TI Designs Packet Processing Engine Reference Design for IEC61850 GOOSE Forwarding

Design Features

With TI-RTOS.

using CMEM.

Featured Applications
Protection Relays

Texas Instruments

Ethernet[®] Packet Switching Logic and GOOSE

Inter Processor Communication (IPC) Using

MessageQ in A15 With Linux, and M4 and DSP

M4/DSP Image Load and Boot by Linux[®] Kernel

Managed Contiguous Memory for Big Data Sharing

Between Processors Running Linux® and TI-RTOS,

Pin Multiplexing, Clock and Peripheral Initialization

TMDXIDK5728 Board and Includes Documentation, Software, Demo Application, and HW Design Files.

Determine Packet Destinations.

Driver Remoteproc Framework.

by U-Boot and Linux[®] Kernel Drivers. This Reference Design is Tested on the

Grid Infrastructure Communication

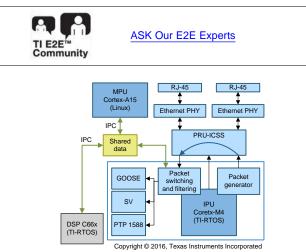
Filtering Algorithm Implemented on an M4 Core to

TI Designs

This design demonstrates packet switching and filtering logic implemented in the M4 core of AM572x based on the Ethertype, MAC address and Application ID (APPID) of GOOSE packets received from the PRU-ICSS. Packets are filtered and routed to destinations to allow the time-critical events defined in the substation communication standard IEC 51680 that must be serviced in a dedicated core. The design also shows multi-core communication between the ARM CortexTM A15 CortexTM M4 and DSP C66xTM core of the AM572x while Linux runs on the A15s and TI-RTOS runs on the M4 and DSP core.

Design Resources

| TIDEP0074 | Design Folder |
|---------------------------|----------------|
| AM572x | Product Folder |
| TMDXIDK5728 | Tool Folder |
| Processor-SDK-LINUX-AM57x | Tool Folder |
| Processor SDK-RTOS-AM57x | Product Folder |



An IMPORTANT NOTICE at the end of this TI reference design addresses authorized use, intellectual property matters and other important disclaimers and information.

Sitara, TMS320C66x, C64x+, C647x, Code Composer Studio are trademarks of Texas Instruments. ARM, Cortex, Thumb, Jazelle are registered trademarks of ARM Limited. EtherCAT is a registered trademark of EtherCAT Technology Group. Linux is a registered trademark of Linus Torvalds. Java is a trademark of Oracle. Wikipedia is a trademark of Wikipedia Foundation, Inc. EtherNet is a registered trademark of Xerox Corp.

AA)



2

www.ti.com

1 Introduction

The AM572x Sitara[™] ARM[®] processor is built to meet the complex processing needs of modern embedded products. AM572x provides high processing performance through the maximum flexibility of a fully integrated mixed processor solution with two ARM Cortex-A15 cores, two ARM Cortex-M4 cores, and two TI C66x DSP cores as well as two industrial communication subsystems (PRU-ICSS) which can be used for real-time communications and I/O applications. While the number of diverse cores on AM572x provides flexibility, the number of device cores also challenge software developers to fully exploit its multicore processing potential. To shorten the software development cycle and accelerate the time to market, TI provides a solution for inter processor communication (IPC) between homogenous and heterogeneous cores on the device. IPC 3.x is an evolution of the IPC product in TI Processor SDK (Software Development Kit), which abstracts the lower layer of processor fabric connection and offers a set of modules and APIs to facilitate inter-process communication.

The substation communication standard IEC61580 is becoming widely adopted in the grid infrastructure industry. In IEC 61850 architecture traditional wiring is eliminated and digitized signals are transmitted via communication interfaces which are largely Ethernet-based. As IEC61850 evolved, new features such as time-critical services, the Generic Object Oriented System Event (GOOSE) and Sample Values (SV), were added and led to requirements for real time communication and processing of events. In this design mixed traffic containing high-priority GOOSE packets, SV packets, PTP 1588 packets and other lower-priority packets is received into a PRU-ICSS Ethernet port. A M4 core is used to filter the traffic and process the GOOSE packets, route other packets to the A15 core, and also drop unwanted packets.



2 System Description

This TI Design leverages the messageQ example bundled in the IPC 3.x release, integrates PRU and ICSS_EMAC TI-RTOS drivers, and provides an implementation of Ethernet packet switching logic and a GOOSE filtering algorithm to demonstrate the packet data transfer from MII_RT of PRU-ICSS to desired destination cores as shown in Figure 1, for further real time processing.

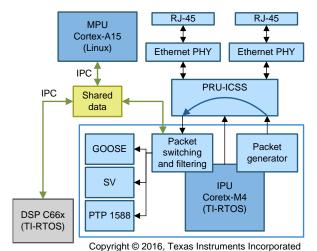


Figure 1. System Block Diagram

2.1 Multi-Core of AM572x Processor

AM572x is a high-performance Sitara processor based on ARM Cortex-A15s and TI C66x DSPs. The AM572x is designed for embedded applications including industrial communication, Human Machine Interface (HMI), grid infrastructure protection and communications, and other industrial use applications [1].

The device includes the following subsystems:

- ARM Cortex®-A15 microprocessor unit (MPU) subsystem, including two ARM® Cortex-A15 cores
- Two digital signal processor (DSP) C66x cores
- Two Cortex-M4 subsystems, each including two ARM Cortex-M4 cores
- Two dual-core Programmable Real-time Unit Industrial Communication Subsystems (PRU-ICSS)
- Graphics, video, real-time clock and debug subsystems

The device supports MMU/MPU:

- MMU used for key masters (Cortex-A15 MPU, Cortex-M4, C66x DSP, EDMA)
- Memory protection of C66x cores
- MMU inside the Dynamic Memory Manager
- The device also integrates:
- On-chip memory

External Memory Faces:

- Memory management
- Level 3 (L3) and level 4 (L4) interconnects
- System and serial peripherals



System Description

🤳 28 nm **High Speed Interconnect** Video Acceleration PRU (Quad Core) IVA HD 1080p Video, VPE Industrial ARM **Graphics Acceleration** Cortex-A15 Communication 3D GPU BB2D 32K/32K L1 ortex-M 32K/32K L1 288KB L2 Subsystem 2x SGX544 GC320 EtherCAT[®], PROFINET[®], 32KB L1 64KB RAM EtherNET/IP™, 2MB L2 Display Subsystem PROFIBUS. POWERLINK, SERCOS 3 HDMI 1.4a 2.5MB L3 Shared RAM w/ECC 3100 1080p Blend/Scale/ Convert 🖬 🕅 Security 32b DDR3/3L 32b DDR3/3L Acceleration Video Input Ports AES, MD5/SHA-256, SHA-2x16b 2x24b, 2x8b 2x24b, 2x8b 512, 3DES, TRNG System Services EDMA 13 mailbox RTC SDMA Secure WDT WDT Spinlock 16 Timer KBD Serial IO Industrial and programmable IO Storage IO 8 McASP GPIO DCAN PCle 3 SD/SDIO OSPI 5 12C SATA PWM/CAP/QEP 1 eMMC/ SD/SDIO NAND/ 10 UART 4 McSP USB3/2 USB2 Not available in AM5726, except VPE

Figure 2 shows the AM572x block diagram.

