# AM335x PRU Linux Application Loader

## User Guide

# Contents

## Articles



## References

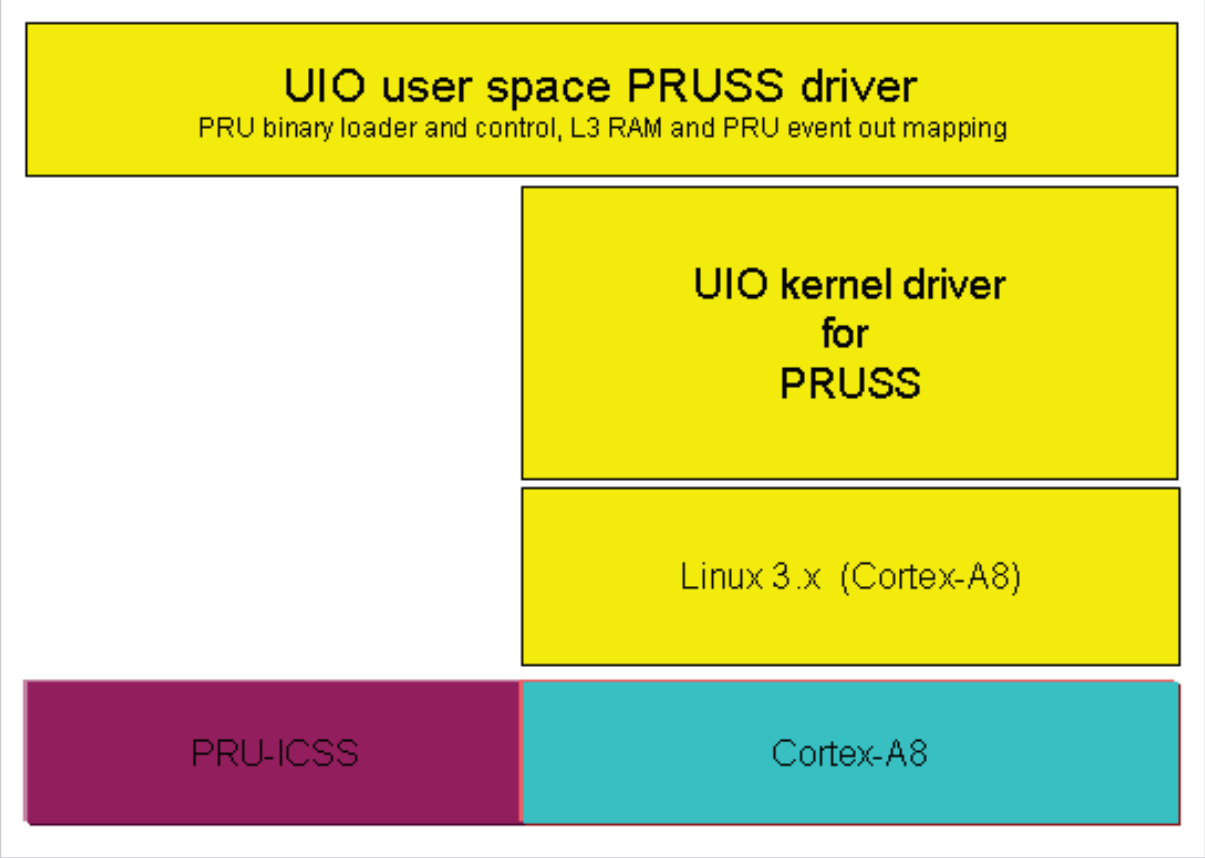
# PRU Linux Application Loader

The software modules and descriptions referred to in this document are ***NOT SUPPORTED*** by Texas Instruments (www.ti.com / e2e.ti.com).

These materials are intended for do-it-yourself (DIY) users who want to use the PRU at their own risk without TI support. "Community" support is offered at BeagleBoard.org/discuss.

## General Overview

The AM335x PRU application loader for Linux is a software tool which can be used to load a binary to PRU's memory area and to manage the code executed in the PRU from the user space. The software stack consists of two main sections: the low level kernel driver and the user space driver. The low level kernel driver (uio_pru) provides the basic foundation for the PRUSSDRV user space driver by powering the PRU, initializing the PRU clocks, allocating the PRU memory space, and registering the PRU IRQ lines. The PRUSSDRV library contains options to start and stop PRU, map PRU, L3, and external memories, and manage PRU-generated interrupts. The software architecture of the Linux-based loader is illustrated in Figure 1.



# Installing PRU application Loader

## Before Getting Started

1. Install the AM335x Linux SDK (http://www.ti.com/tool/linuxezsdk-sitara).

**NOTE**
The PSP release 04.06.00.03 version is required for the PRU application loader.

The patches enabling the uio_pru kernel driver can be found at http://arago-project.org/git/projects/linux-am33x.git:

af5db73cb0ef2402233b447e5d8c043e395d9c5a

ARM:OMAP:AM33XX:pruss: Add platform specific changes for AM33XX in UIO PRUSS driver

