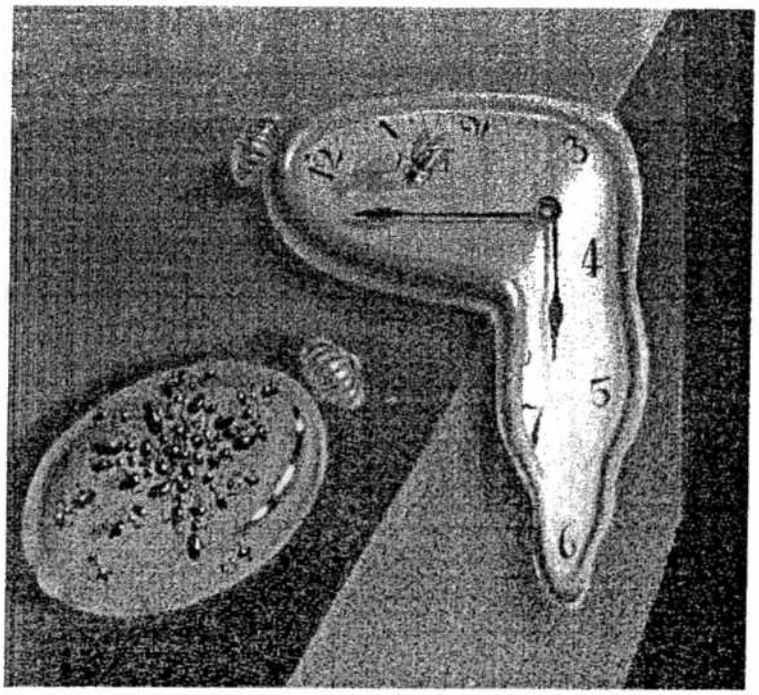
Institut für Automation
Abt. für Automatisierungssysteme

Technische
Universität
Wien

Projektbericht Nr. 183/1-39
Dezember 1993

FWF-Projektantrag SynUTC
Synchronized UTC for Distributed Real-Time Systems

U. Schmid



Ausschnitt aus: Salvador Dali, "Die Beständigkeit der Erinnerung"

Projektantrag SynUTC

U. SCHMID
Technische Universität Wien
Institut für Automation
Treitlstraße 3, A-1040 Wien
Email: s@auto.tuwien.ac.at

D. LOY
Technische Universität Wien
Institut für Computertechnik
Gußhausstraße 27, A-1040 Wien
Email: loy@ict.tuwien.ac.at

November 1993

Gegenstand des vorliegenden Projektantrages SynUTC (*Synchronized UTC for Distributed Real-Time Systems)* ist die Entwicklung eines integrierten Systems zur hochgenauen, fehlertoleranten Synchronisation lokaler Computeruhren auf UTC (*universal time coordinated*) in hierarchischen, LAN-basierenden Echtzeitsystemen im Automatisierungsbereich.

1 Einführung

Zeit spielt in *Echtzeitsystemen* (*real-time systems*), also in unserem Falle in Computersystemen zur Automatisierung (zeitkritischer) technischer Prozesse wie Fertigungsstraßen, Kraftwerken und Verkehrssystemen, eine ganz wesentliche Rolle. In der Ära der *zentralen* Prozeßrechner war die Bereitstellung einer systemweit gültigen Zeit nun kein Problem: Traten irgendwelche Ereignisse E_1, E_2 auf, so konnte diesen durch das Auslesen einer einzelnen *Computeruhr* ein eindeutiger Zeitstempel $t(E_1)$, $t(E_2)$ zugeordnet und somit eine totale zeitliche Ordnung von Ereignissen gewährleistet werden.

Moderne Echtzeitsysteme werden jedoch in immer stärkerem Maße als (heterogene und hierarchische) *verteilte Systeme* (*distributed systems*) realisiert: Die Gesamtaufgabe wird auf mehrere, über verschiedenste Netzwerke gekoppelte (Mikro-)Rechner (*Nodes*) aufgeteilt (*verteilte "Intelligenz"*). Diese Entwicklung hat vor allem positive Auswirkungen auf die Zuverlässigkeit und Erweiterbarkeit derartiger Systeme (Schlagwort *Fehlertoleranz*). Erkauft wird dieser Vorteil allerdings durch das Auftreten von Problemen, die bei zentralen Systemen völlig unbekannt waren.

So kann in einem netzwerkgekoppelten verteilten System vor allem keine zentrale Computeruhr mehr vorausgesetzt werden: Zunächst einmal ist jegliche Zentralisierung in Hinblick auf Fehlertoleranz ungünstig; darüberhinaus würde die Überbrückung kilometerweiter Distanzen einen (dedizierten) Zeitkanal notwendig machen, dessen Realisierung problematisch ist. Üblicherweise wird daher jeder Node eines verteilten Systems mit einer eigenen Computeruhr ausgerüstet. Dabei stellt sich jedoch das Problem, daß selbst hochgenaue Quarzuhren —über genügend lange Zeiträume hinweg beobachtet— beliebig weit "auseinanderlaufen" können und daher untereinander synchronisiert werden müssen.

Neben diesem Problem der *Clocksynchronisation*[1] gibt es natürlich noch eine ganze Reihe weiterer "berühmter" low-level Probleme in fehlertoleranten verteilten Systemen, etwa *distributed (byzantine) agreement*, *(reliable) atomic broadcast* oder *at-most once execution*; eine Übersicht ist z.B. in [Mul93] und [SS90] zu finden. Aufbauend auf deren Lösungen ist es möglich, die in verteilten Applikationen konkret anstehenden high-level Probleme anzugehen. Beispiele dafür sind *authentication* oder *membership protocols* in verteilten Systemen sowie das *transaction management* und die *data replication* in *distributed data bases*; siehe z.B. [Mul93], [Tan92] für eine Einführung.

Die Bereitstellung einer systemweit gültigen Zeit in verteilten Systemen hat zunächst einmal den Vorteil, anstelle von asynchronen System-Modellen (total) synchrone voraussetzen zu können (wodurch manche Probleme wie *byzantine agreement* überhaupt erst lösbar werden). Die auf synchronen Modellen basierenden Algorithmen zur Lösung der oben angedeuteten Probleme sind in der Regel wesentlich einfacher als deren asynchrone Gegenstücke und zeichnen sich darüberhinaus auch durch eine wesentlich verbesserte Performance aus; derartige Aspekte werden etwa in [Lis93] und [Lam84] diskutiert.

Diese Aspekte sind übrigens nicht nur für die Theorie von Bedeutung; die Entwicklung und Einführung des Internet-Standards NTP (*network time protocol*, siehe [Mil91]) zeigt den dringenden praktischen Bedarf sehr deutlich.

Zieht man jetzt noch die Tatsache in Betracht, daß verteilte *Echtzeitsysteme* (die bisherigen Argumente gelten ja für alle verteilten Systeme) aufgrund ihrer Interaktion mit der Außenwelt auch mit realen (Uhr-)Zeiten konfrontiert sind, so führt an der Bereitstellung einer systemweit einheitlichen Zeit kein Weg vorbei. Für kleinere Applikationen (*embedded systems*) ist dabei eine systemintern konsistente Zeit in der Regel ausreichend, für komplexere Anwendungen muß jedoch diese interne Zeit oft auch mit UTC (*universal time coordinated*) synchronisiert werden. In diesem Zusammenhang sollte erwähnt werden, daß UTC die einzige *gesetzlich anerkannte* Zeit und somit die Basis verbindlicher(!) Spezifikationen ist.

Zentralisierte Techniken zur Verbreitung derartiger (UTC-)Zeitinformation sind —auf anderen Gebieten, etwa der Navigation— schon seit langem im Einsatz; man denke nur an (traditionelle) Funkuhrsysteme wie z.B. den deutschen DCF-77 ([Sch77]). Solche Systeme basieren auf einem Sender (manchmal auch mehreren), der einer Trägerfrequenz digitale Zeitinformation aufmoduliert und (drahtlos oder drahtgebunden) an (mehrere) Empfänger sendet. Eine einfachere —allerdings nur für geringere Distanzen verwendbare— Methode

[1] Wir werden den Begriff *Clock* immer dann verwenden, wenn es sich bei der fraglichen Uhr um eine (durch Synchronisation) "manipulierte" handelt.

ist die Übertragung von Zeitinformation vermittels digitaler Signale (digitale Synchronisationssignale z.B. über RS-232, siehe [Lan93]).

Beim Empfang der Zeitinformation treten unvermeidbare Ungenauigkeiten bei der Dekodierung (Demodulation) auf; die resultierende Zeitunsicherheit kann allerdings durch die Verwendung von PLL-Empfängern auf einige μs reduziert werden. Nicht so einfach ist eine Reduktion des Einflusses der Übertragungszeit, die im Falle großer Distanzen zwischen Sender und Empfänger(n) nicht mehr vernachlässigbar klein (und im Falle einer Funkübertragung auch keineswegs konstant) ist, siehe zum Beispiel [EK73].

Wie schon erwähnt sind aber zentralisierte Lösungen für fehlertolerante verteilte Systeme ungünstig; ein zentraler Sender ist aber geradezu ein Musterbeispiel eines *single point of failures*. Wir wollen uns im folgenden daher ausschließlich mit Verfahren auseinandersetzen, die wenigstens prinzipiell für die Bereitstellung einer systemweit gültigen Zeit in verteilten, fehlertoleranten Systemen geeignet sind.

2 Stand der Forschung

Aufgabe dieses Abschnittes ist es, den gegenwärtigen Stand der Forschung auf dem Gebiet der Realisierung einer *global time in fault-tolerant distributed systems* kurz darzulegen. Die hierfür geeigneten Lösungen können wie folgt klassifiziert werden.

2.1 Dedizierte Zeitkanäle

Die Nodes des verteilten Systemes werden hier durch ein eigenes, der Übertragung von Zeitinformation dienendes Netzwerk verbunden[2].

