TU

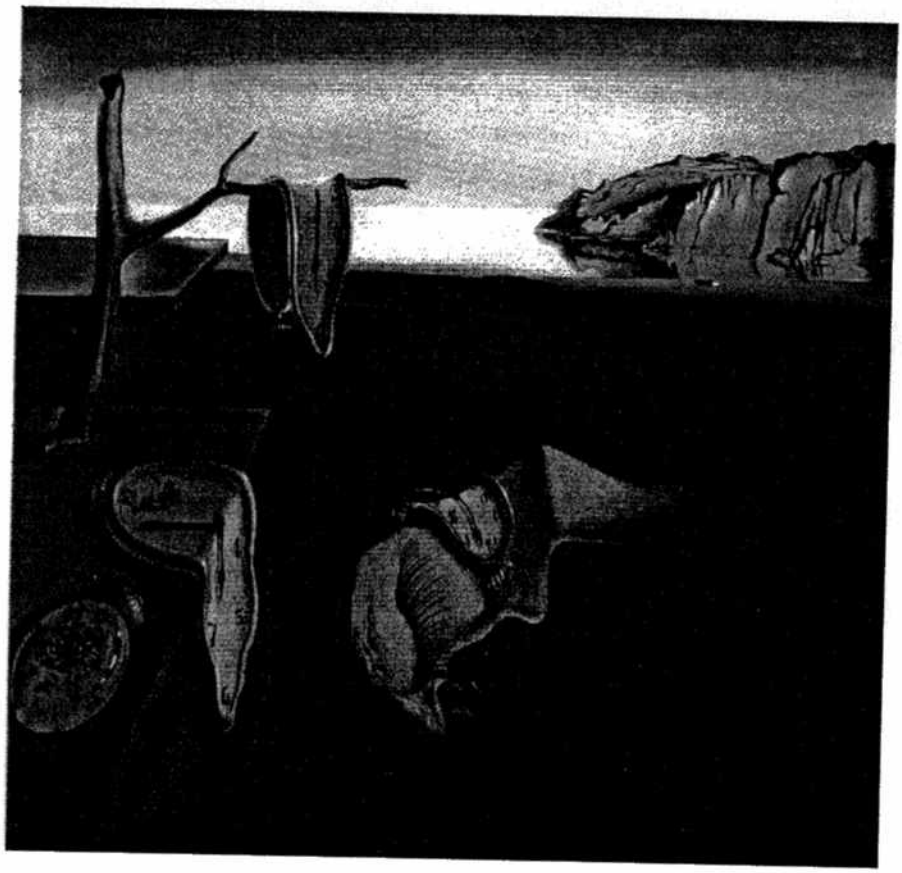
Institut für Automation
Abt. für Automatisierungssysteme

Technische
Universität
Wien

Projektbericht Nr. 183/1-62
August 1995

Untersuchung des Zeitverhaltens Verteilter Echtzeitsysteme

W.A. Halang, F.-J. Koller, U. Schmid and M. Witte



Salvador Dali, "Die Beständigkeit der Erinnerung"

UNTERSUCHUNG DES ZEITVERHALTENS VERTEILTER ECHTZEITSYSTEME

Prof. Dr. Dr. W. A. Halang, Prof. F.-J. Koller, Univ.-Doz. Dr. U. Schmid und Dipl.-Ing. M. Witte*

KURZFASSUNG

An independent test facility is described, which simulates the environments of embedded real-time systems with special emphasis on the exact modeling of the prevailing time conditions. Its main application areas are software verification and safety licensing. Following the black-box approach, just by providing worst-case oriented input patterns to integrated hardware-software-systems and monitoring the corresponding outputs, the time behaviour of such systems can be determined precisely. The concept extends in a straightforward way to the examination of distributed systems by providing one of these test facilities for each node. This became feasible by employing hardware-supported high-accuracy timers synchronised with legal time, viz., Universal Time Coordinated, as received via satellite from GPS, the global navigation and positioning system. Thus, complicated and imprecise clock synchronisation in software is rendered obsolete.

