Application Report
SPRAC08A–December 2015–Revised January 2016

Modifying Memory Usage for IPUMM Applications Using IPC 3.x for DRA7x

Buddy Liong

ABSTRACT

The default Image Processing Unit (IPU) image for DRA7x IPU provides numerous capabilities for a rich multimedia experience. However, not all customers will want to use all of the capabilities, or may wish to add new capabilities. If not all of the capabilities are used, then the memory usage can be reduced. Similarly, if new capabilities are added, the memory usage can be increased.

This document provides the procedure for modifying the memory usage of the IPU in order to increase or decrease the memory usage.

NOTE: All programming models and use cases presented in this document are provided for educative purposes only and may differ from or be optimized for other applications.

All DRA7x peripheral devices presented in this document are provided for illustration purposes and may be different from those in your system.

Contents

List of Tables

1 Introduction

The default IPU Multimedia (IPUMM) image provides many features and algorithms including: video decode and video encode processing algorithms. The default memory usage is configured to account for all of these. In some user configurations, not all of the features and algorithms may be desired or used, in which case, certain sections can be removed or reduced. In other cases, new features or algorithms can be added, in which case, sections may need to be increased. Adding, removing, increasing or decreasing memory usage requires modification of a few files in the IPUMM that is running on the Cortex®-M4 IPU, and potentially the QNX board support package (BSP) build file (used to generate QNX startup) as specified in Section 3. For Linux® or the Android™ HLOS platform, the section that talks about QNX BSP IFS should be ignored.

This document details how to modify the memory usage by adding, removing, increasing, or decreasing memory segments used by the IPC and Multimedia image.

NOTE: This document assumes that you are familiar with build procedures for QNX BSP IFS images, and the QNX 6Q6x.x.x release package from TI. For QNX BSP IFS build procedures, contact QNX support. For building IPC and IPUMM, see the QNX Distributed Codec Engine (DCE) release notes document and the IPUMM build instructions document located in the QNX 6Q6x.x.x release package.

2 Default Memory Segments

Table 1 shows an example of the default memory usage for the IPU image. A description concerning the purpose of each segment is provided.

NOTE: These values are just an example. The actual values for any particular release can be found by looking at the configuration files mentioned in Section 3.

Table 1. Default Memory Segments

Name	Base Address	Size	Description
IPU_MEM_TEXT:L2_ROM	0x00000000	0x00004000	IPU Boot Code
IPU_MEM_TEXT:EXT_CODE	0x00004000	0x005FC000	Remote core IPC and Multimedia Code Section
IPU_MEM_DATA:EXT_DATA	0x80000000	0x00200000	IPC and Multimedia Data Section
IPU_MEM_DATA:EXT_HEAP	0x80200000	0x02900000	Multimedia Heap Section
IPU_MEM_IPC_DATA:TRACE_BUF	0x9F000000	0x00060000	Remote Core Traces
IPU_MEM_IPC_DATA:EXC_DATA	0x9F060000	0x00010000	Remote Core Exception Info (used in case of crash)
IPU_MEM_IPC_DATA:PM_DATA	0x9F070000	0x00020000	Remote Core PM Data
IPU_MEM_IPC_DATA	0x9F090000	0x00070000	Extra memory data bandwidth for IPC on IPU. IPU_MEM_IPC_DATA is 1MB
IPU_MEM_IPC_VRING	0x60000000	0x00100000	IPC Communication VirtQueues and Buffers
IPU_MEM_IOBUFS	0x90000000	0x05A00000	Shared Memory IO Bufs. Used in certain use-cases for non-Tiler IO buffers, such as shmemallocator in QNX.

3 Memory Segment Configuration Files

There are four files that are used to configure all the memory used by the IPU image:

- QNX BSP IFS build file to define the static memory carve out:
  - <qnx_bsp_dir>/src/hardware/startup/boards/<your_board>/build
- IPUMM custom resource table where the memory resource table is defined:
  - For DRA7x (QNX HLOS), ipumm/platform/ti/dce/baseimage/qnx_custom_rsc_table_vayu_ipu.h
  - For DRA7x (Linux or Android HLOS), ipumm/platform/ti/dce/baseimage/custom_rsc_table_vayu_ipu.h
- IPUMM *config* file where the memory map configuration for IPUMM is defined:
  - For DRA7x (differentiate by the value of hdw_type), ipumm/build/config.bld
- DCE *config* file where the DCE heap memory size is configured is defined:
  - For DRA7x, ipumm/platform/ti/dce/baseimage/dce_ipu.cfg

Each of the four files and their purpose are described below.

3.1 QNX BSP IFS Build File

Memory sections with pre-defined physical addresses (excluding register or TILER addresses) must be set aside in the QNX BSP IFS build file so that the memory is not given to other programs and can be used solely by the IPU.

The QNX build file for the DRA7x BSP is located in the QNX BSP package in this path:

<qnx_bsp_dir>/src/hardware/startup/boards/<your_board>/build

Specify a section to be set aside by modifying the startup line to use the “-r” option. For example, to reserve 0x5A0000 bytes, at physical address 0xBA30000, on DRA7x QNX BSP build file the startup-dra74xevm arguments would be (highlighted below):

```
startup-dra74xevm -r 0xBA300000,0x5A00000 -vvv -n852,668
```

3.2 IPUMM Custom Resource Table

The IPUMM custom resource table, located in the IPUMM code, is responsible for defining all the memory used by the IPU. The information in the custom resource table is used by the IPC 3.x for allocating memory sections, programming the MMU, verifying the availability of segments with pre-defined physical addresses, and setting up the IPC communication.

