(19) Deutsches Patent- und Markenamt

(10) **DE 197 28 726 B4** 2015.07.30

(12) # Patentschrift

(21) Aktenzeichen: **197 28 726.3**
(22) Anmeldetag: **04.07.1997**
(43) Offenlegungstag: **08.01.1998**
(45) Veröffentlichungstag der Patenterteilung: **30.07.2015**

(51) Int Cl.: ***B25J 9/18*** (2006.01)

(30) Unionspriorität:
195352/96 **05.07.1996** **JP**

(73) Patentinhaber:
Seiko Epson Corp., Tokyo, JP

(74) Vertreter:
Hoffmann, Eckart, Dipl.-Ing., 80336 München, DE

(72) Erfinder:
Gomi, Kazuhiro, Suwa, Nagano, JP; Miyazawa, Hiroshi, Suwa, Nagano, JP; Okuyama, Masayuki, Suwa, Nagano, JP; Yokoshima, Norio, Narashino, Chiba, JP; Shioda, Toshimi, Narashino, Chiba, JP

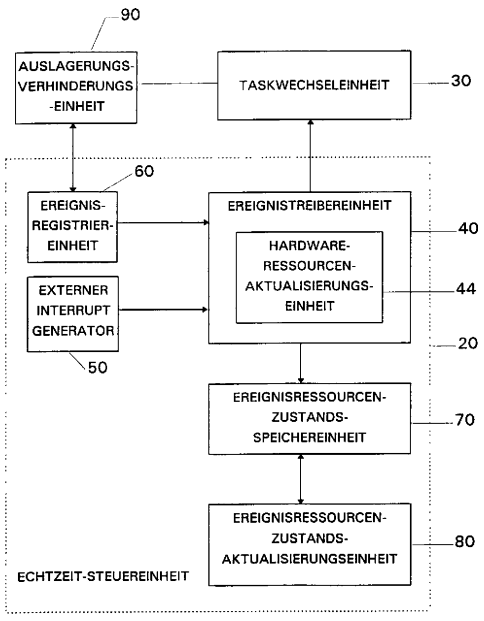
(56) Ermittelter Stand der Technik:

DE	**44 10 775**	**A1**
DE	**44 11 426**	**A1**
DE	**44 45 651**	**A1**
DE	**195 00 957**	**A1**

Bernd Ackermann; "Anforderungen an ein Echtzeit-Betriebssystem für "eingebettete" Systeme"; Fachzeitschrift Elektronik 18/1992, Echtzeit Betriebssysteme, 1992, Seite 120 bis 128,

(54) Bezeichnung: **Robotercontroller und dessen Steuerverfahren**

(57) Hauptanspruch: Robotercontroller umfassend eine Task-Wechseleinrichtung (30), die mit einer preemptiven Multitasking-Funktion versehen ist, und eine Echtzeit-Steuereinrichtung (20), die Steuerungen dadurch ausführt, daß sie die Task-Wechseleinrichtung (30) anweist, Tasks zu wechseln, so daß die Verarbeitung als Reaktion auf das Auftreten eines Ereignisses in Echtzeit ausgeführt wird, wobei
der Robotercontroller ein allgemeines Betriebssystem, welches eine preemptive Multitasking-Funktion aufweist, verwendet,
die Echtzeit-Steuereinrichtung Ereignisse auf regelmäßiger Basis in Wiederhol-Intervallen erfaßt, die kürzer sind als die von dem Betriebssystem zur Verfügung gestellten,
der Robotercontroller eine Ereignistreibereinrichtung (40) enthält, die eine Ereignistreiberverarbeitung ausführt, welche der Task-Wechseleinrichtung einen Wechsel zu dem Task anzeigt, in welchem die mit dem Ereignis verknüpfte Verarbeitung ausgeführt wird, und
die Echtzeit-Steuereinrichtung (20) weiterhin eine externe Interrupt-Generatoreinrichtung (50) umfaßt, die unter Verwendung eines externen Zeitgebers (252) Interruptsignale auf regelmäßiger Basis in den Wiederhol-Intervallen, die zur Ausführung einer Echtzeit-Verarbeitung nötig sind, erzeugt, wobei die Ereignistreibereinrichtung (40) die Ereignistreiberverarbeitung synchron mit diesen Interruptsignalen ausführt.



Beschreibung

[0001] Die Erfindung betrifft einen Robotercontroller und dessen Steuerverfahren. Insbesondere bezieht sich die Erfindung auf Robotercontroller und typische Personal-Computer-(PC)-Betriebssysteme, die die mit dem Auftreten von Ereignissen verknüpfte Verarbeitung auf Echtzeit-Basis ausführen.

[0002] Die meisten herkömmlichen Robotercontroller und ihre Steuerverfahren sind dazu ausgelegt, den Betrieb spezieller Manipulatoren wie SCARA-Manipulatoren und in kartesischen Koordinaten gesteuerte Manipulatoren zu steuern. Zu diesem Zweck setzen die herkömmlichen Robotercontroller (nachfolgend auch als "RC" abgekürzt) unabhängig entwickelte Robotercontroller-Betriebssysteme (RC-Betriebssysteme) ein.

[0003] In den letzten Jahren ist jedoch ein zunehmender Bedarf an Robotercontrollern aufgetreten, die über die Steuerung des Betriebs der Manipulatoren hinaus zur Steuerung des gesamten Systems in der Lage sind, etwa auch periphere Einrichtungen zu steuern oder mit externen Einrichtungen zu kommunizieren. In solchen Fällen muß als RC-Betriebssystem ein zu Multitasking befähigtes Betriebssystem eingesetzt werden. Zur Erfüllung dieser Forderung werden in den meisten Fällen entweder proprietäre Betriebssysteme oder allgemeine Echtzeit-Betriebssysteme verwendet, wie sie etwa unter den Bezeichnungen pSOS® (Warenzeichen der Integrated Corporation), VxWorks® (Warenzeichen der WindRiver Systems Corporation), VTRX® (Warenzeichen der MicroTech Research Corporation), OS-9, etc. bekannt sind.

[0004] Derartige zu Multitasking befähigte Betriebssysteme sind beispielsweise aus der veröffentlichten Patentanmeldung DE 195 00 957 A1 oder aus der Zeitschrift „Elektronik" 18/1992, S. 120-128, ernd Ackermann; "Anforderungen an ein Echtzeit-Betriebssystem für eingebettete Systeme" bekannt.

[0005] Da die Steuerung von Manipulatoren und peripheren Einrichtungen eine Echtzeit-Verarbeitung erfordert, ist die Fähigkeit zu einer solchen Echtzeit-Verarbeitung eine wesentliche Forderung an ein RC-Betriebssystem.

[0006] Die Entwicklung proprietärer Betriebssysteme, die eine Echtzeit-Verarbeitung ermöglichen, erfordert einen enormen Arbeitsaufwand. Während die allgemeinen Echtzeit-Betriebssysteme andererseits problemlos Echtzeit-Verarbeitungsfähigkeit besitzen, führen ihre hohen Kosten zu ökonomischen Problemen.

[0007] In letzter Zeit sind preiswerte PC-Betriebssysteme auf den Markt gekommen und werden in großem Umfang eingesetzt. Wenn solch ein PC-Betriebssystem für einen Robotercontroller eingesetzt werden kann, ist es möglich, sowohl den Aufwand an Entwicklungsarbeit als auch die Kosten wesentlich zu reduzieren. Dieser Weg bietet die zusätzlichen Vorteile des für die Benutzer leichten Trainings, weiterer Verminderung von Entwicklungsvorlaufzeiten wegen der Verfügbarkeit eines ganzen Bündels von Entwicklungswerkzeugen, sowie der Möglichkeit der Ausweitung der Anwendungen durch Verwendung von ab Lager lieferbarer Hardware und Software.

[0008] Obwohl jedoch preiswerte PCs und PC-orientierte Betriebssysteme Multitasking-Fähigkeiten bieten, ist ihr Taskwechselbetrieb langsam, und eine Verminderung der Schaltintervalle ist unmöglich, weshalb es schwierig ist, eine Echtzeit-Verarbeitung sicherzustellen.

[0009] Da, wie zuvor erwähnt, die Steuerung von Manipulatoren und peripheren Einrichtungen eine Echtzeit-Verarbeitung erfordert, kann kein System in der Praxis eingesetzt werden, bei dem eine lange Zeit zwischen dem Auftreten eines Ereignisses und dem Anlauf des Tasks zur Verarbeitung dieses Ereignisses vergeht.

[0010] Ein weiterer komplizierender Faktor ist der, daß, wenn einer der vorerwähnten PCs oder eines der PC-orientierten Betriebssysteme verwendet wird, die Zeitspanne zwischen dem Auftreten eines Ereignisses und dem Hochlauf (Booten) des Tasks zur Verarbeitung des Ereignisses abhängig von dem Zeitpunkt, zu dem das Ereignis auftrat, sehr stark variieren kann. Diese Tatsache vermindert die Qualität der Verarbeitung im Hinblick auf die Reproduzierbarkeit.

[0011] Wenn daher ein schneller Taskwechsel und eine Verminderung der Bootzeitschwankung bei irgendeinem der vorgenannten PCs und PC-Betriebssysteme, den sogenannten Nicht-Echtzeit-Betriebssystemen, erzielbar ist, können Robotercontroller mit geringerem Entwicklungsaufwand, mit höherem Funktionalitätsgrad, mit verbesserter Benutzer-Erlernbarkeit und mit größerem Erweiterungspotential durch die Verwendung von ab Lager erhältlicher Hardware und Software geschaffen werden, während die den vorgenannten PC-Betriebssystemen eigenen verschiedenen Vorteile voll genutzt werden.

[0012] Die vorliegende Erfindung befaßt sich mit diesen Problemen, und ihre Aufgabe ist es, einen Robotercontroller sowie dessen Steuerungsverfahren unter Verwendung eines allgemeinen PCs und eines allgemeinen Betriebssystems für PCs zu schaffen.

[0013] Diese Aufgabe wird erfindungsgemäß mit einem Roboterantriebscontroller gemäß Anspruch 1

bzw. einem Steuerverfahren gemäß Anspruch 6 gelöst.

[0014] Im Rahmen des vorliegenden Textes bezieht sich der Begriff ”preemptive-Methode” allgemein auf die Unterteilung der CPU-Verarbeitung in feste Zeitscheiben, so daß das Betriebssystem den Anwendungen CPU-Zeit nach Maßgabe eines Prioritätenschemas zuweist. Anders ausgedrückt, der Begriff schließt den Fall ein, daß, bevor eine Anwendung die CPU freigibt, das Betriebssystem mit jeder Zeiteinheit die Verarbeitung übernimmt und die CPU zu einer anderen Anwendung umschaltet.

[0015] Wenn folglich mehrere Anwendungen in einem preemptiven Multitasking-Schema ausgeführt werden, braucht eine Anwendung nicht darauf zu warten, daß eine andere Anwendung ihre Verarbeitung beendet. Dies reduziert wesentlich die von dem Benutzer realisierte Wartezeit und verbessert den scheinbaren Wirkungsgrad des Computers.

[0016] Gemäß der im Anspruch 1 definierten Lösung wird in bestimmten Wiederhol-Intervallen festgestellt, ob ein Ereignis aufgetreten ist, so daß der Task, in welchem die mit dem Ereignis verbundene Verarbeitung ausgeführt wird, angesteuert werden kann. Indem somit eine Ereignistreiberverarbeitung bei jedem Wiederhol-Intervall, das zur Handhabung von Ereignissen auf einer Echtzeit-Basis nötig ist, ausgeführt wird, können Steuerungen so bewirkt werden, daß die Verarbeitung zur Handhabung der Ereignisse in Echtzeit ausgeführt wird.

[0017] Ein weiteres Aspekt ist ein integriertes Management der zu verarbeitenden Ereignisse, was das Management von Ereignissen vereinfacht.

