Embedded real time in industrial automation systems

By Andreas Schwope, Renesas

This article introduces the R-IN32M3 SoC family for industrial Ethernet applications suited for the automation area. With the HW-RTOS accelerator inside the chip, it is the first family using this technology to accelerate embedded software processing.

Today distributed industrial automation systems require more and more real-time behaviour with extremely short cycle times in different network protocols. These real-time requirements can be seen in many cases in the system parts, the system communication part and the application part. While the communication part is responsible for the high speed data transfer between the nodes inside the distributed system, the application part processes the data and generates the application results. The input data is received from the system via its communication channel and from its local inputs like sensors. The results on the other side are communicated back to the system via the communication channel or drive the outputs (actuators) in the local node. Generally the term real time is often identified as high speed, but this is not the full truth. Speed requirements in distributed real-time automation systems with many network nodes are much more complex, and need dedicated hardware solutions to be real-time capable. High speed data transfer and data processing is just the basis.

High speed in the communication channel or in the application part can be directly translated into the time parameter low latency or low delay. A low value allows a system to react very fast upon external events. This feature is indeed very important to many applications, but on the other side this is just the basis of real-time behaviour in distributed industrial automation systems. Two further essential real-time characteristics have to come along with low latency/low delay. These are explained in the following.

The reaction time on certain events must be on the one hand as fast as possible – that's clear. On the other hand it must be guaranteed that the reaction always happens within a defined narrow time span independent from the current system state and independent from any other event influencing the system. For system-wide precise time management it is very important that all system delays are well known or can be exactly measured. It's generally true that the faster the processes to be controlled inside an industrial automation system, the shorter the delays which can be tolerated to run without any problems.

Unfortunately the delays are generally not constant and vary around a normal delay time. The variation is often based upon unpredictable events and has a certain dependency upon the current system state. This time characteristic is known as jitter and stands in contrast to the real-time requirement of accurately calculated delays. In an ideal scenario the jitter is zero, meaning the system timing in all nodes shows no variations. But that's a dream and is not possible. In a real scenario the jitter must be small enough not to stress the allowed system timing tolerances. These two behaviours with parameters delay and its jitter are opposed in figure 1.

This directly implies that without specific measures to control or reduce the jitter on the hardware and software levels as much as possible, real-time behaviour can hardly be achieved for a high speed application. This is true even if the different system parts are generally fast enough for that application. These kinds of measures need to be applied for both parts: system communication and application processing. At the end the interaction of all system parts has to fulfil the speed and jitter requirements, leading us directly to the next important real-time characteristic of distributed systems.

Industrial automation systems often consist of some or many system nodes which are more or less placed far away from each other. Thus they need to communicate with each other via a well-tailored communication channel. In most cases IEEE 802.3 Ethernet-based industrial protocols are used to support the speed requirements of real-time systems for the communication part. Well known industrial Ethernet protocols like Profinet IRT, EtherCAT, CC-Link IE and other protocols running at

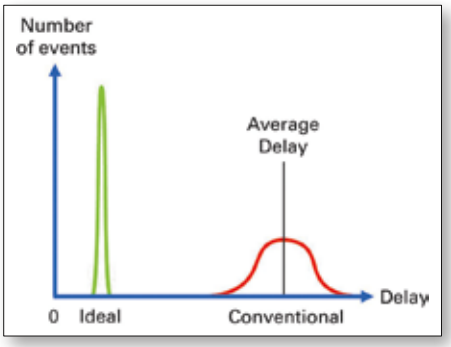


Figure 1. Delay jitter

100 Mbps or above can span several hundred meters, if so required in huge automated production systems. Looking again to the overall system time management for separated nodes in distributed systems, one can easily imagine the importance of time synchronization between the network nodes. The high speed processes running in the different nodes need to be strongly synchronized with each other to build the overall high speed deterministic system. They cannot correctly work together when the nodes are just estimated to act at the same time. In reality there is often a time deviation between the nodes which is unknown and which is probably not fixed and can change with the time (long term jitter). Such a scenario can eventually crash the high speed application. Simply stated, all network nodes have to run the identical timebase and have to act at the same clock cycle ideally without any fluctuations. This ideal synchronized task scenario inside a distributed system is also known as isochronous behaviour and is shown as one of the real-time requirements in figure 2.

