Application Report
SPRAC60–November 2016

AM57x Processor SDK Linux®: Customizing Multicore Applications to Run on New Platforms

ABSTRACT

When customers develop applications that use multiple programmable cores on the AM57x device, they require a clear understanding of roles and configurations of multiple software (SW) components such as IPC, CMEM, CMA, Linux™, and SYS/BIOS on slave cores to arrive at correct configuration for their application. This application report describes memory utilization schemes by A15, DSP, and IPU, how they are related, and what must happen in the process of adapting Processor SDK configuration to a custom one.

Contents

Trademarks

1 Abbreviations

The following list provides abbreviations and their meanings used in this document.

- Processor SDK – Processor software development kit (SDK). In this document, Processor SDK always refers to Processor SDK Linux, not Processor SDK RTOS.
- IPU – Image processing unit (dual-core ARM® Cortex®-M4 subsystem on AM57x devices)
- DSP – Digital signal processor (C66 DSP core on AM57x devices)
- CMA – Linux framework allows setting up a machine-specific configuration for physically-contiguous memory management.
- CMEM – Linux utility provides an application programming interface (API) and library for managing one or more blocks of physically contiguous memory.
- IPC – Inter-processor communication
- DTS – Device tree source of Linux

2 Introduction

The AM57x family of system-on-chip (SoC) devices provides high processing performance through the maximum flexibility of a fully-integrated mixed processor solution with up to two ARM Cortex-A15 cores, up to two TI C66x DSP cores, and two ARM Cortex-M4 cores.

Processor SDK is a SW development kit developed by TI that has tools and components to enable SW development on a SoC and collection of examples aimed at demonstrating SoC capabilities as well as serving as a starting point for application development. While Processor SDK provides a basic memory configuration for typical user cases, customers may need to update the memory configuration to align with their hardware system and applications.

This application report provides the details of memory use by multicore applications running on the AM57x SoC family of devices when the Cortex-A15 is running Linux, describes tools and frameworks used for memory management, and guides through changes required to adapt Processor SDK memory map for custom design. The code snippets referred in this document are based on Processor SDK 3.0.0.4 and IPC 3_43_01_03. The hardware platform is AM572x IDK.

3 Key Components

3.1 IPC 3.0

IPC 3.x is an evolution of the IPC product in the TI Processor SDK that abstracts the lower layer of processor fabric connection and offers a set of modules and APIs to facilitate inter-process communication. IPC 3.0 supports both Linux to SYS/BIOS and SYS/BIOS-to-SYS/BIOS communication. This application report focuses on the Linux to SYS/BIOS IPC including A15 to DSP as well as A15 to IPU. Detailed descriptions of IPC 3.0, user guide, examples, and training materials can be found at the *IPC 3.x* wiki.

3.2 CMA

CMA is a Linux tool allowing for static allocation of big physically contiguous memory blocks. In Processors SDK it is used to allocate static memory regions that are accessible from Linux, as well as DSP and IPU cores. CMA memory pools are used to store DSP and IPU application code (loaded by the Linux during SoC initialization) as well as IPC buffers. A detailed description can be found at the following location: *A deep dive into CMA*.

3.3 CMEM

CMEM is a kernel module developed by TI that allows for dynamic creation and management of one or more blocks of contiguous memory for exchanging data buffers between Linux running on A15 and SYS/BIOS running on DSP or IPU. CMEM enables users to avoid memory fragmentation and ensures large physically contiguous memory blocks are available by using pool-based configuration of CMEM.

In the Processor SDK for the AM57x family, CMEM allocates buffers for data that the A15 sends to the DSP or IPU for processing. A detailed description of CMEM can be found at the *CMEM Overview* wiki page.

3.4 *SYS/BIOS*

SYS/BIOS is a RTOS kernel developed by TI that runs on DSP and IPU cores on an AM57x device. SYS/BIOS can also run on the A15, but this scheme is outside of the scope of this document. For details, refer to Processor SDK RTOS documentation.

In this application scenario, the A15 runs Linux. SYS/BIOS includes the RTOS kernel and memory management facilities, and is used to configure and run the DSP and IPU application code. A detailed description of SYS/BIOS can be found at the *Welcome to SYS/BIOS* wiki page.

4 AM57x Memory Map Configuration

Processor SDK provides default memory map designed to accommodate memory installed on TI EVMs and run all examples and demonstrations. In customers' designs this memory map has to be adapted to the needs of each particular application. This section presents the default PSDK memory map and lists the required changes. This memory map for the use case of A15 running Linux is captured in the Linux device tree for the particular platform.

Table 1 shows the default PSDK memory map.