[0018] Gemäß dem Verfahren nach Anspruch 6 kann ein externer Zeitgeber zur Erzeugung von externen Interruptsignalen zu den Wiederhol-Intervallen eingesetzt werden, die ausreichen, um auf das Auftreten von Ereignissen in Echtzeit zu reagieren. Indem eine Ereignistreiberverarbeitung synchron mit den Interruptsignalen durchgeführt wird, können selbst typische PCs und typische PC-Betriebssysteme, die aufgrund von Zeitscheiben oder internen Interrupts auf der Basis eines Systemzeitgebers die vorgenannten Wiederhol-Intervalle nicht gewährleisten können, eine Ereignistreiberverarbeitung in Echtzeit steuern.

[0019] Die Task-Wechseleinrichtung umfaßt vorzugsweise eine Zeitteilungseinrichtung, die die Zeit mit den vorgenannten Wiederhol-Intervallen unterteilt, das heißt Zeitscheiben bildet, so daß die Ereignistreibereinrichtung einseitig eine Ereignistreiberverarbeitung zu jeder von der Zeiteinteilungseinrichtung erzeugten Zeitscheibe ausführt. Der Ereignistreiberschritt kann so konfiguriert werden, daß er die Ereignistreiberverarbeitung zu jeder der Zeitscheiben ausführt, die das vorgenannte allgemeine Betriebssystem zu den vorgenannten Wiederhol-Intervallen ausführt.

[0020] Auf diese Weise kann die Ereignistreiberverarbeitung als ein Task implementiert werden, der einseitig zu jeder Zeitscheibe verarbeitet wird. Folglich kann eine ereignisgesteuerte Verarbeitung so gesteuert werden, daß sie in einer einfachen Konfiguration in Echtzeit ausgeführt wird.

[0021] Gemäß den Weiterbildungen der Ansprüche 2 bzw. 7 kann eine dynamische Zuordnung zwischen Ereignissen und den die zugehörige Verarbeitung ausführenden Programmen erstellt werden, was eine effektive Nutzung der Systemressourcen sicherstellt. Durch Rücksetzen des Ereigniserwartungszustands am Ende der Ereignistreiberverarbeitung ist es möglich, Ereigniserwartungszustände und Ereigniserwartungsfreigabeaktionenen in Echtzeit zu verfolgen.

[0022] Die Hardwareressourcen in den Ansprüchen 3 bzw. 8 umfassen I/O-(Eingabe/Ausgabe)-Ports, auf den Speicher abgebildete I/O-Ports, die eigentlichen Manipulatoren, System-I/O-Einheiten, die in einer Treiberbox untergebracht sind, und spezielle mit den ISA-Bus verbundene Schaltungskarten.

[0023] Der Begriff ”Programm” bezieht sich auf Benutzertasks, die Manipulatoroperationen und periphere Einrichtungen steuern, sowie auf Programme, die in Systemtasks ausgeführt werden, um unter anderem die internen Zustände von Controllern zu überwachen.

[0024] Indem somit die verschiedenen Zustände, die in einem Roboter auftreten, als Ereignisse behandelt werden, ist es möglich, die Verarbeitung entsprechend diesen Ereignissen in Echtzeit auszuführen.

[0025] Gemäß der Weiterbildung der Ansprüche 4 bzw. 9 sieht die Erfindung eine Ereignisressourcentabelle vor, die die Zustände von Ereignisressourcen in einem gemeinsam genutzten Speicherbereich speichert, der von mehreren Tasks abgefragt und aktualisiert werden kann. Der zutreffende Bereich in der vorgenannten Ereignisressourcentabelle entsprechend den Zuständen der Ereignisressourcen, die durch die Ausgabe von Programmen, die in den verschiedenen Tasks verarbeitet werden, geändert werden, wird aktualisiert. Durch Abfragen der vorgenannten Ereignisressourcentabelle ist es daher möglich, Ereignisse wahrzunehmen, die von Programmausgaben herrühren. Dies erlaubt eine wirkungsvolle Erfassung von Ereignissen.

[0026] Da ferner die tatsächlichen Hardwareressourcen aufgrund von Änderungen in den Ereignisressourcen, die sich in der Ereignisressourcen-

tabelle widerspiegeln, aktualisiert werden können, brauchen die in verschiedenen Tasks ausgeführten Programme die tatsächlichen Hardwareressourcen nicht zu aktualisieren. Somit führt die Integration der Hardwareressourcen-Aktualisierungsprozesse zu einer erhöhten Verarbeitungsgeschwindigkeit.

[0027] Normalerweise müssen Programme, die die Tatsache angemeldet haben, daß sie auf das Auftreten eines Ereignisses warten, in Echtzeit verarbeitet werden, wenn das Ereignis auftritt. Wenn das Programm jedoch ausgelagert wurde, nimmt das Neuladen des Programms mehrere zehn oder sogar hunderte Millisekunden in Anspruch. Das Verarbeitungsschema der vorliegenden Erfindung startet jedoch vorzugsweise die Programme, die auf ein Ereignis warten, auf periodischer Basis unabhängig davon, ob tatsächlich ein Verarbeitungserfordernis besteht. Dies verhindert das Auslagern der Programme und kann damit die Ausführung einer Echtzeit-Verarbeitung gewährleisten.

[0028] Ausführungsbeispiele der Erfindung werden nachfolgend anhand der Zeichnungen im einzelnen erläutert, es zeigen:

[0029] Fig. 1 ein Funktionsblockdiagramm einer Ausführungsform des Robotercontrollers gemäß der vorliegenden Erfindung,

[0030] Fig. 2 die Hardwarekonfiguration des Robotercontrollers in der Ausführungsform von **Fig.** 1,

[0031] Fig. 3 ein Konzeptionsdiagramm eines Beispiels einer von dem Controller ausgeführten Multitasking-Verarbeitung,

[0032] Fig. 4 eine spezielle Abbildung des Ereignisregistrierungsprozesses,

[0033] Fig. 5(A) und (B) Flußdiagramme zur Erläuterung der Prozedur, nach der das System Ereignisse in Echtzeit steuert, und

[0034] Fig. 6 ein Funktionsblockdiagramm des Robotercontrollers gemäß einer anderen Ausführungsform der Erfindung.

[0035] Der Robotercontroller der nachfolgend beschriebenen Ausführungsform der Erfindung steuert entweder Manipulatoraktionen oder die peripheren Einrichtungen. Diese Art von Manipulatoroperation oder Steuerung einer peripheren Einrichtung kann benutzerangepaßt werden, um benutzerspezifischen Steuerungen zu entsprechen. Daher schreibt der Benutzer Anwendungsprogramme für die von den mit dem Controller verbundenen Manipulatoren auszuführenden Operationen in Robotersteuersprachen, die von dem Controller ausgeführt werden können. Der Controller ist so konfiguriert, daß Benutzertasks, in denen benutzergeschriebene Manipulatorsteuerungs-Anwendungsprogramme sowie Tasks, die eine Systemverarbeitung zur Überwachung des internen Zustands des Controllers durchführen, auf Multitasking-Basis ausgeführt werden können.

[0036] Fig. 3 ist ein konzeptionelles Diagramm, das ein Beispiel der Multitasking-Verarbeitung zeigt, die der Controller **10** ausführt. Wie dargestellt, ist der Controller so konfiguriert, daß Programme die die Anwendungen zur Steuerung der Manipulatorsteuerungs-Verarbeitung **120-1**, der Fördersteuerungs-Verarbeitung **120-2**, etc. enthalten, auf Multitasking-Basis als Tasks **110-1**, **110-2**, usw. ausgeführt werden. Genauer gesagt wird die CPU-Zeit in Zeitscheiben unterteilt und mehreren Tasks so zugeordnet, so daß mehrere Tasks scheinbar gleichzeitig ausgeführt werden können.

[0037] Bei dieser Art Robotercontroller muß die Steuerung der Manipulatoren und der peripheren Einrichtungen, die von verschiedenen Programmen ausgeführt werden, in Echtzeit ausgeführt werden. Wenn daher eine lange Zeit zwischen dem Auftreten eines Ereignisses und dem Zeitpunkt vergeht, zu dem der Task, der dieses Ereignis verarbeitet, startet, ist der Controller für praktische Anwendungen nicht sehr nützlich.

[0038] Der Robotercontroller **10** der vorliegenden Ausführungsform enthält die folgende Konfiguration, damit die Verarbeitung der vorerwähnten Ereignisse in Echtzeit ausgeführt werden kann.

[0039] Zunächst soll die erforderliche Hardwarekonfiguration des Systems beschrieben werden, die in **Fig.** 2 gezeigt ist.

[0040] Wie in der Figur gezeigt, sind in dem Controller **10** die Haupt-CPU **210**, die Festplatte (HDD) **220**, das RAM **230** und die Schnittstellenkarte **240** durch den CPU-Bus **280** miteinander verbunden. Über einen Erweiterungsbus **290** sind weiterhin eine Treiberbox **260**, in der Manipulatoren **270** und I/O-Einheiten **262** montiert sind, sowie spezielle Schaltungskarten **250**, die unter anderem einen externen Zeitgeber **252** enthalten, mit der Schnittstellenkarte **240** verbunden.

[0041] Das RAM **230** speichert ein Betriebssystem **232**, welches den Controller steuert, ein Steuerprogramm **234**, das die mit den vorgenannten Ereignissen verbundene Verarbeitung ausführt, und verschiedene Programme und Daten einschließlich vom Benutzer erstellter Anwendungsprogramme für die Robotersteuerung.

[0042] Das Betriebssystem **232** umfaßt ein allgemeines PC-Betriebssystem, bei dem es sich um ein Allzweck-PC-Betriebssystem handelt, das Standardfunktionen wie preemptives Multitasking sowie

die Ereignissynchronisationsfunktion ausweist. Hierbei handelt es sich allerdings nicht um ein sogenanntes Echtzeit-Betriebssystem. Daß es sich nicht um ein Echtzeit-Betriebssystem handelt, verweist auf die Tatsache, daß das Betriebssystem Taskwechsel mit niedriger Geschwindigkeit ausführt und keine Möglichkeit zur Einstellung kurzer Wechselzeiten bietet. Folglich ist das Betriebssystem nicht zur Durchführung einer Echtzeit-Verarbeitung in der Lage. Es sei angemerkt, daß die vorliegende Ausführungsform das als Windows 95® (Warenzeichen der Microsoft Corporation) bekannte Betriebssystem als allgemeines PC-Betriebssystem einsetzt.

[0043] Das Steuerprogramm **234** umfaßt verschiedene Programme und Speicherbereiche, die eine später beschriebene Ereignistreibereinheit **40**, eine Hardwareressourcen-Aktualisierungseinheit **44**, eine Ereignisregistriereinheit **60**, eine Ereignisressourcenzustands-Speichereinheit **70**, eine Ereignisressourcenzustands-Aktualisierungseinheit **80** und eine Auslagerungsverhinderungseinheit **90** implementieren (siehe **Fig.** 1).

[0044] Die später beschriebene, in **Fig.** 1 gezeigte Task-Wechseleinheit **30** umfaßt hauptsächlich die Task-Wechselfunktion des Betriebssystems **232**. In ähnlicher Weise verwenden die Ereignistreibereinheit **40** und die Ereignisregistriereinheit **60** Teile der Ereignissynchronisationsfunktion des Betriebssystems **232**.

[0045] Unter dem Anwendungsprogramm **236** ist ein benutzer-geschriebenes Programm, das in einer Robotersteuersprache codiert ist, zu verstehen, das die von den Manipulatoren auszuführenden Operationen beschreibt. Der Begriff "Ereignis" bezieht sich auf die verschiedenen Ereignisse, die während der Operation eines Manipulators oder während der Steuerung einer peripheren Einrichtung auftreten. Insbesondere schließt der Begriff Ereignisse ein, die in Manipulatoren **270** oder der Treiberbox **260** auftreten und mit Änderungen der System-I/O-Einheiten **262** verbunden sind, Ereignisse die mit Änderungen in Hardwareressourcen wie etwa den Schaltungskarten **250** verbunden sind, sowie Ereignisse, die von den Ausgangsdaten erzeugt werden, die von Programmen **232**, **234** und **236** für Zwecke der gegenseitigen Tasksynchronisation und Kommunikation erzeugt werden. Die System-I/O-Einheiten, die Hardwareressourcen und die Ausgangsdaten, die Ereignisse erzeugen, werden Ereignisressourcen genannt.

