Tutorial

How to change the RAM size to reduce power consumption in extended sleep mode

For The DA14585/586 Devices

Abstract

This tutorial shows the concept and provides guidelines step by step How to squeeze down the RAM size to 32KB and 48KB for the DA14585/586 product.

Contents

Figures

Tables

Term and definition

BLE Bluetooth Low Energy

OTP One Time Programmable memory

Retention RAM Memory that is powered when DA14585/586 is in Deep sleep

ROM Read Only Memory

RAM Random Access Memory

RW Data Read Write Data

ZI Data Zero Initialized Data

SDK Software development kit

IRQ Interrupt request

DLE Data Length Extension

Scatter file Text file used to specify the memory map of an image to the linker

SUOTA Software-Update-Over-the-air

SysRAM System RAM

1 Introduction

1.1 Overview

The DA14585/586 product has a maximum 96 KB of SysRAM. The source of the SysRAM memory content is usually:

- An executable program that is copied by following the OTP-mirroring OR external Flash-content-copying process, This whole operation might take place after either a system power up or a system wake up procedure is finished.
- A set of BLE (Bluetooth Low Energy) global data such that the values are retained when the system goes to extended sleep.

Since the power consumption in extended sleep mode is dominated by the current drawn by the retained RAM cells the user can consider reducing the memory size for application that do not require the full 96KB of memory.

This tutorial show the concept and provides guidelines step by step How to squeeze down the RAM size to 32KB and 48KB.

Note:

- Retention RAM is the part of the memory that retains its content during Extended sleep mode (without OTP copy)
- In Extended sleep mode the full memory which include code and retention data will be retained.
- In Deep sleep nothing is retained (all RAM blocks off).

Refer to **Section 7** of UM-B-079 Software Platform Reference for further details.

See Also:

For each supported product listed below DA14585 and DA14586, please refer to the following documents available from support web site DA14585 Datasheet and DA14586 Datasheet

1.2 RAM Mapping

The complete RAM mapping of the DA14585/586 is shown in **Figure 1**

- SysRAM1 (Block 1): 0x07FC0000 to 0x07FC7FFF
- SysRAM2 (Block 2): 0x07FC8000 to 0x07FCBFFF
- SysRAM3 (Block 3): 0x07FCC000 to 0x07FCFFFF
- SysRAM4 (Block 4): 0x07FD0000 to 0x07FD7FFF

Note:

- The 4 banks are retained independently and configurable in the extended sleep mode. Each one can be selected to be retained or not.
- The 4th RAM block (32KB) is always retained since it contains the ROM data. The ROM data contains the Bluetooth Stack and Boot ROM code.

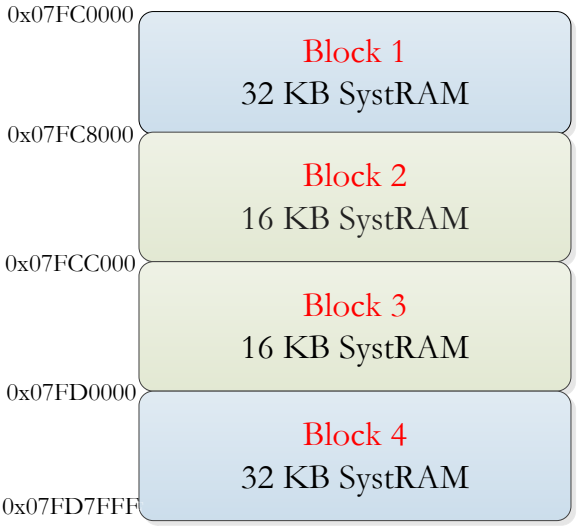

Figure 1: DA14585/586 RAM Block Diagram

Warning:

- The memory map of the DA14585/586 is totally different from the DA14580. For more details please refer to the DA14585/586 SDK 6 Porting Guide

1.3 Memory Size

In KEIL MDK-ARM, the memory size of your application can be known after compiling the whole project.

Note: In this example we took the project: projects\target_apps\ble_examples\prox_reporter

```
Build Output
compiling app.c...
compiling app_task.c...
compiling app_security.c...
compiling app_security_task.c...
compiling app_entry_point.c...
compiling app_msg_utils.c...
compiling app_easy_msg_utils.c...
compiling app_easy_security.c...
compiling app_easy_timer.c...
compiling app_diss.c...
compiling app_diss_task.c...
compiling app_bass.c...
compiling app_bass_task.c...
compiling app_proxr.c...
compiling app_proxr_task.c...
compiling app_suotar.c...
compiling app_bond_db.c...
compiling app_utils.c...
compiling app_suotar_task.c...
compiling user_periph_setup.c...
compiling user_proxr.c...
linking...
Program Size: Code=23090 RO-data=2830 RW-data=4 ZI-data=8840
FromELF: creating hex file...
".\out_585\prox_reporter_585.axf" - 0 Error(s), 1 Warning(s).
Build Time Elapsed:  00:00:18
```

Figure 2: KEIL Compilation using SDK6.0.8 for DA14585/586

- Total code size = 23090 Bytes
- Read Only memory size : 2830 Bytes
- Read Write memory size : 4 Bytes
- Zero-Initialization size : 8840 Bytes
- Total **RAM** size = Read Write memory size + Zero-Initialization size = 4 + 8840 = 8844 bytes
- Total **ROM** size = Code size + Read Only memory size + Read Write memory size = 23090 + 2830 + 4 = 25924 bytes

1.4 RAM layout

The figures below show the 64KB and 96KB RAM layout when main is executed:

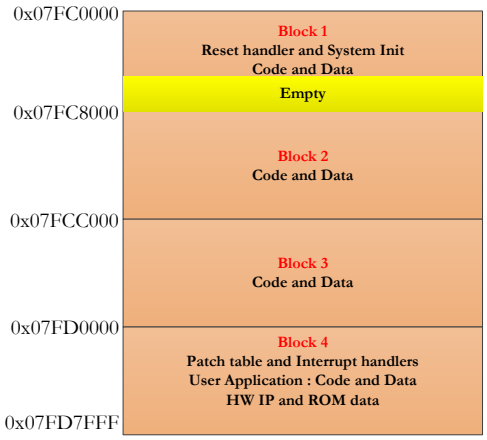

Figure 3 64kB RAM layout: Memory view when main is executed

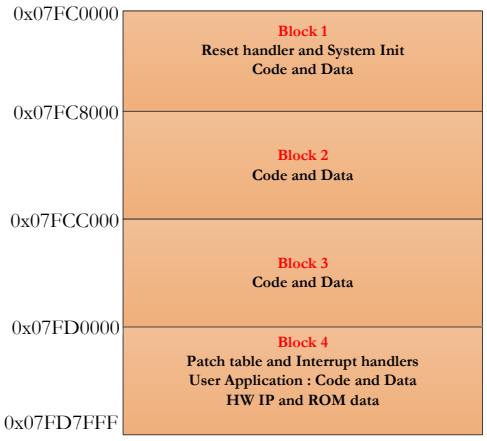

Figure 4 96kB RAM layout: Memory view when main is executed

Note:

- 1KB = 1024 Bytes
- Code is loaded from Block 1 to Block 4 in a top-down approach and the booting address of the SysRAM is located 0x07C0000

2 Concept to achieve 32kB retention RAM

2.1 Objective

- Create an example project where your application is executed using the 32kB RAM (Block 4).
- Only the Block 4 (32kB RAM) is retained.
- How we will do it:
  - Load at 0x7FC0000
  - Execute at 0x7FD0000
  - Rebase 0x7FD0000 at zero.