EINLEITUNG

Qualitätssicherung von Software ist ein schwieriges und bisher noch nicht zufriedenstellend gelöstes Problem. Es verschärft sich für — insbesondere verteilte — Echtzeitanwendungen, bei denen nicht nur Programmkorrektheit im Sinne klassischer Datenverarbeitung, sondern auch das Zeitverhalten integrierter Hardware-Software-Systeme verifiziert werden muß. Dies gilt umso mehr für sicherheitsgerichtete Systeme, die von den Technischen Überwachungsvereinen abzunehmen sind.

Die beim Nachweis richtigen Echtzeitverhaltens auftretenden Probleme sind vielfältig. Nur in trivialen Fällen und bei Verzicht auf Betriebssysteme mag es möglich sein, das Zeitverhalten durch Programmanalyse vorherzusagen. Die weit verbreitete Praxis, Programme zu Testzwecken mit Ausgabeanweisungen zu instrumentieren, um so Informationen über Zwischenzustände zu erhalten, ist für die Zeitanalyse unbrauchbar, da sie das zu studierende Zeitverhalten ändert. Weiterhin erweisen sich Tests in realen Anwendungsumgebungen häufig als zu teuer, zu gefährlich oder aus anderen Gründen als unmöglich. Deshalb müssen solche Umgebungen nachgebildet werden, und zwar nach dem Prinzip des schwarzen Kastens, d.h. die zu untersuchenden Systeme dürfen unter keinen Umständen modifiziert werden. Um maximale Objektivität sicherheitstechnischer Abnahmen zu garantieren, beschränkt sich die (externe) Beobachtung allein darauf zu prüfen, ob die Zeitpunkte und Werte erzeugter Ausgaben den gegebenen Anforderungen entsprechen.

In diesem Artikel wird gezeigt werden, daß sich die Umgebung eines Testlings effektiv nachbilden läßt und daß dessen Zeitverhalten mit herkömmlichen Prozeßrechnern erfaßt

und überwacht werden kann, sofern letztere um einige Hardware- und Software-Elemente ergänzt werden. Die auf dem Testling implementierte Software wird durch Testpläne überprüft, die externe Stimuli und deren Auftrittszeitpunkte vorgeben. Diese Testpläne sind von der untersuchten Software völlig unabhängig und werden auf anderen Rechnern ausgeführt. Daher sehen wir in der vorgestellten Methode einen ernsthaften Kandidaten zur Anerkennung für offizielle sicherheitstechnische Abnahmen.

FUNKTIONEN DER TESTUMGEBUNG

Im Gegensatz zum mehr oder minder klassischen *white-box testing*, bei dem Informationen über die Implementierung der Testlinge benötigt werden, erlaubt das *black-box testing* Tests allein auf der Basis von Spezifikationen (siehe [1] für eine Einführung). Da somit

- die Hard- und Software zur Testdatengenerierung und -auswertung völlig unabhängig von der Implementierung der Testlinge ist, was eine Erstellung durch Dritte — und das sogar gleichzeitig mit der Implementierung — erlaubt, und
- die Test an den in den Spezifikationen festgelegten externen (Standard-) Schnittstellen ansetzen, was den Einsatz universeller Testumgebungen nahelegt,

erscheint der hier verfolgte Ansatz für sicherheitstechnische Abnahmen besonders geeignet zu sein.

Nun wurden natürlich einige der hierfür notwendigen "Ingredienzien" bereits relativ früh für ähnliche Zwecke eingesetzt. So ist es etwa in einigen spezialisierten Bereichen der Industrie durchaus üblich, Simulatoren zum Test von Prozeßrechensystemen[1] einzusetzen; für einige Beispiele sei auf [8,3,4,6,5] verwiesen.

In der wissenschaftlichen Literatur wurde darüberhinaus auch der Einsatz von Erfassungssystemen für die Beobachtung des Verhaltens von Rechensystemen propagiert, insbesondere für die Fehlersuche in verteilten Systemen (siehe [7] für eine Übersicht). Allerdings macht es die Interaktivität dieses Suchprozesses beinahe unabdingbar, Mechanismen zur Ablaufwiederholung (siehe z.B. [14]) vorzusehen. Gerade im Bereich der Echtzeitsysteme ist jedoch auch die reine Erfassung sinnvoll (siehe [9]), und zwar besonders zur Verifikation der Einhaltung logischer und vor allem zeitlicher Bedingungen.