Figure 2. AM572x Block Diagram

2.1.1 Cortex A-15 MPU Subsystem

The Cortex-A15 MPU subsystem integrates the following sub-modules:

- ARM-Cortex-A15 MPCore
 - Two central processing units (CPUs)
 - ARM Version 7 ISA: Standard ARM instruction set plus Thumb[®] –2 Jazelle[®] RCT Java[™] accelerator, hardware virtualization support, and large physical address extensions (LPAE)
 - Neon SIMD co-processor and VFPv4perCPU
 - Interrupt controller with up to 160 interrupt requests
 - One general-purpose timer and one watchdog timer per CPU
 - Debug and trace features
 - 32-KiB instruction and 32-KiB data level 1 (L1) cache per CPU
- Shared 2-MiB level 2 (L2) cache
- 48-KiB bootable ROM
- Local power, reset, and clock management (PRCM) module
- Emulation features
- Digital phase-locked loop (DPLL)

2.1.2 DSP C66x Subsystem

4

There are two DSP subsystems in the device. Each DSP subsystem contains the following sub-modules:

- TMS320C66x[™] VLIW DSP core extends the performance of existing C64x+[™] and C647x[™] DSPs through enhancements and new features [1].
 - 32-KiB L1D and 32-KiB L1P cache or addressable SRAM

System Description



www.ti.com

- 288-KiB L2 cahce
- 256-KiB configurable as cache or SRAM
- 32-KiB SRAM
- Enhanced direct memory access (EDMA) engine for video and audio data transfer
- Memory management units (MMU) for address management
- Interrupt controller (INTC)
- Emulation capabilities

2.1.3 Cortex-M4 IPU Subsystem

There are two Cortex-M4 subsystems in the device [1]:

- IPU1 subsystem is available for general purpose usage
- IPU2 subsystem is dedicated to IVA-HD support and is not available for other processing Each subsystem includes the following components:
- Two Cortex-M4 CPUs
- ARMv7E-M and Thumb-2 instruction set architectures
- Hardware division and single-cycle multiplication acceleration
- Dedicated INTC with up to 63 physical interrupt events with 16-level priority
- Two-level memory subsystem hierarchy L1 (32-KiB shared cache memory) L2 ROM + RAM 64-KiB RAM
- 16-KiB bootable ROM
- MMU for address translation
- Integrated power management
- Emulation feature embedded in the Cortex-M4

2.1.4 PRU-ICSS

There are two Programmable Real-time Unit Industrial Communication Subsystems (PRU-ICSS) in the device. Each PRU-ICSS consists of dual 32-bit RISC cores (Programmable Real-Time Units, or PRUs), shared data and instruction memories, internal peripheral modules, and an interrupt controller (INTC) [1]. Among the interfaces supported by the PRU-ICSS are real-time industrial protocols used in master and slave mode, such as:

- EtherCAT[®]
- PROFINET
- EtherNet®/IP
- PROFIBUS
- Ethernet Powerlink
- SERCOS
- HSR
- PRP

For more details about the processor features, refer to the AM572x Sitara[™] Processors Silicon Revision 2.0, 1.1, Technical Reference Manual (SPRUHZ6E).



System Description

www.ti.com

2.2 IPC3.x

The IPC product provides software connectivity between multiple processors. Each processor may run either an HLOS (for example, Linux®, QNX) or an RTOS (for example, TI-RTOS). The IPC product abstracts device-specific support for hardware spinlocks, inter processor mailbox, and provides same APIs across devices (for example, GateMP implemented with hardware spinlock or software Peterson algorithm as needed).

IPC is an open source project, currently managed with git, and maintained at http://git.ti.com/ipc/ipcdev. Full source browsing, including all changes, are available at http://git.ti.com/ipc/ipcdev.git.

In previous generations of the IPC product, the IPC 1.x product includes implementations of those interfaces for the TI-RTOS, supports communicating between cores running TI-RTOS, as well to HLOS processors running SysLink 2.x. The SysLink 2.x product provides services to control slave processors (e.g. load, start, stop), also provides an implementation of the IPC interfaces for High Level OSs (HLOS) like Linux and QNX. SysLink 2.x supports communicating with slave processors running SYS/BIOS and IPC 1.x. PC 3.x merges the IPC 1.x and SysLink 2.x products, creating a single product that defines multiprocessor communication APIs and provides implementations for several OS's, including TI-RTOS and HLOS's [2]. The IPC product 3.x defines several interfaces to facilitate multiprocessor communication.

- Gate MP (TI-RTOS, Linux, QNX)
- HeapBufMP (TI-RTOS)
- HeapMemMP (TI-RTOS)
- HeapMultiBufMP (TI-RTOS)
- Ipc (TI-RTOS, Linux, QNX)
- ListMP (TI-RTOS)
- MessageQ (TI-RTOS, Linux, QNX)
- MultiProc (TI-RTOS, Linux, QNX)
- NameServer (TI-RTOS, Linux, QNX)
- Notify (TI-RTOS)

- SharedRegion (TI-RTOS)
- IpcPower (TI-RTOS)



2.2.1 Remoteproc and rpmsg

On Linux, IPC 3.x is built upon services available in the mainline Linux kernel (3.4+). These core services include *remoteproc* and *rpmsg*, except the Linux services, a few key services from the IPC API (e.g. MessageQ) are provided in user mode.

(On QNX, IPC 3.x provides feature parity to Linux. The QNX OS doesn't inherently provide primitives like Linux's *remoteproc* and *rpmsg*, and IPC 3.x also includes a loader and rpmsg-compatible communication infrastructure. This rpmsg-compatible MessageQ implementation enables the same TI-RTOS side image to communicate with either Linux or QNX on the HLOS [2].)

The *remoteproc* is a generic kernel component managing remote processors, abstracts hardware difference, and enables users access to these remote processors. The main functionalities implemented by *remoteproc* are device loading and bootup, power management, and exception management and error recovery [3].

When a new remote processor is registered in the *remoteproc* framework, a special section 'resource table' in addition to standard ELF segments of remote processor image is parsed. The resource table specifies the required memory entries by the TYPE_CARVEOUT resource type and the device memory entries through the TYPE_DEVMEM resource type to become MMU programmed, which also minimizes the effort to match the memory configurations between the HLOS and the TI-RTOS sides. The resource table also contains the resource entries of support feature, such as trace buffer.

The *rpmsg*, a communication channel, is a virtio-based messaging bus that allows kernel drivers to communicate with remote processors available on the system.

For more details on *remoteproc* and *rpmsg*, refer to Linux kernel documents at <u>https://www.kernel.org/doc/Documentation/remoteproc.txt</u> and <u>https://www.kernel.org/doc/Documentation/rpmsg.txt</u>.



System Description

8

2.2.2 MessageQ Module

MessageQ is the recommended messaging API for homogeneous and heterogeneous multi-processor messaging, or single-processor messaging between threads. The MessageQ module supports the structured sending and receiving of variable length messages.

The key features of the MessageQ module are [4]:

- Writers and readers may be relocated to another processor with no runtime code changes.
- Timeouts are allowed when receiving messages. ٠
- Readers may determine the writer and reply back.
- Receiving a message is deterministic when the timeout is zero.
- Messages may reside on any message queue.

Messages are sent and received via a message queue. A reader is a thread that gets (reads) messages from a message queue. A writer is a thread that puts (writes) a message to a message queue. Each message queue has one reader and can have many writers. A thread may read from or write to multiple message queues. The reader thread creates and owns the message queue. Writer threads then open a created message queue to get access to them.

Figure 3 shows the flow in which applications typically use the main runtime MessageQ APIs.

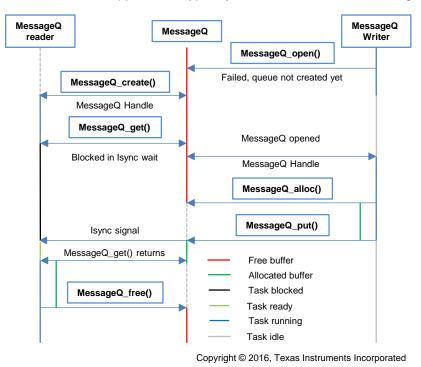


Figure 3. MessageQ Run Time



3 System Design Theory

The IPC demo code is based on ex02_messageq example (ipc_3_x/examples/DRA7XX_linux_elf/) from IPC release, with pruss and icss_emac TI-RTOS drivers integrated. The packet descriptors (data pointers) are transferred from M4 running TI-RTOS to A15 running Linux via messageQ. The large packet frame data is routed to A15 uding CMEM pool if needed. The data flow between A15 and DSP as shown in Figure 4, a series of testing packets which include broadcast, PTP 1588 multi-cast, SV and GOOSE is injected from M4 to PRU-ICSS2 MII_0, and then loops back to ICSS for processing. The packet received by PRU-ICSS is initially stored in its Rx queue in on chip L3 memory. The switch logic and GOOSE filtering algorithm is embedded in icss_emac API to determine if the packet should be dropped, transferred to GOOSE or PTP buffers, or forwarded to A15, see details in Section 3.6 Packet Switching Logic and Section 3.7 GOOSE Filter Algorithm.