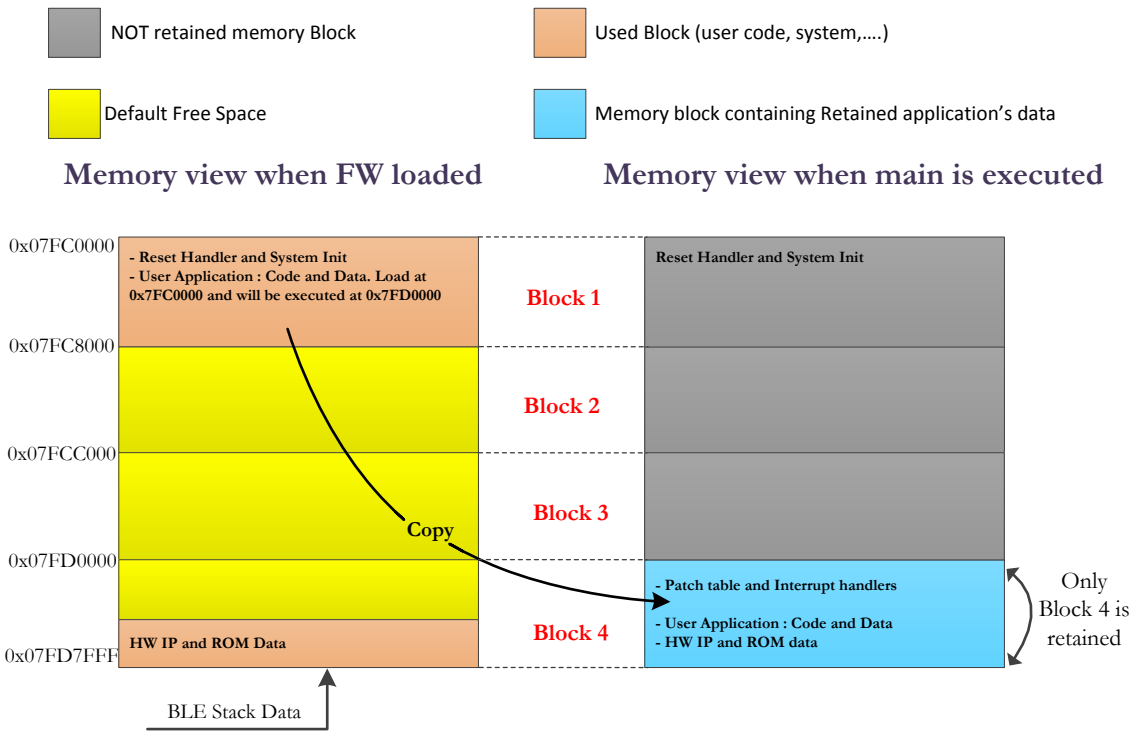

Figure 5: 32kB RAM Memory view

Note:

- The 32kB RAM retained is very useful for small applications with very limited functionalities such as BLE **barebone application**
- The block 4 (32kB RAM) will be used to store the code, patch table, interrupt handler & ROM code.
- The Default boot-rom will directly jumps to SysRAM1 Block 1 (start from 0x07FC0000) that's why the Reset handler and SystemInit are kept in SysRAM1 Block 1.
- The Code for the reset Handler does not need to be retained because during Hardware Reset it is restored from OTP or Flash

Note:

- The Size of the section that contains the HW IP and ROM Data is calculated based on **__SCT_BLE_BASE** macro definition located in :

  sdk\common_project_files\da1458x_scatter_config.h
- During application development the HW IP and ROM Data section in block 4 cannot be used in any way. The total application size which is the maximum size that can be fitted in Block 4 for program execution is given by:

  User Application size = Size of Block 4 - HW IP and ROM Data

3 Steps to achieve 32kB retention RAM

To use only the 32KB Block 4, the following changes are required:

3.1 Copy the patch table in the new address

Create a patch function that copies the contents of 0x07C00C0 array which is 0x80 to 0x07FD00C0. There is a patch table there in the system library:

File location:

projects/target_apps/ble_examples/ble_app_barebone/src/platform/user_periph_setup.c

Code snippet:

```
void my_patch_func( void )
{
  patch_func();

// Function patches start at 0x07fc00c0
  memcpy((uint32_t *)0x07fd00c0, (uint32_t *)0x07fc00c0, 80);
}
```

Replace the **patch_func** call with the **my_patch_func** called in the **periph_init function**

Code snippet:

```
void periph_init(void)
{
   // Power up peripherals' power domain
   SetBits16(PMU_CTRL_REG, PERIPH_SLEEP, 0);
   while (!(GetWord16(SYS_STAT_REG) & PER_IS_UP)) ;

   SetBits16(CLK_16M_REG, XTAL16_BIAS_SH_ENABLE, 1);

   //rom patch
   my_patch_func();

   //Init pads
   set_pad_functions();

   .....
}
```

3.2 Change when the interrupts are enabled

File location: **sdk/platform/arch/boot/rvds/system_ARMCM0.c**

- *Function*: **SystemInit**

Code snippet:

```
void SystemInit (void)
{
....
   //__enable_irq();
}
```

Note: From System Init. The vectors will not be available when this code is executed

3.3 Rebase the Hardfault and NMI Logs

1. *File location:* **sdk/platform/arch/main/nmi_handler.c**

From:

#define STATUS_BASE (0x07FD0050)

To:

#define STATUS_BASE (0x07FC8050)

2. *File location:* **sdk/platform/arch/main/hardfault_handler.c**

From:

#define STATUS_BASE (0x07FD0000)

To:

#define STATUS_BASE (0x07FC8000)

3.4 Rebase 0x7FD0000 at 0x00 and enable interrupts

- File location: **sdk/platform/arch/main/arch_main.c** Add the REMAP function and the IRQ enable as the first commands of the main function.

