AN12600

Memory Pool Optimizer on MKW3xA/KW3xZ

Rev. 0 — 03 March 2020

Application Note

Contents

1 Introduction.. 1

2 Software requirements.......................... 1

3 Hardware requirements......................... 1

4 Memory Manager debug feature........... 1

5 Memory Manager Buffer Pool Optimizer.. 2

6 Buffer Pool Optimizer on Wireless Connectivity Example using IAR Embedded Workbench........................ 2

7 Buffer Pool Optimizer on Wireless Connectivity Example with MCUXpresso..................................... 5

8 Known issues and remedies while using Buffer Pool Optimizer................ 7

9 Conclusion... 9

1 Introduction

This document provides guidance about how to enable the Memory Pool Optimizer on Bluetooth Low Energy (LE) applications to find proper memory pools configuration. The memory pool optimizer is part of Connectivity Framework Memory Manager included in the MKW3xA/KW3xZ SDK packages. The purpose of the optimizer is to help the developer to find the proper number of blocks and sizes of the memory pools allocated at compile time in the application.

2 Software requirements

1. SDK_2.2.1_FRDM-KW36 or later
2. IAR Embedded Workbench or MCUXpresso IDE

3 Hardware requirements

1. FRDM-KW36

4 Memory Manager debug feature

The Memory Manager module incorporates a debug component which allows the possibility to debug, trace, and create memory usage statistics.

To enable the debug support, define one or more of the following macros:

1. MEM_DEBUG_OUT_OF_MEMORY

   When defined, it halts program execution when the memory allocation fails because no suitable memory block was found.

2. MEM_DEBUG_INVALID_POINTERS

   When defined, it halts the program execution if it tries to free an invalid memory address.

3. MEM_TRACKING

   Defining this macro causes the Memory Manager to record the information like memory block address, block size, fragment waste for last allocation for every memory buffer in the memTrack[] array.

4. MEM_DEBUG

   Enabling this macro causes the program execution to halt when an unexpected memory tracking condition occurs, or when the MEM_BufferCheck() function detects a memory block overflow.

5. MEM_STATISTICS

   The scope of this debug component is to allow efficient memory pools configuration. When enabled, the Memory Manager Statistic component records various information like total number of buffers, current and peak number of allocated buffers from memory pool, Number of allocation/free failures for pool for every memory pool

6. DEBUG_ASSERT

This macro enables the MEM_ASSERT for the memory statistics component to catch unexpected conditions.

For more details regarding memory manager module, refer section “3.3.6 Memory manager debug support” from the “Connectivity Framework Reference manual.pdf” available in FRDM-KW36 SDK. The following chapters describe the memory manager buffer pool optimizer which is enabled with some of the macros explained above.

5 Memory Manager Buffer Pool Optimizer

The Connectivity Framework Memory Manager subsystem implements a non-fragmenting memory allocation solution. The subsystem relies on partitions of memory buffer pools. Each partition has a fixed number of partition blocks and each block has a fixed size. The memory management services are implemented using multiple partitions of different sizes. All partitions use memory from a single global array. When a new buffer is requested to be allocated, the subsystem returns the first available partition block of equal or higher size. In other words, if no buffer of the requested size is available, the allocation routine returns a larger buffer. The partitions are defined in the ascending size order with block sizes at multiples of four to ensure the block alignment to 4 bytes:

```
#ifndef PoolsDetails_c
#define PoolsDetails_c \
_block_size_ 64 _number_of_blocks_ 8 _pool_id_(0) _eol_ \
_block_size_ 128 _number_of_blocks_ 4 _pool_id_(0) _eol_ \
_block_size_ 256 _number_of_blocks_ 6 _pool_id_(0) _eol_
#endif
```

For this example, three partitions were created. In addition to the requested amount of memory, each buffer block has a 16 byte header and each defined pool has 20 bytes overhead for internal bookkeeping. The total memory reserved for this example is 2868 bytes.