e1bb6be3935bc7eb195d9ed8616770ed01e95d0f

ARM:DA850:pruss: Add platform specific changes for DA850 in UIO PRUSS driver

af1a009d00600d688ba8857bf4cb6c921360addd

ARM:OMAP:AM33XX:pruss: Disable SRAM support in UIO_PRUSS

eb6bd7f12cb810c53fe549051217fcc8004abc3b

ARM:OMAP:AM33XX:icss: ICSS reset handling

a03a914dd9087475acd062c6fb2c07555cd78e01

ARM:OMAP:AM33XX:pruss: Renamed icss_fck to pruss

539d08660a71701d362c3eff23a5538d1c3b7a22

ARM:OMAP:AM33XX: Add AM33XX_IRQ_GPMC0 as gpmc_irq for AM33XX

2. Update the top level SDK Makefile (ti-sdk-am335x-evm-xx.xx.xx.xx/Makefile) to make menuconfig.

**host$** cd ti-sdk-am335x-evm-xx.xx.xx.xx

**host$** vi Makefile

Add make menuconfig (last line shown below)

```
@echo ================================
@echo Building the Linux Kernel
@echo ================================
$(MAKE) -C $(LINUXKERNEL_INSTALL_DIR) ARCH=arm CROSS_COMPILE=$(CROSS_COMPILE) tisdk_$(PLATFORM)_defconfig
$(MAKE) -C $(LINUXKERNEL_INSTALL_DIR) ARCH=arm CROSS_COMPILE=$(CROSS_COMPILE) menuconfig
```

## Preparing and Building the Kernel

1. Compile the Linux kernel and modules.

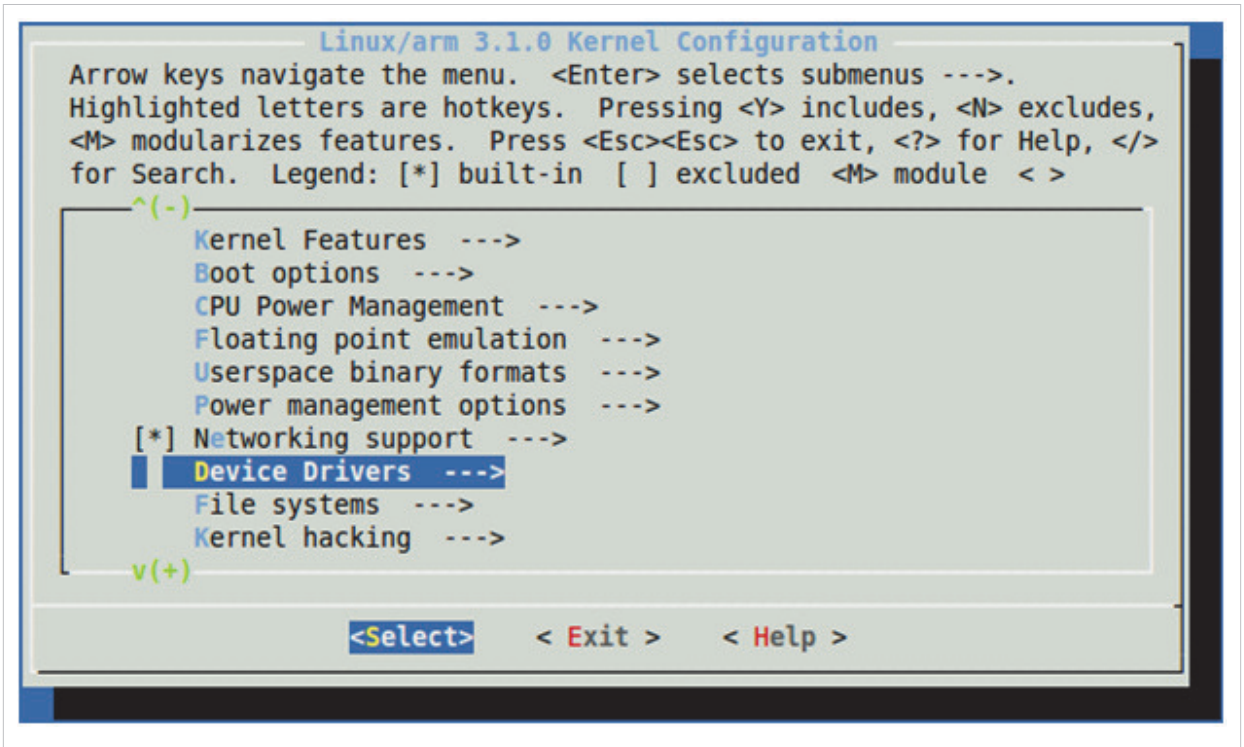
**host$** cd ti-sdk-am335x-evm-xx.xx.xx.xx