The message passing between A15 and DSP is unmodified, only to demonstrate the IPC functionality as original ex02_messageq example in IPC release, where DSP creates a message to pass data around, and A15 sends a message to DSP core with a dummy payload, DSP then sends the message back to A15 core. The PRU-ICSS network traffic is not in the data path.

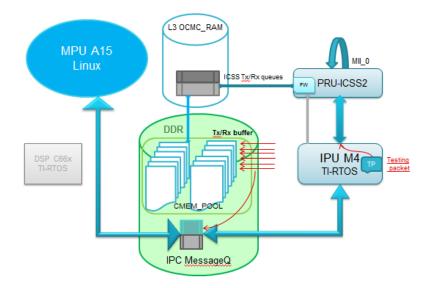


Figure 4. System Data Flow in PRU-ICSS, M4, and A15



System Design Theory

3.1 Pin Multiplexing, PRU-ICSS EMAC Kernel Driver Disabling

Board initialization, pin multiplexing is configured by U-Boot, which resolves the conflicts and specifies the I/O cell characteristics for the processor.

The recent Linux kernel in Processor SDK includes a PRU-ICSS eth driver that initializes the ICSS, IRAM, DRAM, MII_RT, interrupt controller, and registers interrupts for the ICSS EMAC ports during the driver probe. To make packets received by IPU M4 instead of MPU A15, the pruss2_eth port as below defined in the Linux device tree entry must not be brought up after kernel boot-up. PRU firmware will be loaded and started by IPU M4.

Figure 5 shows the pruss2_eth entry in the linux device tree.

```
/* Dual mac ethernet application node on icss2 */
pruss2_eth {
    compatible = "ti,am57-prueth";
    pruss = <&pruss2>;
    sram = <&ocmcram1>;

    pruss2_emac0: ethernet-mii0 {
        phy-handle = <&pruss2_eth0_phy>;
        phy-mode = "mii";
        sysevent-rx = <20>; /* PRU_ARM_EVENT0 */
        /* Filled in by bootloader */
        local-mac-address = [00 00 00 00 00];
    };
```

Figure 5. pruss2_eth Entry in the Linux Device Tree



3.2 ICSS_EMAC Driver APIs

The ICSS EMAC consists of a driver in host and firmware implementation. The driver refers to the code running on the host that is directly associated with the firmware that is loaded by host, but is running on the PRUs. The PRUs are responsible for transfer of packets while the host runs higher level tasks.

ICSS_EMAC Low-Level Driver (LLD) implements the following features [5]:

- Rx Copying packet received from firmware and providing it to 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

Below is the key APIs for packet processing in the ICSS_EMAC driver.

1. Packet Transmission

Figure 6 shows the ICSS_EMAC packet transmission API.

```
/**
 * Obrief Tx packet processing information block that needs to passed into call to
ICSS EmacTxPacket
 */
typedef struct
{
    ICSS_EmacHandle icssEmacHandle; /*! handle to ICSS EMAC Instance*/
    const uint8 t *srcAddress; /*! Base address of the buffer where the frame to
be transmitted resides */
    uint8 t portNumber; /*! Port on which frame has to be transmitted */
    uint8 t queuePriority; /*! Queue number in which frame will be queued for
transmission */
    uint16 t lengthOfPacket;
                               /*! length of the frame in bytes */
} ICSS EmacTxArgument;
/**
    @b Description
 *
 *
    ۵n
 *
        API to queue a frame which has to be transmitted on the
 *
        specified port queue
 *
                txArg defined at @ref ICSS EmacTxArgument
    @param[in]
 *
                userArg custom Tx packet callback packet options only required for
    @param[in]
custom TxPacket implementations,
                default to NULL when calling ICSS EmacTxPacket which is default Tx
Packet API
                0 on scuess, <0 on failure
 *
   @retval
 * /
```

int32_t ICSS_EmacTxPacket(ICSS_EmacTxArgument *txArg, void* userArg)

Figure 6. ICSS_EMAC Packet Transmission API

2. Packet Reception

Figure 7 shows the ICSS_EMAC packet reception API.



System Design Theory

www.ti.com

```
/**
* @brief Rx packet processing information block that needs to passed into call to
ICSS EmacRxPktGet
 */
typedef struct
{
    ICSS EmacHandle icssEmacHandle; /*! handle to ICSS EMAC Instance*/
    uint32 t destAddress; /*! Base address of data buffer where received frame has
to be stored */
    uint8 t queueNumber;
                           /*!Receive queue from which frame has to be copied */
    uint8 t port; /*!Returns port number on which frame was received */
    uint32 t more; /*!Returns more which is set to 1 if there are more frames in
the queue */
} ICSS EmacRxArgument;
/**
 *
    @b Description
 *
    Qn
 *
        Retrieves a frame from a host queue and copies it
 *
             in the allocated stack buffer
 *
 *
   @param[in]
               rxArg defined at @ref ICSS EmacRxArgument
 *
    @param[in]
               userArg custom Rx packet callback packet options only required for
custom RxPacket implementations,
                default to NULL when calling ICSS EmacRxPktGet which is default Tx
Packet API
                Length of the frame received in number of bytes or -1 on Failure
 *
   @retval
 */
int32 t ICSS EmacRxPktGet(ICSS EmacRxArgument *rxArg, void* userArg)
```

Figure 7. ICSS_EMAC Reception API

3. Packet Information

Figure 8 shows the ICSS_EMAC received packet information API.

```
/**
 *
    @b Description
 *
    Qn
 *
         API to retrieve the information about the received frame which
 *
         is then used to dequeue the frame from the host queues
 *
    @param[in] icssEmacHandle handle to ICSS EMAC Instance.
 *
    @param[out] portNumber
                              Return pointer of port number where frame was
received
 * @param[out]
                queueNumber Return pointer of host queue where the received
frame is queued
                           Return pointer of packet type, e.g GOOSE, SV, PTP etc.
* @param[out] pktProc
*
  @retval
               none
 */
int32 t ICSS EmacRxPktInfo(ICSS EmacHandle icssEmacHandle,
                           int32 t* portNumber,
                           int32 t* queueNumber,
                           int32 t* pktProc)
                  Figure 8. ICSS_EMAC Received Packet Information API
```

This is an extended API from default ICSS_EMAC LLD. A pktProc argument is added to report received packet type as the following macro definition:

#define PKT_PROC_NONE 0

Texas

TRUMENTS

```
#define PKT_PROC_GOOSE 1
#define PKT_PROC_PTP 3
#define PKT_PROC_SV 4
#define PKT_PROC_MPU 5
```

4. Callback Function Registration for Packet Receive Interrupt

Figure 9 shows the ICSS_EMAC Rx callback function registration API.

```
/**
 *
    @b Description
 *
    ۵n
 *
        API to register the hardware interrupt receive packet callback function
 *
 *
    @param[in] hwIntRx
                           hardware interrupt receive packet callback function
 *
    @retval
                none
 * /
void ICSS EmacRegisterHwIntRx (ICSS EmacHandle icssEmacHandle, ICSS EmacCallBack
hwIntRx)
```

Figure 9. ICSS_EMAC Rx Callback Function Registration API

5. Callback Function Registration for Packet Transmit Completion Interrupt Figure 10 shows the ICSS_EMAC Tx callback function registration API.

```
/**
 *
    @b Description
 *
    Qn
 *
        API to register the hardware interrupt for Transmit packet complete by PRU-
ICSS firmware
 *
    @param[in]
                           hardware interrupt transmit packet complete callback
                hwIntRx
function
 * @retval
                none
 * /
void ICSS EmacRegisterHwIntTx(ICSS EmacHandle icssEmacHandle, ICSS EmacCallBack
hwIntTx)
```

Figure 10. ICSS_EMAC Tx Callback Function Registration API

TEXAS INSTRUMENTS

www.ti.com

System Design Theory

3.3 PRU Driver APIs

PRU driver is a device abstraction layer APIs for the PRU Subsystem.