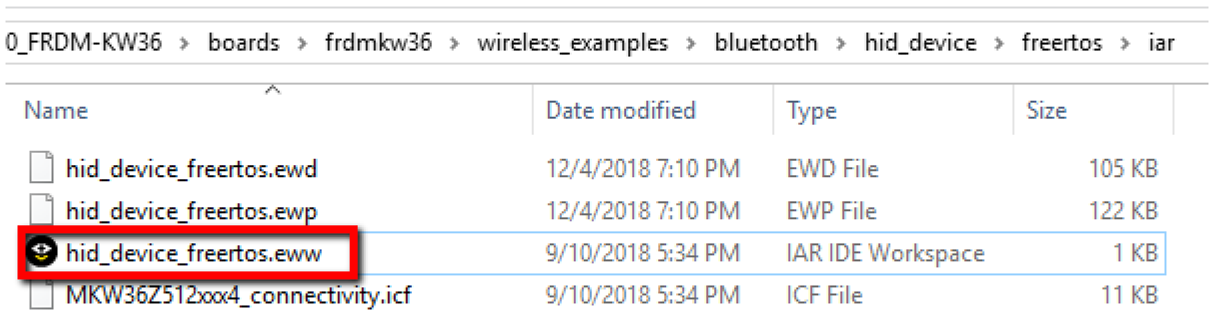
The configuration of the buffer pools requires some insight into the application requirements and a high-level understanding of the usage model for the application. The distribution of buffer allocation sizes and counts can be difficult without some trial and error to find a configuration that works and many times the pool configured is too large for the usage model and ends up wasting RAM.

The Memory Manager buffer pool optimizer profiles the usage of the memory manager buffer pool in real time. It gathers statistics on peak outstanding allocation requests and their sizing on each call to allocate a buffer from the pool. On each allocation request, the optimizer reevaluate what the optimal buffer pool configuration would be to handle the current measured peak utilization. Finally, the optimizer end up with a proposed memory pools configuration that better suits the application needs based on the usage and test scenarios.

6 Buffer Pool Optimizer on Wireless Connectivity Example using IAR Embedded Workbench

The memory buffer pool optimizer is part of the Connectivity Framework Memory Manager; therefore It is available in the wireless connectivity applications. One of the goals of the optimizer is to find a proper memory configuration to make proper usage of the memory available in the device. To enable the optimizer code, follow the below steps:

1. Navigate to the resulting location in the SDK root directory.
2. Open the highlighted IAR workspace file.

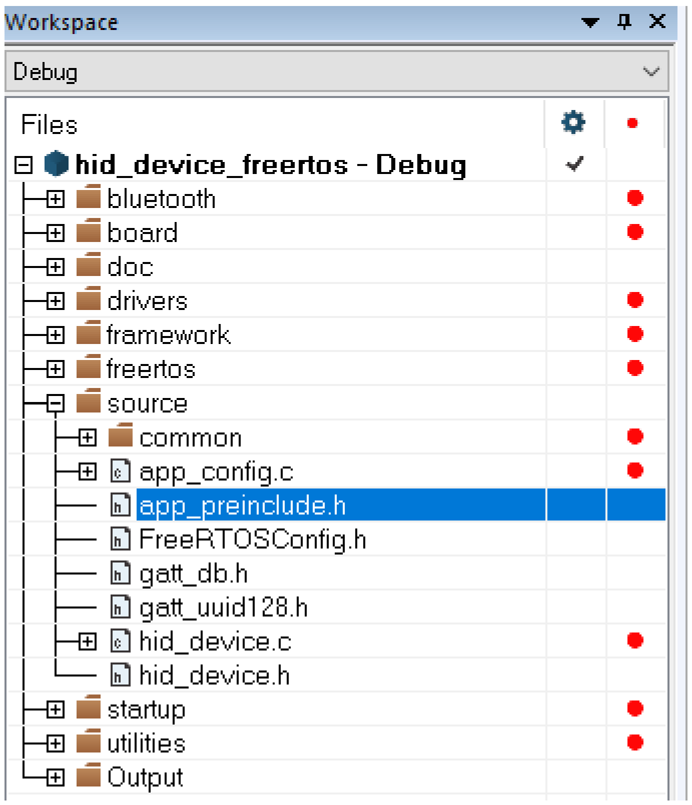


0_FRDM-KW36 › boards › frdmkw36 › wireless_examples › bluetooth › hid_device › freertos › iar

