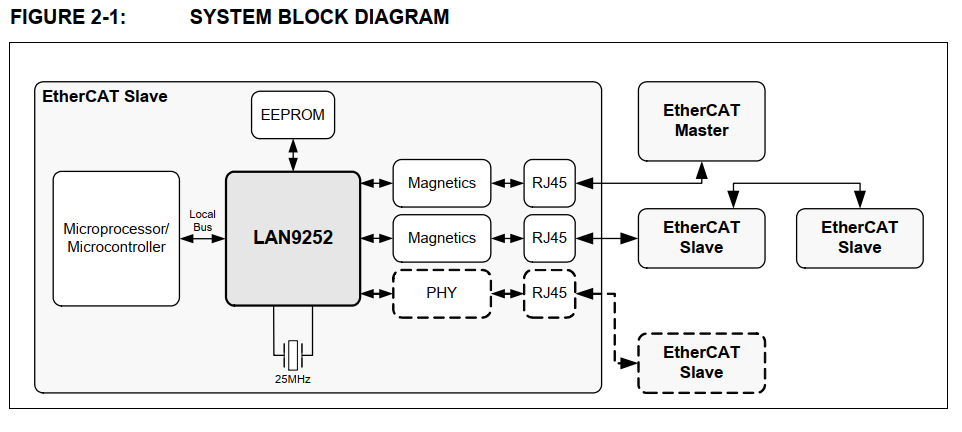
# Introduction of EtherCAT Implementation with LAN9252

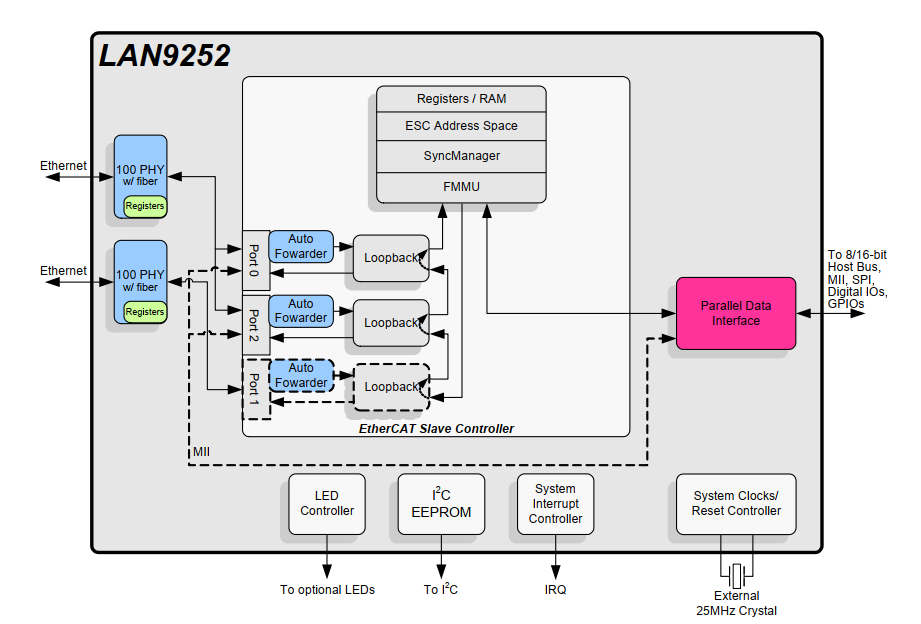
Device featured: Microchip DEVB-LAN9252-SPI

The SPI bit lanes are supported with a clock rate up to 80MHz

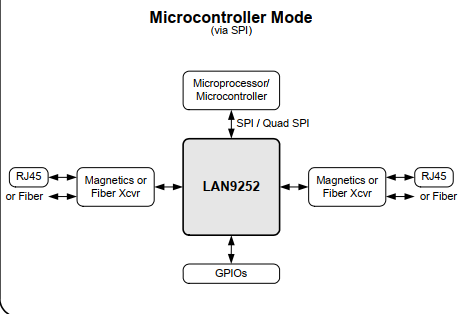
The following is the system block diagram from (Microchip), taking into account that the Microcontroller will be the Nucleo-F446ZE and the EtherCAT Master will be a PLC/IPC from Beckhoff.



The Internal diagram of the chip:



The following is the operation mode in which an external microcontroller uses a SPI or Quad-SPI to communicate and control the EtherCAT Slave.

