HARDWARE IMPLEMENTATION OF THE TIME-TRIGGERED ETHERNET CONTROLLER*

Klaus Steinhammer and Astrit Ademaj
Vienna University of Technology, Real-Time Systems Group
Treitlstr. 3/182-1, A-1040 Vienna, Austria
[klaus,ademaj]@vmars.tuwien.ac.at

Abstract: Time-triggered (TT) Ethernet is a novel communication system that integrates real-time and non-real-time traffic into a single communication architecture. A TT Ethernet system consists od a set of nodes interconnected by a specific switch called TT Ethernet switch. A node consist of a TT Ethernet communication controller that executes the TT Ethernet protocol and a host computer that executes the user application. The protocol distinguishes between event-triggered (ET) and time-triggered (TT) Ethernet traffic. Time-triggered traffic is scheduled and transmitted with a predictable transmission delay, whereas event-triggered traffic is transmitted on a best-effort basis. The event-triggered traffic in TT Ethernet is handled in conformance with the existing Ethernet standards of the IEEE.

This paper presents the design of the TT Ethernet communication controller optimized to be implemented in hardware. The paper describes a prototypical implementation using a custom built hardware platform and presents the results of evaluation experiments.

1. INTRODUCTION

The temporal accuracy interval of information in a distributed real-time system is affected by the precision of the global view of time. This precision of the distributed time-base depends on the jitter of the message transmission delay over the network and the communication protocol. A predictable real-time communication system must therefore guarantee the message transmission within

*__Acknowledgments:__ This work has been supported in part by the European IST project DECOS under project No. IST-511764.

a constant transmission delay and bounded jitter [1]. Due to the open nature of Ethernet it is difficult to guarantee bounded transmission delays in systems that use standard Ethernet networks.

The duration of the message transmission over a network depends on the characteristics of the network traffic. It can be distinguished between two different scenarios: *cooperative senders* and *competing senders*. If senders are competing (as in standard Ethernet [2]) there is always the possibility that two or more messages are sent to a single receiver simultaneously. There are two possibilities to resolve this conflict: back-pressure flow control to the sender (this is the choice of the bus-based "Vintage Ethernet" [3]) or the storage of the messages within the network (this is the choice of switched Ethernet). Both alternatives involve increased message transmission jitter which is unsatisfactory from the point of view of real-time performance. It is therefore necessary that in real-time systems the senders must cooperate to avoid message transmission conflicts. There are many approaches that try to adapt the Ethernet technology such that it can be deployed in applications where temporal guarantees are necessary [4–8]. A summary of these implementations can be found on the Real-Time Ethernet web-page [9] while a comparison can be found in [6] and [7]. All these solutions use the concept of *cooperative senders* in order to guarantee constant transmission delays.

This paper presents the design of a Time-Triggered Ethernet Communication Controller (TTE controller) VHDL module which can be applied in an *System On a Programmable Chip* (SOPC) system and a prototypical implementation of a TTE controller using a custom made hardware board.

This paper is organized as follows: Section 2 gives an overview of the TT Ethernet architecture. The design of the TTE Controller is presented in Section 3 while the prototypical implementation is presented in Section 4. The experimental validation results are presented in Section 5, and the paper is concluded in Section 6.

2. TIME-TRIGGERED ETHERNET

A TT Ethernet system consists of a set of nodes interconnected by a specific switch called TT Ethernet switch. A node can be either a *standard Ethernet node* or a *TT Ethernet node*. A standard Ethernet node consist of a "commercial of-the-shelf" (COTS) Ethernet controller and a host computer transmitting only standard Ethernet messages. A TT Ethernet node consist of a TT Ethernet communication controller that executes the TT Ethernet protocol and a host computer that executes the one or more threads of a distributed (real-time) application.