Name	Date modified	Type	Size
hid_device_freertos.ewd	12/4/2018 7:10 PM	EWD File	105 KB
hid_device_freertos.ewp	12/4/2018 7:10 PM	EWP File	122 KB
hid_device_freertos.eww	9/10/2018 5:34 PM	IAR IDE Workspace	1 KB
MKW36Z512xxx4_connectivity.icf	9/10/2018 5:34 PM	ICF File	11 KB

Figure 1. Bluetooth LE IAR workspace file

3. In the workspace of the project, locate the “app_preinclude.h” file.



```
Workspace
Debug
Files
hid_device_freertos - Debug
├ bluetooth
├ board
├ doc
├ drivers
├ framework
├ freertos
├ source
│ ├ common
│ ├ app_config.c
│ ├ app_preinclude.h
│ ├ FreeRTOSConfig.h
│ ├ gatt_db.h
│ ├ gatt_uuid128.h
│ ├ hid_device.c
│ └ hid_device.h
├ startup
├ utilities
└ Output
```

Figure 2. HID Device project IAR workspace

4. Inside the “app_preinclude.h” file, add the following define.

```
#define MEM_TRACKING 1
```

5. Locate the “MemManager.h” file and configure the pool to optimize. The maximum supported buffer size shall be specified in this file. For this example, we will optimize the pool “0” which is the one used by the application, therefore, look for the “POOL_TO_OPTIMIZE” and “MAX_SUPPORTED_BUFFER_SIZE” defines and configure it as shown below:

```
#ifdef MEM_TRACKING
/* Which pool to optimize */
#define POOL_TO_OPTIMIZE 0
/* Maximum buffer size to track */
```

```
#define MAX_SUPPORTED_BUFFER_SIZE 512
#endif /*MEM_OPTIMIZE_BUFFER_POOL*/
```

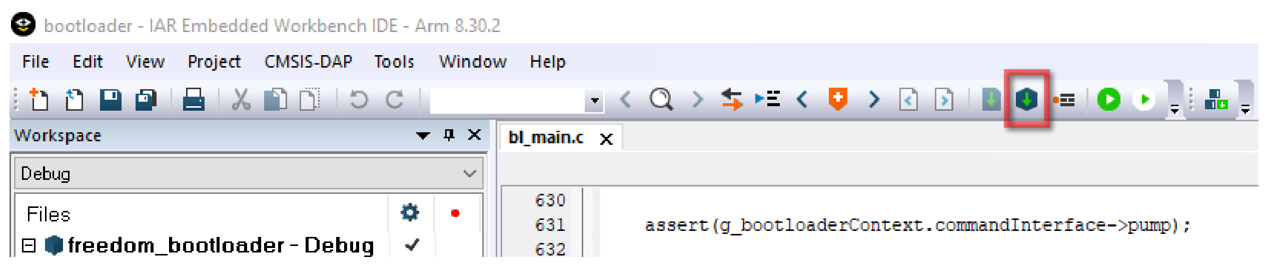
The maximum supported buffer size is obtained from the memory pools defined in the “app_preinclude.h” file. Usually, the maximum buffer size is 512, however, it could be verified in the define “AppPoolsDetails_c” at the “app_preinclude.h” file. For example, in this case, for the HID device project, the following memory pool configuration is set by default:

```
#define AppPoolsDetails_c \
_block_size_ 32 _number_of_blocks_ 6 _eol_ \
_block_size_ 64 _number_of_blocks_ 3 _eol_ \
_block_size_ 128 _number_of_blocks_ 10 _eol_ \
_block_size_ 512 _number_of_blocks_ 4 _eol_
```

NOTE

Before enabling the optimizer, it is required that application reserves enough memory (as many buffers as the applications allow it), so, there is a space for the optimizer to find proper memory pool configuration.