- PRU Low-Level Driver (LLD) implements following features [6]:
- PRU control features, enable, disable, and reset a PRU
- Helper functions, load and execute firmware in PRU
- Memory mapping of PRU, L3, and external memories
- PRU and House event management, for example, map sys_evt/channel/hosts in PRU INTC generate interrupts, wait for occurrence of an event, and acknowledge interrupts
- Interrupt management for A15/C66x CPU targets

The APIs for loading firmware to PRU and enabling PRU are:

1. Load PRU Firmware

Figure 11 shows the PRU memory write API.

```
/**
 *
            This function writes the given data to PRU memory
    @brief
 *
 * @param
                        Pruss's driver handle
           handle
 * @param pruMem
                       PRU Memory Macro [DATARAMO PHYS BASE
 * @param wordoffset Offset at which the write will happen.
 * @param
           source mem Source memory[ Array of uint32 tegers ]
            bytelength Total number of bytes to be writen
  0param
 *
  pruMem can have values
            PRU0 DATARAM\n
 *
            PRUO IRAM\n
 *
            PRU1 DATARAM\n
            PRU1 IRAM\n
            PRUICSS SHARED DATARAM
 *
  @return
                0 in case of successful transition, -1 otherwise.
 *
 **/
uint32 t PRUICSS pruWriteMemory(
                                 PRUICSS Handle handle,
                                 uint32 t pruMem,
                                 uint32 t wordoffset,
                                 const uint32 t *source mem,
                                 uint32 t bytelength
                               )
```

Figure 11. PRU Memory Write API

2. Enable PRU

Figure 12 shows the PRU enable API.

```
/**
 * @brief Enables PRU: \n
 *
 * @param handle Pruss's driver handle
 * @param pruNum PRU instance number[0 or 1].
 *
 * @return 0 in case of successful enable, -1 otherwise.
 **/
int32_t PRUICSS_pruEnable(PRUICSS_Handle handle,uint8_t pruNum)
```

Figure 12. PRU Enable API



3.4 IPC Customized Resource Table

The bit-banding feature in ARM Cortex-M4 architecture, which maps a complete word of memory from 0x4000:0000 to 0x400F:FFFF and 0x4200:0000 to 0x43FF:FFFF into a single bit in certain regions, results in the accesses must be performed indirectly using a virtual memory address. The memory mapping is defined in IPC resource table and as explained in Section 2.2.1, will be parsed by the Linux *remoteproc* to program M4 MMU.

A few more memory mapping entries below are added into the default IPC resource table in the TI design. IPC 3.x provides the capability for users to override the default.

- L3_PERPERIPHERAL_PRUSS (for PRU register access)
- IPU_EMIF_SDRAM (for CMEM section access)
- IPU_OCMC_RAM (for future use)

To use a non-default resource table, the resource custom table in the config file is set to true.

```
/* Override the default resource table */
```

var Resource = xdc.useModule('ti.ipc.remoteproc.Resource');

Resource.customTable = true;

Figure 13. Config to Use a Non-Default Resource Table

3.4.1 Resource Table Format

All ELF section placements are placed in memory allocated from the remoteproc CMA area and mapped to the virtual address as specified in the TYPE_CARVEOUT entries.

These are virtual addresses #defined in the resource table file ipu1/rsc_table_vayu_ipu.h:

| | IPU_MEM_TEXT IPU_MEM_DATA | 0x0 0x80000000 |
|---------|---------------------------------------|--------------------------|
| #define | IPU_MEM_IOBUFS | 0x90000000 |
| | IPU_MEM_IPC_DATA IPU_MEM_IPC_VRING | 0x9F000000 0x60000000 |

Figure 14. Defined Virtual Addresses

The following codes are fixed physical addresses to facilitate a fixed MMU table.

#if defined(VAYU_IPU_1)
#define PHYS_MEM_IPC_VRING
#elif defined (VAYU_IPU_2)
#define PHYS_MEM_IPC_VRING
#endif
Ox95800000

Figure 15. Defined Fixed Physical Address

These PHYS_MEM values match exactly the physical address specified in the remoteproc CMA area in Linux DTS file am572x-idk.dts:

Figure 16. CMA Pool Physical Addresses

Figure 17 shows the first entry in the resource table.

```
/* rpmsg vdev entry */
{
    TYPE_VDEV, VIRTIO_ID_RPMSG,0,
    RPMSG_IPU_C0_FEATURES,0,0,0,2,{0,0},
    /* no config data */
  },
    /* the two vrings */
    {IPU_MEM_RPMSG_VRING0,4096,IPU_RPMSG_VQ0_SIZE,1,0},
    {IPU_MEM_RPMSG_VRING1,4096,IPU_RPMSG_VQ1_SIZE,2,0},
```

Figure 17. First Entry

The above code directs the remoteproc to allocate the vrings and vring buffers from the CMA section. Figure 18 shows the second through the fourth entries.

```
{
    TYPE_CARVEOUT,
    IPU_MEM_TEXT, 0,
    IPU_MEM_TEXT_SIZE, 0, 0, "IPU_MEM_TEXT",
    },
    {
        TYPE_CARVEOUT,
        IPU_MEM_DATA, 0,
        IPU_MEM_DATA_SIZE, 0, 0, "IPU_MEM_DATA",
    },
    {
        TYPE_CARVEOUT,
        IPU_MEM_IPC_DATA, 0,
        IPU_MEM_IPC_DATA_SIZE, 0, 0, "IPU_MEM_IPC_DATA",
    },
}
```

Figure 18. Second Through Fourth Entries

These carveouts tell remoteproc to allocate memory from its CMA area and map the allocated physical address to the virtual address specified in the carveout, to the IPU's MMU.

The TYPE_TRACE entry tells remoteproc where the remote executable's trace buffer is, using its C symbol.

The TYPE_DEVMEM entries are virtual and physical mappings. remoteproc creates an IPU MMU mapping for the entry. The 1st TYPE_DEVMEM entry corresponds to the vrings and creates the IPU MMU mapping needed to access them from the IPU core, as shown in Figure 19.



```
{
    TYPE_DEVMEM,
    IPU_MEM_IPC_VRING, PHYS_MEM_IPC_VRING,
    IPU_MEM_IPC_VRING_SIZE, 0, 0, "IPU_MEM_IPC_VRING",
},
```

Figure 19. TYPE_DEVMEM Address Mapping

3.4.2 NEW TYPE_DEVMEM Entry

The following are the steps to add a new TYPE_DEVMEM entry, for example, to access PRU-ICSS from IPU.

- Specify the physical address of the PRU-ICSS and its virtual address. #define L3_PERIPHERAL_PRUSS 0x4B200000 #define IPU_PERIPHERAL_PRUSS 0x6B200000
- 2. Increase the size of offset[x] array in struct my_resource_table.
- 3. Increase the number of entries in ti_ipc_remoteproc_ResourceTable.
- 4. Add an offset of (struct my_resource_table, devmemY) in ti_ipc_remoreproc_ResourceTable.
- 5. Add an actual entry in ti_ipc_remoteproc_ResourceTable.

Figure 20 shows the actual entry.

```
{
    TYPE_DEVMEM,
    IPU_PERIPHERAL_PRUSS, L3_PERIPHERAL_PRUSS,
    SZ_1M, 0, 0, "IPU_PERIPHERAL_PRUSS",
},
```

Figure 20. A New TYPE_DEVMEM Entry

Refer to the rsc_table_vayu_ipu.h under ex02_messageq_icss/ipu1 for details.