Time-Triggered Ethernet [10] allows competing senders (standard Ethernet node) to *coexist* with cooperative senders (TT Ethernet node) on the same network while yet preserving the temporal predictability of the traffic among the cooperative senders. To integrate competing traffic seamlessly in the same network without interference of the cooperative traffic, Time-Triggered Ethernet introduces two message classes using the standardized Ethernet frame format [2]:

- **Standard Ethernet messages** are used for Event-Triggered (ET) traffic by competing transmitters, and
- **Time-Triggered (TT) messages**, which are transferred among cooperative senders.

Standard Ethernet messages are transmitted in those time intervals where the communication medium is not needed for transmission of time-triggered messages. In order to avoid that standard Ethernet traffic affects the temporal properties of the time-triggered traffic, the TT Ethernet switch preempt all standard Ethernet messages which are in the transmission path of time-triggered messages. This preemption mechanisms allows the transmission of TT messages with a constant transmission delay. As all ET messages are stored within the TT Ethernet switch, the preempted ET messages are retransmitted autonomously by the TT Ethernet switch after the transmission of the TT message has finished. If an ET message has to be dropped by the TT Ethernet switch because of buffer memory overflow, the lost ET message is retransmitted by the upper protocol layers of the sending node. TT messages are not stored within the network.

Introducing these two classes guarantees the compatibility to standard Ethernet COTS components and existing networking protocols without any modification of the competing transmitters on the one hand and the possibility to realize a scheduled message transfer between all cooperative senders by establishing and maintaining a global time base on the other hand.

There are two variations of TT Ethernet: standard TT Ethernet and safety-critical TT Ethernet. Safety-critical TT Ethernet is based on standard TT Ethernet which is described above, but introduces some enhancements to ensure that an arbitrary failure of one device within the cluster is tolerated [11]. This makes TT Ethernet suitable for various application cases - for multimedia applications, the industrial automation up to safety-critical applications like x-by-wire systems in the automotive and aeronautic domain.

The TT Ethernet communication controller presented in this paper is intended to be used in a standard TT Ethernet system.

3. DESIGN OF THE TT ETHERNET CONTROLLER

A TT Ethernet node consist of a TT Ethernet controller and a host computer. The data exchange interface between the host computer and the TT Ethernet controller is denoted as *communication network interface* (CNI). The CNI is implemented as dual ported memory, which serves as temporal firewall between the host and the communication controller [12].

The design consists of: the *transmitter blocks*, the *receiver blocks*, the *timer module*, the *configuration modules* and the *control module*.

The structure of the design of a TT Ethernet controller is depicted in Figure 1. The Communication Network Interface (CNI) and the host interface are not covered by the design of the TT Ethernet controller because they depend on the hardware used for the implementation of the TT Ethernet controller.

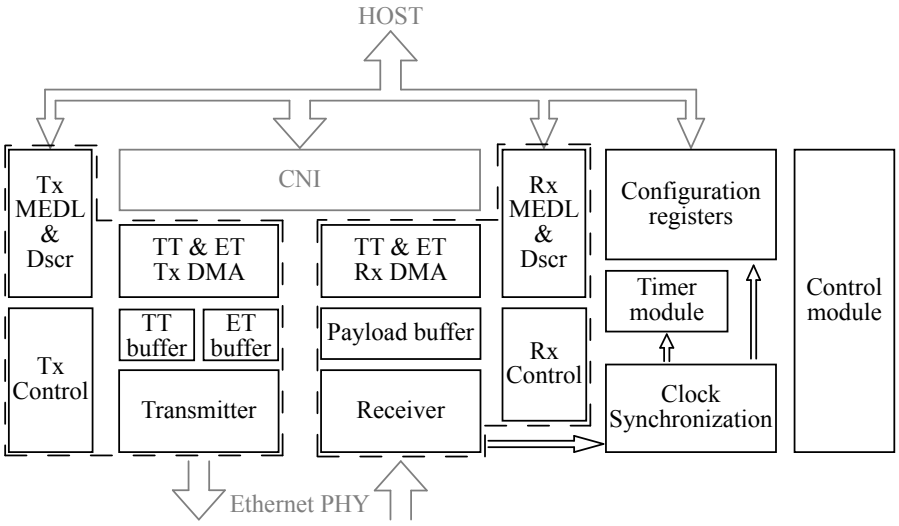

Figure 1. Structure of the TT Ethernet Controller Design

The Control Module

The *control module* coordinates the functions of the different components for TT message handling. ET message handling can be enabled by the host at any time, independent of the state of the control module. The controllers internal state equals to one of the protocol states specified at the TT Ethernet protocol specification [13].