The IPUMM custom resource table is located in the following path in the IPUMM code:

```
For DRA7x (QNX HLOS), ipumm/platform/ti/dce/baseimage/qnx_custom_rsc_table_vayu_ipu.h
For DRA7x (Linux or Android HLOS), ipumm/platform/ti/dce/baseimage/custom_rsc_table_vayu_ipu.h
```

If any changes are made to the number, size, or addresses of memory used by the IPU, they must be reflected in the IPUMM custom resource table. This includes carveouts, device mem, and even register addresses.

There are various types of entries that the resource table can contain, but the two that are important when considering increasing, decreasing, adding or removing memory are the following:

- TYPE_CARVEOUT (Carveout Memory entries)
- TYPE_DEVMEM (Device Memory entries)

TYPE_CARVEOUT entries are used for the IPU code, data, and heap memory, and the physical memory for these is allocated dynamically by the IPC 3.x when loading the IPUMM image.

TYPE_DEVMEM entries are used for memory entries with pre-defined physical addresses. For all the device memory entries that are not register or TILER addresses (with the exception of the VRING entry), the physical addresses must be set aside and made known to the QNX BSP IFS build file.

CAUTION

When changing memory section sizes, the addresses of adjacent sections may be affected and may need to be adjusted accordingly.

3.3 IPUMM Config File

The IPUMM *config* file, located in the IPUMM code, is used when compiling the IPUMM image to specify the memory sections containing the code, data, and heap. These sections can be reduced if the total is not used or increased if more room is needed.

The IPUMM *config* file is located in the IPUMM code - in the following path:

```
ipumm/build/config.bld
```

CAUTION

When changing memory section sizes, the addresses of adjacent sections may be affected and may need to be adjusted accordingly.

if addresses and sizes are changed here, the updates need to be reflected in the IPUMM custom resource table.

3.4 DCE Config File

The DCE *config* file, located in the IPUMM code, is used to define the heap memory size for DCE. These sections can be reduced based on the needed configuration.

The DCE *config* file is located in the IPUMM code, in the following path:

```
ipumm/platform/ti/dce/baseimage/dce_ipu.cfg
```

CAUTION

When changing memory section sizes, the addresses of adjacent sections may be affected and may need to be adjusted accordingly.

If addresses and sizes are changed here, the updates need to be reflected in the IPUMM custom resource table and IPUMM *config* file.

4 Modifying Memory Segments

To modify the memory sections, all or some combination of the configuration files need to be updated. The IPUMM custom resource table definitely has to be modified. Then, depending on which segment is modified, the other files may need to be modified as well. If the segment that is modified is TYPE_CARVEOUT (Carveout Memory Entry) in the resource table, then the IPUMM *config* file may also need to be modified. If the segment that is modified is TYPE_DEVMEM (Device Memory Entry), then the QNX BSP IFS build file may need to be modified.

The steps are as follows:

1. Modify the memory entry in the IPUMM *config.bld* file (if required).

   If it is an existing segment that is being changed, check the *config.bld* file to see if the segment or part of the segment exists in the externalMemoryMap. If it exists, the entry in the *config.bld* file needs to be updated.

   Carefully check for any adjacent memory sections listed in the *config.bld* file. Increasing or reducing the size of a memory section can affect the base address of other sections.

2. Modify the IPUMM custom resource table.

   Whether adding, removing, increasing, or decreasing memory usage, the IPUMM custom resource table needs to be updated. All memory used by the IPUMM is represented in the IPUMM custom resource table.

3. Modify the QNX BSP IFS build file (if required).

   If the modification resulted in a change to a TYPE_DEVMEM entry, or the addition of a TYPE_DEVMEM entry in the DDR memory range, the QNX BSP IFS build file must be updated to set aside the memory.

4. Rebuild

   After modifying the files, rebuild the IPUMM code to generate a new binary that takes the changes.

   If the QNX BSP IFS build file was modified, rebuild the QNX BSP IFS image also.

5 Examples

5.1 Example 1. Decreasing the Size of a Memory Segment

If there is a memory segment that is too large, it can be reduced by modifying the configuration files.

Consider, for an example, that the complete set of use cases requires a smaller heap than the default heap size of the image. The default heap size is 41MB, but you know that you only need 34MB to satisfy your use cases. You are aware that the IPUMM image is putting the heap in the IPU_MEM_DATA:EXT_HEAP section, so that is the section you will want to reduce.

1. Modify the heap memory size in the DCE *dce_ipu.cfg* file (if required).

   The heapMemParams.size was found when checking the *dce_ipu.cfg* file :

```
/* Heap Memory is set to 40 MB.
 * This is considering 2 1080p instances of Mpeg4 Decoders, each
 * requiring 14 MBs and a single instance of H264 Encode requiring
 * 8 MBs running parallely.
 */
var heapMemParams           = new HeapMem.Params;
heapMemParams.size          = 0x2800000; // 40MB
heapMemParams.sectionName   = ".systemHeap";
var heap0                   = HeapMem.create(heapMemParams);
Memory.defaultHeapInstance  = heap0;
Program.global.heap0        = heap0;
```

By default, the heapMemParams.size is set to 40MB within the available IPU_MEM_DATA:EXT_HEAP section of 41MB. The heapMemParams.size needs to be modified to 33MB to satisfy the use cases within the available IPU_MEM_DATA:EXT_HEAP section of 34MB.

