White Paper

Improving Determinism of Real-Time Applications Over Ethernet

The number of Industrial Ethernet devices shipped is expected to double in next two years. Yet Ethernet was not designed for real-time applications.

Implementing Quality of Service (QoS) over Ethernet – that is, guaranteeing that Ethernet will truly work in real-time – involves improving the predictability of the network.

Schneider Electric and Innovasic Semiconductor compared network predictability using a 133-MHz RISC CPU and a 66 MHz fido from Innovasic. Even while running at half the clock rate, the fido dramatically outperformed the RISC, with a much more straightforward software implementation.

AUTHORS:

Schneider Electric
Dinnov Research Department

Pierre Colle,
pierre.colle@schneider-electric.com

Jean-Pierre Desbenoit,
jean-pierre.desbenoit@schneider-electric.com

Jérôme Hamel
jerome.hamel@fr.schneider-electric.com

Innovasic Semiconductor

Bill Browne,
Innovasic Semiconductor
bbrowne@innovasic.com

Volker Goller,
Innovasic Semiconductor
volker.goller@innovasic.de

Abstract

The number of Industrial Ethernet devices shipped is expected to double in next two years. To date, Modbus TCP and Ethernet/IP represent more than 50% of the number of Industrial Ethernet units shipped.

With Modbus/TCP, Schneider Electric has been an early promoter of solutions based on standard Ethernet components. Advantages of this strategy are numerous: continuous chip cost reduction, continuous performance increase, multiple vendor sources, etc.

Still, in order to guarantee determinism of industrial applications, an Ethernet network used to convey real-time data had to be isolated from other networks. One reason was to avoid non real-time data exchanges from impacting transfer time and processing time of critical requests such as those of an I/O scanner. But nowadays many Industrial devices embed a Web Server for diagnostics and configuration and SCADA needs to access production information through web services. So a solution has to be put in place to be able to have different types of flow transferred and processed simultaneously by the network and attached devices without impacting the performance of real time exchanges. Ideally, there should be no extra hardware cost.

One solution to achieve this, still based on standard technology, is the use of Quality of Service (QoS), which is the ability of a network element (application, switch, etc) to have some level of assurance that its traffic and service requirements can be satisfied. QoS provides improved service to certain traffic. This is done by either raising the priority of a flow or limiting the priority of another flow. QoS information can be found at different levels:

- Level 2: Ethernet frames can be tagged with a priority level as defined in IEEE 802.1Q. A VLAN identifier must also to be set within that same tag.
- Level 3: DS field, which supersedes the TOS field in the IP header, can be used to set priority.

In order to enable QoS, it requires the cooperation of all network layers, as well as every network element:

- Application task architecture: must handle/generate priority flows
- IP stack: must be able to set QoS field and possibly be re-entrant
- Switch: must be able to manage a queuing policy from QoS field.

A QoS guarantee is only as good as the weakest link in the "chain" between the sender and the receiver. How to achieve this on the network is beyond the scope of this document since it makes use of well known standard technologies. This paper will concentrate on the implementation at the device level.

After a short description of main concepts (Determinism and Quality of Service), two architectures will be presented and compared:

- Standard architecture making use of a RISC CPU at 133 MHz with an integrated Ethernet switch

- New fido silicon architecture from Innovasic Semiconductor making use of a real-time CISC CPU at 66 MHz

One outcome of this comparison is that fido achieved better performance in terms of jitter, with a much more straightforward software implementation thanks to its innovative chip architecture.

Quality of Service & Determinism

Quality of Service

Quality of Service (QoS) is the ability of a network element (application, switch, etc) to have some level of assurance that its traffic and service requirements can be satisfied.

The first thing to do is to be able to differentiate the different flows of information in order to be able to apply different levels of service. This is achieved by "tagging" flows with a priority level. Two standard techniques exist: Using the DS Field and adding a VLAN & QoS tag.

DS Field

A replacement header field, called the DS field, is defined in RFC 2474, which supersedes the existing definitions of the IPv4 TOS octet [RFC791] and the IPv6 Traffic Class octet.