There are two possible startup scenarios for a controller:

1 In case it is configured to be a *clock rate-master node*, the controller waits a certain period of time for an incoming synchronization message. If a synchronization message is received, another controller acts as primary rate master, and this controller has to behave as backup rate master, synchronizing its clock to the primary rate master until it fails. If no synchronization message can be received at startup, this controller becomes the primary rate-master node. Then the controller begins to transmit TT messages as configured, starting with a synchronization message.

2 If the controller is configured as *timekeeping node*, it adjust its local time to the clock of the rate master. At startup, the controller waits until it receives enough synchronization messages so that the local time can be synchronized. If a certain precision has been reached, the controller starts to transmit TT messages as configured.

Configuration Modules

The *configuration modules* consists of two components: the configuration registers and the MEDL memory.

The *configuration registers* are static and are set by the host before enabling the message transfer of the controller. During runtime, the host can access several read-only status registers to be informed about the current controller operational status.

The schedule information of each packet is stored in the *Message Descriptor List (MEDL) memory*. There is one MEDL memory for receiving and one for transmitting messages. The MEDL is organized in 16 linked lists (TT Ethernet supports periodic transmission of messages with 16 different periods). Each MEDL record contains a *MEDLPTR* entry, which is used to link to the next MEDL record. In each list the entries are sorted with rising transmission offset. If a new entry is inserted into the linked list the host has to make sure that it is inserted at the right position within the list, the actual memory location of the entry is not of importance. An incorrectly sorted list would result in severe schedule errors. A DMA mechanism autonomously stores status information about the actual transmission and fetches the next entry of each list after the actual message transfer has finished. To gain concurrent access to the next messages of each linked list, up to 16 active MEDL entries are duplicated inside of the controller.

The host stores the MEDL data in a dual ported RAM. Only memory with two independent ports allows the simultaneous access for the host and the controller

to the MEDL data. There is only one exception; it has to be ensured that a single memory location is only written by one party at a time. If the message transmission is enabled, the controller has a fixed access schema for the MEDL so the host cannot change a MEDL entry at any time. The host has to follow a schedule when it is allowed to modify a certain MEDL entry. Reading the MEDL entries is allowed at any time, however, the read operation of the host is not atomic and may produce inconsistent data. This problem is solved by deploying the non-blocking-write protocol [14].

TT data transmission

After the controller is configured, the TT message data transmission is done autonomously. If the transmission time has arrived, the controller starts to transmit the message header. At the same time one of the controller's DMA engines transfers the packet's payload data from the CNI memory to an internal buffer memory. After the header transmission has finished, the controller starts to read data from the internal buffer and transmits it after the header, until the final byte is transmitted out of the buffer. If more then 59 bytes are sent, the minimum message length as defined by the IEEE 802.3 standard is met so the packets checksum is transmitted and the operation has finished. If less than 60 bytes are sent the controller adds padding bytes until the required 60 bytes are transmitted before adding the checksum.

The checksum is generated on-the-fly while the data is sent. So it is impossible for the host to void a pre-calculated checksum by changing data while the controllers DMA is active.

TT Ethernet distinguishes between different kinds of TT packets: TTsync, TTdiag and TT data messages. The difference between TTsync and TTdiag on the one hand and TT data messages on the other hand, is that only TT data messages have payload data which is stored in the CNI memory. Therefore it is possible for the sender module to generate TTsync and TTdiag messages on-the-fly without the use of a DMA mechanism. This decreases the processing time of these messages and enables the generation of packets with the shortest possible inter-packet gap.

