EtherCAT offers improved solutions for industrial PLC applications

By Peter Schuller, MicroSys

The EtherCAT fieldbus communication infrastructure has been integrated into the MicroSys families of power-architecture-based system-on-modules and single board computers. This article describes the EtherCAT functionality and its benefits for PLC-based industrial applications.

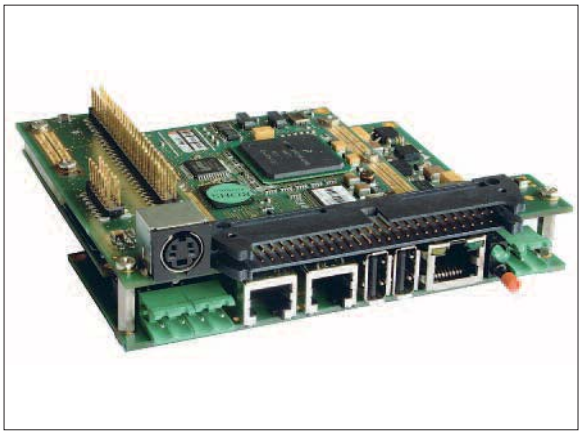

Figure 1: Power architecture MPC5200 CPU module with carrier board and typical physical connections

For deeply embedded solutions in transportation, automotive, aerospace, defence, industrial or medical applications, system platforms require more easily implemented, open and flexible I/O-infrastructures to interconnect to control subsystems. Widely accepted or standardized fieldbuses are one means to achieve this goal. MicroSys has decided to integrate into their power-architecture-based system-on-modules (SOM) and single board computer (SBC) families the Ethernet-based EtherCAT fieldbus communication infrastructure. It has been introduced by Beckhoff, offers deterministic data packet exchange over a physical Ethernet interconnect, and is supported today by more than 800 companies worldwide. The community of adopters is growing fast.

This solution offers an easy means to integrate and access a huge base of I/O and control requirements, from rapid prototyping to production-worthy systems. It was developed in cooperation with the Competence Center for Real Time Networks (CCRN), represented by Professors Seck, Fischer and Sommer of the University of Applied Sciences in Munich.

The MicroSys product portfolio is mainly focused on highly integrated system-on-modules and single board computers to address the low-power and high-performance requirements of deeply embedded demanding industrial applications. So, PowerPC- and ARM-based platforms supporting operating systems like Linux, Microware OS-9, QNX, Vxworks or others are applied in the majority of customer projects. It was therefore decided to have a first implementation of the EtherCAT environment on a power-architecture MPC8349 CPU-based platform.

Embedded system solutions are normally very demanding in terms of performance, environmental requirements and assumed long product life-cycles. In addition these projects may not have the high run-rate of typical consumer-electronic products. So it is hard to allocate the development cost to the board cost. The combination of a system-on-module with the appropriate carrier board for the I/O-functions allows using the complex technology for demanding applications at a reasonable cost. Even at a run-rate of 50 systems/year these innovative solutions can be accomplished. The existing base design of a SOM needs to be developed once and can be used in a variety of different applications. Customer- or market-specific adaptations can be made with the carrier card. System-on-module solutions offer: fast access to proven technologies, cost-effective and ready-to-use development kits, short design cycles for hardware platforms with specific peripherals, parallel start of the software development, cost savings in the development process, and fast market access. Furthermore they enable simple technology upgrades during the product life cycle by exchanging the module with a compatible, newer version with more functionality. The platform concept offers a lot of flexibility in terms of different performance levels and configurations. Additionally they provide support for various OSs with existing board support packages (BSPs). The miriac MPX (MicroSys PPC EXtendable Module) modules are a scalable family of SOMs, mainly based on power architecture, but open as well for ARM or Intel x86 compatible designs. With the corresponding carrier boards the physical I/O connections are available. The modules can be easily adapted for use in different bus-systems such as Compact PCI, VME, 19“ racks or OEM-specific form factors.

System-on-modules (CPU boards), carrier boards to hold the peripheral interfaces and application-specific functional components can be stacked easily by this architecture. This enables the entire functionality provided by this innovative connector technology: low profile for small system dimensions, stacking of modules for flexible and easy functional additions, more than 200 contacts per connector with best quality of transmission, and the transmission of signals as well as supply voltage for currents up to some amperes. It is compliant with shock

and vibration according to DIN EN 60068, has a reliable zero force connection and simple assembly, and includes tight sealing between connector and board to allow easy coating (often needed in the avionic, railway, chemical industry or in other harsh environments).