Code snippet:

```
int main(void)
{
  sleep_mode_t sleep_mode;

  SetBits16(SYS_CTRL_REG, REMAP_ADR0, 3);
  __enable_irq();
  // initialize retention mode
  init_retention_mode();
  ....
}
```

3.5 Rebase the interrupt vectors

- *File location:* **sdk/platform/arch/boot/rvds/boot_vectors.s**
- Create a **Reset Handler** area and move the reset HANDLER and the code that is executed before the **main_func** in there. You can refer to **Figure 6**
- Make a different area for the reset of the Handlers:

```
  .......
  Vectors Size         EQU       Vectors End -   Vectors
; Reset Handler
; DLG Create a Reset Handler area
                AREA    RESET_HANDLER,CODE, READONLY
Reset Handler   PROC
                EXPORT  Reset Handler             [WEAK]
                IMPORT    main
                IMPORT  SystemInit
                LDR     R0, =SystemInit
                BLX     R0
                LDR     R0, =  main
                BX      R0
                ENDP
                AREA    |.text|, CODE, READONLY
 ......
```

boot_vectors.s

```
115                 DCD     RESERVED23_Handler
116 __Vectors_End
117
118 __Vectors_Size          EQU     __Vectors_End - __Vectors
119
120
121 ; Reset Handler
122 ; DLG Create a Reset Handler area
123                 AREA    RESET_HANDLER,CODE, READONLY
124
125 Reset_Handler   PROC
126                 EXPORT  Reset_Handler             [WEAK]
127                 IMPORT  __main
128                 IMPORT  SystemInit
129
130                 LDR     R0, =SystemInit
131                 BLX     R0
132                 LDR     R0, =__main
133                 BX      R0
134                 ENDP
135
136                 AREA    |.text|, CODE, READONLY
137
138 ; Dummy Exception Handlers (infinite loops which can be modified)
139                 IMPORT NMI_HandlerC
140 NMI_Handler\
141                 PROC
142                 movs r0, #4
143                 mov r1, lr
144                 tst r0, r1
145                 beq NMI_stacking_used_MSP
146                 mrs r0, psp
147                 ldr r1,=NMI_HandlerC
148                 bx r1
149 NMI_stacking_used_MSP
150                 mrs r0, msp
151                 ldr r1,=NMI_HandlerC
152                 bx r1
153                 ENDP
154
155                 IMPORT HardFault_HandlerC
156 HardFault_Handler\
157                 PROC
158                 movs r0, #4
159                 mov r1, lr
```

Figure 6: How to Create a Reset Handler area

3.6 Make all the blocks (1,2,3) unretainable

File location: **src/config/da1458x_config_advanced.h**

```
#define CFG_CUSTOM_SCATTER_FILE
#ifdef CFG_CUSTOM_SCATTER_FILE
#undef CFG_RETAIN_RAM_1_BLOCK
#undef CFG_RETAIN_RAM_2_BLOCK
#undef CFG_RETAIN_RAM_3_BLOCK
#endif
```

3.7 Create New Scatter file

The memory map is described in the scatterfile of the Keil project. The scatter file used in a specific project may be found in the Linker Tab of the Project options window.

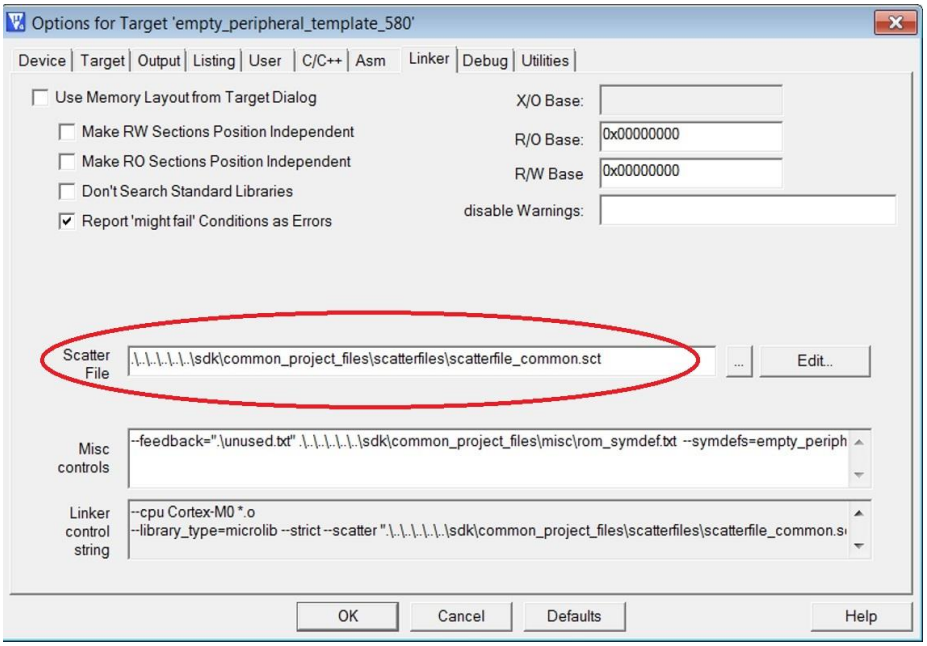

Figure 7: Linker Tab of project options window

The Scatter file can be found at: **sdk/common_project_files/scatterfiles/scatterfile_common.sct**

- Create a scatter file, where :
  - The load area is at 0x7FC0000 and the execution area is 0x7FD0000 and assign it to the project.
  - Move the Reset Handler, the initial Stack Heap and **system_ARMCM0**
  - Change the length of the code accordingly **(EXECUTION_SIZE_1)**