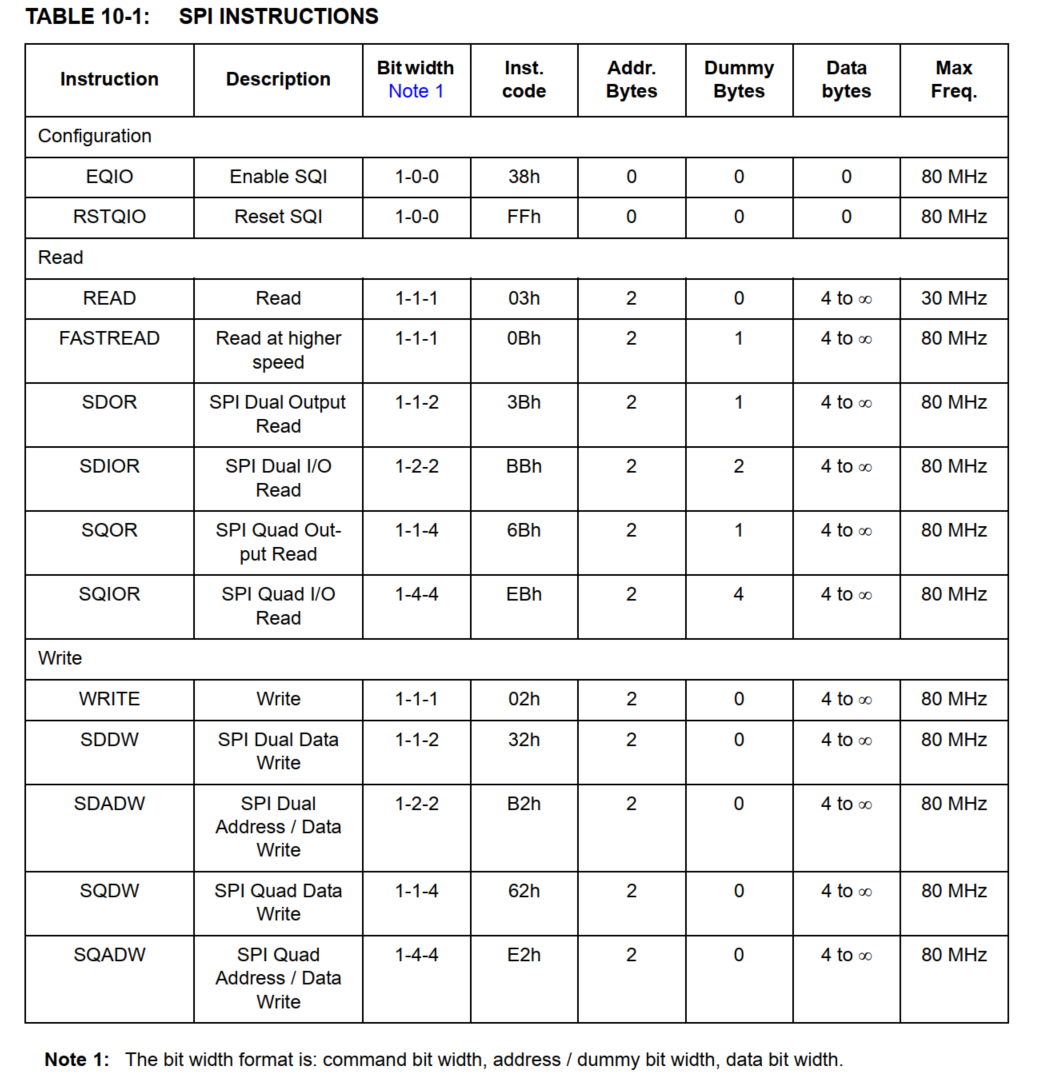


Regarding the implementation of the Protocol Stack

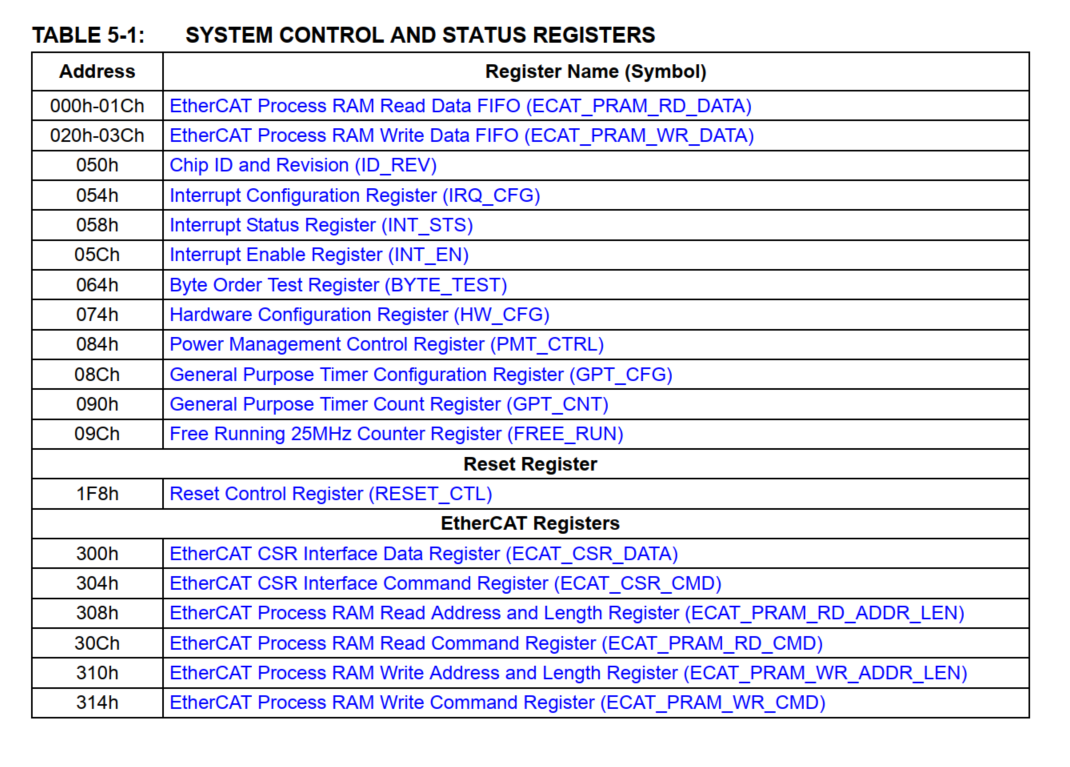
It might be possible that a licensed stack is needed such that the device can be certified as official compatible EtherCAT device.