Having I/O-interfaces settled easily as piggy-pack modules on top of the SBC8349 board is one way to get peripherals attached. Looking over the fence to the process control world, standardization accessing process peripherals like I/O terminals or servo drives is state-of-the-art. The application programmer of a programmable logic controller (PLC) does not want to deal with device driver implementation or hardware development. He wants to start right off the shelf with his application coding. Therefore the software access to the peripherals is standardized, too. The drawback is that using standardized process control hardware, you normally also need to switch to PLC hardware and a style of programming that does not use the preferred real-time operating system and C programming environment which gives maximum flexibility in choosing real-time algorithms. So if volume quantities do not force to build an own piggy pack module, or the application environment requires anyway to gain control over standardized PLC peripherals, it is worth looking to see how such access can be achieved with little effort.

Several peripheral standards exist in the field of process control. Which one should be supported? Most standards are bound primarily to a specific underlying fieldbus system which mostly limits at least the theoretic maximum access speed to the peripherals. Accessing I/O registers on piggy pack modules in sub-microseconds sets your expectations coming from the embedded world. Therefore bus systems like CAN or Profibus are immediately out of focus. But they are very popular and lots of specialized hardware peripherals are available off the shelf, and at this stage of discussion you do not want to lose the chance to use them if necessary.

The idea is to use well-known and widespread cheap and fast bus technology: Ethernet. Ethernet comes with 100 MBit and 1GBit speed, but unfortunately is based on CSMA/CD access methods which makes it basically unsuitable for real-time networking. CSMA/CD access does not

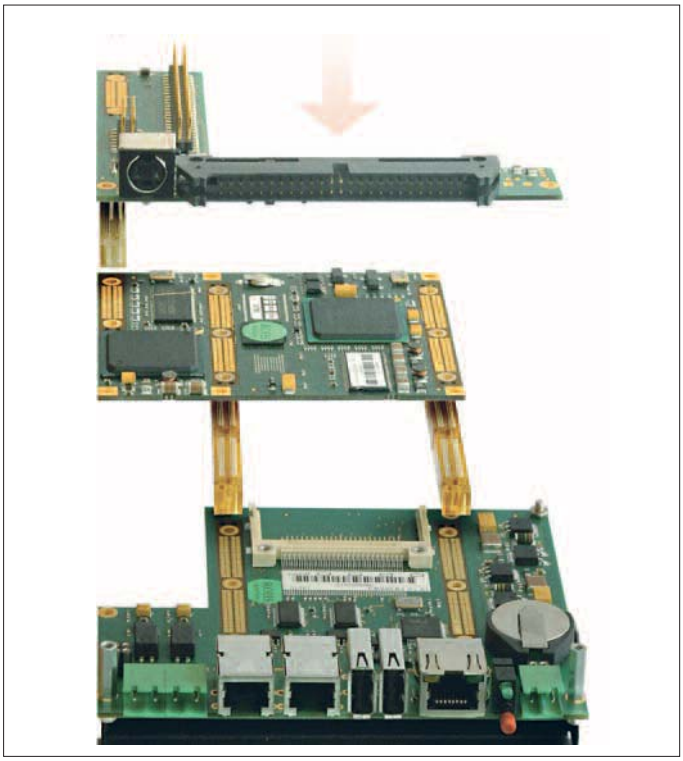

Figure 2: Example of a miriac SOM assembly

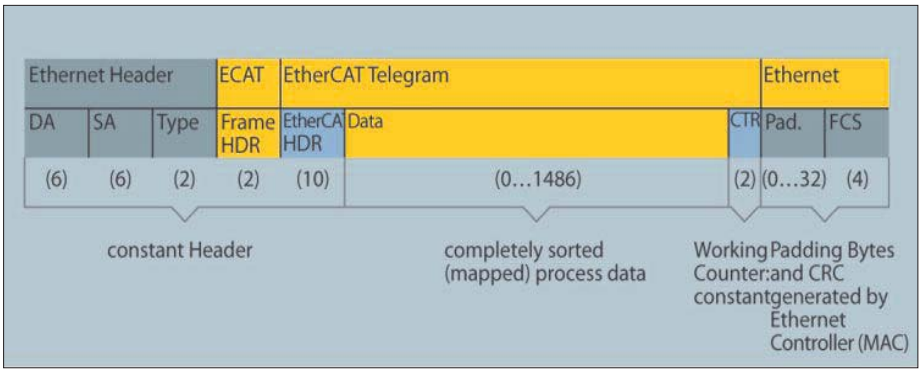

Figure 3: EtherCAT protocol packet

guarantee that a message gets reproducible in a defined time period from sender to receiver if there are several simultaneously active senders and receivers in one bus segment. But in the process control world the PLC is the logical master and the peripherals are the slaves. Therefore the master rules over who may send at a specific time and who should receive on the bus system.