Es gibt allerdings nur wenige Publikationen (wie etwa [13,12]) über integrierte Testumgebungen mit Erfassungs- und Simulationskomponenten. Die erwähnten Arbeiten sind auf spezielle Architekturen hin zugeschnitten und hauptsächlich zum White-Box-Testing ausgelegt. Eine gute Übersicht über diese — in [2] im Jahre 1980 konstatierte und noch immer existierende — "verlorene Welt" der Praxis des industriellen Software-Testens im Echtzeitbereich ist in [11] zu finden.

Gemäß der in der Einleitung erwähnten Anwendungsbedingungen muß eine für sicherheitstechnische Abnahmen taugliche Testumgebung folgende Dienste bereitstellen:

- Simulation der Umgebung, in der ein Testling arbeitet, durch — *reproduzierbare* — Erzeugung an typischen und Worst-Case-Bedingungen orientierter Eingabemuster,

[1] Vom bereits klassischen Testen von (VLSI-) Hardware ganz abgesehen, das ohne Simulation undenkbar ist.

- Überwachung, ob die vom Testling erzeugten Ausgaben korrekt sind und innerhalb gegebener Zeitschranken auftreten,
- keinerlei Beeinflussung des Testlings, insbesondere keine Verlängerung seines Programm-Codes und seiner Ausführungszeit,
- Verwendung derselben Hardware-Schnittstellen zum Testling wie in der eigentlichen Prozeßumgebung,
- Zugang zur *gesetzlichen Zeit* (UTC) zur genauen Zeiterfassung von Ereignissen, zur zeitgenauen Simulation externer Ereignisse und um bei der Untersuchung verteilter Systeme erfaßte Daten korrekt miteinander in zeitliche Beziehung setzen zu können,
- einfache Programmierung (Konfigurierung) beliebiger Testpläne,
- Erzeugung leicht lesbarer, knapper Berichte über die Testergebnisse.

HARDWARE-ARCHITEKTUR DER TESTUMGEBUNG

Wir bauen gerade den Prototyp einer die obigen Anforderung erfüllenden Testumgebung. Für einen Überblick über die Hardware-Struktur der Einheit verweisen wir auf Abb. 1.

Entlang des internen E/A-Busses eines handelsüblichen Prozeßrechners gruppiert sehen wir ein Videoterminal zum Betrieb der Einheit, einen Drucker für Berichte und ein Massenspeichergerät vor. Letzeres kann außer den intern benötigten Dateien auch größere Datenmengen vorhalten, die als Eingaben für den Testling benötigt werden. Gemäß der Ankopplung des Testlings an seine Umgebung ist die Simulationseinheit mit Prozeßperipheriegeräten wie digitalen Schnittstellen und analogen Umsetzern, wie auch IEC 488-Bus- und verschiedenen seriellen Leitungsschnittstellen ausgestattet. Da die Anzahl und der Typ dieser Verbindungen je nach den verschiedenen untersuchten Systemen variiert, gibt es die Möglichkeit, entsprechende Peripheriegeräte als E/A-Einschübe in der Einheit bereitzustellen. Alle externen Leitungen werden auf geeignete Normstecker geführt, um den einfachen Anschluß an verschiedene Testlinge zu ermöglichen.

Im Gegensatz dazu sind die jetzt erwähnten weiteren Zusatzgeräte immer vorhanden. Eine bidirektionale Schnittstelle am E/A-Datenbus der Testumgebung erlaubt die Simulation anderer Peripheriegeräte. Ihre Adressen sind frei wählbar. Es wird eine Anzahl unabhängig und parallel arbeitender Register bereitgestellt, in denen der Prozeßrechner zu beobachtende Adressen ablegen kann. Die Ausgänge dieser Register und die überwachten E/A-Adreßbusleitungen des Testlings werden durch entsprechende Hardware-Komparatoren verglichen, die im Falle von Übereinstimmungen Signale an den Prozeßrechner senden.