You can refer to **Figure 8**

```
; Definition required by da1458x_scatter_config.h
#define ARM_SCATTERFILE
#include "da1458x config basic.h"
#include "da1458x config advanced.h"
#include "da1458x scatter config.h"
; Macro to align val on the multiple of 4 equal or nearest higher
#define ALIGN4_HI(val) (((val)+3) AND (~3))
#if !defined(CFG_RET_DATA_SIZE)
    #error "CFG RET DATA SIZE is not defined!"
#endif
#if !defined(CFG_RET_DATA_UNINIT_SIZE)
    #error "CFG_RET_DATA_UNINIT_SIZE is not defined!"
#endif
#define  RET_MEM_BASE_ADDR         ALIGN4_HI((__SCT_BLE_BASE  -  (CFG_RET_DATA_UNINIT_SIZE  +
CFG RET DATA SIZE + RET HEAP SIZE)))
#if defined(CFG_CODE_LOCATION_OTP) && defined(CFG_CODE_LOCATION_EXT)
```

```
    #error "Only one of CFG CODE LOCATION OTP and CFG CODE LOCATION EXT must be defined!"
#elif defined(CFG_CODE_LOCATION_OTP)
    #define CODE_LOCATION_OTP   1
    #define CODE_LOCATION_EXT   0
#elif defined(CFG_CODE_LOCATION_EXT)
    #define CODE LOCATION OTP   0
    #define CODE LOCATION EXT   1
#else
    #error "One of CFG_CODE_LOCATION_OTP and CFG_CODE_LOCATION_EXT must be defined!"
#endif
#if defined (CFG TRNG)
    #define TRNG BUFFER AREA SZ CFG TRNG
#else
    #define TRNG_BUFFER_AREA_SZ 0
#endif
#if CODE_LOCATION_OTP
   #define CODE AREA BASE      (0x07fc0000 + 0xC0 + 80)
   #define CODE AREA MAX SIZE  (0xF800 - (0xC0 + 80))
#elif CODE_LOCATION_EXT

   #define CODE AREA BASE  (0x07fc0000 + 0xC0 + 80 + TRNG BUFFER AREA SZ)
   #define CODE AREA MAX SIZE  (0x16800 - (0xC0 + 80 + TRNG BUFFER AREA SZ))
#endif
   #define LOAD_BASE  0x07fc0000
   #define LOAD_SIZE  (0x000)
   #define EXECUTION_BASE_1 (0x07fd0000 + 0xC0 + 80+ TRNG_BUFFER_AREA_SZ)
   #define EXECUTION SIZE 1 (0x8000-(0xC0 + 80 + TRNG BUFFER AREA SZ))
LR IROM1 0x07fc0000 0xc0 {                       ;
   ER IROM1 0x07fd0000 0xc0 {                   ; load address = execution address
      *.o (RESET, +First)
   }
}
LR IROM2 0x07fc00c0 80 {                         ; 20 patch function slots
   ER IROM2 0x07fd00c0 EMPTY 80 {               ; load address = execution address
   }
}
#if CODE_LOCATION_EXT
LR TRNG ZI (0x07fc0000 +0xC0+80) TRNG BUFFER AREA SZ {
   ER TRNG ZI (0x07fd0000 +0xC0+80) TRNG BUFFER AREA SZ {
      /* The TRNG buffer area must be located lower than the 64K boundary. */
      .ANY(trng_buffer_area_zi)
   }
}
#endif
LR IROM3 CODE AREA BASE CODE AREA MAX SIZE {
   ER IROM3 CODE AREA BASE EXECUTION SIZE 1 {
      *(InRoot$$Sections)                        ; All library sections that must be in a
                                  ; root region, for example, __main.o,
                                  ; __scatter*.o, __dc*.o, and * Region$$Table
      boot vectors.o (Reset Handler,   user initial stackheap)
      system ARMCM0.o (+RO)
     }
   ER_IROM31 EXECUTION_BASE_1 EXECUTION_SIZE_1 {
      boot_vectors.o (+RO)
      .ANY (+RO)
      .ANY (+RW)
   }
    ;
*********************************************************************************************
    ; * END OF OTP - ANYTHING BELOW THIS POINT IS NOT WRITTEN WHEN THE CODE IS BURNED TO THE
OTP! *
    ;
*********************************************************************************************
    ER_PRODTEST AlignExpr(+0,8) UNINIT {
        .ANY (prodtest_uninit)
    }
```

```
#if CODE_LOCATION OTP
    ER_TRNG_ZI +0 {
        /* The TRNG buffer area must be located lower than the 64K boundary. */
        /* This execution region starts at most 2K before the 64K boundary. */
        .ANY(trng_buffer_area_zi, +FIRST)
    }
    #endif
    ER_ZI +0 {
        .ANY (+ZI)
        .ANY (STACK)
        jump table.o (heap mem area not ret)    ; not retained HEAP
    }
}
LR_RETAINED_RAM0 RET_MEM_BASE_ADDR {
    RET_DATA_UNINIT RET_MEM_BASE_ADDR UNINIT CFG_RET_DATA_UNINIT_SIZE {
        .ANY (retention_mem_area_uninit)        ; uninitialized application data
    }
    RET DATA +0 CFG RET DATA SIZE {
        .ANY (retention_mem_area0)              ; zero initialized SDK + application data
    }
    RET HEAP +0 RET HEAP SIZE {
        jump table.o (heap env area)
        jump table.o (heap db area)
        jump_table.o (heap_msg_area)
    }
}
```

scatterfile_common.sct

```
    #define CODE_LOCATION_EXT   0
#elif defined(CFG_CODE_LOCATION_EXT)
    #define CODE_LOCATION_OTP   0
    #define CODE_LOCATION_EXT   1
#else
    #error "One of CFG_CODE_LOCATION_OTP and CFG_CODE_LOCATION_EXT must be defined!"
#endif

#if defined (CFG_TRNG)
    #define TRNG_BUFFER_AREA_SZ CFG_TRNG
#else
    #define TRNG_BUFFER_AREA_SZ 0
#endif

#if CODE_LOCATION_OTP

   #define CODE_AREA_BASE      (0x07fc0000 + 0xC0 + 80)
   #define CODE_AREA_MAX_SIZE  (0xF800 - (0xC0 + 80))

#elif CODE_LOCATION_EXT

   #define CODE_AREA_BASE  (0x07fc0000 + 0xC0 + 80 + TRNG_BUFFER_AREA_SZ)
   #define CODE_AREA_MAX_SIZE  (0x16800 - (0xC0 + 80 + TRNG_BUFFER_AREA_SZ))

#endif

   #define LOAD_BASE  0x07fc0000
   #define LOAD_SIZE  (0x000)
   #define EXECUTION_BASE_1 (0x07fd0000 + 0xC0 + 80+ TRNG_BUFFER_AREA_SZ)
   #define EXECUTION_SIZE_1 (0x8000-(0xC0 + 80 + TRNG_BUFFER_AREA_SZ))

LR_IROM1 0x07fc0000 0xc0 {                      ;
  ER_IROM1 0x07fd0000 0xc0 {                    ; load address = execution address
     *.o (RESET, +First)
   }
}

LR_IROM2 0x07fc00c0 80 {                        ; 20 patch function slots
  ER_IROM2 0x07fd00c0 EMPTY 80 {                ; load address = execution address
   }
}

#if CODE_LOCATION_EXT
LR_TRNG_ZI (0x07fc0000 +0xC0+80) TRNG_BUFFER_AREA_SZ {
  ER_TRNG_ZI (0x07fd0000 +0xC0+80) TRNG_BUFFER_AREA_SZ {
     /* The TRNG buffer area must be located lower than the 64K boundary. */
     .ANY(trng_buffer_area_zi)
   }
}
#endif

LR_IROM3 CODE_AREA_BASE CODE_AREA_MAX_SIZE {

  ER_IROM3 CODE_AREA_BASE EXECUTION_SIZE_1 {
     *(InRoot$$Sections)                        ; All library sections that must be in a
                                ; root region, for example, __main.o,
                                ; __scatter*.o, __dc*.o, and * Region$$Table
     boot_vectors.o (Reset_Handler, __user_initial_stackheap)
     system_ARMCM0.o (+RO)
     }


  ER_IROM31 EXECUTION_BASE_1 EXECUTION_SIZE_1 {
     boot_vectors.o (+RO)
     .ANY (+RO)
     .ANY (+RW)
  }

    ; ***************************************************************************************************
    ; * END OF OTP - ANYTHING BELOW THIS POINT IS NOT WRITTEN WHEN THE CODE IS BURNED TO THE OTP! *
    ; ***************************************************************************************************

    ER_PRODTEST AlignExpr(+0,8) UNINIT {
        .ANY (prodtest_uninit)
    }

    #if CODE_LOCATION_OTP
    ER_TRNG_ZI +0 {
        /* The TRNG buffer area must be located lower than the 64K boundary. */
```

