RZ/N1D

Rev.0.91

SIMPLE FAST IPCM Usage Notes

Introduction

This code and guide provides a straightforward IPCM (mailbox) sample with which to communicate in a fast manner between the M3 and A7 cores of the RZ/N1D.

Code for both the A7, linux, and the CM3, the HW RTOS realtime side, is provided.

Observe this is not an “API” but rather demonstrates how to use the core-to-core mechanism in a simple straightforward manner and can be used as a starting point for new projects.

For information on the PL320 IPCM, see ARM PL320 PrimeCell® Inter-Processor Communications Module (PL320) documentation.

Terminology

###### IPCM

In Arm and other documentation the “IPCM” is sometimes referred to as “Inter Processor Communication Mailbox”, “Inter Processor Communication Module”, or just “PL320”. Here we try to just use the term IPCM.

###### CTC

Core to core communication. Using the IPCM to communicate between the RZ/N’s M3 core and A7 core.

###### Channel ID

ID number of the core. The RZ/N has one Cortex M3 core and two ARM7 cores. Channel ID means the core’s “number”. The core numbers are 1 for the CM3, and 2 and 3 for the A7 side.

###### Linux, linux

In this document Linux with capital ‘L’ refers to host PC, whereas linux (small ‘l’) refers to the A7 linux system. At time of this writing (K)Ubuntu LTS (Long Term Support) version 16.04 is the referenced PC platform.

References

1. [RZ/N1 homepage](https://www.renesas.com/en-us/products/microcontrollers-microprocessors/rz/rzn/rzn1d.html)
2. [YCONNECT-IT-RZN DVD](https://www.renesas.com/en-us/software/D6002645.html)

Click on 'Downloads'. You should see a DVD image link. Download all the files, then assemble the ISO image by executing the .exe file, then you should be able to unzip it.

1. PL320 ARM standard, PrimeCell® Inter-Processor Communications Module (PL320).
2. Section 14 Mailbox (IPCM) of HW manual R01UH0751EJ0100 *System Control and Peripheral*. See the Documents folder of the DVD.
3. RZ/N Buildroot environment with scripts for building linux, a file system, U-boot etc.  
   <https://github.com/renesas-rz/rzn1_linux-4.19_bsp>

Target Device

RZ/N1D. This MPU is mounted on the RZN1D-DB development board which also features an expansion board “RZN1D-EB”.

Content

[1. Overview 2](#_Toc41398844)

[2. How to Run Code 3](#_Toc41398845)

[2.1 M3 3](#_Toc41398846)

[2.2 A7 (RZ/N linux) 3](#_Toc41398847)

[3. Interrupts 3](#_Toc41398848)

[4. Mailboxes 3](#_Toc41398849)

[5. Shared Memory Region 4](#_Toc41398850)

[5.1 Transfer M3 => A7 4](#_Toc41398851)

[5.1.1 Data exchange mechanism 4](#_Toc41398852)

[5.2 Transfer A7 => M3 4](#_Toc41398853)

[5.2.1 Data exchange mechanism 4](#_Toc41398854)

[6. M3 Source Code 4](#_Toc41398855)

[6.1 Startup 4](#_Toc41398856)

[6.2 CTC Application 4](#_Toc41398857)

[6.2.1 IAR Terminal I/O 5](#_Toc41398858)

[6.3 Tasks 5](#_Toc41398859)

[6.3.1 init\_task 5](#_Toc41398860)

[6.3.2 pl320\_tx\_task 5](#_Toc41398861)

[6.3.1 pl320\_rx\_task 5](#_Toc41398862)

[6.3.2 idle\_task 5](#_Toc41398863)

[6.3.1 monitor\_task 5](#_Toc41398864)

[7. A7 Source Code 6](#_Toc41398865)

[7.1 The CTC Application 6](#_Toc41398866)

# Overview

The RZ/N IPCM peripheral provides three mailboxes with control logic and interrupts to support inter-processor communication between the two Cortex-A7 cores and the Cortex-M3: Three mailboxes, each comprising seven 32-bit data registers to store the message.

Observe that the RZ/N has three cores, but also three mailboxes. These are shared between the cores. That is, each mailbox contains seven data registers to hold the message. *As the RZ/N has both three cores and three mailboxes, do not confuse them with each other.*

* **M3** or CM3 is the real-time 125 MHz clocked real time core of the RZ/N. This is the Renesas R-IN32; where the real time industrial ethernet and related protocols reside. Here is where the HW-RTOS and Ethernet accelerator, checksum enhancement features are.
* **A7** refers to the 500 MHz clocked core of the RZ/N running linux with a large amount of DDR memory.

# How to Run Code

The example code to show usage of (test) the IPCM consists of two parts. One M3 code set, and one A7 code set. The code sends data both ways, but since the M3 is the slower core, the code is written to be able to flexibly test various transfer speeds from A7 to M3 without recompiling. The M3 just sends data every TX\_SLEEP\_TIME\_MS to the A7 as set in *sample\_app.c* (default 1000 ms).

