Lehrstuhl für Realzeit-Computersysteme

PC-basierte Systemarchitekturen für zeitkritische technische Prozesse

Gregor Burmberger

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. rer. nat. habil. Bernhard Wolf

Prüfer der Dissertation:

1. Univ.-Prof. Dr.-Ing. Georg Färber
2. Univ.-Prof. Dr.-Ing. Dr.-Ing. habil. Friedrich Schneider

Die Dissertation wurde am 17.01.2002 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 24.06.2002 angenommen.

Vorwort

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am *Lehrstuhl für Realzeit-Computersysteme* der Technischen Universität München in den Teilprojekten *Dreidimensionales Drucken von Bauteilen* und *Mikro-Stereolithografie* des Forschungsverbundes Mikrosystemtechnik FORMIKROSYS II der Bayerischen Forschungsstiftung.

Herrn Prof. Dr.-Ing. Georg Färber gilt mein besonderer Dank für das in mich gesetzte Vertrauen. Die belebenden Diskussionen und seine wertvollen Ratschläge trugen wesentlich zum Gelingen dieser Arbeit bei. Herrn Prof. Dr.-Ing. Friedrich Schneider danke ich für sein Interesse an dieser Arbeit und die Übernahme des Zweitberichts.

Ganz herzlich bedanken möchte ich mich bei meinem Kollegen Robert Huber, der mir durch seine Fachkenntnisse ein geschätzter Diskussionspartner war. Die gemeinsame Zeit in unserem Büro werde ich vermissen. Für das kollegiale Arbeitsklima und zahlreiche interessante Gespräche danke ich den Mitarbeitern des Lehrstuhls. Besonders nennen möchte ich hier meine beiden Diplomanden Zoran Vujnovic und Raphael Neidinger, sowie Herrn Thomas Andreas, die wichtige Beiträge zur vorliegenden Arbeit geleistet haben.

Nicht zuletzt schulde ich großen Dank meiner Familie, im speziellen meiner Mutter, die mir das Studium ermöglicht hat, und meiner Lebensgefährtin Christina, die mir immer Beistand geleistet und mich in allen Phasen uneingeschränkt unterstützt hat.

München, im Januar 2002

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Zusammenfassung

Seit einigen Jahren dringt der Personal Computer, ausgehend von seinem ursprünglichen Einsatzgebiet im Büro, verstärkt in den Bereich der Automatisierungstechnik vor. Steuerungssysteme für technische Prozesse sind nicht mehr ausschließlich die Domäne speicherprogrammierbarer Steuerungen und eingebetteter Systeme. Das PC-basierte System etabliert sich zunehmend in der Automatisierung der industriellen Produktion. Die Gründe hierfür liegen vor allem in der schnellen Entwicklung der Informationstechnologie. Die Vorteile für den Anwender sind hohe Verfügbarkeit, Wirtschaftlichkeit, Kommunikationsfähigkeit, Modularität, Systemoffenheit und Flexibilität, sowie hohe Leistungsfähigkeit zur Realisierung komplexer Steuerungsalgorithmen für anspruchsvolle Anwendungen. Durch die Verwendung der PC-Softwarearchitektur können Applikationen einfacher realisiert und neue Aufgaben schneller verwirklicht werden. Standardisierte Kommunikationsmethoden machen Produktionsdaten unternehmensweit verfügbar und schlagen so die Brücke zu integrierten Business-Plattformen wie z.B. *SAP*. Die Nutzung der modularen PC-Hardwarearchitektur mit bewährten Komponenten verschiedener Hersteller ermöglicht eine konsequente Verlagerung des Aufwands von der Systemauswahl hin zur Anwendungsentwicklung und erleichtert wesentlich die Integration in vorhandene Systeme. Der größte Nachteil der Systeme auf Basis von Standard-Soft- und Hardware, ist allerdings die ungenügende *Realzeitfähigkeit*. Firmen und Forschungsgruppen haben zwar eine Vielzahl von unterschiedlichsten Methoden und Lösungen entwickelt, die Qualität dieser Systeme ist aber sehr heterogen und ihre technologischen Grenzen und optimalen Anwendungsgebiete sind oft ungeklärt.

Die vorliegende Arbeit schließt diese Lücke durch die Schaffung einer *Klassifizierung der Systemarchitekturen* zur Realisierung von PC-basierten Steuerungssystemen für zeitkritische technische Prozesse. Mit dieser Klassifizierung ist man in der Lage, direkt nach dem Aufstellen der Realzeitbedingungen einer zu realisierenden Anwendung, schnell und unkompliziert einen passenden Lösungsansatz auszuwählen und sich auf die Implementierung zu konzentrieren. Die Unterteilung der Systemarchitekturen erfolgt in die zwei Hauptklassen *Softwarebasierte Systemarchitekturen* und *Hardwarebasierte Systemarchitekturen* mit insgesamt fünf Unterklassen. Zwei weitere nützliche Kombinationsmöglichkeiten entstehen durch Hinzufügen einer *Ein/Ausgabe-Komponente ohne Eigenintelligenz*. Auf der Basis von nur sieben Systemarchitekturen können, durch umsichtige Auswahl von Komponenten oder durch gezieltes Hinzufügen von Software- oder Hardwareerweiterungen, PC-basierte Steuerungssysteme für zeitliche Anforderungen vom Millisekunden- bis Sub-Mikrosekundenbereich konzipiert werden. Die weiteren Kapitel der Arbeit erläutern die spezifischen Eigenschaften der Systemarchitekturen. Grenzen, Probleme und geeignete Einsatzgebiete werden dabei einer gründlichen Analyse unterzogen. Erweiterte Architekturkonzepte mit Vorschlägen für Verbesserungen sowie neue Schnittstellenarchitekturen und Software-Kommunikationsstrukturen geben einen Ausblick auf die zukünftige Entwicklung. Zwei Anwendungsbeispiele aus der industriellen Praxis, die Entwicklung einer Druckkopfansteuerung eines 3D-Druckers und die Entwicklung einer Scannersteuerung einer Mikro-Stereolithografiemaschine, dienen zur Verifikation der erarbeiteten Klassifizierung. Die Ergebnisse dieser Arbeit können als Grundlage für den automatischen Systementwurf verwendet werden.

1 Einleitung

Der Siegeszug des Personal Computers

Im Privatbereich sowie in der Bürowelt ist der Siegeszug des *Personal Computers* ungebrochen. Auch aus der Management- und Leitebene in der Automatisierungstechnik ist der PC heute nicht mehr wegzudenken. Der PC hat sich auf breiter Basis etabliert. Nach dem Motto „schneller und billiger" werden Hard- und Softwarekomponenten immer leistungsfähiger und gleichzeitig kostengünstiger. Einleuchtend, dass von dieser Entwicklung auch die Industrie profitieren will, um Kosten zu reduzieren und die Produktivität zu steigern. Seit geraumer Zeit sind bestimmte Aufgaben fest in der Hand von PC-basierten Systemen. Mit Anwendungen auf Standardbetriebssystemen wie Windows oder Linux werden Datenbanken für Rezepturen und Maschineneinstelldaten verwaltet und Informationen aufbereitet und visualisiert.

Inzwischen machen sich Hersteller und Anwender Gedanken darüber, wie man weitere Aufgaben in den PC verlagern kann. In jüngster Zeit kommen mehr und mehr PC-basierte Systeme für unterschiedlichste Anwendungen auf den Markt, die kostengünstigere Steuerungssysteme auf nur einer Hardwarebasis versprechen. Tatsache ist, dass sich Aufgaben wie Steuern, Visualisieren, Datenverarbeitung und Kommunikation sowohl mit klassischen Systemen (SPS, eingebettete Systeme) als auch mit reinen PC-basierten Systemen lösen lassen. Was letztlich sinnvoll ist, hängt jedoch von vielen Faktoren ab und ist für jedes Projekt individuell zu betrachten. So können neben Rechenleistung auch die Vorkenntnisse und Vorlieben des Anwenders eine große Rolle spielen. Entscheidend ist insbesondere das Attribut *Realzeitfähigkeit* – eine Eigenschaft, die man bisher eher weniger mit einem PC in Verbindung gebracht hat.

Realzeitfähigkeit von Systemen

Unter Realzeitfähigkeit versteht man den Betrieb eines Rechnersystems in einer Weise, dass Programme zur Verarbeitung anfallender Daten ständig betriebsbereit und Ergebnisse innerhalb einer vorgegebenen Zeit verfügbar sind. „Harte" Realzeit bedeutet in diesem Sinn, dass ein System auf äußere Ereignisse in 100 % der Fälle im vorgegebenen Zeitfenster reagiert. Wenn dagegen die vorgegebenen Reaktionszeiten in Einzelfällen auch überschritten werden dürfen, spricht man von „weicher" Realzeit.

Klassische Steuerungssysteme mit ihren Realzeitbetriebssystemen sind für den rauen Industrieeinsatz konzipiert worden. Sie sind und bleiben daher erste Wahl bei allen zeitkritischen Steuerungsaufgaben in Verbindungen mit höchsten Sicherheitsansprüchen oder bei einfachen, kleinen Systemen, bei denen ein PC-basiertes System schlicht und einfach „eine Nummer zu groß" wäre. Sind neben dem eigentlichen Steuerungsprozess auch weitere Aufgaben, wie beispielsweise Datenverarbeitung, Vernetzung oder Visualisierung gefordert, kommt ein PC-basiertes System in Frage. Das gilt in besonderem Maße, wenn bereits ein (industrietauglicher) PC vorhanden ist und existierende Software weiter genutzt werden soll und kann. Die Standardbetriebssysteme Windows und Linux sind zwar für

sich genommen keine (harten) Realzeitbetriebssysteme, trotzdem erreicht man mit ihnen ausreichend schnelle Reaktionszeiten für den „weichen" Realzeitbetrieb. In Verbindung mit geeigneten Software- und Hardwareerweiterungen können sie sogar zur Realisierung von zeitkritischen „harten" Steuerungssystemen eingesetzt werden. Je nach Aufwand lassen sich so unterschiedliche Zeitanforderungen einhalten. Durch die weitgehende Beibehaltung von Standard-Soft- und -hardware und die damit einhergehende Kompatibilität kann sich das *PC-basierte Realzeitsystem* eventuell als zukünftiger Standard für die Steuerung von zeitkritischen technischen Prozessen durchsetzen.

Gestiegene Ansprüche

Die Ansprüche an Soft- und Hardware zur Steuerung von technischen Prozessen steigen ständig. Kunden fordern Systeme, die folgende Eigenschaften erfüllen:

- **Höhere Rechenleistung**, z.B. zur Realisierung von komplexen Steuerungsaufgaben und Berechnungsalgorithmen.
- **Mehr Speicherkapazität**, z.B. zur Datenerfassung von Signalen für die spätere Auswertung, oder bei Einsatz von ressourcenintensiven Programmiersprachen (Java).
- **Geringere Kosten**, z.B. für den ökonomischen Einsatz von vielen Systemen bei großen Industrieanlagen.
- **Kleinere Ausmaße**, z.B. für den Einsatz in mobilen Systemen im Kfz-Bereich.
- **Vernetzbarkeit**, z.B. für Ferndiagnose und Fernwartung, sowie zur Weiterleitung von Produktionsdaten (Intra-/Internet).
- **Modularisierbarkeit,** z.B. zur vereinfachten Rekonfiguration bei wechselnden Einsatzarten.
- **Verwendung von (Quasi-)Standards**, z.B. aus Kompatibilitätsgründen und wegen der möglichen (Wieder-)Verwendung von bereits vorhandenen Softwaremodulen.
- **Einfache Wartbarkeit**, z.B. für den Ersatz von teuren Technikern „vor Ort" durch Fernwartung.
- **Verringerte Entwicklungskosten**, z.B. für den Einsatz in neuen kostensensitiven Anwendungsgebieten.
- **Grafische Visualisierung**, z.B. für eine anwenderfreundliche Mensch-Maschine-Schnittstelle.

Diese Eigenschaften erfüllen klassische Realzeitsysteme nur teilweise. Gerade die stetig steigende Leistungsfähigkeit der PC-basierten Systeme lässt klassische Systeme weiter ins Hintertreffen geraten. Die Vernetzung ist durch die zunehmende Verbreitung des Internets zu einer unentbehrlichen Eigenschaft von Steuerungssystemen geworden. Aktuelle Daten aus der Produktion werden in Business-Plattform-Systeme, z.B. SAP, eingespeist und sind unternehmensweit abrufbar. Die Controllingabteilungen haben somit eine aktuelle Datenbasis als Grundlage für ihre Analysen. Die Vernetzung ermöglicht auch die Ferndiagnose und Fernwartung von Steuerungssystemen, die bei entfernten Kunden installiert wurden. Kostenintensive Reisen von Technikern können verkürzt oder ganz eingespart werden. Auch die grafische Visualisierung der gesteuerten technischen Prozesse wird immer wichtiger oder ist gänzlich unverzichtbar. So ist z.B. der *Teach-In*-Prozess[1] bei Industrie-Robotern ohne grafische Benutzerführung schwerlich vorstellbar. Hier wurden bisher komplexe klassische Systeme mit proprietärer Visualisierung eingesetzt. Die Entwicklungskosten, speziell für die aufwändige Programmierung der grafischen Oberfläche waren entsprechend hoch. Die Ver-

[1] Zum Beispiel Führung des Roboterarms zu Schweißpunkt-Koordinaten durch einen Produktionstechniker.

wendung eingeführter grafischer Benutzerschnittstellen, wie sie in Standardbetriebssystemen integriert sind, verringert die Entwicklungskosten und erhöht die Akzeptanz des Anwenders. Zu lösen ist jetzt noch das Problem der Realzeitfähigkeit.

Für Steuerungssysteme mit weichen Realzeitanforderungen ist dies einfach zu realisieren. Bei Reaktionszeiten im zehn bis hundert Millisekunden-Bereich kann ohne weiteres ein Standard-PC mit einem beliebigen Standardbetriebssystem verwendet werden. Für erhöhte Anforderungen stehen Software-Erweiterungen oder Realzeitbetriebssysteme zur Verfügung. Die untere Grenze der Reaktionszeit der PC-Hardware ist bei ca. 2 μs erreicht. Eine Unterschreitung dieser Zeit ist aufwändig aber dennoch durch Hardware-Erweiterungen möglich. Hierbei ist jedoch anzumerken, dass die überwiegende Zahl von technischen Prozessen keine solchen extremen Realzeitanforderungen stellt.

Ziele dieser Arbeit

Die unüberschaubare Vielfalt und Menge der verschiedenen Lösungen und Erweiterungen macht die Auswahl einer geeigneten Methode zur Lösung einer gestellten Steuerungsaufgabe sehr schwierig und zeitaufwändig. Ein Ziel dieser Arbeit war es, die verschiedenen PC-basierten Lösungsmöglichkeiten aufzuzeigen, in wenige Klassen von Systemarchitekturen zu gruppieren und diese hinsichtlich der Einhaltung von Realzeitkriterien zu analysieren. Weiterhin waren die spezifischen Grenzen der software- und hardwarebasierten Architekturen zu ermitteln. Unter Einbeziehung der gestiegenen Ansprüche an Steuerungssysteme für zeitkritische technische Prozesse und des Fortschritts der PC-Technologie, wurde mit der vorliegenden Arbeit eine *Entscheidungsgrundlage* in Form einer *Klassifizierung* der Lösungsmethoden in nur *fünf grundlegende Systemarchitekturen* geschaffen. Zwei weitere Kombinationen vervollständigen die Gruppierung. Jede erhältliche PC-basierte Systemarchitektur lässt sich so in eine der 5 + 2 Klassen einteilen. Die Realzeitfähigkeiten der Klassen wurden verifiziert, die praktische Relevanz zeigen zwei Anwendungsbeispiele. Erweiterte Architekturkonzepte zeigen neue Lösungsansätze auf. In dem geschaffenen Klassifizierungs-Diagramm kann nun, nach Analyse der Realzeitanforderungen einer vorgegebenen Steuerungsaufgabe, d.h. nach Ermittlung der minimalen Reaktionszeit und maximalen Komplexität, sehr schnell eine geeignete Systemarchitektur gefunden werden. Alle Methoden, deren minimale Reaktionszeiten über den ermittelten Reaktionszeiten liegen, scheiden von vorneherein aus. Eine zeit- und kostenaufwändige *Try-and-Error*-Phase wird vermieden. Die Klassifizierungsergebnisse bilden somit eine wichtige Grundlage für den automatisierten Systementwurf.

2 Grundlagen

2.1 Zeitkritische technische Prozesse

In vielen Anlagen und Maschinen treten Prozesse auf. Ein Prozess besteht aus technischen, physikalischen oder chemischen Abläufen, die festgelegten Gesetzen folgen oder nach solchen gesteuert werden müssen [Fär94]. Kontinuierliche Umwandlungsprozesse der Verfahrenstechnik und die Verteilung von Energie in elektrischen Netzen gehören genauso dazu, wie der Fertigungsprozess in einer Motorenfabrik, der Straßenverkehr oder die Materialbewegung in einem Lagerhaus. Im erweiterten Sinne kann auch die Sammlung, Umformung und Verteilung von Informationen als Prozess gesehen werden. Im Hinblick auf die Struktur der Steuerungssysteme ist eine Einteilung aller Prozesse in vier Klassen nützlich [Heu84]:

- **Kontinuierliche Prozesse**, z.B. Chemische und verfahrenstechnische Prozesse, bei denen eine kontinuierliche Umwandlung von Materie stattfindet (Verbrennung in einer Wärmekraftmaschine).
- **Stückprozesse**, z.B. die Bewegung oder Bearbeitung diskreter, einzeln identifizierbarer Objekte (Materialfluss in einem Walzwerk).
- **Befehlsprozesse**, z.B. Anfahr- und Schaltprozesse, die durch eine festgelegte Folge von Steueranweisungen von einem Zustand in den anderen gebracht werden können (numerische Steuerung einer Werkzeugmaschine).
- **Informationsprozesse**, z.B. der Austausch von Nachrichten unterschiedlicher Länge und Dringlichkeit zwischen einer Vielzahl von Teilnehmern mit Hilfe eines Vermittlungssystems (Telefonanlage).

Technische Prozesse können aus sehr vielen Untersystemen mit unterschiedlich engen technischen und logischen Kopplungen zusammengesetzt sein. Jedes Untersystem wird durch Vorgabe von Sollwerten gesteuert und enthält Mechanismen, um automatisch oder manuell die vorgegebenen Arbeitpunkte einzustellen. Steuerungssysteme können technischer oder organisatorischer Natur sein; für eine Prozesssteuerung sind beide von gleichem Interesse. Aufgabe jeder Prozesssteuerung ist es, mit dem gesteuerten System bestimmte, meist wirtschaftlich optimale Ergebnisse zu erzielen. Diese Zielsetzung gilt auch für das Untersystem und damit auch für die gesamte Steuerungshierarchie. Üblicherweise sind auf bestimmten Ebenen eines Prozesses Menschen als Steuernde oder zu Steuernde vorhanden. In Einschränkung sprechen wir oft nur bei automatischen Systemen von *Prozesssteuerungen*.

Die technischen Prozesse sind von den zugehörigen Anwendungsprozessen (Tasks) einer Prozesssteuerung zu unterscheiden. Sind die Bearbeitungsprozesse aufgrund der physikalischen Randbedingungen des technischen Prozesses an zeitliche Einschränkungen gebunden, spricht man von *Realzeitprozessen*. Realzeitprozesse lassen sich nach ihren charakteristischen Eigenschaften in verschiedene Gruppen unterteilen. Wird z.B. die Art ihres Auftretens gewählt, kann man periodische und aperiodische Prozesse unterscheiden. Eine andere Aufteilung ergibt sich aufgrund der Lebens-

dauer der Realzeitprozesse. Einige sind statisch, d.h. ein permanenter Teil des Realzeitsystems, während andere dynamisch zur Laufzeit erzeugt werden und nur während der Bearbeitung vorhanden sind. In [RSZ89] wird eine Unterteilung der Bearbeitungsprozesse in *(zeit)kritische* (*critical*), *notwendige* (*essential*) und *unwesentliche* (*nonessential*) *Prozesse* vorgeschlagen. Zeitkritische Prozesse müssen ihre Zeitschranke (*deadline*) einhalten, sonst wären die Folgen „katastrophal". *Deadlines* von notwendige Prozesse sollten eingehalten werden, da sie für die Funktion des Realzeitsystems wichtig sind. Sporadische Überschreitungen der Zeitbedingungen können jedoch meistens toleriert werden, ohne fatale Auswirkungen nach sich zu ziehen. Die unwesentlichen Prozesse, z.B. Systemverwaltungsaufgaben, haben niedrigste Priorität und werden nur abgearbeitet, wenn keine kritischen oder notwendigen Prozesse anstehen. Hier ergeben sich unerwünschte Folgen nur bei sehr großen Verzögerungen.

Eine restriktive Sichtweise ergibt sich durch ausschließliche Einteilung der Prozesse nach den Kosten, welche bei einer angenommenen Überschreitung der Zeitbedingung entstehen würden. Bedeutet die Nichteinhaltung der Zeitbedingung eine Verfehlung der Funktion, so spricht man von *harter Realzeit* [Süs98]. Beispiele hierfür sind die Ampelschaltung oder die Regelung eines Flugzeugs. Bei der kleinsten Überschreitung der Zeitgrenze muss mit sehr hohen Kosten gerechnet werden. Führt die Nichteinhaltung der Zeitbedingung hingegen „nur" zu einer Verschlechterung der Funktion, so nennt man dies *weiche Realzeit*. Sporadische Überschreitungen führen zu erhöhten, aber nicht außerordentlichen Kosten. Beispiele hierfür sind die Telefonvermittlung oder die Übertragung von Bildmaterial.

2.2 Klassische Realzeitsysteme

2.2.1 Eigenschaften und Beispiele

Systeme zur Bearbeitung von Realzeitaufgaben gibt es schon seit mehreren Jahrzehnten. Zur Prozesssteuerung wurden in den 60er und 70er Jahren programmierbare Rechensysteme, sogenannte *Prozessrechner* eingesetzt. Die Weiterentwicklung elektronischer Bauelemente und die Möglichkeit, mehrere Rechensysteme über standardisierte Übertragungseinrichtungen miteinander zu verbinden, ermöglichte in den 80er Jahren den Entwurf integrierter Steuerungssysteme, den *speicherprogrammierbaren Steuerungen*. In einer Vielzahl von Produkten der Automobilindustrie, der Verkehrstechnik, der Produktions- und Fertigungstechnik, sowie der Telekommunikationsindustrie findet man heute integrierte mikroelektronische Steuerungen, sogenannte *eingebettete Systeme*. Sie bestehen in der Regel aus für die jeweilige Aufgabe optimierter Hardware (Mikrochips) und darauf lauffähiger Software.

Alle Realzeitsysteme müssen charakteristische Anforderungen erfüllen, um Realzeitprozesse verarbeiten zu können [Law92][Fär94]:

- **Rechtzeitigkeit**: Implizit im Namen *Realzeitsysteme* ist die Forderung nach einer rechtzeitigen Verarbeitung von Daten gegeben. Unter allen Umständen muss die *deadline* eingehalten werden. Ansonsten wird das Ergebnis nutzlos: *„Late data is bad data"*.
- **Gleichzeitigkeit**: Mehrere technische Prozesse erfordern eine quasi-parallele Bedienung durch die Bearbeitungsprozesse. Vorgänge müssen oftmals synchronisiert werden.

- **Vorhersagbarkeit**: Die Verarbeitungszeit im ungünstigsten Fall (*worst case*) als Reaktion auf ein Ereignis muss vorhersagbar sein. Für viele Prozesse gilt die Forderung nach einem *deterministischen*[2] System.
- **Korrektheit**: Die Erfüllung der Aufgabe gemäß der Spezifikation, die Einhaltung aller Zeitbedingungen und die Vermeidung von Verklemmungen (*deadlocks*) ist zu gewährleisten.
- **Robustheit**: Im Fehlerfall muss, sofern vorhanden, in einen *fail-safe*-Zustand übergegangen werden.
- **Kommunikation**: Schnittstellen und Busse ermöglichen den Datenaustausch mit anderen Systemen, Sensoren und Aktoren.

Typische Beispiele für Realzeitsysteme sind:

Einspritz- und Zündungssteuerung eines Verbrennungsmotors: Zahlreiche Signale, wie Kurbelwellenwinkel, Benzindruck, Lufttemperatur, Luftmenge, Drosselklappenwinkel, λ-Wert usw. dienen als Eingangsgrößen des Steuerungssystems. Ein elektronisches Steuergerät, bestehend aus einem leistungsstarken Mikrocontroller und Signalkonditionierungsschaltungen, berechnet optimale Werte für Einspritzzeitpunkt und –menge sowie Zündzeitpunkt aus den Sensorsignalen und gespeicherten Kennfeldern des Motors. Die Ausgangssignale werden an die Aktoren wie Einspritzdüsen, Zündspule, Magnetventile usw. weitergeleitet. Aufgrund der schnellen Kolbenbewegung müssen alle Berechnungen ihre Zeitbedingungen genau eingehalten werden, um einen emissionsarmen und treibstoffsparenden Betrieb zu gewährleisten. Bild 2.1 zeigt schematisch die hochintegrierte Motorsteuerung *Motronic ME7* der Firma Bosch.

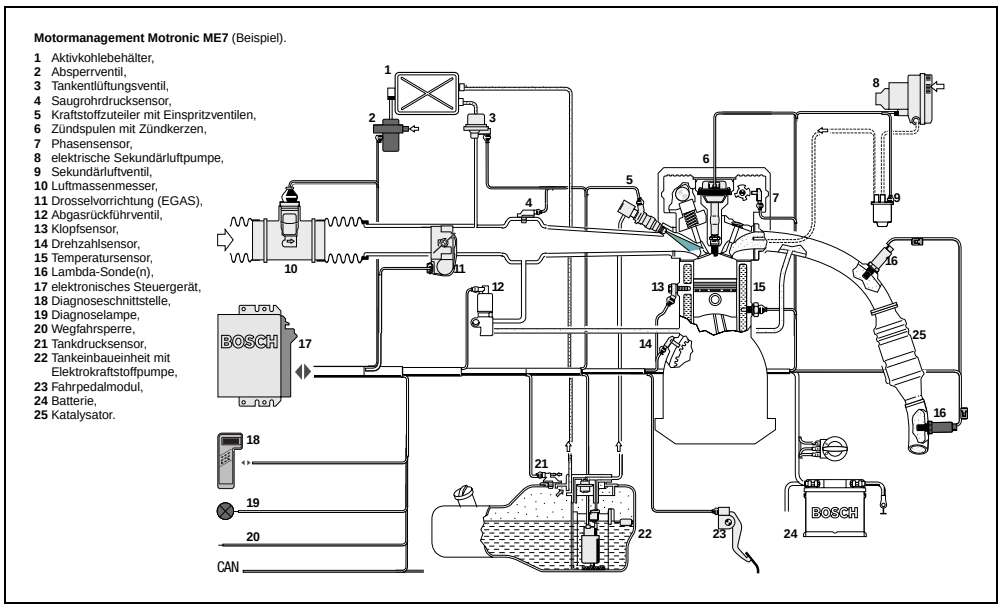


Bild 2.1 Elektronisches Motormanagement *Motronic ME7* (Bosch)

Regelung eines Kernkraftwerks: Die physikalischen Eigenschaften der Kernspaltung erfordern eine exakte Regelung der Betriebsparameter. Die während der Kernspaltung von Uran 235 entstehenden schnellen Neutronen werden durch einen Moderator (z.B. Wasser) auf Spaltgeschwindigkeit abgebremst und treffen als langsame Neutronen auf weitere Uran-Kerne. Die Regelung der Aktivi-

[2] *Deterministisch* heißt ein System, wenn seine Reaktion auf eine bestimmte Eingabe, ausgehend von einem bestimmten Zustand, eindeutig festgelegt ist. Wenn dies nicht der Fall ist, d.h. wenn sich Reaktionen des Modells nur durch Wahrscheinlichkeitsverteilungen beschreiben lassen, nennt man es *stochastisch* (oder *probabilistisch*).

tät der Kettenreaktion (Wachstumsfaktor) erfolgt durch Steuerstäbe aus B_4C oder Cadmium. Die Steuerstäbe haben die Aufgabe, die eventuell für weitere Spaltungen zur Verfügung stehenden Neutronen zu neutralisieren. Der Wachstumsfaktor in Kernkraftwerken beträgt maximal 1,0075 (0,75 % Zuwachs), um genügend Zeit zur Einregelung zu haben, zumal die bei der Kernspaltung frei werdenden Neutronen mit bis zu 20 Sekunden Verzögerung auftreten und eine Kernspaltung selbst nur ca. zehn Nanosekunden dauert.

Mikroperforation von Spiralblockseiten: Bei der Produktion von Schreibblöcken werden die einzelnen Seiten mit einer Perforation versehen, um sie einzeln heraustrennen zu können. Normalerweise wird eine Messerwalze verwendet, die mit dem Antrieb der Papierbahn fest verbunden ist. Zur Realisierung einer feineren Mikroperforation werden winzige Löcher mit kurzen Laserpulsen in das Papier gebrannt. Die Führung des Laserstrahls und die Energie der Laserpulse müssen absolut synchron zur Geschwindigkeit der Papierbahn geregelt werden, damit lineare Perforationsverläufe entstehen.

Ein Realzeitsystem in der Form eines eingebetteten Systems besteht aus verschiedenen Komponenten. Viele davon, wie z.B. Interrupt-Controller, Zähler, Zeitgeber, *Watchdog*, Takterzeugung, A/D- und D/A-Wandler, parallele und serielle Schnittstellen, internes Flash-ROM, ROM und RAM, sind im Mikrocontroller integriert. Zusätzlicher Speicher und Peripheriebausteine können über eine externen Busschnittstelle angebunden werden. Die Struktur zeigt Bild 2.2.

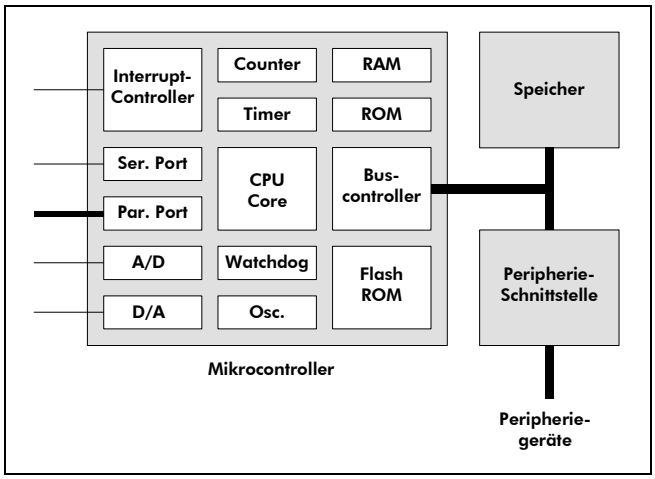


Bild 2.2 Struktur eines eingebetteten Realzeitsystems

2.2.2 Realzeitbetriebssysteme und Anwendungen

Die auf einem Realzeitsystem zu implementierende Steuerungsanwendung wird überwiegend in Assembler oder in einer ressourcensparenden Programmiersprache wie z.B. „C" realisiert. Im Gegensatz zu einfachen sequentiellen Programmen können mit Realzeit-Kerneln oder Realzeitbetriebssystemen (RTOS) komplexe Prozesse gesteuert werden. Die Erstellung von Realzeit-Multitasking-Software erfordert eine andere Denkweise, spart aber bei der Entwicklung unnötige Programmierarbeit. Um die knappen Ressourcen zu schonen, sind moderne RTOS modular skalierbar gehalten, d.h. nur die zur Funktion des Systems benötigten Module werden zur Anwendung hinzugebunden (gelinkt). Außerdem bieten Realzeitbetriebssysteme die notwendigen Protokolle für die immer häufiger gewünschte Vernetzung mehrerer Systeme. Gerade für eine wirtschaftliche Administration der immer aufwändigeren Funktionen bietet die Vernetzung neue Möglichkeiten zur Fernadministration,

-wartung und -überwachung bei gleichzeitiger Kostenreduktion. In aller Munde ist das „Web-basierte Remote-Management“ über einen integrierten HTTP-Server. Hierfür sind leistungsfähige Mikrocontroller mit ausreichendem Speicher vorzusehen.

Moderne Realzeitbetriebssysteme sind hierarchisch strukturiert, modular aufgebaut und taskorientiert implementiert. Hierarchie und Modularität reduzieren die Komplexität des Prüf- bzw. Verifikationsaufwands auf einen linearen Umfang. Das Betriebssystem wird heute aus Bausteinen nach den spezifizierten Anforderungen konfiguriert. Man kann die Struktur als Schichtenmodell beschreiben (Bild 2.3). Jede Schicht repräsentiert hierbei eine abstrakte (virtuelle) Maschine, die spezifische Funktionen bereitstellt. Die Betriebssystemkern-Schicht stellt z.B. sämtliche Funktionen der Taskverwaltung, wie Unterbrechungsverarbeitung, Taskdefinition, -scheduling und -synchronisation zur Verfügung. Ihre zweite Hauptaufgabe ist üblicherweise die Realisierung von Gerätetreibern. Die restlichen Schichten bauen auf der Betriebssystemkern-Schicht auf. Gegenwärtig versucht man, möglichst viele Funktionen in die oberen Schichten zu verlagern um die Abhängigkeiten von der Zielhardware zu minimieren. Damit erhöht sich die Portabilität und die Zuverlässigkeit des Realzeitbetriebssystems.

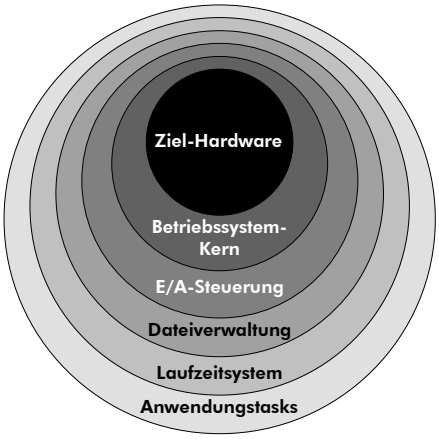


Bild 2.3 Schichtenmodell eines Realzeitbetriebssystems

Wichtige Funktionen eines Realzeitbetriebssystems sind die *Betriebsmittelverwaltung* und die *Taskverwaltung*. Bei der Erstellung der Realzeit-Anwendung teilt man zeitintensive Aufgaben in elementare Teilaufgaben (Tasks) auf. Dadurch können sie weitgehend unabhängig voneinander parallel ausgeführt werden. Voraussetzung hierfür ist das *Multitasking*, die Fähigkeit eines Systems, mehrere Aufgaben (Tasks) gleichzeitig (parallel) zu bearbeiten. Ein Realzeitsystem mit einem Prozessor oder Mikrocontroller kann immer nur eine Task pro Zeiteinheit bearbeiten. Durch schnelles Hin- und Herschalten zwischen den Tasks entsteht der Eindruck einer scheinbar parallelen Verarbeitung; man spricht in diesem Fall von *quasiparalleler* Verarbeitung. Echte Parallelverarbeitung ist nur auf einem Multiprozessorsystem möglich. Das Hin- und Herschalten zwischen den Tasks übernimmt der sogenannte *Dispatcher*, die Entscheidung welche Task als Nächste rechnen darf, der *Scheduler*. *Dispatcher* und *Scheduler* sind Bestandteile jedes Realzeitbetriebssystems. Die Schedulingalgorithmen lassen sich in *statische* und *dynamische* Verfahren unterteilen. Zu den in Realzeitsystemen überwiegend eingesetzten dynamischen Verfahren zählen das prioritätsbasierte *Rate monotonic*-Scheduling und das leistungsfähige *Earliest-deadline-first*-Scheduling. Im Gegensatz zu den statischen Verfahren wird hier erst zur Laufzeit entschieden, welche Task als Nächste vom Prozessor zu bearbeiten ist.

Betriebsmittel sind alle Mittel (Hardware und Software), die eine Task braucht, um durchgeführt werden zu können. Die Betriebsmittelverwaltung stellt somit die Teilfunktionen zum konfliktfreien Belegen, Betreiben und Freigeben von Betriebsmitteln für die einzelnen Tasks zur Verfügung. Neben der Task- und Betriebsmittelverwaltung gehören zwei weitere Maßnahmen zu den wichtigen Bestandteilen:

- *Zuverlässigkeitsmaßnahmen*: Beheben von Speicher- und Übertragungsfehlern, Unempfindlichkeit gegen fehlerhafte Benutzung, Rekonfiguration fehlertoleranter Systeme
- *Schutzmaßnahmen*: Speicherschutzregister, Speicherschutzschlüssel, Schutzbereiche, Zugriffskontrolle, Passwörter

Die Basis aller Zuverlässigkeitsmaßnahmen ist die *Redundanz* oder *Robustheit* (Unempfindlichkeit, z.B. *exception handling* und *graceful degradation*). Zu den Schutzmaßnahmen zählen alle Strategien zum Schutz der Betriebsmittel vor unberechtigten Zugriffen der Tasks. Bei vernetzten Systemen zählt aber auch der Schutz vor ungewollten Eindringlingen und Viren dazu. Die Basis der Schutzmaßnahmen sind das *Berechtigungskonzept* und die *Schutzbereiche*. So kann man sich das Betriebssystem aus verschiedenen Hierarchieebenen vorstellen, bei denen die Rechte zu den äußeren (bzw. oberen) Schichten hin immer mehr abnehmen.

Eine wesentliche strukturelle Schutzmaßnahme ist die Verhinderung von *Verklemmungen* von Prozessen. Die folgenden vier Bedingungen sind notwendig und hinreichend für das Auftreten von Verklemmungen. Indem man sicherstellt, dass mindestens eine der folgenden Bedingungen nicht erfüllt ist, lassen sich Verklemmungen erfolgreich verhindern:

- Betriebsmittel sind nur exklusiv nutzbar.
- Betriebsmittel können nicht entzogen werden.
- Prozesse belegen zugewiesene Betriebsmittel auch dann, wenn sie auf Zuweisung weiterer Betriebsmittel warten.
- Es gibt eine zyklische Kette von Prozessen, von denen jeder ein Betriebsmittel besitzt, das der nächste Prozess in der Kette benötigt.

Entwurfsprozess

Der Entwurfsprozess für eingebettete Realzeitsysteme (Bild 2.4) wird in zahlreichen Forschungsprojekten untersucht. Ziel ist meist die Entwicklung einer Methodik zum durchgängigen Entwurf von eingebetteten Systemen, speziell die automatische Generierung von Software aus der Systemspezifikation und die Synthese einer Schnittstellen- und Hardwarebeschreibung (Hardware/Software Co-Design). Für eine schnelle und kostengünstige Entwicklung komplexer Systeme ist die effiziente Wiederverwendung von Schaltungsbeschreibungen eine wichtige Grundlage. Durch die Verwendung von abstrakten Beschreibungssprachen, beispielsweise VHDL[3], wird eine modulare Systemspezifikation ermöglicht. Jedes Modul dieser Spezifikation besteht aus einer Menge von Elementen, die durch Komponenten, Funktionen und Prozeduren gebildet werden. Das bietet die Möglichkeit, Module separat zu beschreiben und damit anderen Entwürfen wieder zur Verfügung zu stellen (*reuse*).

[3] VHDL: ***V****hsic* ***H****ardware* ***D****escription* ***L****anguage*, vhsic = very high speed integrated circuit.

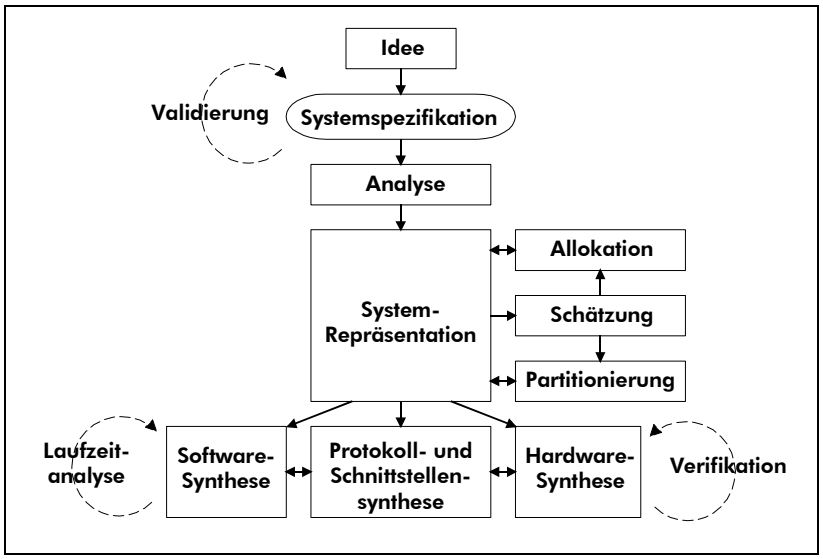


Bild 2.4 Entwurfsprozess für eingebettete Realzeitsysteme

2.3 PC-basierte Realzeitsysteme

PC-basierte Systeme verwenden, im Unterschied zu den proprietären klassischen Realzeitsystemen, die Standard-PC-Hardware als Rechnerplattform. Der Prozessor (x86-CPU) fungiert hierbei als zentrale Recheneinheit. Prozesssignale können entweder über die vorhandenen Schnittstellen oder über separate Module in Form von Steckkarten eingelesen oder ausgegeben werden. Für die Bussysteme des PCs ist eine Vielzahl von Peripheriemodulen erhältlich. Eingangsmodule erfassen physikalische Eingangsgrößen und wandeln sie in digitale Signale um. Ausgangsmodule generieren analoge und digitale Signale zur Ansteuerung von Aktoren. Über Feldbusmodule können intelligente Sensoren und Aktoren oder andere Realzeitsysteme eingebunden werden.

Als Vorteile der PC-basierten Steuerungssysteme gegenüber klassischen Systemen gelten:

- Prozess-Informationen sind inhärent verfügbar und erleichtern die Einbindung der Prozesssteuerung in ERP-Systeme[4] bzw. die Integration von Büro und Produktion.
- Die grafische Visualisierung von Prozesssignalen und –abläufen schafft eine anwenderfreundliche Mensch-Maschine-Schnittstelle und erleichtert so die Bedienung.
- Die Verwendung von standardisierter Hard- und Software ermöglicht eine schnellere Projektierung.
- Der technologische Vorsprung der PC-Technik und die rasante Weiterentwicklung schafft eine leistungsfähige Plattform für die Prozesssteuerung, gerade für komplexe Steuerungsaufgaben.
- Offene Standards machen unabhängig von Herstellern und führen zu modularen Systemen.
- Vielfältige Kommunikationsmethoden ermöglichen dezentrale Lösungen. Besonders nützlich sind die bereits vorhandenen Schnittstellen für Intranet/Internet-Anschluss.
- Zur Programmierung werden meist Ablaufdiagramme eingesetzt, die sich mit geringem Aufwand aus der Spezifikation generieren lassen.

[4] ERP: ***E****nterprise* ***R****esource* ***P****lanning*.

2.3.1 Struktur

Die Systemstruktur des PCs sowie Aufbau und Funktion der eingesetzten Hardwarekomponenten werden durch eine Vielzahl von Industriestandards bestimmt. Wichtigste Grundlage bildet der von den Firmen Intel und Microsoft herausgegebene *PC Design Guide*. Die aktuelle Ausgabe (PC2001, Version 1.0, [IM00]) gibt der Hard- und Softwareindustrie Richtlinien und Vorschläge zur Gestaltung eines PC-Systems, speziell für den Einsatz mit Windows-Betriebssystemen. Innerhalb des *Design Guides* wird auf die entsprechenden Standards und Industrienormen verwiesen. Die Hardwarekomponenten und -struktur eines PC2001-Systems zeigt Bild 2.5.

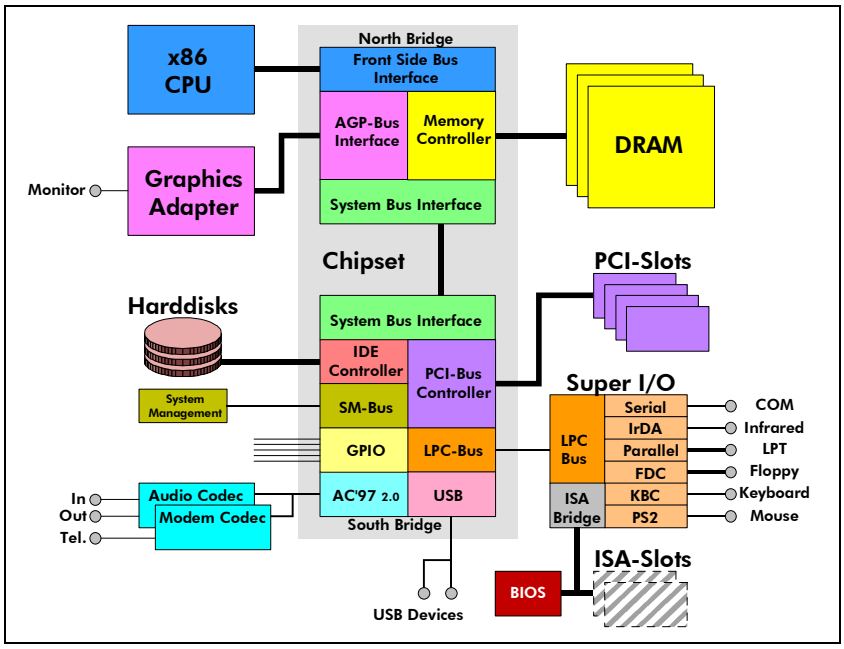


Bild 2.5 Hardwarekomponenten eines Standard-PCs

Der Chipsatz ist das Bindeglied zwischen den einzelnen Komponenten eines Computersystems. Er steuert das Zusammenspiel und den Datenfluss zwischen Prozessor, Arbeitsspeicher, Bussystemen sowie den Schnittstellen und hat großen Einfluss auf die Leistung des Gesamtsystems. Die Chipsätze der verschiedenen Hersteller haben, bei ansonsten identischen Hardwarekomponenten, Leistungsunterschiede von bis zu 15 %. Mit der Auswahl des Chipsatzes werden Typ des Systembusses, des Speichers, der Schnittstellen und des Prozessors festgelegt. Fehlende Komponenten und Schnittstellen werden über Zusatzbausteine, wie z.B. einen *Super-I/O-Chip*, hinzugefügt.

Der Chipsatz ist meist in zwei separate Bausteine unterteilt, die über einen schnellen Systembus verbunden sind. Aus der Historie spricht man hier von *North* und *South Bridge*. Die *North Bridge* enthält Schnittstellen zur Anbindung des Prozessors (*Front Side Bus*), der Grafikkarte (*Advanced Graphics Port*, AGP) und des Systembusses (*System Bus Interface*), sowie eine Steuereinheit für den Arbeitsspeicher (*memory controller*). Die *South Bridge* beinhaltet Controller für die Festplatten (IDE), den PCI-Bus und den USB-Bus. Weiterhin befinden sich dort die Schnittstellen für das Systemmanagement (SM-Bus: Lüfter, Temperatursensoren usw.), Audio- und Modem[5]-Wandlerbausteine (AC97 *Codec*[6]), einige frei verwendbare Anschlusspins (*General Purpose I/O, GPIO*) und ein universeller

[5] Modem: **Mo**dulator und **Dem**odulator.
[6] Codec: ***Co****ding and* ***Dec****oding.*

Low-Pin-Count-Bus zum Anschluss von hochintegrierten Schnittstellenbausteinen. Über diesen können klassische Peripheriegeräte, wie Maus, Tastatur, Floppy, serielle und parallele Schnittstellen usw. an das PC-System angeschlossen werden. In neuen Systemen versucht man auf die Vielfalt dieser nicht immer unproblematischen Schnittstellen zu verzichten. So untersagt z.B. der PC2001-Vorschlag ausdrücklich die Verwendung von ISA-Bus, Floppycontroller, serieller oder paralleler Ports, sowie der PS2-Schnittstelle[7]. Unverzichtbare Geräte, wie Maus oder Tastatur, sind über den USB-Bus anzuschließen.

Für den Einsatz der Standard-PC-Hardware als Realzeitsystem zur Steuerung von technischen Prozessen werden Ein- und Ausgänge für analoge und digitale Signale benötigt. Vorrangig bieten sich die integrierten Schnittstellen an. Acht digitale TTL-Signale können ohne großen Aufwand über die parallele Schnittstelle eingelesen oder ausgegeben werden. Im industriellen Umfeld sind jedoch diese empfindlichen Schnittstellen aufgrund der fehlenden Schutzeinrichtungen nur begrenzt einsetzbar. Störspannungen und –impulse lassen die PC-Hardware einen „schnellen und leisen Tod sterben". Die hohe Integrationsdichte auf dem *Motherboard* verhindert eine kostengünstige Reparatur. Für einen zuverlässigen Betrieb sind also unbedingt Peripheriegeräte und Steckkarten mit galvanischer Trennung und ausreichenden Isolationsabständen einzusetzen; vorzugsweise als externe Komponenten mit eigener Stromversorgung.

2.3.2 Realzeitfähigkeit

In vielen Forschungsprojekten wird die Realzeitfähigkeit von PC-Systemen und der darauf ablaufenden Standardbetriebssysteme für die Realisierung von Steuerungen für zeitkritische technische Prozesse untersucht. Die zeitlichen Anforderungen ergeben sich hierbei aus den charakteristischen Eigenschaften, der in einem technischen Prozess auftretenden Ereignisse. Wichtig sind *Auftrittsverteilung*, d.h. wann und in welchen Abständen treten die Ereignisse auf, sowie die maximal zulässigen *Reaktionszeiten*, d.h. wie lange darf es maximal dauern bis eine Antwort auf ein bestimmtes Ereignis generiert wird. Die Auftrittswahrscheinlichkeit reicht von einfachen konstanten und zyklischen Ereignissen bis hin zu komplexen, rein zufällig auftretenden Ereignissen ohne bekannte Statistik. In einigen Fällen ist das Auftreten mehrerer unterschiedlicher Ereignisse voneinander abhängig.

In der Literatur werden PC-basierte Systeme oft den weichen Realzeitsystemen zugeordnet. Diese Aussage gilt jedoch nicht pauschal, denn die Einhaltung von Realzeit-Bedingungen hängt stark von der eingesetzten Soft- und Hardware sowie den absoluten Zeitgrößen ab. „Realzeit" bzw. „zeitkritisch" ist *relativ*. So bedeutet zeitkritisch bei der Messwerterfassung von Atomzusammenstößen in Beschleunigeranlagen einige Nanosekunden. Bei der Steuerung von Antrieben, Robotern oder in Feldbussystemen wird Realzeit oft in Mikrosekunden gemessen, bei verteilten Prozesssteuerungen reichen Millisekunden. Sind hingegen thermische Prozesse zu steuern genügen oft Sekunden für die Einhaltung der zeitlichen Anforderungen.

Geringe zeitliche Anforderungen

Man ist bestrebt, den Aufwand minimal zu halten, d.h. bei den hier eingesetzten PC-basierten Systemen weitestgehend Standardkomponenten einzusetzen. Aus Kostengründen ist es außerdem sinn-

[7] Hingegen gilt die Bereitstellung dieser Schnittstellen über eine PCI-Steckkarte als PC2001-konform.

voll, zuerst alle Möglichkeiten der Software auszuschöpfen. Man versucht hier, die Standardbetriebssysteme Linux und Windows NT zu verwenden. Wie im Kapitel 4 ausführlich beschrieben wird, erreicht man mit diesen Betriebssystemen (ohne Veränderungen) nur Reaktionszeiten im Bereich von einigen zehn Millisekunden. So untersuchte [WGT97] mit Hilfe des *Hartstone Benchmark* [DSW92] das Realzeitverhalten von Windows NT. Es ließ sich feststellen, dass Windows NT nur bedingt für die Erfüllung von Realzeitaufgaben geeignet ist. Lediglich für den Prozess mit der höchsten Priorität kann die Einhaltung von Realzeitbedingungen garantiert werden. Eine Alternative stellt die Überdimensionierung des PC-Systems in Bezug auf die Rechenleistung für Anwendungen mit Zeitschranken von über 50 ms dar. Bei einer Systemlast von unter 20 % werden Zeitbedingungen auch unter Windows NT relativ genau eingehalten.

Die Reduzierung der großen Variabilität (*Jitter*) von bis zu mehreren hundert Millisekunden ist Gegenstand zahlreicher Forschungstätigkeiten. So implementiert [LCN98] einen *Soft Real-time Scheduling Server* für Windows NT, mit dessen Hilfe der Jitter beim Abspielen von Multimediadaten von 300 ms auf unter 100 ms reduziert wird. Hierzu wird zusätzlich zum Scheduler des NT-Kernels ein weiterer *User level*-Scheduler mit höchster Priorität ausgeführt. Dieser sorgt für eine faire Rechenzeitzuteilung zwischen zeitkritischen und normalen Prozessen. Zur effizienteren Programmierung von parallelen Anwendungen auf Multiprozessorsystemen entwickelt [HBP+00] einen *Kernel-mode Resource Manager* für das Betriebssystem Windows 2000 auf Grundlage des *Nanothreads Programming Model*. Die Aufgabe des *Resource Managers* ist, alle laufenden Anwenderprozesse zu überwachen und sie anhand einer optimalen Schedulingstrategie auf verschiedene Prozessoren aufzuteilen. Damit lassen sich die Ausführungszeiten gegenüber dem Windows-eigenen Schedulingmechanismus um den Faktor 2 verringern. Die von [SP95][SP96] vorgeschlagene Erweiterung *Dreams Subsystem*[8] (Bild 2.6) benützt mehrere Managementeinheiten zur Reservierung von Rechenzeit und ihrer Kontrolle.

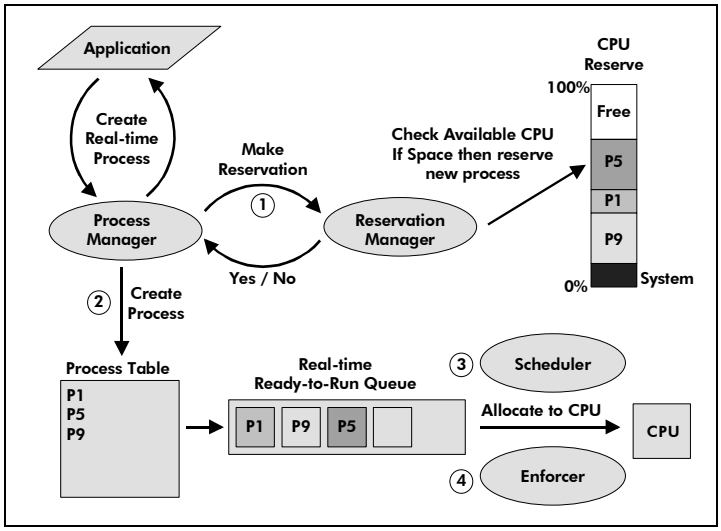


Bild 2.6 Struktur des *Dreams Subsystem*

Die Methode der Rechenzeitbeschränkung und der Prozessor-Reservierung wird auch von anderen Forschungsteams untersucht. Eine Entwicklergruppe der Firma Microsoft [JBF+96] hat mit der *Rialto Real-time Architecture* ein entsprechendes System entwickelt. Mit der Hilfe von *time constraints* kann ein Prozess seine Zeitbedingungen spezifizieren, die er zur korrekten Ausführung benötigt. Hierzu

[8] Ein Subsystem ist die Windows-Version eines Unix User Mode Servers.

stehen die Aufrufe `BeginConstraint()` und `EndConstraint()` zur Verfügung. Der Realzeit-Scheduler versucht diese Zeitbedingungen durch eine zweckmäßige Ablaufreihenfolge einzuhalten. Auch hier steht ein *Resource Manager* zur Verwaltung und Überprüfung der Anforderungen bereit. Die *Rialto*-Architektur greift sehr tief in das Windows-Betriebssystem ein und ersetzt Teile von diesem, sodass bestehende Anwendungen umgeschrieben werden müssen. Weitere Informationen finden sich in [JRR97] und [JR99].

Will man Windows NT für weiche Realzeitaufgaben verwenden, ohne Erweiterungen einzusetzen, so sollten die Ausführungszeiten der Prozesse vermessen werden. Hier bietet sich das von [GN99] entwickelte *Performance Monitoring Tool JewelNT* an, welches eine detaillierte grafische Anzeige und Analyse der Taskausführungszeiten, Task-Umschaltungen und Unterbrechungen erlaubt. In Entwicklungssystemen von Realzeitbetriebssystemen sind ähnliche Monitoring-Werkzeuge enthalten, wie z.B. das in die *Tornado*-Entwicklungsumgebung der Firma Wind River Systems integrierte *WindView* für das Realzeitbetriebssystem *VxWorks*.

Auch das Standardbetriebssystem Linux ist Gegenstand der Forschung. In [HR01] wird *Rapid Reaction Linux* beschrieben, das aus der Vereinigung zweier Linux-Realzeit-Patches entstanden ist. Die Basis bilden der *Low-Latency-Patch* von Molnar, der lange Latenzzeiten im Kernel verringert, und der *UTIME-Patch* der Universität von Kansas, der den PC-Timerchip auf die jeweilige Zeitspanne bis zum nächsten Ereignis umprogrammiert. Auf diese Weise können die Latenzzeiten des Linux-Kernels um den Faktor 10 auf ca. 5 – 10 ms reduziert werden.

Zur Anwendung kommen diese nicht oder nur leicht veränderten Standardbetriebssysteme bei weichen Realzeitanforderungen, wie sie z.B. bei „gutmütigen" technischen Prozessen auftreten. In [HB00] wird die Regelung eines schwebenden Körpers nur mit einem Standard-PC, einer A/D-D/A-Steckkarte, Windows NT und der Multimedia-Bibliothek `WinMM.lib` realisiert. Sporadische Ausreißer der Abtastrate haben aufgrund der Trägheit der Masse und der Induktivität keine gravierenden Auswirkungen auf die Stabilität.

Anforderungen im Sub-Millisekundenbereich

Ergeben sich bei der Analyse eines technischen Prozesses harte Realzeitanforderungen im Sub-Millisekundenbereich, und will man weiterhin Standardbetriebssysteme verwenden, muss man größere Veränderungen vornehmen. Aufgrund des großen Interesses der Industrie für PC-basierte Prozesssteuerungen mit Reaktionszeiten im Bereich von 50 – 500 Mikrosekunden entwickelten verschiedene Firmen Lösungen. Gerade das angestammte Gebiet der klassischen SPS-Systeme bietet einen großen Markt mit hohen Ertragschancen.

Müssen Reaktionszeiten im 5 – 50 Mikrosekunden-Bereich erzielt werden, verwendet die Mehrzahl der Anwender klassische Realzeitsysteme, z.B. in Form eines eingebetteten Systems zusammen mit einem Realzeitbetriebssystem. Aber auch hier lässt sich der PC weiterhin einsetzen, allerdings ebenfalls unter Verwendung eines Realzeitbetriebssystems. Untersuchungen von [Hel96] und [Pop98] ergaben minimale Reaktionszeiten von unter 10 μs mit dem RTOS VxWorks. Bei der Analyse der Reaktionszeiten in belasteten PC-Systemen werden immer wieder Ausreißer gemessen. Die Ursachen sind vielschichtig (siehe Kapitel 5) und nicht vollständig geklärt; die Störungen lassen sich aber durch eine umsichtige Auswahl der PC-Komponenten und entsprechend angepasste Software durchaus vermeiden. Auch Mehrprozessorsysteme mit gemischten Betriebssystemen sind möglich und werden in der Forschung [Sto00] untersucht.

Weitere Steigerungen der Reaktionsgeschwindigkeiten sind nur durch Erweiterungen der PC-Hardware zu erreichen. Mit auf PC-Schnittstellen basierenden Zusatzmodulen kann der PC weiterhin als Hostrechner dienen. Durch die enge Kopplung mit der Erweiterungshardware (Realzeitsystem) können zeitunkritische aber rechenintensive Aufgaben, wie z.B. Visualisierung, Datennachbearbeitung, Entwicklung oder die Berechnung aufwändiger Algorithmen, in das PC-System verlagert und dort ausgeführt werden.

2.3.3 Vereinfachter Entwurfsprozess durch Klassifizierung

Im Gegensatz zur standardisierten PC-Hardware werden proprietäre Realzeitsysteme auf die spezifischen Anforderungen des zugrundeliegenden technischen Prozesses ausgelegt. Die zeitliche Analyse des Prozesses fließt direkt in die Spezifikation des Systems ein; alle realzeitkritischen Anforderungen können garantiert werden. Bei einem PC-basierten System sind die erreichbaren Reaktionszeiten hingegen durch die verwendete Soft- und Hardware in engen Grenzen bzw. Zeitbereichen vorgegeben. Sollen die Zeiten weiter verringert werden, muss ein erhöhter Aufwand betrieben werden; der Wechsel von Soft- und/oder Hardwarekomponenten wird notwendig.

Wird die oftmals implizierte Einschränkung von PC-basierten Realzeitsystemen auf die Verwendung von Standard-PC-Hardware mit einem Desktop-Betriebssystem aufgebrochen, so ist es möglich, eine Vielzahl von zeitkritischen technischen Prozessen mit vertretbarem Aufwand zu steuern. Die größte Schwierigkeit bereitet die Suche nach einer passenden Systemarchitektur, ohne vorher aufwändige Evaluierungen verschiedener Realisierungsmethoden durchführen zu müssen.

Eine erhebliche Vereinfachung bei der Auswahl der passenden PC-Hard- und Software zur Einhaltung einer vorgegebenen zeitlichen Anforderung liefert hier die in dieser Dissertation entwickelte **Klassifizierung der Systemarchitekturen** (Kapitel 3.3). Die Vielzahl der möglichen Kombinationen wird auf *fünf* Systemarchitekturen abgebildet. Mit diesen Realisierungsmethoden lassen sich PC-basierte Systeme zur Steuerung von (fast) allen technischen Prozessen mit zeitlichen Anforderungen im hundert Millisekundenbereich bis zum Sub-Mikrosekundenbereich realisieren. Bestehende Überlappungen der Zeitbereiche lassen Spielraum bei der konkreten Wahl der passenden Systemarchitektur.

3 Einsatz von PCs für Realzeitanwendungen

In diesem Kapitel soll, aufbauend auf den Grundlagen des 2. Kapitels, der Einsatz von PCs für Realzeitanwendungen erörtert werden. Das erste Teilkapitel 3.1 beschreibt die verschiedenen Einsatzgebiete. Es folgt eine Klassifizierung der daraus resultierenden Realzeitanforderungen in Teilkapitel 3.2 mit Klassifizierung in fünf Systemarchitekturen bzw. Implementierungsszenarien in Kapitel 3.3. Diese werden in einem neuartigen zweidimensionalen Diagramm visualisiert.

3.1 Beispiele und Einsatzgebiete

PCs werden heutzutage für unterschiedliche Aufgaben eingesetzt. Hauptbereiche sind das Büroumfeld und in zunehmendem Maß die Prozessautomatisierung, in denen der PC die Steuerungsrechner verdrängt. Die zeitlichen Anforderungen, der auf einem Büro-PC laufenden Applikationen, wie z.B. Textverarbeitung, Simulationen, Serverdienste, werden als nicht zeitkritisch eingestuft. In den folgenden Abschnitten wird deshalb nur auf zeitkritische Anwendungen in technischen Prozessen des industriellen Umfelds eingegangen. Die zunehmend komplexen Automatisierungsanwendungen stellen erweiterte Anforderungen an Installation, Betrieb und Wartung der Prozesssteuerungen. Hier befriedigt der Einsatz von flexiblen PC-basierten Steuerungen die Nachfrage nach offenen, modularen und skalierbaren Systemen. Herstellerspezifische, proprietäre und kostenintensive Entwicklungen werden Schritt für Schritt abgelöst. Moderne verteilte Prozessautomatisierungssysteme (Bild 3.1) lassen sich hierarchisch in (drei bis) vier vertikale Ebenen unterteilen [Pet99].

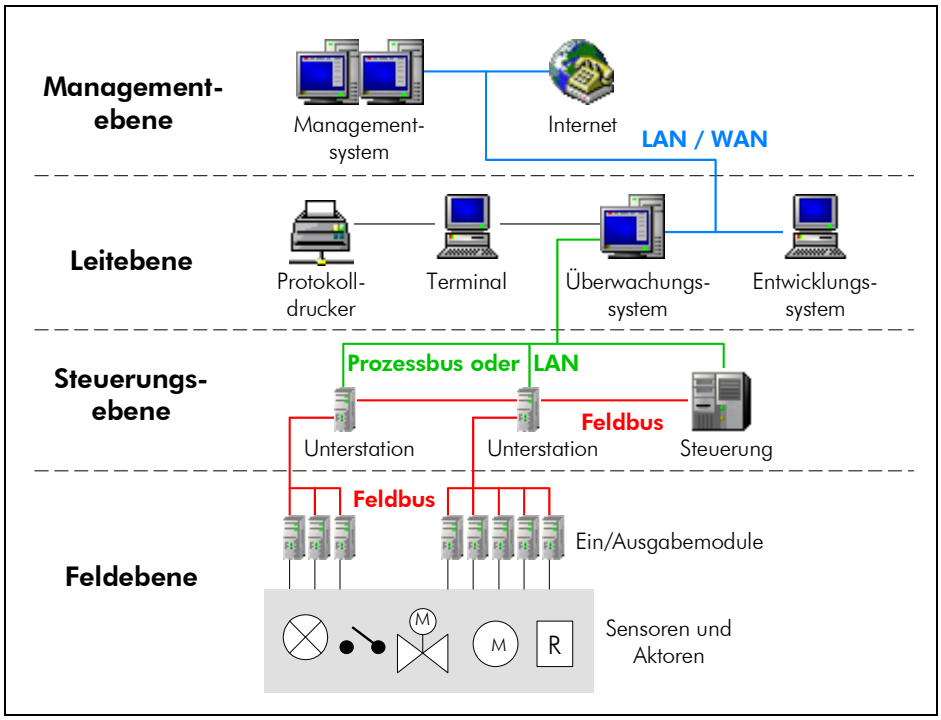


Bild 3.1 Ebenen der Prozessautomatisierung

Diese Ebenen sind:

- **Managementebene:** In dieser Ebene werden die physikalischen und technischen Daten der unteren Ebenen weiter verarbeitet. Sie umfasst Funktionen wie Energie-Management, *Facility Management* und Budgetplanung. Wesentliche Softwaremodule sind *Enterprise Resource Planning* (ERP), *Production Planning System* (PPS) und *Supply Chain Management* (SCM).
- **Leitebene:** Die Leitebene dient zur Beobachtung und Protokollierung der Prozessabläufe in den betriebstechnischen Anlagen. Softwaremodule sind hier *Supervisory Control and Data Acquisition* (SCADA), *Manufacturing Execution System* (MES) und *Process Information Management* (PIM).
- **Steuerungsebene:** Die Steuerungs- und Regelungseinrichtungen (Unterstationen) werden durch digitale Kleinrechnersysteme realisiert. Sie erhalten die zur Verarbeitung notwendigen physikalischen Daten aus der Feldebene und berechnen Stell- und Schaltbefehle. Die Hardwaremodule nennen sich *Distributed Control System* (DCS) oder *Programmable Logic Controller* (PLC/SPS).
- **Feldebene**: Die Feldebene beinhaltet die analogen und digitalen Sensoren und Aktoren der technischen Anlage.

Die Anforderungen bezüglich des Speicherbedarf steigen von der Feldebene zur Managementebene an; zeitliche Anforderungen verhalten sich entgegengesetzt und verringern sich in Richtung der Managementebene. Man kann eine Einteilung entsprechend der erforderlichen *Deadline* vornehmen:

- Weiche *Deadlines*: Mensch-Maschine-Schnittstelle
- Harte *Deadlines*: Kritische Regelschleife oder Steuersequenz eines technischen Prozesses

Die Konzipierung und Realisierung der verschiedenen Ebenen eines komplexen Prozessautomatisierungssystems lag in der Vergangenheit in den Händen mehrerer unterschiedlicher Anbieter. Dies führte zu einer Vielzahl von zueinander inkompatiblen Komponenten und einem „Preis und Technologiediktat der ‚Aller aus einer Hand' Anbieter" [Mun97]. 1994 fanden sich deshalb die großen Automobilhersteller und Großkunden der Automatisierungsindustrie General Motors, Ford und Chrysler zusammen, um eine Beschreibung der Steuerungssysteme der nächsten Generation vorzustellen. Das Ergebnis nennt sich *OMAC* (*Open, Modular Architecture Controller*), die Beschreibung einer neuen Architektur für alle Arten von industriellen Steuerungen wie z.B. SPS, CNC oder Robotersteuerungen mit Hilfe von Standardkomponenten [TYC+96].