The schedule information of each packet is stored in the MEDL memory. The schedule information is the basis for the trigger generator, which signals when each message transmission time has arrived. With this signal, the TX DMA is started (if a TT data message is scheduled) and the sender starts to transmit the packet. If the whole packet is sent, the outcome of the message transmission is updated in the status fields of the MEDL entry and the next MEDL entry of this message period is loaded.

TT data reception

The data reception is similar to the data transmission. The receiver transfers the data into an internal buffer while checking the validity. As the validity of the data can only be determined after the whole packet is examined and the frame checksum is compared, the DMA which copies the data into the CNI memory is started after the whole packet is received. If the packet was invalid, all the data is cleared from the buffer, and only the status information is updated in the CNI memory.

Synchronization messages get a special treatment by the receiver. Additional to the internal buffer, some content of the payload-data and the reception time-stamp are also written into a FIFO buffer which is connected to the clock synchronization interface.

Other than the transmission, the reception needs a validity window with configurable size, because all nodes in a cluster cannot be synchronized perfectly, and this is compensated by this reception window. If a message is received within the bounds of this window, it is treated as valid in terms of reception time. If no message is received within the reception window, the message associated to this window is treated as lost and its status is changed accordingly. The size of this reception window is configurable and chosen according to the quality of the clock synchronization within the cluster. The better the clocks are synchronized among each other, the smaller the size of the reception window can be configured.

The message transmission delay between two TT Ethernet controllers caused by the network is constant. This delay depends on the number of switches and the length of the cables in the network.

In order to simplify the clock synchronization mechanism, the TT Ethernet controller adjusts each time-stamp taken for any incoming message by the message transmission delay. These values have to be configured by the host before the controller is enabled.

ET message handling

If the TT part of the controller is enabled, ET data transmission is done in between two TT messages, when the network is idle. If TT message handling is disabled by the host, ET messages can be transmitted at any time. If the host configures an ET message for transmission, the controller waits for an inter-packet-gap between two scheduled TT messages which is big enough for the ET message to be sent. The controller will not change the order of ET packets if there are several messages to be sent. So it is guaranteed that the receivers

don't have to reorder the packets. This is necessary as especially embedded nodes don't have enough reception buffer space to hold several packets. They depend on the right reception order or they drop all packets which are out of order and request re-transmissions. This would waste network bandwidth, and so it is avoided in this design.

The reception of ET messages follow the same procedure as TT message reception except that ET messages do not have to be received within a receive-time window.

Timer Module

The timer module provides several counters. One of them is used to implement a configurable frequency divider. It is used to apply rate correction in order to adjust the local clocks frequency to the frequency of the global time of the cluster. This counter is decremented with each Microtick. The Microtick frequency is defined by the used hardware oscillator. Each time this counter reaches zero, a pulse (the Macrotick) is generated and it is reloaded with a value computed by the clock synchronization subsystem. The Macrotick pulse is used to increment the local time counter. By changing the length of a Macrotick period, the rate of the local clock can be adjusted.

The timer module provides an interface for the clock synchronization subsystem. The actual computation of the synchronization algorithm is done using a CPU. This has the advantage, that the algorithm can be implemented in software. So it is easy to apply changes to the clock synchronization algorithm.

In case of an incoming TTsync message, the receiver stores the reception time stamp and the timing data included in the message into a buffer. The content of this buffer can be read by the CPU subsystem which uses this information to compute the difference between the local time and the time to synchronize to. It performs the clock state correction such that the actual time difference is compensated and calculates and applies a new Microtick/Macrotick rate value, such that this time difference is reduced until the next TTsynch message will be received. In the case that the time difference is larger than 1 Macrotick, the clock synchronization module will restart and set the local time to the time obtained from the clock synchronization message.

