Real-time Ethernet Deterministic BUS

What is REDBUS ?

- REDBUS is an Ethernet based BUS with a patented* Ethernet topology.
- It provides a fast connections between multiple devices without the need for SWITCH or HUB.
- It requires only one PHY and one transformer per device.

Main characteristics

- Fast, deterministic & real-time communication.
- IEEE 802.3 standard Ethernet frames.
- Small footprint: fits into small FPGA or CPLD (< 300 LUTS).
- Cost-effective solution.

*Ethernet network device and local ethernet network: EU EP2930896, IT 0001423488.

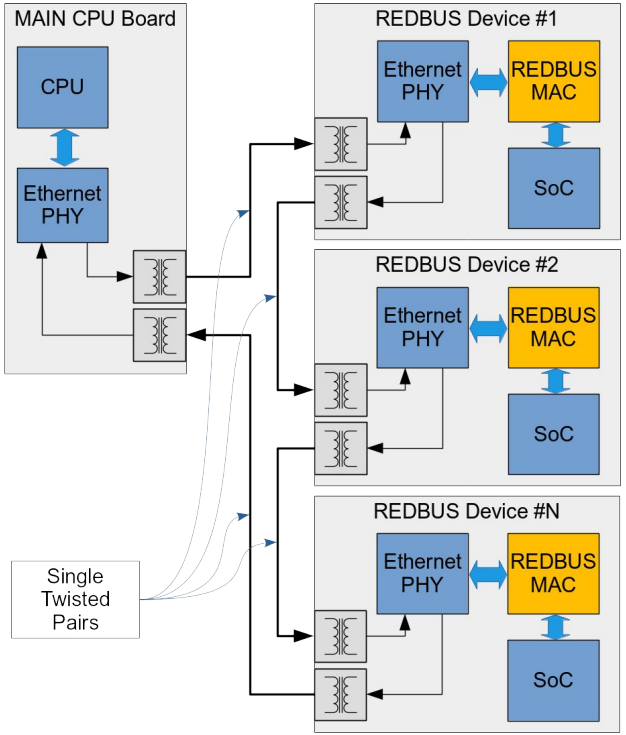
Devices block diagram

The REDBUS system is characterized by a Controller and one or more peripherals in which the communication system is based on the Ethernet standard and a ring interconnection topology.

The uncommon but admitted connections between devices enable the absence of Ethernet switches.

The system is designed to minimize communication latency between the controller and peripherals and to maximize data throughput.

The controller can be implemented on generic microcontroller-based systems provided with an Ethernet communication port and does not require special components.

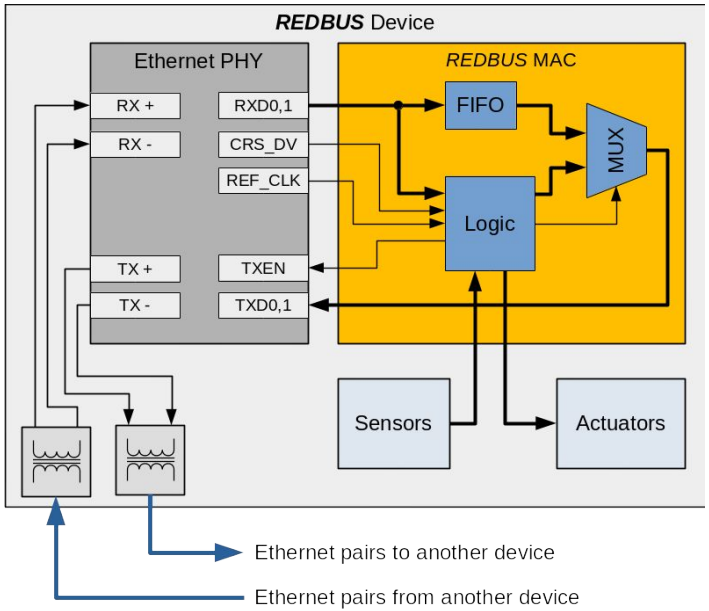


Devices block diagram

REDBUS peripherals consist of two basic components:

- A standard Ethernet physical device
- An SoC (System-on-Chip) realized in FPGA components, which constitutes the special MAC (Media Access Control).

The protocol stack is reduced to the minimum. It requires very few resources and can fit in very small microcontrollers.



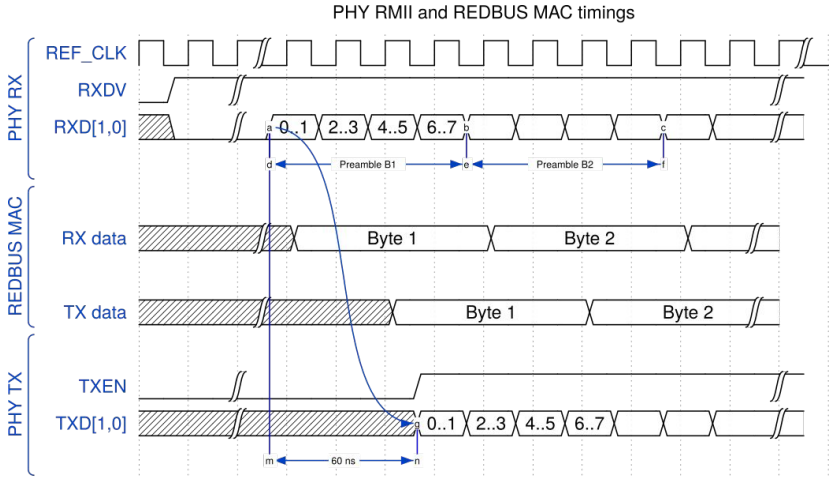
Ethernet timings

The incoming data on the RX channel of the PHY device is sent to the MAC through the RMII port on RXD signals.

The MAC sends the received bits on the RMII TXD signals after a slight delay of 60 ns.

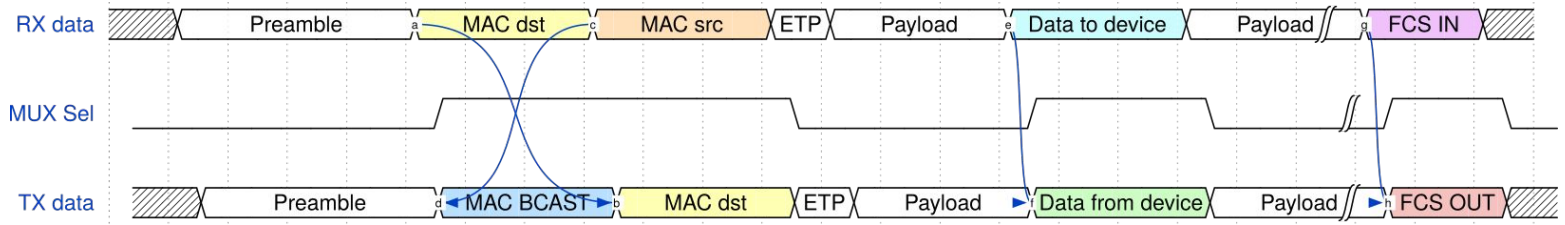
In this way, the ethernet frame traverses each device in the ring and returns to the controller that initiates the transmission.

For large data payloads, the ethernet frame may come back to the controller even if the transmission of the same frame is still in progress.



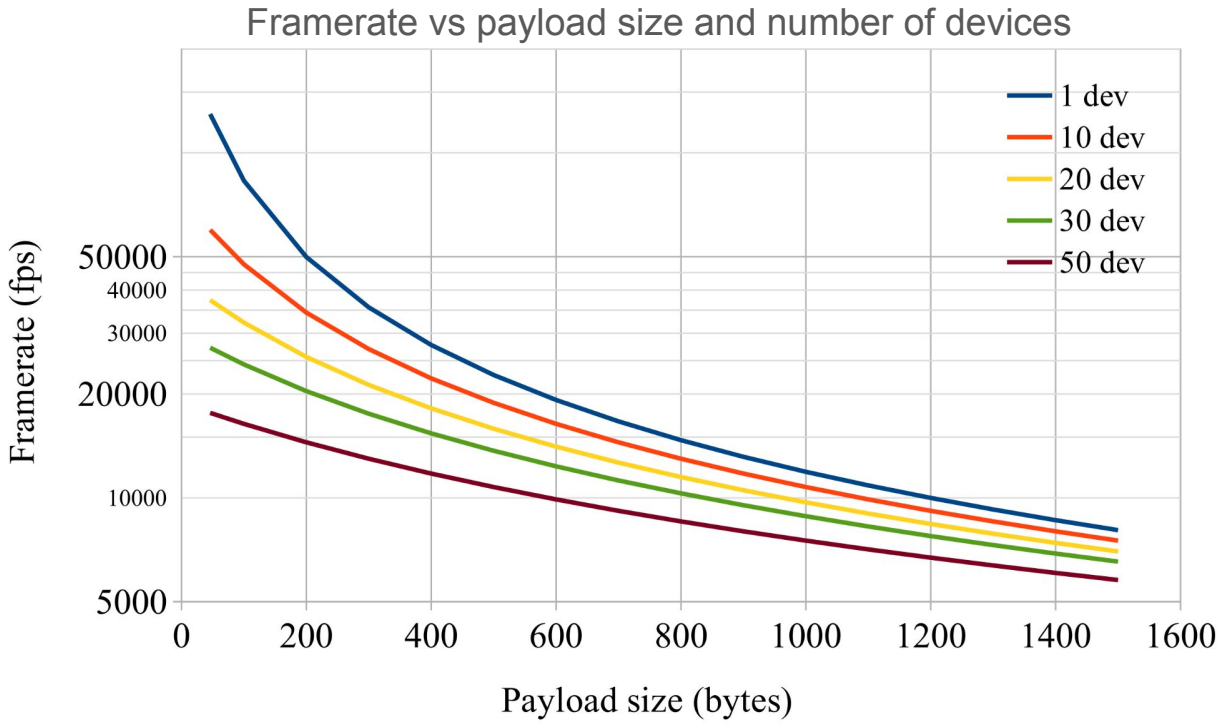
Ethernet timings

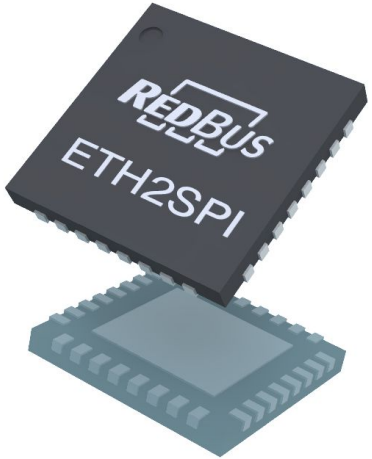
The first REDBUS device exchanges the source and destination MAC address so the controller can receive the repeated frame back.