<https://www.ethercat.org/en/products/54FA3235E29643BC805BDD807DF199DE.htm>

## Set of instructions for LAN9252 (SPI Mode)



## Status and Control Registers

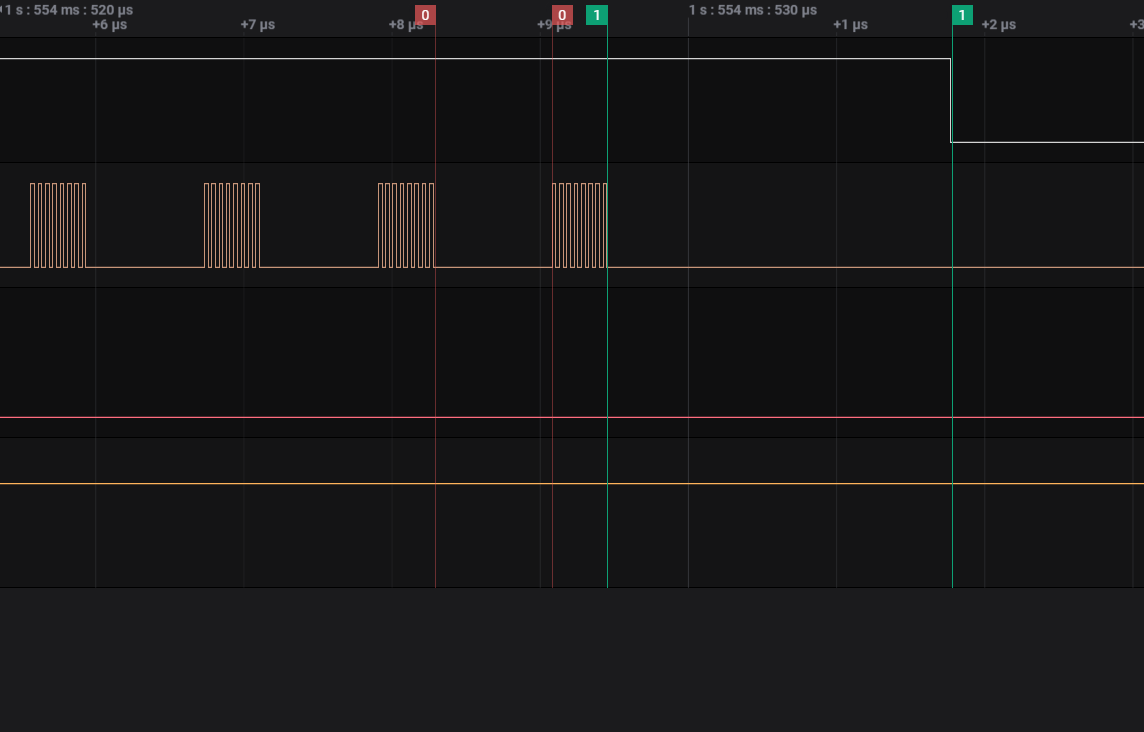


## First communication tests over SPI

Data to be sent is 0x0b 0x00 0x64

With a clock with a prescaler of 8

