Application Report
SPRACH5A–July 2018–Revised January 2019

PRU-ICSS / PRU_ICSSG Getting Started Guide on TI-RTOS

Garrett Ding

ABSTRACT

This application report provides a getting started guide for the PRU-ICSS / PRU_ICSSG of the Sitara™ family of devices, while running TI-RTOS on the ARM core. The application report is a companion document of *PRU-ICSS / PRU_ICSSG Getting Started Guide on Linux*, intended to help new users ramp up PRU-ICSS / PRU_ICSSG-based development quickly. The application report focuses on TI-RTOS-specific topics, and does not cover the basics discussed in the *PRU-ICSS / PRU_ICSSG Getting Started Guide on Linux*, which includes:

- PRU-ICSS / PRU_ICSSG description
- Devices containing the PRU-ICSS / PRU_ICSSG
- What can the PRU-ICSS / PRU_ICSSG do
- PRU-ICSS / PRU_ICSSG usage environments
- PRU C compiler and assembler

Contents

List of Tables

Trademarks

Sitara is a trademark of Texas Instruments.
Linux is a registered trademark of Linus Torvalds.
All other trademarks are the property of their respective owners.

1 PRU-ICSS / PRU_ICSSG TI-RTOS Drivers

1.1 PRUSS Driver

The Programmable Real-time Unit SubSystem (PRUSS) on Texas Instruments' Sitara processors is firmware-programmable, and can execute various functionalities as designed: for example, Ethernet MAC, Ethernet switch, and real-time industrial protocols. The PRUSS driver from the Processor SDK RTOS provides well-defined APIs, which allow an application to control the subsystem and are applicable to both PRU-ICSS and PRU_ICSSG.

The PRUSS driver module available from <PDK>\packages\ti\drv\pruss includes:

- Compiled library
- Sources, unit test code
- API reference guide

The PRUSS driver is featured with the following:

- Control to enable, disable, or reset a PRU
- Helper functions to load and execute firmware in PRU
- Memory mapping of PRU, L3, or external memories
- PRU and host event management to map sys_evt, channel, or hosts in the PRU Interrupt Controller (INTC) to generate interrupts, wait for occurrence of an event, and acknowledge interrupts
- Interrupt management for A15/C66x CPU targets

A typical PRUSS API call flow, for a use case when sending an INTC event from the host to the PRU, is described in Processor SDK RTOS driver section.

All the board-specific configurations, for example, enabling the clock and pin-mux of UART/GPIO/PRUSS pins should be performed before calling any of the driver APIs.

The PRUSS configuration structure (pruss_config) must be provided to the driver when the PRUICSS_create() API is called to create the PRUICSS_Handle. This handle is subsequently required to make any PRUSS LLD API call. A PRUSS example Code Composer Studio (CCS) project with the above call flow is available in the Processor SDK. Refer to the PDK Example and Test Project Creation procedure to get familiar with the example project "PRUSS_BasicExample" creation and PRUSS APIs.

PRUICSS_setPRUBuffer() sets the firmware buffer pointer for the PRU. PRUICSS_pruExecProgram() implements firmware array loading to Instruction RAM (IRAM) using PRUICSS_pruWriteMemory(), PRU reset and enabling to execute the program.

PRUICSS_pruWriteMemory() is the function to write the given data (firmware) to PRU memory, including IRAM and data RAM (DRAM). For example:

```
PRUICSS_pruWriteMemory(ICSS_EMAC_testPruIcssHandle,PRU_ICSS_DATARAM(0)...)
PRUICSS_pruWriteMemory(ICSS_EMAC_testPruIcssHandle,PRU_ICSS_IRAM(0)...)
```

The PRU Code Generation Tool (CGT) 2.2.1 has added an --array option that uses the hexpru utility to take an *.out file and create a 32-bit array of PRU firmware.

1.2 ICSS_EMAC Driver

The Industrial Communications Sub System Ethernet Media Access Controller (ICSS_EMAC) driver provides APIs to transmit and receive packets with a PRUSS firmware-based Ethernet switch implemented on the PRU-ICSS 32-bit RISC cores.

The ICSS EMAC low level driver can be partitioned into:

- Driver software running on the host processor that provides a well-defined set of APIs to configure the driver, send packets to the firmware, and receive packets from the firmware.
- Firmware running on PRU-ICSS cores that implements a 2-port Ethernet switch supporting 802.1d at 100 Mbps.