Eine ähnliche Baugruppe wird mit der Systemuhr kombiniert, um genau zu den in den Testplänen spezifizierten Zeitpunkten Signale zu erzeugen, mit denen dann die Unterbrechungseingänge des Testlings angeregt werden. Der Prozeßrechner lädt immer die nächste dieser Unterbrechungszeitpunkte in das Vergleichsregister der Baugruppe. Bei Gleichheit mit der Uhr löst der Komparator ein Signal aus, das an den Prozeßrechner, wo es eine zugehörige Reaktion auslöst, und an alle Unterbrechungsleitungen, deren entsprechende Bits im Maskenregister gesetzt sind, weitergeleitet wird. Diese Funktionseinheit zur Unterbrechungsgenerierung mit genauem Zeitverhalten ist für ein Gerät zur Software-Validierung

unentbehrlich, da die Umgebung eines Testlings so genau wie möglich nachgeahmt werden muß. Letzteres können konventionelle Rechneruhren, die immer einen Software-Anteil aufweisen, nicht leisten, weshalb ihr Einsatz i.a. zu fehlerhaften Testergebnissen führt. Die Baugruppe wird nicht nur eingesetzt, um Unterbrechungssignale in einen Testling einzugeben, sondern auch, um Dateneingaben nach vorgegebenen zeitlichen Verteilungen auszulösen.

SOFTWARE-KOMPONENTEN DER TESTUMGEBUNG

Die Simulationseinheit ist mit einem Echtzeitbetriebssystem für den Mehrprozeßbetrieb und einer Programmumgebung ausgestattet, die als Hauptkomponente eine höhere, durch einige Elemente erweiterte Ablaufkontrollsprache enthält, die es erlauben, die spezielle Hardware anzusprechen. In dieser Sprache werden Testpläne geschrieben, die die von den Testlingen zu erfüllenden Bedingungen formal spezifizieren. Testpläne enthalten für jede zu simulierende Unterbrechungsquelle einen Zeitplan, der periodische oder sporadisch verteilte Unterbrechungen gemäß der Bedingungen für den schlimmsten Fall erzeugt. Die Identifizierungen und Auftretenszeitpunkte dieser Unterbrechungen werden für den späteren Gebrauch in Ausführungsberichten festgehalten. Gemäß der zeitlichen Muster dieser Unterbrechungssignale stellt der Testplanprozessor geeignete Daten an den verschiedenen Eingabeschnittstellen des Testlings bereit. Analog spezifizieren Testpläne, wann und welche von den untersuchten Systemen erzeugte Ereignisse erwartet werden und welche Reaktionen nach ihrem Auftreten auszuführen sind.

Mit Hilfe der E/A-Adreßvergleichsbaugruppen werden die Eingänge der Testumgebung überwacht, um die Reaktionszeiten des Testlings zu bestimmen und festzustellen, ob er korrekte Ausgabewerte liefert. Zusammen mit ihren Quellen und Ankunftszeiten werden diese Daten auch aufgezeichnet, um für die Abfassung der endgültigen Berichte ausgewertet zu werden. Da nur die externen Reaktionen auf eine gegebene Arbeitslast betrachtet werden, berücksichtigt die Simulationsmethode auch den Aufwand des einer Anwendung unterliegenden Betriebssystems. Mit der möglichen Verbindung der E/A-Busse des testenden und des getesteten Rechners können alle Arten von Peripheriegeräten, einschließlich DMA-Einheiten, simuliert werden. Zur Durchführung muß ein Testplan nur eine geeignete Datenquelle oder -senke und eine angemessene Übertragungsrate spezifizieren. Eine weitere nützliche Funktion, die in Testplänen aufgerufen werden kann, ist die Aufzeichnung des E/A-Busverkehrs oder einzelner E/A-Aktivitäten nach vorgegebenen Zeitspezifikationen. Die dabei erfaßten Daten werden in geeigneten Pufferbereichen der Testumgebung bereitgestellt.

UNTERSUCHUNG VERTEILTER SYSTEME

Die in den vorigen Abschnitten vorgestellte Testumgebung ist für die Untersuchung eines einzelnen Rechners und damit eines Knotens in einem verteilten System geeignet. Das Testprinzip läßt sich in höchst einfacher Weise auf verteilte Systeme übertragen, indem für jeden Knoten eine eigene Testumgebung bereitgestellt wird. Die gewonnenen Ergebnisse sind genau dann aussagefähig und können miteinander in Beziehung gesetzt werden, wenn die lokalen Uhren der einzelnen Testumgebungen synchron laufen. Dies wird durch Bereitstellung eines hochgenauen Zeitgebers in jeder Testumgebung erreicht, der UTC-Zeitsignale von den Satelliten des Global Positioning Systems (GPS) empfängt und der bereits in [15] beschrieben wurde.