[0046] Im folgenden werden die Merkmale erläutert, die den Controller **10** in die Lage versetzen, eine ereignisgebundene Verarbeitung auf Echtzeit-Basis auszuführen.

[0047] **Fig.** 1 ist ein Funktionsblockdiagramm des Controllers **10**. Der Controller **10** umfaßt: die schon erwähnte Task-Wechseleinheit **30**, die Tasks auf preemtiver Basis wechselt, einen externen Interruptgenerator **50**, der einen externen Zeitgeber zur Erzeugung von Interruptsignalen in vorgegebenen Wiederhol-Intervallen verwendet, eine Ereignistreibereinheit **40**, die eine Ereignistreiberverarbeitung in Synchronisation mit den von dem externen Interruptgenerator **50** erzeugten Interruptsignalen ausführt, eine Ereignisregistriereinheit **60**, die die Tatsache registriert, daß das Anwendungsprogramm **136**, das die mit dem Auftreten eines Ereignisses verbundene Verarbeitung ausführt, auf das Auftreten eines Ereignisses wartet, eine Ereignisressourcenzustands-Speichereinheit **70**, die Ereignisressourcenzustände speichert, um das Auftreten von Ereignissen zu verfolgen (sich zu merken), eine Ereignisressourcenzustands-Aktualisierungseinheit **80**, die die Zustände von Ereignisressourcen aktualisiert, welche in der Ereignisressourcenzustands-Speichereinheit **70** gespeichert sind, und eine Auslagerungsverhinderungseinheit **90**, die das Auslagern des Anwendungsprogramms **236** verhindert, welches auf das Auftreten eines Ereignisses wartet. Der externe Interruptgenerator **50**, die Ereignistreibereinheit **40**, die Ereignisregistriereinheit **60**, die Ereignisressourcenzustands-Speichereinheit **70** und die Ereignisressourcenzustands-Aktualisierungseinheit **80** arbeiten als Echtzeit-Steuerungseinrichtung **20**. Diese Komponenten steuern und weisen die Task-Wechseleinheit **30** an, Tasks zu wechseln, so daß die mit dem Auftreten eines Ereignisses verbundene Verarbeitung in Echtzeit ausgeführt werden kann.

[0048] Die Ereignistreibereinheit **40** enthält ferner eine Hardwareressourcen-Aktualisierungseinheit **44**, damit festgelegte Hardwareressourcen, die später beschrieben werden, aktualisiert werden, und zwar aufgrund der durch die Ereignisressourcenzustands-Aktualisierungseinheit **80** aktualisierten Informationen.

[0049] Der externe Interruptgenerator **50** verwendet einen externen Zeitgeber **252**, der auf den mit dem Erweiterungsbus **290** verbundenen Schaltungskarten **250** bzw. einer von ihnen montiert ist, um periodisch in festgelegten Wiederhol-Intervallen Interruptsignale zu erzeugen. Im vorliegenden Text bezieht sich der Begriff "festgelegtes Wiederhol-Intervall" auf das kurze Zeitintervall, das für den Start des ein Ereignis verarbeitenden Programms auf Echtzeit-Basis nötig ist. Bei der vorliegenden Ausführungsform ist dieses Intervall auf 1 ms gesetzt.

[0050] Wie zuvor angemerkt, ist das im RAM **230** des Controllers **10** gespeicherte Betriebssystem **232** kein sogenanntes Echtzeit-Betriebssystem. Daher kann es die Zeit nicht in Zeitscheiben einteilen, die Zeitintervalle darstellen, welche zur Erfassung von ereignisverarbeitenden Programmen innerhalb des Systems kurz genug sind. Daher verwendet die vor-

liegende Ausführungsform einen externen Zeitgeber **252** zur Erzeugung von Interruptsignalen in festgelegten Wiederhol-Intervallen, um den gleichen Effekt zu erzielen wie die Erzeugung von 1 ms Zeitscheiben.

[0051] Wenn das Betriebssystem **232**, das eine Verarbeitung in Reaktion auf das Auftreten eines Ereignisses ausführt, eine Ereigniserwartungsanforderung ausgibt, registriert die Ereignisregistriereinheit **60** die Tatsache, daß das Anwendungsprogramm **236** auf das Auftreten eines Ereignisses warten. Bei dem Anwendungsprogramm **236** handelt es sich um irgendeines der Programme, die auf Multitasking-Basis ausgeführt werden, um die Manipulatorsteuerungs-Verarbeitung **120-1**, die Fördersteuerungs-Verarbeitung **120-2**, etc. zu steuern, wie in **Fig.** 3 gezeigt. Das Programm ist im RAM **230** gespeichert, wie in **Fig.** 2 gezeigt. Das Auftreten eines Ereignisses bezieht sich auf eine Änderung einer Ereignisressource, wie zuvor angemerkt.

[0052] Um das Anwendungsprogramm **236**, das auf Multitasking-Basis in Reaktion auf eine Änderung in einer Ereignisressource ausgeführt wird, zu starten, ist es nötig, eine Zuordnung zwischen Ereignisressourcen und den auf ihre Änderung wartenden Programmen herzustellen. In einem Zustand, wo es auf eine Änderung einer Ereignisressource warten muß, fordert daher jedes Anwendungsprogramm **236** die Ereignisregistriereinheit **60** auf, die Tatsache zu registrieren, daß das Programm auf das Auftreten eines Ereignisses wartet.

[0053] Bei Empfang dieser Anforderung, ordnet die Ereignisregistriereinheit **60** die Ereignisressource einem Ereignisobjekt zu und registriert die Tatsache, daß das Programm auf das Auftreten eines Ereignisses wartet. Der Begriff ”Ereignis” umfaßt in diesem Zusammenhang jedes der verschiedenen zuvor genannten Ereignisse. Ein ”Ereignisobjekt” bezieht sich auf die Einheit, mittels derer das System die verschiedenen Ereignisse verfolgt. Anders ausgedrückt, das System verfolgt (merkt sich) die Ereignisobjekte, und jedes Ereignisobjekt hat ein Ereignis-Handle als seine eigene ID (Identifikation).

[0054] Die Funktion, mit der das System Ereignisse und Ereignisobjekte verfolgt, kann unter Verwendung der Ereignissynchronisationsfunktion des Betriebssystems **232** implementiert werden.

[0055] Genauer gesagt, wird der Registrierprozeß wie folgt ausgeführt: die Ereignisregistriereinheit **60** erzeugt bei der Systeminitialisierung eine geeignete Anzahl von Ereignisobjekten.

[0056] Wenn von dem Anwendungsprogramm **236** eine Ereignisregistrieranforderung erzeugt wird, werden Ereignisressourcen irgendwelchen unbenutzten (nicht mit einer Ereignisressource verknüpfte) Ereignisobjekten zugewiesen. Wenn eine Ereignisregistrieranforderung von einem anderen Anwendungsprogramm **236** bezüglich einer bereits registrierten Ereignisressource erzeugt wird, wird das zuvor mit der Ereignisressource verknüpfte Ereignisobjekt verwendet, statt daß ein neues Ereignisobjekt zugewiesen wird. In diesem Zusammenhang bedeutet der Begriff ”verwenden” daß, wenn ein Ereignis-Handle an ein Anwendungsprogramm zurückgegeben wird, wie später erläutert, dieses Handle für das Ereignisobjekt dem anderen Anwendungsprogramm zurückgegeben wird. Der Begriff ”verwenden” umfaßt auch die Erzeugung eines neuen Ereignisobjekts, wenn dem System die zuvor erzeugten Ereignisobjekte ausgehen.

[0057] Wie später beschrieben, registriert die Ereignisregistriereinheit das Ereignisobjekt zur Verwendung beim Dummy-Booten (ein Aufruf des Programms, ohne daß diese im Moment wirklich benötigt wird), um jegliches Auslagern (Swappen) des Anwendungsprogramms **236**, das auf das Auftreten eines Ereignisses wartet, zu verhindern. Zu diesem Zweck verwendet die Ereignisregistriereinheit ein für jedes Anwendungsprogramm unabhängig erzeugtes Ereignisobjekt. Die Ereignisregistriereinheit weist dieses Ereignisobjekt jedem Anwendungsprogramm **236** zu, das eine Ereigniserwartungsanforderung erzeugt hat, und startet das Programm auf einer Dummy-Basis.

[0058] Nach Abschluß des Registrierungsprozesses gibt die Ereignisregistriereinheit **60** das Ereignis-Handle, bei dem es sich um die Identifikation (ID) für das Ereignisobjekt handelt, an das Anwendungsprogramm **236** zurück, das eine Registrierung angefordert hat. Als Ergebnis empfängt das Anwendungsprogramm **236** sowohl das Ereignis-Handle für ein mit einer Ereignisressource verbundenes Ereignisobjekt als auch das Ereignis-Handle für ein Ereignisobjekt für einen Dummy-Start.

[0059] Anschließend fordert das Anwendungsprogramm **236** das Betriebssystem **232** auf, es zu starten, wenn in dem durch das Ereignis-Handle bezeichneten Ereignisobjekt eine Änderung auftritt. Dies stellt sicher, daß die Programme in einem Bereitschaftszustand sind, bis eine Änderung in der als Ereignisobjekt registrierten Ereignisressource auftritt oder bis ein Dummy-Startprozeß ausgeführt wird, um das Auslagern zu verhindern.

[0060] Die Korrespondenz zwischen einer Ereignisressource und einem Anwendungsprogramm wird erstellt, wenn ein Ereignis-Handle an das Anwendungsprogramm zurückgegeben wird. Dieser Korrespondenzerstellungsprozeß verläßt sich auf die Benutzung der Ereignissynchronisationsfunktion, bei der es sich um ein Standardmerkmal des Betriebssystems handelt. Genauer gesagt, setzt die Ereignissynchronisationsfunktion das Ereignisobjekt in einen si-

gnalisierten Zustand, so daß die Tasks für alle Programme, die das entsprechende Ereignis-Handle haben, gestartet werden können.

[0061] Fig. 4 zeigt ein spezielles Abbild des Ereignisregistrierprozesses. Wie zuvor erwähnt, erzeugt die Ereignisregistriereinheit **60** während der Systeminitialisierung eine geeignete Anzahl von ereignisidentifizierenden Ereignisobjekten (mit entsprechenden Ereignis-Handlen EH1, EH2, EH3, ...) für Anwendungsprogramme, um Ereignisse zu identifizieren, und Dummy-Start-Ereignisobjekte (mit entsprechenden Ereignis-Handlen DH1, DH2, ...) für jedes Anwendungsprogramm (1).

[0062] Wenn während der Ausführung eines mit einem Anwendungsprogramm AP2 verbundenen Tasks eine Situation auftritt, die es erfordert, daß das Programm auf eine Änderung in einer Ereignisressource ER2 wartet, gibt das Anwendungsprogramm AP2 eine Registrieranforderung an die Ereignisregistriereinheit **60** (2). In diesem Prozeß übergibt das Programm die Ereignisressource ER2 als ein Argument an die Ereignisregistriereinheit **60**. Da ein unbenutztes Ereignisobjekt, in diesem Fall ein Ereignisobjekt (mit dem Ereignis-Handle EH1) der Ereignisressource ER1 zugewiesen ist (3), weist die Ereignisregistriereinheit **60** ein anderes Ereignisobjekt (mit dem Ereignis-Handle EH2) der Ereignisressource ER2 zu (4), und gibt das Ereignis-Handle EH2 des Ereignisobjekts und das Ereignis-Handle DH2 des Ereignisobjekts für die Dummy-Verarbeitung an das Anwendungsprogramm AP2 zurück (5). Das Anwendungsprogramm AP2 fordert das Betriebssystem **232** auf, solange zu warten, bis das Ereignis-Handle EH2 und das durch DH2 gekennzeichnete Ereignisobjekt, die beide von der Ereignisregistriereinheit **60** zurückgegeben wurden, den signalisierten Zustand annehmen (6).

