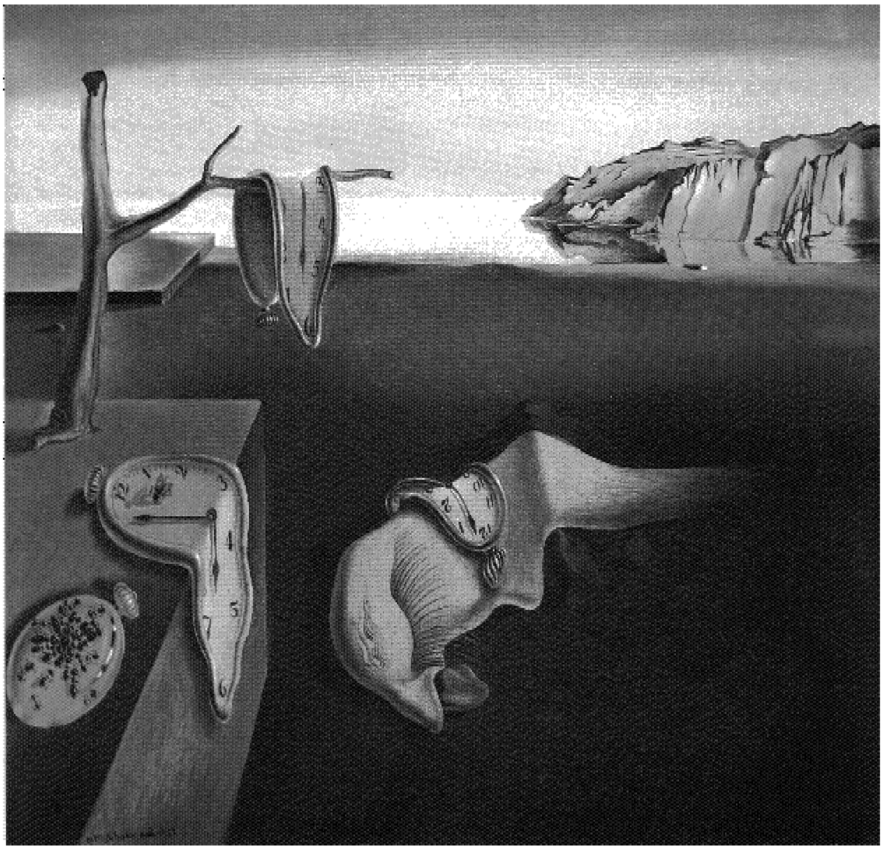
Institut für Automation
Abt. für Automatisierungssysteme

Technische Universität Wien

Projektbericht Nr. 183/1-87
January 1999

Network Time Interface User Manual

Thomas Mandl and Herbert Nachtnebel and Ulrich Schmid



Salvador Dali, "Die Beständigkeit der Erinnerung"

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Datenblatt

Network Time Interface Modul (NTI)

Das Network Time Interface (NTI) MA-Modul unterstützt die hochgenaue Zeitverteilung in LAN-basierenden verteilten Systemen. Es stellt alle Hardware-Funktionen zur Verfügung, die für die interne und externe fehlertolerante Uhrensynchronisation mit μs-Genauigkeit erforderlich sind. Ergänzt durch eine beliebige CPU oder Microcontroller zur Ausführung des Uhrensynchronisationsalgorithmus kann die NTI zusammen mit fast allen Netzwerk-Controllern und GPS-Zeitempfängern eingesetzt werden.

Mögliche Anwendungsgebiete sind verteilte Automatisierungssysteme für schnelle Prozesse, hochauflösende zeitliche Erfassung räumlich weitentfernter Ereignisse und High-Performance Video/Multimedia Stream-Synchronisation.

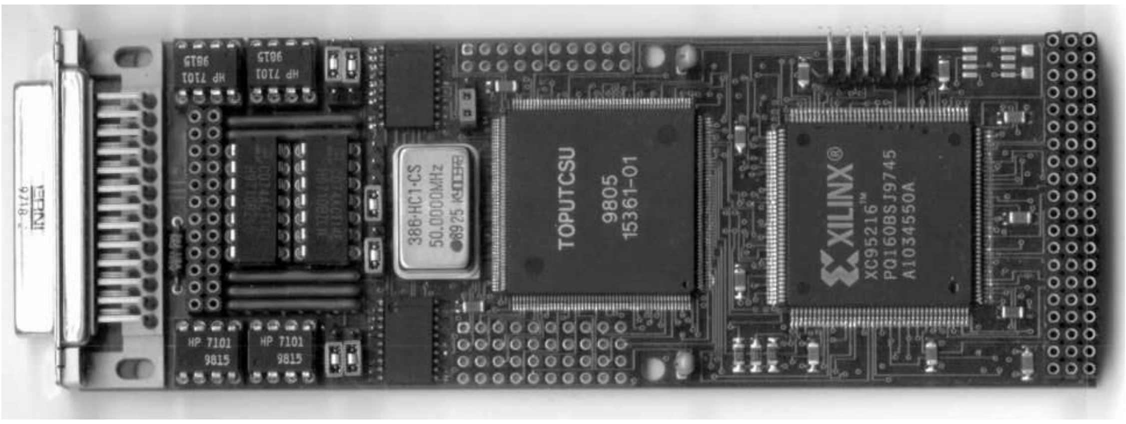


Abbildung 0.1: Network Time Interface (NTI) MA-Modul

Features:
- Stand+Gang-adjustierbare lokale Uhr
- 56-bit NTP-Zeitformat mit 60 ns Auflösung
- Taktgenerierung mit DIP-OCXO oder TCXO mit $f_{osc} \in \{1, \ldots, 25\}$ MHz
- Online-Schranke für Synchronisationsgenauigkeit
- 512 KB SRAM für Netzwerk-Pakete
- optogetrennte/gepufferte Ein-/Ausgänge

MA-Modul Zugriffe:
- A08, D16 (MA I/O-Space)
- A24, D32, D16, INTA (MA Memory-Space)

Externe Anschlüsse:
- 25-poliger D-Sub Stecker (female)
- 24-poliger Base-Board Connector

TTL Eingänge/Ausgänge:
- 1pps und Statuseingänge für 3 GPS-Empfänger (einer via Optokoppler)
- 21 Zeitstempel-Eingänge (einer via Optokoppler)
- 2 externe Takt-Eingänge
- Eingänge für Hardware-Snapshot und synchronen Start
- `+5V`-Versorgungsausgang mit max. 62mA
- 1 zeit-programmierbarer OC-Ausgang

Software-Support:
- pSOS^{+m}-Treiber

Kapitel 1

Projekt SynUTC

Die vorliegende Diplomarbeit entstand im Rahmen des Projekts SynUTC[1] (*Synchronized Universal Time Coordinated for Distributed Fault-Tolerant Real-Time Systems*), das der Fragestellung gewidmet ist, wie – ohne dedizierte GPS-Empfänger oder Zusatzverkabelung heranzuziehen – die Knoten eines netzwerkgekoppelten, fehlertoleranten Rechnersystems mit hochgenau untereinander und auf Weltzeit synchronisierten lokalen Uhren ausgerüstet werden können. Gedacht als eine Art User-Manual für unser *Network Time Interface* MA-Modul (NTI) dokumentiert sie das Endergebnis der Hardware-Entwicklung im Projekt SynUTC, die bis jetzt nur teilweise in Form zahlreicher Einzelarbeiten [SS95], [SSHL97] [HLS96], [Sch97c], [HSS98], [NKC+98], [SN99], [SKM+99] vorliegt. Zusammen mit der Detailbeschreibung des UTCSU-ASICS [Loy96] enhält diese Diplomarbeit somit alle Informationen, die zur Programmierung und Verwendung der NTI notwendig sind.

1.1 Übersicht

Anders als bei herkömmlichen Verfahren, die üblicherweise auf einer zentralen Uhr basieren, sind in SynUTC alle Rechner-Knoten mit einer eigenen lokalen Uhr ausgerüstet. Diese werden allein durch den Austausch von Synchronisationsnachrichten über das verbindende Datennetzwerk untereinander auf μs genau synchronisiert. Der (optionale) Bezug zur Weltzeit wird dadurch hergestellt, daß an mindestens einem der Rechner ein GPS Satellitenempfänger mit 1pps-Zeitausgang angeschlossen sein muß.

Die im Rahmen von SynUTC entwickelte Lösung zeichnet sich primär dadurch aus, daß es nicht notwendig ist, jeden Knoten mit einem dedizierten GPS-Empfänger auszurüsten oder Zeitinformation über zusätzliche Verbindungsleitungen zu verteilen. Die verwendeten Algorithmen basieren auf theoretisch wohlfundierten Grundlagen, die in einer Reihe von wissenschaftlichen Arbeiten [Sch95], [SS97], [Sch97d], [Sch97a] usw. geschaffen wurden. Die umfangreichen konzeptuellen und mathematisch-formalen Analysen garantieren folgende Leistungsmerkmale:

- Überbrückung von LAN-Distanzen unter alleiniger Verwendung des vorhandenen Datennetzwerks. Die Eignung eines bestimmten Netzwerks/Communication-Controllers hängt dabei nicht vom deterministischen Anteil der Nachrichtenverzögerungszeit δ, der automatisch gemessen wird, ab, sondern lediglich vom maximalen *Delay-Jitter* $\delta_{\max} - \delta_{\min}$.
- Hohe Auflösung (Granularität) der lokalen Uhren von $2^{-24} \approx 60$ ns.
- Hohe Synchronisationsgenauigkeit der lokalen Uhren zueinander (genannt *Precision*), die linear vom Delay-Jitter des zugrundeliegenden Netzwerks und der verwendeten Taktfrequenz der lokalen Uhren abhängt; im Falle von Ethernet ist problemlos eine worst-case Genauigkeit im 10 μs-Bereich (average-case um Faktor 10 besser) zu erreichen.

[1]Das Projekt SynUTC wurde gefördert vom Österreichischen Fonds zur Förderung der Wissenschaftlichen Forschung (FWF) unter Projektnummer P10244-ÖMA, dem Jubiläumsfonds der OeNB unter Projekt 6454, der Auftragsforschung des BMfWV unter Vertrag Zl.601.577/2-IV/B/9/96 und dem START-Programm Y41-MAT des BMfWV. Nähere Informationen sind unter *http://www.auto.tuwien.ac.at/Projects/SynUTC/* zu finden.

- Im Normalfall hohe Synchronisationsgenauigkeit der lokalen Uhren zur Weltzeit (genannt *Accuracy*), auch bei Knoten ohne[2] GPS-Empfänger. Darüber hinaus Bereitstellung einer zuverlässigen Schranke (= maximale Abweichung) der momentanen Accuracy.
- Exzellente Fehlertoleranz-Eigenschaften, v.a.
  - Uhrensynchronisation wird durch fehlerhafte Rechner/Uhren nicht beeinträchtigt,
  - (längerer) Ausfall oder Fehlverhalten von GPS-Empfängern verschlechtert lediglich die Accuracy, nicht aber die Precision,
  - mehrere Selbsttest- und Datensicherungsmechanismen.
- Möglichkeit des Aufbaus redundanter Systemarchitekturen, v.a.
  - redundante (mehrfache) Netzwerkverbindungen,
  - redundante Ausführung der lokalen Uhren,
  - Anschluß mehrerer GPS-Empfänger.

Für die auf SynUTC aufbauenden Anwendungen sind – neben der Möglichkeit, jederzeit die aktuelle Uhrzeit per Software auslesen zu können – primär die vielfältigen Zeitstempel-Funktionen wichtig: Zum Zeitpunkt der Zustandsänderung eines digitalen Eingangs wird der Stand der lokalen Uhr in ein dediziertes Register gespeichert, wodurch ein solcher Zeitstempel (zeitlich relativ unkritisch, aber natürlich noch vor der nächsten Zustandsänderung) per Software ausgelesen und weiterverarbeitet werden kann. Weiters stehen Möglichkeiten zur Auslösung von Interrupts bzw. eines Pulses an einem digitalen Ausgang bei Erreichen eines bestimmten Zeitpunkts zur Verfügung.

1.2 Intervall-basierende Uhrensynchronisation

Die gesamte, im Rahmen von SynUTC entwickelte, Hard- und Software basiert auf dem sogenannten *Intervall-basierenden Paradigma* [Mar84], in dem die Zeit t auf einem Knoten p nicht wie üblich durch einen punktförmigen *clock value* $C_p(t)$, sondern durch ein *Accuracy-Intervall* $\boldsymbol{A}_p(t) = [C_p(t) - \alpha_p^-(t), C_p(t) + \alpha_p^+(t)]$ repräsentiert wird.

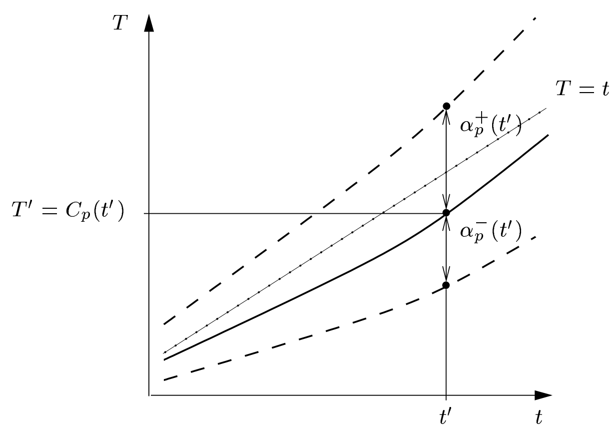


Abbildung 1.1: Accuracy-Intervalle im Projekt SynUTC

Aufgabe der Uhrensynchronisation ist es, die Accuracy-Intervalle so zu steuern, daß für alle fehlerfreien Knoten p und q folgendes garantiert werden kann [SS97]:

(P) *Precision*: $|C_p(t) - C_q(t)| \leq \pi_{\max}$ für ein vorgegebenes $\pi_{\max} \geq 0$,

(A) *Accuracy*: $\boldsymbol{A}_p(t)$, wobei das Genauigkeitsintervall $[-\alpha_p^-, \alpha_p^+]$ minimal sein soll.

[2]Die Accuracy von Knoten mit direktem GPS-Anschluß ist natürlich besser. Übrigens existiert für "isolierte" Konfigurationen (ohne Netzwerk) auch die Möglichkeit, die lokale Uhr mit dem 1 pps-Puls eines direkt angeschlossenen GPS-Empfängers phase-locked zu synchronisieren.

Alle Intervall-basierenden Uhrensynchronisationsalgorithmen basieren auf dem generischen Algorithmus [SS97], in dem jeder Knoten p periodisch (mit Periode P) die folgenden Schritte exekutiert:

1. Wenn $C_p(t) = kP$, $k \geq 1$, wird ein Datenpaket mit $\boldsymbol{A}_p(t)$ zu allen anderen Knoten im System gesendet.

2. Die hereinkommenden Datenpakete werden vorverarbeitet, um die darin enthaltenen Accuracy-Intervalle "vergleichbar" zu machen.

3. Wenn $C_p(t) = kP + \Lambda$ für ein hinreichend großes $\Lambda > 0$, wird eine intervallwertige *Konvergenz-Funktion* auf die empfangenen Intervalle angewendet und die lokale Uhr entsprechend dem Ergebnis korrigiert.

In Schritt 2, in dem die empfangenen Intervalle kompatibel gemacht werden, kommen zwei grundlegende Operationen zur Anwendung: *Delay-Kompensation* wird verwendet, um der nicht völlig konstanten Nachrichtenverzögerung Rechnung zu tragen; dies erfolgt durch eine dem maximalen Jitter entsprechende Vergrößerung der empfangenen Accuracy-Intervalle. *Drift-Kompensation* ist notwendig, um die – natürlich nicht zeitinvarianten – Accuracy-Intervalle mit Hilfe der lokalen Empfängeruhr in der Zeit zu verschieben, ohne die Korrektheitsbedingung zu verletzen. Die Intervalle müssen dabei kontinuierlich, entsprechend der maximalen Drift der Empfängeruhr, vergrößert werden; dies ist natürlich auch während der (langen) Zeit zwischen dem Ende von Schritt 3 und dem nächsten Schritt 1 erforderlich.

In Schritt 3 wird schließlich eine geeignete *Konvergenz-Funktion* auf die Menge der empfangenen und vorverarbeiteten Accuracy-Intervalle angewendet. Ihre Aufgabe ist es, daraus ein neues (kleineres) Intervall für die eigene lokale Uhr zu berechnen, das – systemweit gesehen – die Bedingungen (P) und (A) garantiert. Man beachte, daß der Fehlertoleranzgrad des obigen Algorithmus ausschließlich von der Konvergenz-Funktion bestimmt wird, siehe [Sch97a], [Sch97b].

Der Hauptvorteil der intervall-basierenden Uhrensynchronisation liegt in der Bereitstellung einer lokal verfügbaren on-line Schranke für die momentane Accuracy, was z.B. für die Interaktion in weit verteilten Systemen (wo eine gemeinsame Uhrensynchronisation unmöglich ist) von großer Bedeutung ist. Da Accuracy-Intervalle dynamisch verwaltet werden, sind sie im Mittel sehr klein und daher den statischen worst-case Schranken herkömmlicher Uhrensnchronisationsalgorithmen überlegen. Der Preis dafür ist allerdings die Bereitstellung expliziter Schranken für Clock-Drift und Nachrichtenverzögerungs-Jitter, die entweder *a priori* bekannt sein müssen oder aber dynamisch gemessen – im Falle der Clock-Drift sogar kontrolliert – werden können.

Angesichts unseres Ziels einer Precision/Accuracy im μs-Bereich ist es tatsächlich unumgänglich, eine genaue (Round-Trip-basierende) Messung der Nachrichtenverzögerung durchzuführen und darüberhinaus die Clock-Drift durch einen geeigneten Algorithmus für die *Clock-Ratensynchronisation* zu kontrollieren: Der in [Sch97d], [Sch98] vorgestellte intervall-basierende Algorithmus hat die Aufgabe, die Ganggeschwindigkeiten aller Uhren im System aneinander anzugleichen. Dadurch ist es möglich, die maximale (relative) Clock-Drift klein zu halten, ohne daß dafür extrem stabile Oszillatoren für die lokalen Uhren herangezogen werden müssen.

Kapitel 2

Aufgaben des Network Time Interface Moduls

Dieses Kapitel beschreibt den Aufbau von verteilten Computernetzwerken im Hinblick auf die Problematik der Uhrensynchronisation in solchen Systemen. Im Anschluß an die theoretischen Überlegungen werden die Aufgaben des Network Time Interface Moduls abgesteckt.

2.1 Knoten-Architektur

Wie bereits in der Einleitung der Diplomarbeit erläutert, ist das Forschungsziel des Projekts SynUTC eine hochgenaue interne und externe Uhrensynchronisation in verteilten Echtzeitsystemen, die aus Computerknoten bestehen und über paketorientierte Datennetzwerke kommunizieren. In der Abbildung 2.1 werden die wichtigsten Hardwarekomponenten dargestellt, die für die Uhrensynchronisation notwendig sind.

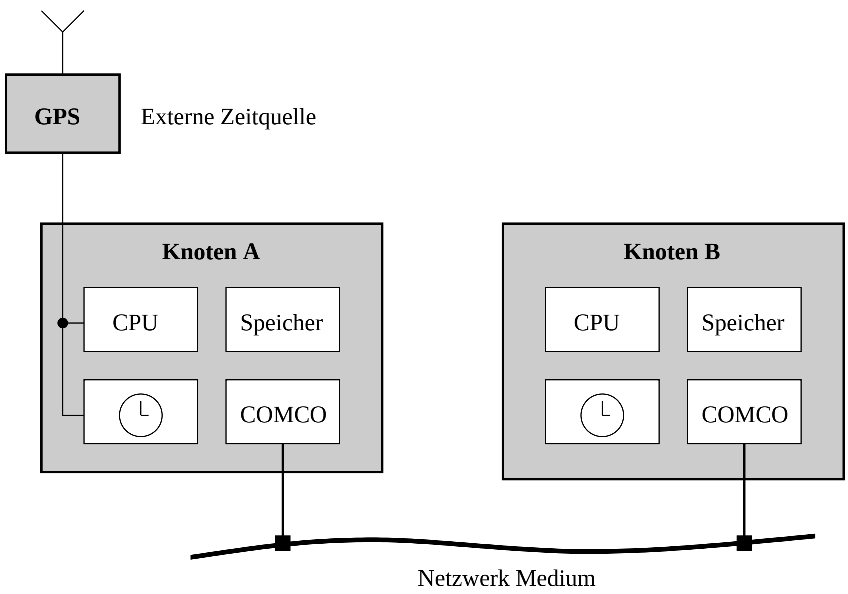


Abbildung 2.1: Zwei Knoten in einem verteilten System

Jeder Knoten muß mit einer Hardware-Uhr (unserem UTCSU-ASIC, siehe Abschnitt 3.3) ausgestattet sein. Eine Central Processing Unit (CPU) wird benötigt um den Softwareteil der Uhrensynchronisationsalgorithmen auszuführen. Der Communication Coprocessor (COMCO) ist für das Senden und Empfangen von CSP's (Clock Synchronisation Packets) verantwortlich, die Kommunikation mit der CPU erfolgt über einen gemeinsamen Speicher (Shared-Memory). Für die externe Synchronisation besteht die Notwendigkeit, daß zumindest ein Knoten mit GPS-Empfängern ausgestattet ist.

2.2 Uhrensynchronisation über Netzwerke

Eine wesentliche Aufgabe bei der hochpräzisen Uhrensynchronisation ist die exakte Zeitstempelung von CSP's (Clock Synchronisation Packets) sowohl beim Sender als auch beim Empfänger. Tatsächlich wird diese Behauptung durch das bekannte Ergebnis von [LL84] untermauert, das besagt, daß n ideale Uhren nicht mit einer kleineren Precision als $\epsilon(1 - 1/n)$ synchronisiert werden können, wenn ϵ die zeitliche Unsichertheit bei der Datenpaketübertragung ist. Welche Schritte bei der Übertragung von Datenpaketen (mit einem DMA-fähigen COMCO) notwendig sind und den Parameter ϵ beeinflußen, wird nun erklärt [HSS98]:

1. Sender-CPU erstellt das Datenpaket im Shared Memory
2. Sender-CPU verständigt den COMCO, daß er Pakete senden soll
3. Sender-COMCO versucht das Netzwerk-Medium zu aquirieren
4. Sender-COMCO liest das Datenpaket aus dem Shared-Memory Bereich und legt die resultierende Bitfolge auf das Medium
5. Empfänger-COMCO liest die Bitfolge vom Medium und schreibt das CSP in seinen Shared-Memory Bereich
6. Empfänger-COMCO teilt der CPU den Empfang eines Datenpaketes durch einen Interrupt mit
7. Empfänger-CPU verarbeitet das CSP

Rein softwarebasierende Ansätze der Uhrensynchronisation generieren die CSP-Zeitstempel beim Senden bzw. Empfangen in den Schritten 1 bzw. 7. Das bedeutet allerdings, daß sich ϵ aus der Mediumzugriffs-Unsicherheit (3→4), einer variablen Übertragungszeit (4→5) und einer nicht konstanten Empfangs-Interrupt Latency (6→7) zusammensetzt. Die erste Unsicherheit kann bei Netzwerken mit gemeinsamen Medium (z.B. Ethernet) beliebig lange dauern. Die tatsächliche Interruptverarbeitung kann z.B. durch Code-Segmente mit abgeschalteten Interrupts undeterministisch lange verzögert werden. Wenigstens können wir bei unserem LAN-basierten Ansatz mit Sicherheit davon ausgehen, daß die Übertragung am Medium (4→5) etwa konstant ist, da wir keine Gatewayknoten mit Warteschlangen-Verzögerungen haben.[1] Daraus ergibt sich eine Sende-/Empfangs-Unsicherheit, die hauptsächlich aus den Schritten (1→4) bzw. (5→7) an der Sender bzw. Empfängerseite entsteht.

Um den Parameter ϵ möglichst klein zu machen, muß die Synchronisationshardware so eng als möglich mit dem Netzwerk gekoppelt werden. Idealerweise sollten die Zeitstempel am Sender bzw. Empfänger genau dann abgezogen werden, wenn beispielsweise das erste Byte auf das Medium gelegt bzw. vom Medium genommen wird. Die dafür benötigte Unterstützung des COMCO's ist allerdings (mit Standardkomponenten) kaum verfügbar, aber siehe [HL98] oder [KO87].

Um existierende Netzwerk-Controller Technologie zu unterstützen, muß eine leicht modifizierte Art der Kopplung gefunden werden. Zu diesem Zweck verwendet unser NTI eine Verbesserung der DMA-basierten Methode, die in [KO87] vorgestellt wurde. Die Kernidee ist es, einen Zeitstempel „On-the-Fly“ in den Speicher, der das CSP beinhaltet, einzufügen, um so die Unsicherheit möglichst zu minimieren. Im Detail bedeutet dies, daß eine modifizierte Adreßdekodierungslogik für den Speicher verwendet wird, die folgende Aufgaben hat:

- Die Erzeugung von Trigger-Signalen, die in dafür vorgesehenen UTCSU-Registern Zeitstempel abziehen, wenn an einem bestimmten Byte eines Transmit-Headers oder Receive-Headers gelesen bzw. geschrieben wird.
- Der so erzeugte Transmit-Timstamp wird transparent in einen bestimmten Teil des Datenpaketes gemappt.

Auf diese Art werden die CSP's genau in den Schritten 4 und 5 mit Zeitstempel versehen. Zu beachten ist noch, daß diese Funktionalität nur dann aktiv ist, wenn der COMCO auf das Shared-Memory zugreift, d.h. CPU-Zugriffe finden ganz normal statt.

[1] Man beachte, daß diese Aussage auch gilt, wenn in einem Multi-Hop Netzwerk alle Gateway-Knoten ebenfalls mit unserem Zeitstempel-Mechanismus (siehe Abschnitt 2.3) ausgestattet sind.

2.3 Datenpakete mit Zeitstempel versehen

Um diesen Vorgang anschaulich zu präsentieren, wird in der Abbildung 2.2 ein Beispiel gezeigt. Weitere Szenarien können in [SS95] nachgelesen werden.

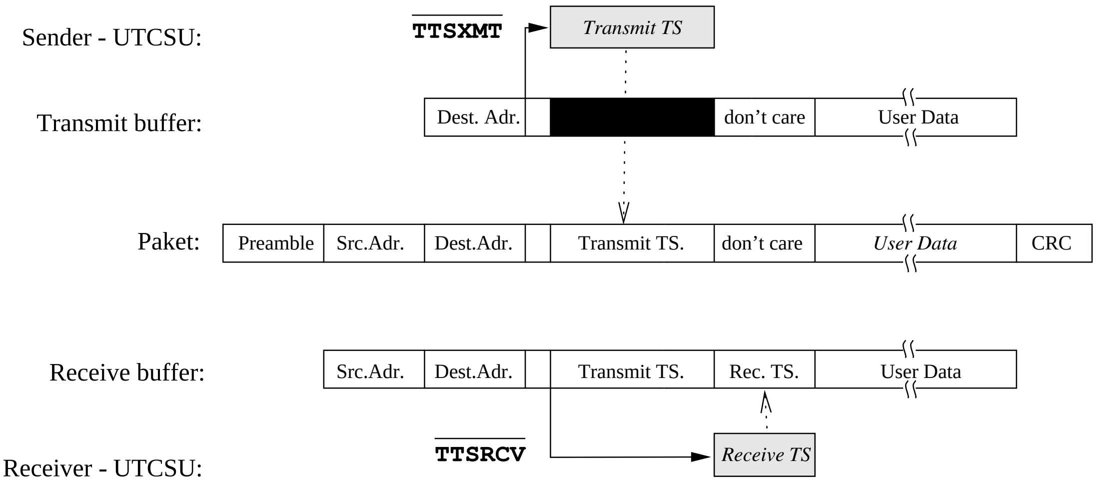


Abbildung 2.2: Einfügen von Zeitstempeln in Datenpakete

Um ein Datenpaket zu verschicken, muß die CPU zuerst das Datenpaket im Shared-Memory erstellen und dann den COMCO verständigen, daß ein CSP zu versenden ist. Immer wenn der COMCO Daten vom Transmit-Buffer des zu sendenden CSP's holt, liest er über eine bestimmte Adresse, an der die Trigger-Logik einen Puls am Signal $\overline{\texttt{TTSXMT}}$ erzeugt. Dieses Signal bewirkt, daß in der Sender-UTCSU ein Transmit/Accuracy-Zeitstempel abgezogen wird, d.h. die aktuelle Zeit/Accuracy wird in das Sample-Register kopiert. Der Inhalt der Register wird von der Dekodierlogik transparent in einen bestimmten Bereich des Datenpakets geschrieben. Wenn der Sender-COMCO die Speicheradressen des Zeitstempels liest, wird nicht vom Speicher, sondern von dem Sample-Register der Sender-UTCSU gelesen. Erwähnenswert ist die Tatsache, daß die Trigger-Adresse und die Mapping-Adresse unterschiedlich sein können.

Wenn das Datenpaket beim Empfänger eintrifft, so wird ebenfalls ein Trigger-Signal $\overline{\texttt{TTSRCV}}$ ausgelöst, wenn der Empfänger-COMCO an eine bestimmten Stelle im Speicher schreibt. In der Empfänger-UTCSU wird ein Receive-Zeitstempel abgezogen. Die Empfänger-CPU wird mit einem Interrupt verständigt, daß ein Zeitstempel abgezogen wurde. Dieser muß von der CPU an die dafür vorgesehene Adresse im Datenpaket geschrieben werden, bevor das nächste Paket empfangen wird. Eine detailliertere Beschreibung wird im Abschnitt 3.4 gegeben.

2.4 Geeignete Netzwerk-Controller

Unsere Methode der engen Kopplung zwischen Synchronisationshardware und Netzwerk funktioniert mit jedem Netzwerk-Controller, der direkten Zugriff auf den Speicher mit den CSP's hat. Geeignete Bausteine gibt es für viele verschiedene Netzwerktypen, von Feldbussen wie dem Profibus über Ethernet bis hin zu extrem schnellen Übertragungsnetzen wie ATM oder FDDI. COMCO's, die den Speicher für die Datenpakete am Chip integriert haben, wie es etwa bei CAN-Controllern üblich ist, sind für diese Methode des Zeitstempelns nicht geeignet, solange sie nicht die notwendigen Trigger-Signale exportieren.

Um abschätzen zu können, wie weit die Sende- und Empfangsdauer der verschiedensten Netzwerk-Prozessoren variieren, muß natürlich eine experimentelle Evaluation durchgeführt werden. Das wird vor allem durch die Tatsache begründet, daß die meistens COMCO's interne FIFO's verwenden, die Unsicherheiten in der Zeitdauer vom Holen eines Bytes des Speichers bis zum Schreiben auf den Übertragungskanal implizieren. Obwohl durch eine „Feineinstellung" der Triggerposition der Transmit-/Receive-Zeitstempel versucht werden kann, solche Probleme zu reduzieren, ist es trotzdem fast unmöglich, ohne genaue Messungen ein optimale Parameter-Setting zu erreichen. Zu bemerken wäre noch, daß dieser Aufwand nicht getrieben werden müßte, wenn der COMCO über Signale verfügen würde, die bei der tatsächlichen Übertragung bestimmter Bytes der CSP's aktiv würden.

2.5 Resultierende Aufgaben des NTI-Moduls

Um die theoretischen Überlegungen der vorangegangenen Abschnitte tatsächlich umsetzen zu können, bedarf es zunächst einer verbalen Beschreibung der Aufgaben, die das Network Time Interface Modul erfüllen muß.

Die wichtigsten Bestandteile des NTI sind die UTCSU [Loy96], die den hardwaremäßigen Teil der Uhrensynchronisation bereitstellt, und der Speicher, der als Kommunikationsschnittstelle zwischen der CPU und einem Communication Coprocessor (COMCO) fungiert. Weder die CPU noch der COMCO sind Komponenten des Network Time Interface Moduls, sondern beide befinden sich auf einem Standard-Prozessorboard und greifen über ein Bussystem auf die UTCSU und den Speicher zu.

Die Aufgaben des NTI-Moduls können folgendermaßen zusammengefaßt werden:

- Die *UTCSU* muß über eine geeignete Schnittstelle zugreifbar sein. Dieses Interface muß *Interrupts* bereitstellen, da die UTCSU die Möglichkeit hat, den Synchronisationsalgorithmen bzw. der Applikation verschiedenste Ereignisse (wie z.B. den Empfang und die Zeitstempelung eines CSP-Pakets) via Interrupt mitzuteilen.

- Der *Speicher* muß sowohl von der CPU als auch vom COMCO verwendbar sein. Eine entsprechende *zusätzliche Logik* muß dafür sorgen, daß nur bei COMCO-Zugriffen auf die Receive- bzw. Transmit-Header die Trigger-Signale $\overline{\texttt{TTSRCV}}$ bzw. $\overline{\texttt{TTSXMT}}$ erzeugt und in die UTCSU geführt werden. Darüber hinaus muß die Steuerlogik dafür sorgen, daß die Zeitinformationen der UTCSU transparent in die Clock Synchronisation Packets (CSP) eingefügt werden.

- Zur Unterstützung der externen Uhrensynchronisation müssen die Ausgangssignale von drei *GPS-Empfängern* an die korrespondierenden UTCSU-Eingänge geführt werden.

- Alle Trigger-Eingänge der UTCSU sollen über Stecker von außen zugänglich sein. Entsprechende Pufferung bzw. Optokopplung der Signale muß vorgesehen werden, um die UTCSU vor ungültigen Signalpegeln zu schützen.

- Um die hohe Synchronisationsgenauigkeit realisieren zu können, müssen Spezialquarze (OCXO oder TCXO) mit niedriger Drift verwendet werden.

Der Entwurf und die genaue Beschreibung des Network Time Interface Moduls können im nächsten Kapitel (Kapitel 3) nachgelesen werden.

Kapitel 3

Entwurf und Funktionsbeschreibung des NTI-Moduls

Beim Entwurf des Network Time Interface Moduls war es für uns ganz besonders wichtig, nur so wenig wie möglich selbst zu entwickeln und weitestgehend auf Standardkomponenten zurückzugreifen. Durch die Erfahrung von anderen Projekten (siehe [Sch94]) entschieden wir uns für ein VME-Bus basierendes System (siehe [Ass97] und [DIN85]), auf das Piggy-Pack Module aufgesteckt werden können. Der Vorteil dabei war, daß wir kein Board mit der komplizierten VME-Schnittstelle sondern „nur“ ein kleines und damit auch deutlich billigeres Submodul entwickeln mußten.

In den letzten Jahren wurden verschiedenste Standards für Aufsteckmodule entwickelt, z.B. Industry Pack Module (IP), M-Module, FLXibus-Module, etc. Leider gibt es keinen Standard, der für alle Anwendungen optimal ist (siehe auch [Bau94]), daher muß man sich je nach gewünschter Leistungsfähigkeit, Flexibilität, Größe, Robustheit, Kosten, usw. für einen Mezzanin-Standard entscheiden.

Die wichtigsten Anforderungen bei der Entwicklung des NTI-Moduls waren (siehe [HLS96] für Details):

- **Hohe Leistungsfähigkeit**: Da sowohl die CPU als auch der COMCO Speicherzugriffe auf das NTI machen, sind sehr schnelle Zugriffe mit 32 Bit Datenbreite notwendig.
- **Einfache Schnittstelle** wegen unserer unzureichenden Erfahrung mit der Entwicklung von komplizierten Interfaces.
- **Größe:** Das Modul sollte so klein als möglich sein, jedoch genügend Platz für die UTCSU, das CPLD und alle anderen Bauteile bieten.
- **Carrier Boards** sollten im Stande sein, mehrere NTI's zu tragen, damit die Kosten des Evaluationssystems reduziert werden.
- **Stecker:** Für die Schnittstellen zu den GPS-Empfängern und den Applikationen ist ein Front-Panel Stecker notwendig.

Das *Network Time Interface* wurde als MA-Modul (siehe [Man96]) konstruiert (Bilder siehe Kapitel und 6.9), da diese Mezzanin-Module eine offene, einfache, robuste und genormte Schnittstelle zu VME-Prozessor Boards haben und die oben gestellten Anforderungen am besten erfüllten.

Im Gegensatz zu M-Modulen haben MA-Module einen 32 Bit Datenbus statt eines 16 Bit breiten. Der Adreßbereich ist zweigeteilt, über die normale M-Modul Schnittstelle wird der 256 Byte große *I/O-Space* angesprochen, durch multiplexen des MA-Datenbusses kann ein 16 MByte großer Adreßraum, der sogenannte *Memory Space*, adressiert werden. Die Datenkommunikation erfolgt über eine asynchrone Schnittstelle, sodaß Zugriffe auf das Modul nur mit einem Acknowledge-Signal bestätigt werden müssen. Mit weiteren Signalen können vektorisierte Interrupts bzw. DMA-Transfers einfach ausgeführt werden.

Das $146 \times 53mm$ (single) M-Modul bietet über einen 25-poligen D-Sub Stecker (FPC) und einen 24-poligen Leistenstecker (CPC) periphere I/O-Anschlüsse. Die Verbindung zum Carrier-Board wird mit einem 60-poligen Leistenstecker (MA-Stecker) realisiert. Zur Kommunikation mehrerer M-Module untereinander wird der Intermodule Port Connector (IPC) verwendet.

Die folgenden Abschnitte erklären den Aufbau und die Arbeitsweise des NTI-Moduls, die beiden Adreßräume *I/O-Space* und *Memory-Space*, die UTCSU und die Interruptbehandlung.

3.1 Blockdiagramm

Die Abbildung 3.1 zeigt die wichtigsten Komponenten des *Network Time Interfaces*. Alle Bauteile (bzw. Bauteilgruppen) sind als dunkle Rechtecke gezeichnet, die Stecker sind als helle und abgerundete Rechtecke dargestellt. Die dicken Pfeile bedeuten, daß Signalgruppen von einem Bauteil (stumpfes Pfeilende) ein anderes Bauteil steuern (Pfeilspitze). So z.B. werden die Signale von den Puffern zur UTCSU geliefert, jedoch nicht umgekehrt. Die dünnen Pfeile bedeuten ein einziges Signal, und zwar ist das die Taktversorgung.

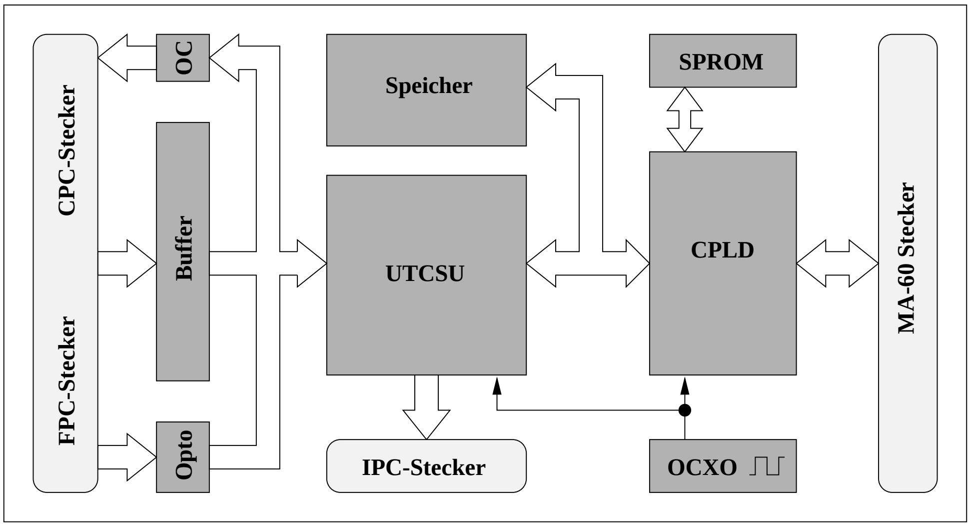


Abbildung 3.1: Blockdiagramm des NTI MA-Moduls

Das „Herz“ des NTI ist der UTCSU-ASIC [Loy96], [SSHL97], der den größten Teil der Hardwareunterstützung für die Uhrensynchronisation beinhaltet. Eine ausführliche funktionelle Beschreibung der UTCSU wird im Kapitel 3.3 gegeben, die technischen Details werden im Abschnitt 6.10.2 präsentiert.

Die gesamte Dekodier- und Steuerlogik des NTI ist in einem in-circuit programmierbaren CPLD (Complex Programmable Logic Device) [NKC+98] realisiert. Das CPLD bindet die UTCSU und den Speicher an das MA-Interface an, es generiert vektorisierte Interrupts, erzeugt das Acknowledgesignal zur Buszyklusterminierung und gestattet den Zugriff auf die Modulidentifikation, die in einem 16×16 Bit großen SPROM (Serial EPROM) gespeichert ist. Der Aufbau des CPLD's ist im Abschnitt 6.10.1 beschrieben. Die Programmierung und der VHDL-Code können im Kapitel A im Anhang nachgelesen werden.

Der Speicher besteht aus vier $64k \times 16$ Bit SRAM-Chips, die als Kontroll- und Datenschnittstelle zwischen der CPU und dem COMCO dienen.

Die Taktung des Moduls erfolgt über einen hochpräzisen OCXO (Ovenized Controlled Quartz Crystall Oscillator) oder einen TCXO (Temperature Compensated Quarz Crystal Oscillator). Darüber hinaus besteht die Möglichkeit, das Board mit dem 10MHz-Ausgang eines High-End GPS-Empfängers zu takten. Für Testzwecke ist es auch möglich, das Modul mit normalen Quarzen mit TTL-Ausgang zu betreiben, für die Uhrensynchronisation müssen jedoch Quarze mit sehr kleiner Drift verwendet werden.

Die Kommunikation des NTI-Moduls mit dem Carrier-Board, die in Abschnitt 3.2 erklärt wird, erfolgt über den 60-poligen MA-Stecker. Alle für die Uhrensynchronisation verfügbaren Eingangs- und Ausgangssignale der UTCSU sind an der Vorderseite des Moduls über den 25-poligen D-Sub Stecker FPC (Front Panel Connector) bzw. über den 24-poligen Leistenstecker CPC (Carrier-Board Plug Connector) zugänglich. Die beiden IPC-Stecker (Intermodule Port Connector) sind für eventuelle zukünftige Erweiterungen, zur unter Umständen notwendigen Verstärkung der Stromversorgung und für den Anschluß modularer GPS-Empfänger ausgelegt.

Alle Eingangssignale werden über sehr schnelle Optokoppler bzw. Buffer geführt, um das Modul vor Überspannungen zu schützen. Die Ausgänge werden von Open-Collector Schaltungen (OC) getrieben.

Die Stecker und die Erklärung der zugehörigen Signale sind im Kapitel 6.4 beschrieben, die wichtigsten technischen Daten der Bauteile werden im Kapitel 6.10 vorgestellt.

3.2 Erklärung der MA-Schnittstelle

Die CPU bzw. der COMCO eines Carrier-Boards können das NTI-Modul über die MA-Schnittstelle ansprechen. Dieses Interface ist im CPLD (Complex Programmable Logic Device) programmiert (siehe dazu Abschnitt 6.10.1 und Kapitel A im Anhang).

Das NTI-Modul wird mit dem 60-poligen MA-Stecker (siehe Abschnitt 6.4.1) auf das Carrier-Board gesteckt. Über diesen Stecker werden alle Steuer-, Adreß-, und Datensignale übertragen. Die wichtigsten Signale für unsere Anwendung sind in Tabelle 3.1 erklärt. Signale, die in der linken Spalte angeführt sind, werden vom Carrier-Board erzeugt (z.B. $\overline{\mathtt{WRITE}}$). Alle Signale, die vom CPLD des NTI's getrieben werden, stehen in der mittleren Spalte (z.B. $\overline{\mathtt{DTACK}}$). Die Datenleitungen `D0 - D31` können sowohl vom Carrier-Board als auch vom NTI getrieben werden.

Carrier Board	NTI	Funktion
`D0 - D31`	`D0 - D31`	Das NTI ist hauptsächlich für 32-Bit Datenzugriffe entworfen, jedoch sind auch 8-Bit und 16-Bit Zugriffe möglich.
`A1 - A7`		Wenn $\overline{\mathtt{AS}}$`='HIGH'` ist, dann werden nur die Leitungen `A1 - A7` zur Adreßierung des 256 Byte großen *I/O-Space* verwendet. Ist $\overline{\mathtt{AS}}$`='LOW'`, dann wird mit `A1-A23` der 16MByte *Memory Space* angesprochen. In diesem Fall müssen die Datenleitungen `D0-D15` mit der fallenden Flanke von $\overline{\mathtt{AS}}$ gespeichert werden, da sie zu diesem Zeitpunkt den Adreßleitungen `A8-A23` entsprechen.
$\overline{\mathtt{WRITE}}$		Bei `'LOW'` schreibt eine Komponente des Carrier-Boards auf das Modul. Ist $\overline{\mathtt{WRITE}}$ hingegen `'HIGH'`, so liest die CPU oder der COMCO vom NTI-Board.
$\overline{\mathtt{AS}}$		Bei `HIGH`→`LOW` Transitionen müssen `D0 - D15` gespeichert werden, da sie den Wert von `A8 - A23` repräsentieren.
$\overline{\mathtt{CS}}$		Diese Leitung signalisiert mit `'LOW'` einen normalen R/W-Zugriff auf das Modul. Bei Lesezugriffen muß das MA-Modul die Datenleitungen solange treiben, bis $\overline{\mathtt{CS}}$ wieder `'HIGH'` wird.
	$\overline{\mathtt{DTACK}}$	Das NTI zeigt dem Carrier Board mit `'LOW'`, daß der Buszyklus terminiert werden kann.
$\overline{\mathtt{DS0}}$ - $\overline{\mathtt{DS2}}$ + `A1`		Werden zum Data Bus Routing verwendet.
$\overline{\mathtt{RESET}}$		Das UTCSU Reset-Signal ist `'HIGH'`-aktiv, darum muß dieses Signal invertiert werden.
	$\overline{\mathtt{IRQ}}$	Mit `'LOW'` signalisiert das MA-Modul einen Interrupt. Die Leitung muß solange auf diesen Pegel getrieben werden, bis das Carrier-Board mit $\overline{\mathtt{IACK}}$ bestätigt.
$\overline{\mathtt{IACK}}$		Während einem Interrupt-Zyklus erfüllt die $\overline{\mathtt{IACK}}$-Leitung die gleiche Funktion wie $\overline{\mathtt{CS}}$ bei normalen Datentransfers. Wenn das Signal `'LOW'` wird, muß das NTI den Interrupt-Vektor an `D0-D7` anlegen. Danach muß der Zugriff mit `'LOW'` an $\overline{\mathtt{DTACK}}$ beendet werden.
	$\overline{\mathtt{DREQ}}$	DMA-Transfer Anforderung vom MA-Modul (wird beim NTI nicht verwendet).
$\overline{\mathtt{DACK}}$		Bestätigung des Carrier Boards für DMA-Zugriffe (wird ebenfalls nicht gebraucht).
`SYSCLOCK`		16MHz Takt, der asymmetrisch sein kann.
`TRIGA, TRIGB`		Benutzerdefinierte Trigger-Signale.

Tabelle 3.1: MA-Signale und deren Funktion

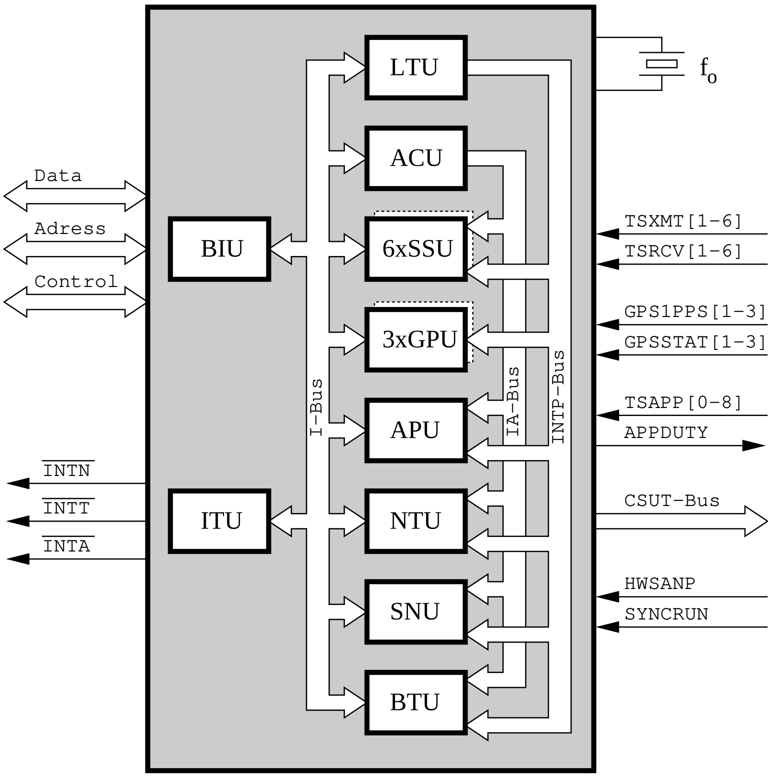
Im wesentlichen benötigen wir I/O-Space Zugriffe, Memory-Space Zugriffe und Interrupt-Anforderungen. Eine genaue Erklärung dieser Zugriffe und alle anderen Funktionen der asynchronen MA-Schnittstelle werden in [Man96] ab Seite 22 mit Zeitdiagrammen dargestellt. Für die Dauer, die Signale getrieben werden dürfen/müssen, gibt es sehr strenge Richtlinien, die unbedingt einzuhalten sind.

3.3 UTCSU - Aufgaben und Kurzbeschreibung

Die UTCSU - mit vollem Namen Universal Time Coordinated Synchronisation Unit - ist der Uhrensynchronisationschip, der die im Abschnitt 1.2 gestellten Anforderungen erfüllt.

In diesem Kapitel wird ein kurzer Ausblick über den Aufbau der UTCSU gegeben. Im Kapitel 6.10.2 sind die wichtigsten Produktionsdaten, das Pinning und eine Tabelle mit allen, nach Funktionsgruppen geordneten, Signalen dargestellt.

Für tiefgründigere Information ist allerdings das Nachlesen in [SSHL97] bzw. in [Loy96] unerläßlich.



Unit	Vollständiger 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	Synchronisation Subnet Unit

Abbildung 3.2: UTCSU Blockdiagramm

Die UTCSU ist, wie man im Blockdiagramm 3.2 sieht, modular aufgebaut, d.h. alle Aufgaben sind in Funktionsgruppen unterteilt, die Units genannt werden. Die Zuordnung der Abkürzungen zu den vollständigen Namen aller Funktionsblöcke ist in der Tabelle neben dem Blockdiagramm alphabetisch geordnet aufgelistet.

Diese Einheiten sind in der UTCSU durch Busse verbunden. Die Busse sind in drei Hauptbusse gegliedert: Der 64 Bit breite INTP-Bus (Internal NTP-Bus) stellt die Zeit, die von der Local Time Unit (LTU) berechnet wird, den anderen Units zur Verfügung. Die Accuracy Unit (ACU) liefert auf dem 32 Bit breiten IA-Bus (Internal Accuracy Bus) die aktuellen Accuracyinformationen. Mit Ausnahme der Bus Interface Unit (BIU) und der Interrupt Unit (ITU) benötigen alle Funktionsgruppen die Zeitinformationen bzw. stellen sie zur Verfügung. Der dritte Bus, im Diagramm mit I-Bus bezeichnet, steuert die Funktionsgruppen.

Die Signale sind jeweils der Funktionsgruppe zugeordnet, der sie im Diagramm am nächsten liegen, also beispielsweise gehören die Signale `HWSNAP` und `SYNCRUN` zur Snapshot Unit (SNU).

In den nächsten Unterkapiteln wird etwas detaillierter erklärt, welche Aufgaben die einzelnen Units bzw. Signale haben.

3.3.1 Local Time Unit (LTU)

Die Local Time Unit (LTU) berechnet die lokale Zeit mit einer Adder Based Clock (ABC). Eine ABC ist ein sehr breiter Addierer (in unserem Fall 91 Bit), der bei jedem Oszillator-Tick einen vorprogrammierten Wert addiert und so die Zeit berechnet. Wie der durch seine extreme Breite und die hohen zeitlichen Anforderungen sehr komplexe Addierer aufgebaut ist, kann in [HL95] nachgelesen werden.

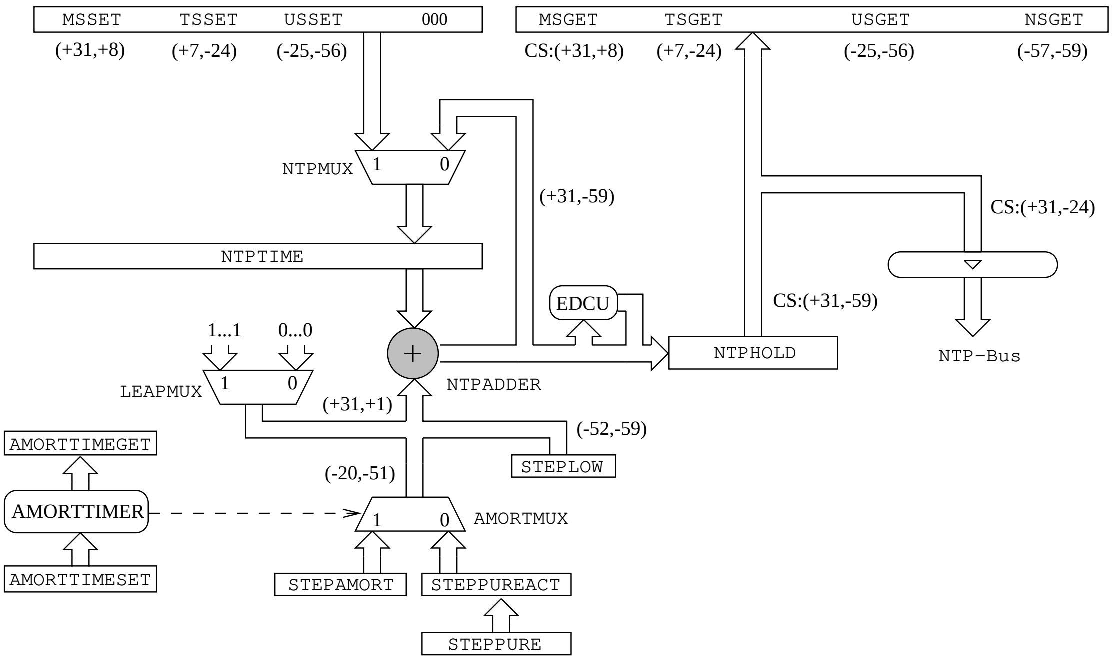


Abbildung 3.3: Local Time Unit der UTCSU

Die Abbildung 3.3 zeigt schematisch den Aufbau der LTU mit dem 91-Bit breiten Addierer NTPADDER im Zentrum. Die Summe wird als positive Rückkopplung des vorherigen Additionswertes im Register NTPTIME gebildet, das die lokale Zeit im NTP-Format `[+31, -59]` beinhaltet. Die Initialisierung erfolgt mit dem Multiplexer NTPMUX, welcher die Vorladeregister `MSSET`, `TSSET` und `USSET` ins NTPTIME-Register lädt.

Während der Pure- bzw. Amortization-Phase wird der Wert, der pro Tick addiert wird und das NTP-Format `[-20, -51]` hat, aus dem Register `STEPPUREACT` bzw. `STEPAMORT` verwendet. Die Granularität wird mit dem Register `STEPLOW`, das einen fixen Wert im Bereich `[-52, -59]` beinhaltet, erweitert, um nicht binäre Quarzfrequenzen genauer darstellen zu können. Mit dem Multiplexer AMORTMUX wird zwischen der Pure-Phase und der Amortization-Phase umgeschaltet. Immer wenn der 32-Bit Counter AMORTTIMER aktiv ist, wird bei jedem Tick der Wert des Registers `STEPAMORT` verwendet, ansonsten wird der Wert von `STEPPUREACT` zur Addition herangezogen. Der Counter kann durch einen Duty-Timer oder direkt über einen dedizierten Registerzugriff gestartet werden. Das Vorladeregister `STEPPURE` ist notwendig, um den Wert für die nächste Pure-Phase zu speichern. Wenn die Amortization-Phase endet, wird der Wert von `STEPPURE` ins Register `STEPPUREACT` geladen.

Damit die UTCSU auch für die nichtchronoskopische Zeitskala UTC verwendet werden kann, muß die Möglichkeit bestehen, Schaltsekunden einzufügen bzw. abzuziehen. Grundsätzlich besteht die Hardware zur Schaltsekundenkorrektur aus einem Multiplexer LEAPMUX, der die obersten 32-Bit des Additionswertes so beeinflußt, daß genau eine Schaltsekunde eingefügt bzw. abgezogen wird. Diese Art der Zeitkorrektur wird über einen Duty-Timer gesteuert.

Die *Error Detection and Correction Unit (EDCU)* berechnet für die Bits `[+31, -24]` des NTPTIME-Registers eine 8-Bit Checksumme (CS) mit einem modifizierten Hamming Code der Distanz 4. Die Prüfsumme und der Wert des NTPADDERS werden gemeinsam bei jeder steigenden Flanke des Oszillatorpulses im Register NTPHOLD zwischengespeichert. Das bewirkt eine Verzögerung von einem Oszillatortick zwischen der Berechnung und der Aufzeichnung, aber die Uhrensynchronisationsalgorithmen werden davon nicht beeinflußt, da die interne Zeit um einen Tick nach vor gesetzt werden kann, um nach außen das korrekte Ergebnis zu bekommen.

Die Ausgabe der Zeitinformation vom Register NTPHOLD wird auf 4 Register geteilt. Der 24-Bit breite Macrostamp im NTP-Bereich `[+31, +8]` bildet gemeinsam mit dem 32-Bit breiten Timestamp im Bereich `[+7, -24]` und der 8-Bit breiten Checksumme (CS) den internen 64-Bit breiten NTP-Bus, von dem alle Zeitstempel abgezogen werden. Die verbleibenden 32-Bit für den Microstamp (NTP-Bereich `[-25, -56]`) sowie der 3-Bit breite Nanostamp im NTP-Bereich `[-57, -59]` werden intern für die Driftratenkorrektur verwendet. Der gesamte Zeitstempel kann für Testzwecke atomar gespeichert werden und von den Registern `CS:MSGET`, `TSGET`, `USGET` und `NSGET` gelesen werden.

3.3.2 Accuracy Unit (ACU)

Die ACU verwaltet die Intervallgrenzen $\alpha^+(t)$ und $\alpha^-(t)$, wie in Abbildung 1.1 in Abschnitt 1.2 dargestellt ist. Diese Werte bilden das lokale Accuracy-Intervall, das die lokale Zeit $C(t)$ einschließt.

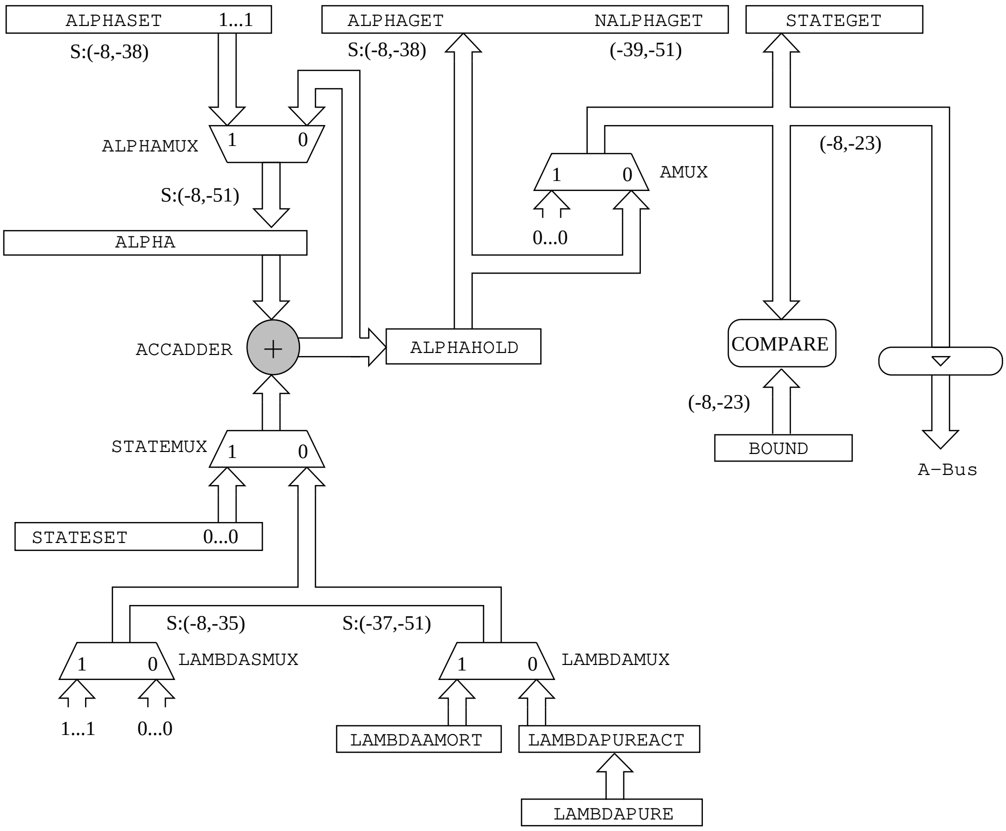


Abbildung 3.4: Accuracy Unit der UTCSU

Die Adder-Based Clock's in der ACU sind sehr ähnlich wie die der LTU aufgebaut. In der schematischen Darstellung in Abbildung 3.4 sieht man in der Mitte den 45-Bit breiten Addierer ACCADDER, der auch negative Zahlen verarbeiten kann. Im Gegensatz zum LTU-Addierer läuft dieser an der Obergrenze nicht über sondern bleibt konstant. Wenn die Obergrenze erreicht wird, so wird ein INT-T Interrupt ausgelöst.

Das Register ALPHA ist in der Rückkopplung des Additionswertes und beinhaltet den aktuellen Accuracy-Wert $\alpha^+(t)$ bzw. $\alpha^-(t)$. Die Initialisierung mit dem Vorladewert des $\pm$`[-8, -38]`-Registers ALPHASET erfolgt mit dem Multiplexer ALPHAMUX, wobei die Bits `[-39, -51]` mit `'1'` initialisiert werden.

Analog zur LTU bestimmt der Multiplexer LAMBDAMUX, ob bei jedem Oszillatortick während der Pure-Phase bzw. der Amortization-Phase die Werte vom $\pm$`[-37, -51]`-Register LAMBDAPUREACT bzw. LAMBDAAMORT addiert werden. Beide Register beinhalten vorzeichenbehaftete Driftwerte, die intern mit dem Multiplexer LAMBDASMUX zu 45-Bit Werten umgewandelt werden.

Während der Amortization-Phase kann der Inhalt des Accuracy-Registers ALPHA negativ werden. Der Multiplexer AMUX wandelt solche negative Werte zu Null um, abhängig vom jeweiligen Vorzeichenbit. Die nichtkonvertierten Accuracy-Werte können von den Registern ALPHAGET und NALPHAGET gelesen werden, die gleichzeitig gespeichert werden, wenn vom LTU-Register TSGETL gelesen wird.

Sowohl negative als auch positive Accuracy-Werte werden im `[-8, -23]`-Bereich exportiert, das entspricht den internen Bussen IA-Bus+ und IA-Bus- der UTCSU. Gemeinsam bilden sie den 32-Bit breiten IA-Bus (siehe Abbildung 3.2), von dem alle Accuracy-Zeitstempel abgezogen werden. Zu beachten ist, daß der Ausgang des Addierers ACCADDER gepuffert wird und mit jeder steigenden Flanke des Oszillatortakts ins Register ALPHAHOLD kopiert wird. Somit sind die Accuracy-Werte am IA-Bus ebenso wie die Zeitwerte am INTP-Bus um einen Taktzyklus verzögert (siehe auch Abschnitt 3.3.1).

Um die Accuracy abzuschranken, vergleicht der 16-bit breite Komparatorblock COMPARE ständig den IA-Bus mit dem `[-8, -23]`-Register BOUND. Wenn der aktuelle Accuracy-Wert größer oder gleich dem Wert des BOUND-Registers ist, wird ein INT-T Interrupt ausgelöst.

3.3.3 Synchronization Subnet Unit (SSU)

Die Anbindung an bis zu 6 Netzwerke[1] erfolgt durch die sechs Synchronization Subnet Units (SSU's), wobei das Senden bzw. der Empfang von Datenpaketen über die UTCSU-Signale `TSXMT[1...6]` bzw. `TSRCV[1...6]` von einer externen Logik (in unserem Fall dem NTI) mitgeteilt wird. Wenn eines dieser Signale, deren Polarität übrigens programmierbar ist, empfangen wird, dann wird in der entsprechenden SSU der NTP-Bus im Bereich `[+31, -24]` und der INTA-Bus im Bereich ±`[-8, -23]` (nur bei Transmit) in Register gespeichert.

Zusätzlich verfügt jede SSU über zwei programmierbare DUTY-Timer für jedes angeschlossene Netzwerk. Alle Duty-Timer sind 48-Bit breit und vergleichen ihren Vorladewert mit dem NTP-Bus. Wenn der Wert des NTP-Busses größer oder gleich dem Vorladewert des Duty-Timers ist, dann wird ein `INTT`-Interrupt ausgelöst, sofern diese Interruptquelle aktiviert ist.

3.3.4 GPS Unit (GPU)

Externe Uhrensynchronisation benötigt den Zugriff auf eine externe, hochpräzise Zeitquelle. Wir haben uns aufgrund unserer hohen Ansprüche bezüglich Accuracy und Zuverlässigkeit für das GPS-System entschieden (siehe [Dan97], [Hoe95] bzw. [HS97]), das UTC in unser System einspeist. Aus Redundanzgründen können bis zu drei Receiver an eine UTCSU angeschlossen werden.