Each device can modify a portion of the payload: data sent by the controller are stored for subsequence analysis and substituted with other peripheral data.

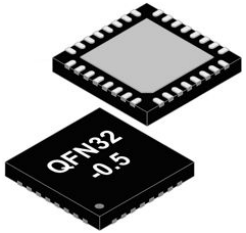
Ethernet timings

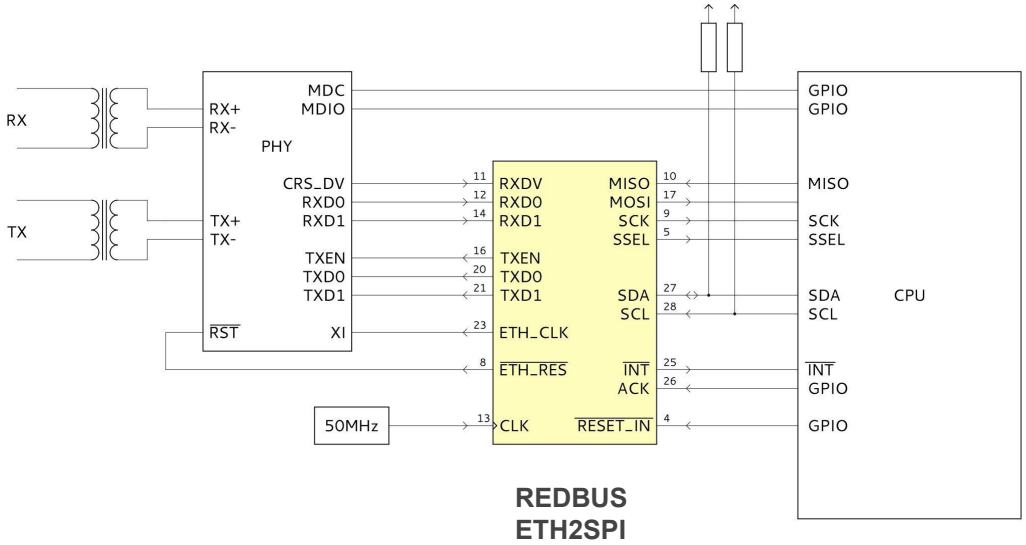




Real-time
Ethernet-to-SPI converter
inside
a very small CPLD

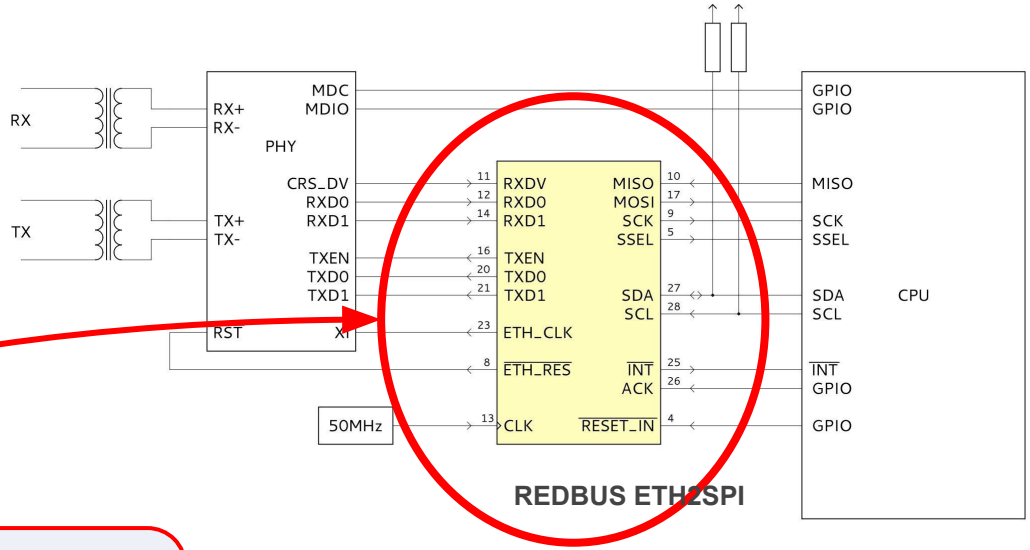
Ethernet to SPI converter





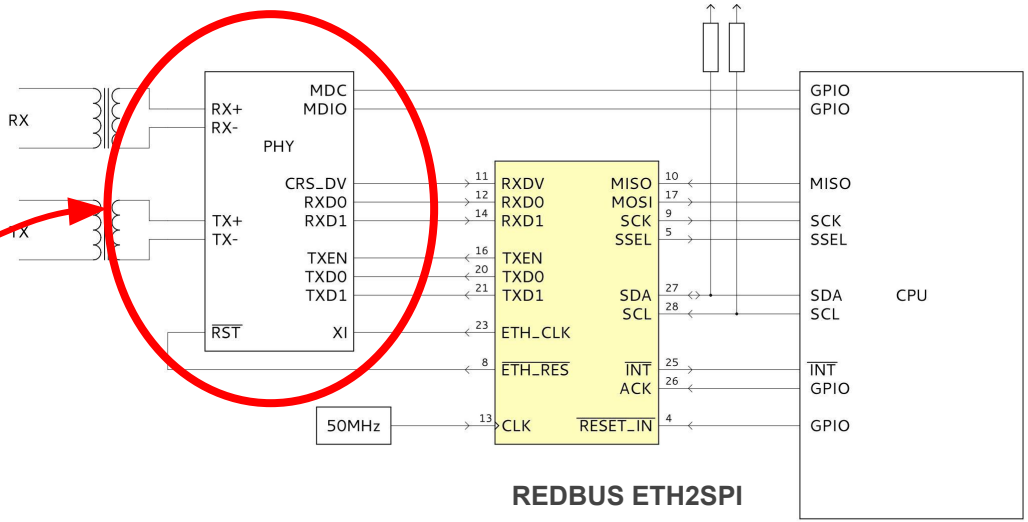
- Ultra small
- Cost-effective
- Based on a MACHXO2-256 FPGA
- QFN32 Package
- RMII interface to PHY device
- Bidirectional real-time data exchange
- I2C interface to set device address
- Up to 8192 device addressing
- Can exchange up to 44 bytes per ethernet frame per device
- Selectable 3.125, 6.25 or 12.5 MHz SPI clock

Ethernet to SPI converter



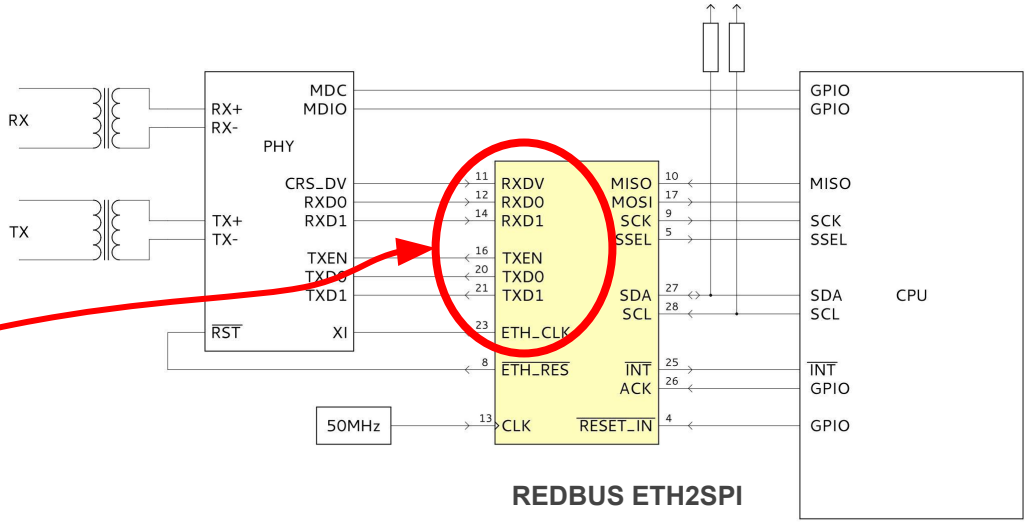
Real-time Ethernet REDBUS MAC on a MachXO2-256

