NTI: A Network Time Interface M-Module for High-Accuracy Clock Synchronization

Martin Horauer[1], Ulrich Schmid[2] and Klaus Schossmaier[2]

[1] Vienna University of Technology, Department of Computer Technology, Gußhausstraße 27, A-1040 Vienna, Austria
[2] Vienna University of Technology, Department of Automation, Treitlstraße 1, A-1040 Vienna, Austria

http://www.auto.tuwien.ac.at/Projects/SynUTC/

Abstract. This paper[⋆] provides a description of our *Network Time Interface* M-Module (NTI) supporting high-accuracy external clock synchronization by hardware. The NTI is built around our custom *Universal Time Coordinated Synchronization Unit* VLSI chip (UTCSU-ASIC), which contains most of the hardware support required for interval-based clock synchronization: A state and rate adjustable clock device with a high resolution, automatically maintained accuracy intervals, interfaces to GPS receivers, and various timestamping features. Designed for maximum network controller and CPU independence, our NTI provides a turn-key solution for adding synchronized clocks to distributed real-time systems built upon hardware with M-Module interfaces.

1 Introduction

Designing distributed fault-tolerant real-time applications is usually considerably simplified when synchronized clocks are available. Temporally ordered events are in fact beneficial for a wide variety of tasks, ranging from relating sensor data gathered at different nodes up to fully-fledged distributed algorithms, see [5] for some examples. Providing synchronized local clocks, whose mutual deviation among non-faulty ones has to be smaller than a certain *precision*, is known as the *internal clock synchronization* problem. Numerous solutions have been worked out —at least in scientific research— under the term *fault-tolerant clock synchronization*, see [11] for an overview.

If system time provided by synchronized clocks must also have a well-defined relation to *Universal Time Coordinated* (UTC), the only official and legal standard time, then the *fault-tolerant external clock synchronization* problem needs to be addressed. Unlike internal synchronization, it did not receive much attention until recently, when highly *accurate* (i.e. close to UTC) and cheap receivers for the *Global Positioning System* (GPS) became widespread, see [1].

[⋆] This research is part of our project SynUTC, which has been supported by the Austrian Science Foundation (FWF) under grant P10244-ÖMA, the OeNB under grant 6454, the BMWV under contract Zl.601.577/2-IV/B/9/96, and the START programme Y41-MAT.

Most published solutions restrict their attention to purely software-based approaches, targeting a synchronization in the ms-range only, like the well-known *Network Time Protocol* (NTP) [8] designed for disseminating UTC among workstations throughout the Internet. Considerably better results can be achieved with any clock synchronization algorithm if some dedicated hardware support is present. For instance, the pioneering *Clock Synchronization Unit* (CSU) of [3] allows to construct synchronized clocks with a precision in the 10 μs-range for broadcast networks.

Our *Network Time Interface* (NTI) has been designed to support fault-tolerant external clock synchronization in distributed system, where the nodes are within a distance of few 100 meters interconnected by a packet-oriented communications subsystem. Implemented as an M-Module (see Section 2.1), the NTI allows to extend state-of-the-art real-time systems technology with synchronized clocks providing a precision/accuracy in the 1 μs-range. Apart from advanced algorithmic issues, this undertaking primarily requires hardware support for a sophisticated rate and state adjustable clock (see Section 2.2) as well as exact timestamping of clock synchronization packets (see Section 2.3 and 2.4). The appropriate features of the NTI are presented in this paper, however, lack of space prohibited us to cover the full size of the underlying technical report [2].

2 NTI Architecture

The basic hardware components required for clock synchronization are outlined in Figure 1. Each node has to be equipped with a hardware clock, in our case the *Universal Time Coordinated Synchronization Unit* (UTCSU), a general purpose CPU (the node's central processor or, preferably, a dedicated microprocessor or microcontroller) responsible for executing the software-part of the clock synchronization algorithm, and a *Communication Coprocessor* (COMCO), which provides access to the network by reading/writing data packets from/to (shared) memory independently of CPU operation (e.g. via DMA). For external synchronization purposes, some nodes need to be provided with external time sources like GPS satellite receivers. Although such nodes have additional functionalities, their hardware architecture remains to be the same.