The DS field structure is presented below:

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|         DSCP          |  CU   |
+---+---+---+---+---+---+---+---+
```

DSCP: differentiated services codepoint
CU: currently unused

DSCP (Differentiated Services Code Point) marking uses 6 bits of the 8 bit ToS field in the IP Header to provide up to 64 classes (or code points) for traffic. DSCP is a layer 3 marking method. Devices that do not support DSCP will simply ignore the tags, or at worst, they will reset the tag value to 0.

VLAN & QoS Tag

This method uses a 4 bytes tag inserted after the ‘source address’ field of Ethernet frame:

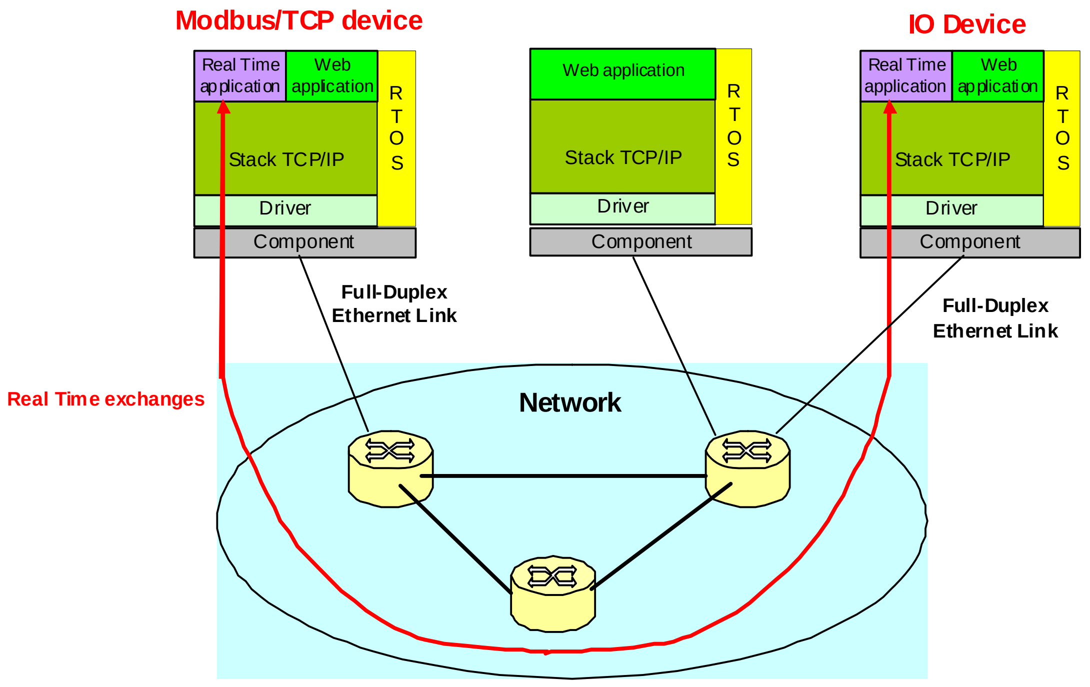
Preamble	Start Frame Delimiter	Destination Address	Source Address	802.1Q Tag Type	Tag Control Information	Length Type	MAC Client Data	Padding	FCS
7 bytes	1 byte	6 bytes	6 bytes	2 bytes	2 bytes	2 bytes	0-n bytes	0-p bytes	4 bytes

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Tag Control Information															
User_priority			CFI	12 bit VID (VLAN ID)											

The 802.1Q tag type is set to 0x8100, denoting the new frame format. A user priority 3-bit field can be used to store a priority level for the frame. Use of this field is defined in IEEE 802.1Q. Highest value (7) corresponds to highest priority.

Both techniques are standard, and by using them, an industrial device is able to set and identify traffic types. They can be used independently or simultaneously. Use of one technique or the other is out of scope of this document as it depends on network policy, switch capabilities, etc.

Determinism



Determinism is the capability to calculate, simulate and guarantee the worst case time to exchange real time information from one device application to another remote device application.

When using Ethernet TCP-UDP/IP network and protocols to exchange real input and output values, such as in the Schneider Electric Modbus/TCP solution, the jitter on applied outputs must be limited to the smallest possible value. This means the determinism of the real time exchanges must be guaranteed, no matter what other network traffic is occurring, such as web exchanges (HTTP, FTP) or non real time exchanges (configuration, management).

Performance tests for network components (switches) have shown that it is possible to provide real-time communication on the network domain using of Quality of Service.

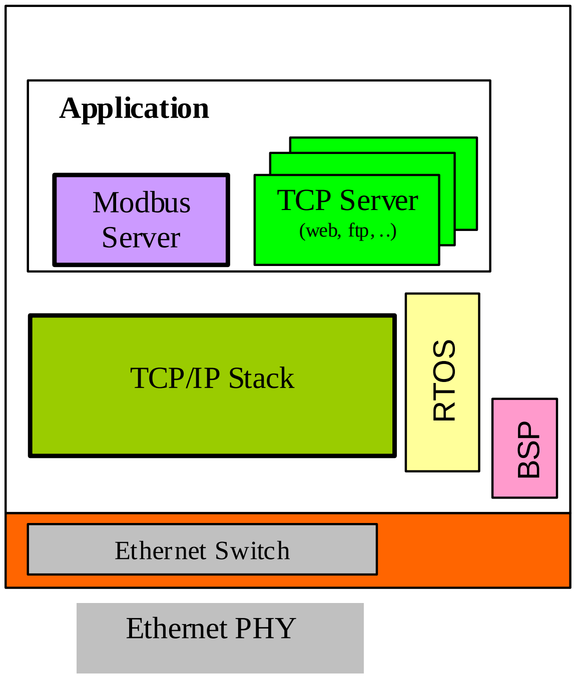
However, the critical issue here is the communication software embedded into the IO device. The processing time of an IP frame by a software communication stack may be quite long, ranging from a few hundred of microseconds to a few tens of milliseconds, depending on the length of the frame and of the device processing resources. Moreover, most of the communication stacks are uninterruptible, meaning that when processing a low priority frame, a higher priority frame cannot interrupt this process and will wait till the end of it, leading to potentially very high jitter and unpredictable processing times.