**host$** make

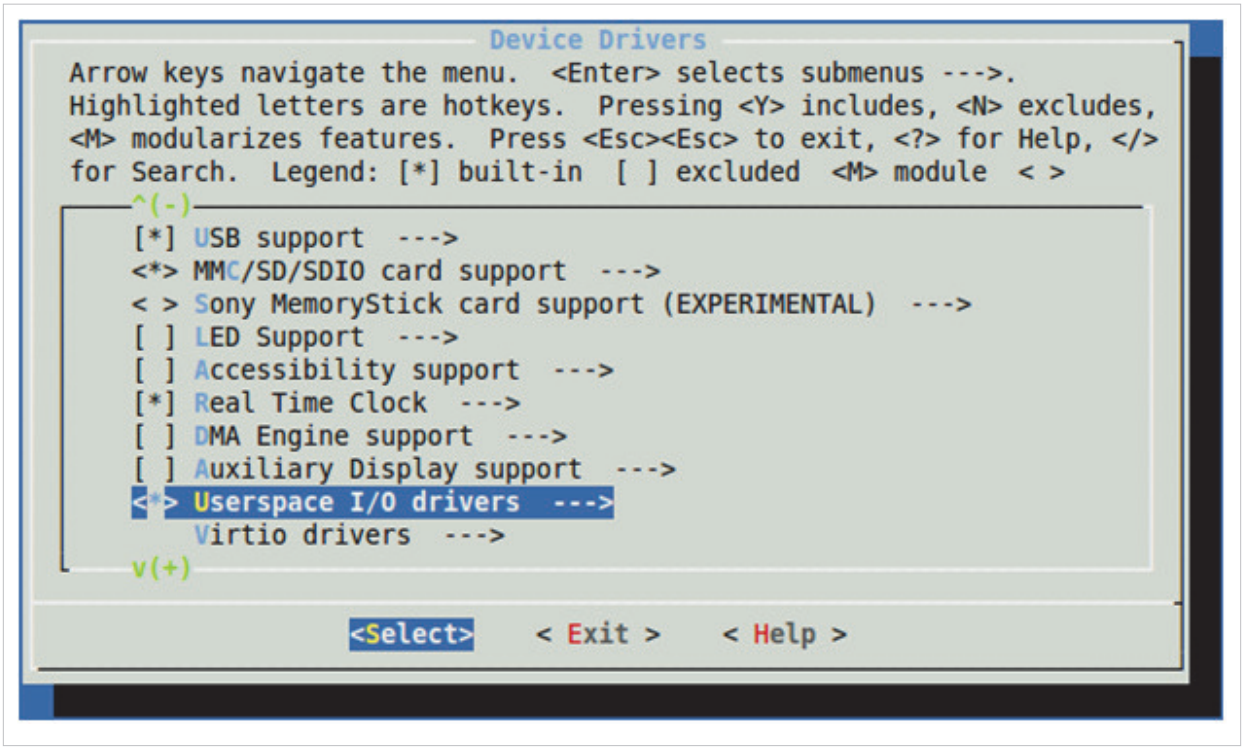
2. Update menuconfig to build UIO_PRUSS into the kernel or as a module.

Within the configuration menu that will automatically open, ensure that UIO support is enabled and that Texas Instruments PRUSS driver is also selected by the following:

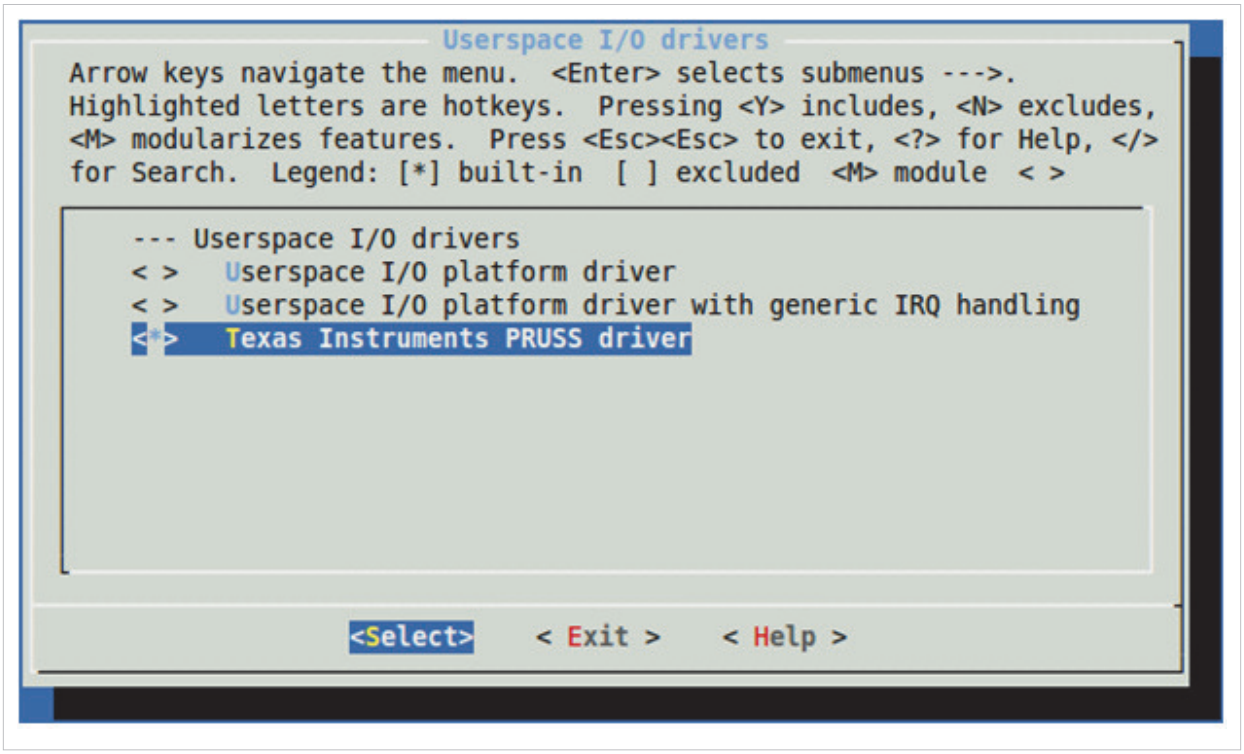
1. From the Linux Kernel Configuration menu, select Device Drivers.




2. In the Device Driver menu, scroll down to Userspace I/O drivers. Press ‘Y’ to enable and then select.




3. In the Userspace I/O drivers, scroll down to Texas Instruments PRUSS driver. Press ‘Y’ to include in kernel, or press 'M' to build as module.




