Application Report

– Dec 2016

Memory Map of Vision SDK

1. Introduction

The document discusses about the Vision SDK (VSDK) memory map implementation on TDA2xx, TDA2Ex and TDA3xx. TI Vision SDK is a multi-processor multi-channel software development platform, which enables the easy integration of new vision applications using different heterogeneous CPUs of TI ADAS SoCs. VISION SDK allows different usecases for different platforms and hence a generic memory map might not be sufficient for all users. This document gives insight on different sections of the memory map that a user can change for their usecases.

It is expected that the user has gone through the below documentations to understand the hardware and software architecture/partitioning.

- TDAxx SoC architecture (TRM)
- Vision SDK (Links & chain) Framework UserGuides
  - VisionSDK_SW_Architecture_Details.pdf
  - VisionSDK_SW_Architecture_Overview.pdf
- VSDK Developer guides
  - VisionSDK_DevelopmentGuide.pdf
  - VisionSDK_Linux_DevelopmentGuide.pdf

2. Memory Sections in VISION SDK Memory Map

VISION SDK has multiple memory map configurations supported based on the usage scenarios and available total memory on various platforms. The total system memory is divided into various sub-sections/processors. The broad classification of the sections is listed below:

Shared regions: SRs are different memory partitions that are shared across processors

- SR0: Shared Region 0. This shared region is used to allocate memory for data structures needed for inter processor communication. This shared region is not cached on any of the processor cores.
- SR1_FRAME_BUFFER_MEM: This memory region is used for allocating data buffers for capturing video data, scaling, Alg processing & displaying video frames. Accessible & cached from all cores.
- SR2_MEM: Used only with VSDK Linux reserving memory for CMEM allocations (contiguous memory for Linux)
- SYSTEM_IPC_SHM_MEM: Non-Cached Memory section for keeping IPC link data structures

Log Mem:

- REMOTE_LOG_MEM: Non-Cached Memory section reserved and accessible from all processor cores to dump the debug/profile print messages. Each processor core uses VPS_printf() to dump the status/debug messages on this memory region. Remote Debug Client running on master cores (IPU1-0 for Bios and A15 in case of Linux) reads this memory region and prints the content on the (UART) console.
- LINK_STATS_MEM: Non Cached Memory section reserved and accessible from all processor cores to dump the Link statistics
- TRACE_BUF: Remote proc logs, non-Cached, used only with VSDK Linux

DDR Code/data:

- CODE_MEM: Partition for code section of each core's executable binary
- DATA_MEM: Partition for data section of each core's executable binary

Internal memory:

- OCMC_RAM: Internal Memory section accessible from all processor cores
- L2_SRAM: DSP L2 Internal Memory section, only accessible from DSP core

Linux mem:

- Memory partitions given to Linux kernel memory manager

Others:

- HDVPSS_DESC_MEM: Memory section used by BSP/STW. It is used by these driver for its internal descriptor data structures.

3. VSDK Memory Map Table

Refer the .xs file under \vision_sdk\build\tdaxxx\ for the complete memory map configuration,
For example, \vision_sdk\build\tda2xx\mem_segment_definition_512mb_bios.xs defines the memory map for TDA2x with 512MB DDR.
Also refer the generated map files under
\vision_sdk\binaries\tdaxxx_evm_xxx_all\vision_sdk\bin\tdaxxx-evm\
For ample, vision_sdk_ipu1_0_release.xem4.map,
Memory sections are arranged as below

Name	Origin	Length	Used	Unused	Attr
L2_ROM	00000000	00004000	000005ec	00003a14	RWIX
L2_RAM	20000000	00010000	00000000	00010000	RWIX
OCMC_RAM1	40300000	00080000	00080000	00000000	RWIX
OCMC_RAM2	40400000	00100000	00000000	00100000	RWIX
OCMC_RAM3	40500000	00100000	00000000	00100000	RWIX
DSP1_L2_SRAM	40800000	00048000	00000000	00048000	RWIX
DSP2_L2_SRAM	41000000	00048000	00000000	00048000	RWIX
NDK_MEM	85800000	00200000	00000000	00200000	RWIX
SR1_FRAME_BUFFER_MEM	85a00000	0fa00000	0fa00000	00000000	RWIX
DSP1_CODE_MEM	99000000	00200000	00000000	00200000	RWIX
DSP1_DATA_MEM	99200000	01800000	00000000	01800000	RWIX
IPU1_1_CODE_MEM	9d000000	00200000	00000000	00200000	RWIX
IPU1_1_DATA_MEM	9d200000	00200000	00000000	00200000	RWIX
IPU1_1_BSS_MEM	9d400000	00200000	00000000	00200000	RWIX
IPU1_0_CODE_MEM	9d600000	00600000	004fa45d	00105ba3	RWIX
IPU1_0_DATA_MEM	9dc00000	00400000	00000000	00400000	RWIX
IPU1_0_BSS_MEM	9e000000	00d00000	00bf5cdc	0010a324	RWIX
DSP2_CODE_MEM	9f000000	00200000	00000000	00200000	RWIX
DSP2_DATA_MEM	9f200000	00a00000	00000000	00a00000	RWIX
SR0	a0100000	00100000	00100000	00000000	RWIX
REMOTE_LOG_MEM	a0200000	00040000	00027890	00018770	RWIX
LINK_STATS_MEM	a0240000	00080000	00020d9c	0005f264	RWIX
SYSTEM_IPC_SHM_MEM	a02c0000	00040000	00035a60	0000a5a0	RWIX
HDVPSS_DESC_MEM	a0300000	00100000	000b07c0	0004f840	RWIX
TRACE_BUF	a0400000	00060000	00008000	00058000	RWIX
EXC_DATA	a0460000	00010000	00000000	00010000	RWIX
PM_DATA	a0470000	00080000	00000000	00080000	RWIX
EVE1_VECS_MEM	a2000000	00080000	00000000	00080000	RWIX
EVE1_CODE_MEM	a2080000	00200000	00000000	00200000	RWIX
EVE1_DATA_MEM	a2280000	00d80000	00000000	00d80000	RWIX
EVE2_VECS_MEM	a3000000	00080000	00000000	00080000	RWIX