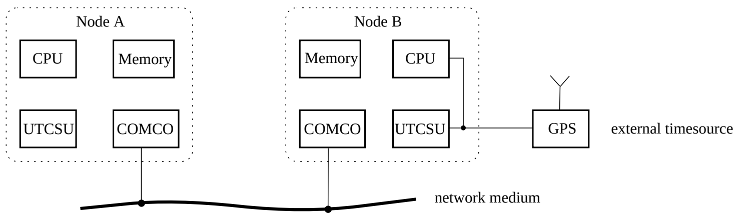


Fig. 1. Basic clock synchronization hardware architecture

2.1 NTI Design

M-Modules [7] are an open, simple, and robust mezzanine bus interface primarily designed for VME carrier boards, which are commonly used in Europe. *MA-Modules* are enhanced M-Modules, providing a 32 bit data bus instead of the 16 bit one of the original M-Modules. The address space consists of 256 bytes *I/O-space* accessible via the standard M-Module interface and up to 16 MB of *memory-space* addressed by multiplexing the MA-Module data bus. The asynchronous bus interface requires the module to generate an acknowledge signal for termination of a bus cycle only, thus minimizing the control logic on-board the M-Module. Further signals in the M-Module interface comprise a single vectorized interrupt line and two additional DMA control lines. The unit construction design of the 146×53 mm (single-height) M-Modules provides a peripheral I/O D-sub connector on the front panel, two plug connectors to the carrier board for peripheral I/O and MA-interface, and an intermodule port connector for interconnecting several M-Modules.

We found MA-Modules well-suited for crafting the prototype of our clock synchronization hardware, since we did not want to bother ourselves with developing a fully-fledged node hardware, but rather to extend existing CPU boards with adequate support. In this and the following subsections, we will provide an overview of the NTI features. Figure 2 shows the major components of an NTI MA-Module.

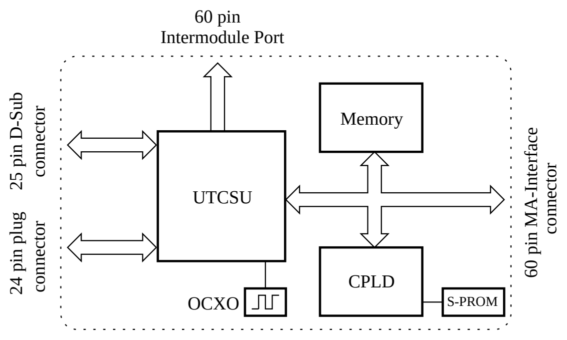


Fig. 2. NTI block diagram

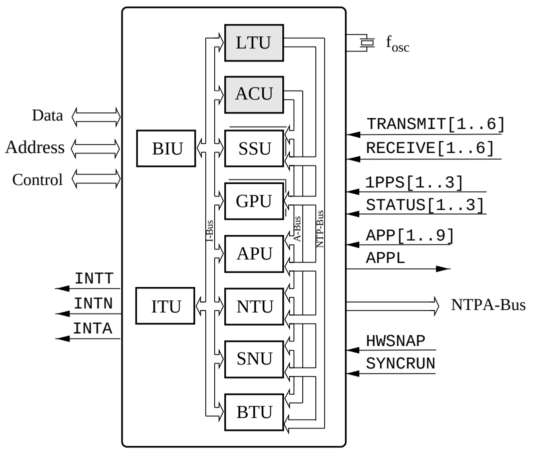
The UTCSU-ASIC contains most of the dedicated hardware support for clock synchronization, see Section 2.2 for more details. It is driven by an on-board temperature-compensated (TCXO) or ovenized (OCXO) quartz oscillator; alternatively, an external frequency source like the 10 MHz output of a high-end GPS receiver can be used. All UTCSU input/output lines are available via the M-Module's front-panel 25 pin D-sub connector and the 24 pin plug connector. The intermodule port is eventually used for future extension modules, and to facilitate internal connection of modularized GPS receivers. High-speed optocouplers are provided for all inputs to ensure a decoupled and reliable interface.