2.1.1 Funkübertragung

Als "Netzwerk" dienen hier (v.a. *spread spectrum*) Funkkanäle, von denen einige für die Übertragung von Zeitinformation reserviert sind. Von besonderer Bedeutung ist hier das (nicht auf einem zentralen Sender basierende) NAVSTAR GPS (*global position system*, s. [Wel87]), das eine weltweit flächendeckende, redundante Versorgung ziviler und militärischer Systeme mit hochpräziser Zeit- und Positionsinformation sicherstellen soll. GPS besteht (zur Zeit) aus 24 Satelliten und einigen Bodenkontrollstationen, die von der *US Air Force* und anderen militärischen Stellen betrieben werden. Die Satelliten senden kontinuierlich hochpräzise Navigations- und Zeitinformation auf zwei Frequenzen L1 (1575.42 MHz) und L2 (1227.6 MHz) aus, die von vier an Bord jedes Satelliten befindlichen Frequenznormalen (und den von den Kontrollstationen "hochgeladenen" Informationen) abgeleitet werden.

Die Navigations- und Zeitinformation wird den Trägerfrequenzen mittels zweier *pseudo-random codes* aufmoduliert, dem für militärische Anwendungen vorgesehenen, hochgenauen *P code* (Übertragungssignalrate 10.23 $Mbps$) und dem auch zivilen Usern zur

[2]Hierzu muß nicht unbedingt ein eigenes System von Verbindungsleitungen existieren; die Zeitinformation kann auch über das reguläre Kommunikationsnetz (mittels Zeit- oder Frequenzmultiplexing) mitübertragen werden.

Verfügung stehenden, weniger genauen *C/A code* (1.023 *Mbps*); nähere Informationen dazu sind in [Wel87] zu finden. Die mittels des leichter zu dekodierenden C/A-Codes erreichbare Genauigkeit wird übrigens von den (militärischen) Betreibern des GPS absichtlich verschlechtert (*selective availability SA*).

An jedem Punkt der Erde (gegeben z.B. durch die Koordinaten x, y, z) soll(t)en die Signale von drei oder vier Satelliten gleichzeitig empfangen werden können. Mit Hilfe der Signale von vier Satelliten sind sowohl die Position (x, y, z) als auch die Uhrzeit (*GPS-Time*[3]) am Ort des Users bestimmbar. Ist die Position eines stationären Empfängers (einmal) bekannt, so genügt in der Folge ein einziger Satellit zur Bestimmung der genauen Uhrzeit.

Durch die Anwendung von (Korrektur-)Modellen für die Signalübertragungszeiten ist dabei eine beeindruckende Genauigkeit sowohl der Positions- (100 *m* mit SA aktiv) als auch der Zeitbestimmung (100 *ns*) erzielbar.

Der Hauptnachteil von GPS (für unser Einsatzgebiet) liegt in der Tatsache begründet, daß die schwachen Signale der Satelliten praktisch nur durch Außenantennen (außerhalb von Gebäuden und anderen abschirmenden Hüllen, an Punkten mit freier Sicht auf weite Teile des Himmels angebracht) empfangen werden können. Eigene Erfahrungen mit einem kommerziell erhältlichen GPS-Empfänger zeigen darüberhinaus, daß auch unter günstigen Bedingungen des öfteren keine oder falsche Zeitinformationen geliefert werden. Systematische Untersuchungen über räumliche und zeitliche Verfügbarkeit sowie mögliche Arten des Fehlverhaltens sind jedoch —nach unserem Stand des Wissens— nicht erhältlich.

Unter diesen Umständen ist klar, daß die —technisch im Prinzip wohl mögliche— Ausrüstung jedes Nodes eines verteilten Systems mit einer GPS-Uhr allein keine gute Lösung darstellt. Außerdem ist es mehr als zweifelhaft, ob die Funktionsfähigkeit eines verteilten Echtzeitsystems völlig dem "good will" der (militärischen) Betreiber des GPS ausgeliefert werden sollte. Als Quelle von UTC ist das GPS allerdings sehr attraktiv.

2.1.2 Kabelgebundene Übertragung

Bei Verwendung dieser Technik werden die Nodes eines verteilten Systems durch zwei im Prinzip unabhängige Netzwerke verbunden: einem für den "normalen" Datenverkehr und einem anderen für die Übertragung von Zeitinformation. Abgesehen von nicht-fehlertoleranten (kommerziell verfügbaren) Lösungen auf der Basis von IRIG (*Inter Range Instrumentation Group*) Zeitcodes oder digitalen Synchronisationssignalen (siehe z.B. [Lan93]) existieren auch in der wissenschaftlichen Literatur über fehlertolerante Systeme zahlreiche Arbeiten über sogenannte "Hardware-Verfahren" (siehe [RSB90] für eine Übersicht): Stimuliert von *fault-tolerant clocking systems* ([Smi81], [Kes84]) wurden hier vor allem Verfahren für *clock voting* mit PLL kombiniert ([KSB85], [SR88], [VM88]). Die über das zusätzliche Netzwerk zu übertragende Zeitinformation sind hier Clock-Signale, deren Übertragungszeit im Falle größerer Distanzen übrigens nicht mehr vernachlässigbar ist; eine mögliche Abhilfe wurde in [SR88] vorgestellt.

Nicht zuletzt existieren auch Arbeiten über die Kombination derartiger Techniken mit den noch vorzustellenden Verfahren zur Clocksynchronisation, siehe z.B. [SR87].

[3]Da zusätzlich zur GPS-Time auch UTC-Korrektursekunden übertragen werden, kann die jeweils gültige UTC errechnet werden.

2.2 Keine dedizierten Zeitkanäle

Unter diesen Punkt fallen die mehr oder minder "klassischen" Techniken der Synchronisation lokaler Uhren, bei denen die Zeitinformation über das in verteilten Systemen notwendigerweise vorhandene Kommunikations-Netzwerk (mit)übertragen wird. Es gibt eine große Anzahl wissenschaftlicher Arbeiten, die sich mit diesem Thema beschäftigen. Zunächst einmal lassen sich die folgenden zwei "Philosophien" unterscheiden.

2.2.1 Logical Time

In [Lam78] wurde die Relation *happened before* $\rightarrow$ und die dadurch induzierte Halbordnung von *computational events* in verteilten Systemen eingeführt. Darauf aufbauend konnte dann ein (verteilter) Algorithmus zur Realisierung logischer "Uhren" formuliert werden. *Logische Clocks* C haben —obschon in keiner Beziehung zur realen (Newton'schen) Zeit stehend— die Eigenschaft, konsistent mit den durch die Relation $\rightarrow$ induzierten Kausalitäten zu sein: Für zwei auf verschiedenen Nodes i, j aufgetretene Ereignisse E_i, E_j mit $E_i \rightarrow E_j$ gilt immer $C_i(E_i) < C_j(E_j)$, wobei C_i, C_j die logischen Clocks der beiden Nodes i, j (bei Auftreten der relevanten Ereignisse) sind.

Ausgehend davon haben sich eine Menge von Autoren der Realisierung verteilter Systeme auf der Basis von logischen Clocks gewidmet, siehe z.B. [Jef85] oder [Fid91]; eine *annotated bibliography* ist in [YM93] zu finden. Der inherente Vorteil dieses konzeptuell eleganten Ansatzes ist es, ohne jeden Hardware-Support auszukommen. Das Defizit logischer Clocks tritt zutage, wenn kausal wirksame *externe* Ereignisse die Operation des verteilten Systems beeinflussen: Diesen liegt nicht die system-interne logische Zeit, sondern (meist) eine reale Zeit (etwa UTC) zugrunde. Für Echtzeitsysteme im speziellen scheidet diese Variante der Lösung der globalen Zeitproblematik daher aus.

2.2.2 Real-Time Synchronization

Unter diesen Punkt fallen die meisten der zur Thematik Clocksynchronisation gehörenden Arbeiten. Angesichts der Bedeutung gerade dieser Techniken für das gegenständliche Projekt soll der Beschreibung des ensprechenden Standes der Forschung etwas[4] breiterer Raum gegeben werden.

Zunächst sind eine ganze Reihe von Arbeiten primär den grundlegenden theoretischen Problemen der Clocksynchronisation gewidmet. Eine zentrale Rolle nehmen dabei Untersuchungen darüber ein, unter welchen Voraussetzungen (System-Modellen, Fehlerannahmen) eine (sinnvolle) Clocksynchronisation überhaupt möglich ist (siehe z.B. [DHS84], [FLM86]). Ein weiteres, sehr wichtiges theoretisches Resultat ist eine *lower bound* für die überhaupt erreichbare Synchronisations-Genauigkeit, die von keinem (nicht-probabilistischen) Algorithmus unterschritten werden kann ([LL84], [DHS84], [Ma92]).

[4] In der Literatur existiert eine beträchtliche Anzahl von Arbeiten zu diesem Thema, die aus Platzgründen selbstverständlich nicht alle erwähnt werden können. Es gibt jedoch einige recht brauchbare Übersichtsartikel zum Thema Clocksynchronisation: Primär theoretische Aspekte werden in [SWL90] diskutiert, eine mehr praxisorientierte Diskussion existierender Verfahren ist in [RSB90] enthalten. *Annotated bibliographies* finden sich schließlich in [YM93] und [S93].

Diesen (und den darauf aufbauenden) Arbeiten liegt in der Regel folgendes (partiell synchrone) System-Modell zugrunde:

- Ein verteiltes System bestehend aus n Prozessoren mit unterschiedlichen Fehlerannahmen, von *fail-safe* über *crash* und *omission failures* bis hin zu *timing* und schließlich *byzantine failures*.
- Vollverbundenes, fehlerfreies Punkt-zu-Punkt-Netzwerk mit garantiert (*upper* und *lower*) *bounded transmission delays* $\delta \in [\delta_l, \delta_u]$.
- Eine Hardware-Uhr pro Node, deren Drift in bezug auf die reale (Newton'sche) Zeit durch eine Konstante $\rho(\approx 10^{-6})$ beschränkt bleibt.