Aus den Nachteilen der herkömmlichen Lösungen wurde ein Katalog der geforderten Eigenschaften abgeleitet und gleichzeitig Vorschläge zur Realisierung eines *OMAC*-Systems erstellt:

- **Kosten:** Niedrigere Anschaffungs- und Betriebskosten durch Nutzung von Standardkomponenten wie PC-Hard- und Software, z.B. Windows als Betriebssystem.
- **Flexibilität:** Einzelne Komponenten sind leicht austausch- und erneuerbar durch *Plug & Play*.
- **Anschlussfähigkeit:** Anschlussmöglichkeit an gängige Netzwerke durch standardisierte Schnittstellen.
- **Pflege/Wartung:** Vereinfachte Ersatzteilhaltung durch Verwendung von Standardkomponenten.

- **Ausbildung:** Einfachere Ausbildung und höhere Akzeptanz bei Mitarbeitern durch Verwendung von Standardkomponenten.
- **Sicherheit, Robustheit und Stabilität:** Diese Eigenschaften sind konträr zur Forderung nach einem flexiblen und offenen System. Hier muss ein Kompromiss gefunden werden.

Weiterhin wurden von den Firmen Vorschläge zur Software-Struktur gemacht. *OMAC*-Systeme besitzen einen Realzeit-Betriebssystemkern, eine Datenbank die auch vom Realzeitkern in Anspruch genommen werden kann und eine grafische Benutzeroberfläche (GUI). Darum herum gruppieren sich standardisierte Programmierschnittstellen für Ein/Ausgabegeräte (Sensoren, Aktoren), Antriebe, Netzwerke sowie Steuerungsprogramme für die nicht realzeitfähige, grafische Oberfläche.

Folgende Probleme der propagierten offenen und modularen Architektur dürfen nicht vernachlässigt werden:

Inkompatibilität: Standards sind oft unzureichend spezifiziert und lassen Freiraum für Interpretationen. Unterschiedliche Implementierungen arbeiten nicht fehlerfrei zusammen.

Instabilität: Die Stabilität des Gesamtsystems hängt von der Güte ihrer Einzelkomponenten ab. Haben verschiedene Hersteller unterschiedliche Auffassungen über die Qualitätssicherung, kann dies zu Instabilitäten des Gesamtsystems führen. Auch der Kompatibilitätsgrad hat hierauf Einfluss.

Eine weitere Architektur ist *OPC* (*OLE*[9] *for Process Control*), ein Industriestandard eines Zusammenschlusses führender Hersteller von Automatisierungs-Hard- und –Software und der Firma Microsoft. Hierbei wurden Methoden definiert, um einen realzeitfähigen Datenaustausch zwischen PC-basierten Steuerungssystemen und Microsoft Betriebssystemen zu ermöglichen.

Auch die *ODVA* (*Open DeviceNet Vendor Association*), eine Organisation zur Verbreitung des *DeviceNet*, standardisiert eine Netzwerktechnologie auf CAN-Bus-Basis zur einfachen Verbindung von Automatisierungssystemen.

3.1.1 Prozessautomatisierung

Die speicherprogrammierbare Steuerung (SPS)

Zur Automatisierung von technischen Prozessen wurde in den letzten 30 Jahren die sogenannte *Speicherprogrammierbare Steuerung* (SPS) eingesetzt. Die typischen kennzeichnenden Eigenschaften einer SPS sind [Bec99]:

- Die Abarbeitung eines Automatisierungsprogramms erfolgt in Zyklen. Diese haben eine typische Wiederholrate (*scan rate*) im Millisekunden-Bereich.
- Die Ausführung ist deterministisch zeitgenau mit nur sehr kleinen zeitlichen Schwankungen von wenigen Mikrosekunden.
- Die Programmierung von SPS-Programmen für sequentielle Logik erfolgt (überwiegend) nach der Norm IEC 61131-3.

[9] OLE: ***O****bject* ***L****inking and* ***E****mbedding*, ein Standard für komponentenbasierte Objekte von Microsoft.

- Antriebe können durch Programmierung nach DIN 66025 (Punkt-zu-Punkt und Interpolation) gesteuert werden.
- Verschiedene Feldbusse, PC-Schnittstellen sowie Schnittstellenkarten von Drittherstellern können angeschlossen werden.
- Es existieren Programmier- und Datenverbindungen zu Anwendungen von Visualisierung bis Tabellenkalkulation per OCX[10].
- Die Ausführungsgeschwindigkeit einer SPS für 1000 SPS-Befehle liegt zwischen 800 μs (Standard-SPS) und 70 μs (Schnelle SPS).
- Die SPS enthält Speicher für Programm, Merker und Prozessabbild.
- Der Betrieb erfolgt in Industrie-Umgebung. Die Gehäuse bieten erhöhte Schutzarten wie z.B. IP 67.
- Die Lieferbarkeit der Komponenten sollte durchschnittlich 10 Jahre betragen.

Definition: SPS (nach IEC 61131, Teil 1)

> *Ein digital arbeitendes elektronisches System für den Einsatz in industriellen Umgebungen mit einem programmierbaren Speicher zur internen Speicherung der anwenderorientierten Steuerungsanweisungen zur Implementierung spezifischer Funktionen wie z.B. Verknüpfungssteuerung, Ablaufsteuerung, Zeit-, Zähl- und arithmetische Funktionen, um durch digitale oder analoge Eingangs- und Ausgangssignale verschiedene Arten von Maschinen und Prozessen zu steuern. (...)*

Eine vereinfachte Aussage: „Eine SPS ist nichts anderes als ein Mikroprozessorsystem mit einem proprietären deterministisch arbeitenden Betriebssystem."

Es gibt eine Vielzahl von technischen Prozessen, die mit einer SPS automatisiert werden können. Einige Beispiele sind [Jan00]:

- Digitale Regelalgorithmen für Antriebe (z.B. PID-Regler)
- Bearbeitung von Ablaufsteuerungen mit Kontrolle von Grenzwerten und Rückmeldung von asynchronen Ereignissen
- Trend- und Protokoll-Informationen mit absoluten Zeiten

Die Leistungsfähigkeit moderner SPS-Systeme hängt stark von der Art der Anwendung und der eingesetzten Hard- und Softwarearchitektur ab [Roc98]. Folgenden Faktoren haben Einfluss auf die effektiv erreichbare Wiederholrate:

- Unterbrechungen durch Systemzeitgeber (*timer interrupts*)
- Bedienung von Kommunikationsschnittstellen (Tastatur, Netzwerk)
- Systeminterne Verwaltungsaufgaben
- Design des Programmcodes

Höchste Anforderungen an eine hohe Wiederholrate stellen komplexe Programmabläufe dar, wie sie z.B. bei synchronisierten mehrachsigen Antriebssteuerungen auftreten.

[10] OCX: *OLE Custom control*, ein unabhängiges Programm-Modul, dass von anderen Anwendungen eines Windows-Systems aufgerufen werden kann.

Die Software-SPS

Um den Anforderungen, z.B. der oben erläuterten *OMAC*-Beschreibung zu genügen, werden in steigendem Maße sogenannte Software-SPSen eingesetzt. Hierzu wird leistungsfähige Standard-PC-Hardware in Verbindung mit einem Standardbetriebssystem und teilweise Realzeiterweiterungen verwendet. Ergänzt mit Feldbus-Schnittstellen, einer Ablauf-Software zur zyklischen Abarbeitung von Automatisierungsprogrammen und einer grafischen Oberfläche ersetzt das System die klassische SPS. Die Hauptgründe sind laut [Hol98] die schneller fortschreitende Leistungsfähigkeit der PC-Hardware im Vergleich zur klassischen SPS-Hardware, die große Verbreitung des anwenderfreundlichen Betriebssystems Windows und die starke Datenzentrierung der aktuellen Steuerungsanwendungen (20 % SPS-Programm, 80 % Daten). Die Wiederholraten einer Software-SPS, hier am Beispiel der Softlogix-5 der Firma Rockwell, erreichen unter Windows NT auf einem Intel Pentium Prozessor (200 MHz) bis zu 8 ms. Bei Einsatz eines Doppelprozessorsystems werden Wiederholraten von < 4 ms erzielt. Ein großes Problem der reinen Software-SPS, d.h. ohne zusätzliche Maßnahmen wie Realzeiterweiterungen, ist die Ermittlung der maximalen Wiederholrate im ungünstigsten Fall (*worst case*). Diese kann, aufgrund des komplexen Verhaltens der PC-Hardware im Zusammenspiel mit dem Windows-Betriebssystem, nur durch eine (Langzeit-)Messung statistisch ermittelt werden. Die Streuung wird minimiert indem man keine zusätzlichen Applikationen auf dem Steuerungs-PC ablaufen lässt. Damit verhindert man aber nicht die Ausführung Windows-interner Systemprozesse, wie Speicherdefragmentierung, Zugriffe auf ausgelagerte Systemdateien und weiterer undokumentierter Funktionen. Eine tatsächliche Obergrenze für die Ausführungszeit lässt sich deshalb nur schwer angeben.

Auf dem Gebiet der Software-SPSen sind viele Firmen tätig. Es existieren zwei Methoden zur Realisierung einer Software-SPS. Die Erste verwendet einen Standard-PC mit Standardbetriebssystem und evtl. einer Realzeiterweiterung. Die zweite Methode basiert auf einer PC-Steckkarte mit eigenem Prozessor, welcher die SPS-Funktionalität bereitstellt. Diese beiden Methoden finden sich auch in der Klassifizierung im Kapitel 3.3. Tabelle 3.1 zeigt eine Auswahl von Produkten verschiedener Hersteller. Allerdings kann nicht jede Software-SPS harte Realzeitfähigkeit garantieren.

Hersteller	**Name**	**Soft-ware**	**Steck-karte**[11]	**Rechen-zeit**[12]	**Methode zur Erreichung der Realzeit-Funktionalität**
3S-Software	CoDeSys SP	✓	(✓)	20 μs	NMI (Steckkarte) + Kernelmode-Treiber: RT-Win, LP-Elektronik
3S-Software	CoDeSys SP NT RTE	✓		20 μs	Windows NT Realzeiterweiterung
ABB	Advant Soft Controller	✓		(30 μs)	Hochpriore Windows Task, im Speicher gelockt, keine harte Realzeitfähigkeit
ABB	Slot SPS 07 SL 97		✓	220 μs	Steckkarte mit μC
Bernecker + Rainer	SlotPLC		✓	1 ms	Steckkarte mit μC
Beckhoff	TwinCAT	✓		15 μs	Realzeitsubsystem als Treiber im Kernelmode
Bosch Rexroth AG	PCL SoftPLC		✓	60 μs	NMI + Realzeitbetriebssystem + μC: VxWin, LP-Elektronik

[11] Angabe in Klammern bei zusätzlich notwendiger Steckkarte zur Umleitung des NMI.
[12] Für 1000 SPS-Befehle, laut Herstellerangaben oder geschätzt, abhängig von der Rechenleistung des eingesetzten Prozessors. Die hier nicht angegebene Zykluszeit beträgt meist ca. 1-4 ms. Angabe in Klammern bei fehlender harter Realzeitfähigkeit.

Hersteller	Name	Soft-ware	Steck-karte[11]	Rechen-zeit[12]	Methode zur Erreichung der Realzeit-Funktionalität
CTC Parker	PCLC	✓	(✓)	20 µs	NMI (Steckkarte) + Kernelmode-Treiber: RT-Win, LP-Elektronik
Digitec	PCmatic		✓	200 µs	Steckkarte mit DIMM-PC
IBH softec	SoftSPS	✓	✓	300 µs	Realzeitfähiges Laufzeitsystem, Option: Auslagerung auf Steckkarte mit µC
Klöpper und Wiege	ProConOS NT	✓	(✓)	30 µs	NMI (Steckkarte) + Realzeitbetriebssystem: VxWin, LP-Elektronik
Nematron	OpenControl	✓		20 µs	Realzeiterweiterung: HyperKernel, Nematron
Rockwell/Allen Bradley	Softlogix	✓		(50 µs)	Hochpriore Windows Task, keine harte Realzeitfähigkeit
Softing	4Control	✓		(50 µs)	keine harte Realzeitfähigkeit
Steeplechase	VLC	✓		500 µs	Realzeiterweiterung: INtime, TenAsys
Wonderware	InControl	✓		(50 µs)	keine harte Realzeitfähigkeit

Tabelle 3.1 Software-SPSen verschiedener Hersteller[13]

Die Zukunft der SPS

In Zukunft werden sich die Anforderungen an Prozesssteuerungen weiter verändern. Der Anteil an offenen und modularen Architekturen wird weiter steigen. In den kommenden Jahren wird sich der Wettbewerb zwischen den SPS-Anbietern aufgrund der zunehmenden Konsolidierung des Marktes verschärfen. Die Unternehmensberatung Roland Berger rechnet damit, dass durch Fusionen oder strategische Allianzen nur fünf große Anbieter im Markt bleiben werden. Der Umbruch von der zentralen Steuerung hin zu dezentralen Ein/Ausgabe-Modulen ist im Gange. Diese intelligenten Module funktionieren autark am Feldbus. Antriebe arbeiten dezentral und beinhalten SPS-Funktionen (*smart drives*). Auch Sensoren und Stecker werden durch integrierte SPS-Funktionalität zunehmend intelligent. Das Erfassen von Bewegungsabläufen mit Kameras und die nachgeschaltete Bildverarbeitung wird störanfällige Sensoren entbehrlich machen. Prozesssteuerungen werden ans Internet angeschlossen und mit PCs und handelsüblichem Webbrowser bedient und gewartet. Die Software-SPS wird große Teile des Marktes übernehmen (Bild 3.2)[14].

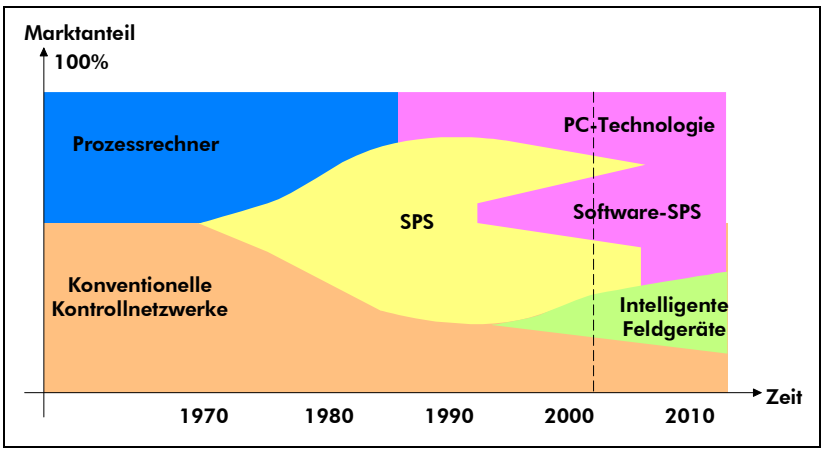


Bild 3.2 Die Zukunft der SPS

[13] siehe auch Marktübersicht in: Markt&Technik, Nr. 1/2, S.79, Jan. 2000.
[14] Quelle: Siemens / Roland Berger Unternehmensberatung.

3.1.2 Regelungen

Regelungen sind ein weiteres Einsatzgebiet für Realzeitanwendungen mit Standard-PCs. Mit leistungsstarker PC-Hardware lassen sich hochkomplexe, digitale Regler entwerfen und implementieren. So dient bei [FSL96] ein PC zur kinematischen Regelung eines 7-Achs-Roboterarms. Die Abtastzeit der Regelschleife beträgt 2,5 ms.

Beim Entwurf digitaler Regelungen und der Implementierung auf PCs müssen die Einflüsse verschiedener Verzögerungen (*delays*) einkalkuliert werden [Tör98]. In einem geregelten System können drei Arten von Verzögerungen entstehen. Erstens Verzögerungen der Sensoren bei der Messung, zweitens Verzögerungen durch Zeitdiskretisierung aufgrund periodischer Abtastung[15] und drittens Verzögerungen durch endliche Berechnungsgeschwindigkeit des Rechners. In der zeitdiskreten Regelungstheorie werden zeitliche Annahmen getroffen, die zu Einschränkungen beim Entwurf eines digitalen Reglers führen. Folgendes wird angenommen: Konstante Abtastzeit (*constant sampling period*) ohne große Schwankungen der Abtastperiode, d.h. die Digitalisierung der analogen Werte wird in äquidistanten Zeitintervallen vorgenommen, Synchronisation aller Komponenten mit einer gemeinsamen Zeitbasis (*clock*) und konstante, nicht zu große Rückkopplungsverzögerungen (*constant feedback delay*) um das dynamische Systemverhalten nicht negativ zu beeinflussen.

In digitalen Regelungen mit verteilten Sensoren und Aktoren treten zwei weitere Verzögerungen auf [Nil96]. Es entsteht sowohl eine Verzögerung bei der Datenübertragung vom Sensor zum Rechner als auch bei der Datenübertragung vom Rechner zum Aktor. Um schwankende Verzögerungen durch unterschiedliche Übertagungsraten (z.B. durch wechselnde Netzwerklast) zu minimieren und damit die Regelgesetze für zeit-invariante Systeme einsetzen zu können, wird bei dezentralisierten Regelungen oft mit Zwischenspeichern (*buffer*) in den Sensor- und Aktormodulen gearbeitet (Bild 3.3). Die dadurch entstehenden zusätzlichen Totzeiten erzeugen eine stärkere Phasenverschiebung und verschlechtern damit die Systemstabilität.

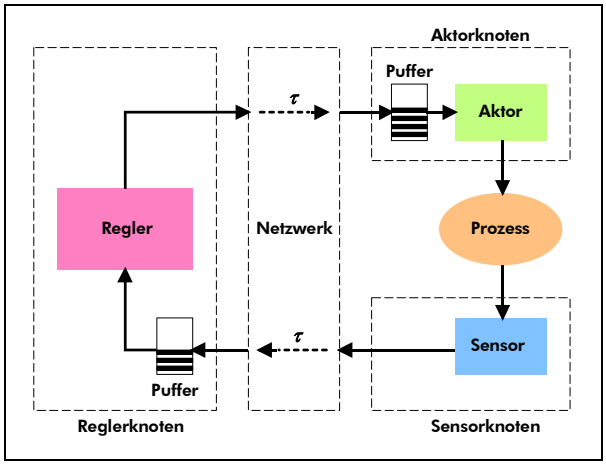


Bild 3.3 Digitaler Regelkreis mit verteilten Sensoren/Aktoren und Zwischenspeichern

Will man die Realzeitanforderungen deterministischer Regelsysteme bestimmen, so muss man immer die Zeitkonstanten des dazugehörigen physikalischen Prozesses betrachten. Als Faustregel gilt laut [Roc98], dass die Abtastrate mindestens das *sechsfache* der Prozesszeitkonstante betragen sollte. Ein damit ausgelegter PID-Regelalgorithmus ist robust, auch bei sporadischen Überschrei-

[15] Sample & Hold mit Digitalisierung = Halteglied nullter Ordnung.

tungen der Abtastrate um bis zu 50 %. Weitere Erläuterungen und praktische Ergebnisse dazu finden sich in [HB00]. Den weitaus größeren Einfluss auf die Stabilität des Reglers haben Abweichungen (Jitter) der Abtastrate. Als Faustregel in der Regelungstechnik gilt, dass die Abweichungen 5 % der Prozesszeitkonstanten nicht überschreiten sollten.

3.1.3 Antriebssteuerungen

PCs werden vermehrt auch zur Steuerung von Antrieben, *Motion Control* genannt, eingesetzt. Die gestiegenen Ansprüche, wie z.B. grafische Benutzerschnittstelle, einfache Parametrierung und komplexe Bahnplanung für Multi-Achsensysteme, erfordern eine höhere Rechenleistung als klassische Antriebssteuerungen bieten können [ERJ+00]. PC-basierte Antriebssteuerungen vereinfachen auch die Implementierung neuer Regelalgorithmen. Die integrierten Schnittstellen, speziell PCI- und USB-Bus, ermöglichen die einfache Anbindung an Feldbusse mittels Steckkarten und die Vernetzung mit intelligenten Sensoren und Aktoren. Gerade die Evolution der Antriebssteuerungsschnittstellen ermöglicht dezentrale digitale Regelkreise. [Pre99] beschreibt den Übergang von der analogen zur digitalen Schnittstelle. Der analoge Schnittstellenstandard (10 V = volle Geschwindigkeit) wurde Ende der 80er Jahre mit Hilfe von digitalen Signalprozessoren zu einer Drehmomentsteuerung (10 V = volles Drehmoment) erweitert. Gleichzeitig begann mit *SERCOS*[16] die Einführung eines Antriebssteuerungs-Bussystems mit rein digitaler Übertragung. *SERCOS* basiert auf einem Master/Slave-Businterface mit mehreren Kanälen zur zyklischen Übertragung von synchronen Daten [QW94]. Die physikalische Anbindung kann als Ring- oder Busstruktur mit RS485- oder optischer Übertragung für maximal 254 Teilnehmer realisiert werden. Bei einer maximalen Datenrate von 4 MBit/s beträgt die minimale Zykluszeit 62 μs. Regelkreise für Position, Geschwindigkeit und Kraft/Drehmoment (Strom) werden innerhalb des Antriebs geschlossen. Deren Abtastzeiten innerhalb der Antriebssteuerung betragen 2, 4, und 16 kHz. Nur die Sollwertvorgaben werden über den *SERCOS*-Bus übertragen. Inzwischen existiert eine einheitliche Software-Schnittstelle mit dem Namen *SoftSERCANS*, welche von der *SERCOS Interessen Gruppe* vorgeschlagen wurde. Um kostengünstige passive *SERCOS*-Schnittstellenkarten einsetzen zu können, läuft diese Software auf einem Standard-PC-System mit Windows NT und der Realzeiterweiterung *RTX* von *VenturCom* (siehe auch Kapitel 4.2.1).

Hier kann der Einsatz eines standardisierten Bussystems, z.B. IEEE 1394 (*FireWire*) aus dem PC-Umfeld die Kosten reduzieren. Zudem bietet er eine wesentlich höhere Übertragungsleistung (bis 400 MBit) als *SERCOS*. In [Hez01] wird die Verwendung von digitalen Antriebssteuerungen mit IEEE 1394-Schnittstelle für komplexe kinematische Regelalgorithmen von modernen Maschinen beschrieben. Diese anspruchsvollen Geräte, wie z.B. *Wafer-Stepper* oder Mehrachsen-Spezialroboter, besitzen eine hohe Dynamik und erfordern eine Synchronisation im μs-Bereich.

Zusammenfassend die wichtigsten Vorteile digitaler Antriebssteuerungen: größere Störfestigkeit, größere Flexibilität, ausführliche Diagnosemethoden, einfache Installation und Vernetzung, Einsatz von komplexeren Regelalgorithmen und geringere Kosten. Aufgrund der zusätzlichen Schnittstellen-Kosten, insbesondere bei kleinen Antrieben, erreichen digitale bus-basierte Antriebssteuerungen erst einen geringen Marktanteil von 9 % (1999) [Pre99].

[16] SERCOS: ***SE****rial* ***R****eal-time* ***CO****mmunication* ***S****ystem* (IEC 61491), standardisierte, digitale, optische Schnittstelle zur Kommunikation zwischen Steuerungssystem und Antrieb.

3.1.4 Datenübertragung

Die Realzeitanforderungen bei der Übertragung zeitkritischer Daten mit PC-basierten Systemen werden durch das Anwendungsgebiet, die Art der Daten und den Empfänger bestimmt. So sind für Audio- und Video-Datenströme im Multimediabereich, aufgrund psychoakustischer und psychooptischer Eigenschaften der menschlichen Sinnesorgane als Empfänger, weniger harte Zeitbedingungen erforderlich, als z.B. für die Steuerung von Maschinen.

Die Datenrate eines unkomprimierten Stereo-Audiosignals beträgt ca. 1,4 MBit/s, mit verlustbehafteter MPEG Audio Layer-3 Komprimierung (MP3) nur zwischen 128 – 256 KBit/s [BSH98]. Unkomprimierte Video-Daten haben wesentlich höhere Datenraten im Bereich von bis zu 270 MBit/s, nach einer Komprimierung im MPEG-2-Format noch ca. 4 – 10 MBit/s. [MHE+97] zeigt, dass Rechner bei der Verarbeitung von Video eine 5 – 10 mal höhere I/O-Bandbreite (Speicherinterface des Prozessors) zur Ausführung der Algorithmen benötigen als für die reine Datenübertragung, d.h. für das Laden und Abspeichern der Bilder.

Die Charakteristiken von Datenströmen in verteilten, kommunizierenden Systemen, speziell für Multimedia-Anwendungen, beschreibt ausführlich [Spi96]. Der Datenstrom wird in Pakete zerlegt und übertragen. Es werden drei Klassen unterschieden: Asynchrone, synchrone und isochrone Datenübertragung. Bei der asynchronen Übertragung sollen die Pakete „so schnell wie möglich“ ihr Ziel erreichen. Bei der synchronen Übertragung wird eine *maximale Ende-zu-Ende-Verzögerung* für die Ankunft eines Pakets garantiert. Pakete dürfen jedoch auch früher eintreffen. Die synchrone Übertragung ist die Basis für Multimedia-Anwendungen. Allerdings müssen zu früh ankommende Daten zwischengespeichert werden; dies erfordert große Pufferspeicher bei hohen Datenraten. Dagegen ist bei der isochronen Übertragung auch die *minimale Ende-zu-Ende-Verzögerung* definiert. Hier kann auf große Zwischenspeicher verzichtet werden. Allerdings wird für die isochrone Übertragung kontinuierlicher Daten eine reservierte Kommunikationsbandbreite benötigt.

Beim Einsatz eines Netzwerks und intelligenten Netzwerk-Interfacekarten mit integriertem Prozessor zur Multimedia-Datenübertragung, kann es nach [FMO+98] sinnvoll sein, diese Rechenleistung zur Datenvorverarbeitung und damit zur Entlastung des Hauptprozessors zu benutzen. Allerdings sollte hierbei die Rechenleistung der Prozessoren in derselben Größenordnung liegen.

3.2 Klassifizierung der Eigenschaften

Die Anforderungen von Realzeitanwendungen sind, wie bereits in Teilkapitel 3.1 dargestellt, sehr unterschiedlich. Es muss immer auf die genauen Eigenschaften des zu realisierenden Systems und dessen Randbedingungen eingegangen werden. Zuvor sollen die Begriffe „Echtzeit“ und „Realzeit“ genauer erläutert werden.

Echtzeit und Realzeit

„Echtzeit“ ist nur ein anderes Wort für „Realzeit“. Leider wird in der Werbung und im täglichen Leben der Begriff „Echtzeit“ häufig missbräuchlich verwendet. Ausschlaggebend hierfür ist der Boom in der PC- und Multimediabranche, der zu einem Wettlauf von Taktraten und Verarbeitungszeiten geführt hat. „'Gut ist, was schnell ist', das suggeriert die Werbung unter dem Begriff Echtzeit“

[HB00]. Im Englischen dagegen gibt es nur den Begriff *Real-time*. Deshalb wollen wir hier fortan *Realzeit* als exakten und wissenschaftlich korrekten Begriff verwenden.

Die DIN-Norm 44300 definiert den Begriff *Echtzeitbetrieb* (Realzeitbetrieb) folgendermaßen:

> *Echtzeitbetrieb ist ein Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorbestimmten Zeitpunkten anfallen.*

Die DIN-Norm reduziert hierbei die Realzeitfähigkeit auf garantierte, maximale Reaktionszeiten (*bounded response*). Eine vorteilhafte Eigenschaft für ein Realzeitsystems ist die *Multitasking-Fähigkeit* [Jan00]. Viele technische Prozesse setzen sich aus mehreren, miteinander gekoppelten und zueinander parallel ablaufenden Teilprozessen zusammen. Oft ist es schwierig und unübersichtlich, diese Prozesse durch ein sequentiell ablaufendes Programm zu steuern. Das *Multitasking*, welches durch Hard- oder Software unterstützt werden muss, ermöglicht eine *quasiparallele* Abarbeitung verschiedener Teilaufgaben in einer Ein-Prozessor-Umgebung, die ein Standard-PC-System üblicherweise darstellt.

Stankovic und Ramamritham [SR93] beschreiben Realzeitsysteme folgendermaßen:

> *Real-time systems are those systems in which the correctness of the system depends not only on the logical result of computations, but also on the time at which the results are produced.*

Die korrekte Funktion eines Realzeitsystems hängt also nicht nur von der *Richtigkeit* des Ergebnisses, sondern auch von der *Rechtzeitigkeit* der Bereitstellung ab.

Voraussetzungen für den Realzeitbetrieb

Systeme für den Realzeitbetrieb müssen grundlegende Voraussetzungen erfüllen:

- Einhalten von Zeitspannen (*deadlines*)
- (Quasi)Simultane Verarbeitung von gleichzeitigen Ereignissen (*multitasking*)
- Sichere Verwaltung von Prozessen und Systemspeicher, Bereitstellen von Mechanismen zur Task-Kommunikation und Synchronisation (*task management*)
- Sicherstellung der Unterbrechungsfähigkeit (*interrupt management*)
- Definierte Reaktion auf Fehlerzustände (*error handling*)
- Management der Hardware und Schnittstellen (*device driver management*)

3.2.1 Harte und weiche Realzeitsysteme

Die Gültigkeitsdauer von Eingangsdaten bzw. Änderungsgeschwindigkeit von externen Ereignissen bestimmen die Zeitspannen (*deadlines*) der Realzeitsysteme. In der Praxis wird oft eine Unterscheidung in *harte* und *weiche* Realzeit vorgenommen [Wol97]. Hierbei wird das Überschreiten einer Zeitspanne bei harter Realzeit als totales Versagen des Gesamtsystems definiert, das fatale Auswir-

kungen hat. Bei weicher Realzeit wird das Überschreiten von Zeitspannen toleriert, wobei der entstehende Schaden gering ausfällt (siehe auch Kapitel 2).

Die Merkmale eines harten Realzeitsystems sind:

- Verzögerungen werden unter keinen Umständen akzeptiert
- Ergebnisse sind nutzlos, wenn sie zu spät vorliegen
- Auswirkungen und Kosten im Fehlerfall sind katastrophal

Beispiele für harte Realzeitsysteme sind:

- Regelung eines Kernreaktors
- Regelung kritischer chemischer Prozesse
- Antiblockiersystem im Auto
- Überwachungssystem auf einer Intensivstation
- Flugzeugregelung

Demgegenüber lassen sich weiche Realzeitsysteme folgendermaßen charakterisieren:

- Verzögerungen ergeben eine tolerierbare niedrigere Leistung
- Verspätetes Eintreffen von Ergebnissen erzeugt vertretbare Kosten

Beispiele für weiche Realzeitsysteme sind:

- Digitale Telefonanlage
- Steuerung unkritischer chemischer Prozesse
- Komfortsysteme im Auto
- Multimediale Anwendungen
- Verkaufsautomat

Die Anzahl der eingesetzten weichen Realzeitsysteme auf Standard-PC-Basis wächst seit einigen Jahren sehr stark. Als Auslöser dieses Trends gelten anspruchsvolle und leistungsintensive Anwendungen, wie Bildverarbeitung, Internet-Applikationen, Telerobotik und virtuelle Realität [BN00]. Diese Anwendungen haben sehr hohe Datendurchsatzraten und erfordern die Einhaltung *weicher* Zeitbedingungen. Ähnlich wie in traditionellen Realzeitsysteme, bei denen Vorwissen über den ungünstigsten Fall (*worst case*) vorhanden ist, sollte die Verarbeitungszeit eine obere Grenze möglichst nicht überschreiten. Alle Routinen, für die es keine Garantie über die maximale Ausführungszeit gibt, sind zu vermeiden. Kommt es dennoch zu Verzögerungen, werden diese toleriert.

Einige Standardbetriebsysteme enthalten Mechanismen, wie z.B. spezielle hohe Realzeit-Prioritäten, für die Ausführung weicher Realzeitanwendungen. Zur Verifikation eines weichen Realzeitsystems werden die realen Ausführungszeiten unter Stressbedingungen und über einen ausreichend langen Zeitraum gemessen, um die praktischen Durchschnitts-, Minimal- und Maximalwerte für die *deadlines* zu ermitteln. Im Gegensatz dazu erfordern harte Realzeitsysteme den theoretischen Nachweis, um die Einhaltung der *deadlines* im *worst case* zu garantieren.

Windows NT ist ein typisches Standardbetriebssystem, das für die Realisierung von weichen Realzeitsystemen eingesetzt wird. Es wird z.B. von der Firma Siemens zur Implementierung von spei-

cherprogrammierbaren Steuerungen (siehe Kapitel 3.1.1) verwendet. Falls auch kritische Prozesse mit harter Deterministik gesteuert werden sollen, besteht die Möglichkeit, das System durch Einbau einer zusätzlichen Steckkarte mit eigenem Realzeitsystem (hier: Slot-SPS) zu erweitern, um so eine hochverfügbare und fehlersichere Verarbeitung zu gewährleisten. Auch mit softwarebasierten Realzeiterweiterungen (Kapitel 4.2) wird versucht, die Latenzzeiten der Standardbetriebssysteme zu verringern um obere Schranken für die Reaktionszeiten garantieren zu können.

Beim Einsatz eines Realzeitbetriebssystems muss zur optimalen Zuordnung der unterschiedlichen Prioritäten der zugrundeliegende technische Prozess genau analysiert werden. Anhand des ermittelten Unterbrechungsszenarios können die *worst case*-Programmpfade gefunden und damit die jeweiligen Reaktions- und Ausführungszeiten bestimmt werden. Der alleinige Einsatz eines Realzeitbetriebssystems ist noch keine Garantie für harte Realzeitfähigkeit. Auch eine hohe Systemgeschwindigkeit stellt kein Maß für die Einhaltung von Realzeitbedingungen dar [Roc98].

3.2.2 Typische Parameter

Bevor auf die typischen Parameter von Realzeitanwendungen eingegangen wird, sollen kurz die Obergrenzen der Anforderungen an Realzeitsysteme an einem anspruchvollen technischen Prozess umrissen werden. Als aktuelles Beispiel dient die in [Gut99] beschriebene Datenübertragung und –verarbeitung für den neuen *Large Hadron Collider* (LHC) der Europäischen Organisation für nukleare Forschung (CERN) in der Schweiz, der 2005 in Betrieb gehen soll.

Obergrenzen von Realzeitanforderungen

In der physikalischen Grundlagenforschung werden große Teilchenbeschleuniger eingesetzt um Kollisionen zwischen Protonen zu erforschen. Jeder Zusammenprall erzeugt in einem Detektor Messdaten in der Größe von 1 MB. Der Abstand zwischen zwei Kollisionen beträgt 25 ns, was einer Rate von 40 MHz entspricht. Das zu bewältigende Datenvolumen würde somit ca. 40 TB/s betragen. Eine solche Menge an Rohdaten kann nicht direkt verarbeitet werden und muss in mehrere Stufen mit jeweiliger Datenvorverarbeitung und –reduktion aufgeteilt werden. Die erste Stufe wird vollständig in Hardware realisiert und räumlich direkt am Detektor platziert. Mustererkennungsalgorithmen und Ausfiltern uninteressanter Messungen reduzieren die Datenrate von 40 MHz auf 100 kHz (Faktor 400). Da die einzelnen Kollisionen voneinander unabhängig sind, kann in einer zweiten Stufe eine Aufspaltung in mehrere parallele Prozesse erfolgen und der Gesamtdatenstrom von 100 GB/s in 1000 einzelne Ströme à 100 MB/s aufgetrennt werden. Eine Rechnerfarm, aufgebaut aus 1000 PCs, kann voraussichtlich zum Zeitpunkt der Inbetriebnahme im Jahr 2005 die erforderliche Gesamtleistung von 5 Tera-OPs/s[17] bereitstellen. Die weitere Verarbeitung erfolgt dann mit schnellen Netzwerken und weiteren Rechnersystemen.

Parameter von Realzeitsystemen

Aus den Anwendungsbeispielen der vorangegangen Kapitel lassen sich Parameter von Realzeitsystemen ableiten. Folgende Parameter werden in den kommenden Abschnitten beschrieben:

[17] Tera-Operationen pro Sekunde.

- Zeiten
  - Reaktionszeit
  - Interrupt-Zeit
  - Kommunikationszeit
  - Erfassungszeit
  - Ausgabezeit
  - Task-Umschaltzeit
- Jitter
- Sicherheit und Zuverlässigkeit

Reaktionszeit und andere Zeiten

Der wichtigste Parameter von Realzeitsystemen ist die auftretende *Reaktionszeit*. Diese setzt sich zusammen aus:

Reaktionszeit = Ausführungszeit + Wartezeit

Die *Ausführungszeit* oder auch *Verarbeitungs-* bzw. *Rechenzeit* entsteht durch die serielle Struktur des Prozessors und der nur quasiparallelen Abarbeitung von Aufgaben im Rechnersystem. Zur Reaktion auf externe Ereignisse wird der Programmfluss durch Unterbrechungen (*interrupts*) angehalten und in eine *Interrupt-Service-Routine* (ISR) verzweigt. Die dabei entstehende *Wartezeit* oder auch *Latenzzeit* ist abhängig von der Prozessor-Architektur, der I/O-Anbindung, den gerade laufenden Tasks, den Prioritäten und weiteren Faktoren.

Die maximal zulässige Reaktionszeit, in der auf das externe Ereignis reagiert werden muss, wird durch den technischen Prozess vorgegeben. Eine Überschreitung des *letztmöglichen Reaktionszeitpunkts* (*deadline*) führt zur Verletzung der Realzeitbedingung und muss bei harten Realzeitsystemen unter allen Umständen verhindert werden. Bei der quasiparallelen Ausführung von mehreren Aufgaben durch ein Rechnersystem (mit einem Prozessor) muss eine geeignete Zuteilungsstrategie (*schedule*) gewählt werden. Beim prioritätsgesteuerten *Rate Monotonic*-Scheduling (RMS), wird den Prozessen eine Priorität zugeteilt, wobei normalerweise Prozessen mit kurzen Prozesszeiten eine hohe, und Prozessen mit langen Prozesszeiten eine niedrige Priorität zugewiesen wird. Andere Schedulingverfahren, wie z.B. das *Earliest Deadline First*-Scheduling (EDF), benötigen keine Prioritäten, sondern bewerten die Rechenprozesse nach ihrer noch zur Verfügung stehenden Zeitspanne. Dieses Verfahren kann, sofern es die gegebenen Anforderungen ermöglichen, eine maximale Auslastung von 100 % erreichen und gilt als optimales Verfahren [Fär94].

Bei ereignisgesteuerten Systemen auf Mikroprozessorbasis sind die Verzögerungszeiten aufgrund der notwendigen Reaktion auf externe Signale und der damit verbundenen Unterbrechung des Programmflusses von großer Bedeutung.

Diese sehr wichtige Zeit wird definiert als:

Interrupt-Latenzzeit = Zeit vom Auftreten des Ereignisses bis zur Ausführung des ersten Befehls der ISR

Die *Interrupt-Latenzzeit* oder *Interrupt-Antwortzeit* (*interrupt latency* or *interrupt response time*) ist die Zeitspanne, die vom Auftreten des externen Ereignisses (Schritt 1) bis zur Ausführung des ersten

Befehls der ISR (Schritt 11) verstreicht. Voraussetzungen zur genauen Ermittlung dieser Zeit sind ein freigeschalteter Interrupt-Eingang und kein Auftreten anderer Interrupts (oder NMI[18]) während der Bearbeitung. Eine genaue Beschreibung der auftretenden Interrupt-Zeiten zeigt der Ablaufplan in Tabelle 3.2. Er gilt im Allgemeinen für alle x86-Prozessorarchitekturen [Int99]. Der Intel 386SX-Prozessor hat eine Interrupt-Latenzzeit von ca. 100 Taktzyklen, was ca. 5 µs bei 20 MHz entspricht. Der Pentium MMX mit 233 MHz benötigt ca. 2 µs, ein Pentium II 400 MHz ca. 1,2 µs.

Schritt	Vorgang	Beschreibung
1	Auftreten des externen Ereignisses (Hardware-Interrupt)	Eine der externen Interrupt-Leitungen geht auf *high* (Leitungen IR0 – IR15 der kaskadierten PIC 8259A[19] Bausteine)
2	Erkennen des Interrupts durch den Prozessor	Der PIC legt seine INT Leitung auf *high*, welche direkt mit dem INTR-Eingang des Prozessors verbunden ist.
3	Ausführen des letzten vorangegangenen Befehls	Der Prozessor führt die aktuelle Instruktion zu Ende aus.
4	Bestätigen (*acknowledge*) des Interrupts	Der Prozessor sendet einen ersten kurzen Impuls auf der \INTA-Leitung[20] zum PIC. Dieser berechnet die Priorität des Interrupts.
5	Auslesen des Interrupt-Vektors	Der Prozessor sendet einen zweiten kurzen Impuls auf der \INTA-Leitung zum PIC. Der PIC legt daraufhin den *interrupt type*[21] auf den Datenbus (D0 – D7).
6	Bestimmen der Interrupt-Quelle	Der Prozessor berechnet daraus den Tabellenort des *interrupt vectors* (*type* x 4).
7	Sichern des Prozessorzustands	Der Prozessor sichert das EFLAGS-Register auf den Stack.
8	Sperren der Interrupts	Der Prozessor löscht Interrupt- und Trap-Flag des EFLAG-Registers.
9	Vorbereitung für Rücksprung	Der Prozessor sichert das *code segment*-Register (CS) und den *instruction pointer* (IP) auf den Stack.
10	Sprung zur ISR	Der Prozessor lädt die Adresse der ISR in CS und IP.
11	Ausführen des ersten Befehls der ISR	Der Prozessor holt die erste Instruktion (der ISR) und führt diese aus.

Tabelle 3.2 Ablauf bei Auftreten eines Interrupts

In Betriebssystemen werden meist Interrupt-Handler für eine einfachere Verwaltung der Interrupt-Service-Routinen integriert. Diese Handler führen aufgrund des erhöhten Verwaltungsaufwands zu einer verlängerten Interrupt-Antwortzeit.

In einem Rechnersystem lassen sich noch weitere Zeiten bestimmen. Um Begriffsirritationen zu Vermeiden sollen die Definitionen der verschiedenen Zeiten hier kurz beschrieben werden. Die stets vorhandenen, physikalisch bedingten Verzögerungen durch Signallaufzeiten werden hier nicht näher betrachtet.

Kommunikationszeit = Arbitrierungszeit + Datenübertragungszeit

Kommunikationszeiten treten bei Datenübertragungen zwischen verschiedenen Komponenten auf. Gerade bei Bussystemen addieren sich zusätzliche Zeiten aufgrund der Arbitrierungsphase vor der

[18] NMI: ***N**on-**M**askable **I**nterrupt*, nicht maskierbarer Interrupt.
[19] PIC: ***P**rogrammable **I**nterrupt **C**ontroller*.
[20] Der *backslash* „\" kennzeichnet *low-active* Signale.
[21] Hinweis: Der *interrupt type* ist nicht identisch mit der *interrupt number* oder dem *interrupt vector*.

eigentlichen Übertragungsphase. Diese Zeiten hängen stark vom verwendeten Arbitrierungsalgorithmus und den maximal möglichen exklusiven Zugriffszeiten ab. So definiert z.B. beim PCI-Bus der *latency timer* die maximale Zeitspanne, die ein Busmaster-Gerät den Bus belegen darf. Typischerweise sind dies 32 Bustakte (= 960 ns bei 33 MHz). Die *Datenübertragungszeit* ist somit die benötigte Zeit für die Übertragung der Daten inklusive sämtlicher durch den Überhang des Busprotokolls bedingten Zeiten. Zusätzliche Kommunikationszeiten entstehen auch bei verteilten Systemen, die Daten über Netzwerke versenden und empfangen [WNT95]. Eine Beschreibung von möglichen Verzögerungen speziell bei Regelungen gibt Kapitel 3.1.2. Durch Verlust oder Verfälschung der Daten während der Netzwerkübertragung können zusätzliche Verzögerungen entstehen. Bei kurzen Störungen und robusten, technischen Prozessen kann eine Extrapolation fehlender Werte Verzögerungen vermeiden. Ist die Störung nicht tolerierbar, muss ein *Fail-safe*-Verhalten sichergestellt sein [Fär94].

Erfassungszeit = Zeitspanne vom Auftreten des Ereignisses bis zum Erkennen des Ereignisses

Die *Erfassungszeit* ist die Zeitspanne, welche zwischen dem Auftreten des externen Ereignisses (meist einer Signalflanke) und dessen sicheren Erkennen (im Prozessor bzw. PIC) vergeht. Die beiden Hauptursachen für Verzögerungen sind Synchronisierungszeiten mit dem Prozessortakt und prozessorinterne Einheiten zur Filterung von Störimpulsen auf den Interrupt-Leitungen.

Ausgabezeit = Zeitspanne vom Anlegen der Antwort am Ausgang bis zur vollständigen Reaktion des Aktors

Die Zeitspanne zwischen Aktivierung eines Ausgangssignals bis zur vollständigen Reaktion eines Aktors nennt man *Ausgabezeit*. Bei physikalischen Prozessen wird als Schwelle, ab der eine Reaktion als *vollständig* gilt, meist ein Erreichen von 90 % des Endwertes definiert.

Eine weitere Verzögerungszeit, die *Task-Umschaltzeit* (*task switching time*), ergibt sich durch den Einsatz von Multitasking-Betriebssystemen zur Steuerung und Regelung von zeitkritischen technischen Prozessen [OL99]:

Task-Umschaltzeit = Zeit vom Aktivieren des Schedulers bis zur Ausführung des ersten (User-)Befehls der neuen Task

Sie bezeichnet die Zeitspanne die zwischen Auftreten z.B. eines Timer-Interrupts zur Aktivierung des Schedulers und dem vollendetem Taskwechsel, d.h. der Ausführung des ersten Befehls der neuen Task verstreicht. Die *Task-Laufzeit* (*task turnaround time*) ist die Zeit vom Aktivieren des Schedulers bis zur Beendigung der neuen Task. Diese Zeit entspricht der maximalen Aktivierungsrate bei periodischen Tasks. Voraussetzung: Keine höherpriore Task wird zwischenzeitlich aktiviert.

Erwähnenswert sind auch die Probleme der hohen Latenzzeiten bei komplexen Programmabläufen wie sie in Software-Protokollen und –Stacks (z.B. TCP/IP) auftreten. Neue Ansätze zur Verringerung dieser Verzögerungen werden in Kapitel 5.3 aufgezeigt.

Jitter

In Kapitel 3.1.2 wurde zum ersten mal der Begriff *Jitter* als Ausdruck für die Abweichung der realen von der *idealen* Ankunftszeit einer Signalflanke bzw. für die Abweichungen von der spezifizierten Dauer eines Zeitintervalls verwendet. Der Begriff stammt aus dem Englischen und bedeutet „Zittern". Eine genaue Definition liefert die IEEE [Tör98]:

> *Jitter is the time–related, abrupt, spurious variation in the duration of any specified related interval.*

In digitalen Systemen gibt es eine Vielzahl von Ursachen für Jitter. Er besteht grundsätzlich aus zwei Komponenten, dem *systematischen* und dem *zufälligen Jitter*. Die Summe beider Jitter-Komponenten ergibt den Gesamt-Jitter. Bei einer Messung ist es ist oft schwierig, die jeweiligen Anteile zu identifizieren.

Systematischer Jitter kann entstehen durch: Art der Implementierung, strukturbedingten Restriktionen, Architekturbeschränkungen, Bauteilspezifikationen u.ä.

Zufälliger Jitter hat meist physikalische Ursachen, wie Temperaturschwankungen, Drift, Alterung, Rauschen, Signalverzerrungen u.ä.. Zwei wichtige zufällige Jitterursachen sind *Rauschen* und *Signalverzerrungen* des Übertragungskanals. Bei der ersten Ursache ist das Histogramm des Jitters gewöhnlich näherungsweise normalverteilt, die zweite Ursache hat ein vom Bitmuster abhängiges Histogramm das im Allgemeinen nicht normalverteilt ist.

Häufig werden die Begriffe Jitter und Latenzzeit fälschlicherweise gleichgesetzt. Die Latenzzeit ist ein Maß für die absolute Verzögerung einzelner autonomer Dateneinheiten zwischen dem Sende- und Empfangsvorgang. Diese gibt die Laufzeit eines Bits vom Sender zum Empfänger an. Im Gegensatz dazu ist Jitter die Schwankung dieser Laufzeit. Datendurchsatz und Jitter sind damit für einen isochronen, kontinuierlichen Datenfluss, wie er bei multimedialen Daten auftritt (siehe Kapitel 3.1.4), die entscheidenden Qualitätsparameter. Latenzzeit und Jitter spielen dagegen in Realzeitsystemen eine wichtige Rolle, wie sie z.B. in Produktionsumgebungen eingesetzt werden. Dort ist es wichtig, dass eine Steuerungsinformation innerhalb einer fest definierten Zeit ohne zu große Abweichungen bei einer Maschine ankommt. Der Jitter hat hierbei einen signifikanten Einfluss auf die Stabilität von digitalen Regelkreisen. Auch bei der Synchronisation von mehreren Aufgaben, z.B. bei der Messdatenerfassung von vielen Kanälen, verschlechtert ein zu großer Jitter die Qualität der anschließenden Auswertung.

Sicherheit und Zuverlässigkeit

Außer dem reinen Zeitverhalten spielen für ein Realzeitsystem mit kritischem Anwendungsgebiet die Faktoren *Sicherheit* und *Zuverlässigkeit* eine tragende Rolle [Roc98]. Ein sicheres System muss im Fehlerfall in den *Fail-Safe*-Zustand übergehen. Die wichtigsten Fehlerfälle sind:

- Anhalten der Befehlsausführung (fehlerhafter Code)
- Steckenbleiben in einer Endlosschleife (fehlerhafter Code)
- Auftreten einer unerwarteten Ausnahme (*exception*) (fehlerhafter Code)
- Versagen der Stromversorgung

- Bitfehler des Systemspeichers

Durch das Ausnahmemanagement des Prozessors können viele Fehler detektiert werden. Eine zeitliche Sicherungseinrichtung (*watch-dog timer*) hilft z.B. bei Verklemmungen durch Zurücksetzen des gesamten Systems.

3.3 Klassifizierung der Systemarchitekturen

3.3.1 Haupt- und Untergruppen

Im folgenden Abschnitt soll nun die Klassifizierung der Systemarchitekturen für technische Prozesse beschrieben werden. Zuerst erfolgt eine Unterteilung der zahlreichen PC-basierten Systeme zur Steuerung und Regelung von zeitkritischen technischen Prozessen in die zwei Hauptgruppen:

1. **Softwarebasierte Systemarchitekturen**
2. **Hardwarebasierte Systemarchitekturen**

Zur ersten Gruppe der softwarebasierten Systemarchitekturen gehören alle Standard-PC-Systeme. Eine weitere Unterteilung wird nun anhand der verschiedenen eingesetzten Betriebssysteme vorgenommen, die zu einer entsprechenden Verbesserung der Realzeiteigenschaften führen.

1.a.	**Reine Software-Implementierungen**	**(OS)**
1.b.	**Realzeiterweiterungen für Standardbetriebssysteme**	**(OS + E)**
1.c.	**Realzeitbetriebssysteme**	**(RTOS)**

Zur ersten Untergruppe zählen alle Standard-Betriebssysteme, wie Windows (NT), Linux usw. Der zweite Bereich mit verbesserten Realzeiteigenschaften besteht aus Standard-Betriebssystemen in Verbindung mit sogenannten Realzeiterweiterungen. Reine Realzeitbetriebssysteme bilden die dritte Untergruppe mit nochmals verringerten Reaktionszeiten. Diese softwarebasierten Systemarchitekturen werden in Kapitel 4 ausführlich erläutert.

Der zweiten Gruppe der hardwarebasierten Systemarchitekturen gehören alle Erweiterungen der Standard-PC-Hardware an. Diese werden nach den verwendeten Methoden in zwei Untergruppen eingeteilt:

2.a.	**Intelligente Controller-Komponente**	**(µC/DSP)**
2.b.	**Programmierbare Logik-Komponente**	**(PLD)**

Zum ersten Bereich zählen intelligente Controller-Komponenten (eingebettete Systeme), die mit ihrer programmierbaren Eigenintelligenz Vorverarbeitung und Entscheidungsfindung vom PC-Prozessor übernehmen können. Die zweite Untergruppe basiert auf programmierbaren Logikbausteinen wie FPGA[22] und CPLD[23] und wird für sehr zeitkritische Aufgaben im Sub-Mikrosekunden-Bereich eingesetzt. Eine ausführliche Analyse dieser hardwarebasierten Systemarchitekturen erfolgt im Kapitel 5.

[22] FPGA: ***F****ield* ***P****rogrammable* ***G****ate* ***A****rray*.
[23] CPLD: ***C****omplex* ***P****rogrammable* ***L****ogic* ***D****evice*.

Zusätzlich gibt es noch zwei weitere, sinnvolle Kombinationsmöglichkeiten. Ein Standard-PC-System mit einem Realzeitbetriebssystem kann durch Hinzufügen einer Ein/Ausgabe-Komponente ohne Eigenintelligenz ergänzt werden. Die minimal erreichbare Reaktionszeit wird hiervon nicht beeinflusst; die Komplexität der zu bearbeitenden Aufgaben kann hingegen bei gleichbleibend kleinen Zeiten weiter erhöht werden. Die zweite Kombination setzt sich aus der intelligenten Controller-Komponente und der Ein/Ausgabe-Komponente ohne Eigenintelligenz zusammen. Auch hier können Anwendungen mit höherer Komplexität bei gleichbleibend kleinen Reaktionszeiten verarbeitet werden:

- **Realzeitbetriebssysteme + Ein/Ausgabe-Komponente (RTOS + I/O)**
- **Intelligente Controller-Komponente + Ein/Ausgabe-Komponente (µC/DSP + I/O)**

Die Grafik in Bild 3.4 zeigt einen Überblick über die **Klassifizierung** der Untergruppen der beiden Systemarchitekturen und zusätzlich die beiden Kombinationen. Die jeweiligen Flächen der verschiedenen Methoden zeigen die damit realisierbaren Bereiche im Bezug auf die erreichbaren Reaktionszeiten und der entsprechenden Komplexität der zu bearbeitenden Aufgabe. Als (fiktive) Einheit für die *Komplexität* wird die Anzahl der auszuführenden Befehle[24] bzw. die Anzahl der erforderlichen Gatterfunktionen[25] verwendet, die zur Generierung der Reaktion auf ein Ereignis nötigt ist. Die Einheit für die *Reaktionszeit* ist Sekunden. Beide Achsen haben eine logarithmische Skalierung.

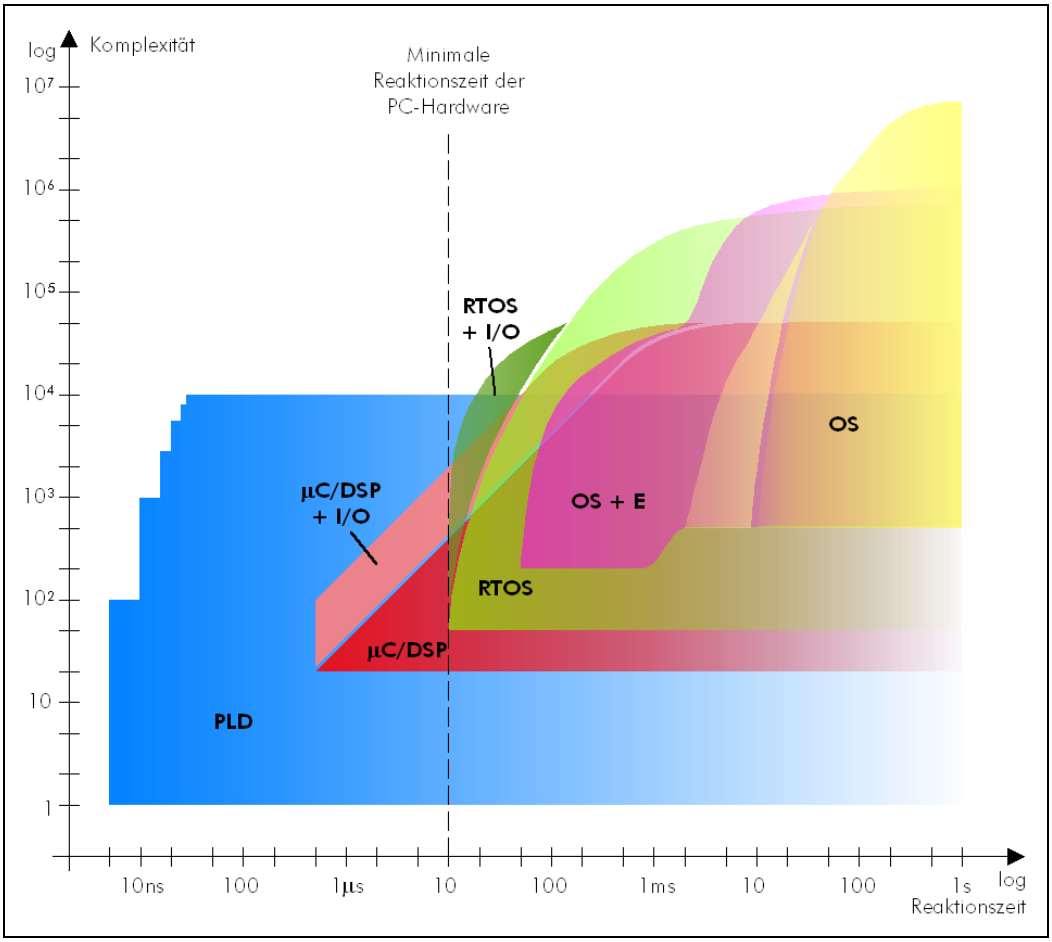


Bild 3.4 Klassifizierung der Systemarchitekturen

[24] „Befehl" bezeichnet hier eine x86-Assembleranweisung.
[25] „Gatterfunktion" bezeichnet hier einen *primitiven* Funktionsblock aus der VHDL-Bibliothek des verwendeten PLDs.

Tabelle 3.3 zeigt eine Auflistung der minimal erreichbaren Reaktionszeiten der verschiedenen Systemarchitekturen. Diese sind bei komplexen Anwendungen stark von den jeweils eingesetzten Algorithmen, den Rechenleistungen der Prozessoren und der Eignung der Komponenten abhängig. Speziell die Entlastung durch die Ein/Ausgabe-Komponente wird durch die Güte der Anpassung an die Vorverarbeitungsaufgabe bestimmt. Eine genaue Beschreibung der Systeme und der Reaktionszeiten findet sich in den jeweiligen Unterkapiteln der Hauptkapitel 4 (Softwarebasierte Systemarchitekturen) und 5 (Hardwarebasierte Systemarchitekturen).

PC + ...	**System**	**Minimale Reaktionszeit bei einfachen Anwendungen[26]**	**Minimale Reaktionszeit bei komplexen Anwendungen[27]**	**Bemerkungen**
OS	Standard-Betriebssystem (Windows, Linux, u.ä.)	20 ms (2 ms mit großem Jitter)	20 ms (2 ms mit großem Jitter)	Jitter hängt stark von der Systemlast und der eingesetzten Hardware ab
OS + E	Standard-Betriebssystem mit Realzeiterweiterung	50 – 100 μs	0,2 – 1 ms	Je nach Methode der Realzeiterweiterung
RTOS	Realzeitbetriebssystem	10 μs	50 – 200 μs	Je nach Betriebssystem-struktur (Kernel oder RTOS)
RTOS + I/O	Realzeitbetriebssystem mit festprogrammierter Ein/Ausgabe-Komponente	10 μs	10 – 100 μs	Abhängig von der Leistungsfähigkeit der Ein/Ausgabe-Komponente
μC/DSP	Mikrocontroller- oder DSP-Komponente	0,5 μs	50 – 500 μs	Stark abhängig von der Rechenleistung des eingesetzten μC/DSP
μC/DSP + I/O	Mikrocontroller- oder DSP-Komponente mit festprogrammierter Ein/Ausgabe-Komponente	0,5 μs	5 – 100 μs	Stark abhängig von der Rechenleistung des eingesetzten μC/DSP und der Leistungsfähigkeit der Ein/Ausgabe-Komponente
PLD	Programmierbare Logik-Komponente (FPGA, CPLD, o.ä.)	5 ns	20 – 100 ns	Je nach Parallelisierbarkeit der Anwendung

Tabelle 3.3 Minimale Reaktionszeiten in Abhängigkeit von der Komplexität der Anwendung

3.3.2 Zusammenhang von Reaktionszeit und Kosten

Aus der Klassifizierungsgrafik wird ersichtlich, dass alle Methoden einem Grundsatz folgen: Die maximal verarbeitbare Komplexität sinkt mit abnehmender Reaktionszeit (= höhere Anforderung). In zeitunkritischen Bereichen gibt es große Überschneidungen, die eine Wahlmöglichkeit zwischen verschiedenen Methoden eröffnen. Jedoch dürfen hierbei die unterschiedlichen Kosten der entsprechenden Realisierungsmöglichkeiten nicht außer acht gelassen werden. Eine entsprechende Grafik zeigt Bild 3.5.

[26] „Einfach" heißt hier: Als Komplexität der Anwendung wird die entsprechende Untergrenze der jeweiligen Systemarchitektur im Bild 3.4 gewählt.
[27] Als Größe für die Komplexität der Anwendung wird das obere Drittel der jeweiligen Systemarchitektur im Bild 3.4 gewählt.

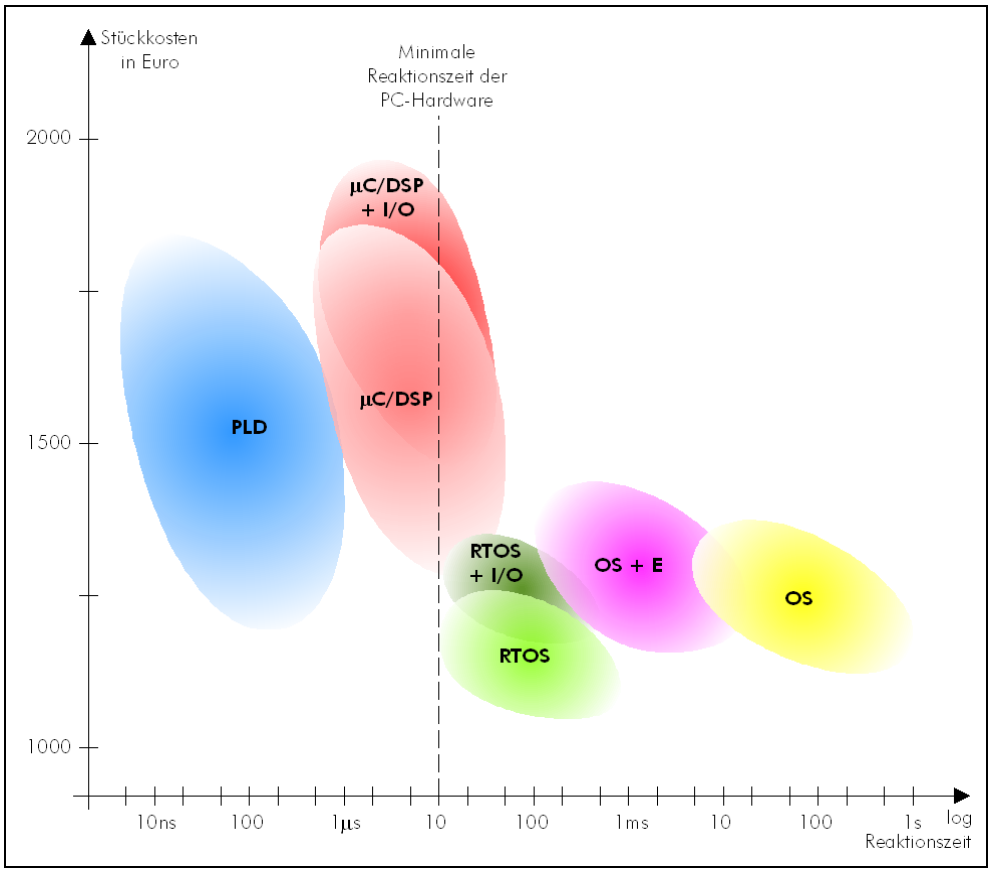


Bild 3.5 Zusammenhang von Reaktionszeit und Kosten der verschiedenen Systemarchitekturen

Die Stückkosten (y-Achse) gelten für geringe bis mittlere Stückzahlen und setzen sich aus den anteiligen Kosten für Softwarelizenzen, PC-Hardware und Zusatzhardware zusammen. *Non-Recurring-Engineering*-Kosten (NRE), wie z.B. Entwicklungssoftware oder Personalkosten, sind nicht mit berücksichtigt. Gerade bei Erweiterungssoftware oder -hardware entstehen jedoch hohe zusätzliche Kosten, welche in einer Gesamtkalkulation berücksichtigt werden müssen. Betrachtet man die Stückkosten im Verhältnis zu den zeitlichen Anforderungen, so zeigt sich, dass die Kosten umso höher sind, je niedriger die minimale Reaktionszeit ist. Die Grenze der minimal erreichbaren Reaktionszeit ist für softwarebasierte Systemarchitekturen scharf abgegrenzt. Kleinere Reaktionszeiten lassen sich nur durch Zusatzhardware erreichen. Wenn es möglich ist, komplexe Funktionen und Berechnungsalgorithmen in zusätzliche, fest programmierte Ein/Ausgabe-Komponenten auszulagern (+ I/O), kann eine höhere Aufgabenkomplexität verarbeitet werden (siehe Bild 3.4). Gleichzeitig steigen jedoch die Stückkosten um die zusätzlichen Kosten der Ein/Ausgabe-Komponente an.

4 Softwarebasierte Systemarchitekturen

In diesem Kapitel soll die Kombination der Standard-PC-Hardware mit unterschiedlicher Software zur Steuerung von zeitkritischen Systemen näher untersucht werden. Zuerst wird auf unterschiedliche Realisierungsmethoden eingegangen. Kapitel 4.1 behandelt die Standard-Softwareimplementierungen, Kapitel 4.2 den Einsatz von Realzeiterweiterungen für Standardbetriebssysteme und Kapitel 4.3 die Verwendung von Realzeitbetriebssystemen. Die Sonderform der Mehrprozessorsysteme beschreibt Kapitel 4.4. Hardwarebasierte Lösungen werden im anschließenden Hauptkapitel 5 beschrieben. Als erstes sollen die Anforderungen an moderne Realzeitsysteme für zeitkritische technische Prozesse dargestellt werden.

Anforderungen an moderne Realzeitsysteme

Verschiedene Aspekte, wie flexible Anforderungen nach Bandbreite, Latenzzeiten, Jitter und Zuverlässigkeit, oder die Integration in vernetzte Rechnerverbünde müssen von modernen Realzeitsystemen erfüllt werden. Wechselnde Einsatzbedingungen während der Lebensdauer erfordern ein flexibles Ressourcenmanagement. Für den Anwender gewinnt die grafische Datenaufbereitung zunehmend an Bedeutung. Im dynamischen Markt der Realzeitsysteme sind Produktivität, Qualität und Kostenvorteile ausschlaggebend.

Hier eine Zusammenstellung der wichtigsten Forderungen für aktuelle, konkurrenzfähige Realzeitsysteme und deren Implementierung [GKL+99]:

- Unterschiedlichste Arten von Daten aus verschiedenen Quellen, wie Sensordaten, Kommandoanweisungen, Anwendereingaben oder Bilddaten müssen verarbeitet werden, ohne die Realzeiteigenschaften zu beeinträchtigen.
- Gemeinsame Ressourcen müssen effektiv genutzt und mit zeitunkritischen Prozessen geteilt werden.
- Realzeitkritische Anwendungen müssen vor zeitunkritischen Anwendungen geschützt werden.
- Geeignete moderne Softwareentwicklungsmethoden sind einzusetzen um Kosten und Zeitbedarf zu verringern und Wiederverwendung und schnelle Fehlersuche zu ermöglichen.
- Eine dynamische Anpassung an wechselnde Systemanforderungen zur Laufzeit unter Beibehaltung aller Realzeitfähigkeiten ist erforderlich.
- Die Verfügbarkeit kann durch zuverlässige Fehlerbehandlung und Redundanzkonzepte erhöht werden.