Der in Abb. 2 dargestellte Zeitgeber besteht aus einem anwendungsspezifischen Schaltkreis (ASIC)[2], einem GPS-Empfänger mit Antenne und einem Mikrocontroller. Die vom GPS-Empfänger gelieferte Information enthält u.a. die gesetzliche Zeit UTC mit einer Genauigkeit besser als 100 ns. Über eine serielle Schnittstelle wird sie an den Mikrocontroller übertragen. Der Prototyp unseres Weckers hat eine Auflösung von 100 μs. Er wird von einem freilaufenden Oszillator getaktet und jede Sekunde durch das vom GPS-Empfänger gelieferte "Time-Mark-Signal", das eine Genauigkeit von ± 1 μs hat, mit UTC synchronisiert. Zur Durchführung des in Testumgebungen notwendigen Zeitstempelns wurde dieser Zeitgeber um eine Signalempfangseinheit erweitert. Wenn ein Signal auf einer der Leitungen $I_1 \ldots I_n$ eintrifft, wird aus der Ankunftszeit und der kodierten Leitungsnummer ein Zeitstempel erzeugt. Dieser wird in den Ereignis-FIFO eingegeben und dann an den Mikrocontroller zur Weiterverarbeitung übertragen.

LITERATUR

[1] R.A. DeMillo, W.M. McCracken, R.J. Martin, J.F. Passafiume: *Software Testing and Evaluation.* Benjamin Cummings Publ., 1987.

[2] R.L. Glass: "Real-Time: The "Lost World" of Software Debugging and Testing", *Comm. ACM*, 23, 5, 264–271, 1980.

[3] G. Hauser: "Test von Steuerungssoftware mit Hilfe der Realzeitsimulation", in F. Breitenecker, W. Kleinert (Hrsg.), *Informatik-Fachberichte* 85, pp. 495–499, Springer-Verlag, 1984.

[4] K. Hellmold: "Multiprozessorsystem für die parallele Simulation von zeitdiskreten Systemen", in M. Goller (Hrsg.), *Informatik-Fachberichte* 56, pp. 227–233, Springer-Verlag, 1982.

[5] H.B. Keller: "Verteilte/modulare Echtzeitsimulation komplexer Systeme", in W. Ameling (Hrsg.), *Informatik-Fachberichte* 179, pp. 73–83, Springer-Verlag, 1988.

[6] R. Kodweiß: "Software-Konzept für Echtzeit-Simulation", in D. Möller (Hrsg.), *Informatik-Fachberichte* 109, pp. 251–253, Springer-Verlag, 1985.

[7] C.E. McDowell, D.P. Helmbold: "Debugging Concurrent Programs", *ACM Computing Surveys*, 21, 4, 593–622, 1989.

[8] J.M. Mohan, M. Geller: "An Environmental Simulator for the FDNY Computer-Aided Dispatch System", in R.L. Glass (Hrsg.), *Real-Time Software*, pp. 75–90, Prentice-Hall, 1983.

[9] U. Schmid: "Monitoring Distributed Real-Time Systems", *Real-Time Systems*, 7, 33–56, 1994.

[10] U. Schmid: "Synchronized Universal Time Coordinated for Distributed Real-Time Systems", erscheint in *IFAC Control Engineering Practice*, 1995.

[11] W. Schütz: *The Testability of Distributed Real-Time Systems.* Dissertation, Technische Universität Wien, 1992.

[2]Das ASIC wurde in einer CMOS-Standardzellentechnologie von der Firma ES2 hergestellt, was im Rahmen des EUROCHIP-Projektes von der Europäischen Union gefördert wurde.