Eine beträchtliche Anzahl von Arbeiten ist dann konkreten Algorithmen für die Clocksynchronisation und deren Analyse gewidmet. Praktisch alle diese Algorithmen basieren auf sukzessiven Phasen, die *Resynchronisationsintervalle* genannt werden. Innerhalb eines solchen Intervalles laufen die Clocks der Nodes ohne äußeren Eingriff entsprechend ihrer Hardware-Uhr; am Ende jedes Resynchronisationsintervalles bestimmt jeder Node mit Hilfe des konkreten Clocksynchronisations-Algorithmus einen Korrekturwert für seinen eigenen Clock. Mit der danach erfolgenden tatsächlichen Korrektur wird das nächste Resynchronisationsintervall initiiert.

Der Klassifikation in [RSB90] folgend kann man folgende Typen von Algorithmen unterscheiden:

- *Convergence-averaging Algorithmen*

  Derartige Algorithmen basieren auf der Idee, daß jeder Node im Laufe eines (kurzen) Intervalls am Ende eines Resynchronisationsintervalles die Clockdifferenzen zu allen anderen Nodes bestimmt und zur Korrekturwertberechnung eine geeignete (fehlertolerante) Averaging-Funktion verwendet; verschiedene solche Funktionen wurden vorgeschlagen, siehe etwa [LM85], [LL88], [MS85] und [PB91].

- *Convergence non-averaging Algorithmen*

  Diese Algorithmen ([HSSD84], [CAS86], [ST87], [VR92], [DB93]) basieren auf der Idee, das Ende der Resynchronisationsintervalle aller Nodes durch das Ende des Resynchronisationsintervalles eines einzelnen Nodes (des *synchronizers*) festzulegen. Fehlertoleranz wird dadurch erreicht, daß jeder Node versucht, *synchronizer* zu werden, jedoch nur einer aus einer geeignet bestimmten Gruppe von Kandidaten tatsächlich "gewinnen" kann.

- *Consistency Algorithmen*

  Derartige Algorithmen ([LM85]) basieren im Prinzip darauf, daß jeder Node am Ende eines Resynchronisationsintervalles den für *byzantine agreement* gedachten *interactive consistency algorithm* ([LSP82]) verwendet, um seinen Clock (in Form von Clock-Differenzen) an alle anderen Nodes zu verschicken. Auf diese Weise hat jeder korrekte Node am Ende dieser Phase einen Satz von Clock-Differenzen zu den Clocks der anderen Nodes (und diese Sätze sind mit denen der anderen Nodes

konsistent), wodurch ein konsistenter Korrekturwert für den eigenen Clock bestimmt werden kann.

Es gibt darüberhinaus auch noch einige andere Ansätze für die Clocksynchronisation: Neben einigen auf Master/Slave-Basis aufgebauten Verfahren wie dem bekannten TEMPO-Algorithmus ([GZ89]) gibt es etwa Intervall-Algorithmen ([Mar82]), bei denen jeder Node die Genauigkeit seines Clocks mitführt und im Zuge der Synchronisation mit anderen Nodes aktualisiert; ein im Prinzip ähnlicher Ansatz wird auch bei NTP ([Mil91]) verwendet.

In unserem Kontext besonders wichtig sind jene Lösungen ([KO87], [RKS90], [DB93], [VR92]), denen statt eines Punkt-zu-Punkt Netzwerkes ein *Broadcast-Netzwerk* zugrundeliegt. Auch in diesem Modell existiert eine nicht unterschreitbare *lower bound* für die erreichbare Synchronisationsgenauigkeit, siehe [HS91]. Allerdings ist hier —trotz einer in diesem Falle zulässigen Asynchronität der (gesamten) Broadcast-Übertragung— eine wesentliche quantitative Verbesserung zu erzielen, da der Empfang einer Broadcast-Message in verschiedenen Nodes in der Regel fast gleichzeitig erfolgt.

Ein vor allem für sehr große verteilte Systeme (wie das Internet, für das das NTP entwickelt wurde, siehe [Mil91]) interessanter Alternativansatz zur Lösung des Problems der Clocksynchronisation sind *probabilistische Algorithmen*. Ihnen liegt eine völlig geänderte Modellierung der Voraussetzungen und Ziele zugrunde: Um den Preis, die Synchronisation nicht mehr mit absoluter Sicherheit, sondern nur mit einer (beliebig großen) Wahrscheinlichkeit $p < 1$ garantieren zu können, kann das den deterministischen Algorithmen zugrundeliegende partiell synchrone Modell zugunsten eines asynchronen (probabilistischen) Modells der Datenübertragung im Netzwerk aufgegeben werden. Darüberhinaus unterliegen probabilistische Algorithmen auch nicht mehr der unvermeidbaren *lower bound* deterministischer Verfahren.

Die beiden existierenden "Basis-Verfahren" in [Cri89] und [Arv89] verwenden die Methode des mehrfachen Versendens von Messages zur (statistischen) Reduktion der Variabilitäten der *transmission delays*. Das hohe Message-Aufkommen läßt jedoch die voll replizierte Verwendung dieser Idee auf allen n Nodes nicht zu; eine Master/Slave-Umgebung mit Auswahl wie in [Mil91] oder eine Replikation längs eines Hamilton'schen Kreises durch das Netzwerk ([OS91], [RT91]) sind mögliche Alternativen.

Bei der Analyse der Performance von Clocksynchronisations-Algorithmen wurde vor allem die *interne* Synchronisation betrachtet, d.h., die Synchronisation der lokalen Clocks untereinander, ohne jeden Bezug zur realen Zeit. Die Güte der internen Synchronisation wird durch die *precision* ausgedrückt, worunter die (maximale) Abweichung zweier beliebiger Clocks voneinander zum selben realen Zeitpunkt zu verstehen ist. Formeln für die *precision* in Abhängigkeit von den diversen Modellparametern sind für alle wesentlichen Algorithmen verfügbar; die Unsicherheit $\delta_u - \delta_l$ des *transmission (broadcast) delays* ist üblicherweise der dominante Faktor darin. Die mit den bisher beschriebenen Verfahren zur Clocksynchronisation konkret erreichbare *precision* liegt im Bereich von Millisekunden; durch zusätzlichen Hardware-Support kann dies jedoch um 1-2 Größenordnungen verbessert werden (siehe [KO87]).

Der im Zusammenhang mit der externen Synchronisation bedeutsamen *accuracy*, worunter die Abweichung eines beliebigen Clocks von der realen Zeit zu verstehen ist, wurde in der Regel nicht dieselbe Aufmerksamkeit geschenkt. Eine wichtige Ausnahme ist der Algorithmus in [ST87], dessen *accuracy* gleich der der zugrundeliegenden Hardware-Uhren (ρ) ist; ein theoretisches Resultat zeigt, daß eine Unterschreitung dieser lower bound nicht möglich ist. In den von [ST87] inspirierten Arbeiten [DB93] und [VR92] wird ebenfalls die Beziehung zur realen Zeit untersucht.

Eine weitere Kenngröße von Algorithmen zur Clocksynchronisation ist das resultierende Message-Aufkommen, das im Falle von vollverbundenen Punkt-zu-Punkt Netzwerken in der Regel $O(n^2)$ ist; für Broadcast-Netzwerke reduziert sich dies natürlich auf $O(n)$. Speziell in sehr großen Systemen ist der durch die Clocksynchronisation verursachte Overhead daher recht hoch; es gibt jedoch einige Verfahren wie [DB93], die es erlauben, die Clocksynchronisation mehr oder weniger transparent dem "normalen" Datenverkehr "aufzuladen". Eine weitere Möglichkeit zur Reduktion des Overheads ist die (hierarchische) Unterteilung des Gesamtsystems in Cluster (wie in [Mil91] oder [OS91]); auf diese Weise ergibt sich jedoch üblicherweisie eine verminderte *intra-cluster precision*.

Im Zusammenhang mit all den vorgestellten Algorithmen für die Clocksynchronisation gibt es natürlich auch jede Menge kleinerer Detailprobleme. Ein solches ist etwa die Frage, wie die Korrektur eines Clocks tatsächlich erfolgen soll. Ein einfaches Umsetzen des Standes der Hardware-Uhr ist problematisch, da zu schnell laufende Exemplare (der *accuracy* wegen) zurückgesetzt werden müssen. Dadurch wird aber eine in Echtzeitsystemen oft lebenswichtige (Eindeutigkeits-)Eigenschaft von Zeitstempeln, die *Monotonie*, verletzt. Abhilfe schafft eine graduelle Clock-Korrektur im nächsten Resynchronisationsintervall ([KO87], [ST87]), deren Berücksichtigung jedoch die ohnedies oft sehr schwierige Analyse der Algorithmen nicht gerade leichter macht (siehe aber das generelle Resultat von [CS90]).

Ebenfalls von Bedeutung ist die Frage der initialen Synchronisation, da die meisten der vorgestellten Algorithmen ein ursprünglich synchronisiertes System voraussetzen; entprechende Algorithmen sind etwa in [LM85], [LL88], [ST87] und [PB91] zu finden. Ebenfalls hierher gehört das Problem des *(late) joins*, also der Integration eines neuen Nodes in ein bereits laufendes System (siehe z.B. [CAS86], [Mil91], [ST87]).

In den bisher besprochenen, im großen und ganzen doch eher theoretisch ausgerichteten Arbeiten wird konkreten "engineering"-Aspekten der Clocksynchronisation verhältnismäßig wenig Beachtung geschenkt; eine Diskussion der im Automatisierungsbereich anfallenden Anforderungen an Clocksynchronisations-Verfahren ist in [GHT88] zu finden. In der industriellen Praxis sind derartige Verfahren unseres Wissens nach bis jetzt auch kaum je eingesetzt worden.