Figure 8: How to change the Scatter file for 32KB Memory

4 Concept to achieve 48kB retention RAM

4.1 Objective

- Create an example project where your application is executed using the 48kB RAM.
- Only the Block 3 and 4 are retained.
- How we will do it:
  - Load at 0x7FC0000
  - Execute at 0x7FD0000
  - Rebase 0x7FD0000 at zero.

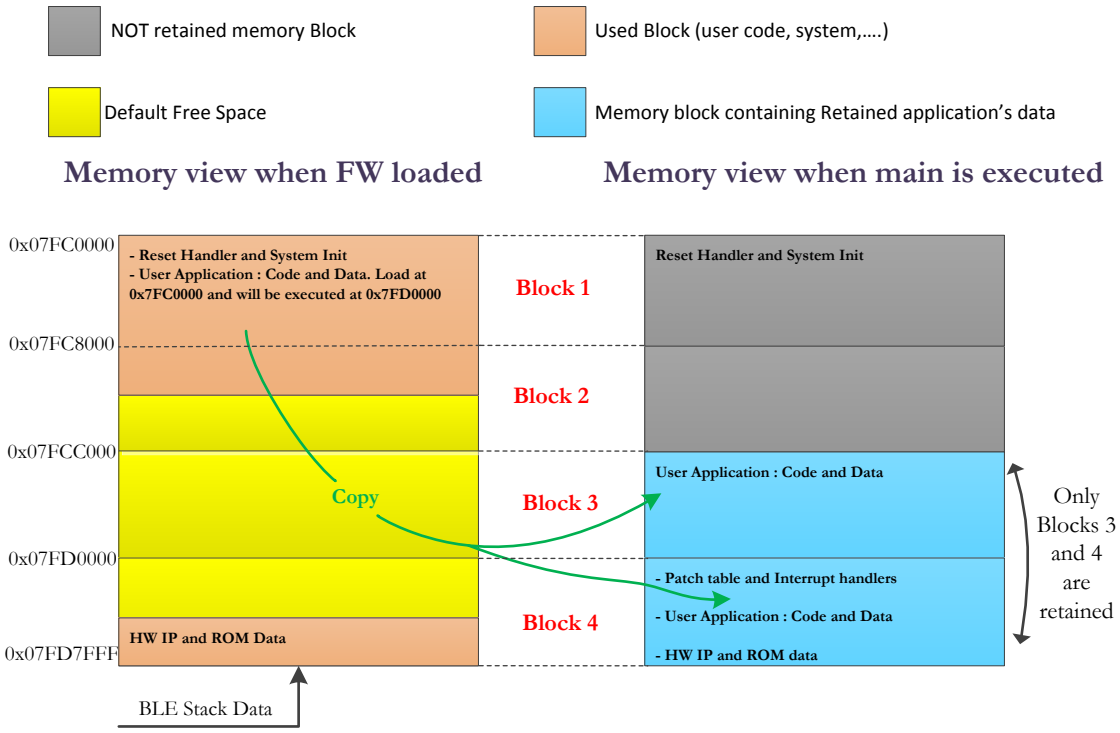

Figure 9: 48kB RAM Memory view

Note:

- The block 4 (32kB RAM) will be used to store the code, patch table, interrupt handler & ROM code.
- The Default boot-rom will directly jumps to SysRAM1 Block 1 (start from 0x07FC0000) that's why the Reset handler and SystemInit are kept in SysRAM1 Block 1.
- The Code for the reset Handler does not need to be retained because during Hardware Reset it is restored from OTP or Flash.

Note:

- The Size of the section that contains the HW IP and ROM Data is calculated based on **__SCT_BLE_BASE** macro definition located in :

  sdk\common_project_files\da1458x_scatter_config.h

- During application development the HW IP and ROM Data section in block 4 cannot be used in any way. The total application size which is the maximum size that can be fitted in Block 4 for program execution is given by:

  User Application size = Size of Block 4 - HW IP and ROM Data

5 Steps to achieve 48kB retention RAM

To use only the 48KB Block 4 + Block 3, the following changes are required:

5.1 Copy the patch table in the new address

Create a patch function that copies the contents of 0x07C00C0 array which is 0x80 to 0x07FD00C0. There is a patch table there in the system library

- *File location:*

  projects/target_apps/ble_examples/ble_app_barebone/src/platform/user_periph_setup.c

Code snippet:

```
void my_patch_func(void)
{
     patch_func();
    // Function patches start at 0x07fc00c0
   memcpy((uint32_t *)0x07fd00c0, (uint32_t *)0x07fc00c0, 80);
}
```

Replace the **patch_func** call with the **my_patch_func** called in the **periph_init function**

Code snippet:

```
void periph_init(void)  // set i2c, spi, uart, uart2 serial clks
{
    // Power up peripherals' power domain
   SetBits16(PMU_CTRL_REG, PERIPH_SLEEP, 0);
   while (!(GetWord16(SYS_STAT_REG) & PER_IS_UP)) ;

   SetBits16(CLK_16M_REG, XTAL16_BIAS_SH_ENABLE, 1);

   //rom patch
   my_patch_func();

   //Init pads
   set_pad_functions();
   .....
}
```

