Nexus
Memory

Broadcom
Corporate Headquarters: San Jose, CA

Web: www.broadcom.com

Revision History

Revision	Date	Change Description
0.1	1/4/11	Initial draft
0.2	10/20/11	Update 7425, remove 7422, update msg_modules=nexus_platform_settings sample
0.3	12/19/11	Clarify fake addressing, add section on debug
1.0	6/15/12	Added Steps to Minimizing Memory Usage
1.1	8/30/12	Added section on Client Heaps, expanded notes on overriding default configurations
1.2	1/11/13	Expand “Monitoring Heaps and Allocations”
1.3	5/3/13	Update “Minimizing Memory Usage”
1.4	7/15/13	Removed “Limitations” section; some info was out of date. Convert diagrams to DBG output; some were out of date.
1.5	8/21/13	Revise “Reducing code size”
1.6	11/20/13	Revise “Reduce heap memory” section using memconfig feature for 14.1. Clarify CMA for ARM and bmem for MIPS.
1.7	3/28/14	Revise memconfig API for 14.2 Refactor Linux CMA/bmem section
1.8	8/11/14	Clarify distinction between OS memory and Nexus device memory Add section on 40 bit access
1.9	5/15/15	Add dynamic CMA section, Clarify box mode selection vs. custom RTS, Add BSEAV/tools/bmemconfig, Remove section on customizing nexus_platform_features.h, Clarify init-tune and run-time applications API’s to reduce memory
2.0	9/1/15	Convert to bmem on ARM. Revise “Linux bmem carve-out” section
2.1	10/6/15	Added section “Reduce Other Heap Sizes”

Table of Contents

Contents

Introduction

Embedded systems usually run with limited memory size, limited bandwidth and limited memory access. Therefore, the application designer must understand these limitations and design for them.

The document begins by describing Nexus heaps and how they are configured. This includes configuring which heaps are used for certain features based on memory requirements.

Next, the document gives step-by-step directions that each project must follow to determine and configure the minimum memory usage.

Finally, the document covers a variety of topics including memory mapping, typical memory usage and debugging memory problems.

Nexus Heaps

What is a heap?

Broadcom set-tops divide system memory into two groups: memory managed by the operating system (OS) and memory managed by Nexus. Memory managed by the OS is called just system memory, or OS memory or Linux memory. Memory managed by Nexus is called device memory or Nexus memory.

Nexus manages device memory by means of heaps. A heap is a region of physically contiguous memory from which Nexus or the application can allocate blocks of memory. Most device memory is allocated inside Nexus based on settings from the user; the app can also allocate directly using NEXUS_MemoryBlock_Allocate and related functions. All Nexus heaps provide physical offsets (also called physical addresses) for non-CPU device access. Some heaps are memory mapped for CPU access. The heap allows you to convert between physical offsets and virtual addresses.

OS memory is allocated using functions including malloc (in user space) or kmalloc (in Linux kernel space). Malloc'd memory is not guaranteed to be physically contiguous and can be paged out or re-located. Also, we cannot easily convert between virtual and physical addresses. Also, malloc/kmalloc doesn't allow selection based on MEMC. For all these reasons this memory is not directly usable by Nexus/Magnum controlled devices.[1] However, memory allocated from Nexus heaps can sometimes be used by OS devices.

Platform Default Heaps

The Nexus platform code will configure a set of heaps for the typical usage of the reference board[2]. Your application can override these settings after calling NEXUS_Platform_GetDefaultSettings and before calling NEXUS_Platform_Init. However, be aware that many chips have complex memory architectures which are required for even basic features like video decode. You can study the default heap layout to learn some of these requirements.

[1] Physically discontiguous memory results from memory mapping. If a device wants direct memory access apart from the OS, it must be contiguous and locked. Nexus Dma and Graphics2D interfaces have support for offsets, not addresses. So, access to kernel allocated memory is possible, provided your application can convert Linux virtual addresses to offset and can lock the pages.

[2] See nexus_platform_$(NEXUS_PLATFORM).c and NEXUS_Platform_P_GetDefaultSettings.

Nexus gets device memory for its heaps from Linux using the "bmem" boot parameter. bmem carves out memory so that it is not usable by Linux and available to Nexus.[3] Each bmem region is physically contiguous. There may be zero, one or more bmem regions on each MEMC. See later section for information on how to configure and read bmem regions in Linux.

Nexus will automatically bound all heaps to fit within the reported bmem regions. It will not allow a heap to be created outside of the bmem regions.

When writing general purpose code, it's important to allow flexibility for heap configuration. The heap layout for one system may be quite different from another system. The application is the final arbiter of the system design and should be able to override any default heap configuration.

Configuring Heaps

You will likely need to customize your heap configuration for your application needs. You will also need to understand the meaning and use of each heap.

The following is a snippet from nexus_platform_init.h for customizing heaps.

```
typedef struct NEXUS_PlatformSettings
{
    struct {
        unsigned memcIndex;  /* MEMC index */
        unsigned subIndex;   /* unused */
        int size;            /* size of heap, with -1 option */
        unsigned alignment;
        unsigned memoryType; /* memory mapping */
        bool optional;       /* ok if heap can't be created */
    } heap[NEXUS_MAX_HEAPS];
}
```

After calling NEXUS_Platform_GetDefaultSettings, you can change any of the heap parameters. The heaps will only be created when you call NEXUS_Platform_Init.

Please see the header file for detailed API-level comments.

- If *size* == -1, the remainder of the memory in the bmem region will be assigned to that heap. You can only have one *size* == -1 per bmem region. This is not supported with ARM/CMA systems.

- *alignment* is the minimum byte alignment of allocations in the heap. This can help reduce fragmentation. Nexus will also apply a chip-specific minimum alignment for

[3] Starting with Nexus 15.3 and Linux 3.14-1.8, ARM platforms also use bmem like MIPS platforms. Previously ARM platforms got device memory using a runtime CMA API. See the "Linux bmem carve-out" section for more details.

cache coherency. The max of the user *alignment* and the internal alignment will be used.

