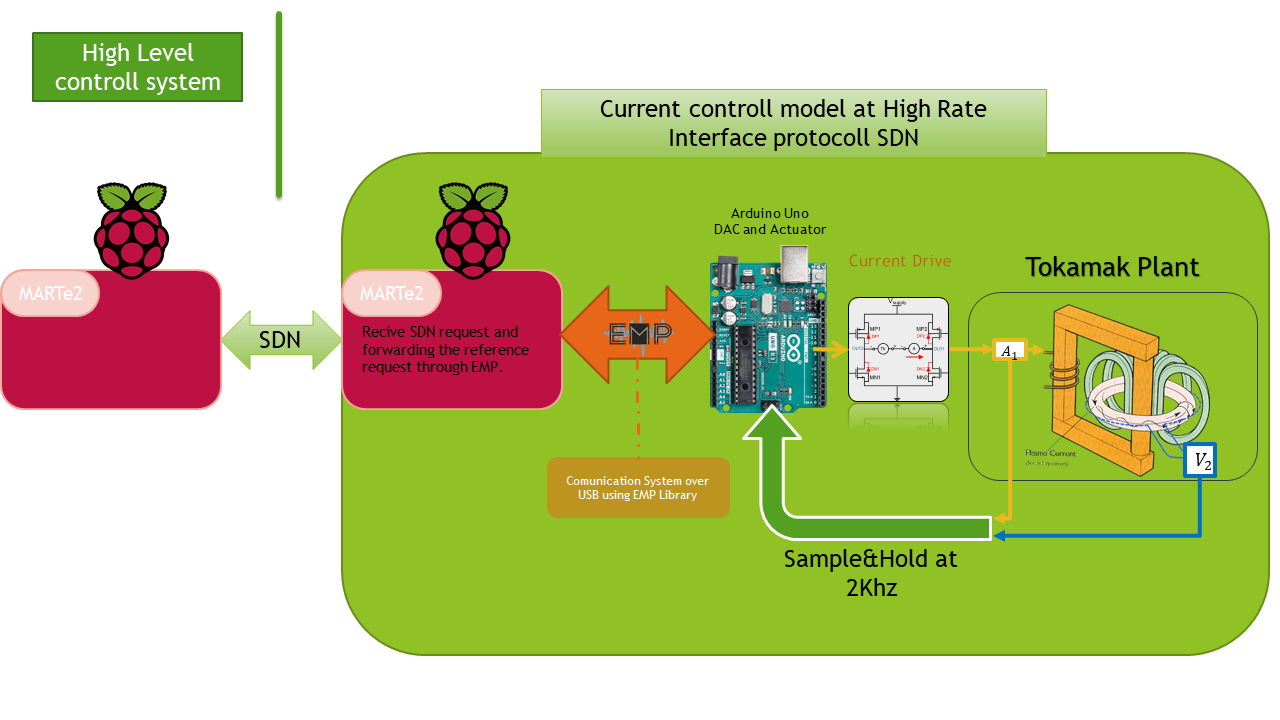
Control Architecture and Sampling for Tokamak prototype Report

# System Project architecture



The “Control Architecture and Sampling system” is designed to divide the current control problem from the reference set system:

1. Arduino have to control the Tokamank Coil followind the reference set point choise from MARTe2
2. MARTe2 receive the current feedback of the system every sample event and export it to the rest of the network