Ethernet to SPI converter



Commercial, low-cost, Ethernet PHY device

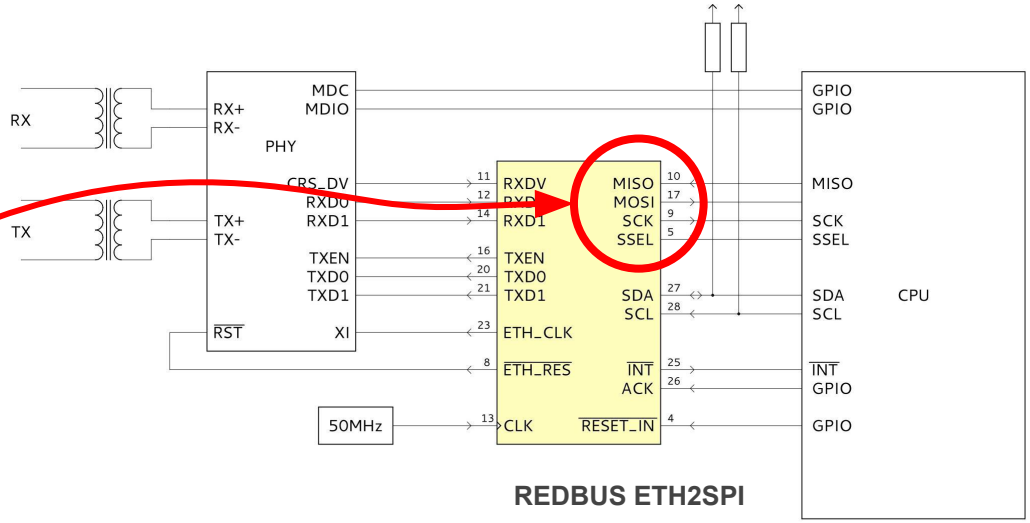
Ethernet to SPI converter



RMII Interface,

Data clocked @ 50 MHz

Ethernet to SPI converter

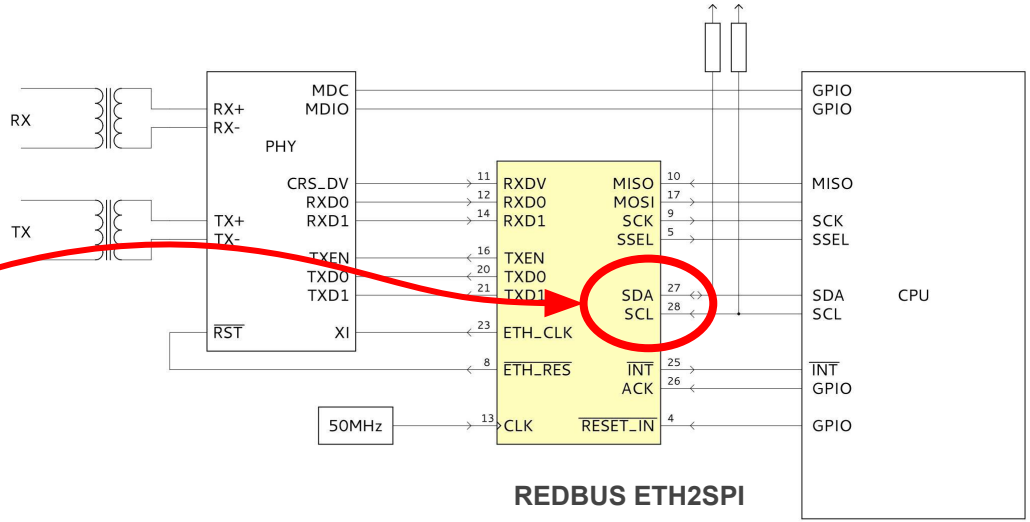


SPI: Serial Peripheral Interface

Data from/to Ethernet is clocked @ 3.125 MHz*

* devices are available on three versions of SPI clock frequency: 3.125, 6.25 or 12.5 MHz

Ethernet to SPI converter



I2C access to registers:

Address, Unique ID

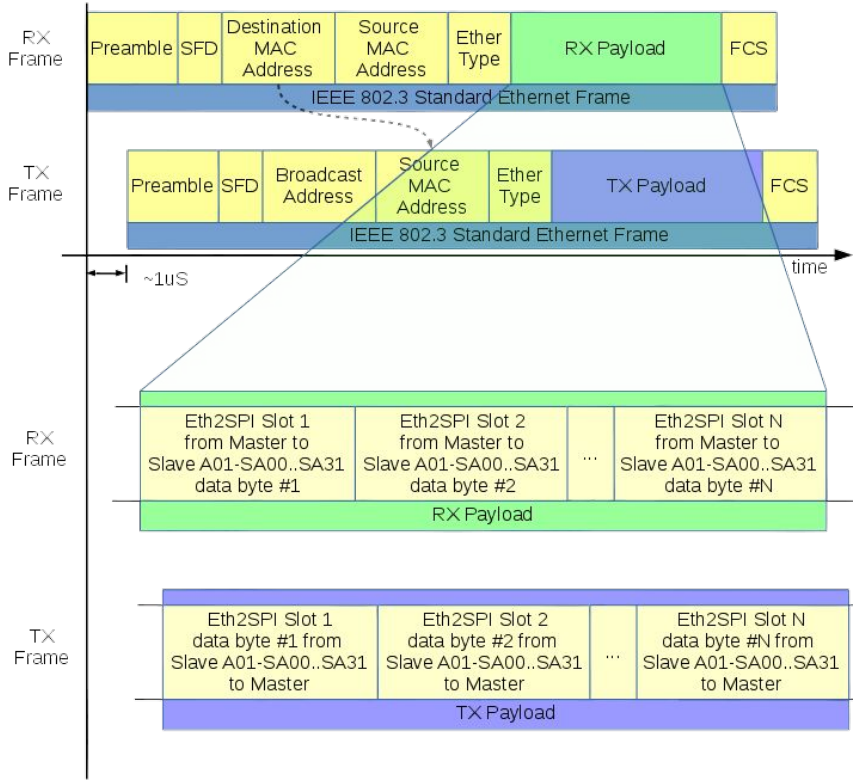
Ethernet to SPI converter

Each Ethernet frame can transport data for a plurality of device's groups. Depending on IP-Core configuration, each group may address 8, 16 or 32 devices.

Number of SUB-Address (#bit)	SPI Clock Frequency	Max bytes per device inside a single Ethernet frame	Addressable devices
8 (#3)	12.5 MHz	150	2032
16 (#4)	6.25 MHz	83	4064
32 (#5)	3.125 MHz	44	8128

For example, the 3.125 MHz version can address 254 groups of 32 devices, for a total of 8128 devices. Each group has a unique 8 bit address (1-254), each device in a group has a 5 bit sub-address ranging from 0 to 31.

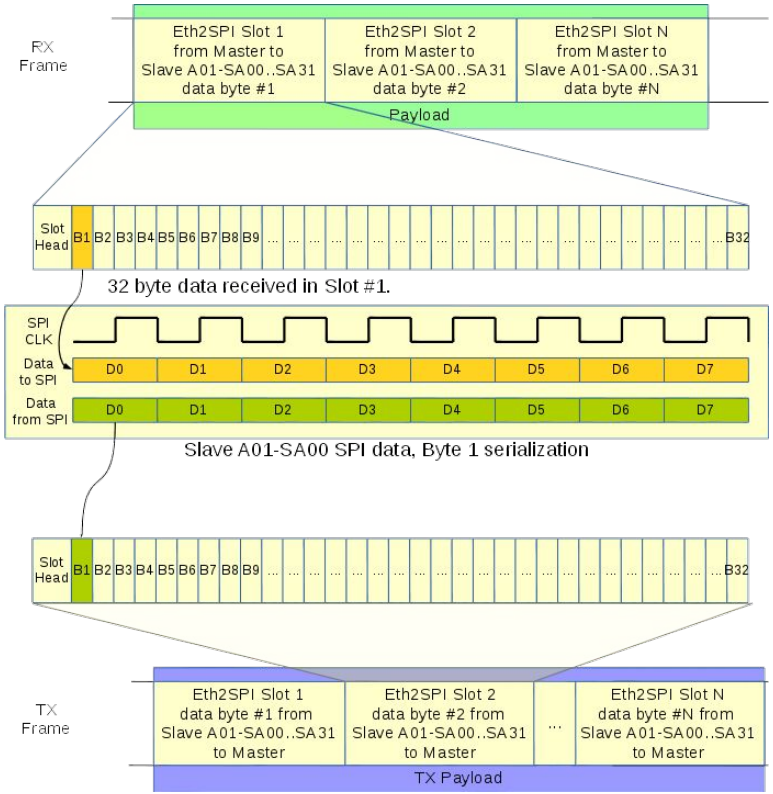
Ethernet to SPI frame



This Example shows the frame of Ethernet-to-SPI devices with an output clock of 3.125 MHz.

The Ethernet bitrate of 100 Mbps is divided by 32, that results in a SPI bitrate of 3.125 Mbps.