Eine rühmliche Ausnahme ist das in [KO87] beschriebene Verfahren zur Synchronisation lokaler Clocks in der LAN-basierenden, verteilten Echtzeitsystem-Architektur MARS, das den *fault-tolerant average algorithmus* [LL88] mit einer hardwaremäßigen *clock synchronization unit CSU* kombiniert. Letztere unterstützt vor allem das Timestamping der Messages beim Senden und Empfangen über das Netzwerk und die graduelle Korrektur der Clocks (*rate* und *state*). Die mit Hilfe der CSU erreichbare *precision* liegt in der Größenordnung von $10 - 100\ \mu s$.

In vielleicht noch stärkerem Maße als 'well-engineered' zu bezeichnen ist natürlich das NTP (siehe [Mil91]). Dessen Aufgabe ist die Bereitstellung eines zuverlässigen Time-Services für die hunderttausenden Hosts des Internets mit all seinen verschiedenen Übertragungscharakteristiken, Ausfällen und teilweise sogar fehlerhaften Implementierungen. Die erreichbare *precision* liegt bei einigen wenigen Millisekunden. NTP verwendet unter anderem:

- Ein selbstorganisierendes *synchronization subnet* von Rechnern, die als *time server* (teilweise mit Zugang zu UTC) für die Internet-Hosts fungieren.
- Eine (probabilistische) Clocksynchronisation basierend auf sorgfältig entworfenen und optimierten Filter- und Selektionsverfahren zur Auswahl der zum jeweiligen Zeitpunkt besten verfügbaren Zeitquelle(n).
- Hardware-Uhren für die *dedicated time server* auf Basis einer *adaptive-parameter PLL* für die kontinuierliche Clock(rate)-Korrektur.

Literatur

[Arv89] K. Arvind. *A New Probabilistic Algorithm for Clock Synchronization*, Proc. IEEE Real Time Systems Symposium, Santa Monica, California, December 1989, p. 330–339.

[BFR87] J. Boudenant, B. Feydel, P. Rolin. *LYNX: An IEEE 802.3 Compatible Deterministic Protocol*, Proc. IEEE INFOCOM '87, San Francisco, California March-April 1987, p. 573–579.

[CAS86] F. Cristian, H. Aghili, R. Strong. *Clock Synchronization on the Presence of Omission and Performance Faults and Processor Joins*, Proc. 16th International Symposium on Fault Tolerant Computing Systems, Vienna, Austria, July 1986, p. 218–223.

[Cri89] F. Cristian. *Probabilistic Clock Synchronization*, Distributed Computing, 3, 1989, p. 146–158.

[CS90] F. Cristian, F. Shmuck. *Continuous Clock Amorization Need not Affect the Precision of fa Clock Synchronization Algorithm*, 9th Annual ACM Symposium on Principles of Distributed Computing, p. 133–143.

[DB93] R. Drummond, Ö. Babaoğlu. *Low-cost clock Synchronization*, Distributed Computing, 6, 1993, p. 193–203.

[DHS84] D. Dolev, J. Halpern, H. R. Strong. *On the Possibility and Impossibility of Achieving Clock Synchronization*, Proc. 16th Annual ACM Symposium on Theory of Computing, Washington, D.C., April/May 1984, p. 504–511.

[EK73] C. E. Ellingson, R. J. Kulpinski. *Dissemination of System Time*, IEEE Transactions on Communications 21(5), May 1973, p. 605–624,

[Fid91] C. Fidge. *Logical Time in Distributed Computing Systems*, IEEE Computer 24(8), August 1991, p. 28–33.

[FLM86] M. J. Fischer, N.A. Lynch, M. Merrit. *Easy impossibility proofs for distributed consensus problem*, Distributed Computing, 1(1), 1986, p. 26–39.

[GHT88] W. Gora, U. Herzog, S. K. Tripathi. *Clock Synchronization on the Factory Floor*, IEEE Transactions on Industrial Electronics, 35(3), August 1988, p. 372–380.

[GZ89] R. Gusella, S. Zatti. *The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD*, IEEE Transactions on Software Engineering, 15(7), July 1989, p. 847–853.

[HS91] J. Y. Halpern, I. Suzuki. *Clock synchronization and the power of broadcasting*, Distributed Computing, 5(2), September 1991, p. 73–82.

[HSSD84] J. Halpern, B. Simons, R. Strong, D. Dolev. *Fault-Tolerant Clock Synchronization*, Proc. 3rd Annual ACM Symposium on Principles of Distributed Computing, August 1984, p. 89-102.

[Jef85] D. R. Jefferson. *Virtual Time*, ACM Transactions on Programming Languages and Systems 7(3), July 1985, p. 404–425.

[Kes84] J. L. W. Kessels. *Two Designs of a Fault-Tolerant Clocking System*, IEEE Transactions on Computers 33(10), October 1984, p. 912–919.

[KS89] H. Kopetz, W. Schwabl. *Global Time in Distributed Real-Time Systems*, Research Report Institut für Technische Informatik, Technical University of Vienna, 15/89, October 1989, p. 1–26.

[KSB85] C. M. Krishna, K. G. Shin, R. G. Butler. *Ensuring Fault Tolerance of Phase-Locked Clocks*, IEEE Transactions on Computers, C34(8), August 1985, p. 752–756.

[KO87] H. Kopetz, W. Ochsenreiter. *Clock Synchronization in Distributed Real-Time Systems*, IEEE Transactions on Computers, C-36(8), August 1987, p. 933–940.

[Lam78] L. Lamport. *Time, Clocks, and the Ordering of Events in a Distributed System*, Communications of the ACM 21(7), July 1978, p. 558–565.

[Lam84] L. Lamport. *Using Time Instead of Timeout for Fault-Tolerant Distributed Systems*, ACM Transactions on Programming Languages and Systems 6(2), April 1984, p. 254–280.

[Lan93] W. R. Lange. *Die verschiedenen Möglichkeiten zur Verteilung von Zeit in Rechnernetzen*, Proc. PTB-Seminar "Funkuhren, Zeitsignale, Normalfrequenzen", Mai 1993, to appear in W. Hilberg (ed.), Verlag Sprache & Technik, Groß Biberau, Germany.

[LM85] L. Lamport, P. M. Melliar-Smith. *Synchronizing Clocks in the Presence of Faults*, Journal of the ACM, 32(1), January 1985, p. 52–78.

[Lis93] B. Liskov. *Practical uses of synchronized clocks in distributed systems*, Distributed Computing 6, 1993, p. 211–219.

[LL84] J. Lundelius-Welch, N. Lynch. *An Upper and Lower Bound for Clock Synchronization*, Information and Control 62(2-3), August-September 1984, p. 190–204.

[LL88] J. Lundelius-Welch, N. Lynch. *A New Fault-Tolerant Algorithm for Clock Synchronization*, Information and Computation 77(1), 1988 p. 1–36.

[LSP82] L. Lamport, R. Shostak, M. Pease. *The Byzantine Generals Problem*, ACM Transactions on Programming Languages and Systems, 4(1), July 1982, p. 382–401.

[LZM90] M. Lu, D. Zhang, T. Murata. *Analysis of Self-Stabilizing Clock Synchronization by Means of Stochastic Petri Nets*, IEEE Transaction of Computers, 39(5), May 1990, p. 597–604.

[Ma92] M. Mavronicolas. *An Upper and a Lower Bound for Tick Synchronization*, Proc. Real-Time Systems Symposium, Phoenix, Arizona, December 1992, p. 246–255.

[Mar82] K. Marzullo. *Loosely Coupled Distributed Services: A Distributed Time Service*, Pd.D. Thesis, Stanford University, 1983.

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

[MS85] S. Mahaney, F. Schneider. *Inexact Agreement: Accuracy, Precision and Graceful Degradation*, Proc. 4th Annual ACM Symposium on Principles of Distributed Computing, August 1985, p. 237–249.

[Mul93] S. Mullender. *Distributed Systems*, Addison Wesley, 1993.

[OS91] A. Olson, K. G. Shin. *Probabilistic Clock Synchronization in Large Distributed Systems*, Proc. 11th IEEE International Conference on Distributed Computing Systems, Arlington, Texas, USA, May 1991, p. 290–297.

[PB91] M. J. Pfluegl, D. M. Blough. *Evaluation of a New Algorithm for Fault-Tolerant Clock Synchronization*, Proc. Pacific Rim Internation Symposium on Fault Tolerant Systems, Kawasaki, Japan, September 1991, p. 38–43.

[PS91] K. H. Prodromides, W. H. Sanders. *Performability Evaluation of CSMA/CD and CSMA/DCR Protocols under Transient Fault Conditions*, Proc. 10th Symposium on Reliable Distributed Systems, Pisa, Italy, September 1991, p. 166–176.

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

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

[RT91] S. Rangarajan, S. K. Tripathi. *Efficient Synchronization of Clocks in a Distributed System*, Proc. IEEE Real-Time Systems Symposium, San Antonio, Texas, December 1991, p. 22–31.

[S93] U. Schmid. *An Annotated Bibliography on Clock Synchronization in Distributed Systems*, Technical Report Department of Automation, Technical University of Vienna, (to appear), 1993.

[Sch77] G.-H. Schildt. *Zeitsynchrones Kommunikationssystem mit zentraler Synchronisation durch einen Normalfrequenzsender für ein bedarfsgesteuertes Verkehrssystem*, Schriftenreihe Institut für Verkehr, Eisenbahnwesen und Verkehrssicherung, Technische Universität Braunschweig, Heft 13, 1977.

[Smi81] T. B. Smith. *Fault-tolerant clocking system*, Digest of Papers International Symposium on Fault-Tolerant Computing FTCS-11, June 1981, p. 262–264.