[0063] Beim Dummy-Start für den Auslagerungsverhinderungsprozeß wird jedoch nur die Dummy-Verarbeitung ausgeführt, damit das Programm nicht ausgelagert wird. Daher ist die Registrierung und Ausführung des Dummy-Startprozesses für die Verwendung transparent, wenn der Benutzer ein Anwendungsprogramm erstellt.

[0064] Es können auch mehrere Ereignisressourcen pro Anwendungsprogramm gleichzeitig registriert werden. Wenn ein Anwendungsprogramm beispielsweise auf Änderungen in mehreren Ereignisressourcen (ER1, ER3, ER4) wartet, können Registrieranforderungen an die Ereignisregistriereinheit **60** unter Verwendung dieser Ereignisressourcen als Argumente ausgegeben werden.

[0065] Da ein Ereignisobjekt EH1 bereits mit der Ereignisressource ER1 verbunden ist, werden dieser Ereignisressource keine neuen Ereignisobjekte zugewiesen. Ein unbenutztes Ereignisobjekt (mit dem Ereignis-Handle EH3) wird der Ereignisressource ER3 zugewiesen. Wenn keine Ereignisobjekte vorhanden sind, die der Ereignisressource ER4 zugewiesen werden können, wird ein neues Ereignisobjekt (mit dem Ereignis-Handle EH4) erzeugt und zugeordnet.

[0066] Die Ereignis-Handle EH1, EH3 und EH4 für die zugewiesenen Ereignisobjekte sowie das Ereignis-Handle DH für eine Dummy-Ereignisobjekt werden an das Anwendungsprogramm zurückgegeben.

[0067] Die Ereignisressourcenzustands-Speichereinheit **70** ist in einem Speicherbereich für gemeinsamen Zugriff vorgesehen, der von den Tasks abgefragt und aktualisiert werden kann. Die Ereignisressourcenzustands-Speichereinheit hält eine Ereignisressourcentabelle, in der die Zustände der Ereignisressourcen gespeichert sind.

[0068] Die Ereignisressourcentabelle ist, wie zuvor erwähnt, so konfiguriert, daß sie die Hardwareressourcenzustände für die System-I/O-Einheiten **262** und die Schaltungskarten **250** hält, die in den Manipulatoren **270** und der Treiberbox **260** aufgetreten sind, sowie die Ausgangsdatengruppe, die es den Programmen **232**, **234** und **236** ermöglicht, eine Synchronisation und Kommunikation unter den Tasks auszuführen.

[0069] Die Zustände von Ereignisressourcen, die in der Ereignisressourcentabelle gespeichert sind, werden von der Ereignistreibereinheit **40** in regelmäßigen Zeitintervallen und von der Ereignisressourcenzustands-Aktualisierungseinheit **80** nach Bedarf aktualisiert, wie später beschrieben.

[0070] Wenn, hauptsächlich von dem Anwendungsprogramm **236**, angefordert, aktualisiert die Ereignisressourcenzustands-Aktualisierungseinheit **80** die Zustände der Ausgangsports und jene der Ausgangsdatengruppe, die zur Synchronisation und Kommunikation unter den Tasks verwendet wird.

[0071] Die vorgenannte Anforderung wird in der Weise ausgegeben, daß ein Anwendungsprogramm **236** während seiner Ausführung eine Funktion aufruft, die die Ereignisressourcenzustands-Aktualisierungseinheit **80** startet. Zu diesem Zweck übergibt das Anwendungsprogramm die Ausgangsdatengruppe als Argument.

[0072] Normalerweise übergeben die Anwendungsprogramme **236** einander und empfangen voneinander die Ausgangsdatengruppe. Bei der vorliegenden Ausführungsform wird jedoch, wenn eine Ausgangsdatengruppe auftritt, die zwischen Anwendungsprogrammen ausgetauscht werden muß, eine Anforderung für einen Ausgangsdatengruppen-Austausch an

die Ereignisressourcenzustands-Aktualisierungseinheit **80** mittels der vorerwähnten Funktion ausgegeben. Bei Empfang der Anforderung aktualisiert die Ereignisressourcenzustands-Aktualisierungseinheit **80** den Zustand der betroffenen Ereignisressource in der Ereignisressourcenzustands-Speichereinheit **70**.

[0073] Daher aktualisieren Anwendungsprogramme **236** nicht die Ausgangsports **242**, die beim tatsächlichen Auftreten einer Ausgangsdatengruppe aktualisiert werden würden. Diese Ausgangsports sind so konfiguriert, daß sie auf einer Gemeinschaftsbasis von der Hardwareressourcen-Aktualisierungseinheit **44** aktualisiert werden, wie später beschrieben.

[0074] Die Ereignistreibereinheit **40** führt eine Ereignistreiberverarbeitung synchron mit den Interruptsignalen aus, die von dem externen Interruptgenerator **50** erzeugt werden. Unter "Ereignistreiberverarbeitung" ist der Prozeß der regelmäßigen Erfassung von Ereignissen in den Wiederhol-Intervallen, die zur Ausführung einer Echtzeit-Verarbeitung nötig sind, und die Benachrichtigung der Task-Wechseleinheit **30** darüber zu verstehen, daß die Steuerung zu dem Task umgeschaltet werden muß, in welchem die dem Ereignis entsprechende Verarbeitung ausgeführt wird.

[0075] Die Ereignistreiberverarbeitung durch die Ereignistreibereinheit **40** tritt auf, wenn die CPU **210** sowohl das im RAM **230** gespeicherte Steuerprogramm **234** als auch die Ereignissynchronisationsfunktion des Betriebssystems **232** ausführt. Das Steuerprogramm **234**, das die Funktion der Ereignistreibereinheit **40** implementiert, ist im System resident. Der Task, der die Verarbeitung ausführt, wird synchron mit den Interruptsignalen getrieben, die von dem externen Interruptgenerator **50** erzeugt werden. Wie später erläutert, ist die Ereignistreibereinheit in einer solchen Weise konfiguriert, daß, wenn ein einem gegebenen Ereignis entsprechendes Programm gestartet wird, die Ereignissynchronisationsfunktion des Betriebssystems **232** benutzt wird. Zur Erfassung von Ereignissen überwacht die Ereignistreibereinheit **40** die Ereignisressourcen wie folgt: obwohl die Ereignistreibereinheit verschiedene Überwachungsmethoden zur Erfassung von Ereignisressourcenänderungen für verschiedene Einrichtungen verwendet, definiert man für den Eingangsport **242** im voraus die Portadresse und die Adressengröße, die zu überwachen sind. Sobald dies erfolgt ist, fragt die Ereignistreibereinheit synchron mit dem Interrupt alle definierten Eingangsports ab und speichert den momentanen Zustand des Eingangsports **242** im zugehörigen Bereich der Ereignisressourcentabelle, die in der Ereignisressourcenzustands-Speichereinheit **70** gespeichert ist. Zur Erfassung eines Ereignisses vergleicht die Ereignistreibereinheit den Zustand des abgefragten Eingangsports **242** mit dem vorigen, in der Ereignisressourcentabelle gespeicherten Zustand.

[0076] Was die anderen Einrichtungen, etwa die Manipulatoren **270**, die in der Treiberbox **260** montierten System-I/O-Einheiten und die speziellen Schaltungskarten **250** angeht, die mit dem Erweiterungsbus **290** verbunden sind, so überwacht die Ereignistreibereinheit diese durch Aufruf der gesondert vorgesehenen Einrichtungstreiber. Sie speichert die momentanen Zustände der Robotersystemeinheit **270**, der System-I/O-Einheiten und der speziellen Schaltungskarten **250** in dem betroffenen Bereich in der Ereignisressourcentabelle, die in der Ereignisressourcenzustands-Speichereinheit **70** gespeichert ist. Zur Erfassung eines Ereignisses vergleicht die Ereignistreibereinheit die Ergebnisse der durch Aufruf der Einrichtungstreiber durchgeführten Überwachung mit dem in der Ereignisressourcentabelle gespeicherten vorherigen Zustand. Was die Ausgangsdatengruppe für verschiedene Anwendungsprogramme **236** angeht, so wird normalerweise das Auftreten eines Anwendungsprogramms in dem Ausgangsport **244** reflektiert.

[0077] Wenn dann unter den Ereignisressourcenänderungen, die festgestellt wurden, eine in der Ereignisregistriereinheit **60** registrierte Ereignisressource ist, weist die Ereignistreibereinheit **40** die Task-Wechseleinheit **30** an, den mit dem Anwendungsprogramm **236** verbundenen Task zu starten, indem das Ereignisobjekt in den signalisierten Zustand versetzt wird.

[0078] Wenn zwei oder mehr Anwendungsprogramme auf eine Änderung in derselben Ereignisressource warten, oder wenn gleichzeitig Änderungen bei mehreren Ereignisressourcen auftreten, nehmen das eine oder die mehreren entsprechenden Ereignisobjekte den signalisierten Zustand an. Danach schaltet eine Funktion im Betriebssystem die Verarbeitung der Anwendungsprogramme in einem Umlaufverfahren um.

[0079] Da, wie oben erwähnt, das Anwendungsprogramm **236** den Ausgangsport **244** nicht tatsächlich aktualisiert, wenn die Ereignisressourcentabelle aktualisiert wird, muß die Ereignisressourcentabelle aktualisiert werden. Wenn daher in einem Anwendungsprogramm eine Ausgangsdatengruppe erzeugt wird, aktualisiert die Hardwareressourcen-Aktualisierungseinheit **44** den tatsächlichen, entsprechenden Ausgangsport **244**.

[0080] Das System ist so konfiguriert, daß die Größe der Ausgangsdatengruppe und die Ausgangsadresse des tatsächlichen Ausgangsports **244** für die Anwendungsprogramme **236**, die für den Aktualisierungsprozeß nötig sind, im voraus registriert werden können.

[0081] Die Task-Wechseleinheit **30** schaltet in vorgeschriebenen Zeitintervallen zu irgendeinem Task

um, um die Verarbeitung auf einer Multitasking-Basis auszuführen. Die Task-Wechseleinheit schaltet auch zu einem Task um, wenn entweder die Ereignistreibereinheit **40** oder die Auslagerungsverhinderungseinheit **90** den Start des Tasks für das entsprechende Anwendungsprogramm **236** dadurch anzeigt, daß ein Ereignisobjekt in den signalisierten Zustand versetzt wird.

[0082] Der Controller **10** führt den Taskwechsel unter Verwendung der Standard-Zeitscheibenfunktion aus, die in dem Betriebssystem **232** verfügbar ist, um die Verarbeitung auf einer Multitasking-Basis auszuführen.

[0083] Das Umschalten zu dem letzteren Task wird dadurch implementiert, daß der Task für das Anwendungsprogramm **236** gestartet wird, das das Ereignis-Handle besitzt, welches dem Ereignisobjekt entspricht, das in einen signalisierten Zustand versetzt wurde. Dies wird bewirkt, wenn die CPU **210** die Ereignissynchronisationsfunktion des Betriebssystems **232** ausführt, das in dem RAM **230** gespeichert ist.

[0084] Wie oben erwähnt werden, wenn mehrere Anwendungsprogramme auf eine Änderung in derselben Ereignisressource warten, oder wenn Änderungen in mehreren Ereignisressourcen gleichzeitig auftreten, das entsprechende oder die mehreren entsprechenden Ereignisobjekte in den signalisierten Zustand versetzt. Dies ermöglicht es der Task-Wechseleinheit **30**, die Anwendungsprogramme, die die entsprechenden Ereignis-Handle besitzen, auf einer Umlaufbasis (round-robin) zu starten.

[0085] Wenn ein Ereignis registriert ist, werden das Ereignisobjekt und das Anwendungsprogramm durch die Standard-Ereignissynchronisationsfunktion miteinander verknüpft, die das Betriebssystem bietet. Daher kann man dadurch, daß ein Ereignisobjekt in den signalisierten Zustand versetzt wird, den Task für das Anwendungsprogramm **236** starten, welches das entsprechende Ereignis-Handle besitzt.