EVE2_CODE_MEM	a3080000	00200000	00000000	00200000	RWIX
EVE2_DATA_MEM	a3280000	00d80000	00000000	00d80000	RWIX
EVE3_VECS_MEM	a4000000	00080000	00000000	00080000	RWIX
EVE3_CODE_MEM	a4080000	00200000	00000000	00200000	RWIX
EVE3_DATA_MEM	a4280000	00d80000	00000000	00d80000	RWIX
EVE4_VECS_MEM	a5000000	00080000	00000000	00080000	RWIX
EVE4_CODE_MEM	a5080000	00200000	00000000	00200000	RWIX
EVE4_DATA_MEM	a5280000	00d80000	00000000	00d80000	RWIX

Refer vision_sdk\include\link_api\systemLink_common_if.h for the available Memory Heaps, Here is a brief list; detailed description is in systemLink_common_if.h file

SYSTEM_HEAPID_DDR_NON_CACHED_SR0: Heap ID of heap in DDR, This is non-cached memory
SYSTEM_HEAPID_DDR_CACHED_SR1: Heap ID of heap in DDR, This is cached memory
SYSTEM_HEAPID_OCMC_SR2: Heap ID of heap in OCMC
SYSTEM_HEAPID_RESERVED1: Heap ID of heap in DDR, This is cached memory
SYSTEM_HEAPID_RESERVED2: Heap ID of heap in L2 Memory, Internal memory

4. Software and Hardware Constraints to Consider For Deciding Memory Map

4.1 Hardware Constraints

- AMMU in IPU1/2 handles large memory segments of size 512MB/32MB only and there can be 4 such segments.
- For EVE we have only 32 TLB entries to map the memory, one TLB can map a max of 16MB

4.2 Software Constraints

- For VSDK Linux, the first 64MB from 0x8000 0000 is reserved for Linux Kernel
- Frame Buffer Shared Region (SR1) is mapped on A15 Linux. But if user wants Linux side app/links to allocate memory/buffers from any other heap, then user need to mmap the physical address in the application code to map those memory/buffers on A15.

Important things to pay attention to, while porting the map file are:

- DDR, OCMC & DSP/EVE SRAM sizes
- Core specific code/data/vecs sizes
- Size of the shared frame buffer pool

DDR is divided into 2 sections: cached and non-cached.

- The cached part is used mainly for frame buffer (SR1) and core specific code/data sections.
- The non-cached part is used mainly for Vision SDK log buffers, IPC shared data structure (SR0) HDVPSS descriptors etc.

5. Memory Allocation

This section describes the different methods by which memory is allocated in the Vision SDK framework. The Vision SDK framework also supports static memory allocation.

Memory in Vision SDK framework is allocated for the following purposes

Purpose	Region in memory used for allocation	Type of allocation (Dynamic, Static)
External Buffer memory for storing algorithms results and/or HW engine results	SR1_FRAME_BUFFER_MEM	Dynamic (heap based) and/or Static
Internal Buffer memory for storing algorithms results and/or HW engine results	OCMC_RAM	Dynamic (heap based) and/or Static
Notify Shared region – ONLY used during Notify setup (IPC_Start()), not used later	SR0	Dynamic (heap based)
Temporary scratch memory in internal memory for algorithms results	DMEM in EVE L2SRAM in DSP	Dynamic (non-heap, linear allocation)
Shared memory for remote core print logs	REMOTE_LOG_MEM	Static
Shared memory for link statistics	LINK_STATS_MEM	Static
Shared memory for inter processor communication	SYSTEM_IPC_SHM_MEM	Static
VPDMA descriptor memory for VIP, VPE HW engines	HDVPSS_DESC_MEM	Static
CPU specific memory for BIOS objects like semaphores, tasks, interrupts, clocks	CPU specific data section	Static