Ethernet to SPI frame

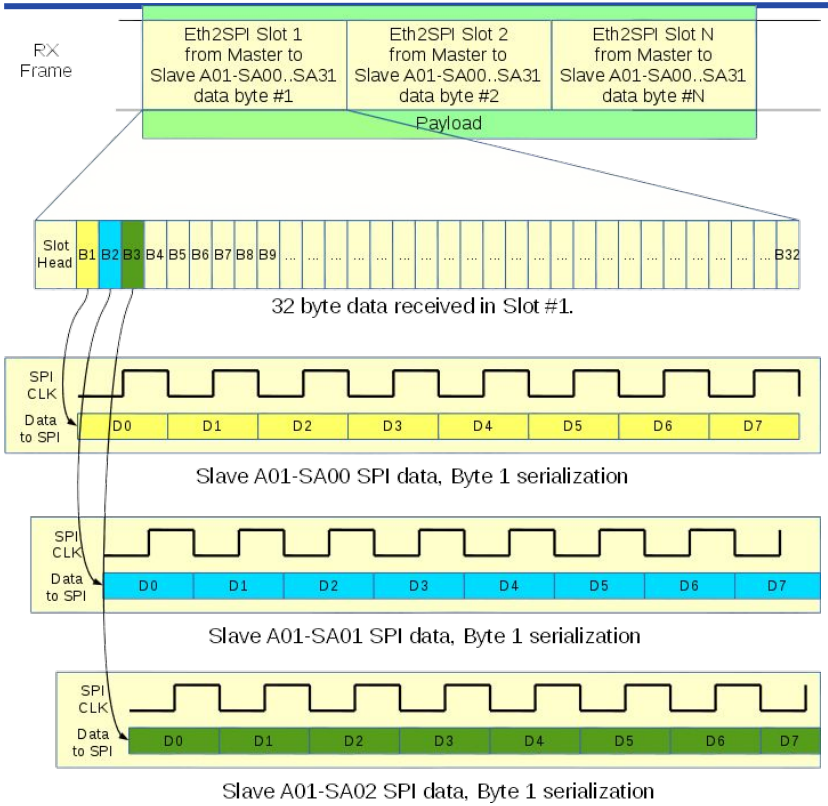


Each slot contains 32 bytes.

One byte on each slot is exchanged through SPI.

The byte offset in the slot coincides with the sub-address of a slave.

Ethernet to SPI frame



The byte 1 is exchanged by the device with sub-address 0.

The byte 2 is exchanged by the device with sub-address 1.

The byte 3 is exchanged by the device with sub-address 1.

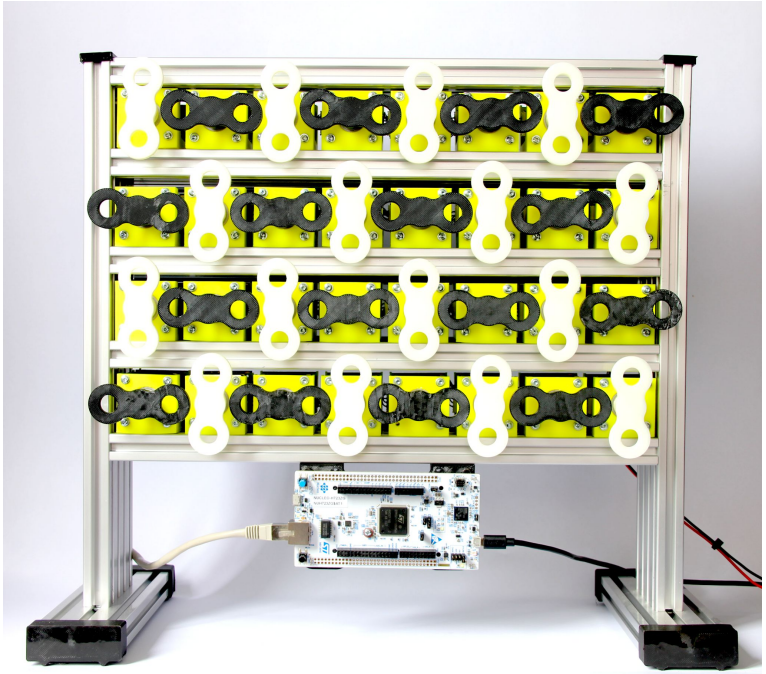
….

The byte 32 is exchanged by the device with sub-address 31.

Stepper Motor control test system

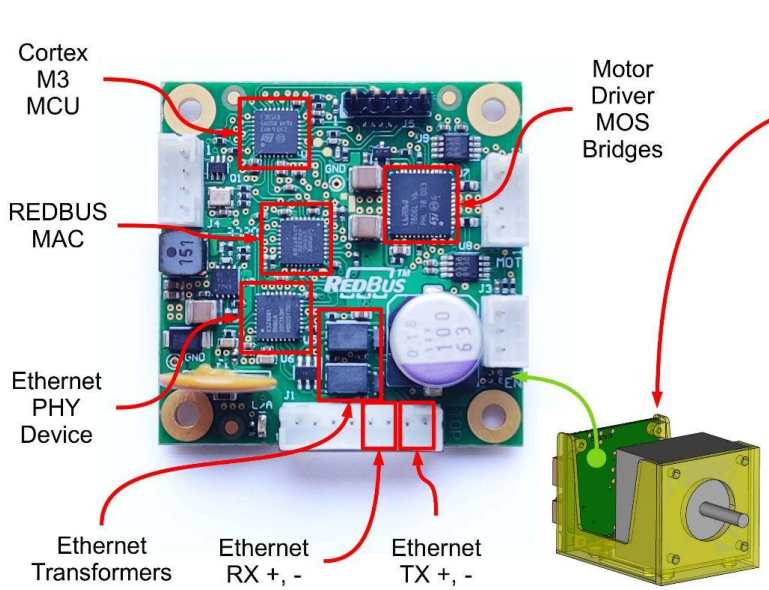
The test system is based on 32 stepper motors each driven by a small driver board. Each board is provided with a REDBUS Ethernet-to-SPI MAC that exchanges real-time data to a small Cortex-M3 processor, used to modulate and measure the motor windings currents.

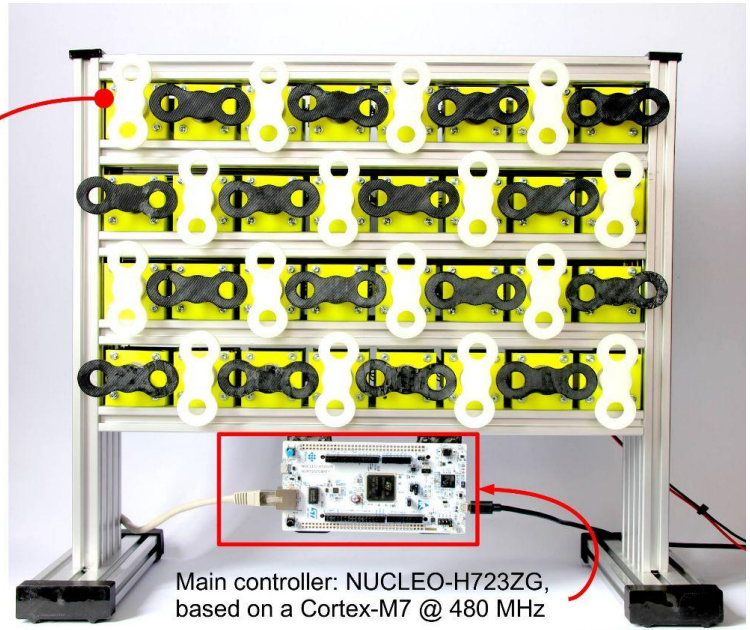
The main controller is realized on a NUCLEO-H723ZG board from ST*



** ST is a trademark of STMicroelectronics* Inc.

Stepper Motor control test system

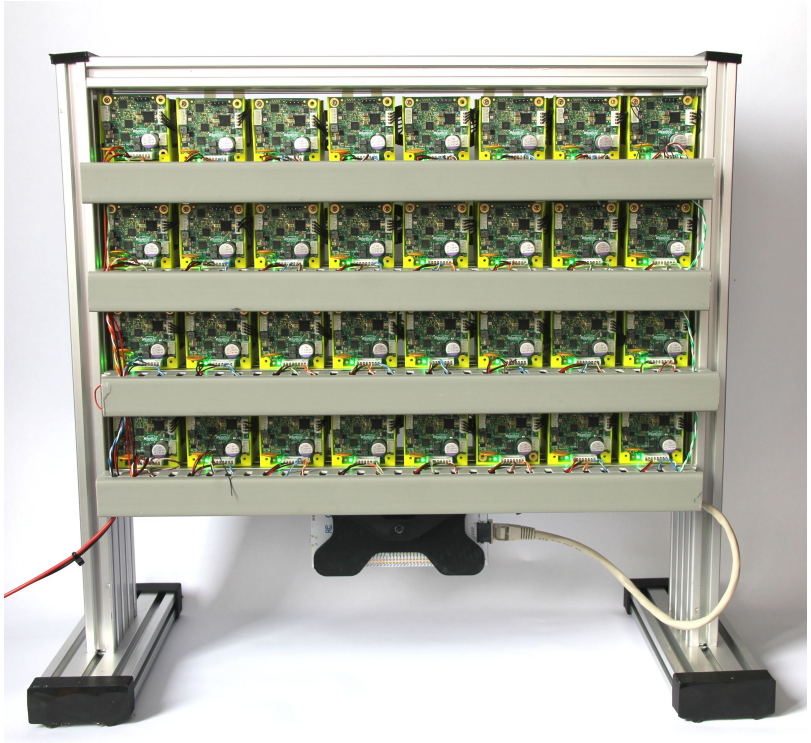




Stepper motors demo

In the rear view are visible the driver boards. A small twisted-pair cable enters and exits from each board, to form a ring chain. The first pair is derived from a standard CAT5 Ethernet cable, connected to the NUCLEO-H723ZG controller board.