The MMU pagetable can be dumped through debugs -

"cat/sys/kernel/debug/omap_iommu/<iommu_name>/pagetable"



System Design Theory

3.5 CMEM

The maximum size of IPC message queue including header is 512 bytes, consequently any larger data more than 512 bytes must be segmented for transfer, which leads to extra overhead. To achieve optimal throughput, we use CMEM pool to share packet data while messsageQ is used to transfer packet descriptors, for example, data pointers.

CMEM is an API and library for managing one or more blocks of physically contiguous memory. It also provides address translation services (for example, virtual and physical translation) and user-mode cache management APIs.

Using its pool-based configuration, CMEM enables users to avoid memory fragmentation, and ensures large physically contiguous memory blocks are available.

CMEM is actively being developed in the publicly maintained, TI-hosted 'ludev' git repository at http://git.ti.com/ipc/ludev.

3.5.1 CMEM Block

The CMEM blocks are defined in the Linux DTS file am57xx-evm-cmem.dtsi, and created when the cmemk.ko driver is installed during boot-up.

Figure 21 shows the CMEM block entry in the Linux device tree.

```
reserved-memory {
    #address-cells = <2>;
    #size-cells = <2>;
    ranges;
    cmem_block_mem_0: cmem_block_mem@a00000000 {
        reg = <0x0 0xa0000000 0x0 0x0c0000000>;
            no-map;
            status = "okay";
    };
```

Figure 21. CMEM Block Entry in the Linux Device Tree

3.5.2 CMEM API Usage

Figure 22 shows the fundamental usage of CMEM APIs (CMEM_init, CMEM_allocPhys, CMEM_map).

```
if (CMEM_init() != 0)
    printf("--->App_Create: ERROR: CMEM_init()\n");

cmem_buf_desc.physAddr = CMEM_allocPhys(4 * MAX_MII_PORTS_NUM *
MAX_PACKET_FRAME_SIZE, &alloc_params);

if (cmem_buf_desc.physAddr == 0 )
    printf("--->App_Create: ERROR: CMEM_allocPhys()\n");
else
    printf("--->App_Create: cmem_buf_desc.physAddr = 0x%x\n",
cmem_buf_desc.physAddr);

cmem_buf_desc.length = 4 * MAX_MII_PORTS_NUM * MAX_PACKET_FRAME_SIZE;
cmem_buf_desc.userAddr = CMEM_map((UInt32)cmem_buf_desc.physAddr,
cmem_buf_desc.length);
```

Figure 22. CMEM API Usage Example



•

3.6 Packet Switching Logic

A simplified packet switching logic implemented in the TI design is as followings:

- Switch on EtherType of the received packet:
- Case 0 × 88B8 (GOOSE packet): perform GOOSE filtering (see Section 3.7)
- Case 0 × 88F7 (PTP 1588 packet): process packets with certain multicast destination addresses, drop other packets
- Case 0 × 88BA (IEC 61850-9-2 Sampled Values packet): drop packets
- Default (all other packets): route packets to A15



System Design Theory

www.ti.com

3.7 GOOSE Filtering Algorithm

The GOOSE filtering algorithm is also simplified for demonstration purpose, and the pseudo code is as shown below:

```
If Goose filtering enabled
    If packet has GOOSE Ehtertpye (88B8)
    Drop packet not having APPID from the list of acceptable APPIDs
Else
    Route packet to A15 core
Else
    If packet has GOOSE Ethertype (88B8)
Else
    Route packet to A15 core
```

*The GOOSE Sever, to analyze and respond trip GOOSE messages, save GOOSE events and analysis results in historical data base, is not included in the design.

Figure 23 shows the GOOSE filtering flow.

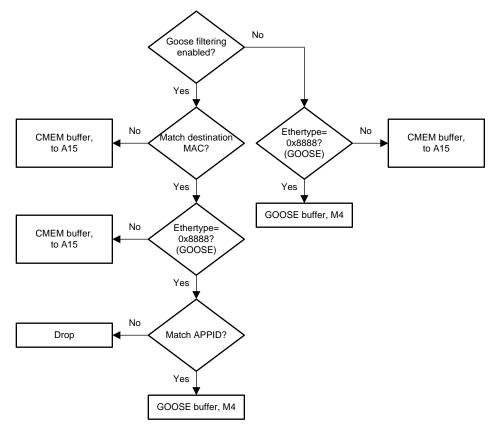


Figure 23. GOOSE Filtering Flow



3.8 Configurable GOOSE Filtering Parameters

Optionally, host A15 provides a text script goose in the etc directory to control and configure the GOOSE filtering parameters, including destination MAC address and APPID.

Figure 24 shows the /etc/goose format.

BEEFBEEF 11 22 33 44 55 66 22 33 44 55 66 77 33 44 55 66 77 88 44 55 66 77 88 99 1111 2222 3333 4444

Figure 24. GOOSE Filtering Parameter Script

Where the first line is the flag to enable or disable Goose filtering, BEEFBEEF = Enable, DEADDEAD = Disable, and other values will be ignored.

The next four lines are GOOSE packet destination mac address.

The last line is four GOOSE packet APPIDs

TEXAS INSTRUMENTS

Getting Started Hardware

www.ti.com

4 Getting Started Hardware

The required hardware for this design includes:

- AM572x IDK
- Ubuntu 12.04 + 64-bit PC
- Router with DHCP server
- Ethernet cable
- Ethernet loopback adaptor
- 4 GB + SD card
- 5.0 VDC Power supply
- 8. XDS-560M JTAG emulator (optional)

4.1 AM572x IDK EVM (TMDXIDK5728)

See Figure 25 while setting up AM572x IDK.

NOTE: The power supply for the IDK is 5.0 VDC whereas other Sitara EVMs may use 12 VDC.

The Ethernet loopback adaptor must be plugged in PRU2 ETH0 port for the application to properly run. The FTDI USB port is used as UART console port (baud 115200, 8N1). XDS560-M JTAG emulator is

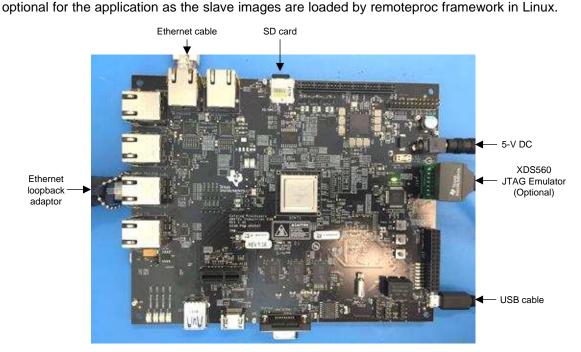


Figure 25. AM572x IDK EVM



5 Getting Started Software

The required software for this design includes:

- Code Composer Studio[™] Linux v6.1.2
- Linaro GCC ARM cross-compile toolchain
- Processor SDK Linux 2.0.2
- Processor SDK RTOS 2.0.2
- ex02_messageq_icss.tar.gz

5.1 Install Code Composer Studio

Code Composer Studio v6.1.2 is available at http://processors.wiki.ti.com/index.php/Download_CCS.

5.2 Install Linaro GCC ARM Cross-Compile Toolchain

Run the commands in the Ubuntu terminal:

\$ wget http://releases.linaro.org/15.05/components/toolchain/binaries/arm-linux-gnueabihf/gcc-linaro-4.9-2015.05-x86_64_arm-linux-gnueabihf.tar.xz

\$ tar -Jxvf gcc-linaro-4.9-2015.05-x86_64_arm-linux-gnueabihf.tar.xz -C \$HOME

5.3 Install Processor SDK Linux

Processor SDK Linux is available at http://software-dl.ti.com/processor-sw/esd/PROCESSOR-SDK-LINUX-AM57X/latest/index_FDS.html

\$ wget http://software-dl.ti.com/processor-sw/esd/PROCESSOR-SDK-LINUX-AM57X/latest/exports/tiprocessor-sdk-linux-am57xx-evm-02.00.01.07-Linux-x86-Install.bin.