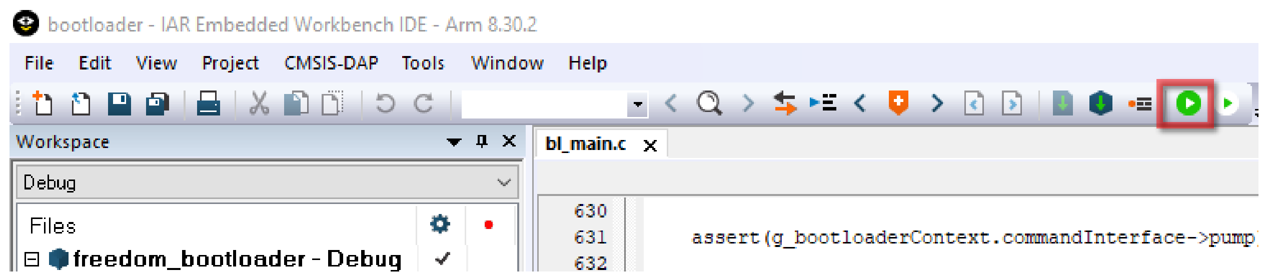
6. At this point, the memory pool optimizer was enabled and configured. Now, compile the project by using Make button.


bootloader - IAR Embedded Workbench IDE - Arm 8.30.2
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
Debug
Files
freedom_bootloader - Debug
bl_main.c

```
assert(g_bootloaderContext.commandInterface->pump);
```

Figure 3. Make button

7. Download the application to the FRDM-KW36 by using “Download and Debug” button.

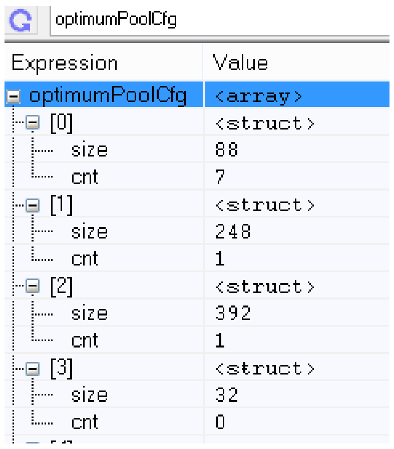

bootloader - IAR Embedded Workbench IDE - Arm 8.30.2
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
Debug
Files
freedom_bootloader - Debug
bl_main.c

```
assert(g_bootloaderContext.commandInterface->pump
```

Figure 4. Download and Debug button

8. Once the application has been downloaded, click “Run” button to start the application. Run the application through various scenarios. The usage should be robust to ensure a good level of data collection over a reasonable amount of time. Stressing the application allows the memory optimizer to collect more data to identify optimum memory pool values.

9. Pause the debugger and look for the “optimumPoolCfg[n]” array using quick watch option to find the suggested pool configuration. The result can be referenced in Figure 5.


optimumPoolCfg

Expression	Value
optimumPoolCfg	<array>
[0]	<struct>
size	88
cnt	7
[1]	<struct>
size	248
cnt	1
[2]	<struct>
size	392
cnt	1
[3]	<struct>
size	32
cnt	0

Figure 5. Optimized memory pool configuration

10. Based on Figure 5, it could be observed the estimated optimized pool configuration for the hid device project as shown below:

```
#define AppPoolsDetails_c \
_block_size_ 88 _number_of_blocks_ 7 _eol_ \
_block_size_ 248 _number_of_blocks_ 1 _eol_ \
_block_size_ 392 _number_of_blocks_ 1 _eol_
```

11. Stop the debug session and replace the default memory pool with the one suggested by the memory pool optimized in the “app_preinclude.h” file.

NOTE

It is better to disable optimizer when configuring optimized memory pool in application; running memory optimizer again on optimized memory pool may result in failures or issues.

12. Compile the project and run it again.

At this point, the memory pool was optimized to the use case of the application. Now, looking at the optimized pool configuration, it can be noticed that the memory decreased saving about 2.5 KB of RAM. However, if there is any modification or new features added to the application, it is recommended to start the process again. It means that application should reserve as much memory as possible, so, the optimizer could identify the peak utilization and provide proper optimized pool configuration.

7 Buffer Pool Optimizer on Wireless Connectivity Example with MCUXpresso

To enable the optimizer code in wireless connectivity example, follow the next steps.