The ICSS_EMAC driver is featured with the following:

- Rx – Copying packet received from firmware and providing it to the TCP/IP stack
- Tx – Providing packet from TCP/IP stack to firmware
- Learning/forwarding data base
- Storm prevention implementation
- Host statistics implementation
- TCP/IP stack-related initialization
- Configuring IP address
- ARM interrupt management

A typical use case of the ICSS_EMAC driver APIs for an EMAC example which implements a single Ethernet MAC using PRU-ICSS Instance 2, ETH0 is described in the Processor SDK RTOS driver section.

All the board-specific configurations, for example, enabling the clock and pin-mux of GPIO, MDIO, and IEP pins should be performed before calling any of the driver APIs.

Hardware attributes, including the base address of various sub-subsystems required by the ICSS_EMAC driver, are required and must be passed in as part of the ICSS_EMAC handle when calling the icss_emacInit( ) API, which initializes and configures PRU-ICSS in EMAC/switch mode. For details on EMAC/switch mode, refer to ICSS_EMAC LLD developers guide.

1.3 ICSSG EMAC Driver

Unlike the ICSS EMAC driver, the ICSSG EMAC driver is unified into the EMAC driver, which provides a well-defined API layer to enable the EMAC peripheral to control the flow of packet data from the processor to the PHY and the MDIO module, to control PHY configuration and status monitoring. The EMAC driver is a common driver for supporting all 1 Gigabit Network interfaces, including CPSW and ICSSG, for applicable SOCs.

All the board-specific configurations, such as the enabling and pin-mux of the RGMII/MDIO pins, should be performed before calling any driver APIs. The emac_soc.c file binds the driver with hardware attributes on the board through the EMAC_Cfg structure. This structure must be provided to the EMAC driver and initialized before the emac_open() function is called.

A calling sequence of EMAC driver APIs for an EMAC example implementing a single Ethernet MAC port is described in the Processor SDK RTOS driver section.

2 PRU-ICSS / PRU_ICSSG TI-RTOS Examples

2.1 PRU Software Support Package

As explained in the *PRU-ICSS / PRU_ICSSG Getting Started Guide on Linux*, the PRU Software Support Package hosted in the git repository contains building block PRU examples for Sitara devices, along with the necessary header files and libraries. The PRU Software Support Package is designed in the context of RemoteProc Linux driver, which is responsible for taking the PRU firmware from the Linux® filesystem, parsing it for any resources that it needs to provide for the PRU (interrupts or shared memory), loading it into PRU instruction memory and data memory, and then running the PRU. A resource table is necessary for the RemoteProc driver to work, even if it is empty.

Except for the RPMsg examples in the PRU Software Support Package, which must use the resource table to request that Linux creates virtio vrings in DDR and configure the PRU system events to be used as notifications, a lot of the examples that have empty resource tables which can be removed or excluded, and can be re-built and run with an ARM TI-RTOS use case.

Similarly, as PRU development on ARM Linux, TI recommends referring to the PRU Software Support Package, and using one of the examples in the PRU Software Support Package as a starting point for developing a new general purpose firmware on TI-RTOS.

2.2 PRU_ICSS Dual EMAC and Switch

The PRU-ICSS firmware of a dual EMAC and switch is a part of the ICSS_EMAC lower level driver. The dual EMAC firmware realizes a single port Ethernet MAC, which can be used independently on two PRUs to implement two independent MACs with two different MAC addresses and two different IP addresses. The switch firmware is a 3-port learning Ethernet switch, and implements a 2-port cut through Ethernet switch. The dual EMAC and switch PRU-ICSS firmware source code is available in TI's Processor SDK, as shown in Table 1.