4.1 Standard-Softwareimplementierungen

Die erste Teilgruppe der softwarebasierten Systemarchitekturen bilden die Standard-Softwareimplementierungen, welche unveränderte Standard-PC-Hard- und –Software verwenden. Als Betriebssystem kann ein beliebiges Standard-(Büro-)Betriebssystem eingesetzt werden. Aufgrund des hohen

Marktanteils[28] und der geringen Unterschiede zwischen den verschiedenen Betriebssystemen soll hier stellvertretend das Microsoft-Betriebssystem *Windows NT* betrachtet werden. Bild 4.1 zeigt den Parameterbereich, welcher durch die Verwendung von Standard-Softwareimplementierungen abgedeckt wird. Die Methode wird in Teilkapitel 4.1.1, ihre Grenzen in Teilkapitel 4.1.2 beschrieben.

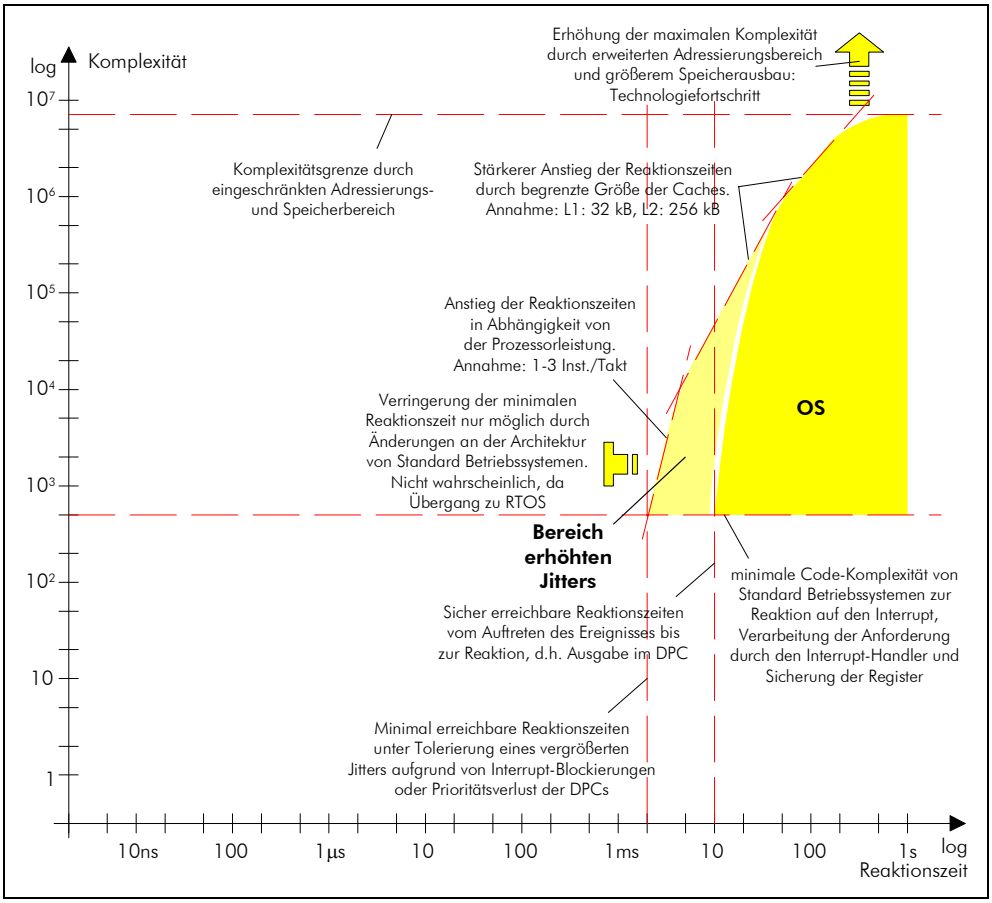


Bild 4.1 Parameterbereich der Standard-Softwareimplementierungen (OS)

4.1.1 Windows NT

Das Standardbetriebssystem Windows NT der Firma Microsoft wird eingesetzt, um günstig und schnell ein Steuerungssystem auf Basis eines Standard-PCs in Verbindung mit einem Standardbetriebssystem zu entwickeln. Hierzu benötigt man außer der zu steuernden Hardware eine Entwicklungssoftware, z.B. das Tool *Visual C++* von Microsoft. Zur Anbindung des zu steuernden technischen Prozesses verwendet man die Standardschnittstellen des PCs (z.B. seriell, parallel, USB) oder zusätzliche I/O-Hardware in Form von Steckkarten (z.B. für den PCI-Bus). Die Idee besteht darin, die Steuerung soweit wie möglich als Anwenderprogramm zu programmieren. Lediglich die Zugriffe auf die Hardware müssen mit einem (Kernel-)Treiber realisiert werden.

Die Kommunikation mit dem Steuerungssystem wird in der von Microsoft empfohlenen Vorgehensweise zur Treiberprogrammierung realisiert und verwendet vorzugsweise das Interrupt-Verfahren. Pollingverfahren existieren zwar, sind aber wegen der ereignisbasierten Signalcharakteristik zeitkritischer Prozesse ungeeignet. Nach dem Aufruf einer Interrupt-Service-Routine (ISR) im Treiber wird

[28] Marktanteil von Windows-Betriebssystemen auf „Bürorechnern“ (keine Server): > 80 %.

ein *Deferred Procedure Call* (DPC) zur eigentlichen Datenübertragung angelegt. Ist der DPC abgearbeitet, kehrt die Ausführung zum Anwenderprogramm zurück. Außer mit dem synchronen Verfahren, bei der die weitere Ausführung bis zur Beendigung der Aufgabe unterbrochen wird, kann eine Kommunikation auch asynchron und damit nicht-blockierend ausgeführt werden. Erst an einem späteren Synchronisationspunkt wird das Ergebnis abgefragt. Ausführliche Informationen hierzu finden sich bei [Bak97] und [VM99].

In dieser Weise kann z.B. eine Robotersteuerung unter Windows NT realisiert werden [MS98]. Die experimentellen Untersuchungen belegen die Funktionsweise dieser reinen Software-Implementierung für schwach belastete Systeme. Die Latenzzeiten vom Auftreten des externen Interrupts bis zur Rückkehr zum Anwenderprogramm liegen bei 150 μs. Die Abweichungen (Jitter) liegen im Bereich von 50 μs. Bei einer gewählten Abtastrate von 1 ms stehen dem Anwenderprogramm und anderen Prozessen die restliche Zeit von ca. 800 μs zur Verfügung. Beim Start von weiteren rechenintensiven Anwendungen erhöhen sich die Latenzzeiten sehr stark. Bei Festplattenzugriffen wurden Reaktionszeiten von bis zu 10 ms gemessen (Hintergründe im Teilkapitel 4.1.2).

Auch von [HB00] wurde die Steuerung eines Systems nur mit Standardkomponenten realisiert. Ein ferromagnetischer Körper wird durch Regelung des Stromflusses einer darüber montierten Spule in der Schwebe gehalten (Bild 4.2). Die Sensorwerte und die Signale für den Stromverstärker werden über eine kombinierte A/D-D/A-Steckkarte eingelesen bzw. ausgegeben. Der Regelalgorithmus ist ein digitaler PID-Regler, der auf dem PC-Prozessor berechnet wird. Bei einer gewählten Abtastperiode von 2 ms kann der Körper ruhig in Schwebe gehalten werden. Bei Messungen ergaben sich einzelne Ausreißer von bis zu 8 ms. Die Messzeit betrug 2 Stunden. Trotzdem verhielt sich das System aufgrund der toleranten Eigenschaften des zugrundeliegenden technischen Prozesses stabil.

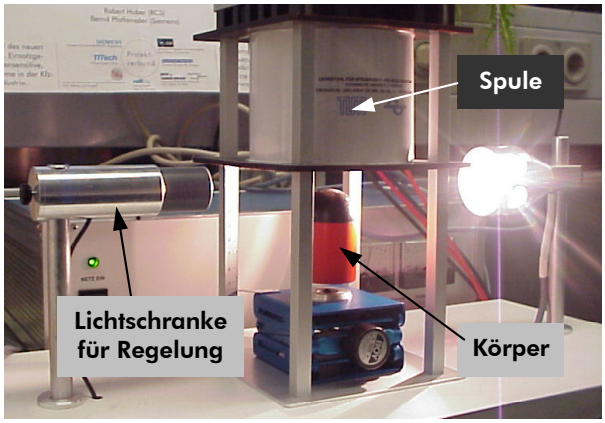


Bild 4.2 Regelung eines schwebenden Körpers

4.1.2 Analyse der Realzeitfähigkeit von Windows NT

Die im vorigen Kapitel in den Messungen beobachteten Ausreißer können zu einer Verletzung der Realzeitbedingungen führen. Deshalb sollen nun die Hintergründe dieser Verzögerungen näher untersucht werden, um genauere Aussagen über die softwaretechnische Eignung von Windows NT für Realzeitanwendungen geben zu können. Nachfolgend eine Checkliste der Anforderungen:

- Einhaltung von Deadlines
- Prioritäten für zeitkritische Prozesse

- Interrupt-Fähigkeit für asynchrone Ereignisse
- Vorschriftsmäßige Implementierung von Systemfunktionen

Nun sollen diese Anforderungen detailliert für das Betriebssystem Windows NT überprüft werden.

Einhaltung von Deadlines

Das Betriebssystem Windows NT verfügt über preemptives Multitasking und erfüllt somit die Forderung *Einhaltung von Deadlines* für Realzeitsysteme.

Vergabe von Prioritäten

Windows NT ist ein Multitasking-Betriebssystem mit 32 Prioritätsstufen, die in zwei Klassen unterteilt sind (Tabelle 4.1). Die Prioritäten 0 – 15 für Anwendungs-Tasks bilden die *Dynamische Klasse*, die Prioritäten 16 – 31 für realzeitkritische Tasks bilden die *Realzeit-Klasse* [Hub97]. Tasks der Realzeit-Klasse haben eine feste Priorität, die nur durch die Anwendung selbst verändert werden kann. Für dynamische Tasks dagegen wird die Priorität vom Scheduler an den Prozess angepasst. Nur die Realzeit-Klasse ist in einem Realzeitsystem sinnvoll, wenn garantierte Antwortzeiten sicherzustellen sind. Tasks werden unter Windows NT auch Prozesse genannt. Ein Prozess setzt sich wiederum aus mehreren Threads zusammen. Jeder Thread kann eine Priorität im Intervall von [-2;2] der Grundpriorität oder die zwei extremen Zustände seiner Klasse annehmen.

Priorität	Verwendung	Klasse
31	time-critical	Realzeit
26	normal	
24 … 22	system level	
16	idle	
15	time-critical	Dynamisch
13 ± 2	high foreground	
9 ± 2	normal foreground	
7 ± 2	normal background	
4 ± 2	low fore-/background	
1	idle	
0	idle thread	

Tabelle 4.1 Prioritätsspektrum von Windows NT

Ein Beispiel: Die Threads eines Prozesses mit der Priorität 24 können jede Priorität zwischen 22 und 26 plus die Prioritäten 16 und 31 (Realzeit-Klasse) annehmen. Reichen diese 7 Prioritätsstufen nicht aus, muss man die Threads auf mehrere Prozesse verteilen. Wird zwischen Threads unterschiedlicher Prozesse umgeschaltet, so verlängern sich die Umschaltzeiten aufgrund des nun notwendigen Kontextwechsels erheblich. Haben zwei Threads die gleiche Priorität, führt NT eine ablauffähige Task für eine gewisse Zeitspanne (*time slice* oder *quantum*) aus und wechselt dann zur nächsten Task. Diese Zeitspanne beträgt bei Windows NT Workstation 20 – 90 ms, bei Windows NT Server 120 – 180 ms.

Problematik der Prioritätsinversion

Die Prioritätsinversion (*priority inversion*) ist ein klassisches Problem von Realzeitsystemen. Systemaufrufe zur Synchronisation, wie Mutexe, Semaphoren und kritische Abschnitte (*critical sections*), sollten in der Lage sein, Prioritäten zu vererben (*priority inheritance*). Windows NT stellt für Tasks der Realzeit-Klasse keine Maßnahmen zur Verhinderung der Prioritätsumkehr, wie z.B. die Prioritätsvererbung, bereit. Lediglich für die dynamische Klasse existiert ein sogenannter *Boost*-Mechanismus. Die Prioritäten „lange" wartender Tasks werden angehoben, um ihnen wieder Rechenzeit zu gewähren. Eine präzise Aussage über die Wartezeitlänge wird von Microsoft nicht angegeben. Erfahrungsberichte sprechen von einigen Sekunden.

Eine weitere Möglichkeit der Prioritätsumkehr tritt in Zusammenhang mit der synchronen Implementierung einiger Systemaufrufe auf. Synchron heißt, dass der aufrufende Thread bis zur Erledigung des aufgerufenen Systemaufrufs unterbrochen wird. Insbesondere Aufrufe der grafischen Benutzerschnittstelle (GUI) des Windows Systems werden mit Prioritäten der dynamischen Klasse ausgeführt und können somit höherpriore Threads der Realzeit-Klasse blockieren [TM97].

Interrupt/Deferred Procedure Call-Architektur von Windows NT

In der Treiber-Programmierung unter Windows NT erfolgt die Bearbeitung von Interrupts nach einem zweistufigen Konzept [VM99][Bak97]. Um möglichst viele Interrupts in kurzer Zeit annehmen zu können und damit die Gesamtleistung des Systems zu maximieren, wird die Bearbeitung der Unterbrechung in eine Interrupt-Service-Routine (ISR) und einen *Deferred Procedure Call* (DPC) aufgeteilt. Eine sehr kurze ISR bestätigt lediglich den Interrupt, erzeugt den DPC (in der die Hauptarbeit geleistet wird) und kehrt umgehend zum normalen Programmablauf zurück. Die DPCs werden vom System aufgerufen, wenn der *Interrupt Request Level* `DISPATCH_LEVEL` erreicht (siehe auch Tabelle 4.2). Dies ist direkt im Anschluss an die ISR noch vor dem Aufruf normaler Tasks der Fall. Ein Problem entsteht nun, wenn Interrupts niedrigerer Priorität auftreten und damit die Bearbeitung der DPCs blockieren. Alle DPCs werden in einer gemeinsamen DPC-Warteschlange abgelegt und in der Reihenfolge ihres Eintrags abgearbeitet, wobei die ursprüngliche Priorisierung der Interrupts verloren geht. Die Bearbeitungsreihenfolge in der DPC-Warteschlange kann man nur in geringem Maße ändern. Es existiert lediglich eine Dreiteilung in `LowImportance` (hinten anhängen), `MediumImportance` (hinten anhängen + `DISPATCH_LEVEL` Interrupt auslösen) und `HighImportance` (vorne einreihen + `DISPATCH_LEVEL` Interrupt auslösen). Bei geringer Systemlast liegen die Latenzzeiten vom Auftreten des Interrupts bis zur Beendigung des DPC bei 150 – 200 μs. Hohe Systemlasten, z.B. bedingt durch häufige Interrupts von gleichzeitig auftretenden Festplattenzugriffen, können bei der Abarbeitung hohe Latenzzeiten von bis zu 10 ms erzeugen.

Semaphoren und Mutexe werden für die Synchronisation mehrerer Tasks während ihrer quasiparallelen Bearbeitung verwendet. Diese Synchronisationsobjekte werden aber genau wie die DPCs in zentralen Warteschlangen verwaltet und verlieren dadurch ihre Prioritätsinformation. So ergaben Messungen der Ausführungszeit des *mutex*-Sytemaufrufs durchschnittlich 35 μs, durch Prioritätsverlust traten aber auch maximale Zeiten von bis zu 670 μs auf [TM97].

Ein Mehrprozessorsystem lässt besseres Realzeitverhalten vermuten. In einem Zwei-Prozessor-System kann festgelegt werden, auf welchem Prozessor welcher Interrupt bearbeitet werden soll. Damit lassen sich mehrere ISRs unterschiedlicher Priorität parallel auf verschiedenen Prozessoren ausfüh-

ren. Es existiert, entgegen widersprüchlicher Aussagen in der Literatur [MS98], eine separate DPC-Warteschlange für jeden Prozessor. Auch eine spezifische Zuordnung von DPCs auf Prozessoren ist möglich, wird aber nur äußerst selten vorgenommen. Da nun mehr als eine DPC-Routine gleichzeitig ausgeführt werden kann – bei Zuweisung auf einen anderen Prozessor sogar schon während der Laufzeit der ISR-Routine – müssen sämtliche Zugriffe auf gemeinsame Daten über *spinlocks* synchronisiert werden. Die Komplexität und Fehleranfälligkeit steigt stark an und die Latenzzeiten ändern sich in Abhängigkeit der Programmierung und der Prozessorzuteilung. Die Berechnung der zu erwartenden Latenzzeiten für Multiprozessorsysteme ist schwierig, da der Scheduling-Algorithmus von Windows NT für die parallele Abarbeitung von DPCs auf verschiedenen Prozessoren von Microsoft nicht dokumentiert ist.

Andererseits ist das zweistufige Interrupt-Konzept nur ein *Vorschlag* von Microsoft, welcher bei der Treiberprogrammierung nicht zwingend angewandt werden muss. So können z.B. realzeitkritische Berechnungen vollständig in die ISR verlagert werden um schnellste Reaktionszeiten zu erreichen (siehe auch Kapitel 4.2.1, LP-Elektronik). Es sollte aber, je nach Anwendung, eine Balance zwischen Verwendung zeitaufwändiger ISRs und befriedigender Systemperformance gefunden werden.

Hardware-Interrupts

Alle auftretenden Hardware-Interrupts werden von Windows NT mit höherer Priorität als die der vorhandenen Anwender-Tasks behandelt. Somit können auch niederpriore Ereignisse von unkritischen Quellen, wie z.B. Mausbewegungen, wichtige Anwender-Tasks verzögern. Nur in der Kernel-Ebene ist es möglich, Interrupts niedriger Priorität zu maskieren. In einem Realzeitbetriebssystem dagegen muss der Programmierer die Kontrolle über die Prioritäten aller Tasks und Interrupts haben, um die gewünschte Bearbeitungsreihenfolge implementieren zu können. Die Übersetzung der Hardware-Interrupts (siehe auch Kapitel 5) in die systeminterne Interrupt-Ebenenstruktur von Windows zeigt Tabelle 4.2.

Quelle	Interrupt-Ebene	IRQL	Interrupt-Definition
Hardware	31	HIGH_LEVEL	Hardware error (machine check or bus error)
	30	POWER_LEVEL	Power failure
	29	IPI_LEVEL	Work request from another processor
	28	CLOCK_LEVEL	Interval clock
	27 – 12	DIRQL	Hardware interrupts 0 – 15
	11 – 4		Not used
Software	3		Software debugger
	2	DISPATCH_LEVEL	Thread dispatching / DPC
	1	APC_LEVEL	Asynchronous procedure call
	0	PASSIVE_LEVEL	Normal thread execution

Tabelle 4.2 Interrupt-Ebenen von Windows NT

Systemfunktionen für die Realzeitfähigkeit

Damit ein Betriebssystem zur Ausführung realzeitkritischer Anwendungen verwendet werden kann, muss es spezifische Systemfunktionen vorschriftsmäßig implementieren [Rea98]. In Windows NT entsprechen einige wichtige Funktionen nicht dieser Regel. So löscht z.B. die Funktion `Termi-`

`nateThread()` des Task-Managements weder den vom Thread benutzten Stack-Speicher, noch überprüft sie die Freigabe reservierter Ressourcen und Synchronisationsobjekte. Ebenso fehlen Funktionen zur Änderung der Zeitscheibenlänge des Schedulers bei Ausführung von Threads gleicher Priorität. Auch Mailboxen und binäre Semaphoren zur Nachrichtenübermittlung fehlen. Dagegen sind Funktionen zum Sperren von Speicher im RAM (*page locking*) vorhanden, um zeitintensives Auslagern von Speicherbereichen (*swapping*) zu vermeiden. Allerdings wurde von [Ric95] ein Fall beschrieben, bei dem Windows NT einen lange Zeit inaktiven Prozess selbständig ausgelagert hat, obwohl dieser gesperrt war.

Optimierungsmethoden unter Windows NT

Verwendet man die systemeigenen Treiber von Windows NT/2000, muss man zur Erzielung von geringen Verzögerungen und größtmöglicher Geschwindigkeit optimale Parametersätze benutzen. Bei Festplattenzugriffen untersuchte [RIG98] das Verhalten der Treiber unter Windows NT, [CGW+00] das unter Windows 2000. Hierbei stellte sich heraus, dass je nach verwendeter Optimierungsmethode (Tabelle 4.3) die Verzögerungszeiten und die Leistung verbessert werden können.

Optimierungsmethode:	**Begründung und Leistungszuwachs:**
Verwendung von möglichst großen Datenblöcken (> 4 KB)	Bei kleinen Blöcke erhöht der *Overhead* die Prozessorlast unverhältnismäßig (2 KB = 30 %, 64 KB = 3 %). Der Dateisystemcache von Windows arbeitet mit 64 KB-Blöcken, was auch die optimale Größe für Dateizugriffe ist. (Faktor 2)
Nutzung des DMA-Modes statt des PIO-Modes	Bei der PIO-Übertragung muss der Prozessor die gesamte Übertragung abwickeln (Polling). Im DMA-Mode macht dies der Festplattencontroller im Busmasterbetrieb. (Faktor ca. 85!)
Vorab-Anforderung des wahrscheinlich zu belegenden Speicherplatzes auf der Festplatte, wenn eine Datei angelegt wird und sequentielles Schreiben der Daten.	Der von Windows NT/2000 erfüllte Sicherheitslevel (Level C2) erfordert beim Lesen einer noch nicht geschriebenen Datei die Rückgabe des Wertes 0. Bei nicht-sequentiellem Zugriff auf eine neue Datei beschreibt Windows den jeweiligen Speicherbereich zuerst mit Nullen, bevor die Daten geschrieben werden. (Faktor 2)
Aktivieren des Write-Cache der Festplatte	Der Cache des Festplattencontrollers ist schneller als der physikalische Schreibvorgang. (max. Faktor 9)
Zerlegung eines Zugriffs in 2, 4 oder 8 einzelne Zugriffe	Die Aufteilung in mehrere kleine Zugriffe ermöglicht die Nutzung von zusätzlicher Parallelität im System. (Faktor 2)
Nutzung von mehreren Festplatten (RAID)	Die Leistung steigt bis zur Sättigung des PCI-Busses fast linear an. Bei sequentiellen Zugriffen ist der begrenzende Faktor der Bus, dessen Sättigung bei ca. 3 Festplatten (3 x 40 MB/s) erreicht wird. Die zusätzliche Prozessorlast speziell bei Software-RAID ist vernachlässigbar.
Verwendung neuester Servicepacks	Z.B. hat Windows NT 4.0 SP3 einen Bug im IDE-Treiber bei der Übertragung von Blöcken > 128 KB, welcher in SP6 behoben worden ist. (Faktor 2)
Verwendung von *gemappten* Laufwerken statt der UNC-Pfade	Das Lesen der Daten von *gemappten* Laufwerken über das Netzwerk ist wesentlich schneller als über den UNC-Pfad. Die Ursachen sind nicht bekannt, es ist aber eine Beeinflussung durch die Art der Implementierung des Netzwerk-Stacks zu vermuten. (Faktor 25)

Tabelle 4.3 Optimierungsmethoden bei Festplattenzugriffen

Fazit

Ein Standardbetriebssystem wie Windows NT ist in Verbindung mit einem Standard-PC-System nicht für harte Realzeitaufgaben einsetzbar. Die Softwarestruktur ist für maximalen mittleren Durchsatz optimiert; eine Verwendung ist deshalb nur für zeitunkritische bzw. fehlertolerante Anwendungen sinnvoll. Die minimalen Interrupt-Latenzzeiten (bei Einsatz von DPCs) liegen bei einem unbelasteten System im Bereich von **200 µs**. In realen Anwendungen ist jedoch immer eine Rechenlast vorhanden; ansonsten wäre das System überdimensioniert und könnte durch Einsatz eines langsameren Prozessors kostengünstiger gestaltet werden. Ist Jitter tolerierbar, kann Windows NT bei mittleren Systemlasten für Reaktionszeit-Anforderungen bis zu ca. **2 ms** eingesetzt werden. Hierbei treten, wie oben beschrieben, sporadische Ausreißer von 8 – 10 ms auf. Die sinnvoll realisierbaren Reaktionszeiten liegen deshalb im **10 ms-Bereich**.

Die Reaktionszeit, d.h. die Zeit vom Auftritt des externen Ereignisses bis zur Ausgabe der Reaktion, eines PC-Systems unter Verwendung der Standard-PC-Hardware und einem Standardbetriebssystem setzt sich zusammen aus:

$$t_{R\,OS} = t_{I\,PC} + t_{ISR} + t_{I\,DPC} + t_{DPC} + t_D \geq 2 - 10\ ms$$

mit: R = Reaktionszeit, I = Interrupt-Latenzzeit des PCs, ISR = Ausführungszeit der Interrupt-Service-Routine, I DPC = Latenzzeit bis zur Ausführung des *Deferred Procedure Calls*, DPC = Ausführungszeit des *Deferred Procedure Calls* bis zur Ausgabeanweisung, D = Datenübertragungszeit CPU → I/O.

Generell können technische Prozesse, welche „gutmütig" auf Ausreißer der Abtastrate reagieren, ohne Spezialhard- und Software alleine mit Standard-PCs und Standardbetriebssystemen kontrolliert werden. Eine theoretische und praktische Überprüfung des Systemverhaltens, besonders für den Fehlerfall, muss aber grundsätzlich erfolgen. Technische Prozesse, bei denen ein Fehlverhalten zu außerordentlich hohen Kosten führt (z.B. Flugzeugabsturz), sind hiervon ausgenommen.

Fazit: *Kritische Realzeitprozesse sollten nicht in Form einer reinen Standard-Software-Implementierung realisiert werden, da eine Bestimmung der maximalen Zeitverzögerung sehr schwierig ist. Große Überschreitungen der Abtastzeit durch Blockierungen von anderen Systemprozessen können nicht ausgeschlossen werden. Das Verhalten des Windows-Kernels ist wegen fehlender systeminterner Informationen nicht vorhersagbar.*

4.2 Einsatz von Realzeiterweiterungen

Um die Vorteile des oben beschriebenen, rein softwarebasierten Systems für harte Realzeitaufgaben nutzbar zu machen, müssen Veränderungen vorgenommen werden. Diese sollen aber möglichst klein und auf die Software beschränkt bleiben. In einem üblichen Ein-Prozessor-System müssen Realzeitanwendung und Standardbetriebssystem koexisiteren. Da Standardbetriebssysteme von sich aus nicht realzeitfähig sind, verwendet man sogenannte *Realzeiterweiterungen*, um ein System zur Steuerung realzeitkritischer Prozesse aufzubauen. Die überwiegende Anzahl von Erweiterungen ist auf Windows NT zugeschnitten. Die nachfolgenden Abschnitte sollen eine Übersicht der verschiedenen Methoden zur Erzielung von Realzeitfähigkeit [Kre97] geben.

Bild 4.3 zeigt den Parameterbereich beim Einsatz von Realzeiterweiterungen. Die folgenden Teilkapitel 4.2.1 und 4.2.2 beschreiben verschiedene Methoden von Erweiterungen für die Standardbetriebssysteme Windows NT und Linux sowie für Java und einen zusammenfassenden Vergleich in Teilkapitel 4.2.3. Die Probleme und Grenzen beschreibt Kapitel 4.2.4.

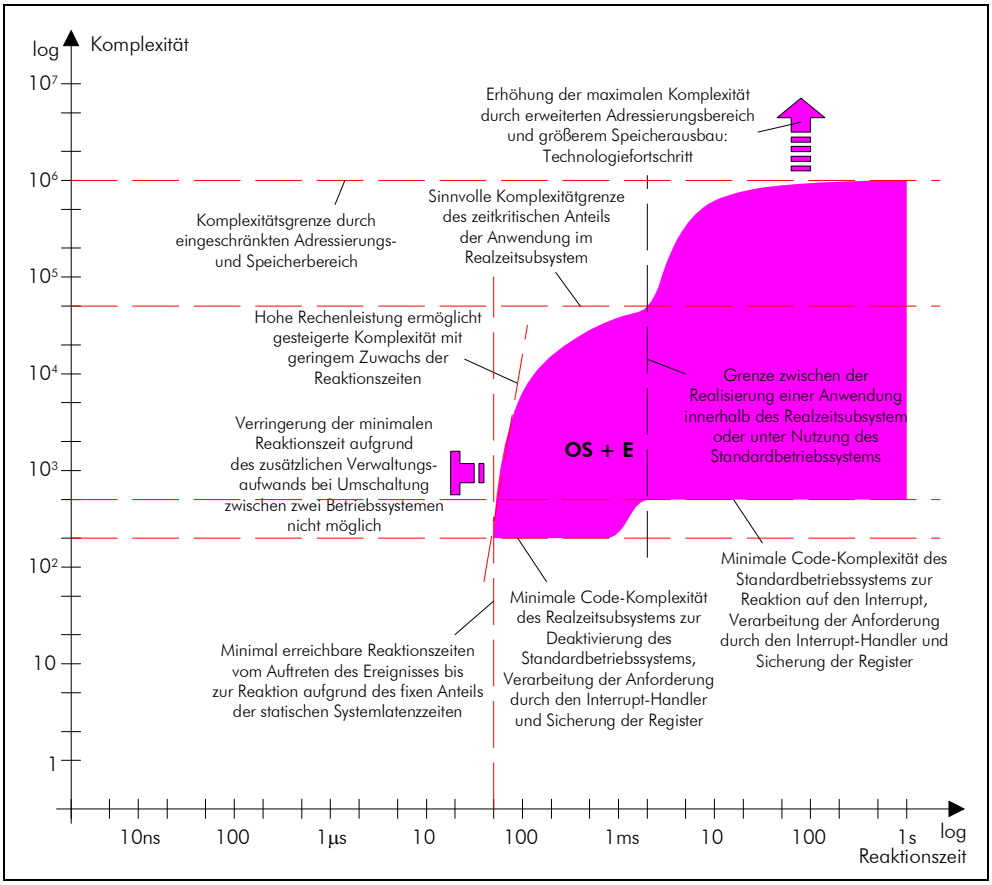


Bild 4.3 Parameterbereich der Realzeiterweiterungen (OS + E)

4.2.1 Erweiterungen für Windows NT

Realzeiterweiterungen für das Standardbetriebssystem Windows NT stellen den Großteil der angebotenen Lösungen dar. Deshalb sollen vorrangig die verschiedenen Methoden dieses Betriebssystems erläutert werden. Es gibt fünf grundlegende Ansätze [RSG+98], deren spezifische Vor- und Nachteile und Leistungsfähigkeit nun in der Reihenfolge ihrer Relevanz erläutert werden:

1) Modifikation des *Hardware Abstraction Layers* (HAL) und zusätzliches Realzeitsubsystem (RTSS)
2) Installation eines Realzeitsubsystems (RTSS) als Treiber im Kernelmode
3) Umleiten des *Non-Maskable Interrupts* (NMI) auf eine Interrupt-Service-Routine (ISR)
4) Bereitstellen einer Win32-API[29] auf einem RTOS
5) Modifikation des NT-Kernels

[29] API: ***A**pplication **P**rogramming **I**nterface*, Software-Funktionen-Bibliothek.

Modifikation des Hardware Abstraction Layers und zusätzliches Realzeitsubsystem

Der erste Ansatz zum Erzielen der Realzeitfähigkeit unter Windows NT beruht auf der Modifikation des *Hardware Abstraction Layers* (HAL). Die Hauptaufgaben des HAL sind die Trennung des Betriebssystems von der Hardware, das Interrupt-Management, der DMA-Betrieb, Verwaltung von Uhren und Zeitgebern und die Bereitstellung einer Schnittstelle zum BIOS. Eine Realzeiterweiterung verändert hier gezielt die Programmierung des Interrupt-Controller-Bausteins[30], ersetzt die globale Abschaltung der Interrupts (`cli/sti`) von Windows durch spezielle Lock/Unlock-Funktionen, stellt zusätzliche präzise Zeitgeber bereit und fängt Systemabstürze (*Blue Screen*) ab. Diese Modifikationen nutzt dann ein zusätzlich implementiertes Realzeitsubsystem (RTSS), welches als Gerätetreiber in das Windows-System eingebunden ist. Es besteht aus einem kleinen Realzeit-Kernel mit realzeitfähigem Scheduler, einer Task-Verwaltung und Kommunikationsfunktionen. Der Windows NT-Kernel selbst wird nicht verändert. Zur Programmierung von Realzeit-Tasks für das RTSS muss eine besondere Programmierschnittstelle benutzt werden.

RTX, VenturCom

Eine Implementierung dieser Methode ist die Realzeiterweiterung *RTX* der Firma VenturCom. Eine weitere Bibliothek (DLL) auf Basis der Win32-Programmierschnittstelle stellt zusätzlich eine „weiche" Realzeitfähigkeit für bereits erstellte unkritische Anwendungen bereit. Das Realzeitsubsystem (RTSS) implementiert preemptives Multitasking mit 128 Prioritätsstufen, erlaubt prioritätserhaltende Synchronisation zwischen Objekten und bietet Warteschlangen (Queues) zur Kommunikation zwischen Subsystem und Windows. Die Tasks werden in der Treiberebene (Ring 0) ausgeführt und verfügen deshalb über keine Speicherschutzmechanismen. Nicht vorschriftsmäßig programmierte Tasks können so die Stabilität das gesamten Systems gefährden. Auch das *Debuggen* muss mit Kerneltreiber-Entwicklungstools durchgeführt werden, welche schwieriger zu bedienen sind und weitaus mehr Fachwissen vom Programmierer erfordern. VenturCom bietet inzwischen eine Implementierung ihres Systems für Doppelprozessorsysteme an, das ähnlich zu dem in Kapitel 4.4 beschriebenen Mehrprozessorsystem mit unterschiedlichen Betriebssystemen ist. Auf dem ersten Prozessor läuft Windows NT, auf dem zweiten die Realzeiterweiterung *RTX*. Das Softwarepaket wird in mehreren Quellen detailliert analysiert [RZN97][FTY98][Süs98]; die Architektur zeigt Bild 4.4.

[30] PIC, siehe auch Kapitel 5.2.

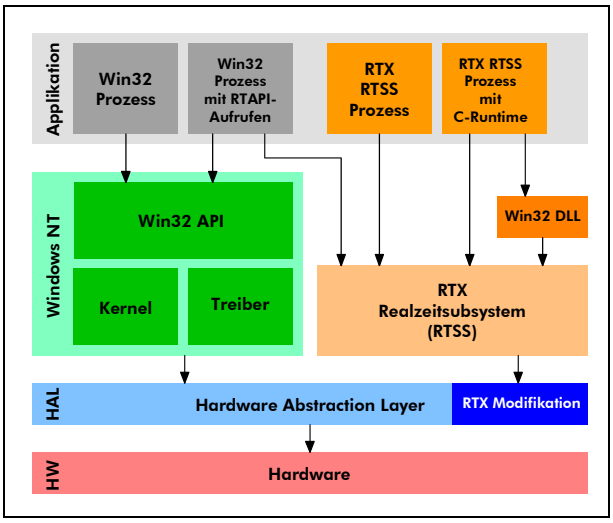


Bild 4.4 Architektur von RTX

INtime, TenAsys

Auch das Produkt *INtime* der Firma TenAsys, eine Tochterfirma von Radisys, beruht auf der Veränderung des HAL und einem zusätzlichen Realzeitsubsystem (RTSS) mit zugehörigem API [Ten00]. Es basiert auf dem bekannten und schon etwas älteren Realzeitbetriebssystem *iRMX* der Firma Intel. Als Kommunikationsmechanismen zwischen dem RTSS und Windows stehen Semaphoren, Mailboxen und gemeinsamer Speicher zur Verfügung, welche über eine zusätzliche API realisiert sind. Die Tasks laufen im Gegensatz zu *RTX* auf User-Ebene und profitieren deshalb von den Windows-internen Speicherschutzmechanismen. Weiterhin besteht die Möglichkeit, Realzeitprozesse unabhängig von einem Absturz des Windows NT-Systems weiterlaufen zu lassen. Diese Fähigkeit ist jedoch in den meisten Fällen unbrauchbar, da Realzeit-Anwendung im Normalfall auch mit dem Anwender oder anderen Systemen kommunizieren, diese Verbindungen aber durch das Windows-Betriebssystem realisiert werden. Die Softwareentwicklung ist durch die Integration der mitgelieferten Programmierwerkzeuge in die Microsoft Visual C++-Umgebung anwenderfreundlich gestaltet. Ein integrierter Debugger für die Realzeitumgebung vereinfacht die Fehlersuche. Task-Debugging kann wegen der Implementierung im User-Mode mit Standardtools erfolgen. Die Struktur zeigt Bild 4.5, weitere Details finden sich in [Fis98][Wol97][Obe99].

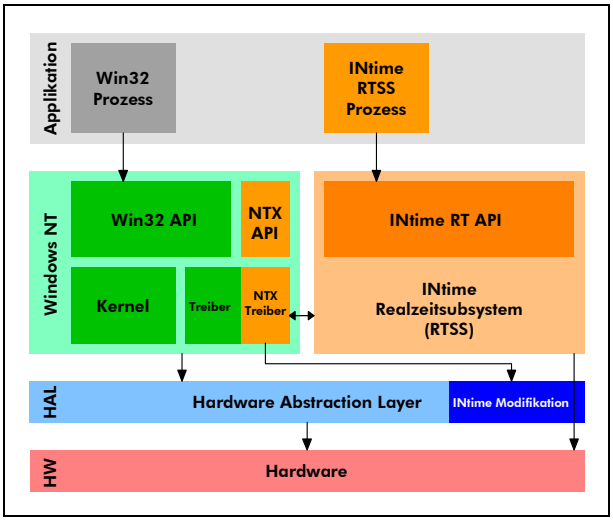


Bild 4.5 Architektur von INtime

HyperKernel, Nematron

Ein leicht abgewandeltes Verfahren benutzt die Software *HyperKernel* der Firma Nematron. Hier werden nur kleine Eingriffe in das Windows NT-System vorgenommen. Die Interrupt-Verwaltung wird durch einen Interrupt-Scheduler erweitert, um in einem Zeitschlitzverfahren abwechselnd Windows NT und das Realzeitsubsystem von *HyperKernel* auszuführen. Die Dauer der Zeitschlitze lässt sich, je nach Art der Anwendung, auf vier verschiedene Zeiten einstellen: 25, 50, 100, 250 μs. Ein Realzeit-Kernel mit preemptivem Scheduler kann bis zu 1024 Realzeit-Tasks mit maximal 32 verschiedenen Prioritäten verarbeiten. Durch Kompatibilität zu Microsofts C-Compiler können die Anwendungen mit diesem Tool programmiert werden. Debugging ist aber nur mit speziellen Kernel-Tools möglich und daher aufwändig. Bild 4.6 gibt einen Einblick in den Aufbau von *HyperKernel*. Zusätzliche Informationen finden sich in [Nem96][TBU98].

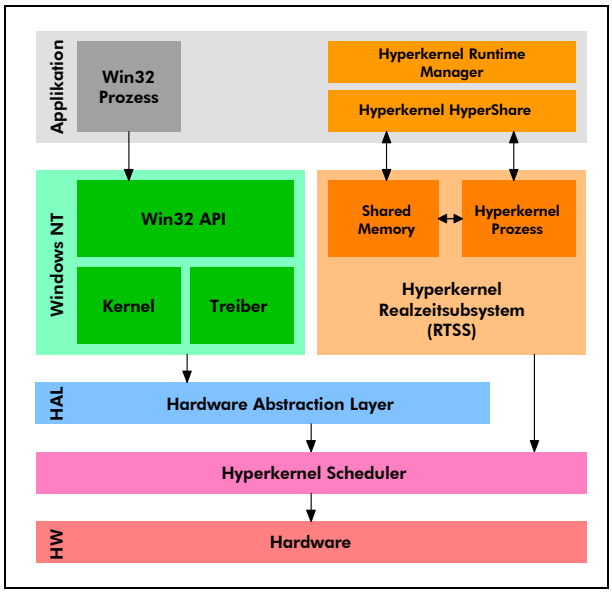


Bild 4.6 Architektur von HyperKernel

Installation eines Realzeitsubsystems als Treiber im Kernelmode

Eine zu Windows konforme Vorgehensweise ist die Kapselung eines Realzeitsubsystems in einem Kernelmode-Treiber (Bild 4.7). Die zu bearbeitenden zeitkritischen Tasks werden parallel zu den Treibern von Windows NT ausgeführt. Hier können höhere Prioritäten als im User-Mode benutzt werden, welche deterministischere Ausführungszeiten ermöglichen. Durch die Koexistenz des Realzeitsubsystems mit betriebssystemeigenen Treibern muss eine sorgfältige Auswahl der Hardware und der zugehörigen Treiber stattfinden. Treibersoftware, welche ungeeignete oder unerlaubte Funktionen des Betriebssystems verwendet, darf nicht installiert werden. Zur Verifikation gibt es Testsoftware, welche die Einhaltung der gewählten Zeitschranken im Betrieb überprüft. Größter Vorteil ist, dass keinerlei Änderungen am Windows NT Betriebssystem vorgenommen werden müssen und damit zukünftigen Updates des Windows-Systems nichts im Wege steht.

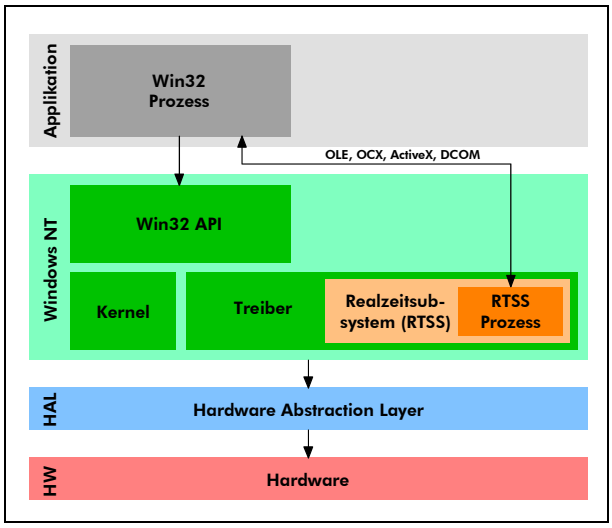


Bild 4.7 Realzeitsubsystem als Treiber im Kernelmode

TwinCAT, Beckhoff

Dieses Verfahren wird in dem System *TwinCAT* der Firma Beckhoff eingesetzt. Hierbei handelt es sich um ein Realzeitsubsystem mit eigenem prioritätsgesteuerten, preemptiven Scheduler, der maximal 64 Realzeit-Tasks mit einer Zykluszeit im 100 µs-Bereich bearbeitet. Hauptanwendung von Beckhoff ist eine Software-SPS (siehe Kapitel 3.1.1) mit verbesserter Realzeitfähigkeit. Es werden bis zu vier IEC 1131-3 konforme SPS-Systeme pro PC unterstützt. Die Anbindung an gängige Feldbusse ist über Steckkarten vorgesehen. Durch die Verwendung des Windows NT Betriebssystems ist eine Datenanbindung mittels der vorhandenen Standards OLE, OCX, ActiveX[31] oder DCOM[32] leicht zu bewerkstelligen. Bild 4.8 zeigt einen Vergleich der Rechenzeiten für 1000 SPS-Befehle zwischen Standard-Hardware-SPSen und der Beckhoff Software-SPS auf verschiedenen PC-Prozessoren.

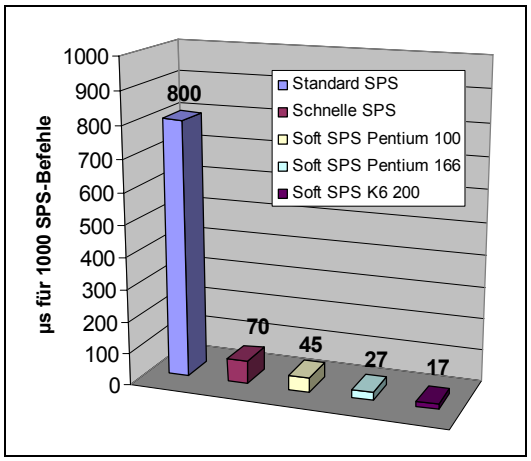


Bild 4.8 Vergleich Hardware-SPS ⇔ Software-SPS mit Realzeiterweiterung

TCX, Bsquare

Auch das Produkt *TCX* der Firma Bsquare erlaubt die Ausführung realzeitkritischer Tasks durch ein Realzeitsubsystem in einem Kernelmode-Treiber. Hier muss der Programmierer die Entwicklung der

[31] ActiveX: von Microsoft, baut auf der OLE-Technologie auf und erlaubt die Einbettung beliebiger Objekte in fremde Dokumente.
[32] DCOM: ***D****istributed* ***C****omponent* ***O****bject* ***M****odel*, eine Technik von Microsoft, um verteilte Anwendungen in einem Netzwerk zu integrieren.

Tasks mit speziellen und wenig anwenderfreundlichen Treibertools durchführen. Die Fehlersuche gestaltet sich entsprechend aufwändig.

Umleiten des Non-Maskable Interrupts auf eine Interrupt-Service-Routine

Eines der Probleme beim Realzeitverhalten des Windows NT Betriebssystems ist die nichtdeterministische Bearbeitung der Interrupts und ihrer DPCs. Wenn man sicherstellt, dass die zu einem Interrupt gehörende Service-Routine direkt nach Auftreten des Ereignisses aufgerufen wird, kann man vorgegebene maximale Reaktionszeiten einhalten. Anstelle der Modifikation des HAL kann man auch den *Non-Maskable Interrupt* (NMI)-Eingang des PC-Prozessors zweckentfremden. Ein Hardware-Interrupt wird auf den NMI umgeleitet und ruft so die Realzeit-Routine auf.

RTWin Toolkit, LP-Elektronik

Die deutsche Firma LP-Elektronik nutzt mit ihrem *RTWin Toolkit* und einer Zusatzkarte für den ISA-Bus den *Non-Maskable Interrupt* (NMI) [Mun97][Mun97/2]. Ein programmierbarer Hardware-Timer (Auflösung 6,7 μs – 27,5 ms) auf der Zusatzkarte löst zyklisch einen NMI aus. Auch eine Umleitung von externen Interrupt-Quellen auf den NMI ist möglich. Der realzeitkritische Code wird in die Interrupt-Service-Routine (ISR) des NMI eingefügt. Eine Sperrung des NMI ist wegen seiner hohen Priorität und des direkten Anschlusses an die PC-Hardware nicht möglich. Die Abarbeitung von Anwendungen direkt in der ISR ist nicht konform zu den Entwicklungsrichtlinien von Microsoft, kann aber für kleine Programmroutinen, etwa einem einfachen Regelalgorithmus, toleriert werden. Die Erstellung des Realzeitcodes erfolgt mit den Microsoft Developer Studio Tools. Eine Kommunikation des Realzeitcodes in der ISR mit Windows-Applikationen kann nur über gemeinsamen Speicher (*shared memory*) erfolgen. Da die ISR im Kernelmode ausgeführt wird, gibt es keinen Speicherschutz. Programmfehler führen zum Absturz des gesamten Systems. Die Softwarestruktur des *Toolkits* zeigt Bild 4.9.

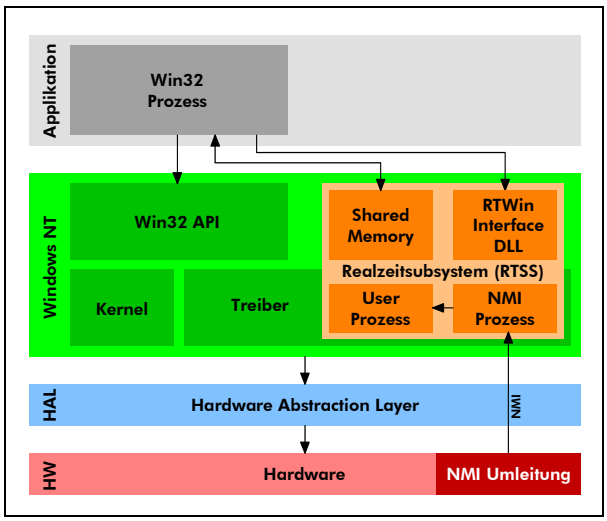


Bild 4.9 Struktur des RTWin Toolkits

Bereitstellen einer Win32-API auf einem RTOS

Ein völlig anderer Ansatz zum Erzielen der Realzeitfähigkeit für Windows NT Anwendungen ist die Emulierung der Win32-API durch ein Realzeitbetriebssystem [Hil97][TM97/2]. Applikationen kön-

nen mit Standardwerkzeugen auf Windows NT Systemen entwickelt werden und sind dann mit der Erweiterung ablauffähig. Von den ca. 3500 Win32-Funktionen[33], die in einem Windows NT System vorhanden sind, werden jedoch nur einige wichtige Funktionen emuliert. Der Verzicht auf die Funktionen der grafischen Oberfläche ist hierbei der größte Nachteil; eine komfortable Anwenderschnittstelle steht nicht zur Verfügung. Auch die funktionelle Treiberschnittstelle von Windows NT kann nicht verwendet werden, d.h. für sämtliche Systemhardware müssen eigene Treiber vorhanden sein bzw. entwickelt werden [Wol97]. Ein Wechsel auf zukünftige Windows-Versionen ist abhängig von Änderungen der API und daher problematisch.

Embedded ToolSuite (ETS), VenturCom

Die *Embedded ToolSuite* (ETS) der Firma VenturCom implementiert die Methode der Win32-API-Realzeiterweiterung mit untergelagertem Realzeitbetriebssystem. Zur Entwicklung kann hier die Microsoft Visual Studio Software verwendet werden. Aufgrund der eingeschränkten Implementierung der Win32-API enthält ETS mehrere eigene Kommunikationskomponenten. Hierzu zählen der TCP/IP-Stack, Treiber für Ethernet-Karten, ein Web-Server und eine einfache grafische Oberfläche. An dieser Stelle kann VenturCom auf ihre Erfahrungen mit dem Produkt *RTX* zurückgreifen (siehe oben).

RTOS-32, OnTime

Die deutsche Firma OnTime schuf mit dem Realzeitbetriebssystem *RTOS-32* und dem integrierten Win32-API *Subset* eine einfache Basis für die Ausführung von zeitkritischen Windows-Applikationen. Auch hier gelten die oben beschriebenen Nachteile, insbesondere werden nur ca. 160 Win32-Funktionen nachgebildet.

Willows RT für Tornado, Wind River Systems

Die Firma Wind River Systems bietet eine angepasste Version der Win32-API *Willows RT* der Firma Willows Software an, um Windows Applikationen in Realzeit auf dem VxWorks Realzeitbetriebssystem ablaufen zu lassen. Damit steht ein System zur Ausführung von Realzeit-Tasks zur Verfügung, welches einen sehr geringen Ressourcenbedarf hat. Wegen der erwähnten Nachteile und der geringen Nachfrage hat Wind River Systems dieses Produkt zwischenzeitlich eingestellt.

Willows, QNX

Auch die Firma QNX vertreibt für ihr Realzeitbetriebssystem QNX eine Version der *Willows RT* API um Windows-Applikationen mit Realzeitfähigkeit zu versehen [Ber97][Hil97]. Auch dieses Produkt ist inzwischen aufgrund der geringen Nachfrage und der eingeschränkten Funktionalität nicht mehr erhältlich.

Modifikation des Kernels

Eine naheliegende Methode um Windows NT realzeitfähig zu machen, wäre die Modifikation des Kernels. Hierzu müssten mehrere interne Strukturen wie Interrupt-Verwaltung, Scheduler, DPC Warteschlangenbearbeitung, Prioritätenvergabe und andere *low-level*-Funktionen verändert wer-

[33] Gilt für Windows NT 4.0 Server.

den. Der große Aufwand und die benötigte Verfügbarkeit des Quellcodes machen dieses Vorgehen nur für Microsoft selbst realisierbar. Microsoft betreibt intensive Forschungen auf diesem Gebiet, was zahlreiche Veröffentlichungen über die *Rialto*-Systemerweiterung belegen [JBF+96][JRR97] [JR99][JR00][JS00]. Diese Erfahrungen werden aber (noch) nicht dazu benutzt, das Standardbetriebssystem Windows NT mit Realzeit-Eigenschaften auszurüsten, sondern münden in das eigenständige Realzeitbetriebssystem Windows CE.

4.2.2 Erweiterungen für andere Standardbetriebssysteme

RT-Linux

Das Standardbetriebssystem Linux ist wie Windows NT nicht realzeitfähig. Eine Realzeiterweiterung für Linux ist *RT-Linux* [Yod99][HRS01]. Hier wird ein kleines Realzeitbetriebssystem (RT-Linux) installiert, welches Linux in der Task mit der niedrigsten Priorität (*idle task*) ausführt. Nur wenn keine Realzeitaufgaben zu bearbeiten sind, bekommt Linux den Prozessor zugeteilt. Der Aufbau von RT-Linux ist modularisiert und besteht aus der RT-Linux-Kernel-Komponente, dem Scheduler, dem Kommunikationsmodul, dem Modul zur Synchronisation und Interprozesskommunikation und dem Parametermodul. RT-Linux unterstützt zwei Typen von Prozessen, die als *sporadische* und *zyklische* Tasks bezeichnet werden. Sporadische Tasks werden durch externe Interrupts ausgelöst, entsprechen also den herkömmlichen Interrupt-Service-Routinen. Die vorhandenen Linux-Treiber müssen nicht angepasst werden, da die ISRs von Linux unverändert bleiben. Dies wird durch eine Emulation der Interrupt-Vektor-Tabelle erreicht (Patch der Dateien `irq.c` und `irq.h`). Tritt ein Interrupt auf, wird geprüft, ob er durch eine sporadische Task zu bearbeiten ist, und entsprechend ausgeführt. Anschließend wird die Interrupt-Verarbeitung von Linux ausgelöst, um eine (weitere) Verarbeitung zu ermöglichen. Die Ausführung einer zyklischen Task wird durch einen Zeitgeber-Interrupt getriggert. Zur Kommunikation und Synchronisation zwischen den Realzeit-Tasks stehen Nachrichten, globale Variablen und Semaphoren zur Verfügung. Für die Kommunikation zum Betriebssystem gibt es sogenannte RT-FIFOs, über die binäre Datenströme ausgetauscht werden können, sowie eine rudimentäre Kommunikation über einen gemeinsamen Speicherbereich (*shared memory*). Die Bearbeitung auf RT-Linux Seite erfolgt dabei immer asynchron um keine Realzeitanwendung zu stören. Eine zeitkritische Kommunikation mit Linux-Prozessen ist daher nicht möglich. Die Parameter der Realzeitumgebung sind unter Linux nicht sichtbar. Der Zugriff ist eingeschränkt nur über das /proc-Dateisystem möglich. Wie bei Kernel-Tasks üblich, fehlen auch bei RT-Linux-Tasks die Speicherschutzmechanismen. Programmierfehler können so das gesamte System zum Absturz bringen. Der größte Vorteil bei Verwendung der Kombination aus RT-Linux und Linux als Basis für Realzeitanwendungen ist die Verfügbarkeit des Quellcodes. Es gibt noch andere Erweiterungen, wie z.B. *RTAI for Linux*, welche Linux realzeitfähig machen. Diese basieren auf den gleichen oder sehr ähnlichen Methoden und sollen deshalb hier nicht näher erläutert werden.

Java

Java ist eine Programmiersprache mit starker Wachstumsrate. Sie hat sich von der Realisierung blinkender Webseiten hin zur flexiblen Sprache für eingebettete Systeme entwickelt. Durch die weitgehende Plattformunabhängigkeit und der damit verbundenen Portabilität wird eine hohe Wiederverwendbarkeit der Module erreicht [Sch98]. Die Java-Programme (*Java Byte Code*) können auf verschiedene Weise ausgeführt werden. So ist es möglich sie zu interpretieren (*Java Virtual Machine*), zur Laufzeit zu compilieren (*Just-in-time Compiler*) oder sie vorab zu compilieren. Die *Java*

Virtual Machine (JVM) bildet eine gesicherte und abgeschlossene Umgebung zur Ausführung der Java-Applikation. Durch Einschränkung des Sprachumfangs, Anpassung der zeitkritischen *Garbage Collection* und Compilieren und Linken zum Zeitpunkt des Ladens auf das Zielsystem[34] kann Realzeitfähigkeit erreicht werden [Süs98][Kro00].

Sicomp RTVM, Siemens

Eine realzeitfähige Implementierung einer *Java Virtual Machine* ist das Softwaremodul *Sicomp RTVM* der Firma Siemens [Fre99][Fre00]. Sicomp RTVM führt *Java Byte Code* in Realzeit aus und ist auf den Betriebssystemen RMOS3 oder Windows NT 4.0 ablauffähig. Es setzt auf der von Hewlett-Packard konzipierten Technologie *ChaiVM* auf. Die VM entspricht in ihrem Sprachumfang einem Subset des *Embedded Java* von Sun. Die Entwicklung von Java-Programmen kann auf einem Standard-PC erfolgen. Wenn das Programm simuliert und ausgetestet ist, erfolgt eine Übertragung auf das Zielsystem.

4.2.3 Vergleich der verschiedenen Realzeiterweiterungen

Tabelle 4.4 listet die beschriebenen Realzeiterweiterungen mit ihren Vor- und Nachteilen und den erreichbaren Interrupt-Latenzzeiten bei hoher Systemlast (*worst case*) auf. Diese Latenzzeiten sind wesentlich von der Architektur des x86-Systems, speziell von den Geschwindigkeiten der internen Busse (*Front Side Bus* (FSB), *North Bridge* ⇔ *South Bridge*, Speicheranbindung usw.) und der internen Struktur der Realzeiterweiterung abhängig. Die Leistungsfähigkeit des Prozessors spielt eine weitere Rolle.

Methode	Erweiterung	Hersteller	Vorteile	Nachteile	Interrupt-Latenzzeiten[35]
Modifikation des Hardware Abstraction Layers und zusätzliches Realzeitsubsystem	RTX 4.2	VenturCom	Keine zusätzliche Hardware Kompatibilität von Realzeitsubsystem- und Win32-Programmen Zusätzliche DLL für weichen Realzeitbetrieb Sicheres Verhalten bei Systemabstürzen Geringer Jitter der Messergebnisse	Mögliche Inkompatibilitäten durch Modifikationen des HAL Zwei APIs Kein eigener geschützter Speicherbereich für das Realzeitsubsystem Nur rudimentäre Kommunikationsmöglichkeiten zwischen Windows NT und dem Realzeitsubsystem Debugging im Kernelmode	Pentium 200: 50 µs Pentium III 500: 30 µs
	Intime 1.20	TenAsys	Keine zusätzliche Hardware Eigener Speicherbereich für jede Realzeit-Task Zuverlässiges RTOS (iRMX) Einbettung der Entwicklungsumgebung in MS Visual C++ Integrierter Debugger vereinfacht Fehlersuche Sehr kleine Zeiten für Taskumschaltung NT → RT → NT	Mögliche Inkompatibilitäten durch Modifikationen des HAL Eingeschränkte Kommunikationsmöglichkeiten zwischen Windows NT und dem Realzeitsubsystem Hohe Kosten für Laufzeitsystem-Lizenzen Fragliche Stabilität/Sinn des Weiterbetriebs bei Absturz (*blue screen*) des Windows NT Systems	Pentium 133: 37 µs PentiumPro 200: 43 µs Pentium II 450: 40 µs

[34] Umsetzung des Java Byte Code in Maschinencode und statische Zuweisung zeitkritischer Routinen.
[35] Alle Zeiten sind Hersteller-Angaben.

Methode	Erweiterung	Hersteller	Vorteile	Nachteile	Interrupt-Latenzzeiten[35]
	HyperKernel 4.3	Nematron	Keine zusätzliche Hardware Keine Modifikationen des HAL Geringer Eingriff in das Windows NT System	Unsauberes Zeitschlitzverfahren Keine ereignisbasierte Ausführung von Realzeit-Tasks Fragliche Stabilität/Sinn des Weiterbetriebs bei Absturz (*blue screen*) des Windows NT Systems	Immer zyklischer Ausführungstakt: 25,50,100,250 µs Pentium 100: 37 µs PentiumPro 200: 34 µs Pentium II 266: 18 µs
Installation eines Realzeitsubsystems als Treiber im Kernelmode	TwinCAT	Beckhoff	Konform zu Windows Keine Probleme beim Upgrade des Windows-Betriebssystems Geringere Komplexität im Vergleich zur Modifikation des HAL	Ungeeignete Treiber verschlechtern das Realzeitverhalten Evaluierung des Gesamtsystems nötig Kein eigener geschützter Speicherbereich für die Realzeit-Tasks Zyklischer Ausführungstakt	Immer zyklischer Ausführungstakt: 100 µs Jitter: ± 15 µs
	TCX	Bsquare	Konform zu Windows Keine Probleme beim Upgrade des Windows-Betriebssystems Geringere Komplexität im Vergleich zur Modifikation des HAL	Ungeeignete Treiber verschlechtern Realzeitverhalten Evaluierung des Gesamtsystems nötig Kein eigener geschützter Speicherbereich für die Realzeit-Tasks Entwicklung und Debugging im Kernelmode	Pentium 200: 37 µs Pentium II 233: 30 µs
Bereitstellen einer Win32-API auf einem Realzeitbetriebssystem	Willows RT RTOS-32	Wind River Systems, QNX OnTime	Keine zusätzliche Hardware Bewährtes RTOS Garantiertes Zeitverhalten Geringer Ressourcenbedarf	Source Code nötig Compilierung nötig (meist) keine Implementierung von GUI-Funktionen Mehrere, teuere Entwicklungssysteme Verlust der leistungsfähigen Windows NT Treiberschnittstelle Keine Ausführung von Windows Programmen möglich	Je nach RTOS und CPU: > 4 µs Elan 486: 5 µs
	ETS	VenturCom	Keine zusätzliche Hardware Garantiertes Zeitverhalten Geringer Ressourcenbedarf Entwicklung mit Standard-Werkzeugen (MS Developer Studio) möglich Zusätzliche Komponenten (TCP/IP-Stack, Webserver, GUI) vorhanden	Source Code nötig Compilierung nötig Verlust der leistungsfähigen Windows NT Treiberschnittstelle Keine Ausführung von Windows Programmen möglich	Pentium 90: 13 µs Pentium III 600: 7 µs
Umleiten des NMI auf eine ISR	RTWin Toolkit	LP Elektronik	Entwicklung mit Standard-Werkzeugen (MS Developer Studio) möglich Garantierte Einhaltung der Realzeitbedingungen Läuft unter Windows 3.x, Windows 95, NT	Nur kleine Codegrößen in der ISR sinnvoll Zusätzliche Hardware nötig Kein Speicherschutz für Realzeitanwendungen Kommunikation nur über gemeinsamen Speicherbereich	5 µs, oder zyklischer Timer: einstellbar von 7 µs – 27 ms
Realzeitsubsystem mit Standardbetriebssystem als Task	RT-Linux 3.0 RTAI	University of New Mexico and FSM Labs University of Milan	Keine zusätzliche Hardware Unverändertes Gastbetriebssystem Kompatibilität von Realzeitsubsystem- und Linux-Programmen Weitere Scheduler-Implementierungen (EDF) Source Code vorhanden	Kein Speicherschutz für Realzeitanwendungen Nur FIFO als Kommunikationsmöglichkeiten zwischen Linux und dem Realzeitsubsystem Umständliche Konfiguration des Realzeitsubsystems durch ein Dateisystem Keine zeitkritische Kommunikation mit Linux-Prozessen Eingeschränkte Qualität und Support für Entwicklungstools	Pentium 200: 30 µs Pentium II 350: 22 µs Pentium II 300: 19 µs Pentium III 650: 16 µs

Methode	Erweiterung	Hersteller	Vorteile	Nachteile	Interrupt-Latenzzeiten[35]
Realzeitfähige Java Virtual Machine auf Realzeitbetriebssystem	Sicomp RTVM	Siemens	Portabilität Relativ geringer *Overhead* Integrierte *Garbage Collection* verringert Programmierfehler Fernladen von Modulen	Lange Startzeit durch compilieren und linken zur Startzeit Erhöhter Ressourcenbedarf für Compiler und Linker Treiber für Zugriff auf Hardware nötig	Pentium 200: 125 µs

Tabelle 4.4 Vergleich der verschiedenen Realzeiterweiterungen

Quellen der Testergebnisse und Messungen: [TBU98][Qui97][FTY98][Nem96][Vmi98][TM97/2] [Yod99][PB00][Fre99][Obe99]

Die Reaktionszeit, d.h. die Zeit vom Auftritt des externen Ereignisses bis zur Ausgabe der Reaktion, eines PC-Systems unter Verwendung der Standard-PC-Hardware und einem Standardbetriebssystem mit Realzeiterweiterung setzt sich zusammen aus:

$$t_{R\,OS+E} = t_{I\,PC} + t_{HAL} + t_{ISR} + t_{RTSS} + t_D \geq 20 - 50\ \mu s$$

mit: R = Reaktionszeit, I = Interrupt-Latenzzeit des PCs, HAL = Latenzzeit durch Umgehung des HAL, ISR = Ausführungszeit der Interrupt-Service-Routine, RTSS = Ausführungszeit im Realzeitsubsystem bis zur Ausgabeanweisung, D = Datenübertragungszeit CPU → I/O. Die erreichbaren Reaktionszeiten variieren je nach Architektur der verwendeten Realzeiterweiterung; sie liegen jedoch überwiegend im Bereich von 20 – 50 µs.

4.2.4 Probleme beim Einsatz von Realzeiterweiterungen

Der Einsatz der Realzeiterweiterungen mit der Methode der Modifikation des *Hardware Abstraction Layers* und einem zusätzlichen Realzeitsubsystem erlaubt es, mit verschiedenen Methoden, zeitkritische Anwendungen unter Windows NT auszuführen. Gemeinsames Merkmal ist die Veränderung der Interrupt-Bearbeitung zur Sicherstellung von geringen Latenzzeiten. Ein Problem ist, dass Unterbrechungen, welche zur Laufzeit von Windows NT auftreten, nicht überwacht werden können. So maskieren manche BIOS- und Kernel-Funktionen während der Ausführung von atomaren Befehlssequenzen (*critical sections*) die Interrupts für einen Zeitraum von 20 – 50 µs [Qui97]. Laut [Por98] existieren nicht-systemkonforme Treiber (z.B. SCSI- und IDE-Treiber von Festplattencontrollern), welche im ungünstigsten Fall alle Interrupts für bis zu 2 ms sperren. Ein Grund könnte das Zusammentreffen von Pollingverfahren und nicht reagierenden Festplatten während einer Kopfkalibrierung sein. Nicht vorschriftsmäßig programmierte Treiber können Interrupts für große Zeitintervalle deaktivieren (maskieren) und damit Realzeitanwendungen vollständig blockieren. Um ein konformes Verhalten der installierten Treiber zu überprüfen, gibt es in den Entwicklungswerkzeugen der meisten Realzeiterweiterungen diverse Verifikationstools zur Langzeitmessung des realen Zeitverhaltens. Ein maximaler *Worst case*-Werts kann auf diese Weise jedoch nicht angegeben werden.

Fazit

Prinzipiell können die Einflüsse beim Einsatz von Realzeiterweiterungen, welche zu erhöhten Latenzzeiten führen, in zwei Fälle eingeteilt werden. Erstens gibt es Verzögerungen, die durch den zusätzlichen Aufwand der Interrupt-Verarbeitung im Realzeitsubsystem entstehen, während Windows ab-

läuft. Zweitens ergeben sich systemimmanente Verzögerungen durch verspätetes Erkennen des Interrupts im Prozessor. Der zweite Fall dominiert, wie die relativ geringe Abhängigkeit der Verzögerungsveränderung von der Leistungssteigerung des Prozessors erkennen lässt. Dagegen verschlechtern sich die Interrupt-Latenzzeiten bei Zunahme der Systemlast erheblich.

4.3 Verwendung von Realzeitbetriebssystemen

Die von Realzeitbetriebssystemen und deren Kombination mit Ein-/Ausgabe-Komponenten abgedeckte Parameterfläche zeigt Bild 4.10.

