

Real time communication protocols

PN202 | Posted on January 13, 2023 | Updated on February 9, 2026



Nicolas CHERIX

Head of Engineering

imperix • in

Table of Contents

- [Fully-supported real-time communication protocols](#)
 - [Ethernet](#)
 - [CAN](#)
 - [RealSync](#)
 - [Serial communication](#)
 - [Custom communication protocols](#)
- [Recommended protocol by application scenario](#)
 - [In-between imperix controllers](#)
 - [From/to third-party devices \(e.g. PLC\)](#)
 - [From/to a user-defined GUI on a computer](#)
 - [For interfacing with additional hardware](#)

Imperix converter controllers (e.g., B-Box family, B-Board PRO, and TPI) utilize various real-time communication protocols to enable coordination and interaction with external devices. In this context, this article aims to:

- **Present** the real-time communication protocols, outlining their intended purpose and required implementation effort.
- **Clearly differentiate** between the available protocols.
- **Assist** users in deciding the optimal approach for their application.

Fully-supported real-time communication protocols

[Imperix controllers](#) provide support for the following real-time communication protocols. Despite minor differences, this support is mostly identical among products. Details on each protocol are given below.

Interface	B-Box 4	B-Box RCP / B-Board PRO	TPI 8032
Ethernet 1x RJ45	– UDP – Modbus TCP (2026.2) – OPC-UA	– UDP – Modbus TCP (2026.2) – OPC-UA	– UDP – Modbus TCP (2026.2) – OPC-UA
CAN 2x RJ45	Two busses supporting – CAN 2.0A, CAN 2.0B, CAN FD	One bus supporting – CAN 2.0A, CAN 2.0B	One bus supporting – CAN 2.0A, CAN 2.0B
Serial 2x RJ45	Two busses supporting – EnDat 2.1, 2.2 / SSI / BiSS-C	Not supported	Not supported
RealSync QSFP+/SFP+	4x QSFP+ (40Gbps) sockets	3x SFP+ (10Gbps) sockets	3x SFP+ (10Gbps) sockets

Supported real-time communication protocols on the different imperix controllers

These technologies, although all considered as real-time protocols, perform unequally in control applications, especially in power electronics. The table below provides some rough performance figures in this respect.

	Latency	Refresh rate	Payload size	Deterministic
UDP over Ethernet	Typ. 1-10 ms	Typ. 0.1-1 kHz	up to 1024 bytes Typ. < 100 blocks	NO
Modbus over TCP/IP (i.e. Ethernet)	Typ. 1-10 ms	Typ. 0.1-10 Hz	up to 253 bytes	NO
CAN	Typ < 1 ms	Typ. 0.1-1 kHz	up to 8 bytes (CAN)	NO (almost)

			up to 64 bytes (CAN FD)	
OPC UA over Ethernet	Typ. > 10ms	Typ. 0.1-100Hz	Typ. 4 bytes	NO
RealSync Native exchanges	Typ. 0.1-1 us (sub-interrupt)	Typ. 0.3-300kHz	Typ. < 200 bytes	YES
RealSync User-defined exchanges	Typ. 1-10 us (next interrupt)	Typ. 0.3-300kHz	=32bits per block Typ. < 100 blocks	YES
EnDat / SSI / BiSS-C	Typ. 1-10 us	Sensor-dependant	Typ. < 40 bytes	NO

Comparative summary of the supported real time communication protocols for various performance criteria.

Ethernet

Ethernet is used as the physical layer for several protocols listed below:

- **UDP** is the preferred choice for non-critical user-defined data exchanges between controllers.
- **TCP** is widely employed for communication between the controller and the host PC. It is notably widely used by [Cockpit](#).
- **Modbus** (over TCP/IP) is an application layer (OSI layer 7), vendor-agnostic, royalty-free protocol specifically intended for automation equipment. Originally developed in the early '80s, it has since become a de-facto standard fieldbus protocol for [PLCs](#) and within [SCADA](#) systems.
- **OPC UA** is an open communication standard for data exchanges between interoperable industrial systems. Compared to Modbus, it offers somewhat more modern capabilities, notably regarding security or the handling of complex systems. On imperix controllers, OPC UA is exclusively supported over Ethernet.