# Current Drive

The selected current drive is the dual Half-Bridge IBT\_2

Immagine che contiene testo, tabellonesegnapunti

Descrizione generata automaticamente

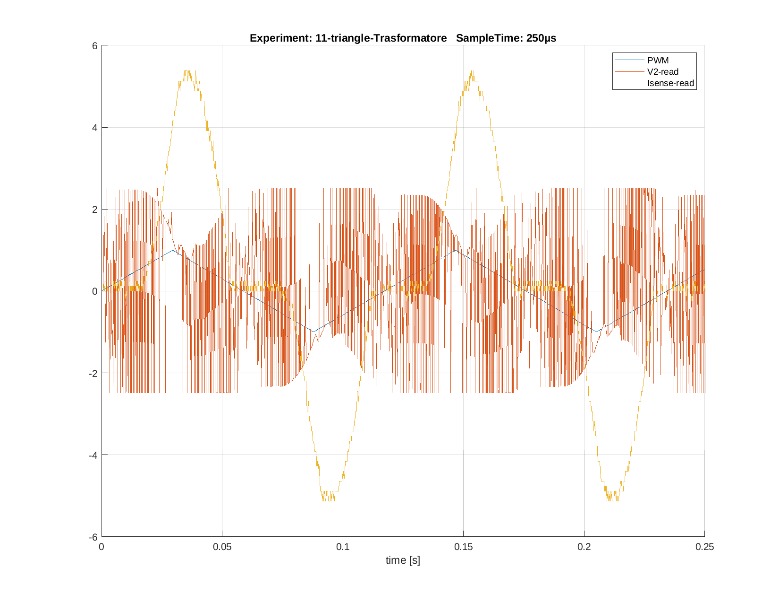
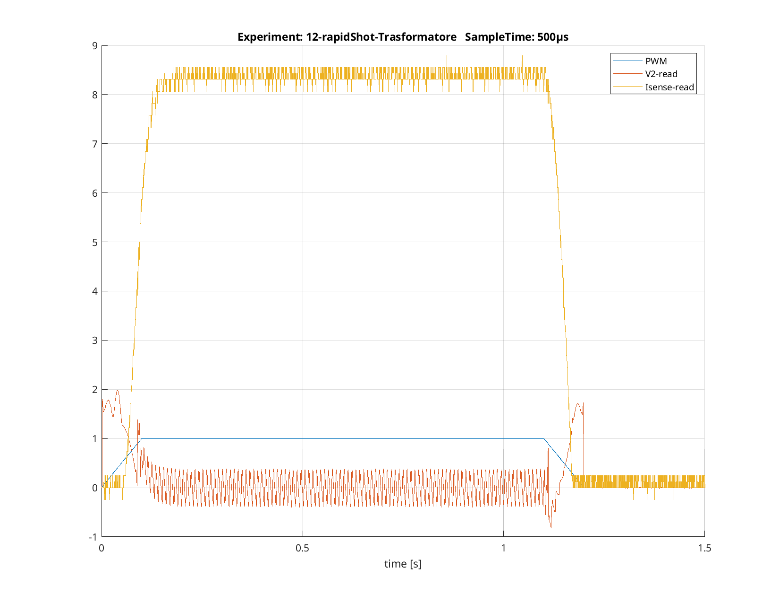
This model of drive can manage hi level current, the peak is 43A. It also isolate the logic region from the power sector, as can be see by the electrical scheme here:



The mosfet connected to the Plant are isolated from the µC inside the BTS7960B

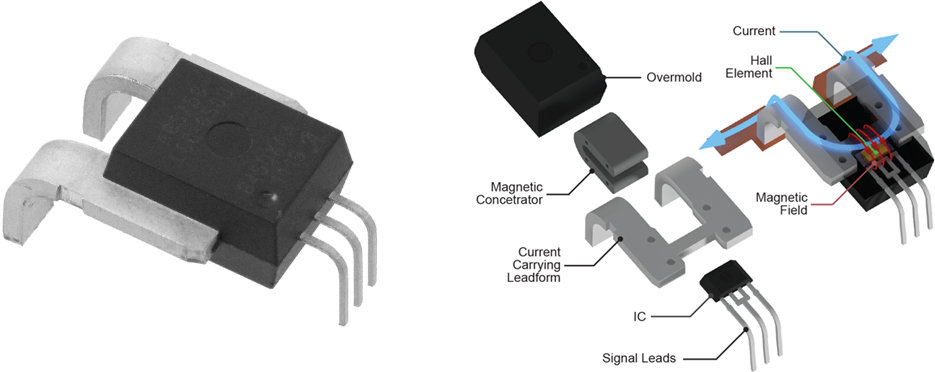
This driver has one nonlinearity problem: it has double Dead zone, at low duty-cycle and at high duty-cycle.