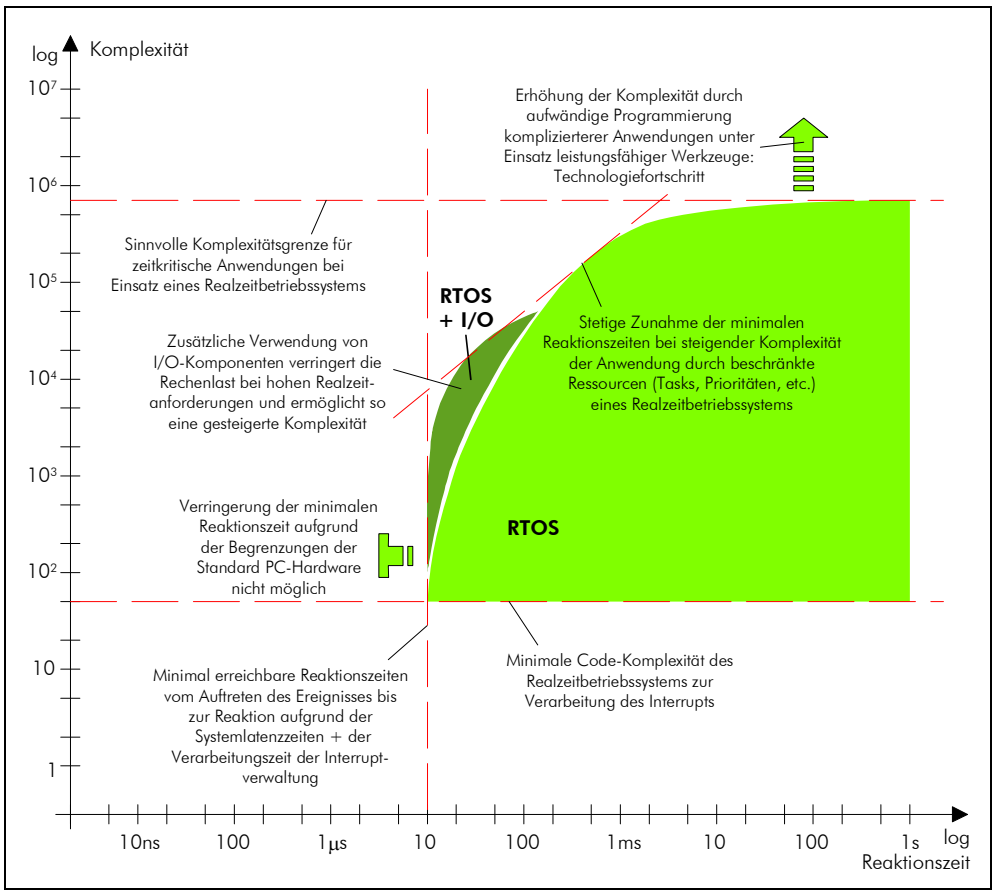


Bild 4.10 Parameterbereich der Realzeitbetriebssysteme (RTOS, RTOS + I/O)

4.3.1 Realzeitbetriebssysteme für zeitkritische Prozesse

Eine klassische Verwendung von Realzeitbetriebssystemen (RTOS) ist der Einsatz in Systemen zur Steuerung und Regelung von zeitkritischen technischen Prozessen. In der Vergangenheit wurden Realzeitbetriebssysteme meist auf proprietären Hardware-Plattformen implementiert. Heute gibt es vermehrt Realzeitbetriebssysteme für die PC-Architektur. Beim Einsatz für Automatisierungsaufgaben ist die Einhaltung der Realzeitbedingungen ausschlaggebend. Hierzu besitzt ein Realzeitbetriebssystem mehrere deterministische Mechanismen: einen Scheduler der preemptives Multitasking unterstützt, eine Task-Verwaltung für die Koordination paralleler Tasks und die Kommunikation zwischen ihnen, und eine Interrupt-Behandlung zur Reaktion auf Ereignisse. Außer der wichtigen Kenngröße *Latenzzeit* werden noch weitere Parameter zur Qualitätsbestimmung eines Realzeit-

betriebssystems herangezogen [Jan00]. So ist ein direkter Zugriff auf die Hardware nötig, um dem Softwareentwickler einen einfachen und schnellen Zugriff auf I/O-Schnittstellen und Speicher zu ermöglichen. Ein Speicherschutz, wie ihn Standardbetriebssysteme bereitstellen, ist nicht integriert. Wichtig für den rauen Industrieeinsatz sind ferner robustes Verhalten im Fehlerfall, z.B. eine *Brown-out*-Erkennung, und sicheres Herunterfahren, z.B. bei einem Not-Aus. Für komplexe Anwendungen mit einer Vielzahl von verschiedenen Tasks sollte eine ausreichende Anzahl von Prioritäten verfügbar sein. Realzeitbetriebssysteme unterstützen deshalb meist 64 – 256 verschiedene Prioritäten. Die Verwaltung zusätzlicher Interrupt-Quellen von kaskadierten Interrupt-Controllern sollte möglich sein. Das Realzeitbetriebssystem muss auch die vom jeweiligen Prozessor angebotenen Schutzmechanismen und deren Ausnahmebehandlung (*exceptions*) unterstützen, um einen sicheren Betrieb zu gewährleisten.

Ein Vergleich gängiger Realzeitbetriebssysteme für x86-Architekturen zeigt Tabelle 4.5 [Jan00] [Wol97]. Eine Normierung auf die Latenzzeit eines Pentium Prozessors mit 200 MHz und anschließende Einteilung in 3 Latenzzeitklassen (A: $< 3\ \mu s$, B: $< 7\ \mu s$, C: $> 7\ \mu s$) ermöglicht einen besseren Vergleich. Hierbei ist der sehr unterschiedlich Umfang der Realzeitbetriebssysteme zu beachten.

RTOS-Name	Hersteller	Prozessor	Interrupt-Latenzzeit[36] in [µs]	Latenzzeit-klasse	Anzahl der Prioritäten
µC/OS-II	Micrium	i486/66	5	A	64
AMX	Kadak	Pentium 100 i486/50	2,2 6,3	A	256
ChorusOS 3.1	Sun	i486/50	8	B	?
LynxOS 3.0	LynxWorks	PII 333	11	C	256
OnCore	OnCore Systems	Pentium III 650	2	B	32-256
pSOSystem 2.26	Integrated Systems	i486/33	4	A	256
PXROS	HighTec EDV	i486/33	28	C	32
QNX 4	QNX	Pentium 166 i486/33	3,3 8	B	32
RMOS 3.20	Siemens	i486/100	4	A	256
RTKernel-32	On-Time	i486/33	5	A	64
RTX	CMX	i486/25	4,7	A	256
VRTX	Mentor	i486/100	2,8	A	256
VxWorks 5.3	Wind River Systems	Pentium 133	5,1	B	256
Windows CE 3.0[37]	Microsoft	Pentium 166	10	C	256
(Windows NT 4.0)[38]	Microsoft	Pentium 90	>14	-	16/32

Tabelle 4.5 Vergleich von Realzeitbetriebssystemen

Die folgende Grafik in Bild 4.11 zeigt den prozentualen Anteil von Standardbetriebssystemen, Realzeiterweiterungen für Standardbetriebssysteme und reinen Realzeitbetriebssystemen in aktuellen Projekten der Industrieautomatisierung.

[36] Interrupt-Latenzzeiten inkl. ISR-Management entsprechend den Herstellerangaben, Definition in Kapitel 3.2.2.
[37] minimale Latenzzeit der ISR ohne IST (*Interrupt Service Thread*, ähnlich DPC), Latenzzeit des IST ca. 100 µs.
[38] minimale Latenzzeit der ISR ohne DPC.

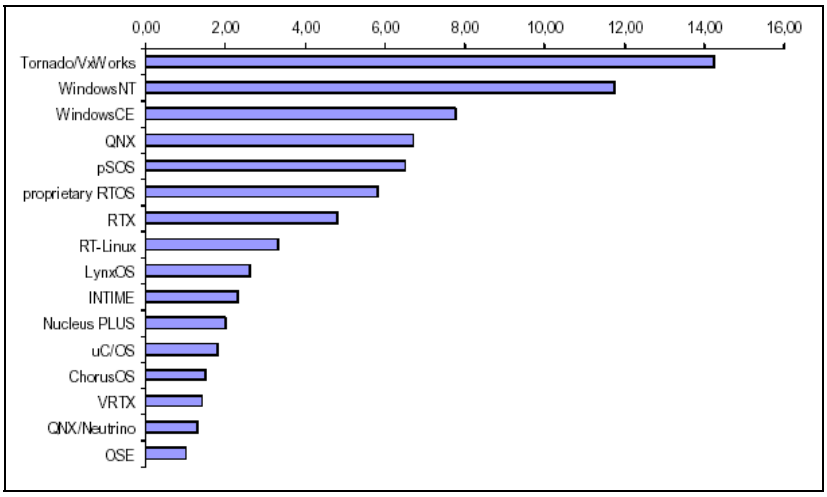


Bild 4.11 Prozentualer Anteil der Betriebssysteme und -Erweiterungen[39]

4.3.2 Grenzen der Realzeitbetriebssysteme

Die vielfältigen Randbedingungen, welche bei der Entwicklung von Systemen für Realzeitanwendungen zu beachten sind, haben starke Einflüsse auf deren praktische Implementierung. So führten die Beschränkungen bezüglich Gewicht, Größe, Leistung und Stromverbrauch bei der Entwicklung von Realzeitsystemen zu einem technischen Rückstand gegenüber den aktuellen Methoden der Soft- und Hardware-Entwicklung für Standardsysteme [GKL+99]. Deshalb gelten Realzeitsysteme immer öfter als teuer, zeitaufwändig in der Entwicklung, Validierung, Optimierung und Wartung und als schwierig aufzurüsten. Durch ihre Spezialisierung können sie nur mit hohem Aufwand an wechselnde Aufgabenstellungen, geänderte Markterfordernisse oder neue technische Innovationen angepasst werden. Auch die wenigen Systemressourcen förderten den Einsatz von unflexiblen Methoden und proprietären Werkzeugen.

Die minimalen Reaktionszeiten von Realzeitbetriebssystemen erreichen die Grenzen der PC-Hardware (siehe auch Kapitel 5.2). Die verschiedenen Realzeitbetriebssysteme lassen sich in zwei Hauptgruppen unterteilen. Die „vollständigen" Realzeitbetriebssysteme enthalten umfangreiche Systemfunktionen, wie komfortable Taskverwaltung, Interrupt-Handler, Speicherverwaltung, zahlreiche Kommunikationsverfahren (Semaphoren, Mailboxen, Queues, Mutexe) und Timerunterstützung. Die zweite Gruppe bilden die Realzeit-Kernel mit nur rudimentärer Betriebssystem-Funktionalität. Dadurch benötigen sie erheblich weniger Ressourcen und erreichen in eingebetteten Systemen auch auf leistungsschwachen Prozessoren sehr geringe Reaktionszeiten. Die Komplexität der Aufgaben ist hierbei natürlich begrenzt. Durch die meist fehlenden Interrupt-Handler kann unmittelbar auf externe Unterbrechungen reagiert werden. Die Interrupt-Latenzzeiten setzten sich in diesem Fall zum größten Teil aus den systeminternen Kommunikationszeiten für die Weitergabe des Interrupts zusammen. Bei vollständigen RTOS addieren sich noch zusätzliche Wartezeiten durch längere Blockierungen (*critical sections*) und Rechenzeiten der aufwändigeren Systemverwaltung. Bei aktuellen PC-Prozessoren (0,5 – 1 GHz) ist diese zusätzliche Rechenzeit allerdings vernachlässigbar ($< 1\ \mu s$).

[39] Quelle: Firma 3Soft, 2001.

Die Reaktionszeit, d.h. die Zeit vom Auftritt des externen Ereignisses bis zur Ausgabe der Reaktion, eines PC-Systems unter Verwendung der Standard-PC-Hardware und einem Realzeitbetriebssystem setzt sich zusammen aus:

$$t_{R\,RTOS} = t_{I\,PC} + t_{ISR} + t_{RTOS} + t_D \geq 2 - 20\ \mu s$$

mit: R = Reaktionszeit, I = Interrupt-Latenzzeit des PCs, ISR = Ausführungszeit der Interrupt-Service-Routine, RTOS = Ausführungszeit des Realzeitbetriebssystems bis zur Ausgabeanweisung, D = Datenübertragungszeit CPU → I/O. Einfache Realzeit-Kernel erreichen *worst case*-Reaktionszeiten von ca. 2 – 5 μs, vollständige RTOS liegen bei ca. 5 – 20 μs.

4.3.3 Kombination mit Ein-/Ausgabe-Komponente

Eine Erweiterungsmöglichkeit eines Standard-PC-Systems mit Realzeitbetriebssystem ist das Hinzufügen einer Ein-/Ausgabe-Komponente oder auch I/O-Komponente. Sie zeichnet sich durch Verwendung von einfachen und kostengünstigen Ein- und Ausgabebausteinen ohne besondere Eigenintelligenz aus. Als Verbindung zum PC können, abhängig von der Leistungsanforderung, alle vorhandenen PC-Schnittstellen dienen. Vielfach werden diese Ein-/Ausgabe-Komponenten als externe Prozessperipherie [Fär94] zur Anpassung und Vorverarbeitung (z.B. Linearisierung) von elektrischen und nichtelektrischen Größen an die Logikpegel des PCs eingesetzt. Auch zur Umsetzung von speziellen Buspegeln, wie z.B. differenzielle Pegel, werden Ein-/Ausgabe-Komponenten verwendet.

Leistungssteigerung durch Ein/Ausgabe-Komponenten

Bei ereignisgesteuerten Systemen (*event-based systems*), welche eine direkte Reaktion auf externe Signale erfordern, kann man mit hinzugefügten Ein-/Ausgabe-Komponenten aufgrund fehlender freiprogrammierbarer Eigenintelligenz keine Verringerung der Reaktionszeiten bzw. Verbesserung der Realzeitfähigkeiten gegenüber der Standard-PC-Hardware erzielen. Dagegen kann die Verarbeitungsleistung bei der Datenübertragung (*streaming*) durch den Einsatz von Ein-/Ausgabe-Komponenten, speziell unter Verwendung von Zwischenspeichern (*buffer*) und schneller *Burst*-Modi, gesteigert werden.

Beispiel: PC-System mit Realzeitbetriebssystem und Ein-/Ausgabe-Komponente (externer UART mit großem integrierten FIFO) zum Anschluss an den PCI-Bus

Die interne serielle Schnittstelle aktueller PCs ist meist mit dem UART-Baustein 16550 bzw. einer dazu kompatiblen Einheit in der *South Bridge* realisiert. Da moderne *„legacy free"*[40] PC-Chipsätze keine Schnittstellen für Tastatur, Maus, Floppy-Laufwerk, parallele und serielle Schnittstelle mehr besitzen (sollen), müssen diese mit einem zusätzlichen *Super-I/O*-Baustein nachgerüstet werden. Die maximale Baudrate[41] beträgt bei Verwendung eines 16 Byte FIFOs maximal 115,2 KBit/s.

Will man höhere Datenübertragungsgeschwindigkeiten erzielen, muss man externe Ein-/Ausgabe-Komponenten mit schnelleren UARTs nachrüsten. Mit einer Einsteckkarte für den PCI-Bus und z.B. dem 16850 UART-Chip erreicht man unter Windows bis zu 921 KBit/s. Benutzt man geringere

[40] Bedeutung: Verzicht auf ältere PC-Technologien, speziell bei Schnittstellen.
[41] Die Baudrate gibt die Anzahl von Zustandsänderungen (Schritte, Takte) pro Sekunde auf einer Übertragungsstrecke an. Dabei ist nicht festgelegt, wie viele Bits pro Schritt übertragen werden.

Geschwindigkeiten, z.B. 115,2 KBit/s, erhöht sich, aufgrund der großen Sende- und Empfangspuffer von 2 x 128 Bytes das Intervall zwischen zwei Interrupts (FIFO_FULL) von ursprünglich 1,53 ms auf 12,2 ms[42]. Eine Übersicht gängiger UART-Bausteine und -Implementierungen zeigt Tabelle 4.6. Die maximale Baudrate des Bausteins wird erreicht durch: Einsatz eines diskreten Bausteins, Verwendung des größtmöglichen Quarzes, Nutzung des kleinsten internen Frequenzteilers (meist 1). Die maximal erreichbare Baudrate eines Standard-PC-UARTs hängt demzufolge von der Art der Implementierung des integrierten UARTs und der Frequenz des eingesetzten Quarzes ab. Geschwindigkeiten über 115,2 KBit/s sind, falls unterstützt, nur mit einem zusätzlichen Software-Treiber erreichbar, der die Register des eingesetzten UARTs bzw. der UART-Implementierung direkt programmieren kann.

Eigenschaft	**8250**	**16450**	**16550**	**16650**	**16850**	**16950**
Max. Baudrate(n) (diskreter Baustein)	9,6 KBit/s	115,2 KBit/s	115,2 KBit/s [1] 128 KBit/s [2] 1,5 MBit/s [5]	115,2 KBit/s [1] 460,8 KBit/s [3] 921,6 KBit/s [4] 1,5 MBit/s [5]	115,2 KBit/s [1] 460,8 KBit/s [3] 921,6 KBit/s [4] 1,5 MBit/s [5]	115,2 KBit/s [1] 921,6 KBit/s [4] 3,68 MBit/s [6] 15 MBit/s [7]
Max. Baudrate(n) (South Bridge-/Super-I/O-Implementierung)	9,6 KBit/s	115,2 KBit/s	**115,2 KBit/s** [1] 460,8 KBit/s [3]			
FIFO Tiefe (Bytes)	2x 1	2x 1	2x 16	2x 32	2x 128	2x 128
Anzahl programmierbarer Interrupt-Schwellwerte	1	1	4	4	4	128
Infrarot (IrDA)	Nein	Nein	Nein	Ja	Ja	Ja
RS485-Enable Ausgang	Nein	Nein	Nein	Nein	Ja	Ja

Tabelle 4.6 UART-Bausteine und –Implementierungen

[1] **115,2 KBit/s** mit 7,3728 MHz Quarz im **Standard-Kompatibilitätsmode**
[2] 128 KBit/s mit 18,432 MHz Quarz beim 16550D und spezieller Registerprogrammierung
[3] 460,8 KBit/s mit 7,3728 MHz Quarz und zusätzlichem Software-Treiber, sofern von Implementierung unterstützt
[4] 921,6 KBit/s mit 14,7456 MHz Quarz und spezieller Registerprogrammierung
[5] 1,5 MBit/s mit 24 MHz Quarz und spezieller Registerprogrammierung
[6] 3,68 MBit/s mit 14,7456 MHz Quarz und spezieller Registerprogrammierung
[7] 15 MBit/s mit 60 MHz Quarz und spezieller Registerprogrammierung

Spezielle Ein-/Ausgabe-Komponenten für PCs

Zum Anschluss komplexer Peripherie an ein PC-System werden spezielle Ein-/Ausgabe-Komponenten eingesetzt. Diese realisieren aufwändige Busprotokolle mit einer Kombination aus anwenderspezifischen Bausteinen (ASICs[43]) und programmierbaren Mikrocontrollern. Die ASICs implementieren meist geschwindigkeitskritische Teilaufgaben der unteren ISO-OSI-Schichten, wie z.B. des *Physical layers* oder des *Data link layers*. Protokollaufgaben darrüberliegender Schichten werden mit Controllern realisiert. Die programmierbare Firmware ermöglicht eine flexible Anpassung an zukünftige Protokollerweiterungen. [FMO+98] beschreibt eine mögliche Nutzung des vorhandenen Mikrocontrollers einer intelligenten Netzwerkkarte zur Entlastung der Haupt-CPU bei Media-Streaming.

[42] Annahme: 11 Bit Rahmen mit 1 Startbit, 8 Datenbits, 1 Parity-Bit, 1 Stoppbit.
[43] ASIC: ***A**pplication-**S**pecific **I**ntegrated **C**ircuit.*

Trotz des Einsatzes von (programmierbaren) Mikrocontrollern dürfen diese Ein-/Ausgabe-Komponenten nicht mit den im nachfolgenden Kapitel beschriebenen intelligenten Mikrocontroller-Komponenten verwechselt werden. Der Grund ist die fehlende Programmierfähigkeit für den Anwender. Nur der Hersteller verfügt über Wissen und Werkzeuge zur Änderung der vorhandenen Firmware. Somit ist es unmöglich, applikationsspezifische Algorithmen und Berechnungen für eine Entscheidungsfindung und damit die geforderte schnelle Reaktion auf externe Ereignisse zu implementieren. Auch gegenüber den in Kapitel 4.3.1 beschriebenen reinen Softwaresystemen (PC + RTOS) ergibt sich keine prinzipielle Verbesserung der Realzeitfähigkeiten (Reaktionszeiten). Jedoch wird durch die Verlagerung von Rechenleistung in leistungsfähige Spezialbausteine eine signifikante Steigerung des maximalen Datendurchsatzes erreicht.

In PC-Systemen können viele Module der Klasse der intelligenten Ein-/Ausgabe-Komponenten zugeordnet werden. Schnittstellen(-karten), die zum Anschluss von PC-Peripherie über leistungsstarke Bussysteme mit aufwändigen Protokollen eingesetzt werden, übernehmen manche Teilentscheidungen selbständig. So werden z.B. bei Übertragungsfehlern selbständig neue Datenpakete angefordert, ohne dass dies dem PC-Prozessor mitgeteilt wird. Eine Verringerung der systematischen Reaktionszeiten resultiert dennoch nicht, da die anschließende Verarbeitung der Daten für eine Entscheidungsfindung (Reaktion) ausschließlich durch den Haupt-Prozessor erledigt wird.

Die Reaktionszeit, d.h. die Zeit vom Auftritt des externen Ereignisses bis zur Ausgabe der Reaktion, eines PC-Systems unter Verwendung der Standard-PC-Hardware, eines Realzeitbetriebssystems und einer Ein-/Ausgabe-Komponente ohne Eigenintelligenz als Zusatzhardware setzt sich zusammen aus:

$$t_{R\,RTOS+I/O} = t_{I\,PC} + t_{ISR} + t_{I/O} + t_{RTOS} + t_{D} \geq 2 - 20\ \mu s$$

mit: R = Reaktionszeit, I = Interrupt-Latenzzeit des PCs, ISR = Ausführungszeit der Interrupt-Service-Routine, I/O = Ausführungszeit der Vorverarbeitung in der I/O-Komponente, RTOS = Ausführungszeit des Realzeitbetriebssystems bis zur Ausgabeanweisung, D = Datenübertragungszeit CPU → I/O. Die Reaktionszeiten entsprechen bei niedriger Aufgabenkomplexität denen der reinen Realzeitbetriebssysteme.

Industrieautomatisierung

Auch Komponenten für die Industrieautomatisierung zählen zur Klasse der Ein-/Ausgabe-Komponenten, sofern sie nicht über frei programmierbare Mikrocontroller verfügen. Bespiele hierfür sind PCI-Steckkarten oder externe USB-Komponenten für den CAN-Bus, Profibus und andere Feldbusse. Sie bestehen zumeist aus Busprozessor-ASIC, zusätzlichem Mikrocontroller zur Realisierung der oberen ISO-OSI-Schichten und einem Schnittstellencontroller, z.B. einer PCI-Bridge oder einem USB-Controller zur Kommunikation mit dem PC. Bei verbreiteten Feldbussen existieren auch Single-Chip Lösungen, welche neben der Steuerungs-CPU auch den Schnittstellen-Baustein auf einen Chip integrieren.

4.4 Mehrprozessorsysteme

Mehrprozessorsystem mit Windows NT

Der steigende Einsatz von Mehrprozessorsystemen, speziell in Form von preisgünstigen Doppelprozessor-Systemen auf x86-Basis, eröffnet eine weitere Möglichkeit der Datenverarbeitung von zeitkritischen Prozessen. Der naheliegende Schritt ist die Verwendung eines Doppelprozessor-Systems mit einem Standardbetriebssystem wie Windows NT. Hier wäre eine Zuordnung der realzeitkritischen und unkritischen Prozesse auf verschiedene Prozessoren denkbar.

Um es vorwegzunehmen: Auch auf einem Doppelprozessor-Systems erreicht man unter Windows NT keine harte Realzeit. Windows NT gestattet zwar eine Zuordnung, welcher Prozess auf welchem Prozessor abläuft, sowie eine Festlegung, auf welchem Prozessor welcher Interrupt ausgelöst wird. Damit lassen sich mehrere Interrupt-Service-Routinen unterschiedlicher Priorität gleichzeitig auf verschiedenen Prozessoren ausführen. Der Flaschenhals in der Abarbeitung liegt aber in der Realisierung der Speicherung der *Deferred Procedure Calls* (DPC) als Warteschlange (*queue*). Erst wenn diese leer ist, kann das System die Anwender-Software weiter bearbeiten. Weiterhin ist es in Windows NT nicht möglich, DPCs Prioritäten oder Zuordnungen auf Prozessoren zuzuweisen. Es gibt nur eine DPC-Warteschlange im gesamten System, wobei der Zugriff mehrerer Prozessoren über *spinlocks* geregelt wird. Das Benutzerprogramm kommt also trotz möglicher Verteilung der Interrupts erst dann wieder an die Reihe, wenn alle DPCs von den Prozessoren bearbeitet wurden. Damit entspricht das Verhalten des Systems dem der reinen Software-Implementierung (Kapitel 4.1), unabhängig von der Anzahl der Prozessoren. Beim Entwurf von Gerätetreibern für Mehrprozessorsysteme sollte man diese Problematik berücksichtigen, um Überraschungen zu vermeiden. Es gilt auch hier die Forderung, keine realzeitkritischen Prozesse mit dieser Methode zu steuern.

Mehrprozessorsystem mit zwei unterschiedlichen Betriebssystemen

Trotz der gerade gezeigten Probleme besticht die Idee der Nutzung eines Doppelprozessor-Systems durch ihre Raffinesse. In der aktuellen Forschung wird untersucht, wie zwei unterschiedliche Betriebssysteme auf den Prozessoren ablaufen können. [Sto00] benutzt hierzu das Standardbetriebssystem Linux und ein dezidiertes Realzeitbetriebssystem (RTEMS). Die ersten Ergebnisse sind vielversprechend, jedoch scheinen Abhängigkeiten zwischen den Hardwarekomponenten oder architekturspezifisches Verhalten die Realzeitfähigkeit zu beeinflussen. Das Forschungsgebiet ist dennoch sehr interessant und eine funktionsfähige Implementierung wäre hinsichtlich Kosten und Entwicklungsaufwand konkurrenzfähig zu existierenden Lösungen.

5 Hardwarebasierte Systemarchitekturen

Im letzten Kapitel wurden softwarebasierte Lösungen zur Realisierung von zeitkritischen technischen Prozessen behandelt. Für Prozesse, bei denen die minimal erreichbaren Reaktions- und Latenzzeiten der Standard-PC-Hardware in Verbindung mit einem Realzeitbetriebssystem nicht ausreichen, muss die Hardware gezielt erweitert werden. Zwei unterschiedliche Realisierungsmethoden sind die *Intelligente Controller-Komponente* (Kapitel 5.1.1) und die *Programmierbare Logik-Komponente* (Kapitel 5.1.2). Die Schwerpunkte bilden die Beschreibung der Implementierung sowie die Analyse der jeweiligen Vor- und Nachteile. Im darauffolgenden Kapitel 5.2 werden die Grenzen der PC-Systemarchitektur aufgezeigt. Speziell die Architektur des PCI-Busses als wichtige PC-Schnittstelle ist Gegenstand einer intensiven Analyse. Um diese Begrenzungen zu überwinden gibt es mehrere Vorschläge, von denen sich einige bereits in der Umsetzungsphase befinden. Das Kapitel 5.3 gibt hier einen Überblick der neuen Architekturkonzepte.

5.1 Auslagerung zeitkritischer Prozesse auf zusätzliche Hardware

Im Unterschied zu den software-basierten Systemarchitekturen des 4. Kapitels werden nun hardwarebasierte Lösungen vorgestellt. *Hardwarebasiert* bedeutet hierbei eine Auslagerung der Bearbeitung zeitkritischer Algorithmen auf zusätzliche Komponenten, welche über verschiedene Schnittstellen eines PC-Systems angebunden sein können. Eine Grenze, ab der es sinnvoll ist, die Verarbeitung besonders zeitkritischer technischer Prozesse durch zusätzliche Hardware zu realisieren, ergibt sich aus den Realzeitanforderungen. Der Übergang ist fließend und stark von Parametern und Randbedingungen des technischen Prozesses sowie den Anforderungen des Anwenders abhängig. Als interessantes Beispiel für einen zeitkritischen technischen Prozess am Übergang von der Soft- zur Hardware soll die Realisierung der Software-Modems für PCs dienen.

Problematik von Software-Modems

Ein interessantes Beispiel für den Übergang von software- zu hardwarebasierten Lösungen ist die Realisierung des Software-Modems (auch *Softmodem* genannt) für den PC. Der Betrieb eines Modems, sowohl in klassischer Implementierung mit Eigenintelligenz als auch in Form eines Softmodems, gilt als harte Realzeit-Anwendung: Ein Modem benötigt eine ausreichende Wandlerrate um Träger und Modulationstöne zu generieren. Die Basis-Modulation und -Demodulation muss die Verarbeitung eines Datensatzes beendet haben, bevor weitere Daten empfangen oder gesendet werden können. Das Trägersignal wird digital erzeugt; um die Verbindung aufrecht zu erhalten darf es zu keiner Zeit unterbrochen werden oder seine Frequenz zu stark ändern. Dies gilt auch für den Fall, dass eine Verbindung besteht aber gerade keine Daten übertragen werden.

Anfang der achtziger Jahre gab es die ersten externen Modems für PCs zur Übertragung von Daten über die Telefonleitung. Die Geschwindigkeit lag bei 300 Bit/s (Bell 103), welche aufgrund der

FSK-Kodierung[44] mit 1 Symbol/Bit einer Symbolrate (= Baudrate oder Schrittgeschwindigkeit) von 300 Baud entsprach. Die gesamte Kodierung und Dekodierung des Datenstroms wurde im Modem erledigt, welches über die serielle Schnittstelle mit dem PC verbunden war. Die zeitkritischen Prozesse wurden mit speziellen integrierten Schaltungen verarbeitet, da die Leistungsfähigkeit der PC-Hardware hierfür (noch) nicht ausreichend war. In den folgenden Jahren stieg sowohl die Übertragungsgeschwindigkeit der Modems als auch die Leistungsfähigkeit der PC-Prozessoren stetig an. Mitte der neunziger Jahre erreichte die Modemtechnologie annähernd die physikalischen Grenzen der Telefonleitung bei einer Übertragungsgeschwindigkeit von 33,6 KBit/s[45]; die Leistung der Prozessoren stieg jedoch weiter an.

Nun wurde es möglich, die brachliegende Leistung des Hauptprozessors für die Dekodierung und Kodierung der zu übertragenden Daten zu verwenden und auf den im Modem vorhandenen Mikrocontroller oder DSP[46] zu verzichten. Das Softmodem mit Markennamen wie *Winmodem*, *Host Signal Processing (HSP) Modem*, *Contollerless Modem*, *Host Accelerated Modem (HaM)* oder *Rockwell Protocol Interface (RPI) Modem* war geschaffen. Diese Softmodems unterschieden sich in der Konsequenz der Softwareimplementierung. So erledigte beim zuerst erschienenen *Winmodem* von *U.S. Robotics* ein dedizierter DSP weiterhin die Basis-Modulation und –Demodulation (*data-pump function*). Die anderen Aufgaben, wie Fehlerkorrektur, Komprimierung und Protokollausführung (*controller function*) erledigte der PC-Prozessor. Bei den *HSP*- oder *Controllerless*-Modems wurden dann alle Aufgaben in die Software verlagert und vom Hauptprozessor ausgeführt.

Man legte als sinnvolle obere Grenze eine maximale Auslastung von 25 % fest, d.h. im normalen Betrieb eines PCs darf der Anteil der Softmodem-Algorithmen an der Prozessorlast 25 % nicht überschreiten (*worst case*), damit Anwender und Vordergrundapplikation möglichst wenig gestört werden. Auf einem PC mit Intel Pentium Prozessor und Taktraten ab 200 MHz ist es mit optimierten Kodierungsalgorithmen möglich, ein Softmodem für Übertragungsgeschwindigkeiten bis 33,6 KBit/s zu implementieren. Die benötigte Rechenleistung für das Softmodem beträgt ca. 40 MIPS. Die ersten Softmodems, sowohl mit als auch ohne eigenem Mikrocontroller oder DSP, erschienen für das Windows-Betriebssystem. Diese Modems waren als ISA-Karten ausgeführt und besaßen in der *Controllerless*-Version außer den *Codecs* zur Wandlung der analogen Signale nur die diskreten Komponenten zur Anschaltung der Telefonleitung. Alle Algorithmen der Datenaufbereitung liefen im Software-Treiber und damit auf dem PC-Prozessor ab. Die Reduzierung auf wenige preiswerte Bauteile senkte die Produktionskosten und ermöglichte einen günstigeren Verkaufspreis.

Nach der anfänglichen Euphorie gerieten die Softmodems in Verruf, denn sie hatten schwerwiegende technische und konzeptionelle Probleme:

1. Wie in Kapitel 4.1 ausführlich erläutert, eignet sich die PC-Architektur in Verbindung mit den von Endkunden überwiegend eingesetzten Windows-Betriebssystemen nur bedingt für (harte) Realzeitaufgaben. Die Kodierung und Dekodierung der zu übertragenden Daten durch das Softmodem ist zeitkritisch; Unterbrechungen der Datenübertragung werden mit Verbindungsabbrüchen quittiert.

---

[44] FSK: ***F****requency* ***S****hift* ***K****eying*.

[45] Nach dem Shannon-Theorem beträgt die maximale Datenrate der Telefonleitung $C = B \log_2(1+S/N)$, d.h. bei einer Bandbreite B von 3,1 kHz und ca. 39 dB max. Rauschabstand (S/N) ergibt sich als maximale Datenrate ca. 40 KBit/s. Aktuelle 56K Modems arbeiten mit speziellen teildigitalen Dekodierungsverfahren und erreichen ihre maximale Übertragungsgeschwindigkeit von 56 KBit/s nur in Richtung Telefonanbieter → Endkunde. In die Gegenrichtung (*Uplink*) sind maximal 33.6 KBit/s möglich.

[46] DSP: ***D****igital* ***S****ignal* ***P****rocessor*.

2. Das Neuaufsetzen der Übertragung nach Verbindungsabbrüchen ist für den Endbenutzer nicht nur aufgrund des Zeitverlusts ärgerlich, sondern ist auch mit erhöhten Kosten (Telefongebühren) verbunden.
3. Der Software-Treiber ist integraler Bestandteil des Modems. Ohne funktionierenden Treiber kann keine Verbindung hergestellt werden. Falls z.B. bei einem Systemfehler (Festplattendefekt o.ä.) der Treiber beschädigt wird, ist das Softmodem nicht mehr einsatzbereit. Eine Verbindung zum Internet um Software-Treiber herunterzuladen ist nicht mehr möglich (*deadlock*).
4. Die Software ist abhängig vom Hersteller und der Version des Betriebssystems. Bei den kurzen Innovations- und Updatezyklen der Softwareindustrie werden die Kosten für aktualisierte Software-Treiber unkalkulierbar.
5. Der Installationsprozess des Softmodems ist für den Endkunden kompliziert und erfordert vermehrte Hilfestellung durch eine geschulte Hotline. Die hohen Personalkosten hierfür müssen vom Modemhersteller in die Kalkulation miteinbezogen werden und verteuern das Produkt; der Preisunterschied zu klassischen Modems mit eigenem Mikrocontroller schwindet.
6. In vielen Fällen sind vom Kunden installierte, inkompatible Programme oder unsauber programmierte Treiber anderer Hersteller für die Ausfälle des Softmodems verantwortlich. Diese Tatsache ist dem Kunden sehr schwer zu vermitteln und löst auch keines seiner Probleme. Der Ruf des Modemherstellers leidet unter der Unzuverlässigkeit des Gesamtsystems.
7. Bei der Integration der *Codecs* innerhalb des PCs oder direkt auf dem Motherboard gibt es Probleme bei der funktechnischen Zulassung.
8. Durch die Implementierung von notwendigen Funktionen innerhalb der Softwareroutinen des Treibers, ist es nicht mehr möglich, diese Softmodems z.B. an PCs mit dem Linux Betriebssystem anzuschließen. Hierzu ist ein entsprechender Treiber unabdingbar, welcher, wenn er denn verfügbar ist, meist nicht als compilierbarer Source-Code vorliegt.

Man erkennt die vielschichtigen Gründe für das problematische Verhalten der Softmodems, obwohl sie „theoretisch" funktionieren. Aber hier muss, wie in vielen Fällen wenn es sich um zeitkritische technische Prozesse handelt, das Gesamtsystem betrachtet werden. So ist es nicht verwunderlich, dass das von Motorola letztes Jahr (2000) als durchschlagende Neuerung propagierte *Soft-ADSL-Modem* bis heute nicht auf den Markt gekommen ist. Es war für den Einsatz mit Prozessoren ab 750 MHz geplant und benötigt bei Übertragungsraten von 1,5 MBit/s ca. 250 MIPS Rechenleistung. Inzwischen hat Motorola die Entwicklung dieser Softmodems komplett eingestellt und den Bereich verkauft.

Im Bereich der Softmodems für analoge Telefonanschlüsse gibt es inzwischen trotz der geschilderten Schwierigkeiten wieder mehrere Anbieter. Der Trend hin zum PC für unter 250 € erfordert hochintegrierte und kostengünstige Lösungen und zwingt die Hersteller vor allem bei den Hardwarekomponenten zu Kosteneinsparungen. Durch spezielle ASICs mit größeren Zwischenspeichern, Integration der analogen Leitungsanschaltung (Übertrager, Relais, Optokoppler usw.), Spezifizierung des AMR-Standards (*Audio Modem Riser Card*), stromsparende Chiptechnologie und verbesserte Software-Treiber konnten die Softmodems zuverlässiger gestaltet werden und zunehmend Marktanteile gewinnen. Speziell im mobilen Bereich, indem inzwischen leistungsfähige Prozessoren (mit 1 GHz und mehr) eingesetzt werden, der Platz aber nach wie vor stark eingeschränkt und der Stromverbrauch entscheiden ist, finden sich vermehrt integrierte Softmodem-Chipsätze.

Eine vereinfachte Integrationsmöglichkeit für Softmodems wurde mit der Einführung der Intel *Audio Codec Standard*-Schnittstelle (AC97) geschaffen, wie sie sich heute in vielen PC-Chipsätze befindet. Diese in der *South Bridge* integrierte 5-draht Schnittstelle dient zum Anschluss von externen seriellen *Codecs*, welche eine einfache und kostengünstige Realisierung der Audio- und Modemfunktionalität ermöglichen. Der *Modem-Codec* wandelt die vollständig vom PC-Prozessor generierten Daten in Audiosignale um, wie sie ein klassisches Modem bereitstellen würde. Aktuelle AC97 *single-chip Modem-Codecs* enthalten auch die Leitungsanschaltung (AFE[47]).

5.1.1 Intelligente Controller-Komponente als eingebettetes System

Soll die Reaktionszeit auf externe Ereignisse verringert werden, so muss die zusätzliche Hardware-Komponente neben der reinen Vorverarbeitung auch Berechnungen durchführen und auf Grundlage dieser Ergebnisse selbsttätig Entscheidungen treffen und neue Daten oder Befehle ausgeben können. Die Intelligenz stellt ein Mikrocontroller (Mikrocontroller-Komponente) oder DSP (DSP-Komponente) mit eigenem Programm- und Datenspeicher zur Verfügung. Die Algorithmen sind für die jeweilige Anwendung zu erstellen und als Firmware in die Komponente zu laden. Ihre Eigenintelligenz ermöglicht dann, je nach Leistungsfähigkeit, eine Reduzierung der Reaktionszeiten auf bis zu 0,5 µs. Den Parameterbereich zeigt Bild 5.1.

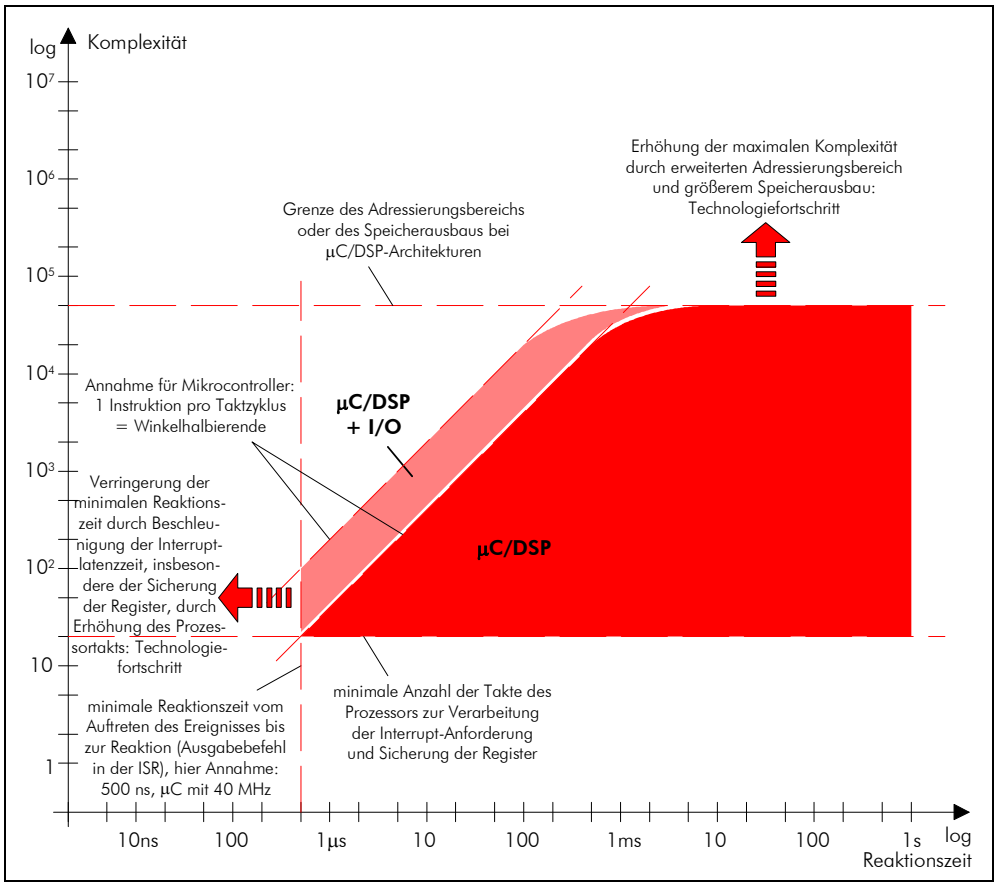


Bild 5.1 Parameterbereich der intelligenten Controller-Komponente (µC/DSP, µC/DSP + I/O)

[47] AFE: ***A**nalog **F**ront **E**nd*.

Mikrocontroller-Komponente

Die Mikrocontroller-Komponente (auch μC-Komponente) verwendet als Recheneinheit einen Mikrocontroller-Baustein und je nach Ausführung zusätzliche Einheiten wie Speicher, Schnittstellen usw. Ein Beispiel für eine μC-Komponente aus der Automatisierung ist die *Slot-SPS*. Die Komponente in Form einer ISA- oder PCI-Steckkarte agiert hierbei als speicherprogrammierbare Steuerung und enthält Prozessor, Speicher, I/O-Schnittstellen zum Anschluss von Sensoren und Aktoren, Feldbus-Schnittstellen und teils einen Anschluss für eine separate Stromversorgung. Diese ermöglicht einen ununterbrochenen Betrieb unabhängig von einem Ausfall des PC-Host-Systems. Kontakt- und Ablaufplan werden mit Software-Werkzeugen auf dem PC erstellt und in die Slot-SPS heruntergeladen. Nach einem *Reset* wird das Programm abgearbeitet und alle Berechnungen und Entscheidungsfindungen lokal ausgeführt. Eine Datenverbindung zum PC-System ermöglicht Monitoring-Funktionen und Wertespeicherung. Idealerweise findet eine Arbeitsteilung bei den durchzuführenden Berechnungen statt. So werden einfache aber reaktionszeit-kritische Berechnungen lokal in der μC-Komponente durchgeführt, komplexe Berechnungen hingegen auf den leistungsstarken Host-Prozessor ausgelagert. Ein Beispiel ist die Realisierung eines PID-Reglers, dessen Parameter mit Hilfe von Host-Prozessor-basierten Adaptionsalgorithmen ständig optimiert werden.

Als Recheneinheit einer μC-Komponente werden vermehrt sogenannte DIMM-PCs eingesetzt. Diese implementieren eine weitestgehend PC-kompatible Umgebung durch Einsatz einer x86-CPU und der entsprechenden Systembausteine. Eine Kosteneinsparung ergibt sich durch Verwendung von preiswerten x86-Entwicklungstools und –betriebssystemen. Man spricht hier vom „PC-im-PC“. Es muss aber eine strenge Auswahl der einzusetzenden Komponenten erfolgen, um nicht hardwarebedingte Verzögerungszeiten hinzuzufügen, wie sie in Kapitel 5.2 für PC-Systeme beschrieben werden.

Ein anderes Einsatzgebiet von intelligenten Controller-Komponenten ist die *Hardware-in-the-Loop*-Simulation (HIL) im Rahmen von *Rapid-Prototyping*-Aufgaben. In der HIL-Technik wird entweder ein simulierter technischer Prozess mit realer Steuerungshardware verbunden, oder ein realer technischer Prozess mit einer Simulation der Steuerungshardware betrieben. Das in beiden Fällen herausragende Merkmal ist die Präsenz real existierender I/O-Schnittstellen. Viele verschiedene Schnittstellenprotokolle können mit programmierten Funktionen in den μC-Komponenten realisiert werden. Für einfache Standardaufgaben werden zusätzliche Ein/Ausgabe-Komponenten ohne Eigenintelligenz eingesetzt (μC + I/O). Der PC-Prozessor übernimmt die komplexen und datenintensiven Berechnungen. Die erzielbare Simulationsgeschwindigkeit hängt von der sinnvollen Aufteilung der Aufgaben auf die verschiedenen μC- und I/O-Komponenten und den PC-Prozessor ab. Das schwächste Glied bestimmt hierbei die erzielbare Leistung.

DSP-Komponente zur Signalverarbeitung

Außer den beschriebenen Mikrocontrollern können auch digitale Signalprozessoren (DSPs) als Recheneinheit eingesetzt werden. Ihre Hauptanwendungsgebiete liegen, wie der Name schon sagt, in der Verarbeitung von Signalen, speziell von Audio- und Videodaten. Hier zeigt sich ihre Stärke im Umgang mit großen Datenmengen. Moderne Kodier- und Komprimier-Algorithmen, z.B. zur Bandbreiten-Reduzierung (MPEG, ADPCM), benötigen hohe Rechenleistungen. Die Programmierung entsprechender Algorithmen wird von angepassten Befehlssätzen unterstützt. Bekannt gewor-

den sind diese Befehlssätze durch MMX[48], SSE[49] (Intel) und 3DNow[50] (AMD), Erweiterungen der x86-Architektur, welche den Desktop-Prozessoren zusätzliche Signalverarbeitungsfähigkeiten hinzufügen. Sie erhöhen die Rechenleistung des Prozessors und helfen Spezialhardware zu ersetzen. Beispielsweise lässt sich die Prozessorlast der in Kapitel 5.1 beschriebenen Soft-Modems durch Verwendung von speziellen MMX-Befehlen um den Faktor 2 verringern.

Spezielle Teilaufgaben der Signalverarbeitung, wie DCT, iDCT[51], MPEG[52], VLE[53] usw. erzeugen auch auf DSPs hohe Rechenlasten. Neue Ansätze zur Integration von Hardware-Erweiterungen für DSPs versuchen aufwändige, immer wieder benötigte, algorithmische Funktionen in Form von Hardware-Logikblöcken zum programmierbaren DSP-Kern hinzuzufügen. Diese erweiterten Funktionalitäten bilden einen Teil des Signalflusses und nutzen die Ressourcen des DSPs. Zur Integration bieten sich in erster Linie international normierte Standards an. Gerade Verfahren für Videosignale liegen voll im Trend. Hier sind beispielhaft die diskrete Cosinus-Tranformation, *Motion Estimation* (ME), *Motion Compensation* (MC) oder das *Variable Length Encoding* zu nennen.

Obwohl diese meist als PCI-Steckkarte realisierten *Multimedia-Beschleuniger* auf DSP-Basis den Host-Prozessor stark entlasten und hohe Datendurchsatzraten erreichen, muss sehr genau analysiert werden, ob mit der Verringerung der Verarbeitungszeiten auch eine Verringerung der Reaktionszeiten einhergeht. Meist sind diese DSP-Komponenten ausschließlich für eine spezielle Anwendung entwickelt und auf maximalen Datendurchsatz optimiert. Sie erlauben oft keine Verlagerung der Entscheidungsfindungen vom Host-Prozessor in den DSP durch den Anwender und sind damit für zeitkritische technische Prozesse nicht geeignet. Es ist also auf vorhandene Programmierbarkeit und entsprechende Unterstützung durch Entwicklungswerkzeuge zu achten. Einige Firmen bieten DSP-Komponenten für verschiedene PC-Schnittstellen an, die über Erweiterungsmöglichkeiten für Eigenentwicklungen verfügen. Software-Werkzeuge und begleitende Bespielprogramme für einen schnellen Einstieg werden mitgeliefert. Eine entsprechende erweiterbare DSP-Karte für den PCI-Bus ist Basis für die in Kapitel 6 beschriebenen Anwendungsbeispiele.

Vergleicht man die DSP-Komponente mit der μC-Komponente, so kann als Vorteil der DSP-Komponente die höhere Rechenleistung und die meist einfachere Architektur (RISC) und damit einhergehend die bessere Vorhersagbarkeit der Realzeiteigenschaften genannt werden. Nachteile sind die begrenzte Anzahl an I/O-Schnittstellen und Peripherieeinheiten, wie Timer, Interrupt-Controller, PWM-Generatoren, ADC, DAC, UART-, SPI- und Feldbus-Schnittstellen, sowie die wenigen frei verfügbaren I/O-Pins.

Die Reaktionszeit, d.h. die Zeit vom Auftritt des externen Ereignisses bis zur Ausgabe der Reaktion, eines PC-Systems unter Verwendung einer zusätzlichen intelligenten Controller-Komponente (μC oder DSP), wahlweise in Verbindung mit einer Ein/Ausgabe-Komponente ohne Eigenintelligenz, setzt sich zusammen aus:

$$t_{R\,\mu C/DSP} = t_{I\,\mu C/DSP} + t_{ISR} + t_{\mu C/DSP} + t_D\ (+\ t_{I/O}) \geq 0{,}5\ \mu s$$

[48] MMX: ***M**ulti-**M**edia e**X**tension*.
[49] SSE: ***S**treaming **S**IMD (Single Instruction, Multiple Data) **E**xtensions*, neue Version: SSE2.
[50] 3Dnow: Multimedia-Befehlssatz von AMD, nicht kompatibel zu MMX.
[51] DCT, iDCT: (***i**nverse) **D**iscrete **C**osine **T**ransformation*.
[52] MPEG: ***M**oving **P**icture **E**xperts **G**roup*, ISO-Arbeitsgruppe zur Standardisierung von digitalen Video-Kompressionsalgorithmen.
[53] VLE: ***V**ariable **L**ength **E**ncoding*.

mit: R = Reaktionszeit, I = Interrupt-Latenzzeit des µCs/DSPs, ISR = Ausführungszeit der Interrupt-Service-Routine, µC/DSP = Ausführungszeit der Anwendung bis zur Ausgabeanweisung, D = Datenübertragungszeit µC/DSP → I/O, I/O = Ausführungszeit der Vorverarbeitung in der I/O-Komponente. Die minimale Reaktionszeit hängt stark von der Leistung des verwendeten Prozessors und der Komplexität der Berechnungen ab. Bei leistungsstarken Prozessoren und wenigen Rechenoperationen können minimale Reaktionszeiten von ca. 0,5 µs erreicht werden.

5.1.2 Programmierbare Logik-Komponente

Die im vorigen Kapitel vorgestellten Mikrocontroller- und DSP-Komponenten erreichen kurze Reaktionszeiten bei geringer und mittlerer Komplexität der Algorithmen. Durch Unterstützung mit speziellen Ein-/Ausgabe-Komponenten können auch Aufgaben höherer Komplexität verarbeitet werden. In einigen Fällen reichen die erzielbaren Reaktionszeiten, respektive der erreichbare Grad der zu verarbeitenden Aufgabenkomplexität nicht aus. Hier kommen die *programmierbaren Logik-Komponenten* zum Einsatz.

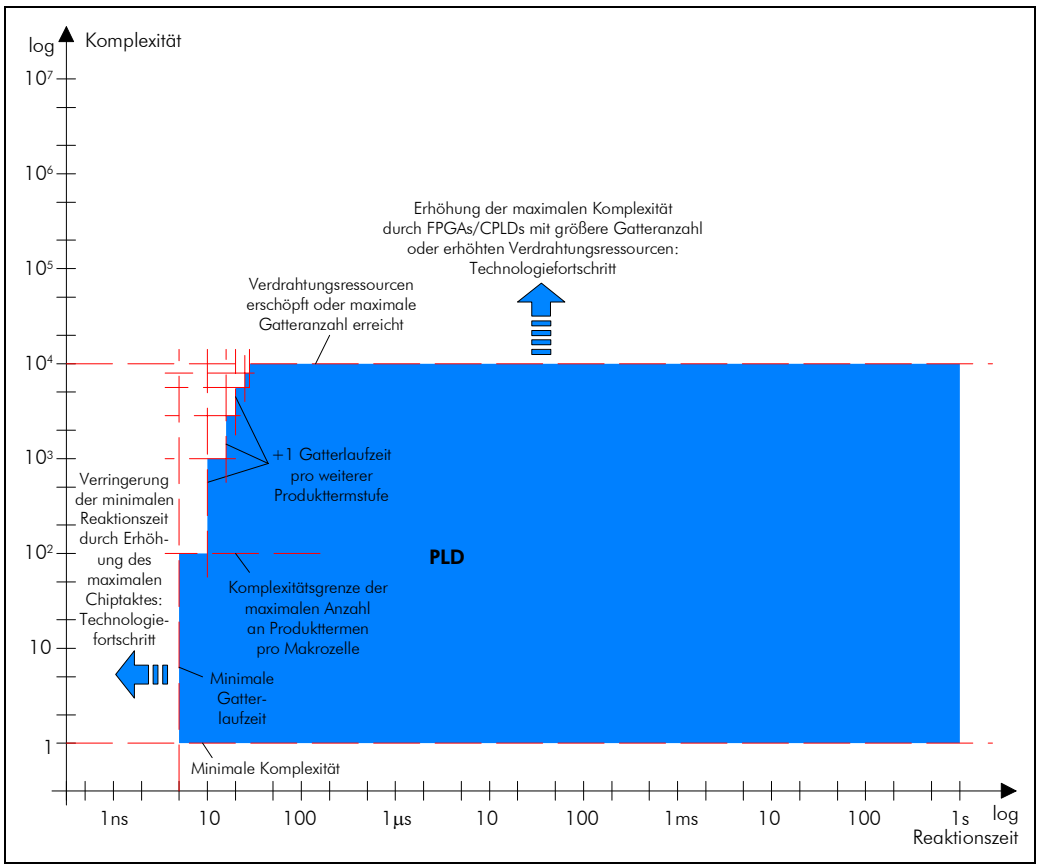


Bild 5.2 Parameterbereich der programmierbaren Logik-Komponente (PLD)

Hauptbestandteile dieser Komponenten sind programmierbare Bausteine mit permanenter oder flüchtiger Programmierung, sogenannte CPLDs oder FPGAs. Sie werden in der Prototypenentwicklung verwendet oder bei geringen Stückzahlen direkt in die Schaltung integriert, und ermöglichen eine fortlaufende Anpassung des Designs. Für die kostengünstige Massenfertigung werden sie durch ASICs ersetzt. Die Flexibilität der schnellen Programmierung wird im Extremfall für das *Reconfigurable Computing* verwendet. Hierbei wird die Rechnerarchitektur während des Betriebs dynamisch an die zu lösende Aufgabe (oder den Befehl) angepasst. Zunehmend werden Hybridschal-

tungen angeboten, die konventionelle Prozessorkerne zusammen mit rekonfigurierbarer Logik auf einem *System-on-a-Chip* (SoC) integrieren. Die erreichbaren Reaktionszeiten der programmierbaren Logik-Komponente in Abhängigkeit von der Komplexität der zu bearbeitenden Aufgaben zeigt Bild 5.2.

Integriert man diese programmierbaren Bausteine auf Steckkarten oder Module zum Anschluss an die PC-Architektur, kann eine Vielzahl von unterschiedlichen Funktionalitäten flexibel realisiert werden. Die Gatterstruktur ermöglicht eine hohe Parallelität; die Herstellung in CMOS-Speicher-Technologie[54] erlaubt hohe Taktraten bei höchster Integrationsdichte. Gleichzeitig schreitet die Entwicklung dieser Technologie sehr rasch voran und ermöglicht die Herstellung von FPGAs mit bis zu 10 Mio. äquivalenten Gatterfunktionen (Xilinx *Virtex II*, 2001).

Die geringen Reaktionszeiten, die genaue Planung von Durchlaufzeiten und die Parallelität ermöglichen hochpräzise Synchronisationsaufgaben, die z.B. bei der Zusammenschaltung mehrerer Komponenten oder bei der Reaktion auf gleichzeitige externe Ereignisse erforderlich sind. So verwenden [Sig98] und [Kra01] spezielle Hochgeschwindigkeitsbusse zur Hardware-Synchronisation mehrerer Komponenten in Realzeit. Ein Einsatzgebiet ist die Verbindung mehrerer DSP-Komponenten, wie sie in HIL-Simulationssystemen eingesetzt werden. Eine weitere Anwendung für programmierbare Logik-Komponenten ist z.B. die Realisierung eines Logik-Analysators als Steckkarte für ein PC-System (Firma Tektronix). Die Logik-Komponente kann bei Auftreten bestimmter programmierter Signalmuster die synchrone Datenerfassung einer Speicheroszilloskop-Karte starten.

Den Vorteilen der programmierbaren Logik-Komponenten stehen einige Nachteile gegenüber. So ist die akkumulierte Verarbeitungszeit durch die Verzögerungszeiten ein oder mehrerer hintereinandergeschalteter Makroblöcke von der Architektur des eingesetzten Bausteins und vom Grad der Parallelisierung des zu implementierenden Algorithmusses abhängig. Kurz, Aufgaben mit einzelnen, unabhängigen Teilen können effektiver implementiert werden als komplexe Aufgaben mit gegenseitigen Abhängigkeiten. [PBC+97] beschreibt die Verwendung eines FPGAs zur Datenerfassung eines Radarsystems. Die Berechnung des gut aufteilbaren *Puls-Pair*-Algorithmusses wird vollständig im FPGA erledigt und ersetzt vier DSPs bei gleichzeitig halbierter Leistungsaufnahme.

Unabdingbar für den Einsatz von programmierbaren Logik-Komponenten sind leistungsfähige und einfach zu bedienende Entwicklungswerkzeuge. Hier gab es viele erfolgreiche Anstrengungen der Software-Hersteller, um die kritischen Schwachpunkte, wie Compilierbarkeit von komplexen Designs, Ressourcenverbrauch bei verdrahtungsintensiven Designs und Zuverlässigkeit der Simulationen zu verbessern.

Die Reaktionszeit, d.h. die Zeit vom Auftritt des externen Ereignisses bis zur Ausgabe der Reaktion, eines PC-Systems unter Verwendung zusätzlichen programmierbaren Logik-Komponente, setzt sich zusammen aus:

$$\mathbf{t_{R\,PLD} = t_{PLD} \geq 5\ ns}$$

mit: R = Reaktionszeit, PLD = Gatterdurchlaufzeit des parallelisierten Algorithmusses. Gut parallelisierbare Prozesse benötigen nur sehr wenige Logikebenen und erreichen minimale Reaktionszeiten von ca. 5 ns.

[54] 0,15 μm, 6 Lagen Metall Prozess (2001).

5.1.3 Probleme bei der Nutzung von Zusatzhardware

Beim Einsatz von Zusatzhardware zur Erweiterung der Realzeitfähigkeit treten mehrere spezifische Probleme auf:

- Missverhältnis der Rechenleistungen
- Stagnierende Übertragungsgeschwindigkeiten
- Hohe Kosten

Missverhältnis der Rechenleistungen

Der erste Aspekt ist das Missverhältnis der Rechenleistungen von PC-Prozessor zu Steuerungs-Prozessor. Die Entwicklung der Prozessoren für Standard-PCs verläuft wesentlich schneller als die der Prozessoren für eingebettete Systeme. Bild 5.3 zeigt einige Typen und deren ungefähre Leistung in *Million Instructions Per Second* (MIPS). Auf der linken Seite finden sich die Desktop-Prozessoren von Intels i386 bis hin zu AMDs aktuellem Athlon. Rechts daneben sind die Leistungsdaten typischer *Embedded*-Prozessoren von der Motorola 683xx-Serie bis zum MPC755 Controller aufgetragen.

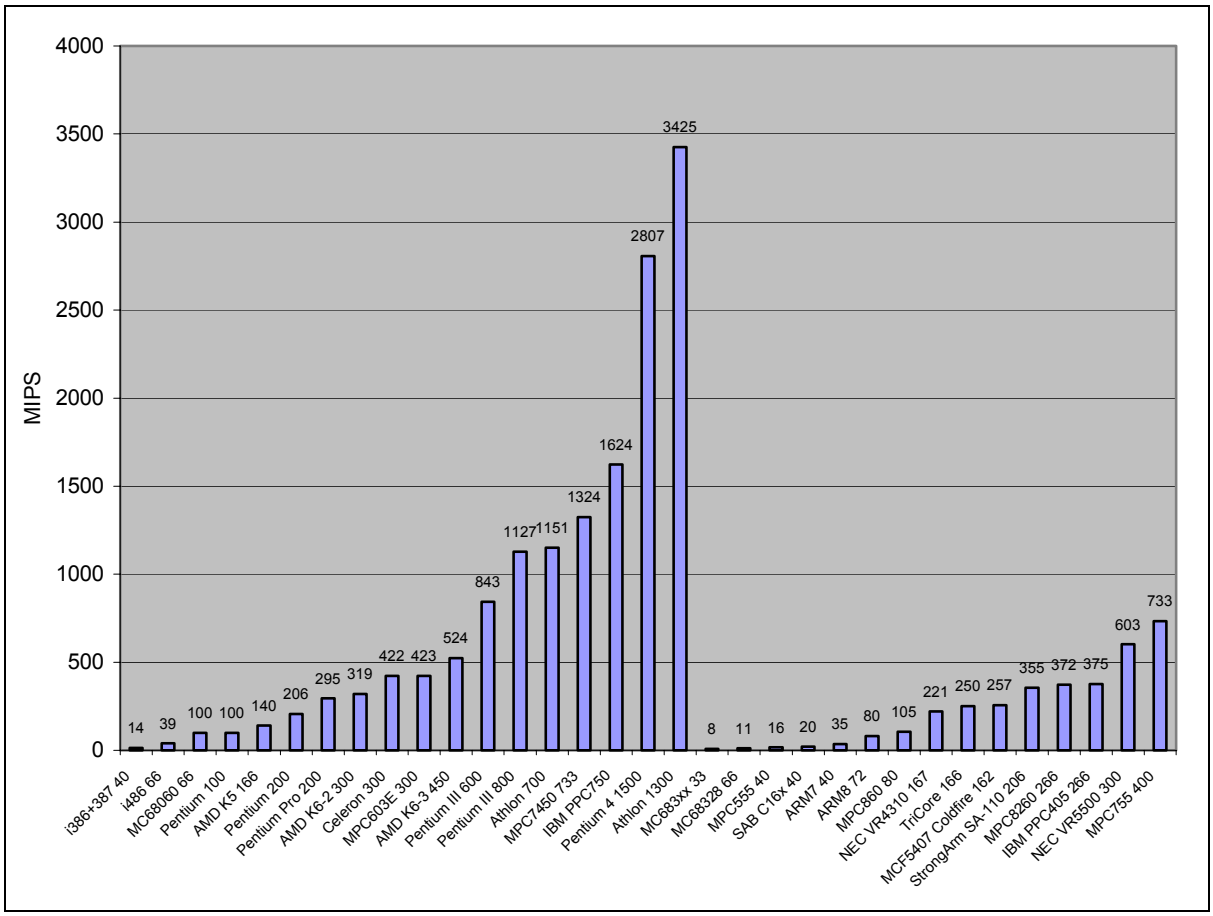


Bild 5.3 Leistungsvergleich: Desktop-Prozessoren – Embedded-Prozessoren

Bei der Aufteilung der Algorithmen muss unbedingt zwischen einfachen, realzeitkritischen Tasks und rechenintensiven, zeitaufwändigen Tasks unterschieden werden. Nur die wenigen realzeitkri-

tischen Aufgaben sollten auf einer leistungsmäßig ausreichenden Zusatzhardware implementiert werden, um die Kosten und die Busbelastung möglichst gering zu halten.

Der Einsatz von DSP-Zusatzkarten für PC-Systeme, insbesondere für den wiederholt vorgeschlagenen Einsatz als Multimediabeschleuniger [BSH98], ergibt ähnliche Probleme. Nur wenige sehr moderne und teuere DPSs (z.B. Texas Instruments TMS320C64xx-Serie) können in die Leistungsbereiche aktueller Desktop-Prozessoren, wie AMDs *Athlon XP* oder Intels *Pentium 4* vorstoßen. Der Hard- und Softwareaufwand zur Erzielung dieser Leistung ist jedoch weitaus größer[55].

Stagnierende Übertragungsgeschwindigkeiten

Der zweite Aspekt betrifft die stagnierenden Übertragungsgeschwindigkeiten (I/O-Leistung) der PC-Datenschnittstellen, wie z.B. des PCI-Busses. Diese steigen durch unterschiedlich hohe Innovationsgeschwindigkeiten bedingt weitaus langsamer an als die Leistungsfähigkeit der Prozessoren. Eine Verbesserung soll durch eine verstärkte Vorverarbeitung in den Komponenten zur Entlastung der vergleichsweise langsamen Schnittstellen erreicht werden [FMO+98]. Die begrenzten Übertragungsgeschwindigkeiten, insbesondere die begrenzte Leistung von Bussystemen bei vielen Teilnehmern, erfordern eine sehr genaue Analyse der Zusatzhardware hinsichtlich ihrer Zukunftssicherheit. Speziell der oft propagierte Einsatz von Multimediabeschleunigern ist zu hinterfragen. Ein Beispiel zeigt der folgende Vergleich:

Vergleich zwischen einer Realisierung mit DSP-Komponente und einer Host-Prozessor Realisierung

Als schnelle Verbindung zwischen einer DSP-Komponente und der PC-Architektur dient meist der PCI-Bus. Mit einer Durchsatzrate von max. 132 MB/s wird er bei datenintensiven Anwendungen zum Engpass. Auch der Rechenaufwand für die Kommunikation ist erheblich. Bis zu 60 % der gesamten Verarbeitungszeit des DSPs entfällt auf die Kommunikation mit dem Host-Prozessor, während die Bearbeitung der Applikation die restlichen 40 % benötigt [Bau95]. Um den Aufwand für die Datenübertragung zu rechtfertigen, ist die Bearbeitung von rechenintensiven Applikationen vorzuziehen. Stellt man nun Ausführungszeiten von Algorithmen auf DSP- und Host-Prozessor gegenüber, zeigt sich der Vorteil des PC-Prozessors. So ist z.B. bei der Dekodierung von MPEG2-kodierten Audiodaten mit einer DSP-Komponente (TMS320C44, 50 MHz) die Streuung der Reaktionszeit nur im Bereich von 1 ms, die absolute Zeit (8,6 – 9,6 ms) ist hingegen fast doppelt so groß wie im schlechtesten Fall (1,7 – 5 ms) der Dekodierung mit dem PC-Prozessor (Pentium 166 MMX) [BSH98].

Vorteile der DSP-Komponenten Realisierung

- Vorhandene, leistungsschwächere PC-Systeme können durch Einsatz von DSP-Komponenten weiterverwendet werden.
- Bei programmierbaren DSP-Komponenten kann eine Entlastung des Host-Prozessors und/oder eine Verringerung der Reaktionszeiten erreicht werden.