Determinism, the standard approach

The implementation using the standard approach is:

- RTOS. Using a popular commercial RTOS, processing of high priority (Modbus server) and low priority (TCP server) requests are separated in different tasks having different priorities. The Modbus server task has a higher priority than other tasks handling low priority flow(s).

- TCP/IP Stack. Ideally, the TCP/IP stack should be re-entrant to allow processing of a low priority frame to be interrupted by a high priority frame. However, no such commercially available TCP/IP stack exists. A compromise approach is to select a stack making the most limited use of critical sections. For example, data transmission



is done in the calling task context by a cascade of function calls. Then, if the sending task has a very high task priority, it will not be interrupted by any other task activity, except by interrupts. For the incoming traffic, we rely on a previous sort of incoming flows by the Ethernet switch. Frames are then processed in order of priority.

- Ethernet switch. For this approach, a CPU integrating an Ethernet switch was selected so that sorting of the different flows was done by the hardware directly. Flows are then stored in different input queues depending on their priority.

When mixing periodic high priority requests with bursts of low priority requests, we obtained the following results on a RISC CPU at 133MHz integrating an Ethernet switch. For a typical performance test where the background traffic is characterized by a burst of two 1400-byte requests and 1400-byte responses, the Round Trip Time (in micro seconds) of the high priority requests/response is given in the table below.

	Without background traffic	With background traffic
RTT Min	970	1 006
RTT Median (at 51% of probability)	1 010	1 170
RTT High at 95% of probability.	1 040	1 360
RTT Max	1 110	1 769
RTT Jitter	***140***	***763***

Using this architecture, a predictable upper limit to response time and reasonable determinism was achieved. Still, jitter up to 700 µs is observed. Origins of this jitter are:

- Critical sections of the TCP/IP stack that cannot be interrupted. Some processing of a low priority frame must complete before being able to process a high priority frame.

- Task context switching time

Also, even if we achieved an upper limit for the jitter, an error in one low priority task (ex: memory corruption) can still have side effects on processing of high priority requests. In other words, the full system must be extensively tested to guarantee reliability and determinism.