The subsequent sections provide more details on each type of memory allocation
In the below description,
<soc> = tda2xx, tda2ex, tda3xx, tda2x-entry
<ddr_size> = 128mb, 256mb, 512mb, 1024mb
<os_type> = Bios, Linux

5.1 External Buffer Memory Allocation

Location where memory map is specified

- The memory region used for external buffer memory allocation is specified via the below file
  - File: vision_sdk\build\<soc>\mem_segment_definition_<ddr_size>_<os_type>.xs
  - Variable SR1_FRAME_BUFFER_SIZE
- The heap from which memory is allocated is defined in file
  - FILE: vision_sdk\src\utils_common\src\utils_mem_ipu1_0.c
    - #pragma DATA_SECTION(gUtils_memHeapDDR, ".bss:heapMemDDR")
  - FILE: vision_sdk\src\utils_common\include\utils_mem_cfg.h
    - #define UTILS_MEM_HEAP_DDR_CACHED_SIZE
  - This heap is placed in "SR1_FRAME_BUFFER" section via the IPU1-0 cfg file
    - FILE: vision_sdk\src\main_app\<soc>\ipu1_0\Ipu1_0.cfg

- Program.sectMap[".bss:heapMemDDR"] = "SR1_FRAME_BUFFER_MEM";

- The heap is defined only on IPU1-0 CPU; all other CPUs sends message to IPU1-0 to allocate memory. This is done internally inside the Utils_memAlloc APIs.

API to allocate and free memory

- Below APIs are used to allocate and free memory
  - FILE: vision_sdk\src\utils_common\include\utils_mem.h
  - API:
    - Utils_memAlloc() with heapId as SYSTEM_HEAPID_DDR_CACHED_SR1
    - Utils_memFree() with heapId as SYSTEM_HEAPID_DDR_CACHED_SR1
    - Utils_memGetHeapStats() with heapId as SYSTEM_HEAPID_DDR_CACHED_SR1
- Other APIs from this file are not recommended to be used by users and are used internally by the framework

Using static memory allocation

- When a system wants to use static memory allocation and avoid the heap, it should set the size of this heap segment as 0 by modifying the #define in utils_mem_cfg.h file
- Define static memory objects (arrays, data structures) in IPU1-0 use-case file. Make sure the objects are placed in data section ".bss:heapMemDDR" via #pragma
- The links which support static memory allocation allow passing of memory region pointers from use-case file via System_LinkMemAllocInfo data structure
- When creating a link from a use-case, user should now pass memory pointer allocated statically from use-case file. This prevents the link for allocating memory internally. Thus dynamic memory allocation is avoided
  - See capture link "captureLink.h" for example
  - See use-case "vision_sdk\examples\tda2xx\src\usecases\vip_single_cam_view" for sample usage of passing user memory pointer to a link
  - NOTE: In the use-case the memory allocation is still done using Utils_memAlloc APIs. In a fully static memory system, this API won't be used by the user.
- The links assert if the memory segment size passed to it is smaller than what is needed. In this case, it also reports the size required by the link.
- When creating user specific AlgPlugins same mechanism should be used, i.e algorithm plugin should take memory pointer passed from use-case file rather than allocating memory internally. See "Capture" link for example

5.2 Internal Buffer Memory Allocation

Location where memory is specified

- The memory region used for buffer memory allocation is specified via the below file
  - File: vision_sdk\build\<soc>\mem_segment_definition_<ddr_size>_<os_type>.xs
  - Variable OCMC1_SIZE
- The heap from which memory is allocated is defined in file
  - FILE: vision_sdk\src\utils_common\src\utils_mem_ipu1_0.c

        - #pragma DATA_SECTION(gUtils_memHeapOCMC, ".bss:heapMemOCMC")
    - FILE: vision_sdk\src\utils_common\include\utils_mem_cfg.h
        - #define UTILS_MEM_HEAP_OCMC_SIZE
    - This heap is placed in "OCMC" section via the IPU1-0 cfg file
        - FILE: vision_sdk\src\main_app\<soc>\ipu1_0\Ipu1_0.cfg
            - Program.sectMap[".bss:heapMemOCMC"] = "OCMC_RAM";
- The heap is defined only on IPU1-0 CPU; all other CPUs send a command to IPU1-0 to allocate memory. This is done internally inside the Utils_memAlloc APIs.