- *memoryType* is a bitmask which controls the memory mapping. See nexus_types.h for bitmasks and macros. The following are typical combinations:

NEXUS_MemoryType	Mapping	Usage
eFull	Driver/Server and Application	Default heap, playback, VBI, DMA descriptors, Userdata
eApplication	Application	Graphics, record
eDeviceOnly	No mapping	Picture buffers
eSecure	No mapping	Decoder CDB, AVD cabac bin buffer

- The *optional* boolean allows NEXUS_Platform_Init to succeed even if the heap cannot be created. By default, Init will fail if all requested heaps cannot be created, which makes system debug much easier.

Using Heaps

Nexus heaps can be specified in two ways: by index or by handle.

When setting NEXUS_PlatformSettings for NEXUS_Platform_Init, no Nexus handle exists. Therefore, all heap configuration must be set using heap indices. These indices refer to heaps that will be created. The following are typical:

API	Usage
NEXUS_DisplayModuleSettings. primaryDisplayHeapIndex	Heap used by VDC if per-window and per-source heaps are not specified
NEXUS_DisplayModuleSettings. videoWindowHeapIndex[]	Default per-window heap for VDC
NEXUS_VideoDecoderModuleSettings. avdHeapIndex[]	XVD heap for picture buffers
NEXUS_VideoDecoderModuleSettings. hostAccessibleHeapIndex	XVD heap for FW, userdata

After NEXUS_Platform_Init has completed, a set of NEXUS_HeapHandles are available using NEXUS_Platform_GetConfigurtion. If you want to customize your memory usage, you can pass the heap handle to the Nexus API. The following are typical uses:

API	Usage
NEXUS_SurfaceCreateSettings.heap	Heap to allocate surface from
NEXUS_VideoWindowSettings.heap	Custom per-window heap for VDC
NEXUS_VideoDecoderOpenSettings.heap	Custom XVD heap for picture buffers
NEXUS_PlaypumpOpenSettings.heap	Heap for CDB allocation
NEXUS_PlaypumpOpenSettings.boundsHeap	Optional bounds check for scatter-gather
NEXUS_Graphics2DOpenSettings.heap	Heap for M2MC HW/SW fifo allocation
NEXUS_Graphics2DOpenSettings.boundsHeap	Optional bounds check for blits

Client Heaps

If you are using Nexus in a multi-process configuration, there is additional heap complexity. The server decides which heaps the client can access. In a secure system, the client should not be given access to any heap that could compromise the server. Therefore, it is recommended that you have a dedicated heap or set of heaps for clients.

Client heap numbering is not necessarily the same as the server heap numbering. Client heap numbering is the index of the NEXUS_ClientConfiguration.heap[] array. Server heap number is the index of the NEXUS_PlatformSettings.heap[] array. When the server assigns heaps to the client, it determines its numbering. It can also be different per client.

If an application uses an interface with a NEXUS_HeapHandle parameter, and if it leaves that parameter as a default NULL value, the nexus module code will select a default heap. If the call came from a client, it will select a default client heap. If the module code requires NEXUS_MemoryType_eFull mapping, it will select the first heap in the client's heap[] array with that mapping. If there is no driver-side mapping requirement, it will select the first heap in the client's heap[] array.

40 Bit Access

Broadcom set-tops use 32 bit CPU's and 32 bit registers. On earlier silicon, most hardware devices also had only 32 bit addressing capability. This translates to a maximum addressing range of 4 GB. However, some hardware devices on 28nm SoC's have been extended to allow 40 bit addressing, extending their addressing range to 1 TB.

OS controlled cores with 40 bit access include CPU (using Linux's Large Physical Addressing Extension or LPAE), Ethernet, MOCA, PCIe, SATA, USB, and Flash.

Nexus controlled cores with 40 bit access include M2MC (NEXUS_Graphics2d) and M2M_DMA (NEXUS_Dma). This limited list means most audio, video, transcode, transport and 3D graphics features must run in 32 bit space. Broadcom will be expanding the number of hardware cores with 40 bit capability.

Because of physical addressing requirements, any system with more than 1 GB per MEMC may require 40 bit access, even if the total memory is less than 4 GB. Because of the limited number of hardware cores which can reach 40 bit space, careful planning must be done before building such a system. Please review your plan with your FAE and get engineering feedback about your memory layout.

Dynamic CMA

On ARM chips Nexus can be configured with a heap which dynamically allocates and frees blocks of CMA memory from Linux. This allows for more Nexus/Linux memory sharing. CMA blocks are created by booting Linux with brcm_cma=SIZE@OFFSET, like bmem syntax. By default there are no CMA blocks.

A heap can be declared as dynamic using NEXUS_PlatformSettings.heap[].memoryType = NEXUS_MEMORY_TYPE_MANAGED|NEXUS_MEMORY_TYPE_ONDEMAND_MAPPED|NEXUS_MEMORY_TYPE_DYNAMIC. The dynamic heap is required to have on-demand memory mapping and no Nexus API which passes a virtual memory pointer to the server can be used because the server will not have any pointer access to the memory. All device memory access must be managed with handles, like NEXUS_MemoryBlockHandle.

Nexus API's to grow and shrink the heap are NEXUS_Platform_GrowHeap and NEXUS_Platform_ShrinkHeap. If the application tries to allocate from the dynamic heap and it fails, it can choose to grow the heap and try again.

This has been integrated with NxClient. Run nxserver with the –growHeapBlockSize option. Also see NxClient_GrowHeap and ShrinkHeap.

Steps to Minimizing Memory Usage

The following steps should be performed for every Nexus project.

Step 1: Select Box Mode or Get RTS Analysis

Broadcom Memory Controllers (MEMC's) use a Real-Time Scheduling (RTS) system to allocate guaranteed or round-robin memory bandwidth to the various memory clients in the system.

For newer systems (generally 28nm), we support multiple RTS configurations using "Box Modes". A single system may have one or more box modes. Each box mode corresponds to one RTS configuration and the use cases it supports. The box mode is set using the Bolt bootloader. Custom box modes can be requested.

For older systems (generally 40nm), there is only one RTS delivered with every reference platform. We recommend customers use that default RTS, but a custom RTS analysis can be requested.

Custom box modes or RTS may cause a change in the layout of memory allocations in the system, so this should be done early in the project.

This document will assume that you are using default box modes or the default RTS programming for your project.