[0086] Die **Fig.** 5A und B sind Flußdiagramme, die die Prozedur zeigen, nach der das System Ereignisse auf einer Echtzeit-Basis treibt. **Fig.** 5A zeigt die Prozedur dafür, daß die Ereignistreiberverarbeitung periodisch in festgelegten Zeitintervallen durch die Ereignistreibereinheit **40** auftritt. **Fig.** 5B zeigt die Prozedur, nach der die ereignisgesteuerten Anwendungsprogramme **236** arbeiten.

[0087] Wie in **Fig.** 5B gezeigt, spezifiziert, wenn ein Task für ein Anwendungsprogramm **236** gestartet wird und ein Ereigniserwartungszustand während der Ausführung der Verarbeitungsstufe 1 erzeugt (Schritt 110), das Anwendungsprogramm **236** die Ereignisressource gegenüber der Ereignisregistriereinheit, um die Registrierung des Ereigniserwartungszustands anzufordern (Schritt 120), und als Ergebnis tritt das Anwendungsprogramm in den Ereigniserwartungszustand ein (Schritt 130). Demgemäß wird die Ereignisressource, deren Änderung von dem Anwendungsprogramm **236** erwartet wird, in Korrespondenz mit einem Ereignisobjekt registriert. Das Anwendungsprogramm **236** empfängt das Ereignis-Handle entsprechend dem Ereignisobjekt und wartet, bis das Ereignisobjekt den signalisierten Zustand annimmt.

[0088] Andererseits überwacht, wie in **Fig.** 5A gezeigt, die Ereignistreibereinheit **40** die verschiedenen Hardwareressourcen, wie oben beschrieben wurde (Schritt 10). Wenn es eine Änderung in der Ausgangsdatengruppe infolge des Anwendungsprogramms (AP) gibt, bei der es sich um eine als ein Ereignisobjekt in der oben erwähnten Ereignisressourcentabelle registrierte Ereignisressource handelt (Schritt 20), aktualisiert die Hardwareressourcen-Aktualisierungseinheit **44** der Ereignistreibereinheit 40 tatsächlich den Ausgangsport **244**, bei dem es sich um die entsprechende Hardwareressource handelt (Schritt 30).

[0089] Wenn es eine Änderung in einer registrierten Ereignisressource gibt (Schritt 40), wird das der Ereignisressource entsprechende Ereignisobjekt, das durch die Ereignisregistriereinheit **60** registriert wurde, in den signalisierten Zustand versetzt (Schritt 50). Wenn das Ereignisobjekt den signalisierten Zustand angenommen hat, startet das Betriebssystem **232**, was mit der Task-Wechseleinheit **30** äquivalent ist, den Task, in welchem das Anwendungsprogramm, das das entsprechende Ereignis-Handle besitzt, ausgeführt wird (Schritt 60).

[0090] Wenn dies auftritt, beginnt das Anwendungsprogramm **236**, das auf das Auftreten eines durch das Ereignis-Handle gekennzeichneten Ereignisses gewartet hat, die Ausführung der Verarbeitungsstufe 2 (Schritt 140). Dies ist in **Fig.** 5B gezeigt.

[0091] Wenn ein Anwendungsprogramm, dessen Verarbeitung in Echtzeit ausgeführt werden muß, wartet, startet die Auslagerungsverhinderungseinheit **90** das Anwendungsprogramm auf periodischer Basis, damit dieses Anwendungsprogramm nicht ausgelagert wird.

[0092] Anwendungsprogramme, die eine Echtzeit-Verarbeitung erfordern, sind solche die auf das Eintreten eines Ereignisses warten. Wenn diese Anwendungsprogramme ausgelagert werden und sie angesteuert werden, nachdem ein Ereignis aufgetreten ist, nimmt das Neuladen des erforderlichen Programms mehrere zehn oder sogar mehrere hundert Millisekunden in Anspruch, was die Fähigkeit des Systems reduziert, auf Ereignisse in Echtzeit zu antworten.

[0093] Daher weist die Auslagerungsverhinderungseinheit **90** die Task-Wechseleinheit **30** an, das Anwendungsprogramm zu starten, das mit einem Dummy-Aktivierungs-Ereignisobjekt in der Ereignisregistriereinheit **60** verknüpft ist, und zwar in festgelegten Zeitintervallen. Zu diesem Zweck werden festgelegte Zeitintervalle unter Benutzung des Systemzeitgebers erzeugt. Die Auslagerungsverhinderungseinheit liefert an die Task-Wechseleinheit **30** die nachfolgend beschriebenen Befehle.

[0094] Wenn ein Anwendungsprogramm die Ereignisregistriereinheit **60** auffordert, die Tatsache zu registrieren, daß es auf die Änderung in einer Ereignisressource wartet, wird das Ereignis-Handle für das Dummy-Verarbeitungs-Ereignisobjekt an das Anwendungsprogramm zurückgegeben, wie oben beschrieben. Die Auslagerungsverhinderungseinheit **90** versetzt das Dummy-Verarbeitungs-Ereignisobjekt periodisch in den signalisierten Zustand, um die Task-Wechseleinheit **30** anzuweisen, den Task für das Anwendungsprogramm, das das entsprechende Ereignis-Handle besitzt, zu starten.

[0095] Das Anwendungsprogramm kann anhand der Art des Ereignisobjekts feststellen, ob ein bestimmter Task durch die Auslagerungsverhinderungseinheit **90** oder durch die Ereignistreibereinheit **40** ausgelöst wurde. Wenn der Task von der Auslagerungsverhinderungseinheit **90** ausgelöst wurde, führt das Anwendungsprogramm eine Dummy-Verarbeitung durch und wartet erneut auf das Auftreten eines Ereignisses. Diese Art Verarbeitung, die von dem Anwendungsprogramm auszuführen ist, sollte von dem Steuerprogramm als eine Funktion vorgesehen werden, die von dem Anwendungsprogramm aufgerufen wird, wenn das Anwendungsprogramm die Tatsache registriert, daß es auf das Auftreten eines Ereignisses wartet. Auf diese Weise braucht ein Benutzer bei der Erstellung eines Anwendungsprogramms nur die Funktion zu benutzen, ohne sich selbst mit der Problematik des involvierten Mechanismus zu befassen.

[0096] Diese Merkmale implementieren die folgenden Charakteristiken, die besser sind als das was in herkömmlichen Produkten zur Verfügung steht:
Die Zeitdauer, die vom Auftreten eines Ereignisses bis zum Start des entsprechenden Programms erforderlich ist, ist kurz, und ihre Schwankung ist gering. Dies führt zu einem hohen Grad an Genauigkeitswiederholbarkeit bei wiederholten Roboteroperationen oder bei den Operationen anderer Steuerungsanlagen, die von einem Benutzer codiert werden. Man nehme an, daß ein Sensor die Position eines Objekts auf einem Förderer feststellt und ein Roboter seine Operation auf der Basis des Sensorsignals beginnt. Wenn eine deutliche Schwankung von dem Zeitpunkt einer Sensoreingabe bis zum Beginn der Roboteroperation vorhanden ist, erfordert es nicht nur eine Menge Arbeit, das gesamte System zu justieren, vielmehr kann in einigen Fällen der Roboter auch versagen, das auf dem Förderer montierte Objekt richtig zu handhaben.

[0097] In einem Echtzeitsystem mit einer begrenzten Verarbeitungskapazität kann eine Zunahme der Anzahl von Tasks, die gleichzeitig ausgeführt werden, zu einer raschen Zunahme der Zeitdauer vom Auftreten eines Ereignisses bis zu dem Zeitpunkt führen, zu dem die entsprechende Verarbeitung ausgeführt wird, was zu einem deutlichen Abfall des Reaktionsvermögens des Systems führt. Im Gegensatz dazu kann ein auf der vorliegenden Erfindung beruhendes System wegen seines überragenden Echtzeit-Verarbeitungsvermögens eine große Anzahl gleichzeitig ausgeführter Tasks unterstützen. Bei der vorliegenden Ausführungsform können 32 Tasks gleichzeitig in der Robotersprache ausgeführt werden, in der Benutzer Programme schreiben.

[0098] Was Ereignisse anbetrifft, die in dem System registriert werden können, so können nicht nur einfache gewöhnliche Ereignisse, wie das Ein- und Ausschalten eines I/O-Signals, sondern auch komplexe Bedingungen als Ereignis registriert werden, etwa daß ein Roboter eine bestimmte Orientierung annimmt, und zwar durch Verwendung einer Ereignisressourcentabelle, einer einen bestimmten Wert annehmenden Variablen im Programm, oder einer Kombination dieser Bedingungen in Form logischer Ausdrücke.

[0099] Durch Implementieren eines Robotercontrollers durch Kombinieren eines PCs mit Windows 95®, wie bei der vorliegenden Ausführungsform, ist es möglich, das große Angebot an preiswerten Erweiterungskarten (Netzwerkverbindungskarten, Instrumentenverbindungskarten und allgemeinen I/O-Karten) zu nutzen, die im Handel zur Verfügung stehen.

[0100] Ferner können Anwendungsprogramme unter Verwendung von im Handel zur Verfügung stehenden Programmiersprachen (Visual C++, Visual Basic, usw.) entwickelt werden, zusätzlich zu speziellen Robotersprachen. Das Softwarebetriebsverfahren, das die vorliegende Erfindung implementiert, basiert auf Windows 95® Betriebsverfahren, so daß alle Benutzer, die mit anderen Anwendungen (Textverarbeitung, Tabellenkalkulation, etc.), die unter Windows 95® laufen, vertraut sind, leicht lernen können, wie die Roboterbetriebsprozeduren gemäß der vorliegenden Erfindung arbeiten.

[0101] Obwohl in Verbindung mit der vorliegenden Ausführungsform die Verwendung von Windows 95® als ein Beispiel eines allgemeinen PC-Betriebssystems erläutert wurde, können auch andere typische PC-Betriebssysteme verwendet werden.

[0102] Die vorliegende Erfindung ist keinesfalls auf die bezüglich der obigen Ausführungsformen gegebene Erläuterung beschränkt. Vielmehr sind modifizierte Ausführungsformen möglich.

[0103] Im folgenden wird eine Ausführungsform eines Robotercontrollers **10** beschrieben, der ein allgemeines PC-Betriebssystem einsetzt, das in der Lage ist, eine Echtzeit-Verarbeitung auszuführen.

[0104] Ein Betriebssystem wird hier dann als zur Durchführung einer Echtzeit-Verarbeitung angesehen, wenn der Taskwechsel durch das Betriebssystem schnell ausgeführt werden kann. Diese Art von Betriebssystem kann mittels eines Systemzeitgebers ausreichend kleine Zeitintervalle erzeugen, und eine Zeitscheibenbildung kann in ausreichend kurzen Zeitintervallen ausgeführt, um eine Echtzeit-Steuerung zuzulassen.

[0105] Im ersteren Fall kann die Erzeugung von Interrupts durch einen externen Zeitgeber in dem externen Interruptgenerator **50** der vorgenannten Ausführungsform durch einen Systemzeitgeber ersetzt werden, und all die anderen Merkmale können in gleicher Weise wie bei der vorerwähnten Ausführungsform implementiert werden.

[0106] Im letzteren Fall kann die Erzeugung von Interrupts durch einen externen Zeitgeber in dem externen Interruptgenerator **50** der vorgenannten Ausführungsform ersetzt werden durch die Verwendung einer Zeitscheibenbildung, und alle anderen Merkmale können in gleicher Weise wie bei der vorgenannten Ausführungsform implementiert werden.

[0107] In beiden Fällen können Zeitintervalle, die kurz genug sind, um eine Echtzeit-Steuerung in der internen Arbeit des Systems zu erlauben, eingestellt werden. Folglich bedarf es nicht des externen Zeitgebers, der in der Hardwarekonfiguration von **Fig.** 2 gezeigt ist.

[0108] Der letztere Fall sei nachfolgend als Beispiel erläutert.

[0109] **Fig.** 6 ist ein Funktionsblockdiagramm eines Robotercontrollers **10**, der ein Betriebssystem mit Echtzeit-Verarbeitungsvermögen einsetzt. Diese Figur unterscheidet sich von **Fig.** 1 darin, daß die Task-Wechseleinheit **30** eine Zeitscheibeneinheit **32** anstelle des externen Interruptgenerators **50** enthält. Allen anderen Komponenten sind dieselben Namen und dieselben Bezugszahlen wie in **Fig.** 1 zugeordnet. Eine Erläuterung der Funktionen, die ähnlich jenen in **Fig.** 1 sind, unterbleibt.