API to allocate and free memory

- Below APIs are used to allocate and free memory
    - FILE: vision_sdk\src\utils_common\include\utils_mem.h
    - API:
        - Utils_memAlloc() with heapId as SYSTEM_HEAPID_OCMC_SR2
        - Utils_memFree() with heapId as SYSTEM_HEAPID_OCMC_SR2
        - Utils_memGetHeapStats() with heapId as SYSTEM_HEAPID_OCMC_SR2
- Other APIs from this file are not recommended to be used by users and are used internally by the framework

5.3 Static Memory sections Allocation

- The memory region used for these sections are specified via the below file
    - File: vision_sdk\build\<soc>\mem_segment_definition_<ddr_size>_<os_type>.xs
    - Variable "REMOTE_LOG_SIZE" for Remote Log memory
    - Variable "SYSTEM_IPC_SHM_SIZE" for inter-processor communication
    - Variable "LINK_STATS_SIZE" for Link Statistics
    - Variable "HDVPSS_DESC_SIZE" for VPDMA descriptors

6. Memory map of the application

Memory map of the entire usecase is governed by following artifacts.

1. DDR_MEM variable in Rules.make (older versions) or
   \vision_sdk\configs\tdaxxx_evm_<OS>_all\cfg.mk (for VSDK version 2.11 and above)

List of VSDK files need to be reviewed & modified

1. */vision_sdk/build/tdaxxx/mem_segment_definition_<DDR_MEM>_<OS>.xs*
2. */vision_sdk/src/utils_common/include/utils_mem_cfg.h*
3. */vision_sdk/src\main_app\tda2xx\ipu1_0 \ Ammu1.cfg or Ammu1_linux.cfg (if you modify the IPU1 memory map)*
4. */vision_sdk/src\main_app\tda2xx\ipu2\ Ammu2.cfg or Ammu2_linux.cfg (if you modify the IPU2 memory map)*
5. */vision_sdk/src/main_app/tda2xx/eve_common/tlb_config_eve_common.c (if you modify any EVE1-4 memory map)*
6. */vision_sdk/include/link_api/system_vring_config.h (only for Linux/HLOS build)*
7. */vision_sdk/hlos/src/osa/include/osa_mem_map.h (only for Linux/HLOS build)*

8. */vision_sdk/src/links_common/system/system_rsc_table_ipu.h (only for Linux/HLOS build, if resource table modification required)*
9. */vision_sdk/src/links_common/system/system_rsc_table_dsp.h (only for Linux/HLOS build, if resource table modification required)*

List of Linux Kernel files need to be modified

10. */ti_components/os_tools/kernel/omap/arch/arm/boot/dts/dra7-evm-infoadas.dts (For TDA2x)*
11. */ti_components/os_tools/kernel/omap/arch/arm/boot/dts/dra7-evm.dts (For TDA2x)*
12. */ti_components/os_tools/kernel/omap/arch/arm/boot/dts/dra72-evm-infoadas.dts (For TDA2Ex)*
13. */ti_components/os_tools/kernel/omap/arch/arm/boot/dts/dra72-evm.dts (For TDA2Ex)*

#1 – DDR_MEM is an environment variable that tells build system which .xs is to be picked up for the final executable.

#2 – The .xs file overrides default implementation for the platform defined by xdc.runtime. This file can be modified to increase/decrease size of a section or add/remove sections from the memory map. For Linux/HLOS, Linux enables L2MMU for each core, so all the addresses mentioned in the .xs file are slave virtual addresses.

#3 – The .dts file is used to reserve memory from Linux, this is a platform specific file. This ensures Linux and bios side don't overwrite into each other. Typically the bios side needs some memory sections and rest all can be given to Linux. Essentially this creates a few holes in Linux memory that is later mapped to user space at the application startup time.

6.1 Adding a new section to memory map

While adding a new section in the memory map of ipu/dsp/eve/A15, following things needs to be taken care of:

1. Add a new section in appropriate .xs file by defining NEW_SECTION_SIZE, NEW_SECTION_ADDR & NEW_SECTION_MEM (just follow the convention used in .xs file)
2. Its advised to remove or reduce some unwanted sections to free-up the memory required for the new section
3. Make sure the total memory should not exceeds the total available physical memory
4. Make sure the new section doesn't overlap with any other sections.
5. If you add any new memory section and, the data/code corresponding to that can be placed into the section by adding Program.sectMap in the appropriate \vision_sdk\src\main_app\tdaxxx\<cpu>**<cpu>.cfg** file.
6. **In case of Linux**, it should lie within the hole of memory declared in .dts file in kernel using /memreserve
7. If needed, /memreserve can be used to increase the size of the hole accommodate new section's memory requirement.
8. If this newly added section has to be mapped into L2MMU of ipu/dsp by Linux and hence it needs to be added in the resource table i.e. in system_rsc_table_ipu.h or system_rsc_table_dsp.h accordingly.
9. If this section is going to be accessed from Linux user space or kernel space, this mapping needs to be taken care by the application or through OSA_mem module in vision_sdk

