Qualcomm Technologies International, Ltd.

Optimizing Memory Pool Configuration

Application Note

CS-00402707

June 16, 2017

For additional information or to submit technical questions, go to: https://createpoint.qti.qualcomm.com

Revision history

Revision	Date	Description
1	June 2017	Initial release

Contents

Figures

Tables

1 Optimizing Memory Pools Configuration

Like all embedded systems, the Qualcomm® Bluecore™ audio devices have limited memory resources. An important part of developing an application is ensuring that the use cases run correctly, by guaranteeing that the required resources are available at the correct times. This requires knowledge of how the resources are made available to the application developers.

This knowledge, and the tools provided by Qualcomm, enable you to analyze how the application uses resources, and how to configure the resources optimally.

NOTE: This document assumes you are familiar with the Bluecore audio devices architecture. For more information, see the ADK Audio Sink Application User Guide.

2 Pool Memory Allocator Module

The BlueCore firmware sets aside a fixed area of RAM for dynamic memory allocation. The size of this area is determined when the firmware is compiled. It is managed by the firmware's pmalloc subsystem.

The pmalloc module divides the available memory into several pools of varying sizes. It determines the sizes of the pools and the number of pools at boot time, by reading persistent store keys (PS Keys). Each PS Key holds a list of pairs of values, where each pair represents the pool size and number of pools of this size.

For example, specifying PSKEY_PMALLOC_BASE = 2 5 10 20 16 8 20 2 creates:

- 5 pools of size 2
- 20 pools of size 10
- 8 pools of size 16
- 2 pools of size 20

Table 2-1 lists the PS Keys that define how the available memory is divided.

Table 2-1 PS Keys

PS Key	Description
PSKEY_PMALLOC_BASE	The pools required for basic operation of the Firmware, specified by Qualcomm.
PSKEY_PMALLOC_PER_ACL	The pools required for a single ACL link at HCI level, specified by Qualcomm.
PSKEY_PMALLOC_PER_SCO	The pools required for a single SCO link at HCI level, specified by Qualcomm.
PSKEY_PMALLOC_APP	The pools required by the VM application, specified by the application author. The firmware default for this key is empty.
PSKEY_PMALLOC_EXTRA	Optional pools to fill the remaining RAM space, specified by Qualcomm.

NOTE: The PSKEYS do not set any ownership of the memory pools. For example, the firmware can use elements created by PSKEY_PMALLOC_APP.

Pool size	Available pools at run-time	PSKEY_PMALLOC_BASE	MAX_ACLS			MAX_SCOS			PSKEY_PMALLOC_APP	PSKEY_PMALLOC_EXTRA
			PSKEY_PMALLOC_PER_ACL	PSKEY_PMALLOC_PER_ACL	...	PSKEY_PMALLOC_PER_SCO	PSKEY_PMALLOC_PER_SCO	...		
2	**47**	45	1	1						
4	**109**	51	5	5					47	1
6	**86**	65	3	3					15	
8	**109**	90				2	2		15	
10	**30**	30								
16	**85**	14							71	
20	**20**	20								
28	**32**								32	
32	**37**	27							10	
36	**7**	7								
40	**2**		1	1						
...	**...**	**...**	**...**	**...**		**...**	**...**		**...**	**...**

Figure 2-1 Example Memory division using PMALLOC PSKEYS

At run time, when the firmware or VM makes a request for a given number of words of memory, pmalloc allocates the smallest available pool of a size greater than or equal to the requested number of words.

For example, if pmalloc receives a request for 13 words of memory, and assuming the pools have a size of 2, 10, 16, and 20 words, then a pool of size 16 is allocated unless all such pools are taken, in which case it allocates a pool of size 20.

Pmalloc configures the pools at boot time using the following procedure:

1. Create the pools in PSKEY_PMALLOC_BASE.
   - □ In case of failure, it raises FAULT_PMALLOC_BASE_INSUFFICIENT_SPACE and uses the firmware defaults.
2. Create one set of pools specified in PSKEY_PMALLOC_PER_ACL for each supported ACL link. The number of ACL links is defined in PSKEY_MAX_ACLS.
   - □ In case of failure, it raises FAULT_PMALLOC_ACL_INSUFFICIENT_SPACE.
   - □ For each ACL link, either all the pools specified in PSKEY_PMALLOC_PER_ACL are created, or none.