\$ chmod +x ti-processor-sdk-linux-am57xx-evm-02.00.02.07-Linux-x86-Install.bin

\$ / ti-processor-sdk-linux-am57xx-evm-02.00.02.07-Linux-x86-Install.bin.

5.4 Install Processor SDK TI-RTOS

Processor SDK TI-RTOS is available at http://software-dl.ti.com/processor-sw/esd/PROCESSOR-SDK-RTOS-AM57X/latest/index_FDS.html.

\$ wget http://software-dl.ti.com/processor-sw/esd/PROCESSOR-SDK-RTOS-AM57X/latest/exports/tiprocessor-sdk-rtos-am57xx-evm-02.00.02.07-Linux-x86-Install.bin

\$ chmod +x ti-processor-sdk-rtos-am57xx-evm-02.00.01.07-Linux-x86-Install.bin

\$./ ti-processor-sdk-rtos-am57xx-evm-02.00.01.07-Linux-x86-Install.bin

5.5 Compile and Build Tools

Table 1 shows the IPC and application build environment variable settings.

Table 1. IPC Library and Application Build Environment Settings

| ENVIRONMENT VARIABLE | SETTING |
|-----------------------|---------------------------------------------------|
| TOOLCHAIN_INSTALL_DIR | gcc-linaro-4.9-2015.05-x86_64_arm-linux-gnueabihf |
| BIOS_INSTALL_DIR | bios_6_45_01_29 |
| XDC_INSTALL_DIR | xdctools_3_32_00_06_core |
| PDK_INSTALL_DIR | pdk_am57xx_1_0_2 |
| CSL_INSTALL_DIR | \$(PDK_INSTALL_DIR)/packages/ti/csl |
| ti.targets.arm.elf.M4 | \$(CCS)/ti-cgt-arm_5.2.5 |
| ti.targets.elf.C66 | \$(CCS)/ti-cgt-c6000_8.1.0 |
| gnu.targets.arm.A15F | \$(CCS)/gcc-arm-none-eabi-4_8-2014q3 |

Packet Processing Engine Reference Design for IEC61850 GOOSE 23 Forwarding



Getting Started Software

5.6 Build CMEM Library

Clone the CMEM source from git repo at http://git.ti.com/ipc/ludev.git, update products,mak file.

\$./configure --host=arm-linux-gnueabihf CC=/home/user/gcc-linaro-4.9-2015.05-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc --prefix=/home/user/ludev/generate

\$ make; sudo make install

5.7 Build IPC Libraries

Prior to building IPC libraries, Linux kernel must be built as IPC modules refer to it. Go to Processor SDK Linux install directory,

\$ cd board-support/linux-4.1.18+gitAUTOINC+01c1359baa-g01c1359

\$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- tisdk_am57xx-evm_defconfig

\$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage

After the Linux kernel is successfully built, change to the IPC release directory to start building the IPC libraries.

- 1. Follow the instructions found at http://processors.wiki.ti.com/index.php/IPC_Install_Guide_Linux#ipc-linux.mak.
- 2. Edit products.mak.
- 3. Build the Linux/TI-RTOS side IPC libraries for the platform.

5.8 Build Application

To build the application:

- Clone the application source code from <u>https://git.ti.com/apps/tidep0074</u>
 \$ git clone git://git.ti.com/apps/tidep0074.git
- 2. update products.mak
- 3. \$ make

The app_host is generated under host/bin/release/, server_dsp1.xe66 under dsp1/bin/release/ and server_ipu1.xem4 under ipu1/bin/release for testing.



6 **Test Setup**

TI recommends to run the application in a private local network. Figure 26 shows the test setup diagram that shows the Ethernet and USB serial port connection in AM572x IDK, Ubuntu 12.04+ 64-bit PC and router with DHCP server. Two Ethernet cards are in the Ubuntu PC for private and enterprise network connection respectively.

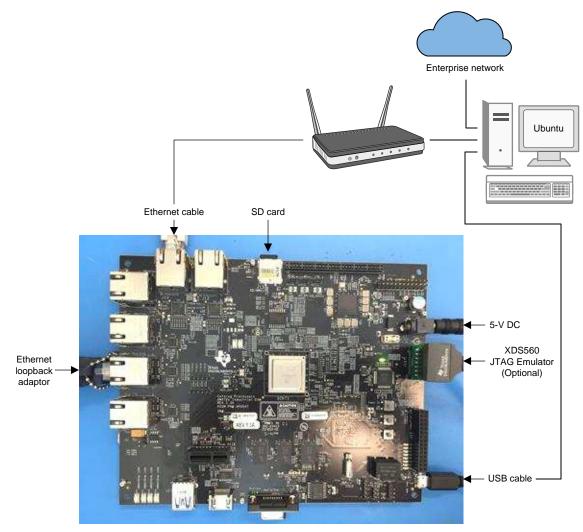


Figure 26. Test Setup

Test Setup



Test Setup

6.1 SD Card Setup

To prepare the SD card, see the *Processor SDK Linux Getting Started Guide* at http://processor_SDK_Linux_Getting_Started_Guide.

Copy the IPU1 and DSP1 binaries to the targe file system in the SD card, as listed in Table 2.

Table 2. IPU and DSP Binaries on the Target File System

| CORE | BINARY ON THE HOST | BINARY ON THE TARGET FILE SYSTEM |
|------|--------------------|-------------------------------------|
| IPU1 | server_ipu1.xem4 | /lib/firmware/dra7-ipu1-fw.xem4 |
| IPU2 | server_ipu2.xem4 | /lib/firmware/dra7-ipu2-fw.xem4 |
| DSP1 | server_dsp1.xe66 | /lib/firmware/dra7-dsp1-fw.xe66 |
| DSP2 | server_dsp2.xe66 | /lib/firmware/dra7-dsp2-fw.xe66 |

\$ cp ex02_messageq_icss/ipu1/bin/release/sever_ipu1.xem4 /media/rootfs/lib/firmware/dra7-ipu1-fw.xem4

\$ cp ex02_messageq_icss/dsp1/bin/release/sever_dsp1.xe66 /media/rootfs/lib/firmware/dra7-dsp1-fw.xe66

Copy the host application host_app binary to the target file system in the SD card.

\$ cp ex02_messageq_icss/host/bin/release/app_host /media/rootfs/home/root/



6.2 Linux Boot-Up

The following is the procedure to boot up Linux and the sanity check for running the application on the AM572x IDK.

- 1. Configure serial port console as 115200 baud 8N1.
- 2. Connect 5.0VDC to the AM572x IDK.
- 3. Press the SW3 button to power on the board.
- 4. Log in as toot without the password after the kernel has booted up.

Figure 27 shows a snapshot of Linux boot-up and login.



Figure 27. Snapshot of Linux Boot-Up and Login

5. Ensure that the necessary remoteproc, rpmsg, and MEM drivers are installed properly. Figure 28 shows the Linux kernel module list.



