EM9305 STANDARD EM-CORE MODEL

Product Family: **BLE SOC**

Part Number: EM9305

Keywords: Emcore standard model

PURPOSE

The purpose of this application note is to introduce the standard EM-Core model.

This model provides standardized bare metal EM-Core software package that includes a subset of low level drivers without any BLE stack integrated.

The main advantage with this model is that the end user application is of a much smaller size because it does need to embeds drivers that are already provided in an installed EM-Core version. Consequently, its size will be smaller which speeds up the process of upgrading to a new image when doing an end user application update over BLE for example. However, it has to be linked with exactly the same version that is stored in the device. To achieve that, a symbol file is provided in the SDK for the link process.

SCOPE

This application note exposes the standard EM-Core model, what it contains, when and why to use it and how to build an application that integrates it.

This package is only applicable to the EM public SDK.

EM-Core standard package

This package is intended to provide the minimum software layers for an application to be able to run on the EM9305 device.

This is particularly true when an end user application does not use the BLE stack. In such a case, there is no need to link the end user application with any BLE stack subpart (link layer and/or host). Only linking with this EM-Core standard package is required.

Memory organization

The Figure 1 shows the NVM organization focusing on the location of the standard EM-Core at the bottom of the NVM.

It is located at the beginning of the first 8kB page just after the page that contains the NVM-Bootloader (which is the first one).

The end of the NVM is the preferred location to contain an optional customer bootloader which by convension has been set to 0x36_0000.

And in between is the memory space to store the main customer application.

Note that the size of the blocks in this figure are not scaled to be representative of their actual size in memory. For that, the addresses on the left hand side shall be considered.

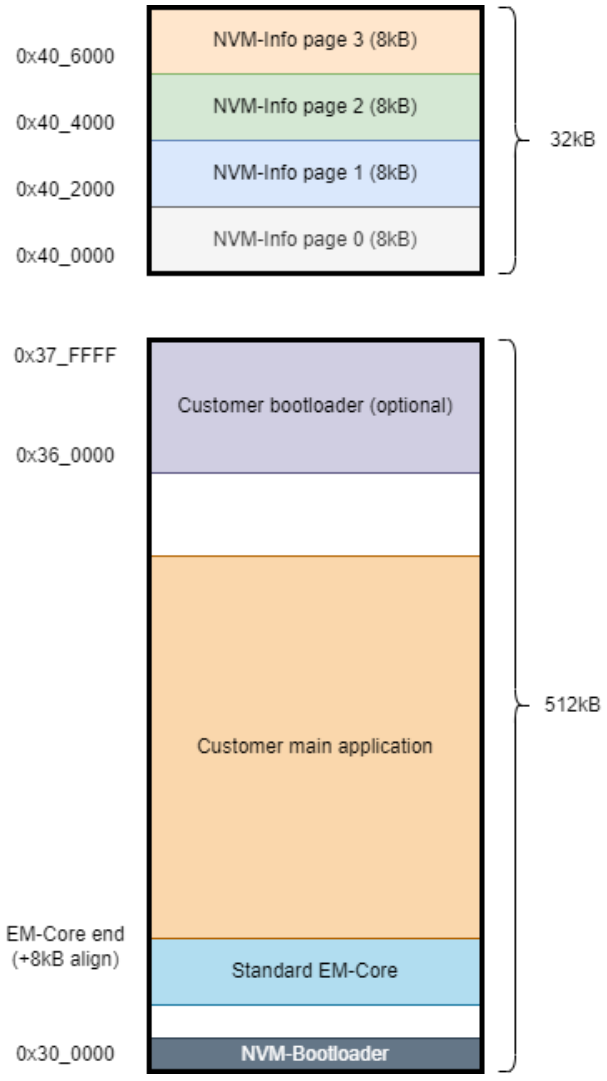

Figure 1: NVM organization

REFERENCE DOCUMENTS

Not applicable

1. PACKAGE CONTENT

The standard EM Core software module contains the following pieces of software:

- EM system libraries
- hardware drivers
- An entry point for the EM-Core application

In addition, an EM Application is included to perform manufacturing diagnostics prior to a customer bootloader or application being installed into the EM9305 NVM.

The standard EM Core Software is statically linked to the starting address of the NVM physical memory. A customer application links to the standard EM Core functions as a static library. Therefore, it is critical that the EM Core linked at build-time MUST match the standard EM Core binary that is programmed into the EM9305 NVM physical memory. Obviously, a mismatch would likely result in corrupted binaries and unexpected behaviour.

**** IMPORTANT ****

The EM-Core module does NOT include all the EM drivers/libraries. For example, the unitimer module is not included in any of the EM-Core modules.

The lists the libraries that are provided in the EM-Core Standard are listed in Table 1 along with a short description and the provider's name.