[0110] Die Zeitscheibeneinheit **32** unterteilt die Zeit in Zeitintervalle, die kurz genug sind, um Echtzeitsteuerungen zu erlauben, beispielsweise je eine Millisekunde. Sie weist CPU-Zeit Tasks in einer Umlaufweise (round-robin) zu, um die Tasks zu starten. Für jede Zeitscheibe treibt die Zeitscheibeneinheit die Ereignistreibereinheit **40** vor Ausführung eines gegebenen Tasks.

[0111] Das Steuerprogramm **234**, das die Funktionen der Ereignistreibereinheit **40** implementiert, wird als ein Task ausgeführt, der einseitig auf einer Zeitscheibe pro Zeitscheiben-Basis ausgeführt wird. Da die Ereignistreiberverarbeitung in der Ereignistreibereinheit **40** innerhalb einer Millisekunde endet, wird, wenn keine Ereignisse aufgetreten sind, die normale auf Zeitscheiben-Basis beruhende Task-Aktivierung für die Restzeit in der Zeitscheibe bewirkt.

[0112] In jeglicher anderer Hinsicht kann dieses Schema die Funktionen eines Robotercontrollers in genau der gleichen Weise implementieren, wie die erste Ausführungsform.

[0113] Obwohl diese Ausführungsform die Verwendung einer Standard-Ereignissynchronisationsfunktion illustriert, die von dem System vorgesehen wird, kann jegliche Ereignissynchronisationsfunktion verwendet werden, solange sie die gleichen Wirkungen ergibt. Anders ausgedrückt, Funktionen, die von dem System oder von Systemaufrufen ausgeführt werden können, können eingesetzt werden. Alternativ können auch Steuerprogramme entwickelt werden, die dieselben Wirkungen wie die Ereignissynchronisationsfunktion bieten.

[0114] Obwohl die vorliegende Ausführungsform den Fall darstellt, bei dem das Betriebssystem **232** und das Steuerprogramm **234** für den Controller **10** im RAM **230** gespeichert sind, können sie auch in einem abnehmbaren externen Speichermedium gespeichert sein. Ebenso können sie von einem externen Speichermedium auf ein internes Speichermedium geladen werden, oder, mittels Kommunikationsverbindungen, von einer externen Einrichtung auf ein internes Speichermedium.

[0115] Weiterhin ist die vorliegende Erfindung keineswegs auf spezielle Robotertypen oder -Konfigurationen beschränkt, sondern ist auf eine weite Vielfalt von Robotercontrollern anwendbar.

[0116] Obwohl die vorliegende Ausführungsform ein Robotercontrollersystem beschreibt, kann ein Roboterfolger (robot sequencer), der von derselben Konfiguration Gebrauch macht, ebenfalls konstruiert werden. Daher liegt auch die Anwendung der vorliegenden Erfindung auf einen Roboterfolger im Rahmen der vorliegenden Erfindung.

Patentansprüche

1. Robotercontroller umfassend eine Task-Wechseleinrichtung (**30**), die mit einer preemptiven Multitasking-Funktion versehen ist, und eine Echtzeit-Steuereinrichtung (**20**), die Steuerungen dadurch ausführt, daß sie die Task-Wechseleinrichtung (**30**) anweist, Tasks zu wechseln, so daß die Verarbeitung als Reaktion auf das Auftreten eines Ereignisses in Echtzeit ausgeführt wird, wobei
der Robotercontroller ein allgemeines Betriebssystem, welches eine preemptive Multitasking-Funktion aufweist, verwendet,
die Echtzeit-Steuereinrichtung Ereignisse auf regelmäßiger Basis in Wiederhol-Intervallen erfaßt, die kürzer sind als die von dem Betriebssystem zur Verfügung gestellten,
der Robotercontroller eine Ereignistreibereinrichtung (**40**) enthält, die eine Ereignistreiberverarbeitung ausführt, welche der Task-Wechseleinrichtung einen Wechsel zu dem Task anzeigt, in welchem die mit dem Ereignis verknüpfte Verarbeitung ausgeführt wird, und
die Echtzeit-Steuereinrichtung (**20**) weiterhin eine externe Interrupt-Generatoreinrichtung (**50**) umfaßt, die unter Verwendung eines externen Zeitgebers (**252**) Interruptsignale auf regelmäßiger Basis in den Wiederhol-Intervallen, die zur Ausführung einer Echtzeit-Verarbeitung nötig sind, erzeugt, wobei die Ereignistreibereinrichtung (**40**) die Ereignistreiberverarbeitung synchron mit diesen Interruptsignalen ausführt.

2. Robotercontroller nach Anspruch 1, bei dem
die Echtzeit-Steuereinrichtung (**20**) weiterhin eine Ereignisregistriereinrichtung (**60**) umfaßt, die registriert, daß das Programm, das die mit dem Auftreten des Ereignisses verbundene Verarbeitung ausführt, auf das Auftreten des Ereignisses wartet, und
die Ereignistreibereinrichtung (**40**), wenn sie das Auftreten des von der Ereignisregistriereinrichtung (**60**) registrierten Ereignisses feststellt, die Task-Wechseleinrichtung (**30**) anweist, zu dem Task zu wechseln, der von dem Programm auszuführen ist, das auf das Auftreten des von der Ereignisregistriereinrichtung (**60**) registrierten Ereignisses gewartet hat.

3. Robotercontroller nach Anspruch 1 oder 2, bei dem die Echtzeit-Steuereinrichtung (**20**) wenigstens eines von folgendem als ein Ereignis behandelt: eine Änderung in den Hardwareressourcen, die der Roboter besitzt, oder in der Ausgangsdatengruppe, die es dem Programm, das Manipulatoraktionen oder periphere Einrichtungen steuert, ermöglicht, Synchronisation und Kommunikation auszuführen.

4. Robotercontroller nach Anspruch 3, bei dem die Echtzeit-Steuereinrichtung (**20**) umfaßt:
eine Ereignisressourcenzustands-Speichereinrichtung (**70**), die in einem gemeinsam genutzten Speicherbereich, der von mehreren Tasks abgefragt und aktualisiert werden kann, Speicherereignisressourcenzustände speichert, um wenigstens eines von folgendem zu verfolgen: eine Änderung in den Hardwareressourcen oder den Ausgangsdaten, die das Programm, das Manipulatoraktionen oder periphere Einrichtungen steuert, in die Lage versetzt, Synchronisation und Kommunikation auszuführen, und
eine Ereignisressourenzustands-Aktualisierungseinrichtung (**80**), die den Ereignisressourcenzustand, der in der Ereignisressourcenzustands-Speichereinrichtung (**70**) gespeichert ist, auf der Basis wenigstens eines von folgendem aktualisiert: den Manipulatoroperationen und der Ausgangsdatengruppe, die es dem Programm, das periphere Einrichtungen steuert, ermöglicht, Synchronisation und Kommunikation auszuführen,
und bei dem die Ereignistreibereinrichtung (**40**) eine Hardwareressourcen-Aktualisierungseinrichtung (**44**) umfaßt, die Hardwareressourcen auf der Basis eines Ereignisressourcenzustands aktualisiert, der von der Ereignisressourcenzustands-Aktualisierungseinrichtung (**80**) aktualisiert wurde.

5. Robotercontroller nach Anspruch 2 oder einem der Ansprüche 3 und 4 in ihrer Abhängigkeit von Anspruch 2, bei dem der Robotercontroller eine Auslagerungsverhinderungseinrichtung (**90**) aufweist, die das Auslagern des Programms dadurch verhindert, daß die Task-Wechseleinrichtung (**30**) regelmäßig in bestimmten Zeitintervallen angewiesen wird, das Programm zu starten, das die Tatsache registriert hat, daß es auf das Auftreten eines Ereignisses wartet.

6. Echtzeit-Steuerverfahren für eine mit dem Auftreten eines Ereignisses in einem Robotercontroller verbundene Verarbeitung, das ein allgemeines Betriebssystem einsetzt, welches eine preemptive Multitasking-Funktion aufweist, bei dem
Ereignisse auf regelmäßiger Basis in Wiederhol-Intervallen erfaßt werden, die kürzer sind als die von dem Betriebssystem zur Verfügung gestellten, umfassend
einen Interrupterzeugungsschritt zur Erzeugung eines externen Interruptsignals unter Verwendung eines externen Zeitgebers auf regelmäßiger Basis in den Wiederhol-Intervallen, die zur Ausführung einer Echtzeit-Verarbeitung nötig sind, und
einen Ereignistreiberschritt, bei dem Ereignisse synchron mit den externen Interruptsignalen erfaßt werden und eine Ereignistreiberverarbeitung ausgeführt wird, bei der das allgemeine Betriebssystem angewiesen wird, auf den Task umzuschalten, in welchem die mit dem Ereignis verknüpfte Verarbeitung ausgeführt wird.

7. Verfahren nach Anspruch 6, bei dem das Programm, das die mit dem Auftreten des Ereignisses verknüpfte Verarbeitung ausführt, einen Ereignisregistrierschritt umfaßt, bei dem die Tatsache regis-

triert wird, daß das Programm auf das Auftreten des Ereignisses wartet, wobei der Ereignistreiberschritt, wenn das Auftreten eines in dem Ereignisregistrierschritt registrierten Ereignisses festgestellt wird, das allgemeine Betriebssystem anweist, zu dem Task zu wechseln, in welchem das Programm ausgeführt wird, das auf das Auftreten des Ereignisses wartet, das in dem Ereignisregistrierschritt registriert wurde.

8. Verfahren nach Anspruch 6 oder 7, bei dem wenigstens eines von folgendem als ein Ereignis behandelt wird: eine Änderung der Hardwareressourcen, die der Roboter besitzt, oder der Ausgangsdaten, die es dem Programm, das Manipulatoraktionen oder periphere Einrichtungen steuert, ermöglicht, Synchronisation und Kommunikation auszuführen.