When, for a certain amount of time, the time difference is smaller than a Macrotick length, the rate and time value of the local clock is adjusted to the reference clock and it signaled that the clocks are synchronized and the message transmission can be enabled.

4. TTE CONTROLLER PROTOTYPE

An overview of our hardware designed for a TTE controller implementation is shown in Figure 2.

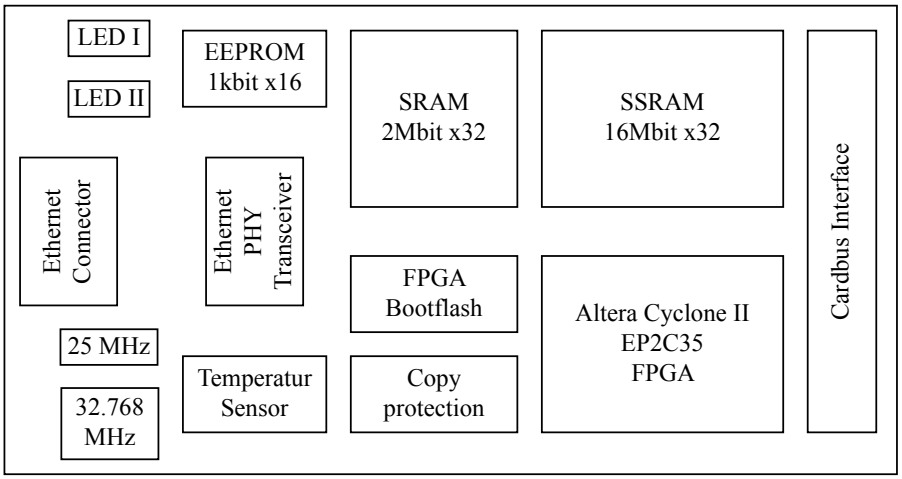

Figure 2. TTE Controller Card Overview

The TTE Controller Card interfaces to the host via a Cardbus interface. The configuration and data area of the card are mapped into the host computers memory space.

All the functionality of the card is implemented in the FPGA, an EP2c35 Altera Cyclone II device [15]. The Ethernet PHY transceiver is used to convert the IO standard of the FPGA to the Ethernet physical level standard. The 2 MB sized SSRAM memory holds the CNI memory for real-time messages as well as buffer space for standard Ethernet messages. The non-volatile memories are holding the configuration data of the FPGA (which bases on volatile SRAM cells and needs therefore to be configured when powered up) and some configuration data, which is unique to each card and needed to identify the device on the network and by the host. A sensor which can be used to monitor the temperature of the card and shutdown the system in case of overheating is also implemented as well as a copy protection circuit to prevent theft of the used IPs. The 25MHz Oscillator is needed by the Ethernet PHY; the FPGA uses the 32.768MHz oscillator. Status information of the controller is shown by the two dual-color LEDs.

The TT Ethernet controller was included in a SOPC system applied to the FPGA. Table 1 shows the compilation results of the TT Ethernet controller module excluding the clock synchronization CPU and the host interface.

Table 1. Compilation results of the TT Ethernet controller module

Size	18598 Logic Cells
Memory usage	144896 bits
Maximum Microtick frequency	70.63 MHz

5. EXPERIMENTAL EVALUATION

The hardware setup consisted of a TT Ethernet switch as described in [16] which was connected to three nodes equipped with a TT Ethernet controller as described in Section 4. Node 0 was configured to be the rate master node, Node1 and 2 were timekeeping nodes. A fourth node equipped with a COTS Ethernet controller and running Windows XP was connected to generate ET packets.

Minimum Inter-Packet-Gap

This experiment has been performed to show that the system can handle packet bursts with a minimum inter-packet-gap of 3 Macroticks. Node 0 was configured to transmit packets according to the schedule listed in Table 2 and Node 1 received all of the messages. Node 2 and the ET packet generator node were disconnected during the experiment.