4. Exit and save configuration.

3. Install the modules.

**host$** make linux_install

4. Install the Linux kernel by copying the compiled kernel (located at ti-sdk-am335x-evm-xx.xx.xx.xx/board-support/linux-x.x.x-pspxx.xx.xx.xx.sdk/arch/arm/boot/uImage) to the location where it is going to be read from (i.e. SD card or /tftpboot directory).

**NOTE**

For more details about compiling and installing the Linux kernel and modules, refer to the Sitara Linux SDK Software Developer's Guide (http:/ / processors. wiki. ti. com/ index. php/ Sitara_Linux_Software_Developer%E2%80%99s_Guide).

## Installing the PRUSSDRV User Space Library

The user space library is provided by am335x_pru_package/pru_sw/app_loader/interface/prussdrv.c.

1. Set environment variables

**$host** . /home/.../ti-sdk-am335x-evm-xx.xx.xx.xx/linux-devkit/environment-setup

2. Compile prussdrv.c

**$host** cd pru_sw/app_loader/interface

**$host** make clean

**$host** make

2. Compile application code using PRUSSDRV APIs.

3. Download executable file and PRU binaries to your file system.

## Running applications

1. Boot the device.
2. If kernel driver is configured as a module (default), initialize driver with following command:

   **EVM #** modprobe uio_pruss

   *Note the following error message indicates that the uImage used to boot the board does not have uio_pruss enabled. Verify that the correct uImage was used when booting.*

   [ 487.655249] uio_pruss: Unknown symbol __uio_register_device (err 0)

   [ 487.661878] uio_pruss: Unknown symbol uio_unregister_device (err 0)

   FATAL: Error inserting uio_pruss (/lib/modules/3.1.0/kernel/drivers/uio/uio_pruss.ko): Unknown symbol in module, or unknown parameter (see dmesg)

   If kernel driver is built-in, the driver should be already loaded.
3. Verify driver is installed by checking if the following is visible in the file system:

   **EVM #** cat /sys/class/uio/uio0/maps/map0/addr
4. Execute application

   **EVM #** cd <path to application>

   **EVM #** ./<application name>

# PRU Linux Loader Functions

## Introduction

The PRUSSDRV User Space Library contains APIs that support the following:

- Basic PRU control (i.e. enable/disable/reset PRU)
- Helper functions (i.e. load and execute code in PRU)
- Memory mapping of PRU/L3/External memories
- PRU and Host event management (i.e. map sys_evt/channel/hosts in PRU INTC, generate interrupts, wait for occurrence of an event, and acknowledge interrupts)

The source code for the PRUSSDRV library is included in the pru_sw/app_loader/interface/prussdrv.c directory.

## API Descriptions

The functions provided by the PRUSSDRV library are described below.

### prussdrv_init

Initializes and allocates memory for the PRU Subsystem driver. This is a required function call.

**Function declaration:** int prussdrv_init (void)

**Return value:** The prussdrv_init() function shall return 0; no return value is reserved to indicate an error.

**Example function call:** prussdrv_init ( );

### prussdrv_open

Opens an event out and initializes memory mapping. This is a required function call. The input is a pru_evtout_num (PRU_EVTOUT_0 - PRU_EVTOUT_7) corresponding to Host2 - Host9 of the PRU INTC. If no events are required for the user code, choose PRU_EVTOUT_0 for the input parameter. Otherwise, call this function for each event required by the user code. Note if multiple events are used, this function is called multiple times.

**Function declaration:** int prussdrv_open (unsigned int pru_evtout_num)

**Return value:** The prussdrv_open() function shall return 0 upon successfully opening an event out and initializing memory mapping. Otherwise, -1 is returned.

**Example function call:** prussdrv_open ( PRU_EVTOUT_0 );

### prussdrv_pru_reset

Resets the PRU by invoking a soft reset in the PRU Control Register.

**Function declaration:** int prussdrv_pru_reset (unsigned int prunum)

**Return value:** The prussdrv_pru_reset() function shall return 0 upon successfully resetting the PRU. Otherwise, -1 is returned.

**Example function call:** prussdrv_pru_reset ( 0 ); //Resets PRU0

### prussdrv_pru_disable

Disables the PRU by writing 0 to the enable bit of the PRU Control Register.

**Function declaration:** int prussdrv_pru_disable (unsigned int prunum)

**Return value:** The prussdrv_pru_disable() function shall return 0 upon successfully disabling the PRU. Otherwise, -1 is returned.

**Example function call:** prussdrv_pru_disable ( 0 ); //Disables PRU0

### prussdrv_pru_enable

Enables the PRU by writing 1 to the enable bit of the PRU Control Register.

**Function declaration:** int prussdrv_pru_enable (unsigned int prunum)

**Return value:** The prussdrv_pru_enable() function shall return 0 upon successfully enabling the PRU. Otherwise, -1 is returned.

**Example function call:** prussdrv_pru_enable ( 0 ); //Enables PRU0

### prussdrv_pru_write_memory

Writes to either PRU Data RAM or Instruction RAM. Selects type of memory writing to and writes content at memarea pointer into memory. This function requires the input parameters described below. ***Note:*** PRUSS0_SHARED_DATARAM is only supported by AM335x.

1. *pru_ram_id* indicates the destination address and is defined within PRUSSDRV as:

   #define PRUSS0_PRU0_DATARAM 0

   #define PRUSS0_PRU1_DATARAM 1

   #define PRUSS0_PRU0_IRAM 2

   #define PRUSS0_PRU1_IRAM 3

   #define PRUSS0_SHARED_DATARAM 4

2. *wordoffset* is the destination offset.

