Real time communication protocols for B-Box RCP

PN202 | Posted on January 13, 2023 | Updated on May 7, 2025

Nicolas CHERIX
Head of Engineering
logo • linkedin icon

Table of Contents

Real time communication is essential in order to enable converter controllers to coordinate with other devices. B-Box RCP and other imperix controllers support several protocols for this purpose. However, selecting the right protocol and implementation among the available options may not be easy. Therefore, this article aims to:

- Present what real time communication protocols are supported by imperix controllers, considering the intended purpose as well as the readiness level of each approach/solution.
- Clearly explain the differences between the different protocols as well as use case scenarios.
- Allow users to select the best approach/solution for their specific needs.

While this article addresses mainly B-Box RCP, it also applies to other imperix controllers, provided that hardware capabilities are present.

Fully-supported real time communication protocols

Imperix B-Box RCP and B-Board PRO support the following real time communication options:

- **Ethernet** is merely the connection link hosting data exchanges using the **IP** protocol family. In practice, Ethernet rather refers to communication implemented using **TCP/IP** or **UDP/IP** protocols.
- **CAN** is a widespread field bus communication standard. While CAN designates the basic, almost hardware-level implementation, more sophisticated variants also exist, built on top of CAN (e.g. CANopen).
- **OPC UA** is an open communication standard for data exchanges between inter-operable industrial systems. It is mostly used for remote configuration, monitoring, and logging purposes (e.g.

SCADA). On imperix controllers, OPC UA is currently only supported over IP/Ethernet.
- **RealSync** is a proprietary imperix protocol built on top of Xilinx **Aurora**. It is used for ultra-low latency data exchanges between controllers. This protocol also serves the perfect synchronization (±2ns) between controllers. At the hardware level, RealSync uses SFP+ sockets and optical fibers.

The table below summarizes the intended use for the different real time communication protocols as a function of the considered scenario (see below).

	B-Box/B-Board	**3rd-party PLC**	**Host PC**	**Board-level ICs**
UDP/IP Ethernet	**Supported*** with Simulink/PLECS/C++	**Supported*** with Simulink/PLECS/C++	**Supported*** with Simulink/PLECS/C++	Not really relevant…
CAN RJ45 port	**Supported*** with Simulink/PLECS/C++	**Supported*** with Simulink/PLECS/C++	**Supported*** with Simulink/PLECS/C++	**Supported*** with Simulink/PLECS/C++
OPC-UA Ethernet	No support available for client mode yet	**Supported*** if PLC can act as a client	**Native**** with Cockpit **Supported*** with any other software	Not really relevant…
RealSync SFP+	**Native**** in I/O extension mode **Supported*** with Simulink/PLECS/C++	Not supported	Not supported	Not supported

Intended use for the supported real time communication protocols.

* **Supported** indicates that some basic configuration by the user is necessary.
* **Native** indicates that the feature is configuration-less and its use is transparent to the user.

UDP/IP

Support for **UDP/IP** over Ethernet is available using the dedicated blocks for Simulink and PLECS, as well as using the suitable C/C++ routines.

- The ETH output mailbox sends data (few Bytes) to any target and port, at a pre-defined rate or on-demand.
- The ETH input mailbox reads data (few Bytes) sent from any target at the beginning of the interrupt.

Transmission delays being non-deterministic over Ethernet, it is impossible to guarantee any latency. Therefore, we discourage using UDP/IP for data transfers involved within a closed control loop, unless that loop is of reasonably slow dynamics (typ. 10-50Hz).

As Ethernet (especially wired) benefits from significant raw bandwidth (typ. 1Gbps), transmission delays aren't much dependent on the amount of data transferred (not as much as for CAN for instance). This means that it is reasonable to expect millisecond-scale data transfer delays for up to dozens of Bytes/words (wired Ethernet).

If it can be accepted that the data transfer latency is higher than the refresh rate, there is – in general – nothing preventing UDP/IP to be used with refresh rates in the kHz range (or higher).

CAN

Support for CAN is available for Simulink and PLECS, as well as using the suitable C/C++ routines.

- The CAN output mailbox sends data (a few Bytes) with the chosen identifier, at a chosen rate or on-demand.
- The CAN input mailbox reads data (a few Bytes) for the chosen identifier at the beginning of the interrupt.

On imperix hardware, support is currently limited to the regular CAN standard. Support for extended IDs (CAN extended) or for the CANopen layer is under development.

Although CAN is in general non-deterministic due to the possible data conflicts on the CAN network, it can be almost deterministic when it can be guaranteed that only one master can emit data. Then, assuming suitable protections against data transmission failures, it is possible to use CAN within closed control loops.

CAN is often limited to 1Mbaud (that notably applies to imperix controllers). Therefore, the limited raw bandwidth makes it only suitable to the transfer of small amounts of data. Nevertheless, operating CAN up to few kHz is, in general, not a challenge.

