Applications in ARM boards

This document describes the possibilities to develop an application for an ARM-based board using the services offered by emBODY framework.

With emBODY it is possible to write applications at different level of complexity: from the very simple main() function up to HW support and more and more structured services.

For instance, the HL library offers the simple main() function paradigm and easy support for MPU peripherals and attached chips. The abstraction layer allows writing libraries and simple applications by isolating the complexity of lower layers under invariant APIs. Finally, embedded objects help to develop complex applications with many concurrent or serialised services (eg., IP sockets, CAN transmission, protocols, control algorithms, error handling, etc.).

As an example, the EMS board uses the services of emBODY at different levels. The eLoader uses the abstraction layer and a utility library to manage its simple duties (reading shared memory, a permanent storage in EEPROM, jumping to the relevant process). The eUpdater uses a basic set of embedded objects as it also must manage IP communication. The eApplication uses additional embedded objects to also manage: a more structured protocol, CAN gateway mode, motor control, error handling, all inside a tight control loop.

Approval History

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Version | Author | | Date | Approved | | Date |
| 0.1 | Accame | iCub Unit |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 0.1 |  | M. Accame | First edit of the document |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 The emBODY for ARM 1](#_Toc377126432)

[1.1 The structure of emBODY 1](#_Toc377126433)

[1.2 The low level 1](#_Toc377126434)

[1.3 The middle level 1](#_Toc377126435)

[1.4 The high level 2](#_Toc377126436)

[2 The applications 3](#_Toc377126437)

[2.1 Low level 3](#_Toc377126438)

[2.1.1 Simple main 3](#_Toc377126439)

[2.1.2 HL STM32 3](#_Toc377126440)

[2.2 Middle level 3](#_Toc377126441)

[2.2.1 HL PLUS 3](#_Toc377126442)

[2.2.2 OOSIIT 3](#_Toc377126443)

[2.3 High level 3](#_Toc377126444)

[2.3.1 Abstraction layer 3](#_Toc377126445)

[2.3.2 Basic embOBJ 3](#_Toc377126446)

[2.3.3 IP communication embOBJ 3](#_Toc377126447)

[2.3.4 Control loop embOBJ 3](#_Toc377126448)

[3 Rumenta utile per editing del documento 4](#_Toc377126449)

[3.1.1 UDP protocol for CAN gateway service 4](#_Toc377126450)

# The emBODY for ARM

The emBODY framework is a layered SW environment which can be used at any level to obtain the desired trade-off of SW complexity and performance for the target application.

The development of a highly efficient PWM controller should be done at low level close to HW registers. On the other hand, an application which must manage IP and CAN communication, structured protocols, motor control, data acquisition, error handling, etc. should be developed in a more structured and encapsulated mode.

Here is what the emBODY framework offers.

## The structure of emBODY

STRUCTURE OF EMBODY

HL - CMSIS

HL - STM32LIB

HL - PLUS

EMBEDDED OBJECTS

SERVICES

ABSTRACTION LAYER (OSAL, HAAL, IPAL)

RTOS (OOSIIT)

EVENTVIEWER

LOW LEVEL

MID LEVEL

HIGH LEVEL

**Figure 1**: The structure of emBODY.

## The low level

It contains the basic structure for developing simple applications. It is grouped under the library HL (Hardware Level) which is the customisation of ARM CMSIS and the STM32 standard peripheral library offered by ST Microelectronics.

The HL so far supports MPUs by ST Microelectronics such as STM32F103, STM32F107 and STM32F407.

## The middle level

It extends the low level in several aspects.

* HL-PLUS simplifies management of some peripherals of the MPU (e.g., I2C communication, CAN initialisation, GPIO basic management) and it gives support to selected external chips (some EEPROMs, gyroscopes, accelerometers, etc.).
* Eventviewer allows viewing timing diagrams in run time in the compiler IDE.
* OOSIIT gives the possibility of organising the application with an open-source fully pre-emptive RTOS optimised for ARM environment.

## The high level

The high level is formed by an abstraction layer which gives separate services using a coherent API. The OS services are provided by OSAL, HW services by HAAL, and basic IP services by IPAL.

These libraries can be used directly, possibly using only one of them, or can be used by some libraries of function (called services) or by the embOBJs, which are a library of objects for the embedded environment.

The applications

## Low level

### Simple main

### HL STM32

## Middle level

### HL PLUS

### OOSIIT

## High level

### Abstraction layer

### Basic embOBJ

dxewdwedw.

### IP communication embOBJ

### Control loop embOBJ

Rumenta utile per editing del documento

|  |  |  |
| --- | --- | --- |
| COMMAND | OPC | Description |
| CMD\_SCAN | 0xFF | The EthLoader sends it in broadcast to query existing devices.  PKT = {OPC}  The EMS sends back a reply of 14 bytes:  PKT = {OPC, D01, .. , D13}, where  D01: module->info.entity.version.major  D02: module->info.entity.version.minor  D03: BOARD\_TYPE\_EMS = 0x0A  D04-D07: IP net mask  D08-D13: MAC address |
| CMD\_CANGTW\_START | 0x20 | The EthLoader sends it to start the CAN gateway mode on EMS.  PKT = {OPC}  At reception, the EMS enters in CAN gateway mode. The EMS initialises the CAN1 and CAN2, sends twice the BOARD command over CAN to force the boards to enter in bootloader, and then it enables the communication CAN1 / CAN2 🡨🡪 UDP port 3334. The whole startup takes two seconds.  It sends back NOTHING. |

**Table 1** – UDP commands on service port 3333.

### UDP protocol for CAN gateway service

When the can cedcedcewcewqcweq. The UDP packets use the following protocol to exchange CAN frames between a host and the attached CAN boards.

UPD PACKET FOR CAN GATEWAY

UDP CAN GTW

HEADER

BODY

8 BYTES

16\* N BYTES

**Figure 2**: The UDP CAN GTW frame used for can gateway service.

HEADER OF UPD PACKET FOR CAN GATEWAY

HEADER

BODY

8 BYTES

SIGN

N

DUMMY

1 BYTE

1 BYTE

6 BYTES

Used to recognize a valid frame: 0x12

Number of CAN frames

**Figure 3**: The header of the UDP CAN GTW frame.

BODY OF UDP PACKET FOR CAN GATEWAY

HEADER

16\*N BYTES

CANFRAME

BODY

i = 1 .. HEADER.N

CANX

LEN

ID

DUMMY

DATA

1 BYTE

1 BYTE

2 BYTES

4 BYTES

8 BYTES

CAN1 (1) or CAN2 (2)

Of DATA field

CAN ID at 11 bits

CAN data

**Figure 4**: The body of the UDP CAN GTW.