Step 2: Reduce Picture Buffer Heap Memory

Nexus provides a platform API for calculating picture buffer heap sizes based on capabilities. This API is referred to as "memconfig". The API is NEXUS_MemoryConfigurationSettings located in nexus/platforms/common/include/nexus_platform_memconfig.h. It is passed into NEXUS_Platform_MemConfigInit, an alternative to NEXUS_Platform_Init.

By default, the application does not need to configure heap memory. Picture buffer heaps will be calculated for maximum capabilities.

The following is an example of how to reduce memory by reducing features:

```
NEXUS_PlatformSettings platformSettings;
NEXUS_MemoryConfigurationSettings memSettings;

NEXUS_Platform_GetDefaultSettings(&platformSettings);
NEXUS_GetDefaultMemoryConfigurationSettings(&memSettings);

memSettings.display[0].maxFormat = NEXUS_VideoFormat_e1080i;
memSettings.display[1].window[0].used = false;
memSettings.videoDecoder[2].used = false;
memSettings.videoDecoder[0].maxFormat = NEXUS_VideoFormat_e1080i;
memSettings.videoDecoder[1].maxFormat = NEXUS_VideoFormat_e1080i;

NEXUS_Platform_MemConfigInit(&platformSettings, &memSettings);
```

There are several ways to use the memconfig API to determine your memory layout. All of them require running on Broadcom set-tops. If you are trying to estimate for a new set-top, you will need to use an existing set-top or be given estimates not driven by memconfig.

- Run BSEAV/tools/bmemconfig. This runs a web-server on the set-top and provides an HTML GUI for reducing heap memory. See the README.txt in that directory for more information.

- Run nexus/examples/memconfig. It contains a variety of canned options to reduce memory, and these can be used as guidelines for pursuing more memory reduction using the memconfig API.

- Run nexus/utils/playback.c with its "-mem {low|medium|high}" command line option. Again, you can customize the app and do more reductions.

- Study the reference platform's default memconfig settings in nexus/platforms/$(NEXUS_PLATFORM)/src/nexus_platform_$(NEXUS_PLATFORM).c.

Typical features which lead to large memory reductions are:

- Number of displays and windows and their maximum format, especially stereoscopic TV (3DTV) support using display[].maxFormat.
- Number of decoders and their maximum format, especially MVC and VP9 decode using videoDecoder[].supportedCodecs[].
- Number of video encoders and their maximum format

System Memory Worksheets

For platforms that don't have preceding Nexus "memconfig" support (generally 65nm), you must calculate picture buffer sizes using the system memory worksheets[4]. Depending on the platform, you may be provided a platform-specific Microsoft® Excel® worksheet called System_Memory_Worksheet.xls. Or you may be given two generic spreadsheets named XVD_Memory_Worksheet.xls and VDC_Allocation_STB.xls. These worksheets can be requested from your FAE.

The system memory worksheets allow you to calculate the heap requirements for your features. You should enable and disable a variety of features to learn what each feature costs in terms of memory. If you simply enable every feature, it may require an unexpectedly large amount of memory.

There are two main blocks handled by the worksheet: VideoDecoder and Display. The requirements for each block will be added together into a "picture buffer heap". If your system has two memory controllers, these numbers may be split into two heaps, one on each MEMC.

The results of the spreadsheet must be programmed into NEXUS_PlatformSettings. Each reference platform has a nexus_platform_$(NEXUS_PLATFORM).c file which programs defaults. You can modify these defaults for your platform, or have your application set your own settings before calling NEXUS_Platform_Init.

Because of the complexity of the system, you will need to test the numbers and verify that they are correct for your usage mode. It is possible that adjustments will need to be made.

[4] If your earlier release does not support memconfig, you can get a newer release, run it on a reference platform, and use the memconfig API to determine ideal heap sizes. This information can then be backported to the earlier release.

Step 3: Reduce Other Heap Sizes

For non-picture buffer heaps, the customer should manually reduce the heap sizes to what is needed. The default Nexus heap sizes are chosen for the Broadcom reference board and are much larger than what most customers need.

To reduce, follow this process:

- Run your application through its various features
- Read peak heap utilization using core proc or NEXUS_Heap_GetStatus. See “Printing heap status” section later in this document for instructions.
- Determine the amount of padding (extra memory) you need in each heap to mitigate fragmentation risk
- Write your application to modify the heap size before calling NEXUS_Platform_Init
- Test your application again to confirm the sizes are right

If you have frequent runtime allocation and de-allocation in a heap, memory will become fragmented. You must have a generous amount of unused space in order to not run out of memory due to fragmentation. Monitoring “largest block available” in combination with “free space” can measure the amount fragmentation. Only rigorous test can confirm that fragmentation is not a risk for your system.

Heaps with only init-time allocation (like the picture buffer heaps) will not have fragmentation problems and can be sized very closely.

In most reference platforms, there are standard heaps which are known by standard macros. They should be handled as follows:

Standard Macro	Use	Reduction
NEXUS_MEMC0_MAIN_HEAP	General purpose	Reduce to peak usage plus padding
NEXUS_MEMCx_DRIVER_HEAP	General purpose	Reduce to peak usage plus padding
NEXUS_MEMCx_GRAPHICS_HEAP	Offscreen graphics	Reduce to peak usage plus padding
NEXUS_MEMCx_PICTURE_BUFFER_HEAP	Picture buffers heaps. For SVP systems, these are called URR and are protected.	Will be automatically sized by Nexus memconfig
NEXUS_VIDEO_SECURE_HEAP	Used for unmapped transport data	Reduce to peak usage

	like RS, XC and CDB buffers. For SVP systems, this is called the CRR and is protected. Still used even if SVP is not enabled.	plus padding
NEXUS_SAGE_SECURE_HEAP	Sage private data	Only needed if Sage is enabled

The following is an example of how to reduce heap sizes in the application:

```
NEXUS_PlatformSettings platformSettings;
NEXUS_Platform_GetDefaultSettings(&platformSettings);
platformSettings.heap[NEXUS_MEMC0_MAIN_HEAP].size = 37 * 1024 * 1024;
platformSettings.heap[NEXUS_VIDEO_SECURE_HEAP].size = 96 * 1024 * 1024;
NEXUS_Platform_Init(&platformSettings);
```

This manual sizing works for all heaps except picture buffer heaps, which are automatically sized.