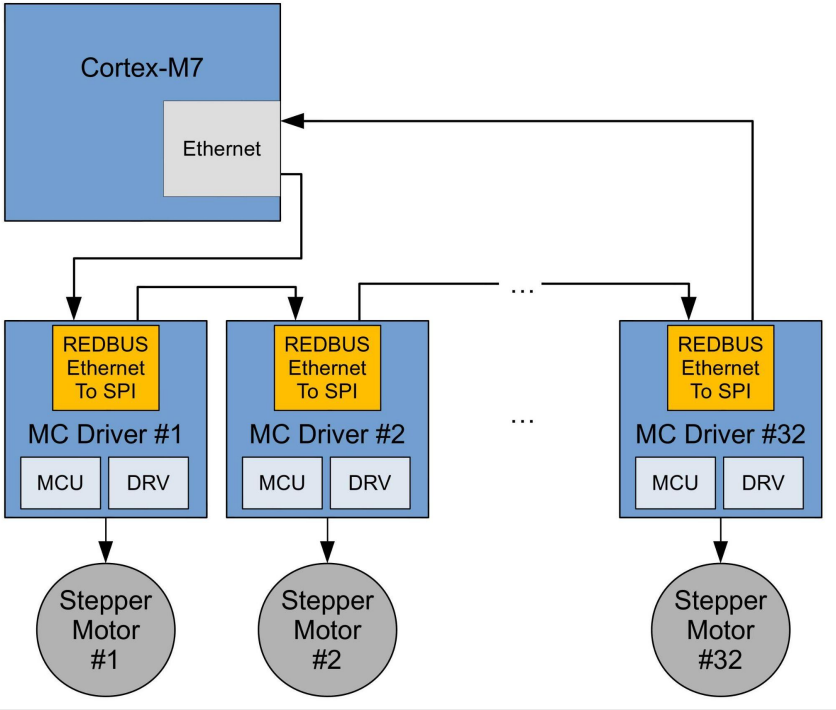
The last pair comes back to the CAT5 cable.



Stepper motors demo

The Cortex-M7 processor is the main controller. A hardware timer is used to start the Ethernet transmission every 100 µs. The Ethernet frame contains position information for all 32 motors.

The Ethernet frame traverses all the boards in about 50 µs and returns to the controller board after collecting the real-time measurements made by the driver boards.



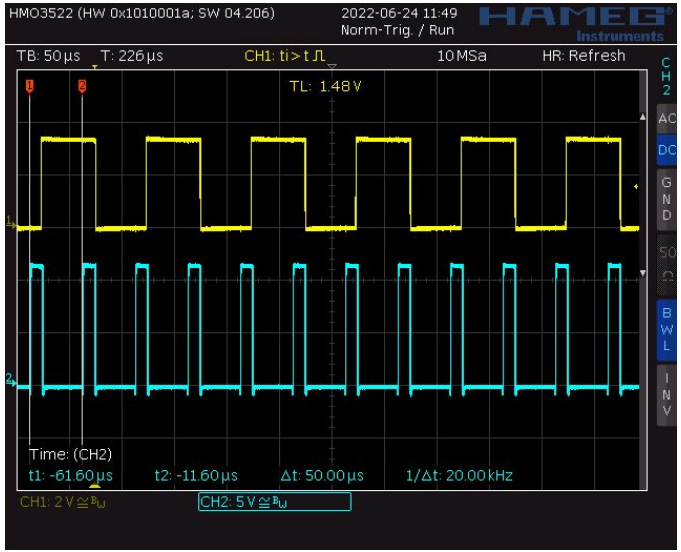
Stepper motors demo

The driver boards use PWM to regulate the windings currents. The PWM frequency is set to 20kHz, corresponding to a period of 50 µs.

The microcontroller firmware constantly measures the phase of the PWM counter each time it receives new data from the Ethernet, adjusting the PWM period to keep it in phase with the Ethernet frame.

In this way, all the PWM cycles of the 32 MC driver boards are kept in phase.

The yellow track in the figure represents the RMII_RXDV signal, and the cyan one the PWM on a motor phase.

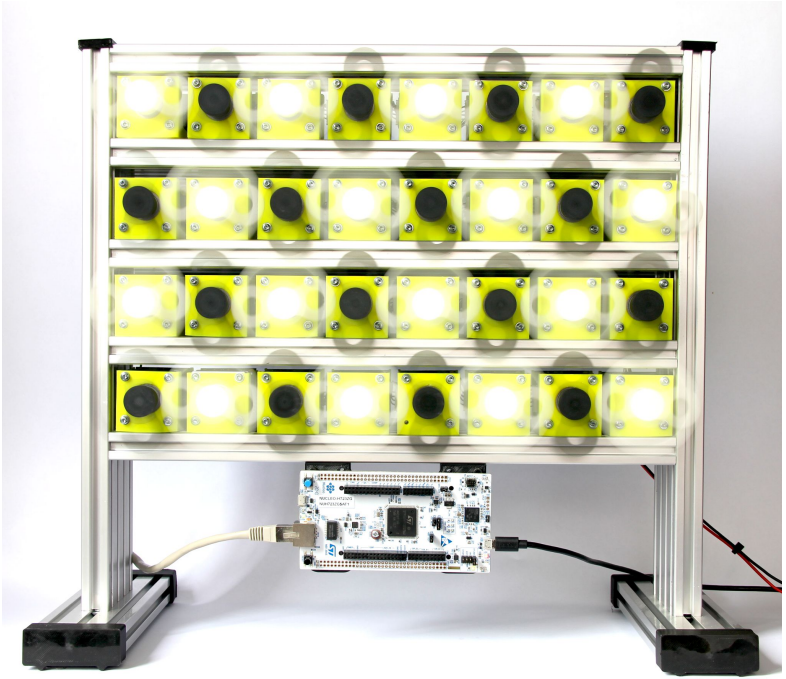


Stepper motors demo

Each motor is independent by each other, the propellers are clamped directly onto the motor shaft and arranged so that they interfere with each other in case of non-synchronous rotation.

The main controller generates the position data for each motor and sends it all in a single Ethernet frame every two PWM cycles.

The real-time and synchronous control of the motors enables speeds up to more than 5000 RPM without propellers collisions.



The above picture has been taken with a shutter of ⅓ second. During the shot, the flash was fired to highlight a still image of the propellers rotating at 5500 RPM. Please follow this link to watch the video: https://youtu.be/WP_cJl7eKDI.

Stepper motors demo

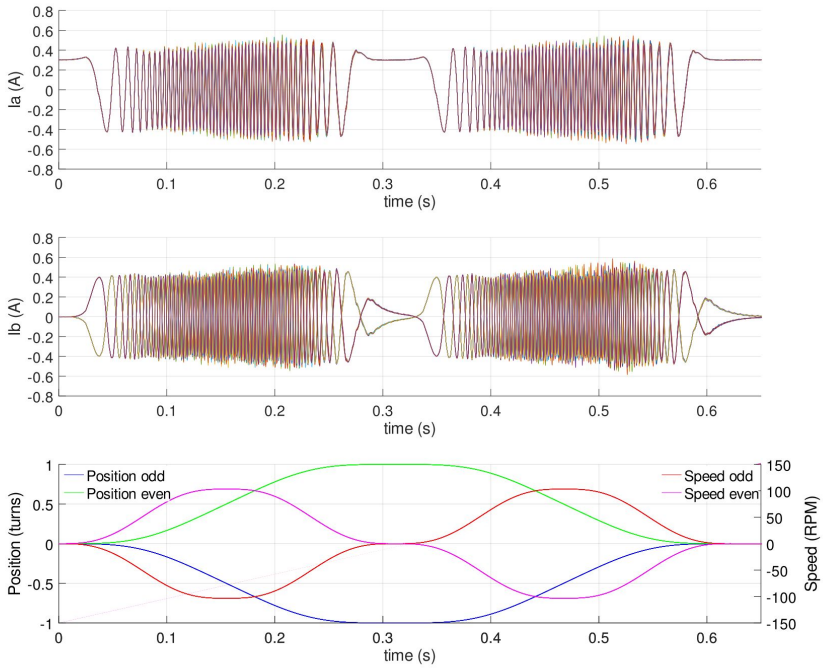
The driver boards measures and regulate the current on the motors. The measures are sent back in the response payload of the returning Ethernet frame.

The controller collects the real-time data returned by all the driver boards, every 10 µs (10k samples/s).

In the following plots, the real-time current of all the 32 motors is reported, it was captured after a fast synchronized movement of 1 turn forward and backward.

In the Ia plot, all the currents present the same phase. In the Ib plot, the currents have a phase shift of 180° depending on motor direction.

The odd-numbered motors started in a counter-clockwise direction, the even-numbered ones in a clockwise direction.