10. If you are changing base addresses and sizes for IPU's, DSP's carve-out sections (code/data) and if you plan to change CMA address in linux kernel (.dts) please ensure you also make this changes to vision_sdk\include\system_vring_config.h.
11. Refer section "6.4: How To – Modify Linux/Bios VSDK Memory Map" for more details

6.2 Changing size of a section in the memory map

While changing the size of the section in the memory map from ipu/dsp/eve/A15, following things needs to be taken care of:

1. Do changes in respective .xs file for the section sizes
2. Its advised to reduce some unwanted sections to free-up the memory required for new size (increase)
3. Make sure the total memory should not exceeds the total available physical memory
4. Make sure the new section doesn't overlap with any other sections.
5. **In case of Linux**, it should lie within the hole of memory declared in .dts file in kernel using /memreserve
6. If needed, /memreserve can be used to increase the size of the hole accommodate new section's memory requirement.
7. As you are modifying existing section, no need to change resource table mappings, the updated value will be picked up in resource table in the build process.
8. If you are changing base addresses and sizes for IPU's, DSP's carve-out sections (code/data) and if you plan to change CMA address in linux kernel (.dts) please ensure you also make this changes to vision_sdk\include\system_vring_config.h.
9. Refer section "6.4: How To – Modify Linux/Bios VSDK Memory Map" for more details

6.3 How To - Add a new memory map

In general, if you are planning to have your own memory map for the application, you can follow these steps

1. Evaluate memory requirements of the sections e.g. (Is 256 MB SR1 sufficient or you need more or less?)
2. Add appropriate .xs file under $INSTALL_DIR/*vision_sdk/build/tdaxxx/,* for example mem_segment_definition_512mb.xs
2. Modify DDR_MEM_XXXX, for example DDR_MEM_512M variable in Rules.make (older versions) or \vision_sdk\configs\tdaxxx_evm_<OS>_all\cfg.mk (for VSDK version 2.11 and above)
3. Modify appropriate platform ISI build files to pick the correct memory map (.xs) file, for example, refer below files for TDA3x,
    a. \vision_sdk\build\tda3xx\config_arp32.bld
    b. \vision_sdk\build\tda3xx\config_c66.bld
    c. \vision_sdk\build\tda3xx\config_m4.bld
    d. Etc.
4. Now follow the section "6.4: How To – Modify Linux/Bios VSDK Memory Map" for more details and how to modify all necessary VSDK and Linux kernel files

6.4 How To – Modify Linux/Bios VSDK Memory Map

The memory map of complete VISION SDK is controlled in

\vision_sdk\build\tdaxxx\config_<ISI>.bld

The mem_segment_definition (for example - mem_segment_definition_1024mb_linux.xs) file is included in this build configuration file, size of each sections are reconfigured in .xs file. For example, DSP code size and DSP data size section can be changed by modifying the following entries.

```
DSP1_CODE_SIZE              = 3*MB;
DSP1_DATA_SIZE              = 13*MB;
```

The base addresses of each section are incremented based on the base address of previous section and the size of the previous section. For example, if sections are created in the numerical order, base address of Section 2 is calculated as below:
<Start Addr of Sect 2> = <Start Addr of Sec 1> + <Size of Sect 1>

To modify the memory map, user needs to consider the following:

- Refer to the hardware limitations and software limitations in section 4:
  - We assume a one-to-one mapping of AMMU virtual address to physical address.
- The other sections like "Remote Debug", "HDVPSS Shared Memory" etc. are read directly from the build configuration file
- Changes in the Linux memory size in build configuration file has to be reflected in the boot arguments of the Linux kernel using "**mem=<SIZE>M**" entry.
- Consider the overall buffer requirement for the specific usecase before modifying the Frame Buffer or Meta data buffer or BitsBuffer section sizes.

The current default memory maps of VSDK (as per 2.12 releases) as below

- TDA2xx Bios – 512 MB
- TDA2xx Linux – 1024 MB
- TDA2Ex Bios – 512 MB
- TDA2Ex Linux – 1024 MB
- TDA3xx Bios – 512 MB
- TDA3xx Bios – 128 MB

6.4.1 Cache and MMU configurations

List of VSDK files sets the Cache and MMU configurations

<1> DSP

DSP L1 & L2 Cache configuration is in
\vision_sdk\src\main_app\tdaxxx\cfg\DSP_common.cfg,
Below the default cache size setting for L1P, L1D and L2 as 32K

```
var Cache = xdc.useModule('ti.sysbios.family.c66.Cache');
Cache.initSize.l1pSize = Cache.L1Size_32K;
Cache.initSize.l1dSize = Cache.L1Size_32K;
Cache.initSize.l2Size  = Cache.L2Size_32K;
```