Step 4: Reduce with NEXUS_PlatformSettings

Whether you use NEXUS_Platform_MemConfigInit or only NEXUS_Platform_Init, there are other init-time settings in NEXUS_PlatformSettings which can reduce memory.

- Reduce parser band and remux buffers using NEXUS_PlatformSettings. transportModuleSettings.clientEnabled.
- Reduce VBI allocations using NEXUS_PlatformSettings.displayModuleSettings.vbi.
- Reduce audio allocations using NEXUS_PlatformSettings.audioModuleSettings .numCompressedBuffers and .maxAudioDspTasks and .maxIndependentDelay and .numPcmBuffers.

NOTE: Previous versions of this document advised customizing nexus_platform_features.h macros to achieve these reductions. We now prefer init-time API code in the application.

Step 5: Set build options to reduce features

Nexus can be customized at build time to remove features. Customizations can be made with environment variables or modifying platform Makefiles. The following are some build flags which make a large difference:

```
# remove DSP features
export BDSP_VIDEO_ENCODE_SUPPORT=n
export BDSP_SCM_SUPPORT=n
export BDSP_ENCODER_SUPPORT=n
export BDSP_H264_ENCODE_SUPPORT=n
export BDSP_VORBIS_SUPPORT=n
export BDSP_WMAPRO_SUPPORT=n
export BDSP_MS10_SUPPORT=n

# remove usermode multiprocess support
export NEXUS_SERVER_SUPPORT=n

# remove video encoder Nexus module
export VIDEO_ENCODER_SUPPORT=n
```

Also, see nexus/platforms/common/build/platform_modules.inc. Each module .inc file is wrapped with a build-time variable. You can customize nexus/platforms/$(NEXUS_PLATFORM)/build/platform_options.inc to remove unused modules.

Step 6: Reduce code size

Nexus code size can be dramatically reduced by limiting features and static linking. Some specific ideas are:

- Removing unused modules and firmware will reduce code size. See previous section on setting build options.

- Compile without debug messages and warnings:

  ```
  export B_REFSW_DEBUG=n
  ```

  This translates to B_REFSW_DEBUG_LEVEL=err B_REFSW_DEBUG_COMPACT_ERR=y B_REFSW_DEBUG_ASSERT_FAIL=n. This removes TRACE, MSG and WRN. It reduces the size of ERR. See nexus/build/nexus_defs.inc for where B_REFSW_DEBUG=n is translates to the other build variables. See magnum/basemodules/dbg/bdbg.inc for the various options.

- To get absolute minimum size, you can also remove error messages. It is nearly impossible to fix problems with this setting, so only use this after system debug is complete.

  ```
  export B_REFSW_DEBUG=no_error_messages
  ```

- Strip the shared library or statically-linked application. For example:

  ```
  mipsel-linux-strip --strip-all libnexus.so

  mipsel-linux-strip --strip-all nexus_app
  ```

- Link statically with Nexus. Nexus will compile and link the code to remove unused symbols[5]. This only works in user mode and requires multi-process to be turned off (otherwise every Nexus is automatically called because of the proxy.) This can be done with:

  ```
  export B_REFSW_SHAREABLE=n
  export NEXUS_SERVER_SUPPORT=n
  ```

  Remember to delete nexus/bin/libnexus.so so that the linker will select nexus/bin/libnexus.a.

[5] That is, CFLAGS = -fdata-sections -ffunction-sections and LDFLAGS = -Wl,--gc-sections

Step 7: Write application for minimal memory usage

After configuring Nexus platform code, there is much your application should do to minimize memory usage.

The following API's allocate the most memory from Nexus heaps:

- NEXUS_Playpump_Open – allocates playback FIFO
- NEXUS_Recpump_Open – allocates CDB/ITB FIFO for record
- NEXUS_VideoDecoder_Open – allocates CDB/ITB FIFO for decode
- NEXUS_AudioDecoder_Open – allocates CDB/ITB FIFO for decode
- NEXUS_Message_Open – allocates message capture buffer
- NEXUS_Surface_Create – allocates graphics memory

Other API's include NEXUS_Dma_Open, NEXUS_PictureDecoder_Open, NEXUS_Graphics2D_Open and more.

For each API, there is a default buffer size. Examine those sizes and determine if they are correct for you. They usually depend on maximum bit rate and system latency.

If you do not use features at the same time, consider using NEXUS_Memory_Allocate, then passing in a user-allocated buffer pointer to each interface when it is used.

Consider using smaller graphics framebuffers and upscaling with the GFD.

Init-time only heap allocations

MVC codec, 4K resolution and 10 bit color depth video are features which consume large amounts of heap memory. For silicon that supports these features, our default heap configuration at init-time includes space for all of these features. However, we don't propagate that init-time configuration to run-time settings by default for MVC or 4K. The application must enable MVC and 4K at run-time as well. 10 bit is enabled at runtime if the init time memory is present.

MVC is enabled at run-time by setting NEXUS_VideoDecoderSettings.supportedCodecs[NEXUS_VideoCodec_eH264_Mvc] = true. See nexus/examples/video/playback_mvc.c for an example.

4K is enabled at run-time by setting NEXUS_VideoDecoderSettings.maxWidth = 3840 and .maxHeight = 2160. See nexus/examples/dvr/playback_4k2k.c for an example.

10 bit can be enabled or disabled at run-time by setting NEXUS_VideoDecoderSettings.colorDepth = 8 or 10.

Linux bmem carve-out

Nexus gets its device memory from Linux using the "bmem" boot parameter. "bmem" carves out memory from Linux so that Nexus has exclusive use of it.

On ARM-based platforms, Linux boots with a large default "bmem" carve-out so that most Nexus configurations will run. However, it does this by minimizing Linux memory (for instance, only 256 MB) and giving everything else to bmem. If you are running large applications, you will need to set custom bmem boot parameters to minimize Nexus memory and maximize Linux memory.