## M3

The M3 code is in an IAR-EW workspace. Please use IAR 8.20 or above to download/debug. The workspace is

.\IPCM\M3\IPCM\rzn1d\_ipcm.eww

To run in debug mode, compile, download and debug after the RZ/N has been started with appropriate U-Boot settings. See the RZ/N QSG of the DVD for this.

## A7 (RZ/N linux)

How to build and download the code to the linux file system is not covered in this manual. See separate documentation under References on how to build and deploy. The sample code comes with a very simple Makefile.

With the M3 side running, and after booting and logging in to linux, navigate to the folder where the A7 binary has been downloaded and enter the name of the binary and add three arguments. The syntax is

#./pl320 sleep\_time\_us shm\_size\_bytes show\_M3\_to\_A7\_content

The text is to be replaced with numbers as follows:

* **sleep time microsec** is the number of microseconds between transmission to M3 side. This is not
* **shm\_size\_bytes** should be at least 8 since two first four bytes (word) is transferred for verification of core number. See 7.1.
* **show M3 to A7 content** is a flag, 0 or 1, controlling whether to present the memory content received from M3.

# Interrupts

RZ/N IPCM has three interrupt vectors. One for each core.

* **#59 IPCM\_Int[0]** Mailbox interrupt for CM3 core.
* **#60 IPCM\_Int[1]** Mailbox interrupt for A7 core. This is the core used in SMP mode which is the standard linux compile (release on DVD). This core corresponds to **/dev/mbox\_2** when linux is running on RZ/N and the mailbox has been opened from the A7 linux application.
* **#61 IPCM\_Int[2]** Mailbox interrupt for A7 core.

Interrupt #59 is the M3 IPCM, and #60 (and #61) are for the A7. Observe that the interrupt numbers are not 0, 1, 2, but rather bits in in a bit mask; bits 0, 1, and 2. Therefore, the code has IPCMxSOURCE SRC\_SET bits are assigned as “1” and ‘2”. “1 is the M3 core. “2” is the A7 which is the only interrupt used for the A7 as the A7 is running in SMP mode (symmetric multiprocessing mode) which is the standard linux compile of the release on DVD. Again, this interrupt corresponds to device **/dev/mbox\_2** when linux is running on RZ/N.

# Mailboxes

The RZ/N IPCM has three mailboxes. These are shared between the cores.

Mailbox 0 is used for M3 to A7 data transfer, and mailbox 1 for A7->M3 transfer.

Each mailbox contains seven data registers to hold a message. Not all of them are used in the test code. Shared memory is used to exchange actual ‘payload’ data.

# Shared Memory Region

Here are the memory regions used for transferring data using the IPCM mailboxes for the example code. Shared memory size is *0x8000 starting at 0x200F8000.*

* M3 to A7 data is written starting at **0x200FC000**.
* A7 to M3 data is written starting at **0x200F8000**.

Data is not exchanged using the mailboxes directly. The mailboxes are merely used to signal that data has been written to shared memory. The mailbox data register usage is described below.

## Transfer M3 => A7

The linux side code requires mailbox data register 0 to be destination core ID. This is the second core (bitmask=2) of the system as seen from the M3 (A7 is destination). This is how the pl320 source comes by default in linux. No other data is sent using the mailbox data registers.

Data registers 1 and 2 are assigned an enumeration cycle pattern to be able to verify consistency on the A7 side after transfer. These contain the first bytes of the shared memory data M3->A7. Can be used e.g. for debug trace.

### Data exchange mechanism

The ‘payload’ test data is exchanged via shared memory.

## Transfer A7 => M3

Again, linux requires mailbox data register 0 to be destination core ID. This is the first core (bitmask=1) of the system as seen from the A7 (M3 is destination).

### Data exchange mechanism

Data register 0 contains the destination core ID. The Linux kernel drive code is by default written so data register 0 is destination core ID. This is the “first core” (0) of the system as seen from the M3.

Data registers 1 and 2 contain the first bytes of shared data A7->M3 to be able to verify consistency on the M3 side after transfer, and just as for the M3 side can be used e.g. for debug trace.

Data register 2 contains second byte of shared data.

Data register 3 contains the number of bytes sent this transfer.

Data registers 4, 5, and 6 are not used.

The ‘payload’ test data is exchanged via shared memory.

# M3 Source Code

## Startup

The example code is based upon the startup code provided in the Bare Metals Driver example code from the RZ/N DVD.

All tasks in *sample\_app.c* are run by HW-RTOS, according to settings in *kernel\_cfg.c*.

## CTC Application

The source code of interest for IPCM/CTC is mainly in files *sample\_app.c* and *kernel\_cfg.c*.

The sample app common data’s purpose is to show the performance of the co-to-core operation.

To follow the CTC progression of transfers, add these global variables of *sample\_app.c* to IAR’s watch window.