Table 1. Default PSDK Memory Map

Memory Section	Physical Address
A15 Linux Kernel	0x80000000
IPU2 CMA	0x95800000
DSP1 CMA	0x99000000
IPU1 CMA	0x9D000000
DSP2 CMA	0x9F000000
CMEM	0xA0000000

4.1 *Change Memory Map From Default Processor SDK to Accommodate Installed Memory*

The AM572x IDK has 2GB DDR3L memory. The memory configuration is defined in the following Linux device tree: am572x-idk.dts at board-support/linux-4.4.12+gitAUTOINC+3639bea54a-g3639bea54a/arch/arm/boot/dts/.

In the following code snippet, the first 0x80000000 is the DDR memory starting address. The second 0x80000000 is the size of the DDR memory.

```
memory {
    device_type = "memory";
    reg = <0x0 0x80000000 0x0 0x80000000>;
};
```

For example, if a system has only 512MB of memory, the memory node in DTS must be updated to:

```
memory {
    device_type = "memory";
    reg = <0x0 0x80000000 0x0 0x20000000>;
};
```

4.2 Change CMA Pool Size and Location

The CMA pools are defined in the following Linux device tree: am572x-idk.dts.

```
reserved-memory {
    #address-cells = <2>;
 #size-cells = <2>;
 ranges;

 ipu2_cma_pool: ipu2_cma@95800000 {
  compatible = "shared-dma-pool";
  reg = <0x0 0x95800000 0x0 0x3800000>;
  reusable;
  status = "okay";
 };

 dsp1_cma_pool: dsp1_cma@99000000 {
     compatible = "shared-dma-pool";
  reg = <0x0 0x99000000 0x0 0x4000000>;
  reusable;
  status = "okay";
 };

 ipu1_cma_pool: ipu1_cma@9d000000 {
  compatible = "shared-dma-pool";
  reg = <0x0 0x9d000000 0x0 0x2000000>;
  reusable;
  status = "okay";
 };

 dsp2_cma_pool: dsp2_cma@9f000000 {
  compatible = "shared-dma-pool";
  reg = <0x0 0x9f000000 0x0 0x800000>;
  reusable;
  status = "okay";
    };
};
```

The four CMA pools in the previous code snippet are dedicated for IPU2, DSP1, IPU1, and DSP2, respectively. The reg entry defines the CMA pool starting address and size. For example, ref = <0x0 0x95800000 0x0 0x3800000>; means the allocated CMA pool starts from 0x95800000 and has a size of 0x3800000 bytes. These are physical addresses in the DDR3 memory.

Modify the entries to accommodate the code, data, and heap memory requirements of custom DSP or IPU applications.

4.3 Change CMEM Configuration and Allocation

The CMEM block is configured in the following Linux device tree: am57xx-evm-cmem.dtsi at am572x-idk.dts at board-support/linux-4.4.12+gitAUTOINC+3639bea54a-g3639bea54a/arch/arm/boot/dts/.

```
/ {
    reserved-memory {
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;

        cmem_block_mem_0: cmem_block_mem@a0000000 {    reg = <0x0 0xa0000000 0x0 0x0c000000>;
            no-map;
            status = "okay";
        };

        cmem_block_mem_1_ocmc3: cmem_block_mem@40500000 {
            reg = <0x0 0x40500000 0x0 0x100000>;
            no-map;
            status = "okay";
        };
    };

    cmem {
        compatible = "ti,cmem";
        #address-cells = <1>;
        #size-cells = <0>;

        #pool-size-cells = <2>;

        status = "okay";

        cmem_block_0: cmem_block@0 {    reg = <0>;   memory-region = <&cmem_block_mem_0>;   cmem-
buf-pools = <1 0x0 0x0c000000>;
    };

    cmem_block_1: cmem_block@1 {
        reg = <1>;
        memory-region = <&cmem_block_mem_1_ocmc3>;
        };
    };
};
```

Two CMEM blocks are defined in Processor SDK. CMEM block 0 is allocated from DDR memory starting from 0xa0000000 with a size of 0x0c000000 bytes (configured in reg = <0x0 0xa0000000 0x0 0x0c000000>;). Entry cmem-buf-pools = <1 0x0 0x0c000000> specifies that one buffer with the size 0x0c000000 is allocated from CMEM block 0.