Die Signale `1pps[1...3]` und `Status[1...3]` sind direkt an die GPU[1...3] der UTCSU geführt, um Zeitstempel abzuziehen bzw. den Empfängerstatus beim Eintreffen eines 1pps-Pulses aufzuzeichnen. Der innere Aufbau einer GPU besteht aus zwei Sampleregistern, die Zeitstempel (Macrostamp und Timestamp) vom INTP-Bus der UTCSU zwischenspeichern, wenn ein 1pps-Puls getriggert wird. Das MSB der Checksumme wird zugunsten des Statussignals geopfert, das zeitgleich in das Macrostamp-Sampleregister eingefügt wird. Um das System mit möglichst vielen, unterschiedlichen GPS-Empfängertypen betreiben zu können, kann im Chip konfiguriert werden, ob die aktive Flanke des 1pps-Pulses eine `'LOW'`→`'HIGH'` oder eine `'HIGH'`→`'LOW'` Transition ist. Die Polarität des gültigen Statussignals kann auch programmiert werden.

Sehr gute GPS-Empfänger bieten zusätzlich einen hochpräzisen Frequenzausgang (z.B. `10MHz`) mit dem die UTCSU getaktet werden kann.

Die 1pps-Ereignisse sind konzeptuell sehr ähnlich wie der Empfang von CSP's. Die genaue Zeitinformation zum Sekundenpuls (der zu jeder vollen UTC-Sekunde kommt) wird nach dem Eintreffen des Pulses üblicherweise über eine serielle Schnittstelle (z.B. RS232) geliefert. Da die meisten Empfänger unterschiedliche Datenformate für die serielle Kommunikation verwenden, ist es die Aufgabe der CPU, die Daten zu interpretieren und zu verarbeiten. Eine Einbettung in Hardware, also das Verarbeiten der seriellen Information in der UTCSU, wäre viel zu umständlich und zu unflexibel.

[1] Der Grund, warum sechs SSU's in der UTSCU sind, ist, damit Gateway-Knoten mit dreifach-redundanten Netzwerken gebaut werden können.

3.3.5 Application Unit (APU)

Die Application Unit (APU) bietet Möglichkeiten zur Interruptgenerierung zu spezifizierten Zeitpunkten und zur Aufzeichnung der Zeit beim Auftreten anwendungsspezifischer Ereignisse. Die erste Möglichkeit wird mit einem Duty-Timer realisiert, bei dessen Ablauf ein `INTA`-Interrupt ausgelöst wird (ähnlich wie bei den Duty-Timern der SSU's). Der zweite Fall ist etwas aufwendiger, da ein atomares Aufzeichnen des INTP-Busses und der IAP(N)-Busse notwendig ist. In der UTCSU werden dafür neun Registersets zur Verfügung gestellt, in denen die Buswerte atomar gespeichert werden. Dies geschieht dann, wenn an einem der neun Eingänge ein Puls auftritt. Die Art der Transition kann natürlich programmiert werden. Zusätzlich wird in einem Statusbit angezeigt, ob ein CPU-Zugriff auf ein Application-Registerset stattgefunden hat.

3.3.6 Snapshot Unit (SNU)

Die Snapshot Unit (SNU) bietet drei Möglichkeiten für Debugging- bzw. Testunterstützung und zwar den *Software Snapshot*, den *Hardware Snapshot* und die *Synchronous Run Operation*. Die Snapshots werden benötigt, um atomare und vollständige Abbilder der internen Register NTPTIME, ALPHAP und ALPHAN zu erzeugen.

Ein Software Snapshot wird getriggert, wenn auf ein dediziertes UTCSU-Register geschrieben wird oder wenn ein Duty-Timer abläuft. Im Chip wird damit bewirkt, daß simultan das gesamte interne LTU-Register NTP-HOLD in die Register `MSGET`, `TSGET`, `USGET` und `NSGET` kopiert werden. Gleichzeitig werden die internen ACU-Register ALPHAPHOLD bzw. ALPHANHOLD in die Register `ALPHAP(N)GET`, `NALPHAP(N)GET` und `STATEP(N)GET` gespeichert. Wenn diese Register gelesen werden, bevor ein weiterer Software Snapshot auftritt, so werden die zuletzt aufgezeichneten Werte ausgegeben.

Ein Hardware Snapshot wird durch einen Puls am Eingang `HWSANP` ausgelöst, dessen Polarität programmiert werden kann. Da solche Ereignisse naturgemäß asynchron auftreten, werden der INTP-Bus und IA-Bus in eigens dafür vorgesehene Register gespeichert.

Abschließend kann am Eingang `SYNCRUN` ein Starten aller drei ABC's simultan erfolgen, was für den gleichzeitigen Start mehrere UTCSU's eine notwendige Voraussetzung ist.

3.3.7 Interrupt Unit (ITU)

Die UTCSU ist imstande, einige asynchrone Ereignisse durch die Generierung von Interrupts an die CPU zu melden. Alle Interrupt-Quellen wurden in drei Kategorien zusammengefaßt, um möglichst wenig Interruptleitungen zu benötigen:

- Der Interrupt `INTN` signalisiert externe Ereignisse, die entweder vom Zeitstempeln empfangener oder gesendeter Datenpakete oder von den 1pps- bzw. Statussignalen der GPS-Empfänger herrühren.
- Der Interrupt `INTT` bezieht sich auf Ereignisse, die für die Uhrensynchronisation relevant sind (Duty Timer) bzw. auf Accuracy-Überläufe.
- Der Interrupt `INTA` signalisiert sowohl anwendungsspezifische Ereignisse als auch Ereignisse der Chip-Test-Unterstützung.

Die ITU beinhaltet die Logik zum Erzeugen der Interrupts. Alle Interrupt-Quellen können in den Registern `UTCINTEN1/2` individuell ein- bzw. ausgeschaltet werden. Anliegende Interrupts werden in den Registern `UTCINTSTAT1/2` angezeigt. Das Löschen der Interrupts erfolgt mit den Registern `UTCINTCLEAR1/2`.

Eine detailliertere Erklärung für die UTSCU-Interrupts sowie ein Beispiel kann im Abschnitt 3.6 nachgelesen werden.

3.3.8 Built-In Test Unit (BTU)

Prinzipiell können alle Sequenzen von Zeit- bzw. Accuracy-Werten der UTCSU auch außerhalb des Chips berechnet werden. Der extrem hohe Durchsatz ermöglicht es allerdings nicht, daß eine kontinuierliche Überprüfung durchgeführt werden kann.

Die UTCSU bietet zwei Kompressionsmethoden, die über eine bestimmte Menge von Zeitstempeln angewendet werden, damit ein kontinuierlicher, aber nicht sehr häufiger Datentausch zwischen der UTCSU und einem externen Vergleichsgerät stattfinden kann. Die beiden Methoden sind die Blocksummen- bzw. die Signaturberechnung des internen NTPTIME-Registers mit den Bits `[+31, -24]` und dem `[+8, -23]`-Bereich der internen Accuracy-Register ALPHAP und ALPHAN, die in einem durch Start- und Stop-Operationen begrenzten Intervall ausgeführt werden. Beide Methoden können unabhängig von einander verwendet werden.

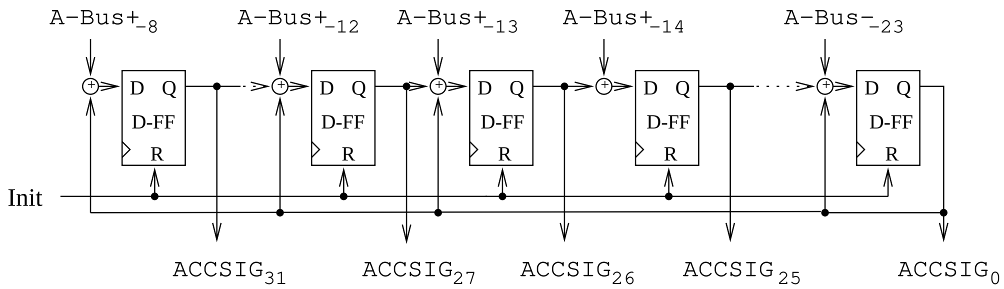


Abbildung 3.5: Signaturerzeugung für Accuracystamps

Die Abbildung 3.5 zeigt die Logik mit der die Signatur für die Accuracy-Werte am A-Bus berechnet werden. Die Accuracy-Werte werden vom A-Bus genommen und in ein *Linear Feedback Shift Register (LFSR)* geschoben (siehe [YK93]). Diese Register implementieren die Auswertung eines Polynoms $P(x)$ mit Koeffizienten aus $\{0, 1\}$. Jede Potenz von x^k wird durch D-Flip-Flop's (DFF) erzeugt, deren Eingang mit einer X-OR Verknüpfung des Ausgangs der Stufe x^{k-1} und dem korrespondierenden Bit des INTA-Busses erzeugt wird. Potenzen, deren Koeffizienten nicht Null sind, haben eine zusätzliche Rückkopplung des DFF_0-Ausgangs. Für die Signaturberechnung des INTP-Busses wird ein Polynom vom Grad 56 verwendet, und zwar

$$P_{INTP-Bus}(x) = x^{56} + x^{22} + x^{21} + x^1 + 1.$$

Das Polynom

$$P_{A-Bus}(x) = x^{32} + x^{28} + x^{27} + x^1 + 1$$

vom Grad 32 wird für die Berechnung der Accuracy-Signaturen verwendet.

Das Prinzip der Blocksummenberechnung für den A-Bus wird in der Abbildung 3.6 dargestellt. Ein 32-Bit Addierer summiert die Werte am A-Bus und den Wert des 32-Bit Registers ACCSUM, der rückgekoppelt wird.

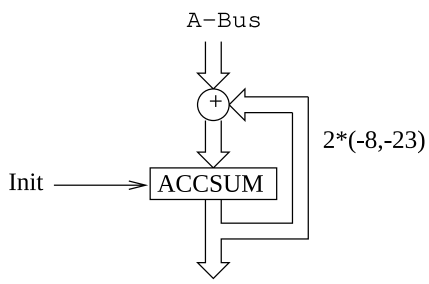


Abbildung 3.6: Blocksummenerzeugung für Accuracystamps

Nur die Verarbeitung einer ganzen Sequenz von Zeit-/Accuracy-Werten kann mit dem LFSR oder Adder-Block berechnet werden, da Start- und Stopzeitpunkte definiert werden müssen. Sehr geeignet dafür sind die Umschaltzeitpunkte zwischen Pure- und Amortization-Phase.

3.3.9 Network Time Interface Unit (NTU)

Für manche Anwendung ist es notwendig, die volle Zeit- und Accuracy-Information kontinuierlich zur Verfügung zu haben. Die UTCSU unterstützt dies, in dem die 64-Bit Zeitinformation des INTP-Busses und die 32-Bit Accuracywerte der IAP(N)-Busse am CSUT-Bus mit jedem Oszillator-Tick nach außen geführt werden. Um Pins zu sparen, werden bei jeder steigenden bzw. fallenden Taktflanke je eine Hälfte der 96-Bit Information am 48-Bit breiten, gemultiplexten CSUT-Bus ausgegeben. Die Verarbeitung dieser Information ist allerdings eine komplizierte Angelegenheit, da sich bei einer Taktfrequenz von $2^{24}Hz$ immerhin eine Datenübertragungsrate von 192MByte/s ergibt.

3.3.10 Bus Interface Unit (BIU)

Die Bus Interface Unit (BIU) beinhaltet jene Logik, mit deren Hilfe die UTCSU in eine breite Vielfalt von Systemarchitekturen eingebunden werden kann. Es wird damit ermöglicht, den ASIC an Datenbusse mit 8-/16-/32-Bit anzuhängen, Little-Endian, bzw. Big-Endian Byteordnung einzustellen bzw. die Zugriffszeiten zu variieren. Die hier verwendete Technik stammt von Motorola und wird als *Dynamic Bus Sizing* bezeichnet. Genaue Details können in [Mot90] nachgelesen werden.

3.4 I/O-Space des Network Time Interface Moduls

Zusätzlich zu den UTCSU-Registern und dem Shared-Memory, die beide im *Memory Space* liegen (siehe Abschnitt 3.5), sind noch ein paar Register im CPLD realisiert, die hauptsächlich für die Interrupt-Generierung notwendig sind. Wie diese Register im 256 Byte großen *I/O-Space* angeordnet sind, wird in der Abbildung 3.7 gezeigt[2].

Lesezugriffe		Schreibzugriffe	
SPROM READ	0xFE	SPROM WRITE	0xFE
OFFSET TYPEFIELD	0x1A	OFFSET TYPEFIELD	0x1A
-	0x18	-	0x18
TRIGGER-OFFSETS	0x16	TRIGGER-OFFSETS	0x16
-	0x14	-	0x14
MISCELLANEOUS	0x12	MISCELLANEOUS	0x12
-	0x10	-	0x10
LA-TRIGGER	0x0E	LA-TRIGGER	0x0E
-	0x0C	-	0x0C
-	0x0A	EN/DIS INT.-LOGIC	0x0A
-	0x08	-	0x08
GET INT.-VECTOR BASE	0x06	SET INT.-VECTOR BASE	0x06
-	0x04	-	0x04
RCV-HEADER BASE	0x02	-	0x02
-	0x00	-	0x00

Abbildung 3.7: Memory-Map des NTI I/O-Space

Die Erklärung der Register erfolgt für Lese- und Schreibzugriffe parallel, da immer logisch verwandte Funktionen an den gleichen Speicherstellen angeordnet sind. [3]

Das `RCV-HEADER BASE` Register an der Adresse `0x02` wird benötigt, damit die Receive-Timestamps korrekt zu den empfangenen Datenpaketen zugeordnet werden können: Nachdem die UTCSU einen Empfangs-Zeitstempel aufgezeichnet hat, muß dieser in einen freien Bereich des aktuellen CSP's eingefügt werden, bevor das nächste Datenpaket empfangen wird. Das kann mit einer Interrupt Service Routine realisiert werden, die immer dann aufgerufen wird, wenn das Signal $\overline{\texttt{TTSRCV}}$ einen Interrupt in der UTCSU auslöst. Aus dem `Receive Header Base` Register wird die Basisadresse des Receive-Headers ausgelesen. Die Zeitstempel werden von der CPU aus der SSU1 der UTCSU in den Speicher an die Adressen: `Basisadresse + Offset` geschrieben.[4]

Zwei weitere Register sind für die Interrupt-Generierung, die im Abschnitt 3.6 genau erklärt wird, notwendig: Die Register `SET/GET INT.-VECTOR BASE` an der Adresse `0x06` werden für die Programmierung des Interrupt-Vektors des Moduls benötigt. Im Vektor sind auch die drei Interrupt-Quellen der UTCSU ($\overline{\texttt{INTN}}$, $\overline{\texttt{INTT}}$ und $\overline{\texttt{INTA}}$) kodiert. Durch Schreibzugriffe an die Adresse `0x0A` kann im Register `Enable/Disable Int. Logic` die Interruptlogik ein- bzw. ausgeschaltet werden. Üblicherweise wird sie einmal nach dem Modul-Reset und am Ende der Interrupt Service Routine aktiviert. Im Register `GET INT.-VECTOR BASE` kann im MSB der Status der Interrupt Logik ausgelesen werden.

[2] Bei der neusten Version des NTI sind alle Register doppelt ausgeführt, und zwar an den Speicherstellen n wie oben und ebenfalls an den Speicherstellen $n-2$, also liegt z.B. das Register `RCV-HEADER BASE` sowohl an der Adresse `0x00` als auch an der Adresse `0x02`.

[3] Zu beachten ist, daß die Registernamen in der Abbildung und im Text (in weiterer Folge auch im Kapitel 5.4, „Programmiermodell der I/O-Space Register“) mit den Registerbezeichnungen in den Handler-Files (siehe Abschnitt 4.5) nicht hundertprozentig identisch sind sondern nur sinngemäß übereinstimmen!

[4] Es gibt natürlich auch andere Möglichkeiten, die allerdings nicht immer funktionieren. Zum Beispiel könnte man versuchen, die Zeitstempel in der Paketempfangs-ISR (wird vom COMCO initiiert) auszulesen und in den Speicher zu schreiben, da dort die Basisadresse des Receive-Headers jedenfalls bekannt ist. Unglücklicherweise könnte dies aber zu spät sein, d.h. Zeitstempel könnten verloren gehen, wenn beispielsweise back-to-back CPS's auftreten. Völlig unzureichend ist es, von einer sequentiellen Ordnung der Datenpakete auszugehen und auf diese Weise die Zeitstempel einzufügen, da CSP's zwar Zeitstempel abziehen könnten, obwohl sie aufgrund eines ungültigen CRC-Codes schließlich weggeworfen werden. Selbstverständlich könnte das Einfügen der Zeitstempel in den Speicher von der Hardware gemacht werden, doch ist das nicht mit geringem Aufwand realisierbar.

Durch Lesen oder Schreiben des Registers `LA-TRIGGER` an der Speicherstelle `0x0E` wird ein Puls am Signal `TRIGGER` des CPLD ausgegeben[5], das mit einem Jumper auf $\overline{\texttt{HWSNAP}}$ verbunden werden kann. Vor allem für Debugging-Zwecke ist diese Möglichkeit sehr dienlich.

An der Adresse `0x12` ist das `MISCELLANEOUS`-Register angeordnet[6]. Damit können die Pegel einiger statischer Signale für die UTCSU erzeugt bzw. rückgelesen werden. Auch ein UTCSU-Reset kann mit Schreibzugriffen auf dieses Register bewirkt werden.

Das Register `TRIGGER-OFFSETS` an der Adresse `0x16` ermöglicht die Programmierung der relativen Adressen in den CSP-Paketen, an denen die Trigger-Signale für die Receive- bzw. Transmit-Timestamps abgezogen werden und das Transparent Timestamp Mapping passiert.

Um die Interrupt-Last beim Empfang von Datenpaketen zu reduzieren, wird mit dem Type-Field jedes Pakets überprüft, ob es sich um ein CSP-Paket oder ein normales Datenpaket handelt. Mit dem Register `OFFSET TYPEFIELD` an der Adresse `0x1A` kann die Adresse im Datenpaket angegeben werden, an der das Type-Field liegt. Diese Maßnahme gewährt ein bißchen „Spielraum“ für den Aufbau der Datenpakete.

Ganz oben im *I/O-Space*, an der Adresse `0xFE`, ist die Modulidentifikation durch die Einbettung der SPROM-Steuerung in die Register `SPROM READ/WRITE` realisiert. Über vier Leitungen kann das SPROM softwaremäßig, also nicht mit einem kompletten Mechanismus im CPLD, gesteuert werden.

[5] Das Signal kann am `Pin 2` des Jumpers `J5` abgegriffen werden (siehe Abschnitte 6.5 und 6.6).

[6] Bei der aktuellen Version des NTI VHDL-Codes ist dieses Register deaktiviert, da im CPLD nicht genug Platz ist. Alle Signale sind im CPLD statisch auf die Default-Werte gesetzt (siehe Abschnitt 5.4.4).

3.5 Memory Space des NTI MA-Moduls

Alle UTCSU- und Speicherzugriffe sind in den *Memory Space* des *Network Time Interfaces* eingebettet. Wie bereits im Abschnitt 2.2 erläutert wurde, ist bei COMCO-Zugriffen auf den Speicher zusätzliche Logik zur Paket-Zeitstempelung notwendig. Um zwischen CPU- und COMCO-Zugriffen zu unterscheiden, dekodiert das CPLD zwei verschiedene Speicherregionen in den gleichen physikalischen Speicher, wie in Abbildung 3.8 dargestellt ist.

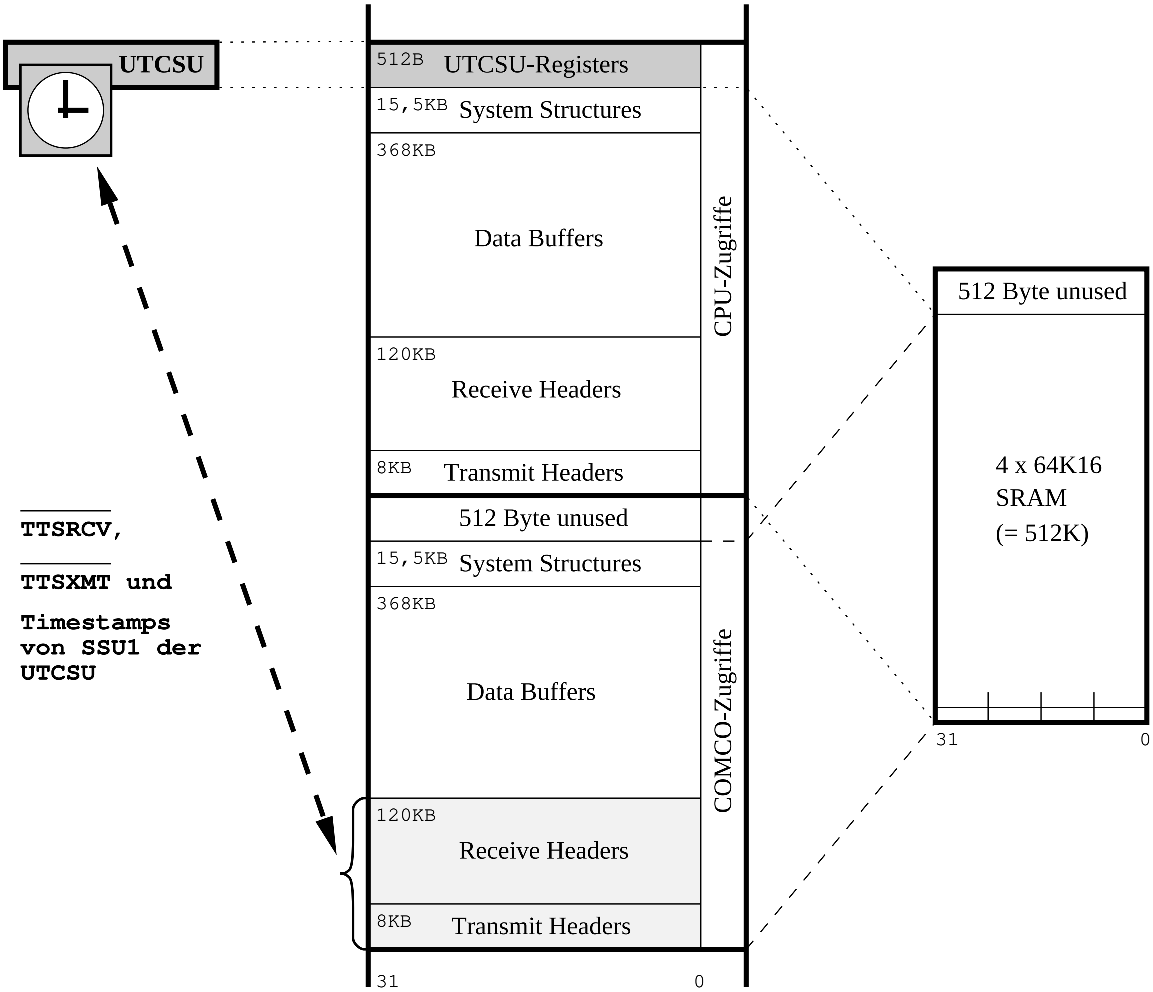


Abbildung 3.8: Memory-Map des NTI Memory-Space

Das NTI-Modul bietet bis zu 512 KByte physikalischen Speicher.[7] Ganz oben im Speicher ist die UTCSU angesiedelt, sie kann also nur im CPU-Bereich angesprochen werden. Da bei unseren Carrier-Boards die Adreßleitungen `A20, ..., A23` für die Adreßierung der MA-Module verwendet werden, mußte die UTCSU so in den *Memory Space* gelegt werden, daß auf 512 Byte des SRAM-Speichers verzichtet wird.

Prinzipiell wird der Speicher in vier Regionen geteilt: *Transmit Header, Receive Header, Data Buffer und System Structures*. Im CPU-Adreßbereich werden die beinahe 512 KByte dieser vier Regionen ganz normal dekodiert. Im *System Structures* Bereich sind die Systemdatenstrukturen und die Befehlsschnittstelle des COMCO's angeordnet. Die *Data Buffer* sind für gewöhnliche Paketdaten vorgesehen. Spezielle Funktionen werden nur im *Receive-Header* und im *Transmit Header* Bereich des COMCO-Bereichs ausgeführt. Die Header beinhalten paketspezifische Kontroll- und Routing-Informationen (z.B. Source- und Destination-Adresse).

[7] Bei den Modulen, die für die Evaluation im SynUTC-Projekt benötigt werden, sind nur 256 KByte SRAM bestückt (und zwar die IC's *U12* und *U14*). Die vier Teilbereiche *Transmit Header, Receive Header, Data Buffer und System Structures* müssen jeweils halbiert werden.

Die hier beschriebene Version des *Network Time Interfaces* ist für den Intel i82596CA Ethernet Coprocessor [Int92] mit 128 Byte Transmit- und 64 Byte Receive-Header ausgelegt. Abbildung 3.9 stellt die Offsets der Header dar, an denen die Triggerlogik aktiv wird.

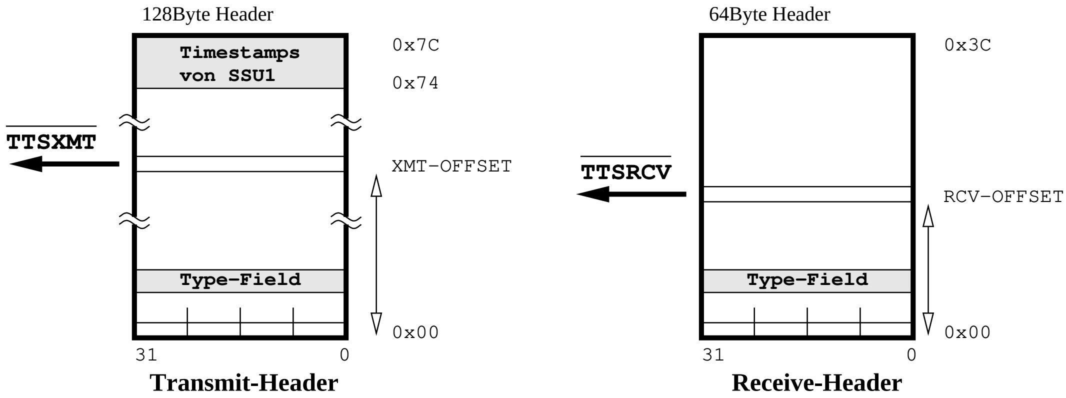


Abbildung 3.9: Transmit und Receive Header für Intel i82596CA

Beim Lesen des Offsets `XMT-OFFSET` [8] im Transmit-Header eines zu sendenden CSP (Clock Synchronisation Packet) wird ein Puls am Trigger-Signal $\overline{\texttt{TTSXMT}}$ vom CPLD erzeugt. Mit diesem Signal werden in der SSU1 der UTCSU die Transmit-Timestamps abgezogen. Die Zeitstempel werden dann vom CPLD transparent in den Transmit-Header eingefügt, wenn der COMCO über die letzten drei Longword-Adressen des Transmit-Packet Headers liest. An der Speicherstelle mit dem Offset `0x74` wird der Inhalt des UTCSU-Registers `MSXMT`, am Offset `0x78` der Wert von `TSXMT` und beim Offset `0x7C` der Accuracy-Wert des Registers `ACCXMT` der SSU1 eingefügt.

Wenn der i82596CA den Offset `RCV-OFFSET` [8] in einem Receive-Header beim Empfang eines CSP schreibt, generiert das CPLD einen Puls am Trigger-Signal $\overline{\texttt{TTSRCV}}$, das in der UTCSU die Receive-Timestamps in der SSU1 abzieht. Zusätzlich wird die Basisadresse des Headers im Register `RCV-HEADER BASE` des CPLD's gespeichert, um eine anschließende Interrupt-Behandlung (siehe Abschnitt 3.6) zu ermöglichen.

Wichtig ist noch die Tatsache, daß die oben genannten Mechanismen (Erzeugung der Trigger-Signale, Transparent Time Stamp Mapping und Speichern der Receive Header Base) nur dann erfolgen, wenn es sich um den Header eines CSP-Paketes handelt. Mit Hilfe des *Type-Fields* kann das CPLD feststellen, welche Art von Datenpaket gesendet bzw. empfangen wird. Wenn im Type-Field, dessen Offset übrigens im I/O-Space programmierbar ist, der Wert `0x5000` steht, dann müssen die speziellen Funktionen für das Clock Synchronization Packet aktiviert werden.

[8] Diese Offsets sind im I/O-Space des NTI-Moduls programmierbar, siehe dazu Abschnitt 3.4 bzw. 5.4.5.

3.6 Interrupt-Behandlung

Wie im Kapitel 3.3.7 beschrieben wurde, sind sämtliche Interrupt-Quellen der UTCSU in drei Funktionsgruppen gegliedert und an die drei Signale $\overline{\mathtt{INTN}}$ (Netzwerk), $\overline{\mathtt{INTT}}$ (Zeitstempel) und $\overline{\mathtt{INTA}}$ (Anwendung) geführt. M-Module haben jedoch nur eine einzige Interrupt-Leitung $\overline{\mathtt{IRQ}}$, können aber Interrupt-Vektoren erzeugen. Im Falle der NTI wird das so gemacht, daß die drei niederwertigsten Datenleitungen `D2`, `D1` und `D0` des 8 Bit breiten Vektors den Status der drei Interruptleitungen $\overline{\mathtt{INTN}}$, $\overline{\mathtt{INTT}}$ und $\overline{\mathtt{INTA}}$ wiederspiegeln. Die oberen fünf Bit des Vektors können im *I/O-Space*-Register `SET INT.-VECTOR BASE` programmiert werden.

Die Abbildung 3.10 zeigt ein Beispiel zur Interrupterzeugung, anhand dessen die Signale und die Interruptbehandlung erläutert werden.

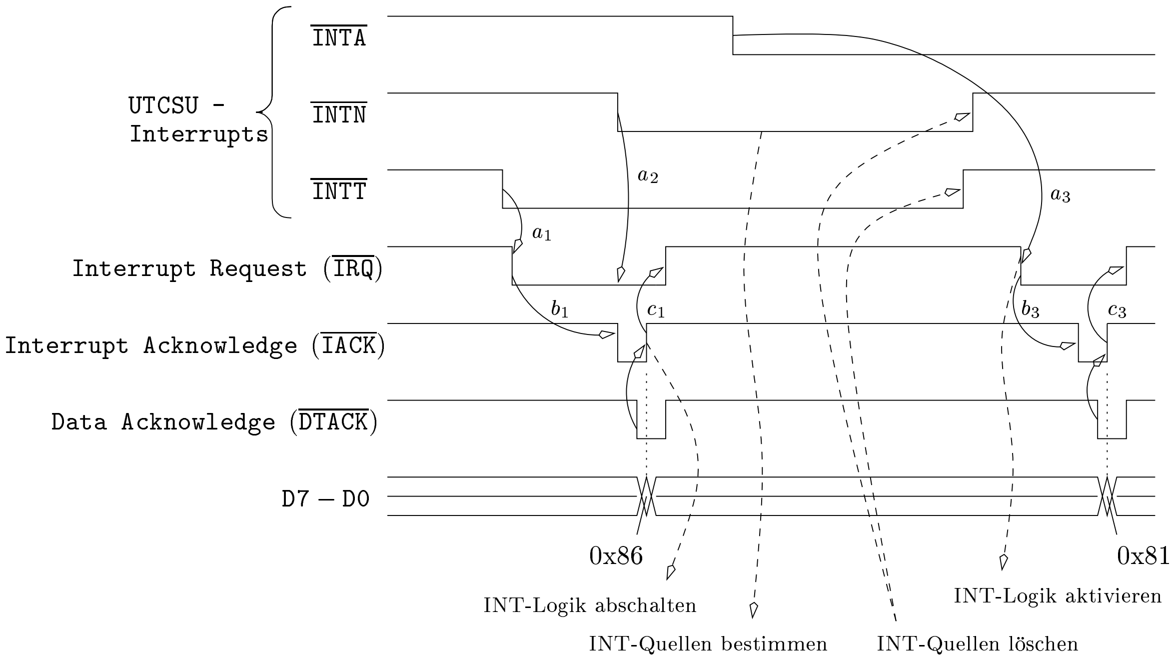


Abbildung 3.10: Beispiel für Interruptbehandlung

Falls die Interrupt Basis Adresse auf `0x80` programmiert wurde, verursacht ein UTCSU-Interrupt am Signal $\overline{\mathtt{INTT}}$ (a_1), daß der gültige, interne Interruptvektor auf `0x82` gesetzt wird[9] und das M-Modul durch Setzen der Leitung $\overline{\mathtt{IRQ}}$ auf `'LOW'` am Carrier-Board einen Interrupt signalisiert.

Noch bevor der Interrupt-Vektor im Interrupt-Acknowledge Zyklus zur CPU transportiert wird, wird ein zweiter Interrupt aktiv, nämlich $\overline{\mathtt{INTN}}$ (a_2). Die CPU akzeptiert die Interrupt-Anforderung des M-Moduls, indem sie das Signal $\overline{\mathtt{IACK}}$ auf `'LOW'` setzt (b_1), womit das NTI den Interrupt-Vektor `0x86` an die Datenleitungen `D7`...`D0` legen muß. Dieser Bus Vektor Zyklus wird vom Modul mit $\overline{\mathtt{DTACK}}$→`'LOW'` terminiert.

Sobald die CPU das $\overline{\mathtt{IACK}}$-Signal wieder inaktiv schaltet (c_1), muß das Modul die $\overline{\mathtt{IRQ}}$-Leitung wieder auf `'HIGH'` legen. In der M-Modul Spezifikation (siehe [Man96]) wird das Zurücksetzen der $\overline{\mathtt{IRQ}}$-Leitung mit dem Löschen der Interrupt-Quellen verbunden („Hardware End of Interrupt“). Beim *Network Time Interface* ist das aufgrund der vielen UTCSU-Interrupt-Quellen nicht ohne weiteres möglich, da diese von der CPU in dafür vorgesehenen UTCSU-Registern gelöscht werden müssen.

Bei der NTI behilft man sich mit einem Trick, und zwar wird die Interrupt-Logik im CPLD abgedreht, wenn am Signal $\overline{\mathtt{IACK}}$ eine steigende Flanke auftritt (c_1). Bevor die CPU die Interrupt Service Routine verläßt, muß sie die Interrupt Logik wieder aktivieren. Wie bereits beschrieben, geschieht dies, indem in das *I/O-Space*-Register `EN/DIS INT.-LOGIC` der Wert `'0x01'` geschrieben wird. In unserem Beispiel wird noch ein UTCSU `INTA`-Interrupt aktiv (a_3), nachdem die anderen Interrupt-Quellen gelöscht wurden aber noch bevor die ISR verlassen wurde. Nach dem neuerlichen Aktivieren der Interrupt-Logik beginnt das Modul einen neuen Interrupt-Zyklus (b_3).

[9] Man beachte, daß die `'LOW'-aktiven` Signale $\overline{\mathtt{INTN}}$, $\overline{\mathtt{INTT}}$ und $\overline{\mathtt{INTA}}$ im CPLD invertiert werden und somit `'HIGH'-aktive` Vektoren produzieren.

3.7 Externe Signale

In diesem Abschnitt wird die Funktion aller externen Signale und deren Auswirkung auf die UTCSU erklärt. Alle Eingangssignale sind nach Funktionsgruppen gegliedert. Mit Abbildungen wird anschaulich erläutert, welche Quellen die Signale treiben und wie sich die Pufferung bzw. die optoelektronische Entkopplung am NTI auf die Signale auswirken.

3.7.1 GPS Interface

Die Einspeisung einer externen Referenzzeit ist am *Network Time Interface* für GPS-Empfänger über den FPC (siehe Abschnitt 6.2) möglich. An einen Primärknoten können aus Redundanzgründen maximal drei GPS-Receiver angeschlossen werden, wie in Abbildung 3.11 dargestellt ist.

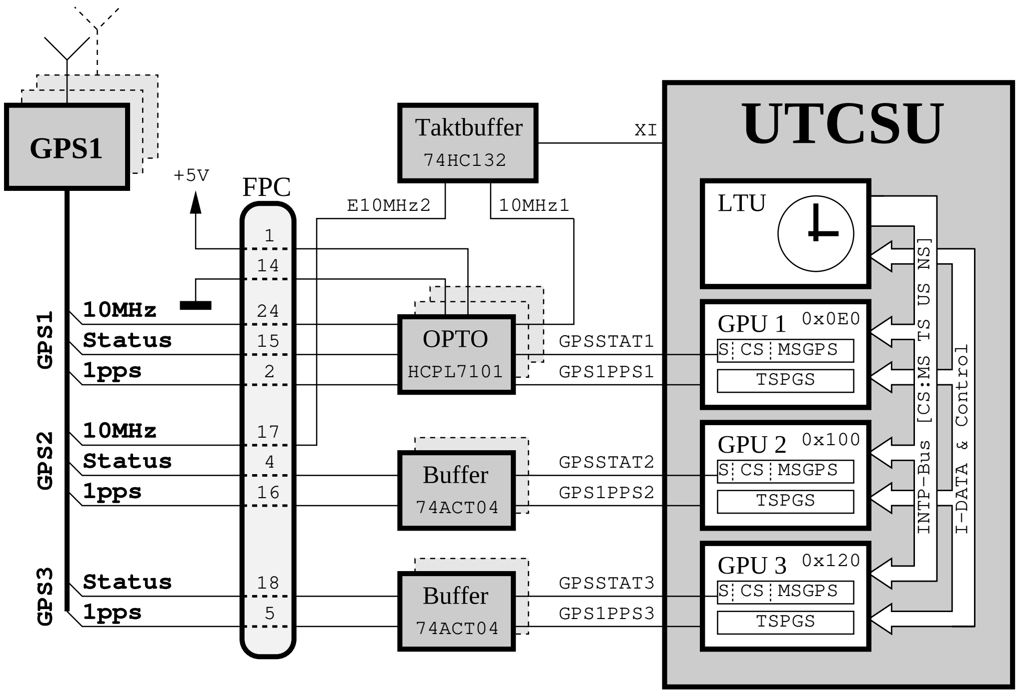


Abbildung 3.11: Ankopplung von GPS-Empfängern an das NTI

Alle Signale des GPS-Empfängers 1 werden am NTI optoelektrisch getrennt. Die Signale werden nichtinvertiert an die UTCSU-Pins geführt. Das Signal `GPS1PPS1` wird am `Pin 2`, das Signal `GPSSTAT1` am `Pin 15` und das Signal `10MHz1` am `Pin 24` des Front Panel Steckers (FPC) angeschlossen. Ganz wichtig ist, daß die Optokoppler unbedingt einen eigenen Versorgungsstromkreis brauchen, deren Masse am `Pin 14 (GND GPS1)` und `+5V`-Spannung am `Pin 1 (+5V GPS1)` geliefert werden müssen.

Die 1pps- und Statussignale der anderen beiden Receiver werden von 74ACT04-SMD-Invertern gepuffert. Das bedeutet, daß diese Signale an der UTCSU entgegengesetzte Polarität zu den korrespondierenden GPS-Empfängerausgängen haben. Es besteht jedoch die Möglichkeit die Polarität bzw. die aktiven Flanken in der UTCSU zu konfigurieren (siehe [Loy96], Kapitel 8.4).

Um das NTI-Modul von einem hochpräzisen Frequenzausgang eines GPS-Empfängers takten zu lassen, können für die Receiver 1 und 2 die Signale `10MHz1` und `E10MHz2` an den Pins 24 bzw. 17 des FPC angeschlossen werden. Beide Signale können das NTI takten, wenn ein 74HC132-Baustein in den Quarzsockel (siehe Kapitel 6.10.3) gesteckt wird. Wichtig ist jedoch, daß nur einer der beiden Eingänge beschaltet werden darf!

Die 1pps-Signale ziehen in der jeweiligen GPU einen Zeitstempel ab, wenn eine aktive Flanke auftritt. Über RS232-Schnittstellen wird die genaue Zeitinformation an die CPU geliefert, die damit die Synchronisationsalgorithmen für die externe Synchronisation „füttert“.

3.7.2 Synchronisation Subnet Interface

Die UTCSU verfügt über sechs Synchronisation Subnet Units, deren Aufgabe es ist, die Anbindung von bis zu sechs Netzwerken zu ermöglichen.

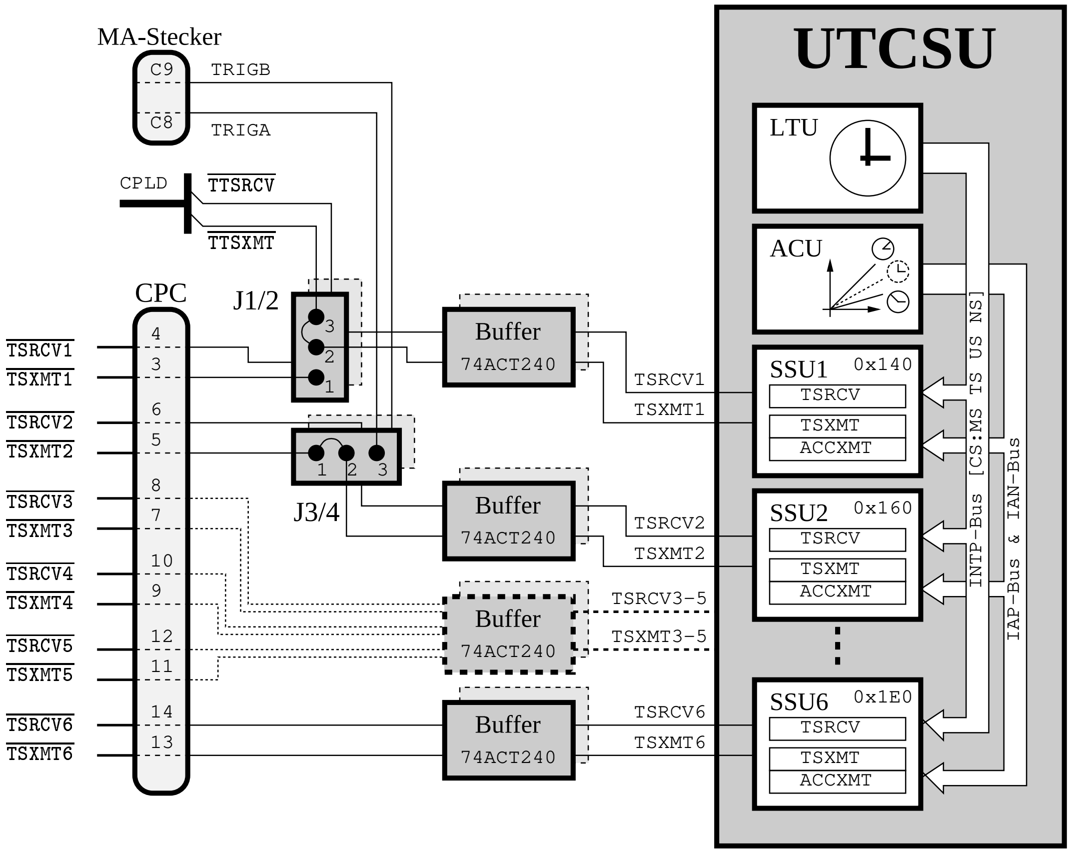


Abbildung 3.12: Eingänge der Synchronisation Subnet Units (SSU 1-6)

Generell kann die Anbindung der Netzwerke, die durch die Signale $\overline{\texttt{TTSRCV}x}$ bzw. $\overline{\texttt{TTSXMT}x}$ mit $x \in \{1 \ldots 6\}$ stattfindet, für alle SSU's über den Carrier Board Plug Connector (CPC, Abschnitt 6.3) erfolgen. Zu beachten ist, daß alle Signale über 74ACT240-Inverter geführt werden, die als Puffer verwendet werden und die UTCSU vor zu hohen Signalpegeln schützen. Das bedeutet wiederum, daß die Signale mit umgekehrter Polarität zur UTCSU gelangen, als sie ins NTI-Modul geführt werden. Bei den SSU's besteht ebenso wie bei den GPU's die Möglichkeit, daß die aktiven Flanken in der UTCSU programmiert werden können (siehe [Loy96], Kapitel 8.3).

Bei den ersten beiden SSU's können jedoch auch andere Signalquellen die Zeitstempel auslösen, sofern die Jumper `J1`...`J4` anders konfiguriert werden. Eine Tabelle mit allen Jumper-Stellungen des NTI-Moduls befindet sich in Abschnitt 6.7.

Mit den Jumpern `J1` bzw. `J2` wird ausgewählt, ob die Steuersignale für die SSU1 vom CPLD oder vom CPC kommen. Wenn die Signale vom CPC die SSU1-Eingänge `TSXMT1` bzw. `TSRCV1` steuern sollen, dann müssen die Jumper in der Stellung `1-2` konfiguriert werden. Die Standardeinstellung der Jumper `J1/2`, nämlich die Konfiguration `2-3`, bedeutet, daß die vom CPLD erzeugten Trigger-Signale beim Senden bzw. Empfangen von CSP-Paketen (siehe Abschnitt 3.5), invertiert an die SSU1 geführt werden.

Die zweite Synchronisation Subnet Unit der UTCSU kann auch durch die Signale `TRIGA` bzw. `TRIGB` des M-Modul Carrier-Boards gesteuert werden. Die MA-Modul Spezifikation [Man96] sieht nämlich vor, daß vom Carrier-Board über die Pins `C8` bzw. `C9` die Triggersignale `TRIGA` bzw. `TRIGB` zum MA-Modul geführt werden. Üblicherweise werden beide Signale durch Registerzugriffe auf das Carrier-Board kontrolliert. Wenn der Jumper `J3` in der Position `1-2` ist, dann wird das Signal $\overline{\texttt{TSXMT2}}$ an den Eingang `TSXMT2` der SSU2 geführt, in der Stellung `2-3` steuert `TRIGA` über den Puffer den Eingang `TSXMT2`. Genauso verhält es sich mit den Signalen $\overline{\texttt{TSRCV2}}$ bzw. `TRIGB` und dem Jumper `J4`, in der Stellung `1-2` wird $\overline{\texttt{TSRCV2}}$ invertiert an den Eingang `TSRCV2` der SSU2 geführt bzw. `TRIGB` in der Jumperposition `2-3`.

3.7.3 Application Interface

Die Application Unit (APU) der UTCSU stellt neun anwendungssteuerbare Eingänge zur Verfügung, die auf Flanken triggern und Zeitstempel abziehen. Darüber hinaus gibt es die Möglichkeit, einen Signalausgang durch einen Duty-Timer steuern zu lassen.

Alle Eingänge können sowohl vom FPC- als auch vom CPC-Stecker angeschlossen werden. Die ersten acht Eingänge, $\overline{\texttt{App0}}\ldots\overline{\texttt{App7}}$ werden über invertierende DIP-Buffer an die entsprechenden UTCSU-Eingänge geführt. Der Eingang `App8` hat eine Sonderstellung, und zwar wird er über einen Optokoppler an die UTCSU geführt. Im Gegensatz zu seinen Kollegen wird er nicht invertiert, dafür benötigt er einen eigenen Versorgungskreis für die Primärseite des Optokopplers.

Selbstverständlich ist wieder die Polarität der Flanken aller Eingänge in der UTCSU programmierbar, siehe [Loy96], Kapitel 8.5.

3.7.4 Hardware-Snapshot und synchrone Startoperation

Die Eingangssignale $\overline{\texttt{SYNCRUN}}$ bzw. $\overline{\texttt{HWSNAP}}$ an den Pins `1` bzw. `2` des CPC-Steckers werden über einen SMD-Inverter an die UTCSU geführt. Die Polarität der Flanken kann wie bei allen anderen Eingängen auch hier programmiert werden, siehe dazu Kapitel 8.6 in [Loy96].

Kapitel 4

Einbindung des NTI in VME-Bus Boards unter pSOS^{+m}

In diesem Kapitel wird – wie der Titel bereits verrät – erklärt, wie das Network Time Interface in ein verteiltes Mutliprozessor-Betriebssystem auf Basis von VME-Prozessorboards im Projekt SynUTC eingebettet wird.

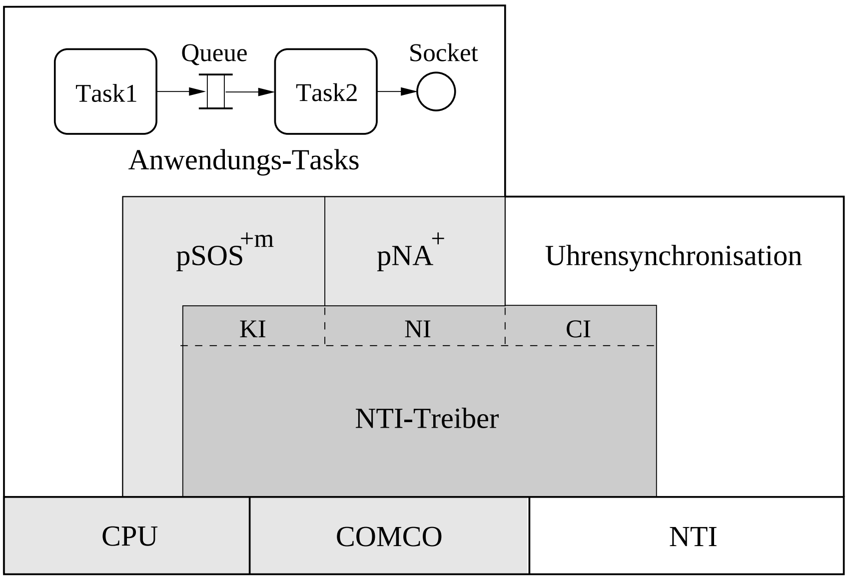


Abbildung 4.1: Aufbau eines pSOS^{+m}-Knotens unter Verwendung des NTI-Treibers

Die Struktur eines synchronisierten Knotens unseres Projekts wird in Abbildung 4.1 schematisch dargestellt. Vorab sei gleich angemerkt, daß die hellgrau unterlegten Felder, also pSOS^{+m}, pNA^{+}, COMCO und CPU, gekaufte Produkte bezeichnen, in die das NTI und die Uhrensynchronisation eingebunden wird. Im Zentrum ist der NTI-Treiber dunkelgrau dargestellt, der die Schnittstelle der Software zur Hardware ist, und im Projekt SynUTC entwickelt wurde.

Aus der Abbildung ist klar ersichtlich, daß der NTI-Treiber [Ric97] der zentrale Softwareteil ist, der die folgenden drei Schnittstellen zum i82596 COMCO und dem NTI implementiert [SN99]:

1. *Kernel Interface* (KI): pSOS^{+m} stellt Multiprozessor-Techniken in Form von verteilten Objekten (Tasks, Queues, Semaphore, etc.) zur Verfügung, die basierend auf Remote Procedure Calls (RPC) implementiert sind. Damit der Kernel vom darunterliegenden Netzwerk weitestgehend unabhängig ist, gibt es eine wohldefinierte Schnittstelle, nämlich das KI, das ein benutzerdefiniertes Nachrichten-Transportsystem zum entsprechenden COMCO herstellt. Nähere Informationen über pSOS^{+m} und das KI werden im Abschnitt 4.1 gegeben.

2. *Network Interface* (NI): Zusätzlich zu den Kernel-Services können Application-Tasks TCP/IP-Sockets zur Kommunikation mit anderen Nodes verwenden, soferne die zusätzliche Softwarekomponente pNA$^+$ verfügbar ist. Ebenso wie beim pSOS^{+m}-Kernel ist das pNA$^+$ hardwareunabhängig gestaltet, indem es ein benutzerdefiniertes NI gibt, das dem KI sehr ähnlich ist aber ein anderes Verfahren zur Nachrichtenübermittlung verwendet. Eine genauere Beschreibung zu pNA$^+$ und zum NI gibt es im Abschnitt 4.2.

3. *Clock Interface* (CI): Die dritte Komponente, die Netzwerkservices benötigt, sind die Algorithmen der Uhrsynchronisation. Das CI bietet ebenfalls ein sehr einfaches Protokoll zur Nachrichtenübermittlung. Das Clock Interface ist das einzige der drei Interfaces, das die Möglichkeit der Paketzeitstempelung durch das NTI verwendet, und wird in Abschnitt 4.3 etwas näher vorgestellt.

In der folgenden Abbildung 4.2 wird veranschaulicht, daß jeder physikalische Knoten des experimentellen Evaluationssystems aus einem Prozessor-Board mit einem COMCO, aber mit vier (oder mehr) Network Time Interface Modulen bestückt ist. Diese Maßnahme ist deswegen notwendig, damit ein kostengünstiges System mit bis zu 20 Nodes (entspricht 5 physikalischen) zur Evaluation der Uhrsynchonisationsalgorithmen aufgebaut werden kann. Wichtig ist dabei, daß zwar alle vier NTI Module Nachrichten senden können, jedoch alle eingehenden Nachrichtenpakete vom COMCO in den Speicher der NTI1 geschrieben werden.

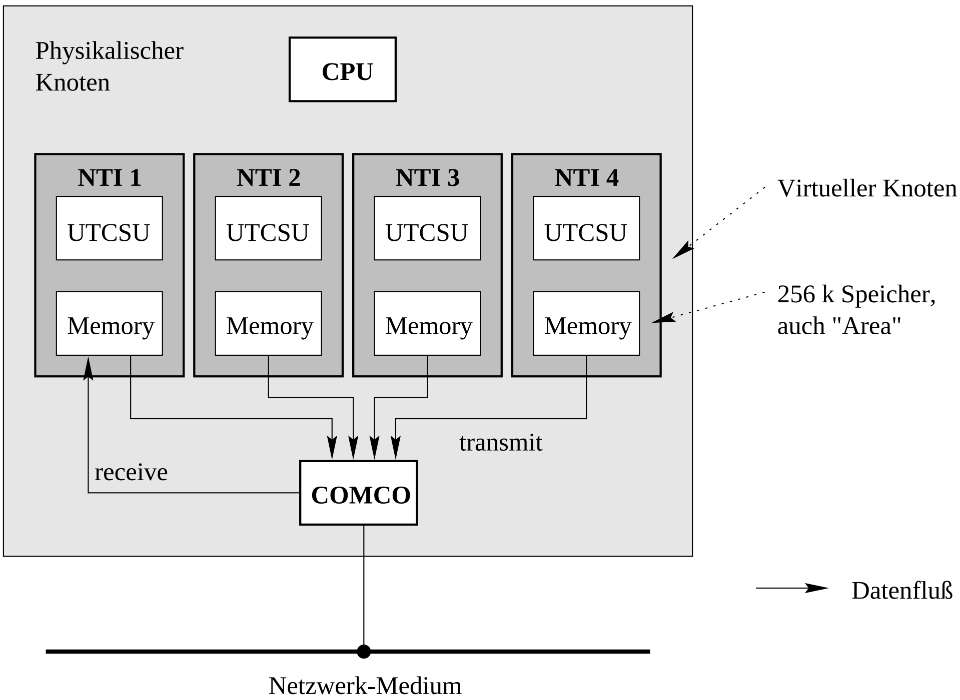


Abbildung 4.2: Struktur eines physikalischen Knotens mit vier NTI's

Ausführliche Dokumentationen zum NTI-Treiber stehen in [Ric97] bzw. in [SS99] zur Verfügung. Alle Dateien des Treibers sind im Projektverzeichnis `$SynUTC/TECH/Software/NTI-Driver/` gespeichert.

4.1 Kurzbeschreibung von pSOS^{+m} und Kernel Interface (KI)

Das Betriebssystem pSOS wird von der Firma ISI (Integrated Systems Inc) entwickelt und weltweit vertrieben. Es besteht aus folgenden Teilen:

- pSOS^{+} Real-Time Operating System (RTOS)
- pSOS^{+m} Multiprozessor-Version von pSOS^{+}
- pNA^{+} TCP/IP Netzwerk-Manager
- pROBE^{+} System-Level Debugger für pSOS-Anwendungen
- pHILE^{+} Filemanager
- pREPC^{+} ANSI C Standardbibliotheken für pSOS-Systeme
- pRPC^{+} Open Network Computing und RPC-Bibliotheken für pSOS-Systeme
- XRAY^{+} Source-Level Debugger für pSOS-Systeme

In den nächsten Abschnitten wird nur ein Überblick über die für uns relevanten Teile von pSOS gegeben, der von [Pus95] entlehnt ist. Nähere Details können in [Int96a] bzw. [Int96b] nachgelesen werden.

pSOS^{+m} ist ein verteiltes Echtzeit-Betriebssystem und stellt verschiedene Funktionalitäten zum Datenaustausch, zur Kommunikation und zur Synchronisation von asynchronen Aktivitäten zur Verfügung. Die Software jedes Computerknotens (in weiterer Folge Node genannt) besteht aus den Application-Tasks, dem pSOS^{+m} und dem KI. Die Application-Tasks der einzelnen Nodes bedienen sich der Funktionalität von pSOS^{+m} um miteinander zu kommunizieren. pSOS^{+m} wiederum bedient sich der Funktionalität des KI, um über das physikalische Netzwerkmedium mit anderen Nodes Daten auszustauschen.[1] Mit Hilfe des KI wird pSOS^{+m} vom jeweils verwendeten Netzwerk unabhängig gemacht. Das KI ist daher jener Teil der pSOS^{+m}-Architektur, der vom NTI-Treiber zur Verfügung gestellt wird.

4.1.1 Funktionalität von pSOS^{+m}

Den pSOS^{+m}-Applikationen stehen die Objektklassen *Tasks, Memory Regions, Memory Partitions, Message Queues* sowie *Semaphore* für die Kommunikation, den Datenaustausch und die Synchronisation zur Verfügung.

Weiters ermöglicht pSOS^{+m} das Handling von *Events, Asynchronen Signalen* und *Interrupt Service Routinen (ISR).* Außerdem bietet dieses Betriebssystem noch ein *Time Management*, sowie ein *Fatal Error Handling.*

Tasks

pSOS^{+m} ermöglicht eine dynamische Erzeugung und Löschung von Tasks. Beim Starten von pSOS^{+m} wird der *ROOT Task* gestartet, von dem aus die jeweiligen benutzerdefinierten Application-Tasks gestartet werden können. Jeder Task besitzt eine *Priorität*, die dynamisch zur Laufzeit verändert werden kann.

Ein pSOS^{+m} Task besitzt die Task-States *Execution, Ready* und *Blocked.* pSOS^{+m} regelt das Umschalten zwischen den einzelnen Task-States. Ein Wechsel eines Tasks in einen anderen Task State wird durch systeminterne Ereignisse, wie z.B. das Warten auf eine Nachricht, hervorgerufen.

Jeder Task besitzt einen *Task Control Block (TCB)*, der alle Task-spezifischen Informationen enthält (z.B. Priorität, Name, Task Context, usw.).

Ein Task kann in verschiedenen Modi laufen:

- *Preemption* Enabled/Disabled: Im Modus Preemtion Disabled läuft ein Task so lange bis er in den Zustand Blocked versetzt wird, sogar wenn sich ein höher priorer Task im Zustand Ready befinden sollte.

[1] Es muß nicht unbedingt eine Netzwerk Architektur verwendet werden. Ein andere Alternative wäre z.B. eine *Shared Memory Architektur*, wie sie z.B. bei VME-Bus basierten Systemen leicht realisierbar ist.

- *Roundrobin* Enabled/Disabled: Im Falle von Roundrobin Enabled wird jedem Task ein *Time Slice* zugeteilt. Wenn dieser Time Slice abgelaufen ist, so wird ein neuer Task gleicher Priorizät zum Laufen gebracht.
- *ASR* Enabled/Disabled: ASR steht für *Asynchronous Signal Service Routine*. Asynchrone Signale können mit Software Interrupts verglichen werden. Wenn ASR auf Enabled gestellt ist, wird dem jeweiligen Task eine ASR zugeordnet.
- *Interrupt Mask Level*: Der Interrupt Mask Level gibt an, ab welchem Interrupt Level der Task unterbrochen werden kann. Dabei muß der Interrupt Level der ISR höher sein, als der Interrupt Mask Level des Task. Die Interrupt Level gehen von 0-7. Der Default Interrupt Mask Level ist 0.

Die Löschung eines Tasks sollte immer vom Task selbst erfolgen, damit bei der Löschung ein konstistenter Zustand gewährleistet werden kann.

Storage Allocation

Man unterscheidet zwischen *Memory Regions* und *Memory Paritions*. Beide können von den Tasks dynamisch angefordert und zum gemeinsamen Datenaustausch verwendet werden. Die Application-Tasks sind für die Mutual Exclusion der gemeinsamen Ressourcen verantwortlich.

Message Queues

Message Queues können zur Kommunikation und zur Synchronisation zwischen verschiedenen Tasks herangezogen werden. Message Queues sind nicht an einen bestimmten Task gebunden. Jeder Task der die *Queue ID* besitzt, kann von der Message Queue Nachrichten sowohl empfangen, als auch an diese senden.

Jede Queue besitzt einen *Queue Control Block*. Mittels der Queue ID kann auf den Queue Control Block zugegriffen werden. Dieser Zugriff kann auch von einem Task aus erfolgen, der sich nicht auf dem gleichen Node wie die Message Queue selbst befindet.

Es gibt Queue Operationen zum Senden und zum Empfangen von Nachrichten. Weiters ist es möglich dringende Nachrichten an den Beginn der Message Queue zu stellten. Ein *Broadcast Feature* ermöglicht es, eine Nachricht an alle Tasks, die momentan an der Queue auf eine Nachricht warten, zu verschicken.

Semaphore

Diese Funktionalität von pSOS^{+m} stellt die bekannten Semaphor-Operationen `sm_p` und `sm_v` zur Verfügung.

Genauso, wie bei Message Queues, besitzt auch ein Semaphor einen *Semaphor Control Block*, der über die Semaphor ID identifiziert wird. Auf einen Semaphor kann ebenfalls wieder von allen pSOS^{+m}-Nodes aus zugegriffen werden.

pSOS^{+m} Events

Mit der Hilfe von Events besteht die Möglichkeit einer *"Synchronization-by-Event"*. Jeder Task kann auf 32 verschiedene Events warten, wovon 16 von einer Benutzer Applikationen definiert werden können.

Events können sowohl "Oder" als auch "Und" verknüpft werden.

Asynchrone Signale (ASR)

Wie bereits bei den Tasks erwähnt, stellen ASR's die Möglichkeit zur Verfügung Software Interrupts zu definieren. Um das Handling von ASR's in einem Task zu ermöglichen, muß der entsprechende Task Mode eingestellt werden.

Time Management

pSOS^{+m} stellt ein Time Management zu Verfügung, das für pSOS interne Verwaltungsaufgaben, die etwa beim Scheduling oder bei Timeouts auftreten, verantwortlich ist. Unter anderem stellt das Time Management auch Funktionen wie *Time Wake After* oder *Time Wake When* zur Verfügung.

Es können mit Hilfe des pSOS^{+m} Time Managements auch Timer verwendet werden. Die Granularität dieser Timer ist jedoch nur 1/100 sec.

Interrupt Service Routines (ISR)

pSOS^{+m} stellt einen leistungfähigen Interrupt Mechanismus zur Verfügung. Beim Eintreten des Interrupts wird direkt über den Interrupt Vektor die entsprechende ISR angesprungen. Ein spezielles Augenmerk muß jedoch dem Austritt aus einer ISR geschenkt werden:

Interrupt Exit: Es gibt in pSOS^{+m} die beiden Möglichkeiten `I_RETURN` und `RTE`, um aus einer ISR zurückzukehren. Im Normalfall sollte `I_RETURN` verwendet werden. Dies gibt pSOS^{+m} die Möglichkeit, mit dem Task Scheduling oder mit einer weiteren ISR fortzufahren. Beim Aufruf von `RTE` wird die Task Steuerung nicht automatisch an pSOS^{+m} übergeben, sondern es wird an der jeweiligen Rücksprungadresse fortgefahren. Dazu folgende Erklärung:

Es kann durch einen System Call, wie z.B. `q_send` (senden einer Message an eine Queue), der aus einer ISR erfolgt, das Task Scheduling insoferne beeinflußt werden, als ein hochpriorer Task READY werden könnte. Da nun bei der Verwendung von `RTE` kein Task Scheduling erfolgt und bei der ursprünglichen Rücksprungadresse - also im vorher höchstprioren Task - fortgefahren wird, bedingt dies eine Verletzung der Regel des Static Priority Schedulings. Dieselbe Situation kann eintreten, wenn die ISR von einer anderen höherprioren ISR unterbrochen wird und diese einen System Call (SC), der das Task Scheduling beeinflußt, ausführt.

`RTE` darf daher nur dann verwendet werden, wenn die ISR mit dem höchsten Interruptlevel ausgeführt wird und selbst keinen Task-Switch durch einen SC hervorruft oder wenn gewährleistet ist, daß, sollte die ISR durch eine höher priore ISR unterbrochen werden, diese ebenfalls keine Aktionen ausführt, die zu einem Task-Switch führen können.

Tasks können Interrupts ausmaskieren, indem sie ihren Interrupt Mask Level entsprechend erhöhen.

Fatal Error Handling

Bei den meisten Errors handelt es sich um gewöhnliche Errors, die an den Caller der aufgerufenen Funktion zurückgeliefert werden.

Fatal Errors repräsentieren daher Fehler, die einen Einfluß auf das gesamten System haben (z.B. no resource error, startup error, usw.). Eine pSOS^{+m}-Application Software kann einen Fatal Error durch Aufruf des SC `k_fatal` generieren.

pSOS^{+m} springt daraufhin den `k_fatal` Entry an, in dem ein Fatal Error-Handler eingetragen ist. Wenn kein Eintrag vorhanden ist und der System Debugger pROBE^{+} existiert, so wird ihm die Kontrolle übergeben. Ansonsten wird von pSOS^{+m} eine "division-by-zero" durchgeführt.

4.1.2 KI Funktionalität für pSOS^{+m}

Objekte

pSOS^{+m} ist ein objektorientierter Kernel. Wie bereits erwähnt, handelt es sich bei den Objekten um Tasks, Memory Regions, Memory Partitions, Message Queues und Semaphore. Prinzipiell können Objekte von jedem Node aus angesprochen werden, wobei pSOS^{+m}-intern ggf. *Remote Procedure Calls (RPC's)* herangezogen werden müssen.