The size of heapMemParams.size is modified as shown below:

```
/* Heap Memory is modified to 33 MB from the default 40 MB configuration.
*/
var heapMemParams           = new HeapMem.Params;
heapMemParams.size          = 0x02100000; // 33 MB
heapMemParams.sectionName   = ".systemHeap";
var heap0                   = HeapMem.create(heapMemParams);
Memory.defaultHeapInstance  = heap0;
Program.global.heap0        = heap0;
```

2. Modify the memory entry in the IPUMM *config.bld* file (if required).

   The externalMemoryMap was found when checking the *config.bld* file:

```
var evmDRA7x_ExtMemMapIpu2 = {
        EXT_CODE:  { name: "EXT_CODE", base: 0x00004000, len:  0x005FC000, space: "code",
access: "RWX" },
        EXT_DATA:  { name: "EXT_DATA", base: 0x80000000, len:  0x00200000, space: "data",
access: "RW" },
        EXT_HEAP: { name: "EXT_HEAP", base: 0x80200000, len:  0x02900000, space: "data",
access: "RW" },
        TRACE_BUF: { name: "TRACE_BUF", base: 0x9F000000, len:  0x00060000, space: "data",
access: "RW" },
        EXC_DATA:  { name: "EXC_DATA", base: 0x9F060000, len:  0x00010000, space: "data",
access: "RW" },
        PM_DATA:   { name: "PM_DATA", base: 0x9F070000, len:  0x00020000, space: "data",
access: "RWX" }
    };
```

   The EXT_HEAP section is presented in the *config.bld* file. So, the EXT_HEAP entry needs to be modified.

   Carefully check for any adjacent memory sections listed in the *config.bld* file. Increasing or reducing the size of a memory section can affect the base address of other sections. In this case, only the EXT_HEAP section needs to be modified.

   The length of the EXT_HEAP section is modified as shown below:

```
var evmDRA7x_ExtMemMapIpu2 = {
        EXT_CODE:  { name: "EXT_CODE", base: 0x00004000, len:  0x005FC000, space: "code",
access: "RWX" },
        EXT_DATA: { name: "EXT_DATA", base: 0x80000000, len:  0x00200000, space: "data",
access: "RW" },
        EXT_HEAP:  { name: "EXT_HEAP", base: 0x80200000, len:  0x02200000, space: "data",
access: "RW" },
        TRACE_BUF: { name: "TRACE_BUF", base: 0x9F000000, len:  0x00060000, space: "data",
access: "RW" },
        EXC_DATA:  { name: "EXC_DATA", base: 0x9F060000, len:  0x00010000, space: "data",
access: "RW" },
        PM_DATA:   { name: "PM_DATA", base: 0x9F070000, len:  0x00020000, space: "data",
access: "RWX" }
    };
```

3. Modify the resource table.

Now, the IPUMM custom resource table is modified to reflect the change in the section size. Inspecting the resource table, it was determined that the entry for the EXT_HEAP is based on the base address and length. The base address for EXT_HEAP is located in the “IPU_MEM_DATA” entry, so this entry needs to be modified.

If the original entry in the resource table looks like the following:

```
#define IPU_MEM_DATA            0x80000000

[...]

/*
 * IPU_MEM_DATA_SIZE contains the size of EXT_DATA + EXT_HEAP
 * defined in the dce_ipu.cfg
 */
#define IPU_MEM_DATA_SIZE       (SZ_1M * 43)

[...]

struct my_resource_table ti_ipc_remoteproc_ResourceTable = {
    [...]

    {
        TYPE_CARVEOUT,
        IPU_MEM_DATA, 0,
        IPU_MEM_DATA_SIZE, 0, 0, "IPU_MEM_DATA",
    },

    [...]
};
```

The address and size are defined by IPU_MEM_DATA and IPU_MEM_DATA_SIZE. The starting address of the heap was not modified, so no need to change that, but the size of the heap has been reduced from 41MB to 34MB, a reduction of 7MB.

So, the size of the IPU_MEM_DATA_SIZE was reduced by 7MB:

```
#define IPU_MEM_DATA            0x80000000

[...]

/*
 * IPU_MEM_DATA_SIZE contains the size of EXT_DATA + EXT_HEAP
 * defined in the dce_ipu.cfg
 */
#define IPU_MEM_DATA_SIZE       (SZ_1M * 36)

[...]

struct my_resource_table ti_ipc_remoteproc_ResourceTable = {
    [...]

    {
        TYPE_CARVEOUT,
        IPU_MEM_DATA, 0,
        IPU_MEM_TEXT_SIZE, 0, RPROC_MEMREGION_CODE, “IPU_MEM_TEXT”,
    },

    [...]
};
```

4. Modify the QNX BSP IFS build file (if required).

   To tell whether the QNX BSP IFS build file needs to be modified, look at the type of resource that was modified. If it was a resource table TYPE_CARVEOUT memory entry, then do not modify the build file. If it was a resource table TYPE_DEVMEM memory entry in the DDR memory range, then you must modify the build file.