The memory consists of four 64K × 16 bit SRAM chips, which serves as control and data interface between the CPU and the COMCO, and provides special functionality for COMCO accesses, see Section 2.3 and 2.4.

All required decoding and glue logic of the NTI is incorporated in a single, in-circuit programmable *complex programmable logic device* (CPLD), which has been programmed in VHDL. It adapts the UTCSU and the memory to the MA-Module interface, forwards interrupt requests from the UTCSU to the carrier-board, generates the acknowledgement signal terminating a bus cycle, and gives access to the serial PROM that stores identification and revision information.

2.2 UTCSU Features

In this subsection, we provide a succinct overview of the wealth of functionality of our custom *Universal Time Coordinated Synchronization Unit* (UTCSU); further information is available in [10]. Manufactured as an ASIC in 0.7 μm CMOS technology, the UTCSU accommodates about 80,000 gates on a 100 mm^2 die packed into a 208-pin MQFP case. Due to its flexible bus interface, featuring dynamic bus sizing and little/big endian byte ordering, the UTCSU can be used in conjunction with virtually any 8, 16 and 32 bit CPU. Figure 3 gives an overview of the major functional blocks inside the UTCSU.



unit	full name
ACU	Accuracy Unit
APU	Application Unit
BIU	Bus Interface Unit
BTU	Built-In Test Unit
GPU	GPS Unit
ITU	Interrupt Unit
LTU	Local Time Unit
NTU	Network Time Interface Unit
SNU	Snapshot Unit
SSU	Synchronization Subnet Unit

Fig. 3. Interior of the UTCSU

The centerpiece of our chip is a local clock (LTU) utilizing a 56 bit NTP-time format, which maintains a fixed point representation of the current time with 32 bit integer part and 24 bit fractional part, see [8]. Clock time can be read atomically as a 32 bit *timestamp* with resolution $2^{-24} \approx 60$ ns that wraps around every 256 s, and a 32 bit *macrostamp* containing the remaining 24 most-significant bits of seconds along with an 8 bit checksum protecting the entire time information.

The local clock of the UTCSU can be paced with any oscillator frequency f_{OSC} in the range of $1 \ldots 20$ MHz, is fine-grained rate adjustable in steps of about 10 ns/s, and supports state adjustment via continuous amortization as well as (optional) leap second corrections in hardware. Those outstanding features are primarily a consequence of our novel *adder-based* clock design, which uses a large (91 bit), high-speed adder instead of a simple counter for summing up the elapsed time between succeeding oscillator ticks.

To achieve both internal and external clock synchronization, our approach relies on an interval-based paradigm: Real-time t (usually UTC) is not just represented by a single time-dependent clock value $C(t)$ here, but rather by an *accuracy interval* $\boldsymbol{A}(t)$ that must satisfy $t \in \boldsymbol{A}(t)$. More specifically, accuracy intervals are provided by combining an ordinary clock $C(t)$ with a time-dependent interval of accuracies $[-\alpha^-(t), \alpha^+(t)]$ taken relatively to the clock's value, leading to $\boldsymbol{A}(t) = [C(t) - \alpha^-(t), C(t) + \alpha^+(t)]$. Since accuracy intervals need to be maintained dynamically, they are quite small on the average. For an in-depth treatment of system modelling and clock synchronization algorithms based on this paradigm consult [9].