Since transmission delays are non-deterministic over Ethernet, latency cannot be guaranteed. However, its high maximum throughput (typically 1 Gbps) ensures that transmission delays aren't much dependent on data volume (much less than CAN, for instance). This translates to millisecond-scale data transfer delays for up to

dozens of words over a wired connection. Therefore, provided a data transfer latency greater than the refresh period is acceptable, **UDP/IP** can generally be employed with refresh rates in the kHz range.

[OPC UA](#) is used for the real-time configuration and monitoring of imperix controllers from the host PC. It is notably used by [Cockpit](#). Besides, the same interface is also available to third-party clients, such as computer software or dedicated hardware (e.g. [PLCs](#)). More information on this topic is given in [PN177](#). **OPC UA** isn't particularly optimized for speed. It is therefore not recommended for closed-loop control applications.

Modbus (more specifically Modbus TCP) is supported for compatibility with PLCs and other third-party controllers. This protocol is widely supported and convenient for supervision and configuration, but is not well-suited for hard real-time applications.

CAN

CAN is a widespread field bus communication standard. Its support on imperix hardware differs slightly:

- **CAN 2.0** is available on all controllers. It supports both standard (CAN 2.0A, 11-bit identifiers) and extended (CAN 2.0B, 29-bit identifiers) formats, with a configurable bitrate up to 1 Mbps.
- **CAN FD** is available on the [B-Box 4](#) only. It supports standard and extended identifiers, payloads up to 64 bytes, and a configurable data bitrate of up to 5Mbps.

While CAN is inherently non-deterministic due to potential data conflicts, it achieves near-deterministic performance when only one master is guaranteed to transmit data and transmission failures can be avoided. Under these conditions and for small payloads, CAN can be reliably used within closed-loop control applications up to few kHz. CANopen is, however, not (yet) supported on imperix products.

RealSync

Imperix [RealSync](#) is a proprietary protocol built on top of AMD/Xilinx **Aurora**. It is used for ultra-low latency data exchanges between controllers. This protocol also natively implements synchronization ($\pm 2\text{ns}$) between controllers. At the hardware level, RealSync is supported by two connector form factors:

- **SFP+** is available on B-Box RCP 3, B-Board PRO, and TPI8032. Three sockets are available, supporting up to 10Gbps transfers each.
- **QSFP+** is available on B-Box RCP 4. Four quadruple sockets are available, supporting up to 4x10Gbps transfers per socket. Fork cables (one QSFP+ to four SFP+ connectors) are available for inter-connectivity between units of different generations.

RealSync is heavily optimized for achieving an extremely low data transfer and low latency. This is notably achieved thanks to data aggregation (respectively de-aggregation) for upstream transfers (respectively downstream transfers) along the network tree. Furthermore, all transfers are 100% deterministic: data transfers never exceed the planned delay.

RealSync can be involved (even simultaneously) in two types of data exchanges:

- Native data exchanges, such as those occurring in **master-slave** configurations (also called **I/O extension mode**), between the operating CPU and FPGAs (read more [here](#)) at the beginning (upstream) and at the end (downstream) of each control period. These data transfers are configured automatically and are entirely transparent to the user.
- Manually-defined data exchanges using the SFP_in and SFP_out blocks (or routines), occurring between two CPUs in [multi-master operation](#). In this case, the data transfers are configured by the user as a function of the application-level requirements.

Transmission delays are perfectly deterministic in both cases. Native data exchanges are largely sub-microsecond, while manually-defined transfers are updated at the next interrupt cycle. **RealSync** is perfectly suited for data transfers involved within closed-loop control up to several hundred kHz. More information on latency can be found on the [technology page](#).

Serial communication

Serial communication with digital motor encoders is available on B-Box 4, thanks to two distinct busses. Support for the following protocols is currently implemented:

- **SSI** is a serial interface standard, originated in the 80s', for the interfacing of absolute position encoders. Based on RS-422 clocked at up to 2MHz, SSI remains widely used, although BiSS-C now offers superior performance and capabilities.
- **BiSS-C**, introduced in 2007, somewhat adds bidirectional transfer capabilities to SSI (over RS-485) as well as an increased maximum clock rate (max.

10MHz) and support for CRC. It is now an open standard, widely adopted across the industry.

- **EnDat 2.1 and 2.2** are similar standards released by Heidenhain in 2003, and adopted by many other manufacturers since then. It operates at up to 16 MHz in full-duplex mode. Sensors with EnDat 2.2 interfaces generally offer rich configuration and diagnostics information as well as simple configuration.