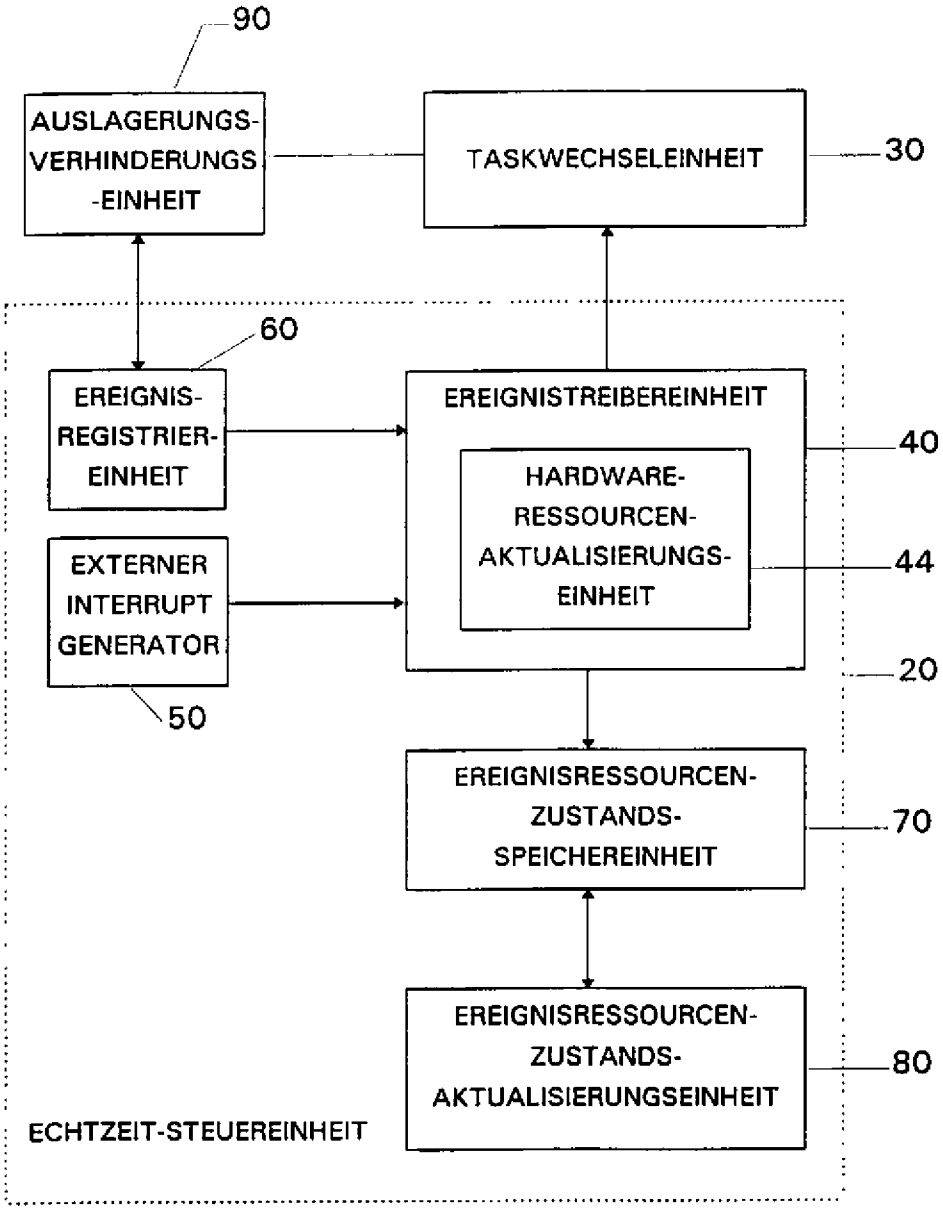
9. Verfahren nach Anspruch 8, ferner umfassend
einen Ereignisressourcenzustands-Speicherschritt, der in einem gemeinsam genutzten Speicherbereich, der von mehreren Tasks abgerufen und aktualisiert werden kann, Speicherereignisressourcenzustände speichert, um wenigstens eines von folgendem zu verfolgen: eine Änderung in den Hardwareressourcen oder den Ausgangsdaten, die es dem Programm, das Manipulatoraktionen oder periphere Einrichtungen steuert, ermöglicht, Synchronisation und Kommunikation auszuführen, und
einen Ereignisressourcenzustands-Aktualisierungsschritt, bei dem eine Aktualisierung auf der Basis wenigstens eines von folgendem ausgeführt wird: den Manipulatoroperationen und der Ausgangsdatengruppe, die es dem Programm, das periphere Einrichtungen steuert, ermöglicht, Synchronisation und Kommunikation auszuführen,
wobei der Ereignistreiberschritt einen Hardwareressourcen-Aktualisierungsschritt umfaßt, bei dem Hardwareressourcen auf der Basis der Ereignisressourceninformation aktualisiert werden, die von dem Ereignisressourcenzustands-Aktualisierungsschritt aktualisiert wurden.

10. Verfahren nach Anspruch 7 oder einem der Ansprüche 8 und 9 in ihrer Abhängigkeit von Anspruch 7, bei dem das Steuerverfahren einen Auslagerungsverhinderungsschritt umfaßt, der das Auslagern des Programms dadurch verhindert, daß das allgemeine Betriebssystem angewiesen wird, das Programm regelmäßig in bestimmten Zeitintervallen zu starten, das die Tatsache registriert hat, daß es auf das Auftreten eines Ereignisses wartet.

Es folgen 7 Seiten Zeichnungen

Anhängende Zeichnungen

FIG. 1



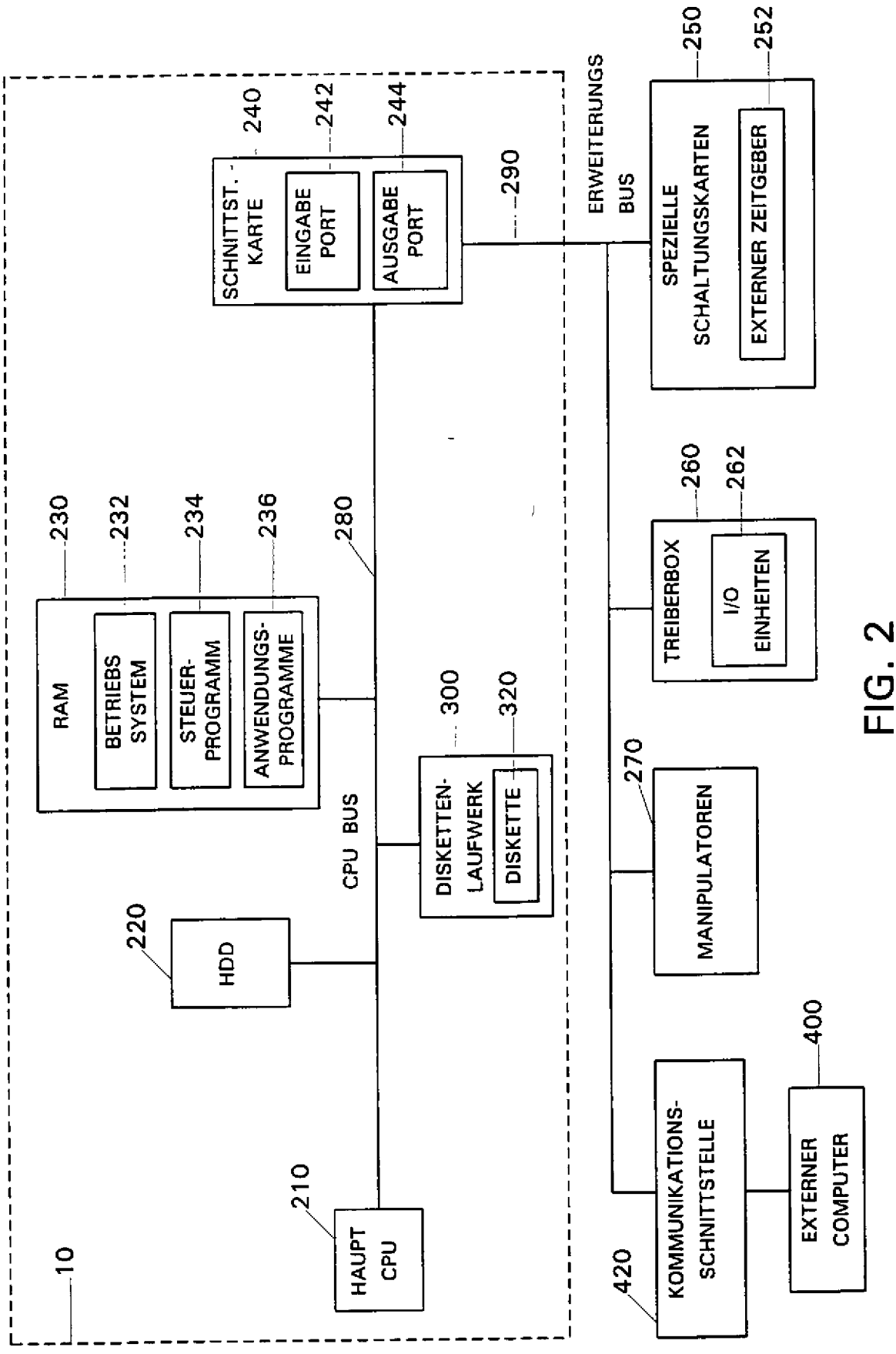

10
HAUPT CPU
210
HDD
220
RAM
230
BETRIEBS SYSTEM
232
STEUER-PROGRAMM
234
ANWENDUNGS-PROGRAMME
236
CPU BUS
280
DISKETTEN-LAUFWERK
300
DISKETTE
320
SCHNITTST.-KARTE
240
EINGABE PORT
242
AUSGABE PORT
244
290
ERWEITERUNGS BUS
SPEZIELLE SCHALTUNGSKARTEN
250
EXTERNER ZEITGEBER
252
TREIBERBOX
260
I/O EINHEITEN
262
MANIPULATOREN
270
KOMMUNIKATIONS-SCHNITTSTELLE
420
EXTERNER COMPUTER
400

FIG. 2

Fig. 3

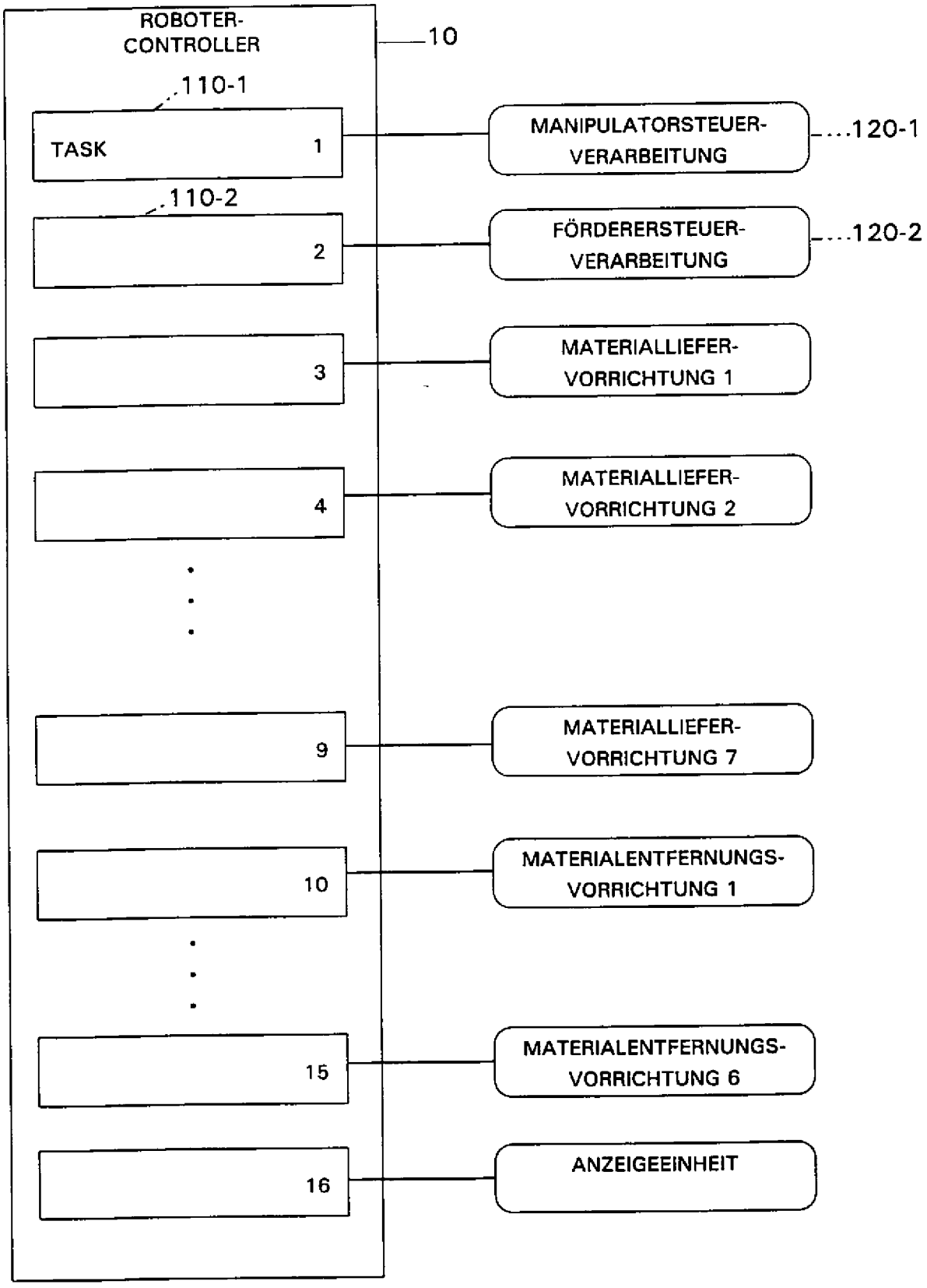

ROBOTER-
CONTROLLER
10
110-1
TASK 1
110-2
2
3
4
9
10
15
16
MANIPULATORSTEUER-
VERARBEITUNG
120-1
FÖRDERERSTEUER-
VERARBEITUNG
120-2
MATERIALLIEFER-
VORRICHTUNG 1
MATERIALLIEFER-
VORRICHTUNG 2
MATERIALLIEFER-
VORRICHTUNG 7
MATERIALENTFERNUNGS-
VORRICHTUNG 1
MATERIALENTFERNUNGS-
VORRICHTUNG 6
ANZEIGEEINHEIT