In order to support interval-based clock synchronization, the UTCSU contains two more adder-based "clocks" in the ACU that are also driven by the oscillator frequency f_{OSC}. They are responsible for holding and automatically deteriorating the 16 bit accuracies $\alpha^-(t)$ and $\alpha^+(t)$ to account for the maximum oscillator drift. Both can be (re)initialized atomically in conjunction with the clock register in the LTU. In addition, some extra logic suppresses a wrap-around and zero-masks potentially negative accuracies during continuous amortization.

Three different functional blocks are in charge of time/accuracy-stamping: First, trigger signals sample the current local time/accuracy into dedicated UTCSU registers in the SSU, when a *clock synchronization packet* (CSP) arrives or leaves a node, see Section 2.3. Six independent SSUs are provided to facilitate fault-tolerant (redundant) communications architectures or gateway nodes. Second, three independent GPUs are provided for timestamping the *one pulse per second* (1pps) signal —indicating the exact beginning of a UTC second— from up to three GPS receivers. Note that this simple interface is sufficient for coupling GPS receivers, since additional and less time critical information is usually provided via a serial interface and handled off-chip the UTCSU. Finally, nine independent application time/accuracy-stamping inputs are provided by the APU. Note that additional application-related features can be realized off-chip by tapping the 48 bit wide multiplexed *NTPA-Bus*, which exports the entire local time and accuracy information at full speed.

There are many different interrupt sources inside the UTCSU, which are all (statically) mapped onto three dedicated UTCSU interrupt outputs INTN (network-related), INTT (timer-related) and INTA (application-related). Since M-Modules provide only a single interrupt line for signalling a vectorized interrupt, the NTI is responsible for further mapping INTN, INTT, and INTA onto a single interrupt line and generating the appropriate interrupt vector.

Last but not least, the UTCSU is equipped with features for test (BTU) and debugging purposes (SNU). This includes calculation of checksums, blocksums and signatures for local time, snapshots of certain registers to facilitate an experimental evaluation of precision/accuracy, and (re)start operations.

2.3 Timestamping Features

Providing hardware support for highly accurate/precise clock synchronization is primarily driven by the requirement of exact timestamping of CSPs at both sending and receiving side. In fact, the work of [6] revealed that even n ideal clocks cannot be synchronized with a worst case precision less than $\varepsilon\,(1 - 1/n)$ in presence of a *transmission/reception time uncertainty* ε, which is defined as the variability of the difference between the real times of CSP timestamping at the peer nodes. Unfortunately, there are several steps involved in packet transmission/reception that could contribute to ε, cf. [3]:

1. Sender-CPU assembles the CSP
2. Sender-CPU signals sender-COMCO to take over for transmission
3. Sender-COMCO tries to acquire the network medium
4. Sender-COMCO reads CSP data from memory and pushes the resulting bit stream onto the medium
5. Receiver-COMCO pulls the bit stream from the medium and writes CSP data into memory
6. Receiver-COMCO notifies receiver-CPU of packet reception via interrupt
7. Receiver-CPU processes CSP

Purely software-based clock synchronization approaches perform CSP timestamping upon transmission resp. reception in steps 1 resp. 7, which means that ε incorporates the medium access uncertainty $3 \to 4$, any variable network delay $4 \to 5$, and the reception interrupt latency $6 \to 7$. The first one can be quite large for any network utilizing a shared medium, and the last one is seriously impaired by code segments with interrupts disabled. Fortunately, in our LAN-based setting, we can safely neglect the contribution from $4 \to 5$ since there are no (load- and hop-dependent) queueing delays from intermediate gateway nodes[1]. Therefore, the resulting transmission/reception uncertainty emerges primarily from $1 \to 4$ resp. $5 \to 7$ at the sending resp. receiving node itself.

In an effort to reduce ε, clock synchronization hardware should thence be placed as close as possible to the network facilities. Ideally, a CSP should be timestamped at the sender resp. receiver exactly when, say, its first byte is pushed on resp. pulled from the medium. However, this needs support from the interior of the COMCO, which is usually not available. In order to support existing network controller technology, a less tight method of coupling has to be considered.