[55] Man denke nur an den früheren Texas Instruments DSP TMS320C80 mit 4 parallelen Einheiten, dessen Leistung mit max. 2400 MIPS angegeben war. Der verfügbare Compiler konnte C-Code nur ansatzweise parallelisieren, sodass lediglich Bruchteile der Gesamtleistung zur Verfügung standen.

- Mehrere Funktionalitäten können durch eine leistungsfähige und flexibel programmierbare DSP-Komponente zusammengefasst und damit ersetzt werden.
- Die heutzutage geforderte Audiofunktionalität eines PC-Systems kann durch Einsatz von DSPs mit integrierten ADCs und DACs mit moderater Kostensteigerung bereitgestellt werden.
- Für die Spracherkennung und Bildverarbeitung existieren eine Vielzahl von hoch optimierten Algorithmen für DSPs.
- Die zeitliche Variation (Jitter) bei der Bearbeitung von Algorithmen mit DSP-basierten Hardware-Komponenten ist wesentlich geringer als bei Verwendung des PC-Prozessors.

Nachteile der DSP-Komponenten Realisierung

- Die Kommunikation zwischen Host-Prozessor und DSP ist unökonomisch und rechenintensiv. Die Einführung neuer und schnellerer PC-Schnittstellen geht nur langsam voran.
- Zur Programmierung der DSPs sind spezielle Compiler erforderlich, um deren Leistungsfähigkeit auszuschöpfen.
- Leistungsfähige DSP-Prozessoren sind teuer im Vergleich zu PC-Prozessoren (€/MIPS).
- Die Leistungsfähigkeit von PC-Prozessoren ist wesentlich höher als die der DSPs und der Unterschied vergrößert sich kontinuierlich.
- Die Verwendung von erweiterten Befehlssätzen (MMX, SSE, 3DNow) für PC-Prozessoren beschleunigt Audio- und Videoalgorithmen signifikant.

Hohe Kosten

Der dritte wichtige Aspekt sind die hohen Kosten, welche durch den Einsatz von Zusatzhardware und deren Entwicklung entstehen. Auch hier kann durch die Beschränkung auf zeitkritische Funktionen eine Kostenreduzierung erreicht werden. Dennoch sind zusätzliche Ausgaben für Entwicklungswerkzeuge und entsprechende Schulungen unumgänglich.

5.2 Analyse der Grenzen der PC-Hardware

5.2.1 Architekturbeschränkungen

Die offene Systemarchitektur eines PC-Systems erlaubt es, Komponenten vieler verschiedener Hersteller einzusetzen. Einige Standards sind detailliert beschrieben; andere ermöglichen eine weitgehende Interpretation und lassen Raum für sehr unterschiedliche Implementierungen. Obwohl die Kompatibilität meist gewahrt bleibt, sind doch erhebliche Leistungsunterschiede festzustellen. Gerade bei zeitkritischen Systemen kann die Verwendung von komplexen Beschleunigungsmethoden wie Pipelines, Caches, zusätzlichen Puffern usw. zu großen zeitlichen Abweichungen und damit zur Verletzung der geforderten Realzeitfähigkeit führen.

Prozessoren

In aktuellen PCs kommen moderne Prozessoren mit RISC-Kernen zum Einsatz. Viele Forschungstätigkeiten beschäftigen sich mit den Architekturmerkmalen hinsichtlich *Best case*- und *Worst case*-Ausführungszeit, Fehlerquoten und Strafzeiten. Hier muss sehr genau unterschieden werden, inwieweit sich eine theoretisch ermittelte große Streuung im praktischen Betrieb auswirkt. Es gibt zahlrei-

che Forschungstätigkeiten, welche sich mit der Übertragbarkeit von theoretischen Berechnungen auf reale Messergebnisse beschäftigen. Die Ermittlung der *Worst Case Execution Time* (WCET) ist unabdingbar für die Garantie der zeitlichen Bedingungen von Realzeitsystemen.

Pipeline / Superskalar

In modernen Prozessoren wird die Befehlsbearbeitung in mehrere Teile zerlegt und einer *Pipeline* mit bis zu 20 Stufen (Intel Pentium 4) zugeführt (Bild 5.4). Die verschiedenen Befehlsteile werden unabhängig voneinander quasiparallel verarbeitet[56]. Dies führt zu einer linearen Leistungssteigerung in Relation zur Pipelinelänge und ermöglicht höhere Taktraten des Prozessorkerns. Abhängigkeiten zwischen Daten in den verschiedenen Pipelinestufen erzeugen Konflikte, welche die kontinuierliche Ausführung stören und zu einem sogenannten *pipeline stall* führen. Mit verschiedenen Methoden wird versucht, jede Verzögerung zu verhindern, um die maximale Verarbeitungsleistung des Prozessors zu erhalten. *Superskalar* bedeutet das Vorhandensein von mehreren Recheneinheiten zur parallelen Ausführung von Befehlen. Oft werden mehrere Pipelines zur Versorgung der verschiedenen spezialisierten Rechenwerke eingesetzt.

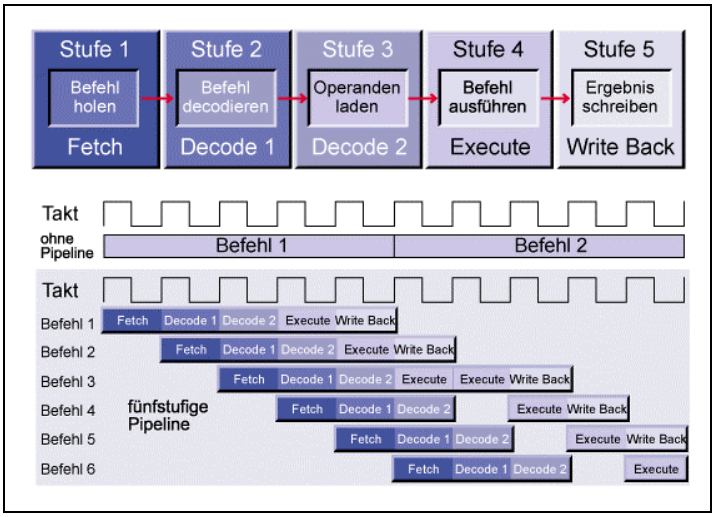


Bild 5.4 Beispiel einer fünfstufigen Pipeline

Ausnahmebehandlung

Treten Ausnahmen (*exceptions*) oder Unterbrechungen (*interrupts*) während der Bearbeitung des Codes auf, so hängt der Einfluss auf die Verzögerung von den gerade benutzten Beschleunigungsmethoden ab. Eine *Out-of-Order*-Bearbeitung muss bei Auftreten der Ausnahme ordentlich abgeschlossen werden, d.h. alle bereits bearbeiteten Befehle, die sich im zeitlichen Verlauf hinter der Ausnahme befinden, müssen verworfen und die Pipeline geleert werden. Bei superskalaren Architekturen muss die Ausnahmebehandlung solange verzögert werden, bis alle ausstehenden Operationen abgearbeitet wurden. Erst dann kann die Bearbeitung des Ausnahmecodes beginnen. Für den realzeitkritischen Einsatz erlauben manche Prozessoren eine Umschaltung auf eine zwangsweise Serialisierung von langen Operationen, um schneller auf Ausnahmen und Unterbrechungen reagieren zu können. Weiterhin haben auch Cachezugriffe einen Einfluss auf die Ausführungszeiten. Man erkennt, dass für die Analyse, der durch die Ausnahmebehandlung auftretenden Effekte, weitere Architekturmerkmale einbezogen werden müssen. Für eine Berechnung des Ausführungs-

[56] Hierbei werden jedoch keine x86-Instruktionen, sondern 72-bit Micro-Operations verarbeitet.

zeit-Verhältnisses kann angenommen werden, dass durchschnittlich eine Ausnahme alle 100 Befehle (1 %) auftritt.

ALU-Befehle

Bei der Bearbeitung von arithmetischen Operationen ergeben sich unterschiedliche Ausführungszeiten. Für eine weitere Analyse wird auf die Ergebnisse von [Dro95] zurückgegriffen. Diese Messungen ergaben, dass im Befehlsstrom einer Applikation (hier SPEC89 Benchmark) durchschnittlich 1,2 % Schiebeoperationen[57], 3 % Multiplikationen und 0,5 % Divisionen vorkommen. Für die Grenzen der Ausführungszeiten können die in Tabelle 5.1 beschriebenen Zeiten eines fiktiven PC-Prozessors als Anhaltspunkt verwendet werden. Natürlich unterscheiden sich die verschiedenen x86-Prozessoren hinsichtlich dieser Werte; gerade moderne RISC-Implementierungen benötigen hier weniger Taktzyklen. Dennoch gelten die *Verhältnisse* der Ausführungszeiten auch für aktuelle Prozessoren.

	Taktzyklen *best case*	**Taktzyklen *worst case***	**Verhältnis**
Schiebeoperation	1	2	1 : 2
Multiplikation	5	15	1 : 3
Division	16	144	1 : 9

Tabelle 5.1 Verhältnis der Ausführungszeiten für unterschiedliche Operationen

Write Combining

Eine spezielle Beschleunigungsmethode wurde von der Firma Intel mit dem PentiumPro Prozessor eingeführt. Das *Write Combining* genannte Verfahren ermöglicht es, einzelne 8-, 16-, oder 32-bit Schreibzugriffe z.B. auf die eingeblendeten Speicherbereiche einer Grafikkarte zu bündeln und in einem *Burst*-Zyklus zu übertragen. Zusammen mit einer weiteren Bündelung der Daten durch den Chipsatz (*CPU-to-PCI write = PCI Posting*) ergeben sich auf einem System mit Intel PentiumPro Prozessor (200 MHz) die Datenübertragungsraten in Tabelle 5.2 [Int98].

Prozessor / System Konfiguration	**Typische Datenübertragungsrate [MB/s]**
CPU → PCI	8
CPU → PCI (+ PCI Posting)	20
CPU → PCI (+ Write Combining)	40
CPU → PCI (+ PCI Posting + Write Combining)	>100

Tabelle 5.2 Datenübertragungsraten mit *Write Combining* und *PCI Posting*

Ein Intel Pentium System mit einem Intel Chipsatz der 430xx-Reihe erreicht dagegen nur eine maximale Übertragungsrate von ca. 70 MB/s. Auch AMD integriert im Athlon Prozessor die *Write Combining*-Methode zusammen mit vier *„aggressively programmed"* 64-bit Puffern. Ein weiterer Geschwindigkeitszuwachs ergibt sich dort durch die Möglichkeit auch *Out-of-Order*-Schreibzugriffe auszuführen.

Trotz der hohen Leistungszunahme durch das *Write Combining* darf man nicht vergessen, dass Zugriffe für einen optimierten Ablauf sowohl umsortiert als auch verzögert werden können. Beim

[57] *Shift operations.*

Schreiben von Daten in den Bildspeicher einer Grafikkarte mag das Verfahren keine Nachteile aufweisen, beim Einsatz des PCs zur Steuerung realzeitkritischer Prozesse kann es jedoch fatale Auswirkungen haben. Eine Verifikation ist deshalb auch in diesem Fall unerlässlich. Im Zweifelsfall darf das *Write Combining* nicht verwendet werden.

Architekturmerkmale

Die nachfolgende Tabelle 5.3 zeigt mittlere Werte für Fehlerquote (*miss rate*) und Strafzeit (*penalty*) einiger Architekturmerkmale von x86-Prozessoren [Dro95][BD97].

Architekturmerkmal	**Fehlerfall**	**Fehlerquote (*miss rate*)**	**Strafzeit (*penalty*) [*cycles*]**
Zugriffe auf den TLB	*No entry*	0,01 – 1 %	20 – 100
Sprungvorhersage	*Wrong prediction*	10 – 15 %	10 – 15
Speicherhierarchie			
1. Zugriff auf den L1-Cache	*L1-Cache miss*	1 – 10 %	5 – 10
2. Zugriff auf den L2-Cache	*L2-Cache miss*	1 – 2 %	50 – 100
3. Zugriff auf den Hauptspeicher	*Page fault*	0,0001 – 0,001 %	> 1.000.000

Tabelle 5.3 Fehlerquoten und Strafzeiten der Architekturmerkmale

Zugriffe auf den Translation-Lookaside-Buffer

Betriebssysteme mit Multitasking-Unterstützung verwenden meist die Speicherschutzmechanismen der im Prozessor integrierten *Memory Management Unit* (MMU). Diese stellt jedem Prozess ihren eigenen, virtuellen Adressraum zur Verfügung. Die Übersetzung in physikalische d.h. reale Adressen geschieht transparent während der Speicherzugriffe. Durch Lesen und Schreiben von Einträgen in mehrstufigen Umsetzungstabellen werden entsprechende Beziehungen zwischen den Adressräumen geschaffen. Um Zugriffe auf die Tabellen zu beschleunigen, existiert ein schneller Zwischenspeicher, der *Translation Lookaside Buffer* (TLB). Kann eine Umsetzung in diesem Zwischenspeicher nicht gefunden werden, muss ein Zugriff auf die Umsetzungstabelle erfolgen. Dieser Zugriff erfordert aufgrund des notwendigen Interrupts, der mehrfachen Speicherzugriffe und in Abhängigkeit von der implementierten Umsetzungsmethode und Anzahl der Tabellenstufen, 50 – 100 zusätzliche Prozessorzyklen. In realen Anwendungen beträgt die durchschnittliche Fehlerquote (*miss rate*) durch nicht vorhandene TLB-Einträge ca. 0,1 – 1 % [JJ01].

Sprungvorhersage

Die vielen Pipelinestufen müssen für einen optimalen Ablauf immer gut gefüllt sein. Ein Problem entsteht bei Sprüngen aufgrund bedingter Verzweigungen. Deshalb versucht man mit ausgeklügelten Prädiktionsmechanismen, wie vorauseilendem Sprungzielspeicher (*Branch Target Buffers* (BTB)) oder einer spekulativen Sprungvorhersage (*Branch Prediction*), die durch eine Verzweigung entstehenden Verzögerungen zu minimieren. Die Sprungvorhersage erzielt im Befehlsstrom einer „normalen“ Anwendung eine Trefferrate von ca. 85 % (Intel Pentium) bis 90 % (Intel Pentium III) [Pat01].

Speicherhierarchie

Moderne Prozessorarchitekturen besitzen eine Speicherhierarchie mit besonders schnellen, lokalen Zwischenspeichern (*Caches*) für Daten und Befehle. Erfolgt ein Zugriff auf eine Adresse werden die

Daten aus dem Speicher in den Prozessor geladen. Gleichzeitig wird eine Kopie dieser Daten im Zwischenspeicher abgelegt. Bei wiederholtem Zugriff auf dieselbe Speicherzelle wird jetzt nur die Kopie aus dem lokalen Zwischenspeicher geladen. Man spricht von einem *Cache Hit*. Die wesentlich höhere Zugriffsgeschwindigkeit (oft mit Prozessortakt) in Verbindung mit sehr breiten Datenbussen von bis zu 256-bit führt zu einer enormen Leistungssteigerung. Ist das Datum dagegen nicht im Cache vorhanden oder ungültig, so muss ein kompletter Zugriff auf den vergleichsweise langsamen Hauptspeicher erfolgen. Es entsteht ein *Cache miss*. Je nach Lokalität der Daten und Befehle, der implementierten Cache-Strategie und der Anzahl der Ladeoperationen ergibt sich eine Fehlerquote von 1 – 10 % mit einer Strafzeit von 5 – 100 Zyklen. Ist das Datum auch im Hauptspeicher nicht vorhanden, liefert die MMU einen Seitenfehler (*page fault*) und die fehlende Speicherseite wird von der Festplatte (oder aus dem Festplatten-Cache) nachgeladen. Dies hat eine Strafzeit von über 1 Mio. Zyklen zur Folge.

Chipsätze

Moderne Chipsätze von Standard-PCs übernehmen in immer stärkerem Maße die optimale Verteilung von Daten an die verschiedenen Systemkomponenten. Durch größere Zwischenspeicher werden Datenpakete gebildet, um die Effektivität der Busse durch leistungsfähige *Burst*-Zugriffe zu steigern und die Übertragungszeiten zu verkleinern. So verfügen aktuelle Chipsätze über fünf Puffer für die verschiedenen Flussrichtungen:

- CPU-to-DRAM write buffer (*Double word merging, Burst merging*)
- CPU-to-DRAM read buffer
- PCI-to-DRAM write buffer (*post*)
- PCI-to-DRAM read buffer (*prefetch*)
- CPU-to-PCI write buffer (*post*)

Die Größe der Puffer in [Bytes] unterscheiden sich je nach Chipsatz und Hersteller (Tabelle 5.4).

	CPU-to-DRAM write	**CPU-to-DRAM read**	**CPU-to-PCI write (post)**	**PCI-to-DRAM write (post)**	**PCI-to-DRAM read (prefetch)**
Intel 430 FX	32	-	16	48	-
Intel 430 TX	40	-	20	72	40
Intel 430 HX	64	-	24	88	80
Intel 430 VX	32	-	20	72	40
Intel 450 NX	?	?	?	192	64
Intel 450 GX	128	128	128	128	128
SiS 5591/92	64	32	64	64	64
VIA Apollo P6	128	128	128	128	128
VIA Apollo VP1	128	?	24	256	128
VIA Apollo VP2	128	?	24	256	128
VIA Apollo Pro	128	32	20	192	64
VIA Apollo Pro+	128	128	64	192	64
VIA KX133	256	256	128	128	64

Tabelle 5.4 Puffergrößen in Chipsätzen verschiedener Hersteller

Besonderer Augenmerk muss auf die Funktionsfähigkeit der Puffer und deren Aktivierung im BIOS gelegt werden. So können aufgrund eines Fehlers im A2-Stepping des Bausteins 82450 OBP (Intel 450 GX Chipsatzes, *Orion*) die *CPU-to-PCI write* Puffer nicht benutzt werden; die Datenrate bei Schreibzugriffen auf den PCI-Bus erreicht damit nur 4 MB/s.

Auch bei der Anbindung des IDE-Busses durch die Intel *South Bridge* PIIX4E (82371AB PCI-to-ISA/IDE Chip) treten Geschwindigkeitsprobleme auf. [Wor99] hat eine maximale Übertragungsrate von max. 21 MB/s beim Lesen und 28 MB/s beim Schreiben ermittelt. Die Ursache liegt in einem mit nur 64 Byte zu klein dimensionierten Puffer (*speed matching FIFO*). Der PIIX4E Chip beginnt die Datenübertragung auf dem IDE-Bus, sobald er über den PCI-Bus 32 Bytes vom Prozessor empfangen hat. Danach beendet der Prozessor den PCI-Bus-Zyklus. Der PIIX4E fordert sofort weitere 32 Bytes an. Er empfängt diese umgehend, was lediglich zu einer kurzen Unterbrechung im IDE-Datenstrom führt. Daraufhin schließt der PIIX4E den PCI-Zyklus ab und beendet die Übertragung. Erst nach einer Wartezeit von 400 ns fordert er weitere Daten an. Dies bedingt eine Verzögerung von ca. 1 μs nach der Übertragung von jeweils 64 Bytes. Die maximale Datenübertragungsrate wird somit durch eine ungeschickte Auslegung des Puffers begrenzt und zusätzliche Verzögerungen werden hinzugefügt.

Systembusse

Bei der Analyse der Grenzen der PC-Architektur stellt der *Systembus* eine wichtige Komponente dar. Als *Systembus* wird die Verbindung zwischen den Chipsatzbausteinen, insbesondere zwischen dem Controller für Prozessor- und Speicheranbindung (*North Bridge*) und dem Controller für PCI-Bus und Peripheriegeräte (*South Bridge*) bezeichnet.

Systembus ist sehr oft der PCI-Bus. Hierbei ist die *South Bridge* als normaler PCI-Busteilnehmer angeschlossen. Diese sehr einfach und kostengünstig zu realisierende Struktur hat einen wesentlichen Nachteil. Die vorhandene Bandbreite muss unter allen PCI-Geräten aufgeteilt werden. In modernen *South Bridges* sind verschiedene Komponenten mit unterschiedlichen Datenübertragungsraten integriert (Tabelle 5.5).

Komponente	Anzahl der Ports	Datenrate pro Port
USB Controller (v1.1)	6	1,5 MB/s
IDE Controller (Ultra DMA/100)	4	ca. 40 MB/s
ISA-Bus (Audio, BIOS, Tastatur, Seriell, Parallel, Floppy, PIC)	1	5 MB/s
System Management Bus	1	12 KB/s
Summe ca.		174 MB/s

Tabelle 5.5 Datenraten der Komponenten in der *South Bridge*

Betrachtet man die Summe der Datenraten der verschiedenen Komponenten, erkennt man, dass schon bei Anschluss von drei bis vier modernen IDE-Festplatten der PCI-Bus in die Sättigung kommt. In der Intel *South Bridge* 82371AB (PIIX4) ist ein 82C59-kompatibler Interrupt-Controller integriert, welcher am ISA-Bus angeschlossen ist, und bei Auftreten eines Interrupts den Interrupt-Typ[58] auf den Datenbus legt. Da die Daten des ISA-Busses vom Chipsatz über den PCI-Bus zum

[58] Dies ist nicht der Interrupt-Vektor.

Prozessor transferiert werden, können bei hoher Belastung des PCI-Busses Verzögerungen der Interrupt-Weiterleitung auftreten. Diese beeinflussen die Realzeitfähigkeit der PC-Architektur negativ.

Eine Verbesserung bringen neue Verfahren der Inter-Chipsatz-Kommunikation. Intel setzt in den neuen Chipsätzen der Serie i8xx auf die (patentierte) *Hub Interface Architecture* mit dezidierter Punkt-zu-Punkt-Verbindung und einer maximalen Übertragungsrate von 266 MB/s [Int99/2]. Auch VIA stellt mit seiner neuen 266 MB/s schnellen *V-Link Hub Architecture* ausreichend Übertragungskapazität zur Verfügung, um Daten ohne Verzögerung zwischen CPU und Peripherie zu übertragen [Via01]. AMD hat mit *HyperTransport* (ehemals *Lightning Data Transport,* LDT) eine mit maximal 6,4 GB/s sehr schnelle und skalierbare Punkt-zu-Punkt-Verbindung entwickelt [Api00].

Anbindung des Hauptspeichers

Die Anbindung des Hauptspeichers an den Prozessor sollte so leistungsfähig wie möglich sein, unabhängig davon, ob es sich um eine zeitkritische Anwendung handelt. Der Hauptspeicher wird bei aktuellen PC-Architekturen durch den Speichercontroller der *North Bridge* gesteuert. Hier kommen zur Leistungssteigerung mehrere Zwischenpuffer, ähnlich den *Caches* im Prozessor, zum Einsatz. Der Hauptspeicher wird aus Kosten- und Kapazitätsgründen mit dynamischen RAM-Bausteinen realisiert. Diese benötigen in zyklischen Zeitabständen eine Auffrischung (*Refresh*) ihrer Speicherzellen. Bei aktuellen SDRAM-Bausteinen wird im aktiven Betrieb ein *Auto Refresh* (ähnlich dem *CAS-before-RAS* (CBR) *Refresh*) durchgeführt. Hierbei triggert der Speichercontroller mit mehreren Signalen (RAS und CAS *low*, CKE und WE *high*) eine separate Auffrischungslogik innerhalb der Speicherbausteine. Diese erneuern daraufhin selbständig den Inhalt einer Speicherspalte. Der mit den EDO-Speicherbausteinen eingeführte *Hidden Refresh* wird wegen der fehlenden Zugriffspausen bei hohen Busgeschwindigkeiten nicht mehr verwendet. Auch RAMBUS-Bausteine sind aus dynamischen RAM-Zellen aufgebaut und müssen zyklisch aufgefrischt werden. Hierzu wird das `REFA`-Kommando an den Speicherchip gesendet.

Das maximal zulässige Auffrischungsintervall hängt von der Speichertechnologie, Speichergröße und der Organisation der Speicherzellen ab. Typische Werte zeigt Tabelle 5.6.

Technologie	Größe	Organisation	Zugriffszeit	*Refresh*-Intervall	*Refresh*-Rate [µs]
FPM / EDO	16 MBit	1024 x 1 KBit x 16	50 ns / 20 MHz	1024 cycles, 16 ms	15,625
SDRAM PC133	128 MBit	4 banks x 4 MBit x 8	7ns / 143 MHz	4096 cycles, 64 ms	15,625
SDRAM PC133	256 MBit	4 banks x 8 MBit x 8	7ns / 143 MHz	8192 cycles, 64 ms	7,812
DDR SDRAM PC266	128 MBit	4 banks x 4 MBit x 8	7ns / 143 MHz	4096 cycles, 64 ms	15,625
DDR SDRAM PC266	256 MBit	4 banks x 8 MBit x 8	7ns / 143 MHz	8192 cycles, 64 ms	7,812
RDRAM PC800 ECC	144 MBit	32 banks x 256 KBit x 18	2,5 ns / 400 MHz	16384 cycles, 32 ms	1,953
RDRAM PC800 ECC	288 MBit	32 banks x 512 KBit x 18	2,5 ns / 400 MHz	16384 cycles, 32 ms	1,953

Tabelle 5.6 Auffrischungsintervalle gängiger Speichertechnologien

Der *Refresh* des Speichers kann bei Realzeitsystemen zu zusätzlichen Verzögerungen führen [Dro95]. Will eine Applikation während eines gerade stattfindenden *Refresh*-Zyklusses auf den Speicher zugreifen, so muss der Speichercontroller die Ausführung für einige Takte blockieren. Eine Beeinflussung des hochintegrierten Controllers durch den Anwender ist bei der PC-Architektur nicht vorgesehen. Der Verzögerungseffekt ist zwar im Verhältnis zu anderen hardwarebedingten Einflüssen klein und wird durch nachfolgend beschriebene Methoden weiter verringert, darf aber bei sehr zeitkritischen Systemen nicht vernachlässigt werden.

In der PC-Architektur gilt die Speicheranbindung als eine der leistungskritischen Komponenten, was den Begriff des *Speicher-Flaschenhalses* (*memory bottleneck*) geprägt hat. Es wird daher versucht, durch die im Folgenden beschriebenen Maßnahmen die Geschwindigkeit der Speicherzugriffe und der Auffrischung zu erhöhen und die Verzögerungen während eines *Refreshs* bei gleichzeitigem Zugriff zu minimieren:

Smart refresh — Es werden nur die Speicherbänke aufgefrischt, die tatsächlich mit Speicher bestückt sind.

Refresh queue — Eine Warteschlange für *Refresh*-Anforderungen wird implementiert und mit niedrigster Priorität abgearbeitet. Erfolgt zum Zeitpunkt einer Anforderung gerade ein Speicherzugriff, wird die Anforderung in der Warteschlange abgelegt und zum nächstmöglichen Zeitpunkt ausgeführt. Ist die Warteschlange voll, bekommt sie höchste Priorität und die Auffrischungen werden unmittelbar nach dem laufenden Zugriff ausgeführt. Diese zeitweilige Vergrößerung des *Refresh*-Intervalls muss vom Speicher unterstützt werden. Die Länge der Warteschlange ist abhängig vom Chipsatz: 4 (Intel 440LX), 7 (Intel i810) oder 1024 (Intel i815EP).

Burst refresh — Wenn die *Refresh*-Warteschlange voll ist, können einige Chipsätze auch einen *Burst* von Auffrischungsanforderungen abarbeiten (Intel 82100).

Concurrent refresh — Verzögerungen werden verhindert, wenn man den *Refresh* während Datenübertragungen zwischen Prozessor und AGP- oder PCI-Bus ausführt.

Memory interleaving — Zur Steigerung der Bandbreite ermöglichen einige Chipsätze einen verschachtelten Speicherzugriff. Hierbei werden Daten abwechselnd von mehreren Speicherbänken gelesen. In der Zwischenzeit können sich die anderen Speicherbänke erholen (*precharge time*). Es existieren 2- und 4-fach verschachtelte Architekturen, welche einen Speicherausbau mit entsprechend vielen gleichartigen Speicherbänken erfordern. Das 4-fach *Interleaving* (32 Bytes) des Intel 450 NX Chipsatzes in Bild 5.5 entspricht genau der *Cache-Line* des unterstützten Intel Pentium II Xeon Prozessors. Das *Interleaving* lässt sich auf Speichermodulebene weiterführen. Werden z.B. zwei Speichermodule mit je 4 Speicherbänken verwendet, kann die Verschachtelung der Daten auch zwischen den beiden Modulen stattfinden (*Card-to-Card Interleaving*).

Address bit permuting — Die *Permutation* erhöht ähnlich dem *Interleaving* die Wahrscheinlichkeit, dass Zugriffe auf bereits erholte (vorgeladene) Speicherbausteinen treffen.

Dies geschieht durch Verteilen der Daten mit einer Granularität in der Größe der *Cache-Line* über zwei oder vier Speicherbänke (Bild 5.6). Die unteren Adressbits werden für die Bankauswahl genutzt.

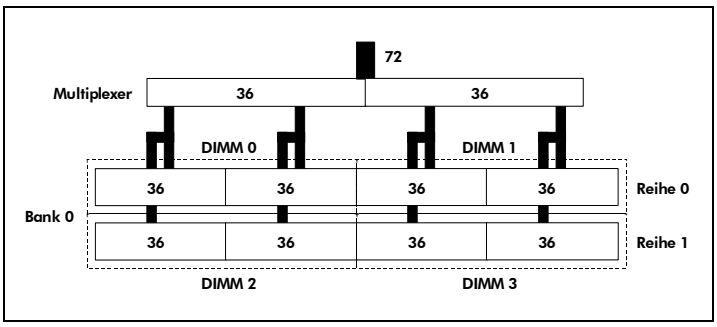


Bild 5.5 4-fach Interleaving mit 72-bit ECC-Speichermodulen

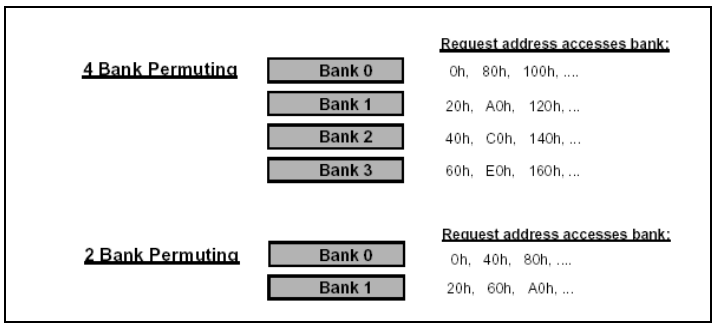


Bild 5.6 Adress-Bit Permutation

Ein Problem tritt bei PC-Systemen mit hoher Speicherbestückung auf. Eine Auffrischung aller Speicherbänke zum gleichen Zeitpunkt würde zu einer Überlastung der Stromversorgung führen, da die Stromaufnahme während des *Refresh* fast doppelt so hoch ist wie im normalen Betrieb. Um diesem Vorzubeugen und gleichzeitig eine Reduzierung der Störstrahlung zu erreichen, ermöglichen viele Chipsätze einen *staggered refresh*, bei dem die Auffrischung der verschiedenen Bänke in kurzen Abständen nacheinander erfolgt (z.B. Intel 450 GX). Laut Intel kann sich jedoch mit dieser Methode die Wahrscheinlichkeit einer Zugriffskollision erhöhen.

Die Forderung nach möglichst stromsparenden Komponenten macht auch vor dem Hauptspeicher nicht halt. Die aktuellen SDRAM-Bausteine bieten für den *Suspend*-Zustand einen integrierten automatischen *Refresh*-Modus (*Self Refresh*) an, der keine Steuerung durch den Speichercontroller benötigt. Speziell für mobile Systeme kann das Deaktivieren des Hauptspeichers auch dynamisch erfolgen. Für realzeitkritische Anwendungen ist diese Betriebsart zu vermeiden, da das „Aufwecken" der Bausteine mehrere zusätzliche Zyklen dauert und somit eine merkliche Zugriffsverzögerung entsteht. Für eingebettete Systeme gibt es spezielle *low-power* Speicherchips, deren *Refresh*-Intervall 128 ms (bei 1024 Zyklen) beträgt.

Interrupt-Verarbeitung

Die Unterbrechungen (*interrupts*) in einem PC werden von zwei kaskadierten Interrupt-Controllern (Intel PIC 8259) entgegengenommen. Diese signalisieren dem x86-Prozessor über die INT-Leitung die Unterbrechungsanforderung und legen den zugehörigen Interrupt-Typ auf den (ISA-)Datenbus. Seit mehreren Jahren werden keine separaten PIC-Bausteine mehr verwendet, sondern die Funk-

tionalität ist im Chipsatz in der *South Bridge* integriert. Es gibt 16 Interrupt-Eingänge, von denen nur ein Teil frei verfügbar ist. Gerade Systeme für die Industrieautomatisierung bestehen aus zahlreichen Komponenten, wie Feldbus- oder Schnittstellenkarten, die jeweils einen Interrupt-Eingang benötigen. Steht kein freier Interrupt zur Verfügung müssen Interrupts gemeinsam benutzt werden (*interrupt sharing*). Beim Auftreten eines Interrupts müssen alle zugeordneten Treiber nacheinander aufgerufen werden [Mic99]. Dadurch erhöht sich die Interrupt-Latenzzeit, die Zeit welche von der Unterbrechungserkennung bis zum Beginn der Unterbrechungsbehandlung verstreicht. Eine weitere Verzögerung ergibt sich, da der PIC als virtuelle Komponente am ISA-Bus der *South Bridge* realisiert ist. Die Übertragung des Interrupt-Typs zum Prozessor erfolgt über den ISA-Bus, *South Bridge*, Systembus (PCI-Bus oder Hub-Struktur) und *North Bridge*. Die spezifischen Verzögerungszeiten (Busarbitrierung, Bandbreite, Konflikte) aller Komponenten summieren sich auf ca. 5 μs (Intel PentiumPro System).

Die Problematik der gemeinsamen Interrupts tritt speziell in Systemen mit mehreren PCI-Komponenten auf. Hier müssen alle Interrupts auf nur vier Interrupt-Eingänge (INTA-D) verteilt werden. Wenn nun z.B. die Grafikkarte und die Antriebssteuerung denselben Interrupt benutzen, müssen die Treiber beider Komponenten alle Regeln des Interrupt-Sharings vorschriftsmäßig implementiert haben. Leider ist dies sehr häufig nicht der Fall; es kommt zu gegenseitigen Blockierungen.

Eine andere potentielle Quelle von unerwünschten Verzögerungszeiten auf PC-Systemen, ist der *System Management Interrupt* (SMI). Dieser wird vollständig durch die Hardware bearbeitet und ist für Betriebssystem und Anwendungen unsichtbar. Der SMI ist ein nicht maskierbarer Interrupt, der andere gerade aktive Interrupts unterbricht und verdrängt (*preemption*) und den Prozessor in den *System Management Mode* (SMM) versetzt. Die darauf aufbauenden Funktionalitäten werden, je nach *Motherboard*-Hersteller und BIOS-Entwickler, unterschiedlich implementiert. Eine allgemeine Aussage zur Häufigkeit und Länge des SMI kann daher nicht angegeben werden.

APIC- und I/O-APIC-Architektur

Eine Weiterentwicklung der Interrupt-Architektur stellt die Kombination aus prozessor-internem *Advanced Programmable Interrupt Controller* (APIC) und externem I/O-APIC dar. Die Kommunikation findet über einen separaten *Interrupt Controller Communications Bus* (ICC), auch APIC-Bus, statt (Bild 5.7). Er ist ein 3-Draht Bus mit zwei Datenleitungen (APICD[0..1]) und einer Taktleitung (APICCLK). Die Busfrequenz beträgt 16,6 MHz.

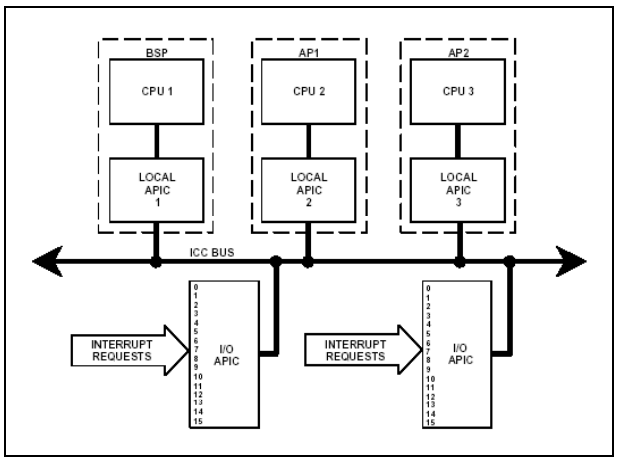


Bild 5.7 APIC – I/O-APIC Konfiguration

In einem Multiprozessorsystem können vier verschiedene Interrupt-Sequenzen über den APIC-Bus übertragen werden. Tabelle 5.7 zeigt eine Auflistung mit den benötigten Übertragungszeiten.

Interrupt-Sequenz	APIC Bustakte	Übertragungs-zeit	Richtung
EOI	14	0,84 µs	EOI from APIC → I/O-APIC to reset IRR bit
Short	21	1,26 µs	Int. from I/O-APIC → APIC with focus
Lowest	33	1,98 µs	Int. from I/O-APIC → APIC with no focus
Remote	39	2,34 µs	APIC reads register of another local APIC

Tabelle 5.7 Interrupt-Sequenzen des APIC-Busses

Der Einsatz der I/O-APIC-Architektur beseitigt die starren Grenzen der historisch gewachsenen Interrupt-Architektur. Eine Verringerung der Interrupt-Latenzzeit auf $< 3\ \mu s$ und eine Verbesserung des Interrupt-Verhaltens ergibt sich durch:

- Verlagerung der Übertragung des Interrupt-Typs auf einen eigenen Bus → Entlastung der Systembusse
- Erhöhung der Anzahl an verfügbaren Interrupts → keine mehrfach verwendeten Interrupts und dadurch Vermeidung von Treiberkonflikten
- Verteilung der Interrupt-Verarbeitung (bei Multiprozessorsystemen) → Lastverteilung

Der APIC unterstützt drei Modi: *PIC-Mode*, *Virtual wire-Mode* und *Symmetrischer Mode*. Im PIC-Mode werden der I/O-APIC und alle APICs der Prozessoren deaktiviert. Ein klassischer PIC (oder eine PIC-Emulation der *South Bridge*) übernimmt die Interrupt-Verarbeitung. Im *Virtual wire*-Mode fungiert der ICC-Bus zwischen I/O-APIC und dem APIC des ersten Prozessors lediglich als transparenter Übertragungsweg für die Unterbrechungsanforderung und den Interrupt-Typ. Erst der symmetrische Mode nutzt I/O-APIC und prozessorinternen APIC zur verteilten Interrupt-Verarbeitung. Die ersten beiden Modi sind DOS-kompatibel und werden für den Boot-Vorgang benötigt. Das Betriebssystem schaltet dann in den Symmetrischen Mode. Es gibt außer der von Intel patentierten APIC-Implementierung noch den *OpenPIC*-Vorschlag und eine weitere Implementierung der Firma Cyrix. In Hardware verfügbar und von aktuellen Betriebssystemen unterstützt wird jedoch nur die Intel APIC-Version.

Aufgrund der Geschwindigkeitsvorteile der APIC-I/O-APIC-Kombination unter Windows 2000 wird eine Integration dieser modernen Interrupt-Verarbeitung in alle zukünftigen PC-Systeme gefordert [Mic99]. Die Funktionsweise des Windows-Kernels (siehe hierzu Kapitel 4.1.2) basiert auf einer Priorisierung mit Interrupt-Ebenen (*Interrupt Request Levels* (IRQL)). Sollen Interrupts gesperrt werden, muss das Betriebssystem die zu blockierenden Interrupts im PIC maskieren. Diese Daten müssen nun den ganzen Weg vom Prozessor bis zum ISA-Bus zurücklegen. Im normalen Betrieb führt Windows 2000 einige 100 Interrupt-Ebenenwechsel pro Sekunde durch. Dagegen erfordert der Ebenenwechsel in einem APIC nur einen einzigen `mov`-Befehl zur Änderung des *Task Priority Registers*.

Die meisten Chipsatzhersteller integrieren die I/O-APIC-Funktionalität in ihren aktuellen Produkten. So gibt es z.B. bei dem neuen Multiprozessorchipsatz 760MP von AMD einen in der *South Bridge* 766 integrierten I/O-APIC, der zum Intel-Baustein (i82093) registerkompatibel ist. Auch die Firmen ServerWorks und VIA produzieren Chipsätze mit integriertem I/O-APIC. Die vielen Vorteile der APIC-I/O-APIC-Kombination machen den Einsatz auch in Einzelprozessorsystemen sinnvoll.

Verringerung der Interrupt-Häufigkeit

Eine indirekte Methode die auftretenden Verzögerungen durch Interrupts zu minimieren, ist die Verringerung der Interrupt-Häufigkeit. Durch Einsatz von größeren Zwischenspeichern (z.B. FIFOs im UART) kann die Häufigkeit um den Faktor 10 gesenkt werden. Auch die Verwendung von *Jumbo frames* in entsprechend konfigurierten TCP/IP-Netzwerken senkt die Interrupt-Häufigkeit durch Versenden größerer Datenpakete. Beim Gigabit-Ethernet verringert das *Interrupt coalescing* die Interrupt-Häufigkeit. Hierbei löst die Hardware erst nach Eingang einer bestimmten Anzahl von Paketen oder nach Erreichen eines *Timeout* einen Interrupt aus. Die Folge ist eine Steigerung der Übertragungsgeschwindigkeit, leider zu Lasten der Reaktionszeit.

5.2.2 Grenzen des PCI-Busses

Das meistverwendete Bussystem der PC-Architektur für Erweiterungskarten ist der PCI-Bus. Es gibt eine große Auswahl an Karten für die verschiedensten Anwendungsbereiche. Hier sollen nun die speziellen Einflüsse des PCI-Busses auf die Realzeitfähigkeit der PC-Architektur analysiert werden [Ano99][Mai00][RST98][MS97].

Eigenheiten und optimaler Betrieb des PCI-Busses

Blockierende PCI-Wiederholungen

Einige Grafikkartenhersteller halten sich nicht an die Spezifikationen der PCI-Nutzerorganisation. Um bestmögliche Ergebnisse bei Geschwindigkeitstests zu erreichen, versuchen speziell programmierte Grafikkarten-Treiber mit ständig wiederholten *PCI-Retry*-Kommandos Daten in den Speicher der Grafikkarte zu übertragen, unabhängig davon ob dieser ansprechbar ist. Andere Übertragungen auf dem PCI-Bus werden nach Messungen von Intel [Sta98] dadurch um bis zu 20 μs verzögert. Ist der Systembus auf PCI-Bus-Basis realisiert sind davon auch Komponenten am ISA-Bus, wie z.B. der Interrupt-Controller betroffen. Hier wird durch nicht vorschriftsmäßige Programmierung zum Erreichen von wenigen Prozent Leistungssteigerung, das Zeitverhalten des Gesamtsystems wesentlich verschlechtert. Einzige Möglichkeit zum Abschalten dieser Funktion ist, falls vorhanden, ein entsprechender Eintrag in der Registrierdatenbank.

Cache-Alignment und Cache-Line Größe

Für optimale Performance sollten alle Leseoperationen (Speicher → PCI-Bus) mindestens eine Cache-Line (8 Doppelwörter) oder ein Vielfaches davon anfordern und *cache-aligned* ausgerichtet sein. Hierzu werden die Befehle *Memory Read Line* (MRL) bzw. *Memory Read Multiple* (MRM) verwendet, welche einen einzelnen *Snoop*- bzw. einen *Snoop/Snoop ahead*-Zugriff auf dem Host-Bus auslösen. Für die Schreiboperationen *Memory Write* und *Memory Write and Invalidate* verwendet die *North Bridge* Zwischenpuffer, bevor ein *Snoop cycle* initiiert werden muss.

DMA

Der PCI-Bus ist busmasterfähig. Jeder Teilnehmer kann als Master fungieren. Der Busmaster kann Daten im *Burst*-Mode übertragen und benötigt dafür keine Rechenzeit vom Systemprozessor. Der

DMA-Betrieb ist dem langsameren PIO-Mode unbedingt vorzuziehen. Zu Beachten ist, dass manchmal nicht alle vorhandenen PCI-Slots busmasterfähig sind.

Lokaler Zwischenspeicher

Sollen große Datenmengen isochron übertragen werden, muss auf dem PCI-Gerät ein lokaler Zwischenspeicher (FIFO) vorhanden sein. Sind die Zwischenspeicher unterdimensioniert, wie es bei einigen Bausteinen (z.B. AMCC S5933 PCI-Bridge, 2 x 32 Bytes) der Fall ist, wird die maximale *Burst*-Übertragungsrate des PCI-Busses nicht erreicht. Die Zugriffsregelung der Komponenten (Prozessor, AGP, PCI) auf den Hauptspeicher erfolgt durch die *North Bridge*. Jede Komponente muss ihre Daten für die Dauer eines gleichzeitigen Zugriffs (*concurrent access*) zwischenspeichern können.

Flags

Für die Kommunikation zwischen PCI-Geräten und Prozessor werden häufig Flags eingesetzt. Sind diese als Bits oder Bytes im Speicher des PCI-Geräts realisiert, muss für jede Statusabfrage ein PCI-Buszyklus erfolgen. Bei hohem Datenaufkommen zwischen verschiedenen Geräten kommt es zum ständigen Umschalten des Bus-Arbiters[59], was die effektive Datenrate sinken und die Latenzzeiten ansteigen lässt. Besser ist die Verwendung von lokalen Variablen, welche zyklisch abgefragt (gepollt) und deren Inhalt im Cache zwischengespeichert werden. Hat das PCI-Gerät eine Aktion beendet, macht es einen Schreibzugriff auf die Variable (Flag) im Hauptspeicher. Der Zugriff löst eine *Bus snoop*-Aktion aus, um den Cache-Inhalt zu aktualisieren. Beim nächsten Lesezugriff des Prozessors erfährt dieser den neuen Status des Flags. Eine minimale Prozessorlast mit maximaler Reaktionsgeschwindigkeit lässt sich durch Verwendung von Interrupts erreichen. Der PCI-Bus stellt hierfür vier Interrupt-Leitungen (INTA-D) zur Verfügung.

Write Combining

Das in Kapitel 5.2.1 vorgestellte *Write Combining* kann, falls es die Anwendung erlaubt, auch zur Kennzeichnung von *gemappten* physikalischen Speicher (Programmierung der *Memory Type Range Register* (MTRR)) von PCI-Geräten genutzt werden. Schreibzugriffe können dann gesammelt und zwischengespeichert werden, bis *Burst*-Transfers von Datenpaketen in der Größe von einer oder mehreren Cache-Lines möglich sind.

Wait states

Wenn ein PCI-Busmaster-Gerät den Bus anfordert, muss es nach Erhalt des Busses durch das `GNT#`–Signal innerhalb einer sehr kurzen Zeitspanne mit der Datenübertragung beginnen. Diese Zeitspanne ist abhängig vom Chipsatz und sank bei neuen Intel Bausteinen von 8 (430 TX) auf 5 Zyklen (440 BX). Innerhalb dieser Zeit muss das PCI-Gerät das `FRAME#`–Signal anlegen, sonst entzieht der *Arbiter* dem Gerät wieder den Buszugriff. Verzögern sich Daten, sind *wait states* nur mit Hilfe des `IRDY#`–Signals zu realisieren.

[59] *to arbitrate*: vermitteln, *Arbiter*: Vermittlungseinheit zur Vergabe des Buszugriffs.

Arbitrierung

In der PCI-Spezifikation wird kein spezifischer Arbitrierungsalgorithmus vorgeschrieben. Lediglich einige einschränkenden Erläuterungen werden gegeben. So muss das Arbitrierungsschema „fair" sein, um Verklemmungen (*deadlocks*) zu verhindern und die Latenzzeitbedingungen des PCI-Busses einzuhalten. Gerade beim Betrieb von vielen PCI-Geräten und hohem wechselnden Datenaufkommen spielt der Arbitrierungsalgorithmus eine nicht zu unterschätzende Rolle [Mai00]. Bild 5.8 zeigt die entstehenden Latenzzeiten bei Einsatz unterschiedlicher Arbitrierungsalgorithmen. Meist ist der Algorithmus durch den Chipsatz vorgegeben und nicht veränderbar oder sogar unbekannt. Für eine eigene Implementierung bietet sich z.B. der *Round-Robin*-Algorithmus an.

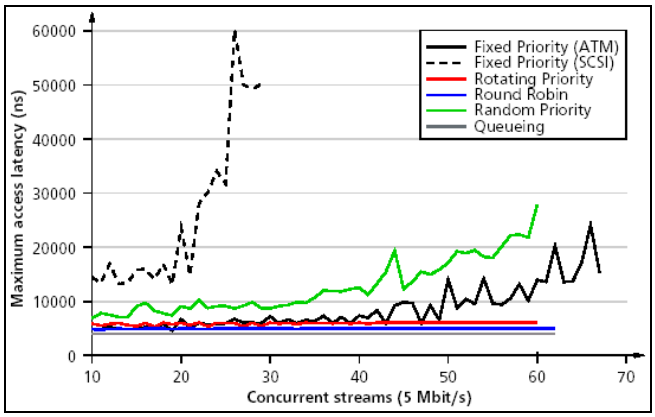


Bild 5.8 Latenzzeiten verschiedener PCI-Bus-Arbitrierungsalgorithmen

Latency Timer

Der *Latency Timer* des PCI-Busses ist immer wieder Gesprächsstoff, wenn es um die Bewertung der Realzeitfähigkeit des PCs geht. Bei der Entwicklung des PCI-Busses gab es die Anforderung, einen schnellen Bus mit niedriger Latenzzeit zu schaffen. So kann ein PCI-Busmaster-Gerät den Bus für ein bestimmtes Zeitintervall exklusiv für sich beanspruchen. Nach Ablauf des konfigurierbaren *Latency Timers* entzieht der Bus-Arbiter dem Gerät den Bus und gewährt einem wartenden PCI-Gerät den Zugriff. Die Anforderungen nach hoher Bandbreite und niedriger Latenzzeit sind jedoch gegensätzlich. Man muss also einen Kompromiss eingehen. In der PCI-Spezifikation [PCI98] wird diese Wechselwirkung ausführlich erläutert. Ein für viele Anwendungen optimaler Wert wurde von [Mai00] mit ca. 32 Taktzyklen ermittelt; die Latenzzeit beträgt dann ca. 1 µs.

Datenphasen	Übertragene Bytes	Takte	Latency Timer [Takte]	Bandbreite [MB/s]	Latenzzeit [µs]
8	32	16	14	60	0,48
16	64	24	22	80	0,72
32	128	40	38	96	1,20
64	256	72	70	107	2,16

Tabelle 5.8 Einfluss des *Latency Timers* auf Bandbreite und Latenzzeit des PCI-Busses

Compiler-Optimierungen

Bei der Übersetzung von Treibern für PCI-Geräte werden oft Optimierungsoptionen des Compilers aktiviert. [RST98] ermittelt hier eine teilweise Verschlechterung der Durchsatzrate des PCI-Busses.

Ursache ist die Umkehr der Ablaufrichtung von Schleifen durch den *Optimizer*, um eine `compare`-Operation einzusparen. Da aber kein dekrementierender *Burst*-Mode existiert, muss die Datenübertragung in viele einzelne PCI-Transaktionen aufgespalten werden. Die Transferrate sinkt stark ab.

PCI-Controller und -Bridges

Viele Hersteller bieten heutzutage PCI-Bausteine an. Der PCI-Hostcontoller mit dem *Arbiter* ist meist in die Chipsätze integriert, die von Firmen wie Intel, AMD, VIA Technologies und Acer Laboratories Inc. (ALI) produziert werden. Die Faktoren zur Steigerung des Datendurchsatzes sind, wie vorstehend beschrieben, weitgehend bekannt, die optimale Implementierung dieser ist allerdings ein gut gehütetes Geschäftsgeheimnis. So zeigen Messungen der maximalen Bandbreiten durch [Mai00] und [RST98] teilweise erhebliche Unterschiede bei den maximalen Datenübertragungsraten (Tabelle 5.9). Hier ist der neuere Chipsatz nicht unbedingt der schnellere (siehe z.B. Intel 430 FX). Auch die auf den Steckkarten eingesetzten PCI-Controller unterscheiden sich erheblich in Funktionsumfang, Unterstützung des Masterbetriebs, Größe der FIFOs und Verfügbarkeit von Entwicklungswerkzeugen und Programmierbeispielen. Ähnliches gilt für die zur Kopplung von mehreren Bussen eingesetzten PCI-Bridges. Große Leistungsunterschiede treten oft durch unterdimensionierte Zwischenpuffer oder schlichtweg durch Designfehler (*Bugs*) auf [MS97].

Prozessor	Chipsatz	DMA Schreiben (PCI → RAM)	DMA Lesen (PCI ← RAM)	PIO Schreiben (CPU → PCI)	PIO Lesen (CPU ← PCI)
Pentium 90	430 NX	68	52	54	13
Pentium 133	430 FX	131	125	88	13
PentiumPro 200	450 KX	129	110	19	7
PentiumPro 200	440 FX	130	125	29	10
Dual PentiumPro 200	440 FX	131	121	29	9
Dual Pentium II 266	440 LX	131	121	37	9

Tabelle 5.9 Übertragungsraten verschiedener PCI-Hostcontroller (in [MB/s])

Weitere Besonderheiten

Die Anbindung des ISA-Busses an den PCI-Bus in der *South Bridge* ist oft für auftretende Verzögerungen und niedrige Bandbreiten verantwortlich [MS97]. Der Chipsatz unterstützt weitere Besonderheiten um diese Anbindung zu verbessern:

- *Passive release* ermöglicht dem PCI-Bus weiterzuarbeiten, während Daten vom ISA-Bus empfangen und zwischengespeichert werden.
- *Delayed transaction* erlaubt PCI-Busmaster-Geräten weiterzuarbeiten, indem Übertragungen zu ISA-Karten verzögert werden.
- *Write merging* fasst Byte-, Word- und DWord-Zyklen zu einem Schreibzugriff zusammen.
- *Concurrent-PCI* erlaubt es, mit dem *Multi Transaction Timer* mehrere Übertragungen ohne Neu-Arbitrierung in einer PCI-Anforderung zu bündeln.
- *Transaction Pipelining* ermöglicht eine quasiparallele Bearbeitung von Datenübertragungen der drei Schnittstellen (CPU, AGP, PCI) innerhalb der *North Bridge*.

5.3 Erweiterte Architekturkonzepte

In den Hauptkapiteln 4 und 5 wurden software- und hardwarebasierte Systemarchitekturen zur Bearbeitung zeitkritischer technischer Prozesse vorgestellt. Dabei erreichen die softwarebasierten Systeme unter Verwendung von Realzeitbetriebssystemen die auf einem Standard-PC-System möglichen minimalen Reaktionszeiten. Eine weitere Verringerung der Reaktionszeiten kann nur durch Einsatz zusätzlicher Hardware-Komponenten, wie in Kapitel 5.1 beschrieben, erzielt werden.

Die Nachteile dieser Hardwareerweiterungen, z.B. höhere Kosten durch zusätzliche Komponenten und Entwicklungswerkzeuge, Inkompatibilität zur Standard-PC Softwarearchitektur aufgrund proprietärer Firmware, zusätzlicher Kommunikationsaufwand zwischen den Systemen, sowie meist geringere Leistungsfähigkeit der verwendeten Recheneinheiten, sind augenscheinlich. Viele Firmen versuchen aus diesen Gründen, die Leistungsfähigkeit der PC-Architektur zu erhöhen und die Reaktionszeiten zu verringern.

5.3.1 Probleme und Verbesserungsansätze

Die wichtigsten Problemstellen und Flaschenhälse der klassischen PC-Systemarchitektur sind:

- Flaschenhals *Front Side Bus*
- Zweigeteilte Systemarchitektur
- Zwischenspeicher des Prozessors und Chipsatzes
- Konkurrierende Datenpfade
- Mangelnde Leistungsfähigkeit des Speichersubsystems
- Stagnierende I/O-Leistung des PCI-Busses

Im Folgenden werden verschiedene Verbesserungsmethoden beschrieben.

Flaschenhals *Front Side Bus*

Die Busverbindung (*Front Side Bus*, FSB) zwischen Prozessor, oder mehreren Prozessoren, und dem Chipsatz ist ein Ziel von Verbesserungsmaßnahmen. Die Taktraten erhöhten sich in den letzten 3 Jahren von 66 MHz auf 100 MHz (x 4), die Übertragungsraten entsprechend von 0,5 auf 3,2 GB/s. Leistungsfähigster Bus ist zur Zeit der FSB des Intel Pentium 4 Prozessors, welcher durch seine *quad pumped*-Technik in der Lage ist, 4 Datenpakete pro Takt (100 MHz) zu übertragen. Der nächste Schritt von Intel ist eine Erhöhung der Taktrate auf 133 MHz, um die Übertragungsrate auf 4,25 GB/s zu erhöhen. Eine ähnliche Technik mit 2 Datenpaketen pro Takt, einer Busfrequenz von 133 MHz und einer maximalen Übertragungsrate von 2,1 GB/s setzt die Firma AMD bei ihrem *Athlon* Prozessor ein. Auch hier ist eine Erhöhung auf 200 MHz (x 2) anvisiert.

Bei einem Vergleich der Datenraten muss jedoch Busstruktur und Busprotokoll in die Analyse miteinbezogen werden. An den Prozessorbus des Pentium 4 können mehrere CPUs angeschlossen werden, um ein Multiprozessorsystem aufzubauen [TFG+01]. *Symmetric agents* (CPUs) und *priority agent* (Chipsatz *North Bridge*) müssen sich die zur Verfügung stehende Übertragungsrate teilen.

Der notwendige Arbitrierungsprozess[60] und das transaktions-basierte Protokoll reduzieren die Effizienz. Der von DEC entwickelte und von AMD für den Athlon lizenzierte Alpha-EV6-Bus ist im strengen Sinne kein Bus, sondern basiert auf einer Punkt-zu-Punkt-Verbindung, die in Multiprozessor-Systemen jedem Prozessor die volle Bandbreite zur Verfügung stellt. Nachteil ist der höhere Verdrahtungsaufwand der Systemplatine bei mehreren Prozessoren. Eine Übersicht der Prozessorbusse in aktuellen PC-Systemen zeigt Tabelle 5.10, die Struktur einer modernen Link-basierten Architektur zeigt Bild 5.9.

FSB Takt	Übertragungsrate	Busprotokoll	CPU
66 MHz	528 MB/s	AGTL+	Intel Celeron < 800 MHz
100 MHz	800 MB/s	AGTL+	Intel Pentium II, III, Celeron > 800 MHz
133 MHz	1,06 GB/s	AGTL+	Intel Pentium III
100 MHz (x2)	1,6 GB/s	EV6	AMD Athlon, Duron
133 MHz (x2)	2,1 GB/s	EV6	AMD Athlon
100 MHz (x4)	3,2 GB/s	AGTL+	Intel Pentium 4
133 MHz (x4)	4,25 GB/s	AGTL+	Intel Pentium 4 (ab Q3/2002)

Tabelle 5.10 Übertragungsraten des *Front Side Bus*

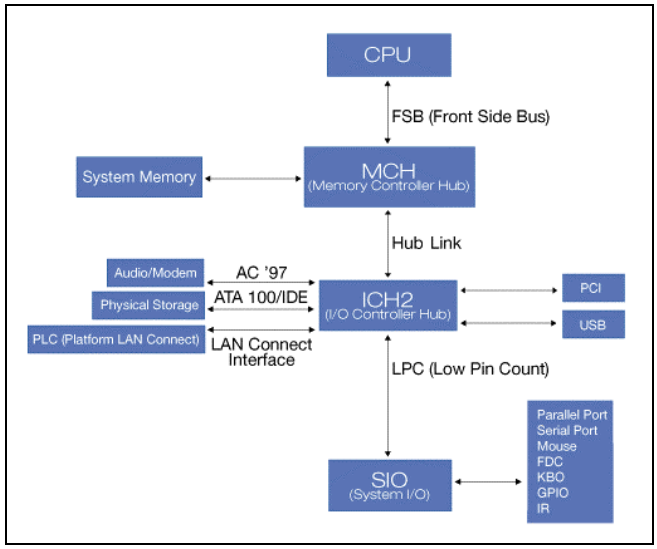


Bild 5.9 Moderne PC-Systemarchitektur mit Intel *Hub-Link*

Zweigeteilte Systemarchitektur

Das klassische PC-System besteht aus einer Vielzahl von integrierten Schaltungen. Die Hauptaufgaben werden heutzutage überwiegend von zwei hochintegrierten Bausteinen, dem Chipsatz, erledigt (siehe hierzu auch Grundlagenkapitel 2). Der als *North Bridge* oder *Memory Controller* bezeichnete Chip enthält verschiedene Systembusse und eine Verbindung zum zweiten Baustein, der *South Bridge* (oder *I/O Controller*). In diesem Baustein sind zahlreiche Peripherieschnittstellen sowie weitere Systemfunktionalitäten integriert. Die Verbindung zwischen *North* und *South Bridge* nutzte bisher den PCI-Bus als Kommunikationsmedium. Das stetig gestiegene Datenaufkommen, gerade durch den enormen Geschwindigkeitszuwachs moderner Festplatten, führt zur vollständigen

[60] Der FSB von Intel benutzt das „*Round Robin*" Arbitrierungsverfahren. Die *North Bridge* kann mit dem hochprioren Signal `BPRI#` den Bus übernehmen.

Auslastung des PCI-Busses und dadurch zu Verzögerungen bei der Datenübertragung (siehe auch Kapitel 5.2.2, Grenzen des PCI-Busses).

Abhilfe schafft hier die Einführung eines schnellen, dezidierten Bussystems zur ausschließlichen Datenübertragung zwischen den Systembausteinen. Hier sind einige Chipsatz-Hersteller aktiv geworden und haben firmeneigene Bussysteme entwickelt. Diese werden bereits in aktuellen und auch in zukünftigen Produkten eingesetzt, wie Tabelle 5.11 zeigt.

Chipsatz-Bus	Chipsatz-Hersteller	Beschreibung	Eingesetzt im Chipsatz	Silizium
IMB (Inter Module Bus)	ServerWorks	Datenbus: 16-Bit, Takt: 266 MHz, Datenrate: 533 MB/s, Takt: 500 MHz, Datenrate: 1 GB/s	ServerSet III HE ServerSet III HEls	Q1/2000 Q1/2002
Hub-Link	Intel	Datenbus: 8-Bit, Takt: 133 MHz, DDR, Datenrate: 266 MB/s Datenbus: 16-Bit, Takt: 133 MHz, DDR, Datenrate: 533 MB/s, (für PCI64-Anschluss)	alle i8xx i840, i860	Q3/2000
V-Link	VIA	Datenbus: 8-Bit, Takt: 133 MHz, DDR, Datenrate: 266 MB/s	KT 266, P4X266	Q1/2001
MuTIOL	SIS	Datenbus: 16-Bit je Richtung, Takt: 266 MHz, Datenrate: 2 x 533 MB/s	SIS 645	Q3/2001
Hyper-Transport	nVidia	Datenbus: 8-Bit je Richtung, LVDS[61], Takt: 400 MHz, Datenrate: 2 x 400 MB/s	nVidia nForce	Q4/2001
GeodeLink	National Semicon-ductor	Datenbus: 16 – 256 Bits, Takt: 33 – 300 MHz, Datenrate: max. 6 GB/s, Switched Fabric für SoC	Geode GX2 CPU	Q1/2002

Tabelle 5.11 Bussysteme für Chipsatz-Kommunikation

Zwischenspeicher des Prozessors und Chipsatzes

Der PC ist, gemäss seines Hauptanwendungsgebiets als universelle Rechenmaschine für grafische Applikationen, für hohe, durchschnittliche Datentransferraten und Rechenleistungen optimiert. Dazu sind eine Vielzahl von Zwischenspeichern[62] und Warteschlangen im Prozessor und in den verschiedenen Komponenten des System-Chipsatzes integriert. Entsprechend den Bestrebungen der Industrie, die Leistungsfähigkeit der PC-Plattform fortwährend zu erhöhen, werden diese Zwischenspeicher mit Fortschreiten der technologischen Fähigkeiten ständig vergrößert. Zwar sinken die minimalen Verzögerungszeiten durch die Beschleunigung der Kommunikationskanäle innerhalb der PC-Architektur (FSB, Systembus), die Varianz der Verzögerungszeiten steigt jedoch aufgrund der großen Zwischenspeicher bei hohen Systembelastungen stark an. Die Ermittlung dieser maximalen Zeiten, insbesondere für die Systemverifikation vor Einsatz in zeitkritischen technischen Prozessen, ist sehr aufwändig und in immer stärkerem Maße von Art, Komplexität und dem Zusammenwirken der Aufgaben abhängig.

[61] LVDS: ***L**ow **V**oltage **D**ifferential **S**ignalling*.
[62] hier sind Puffer und nicht die Caches gemeint.

Konkurrierende Datenpfade

Ein weiterer Ursprung verminderter Datendurchsätze und erhöhter Verzögerungszeiten ist die hohe Anzahl an Bus-Schnittstellen in der *North Bridge* (bzw. *Memory Controller*) des Chipsatzes. Die verschiedenen Datenpfade müssen entsprechend der Zieladressen geroutet werden. Verdrängungen und Blockierungen lassen sich, insbesondere bei datenintensiven Applikationen, nicht vollständig vermeiden. Bild 5.10 zeigt beispielhaft die interne Struktur.

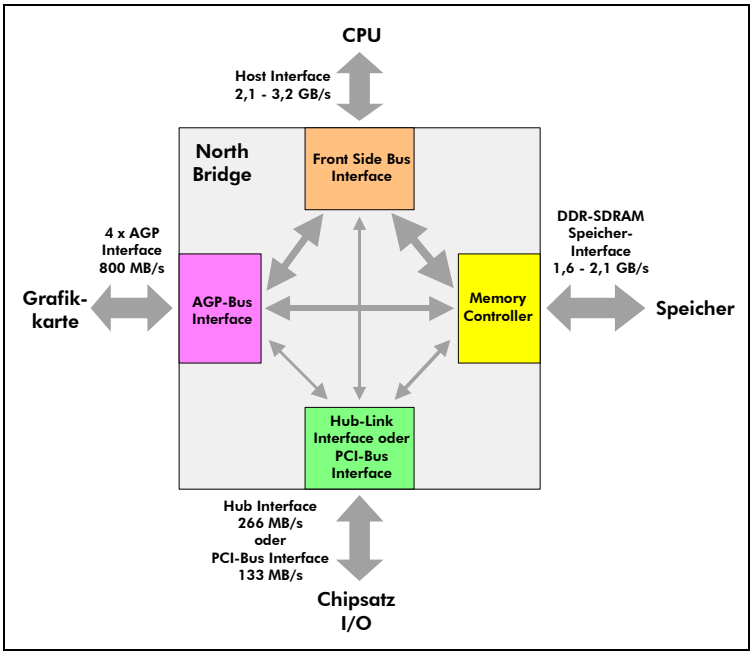


Bild 5.10 Interne Busse und Datenpfade der *North Bridge*

Ein neuer Ansatz zur Entkopplung der konkurrierenden Datenpfade innerhalb der *North Bridge*, ist die Nutzung geschalteter Verbindungen mit einem *Switch*[63]. Durch eine chipinterne Schaltkomponente werden für die Dauer einer Datenübertragung Punkt-zu-Punkt-Verbindungen zwischen den einzelnen Kommunikationspartnern aufgebaut. Die maximale Übertragungsgeschwindigkeit des *Switches* muss der akkumulierten Datenrate aller angeschlossenen Komponenten entsprechen. Geschaltete Verbindungen werden in den Chipsätzen *Profusion* (Corollary/Intel), *nForce* (nVidia) und *Summit* (IBM) implementiert. Auch der aufstrebende Bereich der *System-On-a-Chip* (SoC) Systeme verwendet geschaltete Verbindungen zur Kommunikation zwischen den einzelnen Schaltungsteilen. Eine aktuelle *On-Chip-Switch*-Struktur für leistungsstarke Mikrocontroller ist z.B. *OCeaN* von Motorola mit einer maximalen Summendurchsatzrate von 128 GBit/s.