Those are errors in the channel



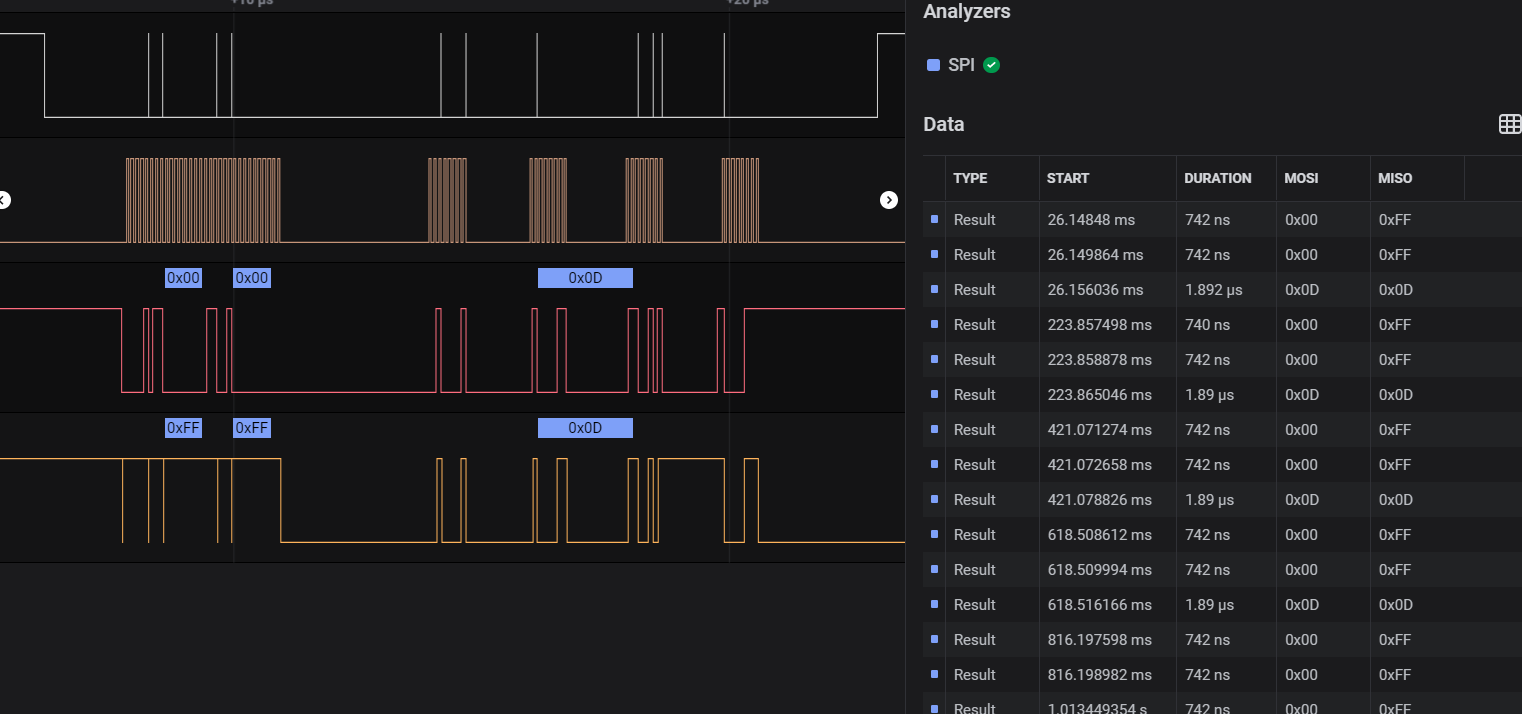
0 is the time between send instructions, even if they are executed with the library HAL\_SPI\_TransmitIT();

788 ns

6.25 ns cpu clock

1Is the period of time that the interrupt needs to be executed (first instruction)