[SR87] K. G. Shin, P. Ramanathan. *Clock Synchronization of a Large Multiprocessor System in the Presence of Malicious Faults*, IEEE Transactions on Computers C-36(1), January 1987, p. 2–12.

[SR88] K. G. Shin, P. Ramanathan. *Transmission Delays in Hardware Clock Synchronization*, IEEE Transactions on Computers 37(11), November 1988, p. 1465–1467.

[SS90] B. Simons, A. Spector (Eds.). *Fault-Tolerant Distributed Computing*, Lecture Notes in Computer Science 448, Springer-Verlag Berlin, Heidelberg, 1990.

[SWL90] B. Simons, L. Lundelius-Welch, N. Lynch. *An Overview of Clock Synchronization*, B. Simons, A. spector, editors: Fault-Tolerant Distributed Computing, Lecture Notes on Computer Science 448, 1990, p. 84–96,

[ST87] T. K. Srikanth, S. Toueg. *Optimal Clock Synchronization*, Journal of the ACM, 34(3), July 1987, p. 626–645.

[Tan92] A. S. Tanenbaum *Modern Operating Systems*, Prentice-Hall, New Jersey, 1992.

[VM88] N. Vasanthavada, P. N. Marinos. *Synchronization of Fault-Tolerant Clocks in the Presence of Malicious Failures*, IEEE Transactions on Computers, C-37(4), April 1988, p. 440–448.

[VR92] P. Veríssimo, L. Rodrigues. *A posteriori Agreement for Fault-tolerant Clock Synchronization on Broadcast Networks*, Proc. 22nd International Symosium on Fault-Tolerant Computing, Boston, Massachusetts, July 1992, p. 527–535.

[Wel87] D. Wells. *Guide to GPS Positioning*, Canadian GPS Associates.

[YM93] Z. Yang, T. A. Marsland. *Annotated Bibliography on Global States and Times in Distributed Systems*, ACM SIGOPS Operating Systems Review, June 1993, p. 55–72.

3 Projektdefinition

Dieser Abschnitt hat die Aufgabe, die wissenschaftlichen Fragestellungen (konkreten Ziele) des beantragten Projektes sowie die zu deren Lösung geplante Vorgangsweise darzulegen. Schon hier sei festgehalten, daß es sich bei dem gegenständlichen Projekt um ein bewußt *wirtschaftsnahes Forschungsprojekt* (*oriented basic research*) handelt.

3.1 Wissenschaftliche Fragestellungen/Projektziele

Grob umrissen hat das Projekt die Entwicklung eines den tatsächlichen Praxisanforderungen gerecht werdenden *Konzeptes* inklusive *Prototypen-Realisierung* für die Clocksynchronisation in verteilten, hierarchischen, fehlertoleranten Echtzeitsystemen zum Inhalt. Konkrete Ziele sind:

1. **Externe Synchronisation**

   Der Problemkreis der fehlertoleranten externen Clocksynchronisation, d.h., der Synchronisation lokaler Clocks unter Einbeziehung von externen Referenzzeitquellen, *"constitutes a research topic in its own right"*, wie in [Cri89, page 151] formuliert wird. Tatsächlich gibt es unseres Wissens nach keine Arbeiten zu diesem Thema, die über das Stadium der Problemformulierung ([KS89]) oder "optimistischer" Ansätze ([Cri89], [Mil91]) hinausgehen. In diesem Zusammenhang ist die —bisher vernachlässigte— Tatsache besonders wichtig, daß nur UTC —und keine irgendwie geartete rechnerinterne Zeit— die *gesetzliche Zeit* ist, in welcher verbindlich Benutzeranforderungen spezifiziert und Ereignisse protokolliert werden können. Dem gegenständlichen Projekt kommt hier unzweifelhaft —auch international gesehen— eine Vorreiter-Rolle zu.

   Insbesondere sind in diesem Zusammenhang folgende Punkte (sowohl theoretisch als aus praktisch) zu lösen:

   - Wie kann die Integration von interner und externer Synchronisation erfolgen?
   - Durch welche Maßnahmen können Fehler der externen Zeitquellen (GPS-Empfänger: *loss of satellite* oder falsche Information) aufgedeckt und "maskiert" werden? Welche Fehlerannahmen sollten getroffen werden, und wie wirken sie sich auf *precision* und *accuracy* aus?
   - Wie kann sowohl ein für die interne Verwendung geeigneter, Ressourcen-ökonomischer Timestamp als auch —für komplexe Systeme— die volle UTC-Zeit bereitgestellt werden, die (garantiert) konsistent zueinander sind?

2. **Hohe Synchronisationsgenauigkeit**

Wie können existierende Verfahren zur Clocksynchronisation adaptiert bzw. (wenn nötig) weiterentwickelt werden, um eine Synchronisationsgenauigkeit in der Größenordnung von 1-2 μs zu erreichen? Angesichts der *precision* existierender Verfahren (ms-Bereich für reine "Software-Lösungen", einige $10\mu s$ mit Hardware-Support, siehe den vorigen Abschnitt) entspricht dies eine Verbesserung um eine Größenordnung.

Dabei sind folgende Randbedingungen zu berücksichtigen:

- *Verwendbarkeit zusammen mit existierender Hardware/Software*

  Moderne, verteilten Automatisierungssysteme können etwa wie folgt charakterisiert werden ([GHT88]):

  - Oft sehr große, geographisch weit verteilte Systeme hierarchischer Struktur.
  - Verschiedenste Netzwerke, etwa Ethernet auf Leitrechnerebene und Feldbusse wie *Profibus* oder *CAN* (*controller area network*) auf Zellen- bzw. Maschinenebene.
  - Extreme Störeinfüsse.
  - Netzwerke verbinden oftmals schwer zugängliche Nodes (v.a. intelligenten Sensoren), wodurch die Leitungsführung schwierig ist.

  Angesichts des sehr gut entwickelten Marktes der Automatisierungssysteme hat es nun —im Sinne der Wirtschaftsnähe— keinen Sinn, eine Lösung zu entwickeln, die nur in einer ganz bestimmten Systemarchitektur verwendbar ist. Das zu entwickelnde Konzept sollte vielmehr für möglichst alle der obigen Charakterisierung genügenden Systeme Gültigkeit haben und mit vertretbarem Aufwand realisierbar sein.

  Übrigens machen der erste sowie die letzten beiden der obengenannten Charakteristika klar, daß eine Lösung auf der Basis eines zusätzlichen Verbindungs-Netzwerkes (dedizierter Zeitkanäle) keine brauchbare Alternative darstellt. Ein in der Praxis verwendbares System muß daher mit dem existierenden Netzwerk das Auslangen finden und sollte darüberhinaus auch den Einsatz in intelligenten Sensoren (mit Feldbus-Anschluß) zulassen.

- *Realistische Fehlerannahmen*

  Welche Modell- und Fehlerannahmen sind für die im vorigen Punkt umschriebene Klasse von Automatisierungssystemen sinnvoll?

  In diesem Zusammenhang ist zu bemerken, daß einige der im vorigen Abschnitt erwähnten Modell-Annahmen in der Praxis schwer einzuhalten sind. So liegt etwa (fast) allen Verfahren, die mit byzantinischen Fehlern fertigwerden können, die Annahme zugrunde, daß zur selben Zeit weniger als $k = n/3$ Nodes fehlerhaft sind. Die (on-line) Überprüfung dieser Voraussetzung in der Praxis ist schwierig; in [LZM90] wird dafür z.B. ein (statistische) Selbstdiagnose-System eingesetzt.

Deswegen kommt der Frage nach *graceful degradation* im Falle der Verletzung von Modell-Annahmen besondere Bedeutung zu. Ihr wird jedoch in den einschlägigen Arbeiten praktisch keine Beachtung geschenkt (löbliche Ausnahmen sind [MS85] und [PB91]).

Auf der anderen Seite erlauben die in unserem Fall vorliegenden Broadcast-Netzwerke die Festlegung von Zusatzannahmen (v.a. in bezug auf byzantinische Fehler), die u.U. wesentlich genauere Aussagen über die reale Performance eines Clocksynchronisations-Algorithmus zulassen.

- *Geringer Clocksynchronisations-Overhead*

  Durch welche Maßnahmen kann die durch die Clocksynchronisation verursachte System-Belastung (Nodes und Netzwerk) gering gehalten werden, und zwar insbesondere in den (*low-capacity*) Feldbussen?

3. **Theoretische und experimentelle Analyse**

   Was sind die wichtigsten Kenngrößen des gewählten Algorithmus? Wie sieht deren (realistische) theoretische und experimentelle Analyse aus?

   Wie im vorigen Abschnitt erwähnt sind *worst-case* Resultate für *precision* und teilweise auch für *accuracy* für praktisch alle existierenden Verfahren verfügbar. Allerdings sind manche der den Analysen zugrundeliegenden Annahmen diskussionswürdig. Zum Beispiel sind *continuous clocks* im Falle einer *precision* im μs-Bereich keine realistische Modellierung mehr ([KS89], [VR92]). Es sollte daher eher eine Analyse auf der Basis von *discrete clocks* angestrebt werden, wie sie z.B. in [KS89] versucht wurde.

   Über den *worst-case* hinausgehende Ergebnisse sind so gut wie keine vorhanden. So sind z.B. (außer bei probabilistischen Algorithmen) keinerlei analytische Formeln für die *average case precision* bekannt; experimentelle Resultate (z.B. [KS89]) und Simulationsergebnisse ([PB91]) sind teilweise verfügbar. Es sollte daher im gegenständlichen Projekt versucht werden, derartige Resultate zu erzielen; sie könnten vor allem für die Analyse der *graceful degradation properties* sehr hilfreich sein.