fido

fido (Flexible Input, Deterministic Output) is a new family of 32-bit microcontrollers, with a unique new architecture that is optimized for embedded control and communications applications. This is the first completely new embedded microcontroller architecture in many years.

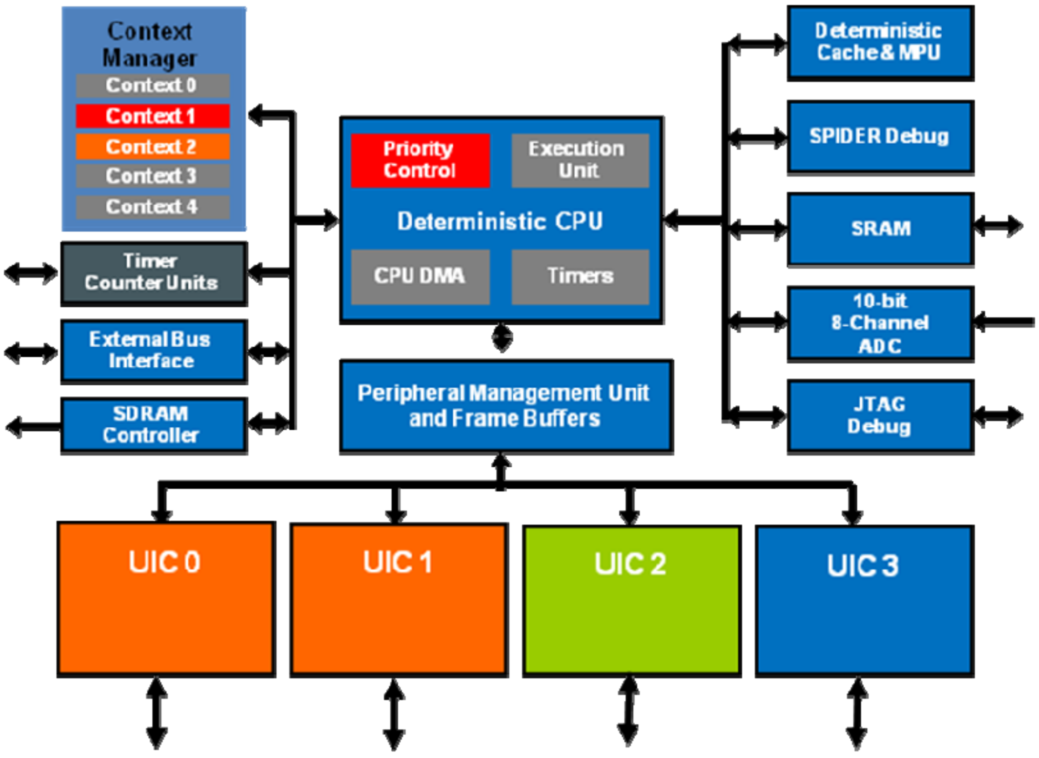
With fido, many tasks that were once conducted in software have been hardwired into the chip to provide deterministic real-time response, reliable separation of tasks for fail-safe operation and new on-chip debug features. By designing the functionality of a Real-Time Operating System (RTOS) Kernel into the silicon, system designers now have complete control over real-time task execution enabling deterministic response with faster development time and more secure operation. Although the architecture is new, the fido microcontroller executes the Freescale 68000 (CPU32+) instruction set, thereby utilizing tools that are well proven and reliable.

fido's "RTOS Kernel in a Chip" concept provides the following unique capabilities:

- An on-chip priority-based preemptive scheduler manages context execution automatically with no software overhead
- Five on-chip "Hardware Contexts" provide physical separation of task registers and memory access
- Context switching is deterministic and can be accomplished in a single clock cycle
- Priority control is unified and priorities are assigned to interrupts, contexts and even DMA, peripheral I/O data transfers and external bus masters – a high priority context will interrupt a lower priority DMA transfer with no software overhead or jitter
- Context timers ensure that threads are exited whenever a task takes too long or gets locked up – preventing system crashes
- On-chip memory protection ensures that one thread cannot step on the protected memory space of another – code changes in one task cannot corrupt previously tested threads
- Fail-safe capabilities that are impossible with an RTOS are now available - a fatal fault (such as a double bus fault) will stop only the affected context and is reported to the supervisor context – there is no interruption of other tasks and the supervisor context can correct the error

When tasks are managed by the RTOS, there is room for error. Depending upon the exact state of the application software and the RTOS, it can take an unpredictable amount of clock cycles to switch contexts. This causes jitter. It is the system designer's task to spend as much time as necessary tweaking the code or increasing the clock speed to arrive at the point that the jitter is acceptable. Even then, without endless hours of testing, it's hard to be 100 percent certain that jitter will always be under control under all operating conditions, especially when attached to an Ethernet network.

