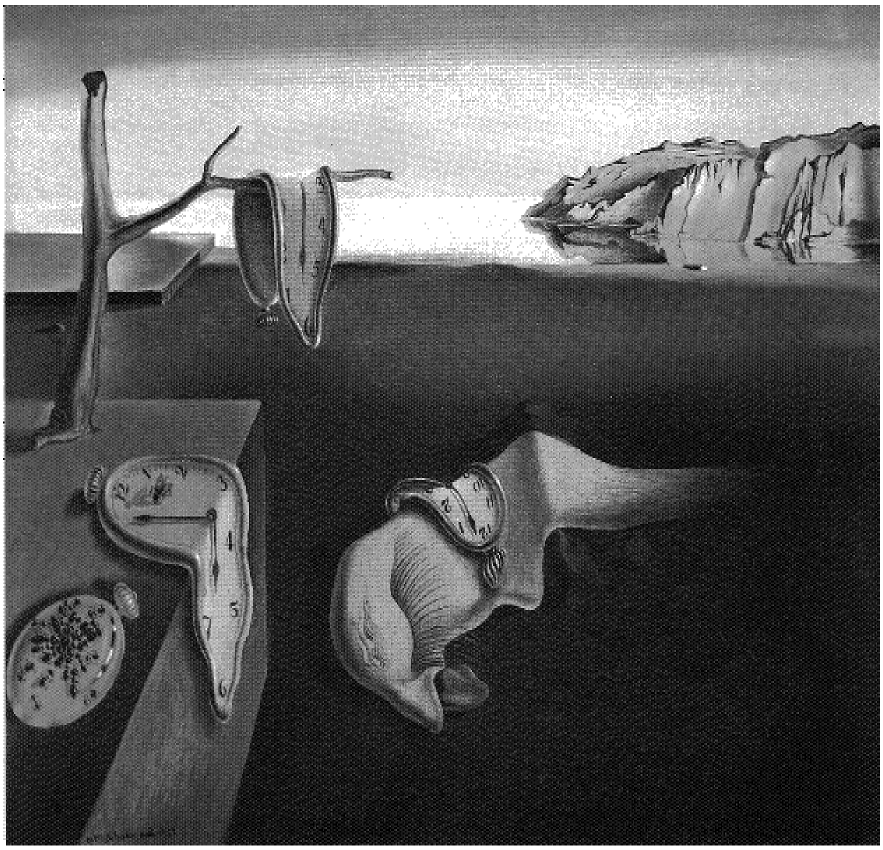
Institut für Automation
Abt. für Automatisierungssysteme

Technische Universität Wien

Projektbericht Nr. 183/1-69
December 1996

NTI Functional and Architectural Specification

Martin Horauer, Dietmar Loy, Ulrich Schmid



Salvador Dali, "Die Beständigkeit der Erinnerung"

NTI Functional and Architectural Specification

Martin Horauer, Dietmar Loy
Technical University of Vienna
Department of Computer Technology
Gußhausstraße 27, A-1040 Vienna
Email: {horauer, loy}@ict.tuwien.ac.at

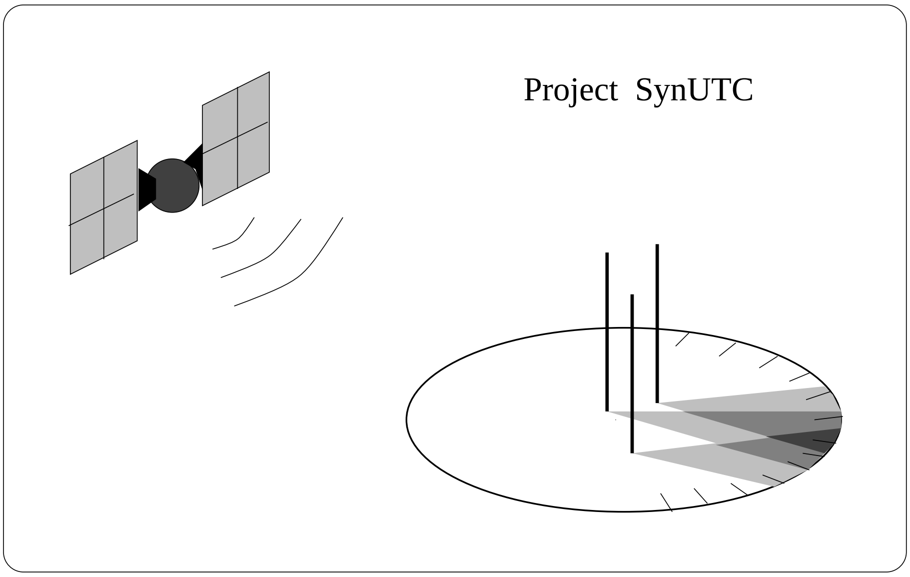
Ulrich Schmid
Technical University of Vienna
Department of Automation
Treitlstraße 1, A-1040 Vienna, Austria
Email: s@auto.tuwien.ac.at

October 13, 1997

Abstract

This paper[1] presents the specification and prototype implementation of a printed circuit board termed *Network Time Interface* (NTI). Based on the M-Module industry standard, the NTI provides a turn-key solution for adding high-resolution synchronized clocks to fault-tolerant distributed systems built upon off-the-shelf hardware. It is built around our UTCSU-ASIC, which contains most of the hardware support required for external clock synchronization in distributed systems, like a rate- and state-adjustable local clock, automatically maintained accuracy intervals, and provisions for connecting GPS receivers. Apart from interfacing the UTCSU to the rest of the system, the primary purpose of the NTI is to support accurate timestamping of clock synchronization data packets transmitted/received by network controllers with DMA-capabilities.

Keywords: M-Module, external clock synchronization, hardware support, Global Positioning System (GPS), ASIC, VHDL.



[1]This research is part of our project SynUTC supported by the Austrian Science Foundation (FWF) under grant P10244-ÖMA.

Contents

1 Introduction

Real-time systems need to interact both correctly and timely with their environment. For tasks starting at time r and having their strict dead-line at d the probability of correct task execution is stated as $P(r + \Delta e \leq d|B) = 1$ with Δe denoting the required execution time under the conditions B, which cover all possible faults, see [Zoe95]. It seems obvious that predictable operation of a real-time system requires a system design that incorporates a close relation to time. The need for this requirement aggravates when the system becomes distributed, in the sense that tasks are processed concurrently on seperate nodes. A mechanism to synchronize the clocks of such a system must be enforced in order to guarantee the correct and timely interaction of such processes. In general such mechanism come in two flavors namely the *internal* and *external clock synchronization.*

The purpose of internal clock synchronization is to keep the clocks within well-defined bounds of each other by disseminating and processing time information. If $C_p(t)$ and $C_q(t)$ denote any two clocks, they are said to have precision π when $|C_p(t) - C_q(t)| \leq \pi \quad \forall t \geq t_0$.

Most internal synchronization algorithms are purely software-based and provide a precision in the 10 ms-range only. Considerably better results can be achieved with dedicated hardware support. Clocks synchronized in the 10 μs-range can be built on top of the pioneering *Clock Synchronization Unit* (CSU) described in [KO87]; about 100 μs are achieved by the hardware assisted clock synchronization scheme of [RKS90]. Even much smaller precisions are attained by phase-locked-loops incorporating clock voting, cf. [RSB90]. This approach, however, is not directly comparable to the other ones since it requires a dedicated clocking network.

If system time provided by synchronized clocks must also have a well-defined relation to real-time t (e.g. *Universal Time Coordinated* (UTC), the only official and legal standard time, or GPS-time), then the so-called external synchronization problem needs to be solved. One has to ensure here that there is some well-defined and small accuracy $\alpha_p(t)$ sucht that any local clock $C_p(t)$ satisfies $|C_p(t) - t| \leq \alpha_p(t)$ for all $t \geq t_0$.

The most widely used external clock synchronization scheme is doubtlessly the *Network Time Protocol* (NTP) designed for disseminating UTC among workstations throughout the Internet, see [Mil91] for an overview. Under realistic conditions, worst case accuracies of approximately 20 ms were observed by [Tro94].

Of course, NTP was designed for partially asynchronous systems, exhibiting potentially unbounded communication delays. More can be done in fully synchronous systems, see [Sch97a] for a quite comprehensive collection of recent papers on this topic. For instance, there is a software-based solution [VRC97] that "sprays" external time obtained via GPS into broadcast-type LANs, achieving a precision/accuracy in the 10 μs-range.

This paper is devoted to a detailed description of the hardware support for *interval-based clock validation* introduced in [Sch95]. Implemented as a *Network Time Interface* M-Module (NTI), our solution allows to extend state-of-the-art hardware technology with fault-tolerant synchronized clocks providing a precision/accuracy in the 1 μs-range. This is accomplished by integrating our custom *Universal Time-coordinated Clock Synchronization Unit* (UTCSU-ASIC) with memory and special-purpose decoding logic in a way that facilitates accurate transmission and reception timestamping for DMA-type network controllers.

The remaining sections of our paper are organized as follows: The rest of Section 1 is devoted to the description of the general node architecture and an assessment of the uncertainties incorporated in time dissemination. Section 2 sheds some light on the interior of the UTCSU, providing the necessary knowledge for integration on the NTI. The detailed NTI requirements along with the resulting functional/architectural issues are provided in Section 3. Section 4 elaborates on implementation issues, which are complemented by an appendix containing timing diagrams, schematics, and VHDL source code of our NTI prototype. In Section 5, a VHDL-based simulation model of the NTI is briefly outlined, which greatly simplifies board development and testing. Finally, a comprehensive list of further enhancements is appended in Section 7.

1.1 Node Architecture

A distributed real-time system consists of a set of nodes which are interconnected by a suitable network. The nodes are self-contained computers, which run applications that need to interact both correctly and timely with the environment. Actions and observations at different nodes require a time service to coordinate them appropriately. The distributed nature aggravates the installation of a time service considerably, because autonomous running clocks have a tendency to drift apart or might fail grossly. Hence, clocks need to be synchronized by virtue of a suitable algorithm executed on each node. Figure 1 identifies the major components of a system with two generic nodes.

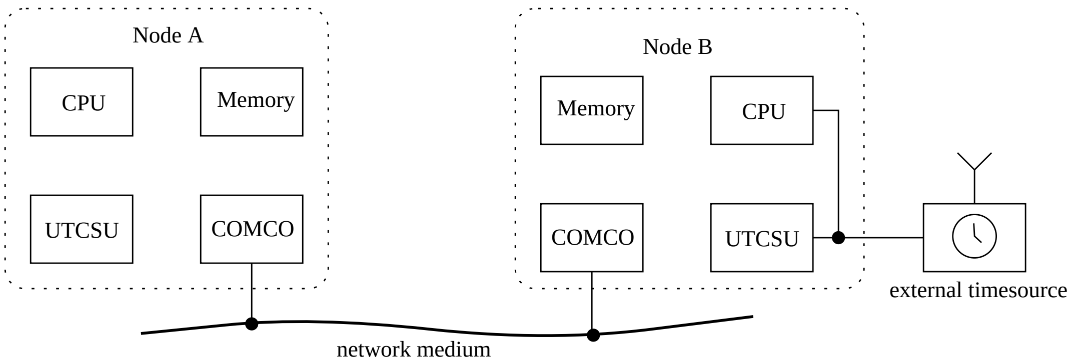


Figure 1: *Two nodes of a distributed system*

Each node contains a local clock, that is, our *UTCSU*-ASIC (see Section 2), a *Communication Coprocessor* (COMCO) providing access to the network, a CPU responsible for executing the software-part of the clock synchronization algorithm, and some kind of memory to store receive/transmit data packets. The (general purpose) CPU can be the node's central processor or, preferably, a dedicated microprocessor or microcontroller. Similarily, we do not impose a particular network type and/or COMCO, except that the latter must be able to send/receive data packets indepently of CPU operation. More specifically, we assume that the COMCO fetches/stores packet data (e.g. via DMA) from/in the memory when it transmits/receives a *Clock Synchronization Packet* (CSP). We will return to the problem of deciding whether a particular COMCO is suitable for one approach in Section 1.4.

1.2 Basic Operation Principle

A major concern in highly accurate/precise clock synchronization is exact timestamping of CSPs at both sending and receiving side. In fact, the well known result of [LL84], saying that even n ideal clocks cannot be synchronized with a precision less than $\epsilon\,(1-1/n)$ in case of packet transmission time uncertainty ϵ, justifies this requirement. However, packet transmission/reception involves several steps that potentially contribute to ϵ. For COMCOs with DMA-capabilities, those steps are as follows, cf. [KO87]:

1. Sender-CPU assembles the CSP
2. Sender-CPU aquires the memory to write the CSP
3. Sender-CPU signals sender-COMCO to take over for transmission
4. Sender-COMCO tries to aquire the channel
5. Sender-COMCO arbitrates the memory, reads the CSP data and pushes the corresponding bit stream onto the channel
6. Receiver-COMCO pulls bit stream from the channel
7. Receiver-COMCO arbitrates the memory and writes the CSP
8. Receiver-COMCO notifies the receiver-CPU of packet reception via an interrupt
9. Receiver-CPU acquires the memory and processes the CSP

Software-based clock synchronization performs CSP timestamping upon transmission resp. reception in steps 1 resp. 9. Thus, ϵ incorporates the latency to arbitrate memory (2, 5, 7 and 9), the channel access uncertainty $4 \rightarrow 5$, any variable network delay $5 \rightarrow 6$, and the reception interrupt latency $8 \rightarrow 9$. The dominating contributions are $4 \rightarrow 5$, which can be quite large for any network utilizing a shared channel, and $8 \rightarrow 9$, which is seriously impaired by code segments with interrupts disabled. Fortunately, in our LAN-based setting, we can safely neglect the contribution from $5 \rightarrow 6$ since there are no (load- and hop-dependent) queueing delays from intermediate gateway nodes.[2]

To reduce the variance of ϵ, clock synchronization hardware must be placed as close to the network as possible. Ideally, a CSP should be timestamped at the sender resp. receiver side when, say, its first byte is pushed on resp. pulled from the channel. However, this requires support from the interior of the network controller, which is usually not available.

In order to make our approach compatible with existing network technology, we insert a timestamp on-the-fly into a CSP in a way that minimizes the uncertainty of the transmission/reception latency instead. This is accomplished by employing a refinement of the DMA-based coupling method proposed in [KO87], which is based on a modified address decoding logic for the memory that

- generates a trigger signal for sampling a timestamp into a dedicated UTCSU holding-register when a certain byte within the transmit/receive buffer for a CSP is read/written, and
- transparently maps the appropriate UTCSU holding-register into some portion of the transmit/receive buffer.

This way, CSP timestamping occurs in step 5 resp. 7 of the data transmission/reception sequence introduced above. Note that this special functionality is only present when a transmit/receive buffer is accessed by the COMCO, whereas CPU-accesses act as plain memory accesses.

[2]Note that this statement would remain true if all intermediate nodes in a multi-hop network were also equipped with the NTI's packet timestamping mechanism described below.

1.3 Packet Timestamping Example

To illustrate the resulting process of packet timestamping, we briefly discuss one possible scenario depicted in Figure 2; alternative scenarios may be found in [SS95].

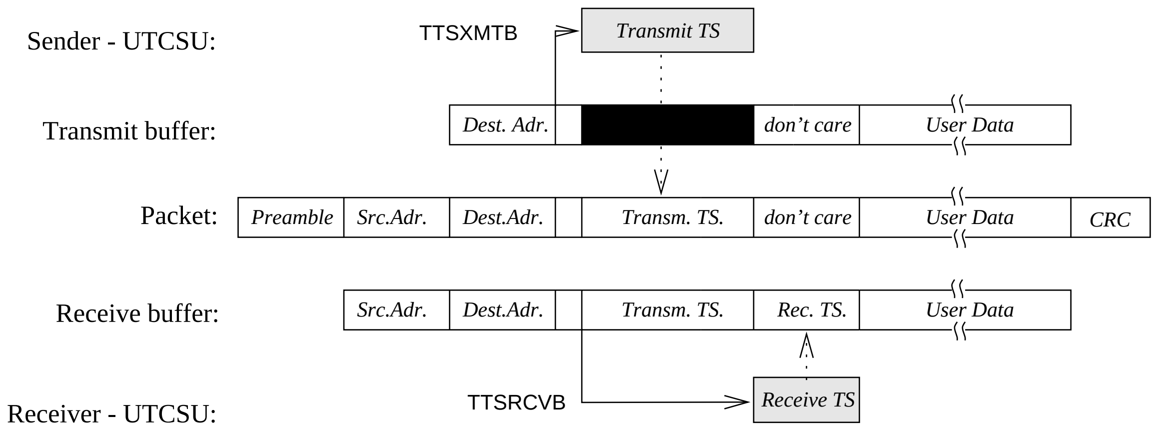


Figure 2: *Packet timestamping*

In order to send a CSP, the CPU writes the packet into the memory and signals the COMCO to take over for transmission. The COMCO in turn sets up communication and eventually fetches CSP data from memory, performing the required parallel to serial conversion and pushing the bit stream onto the channel. The CPU, which executes concurrently with the COMCO, is notified when the CSP has been sent. What really matters here is that whenever the COMCO fetches data from the transmit buffer, it has to read across the destination address causing the decoding logic to generate the trigger signal TRANSMIT. Upon occurrence of this signal, the UTCSU samples a *transmit timestamp* into an internal register, which is transparently mapped into a certain portion of the transmit buffer and hence inserted into the outgoing packet. Note that trigger and mapping address may be different.