[1] Note that our approach can also be adopted to more general topologies commonly known as WANs-of-LANs, provided that all gateway nodes are also equipped with the NTI.

For this purpose, our NTI uses a refinement of the widely applicable DMA-based coupling method proposed in [3]. The key idea is to insert a timestamp on-the-fly into the memory holding a CSP in a way that minimizes the transmission/reception uncertainty. More specifically, a modified address decoding logic for the memory is used, which

– generates trigger signals that sample a timestamp into dedicated UTCSU registers when a certain byte within the transmit resp. receive buffer for a CSP is read resp. written by the COMCO, and
– transparently maps the sampled transmit timestamp into some portion of the transmit buffer.

To illustrate the entire process of CSP stamping, we briefly discuss one possible scenario depicted in Figure 4. Whenever the COMCO fetches data from the transmit buffer holding the CSP for transmission, it has to read across the particular address that causes the CPLD to generate the trigger signal `TRANSMIT`. Upon occurrence of this signal, the UTCSU puts a *transmit timestamp* into a dedicated sample register, which is transparently mapped into a certain succeeding portion of the transmit buffer and hence automatically inserted into the outgoing packet. Note that the trigger address and the mapping address may be different. By the same token, when the COMCO at the receiving side writes a certain portion of the receive buffer in memory, the trigger signal `RECEIVE` is generated by the CPLD, which causes the UTCSU to sample the *receive timestamp* into a dedicated register. Subsequently, the timestamp can be saved in an unused portion of the receive buffer by the CPU upon reception notification or by a similar transparent mapping technique, see Section 2.4.

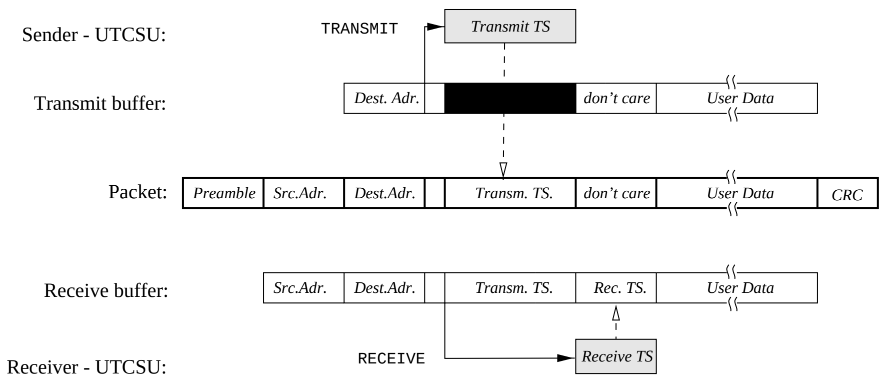


Fig. 4. Packet timestamping

The proposed approach works for any COMCO that directly accesses CSP data in memory. Suitable chipsets are available for a wide variety of networks, ranging from fieldbusses over Ethernet up to advanced high-speed FDDI or ATM networks. COMCOs that provide on-chip storage for entire packets, as is the case for most CAN controllers, cannot be used unless clock synchronization is explicitly supported by exporting the required trigger signals, cf. [4].

The current version of the NTI has been developed for Motorola's MVME-162 board (M68040 CPU + Intel's 82596CA Ethernet coprocessor) in conjunction with a passive VME carrier-board hosting up to four NTI M-Modules. A simple test application has been written for this system, which allowed us to assess the resulting transmission/reception time uncertainty ε. It turned out that the actual ε of the 82596CA in conjunction with our NTI is well below 200 ns.

2.4 Hardware/Software Interface

All accesses to UTCSU registers and NTI memory are performed by addressing the M-Modules memory-space. As explained in Section 2.3, read/writes of the COMCO require additional logic to provide timestamping functionalities. To distinguish between CPU and COMCO accesses, the CPLD maps two address regions to the same physical memory as illustrated in Figure 5.