3. **memarea* is a pointer to the starting address where data/ content is stored (source).

4. *bytelength* is the size of the content being written to PRU memory.

**Function declaration:** int prussdrv_pru_write_memory (unsigned int pru_ram_id, unsigned int wordoffset, unsigned int *memarea, unsigned int bytelength)

**Return value:** The prussdrv_pru_write_memory() function shall return 0 upon successfully writing to PRU Data RAM or Instruction RAM. Otherwise, -1 is returned.

**Example function call:**

*See prussdrv_exec_program() in prussdrv.c for example use case*

```
unsigned char fileDataArray[PRUSS_MAX_IRAM_SIZE];
...
prussdrv_pru_write_memory(PRUSS0_PRU0_IRAM, 0, (unsigned int *) fileDataArray, fileSize);
```

## prussdrv_pruintc_init

Initializes and enables the PRU interrupt controller. The input is a structure of arrays that determine which system events are enabled and how each is mapped to a host event. This structure (PRUSS_INTC_INITDATA) is defined in pruss_intc_mapping.h and can be modified by the user. Below is the PRUSS_INTC_INITDATA definition and description of each array in the structure:

```
#define PRUSS_INTC_INITDATA { \
    { PRU0_PRU1_INTERRUPT, PRU1_PRU0_INTERRUPT, PRU0_ARM_INTERRUPT, PRU1_ARM_INTERRUPT, ARM_PRU0_INTERRUPT, ARM_PRU1_INTERRUPT, -1 }, \
    { {PRU0_PRU1_INTERRUPT,CHANNEL1}, {PRU1_PRU0_INTERRUPT, CHANNEL0}, {PRU0_ARM_INTERRUPT,CHANNEL2}, {PRU1_ARM_INTERRUPT, CHANNEL3}, {ARM_PRU0_INTERRUPT, CHANNEL0},
        {ARM_PRU1_INTERRUPT, CHANNEL1}, {-1,-1}}, \
    { {CHANNEL0,PRU0}, {CHANNEL1, PRU1}, {CHANNEL2, PRU_EVTOUT0}, {CHANNEL3, PRU_EVTOUT1}, {-1,-1} }, \
    (PRU0_HOSTEN_MASK | PRU1_HOSTEN_MASK | PRU_EVTOUT0_HOSTEN_MASK | PRU_EVTOUT1_HOSTEN_MASK) /*Enable PRU0, PRU1, PRU_EVTOUT0 */ \
} \
```

Array 1: enables the system event numbers listed

Array 2: assigns system event numbers to channel numbers

Array 3: links channel numbers to host numbers

Array 4: creates mask to enable host interrupts (or event out numbers)

**Function declaration:** int prussdrv_pruintc_init (tpruss_intc_initdata *prussintc_init_data);

**Return value:** The prussdrv_pruintc_init() function shall return 0 upon successfully initializing and enabling the PRU interrupt controller. Otherwise, -1 is returned.

**Example function call:**

```
#include <pruss_intc_mapping.h>
void main (void) {
    ...
    tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;
    prussdrv_pruintc_init(&pruss_intc_initdata);
```

```
}
```

### prussdrv_map_l3mem

Maps the L3 memory to input pointer. Memory is then accessed by an array.
**Note:** this function is not supported for AM335x and is disabled by default ("#define DISABLE_L3RAM_SUPPORT" in __prussdrv.h).

**Function declaration:** int prussdrv_map_l3mem (void **address)

**Return value:** The prussdrv_map_l3mem() function shall return 0; no return value is reserved to indicate an error.

**Example function call:**

```
unsigned int *l3mem;
prussdrv_map_l3mem(&l3mem);
```

### prussdrv_map_extmem

Maps the DDR external memory to input pointer. Memory is then accessed by an array. **Note:** on AM18x, the base address of prussdrv_map_extmem is 0xC1000000 (not 0xC0000000).

**Function declaration:** int prussdrv_map_extmem(void **address)

**Return value:** The prussdrv_map_extmem() function shall return 0; no return value is reserved to indicate an error.

**Example function call:**

```
unsigned int *extmem;
prussdrv_map_extmem(&extmem);
```

### prussdrv_map_prumem

Maps the PRU DRAM and IRAM memory to input pointer. Memory is then accessed by an array. Minimum one event needs to be opened to access memory map. Call this function (prussdrv_map_prumem()) after the prussdrv_open(PRU_EVTOUT_x) function. The supported pru_ram_id values are below. ***Note:*** PRUSS0_SHARED_DATARAM is only supported by AM335x.

```
#define PRUSS0_PRU0_DATARAM 0
#define PRUSS0_PRU1_DATARAM 1
#define PRUSS0_PRU0_IRAM 2
#define PRUSS0_PRU1_IRAM 3
#define PRUSS0_SHARED_DATARAM 4
```

**Function declaration:** int prussdrv_map_prumem (unsigned int pru_ram_id, void **address)

**Return value:** The prussdrv_map_prumem() function shall return 0 upon successfully mapping the PRU DRAM or IRAM. Otherwise, -1 is returned.

