

## EM9305 IRAM0 REMAP INTO DATA RAM (DRAM)

Product Family: **BLE SOC**

Part Number: EM9305

Keywords: JLI table, IRAM, DRAM extension

### PURPOSE

The purpose of this document is to provide instructions on reclaiming the 4kB RAM section typically used by the JLI (*Jump and Link Indexed*) table. The aim is to repurpose it for general usage, effectively switching it from IRAM (instruction RAM) to DRAM (data RAM).

### SCOPE

The EM9305 embeds seven different blocks of RAM, as follows:

Block	Start address	Length	Type	Retention possible
DRAM0/IRAM0	0x00800000	4kB	Data/Instruction	✓
DRAM1	0x00801000	4kB	Data	✓
DRAM2	0x00802000	4kB	Data	✓
DRAM3	0x00803000	4kB	Data	✓
DRAM4	0x00804000	16kB	Data	✓
DRAM5/IRAM2	0x00808000	16kB	Data/Instruction	✓
DRAM6/IRAM1	0x0080C000	16kB	Data/Instruction	✓

By default, IRAM0 is configured as instructions RAM and hosts the JLI table. On the other end, DRAM5 and DRAM6 are configured as data RAM.

This application note will focus on changing the scope of IRAM0 to data RAM, hereon referred to as DRAM0.

### MEMORY LAYOUT

Figure 1 below shows how the blocks of RAM are mapped to the instruction and data interfaces at hardware level.

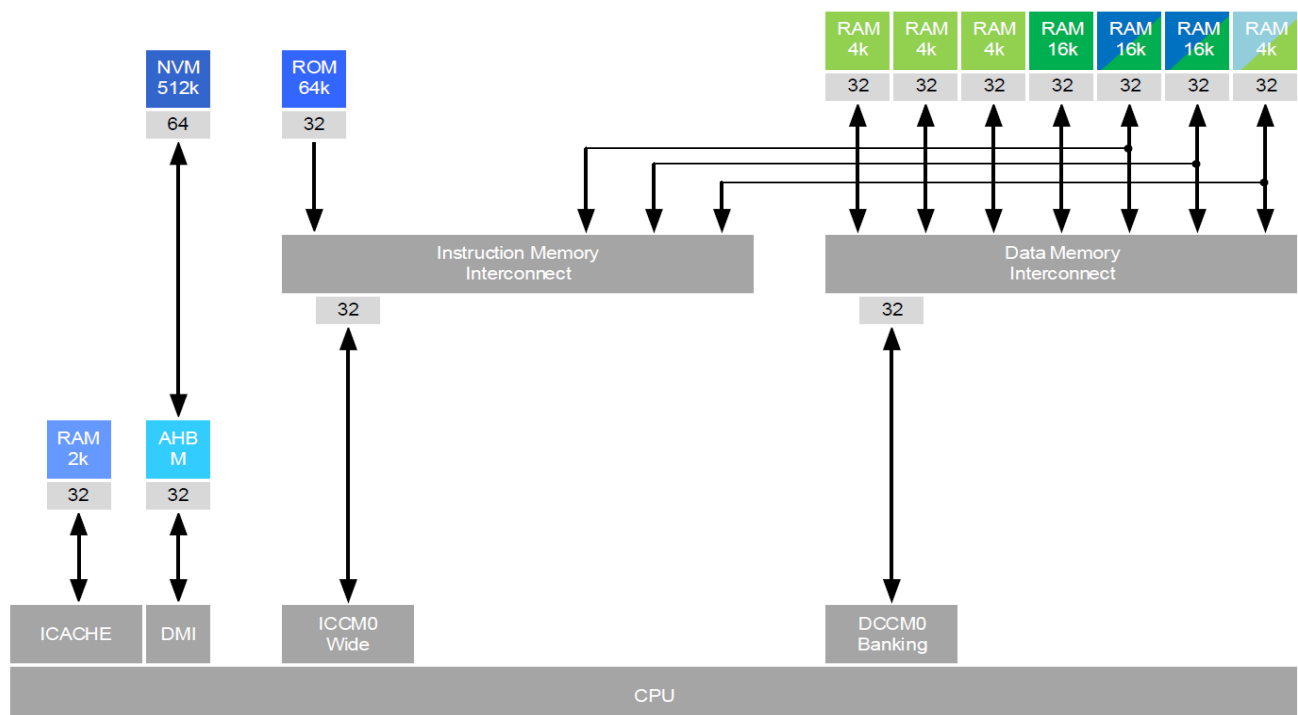


Figure 1: RAM blocks hardware architecture

## 1. MEMORY MANAGER

The EM9305 integrates a memory manager (residing in ROM) which manages the RAM. It configures and allocates RAM blocks depending on how the memory is used by the application, thus providing functionality to dynamically allocate memory from a memory pool composed of the DRAM memories.

It is also used to control the retention of RAM. The persistence option is enabled only once a retention memory allocation is made in that memory. Once the persistence option is enabled for a DRAM, it is not disabled during operation.