5.2 Change when the interrupts are enabled

- *File location:* **sdk/platform/arch/boot/rvds/system_ARMCM0.c**
- *Function:* **SystemInit**

Code snippet:

```
void SystemInit (void)
{
....
  //__enable_irq();
}
```

Note: From System Init. The vectors will not be available when this code is executed

5.3 Rebase the Hardfault and NMI Logs

1. *File location:* **sdk/platform/arch/main/nmi_handler.c**

From:

#define STATUS_BASE (0x07FD0050)

To:

#define STATUS_BASE (0x07FC8050)

2. *File location:* **sdk/platform/arch/main/hardfault_handler.c**

From:

#define STATUS_BASE (0x07FD0000)

To:

#define STATUS_BASE (0x07FC8000)

5.4 Rebase 0x7FD0000 at 0x00 and enable interrupts

- *File location:* **sdk/platform/arch/main/arch_main.c** Add the REMAP function and the IRQ enable as the first commands of the main function

Code snippet:

```
int main(void)
{
  sleep_mode_t sleep_mode;

  SetBits16(SYS_CTRL_REG, REMAP_ADR0, 3);
  __enable_irq();
  // initialize retention mode
  init_retention_mode();
  ....
}
```

5.5 Rebase the interrupt vectors

- *File location:* **sdk/platform/arch/boot/rvds/boot_vectors.s**
- Create a **Reset Handler** area and move the reset HANDLER and the code that is executed before the **main_func** in there.
- Make a different area for the reset of the Handlers.

5.6 Make all the blocks (1, 2) unretainable

File location: **src/config/da1458x_config_advanced.h**

```
#define CFG_CUSTOM_SCATTER_FILE
#ifdef CFG_CUSTOM_SCATTER_FILE
#undef CFG_RETAIN_RAM_1_BLOCK
#undef CFG_RETAIN_RAM_2_BLOCK
#define CFG_RETAIN_RAM_3_BLOCK
#endif
```

5.7 Create New Scatter file

The Scatter file can be found at: **sdk/common_project_files/scatterfiles/scatterfile_common.sct** you can refer to Linker Tab of project options window.

- Create a scatter file where
  - The load area is at 0x7FC0000 and there are 2 execution areas.
  - The first one **(EXECUTION_BASE_1)** is 0x7FD0000 and there is where you should place the interrupt vector table
  - The second one **(EXECUTION_BASE_2)** is at 0x7FCC000 and can host more data.
- Always execute the Reset Handler, the initial Stack Heap and the **system_ARMCM0** from the initial 0x7FC0000

Change the length of the code accordingly. You can refer to **Figure 10**

```
; Definition required by da1458x_scatter_config.h
#define ARM_SCATTERFILE

#include "da1458x_config_basic.h"
#include "da1458x_config_advanced.h"
#include "da1458x_scatter_config.h"

; Macro to align val on the multiple of 4 equal or nearest higher
#define ALIGN4_HI(val) (((val)+3) AND (~3))

#if !defined(CFG_RET_DATA_SIZE)
    #error "CFG_RET_DATA_SIZE is not defined!"
#endif

#if !defined(CFG_RET_DATA_UNINIT_SIZE)
    #error "CFG_RET_DATA_UNINIT_SIZE is not defined!"
#endif

#define   RET_MEM_BASE_ADDR           ALIGN4_HI((__SCT_BLE_BASE   -   (CFG_RET_DATA_UNINIT_SIZE   +
CFG_RET_DATA_SIZE + RET_HEAP_SIZE)))


#if defined(CFG_CODE_LOCATION_OTP) && defined(CFG_CODE_LOCATION_EXT)
    #error "Only one of CFG_CODE_LOCATION_OTP and CFG_CODE_LOCATION_EXT must be defined!"
#elif defined(CFG_CODE_LOCATION_OTP)
    #define CODE_LOCATION_OTP   1
    #define CODE_LOCATION_EXT   0
#elif defined(CFG_CODE_LOCATION_EXT)
    #define CODE_LOCATION_OTP   0
    #define CODE_LOCATION_EXT   1
#else
    #error "One of CFG_CODE_LOCATION_OTP and CFG_CODE_LOCATION_EXT must be defined!"
#endif

#if defined (CFG_TRNG)
    #define TRNG_BUFFER_AREA_SZ CFG_TRNG
#else
    #define TRNG_BUFFER_AREA_SZ 0
#endif

#if CODE_LOCATION_OTP

   #define CODE_AREA_BASE      (0x07fc0000 + 0xC0 + 80)
   #define CODE_AREA_MAX_SIZE  (0xF800 - (0xC0 + 80))

#elif CODE_LOCATION_EXT

   #define CODE_AREA_BASE  (0x07fc0000 + 0xC0 + 80 + TRNG_BUFFER_AREA_SZ)
   #define CODE_AREA_MAX_SIZE  (0x16800 - (0xC0 + 80 + TRNG_BUFFER_AREA_SZ))


#endif

#define LOAD_BASE  0x07fc0000
#define LOAD_SIZE  (0x000)
#define EXECUTION_BASE_1 (0x07fd0000 + 0xC0 + 80+ TRNG_BUFFER_AREA_SZ)
#define EXECUTION_SIZE_1 (0x8000-(0xC0 + 80 + TRNG_BUFFER_AREA_SZ))
#define EXECUTION_BASE_2 (0x07fCC000)
#define EXECUTION_SIZE_2 (0x4000)

LR_IROM1 0x07fc0000 0xc0 {                      ;
   ER_IROM1 0x07fd0000 0xc0 {                  ; load address = execution address
      *.o (RESET, +First)
   }
}

LR_IROM2 0x07fc00c0 80 {                        ; 20 patch function slots
   ER_IROM2 0x07fd00c0 EMPTY 80 {              ; load address = execution address
   }
}

#if CODE_LOCATION_EXT
LR_TRNG_ZI (0x07fc0000 +0xC0+80) TRNG_BUFFER_AREA_SZ {
   ER_TRNG_ZI (0x07fd0000 +0xC0+80) TRNG_BUFFER_AREA_SZ {
```

```
      /* The TRNG buffer area must be located lower than the 64K boundary. */
      .ANY(trng_buffer_area_zi)
   }
}
#endif

LR_IROM3 CODE_AREA_BASE CODE_AREA_MAX_SIZE {

   ER_IROM3 CODE_AREA_BASE EXECUTION_SIZE_1 {
      *(InRoot$$Sections)                        ; All library sections that must be in a
                                    ; root region, for example, __main.o,
                                    ; __scatter*.o, __dc*.o, and * Region$$Table
       boot_vectors.o (Reset_Handler, __user_initial_stackheap)
      system_ARMCM0.o (+RO)
      .ANY (+RO)
      .ANY (+RW)
   }

      ER_IROM31 EXECUTION_BASE_1 EXECUTION_SIZE_1 {

       boot_vectors.o (+RO)
      .ANY (+RO)
      .ANY (+RW)
   }

   ; ***************************************************************************************************
   ; * END OF OTP - ANYTHING BELOW THIS POINT IS NOT WRITTEN WHEN THE CODE IS BURNED TO THE OTP! *
   ; ***************************************************************************************************

   ER_PRODTEST AlignExpr(+0,8) UNINIT {
      .ANY (prodtest_uninit)
   }

   #if CODE_LOCATION_OTP
   ER_TRNG_ZI +0 {
      /* The TRNG buffer area must be located lower than the 64K boundary. */
      /* This execution region starts at most 2K before the 64K boundary. */
      .ANY(trng_buffer_area_zi, +FIRST)
   }
   #endif

   ER_ZI +0 {

      .ANY (+ZI)
      .ANY (STACK)
      jump_table.o (heap_mem_area_not_ret)     ; not retained HEAP
   }
      ER_IROM32 EXECUTION_BASE_2 EXECUTION_SIZE_2 {
      .ANY (+RO)
      .ANY (+RW)
   }
}

LR_RETAINED_RAM0 RET_MEM_BASE_ADDR {

    RET_DATA_UNINIT RET_MEM_BASE_ADDR UNINIT CFG_RET_DATA_UNINIT_SIZE {
        .ANY (retention_mem_area_uninit)         ; uninitialized application data
    }

    RET_DATA +0 CFG_RET_DATA_SIZE {
        .ANY (retention_mem_area0)               ; zero initialized SDK + application data
    }

    RET_HEAP +0 RET_HEAP_SIZE {
        jump_table.o (heap_env_area)
        jump_table.o (heap_db_area)
        jump_table.o (heap_msg_area)
    }
}
```