Industrial Ethernet protocols named already require a dedicated controller which takes care of the high speed low delay and low jitter data transport. On the other hand they implement certain communication functions of the lower OSI levels of the related protocol. This includes certain functions to measure the delay and keep the communication jitter as small as possible. They also take care for the exact time and clock synchronization between the connected and distributed network nodes to form the isochronous system. Often this synchronization function is based upon IEEE 1588, Precision Time Protocol (PTP). This also requires dedicated hardware functions to exactly measure the communication delays between the connected network nodes. Such a low-jitter and isochronous time management definitely cannot be achieved by a pure software solution.

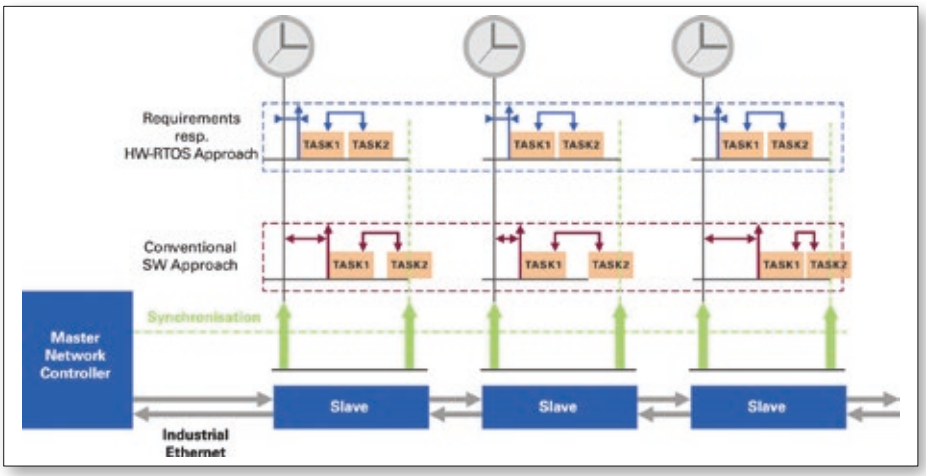


Figure 2. Real-time requirements and conventional solution

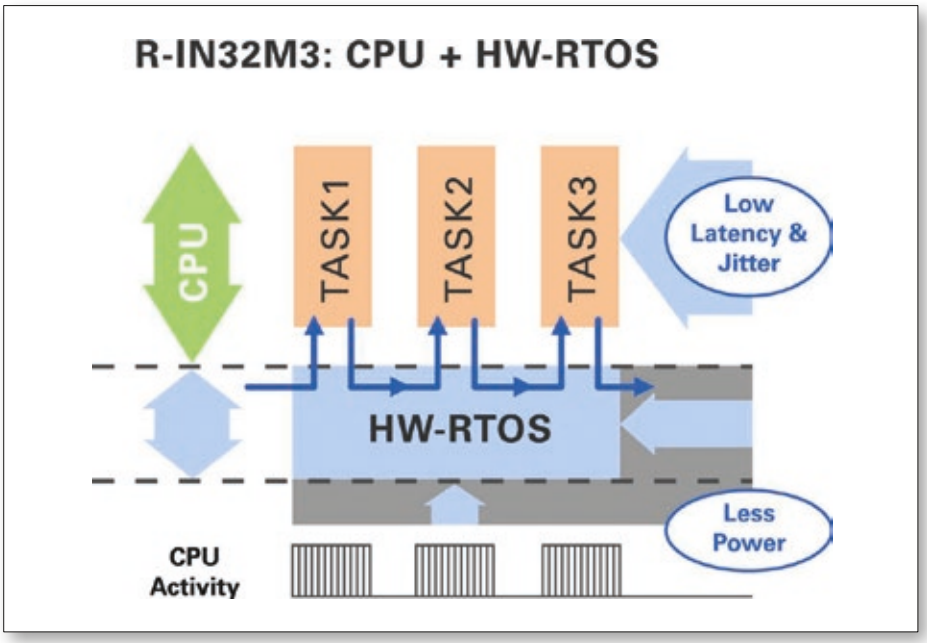


Figure 3. Hardware-RTOS acceleration

With the integration of a specific industrial Ethernet protocol controller all real-time requirements are more or less automatically fulfilled for the system communication. But this is just one part of the solution as introduced already. The same requirements must also be taken into account for the application side in each and every network node of the whole system. So how can you achieve this real-time application performance?