   In this example, a resource table TYPE_CARVEOUT memory entry was modified. TYPE_CARVEOUT memory entries do not have pre-defined physical addresses, so there is no need to modify the build file.

5. Rebuild

   Rebuild the IPUMM code to generate a new binary that takes the changes.

   There is no need to rebuild the QNX BSP IFS, because the build file was not modified.

5.2 Example 2. Removing a Memory Segment

If there is a memory segment that is not being used, it can be removed by modifying the configuration files.

Take for an example that the complete set of use cases never used the IPU_MEM_IOBUFS section. In that case, you can completely remove this section.

1. Modify the IPUMM *config.bld* file (if required).

   When checking the *config.bld* file, you will see the following:

```
var evmDRA7x_ExtMemMapIpu2 = {
        EXT_CODE:  { name: "EXT_CODE", base: 0x00004000, len:  0x005FC000, space: "code",
access: "RWX" },
        EXT_DATA:  { name: "EXT_DATA", base: 0x80000000, len:  0x00600000, space: "data",
access: "RW" },
        EXT_HEAP:  { name: "EXT_HEAP", base: 0x80600000, len:  0x02900000, space: "data",
access: "RW" },
        TRACE_BUF: { name: "TRACE_BUF", base: 0x9F000000, len:  0x00060000, space: "data",
access: "RW" },
        EXC_DATA:  { name: "EXC_DATA", base: 0x9F060000, len:  0x00010000, space: "data",
access: "RW" },
        PM_DATA:   { name: "PM_DATA", base: 0x9F070000, len:  0x00020000, space: "data",
access: "RWX" }
    };
```

   The IPU_MEM_IOBUFS section is not seen in the *config.bld* file. So, you do not need to modify this file.

2. Modify the resource table.

   The resource table can be modified to remove the IPU_MEM_IOBUFS section. Inspecting the resource table, you will find the entry for the IPU_MEM_IOBUFS.

   The original entry in the IPUMM custom resource table looks like the following:

```
#define IPU_MEM_IOBUFS          0x90000000
#define IPU_MEM_IOBUFS_SIZE     (SZ_1M * 90)
#define PHYS_MEM_IOBUFS         0xBA300000

[...]

struct my_resource_table {
    struct resource_table base;

    UInt32 offset[18];  /* Should match 'num' in actual definition */

    /* rpmsg vdev entry */
    struct fw_rsc_vdev rpmsg_vdev;
```

```
    [...]

    /* devmem entry */
    struct fw_rsc_devmem devmem0;

    /* devmem entry */
    struct fw_rsc_devmem devmem1;

    [...]
};

[...]

struct my_resource_table ti_ipc_remoteproc_ResourceTable = {
    1,      /* we're the first version that implements this */
    18,     /* number of entries in the table */
    0, 0,   /* reserved, must be zero */
    /* offsets to entries */
    {
        offsetof(struct my_resource_table, rpmsg_vdev),

        [...]

        offsetof(struct my_resource_table, devmem0),
        offsetof(struct my_resource_table, devmem1),

        [...]
    },

    [...]

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

    {
        TYPE_DEVMEM,
        IPU_MEM_IOBUFS, PHYS_MEM_IOBUFS,
        IPU_MEM_IOBUFS_SIZE, 0, 0, "IPU_MEM_IOBUFS",
    },

    [...]
};
```

After finding the IPU_MEM_IOBUFS entry, a TYPE_DEVMEM can be seen. All the TYPE_DEVMEM entries are grouped together and you also see that, in this example, the entry is the second TYPE_DEVMEM entry in the resource table.

Delete the IPU_MEM_IOBUFS entry from the resource table. Once that is done, update the offsets array need to be updated to remove the corresponding devmem1 entry. Also, update the 'num' parameter of the resource table by reducing it by 1, indicating that an entry has been removed.

Additionally, update the definition of the resource table struct. The size of the offset array should be reduced by 1 and the corresponding devmem entry should be removed.

The updated example will look like the following:

```
struct my_resource_table {
    struct resource_table base;

    UInt32 offset[17];  /* Should match 'num' in actual definition */

    /* rpmsg vdev entry */
    struct fw_rsc_vdev rpmsg_vdev;

    [...]

    /* devmem entry */
    struct fw_rsc_devmem devmem0;

    /* removed devmem entry 1 */

    /* devmem entry */
    struct fw_rsc_devmem devmem2;

    [...]
};

[...]

struct my_resource_table ti_ipc_remoteproc_ResourceTable = {
    1,      /* we're the first version that implements this */
    17,     /* number of entries in the table */
    0, 0,   /* reserved, must be zero */
    /* offsets to entries */
    {
        offsetof(struct my_resource_table, rpmsg_vdev),

        [...]

        offsetof(struct my_resource_table, devmem0),

        /* removed devmem entry 1 */

        offsetof(struct my_resource_table, devmem2),

        [...]
    },

    [...]
};
```

3. Modify the QNX BSP IFS build file (if required).

   To tell whether the QNX BSP IFS build file needs to be modified, look at the type of resource that was modified. If it was a resource table TYPE_CARVEOUT (Carveout Memory entry), the build file does not need to be modified. If it was a resource table TYPE_DEVMEM (Device Memory entry) in the DDR memory range, the build file must be modified.

   In this example, the entry was a resource table TYPE_DEVMEM, which had a pre-defined physical address. So, the build file needs to be modified.