By the same token, when the COMCO at the receiving end writes a certain portion of the receive buffer in memory, the trigger signal RECEIVE is generated by the decoding logic, which causes the UTCSU to sample the *receive timestamp* into a dedicated register. It can be saved by the CPU upon reception notification or even immediately in an unused portion of the receive buffer, see [SS95] for the appropriate details.

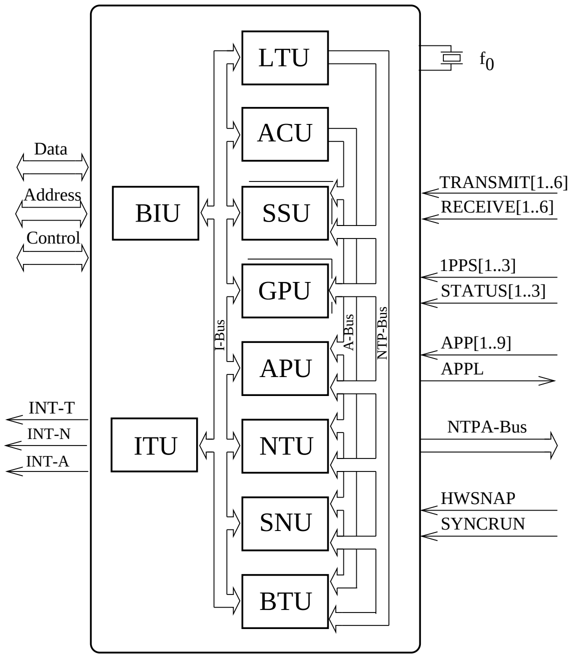
1.4 Suitable Communication Coprocessors

DMA-type COMCOs are available for a wide variety of networks, ranging from fieldbusses like Profibus over Ethernet-based networks up to advanced high-speed FDDI or ATM networks. Excluded from being used in conjunction with our NTI are COMCOs providing on-chip storage for packets, as is the case for most CAN controllers. Integrating our clock synchronization hardware with this type of controllers is only possible if appropriate transmit and receive timestamping signals are generated by the COMCO.

However, we should mention that assessing transmission/reception delay uncertainty ε of a particular COMCO usually means experimental evaluation. This is due to the fact that most controllers utilize internal FIFOs, which introduce uncertainties in the time between fetching a byte from memory and putting it on the channel. Whereas adjusting the triggering position of transmit/receive timestamp might help in reducing/circumventing such impairments, it is nevertheless not easy to find and justify a suitable choice without actual measurements. Note again that such problems would vanish entirely if additional information from the COMCO is available, i.e. an explicit signal marking actual transmission/reception of a byte of the CSP.

2 UTCSU

In this subsection, we provide a succinct overview of the wealth of functionalities of our custom *Universal Time Coordinated Synchronization Unit* (UTCSU); further information is available in [SSHL97], [SL96], [SS95] and especially in [Loy96]. 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 180-pin PGA case. 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

Figure 3: *Interior of the UTCSU*

2.1 Local Clock

The centerpiece of our chip is a local clock unit (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, cf. [Mil91]. 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. Of course, a proper augend value (in multiples of $2^{-51}\ s \approx 0.44\ fs$) must be provided by the atop running clock rate synchronization algorithm to enforce the desired rate adjustment.

To support interval-based clock synchronization, our 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 $\alpha^+(t)$ in a way that sustains inclusion of external time t, i.e., $t \in [C(t) - \alpha^-(t), C(t) + \alpha^+(t)]$. Both accuracies can be (re)initialized atomically in conjunction with the clock register in the LTU. In addition, some extra logic suppresses a wrap-around of α^- and α^+ and zero-masks potentially negative accuracies during continuous amortization.

2.2 Timestamping-Support

A number of external events, supplied to the UTCSU via polarity programmable input lines, can be *time/accuracy-stamped* with local time and accuracy (i.e., α^- and α^+), which are atomically sampled into dedicated registers upon the appropriate input transition. Optionally, an interrupt can be raised on such an event as well. Due to the asynchronous nature of these inputs, internal synchronizer stages are utilized that introduce a timing uncertainty of at most $1/f_{osc}$. Note that a one- or two-stage synchronizer is employed, depending on the status of the UTCSUs RELIABLE-pin; the recovery time for metastability phenomenons is therefore $1/f_{osc}$ or $2/f_{osc}$, resulting in a reasonably small probability of failure, see [Loy96] for more information.

Three different functional blocks in the UTCSU utilize time/accuracy-stamping:

1. Trigger signals generated by the decoding logic at CSP transmission/reception (as explained in Section 1.3) sample the current local time/accuracy into dedicated UTCSU registers in the SSU. Six independent SSUs are provided to facilitate fault-tolerant (redundant) communications architectures or gateway nodes.

2. 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. An optional status line provided by some high-end models is also sampled upon each occurrence of the 1 pps pulse. Note that this simple interfacing 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.

3. 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.

2.3 Duty Timers

The above timestamping features are complemented by several 48 bit programmable *duty timers* accommodated in several functional blocks of the UTCSU. Whenever an armed duty timer goes off (due to the fact that local time reaches the programmed one), an interrupt is raised. Duty timers are used to execute the protocol for CSP exchange governed by the clock synchronization algorithm, to control continuous amortization, to insert/delete leap seconds, and to generate application-related events.

2.4 Selftest Support

Last but not least, the UTCSU is equipped with many features for test (BTU) and debugging purposes (SNU). The BTU handles calculation of checksums, blocksums and signatures (via two different multiple input linear feedback registers) for local time. The SNU hosts three features for debugging and test purposes: A hardware snapshot, controlled by the HWSNAP-pin, is used to retrieve coarse information of the chip by sampling local time and accuracy information into dedicated registers. More detailed information is provided by a software snapshot, which samples all dynamically changing registers. Finally, there is a SYNCRUN-pin facilitating testing as well as synchronous operation of redundant UTCSUs. Note that such provisions are required for building up fault-tolerant applications based on self-checking and/or redundant units.

2.5 System Interfacing

The *Bus Interface Unit* (BIU) contains logic for embedding the UTCSU in a wide variety of system architectures. Facilitating dynamic bus sizing, little/big endian byte ordering, and different access times, the UTCSU can be used in conjunction with virtually any 8, 16 and 32 bit CPU. Note that the UTCSU synchronizes its chip select signal (and all other asynchronous inputs, e.g. all timestamp inputs) by the already mentioned synchronizer stage(s).

However, some care must be exercised here due to a design limitiation/fault: We incorrectly assumed that 8 bit-accesses resp. 16 bit ones are always using D32–D24 resp. D32–D16, irrespectively of the port size and the address alignment. Thus, additional transceivers are usually required for 32 bit port size.

The *Interrupt Unit* (ITU) finally collects the numerous interrupt sources inside the UTCSU and assigns them statically to three dedicated interrupt pins, viz. INTN (network related), INTT (timer related) and INTA (application related). Moreover the ITU allows enabling and disabling of each interrupt source and provides additional status information.

3 NTI Requirements and Specification

Biased by hardware and software know-how gathered during former projects, see [Sch94], we decided to use a VMEbus-based prototype system for experimental evaluation in our SynUTC-project. Fortunately, our basic assumption about CPU and COMCO independence (recall Section 1.2) allows us to avoid the development of a fully-fledged node hardware. Instead, our NTI aims at extending existing CPU boards with adequate support. It became apparent soon that designing a suitable mezzanine bus module should provide an appealing alternative to the costly development of a VME board.

3.1 VME mezzanines

There are several mezzanine standards available on the market, e.g. *Industry Pack* (IP), *M-Modules*, *FLXibus modules*, and others. As argued in most comparisons like [Bau94], however, there is no single "ideal" instance for all applications. Consequently, one has to judge the flexibility, performance, size, robustness, cost etc. provided by a certain mezzanine bus against the requirements of the the particular application to find the best choice. Table 1 summarizes the major features of the mezzanines considered for the NTI.

Module	IP-Module	M(A)-Module	FLXibus Module
mode of operation	synchronous	asynchronous	asynchronous
data interface	16 bit	16/32 bit	32 bit
address space	128 byte I/O 8 Mbyte memory (multiplexed)	256 byte I/O 16 Mbyte memory (multiplexed)	32 address lines non-multiplexed
DMA capability	2 DMA channels	1 DMA channel	4 DMA channels
Interrupt	2 Interrupts with Vector	1 Interrupt with Vector	8 Interrupts with Vector
Max. No. of Modules/Carrier	4	4	2
Front Panel Connector	No	Yes	Yes
Physical Dim.	100 x 47 mm	150 x 47 mm	150 x 82 mm

Table 1: *IP, M- and FLXibus Modules Overview*

The most important general requirements put on the development of the NTI are as follows:

- *High performance*: Since both CPU and COMCO access the memory on board the NTI, fast accesses are mandatory; this means both high speed and 32 bit bus width.
- *Simplicity*: We do not have enough experience to bother ourselves with the development of a complicated bus interface.
- *Size*: The module should be as small as possible but capable of accomodating the 180-pin PGA of the UTCSU and the surrounding logic.
- *Carrier boards*: There should be carrier boards capable of hosting several NTIs to reduce the costs of the overall evaluation system.
- *Connectors*: The required interface to GPS-receivers and applications make a direct front panel connector desireable.

3.2 General Outline

Comparing our general requirements with the features listed in Table 1, the benefits[3] of M-Modules over the other candidates are apparent. We thus decided to design the NTI as an *MA-Module*, that is, as an M-Module with a 32 bit data bus, which is characterized as follows [MM96]: The address space consists of a 256 byte *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. The M-Module interface also provides a single vectorized interrupt line and two additional DMA control lines. The unit construction design of the 147 x 53 mm (single-height) M-Modules provides a peripheral 25-pin D-sub connector on the front panel, a 24-pin resp. 60-pin plug connector to the carrier board for peripheral I/O resp. MA-interface, and two optional intermodule port plug connectors for interconnecting several M-Modules.

Figure 4 shows the major components of the NTI M-Module along with its interfaces, the requirements of which are detailed in Section 3.3.

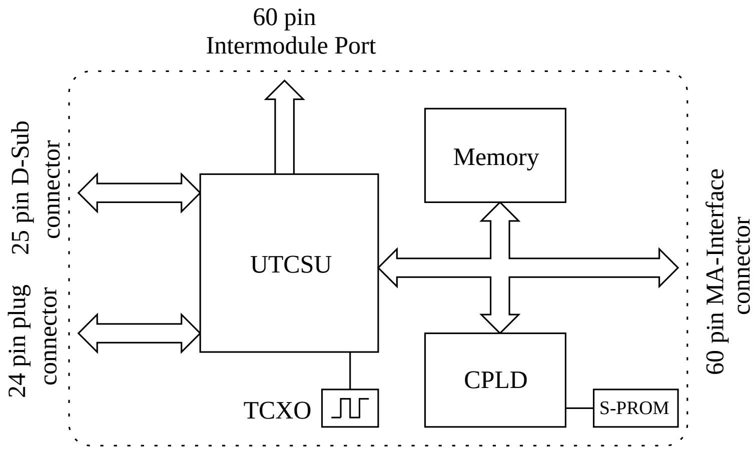


Figure 4: *NTI block diagram*

The pivotal UTCSU-ASIC performs most of the relevant operations for clock synchronization, recall Section 2. It is paced by a temperature compensated (TCXO) or ovenized (OCXO) quartz oscillator, or, alternatively, by an external frequency source like the 10 MHz output of a high-end GPS receiver.

The memory serves as control and data interface between the CPU and the COMCO, namely, Intel's i82596CA Ethernet coprocessor, providing the special functionality for COMCO accesses as outlined in Section 1.2. Note that it must support byte, word, and longword read/write accesses, since it might cause problems with existing (operating system) software if CPU-accesses to packet memory are constrained.

Most of the required glue logic of the NTI is incorporated in a single, in-circuit programmable *Complex Programmable Logic Device* (CPLD), which adapts the signals of the MA-Module interface, handles interrupt requests and gives access to the serial PROM storing module identification and revision information.

[3]The only serious restriction is the single interrupt line, which causes some problems when mapping the three UTCSU-interrupt outputs onto it.

3.3 Interfacing Considerations

Limited pin-count and size of the (single-sized) M-Modules as well as the following requirements dictate the signal grouping and connector layout to be used for the NTI interfaces.

- All UTCSU-signals that are likely to be used externally should be buffered to protect the UTCSU from damage. In addition, pull-up or pull-down resistors must be provided for all possibly open input lines to avoid floating.

- All UTCSU-signals of the APU must be accessible via the front panel connector and should be opto-decoupled or at least buffered by socketed DIP devices for easy replacement.

- All UTCSU-signals of the three GPUs must be provided at both the front panel connector and the intermodule port connector for easy attachment of both external and modularized GPS receivers. All signals of at least one GPU must be opto-decoupled[4] via very high-speed, socketed devices to avoid ground loops in case of an external GPS-receiver.

- All the UTCSU's timestamping inputs of the six SSUs must be provided at the I/O plug connector to the carrier board to facilitate redundant architectures. This also includes signals TRANSMIT (TSXMT1) and RECEIVE (TSRCV1) of SSU1, which are usually generated by the on-board decoding logic of the NTI; a pair of jumpers must be provided to select internal or external supply of those signals for building up exotic architectures.

- All the UTCSU's timestamping inputs (of any unit) as well as the special-purpose signal SYNCRUN and the APPDUTY-output of the APU must be available at the I/O plug connector to the carrier board. This requirement is set forth by our evaluation hardware (consisting of 16 or perhaps 20 NTIs), which assumes that the APPDUTY-output of any NTI is internally wired (by a flat cable) to a dedicated timestamp input at all other NTIs to facilitate measurement of the transmission delay uncertainty. Note that APPDUTY must be driven by the CPLD-generated transmit timestamp signal (TSXMT1) here instead of the UTCSU's APPDUTY-signal (this must be jumper-selectable).

- As far as the actual pinning of the connectors is concerned, it should not cause harm if a connector is erroneously plugged in wrongly. This is particularily true for the front panel connector, which should be designed symmetrically w.r.t. input/outputs, so that even plugging in an (erroneously) mirrored ($1 \leftrightarrow 25$ etc.) connector cannot cause damage. For example, if pin 1 is an output, then pin 13 should be an output as well.

  Note also that the limited size of an M-Module severely constrains the area where opto-couplers can be placed, which in turn determines part of the front panel connector layout.

3.4 Memory Address Space

All accesses to UTCSU registers and NTI memory are performed by addressing the M-Modules memory-space. As explained in Section 1.2, 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.

On top of the memory map is the 512 byte segment containing the UTCSU registers preceeded by the NTI memory's 256 KB address region for CPU-accesses, which are both decoded without special functionality. The 256 KB region for COMCO-accesses to NTI memory starts

[4] Line driver inputs with high common mode input voltage might also be considered here.