| root@am57xx-evm:~# | lsmod | | |
|----------------------|--------|-------------|---------------------------------------------|
| Module | Size | Us | ed by |
| bluetooth | 314443 | 2 | |
| ipv6 | 321017 | 26 | 5 |
| bc_example | 6681 | И | |
| adbserverproxy | 4597 | Ø | |
| cryptodev | 36165 | 5 | |
| cnenk | 30509 | 5 | |
| xhci plat hcd | 4517 | Ø | |
| xhci_hcd | 103063 | 1 | xhci_plat_hcd |
| usbcore | 192315 | 2 | xhci_plat_hcd,xhci_hcd |
| pru_rproc | 9301 | Я | |
| rpmsg_proto | 7445 | SS | |
| eudeu | 9585 | ø | |
| dwc3 | 61087 | Й | |
| udc_core | 11572 | 1 | dwc3 |
| virtio_rpmsg_bus | 12427 | | rpmsg_proto |
| snd_soc_omap_hdmi_ | | 541 | |
| palmas_pwrbutton | 3483 | Ø | |
| rtc_palmas | 5019 | ø | |
| leds_gpio | 3497 | ø | |
| gpio_pisosr | 2826 | | |
| gpio_tpic2810 | 2705 | 0 2 0 | |
| ov2659 | 9230 | Й | |
| phy_omap_usb2 | 5693 | 2 | |
| ti_vip | 29352 | 2 Ø | |
| c_can_platform | 6774 | ø | |
| c_can | 9582 | 1 | c_can_platform |
| extcon_usb_gpio | 2842 | 10 | o_odn_provorm |
| pursrukm | 413024 | 1 | bc_example |
| can_dev | 11969 | î | c_can |
| omap_sham | 17854 | Ø | 9_0(ii |
| pruss | 13855 | ĭ | pru_rproc |
| omap_aes_driver | 17710 | ñ | hre hree |
| omap_wdt | 4121 | 22 | |
| ahci nlatform | 2880 | й | |
| libahci_platform | 8445 | ĭ | ahci_platform |
| libahci | 27461 | | libahci_platform,ahci_platform |
| libata | 195836 | 3 | libahci,libahci_platform,ahci_platform |
| ti_vpe | 19368 | ğ | rindler, rindler_pressorer_pressorer |
| ti_sc | 24526 | 2 | ti_vip,ti_vpe |
| ti_vpdma | 14682 | 5 | ti_vip,ti_vpe |
| dwc3_omap | 4516 | õ | cr_vrp;cr_vpc |
| extcon | 13349 | ă | dwc3,dwc3_omap,extcon_usb_gpio |
| rtc_omap | 7505 | ğ | dweb,dweb_bmap,exceon_dsn_gpib |
| omap_des | 11617 | õ | |
| omap_ues omap_rng | 4533 | ŏ | |
| rng_core | 8050 | 2 | onap_rng |
| omap_remoteproc | 12083 | í | onop_rng |
| remoteproc | 26813 | à | omap_remoteproc,pruss,pru_rproc,rpmsg_proto |
| virtio | 7364 | 5 | remoteproc,virtio_rpmsg_bus |
| virtio_ring | 11848 | 5 | remoteproc,virtio_rpmsg_bus |
| debugss_kmodule | 6702 | ő | remoteprocivireio_rpmsg_bus |
| root@am57xx-evm:~# | | | |
| roorecansixx evil- # | | | |
| | | | |

Figure 28. Linux Kernel Module List

6. Type the following command into the console to verify that the CMEM pool is created.

\$ cat/proc/iomem

Figure 29 shows the CMEM pool list.

| 80000000-9fffffff : | System RAM |
|---------------------|------------|
| 80008000-80866073 | |
| 808ba000-809448cb | |
| a0000000-abffffff : | |
| ac000000-ffcfffff : | System RAM |
| root@am57xx-evm:~# | |

Figure 29. CMEM Pool List



7 Test Data

7.1 MPU A15 and IPU1 M4

When the kernel is booted up, remoteproc loads the IPU1 M4 binary gets the core booted up as well. To check the core status, look into its trace buffer and type in the following command in the Linux console.

\$ cat /sys/kernel/debug/remoteproc/remoteproc0/trace0

Figure 30 shows the IPU trace buffer log.

| | | "# cat /sys/kernel/debug/remoteproc/remoteproc0/trace0 |
|-----------|----------------|----------------------------------------------------------------------------|
| [0][| | Watchdog disabled: TimerBase = 0x68824000 ClkCtrl = 0x6a005568 |
| [0][| | Watchdog disabled: TimerBase = 0x68826000 ClkCtrl = 0x6a005570 |
| [0][| 0.0001 | 21 Resource entries at 0x3000 |
| [0][| 0.0001 | [t=0x000eead9] xdc.runtime.Main:> main: |
| [0][| 0.0001 | [t=0x00126ce1] xdc.runtime.Main: Pin Mux Config parameters addr |
| ess mappi | ng | |
| [0][| 0.000] | <pre>[t=0x0023f8bf] xdc.runtime.Main: main_pruss: initialization don</pre> |
| e! | | _ |
| [0][| 0.0001 | registering rpmsg-proto:rpmsg-proto service on 61 with HOST |
| 100 | | [t=0x00282ab5] xdc.runtime.Main: NameMap_sendMessage: HOST 53, |
| port=61 | | |
| 101 | 0.0001 | Watchdog disabled: TimerBase = 0x68824000 ClkCtrl = 0x6a005568 |
| 101 | | Watchdog disabled: TimerBase = 0x68826000 ClkCtrl = 0x6a005570 |
| 101 | | [t=0x002d2e2f] xdc.runtime.Main: taskPruss: Load firmware to PG |
| 2.0 AM572 | | |
| [0][| 0.0001 | |
| 1111 | 0.0001 | [t=0x002f210d] xdc.runtime.Main:> smain: |
| [0][| | [t=0x002fb50d] xdc.runtime.Main: link is finally up |
| [0][| 0.0001 | |
| [1]] | | [t=0x0031c8b5] Server: Server_create: server is ready |
| 1111 | | [t=0x0032ed05] Server: < Server_create: 0 |
| [1][| | [t=0x0033c0cb] Server:> Server_exec: |

Figure 30. IPU Trace Buffer Log

The message *Xdc.runtime.Main: link is finally up* indicates the MII_RT of PRU-ICSS has successfully established the Ethernet link.

Table 3 lists the debug trace file for each core.

| Table 3 | 3. IPU | and | DSP | Debug | Trace |
|---------|--------|-----|-----|-------|-------|
|---------|--------|-----|-----|-------|-------|

| CORE | DEBUG TRACE |
|------|-------------------------------------------------|
| IPU1 | /sys/kernel/debug/remoteproc/remoteproc0/trace0 |
| IPU2 | /sys/kernel/debug/remoteproc/remoteproc1/trace0 |
| DSP1 | /sys/kernel/debug/remoteproc/remoteproc2/trace0 |
| DSP2 | /sys/kernel/debug/remoteproc/remoteproc3/trace0 |



Test Data

www.ti.com

To run the application, execute the program host with IPU1 as an argument.

\$./app_host IPU1

Figure 31 shows the A15 and M4 communication log.

```
Main_main:
--> App_create:
--> App_Create:
--->App_Create: CMEM_allocPhys and map
--->App_Create: cmem_buf_desc.physAddr = 0xa00000000
Yetc/goose doesn't exsit, use default goose parameters
App_create: Host is ready
   App_create:
-> App_create:

-> App_exec:

pp_exec: sending message 1

pp_exec: sending message 2

pp_exec: sending message 3

pp_exec: sending message 4
App_exec: message received, sending message 5
App_exec: message received, sending message 6
App_exec: message received, sending message 7
App_exec: message received, sending message 8
 00000000000000000
                        eceived packet frame size: 256
received packet frame size: 256
App_exec: message received, sending message 10
App_exec: packet received:
received packet frame size: 0
App_exec: message received, sending message 11
App_exec: packet received:
received packet frame size: Ø
```

Figure 31. A15 and M4 Communication Log

Observe the packet stream received and routed to the MPU A15, with each frame size 256 bytes. Also check the IPU M4 trace buffer which shows the packet switching underneath.