2.33us



SPI with DMA

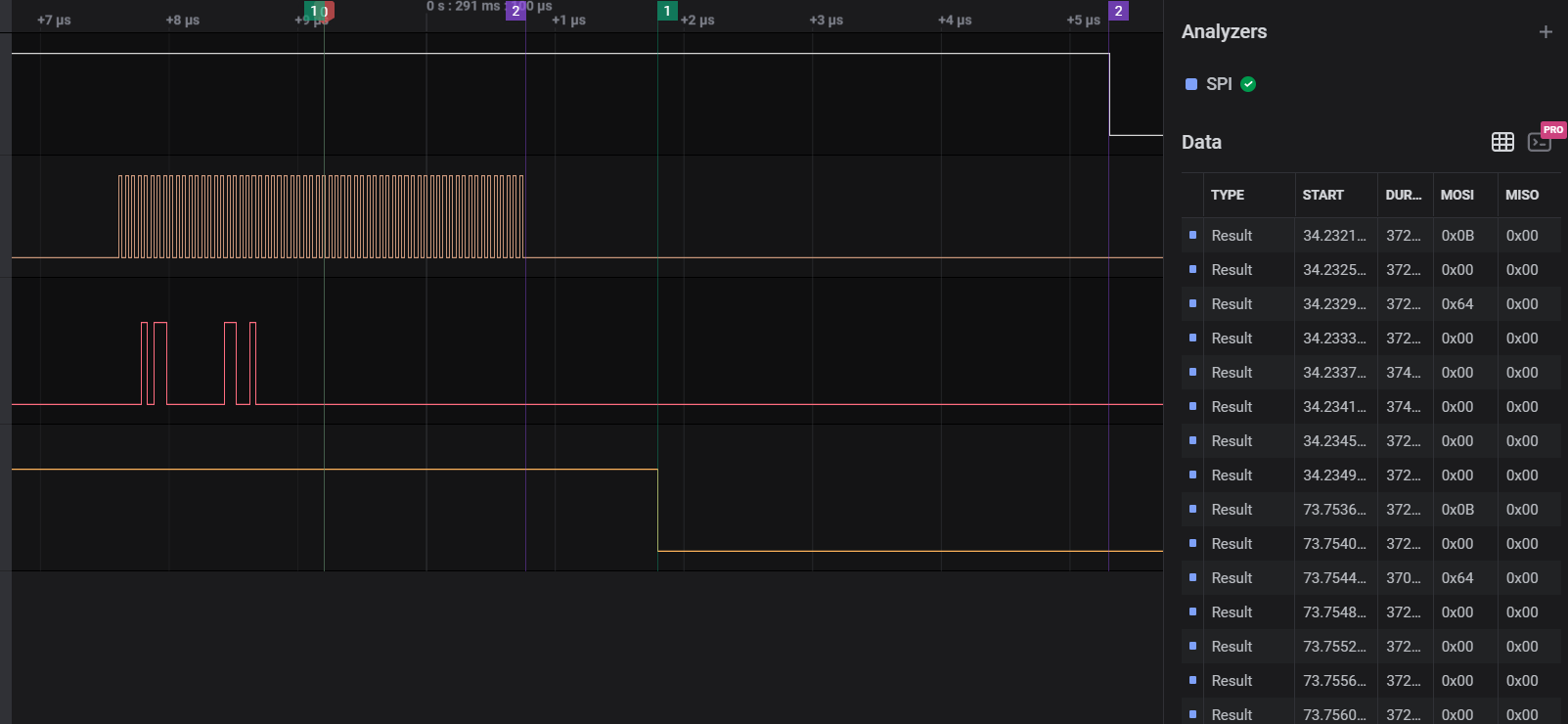
The following pictures shows the delay introduced by interruption handling: marker 1 is the delay until Half Buffer has been sent interruption, as for marker 2 corresponds the Full Buffer Interruption.