Man unterscheidet bei den genannten Objekten zwischen lokalen und globalen Objekten. Die *lokalen Objekte* sind nur einem einzelnen Node bekannt, wohingegen die *globalen Objekte* in allen Nodes des

Multiprozessor-Systems bekannt und verwendbar sind. Das pSOS^{+m} jedes Nodes führt dazu eine lokale und eine globale *Objekt Table*, über die Zugriffe auf Objekte eines *Remote Node* (entfernter Node) organisiert werden.

Jedem Objekt, egal ob global oder lokal, wird von pSOS^{+m} eine systemweit eindeutige *Objekt ID* zugeordnet. Wird nun ein globales Objekt erzeugt, so wird dessen Objekt ID in die globalen Object Table jedes Nodes eingetragen. Man bemerke, daß die Eintragung in den verteilten Nodes als "Atomic Action" zu erfolgen hat. Ebenso gilt dies für die Löschung eines globalen Objektes.

Für die Realisierung der diversen Operationen auf den globalen Objekten wird das KI benötigt. Dessen Aufgabe ist die Abwicklung der Node-to-Node Kommunikation bei folgenden Aktionen:

- Global Object Create
- Global Object Delete
- Global Startup and System Verification
- Global Shutdown
- Local Shutdown and Late Join
- Remote pSOS^{+m} Call (RPC)

Global Object Create

Wird auf einem pSOS^{+m}-Node ein globales Objekt erzeugt, so wird dieses Ereignis den restlichen pSOS^{+m}-Nodes durch eine spezielle Message (MSG) mitgeteilt. Jeder Node, der diese MSG empfängt, trägt das darin definierte Objekt in seiner global Object Table ein. Die Mitteilung kann am effizientesten durch eine Broadcast Message verteilt werden.

Global Object Delete

Das Löschen eines globalen Objektes kann nur von jenem Node aus erfolgen, auf dem das Objekt erzeugt und daher auch physikalisch gespeichert wurde. Die Mitteilung über den Löschvorgang erfolgt ebenfalls wieder mittels Broadcast einer speziellen MSG.

Global Startup and System Verification

Beim Global Startup werden die globalen Object Tables initialisiert. Nach der Initialisierungsphase erfolgt eine Global System Verification, bei der die Kohärenz der einzelnen Object Tables überprüft wird. Ab diesem Zeitpunkt können Higher-Level Startup Prozesse durchgeführt werden.

Global Shutdown

Tritt im pSOS^{+m}-System ein Fatal Error auf, so wird ein Global Shutdown eingeleitet. Recovery Mechanismen sind bis dato noch nicht vorhanden, sollten beim Entwurf des KI jedoch wenigstens prinzipiell einbezogen werden.

Local Shutdown and Late Join

Ein Local Shutdown dient dazu, um den lokalen Node aus dem pSOS^{+m}-System zu entfernen. Das Late Join wird zur Verfügung gestellt, damit ein pSOS^{+m}-Node zu einem beliebigen Zeitpunkt in eine bereits laufendes pSOS^{+m}-System eingehängt werden kann.

Remote pSOS^{+m} Calls (RPC)

RPC's sind Operationen, wie z.B. `q_receive`, `t_suspend`, `ev_send`, usw., die auf globalen Objekten durchgeführt werden (siehe auch [Int96a]).

4.2 Kurzbeschreibung von pNA$^+$ und Network Interface (NI)

pNA$^+$ ist ein High Performance Netzwerk Manager für das Echtzeitbetriebssystem pSOS$^+$. Mit Hilfe dieses Netzwerk Managers kann ein pSOS$^+$-System an andere Systeme, die das *Transmission Control Protocol - Internet Protocol (TCP/IP)* unterstützen, angekoppelt werden.

pNA$^+$ stellt somit eine Basis für verteilte heterogene Multiprozessor-Applikationen zur Verfügung. pNA$^+$ kann in Verbindung mit pHILE$^+$ und pRPC$^+$ dazu verwendet werden, um via *Network File System (NFS)* auf remote Files zugreifen zu können (siehe auch [Int96a]).

pNA$^+$ kann in vier Schichten unterteilt werden. Abbildung 4.3 zeigt die Architektur von pNA$^+$.

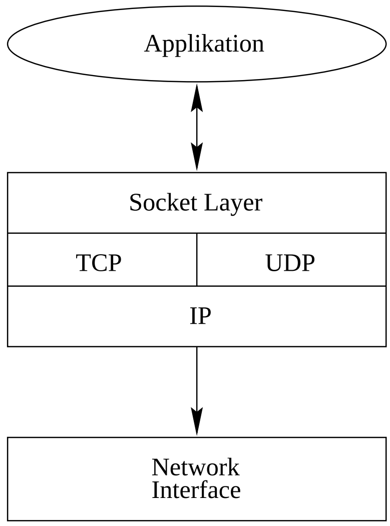


Abbildung 4.3: Architektur von pNA$^+$

Socket Layer

Der Socket Layer ist das *Application Programmers Interface (API)* von pNA$^+$. Sockets werden dazu verwendet, um eine Kommunikation zwischen zwei oder mehreren Applikationen aufzubauen. Ein Socket dient dabei als Kommunikations-Endpunkt.

Man kann Sockets entsprechend dem verwendeten Übertragungsprotokoll in folgende 3 Gruppen einteilen:

- *Stream Sockets* verwenden TCP und stellen daher eine verbindungsorientierte, sichere Kommunikation zur Verfügung.
- *Datagram Sockets* verwenden UDP, womit eine verbindungslose Kommunikationen durchgeführt werden kann.
- *Raw Sockets* ermöglichen den direkten Zugriff auf den IP Layer. Dies ermöglicht es dem User, ein anderes Protokoll als TCP/UDP zu verwenden.

User Signal Handler

pNA$^+$ stellt einen Satz von Signalen zur Verfügung, um auf asynchrone Ereignisse reagieren zu können (z.B. `SIGURG`, wenn dringende MSG's empfangen wurden).

Der User ist dafür verantwortlich, einen Signal Handler zu definieren, der entsprechende Aktionen beim Auftreten eines Signales startet. Die Adresse der Signalbehandlungsroutine wird in die pNA$^+$ Configuration Table unter `NC_SIGNAL` eingetragen.

Error Handling

pNA$^+$ verwendet den UNIX BSD 4.3 Socket Level Error Reporting Mechanismus: Error Codes werden in der Variable `errno` abgelegt.

Subcomponents

pNA$^+$ kann noch um sogenannte Subcomponents, wie z.B. RPC Services oder Netzwerk Management Tools, erweitert werden. Diese dienen dazu, um pNA$^+$ in seiner Leistungsfähigkeit zu erweitern. Subcomponents stehen voll unter der Kontrolle von pNA$^+$.

Subcomponents werden in die Subcomponent Configuration Table eingetragen, dessen Adresse wiederum in die pNA$^+$-Configuration Table unter `NC_CFGTAB` eingetragen wird. In die Subcomponent Configuration Table werden Pointer auf die einzelnen Subcomponents eingetragen.

Packet Routing

pNA$^+$ stellt einen kompletten Routing Mechnismus zur Verfügung. Um eine End-to-End Kommunikation zu bewerkstelligen, bedient sich pNA$^+$ der Einträge in einer Routingtabelle. Die Adresse der Routingtabelle muß in die pNA$^+$-Configuration Table unter `NC_ROUTE` eingetragen werden.

Dynamisch kann diese Tabelle mit Hilfe der Funktion `chng_route()` verändert werden. Ein Default Gateway kann in die Configuration Table unter `NC_DEFGN` eingetragen werden.

Address Resolution und ARP

Jedes NI besitzt eine Internet Adresse und eine Hardware Adresse. Da nun der IP Layer Internet Adressen und der Netzwerk Treiber Hardware Adressen verwendet, ist es notwendig, die Internet Adressen auf Hardware Adressen abzubilden.

Der Vorgang dieser Abbildung, der als Address Resolution bezeichnet wird, bedient sich zweier Methoden.

Zum einen wird eine ARP Tabelle zur Verfügung gestellt, in der sich Tupel der Form $< InternetAdresse$, $HardwareAdresse >$ befinden. Die Adresse dieser Tabelle wird in die pNA$^+$-Configuration Table unter `NC_IARP` eingetragen.

Wird kein entsprechendes Tupel in dieser Tabelle gefunden, so wird das Address Resolution Protocol (ARP) ausgeführt.

Beim ARP wird ein Paket, das die Internet-Adresse des gesuchten Nodes enthält, mit der Kennung ARP-Type an alle im Netz befindlichen Nodes geschickt. Beim Empfang eines ARP Paketes durch den gesuchten Node wird ein ARP Reply Packet mit der Hardware-Adresse zusammengestellt und zurückgeschickt. Es muß gewährleistet sein, daß alle im selben Netz befindlichen Stationen dieses Protokoll beherrschen.

ICMP

Dabei handelt es sich um ein Control and Error Message Protokoll für IP. Auf ICMP kann eine Applikation über Raw Sockets zugreifen. ICMP-Packets können z.B. dazu verwendet werden, um festzustellen, ob ein bestimmtes Netzwerk erreichbar ist.

Das bekannte `ping` Kommando verwendet dieses Feature.

pNA$^+$ Daemon Task

pNA$^+$ unterscheidet zwischen drei Arten von Aktivitäten, die bei der Bearbeitung eines pNA$^+$-System Calls (SC) notwendig werden können:

- Ein System Call wird sofort nach seinem Aufruf bedient, d.h., daß der aufrufende Task nach der Abarbeitung des SC mit seiner Arbeit fortsetzen kann.

- Ein SC wird erst nach längerer Verzögerung bedient (z.B. nach Retransmissions). Der aufrufende Task wird jedoch nicht blockiert und setzt seine Arbeit unmittelbar, ohne auf das Ergebnis des SC zu warten, fort.
- Ein SC wird nach längerer Verzögerung bedient, wobei der aufrufende Task das Ergebnis des SC abwarten muß, d.h. er wird blockiert. Nach Eintreffen des Ergebnisses kann der Task seine Arbeit fortsetzen.

Um nun die letzteren beiden Fälle behandeln zu können, wird bei der Initialisierung von pNA$^+$ ein eigener pNAD-Task (mit Priorität 255) generiert.

Der pNAD Task befindet sich normalerweise im Zustand Blockiert und wartet auf eines von zwei Events (welche in den Bits 30 und 31 kodiert sind).

Ist Bit 31 gesetzt, so bedeutet das, daß eine MSG empfangen wurde. Entsprechend dem Inhalt der MSG wird vom pNAD eine Aktion, wie z.B. das Aufwecken eines blockierten Tasks, durchgeführt.

Bit 32 wird alle 100 ms durch den pSOS^{+m}-System Call `tm_evevery` gesetzt. pNAD führt daraufhin zeitabhängige Aktivitäten für TCP durch. Diese Aktivitäten sind z.B. Retries und Timeouts.

Weiters wird von pNAD der SC `ni_poll` für jene NI's aufgerufen, die ihr POLL Flag gesetzt haben. Diese - nicht Interrupt gesteuerten - NI's benötigen `ni_poll` um nach eventuell eingetroffenen MSG's Ausschau zu halten.

Memory Management

pNA$^+$ verwendet intern ein eigenes Memory Management, um den für pNA$^+$ reservierten Speicher optimal zu nutzen und um Memory-to-Memory Copies weitgehend zu vermeiden.

Network Interface (NI)

pNA$^+$ greift auf den jeweiligen Netzwerk-Controller über das NI zu. pNA$^+$ wird also durch das NI von den speziellen Hardware-Charakteristika isoliert.

Beim NI handelt es sich daher um eine Schnittstelle zum eigentlichen Netzwerktreiber. Es ist möglich, mehrere NI's parallel zu betreiben. Jedes NI setzt auf seine eigene Hardware auf und muß somit eine eigene Internet Adresse besitzen.

Starting pNA$^+$

pNA$^+$ kann entweder von pSOS^{+m}, pROBE$^+$oder von beiden gestartet werden.

By pROBE$^+$: Soll Distributed Debugging durchgeführt und dabei mit dem XRAY$^+$ kommuniziert werden, so muß pNA$^+$ vom pROBE$^+$ gestartet werden. Zu beachten ist, daß in diesem Fall noch kein pSOS^{+m} gestartet wurde, und das NI daher weder pSOS^{+m}-SC's noch `I_RETURN` verwenden darf.

By pSOS^{+m} : Immer wenn pSOS^{+m} gestartet wird, wird auch pNA$^+$ gestartet. Sollte pNA$^+$ bereits vom pROBE$^+$ gestartet worden sein, so werden einfach bereits durchgeführte Initialisierungssequenzen übersprungen.

4.3 Beschreibung des Clock Interfaces (CI)

Dieser Teil des NTI-Treibers ist im Gegensatz zum KI und zum NI kein Teil von pSOS^{+m}, sondern er ist nötig, um die Ziele des Projekts SynUTC zu realisieren.

Ebenso wie das Kernel Interface die hardwareabhängige Schnittstelle zu pSOS^{+m} ist oder das Network Interface für pNA^{+}, so ist das Clock Interface die hardwarabhängige Schnittstelle zum NTI-Treiber. Das CI stellt die Möglichkeit zur Verfügung, Nachrichten mit Zeitstempeln zu senden bzw. zu empfangen und ist somit die unterste Schicht der Uhrensynchronisationsalgorithmen (siehe Abschnitt 1.2).

Die Uhrensynchronisation benötigt ein einfaches System um Nachrichten zu versenden, das die Möglichkeit bietet, hochpriore Nachrichten zu verschicken. Dieser Vorzug des NTI-Treibers sendet die Clock Synchronisation Packets (CSP) so schnell als möglich. Auf diese Weise kann die Verzögerung beim Broadcast, bei dem alle anderen Knoten verständigt werden, erheblich verkleinert werden.

Ein detaillierte Erklärung bzw. der genaue Aufbau des CI's und seiner Funktionen sind in [Ric97] und [SS99] dokumentiert.

4.4 VME-Prozessorboards

Das Evaluationssystem im Projekt SynUTC wird mit VME-Bus Boards aufgebaut. Der VME-Bus [DIN85], [Ass97] ist ein offenes, asynchrones Bussystem, bei dem bis zu 20 Prozessorboards über eine Backplane gemeinsam kommunizieren können. Wir benötigen drei verschiedene Typen von Boards, und zwar die Motorola MVME162-512A, auf der das pSOS^{+m} läuft, und passive Carrier-Boards, auf denen die NTI M-Module aufgesteckt werden. Der „Umweg“ über die passiven Carrier-Boards ist deswegen notwendig, da die MVME162-512A nicht mit M-Modulen bestückt werden kann.

4.4.1 Motorola MVME162-512A

Auf der Prozessorkarte MVME162-512A läuft das pSOS^{+m} mit dem NTI-Treiber und den Uhrensynchronisationsalgorithmen. Damit sowohl die 68040-CPU als auch der i82596-COMCO auf die NTI's zugreifen können, muß die Karte als VME-Bus-Master programmiert werden. Außerdem müssen die VME-Interrupts eingeschaltet werden, damit die NTI-Interrupts am Board auslösen können. Informationen zur MVME162-512A und zur Programmierung können in [Mot96a] bzw. [Mot96b] nachgelesen werden.

Alle für uns relevanten Einstellungen sind in den Listings `pcb_install_i6360.c` (siehe C.9) für das AcQ-i6360 Carrier-Board bzw. `pcb_install_a203.c` (siehe C.8) für die MEN A203 Karte im Anhang dokumentiert.

4.4.2 AcQ i6360 + i6Ext Extension Card

Das AcQ i6360 Board ist an sich ein Dual-Prozessor Board, für das es allerings leider noch kein pSOS^{+m} Board Support Package (BSP) gibt. Wir verwenden diese Boards deshalb vorerst nur als passive Carrier-Boards, die ihre MA-Bereiche auf den VME-Bus mappen, damit die MVME162 mit VME-Zugriffen die NTI's ansprechen kann. Diese VME-Karten können mit bis zu 6 M-Modulen bestückt werden.

Nähere Informationen sind in [AcQ96] beschrieben.

4.4.3 MEN A203

Die MEN A203 sind echte passive Carrier-Boards, d.h. sie haben keinen Prozessor, sondern nur eine Logik, die M-Module am VME-Bus zugreifbar machen. Im Gegensatz zu den AcQ-Boards funktionieren die VME-Zugriffe bei diesen Karten viel schneller.

In [MEN96] ist das MEN A203-Board ausreichend dokumentiert.

4.5 NTI-Handler

Der NTI-Handler ermöglicht den Zugriff auf ein oder mehrere Network Time Interface Module auf einem MEN A203 oder einem AcQ i6360 VME Carrier Board in einem MVME-162 basierenden pSOS^{+m} -System. Die Dateien des Handlers sind im Verzeichnis `$SynUTC/TECH/Software/NTI-Handler` abgelegt und im Kapitel C im Anhang ausgedruckt.

Der Network Time Interface - Handler besteht aus folgenden Teilen:

1. Definitionen für das NTI und die Carrier-Boards

`pcb_conf_i6360.h`	`pcb_conf.h` für das AcQ i6360 Carrier Board
`pcb_conf_a203.h`	`pcb_conf.h` für das MEN A203 Carrier Board
`nti_version.h`	Gültige Versionsbezeichnungen der NTI's und Carrier Boards
`nti.h`	Strukturen für alle NTI Versionen
`pcb_install.h`	Generische Datenstruktur für die NTI-Konfiguration
`utcsu.h`	UTCSU Register
`a203.h`	Register des MEN A203 Carrier Boards

2. Installation, Konfiguration and Interrupt Behandlung des NTI-Moduls

`ci_wrapper.c`	Generisches High-Level NTI Setup für den i82596 NTI-Treiber
`pcb_install_i6360.c`	Low-Level Setup für das AcQ i6360 Carrier Board
`pcb_install_a203.c`	Low-Level Setup für das MEN A203 Carrier Board

3. Sonstiges

`ntidet.s`	Funktionen vom pSOS^{+m} -BSP (nur für das Erzeugen von syn_app notwendig)

Die Applikationen müssen je nach verwendetem Carrier Board entweder das Header-File `pcb_confı6360.h` oder `pcb_conf_a203.h` inkludieren. Üblicherweise wird die verwendete Datei in `pcb_conf.h` umbenannt. Weiters ist es notwendig, die Dateien `ci_wrapper.c` und `pcb_install_i6360.c` oder `pcb_install_a203.c` zu compilieren und zu linken. Falsche oder inkompatible Benützung der Dateien führt zu Compile-Time Fehlern!

In den nächsten zwei Abschnitten wird eine kurze Einführung in die Arbeitsweise des NTI-Handlers gegeben, allerdings ersetzt er nicht die Kommentare in den C-Soure- und Headerfiles, in denen sehr ausführlich die Aufgaben der einzelnen Dateien erklärt werden.

4.5.1 Low-level Setup (`pcb_install_i6360.c`, `pcb_install_a203.c`)

Das Low-Level Setup der Carrier-Boards konfiguriert die ausgewählten und zur Laufzeit tatsächlich gefundenen NTI-Module. In die globale Variable `NTI_CONFIG ntidir[]` (siehe `pcb_install.h`, Abschnitt C.7) werden die Parameter (z.B. Interrupt Vector Base, UTCSU Clock Speed, etc.) eingetragen. Alle Einträge müssen initialisiert sein bevor die Initialisierungsfunktion `pcb_install()` aufgerufen wird. Diese Funktion setzt zuerst diverse Parameter (wie Startadresse der UTCSU, etc) in `nti_dir` und versucht dann die NTI-Module zu installieren. Nachdem der fehlerfreie Zugriff auf das Modul eines Slots verifiziert wurde, werden die NTI-Register programmiert sowie die UTCSU initialisiert und deren Uhr gestartet.

Danach werden die Interrupt-Vektoren in die Vektortabelle der CPU eingetragen, sofern das in `pcb_conf.h` ausgewählt ist. Abschließend werden noch die VME-Interrupts der MVME-162 aktiviert. Zu beachten ist, daß nach der Ausführung von `pcb_install` zwar die VME-Bus- und NTI-Interrupts aktiv sind, jedoch alle Interrupts der UTCSU abgeschaltet sind.

Weiters übernimmt das Low-Level Setup die grundlegende Interruptbehandlung indem es Interrupt Service Rountine Wrapper Funktionen bereitstellt, die externe C-Funktionen `int_nti`x`_`y aufrufen, wobei $x \in \{0, \ldots, 5\}$ (i6360) bzw. $\in \{0, \ldots, 3\}$ (A203) die NTI Slot-Nummer bezeichnet und $y \in \{\mathrm{n}, \mathrm{t}, \mathrm{a}\}$ für die Interrupts `INTN`, `INTT` und `INTA` der zugehörigen UTCSU steht. Diese ISR's können als normale C-Funktionen geschrieben werden, die nur die UTCSU-Interrupts, jedoch nicht die NTI-Interrupts, bearbeiten müssen.

Abschließend gibt es noch atomare Funktionen zum Ein- und Ausschalten der UTCSU-Interrupts, die als Read-Back Funktionen der UTCSU-Register `UTCINTEN1,2` fungieren.

4.5.2 High-level Setup (`ci_wrapper.c`)

Dieses generische Modul basiert auf dem Low-Level Setup und stellt das NTI-Setup für den NTI-Treiber [Ric97] zur Verfügung. Dieser Treiber bietet aus Kostengründen die Möglichkeit, mehrere NTI's auf einem einzigen Carrier-Board (= „Areas“) anzusprechen, um damit mehrere Knoten in einem Netzwerk zu emulieren. Dabei werden nur eine CPU und ein COMCO für mehrere NTI-Module verwendet (siehe Abbildung 4.2). Beim Empfangen von Datenpaketen weiß der COMCO natürlich nicht, welchem Knoten das Paket zugeordnet werden muß, deshalb werden alle Emfangsdaten in die Area 0 geschrieben. Weiters sind alle SSU1 Receive Timestamp Trigger Eingänge miteinander verbunden und werden vom CPLD der Area 0 getrieben. Zusätzlich darf nur das NTI-Modul der Area 0 den Receive Timestamp Interrupt `INTT` erzeugen. Dann ist es die Aufgabe des Treibers in der Interrupt Service Routine (ISR) die Empfangsdaten richtig zuzuordnen.

Das Modul `ci_wrapper.c` stellt eine Funktion `nti_init` zur Verfügung, die alle NTI-Module entsprechend initialisiert. Zusätzlich ist in dieser Datei die Funktion `INT_ISR` implementiert, welche die Receive Timestamps aller Areas in die entsprechenden Receive Header kopiert. Mit der Funktion `gps_isr` werden Interrupts der GPS-Units (GPU1-3) an Anwendungsfunktionen weitergeleitet. Abschließend gibt es in diesem Modul Wrapper-ISR's für `INTA`- und `INTN`-Interrupts, die eine benutzerdefinierte Interrupt Service Routine `ci_int`x aufrufen, sofern kein `INTT`-Interrupt anliegt.

Ganz wichtig ist, daß alle Interrupt Service Routinen (`ci_intt`, `ci_inta`, `gps_isr` so kurz als möglich sind, da damit das Verschieben der Zeitstempel in der ISR `isr_n` verzögert wird und Receive Timestamps verloren gehen könnten. Die Ausführungszeiten der ISR's sollten unbedingt $< 50 \ldots 100\ \mu$s sein!

Kapitel 5

Programmiermodell

In diesem Kapitel wird eine detaillierte Beschreibung zur Programmierung des NTI-Moduls gegeben. Am Beginn wird die Memory-Map des NTI tabellarisch dargestellt. Der nächsten Unterabschnitt beinhaltet eine Gesamtübersicht aller UTCSU-Register. Nach diesem Übersichtteil werden alle Funktionen des Network Time Interfaces, die nicht direkt die UTCSU betreffen, ausführlich beschrieben.

5.1 Speicherbelegung

Die Tabelle 5.1 beinhaltet die Speicheraufteilung des NTI-RAMs. Die Adreßbits `A31...A20` hängen von der Programmierung des Carrier-Boards bzw. vom MA-Slot ab, auf dem das NTI sitzt. Diese Bits sind in der Tabelle mit `'X'` bezeichnet. In den Abschnitten 4.4.2 und 4.4.3 wird die Berechnung der Adressen erklärt.

Die Adreßleitung `A19` wird verwendet, um zwischen CPU- und COMCO- Speicherzugriffen zu unterscheiden. Wenn `A19='0'` ist, dann erfolgt ein COMCO-Zugriff auf den Speicher. Bei den Transmit- bzw. Receive-Header wird die Zeitstempelung und das Memory-Remapping zu den dafür vorgesehenen UTCSU-Registern vorgenommen. Ist `A19='1'`, dann wird von der CPU auf den Speicher zugegriffen (vgl. Abbildung 3.8).

Die höchsten 512 Byte des 512 KByte[1] großen SRAM-Speichers werden für die UTCSU benötigt. Diese kann nur von der CPU angesprochen werden.

Startadresse	**Endadresse**	**Größe**	**Komponente**	**Funktion**
`0xXXX0 0000`	`0xXXX0 1FFF`	8 KByte	Speicher (COMCO)	Transmit-Header
`0xXXX0 2000`	`0xXXX1 FFFF`	120 KByte	Speicher (COMCO)	Receive-Header
`0xXXX2 0000`	`0xXXX7 BFFF`	368 KByte	Speicher (COMCO)	Data Buffers
`0xXXX7 C000`	`0xXXX7 FDFF`	15,5 KByte	Speicher (COMCO)	Systemstructures
`0xXXX7 FE00`	`0xXXX7 FFFF`	512 Byte	nicht benützt	-
`0xXXX8 0000`	`0xXXX8 1FFF`	8 KByte	Speicher (CPU)	Transmit-Header
`0xXXX8 2000`	`0xXXX9 FFFF`	120 KByte	Speicher (CPU)	Receive-Header
`0xXXXA 0000`	`0xXXXF BFFF`	368 KByte	Speicher (CPU)	Data Buffers
`0xXXXF C000`	`0xXXXF FDFF`	15,5 KByte	Speicher (CPU)	Systemstructures
`0xXXXF FE00`	`0xXXXF FFFF`	512 Byte	UTCSU	UTCSU-Register [a]

[a] Die genaue Registerbelegung des 512 Byte großen UTCSU Adreßraumes ist im Abschnitt 5.2 auf den nächsten beiden Seiten dargestellt.

Tabelle 5.1: NTI Speicherbelegung und Adreßbereiche

[1] Bei den Modulen, die für die Evaluation im SynUTC-Projekt benötigt werden, sind nur 256 KByte SRAM bestückt (und zwar die IC's *U12* und *U14*). Die vier Teilbereiche *Transmit Header, Receive Header, Data Buffer und System Structures* müssen jeweils halbiert werden.

5.2 Registerbelegung der UTCSU

Die Tabellen 5.2 und 5.3 stellen eine nach funktionalen Blöcken geordnete Gesamtübersicht aller UTCSU-Register dar. Adressen die mit '-' gekennzeichnet sind bezeichnen keine gültigen UTCSU Register und sind für zukünftige Erweiterungen vorgesehen. Alle Register wurden für 32 Bit Datenzugriffe entworfen. Bei 16 bzw. 8 Bit Datenzugriffen muß auf Big-Endian bzw. Little-Endian Byteordnung besonders geachtet werden (siehe dazu [Loy96], Kapitel 7.1.1).

Address	Bits	READ-Register Name	Bits	WRITE-Register Name	Unit
0x000	[23:0]	MSGET - Macro Stamp Get	[23:0]	MSSET - Macro Stamp Set	**LTU**
0x000	[31:24]	CS - Checksum Get	[31:24]	STEPLOW - Step Low	**LTU**
0x004	[31:0]	TSGETL - Time Stamp Get Latch	[31:0]	TSSET - Time Stamp Set	**LTU**
0x008	[31:0]	USGET - Micro Stamp Get	[31:0]	USSET - Micro Stamp Set	**LTU**
0x00C	[31:0]	TSGET - Time Stamp Get	[7:0]	NTPSET - NTP-Time Set	**LTU**
0x010	[2:0]	NSGET - Nano Stamp Get	[31:0]	STEPAMORT - Step Amort. Ph.	**LTU**
0x014	[31:0]	–	[31:0]	STEPPURE - Step Pure Phase	**LTU**
0x018	[31:0]	AMORTTIMEGETL	[31:0]	AMORTTIMESET	**LTU**
0x01C	[31:0]	AMORTTIMEGET	[23:0]	STARTAMORT	**LTU**
0x020	[15:0]	STATEPGETL	[15:0]	LAMBDAPPURE	**ACU**
0x020	[31:16]	STATENGETL	[31:16]	LAMBDAPAMORT	**ACU**
0x024	[15:0]	STATEPGET	[15:0]	LAMBDANPURE	**ACU**
0x024	[31:16]	STATENGET	[31:16]	LAMBDANAMORT	**ACU**
0x028	[31:0]	ALPHAPGETL	[31:0]	STATEPSET	**ACU**
0x02C	[31:0]	ALPHANGETL	[31:0]	STATENSET	**ACU**
0x030	[31:0]	ALPHAPGET	[15:0]	BOUNDP	**ACU**
0x030	–	–	[31:16]	BOUNDN	**ACU**
0x034	[31:0]	ALPHANGET	[31:0]	ALPHASET	**ACU**
0x038	[15:0]	NALPHAPGET	[7:0]	ALPHAPNSET	**ACU**
0x038	[31:16]	NALPHANGET	–	–	**ACU**
0x040	–	–	[7:0]	SWRESET	**CLU**
0x060	[31:0]	IDENTIFICATION	[31:0]	UTCCONF1	**ITU**
0x064	[31:0]	UTCSTAT1	[31:0]	UTCCONF2	**ITU**
0x068	[31:0]	UTCSTAT2	[31:0]	UTCINTEN1 - Int. Enable 1	**ITU**
0x06C	[31:0]	UTCCONF1	[31:0]	UTCINTEN2 - Int. Enable 2	**ITU**
0x070	[31:0]	UTCCONF2	[31:0]	UTCINTCLEAR1 - Int. Clear 1	**ITU**
0x074	[31:0]	UTCINTSTAT1	[31:0]	UTCINTCLEAR2 - Int. Clear 2	**ITU**
0x078	[31:0]	UTCINTSTAT2	–	–	**ITU**
0x080	[23:0]	MSSIG - Macro Stamp MISR sign.	[15:0]	TESTSEL - Test Select	**BTU**
0x084	[31:0]	TSSIG - Time Stamp MISR sign.	[7:0]	TESTGATE - Test Gate Pulse	**BTU**
0x088	[31:0]	ACCSIG - Accuracy MISR sign.	–	–	**BTU**
0x08C	[31:0]	MSSUM - Macro Stamp Blocksum	–	–	**BTU**
0x090	[31:0]	TSSUM - Time Stamp Blocksum	–	–	**BTU**
0x094	[31:0]	ACCSUM - Accuracy Blocksum	–	–	**BTU**
0x098	[31:0]	PUREPHASE - Purephase Counter	–	–	**BTU**
0x0A0	[31:0]	MSSNU - Macro Stamp Snapshot	[31:0]	SWSNAP - Software Snapshot	**SNU**
0x0A4	[31:0]	TSSNU - Time Stamp Snapshot	[31:0]	DUTYH - Duty-Timer High Prel.	**SNU**
0x0A8	[15:0]	ACCPSNU - Accuracy P Snapshot	[15:0]	DUTYL - Duty-Timer Low Prel.	**SNU**
0x0A8	[31:16]	ACCNSNU - Accuracy N Snapshot	[16]	E - Enable Bit Duty-Timer	**SNU**
0x0C0	[31:0]	MSAPP - Macro Stamp Appl.	[31:0]	DUTYH - Duty-Timer High Prel.	**APU**
0x0C4	[31:0]	TSAPP - Time Stamp Appl.	[16:0]	DUTYL - Duty-Timer Low Prel.	**APU**
0x0C8	[15:0]	ACCPAPP - Accuracy P Appl.	[2:0]	APPSEL - Application Select	**APU**
0x0C8	[31:16]	ACCNAPP - Accuracy N Appl.	–	–	**APU**
0x0CC	[31:0]	MSAPPX - Macro Stamp Appl. X	[7:0]	APPCLEAR	**APU**
0x0D0	[31:0]	TSAPPX - Time Stamp Appl. X	–	–	**APU**
0x0D4	[15:0]	ACCPAPPX - Accuracy P Appl. X	–	–	**APU**
0x0D4	[31:16]	ACCNAPPX - Accuracy N Appl. X	–	–	**APU**
0x0E0	[31:0]	MSGPS - Macro Stamp GPS	–	–	**GPU1**
0x0E4	[31:0]	TSGPS - Time Stamp GPS	–	–	**GPU1**

Tabelle 5.2: UTCSU Registerbelegung Teil 1 (0x000 - 0x0FF)

Address	Bits	READ-Register Name	Bits	WRITE-Register Name	Unit
0x100	[31:0]	MSGPS - Macro Stamp GPS	–	–	**GPU2**
0x104	[31:0]	TSGPS - Time Stamp GPS	–	–	**GPU2**
0x120	[31:0]	MSGPS - Macro Stamp GPS	–	–	**GPU3**
0x124	[31:0]	TSGPS - Time Stamp GPS	–	–	**GPU3**
0x140	[31:0]	MSRCV - Macro Stamp Receive	[31:0]	DUTYAH - Duty-Timer A High	**SSU1**
0x144	[31:0]	TSRCV - Time Stamp Receive	[16:0]	DUTYAL - Duty-Timer A Low	**SSU1**
0x148	[31:0]	MSXMT - Macro Stamp Transmit	[31:0]	DUTYBH - Duty-Timer B High	**SSU1**
0x14C	[31:0]	TSXMT - Time Stamp Transmit	[16:0]	DUTYBL - Duty-Timer B Low	**SSU1**
0x150	[15:0]	ACCPXMT - Accuracy P Transmit	–	–	**SSU1**
0x150	[31:16]	ACCNXMT - Accuracy N Transmit	–	–	**SSU1**
0x154	[31:0]	ACCNPNPLXMT	–	–	**SSU1**
0x158	[31:0]	ACCNPNPHXMT	–	–	**SSU1**
0x160	[31:0]	MSRCV - Macro Stamp Receive	[31:0]	DUTYAH - Duty-Timer A High	**SSU2**
0x164	[31:0]	TSRCV - Time Stamp Receive	[16:0]	DUTYAL - Duty-Timer A Low	**SSU2**
0x168	[31:0]	MSXMT - Macro Stamp Transmit	[31:0]	DUTYBH - Duty-Timer B High	**SSU2**
0x16C	[31:0]	TSXMT - Time Stamp Transmit	[16:0]	DUTYBL - Duty-Timer B Low	**SSU2**
0x170	[15:0]	ACCPXMT - Accuracy P Transmit	–	–	**SSU2**
0x170	[31:16]	ACCNXMT - Accuracy N Transmit	–	–	**SSU2**
0x174	[31:0]	ACCNPNPLXMT	–	–	**SSU2**
0x178	[31:0]	ACCNPNPHXMT	–	–	**SSU2**
0x180	[31:0]	MSRCV - Macro Stamp Receive	[31:0]	DUTYAH - Duty-Timer A High	**SSU3**
0x184	[31:0]	TSRCV - Time Stamp Receive	[16:0]	DUTYAL - Duty-Timer A Low	**SSU3**
0x188	[31:0]	MSXMT - Macro Stamp Transmit	[31:0]	DUTYBH - Duty-Timer B High	**SSU3**
0x18C	[31:0]	TSXMT - Time Stamp Transmit	[16:0]	DUTYBL - Duty-Timer B Low	**SSU3**
0x190	[15:0]	ACCPXMT - Accuracy P Transmit	–	–	**SSU3**
0x190	[31:16]	ACCNXMT - Accuracy N Transmit	–	–	**SSU3**
0x194	[31:0]	ACCNPNPLXMT	–	–	**SSU3**
0x198	[31:0]	ACCNPNPHXMT	–	–	**SSU3**
0x1A0	[31:0]	MSRCV - Macro Stamp Receive	[31:0]	DUTYAH - Duty-Timer A High	**SSU4**
0x1A4	[31:0]	TSRCV - Time Stamp Receive	[16:0]	DUTYAL - Duty-Timer A Low	**SSU4**
0x1A8	[31:0]	MSXMT - Macro Stamp Transmit	[31:0]	DUTYBH - Duty-Timer B High	**SSU4**
0x1AC	[31:0]	TSXMT - Time Stamp Transmit	[16:0]	DUTYBL - Duty-Timer B Low	**SSU4**
0x1B0	[15:0]	ACCPXMT - Accuracy P Transmit	–	–	**SSU4**
0x1B0	[31:16]	ACCNXMT - Accuracy N Transmit	–	–	**SSU4**
0x1B4	[31:0]	ACCNPNPLXMT	–	–	**SSU4**
0x1B8	[31:0]	ACCNPNPHXMT	–	–	**SSU4**
0x1C0	[31:0]	MSRCV - Macro Stamp Receive	[31:0]	DUTYAH - Duty-Timer A High	**SSU5**
0x1C4	[31:0]	TSRCV - Time Stamp Receive	[16:0]	DUTYAL - Duty-Timer A Low	**SSU5**
0x1C8	[31:0]	MSXMT - Macro Stamp Transmit	[31:0]	DUTYBH - Duty-Timer B High	**SSU5**
0x1CC	[31:0]	TSXMT - Time Stamp Transmit	[16:0]	DUTYBL - Duty-Timer B Low	**SSU5**
0x1D0	[15:0]	ACCPXMT - Accuracy P Transmit	–	–	**SSU5**
0x1D0	[31:16]	ACCNXMT - Accuracy N Transmit	–	–	**SSU5**
0x1D4	[31:0]	ACCNPNPLXMT	–	–	**SSU5**
0x1D8	[31:0]	ACCNPNPHXMT	–	–	**SSU5**
0x1E0	[31:0]	MSRCV - Macro Stamp Receive	[31:0]	DUTYAH - Duty-Timer A High	**SSU6**
0x1E4	[31:0]	TSRCV - Time Stamp Receive	[16:0]	DUTYAL - Duty-Timer A Low	**SSU6**
0x1E8	[31:0]	MSXMT - Macro Stamp Transmit	[31:0]	DUTYBH - Duty-Timer B High	**SSU6**
0x1EC	[31:0]	TSXMT - Time Stamp Transmit	[16:0]	DUTYBL - Duty-Timer B Low	**SSU6**
0x1F0	[15:0]	ACCPXMT - Accuracy P Transmit	–	–	**SSU6**
0x1F0	[31:16]	ACCNXMT - Accuracy N Transmit	–	–	**SSU6**
0x1F4	[31:0]	ACCNPNPLXMT	–	–	**SSU6**
0x1F8	[31:0]	ACCNPNPHXMT	–	–	**SSU6**

Tabelle 5.3: UTCSU Registerbelegung Teil 2 (0x100 - 0x1FF)

In der Datei `utcsu.h` (siehe Abschnitte 4.5 bzw. C.10) sind die Datenstrukturen für 8, 16, und 32 Bit Schreib- und Lesezugriffe deklariert.

5.3 Reset

Ein Hardware-Reset bewirkt, daß am gesamten NTI-Board ein Reset durchgeführt wird, d.h. sowohl die UTCSU als auch das CPLD setzen ihre Register in den Ruhezustand. Diese Art der Rücksetzung kann durch Drücken des Reset-Tasters am Carrier-Board oder am VME-Bus Master erzeugt werden. Beim Einschalten der Spannungsversorgung wird selbstverständlich auch ein Reset im CPLD und in der UTCSU durchgeführt.

5.4 Programmiermodell der I/O-Space Register

In diesem Abschnitt werden die I/O-Space Register der NTI in der gleichen Reihenfolge wie in Abschnitt 3.4, also beginned beim Register `RCV-HEADER BASE` an der Adresse `0x02` bis zu den `SPROM`-Registern an der Adresse `0xFE`, beschrieben. Neben dem genauen Registeraufbau und einer nochmaligen kurzen funktionellen Beschreibung werden Beispiele für die Programmierung gegeben.

5.4.1 Schreib- und Lesezugriffe auf NTI-Register

In der Datei `nti.h` (siehe Abschnitt C.3) sind zwei Strukturen für die I/O-Space Zugriffe definiert, und zwar `NTI_IO_SPACE16` für 16 Bit Schreib- und Lesezugriffe und `NTI_IO_SPACE32` für Schreib- und Lesezugriffe mit 32 Bit Datenbreite.

Zugriffe mit 16 Bit Datenbreite

Die Datenstruktur `NTI_IO_SPACE16` repräsentiert den I/O-Space des NTI-Moduls als C-Code (vgl. Abbildung 3.7). Alle Register sind als `INT16` definiert, das entspricht einem `unsigned short`, also einem nicht vorzeichenbehafteten 16 Bit breiten Integer-Wert (Wertebereich: `0 - 65535`). Alle mit `unused`x benannten Elemente kennzeichnen nicht verwendete Speicherplätze im I/O-Space.

```
typedef volatile struct
{
  INT16 unused1;
  INT16 rcv_header_base;    /* Adresse des zuletzt empfangenen Datenpakets */
  INT16 unused2;
  INT16 vector_base;        /* Interrupt-Vector Base */
  INT16 unused3;
  INT16 int_logic;          /* Enable/Disable NTI-Interrupts */
  INT16 unused4;
  INT16 la_trigger;         /* Trigger Signal f"ur Logic Analyzer */
  INT16 unused5;
  INT16 utcsu_misc;         /* Setting statischer UTCSU-Signale */
  INT16 unused6;
  INT16 ts_offsets;         /* Trigger-Offsets f"ur Transmit/Receive-TS  */
                            /*                        (Upper/Lower Byte) */
  INT16 unused7;
  INT16 type_offset;        /* Offset Type-Field in Rec.-Header (Lower Byte) */
  INT16 unused8[0x80-14-1];
  INT16 s_prom;             /* ID Serial PROM */
} NTI_IO_SPACE16;
```

Nun folgen ein paar Beispiele für Zugriffe auf I/O-Space Register, bei denen jeweils die Variable `tmp16` vom Typ `INT16` ist.

Beispiel 1:

```
ntidir[0].io->la_trigger = 0x789A;
tmp16 = ntidir[0].io->la_trigger;
```

Das erste Beispiel ist eher konservativ, da das entsprechende Register `la_trigger` *über die Strukturen des NTI-Handlers (siehe Abschnitt 4.5) angesprochen wird. In der ersten Zeile wird in das Register der*

Wert `0x789A (= 30874)` *geschrieben und dann in der zweiten Zeile wieder ausgelesen. Dieses Register wurde hier, so wie auch in den anderen Beispielen dieses Abschnittes, ganz zufällig ausgewählt, ohne auf die tatsächliche Funktionalität einzugehen.*

Beispiel 2:

```
(NTI_IO_SPACE16 *)0xE0700000 = 0x789A;
tmp16 = (NTI_IO_SPACE16 *)0xE0700000;
```

Im zweiten Beispiel wird wieder die Struktur `NTI_IO_SPACE16` *verwendet, allerdings wird sie über die Adresse direkt referenziert. Diese Vorgangsweise ist für Testprogramme sehr sinnvoll, bei denen nicht der NTI-Handler benötigt wird, die aber trotzdem strukturiert programmiert sein sollen. Anstatt die Adresse direkt zu verwenden, kann selbstverständlich auch eine Variable vom Typ* `INT32` *eingesetzt werden, der die Adresse vorher zugewiesen wurde.*

Beispiel 3:

```
(INT16 *)0xE070000E = 0x789A;
tmp16 = (INT16 *)0xE070000E;
```

Beim dritten Beispiel wird auf dasselbe Register zugegriffen, allerdings direkt über die Adresse mit einem Zeiger auf INT16.[2] *Diese Art der Zugriffe ist für kleine Tests oder für die direkte Eingabe bzw. Abfrage durch den Debugger sehr nützlich.*

Zugriffe mit 32 Bit Datenbreite

Mit Hilfe der Datenstruktur `NTI_IO_SPACE32` kann auf die I/O-Space Register des NTI-Moduls mit 32 Bit Datenbreite zugegriffen werden. Alle Register sind als INT32 deklariert, das entspricht einem nicht vorzeichenbehafteten 32 Bit breiten Integer-Wert.

```
typedef volatile struct
{
  INT32 rcv_header_base;    /* Adresse des zuletzt empfangenen Datenpakets */
  INT32 vector_base;        /* Interrupt-Vector Base */
  INT32 int_logic;          /* Enable/Disable NTI-Interrupts */
  INT32 la_trigger;         /* Trigger Signal f"ur Logic Analyzer */
  INT32 utcsu_misc;         /* Setting statischer UTCSU-Signale */
  INT32 ts_offsets;         /* Trigger-Offsets f"ur Transmit/Receive-TS  */
                            /*                         (Upper/Lower Byte) */
  INT32 type_offset;        /* Enable + Offset Type-Field in Rec.-Header */
  INT32 unused[0x40-8];
  INT32 s_prom;             /* ID Serial PROM */
} NTI_IO_SPACE32;
```

Hier wird nur ein Beispiel für die Zugriffe auf die I/O-Space Register gegeben, da sie analog zu den 16 Bit Zugriffen funktionieren.

Beispiel:

```
ntidir[0].io->la_trigger = 0x0000789A;
tmp16 = ntidir[0].io->la_trigger;
```

Wichtig beim Schreiben ins Register ist, daß die untersten 16 Datenbits ins Register geschrieben werden und die oberen 16 Datenbits ohne Bedeutung sind!

Ob die 16 Bit Zugriffe mit der Struktur `NTI_IO_SPACE16` oder die 32 Bit Zugriffe über die Struktur `NTI_IO_SPACE32` verwendet werden, muß statisch in der Datei `nti.h` (Abschnitt C.3) festgelegt werden. In der Zeile 405 steht

```
typedef NTI_IO_SPACE16 NTI_IO_SPACE;
```

womit die 16-Bit Struktur standardmäßig verwendet wird.

[2] Diese Adresse gilt aber nur für ein AcQ i6360 Carrier-Board, dessen Basisadresse auf `0xE0000000` gesetzt ist und auf dessen NTI-Modul im Slot 2 zugegriffen wird. Nähere Informationen über die Carrier-Boards und ihre Adreßräume stehen in den Abschnitten 4.4.2 und 4.4.3.

5.4.2 NTI-Interrupt Programmierung

In diesem Abschnitt werden ausschließlich die Programmierung und die Funktionen der NTI-Register erklärt und mit Beispielen illustriert. Weder die Erklärung des Interruptzyklus (Abschnitt 3.6) noch die Programmierung der UTCSU-Interrupts ([Loy96], Kapitel 8.8) sind Gegenstand der Ausführungen.

	15:12	11:0	**Read**
RCV-HEADER BASE	–	Receive-Header Base [11:0]	**0x00**
	–	0x000	Reset

Abbildung 5.1: `RCV-HEADER BASE` - Register `0x00` im I/O-Space

Das 12-Bit Register `RCV-HEADER BASE` an der Speicherstelle `0x00` im I/O-Space speichert in den Bits `[11:0]` die Adressen des Headers des zuletzt empfangenen Datenpakets. Konkret werden die Adreßleitungen `A16`, ..., `A5` beim Paketempfang in das Register gespeichert, d.h. um die Adresse des Pakets zu berechnen, muß der Registerwert mit 32 multipliziert werden (oder shift left um 5). Beim Hardware-Reset des Carrier-Boards wird dieses Register gelöscht.

Beispiel:

```
rcv_header = ntidir[0].io->rcv_header_base << 5;
```

Die tatsächliche Adresse des zuletzt empfangenen Receive-Headers erhält man, indem das Register ausgelesen und um fünf Stellen nach links geschoben (oder mit 32 multipliziert) wird. Schreibzugriffe auf das Register haben keine Auswirkung.

	15:8	7:3	2:0	**Write**
SET INT.-VECTOR BASE	-	Int. Vector Base [7:3]	-	**0x02**
	–	0b01000	–	Reset

Abbildung 5.2: `SET INT.-VECTOR BASE` - Register `0x02` im I/O-Space

Das Register `SET INT.-VECTOR BASE` an der Speicherstelle `0x02` im I/O-Space ermöglicht das Setzen der obersten 5-Bits des 8-Bit breiten Interrupt-Vektors. Dies geschieht in den Bits `[7:3]`. In den untersten 3 Bits `[2:0]` sind die invertierten Pegel der UTCSU Interruptleitungen $\overline{\texttt{INTN}}$, $\overline{\texttt{INTT}}$ und $\overline{\texttt{INTA}}$ fest kodiert. Beim Reset wird dieses Register auf `0x40` gesetzt, da das der Beginn der M68K User Interrupt Vektoren ist.

Beispiel:

```
ntidir[0].io->vector_base = 0x0080;
ntidir[0].io->vector_base = 0x00A8;
```

Im ersten Fall wird das Register auf den Wert `0x0080` *gesetzt, sodaß alle von der NTI ausgelösten und gültigen Interrupts im Bereich* `0x81`...`0x87` *liegen. Im zweiten Fall wird der Vektor* `0x00A8` *programmiert, womit nur Interrupts im Bereich* `0xA9`...`0xAF` *ausgelöst werden können. Wichtig ist, daß sich die Interrupts nicht mit ebensolchen des Betriebssytems pSOS^{+m}, anderen NTI-Interrupts oder sonstigen Treibern überschneiden dürfen. Vor dem Aktivieren der Interrupts müssen die ISR's in die Vektortabelle der CPU eingetragen werden (geschieht beim NTI-Handler automatisch in der Funtion* `pcb_install()`*, siehe Abschnitt 4.5).*

	15	14:8	7:3	2:0	**Read**
GET INT.-VECTOR BASE	E	0000000	Int. Vector Base [7:3]	Int.-Vector	**0x02**
	0	0b0000000	0b01000	0b000	Reset

Abbildung 5.3: `GET INT.-VECTOR BASE` - Register `0x02` im I/O-Space

Dieses Register liegt ebenso wie das Register `SET INT.-VECTOR BASE` an der Speicherstelle `0x02` im I/O-Space. Die Bits `[7:0]` des Registers `GET INT.-VECTOR BASE` beinhalten den Interrupt-Vektor des MA-Moduls. Damit kann überprüft werden, welche Basisadresse gesetzt ist und ob Interruptquellen aktiv sind. Im Bit `15` ist das `ENABLE`-Signal der Interrupt-Logik eingeblendet, damit getestet werden kann, ob die Interrupt-Logik aktiv ist. Die Interrupt-Logik ist dann aktiv, wenn das `ENABLE`-Bit `E` den Wert `'1'` hat.

Beispiel 1:

```
tmp16 = ntidir[0].io->vector_base & 0x00FF;
if (tmp16 & 0x0004)
 ... /* INTN-Interrupt liegt an */
if (tmp16 & 0x0002)
 ... /* INTT-Interrupt liegt an */
if (tmp16 & 0x0001)
 ... /* INTA-Interrupt liegt an */
```

In diesem Beispiel soll gezeigt werden, wie man den Interruptvektor ausliest und auswertet. Beim Lesen des Registers werden die obersten 8 Bit maskiert damit ein gültiger 8-Bit Vektor entsteht, falls die Interrupt-Logik (und damit das Bit 15) aktiv ist. Nach einem Reset würde in der Variable `tmp16` *der Wert* `0x40` *stehen. Liest man aber z.B. den Wert* `0x86` *aus, dann bedeutet das, daß die Vektorbasis* `0x80` *ist und sowohl der INTN-Interrupt (Bit 2) als auch der INTT-Interrupt (Bit 1) anliegen.*

Beispiel 2:

```
#define NTI_INT_LOGIC_ON (1 << 15)

tmp16 = ntidir[0].io->vector_base;
if (tmp16 & NTI_LOGIC_ON)
 ... /* Interrupt-Logik aktiv */

oder

if (tmp16 & 0x8000)
 ... /* Interrupt-Logik aktiv */
```

Beispiel 2 zeigt, wie man prüft, ob die Interrupt-Logik aktiv ist. Entweder kann man aus der Datei `nti.h` *das* `#define NTI_INT_LOGIC_ON` *oder den Wert 0x8000 zum maskieren des Bits verwenden.*

	15:0	0	**Write**
EN/DIS INT.-LOGIC	-	S	**0x04**
	–	-	Reset

Abbildung 5.4: `EN/DIS INT.-LOGIC` - Register `0x04` im I/O-Space

Beim Schreiben des niederwertigsten Bits `S` des `EN/DIS INT.-LOGIC` Registers an der Speicherstelle `0x04` im I/O-Space kann die Interruptgenerierung des NTI MA-Moduls ein- bzw. ausgeschaltet werden. Wird das `S`-Bit auf `'1'` gesetzt, so ist die Interrupt-Logik aktiv. Durch Schreiben von `'0'` wird die Interrupt-Logik abgeschaltet. Das CPLD schaltet die Interrupt-Logik automatisch ab, wenn ein Interrupt ausgelöst wurde (siehe Kapitel 3.6).

Beispiel:

```
#define NTI_DIS_INT_LOGIC 0
#define NTI_EN_INT_LOGIC 1

ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
ntidir[0].io->int_logic = NTI_DIS_INT_LOGIC;
```

Die Konstanten `NTI_DIS_INT_LOGIC` *(ausschalten) und* `NTI_EN_INT_LOGIC` *(einschalten) sind ebenfalls in der Datei* `nti.h` *definiert. Im Beispiel oben wird zuerst die Interrupt-Logik aktiviert und dann wieder ausgeschaltet.*

5.4.3 Logikanalysator Trigger

Das Register `LA-TRIGGER` an der Adresse `0x0E` des I/O-Space ermöglicht die Erzeugung eines Pulses am CPLD-Ausgang `TRIGGER`.[3]

	15:0	**Read/Write**
LA-TRIGGER	-	**0x0E**
	-	Reset

Abbildung 5.5: `LA-TRIGGER` - Register `0x0E` im I/O-Space

Der Puls kann durch Lese- und Schreibzugriffe mit beliebigem Datenwort erzeugt werden. Dieses Register ist vor allem für die Fehlersuche mit einem Logikanalysator gedacht, da durch einen einfachen I/O-Zugriff ein Trigger-Puls erzeugt werden kann.

Beispiel:

```
ntidir[0].io->la_trigger = 0xABCD;
tmp16 = ntidir[0].io->la_trigger;
```

Das Beispiel zeigt, wie mit einem Schreibzugriff mit dem Datenwort `0xABCD` *(oder jedem beliebigen anderem Datenwort) bzw. mit einem Lesezugriff auf das Register* `LA-TRIGGER` *zugegriffen und damit der Trigger-Puls ausgelöst wird.*

5.4.4 NTI `MISCELLANEOUS` Register

An der Adresse `0x12` im NTI I/O-Space liegt das `MISCELLANEOUS`-Register, dessen Aufgabe es ist, UTCSU-Steuersignale statisch zu setzen (bei Schreibzugriffen) bzw. das Setting zurückzulesen (bei Lesezugriffen). Bei der aktuellen Version des NTI-Moduls ist das `MISCELLANEOUS`-Register allerdings deaktiviert, da im CPLD zuwenig Platz dafür ist. Alle Signale, die man im Register setzen könnte sind im CPLD fix auf die Standardwerte gesetzt. Der UTCSU-Reset über den I/O-Space kann damit natürlich auch nicht ausgeführt werden.

	15	14:8	7	6	5	4	3	2	1	0	**Read/Write**
MISCELLANEOUS	Res	0000000	W	RH	RE	DI	TM	B16	B8	BE	**0x08**
	0	0b0000000	0	0	0	0	0	0	0	1	Reset

Abbildung 5.6: `MISCELLANEOUS` - Register `0x12` im I/O-Space

Mit diesem Register werden neun Steuerleitungen für die UTCSU getrieben, die in aufsteigender Reihenfolge erklärt werden.

Bit 0 `BIGENDIAN (BE):` [4]Mit diesem Signal wird festgelegt, ob die CPU die Daten im Little Endian Format (`BE='0'`) oder im Big Endian Format (`BE='1'`) anlegt.

Bit 1 `BUS8 (B8):` [4] Wenn `BUS8='1'`[5] ist, so nimmt die UTCSU an, daß von der CPU nur 8-Bit breite Datenzugriffe möglich sind.

Bit 2 `BUS16 (B16):` [4] Wie `BUS8`, nur mit 16-Bit Datenbreite.

Bit 3 `TESTMUX (TM):` Das Signal steuert den UTCSU-Eingang `TESTMUX`, der auf den Test-Multiplexer geführt ist. Wenn `TESTMUX='0'` ist, so wird der INTPBUS am CSUT-Bus der UTCSU ausgegeben (Zeitinformation). Stattdessen können auch chipinterne Testsignale nach außen geführt werden, wenn `TESTMUX` auf `HIGH` gesetzt wird.

[3]Das Signal kann am `Pin 2` des Jumpers `J5` abgegriffen werden (siehe Abschnitte 6.5 und 6.6).

[4]Die Signale `BIGENDIAN, BUS8, BUS16 und BE3...BE0` werden von der UTCSU für das Data Port Routing verwendet. Die genaue Funktionsweise wird unter anderem in [Mot90] ausführlich erläutert.

[5]Ist `BUS8` $\equiv$ `BUS16`, so wird damit ein 32-Bit breiter Datenbus signalisiert!

Bit 4 `DIRECTIN (DI)`: Wenn die CPU, der COMCO und die UTCSU mit demselben Takt betrieben werden ist keine Synchronisation der asynchronen Eingänge notwendig. Wenn `DIRECTIN` auf `HIGH`-Pegel gelegt wird, werden die Synchronisationsstufen abgeschaltet und die Zugriffszeiten werden kürzer.

Bit 5 `RELIABLE (RE)`: [6] In der UTCSU werden alle asynchronen Eingänge über Synchronisationsstufen geführt, d.h. die Signale werden erst mit der nächsten steigenden Taktflanke übernommen. Sollten trotzdem Metastabilitätsprobleme auftreten, so kann, wenn `RELIABLE` auf `HIGH` gesetzt wird, zu allen bereits synchronisierten Eingängen eine zweite Synchronisationsstufe zugeschaltet werden, d.h. die asynchronenen externen Signale werden chipintern erst bei der übernächsten steigenden Flanke gültig.

Bit 6 `RELH (RH)`: [6] Das Signal `RELH`, dessen vollständige Bezeichnung `Reliable Host` ist, wird benötigt, um eventuelle Metastabilitätsprobleme mit dem asynchronen $\overline{\texttt{CS}}$-Eingang zu vermeiden. Sollten Probleme auftreten, so wird eine Synchronisationsstufe zum $\overline{\texttt{CS}}$-Signal geschaltet, wenn `RH='1'` gesetzt wird.

Bit 7 `WAIT (W)`: [6] Um Microcontrollern mit langsamen Buszugriffszeiten die Ankopplung an die UTCSU zu ermöglichen, kann mit `WAIT='1'` bei jedem Schreib-/Lesezugriff ein Wartezyklus eingefügt werden, d.h. es wird einfach eine Synchronisationsstufe zwischengeschaltet.

Bit 8 - 14 werden nicht verwendet

Bit 15 `Reset (Res)`: Wenn `Reset='1'` ist, wird ein Hardware-Reset an der UTCSU ausgeführt. Dieses Signal treibt gemeinsam mit dem $\overline{\texttt{RESET}}$-Signal von der MA-Schnittstelle den UTCSU-Eingang `RESET`.

Nach einem hardwaremäßigen Reset des NTI-Moduls werden alle Signale, die im `MISCELLANEOUS`-Register gesetzt werden können, auf `LOW` getrieben.

In der Datei **`nti.h`** (siehe Abschnitt C.3) sind folgende Konstanten für das `MISCELLANEOUS`-Register definiert:

```
#define NTI_MISC_BIGENDIAN    1          /* UTCSU big-endian */
#define NTI_MISC_BUS8         (1 << 1)   /* UTCSU 8-bit Port-Size */
#define NTI_MISC_BUS16        (1 << 2)   /* UTCSU 16-bit Port-Size */
#define NTI_MISC_TESTMUX      (1 << 3)   /* Export Test-Signale am NTPA-Bus */
#define NTI_MISC_DIRECTIN     (1 << 4)   /* Ausschalten Synchronizer */
#define NTI_MISC_RELIABLE     (1 << 5)   /* 2-stufiger Synchronizer */
#define NTI_MISC_RELH         (1 << 6)   /* 2-stufiger Synchronizer CPU */
#define NTI_MISC_WAIT         (1 << 7)   /* 1 Waitstate-Operation */
#define NTI_MISC_RESET        (1 << 15)  /* UTCSU Reset */
```

Beispiel:

```
ntidir[0].io->utcsu_misc = NTI_MISC_RESET;
ntidir[0].io->utcsu_misc &= !NTI_MISC_RESET;

ntidir[0].io->utcsu_misc = NTI_MISC_BIGENDIAN | NTI_MISC_BUS16 | NTI_MISC_DIRECTIN;
ntidir[0].io->utcsu_misc &= !NTI_MISC_DIRECTIN;
tmp16 = ntidir[0].io->utcsu_misc;
```

Im Beispiel wird zuerst der UTCSU-Reset ausgeführt. Durch die UND-Verknüpfung mit der negierten Konstante `NTI_MISC_RESET` *wird das Reset-Bit wieder ausmaskiert und das Signal für den UTCSU-Reset wieder auf inaktiv gesetzt.*

Beim zweiten Beispiel verhält es sich genauso. Zuerst werden die Signale `BIGENDIAN`, `BUS16` und `DIRECTIN` gesetzt. Danach wird `DIRECTIN` deaktiviert und beim anschließenden Lesen des Registers muß der Wert `0x0005` (= `NTI_MISC_BIGENDIAN` *und* `NTI_MISC_BUS16`*) in der Variablen `tmp16` stehen.*

[6] `HIGH` an den Signalen `RELIABLE, RELH bzw. WAIT` führt zu einem langsameren Zugriffsverhalten der UTCSU!

5.4.5 Trigger Offsets

Bei den ersten Tests mit dem Evaluationssystem mußten wir feststellen, daß ϵ – die maximale Unsicherheit bei der Dauer der Datenpaketübertragung (siehe Abschnitt 2.2) – ca. 1.2 ms war, was ungefähr der Dauer eines maximal großen Ethernet-Pakets entspricht. Wie sich herausstellte, hing dieser Effekt mit dem TX-FIFO des i82596 und unserem fix programmierten Trigger-Offset an der Stelle `0x14` im Paket-Header zusammen (näheres siehe [SN99] bzw. [SKM+99]). Um dieser unangenehmen Probleme Herr zu werden, entschieden wir uns, ein Register `TRIGGER-OFFSET` in die Speicherstelle `0x16` des I/O-Space einzubauen, damit der optimale Triggerzeitpunkt ausgelotet werden konnte.

	15:8					7:0		**Read/Write**
TRIGGER-OFFSET	-	Transmit-Timestamp	-	-	-	Receive-Timestamp	-	**0x16**
	0	0b111000	0	0	0	0b01110	0	Reset

Abbildung 5.7: `TRIGGER-OFFSETS` - Register `0x16` im I/O-Space

Das Register erlaubt das Justieren der Trigger-Offsets für die 128 Byte großen Transmit-Header und die 64 Byte großen Receive-Header. Die Adresse des Transmit-Timestamp Triggers muß, um die oben genannten Effekte auszulöschen, größer als `88 (0x58)` sein. Der Adreßwert `92 (0x5C)` hat sich als durchaus brauchbar erwiesen. Das `TRIGGER-OFFSET`-Register ist so aufgebaut, daß die oberen 8 Bit als Transmit- und die unteren 8 Bit als Receive-Timestamp Trigger-Adresse schreib- und lesbar sind.

Beispiel:

```
ntidir[0].io->ts_offsets &= ((92 << 8) | 0x00FF);
```

Das Beispiel veranschaulicht, wie die Trigger-Adresse des Transmit-Timestamps vom Default-Wert `112 (0x70)` *auf den Wert* `92 (0x5C)` *gesetzt wird. Die Trigger-Adresse des Receive-Timestamps wird dabei nicht verändert und bleibt für den Fall, daß sie seit dem Reset nicht neu geschrieben wurde, auf den Wert* `28 (0x1C)` *gesetzt.*

5.4.6 Offset Typefield

Dieses Register ermöglicht eine Unterscheidung der Datenpakete und damit eine selektive Aktivierung der Mechanismen für Clock Synchronization Packets (vgl. Abschnitt 3.4).

		15:8			7:0		**Read/Write**
OFFSET-TYPEFIELD	E	-	-	-	Typefield-Offset	-	**0x1A**
	1	0b0000000	0	0	0b01110	0	Reset

Abbildung 5.8: `OFFSET TYPEFIELD` - Register `0x1A` im I/O-Space

Im Register `OFFSET-TYPEFIELD` an der Adresse `0x1A` des NTI I/O-Space kann die Type-Field Überprüfung im `Bit 15 (E)` ein- bzw. ausgeschaltet werden. Wenn dieses Bit auf `1` (Default-Wert) gesetzt wird, vergleicht das CPLD das Type-Field im Paket-Header mit dem Wert `0x5000` an der Adresse, die in den unteren 8 Bit programmiert werden kann. Zu beachten ist, daß die Adresse nur aus 6 Bit besteht, also kleiner als `64 (0x40)` sein muß.

Beispiel:

```
ntidir[0].io->type_offset = 0x103E;
```

Im Beispiel wird die Vergleichslogik aktiviert, da das Bit `15` *auf* `1` *gesetzt wird. Außerdem wird die Adresse für den Vergleich auf den maximalen Wert, nämlich* `62 (0x3E)` *gesetzt. Zu beachten ist, daß das Type-Field nur 16 Bit lang ist und deshalb auch Word-Adressen für den Vergleich relevant sind!*

5.4.7 Modul-Identifikation

Das NTI MA-Modul ist mit dem in [Man96] vorgesehenen Identifikationsmechanismus ausgestattet. Dieser wird mit einem seriellen EPROM (SPROM) mit 16 × 16-Words realisiert. Das SPROM muß ein zum Standard-IC 9306 (siehe [Nat96]) kompatibler Baustein sein. Die Datenkommunikation erfolgt über 4 Leitungen, die im Abschnitt 6.10.6 erklärt werden.