However, as seen in Figure 2, which shows the pool used by the memory manger, the DRAM0 block is out of the scope of the memory manager, as the assumption is that block is reserved for storing the JLI table.

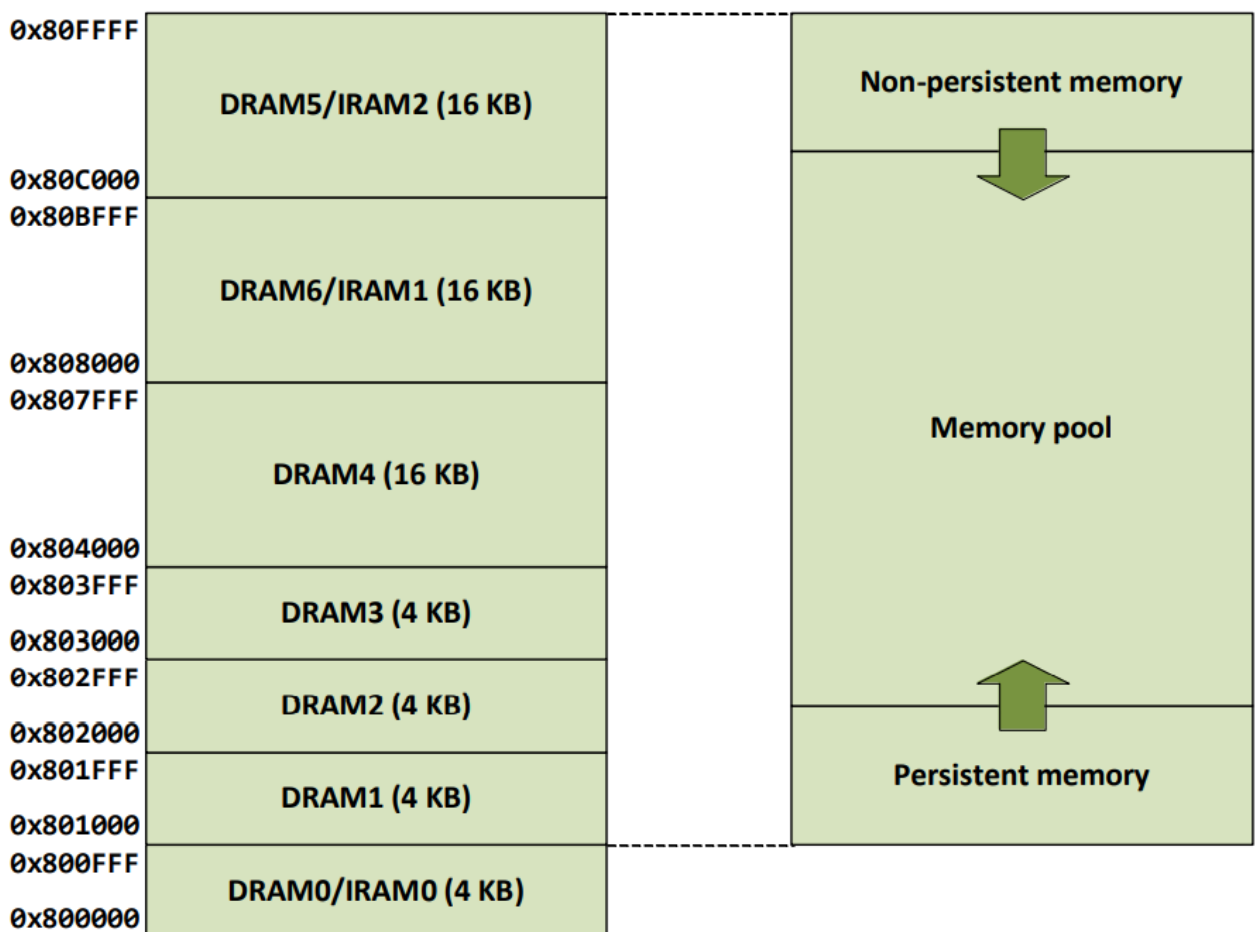


Figure 2: Memory pool for persistent and non persistent allocation

## 2. MOVING DRAM0 TO USABLE SPACE

As the DRAM0 block is not managed by the memory manager because it is configured by default as an instruction block RAM, it needs to be reconfigured by the application itself. In order to do that, the following steps must be followed:

1. A new memory section must be created in the linker script used.
2. This section must then be defined with the **NOLOAD** attribute to allow retention.
3. The application code must define this new section.
4. The application code must modify the system memory configuration register to remap DRAM0 block to the data memory (DCCM) space.
5. The application must explicitly place variables within this new sector.

### 3. LINKER SCRIPT

There are two linker scripts provided in the SDK. The linker script to modify can be:

- `<SDK_ROOT>/common/9305/NVM/linker.cmd` to build an application that does use EM-Core or that is statically linked with EM-Core
- `<SDK_ROOT>/common/9305/APP/linker.cmd` to build an application that uses EM-Core assuming that this piece of software is already stored in the NVM (in this case, no need to statically link the app with EM-Core).