This fido architecture also includes four Universal I/O Controllers (UIC). Each of these is a programmable RISC engine that is optimized to emulate various I/O peripherals including 10/100 Ethernet, UART, GPIO, SPI, CAN etc. The UICs allow fido to be used in a wide range of applications with different I/O requirements. They also permit custom I/O architectures and protocols to be supported.



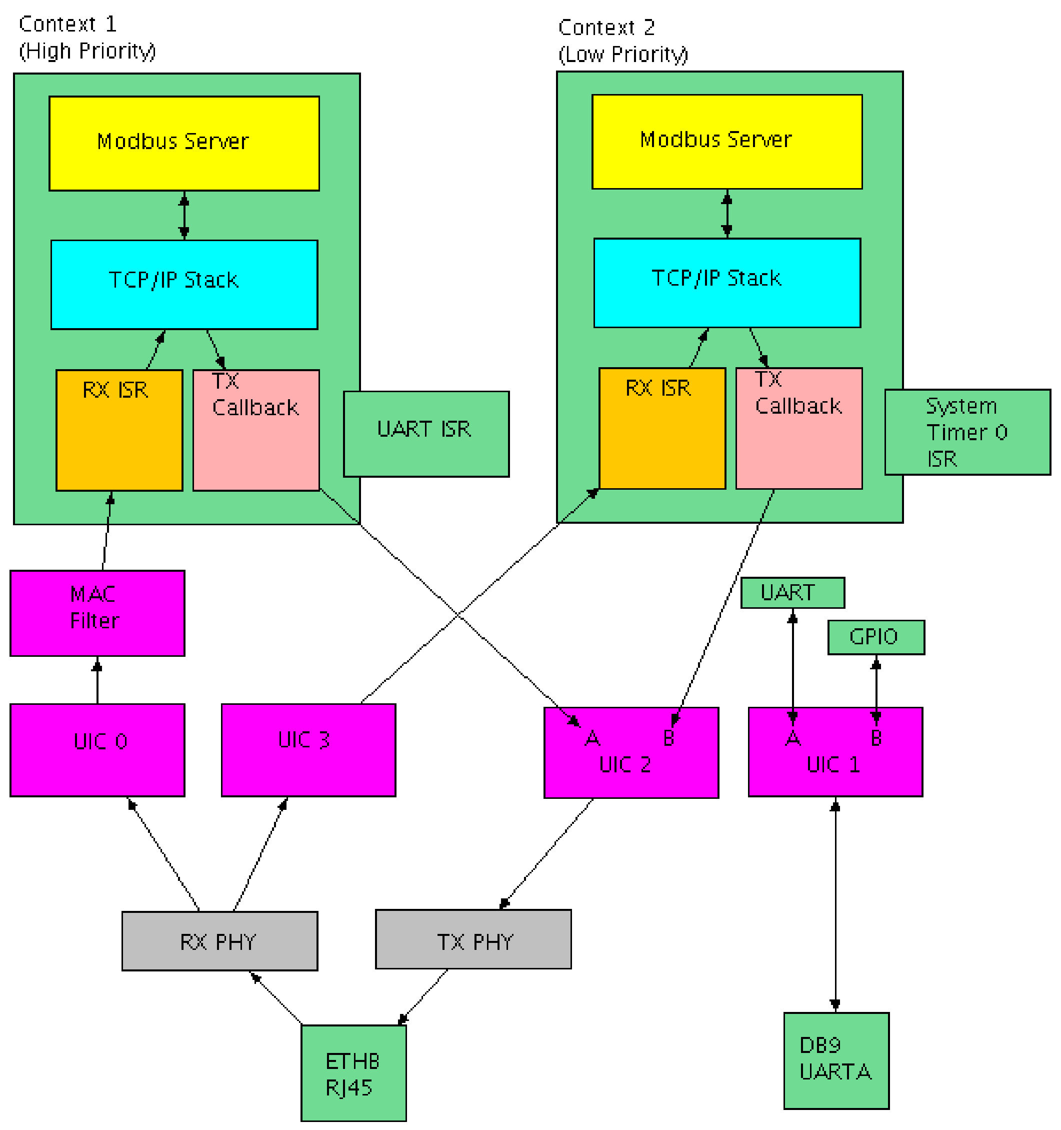
Block Diagram of the fido1100

The fido “RTOS Kernel in a Chip” and UICs provided specific advantages in the context of this White Paper. Two of fido's hardware contexts were used to provide environments for a low priority copy of the TCP/IP stack and a high priority copy of the stack. Having two copies eliminated a key component of the jitter encountered on the standard microprocessor described above.

fido's UICs are used as programmable Ethernet MACs. The firmware was tailored specifically for this application to provide separate data paths for high- and low-priority messages on both transmit and receive.

Additional UIC resources are still available to serve other, device-specific, I/O needs (SPI, CAN, UART, GPIO, custom protocol, etc.). There are also 3 remaining hardware contexts to manage additional processing tasks.

Determinism, the fido approach



The implementation using the fido approach is:

- Processing of high priority (Modbus server) and low priority (TCP server) requests are separated in different Hardware Contexts having different priorities. No conventional RTOS is used, rather the "RTOS Kernel in a Chip" architecture of fido is used to provide independent tasks of differing priorities.

- Two instances of the IP stack are created, one per Hardware Context. No code change was required on the IP stack. Because the hardware provides a platform to completely separate the data flows, no critical sections are necessary within the TCP/IP stack. Only the makefile/link process had to be modified to allow this.

- For incoming traffic, the standard Ethernet 100 firmware (used here in UIC's 0 and 3) was modified to filter incoming frames depending on their priority and MAC address: high priority frames are forwarded to high priority context, low priority frames are forwarded to low priority context.

- For outgoing traffic, the standard Ethernet 100 firmware (UIC 2) was modified to handle two logical input queues with different priorities. This provides two completely independent data channels.

NOTE: It is important to notice here that the standard fido silicon was used for this unique I/O application simply by modifying the firmware used in the UIC I/O processors. The programmability of the UIC permitted a new custom Ethernet MAC architecture to be created that is completely compatible with standard Ethernet, yet separates the data paths for high- and low-priority packets.