   If, in the QNX BSP IFS build file, memory is being set aside for the IPU that corresponds to the physical address of the TYPE_DEVMEM entry, then it should be modified so that the memory is no longer set aside for the IPU.

   For example, if the QNX BSP IFS build file was setting aside the memory for the IPU_MEM_IOBUFS entry (as shown in the following example):

```
startup-dra74xevm -r 0xBA300000,0x5A00000 -vvv -n852,668
```

   Then, it should be changed to the following:

```
startup-dra74xevm –vvv -n852,668
```

4. Rebuild

   Rebuild the IPUMM code to generate a new binary that takes the changes. Note that since only a header file was modified in this case, you have to clean and build the IPUMM.

   Rebuild the QNX BSP IFS, since the build file was modified.

5.3 Example 3. Modifying IPUMM to Support H.264 Decode and Encode Only

By default, the IPUMM memory configuration is based on simultaneously running two instances of 1080p MPEG4 decoding (requires 14MB HEAP memory for each instance) and one instance of 1080p H.264 encoding (requires 8MB HEAP memory). The default IPUMM memory configuration is also configured to support five decoder codecs (H.264, MPEG4/H.263, VC1, MPEG2, and MJPEG) and two encoder codecs (H.264 and MPEG4/H.263).

When starting IPC with the default IPUMM firmware image on IPU2, the default heap memory will be 52MB.

```
# ipc IPU2 /sd/stage/usr/lib/DRA7x-m4-ipu2.xem4
# pidin -p ipc mem
     pid tid name               prio STATE            code  data        stack
  200732   1 sd/stage/bin/ipc    10r SIGWAITINFO         0   52M   12K(516K)*
  200732   2 sd/stage/bin/ipc    10r CONDVAR             0   52M  4096(132K)
  200732   3 sd/stage/bin/ipc    29r CONDVAR             0   52M  4096(132K)
  200732   4 sd/stage/bin/ipc    29r RECEIVE             0   52M  4096(132K)
  200732   5 sd/stage/bin/ipc    29r CONDVAR             0   52M  8192(132K)
  200732   6 sd/stage/bin/ipc    30r RECEIVE             0   52M  4096(132K)
  200732   7 sd/stage/bin/ipc    29r CONDVAR             0   52M  4096(132K)
  200732   8 sd/stage/bin/ipc    29r RECEIVE             0   52M  4096(132K)
  200732   9 sd/stage/bin/ipc    29r CONDVAR             0   52M  4096(132K)
  200732  10 sd/stage/bin/ipc    29r RECEIVE             0   52M  4096(132K)
  200732  11 sd/stage/bin/ipc    29r CONDVAR             0   52M  4096(132K)
  200732  12 sd/stage/bin/ipc    29r CONDVAR             0   52M  4096(132K)
  200732  13 sd/stage/bin/ipc    10r RECEIVE             0   52M  4096(132K)
  200732  14 sd/stage/bin/ipc    10r RECEIVE             0   52M  4096(132K)
  200732  15 sd/stage/bin/ipc    10r RECEIVE             0   52M  4096(132K)
            ipc                @ 8048000             240K   24K
            libc.so.3          @ 1000000             560K   16K
            /dev/mem           @28000000 (48200000)         4096
            /dev/mem           @28100000 (       0)        1024K
            /dev/mem           @28200000 (       0)        6144K
            /dev/mem           @28800000 (       0)          43M
            /dev/mem           @2b300000 (       0)        1024K
            /dev/mem           @28001000 (48840000)         4096
            /dev/mem           @28002000 (48842000)         4096
            /dev/mem           @28003000 (4a002000)         4096
            /dev/mem           @28004000 (4a0f6000)         4096
            /dev/mem           @28005000 (4a0f6000)         4096
            /dev/mem           @28006000 (4a005000)         4096
            /dev/mem           @28007000 (48036000)         4096
            /dev/mem           @28008000 (4803e000)         4096
            /dev/mem           @28009000 (4ae06000)          12K
            /dev/mem           @2800c000 (55082000)         4096
            /dev/mem           @2800d000 (4a008000)          12K
            /dev/mem           @28010000 (a4300000)         288K
            /dev/mem           @28058000 (aaa00000)          32K
```

For an example, the complete set of use cases is used to support a single instance of decoding H.264 1080p L4.1 or encoding H.264 1080p L4.1.

1. Modify the heap memory size in the DCE *dce_ipu.cfg* file (if required).

   Checking the *dce_ipu.cfg* file, the heapMemParams.size and the loaded codec lists are shown:

```
/* Heap Memory is set to 40 MB.
 * This is considering 2 1080p instances of Mpeg4 Decoders, each
 * requiring 14 MBs and a single instance of H264 Encode requiring
 * 8 MBs running parallely.
 */
var heapMemParams           = new HeapMem.Params;
heapMemParams.size          = 0x2800000; // 40MB
heapMemParams.sectionName   = ".systemHeap";
var heap0                   = HeapMem.create(heapMemParams);
Memory.defaultHeapInstance  = heap0;
Program.global.heap0        = heap0;
[...]
loadCodec('ti.sdo.codecs.mpeg4vdec.ce.MPEG4VDEC', 'ivahd_mpeg4dec');
loadCodec('ti.sdo.codecs.h264vdec.ce.H264VDEC',   'ivahd_h264dec');
loadCodec('ti.sdo.codecs.jpegvdec.ce.JPEGVDEC',   'ivahd_jpegvdec');
loadCodec('ti.sdo.codecs.vc1vdec.ce.VC1VDEC',     'ivahd_vc1vdec');
loadCodec('ti.sdo.codecs.mpeg2vdec.ce.MPEG2VDEC', 'ivahd_mpeg2vdec');
loadCodec('ti.sdo.codecs.h264enc.ce.H264ENC',     'ivahd_h264enc');
loadCodec('ti.sdo.codecs.mpeg4enc.ce.MPEG4ENC',   'ivahd_mpeg4enc');
[...]
```