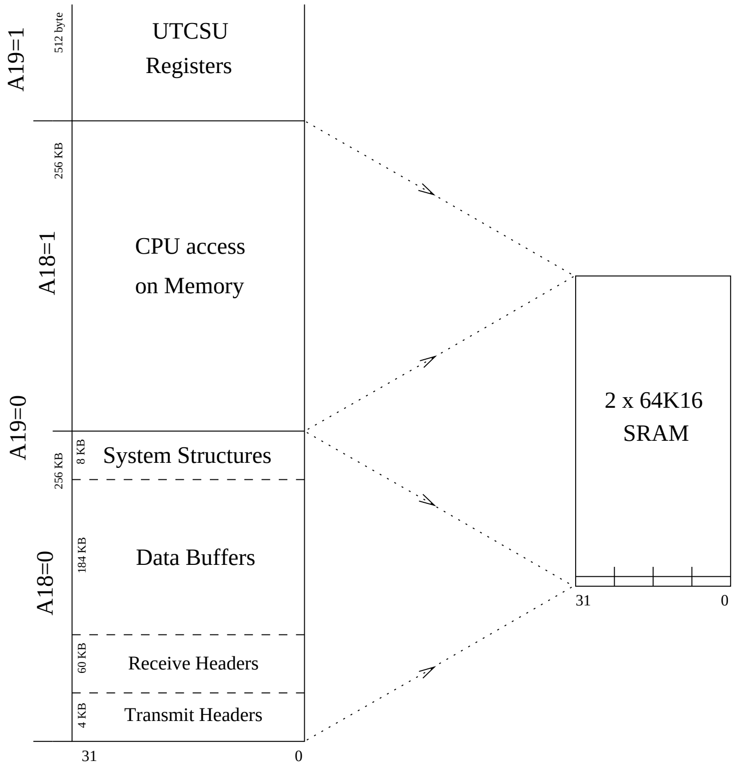


Figure 5: *Memory map of the NTI memory space*

at address 0 and is divided into four different sections: The *System Structures* section holds the command interface and system data structures usually required by the COMCO, the *Data Buffers* are available for ordinary packet data. Special functionalities 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.

The version of the NTI described in this paper supports Intel's 82596CA Ethernet coprocessor using 64 byte receive and transmit headers. Figure 6 outlines the offsets within each header that need to be supervised here.

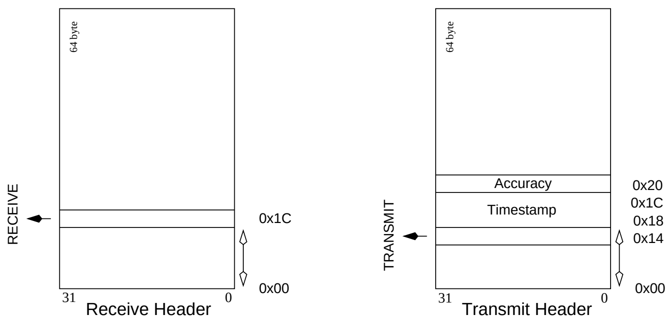


Figure 6: *Receive and Transmit Header*

When the 82596CA writes offset 0x1C within a receive header upon reception of a CSP, the timestamp trigger signal RECEIVE for the UTCSU is generated. In addition, the base address of the accessed receive header is stored into a dedicated NTI-register to facilitate further interrupt processing, as explained below. Similarly, when the 82596CA reads offset 0x14 within a transmit header upon transmission of a CSP, a timestamp trigger signal TRANSMIT is issued to the UTCSU. Since the UTCSU registers holding the sampled time/accuracystamp are mapped to the memory addresses with offset 0x18 and 0x20 in the transmit header, they are transparently inserted into the outgoing data packet.

3.5 I/O Address Space

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 interrupt handling. They are accessible via the M-Modules I/O-space according to the memory map in Figure 7.

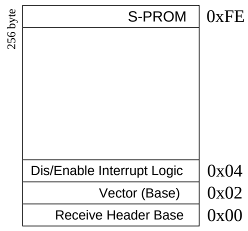


Figure 7: *Memory map of the I/O-space of the NTI*

The *Receive Header Base* register at address 0x00 is required for 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 CSP drops in. This could be done in an ISR activated by a RECEIVE transition interrupt, which, however, cannot reliably[5] determine the address of the receive header associated with the sampled timestamp. Therefore, the NTI latches this address into the *Receive Header Base* register upon the occurrence of the RECEIVE-signal.

Two additional NTI registers control interrupt generation (which is explained in some detail in Section 3.6): The *Vector (Base)* register at address 0x02 can be used to program the interrupt vector generated upon an UTCSU interrupt. Note that the final vector also includes the state of the three UTCSU interrupt pins INTT, INTN, and INTA. Accessing register *Dis/Enable Interrupt Logic* at address 0x04 eventually enables (further) NTI interrupts; it is usually written once after a module reset and regularily prior to leaving the interrupt service routine.

The unused I/O address space between address 0x06 and the access-byte to the M-Module's Serial PROM at 0xFE can be used for adding special functionality, see Section 5 for an example.

[5]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. Unfortunately, this might be too late for avoiding a timestamp loss in case of back-to-back CSPs. Also inappropriate are schemes that try to exploit a sequential order of received packets, since there might be CSPs that trigger a timestamp but are eventually discarded, e.g., due to an incorrect CRC.

3.6 Interrupt Handling Example

As noted in Section 2, the numerous internal interrupt sources of the UTCSU-ASIC are hard-wired to three different interrupt outputs INTN (network-related), INTT (timer-related) and INTA (application-related). M-Modules provide only a single interrupt line IRQB that, however, can be used for signaling a vectorized interrupt with its vector provided by the module. On the NTI, the three least significant bits $D2, D1, D0$ of the 8 bit interrupt vector reflect the current status of INTN, INTT, INTA, and the remaining 5 bits are taken from the programmable *Vector Base*. A possible scenario of an interrupt interaction is depicted in Figure 8.

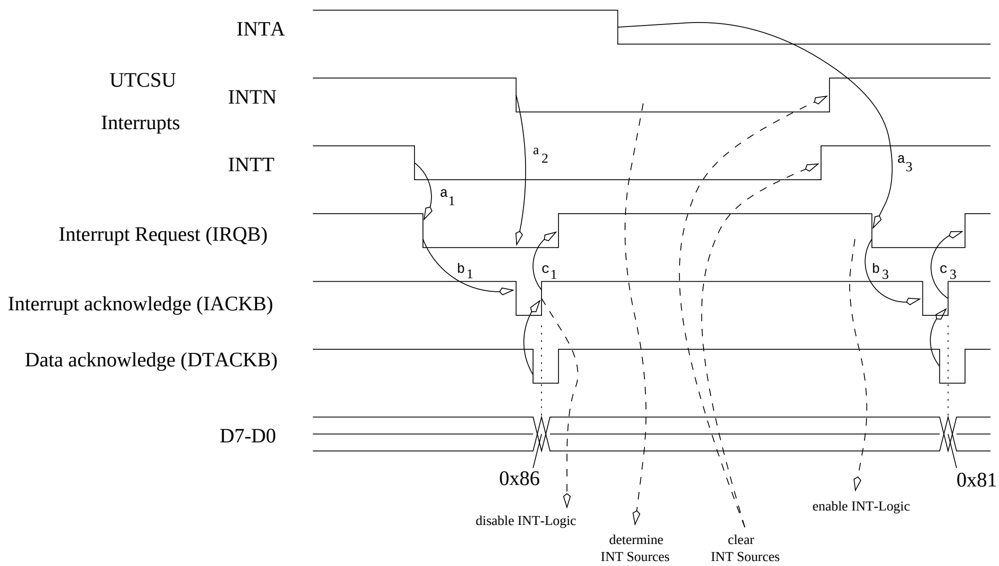


Figure 8: *Interrupt service example*

Assuming that the vector base has been programmed to 0x80, an UTCSU-interrupt on INTT (a_1) causes the current (internal) interrupt vector to be set to 0x82 and forces the M-Modules IRQB to go active (low). Note that another UTCSU-interrupt INTN (a_2) occurs before the vector is provided to the CPU in the interrupt acknowledge cycle. The CPU accepts the M-Modules interrupt request by driving IACKB active (b_1), causing the NTI to put the current interrupt vector 0x86 onto the data lines D0–D7 and to signal the CPU via DTACKB to terminate this bus (vector) cycle. In addition, the IRQB line must be driven inactive (c_1) after IACKB became inactive again. In the M-Module specification, the latter event is the one that is assumed to clear the interrupt condition ("hardware end-of-interrupt"). However, this simple scheme does not work in our application, since the CPU must clear the UTCSU-internal interrupts by writing certain UTCSU-registers in the interrupt service routine.

Therefore, the rising edge of IACKB deactivates the interrupt logic of the NTI and forces the IRQB signal inactive. Before the CPU leaves the ISR, it has to explicitly re-enable the NTIs interrupt logic by writing a certain NTI-register, i.e., *Dis/Enable Interrupt Logic*. Hence, any active UTCSU-interrupt source that has not been recognized within the ISR raises another interrupt (a_3). Of course, implementing this feature requires additional logic, which finally justifies the memory map of the NTI I/O-space in Figure 7.

4 NTI Implementation

Two different prototypes of the NTI are required for debugging resp. evaluation purposes. Throughout the paper, we will call them debug (DB) resp. evaluation board (EB), with the following meaning:

- **Evaluation board (EB)**

  The EB must completely fit on a single-sized M-Module and is used for final experimental evaluation. Lacking space is in fact the dominating constraint for the NTI design, since the 180-pin (socketed) PGA of the UTCSU eats up most of the available high component space on the M-module. Note that the M-module specification defines a high component space (max. height 10.5 mm) resp. a low component space (max. height 5.25 mm) located near the front panel connector resp. the MA-interface connector. Any socketed device (opto-coupler, DIP-buffer), jumper field, and also most TCXOs/OCXOs must hence be placed in the remaining high component space. Similarily, the available low component space rules out any CPLD that exceeds the size of a PQFP-100 or TQFP-100 package.

- **Debug board (DB)**

  The DB is primarily required for verifying proper operation of the UTCSU-ASIC[6] by conducting a scan-test. For this purpose, it must be possible to drive all input signals and monitor all output signals by software, i.e., by accessing certain registers in the NTI over the MA-interface. This way, measurement data can be compared against simulation data gathered during the chip design process.

  Moreover, the DB is required for conducting an in-depth software-based verification of the UTCSU's functionality. Apart from spotting design faults, this also allows automatic and fast validation of the chips from an untested production run. In addition, the need for in-depth debugging may also arise from unexplained problems during the evaluation phase. To track down such problems, it must be possible to run the evaluation on a system where an EB is replaced by a DB connected to a logic analyzer. Of course, this means that the DB must provide the same hardware and software functionality as EB does. Hence, several restrictions set forth by the design of the EB (like the interfacing architecture) apply to the DB as well.

  The above requirements primarily imply that a larger CPLD and numerous additional connectors for the logic analyzer pods must be provided on the PCB of the DB. On the other hand, the resulting size of the DB does not matter — a dual- or tripe-sized M-Module is acceptable here. It is important, however, to use either the MA-interface connector #0 or, preferably, #1 (cf. Figure 11) on a multiple-sized M-module to ensure compatibility with certain VME carrier boards (like AcQ's i6040 CPU). Note finally that, with the exception of the connectors to the carrier board (MPC and CPC, see below), all components of the DB should be mounted on the soldering side of a standard M-module for easy access (and socketed for easy replacement as well).

We will now elaborate on implementation details of our prototype version of the DB. We will guide our description primarily by the NTI's interfaces, which are as follows (recall Figure 4):

[6] To reduce the risks of failures in the ASIC design flow [Hor96], we developed a stripped-down version of the UTCSU called UTCLIENT, see [HL97] for an overview. Providing considerably reduced functionality only, the UTCLIENT's pinning is nevertheless the same as the pinning of the UTCSU, which allows replacement without any modification of the NTI.

- 25-pin D-sub front panel connector (FPC)
- 24-pin plug connector to the carrier board for peripheral I/O (CPC)
- 60-pin plug connector to the carrier board for MA-interface (MPC)
- 2 x 30-pin plug connectors for interconnecting M-Modules (IPC)

In addition, there is an additional 10-pin JTAG plug connector for programming the Altera CPLD on board the NTI as well as various jumpers. (The DB version of the NTI also carries plug connectors for logic analyzer.)

4.1 General Issues

The oscillator pacing the UTCSU is a critical component in high-accuracy clock synchronization. The most suitable frequency would be 2^{24} Hz $\approx$ 16 MHz (2^{25} Hz are unfortunately beyond the current UTCSU's capabilities ...), although the UTCSU can in principle be paced with any frequency. A TCXO or OCXO with TTL compatible output should be given preference, since the usual clipped sinewave output of standard oscillators would require a negatively supplied comparator with very low threshold and offset voltage for coupling such a device to standard logic. However, this requirement in conjunction with the severely limited space on the EB considerably restricts the number of suitable TCXOs and OCXOs. This is particularily true since the device is to be socketed (and hence not SMD), which is required for our experimental evaluation. Note that a very low-height socket is required here, since OCXOs have considerable height!

Most of the suitable TCXOs/OCXOs offered on the market have a DIP-14-compatile pinning as shown in Figure 2. In addition, there are also DIP-8-compatible TCXO's around, with an analogous pin-out. The supply-voltage should be jumper-selectable 5 V or else 12 V, since some OCXOs require 12 V for the oven.

Pin	**Signal**
1	n.c./tuning
7	GND
8	frequency out
14	$+V_{supply}$

Table 2: *Usual pin-layout of DIP-14-compatible TCXOs/OCXOs*

Finally, there must be a jumper for selecting the NTI clock source, which can be either the TCXO/OCXO or a 10 MHz reference frequency input from a GPS-receiver (opto-coupled as well as non opto-coupled), see Section 4.3.

4.2 MA-Module Interface (MPC)

A CPU and/or COMCO on a carrier-board can access the NTIs features via the MA-Module interface. The following signals of the MA-interface are relevant for our NTI board:

Carrier Board	NTI	Notes
D0–D31	D0–D31	Primarily, the NTI is targeted for a port size of 32 bit, but supports 8- resp. 16-bit accesses as well.
A1–A7		As long as ASB is "H", only A1–A7 are valid to address the 256 byte I/O space. When ASB is "L", A1–A23 address 16 Mbytes of memory. In the latter case, the lines D0–D15 must be latched at the falling edge of ASB. The latched value serves as A8–A23.
WRITEB		When "L" the carrier board writes to the NTI.
ASB		On the "H" to "L" transition of this signal, D0–D15 must be latched. The contents of this register then serves as address lines A8–A23.
CSB		This indicates with "L" a normal R/W access to the NTI. When the carrier board reads data from the NTI, the module must drive the data lines as long as CSB is asserted.
	DTACKB	The NTI signals the Carrier board that it is ready to terminate the current bus cycle.
DS0B–DS2B + A1		Determine the valid bytes for data bus routing.
RESETB		The UTCSU's RESET-signal is active high, thus RESETB must be inverted.
	IRQB	The NTI tries to interrupt the carrier board. The signal must be driven active until the CPU acknowledges with IACKB.
IACKB		IACKB serves as select line instead of CSB during interrupt cycles. When this signal goes low, the NTI should put an interrupt vector onto D0–D7. The highest priority interrupt pending should always be presented first. The module finally asserts DTACKB.
	DREQB	DMA transfer requested by the module (not used for the NTI prototype)
DACKB		DMA access granted by the carrier board to the module (not used for the NTI prototype).
SYSCLOCK		16 MHz clock that can be fairly asymetric.
TRIGA, TRIGB		User-defineable trigger signals.

Note that we use the convention of appending "B" to the names of active-low signals instead of a preceeding "/", i.e., DS0B instead of /DS0. The signals are split between those generated on the carrier board and those originating at the NTI. Bi-directional signals are marked on both sides. The layout of the MA-interface plug connector is given in Table 3.

Pin	Row A	Row B	Row C
1	CSB	GND	ASB
2	A01	+5V	D16
3	A02	+12V	D17
4	A03	-12V	D18
5	A04	GND	D19
6	A05	DREQB	D20
7	A06	DACKB	D21
8	A07	GND	D22
9	D08/A16	D00/A08	TRIGA
10	D09/A17	D01/A09	TRIGB
11	D10/A18	D02/A10	D23
12	D11/A19	D03/A11	D24
13	D12/A20	D04/A12	D25
14	D13/A21	D05/A13	D26
15	D14/A22	D06/A14	D27
16	D15/A23	D07/A15	D28
17	DS1B	DS0B	D29
18	DTACKB	WRITEB	D30
19	IACKB	IRQB	D31
20	RESETB	SYSCLK	DS2B

Table 3: *Layout 60-pin MA-connector (MPC)*

The glue logic for adapting the MA-interface to the components on board the NTI (primarily UTCSU and memory) is primarily contained in an Altera EPM7128 CPLD. It performs address decoding with and without special functionality, hosts the NTI-internal registers, and handles bus cycle termination as well as interrupt generation. Apart from the serial PROM, external devices are only required for data bus routing. Figure 9 illustrates the interfacing of the data bus between the carrier-board, the UTCSU, and the memory along with all necessary signals for control.

