Multi-protocol controller for Industry 4.0

Andreas Schwope, Renesas Electronics Europe

With the R-IN Engine architecture described in this article, a device can process both network communications and complex applications simultaneously, with extremely low delays, low jitter and minimum power consumption. Thus a multi-protocol industrial automation product can be simply implemented using a flexible, low-cost R-IN single-device approach.

The automation industry is currently stepping into a new era often referred to as Industry 4.0. This revolutionary step in production facilities is based upon cyber physical systems. These combine available technologies like internet communication and automated operation controlled by sensor networks under real-time processing conditions. Industry 4.0 requires the provision of big internal and external databases in secured environments, and allows just-in-time production of highly customized products with a lot size of one. Having this in mind it’s easy to see that the new industry phase needs a much higher degree of automation than before. To facilitate cooperation inside a complex production plant, the different system components must be safely interconnected via a high-speed network guaranteeing deterministic and fast data exchange.

A frequent production requirement at this point is real-time processing with extremely short cycle times. In addition to the pure speed necessary, further time-critical parameters must be met under all circumstances. The most important of these are low jitter (small deviations in the system causing recurring delays) and isochronous behaviour (identical timebase for synchronized processes in all network nodes). In particular this extreme real-time capability of dedicated system components requires special hardware functions in certain points of the communication layer. This allows fast and smooth data exchange under all possible system conditions targeted by Industry 4.0.

Is communication always the same? Looking at the different standards used in automation the answer is quite simple: unfortunately not, we have different types of communication in the industrial network arena. In the past many communication protocols were developed, which are nearly all based on the same Ethernet standard IEEE 802.3. Due to different company interests, strategies and local distribution in the world, only the bottom two layers of the OSI model are roughly compatible between the different Industrial Ethernet (IE) standards. However the physical layer of the IEEE 802.3 Ethernet and the general frame format are in fact the same through the different standards. Some examples of established industrial network protocols are EtherCAT, Profinet, EtherNet/IP, CC-Link IE, Modbus TCP, Sercos III and Powerlink.

Applying a simplified model all of these communication technologies can be divided into two groups. In one group we find those protocols which are running with the "standard IEEE 802.3" hardware which you can also find in each and every PC of the world. In most implementations this communication hardware consists of an Ethernet MAC and an Ethernet switch with one internal and two external ports. The two external ports are customary in industrial networks to realize secured ring structures, while the single internal device port inserts and extracts data from the network traffic to run the local node functions. Only the upper communication layers of this first group include the special protocol functions in software. Members of this group are Profinet, Profinet RT (Real-Time), EtherNet/IP and Modbus TCP. To be even more precise most of these protocols are based upon the same TCP/IP and UDP/IP software stacks running on the standard IEEE 802.3 hardware. Others, like Profinet RT use a modified stack with lower processing latencies optimizing their speed and real-time capability.

The lower communication layers of the second IE protocol group require certain special, non-standard and sometimes unique functions. Among others these functions are used for real-time time management including network synchronization, and control the automatic extraction and insertion of Ethernet frame data. Since this runs under high speed real-time conditions, these protocols deviate from the Ethernet standard and can be implemented only in hardware in the form of a protocol controller. The port structure of such controllers used in protocols like Profinet IRT (Isochronous Real
Time), EtherCAT, Sercos III and CC-Link IE is generally the same as for group 1. For both groups figure 2 compares the protocol hardware/software layers of the Industrial Ethernet standards mentioned above.

From the perspective of a manufacturer, bearing all the various protocols in mind, it is of particular importance to be able to cover all these communication types with own network products. Aside from the functional view, commercial aspects are also important for success in the industrial automation market. Looking to product cost parameters like system simplicity, time to market, support, maintenance and many others, a product change from one protocol to the other should be possible with the identical product hardware. In this context one also speaks of the term multi-protocol device. Such devices can be easily adjusted to a certain protocol by just a simple replacement of the SW without any hardware modification. Multi-protocol support of an industrial automation product can follow various strategies. Companies have developed different solutions to run several industrial automation protocols in their products. All those solutions are generically described in this article, a mixture between them is possible of course. They have their own pros and cons as described later. Sometimes the focus of the ideal solution can change, especially when taking into account different product phases and volumes in their overall lifetime.

FPGAs are very flexible components which allow a hardware function change while using the same device. This is true in different situations when changing the function as such (specification change), when modifying a certain part of it (customization or optimization) or when applying a hardware fix (removing a bug). They are also a good choice when certain system components are to be combined in one device in order to save board space and reduce the PCB complexity. Sometimes boards have single gates or other low-complexity logic (also known as glue-logic) which can be completely integrated into the FPGA. FPGAs are also flexible when the complexity level of an implemented function has to be changed. Pin-compatible FPGAs allow the selection of the right number of usable gates. With a bigger FPGA a more complex func-

---

Figure 1. Industry 4.0 - The fourth industrial revolution
tion can be implemented when replacing the smaller FPGA. This allows the reduction of hardware cost in case of an optimization and the new function can be implemented with the right number of logic gates. Being very expensive, FPGAs are primarily used in the prototype phase of a product ahead of any cost optimization.