In general, the choice of the encoder protocol is guided by the selection of the position or speed sensor in the first place. Nonetheless, when flexibility exists, the clock rate should be considered a key indicator of the transmission speed and, therefore, of the achievable angular resolution.

Custom communication protocols

For applications requiring non-standard interfaces or performance levels beyond native support, imperix controllers offer a dedicated **FPGA sandbox**. This user-programmable area allows engineers to implement custom drivers, such as for specialized sensors or high-speed serial links, directly on the FPGA using Vivado Design Suite.

Communication between the custom FPGA logic and the CPU is handled transparently. Custom registers are accessible through the dedicated [SBI](#) and [SBO](#) blocks part of the ACG SDK, enabling seamless integration with Simulink or PLECS models. Advanced users can also repurpose the SFP/QSFP+ ports for custom Aurora-based communication with third-party hardware.

For further details on this workflow, refer to [PN169: FPGA development on imperix controllers](#) and [PN159: Getting started with FPGA control development](#).

Recommended protocol by application scenario

In-between imperix controllers

This would typically take place in stacked controller configurations, or in-between parallel-operated converters.

1. **Use RealSync!** Performance is a few hundreds to millions of times superior to that of Ethernet or CAN depending on the considered criteria.
2. The choice of native vs user-defined data transfers is usually imposed by the application (see [distributed converter control](#)). However, native transfers

perform best and should be preferred whenever possible.

3. In case RealSync couldn't be used for some reason, then the backup choices would be Ethernet or CAN.

From/to third-party devices (e.g. PLC)

This typically corresponds to situations where an imperix controller must receive instructions or set points from a [PLC](#), or respectively return basic status information or measurements.

1. If the device supports OPC UA as a client (not as an OPC UA server), then **OPC UA is the most comfortable** to use in the long run. Indeed, as it isn't even necessary to place blocks/functions in the developed code, OPC UA offers the deepest and easiest access to the running software.
2. If OPC UA can not be easily supported, two cases should be distinguished:
 - o If numerous variables (typically >20) must be read or sent at once, and if time determinism isn't absolutely essential, then **Ethernet could be preferred** due to its superior bandwidth.
 - o Opposedly, if only a few variables must be exchanged and if packet arbitration can be avoided, then the lower (and more constant) latency of **CAN** could make it a better choice.
3. If none of the above protocols is usable, then analog inputs/outputs or custom serial protocol implementations should be considered.

OPC UA	UDP/IP	CAN
<ul style="list-style-type: none">++ Instant access to all variables+ No efforts on the control side+ Access to auxiliary routines (start, stop, enable, etc.).- Non-constant latency	<ul style="list-style-type: none">++ Generally well supported by 3rd party devices+ Good for exchanging numerous variables- Non-constant latency	<ul style="list-style-type: none">++ Relatively constant latency- Low total data bandwidth (1Mbaud)- CANopen not available
OPC UA documentation	UDP_in , UDP_out	CAN_in , CAN_out

Comparison of supported protocols for communication with 3rd-party devices.

From/to a user-defined GUI on a computer

In this case, two cases should be distinguished:

- Using the [GUI builder](#) in Cockpit. In this case, OPC UA is used and data exchanges occur automatically, similarly to other variables read/write accesses in Cockpit.
- Implementing an independent GUI, for instance using [Matlab App Designer](#). In this case, OPC UA is recommended, but not mandatory. Further information is provided in [TN175](#).

[Imperix Cockpit](#) or a user-defined GUI. Regarding real-time communication, the above-presented selection procedure for third-party devices also applies. Furthermore, as **OPC UA** becomes widely supported with easy-to-use libraries, this option shall be privileged whenever possible.

For interfacing with additional hardware

This case occurs when implementing communication with integrated circuits or similar devices, such as ADCs, sensors, etc. Often, with the exception of motor drive applications (see below), a device-specific peripheral driver must be implemented somehow.

1. Implement **whatever is supported** by the to-be-interfaced integrated circuit or equipment, knowing that device-specific implementation efforts are often unavoidable.
2. When facing multiple options, the following guideline may help:
 - SPI can offer attractive performance due to high maximum clock rates. Its support is also often coupled with relatively rigid timings, which may be attractive in hard real-time applications.
 - I2C may not be as fast as SPI, but generally slightly easier to implement.
 - RS235, RS422, RS485, and other UART-based communication interfaces are by essence asynchronous, hence certainly less attractive than the two first options whenever available.

More information on how to design a suitable carrier board for B-Board PRO is given in [PN201](#).