Die für das Auslesen notwendigen Schreib- und Lesezugriffe (vgl. Abbildung 6.10, Seite 71) sind im NTI I/O-Space an der Adresse `0xFE` realisiert.

	15:3	2	1	0	**Write**
SPROM WRITE	-	CS	CK	DI	**0xFE**
	–	0	0	0	Reset

Abbildung 5.9: `SPROM WRITE` - Register `0xFE` im I/O-Space

Das Register `SPROM WRITE` steuert die Signale `CS (Chip Select)`, `SK (Clock)` und `DI (Data In)` des SPROM's. Dabei sind die Datenbits `D2` dem `CS-Signal`, `D1` dem `SK-Signal` und `D0` dem `DI-Signal` zugeordnet. Die Signale werden vom CPLD getrieben und sind nach einem Reset auf `'0'` gesetzt.

	15:1	0	**Read**
SPROM READ	000000000000000	DO	**0xFE**
	0b000000000000000	-	Reset

Abbildung 5.10: `SPROM READ` - Register `0xFE` im I/O-Space

Greift man hingegen lesend auf die Adresse `0xFE` zu, so liest man das Register `SPROM READ` des NTI I/O-Spaces aus. In das Datenbit `D0` wird das Signal `DO (Data Out)` des SPROM's vom CPLD-geschrieben. Alle anderen Datenbits sind immer auf `'0'` gesetzt.

Die Bedeutungen der 16 Datenworte, die vom SPROM nach dem Schreiben der Adressen selektiv ausgelesen werden können, sind in der folgenden Tabelle beschrieben.

Word	**D15-D0**	**Bedeutung**
0	`0x5346`	Synchronisations-Code
1	`0x????`	Modulnummer (binär)

2	`0x0100`	Revisionsnummer
		`D15...D08: ganzzahliger Teil (binär)`
		`D07...D00: Dezimalteil (binär)`

3	`0x1677`
	`D12='1'`
	`D11='1'`
	`D09='1'`
	`D06...D05='11'`
	`D04...D03='10'`
	`D02...D01='11'`
	`D00='1'`
		Moduleigenschaften [a]
		`Modul braucht +12V`
		`Modul braucht +5V`
		`Modul mit Trigger-Eingängen`
		`Interrupter Methode C`[b]
		`32-Bit Datenbus (D32)`
		`24-Bit Adreßbus (A24)`
		`Modul unterstützt Memory-Accesses`

4-7	`0x0000`	laut MA-Standard reserviert
8	`0xXXXX`	NTI Produktionsnummer (binär), fortlaufend

9	`0x0100`	UTCSU-Version (binär), '1.0'
		`D15...D08: ganzzahliger Teil (binär)`
		`D07...D00: Dezimalteil (binär)`

10-15	`0x0000`	reserviert für weitere Informationen

[a] Hier sind nur die Datenbits erklärt, die auf `'1'` gesetzt werden müssen, um die Moduleigenschaften des NTI's zu charakterisieren. Der interessierte Leser kann die vollständige Beschreibung des Registers in [Man96, Seite 3] nachlesen.

[b] Wir verwenden die Interrupter-Methode mit Vektorgenerierung (siehe Abschnitt 3.6).

Tabelle 5.4: NTI-Identifikation

Der Registerzugriff erfolgt analog zu allen anderen Registern, und zwar mit `ntidir[0].io->s_prom`.

5.5 Memory-Space Zugriffe

Wie im Abschnitt 3.5 beschrieben wurde, besteht der Memory-Space des NTI-Moduls aus einem 512 KByte[7] großen Speicherbereich in dessen obersten 512 Byte die UTCSU eingebettet ist. Auf den NTI-Speicher kann über zwei Adreßbereiche zugegriffen werden, und zwar über den COMCO-Bereich, der nur dem Netzwerk Coprozessor und damit dem NTI-Treiber vorbehalten sein sollte, und über den CPU-Bereich, in dem die Pakete zum Senden vorbereitet oder nach dem Empfangen ausgelesen werden. Die Speicherbelegung des NTI's sowie die Start- und Endadressen der einzelnen Bereiche sind in Tabelle 5.1, Abschnitt 5.1 veranschaulicht.

Der Datei `nti.h` (Abschnitt C.3) sind folgende Definitionen entnommen, mit denen die oben beschriebene Speicheraufteilung in C strukturiert ist.

```
#define NTI_MEMORY_CPU   XMT_HEADER_REGION + RCV_HEADER_REGION + BUFFER_REGION +
                         SYS_STRUCT_REGION + UTCSU_GAP

typedef volatile union
{
  INT8  rw8[NTI_MEMORY_CPU];
  INT16 rw16[(NTI_MEMORY_CPU)/2];
  INT32 rw32[(NTI_MEMORY_CPU)/4];
} NTI_CPU2MEM;
```

Mit der Definition `NTI_MEMORY_CPU` wird die Größe des verfügbaren NTI-Speichers berechnet. Die Größe der Teilbereiche `XMT_HEADER_REGION` ... `UTCSU_GAP` wird weiter oben im Code definiert, und zwar in Abhängigkeit von `NTI_VERSION`, das in der Datei `nti_version.h` (Abschnitt C.4) festgelegt ist und bestimmt, ob das NTI-Modul über 256 KByte oder 512 KByte physikalischen Speicher verfügt. Im ersten Fall können die gesamten 256 KByte des NTI-Speichers als Schnittstelle zwischen CPU und COMCO verwendet werden. Dabei sorgt `UTCSU_GAP` dafür, daß das Speicherloch zwischen dem Ende des physikalischen Speichers und dem Beginn der UTCSU überbrückt wird. Im zweiten Fall ist `UTCSU_GAP` = 0, da die UTCSU unmittelbar nach dem nutzbaren Speicher (`= 512 kByte - 512 Byte`) angeordnet ist. Die Union `NTI_CPU2MEM` ermöglicht, daß mit den Feldern `rw8`, `rw16` und `rw32` der verfügbare Speicher mit 8-, 16- und 32-Bit Datenbreite zugreifbar ist.

```
typedef volatile struct
{
  NTI_COMCO2MEM comco; /* double-mapped RAM f"ur COMCO-Accesses */
  NTI_CPU2MEM   cpu;   /* double-mapped RAM f"ur CPU-Accesses */
} NTI_MEMORY;
```

Die Struktur `NTI_MEMORY` wird der Tatsache gerecht, daß zwei Speicherbereiche auf einen physikalischen Speicher verweisen. Die untere Hälfte des Speichers wird durch die Struktur `NTI_COMCO2MEM` beschrieben, die hier allerdings nicht näher erläutert wird (siehe dafür `nti.h` in Abschnitt C.3), und als `comco` verwendet wird. Das Schreiben und Lesen in diesem Speicherbereich ist mit Vorsicht zu betreiben, da an gewissen Adressen Trigger-Signale ausgelöst werden. Abgesehen von Testprogrammen ist der Zugriff auf diesen Bereich nicht notwendig!

Mit der Komponente `cpu` vom oben erklärten Typ `NTI_CPU2MEM` wird normalerweise auf den Speicher zugegriffen. Weiter unten illustrieren einige Beispiele diesen Vorgang.

```
typedef volatile struct
{
  NTI_MEMORY mem;     /*  double mapped RAM */
  UTCSU      utcsu;
} NTI;
```

Die Strukur `NTI` vereint schließlich die Strukturen des Speichers mit der UTCSU. Der Typ `UTCSU` ist in der Datei `utcsu.h` (Abschnitt C.10) definiert und wird im nächsten Kapitel ausführlich erläutert. Ebenso wie der I/O-Space (Abschnitt 5.4) ist die Struktur `NTI` im NTI-Handler eingebettet (siehe 4.5).

[7] Vorsicht: Die meisten NTI-Module des Evaluationssystems sind nur mit 256 KByte bestückt!

Die folgenden Beispiele zeigen verschiedenste Möglichkeiten für Speicherzugriffe, die alle Big-Endian Byte-Ordnung voraussetzen.

Zugriffe mit 32 Bit Datenbreite

Beispiel 1:

```
ntidir[0].memory->mem.cpu.rw32[0x100] = 0x12345678;
tmp32 = ntidir[0].memory->mem.cpu.rw32[0];
```

Im ersten Beispiel wird das Datenwort `0x12345678` *über die bereits bekannten Strukturen als 256. Longword(!) (=*`0x100`*) in den Speicher geschrieben. Anschließend wird die erste Speicherstelle des CPU-Bereichs als 32 Bit Datenwort ausgelesen und in* `tmp32` *geschrieben.*

Beispiel 2:

```
(INT32 *)0xE0280400 = 0x44556677;
ntidir[0].memory->mem.comco.rw32.txhead[8].addr_data[0] = 0x12345678;
tmp32 = (INT32 *)0xE0200400;
```

Das zweite Beispiel beinhaltet ein paar kompliziertere Speicherzugriffe. In der ersten Zeile wird das Datenwort an die gleiche Stelle wie in Beispiel 1 geschrieben. Allerdings geschieht es hier in „Hacker-Manier" und wird deshalb etwas genauer erklärt. Die Speicheradresse `0xE0280400`[8] *setzt voraus, daß die VME-Basisadresse des Carrier-Boards auf* `0xE0xxxxxx` *programmiert wurde. Weiters wird das Modul im Slot 2 über den CPU-Bereich des NTI-Speichers adressiert. Über einen Zeiger wird auf die Speicherstelle des* `0x400ten` *Bytes ein* `INT32` *Schreibzugriff gemacht. Das entspricht aber genau dem* `0x100ten` *Longword des CPU-Bereichs, also ebenso wie im ersten Beispiel.*

In der zweiten Zeile findet ein Schreibzugriff im COMCO-Bereich statt, der über die NTI-Strukturen ebenfalls auf die gleiche Speicherstelle den Wert `0x12345678` *schreibt. Das mag vielleicht nicht auf den ersten Blick ersichtlich sein, aber bei näherer Betrachtung erkennt man, daß in das erste Longword des 8. Transmit-Headers geschrieben wird. Da die Transmit-Header bekannterweise* `128 Byte` *lang sind, wird wieder an die Stelle* `0x400` *geschrieben. Der Zugriff passiert hier zwar vom COMCO-Bereich aus, aber für den physikalischen Speicher macht das keinen Unterschied. Im Longword* `0x100` *des Speichers steht somit nach Ausführung der zweiten Zeile nicht mehr* `0x44556677` *sondern* `0x12345678`*.*

Die abschließende Leseoperation erfolgt mit einem Zeiger auf `INT32` *direkt über die Adresse. Und – man staune – es wird wieder die gleiche Adresse, nämlich 0x400 des physikalischen Speichers, ausgelesen, aber diesmal vom COMCO-Bereich. In diesem Fall ist die Leseoperation harmlos, da an der Adresse keine Timestamp-Signale ausgelöst werden. Es wird aber nochmals eindringlich vor bösen Seiteneffekten gewarnt, wenn direkte Zugriffe nicht im CPU-Bereich des Speichers durchgeführt werden!*

Zugriffe mit 8 Bit und 16 Bit Datenbreite

Beispiel:

```
ntidir[0].memory->mem.cpu.rw16[0x200] = 0x1234;
ntidir[0].memory->mem.cpu.rw16[0x202] = 0x5678;

tmp32 = ntidir[0].memory->mem.cpu.rw8[0x400] << 24;
tmp32 &= (ntidir[0].memory->mem.cpu.rw8[0x401] << 16);
tmp32 &= (ntidir[0].memory->mem.cpu.rw8[0x402] << 8);
tmp32 &= ntidir[0].memory->mem.cpu.rw8[0x403];
```

Dieses Beispiel soll veranschaulichen, wie 32 Bit Datenwörter korrekt mit 8 bzw. 16 Bit Zugriffen geschrieben bzw. gelesen werden. Die Zugriffe oben finden ausschließlich über die bereits bekannten Strukturen statt. Bei Schreiboperationen müssen die höchstwertigen Datenbytes an die niederwertigste Adresse geschrieben werden, z.B. das 16 Bit Datenwort `0x1234` *wird an die niedrigere,* `0x5678` *an die höhere Speicherstelle geschrieben. Würde man nun einen Longword-Lesezugriff auf die untere Adresse machen, dann würde man das korrekte Datenwort* `0x12345678` *lesen. Die 8 Bit Lesezugriffe funktionieren analog dazu. Dem aufmerksamen Leser wird hoffentlich nicht entgangen sein, daß alle Zugriffe dieses Beispiels auf die gleiche Adresse wie in den vorigen Beispielen zugreifen.*

[8] Bei diesem Beispiel wird vorausgesetzt, daß ein AcQ i6360 Carrier-Board mit Extension-Card verwendet wird. Nähere Informationen dazu in Abschnitt 4.4.2.

5.6 Programmiermodell der UTCSU

Dieser Abschnitt beschreibt, wie die UTCSU in die C-Strukturen des NTI-Handlers eingebunden ist. Einige Beispiele ergänzen die Ausführungen und erhöhen damit hoffentlich das Verständnis des Lesers. Die Funktionsweise der UTCSU-Register inklusive der Bedeutung aller Bits sowie die Programmierung der UTCSU-Funktionsblöcke (z.B. Starten der Uhr, Interruptbearbeitung, etc.) werden hier nicht behandelt sondern müssen im Kapitel 8 von [Loy96] studiert werden, wo jede UTCSU-Unit ausführlichst beschrieben ist.

Die folgende (bereits bekannte) Struktur ist aus der Datei `nti.h` (Abschnitt C.3) entnommen. Damit können UTCSU-Zugriffe prinzipiell genauso wie Memory-Space Zugriffe (siehe voriges Kapitel) programmiert werden.

```
typedef volatile struct
{
  NTI_MEMORY mem;     /*  double mapped RAM */
  UTCSU      utcsu;
} NTI;
```

Das Strukturelement `utcsu` ist vom Typ `UTCSU`, der wiederum in der Datei `utcsu.h` (Abschnitt C.10) definiert ist. Die Gliederung erfolgt nach Schreib- und Lesezugriffen sowie nach Datenbreiten.

```
typedef union
{
  UTCSU_WRITE32 w32;
  UTCSU_READ32  r32;

  UTCSU_WRITE16 w16;
  UTCSU_READ16  r16;

  UTCSU_WRITE8  w8;
  UTCSU_READ8   r8;
} UTCSU;
```

Wie leicht zu sehen ist, besteht der Typ `UTCSU` aus sechs Unterstrukturen, die alle an der gleichen Speicheradresse stehen (da Union-Typ) und die oben erwähnte Einteilung nach Zugriffen und Datenbreiten bewirken. Bei `w32` und `r32` sind alle Register (mit Ausnahme von SSU- und GPU-Registern) als `INT32` ausgeführt. Die SSU- und GPU-Register sind als Felder von Unterstrukturen ausgeführt, da sie jeweils mehrfach vorkommen und gleich aufgebaut sind.

Bei den 8 Bit bzw. 16 Bit Strukturen `r8` ... `w16` sind die 32 Bit breiten UTCSU-Register als Felder vom Typ `INT8 [4]` bzw. `INT16 [2]` ausgeführt. In der Beschreibung klingt das alles etwas kompliziert, aber ein Blick auf die vollständigen Strukturen bzw. auf die unten angeführten Beispiele sollte alle Unklarheiten beseitigen.

Beispiel 1:

```
ntidir[0].utcsu->w32.tsset = 0x12345678;
tmp32 = ntidir[0].utcsu->r32.utcstat2;
oder
ntidir[0].memory->utcsu.w32.tsset = 0x12345678;
tmp32 = ntidir[0].memory->utcsu.r32.utcstat2;
```

Im ersten Beispiel werden 32 Bit Zugriffe mittels der bereits bekannten Handler-Strukturen (siehe 4.5) gezeigt. Es gibt zwei Möglichkeiten um auf die UTCSU über die Variable `ntidir[0]` *zuzugreifen, und zwar über die Strukturkomponente* `utcsu` *(oben) oder mit einem kleinen Umweg über das Strukturelement* `memory` *(unten). Im Beispiel wird für beide Varianten je ein Schreib- und ein Lesezugriff gezeigt, wobei der Wert* `0x12345678` *in das UTCSU-Register* `TSSET` *geschrieben und das Register* `UTCSTAT2` *ausgelesen wird.*

Beispiel 2:

```
ntidir[0].utcsu->w32.ssu[2].dutyal = 1000;
tmp32 = ntidir[0].utcsu->r32.gpu[0].tsgps;
```

Im zweiten Beispiel wird gezeigt, wie auf die Register der Synchronisation Subnet Units (SSU) bzw. der GPS Units (GPU) zugegriffen wird. Der Schreibzugriff in der ersten Zeile bewirkt, daß der Wert `1000` *in das Register* `DUTYAL` *der dritten*[9] *SSU geschrieben wird. Beim nächsten Zugriff wird das* `TSGPS`*-Register der GPU 1 gelesen.*

Beispiel 3:

```
ntidir[0].utcsu->w16.tsset[0] = 0x1234;
ntidir[0].utcsu->w16.tsset[1] = 0x5678;

tmp32 = (ntidir[0].utcsu->r16.utcstat2[0] << 16) | ntidir[0].utcsu->r16.utcstat2[1];
```

Beipiel 3 soll zeigen, wie die 32 Bit breiten Register der UTCSU mit 16 Bit Zugriffen korrekt beschrieben bzw. gelesen werden.

Beispiel 4:

```
ntidir[0].utcsu->w8.tsset[0] = 0x12;
ntidir[0].utcsu->w8.tsset[1] = 0x34;
ntidir[0].utcsu->w8.tsset[2] = 0x56;
ntidir[0].utcsu->w8.tsset[3] = 0x78;

tmp32 = ntidir[0].utcsu->r8.utcstat2[0] << 24;
tmp32 &= (ntidir[0].utcsu->r8.utcstat2[1] << 16);
tmp32 &= (ntidir[0].utcsu->r8.utcstat2[2] << 8);
tmp32 &= ntidir[0].utcsu->r8.utcstat2[3];
```

Das vierte Beispiel ist dem dritten sehr ähnlich, nur die Breite der Datenzugriffe ist 8 Bit statt 16 Bit.

Beispiel 5:

```
(INT32 *)0xE02FFFE04 = 0x12345678;
```

Das fünfte Beispiel zeigt wieder direkten Zugriff auf die Adresse `0xE02FFFE04` *mit einem* `INT32`*-Zeiger. Unter denselben Voraussetzungen wie bei den anderen Beispielen mit direkter Adressierung wird hier ebenfalls ein Schreibzugriff auf das Register* `TSSET` *durchgeführt.*

[9] In C beginnt die Indizierung der Arrays bekanntlicherweise mit 0.

Kapitel 6

Technische Details

6.1 Installation

Ein MA-Modul wird so auf ein Carrier-Board aufgesteckt, daß die Bauteilseiten zueinander gerichtet sind. Die Kommunikation sowie die Spannungsversorgung erfolgen über den 60-poligen MA-Stecker (siehe Abschnitt 6.4.1). Jedes Modul wird mit zwei M3∗6 und zwei M3∗8 Flachkopfschrauben am Carrier-Board befestigt.

Je nach Carrier-Board können 2 (AcQ i6360 ohne i6Ext), 4 (MEN A203) bzw. 6 (AcQ i6360 mit i6Ext) MA-Module aufgesteckt werden.

Periphere Signale können über den 25-poligen DSub-Stecker FPC (siehe Tabelle 6.2), den 20-poligen Reihenstecker CPC (siehe Tabelle 6.3) bzw. die Stecker IPC 1 und 2 (siehe Tabellen 6.4 und 6.5) angeschlossen werden.

6.2 Erste Schritte

Um prinzipiell die Funktion des NTI-Moduls zu testen, kann folgendermaßen vorgegangen werden:

- System mit dem Carrier-Board ohne NTI installieren.
- Carrier-Board testen, und zwar I/O-Space und Memory-Space Zugriffe auf den Slot 0. [1]
- Nach erfolgreichem Test das NTI in den Slot 0 des Carrier-Boards stecken.
- Einen geeigneten Debugger laden.
- Zugriffe auf die Basisadresse, z.B. Lesen des MA Memory-Space (Longword).
- Wenn ein Bus-Error auftritt, ist das Modul nicht richtig eingesteckt.
- Schreiben eines Words (16 Bit) auf den I/O-Space (z.B. Adresse `0x10`).
- Schreiben des Longwords `0x00000001` auf die Adresse `0xXX0FFE40`[2].
- Lesen eines Longwords von der Adresse `0xXX0FFE60` liefert die UTCSU-Identifikation

6.3 Spannungsversorgung

Die Spannungsversorgung des Moduls erfolgt durch das Carrier-Board. Die notwendigen Spannungen `+5V` und `+12V` werden über den 60-poligen MA-Stecker (siehe Tabelle 6.1) an das NTI geführt.[3]

[1] Beipielsweise könnte der Test mit M-Modul Zugriffen auf einen Slot gemacht werden, auf dem $\overline{\texttt{CS}}$ und $\overline{\texttt{DTACK}}$ mit einem Draht kurzgeschlossen sind.

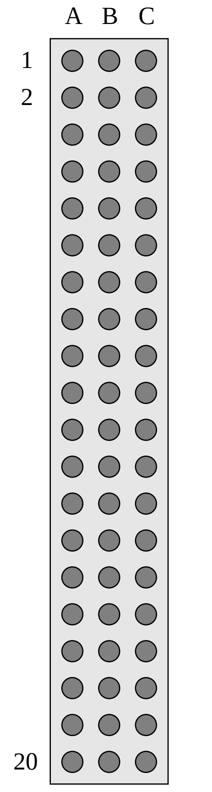
[2] `XX` hängt vom M-Modul Bereich des Carrier-Boards ab. Nährer Informationen gibt es in den Abschnitten 4.4 und 5.5.

[3] Da das Carrier-Board laut [Man96] pro MA-Slot nur `1A` bei der `+5V` Versorgung liefern muß, gibt es die Möglichkeit das NTI über den IPC1-Stecker (siehe Tabelle 6.4) direkt vom Rack aus zu versorgen.

6.4 Steckerbelegungen

6.4.1 MA-Stecker

Die Network Time Interface Funktionen können von einer CPU bzw. einem COMCO über die MA-Schnittstelle verwendet werden. Das MA-Modul ist mit einem 60-poligen Stecker (female) bestückt, der drei Reihen zu je 20 Pins hat und auf das Carrier-Board gesteckt wird. Die Signalbelegung des MA-Steckers kann aus der folgenden Tabelle entnommen werden.



Pin	A	B	C
1	$\overline{\mathtt{CS}}$	GND	$\overline{\mathtt{AS}}$
2	A01	+5V	D16
3	A02	+12V	D17
4	A03	-12V	D18
5	A04	GND	D19
6	A05	$\overline{\mathtt{DREQ}}$	D20
7	A06	$\overline{\mathtt{DACK}}$	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	$\overline{\mathtt{DS1}}$	$\overline{\mathtt{DS0}}$	D29
18	$\overline{\mathtt{DTACK}}$	$\overline{\mathtt{WRITE}}$	D30
19	$\overline{\mathtt{IACK}}$	$\overline{\mathtt{IRQ}}$	D31
20	$\overline{\mathtt{RESET}}$	SYSCLK	$\overline{\mathtt{DS2}}$

Tabelle 6.1: Steckerbelegungen des 60-poligen MA-Steckers

Die Bedeutung der Signale und die prinzipielle Funktion der MA-Schnittstelle werden im Abschnitt 3.2 erklärt.

6.4.2 Front-Panel Stecker und Carrier-Board Plug Connector

Die MA-Spezifikation sieht vor, daß externe Signale zu oder von MA-Modulen entweder über einen 25-poligen D-Sub Stecker (female) an der Vorderseite des Moduls oder einen 24-poligen Leistenstecker geführt werden. Der erste Stecker wird Front-Panel Connector (FPC) genannt, der zweite wird üblicherweise als Carrier-Board Plug Connector (CPC) bezeichnet, da er auf die Trägerkarte aufgesteckt werden kann (optional). Die Signale des CPC werden am Carrier-Board zu freien Pins des VME-P2 Steckers geleitet, sofern es mit den Steckern ausgerüstet ist.[4]

Beim NTI werden beide Stecker verwendet, um möglichst alle UTCSU-Eingänge und Ausgänge herausführen zu können.

Beim Entwurf der Steckerbelegung wurde ein besonderes Augenmerk auf eine symmetrische Anordnung der Signale gelegt, um Zerstörungen des NTI-Moduls durch falsches Anstecken externer Signale vorzubeugen. Weiters werden am Modul alle Eingänge über Optokoppler bzw. über Buffer in DIL-Gehäusen geführt, damit zu hohe Signalpegel nur einen leicht zu ersetzenden Baustein und nicht das ganze NTI zerstören.

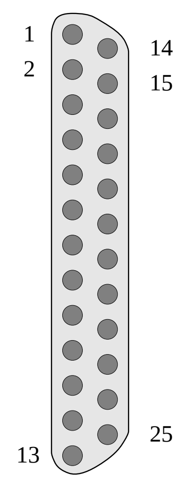
[4] Sowohl bei den AcQ i6360/i6040 als auch bei den (für uns neu entwickelten) i6Ext-Extender Cards von AcQ ist das nicht der Fall!

Am Front-Panel Connector, dessen Steckerbelegung in Tabelle 6.2 abgebildet ist, werden alle Signale (`1pps GPS1`, `Status GPS1` und `10MHz GPS1`) vom GPS-Empfänger 1 über extrem schnelle Optokoppler (siehe Abschnitt 6.10.4) geführt. Die Opto's werden mit den externen Versorgungsleitungen `+5V GPS1` und `GND GPS1` betrieben. [5]

Der Application-Eingang `App8` wird ebenfalls über einen Optokoppler an die UTCSU geführt, die Spannungsversorgung erfolgt hier über die Eingänge `+5V App8` und `GND App8`.

Alle anderen Signale der GPS-Empfänger 1 und 2 sowie die Application-Eingänge $\overline{\texttt{App0}}$,..., $\overline{\texttt{App7}}$ werden über sehr schnelle, gesockelte DIP Buffer (74ACT04) zu den entsprechenden UTCSU-Eingängen geführt.[6] Alle Signale sind mit 10kΩ Pull-Up Widerständen versehen, um Seiteneffekte von Floating Inputs zu vermeiden.

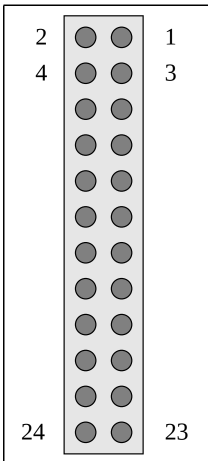
Der UTCSU-Ausgang $\overline{\texttt{APPDUTY}}$ wird von einem 74ACT240 gepuffert. Über den Pin `+5V` wird eine Spannungsversorgung zur Verfügung gestellt, die mit einer 125-mA Sicherung gegen Überlast geschützt ist.



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

Tabelle 6.2: Steckerbelegungen des 25-poligen Front-Panel Steckers (FPC)

Die Steckerbelegung des 24-poligen Carrier-Board Plug Connectors ist in Tabelle 6.3 dargestellt. Alle Signale sind gepuffert und mit 10kΩ Pull-Up's versehen. Mit Ausnahme der Signale $\overline{\texttt{TSAPP0}}$, ..., $\overline{\texttt{TSAPP8}}$, die den Signalen $\overline{\texttt{App0}}$,..., $\overline{\texttt{App8}}$ am FPC entsprechen, werden alle Signale über 74ACT240 Buffer geführt.



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

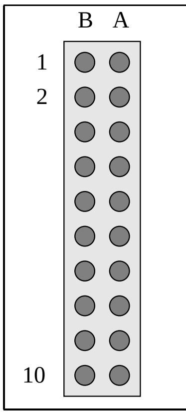
Tabelle 6.3: Steckerbelegungen des 24-poligen Carrier-Board Steckers (CPC)

[5] Achtung: Die Versorgung darf nicht vom Rack aus erfolgen, da eine wirkliche Potentialtrennung erforderlich ist, damit die (sehr empfindlichen) Optokoppler funktionieren!

[6] Die 74ACT04-Bausteine sind Inverter, folgedessen sind hier, im Gegensatz zu den Signalen über die Optokoppler, alle Eingänge invertierend (siehe Schaltplan, Abschnitt 6.6)!

6.4.3 Intermodule-Port Stecker 1 und 2

Die UTCSU hat deutlich mehr Ausgänge als über den Front-Panel und den Carrier-Board Plug Connector nach außen geführt werden können. Diese werden mittels der Intermodule-Port Stecker 1 (IPC1, Tabelle 6.4) und 2 (IPC2, Tabelle 6.5) zugänglich gemacht.



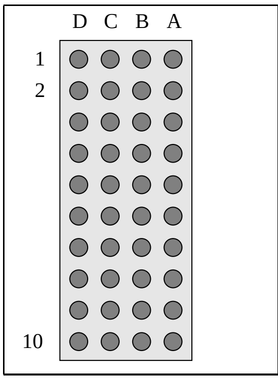
	B	A
1	CSUT8	CSUT9
2	CSUT6	CSUT7
3	CSUT4	CSUT5
4	$\overline{\texttt{1pps GPS2}}$	GND
5	$\overline{\texttt{Status GPS2}}$	+5V
6	10MHz GPS2	GND
7	$\overline{\texttt{1pps GPS3}}$	+5V
8	$\overline{\texttt{Status GPS2}}$	GND
9	CSUT2	CSUT3
10	CSUT0	CSUT1

Tabelle 6.4: Steckerbelegungen des 20-poligen Intermodule Plug Connectors 1 (IPC1)

Am IPC1 können sämtliche Signale der GPS-Empfänger 2 und 3 angehängt werden, um den Aufbau von Knoten mit modularen GPS-Empfängern zu erleichtern.

Die `+5V` und `GND` Pins sind vorgesehen, um eine externe Stromversorgung anzuschließen, falls das NTI-Modul mehr als den maximal zulässigen Strom von 1A verbrauchen sollte.

Die restlichen Pins der IPC's führen den CSU-Bus der UTCSU ungepuffert nach außen. Die Signale `CSUT0`, …, `CSUT47` liefern entweder Zeitinformation (den Wert des internen NTP-Busses) oder interne Steuersignale, jenachdem wie der Eingang `TESTMUX` gesetzt ist (Details siehe [Loy96], Kapitel 8.9).

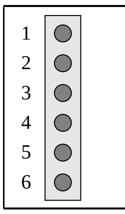


	D	C	B	A
1	CSUT13	CSUT12	CSUT11	CSUT10
2	CSUT17	CSUT16	CSUT15	CSUT14
3	CSUT21	CSUT20	CSUT19	CSUT18
4	CSUT25	CSUT24	CSUT23	CSUT22
5	CSUT29	CSUT28	CSUT27	CSUT26
6	CSUPHASE	CSUT32	CSUT31	CSUT30
7	CSUT36	CSUT35	CSUT34	CSUT33
8	CSUT40	CSUT39	CSUT38	CSUT37
9	CSUT44	CSUT43	CSUT42	CSUT41
10	N.C.	CSUT47	CSUT46	CSUT45

Tabelle 6.5: Steckerbelegungen des 40-poligen Intermodule Plug Connectors 2 (IPC2)

6.4.4 JTAG-Interface Stecker

Die Programmierung des Complex Programmable Logic Device (CPLD) wird über das JTAG-Interface durchgeführt. Das Layout dieses einreihigen, 6-pinnigen Steckers ist in Tabelle 6.6 abgebildet.



1	+5V
2	TMS
3	TDO
4	TDI
5	CLK
6	GND

Tabelle 6.6: Steckerbelegungen des 6-poligen JTAG-Interfaces

6.5 Jumper

In der folgenden Tabelle 6.7 wird die Funktion der Jumper `J1 - J9` beschrieben. Die Einstellungen für den Standardbetrieb des Moduls sind in fetter Schrift dargestellt (z.B. **1 - 2**).

Jumper	Pos.	Funktion
J1	1 - 2	Das Signal $\overline{\mathtt{TSXMT1}}$ vom CPC wird über einen Buffer an den UTCSU-Eingang `TSXMT1` geführt.
	2 - 3	Das Signal $\overline{\mathtt{TTSXMT}}$ vom CPLD wird über einen Buffer an den UTCSU-Eingang `TSXMT1` geführt.[a]
	-	`NICHT ERLAUBT!`
J2	1 - 2	Das Signal $\overline{\mathtt{TSRCV1}}$ vom CPC wird über einen Buffer an den UTCSU-Eingang `TSRCV1` geführt.
	2 - 3	Das Signal $\overline{\mathtt{TTSRCV}}$ vom CPLD wird über einen Buffer an den UTCSU-Eingang `TSRCV1` geführt.[b]
	-	`NICHT ERLAUBT!`
J3	**1 - 2**	Das Signal $\overline{\mathtt{TSXMT2}}$ vom CPC wird über einen Buffer an den UTCSU-Eingang `TSXMT2` geführt.
	2 - 3	Das Trigger-Signal `TRIGA` vom 60-poligen MA-Stecker wird über einen Buffer an den UTCSU-Eingang `TSXMT2` geführt.
	-	`NICHT ERLAUBT!`
J4	**1 - 2**	Das Signal $\overline{\mathtt{TSRCV2}}$ vom CPC wird über einen Buffer an den UTCSU-Eingang `TSRCV2` geführt.
	2 - 3	Das Trigger-Signal `TRIGB` vom 60-poligen MA-Stecker wird über einen Buffer an den UTCSU-Eingang `TSRCV2` geführt.
	-	`NICHT ERLAUBT!`
J5	**-**	Das Signal $\overline{\mathtt{HWSNAP}}$ für den UTCSU-Eingang `HWSNAP` wird vom CPC über einen Buffer (invertiert) erzeugt.
	1 - 2	Das `LA-TRIGGER`-Signal des CPLD's wird über eine invertierte Transistorschaltung (OC) geführt und steuert den Eingang `HWSNAP` der UTCSU.
J6 [c]	**1 - 2**	Die `+12V`-Versorgung wird an `Pin 14` des Quarzsockels geführt.
	2 - 3	`Pin 14` des Quarzsockels wird mit `+5V` versorgt.
	-	`Pin 14` des Quarzsockels ist offen.
J7 [c]	**1 - 2**	Die Pins 1 und 4 des Quarzsockels werden an `GND` gelegt.
	-	Die Pins 1 und 4 des Quarzsockels sind offen
J8	**1 - 2**	Der Pin $\overline{\mathtt{APPDUTY}}$ am FPC und am CPC wird vom UTCSU-Ausgang `APPDUTY` über einen invertierenden Transistorschalter (OC) geführt.
	2 - 3	Das CPLD-Signal $\overline{\mathtt{TTSXMT}}$ wird über einen invertierenden Transistorschalter (OC) an den Pin $\overline{\mathtt{APPDUTY}}$ am FPC und CPC geführt.
J9 [c]	1 - 2	Die `+12V`-Versorgung wird an `Pin 11` des Quarzsockels geführt.
	2 - 3	`Pin 11` des Quarzsockels wird mit `+5V` versorgt.
	-	`Pin 11` des Quarzsockels ist offen.

[a] Nur in dieser Position werden die Transmit-Timestamps beim Senden von Datenpaketen richtig abgezogen und in das Datenpaket eingefügt!

[b] Nur in dieser Position werden die Receive-Timestamps beim Empfangen von Datenpaketen richtig abgezogen!

[c] Die Quarzbeschaltung des NTI-Moduls wird im Abschnitt 6.10.3 erklärt.

Tabelle 6.7: Bedeutung der Jumper am NTI

6.6 Schaltplan

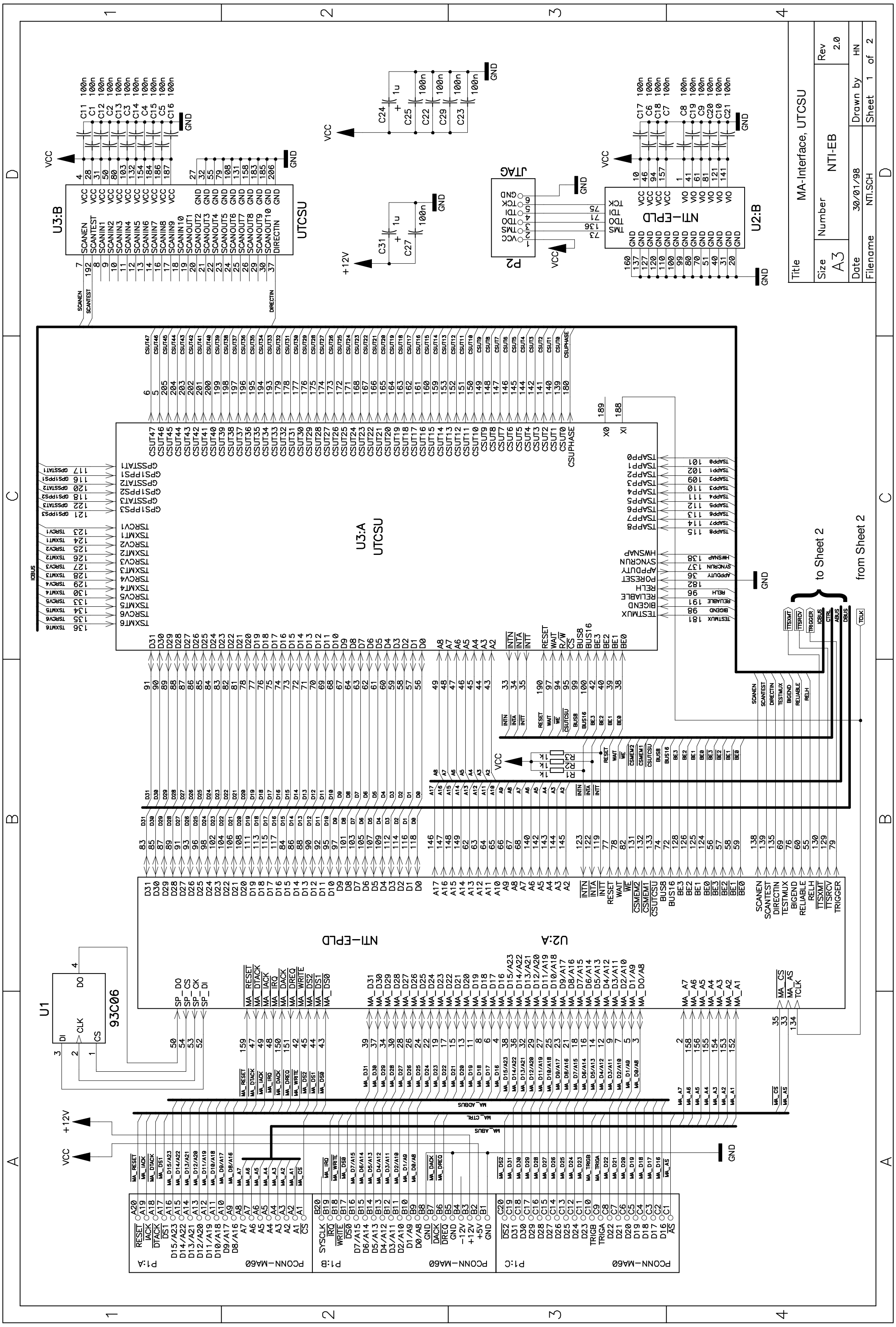


Abbildung 6.1: NTI-Schaltplan Seite 1 (MA-Interface und UTCSU)

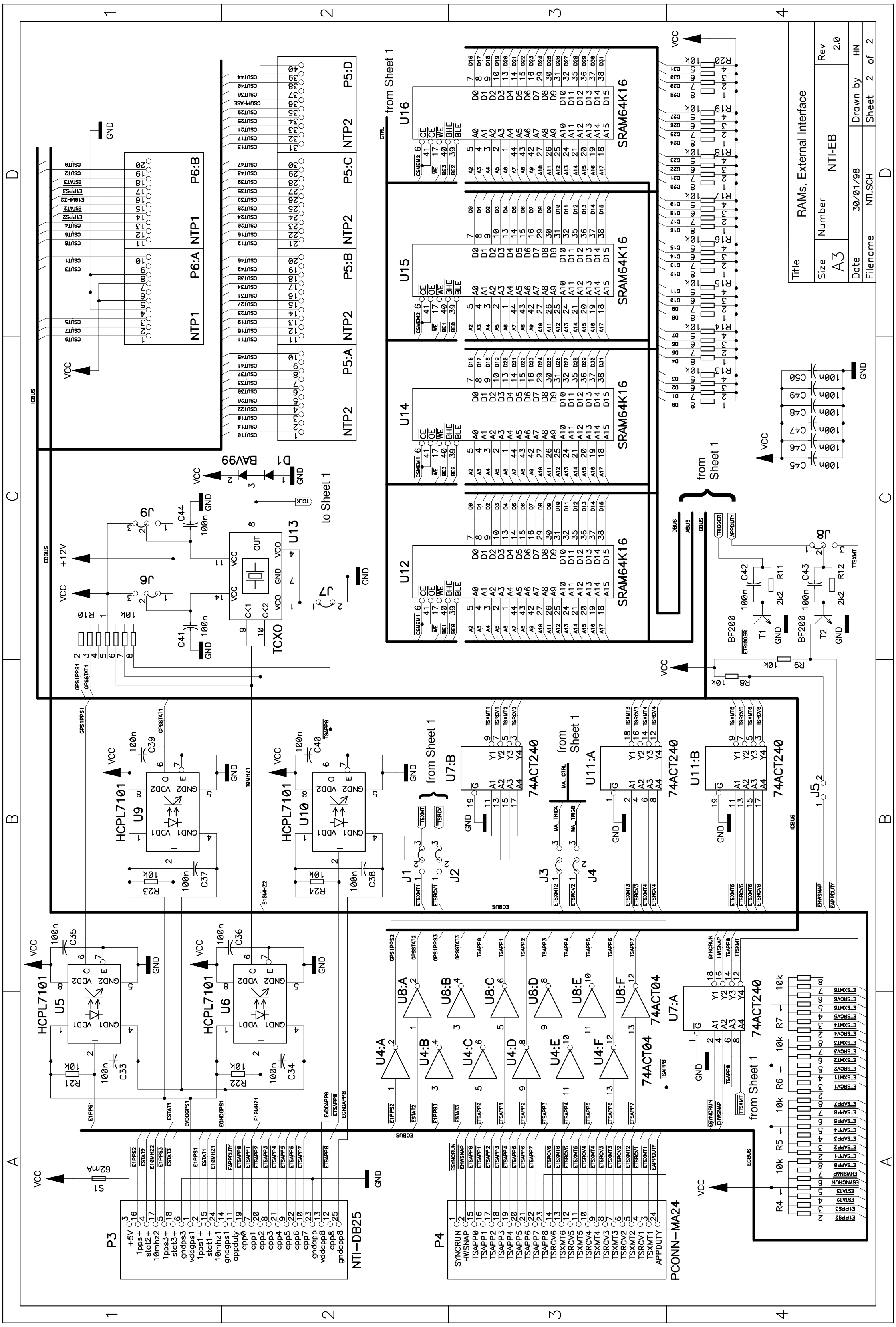


Abbildung 6.2: NTI-Schaltplan Seite 2 (Speicher und externe Interfaces)

6.7 Bestückungsplan

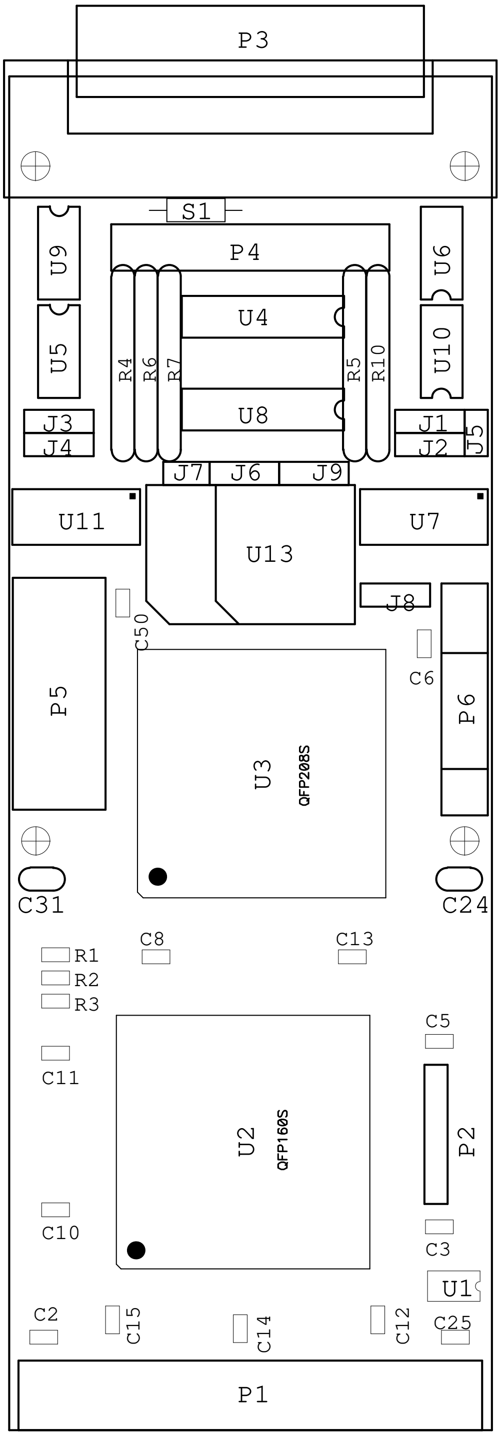


Abbildung 6.3: Network Time Interface Bestückungsplan der Oberseite

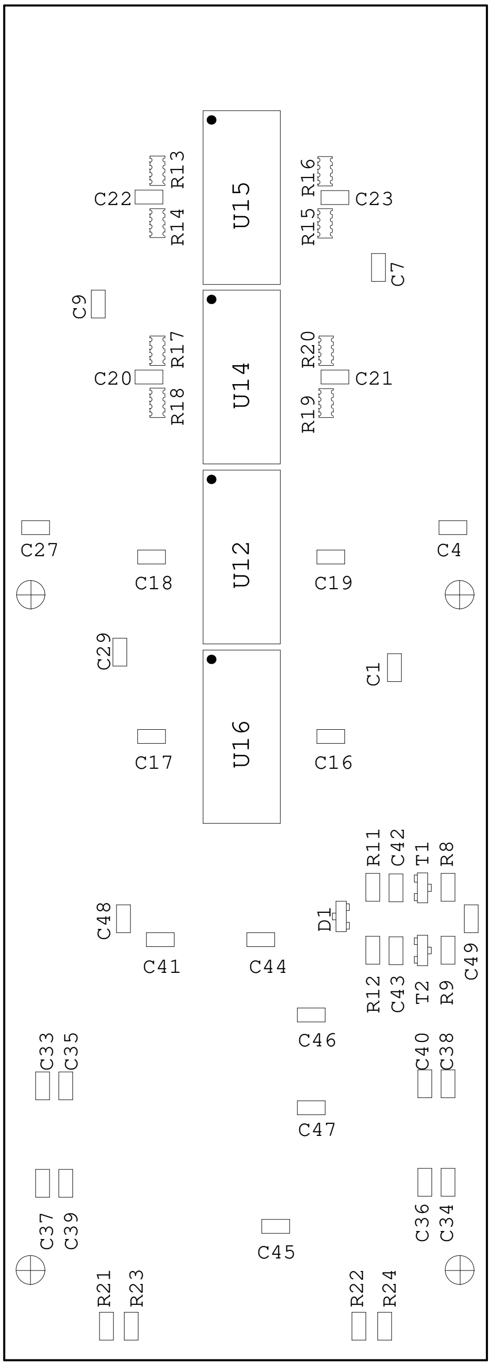


Abbildung 6.4: Network Time Interface Bestückungsplan der Unterseite

6.8 Bauteilliste

Bezeichnung	Typ	Bauteilbeschreibung
C1...C23, C25, C27, C29, C33...C50	Kondensator	100nF SMD Kondensator
C24, C31	Kondensator	10μF SMD Tantal-Kondensator
D1	Diode	BAV99 im SOT-23 Gehäuse
J1...J4, J6, J8, J9	Jumper	Jumper mit 3 Stiften
J5, J7	Jumper	Jumper mit 2 Stiften
P1	Stecker	60-poliger Stiftleistenstecker (female, 3×20)
P2	Stecker	6-polige, abgewinkelte Stiftleiste (1×6)
P3	Stecker	25-poliger D-Sub Stecker (female)
P4	Stecker	24-poliger Stiftleistenstecker (female, 2×12)
P5	Stecker	40-polige Stiftleistenstecker (female, 4×10)
P6	Stecker	20-polige Stiftleistenstecker (female, 2×10)
R1...R3, R8, R9, R11, R12, R21...R24	Widerstand	10kΩ SMD
R4...R7, R10	Widerstand	10kΩ 7-fach Widerstands-Array im SIP8
R13...R20	Widerstand	10kΩ 4-fach SMD Widerstands-Array
S1	Sicherung	125mA Minisicherung
T1, T2	Transistor	BSS138 im SOT-23 Gehäuse
U1	IC	93C06 SPROM im SO8-Gehäuse
U2	IC	XC-95216-10 CPLD im QFP160S-Gehäuse
U3	IC	UTCSU im MQFP208S-Gehäuse
U4, U8	IC	74ACT14 SMD Inverter (DIP14)
U5, U6, U9, U10	IC	HCPL7101 Optokoppler mit 8-poligen Sockeln
U7, U11	IC	74ACT240-Inverter mit 20-poligen Sockeln
U12, U14...U16	IC	CYC7C1021 SRAM's im TSOPII-44 Gehäuse
U13	Quarz	OCXO mit 14-poligem, sehr flachem Sockel

Tabelle 6.8: Bauteilliste des Network Time Interface Moduls

6.9 Bildbeschreibung des NTI-Moduls

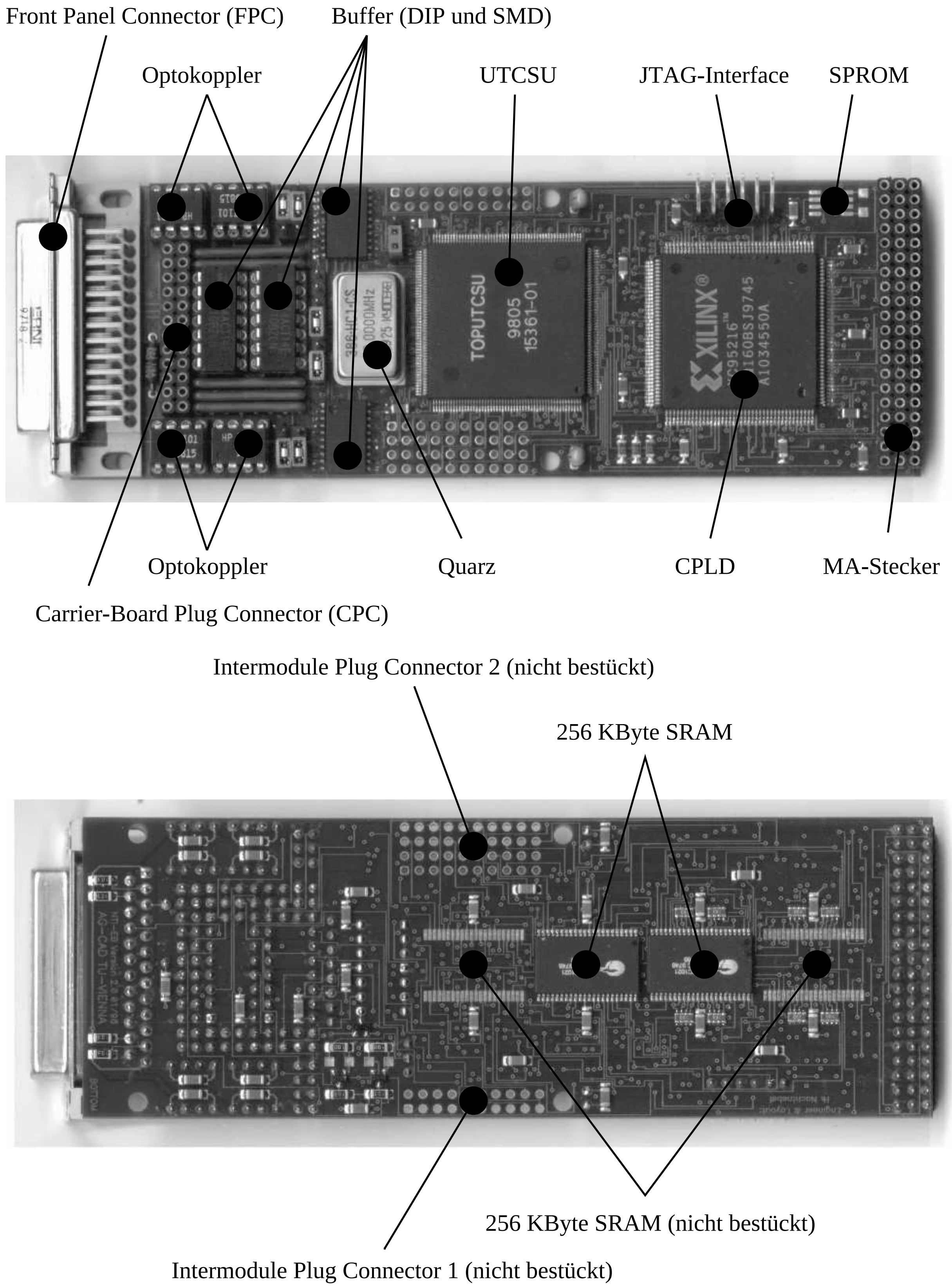


Abbildung 6.5: Bildbeschreibung der Ober- und Unterseite des NTI-Moduls

6.10 Bauteile

In diesem Abschnitt werden die wichtigsten Bauteile des *Network Time Interfaces* beschrieben. Neben einer kurzen Funktionsbeschreibung und den essentiellen technischen Details sind zu jedem Baustein Literaturverweise angegeben, damit genauere Informationen schnell gefunden werden.

6.10.1 CPLD

Um ein flexibles und kompaktes Interface zwischen dem NTI-Modul und einem Carrier-Board realisieren zu können, haben wir ein Complex Programmable Logic Device (CPLD) der Firma XILINX verwendet (siehe [XIL98]).

Wir haben uns für den Typ XC95216 entschieden, der folgende (für uns sehr wichtigen) Eigenschaften besitzt:

- 10ns Pin-to-Pin Logikverzögerung
- 216 Makrozellen mit 4800 verwendbaren Gattern
- 100 benutzerdefinierte I/O-Pins
- 160-pin PQFP-Gehäuse
- 5V-Insystem programmierbar über JTAG-Interface

Ein Baustein der Serie X9500 besteht aus mehreren parallel geschalteten Funktionsblöcken (Abbildung 6.6), von denen jeder 18 Makrozellen (Abbildung 6.7) besitzt. Die 5 bis 9 Produktionsterme dieser

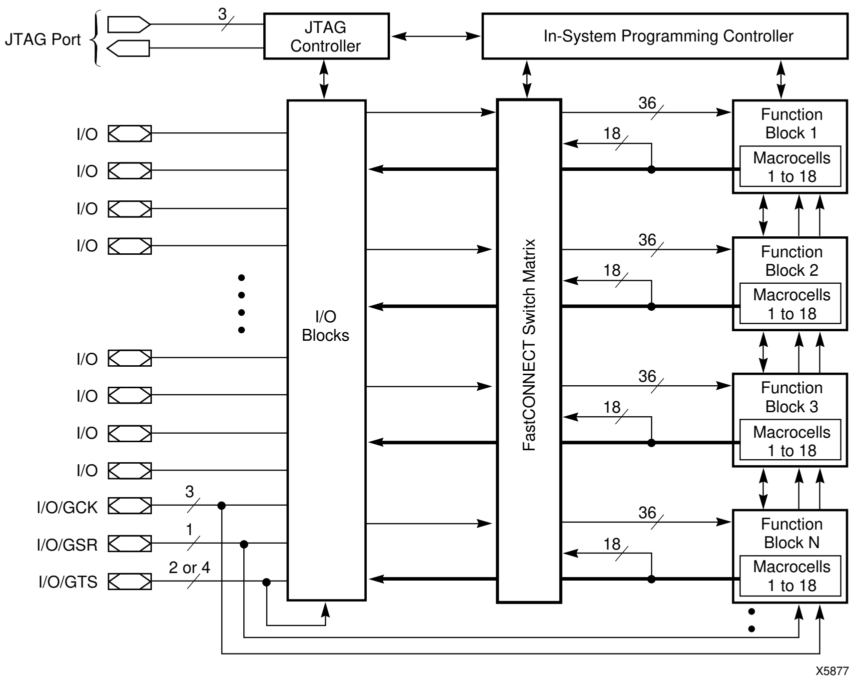


Abbildung 6.6: Architektur der XILINX 9500 Serie

Makrozellen können mit 36 Eingängen verbunden werden (36V18 And-Or Struktur), danach folgt ein wegschaltbares Flip-Flop. Bemerkenswert an dieser Architektur ist, daß nur solche Designs effizient auf sie abgebildet werden können, welche aus reiner Logik bestehen oder bei denen die Logikoperationen direkt vor einem Register liegen.

Die Realisierung eines Designs, das Eingangswerte in ein Register übernimmt und erst danach logische Berechnungen durchführt, führt hingegen zu einer Implementierung, die in zweierlei Hinsicht suboptimal ist: Erstens muß die Synthesesoftware diese Struktur in zwei Makrozellen aufteilen, wobei von der ersten Makrozelle nur das Flip-Flop verwendet wird (die Produktmatrix bleibt ungenutzt), und zweitens verdoppelt durch diese Aufteilung die Laufzeit durch den Baustein. Weiter Informationen zu Optimie-

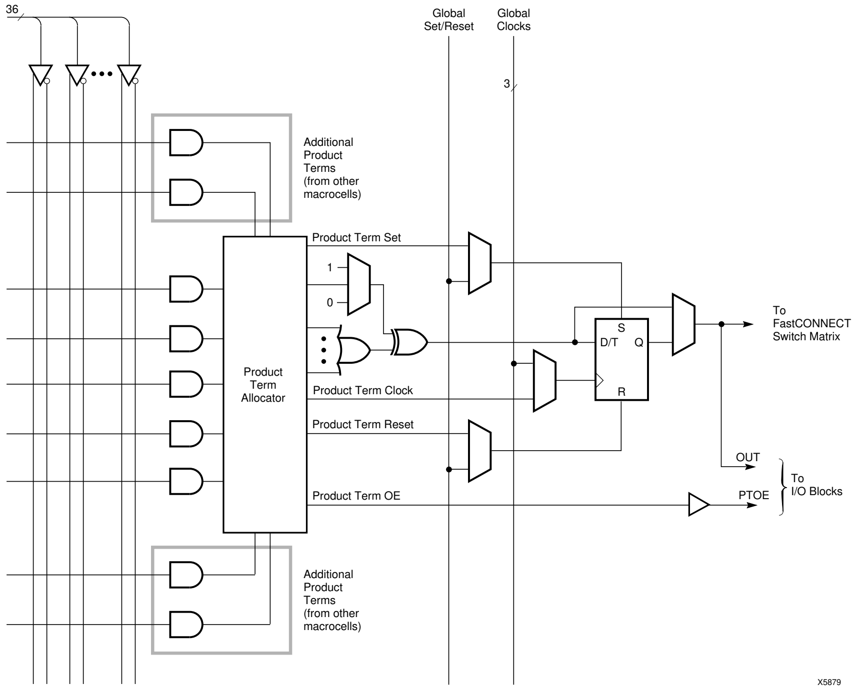


Abbildung 6.7: Makrozelle der XILINX 9500 Serie

rungsmethoden und zur CPLD-Programmierung gibt es in [NKC+98] und [MNS99].

6.10.2 UTCSU (Universal Time Coordinated Synchronization Unit)

Die UTCSU [Loy96] wurde gemeinsam vom Institut für Automation und dem Institut für Computertechnik entwickelt. Die Modellierung des Chips erfolgte in VHDL (Very high speed integrated circuit Hardware Description Language). Mit den Tools Synopsys und Cadence erfolgte die Übersetzung bzw. das Layouting [Hor96], damit der Chip als ASIC produziert werden konnte. Die wichtigsten Produktionsdaten sind in der folgenden Tabelle angeführt.

Technologie:	CMOS 0.7μm
Hersteller:	Amtel-ES2: ECPD07
Tools:	Synopsys/Cadence
Transitoren:	ca. 323.000
Chipfläche:	103.27mm^2
Gehäuse:	MQFP 208
produziert:	Februar 1998
Preis:	34.950 ECU (250 Stück)

Tabelle 6.9: Produktionsdaten der UTCSU

Eine ausführliche funktionelle Beschreibung des Chips erfolgt im Kapitel 3.3. Eine Kurzübersicht der UTCSU kann in [HL97] und in [SL96] nachgelesen werden. Sowohl [SSHL97] als auch [SS95] bieten eine relativ genaue Beschreibung der UTCSU-Funktionalität. Viele theoretischen Details, die genaue Blockstruktur, der interne Aufbau und ein Programmiermodell sind in [Loy96] niedergeschrieben. Eine Kurzbeschreibung des UTClients, dem Prototyp der UTCSU, bietet [HL97].

Abschließend folgt eine grafische Veranschaulichung der UTCSU-Pinbelegung sowie die Erklärung aller Signale. Alle Eingänge und Ausgänge von Abbildung 6.8 werden in Tabelle 6.10 beschrieben. Die Gliederung erfolgt nach den Funktionen, jeweils logisch verwandte Signale sind in einem Block der Tabelle zusammengefaßt.

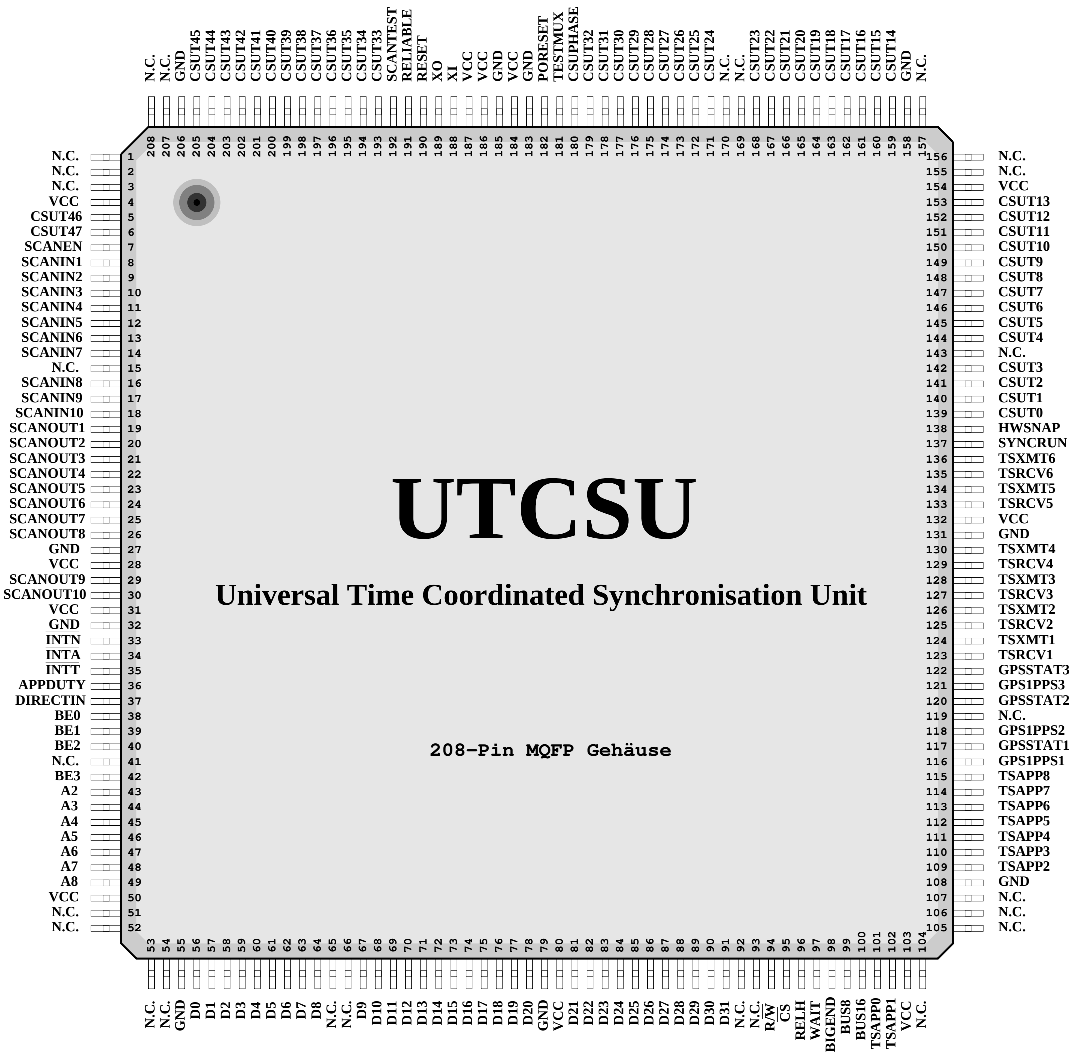


Abbildung 6.8: Pin-Belegung der UTCSU im MQFP-208 Gehäuse

Die erste Signalgruppe mit den Datenleitungen `D0`...`D31`, den Adreßleitungen `A2`...`A8` und den Data Port Control Signalen `BUS8, BUS16, BE0`...`BE3 und BIGEND` dient der Registeransteuerung. Die Signale $\overline{\texttt{CS}}$ und `R/`$\overline{\texttt{W}}$ ermöglichen die Schreib-/Lesefunktionen der UTCSU. Die nächste Funktionsgruppe ist einerseits den Interrupts ($\overline{\texttt{INTN}}$, $\overline{\texttt{INTT}}$, $\overline{\texttt{INTA}}$) als auch dem hardwaremäßigen Start der Uhr (`SYNCRUN`) und dem Hardware-Snapshot (`HWSNAP`) zugedacht. Der nächste Funktionsblock beinhaltet die Zeitstempeleingänge, die von GPS-Receivern (`GPS1PPS1`...`GPS3PPS1, GPSSTAT1`...`GPSSTAT3`), vom Senden (`TSXMT1`...`TSXMT6`) oder Empfangen (`TSRCV1`...`TSRCV6`) oder von Anwendungen (`TSAPP0`...`TSAPP8`) gesteuert werden. Der CSUT-Bus sowie der Scan-Path und ihre Steuersignale bilden die nächsten beiden Signalgruppen. Abschließend sind noch die restlichen Signale und ihre Funktionen erklärt.