In the section definitions, a new group must be defined (named here *dram0\_persistent*) with the NOLOAD attribute, between the PRAM sections, as shown below:

```
[...]
# Persistent, non-initialized memory.
GROUP: {
    .persistent? NOLOAD: {
        *(.persistent*)
    }
} > PRAM

# DRAM0 persistent block, non-initialized memory.
GROUP: {
    .dram0_persistent? NOLOAD: {
        *(.dram0_persistent*)
    }
} > DRAM0

# Persistent memory.
GROUP: {
    # TLS, thread local storage
    .tls: {
        *(.tls*)
    }
}

[...]
```

**Note:** The NOLOAD attribute is needed to prevent this block from being initialized each time the software starts-up or resumes from sleep (in this case, retention data would be overwritten).

### 4. APPLICATION CODE

The application must now integrate the newly created section. In order to this, the new section must first be defined within the application. This can be done by creating the following definition:

```
/**
 * @brief Define custom memory section DRAM0
 */
#define SECTION_DRAM0 SECTION(".dram0_persistent")
```

Now the new memory is ready to be used. For example, the following lines declare two variables and an array within the DRAM0 block:

```
SECTION_DRAM0 uint8_t a;
SECTION_DRAM0 uint16_t b;
SECTION_DRAM0 uint32_t my_buffer[1024];
```

**Note:** All variables that need to be placed in this section must be explicitly placed there (using the **SECTION\_DRAM0** in this case) and must be globals.

Next, the application must modify the system memory configuration register and configure DRAM0 block as data RAM. This is done in the **NVM\_ConfigModules** function, as shown below:

```
void NVM_ConfigModules(void)
{
    [...]
    if (!PML_DidBootFromSleep())
    {
        SYS->RegMemCfg.r32 = 0;

        // Enable RAM Retention on DRAM0
        // 6:0 PmlDRAMRetOn RW-F 0x00 rst_n DRAM retention enable
        uint8_t reg = PML_GetRamRetentionEnable();

        // Enable RAM Retention for DRAM0, bit 0
        reg |= PML_DRAM_RET_ON_R(0x01);

        PML_SetRamRetentionEnable(reg);
    }
    [...]
}
```

## 5. USING THE EXAMPLE CODE

Along with this document, an example project implementing all the above mentioned changes is provided. In order to compile and test the project, the following steps must be done:

1. The **NO\_JLI\_TABLE** folder containing the **linker.cmd** file must be moved into the **<SDK/common/9305>** folder.
2. The **iram0\_remap\_example** folder containing the source files must be moved into the **<SDK/projects>** folder.
3. The **CMakeLists.txt** file within the **<SDK/projects>** folder must be modified to add the following line:

```
ADD_SUBDIRECTORY(iram0_remap_example)
```


4. The **init.bat** from within the top level SDK folder must be run.
5. The project can then be built by running the following command:

```
cmake --build . --target iram0_remap_example_di03
```

6. And finally, programmed using:

```
python blengine_cli.py --port COMxx run emsystem_prog
..\..\build\projects\iram0_remap_example\iram0_remap_example_di03.ihex -progress
```

7. The UART output can be observed via a terminal software.

 COM5 - PuTTY

```
EM Microelectronic
EM9305
my_buffer address = 0x00800000 RegMemCfg 00000000
RAM retention reg 03
RAM retention success, buffer address 0x00800000 !
```

**Important note:** the example project implements a custom linker script in order to avoid interfering with the standard example projects and functionalities. This linker script is used by adding the following line in the project **CMakeLists.txt**:

```
[...]  
ELSE()  
  APP_IN_NVM()  
  SET(APP_LINKER_FOLDER "NO_JLI_TABLE")  
[...]
```

## 6. CONCLUSION

It is possible to remap IRAM0 to DRAM0, thus freeing up the 4kB of RAM usually assigned to the JLI table. The process to do that is described and an example project is provided. As the memory manager, which is part of the ROM code, does not manage DRAM0, it is the application code that must do that, thus the following considerations and precautions must be taken:

- The application must initialize any and all variables that shall reside in the DRAM0 block by explicitly placing them in that section.
- Due to the above use of the SECTION attribute, the variables can only be global.

EM Microelectronic-Marin SA ("EM") makes no warranties for the use of EM products, other than those expressly contained in EM's applicable General Terms of Sale, located at <http://www.emmicroelectronic.com>. EM assumes no responsibility for any errors which may have crept into this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein.

No licenses to patents or other intellectual property rights of EM are granted in connection with the sale of EM products, neither expressly nor implicitly.

In respect of the intended use of EM products by customer, customer is solely responsible for observing existing patents and other intellectual property rights of third parties and for obtaining, as the case may be, the necessary licenses.

**Important note: The use of EM products as components in medical devices and/or medical applications, including but not limited to, safety and life supporting systems, where malfunction of such EM products might result in damage to and/or injury or death of persons is expressly prohibited, as EM products are neither destined nor qualified for use as components in such medical devices and/or medical applications. The prohibited use of EM products in such medical devices and/or medical applications is exclusively at the risk of the customer.**