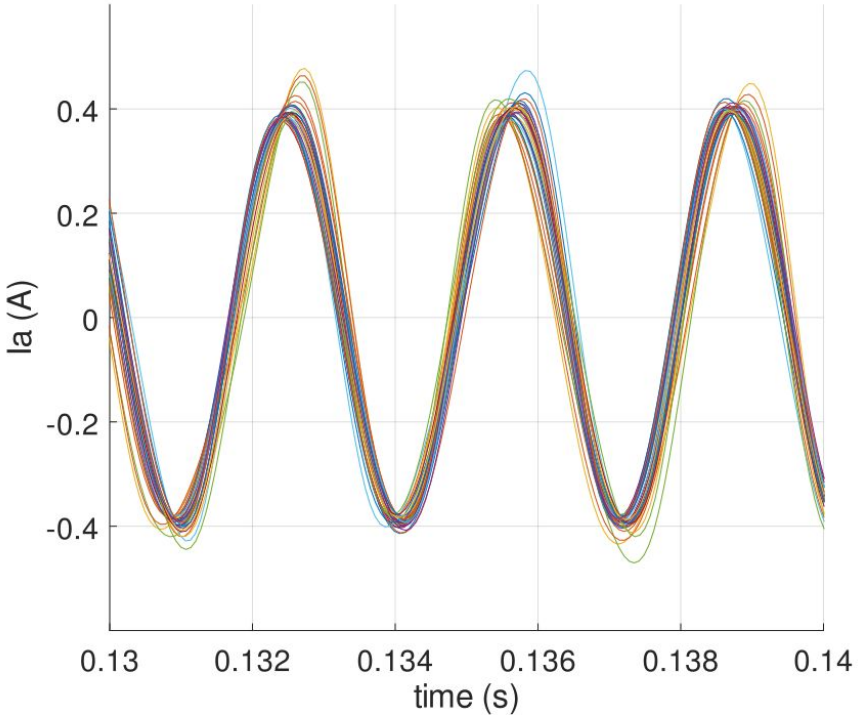


Stepper motors demo

The perfect overlapping of the waveforms confirms the excellent synchronism between the motors.

This plot shows a zoom-in of the winding A currents when the 32 motors run at about 100 RPM.

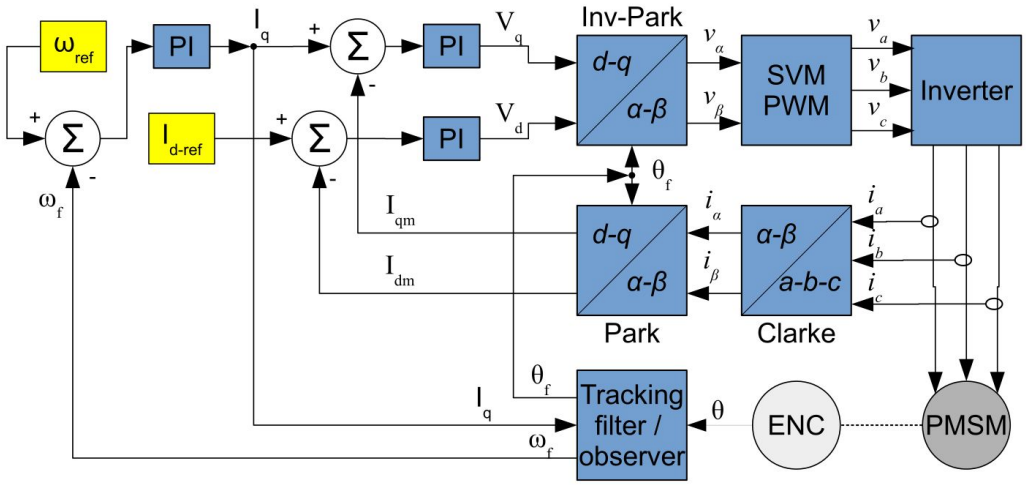
All 32 simultaneous measures have been reported.



BLDC motor control with Field Oriented Control (FOC)

Typical brushless motor control with FOC requires several calculation algorithms to maintain the vectors of the driving currents in quadrature against the magnetic flux of the rotor.

For example, some PI algorithms determine the driving current or keep the derived current vector in quadrature with the rotating magnetic flux vector.



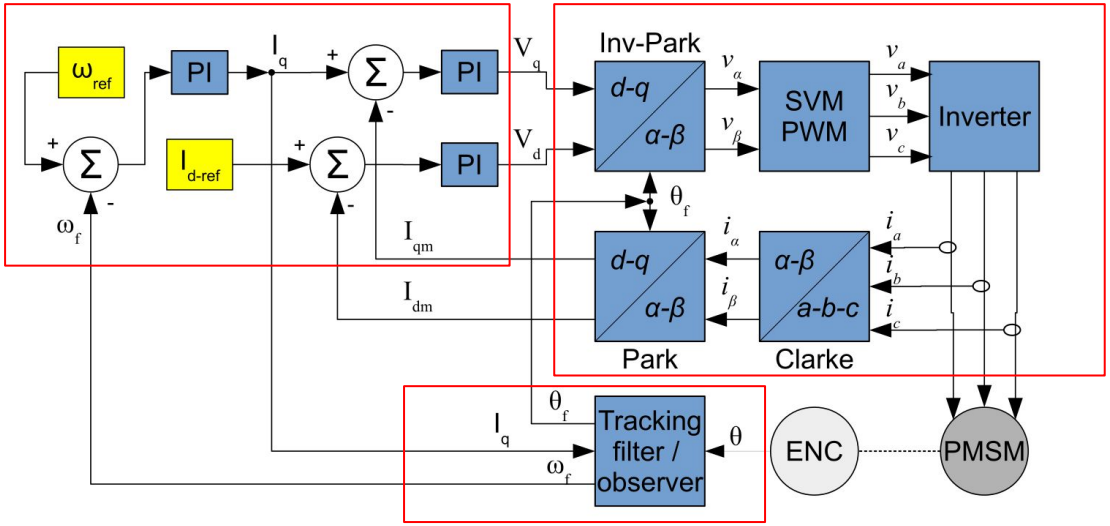
BLDC and FOC demo

Real-time Ethernet allows data to be transferred easily between devices and grants strict timing requirements.

The minimum loop time required by the FOC algorithms depends on the modulation frequency of the Inverter, for example, 20 kHz, which implies a loop period of 50 µs.

The REDBUS system supports data exchange on several peripherals within 50 µs or less.

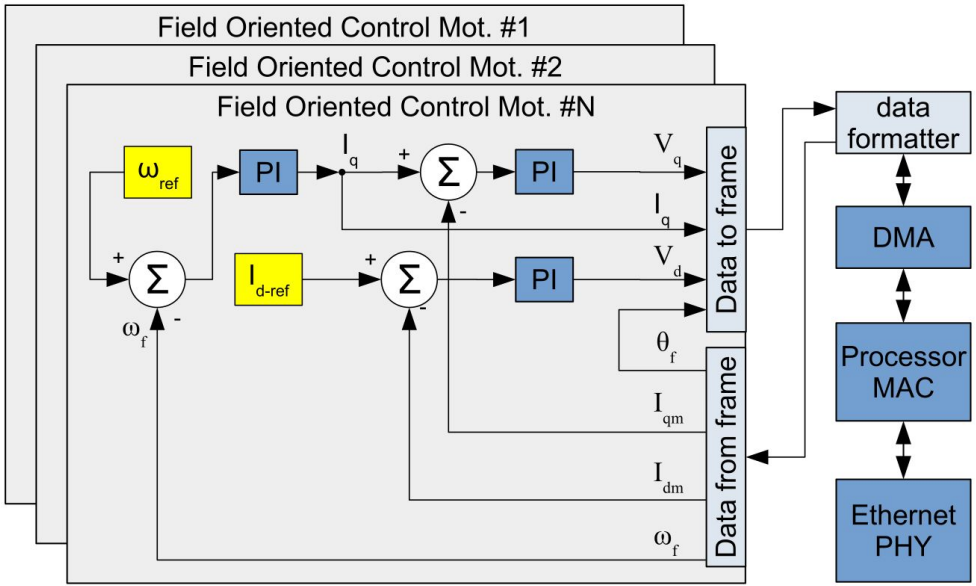
In our test system, we've split the FOC into three groups of algorithms executed in different processors on remote boards.



Group 1: the main controller

A main controller based on a NUCLEO-H743 board from ST is used to compute the PI algorithms and determine the Vd, Vq, and Iq vector modules for up to eight motors.

The embedded Ethernet MAC of the Cortex-M7 processor is used as the primary interface.



Group 2: The driver boards

Each driver boards receive the Vd, Vq, Iq and flux angle.