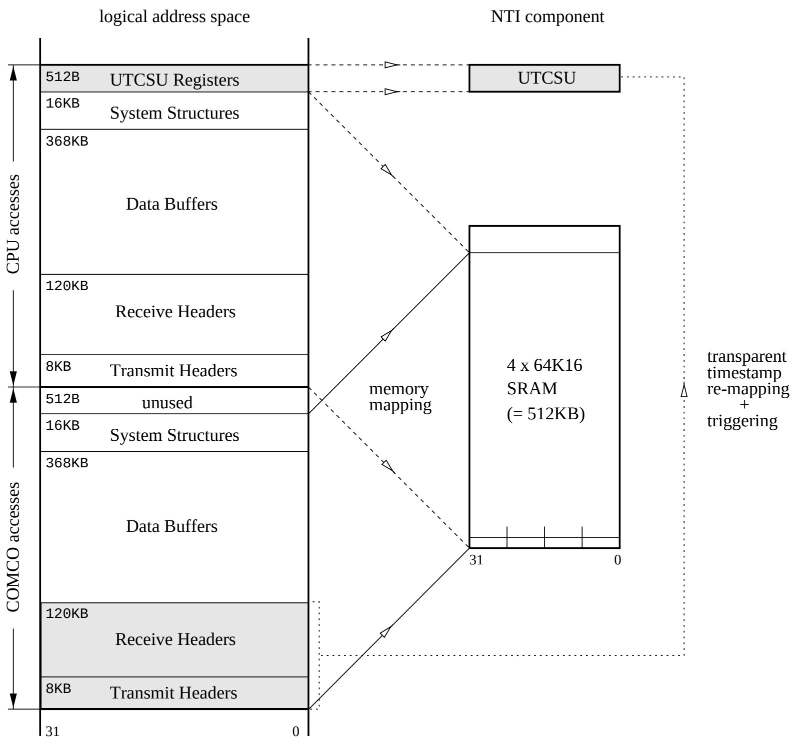


Fig. 5. Memory map of the NTI

On top of the memory map is the NTI memory's 512 KB address region for CPU-accesses, which is decoded without special functionality, and ends with a 512 byte segment containing the UTCSU registers. The 512 KB region for COMCO-accesses to NTI memory starts at address 0 and is divided into four sections: The *System Structures* section holds the command interface and system data structures required by the COMCO, the *Data Buffers* are available for ordinary packet data. Special functionalities in regard of timestamp triggering and remapping apply only to accesses in the *Receive Headers* resp. *Transmit Headers* sections, which hold packet-specific control and routing information (e.g. source and destination addresses) for received resp. transmitted CSPs.

In addition to the UTCSU registers and the shared memory, there are also a few registers provided by the NTI itself, primarily for the purpose of correctly assigning receive timestamps to data packets: After the UTCSU has sampled a receive time/accuracystamp, it must be moved to an unused portion of the appropriate CSP before the next one drops in. This could be simply done in an *Interrupt Service Routine* (ISR) activated on packet reception. Unfortunately, the ISR cannot reliably determine the address of the receive header associated with the sampled timestamp, thus the NTI latches this address into the *Receive Header Base* register upon the occurrence of the trigger signal `RECEIVE`. There are of course alternatives, which, however, do not work in general. For example, one might try to move the timestamp in the packet reception ISR, where the base address of the receive buffer is of course known, but in case of back-to-back CSPs this might be too late for avoiding timestamp loss. Also inappropriate are schemes that try to exploit a sequential order of received packets, since there might be CSPs that trigger a timestamp but get eventually discarded due to an incorrect CRC.