1.Half-Completed Transmission

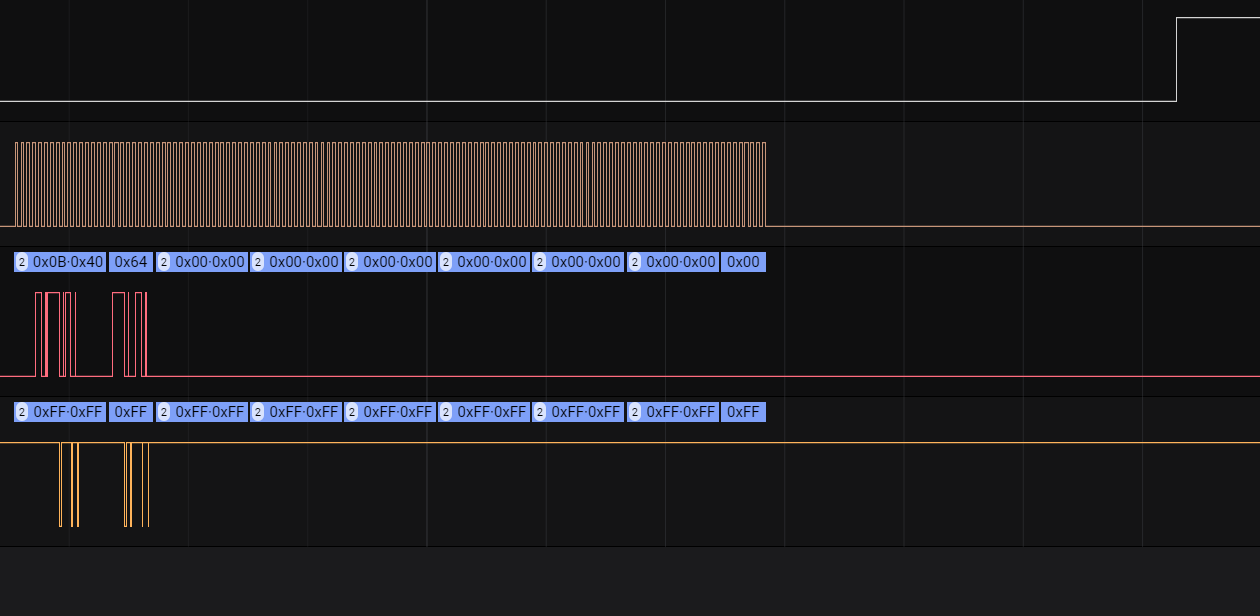
2.59 us

2.Full-Completed Transmission

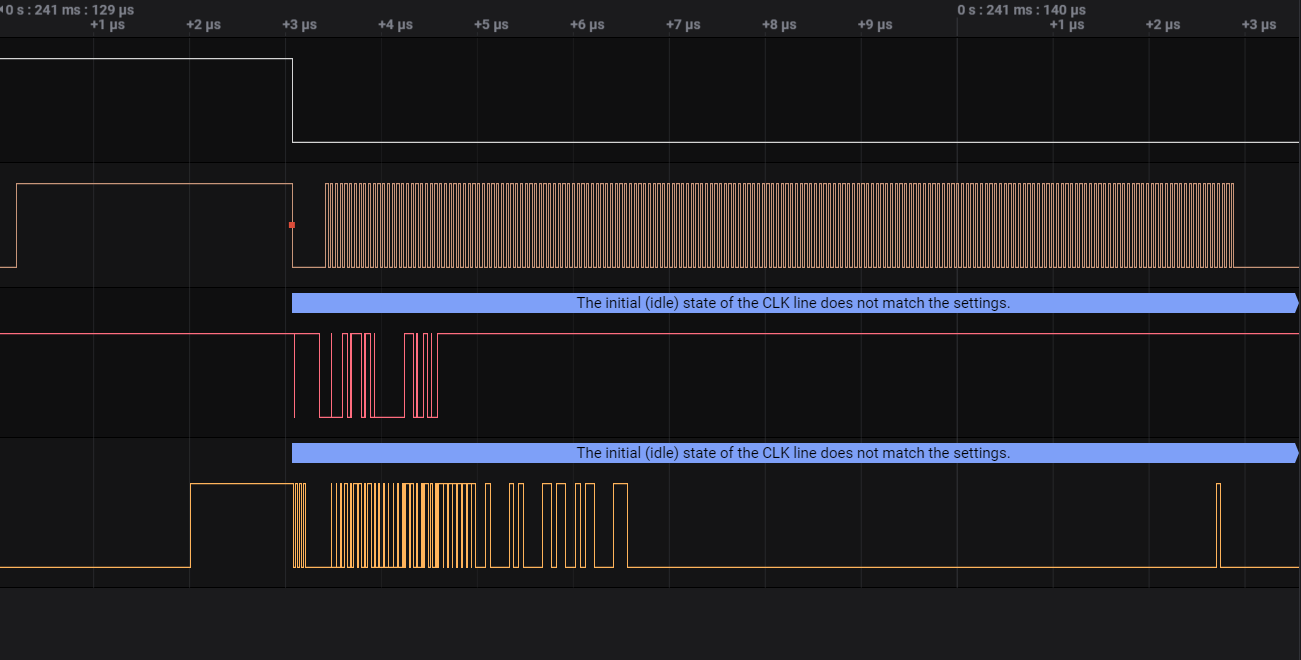
4.53us



SPI with DMA (only transmit instruction since the transmit receive one is getting lost)



The following is a right reading procedure of the BYTE\_TEST, the result is coming in Little Endian format.



# Design of the EtherCAT Slave Controller

## Preliminars

The application process data which will be the description for the Fieldbus of the data that is going to be transmitted is based on three different files:

* ESI
* SII-EEPROM
* CoE Object Dictionary (matrix for complex devices)

The ESI and SII-EEPROM files can be kept as small as possible to pass the CTT -Compliance Test for EtherCAT devices-. Optional Data (OD) will be stored in the Device itself and transmitted with CoE.

NOTE: The Device will assume a Slave Device with fixed PDO and OD data.