Cache ON/OFF setting of DSP also in
\vision_sdk\src\main_app\tdaXxx\cfg\DSP_common.cfg

```
/* Set cache sections */
/* configure MARs, by default cache is enabled for the entire memory region */
for (var i = 0; i < Program.cpu.memoryMap.length; i++)
{
  memSegment = Program.cpu.memoryMap[i];
  Cache.setMarMeta(memSegment.base, memSegment.len, Cache.Mar_ENABLE);
}
```

```
/* set non-cached sections */
for (var i = 0; i < Program.cpu.memoryMap.length; i++)
{
  memSegment = Program.cpu.memoryMap[i];

  if ((memSegment.name == "SR0") ||
      (memSegment.name == "REMOTE_LOG_MEM") ||
      (memSegment.name == "LINK_STATS_MEM") ||
      (memSegment.name == "SYSTEM_IPC_SHM_MEM") ||
      (memSegment.name == "OPENVX_SHM_MEM"))
     {
       Cache.setMarMeta(memSegment.base, memSegment.len, Cache.Mar_DISABLE);
     }
}
```

If you plan to add any new non-cached DSP section, then add the same section in above code snippet in DSP_common.cfg.

<2> IPU

Refer below files which set the AMMU of IPU to configure MMU and Cache settings
\vision_sdk\src\main_app\tdaXxx\ipu1_0\Ammu1_bios.cfg or Ammu1_linux.cfg
\vision_sdk\src\main_app\tdaXxx\ipu2\Ammu2_bios.cfg or Ammu2_linux.cfg
Map program code/data & other cached memory into ammu (cacheable) by AMMU.largePages[1]
Map SR_0 & other non-cached data memory into ammu (non-cacheable) by AMMU.largePages[2]

Note: Only for TDA3x, IPU1 AMMU setting is done in SBL.
Refer \vision_sdk\src\main_app\tda3xx\ipu1_0\Ammu1_bios.cfg,
AMMU.configureAmmu = false;
If configureAmmu is set to false, then AMMU setting is done in SBL.

<3> EVE

Refer \vision_sdk\src\main_app\tda2xx\eve_common\tlb_config_eve_common.c for EVE memory mapping by programming the TLB registers, each TLB can map a max of 16MB contiguous region and 16MB aligned.
const UInt32 tlbMapping[EVE_TLB_NUM_ENTRIES*2U] ={….}

<4> A15 (Bios)

If A15 running Bios, then \vision_sdk\src\main_app\tda2xx\a15_0\ a15_0.cfg does the MMU & cache configurations,
Mmu.setSecondLevelDescMeta();

6.4.2 Modify default memory maps of VSDK

Please revisit & modify the list of below files (as required)

List of VSDK files

<1>

/vision_sdk/build/tdaxxx/**mem_segment_definition_<DDR_SIZE>_<OS>.xs**.
A single file that configures the entire memory map of all cores in the SoC, and the major file to be modified if you want to make changes in VSDK memory map. This file defines all

the memory sections for IPU, DSP, EVE, A15 and other heaps such as SR1, SR0 etc. Below a sample section,

```
DSP1_START_ADDR          = 0x99000000;
DSP1_CODE_SIZE           = 2*MB;
DSP1_DATA_SIZE           = 24*MB;
……
……

DSP1_CODE_ADDR           = DSP1_START_ADDR;
DSP1_DATA_ADDR           = DSP1_CODE_ADDR       + DSP1_CODE_SIZE;
……
……

function getMemSegmentDefinition_external(core)
{
   memory[index++] = ["DSP1_CODE_MEM", {
         comment : "DSP1_CODE_MEM",
         name    : "DSP1_CODE_MEM",
         base    : DSP1_CODE_ADDR,
         len     : DSP1_CODE_SIZE
      }];

……
……
}
```

If you add any new memory section and, the data/code corresponding to that can be placed into the section by adding **Program.sectMap[]** in the appropriate \vision_sdk\src\main_app\tdaxxx\<cpu>**<cpu>.cfg** file.
For example, \vision_sdk\src\main_app\tda2xx\dsp1\Dsp1.cfg is the file for DSP1 of TDA2x, and below code place "bss:extMemNonCache:ipcShm" into the memory section "SYSTEM_IPC_SHM_MEM"

```
Program.sectMap[".bss:extMemNonCache:ipcShm"] = "SYSTEM_IPC_SHM_MEM";
Program.sectMap[".bss:extMemNonCache:linkStats"] = "LINK_STATS_MEM";
```

Same apply for all cores like IPU1-0, IPU1-1, IPU2, A15, DSP1-2 and EVE1-4 of TDA2x, TDA2Ex and TDA3x.