   Nicht zuletzt kommt aber schließlich der systematischen, experimentellen Evaluation der zu entwickelnden Lösungsalternative große Bedeutung zu. Dies ist nicht nur für die Performance-Messungen wichtig; die in [Mil91] publizierten Erfahrungen bei der Entwicklung des NTP zeigen, daß bestimmte Design-Parameter in der Regel einer experimentellen Optimierung bedürfen. Die Untersuchungen sollten wesentlich umfassender als die in [KS89] erwähnten Experimente sein, vor allem hinsichtlich der Dauer und der selektiven Injektion von Fehlern.

   Ebenfalls sehr wichtig —und zwar schon für die Modellbildung— ist eine systematische experimentelle Untersuchung der bei den verwendeten GPS-Uhren auftretenden Fehler.

3.2 Gewählte Methodik

In diesem Abschnitt werden die Ansätze dargelegt, mit denen die soeben aufgeworfenen wissenschaftlichen Fragestellungen angegangen werden sollen. Aufgrund der bereits

getätigten, umfangreichen Vorarbeiten existieren teilweise schon sehr klare Vorstellungen davon.

Grundsätzlich ist, wie bereits erwähnt, eine Zweiteilung der Projektarbeit geplant:

- *Entwicklung der Konzepte*

  Hierunter fallen die theoretischen Grundlagen (Modellierung, Analyse) und die Auswahl des einzusetzenden Clocksynchronisations-Algorithmus. Von zentraler Bedeutung ist schließlich die genaue Spezifikation der zur Erreichung einer Synchronisationsgenauigkeit im μs-Bereich unabdingbar notwendigen Hardware-Unterstützung. Diese soll im Form eines universell einsetzbaren ASICs (*universal time coordinated synchronization unit UTCSU*) bereitgestellt werden.

- *Realisierung einer Prototyp-Implementierung*

  Auf Basis der Spezifikation der UTCSU soll zunächst einmal das ASIC tatsächlich realisiert werden. Mit Hilfe dieses Chips sollte die (spätere) Adaption der Projektresultate für beliebige Netzwerke (v.a. Feldbussysteme) ohne großen Aufwand möglich sein.

  Die Eignung der Konzepte (v.a. der UTCSU) soll schließlich an einer konkreten Prototyp-Implementierung, dem *Network Clock Synchronization Coprocessor NCSC* mit (optionalen) GPS-Funkuhren, demonstriert werden. Rein technisch gesehen stellt der NCSC einen LAN-Coprozessor für Ethernet (802.3) dar, der mit Hilfe eines ASICs die Clocksynchronisations-Software unterstützt; die Einheit soll den originalen, als Piggyback-Board ausgeführten LAN-Coprozessor einer *Force CPU-30* (einer auf dem M68030 basierenden VMEbus-CPU) ersetzen.

Dabei soll die Entwicklung der Konzepte (Modellierung, Analyse) mehr oder weniger parallel zu der Realisierung der konkreten Implementierung erfolgen.

Die daraus resultierenden Vorteile sind:

- Grundsätzlich wird durch die Realisierung der Konzepte sichergestellt, daß keine wichtigen "engineering"-Aspekte übersehen werden; darüberhinaus erfordert die experimentelle Evaluation eine konkrete Implementierung.

- Es können kommerziell erhältliche VMEbus GPS-Uhren eingesetzt werden.

- Es wird ein Ethernet-Controller verwendet, der optional eine deterministische Kollisionsbehandlung (802.3DCR, siehe [PS91], [BFR87]) unterstützt, wodurch als sehr erwünschter Nebeneffekt die Realisierung eines für Echtzeitsysteme sehr gut geeigneten Ethernet-LANs stattfindet.

- Nutzung von Know-How und Ressourcen aus einem anderen FWF-Projekt P8390 (U. Schmid, *Versatile Timing Analyzer* VTA[5], Laufzeit bis Ende 1994). Für letzteres

[5]Ursprünglich war die Entwicklung eines Vorläufers des NCSC durch einen Zusatzantrag im Rahmen des Projektes *Versatile Timing Analyzer* (VTA) geplant, wodurch eine wesentliche Verbesserung der Meßgenauigkeit des VTAs zu erzielen gewesen wäre. Die damaligen Fachgutachter waren jedoch der

wurden insgesamt 4 der erwähnten Force CPU-30 CPUs angeschafft, die durchaus im Rahmen des gegenständlichen Projektes (mit)verwendet werden können; dasselbe trifft auf die bereits existierende Entwicklungsumgebung (Sun Workstation + Software) zu.

Für die im vorigen Abschnitt dargelegten Projektziele sind zum gegenwärtigen Zeitpunkt folgende Lösungsansätze denkbar (wir folgen der dortigen Einteilung):

1. **Externe Synchronisation**

   Die zur Zeit erfolgversprechendste Idee ist, die externe Synchronisation in zwei Teile zu gliedern:

   - Einbeziehung eines extern gelieferten, UTC-synchronen 1 *pps* (*pulse per second*) Signales in die interne Synchronisation, die damit eine globale, UTC-synchronisierte interne Zeit bereitstellen kann.
   - (Nacheilende) Kontrolle der internen Zeit mit Hilfe der externen UTC-Zeit; Heranziehung von Abweichungen zur Fehlererkennung in bezug auf die externe Zeitquelle.

2. **Hohe Synchronisationsgenauigkeit**

   Grundsätzlich erscheint es sinnvoll, ein im wesentlichen der (hierarchischen) Struktur des zugrundeliegenden Netzwerks entsprechendes System von *synchronisation clusters* mit *Stratum-Levels* einzuführen, wie dies auch in NTP ([Mil91]) der Fall ist. Allerdings muß ein Weg gefunden werden, die *intra-cluster precision* klein (d.h., genau) zu halten.

   Der konkret einzusetzende Algorithmus für die (*inter-cluster*) Clocksynchronisation kann durch Kombination/Variation existierender Verfahren für Broadcast-Netzwerke bestimmt werden, wobei folgende Charakteristika maßgeblich sind:

   - *Graceful degradation*
   - Selbst-Parametrisierung (etwa Messung von *average transmission delays*)
   - Initiale Synchronisation und Reintegration (wieder) funktionierender Nodes
   - Geringer Overhead

   Der durch die Clocksynchronisation verursachte Overhead kann durch die folgenden Maßnahmen verringert werden:

   - Unterteilung des Gesamtsystems in *synchronization clusters*.

Meinung, man sollte ein ohnedies recht großes Projekt nicht durch ein "Projekt im Projekt" noch weiter aufblähen. Vielmehr wurde von einem der (anonymen) Gutachter dessen Durchführung —unter Einbeziehung der externen Synchronisation via GPS— im Rahmen eines eigenen Forschungsprojektes angeregt, was hiermit dankbar gewürdigt wird. Nichts.desto trotz wurden damals zum Teil sehr weit gediehene Vorarbeiten geleistet, die dem gegenständlichen Projekt nun zugute kommen könnten; umgekehrt würde dessen Ergebnis doch noch die Erreichung der im Zusatzantrag zum Projekt VTA formulierten Ziele erlauben.

- Partitionierung der Nodes eines *synchronization clusters* in aktive und passive Teilnehmer an der Clocksynchronisation.
- "Piggybacking" der Clocksynchronisation auf den regulären Datenverkehr, wie z.B. in [DB93].

Die zur Erreichung der Synchronisationsgenauigkeit im μs-Bereich unabdingbar notwendige Hardware-Unterstützung der Clocksynchronisation soll in Form eines ASICs (*universal time coordinated synchronization unit UTCSU*) entwickelt werden, welches folgende Funktionen bietet:

- Hilfestellung bei der Reduktion der Variabilitäten im Timestamping von Netzwerkpaketen, de facto die wichtigste Voraussetzung für eine *precision* im μs-Bereich.
- Bereitstellung einer Hardware-Uhr (100 ns Auflösung) mit Möglichkeiten zur *rate* und *state correction*.
- Unterstützung von *synchronization clusters*.
- Support der internen und externen Synchronisation.
- Support von (konsistenten) Timestamps und UTC, inklusive Konversionsfunktionen.
- Diverse Time(out)-Funktionen.
- Implementierung von BIST-Strukturen (*built-in self test*).
- Implementierung von *Boundary scan*-Strukturen.

Das Design unserer UTCSU wird sinnvollerweise von der *clock synchronization unit* (CSU) von [KO87] und natürlich dem NTP ([Mil91]) ausgehen. Vor allem die der CSU zugrundeliegenden Konzepte wie das Timestamping via DMA oder die digitale *rate/state correction* sind in Hinblick auf die angestrebte universelle Verwendbarkeit und leichten Anbindung an existierende Netzwerke als sehr brauchbare Ausgangsbasis zu bewerten.

3. Theoretische und experimentelle Analyse

Neben einer sauberen Modellierung soll für die Lösung auch eine angemessene theoretische Analyse der Performance gefunden werden; die in der Literatur entwickelten Ansätze sollten eine tragfähige Grundlage dafür bilden.

Von wesentlicher Bedeutung ist die Planung und Durchführung der experimentellen Evaluatuion des NCSC. Hierfür ist zunächst ein System zur Langzeiterfassung und Auswertung der Meßwerte notwendig. Darüberhinaus ist aber auch eine ausreichende Anzahl von Nodes und GPS-Uhren erforderlich; der konkrete Bedarf wird im folgenden Abschnitt detailliert.

3.3 Projektdurchführung

Wie schon erwähnt wurden im Rahmen eines anderen FWF-Projektes P8390 (*Versatile Timing Analyzer*) beträchtliche Vorarbeiten zum gegenständlichen Projekt geleistet. So wurde das Konzept eines Vorläufers des NCSC entwickelt, der bereits einige der zuvor beschriebenen Eigenschaften erfüllt; eine gerade entstehende Diplomarbeit (*M. Horauer*, Institut für Computertechnik) ist der Entwicklung der entsprechenden Schaltung (natürlich ohne UTCSU) gewidmet.