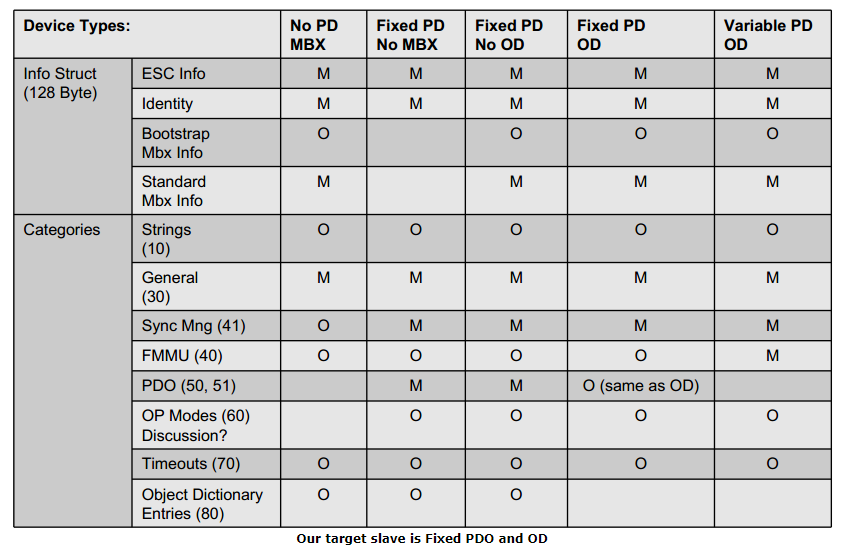


Figure 1 From SOES example

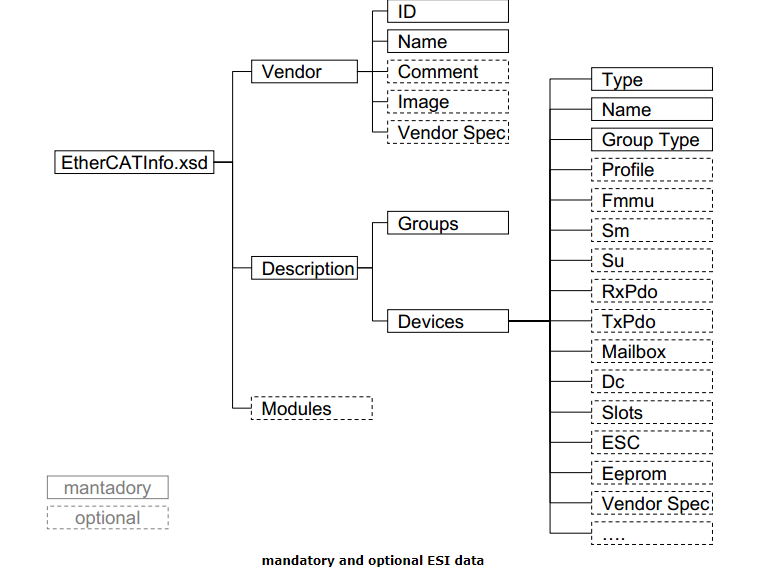


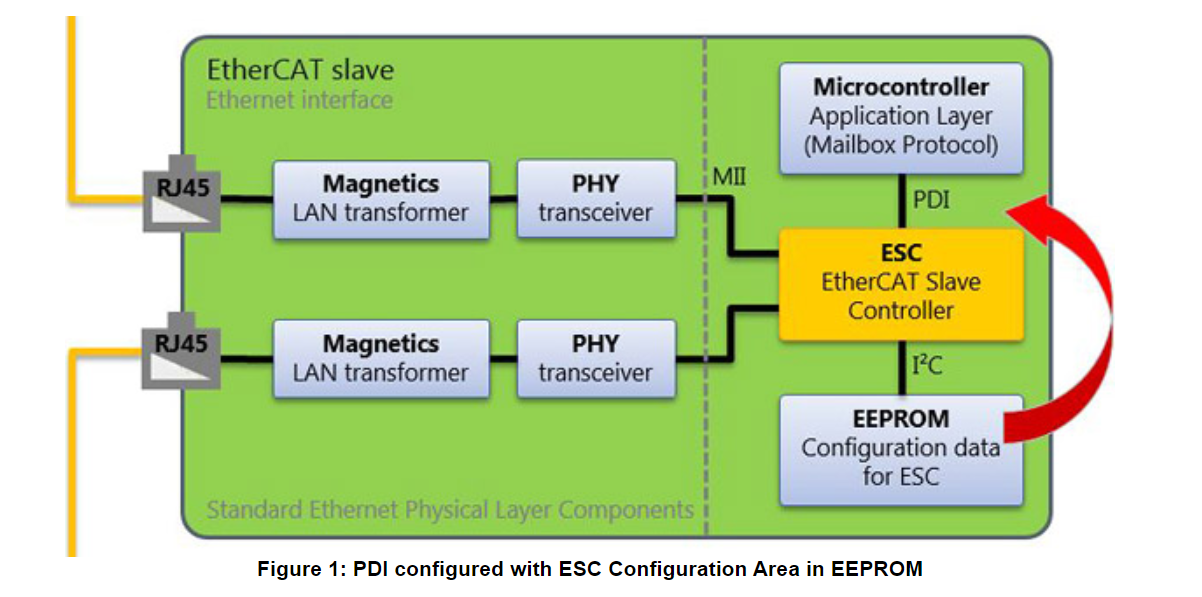
Figure 2 ESI data that is mandatory

Architecture of the EtherCAT slave, and how the EEPROM works as the definition of the Physical Device Interface (PDI) with the microcontroller over the Mailbox Protocol.