Mangelnde Leistungsfähigkeit des Speichersubsystems

Die gestiegene Rechenleistung der Prozessoren und der Zuwachs an Speicherbedarf für komplexere Anwendungen bedingt eine Erhöhung der Zugriffsrate auf den Hauptspeicher. Die Übertragungsraten der Chipsatz-Speicherschnittstellen werden deshalb stetig verbessert. Hierbei kommen evolutionäre Techniken, z.B. die schrittweise Erhöhung der Taktfrequenz, aber auch revolutionäre Techniken, z.B. die Übertragung der Daten an beiden Taktflanken oder der Einsatz neuer Spei-

[63] *non-blocking crossbar switch fabric*, in etwa: blockierungsfrei geschaltetes Verbindungsnetzwerk.

cherbussysteme, zum Einsatz. Die maximale Taktfrequenz der SDRAM-Speichermodule[64] erreicht inzwischen 133 MHz (PC133[65]). Damit können Daten, bei hintereinanderfolgenden Zugriffen und einer Busbreite von 64-Bit, mit maximal 1,066 GB/s übertragen werden. Einige, nicht-standardisierte Speichermodule arbeiten sogar mit 145 oder 166 MHz. Überträgt man ein Datenpaket an jeder Taktflanke (DDR[66]), so steigt die maximale Datenrate bei 133 MHz auf 2,1 GB/s an. Man verwendet hierfür die Bezeichnung DDR266, entsprechend dem Takt eines vergleichbaren Moduls ohne DDR, oder PC2100, entsprechend der maximalen Datenrate. Aktuell (Q4/2001) wird an einer weiteren Erhöhung der Frequenzen gearbeitet. Erste Muster mit einer Taktfrequenz von 166 MHz (PC2700) wurden bereits vorgestellt. Tabelle 5.12 gibt einen Überblick der verschiedenen Speichertechnologien.

RAM-Technologie	Kürzel	Takt [MHz]	JEDEC-Standard	Max. Datenrate [MB/s]
Fast Page Mode	FPM-DRAM	66		200
Extended Data Out	EDO-DRAM	66		300
Burst Extended Data Out	BEDO-DRAM	66		500
Synchronous	SDRAM	66	PC66	500
		100	PC100	800
		133	PC133	1066
Double Data Rate Synchronous	DDR-SDRAM	100	PC1600	1600
		133	PC2100	2100
		166	PC2700	2700
Rambus	RDRAM	300	PC600	1200
		400	PC800	1600

Tabelle 5.12 Daten- und Taktraten unterschiedlicher Speichertechnologien

Einen anderen Weg geht das Unternehmen Rambus mit ihrem patentierten Speichersystem RDRAM. Diese Entwicklung (vorgestellt 1996) verwendet eine sehr hoch getaktete aber schmale (8- oder 16-Bit) Busstruktur. Die Integration auf Leiterplatten erfordert angepasste Leitungslängen und spezielle Übertragungsmechanismen für den Takt (*clock*). Mit 400 MHz, einer Busbreite von 2x8-Bit und DDR-Technik wird eine maximale Übertragungsrate von 1,6 GB/s erreicht (PC800). Auch hier gibt es aktuelle Erweiterungen hinsichtlich Taktrate (500 MHz) und Busbreite (2x16-Bit) für zukünftige Module mit 4 GB/s (PC2000).

Diese nur vom Marktführer Intel unterstützte Technologie, die hohen Lizenzgebühren der Firma Rambus sowie komplizierte Layoutvorschriften führten zum Zusammenschluss konkurrierender Speicher- und Chipsatzhersteller, mit dem Ziel, DDR-SDRAMs als preisgünstigen und schnellen Speichertyp verstärkt weiterzuentwickeln. Den Erfolg dieses Unterfangens zeigen die, im Vergleich zu RDRAM, höheren Übertragungsraten und der aktuelle Umstieg Intels zu DDR-SDRAM mit dem i845-Chipsatz. Nichtsdestotrotz ist die den RDRAM-Modulen zugrundeliegende Speichertechnologie zukunftsträchtig, gerade bei stetig steigenden Signalfrequenzen. In einigen PC-Systemen, besonders für Anwendungen mit hohen Speichertransferraten, werden Chipsätze mit zwei (Intel i840 – 860, nVidia *nForce*) oder sogar vier (ServerWorks *ServerSet HE*) Speicher-Controllern integriert.

[64] SDRAM: ***S**ynchronous **D**ynamic **R**andom **A**ccess **M**emory*.
[65] PCxxx: PC66 – PC2700, JEDEC-Standard *JESD21-C* für die verschiedenen Speichermodule.
[66] DDR: ***D**ouble **D**ata **R**ate*, Übertragung von Daten an beiden Taktflanken, JEDEC-Standard *JESD79*.

Ausschlaggebend für zeitkritische technische Prozesse ist jedoch nicht die maximale Übertragungsrate sondern die Latenzzeit zwischen Anforderung des Speicherinhalts und Transfer zum Prozessor. Die höheren Taktfrequenzen in Verbindung mit kürzeren Zugriffszeiten verbessern hier erfolgreich die Latenzzeiten.

Eine weitere Technologie zur Verringerung der Latenzzeiten bei Hauptspeicherzugriffen entwickelt die Firma IBM mit ihrer *Memory eXpansion Technology* (MXT) [AHM+00]. Sie ist in IBMs *Summit*-Chipsatz für Intels 32-Bit Prozessor *Foster* (Nachfolger des *Xeon*) und 64-Bit Prozessor *McKinley* (Nachfolger des *Itanium*) sowie in ServerWorks *ServerSet III*-Chipsatz implementiert. Durch Reorganisations- und Komprimierungsalgorithmen können die Latenzzeiten laut IBM um bis zu 50 % gesenkt werden. Zusätzlich hilft ein vom Chipsatz kontrollierter *Level-4*-Cache mit maximal 64 MB DDR-SDRAM die Frequenz der Hauptspeicherzugriffe weiter zu senken.

Stagnierende I/O-Leistung des PCI-Busses

Der prozentuale Anteil von I/O-Daten bei der Ausführung von Programmen hat inzwischen 50 % überschritten. Im Gegensatz zur kontinuierlichen Leistungssteigerung der Prozessoren vollzieht sich die Weiterentwicklung der I/O-Schnittstellen der PC-Architektur nur langsam (Bild 5.11).

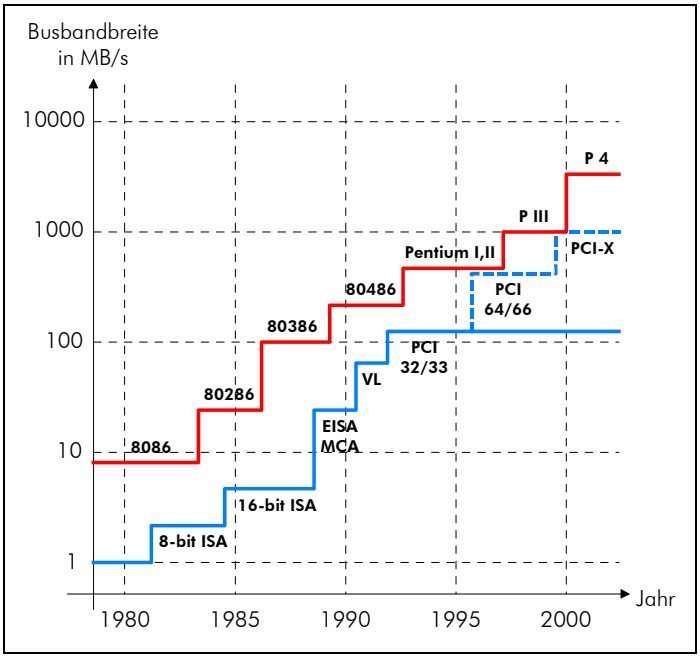


Bild 5.11 Entwicklung der Busbandbreiten von Prozessoren und I/O-Schnittstellen

Bisher konnten sich Weiterentwicklungen des Standard-PCI-Busses (32-Bit, 33 MHz) durch höhere Taktraten (66 MHz) und breitere Busse (64-Bit) bei *PCI-64/66* oder mittels der neuen Spezifikationen *PCI-X* (Bild 5.12) nicht durchsetzen.

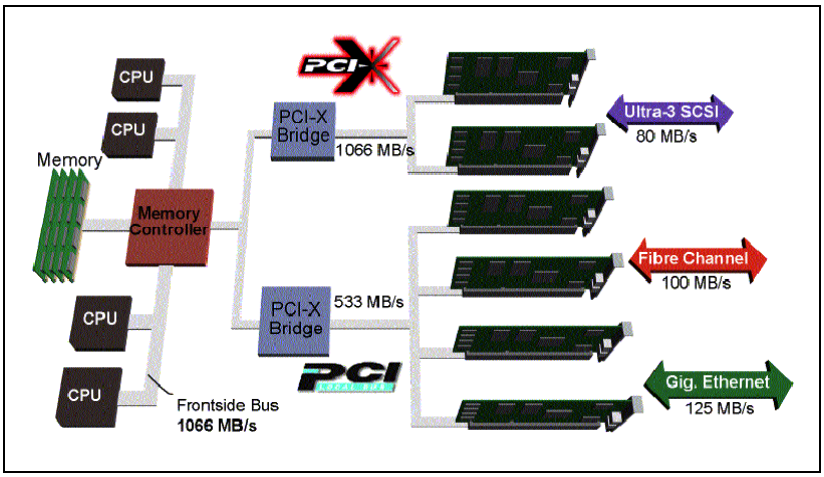


Bild 5.12 Erweiterung des PC-Systems mit PCI-X

Gründe sind z.B. die wesentlich höheren Kosten aufgrund hoher Kontaktzahlen (*PCI-64/66* und *PCI-X*: 184 Kontakte) oder die Unzulänglichkeiten des Designs. So erlaubt der *PCI-X*-Standard bei Busfrequenzen von 133 MHz lediglich den Anschluss einer einzigen Steckkarte (Tabelle 5.13). Ein sinnvoller Ersatz des PCI-Busses ist so nicht möglich[67].

Bus	Busbreite [Bit]	Taktfrequenz [MHz]	Max. Anzahl an Standard-Steckplätzen[68]	Max. Übertragungs-rate[69] [Mio. B/s]
PCI	32	33	4 (6)[70]	132
	32	66	4	264
	64	33	4	264
	64	66	2	528
PCI-X	64	66	4	528
	64	100	2	800
	64	133	1	1064

Tabelle 5.13 Vergleich zwischen PCI und PCI-X

5.3.2 Neue Schnittstellen-Architekturen

Hersteller von PC-Systemen und –Komponenten arbeiten zusammen, um völlig neue, leistungsfähige I/O-Schnittstellen zu entwerfen. Mehrere Gruppen unter Federführung großer Halbleiterfirmen konkurrieren mit verschiedenen Realisierungen und Entwürfen um die Etablierung zukünftiger Standards. Die Architekturen lassen sich in zwei Hauptgruppen unterteilen, die beide geschaltete Punkt-zu-Punkt-Verbindungen verwenden (*switch fabric*). Die erste Gruppe konzentriert sich auf den Einsatz als Systembus, lokaler I/O-Bus oder Backplane:

- HyperTransport (AMD)
- RapidIO (Motorola)

[67] sofern man bei Anschluss vom einem einzigen Gerät noch von einem Bus sprechen kann.
[68] In der PCI-Spezifikation ist lediglich die maximale Anzahl an Buslasten (10) spezifiziert. Ein Standard-PCI-Slot entspricht ca. 2 Buslasten, ein Compact-PCI-Slot entspricht 1 Buslast.
[69] Brutto-Übertragungsrate im *Burst*-Mode ohne Overhead.
[70] Durch exaktes Platinendesign kann die Buslast eines Standard-PCI-Slots auf ca. 1,5 Buslasten gesenkt werden.

- 3GIO (Intel)
- InfiniBand (Compaq)

Eine Sonderstellung nimmt hier die Entwicklung *InfiniBand* ein, dessen Haupteinsatzgebiet die Vernetzung mehrerer PC-Systeme ist, sogenanntes *Distributed I/O*. Die zweite Gruppe erweitert die bestehende PCI-Bus-Struktur mit geschalteten Verbindungen und differentieller Übertragungstechnik zu verteilten I/O-Systemen:

- StarFabric (StarGen)
- GigaBridge (PLXTech)

Die jeweils typischen Einsatzgebiete der Schnittstellen zeigt Bild 5.13.

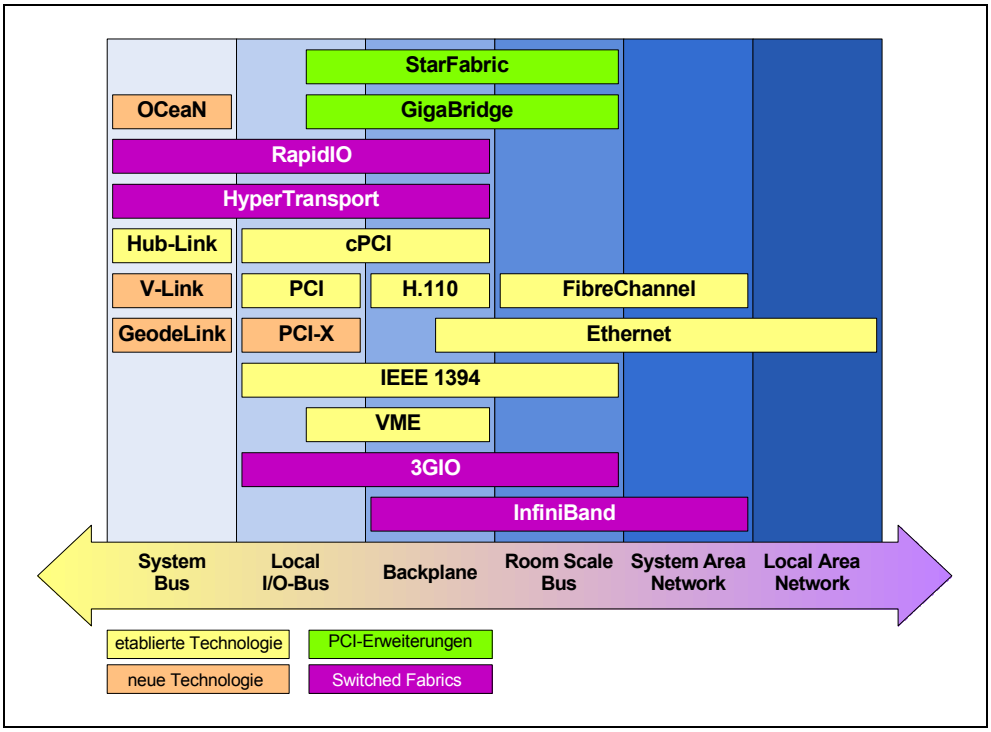


Bild 5.13 Einsatzgebiete der Schnittstellen-Architekturen

Tabelle 5.14 stellt die unterschiedlichen spezifischen Eigenschaften, wie z.B. Bandbreite, Frequenz und Verfügbarkeit dar ([Bec00][Api00][Com00][Ali00][Rap00][Str98][Wil00][RH00]).

	HyperTransport	RapidIO	3GIO	Infiniband	StarFabric	GigaBridge
Konsortium-führer	AMD, API Networks	Motorola	Intel	Compaq, Intel, IBM	StarGen	PLXTech
Bustyp	LVDS, unidirektionale Punkt-zu-Punkt-Verbindungen	LVDS, unidirektionale Punkt-zu-Punkt-Verbindungen	LVDS, unidirektionale Punkt-zu-Punkt-Verbindungen	LVDS/optisch, unidirektionale Punkt-zu-Punkt-Verbindungen	LVDS, unidirektionale Punkt-zu-Punkt-Verbindungen	LVDS, unidirektionale Punkt-zu-Punkt-Verbindungen
Busbreite (Bits pro Richtung)	2, 4, 8, 16, 32	8, 16	1, 2, 4, 8, 12, 16, 32	1, 4, 12	4	16
Bandbreite (pro Richtung)	0,8 – 64 GBit/s (200 MHz/2-Bit – 1 GHz/32-Bit), aktuell: 6,4 GBit/s (400 MHz/8-Bit)	4 – 32 GBit/s (250 MHz/8-Bit – 1 GHz/16-Bit), aktuell: 8 GBit/s (500 MHz/8-Bit)	2,5 – 80 GBit/s (1-Bit – 32-Bit)	2.5 – 30 GBit/s (1-Bit – 12-Bit)	2,5 GBit/s (622 MHz/4-Bit)	6,4 GBit/s (400 MHz/16-Bit)

	HyperTransport	RapidIO	3GIO	Infiniband	StarFabric	GigaBridge
Frequenz	200 MHz – 1 GHz, DDR	250 MHz – 1 GHz, DDR		1,25 GHz, DDR	622 MHz	400 MHz
Maximale Leitungslänge	60 cm (400 MHz)	75 cm (250 MHz)		10 m (Kupfer), 300 m (optisch)	5 m (Cat5)	5 m (Cat5)
Anzahl der Kontakte	55 bei 2 x 6,4 GBit/s	40 bei 2 x 16 GBit/s	40 bei 2 x 20 GBit/s	25 bei 2 x 10 GBit/s		
Protokoll	Paketbasiert, 4 – 64 Bytes	Paketbasiert, 1 – 256 Bytes	Paketbasiert	Paketbasiert, 0 (256) – 4096 Bytes	Paketbasiert, transparent zu PCI	Paketbasiert, transparent zu PCI, Redundanz
Silizium	PCI-Bridge, 4-Port Switch, MIPS CPUs, Chipsatz (nForce)	PCI/PCI-X-Bridge, 6-Port Switch, MPC8540 PowerPC	Ende 2003	PCI-X Bridge mit 4-Port (1x) InfiniBand Switch, 16-Port Switch (1x), 8-Port Switch (4x)	PCI-to-StarFabric Bridge, H110-to-StarFabric Bridge, 6-Port Switch, 21-Slot StarFabric/ cPCI Backplane	PCI64-to-GigaBridge Bridge

Tabelle 5.14 Eigenschaften der neuen Schnittstellen-Architekturen

Beispiele für die typische Systemanordnung der vier Schnittstellen-Architekturen der ersten Gruppe zeigt Bild 5.14. Deutlich zu erkennen sind die zuerst implementierten Einsatzgebiete der Systeme *HyperTransport* und *3GIO* als schneller Systembus zur Inter-Chipsatz-Kommunikation. Bei *RapidIO* soll die Schnittstellen-Funktionalität direkt in Prozessoren integriert werden, um so einen dedizierten Chipsatz überflüssig zu machen. Die Kombination von Schaltbausteinen (*switches*) und in I/O-Geräte implementierten Schnittstellen-Funktionalitäten zeigen den von *InfiniBand* anvisierten Verwendungszweck der Systemvernetzung, speziell für den Server- und *Clustering*-Bereich. Mit Brückenbausteinen (*bridges*) lassen sich Verbindungen zu konventionellen Bussystemen oder zu Architekturen der Mitbewerber herstellen.

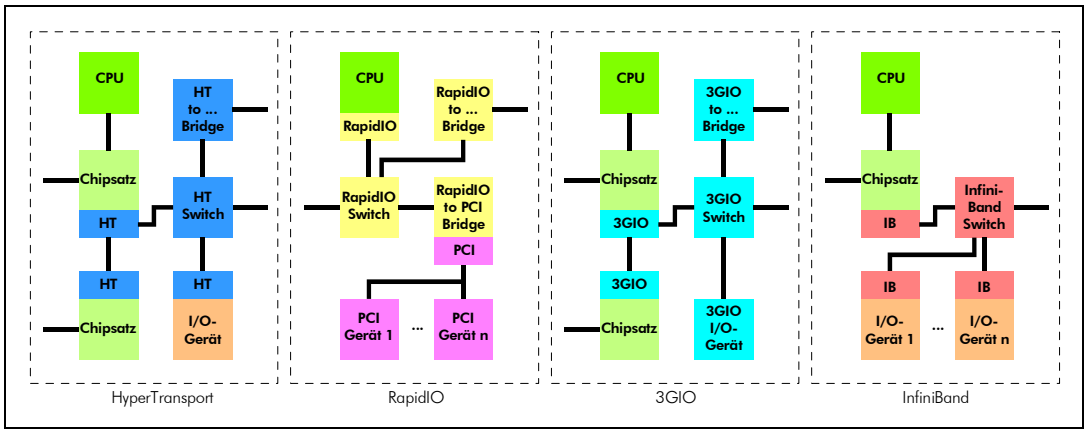


Bild 5.14 Beispiele zur Systemanordnung der Schnittstellen-Architekturen

Die Struktur der Schnittstellen-Architekturen der zweiten Gruppe zeigt Bild 5.15. Geschaltete Verbindungen mit differentieller Übertragungstechnik dienen hier zur Erweiterung vorhandener Systeme. So kann der Standard-PCI-Bus mit der *GigaBridge*-Technologie transparent, d.h. nicht sichtbar für die Software und deshalb ohne Anpassung der Treiber, auf bis zu 224 separate Bussegmente erweitert werden. Der zweite parallel geschaltete Ring übernimmt im Fehlerfall die Datenübertragung und gewährleistet einen ununterbrochenen Betrieb (Redundanz). *StarFabric* von StarGen zielt auf die Verbindung mehrerer Komponenten des Telekommunikationsbereichs über eine Backplane. Brückenbausteine verbinden hierbei H.110-Bussysteme mit der *StarFabric*.

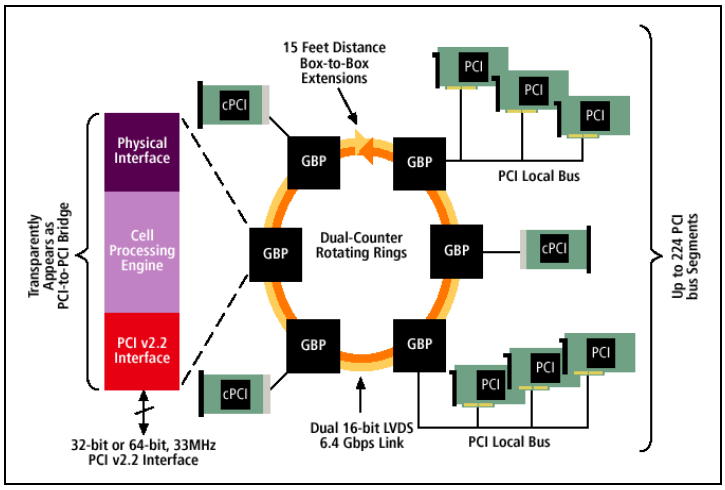


Bild 5.15 Systemstruktur der *GigaBridge*-Architektur

5.3.3 Verbesserte Software-Kommunikationsstrukturen

Die im vorigen Kapitel beschriebenen neuen Schnittstellen-Architekturen haben zugehörige Software-Protokollschichten, die insbesondere bei *InfiniBand* sehr umfangreich ausfallen. Die entstehende Komplexität bei einer Implementierung nach dem klassischen ISO-OSI-Schichtenmodell hat mehrere Nachteile:

- Mehrfache Kopien der Daten in Zwischenspeichern der Software-Protokollschichten und der Hardware
- Treiber erzeugen viele Funktionsaufrufe des Kernels (*Overhead*)
- Umsetzung von virtuellen in physikalische Speicherbereiche
- Zahlreiche *Context switches* des Betriebssystems
- Komplexe, in Software realisierte Protokoll-Stacks und Sicherungsmaßnahmen (Checksummen, CRC) benötigen hohe Rechenleistungen

Das Resultat sind hohe Latenzzeiten bei der Datenübertragung zwischen zwei verteilten Anwendungen. Untersuchungen an Ethernet-basierten Netzwerken bei hohen Übertragungsraten (Bild 5.16) zeigen den steigenden Anteil der Ausführungszeiten durch den Software-*Overhead* [Bia01].

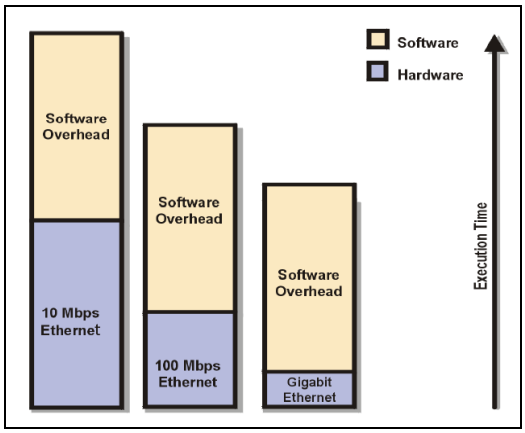


Bild 5.16 Software- und Hardware-Ausführungszeiten der Ethernet-Übertragung

Aus diesem Grund wurde ab 1996 von mehreren Hersteller (Intel, Compaq, Dell, Microsoft u.a.) die skalierbare *Virtual Interface (VI) Architecture* entwickelt. Ziel ist die Verringerung von Verzögerungszeiten durch Software-Kommunikationsstrukturen mittels einer effizienten und ressourcensparenden Schnittstellenstruktur. Hierbei wird einem Anwenderprogrammen der direkte Zugriff auf die Netzwerk-Schnittstelle über einen kleinen *kernel agent* ermöglicht. Jedem Programm wird ein eigener, virtueller Punkt-zu-Punkt-Übertragungskanal (*Virtual Interface*) zur Verfügung gestellt. Die Kommunikation erfolgt über einen gemeinsamen Speicherbereich, der zusammen mit Speicherschutz- und Speicherumsetzungsmechanismen in der intelligenten Netzwerkkarte implementiert ist. Die Kommunikationsstruktur zeigt Bild 5.17.

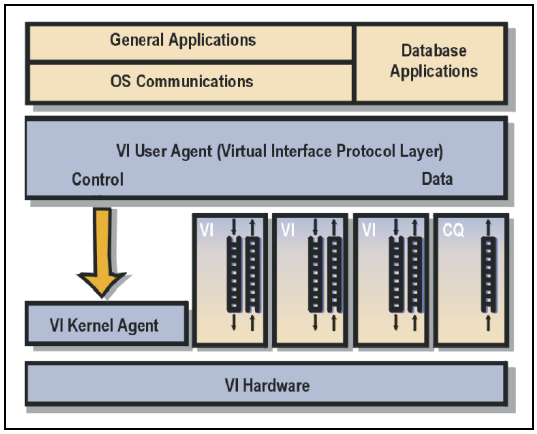


Bild 5.17 Kommunikationsstruktur der *Virtual Interface Architecture*

Messungen an Prototypen ergaben sehr kurze Latenzzeiten. Mit einer software-basierten Implementierung der VI-Architektur im Netzwerk-Adapter entstehen bei der Übertragung einer 4-Byte Nachricht Einweg-Latenzzeiten von 18,2 μs (Windows NT, Pentium III, 350 MHz) [BAH+00]. Reine Hardware-Implementierungen erreichen minimale Latenzzeiten von 3,5 μs (Windows NT, Dual Pentium III, 600 MHz) [Gig00].

Natürlich gibt es außer den beschriebenen Vorteilen auch einige Nachteile, welche gerade den kommerziellen Erfolg dieser Technologie verzögern [Pan99]. Hier sind zu nennen:

- Keine Unterstützung von *Multicast* und *Broadcast*
- Keine verbindungslosen Kommunikations-Modi
- Stellenweise mehrdeutige oder unklare Spezifikation ermöglicht unterschiedliche Interpretation
- Schwerfällige Softwarebibliotheken erschweren Programmierung
- Undokumentierte Erweiterungen der Hersteller

Aus den beschriebenen Problemen der PC-Systemarchitektur erkennt man, dass eine Überarbeitung und Modernisierung der PC-Architektur zwingend notwendig ist. Die verschiedenen Hersteller müssen sich für eine kundenorientierte Umsetzung auf eine (oder wenige) neue, durchdachte Hardware-Schnittstellen-Architekturen und Software-Kommunikationsstrukturen einigen. Durch Weiterverwendung von bewährten Techniken, z.B. der oberen Protokollschichten des PCI-Busses, und Weglassen von veralteten, unnötigen Techniken (z.B. ISA-Bus, A20-Gate usw.) kann eine neue, leistungsfähige PC-Architektur mit verringerten Latenzzeiten für zeitkritische technische Anwendungen geschaffen werden.

6 Anwendungsbeispiele

In den Kapiteln 3,4 und 5 wurden die theoretischen Hintergründe und verschiedenen Architekturen zur Realisierung von zeitkritischen, technischen Prozessen auf PC-Systemen beleuchtet. In diesem Kapitel werden nun zwei praktische Anwendungsbeispiele beschrieben, welche im Rahmen zweier Forschungsprojekte des Forschungsverbundes FORMIKROSYS II[71] entstanden. Die beteiligten universitären Projektpartner waren das Forschungszentrum für wissensbasierte Systeme (FORWISS) der Universität Passau, der Lehrstuhl für Feingerätebau und Mikrotechnik und der Lehrstuhl für Realzeit-Computersysteme der Technischen Universität München. Ziel der beiden Projekte war die Erforschung und Entwicklung moderner *Rapid Prototyping*-Verfahren. Im ersten Projekt wurde die realzeitkritische Druckkopfansteuerung für den Prototyp eines 3D-Druckers entwickelt. Das zweite Projekt beschäftigte sich mit der Steuerung der Laserscanner einer Mikrostereolithografiemaschine. Der Fokus beider Projekte lag in der Verwendung von PC-basierten Systemen für sämtliche Teilaufgaben. Nach Analyse der jeweiligen zeitlichen Anforderungen und technischen Randbedingungen wurde die Entscheidung über den Einsatz von software- oder hardwarebasierten Systemarchitekturen anhand der in Kapitel 3.3 beschriebenen Klassifizierung getroffen.

6.1 Druckkopfansteuerung eines 3D-Druckers

6.1.1 Projektbeschreibung

Beschreibung des dreidimensionalen Druckverfahrens

Im ersten Projekt *Dreidimensionales Drucken von Bauteilen* (3DD) wurde ein Verfahren zur schnellen Modellherstellung (*Rapid Prototyping,* RP) entwickelt. Dabei werden mit piezoelektrischen Druckköpfen flüssige Stoffe auf ein Baufeld dosiert, welche anschließend einen Phasenwechsel erfahren. Dies können z.B. warmschmelzende Wachse, Thermoplaste, Farben oder Teile von Zweikomponentenverbindungen sein. Der Bauprozess erfolgt *schichtweise* und *selektiv*, vergleichbar mit dem Druckvorgang eines Tintenstrahldruckers (Bild 6.1) . Durch abwechselndes Absenken der Bauplattform und Dosieren der jeweiligen Schicht entsteht das dreidimensionale Bauteil. Zur Gestaltung von Überhängen und Absätzen sind Stützstrukturen nötig. Bei entsprechender Modifikation des Verfahrens lassen sich auch Negativformen herstellen, welche in Folgeprozessen z.B. in Metall abgeformt werden.

Die grundsätzliche Technik des 3D-Drucks ist seit mehreren Jahren bekannt. Es existieren verschiedene Methoden, die nach dem Auftragsprinzip in vektor- und rasterbasierte Verfahren eingeteilt sind. Beide Verfahren verfügen über eine Mechanik, deren Zweck die dreidimensionale Positionierung eines oder mehrerer Druckköpfe ist.

---

[71] FORMIKROSYS II: **FOR**schungsverbund **MIKROSYS**temtechnik **II** (1998-2000), in Zusammenarbeit mit universitären und industriellen Partnern, gefördert von der *Bayerischen Forschungsstiftung*.

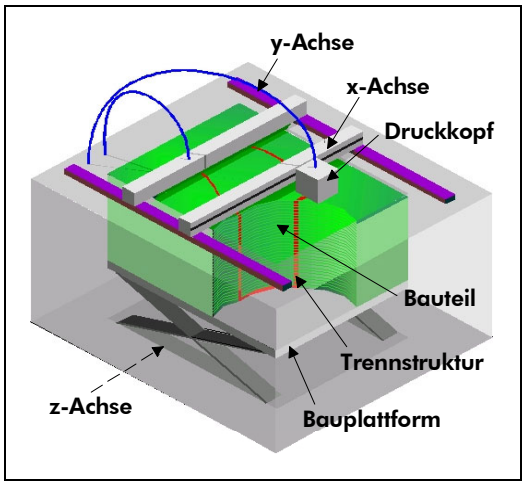


Bild 6.1 Schematischer Aufbau eines 3D-Druckers

Bei einem der ersten 3D-Druckverfahren, dem *3DP™ Process* des amerikanischen MIT (*Massachusetts Institute of Technology*), werden dünne Pulverschichten auf eine Grundplatte aufgebracht, die dann durch gezielte Klebstoffzugabe entsprechend des aktuellen Werkstückquerschnitts verfestigt werden. Der Klebstoff wird tröpfchenweise mittels eines Druckkopfs aufgetragen. Das Baumaterial besteht aus dem gebundenen Pulver. Das lose Pulver übernimmt die Stützfunktion und wird nach dem Prozessende entfernt. Dieses Verfahren wird hauptsächlich zum Bau von Feingussformen verwendet.

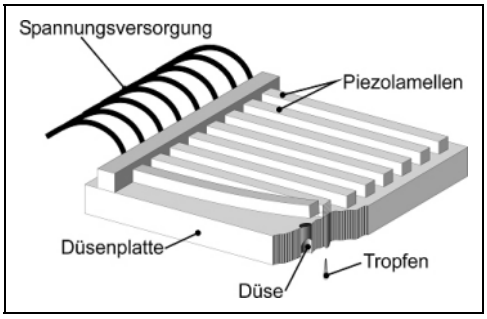


Bild 6.2 Konstruktion eines Mehrdüsen-Druckkopfs

Maschinen mit dem *Multi-Jet-Modeling*-Verfahren spritzen Thermoplast-Tröpfchen mit einem Mehrdüsen-Druckkopf (Bild 6.2) auf die Bauplattform. Das rasterbasierte Verfahren mit Auflösungen von 300 – 400 dpi[72] verwendet eine x-y-Mechanik zur Bewegung des Druckkopfs. Beim *Fused Deposition Modeling*-Verfahren wird das Modell durch Ausscheiden des Werkstoffs aus einer beheizten Düse generiert. Der drahtförmige Kunststoff (ABS) wird der Düse mittels einer Fördereinheit zugeführt und hierin aufgeschmolzen. Die Düsentemperatur wird nur wenige Grad Celsius über der Schmelztemperatur des Werkstoffs gehalten, damit der Werkstoff direkt nach dem Austritt aus der Düse erstarrt. Die bahngesteuerte Bewegung des Düsenkopfes erfolgt durch einen Plottermechanismus. Beim *Ballistic Particle Manufacturing*-Verfahren wird das geschmolzene Material mit einer freibeweglichen Einzeldüse aufgespritzt. Die Düse ist hierbei auf einem Mehr-Achs-Roboter befestigt und kann in drei Dimensionen bewegt werden. Stützkonstruktionen sind überflüssig. Ein anderer Ansatz der Firma SolidScape (ehem. Sanders) basiert auf der Dosierung von zwei verschiedenen warmschmelzenden Materialien über zwei eindüsige Tropfenerzeuger. Das erste Material dient zur

[72] dpi: ***d****ots* ***p****er* ***i****nch*, Punkte pro Inch.

Erzeugung des Werkstückquerschnitts, das zweite zum Aufbau einer Stützstruktur an den notwendigen Stellen. Nach Beendigung der Bauphase wird die Stützstruktur durch Eintauchen in ein Lösungsmittelbad entfernt. Übrig bleibt das fertige Werkstück, das aus unlöslichem Material besteht. Modifikationen des Verfahrens bestehen in der Verwendung des eigentlichen Baumaterials auch für die Stütztechnik. Struktur und Stütze werden durch Sollbruchstellen getrennt. Das zur Speicherung der 3D-Daten verwendete Dateiformat ist meist *STL*[73]. Dieses Format beschreibt Oberflächen und Konturen in Form von Polygonen.

Im Projekt 3DD wurden zwei rasterbasierte Verfahren mit einem Mehrdüsen-Druckkopf entwickelt. Entsprechend dem jeweiligen Anwendungsgebiet wird als Baumaterial Sand oder Wachs eingesetzt. Bild 6.3 zeigt die Funktionsprinzipien der beiden Verfahren.

- **Sanddruckprozess:** Spritzauftrag von flüssigem Härter auf harzbeschichteten Gusssand (Negativdruck) zur Herstellung von Sandgussformen.
- **Wachsdruckprozess:** Spritzauftrag von wasserlöslichem Wachs im Schalendruckverfahren und Massenauftrag eines wasserunlöslichen Baumaterials zur Modellherstellung.

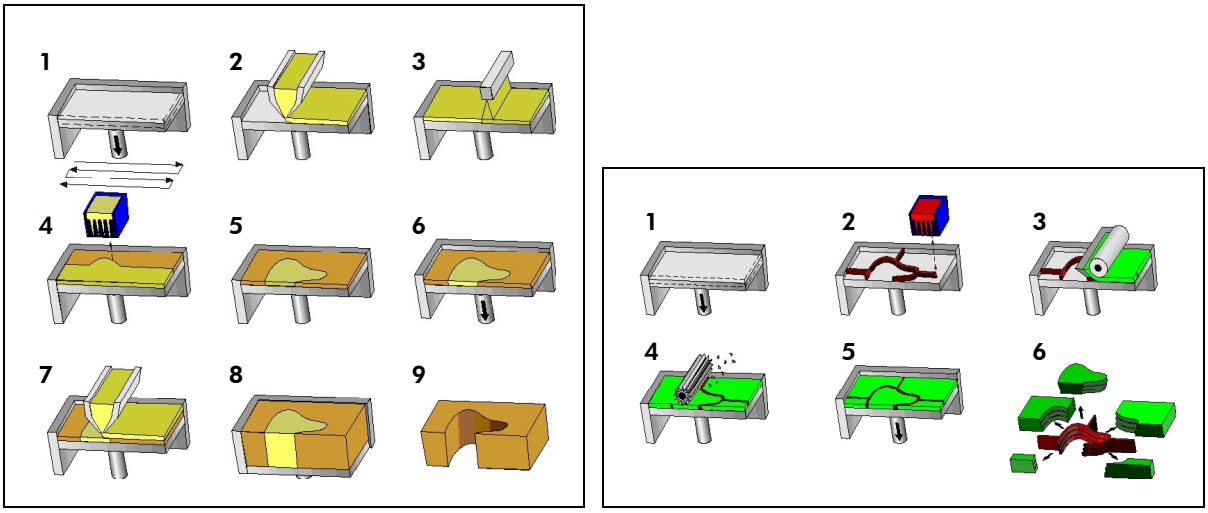


Bild 6.3 Sanddruck- und Wachsdruckprozess

Weitere Anwendungsgebiete, die mit den entwickelten Verfahren realisiert werden können, sind z.B. der Spritzauftrag von Farben für den digitalen Siebdruck, der Spritzauftrag von thermoplastischen Kunststoffen für Funktionsmuster oder Designmodelle und der Spritzauftrag von Fetten zur Kontaktbefettung von Steckverbindern.

Projektziele 3DD

Ziel des 3DD-Projekts war es, ein neuartiges Mikrodosiersystem mit PC-basierten Steuerungskomponenten zu entwerfen und eine prototypische Umgebung (Laborgerät) zur Darstellung des dreidimensionalen Druckprozesses aufzubauen. Die Modulbauweise erlaubt die Untersuchung verschiedenartiger Prozesstechniken. Folgende Entwicklungsziele wurden im Projekt definiert:

- Herstellung eines einfachen und kostengünstigen 3D-Druckers mit einfacher Bedienung und Selbstkalibrierung, speziell für das Büroumfeld

[73] STL: ***S**tandard **T**riangle **L**anguage*, Dateiformat für CAD-Daten, nicht zu verwechseln mit der Abkürzung für *Stereolithografie*.

- Flexible Modulbauweise mit PC-basierten Standard-Komponenten (Steckkarten)
- Verwendung eines Baukastensystems für Antriebe und Antriebssteuerung
- Weitgehende Verwendung der Programmiersprachen C und C++ und dem Standardbetriebssystem Windows NT
- Zeitoptimiertes 3D-Verfahren zur automatischen Generierung der Ansteuerdaten direkt aus den vorhandenen CAD-Modelldaten
- Softwarebasierte Optimierung der Prozessstrategien
- Entwicklung eines Multidüsensystems zur schnellen Modellherstellung
- Erzielung hoher Genauigkeit und Auflösung durch selektiven Tröpfchenauftrag (Düsen) und Einzeldüsenansteuerung
- Auslegung des Dosiersystems für den Einsatz mit unterschiedlichen Materialen: Kunststoff, Wachs, Farbe, Härter
- Integration des Piezodrucksystems und der Ansteuerung zu einem Mikrosystem
- Entwicklung einer Druckkopfheizung für thermoplastische Kunststoffe und Wachse

6.1.2 Analyse der zeitkritischen Prozesse und Aufstellen der Realzeit-Anforderungen

Um die zeitkritischen Funktionen und deren Anforderungen aufstellen zu können, muss der technische Prozess mit seinen Rahmenbedingungen analysiert werden. Dazu wird der Gesamtprozess in mehrere Prozessphasen zerlegt, um die zeitkritischen Teile des Prozessablaufs zu identifizieren. Die Prozessphasen sind:

1. Erzeugung des Modells in der CAD-Applikation
2. Sicherung der Modelldaten in einem programmspezifischen Dateiformat und Umwandlung in das polygonbasiert STL-Format oder direkte Speicherung im STL-Format
3. Übertragung der STL-Daten in den Steuerrechner des 3D-Druckers
4. Generierung der Schichten aus den STL-Daten (*slicing*)
5. Umwandlung der vektorisierten Schichtdarstellung in pixelbasierte *Bitmaps* entsprechend der gewählten Druckauflösung
6. Übertragung der Pixeldaten an die Druckkopfansteuerung
7. Vorbereitungen für die erste Schicht (Initialisierungsvorgänge, Plattformjustage usw.)
8. Vorschub der x-Achse und synchronisierte, zeilenweise Ausgabe der Druckpixeldaten während der Bewegung des Druckkopfs
9. Verstärkung der Signale zur Ansteuerung der piezoelektrischen Aktoren
10. Vorschub der y-Achse (nach jeder Zeile)
11. Absenkung der Bauplattform (z-Achse) um eine Schichthöhe (am Ende jeder Schicht)
12. Herausfahren des fertigen Bauteils nach Bauprozessende

Für eine erfolgreiche Identifikation müssen die Datenraten und –mengen in und zwischen den einzelnen Prozessphasen sowie der jeweilige Zeitbedarf ermittelt werden. Die Abschätzung ergibt:

Zu 1. Die Erzeugung des Modells in der CAD-Applikation erfolgt stets *offline*, d.h. die Konstruktion durch den Anwender erfordert, je nach Komplexität, einen Zeitraum von Stunden bis Tagen.

Zu 2. Die Größe der STL-Datei bewegt sich bei mechanischen Modellen zwischen 0,2 – 2 MB, bei RP-Daten aus Computertomografie-Bildern zwischen 5 – 50 MB.

Zu 3. Die Übertragung erfolgt über wechselbare Medien oder eine Netzwerkverbindung. Nach dem Transfer wird meist eine Überprüfung der Datei auf Fehler bei der Umsetzung der Geometriedaten vorgenommen. Dieser Verifikationsprozess erkennt und behebt Konstruktionsfehler, Umsetzungsprobleme und Beschreibungsfehler. Die Zeitdauer ist von der Anzahl der Fehler, Dateigröße und Rechengeschwindigkeit abhängig und beträgt wenige Minuten bis einige Stunden.

Zu 4. Die dreidimensionalen Modelldaten werden entsprechend der gewünschten Schichtstärke in Schichten zerlegt (geschnitten). Die Einzelschichtdaten werden im Vektorformat gespeichert und ergeben Dateien in der Größenordnung der zugrundeliegenden 3D-Daten. Die Schichtzerlegung dauert, je nach Rechenleistung, Algorithmus und Schichtenzahl, einige Sekunden bis einige Minuten.

Zu 5. Die Umwandlung der Schichtdaten in pixelbasierte *Bitmaps* erfolgt meist in einem Schritt mit Phase 4. Bei einem minimalen Druckraster von 50 μm, einem Baufeld von max. 1 x 1 m^2 und der Speicherung von einem Bit/Pixel beträgt der resultierende Speicherbedarf für eine Schicht ca. 47,7 MB. Mit einer minimalen Schichtdicke von 100 μm und einer max. Bauhöhe von 1 m hätte die Pixel-Datei eine Gesamtgröße von über 460 GB. Diese Datenmenge kann mit Standard-PC-Hardware nicht vernünftig verarbeitet werden; die Bitmap-Generierung muss also *online*, d.h. schritthaltend zum Druckprozess erfolgen.

Zu 6. Gleiches gilt für die Übertragung der Daten zur Druckkopfansteuerung. Es kann beispielsweise die Bitmap einer Schicht berechnet und dann zeilenweise übertragen werden. Die Datenmenge für eine Zeile beträgt ca. 78 KB bei einem Druckkopf mit 32 Düsen.

Zu 7. Sämtliche Initialisierungsvorgänge werden vor Baubeginn abgearbeitet und sind deshalb zeitunkritisch.

Zu 8. Die Auflösung der x-Achse beträgt 5 μm pro Schritt. Während der Druckkopf auf der x-Achse um 10 Schritte (50 μm) verfahren wird, müssen gleichzeitig die Pixeldaten einer Druckspalte (32 Düsen = 4 Bytes) an den Druckkopf weitergeleitet werden. Der Ausstoß-Befehl muss synchron mit dem Überfahren der Druckposition erteilt werden. Bei einer maximalen Frequenz der piezoelektrischen Aktoren von 10 kHz wird eine Verfahrgeschwindigkeit von 0,5 m/s erreicht. Aus der prozessbedingten maximal zulässigen Abweichung des Tropfenausstoßes von $\pm$ 1 μm errechnet sich eine maximal tolerierbare zeitliche Verzögerung von 2 μs. Der Zeitbedarf für den Druck einer kompletten Zeile beträgt ca. 2 s.

Zu 9. Die digitalen Daten einer Druckspalte werden zur Ansteuerung der piezoelektrischen Aktoren mit einem Verstärkerbaustein auf einen Signalpegel von ca. 60 V angehoben. Diese in Analogtechnik realisierte Schaltung hat keine relevante Verzögerungszeit.

Zu 10. Am Ende jeder Druckzeile wird die y-Achse um 32 Druckzeilen weiterbewegt. Da hierbei keine Datenausgabe stattfindet, sind die zeitlichen Anforderungen unkritisch.

Zu 11. Die Absenkung der Bauplattform durch die z-Achse am Ende jeder fertigen Schicht ist unkritisch.

Zu 12. Das Anheben der z-Achse zum Herausnehmen des Bauteils nach Prozessende ist ebenfalls kein zeitkritischer Vorgang.

Aufgrund der durchgeführten Analyse der Prozessphasen erkennt man, dass sämtliche realzeitkritischen Vorgänge direkt mit der Druckdatenausgabe während der Bewegung der x-Achse zusammenhängen. Weiterhin muss die Datenaufbereitung, d.h. die Generierung der Bitmaps, sowie die Übertragung an die Druckeransteuerung *online*, d.h. schritthaltend erfolgen. Die dabei entstehende Datenübertragungsrate von ca. 39 KB/s muss während der gesamten Druckphase gewährleistet werden. Laut Projektbeschreibung soll eine parallele Verwendung von bis zu 6 Druckköpfen möglich sein. Die maximale Datenübertragungsrate steigt auf ca. 234 KB/s (6 x 39 KB/s). Sämtli-

che Ausstoß-Befehle müssen synchron erfolgen. Initialisierungsphasen vor dem Bauprozess und sämtliche Bewegungsphasen nach Zeilenende, Schichtende oder Prozessende stellen keine Real-zeit-Anforderungen. Die Achsantriebssteuerung ist durch eine gemeinsame Hardware zu realisieren, um zu jedem Zeitpunkt die genaue Position der Druckköpfe verfolgen zu können. Alternativ kann eine separate Antriebssteuerung z.B. in Form einer PCI-Steckkarte oder eines SPS-Moduls verwendet werden, wenn eine genaue Positionsrückmeldung, z.B. durch ein separates *Encoder*-Signal, gewährleistet wird.

Ein Zustandsdiagramm der wichtigen Prozessphasen, sowie weiterer parallel ablaufender zeitunkritischer Prozesse zeigt Bild 6.4.

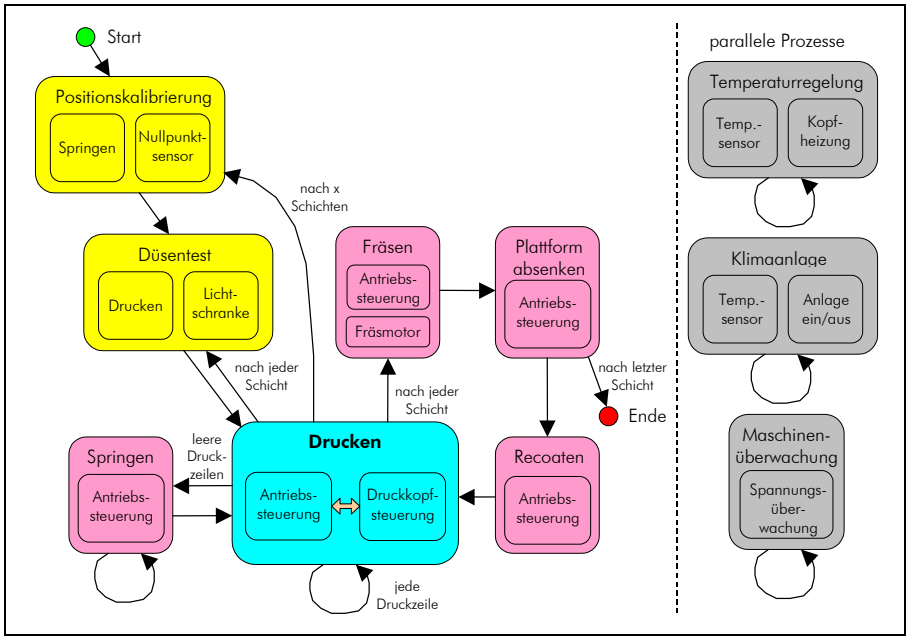


Bild 6.4 Zustandsdiagramm der Prozessphasen (3DD)

Einige der verwendeten Materialen (Thermoplaste und Wachse) benötigen eine Beheizung des Druckkopfs. Die Temperatur muss zwischen 30 – 120 °C einstellbar sein und auf ± 1 °C eingehalten werden. Die Regelung stellt keine zeitkritischen Anforderungen und kann z.B. als unabhängige Zwei-Punkt-Regelung implementiert werden.

Zusammenfassung der zeitlichen Anforderungen:

- Zeitunkritische *offline* Erstellung der CAD-Modelle und Dateien
- Zeitkritische *online* Generierung der Druckdaten (*Bitmaps*)
- Minimale Datenübertragungsrate zur Druckkopfansteuerung: 234 KB/s
- Frequenz der Ausstoß-Befehle: 10 kHz (100 μs)
- Maximale Verzögerung des Tropfenausstoßes: 2 μs
- Zeitintervall für eine komplette Zeile: ca. 2 s
- Synchronisation der Ausstoß-Befehle aller Druckköpfe mit der Bewegung der x-Achse
- Gemeinsame Antriebs- und Druckkopfansteuerung oder enge Kopplung durch Rückspeisung eines Synchronisationssignals
- Unabhängige Zwei-Punkt-Temperaturregelung

6.1.3 Hardware- und Softwarekonzept

Hardwarekonzept

Bei der Erstellung des 3D-Drucker-Prototypen sieht die Projektbeschreibung die weitgehende Verwendung von Standard-PC-Komponenten und des Betriebssystems Windows NT vor. Um ein sinnvolles Hardwarekonzept aufstellen zu können, verwendet man das in Kapitel 3.3 vorgestellte Klassifizierungsdiagramm. Anhand der im vorigen Teilkapitel ermittelten Realzeitanforderungen und Datenübertragungsraten der realzeitkritischen Prozessphasen wird die passende Systemarchitektur gewählt.

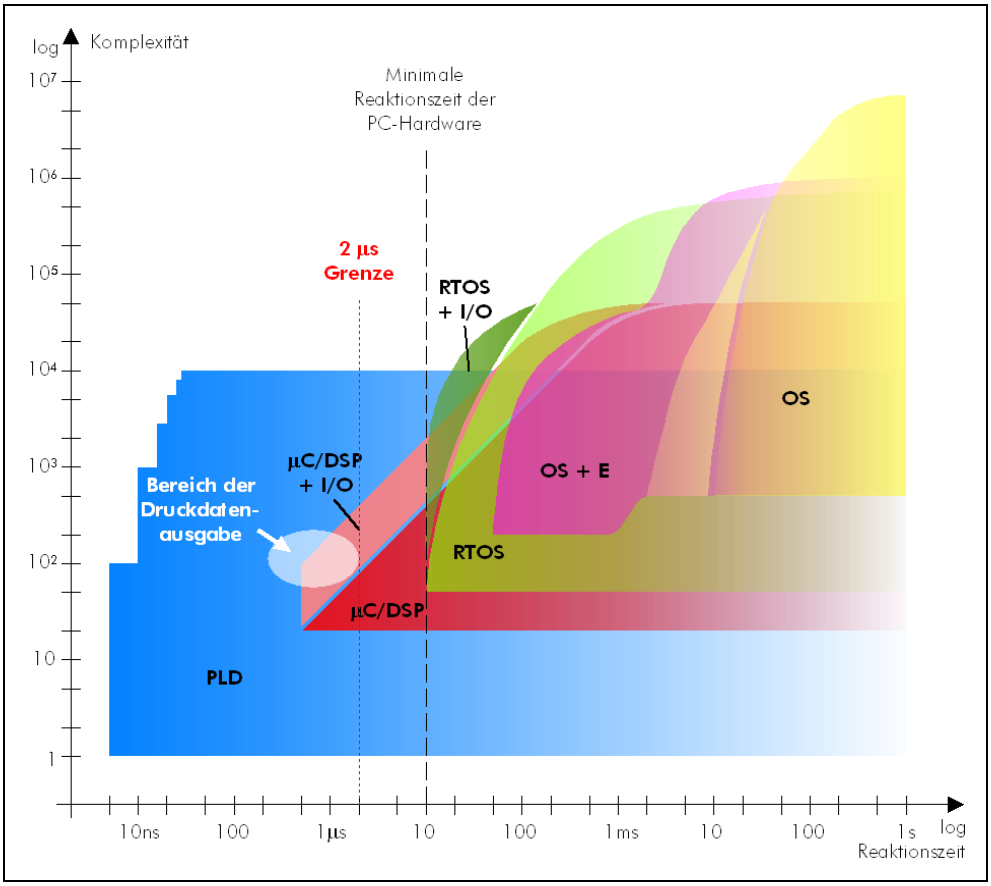


Bild 6.5 Auswahl der passenden Systemarchitektur (3DD)

Die strengsten zeitlichen Anforderungen ergeben sich beim Druckvorgang während der Bewegung der x-Achse. Nach Eintragung der zeitlichen Randbedingungen in Bild 6.5 scheiden reine softwarebasierte Lösungen aufgrund der nicht erreichbaren maximalen Ausgabeverzögerung von 2 μs aus. Die **PLD-Architektur** und die ***µC/DSP + I/O-Architektur*** verbleiben als mögliche Realisierungsmethoden. Bei der in Betracht gezogenen Synchronisierung der Ausstoß-Befehle über die Rückführung eines *Encoder*-Signals, darf das vom Auftreten des Signals bis zur Reaktion des piezoelektrischen Aktors verstreichende Zeitintervall (Reaktionszeit) maximal 2 μs betragen.

Eine vollständige Integration der Druckkopfansteuerung in den Druckkopf ist prototypisch nicht zu realisieren. Es erfolgt eine Aufteilung in Druckkopflogik (**Piezocontroller**) und Ansteuerschaltung (**DSP-Karte**) im PC (Bild 6.6).

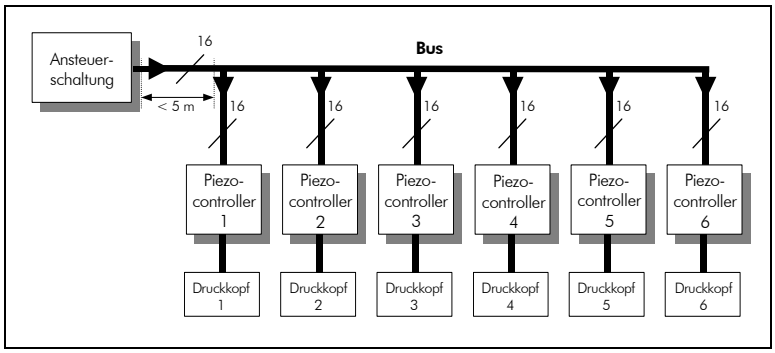


Bild 6.6 Aufteilung in Ansteuerschaltung und 6 Piezocontroller

Da die zu überbrückende Distanz zwischen beiden Schaltungsteilen ca. 5 m beträgt, würde die Kapazität der Leitungen die der piezoelektrischen Aktoren (4 – 7 nF) übersteigen und so die Signalform stark verzerren. Es ist außerdem nicht praktikabel, mehr als 192 Signalleitungen für den projektierten Maximalausbau mit 6 Druckköpfen à 32 Düsen zu verlegen. Hier ist der Einsatz eines Bussystems angezeigt, wobei eine Auswahl der 6 baugleichen Piezocontroller über eindeutig zugewiesene Adressen per 3-bit Konfigurationsschalter vorgenommen wird. Da die synchrone Auslösung des Ausstoß-Befehls nicht über separate Leitungen erfolgen soll, wurde ein einfaches Busprotokoll (Bild 6.7) mit 8 Daten- und 8 Steuerleitungen entwickelt.

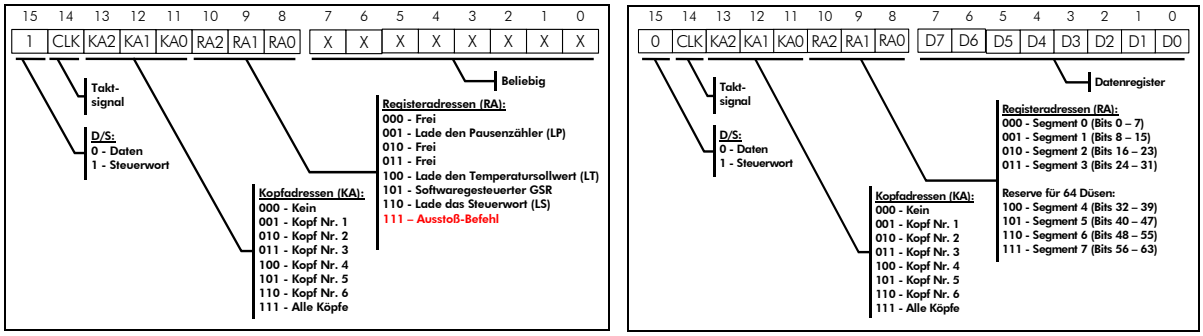


Bild 6.7 Busprotokoll bei Übertragung von Steuerdaten und Druckdaten

Das Vorliegen des Busprotokolls in Form einer schematischen Hardwarebeschreibung vereinfacht die Realisierung in einem CPLD- oder FPGA-Baustein. Aus diesem Grund wird die **PLD-Architektur** als praktikable Lösung ausgewählt. Weitere Funktionen, wie Puls-Pausen-Generator zur Erzeugung der Ansteuersignale für die Aktoren und Zweipunkt-Temperaturregler für die Druckkopfheizung werden zusammen mit dem Bus-Demultiplexer in das FPGA des Piezocontrollers integriert. An zusätzlichen Bauteilen auf der Platine befinden sich nur noch der 32-kanalige Signalverstärker (4 x 8-Kanal-ICs), einige Widerstände und Kondensatoren zur Spannungs-Frequenzwandlung des NTC-Temperatursensors[74] und der Leistungstransistor für den im Druckkopf integrierten Heizwiderstand. Ein Blockschaltbild des Piezocontrollers zeigt Bild 6.8.

[74] NTC: ***N**egative **T**emperature **C**oefficient Thermistor*, Thermistor/Heißleiter: Widerstand mit negativem Widerstandskoeffizienten.

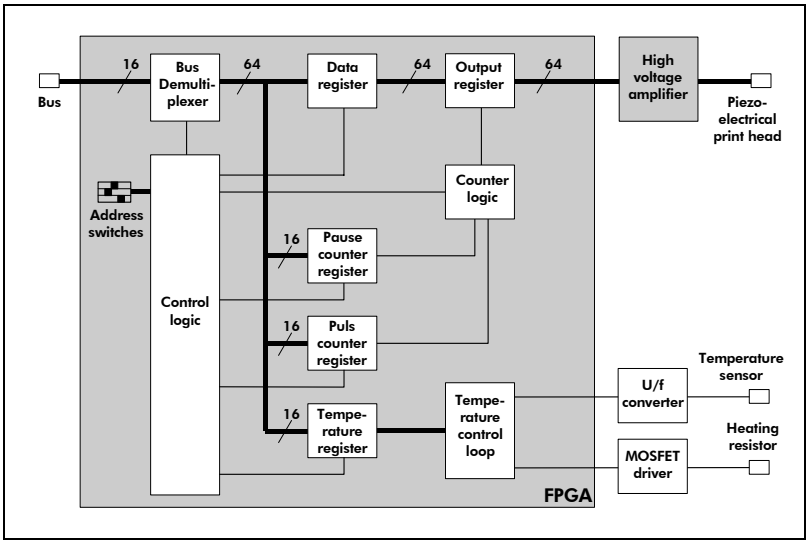


Bild 6.8 Blockschaltbild des Piezocontrollers

Für die Ansteuerschaltung im PC wird eine frei programmier- und erweiterbare DSP-Steckkarte für den PCI-Bus ausgewählt. Sie gewährleistet die benötigte Flexibilität für spätere Optimierungen aufgrund der Prozessergebnisse, stellt hierfür ausreichend Ressourcen (Rechenleistung, Speicher) zur Verfügung und erreicht die erforderliche Datenrate von 234 KB/s. An die vorhandene Erweiterungsschnittstelle der DSP-Karte wird eine Zusatzplatine (**Add-On-Modul**) angeschlossen.

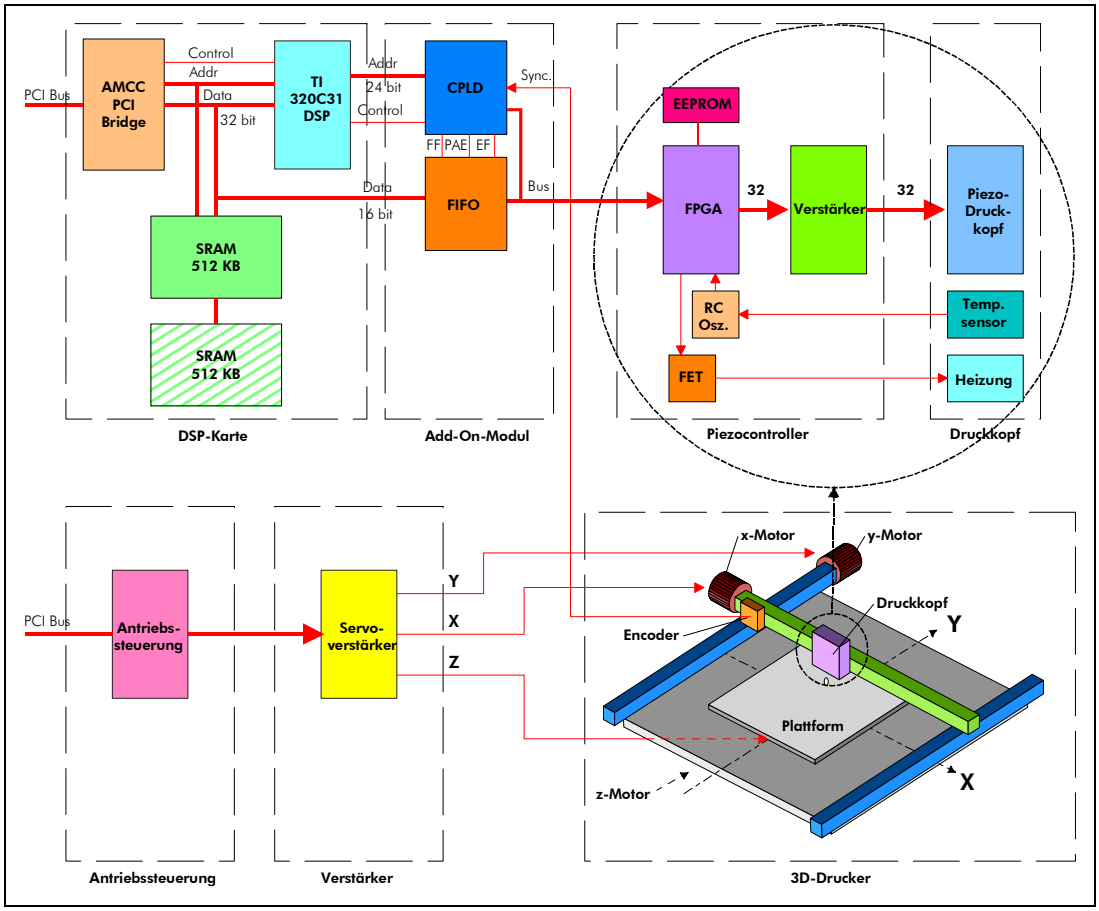


Bild 6.9 Systemdiagramm und Datenfluss des 3D-Druckers

Hier befindet sich das CPLD mit programmiertem Busprotokoll und integrierter Ausstoßbefehl-Synchronisation sowie ein getakteter Daten-FIFO zur Gewährleistung der schritthaltenden Datenausgabe an alle Piezocontroller. Die DSP-Karte ist als *PCI-Busmaster* in der Lage, selbständig neue Druckdaten aus dem PC-Hauptspeicher zu laden, zwischenzuspeichern und bei entsprechendem Füllstand (PAE-Signal[75]) in den FIFO zu übertragen. Auf diese Weise lässt sich unter Einbeziehung von flexiblen Standard-Komponenten und Einhaltung aller Realzeit-Anforderungen eine modulare Druckkopfansteuerung aufbauen (Bild 6.9). Die Bewegung der 3 Achsen erfolgt über eine separate Antriebssteuerung für den PCI-Bus und mehrere Servoverstärker. Die Synchronisationsimpulse liefert ein an der x-Achse montierter *Encoder*.

Softwarekonzept

Nachdem die verschiedenen Teilkomponenten des 3D-Druckers identifiziert und ein passendes Hardwarekonzept aufgestellt wurde, erfolgt die Erstellung eines geeigneten Softwarekonzepts. Es bietet sich an, die modulare Hardwarestruktur auch für die Realisierung der Softwarekomponenten zu verwenden:

3-stufiges Softwarekonzept entsprechend den Realzeitanforderungen:

PC-Software

- CAD-Software
  - Zum Beispiel *Pro/Engineer*, zur Erstellung der Modelle auf einem Windows-PC (dies kann ein separater PC bzw. eine Workstation sein)
- PC-Applikation (auf dem Steuerrechner)
  - Laden und Überprüfen der STL-Dateien
  - Platzieren der Objekte auf dem Baufeld
  - Erzeugen der Schichten und Generieren der Ansteuerdaten (Druck-Bitmaps)

DSP-Software

- DSP-Software
  - Herunterladen der Druckdaten aus dem PC-Arbeitsspeicher
  - Blockweise Übertragung in den FIFO des Add-On-Moduls (Interrupt-Steuerung)
  - Initialisierung und Überwachung des Druckprozesses (Temperatur, Puls-Pausen-Verhältnis, Not-Aus usw.)

PLD-Firmware

- CPLD-Firmware des Add-On-Moduls
  - Steuerung des FIFOs (*full, almost empty, empty*)
  - Synchronisierung des Ausstoß-Befehls mit dem *Encoder*-Impuls der x-Achse
  - Generierung des Busprotokolls (Multiplexer) für die Übertragung von Daten und Befehlen zum Piezocontroller
- FPGA-Firmware des Piezocontrollers:

[75] PAE: ***P****rogrammable* ***A****lmost* ***E****mpty*, programmierbares Signal, welches bei Unterschreiten eines bestimmten Füllstandes (hier 127 Einträge) ausgelöst wird.