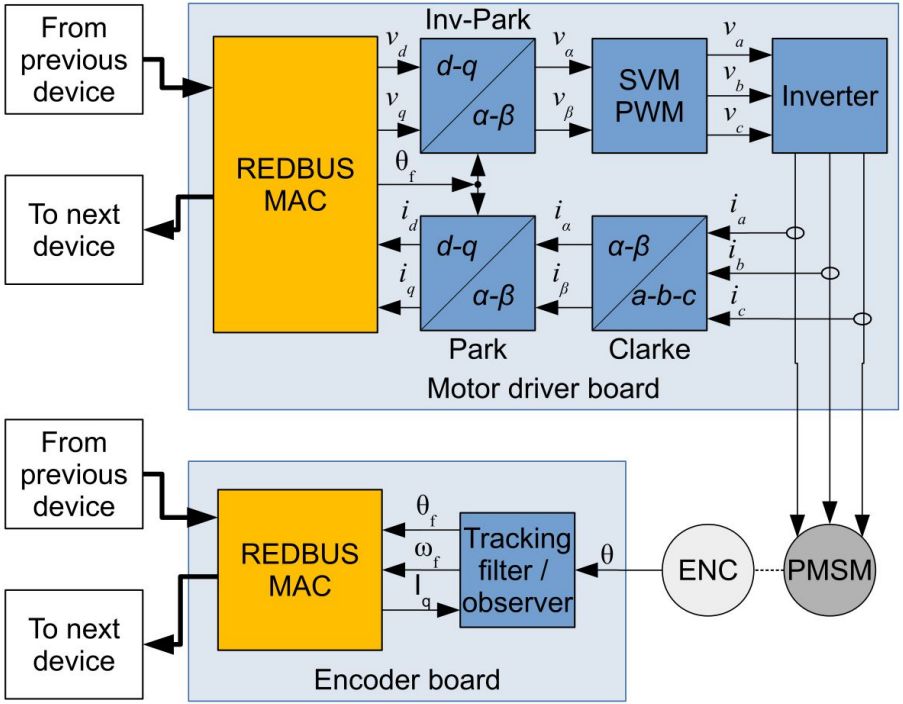
The Inv-Park and SVM modulation blocks are use to generate the three phase voltages on the BLDC.

The winding currents are read and sent back on each Ethernet frame.

Group 3: The encoder boards

A small board is used to interface an analog encoder, based on a magnetic ring and an AMR sensor that convert the magnetic field into two sine and cosine signals.

A tracking filter is also provided to predict and reduce position noise and to derive the speed value.

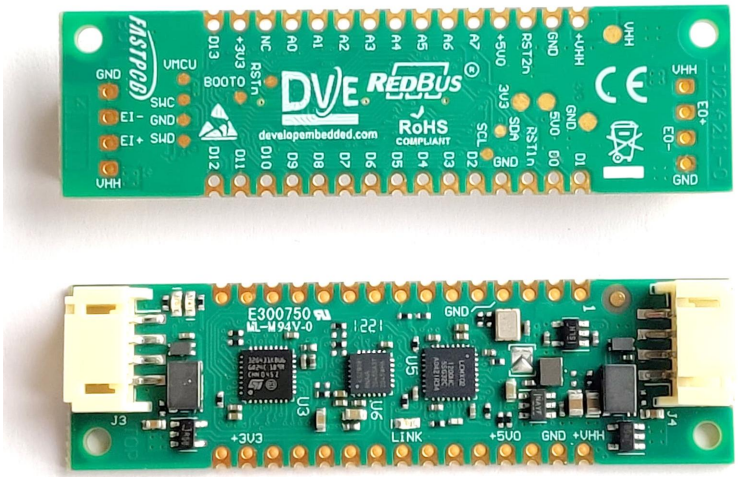


BLDC and FOC demo

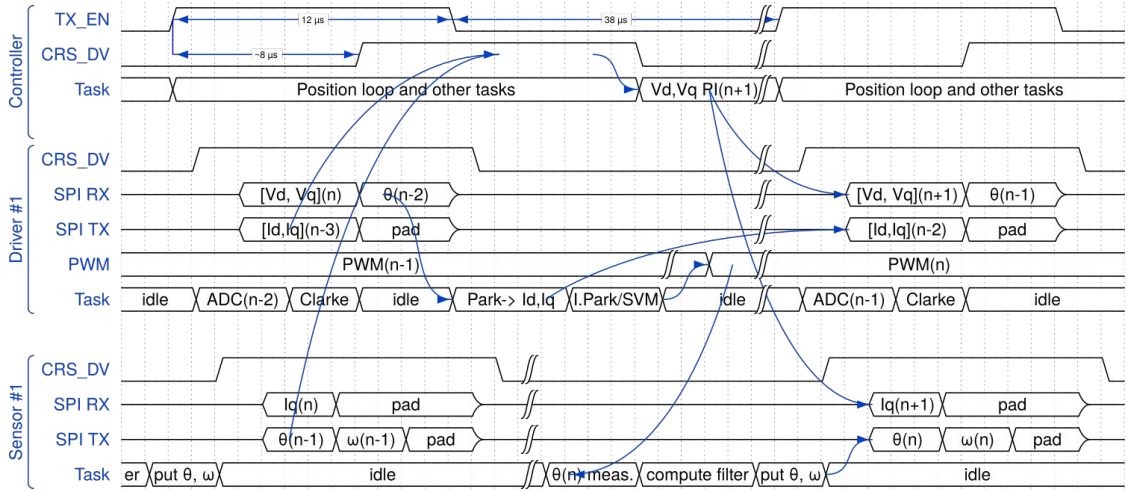
This is the board used to interface the AMR sensor, it is based on REDBUS Ethernet-to-SPI MAC and an STM32G431 microcontroller.

A single twisted pair with the Ethernet TX signals from the previous device enters the J3 connector, on the left. The PHY device U6 is visible on the center of the board, the processor U3 on the left. The REDBUS Ethernet-to-SPI MAC is placed on the right, marked as U5.

The Ethernet frame is re-transmitted on a second twisted pair placed on the J4 connector on the right.

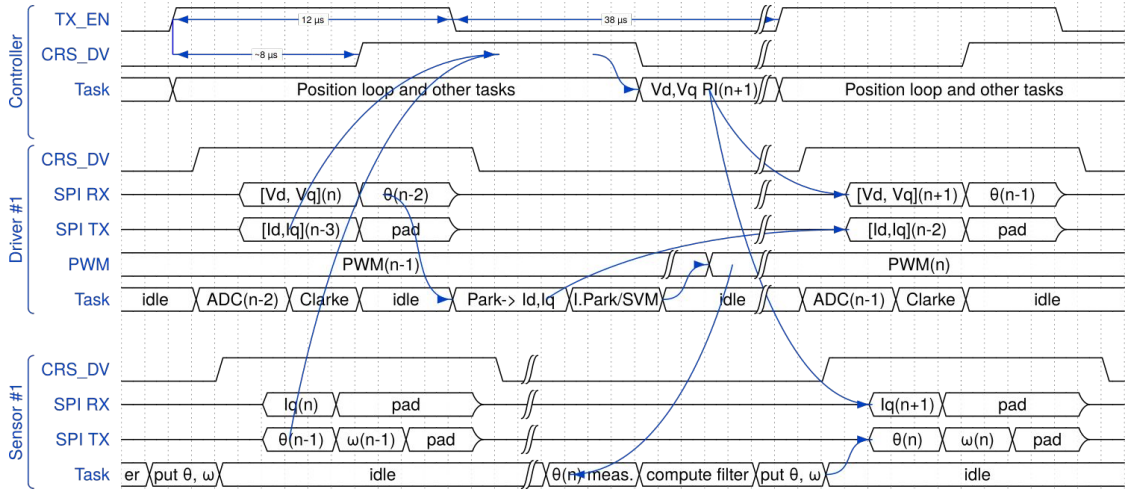


BLDC and FOC demo



The first driver board receives the Vd, Vq driving parameters, and the θ angle that is derived from the tracking filter algorithm of the encoder board. The next PWM signal will consider those new driving parameters. The driver board replies by sending the Id, Iq measures, after computing the Clarke and Park algorithm to convert the three measured stationary currents into two orthogonal vectors in the rotating reference frame.

BLDC and FOC demo



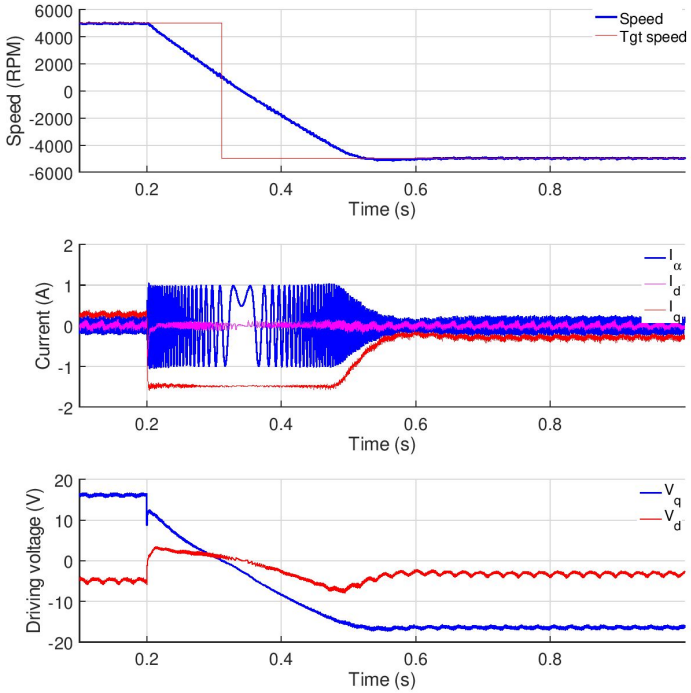
The encoder board receives the commanding current Id, which will be used in the tracking filter to predict the next position, corrected by the measured angle θ. In the reply data, the filtered angle θ and the angular speed ω will be sent back to the controller board. The measures from the driver board and the encoder board will be used by the PI algorithms in the controller to compute the next Vd, Vq. The same behavior happens in the driver and sensor boards 2, 3, and 4, not included in the figure.

BLDC and FOC demo

The figure shows some captured data of a BLDC motor when the requested speed has been changed in sign, causing the inversion of the rotation.

The target speed was set to 5000 RPM and then to -5000 RPM.