Insgesamt sind die Forschungsgebiete der Antragsteller als sehr gute Ausgangsbasis für das gegenständliche Projekt zu bezeichnen (siehe beiliegende Publikationslisten):

- *Dr. Schmid* (*Institut für Automation*) beschäftigt sich schon seit Jahren mit der Theorie von verteilten Echtzeitsystemen und Netzwerken.
- Die Arbeitsschwerpunkte von *DI Loy* (*Institut für Computertechnik*) liegen auf den Gebieten ASIC Design und Feldbus-Systemen.

Darüberhinaus hat der Vorstand des Instituts für Automation, *Prof. G.-H. Schildt*, der sich im Zuge seiner wissenschaftlichen Tätigkeiten sehr intensiv mit Funkuhr-Systemen beschäftigt hat, seine aktive Unterstützung zugesichert.

Des weiteren sprechen auch "infrastrukturelle" Gründe für die Zweckmäßigkeit des Projekts:

- Die bereits erwähnte Möglichkeit der Nutzung der (zum größten Teil aus FWF-Mitteln gekauften) Geräte aus dem Projekt P8390 (*Versatile Timing Analyzer* VTA) ist äußerst sinnvoll und kostensparend.
- Das *Institut für Computertechnik*, dem einer der Antragsteller (*DI Loy*) angehört, ist Teilnehmer am ESPRIT-Projekt *Eurochip* und hat dadurch die Möglichkeit, ASICs zu entwickeln und bei den beteiligten Halbleiterherstellern fertigen zu lassen. Die dafür aufzuwendenden Kosten betragen einen Bruchteil der dafür normalerweise notwendigen Mittel.
- Am *Institut für Technische Informatik*, Abteilung für Softwaretechnologie (*Prof. Kopetz*) wurde die *clock synchronization unit* CSU (im Rahmen des MARS-Projektes, siehe [KO87]) entwickelt. *Prof. Kopetz* erklärte sich bereit, die Entwicklung unserer UTCSU zu unterstützen.
- Am *Institut für Elektrische Meßtechnik*, Abteilung Meßsysteme (*Prof. Schweinzer*) läuft gegenwärtig ein FWF-Projekt P7582-TEC (*Sichere Computersysteme*), welches die Entwicklung einer M88000-Hardware für das MARS-System zum Ziel hat. Da das MARS-System auf der Verwendung der CSU aufbaut, ergibt sich die glückliche Konstellation, daß auch in der Folge dieses Projektes die Entwicklung eines NCSC erforderlich wird. Demzufolge ergibt sich die Möglichkeit, den Synergieeffekt dieser parallelen Bedürfnisse auszunutzen. *Prof. Schweinzer* bzw. *DI A. Steininger* haben sich bereiterklärt, uns bei der Konzeption und Entwicklung des NCSC zu unterstützen.

Der konkrete Zeit- und Arbeitsplan des 2-jährig geplanten Projektes ist aufgrund der bereits geleisteten Vorarbeiten bereits recht klar zu definieren:

- **0. Jahr** (d.h., wenn möglich vor Projektbeginn)
  - *Alle Mitarbeiter*
    * Entwicklung des Gesamtkonzeptes
    * Spezifikation der UTCSU
    * Spezifikation des NCSC
- **1. Jahr**
  - *DI*[6] *G. Natiesta* (Dienstvertrag)
    * Ausarbeitung der Modellierung (U. Schmid)[7]
    * Auswahl/Adaptierung sowie Analyse des konkreten Algorithmus für die Clocksynchronisation (U. Schmid)
    * Implementierung des Clocksynchronisations-Algorithmus (U. Schmid)
  - *DI*[8] *M. Horauer* (Dienstvertrag)
    * Entwicklung der UTCSU (D. Loy)
    * Entwicklung des NCSC (D. Loy)
  - *NN1* (Diplomarbeit, Förderungsstipendium)
    * Entwicklung der Meß-Hardware und Software (D. Loy)
  - *NN2* (Diplomarbeit, Förderungsstipendium)
    * Experimentelle Fehlererfassung bei den GPS-Uhren (G.-H. Schildt)
- **2. Jahr**
  - *DI G. Natiesta* (Dienstvertrag)
    * Implementierung Clocksynchronisations-Algorithmus (Fortsetzung)
    * Planung und Durchführung der experimentellen Evaluation (U. Schmid)
  - *DI M. Horauer* (Dienstvertrag, 0.5 Jahre)
    * Entwicklung der UTCSU (Fortsetzung)
    * Entwicklung des NCSC (Fortsetzung)

Diesem Arbeitsplan liegt die schon erwähnte Zweiteilung der Projektarbeit zugrunde:

- *Modellierung, Software-Aspekte, Analyse und Evaluation* (DV G. Natiesta, 2 Jahre)

  Die Modellierung sowie die Implementierung der Clocksynchronisations-Software sollten innerhalb von 1.5 Jahren abzuschließen sein. Für die Planung und vor allem Durchführung der experimentellen Langzeit-Evaluation ist ein weiteres halbes Forschungsjahr erforderlich.

[6]Herr Natiesta ist gegenwärtig dabei, seine Diplomarbeit fertigzustellen.
[7]In Klammern ist der primär zuständige Betreuer angegeben.
[8]Auch Herr Horauer ist gegenwärtig dabei, seine Diplomarbeit fertigzustellen.

- *Realisierung des NCSC* (DV M. Horauer, 1.5 Jahre)

  Die Realisierung des NCSC beinhaltet vor allem die Entwicklung des ASICs der UTCSU, sodaß —nicht zuletzt auch wegen der beträchtlichen Einarbeitungszeit— für diese Tätigkeiten sicherlich mehr als ein Jahr veranschlagt werden muß.

Die Teilaufgaben der

- *Entwicklung der Meß-Hardware und Software* (Förderungsstipendium NN1, 1 Jahr)
- *Experimentelle Fehlererfassung GPS-Uhren* (Förderungsstipendium NN2, 1 Jahr)

können im Rahmen von Diplomarbeiten abgewickelt werden. Aufgrund der anspruchsvollen Aufgabenstellungen und des damit zu erwartenden, über das gewöhnliche Maß hinausgehenden Aufwandes sollte jeweils ein Förderungsstipendium vergeben werden; konkrete Kandidaten können erst zu einem späteren Zeitpunkt nominiert werden.

4 Forschungsstätte und Förderungsmittel

4.1 Ort der Forschung

Das Projekt soll in den Räumen und unter Nutzung der Infrastruktur des

- *Instituts für Automation*, Abteilung für Automatisierungssysteme (*Prof. Schildt*)
- *Instituts für Computertechnik* (*Prof. Dietrich*)

durchgeführt werden; entsprechende Einverständniserklärungen liegen bei.

4.2 Personalressourcen

4.2.1 Vorhandenes Personal

An der Forschungsstelle arbeiten folgende Personen in dem Projekt mit:

- *Univ. Ass. Dr. Ulrich Schmid* (*Institut für Automation*)
- *O.Prof. Dr. G.-H. Schildt* (*Institut für Automation*)
- *Univ. Ass. DI Dietmar Loy* (*Institut für Computertechnik*)

4.2.2 Beantragtes Personal

Laut Arbeitsplan im Abschnitt 3.3 werden folgende zusätzliche Stellen benötigt (die Begründung ist dort zu finden):

- Im **1. Jahr**:
  - Dienstvertrag *DI G. Natiesta*
  - Dienstvertrag *DI M. Horauer*

- Förderungsstipendium *NN1* (5.000,-/Monat)
- Förderungsstipendium *NN2* (5.000,-/Monat)

- Im **2. Jahr**:
  - Dienstvertrag *DI G. Natiesta*
  - Dienstvertrag *DI M. Horauer* (0.5 Jahre)

Die folgende Tabelle faßt die dadurch entstehenden Kosten zusammen:

Pos	Ges	1J	2J	Bezeichnung	öS (incl.)
1	1.5	1	0.5	Dienstvertrag *DI Horauer*, öS 260.000,-/Jahr	390.000,-
2	2	1	1	Dienstvertrag *DI Natiesta*, öS 260.000,-/Jahr	520.000,-
3	1	1	-	Förderungsstip. NN1, öS 5.000,/-Monat	60.000,-
4	1	1	-	Förderungsstip. NN2, öS 5.000,-/Monat	60.000,-

4.3 Sachausstattung

4.3.1 Vorhandene Sachausstattung

An den beiden, die Forschungsstelle konstituierenden Abteilungen sind alle für das Projekt notwendigen, zur Infrastruktur gehörenden Geräte vorhanden.

Insbesondere existiert am Institut für Computertechnik eine Workstation mit der ASIC-Entwicklungsumgebung der Firma *Cadence* (USA), die von dem gegenständlichen Projekt mitbenutzt werden kann.

Darüberhinaus sind folgende, zum größten Teil aus den FWF-Mitteln für das Projekt P8390 (*Versatile Timing Analyzer*) finanzierten Geräte vorhanden, die unmittelbar im gegenständlichen Projekt (mit-)genutzt werden können:

Anz.	Bezeichnung
1	Sun Sparcstation IPC
1	C-Cross-Entwicklungsumgebung (MCC68K, XRAY+) für M68030
4	Force CPU-30ZBE M68030 CPUs (in einem VME-Rack)
x	pSOS^{+m} Multiprozessor-Betriebssystem für CPU-30

4.3.2 Beantragte Geräte

Zunächst sind für das Projekt mindestens zwei *VME*[9] *GPS Funkuhren* notwendig, die in das VME-Rack eingesteckt und somit von (jeweils) einer der vorhandenen CPU-30 bedient werden können. Die Anzahl von zwei ergibt sich daraus, daß das Projekt vor allem die fehlertolerante Einbeziehung externer Zeitquellen untersucht, was ohne Replikation klarerweise unmöglich ist. Wegen der notwendigen experimentelle Erfassung der Fehlermöglichkeiten von GPS-Uhren werden beide noch im 1. Jahr benötigt.