The routing of multibyte transfers is accomplished as described in [MM96, p. 24]. Accordingly, the select signals of the MA interface indicate the active byte. The signals TROELB and TROEHB controlling the transceivers (74245) must be driven appropriately to suit the needs of memory or UTCSU accesses. The upper part of Table 4 specifies the routing required for memory accesses. The data and select signals on the left side specify the intended data bus routing, the signals on the right side specify the output enables for the transceivers and the appropriate byte and chip select signals.

Unlike the memory, which can also be written byte-wise (since application software could directly access packet data in memory), the registers of the UTCSU can only be accessed in words or longwords, where word accesses are (incompatibly[7]) always expected at the higher data bus interface (D31–16). The lower part of Table 4 gives the resulting data bus routing for UTCSU-accesses; again, the right side provides the output enables for the transceivers, whereas the left side states the appropriate (byte-)select signals for UTCSU accesses.

[7]Recall our remark in Section 2; as a remedy, we employ additional 2 x 8 bit transceivers (74245).

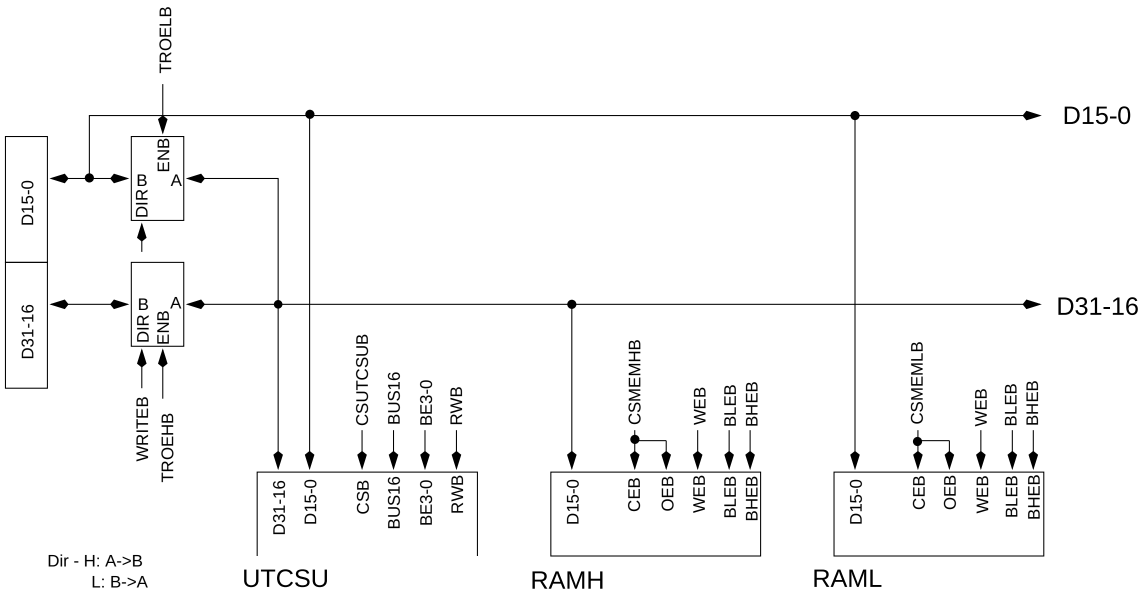


Figure 9: *The data bus interface*

Memory Accesses					
Data		DS2B–DS0B			
MA	NTI	& A1	TROEH(L)B	BH(L)EB	CSMEMH(L)B
D00–D07	D00–D07	LHLH	HH	HL	HL
D08–D15	D08–D15	LLHH	HH	LH	HL
D16–D23	D16–D23	LHLL	LH	HL	LH
D24–D31	D24–D31	LLHL	LH	LH	LH
D00–D15	D00–D15	LLLH	HH	LL	HL
D16–D31	D16–D31	LLLL	LH	LL	LH
D00–D31	D00–D31	LHHx	LH	LL	LL

UTCSU accesses						
	Data		DS2B–DS0B		BE3-BE0	
MA	NTI	UTCSU	& A1	TROEH(L)B	& Bus16	CSUTCSUB
D00–D15	D00–D15	D16–D31	LLLH	HL	LLHHH	L
D16–D31	D16–D31	D16–D31	LLLL	LH	HHLLH	L
D00–D31	D00–D31	D00–D31	LHHx	LH	HHHHL	L

Table 4: *Enable signals for memory resp. UTCSU accesses*

4.3 Front Panel Connector (FPC)

The layout of the 25-pin D-Sub connector of the NTI is given in Table 5.

Pin	Signal
1	+5V GPS1
14	GND GPS1
2	1pps GPS1
15	Status GPS1
3	+5V
16	1pps GPS2
4	Status GPS2
17	10MHz GPS2
5	1pps GPS3
18	Status GPS3
6	GND
19	App0
7	App1
20	App2
8	App3
21	App4
9	App5
22	App6
10	App7
23	GND
11	APPDUTY
24	10MHz GPS1
12	App8
25	GND App8
13	+5V App8

Table 5: *Layout of the 25-pin DSUB front panel connector (FPC)*

All signals (1pps, Status, and 10MHz) for the first GPS-interface (GPS1), along with dedicated +5V and GND lines, are fed via fast opto-couplers to the corresponding UTCSU input signals. The same is true for the application timestamping input App8. All signals of the other two GPS receivers and the remaining application timestamping inputs App0–App7 are connected via fast, socketed DIP buffers (7404 or 7414) to the according UTCSU pins; 10 kOhms pull-up resistors are also provided to circumvent floating inputs. The UTCSU's APPDUTY-Output ist buffered by a 74240. The exported +5V are fused to avoid damage of the PCB in case of short circuits.

4.4 Carrier-board Plug Connector (CPC)

The layout of the 24-pin plug connector that connects the NTI to the carrier-board for I/O is given in Table 6. All Signals are pulled up by 10k resistors and buffered to protect the UTCSU from damage; 74240 SMD devices are used for all signals except TSAPP0–TSAPP8, which are the same as App0–App8 at the FPC.

Pin		Signal
1		SYNCRUN
	2	HWSNAP
3		TSXMT1
	4	TSRCV1
5		TSXMT2
	6	TSRCV2
7		TSXMT3
	8	TSRCV3
9		TSXMT4
	10	TSRCV4
11		TSXMT5
	12	TSRCV5
13		TSXMT6
	14	TSRCV6
15		TSAPP0
	16	TSAPP1
17		TSAPP2
	18	TSAPP3
19		TSAPP4
	20	TSAPP5
21		TSAPP6
	22	TSAPP7
23		TSAPP8
	24	APPDUTY

Table 6: *Layout of the 24-pin plug connector to the carrier-board (CPC)*

4.5 Intermodule Port Plug Connector (IPC)

The layout of the 2 x 30-pin intermodule port plug connector that can be used for further interconnections is given in Table 7. Special care must be exercised when using the IPC, since only the signals fed to the three GPUs are buffered — the 49 UTCSU-outputs comprising the NTPA-bus are directly wired to the IPC due to lacking space for buffers.

4.6 Miscellaneous Connectors

The layout of the 10-pin JTAG plug connector for programming the Altera CPLD is given in Table 8. Note that it should be accessible from the front panel, see Figure 11.

4.7 Debug Board Connectors

The DB-version of the NTI must provide numerous additional connectors for feeding all UTCSU and MA-interface signals to a HP 16550A logic analyzer. Layout and placement on the PCB should[8] ensure compatibility with the pods shown in Figure 10. Note that those figures show the pods and not the connectors to be mounted on the PCB.

Any pod (16 channels) of the logic analyzer must be connected to the PCB via a 20-pin 100 kOhm termination adapter 01650-63203, which is just plugged onto the 40-pin cable plug of the

[8]Note that the prototype version does not comply to this requirement!

Pin	Signal
1	CSUPHASE
2	CSUT47
3	CSUT46
4	CSUT45
5	CSUT44
6	CSUT43
7	CSUT42
8	CSUT41
9	CSUT40
10	CSUT39
11	CSUT38
12	CSUT37
13	CSUT36
14	CSUT35
15	CSUT34
16	CSUT33
17	CSUT32
18	CSUT31
19	CSUT30
20	CSUT29
21	CSUT28
22	CSUT27
23	CSUT26
24	CSUT25
25	CSUT24
26	+5V GPS1
27	1pps GPS1
28	Status GPS1
29	10MHz GPS1
30	GND GPS1

Pin	Signal
1	CSUT23
2	CSUT22
3	CSUT21
4	CSUT20
5	CSUT19
6	CSUT18
7	CSUT17
8	CSUT16
9	CSUT15
10	CSUT14
11	CSUT13
12	CSUT12
13	CSUT11
14	CSUT10
15	CSUT9
16	CSUT8
17	CSUT7
18	CSUT6
19	CSUT5
20	CSUT4
21	CSUT3
22	CSUT2
23	CSUT1
24	CSUT0
25	1pps GPS2
26	Status GPS2
27	10MHz GPS2
28	1pps GPS3
29	Status GPS3
30	GND

Table 7: *Layout of the 2 × 30-pin plug connectors of the intermodule port (IPC)*

Pin	Signal
1	TCK
2	GND
3	TDO
4	+5V
5	TMS
6	n.c.
7	n.c.
8	n.c.
9	TDI
10	GND

Table 8: *Layout of JTAG plug connector*

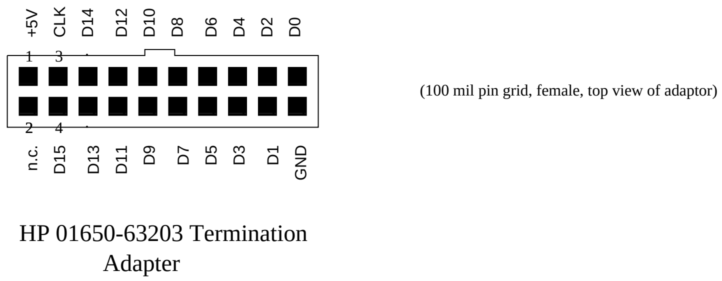


Figure 10: *Connector layout of a pod of the HP 16550A logic analyzer*

appropriate flat cable. Note that this requires considerable[9] (about 10 cm) inter-board space if the NTI (with connected logic analyzer pods) is plugged into a VME rack. Note that the +5 V supplied by the logic analyzer must not be connected to the PCB!

As far as signal grouping w.r.t. the various connectors is concerned, there are basically four sets of signals of independent interest:

- UTCSU ordinary signals (data, address, control, timestamp-signals)
- UTCSU testbus (NTPA-bus, CSUPHASE)
- UTCSU scantest (SCANIN, SCANOUT, SCANEN, SCANTEST)
- MA-interface

Grouping them appropriately, i.e., assigning a particular signal to a particular connector, allows the appropriate measurements to be carried out without needing all channels of the logic analyzer. Apart from this requirement, signal grouping is also constrained by the need to assign input and output signals to different connectors.

Figure 9 shows a signal grouping example that conforms to the above requirements. Note that the UTCSU's TCLK signal and the CPLD's I/CLK1 signal should be fed to the CLK-pin of at least one logic analyzer connector per signal set, e.g., to P1/P2, P6/P7, P10/P11, P13/P12; connector P14 should be provided with the CPLD's I/CLK1. Note that those connector numbers do not correspond with those used in the schematics of our NTI prototype version provided in the appendix.

Finally, there is an additional requirement that showed up in the course of testing our prototype version: Any signal fed to a logic analyzer connector should also have a dedicated via near the approprate pin of the connector, which facilitates measurements with an oscilloscope (for detecting ringing and similar phenomenons) in spite of plugged-in logic analyzer pods.

It is important to use the same orientation (preferably vertical) of <u>all</u> logic analyzer connectors on the PCB. In fact, they should be arranged in aligned rows (with enough space between them to allow plugging in the pods!) to avoid problems with cabling. A suitable arrangement is outlined in Figure 11.

[9]Alternatively, one might try to directly use the 40-pin cable plug, which would allow less inter-board space. However, a termination network must be provided on the PCB in this case; the one inside the termination adapter routes a any logic analyzer input via 8.2 pF (±2%) || 90 kOhms (±1%) followed by a serial resistor 250 Ohms (±1%) to the test pin.

Conn.No.	Signals
P1	DA0–DA15
P2	D16–D31
P3	BE0–BE3, A2–A8, WEB, CSUTCSUB, WAITCYCLE, BUS8, BUS16
P4	TSAPP0–TSAPP8, 1PPS1, STAT1, 1PPS2, STAT2, 1PPS3, STAT3, APPDUTY
P5	TSRCV1–TSRCV6, TSXMT1–TSXMT6, HWSNAP, SYNCRUN, RESET, RELIABLE
P6	CSUT0–CSUT15
P7	CSUT16–CSUT31
P8	CSUT32–CSUT47
P9	INTT, INTN, INTA, XO, CSUPHASE, A9–A19
P10	RELH, BIGEND, TESTMUX, DIRECTIN, SCANTEST, SCANEN, SCANIN1–SCANIN10
P11	SCANOUT1–SCANOUT10
P12	ADDR1–ADDR7, ASB, CSB, WRITEB, DSB0–DSB2, RESETB, IACKB, DACKB
P13	DTACKB, IRQB, DREQB, TROELB, TROEHB, CSMEMHB, CSMEMLB, BHEB, BLEB, WEB, CS, CK, DI, DOUT, LA_TRIGGER
P14	16 debug pins from the CPLD

Table 9: *Signal grouping for logic analyzer connectors*

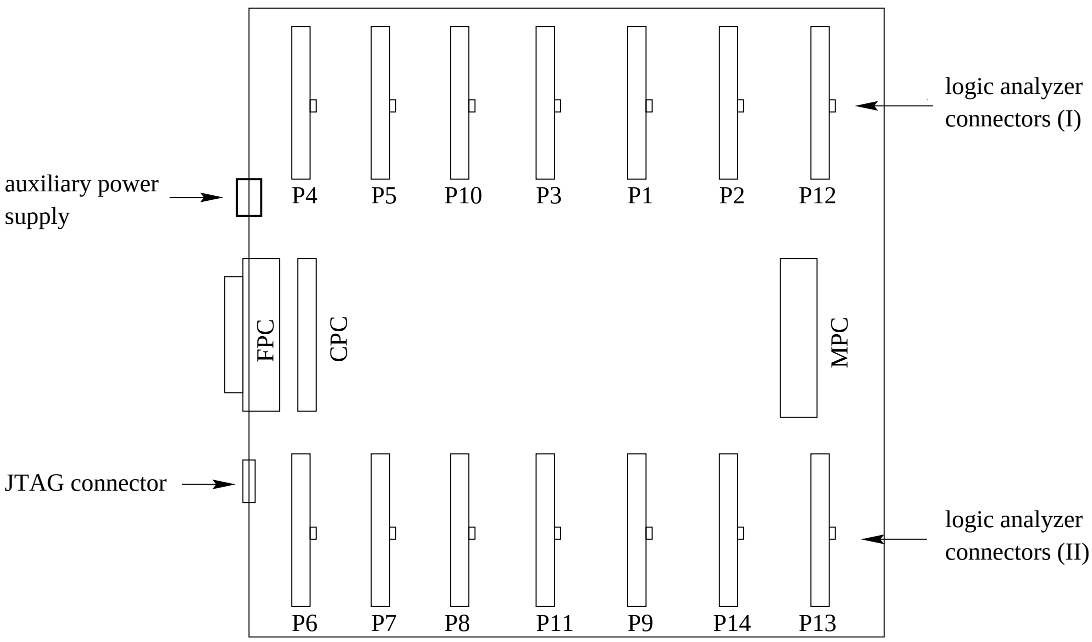


Figure 11: *Principal layout of logic analyzer connectors on the DB*

5 CPLD Description

Most of the glue logic of the NTI is accommodated in a single Altera EPM7128 CPLD, which can be easily re-programmed in place for modification purposes. Further advantages of using a CPLD over generic logic are the possibility of high level language (VHDL) description and simulation capabilities. The entire CPLD is thus coded in VHDL, with the listings provided in the appendix. This section gives a brief description of our implementation.