Important to notice: SII (Slave Information Interface) contains the same information as the ESI but it is stored in EEPROM. This hex/bin file needs to be written by the EtherCAT Master or the ESC.

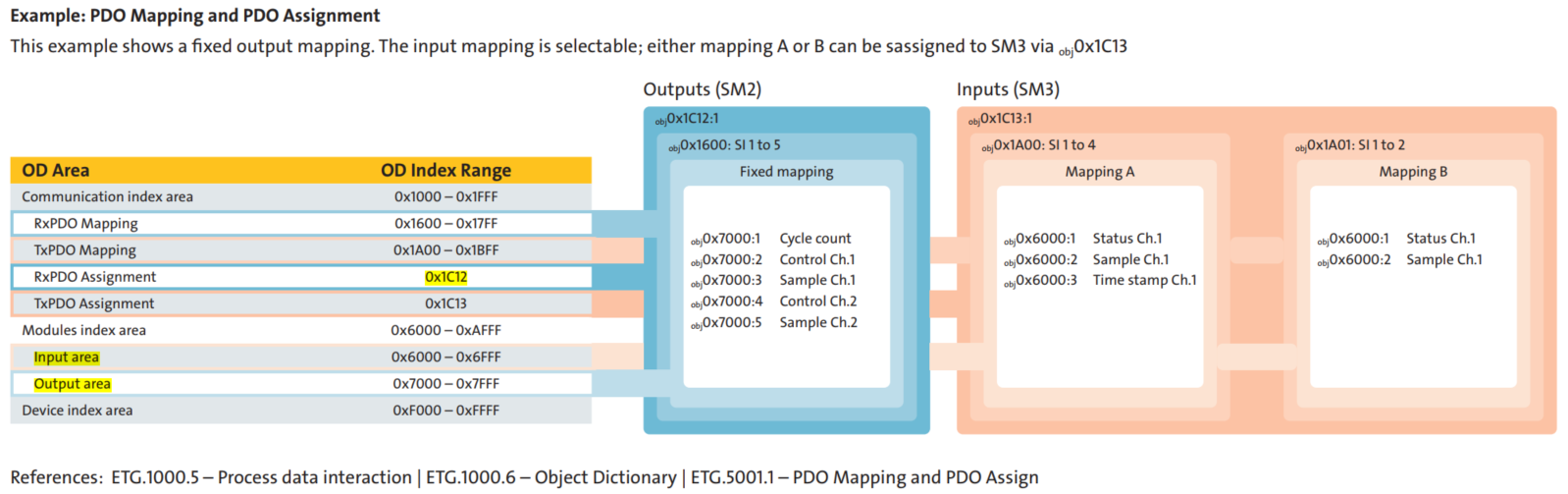
EEPROM file can be generated directly from an ESI File.

Taken from



## Process Data Configuration

To assign data to be exchanged, the microcontroller maps the data onto OD? Areas. The following can be seen in the diagrama taken from the EtherCAT\_Device\_Protocol\_Poster.



## Design constrains

Update: 2020.07.16

Firmware update will not be supported, therefore the FoE mailbox protocol will not be implemented.

This implementation could be done in the future but it would imply the integration of EEPROM writing/reading functions on the STM32F446. So far the consequences of this decision are not affecting the overall functionality of the EtherCAT Slave Device, it only will toggle between the Init, Pre-OP, Safe-Op, and Operation State. Only the state BOOT will be missed, that means that update of firmwares from master devices is not going to be available. This Feature is already implemented in the ESC.

Great question would be, whether this will affect that the master could write to EEPROM from ESC.

Because it could be that this FOE is only needed when the Microcontroller tries to do that from the SOEM.

## Interruption restrictions and Distributed Clocks

From AN2655 2.2.1 PDI Interrupt

Currently the LAN9252 EVB SPI only counts with the SPI interface connected to the MCU, which means that the DC and IRQ pins cannot be used to detect Synchronization clocks nor AL interruption requests.

This could be added afterwards to jump from a Polling CSR access to an ISR scope approach, such that the MCU could work on other tasks (if RTOS used).

This translates to no IRQ nor DC feature considered during initial hardware configuration.