- Realisierung des Busprotokolls (Demultiplexer)
- Steuerung des Puls-Pausen-Verhältnisses für den Druckkopf
- Triggerung der Datenausgabe an den Signalverstärker bei Empfang des Ausstoß-Befehls
- Zweipunkt-Regler-Algorithmus für die Temperaturregelung der Druckkopfheizung

6.1.4 Realisierung

Die Realisierung des Projekts wurde in drei parallele Teilaufgaben entsprechend der Kernkompetenzen der drei beteiligten Forschungsgruppen aufgegliedert. Eine Teilaufgabe war die mechanische Konstruktion und der Aufbau des Prototypen mit Antrieben, Leistungselektronik und Stromversorgung. Mit einer vorhandenen einfachen Steuerung konnten bereits im Frühstadium erste Prozessuntersuchen begonnen werden. Die zweite Aufgabe beinhaltete die Erstellung der PC-Applikation, der Datengenerierungsalgorithmen und die Schaffung einer anwenderfreundlichen, grafischen Benutzerschnittstelle. Die dritte Aufgabe war die Entwicklung der Datenkommunikation zwischen Steuerungs-PC (Industrie PC) und dem 3D-Drucker. Hierzu war die Programmierung der DSP-Software sowie die Entwicklung des Add-On-Moduls, des Piezocontrollers und der entsprechenden Firmware vorzunehmen. Den endgültigen schematischen Aufbau der 3D-Drucker-Hardware zeigt Bild 6.10.

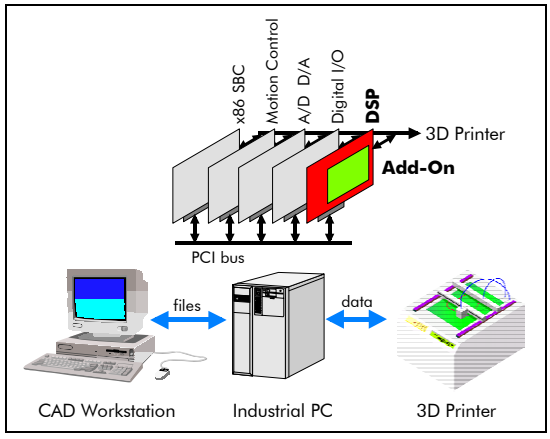


Bild 6.10 Schematischer Aufbau der 3D-Drucker-Hardware

Industrie-PC

Der Steuerungs-PC besteht aus einem Industrie-PC mit aktiver 7-Slot PCI-Bus-Backplane und einem Einplatinenrechner[76] mit Intel Celeron Prozessor (400 MHz). Das Gehäuse beherbergt die verschiedenen Steckkarten für Antriebssteuerung, analoge Messwerterfassung und Signalausgabe (A/D, D/A), digitale Ein- und Ausgänge (Digital I/O) und die DSP-Karte mit aufgestecktem Add-On-Modul. Die mitgelieferten Windows NT-Treiber vereinfachen die Programmierung und Systemintegration erheblich. Außerdem ergeben sich durch den aktiven Wettbewerb der Hersteller von PCI-Bus Steckkarten moderate Preise und ein vielfältiges Produktspektrum. Damit kann der Fokus auf die Entwicklung der realzeitkritischen Komponenten gelegt werden.

[76] Auch SBC: ***S**ingle **B**oard **C**omputer*.

DSP-Karte und Add-On-Modul

Die DSP-Karte wurde von der Firma Sheldon Instruments[77] bezogen und integriert den digitalen Signalprozessor (DSP) TMS320C31 der Firma Texas Instruments, 1 MB Speicher (SRAM) und eine PCI-Bus Schnittstelle auf einer Steckkarte (Bild 6.11). Der PCI-Bus Schnittstellenbaustein (AMCC S5933) bietet bidirektionale Busmaster-Fähigkeiten. Auf der Platine befindet sich weiterhin der Steckplatz für das Add-On-Modul.

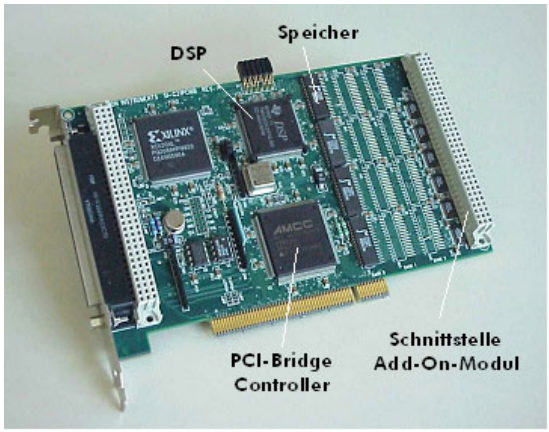


Bild 6.11 DSP-Karte

Die Aufgaben der DSP-Karte, entsprechend dem Konzept aus Kapitel 6.1.3, sind das eigenständige Herunterladen der Druckdaten vom PC-Arbeitsspeicher in den DSP Speicher als Hintergrundprozess (PCI-Busmaster Funktionalität), die Kontrolle der FIFO-Statussignale des Add-On-Moduls und blockweise Übertragung neuer Druckdaten in den FIFO, und die Initialisierung und Überwachung des Druckprozesses.

Das Add-On-Modul (Bild 6.12) wird über zwei Steckerleisten mit der DSP-Karte verbunden. Es besteht aus einem CPLD (Xilinx XC95144), dem FIFO (Cypress CY7C4245 4Kx18), Bustreiberbausteinen und Diagnose-LEDs.

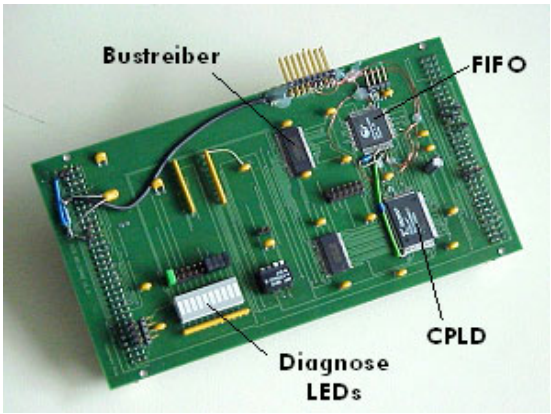


Bild 6.12 Add-On-Modul

Dieses intelligente Schnittstellenmodul garantiert den Datenfluss mit niedriger Latenzzeit und niedrigem Jitter für die realzeitkritische Steuerung des Piezocontrollers. Wichtigste Aufgabe ist die Ge-

[77] http://www.sheldoninst.com/

nerierung des Ausstoß-Befehls bei Empfang des Synchronisationsimpulses der x-Achse. Das dem Ausstoß-Befehl entsprechende Bitmuster wird im Piezocontroller separat dekodiert um Verzögerungszeiten zu minimieren. Piezocontroller und Add-On-Modul sind die beiden einzigen proprietären Entwicklungen des Projekts, welche nicht durch Standardkomponenten realisiert werden konnten. Die hohe Integration der Platinen ermöglicht aber verhältnismäßig niedrige Produktionskosten (ca. 150 €/Modul).

Software des Industrie-PCs und der DSP-Karte

Windows NT 4.0 dient als Betriebssystem des Steuerungs-PCs. Die Steuerungsapplikation wurde mit der Visual C++ 6.0 Entwicklungsumgebung von Microsoft erstellt. Alle zeitunkritischen Programmteile, wie Lade-Dialog der STL-Dateien, Überprüfungsalgorithmen und interaktives grafisches Platzierungsmodul sind als Threads programmiert und werden unabhängig vom Druckprozess ausgeführt. Der Druckprozess ist als höherpriorer Thread gestaltet und übernimmt die Bitmap-Generierung der aktuell zu druckenden Schicht. Nach Fertigstellung einer Schicht und Speicherung der Druckmatrix im Hauptspeicher wird die DSP-Karte benachrichtigt. Diese lädt Konfigurationsdatensatz und erste Druckzeile in das SRAM, initialisiert Add-On-Modul (CPLD und FIFO) und Piezocontroller und füllt anschließend den FIFO mit den ersten Druckdaten. Jetzt wird ein Signal an die wartende Steuerungsapplikation des PCs gesendet und die Bewegung der x-Achse durch die Antriebssteuerung gestartet. Während der Bewegung des Druckkopfs triggern die Synchronisationsimpulse des *Encoders* das Add-On-Modul. Dieses sendet umgehend den Befehl für den Tröpfchenausstoß an den/die Piezocontroller. Die DSP-Karte erledigt das Nachladen des FIFOs im Hintergrund. Nach Beendigung einer Druckzeile wird die y-Achse zur nächsten Zeile verfahren und der Ablauf beginnt erneut. Die PC-Applikation berechnet währenddessen die Druckmatrix der nächsten Schicht, sobald wieder ausreichend Hauptspeicher zur Verfügung steht. Sind am Ende einer Schicht noch weitere Schichten zu drucken wird der Vorgang von der DSP-Karte bis zur Fertigstellung des Modells wiederholt. PC-Applikation und DSP-Karte können im laufenden Prozess Nachrichten austauschen (Status, Druckfortschritt usw.). Somit kann der Anwender den Prozess über die grafische Schnittstelle (Bild 6.13) jederzeit anhalten und wieder starten. Im Fall eines PC-Systemfehlers kann nach Neustart die Abbruchstelle ermittelt und der Bauprozess lückenlos fortgesetzt werden. Prozessbedingt empfiehlt sich aber ein Abfräsen der unvollständigen Schicht und ein Neustart des Druckprozesses am Beginn der entsprechenden Schicht.

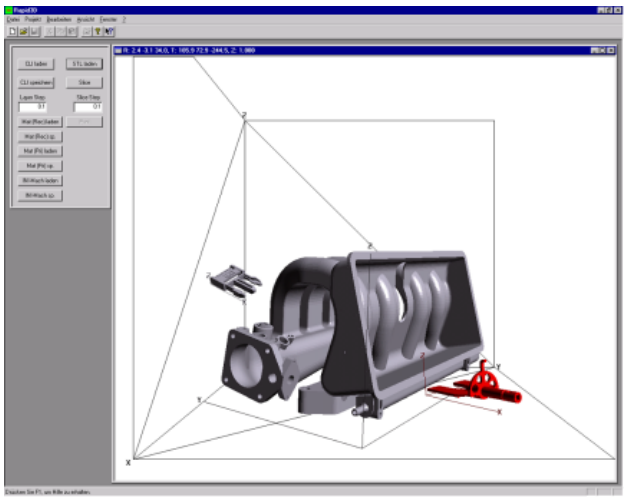


Bild 6.13 *Screenshot* der Steuerungsapplikation (3DD)

Der DSP hat mehrere parallele Aufgaben zu bearbeiten. Einige von ihnen sind zeitkritisch; andere dienen lediglich Verwaltungsaufgaben. Das Laden der Druckdaten in den FIFO des Add-On-Moduls ist der wichtigste Prozess. Es muss sichergestellt werden, dass keine Verzögerungen auftreten, welche den Datenstrom zum Piezocontroller abreißen lassen könnten[78]. Hierzu wurde der kleine Realzeit-Kernel *MicroC/OS-II* (siehe Kapitel 4.3) eingesetzt, um die Programmierung der parallelen Tasks und die Vergabe der Prioritäten zu erleichtern. Ein wesentlicher Vorteil des Kernels ist die Verfügbarkeit des C-Sourcecodes, die einfache Portierung[79] auf andere Mikroprozessor- und DSP-Architekturen und die lizenzfreie Verwendung für Universitäten[80]. MicroC/OS-II ist, bis auf wenige systemspezifische Funktionen, fast vollständig in ANSI-C geschrieben. Alle wichtigen Eigenschaften eines Realzeit-Betriebssystems, wie preemptives Multitasking mit 64 Prioritäten, Kommunikationsmechanismen (Warteschlangen, Mailboxen, Semaphoren) sowie Task-, Speicher- und Interrupt-Verwaltung sind vorhanden. Der Speicherbedarf des Kernels beträgt nur ca. 6 – 15 KB.

Die Programmierung und Compilierung der DSP-Software erfolgten mit den *Code Generation Tools* der Firma Texas Instruments. Sie beinhalten C-Compiler, Assembler, Optimierer, Emulator, Simulator und Hardware Debugger. Einige Register des DSPs, z.B. das Interrupt-Kontroll-Register, können nur im Assembler-Code angesprochen werden, da sie nicht *memory mapped* sind. Auch die Initialisierung der Interrupt-Vektor-Tabelle muss mit Assembler-Code gemacht werden. Die Firmware der DSP-Karte liegt im COFF-Format auf dem Steuerungs-PC und wird bei Start der 3D-Druckeranwendung automatisch über den PCI-Bus in den DSP geladen und ausgeführt. Die Kommunikation mit der PC-Anwendung findet über Mailbox-Register des PCI-Controllers statt.

Firmware des Add-On-Moduls und des Piezocontrollers

Die Funktionen des CPLDs auf dem Add-On-Modul wurden in der Beschreibungssprache *ABEL* definiert. Für kleine und mittlere Schaltungen, wie in diesem Projekt, bietet sie Vorzüge hinsichtlich Implementierungsgeschwindigkeit und Einfachheit gegenüber abstrakten Sprachen wie VHDL oder Verilog. Einfach strukturierte Zustandsautomaten realisieren Busmultiplexer, Befehlsausgabe (z.B. Ausstoß-Befehl) und Steuerung des FIFO-Füllstands. Das Add-On-Modul ist an Adress- und Datenbus des DSP angeschlossen und kann über einen Speicherbereich angesprochen werden. Ereignisse meldet das Add-On-Modul der DSP-Karte durch Auslösen eines Interrupts.

Die Firmware des Piezocontroller-FPGAs erfolgte mit dem Xilinx Entwicklungstool *Foundation*. Gerade die umfangreichen logischen und zeitlichen Simulationsfähigkeiten ermöglichten eine frühe Verifikation der Schaltung. Auch für die Aufdeckung und Optimierung von zeitkritischen Pfaden bei der Logik zur Dekodierung des Ausstoß-Befehls war die Software sehr nützlich. Grafische (Schematische Logik, Zustandsdiagramme) und sprachliche (VHDL, Verilog) Eingabemöglichkeiten ermöglichen eine anwenderfreundliche Entwicklung.

Die Schaltung des FPGAs erledigt folgende Aufgaben: Demultiplexen des Busprotokolls, Zwischenspeicherung einer Druckspalte, Generierung des Puls-Pausen-Verhältnisses der Ansteuerimpulse für die piezoelektrischen Aktoren, Temperaturregelung der Druckkopfheizung mittels Zwei-Punkt-Regler und direkte Dekodierung des Ausstoß-Befehls.

---

[78] Diese Bedingung ist vergleichbar mit der Vermeidung des *„buffer underrun"* beim Brennen von CD-ROMs.
[79] In diesem Projekt wurde der Kernel in kurzer Zeit auf den TMS320C31 DSP portiert.
[80] Bei kommerziellem Einsatz fallen geringe Lizenzgebühren an.

6.1.5 Verifikation und Ergebnisse

Im Rahmen des Projekts wurden verschiedene 3D-Drucker Prototypen aufgebaut. Nach der Integration aller Hard- und Softwarekomponenten wurde die Einhaltung der Spezifikation durch Vermessung der Ausführungszeiten der zeitkritischen Funktionen von DSP-Karte, Add-On-Modul und Piezocontroller verifiziert.

DSP-Karte

Der verwendete DSP TMS320C31 leistet ca. 30 MIPS bei einer Taktfrequenz von 60 MHz. Der Zyklenbedarf wichtiger Betriebssystemfunktionen wurde theoretisch ermittelt. Mit einem Logikanalysator erfolgten die Messungen der tatsächlichen Ausführungszeiten. Tabelle 6.1 zeigt die Übereinstimmung von Berechnung und Messung.

Funktion	Zyklen (1 Zyklus = 2 Takte = 33,3 ns = ca. 30 MIPS)	Zeit [µs]
Interrupt latency time	12 = 5 + 1 + 1 + 1 + 4 (longest instruction + disable ints. + read int. vector + store return addr. + pipeline fill)	0,4
Interrupt response time	40 = 12 + 28 (int. latency + save context)	1,33
Save, restore context	28	0,93
`OSIntEnter()`	19	0,63
`OSIntExit()`	41 bis 85 = 45 + 8 + 28 + 4 (scheduling? + load new stack pointer + restore context + reti)	1,36 – 2,83
`OSSched()`	39 bis 110 = 38 + 5 + 28 + 7 + 28 + 4 (scheduling? + trap + save context + load new stack pointer + restore context + reti)	1,30 – 3,66
`OSTickISR()`	130 + (Taskanzahl x 40)	5,66 (1 Task)

Tabelle 6.1 Ausführungszeiten wichtiger Funktionen

Der Speicherbedarf von Realzeit-Kernel und DSP-Anwendung wurde getrennt ermittelt (Tabelle 6.2). Ingesamt werden ca. 23 KB belegt. Der Rest des DSP-Speichers von ca. 1000 KB (1 MB – 23 KB) steht für Druckdaten zur Verfügung.

	Code [Bytes]	Daten [Bytes]
MicroC/OS-II	8304	3464
DSP-Anwendung	5836	5476

Tabelle 6.2 Speicherbedarf von Kernel und Anwendung

Add-On-Modul

Bei der Synthese der CPLD-Hardwarebeschreibung des Add-On-Moduls musste der Compiler für einige Signale bis zu drei Hierarchieebenen benutzen (siehe auch Kapitel 5.1.2). Durch geschickte Priorisierung der zu synthetisierenden Logikblöcke benötigen alle reaktionszeitsensitiven Signale nur eine einzige Hierarchiestufe. Die Pin-zu-Pin-Verzögerungszeit des CPLDs ist mit 7,5 ns spezifiziert. Die *worst case*-Latenzzeit der Schaltung beträgt 3 x 7,5 ns = 22,5 ns.

Das Diagramm in Bild 6.14 zeigt die Übertragung einer Druckspalte (4 Bytes = 32-Bit für 32 Düsen) auf dem Bus vom Add-On-Modul zum Piezocontroller. Die Übertragungszeit für ein Byte be-

trägt 1,32 μs, die gesamte Zeit für vier Bytes inkl. *Overhead* addiert sich auf 6,32 μs. Daraus errechnet sich eine (theoretische) Baudrate von ca. 5 MBit/s.

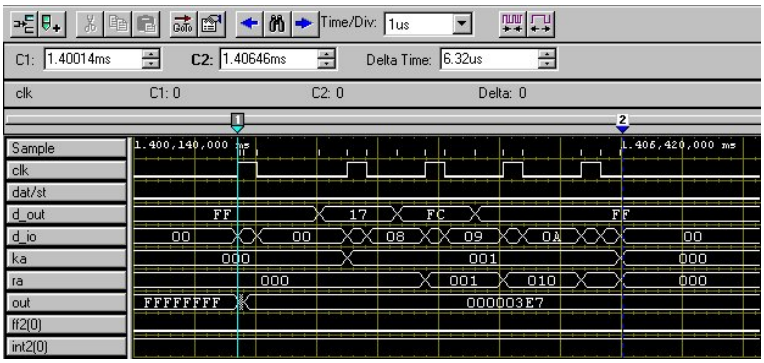


Bild 6.14 Übertragung einer Druckspalte (4 Bytes)

Bild 6.15 zeigt die Übertragung von Daten für einen Druckkopf mit 5 kHz Ausstoß-Frequenz (Intervall 200 μs) während des Druckprozesses.

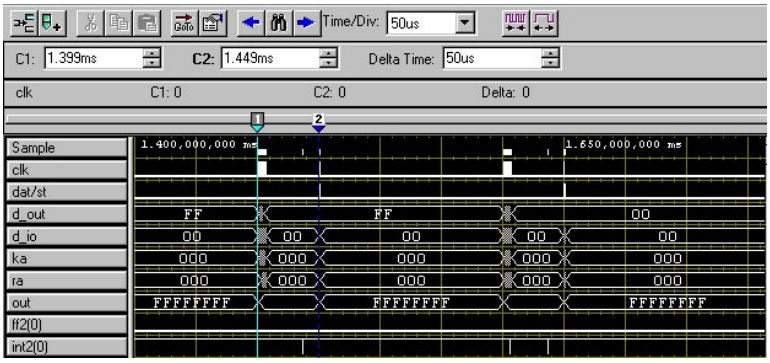


Bild 6.15 Datenübertragung für 5 kHz Ausstoß-Frequenz

Zusammenfassung

Die Reaktionszeit vom Auftreten des Synchronisationsimpulses des *Encoders* bis zum Ausstoß der Tröpfchen beträgt < 1,6 μs. Sie setzt sich zusammen aus:

- Verarbeitungszeit im CPLD des Add-On-Moduls, Generieren des Ausstoß-Befehls (< 50 ns)
- Busübertragungszeit für den Befehl (1 Byte) inkl. *Overhead* (ca. 1,5 μs)
- Dekodierung des Befehls im FPGA des Piezocontrollers und Freigabe der Ausgangstreiber des Druckkopfs (< 50 ns)

Somit liegt die Reaktionszeit im Bereich der geforderten 2 μs. Alle weiteren spezifizierten Anforderungen des 3D-Druckprozesses wurden erreicht. Das Konzept der weitgehenden Verwendung von Standardkomponenten in Verbindung mit wenigen proprietären Modulen führte zu kostengünstigen Prototypen. Die Flexibilität der Soft- und Firmware ermöglicht spätere Erweiterungen, z.B. die Verwendung von Druckköpfen mit 64 Düsen.

Prototyp und Musterstücke

Zur Herstellung von Sandgussformen wurden zwei Prototypen eines 3D-Druckers für den Sanddruckprozess entwickelt und aufgebaut. Bild 6.16 zeigt den ersten Prototyp mit Schrittmotoren und den zweiten Prototyp mit Servomotoren für die Bewegung der x- und y-Achse.

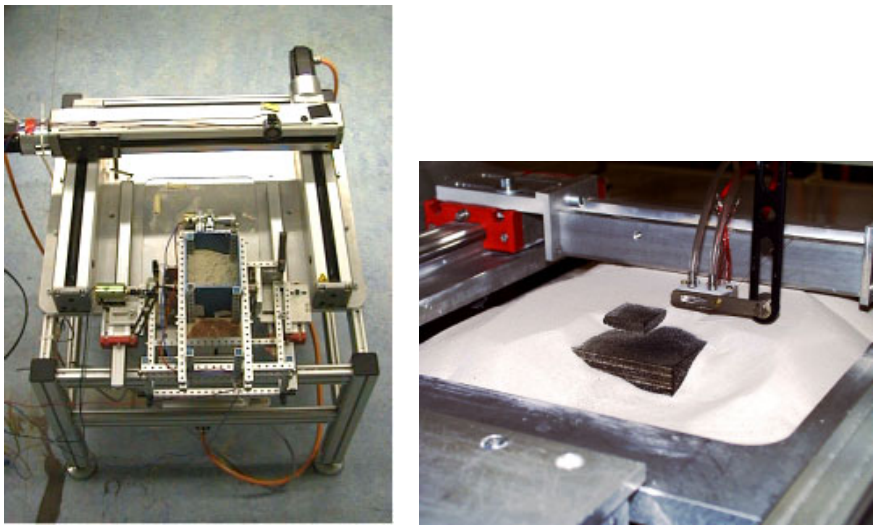


Bild 6.16 Prototypen des 3D-Druckers für den Sanddruckprozess

Mit diesen Maschinen wurden verschiedene Musterstücke (Positiv- und Negativdruck) erzeugt, um Prozessparameter wie z.B. Auflösung, Bauzeiten und Genauigkeit zu untersuchen (Bild 6.17). Für die Erzeugung der Ansteuerungsdaten sind CAD-Modelle verwendet worden.

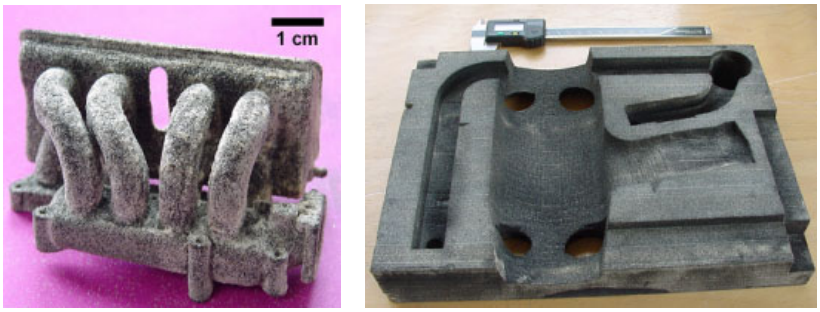


Bild 6.17 Musterstücke (Sanddruckprozess)

Für den Wachsdruckprozess wurde ein weiterer 3D-Drucker-Prototyp aufgebaut. Der Bauprozess besteht hierbei aus dem Drucken der Bauteilberandung mit einem wasserlöslichen Trennwachs (dunkel) und dem Massenauftrag des eigentlichen Baumaterials (hell) mittels *Recoating*. Bild 6.18 zeigt den 3D-Drucker, Bild 6.19 einen Testdruck und ein Musterstück nach dem Auflösen des Trennwachses.

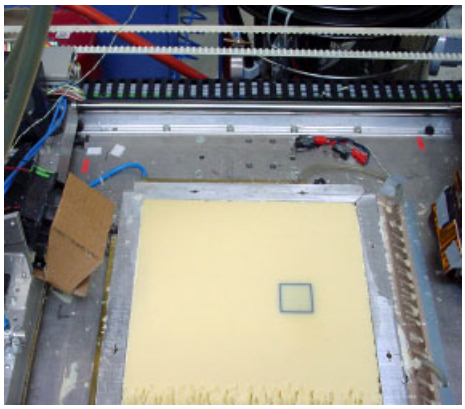
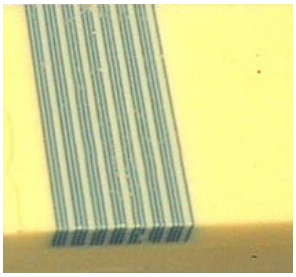


Bild 6.18 Prototyp des 3D-Druckers für den Wachsdruckprozess



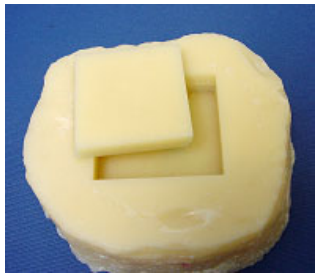


Bild 6.19 Testdruck und Musterstück für Separation (Wachsdruckprozess)

6.2 Steuerung der X-Y-Scanner einer Mikro-Stereolithografiemaschine

6.2.1 Projektbeschreibung

Beschreibung des Stereolithografie-Verfahrens

Das zweite Projekt *Mikro-Stereolithografie* (MSTL) beschäftige sich mit der Weiterentwicklung des Stereolithografieverfahrens (STL) zur Erzeugung von Mikrobauteilen. Ziel ist die Herstellung von Kunststoffbauteilen durch schichtweise Generierung dreidimensionaler Strukturen direkt aus den Daten des CAD-Modells, ähnlich dem in Kapitel 6.1 beschriebenen Projekt *Druckkopfansteuerung eines 3D-Druckers*. Mit aktuellen Stereolithografieverfahren werden Strukturbreiten von ca. 200 μm und eine Positioniergenauigkeit des Laserstrahls von $\pm$ 50 μm realisiert. Für die Herstellung von Mikrobauteilen sind diese Auflösungen zu gering. Aufgabe war nun, dieses Verfahren so weiterzuentwickeln, dass Bauteile mit den in der Mikromechanik üblichen Anforderungen an Strukturauflösung und Genauigkeit hergestellt werden können. Der anvisierte Prototyp der MSTL-Maschine musste hierzu eine x-y-Auflösung von 10 μm mit einer Genauigkeit von $\pm$ 2 μm und eine Schichtdicke von 20 μm erreichen. Diese Mikrobauteile können dann entweder direkt genutzt oder mit entsprechenden Verfahren, z.B. dem Kunststoffspritzguss, reproduziert werden.

Bei der STL wird eine Aufzugmechanik in einen Behälter mit flüssigem Photopolymer konstanter Füllhöhe eingebracht (Bild 6.20). In einem ersten Schritt wird über der abgesenkten Plattform ein Polymerfilm in gewählter Dicke durch einen Wischer (*recoater*) aufgebracht. Der photosensitive Kunststoff wird anschließend selektiv durch Belichtung mit einem UV-Laserstrahl ausgehärtet.

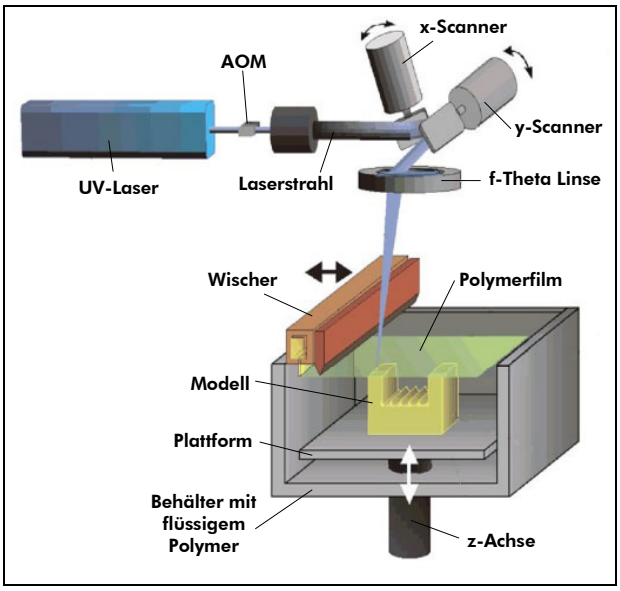


Bild 6.20 Stereolithografie-Prinzip

Der von einem rechnergesteuerten x-y-Scanner[81] abgelenkte Laserstrahl zeichnet hierzu die Konturlinien bzw. die Innenbereiche einer Bauteilschicht, Schraffur oder *Hatch* genannt, auf der Polymeroberfläche nach (Bild 6.21). Die Unterbrechung des Laserstrahls wird mit einem *akusto-optischen Modulator* (AOM) realisiert. Anschließend wird die Plattform im Polymerbad um eine Schichtdicke abgesenkt, eine frische Kunststoffschicht aufgebracht und wieder belichtet. Dieser Vorgang wird bis zur Vollendung des Bauteils wiederholt. Das fertige Modell entnimmt man dann dem Bad, reinigt es von Polymerresten, und entfernt die Stützstrukturen. Ein anschließender Nachhärtungsprozess unter einer UV-Lampe bringt das Bauteil auf Endfestigkeit. Die maximale Baufeldgröße beträgt ca. 600 x 600 mm^2 auf. Die Maßhaltigkeit der Bauteile wird zusätzlich durch verfahrensbedingte Störungen, wie z.B. Verzüge des Bauteils oder Schwankungen der Schichtdicke, beeinflusst, welche durch optimierte Prozessstrategien minimiert werden können.

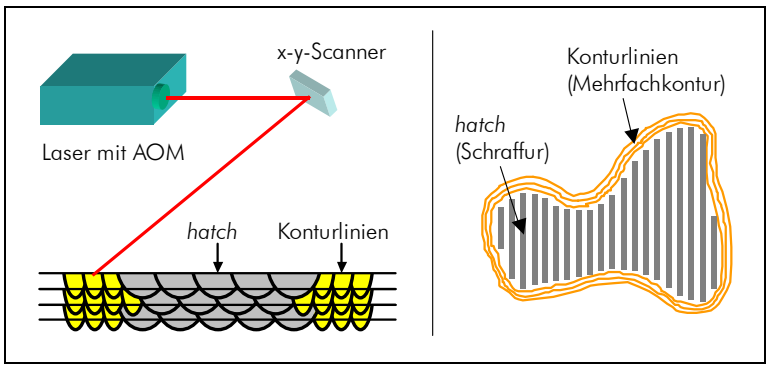


Bild 6.21 Belichtung von Konturlinien und *Hatch*

Eignung des Verfahrens zur Erzeugung von Mikrobauteilen

Durch die Verwendung eines stärker fokussierten UV-Laserstrahls besteht die Möglichkeit, den schreibenden Laserspotdurchmesser auf wenige Mikrometer zu begrenzen. Neue Photopolymere

[81] Scanner: Auf einer Achse beweglich gelagerter Spiegel, der mit einer stromdurchflossenen Spule gedreht werden kann. Die Regelung der Ablenkung erfolgt über einen geschlossenen Regelkreis durch Rückführung eines Positionssignals (Winkelgeber).

weisen eine sehr scharf umrissene und im wesentlichen von der eingestrahlten Energie abhängige Härtung auf, sodass von einer nur unwesentlichen Spurvergrößerung durch Randhärtungserscheinungen ausgegangen werden kann. Die Tiefenhärtung lässt sich bei diesen neuen Polymeren nahezu linear mit der Laserleistung einstellen. Im Bezug auf die vorhandenen Materialien sind alle Voraussetzungen für die Mikrostrukturierung gegeben. Außerdem bietet die Scannertechnik beste Voraussetzungen, um den Laserstrahl exakt abzulenken. Die Beschichtungstechnik für den dünnen Polymerfilm sowie die realzeitkritische präzise Ansteuerung der Laserscanner stellen bei diesem Projekt die größten Herausforderungen dar.

Andere Verfahren

Beim *COLAMM*-Verfahren der japanischen Firma Mitsui wird ein rundum geschlossener Polymerbehälter verwendet. Die Belichtung erfolgt durch eine Glasplatte. Der Vorteil liegt in einer unproblematischen Beschichtung, da sich die richtige Schichtstärke automatisch einstellt. Auf der anderen Seite muss das Ablösen des Modells von der Glasplatte nach jedem Belichtungsvorgang gewährleistet werden. Gerade bei filigranen Strukturen und geringen Schichtdicken ist dies mit großen Schwierigkeiten verbunden. Zwar ist eine Anti-Haftbeschichtung der Glasplatte denkbar, aber die Kräfte, die im entstehenden Spalt zwischen Modell und Platte beim Ablösen auftreten, können nur durch Verformung der Platte oder des Modells kompensiert werden. Auch die Belichtung wird durch die Glasplatte erschwert, da abhängig vom Einfallswinkel ein zu berücksichtigender Strahlversatz entsteht. Das *Solid Ground Curing*-Verfahren (SGC) der israelischen Firma Cubital verwendet für den Härtungsprozess nicht einen gebündelten Laser, sondern UV-Lampen, welche die Schichtstruktur durch eine Maske belichten. Der große Vorteil dieses Maskenprojektionsverfahren ist der prinzipbedingt nicht vorhandene Strahlversatz. Es wurde vorgeschlagen, die Masken durch digital ansteuerbare Flüssigkristall-Displays (LCDs[82]) zu ersetzen. Allerdings besitzen aktuell erhältliche LCDs weder eine ausreichende Auflösung für die digitale Maskengenerierung, noch haben sie einen ausreichenden Kontrast. Die weitere Entwicklung sollte jedoch beobachtet werden, da bei diesem Verfahren auf die aufwändige und teure Scannereinheit verzichtet werden kann.

Projektziele MSTL

Die Voruntersuchungen ergaben, dass es prinzipiell möglich ist, die Stereolithografie zur Generierung von Mikrostrukturen weiterzuentwickeln. Ziel des Projekts war deshalb die Herstellung des Prototyps einer rechnergesteuerten Mikro-Stereolithografiemaschine zur Darstellung des Belichtungs- und Bauprozesses für Mikrobauteile sowie zur Herstellung von ersten Musterstücken. Folgende Entwicklungsziele wurden im Projekt definiert:

- Herstellung einer Mikrostereolithografiemaschine zur Herstellung von Mikroteilen
- Flexible Modulbauweise mit PC-basierten Standardkomponenten (Steckkarten)
- Entwicklung einer geeigneten Belichtungssteuerung für die Laserscanner
- Verwendung eines Baukastensystems für Antriebe und Antriebssteuerung
- Weitgehende Verwendung der Programmiersprachen C und C++ und dem Standardbetriebssystem Windows NT
- Zeitoptimiertes 3D-Verfahren zur automatischen Generierung der Ansteuerdaten direkt aus den vorhandenen CAD-Modelldaten
- Softwarebasierte Optimierung der Prozessstrategien

[82] LCD: ***L****iquid* ***C****rystal* ***D****isplay*.

- Entwicklung und Aufbau eines Optikmoduls (f-Theta-Linse), einschließlich geeigneter Kalibriereinrichtungen
- Belichtung der Umrandungsstrukturen mit kleinem Laserspot und reduzierter Laserleistung
- 4-fach Aufweitung des Laserstrahls und volle Strahlleistung zur schnellen Belichtung von Füllstrukturen (*Hatch*) innerhalb des Bauteils
- Entwicklung und Aufbau einer geeigneten Beschichtungstechnik und –mechanik

Weiterhin sind Folgetechniken und Abformprozesse zu untersuchen, welche die Einbindung der Stereolithografie als Bestandteil der Prototypenentwicklung in der Mikromechanik ermöglichen. Die herausragenden Eigenschaften der projektierten MSTL-Maschine beschreibt Tabelle 6.3.

Eigenschaften	**Standard-Stereolithografie**	**Mikro-Stereolithografie**
Min. Auflösung (Laserspotdurchmesser)	200 µm	10 µm
Max. Genauigkeit (x-y-Scanner)	40 µm	2 µm
Min. Schichtdicke	200 µm	20 µm
Max. Baufeldgröße	600 x 600 x 600 mm³	100 x 100 x 100 mm³
Komponenten	Proprietär	Standard (COTS[83])
Controller / Bus	Proprietär	Industrie-PC mit PCI-Bus
Programmierung	?	C / C++
Bedienung	Proprietär	GUI von Windows NT

Tabelle 6.3 Herausragende Eigenschaften der projektierten MSTL-Maschine

Aus den Entwicklungszielen des MSTL-Projekts ergibt sich eine Verwandtschaft zum Projekt *Dreidimensionales Drucken von Bauteilen*, da hier die Generierung der Baudaten und die Ansteuerung in sehr ähnlicher Weise behandelt werden. Deshalb sollen, soweit möglich Synergieeffekte genutzt werden, um Ressourcen einzusparen und auf die vorteilhafte Verwendung von flexiblen Standardkomponenten zurückgegriffen werden. Einen Vergleich bezüglich Auflösung, Systempreis und Baugröße der beiden in den Projekten eingesetzten *Rapid Prototyping*-Verfahren zeigt Bild 6.22.

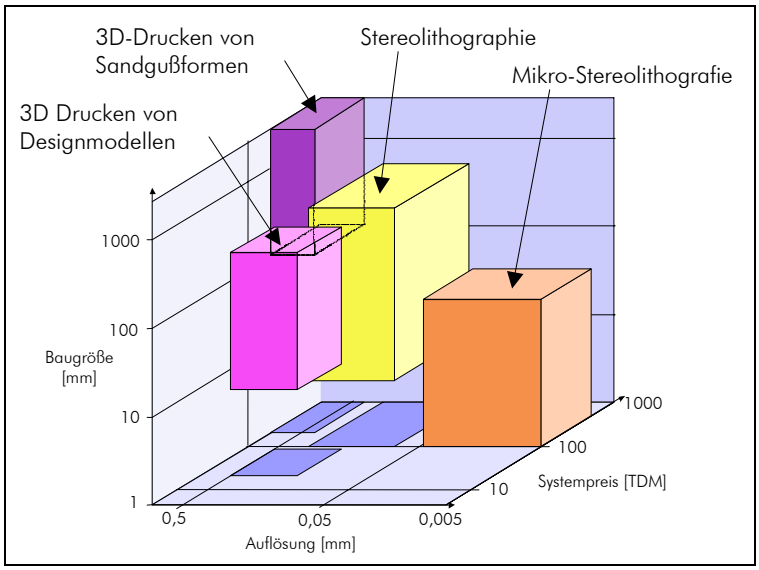


Bild 6.22 Vergleich der *Rapid Prototyping*-Verfahren

[83] COTS: ***C**ommercial **O**ff-**T**he-**S**helf*, Standardkomponenten „von der Stange".

6.2.2 Analyse der zeitkritischen Prozesse und Aufstellen der Realzeit-Anforderungen

Vor dem Aufstellen der Realzeit-Anforderungen muss, wie zuvor beim Projekt 3D-Drucken, der technische Prozess analysiert und in einzelne Prozessphasen zerlegt werden. Diese sind:

1. Erzeugung des Modells in der CAD-Applikation
2. Sicherung der Modelldaten in einem programmspezifischen Dateiformat und Umwandlung in das polygonbasiert STL-Format oder direkte Speicherung im STL-Format
3. Übertragung der STL-Daten in den Steuerrechner der Mikro-Stereolithografiemaschine
4. Generierung der Schichten aus den STL-Daten (*slicing*)
5. Generierung und Separierung der Belichtungspolygone aus den Polygonen der Schichtdarstellung entsprechend der gewählten Belichtungsbreiten. Die beiden Polygontypen sind: Umrandungspolygone (Offene und geschlossene Polygone) und *Hatch*-Polygone (Füllstruktur)
6. Zerlegung der Belichtungspolygone in lineare Mikrovektoren entsprechend der gewählten Sekantennäherung (Schrittweite) und Zuweisung eines Laser-An/Aus-Zustands (Flag)
7. Softwarebasierte Korrektur der Linsen- und Spiegelfehler (z.B. mit Polynominterpolation)
8. Übertragung der Mikrovektoren an eine Scanneransteuerung
9. Vorbereitungen für die erste Schicht (Initialisierungsvorgänge, Plattformjustage, Auftragen des Polymerfilms usw.)
10. Belichtung der Polymerschicht durch Bewegung der x- und y-Scanner entsprechend der Mikrovektorkoordinaten, Aufweitung des Laserstrahls bei *Hatch*-Strukturen und Ein-/Ausschalten des Laserstrahls entsprechend dem Laser-An/Aus-Flag durch den AOM
11. Absenkung der Bauplattform (z-Achse) um eine Schichthöhe (am Ende jeder Schicht) und Auftrag des neuen Polymerfilms
12. Herausfahren des fertigen Bauteils nach Bauprozessende

Der Vergleich zeigt, dass die Prozessphasen 1 – 4, 9, 11 und 12 mit denen des 3-Druckens sehr ähnlich sind. Hier können später die bereits vorhandenen Lösungsansätze direkt oder mit kleinen Anpassungen übernommen werden. Nun muss eine Abschätzung bzw. eine Überprüfung der maximal zulässigen Reaktionszeiten und minimal zu gewährleistenden Datenraten und –mengen für alle Prozessphasen vorgenommen werden:

Zu 1. Die Erzeugung des Modells in der CAD-Applikation erfolgt stets *offline*, d.h. die Konstruktion durch den Anwender erfordert, je nach Komplexität, einen Zeitraum von Stunden bis Tagen.

Zu 2. Die Größe der STL-Datei bewegt sich bei mechanischen Modellen zwischen 0,2 – 2 MB, bei RP-Daten aus Computertomografie-Bildern zwischen 5 – 50 MB.

Zu 3. Die Übertragung erfolgt über wechselbare Medien oder Netzwerkverbindungen. Nach dem Transfer wird meist eine Überprüfung der Datei auf Fehler bei der Umsetzung der Geometriedaten vorgenommen. Dieser Verifikationsprozess erkennt und behebt Konstruktionsfehler, Umsetzungsprobleme und Beschreibungsfehler. Die Zeitdauer ist von der Anzahl der Fehler, Dateigröße und Rechengeschwindigkeit abhängig und beträgt wenige Minuten bis einige Stunden.

Zu 4. Die dreidimensionalen Modelldaten werden entsprechend der gewünschten Schichtstärke in Schichten zerlegt (geschnitten). Die Einzelschichtdaten werden im Vektorformat gespei-

chert und ergeben Dateien in der Größenordnung der zugrundeliegenden 3D-Daten. Die Schichtzerlegung dauert, je nach Rechenleistung, Algorithmus und Schichtenzahl, einige Sekunden bis einige Minuten.

Zu 5. Die Generierung und Separierung der Belichtungspolygone aus den Modellpolygonen erfolgt meist in einem Schritt mit Phase 4. Die Anzahl der Belichtungspolygone hängt von der gewählten Belichtungsstärke und der Größe des Laserspots auf der Polymeroberfläche ab. Als Standard gelten 3 – 4 Umrandungspolygone (30 – 40 μm) sowie 1 – 2 *Hatch*-Polygone (40 – 80 μm) mit aufgeweitetem Laserstrahl.

Zu 6. Bei einer max. Auflösung der Laserscanner von je 16-Bit für x- und y-Spiegel und einem Belichtungsfeld von 125 x 125 mm^2 ergibt sich die geforderte Positioniergenauigkeit des Laserstrahls von ca. 2 μm. Mit einer minimalen Mikrovektorzerlegung von 10 μm (Belichtungsraster), einem vollständig gefüllten Baufeld[84] von max. 100 x 100 mm^2, der Speicherung eines Koordinatenpaars in vier Bytes und der Speicherung des Laser-An/Aus-Flags mit einem Bit/Koordinatenpaar beträgt der resultierende Speicherbedarf für eine Schicht ca. 394 MB. Mit einer minimalen Schichtdicke von 20 μm und einer max. Bauhöhe von 100 mm hätte die Mikrovektordatei eine Gesamtgröße von fast 2 TB. Diese Datenmenge kann mit Standard-PC-Hardware nicht vernünftig verarbeitet werden; die Polygonzerlegung in Mikrovektoren muss also *online*, d.h. schritthaltend zum Belichtungsprozess erfolgen.

Zu 7. Die softwarebasierten Algorithmen zur Korrektur der Linsen- und Spiegelfehler werden auf die generierten Mikrovektoren angewendet. Bei einer hierfür geeigneten Polynominterpolation müssen für jeden zu korrigierenden Belichtungspunkt (x, y) $\rightarrow$ (b_x, b_y) folgende Berechnungen durchgeführt werden: 16 Multiplikationen + 10 Additionen (Spiegelkorrektur) und 10 Multiplikationen + 5 Additionen (Linsenfehler). Moderne x86-Prozessoren schaffen durchschnittlich 0,5 – 1 Fliesskommaberechnungen pro Takt, d.h. der eingesetzte Intel Celeron mit 400 MHz Takt benötigt pro Koordinatenpaar ca. 100 – 200 ns für den Korrekturalgorithmus.

Zu 8. Die Übertragung der Mikrovektoren zur Scanneransteuerung ist aufgrund der Datenmengen kritisch. Bei wenigen Objekten beträgt die Datenmenge der Mikrovektoren nach Zerlegung der berechneten Polygone einer kompletten Schicht typischerweise 5 – 50 MB und passt somit in den Hauptspeicher des Steuerungs-PCs. Anschließend werden die Mikrovektorkoordinaten einzelner Objekte oder Polygongruppen *online* zur Scanneransteuerung übertragen. Die Datenmenge für ein Objekt bzw. Polygongruppe beträgt, je nach Komplexität, ca. 4 – 100 KB.

Zu 9. Sämtliche Initialisierungsvorgänge werden vor Baubeginn abgearbeitet und sind deshalb zeitunkritisch.

Zu 10. Die Auflösung der x- und y-Laserscanner (16-Bit) auf der Polymeroberfläche beträgt ca. 2 μm, die Länge der Mikrovektoren ca. 10 μm. Bei einer vorgegebenen Schichtstärke von 20 μm und einer von Eindringtiefe und Laserleistung abhängigen Belichtungsgeschwindigkeit von 0,1 – 1 m/s muss alle 10 μs ein neues Koordinatenpaar ausgegeben werden. Ein Jitter von ± 2 μs ist hierbei zulässig, da die Trägheit der Scanner eine Integration der diskreten Koordinatenwerte bewirkt. Die erforderliche Datentransferrate von der Ansteuerung zu den Scannern beträgt ca. 39 – 391 KB/s.

Zu 11. Die Absenkung der Bauplattform durch die z-Achse am Ende jeder fertigen Schicht ist unkritisch.

Zu 12. Das Anheben der z-Achse zum Herausnehmen des Bauteils nach Prozessende ist ebenfalls kein zeitkritischer Vorgang.

[84] aufgrund der Verzerrungen im Randbereich muss das Belichtungsfeld größer als das Baufeld sein.

Die Analyse der verschiedenen Prozessphasen ergibt, dass sämtliche realzeitkritischen Vorgänge bei der Ansteuerung der Laserscanner während des Belichtungsprozesses auftreten. Die hierfür erforderlichen Koordinatenpaare müssen im PC *online* berechnet, korrigiert und anschließend an die Scannersteuerung übertragen werden. Die maximale Datenrate von ca. 391 KB ist während der Belichtungsphase jedes zusammenhängenden Objekts zu gewährleisten. Damit nicht nur die Belichtungspausen, d.h. die Sprungphasen zu neuen Koordinaten bei ausgeschaltetem Laserstrahl, zum Nachladen von neuen Daten genutzt werden können, muss eine kontinuierliche arbeitende, unabhängige Ausgabeeinheit (Soft- und/oder Hardware) vorgesehen werden. Sämtliche Prozessphasen werden vom Steuerungs-PC initiiert, sodass keine externe Synchronisation nötig ist. Initialisierungsphasen vor dem Bauprozess und sämtliche Bewegungsphasen nach Schichtende oder Prozessende stellen keine Realzeit-Anforderungen.

Zusammenfassung der zeitlichen Anforderungen:

- Zeitunkritische *offline* Erstellung der CAD-Modelle und Dateien
- Zeitkritische *online* Generierung und Separierung der Belichtungspolygone
- Zeitkritische *online* Zerlegung der Belichtungspolygone in Mikrovektoren
- Zeitkritische *online* Korrektur der Mikrovektoren
- Minimale Datenübertragungsrate zur Belichtungssteuerung: 391 KB/s
- Frequenz der Datenausgabe: 100 kHz (10 μs)
- Maximaler Jitter der Belichtungsdatenausgabe an die Scanner: $\pm$ 2 μs
- Zeitunkritische Antriebssteuerung der z-Achse und des Wischers (*recoater*)

Ein Zustandsdiagramm der wichtigen Prozessphasen sowie weiterer parallel ablaufender zeitunkritischer Prozesse zeigt Bild 6.23.

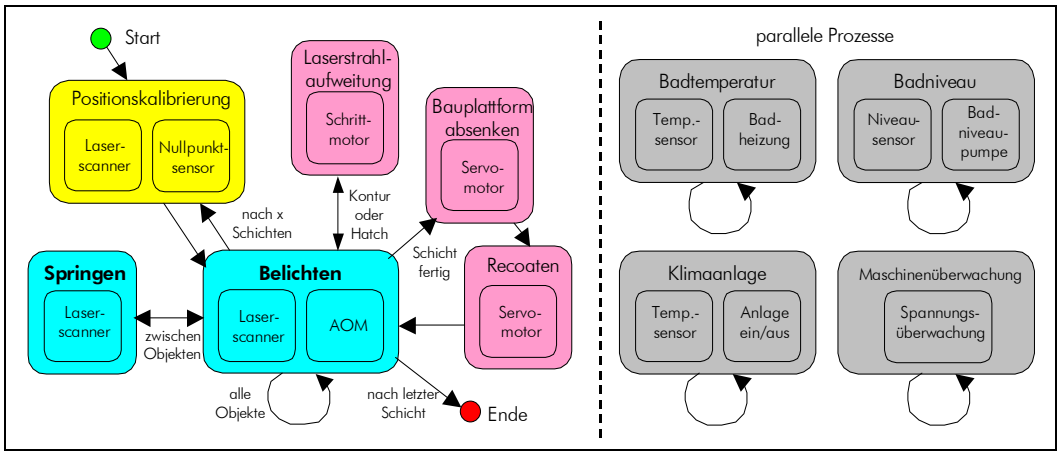


Bild 6.23 Zustandsdiagramm der Prozessphasen (MSTL)

6.2.3 Hardware- und Softwarekonzept

Die ermittelten zeitlichen Anforderungen ermöglichen die Auswahl einer sinnvollen Systemarchitektur mit Hilfe des Klassifizierungsdiagramms von Kapitel 3.3 (Bild 6.24).

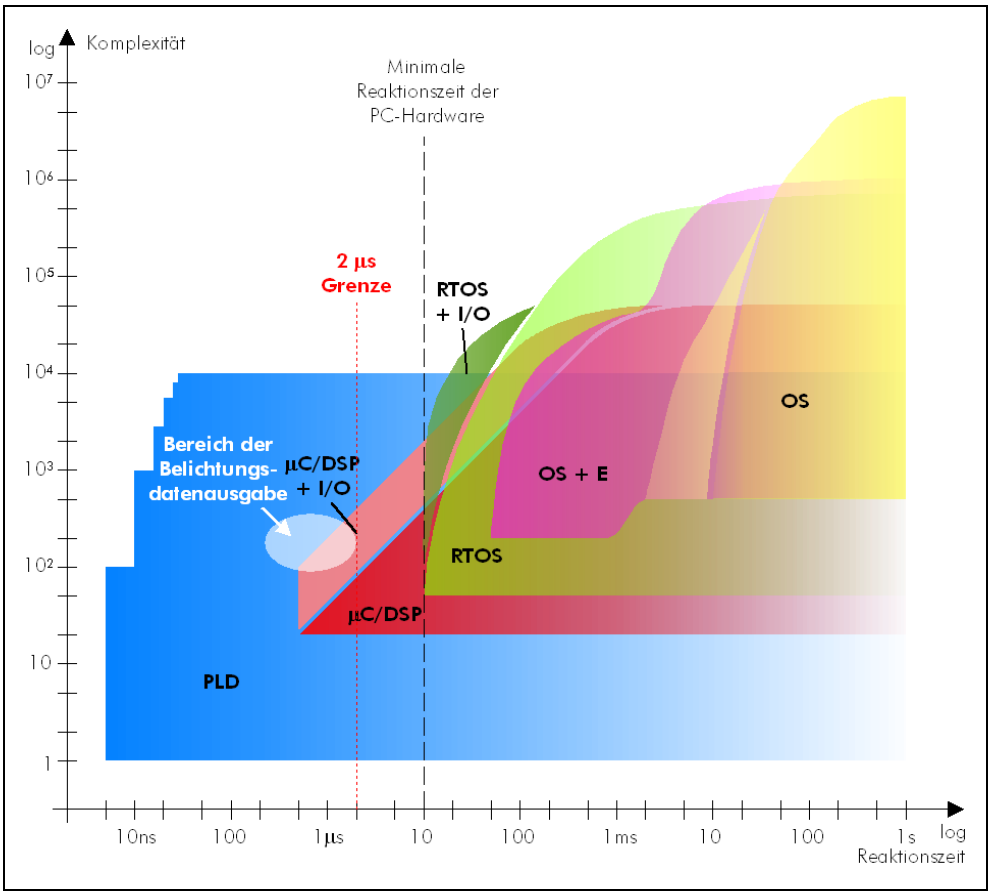


Bild 6.24 Auswahl der passenden Systemarchitektur (MSTL)

Die Ähnlichkeiten mit dem Projekt 3D-Drucken zeigen sich in den analysierten Zeitbedingungen. Hier ergeben sich die zeitkritischen Phasen während der Belichtung des Polymerfilms. Der Blick auf das Klassifizierungsdiagramm schränkt die Auswahl der Systemarchitektur auf die *PLD-Architektur* und die *µC/DSP + I/O-Architektur* ein. Untersuchungen der im Projekt 3DD entwickelten Hardware zeigen, dass diese auch für die Steuerung des Mikro-Stereolithografie-Prozesses geeignet ist. Die modulare Struktur mit Steuerungs-PC, DSP-Karte und Add-On-Modul (*PLD-Architektur*) wird deshalb auch zur Realisierung der Mikro-Stereolithografiemaschine verwendet. Bild 6.25 zeigt die Datenübertragung der Belichtungssteuerung.

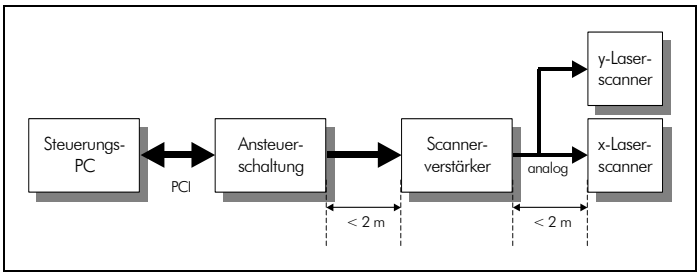


Bild 6.25 Datenübertragung der Belichtungssteuerung vom Steuerungs-PC zum Laserscanner

Der zu den Laserscannern mitgelieferte Scannerverstärker besitzt einen 16-Bit Digitaleingang. Die x- und y-Daten werden verschachtelt (*multiplexing*) und durch wechselweise Aktivierung zweier Strobe-Eingänge übernommen. Nach der Digital-Analog-Wandlung werden die x- und y-Signale

auf einen analogen PID-Regler geführt. Eine Verstärkerstufe erzeugt das Ansteuersignal für die Laserscanner. Das Blockschaltbild des Scannerverstärkers zeigt Bild 6.26.

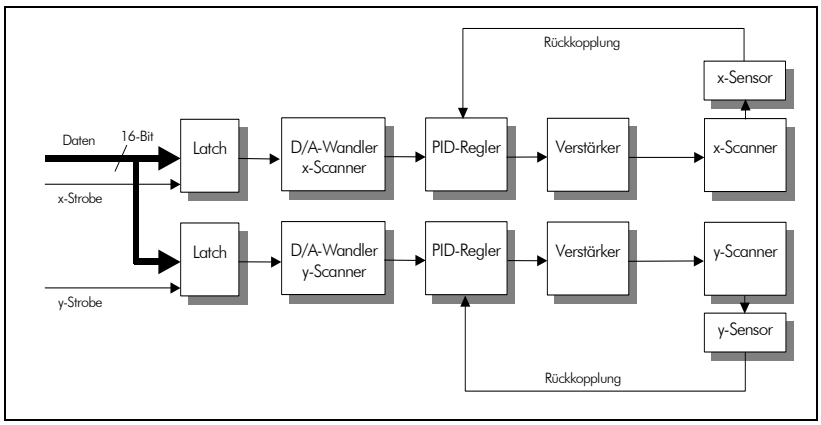


Bild 6.26 Blockschaltbild des Scannerverstärkers

Aufgrund der bereits beschriebenen Systemähnlichkeiten werden DSP-Karte und Add-On-Modul wiederverwendet. Das Add-On-Modul stellt 18 digitale Ausgänge des FIFOs als Datenleitungen und weitere 16 I/O-Ports des CPLDs als Steuerleitungen zur Verfügung. 16 + 1 FIFO-Ausgänge werden mit dem 16-Bit breiten Datenport des Scannerverstärkers und der Steuerleitung des AOMs verbunden. Die schnellen Schaltzeiten des AOMs (20 – 80 ns) stellen sicher, dass das Laser-An/Aus-Signal immer passend zu den Koordinaten anliegt. Die Signale der beiden Strobe-Leitungen werden direkt vom CPLD generiert.

Die DSP-Karte ist für das Nachladen des FIFOs mit neuen Koordinatenwerten zuständig. Nachdem von der DSP-Karte die x- und y-Werte der Mikrovektoren und die Laser-An/Aus-Flags eines Objekts aus dem PC-Speicher geholt wurden (Busmaster-Betrieb), müssen diese paarweise mit dem jeweils zugehörigen Laserflag als 2 x (16 + 1) Bitmuster im FIFO abgespeichert werden. Je nach erreichtem Füllstand (FF, PAE, EF-Signal[85]) löst das CPLD einen Interrupt aus. Der DSP liest aus einem Register die Interrupt-Ursache und reagiert entsprechend. Bei voller Belichtungsgeschwindigkeit wird die errechnete Datentransferrate von knapp 400 KB/s erreicht. Bild 6.27 zeigt Systemdiagramm und Datenfluss.

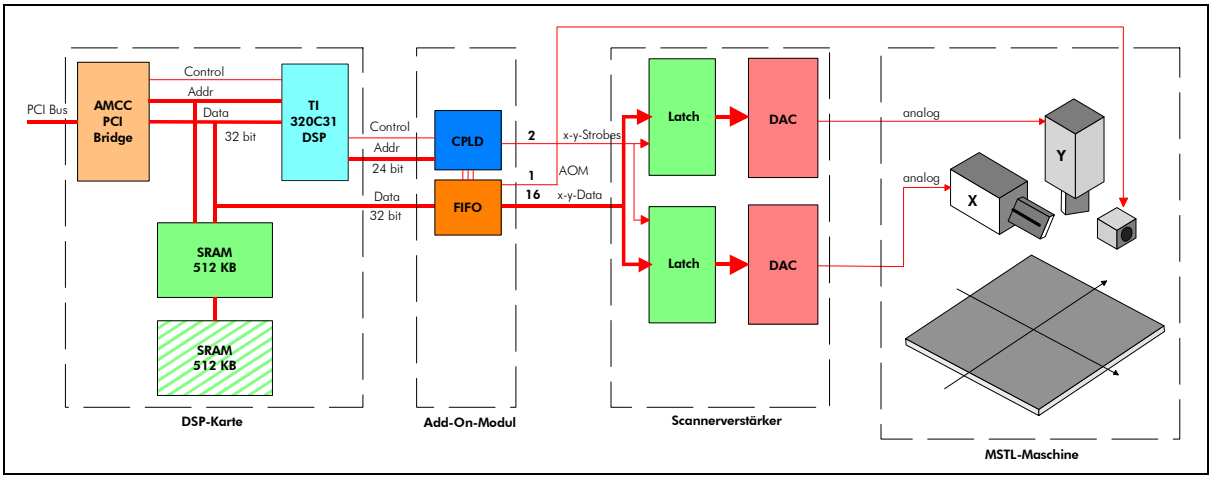


Bild 6.27 Systemdiagramm und Datenfluss der MSTL-Maschine

[85] FF: ***F**ull **F**lag*, PAE: ***P**rogrammable **A**lmost **E**mpty Flag*, EF: ***E**mpty **F**lag*.

Softwarekonzept

Das Softwarekonzept übernimmt die modulare Struktur des Hardwarekonzepts. Entsprechend den verschiedenen Komponenten der MSTL-Maschine wird, unter Berücksichtigung der zeitlichen Rahmenbedingungen des Gesamtprozesses, eine mehrstufige Unterteilung vorgenommen.

3-stufiges Softwarekonzept entsprechend den Realzeitanforderungen:

PC-Software

- CAD-Software
  - Zum Beispiel *Pro/Engineer*, zur Erstellung der Modelle auf einem Windows-PC (dies kann ein separater PC bzw. eine Workstation sein)
- PC-Applikation (auf dem Steuerrechner)
  - Laden und Überprüfen der STL-Dateien
  - Platzieren der Objekte auf dem Baufeld
  - Erzeugen der Schichten
  - Generieren und Separieren der Belichtungspolygone
  - Zerlegung in Mikrovektoren (Koordinatenpaare) und Zuweisung des Laser-An/Aus-Flags
  - Softwarebasierte Korrektur der Mikrovektoren aufgrund optischer Fehler von Spiegel und *f-Theta*-Linse

DSP-Software

- DSP-Software
  - Herunterladen der Mikrovektorkoordinaten aus dem PC-Arbeitsspeicher
  - Umsortierung zu 2 x (16 +1)-Bit Ansteuerdaten
  - Blockweise Übertragung in den FIFO des Add-On-Moduls (Interrupt-Steuerung)
  - Initialisierung und Überwachung des Belichtungsprozesses

PLD-Firmware

- CPLD-Firmware des Add-On-Moduls
  - Steuerung des FIFOs (*full, almost empty, empty*)
  - Generierung der x- und y-Strobe-Impulse zur Steuerung der Datenübernahme im Scannerverstärker (Multiplex-Betrieb)
  - Timergesteuerte Ausgabe der Ansteuerungskoordinaten und des Laser-An/Aus-Flags im 10 μs-Takt

6.2.4 Realisierung

Aufgrund der Ähnlichkeiten mit dem 3D-Drucker-Projekt wurde die Realisierungsstrategie übernommen. Die Aufgaben wurden passend zu den Tätigkeitsfeldern der drei Forschungsgruppen in die Bereiche Mechanik, Ansteuerung und Software aufgeteilt. Zur Mechanik gehörten die Komponenten Antriebe, Antriebssteuerung, Wischer, Plattform, Polymertransport, Laser und Stromversorgung. Die Ansteuerung beinhaltete die Hardware zur Datenübertragung (PC, DSP-Karte, Add-On-

Modul), Scannerverstärker und Laserscanner sowie die zugehörige Soft- und Firmware dieser Komponenten. Der letzte Bereich Software umfasste die Entwicklung der Softwarealgorithmen und der PC-Applikation.

Bild 6.28 zeigt den schematischen Aufbau des Prototypen der Mikro-Stereolithografiemaschine. Man erkennt deutlich die Analogien zur Struktur des 3D-Druckers.

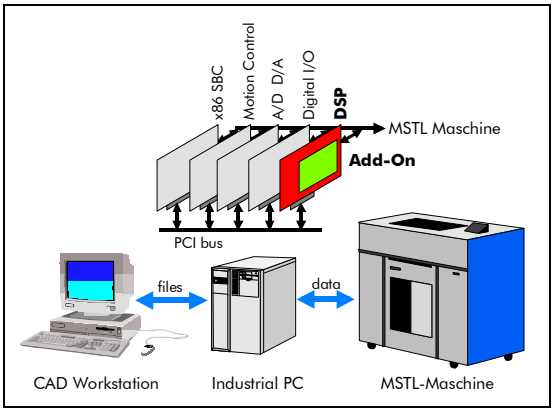


Bild 6.28 Schematischer Aufbau der MSTL-Maschine

Die Speichergrößen der für die Belichtungsansteuerung verwendeten Komponenten, verringerte sich mit den zunehmenden Realzeitanforderungen wie folgt:

PC SDRAM 256 MB → **DSP-Karte SRAM 1 MB** → **Add-On-Modul FIFO 4 KB** → **Scannerverstärker Latch 16 Bit**

Der detaillierte Aufbau von Industrie-PC, DSP-Karte und Add-On-Modul sowie die Programmierung der Software von Industrie-PC, DSP-Karte und Firmware des Add-On-Moduls sind in Kapitel 6.1.4 beschrieben. Die Oberfläche der Steuerungsapplikation unterscheidet sich speziell bei der Darstellung der Ausgabedaten. Wird in der 3D-Drucker-Anwendung die diskretisierte Bitmap in Druckkopfauflösung dargestellt, so zeigt die Softwareoberfläche der Belichtungssteuerung ein zweifarbiges Bild mit den Vektoren der Konturen und Füllstrukturen (Bild 6.29).

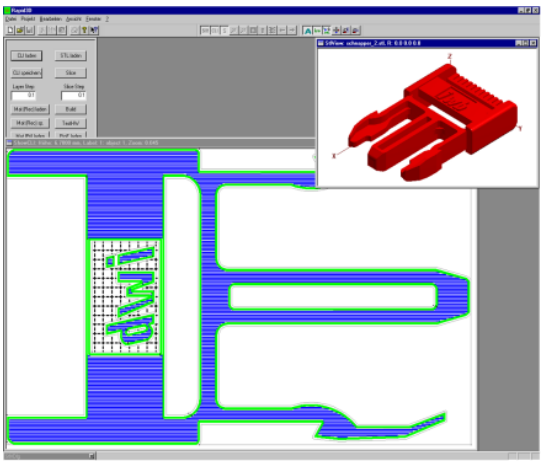


Bild 6.29 *Screenshot* der Steuerungsapplikation (MSTL)

6.2.5 Verifikation und Ergebnisse

Die Einhaltung sämtlicher Realzeitbedingungen und Spezifikationen musste nach Entwicklung und Integration der Hard- und Softwarekomponenten überprüft werden. Auf die Ermittlung des Zyklenbedarfs wichtiger Betriebssystemfunktionen konnte verzichtet werden, da alle Basisfunktionalitäten der DSP-Karte unverändert übernommen wurden. Die Messergebnisse werden in Kapitel 6.1.5 dargestellt. Auch der Speicherbedarf von Realzeit-Kernel und DSP-Anwendung liegt in der Größenordnung des 3DD-Projekts.

Add-On-Modul

Die höchste Hierarchiestufe (bzw. Komplexitätsebene), welche bei der Synthetisierung der Hardwarebeschreibung des CPLDs auf dem Add-On-Modul erreicht wurde, ist 3, d.h. die *worst case*-Latenzzeit beträgt 3 x 7,5 ns = 22,5 ns, bei einer spezifizierten Pin-zu-Pin-Verzögerungszeit des CPLDs von 7,5 ns. Hier erreichte man durch geschickte Priorisierung der zu synthetisierenden Logikblöcke, dass alle reaktionszeitsensitiven Signale nur ein oder zwei Hierarchiestufen benötigen.