Table 1. ICSS_EMAC Modules and Locations

Module	Location
Common baseline code	<PDK>/packages/ti/drv/icss_emac/firmware/icss_dualemac
Dual_EMAC	<PDK>/packages/ti/drv/icss_emac/firmware/icss_dualemac
Switch	<PDK>/packages/ti/drv/icss_emac/firmware/icss_switch

For details of building and running the dual EMAC and switch firmware and example, refer to Processor SDK PRU-ICSS firmware section

The Processor SDK provides a variety of ARM/DSP/M4 core-based application examples of dual EMAC and switch, such as:

- ICSS_EMAC_BasicExample
- ICSS_EMAC_SwitchExample
- NIMI_ICSS_BasicExample
- NIMU_ICSS_FtpExample
- NIMU_ICSS_CCLinkMaster
- NIMU_ICSS_CCLinkSlave

2.3 PRU-ICSSG EMAC

The Processor SDK provides A53 and R5F core-based application examples of PRU_ICSSG EMAC, such as:

- Emac_Icssg_TestApp
- NIMU_FtpIcssg_ExampleApp
- NIMU_Icssg_ExampleApp

The linker and make files of these example projects are available from:

- <PDK>/packages/ti/drv/emac/test/EmacLoopbackTest/am65xx
- <PDK>/packages/ti/transport/ndk/nimu/example/am65xx

An advanced application, such as the ICSSG example that uses resource management for UDMAs, interrupt setup, power management to setup clock modules, or wakeup/power of slave cores, requires loading the SYSFW (DMSC firmware) on the M3 core so that the application can make API calls to leverage its services.

To load the SYSFW firmware, the DMSC ROM expects the R5F secondary bootloader/application to provide a board configuration message to initialize the cores and SOC services. The R5F application provided in SciClient uses a default board configuration message to the SYSFW, and sets up the device for application debugging.

For details, refer to Advanced AM65x Debug Setup with DMSC Firmware Load.

2.4 Simple Open Real-Time Ethernet Protocol (SORTE)

The Simple Open Real-time Ethernet protocol (SORTE) is a TI-developed industrial Ethernet protocol that supports 4-µs cycle time and operates exclusively on the PRU-ICSS. The protocol is fully documented and released in source code. It is open to customers to learn, adapt, and enhance the protocol for their application requirements.

THE SORTE ARM application and firmware sources can be found under the following directory:

<PDK>/packages/ti/drv/pruss/example/apps/sorte/

Refer to the README.txt for high-level overview of how the protocol is implemented for master and slave device network components:

<PDK>/packages/ti/drv/pruss/example/apps/sorte/firmware/src/master/README.txt

<PDK>/packages/ti/drv/pruss/example/apps/sorte/firmware/src/slave/README.txt

For details of the SORTE building and running procedure, refer to Processor SDK PRU-ICSS firmware section.

2.5 I2C

PRU-ICSS I2C provides additional I2C instances for Sitara processors. The firmware supports standard two-pin I2C interface through three GPIO pins from PRU-ICSS peripheral. The I2C SCL pin for firmware is implemented with a single GPIO configured in GPI mode. While the I2C SDA pin is implemented with two GPIO pins, one pin configured in GPI mode for taking input sample and a second pin configured in GPO mode for driving the line. Depending on Baud rate, firmware can emulate multiple instances of the I2C interface from a single PRU core. The I2C firmware source code is available from <PDK>/packages/ti/drv/i2c/firmware/icss_i2c/src. For details of the I2C firmware build procedure and its example application, refer to Processor SDK PRU-ICSS firmware section.

2.6 Industrial SW

The PRU-ICSS / PRU_ICSSG industrial software for Sitara processors contains many optimized real-time industrial communication protocols firmware including EtherCAT, Profinet, EtherNet/IP, Profibus, HSR/PRP, and industrial drive interfaces. The PRU-ICSS / PRU_ICSSG firmware runs on the PRU cores, offloading the time-critical link layer processing from the main ARM processor running TI-RTOS. The application examples in the software illustrate how to integrate low-level firmware with the protocol stack and application software.

The industrial SW features:

- PRU-ICSS / PRU_ICSSG firmware binary images and driver sources
- Third-party stacks and evaluation libraries
- Scripts to generate CCS projects
- Example application for evaluation
- Documentation (release notes, protocol data sheets, user guides, porting guides, and so forth)

3 More Information

3.1 Training Materials

The PRU training series covers:

- Sitara Processors Building Blocks for PRU Development
- Tools for PRU development
- Running Industrial Protocols on PRU

3.2 TI Designs

SORTE Master with PRU-ICSS Reference Design – This design implements a Simple Open Real-Time Ethernet (SORTE) master with PRU-ICSS. SORTE protocol enables customer applications to exchange process data between the master and devices in a 4-µs cycle time. The PRU firmware in source code enables customers to differentiate their products.