The problem shows up form this 2 tests:



And to solve it in the software, anti-dead zone code was realized.

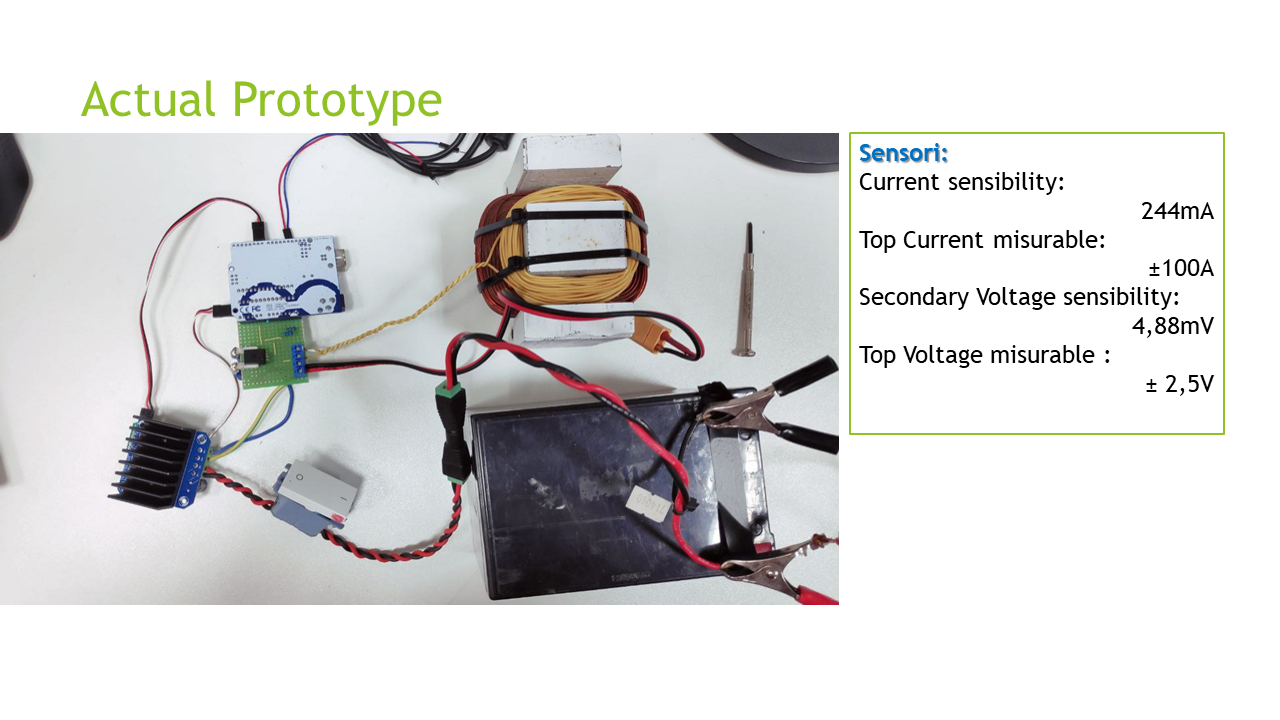
# Current Sense



To check the current on the primary the Allegro ACS770 High-Precision Linear Current Sensor IC was used.

The main feature of the device is the reading of the current directly with a Hall sensor.

We selected the “ACS770LCB-100B-PFF-T” variant, with the capability to read until ±100A.



## Immagine che contiene testo, segnale, luce Descrizione generata automaticamenteEmbedded Message Pack

This is a C++ library to perform the send/receive data between any platform (Arduino, Unix, etc...) with any type of data pack using asynchronous byte streams.

The key feature is the possibility to transmit between 2 different machines, without any necessity to have the same type device on the other hand.