This shows, by default, that the heapMemParams.size is set to 40MB. Modify the heapMemParams.size to 10MB because 1080p H.264 Level 4.1 requires around 9MB, and 1080p H.264 Level 4.1 requires around 8MB.

The size of heapMemParams.size is modified to the following:

```
/* Heap Memory is modified to 10 MB from the default 40 MB configuration.
*/
var heapMemParams           = new HeapMem.Params;
heapMemParams.size          = 0x00A00000; // 10MB
heapMemParams.sectionName   = ".systemHeap";
var heap0                   = HeapMem.create(heapMemParams);
Memory.defaultHeapInstance  = heap0;
Program.global.heap0        = heap0;
[...]
/* removed 'ivahd_mpeg4dec'*/
loadCodec('ti.sdo.codecs.h264vdec.ce.H264VDEC',   'ivahd_h264dec');
/* removed 'ivahd_jpegvdec'*/
/* removed 'ivahd_vc1vdec'*/
/* removed 'ivahd_mpeg2vdec'*/
loadCodec('ti.sdo.codecs.h264enc.ce.H264ENC',     'ivahd_h264enc');
/* removed 'ivahd_mpeg4enc'*/
 [...]
```

2. Modify the memory entry in the IPUMM *config.bld* file (if required).

When checking the *config.bld* file, you will see the externalMemoryMap:

```
var evmDRA7x_ExtMemMapIpu2 = {
        EXT_CODE:  { name: "EXT_CODE", base: 0x00004000, len:  0x005FC000, space: "code",
access: "RWX" },
        EXT_DATA:  { name: "EXT_DATA", base: 0x80000000, len:  0x00200000, space: "data",
access: "RW" },
        EXT_HEAP:  { name: "EXT_HEAP", base: 0x80200000, len:  0x02900000, space: "data",
access: "RW" },
        TRACE_BUF:  { name: "TRACE_BUF", base: 0x9F000000, len:  0x00060000, space: "data",
access: "RW" },
        EXC_DATA:   { name: "EXC_DATA", base: 0x9F060000, len:  0x00010000, space: "data",
access: "RW" },
        PM_DATA:    { name: "PM_DATA", base: 0x9F070000, len:  0x00020000, space: "data",
access: "RWX" }
    };
```

The EXT_CODE, EXT_DATA, and EXT_HEAP sections are included in the *config.bld* file. Due to removal of codec libraries on DCE *config* file (dce_ipu.cfg) the EXT_CODE and EXT_DATA where the codec libraries are loaded and configured can be reduced. The EXT_DATA memory usage can be reduced based on the used information after building the IPUMM firmware and checking the generated map files at:

- ipumm/platform/ti/dce/baseimage/package/cfg/out/ipu/release/ipu.xem4.map

The example of the default memory configuration is shown below:

```
MEMORY CONFIGURATION
         name             origin    length      used     unused   attr    fill
----------------------  --------  ---------  --------  --------  ----  --------
  L2_ROM                00000000   00004000  00000664  0000399c  RWIX
  EXT_CODE              00004000   005fc000  003327ee  002c9812  RW X
  L2_RAM                20000000   00010000  00000000  00010000  RWIX
  OCMC_RAM1             40300000   00080000  00000000  00080000  RWIX
  OCMC_RAM2             40400000   00100000  00000000  00100000  RWIX
  OCMC_RAM3             40500000   00100000  00000000  00100000  RWIX
  EXT_DATA              80000000   00200000  0005732c  001a8cd4  RW
  EXT_HEAP              80200000   02900000  02800000  00100000  RW
  TRACE_BUF             9f000000   00060000  00008000  00058000  RW
  EXC_DATA              9f060000   00010000  00000200  0000fe00  RW
  PM_DATA               9f070000   00020000  0001027c  0000fd84  RW X
```

The EXT_HEAP can be reduced because the heap memory size on the DCE *config* file (dce_ipu.cfg) is being reduced from 40MB to 10MB.

Carefully check for any adjacent memory sections listed in the *config.bld* file. Increasing or reducing the size of a memory section can affect the base address of other sections.

The length of the EXT_CODE, EXT_DATA, and EXT_HEAP sections are modified as shown below:

```
var evmDRA7x_ExtMemMapIpu2 = {
        EXT_CODE:  { name: "EXT_CODE", base: 0x00004000, len:  0x001FC000, space: "code",
access: "RWX" },
        EXT_DATA:  { name: "EXT_DATA", base: 0x80000000, len:  0x00100000, space: "data",
access: "RW" },
        EXT_HEAP:  { name: "EXT_HEAP", base: 0x80200000, len:  0x00A00000, space: "data",
access: "RW" },
        TRACE_BUF:  { name: "TRACE_BUF", base: 0x9F000000, len:  0x00060000, space: "data",
access: "RW" },
        EXC_DATA:   { name: "EXC_DATA", base: 0x9F060000, len:  0x00010000, space: "data",
access: "RW" },
        PM_DATA:    { name: "PM_DATA", base: 0x9F070000, len:  0x00020000, space: "data",
access: "RWX" }
    };
```

3. Modify the resource table.

   Modify the IPUMM custom resource table to reflect the change in the section size. When inspecting the resource table, it was determined that the entry for the EXT_HEAP is based on the base address and length. The base address for EXT_HEAP is located in the “IPU_MEM_DATA” entry, which is is the entry that needs modification.