Recommended Boot Parameters

When Nexus starts, it calculates its heap sizes and compares with current bmem. Any unused bmem space is wasted, so Nexus will print the following:

```
current Linux boot parameters: 'bmem=768m@256m bmem=1024m@1024m
bmem=1024m@2048m'
unused 1291MBytes of BMEM memory (used 1524MBytes)
recommended Linux boot parameters: 'vmalloc=528m bmem=544m@480m
bmem=352m@1696m bmem=663m@2409m'
```

If you copy and paste the recommended Linux boot parameters and reboot Linux, you'll see:

```
current Linux boot parameters: 'bmem=544m@480m bmem=352m@1696m
bmem=663m@2409m'
```

This means there is no unused bmem memory and therefore no new recommended boot parameters. You are running with an optimal memory configuration.

If Nexus cannot find enough memory to allocate its heaps, NEXUS_Platform_Init will fail. The failure will look like:

```
# nexus playback videos/cnnticker.mpg
[   23.273179] BCMDRV: Cleanup_modules...
[   23.277103] BCMDRV: Cleanup complete
[   23.293761] wakeup_drv: Cleanup wakeup driver
[   23.298189] wakeup_drv: Cleanup complete
[   23.307407] BCMDRV: Initializing bcmdriver version $ 4
[   23.312596] BCMDRV: Total intc words=4,Total Irqs=129
[   23.318031] BCMDRV: Initialization complete
[   23.327494] wakeup_drv: Initializing wakeup driver
[   23.332434] wakeup_drv: Initialization complete
    00:00:00.001 nexus_platform_os: current Linux boot parameters:
'bmem=100m@480m bmem=352m@1696m bmem=663m@2409m'
```

```
*** 00:00:00.000 nexus_platform: 97445 D0 15.4 boxmode 1
*** 00:00:00.003 nexus_platform_os: Can't allocate heap 0 (218103808
bytes) on MEMC0
    00:00:00.003 nexus_platform_os: region[0] MEMC0.0 TOTAL:100 MBytes
!!!Error BERR_OUT_OF_DEVICE_MEMORY(0x4) at
nexus/platforms/common/src/linuxuser/nexus_platform_cma.c:1435
    00:00:00.003 nexus_platform_os:
    00:00:00.003 nexus_platform_os:
    00:00:00.003 nexus_platform_os: required Linux boot parameters:
'vmalloc=528m bmem=544m@480m bmem=352m@1696m bmem=663m@2409m'
    00:00:00.003 nexus_platform_os:
    00:00:00.003 nexus_platform_os:
!!!Error BERR_OUT_OF_DEVICE_MEMORY(0x4) at
nexus/platforms/common/src/nexus_platform_core.c:136
!!!Error BERR_OUT_OF_DEVICE_MEMORY(0x4) at
nexus/platforms/common/src/nexus_platform.c:741
### 00:00:00.015 nexus_platform: NEXUS_Platform_Init has failed. The
system is not usable. Handles returned by NEXUS_Platform_GetConfiguration
are likely to be invalid.
```

Inside the error messages are the recommended Linux boot parameters. If you use them, Nexus should start.

When changing box modes or if adjusting device memory requirements with the Nexus memconfig API, it is possible that Nexus picture buffer heap sizes will change. If they decrease, there will be unused bmem memory. If they increase, Nexus may not start. You may need to increase your bmem carve outs to provide more device memory. If you want to run multiple box modes on a single Linux boot, you must use the largest bmem parameters for each MEMC and accept the unused bmem memory for the smaller configurations.

bmem syntax

The syntax of the bmem boot parameter is "bmem=size@physical_address". For example, the following reserves 256 MB starting at physical address 512M (aka 0x20000000) on an ARM system:

```
boot flash0.kernel: 'bmem=256M@512M'
```

MEMC0, MEMC1 and MEMC2 memory is differentiated by the physical address. The following reserves one bmem region on each MEMC because MEMC0 starts at 0, MEMC1 starts at 1024M (0x40000000) and MEMC2 starts at 2048M (0x80000000):

```
boot flash0.kernel: 'bmem=544m@480m bmem=352m@1696m bmem=663m@2409m'
```

You can learn the physical address of each MEMC by reading what Nexus prints for its recommended Linux boot parameters.

bmem statusAfter Linux starts, you can learn what memory Linux has reserved for bmem regions through sysfs. For example:

```
cat /sys/devices/platform/brcmstb/bmem.*
0x04000000 0x0c000000
0x90000000 0x40000000
```

The first number is a physical base address. The second number is the size. The bmem regions are listed in order. Be aware that bmem.1 does not necessarily mean MEMC1. It simply means the second bmem region and there may be more than one bmem region per MEMC.

Nexus will read these sysfs nodes to learn the bmem memory it has. Nexus will create one or more heaps in each bmem region.

ARM vmalloc syntax

On ARM systems, Nexus will also recommend a vmalloc boot parameter. The vmalloc number tells Linux how much high memory to configure on MEMC0. The remaining memory is low memory. The number is chosen so that the transition from low to high memory does not occur in the middle of bmem space, which creates a virtual memory discontinuity.

If you follow the Nexus recommended Linux boot parameters for both vmalloc and bmem, you should have no discontinuity. If you don't follow them, Nexus may have to places heaps within a lower and upper region of a bmem carve-out, and certain configurations are not possible.

MIPS bmem defaults

On MIPS systems, bmem only works on MEMC0. Also, Linux creates no default bmem region on MEMC0, so Nexus will not run without specifying those MEMC0 bmem boot parameters. To learn what parameters are required, run Nexus, let it fail, then copy and paste the recommended bmem boot parameters into the next Linux boot.

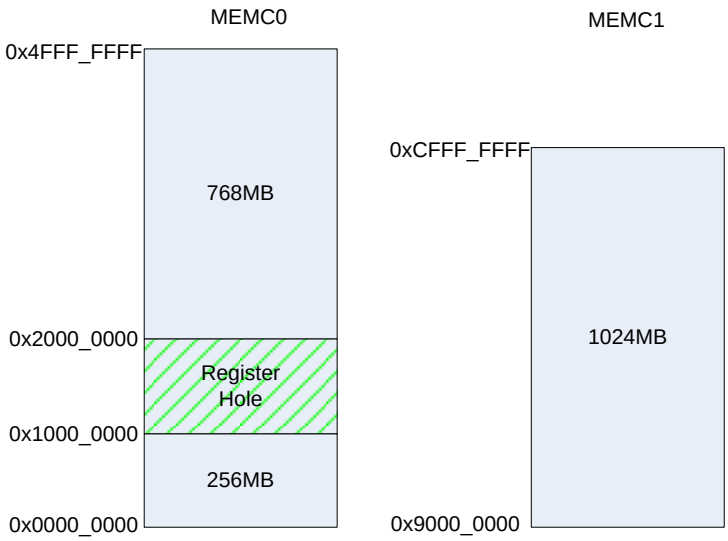
By default, all memory on MEMC1 is reserved for Nexus in a bmem region. If you want Linux to use some memory on MEMC1, you must boot Linux with the following parameter which tells Linux how much memory to use:

```
memc1=256M
```

The remainder will be placed in a bmem region.