Turning classical Ethernet to some kind of real-time Ethernet starts at this point, to get rid of the consequences of CSMA/CD. There is always only one sender at a time who wants to send a message and then there are no collisions to be detected on the classical Ethernet. Due to this master/slave rule you can start using standard Ethernet hardware. The second approach to eliminate collisions is to split the domains using switches. Most of the Ethernet installations today do in fact not use a bus structure anymore, but have a star or tree topology. Here the devil is in the details – meaning the delay in the switches. The impact of switch delay has moved into focus and there is quite some research to minimize the effect.

Several real-time Ethernet solutions are based on these ideas. Most popular, in respect of the number of available products from different companies on the market, are two popular real-time standards in Germany: Profinet and EtherCAT. Both are true real-time Ethernet protocol standards, Profinet mostly promoted by Siemens and EtherCAT by Beckhoff Automation. EtherCAT has one big advantage over Profinet: the master only needs a classical Ethernet chip without any extensions. The tricky real-time enhancements are all done by the EtherCAT hardware of the slaves, the process peripherals. By the way: EtherCAT is much faster than the Profinet solution, and you can easily buy an EtherCAT to ProfNET gateway if access to Profinet devices is needed. The advantage is you only need to learn once how to access the peripherals by using EtherCAT access; the gateway to Profinet does the translation from EtherCAT and vice versa. There are also gateways for Profibus and CAN/CANOpen and several other slower bus systems available. There are setups where Profinet could have a slight advantage, but only if you want to transmit big frames (e.g. 1 kbyte) for each slave, but this not a typical control setup.

The standard feature of EtherCAT to use built-in clocks (distributed clocks) even allows some jitter in the operation of the real-time system of the master, without affecting the timely behavior of the whole system. Using EtherCAT controlling some simple digital and analog I/O terminals as peripherals (figure 1), the minimum cycle time seen by the software writing/reading I/O bits on the MicroSys board comes down to 60 microseconds. Within 60 microseconds an Ethernet packet consisting of typically 60 bytes is send from the master, the MicroSys board, and shifted through the slaves where I/O-bits-bytes-words are read and/or written and then shifted back to the master again. How is that done? The EtherCAT standard is very flexible. However, between master and the slaves it is advisable not to use a classical store-and-forward switch or router. Taking the easiest case, the master is directly connected to the first slave. The first slave normally has at least two EtherCAT ports: one to the master and one the next slave in the row. Coming back to the simple example, the MicroSys board as master is only connected to one EtherCAT bus coupler EK1100 from Beckhoff. Connected to the bus coupler are the three digital I/O terminals EL1004,EL2004, and EL1018 (figure 1). The full-duplex Ethernet bus cycle starts at the master.

The master broadcasts a standard Ethernet packet tagged in the header as an EtherCAT protocol packet (figure 3). Looking at the bit timing on the bus the Ethernet chip of the master starts to shift out the bits on the wire and after a short time the bus coupler starts receiving and processing the bits. Therefore the slave does not wait until the whole packet is received, instead the processing (i.e. reading and writing data) of the packet is done on the fly. Each of the bus terminals connected to the bus coupler adds a delay to the packet time of about 0.3 microseconds, the bus coupler itself adds 1.2 microseconds delay time to the packet time length. Taking our Ethernet packet on the wire with 60 bytes packet plus 7 bytes preamble and start frame delimiter (SFD) byte and 100 Mbit transmission speed, the time duration of the packet is roundabout 5.45 microseconds. Leaving the delay (5 nanoseconds/meter) of the wire itself out of focus our packet is delayed by the peripherals: 1. 2 plus 3*0.3 microseconds which is 2.1 microseconds in total.