Typically the applications run on embedded CPU systems and control sensors and actors in dedicated programs with special algorithms. Sensors and actors are connected with the CPU by serial or parallel interfaces. Some of the interfaces are generic while others implement a very special feature set with a dedicated controller tailored for a certain application area only. Increasingly, the CPU core and all required system components like memory, DMA, interrupt controller and several interfaces with specific controllers are integrated in one single semiconductor device. Such System on Chip (SoC) devices for industrial automation or industrial Ethernet applications are equipped with one or more network controllers for specific Ethernet protocols. When looking at the application processing, we basically have to distinguish between two options. The communication SoC is connected to a host system which runs the application. This divides the node into two separate parts introducing additional delay and jitter with the interface in between. Or the communication SoC has to run both the communication and the application. Both operate very close together, minimizing the negative timing influence on delay and jitter.

From this perspective the latter option is clearly the best choice for real-time systems. When applying the described requirements from above to a generic software structure with different processes and tasks, the software has to be executed fast enough. This represents the high speed characteristic of real

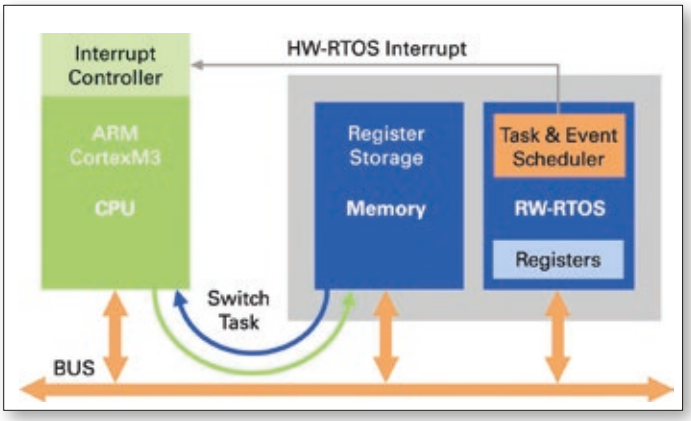
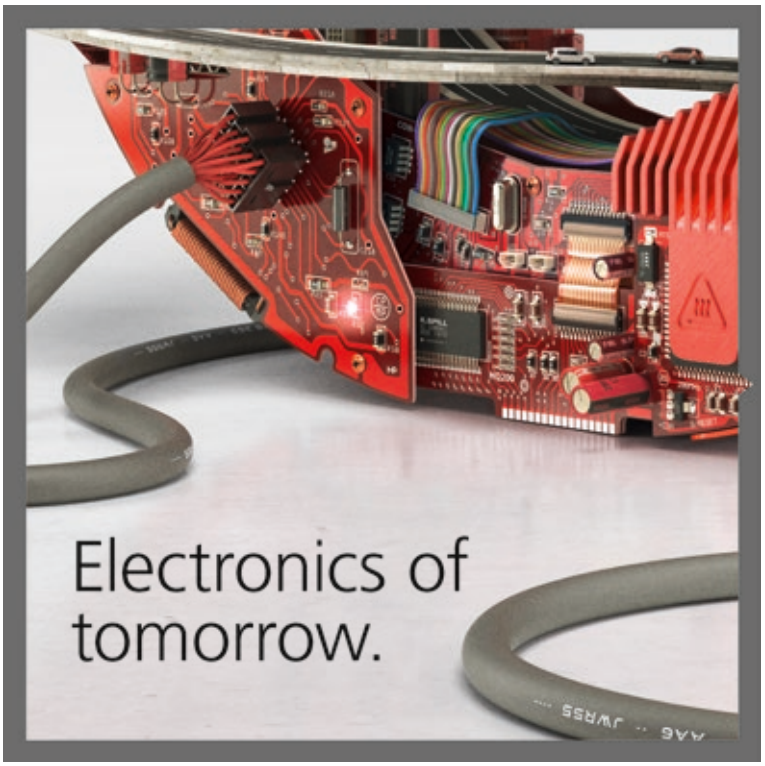


Figure 4. Hardware-RTOS connection to the CPU

time. More important, and crucial for isochronous system behaviour, is deterministic switching of the CPU tasks with short delays and low jitter. Comparably with the communication level, these two parameters are also the basis for isochronous application operation between all nodes in the industrial automation system. The second row in figure 2 compares the conventional node implementation approach with the real-time requirements.