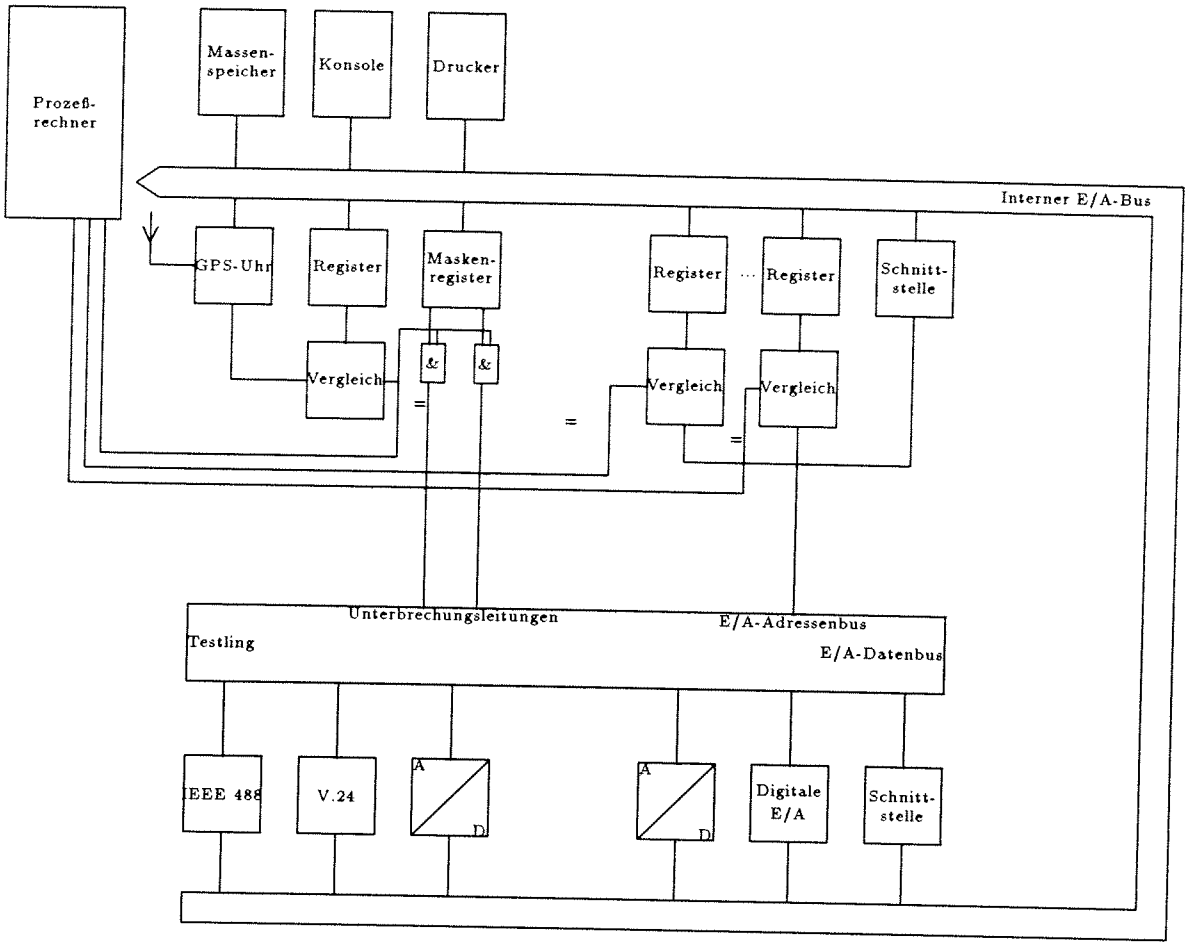


Figure 1: Architektur der Testumgebung

[12] E. Schoitsch, E. Dittrich, S. Grasegger, D. Kropfitsch, A. Erb, P. Fritz, H. Kopp: "The ELEKTRA Testbed: Architecture of a Real-Time Test Environment for High Safety and Reliability Requirements", Proc. *IFAC SAFECOMP '90*, Gatwick, pp. 59–65, 1990.

[13] W. Schütz: "A Test Strategy for the Distributed Real-Time System MARS", Proc. *IEEE CompEuro'90*, Tel Aviv, pp. 20–27, 1990.

[14] J.P. Tsai, K.-Y. Fang, H.-Y. Chen, Y.-D. Bi: "A Noninterfering Monitoring and Replay Mechanism for Real-Time Software Testing and Debugging", *IEEE Transactions on Software Engineering*, SE-16, 8, 897–916, 1990.