3. Create one set of pools specified in PSKEY_PMALLOC_PER_SCO for each supported SCO link. The number of SCO links is defined in PSKEY_MAX_SCOS.
   - In case of failure, it raises FAULT_PMALLOC_SCO_INSUFFICIENT_SPACE.
   - For each SCO link, either all the pools specified in PSKEY_PMALLOC_PER_SCO are created, or none.
4. Create the pools specified by PSKEY_PMALLOC_APP.
   - In case of failure, it raises FAULT_PMALLOC_APP_INSUFFICIENT_SPACE and stops processing the PS Key.
5. Create the pools specified in PSKEY_PMALLOC_EXTRA.
   - If at any point there is insufficient space for the remaining pools, pmalloc stops processing the PS Key but does not raise any fault.
6. If there is any remaining space, pmalloc fills it with whole or partial instances of PSKEY_PMALLOC_PER_ACL, but does not emit any faults to indicate insufficient space.

NOTE: For more information, see the *Configuring the Qualcomm® BlueCore™* Technology Memory Allocator Application Note.

3 VM Memory Mapping

The VM application running on the BlueCore firmware does not access the memory allocated by pmalloc directly. The firmware controls the VM application's view of the available RAM, to prevent corruption of sensitive registers or firmware data structures.

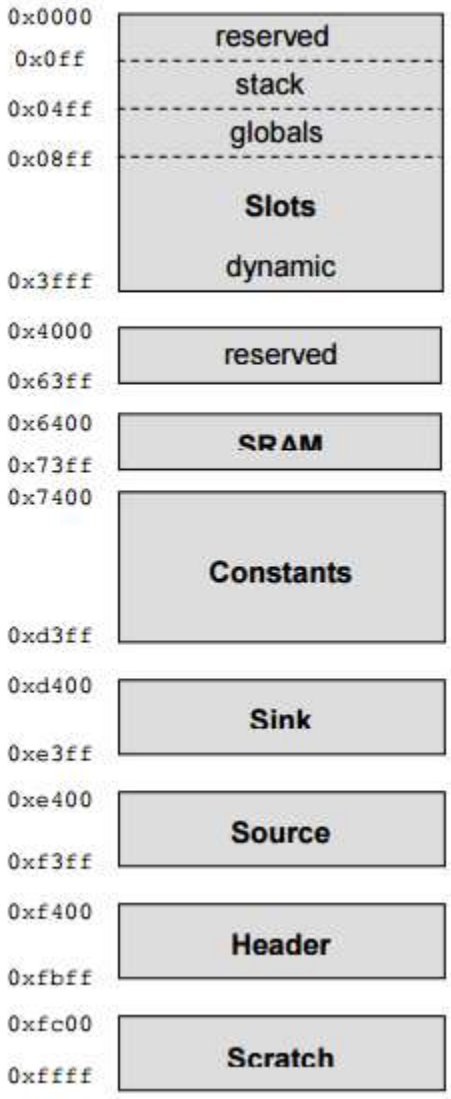


Figure 3-1 VM Memory Map

The **Slots region** divides the address range 0x0000 to 0x3fff in 64 slots that start on 256-word boundaries (such as 0x0000, 0x0100 and 0x0200).

Table 3-1 Slots

Slot	Description
Slot 0	Reserved to trap any NULL pointers. Attempts to access memory from this slot is recognized as an error and the VM panics the application.
Slots 1 to 4	Reserved for the application stack.
Slots 5 to 8	Reserved for storing the global variables.
Slots 9 to 63	Available for use as dynamic memory. However, one slot must always be available to enable messages with a payload to be handled. If no slot is available, then the VM panics the application.

When the application requests a new block of dynamic memory (via malloc, PanicUnlessMalloc, or PanicUnlessNew) the VM attempts to claim a memory block from the firmware pmalloc module. If pmalloc is successful, the VM finds an unused slot and maps the memory block to it.

If the size of the requested allocation is larger than the 256-word boundary, then the firmware attempts to find a multi-slot region to accommodate the size requested.

Although the size of the block allocated by pmalloc may be larger than requested, the VM does not allow attempts to write or read data outside of the requested memory, and panics the application if this happens. For example, if a 16-word buffer is requested and a 20-word pool is allocated by pmalloc, any access outside of the 16-words panics the application.

The **Constants region** maps to ROM/Flash memory and stores constant data that is required to persist for the duration of the program.