2.1 microseconds after leaving the master Ethernet chip the first bit of the packet is back at the incoming pin of the master Ethernet chip. Concerning only the roundtrip, without processing through the application and protocol software running on the master, it takes 5.45 microseconds plus 2.1 microseconds which

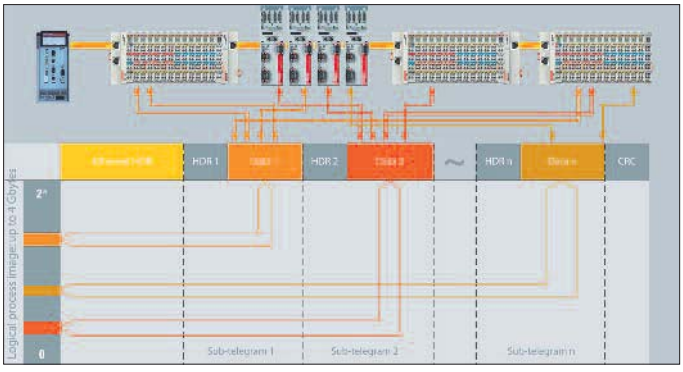

Figure 4: Mapping EtherCAT peripherals to a software process image

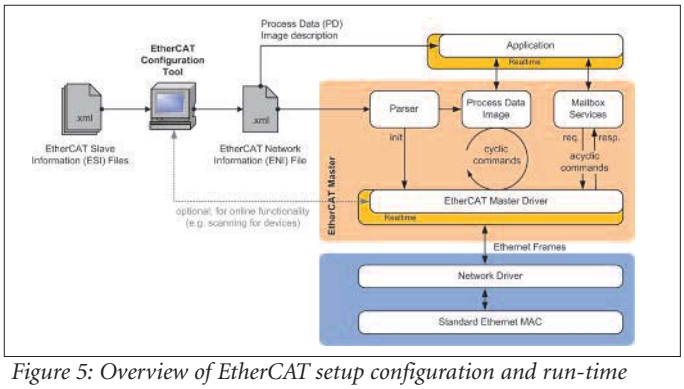

Figure 5: Overview of EtherCAT setup configuration and run-time environment

makes 7.56 microseconds sending one packet from the master, getting it processed on the fly by the peripherals and having it back with the I/O answers of the peripherals. The rest up to the measured 60 microseconds cycle time is consumed by hardware access to the Ethernet controller, interrupt reaction time, the EtherCAT protocol driver and by the trivial application just doing the loop as fast as possible, sending one packet then receiving it.

Classical Ethernet communication would be slower because the receiver Ethernet chip would not pass the packet to the slave hardware until the packet is received completely. There the roundtrip time would be n times 2*5.45 microseconds where n is the number of connected slaves. Adding more slaves to the advantage of the EtherCAT bus system minimum time delay becomes more important. How handsome is the interface to the software programmer? Does he have to learn the EtherCAT protocol before he can use the peripherals? No, with the MicroSys board and EtherCAT development environment it is a piece of cake.

The application programmer model of the EtherCAT access to the standard EtherCAT peripherals is thinking of a big shared memory, the so-named process image (figure 4). Each bit in the process image belongs to an I/O peripheral and can be read or written easily by the application program. During startup phase of the application a configuration file is loaded and the EtherCAT process image is configured. There address mapping takes place by which bit, byte word of the I/O peripheral is mapped to a fixed address in the process image (figure 4). After that startup phase the EtherCAT bus goes into the operational state and the once configured process image is cyclically shifted from the master through the slaves back to the master. The EtherCAT run-time environment, EtherCAT-Mastertask (EMASTER) available with the MicroSys board performs the mentioned task and runs on standard Linux for soft-real time applications and RadiSys Microware OS-9 real-time operation system for true real-time tasks.

EMASTER is built upon the ported and PPC-optimized EtherCAT Master sample code of Beckhoff and offers the mentioned process image to the application programmer by shared memory access. EMASTER is a standalone task which hides the whole EtherCAT startup and cyclic operating phase of the protocol from the application program. The application itself can be spread over one or more standalone multithreaded tasks and can easily get synchronized access to the process image via exported shared memory by a C-application library API.

At startup time, EMASTER requires an XML configuration file containing the initialization of the I/O peripherals, the timing and the mapping to the process image. This XML file is interactively generated by using a PC based EtherCAT configuration tool e.g. Beckhoff TwinCAT I/O. Figure 5 shows the interdependencies between the configuration tool and the EtherCAT EMASTER run-time environment.

The configuration tool is able to automatically scan the EtherCAT bus for slaves, then the configurator automatically builds up the mapping of the I/Os. At that stage you are free to name the I/O bits and bytes (now called I/O variables) including the determination of access width by readable symbolic names. These names are written to the XML file, and the application can then access the I/O variables during run-time in the process image by names instead of magic mapping addresses. EMASTER offers an interface where you can query the symbolic names of your variables during run-time. ■