MIPS Register Hole

The MIPS physical address map includes a MEMC0 register hole. The following is a MIPS system with two MEMC's with 1 GB of memory each:



On MEMC0, the lower 256 MB starts at physical address 0x0. The upper 768 MB starts at physical address 0x2000_0000 (512 MB), not 0x1000_0000 (256 MB). Like ARM, the base address of MEMC1 memory can vary per chip.

Memory below the register hole is generally called “lower memory” and the memory above the register hole is generally called “upper memory”. Nexus has an algorithm to automatically place heaps in lower and upper memory.[6]

For most MIPS systems with 256 MB or less, only one bmem parameter is needed for lower memory:

```
bmem=192M@64M
```

For most MIPS systems with more than 256 MB, two bmem parameter are needed, one for lower MEMC0 memory, one for upper MEMC0 memory, and none for MEMC1 memory:

```
bmem=192M@64M bmem=512M@512M
```

[6] Prior to Nexus 15.1, users were required to set NEXUS_PlatformSettings.heap[].subIndex = 0 for lower memory and = 1 for upper memory. That has been replaced by the heap placement algorithm.

Pairing older Nexus and Linux versions

Starting with Nexus 15.3 and Linux 3.14-1.8, ARM platforms will use bmem. These two versions should work easily together. However, some customers may not be able to switch Nexus or Linux versions.

This section does not document Linux/Bolt dependencies which may exist. You should check each Linux release for its required Bolt bootloader version.

The following Nexus/Linux pairing options are available:

If you use Nexus 15.1 or newer with Linux 3.14-1.5 or older, and you run with Sage enabled (build with SAGE_SUPPORT=y), you must boot Linux with vmalloc=XXX to avoid an overlap of low memory and Nexus heaps. This overlap could lead to system failure due to ARM speculative fetch into SVP secure regions like the CRR and URR. Starting with Linux 3.14-1.6, Linux will default the vmalloc parameter to a large value which avoids the problem. With Nexus 15.3 and later, we give a recommended vmalloc parameter which also avoids the problem. If you are not running with SVP secure regions, the vmalloc parameter is still highly recommended because it allows bmem and/or CMA regions to not be fragmented and have better memory utilization.

If you run with Nexus 15.2 or older with Linux 3.14-1.8 or newer, Nexus will fail to start. Linux will not create the CMA regions that the older Nexus expects. This can be worked around by booting Linux with brcm_cma=SIZE@OFFSET parameters which explicitly create the CMA regions that Linux created before.

If you run Linux 3.14-1.7, the ARM bmem option is available to you, but is not enabled by default (can be enabled using CONFIG_BRCMSTB_BMEM). The previous CMA regions are still created, but if the user boots with any bmem option, the previous CMA regions will not be created and the bmem parameters will be followed, just like 3.14-1.8.

If you run Linux 3.14-1.6 even with Nexus 15.3, Nexus will still support the old CMA method of obtaining device memory. However, we no longer recommend this approach. CMA allocation is not deterministic; it can fail when other software in the system pins pages. Linux can also become unusable as it tries to proactively relocate pages within the large amount of device memory that Broadcom SoC's require. The bmem approach provides this dedicated device memory without these risks, but at the cost of some boot configuration.

Monitoring Heaps and Allocations

After doing the configuration described above, you must verify your heaps and the allocations that are made. There are four basic methods:

1. Printing heap creation
2. Printing driver heap memory mapping
3. Printing heap status
4. Printing every allocation/free

Printing heap creation

You can see how Nexus creates heaps at init-time by running with:

```
export msg_modules=nexus_platform_settings
```

Output looks like this:

```
os bmem.0 offset 0x4000000, size 201326592
os bmem.1 offset 0x20000000, size 536870912
os bmem.2 offset 0x90000000, size 1073741824
request heap[0]: MEMC0/0, size -1, eFull
request heap[1]: MEMC1/0, size 268435456, eApplication
request heap[2]: MEMC0/1, size 268435456, eApplication
request heap[3]: MEMC1/0, size 31457280, eFull
request heap[4]: MEMC0/1, size 153092096, eDeviceOnly
request heap[5]: MEMC1/0, size 199229440, eDeviceOnly
request heap[6]: MEMC0/1, size 102760448, eFull
creating heap[0]: MEMC0, offset 0x4000000, size 201326592, eFull
creating heap[1]: MEMC1, offset 0x90000000, size 268435456, eApplication
creating heap[2]: MEMC0, offset 0x20000000, size 268435456, eApplication
creating heap[3]: MEMC1, offset 0xa0000000, size 31457280, eFull
creating heap[4]: MEMC0, offset 0x30000000, size 153092096, eDeviceOnly
creating heap[5]: MEMC1, offset 0xa1e00000, size 199229440, eDeviceOnly
creating heap[6]: MEMC0, offset 0x39200000, size 102760448, eFull
```

There are three sections to this output: “os bmem.X” reports what regions Nexus receives from the OS; “request heap[X]” reports what heaps Nexus will try to create. “creating heap[X]” reports what heaps Nexus actually created.