1. Import SDK example (i.e. HID, HRS or any other). Refer to Section 7 - Run a demo using MCUXpresso IDE from the “Getting Started with MCUXpresso SDK for MKW36 Derivatives.pdf” document in SDK.
2. Once project is imported, add following define in “app_preinclude.h” file.

```
#define MEM_TRACKING 1
```

3. Locate the “MemManager.h” file and configure the pool to optimize. The maximum supported buffer size shall be specified in this file. For this example, we will optimize the pool “0” which is the one used by the application, thus, look for the “POOL_TO_OPTIMIZE” and “MAX_SUPPORTED_BUFFER_SIZE” defines and configure it as shown next:

```
#ifdef MEM_TRACKING
/* Which pool to optimize */
#define POOL_TO_OPTIMIZE                0
/* Maximum buffer size to track */
```

```
#define MAX_SUPPORTED_BUFFER_SIZE        512
#endif /*MEM_OPTIMIZE_BUFFER_POOL*/
```

The maximum supported buffer size is obtained from the memory pools defined in the “app_preinclude.h” file. Usually, the maximum buffer size is 512, however, it could be verified in the define “AppPoolsDetails_c” at the “app_preinclude.h” file. For example, in this case, for the HID device project, the following memory pool configuration is set by default:

```
#define AppPoolsDetails_c \
_block_size_  32  _number_of_blocks_    6 _eol_  \
_block_size_  64  _number_of_blocks_    3 _eol_  \
_block_size_ 128  _number_of_blocks_   10 _eol_  \
_block_size_ 512  _number_of_blocks_    4 _eol_
```

4. At this point, the memory pool optimizer was enabled and configured. Now, compile the project with Debug build configuration as shown in Figure 6.

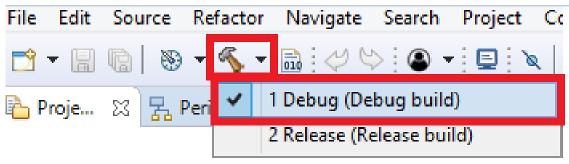

File Edit Source Refactor Navigate Search Project Cc
Proje...
Peri
1 Debug (Debug build)
2 Release (Release build)

Figure 6. Compilation using Debug Build configuration

5. Download the application to the FRDM-KW36 by using “Debug” button.

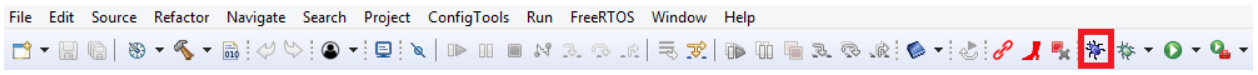

File Edit Source Refactor Navigate Search Project ConfigTools Run FreeRTOS Window Help

Figure 7. Download the application using Debug button

6. Once the application has been downloaded, click on “Resume” button to start the application. Run the application through various scenarios. The usage should be robust to ensure a good level of data collection over a reasonable amount of time. Stressing the application will allow the memory optimizer to collect more data to identify optimum memory pool values.

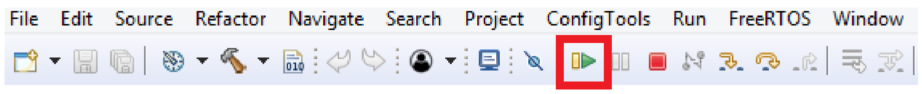
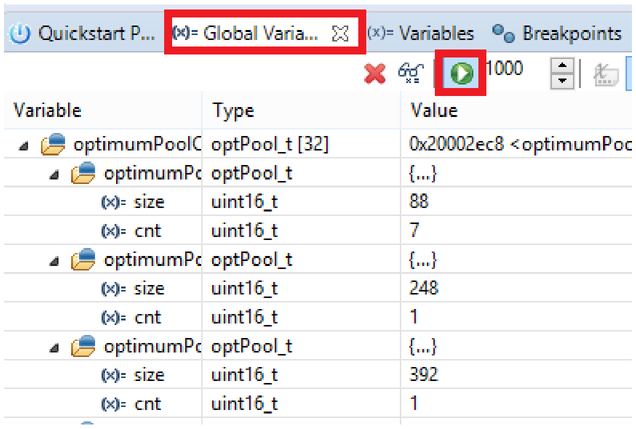

File Edit Source Refactor Navigate Search Project ConfigTools Run FreeRTOS Window

Figure 8. Start the application using Resume Button

7. While running the application add variable “optimumPoolCfg” into Global variables tab and Enable runtime update of variables.