5.1 I/O logic

The following signals resp. registers are located within the I/O address space, the access of which requires the following pre-conditions:

- CSB is active low, indicating an access to the NTI
- ASB stays high, thus the addresses are not latched from the data bus; A7–1 form the valid addreses

⇒ Signal IOACCESS is active high when both conditions from above are valid.

Only a small portion of the I/O space is used, the rest is reserved for future enhancements.

	31:24	23:16	15:8	7:3	2:0	READ/WRITE
SPROM	-	-	-	-		0xFE

At address 0xFE the serial PROM is selected. Only the three least significant data lines are meaningful here: D0 serves as READ/WRITE, D1 as CLOCK and D2 as DATA line for accessing the serial PROM.
The active high signal SPROM follows the MA-Module chip select CSB. The address lines and the address strobe ASB must be settled properly in advance of CSB. SPROM is in turn used for generating the signals that actually control writing resp. reading of the serial PROM.

	31:24	23:16	15:8	7:0	READ/WRITE
LA_TRIGGER	-	-	-	-	0x06

On the DB-version of the NTI, LA_TRIGGER is provided to post-trigger the logic analyzer by software. Whenever this NTI-internal register is read/written, the CPLD generates a pulse at a dedicated pin at the NTI.

	31:24	23:16	15:8	7:0	READ/WRITE
ENABLEIRQ	-	-	-	-	0x04

Signal ENABLEIRQ (re-)enables the interrupt logic, which is usually done once after NTI initializaton and regularily as the last command of an interrupt service routine. The active high signal ENABLEIRQ follows the MA-Module chip select CSB. The address lines and the address strobe ASB must be settled properly in advance of CSB.
The interrupt logic must be disabled following a system reset and at every rising edge of signal IACKB provided by the MA-interface.

	31:24	23:16	15:8	7:0	READ
SINTVEC	-	-	-	Interupt Vector	0x02
				00	Reset

	31:24	23:16	15:8	7:4	3:0	WRITE
SINTVEC	-	-	-	Vector	-	0x02
				0000	0—	Reset

Accessing the interrupt vector needs a distinction between a read and a write operation. During read the interrupt vector is presented on D7–0, during write the base address of the vector is programmed with D7–4. In both cases this register within the CPLD is selected with SINTVEC, which follows CSB.

	31:24	23:16	15:13	12:0	READ
RADL	-	-	-	RFD Base Address	0x00
				00..00	Reset

On packet reception the base address of the memory location, where the packet header is written to, is stored in the NTI-internal *Receive Header Base* register within the CPLD. Signal RADL follows CSB and selects this register for read-out.

5.2 Memory Addressing Logic

The memory address space is formed by latching the least significant data word at the beginning of a bus cycle and using this information as extended address in conjunction with the value derived from the address lines A1–A7. Thus the memory space has the following pre-conditions:

- CSB is active low, indicating an access to the NTI
- ASB has a high to low transition at the beginning of the bus cycle, indicating that valid addresses are present on the data bus. This high to low transition is used to latch the higher addresses.

	31:24	23:16	15:8	7:0	READ/WRITE
AH	-	-	DATA Bus 15 - 0		ASB high - low
	00	00	00	00	Reset

At any high to low transition of ASB, the data lines D15–D0 are latched into an internal CPLD register AH.

In addition certain regions of the memory space are selected to decide if either the Receive Headers or Transmit Headers section are addressed, cf. Section 3.4.

- Select the Transmit Headers Section via signal MEMTCBHEADER when the addresses form A19–A12 = 0b00000000.
- Select the Receive Headers Section via signal MEMRFDHEADER when the addresses form A19–A16 = 0b0000 and A15-A12 $\neq$ 0b0000.

Within the Transmit Headers section the following decodings are valid:

	31:24	23:16	15:8	7:0	READ
TSXMTB	-	-	-	-	0b..x01010x

Offset in memory	UTCSU register (address)
0x18	MSXMT (0x80148)
0x1C	TSXMT (0x8014C)
0x20	ACCXMT (0x80150)

Table 10: *Address re-mapping*

Signal TSXMTB triggers a transmit timestamp of the UTCSU and follows the CSB signal, thus it requires the address lines (higher and lower part) and the WRITEB signal properly settled.

The UTCSU-registers holding the transmit timestamp, which need to be re-mapped into memory, are summarized in Table 10.

The 32 bit UTCSU register MSXMT holds the Macrostamp value, consisting of the upper bits of NTP time as well as an 8 bit checksum. Register TSXMT contains the lower 32 bits of NTP time, with an LSB representing about 60 ns. Finally, the 32 bit register ACCXMT concatenates two 16 bit words formed by positive and negative accuracy.

	31:24	23:16	15:8	7:0	READ
MEMMSXMT	-	-	-	-	0b..x01100x

	31:24	23:16	15:8	7:0	READ
MEMTSXMT	-	-	-	-	0b..x01110x

	31:24	23:16	15:8	7:0	READ
MEMACCXMT	-	-	-	-	0b..x10000x

Signals MEMMSXMT, MEMTSXMT resp. MEMACCXMT indicate the request to remap the current address to the UTCSU registers MSXMT, TSXMT resp. ACCXMT. These signals are directly decoded from the address lines and the WRITEB signal so they may have some hazards. Note that these signals need a valid chip select otherwise re-mapping will not occur.

Within the Receive Headers section, the following decodings are valid:

	31:24	23:16	15:8	7:0	WRITE
TSRCVB	-	-	-	-	0b..x01110x

On packet reception, when the frame is written to the memory, the CPLD may decode a receive timestamp via signal TSRCVB. It is activated with CSB and requires proper addresses and a valid inactive WRITEB signal.

	31:24	23:16	15:13	12:0	WRITE
TADL	-	-	-	RFD Base Address	0b..x01110x
				00..00	Reset

In addition, the base address of the currently received packet header has to be stored in the NTI-register *Receive Header Base*, to get a cross-reference between the sampled timestamp and its according packet. Signal TADL, which follows CSB and requires the addresses and the WRITEB signal settled properly, triggers this operation. Note, that signal TSRCVB and TADL are generated at the same address offset in the current implementation.

Finally the chip select signals for the memory and the UTCSU must be generated. They are derived from the NTI module select CSB signal, address line A19 and the remapping signals for the timestamp transmit signals.

- PRECSMEMB becomes active low when CSB becomes low and ASB, A19, MEMMSXMT, MEMTSXMT and MEMACCXMT are all zeroes. From this signal the chip select for the memory is generated in combination with the data select lines.
- PRECSUTCSUB becomes active low when CSB becomes low and ASB is low and A19 is high, or when one of the remapping signals MEMMSXMT, MEMTSXMT or MEMACCXMT becomes high. Again this signal in combination with the data selects is responsible for generation of the UTCSU chip select.

In addition several byte selects must be valid according to Table 4, and ASB should be active low, otherwise signals from a previous bus cycle could intervene. The chip, byte and bus select signals are derived via PRECSMEMB, PRECSUTCSUB and these tables.
The signals to control the transceiver for data bus routing TROELB and TROEHB are also controlled by the byte enables as enlisted in these tables. However, instead of the signals PRECSMEMB and PRECSUTCSUB, the module select CSB can be used.

Figure 12 summarizes the provisions making up the whole address re-mapping logic.

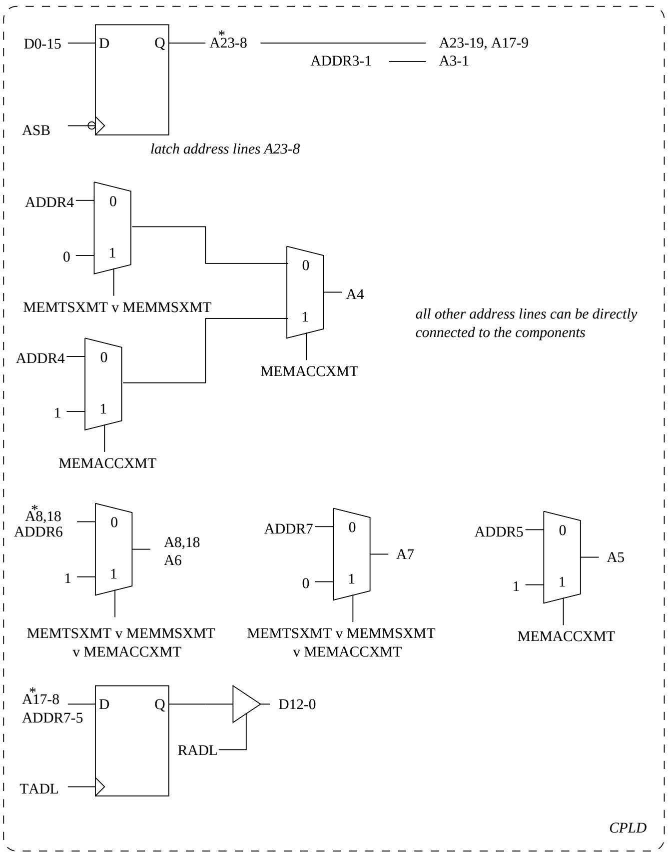


Figure 12: Special address mapping for UTCSU registers

Number	Characteristic	typical timings [ns]
1	t_addr_data_valid	20
2	t_addr_valid	10
3	t_asb_valid	10
4	t_dataout_valid	0
5	t_dtackb2csb_invalid	30
6	t_dtackb_valid	0
7	t_dataout_invalid	40
	t_addr_invalid	40
	t_asb_invalid	40
8	t_dtackb_invalid	40
9	t_dtack2datain_valid	25
10	t_dtack2datain_ invalid	30

Table 11: *Timing parameters for the simulation model*

5.3 Miscellaneous Features

Finally the CPLD has to control the generation of the acknowledgement signal DTACKB and the interrupt request IRQB. The former is generated by the NTI to terminate a valid bus-cycle, so that it is necessarily synchronous w.r.t. the CPLD's clock signal. More specifically, DTACKB becomes active one resp. two clock periods after the CSB in case of memory resp. UTCSU accesses. Note that accesses to the UTCSU-register TSGETL currently involve an additional waitstate cycle (which could possibly be circumvented). Of course, DTACKB becomes inactive *high* again as soon as CSB is withdrawn.

The interrupt request line (IRQB) is activated asynchronously by the NTI after the occurence of an interrupt, that is, when one of the interrupt lines of the UTCSU becomes active. This interrupt request is eventually acknowledged by the carrier board, which asserts IACKB to start a hardware-end-of-interrupt acknowledge cycle. The latter requires the NTI to put an interrupt vector onto the data bus and to assert DTACKB. Note that, in order to avoid potential problems[10] with interfering access cycles, the current version of the CPLD delays the generation of IRQB until the current access is terminated — this is probably an overkill.

6 Simulation Environment

To verify design and operation of the NTI, in particular, w.r.t. interoperability of all embodied devices, a board-level simulation was performed prior to the realization of the module. This is possible, since the most complex components (UTCSU and CPLD) are both coded in VHDL and synthetisized by SYNOPSYS V3.4A[11]. Thus, the idea of building a simulation environment of the whole NTI on top of those descriptions is quite obvious. In fact, we only had to provide a simple logic model of the memory and the external transceivers as well as a testbench modeling an MA-interface according to [MM96] to achieve this goal. Figure 13 along with Table 11 gives the timing diagram describing MA-accesses, which forms the basis of the testbench.

After logic simulation and synthesis with SYNOPSYS V3.4A, the CPLD code was transferred

[10] After all, DTACK is used for both signaling the presence of the interrupt vector on D0–D7 and terminating ordinary read/write accesses; it is not clearly specified in the M-Module specification whether carrier boards are required to exclude any interference.

[11] SYNOPSYS is a registered trademark of SYNOPSYS INC.

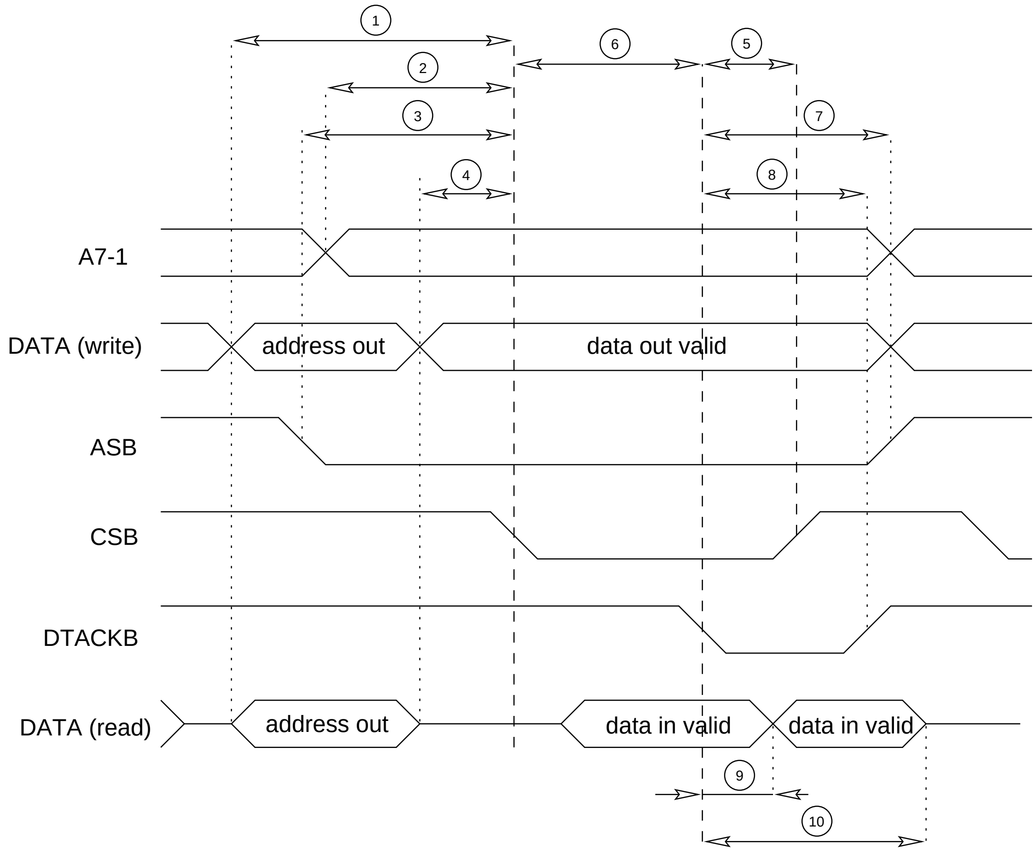


Figure 13: *Timing model of the MA-Module interface*

in *edif* format to MAXPLUS II[12] along with a constraint file (.acf). This tool is capable of fitting the device and calculating back-annotation timings, which were re-exported via the vhdl interface. The final simulation was performed in the SYNOPSYS environment using these extracted data to verify the logic after fitting. Finally, MAXPLUS II was used for in-circuit programming of the CPLD on board the NTI via the JTAG interface.

[12] Maxplus II is a registered Trademark of Altera Corp.

7 Further Enhancements

In this final section, we provide a list of enhancements that should be incorporated in a redesign of the NTI. In fact, our prototype of the DB-version of the NTI exhibits several deficiencies that must be mended in a professional redesign.

7.1 General Enhancements

We start with enhancements that apply to both the DB and the EB-version of the NTI:

- The signals fed into the CPLD should be (slightly) changed, namely,
  - fixed WAITCYCLE-output instead of optional (J22/23-selectable) SYSCLOCK input,
  - do not connect DA0 to both pin 38 and pin 1,
  - do not export A1 on pin 51, since it is not used externally,
  - provide a LA_TRIGGER-output (see below),
  - feed the MA-interface's DREQB and DACKB to the CPLD,
  - if an alternative CPLD with more than the 80 useable I/O-pins of the Altera EPM7128S can be found, RELIABLE should also be connected to the CPLD. Otherwise, a pull-down + jumper for static assignment of RELIABLE should be provided.

  Generally, it is important to devise a pin-assignment for the CPLD that supports the internal structure of the CPLD logic, since (in case of Altera) it is sometimes difficult to fit even small designs in case of an improperly laid out pinning.