The entire NTI software is embedded in an "add-on" of the industrial multiprocessing/multitasking real-time kernel pSOS^{+m} (Integrated Systems, Inc.). At the heart of our implementation is a driver that actually multiplexes three different interfaces to the 82596CA Ethernet coprocessor: The *Kernel Interface* (KI) enables multiprocessing by providing remote objects (tasks, queues, semaphores, etc.), the *Network Interface* (NI) provides TCP/IP sockets if the additional software component pNA^{+} is present, and the *Clock Interface* (CI) finally handles timestamped CSPs. In fact, apart from the created computing and networking load, clock synchronization is performed totally transparent to the application.

3 Conclusions and Future Research

In this paper, we surveyed our *Network Time Interface* (NTI) dedicated to support interval-based external clock synchronization by hardware. Implemented as an M-Module around our novel UTCSU-ASIC, the NTI provides a cheap way of extending state-of-the-art real-time systems technology with fault-tolerant synchronized clocks with a worst case precision/accuracy in the 1 μs-range. Apart from the principal advantages of interval-based clock synchronization, our approach thus allows an improvement of at least one order of magnitude over existing ones. The NTI can be used in conjunction with any network controller with DMA-capabilities and is easily incorporated in usual real-time operating system kernels.

We are still working on a few algorithmic/theoretical issues concerning external clock synchronization and a thorough experimental evaluation including a transition to another target hardware (AcQ's i6040). In addition, we are currently negotiating a pilot industrial application in the area of on-line fault location for underground power cables. Although this is by now the only application we are aware of that really requires our high precision/accuracy, we are nevertheless convinced that others will eventually emerge when enabling technology like our NTI is available.

Acknowledgements

We would like to thank Nikolaus Kerö, Herbert Nachtnebel, and Thomas Mandl for building and testing our NTI M-Modules.

References

1. P.H. Dana. *Global Positioning System (GPS) Time Dissemination for Real-Time Applications*, J. of Real-Time Systems 12(1), p. 9–40, January 1997.
2. M. Horauer, U. Schmid, K. Schossmaier. *NTI: A Network Time Interface M-Module for High-Accuracy Clock Synchronization*, TR 183/1-76, Department of Automation, Vienna University of Technology, January 1997.
3. H. Kopetz, W. Ochsenreiter. *Clock Synchronization in Distributed Real-Time Systems*, IEEE Transactions on Computers C-36(8), p. 933–939, August 1987.
4. H. Kopetz, A. Krüger, D. Millinger, A. Schedl. *A Synchronization Strategy for a Time-Triggered Multicluster Real-Time System*, Proc. of the 14th IEEE Symposium on Reliable Distributed Systems, Bad Neuenahr, Germany, 13–15 September 1995.
5. B. Liskov. *Practical uses of synchronized clocks in distributed systems*, Distributed Computing 6(4), p. 211–219, 1993.
6. J. Lundelius, N. Lynch. *An Upper and Lower Bound for Clock Synchronization*, Information and Control 62(2/3), p. 190–204, June 1984.
7. Manufacturers and Users of M-Modules (MUMM) e.V., *M-Module Specification*, April 1996.
8. D.L. Mills. *Internet Time Synchronization: The Network Time Protocol*, IEEE Transactions on Communications 39(10), p. 1482–1493, October 1991.
9. U. Schmid, K. Schossmaier. *Interval-based Clock Synchronization*, J. of Real-Time Systems 12(2), p. 173–228, March 1997.
10. K. Schossmaier, U. Schmid, M. Horauer, D. Loy. *Specification and Implementation of the Universal Time Coordinated Synchronization Unit (UTCSU)*, J. of Real-Time Systems 12(3), p. 295–327, May 1997.
11. B. Simons, J. Lundelius-Welch, N. Lynch. *An Overview of Clock Synchronization*, in B. Simons, A. Spector (eds.): Fault-Tolerant Distributed Computing, Springer LNCS 448, p. 84–96, 1990.

This article was processed using the LaTeX macro package with LLNCS style