Variable	Type	Value
optimumPoolC	optPool_t [32]	0x20002ec8 <optimumPoc
optimumPc	optPool_t	{...}
size	uint16_t	88
cnt	uint16_t	7
optimumPc	optPool_t	{...}
size	uint16_t	248
cnt	uint16_t	1
optimumPc	optPool_t	{...}
size	uint16_t	392
cnt	uint16_t	1

Figure 9. Optimized memory pool configuration

8. Based on Figure 9, it could be observed the optimized pool configuration for the HID device project is:

```
#define AppPoolsDetails_c \
_block_size_  88   _number_of_blocks_   7 _eol_  \
_block_size_  248  _number_of_blocks_   1 _eol_  \
_block_size_  392  _number_of_blocks_   1 _eol_
```

9. Terminate the debug session and replace the default memory pool with the one suggested by the memory pool optimized in the “app_preinclude.h” file.

NOTE

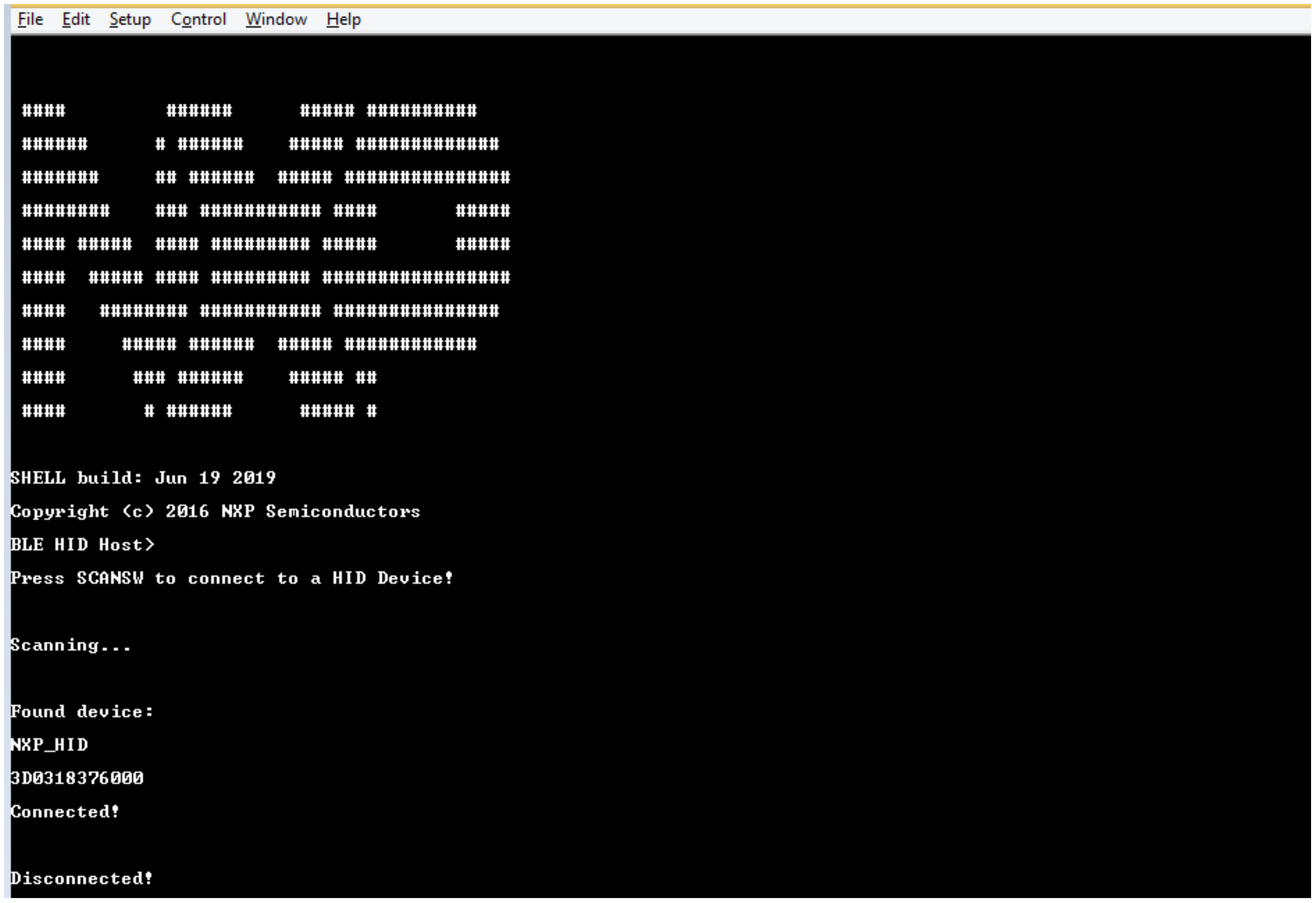
It is better to disable optimizer when configuring optimized memory pool in application, as running memory optimizer again on optimized memory pool may result in failures or issues.