- The CPLD logic must be modified/extended, namely,
  - the default value of the NTI's Vector Base Register should be 0x40 instead of 0x00,
  - a software-reset feature should be provided in the CPLD.

  Moreover, the entire CPLD logic should be reconsidered w.r.t. correct timing behavior, increased performance, and less complexity. This is particularly true w.r.t. DTACKB-generation and interrupt handling. A suitable pin-out for easy fitting should also be considered (see above).

- Do not use the MA-interface's WRITEB for R/W-control of the components —namely, the UTCSU— on-board the NTI, but only WEB. Otherwise, bad signal conditions on the MA-interface lines (ringing!) could produce strange misbehavior.

- Pull-up resistors for the NTI-internal data bus D0–D15 should be provided.

- A dedicated ELA_TRIGGER open-collector output must be provided on the NTI, which can be used for software triggering a logic analyzer or for triggering system snapshots in the evaluation environment. To further support the latter, ELA_TRIGGER should in fact be (one pin of) a jumper that can connect ELA_TRIGGER to the EHWSNAP-signal (CPC).

- There should be clamp diodes for the EAPPDUTY-output to protect the 74240 SMD buffer. An even more preferable solution would be to provide a dedicated o.c. output here, i.e., avoiding to go over the SMD buffer. (Are there dual fast o.c. driver in a single small package, which could be used for EAPPDUTY and ELA_TRIGGER?).

- The n.c.-pin of the opto-couplers must not be connected at all; reconsideration of the data-sheet is generally advisible.
- Another pair of jumpers (like J1–J4) should be provided, which allows to select TSXMT2 resp. TSRVC2 from either the 24-pin I/O connector or else from TRIGA resp. TRIGB of the MA-interface.
- A more suitable (easy to replace) fuse for the exported +5V must be provided.
- The TCXO/OCXO layout must be adopted to the one of Section 4.1; both DIP-14 and DIP-8 vias should be provided. A jumper for selecting the supply voltage must be provided, but the TL712 comparator U10 as well as J9, J11 can be omitted. Note that an extremely low-height socket is required since typical OCXOs have a height of 8 mm!
- The UTCSU must of course be socketed (ZIF on the DB and ordinary one on the EB); a PGA socket without central (unused) pins might be advantageous w.r.t. routing the EB's PCB.
- More and/or better placed decoupling capacitors close to the CPLD (see Altera manual), the UTCSU, the transceivers, and the opto-couplers (see data sheet!) should be provided.
- The PCB design must be suitable for clock frequencies up to 20 MHz (if possible, 32 MHz should be envisioned). Note that there are suggestions in the Altera CPLD manuals how to proper route wires on such a PCB.

The prototype of the EB should employ wire-wrap versions of MPC and CPC as well as an wire-wrap UTCSU-socket. This way, it is possible to connect measurement equipment even if the NTI is plugged onto a carrier board.

7.2 DB Enhancements

In addition to the general issues, the DB-version of the NTI calls for the following improvements:

- A (much) larger, socketed CPLD (with at least 178 available I/O-pins, e.g. an Altera EPM9560S in a 240-pin RQFP) for extended debugging should be provided, which is connected to additional signals:
  - UTCSU's NTPA-bus CSUT0–CSUT47 and CSUPHASE
  - UTCSU-pins SYNCRUN, HWSNAP, APPDUTY, PORESET, SYNCRUN, RELIABLE, RELH, BIGEND, TESTMUX, DIRECTIN, SCANTEST, SCANEN, SCANIN1–SCANIN10, SCANOUT1–SCANOUT10
  - 16 additional debug I/O-pins routed to an additional logic analyzer connector (see below)
  - An additional Output-pin routed to the clock selection logic (see below) for generating a software-clock for the UTCSU (scan-test).
- A slightly more flexible clock frequency selection outlined in Figure 14 should be provided, which allows to supply the CPLD with twice the input clock of the UTCSU or the MA-interface's SYSCLOCK.

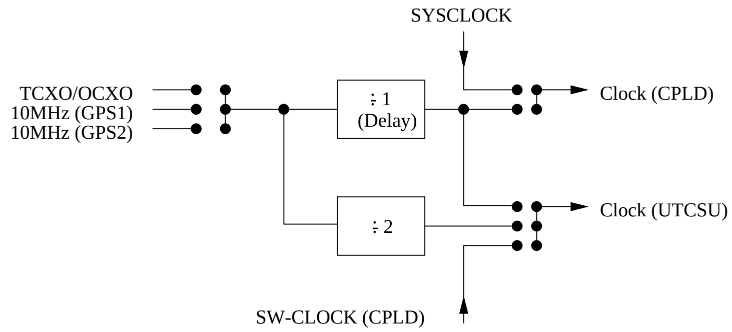


Figure 14: *Flexible clock selection for the DB*

- The MPC and CPC to the carrier board must be positioned such that they plug directly into the M-module slot #0- or, preferably, #1-connector of a standard carrier board. All other components must be mounted on the soldering side of a standard M-module.
- The connectors for the logic analyzer must be adopted to the connector layout and signal grouping described in Section 4.7. Moreover, the additional vias for tapping with an oscilloscope are to be provided.
- The auxiliary power supply connector should be fused to avoid damaging the PCB when a defective UTCSU is plugged in. A power-LED should also be provided.

References

[Bau94] B. Baumgarten. *Hochgestapelt. Aufsteckmodule für VMEbus-Karten*, ELRAD, 4, 1994.

[Hoe96] D. Höchtl. *Verfügbarkeitsuntersuchung von GPS-Satellitenempfängern*, Diploma Thesis, Technische Universität Wien, Dept. of Automation, 1996.

[Hor96] M. Horauer. *A Primer to Digital Design with Synopsys and Cadence*, TR 183/1-68, Technische Universität Wien, Dept. of Automation, October 1996.

[HL97] M. Horauer, D. Loy. *UTCLIENT*, Austrochip 97, to appear.

[HSS97] M. Horauer, K. Schossmaier, U. Schmid. *NTI: A Network Time Interface M-Module for High-Accuracy Clock Synchronization*, (paper submitted).

[KO87] H. Kopetz, W. Ochsenreiter. *Clock Synchronization in Distributed Real-Time Systems*, IEEE Transactions on Computers, C-36(8), p. 933–939, August 1987.

[LL84] J. Lundelius, N. Lynch. *An Upper an Lower Bound for Clock Synchronization*, Information and Control, vol. 62, p. 190–204, 1984.

[Loy96] D. Loy. *GPS-Linked High Accuracy NTP Time Processor for Distributed Fault-Tolerant Real-Time Systems,* PhD thesis, Vienna University of Technology, Department of Computer Technology, April 1996.

[MM96] Manufacturers and Users of M-Modules (MUMM) e.V., *M-Module Specification*, 1996.

[Mic93] Micron Inc. *SRAM 1993 Data Book*, 1993.

[Mil91] Mills D.L., "Internet Time Synchronization: The Network Time Protocol", *IEEE Transactions on Communications,* 39(10), pp. 1482–1493, October 1991.

[Pfi89] A. Pfister. *Metastability in digital circuits with emphasis on CMOS technology amplifier*, PhD thesis, ETH Zürich 1989.

[Ri97] G. Richter. *Drivers for Real-Time Communication Coprocessors*, Diploma thesis, Technische Universität Wien, Departement of Automation, to appear 1997.

[RKS90] P. Ramanathan, D.D. Kandlur, K.G. Shin. *Hardware-Assisted Software Clock Synchronization for Homogeneous Distributed Systems*, IEEE Transactions on Computers, 39(4), p. 514–524, April 1990.

[RSB90] P. Ramanathan, K. G. Shin, R. W. Butler. *Fault-Tolerant Clock Synchronization in Distributed Systems*, IEEE Computer, 23(10), p. 33–42, October 1990.

[Sch94] U. Schmid. *Monitoring Distributed Real-Time Systems*, Journal of Real-Time 7, 1994.

[Sch95] U. Schmid. *Synchronized Universal Time Coordinated for Distributed Real-Time Systems*, Control Engineering Practice 3(6), p. 877–884, 1995.

[Sch97a] U. Schmid (ed.). Special Issue J. Real-Time Systems on *The Challenge of Global Time in Large-Scale Distributed Real-Time Systems*, Journal of Real-Time 12, 1997.

[Scho97] K. Schossmaier. *An Interval-Based Framework for Clock Rate Synchronization Algorithms*, (manuscript submitted).

[SL96] K. Schossmaier, D. Loy. *An ASIC Supporting External Clock Synchronization for Distributed Real-Time Systems*, Proceedings of the 8th Euromicro Workshop on Real-Time Systems, June 1996.

[SS95] K. Schossmaier, U. Schmid. *UTCSU Functional Specification*, Technical Report 183/1-56, Vienna University of Technology, Department of Automation, July 1995.

[SS97] U. Schmid, K. Schossmaier. *Interval-based Clock Synchronization*, Journal of Real-Time Systems 12:2, March 1996.

[SSHL97] K. Schossmaier, U. Schmid, M. Horauer, D. Loy. *Specification and Implementation of the Universal Time Coordinated Synchronization Unit (UTCSU)*, Journal of Real-Time Systems 12:3, May 1997.

[Tro94] G.D. Troxel. *Time Surveying: Clock Synchronization over Packet Networks*, PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institut of Technology, May 1994.

[VITA97] VMEbus International Trade Association, `URL: http:\\www.vita.com`.

[VRC97] P. Veríssimo, L. Rodrigues, A. Casimiro. *CesiumSpray: a Precise and Accurate Global Clock Service for Large-scale Systems*, J. Real-Time Systems 12(3), May 1997.

[Zoe95] Dieter Zöbel, Wolfgang Albrecht. *Echtzeitsysteme, Grundlagen und Techniken*, International Thomson Publishing GmbH, 1995.

A Timing Aspects

The timing relations of several signals incorporated on the NTI are elaborated in Figure 15. The clock signal CLK is used on the entire NTI. The address lines ADDR are used to address the 256 byte I/O space and WRITEB indicates if either the carrier board reads/writes to the NTI.

When the carrier board writes to the NTI (first access cycle in Figure 15), the MA data bus is valid as indicated. In addition, the value on D15–0 must be latched on the falling edge of the address strobe ASB and provides the address lines A23–8 for memory space accesses. Depending on the chip select CSB and the value on the address lines, signals ENABLEIRQ (enables the interrupt logic), SPROM (selects the serial PROM), SINTVEC (selects the interrupt vector), and RADL (read the stored Receive Header Base) are generated by the CPLD. The following group of signals MEMMSXMT, MEMTSXMT, MEMACCXMT resp. TTSXMT, TTSRCV, TADL cause receive and transmit timestamping and address re-mapping according to Figure 12. Signals CSMEMB or CSUTCSUB are active when the memory or the UTCSU are selected. Finally, signal DTACKB is activated for termination of the bus cycle either one or two consecutive clock periods after the falling edge of CSUTCSUB, or immediately following CSMEMB.

Similarily, the second access in Figure 15 shows the required timings when the carrier board reads data from the NTI.