One direct requirement is the use of a real-time operating system (RTOS). Typically CPU systems use more powerful CPU cores running at a higher clock speed. In order to get enough real-time margin for each and every system condition, the CPU performance has to exceed by far the performance required to satisfy the application as such. Often communication and application parts are intentionally separated and run on different sub-systems with independent CPUs. Aside from the negative timing impact, such a solution can dramatically increase the board space. This comes with more complex system design and non-straightforward efforts for CE and EMV certification with a lot of potential problems. In addition to increased cost and time-to-market, all these solutions have higher current consumption. The result of this would normally be higher chip temperature, generating problems in small housings with passive cooling measures.

With respect to their runtime requirements, many algorithms and control structures in the application software can be implemented more or less in a deterministic way (speed, delay, latency). On the other hand the interaction between the different software tasks often depends upon the internal states and external events. Based upon this, the system timing is hardly and sometimes even not predictable. Whenever an event forces the application software to switch from the active software task to another task, the CPU needs to identify the next task, save the current and load the new task context (CPU registers). Even in an optimised RTOS software environment such kinds of effects occur and cannot be neglected.

The complexity of this context change process directly introduces a certain amount of CPU runtime delay before the new task can start or continue its work. Typically this effect can be compensated when using a higher CPU clock speed or having a more powerful CPU (see the disadvantage described above). Seen on a higher task level (e.g. non-interrupted CPU run time between starting and ending a certain computation) an undefined number of context switches introduce an average runtime delay which is naturally combined with a certain amount of jitter. This makes it very clear that software context switching is often the weak point in well balanced real-time systems as. But how to optimise task switching, especially with the background that the real-time operating system (RTOS) is already highly optimised in that sense?

The application software with its algorithms and control structures is already in a good shape and close to an optimal implementation. The solution to the negative effects of software context switching can only be a type of a hardware-based task scheduler. Such a scheduler would take over certain RTOS functions performing its work in hardware much faster with much less delay and thus with less jitter. As a matter of form such a kind of hardware function can be described as an RTOS accelerator. The advantages in the time domain of such an accelerator are directly visible: reduction of processing delay and jitter in the application and protocol stack software while the isochronous capability of the distributed automation system is optimised. But such an implementation improves other parameters as well. The resulting CPU performance inside an SoC using such a kind of RTOS accelerator appears to be higher compared to the same system without this accelerator. From the application perspective this advantage can generically be used in two opposite ways.

Firstly, reduce power consumption. The increased SoC performance processes the application tasks much faster. This results in more idle times and saves power when the CPU is stepping into a sleep mode. Alternatively a reduced system clock can be applied to process the same application with fewer idle times for even more power savings.

Secondly, increase software functions. The increased SoC performance can be used to compute even more complex software. Especially the combination of protocol stack combined with application software easily allows an industrial automation system running on a single chip device. With this it is possible to build very small, low-power and fully functioning network nodes for industrial automation.

Renesas has developed the ARM device family R-IN32M3 (Renesas Industrial Network) with a completely new RTOS accelerator. This HW-RTOS (Hardware-RTOS) supports the embedded ARM CPU with the RTOS task scheduling and other typical real-time OS functions which are normally implemented on the software level. With this the

CPU processing speed can be dramatically improved while reducing the typical software jitter. Aside from pure task scheduling with priority options, HW-RTOS can also perform task synchronization with event flags, semaphores and mail boxes, and support general task and time management. Further, HW-RTOS allows to directly execute certain functions triggered by an interrupt without any CPU involvement. Typically this implementation increases the task switching speed by a factor of five compared to similar architectures with a pure software RTOS approach. As the hardware implementation works more efficiently than software, the Renesas SoC architectures with HW-RTOS come with reduced power consumption and improved stability (figure 3).

In principle HW-RTOS is an independent functional block connected via an internal system bus to the ARM Cortex-M3 CPU. This avoids making any kind of adaptation to the CPU itself or the software which is running on it. Only a special HW-RTOS interrupt signal is connected to the CPU interrupt controller to allow direct and low latency interaction between the HW-RTOS and the CPU (figure 5). With an interrupt the HW-RTOS informs the CPU to directly exchange the current task context with the new context without the typical long delays found in software scheduling. This acceleration reduces the CPU load and allows saving performance for the application software. In addition to many other parts like drivers, middleware and sample code, the R-IN32M3 software package includes a HW-RTOS µItron library. This library allows making completely transparent use of the accelerator without any deep know-how about its details and how to run it in software. ■