SORTE Device With PRU-ICSS Reference Design – This design shows the different industrial Ethernet operation modes, including auto-forwarding (AF), host receive (HR), cut-through (CT), and time-triggered-send (TTS). The application example on top of the SORTE protocol controls the digital 8-bit output LEDs on the TMDSICE3359 evaluation module (EVM).

DDR-less EtherCAT® Slave on AMIC110 Reference Design – This design shows how to run a full EtherCAT slave stack entirely on the internal memory of the Sitara AMIC110 SoC, while offloading the EtherCAT data link layer to PRU-ICSS.

EnDat 2.2 System Reference Design – This design implements the EnDat 2.2 Master protocol stack on PRU-ICSS, and a hardware interface solution based on the HEIDENHAIN EnDat 2.2 standard for position or rotary encoders.

3.3 E2E Forum Support

Any questions not addressed by this guide can be posted on the Sitara E2E Forums.

4 Frequently Asked Questions

This section includes a few of the most common questions regarding using the PRU-ICSS with TI-RTOS on the ARM core. For a comprehensive list of Frequently Asked Questions on the PRU-ICSS, see the PRU-ICSS FAQ wiki.

4.1 How can I load PRU firmware from a host core with TI-RTOS?

Refer to Section 1.1.

4.2 I'm setting a GPIO pin in PRU-ICSS to direct input mode; why is the value reflecting the GPIO pin status read from R31 register unchanged?

In addition to the setting to the pin mux mode to GPIO, ensure that the INPUTENABLE bit is set to get 'receive mode' enabled in the pin's CTRL_CORE_PAD register. Because the PRU does not have privileges to edit the pinmux or pad config settings in the device-level control module, the INPUTENABLE bit in CTRL_CORE_PAD register can only be set from a host core.

4.3 How can I establish the communication between a host core and PRU?

Shared memory (DDR, OCMC RAM, or PRU data RAM) with an interrupt between a host core and PRU can be used to implement the functionality.

For an ARM core, a memory fence instruction must be used to insure that all writes have completed before generating an interrupt and any access from the PRU. While using the HW_WR_REG32_RAW() function, which has HW_MEM_BARRIER() embedded, ensure that __ARMv7 or __TI_ARM_V7__ is defined in your project to get the barrier taking effect; see the implementation of HW_MEM_BARRIER below.

```
static inline void HW_MEM_BARRIER(void)
{
#if (defined(__ARMv7) || defined(__TI_ARM_V7__))
#ifndef MEM_BARRIER_DISABLE
    asm("    dsb");
#endif
#endif
}
```

While using the DDR as shared memory, ensure the MMU has the memory attribute set to shareable, as in the following example:

```
// descriptor attribute structure
var attrs1 = new Mmu.DescriptorAttrs();

Mmu.initDescAttrsMeta(attrs1);
attrs1.type = Mmu.DescriptorType_BLOCK;    // BLOCK descriptor
attrs1.shareable = 2;                      // shareable
attrs1.attrIndx = 2;                       // Cached, normal memory

// Set the descriptor for each entry in the address range
for (var i=0x80000000; i < 0xA0000000; i = i + 0x00200000) {
    Mmu.setSecondLevelDescMeta(i, i, attrs1);
}
```

The PRU can interrupt the ARM by writing to R31 and generating a system event. The PRU INTC should be pre-configured to map this system event to a host interrupt connected to the ARM. The ARM can interrupt a PRU by writing to the PRU INTC SRSRx register and setting a pr<k>_pru_mst_intr<x>_intr_req system event. The PRU INTC should be pre-configured to map this system event to a host interrupt connected to the PRU. The PRU can poll R31 bit 30 or 31 to detect an interrupt on host 0 or 1, respectively.

The example PRU_ARMtoPRU_Interrupt and PRU_PRUtoARM_Interrupt projects in the PRU Software Support Package can be used as reference for interrupt configuration and handling.

5 References

- RU-ICSS Getting Started Guide on Linux
- Processor SDK Device Drivers
- SORTE Master with PRU-ICSS Reference Design
- SORTE Device With PRU-ICSS Reference Design
- DDR-less EtherCAT® Slave on AMIC110 Reference Design
- EnDat 2.2 System Reference Design
- PRU-ICSS FAQ

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (July 2018) to A Revision	Page
• Updated Title.	1
• Added ICSSG EMAC Driver section.	3
• Added PRU-ICSSG EMAC section.	4

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265