scatterfile_common.sct

```
61
62      #define CODE_AREA_BASE   (0x07fc0000 + 0xC0 + 80 + TRNG_BUFFER_AREA_SZ)
63      #define CODE_AREA_MAX_SIZE   (0x16800 - (0xC0 + 80 + TRNG_BUFFER_AREA_SZ))
64
65
66  #endif
67
68  #define LOAD_BASE  0x07fc0000
69  #define LOAD_SIZE  (0x000)
70  #define EXECUTION_BASE_1 (0x07fd0000 + 0xC0 + 80+ TRNG_BUFFER_AREA_SZ)
71  #define EXECUTION_SIZE_1 (0x8000-(0xC0 + 80 + TRNG_BUFFER_AREA_SZ))
72  #define EXECUTION_BASE_2 (0x07fCC000)
73  #define EXECUTION_SIZE_2 (0x4000)
74
75  LR_IROM1 0x07fc0000 0xc0 {                          ;
76      ER_IROM1 0x07fd0000 0xc0 {                      ; load address = execution address
77          *.o (RESET, +First)
78      }
79  }
80
81  LR_IROM2 0x07fc00c0 80 {                            ; 20 patch function slots
82      ER_IROM2 0x07fd00c0 EMPTY 80 {                  ; load address = execution address
83      }
84  }
85
86  #if CODE_LOCATION_EXT
87  LR_TRNG_ZI (0x07fc0000 +0xC0+80) TRNG_BUFFER_AREA_SZ {
88      ER_TRNG_ZI (0x07fd0000 +0xC0+80) TRNG_BUFFER_AREA_SZ {
89          /* The TRNG buffer area must be located lower than the 64K boundary. */
90          .ANY(trng_buffer_area_zi)
91      }
92  }
93  #endif
94
95  LR_IROM3 CODE_AREA_BASE CODE_AREA_MAX_SIZE {
96
97      ER_IROM3 CODE_AREA_BASE EXECUTION_SIZE_1 {
98          *(InRoot$$Sections)                         ; All library sections that must be in a
99                                                      ; root region, for example, __main.o,
100                                                     ; __scatter*.o, __dc*.o, and * Region$$Table
101         boot_vectors.o (Reset_Handler, __user_initial_stackheap)
102         system_ARMCM0.o (+RO)
103         .ANY (+RO)
104         .ANY (+RW)
105     }
106
107         ER_IROM31 EXECUTION_BASE_1 EXECUTION_SIZE_1 {
108
109          boot_vectors.o (+RO)
110         .ANY (+RO)
111         .ANY (+RW)
112     }
113
114
115     ; *************************************************************************************************
116     ; * END OF OTP - ANYTHING BELOW THIS POINT IS NOT WRITTEN WHEN THE CODE IS BURNED TO THE OT
117     ; *************************************************************************************************
118
119     ER_PRODTEST AlignExpr(+0,8) UNINIT {
120       .ANY (prodtest_uninit)
121     }
122
123     #if CODE_LOCATION_OTP
124     ER_TRNG_ZI +0 {
125        /* The TRNG buffer area must be located lower than the 64K boundary. */
126        /* This execution region starts at most 2K before the 64K boundary. */
127        .ANY(trng_buffer_area_zi, +FIRST)
128     }
129     #endif
130
131     ER_ZI +0 {
132
133         .ANY (+ZI)
134         .ANY (STACK)
135         jump_table.o (heap_mem_area_not_ret)    ; not retained HEAP
136     }
137         ER_IROM32 EXECUTION_BASE_2 EXECUTION_SIZE_2 {
138         .ANY (+RO)
139         .ANY (+RW)
140     }
141 }
142
143 LR_RETAINED_RAM0 RET_MEM_BASE_ADDR {
144
145     RET_DATA_UNINIT RET_MEM_BASE_ADDR UNINIT CFG_RET_DATA_UNINIT_SIZE {
146         .ANY (retention_mem_area_uninit)        ; uninitialized application data
```

Figure 10: How to change the Scatter file for 48KB Memory

6 Extended Sleep mode consumption

Now, you have the ability to use your customized scatter file to squeeze RAM memory size. For Sleep mode configurations and power measurement you can refer to this Tutorial. This tutorial gives you details on the firmware, hardware, software/tools environment used to measure the current consumption of different power modes mainly for extended sleep mode.