The .xs file to look at depends on SoC, A15 OS and DDR memory config selected; here is a list of default memory map of VSDK 2.12.0.0

SoC	A15 OS	DDR config	.xs file
TDA2XX_BUILD	Bios	TDA2XX_512MB_DDR	vision_sdk\build\tda2xx\mem_segment_definition_512mb_bios.xs
TDA2XX_BUILD	Linux	TDA2XX_1024MB_DDR	vision_sdk\build\tda2xx\mem_segment_definition_1024mb_linux.xs
TDA3XX_BUILD	NA	TDA3XX_128MB_DDR	vision_sdk\build\tda3xx\mem_segment_definition_128mb.xs
TDA3XX_BUILD	NA	TDA3XX_512MB_DDR	vision_sdk\build\tda3xx\mem_segment_definition_512mb.xs
TDA2EX_BUILD	Bios	TDA2EX_512MB_DDR	vision_sdk\build\tda2ex\mem_segment_definition_512mb_bios.xs
TDA2EX_BUILD	Linux	TDA2EX_1024MB_DDR	vision_sdk\build\tda2ex\mem_segment_definition_1024mb_linux.xs

Modify "start address", "size" or even add/remove a memory section to change the memory map as per your requirement.

<2>
\vision_sdk\src\utils_common\include**utils_mem_cfg.h**

This file defines the size of major memory heaps, and these sizes need to be in sync with above .xs file. The memory allocation utility/API refers this file for the heap size. Any modification of these heap size to be updated in both **utils_mem_cfg.h & mem_segment_definition_<DDR_SIZE>_<OS>.xs** file.

#define UTILS_MEM_HEAP_L2_SIZE (224*1024) – DSP internal memory (SRAM)
#define UTILS_MEM_HEAP_L2_SIZE (24*1024) - EVE internal memory (SRAM)
#define UTILS_MEM_HEAP_OCMC_SIZE (512*1024) – Shared OCMC internal memory
#define UTILS_MEM_HEAP_DDR_CACHED_SIZE (256*1024*1024) – Shared cached DDR heap memory.

<3>
\vision_sdk\src\main_app\tda2xx\ipu1_0**Ammu1_bios.cfg** or **Ammu1_linux.cfg** (if you modify the IPU1 memory map)
\vision_sdk\src\main_app\tda2xx\ipu2**Ammu2_bios.cfg** or **Ammu2_linux.cfg** (if you modify the IPU2 memory map)
This file set IPU subsystem (core 0 and core 1) AMMU and Cache configurations. IPU can access only the memory sections mapped via AMMU. A sinle AMMU sets the memory map of both cores (core 0 & core 1) of an IPU subsystem.

```
function init()
{
   …….
   …….

    Map program code/data & other cached memory into ammu (cacheable) by
    AMMU.largePages[1]
    Map SR_0 & other non-cached data memory into ammu (non-cacheable) by
    AMMU.largePages[2]
   …….
   …….
}
```

<4>
/vision_sdk/src/main_app/tda2xx/eve_common/**tlb_config_eve_common.c** (if you modify any EVE1-4 memory map)
This file implements common MMU configuration for all EVE as per Vision SDK requirements
There are only 32 TLB entries in EVE, Each TBL entry can maps a max of 16MB, and need 16MB alignment,

```
const UInt32 tlbMapping[EVE_TLB_NUM_ENTRIES*2U] =
{
   …….
   …….
   0x84000000U, 0x84000000U, /* 08 - For SR1 */
   0x85000000U, 0x85000000U, /* 09 - For SR1 */
   0x86000000U, 0x86000000U, /* 10 - For SR1 */
   0x87000000U, 0x87000000U, /* 11 - For SR1 */
   …….
```

```
    ……..
}
eveCommonMmuConfig();
```

<5>
/vision_sdk/include/link_api/**system_vring_config.h** (used only in A15 Linux Build)

Vring virtual addresses (For Start address) of IPU & DSP are used by IPC, if it is changed in .XS file, same need to be updated in this system_vring_config.h also.

```
#ifdef BUILD_M4_0
#define IPU_PHYS_MEM_IPC_VRING      0x9d000000
#endif

#ifdef BUILD_DSP_1
#define DSP_PHYS_MEM_IPC_VRING      0x99000000
#endif

#ifdef BUILD_DSP_2
#define DSP_PHYS_MEM_IPC_VRING      0x9f000000
#endif

#ifdef BUILD_M4_2
#define IPU_PHYS_MEM_IPC_VRING      0x95800000
#endif
```

<6>
/vision_sdk/<hlos>/src/osa/include/**osa_mem_map.h** (used only in A15 Linux Build)

This is an auto generated file from [gen_system_mem_map.xs], please check and confirm the entries in osa_mem_map.h is matching with .XS file or the MAP file

```
#define SR0_ADDR    0xa0100000
#define SR0_SIZE    0x100000

#define SYSTEM_IPC_SHM_MEM_ADDR    0xa02c0000
#define SYSTEM_IPC_SHM_MEM_SIZE    0x80000

#define REMOTE_LOG_MEM_ADDR       0xa0200000
#define REMOTE_LOG_MEM_SIZE       0x40000

#define SR1_FRAME_BUFFER_MEM_ADDR 0x84203000
#define SR1_FRAME_BUFFER_MEM_SIZE 0xfa00000
```

<7>
If a newly added section has to be mapped into L2MMU of ipu/dsp by Linux and hence it needs to be added in the resource table i.e. in system_rsc_table_ipu.h or system_rsc_table_dsp.h accordingly.