Application-specific devices are sometimes referred as ASSP (application-specific standard products). They normally have a high degree of integration with some or many dedicated functions required for a specific application. In contrast to an ASIC (application-specific IC), ASSPs can be used for a wider range of products in the dedicated market segment. Nevertheless many ASSPs often have an important commercial focus. They need to be as cheap as possible, but with every piece of flexibility or function enhancement their price increases. With the focus on a specific function and only the required logic, ASSPs do not generally provide that much flexibility. Based on such devices a traditional PCB design may result in a complex board integrating many different components as indicated in figure 3. This is directly followed by a huge amount of effort for the development, testing and certification, and last but not least for system support and maintenance during the product lifetime.

ASICs on the other hand provide with their specific function set the ideal solution for a certain device. By definition an ASIC is specified and developed by a particular company for its own products. ASICs are normally not sold to other companies. Seen from the development perspective it takes a long time and a huge amount of money to develop such devices today. Thus ASICs are the right choice only for very high volumes.

Another solution in the sense of multi-protocol can be seen as a combination of the two mentioned solutions. A module-based approach in an automation product foresees the exchange of rather small communication modules when another communication protocol is required. Using small modules from an external provider to modify the higher-level protocol of a system only partially follows the idea of an unmodified hardware: the more complex main system remains unchanged while only the less complex module is changed. This is a really good approach for prototype systems and low-volume products. But having regard to legacy products which cannot be changed, a module approach can easily allow the adaptation of a legacy system to a new communication protocol. Another advantage of the module approach is completely separating the application side from the communication hardware and stack software. As communication modules are relatively expensive, they are at a big commercial disadvantage. Mechanically they sometimes need more space and volume (area x height of module plus its connector) to be integrated into a system. This is often the module-killer criterion for a small product with a narrow or flat housing.

As described, all these approaches to multi-protocol have their own strength and are a good choice under certain conditions or in certain product phases (prototype, legacy product extension). But looking into the detail, all solutions have a common disadvantage: pricing, especially under high volume conditions!

How to escape from this conflict? Let us now think about an ideal solution which allows the simple product structure in figure 3 compared with the traditional approach. First our solution should be based on a single-chip device with more or less the characteristics of a small communication module provided by a single and well-defined hardware connection with different interface options. The application processing can be seen as an option in case
our device has to provide just the communication part of the system. In this case the device must support a flexible and high speed interface to the system CPU (host) with certain synchronization capabilities for event handling and data exchange. To support also a wide range of legacy systems, other lower speed interfaces like UART and SPI should be available too.

Turning to small network nodes requiring low- to mid-performance to compute an application, our ideal device should be a SoC (System on Chip) with its own CPU able to process both the communication protocol and the application. Performance often comes along with high power dissipation. Many products in the automation arena are sensitive with respect to power and temperature. Reasons include that they run in a high-temperature environment, use small housing without active cooling measures, and others. Thus our solution should include certain power-saving features in order to relax the typical performance/power barrier. Last but not least our ideal single-chip device provides a kind of communication-specific flexibility to support a multi-protocol capability for a wide range of the industrial Ethernet protocols. This should cover both groups as already described. All this should be available in an ASSP-type device to enjoy a price advantage from low to high volumes. OK, so far so good. Does anybody have this ideal device?

At this point, thinking about the requirements of Industry 4.0 networks and multi-protocol capabilities, Renesas developed the R-IN Engine hardware (R-IN: Renesas Industrial Network). The architecture of this function is perfectly suited for different Industrial Ethernet protocols and is more or less, as a distinct and independent block, already used in different Renesas product families.

The R-IN Engine architecture shown in figure 4 is first and foremost a sub-system with an ARM Cortex-M3 CPU and all on SoC-required components: memory blocks for instructions, data and other functions, interrupt controller and the corresponding multi-master bus system to different internal and external device re-sources, as well as the ability to debug. The multi-master bus allows concurrent data transfers between different areas without the requirement to interrupt the internal or external host CPU. When looking to the networking further components for Ethernet communication are included as well: a Gigabit Ethernet switch with three ports, one internal and two external (as already described). Further on the R-IN Engine includes an Ethernet MAC with associated DMA controller and required buffer memory explicitly used for Ethernet data transfers. In order to implement one protocol of the second group, the communication data path can be switched with respective multiplexers from the standard Ethernet path (IEEE 802.3 Switch and MAC) to the IE protocol controller of the second group.