[9]Unseres Wissens nach gibt es außer der Firma *Lange* (Produkte von *Bancomm*/USA) keinen Anbieter von VME-kompatiblen GPS-Funkuhren.

Im zweiten Förderungsjahr wird des weiteren eine zusätzliche *Force CPU-30* für die experimentelle Evaluation benötigt. Insgesamt sind nämlich mindestens 5 Nodes für einen realistischen Test erforderlich:

- Unser Ansatz basiert auf mehreren *synchronization clusters*, also brauchen wir mindestens zwei davon.
- Aus der Theorie ist bekannt, daß ein einziger byzantinischer Fehler nur in einem Netzwerk von mindestens 4 Nodes ausgeschaltet werden kann. Eines der *synchronization cluster* muß also mindestens 4 Nodes haben.

Aus dem Projekt *Versatile Timing Analyzer* sind jedoch nur vier CPU-30 vorhanden.

Für die experimentelle Evaluation ist dann noch die Bereitstellung eines speziellen, programmierbaren *Meßsystems* zur Erfassung externer Ereignisse notwendig. Auch hier wurden bereits wichtige Vorarbeiten im Projekt P8390 (*Versatile Timing Analyzer*) geleistet, nämlich das Detailkonzept einer Hard- und Software für die hochgenaue Erfassung externer ("sporadischer") Ereignisse. Das zugrundeliegende Meßprinzip ist nicht kontinuierliches Sampling (wie bei Logikanalysatoren), sondern dedizierte Ereigniserfassung.

Die experimentelle Evaluierung des NCSC erfordert nämlich Messungen, die mit vorhandenen Logikanalysatoren nicht durchführbar sind: Die NCSCs liefern ein digitales 1 *pps* Signal, d.h., sie zeigen periodisch ihren jeweiligen Sekundenbeginn z.B. durch eine steigende Flanke auf einer Leitung an. Erfaßt werden sollte nun —neben vielen anderen wichtigen Größen— die Zeit zwischen der positiven Flanke der jeweils "frühesten" und "spätesten" Signale, mit einer Auflösung von 100 *ns* oder besser. Diese Meßwerte müssen über sehr lange Zeiträume (Monate) hinweg kontinuierlich erfaßt und (laufend) etwa in Form eines Histogramms ausgegeben werden.

Es erscheint daher sinnvoll, das benötigte Meßsystem auf Basis des existierenden Konzeptes zu realisieren. Dieses sieht eine Konfiguration bestehend aus einer *Workstation* für die Datenanalyse und Visualisierung in Verbindung mit einer speziellen *Datenerfassungseinheit* vor. Letztere besteht aus einem *VME-Rack* mit einer *VME-CPU* für die notwendige Vorverarbeitung und mehreren (die Anzahl hängt von den zu erfassenden Signalen ab) eigenentwickelten *M-Modulen*[10] auf einem standardmäßigen *VME-Trägerboard*.

Für die existierende Sun-Workstation sollte daher die *Analyse- und Visualisierungs-Software LabVIEW* angeschafft werden, die eine komplette Umgebung für die (graphische) Programmierung von Meßaufgaben bereitstellt[11]. Für die Datenerfassungs-Hardware wird sinnvollerweise eine weitere *Force CPU-30* eingesetzt; für diese CPU liegt die Software-Entwicklungsumgebung ja fix und fertig vor. Die Kopplung zwischen Workstation und Datenerfassungseinheit erfolgt mittels Ethernet (TCP/IP). Die für die Signalerfassung nötigen M-Module müssen selbst entwickelt werden (siehe dazu Abschnitt 4.4).

Insgesamt sind daher folgende Geräte notwendig:

[10] Wir haben mit dem herstellerunabhängigen und kostengünstigen I/O-System der M-Module bisher sehr gute Erfahrungen gemacht. Die offengelegte Schnittstellen-Definition erlaubt die Entwicklung eigener Module mit relativ wenig Aufwand, und entsprechendes Know-How ist bereits vorhanden.

[11] Es gibt unseres Wissens derzeit kein mit LabVIEW vergleichbares Produkt.

Pos	Ges	1J	2J	Bezeichnung	öS (incl.)
5	2	1	1	CPU-30ZBE, à öS 53.964,–	107.928,–
6	1	1	–	5-slot VME-Rack, à DM 2.240,–	18.816,–
7	1	1	–	LabVIEW Software für Sun Sparcstation	84.348,–
8	2	2	–	VME GPS Funkuhr, à DM 10.200,–	171.360,–

4.4 Beantragtes Material

Materialkosten fallen zunächst einmal für die Entwicklung und Fertigung des NCSC an:

- Entflechtung und Herstellung eines Multilayer-Boards inklusive der Bestückungskosten (SMD-Technik!); hier muß ein vollständiges Redesign eingerechnet werden.
- Bauteilkosten für 7 NTSU-Boards (1 Prototyp + 6 für alle Force CPU-30), die sich in etwa auf öS 4.000,–/Board belaufen.

Weiters sind Materialkosten für die Entwicklung des speziellen M-Moduls für die Datenerfassungseinheit des Meßsystems erforderlich:

- *Entflechtung und Herstellung* der (doppelseitigen) Leiterplatte der speziellen M-Module; hier sollte ebenfalls ein Redesign eingerechnet werden.
- *Bauteilkosten* für fünf derartige M-Module (1 Prototyp + 4 für insgesamt 8 erfassbare Signale), diese belaufen sich in etwa auf öS 500,–/Board.
- Eine standardmäßige *VME-Trägerplatine* für 4 M-Module.

Die resultierenden Kosten sind in der folgenden Tabelle zusammengefaßt:

Pos	Ges	1J	2J	Bezeichnung	öS (incl.)
9	2	2	–	Herstellung NCSC-Board, à öS 20.000,–	40.000,–
10	7	7	–	Bauteilkosten NCSC, à öS 4.000,–	28.000,–
11	1	1	–	MEN A201N Träger f. M-Module, à DM 760,–	6.384,–
12	2	2	–	Herstellung M-Modul Platine, à öS 5.000,–	10.000,–
13	5	5	–	Bauteilkosten M-Module, à öS 500,–	2.500,–

4.5 Reisekosten

Die aus unserer Sicht sehr wichtige Präsentation der Projektergebnisse auf wissenschaftlichen Tagungen erfordert auch Reisekosten. Unter der Voraussetzung, daß die an der TU-Wien existierenden Reisekostenzuschüsse nicht (wie schon oft) ausfallen, werden projektspezifische Reisekosten für *zwei internationale Konferenzen* —eine davon in den USA— ausreichen:

Pos	Ges	1J	2J	Bezeichnung	öS (incl.)
14	1	1	–	Reisekosten Tagung (USA)	30.000,–
15	1	–	1	Reisekosten Tagung	20.000,–

4.6 Beantragte Sonstige Kosten

Für die Entwicklung des ASICs UTCSU sind natürlich ebenfalls finanzielle Mittel erforderlich. So ist zunächst für einen der beantragten Mitarbeiter ein *Einschulungskurs* in die Entwicklungs-Werkzeuge (bei Firma *Cadence* in München) sowie ein Satz Dokumentation notwendig.

Darüberhinaus erscheint es zweckmäßig, sich der beratenden Tätigkeit des Entwicklers der CSU ([KO87]), *Dr. Wilhelm Ochsenreiter*, zu versichern. Dieser ist mittlerweile in der Privatwirtschaft tätig und sollte mit einem *Konsulentenhonorar* für seine Mitarbeit entschädigt werden; auf diese Weise kann der Designprozeß sicherlich wesentlich rascher und effektiver abgewickelt werden.

Ebenfalls notwendig sind dann natürlich die finanziellen Mittel für die *Fertigung einer Kleinserie* (20 Stück) des ASICs. In den dafür zu kalkulierenden Mitteln sind zu berücksichtigen:

- Chipkomplexität ($\Longrightarrow$ Chipfläche), wobei die UTCSU sicherlich ein sehr komplexer Baustein ist (etwa 15000 Gatteräquivalente). Die im Rahmen von ESPRIT *Eurochip* am ehesten in Frage kommende 1.5 μm CMOS-Technologie läßt ca. 300 Gatter/mm^2 Chipfläche zu, wobei die Kosten 110 ECU/mm^2 (excl. Mwst.) betragen.
- Die Implementierung des *boundary scan* Supports erhöht die notwendige Chipfläche um ca. 10%.
- Gehäuse (Anzahl der Pins), wobei aufgrund der erwünschten externen Erweiterbarkeit ("Kaskadierbarkeit") ca. 100 Pins notwendig werden; hierfür muß eine zusätzliche Chipfläche von 12 mm^2 für das Bonding eingerechnet werden.
- Ein komplettes Redesign, das bei so komplexen Bausteinen erfahrungsgemäß immer eingerechnet werden muß.

Die folgende Tabelle enthält die daraus resultierenden Kosten (1 ECU $\approx$ 15 öS):

Pos	Ges	1J	2J	Bezeichnung	öS (incl.)
16	1	1	–	Einschulungskurs (Firma *Cadence*, München)	30.000,–
17	1	1	–	Satz Dokumentation für ASIC Design-Tools	20.000,–
18	1	1	–	Konsulentenhonorar *Dr. Ochsenreiter*	20.000,–
19	2	2	–	Fertigung ASIC, à öS 132.500,–	265.000,–

5 Danksagung

Wir möchten es nicht versäumen, denjenigen zu danken, die frühere Versionen dieses Projektantrages gelesen und Verbesserungsvorschläge gemacht haben: *Prof. D. Dietrich* (TU-Wien), *Prof. W. A. Halang* (Fernuniversität Hagen), *Prof. G.-H. Schildt* (TU-Wien), Prof. *H. Schweinzer* (TU-Wien) sowie *DI A. Steininger*, *DI S. Stöckler*, *DI H. Haberstroh* und *DI W. Kastner* (alle TU-Wien).