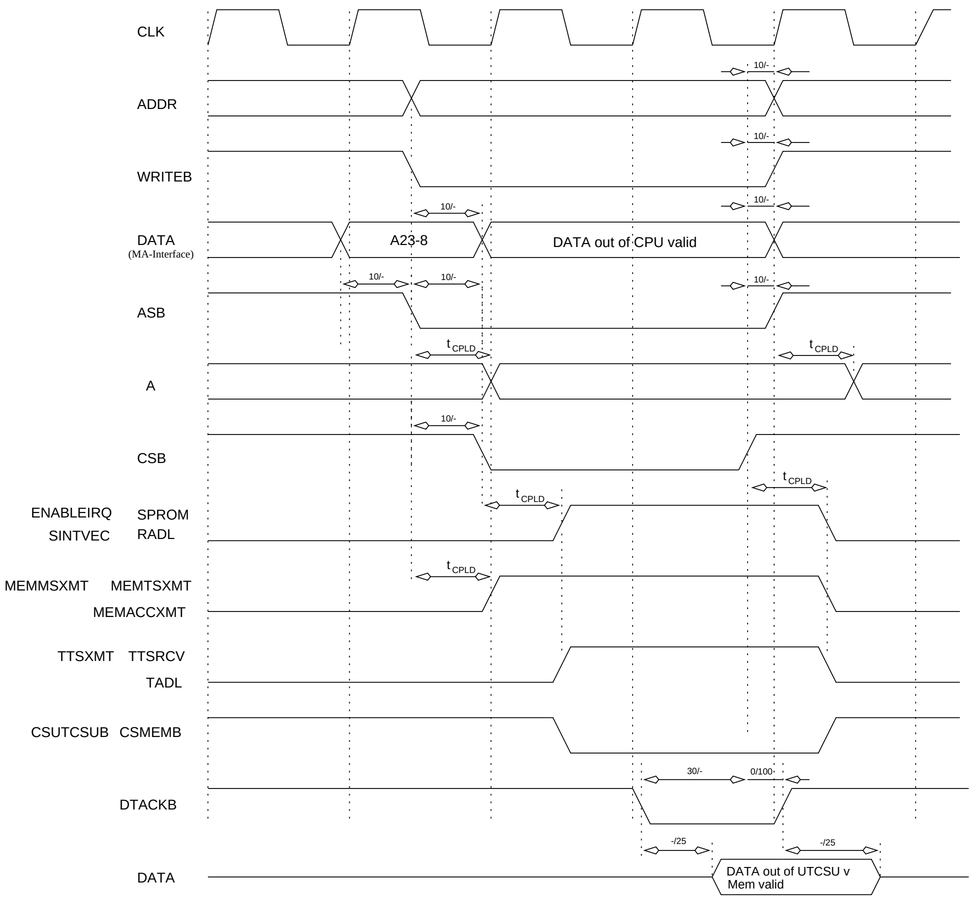


Figure 15: *Timing diagramm*

B VHDL source code of the CPLD

```
-----------------------------------------------------------------------------
--                            ICT - TU Vienna
--
--                             Copyright 1996
--                           All Rights Reserved
--                       CONFIDENTIAL & PROPRIETARY
--
--   File name :  ma_int_cpld.vhd
--   Project   :  NTI
--   Module    :  ma_cpld
--   Purpose   :  adopt MA-Interface to UTCSU requirements
--
-----------------------------------------------------------------------------
--   Modification History :
--          Date     Author  Revision Comments
--       12/1996 Martin Horauer    Rev A Creation
-----------------------------------------------------------------------------

LIBRARY IEEE;
  use IEEE.std_logic_1164.all;

ENTITY ma_cpld IS
  port (
        d0:             out     std_logic;                      --  MA-Interface
        la_trigger:     out     std_logic;
        resetb:         in      std_logic;
        asb:            in      std_logic;
        csb:            in      std_logic;
        writeb:         in      std_logic;
        addr:           in      std_logic_vector(7 downto 1);
        dsb:            in      std_logic_vector(2 downto 0);
        data:           inout   std_logic_vector(15 downto 0);
        dtackb:         out     std_logic;
        irqb:           out     std_logic;
        iackb:          in      std_logic;

        a:              out     std_logic_vector(19 downto 1);  --  UTCSU+RAM

        troelb:         out     std_logic;                      --  Transceiver
        troehb:         out     std_logic;

        di:             out     std_logic;                      --  Serial Prom
        cs:             out     std_logic;
        dout:           in      std_logic;
        ck:             out     std_logic;

        csutcsub:       out     std_logic;                      --  UTCSU
        reset:          out     std_logic;
        tclk:           in      std_logic;
        intt:           in      std_logic;
        inta:           in      std_logic;
        intn:           in      std_logic;
        ttsrcvb:        out     std_logic;
        ttsxmtb:        out     std_logic;
        be:             out     std_logic_vector(3 downto 0);
        bus16:          out     std_logic;
        bus8:           out     std_logic;

        csmemhb:        out     std_logic;                      --  Memory
        csmemlb:        out     std_logic;
        bheb:           out     std_logic;
        bleb:           out     std_logic;
        web:            out     std_logic
       );
END ma_cpld;


ARCHITECTURE behavioral OF ma_cpld IS


  ---------------------------------------------------------------------------
  --  the following signals are used for interconnections
  ---------------------------------------------------------------------------


  SIGNAL        ioaccess, sintvec, tsprom,
                rdidb, wridb, enableirq, disableirq, rint, tirqb, radl,
                readint, dis : std_logic;

  SIGNAL        memaccess, memheader, memrfdheader, memtcbheader,
                memmsxmt, memtsxmt, memaccxmt, tadl,
                a4mt, a4a : std_logic;

  SIGNAL        ah      : std_logic_vector(23 downto 8);
  SIGNAL        rfdbase : std_logic_vector(12 downto 0);

  SIGNAL        precsutcsub, precsmemb, tcsutcsub,
                tcsmemhb, tcsmemlb  : std_logic;

  SIGNAL        tinti : std_logic_vector(2 downto 0);
```

```
  SIGNAL        tintd : std_logic_vector(7 downto 3);

  SIGNAL        tdtackb, tndtackb, tydtackb, tzdtackb, twdtackb,
                intah, intth, intnh : std_logic;


  SIGNAL        enaradl, writeblatched, enaprecsutcsub,
                enadtackb: std_logic;


BEGIN  --  behavioral

----------------------------------------------------------------------------
--  interrupts of UTCLIENT are of wrong polarity(high active): -> invert them
----------------------------------------------------------------------------
intah <= not(inta);
intth <= not(intt);
intnh <= not(intn);

----------------------------------------------------------------------------
--  initialisation
----------------------------------------------------------------------------
csutcsub <= tcsutcsub;         -- chip select for the UTCSU
csmemhb <= tcsmemhb; -- chip selects for the memories
csmemlb <= tcsmemlb;
reset <= not(resetb); -- UTCSU reset is high active
irqb <= tirqb; -- interrupt request
bus8 <='0';

------------------------------------------------------------------------------
--  MEMORY OR I/O ?
------------------------------------------------------------------------------

memaccess <= '1' WHEN (csb='0' AND asb='0') ELSE
             '0';                        -- the CPU performs a memory access
                                         -- to the NTI

ioaccess <= '1' WHEN (asb='1' AND csb='0') ELSE
            '0';                         -- the CPU performs an I/O access

------------------------------------------------------------------------------
--  ##########################################################################
--                                I/O section
--  ##########################################################################
------------------------------------------------------------------------------
--  ==========================================================================
--   generate the select signal for the Serial PROM (active high)
--   SPROM
--  ==========================================================================
p_tsprom : PROCESS (ioaccess,resetb)
BEGIN  --  PROCESS p_tsprom
  IF (resetb='0') THEN
    tsprom <= '0';
  ELSE
    IF (ioaccess='1' AND (addr(7 downto 1)="1111111")) THEN
           tsprom <= '1';
    ELSE
           tsprom <= '0';
    END IF;
  END IF;                                  --  select Serial PROM
END PROCESS p_tsprom;


--  ==========================================================================
--   provide a signal to trigger the Logic Analyzer by Software read/write
--   la_trigger pulse is generated when i/o address 0x6 is accessed
--   LA_TRIGGER
--  ==========================================================================

p_latrigger : PROCESS (ioaccess,resetb)
BEGIN  --  PROCESS p_latrigger
  IF (resetb='0') THEN
    la_trigger <= '0';
  ELSE
    IF (ioaccess='1' AND addr(7 downto 1)="0000011") THEN
        la_trigger <= '1';
    ELSE
        la_trigger <= '0';
    END IF;
  END IF;                                 --  trigger the Logic Analyzer
END PROCESS p_latrigger;


--  ==========================================================================
--   enable the interrupt logic by resetting the Flip Flop used
--   to control enabling/disabling the IRQ (Interrupt) logic
--   ENABLEIRQ
--  ==========================================================================
p_enableirq : PROCESS (ioaccess,resetb)
BEGIN  --  PROCESS p_enableirq
  IF (resetb='0') THEN
    enableirq <= '0';
  ELSE
```

```
    IF (ioaccess='1' AND (addr(7 downto 1)="0000010")) THEN
          enableirq <= '1';
    ELSE
      enableirq <= '0';
    END IF;
  END IF;                                  --  enable IRQ Logic signal decod.
END PROCESS p_enableirq;

--  ======================================================================
--   decode an access to the interrupt vector base at 0x2
--   per default the int-vector base is set to 0x40
--  ======================================================================
p_sintvec_write : PROCESS (ioaccess,resetb)
BEGIN  --  PROCESS p_sintvec_write
  IF (resetb='0') THEN
    tintd <= "01000";                    --  INT-VEC Base 0x40 per default
  ELSE
    IF (ioaccess='1' AND (addr(7 downto 1)="0000001" AND writeb='0')) THEN
        tintd(7 downto 3) <= data(7 downto 3);
    END IF;
  END IF;                                  --  write INT Base
END PROCESS p_sintvec_write;

--  ======================================================================
--   register set for the base address of the Receive Header
--   the following process allows reading of the stored value by the CPU
--
--   in addition the interrupt vector is presented as stated above
--   SINTVEC READ + RADL
--  ======================================================================
p_readint : PROCESS (ioaccess,resetb)
BEGIN  --  PROCESS p_readint
  IF (resetb='0') THEN
    readint <= '0';
  ELSE
    IF (ioaccess='1'
        AND ((addr(7 downto 1)="0000001" AND writeb='1')
          OR iackb='0')) THEN
        readint  <= '1';
    ELSE
       readint <= '0';
    END IF;
  END IF;                                --  Normal Read of INT-Vector
END PROCESS p_readint;

--   signal rint controls reading from the interrupt vector
--   INT Vector must be presented onto the DATA Bus when
--   1.) Normal Read of INT-Vector
--   2.) During IACKB = 0
--                             NOTE: generate DTACKB in both cases!
rint <= (readint);


--   register set for the base address of the Receive Header
--   the following process allows reading of the stored value by the CPU
--
p_radl: PROCESS (radl,rfdbase)
BEGIN  --  PROCESS p_radl
  IF (radl='1') THEN
    data(12 downto 0) <= rfdbase;
    data(15 downto 13) <= "000";
  ELSE
    data(15 downto 0) <= "ZZZZZZZZZZZZZZZZ";
  END IF;                                --  tristate output buffer !! Note:
END PROCESS p_radl;                      --  don't forget pull-up's on the
                                         --  upper 16 bit's of the data-bus

--   decode the Receive Header Base store during reception of the last
--   network packet
--   RADL
enaradl <= '1' WHEN (addr(7 downto 1)="0000000" AND writeb='1') ELSE
           '0';

p_rfdread: PROCESS (ioaccess,resetb)
BEGIN  --  PROCESS p_rfdread
  IF (resetb='0') THEN
    radl <= '0';
  ELSE
    IF (ioaccess='1' AND (enaradl='1')) THEN
         radl <= '1';
    ELSE
         radl <= '0';
    END IF;
  END IF;                                  --  read stored Receive Header Base
END PROCESS p_rfdread;

-------------------------------------------------------------------------------
--  IRQ Logic
-------------------------------------------------------------------------------
p_dis : PROCESS (iackb,resetb,enableirq)
BEGIN  --  PROCESS p_dis
  IF (resetb='0' OR enableirq='1') THEN
    dis <= '0';
```

```
  ELSE
    IF (iackb='1' AND iackb'event) THEN
      dis <= '1';
    END IF;
  END IF;
END PROCESS p_dis;

p_disableirq: PROCESS (resetb,enableirq,dis)
BEGIN  --  PROCESS p_disableirq
  IF (resetb='0' OR dis='1') THEN
    disableirq <= '1';
  ELSE
    IF (enableirq='1' AND enableirq'event) THEN
        disableirq <= '0';
    END IF;
  END IF;                                --  Interrupt-Logic enable/disable
END PROCESS p_disableirq;


--   the UTCSU interrupt sources are hardwired to the interrupt vector

P_IRQINT : PROCESS(tclk,resetb)
BEGIN  --  PROCESS P_IRQINT
  IF (resetb='0') THEN
    tinti <= "111";
  ELSE
    IF (tclk='1' AND tclk'event) THEN
      tinti(0) <= intah;                   --   mapped to Data Bus 0
      tinti(1) <= intth;                   --                      1
      tinti(2) <= intnh;                   --                      2
    END IF;
  END IF; -- Interrupt Vector (interrupts)
END PROCESS P_IRQINT;


--   the whole interrupt vector is put -rint=1- onto the data bus
--   when either the IRQ vector is read (i/o address 0x2) or when
--   a pending interrupt is acknowledged by the CPU -IACKB-
--
P_IRQINTR: PROCESS (rint,tinti,tintd,iackb)
BEGIN  --  PROCESS P_IRQINTR
  IF (rint='1' OR iackb='0') THEN
    data(2 downto 0) <= tinti(2 downto 0);
    data(7 downto 3) <= tintd;
  ELSE
    data(7 downto 0) <= "ZZZZZZZZ";
  END IF;
END PROCESS P_IRQINTR;                   --  tristate buffer for IRQ logic



--   the following signal generates the interrupt request from the NTI
--   to the CPU on thecarrier board
--   IRQB is not allowed to become active when the NTI is currently selected
P_IRQB: PROCESS (resetb,tclk)
BEGIN  --  PROCESS P_IRQB
  IF (resetb='0') THEN
     tirqb <= '1';
  ELSE
    IF (tclk='1' AND tclk'event) THEN
      IF (csb='1' AND (disableirq='0' AND (tinti(0)='0'
         OR tinti(1)='0' OR tinti(2)='0'))) THEN
        tirqb <= '0';
      ELSE
        tirqb <= '1';
      END IF;
    END IF;
  END IF;
END PROCESS P_IRQB;                   --  tristate buffer for IRQ logic


-------------------------------------------------------------------------------
--  Serial PROM
--  the following logic controlls the access schema for the 93C06 E-Eprom
--  that holds the Module Identification Logic
-------------------------------------------------------------------------------
p_wridb: PROCESS (tsprom,resetb)
BEGIN  --  PROCESS p_wridb
  IF (resetb='0') THEN
    wridb <= '1';
  ELSE
    IF (tsprom='1' AND (writeb='0' AND dsb(0)='0')) THEN
        wridb <= '0';
    ELSE
      wridb <= '1';
    END IF;
 END IF;                                --  this signals registers data(1)
END PROCESS p_wridb;                     --  and data(0) to ck and di resp.


p_rdidb: PROCESS (tsprom,resetb)
BEGIN  --  PROCESS p_rdidb
  IF (resetb='0') THEN
```

```
    rdidb <= '1';
  ELSE
    IF (tsprom='1' AND (writeb='1' AND dsb(0)='0')) THEN
        rdidb <= '0';
    ELSE
      rdidb <= '1';
    END IF;
 END IF;                               --  this signal controls the output
END PROCESS p_rdidb;                    --  tristate buffer


p_di_ck: PROCESS (resetb,wridb)
BEGIN  --  PROCESS p_di_ck
  IF (resetb='0') THEN
    cs <= '0';
    di <= '0';
    ck <= '0';
  ELSE
    IF (wridb='0' AND wridb'event) THEN
      cs <= data(2);
      ck <= data(1);
      di <= data(0);
    END IF;
  END IF;                               --  this register generates ck and di
END PROCESS p_di_ck;                    --  the software has to provide the
                                        --  correct values therefore on
                                        --  data(0) and data(1)

p_do: PROCESS (resetb,rdidb)
BEGIN  --  PROCESS p_do
  IF (resetb='0') THEN
    d0 <= 'Z';
  ELSE
    IF (rdidb='0') THEN
       d0 <= dout;
    ELSE
       d0 <= 'Z';
    END IF;
  END IF;                               --  when the Serial Prom is read this
END PROCESS p_do;                       --  tristate buffer connects dout
                                        --  from the Prom to data(0)

-------------------------------------------------------------------------------
--  ###########################################################################
--                                Memory section
--  ###########################################################################
-------------------------------------------------------------------------------
--  ===========================================================================
--  register set for a23-8 derived by latching D15-0 on ASB H->L trans.
--  AH
--  ===========================================================================
p_ah : PROCESS (asb,resetb)
BEGIN  --  PROCESS p_ah
  IF (resetb='0') THEN
      ah(23 downto 8) <= "0000000000000000";
      writeblatched <= '0';
  ELSE
    IF (asb='0' and asb'event) THEN
        ah(23 downto 8) <= data(15 downto 0);
        writeblatched <= writeb;
    END IF;
  END IF;                               --  some of these latched addresses
END PROCESS p_ah;                       --  must be re-mapped others are
                                        --  directly fed to the Memory and
                                        --  the UTCSU Asic

--   most address lines can be directly connected to the external pins
--   without modifications
--
a(18 downto 9) <= ah(18 downto 9);  -- a19,a4-8 are re-mapped
a(3 downto 1) <= addr(3 downto 1);

--   decode the memory header regions
--
p_memheader : PROCESS (ah,resetb)
BEGIN  --  PROCESS p_memheader
  IF (resetb='0') THEN
     memheader <= '0';
  ELSE
     IF ah(19 downto 16)="0000" THEN
        memheader <= '1';
     ELSE
        memheader <= '0';
     END IF;
  END IF;
END PROCESS p_memheader;
--   now decode receive resp. transmit header
--   MEMTCBHEADER + MEMRFDHEADER
p_memtcbrfd : PROCESS (resetb,ah,memheader)
BEGIN  --  PROCESS p_memtcbrfd
  IF (resetb='0') THEN
     memtcbheader <= '0';
     memrfdheader <= '0';
```

```
  ELSE
    IF (memheader='1') THEN
       IF (ah(15 downto 12)="0000") THEN
         memtcbheader <= '1';
         memrfdheader <= '0';
       ELSE
         memtcbheader <= '0';
         memrfdheader <= '1';
       END IF;
    ELSE
       memtcbheader <= '0';
       memrfdheader <= '0';
    END IF;
  END IF;
END PROCESS p_memtcbrfd;



----  ========================================================================
----  transmit packet specialities
----  ========================================================================
--   generate a transmit timestamp signal (low active) -ttsxmtb-
--   TTSXMTB
p_ttsxmtb : PROCESS (memaccess,memtcbheader,resetb)
BEGIN  --  PROCESS p_ttsxmtb
  IF (resetb='0') THEN
    ttsxmtb <= '1';
  ELSE
     IF (memaccess='1' AND memtcbheader='1') THEN