Signal	Typ	Bezeichnung	Funktion
D31...D0	In/Out	Datenbus	32-Bit Datenkommunikation
A8...A2	In	Adreßbus	Adressierung der 128 32-Bit Register.
BUS8	In	8-Bit Datenbus	Die Signale BUS8, BUS16, BE3, ..., BE0 und BIGEND werden für dynamische Datenbusbreiten und die verschiedenen Byteordnungen benötigt.[a]
BUS16	In	16-Bit Datenbus	
BE3...BE0	In	Bus Enable Signale	
BIGEND	In	Big/Little Endian	
$\overline{\texttt{CS}}$	In	Chip Select	'LOW' an diesem Eingang schaltet den Chip aktiv.
R/$\overline{\texttt{W}}$	In	Read/Write	'HIGH' signalisiert Lesezugriffe, 'LOW' bedeutet Schreibzugriff.
$\overline{\texttt{INTN}}$	Out	Network Interrupt	'HIGH' bedeutet, daß ein Network Interrupt aktiv ist.
$\overline{\texttt{INTT}}$	Out	Timer Interrupt	'HIGH' bedeutet, daß ein Timer Interrupt aktiv ist.
$\overline{\texttt{INTA}}$	Out	Application Interrupt	'HIGH' bedeutet, daß ein Application Interrupt aktiv ist.
SYNCRUN	In	Synchronous Run	Eine Transition[b] bewirkt einen hardwaremäßigen Start der Uhr (LTU und ACU).
HWSANP	In	Hardware Snapshot	Ein Signalwechsel[b] bewirkt einen Hardware-Snapshot.
GPS1PPS1...GPS1PPS3	In	1pps Signal von GPS X	Eingänge für die 1pps-Impulse von GPS-Empfängern.
GPSSTAT1...GPSSTAT3	In	Status von GPS X	Eingänge für die Statussignale von GPS-Empfängern.
TSXMT1...TSXMT6	In	Transmit Timestamp X	6 Transmit Timestamp Eingänge.
TSRCV1...TSRCV6	In	Receive Timestamp X	6 Receive Timestamp Eingänge.
TSAPP0...TSAPP8	In	Application Timestamp X	9 Application Timestamp Eingänge.
APPDUTY	Out	Application Duty	Ausgang des Application Duty Timers.
TESTMUX	In	Test Multiplexer	Wenn TESTMUX='LOW' ist, dann wird der INTPBUS am CSUT-Bus ausgegeben (Zeitinformation). Wird TESTMUX auf 'HIGH' gesetzt, so liefert der CSUT-Bus interne Steuersignale.
CSUPHASE	Out	CSU Phase	Da der INTP-Bus 96-Bit breit ist, der CSUT-Bus jedoch nur 48-Bit, wird mit jeder steigenden bzw. fallenden Taktflanke eine Hälfte des INTP-Busses ausgegeben. Der Ausgang CSUPHASE signalisiert, welcher Teil des INTP-Busses vom CSUT-Bus geliefert wird ('HIGH' für die obere, 'LOW' für die untere Hälfte.
CSUT47...CSUT0	Out	CSUT Bus	Der 48-Bit breite Clock Synchronisation Unit Time Bus (CSUT-Bus) liefert wahlweise Zeit- oder Steuersignalinformationen.
SCANTEST	In	Scantest Mode	SCANTEST muß während des Scantests auf 'HIGH' gesetzt werden, damit die Software-Reset Logik und der CSU-Time Multiplexer deaktiviert werden.
SCANEN	In	Scantest Enable	'HIGH' an diesem Eingang schaltet den Scan-Test ein.
SCANIN1...SCANIN10	In	Scan Inputs	Eingänge für Scan-Pattern.
SCANOUT1...SCANOUT10	Out	Scan Outputs	Ausgänge der Scan-Pattern.
RESET	In	Reset	Wenn dieses Signal 'HIGH' ist, dann wird der Rücksetzvorgang im Chip ausgeführt.
PORESET	In	Power On Reset	Dieser Eingang dient einerseits als GND-Pad, andererseits zum Erkennen des Einschaltvorganges, damit ein Power-Up Reset durchgeführt wird.
XI	In	Takt In	Takteingang.
XO	Out	Takt Out	Entspricht dem Oszillatortakt vor dem Clock Buffer.
WAIT	In	Wait	'HIGH' an diesem Signal führt bei allen Schreib- und Lesezugriffen zu einem zusätzlichen Wartezyklus.
RELIABLE	In	Reliable	Schaltet alle Synchronisationsstufen von einstufigen auf zweistufige Synchronizer um, wenn RELIABLE='HIGH'.
RELH	In	Reliable Host	Sollte bei Metastabilitätsproblemen des asynchronen Eingangs $\overline{\texttt{CS}}$ auf 'HIGH' gesetzt werden. Damit wird eine Synchronisationsstufe aktiviert.
DIRECTIN	In	Direct In	Wenn die CPU, der COMCO und die UTCSU synchron betrieben werden, können die Synchronisationsstufen ausgeschaltet werden, indem DIRECTIN auf 'HIGH' gesetzt wird.

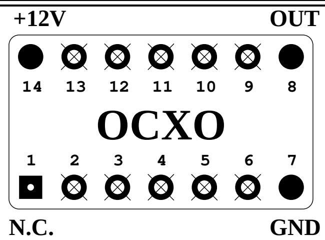
[a] Die genaue Bedeutung kann beispielsweise in [Mot90] nachgelesen werden.
[b] Ob das Signal bei 'LOW'→'HIGH' oder bei 'HIGH'→'LOW' Übergängen aktiv ist, kann im *UTCSU Configuration Register 1* eingestellt werden.

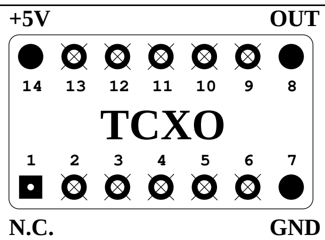
Tabelle 6.10: Eingänge und Ausgänge der UTCSU

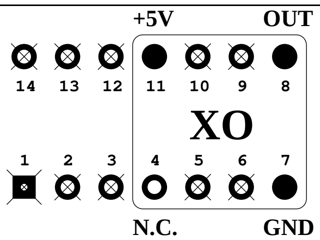
6.10.3 Quarze

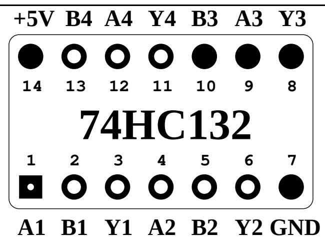
Die Tabelle 6.11 zeigt die verschiedenen Möglichkeiten der Taktgenerierung am NTI-Modul. Für den Normalbetrieb wird ein hochgenauer Oven Controlled Quarz Oscillator (OCXO) verwendet [Mic]. Stattdessen kann das Board auch mit einem Temperature-Compensated Quarz Oscillator (TCXO) oder billigen Standardquarzen (im DIL8- oder DIL14-Gehäuse) betrieben werden. Auch die Taktung von einem GPS-Receiver mit hochgenauem Frequent-Ausgang ist vorgesehen. Alle Frequenzen, die in Tabelle 6.11 angegeben sind, entsprechen den Werten, die wir zum Testen verwendeten, haben aber sonst keine Bedeutung.

Quarz	Frequenz	Pinning	Jumper[a]	Zweck

OCXO	`10MHz`		`J6: 1 - 2`
			`J7: 2 - 3`
			`J9: -`
				Hochgenauer Takt für Uhrensynchronisation

TCXO	2^{24}`Hz`
	(≈ `16.7MHz`)
			`J6: 1 - 2`
			`J7: 1 - 2`
			`J9: -`
				Genauer, durch 2 teilbarer Takt für Uhrensynchronisation

XO	`4, 8, 10, 12,`
	`16, 20, 24,`
	`32, 40MHz`
			`J6: -`
			`J7: -`
			`J9: 1 - 2`
				Ungenauer Takt für Geschwindigkeitstests

10MHz1[b]
10MHz2
	`10MHz`		`J6: -`
			`J7: 1 - 2`
			`J9: -`
				Genauer 10MHz-Takt von GPS-Empfänger 1 oder 2 über 74HC132-Buffer.[c]

[a]Die Jumper `J6, J7 und J9` werden für die Spannungsversorgung der Quarze benötigt. Tabelle 6.7 auf Seite 58 beinhaltet die Bedeutung aller Jumper am *Network Time Interface*.

[b]Das Signal `10MHz1` vom GPS-Empfänger 1 wird über einen Optokoppler an den Anschluß `A3 (Pin 9)` des 74HC132 (Quad-Dual NAND) geführt. Am Anschluß `B3 (Pin 10)` ist der Eingang `10MHz2` vom FPC dirket angehängt. Es darf immer nur ein Eingang beschaltet werden, der andere wird durch einen 10kΩ Widerstand auf `'HIGH'` gezogen.

[c]Der 74HC132-Baustein wird verwendet, da er von der Versorgung kompatibel zu einem TCXO ist und bei richtiger Beschaltung der Eingänge (in unserem Fall `A3 und B3`) den gleichen Ausgangspin (`Y3, Pin 8`) wie die Quarze hat. Ganz wichtig ist allerdings, daß die anderen Ausgangspins (`Pin 3, 6 und 11`) nicht an die Versorgungsspannung angeschlossen sind, daher müssen die Jumper `J6 und J9` entfernt werden!

Tabelle 6.11: Übersicht der Takterzeugung fürs Network Time Interface

6.10.4 Optokoppler

Für unsere Anwendung ist es ganz besonders wichtig Optokoppler mit sehr kurzer Signalverzögerung und sehr konstanter Schaltzeit (Jitter) zu verwenden. Wir haben uns nach langer, intensiver Suche für die HCPL-7101 von HP entschieden (siehe [Hew97]), die eine sehr konstante Schaltzeit von etwa 25ns haben. In einem Versuchsaufbau mit Pattern-Generator und Oszilloskop haben wir die Schaltzeit über längere Zeit aufgezeichnet und die Kurve abgespeichert (siehe Abbildung 6.9).

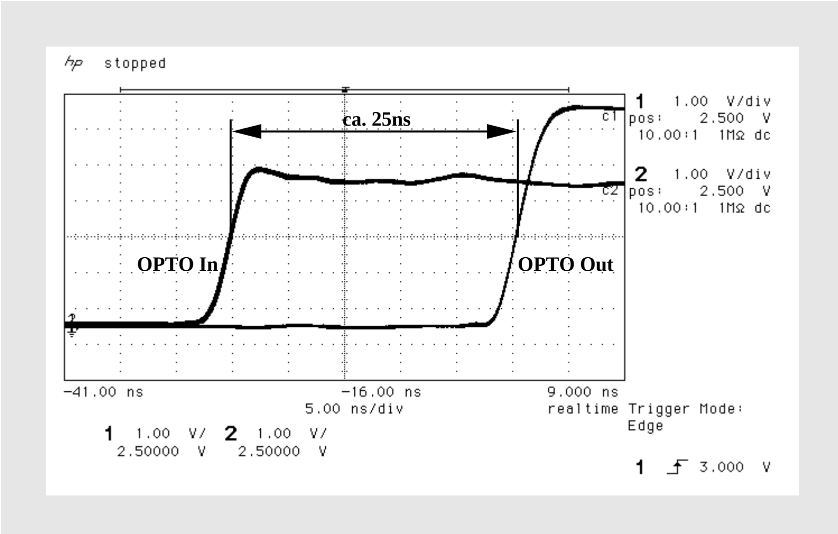


Abbildung 6.9: Jitter-Messung des HCPL-7101 mit HP-Pattern-Generator und Oszilloskop

Für den Schaltungsaufbau gibt es ein paar Details, die unbedingt beachtet werden müssen. So darf etwa der `N.C.-Pin 3` keinesfalls angeschlossen werden. Weiters müssen die Kondensatoren unmittelbar und symmetrisch mit dem Optokoppler verbunden werden.

Wichtig ist auch noch anzumerken, daß die Optokoppler nur dann funktionieren, wenn die beiden Versorgungskreise nicht von der gleichen Spannungsversorgung kommen.

6.10.5 SRAM

Für das Network Time Interface Modul werden vier SRAM-Speicherbausteine mit je 128 KByte, und zwar als 64K $\times$ 16 aufgebaut, von der Firma Cypress verwendet. Die genaue Bauteilbezeichnung lautet CY7C1021, alle Daten über die Speicher können in [CYP98] nachgelesen werden. Wir verwenden die SRAM-Bausteine mit 15ns Zugriffszeit.

Der NTI-Speicher ist so aufgebaut, daß jeweils zwei der oben genannten SRAM-Bausteine eine Speicherbank mit 32 Bit Datenbreite bilden. Die Speicher mit den Bezeichnungen `U12` und `U14` stellen die unteren 128 KByte, die SRAM's `U15` und `U16` die oberen 128 KByte zur Verfügung.

Die Steuerung der SRAM's wird vom CPLD gemacht, und zwar generiert es für die jeweils korrespondierenden Speicher ein gemeinsames $\overline{\texttt{CS}}$ (Chip Select) Signal. Das $\overline{\texttt{WE}}$ (Write Enable) Signal wird für alle vier Speicherbausteine gemeinsam erzeugt und ist ebenfalls `LOW`-aktiv. Um 8 Bit, 16 Bit und 32 Bit Zugriffe auf die Speicher durchführen zu können, muß das CPLD noch die Signale $\overline{\texttt{BE3}}\ldots\overline{\texttt{BE0}}$ erzeugen. Die genaue Beschaltung ist im Abschnitt 6.6 den Schaltplänen zu entnehmen.

Zu beachten ist, daß bei fast allen NTI-Modulen des Evaluationssystems aus Kostengründen nur die unteren 128 KByte bestückt sind. Ebenfalls erwähnenswert ist, daß bei voller Bestückung mit allen SRAM's die obersten 512 Byte des 512 KByte großen Speicherbereichs nicht verfügbar sind, da die UTCSU in denselben Adreßbereich eingebettet ist (siehe Abschnitt 3.5).

6.10.6 SPROM

Das SPROM ist ein 16×16 großes Serial-EPROM, das bei MA-Modulen zur Modul-Identifikation verwendet wird. Laut MA-Spezifikaition (siehe [Man96]) muß ein zum Standard-IC 9306 kompatibler Baustein verwendet werden.

Diese IC's werden über eine sehr einfache Schnittstelle, nämlich mit nur 4 Leitungen, angesprochen. Wie der Name schon verrät, erfolgt die Kommunikation seriell. Die Leitung `CS (Chip Select, Pin 1)` wird verwendet um dem Chip mitzuteilen, daß er zur Datenkommunikation angesteuert wird. An den Eingang `SK (Clock, Pin2)` muß ein Taktsignal angelegt werden, damit die serielle Kommunikation stattfinden kann. Das SPROM benötigt den Takt, um Daten vom Eingang `DI (Data In, Pin3)` zu lesen bzw. um Bits am Ausgang `DO (Data Out, Pin 4)` zu schreiben.

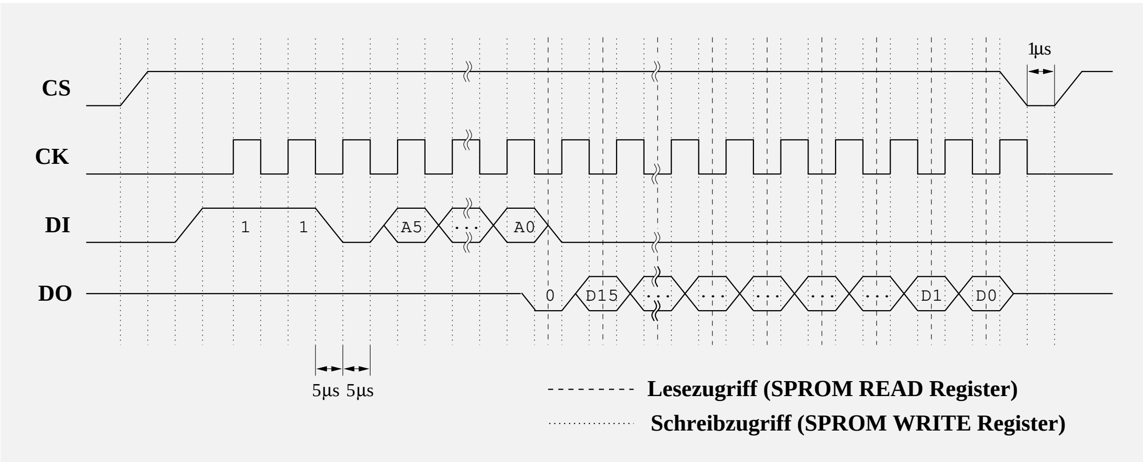


Abbildung 6.10: Zeitdiagramm zum Auslesen des SPROM's

Das genaue Zeitdiagramm für Lesezugriffe ist in Abbildung 6.10 grafisch dargestellt. Nachdem `CS` auf `'HIGH'` gesetzt wurde, beginnt mit dem Anlegen des Taktes an `SK` die serielle Kommunikation. Da serielle EPROM's ein eher langsames Zugriffsverhalten haben, muß darauf geachtet werden, daß der Takt eine Mindestperiodendauer von 10μs hat (d.h. der Pegelwechsel darf schnellstens alle 5μs erfolgen). Beim Schreiben auf das SPROM müssen die Daten stabil sein, während das Taktsignal auf `'HIGH'`-Pegel ist. Lesezugriffe sind nur gültig, wenn der Pegel des Taktsignals auf `'LOW'` liegt. Alle nachfolgend beschriebenen Signalzustände beziehen sich jeweils auf genau eine Periodendauer. Alle Funktionen des 9306 werden eingeleitet, indem das `DI`-Signal auf `'HIGH'` gelegt wird. Die nächsten zwei Bits betreffen den Funktionscode, der bei Lesezugriffen `'10'` ist. Um eines der 16-Datenworte zu adressieren, muß in den nächsten 6 Taktperioden die Adresse (Adreßbits `A5, ..., A0`) angelegt werden. Mit einem `0-Bit` an `DO` beginnt die Lesephase. Die Datenbits `D16, ..., D0` werden in absteigender Reihenfolge ausgegeben. Jede Aktion dieses SPROM's wird beendet, indem das `CS`-Signal für mindestens 1μs auf `'LOW'` gesetzt wird.

Wie die Kommunikation mit dem SPROM in das NTI-Modul eingebettet ist, wird in Kapitel 5.4.7 genau beschrieben. Details zu diesem Baustein können in [Nat96] nachgelesen werden.

Kapitel 7

Probleme und Erweiterungen

Bei der aktuellen NTI-Version gibt es folgende Probleme bzw. Verbesserungsvorschläge, die beim nächsten Entwurf berücksichtigt werden sollten:

- Die Application-Eingangssignale $\overline{\texttt{TSAPP0}}$ und $\overline{\texttt{TSAPP8}}$ sollen getauscht werden, da der $\overline{\texttt{TSAPP0}}$ in der UTCSU eine Sonderstellung hat und somit auch als einziger dieser Eingänge optoelektrisch gekoppelt sein sollte.
- Der zusätzliche Aufwand für ein hardwaremäßiges Einfügen der Receive-Timestamps in den Speicher würde eine große Entlastung der Interrupts bringen.
- Der Dekodierbereich der Transmit-Timestamp Header im CPLD sollte größer als 4K Byte gemacht werden, da die Transmit-Header von 64 Byte auf 128 Byte geändert wurden.
- Alle Buffer sollten als SMD-Bauteile ausgeführt werden, damit die Platzprobleme am NTI-Board reduziert werden.
- Vier I/O-Pins des CPLD's könnten für andere Zwecke verwendet werden, wenn die acht Byte-Enable Signale wieder zusammengelegt werden. Momentan gibt es vier nichtinvertierende Byte-Enable Signale `BE3`...`BE0`, die an die UTCSU geführt sind, und vier weitere invertierte Byte-Enable Signale $\overline{\texttt{BE3}}$...$\overline{\texttt{BE0}}$, die das CPLD steuern.

Weiters sind bisher folgende Probleme mit der UTCSU aufgetreten:

- Die Register `UTCINTEN` und `UTCCONF` sollten auch lesbar gemacht werden, damit keine Software-Schattenregister notwendig sind.
- Im Register `MSSET` sollten der Makrostamp `MS` und der Wert `STEPLOW` entkoppelt werden, d.h. auf zwei Register aufgeteilt werden.
- Die Interrupt-Polarität der UTCSU ist falsch, deswegen müssen die Interrupt-Signale `INTN`, `INTT` und `INTA` im CPLD invertiert werden, um die richtigen Interrupt-Vektoren zu erzeugen.

Anhang A

CPLD-Programmierung des NTI

In diesem Kapitel ist der VHDL-Code – und zwar die Version 2.0.12 – ausgedruckt. Diese und alle anderen Versionen sind im Projektverzeichnis `$SynUTC/TECH/Hardware/NTI/NTI_VERSION_1/CPLD/` abgelegt.

Die Programmierung des CPLD's funktioniert über ein JTAG-Interface, über das der synthetisierte VHDL-Code transferiert wird. Die Details über die Tricks und Tücken der XILINX-Architektur sowie die Dokumentation zum VHDL-Code kann in [MNS99] nachgelesen werden.

A.1 VHDL-Code des Network Time Interface

```
--------------------------------------------------------------------------------
--
--                          NTI Evaluation Board
--
--------------------------------------------------------------------------------
--
--                           Arbeitsgruppe CAD
--         Institut fuer Allgemeine Elektrotechnik und Elektronik
--                      Technische Universitaet Wien
--
--------------------------------------------------------------------------------
--
--                 Abteilung fuer Automatisierungstechnik
--                        Institut fuer Automation
--                      Technische Universitaet Wien
--
--------------------------------------------------------------------------------
--
-- ENTITYNAME:     nti
--
-- ENGINEER:       Herbert Nachtnebel, Gerhard R. Cadek
--
-- DATE:           26. Juni 1998
--
-- VERSION:        2.0.12
--
--------------------------------------------------------------------------------
--
-- DESCRIPTION:    This code describes the function of the CPLD that
--                 controls all operations on the NTI evaluation board.
--                 The CPLD performs mainly memory and I/O address
--                 decoding operations as well as generating select
--                 signals required by external memories, buffers, and
--                 other devices.
--
--------------------------------------------------------------------------------
--
-- HIERARCHY INFORMATION
--
--
--------------------------------------------------------------------------------
--
-- REFERENCES:
--    MODULES:
--
--   PACKAGES:     std_logic_1164
--                 std_logic_arith
--                 XC9000.components
--
--------------------------------------------------------------------------------
```

```
--
-- CHANGES:         Version 2.0.12 - GC - 24. September 1998
--
--          2.0.11: Debugregister implemented.
--          2.0.12: Timestamp-register programmable? Took Debugregister out.
--
-------------------------------------------------------------------------------
library  IEEE;
use      IEEE.std_logic_1164.all;
use      IEEE.std_logic_arith.all;

Entity nti is
   Port( -- MA Interface
         ma_resetbar :   In      std_logic;
         ma_asbar :      In      std_logic;
         ma_csbar :      In      std_logic;
         ma_writebar :   In      std_logic;
         ma_a :          In      std_logic_vector(7 downto 1);
         ma_d :          InOut   std_logic_vector(31 downto 0);
         ma_dsbar :      In      std_logic_vector(2 downto 0);
         ma_dtackbar :   Out     std_logic;
         ma_dreqbar :    Out     std_logic;
         ma_dackbar :    In      std_logic;
         ma_irqbar :     Out     std_logic;
         ma_iackbar :    In      std_logic;

         -- Memory
         a :             Out     std_logic_vector(17 downto 2);
         d :             InOut   std_logic_vector(31 downto 0);
         webar :         Out     std_logic;
         csmem1bar :     Out     std_logic;
         csmem2bar :     Out     std_logic;

         -- UTCSU Interface
         csutcsubar :    Out     std_logic;
         be :            Out     std_logic_vector(3 downto 0);
         bebar :         Out     std_logic_vector(3 downto 0);
         bus16 :         Out     std_logic;
         bus8 :          Out     std_logic;

         inttbar :       In      std_logic;
         intabar :       In      std_logic;
         intnbar :       In      std_logic;

         reset :         Out     std_logic;
         waitcycle :     Out     std_logic;
         ttsrcvbar :     Out     std_logic;
         ttsxmtbar :     Out     std_logic;

         reliable :      Out     std_logic;
         testmux :       Out     std_logic;
         bigend :        Out     std_logic;
         relh :          Out     std_logic;

         -- SPROM Interface
         sp_do :         In      std_logic;
         sp_di :         Out     std_logic;
         sp_cs :         Out     std_logic;
         sp_ck :         Out     std_logic;

         -- Miscellanous
         tclk :          In      std_logic;
         trigger :       Out     std_logic;

         scanen :        Out     std_logic;
         scantest :      Out     std_logic;
         directin :      Out     std_logic

       ); -- end port definition
end nti;

-------------------------------------------------------------------------------
--
--                     Port Signal Description
--
-------------------------------------------------------------------------------
-- Signal Name      Signal Description
-------------------------------------------------------------------------------
--
-- ma_resetbar      master reset signal, asynchronous, low active
-- ma_asbar         master interface: address strobe, low active
-- ma_csbar         master interface: chip select, low active
-- ma_writebar      master interface: write enable, low active
-- ma_a             master interface: address strobe low activebus
--                  (A1 to A7)
-- ma_d             master interface: bidirectional data/address bus,
--                  holds 32 bit data or upper 16 bit address information
-- ma_dsbar         master interface: data strobe, low active
```

```
-- ma_dtackbar       master interface: data take acknowledge, low active
-- ma_dreqbar        master interface: data request, low active
-- ma_dackbar        master interface: data acknowledge, low active
-- ma_irqbar         master interface: interrupt request, low active
-- ma_iackbar        master interface: interrupt acknowledge, low active
-- a                 internal address bus (A2 to A17)
-- d                 internal data bus (D0 to D31)
-- webar             memory, UTCSU: write enable, low active
-- csmem2bar         memory bank 2: chip select signal
-- csmem1bar         memory bank 1: chip select signal
-- csutcsubar        UTCSU: chip select, low active
-- be                UTCSU and memory: bus enable signal (4 bit)
--                   high active for UTCSU
--                   low active for memory
-- bus16             UTCSU: bus enable signal for 16 bit access
-- bus8              UTCSU:bus enable signal for 8 bit access
-- inttbar           UTCSU: interrupt "t", low active
-- intabar           UTCSU: interrupt "a", low active
-- intnbar           UTCSU: interrupt "n", low active
-- reset             UTCSU: reset signal, high active
-- waitcycle         UTCSU: wait signal, high active
-- ttsrcvbar         UTCSU: generate receive time stamp command,
--                   low active
-- ttsxmtbar         UTCSU: generate transmit time stamp command,
--                   low active
-- reliable          UTCSU: reliable mode control signal
-- testmux           UTCSU: unknown signal
-- bigend            UTCSU: big endian control signal
-- relh              UTCSU: reliable host control signal
-- sp_do             SPROM: data output
-- sp_di             SPROM: data input
-- sp_cs             SPROM: chip select
-- sp_ck             SPROM: clock signal
--                   1 .. big endian
--                   0 .. little endian
-- tclk              master clock signal 16 MHz
-- trigger           auxiliary signal for debugging purpose
--
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
--
--                     General Structure
--
-------------------------------------------------------------------------------
-- UTCSU Memory Decoding and Remapping Generation
-------------------------------------------------------------------------------
-- .) 1 M adresses are decoded (0x00000 to 0xfffff).
-- .) The uppermost 512 bytes are mapped to the UTCSU registers
--    (0xffe00 - 0xfffff).
-- .) The upper 512k byte excluding the uppermost 512 bytes are directly
--    mapped to the on board RAM (0x80000 to 0xffdff).
-- .) Lower 512k are mapped to the on board RAM with memory translation
--    (0x00000 to 0x7ffff).
-- .) The transmit remap region occupies the lowest 4k
--    (0x00000 to 0x00fff).
-- .) The receive remap region occupies the rest of the lowest 64k
--    (0x01000 to 0x0ffff).
-- .) In the transmit remap region every access to address (n*0x40 + 0x18)
--    is remapped to UTCSU register 0x148, (n*0x40 + 0x1c) to 0x14c, and
--    (n*0x40 + 0x20) to 0x150.
-- .) If there is a memory access to address (n*0x40 + 0x1c) in the recieve
--    remap region, than the signal ttsrcvbar must be generated.
-- .) If there is a memory access to address (n*0x40 + 0x14) in the transmit
--    remap region, than the signal ttsxmtbar must be generated.
--
-------------------------------------------------------------------------------
-- UTCSU I/O area
-------------------------------------------------------------------------------
-- .) Receive header base address register (0x02):
--                        Holds the address of the last receive timestamp
--                        Read: d(11) ... d(0) <= addr(16) ... addr(5)
-- .) Interrupt vector base register (0x06):
--                        Holds the interrupt vector base
--                        Write: d(7) ... d(3) => vector base
--                        Read: d(15) <= interrupt enable flag,
--                  d(8) ... d(0) <= interrupt vector
-- .) Interrupt enable register (0x0A):
--                        Holds the interrupt enable flag
--                        Write: d(0) => interrupt enable flag
-- .) Logic analyzer trigger register (0x0E):
--                        An access to this address triggers the trigger output
-- .) Miscellanous register (0x12):
--                        Holds diverse signals mapped to each bit
--                        Read:   see below
--                        Write:  see below
-- .) SPROM Register (0xFC and 0xFE):
--          Holds the signals to control the serial prom.
```

```
--                      Write: d(2) => sp_cs, d(1) => sp_ck, d(0) => sp_di
--                      Read:  d(0) <= sp_do

Architecture syn of nti is

  Signal  addr :               std_logic_vector(19 downto 1);
  Signal  addr_remap :         std_logic_vector(8 downto 1);
  Signal  addr_remap_active :  std_logic;

  Signal  mem_access :         std_logic;
  Signal  io_access :          std_logic;
  Signal  dma_access :         std_logic;
  Signal  iack_access :        std_logic;
  Signal  nti_access :         std_logic;
  Signal  utcsu_access :       std_logic;
  Signal  transmit_remap :     std_logic;
  Signal  receive_remap :              std_logic;

  Signal  rxts_match :         std_logic;
  Signal  txts_match :         std_logic;
  Signal  rxtf_match :         std_logic;

  Signal  access_mf :          std_logic_vector(1 downto 0);
  Signal  rx_ts_mf :           std_logic_vector(1 downto 0);
  Signal  tx_ts_mf :           std_logic_vector(1 downto 0);
  Signal  la_trig_mf :         std_logic_vector(1 downto 0);

  Signal  enable_irq :         std_logic;
  Signal  tx_timestamp :       std_logic;
  Signal  tx_timestamp_clk :   std_logic;
  Signal  rx_timestamp :       std_logic;
  Signal  rx_timestamp_clk :   std_logic;
  Signal  rx_typefield :       std_logic;
  Signal  typefield_match :    std_logic;
  Signal  enable_typefield :   std_logic;
  Signal  ttsrcvbar_int :      std_logic;
  Signal  ttsxmtbar_int :      std_logic;
  Signal  tf_byte_match :      std_logic_vector(3 downto 0);
  Signal  tf_byte_match_kept : std_logic_vector(3 downto 0);
  Signal  rfdbase :            std_logic_vector(11 downto 0);
  Signal  tintb :              std_logic_vector(7 downto 3);
  Signal  tinti :              std_logic_vector(2 downto 0);
  Signal  syncint :            std_logic_vector(3 downto 0);
  Signal  memxmt :             std_logic_vector(2 downto 0);
  Signal  bus_enable_select :  std_logic_vector(3 downto 0);
  Signal  be_int :             std_logic_vector(3 downto 0);
  Signal  datarouting :        std_logic_vector(2 downto 0);
  Signal  outreg :             std_logic_vector(15 downto 0);
  Signal  tx_timestamp_base :  std_logic_vector(6 downto 1);
  Signal  rx_timestamp_base :  std_logic_vector(5 downto 1);
  Signal  rx_typefield_base :  std_logic_vector(5 downto 1);
  Signal  debug :              std_logic_vector(15 downto 0);

  -- change these values for another register mapping
  Constant RcvHeaderBaseReg  : std_logic_vector(7 downto 2) := "000000";   -- 0x00, 0x02
  Constant IntVectorBaseReg  : std_logic_vector(7 downto 2) := "000001";   -- 0x04, 0x06
  Constant IntEnableReg      : std_logic_vector(7 downto 2) := "000010";   -- 0x08, 0x0A
  Constant LaTriggerReg          : std_logic_vector(7 downto 2) := "000011";   -- 0x0C, 0x0E
--Constant MiscelleanousReg      : std_logic_vector(7 downto 2) := "000100";   -- 0x10, 0x12
  Constant TimestampBaseReg  : std_logic_vector(7 downto 2) := "000101";   -- 0x14, 0x16
  Constant TypefieldBaseReg  : std_logic_vector(7 downto 2) := "000110";   -- 0x18, 0x1A

  Constant DebugReg          : std_logic_vector(7 downto 2) := "111100";   -- 0xF0, 0xF2
  Constant SPromReg              : std_logic_vector(7 downto 2) := "111111";   -- 0xFC, 0xFE

  -- change these two values for another remapping scheme
  Constant XMTRMPBITS : integer := 12;
  Constant RCVRMPBITS : integer := 16;

  Constant INTVECBASE_DEFAULT : std_logic_vector(7 downto 3) := "01000";

  -- change this value for another typefield compare value
  Constant TF_VALUE : std_logic_vector(15 downto 0) := "0101000000000000";

  -- change this values for other default address base values
  Constant TX_TS_DEFAULT_BASE : std_logic_vector(6 downto 1) := "111000";
  Constant RX_TS_DEFAULT_BASE : std_logic_vector(5 downto 1) :=  "01110";
  Constant RX_TF_DEFAULT_BASE : std_logic_vector(5 downto 1) :=  "01110";

-- a trick needed for pre-layout simulation
-- pragma translate_off
Function "=" (l, r : std_logic) Return boolean is
  TYPE boolean_table IS ARRAY(std_ulogic, std_ulogic) OF boolean;
  Constant resolution_table : boolean_table :=
    (
    --   U      X      0      1      Z      W      L      H      -
      (false, false, false, false, false, false, false, false, false), -- U
      (false, true,  false, false, true,  true,  false, false, false), -- X
```

```
      (false, false, true,  false, false, false, true,  false, true),  -- 0
      (false, false, false, true,  false, false, false, true,  true),  -- 1
      (false, true,  false, false, true,  true,  false, false, false), -- Z
      (false, true,  false, false, true,  true,  false, false, false), -- W
      (false, false, true,  false, false, false, true,  false, true),  -- L
      (false, false, false, true,  false, false, false, true,  true),  -- H
      (false, false, true,  true,  false, false, true,  true,  true)   -- -
    );
begin
  return resolution_table(l,r);
end;
-- pragma translate_on

component KEEP port ( O: out std_logic;
                      I: in  std_logic );
end component;

begin

-------------------------------------------------------------------------------
-- Generate internal master access signals
-------------------------------------------------------------------------------
-- Distinct memory and io accesses
mem_access <= ’1’ when ((ma_csbar = ’0’) and (ma_asbar = ’0’)) else ’0’;
io_access  <= ’1’ when ((ma_csbar = ’0’) and (ma_asbar = ’1’)) else ’0’;

-- Detect other access types
dma_access  <= ’1’ when (ma_dackbar = ’0’) else ’0’;
iack_access <= ’1’ when (ma_iackbar = ’0’) else ’0’;

-- Signals any access type
nti_access <= mem_access or io_access or dma_access or iack_access;

-------------------------------------------------------------------------------
-- Generate internal addressing signals
-------------------------------------------------------------------------------
-- Address remap signals                                 87654321      87654321
--   remap the mem address 0x74 to UTCSU address 0x148:   "111010" => "10100100"
--   remap the mem address 0x78 to UTCSU address 0x14c:   "111100" => "10100110";
--   remap the mem address 0x7C to UTCSU address 0x150:   "111110" => "10101000";
addr_remap <= "1010" & (ma_a(3) and ma_a(2)) & (ma_a(3) nand ma_a(2)) & not(ma_a(2)) & ma_a(1);
addr_remap_active <= ’1’ when (ma_a(6 downto 2) = "11101" or
                               ma_a(6 downto 2) = "11110" or
                               ma_a(6 downto 2) = "11111") else
                     ’0’;

-- Address strobe process
addr_strobe_process : process (ma_resetbar, ma_asbar)
begin
  if (ma_resetbar = ’0’) then

     transmit_remap <= ’0’;
     receive_remap  <= ’0’;
     utcsu_access   <= ’0’;
     addr <= "00000000000000000000";
     txts_match <= ’0’;
     rxts_match <= ’0’;
     rxtf_match <= ’0’;

  elsif (ma_asbar’event and (ma_asbar = ’0’)) then

     if (ma_d(11 downto RCVRMPBITS-8) = conv_std_logic_vector(0,20-RCVRMPBITS)) then
       receive_remap <= ’1’;
     else
       receive_remap <= ’0’;
     end if;

     if (ma_a(6 downto 1) = tx_timestamp_base) then
       txts_match <= ’1’;
     else
       txts_match <= ’0’;
     end if;

     if (ma_a(5 downto 1) = rx_timestamp_base) then
       rxts_match <= ’1’;
     else
       rxts_match <= ’0’;
     end if;

         if (ma_a(5 downto 1) = rx_typefield_base) then
       rxtf_match <= ’1’;
     else
       rxtf_match <= ’0’;
     end if;

     addr(19 downto 9) <= ma_d(11 downto 1);

     -- in the transmit region A19-A12 are ’0’
```

```
        if (ma_d(11 downto XMTRMPBITS-8) = conv_std_logic_vector(0,20-XMTRMPBITS)) then

          transmit_remap <= '1';

          if (addr_remap_active = '1') then
            addr(8 downto 1) <= addr_remap;
          else
            addr(8 downto 1) <= ma_d(0) & ma_a(7 downto 1);
          end if;

              utcsu_access <= addr_remap_active;

        else

          transmit_remap <= '0';

          addr(8 downto 1) <= ma_d(0) & ma_a(7 downto 1);

          if (ma_d(11 downto 1) = "11111111111") then
            utcsu_access <= '1';
          else
            utcsu_access <= '0';
          end if;

        end if;
      end if;
    end process addr_strobe_process;

    --------------------------------------------------------------------------------
    -- Generation of Chip Select Signals
    --------------------------------------------------------------------------------
    -- Generate the chip select signal for the UTCSU.
    csutcsubar <= '0' when ((mem_access = '1') and (utcsu_access = '1'))
                  else '1';

    -- Generate the chip select signals for the two memory banks.
    cs_mem_process: process (mem_access, addr, utcsu_access)
    begin
      if ((mem_access = '1') and (utcsu_access = '0')) then
        if (addr(18) = '0') then
          csmem1bar <= '0';
          csmem2bar <= '1';
        else
          csmem1bar <= '1';
          csmem2bar <= '0';
        end if;
      else
        csmem1bar <= '1';
        csmem2bar <= '1';
      end if;
    end process cs_mem_process;
    --------------------------------------------------------------------------------
    --------------------------------------------------------------------------------

    --------------------------------------------------------------------------------
    -- MA Interface Signal Generation
    --------------------------------------------------------------------------------
    -- Auxiliary signal for controlling the bus enable generation
    -- in a single case statement.
    bus_enable_select <= ma_a(1) & ma_dsbar;

    -- Transceiver control logic.
    transceiver_process: process (ma_resetbar, bus_enable_select, ma_asbar)
    begin
      if (ma_asbar = '1') then                        -- io access
        case bus_enable_select is
          -- 16 bit CPU, 16 bit access, high word
          when "0100" =>  datarouting <= "100";
          -- 16 bit CPU, 8 bit access, high word, high byte
          when "0101" =>  datarouting <= "100";
          -- 16 bit CPU, 8 bit access, high word, low byte
          when "0110" =>  datarouting <= "100";
          -- NEVER OCCURES
          when "0111" =>  datarouting <= "100";

          -- 16 bit CPU, 16 bit access, low word
          when "1100" =>  datarouting <= "100";
          -- 16 bit CPU, 8 bit access, low word, high byte
          when "1101" =>  datarouting <= "100";
          -- 16 bit CPU, 8 bit access, low word, low byte
          when "1110" =>  datarouting <= "100";
          -- NEVER OCCURES
          when "1111" =>  datarouting <= "100";

          -- 32 bit CPU, 16 bit access, high word
          when "0000" =>  datarouting <= "010";
          -- 32 bit CPU, 8 bit access, high word, high byte
```

```
        when "0001" =>  datarouting <= "010";
        -- 32 bit CPU, 8 bit access, high word, low byte
        when "0010" =>  datarouting <= "010";
        -- 32 bit CPU, 32 bit access
        when "0011" =>  datarouting <= "011";

        -- 32 bit CPU, 16 bit access, low word
        when "1000" =>  datarouting <= "100";
        -- 32 bit CPU, 8 bit access, low word, high byte
        when "1001" =>  datarouting <= "100";
        -- 32 bit CPU, 8 bit access, low word, low byte
        when "1010" =>  datarouting <= "100";
        -- 32 bit CPU, 32 bit access
        when "1011" =>  datarouting <= "011";

        -- invalid entries
        when others =>  datarouting <= "000";
      end case;
    else                                                                                   -- memory access
      case bus_enable_select is
        -- 16 bit CPU, 16 bit access, high word
        when "0100" =>  datarouting <= "100";
        -- 16 bit CPU, 8 bit access, high word, high byte
        when "0101" =>  datarouting <= "100";
        -- 16 bit CPU, 8 bit access, high word, low byte
        when "0110" =>  datarouting <= "100";
        -- NEVER OCCURES
        when "0111" =>  datarouting <= "100";

        -- 16 bit CPU, 16 bit access, low word
        when "1100" =>  datarouting <= "001";
        -- 16 bit CPU, 8 bit access, low word, high byte
        when "1101" =>  datarouting <= "001";
        -- 16 bit CPU, 8 bit access, low word, low byte
        when "1110" =>  datarouting <= "001";
        -- NEVER OCCURES
        when "1111" =>  datarouting <= "001";

        -- 32 bit CPU, 16 bit access, high word
        when "0000" =>  datarouting <= "010";
        -- 32 bit CPU, 8 bit access, high word, high byte
        when "0001" =>  datarouting <= "010";
        -- 32 bit CPU, 8 bit access, high word, low byte
        when "0010" =>  datarouting <= "010";
        -- 32 bit CPU, 32 bit access
        when "0011" =>  datarouting <= "011";

        -- 32 bit CPU, 16 bit access, low word
        when "1000" =>  datarouting <= "001";
        -- 32 bit CPU, 8 bit access, low word, high byte
        when "1001" =>  datarouting <= "001";
        -- 32 bit CPU, 8 bit access, low word, low byte
        when "1010" =>  datarouting <= "001";
        -- 32 bit CPU, 32 bit access
        when "1011" =>  datarouting <= "011";

        -- invalid entries
        when others =>  datarouting <= "000";
      end case;
    end if;
  end process transceiver_process;

  -- ma-bus lowest data byte transceiver process
  ma_db0_process : process (ma_csbar, ma_writebar, datarouting,
                            d, iack_access, tintb, tinti)
  begin
    if ( ( (ma_csbar = '0') and (ma_writebar = '1') and
           (datarouting(0) = '1' or datarouting(2) = '1') ) or
         (iack_access = '1') ) then
      if (iack_access = '1') then
        ma_d(7 downto 0) <= tintb & tinti;
      else
        if (datarouting(2) = '1') then
          ma_d(7 downto 0) <= d(23 downto 16);
        else
          ma_d(7 downto 0) <= d(7 downto 0);
        end if;
      end if;
    else
      ma_d(7 downto 0) <= "ZZZZZZZZ";
    end if;
  end process ma_db0_process;

  -- ma-bus lower data word, upper data byte transceiver process
  ma_db1_process : process (ma_csbar, ma_writebar, datarouting,
                            d)
  begin
    if ( (ma_csbar = '0') and (ma_writebar = '1') and
```

```
       (datarouting(0) = '1' or datarouting(2) = '1') ) then
    if (datarouting(2) = '1') then
      ma_d(15 downto 8) <= d(31 downto 24);
    else
      ma_d(15 downto 8) <= d(15 downto 8);
    end if;
  else
    ma_d(15 downto 8) <= "ZZZZZZZZ";
  end if;
end process ma_db1_process;

-- ma-bus upper data word transceiver process
ma_dw1_process : process (ma_csbar, ma_writebar, datarouting, d)
begin
  if ( (ma_csbar = '0') and (ma_writebar = '1') and
       (datarouting(1) = '1') ) then
      ma_d(31 downto 16) <= d(31 downto 16);
  else
    ma_d(31 downto 16) <= "ZZZZZZZZZZZZZZZZ";
  end if;
end process ma_dw1_process;

-- d-bus lower data word transceiver process
dw0_process : process (ma_csbar, ma_writebar, datarouting, ma_d)
begin
  if ((ma_csbar = '0') and (ma_writebar = '0') and (datarouting(0) = '1')) then
    d(15 downto 0) <= ma_d(15 downto 0);
  else
    d(15 downto 0) <= "ZZZZZZZZZZZZZZZZ";
  end if;
end process dw0_process;

-- d-bus upper data word transceiver process
dw1_process : process (ma_csbar, ma_writebar, ma_asbar,
                       datarouting, ma_d, outreg)
begin
  if ( (ma_csbar = '0') and
       (ma_writebar = '0' or ma_asbar = '1') and
       (datarouting(1) = '1' or datarouting(2) = '1') ) then
    if (ma_writebar = '0') then
      if (datarouting(2) = '1') then
        d(31 downto 16) <= ma_d(15 downto 0);
      else
        d(31 downto 16) <= ma_d(31 downto 16);
      end if;
    else
      d(31 downto 16) <= outreg;
    end if;
  else
    d(31 downto 16) <= "ZZZZZZZZZZZZZZZZ";
  end if;
end process dw1_process;

-- Generation of the data acknowledge signal to the MA interface
-- (signal dtackbar). This signal flags the base board that the MA-Module
-- (NTI) is ready to terminate the bus-cycle. This signal is critical due
-- to the fact that it determines the time the units on the NTI are granted
-- to perform their computations. If the UTCSU requires a waitstate
-- this has to be inserted within the following statements.
dtack_generate_process: process (mem_access, io_access, utcsu_access,
                                 iack_access, access_mf)
begin
  if (  -- internal memory is fast enough
        ((mem_access = '1') and (utcsu_access = '0')) or
        -- nti epld access is fast enough
                (io_access = '1') or
        -- iack delayed by 2-3 tclk's since ACQ carrier board have bugs
        (iack_access = '1') or
        -- utcsu access delayed by 3-4 tclk's
        ((mem_access = '1') and (utcsu_access = '1') and (access_mf = "11")) ) then
     ma_dtackbar <= '0';
  else
     ma_dtackbar <= '1';
  end if;
end process dtack_generate_process;

access_mf_process: process (nti_access, tclk)
begin
   if (nti_access = '0') then
      access_mf <= "00";
   elsif (tclk'event and (tclk = '1')) then
      case access_mf is
         when "00"   => access_mf <= "01";
         when "01"   => access_mf <= "10";
         when "10"   => access_mf <= "11";
         when others => access_mf <= "11";
      end case;
   end if;
```

```
end process access_mf_process;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
--  IRQ Logic
--------------------------------------------------------------------------------
-- Strobe the UTCSU interrupt signals every rising edge of tclk.
-- Don't change the contents of the register if there is currently an
-- interrupt acknowledge cylce active.
sync_int_process : process (ma_resetbar, tclk)
begin
  if (ma_resetbar = '0') then
    syncint <= "0000";
  elsif (tclk'event and tclk = '1') then
       syncint(0) <= intabar;
       syncint(1) <= inttbar;
       syncint(2) <= intnbar;
       syncint(3) <= ma_iackbar;
  end if;
end process;

int_process : process (ma_resetbar, tclk, ma_iackbar)
begin
  if (ma_resetbar = '0') then
    tinti <= "000";
  elsif (tclk'event and tclk = '1') then
    if (syncint(3) = '1') then
       tinti(0) <= not syncint(0);
       tinti(1) <= not syncint(1);
       tinti(2) <= not syncint(2);
    end if;
  end if;
end process int_process;

-- Detect a write access to the interrupt vector base at
-- address 0x06. Take the interrupt vector from MA data bus
-- and memorize the data.
write_intvectorbase_process : process (ma_resetbar, ma_csbar)
begin
  if (ma_resetbar = '0') then
    tintb <= INTVECBASE_DEFAULT;
  elsif (ma_csbar'event and (ma_csbar = '1')) then
    if ((io_access = '1') and
        (ma_a(7 downto 2) = IntVectorBaseReg) and
        (ma_writebar = '0')) then
      tintb <= d(23 downto 19);
    end if;
  end if;
end process write_intvectorbase_process;

-- Detect a write access to the interrupt enable register at
-- address 0x0A. Take the interrupt enable flag from MA data bus.
write_enableirq_process : process (ma_resetbar, ma_iackbar, ma_csbar)
begin
  if ((ma_resetbar = '0') or (ma_iackbar = '0')) then
    enable_irq <= '0';
  elsif (ma_csbar'event and (ma_csbar = '1')) then
    if ((io_access = '1') and
        (ma_a(7 downto 2) = IntEnableReg) and
        (ma_writebar = '0')) then
      enable_irq <= d(16);
    end if;
  end if;
end process write_enableirq_process;

-- Generate the interrupt request from the NTI board.
ma_irqbar <= '0' when ((enable_irq = '1') and (tinti /= "000")) else '1';
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
-- UTCSU Signals Generation
--------------------------------------------------------------------------------
-- Read the MA address bus and store the base address of
-- the receive header base.
rfdbase_process: process (rx_timestamp_clk, ma_resetbar)
begin
  if (ma_resetbar = '0') then
    rfdbase <= "000000000000";
  elsif (ttsrcvbar_int'event and (ttsrcvbar_int = '0')) then
          rfdbase(11 downto 0) <= addr(16 downto 5);
  end if;
end process rfdbase_process;

-- Generate the internal transmit remap control signal and
```

```
-- the UTCSU register remap control signals.
tx_timestamp <= '1' when ( (mem_access = '1') and
                           (transmit_remap = '1') and
                           (txts_match = '1') and
                           (ma_writebar = '1') )
                    else '0';

-- Generate the internal receive remap control signal.
rx_timestamp <= '1' when ( (mem_access = '1') and
                           (receive_remap = '1') and
                           (rxts_match = '1') and
                           (transmit_remap = '0') and
                           (ma_writebar = '0') )
                else '0';

-- Generate the internal receive remap control signal.
rx_typefield <= '1' when ( (mem_access = '1') and
                           (receive_remap = '1') and
                           (rxtf_match = '1') and
                           (transmit_remap = '0') and
                           (ma_writebar = '0') )
                else '0';
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- Timestamp Generation
--------------------------------------------------------------------------------
-- Generate a receive timestamp (signal ttsrcvbar) and generate the sample
-- clock for the receive base address register.

-- !!! THE REGISTERS GENERATED IN THIS PROCESS ARE PATCHED TO BUFFERS IN
-- !!! THE XNF-FILE. SO WE CAN FORCE THE FITTER TO KEEP THESE SIGNALS.
tf_byte_match_process : process (d, ma_csbar)
begin
  if (ma_csbar'event and (ma_csbar = '1')) then
    if (ma_d(31 downto 24) = TF_VALUE(15 downto 8)) then
       tf_byte_match_kept(3) <= '1';
    else
      tf_byte_match_kept(3) <= '0';
    end if;
    if (ma_d(23 downto 16) = TF_VALUE(7 downto 0)) then
      tf_byte_match_kept(2) <= '1';
    else
      tf_byte_match_kept(2) <= '0';
    end if;
    if (ma_d(15 downto 8) = TF_VALUE(15 downto 8)) then
      tf_byte_match_kept(1) <= '1';
    else
      tf_byte_match_kept(1) <= '0';
    end if;
    if (ma_d(7 downto 0) = TF_VALUE(7 downto 0)) then
      tf_byte_match_kept(0) <= '1';
    else
      tf_byte_match_kept(0) <= '0';
    end if;
  end if;
end process;

-- tf_byte_match_kept(3) <= '1' when (ma_d(31 downto 24) = TF_VALUE(15 downto 8)) else '0';
-- tf_byte_match_kept(2) <= '1' when (ma_d(23 downto 16) = TF_VALUE( 7 downto 0)) else '0';
-- tf_byte_match_kept(1) <= '1' when (ma_d(15 downto  8) = TF_VALUE(15 downto 8)) else '0';
-- tf_byte_match_kept(0) <= '1' when (ma_d( 7 downto  0) = TF_VALUE( 7 downto 0)) else '0';

typefield_match_process: process (ma_resetbar, ma_csbar)
begin
  if (ma_resetbar = '0') then
     typefield_match <= '0';
  elsif (ma_csbar'event and (ma_csbar = '1')) then
    if (rx_typefield = '1') then
      typefield_match <= (tf_byte_match_kept(3) and tf_byte_match_kept(2)) or
                         (tf_byte_match_kept(1) and tf_byte_match_kept(0));
    end if;
  end if;
end process;

receive_ts_process: process (ma_resetbar, ma_csbar)
begin
  -- Write access to address (n*0x40 + 0x1c) while receive remap is active.
  if (ma_resetbar = '0') then
     rx_timestamp_clk <= '0';
  elsif (ma_csbar'event and (ma_csbar = '1')) then
     if (rx_timestamp = '1') then
          rx_timestamp_clk <= '1';
        else
          rx_timestamp_clk <= '0';
        end if;
  end if;
```

```
end process receive_ts_process;

rx_ts_monoflop_process: process (ma_resetbar, enable_typefield,
                                 rx_timestamp_clk, typefield_match, tclk)
begin
  if (ma_resetbar = '0') then
    rx_ts_mf <= "00";
  elsif ( (rx_timestamp_clk = '1') and
        ((enable_typefield = '0') or
        ((enable_typefield = '1') and (typefield_match  = '1'))) ) then
    rx_ts_mf <= "11";
  elsif (tclk'event and (tclk = '1')) then
    case rx_ts_mf is
      when "11"   => rx_ts_mf <= "10";
      when "10"   => rx_ts_mf <= "01";
      when others => rx_ts_mf <= "00";
    end case;
  end if;
end process rx_ts_monoflop_process;

ttsrcvbar_int <= '1' when (rx_ts_mf = "00") else '0';
ttsrcvbar <= ttsrcvbar_int;

-- Generate a transmit timestamp signal (low active).
transmit_ts_process : process (tx_timestamp, ma_dsbar)
begin
  -- Write access to address (n*0x40 + 0x14) while transmit remap is active.
  if ( tx_timestamp = '1' ) then
    tx_timestamp_clk <= '1';
  else
    tx_timestamp_clk <= '0';
  end if;
end process transmit_ts_process;

tx_ts_monoflop_process: process (ma_resetbar, tx_timestamp_clk, tclk)
begin
  if (ma_resetbar = '0') then
    tx_ts_mf <= "00";
  elsif (tx_timestamp_clk = '1') then
    tx_ts_mf <= "11";
  elsif (tclk'event and (tclk = '1')) then
    case tx_ts_mf is
      when "11"   => tx_ts_mf <= "10";
      when "10"   => tx_ts_mf <= "01";
      when others => tx_ts_mf <= "00";
    end case;
  end if;
end process tx_ts_monoflop_process;

ttsxmtbar_int <= '1' when (tx_ts_mf = "00") else '0';
ttsxmtbar <= ttsxmtbar_int;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
--  Internal base registers
--------------------------------------------------------------------------------
write_timestamp_base_process: process (ma_resetbar, ma_csbar)
begin
  if (ma_resetbar = '0') then
     rx_timestamp_base <= RX_TS_DEFAULT_BASE;
     tx_timestamp_base <= TX_TS_DEFAULT_BASE;
  elsif (ma_csbar'event and (ma_csbar = '1')) then
    if ((io_access = '1') and
        (ma_a(7 downto 2) = TimestampBaseReg) and
        (ma_writebar = '0')) then
        rx_timestamp_base <= d(21 downto 17);
        tx_timestamp_base <= d(30 downto 25);
    end if;
  end if;
end process write_timestamp_base_process;

write_typefield_base_process: process (ma_resetbar, ma_csbar)
begin
  if (ma_resetbar = '0') then
     enable_typefield  <= '1';
     rx_typefield_base <= RX_TF_DEFAULT_BASE;
  elsif (ma_csbar'event and (ma_csbar = '1')) then
    if ((io_access = '1') and
        (ma_a(7 downto 2) = TypefieldBaseReg) and
        (ma_writebar = '0')) then
        enable_typefield  <= d(31);
        rx_typefield_base <= d(21 downto 17);
    end if;
  end if;
end process write_typefield_base_process;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
--  SPROM:
-------------------------------------------------------------------------------
--  The following logic controls the access scheme for the 93C06 E-Eprom
--  that holds the module identification logic.
-------------------------------------------------------------------------------
-- This register holds the clock signal, the input data, and the
-- chip select signal for the SPROM. The software has to write the
-- values on MA data bus bit (2 downto 0) at I/O address 0xFC or 0xFE.
write_sprom_process: process (ma_resetbar, ma_csbar)
begin
  if (ma_resetbar = ’0’) then
    sp_cs <= ’0’;
    sp_di <= ’0’;
    sp_ck <= ’0’;
  elsif (ma_csbar’event and (ma_csbar = ’1’)) then
    if ((io_access = ’1’) and
        (ma_a(7 downto 2) = SPromReg) and
        (ma_writebar = ’0’)) then
      sp_cs <= d(18);
      sp_ck <= d(17);
      sp_di <= d(16);
    end if;
  end if;
end process write_sprom_process;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-- Bus Enable Signals
-------------------------------------------------------------------------------
-- UTCSU port width control signals.
-- This feature will not be used for the NTI board.
-- Select 32 bit access.
-- bus8 <= ’0’;
-- bus16 <= ’0’;

-- Transceiver control logic.
be_int_process: process (bus_enable_select)
begin
  case bus_enable_select is
    -- 16 bit CPU, 16 bit access, high word
    when "0100" => be_int <= "1100";
    -- 16 bit CPU, 8 bit access, high word, high byte
    when "0101" => be_int <= "1000";
    -- 16 bit CPU, 8 bit access, high word, low byte
    when "0110" => be_int <= "0100";
    -- 16 bit CPU, 16 bit access, low word
    when "1100" => be_int <= "0011";
    -- 16 bit CPU, 8 bit access, low word, high byte
    when "1101" => be_int <= "0010";
    -- 16 bit CPU, 8 bit access, low word, low byte
    when "1110" => be_int <= "0001";
    -- 32 bit CPU, 16 bit access, high word
    when "0000" => be_int <= "1100";
    -- 32 bit CPU, 8 bit access, high word, high byte
    when "0001" => be_int <= "1000";
    -- 32 bit CPU, 8 bit access, high word, low byte
    when "0010" => be_int <= "0100";
    -- 32 bit CPU, 32 bit access
    when "0011" => be_int <= "1111";
    -- 32 bit CPU, 16 bit access, low word
    when "1000" => be_int <= "0011";
    -- 32 bit CPU, 8 bit access, low word, high byte
    when "1001" => be_int <= "0010";
    -- 32 bit CPU, 8 bit access, low word, low byte
    when "1010" => be_int <= "0001";
    -- 32 bit CPU, 32 bit access
    when "1011" => be_int <= "1111";
    -- invalid entries
    when others => be_int <= "0000";
  end case;
end process be_int_process;

-- Generation of the bus enable signals.
-- Since there is a routing error on the pcb board, we share the be and be_bar
-- signals and inverse the meaning of the select signals if there is an access
-- to the on board memory.
be    <= be_int when (utcsu_access = ’1’) else not be_int;
bebar <= be_int when (utcsu_access = ’1’) else not be_int;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-- Logic Analyzer Trigger Signal
-------------------------------------------------------------------------------
la_trig_mf_process: process (ma_resetbar, tclk, io_access, ma_a)
```

```
begin
  if (ma_resetbar = '0') then
    la_trig_mf <= "00";
  elsif ( (io_access = '1') and
          (ma_a(7 downto 2) = LaTriggerReg) ) then
    la_trig_mf <= "11";
  elsif (tclk'event and (tclk = '1')) then
    case la_trig_mf is
      when "11"   => la_trig_mf <= "10";
      when "10"   => la_trig_mf <= "01";
      when others => la_trig_mf <= "00";
    end case;
  end if;
end process la_trig_mf_process;

-- Generate a signal to trigger the logic analyzer by software read/write.
-- The trigger pulse is generated when I/O address 0x0E is accessed.
trigger <= '0' when ( la_trig_mf = "00" ) else '1';

------------------------------------------------------------------------------
-- Miscellaneous signals
------------------------------------------------------------------------------
write_debug_process: process (ma_resetbar, ma_csbar)
begin
  if (ma_resetbar = '0') then
     debug <= "0000000000000000";
  elsif (ma_csbar'event and (ma_csbar = '1')) then
    if ((io_access = '1') and
        (ma_a(7 downto 2) = DebugReg) and
        (ma_writebar = '0')) then
      debug(15 downto 0) <= d(31 downto 16);
    end if;
  end if;
end process write_debug_process;

-- Generation of the DMA request signal.
-- Currently not used - set to inactive level.
ma_dreqbar <= '1';

-- UTCSU reset may be forced
reset <= '1' when (ma_resetbar = '0') else '0';

-- Unused Signals
scanen   <= '0';
scantest <= '0';

-- The read/write signal for the UTCSU/memory is a latched copy
-- from the MA ma_writebar signal.
webar <= ma_writebar;

-- Assignments, since output signals are used internally.
a(17 downto 2) <= addr(17 downto 2);

-- Miscellaneous UTCSU signals
bigend    <= '1';
bus8      <= '0';
bus16     <= '0';
testmux   <= '0';
directin  <= '0';
reliable  <= '0';
relh      <= '0';
waitcycle <= '0';
------------------------------------------------------------------------------
------------------------------------------------------------------------------

------------------------------------------------------------------------------
-- Read internal registers
------------------------------------------------------------------------------
-- A read in the I/O region is a two stage process:
-- At the falling edge of ma_cs the internal register are stored in the output
-- register. This is possible, since the addresses are valid during this event.
-- After that, the output register is written to the data bus.
-- The use of this structure minimizes the product terms in the resulting CPLD
-- used for each data bit.
clocked_multiplexer_process: process (ma_a, ma_csbar, ma_resetbar)
begin
  if (ma_resetbar = '0') then
    outreg <= "1111111111111111";
  elsif (ma_csbar'event and (ma_csbar = '0')) then
    case ma_a(7 downto 2) is
      -- read access at addr 0xFC or 0xFE: read sprom bit
      when SPromReg  => outreg <= "000000000000000" & sp_do;

      -- read access at addr 0x00 or 0x02: read receive header base address
      when RcvHeaderBaseReg  => outreg <= "0000" & rfdbase;

      -- read access at addr 0x04 or 0x06: read interrupt vector base
      when IntVectorBaseReg  => outreg <= enable_irq & "0000000" & tintb & tinti;
```

```
        -- Note! read access at addr 0x0C or 0x0E triggers logic analyzer

        -- read access to timestamp base registers (0x14, 0x16)
        when TimestampBaseReg  => outreg <= '0'  & tx_timestamp_base & '0' &
                                          "00" & rx_timestamp_base & '0';

        -- read access to typefield base registers (0x14, 0x16)
        when TypefieldBaseReg  => outreg <= enable_typefield & "0000000" &
                                          "00" & rx_typefield_base & '0';

        -- read access to debug registers (0xF0, 0xF2)
--      when DebugReg => outreg <= debug;

        -- default: no bits read
        when others => outreg <= "1111111111111111";
      end case;
    end if;
end process clocked_multiplexer_process;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

end syn;
```

Anhang B

Tests für die Inbetriebnahme

In diesem Abschnitt wird der Testplan für die Inbetriebnahme des *Network Time Interfaces (NTI)* vorgestellt. Zu jedem Testfall sind das Testziel, die Hardwareumgebung, die Testmuster und eine Testanleitung genau spezifiziert.

Alle Testfälle sollen mit verschiedenen (und sinnvollen) Testmustern und mit variablen Geschwindigkeiten (Single-Step oder permanenter Zugriff ohne Verzögerung) durchgeführt werden. Weiters müssen die Modulzugriffe mit verschiedenen Datenbreiten getestet werden.

Beim NTI werden, nachdem Low-Level Hardwaretests durchgeführt wurden, I/O-Space Zugriffe getestet. Danach werden Speicherzugriffe ausgeführt, um die korrekte CPLD-Funktionalität für den MA-Memory-Space zu überprüfen. Abschließend werden die UTCSU, die externen Eingänge und die Interruptlogik getestet. Die NTI-Tests können als positiv abgeschlossen betrachtet werden, wenn mehrere *Sample Rounds* (d.h. alle Aufgaben, die für eine Uhrensynchronisationsrunde benötigt werden) fehlerfrei funktionieren.

Abschließend wird in diesem Kapitel ein kurzer Verweis auf ein „Quick and Dirty“-Testprogramm gegeben, das die wichtigsten Funktionen des NTI-Moduls testet.