**Example function call:**

```
unsigned int *pruDataMem;
#define PRUSS0_PRU0_DATARAM 0
prussdrv_map_prumem (PRUSS0_PRU0_DATARAM, &pruDataMem);
```

### prussdrv_map_peripheral_io

Maps the PRU subsystem peripherals memory in input pointer. Memory is then accessed by an array.
***Note:*** The prussdrv_map_peripheral_io function is only supported by AM335x.
The supported pru_id values are below:

```
#define PRUSS0_SHARED_DATARAM 4
#define PRUSS0_CFG 5
#define PRUSS0_UART 6
#define PRUSS0_IEP 7
#define PRUSS0_ECAP 8
#define PRUSS0_MII_RT 9
#define PRUSS0_MDIO 10
```

**Function declaration:** int prussdrv_map_peripheral_io (unsigned int per_id, void **address)

**Return value:** The prussdrv_map_peripheral_io() function shall return 0 upon successfully memory mapping the PRU subsystem peripheral. Otherwise, -1 is returned.

**Example function call:**

```
unsigned int *pruCFGmem;
#define PRU_CFG 5
prussdrv_map_peripheral_io(PRU_CFG, &pruCFGmem);
```

### prussdrv_get_phys_addr

Pass in a PRU/L3(if L3 RAM support enabled)/external memories pointer, or mmap returned value, and returns the corresponding physical address.

**Function declaration:** unsigned int prussdrv_get_phys_addr (void *address)

**Return value:** The prussdrv_get_phys_addr() function shall return the physical address associated with the mmap value input if successful. Otherwise, 0 is returned.

**Example function call:**

```
unsigned int phys_addr;
phys_addr = prussdrv_get_phys_addr (l3mem); //l3mem is a mmap returned value
```

### prussdrv_get_virt_addr

Pass in a PRU/L3(if L3 RAM support enabled)/external memory physical address, and returns a pointer containing the corresponding virtual address.

**Function declaration:** void *prussdrv_get_virt_addr (unsigned int phyaddr)

**Return value:** The prussdrv_get_virt_addr() function shall return the virtual address corresponding to the physical address input if successful. Otherwise, 0 is returned.

**Example function call:**

```
void *virt_addr;
unsigned int l3_phys_addr = 0x8000000;
virt_addr = prussdrv_get_virt_addr (l3_phys_addr);
```

### prussdrv_pru_wait_event

Waits for PRU event out. prussdrv_pru_wait_event() shall block the calling thread until an event corresponding to the event number input occurs.

**Function declaration:** int prussdrv_pru_wait_event (unsigned int pru_evtout_num)

**Return value:** The prussdrv_pru_wait_event() function shall return 0; no return value is reserved to indicate an error.

**Example function call:** prussdrv_pru_wait_event ( PRU_EVTOUT_0 );

### prussdrv_pru_send_event

Sends system event to PRU.

**Function declaration:** int prussdrv_pru_send_event (unsigned int eventnum)

**Return value:** The prussdrv_pru_send_event() function shall return 0; no return value is reserved to indicate an error.

**Example function call:** prussdrv_pru_send_event ( 32 );

### prussdrv_pru_clear_event

Clears system event.

**Function declaration:** int prussdrv_pru_clear_event (unsigned int eventnum)

**Return value:** The prussdrv_pru_clear_event() function shall return 0; no return value is reserved to indicate an error.

**Example function call:** prussdrv_pru_clear_event ( 32 );

### prussdrv_pru_send_wait_clear_event

Sends system event to PRU, waits for PRU event out, and clears system event. prussdrv_pru_send_wait_clear_event() shall block the calling thread until an event corresponding to the event number input occurs.

**Function declaration:** int prussdrv_pru_send_wait_clear_event (unsigned int send_eventnum, unsigned int pru_evtout_num, unsigned int ack_eventnum)

**Return value:** The prussdrv_pru_send_wait_clear_event() function shall return 0; no return value is reserved to indicate an error.

**Example function call:** prussdrv_pru_send_wait_clear_event ( 32, PRU_EVTOUT_0, 32);

### prussdrv_exit

Releases PRUSS clocks and disables prussdrv module.

**Function declaration:** int prussdrv_exit (void)

**Return value:** The prussdrv_exit() function shall return 0 upon successfully releasing PRU clocks and disabling the prussdrv module.

**Example function call:** prussdrv_exit ( );

### prussdrv_exec_program

Executes a binary file of opcodes on the PRU using the following steps:

1. Disable PRU
2. Write contents to PRU iRAM
3. Enable PRU to begin executing instructions stored in iRAM

**Function declaration:** int prussdrv_exec_program (int prunum, char *filename)

**Return value:** The prussdrv_exec_program() function shall return 0 upon successfully executing a program on the PRU. Otherwise, -1 is returned.

**Example function call:** prussdrv_exec_program ( 0, "./PRU_example.bin");

### prussdrv_start_irqthread

Initializes interrupt handler (or IRQ thread) for particular event. This function requires the following input parameters:

1. *pru_evtout_num*: the event number associated with (or hooked to) the interrupt handler.
2. *priority*: an integer, where a lower pririty number is given higher priority.
3. *irqhandler*: the interrupt handler name.

**Function declaration:** int prussdrv_start_irqthread (unsigned int pru_evtout_num, int priority, function_handler irqhandler)

**Return value:** The prussdrv_start_irqthread() function shall return 0 upon successfully initializing the IRQ thread. Otherwise, -1 is returned.