Printing driver heap memory mapping

You can see now Nexus memory maps heaps at init-time by running with:

```
export msg_modules=nexus_memory
```

Output looks like this:

```
NEXUS_Heap_Create 0: MEMC0, offset 0x4000000, length 201326592, addr (nil), cached
0x5c4a2000
nexus_memory: NEXUS_Heap_Create 0: MEMC0, offset 0x4000000, length 201326592, addr
(nil), cached 0x5c4a2000
nexus_memory: NEXUS_Heap_Create 1: MEMC1, offset 0x90000000, length 268435456, addr
(nil), cached 0x4c4a2000
nexus_memory: NEXUS_Heap_Create 2: MEMC0, offset 0x20000000, length 268435456, addr
(nil), cached 0x3c4a2000
nexus_memory: NEXUS_Heap_Create 3: MEMC1, offset 0xa0000000, length 31457280, addr
(nil), cached 0x388a2000
nexus_memory: NEXUS_Heap_Create 4: MEMC0, offset 0x30000000, length 153092096, addr
(nil), cached 0xa0000000
nexus_memory: NEXUS_Heap_Create 5: MEMC1, offset 0xa1e00000, length 199229440, addr
(nil), cached 0xa9200000
nexus_memory: NEXUS_Heap_Create 6: MEMC0, offset 0x39200000, length 102760448, addr
(nil), cached 0x2c4a2000
```

“offset” is the physical address. “addr” is the uncached virtual address, which is no longer supported; cached is the cached virtual address.

Printing heap status

You can also get heap status at runtime with the /proc interface. In user mode, `echo core >/proc/bcmdriver/debug`. In kernel mode, `cat /proc/brcm/core`. Output looks like this:

```
nexus_core: Core:
nexus_core: heap offset     size       mapping used peak largestavail
nexus_core: 0    0x04000000 0x0c000000 eFull    41%  41% 0x0703c6fc
nexus_core: 1    0x90000000 0x10000000 eApp      0%   0% 0x10000000
nexus_core: 2    0x20000000 0x10000000 eApp      6%   6% 0x0f000000
nexus_core: 3    0xa0000000 0x01e00000 eFull     6%   6% 0x01bfdefc
nexus_core: 4    0x30000000 0x09200000 eDevice  86%  86% 0x0144cc80
nexus_core: 5    0xa1e00000 0x0be00000 eDevice  99%  99% 0x0013a900
nexus_core: 6    0x39200000 0x06200000 eFull    74%  74% 0x01934778
```

“used” is the percentage of the heap currently allocated. “peak” is the largest used percentage since Nexus was initialized. “largestavail” is the largest available contiguous free block. A system with memory fragmentation may show a low used percentage, but still a small largestavail.

Your program can also access the heaps by calling NEXUS_Platform_GetConfiguration and using NEXUS_PlatformConfiguration.heap[] and NEXUS_Heap_GetStatus.

Printing every allocation

You can monitor all heap allocations using the DBG interface. Run with:

```
export msg_modules=BMMA_Alloc
```

You will see each allocation and free printed on the console:

```
Alloc:0x2a3a0 size=16384 align=0 handle=0x3a867c
nexus/modules/video_decoder/src/nexus_video_decoder.c:679
Alloc:0x2b7c0 size=5242880 align=256 handle=0xf9c34
magnum/portinginterface/xpt/src/core28nm/bxpt_rave.c:4684
Alloc:0x2a3a0 size=524288 align=128 handle=0x3a870c
magnum/portinginterface/xpt/src/core28nm/bxpt_rave.c:4782
Alloc:0x2a3a0 size=524287 align=256 handle=0x3a882c
magnum/portinginterface/xpt/src/core28nm/bxpt_rave.c:7280
Alloc:0x2b7c0 size=5242880 align=256 handle=0xf9cc4
magnum/portinginterface/xpt/src/core28nm/bxpt_rave.c:4684
Alloc:0x2a3a0 size=524288 align=128 handle=0x3a894c
magnum/portinginterface/xpt/src/core28nm/bxpt_rave.c:4782
Alloc:0x2a3a0 size=524287 align=256 handle=0x3a89dc
magnum/portinginterface/xpt/src/core28nm/bxpt_rave.c:7280
Alloc:0x2b7c0 size=5996416 align=0 handle=0xf9d54
nexus/modules/video_decoder/src/nexus_video_decoder.c:1561
```

You should copy-and-paste this information to a document, then analyze every large allocation. You may need to go to the file and line number of the code to learn what the meaning of the allocation is. It is important that you be able to understand the meaning of the various allocations and correlate them with your requirements.

Debugging Memory Management Failures

Your first encounter with Nexus memory configuration may be trying to figure out what just went wrong. The following are typical problems and recommended actions.

Out of OS Memory

This occurs when malloc or kmalloc fails. When it occurs, any driver or application should try to recover; it should not be fatal. You should call BKNI_DumpMallocs() periodically before the failure to find a memory leak. Nexus and Magnum always use BKNI_Malloc and these allocations are tracked with file and line number.

Your own driver or application may be leaking memory. If you use BKNI_Malloc instead of malloc, you can benefit from this tracking as well. There are also general purpose debuggers that can track this.

Out of Device Memory

This occurs when a heap allocation fails. The heap may have run out of space or you may be allocating from the wrong heap.

BMMA will print the size of the failed allocation and the line number of code. It will also dump all current allocations in the heap. If there's a leak, you will see a repeated file and line number.

Memory Fragmentation

If a heap memory allocation fails, you should compare NEXUS_MemoryStatus.free and NEXUS_MemoryStatus.largestFreeBlock. If you have a large amount free, but the largestFreeBlock is too small, you have memory fragmentation.

Memory fragmentation is a difficult problem because it may only be detected after long use and there are few good solutions once it occurs.

To avoid fragmentation we recommend:

- Avoid calling Close functions after system init time. Instead, Open interfaces at system init time and leave them open.
- If you must close interfaces at run time, consider using NEXUS_Memory_Allocate at init time, then passing in a user-allocated buffer at run time. For example, see NEXUS_SurfaceCreateSettings.pMemory.

- Writing application code to divide the default heaps created by your platform into smaller heaps. Spread your allocations into different heaps based on size, or reallocation characteristics, or whatever else helps you avoid fragmentation.

- In a multi-process system, clients should only access dedicated client heaps. Then, when they exit, all allocations will be freed. A client should not be allowed to create fragmentation problems for the server.

- If you do run out of memory due to fragmentation, write your application to do a graceful shutdown and restart. This will naturally defragment the memory, but there will be an interruption to the user experience. You may want to monitor your heaps and do a pre-emptive restart to avoid running out of memory in an inopportune context that can't easily recover.

Inaccessible Memory

Some failures result from the CPU trying to access memory that has no memory mapping.