B.1 Tests ohne CPU-Unterstützung

Dieser Abschnitt umfaßt alle Tests, die ohne ein Prozessorboard ausgeführt werden können und damit jede Menge Ärger bei den komplexeren Tests ersparen helfen.

B.1.1 Kurzschlußtest

Um die Zerstörung von teuren Bauteilen auf dem *Network Time Interface* bzw. des Carrier-Boards durch falsche Leitungsführung oder Kurzschlüsse zu vermeiden, sollten noch vor dem Anlegen einer Spannung einige Tests durchgeführt werden.

Vor der Bestückung der Leiterplatte sollten die wichtigsten Signale auf Kurzschlüsse geprüft werden. Dazu sollten beispielsweise alle Signale des MA-60 Steckers und der anderen externen Interfaces (FPC, CPC und IPC) mit einem Durchgangsprüfer paarweise auf Kontakt getestet werden. Wenn der gleiche Vorgang bei allen CPLD-Pins wiederholt wird und keine ungewollte Leitungsbeeinflußung festgestellt wird, ist der Test positiv abgeschlossen. Besondere Vorsicht ist bei der `+12V`-Leitung geboten, da diese Spannung alle IC's ins „Nirvana“ befördert, an denen sie irrtümlich anliegt.

Insbesondere nach dem Einlöten der SMD-Bauteile (UTCSU, CPLD, etc.) müssen jeweils benachbarte Pins der IC's auf Kurzschlüsse getestet werden, um ungewollte Signalkopplungen auszuschließen.

B.1.2 Optokoppler

Die Optokoppler (OC) können vorab getestet werden, indem das Modul mit `+5V` betrieben wird und die Primärseite der OC mit einem zweiten, vom Potential getrennten `+5V`-Stromkreis versorgt wird. Durch Anlegen eines Rechtecksignals von einem Funktionsgenerator am Eingang des Optokopplers muß am Ausgang ein um etwa 25ns verzögertes Rechtecksignal mit TTL-Pegel mit einem Oszilloskop gemessen werden.

Dieser Test ist notwendig, da bei der Inbetriebnahme des Prototyp-Boards die Optokoppler nicht funktioniert haben, da irrtümlich der `N.C.-Pin 3` an `GND` gelegt war und die parallel zu den Versorgungspins notwendigen Kondensatoren zuweit weg platziert waren.

Bei den Optokopplern *U5, U6 und U9* für die Signale des GPS-Empfängers 1 kann die externe `+5V`-Versorgung an den Pins `+5V GPS1` und `GND GPS1` des FPC angelegt werden. Das Rechtecksignal wird beim *U5* an `1pps GPS1`, bei *U6* an `10 MHz GPS1` und bei *U9* an `Status GPS1` angelegt (siehe Front Panel Connector im Abschnitt 6.2).

Der Optokoppler *U10* für das $\overline{\texttt{APP8}}$-Signal wird von den Pins `+5V App8` und `GND App8` versorgt. Das korrespondierende Eingangssignal am FPC ist $\overline{\texttt{App8}}$, das Ausgangssignal kann wie bei allen OC direkt am `Pin 6` gemessen werden. Wird das Signal hingegen am korrespondierenden UTCSU-Pin gemessen, so ist es invertiert und um *(25ns + 1 Gatterlaufzeit)* verzögert, da es zusätzlich über einen invertierenden Puffer geführt wird.

B.1.3 Externe Interfaces

Bei diesen Tests sollen alle externen Eingänge des *Network Time Interfaces*, die nicht über Optokoppler sondern über Puffer geführt sind, überprüft werden. Dies kann erfolgen, indem an einem Eingangspin des FPC oder CPC ein Rechtecksignal angelegt wird und am Pad des zugehörigen UTCSU-Eingangs mit einem Oszilloskop aufgezeichnet wird. Da alle Puffer invertierend sind, muß das Signal an den UTCSU-Pins im Vergleich zum Eingangssignal invertiert und etwa eine Gatterlaufzeit verzögert sein.

Vom *Front Panel Connector (FPC)* sollen die Signale $\overline{\texttt{1pps GPS2}}$, $\overline{\texttt{Status GPS2}}$, `10MHz GPS2`, $\overline{\texttt{1pps GPS3}}$, $\overline{\texttt{Status GPS3}}$ und $\overline{\texttt{App0}}$, …, $\overline{\texttt{App7}}$ auf diese Art geprüft werden.

Die Signale $\overline{\texttt{HWSNAP}}$, $\overline{\texttt{SYNCRUN}}$, $\overline{\texttt{TSRCV1}}$, …, $\overline{\texttt{TSRCV6}}$, $\overline{\texttt{TSXMT1}}$, …, $\overline{\texttt{TSXMT6}}$ [1] und $\overline{\texttt{TSAPP0}}$, …, $\overline{\texttt{TSAPP8}}$ [2] des CPC werden ebenso getestet.

Zusätzlich soll der Ausgang $\overline{\texttt{APPDUTY}}$ getestet werden, indem ein Rechtecksignal am UTCSU-Ausgang `APPDUTY` angelegt wird und am Pin $\overline{\texttt{APPDUTY}}$ des FPC's oder an $\overline{\texttt{APPDUTY}}$ am CPC gemessen wird. Das Ausgangssignal ist etwa das invertierte Eingangssignal.[3]

B.1.4 Taktverteilung

Die vielfältige Taktgenerierung und vor allem die Signalform des Taktes soll mit einem Oszilloskop verifiziert werden.

Für alle möglichen Takte soll am `Pin 8` des Quarzsockels *U13* das Ausgangssignal mit einem Oszilloskop gemessen werden. Interessant erscheint es auch, das Signal an den Takteingängen der UTCSU und des CPLD's zu prüfen. Besonderes Augenmerk soll neben der Signalform auf die korrekte Frequenz gelegt werden. Diese könnte etwa mit einem Frequenzzähler gemessen werden.

Neben dem OCXO müssen auch Standardquarze und die 10MHz-Eingänge der GPS-Empfänger (`10MHz GPS1` und $\overline{\texttt{10MHz GPS2}}$ am FPC) getestet werden. Die 10MHz-Signale sollen als Rechtecksignal von einem Funktionsgenerator am *Front Panel Connector* eingespeist werden, in den Sockel *U13* muß ein 74HC132 gesteckt werden. Wichtig ist, daß die Jumper *J6, J7 und J9* jeweils richtig gesetzt sind (siehe Tabelle 6.11).

B.1.5 Messung des Ruhestromverbrauchs

Nach der vollständigen Bestückung des *NTI-Modules* soll der Ruhestromverbrauch gemessen werden. Dies kann geschehen, indem das Modul von einem Netzgerät mit `+5V` versorgt wird und die Stromaufnahme mit einem Amperemeter (besser wäre eine hochempfindliche Stromzange, die bis in den MHz-Bereich mißt) gemessen wird. Wichtig ist, daß das Modul getaktet wird (d.h. wenn der OCXO bestückt ist, muß auch `+12V` angelegt werden) und der **gemessene Stromverbrauch deutlich kleiner als 1A** ist. Bei

[1] Bei den Signalen $\overline{\texttt{TSXMT1}}$, $\overline{\texttt{TSRCV1}}$, $\overline{\texttt{TSXMT2}}$ und $\overline{\texttt{TSRCV2}}$ ist zu beachten, daß sie über die Jumper `J1, ..., J4` geführt sind (siehe Tabelle 6.7).

[2] Achtung: Der Eingang $\overline{\texttt{TSAPP8}}$ wird direkt zu einem Buffer geführt und hängt deshalb am Ausgang des Optokopplers *U10*. Beim Test dieses Eingangssignals muß deshalb dieser OC aus dem Sockel gezogen werden!

[3] Der Jumper *J8* muß dabei in der Stellung **1 - 2** sein, damit das Signal zu den Steckern geführt wird.

zu großem Stromverbrauch[4] kann es zur Zerstörung des Carrier-Boards kommen, da dieses laut [Man96] maximal 1A pro MA-Slot liefern muß.

B.2 Tests der MA-Schnittstelle

Für diese Tests wird es unumgänglich sein, einen Logikanalysator zum Aufzeichnen des Zeitverhaltens der MA-Signale anzuschließen. Damit dies möglich ist, muß die Testumgebung aus folgenden Komponenten aufgebaut sein:

VME-Rack

Motorola MVME162-512A CPU-Board

AcQ i6360 + i6Ext Carrier-Board mit VME-Zwischensteckern, um möglichst viel Platz zum Messen zu haben.

NTI MA-Modul bei dem der MA-60 Stecker als gewinkelter Wire-Wrap Stecker ausgeführt ist, damit an den Wires Logikanalysator-Pods bzw. Tastköpfe eines Oszilloskops angehängt werden können. Das Modul darf nur mit den Speicherchips *U12 und U14* bestückt sein, da die anderen beiden Footprints mit Adaptern versehen werden, die alle Daten- und Adreßleitungen sowie die Steuersignale der Speicher zum Messen zugänglich machen.

Logikanalysator zum Aufzeichnen der Signale, um ihr richtiges logisches Verhalten und das Timing verifizieren zu können.

Oszilloskop zum Messen der Signalformen und des genauen Zeitverhaltens der Signale.

Schrittweise sollen nun alle I/O-Space und Memory-Space Zugriffe auf das NTI-Modul getestet werden.

B.2.1 I/O-Space Zugriffe

Die Funktionen des NTI-Moduls, die im I/O-Space realisiert sind, sollen schrittweise (vom einfachsten beginnend) getestet werden.

Ohne großen Aufwand kann der `Logic Analyzer Trigger` an der Adresse `0x0E` sowohl für Lese- als auch für Schreibzugriffe getestet werden. Am `Trigger`-Ausgang des CPLD's muß bei jedem Zugriff ein Trigger-Puls ausgegeben werden. Mit dem Oszilloskop kann das einfach am `Pin 2` des Jumpers `J5` gemessen werden. Die Länge des Trigger-Impulses muß etwa solange wie der I/O-Space Zugriff dauern (abgesehen von den Schaltzeiten). Mit dem Logikanalysator sollte das leicht zu verifizieren sein.

Als nächstes soll das `MISCELLANEOUS`-Register, das an der Adresse `0x12` angeordnet ist, getestet werden. Bei Schreibzugriffen hat es die Funktion, einige UTCSU-Signale mit Sonderfunktionalität auf `'LOW'`- oder `'HIGH'`-Pegel zu setzen (siehe Abschnitt 5.4.4). Beim Lesen dieses Registers wird der Sollpegel der Signale ausgegeben. Sollpegel bedeutet in diesem Fall, daß das CPLD die Signale nicht einliest, sondern nur den intern gespeicherten, und auch nach außen getriebenen Wert an die Datenleitungen legt. Der Test soll nun so erfolgen, daß zuerst ein Hardware-RESET durchgeführt wird, um das Register mit `0x0001` zu initialisieren. Dann soll jedes Signal durch Schreibzugriffe auf `'HIGH'` bzw. wieder auf `'LOW'` gesetzt werden. Die Pegel können mit einem Voltmeter oder mit dem Oszilloskop an den korrespondierenden UTCSU-Pins gemessen werden. Wichtig ist, daß nach jedem Schreibzugriff ein Lesezugriff stattfindet und die Signalwerte verglichen werden.

Nun sollen die Register für die Interruptgenerierung getestet werden. Zunächst einmal wird ein Lesezugriff auf das Register `Get Interrupt Vector Base` (Adresse `0x06`) gemacht, der als Ergebnis `0x40` liefern sollte, wenn kein UTCSU-Interrupt aktiv ist (vorher UTCSU-Reset ausführen). Durch einen Schreibzugriff auf `Set Interrupt Vector Base` (ebenfalls an Adresse `0x06` platziert) kann eine neue Interrupt-Basisadresse konfiguriert werden, die leicht durch Lesen des `GET INT.-VECTOR BASE`-Registers verifiziert werden kann. Um die Interruptgenerierung des NTI-Moduls ein/ausschalten zu können, gibt es das Register `EN/DIS INT.-LOGIC` an der Adresse `0x0A` im I/O-Space. Wird der Wert `0x0001` in dieses Register

[4]Bei der CMOS-Logik steigt der Stromverbrauch mit der Anzahl der Schaltvorgänge, deshalb ist im Vollastbetrieb ein deutlich höherer Stromverbrauch zu befürchten!

geschrieben, so ist die Interruptlogik aktiviert. Durch Schreiben eines Wertes, bei dem das LSB = `'0'` ist, wird die Interruptgenerierung wieder ausgeschaltet. Ob die Interrupt-Logik ein- oder ausgeschaltet ist, kann geprüft werden, indem das MSB des `GET INT.-VECTOR BASE`-Registers gelesen wird (`'1'` bedeutet aktiv).

B.2.2 Memory Space Zugriffe

Diese Art von MA-Zugriffen hat die Besonderheit, daß ein Teil der Adressen auf den Datenleitungen gemultiplext wird. Das CPLD hat die Aufgabe mit der fallenden Flanke von $\overline{\texttt{AS}}$ die Adressen `A8...A23` an den Datenleitungen `D0...D15` zu latchen. Es erscheint sinnvoll, gleich mit den Speichertests und später mit den UTCSU-Tests fortzusetzen, ohne die Signale vorher zu messen. Anhand der nächsten Tests wird ohnehin ersichtlich, ob das Timing der Zugriffe und das Adreßlatching funktioniert.

Bei den UTCSU-Zugriffen muß darauf geachtet werden, daß die Zugriffe, bei denen Register gepuffert und dann gelesen werden, einen Taktzyklus länger als normale UTCSU-Zugriffe brauchen. Die Speicherzugriffe sollten generell am schnellsten funktionieren. Mit dem Logikanalysator und High-Perfomance Tests soll ein optimales Timing realisiert werden.

B.3 Speichertests

B.3.1 Einfache Speichertests

Test 1: Read-Write Test im COMCO-Bereich des Speichers
Chip: `kein Chip` *Testmuster:* `???` *ext. Hardware:* `keine`

Testablauf: sinnvolle Testmuster mit 8/16/32-Bit Datenbreite schreiben • lesen (8/16/32-Bit Zugriffe) und vergleichen • überprüfen, ob andere Bereiche beeinflußt wurden

Test 2: Gemischter Read-Write Test 1[5]
Chip: `kein Chip` *Testmuster:* `???` *ext. Hardware:* `keine`

Testablauf: sinnvolle Testmuster mit 8/16/32-Bit Datenbreite in den CPU-Bereich schreiben • vom COMCO-Bereich lesen (16/32-Bit Zugriffe) und vergleichen • überprüfen, ob andere Bereiche beeinflußt wurden

Test 3: Gemischter Read-Write Test 2[5]
Chip: `kein Chip` *Testmuster:* `???` *ext. Hardware:* `keine`

Testablauf: sinnvolle Testmuster mit 16/32-Bit Datenbreite in den COMCO-Bereich schreiben • vom CPU-Bereich lesen (8/16/32-Bit Zugriffe) und vergleichen • überprüfen, ob andere Bereiche beeinflußt wurden

Test 4: Read-Write Test im COMCO-Bereich des Speichers
Chip: `kein Chip` *Testmuster:* `???` *ext. Hardware:* `keine`

Testablauf: sinnvolle Testmuster mit 16/32-Bit Datenbreite schreiben • lesen (8/16/32-Bit Zugriffe) und vergleichen • überprüfen, ob andere Bereiche beeinflußt wurden

Test 5: Transmit Timestamp Triggering 1
Chip: `kein Chip` *Testmuster:* `???` *ext. Hardware:* `Oszilloskop`

Testablauf: Oszilloskop am `Pin 3` des Jumpers *J1* anhängen • mit 16/32-Bit Datenbreite im Transmit Header Bereich (`0x00000...0x01FFF`) des COMCO-Teils an den Adressen $n * 0x040 + 0x014$ lesen • Trigger-Impuls am Oszi verifizieren

Test 6: Receive Timestamp Triggering 1
Chip: `kein Chip` *Testmuster:* `???` *ext. Hardware:* `Oszilloskop`

[5] Das Vergleichen der Speichereinträge im CPU- und COMCO-Teil ist natürlich nur dort sinnvoll, wo kein Transparent Mapping stattfindet!

Testablauf: Oszilloskop am `Pin 3` des Jumpers *J2* anhängen • mit 16/32-Bit Datenbreite im Receive Header Bereich (`0x02000...0x1FFFF`) des COMCO-Teils an den Adressen $n * 0x040 + 0x01C$ lesen • Trigger-Impuls am Oszi verifizieren

Test 7: Transmit Timestamp Triggering 2[6]

Chip: `UTCSU` *Testmuster:* `???` *ext. Hardware:* `keine`

Testablauf: Chip für `TSXMT1`-Interrupts initialisieren • mit 16/32-Bit Datenbreite im Transmit Header Bereich (`0x00000...0x01FFF`) des COMCO-Teils an den Adressen $n * 0x040 + 0x014$ lesen • nach jedem Lesezugriff auf das Anstossen der Interrupt Service Routine warten

Test 8: Receive Timestamp Triggering 2[6]

Chip: `UTCSU` *Testmuster:* `???` *ext. Hardware:* `keine`

Testablauf: Chip für `TSRCV1`-Interrupts initialisieren • mit 16/32-Bit Datenbreite im Receive Header Bereich (`0x02000...0x1FFFF`) des COMCO-Teils an den Adressen $n*0x040+0x01C$ lesen • nach jedem Lesezugriff auf das Anstossen der Interrupt Service Routine warten

Test 9: Transparent Mapping[6]

Chip: `UTCSU` *Testmuster:* `???` *ext. Hardware:* `keine`

Testablauf: Chip in *Halt Mode* bringen • Chip für `TSXMT1` initialisieren • jeweils einen ganzen Transmit Header aus dem COMCO-Bereich mit 16/32-Bit Datenbreite auslesen • prüfen, ob die Zeitstempel `MSXMT`, `TSXMT` und `ACCXMT` der *SSU1* an den richtigen Stellen $n * 0x040 + 0x018/0x01C/0x020$ eingefügt sind

B.3.2 MATS+ und March-B Speichertests

Mit Hilfe dieser beiden Tests können beinahe alle Speicherfehler (z.B. Fehler in der Adressierungslogik, Übersprechen von Leitungen, etc.) lokalisiert werden. Die Algorithmen und die Erklärung der Tests (und auch vieler anderer) kann im Kapitel 4 von [vdG94] nachgelesen werden. Prinzipiell sind die Tests so aufgebaut, daß Schreib- und Lesezugriffe in auf- bzw. absteigender Reihenfolge mit Datenworten und den invertierten Datenworten durchgeführt werden.

B.4 Tests der wichtigsten UTCSU-Funktionen

Mit Hilfe von [Loy96] können die wichtigsten Tests der UTCSU erstellt werden. Als günstig wird erachtet, wenn eine Unit nach der anderen auf ihr korrektes Funktionieren hin überprüft wird. Sofern die UTCSU richtig adressiert werden kann, haben diese Tests (fast) nichts mit der richtigen Funktion des NTI-Moduls zu tun, sondern entsprechen einer vollständigen Überprüfung der UTCSU-Funktionen. Das ist allerdings nicht Gegenstand dieser Arbeit.

B.5 Interrupt Tests

Bei den Interrupt-Tests soll einerseits verifiziert werden, ob die drei Interruptleitungen $\overline{\texttt{INTN}}$, $\overline{\texttt{INTT}}$ und $\overline{\texttt{INTA}}$ der UTCSU vom CPLD zu einem vektorisierten MA-Interrupt vereint werden. Andererseits gilt es zu testen, wie sich die Interruptlogik verhält, wenn mehrere Interruptquellen gleichzeitig oder kurz hintereinander ausgelöst werden.

B.6 Lesen des SPROM's

Der letzte und unwichtigste Test betrifft die Modulidentifikation. Am *Network Time Interface* ist ein 9306 Serial PROM (siehe Abschnitt 6.10.6) in den MA-I/O Space integriert. Die Funktionsweise und das Programmiermodell wird im Abschnitt 5.4.7 erläutert.

[6] Achtung auf die Jumper *J1 und J2*!

Die Taktgenerierung könnte mit einem freien Timer der MVME162-512A realisiert werden. Dieser kann z.B. auf 5μs programmiert werden und Interrupts auslösen. Wenn man jeden zweiten Interrupt verwendet, um die Leitungen `CS`, `CK` und `DI` gemäß dem Zeitdiagramm zu setzen, so kann bei den anderen Interrupts das Signal `DO` ausgelesen werden. Wichtig ist, daß das Signal `CK` nicht schneller als mit 5μs den Pegel wechselt.

B.7 Testsoftware

Im Verzeichnis `$SynUTC/TECH/Hardware/NTI/NTI_Version_1/Testprog/` ist der Code für das Testprogramm abgelegt. Das Testprogramm prüft die wichtigsten UTCSU-Funktionen, wie z.B. den Speicher mit dem March-B Test, einfache UTCSU-Zugriffe, das Auslösen von Interrupts, die richtige Funktion der Trigger-Signale, etc.

Anhang C

Dateien für den NTI-Handler

C.1 a203.h

```
/* a203.h */
/*************************************************************************
 *
 * Register-Definitionen fr A203 VME M-Modul carrier board
 *
 * Thomas MANDL
 *
 * $Log: a203.h,v $
 * Revision 2.0  1999/01/20 12:55:40  s
 * Register-Definitions MEN A203
 *
 *
 * 5.3.1997 Thomas MANDL
 *   Erste Version fertiggestellt.
 *
 ************************************************************************/

#ifndef _A203_H
#define _A203_H

#define A203_SLOT_0 0
#define A203_SLOT_1 1
#define A203_SLOT_2 2
#define A203_SLOT_3 3

/* defines for Interrupts */
#define A203_IPEND (1 << 7)    /* Interrupt is pending */
#define A203_IEXT  (1 << 5)    /* Vector is generated by M-Module */
#define A203_IRE   (1 << 4)    /* Interrupt enable */
#define A203_IRAC  (1 << 3)    /* Interrupt auto clear */

typedef volatile struct /* A203_SLOT */
{
  INT16 io_space[0x80];        /* M-Module I/O-space (256 Bytes) */
  INT16 micr;                  /* M-Module Interrupt Control Register */
  INT16 mivr;                  /* M-Module Interrupt Vector Register */
  INT16 unused1;
  INT16 divr;                  /* DMA Interrupt Vector Register */
  INT16 unused2[0x3C];
  INT16 tdar;                  /* Trigger Data Register */
  INT16 tdir;                  /* Trigger Direction Register */
  INT16 unused3[0x3F];
  INT16 dmacr;                 /* DMA Control Register */
  INT16 unused4[0x3E];
  INT16 dma_unused[0x0E];      /* DMA Registers (NOT USED) */
  INT16 unused5[0x32];
  INT16 eeprom;                /* EEPROM-Register */
  INT16 sacr;                  /* A32 slave address compare register */
  INT16 unused6[0x3E];
  INT16 jtag;                  /* JTAG Interface */
  INT16 unused7[0x3F];
} A203_SLOT;

typedef volatile struct
{
  A203_SLOT slot[4];
} A203;

#endif
```

C.2 ci_wrapper.c

```
/* ci_wrapper.c */
/**********************************************************************
 *
 * NTI-Handling fr i82596 NTI-Driver
 *
 *
 * $Log: ci_wrapper.c,v $
 * Revision 2.0  1999/01/20 13:02:22  s
 * First RCS revision
 *
 * 20.1.99 U. Schmid
 *   "Poll" pending INTNs before INTA, INTT and GPU-ISR to minimize
 *   blocking of receive timestamp moving
 *
 * 12.10.98 U. Schmid
 *   Use pcb_install's functions for enable/disable UTCSU-Interrupts
 *   Call GPU-ISRs
 *
 * 14.9.1998 U. Schmid
 *   Erste Version fertiggestellt
 *
 *---------------------------------------------------------------------
 *
 * Dieses File beinhaltet das NTI Setup und Interrupt-Handling
 * f"ur den NTI Device-Drivers f"ur den i82596 Ethernet-Controller.
 * Unterst"utzt werden dabei max. 4 NTIs (= "Areas") auf dem
 * Extender-Board der AcQ i6360 (Slot #2 .. #5; #0 und #1 werden
 * wohl initialisiert, aber nicht vom NTI-Driver verwendet) oder der
 * MEN A203 (Slot #0 .. #3). Funktionsumfang:
 *
 * 1. Installation NTIs
 *
 * 2. INTN-Handling:
 *    - Receive-Timestamps aller Areas in Receive-Header kopieren
 *    - GPU-Interrupts weiterleiten an Anwendungs-Funktion
 *        void gps_isr(short slot, short gpu, UTCSU *utcsu)
 *
 * 3. Weiterleiten INTA und INTT an Anwendungs-Funktionen
 *       void ci_inta(NTI_CONFIG *nti);
 *       void ci_intt(NTI_CONFIG *nti);
 *
 * ACHTUNG Anwender: Die externen ISRs (ci_inta, ci_intt und gps_isr)
 * m"ussen m"oglichst kurz sein, da sie das Receive-Timestamp-Move aufhalten.
 * Mehr als etwa 50 us sind da kaum drin!
 *
 **********************************************************************/

#include <types.h>
#include "pcb_conf.h"
#include "/ABT/PROJEKTE/SynUTC/TECH/Software/NTI-Handler/pcb_install.h"

/*------------------------------------------------------------------------
 * Externals
 *------------------------------------------------------------------------*/

extern void ci_inta(NTI_CONFIG *nti); /* Anwendungs-ISR UTCSU-INTA */
extern void ci_intt(NTI_CONFIG *nti); /* Anwendungs-ISR UTCSU-INTT */
extern void gps_isr(short slot, short gpu, UTCSU *utcsu); /* Anwendungs-ISR GPU */

/*------------------------------------------------------------------------
 * Defines
 *------------------------------------------------------------------------*/

#if NTI_PCB == MVME_I6360
#define FIRSTNTI 2    /* Area 0 == M-Module Slot #2 */
#else
#define FIRSTNTI 0    /* All other carriers: Area 0 == Slot #0 */
#endif

#define GPU_NUM       3    /* Max. Anzahl aktiver GPUs */
#define FIRSTVECTOR   0x90 /* Vector-Base M-Module FIRSTNTI; others consecutive */
#define UTCSU_STEP    0x0D6BF94D /* Step-Value f"ur UTCSU Clock 10 MHz */
#define UTCSU_STEPLOW 0x5E000000 /* StepLow-Value f"ur UTCSU Clock 10 MHz */
#define UTCSU_CONF1   0          /* Initial setup UTCCONF1-register */
#define UTCSU_CONF2   ((1<<26)|(1<<20))  /* Initial setup UTCCONF2-register */
                                         /* rising edge Rcv & Trans.-TS */

/*------------------------------------------------------------------------
 * Globals
 *------------------------------------------------------------------------*/

NTI_CONFIG ntidir[MAX_NTI];  /* Directory der NTIs auf Carrier-Board */
```

```
/*---------------------------------------------------------------------------
 * Function nti_init(.) --- Install and setup M-Modules f"ur NTI-Driver
 * F"ullt einen Parameter INT32 da[4] mit den Anfangsadressen des Mem-Spaces
 * der NTIs (oder 0 wenn nicht best"uckt), die f"ur die Areas zust"andig sind.
 * Return-Wert ist die Anzahl der erfolgreich installierten Module.
 * Note: Es werden grunds"atzlich alle best"uckten NTIs initialisiert und
 * in ntidir eingetragen.
 *---------------------------------------------------------------------------*/

int nti_init(NTI *na[4])
{
  short slot, count;
  INT32 i32_temp;
  NTI_IO_SPACE *nti_io;
  UTCSU *utcsu;
  XMT_HEAD_32 xmt;
  RCV_HEAD_32 rcv;

  /* Initialize NTI-directory */
  for (slot=0;slot<MAX_NTI;slot++)
  {
    ntidir[slot].installed = 1;   /* We want it if present and not excluded */
    ntidir[slot].nti_irq_vector = FIRSTVECTOR + 8*(slot - FIRSTNTI);
    /* Offset Rec. und Trans. TS-Trigger setzen */
    ntidir[slot].nti_ts_offsets = ((INT32)&rcv.ttsrcv - (INT32)&rcv)|(((INT32)&xmt.ttsxmt - (INT32)&xmt)<<8);
    ntidir[slot].nti_type_offset = TS_COMP_ON | ((INT32)&rcv.ttsrcv - (INT32)&rcv);  /* Type-Field Offset */
    ntidir[slot].utcsu_step = UTCSU_STEP; /* Clock Speed UTCSU */
    ntidir[slot].utcsu_steplow = UTCSU_STEPLOW;
    ntidir[slot].utcsu_conf1 = UTCSU_CONF1;
    ntidir[slot].utcsu_conf2 = UTCSU_CONF2;
  }

  count = pcb_install(ntidir); /* Installieren NTIs */

  /* Enable UTCSU-RCV Interrupts ONLY for the NTI Area 0 */
  utcsu_int_enable2(FIRSTNTI, 1 << 8);

  /* Return pointers to NTI memory space in na[] */
  for (slot=0;slot<4;slot++)
    if (ntidir[slot+FIRSTNTI].installed)
      na[slot] = ntidir[slot+FIRSTNTI].memory;
    else
      na[slot]=(NTI *)0; /* Modul nicht eingesteckt (oder exkludiert) */

  return count;
}



/*---------------------------------------------------------------------------
 * Function isr_n(.) --- Interrupt-Service UTCSU-INTN
 *---------------------------------------------------------------------------*/

void isr_n(NTI_CONFIG *nti)
{
short slot;
short gpu=0;
UTCSU *utcsu = nti->utcsu;  /* Damit der Zugriff schneller geht */
NTI_IO_SPACE *io = nti->io;

  do
  {
    /* First check RCV-Timestamp interrupt (ONLY FOR AREA 0!) */
    if ((utcsu->r32.utcintstat2 & nti->readback_utcinten2 & (1 << 8)) != 0)
    {
      /* Locate the guilty CSP ... */
      RCV_HEAD_32 *csp = (RCV_HEAD_32 *)((INT32)&(nti->memory->mem.cpu) + (io->rcv_header_base << 5));

      utcsu->w32.utcintclear2 = 1 << 8;   /* Clear this Interrupt */

      /* Now move the timestamps into the CSP (at the end of rcv header) */
      for (slot=FIRSTNTI;slot<FIRSTNTI+4;slot++)   /* Go over all NTIs */
      {
        if (ntidir[slot].installed)   /* M-Module is present */
        {
          csp->user_data[2*(slot-FIRSTNTI)] = (ntidir[slot].utcsu->r32.ssu[0].msrcv & 0x00FFFFFF); /* Macrostamp */
          csp->user_data[2*(slot-FIRSTNTI)+1] = ntidir[slot].utcsu->r32.ssu[0].tsrcv; /* Timestamp */
        }
      }
    } /* if(INTT SSU1) */

    /* Hier k"onnen jetzt noch die ISR-Aktivit"aten von anderen Dingen */
    /* wie GPUs 1-3 und/oder anderen SSUs behandelt werden. */

    if ((utcsu->r32.utcintstat2 & (1 << gpu)) != 0) gps_isr(nti->slot, gpu+1, utcsu);

  } while (++gpu < GPU_NUM);
}
```

```
/*------------------------------------------------------------------------
 * NTI Interrupt Service Routines required by pcb_install_<PCB>.c
 *------------------------------------------------------------------------*/

/* Note that only the FIRSTNTI (Area 0) causes RCV-interrupts that trigger */
/* moving the receive timestamps of ALL areas into the CSP. However, */
/* there could be other reasons for INTN, so it is routed to our isr_n(.) */
/* for all Slots */

void isr_nti0_n()
{
  isr_n(&ntidir[0]);
}

void isr_nti0_t()
{
  while (ntidir[0].io->vector_base & 0x0004) isr_n(&ntidir[0]);
  ci_intt(&ntidir[0]);
}

void isr_nti0_a()
{
  while (ntidir[0].io->vector_base & 0x0004) isr_n(&ntidir[0]);
  ci_inta(&ntidir[0]);
}


void isr_nti1_n()
{
  isr_n(&ntidir[1]);
}

void isr_nti1_t()
{
  while (ntidir[1].io->vector_base & 0x0004) isr_n(&ntidir[1]);
  ci_intt(&ntidir[1]);
}

void isr_nti1_a()
{
  while (ntidir[1].io->vector_base & 0x0004) isr_n(&ntidir[1]);
  ci_inta(&ntidir[1]);
}


void isr_nti2_n()
{
  isr_n(&ntidir[2]);
}

void isr_nti2_t()
{
  while (ntidir[2].io->vector_base & 0x0004) isr_n(&ntidir[2]);
  ci_intt(&ntidir[2]);
}

void isr_nti2_a()
{
  while (ntidir[2].io->vector_base & 0x0004) isr_n(&ntidir[2]);
  ci_inta(&ntidir[2]);
}


void isr_nti3_n()
{
  isr_n(&ntidir[3]);
}

void isr_nti3_t()
{
  while (ntidir[3].io->vector_base & 0x0004) isr_n(&ntidir[3]);
  ci_intt(&ntidir[3]);
}

void isr_nti3_a()
{
  while (ntidir[3].io->vector_base & 0x0004) isr_n(&ntidir[3]);
  ci_inta(&ntidir[3]);
}

void isr_nti4_n()
{
  isr_n(&ntidir[4]);
}

void isr_nti4_t()
```

```
{
  while (ntidir[4].io->vector_base & 0x0004) isr_n(&ntidir[4]);
  ci_intt(&ntidir[4]);
}

void isr_nti4_a()
{
  while (ntidir[4].io->vector_base & 0x0004) isr_n(&ntidir[4]);
  ci_inta(&ntidir[4]);
}

void isr_nti5_n()
{
  isr_n(&ntidir[5]);
}

void isr_nti5_t()
{
  while (ntidir[5].io->vector_base & 0x0004) isr_n(&ntidir[5]);
  ci_intt(&ntidir[5]);
}

void isr_nti5_a()
{
  while (ntidir[5].io->vector_base & 0x0004) isr_n(&ntidir[5]);
  ci_inta(&ntidir[5]);
}
```

C.3 nti.h

```
/* nti.h */
/************************************************************************
 *
 * pSOS+ Typdefinitionen f"ur die NTI
 *
 * NTI Typdefinitionen f"ur IO- und Memory-Space (inklusive Transmit-
 * und Receive-Header f"ur den jeweiligen COMCO) f"ur alle existierenden
 * NTI-Varianten.
 *
 * Thomas MANDL
 *
 * $Log: nti.h,v $
 * Revision 2.0  1999/01/20 13:04:52  s
 * First RCS revision
 *
 * 6.11.98 U. Schmid
 *   i82596 COMCO auf 128 Bytes TX-Header umgestellt; alte Version als
 *   i82596_TX64 renamed.
 *
 * 11.9.1998 U. Schmid
 *   Komplette Restrukturierung Header-Files, Anpassung an aktuelle NTI
 *
 * 27.3.1998 B. Weiss
 *      Adaption f"ur neues NTI Modul
 *
 * 5.3.1997 Thomas MANDL
 *   Erste Version fertiggestellt.
 *
 * 22.7.1997 Thomas MANDL
 *   Erg"anzungen f"ur das NTI-Debug Board.
 *----------------------------------------------------------------------
 *
 * ANWENDER: Das File erwartet die bereits gesetzten #defines
 *
 *    NTI_COMCO    gibt den COMCO an:
 *                 i82596  ...  Intel i82596 Ethernet-Controller
 *
 *    NTI_VERSION  mu"s eine g"ultige NTI-Versionsnummer sein:
 *                 NORMAL512K  ...  Normalversion mit 512k RAM
 *                 NORMAL256K  ...  Normalversion mit 256k RAM
 *
 * Die existierenden Typen/Versionen sind in nti_versions.h zu finden;
 * die obigen #defines sollten in pbc_conf.h (wo die Features
 * der Carrierboard/CPU-Konfiguration deklariert werden) gesetzt werden;
 * pcb_conf.h inkludiert dazu automatisch nti.h.
 *
 * ENTWICKLER: Bei Modifikationen oder Hinzuf"ugen neuer Versionen beachten,
 * da"s die typedefs exakt mit den im CPLD der NTI festgelegten Memory-Maps
 * "ubereinstimmen m"ussen. Dies betrifft insbesondere die typedefs
 * f"ur 32-/16-/8-Bit Zugriffe auf die Transmit- und Receive-Header f"ur
 * den jeweiligen COMCO und das Layout der verschiedenen Memory-Regions.
 *
 * Sollten neue NTI-Varianten dazukommen, bitte auf Eintrag in
 * nti_versions.h nicht vergessen!
 *
 ************************************************************************/

#ifndef _NTI_H
#define _NTI_H

#include "/ABT/PROJEKTE/SynUTC/TECH/Software/NTI-Handler/utcsu.h"


/***************************************************************************
 * NTI Memory-Space Typdefinitionen
 *
 * Der NTI Memory-Space enth"alt zwei double-mapped Memory-Windows
 * f"ur COMCO- bzw. CPU-Zugriffe auf das NTI RAM und die UTCSU-Register
 ***************************************************************************/

#if NTI_COMCO == i82596
/***************************************************************************
 ***************************************************************************
 * NTIs f"ur Intel i82596 Ethernet-Controller COMCO  (128 Bytes TX-Header)
 ***************************************************************************
 ***************************************************************************/

#define XMT_HEADER_SIZE     0x80    /* 128 Byte Transmit Headers */
#define RCV_HEADER_SIZE     0x40    /* 64 Byte Receive Headers */

/+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * Typedefs f"ur 32/16/8-Bit Zugriff auf XMT_HEADER bzw. RCV_HEADER
 * Sie m"ussen alle XMT_HEADER_SIZE bzw. RCV_HEADER_SIZE haben!
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

```

```
typedef volatile struct
{
  INT32 addr_data[25];
  INT32 ttsxmt;          /* Transmit-Timestamp Trigger */
  INT32 user_data1[3];
  INT32 ms;              /* Macrostamp */
  INT32 ts;              /* Timestamp */
  INT32 as;              /* Accuracystamp */
} XMT_HEAD_32;


typedef volatile struct
{
  INT32 addr_data[7];  /* 0x00 - 0x18 */
  INT32 ttsrcv;        /* Receive-Timestamp Trigger (Offset 0x1C) */
  INT32 user_data[8];  /* => 64 Byte Header-Size */
} RCV_HEAD_32;

typedef volatile struct
{
  INT16 addr_data[50];
  INT16 ttsxmt[2];          /* Transmit-Timestamp Trigger */
  INT16 user_data1[6];
  INT16 ms[2];              /* Macrostamp */
  INT16 ts[2];              /* Timestamp */
  INT16 as[2];              /* Accuracystamp */
} XMT_HEAD_16;

typedef volatile struct
{
  INT16 addr_data[14];    /* 0x00 - 0x18 */
  INT16 ttsrcv[2];        /* Receive-Timestamp Trigger (Offset 0x1C) */
  INT16 user_data[16];    /* => 64 Byte Header-Size */
} RCV_HEAD_16;

typedef volatile struct
{
  INT8 addr_data[100];
  INT8 ttsxmt[4];         /* Transmit-Timestamp Trigger */
  INT32 user_data1[12];
  INT8 ms[4];             /* Macrostamp */
  INT8 ts[4];             /* Timestamp */
  INT8 as[4];             /* Accuracystamp */
} XMT_HEAD_8;

typedef volatile struct
{
  INT8 addr_data[28];     /* 0x00 - 0x18 */
  INT8 ttsrcv[4];         /* Receive-Timestamp Trigger (Offset 0x1C,0x1D) */
  INT8 user_data[32];     /* => 64 Byte Header-Size */
} RCV_HEAD_8;


/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * NTI - Memory-Layout
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

#if NTI_VERSION == NORMAL512K     /* NTI mit 512KB Memory */

#define NTI_MEMORY_WINDOW 0x80000 /* Gr"o"se CPU / COMCO Memory-Fenster */

#define XMT_HEADER_REGION  0x2000 /* 8KB Trans.Header (X * XMT_HEADER_SIZE) */
#define RCV_HEADER_REGION 0x1E000 /* 120KB Rcv.Header (X * RCV_HEADER_SIZE) */
#define BUFFER_REGION     0x5C000 /* 376KB Buffers (X * 4) */
#define SYS_STRUCT_REGION  0x3E00 /* 16-0.5KB System Structures (X * 4) */
#define UTCSU_GAP               0 /* Gap zwischen Ende (CPU-)RAM und UTCSU */

#else                             /* NTI mit 256KB Memory */

#define NTI_MEMORY_WINDOW 0x80000 /* Gr"o"se CPU / COMCO Memory-Fenster */

#define XMT_HEADER_REGION  0x1000 /* 4KB Trans.Header (X * XMT_HEADER_SIZE) */
#define RCV_HEADER_REGION  0xF000 /* 60KB Rcv.Header (X * RCV_HEADER_SIZE) */
#define BUFFER_REGION     0x2E000 /* 184KB Buffers (X * 4) */
#define SYS_STRUCT_REGION  0x2000 /* 8KB System Structures (X * 4) */
#define UTCSU_GAP   0x40000-0x200 /* Gap zwischen Ende (CPU-)RAM und UTCSU */

#endif


#else
#if NTI_COMCO == i82596_TX64

/******************************************************************************
 ******************************************************************************
 * NTIs f"ur Intel i82596 Ethernet-Controller COMCO (64 Bytes TX-Header)
 ******************************************************************************
 ******************************************************************************/
```

```
#define XMT_HEADER_SIZE     0x40    /* 64 Byte Transmit Headers */
#define RCV_HEADER_SIZE     0x40    /* 64 Byte Receive Headers */

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * Typedefs f"ur 32/16/8-Bit Zugriff auf XMT_HEADER bzw. RCV_HEADER
 * Sie m"ussen alle XMT_HEADER_SIZE bzw. RCV_HEADER_SIZE haben!
 *+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

typedef volatile struct
{
  INT32 addr_data[5];  /* 0x00 - 0x10 */
  INT32 ttsxmt;        /* Transmit-Timestamp Trigger (Offset 0x14) */
  INT32 ms;            /* Macrostamp */
  INT32 ts;            /* Timestamp */
  INT32 as;            /* Accuracystamp */
  INT32 user_data[7];  /* => 64 Byte Header-Size */
} XMT_HEAD_32;

typedef volatile struct
{
  INT32 addr_data[7];  /* 0x00 - 0x18 */
  INT32 ttsrcv;        /* Receive-Timestamp Trigger (Offset 0x1C) */
  INT32 user_data[8];  /* => 64 Byte Header-Size */
} RCV_HEAD_32;

typedef volatile struct
{
  INT16 addr_data[10];    /* 0x00 - 0x10 */
  INT16 ttsxmt[2];        /* Transmit-Timestamp Trigger (Offset 0x14) */
  INT16 ms[2];            /* Macrostamp */
  INT16 ts[2];            /* Timestamp */
  INT16 as[2];            /* Accuracystamp */
  INT16 user_data[14];    /* => 64 Byte Header-Size */
} XMT_HEAD_16;

typedef volatile struct
{
  INT16 addr_data[14];    /* 0x00 - 0x18 */
  INT16 ttsrcv[2];        /* Receive-Timestamp Trigger (Offset 0x1C) */
  INT16 user_data[16];    /* => 64 Byte Header-Size */
} RCV_HEAD_16;

typedef volatile struct
{
  INT8 addr_data[20];     /* 0x00 - 0x10 */
  INT8 ttsxmt[4];         /* Transmit-Timestamp Trigger (Offset 0x14,0x15!) */
  INT8 ms[4];             /* Macrostamp */
  INT8 ts[4];             /* Timestamp */
  INT8 as[4];             /* Accuracystamp */
  INT8 user_data[28];     /* => 64 Byte Header-Size */
} XMT_HEAD_8;

typedef volatile struct
{
  INT8 addr_data[28];     /* 0x00 - 0x18 */
  INT8 ttsrcv[4];         /* Receive-Timestamp Trigger (Offset 0x1C,0x1D) */
  INT8 user_data[32];     /* => 64 Byte Header-Size */
} RCV_HEAD_8;


/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * NTI - Memory-Layout
 *+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

#if NTI_VERSION == NORMAL512K     /* NTI mit 512KB Memory */

#define NTI_MEMORY_WINDOW 0x80000 /* Gr"o"se CPU / COMCO Memory-Fenster */

#define XMT_HEADER_REGION  0x2000 /* 8KB Trans.Header (X * XMT_HEADER_SIZE) */
#define RCV_HEADER_REGION 0x1E000 /* 120KB Rcv.Header (X * RCV_HEADER_SIZE) */
#define BUFFER_REGION     0x5C000 /* 376KB Buffers (X * 4) */
#define SYS_STRUCT_REGION  0x3E00 /* 16-0.5KB System Structures (X * 4) */
#define UTCSU_GAP               0 /* Gap zwischen Ende (CPU-)RAM und UTCSU */

#else                             /* NTI mit 256KB Memory */

#define NTI_MEMORY_WINDOW 0x80000 /* Gr"o"se CPU / COMCO Memory-Fenster */

#define XMT_HEADER_REGION  0x1000 /* 4KB Trans.Header (X * XMT_HEADER_SIZE) */
#define RCV_HEADER_REGION  0xF000 /* 60KB Rcv.Header (X * RCV_HEADER_SIZE) */
#define BUFFER_REGION     0x2E000 /* 184KB Buffers (X * 4) */
#define SYS_STRUCT_REGION  0x2000 /* 8KB System Structures (X * 4) */
#define UTCSU_GAP   0x40000-0x200 /* Gap zwischen Ende (CPU-)RAM und UTCSU */

#endif

#else
```

```
!!!!! ERROR --- Illegal COMCO-Type (Check pcb_conf.h) !!!!!!
#endif
#endif


/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * NTI - Memory-Space: COMCO-Accesses
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

#define UNUSED_REGION_COMCO   NTI_MEMORY_WINDOW - (XMT_HEADER_REGION + RCV_HEADER_REGION +
                                                  BUFFER_REGION + SYS_STRUCT_REGION)

typedef volatile union
{
  volatile struct /* 8-Bit Zugriffe */
  {
    XMT_HEAD_8 tx_head[(XMT_HEADER_REGION) / sizeof(XMT_HEAD_8)];
    RCV_HEAD_8 rcv_head[(RCV_HEADER_REGION) / sizeof(RCV_HEAD_8)];
    INT8 buffers[BUFFER_REGION];
    INT8 sys_struct[SYS_STRUCT_REGION];
    #if UNUSED_REGION_COMCO != 0
    INT8 unused[UNUSED_REGION_COMCO];
    #endif
  } rw8;

  volatile struct /* 16-Bit Zugriffe */
  {
    XMT_HEAD_16 tx_head[(XMT_HEADER_REGION) / sizeof(XMT_HEAD_16)];
    RCV_HEAD_16 rcv_head[(RCV_HEADER_REGION) / sizeof(RCV_HEAD_16)];
    INT16 buffers[(BUFFER_REGION)/2];
    INT16 sys_struct[(SYS_STRUCT_REGION)/2];
    #if UNUSED_REGION_COMCO != 0
    INT16 unused[(UNUSED_REGION_COMCO)/2];
    #endif
  } rw16;

  volatile struct /* 32-Bit Zugriffe */
  {
    XMT_HEAD_32 tx_head[(XMT_HEADER_REGION) / sizeof(XMT_HEAD_32)];
    RCV_HEAD_32 rcv_head[(RCV_HEADER_REGION) / sizeof(RCV_HEAD_32)];
    INT32 buffers[(BUFFER_REGION)/4];
    INT32 sys_struct[(SYS_STRUCT_REGION)/4];
    #if UNUSED_REGION_COMCO != 0
    INT32 unused[(UNUSED_REGION_COMCO)/4];
    #endif
  } rw32;
} NTI_COMCO2MEM;


/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * NTI - Memory-Space: CPU-Accesses
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

#define NTI_MEMORY_CPU   XMT_HEADER_REGION + RCV_HEADER_REGION + BUFFER_REGION + SYS_STRUCT_REGION + UTCSU_GAP

typedef volatile union
{
  INT8  rw8[NTI_MEMORY_CPU];
  INT16 rw16[(NTI_MEMORY_CPU)/2];
  INT32 rw32[(NTI_MEMORY_CPU)/4];
} NTI_CPU2MEM;


/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * NTI - Memory Space: Gesamte Typdefinition
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

typedef volatile struct
{
  NTI_COMCO2MEM comco; /* double-mapped RAM f"ur COMCO-Accesses */
  NTI_CPU2MEM   cpu;   /* double-mapped RAM f"ur CPU-Accesses */
} NTI_MEMORY;


typedef volatile struct
{
  NTI_MEMORY mem;     /*  double mapped RAM */
  UTCSU      utcsu;
} NTI;



/****************************************************************************
 * NTI IO-Space Typdefinitionen
 *
 * "Uber den NTI IO-Space k"onnen die NTI-Register wie z.B.
 * Receive-Vector Base, Interrupt Vector Base, Int-Logik und S-PROM
 * accessed werden.
```

```
 ****************************************************************************/

/****************************************************************************
 ****************************************************************************
 * Derzeit haben alle NTI-Varianten denselben IO-Space.
 *
 * Um die Verwendbarkeit der Autoinstallations-Funktion pcb_init()
 * garantieren zu k"onnen, sollte das untenstehende Layout (bis inklusive
 * type_offset) auch bei eventuellen NTI-Varianten eingehalten werden.
 ****************************************************************************
 ****************************************************************************/

/* Typedef f"ur 16-bit Accesses */

typedef volatile struct
{
  INT16 unused1;
  INT16 rcv_header_base;    /* Adresse des zuletzt empfangenen Datenpakets */
  INT16 unused2;
  INT16 vector_base;        /* Interrupt-Vector Base */
  INT16 unused3;
  INT16 int_logic;          /* Enable/Disable NTI-Interrupts */
  INT16 unused4;
  INT16 la_trigger;         /* Trigger Signal f"ur Logic Analyzer */
  INT16 unused5;
  INT16 utcsu_misc;         /* Setting statischer UTCSU-Signale */
  INT16 unused6;
  INT16 ts_offsets;         /* Trigger-Offsets fr Transmit/Receive-TS  */
                            /*                         (upper/lower byte) */
  INT16 unused7;
  INT16 type_offset;        /* Offset Type-Field in Rec.-Header (lower byte) */
  INT16 unused8[0x80-14-1];
  INT16 s_prom;             /* ID Serial PROM */
} NTI_IO_SPACE16;


 /* Typedef f"ur 32-bit Accesses */

typedef volatile struct
{
  INT32 rcv_header_base;    /* Adresse des zuletzt empfangenen Datenpakets */
  INT32 vector_base;        /* Interrupt-Vector Base */
  INT32 int_logic;          /* Enable/Disable NTI-Interrupts */
  INT32 la_trigger;         /* Trigger Signal f"ur Logic Analyzer */
  INT32 utcsu_misc;         /* Setting statischer UTCSU-Signale */
  INT32 ts_offsets;         /* Trigger-Offsets fr Transmit/Receive-TS  */
                            /*                         (upper/lower byte) */
  INT32 type_offset;        /* Enable + Offset Type-Field in Rec.-Header */
  INT32 unused[0x40-8];
  INT32 s_prom;             /* ID Serial PROM */
} NTI_IO_SPACE32;


/* Default Typedef */

typedef NTI_IO_SPACE16   NTI_IO_SPACE;


/* Defines f"ur NTI-Register Setting */

/* NTI-Register int_logic */
#define NTI_DIS_INT_LOGIC    0
#define NTI_EN_INT_LOGIC     1
#define NTI_INT_LOGIC_ON     (1 << 15)   /* F"ur Status-Abfrage NTI Int.Logik */

/* NTI-Register utcsu_misc               Bedeutung gesetztes Bit: */
#define NTI_MISC_BIGENDIAN    1          /* UTCSU big-endian */
#define NTI_MISC_BUS8         (1 << 1)   /* UTCSU 8-bit Port-Size */
#define NTI_MISC_BUS16        (1 << 2)   /* UTCSU 16-bit Port-Size */
#define NTI_MISC_TESTMUX      (1 << 3)   /* Export Test-Signale am NTPA-Bus */
#define NTI_MISC_DIRECTIN     (1 << 4)   /* Ausschalten Synchronizer */
#define NTI_MISC_RELIABLE     (1 << 5)   /* 2-stufiger Synchronizer */
#define NTI_MISC_RELH         (1 << 6)   /* 2-stufiger Synchronizer CPU */
#define NTI_MISC_WAIT         (1 << 7)   /* 1 Waitstate-Operation */
#define NTI_MISC_RESET        (1 << 15)  /* UTCSU Reset */

/* NTI-Register type_offset */
#define TS_COMP_ON            0x8000     /* Enable Typefield Comparison Rcv.TS */


#endif
```

C.4 nti_version.h

```
/* nti_versions.h */
/************************************************************************
 *
 * Defines f"ur die verschienden NTI-Versionen (COMCO und Versionsnummer)
 * und Carrier-Board/CPU-Kombinationen ("PCB's")
 *
 * U. Schmid
 *
 * $Log: nti_versions.h,v $
 * Revision 2.0  1999/01/20 13:06:31  s
 * First RCS revision
 *
 * 6.11.1998 U. Schmid
 *   i82596 mit 128 Bytes TX-Header; alte Version als i82596_TX64 renamed
 *
 * 13.9.1998 U. Schmid
 *   Erste Version fertig
 *-----------------------------------------------------------------------
 *
 * Jede Version eines NTI M-Moduls wird durch den COMCO und eine Versions-
 * nummer eindeutig bezeichnet; letzere dient zur Unterscheidung unter-
 * schiedlicher Modul-Konfigurationen z.B. in punkto Memory.
 * Die weiteren *.h-Files f"ur die NTI erwarten zwei entsprechende #defines
 *    NTI_COMCO
 *    NTI_VERSION
 * deren m"ogliche Settings in diesem File nti_versions.h zu finden sind.
 *
 * Weiters mu"s selektiert werden, auf welcher Carrier-Board/CPU-Kombination
 * die NTIs verwendet werden sollen. Dies wird durch ein #define
 *    NTI_PCB
 * bekanntgegeben, dessen m"ogliche Settings ebenfalls hier zu finden sind.
 *
 * ENTWICKLER: Sollten neue NTI-Versionen dazukommen, bitte auf ev.
 * Modifikation von nti.h nicht vergessen!
 *
 ************************************************************************/
#ifndef _NTI_VERSIONS_H
#define _NTI_VERSIONS_H


/* Settings f"ur NTI_COMCO */

#define i82596_TX64 8259664 /* Intel i82596 (MVME-162) mit 64 Bytes TX-Header */
#define i82596      82596   /* Intel i82596 (MVME-162) mit 128 Bytes TX-Header */
#define M68EN360    68360   /* Motorola QUICC (AcQ i6360) */


/* Settings f"ur NTI_VERSION */

#define NORMAL256K 1       /* Normalversion NTI mit 256k RAM */
#define NORMAL512K 2       /* Normalversion NTI mit 512k RAM */


/* Settings f"ur NTI_PCB */

#define MVME_I6360 1      /* MVME-162 CPU mit AcQ i6360 VME-Carrier */
#define MVME_A203  2      /* MVME-162 CPU mit MEN A203 VME-Carrier */
#endif
```

C.5 pcb_conf_a203.h

```
/* pcb_conf.h */
/************************************************************************
 *
 * Configuration-File f"ur Setup NTI Carrier-Boards  (f"ur A203)
 *
 * Thomas MANDL
 *
 * $Log: pcb_conf_a203.h,v $
 * Revision 2.0  1999/01/20 13:09:52  s
 * First RCS revision
 *
 * 12.9.1998 U. Schmid
 *   Restrukturierung aller Header-Files
 *
 * 5.3.1997 Thomas MANDL
 *   Erste Version fertiggestellt.
 *
 *-----------------------------------------------------------------------
 *
 * Dieses Header-File ist das NTI-Analogon zum zentralen Konfigurations-
 * File sys_conf.h des pSOS-Systems. Die hier enthaltenen #defines,
 * die vom Anwender festzulegen sind, werden von den NTI-spezifischen
 * Header- und Programm-Files verarbeitet. Entsprechend der Tatsache, da"s es
 *
 * (1) f"ur jeden COMCO eine eigene NTI (eventuell in mehreren verschiedenen
 *     Versionen)
 * (2) mehrere verschieden anzusprechende M-Modul Carrier-Boards (m"oglicher-
 *     weise von mehreren verschiedenen CPUs aus verwendet)
 *
 * gibt, wurde folgende File-Struktur gew"ahlt:
 *
 * pcb_conf.h           (Dieses File)
 * nti_versions.h       Existierende NTI-Versionen
 * nti.h                Typdefinitionen f"ur die existierenden NTI-Varianten
 * pcb_install.h        Typdefinitionen f"ur die Autoinstall-Funktion
 * pcb_install_<PCB>.c  Autoinstall-Funktion f"ur NTI auf Carrier+CPU <PCB>
 *
 * ANWENDER: Man beachte, da"s pcb_conf.h auch automatisch nti.h (und
 * somit auch utcsu.h) inkludiert.
 *
 ************************************************************************/

#ifndef _PCB_CONF_H
#define _PCB_CONF_H

#include "/ABT/PROJEKTE/SynUTC/TECH/Software/NTI-Handler/nti_versions.h"

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * Definition der verwendeten NTI-Version
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/* COMCO-Typ (siehe nti_versions.h) */
#define NTI_COMCO i82596   /* Intel i82596 Ethernet-Controller (MVME-162) */

/* NTI Versionsnummer (siehe nti_versions.h) */
#define NTI_VERSION NORMAL256K          /* Normalversion NTI mit 256k RAM */


/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * Einbinden der Typdefinitionen + #defines der gew"ahlten NTI
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
#include "/opt/isi/apps/SynUTC/nti/nti.h"


/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * Definition der verwendeten PCB-Variante und Festlegung der Konfigurations-
 * parameter
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/* Einstellung des gew"unschten Carrier-Boards/CPU (siehe nti_versions.h) */
#define NTI_PCB    MVME_A203   /* MVME-162 CPU mit MEN A203 VME-Carrier */


/* Auswahl der M-Module, bei denen (falls montiert) die ISR's in den */
/* Vektoren installiert werden sollen (Grundinstallation und Rest des */
/* Interrupt-Handlings wird immer aufgesetzt (1 == ISRs installieren). */

#define ISR_NTI0_INST         1
#define ISR_NTI1_INST         1
#define ISR_NTI2_INST         1
#define ISR_NTI3_INST         1


/* Auswahl "VME-Page" (0xE0, 0xE4, 0xE8, 0xEC), innerhalb der der
 * Memory-Space aller M-Module des Carrier-Boards adressierbar ist (z.B.
```

```
 * VME_PAGE==0xE0 f"ur den Bereich 0xE0000000 .. 0xE3FFFFFF)
 *            ACHTUNG: Die A203 verbraucht de facto einen ganzen 256MB
 *                     Block, in dem die M-Module 4 Mal sichtbar sind!
 *                     Daher ist 0xE0, 0xE4, 0xE8 und 0xEC in Wirklichkeit
 *                     gleichbedeutend!!! */

#define VME_PAGE         0xE8

/* MVME_SHORT : Startadresse des IO-Spaces M-Modul 0, die in dem
 * Bereich liegen muss, in dem die MVME Short/D16
 * Accesses am VMEbus macht. Der IO-space der A203 muss also im VME
 * Short-Bereich liegen. */

#define MVME_SHORT      0xFFFFE000

/* Bekanntgabe Carrier Board Interrupt Level am VMEbus;  */
/* ACHTUNG: A203 erlaubt nur 1, 3, 5, 7 (?) */

#define VME_IRQ_LEVEL      5          /* VME IRQ-Level */


/* Auswahl Interrupt-Level, mit dem die M-Modul-Interrupts auf der MVME-162 */
/* generiert werden sollen */

#define NTI_IRQ_LEVEL      6          /* Int-Level am lokalen Bus */


/* Festlegung, wie die via NTI-Register utcsu_misc konfigurierbaren */
/* UTCSU-Pins gesetzt werden sollen (siehe nti.h) */

#define NTI_UTCSU_SIGNALS   NTI_MISC_BIGENDIAN

#endif
```

C.6 pcb_conf_i6360.h

```
/* pcb_conf.h */
/************************************************************************
 *
 * Configuration-File f"ur Setup NTI Carrier-Boards (f"ur i6360)
 *
 * Thomas MANDL
 *
 * $Log: pcb_conf_i6360.h,v $
 * Revision 2.0  1999/01/20 13:11:21  s
 * First RCS revision
 *
 * 12.9.1998 U. Schmid
 *   Restrukturierung aller Header-Files
 *
 * 5.3.1997 Thomas MANDL
 *   Erste Version fertiggestellt.
 *
 *-----------------------------------------------------------------------
 *
 * Dieses Header-File ist das NTI-Analogon zum zentralen Konfigurations-
 * File sys_conf.h des pSOS-Systems. Die hier enthaltenen #defines,
 * die vom Anwender festzulegen sind, werden von den NTI-spezifischen
 * Header- und Programm-Files verarbeitet. Entsprechend der Tatsache, da"s es
 *
 * (1) f"ur jeden COMCO eine eigene NTI (eventuell in mehreren verschiedenen
 *     Versionen)
 * (2) mehrere verschieden anzusprechende M-Modul Carrier-Boards (m"oglicher-
 *     weise von mehreren verschiedenen CPUs aus verwendet)
 *
 * gibt, wurde folgende File-Struktur gew"ahlt:
 *
 * pcb_conf.h           (Dieses File)
 * nti_versions.h       Existierende NTI-Versionen
 * nti.h                Typdefinitionen f"ur die existierenden NTI-Varianten
 * pcb_install.h        Typdefinitionen f"ur die Autoinstall-Funktion
 * pcb_install_<PCB>.c  Autoinstall-Funktion f"ur NTI auf Carrier+CPU <PCB>
 *
 * ANWENDER: Man beachte, da"s pcb_conf.h auch automatisch nti.h (und
 * somit auch utcsu.h) inkludiert.
 *
 ************************************************************************/

#ifndef _PCB_CONF_H
#define _PCB_CONF_H

#include "/ABT/PROJEKTE/SynUTC/TECH/Software/NTI-Handler/nti_versions.h"

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * Definition der verwendeten NTI-Version
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/* COMCO-Typ (siehe nti_versions.h) */
#define NTI_COMCO i82596   /* Intel i82596 Ethernet-Controller (MVME-162) */

/* NTI Versionsnummer (siehe nti_versions.h) */
#define NTI_VERSION NORMAL256K          /* Normalversion NTI mit 256k RAM */


/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * Einbinden der Typdefinitionen + #defines der gew"ahlten NTI
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
#include "/ABT/PROJEKTE/SynUTC/TECH/Software/NTI-Handler/nti.h"


/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * Definition der verwendeten PCB-Variante und Festlegung der Konfigurations-
 * parameter
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/* Einstellung des gew"unschten Carrier-Boards/CPU (siehe nti_versions.h) */
#define NTI_PCB    MVME_I6360   /* MVME-162 CPU mit AcQ i6360 VME-Carrier */


/* Auswahl der M-Module, bei denen (falls montiert) die ISR's in den */
/* Vektoren installiert werden sollen (Grundinstallation und Rest des */
/* Interrupt-Handlings wird immer aufgesetzt (1 == ISRs installieren). */

#define ISR_NTI0_INST         0
#define ISR_NTI1_INST         0
#define ISR_NTI2_INST         1
#define ISR_NTI3_INST         1
#define ISR_NTI4_INST         1
#define ISR_NTI5_INST         1

```

```
/* Auswahl "VME-Page" (im Bereich 0xE0 .. 0xEF), innerhalb der alle */
/* M-Module (IO- und Memory-Space) des Carrier-Boards adressierbar sein */
/* m"ussen (z.B. VME_PAGE==0xE0 f"ur den Bereich 0xE0000000 .. 0xE0FFFFFF) */

#define VME_PAGE         0xE0          /* VME Slave Adressen 0xE0xxxxxx */


/* Bekanntgabe Carrier Board Interrupt Level am VMEbus */
#define VME_IRQ_LEVEL       6          /* VME IRQ-Level */


/* Auswahl Interrupt-Level, mit dem die M-Modul-Interrupts auf der MVME-162 */
/* generiert werden sollen */

#define NTI_IRQ_LEVEL       6          /* Int-Level am lokalen Bus */


/* Festlegung, wie die via NTI-Register utcsu_misc konfigurierbaren */
/* UTCSU-Pins gesetzt werden sollen (siehe nti.h) */

#define NTI_UTCSU_SIGNALS   NTI_MISC_BIGENDIAN

#endif
```

C.7 pcb_install.h

```
/* pcb_install.h */
/***********************************************************************
 *
 * pSOS+ Typdefinitionen f"ur die NTI Autoinstallation
 *
 * U. Schmid
 *
 * $Log: pcb_install.h,v $
 * Revision 2.0  1999/01/20 13:12:53  s
 * First RCS revision
 *
 * 12.10.98 U. Schmid
 *   Added Read-Back Registers for Enable/Disable UTCSU-Interrupts
 *
 * 13.9.1998 U. Schmid
 *   Erste Version
 *
 *----------------------------------------------------------------------
 *
 * Dieses File enth"alt jene generisch Typdefinition NTI_CONFIG, mit
 * der die Konfiguration (Adressen Memory- und IO-Space, Interrupt-Vektor
 * usw.) jeder existierenden NTI-Variante beschrieben werden kann.
 * Sie wird prim"ar von der Autoinstallations-Funktion pcb_install(.)
 * ben"otigt.
 *
 * ANWENDER: Dieses File sollte erst nach pcb_conf.h inkludiert werden.
 *
 ***********************************************************************/

#ifndef _NTI_INSTALL_H
#define _NTI_INSTALL_H

#define MAX_NTI  10              /* Maximale Anzahl NTIs/Carrier-Board */

typedef struct
{
  INT16 installed;      /* In/Out: TRUE(1) wenn NTI installiert werden soll/konnte */
  INT16 slot;           /* Out: Slot-Nummer des M-Moduls */
  NTI *memory;          /* Out: Anfangsadresse Memory-Space */
  NTI_IO_SPACE *io;     /* Out: Anfangsadresse IO-Space */
  UTCSU *utcsu;         /* Out: Anfangsadresse UTCSU */
  INT32 readback_utcinten1; /* Read-Back UTCSU-Register UTCINTEN1 */
  INT32 readback_utcinten2; /* Read-Back UTCSU-Register UTCINTEN2 */
  INT16 nti_irq_vector; /* In: Interrupt Basis-Vektor */
  INT16 nti_ts_offsets; /* In: Timestamp-Offsets (low byte: Transm., high: Recv.) */
  INT16 nti_type_offset;/* In: Typefield-Offset (lowbyte) */
  INT32 utcsu_step;     /* In: Speed UTCSU-Clock (High Part) */
  INT32 utcsu_steplow;  /* In: Speed UTCSU-Clock (Low Part) */
  INT32 utcsu_conf1;    /* In: Konfigurationsregister UTCCONF1 */
  INT32 utcsu_conf2;    /* In: Konfigurationsregister UTCCONF2 */
} NTI_CONFIG;


/* Function-Prototypes f"ur pcb_init_<PCB>.c */

extern int pcb_install();
extern INT32 utcsu_int_enable1(short slot, INT32 interrupts);
extern INT32 utcsu_int_disable1(short slot, INT32 interrupts);
extern INT32 utcsu_int_enable2(short slot, INT32 interrupts);
extern INT32 utcsu_int_disable2(short slot, INT32 interrupts);

#endif
```