**Example function call:**

```
#include pthread
// Example interrupt handler thread
void *pruevtout0_thread(void *arg) {
    do {
        prussdrv_pru_wait_event (PRU_EVTOUT_0);
        // Handle event
        prussdrv_pru_clear_event (PRU0_ARM_INTERRUPT);
    } while (1);
}
void main (void) {
    ...
    prussdrv_start_irqthread (PRU_EVTOUT_0, sched_get_priority_max(SCHED_FIFO) - 2,
    pruevtout0_thread);
```

## Skeleton Application Code

The following is a skeleton application code that uses the PRUSSDRV APIs to load and execute a set of instructions to the PRU and handle an PRU-generated interrupt.

```
/* Driver header file */
#include <prussdrv.h>
#include <pruss_intc_mapping.h>
#define PRU_NUM 0
/* IRQ handler thread */
void *pruevtout0_thread(void *arg) {
        do {
                prussdrv_pru_wait_event (PRU_EVTOUT_0);
                prussdrv_pru_clear_event (PRU0_ARM_INTERRUPT);
        } while (1);
}
void main (void)
{
        /* Initialize structure used by prussdrv_pruintc_intc */
        /* PRUSS_INTC_INITDATA is found in pruss_intc_mapping.h */
        tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;
        /* Allocate and initialize memory */
        prussdrv_init ();
        prussdrv_open (PRU_EVTOUT_0);
        /* Map PRU's INTC */
        prussdrv_pruintc_init(&pruss_intc_initdata);
        /* Load and execute binary on PRU */
        prussdrv_exec_program (PRU_NUM, "./PRU_example.bin");
        /* Wait for event completion from PRU */
        prussdrv_pru_wait_event (PRU_EVTOUT_0); // This assumes the PRU generates an interrupt
                                                // connected to event out 0 immediately
                                                before halting
        /* Disable PRU and close memory mappings */
        prussdrv_pru_disable(PRU_NUM);
        prussdrv_exit ();
}
```

# AM335x PRU Linux-based Example Code

## Overview

The AM335x PRU support package contains example code demonstrating basic tasks executed by the PRU Subsystem. This page describes each example and illustrates the interaction between the ARM, PRU, peripherals, and memory.

Other PRU reference code can be found at http:/ / processors. wiki. ti. com/ index. php/ PRU_Linux-based_Example_Code.Note that this reference code is specific for AM18x and would need to be ported to run on AM335x.

## Building and installing examples

The following describes how to build and run the PRU Linux-based examples.

**NOTE**
Refer to the **PRU Linux Application Loader** article for details on steps 1, 2, and 4.

1. Build the UIO kernel driver.
2. Compile the PRUSSDRV user space library.
3. Compile example applications

   **$host** cd pru_sw/example_apps

   **$host** . /home/.../ti-sdk-am335x-evm-xx.xx.xx.xx/linux-devkit/environment-setup

   **$host** make

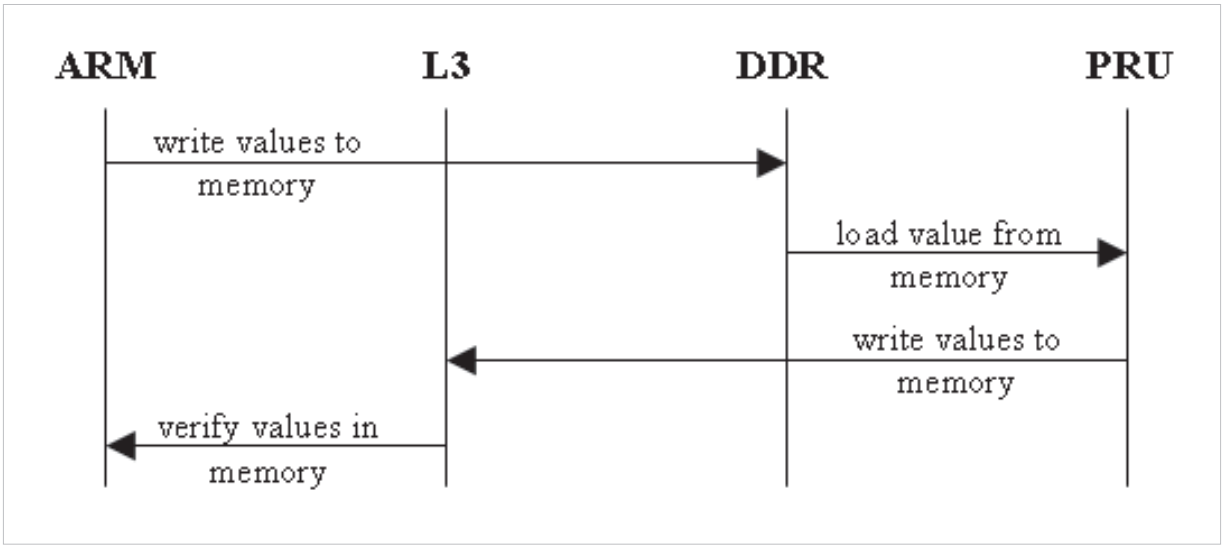
   **$host** cp bin/* <filesystem>
4. Run example applications on target platform.

## Example descriptions

### PRU_memAccessDDRandSharedRAM

The PRU reads three values from external DDR memory and stores these values in shared PRU RAM using the programmable constant table entries. The example initially loads 3 values into the external DDR RAM. The PRU configures its Constant Table Programmable Pointer Register 0 and 1 (CTPPR_0, 1) to point to appropriate locations in the DDR memory and the PRU shared RAM. The values are then read from the DDR memory and stored into the PRU shared RAM using the values in the 28th and 31st entries of the constant table.