Client-side invalid memory – the heap does not have NEXUS_MemoryType_eApplication mapping for client-side CPU access.

- If Nexus detected the error, you will get a clean error message with the line number of code. Inspect the code and see if it makes sense. If Nexus does not, you may just get a segmentation fault. Use a core dump and stack trace to get the same information.
- Recreate the failure using export msg_modules=nexus_platform_settings. Determine if the heap you are using has eApplication memory mapping.
- You may need to modify nexus_platform_$(NEXUS_PLATFORM).c, or you may need to use a different heap.

Server-side invalid memory – the heap does not have NEXUS_MemoryType_eDriver mapping for driver-side CPU access.

- Recreate the failure using export msg_modules=nexus_platform_settings. Determine if the heap you are using has eApplication memory mapping.
- You may need to modify nexus_platform_$(NEXUS_PLATFORM).c, or you may need to use a different heap.

Default Memory Mapping per Platform

The following is total memory, Linux bmem params, and
"msg_modules=nexus_platform_settings" debug output for various reference platforms:

97445

```
544MB on MEMC0, 352MB on MEMC1, 663MB on MEMC2
vmalloc=528m bmem=544m@480m bmem=352m@1696m bmem=663m@2409m

heap[0]: MEMC0, offset 0x010000000, size 218103808, DRIVER APP
heap[1]: MEMC0, offset 0x01d000000, size 176160768, SECURE
heap[2]: MEMC0, offset 0x027800000, size 156303360, MANAGED NOT_MAPPED
heap[4]: MEMC1, offset 0x040000000, size 33554432, SECURE
heap[6]: MEMC1, offset 0x042000000, size 311013376, MANAGED NOT_MAPPED
heap[7]: MEMC1, offset 0x05489b000, size 8388608, DRIVER APP
heap[8]: MEMC2, offset 0x080000000, size 536870912, APP
heap[9]: MEMC2, offset 0x0a0000000, size 152756224, MANAGED NOT_MAPPED
heap[10]: MEMC2, offset 0x0a91ae000, size 5242880, DRIVER APP MANAGED
```

97435

```
1GB on MEMC0, 1 GB on MEMC1
bmem=192M@64M bmem=512M@512M

heap[0]: MEMC0, offset 0x4000000,  size 201326592, eFull
heap[1]: MEMC1, offset 0x90000000, size 268435456, eApplication
heap[2]: MEMC0, offset 0x20000000, size 268435456, eApplication
heap[5]: MEMC1, offset 0xa0000000, size 31457280,  eFull
heap[6]: MEMC0, offset 0x30000000, size 153092096, eDeviceOnly
heap[7]: MEMC1, offset 0xa1e00000, size 199229440, eDeviceOnly
heap[8]: MEMC0, offset 0x39200000, size 102760448, eFull
```

97425

```
1GB on MEMC0, 1 GB on MEMC1
bmem=192M@64M bmem=512M@512M

heap[0]: MEMC0, offset 0x4000000,  size 201326592, eFull
heap[1]: MEMC0, offset 0x20000000, size 325058560, eApplication
heap[2]: MEMC1, offset 0x90000000, size 887095296, eApplication
heap[3]: MEMC1, offset 0xc4e00000, size 4194304,   eFull
heap[4]: MEMC0, offset 0x33600000, size 130023424, eDeviceOnly
heap[5]: MEMC1, offset 0xc5200000, size 182452224, eDeviceOnly
heap[6]: MEMC0, offset 0x3b200000, size 81788928,  eFull
```

Appendix: Terminology

The following terms are used within this document and in Broadcom source code that may not be widely known. Many terms listed here may not be necessary to understand in order to use Nexus, but are included for completeness.

Term	Definition
CMA	Contiguous Memory Allocator. Allows Nexus to share memory with Linux.
Device Memory	Memory allocated from Nexus/Magnum heap. Contrast with operating system memory.
Dynamic Mapping	A dynamic TLB entry must be created for the CPU to access memory. This is the only way to access memory in user mode (mmap system call). It is required for access to high memory in kernel mode (ioremap kernel function). Contrast with fixed mapping.
Fixed mapping	Linux kernel has pre-assigned addresses for cached and/or uncached access to memory. Contrast with dynamic mapping. Same as “zone normal” and “low mem”
High Mem	Memory that requires dynamic mapping in the kernel (ioremap kernel function)
Low Mem	Memory with fixed mapping in the kernel. Same as “zone normal”. **CAUTION:** Low Mem memory can still be **above** the register hole.
MEMC	Memory Controller. In Nexus/Magnum, DDR blocks are typically referred to by the MEMC that controls access to them. For example, MEMC0 is synonymous with DDR0.
MEMC Region	A contiguous physical addressing range within a single MEMC. Nexus refers to MEMC regions as NEXUS_PlatformSettings.heap[].subIndex. The MIPS register hole creates two regions on MEMC0.
Offset	In general, any offset from a base address. However, “offset” is often used synonymously with physical address. This is a valid use only if the “base” is understood to be physical address 0x0. Otherwise the term “physical address” is preferred.
Physical Address	Address used on memory bus to access memory. Not directly usable by the CPU. Contrast with virtual address.
Register hole	A region of physical addresses reserved for register access, not memory access. For MIPS chips, this is physical address 0x1000_0000 – 0x2000_0000 which creates two “MEMC regions” on MEMC0.
Strapping Option	The machine-readable board configuration which tells the OS and Nexus how much memory is available on each MEMC. See nexus_platform_$(NEXUS_PLATFORM).c for code.
Operating System Memory	Memory allocated from operating system. Contrast with device memory.
TLB	Translation Lookaside Buffer. An entry in the page table used for dynamic mapping of virtual address to physical address.
Upper memory	“Zone normal” memory above the register hole **CAUTION:** There is some “Upper Memory” which is “Low Mem”

Virtual Address	An address usable by the CPU to access memory. May be fixed or dynamic mapping. Contrast with physical address.
XKS01	Feature on 40nm silicon which allows the kernel to access a full 1GB without HIGHMEM. See 2.6.37 memory app note.
Zone HighMem	Linux term for memory without fixed mapping. Requires dynamic mapping.
Zone Normal	Linux term for memory with fixed mapping