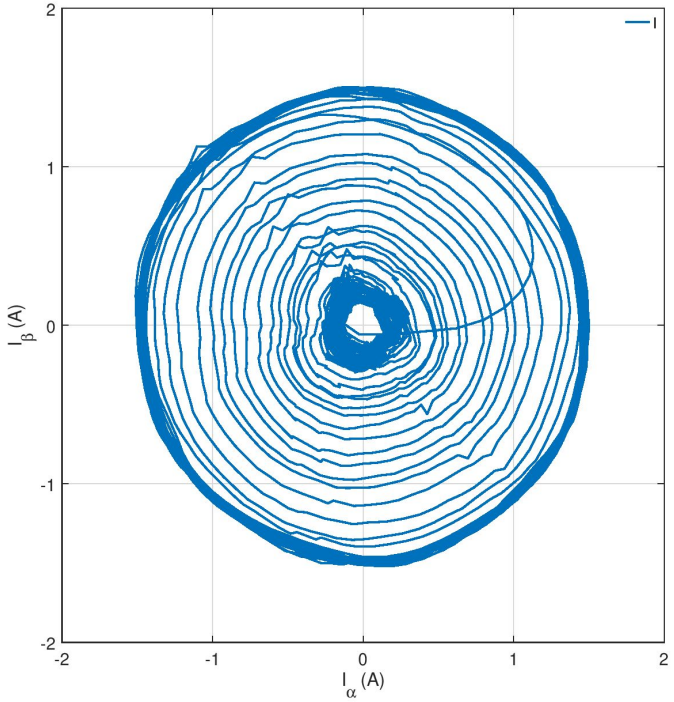
The control loop is closed in 50 microseconds even if the tasks have been split into three different boards.



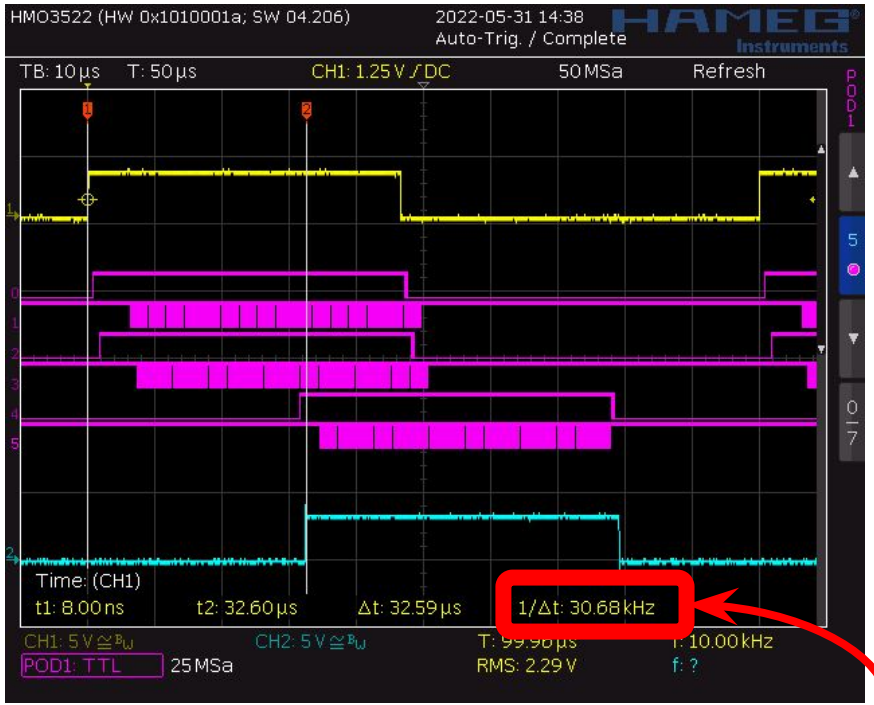
BLDC and FOC demo

This is an X, Y representation of Ia and Ib currents during a BLDC test running at 4000 RPM CW and a change in direction on the opposite side, 4000 RPM CCW.

The currents change from about 0.2A to 1.5A, initially, to brake the rotation, then to accelerate it. Then, the currents go down at 0.2A when the speed has reached the setpoint.



Measures



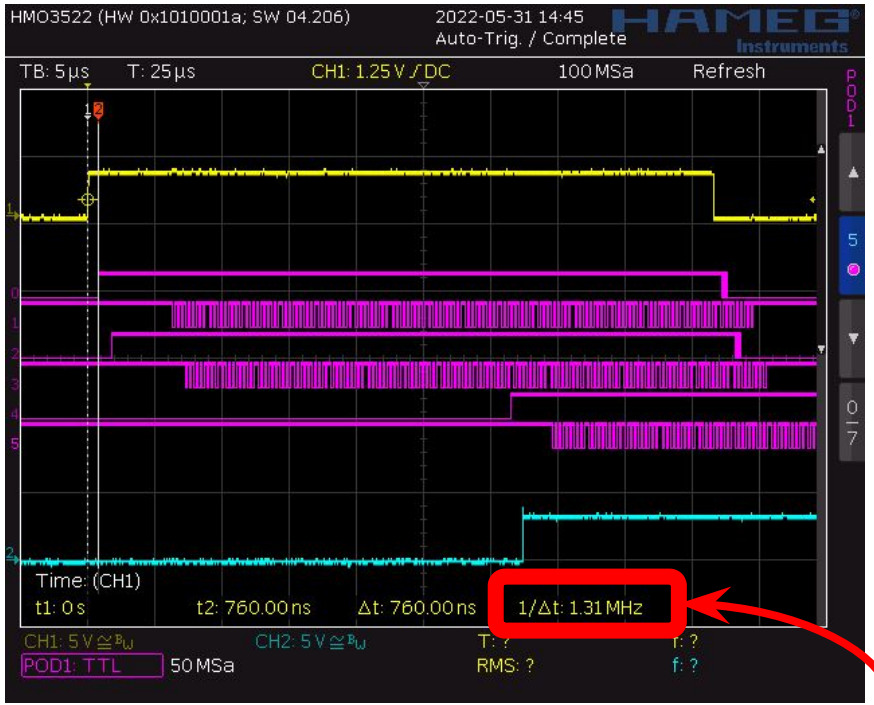
The total latency is 32.59 µs.

Latency measurement on a system based on a main controller and 32 REDBUS based devices.

The yellow track is connected to the RMII_TXEN signal of the main controller (host). It is active when the CPU starts Ethernet transmission.

The cyan track is connected to the RMII_CRSDV signal of the main controller. It is active when the Ethernet frame is received back by the PHY device on the host.

Measures



The latency on first device is 760 ns.

Latency measurement on the first REDBUS based device.

The first magenta track is connected to the RMII_CRSDV signal on the first REDBUS device.
The second magenta track is connected to the SPI_SCK signal on the first REDBUS device.
The data is serialized in and out on the rising edge of this clock.

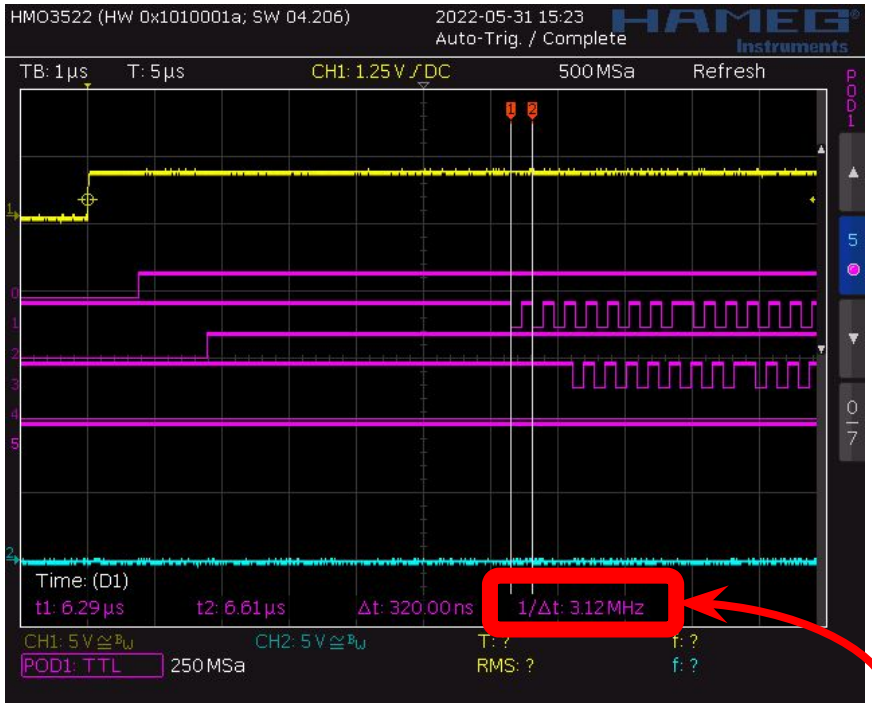
The third and fourth magenta tracks are connected to CRSDV and SCK signals on the second REDBUS device.

The last two magenta tracks are connected to CRSDV and SCK signals of last device.

Measures

Latency measurement on the last (32th) REDBUS based device.

The latency on 32th device is 31.5 µs.



Zoom in on the SPI_SCK signal and frequency measurement on the first device.

The SPI_SCK frequency is 3.125 MHz.

email: info@developembedded.com

www.developembedded.com

Thanks for your attention