10. Compile the project and run it again.

At this point, the memory pool was optimized to the use case of the application. Now, looking at the optimized pool configuration, it can be noticed that the memory decreased saving about 2.5 KB of RAM. However, if there is any modification or new features added to the application, it is recommended to start the process again. This means that application should reserve as much memory as possible, so, the optimizer could identify the peak utilization and provide proper optimized pool configuration.

8 Known issues and remedies while using Buffer Pool Optimizer

The Buffer Pool Optimizer may produce results that are too optimized and may result in a failure of the application. Refer Figure 10 which shows Disconnection of HID device.



```
####        ######     ##### ##########
######     # ######    ##### #############
#######    ## ######  ##### ##############
########   ### ########### ####      #####
#### #####  #### ######### #####      #####
####  ##### #### ######### ################
####   ######## ########### ##############
####     ##### ######  ##### ############
####      ### ######    ##### ##
####       # ######      ##### #

SHELL build: Jun 19 2019
Copyright (c) 2016 NXP Semiconductors
BLE HID Host>
Press SCANSW to connect to a HID Device!

Scanning...

Found device:
NXP_HID
3D0318376000
Connected!

Disconnected!
```

Figure 10. Disconnection issue after configuring optimized value

To overcome the issues, increase the memory pools. Here are some approaches:

1. Increase the memory pool by 10 % of total memory. In HID application total memory given by optimizer is 1256 so increasing 10 % of total memory is shown below:

```
#define AppPoolsDetails_c \
_block_size_ 88 _number_of_blocks_ 7 _eol_ \
 _block_size_ 128 _number_of_blocks_ 1 _eol_ \
_block_size_ 248 _number_of_blocks_ 1 _eol_ \
_block_size_ 392 _number_of_blocks_ 1 _eol_
```

   Or else we can increase buffer count of buffer size which is greater than 125 by 1 to increase total memory by 10 % or more

```
#define AppPoolsDetails_c \
_block_size_ 88 _number_of_blocks_ 7 _eol_ \
_block_size_ 248 _number_of_blocks_ 2 _eol_ \
_block_size_ 392 _number_of_blocks_ 1 _eol_
```

2. Increase buffer count by 10 %. For HID application increasing 10 % of buffer count is shown below:

```
#define AppPoolsDetails_c \
_block_size_ 88 _number_of_blocks_ 8 _eol_ \
_block_size_ 248 _number_of_blocks_ 2 _eol_ \
_block_size_ 392 _number_of_blocks_ 2 _eol_
```

For increasing 10 % of buffer count, increase buffer count by 1 for count value between 1 to 10, increase buffer count by 2 for count value between 11-20, increase similarly for other decimal range.

3. Increase buffer count by 1 for each buffer size. Increase buffer count by 1 in HID application as shown below:

```
#define AppPoolsDetails_c \
_block_size_  88  _number_of_blocks_    8 _eol_ \
_block_size_  248  _number_of_blocks_   2 _eol_  \
_block_size_  392  _number_of_blocks_   2 _eol_
```

9 Conclusion

The Connectivity Framework Memory Manager allocates buffers based on memory pools defined at compile time in the application. You can use the Buffer Pool Optimizer to identify the number of blocks and sizes of the memory pools based on application needs. This allows you to save some valuable RAM, by reducing the number of unused blocks and block sizes from the initial memory pool configuration.

Date of release: 03 March 2020
Document identifier: AN12600