   Consider that the original entry in the resource table looks like the following:

```
#define IPU_MEM_DATA            0x80000000

[...]

#define IPU_MEM_TEXT_SIZE       (SZ_1M * 6)

/*
 * IPU_MEM_DATA_SIZE contains the size of EXT_DATA + EXT_HEAP
 * defined in the dce_ipu.cfg
 */
#define IPU_MEM_DATA_SIZE       (SZ_1M * 43)

[...]

struct my_resource_table ti_ipc_remoteproc_ResourceTable = {
    [...]

    {
        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",
    },

    [...]
};
```

The address and size are defined by IPU_MEM_DATA, IPU_MEM_DATA_SIZE and IPU_MEM_TEXT_SIZE. The starting address of the heap was not modified; there is no need to change that, but the size of the heap “EXT_HEAP” has been reduced from 41MB to 10MB (a reduction of 31MB). The size of the “EXT_DATA” has been reduced as well from 2MB to 1MB (a reduction of 1MB).

The size of the IPU_MEM_DATA_SIZE, which is a combination of EXT_HEAP and EXT_DATA to 11MB, needs to be reduced.

The size of “EXT_CODE” has been reduced from approximately 6MB (0x005FC000) to approximately 2MB (0x001FC000), a reduction of 4MB. Reduce the size of the IPU_MEM_TEXT_SIZE from 6MB to 2MB.

```
#define IPU_MEM_DATA            0x80000000

[...]

#define IPU_MEM_TEXT_SIZE       (SZ_1M * 2)

/*
 * IPU_MEM_DATA_SIZE contains the size of EXT_DATA + EXT_HEAP
 * defined in the dce_ipu.cfg
 */
#define IPU_MEM_DATA_SIZE       (SZ_1M * 11)

[...]

struct my_resource_table ti_ipc_remoteproc_ResourceTable = {
    [...]

    {
        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",
    },

    [...]
};
```

4. Modify the QNX BSP IFS build file (if required).

   To tell whether the QNX BSP IFS build file needs to be modified, look at the type of resource that was modified. If it was a resource table TYPE_CARVEOUT memory entry, no modification to the build file is necessary. If it was a resource table TYPE_DEVMEM memory entry in the DDR memory range, then you must modify the build file.

   In this example, a resource table TYPE_CARVEOUT memory entry is modified. TYPE_CARVEOUT memory entries do not have pre-defined physical addresses, so there is no need to modify the build file.

5. Rebuild

Rebuild the IPUMM code to generate a new binary that takes the changes.

There is no need to rebuild the QNX BSP IFS, because the build file was not modified.

Checking the generated map files at:

- ipumm/platform/ti/dce/baseimage/package/cfg/out/ipu/release/ipu.xem4.map

The example of memory configuration after the modification is shown, see below:

```
MEMORY CONFIGURATION
         name            origin    length      used     unused   attr    fill
----------------------  --------  ---------  --------  --------  ----  --------
  L2_ROM                00000000   00004000  00000664  0000399c  RWIX
  EXT_CODE              00004000   001fc000  0017935c  00082ca4  RW X
  L2_RAM                20000000   00010000  00000000  00010000  RWIX
  OCMC_RAM1             40300000   00080000  00000000  00080000  RWIX
  OCMC_RAM2             40400000   00100000  00000000  00100000  RWIX
  OCMC_RAM3             40500000   00100000  00000000  00100000  RWIX
  EXT_DATA              80000000   00100000  00056ba8  000a9458  RW
  EXT_HEAP              80100000   00a00000  00a00000  00000000  RW
  TRACE_BUF             9f000000   00060000  00008000  00058000  RW
  EXC_DATA              9f060000   00010000  00000200  0000fe00  RW
  PM_DATA               9f070000   00020000  0001027c  0000fd84  RW X
```