C.8 pcb_install_a203.c

```
/* pcb_install_a203.c */
/**********************************************************************
 *
 * MVME-162 Installationsroutine f"ur die Verwendung von NTI M-Modulen
 * auf A203 VME Carrier-Boards
 *
 * Thomas MANDL
 *
 * 17.2.1997 Thomas MANDL
 *   Erste Versuchsversion fertiggestellt
 *
 * 21.9.1998 U. Schmid
 *   Major Revision
 *
 * 12.10.98 U. Schmid
 *   Read-Back-Feature f"ur Enable/Disable UTCSU-Interrupts
 *
 * 17.12.98 U. Schmid
 *   Moved ExcludedVectors to root.c of application.
 *
 *----------------------------------------------------------------------
 *
 * Dieses File stellt eine Funktion nti_install(.) zur Verf"ugung, die
 * es erlaubt, NTI M-Module auf (genau) einem Men A203 VME Carrier-Board
 * von der MVME-162 aus anzusprechen. Prototype:
 *
 *    int pcb_install()
 *
 * wobei ein globales "NTI-Directory" NTI_CONFIG ntidir[4] (siehe
 * pcb_install.h) f"ur den Datenaustausch verwendet wird. Konkret
 * mu"s in ntidir[X] der Interrupt Basis-Vektor sowie der Clock-Speed
 * der UTCSU (STEP, STEPLOW) der NTI im Slot X eingetragen sein.
 * Weiters wird hierin angegeben, welche der Module tats"achlich installiert
 * werden sollen. Nach Aufruf wird in ntidir[X] die Adressen von Memory-Space,
 * IO-Space und UTCSU zur"uckgeliefert und eingetragen, ob das M-Modul X
 * ansprechbar war.
 * Der Return-Wert liefert die Anzahl der erfolgreich installierten Module.
 *
 * Die Funktion mu"s dazu intern folgendes tun:
 *
 * 1. VMEbus-Interface (Master) der MVME-162 konfigurieren
 * 2. Carrier-Board konfigurieren
 * 3. Ansprechbarkeit der NTIs "uberpr"ufen
 * 4. Interrupt-Handling auf den NTIs konfigurieren
 * 5. Interrupt-Handling auf der MVME-162 konfigurieren
 *
 * Die restliche Konfigurations-Information wird "uber #defines importiert,
 * die in pcb_conf.h gesetzt werden:
 *
 * VME_PAGE : "VME-Page" (0xE0, 0xE4, 0xE8, 0xEC), innerhalb der der
 *           Memory-Space aller M-Module des Carrier-Boards
 *           adressierbar ist (z.B. VME_PAGE==0xE0 f"ur
 *           den Bereich 0xE0000000 .. 0xE3FFFFFF).
 *           ACHTUNG: Die A203 verbraucht de facto einen ganzen 256MB
 *                    Block, in dem die M-Module 4 Mal sichtbar sind!
 *                    Daher ist 0xE0, 0xE4, 0xE8 und 0xEC in Wirklichkeit
 *                    gleichbedeutend!!!
 *
 * MVME_SHORT : Startadresse des IO-Spaces M-Modul 0, die in dem
 *             Bereich liegen muss, in dem die MVME Short/D16
 *             Accesses am VMEbus macht. Der IO-space der A203 muss
 *             also im VME Short-Bereich liegen.
 *
 * NTI_UTCSU_MISC : Setting der UTCSU-Eingangssignalen im Register
 *                 utcsu_misc im NTI IO-Space (s. nti.h)
 *
 * VME_IRQ_LEVEL  : Carrier Board Interrupt Level am VMEbus;
 *                 ACHTUNG: Mu"s 1, 3, 5 (oder 7) sein!!!
 *
 * IRQ_LEVEL : Interrupt-Level, mit dem die M-Modul-Interrupts auf
 *            der MVME-162 generiert werden sollen.
 *
 * ISR_NTI<x>_INST : TRUE(1) => ISR M-Modul Slot <x> installieren
 *                              (wenn Installation zur Laufzeit gew"unscht und
 *                               M-Modul ansprechbar)
 *
 * Dazu m"ussen f"ur jedes zu installierende M-Modul<X> ISRs (Funktionen)
 * mit Namen
 *
 *   void isr_nti<X>_n()    ...    INTN Interrupt der NTI/UTCSU<X>
 *   void isr_nti<X>_t()    ...    INTT Interrupt der NTI/UTCSU<X>
 *   void isr_nti<X>_a()    ...    INTA Interrupt der NTI/UTCSU<X>
 *
 * geschrieben werden. Diese werden beim Linken des Moduls dann automatisch
 * eingebunden. Die Funktionen sollten die NTI-Interrupt-Logik nicht
```

```
 * re-enablen, da dies in den Wrapper-Funktionen (siehe unten) passiert.
 * Weiters mu"s eine globale Variable
 *
 *   INT32 PSOS_FLAG
 *
 * existieren, die 1 enth"alt, wenn pSOS l"auft (0 sonst); dies wird
 * f"ur das ISR-Return der Wrapper-ISRs ben"otigt.
 *
 *
 * Dar"uberhinaus werden Funktionen f"ur das Enable/Disable von UTCSU-
 * Interrupts bereitgestellt, die das fehlende Feature des Read-Back
 * der UTCINTEN1 und UTCINTEN2-Register ersetzen. Mit Hilfe von
 *
 * INT32 utcsu_int_enable1(short slot, INT32 interrupts)
 * INT32 utcsu_int_disable1(short slot, INT32 interrupts)
 * INT32 utcsu_int_enable2(short slot, INT32 interrupts)
 * INT32 utcsu_int_disable2(short slot, INT32 interrupts)
 *
 * k"onnen voneinander unabh"angige Anwendungen Interrupts der UTCSU[slot]
 * en/disablen, ohne einander zu beeinflussen.
 *
 *
 * ANWENDER: Das File mu"s mit jeder Anwendung neu compiliert werden!
 *
 ***********************************************************************/

#include <types.h>
#include "pcb_conf.h"

#if NTI_PCB != MVME_A203
!!!!! ERROR: Illegal PCB-Type (Expected: MVME_A203; Check pcb_conf.h) !!!!!
#endif

#include "/usr/isi/apps/SynUTC/nti/pcb_install.h"
#include "/usr/isi/bsps/m162_5xx/src/mvme162.h"
#include <util.h>
#include "/usr/isi/apps/SynUTC/nti/a203.h"

/*-----------------------------------------------------------------------
 * Externals
 *-----------------------------------------------------------------------*/

extern NTI_CONFIG ntidir[];  /* NTI-Directory, containing M-Module config. */
extern INT32 PSOS_FLAG;     /* 1 ... pSOS running, 0 else */


#if ISR_NTI0_INST  /* Slot 0 is to be considered */
extern void isr_nti0_n();
extern void isr_nti0_t();
extern void isr_nti0_a();
#endif
#if ISR_NTI1_INST  /* Slot 1 is to be considered */
extern void isr_nti1_n();
extern void isr_nti1_t();
extern void isr_nti1_a();
#endif
#if ISR_NTI2_INST  /* Slot 2 is to be considered */
extern void isr_nti2_n();
extern void isr_nti2_t();
extern void isr_nti2_a();
#endif
#if ISR_NTI3_INST  /* Slot 3 is to be considered */
extern void isr_nti3_n();
extern void isr_nti3_t();
extern void isr_nti3_a();
#endif

/*-----------------------------------------------------------------------
 * Defines
 *-----------------------------------------------------------------------*/

#define SLOT_MEMSIZE 0x01000000  /* A203 memory-space per slot */


/*-----------------------------------------------------------------------
 * Globals
 *-----------------------------------------------------------------------*/

INT32 carrier_base = VME_PAGE << 24; /* We always require 0x<PAGE>000000 */
A203 *a203 = (A203 *) (MVME_SHORT); /* A203 IO-Spaces */

/*-----------------------------------------------------------------------
 * Wrapped ISRs
 *-----------------------------------------------------------------------*/

#if ISR_NTI0_INST  /* Slot 0 is to be considered */
INT32 wrisr_nti0_nta()
{
```

```
  a203->slot[0].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti0_n(); isr_nti0_t(); isr_nti0_a();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti0_nt()
{
  a203->slot[0].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti0_n(); isr_nti0_t();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti0_na()
{
  a203->slot[0].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti0_n(); isr_nti0_a();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti0_ta()
{
  a203->slot[0].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti0_t(); isr_nti0_a();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti0_n()
{
  a203->slot[0].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti0_n();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti0_a()
{
  a203->slot[0].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti0_a();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti0_t()
{
  a203->slot[0].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti0_t();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
#endif

#if ISR_NTI1_INST  /* Slot 1 is to be considered */
INT32 wrisr_nti1_nta()
{
  a203->slot[1].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti1_n(); isr_nti1_t(); isr_nti1_a();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti1_nt()
{
  a203->slot[1].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti1_n(); isr_nti1_t();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti1_na()
{
  a203->slot[1].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti1_n(); isr_nti1_a();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti1_ta()
{
  a203->slot[1].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti1_t(); isr_nti1_a();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti1_n()
{
  a203->slot[1].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti1_n();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti1_a()
{
  a203->slot[1].micr |= A203_IPEND; /* Clear A203-Interrupt */
```

```
  isr_nti1_a();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti1_t()
{
  a203->slot[1].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti1_t();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
#endif

#if ISR_NTI2_INST  /* Slot 2 is to be considered */
INT32 wrisr_nti2_nta()
{
  a203->slot[2].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti2_n(); isr_nti2_t(); isr_nti2_a();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti2_nt()
{
  a203->slot[2].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti2_n(); isr_nti2_t();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti2_na()
{
  a203->slot[2].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti2_n(); isr_nti2_a();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti2_ta()
{
  a203->slot[2].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti2_t(); isr_nti2_a();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti2_n()
{
  a203->slot[2].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti2_n();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti2_a()
{
  a203->slot[2].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti2_a();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti2_t()
{
  a203->slot[2].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti2_t();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
#endif

#if ISR_NTI3_INST  /* Slot 3 is to be considered */
INT32 wrisr_nti3_nta()
{
  a203->slot[3].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti3_n(); isr_nti3_t(); isr_nti3_a();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti3_nt()
{
  a203->slot[3].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti3_n(); isr_nti3_t();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti3_na()
{
  a203->slot[3].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti3_n(); isr_nti3_a();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti3_ta()
```

```
{
  a203->slot[3].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti3_t(); isr_nti3_a();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti3_n()
{
  a203->slot[3].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti3_n();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti3_a()
{
  a203->slot[3].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti3_a();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti3_t()
{
  a203->slot[3].micr |= A203_IPEND; /* Clear A203-Interrupt */
  isr_nti3_t();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
#endif



INT32 isr_nti_illegal()
{
  asm(void, " TRAP #$0");   /* Trap to pROBE */
  return PSOS_FLAG;
}



/*------------------------------------------------------------------------
 * Function utcsu_int_enable1(.) --- Enable UTCSU-Interrupts in UTCSUINTEN1
 * Enable aller im Parameter interrupts mit Bit=1 markierten UTCINTEN1-Interrupts
 * der UTCSU[slot] und retournieren des neuen Settings.
 *------------------------------------------------------------------------*/

INT32 utcsu_int_enable1(short slot, INT32 interrupts)
{
  INT32 oldmask = splx(7); /* Disable all CPU-Interrupts */
  INT32 newsettings = (ntidir[slot].readback_utcinten1 |= interrupts);
  ntidir[slot].utcsu->w32.utcinten1 = newsettings; /* Set UTCSU-Register UTCINTEN1 */
  splx(oldmask); /* Re-Enable CPU-Interrupts */
  return newsettings;
}


/*------------------------------------------------------------------------
 * Function utcsu_int_disable1(.) --- Disable UTCSU-Interrupts in UTCSUINTEN1
 * Disable aller im Parameter interrupts mit Bit=1 markierten UTCINTEN1-Interrupts der
 * der UTCSU[slot] und retournieren des neuen Settings.
 *------------------------------------------------------------------------*/

INT32 utcsu_int_disable1(short slot, INT32 interrupts)
{
  INT32 oldmask = splx(7); /* Disable all CPU-Interrupts */
  INT32 newsettings = (ntidir[slot].readback_utcinten1 &= ~interrupts);
  ntidir[slot].utcsu->w32.utcinten1 = newsettings; /* Set UTCSU-Register UTCINTEN1 */
  splx(oldmask); /* Re-Enable CPU-Interrupts */
  return newsettings;
}

/*------------------------------------------------------------------------
 * Function utcsu_int_enable2(.) --- Enable UTCSU-Interrupts in UTCSUINTEN2
 * Enable aller im Parameter interrupts mit Bit=1 markierten UTCINTEN2-Interrupts
 * der UTCSU[slot] und retournieren des neuen Settings.
 *------------------------------------------------------------------------*/

INT32 utcsu_int_enable2(short slot, INT32 interrupts)
{
  INT32 oldmask = splx(7); /* Disable all CPU-Interrupts */
  INT32 newsettings = (ntidir[slot].readback_utcinten2 |= interrupts);
  ntidir[slot].utcsu->w32.utcinten2 = newsettings; /* Set UTCSU-Register UTCINTEN2 */
  splx(oldmask); /* Re-Enable CPU-Interrupts */
  return newsettings;
}


/*------------------------------------------------------------------------
 * Function utcsu_int_disable2(.) --- Disable UTCSU-Interrupts in UTCSUINTEN2
```

```
 * Disable aller im Parameter interrupts mit Bit=1 markierten UTCINTEN2-Interrupts der
 * der UTCSU[slot] und retournieren des neuen Settings.
 *-----------------------------------------------------------------------*/

INT32 utcsu_int_disable2(short slot, INT32 interrupts)
{
  INT32 oldmask = splx(7); /* Disable all CPU-Interrupts */
  INT32 newsettings = (ntidir[slot].readback_utcinten2 &= ~interrupts);
  ntidir[slot].utcsu->w32.utcinten2 = newsettings; /* Set UTCSU-Register UTCINTEN2 */
  splx(oldmask); /* Re-Enable CPU-Interrupts */
  return newsettings;
}



/*-----------------------------------------------------------------------
 * Function pcb_install() --- Install M-Modules
 *-----------------------------------------------------------------------*/

int pcb_install()
{
  short slot, count, spin;
  INT32 i32_temp;
  NTI_IO_SPACE *nti_io;
  UTCSU *utcsu;
  VMEchip2_LCSR_REGS *vme = VMECHIP2; /* Basis-Adresse MVME-162 VME-Chip */

  count = 0; /* Anzahl erfolgreich installierter Module retour */

  /* Initialize Return-Felder in NTI-directory */
  for (slot=0;slot<4;slot++)
  {
    /* Clear Read-Back Registers for UTCINTEN1, UTCINTEN2 */
    ntidir[slot].readback_utcinten1 = 0;
    ntidir[slot].readback_utcinten2 = 0;

    /* Setup pointers to memory space, io-space, and UTCSU */
    ntidir[slot].slot = slot;
    ntidir[slot].memory = (NTI *)(carrier_base + slot * SLOT_MEMSIZE);
    ntidir[slot].utcsu = &(ntidir[slot].memory->utcsu);
    ntidir[slot].io = (NTI_IO_SPACE *)(&a203->slot[slot]);
  }

  /* First, ensure that pSOS's VME A32/D32 address range covers all the */
  /* M-Module's Memory-Space <VME_PAGE> ... <VME_PAGE>+4*SLOT_MEMSIZE-1 */
  /* but nothing else; this is done by VME-Master 1 */
  if ((vme->master_addr_reg1 & 0xFFFF0000) < (carrier_base + 4*SLOT_MEMSIZE - 0x10000))
  {
    vme->master_addr_reg1 = (vme->master_addr_reg1 & 0x0000FFFF) | (carrier_base + 4*SLOT_MEMSIZE - 0x10000);
  }

  /* Initialize Interrupt-Vector Registers, Interrupt-Control Registers */
  /* and DMA-Control Registers of the A203 */
  for (slot=0; slot<4; slot++)
  {
    /* Set the A203 Slave Address Compare Registers to 0x<VME_PAGE>000000 */
    a203->slot[slot].sacr = VME_PAGE;

    /* Set Vector to external and set level */
    a203->slot[slot].micr = A203_IEXT | VME_IRQ_LEVEL;
    a203->slot[slot].mivr = 0;  /* set A203's vectors for M-Module (don't care) */
    a203->slot[slot].divr = 0;
  }

  /* Check whether accesses to NTI M-Module work. */
  for (slot=0;slot<4;slot++)
  {
    if (ntidir[slot].installed)
      if (MemAccessible((void *)ntidir[slot].memory))
        ntidir[slot].installed = MemAccessible((void *)ntidir[slot].io);
      else
        ntidir[slot].installed = 0;  /* Something wrong with memory-space */

    /* We clear Local Bus Error of the MVME (MemAccessible) */
    i32_temp = vme->dma_int_ctrl_regs;
    vme->dma_int_ctrl_regs = i32_temp | (1 << 11);
    i32_temp = vme->dma_int_ctrl_regs;

    if (ntidir[slot].installed)
    {
      count++;               /* Eine erfolgreiche Installation mehr ... */
      nti_io = ntidir[slot].io;
      nti_io->int_logic = NTI_DIS_INT_LOGIC; /* Disable NTI interrupts */
      nti_io->vector_base = ntidir[slot].nti_irq_vector;
      nti_io->type_offset = ntidir[slot].nti_type_offset; /* Typefield Offsets */
      nti_io->ts_offsets =  ntidir[slot].nti_ts_offsets; /* TS-Trigger Offsets */

      /* Now enable A203 interrupt controller */
```

```
        a203->slot[slot].micr = a203->slot[slot].micr; /* Clear IPEND if set ... */
        a203->slot[slot].micr = a203->slot[slot].micr | A203_IRE;

        /* Set UTCSU-signals and force UTCSU reset */
        nti_io->utcsu_misc = NTI_UTCSU_SIGNALS | NTI_MISC_RESET;
        for (spin=0;spin<100;spin++);  /* Wait a bit  ... */
        nti_io->utcsu_misc = NTI_UTCSU_SIGNALS;

        utcsu = ntidir[slot].utcsu;
        utcsu->w32.swreset = 1;        /* Software-reset as well */

        /* Now start the UTCSU-clock */

        utcsu->w32.stepamort = 0;
        utcsu->w32.steppure = ntidir[slot].utcsu_step; /* Register STEP */
        utcsu->w32.amorttimeset = 1; /* Das Ganze aktivieren ... */
        utcsu->w32.startamort = 1;   /* W"urde INTT ausl"osen ... */

        /* Preset Time */
        utcsu->w32.msset = ntidir[slot].utcsu_steplow & 0xFF000000; /* Register STEP_LOW dabei !*/
        utcsu->w32.tsset = 0x00000000;
        utcsu->w32.usset = 0x00000000;
        utcsu->w32.ntpset = 1;         /* Los ... */

        /* Setup UTCCONF-registers */
        utcsu->w32.utcconf1 = ntidir[slot].utcsu_conf1;
        utcsu->w32.utcconf2 = ntidir[slot].utcsu_conf2;

        /* Clear and disable all UTCSU int-soures */
        utcsu->w32.utcinten1 = 0x00000000;
        utcsu->w32.utcinten2 = 0x00000000;
        utcsu->w32.utcintclear1 = 0xFFFFFFFF;
        utcsu->w32.utcintclear2 = 0xFFFFFFFF;

        nti_io->int_logic = NTI_EN_INT_LOGIC; /* UTCSU-Int's are still off */

      } /* if(installed) */
    } /* for(slot) */

    /* Installieren der ISRs in den Vektoren */

  #if ISR_NTI0_INST
    if (ntidir[0].installed)
    { SysSetVector(ntidir[0].nti_irq_vector, isr_nti_illegal, NI_WRAPPER);
      SysSetVector(ntidir[0].nti_irq_vector | (0x01), wrisr_nti0_a, NI_WRAPPER);
      SysSetVector(ntidir[0].nti_irq_vector | (0x02), wrisr_nti0_t, NI_WRAPPER);
      SysSetVector(ntidir[0].nti_irq_vector | (0x03), wrisr_nti0_ta, NI_WRAPPER);
      SysSetVector(ntidir[0].nti_irq_vector | (0x04), wrisr_nti0_n, NI_WRAPPER);
      SysSetVector(ntidir[0].nti_irq_vector | (0x05), wrisr_nti0_na, NI_WRAPPER);
      SysSetVector(ntidir[0].nti_irq_vector | (0x06), wrisr_nti0_nt, NI_WRAPPER);
      SysSetVector(ntidir[0].nti_irq_vector | (0x07), wrisr_nti0_nta, NI_WRAPPER);
    }
  #endif
  #if ISR_NTI1_INST
    if (ntidir[1].installed)
    { SysSetVector(ntidir[1].nti_irq_vector, isr_nti_illegal, NI_WRAPPER);
      SysSetVector(ntidir[1].nti_irq_vector | (0x01), wrisr_nti1_a, NI_WRAPPER);
      SysSetVector(ntidir[1].nti_irq_vector | (0x02), wrisr_nti1_t, NI_WRAPPER);
      SysSetVector(ntidir[1].nti_irq_vector | (0x03), wrisr_nti1_ta, NI_WRAPPER);
      SysSetVector(ntidir[1].nti_irq_vector | (0x04), wrisr_nti1_n, NI_WRAPPER);
      SysSetVector(ntidir[1].nti_irq_vector | (0x05), wrisr_nti1_na, NI_WRAPPER);
      SysSetVector(ntidir[1].nti_irq_vector | (0x06), wrisr_nti1_nt, NI_WRAPPER);
      SysSetVector(ntidir[1].nti_irq_vector | (0x07), wrisr_nti1_nta, NI_WRAPPER);
    }
  #endif
  #if ISR_NTI2_INST
    if (ntidir[2].installed)
    { SysSetVector(ntidir[2].nti_irq_vector, isr_nti_illegal, NI_WRAPPER);
      SysSetVector(ntidir[2].nti_irq_vector | (0x01), wrisr_nti2_a, NI_WRAPPER);
      SysSetVector(ntidir[2].nti_irq_vector | (0x02), wrisr_nti2_t, NI_WRAPPER);
      SysSetVector(ntidir[2].nti_irq_vector | (0x03), wrisr_nti2_ta, NI_WRAPPER);
      SysSetVector(ntidir[2].nti_irq_vector | (0x04), wrisr_nti2_n, NI_WRAPPER);
      SysSetVector(ntidir[2].nti_irq_vector | (0x05), wrisr_nti2_na, NI_WRAPPER);
      SysSetVector(ntidir[2].nti_irq_vector | (0x06), wrisr_nti2_nt, NI_WRAPPER);
      SysSetVector(ntidir[2].nti_irq_vector | (0x07), wrisr_nti2_nta, NI_WRAPPER);
    }
  #endif
  #if ISR_NTI3_INST
    if (ntidir[3].installed)
    { SysSetVector(ntidir[3].nti_irq_vector, isr_nti_illegal, NI_WRAPPER);
      SysSetVector(ntidir[3].nti_irq_vector | (0x01), wrisr_nti3_a, NI_WRAPPER);
      SysSetVector(ntidir[3].nti_irq_vector | (0x02), wrisr_nti3_t, NI_WRAPPER);
      SysSetVector(ntidir[3].nti_irq_vector | (0x03), wrisr_nti3_ta, NI_WRAPPER);
      SysSetVector(ntidir[3].nti_irq_vector | (0x04), wrisr_nti3_n, NI_WRAPPER);
      SysSetVector(ntidir[3].nti_irq_vector | (0x05), wrisr_nti3_na, NI_WRAPPER);
      SysSetVector(ntidir[3].nti_irq_vector | (0x06), wrisr_nti3_nt, NI_WRAPPER);
      SysSetVector(ntidir[3].nti_irq_vector | (0x07), wrisr_nti3_nta, NI_WRAPPER);
    }
```

```
#endif


  /* Finally setup VME interrupt service */

  /* Set MVME162-512A Local Bus Interrupt Level */
  vme->int_lvl_reg4 = (vme->int_lvl_reg4 & 0xF0000000) |
                      (NTI_IRQ_LEVEL << (4 * (VME_IRQ_LEVEL - 1)));

  /* VME Interrupt-Level enable (= Level generated by i6360) */
  vme->loc_bus_int_en_reg = (vme->loc_bus_int_en_reg & 0xFFFFFF80) |
                            (1 << (VME_IRQ_LEVEL - 1));

  i32_temp = vme->vect_base_io_ctrl_reg;
  vme->vect_base_io_ctrl_reg = i32_temp | 0x00800000; /* Master-Int. enable */

  return count;
}
```

C.9 pcb_install_i6360.c

```
/* pcb_install_i6360.c */
/**********************************************************************
 *
 * MVME-162 Installationsroutine f"ur die Verwendung von NTI M-Modulen
 * auf i6360 VME Carrier-Boards
 *
 * Thomas MANDL
 *
 * 17.2.1997 Thomas MANDL
 *   Erste Version fertiggestellt
 *
 * 14.9.1998 U. Schmid
 *   Major Revision fertig
 *
 * 12.10.98 U. Schmid
 *   Read-Back-Feature f"ur Enable/Disable UTCSU-Interrupts
 *
 * 17.12.98 U. Schmid
 *   Moved ExcludedVectors to root.c of application.
 *
 *----------------------------------------------------------------------
 *
 * Dieses File stellt eine Funktion nti_install(.) zur Verf"ugung, die
 * es erlaubt, NTI M-Module auf (genau) einem AcQ i6360 VME Carrier-Board
 * von der MVME-162 aus anzusprechen. Prototype:
 *
 *    int pcb_install()
 *
 * wobei ein globales "NTI-Directory" NTI_CONFIG ntidir[6] (siehe
 * pcb_install.h) f"ur den Datenaustausch verwendet wird. Konkret
 * mu"s in ntidir[X] der Interrupt Basis-Vektor sowie der Clock-Speed
 * der UTCSU (STEP, STEPLOW) der NTI im Slot X eingetragen sein.
 * Weiters wird hierin angegeben, welche der Module tats"achlich installiert
 * werden sollen. Nach Aufruf wird in ntidir[X] die Adressen von Memory-Space,
 * IO-Space und UTCSU zur"uckgeliefert und eingetragen, ob das M-Modul X
 * ansprechbar war.
 * Der Return-Wert liefert die Anzahl der erfolgreich installierten Module.
 *
 * Die Funktion mu"s dazu intern folgendes tun:
 *
 * 1. VMEbus-Interface (Master) der MVME-162 konfigurieren
 * 2. Carrier-Board konfigurieren
 * 3. Ansprechbarkeit der NTIs "uberpr"ufen
 * 4. Interrupt-Handling auf den NTIs konfigurieren
 * 5. Interrupt-Handling auf der MVME-162 konfigurieren
 *
 * Die restliche Konfigurations-Information wird "uber #defines importiert,
 * die in pcb_conf.h gesetzt werden:
 *
 * VME_PAGE : "VME-Page" (im Bereich 0xE0 .. 0xEF), innerhalb der alle
 *            M-Module (IO- und Memory-Space) des Carrier-Boards
 *            adressierbar sein m"ussen (z.B. VME_PAGE==0xE0 f"ur
 *            den Bereich 0xE0000000 .. 0xE0FFFFFF)
 *
 * NTI_UTCSU_MISC : Setting der UTCSU-Eingangssignalen im Register
 *                  utcsu_misc im NTI IO-Space (s. nti.h)
 *
 * VME_IRQ_LEVEL  : Carrier Board Interrupt Level am VMEbus
 *
 * IRQ_LEVEL : Interrupt-Level, mit dem die M-Modul-Interrupts auf
 *             der MVME-162 generiert werden sollen.
 *
 * ISR_NTI<x>_INST : TRUE(1) => ISR M-Modul Slot <x> installieren
 *                              (wenn Installation zur Laufzeit gew"unscht und
 *                               M-Modul ansprechbar)
 *
 * Dazu m"ussen f"ur jedes zu installierende M-Modul<X> ISRs (Funktionen)
 * mit Namen
 *
 *   void isr_nti<X>_n()    ...    INTN Interrupt der NTI/UTCSU<X>
 *   void isr_nti<X>_t()    ...    INTT Interrupt der NTI/UTCSU<X>
 *   void isr_nti<X>_a()    ...    INTA Interrupt der NTI/UTCSU<X>
 *
 * geschrieben werden. Diese werden beim Linken des Moduls dann automatisch
 * eingebunden. Die Funktionen sollten die NTI-Interrupt-Logik nicht
 * re-enablen, da dies in den Wrapper-Funktionen (siehe unten) passiert.
 * Weiters mu"s eine globale Variable
 *
 *   INT32 PSOS_FLAG
 *
 * existieren, die 1 enth"alt, wenn pSOS l"auft (0 sonst); dies wird
 * f"ur das ISR-Return der Wrapper-ISRs ben"otigt.
 *
 *
 * Dar"uberhinaus werden Funktionen f"ur das Enable/Disable von UTCSU-
```

```
 * Interrupts bereitgestellt, die das fehlende Feature des Read-Back
 * der UTCINTEN1 und UTCINTEN2-Register ersetzen. Mit Hilfe von
 *
 * INT32 utcsu_int_enable1(short slot, INT32 interrupts)
 * INT32 utcsu_int_disable1(short slot, INT32 interrupts)
 * INT32 utcsu_int_enable2(short slot, INT32 interrupts)
 * INT32 utcsu_int_disable2(short slot, INT32 interrupts)
 *
 * k"onnen voneinander unabh"angige Anwendungen Interrupts der UTCSU[slot]
 * en/disablen, ohne einander zu beeinflussen.
 *
 *
 * ANWENDER: Das File mu"s mit jeder Anwendung neu compiliert werden!
 *
 ************************************************************************/

#include <types.h>
#include "pcb_conf.h"

#if NTI_PCB != MVME_I6360
!!!!! ERROR: Illegal PCB-Type (Expected: MVME_I6360; Check pcb_conf.h) !!!!!
#endif

#include "/usr/isi/apps/SynUTC/nti/pcb_install.h"
#include "/usr/isi/bsps/m162_5xx/src/mvme162.h"
#include <util.h>

/*------------------------------------------------------------------------
 * Externals
 *------------------------------------------------------------------------*/

extern NTI_CONFIG ntidir[];  /* NTI-Directory, containing Module config. */
extern INT32 PSOS_FLAG;     /* 1 ... pSOS running, 0 else */


#if ISR_NTI0_INST  /* Slot 0 is to be considered */
extern void isr_nti0_n();
extern void isr_nti0_t();
extern void isr_nti0_a();
#endif
#if ISR_NTI1_INST  /* Slot 1 is to be considered */
extern void isr_nti1_n();
extern void isr_nti1_t();
extern void isr_nti1_a();
#endif
#if ISR_NTI2_INST  /* Slot 2 is to be considered */
extern void isr_nti2_n();
extern void isr_nti2_t();
extern void isr_nti2_a();
#endif
#if ISR_NTI3_INST  /* Slot 3 is to be considered */
extern void isr_nti3_n();
extern void isr_nti3_t();
extern void isr_nti3_a();
#endif
#if ISR_NTI4_INST  /* Slot 4 is to be considered */
extern void isr_nti4_n();
extern void isr_nti4_t();
extern void isr_nti4_a();
#endif
#if ISR_NTI5_INST   /* Slot 5 is to be considered */
extern void isr_nti5_n();
extern void isr_nti5_t();
extern void isr_nti5_a();
#endif


/*------------------------------------------------------------------------
 * Defines
 *------------------------------------------------------------------------*/

#define SLOT_MEMSIZE 0x00100000  /* i6360 memory-space per slot */
#define IOOFFSET0    0x00600000  /* Offset carrier base to iospace slot #0 */
#define IOOFFSET2    0x00700000  /* Offset carrier base to iospace slot #2 */
#define SLOT_IOSIZE  0x100       /* i6360 iospace per slot */


/*------------------------------------------------------------------------
 * Globals
 *------------------------------------------------------------------------*/

INT32 carrier_base = VME_PAGE << 24; /* We always require 0x<PAGE>000000 */


/*------------------------------------------------------------------------
 * Wrapped ISRs
 *------------------------------------------------------------------------*/
```

```
#if ISR_NTI0_INST  /* Slot 0 is to be considered */
INT32 wrisr_nti0_nta()
{
  isr_nti0_n(); isr_nti0_t(); isr_nti0_a();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti0_nt()
{
  isr_nti0_n(); isr_nti0_t();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti0_na()
{
  isr_nti0_n(); isr_nti0_a();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti0_ta()
{
  isr_nti0_t(); isr_nti0_a();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti0_n()
{
  isr_nti0_n();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti0_a()
{
  isr_nti0_a();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti0_t()
{
  isr_nti0_t();
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
#endif

#if ISR_NTI1_INST  /* Slot 1 is to be considered */
INT32 wrisr_nti1_nta()
{
  isr_nti1_n(); isr_nti1_t(); isr_nti1_a();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti1_nt()
{
  isr_nti1_n(); isr_nti1_t();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti1_na()
{
  isr_nti1_n(); isr_nti1_a();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti1_ta()
{
  isr_nti1_t(); isr_nti1_a();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti1_n()
{
  isr_nti1_n();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti1_a()
{
  isr_nti1_a();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti1_t()
{
  isr_nti1_t();
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
```

```
#endif

#if ISR_NTI2_INST  /* Slot 2 is to be considered */
INT32 wrisr_nti2_nta()
{
  isr_nti2_n(); isr_nti2_t(); isr_nti2_a();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti2_nt()
{
  isr_nti2_n(); isr_nti2_t();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti2_na()
{
  isr_nti2_n(); isr_nti2_a();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti2_ta()
{
  isr_nti2_t(); isr_nti2_a();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti2_n()
{
  isr_nti2_n();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti2_a()
{
  isr_nti2_a();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti2_t()
{
  isr_nti2_t();
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
#endif

#if ISR_NTI3_INST  /* Slot 3 is to be considered */
INT32 wrisr_nti3_nta()
{
  isr_nti3_n(); isr_nti3_t(); isr_nti3_a();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti3_nt()
{
  isr_nti3_n(); isr_nti3_t();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti3_na()
{
  isr_nti3_n(); isr_nti3_a();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti3_ta()
{
  isr_nti3_t(); isr_nti3_a();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti3_n()
{
  isr_nti3_n();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti3_a()
{
  isr_nti3_a();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti3_t()
{
  isr_nti3_t();
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
```

```
  return PSOS_FLAG;
}
#endif

#if ISR_NTI4_INST  /* Slot 4 is to be considered */
INT32 wrisr_nti4_nta()
{
  isr_nti4_n(); isr_nti4_t(); isr_nti4_a();
  ntidir[4].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti4_nt()
{
  isr_nti4_n(); isr_nti4_t();
  ntidir[4].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti4_na()
{
  isr_nti4_n(); isr_nti4_a();
  ntidir[4].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti4_ta()
{
  isr_nti4_t(); isr_nti4_a();
  ntidir[4].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti4_n()
{
  isr_nti4_n();
  ntidir[4].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti4_a()
{
  isr_nti4_a();
  ntidir[4].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti4_t()
{
  isr_nti4_t();
  ntidir[4].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
#endif

#if ISR_NTI5_INST   /* Slot 5 is to be considered */
INT32 wrisr_nti5_nta()
{
  isr_nti5_n(); isr_nti5_t(); isr_nti5_a();
  ntidir[5].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti5_nt()
{
  isr_nti5_n(); isr_nti5_t();
  ntidir[5].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti5_na()
{
  isr_nti5_n(); isr_nti5_a();
  ntidir[5].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti5_ta()
{
  isr_nti5_t(); isr_nti5_a();
  ntidir[5].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti5_n()
{
  isr_nti5_n();
  ntidir[5].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti5_a()
{
  isr_nti5_a();
  ntidir[5].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
INT32 wrisr_nti5_t()
{
```

```
  isr_nti5_t();
  ntidir[5].io->int_logic = NTI_EN_INT_LOGIC;
  return PSOS_FLAG;
}
#endif


INT32 isr_nti_illegal()
{

/*  asm(void, " TRAP #$0");   */ /* This is the usual code */

/*----------------------------------------------------------------------
 * Wir mi"sbrauchen diese ISR als Workaround f"ur die Vektor-Probleme der
 * AcQ i6360 (bis zu deren Behebung). Dazu wird die i6360 so programmiert,
 * da"s sie f"ur alle M-Module nur einen einzigen Vektor liefert (der
 * von den M-Modulen gelieferte Vektor wird ignoriert). Praktischerweise
 * verwenden wir dazu einen der Basis-Vektoren, der selbst normalerweise
 * nie auftreten kann (da die letzten Bits==000 sind). Die f"ur diese
 * Vektoren installierte ISR (eben isr_nti_illegal) wird als Interrupt-
 * Dispatcher verwendet: Sie checked die IntStatus-Register aller M-Module
 * und ruft die entsprechenden Funktionen direkt auf.
 *----------------------------------------------------------------------*/

#if ISR_NTI0_INST  /* Slot 0 is to be considered */
  if (ntidir[0].installed)
  {
    if ((ntidir[0].utcsu->r32.utcintstat2 & ntidir[0].readback_utcinten2 & 0x000FFFFF) != 0)
      isr_nti0_n();   /* Ein INTN ist pending */
    if (((ntidir[0].utcsu->r32.utcintstat1 & ntidir[0].readback_utcinten1 & 0x00000FFF) != 0) ||
        ((ntidir[0].utcsu->r32.utcintstat2 & ntidir[0].readback_utcinten2 & 0xFFF00000) != 0))
      isr_nti0_t();   /* Ein INTT ist pending */
    if ((ntidir[0].utcsu->r32.utcintstat1 & ntidir[0].readback_utcinten1 & 0xFFFFF000) != 0)
      isr_nti0_a();   /* Ein INTA ist pending */
  ntidir[0].io->int_logic = NTI_EN_INT_LOGIC;
  }
#endif
#if ISR_NTI1_INST  /* Slot 1 is to be considered */
  if (ntidir[1].installed)
  {
    if ((ntidir[1].utcsu->r32.utcintstat2 & ntidir[1].readback_utcinten2 & 0x000FFFFF) != 0)
      isr_nti1_n();   /* Ein INTN ist pending */
    if (((ntidir[1].utcsu->r32.utcintstat1 & ntidir[1].readback_utcinten1 & 0x00000FFF) != 0) ||
        ((ntidir[1].utcsu->r32.utcintstat2 & ntidir[1].readback_utcinten2 & 0xFFF00000) != 0))
      isr_nti1_t();   /* Ein INTT ist pending */
    if ((ntidir[1].utcsu->r32.utcintstat1 & ntidir[1].readback_utcinten1 & 0xFFFFF000) != 0)
      isr_nti1_a();   /* Ein INTA ist pending */
  ntidir[1].io->int_logic = NTI_EN_INT_LOGIC;
  }
#endif
#if ISR_NTI2_INST  /* Slot 2 is to be considered */
  if (ntidir[2].installed)
  {
    if ((ntidir[2].utcsu->r32.utcintstat2 & ntidir[2].readback_utcinten2 & 0x000FFFFF) != 0)
      isr_nti2_n();   /* Ein INTN ist pending */
    if (((ntidir[2].utcsu->r32.utcintstat1 & ntidir[2].readback_utcinten1 & 0x00000FFF) != 0) ||
        ((ntidir[2].utcsu->r32.utcintstat2 & ntidir[2].readback_utcinten2 & 0xFFF00000) != 0))
      isr_nti2_t();   /* Ein INTT ist pending */
    if ((ntidir[2].utcsu->r32.utcintstat1 & ntidir[2].readback_utcinten1 & 0xFFFFF000) != 0)
      isr_nti2_a();   /* Ein INTA ist pending */
  ntidir[2].io->int_logic = NTI_EN_INT_LOGIC;
  }
#endif
#if ISR_NTI3_INST  /* Slot 3 is to be considered */
  if (ntidir[3].installed)
  {
    if ((ntidir[3].utcsu->r32.utcintstat2 & ntidir[3].readback_utcinten2 & 0x000FFFFF) != 0)
      isr_nti3_n();   /* Ein INTN ist pending */
    if (((ntidir[3].utcsu->r32.utcintstat1 & ntidir[3].readback_utcinten1 & 0x00000FFF) != 0) ||
        ((ntidir[3].utcsu->r32.utcintstat2 & ntidir[3].readback_utcinten2 & 0xFFF00000) != 0))
      isr_nti3_t();   /* Ein INTT ist pending */
    if ((ntidir[3].utcsu->r32.utcintstat1 & ntidir[3].readback_utcinten1 & 0xFFFFF000) != 0)
      isr_nti3_a();   /* Ein INTA ist pending */
  ntidir[3].io->int_logic = NTI_EN_INT_LOGIC;
  }
#endif
#if ISR_NTI4_INST  /* Slot 4 is to be considered */
  if (ntidir[4].installed)
  {
    if ((ntidir[4].utcsu->r32.utcintstat2 & ntidir[4].readback_utcinten2 & 0x000FFFFF) != 0)
      isr_nti4_n();   /* Ein INTN ist pending */
    if (((ntidir[4].utcsu->r32.utcintstat1 & ntidir[4].readback_utcinten1 & 0x00000FFF) != 0) ||
        ((ntidir[4].utcsu->r32.utcintstat2 & ntidir[4].readback_utcinten2 & 0xFFF00000) != 0))
      isr_nti4_t();   /* Ein INTT ist pending */
    if ((ntidir[4].utcsu->r32.utcintstat1 & ntidir[4].readback_utcinten1 & 0xFFFFF000) != 0)
      isr_nti4_a();   /* Ein INTA ist pending */
  ntidir[4].io->int_logic = NTI_EN_INT_LOGIC;
  }
```

```
#endif
#if ISR_NTI5_INST  /* Slot 5 is to be considered */
  if (ntidir[5].installed)
  {
    if ((ntidir[5].utcsu->r32.utcintstat2 & ntidir[5].readback_utcinten2 & 0x000FFFFF) != 0)
      isr_nti5_n();   /* Ein INTN ist pending */
    if (((ntidir[5].utcsu->r32.utcintstat1 & ntidir[5].readback_utcinten1 & 0x00000FFF) != 0) ||
        ((ntidir[5].utcsu->r32.utcintstat2 & ntidir[5].readback_utcinten2 & 0xFFF00000) != 0))
      isr_nti5_t();   /* Ein INTT ist pending */
    if ((ntidir[5].utcsu->r32.utcintstat1 & ntidir[5].readback_utcinten1 & 0xFFFFF000) != 0)
      isr_nti5_a();   /* Ein INTA ist pending */
  ntidir[5].io->int_logic = NTI_EN_INT_LOGIC;
  }
#endif

  return PSOS_FLAG;
}


/*-------------------------------------------------------------------------
 * Function utcsu_int_enable1(.) --- Enable UTCSU-Interrupts in UTCSUINTEN1
 * Enable aller im Parameter interrupts mit Bit=1 markierten UTCINTEN1-Interrupts
 * der UTCSU[slot] und retournieren des neuen Settings.
 *-------------------------------------------------------------------------*/

INT32 utcsu_int_enable1(short slot, INT32 interrupts)
{
  INT32 oldmask = splx(7); /* Disable all CPU-Interrupts */
  INT32 newsettings = (ntidir[slot].readback_utcinten1 |= interrupts);
  ntidir[slot].utcsu->w32.utcinten1 = newsettings; /* Set UTCSU-Register UTCINTEN1 */
  splx(oldmask); /* Re-Enable CPU-Interrupts */
  return newsettings;
}

/*-------------------------------------------------------------------------
 * Function utcsu_int_disable1(.) --- Disable UTCSU-Interrupts in UTCSUINTEN1
 * Disable aller im Parameter interrupts mit Bit=1 markierten UTCINTEN1-Interrupts der
 * der UTCSU[slot] und retournieren des neuen Settings.
 *-------------------------------------------------------------------------*/

INT32 utcsu_int_disable1(short slot, INT32 interrupts)
{
  INT32 oldmask = splx(7); /* Disable all CPU-Interrupts */
  INT32 newsettings = (ntidir[slot].readback_utcinten1 &= ~interrupts);
  ntidir[slot].utcsu->w32.utcinten1 = newsettings; /* Set UTCSU-Register UTCINTEN1 */
  splx(oldmask); /* Re-Enable CPU-Interrupts */
  return newsettings;
}

/*-------------------------------------------------------------------------
 * Function utcsu_int_enable2(.) --- Enable UTCSU-Interrupts in UTCSUINTEN2
 * Enable aller im Parameter interrupts mit Bit=1 markierten UTCINTEN2-Interrupts
 * der UTCSU[slot] und retournieren des neuen Settings.
 *-------------------------------------------------------------------------*/

INT32 utcsu_int_enable2(short slot, INT32 interrupts)
{
  INT32 oldmask = splx(7); /* Disable all CPU-Interrupts */
  INT32 newsettings = (ntidir[slot].readback_utcinten2 |= interrupts);
  ntidir[slot].utcsu->w32.utcinten2 = newsettings; /* Set UTCSU-Register UTCINTEN2 */
  splx(oldmask); /* Re-Enable CPU-Interrupts */
  return newsettings;
}


/*-------------------------------------------------------------------------
 * Function utcsu_int_disable2(.) --- Disable UTCSU-Interrupts in UTCSUINTEN2
 * Disable aller im Parameter interrupts mit Bit=1 markierten UTCINTEN2-Interrupts der
 * der UTCSU[slot] und retournieren des neuen Settings.
 *-------------------------------------------------------------------------*/

INT32 utcsu_int_disable2(short slot, INT32 interrupts)
{
  INT32 oldmask = splx(7); /* Disable all CPU-Interrupts */
  INT32 newsettings = (ntidir[slot].readback_utcinten2 &= ~interrupts);
  ntidir[slot].utcsu->w32.utcinten2 = newsettings; /* Set UTCSU-Register UTCINTEN2 */
  splx(oldmask); /* Re-Enable CPU-Interrupts */
  return newsettings;
}



/*-------------------------------------------------------------------------
 * Function pcb_install() --- Install M-Modules
 *-------------------------------------------------------------------------*/

int pcb_install()
```

```
{
  short slot, count, spin;
  INT32 i32_temp;
  NTI_IO_SPACE *nti_io;
  UTCSU *utcsu;
  VMEchip2_LCSR_REGS *vme = VMECHIP2; /* Basis-Adresse MVME-162 VME-Chip */


  count = 0; /* Anzahl erfolgreich installierter Module retour */

  /* Initialize Return-Felder in NTI-directory */
  for (slot=0;slot<6;slot++)
  {
    /* Clear Read-Back Registers for UTCINTEN1, UTCINTEN2 */
    ntidir[slot].readback_utcinten1 = 0;
    ntidir[slot].readback_utcinten2 = 0;

    /* Setup pointers to memory space, iospace and UTCSU */
    ntidir[slot].slot = slot;
    ntidir[slot].memory = (NTI *)(carrier_base + slot * SLOT_MEMSIZE);
    ntidir[slot].utcsu = &(ntidir[slot].memory->utcsu);
    if (slot < 2)
      ntidir[slot].io = (NTI_IO_SPACE *)(carrier_base+IOOFFSET0+slot*SLOT_IOSIZE);
    else
      ntidir[slot].io = (NTI_IO_SPACE *)(carrier_base+IOOFFSET2+(slot-2)*SLOT_IOSIZE);
  }

  /* First, ensure that pSOS's VME A32/D32 address range goes far enough */
  /* (default 0xEFFFFFFF) to be able to access the entire VME_PAGE area */
  i32_temp = vme->master_addr_reg1;
  if ((i32_temp & 0xFF000000) < carrier_base)
  {
    vme->master_addr_reg1 = (i32_temp & 0x0000FFFF) | carrier_base | 0x00FF0000;
  }

  /* Check whether accesses to NTI M-Module work. */
  for (slot=0;slot<6;slot++)
  {
    if (ntidir[slot].installed)
      if (MemAccessible((void *)ntidir[slot].memory))
        ntidir[slot].installed = MemAccessible((void *)ntidir[slot].io);
      else
        ntidir[slot].installed = 0;  /* Something wrong with memory-space */

    /* We clear Local Bus Error of the MVME (MemAccessible) */
    i32_temp = vme->dma_int_ctrl_regs;
    vme->dma_int_ctrl_regs = i32_temp | (1 << 11);
    i32_temp = vme->dma_int_ctrl_regs;

    if (ntidir[slot].installed)
    {
      count++;               /* Eine erfolgreiche Installation mehr ... */
      nti_io = ntidir[slot].io;
      nti_io->int_logic = NTI_DIS_INT_LOGIC; /* Disable NTI interrupts */
      nti_io->vector_base = ntidir[slot].nti_irq_vector;
      nti_io->type_offset = ntidir[slot].nti_type_offset; /* Typefield Offsets */
      nti_io->ts_offsets =  ntidir[slot].nti_ts_offsets; /* TS-Trigger Offsets */

      /* Set UTCSU-signals and force UTCSU reset */
      nti_io->utcsu_misc = NTI_UTCSU_SIGNALS | NTI_MISC_RESET;
      for (spin=0;spin<100;spin++);  /* Wait a bit  ... */
      nti_io->utcsu_misc = NTI_UTCSU_SIGNALS;

      utcsu = ntidir[slot].utcsu;
      utcsu->w32.swreset = 1;        /* Software-reset as well */

      /* Now start the UTCSU-clock */

      utcsu->w32.stepamort = 0;
      utcsu->w32.steppure = ntidir[slot].utcsu_step; /* Register STEP */
      utcsu->w32.amorttimeset = 1; /* Das Ganze aktivieren ... */
      utcsu->w32.startamort = 1;   /* W"urde INTT ausl"osen ... */

      /* Preset Time */
      utcsu->w32.msset = ntidir[slot].utcsu_steplow & 0xFF000000; /* Register STEP_LOW dabei !*/
      utcsu->w32.tsset = 0x00000000;
      utcsu->w32.usset = 0x00000000;
      utcsu->w32.ntpset = 1;         /* Los ... */

      /* Setup UTCCONF-registers */
      utcsu->w32.utcconf1 = ntidir[slot].utcsu_conf1;
      utcsu->w32.utcconf2 = ntidir[slot].utcsu_conf2;

      /* Clear and disable all UTCSU int-soures */
      utcsu->w32.utcinten1 = 0x00000000;
      utcsu->w32.utcinten2 = 0x00000000;
      utcsu->w32.utcintclear1 = 0xFFFFFFFF;
      utcsu->w32.utcintclear2 = 0xFFFFFFFF;
```

```
      nti_io->int_logic = NTI_EN_INT_LOGIC; /* UTCSU-Int's are still off */

    } /* if(installed) */
  } /* for(slot) */

  /* Installieren der ISRs in den Vektoren */

#if ISR_NTI0_INST
  if (ntidir[0].installed)
  { SysSetVector(ntidir[0].nti_irq_vector, isr_nti_illegal, NI_WRAPPER);
    SysSetVector(ntidir[0].nti_irq_vector | (0x01), wrisr_nti0_a, NI_WRAPPER);
    SysSetVector(ntidir[0].nti_irq_vector | (0x02), wrisr_nti0_t, NI_WRAPPER);
    SysSetVector(ntidir[0].nti_irq_vector | (0x03), wrisr_nti0_ta, NI_WRAPPER);
    SysSetVector(ntidir[0].nti_irq_vector | (0x04), wrisr_nti0_n, NI_WRAPPER);
    SysSetVector(ntidir[0].nti_irq_vector | (0x05), wrisr_nti0_na, NI_WRAPPER);
    SysSetVector(ntidir[0].nti_irq_vector | (0x06), wrisr_nti0_nt, NI_WRAPPER);
    SysSetVector(ntidir[0].nti_irq_vector | (0x07), wrisr_nti0_nta, NI_WRAPPER);
  }
#endif
#if ISR_NTI1_INST
  if (ntidir[1].installed)
  { SysSetVector(ntidir[1].nti_irq_vector, isr_nti_illegal, NI_WRAPPER);
    SysSetVector(ntidir[1].nti_irq_vector | (0x01), wrisr_nti1_a, NI_WRAPPER);
    SysSetVector(ntidir[1].nti_irq_vector | (0x02), wrisr_nti1_t, NI_WRAPPER);
    SysSetVector(ntidir[1].nti_irq_vector | (0x03), wrisr_nti1_ta, NI_WRAPPER);
    SysSetVector(ntidir[1].nti_irq_vector | (0x04), wrisr_nti1_n, NI_WRAPPER);
    SysSetVector(ntidir[1].nti_irq_vector | (0x05), wrisr_nti1_na, NI_WRAPPER);
    SysSetVector(ntidir[1].nti_irq_vector | (0x06), wrisr_nti1_nt, NI_WRAPPER);
    SysSetVector(ntidir[1].nti_irq_vector | (0x07), wrisr_nti1_nta, NI_WRAPPER);
  }
#endif
#if ISR_NTI2_INST
  if (ntidir[2].installed)
  { SysSetVector(ntidir[2].nti_irq_vector, isr_nti_illegal, NI_WRAPPER);
    SysSetVector(ntidir[2].nti_irq_vector | (0x01), wrisr_nti2_a, NI_WRAPPER);
    SysSetVector(ntidir[2].nti_irq_vector | (0x02), wrisr_nti2_t, NI_WRAPPER);
    SysSetVector(ntidir[2].nti_irq_vector | (0x03), wrisr_nti2_ta, NI_WRAPPER);
    SysSetVector(ntidir[2].nti_irq_vector | (0x04), wrisr_nti2_n, NI_WRAPPER);
    SysSetVector(ntidir[2].nti_irq_vector | (0x05), wrisr_nti2_na, NI_WRAPPER);
    SysSetVector(ntidir[2].nti_irq_vector | (0x06), wrisr_nti2_nt, NI_WRAPPER);
    SysSetVector(ntidir[2].nti_irq_vector | (0x07), wrisr_nti2_nta, NI_WRAPPER);
  }
#endif
#if ISR_NTI3_INST
  if (ntidir[3].installed)
  { SysSetVector(ntidir[3].nti_irq_vector, isr_nti_illegal, NI_WRAPPER);
    SysSetVector(ntidir[3].nti_irq_vector | (0x01), wrisr_nti3_a, NI_WRAPPER);
    SysSetVector(ntidir[3].nti_irq_vector | (0x02), wrisr_nti3_t, NI_WRAPPER);
    SysSetVector(ntidir[3].nti_irq_vector | (0x03), wrisr_nti3_ta, NI_WRAPPER);
    SysSetVector(ntidir[3].nti_irq_vector | (0x04), wrisr_nti3_n, NI_WRAPPER);
    SysSetVector(ntidir[3].nti_irq_vector | (0x05), wrisr_nti3_na, NI_WRAPPER);
    SysSetVector(ntidir[3].nti_irq_vector | (0x06), wrisr_nti3_nt, NI_WRAPPER);
    SysSetVector(ntidir[3].nti_irq_vector | (0x07), wrisr_nti3_nta, NI_WRAPPER);
  }
#endif
#if ISR_NTI4_INST
  if (ntidir[4].installed)
  { SysSetVector(ntidir[4].nti_irq_vector, isr_nti_illegal, NI_WRAPPER);
    SysSetVector(ntidir[4].nti_irq_vector | (0x01), wrisr_nti4_a, NI_WRAPPER);
    SysSetVector(ntidir[4].nti_irq_vector | (0x02), wrisr_nti4_t, NI_WRAPPER);
    SysSetVector(ntidir[4].nti_irq_vector | (0x03), wrisr_nti4_ta, NI_WRAPPER);
    SysSetVector(ntidir[4].nti_irq_vector | (0x04), wrisr_nti4_n, NI_WRAPPER);
    SysSetVector(ntidir[4].nti_irq_vector | (0x05), wrisr_nti4_na, NI_WRAPPER);
    SysSetVector(ntidir[4].nti_irq_vector | (0x06), wrisr_nti4_nt, NI_WRAPPER);
    SysSetVector(ntidir[4].nti_irq_vector | (0x07), wrisr_nti4_nta, NI_WRAPPER);
  }
#endif
#if ISR_NTI5_INST
  if (ntidir[5].installed)
  { SysSetVector(ntidir[5].nti_irq_vector, isr_nti_illegal, NI_WRAPPER);
    SysSetVector(ntidir[5].nti_irq_vector | (0x01), wrisr_nti5_a, NI_WRAPPER);
    SysSetVector(ntidir[5].nti_irq_vector | (0x02), wrisr_nti5_t, NI_WRAPPER);
    SysSetVector(ntidir[5].nti_irq_vector | (0x03), wrisr_nti5_ta, NI_WRAPPER);
    SysSetVector(ntidir[5].nti_irq_vector | (0x04), wrisr_nti5_n, NI_WRAPPER);
    SysSetVector(ntidir[5].nti_irq_vector | (0x05), wrisr_nti5_na, NI_WRAPPER);
    SysSetVector(ntidir[5].nti_irq_vector | (0x06), wrisr_nti5_nt, NI_WRAPPER);
    SysSetVector(ntidir[5].nti_irq_vector | (0x07), wrisr_nti5_nta, NI_WRAPPER);
  }
#endif


  /* Finally setup VME interrupt service */

  /* Set MVME162-512A Local Bus Interrupt Level */
  vme->int_lvl_reg4 = (vme->int_lvl_reg4 & 0xF0000000) |
                      (NTI_IRQ_LEVEL << (4 * (VME_IRQ_LEVEL - 1)));
```

```
/* VME Interrupt-Level enable (= Level generated by i6360) */
vme->loc_bus_int_en_reg = (vme->loc_bus_int_en_reg & 0xFFFFFF80) |
                          (1 << (VME_IRQ_LEVEL - 1));

i32_temp = vme->vect_base_io_ctrl_reg;
vme->vect_base_io_ctrl_reg = i32_temp | 0x00800000; /* Master-Int. enable */

return count;
}
```