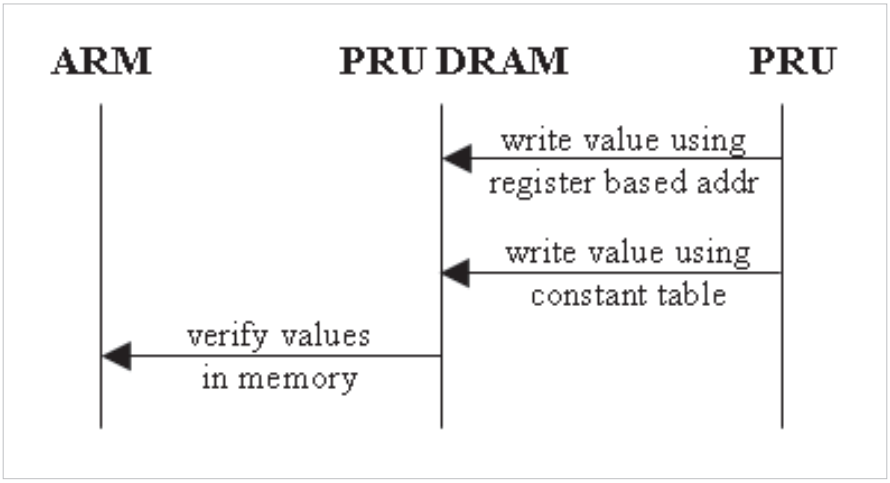
The diagram below illustrates the basic interaction between the ARM, PRU, and memory.




### PRU_memAccessPRUDataRam

The PRU reads and stores values into the PRU Data RAM memory. PRU Data RAM memory has an address in both the local data memory map and global memory map. The example accesses the local Data RAM using both the local address through a register pointed base address and the global address pointed by entries in the constant table.

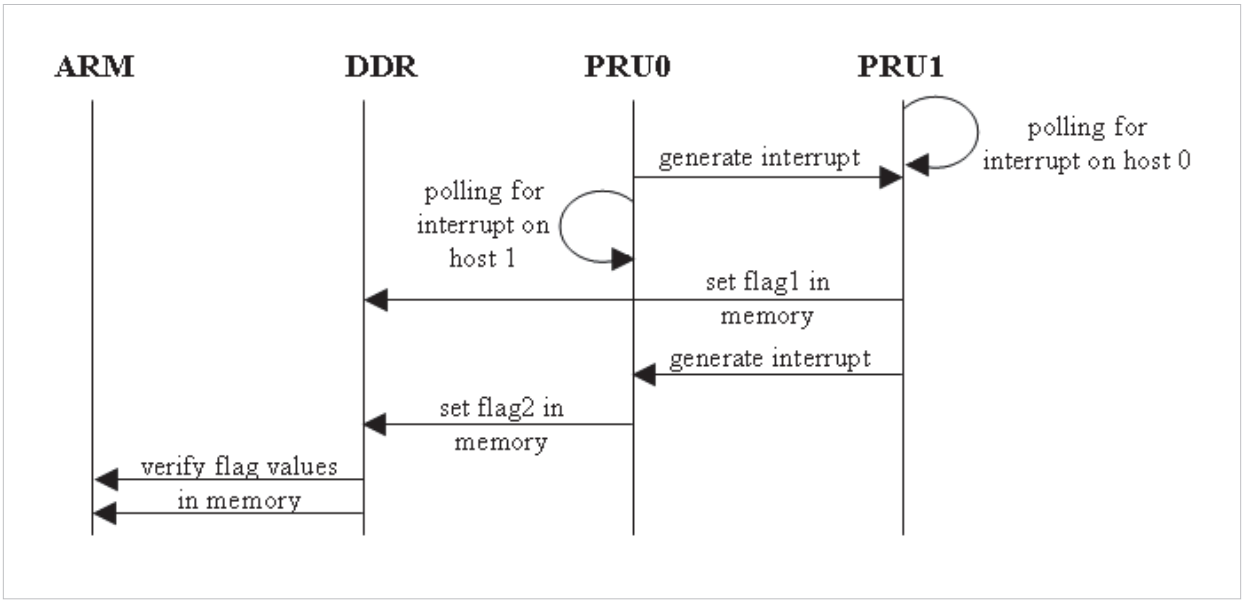
The diagram below illustrates the basic interaction between the ARM, PRU, and memory.




### PRU_PRUtoPRU_Interrupt

This example illustrates how two PRUs can communicate between each other by interrupting each other during a process. In this example code, the PRU0 configures the PRU INTC registers and connects system event 32 to channel 0 which in turn is hooked to the host port 0. The PRU0 then generates a system event 32 by writing into its R31 register which sends an interrupt to PRU1 which is polling for it. On receiving the interrupt, the PRU1 performs certain functionality and sets an external flag in DDR memory. The PRU1 completes its task and interrupts PRU0 once the task is done using system event 33 by first mapping this system event number to channel 1 and channel 1 to host 1 and then writing into its R31 register. The PRU0 polls for the interrupt and acknowledges the completion of task by setting another flag in DDR memory. The ARM checks the flag values in DDR memory to verify the example was successful.

The diagram below illustrates the basic interaction between the ARM, PRU, and memory.


ARM
DDR
PRU0
PRU1
polling for interrupt on host 0
generate interrupt
polling for interrupt on host 1
set flag1 in memory
generate interrupt
set flag2 in memory
verify flag values in memory