* **M3\_to\_A7\_mem\_write\_try\_cnt** Number of attempted writes to shared memory.
* **M3\_to\_A7\_mem\_write\_cnt** Number of actual writes to shared memory
* **M3\_to\_A7\_mem\_write\_delta\_cnt**  Number of failed attempts writing to shared memory.
* **M3\_to\_A7\_mem\_write\_sleep\_cnt** Number of program loops waiting for mailbox send register  
   to become active.
* **Error** Number of write errors to shared memory.

### IAR Terminal I/O

The demo shows data transfer updates every DISPLAY\_UPDATE\_MAXCOUNT in the IAR Terminal I/O window. Usage of Terminal I/O window may significantly compromise CPU performance at the high rates of CTC. So best is to stay with using only Live Watch monitoring.

## Tasks

The code runs in HW-RTOS tasks.

### init\_task

This task configures the HWOS system tick, then just sleeps and wakes up once per second.   
Can be used for additional features.

### pl320\_tx\_task

This task sleeps TX\_SLEEP\_TIME\_MS then writes to shared memory a number of bytes given by M3\_TO\_A7\_MEM\_WRITE\_SIZE\_BYTES. It then waits for the IPCM mailbox send register to become active again. The function *write\_to\_shm* writes the data to the shared memory area.

The Linux kernel requires to fill data reg. 0 with destination core ID. See 5.1.1.

The payload data is simply an enumeration of ever-increasing numbers.

M3 starts to run much earlier than Linux. So it does not start reading shared memory until data from Linux becomes operational. This is determined by looking at data register 0, or IPCM->IPCM1DR3. (Destination core nr, see above.)

### pl320\_rx\_task

This task sets up the mailboxes; which core is source and which destination, then goes into a forever loop that awakens via HW-RTOS when the RZN1\_IRQ\_IPCM\_0 interrupt triggers. This interrupt triggers when the first of the three cores, the M3, receives data.   
See also *static\_hwisr\_table[]* in *kernel\_cfg.c*.

*pl320\_rx\_task()* calls *check\_memory()* which checks that the data is an expected enumeration sequence. This could be sped up using a checksum.

### idle\_task

This task does nothing except spin in a forever spin at the lowest priority level - as is the ideal case for the Idle task.

### monitor\_task

This is an added mechanism to monitor M3 CPU load. It is possible by having added the ISR *systick\_handler()* to *static\_interrupt\_table[]*. Both the monitor task and *systick\_handler()* can of course be commented out to further improve performance.

# A7 Source Code

The code operates as an EXT4 executable under linux. The example executable communicates with the M3 to show usage of the IPCM.

The source files of main interest are in the *../Linux/pl320* folder; *shm.h/c*. The kernel source is in *../rzn1\_linux/drivers/mailbox*, files *shm.h/c, mailbox.h/c, pl320.h/c, pl320-ipc.h/c*.   
See also ..*/rzn1\_linux-4.19\_bsp/output/rzn1\_linux/include/linux/interrupt.h*.

## The CTC Application

The main application is in file *pl320.c*.

The main function parses the user command line arguments as explained in 2.2, namely **sleep\_time\_us**, **shm\_size\_bytes**, and **show\_M3\_to\_A7\_content\_flag**. It then opens the mailbox device *MBOX\_DEVICE\_MASTER\_NAME*, as defined in *./rzn1\_linux-4.19\_bsp/output/rzn1\_linux/drivers/char/mailbox*.

The mailbox device is the second core (2) two (of three) of the RZ/N. There are two A7 cores running in SMP mode. This results in the ability to open **/dev/mbox\_2** in RW mode.

The task then sleeps for the user specified time and then writes to the shared memory area the specified number of bytes.

Again, the in linux the first four bytes of IPCM (the first data register) is used for Core ID#. The main application then fills IPCM data registers #1 and #2 with a never ending enumeration sequence so as to to easily verify consistency on the M3 side.

Website and Support

Renesas Electronics Website

<http://www.renesas.com/>  
[RZ/N1-1 homepage](https://www.renesas.com/en-us/products/microcontrollers-microprocessors/rz/rzn/rzn1d.html)  
[RZN1 DVD](https://www.renesas.com/en-us/software/D6002275.html)

Inquiries

<http://www.renesas.com/contact/>

All trademarks and registered trademarks are the property of their respective owners.

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Date | Description | |
| Page | Summary |
| 0.90 | May 19, 2020 |  | First version. |
| 0.91 | May 26, 2020 |  | Added comment about monitor task , minor changes p 1, updated TOC. |

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

|  |
| --- |
| 1. Handling of Unused Pins  Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.   * The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.   2. Processing at Power-on  The state of the product is undefined at the moment when power is supplied.   * The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.   3. Prohibition of Access to Reserved Addresses  Access to reserved addresses is prohibited.   * The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.   4. Clock Signals  After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.   * When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.   5. Differences between Products  Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.   * The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product. |