The constants region is limited to a maximum of 24K words and is used for:

- Constant global variables
- String literals, such as “Hello world”
- Jump tables for switch statements
- Initializers for non-constant global variables

The linker produces an error message if the VM application exceeds the 24 K limit.

NOTE: For more information, see the BlueCore Memory Mapping and Memory Usage Application Note.

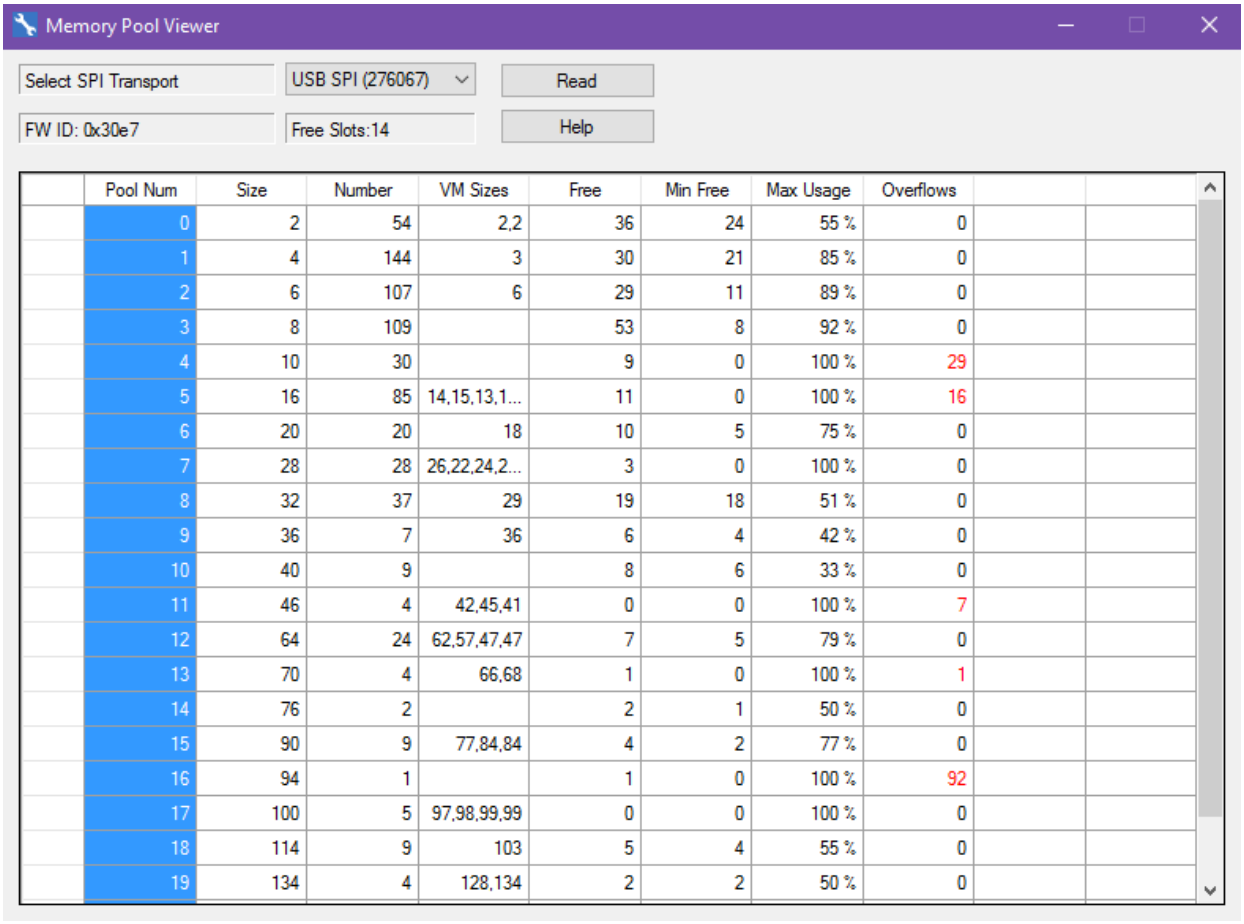
4 Memory Pool Viewer Application

The Memory pool viewer application communicates with the BlueCore firmware to provide run-time information on the pmalloc module.

This application provides the run-time status of the memory pools by showing the following information for each pool:

- Size, in words, of the pool elements
- Number of elements allocated in the pool
- Size, in words or slot number, of each VM allocation
- Number of elements currently free in the pool
- Minimum number of free elements since the last reset
- Maximum number of elements requested from the pool since the last reset
- Number of times the pool has run out of elements

Additionally, the application always displays the number of free slots available to the VM.



