The booting strategy in emBODY for ARM

This document describes the booting strategy in emBODY for ARM.

Approval History

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Version | Author | | Date | Approved | | Date |
| 0.1 | Accame | IIT - iCub Facility |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 0.1 | 21 Mar 12 | Accame | First WIP version. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 Introduction 1](#_Toc321404576)

[2 Partitioning of the MPU 1](#_Toc321404577)

[2.1 The eLoader 2](#_Toc321404578)

[2.2 The eUpdater 5](#_Toc321404579)

[2.3 The eApplication 8](#_Toc321404580)

[2.3.1 A sample application 9](#_Toc321404581)

[2.3.2 A special application: the EOMeApplMaintainer 9](#_Toc321404582)

[2.4 The Shared Storage 9](#_Toc321404583)

# Introduction

This document describes the booting strategy in emBODY for ARM used in boards such as the EMS001.

The system in emBODY is formed by at least three e-processes: the eLoader, the eUpdater, and the eApplication. Other e-processes of the application kind are possible, depending on the size of the FLASH.

At start-up the MPU executes the eLoader which typically launches the eUpdater and later on the eApplication.

Further details in the following.

# Partitioning of the MPU

The FLASH, the RAM and the EEPROM of the ARM are allocated as it follows.

The sizes of the sections and their positions are contained in the file eEmemorymap.h.

MAPPING OF FLASH ON ARM USING EMBODY

eLoader

eUpdater

eApplication

It is executed at bootstrap and its only responsibility is to launch another e-process. If it cannot load any e-process, it enters in idle mode. In case of any HW error it enters in fatal error mode.

It is the e-process that the loader launches when electrical power is switched on. It opens a listening socket and if not contacted by a host it ask the eLoader to launch the default e-process. Otherwise it enters in maintenance mode. It also enters in maintenance mode if the eLoader launches it in such a mode.

Runs the normal application. It is launched by the eLoader after a request of the eUpdater.

**Figure 1**: Partitioning of the FLASH of the ARM.

MAPPING OF RAM ON ARM USING EMBODY

Reserved for running e-process

Cache for sharedStorage

The running e-process uses it as it wishes.

Keeps **cached** information used by every e-process: partition table, MAC, IP and CAN addresses, IP mask, etc.

shalPART

shalNFO

IPC

Keeps is a NZI (not-zero-initialized) RAM section whose value is maintained across a SW reset and is used for Inter-Process-Communication such as messages to the eLoader about which e-process to load etc

**Figure 2**: Partitioning of the RAM of the ARM.

MAPPING OF EEPROM ON ARM USING EMBODY

Reserved for eApplication

sharedStorage

The eApplication can use it as it wishes.

Keeps information used by every e-process: partition table, MAC, IP and CAN addresses, IP mask, etc.

Inter-Process-Communication such as messages to the eLoader about which e-process to load are implemented using NZI (not-zero-initialized) RAM.

shalPART

shalNFO

Reserved for IPC

**Figure 3**: Partitioning of the EEPROM of the ARM.

## The eLoader

It is executed at bootstrap and its only responsibility is to launch another e-process. If it cannot load any e-process, it enters in idle mode. In case of any HW error it enters in fatal error mode.

STATE MACHINE FOR THE ELOADER

init

fatal error

launcher

idle

KO

OK

For instance: HW failure such as EEPROM or ETH switch

cannot jump to any

jump to an e-process

**Figure 4**: State Machine of the eLoader.

THE INIT STATE IN THE ELOADER

init

Init HAL and sharedStorage

KO

Runs for the first time

In sharedStorage: save boardInfo and loaderInfo, set defPROC equal to eApplication

YES

NO

OK

Any failure

**Figure 5**: The init state in the eLoader.

THE LAUNCHER STATE IN THE ELOADER

launcher

Check a jump to IPCREQPROC

There is an IPC jump request to IPCREQPROC

YES

jump to an e-process

NO

Check a jump to eUpdater

cannot jump

cannot jump

cannot jump to any

/ IPCrequest< IPCREQPROC = eUpdater >

IPCREQPROC is for the eUpdater

NO

YES

can jump

can jump

**Figure 6**: The launcher state in the eLoader.

THE FATAL ERROR STATE IN THE ELOADER

fatal error

Blink LEDs at 10 Hz, 20% ON

**Figure 7**: The fatal error state in the eLoader.

THE IDLE STATE IN THE ELOADER

fatal error

Blink LEDs at 10 Hz, 80% ON

**Figure 8**: The idle state in the eLoader.

## The eUpdater

It is the first e-process that the eLoader launches when electrical power is switched on.

Normally, the eUpdater remains active only for a few seconds (default is 5, but this time is configurable), opens a listening socket and then, if not contacted by a host, it passes control back to the eLoader which launches the default process.

Instead, if it receives a network command from a host, then the eUpdater enters a maintenance state in which with the aid of an external program it is possible to configure the system (change IP, default process, etc) and to load new FW.

The eUpdater enters the maintenance state also if it is launched by the eLoader with an IPC command to enter its maintenance state. It happens that if the eUpdater is the default process or if the eApplication ask the eLoader to launch the eUpdater in maintenance mode.

STATE MACHINE FOR THE EUPDATER

init

wait

maintenance

YES

goto-maintenance

There is an IPC jump request IPCREQPROC = eUpdater

timeout / IPCrequest< IPCREQPROC = defPROC >, restart()

restart / restart()

NO

**Figure 9**: State Machine of the eUpdater.

THE WAIT STATE IN THE EUPDATER

wait

0

timeout

infinite

Start countdown timer for Tstay

goto-maintanance

Received a UDP message

eval Tstay

Blink LED at 1Hz, 50% ON

manage UDP communication

defPROC is eUpdater

YES

NO

**Figure 10**: The wait state in the eUpdater.

THE MAINTENANCE STATE IN THE EUPDATER

maintenance

restart

Received a UDP message

Blink LED at 0.5Hz, 50% ON

manage UDP communication

process command

IP address (set / get)

defPROC (set / get)

Process info (get)

FW update

Blink LED very fast

restart

**Figure 11**: The maintenance state in the eUpdater.

## The eApplication

It is the normal application executed on the device.

Even if the eApplication can be written in any form, there are some basic mandatory requirements and some good practices to follow.

The basic mandatory requirements are the following.

* The eApplication shall place at a given position a const eEmoduleInfo variable which contains information about its build date, version, name, etc. Such a variable is used by the eUpdater to give information to the connected host.
* Use the scatter file provided by the emBODY: eApplication.sct.
* Relocate the ISR vector to the address of the eApplication, as defined by file eMemorymapping.h.
* Others

The good practices are the following.

* Use embOBJ to write the application by following the example in project EOMeApplBasic.test or EOMeApplBasic.full.
* If the project is compiled with macro EENV\_EAPPLICATION\_FORCE\_CODE\_OFFSET\_TO\_ZERO and it is used the scatter file eApplication-zerooffset, then the same application can run without eLoader and eUpdater for a simpler debug.
* Others.

### A sample application

See EOMeApplBasic.test or EOMeApplProtocol.full

### A special application: the EOMeApplMaintainer

This application allows to update the eLoader and the eUpdater.

## The Shared Storage

The shared storage contains information used by the e-processes. The storage is managed via the library sharedStorage which contains shalBASE, shalPART, shalINFO.

The library stored its data in EEPROM and uses a cached copy in dedicated RAM to speed up the reading process.

The position of its data structure is configurable via the eEmemorymapping.h file.