The control loop applications in ARM boards

This document describes a particular application which is used as a skeleton for the EMS board: the control loop application. This application is built using the embOBJ framework and some HW timers provided by HAL. The HW timers offer a configurable but precise periodic control loop executed at maximum priority and the embOBJ framework offers more advanced services such as multitasking for lower priority tasks, UDP networking, error management etc.

The control loop executes in round robin three high priority tasks: one for reception, one for doing operations, and one for transmission. When the MPU is not involved in any of the above three tasks the scheduler of OSAL allows other tasks to execute. Typically those tasks are embedded in suited objects (the backdoor, the listener, the error handler, etc.).

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 control loop application 1](#_Toc377145175)

[1.1 Configuration 3](#_Toc377145176)

[2 Rumenta utile per editing del document 4](#_Toc377145177)

[2.1 dxxwqcwe 4](#_Toc377145178)

[2.2 sdcewcxzc 4](#_Toc377145179)

[2.2.1 UDP protocol for EMS service 4](#_Toc377145180)

[2.2.2 UDP protocol for CAN gateway service 5](#_Toc377145181)

# The control loop application

The main() function must initialise an instance of EOMtheEMSapplCfg which contains the configuration the user specifies.

Then it initialises the EOMtheSystem singleton with a configuration taken from the instantiated EOMtheEMSapplCfg. The configuration tells how to launch the standard services: the timer manager, the callback manager, the memory pool, the error manager. It also keeps configuration of the HAL and of OSAL.

Finally the main() starts the EOMtheSystem with the configured basic services and it launches the user-defined function.

In this function the user must initialise the EOMtheEMSappl object which prepares whatever is needed for the control loop.

What is needed is the following.

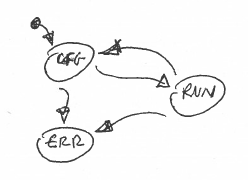
* The EOtheARMenvironment manages all storage services in EEPROM.
* The EOMtheIPnet starts the UDP stack with proper MAC and IP addresses and provides UDP socket communication.
* The EOMtheEMSbackdoor allows debug communication via UDP socket.
* The EOMtheEMSsocket prepares a UDP socket for Ethernet protocol communication with the host (the PC104).
* The EOMtheEMStransceiver allows encoding and decoding of UDP packets over the socket in EOMtheEMSsocket and performs the required callback actions.
* The EOMtheEMSdiscoverylistener allows the EMS to listen to UDP commands coming from the EthLoader.
* The EOMtheEMSerror allows managing the error state of the EMS (ERR).
* The EOMtheEMSconfigurator manages the initial configuration phase (CFG) of the EMS before it enters the control loop.
* The EOMtheEMSrunner object finally contains the execution of the control-loop (RUN) with a period specified inside EOMtheEMSapplCfg.

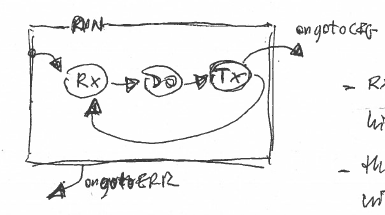
The above singletons instantiate one or more tasks (via the EOMtask object) which execute at various priority levels, according to external events (basically a UDP packet which arrives).

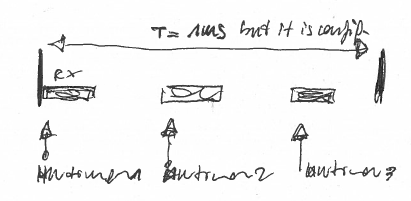
At startup, the EMS is in CFG mode and the task inside the EOMtheEMSconfigurator works. Then when a given UDP message arrives, it enters in RUN mode and the EOMtheEMSrunner enters in execution.

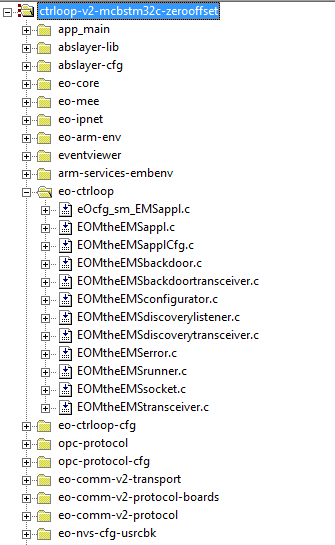
When the EOMtheEMSrunner enters in execution, its three tasks (RX, DO, TX) are executed at maximum priority in round robin and triggered by HW timers. They contain the control loop. All other tasks executes when RX, DO, and TX are idle.

An error sends the EMS in ERR mode.



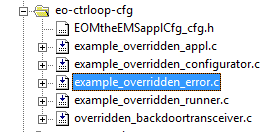






## Configuration

What the application does inside the EOMtheEMSrunner (and other objects) is configured by overriding weakly defined functions.



Rumenta utile per editing del document

BEHAVIOUR OF EMS AT POWER ON

eUpdater

WAIT MODE

Timeout of 5 seconds

SERVICE MODE

eApplication

EMS services

(info + FW update)

CAN gateway

(FW update of CAN dev)

Received a UDP packet on 3333

eLoader

**Figure 1**: The behaviour of the EMS at power on.

## dxxwqcwe

c wedcdcxswcxs.

## sdcewcxzc

b nytnty nt.

### UDP protocol for EMS service

dxewdwedw.

|  |  |  |
| --- | --- | --- |
| 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.