Figure 32 shows the IPU trace buffer log after application execution.

| [1][| 255.958] | [t=0x00000018:5db97653] | Server: Server_exec: processed cmd=0x40000000 |
|--------------|----------|-------------------------|------------------------------------------------|
| [0][| 255.9591 | [t=0x00000018:5dbc13ad] | xdc.runtime.Main: place packet in A15 queue |
| 1111 | 255 9591 | [t=0v00000018:5dbdeb8b1 | xdc.runtime.Main: Packet Received! |
| | | | Server: Server_exec: processed cmd=0x40000000 |
| [1][[0][| | | |
| | | | xdc.runtime.Main: place packet in A15 queue |
| [1][| 255.9601 | [t=0x00000018:5dc44cd5] | xdc.runtime.Main: Packet Received! |
| [1][| | | Server: Server_exec: processed cmd=0x40000000 |
| 1001 | | | xdc.runtime.Main: place packet in Goose buffer |
| [1][| 255.961] | [t=0x00000018:5dcab0bb] | xdc.runtime.Main: Packet Received! |
| [1][| 255.961] | [t=0x00000018:5dcc4c51] | Server: Server_exec: processed cmd=0x40000000 |
| [0][| 255.9621 | [t=0x00000018:5dcecadf] | xdc.runtime.Main: place packet in Goose buffer |
| [1][| 255.962] | [t=0x00000018:5dd0a54d] | xdc.runtime.Main: Packet Received! |
| [1][| 255.962] | [t=0x00000018:5dd23af9] | Server: Server_exec: processed cmd=0x40000000 |
| [0][| | | xdc.runtime.Main: place packet in A15 queue |
| [1][| 255.9631 | [t=0x00000018:5dd6a12f] | xdc.runtime.Main: Packet Received! |
| [1][| 255.963] | [t=0x00000018:5dd83c3f] | Server: Server_exec: processed cmd=0x40000000 |
| [0][| | | xdc.runtime.Main: place packet in A15 queue |
| [1][| | | xdc.runtime.Main: Packet Received! |
| [1][| 255.9641 | [t=0x00000018:5ddea64d] | Server: Server_exec: processed cmd=0x40000000 |
| [0][| 255.9651 | [t=0x00000018:5de182d9] | xdc.runtime.Main: place packet in A15 queue |
| [1][| 255.9651 | [t=0x00000018:5de37067] | xdc.runtime.Main: Packet Received! |

Figure 32. IPU Trace Buffer Log After Application Execution



7.2 MPU A15 and DSP C66x

Execute the host program with DSP1 as an argument.

\$./app.host DSP1

Figure 33 shows the A15 and DSP communication log.

| > App_create: >App_Create: CMEM_allocPhys and map >App_Create: cme_buf_desc.physAddr = 0xa0000000 /etc/goose doesn't exsit, use default goose parameters App_create: Host is ready < App_create: -> App_exec: |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| App_exec: sending message 1 |
| App_exec: sending message 2 |
| App_exec: sending message 3 |
| App_exec: sending message 4 |
| App_exec: message received, sending message 5 |
| App_exec: message received, sending message 6 |
| App_exec: message received, sending message ? |
| App_exec: message received, sending message 8 |
| App_exec: packet received: |
| received packet frame size: 0 |
| App_exec: message received, sending message 9 |
| App_exec: packet received: |
| |
| received packet frame size: Ø |

Figure 33. A15 and DSP Communication Log

The packet transfer is implemented in IPU1 M4, thus, the log only shows message transfer without packet data.

7.3 IPU M4 and DSP C66x

The application does not create a message queue between IPU M4 and DSP C66x. Fore more information, refer to the example code ex41_forwardmsg under ipc_3_x/examples/DRA7XX_linux_elf, which demonstrates message passing between each core. The host sends a message to each slave core with a dummy payload. Each slave then forwards the message to the other slave. After a slave core receives a message from both the host and another slave core, the slave core allocates a new message and sends the message back to the host. The slave shuts itself down and reinitializes for future runs.

7.4 IPC Benchmarking

IPC release provides the benchmark application that may be used to measure the round-trip latency of messageQ. See the Wikipedia[™] page at <u>http://processors.wiki.ti.com/index.php/IPC_BenchMarking</u>. Table 4 lists the benchmark result using the MessageQBench collected with Processor SDK 2.0.2 release on AM572x IDK.

| CORES | MESSAGEQ Round-TRIP LATENCY (us) |
|-----------------|----------------------------------|
| ARM <->DSP C66x | 125 |
| ARM <->IPU M4 | 140 |

Table 4. IPC MessageQ Round Trip Latency



Design Files

8 Design Files

8.1 Bill of Materials (BOM)

To download the bill of materials (BOM), see the design files at http://www.ti.com/tool/TIDEP0074.

8.2 Software Files

To download the software files, see the design files at http://www.ti.com/tool/TIDEP0074

9 References

- 1. AM572x Sitara Processors Silicion Revision 2.0, 1.1, Technical Reference Manual, (SPRUHZ6)
- 2. TI wiki, IPC 3.xhttp://processors.wiki.ti.com/index.php/IPC_3.x
- 3. OMAPpedia, http://omappedia.org/wiki/Design_Overview_-_RPMsg
- 4. TI wiki, IPC Users Guide/Message Q Module, http://processors.wiki.ti.com/index.php/IPC_Users_Guide/MessageQ_Module
- 5. TI wiki, Processor SDK RTOS ICSS-EMAC-Design, http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_ICSS-EMAC-Design
- 6. TI wiki, *Processor SDK RTOS PRUSS*, <u>http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_PRUSS</u>
- 7. TI wiki, IPC Resource customTable, http://processors.wiki.ti.com/index.php/IPC_Resource_customTable
- 8. *IPC Users Guide*, <u>http://processors.wiki.ti.com/index.php/IPC_Users_Guide</u>
- 9. Processor SDK RTOS ICSS-EMAC-Design, http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_ICSS-EMAC-Design
- 10. PRU-ICSS Sub-System, http://www.ti.com/lsds/ti/processors/applications/industrial/industrial_communications.page
- 11. CMEM Overview, http://processors.wiki.ti.com/index.php/CMEM_Overview

10 Terminology

- **APPID** Application ID
- **CMA** Contiguous Memory Allocator
- **CMEM** Contiguous Memory Management
- **GOOSE** Generic Object Oriented System Event
- IPC Inter Processor Communication
- ICSS Industrial Communication Subsystem
- MII Media Independent Interface
- **MMU** Memory Management Unit
- PRU Programmable Real-time Unit
- RTOS Real-Time Operating System
- SV Sampled Value



11 About the Author

GARRETT DING is a software applications engineer for the Embedded Processing Group at Texas Instruments, where he is responsible for developing reference design solutions and providing technical support to customers for the industrial segment. Garrett brings to this role his extensive experience in multi-core Linux and TI-RTOS software design expertise. Garrett earned his Master of Science in Electrical Engineering (MSEE) from NanJing University of Science & Technology, China.

IMPORTANT NOTICE FOR TI REFERENCE DESIGNS

Texas Instruments Incorporated ('TI') reference designs are solely intended to assist designers ("Designer(s)") who are developing systems that incorporate TI products. TI has not conducted any testing other than that specifically described in the published documentation for a particular reference design.

TI's provision of reference designs and any other technical, applications or design advice, quality characterization, reliability data or other information or services does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such reference designs or other items.

TI reserves the right to make corrections, enhancements, improvements and other changes to its reference designs and other items.

Designer understands and agrees that Designer remains responsible for using its independent analysis, evaluation and judgment in designing Designer's systems and products, and has full and exclusive responsibility to assure the safety of its products and compliance of its products (and of all TI products used in or for such Designer's products) with all applicable regulations, laws and other applicable requirements. Designer represents that, with respect to its applications, it has all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. Designer agrees that prior to using or distributing any systems that include TI products, Designer will thoroughly test such systems and the functionality of such TI products as used in such systems. Designer may not use any TI products in life-critical medical equipment unless authorized officers of the parties have executed a special contract specifically governing such use. Life-critical medical equipment is medical equipment where failure of such equipment would cause serious bodily injury or death (e.g., life support, pacemakers, defibrillators, heart pumps, neurostimulators, and implantables). Such equivalent classifications outside the U.S.

Designers are authorized to use, copy and modify any individual TI reference design only in connection with the development of end products that include the TI product(s) identified in that reference design. HOWEVER, NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of the reference design or other items described above may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI REFERENCE DESIGNS AND OTHER ITEMS DESCRIBED ABOVE ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY DESIGNERS AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS AS DESCRIBED IN A TI REFERENCE DESIGN OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TI's standard terms of sale for semiconductor products (<u>http://www.ti.com/sc/docs/stdterms.htm</u>) apply to the sale of packaged integrated circuit products. Additional terms may apply to the use or sale of other types of TI products and services.

Designer will fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of Designer's noncompliance with the terms and provisions of this Notice.

> Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265 Copyright © 2016, Texas Instruments Incorporated