Library	Notes	Provider
Metaware	Metaware common libraries	Synopsys
em_core	EM-Core standalone application	EM Microelectronic
em_hw_api	Hardware support for EM930x family of SoCs	EM Microelectronic
em_system_stack	Reduced BLE stack for transmitting packets using the radio	EM Microelectronic
em_system_emcore	EM System support for EM-Core (set of EM System Commands)	EM Microelectronic
header_access	API to access firmware images header	EM Microelectronic
nvm_entry_emcore	Entry function in NVM and boot selection	EM Microelectronic
pml	Library for controlling the device power management.	EM Microelectronic
prot_timer	The protocol timer used for BLE operations.	EM Microelectronic
radio	The radio driver abstraction layer.	EM Microelectronic
rc_calib	The RC calibration library	EM Microelectronic
sleep_manager	The sleep manager that manages how and when to enter sleep mode.	EM Microelectronic
sleep_timer	The sleep timer library.	EM Microelectronic
transport	The transport layer abstraction layer on top of SPI or UART.	EM Microelectronic
QP/C	The real time multi-tasking scheduler.	Quantum Leap

Table 1: EM-Core standard package content

The EM Core Application is provided to enable device management and test support for the integration of the EM9305 into a customer manufacturing process. The application provides support to a list of EM commands that are required for a manufacturing flow. EM command routines are in either ROM or NVM memory sections. In addition to the

standard ROM and NVM EM commands, the EM Application also includes a set of diagnostic commands called the EM-Core commands.

When linking with EM-Core standard package, there is no need to explicitly link the application with one of these libraries. It will be automatically done if in the CMakeList.txt file, the macro GENERATE_EMCORE_APPLICATION() is invoked.

The SDK provides application examples that are built against the standard EM-Core and that showcase how to use the macro GENERATE_EMCORE_APPLICATION(). These examples can be found in the projects folder and are nvm_emb_controller and nvm_emb_hrs. Refer to these code examples for more details.

The Table 2 lists the commands supported by the EM-Core application along with their op code..

Command	OpCode
EM System General	
EMSG_ClearBootModeFlags	0xFC55
EMSG_CpuReset	0xFC32
EMSG_EnterConfigurationMode	0xFC4F
EMSG_GetEmCoreInformation	0xFC53
EMSG_ReadProductInformation	0xFC01
EMSG_ReadMACAddress	0xFC52
EMSG_SetBootModeFlags	0xFC54
EMSG_SetSleepOptions	0xFC2D
EM Radio Control	
EMSRC_ReceiverTest	0xFCC5
EMSRC_ReceiverTestEnd	0xFCC6
EMSRC_StartAdvPattern	0xFCC9
EMSRC_TransmitterTest	0xFCC1
EMSRC_TransmitterTestEnd	0xFCC2
EM Security System	
EMSS_LeEncrypt	0xFC88
EMSS_LeEncryptKC	0xFC89
EMSS_LeDecrypt	0xFC8A
EMSS_LeDecryptKC	0xFC8B
EMSS_LeRand	0xFC87
EM Memory Management	
EMSMM_ReadAtAddress	0xFD01
EMSMM_WriteAtAddress	0xFD03

Table 2: EM-Core standard package supported commands

2. WHY AND WHEN TO USE IT?

It is quite usual to build an end user application by linking with the EM-Core standard package since it contains a subset of the low level libraries that are needed for working on the EM9305 chip.

Having an EM-Core standard package allows the end user to create an application which is Bluetooth free when there is no need to have it embedded for some particular cases. The user is then free to add the BLE link layer or the BLE link layer + host depending on what kind of BLE application he wants to create.

On the other hand, there are some specific cases where an end user needs to build a very small application which goal is to achieve one very specific task. In such a case, the end user application only linked with the libraries he needs without linking with the complete EM-Core standard package. This results on a very small specialized application.

3. HOW TO BUILD AN EM-CORE STANDARD PACKAGE LINKED APPLICATION?

Integrating the standard EM-Core standard package has to be done in the CMakeLists.txt file of the application to build.

This special flavour can be selected by adding the following statement:

```
SET(targets_emcore_flavor “standard”)
```

This statement instructs the build system to link the end user application with the EM-Core standard and with all the libraries that are inside. Note that this statement is just the definition of the variable targets_emcore_flavor. It needs then to be used somewhere.

Still in the CMakeLists.txt file, this EM-Core flavour has to be specified for the executable to be built like in the following statement:

```
GENERATE_EMCORE_APPLICATION (
<Project name>
“${targets_emcore_flavor}”
<project source files>
<project include files>
)
```

Here is an example of how to add the standard EM-Core to an application that does not use the BLE stack:

```
GENERATE_EMCORE_APPLICATION (${PROJECT_NAME}_myapp
“${targets_emcore_flavor}”
“${ ${PROJECT_NAME}_SRSCS}”
“${ ${PROJECT_NAME}_INCLUDES}”
)
```

If this example, the ${PROJECT_NAME} is a CMake variable that contains the project name, up to the end user. Then it becomes possible to define the list of source files stored into the ${PROJECT_NAME}_SRCS variable and the lists of include files stored in ${PROJECT_NAME}_INCLUDES variable.