MCU Developer Guide

Release Version：1.1

E-mail：frank.wang@rock-chips.com

Release Date：2017.12

Classified Level：Publicity

Preface

This document introduces the Cortex-M0 Devices Generic User Guide.

Chipset Version

Name	Linux Kernel Version
RK3399	4.4

Intended audience

This document (this guide) is intended primarily for the following readers:

Field Application Engineer

Software Engineer

Revision history

Revision Date	Version	Author	Revision Description
2017-09-20	V1.0	Frank.Wang	Initial version
2017-12-27	V1.1	Frank.Wang	modify format

1 Rockchip MCU Overview

ARM® Cortex®-M processor family provide some benefits to developers, including: simple, easy-to-use, highly efficient, ultra-low power operation.

At the same time, the Cortex-M processor provides developers more functions at a lower cost, which has significant advantages in code reuse and development efficiency. For this advantage, it is widely used in the field of embedded devices. The simply introduction of Cortex-M0 and Cortex-M3 is as follows:

- The Cortex-M0 processor uses the ARMv6-M architecture based on a highly integrated, low-power 32-bit processor core; it uses the von Neumann architecture and base on the 16-bit Thumb instruction set included Thumb-2 technology.
- The Cortex-M3 is a 32-bit processor core used the ARMv7-M architecture. It uses Harvard architecture, which has an independent instruction bus and data bus, which allows the fetch and data access to be parallel.

Based on the above advantages of ARM® Cortex®-M, the MCUs currently integrated on the Rockchip SoC are described below:

- The RK3399 integrates two Cortex-M0s, "PMU M0" is used by ATF and the other "Perilp M0" are open to customers developed by themselves.

2 Primary Development

2.1 Configuration Preparing

This chapter mainly introduces the basic method of Rockchip MCU development based on RK3399 Perilp M0.

2.1.1 Boot Address

Take genetic boot way "miniloader + ATF + u-boot" as an example.

With this method, usually need to pack the MCU bin and ATF bin as "trust.img ". Therefore, the following package configuration is required to add to the U-Boot.

```
tools/rk_tools/RKTRUST/RK3399TRUST.ini
...
[BL30_OPTION]
SEC=1
PATH=tools/rk_tools/bin/rk33/rk3399bl30_v1.00.bin
ADDR=0x00080000
...
```

As above shows, "PATH" is the storage path of MCU bin file and "ADDR" is the DDR address where MCU load start (also means the MCU's start address). Of course, this address needs a secure address reserved in DDR. This address will be passed to the miniloader, which will load the MCU bin from ROM to this address of DDR.

Of course, if you use U-Boot or other loader as the first-level boot loader, you can also pack and load the M0 firmware referred to above method.

For the MCU bin compiling, refer to chapter3.2.2 Compilation.

2.1.2 Address mapping

The Cortex-M0/Cortex-M3 has a fixed Memory Map, which facilitates easy porting of software between different systems. The address space is divided into many different segments, you can refer to Chapter 2.2 chapter Memory model in Cortex-M0 Devices Generic User Guide and Chapter 7.4.2 in Rockchip RK3399 TRM. Commonly, we only need configuring the MCU's 0x00000000-0x1FFFFFFF address mapping.

It is note that the address mapping of the RK3399 M0 needs to be configured via SGRF, so please be sure to configure it in the module that can access the SGRF (usually in the miniloader or ATF). Take 2.1.1 Boot Address Load address as an example. The specific memory mapping configuration is as follows:

- Boot address configuration

```
sgrf_perilp_m0_con7 = 0xf << (4 + 16) | (0x080000 >> 28) & 0x0f
sgrf_perilp_m0_con15 = 0xffff << 16 | (0x80000 >> 12) & 0xffff
```

- Peripheral address configuration

The Rockchip MCU has configured the map address of the peripheral (0x40000000-0x5FFFFFFF) by default, which like:

```
ADDR_MCU = ADDR_CA72 – 0xB8000000
```

The peripherals here are those listed in the Rockchip RK3399 TRM Chapter 2 System Overview.

2.1.3 Clock Configuration

The Rockchip MCU clock source can be selected from CPLL or GPLL. Refer to the RK3399 TRM Chapter 3 CRU. In that chapter it is point out that the CLK configuration register is "CRU_CLKSEL_CON24 (0x0160)", where:

bit[15] : Select clock source，1’b0: CPLL；1’b1: GPLL
bit[12:8] : Set the frequency-division , which is used to configure the operating frequency of the MCU.