### Supported Device form the library NOW

This library has the source for different type of the HW:

Linux

Arduino AvR (boot for Arduino-Ide and platformIo)

STM32 (Tested on the discovery board)

For now the library was tested with success on:

*  Arduino <==> Linux (Raspberry Pi for example, or Any Linux PC)
*  Linux <==> Linux (Using Socket abstraction)
*  STM32 <==> Linux

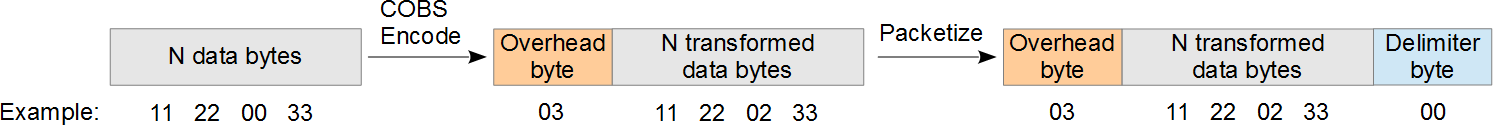
But we are confident for the correctly of the library, because the source file is always the same

### Pack Protocol info

The magic trick behind the library is the protocol used to encode/decode and send the packs use this element:

* [COBS codification](https://en.wikipedia.org/wiki/Consistent_Overhead_Byte_Stuffing) to transform any data-pack in **self ended pack data**, different data-pack haven't the necessity to have the same size.
* [CRC8](https://en.wikipedia.org/wiki/Cyclic_redundancy_check) to and minimal security check of the pack transmission (This feature can be disable, but here are suppose always active).

These elements permit to the library the Send/Receive operation not only for 1 type of message, but **any number of different pack** all in the same stream and with the same library.



Dist = 3

Dist = 2

### Algorithm Flow

#### Send Procedure

Fill the pack to be send, if there are more type of pack, specify the type in an opportune manner.

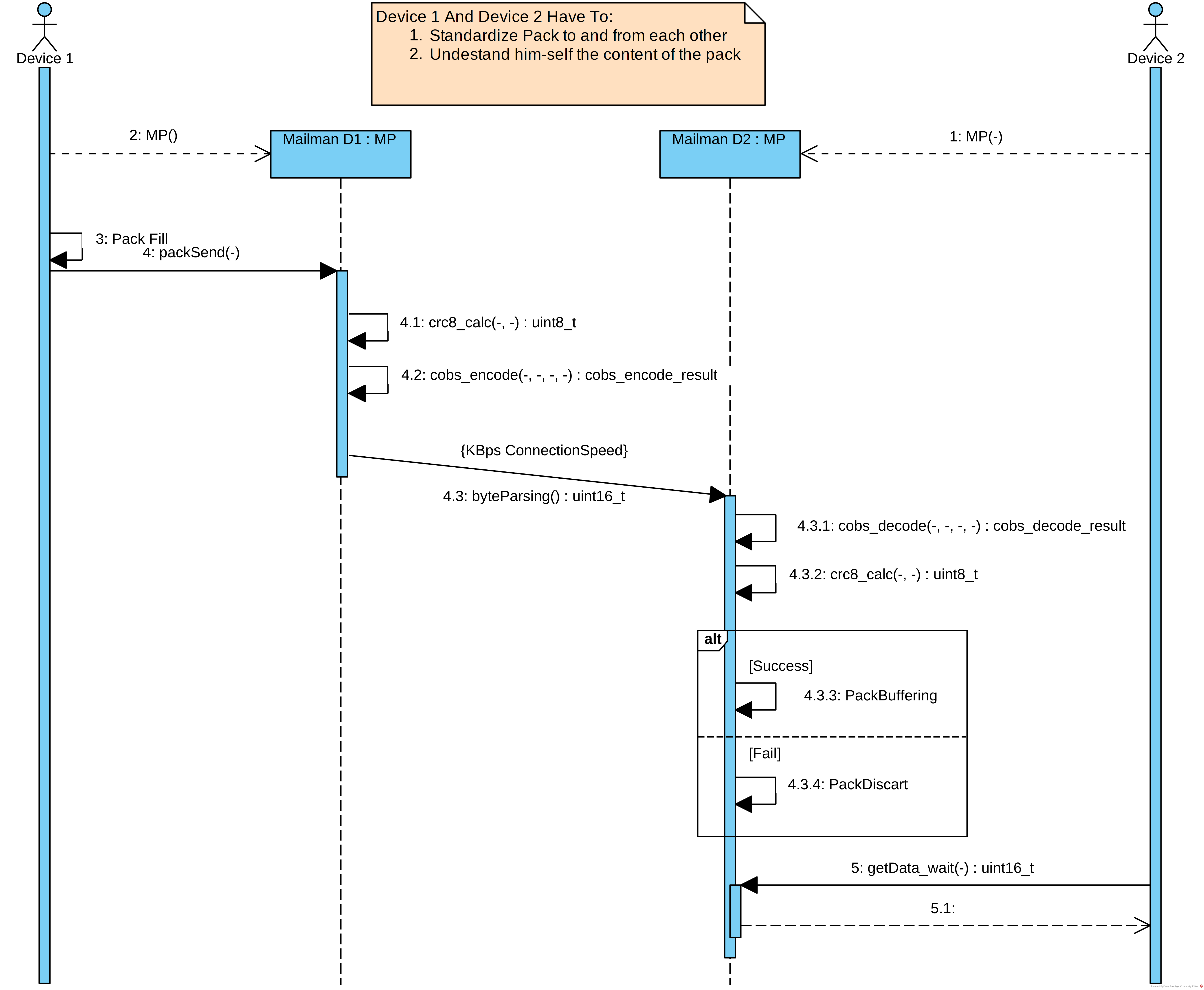
Library now start:

1. The sending stream is used to calculate his CRC8
2. The CRC8 calculated are append at the end of the stream
3. The new data-stream are COBS-Encoded
4. The result byte-string are send to the outputting device with \0 byte at the end

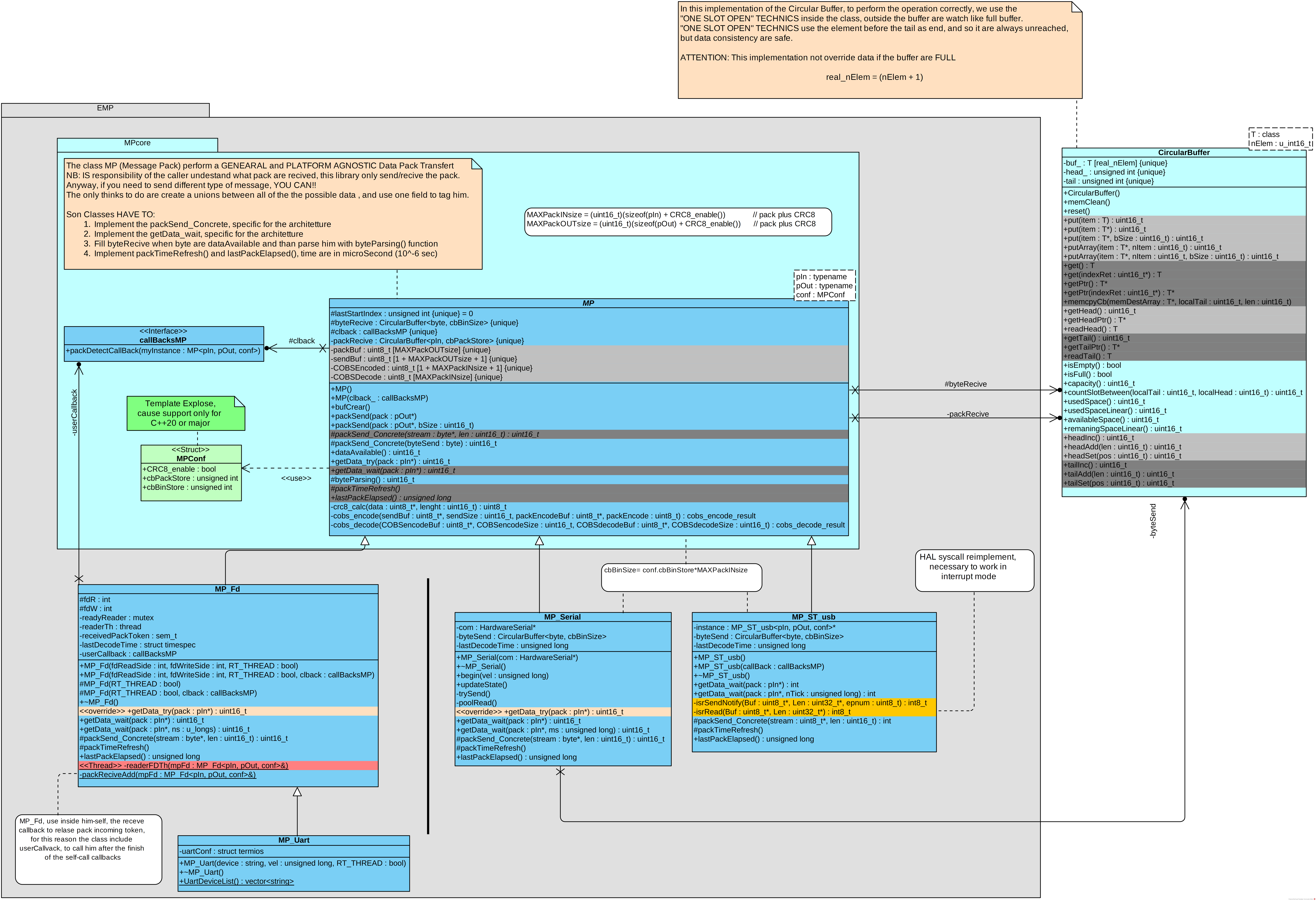
#### Receive Procedure

Now the receive device read this stream, and on the first \0 receive the library start parsing procedure: 6. The byte stream between this new \0 and the last-one are COBS-decoded 7. The CRC8 of the first n-1byte are calculate and compared with the CRC8 send

* [Compare Success] Pack is store and wait to be read
* [Compare Fail] Pack is discarded and unambiguous pack problem is solve for sure, the only solution is wait new pack incoming.



### EMP Class Diagram



it is easy to see that all concrete classes descend form the MP classes in MP core Package, this design simplify the developing process because the concrete, hard and critical work is coded all in the same manner for all, and after the debugging of the core, all the rest of the problem have to be in the son-class and limit his damage only to that zone.

## Pack Define

The system does not need any specific format for the message, are simply a C struct with the \_\_attribute\_\_((packed)) to reduce the size.

With this simple rule is possible define 2 different and asymmetric pack:

#include <stdint.h> // To be sure for the footPrint in any platform

struct \_packLinux2Ard {

int16\_t num;

char buf[20];

} \_\_attribute\_\_((packed));

typedef struct \_packLinux2Ard packLinux2Ard;

struct \_packArd2Linux {

int16\_t num;

char buf[10];

} \_\_attribute\_\_((packed));

typedef struct \_packArd2Linux packArd2Linux;

### Multiple pack define

To add the multiple pack capability, use a Union of the interest pack to create the final pack, with at the begin of the pack, one common byte for all.

This byte is a type byte and permit to specify the type, and during the receive phase detect the incoming pack type.

//DataType1 ,DataType2, DataType3 typedef above;

enum packTypeSend : uint8\_t{

DataType1code,

DataType2code,

DataType3code,

};

typedef union{

DataType1 t1;

DataType2 t2;

DataType3 t3;

} \_\_attribute\_\_((packed)) dataSendUnion;

typedef struct{

packTypeSend type;

dataSendUnion pack;

} \_\_attribute\_\_((packed)) dataSend;

The library can send less then the maximum size of the pack, so, are strongly recommended code like:

//...

dataSend.type = DataType2code;

dataSend.pack.t2 = /\*{data}\*/ // fill correctly dataSend

//...

MP->packSend(&data, sizeof(packTypeSend) + sizeof(DataType2));

//...

To avoid the sending of unused bytes.