When starting IPC with the modified IPUMM firmware image on IPU2, the modified heap memory will be around 16MB.

```
# pidin -p ipc mem
     pid tid name               prio STATE           code  data        stack
  221212   1 sd/stage/bin/ipc    10r SIGWAITINFO      360K   16M   20K(516K)*
  221212   2 sd/stage/bin/ipc    10r CONDVAR          360K   16M  4096(132K)
  221212   3 sd/stage/bin/ipc    29r CONDVAR          360K   16M  4096(132K)
  221212   4 sd/stage/bin/ipc    29r RECEIVE          360K   16M  4096(132K)
  221212   5 sd/stage/bin/ipc    29r CONDVAR          360K   16M  8192(132K)
  221212   6 sd/stage/bin/ipc    30r RECEIVE          360K   16M  4096(132K)
  221212   7 sd/stage/bin/ipc    29r CONDVAR          360K   16M  4096(132K)
  221212   8 sd/stage/bin/ipc    29r RECEIVE          360K   16M  4096(132K)
  221212   9 sd/stage/bin/ipc    29r CONDVAR          360K   16M  4096(132K)
  221212  10 sd/stage/bin/ipc    29r RECEIVE          360K   16M  4096(132K)
  221212  11 sd/stage/bin/ipc    29r CONDVAR          360K   16M  4096(132K)
  221212  12 sd/stage/bin/ipc    29r CONDVAR          360K   16M  4096(132K)
  221212  13 sd/stage/bin/ipc    10r RECEIVE          360K   16M  4096(132K)
  221212  14 sd/stage/bin/ipc    10r RECEIVE          360K   16M  4096(132K)
           libc.so.3          @ 1000000             464K   16K
           /dev/mem           @28000000 (48200000)        4096
           /dev/mem           @28100000 (       0)       1024K
           /dev/mem           @28200000 (       0)       2048K
           /dev/mem           @28400000 (       0)         11M
           /dev/mem           @28f00000 (       0)       1024K
           /dev/mem           @28001000 (48840000)        4096
           /dev/mem           @28002000 (48842000)        4096
           /dev/mem           @28003000 (4a002000)        4096
           /dev/mem           @28004000 (4a0f6000)        4096
           /dev/mem           @28005000 (4a0f6000)        4096
           /dev/mem           @28006000 (4a005000)        4096
           /dev/mem           @28007000 (48036000)        4096
           /dev/mem           @28008000 (4803e000)        4096
           /dev/mem           @28009000 (4ae06000)         12K
           /dev/mem           @2800c000 (55082000)        4096
           /dev/mem           @2800d000 (4a008000)         12K
           /dev/mem           @28010000 (d1000000)        288K
           /dev/mem           @28058000 (d0f00000)         32K
```

5.4 Example 4 (QNX only): Modifying Shared Memory IO Bufs

QNX IPC provides carved out memory through the shmemallocator component. For more information, see http://processors.wiki.ti.com/index.php/IPC_3.x_FAQ#QNX_Build.

An example is to reduce the Shared IO Buffer size from 90MB to 8MB.

Table 2 shows the memory layout in 1GB DRR address space for 8MB Shared IO Buffer.

Table 2. Memory Layout

Start of 1GB DDR	0x80000000
DDR Memory size 0x3F500000 (1013MB)	Start at 0x80000000
IPU_MEM_IOBUFS size 0x800000 (8MB)	Start at 0xBF500000
Unreserved size 0x300000 (3MB)	Start at 0xBFD00000
End of 1GB DDR	0xC0000000

1GB DRR end address is 0xC0000000. There should be an 3MB (0x300000) should be un reserved memory.

Shared IO Buffer Physical address start: 0xC0000000 – 0x300000 (3 MB) – 0x800000 (8MB) =>0xBF50000

Below are the modifications required for 8 MB Shared IO Buffers.

1. Modify the resource table.

   The QNX custom resource table has to be modified to change the IPU_MEM_IOBUFS section. On inspection of the resource table, the entry for the IPU_MEM_IOBUFS was found.

```
For DRA7x (QNX HLOS), ipumm/platform/ti/dce/baseimage/qnx_custom_rsc_table_vayu_ipu.h
```

   Consider that the original entry in the resource table looks like the below:

```
#define IPU_MEM_IOBUFS          0x90000000
#define PHYS_MEM_IOBUFS         0xBA300000
#define IPU_MEM_IOBUFS_SIZE     (SZ_1M * 90)
```

   After modifications for 8 MB:

```
#define IPU_MEM_IOBUFS          0x90000000
#define PHYS_MEM_IOBUFS         0xBF500000
#define IPU_MEM_IOBUFS_SIZE     (SZ_1M * 8)
```

2. Modify the shared memory allocator.

   Modify the shared memory block start address and block size in SharedMemoryAllocator.c.

   SharedMemoryAllocator.c is located at /ipc_3_xx_xx_xx/qnx/src/ipc3x_dev/sharedmemallocator/resmgr/SharedMemoryAllocator.c

```
#define SH_MEM_BLOCK1_START          0xBA300000
#define SH_MEM_BLOCK1_SIZE           0x5A00000
```

   After modifications for 8MB:

```
#define SH_MEM_BLOCK1_START          0xBF500000
#define SH_MEM_BLOCK1_SIZE           0x800000
```

3. Modify the QNX IFS build file.

   Modify the QNX IFS build file to change the carve out address and size.

   For example, if the QNX IFS build file was setting aside the memory for the IPU_MEM_IOBUFS entry, as below:

```
startup-dra74xevm -r 0xBA300000,0x5A00000 -vvv -n852,668
```

   It should be changed to this:

```
startup-dra74xevm -r 0xBF500000,0x800000 -vvv -n852,668
```

4. Rebuild.

   Rebuild the QNX IPC (http://processors.wiki.ti.com/index.php/IPC_Install_Guide_QNX#Build) since SharedMemoryAllocator.c is modified.

   Rebuild the IPUMM code to generate a new firmware binary that takes the changes. Note that since only a header file was modified in this case, you have to clean and build the IPUMM

   Rebuild the QNX BSP IFS, since the build file was modified.

Revision History

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

Changes from Original (December 2015) to A Revision	Page
• Added new Section 5.4	18

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components 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 such information 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.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

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		

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