        IF (writeb='1' AND addr(5 downto 1)="01010"
           AND (dsb(1 downto 0)="11" OR dsb(1 downto 0)="00"
           OR dsb(1 downto 0)="10")) THEN
                 ttsxmtb <= '0';
        END IF;
     ELSE
      ttsxmtb <= '1';
     END IF;
  END IF;                                  --  trigger a transmit timestamp
END PROCESS p_ttsxmtb;


--  ========================================================================
--   re-map the data/address bus next byte and chip selects from
--   the memory to the UTCSU register MSXMT 0x..18  => 0x080148
--   MEMMSXMT
--  ========================================================================
p_memmsxmt : PROCESS (memaccess,memtcbheader,resetb)
BEGIN  --  PROCESS p_memmsxmt
  IF (resetb='0') THEN
    memmsxmt <= '0';
  ELSE
     IF (memaccess='1' AND (memtcbheader='1' AND writeb='1'
         AND addr(5 downto 1)="01100")) THEN
        memmsxmt <= '1';
     ELSE
        memmsxmt <= '0';
     END IF;
  END IF;                                  --  select UTCSU register MSXMT
END PROCESS p_memmsxmt;                    --  instead of this memory address

--  ========================================================================
--   re-map the data/address bus next byte and chip selects from
--   the memory to the UTCSU register TSXMT 0x..1C  => 0x08014C
--   MEMTSXMT
--  ========================================================================
p_memtsxmt : PROCESS (memaccess,memtcbheader,resetb)
BEGIN  --  PROCESS p_memtsxmt
  IF (resetb='0') THEN
    memtsxmt <= '0';
  ELSE
    IF (memaccess='1' AND (memtcbheader='1' AND writeb='1'
        AND addr(5 downto 1)="01110")) THEN
         memtsxmt <= '1';
    ELSE
      memtsxmt <= '0';
    END IF;
  END IF;                                  --   select UTCSU register TSXMT
END PROCESS p_memtsxmt;                    --  instead of this memory address

--  ========================================================================
--   re-map the data/address bus next byte and chip selects from
--   the memory to the UTCSU register ACCXMT 0x..20  => 0x080120
--   MEMACCXMT
--  ========================================================================
p_memaccxmt: PROCESS (memaccess,memtcbheader,resetb)
BEGIN  --  PROCESS p_memaccxmt
  IF (resetb='0') THEN
    memaccxmt <= '0';
  ELSE
     IF (memaccess='1' AND (memtcbheader='1' AND writeb='1'
         AND addr(5 downto 1)="10000")) THEN
          memaccxmt <= '1';
     ELSE
```

```
          memaccxmt <= '0';
     END IF;
  END IF;                                 --  select UTCSU register ACCXMT
END PROCESS p_memaccxmt;                  --  instead of this memory address

-------------------------------------------------------------------------------
--  address re-mapping for transparent timestamp insertion into the
--  outgoing packet
-------------------------------------------------------------------------------
p_a4mt: PROCESS (memmsxmt,memtsxmt,addr)
BEGIN  --  PROCESS p_a4mt
  IF (memmsxmt='1' OR memtsxmt='1') THEN
    a4mt <= '0';
  ELSE
    a4mt <= addr(4);
  END IF;                                   --  in multiplexer for address
END PROCESS p_a4mt;                         --  re-mapping of a4

p_a4a: PROCESS (memaccxmt,addr)
BEGIN  --  PROCESS p_a4a
  IF (memaccxmt='1') THEN
    a4a <= '1';
  ELSE
    a4a <= addr(4);
  END IF;                                   --  in multiplexer for address
END PROCESS p_a4a;                          --  re-mapping of a4

p_a4: PROCESS (memaccxmt,a4a,a4mt)
BEGIN  --  PROCESS p_a4
  IF (memaccxmt='1') THEN
    a(4) <= a4a;
  ELSE
    a(4) <= a4mt;
  END IF;                                   --  out multiplexer for address
END PROCESS p_a4;                           --  re-mapping of a4

p_a6: PROCESS (memtsxmt,memmsxmt,memaccxmt,addr)
BEGIN  --  PROCESS p_a6
  IF (memtsxmt='1' OR memmsxmt='1' OR memaccxmt='1') THEN
    a(6) <= '1';
  ELSE
    a(6) <= addr(6);
  END IF;                                   --  multiplexer for address
END PROCESS p_a6;                           --  re-mapping of a6

p_a8: PROCESS (memtsxmt,memmsxmt,memaccxmt,ah)
BEGIN  --  PROCESS p_a8
  IF (memtsxmt='1' OR memmsxmt='1' OR memaccxmt='1') THEN
    a(8) <= '1';
  ELSE
    a(8) <= ah(8);
  END IF;                                   --  multiplexer for address
END PROCESS p_a8;                           --  re-mapping of a8


p_a19: PROCESS (memtsxmt,memmsxmt,memaccxmt,ah)
BEGIN  --  PROCESS p_a19
  IF (memtsxmt='1' OR memmsxmt='1' OR memaccxmt='1') THEN
    a(19) <= '1';
  ELSE
    a(19) <= ah(19);
  END IF;                                   --  multiplexer for address
END PROCESS p_a19;                          --  re-mapping of a19


p_a7: PROCESS (memtsxmt,memmsxmt,memaccxmt,addr)
BEGIN  --  PROCESS p_a7
  IF (memtsxmt='1' OR memmsxmt='1' OR memaccxmt='1') THEN
    a(7) <= '0';
  ELSE
    a(7) <= addr(7);
  END IF;                                   --  multiplexer for address
END PROCESS p_a7;                           --  re-mapping of a7

p_a5: PROCESS (memaccxmt,addr)
BEGIN  --  PROCESS p_a5
  IF (memaccxmt='1') THEN
    a(5) <= '1';
  ELSE
    a(5) <= addr(5);
  END IF;                                   --  out multiplexer for address
END PROCESS p_a5;                           --  re-mapping of a5


--  ===========================================================================
--  receive packet specialities
--   generate a receive timestamp -ttsrcvb- and sample the receive base address
--   into a dedicated register -tadl-
--   TTSRCVB
--  ===========================================================================
p_ttsrcvb_tadl: PROCESS (memaccess,memrfdheader,resetb)
BEGIN  --  PROCESS p_ttsrcvb_tadl
  IF (resetb='0') THEN
```

```
    ttsrcvb <= '1';
    tadl <= '0';
  ELSE
    IF (memaccess='1' AND (memrfdheader='1' AND writeb='0'
        AND addr(5 downto 1)="01110"
        AND (dsb(1 downto 0)="11" OR dsb(1 downto 0)="00"
        OR dsb(1 downto 0)="10"))) THEN
          ttsrcvb <= '0';
          tadl <= '1';
    ELSE
      ttsrcvb <= '1';
      tadl <= '0';
    END IF;
  END IF;                               --  trigger a receive timestamp and
END PROCESS p_ttsrcvb_tadl;             --  store the base address of the
                                        --  RFD-Header in an intermediate
                                        --  register

--  ========================================================================
--   register set for the base address of the Receive Header
--   the following process stores the incoming address
--   TADL
--  ========================================================================
p_rfdbase: PROCESS (tadl,resetb)
BEGIN  --  PROCESS p_rfdbase
  IF (resetb='0') THEN
    rfdbase <= "0000000000000";
  ELSE
     IF (tadl='1' AND tadl'event) THEN
        rfdbase(2 downto 0) <= addr(7 downto 5);
        rfdbase(12 downto 3) <= ah(17 downto 8);
     END IF;
  END IF;                                 --  save the RFD-base address
END PROCESS p_rfdbase;

--  ========================================================================
--   precsmemb is activated (low active) whenever a memory address is decoded
--   PRECSMEMB
--  ========================================================================
p_precsmemb: PROCESS (memaccess,memmsxmt,memtsxmt,memaccxmt,resetb)
BEGIN  --  PROCESS p_precsmemb
  IF (resetb='0') THEN
    precsmemb <= '1';
  ELSE
    IF (memaccess='1' AND (memmsxmt='0' AND memtsxmt='0'
        AND memaccxmt='0' AND ah(19)='0')) THEN
        precsmemb <= '0';
    ELSE
      precsmemb <= '1';
    END IF;
  END IF;                                 --  select the MA-Modul Memory
END PROCESS p_precsmemb;


----  ========================================================================
----   generate the byte and chip select signals for the memory chips
----   for both lower and higher bank
----   CSMEML/HB + BL/HEB
----  ========================================================================
p_csmemb: PROCESS (precsmemb,resetb)
BEGIN  --  PROCESS p_
  IF (resetb='0') THEN
    tcsmemhb <= '1';
    tcsmemlb <= '1';
    bheb <= '1';
    bleb <= '1';
  ELSE
    IF (precsmemb='0') THEN
      IF (dsb(1 downto 0)="11") THEN
        tcsmemhb <= '0';
        tcsmemlb <= '0';
        bheb <= '0';
        bleb <= '0';  --  32 bit access
      ELSIF (dsb(1 downto 0)="00" AND addr(1)='0') THEN
        tcsmemhb <= '0';
        tcsmemlb <= '1';
        bheb <= '0';
        bleb <= '0';  --  16 bit access - higer part
      ELSIF (dsb(1 downto 0)="00" AND addr(1)='1') THEN
        tcsmemhb <= '1';
        tcsmemlb <= '0';
        bheb <= '0';
        bleb <= '0';  --  16 bit access - lower part
      ELSIF (dsb(1 downto 0)="10" AND addr(1)='1') THEN
        tcsmemhb <= '0';
        tcsmemlb <= '1';
        bheb <= '1';
        bleb <= '0';  --  8bit access 1
      ELSIF (dsb(1 downto 0)="01" AND addr(1)='1') THEN
        tcsmemhb <= '0';
        tcsmemlb <= '1';
        bheb <= '0';
        bleb <= '1';  --  8bit access 2
```

```
      ELSIF (dsb(1 downto 0)="10" AND addr(1)='0') THEN
        tcsmemhb <= '1';
        tcsmemlb <= '0';
        bheb <= '1';
        bleb <= '0';  --  8bit access 3
      ELSIF (dsb(1 downto 0)="01" AND addr(1)='0') THEN
        tcsmemhb <= '1';
        tcsmemlb <= '0';
        bheb <= '0';
        bleb <= '1';  --  8bit access 4
      ELSE
        tcsmemhb <= '1';
        tcsmemlb <= '1';
        bheb <= '1';
        bleb <= '1';
      END IF;
    ELSE
        tcsmemhb <= '1';
        tcsmemlb <= '1';
        bheb <= '1';
        bleb <= '1';
    END IF;
  END IF;
END PROCESS p_csmemb;

--   the read/write signal for the memory - direct driven by the writeb
--   signal of the MA-Modul interface
--
web <= writeb;                          --  read/write to the Memory



-- ========================================================================
-- UTCSU access
--   the precsutcsub becomes active (low) whenever either a UTCSU access is
--   decoded or the memory re-mapping logic is acvtivated
-- PRECSUTCSUB
-- ========================================================================
enaprecsutcsub <= '1' WHEN (ah(19)='1' OR (memmsxmt='1'
                            OR memtsxmt='1' OR memaccxmt='1')) ELSE
                  '0';

p_precsutcsub: PROCESS (memaccess,enaprecsutcsub,resetb)
BEGIN  -- PROCESS p_precsutcsub
  IF (resetb='0') THEN
    precsutcsub <= '1';
  ELSE
    IF (memaccess='1' AND (enaprecsutcsub='1')) THEN
        precsutcsub <= '0';
    ELSE
        precsutcsub <= '1';
    END IF;
  END IF;                        --  select the UTCSU Asic
END PROCESS p_precsutcsub;

--   the read/write signal of the UTCSU was omitted because a lack of pins
--   it can usually be driven by the MA-Modul interface writeb signal
-- rwb <= writeb;                         --  read/write to the UTCSU
--   external hardwired (!!!!!!!!!!) fault on the NTI prototype board


-- ========================================================================
--   generate the byte and chip select signals for the UTCSU
--   CSUTCSUB + BE + BUS16
-- ========================================================================
p_utcsu: PROCESS (precsutcsub,resetb)
BEGIN  -- PROCESS p_utcsu
  IF (resetb='0') THEN
    tcsutcsub <= '1';
    be(3 downto 0) <= "0000";
    bus16 <= '0';
  ELSIF (precsutcsub='0') THEN
      IF (dsb(1 downto 0)="11") THEN
        tcsutcsub <= '0';
        be(3 downto 0) <= "1111";
        bus16 <= '0'; --  32 bit access
      ELSIF (dsb(1 downto 0)="00" AND addr(1)='0') THEN
        tcsutcsub <= '0';
        be(3 downto 0) <= "1100";
        bus16 <= '1'; --  16 bit access - higer part
      ELSIF (dsb(1 downto 0)="00" AND addr(1)='1') THEN
        tcsutcsub <= '0';
        be(3 downto 0) <= "0011";
        bus16 <= '1'; --  16 bit access - lower part
      ELSE
        tcsutcsub <= '1';
        be(3 downto 0) <= "0000";
        bus16 <= '0';
      END IF;
  ELSE
      tcsutcsub <= '1';
      be(3 downto 0) <= "0000";
      bus16 <= '0';
```

```
  END IF;
END PROCESS p_utcsu;



-------------------------------------------------------------------------------
--  ###########################################################################
--                                Global section
--  ###########################################################################
-------------------------------------------------------------------------------
--  ===========================================================================
--  DTACKB - signals the base board that the MA-Module (NTI) is ready to
--  terminate the bus-cycle - this signal is a bit critical due to the fact
--  that it determines the time the units on the NTI are granted to perform
--  their computations - when the UTCSU requires a waitstate this should be
--  inserted within the following statements
--   TNDTACKB
--  ===========================================================================
p_tndtackb: PROCESS (tclk,resetb)
BEGIN  --  PROCESS p_tndtackb
  IF (resetb='0') THEN
          tndtackb <= '1';
  ELSE
     IF (tclk='1' AND tclk'event) THEN
        IF (tcsutcsub='0' AND NOT(writeblatched='1'
            AND addr(7 downto 1)="0000010")
           ) THEN
            tndtackb <= '0';
        ELSE
            tndtackb <= '1';
        END IF;
     END IF;
  END IF;
END PROCESS p_tndtackb;                  --  generate a synchronous signal due
                                         --  to the fact the dtackb needs to
                                         --  be delayed by two clock periods
                                         --  for the UTCSU
-------------------------------------------------------------------------------
--  if required insert an additional clock cycle for the UTCSU
--  TSGETL
-------------------------------------------------------------------------------
p_tydtackb: PROCESS (tclk,resetb)
BEGIN  --  PROCESS p_tydtackb
  IF (resetb='0') THEN
          tydtackb <= '1';
  ELSE
     IF (tclk='1' AND tclk'event) THEN
        IF (tcsutcsub='0' AND writeblatched='1'
            AND addr(7 downto 1)="0000010") THEN
            tydtackb <= '0';
        ELSE
            tydtackb <= '1';
        END IF;
     END IF;
  END IF;
END PROCESS p_tydtackb;
p_tzdtackb: PROCESS (tclk,resetb)
BEGIN  --  PROCESS p_tzdtackb
  IF (resetb='0') THEN
          tzdtackb <= '1';
  ELSE
     IF (tclk='1' AND tclk'event) THEN
        IF (tydtackb='0') THEN
            tzdtackb <= '0';
        ELSE
            tzdtackb <= '1';
        END IF;
     END IF;
  END IF;
END PROCESS p_tzdtackb;

enadtackb <= '1' WHEN ((((tndtackb='0' OR tzdtackb='0'
                       OR tcsmemhb='0' OR tcsmemlb='0'
                       OR ioaccess='1') AND csb='0'))
                       OR iackb='0') ELSE
             '0';


p_twdtackb: PROCESS (tclk,resetb)
BEGIN  --  PROCESS p_twdtackb
  IF (resetb='0') THEN
        twdtackb <= '1';
  ELSIF (tclk='1' AND tclk'event) THEN
          IF (enadtackb='1') THEN
              twdtackb <= '0';
          ELSE
              twdtackb <= '1';
          END IF;
  END IF;
END PROCESS p_twdtackb;                   --  generate a synchronous signal due


p_tdtackb: PROCESS (tclk,resetb)
```

```
BEGIN  --  PROCESS p_twdtackb
  IF (resetb='0') THEN
        tdtackb <= '1';
  ELSIF (tclk='1' AND tclk'event) THEN
          IF (twdtackb='0' AND (csb='0' OR iackb='0')) THEN
              tdtackb <= '0';
          ELSE
              tdtackb <= '1';
          END IF;
  END IF;
END PROCESS p_tdtackb;                   --  generate a synchronous signal due

dtackb <= (tdtackb);


--   when WAITCYCLES are required signal -waitcycle- is in charge for
--   the current implementation does not require this feature
-- waitcycle <= '0';



--  ========================================================================
--  TRANSCEIVER control logic
--  ========================================================================
p_transceiver: PROCESS (csb,resetb)
BEGIN  --  PROCESS p_transceiver
  IF (resetb='0') THEN
    troelb <= '1';
    troehb <= '1';
  ELSE
    IF (dsb(1 downto 0)="11" AND (csb='0')) THEN
       troelb <= '1';
       troehb <= '0';  --  32 bit access mem
    ELSIF (dsb(1 downto 0)="00" AND addr(1)='0' AND (csb='0')) THEN
       troelb <= '1';
       troehb <= '0';  --  16 bit access - higer part mem
    ELSIF (dsb(1 downto 0)="00" AND addr(1)='1' AND (csb='0')) THEN
       troelb <= '1';
       troehb <= '1';  --  16 bit access - lower part mem
    ELSIF (dsb(1 downto 0)="10" AND addr(1)='1' AND (csb='0')) THEN
       troelb <= '0';
       troehb <= '1';  --  8bit access 1
    ELSIF (dsb(1 downto 0)="01" AND addr(1)='1' AND (csb='0')) THEN
       troelb <= '0';
       troehb <= '1';  --  8bit access 2
    ELSIF (dsb(1 downto 0)="10" AND addr(1)='0' AND (csb='0')) THEN
       troelb <= '1';
       troehb <= '1';  --  8bit access 3
    ELSIF (dsb(1 downto 0)="01" AND addr(1)='0' AND (csb='0')) THEN
       troelb <= '1';
       troehb <= '1';  --  8bit access 4
    ELSE
       troelb <= '1';
       troehb <= '1';
    END IF;
  END IF;
END PROCESS p_transceiver;


END behavioral;
```

C Schematics of the NTI prototype