OPC UA

OPC UA is used for the real-time configuration and monitoring of imperix controllers from the host PC. The interface is also available to any other clients, such as computer software or dedicated hardware (e.g. PLCs). Client-server roles are as follows:

- The B-Box / B-Board, acting as an OPC UA **server**, publishes a list of accessible variables (R/W).
- The software (or PLC), acting as an OPC UA **client**, subscribes to updates (at a selectable refresh rate) of those variables. The client can also write new values onto variables whenever permitted.

With imperix controllers, **all variables are always automatically published as accessible**. As such, it isn't even necessary to place any blocks (or function calls) inside the control software. Read/write access is instantly available through OPC-UA (unless configured otherwise).

Cockpit uses OPC UA for real-time communication with B-Box/B-Board and their configuration. However, the transfer of large files (notably .elf and .bit files as well as oscilloscope date) is handled on the side using regular TCP/IP traffic. For this reason, access to large data buffers (e.g. waveforms) is not available outside Cockpit.

OPC UA isn't particularly optimized for speed. As such, although it relies on Ethernet for transport, latency can be significant and total bandwidth isn't great. We recommend not using UPC UA for closed-loop control.

RealSync

RealSync is heavily optimized for reducing data transfer latency between imperix controllers. This is notably achieved thanks to data aggregation (respectively de-aggregation) for upstream transfers (respectively downstream) 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 occur in **master-slave** configurations (also called **I/O extension mode**). In this case, only one processing core is active, which receives all sensor data and emits all modulation parameters. These data transfers are configured automatically and are entirely transparent to the user.
- User-defined data exchanges occur in **master-master** configurations. This is also referred to as **operation in multi-master mode**. In this case, several processing cores are executing individual sets of control algorithms and exchanging some data periodically or punctually. These data transfers are configured by the user as a function of the application-level requirements.

Transmission delays are perfectly deterministic in both cases. Master-slave data exchanges are largely sub-microsecond, while master-master 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.

Finally, the table below summarizes the expected performance of each protocol for different criteria.

	Latency	Refresh rate	Payload size	Deterministic
UDP/IP Ethernet	Typ. 1-10 ms	Typ. 0.1-1 kHz	=32bits per block Typ. < 100 blocks	NO
CAN RJ45 socket	Typ < 1 ms	Typ. 0.1-1 kHz	<64bits per block Typ. < 10 blocks	NO (almost)
OPC UA Ethernet	Typ. > 10ms	Typ. 0.1-100Hz	?	NO
RealSync (native exchanges) SFP+	Typ. 0.1-1 us (sub-interrupt)	Typ. 0.3-300kHz	Typ. < 200 Bytes	YES
RealSync (user-defined exchanges) SFP+	Typ. 1-10 us (next interrupt)	Typ. 0.3-300kHz	=32bits per block Typ. < 100 blocks	YES

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

Recommended protocols by scenario

In-between imperix controllers

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

1. **In general, use RealSync!** Performance is hundreds to thousands of times superior to Ethernet or CAN.
2. In order to choose between native and user-defined data exchanges, read the related TN.

3. Keep in mind that it is always possible to implement SFP blocks plus those for another protocol on the same variable(2). The extra burden is often negligible.
4. In case there is any reason why RealSync couldn't be used, then backup choices would be Ethernet or CAN.
5. Since B-Boxes are generally connected to an Ethernet network anyway, UDP/IP is often the easiest choice.

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

This typically corresponds to situations where a B-Box/B-Board must receive instructions or set points from a PLC, or respectively return basic status information or measurements.

1. If the device is able to support OPC-UA (as a client, not 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 isn't supported, or its support is considered excessively tiresome, two cases should be distinguished:
    - If numerous variables (typically >20) must be read or sent and if time determinism isn't absolutely essential, then **Ethernet could be preferred** for its larger bandwidth.
    - Opposedly, if only a few variables must be exchanged and if arbitration can be avoided on the CAN network, then the lower (and more constant) latency of CAN could make it a better choice. In that case, it must also be checked that the support of the regular CAN standard is sufficient (CANopen not implemented).
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**
++ Instant access to all variables + No efforts on the control side + Access to auxiliary routines (start, stop, enable, etc.). – Non-constant latency	++ Generally well supported by 3rd party devices + Good for exchanging numerous variables – Non-constant latency	++ Relatively constant latency – Low total data bandwidth (1Mbaud) — CANopen not available
OPC UA documentation		

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

From/to a user-defined GUI on a computer

In this case, the main objective is to monitor/configure the controller from the computer, e.g. using 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 is at play when implementing communication with integrated circuits or similar devices, such as ADCs, sensors, etc. Often, if not always, a device-specific peripheral driver must be implemented

somehow. Furthermore, hardware timings are often critical.

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.