CMEM block 1 is allocated from OCMC memory, starting from 0x40500000 with a size of 0x0100000 bytes. Users can add more CMEM blocks or modify the CMEM block size as needed (see the following code snippet).

```
cmem_block_mem_2: cmem_block_mem@d0000000 {
    reg = <0x0 0xd0000000 0x0 0x0c000000>;
    no-map;
    status = "okay";
};

cmem_block_2: cmem_block@2 {
    reg = <0>;
    memory-region = <&cmem_block_mem_2>;
    cmem-buf-pools = <1 0x0 0x0c000000>;
};
```

4.4 Change IPU and DSP Resource Table to Define Memory Use by Firmware

The resource table is a Linux construct that informs the Linux kernel remoteproc driver about the available resources of the remote processor, and typically refers to memory and local peripheral registers. When a remote processor image is loaded, the remoteproc driver will parse the system resources defined in the resource table, which is linked into the remote processor image. Also, the remoteproc allocates rpmsg vring buffers, trace buffers, and configures MMUs according to the resource table. DSP and IPU images need to be built with appropriate resource table to match the partitioned memory from the Linux device tree.

The DSP and IPU resource table for the AM57x is distributed in the Processor SDK RTOS package, and is located in the IPC_<version> directory. For example, at the IPC_3_43_01_03 directory.

- packages/ti/ipc/remoteproc/rsc_table_vayu_dsp.h
- packages/ti/ipc/remoteproc/rsc_table_vayu_ipu.h

To use a customized resource table, users must modify the SYS/BIOS configuration file for DSP or IPU to set the Resource.customTable parameter to true. For example, in IPC messageQ example the SYS/BIOS configuration file resides under Ipc_xx_xx_xx/examples/DRA7XX_linux_elf/ex02_messageq/dsp1.

```
/* Override the default resource table with my own */
var Resource = xdc.useModule('ti.ipc.remoteproc.Resource');
Resource.customTable = true;
```

When Resource.customTable is set to true, the IPC will no longer generate a default table and the user will be able to supply their own table to the DSP or IPU codebase by using a specially-named C structure (ti_ipc_remoteproc_ResourceTable).

To add a new entry (such as physical or virtual memory translation of CMEM DDR memory) to the resource table:

1. Specify the CMEM physical address, desired virtual address, and size.

   ```
   #define DSP_CMEM_IOBUFS         0x88000000
   #define PHYS_CMEM_IOBUFS        0xA0000000
   #define DSP_CMEM_IOBUFS_SIZE    (SZ_1M * 16)
   ```

   Two-level memory address translation is supported in the remoteproc framework (IOMMU driver) on the AM572x device. The supported page sizes are 4K, 64K (L2-entries), 1M and 16M (L1-entries). Use the largest page size possible to avoid page cache misses if the DSP firmware accesses different regions that cannot be cached within the 32-entry translation lookaside buffer (TLB).

NOTE: Typically, the IPC application puts text, data, and heap sections in the CMA pool with entry TYPE_CARVOUT. Ensure the total size of these sections with the entry TYPE_CARVEOUT in the resource table is less than the CMA pool defined in the Linux device tree. For example, DSP_MEM_TEXT_SIZE in the following code snippet must be less than the size of cmem_block_mem_0.

```
{
    TYPE_CARVEOUT,
    DSP_MEM_TEXT, 0,
    DSP_MEM_TEXT_SIZE, 0, 0, "DSP_MEM_TEXT",
},
```

The following dma_alloc_coherent error will occur if the size of TYPE_CARVEOUT entry is larger than size of cmem_block_mem_0: [596.342604] omap-rproc 40800000.dsp: dma_alloc_coherent err: 134217728.

2. Increase the size of offset[X] array in struct my_resource_table {}.
3. Add a new struct fw_rsc_devmem devmemY entry in struct my_resource_table.
4. Increase the number of entries in ti_ipc_remoteproc_ResourceTable.
5. Add the actual entry in ti_ipc_remoteproc_ResourceTable.

```
{
    TYPE_DEVMEM,
    DSP_CMEM_IOBUFS, PHYS_CMEM_IOBUFS,
    DSP_CMEM_IOBUFS_SIZE, 0, 0, "DSP_CMEM_IOBUFS",
},
```

Refer to the *IPC Resource customTable* wiki page for customized resource table details.

4.4.1 UniCache on Dual Core Cortex®-M4 Subsystems

The dual core Cortex-M4 (IPU) subsystem incorporates UniCache memory with attribute MMU (AMMU) allowing for efficient memory use. SYS/BIOS supports AMMU configuration and usage through the configuration script. For more information on UniCache and AMMU, refer to *AM572x Sitara ™ Processors Silicon Revision 2.0* and ipc_3_43_01_03/examples/DRA7XX_linux_elf/ex02_messageq/ipu1/IpuAmmu.cfg as an example.

