hence the Ethernet driver is called sh-eth. Basically, you were building an ARM device with SH4A family peripherals. Renesas does have a plan to eventually rename the mach-shmobile directory under the ARM tree to something more generic like “mach-renesas”, but that will take some time to happen.

Revision Record

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Dec 22, 2015	—	First edition issued
1.01	April 15,2016	13	Missing line for SCIF3 example
		7,9,10	Use jtagconf for J-Link 5.10+
1.02	Oct 24, 2016	2	Add instructions to git clone from github