Bild 6.30 zeigt die Auslösung der x- und y-Strobe-Impulse (`stbx`, `stby`), während die jeweiligen x- und y- Mikrovektorkoordinaten (`C479`, `E264`) am Datenbus (`d_out`) gültig sind. Die Daten werden mit steigender Flanke der Strobe-Leitung in das Latch des Scannerverstärkers übernommen. Das Signal `tmck0` ist der Ausgangspin `TCLK0` des ersten DSP-Timers. Er löst die Triggerung der CPLD-Ausgabe in einer Interrupt-Service-Routine aus. Die Verzögerung zwischen Timer-Interrupt und Ausgabe des ersten Strobe-Impulses wird größtenteils durch die Interrupt-Latenzzeit verursacht (siehe hierzu Kapitel 6.1.5). Im Bild werden Daten- und Strobe-Signale mit einem minimalen Intervall von ca. 5,3 μs (188 kHz) erzeugt. Die maximal erreichbare Frequenz liegt somit über der spezifizierten Frequenz von 100 kHz; damit ist sichergestellt, dass das Zeitintervall von 10 μs erreicht wird. Die maximal erreichbare Datentransferrate errechnet sich zu 737 KB/s (gefordert 391 KB/s).

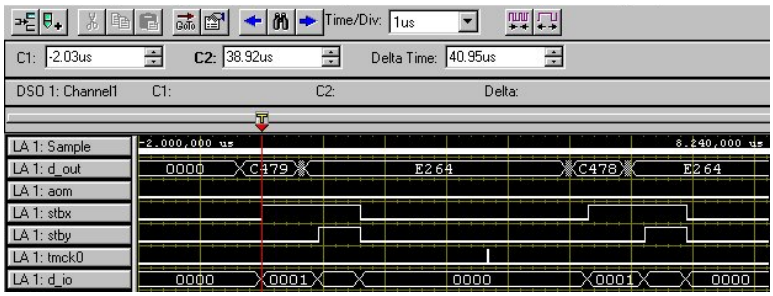


Bild 6.30 Timing des Datenbusses und der Strobe-Signale (ISR)

Bei der Ansteuerung der Laserscanner muss nicht auf externe Ereignisse reagiert werden. Die zyklische Datenausgabe und die Erzeugung der Strobe-Impulse wird über das Triggersignal eines DSP-Timers gestartet. Es kann entweder eine Interrupt-Service-Routine im DSP aufgerufen werden oder das Signal triggert direkt die Datenausgabe im CPLD (FIFO-Ausgabe) über einen separaten Pin. Die erste Methode erlaubt eine einfache Anpassung der Verzögerungszeiten an die verhältnismäßig langsame Scannerverstärker-Hardware durch Einfügen von Wartezyklen in der ISR (`nop`-Befehle). Ist ein Scannerverstärker mit schnelleren Eingangsregistern (*latches*) verfügbar, sollte die hardwarebasierte Datenübertragung und Strobe-Impuls-Auslösung des CPLDs verwendet werden (Bild 6.31). Hiermit sind wesentlich höhere Datentransferraten zum Scannerverstärker möglich. Die Zeit vom

Auftreten des Timer-Impulses bis zur steigenden Flanke des zweiten Strobe-Signals (y-Strobe) beträgt dann nur noch ca. 280 ns. Damit wird eine maximale (theoretische) Übertragungsrate von über 5 MB/s erreicht.

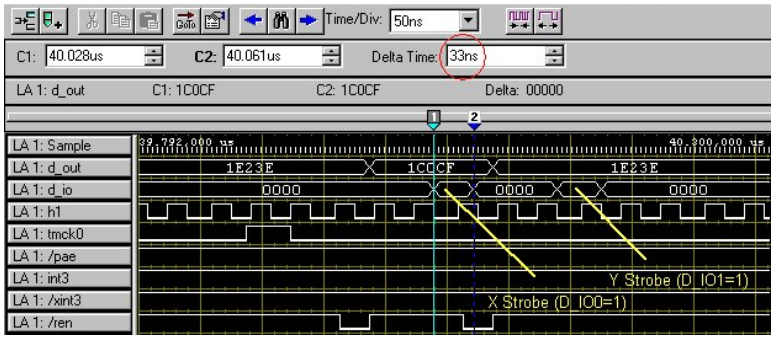


Bild 6.31 Timing des Datenbusses und der Strobe-Signale (CPLD)

Zusammenfassung

Alle zeitlichen Anforderungen werden erfüllt, minimale Datenübertragungsraten eingehalten. Die Verzögerungszeit zwischen Timerimpuls und Datenausgabe bzw. Strobe-Impulsgenerierung beträgt ca. 3,4 µs. Eine Verzögerungszeit (Jitter) von ca. 3 µs ist bei Auftreten des `INT2`-Interrupts hinzuzuaddieren. Dieser wird vom PCI-Controller ausgelöst und hat eine nicht veränderbare, höhere Priorisierung als der `TINT0`-Interrupt des Timers. Aufgrund der geringen Auftrittswahrscheinlichkeit des Interrupts und der Trägheit der Laserscanner sind jedoch zufriedenstellende Belichtungsergebnisse zu erzielen. Ist eine perfekte, jitterfreie Belichtungsansteuerung erforderlich, muss die CPLD-Triggerung verwendet werden. Für ISR-Triggerung und CPLD-Triggerung lassen sich somit zwei unterschiedliche Verzögerungszeiten angeben:

ISR-Triggerung (softwarebasiert): ca. 3,4 µs (– 6,4 µs)

- Interrupt-Latenzzeit vom Auftreten des Timer-Interrupts bis zum Eintritt in die Timer-ISR (ca. 1,33 µs)
- Evtl. Jitter durch Auftreten des höherprioren INT2-Interrupts (ca. 3 µs)
- Schreibzugriff auf den CPLD für Triggerung des FIFOs zur Datenausgabe (< 70 ns)
- Programmierte Wartezeit zur Stabilisierung der Datenleitungen (ca. 250 ns)
- Generieren des x-Strobe-Signals (< 70 ns)
- Programmierte Wartezeit zur Übernahme der x-Mikrovektorkoordinaten durch den Laserscanner (ca. 600 ns)
- Schreibzugriff auf den CPLD für Triggerung des FIFOs zur Datenausgabe (< 70 ns)
- Programmierte Wartezeit zur Stabilisierung der Datenleitungen (ca. 250 ns)
- Generieren des y-Strobe-Signals (< 70 ns)
- Programmierte Wartezeit zur Übernahme der y-Mikrovektorkoordinaten durch den Laserscanner (ca. 600 ns)
- Rücksetzen der Strobe-Signale (< 70 ns)

CPLD-Triggerung (hardwarebasiert): ca. 280 ns

- Verarbeitungszeit im CPLD vom Auftreten des Timer-Interrupts bis zur Triggerung des FIFOs zur Datenausgabe (< 100 ns)
- Programmierte Wartezeit im CPLD zur Stabilisierung der Datenleitungen (ca. 50 ns)
- Generieren des x-Strobe-Impulses und Übernahme der x-Mikrovektorkoordinaten durch den Laserscanner (Dauer ca. 33 ns)
- Verarbeitungszeit im CPLD bis zur Triggerung des FIFOs zur Datenausgabe (ca. 15 ns = 2 Komplexitätsebenen)
- Programmierte Wartezeit im CPLD zur Stabilisierung der Datenleitungen (ca. 50 ns)
- Generieren des y-Strobe-Impulses und Übernahme der y-Mikrovektorkoordinaten durch den Laserscanner (Dauer ca. 33 ns)

Die erzielten Reaktionszeiten ermöglichen Datenausgabeintervalle unterhalb der erforderlichen 10 μs. Wird die ISR-Triggerung eingesetzt, so muss, aufgrund der ungünstigen Priorisierung der Interrupts, im *worst case* mit einem Jitter von ca. 3 μs gerechnet werden. Durch Einsatz der hardwarebasierten CPLD-Triggerung lassen sich wesentlich kürzere Reaktionszeiten mit vernachlässigbarem Jitter erreichen. Die Datenschnittstelle des angeschlossenen Scannerverstärkers muss aber in diesem Fall für die höheren Übertragungsraten ausgelegt sein. Dieses Implementierungsbeispiel zeigt die im Kapitel 5.1.1 beschriebene, sinnvolle untere Realisierungsgrenze der μC/DSP-Architektur im Bereich von einigen Mikrosekunden.

Prototyp und Musterstücke

Nach der Entwicklung aller Komponenten wurden diese in einem Mikro-Stereolithografiemaschinen-Prototyp integriert. Bild 6.32 zeigt den Aufbau der Belichtungskammer mit dahinterliegender Leistungselektronik auf einer luftgelagerten, schwingungsarmen Platte und den separaten Steuerungsrechner auf der rechten Seite. Die durchsichtigen Türen aus Spezialkunststoff blockieren das UV-Licht während des Bauprozesses; sowohl einstrahlendes Streulicht als auch Reflexionen des UV-Laserstrahls werden gefiltert. Die massive Stahlkonstruktion mit den leistungsstarken Achsantrieben für Bauplattform (z-Achse) und Wischer (x-Richtung) erlauben den diffizilen 20 μm-Schichtauftrag.

Bild 6.32 Prototyp der Mikro-Stereolithografiemaschine

Mit Hilfe dieser Maschine wurden nach Anpassungen und Optimierungen des Belichtungsprozesses die ersten Musterstücke hergestellt. Das folgende Bild 6.33 zeigt verschiedene Modelle. Links sieht man den miniaturisierten Ansaugkrümmer eines PKW-Motors. Zum Größenvergleich dient das Streichholz. Das rechte Bild zeigt eine Einbettungsstruktur, die verhindert, dass kleine Bauteile durch die vorhandenen Strömungen im flüssigen Polymerbad wegdriften. Die Gitterstruktur ist nur durch wenige, feine Kontaktpunkte mit dem Musterstück verbunden und kann nach Prozessende leicht aufgetrennt werden; das Bauteil fällt heraus. Man erkennt auch die prozessbedingte Schichtstruktur.

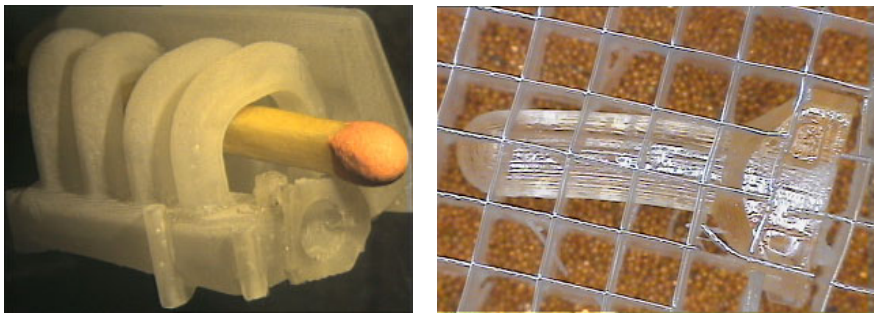


Bild 6.33 Ansaugkrümmer eines PKW-Motors und Struktur zum Einbetten kleiner Teile

Weitere Musterstücke zeigt das nächste Bild 6.34. Das linke Bild stellt einen Faserkoppler zur Verbindung von Glasfaserkabeln dar, im mittleren Bild sieht man ein Turbinenrad und rechts einen Schachturm mit innenliegender Wendeltreppe und einer Treppenstufenhöhe von ca. 100 μm.

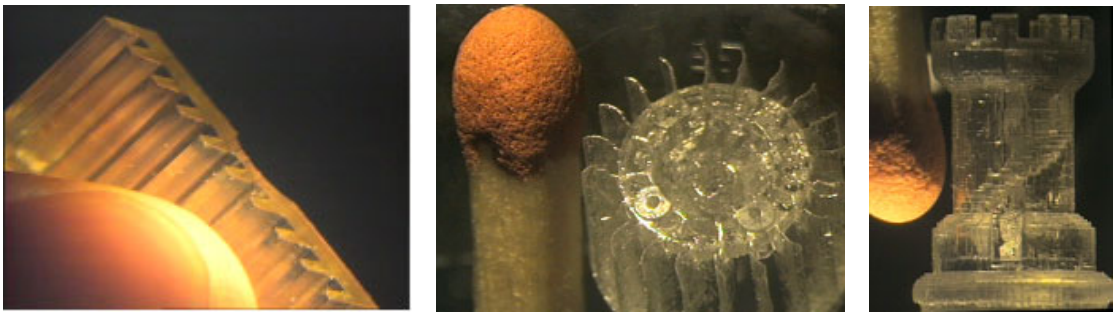


Bild 6.34 Musterstücke: Faserkoppler, Turbine und Schachturm

Weitere Verwendung

Aktuell wird ein zweiter Prototyp der Mikro-Stereolithografiemaschine für weitere Prozessuntersuchungen aufgebaut. Der 3D-Drucker für den Sanddruckprozess hat das Prototypenstadium inzwischen verlassen. Erste marktreife Produkte werden von einem neugegründeten Unternehmen hergestellt und kommerziell vertrieben. Ein serientauglicher 3D-Drucker für den Wachsdruckprozess befindet sich in der Entwicklung.

7 Ausblick

Die PC-Technik unterliegt einer rasanten Weiterentwicklung. Durch die stetig fortschreitende Leistungssteigerung und Kostenreduzierung kann in immer mehr Anwendungsgebieten auf proprietäre Systeme verzichtet werden. Die verbreiteten *De-facto*-Standards der PC-Welt und die vielfältigen Kommunikationsmethoden kommen dem Wunsch des Anwenders nach offenen verteilten Systemen entgegen. Durch Erweiterungen der Soft- und Hardware dringen PC-basierte Steuerungssysteme in Bereiche vor, in denen die Einhaltung von harter Realzeit gefordert wird. Die Verwendung von Komponenten „von der Stange" (COTS) hilft Entwicklungs- und Produktionskosten einzusparen. Die Auswahl geeigneter PC-basierter Systemarchitekturen zur Realisierung dieser Steuerungssysteme wird durch die in dieser Arbeit entwickelte *Klassifizierung von Systemarchitekturen* erheblich vereinfacht. In zwei Anwendungsbeispielen wurde diese Klassifizierung erfolgreich verifiziert. Ein weiterer zukünftiger Verwendungsbereich ist die Nutzung als Basis für den Prozess des Hardware-Software Co-Designs. Die gewonnenen Erkenntnisse können als Parameter für den automatisierten Systementwurf dienen. Bereits direkt nach der Analyse der Realzeitanforderungen kann eine passende Systemarchitektur ausgewählt und als möglicher Implementierungsvorschlag genutzt werden. Trifft man im Auswahlprozess aufgrund der Analyse der Realzeitbedingungen und Komplexitätsanforderungen die Grenzbereiche der Architekturklassen, so sollte für die Wahl der endgültigen Architektur die zukünftige Entwicklung der zugrundeliegenden Technologie mit in die Entscheidung einbezogen werden. Hier sind auch Ansätze für eine weitergehende Forschung gegeben. Auch die einzelnen Systemarchitekturen lassen sich eventuell in weitere Untergruppen gliedern. So könnten z.B. Mehrprozessorsysteme mit gemischten Betriebssystemen (Standardbetriebssystem + Realzeitbetriebssystem) geringe Reaktionszeiten bieten, ohne auf die Vorteile eines Standardbetriebssystems verzichten zu müssen.

Allerdings ist ein PC-basiertes Steuerungssystem nur so leistungsfähig wie seine Komponenten. Hier gibt es vielversprechende Ansätze und Vorschläge, um die Realzeitfähigkeit der Soft- und Hardware zu verbessern. Für die softwarebasierten Systemarchitekturen ist die Eignung des neuen Microsoft Betriebssystems Windows CE 3.0 ein interessantes Forschungsgebiet, da es die Vorteile der bekannten Windows-Oberfläche mit einem harten Realzeitbetriebssystem-Kern zu verbinden versucht. Auch die Realzeiterweiterungen des Linux Betriebssystems, wie RT-Linux oder RTAI, sind hinsichtlich ihres Realzeitverhaltens näher zu analysieren. Bei Verwendung von standardisierte Software ist aber zu beachten, dass Entwicklungen wie automatische Konfigurationsmechanismen (*Plug & Play*), Stromsparmodi oder Systemmanagementfunktionen einen großen negativen Einfluss auf das Realzeitverhalten des Gesamtsystems haben können. Hier fehlen vor allem detaillierte Informationen der Hersteller und entsprechende Analysen des Zeitverhaltens. Bevor man Änderungen an der Software durchführt, ist es sinnvoll, alle vorhandenen Möglichkeiten zur Verbesserung des Realzeitverhaltens auszuschöpfen. Hilfreich sind z.B. der Verzicht auf Auslagerungsdateien, Verwendung von *Page-locking*-Verfahren oder die Nutzung systemkonformer Treiber, um nur einige zu nennen. Gerade die Problematik mit unsachgemäß programmierten Gerätetreibern und den damit verbundenen Instabilitäten und Zeitverzögerungen hat Microsoft in jüngster Zeit dazu bewegt, die Zertifizierung der Gerätetreiber von Drittherstellern zu forcieren. Die Verwendung ausgereifter Software und Gerätetreiber, welche zwar nicht die allerneuesten „Gimmicks" bieten, dafür aber nicht die

Systemstabilität gefährden, ist oftmals die bessere Wahl. Zur Reduzierung der Verzögerungszeiten, gerade im Hinblick auf die immer stärkere Vernetzung von Systemen, kann man auch eine verbesserte Treiberarchitektur, z.B. die *Virtual Interface*-Architektur einsetzten.

Ähnlich der Software bietet auch die PC-Hardware zahlreiche Möglichkeiten zu Verbesserungen. Die historisch gewachsene PC-Architektur enthält, aus übertriebenem Streben nach Kompatibilität, viele antiquierte Techniken. Durch gezieltes Weglassen überflüssiger Komponenten oder Schnittstellen, z.B. des ISA-Busses, können die Reaktionszeiten und deren Schwankungen (Jitter) in einfacher Weise verringert werden. Auch hier können, wie bereits bei den Softwareverbesserungen beschrieben, durch Einsatz bewährter anstatt allerneuester Chipsätze Probleme mit fehlerhaften Implementierungen (*B-stepping*) vermieden werden. Ein anderes Bild ergibt sich bei der Nutzung von Chipsätzen mit neuen Architekturmerkmalen. Leistungsfähigere Schnittstellen und Busse, z.B. durch Verwendung eines neuen Systembusses auf *Hub-Link*-Basis, entschärfen entscheidend die Verzögerungsproblematik, insbesondere bei stark belasteten Systemen. Generell kommt es auf die sorgfältige Auswahl und Zusammenstellung aller beteiligten Komponenten an. Zu Beachten ist auch, das die Verwendung von geringfügig teurerer Hardware bekannter Hersteller zur Systemstabilität beiträgt. So ist z.B. nur auf wenigen Einzelprozessorplatinen großer Hersteller ein I/O-APIC-Baustein zur verzögerungszeitreduzierten Weiterleitung der Interrupt-Anforderungen integriert. Auch spezielle *Embedded*-Versionen des BIOS mit verbessertem Reaktionszeitverhalten sind nicht grundsätzlich üblich. Höhere Leistung mit zugleich verbessertem Realzeitverhalten könnte durch Systemarchitekturen und Chipsätze ermöglicht werden, die auf der neuen *Switched-fabric*-Technologie basieren. Die Kompatibilität zur existierenden PC-Architektur soll hierbei gewahrt bleiben. Wie in der Arbeit aufgezeigt wird, gibt es Systemarchitekturen, bei denen eine Verringerung der Reaktionszeiten nur durch eine gezielte Erweiterung des PC-Systems mittels zusätzlicher Hardwarekomponenten erreichbar ist. Hier kann eine Aufteilung der rechenintensiven zeitunkritischen Algorithmen auf den PC-Prozessor und der zeitkritischen Antwortfunktionen auf den Mikrocontroller oder DSP erfolgen. In diesem Bereich gibt es Bedarf für weitere Forschung, gerade im Bezug auf die optimierte Aufteilung dieser Aufgaben. Die automatisierte Abbildung und Codegenerierung wäre im Hinblick auf ein vollständig rechnerbasiertes Hardware-Software Co-Design zu untersuchen.

Der steigende Einsatz von Steuerungssystemen, insbesondere der PC-basierten Systeme, beschäftigt auch verstärkt die Normungsgremien wie DIN, ISO oder IEC. Die von ihnen geschaffenen Standards spezifizieren Programmierschnittstellen und Programmiersprachen (IEC 61131), Funktionsblocksysteme (IEC 61499, 61804), Datenmodelle für die Realzeit-Kommunikation in verteilten Systemen (IEC 61850), Nachrichtentelegramme (ISO/IEC 9506) und sogar offene Entwurfsmodelle zur Integration von Applikationen (ISO 15745) unter Verwendung der Internet-Metasprache XML (*eXtensible Markup Language*) für Automatisierungs- und Steuerungssysteme. Leider „hinken" die Normungsgremien dem schnellen Fortschritt der PC-Technik immer öfters hinterher. *De-facto*-Standards werden vermehrt von großen Firmen wie Intel, IBM oder Microsoft geschaffen, von anderen Anbietern übernommen und in Produkte integriert, bevor es zu einer Normung kommt. Umgekehrt dringen Techniken der PC-Welt, wie *ActiveX*, DCOM oder OPC in die Welt der Automatisierungstechnik ein. Inzwischen werden auch objektorientierte Entwurfsmethoden wie *UML-RealTime* für die Kapselung von Funktionsblöcken eingesetzt, um Schnittstellen von Automatisierungssystemen sehr früh in der Spezifikationsphase beschreiben zu können, ohne bereits detailliertes Wissen über die Kommunikationshardware besitzen zu müssen. Der Bedarf an Spezialisten mit entsprechendem Fachwissen wird in Zukunft durch den Einsatz von objektorientierten Entwurfsverfahren in der Steuerungssystem-Entwicklung und -Konfiguration stark ansteigen.

Abkürzungsverzeichnis

μC	Mikrocontroller
3DD	3-dimensionaler Drucker
3Dnow	Erweiterung der x86-Prozessorarchitektur von AMD
ABEL	Advanced Boolean Equation Language
ABS	Acrylonitrile-Butadiene-Styrene, Kunststoff
AC97	Audio Codec 97, Standard für Anschluss von seriellen Wandlerbausteinen
ADPCM	Adaptive Differential Pulse-Code Modulation
ADSL	Asymmetric Digital Subscriber Line
AFE	Analogue Front End
AGP	Accelerated Graphics Port
AOM	Acusto-optical Modulator
API	Application Program(ming) Interface
APIC	Advanced Programmable Interrupt Controller
ASIC	Application-Specific Integrated Circuit
BEDO	Burst Extended Data Out (RAM)
BFS	Bayerische Forschungsstiftung
BIOS	Basic Input Output System
BTB	Branch Target Buffer
CAD	Computer-Aided Design
CAS	Column Address Select
CBR	CAS-before-RAS
CMOS	Complementary Metal Oxide Semiconductor
CNC	Computer Numeric(al) Control
COTS	Commercial Off-The-Shelf
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CRC	Cyclic Redundancy Code
CS	Code Segment
DCOM	Distributed Component Object Model (Microsoft)
DCS	Digital Control System
DCT	Discrete Cosine Transformation
DDR	Double Data Rate
DIMM	Dual In-line Memory Module
DIN	Deutsches Institut für Normung
DMA	Direct Memory Access
DOS	Disk Operating System
DPC	Deferred Procedure Call
DSP	Digital Signal Processor
EDF	Earliest Deadline First Scheduling
EDO	Extended Data Out (RAM)
ERP	Enterprise Resource Planning

FIFO	First In First Out, Pufferbaustein
FPGA	Field-programmable Gate Array
FPM	Fast Page Mode (RAM)
FSB	Front Side Bus
FSK	Frequency Shift Keying
GPIO	General Purpose Input/Output
H110	Computer Telephony Bus System
HAL	Hardware Abstraction Layer
HaM	Host accelerated Modems
HIL	Hardware-In-The-Loop
HSP	Host Signal Processing
HTTP	Hyper Text Transfer Protocol (WWW protocol)
I/O	Input/Output
ICC	Interrupt Controller Communications Bus
IDCT	Inverse Discrete Cosine Transformation
IDE	Integrated Drive Electronics (hard disk)
IEC	International Electrotechnical Commission
IEEE1394	Serieller Bus, genormt vom Institute of Electrical & Electronics Engineers
IRQL	Interrupt Request Level
ISA	Industry Standard Architecture (PC slot type)
ISO	steht für: International Organization for Standardization
ISR	Interrupt Service Routine
IST	Interrupt Service Thread
JVM	Java Virtual Machine
LCD	Liquid Crystal Display
LDT	Lightning Data Transfer, frühere Bezeichung für HyperTransport von AMD
LVDS	Low Voltage Differential Signaling
MC	Motion Compensation
ME	Motion Estimation
MES	Manufacturing Execution Systems
MIPS	Million Instructions Per Second
MMU	Memory Management Unit
MMX	Multi-Media Extensions, Erweiterung der x86-Prozessorarchitektur von Intel
MP3	Audiodatenformat der Moving Picture Experts Group
MPEG2	Videodatenformat der Moving Picture Experts Group
MRL	Memory Read Line
MRM	Memory Read Multiple
MSTL	Mikro-Stereolithografie
MTRR	Memory Type Range Register
MXT	Memory eXtension
NMI	Non-Maskable Interrupt
NRE	Non-Recurring Engineering (Costs)
NTC	Negative Temperature Coefficient
OCX	OLE Custom Control
ODVA	Open DeviceNet Vendor Association
OLE	Object Linking and Embedding
OMAC	Open Modular Architecture Controller
OPC	OLE for Process Control

OS	Operating System
PC	Personal Computer
PCI	Peripheral Component Interconnect (personal computer bus)
PIC	Programmable Interrupt Controller
PID	Proportional-Integral-Differential-Regler
PIM	Process Information Management
PIO	Programmed Input/Output
PLC	Programmable Logic Control(ler)
PLD	Programmable Logic Device
PPS	Production Planning System
PS2	Personal System 2 (IBM PC and a connector format)
PWM	Pulsweiten-Modulation
RAID	Redundant Array of Inexpensive Disk
RAM	Random Access Memory
RAS	Row Address Select
RCS	Lehrstuhl für Realzeit-Computersysteme
RDRAM	RAMBUS Dynamic Random Access Memory
RISC	Reduced Instruction Set Computer
RMS	Rate Monotonic Scheduling
ROM	Read-Only Memory
RPI	Rockwell Protocol Interface (Modem)
RTOS	Real-time Operating System
RTSS	Real-time Subsystem
SAP	Systeme, Anwendungen, Produkte in der Datenverarbeitung, SAP AG
SBC	Single Board Computer
SCADA	Supervisory Control and Data Acquisition
SCM	Supply Chain Management
SCSI	Small Computer System Interface
SDRAM	Synchronous Dynamic Random Access Memory
SERCOS	Serial Real-time Communication System (IEC 61491)
SMB	System Management Bus
SMI	System Management Interrupt
SoC	System On a Chip
SPEC	Standard Performance Evaluation Corp
SPI	Serial Peripheral Interface
SPS	Speicherprogrammierbare Steuerung
SSE	Streaming SIMD Extensions, SIMD: Single Instruction-Stream, Multiple Data-Stream
STL	Stereolithography oder Standard Triangle Language, Dateiformat für CAD-Daten
TCP/IP	Transmission Control Protocol/Internet Protocol
TLB	Translation Look-aside Buffer
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
UNC	Universal Naming Convention
USB	Universal Serial Bus
VHDL	Vhsic Hardware Description Language, Vhsic: Very High-Speed Integrated Circuit
VLE	Variable Length Encoding
VME	Versa Module Eurocard (IEEE 1014)

WCET Worst Case Execution Time

Literaturverzeichnis

[AHM+00] S. Arramreddy, D. Har, K. Mak et al. *IBM Memory eXpansion Technology (MXT) Debuts in a ServerWorks North Bridge*. Paper, ServerWorks Inc., IBM, USA, 2000.

[Ali00] F. Alicke. *Serie Buslösungen: Einfädeln auf der Leitung, Parallele Daten zu serialisieren spart Platz und Leiterpaare*. Artikel, ElektronikPraxis, Heft 10/00, S.48-52, Vogel Verlag, Mai 2000.

[Ano99] Anonymous. *Techniques for Increasing PCI Performance*. Application Note, AP-666, Order Number: 292224-001, Intel Corporation, USA, Feb. 1999.

[Api00] API NetWorks. *The Lightning Data Transport I/O Bus Architecture*. White Paper, API NetWorks Inc., USA, 2000.

[BAH+00] M. Banikazemi, B. Abali, L. Herger and D. Panda. *Design Alternatives for Virtual Interface Architecture and an Implementation on IBM Netfinity Cluster*. Paper, Dept. of Computer and Information Science, The Ohio State University, USA, 2000.

[Bak97] A. Baker. *The Windows NT Device Driver Book*. Prentice Hall, 1997.

[Bau95] R. Baumgartl. *Integration von Signalprozessor-Hardware in ein Echtzeit-Betriebssystem*. Diplomarbeit, Lehrstuhl Betriebssysteme, Technische Universität Dresden, 1995.

[BD97] D. Bhandarkar and J. Ding. *Perfomance Characterization of the Pentium Pro Processor*. Paper, Intel Corporation, USA, 1997.

[Bec00] A. Becker. *Tomorrow's IO interfaces: Switching to switched?* Artikel, Integrated Communications Design, USA, Aug. 2000.

[Bec99] Beckhoff. *TwinCat System Übersicht*. Produktbeschreibung, Beckhoff Industrie Elektronik,1999.

[Ber97] G. Bergsma. *Real-time Extensions to Windows NT – Are they right for your next real-time project?* Artikel, Real-Time Magazine, 2/97, 1997.

[Bia01] B. Bialek. *Virtual Interface Architecture and Microsoft SQL Server*. White Paper, Dell, USA, 2001.

[BN00] S. Brandt and G. Nutt. *Flexible Soft Real-Time Processing in Middleware*. Paper, Computer Science Department, University of California, Santa Cruz, USA, 2000.

[BSH98] R. Baumgartl, K. Stuhlemmer und H. Härtig. *Data Dependency in MPEG Audio Frame Decoding Time on a DSP Accelerator*. Paper, In Proceedings of ICSPAT'98, Toronto, Canada, Sep. 1998.

[CGW+00] L. Chung, J. Gray, B. Worthington and R. Horst. *Windows 2000 Disk IO Performance*. Technical Report, Microsoft Research, Microsoft Corporation, Redmond, USA, 2000.

[Com00] Compaq. *InfiniBand Architectural Technology*. Technology Brief, Compaq Computer Corporation, USA, July 2000.

[Dro95] S. Dropsho. *Real-time Penalties in RISC Processing*. Paper, Amherst Department of Computer Science, University of Massachusetts, USA, 1995.

[DSW92] P. Donohoe, R. Shapiro and N. Weiderman. *Hartstone Benchmark Users Guide, Version 1.0*. Paper, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1990.

[ERJ+00] M. Evans, W. Red, G. Jensen et al. *Open Architecture for Servo Control Interface using a Digital Control Interface*. Paper, Mechanical Engineering Department, Brigham Young University, Provo, Utah, USA, 2000.

[Fär94] G. Färber. *Prozessrechentechnik*. 3. Auflage, Springer Verlag, 1994.

[Fis98] P. Fischer. *A Comparison of Approaches*. Artikel, Real-Time Magazine, 3/98, 1998.

[FMO+98] M. Fiuczynski, R. Martin, T. Owa and B. Bershad. *On Using Intelligent Network Interface Cards to support Multimedia Applications*. Paper, In Proceedings of NOSSDAV '98, New Hall, Cambridge, UK, 1998.

[Fre00] J. Freitag. *Echtzeitfähige Java Virtual Machine*. Artikel, SPS-Magazin, Heft 1+2/00, TeDo-Verlag, 2000.

[Fre99] J. Freitag. *Java und Echtzeit*. White Paper, Siemens, 1999.

[FSL96] P. Fiorini, H. Seraji and M. Long. *A PC-Based Configuration Controller for Dexterous 7-DOF Arms*. Paper, JPL, California Institute of Technology, Pasadena, USA, 1996.

[FTY98] N. Frampton, J. Tsao and J. Yen. *Hard Real-time Extensions of Windows NT Evaluation Report*. White Paper, General Motors Powertrain Group, USA, 1998.

[Gig00] Giganet. *cLAN High Performance Host Adapters*. Data Sheet, Giganet Inc., Concord, Massachusetts, USA, 2000.

[GKL+99] C. Gill, F. Kuhns, D. Levine et al. *Applying Adaptive Real-time Middleware to Address Grand Challenges of COTS-based Mission-Critical Real-Time Systems*. Paper, In Proceedings of the 1st International Workshop on Real-Time Mission-Critical Systems, IEEE, Phoenix, Arizona, USA, Nov. 1999.

[GN99] M. Gergeleit and E. Nett. *JewelNT: Monitoring of Distributed Real-Time Application on Windows NT*. Paper, In Proceedings of the 3rd Annual IASTED International Conference on Software Engineering and Applications (SEA'99), pp. 325-328, Scottsdale, Arizona, USA, Oct. 1999.

[Gut99] J. Gutleber. *Challenges in Data Acquisition at the Beginning of the New Millenium*. Paper, In Proceedings of the 1st International Workshop on Real-Time Mission-Critical Systems, IEEE, Phoenix, Arizona, USA, Nov. 1999.

[HB00] R. Huber und G. Burmberger. *Regelung eines schwebenden Körpers unter Windows NT*. Artikel, Elektronik, Heft 18/00, S.90ff., Weka Fachzeitschriften-Verlag, 2000.

[HBP+00] P. Hadjidoukas, V. Barekas, E. Polychronopoulos and T. Papatheodorou. *A Portable Kernel-Mode Resource Manager on Windows 2000 Platforms*. Paper, HPIS Laboratory, University of Patras, Greece, 2000.

[Hel96] A. Helmke. *Laufzeitbetrachtungen an einem PC mit einem PentiumPro Prozessor und dem Realzeit-Betriebssystem VxWorks*. Diplomarbeit, Lehrstuhl für Realzeit-Computersysteme, Technische Universität München, 1996.

[Heu84] W. Heusler. *Lexikon der modernen Wirtschaftspraxis*. Band 8, Zweiburgen Verlag, Weinheim, 1984.

[Hez01] E. Hezemans. *Firewire – Echtzeit-Garant für Motion Control?* Artikel, Computer & Automation, Heft 1/01, S.76-79, Weka Fachzeitschriften-Verlag, 2001.

[Hil97] D. Hildebrand. *Implementing the Win32 API over a Posix Real-time OS*. Artikel, Real-Time Magazine, 2/97, 1997.

[Hol98] E. Holtkamp. *Das zweite Leben der SPS: Löst die Software-SPS die klassische SPS ab?* Artikel, Elektronik, Heft 8/98, S.50-54, Weka Fachzeitschriften-Verlag, 1998.

[HR01] A. Heursch and H. Rzehak. *Rapid Reaction Linux*. Paper, In Proceedings of the 5.th Annual Linux Showcase & Conference, Oakland, California, USA, Nov. 2001.

[HRS01] D. Hatebur, T. Rottke und R. Swik. *Windows CE versus Embedded Linux*. Artikel, Computer & Automation, Heft 4/01, S.88-94, Weka Fachzeitschriften-Verlag, 2001.

[Hub97] R. Huber. *Ansätze für Realzeitsysteme mit Windows NT*. Paper, Lehrstuhl für Realzeit-Computersysteme, Technische Universität München, 1997.

[IM00] Intel and Microsoft. *PC2001 System Design Guide: A Technical Reference for Designing PCs and Peripherals for the Microsoft Windows Family of Operating Systems*. White Paper, Intel, Microsoft, USA, 2000.

[Int98] Intel. *Write Combining Memory Implementations Guidelines*. White Paper, Intel, USA, 1998.

[Int99/2] Intel. *Intel 820 Chipset*. White Paper, Nr. 298162-001, Intel, USA, 1999.

[Int99] Intel. *Interrupt Latency in 80386EX Based System*. White Paper, Intel Knowledge Base, USA, 1999.

[Jan00] M. Janetschke. *Echtzeit in der Industrie, Aktuell zum Thema: SICOMP*. White Paper, Siemens, 2000.

[JBF+96] M. Jones, J. Barrera, A. Forin et al. *An Overview of the Rialto Real-time Architecture*. Technical Report, MSR-TR-96-13, Microsoft, Proceedings of the ACM SIGOPS 96, Sep. 1996.

[JJ01] A. Jaleel and B. Jacob. *Improving the Precise Interrupt Mechanism of Software-Managed TLB Miss Handlers*. Paper, Electrical & Computer Engineering, University of Maryland, USA, 2001.

[JR00] M. Jones and J. Regher. *Predictable Scheduling for Digital Audio*. Technical Report, MSR-TR-2000-87, Microsoft Research, Redmond, USA, 2000.

[JR99] M. Jones and J. Regher. *CPU Reservations and Time Constraints: Implementation Experience on Windows NT*. Technical Report, MSR-TR-99-59, Microsoft Research, Redmond, USA, 1999.

[JRR97] M. Jones, D. Rosu and M. Rosu. *CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities*. Technical Report, MSR-TR-97-19, Microsoft Research, Redmond, USA, 1997.

[JS00] M. Jones and S. Saroiu. *Predictable Scheduling for a Soft Mode*m. Technical Report, MSR-TR-2000-88, Microsoft Research, Redmond, USA, 2000.

[Kra01] T. Kramer. *Hardware-synchronisierte Datenerfassung in Echtzeit*. Artikel, Embedded Systems, Heft 3/01, S.32f, AWi Aktuelles Wissen Verlag, März 2001.

[Kre97] D. Kresta. *Reliable Control with Windows NT*. Paper, Radisys Corporation, Hillsboro, USA, 1997.

[Kro00] J. Kroll. *Java in Embedded Systemen*. Artikel, Elektronik, Heft 3/00, Weka Fachzeitschriften-Verlag, 2000.

[Law92] H. Lawson. *Parallel Processing in Industrial Real-Time Applications*. Prentice Hall, 1992

[LCN98] C. Lin, H. Chu and K. Nahrstedt. *A Soft Real-time Scheduling Server on Windows NT*. Paper, Department of Computer Science, University of Illinois, USA, 1998.

[Mai00] M. Maierhofer. *Efficient Arbitration and Bridging Techniques for High-Performance Conventional Multimedia Servers*. Ph.D. Thesis, University of Teesside, Belgium, 2000.

[MHE+97] E. Maas, D. Herrmann, R. Ernst et al. *A Processor-Coprocessor Architecture for High End Video Applications*. Paper, In Proceedings of ICASSP'97, p.595-598, Munich, Germany, April 1997.

[Mic99] Microsoft. *Key Benefits of the I/O APIC*. Technical Paper, Microsoft, Redmond, USA, 1999.

[MS97] L. Moll and M. Shand. *Systems Performance Measurement on PCI Pamette*. Paper, Digital Equipment Corporation, Systems Research Center, Palo Alto, USA, 1997.

[MS98] A. Mertin und P. Sieverding. *Bloß nicht zu viele Interrupts, Robotersteuerung unter Windows NT*. Artikel, Elektronik, Heft 8/98, S.118-122, Weka Fachzeitschriften-Verlag, 1998.

[Mun97/2] H. Munz. *LP-RTWin Toolkit*. Artikel, Real-Time Magazine, 2/97, 1997.

[Mun97] H. Munz. *Mit dem Windows-PC in harter Echtzeit steuern*. Artikel, Elektronik, Kongress EchtZeit '97, 1997.

[Nem96] Nematron. *Windows NT for Real-time Control*. White Paper, Nematron, USA, 1996.

[Nil96] J. Nilsson. *Analysis and Design of Real-time Systems with Random Delays*. Licentiate Thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, 1996.

[Obe99] K. Obenland. *Dual Real-Time/Non-Real-Time COE on NT*. Project Presentation, Mitre Corporation, Bedford, USA, 1999.

[OL99] R. Otterbach und R. Leinfellner. *Virtuelles Ausprobieren*. Artikel, Elektronik, Heft 8/99, S.128-134, Weka Fachzeitschriften-Verlag, 1999.

[Pan99] A. Pant. *High Performance Stream Sockets on Virtual Interface Architecture (VIA)*. Paper, NCSA, University of Illinois, USA, 1999.

[Pat01] D. A. Patterson. *Computer Architecture*. Course Paper, Computer Science Division, University of California, Berkeley, USA, 2001.

[PB00] M. Papini and P. Baracos. *Real-time Simulation, Control and HIL with Cots Computing Clusters*. Paper, Opal-RT, Montreal, Canada, 2000.

[PBC+97] M. Petronino, R. Bambha, J. Carswell and W. Burleson. *An FPGA-Based Data Acquisition System for a 95 GHz W-Band Radar*. Paper, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, USA, 1997.

[PCI98] PCISIG. *PCI Local Bus Specification, Revision 2.2*. Spezifikation, PCI Special Interrest Group, 1998.

[Pet99] K. Peteler. *Datenfluss ohne Barrieren*. Artikel, Computer & Automation, Heft 5/99, S.32-33, Weka Fachzeitschriften-Verlag, 1999.

[Pop98] S. Popig. *Untersuchung des Laufzeitverhaltens rechenintensiver Realzeitprogramme auf einem PC mit einem PentiumPro Prozessor und dem Realzeitbetriebssystem VxWorks*. Diplomarbeit, Lehrstuhl für Realzeit-Computersysteme, Technische Universität München, 1998.

[Por98] S. Porter. *Windows NT Real-time Extensions as used in Industrial Automation*. Artikel, Real-Time Magazine, 3/98, 1998.

[Pre99] G. Presher. *Networking Servodrives using IEEE 1394*. Technical Paper, Ormec Inc., USA, 1999.

[Qui97] R. Quinnell. *Tackle real-time applications with Windows NT*. Artikel, EDN Magazine, Cahners Publishing Company, USA, Sept. 1997.

[QW94] H. Quast und R. Wesche. *Der Sprinter – Das SERCOS Interface*. Artikelreihe, Elrad, Heft 4/94, S.71-73; 5/94, S.80-83; 06/94, S.76-79, Heinz Heise Verlag, 1994.

[Rap00] RapidIO Trade Association. *RapidIO*. Presentation Slides, RapidIO Trade Association, http://www.rapidio.org, USA, 2000.

[Rea98] Real-time Consult. *Windows NT Workstation 4.0, Evaluation Report 1.0*. Artikel, Real-Time Magazine, Dec. 1998.

[RH00] J. Regula and B. Huynh. *Ringed Bus Puts a Twist on Switching*. Artikel, EETimes, CMP Media Inc., USA, Oct. 2000.

[Ric95] J. Richter. *Microsoft-Windows-Programmierung für Experten*. Microsoft Press, 1995.

[RIG98] E. Riedel, C. van Ingen and J. Gray. *A Performance Study of Sequential I/O on Windows NT 4*. Technical Report, Carnegie Mellon University and Microsoft Research, In Proceedings of 2nd USENIX Windows NT Symposium, Seattle, USA, Aug. 1998.

[Roc98] Rockwell. *Using the Windows NT Operating System for Soft Real-time Control – Separating Fact from Fiction*. White Paper, Rockwell Automation, Milwaukee, USA, 1998.

[RSG+98] K. Ramamritham, C. Shen, O. Gonzales et al. *Using Windows NT for Real-time Applications*. Paper, Computer Science Department, University of Massachusetts, Amherst, USA, 1998.

[RST98] W. Rehm, F. Seifert und M. Trams. *PCI-Analysator entlarvt lahme Chipsätze*. Artikel, Elektronik, Heft 11/98, S.88-94, Weka Fachzeitschriften-Verlag, 1998.

[RSZ89] K. Ramamritham, J. Stankovic and W. Zhao. *Distributed scheduling of tasks with deadlines and resource requirements*. Artikel, IEEE Transactions on Computers, 38(8), USA, Aug. 1989.

[RZN97] M. Rogosin, M. Zimmerman and N. Nachiappan. *Extending Windows NT for Embedded and Hard Real-time Systems*. Artikel, Real-Time Magazine, 2/97, 1997.

[Sch98] A. Scharf. *Java und Echtzeit: Die Warterei hat ein Ende*. Artikel, Elektronik, Heft 18/98, S.88-94, Weka Fachzeitschriften-Verlag, 1998.

[Sig98] Signatec. *Signatec Auxiliary Bus, Version 3*. Product Information Sheet, Signatec Inc., Corona, USA, 1998.

[SP95] S. Sommer and J. Potter. *Operating System Extensions for Dynamic Real-time Applications*. Technical Paper, Microsoft Research Institute and Macquarie University, Australia, 1995.

[SP96] S. Sommer and J. Potter. *An Overview of the Real-time DREAMS Extensions*. Paper, In Proceedings of the 3rd Conference on Parallel and Real-time Systems (PART'96), Brisbane, Australia, Sep. 1996.

[Spi96] M. Spiel. *Multimedia – Das Wort des Jahres 1995*. Seminar „Verteilte Multimediale Systeme", Lehrstuhl Prof. Effelsberg, Universität Mannheim, Juni 1996.

[SR93] J. Stankovic and K. Ramamritham. *What is Predictability for Real-time Systems?* Editorial, Dept. of Computer and Information Science, University of Massachusetts, Amherst, USA, 1993.

[Sta98] N. Stam. *Inside PC Labs: PCI Problems*. Artikel, PC Magazine, Ziff Davis Media Inc., USA, Jun. 1998.

[Sto00] J. Stohr. *Untersuchungen zur Eignung der Intel SMP Architektur für Realzeitsysteme*. Diplomarbeit, Lehrstuhl für Realzeit-Computersysteme, Technische Universität München, 2000.

[Str98] H. Strass. *PCI-Bus um Faktor 32 beschleunigt*. Artikel, Elektronik, Heft 21/98, S.88-96, Weka Fachzeitschriften-Verlag, 1998.

[Süs98] G. Süss. *Weiche und harte Steuerungskonzepte für PCs*. Artikel, Elektronik, Heft 8/98, S.56-58, Weka Fachzeitschriften-Verlag, 1998.

[TBU98] M. Timmermann, B. van Beneden and L. Uhres, *Windows NT Real-time Extensions better or worse?* Artikel, Real-Time Magazine, 3/98, 1998.

[Ten00] TenAsys. *Intime Software Overview Guide*. Technical Paper, TenAsys Corporation, Beaverton, Oregon, USA, 2000.

[TFG+01] R. Scott Tetrick, B. Fanning, R. Greiner et al. *A Discussion of PC Platform Balance: The Intel Pentium 4 Processor-Based Platform Solutions*. Technical Paper, Desktop Platforms Group, Intel Corporation, USA, 2001.

[TM97/2] M. Timmerman and J. Monfret. *Windows NT Real-time Extensions – An Overview*. Artikel, Real-Time Magazine, 2/97, 1997.

[TM97] M. Timmerman and J. Monfret. *Windows NT as Real-time OS?* Artikel, Real-Time Magazine, 2/97, 1997

[Tör98] M. Törngren. *Fundamentals of Implementing Real-time Control Applications in Distributed Computer Systems*. Paper, Dept. of Machine Design, Royal Institute of Technology, Stockholm, Sweden, 1998.

[TYC+96] M. Taylor, T. Yager, R. Caille et al. *Open, Modular Architecture Controls at GM Powertrain*. White Paper, v1.0, General Motors Powertrain Group, USA, May 1996.

[Via01] VIA Technologies. *VIA Apollo KT266 Chipset*. White Paper, VIA Technologies, Taiwan, 2001.

[VM99] P. Viscarola and W. Mason. *Windows NT Device Driver Development*. Macmillan Technical Publishing, USA, 1999.

[Vmi98] VMIC. *Hard Real-time Extensions of Windows NT Evaluation Report*. White Paper, General Motors Powertrain Group, USA, 1998.

[WGT97] E. Woitzel, F. Golatowski und D. Timmermann. *Windows NT im industriellen Einsatz*. Paper, Institut für Angewandte Mikroelektronik und Datentechnik, Universität Rostock, 1997.

[Wil00] T. Williams. *PLX Debuts Adaptive Switch Fabric Architecture*. Artikel, IndustryWatch, USA, Nov. 2000.

[WNT95] B. Wittenmark, J. Nilsson and M. Törngren. *Timing Problems in Real-time Control Systems*. Paper, In Proceedings of the American Control Conference, Seattle, Washington, USA, 1995.

[Wol97] J. F. Wollert. *Doping für den Betriebssystem-Kern, Echtzeit-Lösungen für Windows NT im Vergleich*. Artikel, Elektronik, Heft 20/97, S.46-59, Weka Fachzeitschriften-Verlag, 1997.

[Wor99] B. Worthington. *IDE Ultra/33 Performance: Intel PIIX4E*. Final Report, 1/8/99, NT Performance Group, Microsoft Corporation, 1999.

[Yod99] V. Yodaiken. *The RTLinux Manifesto*. Paper, In Proceedings of the 5th Linux Expo, Raleigh, USA, March 1999.

Weiterführende Literatur

[AHK+96] G. Acher, H. Hellwanger, W. Karl and M. Leberecht. *A PCI-SCI Bridge for Building a PC Cluster with Distributed Shared Memory*. Paper, SCIzzl Workshop, LRR, Technische Universität München, 1996.

[Amd00] AMD. *AMD 200MHz System Bus Technology*. White Paper, AMD, Sunnyvale, USA, 2000.

[Ano00] Anonymous. *Softe Steuerung für harte Echtzeit*. Artikel, Fachzeitschrift elektrotechnik, Heft 4/00, S.34-36, Vogel Verlag, 2000.

[Ano98] Anonymous. *Can NT 4 be used as an RTOS?* Artikel, Real-time Magazine, Dedicated Systems Experts Verlag, Nov. 1998.

[BH97] R. Baumgartl und H. Härtig. *Efficient Communication Mechanisms for DSP-based Multimedia Accelerators*. Paper, In Proceedings of ICSPAT´97, San Diego, USA, Sep. 1997.

[BH98] R. Baumgartl und H. Härtig. *DSPs as flexible Multimedia Accelerators*. Paper, In Proceedings of EDRC'98, Paris, France, Sep. 1998.

[BHP99] L. Böszörmenyi, G. Hölzl und E. Pirker. *Parallel Cluster Computing with IEEE1394-1995*. Paper, Institut für Informationstechnologie, Universität Klagenfurt, Österreich, 1999.

[BJ95] G. Bollella and K. Jeffay. *Support for Real-time Computing within General Purpose OS*. Paper, In Proceedings of the RTAS'95, Chicago, USA, 1995.

[BJE+98] R. Bhargava, L. John, B. Evans and R. Radhakrishnan. *Evaluating MMX Technology Using DSP and Multimedia Applications*. Paper, Electrical and Computer Engineering Department, University of Texas, Austin, USA, 1998.

[Bla00] J. Blanco. *Easing Device Driver Development for Embedded Systems*. Paper, In Proceedings of Embedded Systems Conference (ESC), Chicago, USA, Feb. 2000.

[Bol97] G. Bollella. *Slotted Priorities: Supporting Real-time Computing Within General-Purpose Operating Systems*. Ph.D. Thesis, Department of Computer Science, University of North Carolina, Chapel Hill, USA, 1997.

[BOS+00] R. Baumgartl, I. Oeser, D. Schreiber und M. Schwind. *Signalprozessoren als Koprozessoren für Linux*. Paper, In Proceedings of PEARL 2000 – Echtzeitbetriebssysteme und Linux – Workshop, Boppard, Deutschland, Nov. 2000.

[BS01] H. Baden und T. Schröder. *Auf neuem Terrain – Windows CE*, Artikel, Computer & Automation, Heft 4/01, S.82-87, Weka Fachzeitschriften-Verlag, 2001.

[Bur98/2] D. Burger. *Hardware Techniques to Improve the Performance of the Processor/Memory Interface*. Ph.D. Thesis, University of Wisconsin-Madison, USA, 1998.

[CEC+96] J. Chen, Y. Endo, K. Chan et al. *The Measured Performance of Personal Computer Operating Systems*. Artikel, ACM Transactions on Computer Systems (TOCS), USA, Feb. 1996.

[CH99] E. Cota-Robles and J. Held. *A Comparison of Windows Driver Model Latency Performance on Windows NT and Windows 98*. Paper, Intel, In Proceedings of USENIX OSDI'99, USA, 1999.

[Cha98] A. Charlesworth. *Starfire – Extending the SMP Envelope*. Artikel, IEEE Micro Journal, IEEE Organization, USA, Jan/Feb. 1998.

[CHS+98] J. Carter, W. Hsieh, L. Stoller et al. *Impulse: Building a Smarter Memory Controller*. Paper, Department of Computer Science, University of Utah, Salt Lake City, USA, 1998.

[Com98] Compaq. *Highly Parallel System Architecture (HPSA)*. Technology Brief, Compaq, USA, 1998.

[Com99] Compaq. *8-Way Multiprocessing Architecture*. Technology Brief, Compaq, USA, 1999.

[Deh00] F. Dehmelt. *Serie Buslösungen: Störungen ausgleichen, Differenzielle Übertragungsstandards*. Artikel, ElektronikPraxis, Heft 9/00, S.60-64, Vogel Verlag, Mai 2000.

[Dem93] K. Dembowski. *Computerschnittstellen und Bussysteme*. Markt & Technik Verlag, 1993.

[Des00] J. Desposito. *Triple-Port PCI-to-PCI Bridge Increases Support for System Expansion Slots*. Artikel, Electronic Design, 5/00, p.60-62, Penton Media, USA, 2000.

[Dib00] P. Dibble. *Introduction to Device Driver Design*. Paper, Microware Systems Corporation, In Proceedings of Embedded Systems Conference (ESC), USA, Fall 2000.

[Dra97] T. Drayer. *A Design Methodology for Creating Programmable Logic-based Real-time Image Processing Hardware*. Ph.D. Thesis, Bradley Department of Electrical Engineering, Virginia State University, USA, 1997.

[Dy99] DY4 Systems. *Virtual Interface Architecture Primer*. Technical Paper, DY4 Systems Inc., USA, 1999.

[FCH+96] D. Feitelson, P. Corbett, Y. Hsu and J. Prost. *Parallel I/O Systems and Interfaces for Parallel Computers*. Artikel in: C. L. Wu, Multiprocessor Systems – Design and Integration, World Scientific Pub Co., USA, 1996.

[FKM+97] F. Fischer, T. Kolloch, A. Muth and G. Färber. *A configurable target architecture for rapid prototyping high performance control systems*. Paper, In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97), Las Vegas, Nevada, USA, June 1997.

[FP99] W. Flanagan and D. Passmore. *Fiber Futures: Circuit-Switched IP Backbones*. White Paper, The Burton Group, USA, 1999.

[FTY98/2] N. Frampton, J. Tsao and J. Yen. *Windows CE Evaluation Report*. White Paper, General Motors Powertrain Group, USA, 1998.

[Gen00] General Micro Systems. *VxWorks R.A.M.P/MK*. Technical Paper, General Micro Systems, Rancho Cucamonga, USA, 2000.

[Gle00] S. Glenn. *AMD-760 Chipset With DDR SDRAM Memory Support*. White Paper, AMD, Sunnyvale, USA, 2000.

[GO98] J. Gutleber and L. Orsini. *Gather/Scatter – A Behavioural Pattern for Efficient I/O*. Paper, CERN, Geneva, Switzerland, 1998.

[Gop00] K. Gopalan. *Real-time Support in General Purpose Operating Systems*. Paper, Computer Science Dep. University of New York, USA, 2000.

[GP98] W. Gerth und T. Probol. *Durch abbrechbare Systemaufrufe zu minimalen Antwortzeiten*. Artikel, Elektronik, Heft 13/98, S.114-120, Weka Fachzeitschriften-Verlag, 1998.

[GTL98] S. Gerard, F. Terrier and A. Lanusse. *Developing applications with the Real-time Object Paradigm: A Way to Implement Real-time System on Windows NT*. Artikel, Real-Time Magazine, 3/98, 1998.

[Hen97] N. Henderson. *Nucleus MNT and Nucleus VNET Windows NT and 95 Based Version of Nucleus PLUS*. Artikel, Real-Time Magazine, 2/97, 1997.

[Hew00] Hewlett-Packard. *HP NetServer 6000 Familiy – Technology Innovations*. White Paper, Hewlett-Packard, USA, 2000.

[HJ96] J. Hansen and E. Jul. *A Scheduling Scheme for Network Saturated NT Multiprocessors*. Paper, Department of Computer Science, University of Copenhagen, Denmark, 1996.

[Hof99] A. Hoffmann. *Schnelle Bussysteme*. Artikel, PC Professionell. Heft 6/99, S.256ff., VNU Business Publications, 1999.

[HP95] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. 2nd Ed., Morgan Kaufmann Publishers, 1995.

[HST99] L. Herbert, A. Skjellum and C. Taylor. *MPI for Windows NT*. Paper, Mississipi State University, USA, 1999.

[Huc00] J. Huchzermeier. *Serie Buslösungen: Im Dschungel der Bus-Konzepte, Übertragungsmethoden und –topologien der Datenkommunikation*. Artikel, ElektronikPraxis, Heft 7/00, S.28-34, April 2000.

[Huc00/2] J. Huchzermeier. *Serie Buslösungen: Die Masse macht's, Datenübertragung über parallele Bussysteme*. Artikel, ElektronikPraxis, Heft 8/00, S.74-78, Vogel Verlag, April 2000.

[Int98/2] Intel. *Tools and Techniques for Softmodem IHV/ISV Self-Validation of Compliance with PC 99 Guidelines for Driver-Based Modems*. Technical Paper, Revision 1.0, Intel Architecture Labs, Intel Corporation, USA, July 1998.

[Int99/3] Intel. *PCI/ISA Bridge 82371(PIIX4) Functional Description*. Data Sheet, Intel, USA, 1999.

[Int99/4] Intel. *Profusion – An 8-way Symmetric Multiprocessing Chipset*. Data Sheet, Intel, USA, 1999.

[JC98] C. Jones and M. Cherepov. *Windows-based Systems and the Win32 API*. Artikel, Real-Time Magazine, 3/98, 1998.

[JR98] M. Jones and J. Regher. *Results from a Latency Study of Windows NT*. Paper, Microsoft Research, Redmond, USA, 1998.

[Kal00] D. Kalinsky. *Architecture of Device I/O Drivers*. Paper, In Proceedings of the Embedded Systems Conference (ESC), Chicago, USA, Feb. 2000.

[KB98] U. Kiffmeier and M. Beine. *Block Diagram Based Real-time Simulation on a Network of Alpha Processors and C40 DSPs*. Paper, In Proceedings of DIPES'98, Paderborn, Germany, 1998.

[Kre97/2] D. Kresta. *Getting Real with NT – Approaches to Real-time Windows NT*. Artikel, Real-Time Magazine, 2/97, 1997.

[KSL99] F. Kuhns, D. Schmidt and D. Levine. *The Performance of a Real-time I/O Subsystem for QoS-enabled ORB Middleware*. Paper, In Proceedings of the International Symposium on Distributed Objects and Applications (DOA'99), Edinburgh, Scotland, September 1999.

[KTA97] K. Kailas, B. Trinh and A. Agrawala. *Temporal Accuracy and Modern High Performance Processors: A Case Study using Pentium Pro*. Technical Report, Department of Computer Science, University of Maryland, USA, 1997.

[Küf98] H. Küfner. *Gesucht: Das optimale PCI-Interface*. Artikelreihe, Elektronik, Heft 11/98, S.104-108; 13/98, S.102-109; 15/98, S.100-106; 17/98, S.85-91, Weka Fachzeitschriften-Verlag, 1998.

[LCB+98] D. Lee, P. Crowley, J. Baer et al. *Execution Characteristics of Desktop Applications on Windows NT*. Paper, In Proceedings of ISCA'98, Barcelona, Spain, 1998.

[LL97] J. Laudon and D. Lenoski. *The SGI Origin: A ccNUMA Highly Scalable Server*. Paper, Silicon Graphics, Mountain View, CA, USA, 1997.

[Mad97] T. Madhyastha. *Automatic Classification of Input/Output Access Patterns*. Ph.D. Thesis, University of Illinois at Urbana-Champaign, USA, 1997.

[Mes97] H. P. Messmer. *PC-Hardwarebuch*. 5. Auflage, Addison-Wesley Longman, 1997.

[Mic95] Microsoft. *Real-Time Systems and Microsoft Windows NT*. Technology Brief, Microsoft Corporation, Redmond, USA, 1995.

[Mot00] Motorola. *RapidIO – An Embedded System Component Network Architecture*. White Paper, Motorola Semiconductor Product Sector, Austin, Texas, USA, 2000.

[Myr00] Myriad Logic. *Fibre Channel for Real-Time Applications*. White Paper, Myriad Logic, Silver Spring, MD, USA, 2000.

[Naj00] H. Najafzadeh. *Source Code Instrumentation and its Perturbation Analysis in Pentium II*. Paper, Computer Science Department, University At Albany, USA, 2000.

[Nil00] K. Nilsen. *Real-time Java*. Article, Java Developers Journal, SYS-CON Media, USA, May 2000.

[Nil98] J. Nilsson. *Real-time Control Systems with Delays*. Ph.D. thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, 1998.

[Now01] J. Nowling. *Ethernet – Just how fast is it?* Artikel, Control Solutions, 03/01, PennWell Corporation, USA, 2001.

[Pei99] P. Peisker. *High-Speed-Bus IEEE 1394: Reif für die Anwendung*. Artikelreihe, Elektronik, Heft 6/99, S.42-46; 8/99, S.108-111, Weka Fachzeitschriften-Verlag, 1999.

[Pla01] B. Plagemann. *Blick zurück nach vorn?* Artikel, Computer & Automation, Heft 4/01, S.78-81, Weka Fachzeitschriften-Verlag, 2001.

[Pre99/2] G. Presher. *Firewire sparks new drive network*. Artikel, Motion System Design, 2/99, p.25-27, Penton Media Inc., USA, 1999.

[PS98] P. Petersen and T. Schotland. *Win32: A Suitable Standard for Real-time Embedded Systems*. Artikel, Real-Time Magazine, 3/98, 1998.

[Qui98] QuickLogic. *PCI Master Target Application Note*. Technical Paper, QAN15, QuickLogic, USA, 1998.

[Rad98] Radisys. *Windows NT Thread Latency Measurements*. Technical Paper, Radisys Corporation, Hillsboro, Orgeon, USA, 1998.

[Ram96] Rambus. *Rambus Memory: Multi-Gigabytes/Second And Minimum System Cost*. Technical Paper, Rambus Inc., USA, 1996.

[RBK00] F.-M. Renner, J. Becker, A. Kirschbaum et al. *Synthese von Kommunikationsstrukturen und architekturgenaues Rapid-Prototyping eingebetteter Echtzeitsysteme*. Artikel, it+ti, Heft 2/00, S.27-33, Oldenbourg Verlag, 2000.

[Rea00] Real-time Consult. *What Makes a Good RTOS? Evaluation Report 2.0*. Artikel, Real-Time Magazine, 2/00, 2000.

[Ril99] D. Riley. *PCI-X Architecture Overview*. Presentation, Industry Standard Server Division, Compaq, USA, 1999.

[RO99] L. Rosen and K. Obenland. *The Interoperability Between Real-time and Non-real-time Processing on a Windows NT Platform*. Technical Report, MTR99W0000131, MITRE, McLean, Virginia, USA, 1999.

[SBW+97] B. Spitzer, A. Burst, M. Wolff and K. Müller-Glaser. *Interface Technologies for Versatile Rapid-Prototyping Systems*, Paper, In Proceedings of RSP'99, Tampa, Florida, USA, June 1999.

[Sch97] A. Schwarz. *Automatisieren mit Windows – die größte Versuchung, seit es Software gibt*. Artikelreihe, Elektronik, Heft 15/97, S.64-70; 17/97, S.72-77; 19/97, S.86-92, Weka Fachzeitschriften-Verlag, 1997.

[SG98] C. Shen and O. Gonzales. *Real-time Communicating Tasks on COTS-based Distributed Platforms*. Paper, Mitsubishi Electric Research Lab, Cambridge, USA, 1998.

[SH97] J. Simons and O. Heinz. *SCI Multiprocessor PC Cluster with Windows NT*. Paper, Paderborn Center for Parallel Computing, 1997.

[SNR92] J. Stankovic, D. Niehaus and K. Ramamritham. *SpringNet – A Scalable Architecture for High Performance Predictable and Distributed Real-time Computing*. Paper, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, USA, 1992.

[Str00] H. Strass. *Abschied vom Bus: RapidIO*. Artikel, Elektronik, Heft 24/00, S.52-55, Weka Fachzeitschriften-Verlag, 2000.

[Str99] H. Strass. *Verteilte Rechenaufgaben, Lose gekoppeltes Multiprocessing mit CompactPCI*. Artikel, Elektronik, Heft 6/99, S.96-100, Weka Fachzeitschriften-Verlag, 1999.

[SYS+96] M. Seltzer, Y. Endo, C. Small and K. Smith. *Dealing with Disaster: Surviving Misbehaved Kernel Extensions*. Paper, In Proceedings of the 2nd Usenix Symposium, Seattle, Washington, USA, Oct. 1996.

[Tal98] N. Talbert. *The Cost of COTS*. Artikel, Computer, IEEE Computer, IEEE Organization, USA, June 1998.

[TBU98/2] M. Timmermann, B. van Beneden and L. Uhres. *RTOS Evaluation Kick Off*. Artikel, Real-Time Magazine, 3/98, 1998.

[Thi94] K.-D. Thies. *Multitasking*. Hanser Verlag, München, 1994.

[TLG98] R. Thakur, E. Lusk and W. Gropp. *I/O in Parallel Applications: The Weakest Link*. Artikel, International Journal of Supercomputer Applications and High Performance Computing, Issue on I/O in Parallel Applications, MIT Press, USA, 1998.

[Tör95] M. Törngren. *A Perspective to the Design of Distributed Real-time Control Applications based on CAN*. In Proceedings of the International CiA CAN Conference, London, Oct. 1995.

[TRS99] M. Trams, W. Rehm and F. Seifert. *An Advanced PCI-SCI Bridge with VIA Support*. Paper, Fakultät für Informatik. Technische Universität Chemnitz, 1999.

[TSR00] M. Trams, R. Schlosser and W. Rehm. *Design Choices and First Results of our VIA-capable PCI-SCI Bridge*. Paper, In Proceedings of CLUSTER2000, IEEE International Conference on Cluster Computing, p.349-350, Chemnitz, Germany, 2000.

[Via00] VIA Technologies. *VIA High-Bandwidth Differential Interconnect Technology (HDIT V-Link) Architecture*. Press Release, VIA Technologies, Taipei, Taiwan, Aug. 2000.

[Vmi95] VMIC. *VMICs Rtnet – Real-Time Networking*. White Paper, VME Microsystems International Corporation, Huntsville, Alabama, USA, 1995.

[Vmi98/2] VMIC. *The RTX Real-time Subsystem for Windows NT*. White Paper, General Motors Powertrain Group, USA, 1998

[Vog01] K. Vogel. *PCI-X-to-PCI-X-Bridge-Baustein beschleunigt Datentransfer*. Artikel, Embedded Systems, Heft 3/01, S.56f, AWi Aktuelles Wissen Verlag, März 2001.

[VRR00] E. Vonnahme, S. Rüping and U. Rückert, *Measurement in Switched Ethernet Networks Used for Automation Systems*. Paper, Heinz Nixdorf Institute, University of Paderborn, Germany, 2000.

[Wal01] K-D. Walter. *Ein Kopf-an-Kopf-Rennen*. Artikel, Computer & Automation, Heft 4/01, S.96-100, Weka Fachzeitschriften-Verlag, 2001.

[Wap00] M. Wappler. *Entwicklung eines Simulationsmodells für einen IO-Prozessor*. Großer Beleg, Lehrstuhl für Betriebssysteme, Technische Universität Dresden, Deutschland, 2000.

[WBL99] M. Wolff, A. Burst und H. M. Lipp. *Kommunikation und Schnittstellen in verteilten Systemen – eine Übersicht*. Artikel, it+ti, Heft 6/99, S.5-16, Oldenbourg Verlag, 1999.

[WD95] C. Weems and S. Dropsho. *Real-Time Computing: Implications for General Microprocessors*. Technical Report, TR-95-42, Amherst Computer Science Department, University of Massachusetts, USA, 1995.

[WD95/2] C. Weems and S. Dropsho. *Real-time RISC Processing*. Paper, Amherst Computer Science Department, University of Massachusetts, USA, 1995.

[WT00] T. Wu and M. Tramontano. *SoftDSL Technology Backgrounder*. White Paper, Motorola, USA, 2000.

[Zoe95] D. Zöbel. *Echtzeitsysteme: Grundlagen und Technik*. International Thomson Publishing, 1995.