Table 2. TT message schedule used at this experiment

type	*length*	*period*	*offset*	*duration*
TTsync	64 Bytes	7,8ms	0,00μs	5,80μs
TTpM	666 Bytes	7,8ms	7,60μs	54,00μs
TTpM	122 Bytes	7,8ms	62,94μs	10,48μs

The schedule listed in Table 2 creates message bursts with an inter-packet-gap between the two data messages of 1.34μs which is equal to 2.75 Macroticks. The host CPU on Node 1 was configured to monitor the packet reception. After 10000 transmission periods, the data recorded by Node 1 showed that each packet was received without errors and in time.

Clock Synchronization

This experiment deals with the synchronization of the local clocks within the nodes to the global time of the cluster. For these experiment a simple *master-slave synchronization algorithm* was used. The cluster consisted of three TT Ethernet nodes whereas one node is configured as rate-master node, sending TTsync messages with a period of 7.8ms. The other two nodes were timekeeping nodes, which synchronized their local clocks to the clock of the rate-master. Each of the three nodes was configured to transmit and receive at least one packet per period. The transmission offset of the data messages was constant; the period was changed in the course of the experiment (see Table 3).

Table 3. Used message scheduling of the experiments

Period	*Offset*	*Sender*	*Receiver*
for all experimental runs :			
7.8ms	0	Node 0	Node 1, Node 2
for experimental run (*x*) $0-3$ **:**			
61μs - 488μs	2^{-16+x}	Node 0	Node 1, Node 2
61μs - 488μs	2^{-15+x}	Node 2	Node 0, Node 1
61μs - 488μs	$2^{-15+x} + 2^{-16+x}$	Node 1	Node 2
for experimental run (*x*) $4-15$ **:**			
976μs - 2s	$2^{-16+x} + 2^{-23+x}$	Node 0	Node 1, Node 2
976μs - 2s	$2^{-15+x} + 2^{-23+x}$	Node 2	Node 0, Node 1
976μs - 2s	$2^{-15+x} + 2^{-16+x} + 2^{-23+x}$	Node 1	Node 2

The experiment was repeated 32 times. At the first run, the data messages were transmitted using a period of 61μs. Then the period was doubled at each subsequent run. After 16 runs Node 4, a standard PC computer with a COTS Ethernet controller running Windows XP, was enabled to send one ET message with a length of 1500 bytes every 250μs and the period was reset to 61μs to repeat the first 16 runs with enabled ET traffic.

For each experimental run, all nodes measure the receive time deviation of each message from the scheduled arrival time of the message. As each message is only accepted if it is received within a predefined reception window, the reception time deviation of a message is of particular importance. The maximum measured deviation of all experimental runs are summarized at Table 4.

As the measurement results listed in Table 4 are showing, the receive time deviation with ET traffic enabled is the same as without ET traffic, the maxi-

Table 4. Measured time deviations of arriving data messages

	Node 0 earliest	*Node 0 latest*	*Node 1 earliest*	*Node 1 latest*	*Node 2 earliest*	*Node 2 latest*
max. deviation without ET	-9μT	9μT	-9μT	8μT	-7μT	7μT
max. deviation with ET	-6μT	11μT	-5μT	8μT	-5μT	8μT

mum measured receive time deviations of the messages are still below half a Macrotick (25μT), so in both experiments there were no lost messages.

6. CONCLUSION

A node within a TT Ethernet system consists of a host computer and a communication controller. The host computer executes one or more application tasks, while the communication controller is in charge of executing the TT Ethernet communication protocol. The host and the communication controller are connected via the Communication Network Interface (CNI). The TT Ethernet communication controller can handle both, ET and TT messages.

There are several ways to implement a communication controller for TT Ethernet:

- In software as additional task running on the host computer
- As software running on a dedicated CPU
- As a dedicated hardware device

In this paper a design of a TT Ethernet controller is introduced, which is optimized for implementation in hardware. For the evaluation of the design of the TT Ethernet controller, a prototype implementation was developed.