/vision_sdk/src/links_common/system/**system_rsc_table_ipu.h** (modify if required, used only in A15 Linux Build)
struct my_resource_table {…}
struct my_resource_table ti_ipc_remoteproc_ResourceTable = {…}

system_rsc_table_ipu.h define the resource table entries for all IPU cores. This will be incorporated into corresponding base images, and used by the remoteproc on the host-side to allocated/reserve resources.

/vision_sdk/src/links_common/system/**system_rsc_table_dsp.h** (modify if required, used only in A15 Linux Build)
struct my_resource_table {…}
struct my_resource_table ti_ipc_remoteproc_ResourceTable = {…}

system_rsc_table_dsp.h define the resource table entries for all DSP cores. This will be incorporated into corresponding base images, and used by the remoteproc on the host-side to allocated/reserve resources.

List of Linux Kernel files

<1>
/ti_components/os_tools/kernel/omap/arch/arm/boot/dts/**dra7-evm-infoadas.dts** (for TDA2x)
Modify the start address of IPU1, IPU2, DSP1, DSP2 or CMEM, if any of this is changed in the memory map,

```
/* Update the CMA regions for Vision SDK binaries */
&ipu2_cma_pool {
    reg = <0x94000000 0x5000000>;
};

&dsp1_cma_pool {
    reg = <0x99000000 0x4000000>;
};

&ipu1_cma_pool {
    reg = <0x9d000000 0x2000000>;
};

&dsp2_cma_pool {
    reg = <0x9f000000 0x1000000>;
};

&reserved_mem  {
    cmem_pool: cmem@A6000000 {
            compatible = "shared-dma-pool";
            reg = <0xA8000000 0x4000000>;
            no-map;
            status = "okay";
    };
};
```

<2>
/ti_components/os_tools/kernel/omap/arch/arm/boot/dts/**dra7-evm.dts** (for TDA2x)

This file reserves the memory for SR1, SR0 and EVE data/code memory sections and the size. If any changes in the memory map for any of these sections, then update the below code,

```
/memreserve/ 0x84000000 0x10000000;
/memreserve/ 0xA2000000 0x4000000;
/memreserve/ 0xA0000000 0x2000000;
```

Modify the start address of IPU1, IPU2, DSP1 or DSP2, if any of this is changed in the memory map,

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

<3>
/ti_components/os_tools/kernel/omap/arch/arm/boot/dts/**dra72-evm-infoadas.dts** (for TDA2Ex)
Modify the start address of IPU1, IPU2, DSP1 or CMEM, if any of this is changed in the memory map,

```
/* Update the CMA regions for Vision SDK binaries */
&ipu2_cma_pool {
    reg = <0x94000000 0x5000000>;
};

&dsp1_cma_pool {
    reg = <0x99000000 0x4000000>;
};

&ipu1_cma_pool {
    reg = <0x9d000000 0x2000000>;
};

&reserved_mem  {
    cmem_pool: cmem@A6000000 {
            compatible = "shared-dma-pool";
            reg = <0xA8000000 0x2000000>;
            no-map;
            status = "okay";
    };
};
```

<4>
/ti_components/os_tools/kernel/omap/arch/arm/boot/dts/**dra72-evm.dts** (for TDA2Ex)

This file reserves the memory for SR1, SR0 memory sections and the size. If any changes in the memory map for any of these sections, then update the below code,

```
/memreserve/ 0x84000000 0x10000000;
/memreserve/ 0xA0000000 0x2000000;
```

Modify the start address of IPU1, IPU2 or DSP1, if any of this is changed in the memory map,

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

Clean and Rebuild Kernel & VSDK.

Note1: If SR1 or IPU1-2 memory needs to be increased to very high value, then Move DSP1-2 or EVE1-4 out of 0xA000 0000 address space. This will avoid the need of any IPU AMMU reconfiguration.

Note2: To build SBL, Build appropriate secondary boot loader as per your memory configuration. Refer file \vision_sdk\build\makerules\build_sbl.mk for all valid configurations. For examples, EMIFMODE = DUAL_EMIF_1GB_512MB (default) or DUAL_EMIF_2X512MB or SINGLE_EMIF_256MB

Note3: In case of A15 Linux, You also need to change DMM configuration in Uboot to set the DDR configuration, i.e., the EMIF and LISA map configuration as per custom board or memory map.

7. Revision History

Version #	Date	Author Name	Revision History
0.1	26/12/2016	Shiju S	First draft