- U-Boot reference code

```
arch/arm/cpu/armv8/rk33xx/clock-rk3399.c
#ifdef CONFIG_PERILP_MCU
    /* peril m0 clk = 300MHz, select gpll as the source clock */
    clk_parent_hz = RKCLK_GPLL_FREQ_HZ;
    clk_child_hz = 300000000; /* HZ */
    div = rkclk_calc_clkdiv(clk_parent_hz, clk_child_hz, 1);

    div = div ? (div - 1) : 0;
    cru_writel((1 << 31) | (0x1F << 24) | (1 << 15) | (div << 8),
    CRU_CLKSELS_CON(24));
#endif
```

2.1.4 Reset Cancellation

The final step in the MCU's implement is to perform a reset cancellation. Read the register information in the Chapter 3 CRU Rockchip RK3399 TRM . The reset cancel register of the RK3399 Perilp M0 is:

`PMUCRU_SOFTRST_CON0(0x0110)`

`PMUCRU_SOFTRST_CON0[5:0] = 4b'0000`.

- reference code

```
arch/arm/cpu/armv8/rk33xx/clock-rk3399.c
#ifdef CONFIG_PERILP_MCU
    /* perilp m0 dereset */
    cru_writel(0x00160000, CRU_SOFTRSTS_CON(11));
#endif
```

Tip: The configuration of the clock and reset can be placed where the MCU expected to run. The Rockchip SDK is currently placed in U-Boot.

2.2 Other Configuration

2.2.1 Enable JTAG Configuration

In MCU development, it is often necessary to use JTAG to track, debug and solve problems. The Rockchip MCU JTAG interface implements SWD (2-wire) mode and requires configuration of JTAG iomux to connect.

The iomux configuration details of the RK3399 Perilp M0 can be found in Chapter 7.3 RK3399 TRM, including the following two registers:

```
GRF_GPIO4B_IOMUX[9:8] = 2'b10
GRF_GPIO4B_IOMUX[9:8] = 2'b10
```

2.3 MCU Communicated with SoC

2.3.1 Mailbox

The integrated mailbox on the Rockchip SoC has 4 channels triggered by interrupts. And data is passed through shared memory. Rockchip MCU can communicate with SoC via the mailbox peripheral. RK3399 Mailbox programming can be found in Chapter 21 Mailbox Rockchip RK3399 TRM ; RK3368 Mailbox refer to Chapter 11 Mailbox chapter in Rockchip RK3368 TRM.

Currently in the Linux 4.4 Kernel, the upper layer of the Mailbox Driver framework uses the ARM SCPI protocol, so you need to enable "CONFIG_RK3368_MBOX" and "CONFIG_RK3368_SCPI_PROTOCOL" in Kernel. At the same time, the MCU code also needs the mail driver and SCPI protocol support.

Kernel DTS can be configured with reference to the code below.

```
mailbox: mailbox@ff6b0000 {
    compatible = "rockchip,rk3368-mbox-legacy";
    reg = <0x0 0xff6b0000 0x0 0x1000>,
          <0x0 0xff8cf000 0x0 0x1000>; /* the end 4k of sram */
    interrupts = <GIC_SPI 146 IRQ_TYPE_LEVEL_HIGH>,
                 <GIC_SPI 147 IRQ_TYPE_LEVEL_HIGH>,
                 <GIC_SPI 148 IRQ_TYPE_LEVEL_HIGH>,
                 <GIC_SPI 149 IRQ_TYPE_LEVEL_HIGH>;
```

```
    clocks = <&cru PCLK_MAILBOX>;
    clock-names = "pclk_mailbox";
    #mbox-cells = <1>;
    status = "disabled";
};

mailbox_scpi: mailbox-scpi {
    compatible = "rockchip,rk3368-scpi-legacy";
    mboxes = <&mailbox 0>, <&mailbox 1>, <&mailbox 2>;
    chan-nums = <3>;
    status = "disabled";
};
```

2.3.2 Share Memory

The Rockchip MCU can also communicate with SoC via sharing memory. For example, partitioning a space in INTMEM (SRAM) and configuring it to be accessible to both the master SoC and the MCU, which implements share memory communication.

The Rockchip MCU can also communicate with the master SoC via UART or other methods.

3 Demo Code

3.1 Fetch the Code

Git repository path:

- ssh://git@10.10.10.29/rk/mcu or https://github.com/frawang/rk-mcu.git.
- The code at "10.10.10.29" can refer to the rk3399-pmu-m0 branch
- the github code can refer to the rk3399-box-m0 branch.