Beyond this purely functional approach to support industrial networks, the R-IN Engine includes some accelerators replacing functions which nowadays are typically implemented solely in software. In terms of real-time requirements the processing of network functions is decisively speeded up with these accelerators at central points in the communication.

The HW-RTOS accelerator primarily supports the software execution with automated and prioritized task scheduling. Moreover this special hardware supports task synchronization via event flags, semaphores and mailboxes, as well as a general task and time management. Last but not least certain RTOS functions can directly be executed by a number of interrupt signals without any involvement of the R-IN CPU. Being closely connected to the CPU, HW-RTOS can sometimes highly accelerate both the processing of the stack software and the actual application. As the term accelerator clearly indicates, everything inside HW-RTOS is calculated by hardware in a very fast and deterministic way without the typical delays and jitter found in software solutions. From the software perspective the use of the HW-RTOS accelerator is based on a μItron library with a documented API and the related SW parts allowing a smooth project start. Therefore HW-RTOS is completely transparent for the user and does not require a detailed knowledge of the control structures of this accelerator.

When receiving or sending a frame byte the CheckSum accelerator automatically calculates on-the-fly the 4-byte checksum placed at the end of Ethernet frames. This calculation is solely done by the accelerator without loading the R-IN CPU. In the receive direction the correctness of the data can be checked in a single step by comparing the calculated FCS (frame check sequence) value with the frame FCS field in the received frame. By contrast
a typical software solution calculating the 32-bit CRC value consumes nearly 30% of the overall performance that is generally required for the Ethernet communication. Thus the CheckSum accelerator obtains a correspondingly large saving in CPU performance in high Ethernet traffic situations.

The organization of the frame data buffer for transmission or reception is normally byte-wise. Read-access to certain frame header information requires the collection of all necessary bytes in the frame buffer and their rearrangement into the right sequence. For the transmit direction the rearrangement must be done in the opposite direction into a compressed frame format. This data processing typically requires about 15% of the overall CPU performance for a pure software-based TCP/IP stack. The Header EnDec accelerator has the task of automatically rearranging the data between the compressed frame format and the CPU-oriented 32-bit aligned format.

With this accelerator the CPU has a well-suited, fast and direct read and write access to all frame header information without any latencies.

The Buffer Management accelerator automatically controls the buffer allocation and release functions in hardware for the Ethernet processing.

The basic structure of the R-IN Engine also provides the capabilities of a flexible host interface with required functions for process synchronization and fast and direct access to the communication data. For a single-core implementation this interface can be used for an external host running the system application. In a dual-core implementation it is a chip-internal interface between the R-IN Engine (communication part) and the main CPU of the device (application part). In this sense the R-IN host interface is of course not an accelerator, but without the need of a typical communication interface it allows direct and zero-latency access into the R-IN Engine and its resources.

Compared with other architectures, the advantages of the R-IN Engine with its different accelerators are reflected in higher CPU performance and increased stability while cutting the overall power consumption. The special hardware works much more efficiently and greatly relieves the CPU load. Thus R-IN architectures optionally run the network communication at significantly reduced power dissipation, or they deliver a significant margin to compute additional complex tasks in the application.

With the R-IN Engine architecture described, a device is able to process at the same time both network communications and complex applications, with extremely low delays and low jitter and minimum power consumption. Due to the network functions and underlying structures the R-IN Engine covers not only all the protocols of the first group using a standard IEEE 802.3 hardware, but also one protocol of the second group using a specific communication controller. Thus a multi-protocol industrial automation product can simply be implemented using the flexible and low cost R-IN single-device approach.

Designed for industrial networks the R-IN Engine is already successfully integrated into the R-IN32M3 and in the RZ-T1 families. While R-IN32M3 is a single-core solution with members for EtherCAT and CC-Link IE protocols, the RZ-T1 family is conceptually dual-core architecture. It basically has two separate CPUs for communication (ARM Cortex-M3 inside the R-IN Engine) and Application (ARM Cortex-R4). RZ/T1 comes with several derivatives for different product types. Other devices based on R-IN Engine are already under preparation or are being planned.

A further but not negligible advantage for R-IN software development is the quite simple protocol porting based upon the re-use of R-IN Engine hardware in different families. This is especially true for all protocols of the first group which run basically on the identical standard Ethernet hardware. When looking to the R-IN32M3-EC device example (EC: including EtherCAT Slave Controller, the basic structure directly correlates with the ideal solution as shown in figure 3. It also includes the 100 Mbit/s Ethernet PHYs and requires only a few external components to run the application and protocol in a single device. Thus R-IN32M3-EC is indeed a very good candidate for use in many small industrial Ethernet products. At the same time it is also perfectly suited for Industry 4.0.