A TT Ethernet system consisting of a TT Ethernet switch and several TT Ethernet nodes was evaluated in the course of this work. The achievable precision of the system was measured first. One experimental run for each possible message transmission period defined by the TT Ethernet specification was made. The measurements showed a worst case precision of 18μT or 342 ns. Then the experiments where repeated applying ET traffic to the network. The measurements of these experiments show the same result.

The last experiment was executed to show that the TT Ethernet controller is able to handle TT packet bursts with a minimum inter-packet-gap of 3 Macroticks.

By analyzing the evaluation experiments it can be concluded that a design which applies the mechanisms of TT Ethernet and which is carefully implemented is able to provide a message transmission environment which exhausts the limits of the Ethernet technology while still being able to support standard Ethernet devices without changes at any device within the cluster.

REFERENCES

[1] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997. ISBN 0-7923-9894-7.

[2] IEEE Standard 802.3, 2000 Edition. Carrier Sense Multiple Access with Collision Detect on (CSMA/CD) Access Method and Physical Layer Specifications., 2000.

[3] Bernard Jouga. How Ethernet becomes Industrial. *Workshop: The use of Ethernet as a fieldbus*, September 2001.

[4] Chitra Venkatramani and Tzi-cker Chiueh. Supporting Real-Time Traffic on Ethernet. In *Real-Time Systems Symposium (RTSS)*, pages 282–286, December 1994.

[5] Seok-Kyu Kweon and Kang G. Shin. Achieving Real-Time Communication over Ethernet with Adaptive Traffic Smoothing. In *Sixth IEEE Real-Time Technology and Applications Symposium*, pages 90–100, May 2000.

[6] Jean-Dominique Decotignie. Ethernet-Based Real-Time and Industrial Communications. *Proceedings of the IEEE, Volume: 93, Issue: 6*, 2005.

[7] Max Felser. Real-Time Ethernet-Industry Prospective. *Proceedings of the IEEE, Volume: 93, Issue: 6*, pages 1118–1129, June 2005.

[8] Paulo Pedreiras, Luis Almeida, and Paolo Gai. The FTT-Ethernet Protocol: Merging Flexibility, Timeliness and Efficiency. In *14th Euromicro Conference on Real-Time Systems*, June 2002.

[9] Website: Information about Real-Time Ethernet in Industry Automation. $http://www.Real-Time-Ethernet.de$, 2004. Online; Accessed 19-October-2006.

[10] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. The Time-Triggered Ethernet (TTE) Design. *8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC), Seattle, Washington*, May 2005.

[11] Astrit Ademaj, Hermann Kopetz, Petr Grillinger, Klaus Steinhammer, and Alexander Hanzlik. Fault-tolerant time-triggered ethernet configuration with star topology. *Dependability and Fault Tolerance Workshop*, Mar. 2006.

[12] Hermann Kopetz. The CNI: A Generic Interface for Distributed Real-Time Systems. Confidential draft, Technische Universitat Wien, Institut fur Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1998.

[13] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. Time-Triggered Ethernet Protocol Specifcation. currently unpublished, Technische Universitat Wien, Institut fur Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2006.

[14] Hermann Kopetz and Johannes Reisinger. NBW: A Non-Blocking Write Protocol for Task Communication in Real-Time Systems. In *Proceedings of the 14th Real-Time Systems Symposium*, pages 290–299, Raleigh-Durham, North Carolina, USA, December 1993.

[15] Altera Corp. *Cyclone Device Handbook*. $http://www.altera.com/literature/hb/cyc/cyc_c5v1.pdf$, August 2005. Online; Accessed 19-October-2006.

[16] Klaus Steinhammer, Petr Grillinger, Astrit Ademaj, and Hermann Kopetz. A Time-Triggered Ethernet (TTE) Switch. *Design, Automation and Test in Europe, Munich, Germany*, March 2006.