C.10 utcsu.h

```
/************************************************************************
 *
 * pSOS+ Typdefinitionen f"ur die UTCSU
 *
 * Thomas MANDL
 *
 * $Log: utcsu.h,v $
 * Revision 2.0  1999/01/20 13:22:52  s
 * First RCS revision
 *
 * 12.9.1998 U. Schmid
 *   Kosmetische "Anderungen, kleinere Korrekturen
 *
 * 17.2.1997 Thomas MANDL
 *   Erste Version fertiggestellt (UTClient).
 *
 * 18.7.1997 Thomas MANDL
 *   Strukturen f"ur 8-Bit WRITE und 16-/8-Bit READ erg"anzt. Diese
 *   werden f"ur den funktionellen Test der UTCSU ben"otigt.
 *----------------------------------------------------------------------
 *
 * Dieses Modul stellt UTCSU-Typdefinitionen f"ur 32-/16-Bit WRITE-
 * und 32-Bit READ-Zugriffe  f"ur den UTClient zur Verf"ugung.
 *
 * Konventionen:
 *   1) Die Register sind wenn m"oglich so wie in der Loy-Diss. (S.74f)
 *      bezeichnet. Statt Gro"sbuchstaben werden hier nur Kleinbuchstaben
 *      verwendet.
 *   2) Die Typdefinitionen sind f"ur Big-Endian CPU geschrieben.
 *
 ***********************************************************************/
#ifndef _UTCSU_H
#define _UTCSU_H

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * UTCSU - Typdefinitionen
 * f"ur WRITE and READ mit 32-/16-/8-Bit Datenbreite
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/*--- UTCSU - Registerstruktur f"ur Write 32                            ---*/
typedef volatile struct
{                       /* Addr  Register WRITE-Function               Bits */
/* SSU */               /* ------------------------------------------------ */
  INT32 dutyah;         /* 1x0h  Duty Timer A High                   [31:0] */
  INT32 dutyal;         /* 1x4h  Duty Timer A Low + Enable  EN[16] DTL[15:0] */
  INT32 dutybh;         /* 1x8h  Duty Timer B High                   [31:0] */
  INT32 dutybl;         /* 1xCh  Duty Timer B Low + Enable  EN[16] DTL[15:0] */
  INT32 unused[4];      /*  |                                               */
} SSU_W32;              /*  |-- x = {4|6|8|A|C|E}                           */

typedef volatile struct
{                       /* Addr  Register WRITE-Function               Bits */
/* LTU */               /* ------------------------------------------------ */
  INT32 msset;          /* 000h  Step Low + Macro Stamp   SL[31:24] MS[23:0] */
  INT32 tsset;          /* 004h  Time Stamp                          [31:0] */
  INT32 usset;          /* 008h  Micro Stamp                         [31:0] */
  INT32 ntpset;         /* 00Ch  NTP-Time Set                         [7:0] */
  INT32 stepamort;      /* 010h  Step Amort. Phase                   [31:0] */
  INT32 steppure;       /* 014h  Step Pure Phase                     [31:0] */
  INT32 amorttimeset;   /* 018h  Set Amortization Time               [31:0] */
  INT32 startamort;     /* 01Ch  Start Amortization                  [31:0] */
/* ACU */               /*                                                  */
  INT32 lambdap;        /* 020h  positiv Lambda       AMOR[31:16] PURE[15:0] */
  INT32 lambdan;        /* 024h  negativ Lambda       AMOR[31:16] PURE[15:0] */
  INT32 statepset;      /* 028h  STATEPSET                           [31:0] */
  INT32 statenset;      /* 02Ch  STATENSET                           [31:0] */
  INT32 bound;          /* 030h  pos. and neg. BOUND   NEG[31:16]  POS[15:0] */
  INT32 alphaset;       /* 034h  ALPHASET                            [31:0] */
  INT32 alphapnset;     /* 038h  ALPHAPNSET                           [7:0] */
  INT32 acu_unused;     /*                                                  */
/* CLU */               /*                                                  */
  INT32 swreset;        /* 040h  Software RESET                       [7:0] */
  INT32 clu_unused[7];  /*                                                  */
/* ITU */               /*                                                  */
  INT32 utcconf1;       /* 060h  Interrupt Config. Reg. 1            [31:0] */
  INT32 utcconf2;       /* 064h  Interrupt Config. Reg. 2            [31:0] */
  INT32 utcinten1;      /* 068h  Interrupt Enable  Reg. 1            [31:0] */
  INT32 utcinten2;      /* 06Ch  Interrupt Enable  Reg. 2            [31:0] */
  INT32 utcintclear1;   /* 070h  Interrupt Clear   Reg. 1            [31:0] */
  INT32 utcintclear2;   /* 074h  Interrupt Clear   Reg. 2            [31:0] */
  INT32 itu_unused[2];  /*                                                  */
/* BTU */               /*                                                  */
  INT32 testsel;        /* 080h  Test Select                         [15:0] */
  INT32 testgate;       /* 084h  Test Gate Impuls                     [7:0] */
  INT32 btu_unused[6];  /*                                                  */
```

```
/* SNU */              /*                                                */
  INT32 swsnap;        /* 0A0h  Software Snapshot                 [31:0] */
  INT32 snu_dutyh;     /* 0A4h  Duty Timer High Preload           [31:0] */
  INT32 snu_dutyl;     /* 0A8h  Duty Timer Low + Enable    EN[16] DTL[15:0] */
  INT32 snu_unused[5]; /*                                                */
/* APU */              /*                                                */
  INT32 apu_dutyh;     /* 0C0h  Duty Timer High Preload           [31:0] */
  INT32 apu_dutyl;     /* 0C4h  Duty Timer Low + Enable    EN[16] DTL[15:0] */
  INT32 appsel;        /* 0C8h  Application Select                 [2:0] */
  INT32 appclear;      /* 0CCh  Application Clear                  [7:0] */
  INT32 apu_unused[4]; /*                                                */
/* GPU */              /*                                                */
  INT32 gpu1_unused[8]; /*                                               */
  INT32 gpu2_unused[8]; /*                                               */
  INT32 gpu3_unused[8]; /*                                               */
/* SSU */              /*                                                */
  SSU_W32 ssu[6];      /* 140h Synchronisation Subnet Unit 1             */
                       /* 160h Synchronisation Subnet Unit 2             */
                       /* 180h Synchronisation Subnet Unit 3             */
                       /* 1A0h Synchronisation Subnet Unit 4             */
                       /* 1C0h Synchronisation Subnet Unit 5             */
                       /* 1E0h Synchronisation Subnet Unit 6             */
} UTCSU_WRITE32;


/*--- UTCSU - Registerstruktur f"ur Write 16                          ---*/
typedef volatile struct
{                      /* Addr  Register WRITE-Function [WORD=16Bit]   Bits */
/* SSU */              /* ------------------------------------------------ */
  INT16 dutyah[2];     /* 1x0h  Duty Timer A High         0[31:16] 1[15:0] */
  INT16 ssu_ea;        /* 1x4h  Enable Duty Timer A                    [0] */
  INT16 dutyal;        /* 1x6h  Duty Timer A Low                    [15:0] */
  INT16 dutybh[2];     /* 1x8h  Duty Timer B High         0[31:16] 1[15:0] */
  INT16 ssu_eb;        /* 1xCh  Enable Duty Timer B                    [0] */
  INT16 dutybl;        /* 1xEh  Duty Timer B Low                    [15:0] */
  INT16 unused[8];     /*  |                                               */
} SSU_W16;             /*  |-- x = {4|6|8|A|C|E}                           */

typedef volatile struct
{                      /* Addr  Register WRITE-Function                Bits */
/* LTU */              /* ------------------------------------------------ */
  INT16 msset[2];      /* 000h  Step Low + Macro Stamp   SL[31:24] MS[23:0] */
  INT16 tsset[2];      /* 004h  Time Stamp                0[31:16] 1[15:0] */
  INT16 usset[2];      /* 008h  Micro Stamp               0[31:16] 1[15:0] */
  INT16 ltu_unused1;   /*                                                  */
  INT16 ntpset;        /* 00Eh  NTP-Time Set                         [7:0] */
  INT16 stepamort[2];  /* 010h  Step Amort. Phase         0[31:16] 1[15:0] */
  INT16 steppure[2];   /* 014h  Step Pure Phase           0[31:16] 1[15:0] */
  INT16 amorttimeset[2];/* 018h  Set Amortization Time     0[31:16] 1[15:0] */
  INT16 ltu_unused2;   /*                                                  */
  INT16 startamort;    /* 01Eh  Start Amortization                  [15:0] */
/* ACU */              /*                                                  */
  INT16 lambdapamort;  /* 020h  positive Lambda - Amort. Phase      [15:0] */
  INT16 lambdappure;   /* 022h  positive Lambda - Pure Phase        [15:0] */
  INT16 lambdanamort;  /* 024h  negativ Lambda - Amort. Phase       [15:0] */
  INT16 lambdanpure;   /* 026h  negativ Lambda - Pure Phase         [15:0] */
  INT16 statepset[2];  /* 028h  STATEPSET                 0[31:16] 1[15:0] */
  INT16 statenset[2];  /* 02Ch  STATENSET                 0[31:16] 1[15:0] */
  INT16 boundp;        /* 030h  positive Bound                      [15:0] */
  INT16 boundn;        /* 032h  negativ Bound                       [15:0] */
  INT16 alphaset[2];   /* 034h  ALPHASET                  0[31:16] 1[15:0] */
  INT16 acu_unused1;   /*                                                  */
  INT16 alphapnset;    /* 03Ah  ALPHAPNSET                           [7:0] */
  INT16 acu_unused2[2]; /*                                                 */
/* CLU */              /*                                                  */
  INT16 clu_unused1;   /*                                                  */
  INT16 swreset;       /* 042h  Software RESET                       [7:0] */
  INT16 clu_unused2[14];/*                                                 */
/* ITU */              /*                                                  */
  INT16 utcconf1[2];   /* 060h  Interrupt Config. Reg. 1  0[31:16] 1[15:0] */
  INT16 utcconf2[2];   /* 064h  Interrupt Config. Reg. 2  0[31:16] 1[15:0] */
  INT16 utcinten1[2];  /* 068h  Interrupt Enable  Reg. 1  0[31:16] 1[15:0] */
  INT16 utcinten2[2];  /* 06Ch  Interrupt Enable  Reg. 2  0[31:16] 1[15:0] */
  INT16 utcintclear1[2];/* 070h  Interrupt Clear   Reg. 1  0[31:16] 1[15:0] */
  INT16 utcintclear2[2];/* 074h  Interrupt Clear   Reg. 2  0[31:16] 1[15:0] */
  INT16 itu_unused[4]; /*                                                  */
/* BTU */              /*                                                  */
  INT16 btu_unused1;   /*                                                  */
  INT16 testsel;       /* 082h  Test Select                         [15:0] */
  INT16 btu_unused2;   /*                                                  */
  INT16 testgate;      /* 086h  Test Gate Impuls                     [7:0] */
  INT16 btu_unused3[12];/*                                                 */
/* SNU */              /*                                                  */
  INT16 snu_unused1;   /*                                                  */
  INT16 swsnap;        /* 0A2h  Software Snapshot                   [15:0] */
  INT16 snu_dutyh[2];  /* 0A4h  Duty Timer High Preload   0[31:16] 1[15:0] */
  INT16 snu_e;         /* 0A8h  Enable Duty Timer                      [0] */
  INT16 snu_dutyl;     /* 0AAh  Duty Timer Low                      [15:0] */
```

```
  INT16 snu_unused2[10];/*                                                  */
/* APU */              /*                                                  */
  INT16 apu_dutyh[2];  /* 0C0h  Duty Timer High Preload    0[31:16] 1[15:0] */
  INT16 apu_e;         /* 0C4h  Enable Duty Timer                       [0] */
  INT16 apu_dutyl;     /* 0C6h  Duty Timer Low                       [15:0] */
  INT16 apu_unused1;   /*                                                  */
  INT16 appsel;        /* 0C8h  Application Select                    [2:0] */
  INT16 apu_unused2;   /*                                                  */
  INT16 appclear;      /* 0CCh  Application Clear                     [7:0] */
  INT16 apu_unused3[8]; /*                                                 */
/* GPU */              /*                                                  */
  INT16 gpu1_unused[16];/*                                                 */
  INT16 gpu2_unused[16];/*                                                 */
  INT16 gpu3_unused[16];/*                                                 */
/* SSU */              /*                                                  */
  SSU_W16 ssu[6];      /* 140h Synchronisation Subnet Unit 1               */
                       /* 160h Synchronisation Subnet Unit 2               */
                       /* 180h Synchronisation Subnet Unit 3               */
                       /* 1A0h Synchronisation Subnet Unit 4               */
                       /* 1C0h Synchronisation Subnet Unit 5               */
                       /* 1E0h Synchronisation Subnet Unit 6               */
} UTCSU_WRITE16;

/*--- UTCSU - Registerstruktur f"ur Write 8                              ---*/
typedef volatile struct
{                      /* Addr  Register WRITE-Function [WORD=16Bit]   Bits */
/* SSU */              /* ------------------------------------------------- */
  INT8 dutyah[4];      /* 1x0h  Duty Timer A High       0[31:24] ... 3[7:0] */
  INT8 unused1;        /*                                                  */
  INT8 ssu_ea;         /* 1x5h  Enable Duty Timer A                     [0] */
  INT8 dutyal[2];      /* 1x6h  Duty Timer A Low                0[15:8] 1[7:0] */
  INT8 dutybh[4];      /* 1x8h  Duty Timer B High       0[31:16] ... 3[7:0] */
  INT8 unused2;        /*                                                  */
  INT8 ssu_eb;         /* 1xDh  Enable Duty Timer B                     [0] */
  INT8 dutybl[2];      /* 1xEh  Duty Timer B Low                0[15:8] 1[7:0] */
  INT8 unused[16];     /*  |                                               */
} SSU_W8;              /*  |-- x = {4|6|8|A|C|E}                           */

typedef volatile struct
{                      /* Addr  Register WRITE-Function                Bits */
/* LTU */              /* ------------------------------------------------- */
  INT8 steplow;        /* 000h  Steplow                               [7:0] */
  INT8 msset[3];       /* 001h  Macro Stamp             0[23:16] ... 2[7:0] */
  INT8 tsset[4];       /* 004h  Time Stamp              0[31:24] ... 3[7:0] */
  INT8 usset[4];       /* 008h  Micro Stamp             0[31:24] ... 3[7:0] */
  INT8 ltu_unused1[3]; /*                                                  */
  INT8 ntpset;         /* 00Fh  NTP-Time Set                          [7:0] */
  INT8 stepamort[4];   /* 010h  Step Amort. Phase       0[31:24] ... 3[7:0] */
  INT8 steppure[4];    /* 014h  Step Pure Phase         0[31:24] ... 3[7:0] */
  INT8 amorttimeset[4]; /* 018h  Set Amortization Time   0[31:24] ... 3[7:0] */
  INT8 ltu_unused2[3]; /*                                                  */
  INT8 startamort;     /* 01Fh  Start Amortization                    [7:0] */
/* ACU */              /*                                                  */
  INT8 lambdapamort[2]; /* 020h  pos. Lambda-Amort. Phase     0[15:8] 1[7:0] */
  INT8 lambdappure[2]; /* 022h  pos. Lambda-Pure Phase      0[15:8] 1[7:0] */
  INT8 lambdanamort[2]; /* 024h  neg. Lambda-Amort. Phase     0[15:8] 1[7:0] */
  INT8 lambdanpure[2]; /* 026h  neg. Lambda-Pure Phase      0[15:8] 1[7:0] */
  INT8 statepset[4];   /* 028h  STATEPSET               0[31:24] ... 3[7:0] */
  INT8 statenset[4];   /* 02Ch  STATENSET               0[31:24] ... 3[7:0] */
  INT8 boundp[2];      /* 030h  positive Bound                0[15:8] 1[7:0] */
  INT8 boundn[2];      /* 032h  negativ Bound                 0[15:8] 1[7:0] */
  INT8 alphaset[4];    /* 034h  ALPHASET                0[31:24] ... 3[7:0] */
  INT8 acu_unused1[3]; /*                                                  */
  INT8 alphapnset;     /* 03Bh  ALPHAPNSET                            [7:0] */
  INT8 acu_unused2[4]; /*                                                  */
/* CLU */              /*                                                  */
  INT8 clu_unused1[3]; /*                                                  */
  INT8 swreset;        /* 043h  Software RESET                        [7:0] */
  INT8 clu_unused2[28]; /*                                                 */
/* ITU */              /*                                                  */
  INT8 utcconf1[4];    /* 060h  Interrupt Config. Reg.1 0[31:24] ... 3[7:0] */
  INT8 utcconf2[4];    /* 064h  Interrupt Config. Reg.2 0[31:24] ... 3[7:0] */
  INT8 utcinten1[4];   /* 068h  Interrupt Enable  Reg.1 0[31:24] ... 3[7:0] */
  INT8 utcinten2[4];   /* 06Ch  Interrupt Enable  Reg.2 0[31:24] ... 3[7:0] */
  INT8 utcintclear1[4]; /* 070h  Interrupt Clear   Reg.1 0[31:24] ... 3[7:0] */
  INT8 utcintclear2[4]; /* 074h  Interrupt Clear   Reg.2 0[31:24] ... 3[7:0] */
  INT8 itu_unused[8];  /*                                                  */
/* BTU */              /*                                                  */
  INT8 btu_unused1[2]; /*                                                  */
  INT8 testsel[2];     /* 082h  Test Select                   0[15:8] 1[7:0] */
  INT8 btu_unused2[3]; /*                                                  */
  INT8 testgate;       /* 087h  Test Gate Impuls                      [7:0] */
  INT8 btu_unused3[24]; /*                                                 */
/* SNU */              /*                                                  */
  INT8 snu_unused1[3]; /*                                                  */
  INT8 swsnap;         /* 0A3h  Software Snapshot                     [7:0] */
  INT8 snu_dutyh[4];   /* 0A4h  Duty Timer High Preload 0[31:24] ... 3[7:0] */
  INT8 snu_unused2;    /*                                                  */
```

```
  INT8 snu_e;           /* 0A9h  Enable Duty Timer                        [0] */
  INT8 snu_dutyl[2];    /* 0AAh  Duty Timer Low               0[15:8] 1[7:0] */
  INT8 snu_unused3[20]; /*                                                   */
/* APU */               /*                                                   */
  INT8 apu_dutyh[4];    /* 0C0h  Duty Timer High Preload 0[31:24] ... 3[7:0] */
  INT8 apu_unused1;     /*                                                   */
  INT8 apu_e;           /* 0C5h  Enable Duty Timer                        [0] */
  INT8 apu_dutyl[2];    /* 0C6h  Duty Timer Low               0[15:8] 1[7:0] */
  INT8 apu_unused2[3];  /*                                                   */
  INT8 appsel;          /* 0CBh  Application Select                     [2:0] */
  INT8 apu_unused3[3];  /*                                                   */
  INT8 appclear;        /* 0CFh  Application Clear                      [7:0] */
  INT8 apu_unused4[16]; /*                                                   */
/* GPU */               /*                                                   */
  INT8 gpu1_unused[32]; /*                                                   */
  INT8 gpu2_unused[32]; /*                                                   */
  INT8 gpu3_unused[32]; /*                                                   */
/* SSU */               /*                                                   */
  SSU_W8 ssu[6];        /* 140h Synchronisation Subnet Unit 1                */
                        /* 160h Synchronisation Subnet Unit 2                */
                        /* 180h Synchronisation Subnet Unit 3                */
                        /* 1A0h Synchronisation Subnet Unit 4                */
                        /* 1C0h Synchronisation Subnet Unit 5                */
                        /* 1E0h Synchronisation Subnet Unit 6                */
} UTCSU_WRITE8;


/*--- UTCSU - Registerstruktur f"ur Read 32                              ---*/
typedef volatile struct
{                       /* Addr  Register READ-Function                 Bits */
/* GPU */               /* ------------------------------------------------- */
  INT32 msgps;          /* xx0h  S+CS+Mac.Stamp GPS S[31] CS[30:24] MS[23:0] */
  INT32 tsgps;          /* xx4h  Time Stamp GPS                        [31:0] */
  INT32 unused[6];      /*  |                                                */
} GPU_R32;              /*  |-- xx = {0E|10|12}                              */

typedef volatile struct
{                       /* Addr  Register READ-Function                 Bits */
/* SSU */               /* ------------------------------------------------- */
  INT32 msrcv;          /* 1x0h  CS+Mac.Stamp Rec.  S[31] CS[30:24] MS[23:0] */
  INT32 tsrcv;          /* 1x4h  Time Stamp Receive                    [31:0] */
  INT32 msxmt;          /* 1x8h  Macro Stamp Trans. S[31] CS[30:24] MS[23:0] */
  INT32 tsxmt;          /* 1xCh  Time Stamp Transmit                   [31:0] */
  INT32 accxmt;         /* 1y0h  Accuracy X Transmit        N[31:16] P[15:0] */
  INT32 accnpnplxmt;    /* 1y4h  Acc.NP low N[31:24] P[23:16] N[15:8] P[7:0] */
  INT32 accnpnphxmt;    /* 1y8h  Acc.NP hi  N[31:24] P[23:16] N[15:8] P[7:0] */
  INT32 unused;         /*  |                                                */
} SSU_R32;              /*  |-- x = {4|6|8|A|C|E}, y = {5|7|9|B|D|F}         */

typedef volatile struct
{                       /* Addr  Register READ-Function                 Bits */
/* LTU */               /* ------------------------------------------------- */
  INT32 msget;          /* 000h  Check Sum + Macro Stamp  CS[31:24] MS[23:0] */
  INT32 tsgetl;         /* 004h  Time Stamp Get with Latching          [31:0] */
  INT32 usget;          /* 008h  Micro Stamp Get                       [31:0] */
  INT32 tsget;          /* 00Ch  Time Stamp Get                        [31:0] */
  INT32 nsget;          /* 010h  Nano Stamp Get                         [2:0] */
  INT32 ltu_unused;     /*                                                   */
  INT32 amorttimegetl;  /* 018h  Amort.Time Get with Latching          [31:0] */
  INT32 amorttimeget;   /* 01Ch  Amort.Time Get                        [31:0] */
/* ACU */               /*                                                   */
  INT32 stategetl;      /* 020h  Accuracy Get with Latching N[31:16] P[15:0] */
  INT32 stateget;       /* 024h  Accuracy Get               N[31:16] P[15:0] */
  INT32 alphapgetl;     /* 028h  Full pos. Acc. Get with Latching      [31:0] */
  INT32 alphangetl;     /* 02Ch  Full neg. Acc. Get with Latching      [31:0] */
  INT32 alphapget;      /* 030h  Full pos. Accuracy Get                [31:0] */
  INT32 alphanget;      /* 034h  Full neg. Accuracy Get                [31:0] */
  INT32 nalphaget;      /* 038h  Fractional Accuracy Get    N[31:16] P[15:0] */
  INT32 acu_unused;     /*                                                   */
/* CLU */               /*                                                   */
  INT32 clu_unused[8];  /*                                                   */
/* ITU */               /*                                                   */
  INT32 id;             /* 060h  Get ID                                [31:0] */
  INT32 utcstat1;       /* 064h  Status  Reg. 1                        [31:0] */
  INT32 utcstat2;       /* 068h  Status  Reg. 2                        [31:0] */
  INT32 utcconf1;       /* 06Ch  Interrupt Config. Reg. 1              [31:0] */
  INT32 utcconf2;       /* 070h  Interrupt Config. Reg. 2              [31:0] */
  INT32 utcintstat1;    /* 074h  Interrupt Status  Reg. 1              [31:0] */
  INT32 utcintstat2;    /* 078h  Interrupt Status  Reg. 2              [31:0] */
  INT32 itu_unused;     /*                                                   */
/* BTU */               /*                                                   */
  INT32 mssig;          /* 080h  Macro Portion MISR signature          [23:0] */
  INT32 tssig;          /* 084h  Tiny Portion MISR signature           [31:0] */
  INT32 accsig;         /* 088h  Accuracy MISR signature               [31:0] */
  INT32 mssum;          /* 08Ch  Macro Portion Blocksum                [23:0] */
  INT32 tssum;          /* 090h  Tiny Portion Blocksum                 [31:0] */
  INT32 accsum;         /* 094h  Accuracy Blocksum                     [31:0] */
  INT32 purephase;      /* 098h  Pure Phase Counter                    [31:0] */
```

```
  INT32 btu_unused;     /*                                                */
/* SNU */               /*                                                */
  INT32 mssnu;          /* 0A0h  CS+Macro Stamp Snapshot  CS[31:24] MS[23:0] */
  INT32 tssnu;          /* 0A4h  Time Stamp Snapshot                  [31:0] */
  INT32 accsnu;         /* 0A8h  Accuracy N/P Snapshot      N[31:16] P[15:0] */
  INT32 snu_unused[5];  /*                                                */
/* APU */               /*                                                */
  INT32 msapp;          /* 0C0h  CS+Macro Stamp Applic.   CS[31:24] MS[23:0] */
  INT32 tsapp;          /* 0C4h  Time Stamp Application               [31:0] */
  INT32 accapp;         /* 0C8h  Accuracy N/P Application   N[31:16] P[15:0] */
  INT32 msappx;         /* 0CCh  CS+Macro Stamp Applic.X  CS[31:24] MS[23:0] */
  INT32 tsappx;         /* 0D0h  Time Stamp Application X             [31:0] */
  INT32 accappx;        /* 0D4h  Accuracy N/P Application X N[31:16] P[15:0] */
  INT32 apu_unused[2];  /*                                                */
/* GPU */               /*                                                */
  GPU_R32 gpu[3];       /* 0E0h GPS Unit 1                                */
                        /* 100h GPS Unit 2                                */
                        /* 120h GPS Unit 3                                */
/* SSU */               /*                                                */
  SSU_R32 ssu[6];       /* 140h Synchronisation Subnet Unit 1             */
                        /* 160h Synchronisation Subnet Unit 2             */
                        /* 180h Synchronisation Subnet Unit 3             */
                        /* 1A0h Synchronisation Subnet Unit 4             */
                        /* 1C0h Synchronisation Subnet Unit 5             */
                        /* 1E0h Synchronisation Subnet Unit 6             */
} UTCSU_READ32;

/*--- UTCSU - Registerstruktur f"ur Read 16                            ---*/
typedef volatile struct
{                       /* Addr  Register READ-Function              Bits */
/* GPU */               /* ------------------------------------------------ */
  INT16 msgps[2];       /* xx0h  S+CS+Mac.Stamp GPS S[31] CS[30:24] MS[23:0] */
  INT16 tsgps[2];       /* xx4h  Time Stamp GPS             0[31:16] 1[15:0] */
  INT16 unused[12];     /*  |                                             */
} GPU_R16;              /*  |-- xx = {0E|10|12}                           */

typedef volatile struct
{                       /* Addr  Register READ-Function              Bits */
/* SSU */               /* ------------------------------------------------ */
  INT16 msrcv[2];       /* 1x0h  CS+Macro Stamp Receive   CS[31:24] MS[23:0] */
  INT16 tsrcv[2];       /* 1x4h  Time Stamp Receive         0[31:16] 1[15:0] */
  INT16 msxmt[2];       /* 1x8h  Macro Stamp Transmits    CS[31:24] MS[23:0] */
  INT16 tsxmt[2];       /* 1xCh  Time Stamp Transmit        0[31:16] 1[15:0] */
  INT16 accnxmt;        /* 1y0h  Accuracy N Transmit                  [15:0] */
  INT16 accpxmt;        /* 1y2h  Accuracy P Transmit                  [15:0] */
  INT16 accnpnplxmt[2]; /* 1y4h  Accuracy N and P low         N[15:8] P[7:0] */
  INT16 accnpnphxmt[2]; /* 1y8h  Accuracy N and P high        N[15:8] P[7:0] */
  INT16 unused[2];      /*  |                                             */
} SSU_R16;              /*  |-- x = {4|6|8|A|C|E}, y = {5|7|9|B|D|F}      */

typedef volatile struct
{                       /* Addr  Register READ-Function              Bits */
/* LTU */               /* ------------------------------------------------ */
  INT16 msget[2];       /* 000h  Check Sum + Macro Stamp  CS[31:24] MS[23:0] */
  INT16 tsgetl[2];      /* 004h  Time Stamp Get with Latching         [31:0] */
  INT16 usget[2];       /* 008h  Micro Stamp Get                      [31:0] */
  INT16 tsget[2];       /* 00Ch  Time Stamp Get                       [31:0] */
  INT16 ltu_unused1;    /*                                                */
  INT16 nsget;          /* 012h  Nano Stamp Get                        [2:0] */
  INT16 ltu_unused2[2]; /*                                                */
  INT16 amorttimegetl[2];/*018h  Amort.Time Get with Latching         [31:0] */
  INT16 amorttimeget[2];/* 01Ch  Amort.Time Get                       [31:0] */
/* ACU */               /*                                                */
  INT16 statengetl;     /* 020h  Neg. Accuracy Get with Latching      [15:0] */
  INT16 statepgetl;     /* 022h  Pos. Accuracy Get with Latching      [15:0] */
  INT16 statenget;      /* 024h  Neg. Accuracy Get                    [15:0] */
  INT16 statepget;      /* 026h  Pos. Accuracy Get                    [15:0] */
  INT16 alphapgetl[2];  /* 028h  Full pos. Acc. Get with Latching     [31:0] */
  INT16 alphangetl[2];  /* 02Ch  Full neg. Acc. Get with Latching     [31:0] */
  INT16 alphapget[2];   /* 030h  Full pos. Accuracy Get               [31:0] */
  INT16 alphanget[2];   /* 034h  Full neg. Accuracy Get               [31:0] */
  INT16 nalphanget;     /* 038h  Fractional neg. Accuracy Get         [15:0] */
  INT16 nalphapget;     /* 03Ah  Fractional pos. Accuracy Get         [15:0] */
  INT16 acu_unused[2];  /*                                                */
/* CLU */               /*                                                */
  INT16 clu_unused[16]; /*                                                */
/* ITU */               /*                                                */
  INT16 id[2];          /* 060h  Get ID                     0[31:16] 1[15:0] */
  INT16 utcstat1[2];    /* 064h  Status  Reg. 1             0[31:16] 1[15:0] */
  INT16 utcstat2[2];    /* 068h  Status  Reg. 2             0[31:16] 1[15:0] */
  INT16 utcconf1[2];    /* 06Ch  Interrupt Config. Reg. 1   0[31:16] 1[15:0] */
  INT16 utcconf2[2];    /* 070h  Interrupt Config. Reg. 2   0[31:16] 1[15:0] */
  INT16 utcintstat1[2]; /* 074h  Interrupt Status  Reg. 1   0[31:16] 1[15:0] */
  INT16 utcintstat2[2]; /* 078h  Interrupt Status  Reg. 2   0[31:16] 1[15:0] */
  INT16 itu_unused[2];  /*                                                */
/* BTU */               /*                                                */
  INT16 mssig[2];       /* 080h  Macro Portion MISR sign.   0[23:16] 1[15:0] */
  INT16 tssig[2];       /* 084h  Tiny Portion MISR sign.    0[31:16] 1[15:0] */
```

```
    INT16 accsig[2];      /* 088h  Accuracy MISR sign.       0[31:16] 1[15:0] */
    INT16 mssum[2];       /* 08Ch  Macro Portion Blocksum    0[23:16] 1[15:0] */
    INT16 tssum[2];       /* 090h  Tiny Portion Blocksum     0[31:16] 1[15:0] */
    INT16 accsum[2];      /* 094h  Accuracy Blocksum         0[31:16] 1[15:0] */
    INT16 purephase[2];   /* 098h  Pure Phase Counter        0[31:16] 1[15:0] */
    INT16 btu_unused[2];  /*                                                  */
  /* SNU */               /*                                                  */
    INT16 mssnu[2];       /* 0A0h  CS+Macro Stamp Snapshot  CS[31:24] MS[23:0] */
    INT16 tssnu[2];       /* 0A4h  Time Stamp Snapshot       0[31:16] 1[15:0] */
    INT16 accnsnu;        /* 0A8h  Accuracy N Snapshot                 [15:0] */
    INT16 accpsnu;        /* 0AAh  Accuracy P Snapshot                 [15:0] */
    INT16 snu_unused[10]; /*                                                  */
  /* APU */               /*                                                  */
    INT16 msapp[2];       /* 0C0h  CS+Macro Stamp Applic.   CS[31:24] MS[23:0] */
    INT16 tsapp[2];       /* 0C4h  Time Stamp Application    0[31:16] 1[15:0] */
    INT16 accnapp;        /* 0C8h  Accuracy N Application              [15:0] */
    INT16 accpapp;        /* 0CAh  Accuracy P Application              [15:0] */
    INT16 msappx[2];      /* 0CCh  CS+Macro Stamp Applic.X  CS[31:24] MS[23:0] */
    INT16 tsappx[2];      /* 0D0h  Time Stamp Application X  0[31:16] 1[15:0] */
    INT16 accnappx;       /* 0D4h  Accuracy N Application X            [15:0] */
    INT16 accpappx;       /* 0D6h  Accuracy P Application X            [15:0] */
    INT16 apu_unused[4];  /*                                                  */
  /* GPU */               /*                                                  */
    GPU_R16 gpu[3];       /* 0E0h GPS Unit 1                                  */
                          /* 100h GPS Unit 2                                  */
                          /* 120h GPS Unit 3                                  */
  /* SSU */               /*                                                  */
    SSU_R16 ssu[6];       /* 140h Synchronisation Subnet Unit 1               */
                          /* 160h Synchronisation Subnet Unit 2               */
                          /* 180h Synchronisation Subnet Unit 3               */
                          /* 1A0h Synchronisation Subnet Unit 4               */
                          /* 1C0h Synchronisation Subnet Unit 5               */
                          /* 1E0h Synchronisation Subnet Unit 6               */
  } UTCSU_READ16;

  /*--- UTCSU - Registerstruktur f"ur Read 8                              ---*/
  typedef volatile struct
  {                       /* Addr  Register READ-Function               Bits */
  /* GPU */               /* ------------------------------------------------ */
    INT8 cs;              /* xx0h  Status+CS for GPU             S[7] CS[6:0] */
    INT8 msgps[3];        /* xx1h  Macro Stamp GPS        0[23:16] ... 2[7:0] */
    INT8 tsgps[4];        /* xx4h  Time Stamp GPS         0[31:24] ... 3[7:0] */
    INT8 unused[24];      /*  |                                               */
  } GPU_R8;               /*  |-- xx = {0E|10|12}                             */

  typedef volatile struct
  {                       /* Addr  Register READ-Function               Bits */
  /* SSU */               /* ------------------------------------------------ */
    INT8 csrcv;           /* 1x0h  CS for RCV-TS                       0[7:0] */
    INT8 msrcv[3];        /* 1x1h  Macro Stamp Receive    0[23:16] ... 2[7:0] */
    INT8 tsrcv[4];        /* 1x4h  Time Stamp Receive     0[31:24] ... 3[7:0] */
    INT8 csxmt;           /* 1x8h  CS for XMT-TS                       0[7:0] */
    INT8 msxmt[3];        /* 1x9h  Macro Stamp Transmits  0[23:16] ... 2[7:0] */
    INT8 tsxmt[4];        /* 1xCh  Time Stamp Transmit    0[31:24] ... 3[7:0] */
    INT8 accnxmt[2];      /* 1y0h  Accuracy N Transmit         0[15:8] 1[7:0] */
    INT8 accpxmt[2];      /* 1y2h  Accuracy P Transmit         0[15:8] 1[7:0] */
    INT8 accnpnplxmt[4];  /* 1y4h  Accuracy N and P low   0[31:24] ... 3[7:0] */
    INT8 accnpnphxmt[4];  /* 1y8h  Accuracy N and P high  0[31:24] ... 3[7:0] */
    INT8 unused[4];       /*  |                                               */
  } SSU_R8;               /*  |-- x = {4|6|8|A|C|E}, y = {5|7|9|B|D|F}        */

  typedef volatile struct
  {                       /* Addr  Register READ-Function               Bits */
  /* LTU */               /* ------------------------------------------------ */
    INT8 checksum;        /* 000h  Check Sum                            [7:0] */
    INT8 msget[3];        /* 001h  Macro Stamp            0[23:16] ... 2[7:0] */
    INT8 tsgetl[4];       /* 004h  Time Stamp Get w. Latch 0[31:24] ... 3[7:0] */
    INT8 usget[4];        /* 008h  Micro Stamp Get        0[31:24] ... 3[7:0] */
    INT8 tsget[4];        /* 00Ch  Time Stamp Get         0[31:24] ... 3[7:0] */
    INT8 ltu_unused1[3];  /*                                                  */
    INT8 nsget;           /* 013h  Nano Stamp Get                       [2:0] */
    INT8 ltu_unused2[4];  /*                                                  */
    INT8 amorttimegetl[4];/*018h  Amort.Time Get w. Latch 0[31:24] ... 3[7:0] */
    INT8 amorttimeget[4]; /* 01Ch  Amort.Time Get         0[31:24] ... 3[7:0] */
  /* ACU */               /*                                                  */
    INT8 statengetl[2];   /* 020h  Neg. Accuracy Get w. Latch   0[15:8] 1[7:0] */
    INT8 statepgetl[2];   /* 022h  Pos. Accuracy Get w. Latch   0[15:8] 1[7:0] */
    INT8 statenget[2];    /* 024h  Neg. Accuracy Get            0[15:8] 1[7:0] */
    INT8 statepget[2];    /* 026h  Pos. Accuracy Get            0[15:8] 1[7:0] */
    INT8 alphapgetl[4];   /* 028h  Full pos.Acc. Get Latch 0[31:24] ... 3[7:0] */
    INT8 alphangetl[4];   /* 02Ch  Full neg.Acc. Get Latch 0[31:24] ... 3[7:0] */
    INT8 alphapget[4];    /* 030h  Full pos. Accuracy Get  0[31:24] ... 3[7:0] */
    INT8 alphanget[4];    /* 034h  Full neg. Accuracy Get  0[31:24] ... 3[7:0] */
    INT8 nalphanget[2];   /* 038h  Fractional neg. Acc. Get     0[15:8] 1[7:0] */
    INT8 nalphapget[2];   /* 03Ah  Fractional pos. Acc. Get     0[15:8] 1[7:0] */
    INT8 acu_unused[4];   /*                                                  */
  /* CLU */               /*                                                  */
    INT8 clu_unused[32];  /*                                                  */
```

```
/* ITU */              /*                                                */
  INT8 id[4];          /* 060h  Get ID                  0[31:24] ... 3[7:0] */
  INT8 utcstat1[4];    /* 064h  Status  Reg. 1          0[31:24] ... 3[7:0] */
  INT8 utcstat2[4];    /* 068h  Status  Reg. 2          0[31:24] ... 3[7:0] */
  INT8 utcconf1[4];    /* 06Ch  Interrupt Config.Reg 1  0[31:24] ... 3[7:0] */
  INT8 utcconf2[4];    /* 070h  Interrupt Config.Reg.2  0[31:24] ... 3[7:0] */
  INT8 utcintstat1[4]; /* 074h  Interrupt Status Reg.1  0[31:24] ... 3[7:0] */
  INT8 utcintstat2[4]; /* 078h  Interrupt Status Reg.2  0[31:24] ... 3[7:0] */
  INT8 itu_unused[4];  /*                                                */
/* BTU */              /*                                                */
  INT8 mssig[4];       /* 080h  Macro Port. MISR sign.  0[23:24] ... 3[7:0] */
  INT8 tssig[4];       /* 084h  Tiny Port. MISR sign.   0[31:24] ... 3[7:0] */
  INT8 accsig[4];      /* 088h  Accuracy MISR sign.     0[31:24] ... 3[7:0] */
  INT8 mssum[4];       /* 08Ch  Macro Port. Blocksum    0[23:24] ... 3[7:0] */
  INT8 tssum[4];       /* 090h  Tiny Port. Blocksum     0[31:24] ... 3[7:0] */
  INT8 accsum[4];      /* 094h  Accuracy Blocksum       0[31:24] ... 3[7:0] */
  INT8 purephase[4];   /* 098h  Pure Phase Counter      0[31:24] ... 3[7:0] */
  INT8 btu_unused[4];  /*                                                */
/* SNU */              /*                                                */
  INT8 cssnu;          /* 0A0h  CS for SNU Time Stamp                 [7:0] */
  INT8 mssnu[3];       /* 0A1h  Macro Stamp Snapshot    0[23:16] ... 2[7:0] */
  INT8 tssnu[4];       /* 0A4h  Time Stamp Snapshot     0[31:24] ... 3[7:0] */
  INT8 accnsnu[2];     /* 0A8h  Accuracy N Snapshot          0[15:8] 1[7:0] */
  INT8 accpsnu[2];     /* 0AAh  Accuracy P Snapshot          0[15:8] 1[7:0] */
  INT8 snu_unused[20]; /*                                                */
/* APU */              /*                                                */
  INT8 csapp;          /* 0C0h  CS for APP Time Stamp                 [7:0] */
  INT8 msapp[3];       /* 0C1h  Macro Stamp Application 0[23:16] ... 2[7:0] */
  INT8 tsapp[4];       /* 0C4h  Time Stamp Application  0[31:24] ... 3[7:0] */
  INT8 accnapp[2];     /* 0C8h  Accuracy N Application       0[15:8] 1[7:0] */
  INT8 accpapp[2];     /* 0CAh  Accuracy P Application       0[15:8] 1[7:0] */
  INT8 csappx;         /* 0CCh  CS for APP Time Stamp X               [7:0] */
  INT8 msappx[3];      /* 0CDh  Macro Stamp Appl. X     0[23:16] ... 2[7:0] */
  INT8 tsappx[4];      /* 0D0h  Time Stamp Appl. X      0[31:24] ... 3[7:0] */
  INT8 accnappx[2];    /* 0D4h  Accuracy N Appl. X         0[31:24] 1[23:16] */
  INT8 accpappx[2];    /* 0D6h  Accuracy P Appl. X           0[15:8] 1[7:0] */
  INT8 apu_unused[8];  /*                                                */
/* GPU */              /*                                                */
  GPU_R8 gpu[3];       /* 0E0h GPS Unit 1                                */
                       /* 100h GPS Unit 2                                */
                       /* 120h GPS Unit 3                                */
/* SSU */              /*                                                */
  SSU_R8 ssu[6];       /* 140h Synchronisation Subnet Unit 1             */
                       /* 160h Synchronisation Subnet Unit 2             */
                       /* 180h Synchronisation Subnet Unit 3             */
                       /* 1A0h Synchronisation Subnet Unit 4             */
                       /* 1C0h Synchronisation Subnet Unit 5             */
                       /* 1E0h Synchronisation Subnet Unit 6             */
} UTCSU_READ8;


/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * UTCSU - Typdefinitionen
 * 32-/16-/8-Bit WRITE/READ in eine Union
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
typedef union
{
  UTCSU_WRITE32 w32;
  UTCSU_READ32  r32;

  UTCSU_WRITE16 w16;
  UTCSU_READ16  r16;

  UTCSU_WRITE8  w8;
  UTCSU_READ8   r8;
} UTCSU;

#endif
```

C.11 ntidet.s

```
* @(#) pSOSystem/68k-MRI V2.2.0: bsps/template/src/68k/misc.s (mri) 2.9 95/11/27 10:47:35 *
**************************************************************************
*                                                                        *
*   MODULE:  devices/misc.s                                              *
*   DATE:    95/11/27                                                    *
*   PURPOSE: miscellaneous common functions                              *
*                                                                        *
*------------------------------------------------------------------------*
*                                                                        *
*          COPYRIGHT 1991 - 1995, INTEGRATED SYSTEMS, INC.               *
*                       ALL RIGHTS RESERVED                              *
*                                                                        *
*    PERMISSION IS HEREBY GRANTED TO LICENSEES OF INTEGRATED SYSTEMS,    *
*    INC. PRODUCTS TO USE OR ABSTRACT THIS COMPUTER PROGRAM FOR THE      *
*    SOLE PURPOSE OF IMPLEMENTING A PRODUCT BASED ON INTEGRATED          *
*    SYSTEMS, INC. PRODUCTS.   NO OTHER RIGHTS TO REPRODUCE, USE,        *
*    OR DISSEMINATE THIS COMPUTER PROGRAM, WHETHER IN PART OR IN         *
*    WHOLE, ARE GRANTED.                                                 *
*                                                                        *
*    INTEGRATED SYSTEMS, INC. MAKES NO REPRESENTATION OR WARRANTIES      *
*    WITH RESPECT TO THE PERFORMANCE OF THIS COMPUTER PROGRAM, AND       *
*    SPECIFICALLY DISCLAIMS ANY RESPONSIBILITY FOR ANY DAMAGES,          *
*    SPECIAL OR CONSEQUENTIAL, CONNECTED WITH THE USE OF THIS PROGRAM.   *
*                                                                        *
*                                                                        *
*------------------------------------------------------------------------*
*                                                                        *
*                                                                        *
*                                                                        *
**************************************************************************
*-------------------------------------------------------------------------
* TURN ON CASE SENSITIVITY (_abc AND _ABC ARE DIFFERENT SYMBOLS)
*-------------------------------------------------------------------------
        OPT     CASE
        SECTION code
        XREF    _VectorPage,_ExcludedVectors
        XREF    _anchor


        XCOM _ExceptionHandlers,1024

**************************************************************************
* MemAccessible: CHECK WHETHER A MEMORY LOCATION IS ACCESSIBLE           *
*                                                                        *
*       SYNTAX: ULONG MemAccessible(volatile void *address);             *
*      RETURNS: 1 IF MEMORY IS ACCESSIBLE, 0 IF NOT                      *
*                                                                        *
*         NOTE: THIS USES THE VBR RATHER THAN _VectorPage TO LOCATE      *
*               THE VECTOR PAGE.                                         *
*                                                                        *
**************************************************************************
        XDEF    _MemAccessible
_MemAccessible:
*
*-------------------------------------------------------------------------
* DISABLE INTERRUPTS FOR THIS
*-------------------------------------------------------------------------
        MOVE.L  #7,-(SP)        ;NEW MASK LEVEL WILL BE 7
        JSR     _splx
        ADDQ.L  #4,SP           ;CLEAN UP STACK
        MOVE.L  D0,D1           ;D1: OLD MASK
*
*-------------------------------------------------------------------------
* TEMPORARILY REPLACE THE BUS ERROR VECTOR
*-------------------------------------------------------------------------
        JSR     GET_VBR         ;A0->VECTOR PAGE
        MOVE.L  8(A0),-(SP)     ;SAVE EXISTING BUS ERROR VECTOR
        MOVE.L  #MEM_BERR,8(A0) ;INSTALL NEW VECTOR
*
*-------------------------------------------------------------------------
* NOW DO THE TEST BY PROBING THE LOCATION IN QUESTION
*-------------------------------------------------------------------------
        MOVE.L  SP,A1           ;SAVE STACK POINTER
        MOVE.L  8(SP),A0        ;A0: ADDRESS TO ACCESS
        MOVEQ.L #1,D0           ;PRESET RESULT TO "ACCESSIBLE"
        TST.W   (A0)            ;PROBE THE WORD
        BRA.S   MEM_OK          ;IT'S ACCESSIBLE IF WE GOT HERE!
MEM_BERR:
        CLR.L   D0              ;SET RESULT TO "INACCESSIBLE"
MEM_OK:
        MOVEA.L A1,SP           ;RESTORE ORIGINAL STACK POINTER
*
*-------------------------------------------------------------------------
* RESTORE THE ORIGINAL BUS ERROR VECTOR
*-------------------------------------------------------------------------
```

```
        JSR     GET_VBR         ;A0->VECTOR PAGE
        MOVE.L  (SP)+,8(A0)     ;RESTORE BUS ERROR VECTOR
*
*-----------------------------------------------------------------------
* RESTORE ORIGINAL INTERRUPT LEVEL
*-----------------------------------------------------------------------
        MOVE.L  D0,-(SP)        ;SAVE VALUE TO RETURN
        MOVE.L  D1,-(SP)        ;OLD MASK LEVEL
        JSR     _splx           ;CHANGE THE LEVEL
        ADDQ.L  #4,SP           ;CLEAN UP STACK
        MOVE.L  (SP)+,D0        ;VALUE TO RETURN TO CALLER
*
        RTS                     ;RETURN

************************************************************************
*         splx: CHANGE THE PROCESSOR'S INTERRUPT MASK                  *
*                                                                      *
*       SYNTAX: ULONG splx(ULONG newmask);                             *
*       INPUTS: newmask = NEW PRIORITY LEVEL (0 - 7)                   *
*      RETURNS: PREVIOUS PRIORITY LEVEL                                *
*                                                                      *
*         NOTE: SETTING THE INTERRUPT MASK TO ANYTHING OTHER THAN 7 OR *
*               THE TASK'S INTERRUPT LEVEL IS STRONGLY DISCOURAGED,    *
*               AS IT CAN CAUSE THE pSOS+ SCHEDULER TO FUNCTION        *
*               IMPROPERLY. The tasks interrupt level is set by the    *
*               t_start and t_mode pSOS+ calls. These call may set     *
*               a tasks interrupt level to a value other then 0 so     *
*               changing the interrupt level to a level below the      *
*               tasks interrupt level can cause the task to be         *
*               interrupted when it should not be interrupted.         *
*                                                                      *
************************************************************************
        XDEF     _splx
_splx:
        MOVE.W   SR,D1               ;GET CURRENT SR
        ROR.L    #8,D1               ;PUT MASK IN LEAST-SIGNFCNT NBL
        MOVE.L   D1,D0               ;SAVE IT IN D0 TO RETURN
        ANDI.L   #7,D0               ;JUST RETURN MASK VALUE

        ANDI.L   #-8,D1              ;CLEAR OUT CURRENT VALUE
        OR.L     4(SP),D1            ;NEW MASK IN
        ROL.L    #8,D1               ;AND RESTORE TO GOOD LOOKING
        MOVE.W   D1,SR               ;NOW ACTUALLY CHANGE INTRPT LEV
        RTS


************************************************************************
*      GET_VBR: Moves the value of the current VBR into A0             *
*                                                                      *
*       SYNTAX: NONE not C callable.                                   *
*                                                                      *
************************************************************************
        XDEF    GET_VBR
GET_VBR:
        MOVEC   VBR,A0
        RTS
 ************************************************************************
*  SysSetVector                                                        *
*                                                                      *
*       SYNTAX: SysSetVector(vector number, (void (*)())isr,           *
*                            wrapper number);                          *
*       INPUTS:                                                        *
*               vector number is the index into the vector table       *
*                   to install this handler.                           *
*               isr is the address of the Interrupt Service Routine    *
*                   that will service this interrupt                   *
*               wrapper number This is the number of the assemble code *
*                   wrapper that will receive the interrupt and        *
*                   transfer control to a C coded fuction after        *
*                                                                      *
*      RETURNS: NONE                                                   *
*      OUTPUTS: NONE                                                   *
*      NOTE(S): SysSetVector first searches the list of                *
*               Excluded Vectors. The ExcludedVectors array is set in  *
*               the configs/std/sysinit.c. It uses a #define called    *
*               EXCLUDED_VECTORS found in ../bsp.h. This is a list of  *
*               vectors which should be left alone by pSOSystem.       *
*               Vectors in the "excluded" list will not be initialized. *
*                                                                      *
*                                                                      *
************************************************************************

        XDEF    _SysSetVector
_SysSetVector:
        MOVE.W  6(SP),D0                ;get the vector number off the
*                                       ;stack and store it in D0
*
*-----------------------------------------------------------------------
```

```
* SEE IF VECTOR IS in the "EXCLUDED" list
*-----------------------------------------------------------------------
        LEA     _ExcludedVectors,A0
CHK_LOOP:
        MOVE.W  (A0)+,D1                  ;Move vector off excluded list to
*                                         ;D0 and increment to next vector
*                                         ;in excluded list
        BEQ.S    NOT_EXCLUDED             ;0 INDICATES END OF LIST
        CMP.W    D0,D1                    ;Check for match
        BNE.S    CHK_LOOP                 ;branch to top of loop if no
*                                         ;match
        RTS                               ;VECTOR IS "EXCLUDED" - JUST RETURN
NOT_EXCLUDED:
        LSL.W   #2,D0                     ;multiply vector number by 4
        MOVEA.L #_ExceptionHandlers,A0    ;move address of indirect handler
*                                         ;table to A0
        MOVE.L  8(SP),(A0,D0.W)           ;move ISR address to indirect
*                                         ;handler table using vector
*                                         ;number as offset into table
        TST.L   12(SP)                    ;check for wrapper type
        BNE.S   L1                        ;if not 0 branch to wrapper code
        MOVE.L  4(SP),D0                  ;wrapper type was 0 no wrapper
*                                         ;needed
        LSL.L   #2,D0                     ;multiply vector number by 4
        MOVEC   VBR,A0                    ;get vector base address
        MOVE.L  8(SP),(A0,D0.L)           ;move ISR address to vector table
*                                         ;using vector as an offset
        RTS                               ;done
L1:
        MOVEQ   #1,D0                     ;move NORMAL_WRAPPER # to D0
        CMP.L   12(SP),D0                 ;check wrapper argument with D0
        BNE.S   L3                        ;branch if argument not
*                                         ;NORMAL_WRAPPER
        MOVE.L  4(SP),D0                  ;move vector number to D0
        LSL.L   #2,D0                     ;multiply vector number by 4
        MOVEA.L #_VectorPage,A0           ;move vector table address to A0
        MOVE.L  #_NormalWrapper,(A0,D0.L) ;move address of NORMAL_WRAPPER
*                                         ;code to vector table using
*                                         ;vector number as an offset
        RTS                               ;Done
L3:
        MOVEQ   #2,D0                     ;move NI_WRAPPER # to D0
        CMP.L   12(SP),D0                 ;check wrapper argument with D0
        BNE.S   L5                        ;branch if argument not
*                                         ;NI_WRAPPER
        MOVE.L  4(SP),D0                  ;move vector number to D0
        LSL.L   #2,D0                     ;multiply vector number by 4
        MOVEA.L #_VectorPage,A0           ;move vector table address to A0
        MOVE.L  #_NiWrapper,(A0,D0.L)     ;move address of NI_WRAPPER
*                                         ;code to vector table using
*                                         ;vector number as an offset
        RTS                               ;Done
L5:
        MOVEQ   #3,D0                     ;move MBRK_WRAPPER # to D0
        CMP.L   12(SP),D0                 ;check wrapper argument with D0
        BNE.S   L2                        ;branch if argument not
*                                         ;MBRK_WRAPPER
        MOVE.L  4(SP),D0                  ;move vector number to D0
        LSL.L   #2,D0                     ;multiply vector number by 4
        MOVEA.L #_VectorPage,A0           ;move vector table address to A0
        MOVE.L  #_MbrkWrapper,(A0,D0.L)   ;move address of MBRK_WRAPPER
*                                         ;code to vector table using
*                                         ;vector number as an offset
L2:
        RTS

*
*-----------------------------------------------------------------------
* DEFINE THE TRAP NUMBER USED FOR THE pSOS+ "RETURN FROM INTERRUPT"
* ENTRY POINT.
*-----------------------------------------------------------------------
I_RETURN    EQU      13                   ;TRAP # FOR pSOS+ I_RETURN CALL


************************************************************************
* NormalWrapper: GENERAL-PURPOSE INTERRUPT HANDLER WRAPPER             *
*                                                                      *
*        NOTES: CALLS THE APPROPRIATE HANDLER FOR THE INTERRUPT OR     *
*               EXCEPTION THAT OCCURRED.  THE HANDLER MAY BE WRITTEN   *
*               IN C BECAUSE THIS "WRAPPER" CODE SAVES & RESTORES THE  *
*               SCRATCH REGISTERS.                                     *
*                                                                      *
************************************************************************
_NormalWrapper:
        MOVEM.L  A0/A1/D0/D1,-(SP)   ;SAVE MICROTEC "C" SCRATCH REGS
        MOVE.W   22(SP),D0           ;D0: FORMAT/OFFSET WORD
        ANDI.L   #$FFF,D0            ;D0: VECTOR OFFSET
        LEA      _ExceptionHandlers,A0 ;A0-> INDIRECT HANDLER TABLE
```

```
        MOVEA.L  0(A0,D0),A0         ;A0-> HANDLER
        JSR      (A0)                ;CALL THE HANDLER
        MOVEM.L  (SP)+,A0/A1/D0/D1   ;RESTORE SAVED REGISTERS
        TRAP     #I_RETURN           ;LET pSOS CHECK FOR DISPATCH

*************************************************************************
*    NiWrapper: "NETWORK INTERFACE" INTERRUPT HANDLER WRAPPER           *
*                                                                       *
*       NOTES: CALLS THE APPROPRIATE HANDLER FOR THE INTERRUPT OR      *
*              EXCEPTION THAT OCCURRED.  THE HANDLER MAY BE WRITTEN    *
*              IN C BECAUSE THIS "WRAPPER" CODE SAVES & RESTORES THE   *
*              SCRATCH REGISTERS.  THE WRAPPER TERMINATES WITH AN RTE  *
*              INSTRUCTION IF THE HANDLER RETURNS 0.  OTHERWISE IT     *
*              TERMINATES BY TRAPPING TO pSOS+.  THIS WRAPPER SHOULD   *
*              BE USED AROUND ISR'S FOR NETWORK INTERFACE DRIVERS.     *
*                                                                       *
*************************************************************************
_NiWrapper:
        MOVEM.L  A0/A1/D0/D1,-(SP)   ;SAVE MICROTEC "C" SCRATCH REGS
        MOVE.W   22(SP),D0           ;D0: FORMAT/OFFSET WORD
        ANDI.L   #$FFF,D0            ;D0: VECTOR OFFSET
        LEA      _ExceptionHandlers,A0 ;A0-> INDIRECT HANDLER TABLE
        MOVEA.L  0(A0,D0),A0         ;A0-> HANDLER
        JSR      (A0)                ;CALL THE HANDLER
        TST.B    D0                  ;IS pSOS INITIALIZED?
        MOVEM.L  (SP)+,A0/A1/D0/D1   ;RESTORE SCRATCH REGS (LEAVES
        BEQ.S    DORTE               ;  CONDITION CODES ALONE)
        TRAP     #I_RETURN           ;USE I_RET IF pSOS INIT
DORTE:
        RTE                          ;OTHERWISE JUST USE RTE

*************************************************************************
*  MbrkWrapper: "MANUAL BREAK" INTERRUPT HANDLER WRAPPER                *
*                                                                       *
*       NOTES: CALLS THE APPROPRIATE HANDLER FOR THE INTERRUPT, AND    *
*              THEN TRANSFERS CONTROL TO THE "MANUAL BREAK" ENTRY OF   *
*              pROBE+.  THE HANDLER MAY BE WRITTEN IN C BECAUSE THIS   *
*              WRAPPER SAVES AND RESTORES THE SCRATCH REGISTERS.       *
*                                                                       *
*************************************************************************
_MbrkWrapper:
        SUBQ.L   #4,SP               ;ALLOCATE A SPACE ON STACK
        MOVEM.L  A0/A1/D0/D1,-(SP)   ;SAVE MICROTEC "C" SCRATCH REGS
        MOVE.W   26(SP),D0           ;D0: FORMAT/OFFSET WORD
        ANDI.L   #$FFF,D0            ;D0: VECTOR OFFSET
        LEA      _ExceptionHandlers,A0 ;A0-> INDIRECT HANDLER TABLE
        MOVEA.L  0(A0,D0),A0         ;A0-> HANDLER
        JSR      (A0)                ;CALL THE HANDLER
        MOVE.L   _anchor,A0          ;A0-> node config table
        MOVE.L   12(A0),A0           ;A0-> pROBE config table
        MOVE.L   (A0),A0             ;A0-> pROBE code
        ADDA.L   #$38,A0             ;A0-> pROBE mbrk entry
        MOVE.L   A0,16(SP)           ;PUT pROBE mbrk entry on stack

        MOVEM.L  (SP)+,A0/A1/D0/D1   ;RESTORE SCRATCH REGS
        RTS


        END
```

Anhang D

Abkürzungsverzeichnis

1pps	**1 P**uls **p**ro **S**ekunde
ABC	**A**dder **B**ased **C**lock
ACU	**Ac**curacy **U**nit
API	**A**pplication **P**rogrammer **I**nterface
APU	**Ap**plication **U**nit
ARP	**A**ddress **R**esolution **P**rotocol
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
ASR	**A**synchronous **S**ignal Service **R**outine
BIU	**B**us **I**nterface **U**nit
BSP	**B**oard **S**upport **P**ackage
BTU	**B**uilt-In **T**est **U**nit
CI	**C**lock **I**nterface
COMCO	**Com**munication **Co**processor
CPC	**C**arrier Board **P**lug **C**onnector
CPLD	**C**omplex **P**rogrammable **L**ogic **D**evice
CPU	**C**entral **P**rocessing **U**nit
CS	**C**heck **S**um
CSP	**C**lock **S**ynchronisation **P**acket
DFF	**D** - **F**lip **F**lop
DIP	**D**ual **I**n-Line **P**ackage
DMA	**D**irect **M**emory **A**ccess
EDCU	**E**rror **D**etection and **C**orrection **U**nit
EPROM	**E**rasable **P**rogrammable **R**ead **O**nly **M**emory
FIFO	**F**irst **I**n - **F**irst **O**ut
FPC	**F**ront **P**anel **C**onnector
GPS	**G**lobal **P**ositioning **S**ystem
GPU	**GPS U**nit
IC	**I**ntegrated **C**ircuit
ICMP	**I**nternet **C**ontrol **M**essage **P**rotocol
ID	**Id**entification
I/O	**I**nput/**O**utput
IP	**I**nternet **P**rotocol
IPC	**I**ntermodule **P**lug **C**onnector
ISR	**I**nterrupt **S**ervice **R**outine
ITU	**I**nterrupt **U**nit
KI	**K**ernel **I**nterface
LFSR	**L**inear **F**eedback **S**hift **R**egister
LSB	**L**east **S**ignificant **B**it

LTU	**L**ocal **T**ime **U**nit
MSB	**M**ost **S**ignificant **B**it
MSG	**M**e**s**sa**g**e
MTU	**M**aximum **T**ransmission **U**nit
NFS	**N**etwork **F**ile **S**ystem
NI	**N**etwork **I**nterface
NTI	**N**etwork **T**ime **I**nterface
NTP	**N**etwork **T**ime **P**rotocol
NTU	**N**etwork **T**ime Interface **U**nit
OCXO	**O**ven **C**ontrolled Quarz Crystal **O**szillator
OS	**O**perating **S**ystem
RAM	**R**andom **A**ccess **M**emory
RPC	**R**emote **P**rocedure **C**all
RTOS	**R**eal-**T**ime **O**perating **S**ystem
SC	**S**ystem **C**all
SNU	**Sn**apshot **U**nit
SSU	**S**ynchronization **S**ubnet Interface **U**nit
SRAM	**S**tatic **R**andom **A**ccess **M**emory
SynUTC	**Syn**chronized **U**niversal **T**ime **C**oordinated for Distributed Fault-Tolerant Real-Time Systems
TCB	**T**ask **C**ontrol **B**lock
TCP	**T**ransmission **C**ontrol **P**rotocol
TCXO	**T**emperature **C**ompensated Quarz Crystal **O**szillator
UDP	**U**ser **D**atagram **P**rotocol
UTC	**U**niversal **T**ime **C**oordinated
UTCSU	**U**niversal **T**ime **C**oordinated **C**lock **S**ynchronisation **U**nit
VHDL	**V**ery **H**igh Speed Integrated Circuit Hardware **D**escription **L**anguage
VME	**V**ersa **M**odule **E**urope
XO	Quarz Crystal **O**szillator

Anhang E

Notationen

- **Signale:** Invertierte Signale werden überstrichen und in Typewriter-Schrift (z.B. $\overline{\texttt{TSAPP8}}$), nichtinvertierte Signale (z.B. `1pps GPS1`) werden nur in Typewriter-Schrift dargestellt.
- **Datenworte und Adressen** werden entweder dezimal, binär oder hexadezimal dargestellt. In allen drei Fällen werden die Zahlen mit einem Typewriter-Schrifttyp gesetzt. Hexadezimale Zahlen werden gekennzeichnet, indem "0x" vor den Wert gesetzt wird (z.B. `0x12345678`). Binäre Zahlen werden mit einem vorangestellten "0b" markiert (z.B. `0b01100101`).
- **pSOS-Befehle** werden in einer Typewriter-Schrift gesetzt, z.B. `q_send`.
- **Listings** `werden ebenfalls in der Typewriter-Schrift gesetzt.`
- **Beispiele** *sind kursiv dargestellt.*

Literaturverzeichnis

[AcQ96] AcQisition Technology. *i6360: High-Performance Dual CPU VMEbus Processor Board Hardware Manual*, revision 1.3 edition, 1996.

[Ass97] VMEbus International Trade Association. `URL: http://www.vita.com/`, 1997.

[Bau94] B. Baumgarten. Hochgestapelt. *ELRAD*, 4, April 1994.

[CYP98] CYPRESS, `http://www.cypress.com/`. *CY7C1021 64K × 16 SRAM Data Sheet*, 1998.

[Dan97] P. H. Dana. Global Positioning System (GPS) time dissemination for real-time applications. *Real-Time Systems*, 12(1):9–40, January 1997.

[DIN85] DIN. *IEC 47B(C07) Mikroprozessorsysteme (VME-Bus)*, 1985.

[Hew97] Hewlett Packard, `http://www.hp.com/`. *HCPL-7101 High Speed CMOS Optocouplers Technical Data*, 1997.

[HL95] M. Horauer and D. Loy. Adder synthesis. In *Proceedings of Austrochip '95*, Graz, Austria, 1995.

[HL97] M. Horauer and D. Loy. UTCLIENT – an ASIC supporting clock synchronization in distributed real-time systems. In *Proceedings of Austrochip '97*, Linz, Austria, 9 April 1997.

[HL98] Martin Horauer and Dietmar Loy. Hardware-unterstützte Uhrensynchronisation in Verteilten Systemen. In *Proc. Austrochip 1998*, pages 67–72, Wiener Neustadt, Austria, October 1998. (ISBN 3-901578-03-X).

[HLS96] M. Horauer, D. Loy, and U. Schmid. NTI functional and architectural specification. Technical Report 183/1-69, Department of Automation, Vienna University of Technology, December 1996.

[Hoe95] Dieter Hoechtl. *Fehlverhalten von GPS-Satellitenempfängern im Hinblick auf ihren Einsatz im Projekt SynUTC*. Diplomarbeit, Technische Universität Wien, Department of Automation, October 1995.

[Hor96] M. Horauer. A primer to digital design with synopsys and cadence. Technical Report 183/1-68, Department of Computer Technology, Vienna University of Technology, October 1996.

[HS97] D. Höchtl and U. Schmid. Long-term evaluation of GPS timing receiver failures. In *Proceedings of the 29th Precise Time and Time Interval (PTTI) Systems and Application Meeting*, Long Beach, USA, 2–4 December 1997.

[HSS98] Martin Horauer, Ulrich Schmid, and Klaus Schossmaier. NTI: A Network Time Interface M-Module for high-accuracy clock synchronization. In *Proc. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'98)*, pages 1067–1076, Orlando, Florida, March-April 1998.

[Int92] Intel Corporation. *Intel 82596 32-bit LAN Component Users Manual*, 1992.

[Int96a] Integrated Systems Inc. *pSOS System*, 1996.

[Int96b] Integrated Systems Inc. *pSOS System Programmer's Reference*, 1996.

[KO87] H. Kopetz and W. Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers*, C-36(8):933–940, August 1987.

[LL84] J. Lundelius and N. Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62(2/3):190–204, 1984.

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

[Man96] Manufacturers and Users of M-Modules (MUMM) e.V. *M-Module Specification*, 1996.

[Mar84] K. Marzullo. *Maintaining the time in a distributed system. An example of a loosely-coupled distributed service.* PhD thesis, Department of Electrical Engineering, Stanford University, February 1984.

[MEN96] MEN Mikro Elektronik GmbH, Wiesentalstr. 40, 90419 Nürnberg. *A203: 6U High-Speed M-Module Base Board Hardware Documentation*, 20a203-00 edition, January 1996.

[Mic] MicroCrystal. *OCXO 8620 DIL Oven Controlled Crystal Oszillator Data Sheet.*

[MNS99] Thomas Mandl, Herbert Nachtnebel, and Ulrich Schmid. Network time interface user manual. Technical Report 183/1-87, Department of Automation, TU Vienna, January 1999.

[Mot90] Motorola, Prentice Hall. *MC68030 Enhanced 32-Bit Microprocessor User's Manual*, prentice hall edition, 1990.

[Mot96a] Motorola Inc. *MVME162FX Embedded Controller Installation and Use*, 1996.

[Mot96b] Motorola Inc. *MVME162FX Embedded Controller Programmer's Reference Guide*, 1996.

[Nat96] National Semiconductor Corporation, `http://www.national.com/`. *NM93C06L Data Sheet*, November 1996.

[NKC+98] Herbert Nachtnebel, Nikolaus Kerö, Gerhard R. Cadek, Thomas Mandl, and Ulrich Schmid. Rapid Prototyping mit programmierbarer Logik: Ein Fallbeispiel. In *Proc. Austrochip 1998*, pages 99–104, Wiener Neustadt, Austria, October 1998. (ISBN 3-901578-03-X).

[Pus95] A. Pusterhofer. *NTSU Network Timestamp Unit - Ein Modul für Eventtimestamping und Uhrensynchronisation via LAN.* Diplomarbeit, Technische Universität Wien, Department of Automation, 1995.

[Ric97] G. Richter. *Drivers for Real-Time Communications Coprocessors.* Diploma thesis, Department of Automation, Vienna University of Technology, June 1997.

[Sch94] U. Schmid. Monitoring distributed real-time systems. *Real-Time Systems*, 7:33–56, 1994.

[Sch95] Ulrich Schmid. Synchronized universal time coordinated for distributed real-time systems. *Control Engineering Practice*, 3(6):877–884, 1995. (Reprint from Proc. IFAC Workshop on Real-Time Programming WRTP'94, Lake Reichenau/Germany, 1994, p. 101–107.).

[Sch97a] U. Schmid. Interval-based clock synchronization with optimal precision. Technical Report 183/1-78, Department of Automation, Vienna University of Technology, December 1997.

[Sch97b] U. Schmid. Orthogonal accuracy clock synchronization. Technical Report 183/1-77, Department of Automation, Vienna University of Technology, March 1997.

[Sch97c] Ulrich Schmid. Hochgenaue Uhrensynchronisation über LANs. *Elektronikschau*, 10:20–23, October 1997.

[Sch97d] K. Schossmaier. An interval-based framework for clock rate synchronization. In *Proceedings of the 16th Annual ACM Symposium on Principles on Distributed Computing (PODC)*, pages 169–178, Santa Barbara, USA, 21–24 June 1997.

[Sch98] Klaus Schossmaier. *Interval-based Clock State and Rate Synchronization.* Dissertation, Technische Universität Wien, Faculty of Technical and Natural Sciences, 1998.

[SKM+99] Ulrich Schmid, Johann Klasek, Thomas Mandl, Herbert Nachtnebel, Gerhard R. Cadek, and Nikolaus Kerö. A Network Time Interface M-Module for distributing GPS-time over LANs. Technical report, Technische Universität Wien, Department of Automation, 1999. (forthcoming).

[SL96] K. Schossmaier and D. Loy. An ASIC supporting external clock synchronization for distributed real-time systems. In *Proceedings of the 8th Euromicro Workshop on Real-Time Systems*, pages 277–282, L'Aquila, Italy, 21–24 June 1996.

[SN99] Ulrich Schmid and Herbert Nachtnebel. Distributing GPS time over LANs using COTS components. In *Proc. IFAC Workshop on Real-Time Programming (WRTP'99)*, 1999.

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

[SS97] U. Schmid and K. Schossmaier. Interval-based clock synchronization. *Real-Time Systems*, 12(2):173–228, March 1997.

[SS99] Michael Schmidt and Ulrich Schmid. i82596 nti device-driver software documentation. Technical Report 183/1-90, Department of Automation, TU Vienna, February 1999.

[SSHL97] K. Schossmaier, U. Schmid, M. Horauer, and D. Loy. Specification and implementation of the Universal Time Coordinated Synchronization Unit (UTCSU). *Real-Time Systems*, 12(3):295–327, May 1997.

[vdG94] A.J. van de Goor. *Testing Semiconductor Memoriers - Theory and Practice*. Wiley Publishers, 1994.

[XIL98] XILINX Coorporation. *The Programmable Logic Databook*, 1998.

[YK93] V. N. Yarmolik and I. V. Kachan. *Self-Testing VLSI Design*. Elsevier Science Publisher, 1993.

Stichwortverzeichnis