Pool Num	Size	Number	VM Sizes	Free	Min Free	Max Usage	Overflows
0	2	54	2,2	36	24	55 %	0
1	4	144	3	30	21	85 %	0
2	6	107	6	29	11	89 %	0
3	8	109		53	8	92 %	0
4	10	30		9	0	100 %	29
5	16	85	14,15,13,1...	11	0	100 %	16
6	20	20	18	10	5	75 %	0
7	28	28	26,22,24,2...	3	0	100 %	0
8	32	37	29	19	18	51 %	0
9	36	7	36	6	4	42 %	0
10	40	9		8	6	33 %	0
11	46	4	42,45,41	0	0	100 %	7
12	64	24	62,57,47,47	7	5	79 %	0
13	70	4	66,68	1	0	100 %	1
14	76	2		2	1	50 %	0
15	90	9	77,84,84	4	2	77 %	0
16	94	1		1	0	100 %	92
17	100	5	97,98,99,99	0	0	100 %	0
18	114	9	103	5	4	55 %	0
19	134	4	128,134	2	2	50 %	0

Figure 4-1 Memory pool viewer run-time example

For more information, see the Memory Pool Viewer Application Note.

5 Tuning the memory pool configuration

A VM application should be optimized to ensure that the memory pool division is sufficient for the scenarios it targets.

Pre-requisites:

- Identify the use cases that the application will target.
- Start the Memory Pool Viewer application and connect to the application.

To tune the memory pool configuration:

1. Initialize the application. The memory pools in use at this point are mostly for configurations and their size is unlikely to change.
2. Compare the VM Sizes column with the pool size to identify wasted memory.
3. Run one of the targeted use cases.
4. Monitor the Max Usage and Overflows columns because they identify pools that should have more elements.
5. Restart your application.
6. Go back to step 4 until all the use cases have been analyzed.
7. Use the gathered data to modify the memory pool configurations (described in Section 5.1) and repeat steps 1 to 6 to verify the improvements.

When reviewing the overflow data, it is important to identify:

- Overflows that trigger more overflows. In Figure 4-1, pool 4 overflows to the pool 5 and then to the pool 6. This means a memory allocation of 9-10 words required 20 words.
- Pools that are wasting memory. In Figure 4-1, pool 12 has two allocations of 47 bytes, wasting 17 words each.
- Overflows that are not problematic. In Figure 4-1, pool 16 only has one element. This means it is likely to overflow often, but the next pool size is not much larger so the memory wasted is a small percentage.

5.1 Changing the memory pool configuration

To maintain system functionality, execute changes to the memory pool with some restrictions.

Pre-requisites:

- Identify the type of adjustment needed in the memory pool configuration. For example:
  - Increase the size of an existing pool
  - Create a new pool

To change the memory pool configuration:

1. Remove the amount of memory you want to add from another pool. There are 4 options from where this memory could be removed:
    a. Reduce the number of SCO links supported: Decrement the PSKEY_MAX_SCOS to free the amount of memory configured in PSKEY_PMALLOC_PER_SCO.
    b. Reduce the number of ACL links supported: Decrement the PSKEY_MAX_ACLS to free the amount of memory configured in PSKEY_PMALLOC_PER_ACL.
    c. Take from PSKEY_PMALLOC_EXTRA.
    d. Take from PSKEY_PMALLOC_APP.
2. Add the total removed in step 1 to PSKEY_PMALLOC_APP by:
    a. Adding elements to existing pools
    b. Creating new pools.

NOTE: Do not create many different pool sizes because the pmalloc module needs to allocate memory for this. When possible, add to the existing pools instead.

Document references

Document	Reference
Configuring the Qualcomm® BlueCore™ Technology Memory Allocator	CS-00128085-AN
VM Memory Mapping and Memory Usage Application Note	CS-00110364-AN
Bluecore Memory Pool Viewer Application Note	CS-00238990-AN

Terms and definitions

Term	Definition
ACL	Asynchronous Connection-oriented
BlueCore	Group term for QTIL's range of Bluetooth wireless technology chips
PS Key	Persistent Store Keys
QTIL	Qualcomm Technologies International, Ltd.
RAM	Random Access Memory
SCO	Synchronous Connection-Oriented
VM	Virtual Machine