The DA14585 Datasheet and DA14586 Datasheet show current consumption values in the section **DC Characteristics** for 32 KB RAM retained (48 KB RAM retained are not given in the datasheet) for **25degC** and with **optimized LDO** settings here are **typical Buck application** values for extended sleep current consumption:

Table 1: DC Characteristics

Retained RAM size	Values
32kB (Block 4 Retained)	1.8uA
64kB (Block 1 and 4 Retained)	2.9uA
96kB (Block 1, 2, 3 and 4 Retained)	4uA

Note: Please be aware that many factors can influence the overall power consumption measurements. The results given in the DC Characteristics are done in a specific laboratory characterization conditions.

Note: If you have any question/issue, please contact Dialog Bluetooth Forum.

7 Optimization

How can the memory size be reduced ?

1. If **DLE** is not required set the **CFG_MAX_TX_PACKET_LENGTH** and **CFG_MAX_RX_PACKET_LENGTH** to the minimum value, which is 27:

File location:

projects/target_apps/ble_examples/your_application/src/config/da1458x_config_advanced.h

Check the following:

```
#define CFG_MAX_CONNECTIONS (1)
#define CFG_MAX_TX_PACKET_LENGTH (27)
#define CFG_MAX_RX_PACKET_LENGTH (27)
```

2. **SUOTA** can be removed if not used.
3. Undefine the **CFG_TRNG** flag to compile out the **TRNG** module if it is not required **projects/target_apps/ble_examples/ble_app_barebone/src/config/da1458x_config_advanced.h**
4. Decrease reserved data RAM size by selecting the appropriate size for the retention_mem_area0 data. This size is controlled using the **CFG_RET_DATA_SIZE** flag. Its value defines the **RET_DATA** address, which is the base address of the system's retained data. The selected value depends on the application.
5. Development debug flag can be undefined.

   File location: **projects/target_apps/ble_examples/prox_reporter/src/config**

   Check the following: **#define CFG_DEVELOPMENT_DEBUG**
6. If **UART2** is not required (used only for debugging) then you can undefine the **CFG_PRINTF** flag and reduce further the used RAM size.
7. Check **KEIL MDK-ARM** for how to optimize your code for best performance or smallest code-size. You can refer to this ARM application Note MDK-ARM Compiler Optimizations

8 Conclusions

- The scatter file instructs the Linker where to place code and data. It is comprised of Load Region descriptors and Execution Region descriptors. A Load Region instructs the linker where to place code and data and the initialization code for loading the code and data. Before the code reaches int **main(void)** in **arch_main.c**, the initialization code is executed. During initialization code and data will be copied, if necessary, from the Load Regions to the Execution Regions.
- If the user uses the default SDK scatter file, then the SDK code knows which RAM blocks to retain, depending on the selected DA14585/586 extended sleep mode (without or with OTP copy).
- If the customer uses his own custom scatter file, then he has to define himself the retained RAM blocks.
- RAM 4 block is always retained, due to the fact that it contains the ROM data variables.
- We always advise you to check the map file (linker listing) of your application, when you need to check for resources consumed or you want to optimize your code.
- The map file that is generated by the Linker is useful to analyze the occupied resources. It is advised to enable the generation of this as indicated in Map File. To enable the generation of the Linker MAP file in Keil MDK-ARM, please check the *Memory map* Box in: **Project->Options for Target 'Your target Project name'->Listing Tab->Linker Listing**

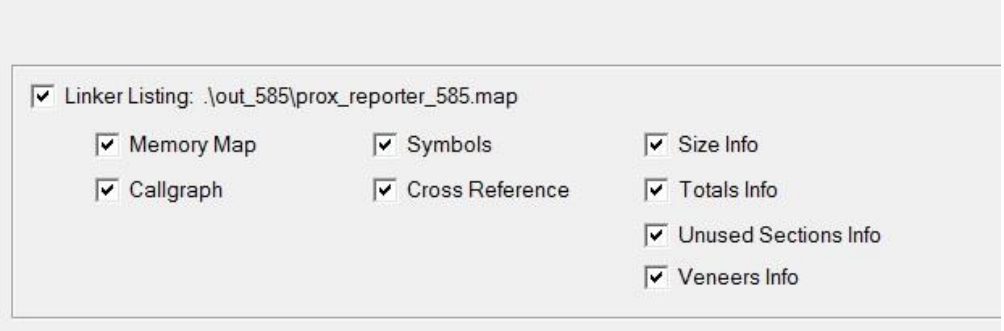

Figure 11: Map File

9 Revision history

Table 2: Revision History

Revision	Date	Description
1.0	17-Aug-2018	First Release

10 Status definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

11 Disclaimer

Information in this document is believed to be accurate and reliable. However, Dialog Semiconductor does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information. Dialog Semiconductor furthermore takes no responsibility whatsoever for the content in this document if provided by any information source outside of Dialog Semiconductor.

Dialog Semiconductor reserves the right to change without notice the information published in this document, including without limitation the specification and the design of the related semiconductor products, software and applications.

Applications, software, and semiconductor products described in this document are for illustrative purposes only. Dialog Semiconductor makes no representation or warranty that such applications, software and semiconductor products will be suitable for the specified use without further testing or modification. Unless otherwise agreed in writing, such testing or modification is the sole responsibility of the customer and Dialog Semiconductor excludes all liability in this respect.

Customer notes that nothing in this document may be construed as a license for customer to use the Dialog Semiconductor products, software and applications referred to in this document. Such license must be separately sought by customer with Dialog Semiconductor.

12 Contacting Dialog Semiconductor

United Kingdom (Headquarters)
Dialog Semiconductor (UK) LTD
Phone: +44 1793 757700

Germany
Dialog Semiconductor GmbH
Phone: +49 7021 805-0

The Netherlands
Dialog Semiconductor B.V.
Phone: +31 73 640 8822

Email:
enquiry@diasemi.com

North America
Dialog Semiconductor Inc.
Phone: +1 408 845 8500

Japan
Dialog Semiconductor K. K.
Phone: +81 3 5769 5100

Taiwan
Dialog Semiconductor Taiwan
Phone: +886 281 786 222

Web site:
www.dialog-semiconductor.com

Hong Kong
Dialog Semiconductor Hong Kong
Phone: +852 2607 4271

Korea
Dialog Semiconductor Korea
Phone: +82 2 3469 8200

China (Shenzhen)
Dialog Semiconductor China
Phone: +86 755 2981 3669

China (Shanghai)
Dialog Semiconductor China
Phone: +86 21 5424 9058

13 RoHS Compliance

Dialog Semiconductor complies to European Directive 2001/95/EC and from 2 January 2013 onwards to European Directive 2011/65/EU concerning Restriction of Hazardous Substances (RoHS/RoHS2). Dialog Semiconductor's statement on RoHS can be found on the DA1458x RoHS 2 declaration. RoHS certificates from our suppliers are available on request.