4.5 Change IPU and DSP SYS/BIOS Configuration to Reflect Resource Table Changes

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

The following code snippet is for the DSP core, but the same principle apply to the IPU as well. The virtual addresses and sizes of these sections are defined in resource table:

```
#define DSP_MEM_TEXT            0x95000000 #define DSP_MEM_DATA            0x95100000 #define
DSP_MEM_HEAP            0x95200000  #define DSP_MEM_TEXT_SIZE       SZ_1M #define
DSP_MEM_DATA_SIZE       SZ_1M #define DSP_MEM_HEAP_SIZE       (SZ_1M * 3)

    {
        TYPE_CARVEOUT,
     DSP_MEM_TEXT, 0,
     DSP_MEM_TEXT_SIZE, 0, 0, "DSP_MEM_TEXT",
    },

 {
  TYPE_CARVEOUT,
  DSP_MEM_DATA, 0,
  DSP_MEM_DATA_SIZE, 0, 0, "DSP_MEM_DATA",
 },

 {
  TYPE_CARVEOUT,
  DSP_MEM_HEAP, 0,
  DSP_MEM_HEAP_SIZE, 0, 0, "DSP_MEM_HEAP",
 },
```

Reference the following code snippet and reflect the changes in the SYS/BIOS build configuration file (config.bld at ipc_3_43_01_03/examples/DRA7XX_linux_elf/ex02_messageq/shared/).

```
var evmDRA7XX_ExtMemMapDsp = {
 EXT_CODE: {
  name: "EXT_CODE",
  base: 0x95000000,   len:  0x00100000,
  space: "code",
  access: "RWX"
 },
 EXT_DATA: {
  name: "EXT_DATA",
  base: 0x95100000,   len:  0x00100000,
  space: "data",
  access: "RW"
 },
 EXT_HEAP: {
  name: "EXT_HEAP",
  base: 0x95200000,   len:  0x00300000,
  space: "data",
  access: "RW"
 },
```

The SYS/BIOS configuration file should be updated accordingly if there are any changes of the virtual address and size of these sections in the resource table.

CMEM API Usage

A.1 CMEM Buffer Initialization

The following code snippet of the CMEM buffer initialization shows CMEM API usage.

```
#include <ti/cmem.h>

typedef struct  bufmgrDesc_s {
    UInt32 physAddr;            /* physical address */
    UInt32 *userAddr;          /* Host user space Virtual address */
    UInt32 length;              /* Length of host buffer */
} bufmgrDesc_t;

CMEM_AllocParams  alloc_params;
bufmgrDesc_t  cmem_buf_desc;

Void initCmemBufs()
{
    CMEM_AllocParams  alloc_params;
    int i;

    printf("--->App_Create: CMEM_allocPhys and map\n");
    alloc_params.flags = CMEM_NONCACHED;     alloc_params.type = CMEM_POOL;
alloc_params.alignment = 0;     if(CMEM_init() != 0)
        printf("--->App_Create: ERROR: CMEM_init()\n");

    cmem_buf_desc.physAddr = CMEM_allocPhys(256, &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 = 256;

    cmem_buf_desc.userAddr = CMEM_map((UInt32)cmem_buf_desc.physAddr, cmem_buf_desc.length);
    if(cmem_buf_desc.userAddr == NULL)
        printf("--->App_Create: ERROR: CMEM_map()\n");
    }
```

A.2 Buffer Address to DSP

The following code is a snippet of sending the buffer address to the DSP.

```
/* allocate message */
        msg = (App_Msg *)MessageQ_alloc(Module.heapId, Module.msgSize);
        if (msg == NULL) {
            status = -1;
            goto leave;
        }

        /* set the return address in the message header */
        MessageQ_setReplyQueue(Module.hostQue, (MessageQ_Msg)msg);

        /* fill in message payload */
        msg->cmd = App_CMD_NOP;
        msg->physAddr = cmem_buf_desc.physAddr;

        /* send message */
        MessageQ_put(Module.slaveQue, (MessageQ_Msg)msg);
```

A.3 DSP Receiving Buffer Address

The following code snippet shows the DSP receiving the physical address and accessing the buffer.

```
/* wait for inbound message */
        status = MessageQ_get(Module.slaveQue, (MessageQ_Msg *)&msg,
            MessageQ_FOREVER);

        if (status < 0) {
            goto leave;
        }

        Log_print1(Diags_INFO, "Server_exec: physAddr=0x%x", msg->physAddr);
        ret = Resource_physToVirt(msg->physAddr, &va);
        if(ret == Resource_S_SUCCESS) {
            for (i=0; i<4; i++)
               Log_print1(Diags_INFO, "Server_exec: *VA=0x%x", *((UInt32 *)va + i));
        }
```

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		