[15] M. Wannemacher, W.A. Halang: "GPS-based timing and clock synchronisation for real time computers", *IEE Electronics Letters*, 30, 20, 1653 - 1654, 1994.

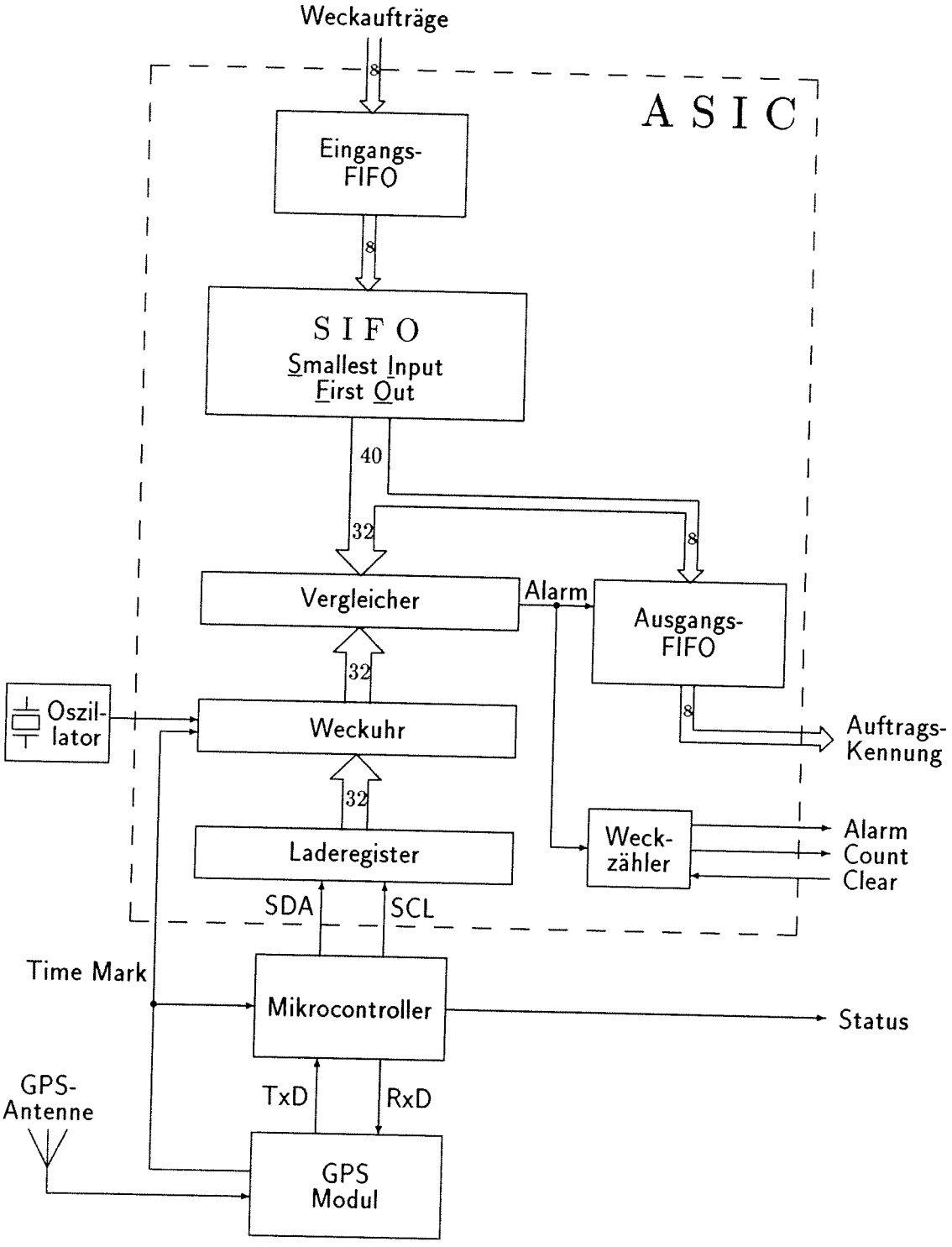


Figure 2: Blockschaltbild des Zeitgebers