3.2 Code Intruduction

3.2.1 Directory

```
rk-mcu>ls -R
.:
build  include  Makefile  src

./build:
arm-gcc-link.ld  RK3399M0

./build/RK3399M0:
bin  obj

./build/RK3399M0/bin:
RK3399M0.bin  RK3399M0.dump  RK3399M0.elf  RK3399M0.map

./build/RK3399M0/obj:
main.o  startup.o

./include:
mcu.h  remotectl_pwm.h  rk3399.h

./src:
main.c  main.c.bk  remotectl_pwm.c  startup.c
```

- build：Use to store the compiled "obj" file and "bin" file.
- include：Header files.
- src：Source files.

  The "startup.c " is the M0 entry program, which mainly includes the M0 interrupt vector table and the interrupt execution function.
  The "main.c" is the main functions of the M0 program.

3.2.2 compiling

The cross-compilation tool chain implements gcc-arm-none-eabi-v4.8 or newer.
The compilation method as follows:

```
rk-mcu>make help
usage: make PLAT=<RK3399M0> <all|clean|distclean>

PLAT is used to specify which platform you wish to build.
If no platform is specified in first time, PLAT defaults to:

Supported Targets:
  all   Build all the project
  clean Clean the current platform project
  distclean Clean the current project and delete .config

example: build the targets for the RK3399M0 project:
  make PLAT=RK3399M0
```

After compiling, "build/RK3399M0/bin/RK3399M0.bin" will be generated, Then copy "RK3399M0.bin" to the directory "tools/rk_tools/bin/rk33/ " in the U-Boot and rename it to "rk3399bl30_v1.00.bin", After that, follow 2.1.1 Boot Address instruction, recompile U-Boot, and packing M0 bin into trust.img .

3.2.3 Interrupt Development

M0 interrupt vector table can be seen at Cortex-M0 Devices Generic User Guide
Chapter 2.3 Exception model.

The reference Demo program in ”src/startup.c“ has the following reference code:

```
/**
  * The minimal vector table for a Cortex M3.  Note that the proper constructs
  * must be placed on this to ensure that it ends up at physical address
  * 0x00000000.
  */
__attribute__ ((used,section(".isr_vector")))
void (* const g_pfnVectors[])(void) =
{
    /* core Exceptions */
    (void *)&pstack[STACK_SIZE], /* the initial stack pointer */
    reset_handler,
    nmi_handler,
    hardware_fault_handler,
    0,0,0,0,0,0,0,
    svc_handler,
    0,0,
    pend_sv_handler,
```

```
    systick_handler,

    /* external exceptions */
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
};
```

There are 16 internal exceptions of Coretex-M0 ,which from "g_pfnVectors[0]" to "g_pfnVectors[15]", you can register and implement the reference exception handling function according to the actual application.

The Cortex-M0 can handle 32 external interrupts, refer to "g_pfnVectors[16]" to "g_pfnVectors[47]".

RK3399 Perilp M0 introduces an interrupt arbiter that extends 32 external interrupts to 256, see Chapter 7.4.6 Interrupt Source Arbiter for PERILPM0 in Rockchip RK3399 TRM. Of course, to use the arbiter, you are required to configure the reference mask bit. After receiving the interrupt, M0 needs to judge the specific interrupt source according to the corresponding mask bit.

The Arbiter's interface can be found in" include/rk3399.h"

M0_INT_ARB_SET_MASK() // Set interrupt mask
M0_INT_ARB_GET_FLAG() // Get interrupt bit

About the external interrupt supported by RK3399 Perilp M0 , please refer to Chapter 2.4 System Interrupt Connection for Cortex-M0 in Rockchip RK3399 TRM.

4 MCU Debug

4.1 JTAG Debug

- Set iomux, tck, tms about JTAG in GRF, please refer to 2.2.1 Enabling JTAG configuration.
- Development board JTAG switch or "tck/tms" switch to the MCU;
- DS-5, ICE or Jlink connect to m3/m0 for debugging.

4.2 Console Printing

- M0 can access the UART register directly for printing and debugging.
- MCU implements the same UART as the SoC master, it is recommend to turn off M0 printing during normal operation to prevent the system being abnormal due to irregular UART access.

4.3 Read and Write Register

- The system can be stay to the U-Boot or Kernel command line and the MCU status register can be read via io to view the MCU status.
- It is also one method to write the running status of the MCU key point to the idle GRF register and then read its value on the U-Boot or kernel command line to judge the current running state of the MCU.

Reference Document

This section lists relevant documents published by third parties:

Cortex-M0 Devices Generic User Guide
Cortex-M0 Technical Reference Manual
ARM Cortex-M3 Processor Technical Reference Manual
Cortex-M3 Devices Generic User Guide
Rockchip RK3399 TRM V0.4
Rockchip RK3368 TRM V2.0