When mixing periodic high priority requests with bursts of low priority requests, the following results were obtained on a fido CPU at 66MHz:

	Without background traffic	With background traffic
RTT Min	937	957
RTT Median (upper than 80% of probability)	950	970
RTT High at 95% of probability.	960	1 020
RTT Max	1 059	1 087
RTT Jitter	***122***	***130***

A very favorable and predictable upper limit to response time and determinism was achieved. Compared to previous architecture we observe that:

- Jitter observed with fido is five to six times lower than 133 MHz RISC CPU.

- fido CISC CPU at 66 MHz achieved lower maximum response time compared to 133 MHz RISC CPU. fido, being CISC at 66 MHz, achieved better maximum response time and has less heating compared to 133 MHz RISC CPU. Another advantage of the CISC architecture is that the same software program requires less memory.

- Thanks to fido's innovative architecture, the high priority context is fail safe against errors in other contexts. It eases system testing as the code in each context can be qualified independently of code in the other contexts.

Conclusion

It is possible to achieve the deterministic performance required by real-time industrial applications using standard Quality of Service techniques. In order to do so, the software architecture of the device has to be adapted in order to separate as much as possible the processing of low and high priority flows. The main difficulty in achieving this on a standard CPU comes from jitter introduced by the TCP/IP stack (which is non re-entrant) and the task context switching time. These issues are easily solved thanks to the new fido architecture.

Advantages of the fido silicon architecture in this application are:

- Except for the physical interface, high & low priority flows are totally separated. There are no critical sections shared by the two as found in a standard architecture because of the IP stack.
- No RTOS is required thus saving costs.
- Use of hardware contexts reduces the jitter associated with switching from one context to the other to the bare minimum compared to a software task context switch.
- Hardware resources (input & output buffers, memory) are dedicated to each context. This ensures that low priority traffic bursts do not jeopardize resources necessary for the processing of high priority traffic.
- The high priority context is fail safe against potential errors in processing low priority requests which also eases system qualification.

With this approach, the fido CISC CPU running at half the speed of the RISC CPU achieved faster response time and jitter was reduced almost 6 times with higher system reliability.