Fig. 4

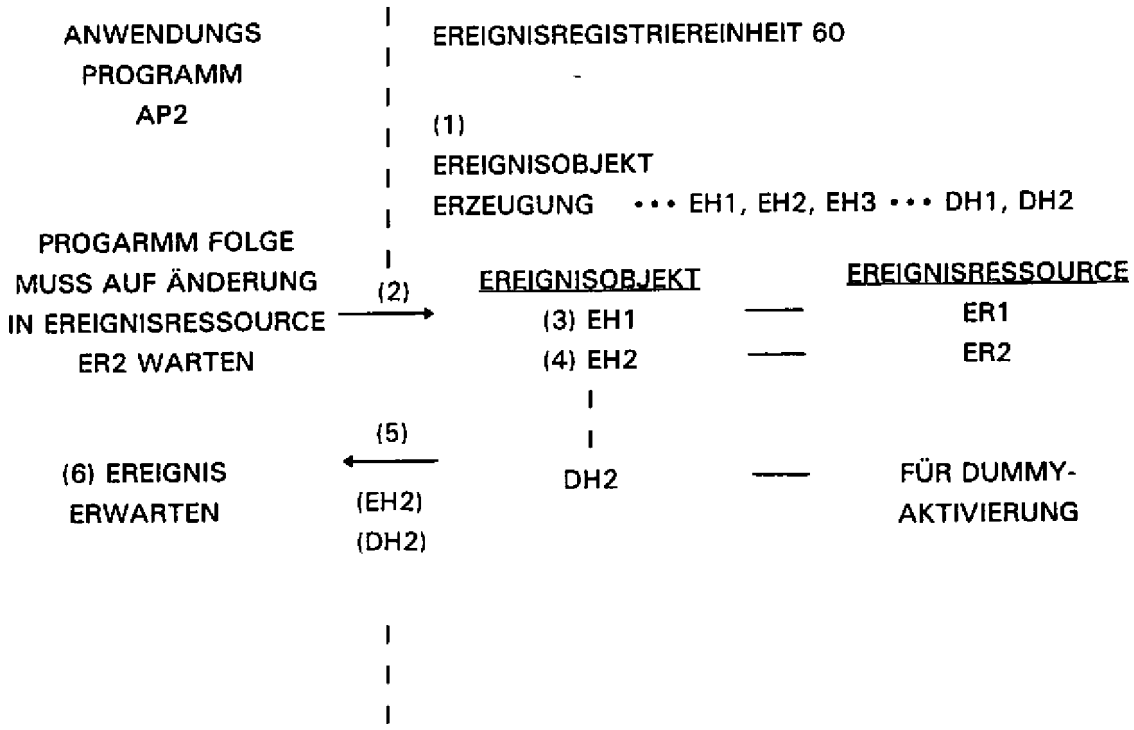

ANWENDUNGS PROGRAMM AP2
EREIGNISREGISTRIEREINHEIT 60
(1)
EREIGNISOBJEKT ERZEUGUNG ••• EH1, EH2, EH3 ••• DH1, DH2
PROGARMM FOLGE MUSS AUF ÄNDERUNG IN EREIGNISRESSOURCE ER2 WARTEN
(2)
EREIGNISOBJEKT
(3) EH1
(4) EH2
DH2
EREIGNISRESSOURCE
ER1
ER2
FÜR DUMMY-AKTIVIERUNG
(5)
(6) EREIGNIS ERWARTEN
(EH2)
(DH2)

FIG. 5A

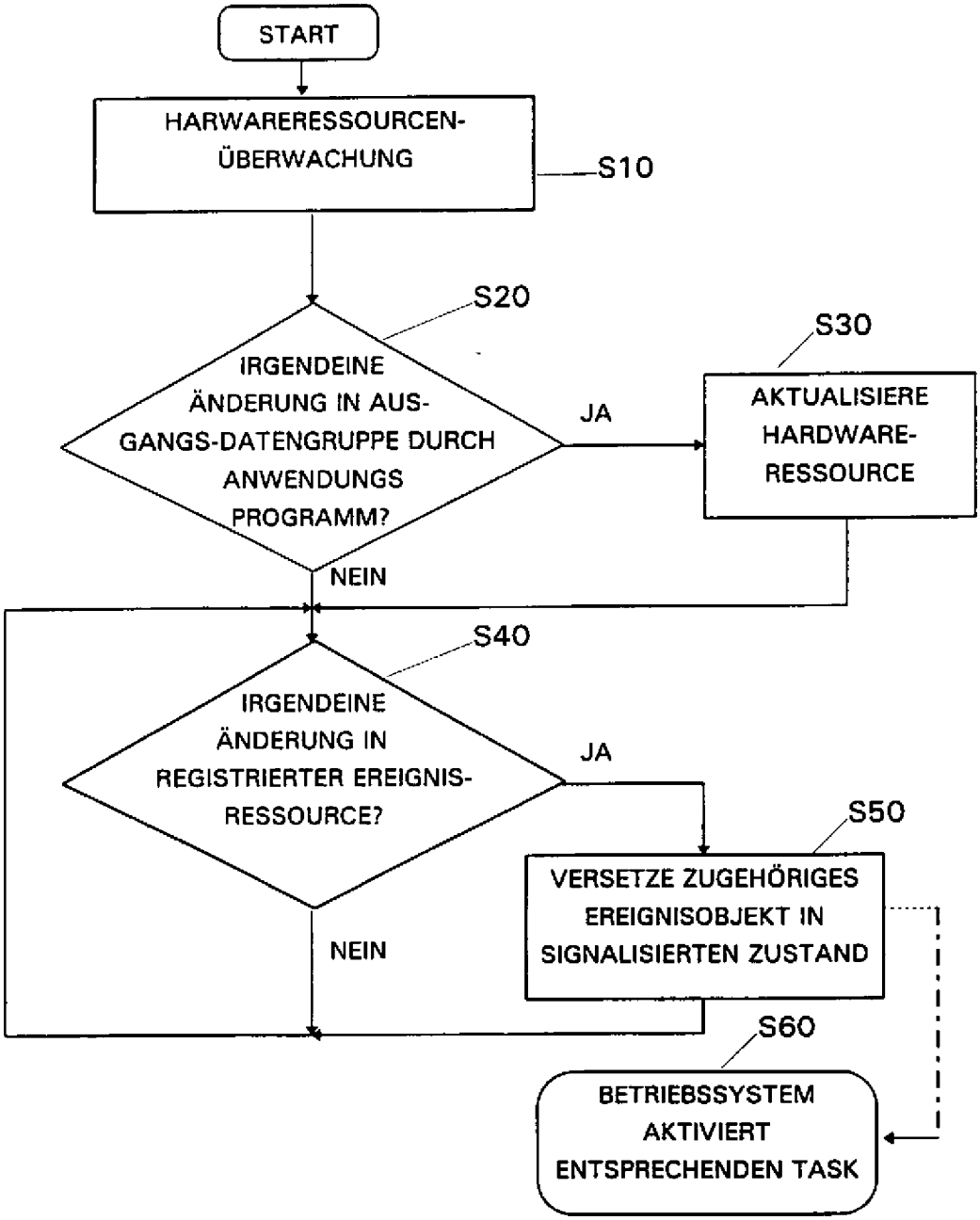

START
HARWARERESSOURCEN-ÜBERWACHUNG
S10
S20
IRGENDEINE ÄNDERUNG IN AUS-GANGS-DATENGRUPPE DURCH ANWENDUNGS PROGRAMM?
JA
S30
AKTUALISIERE HARDWARE-RESSOURCE
NEIN
S40
IRGENDEINE ÄNDERUNG IN REGISTRIERTER EREIGNIS-RESSOURCE?
JA
S50
VERSETZE ZUGEHÖRIGES EREIGNISOBJEKT IN SIGNALISIERTEN ZUSTAND
NEIN
S60
BETRIEBSSYSTEM AKTIVIERT ENTSPRECHENDEN TASK

FIG. 5B

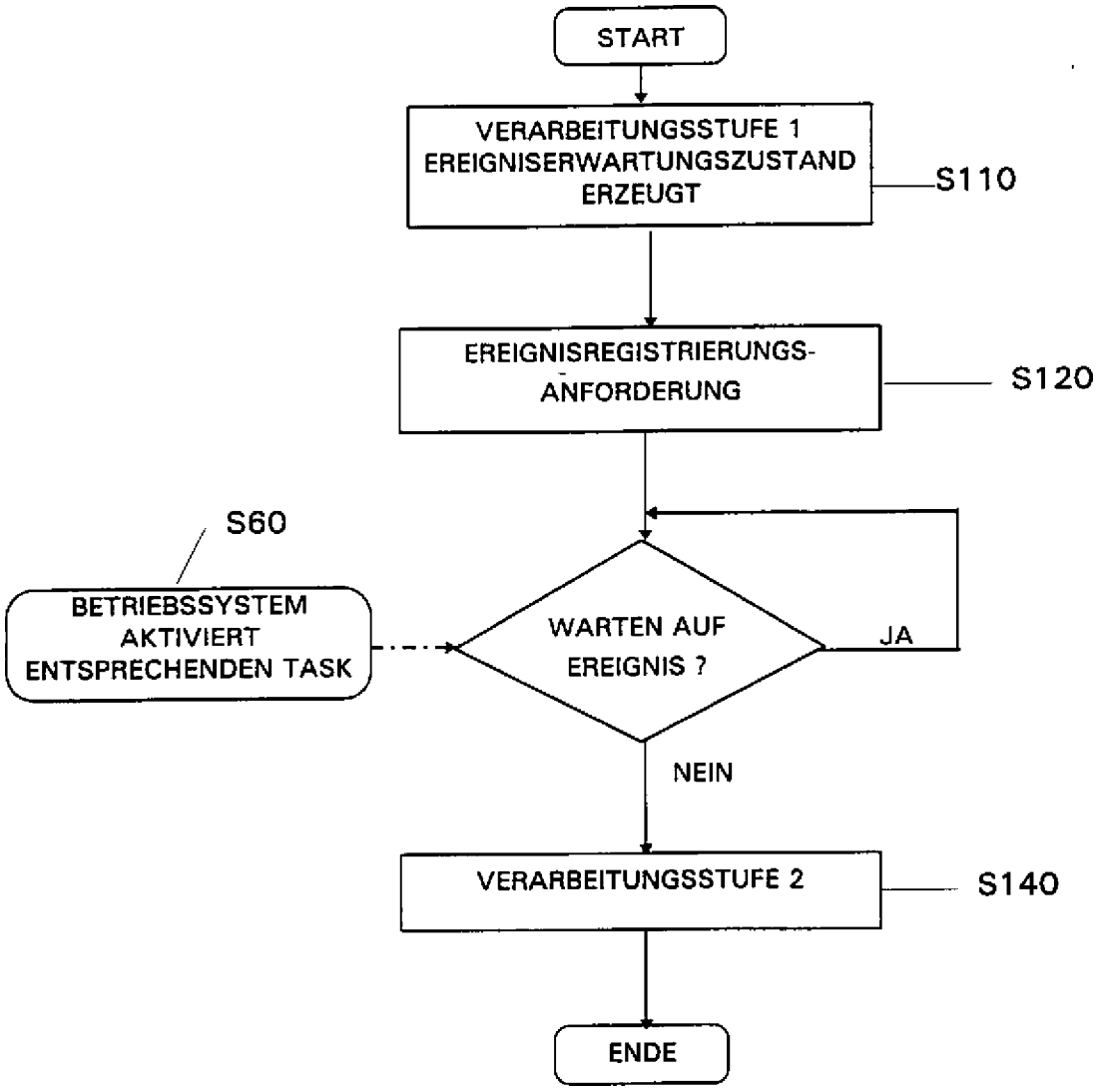
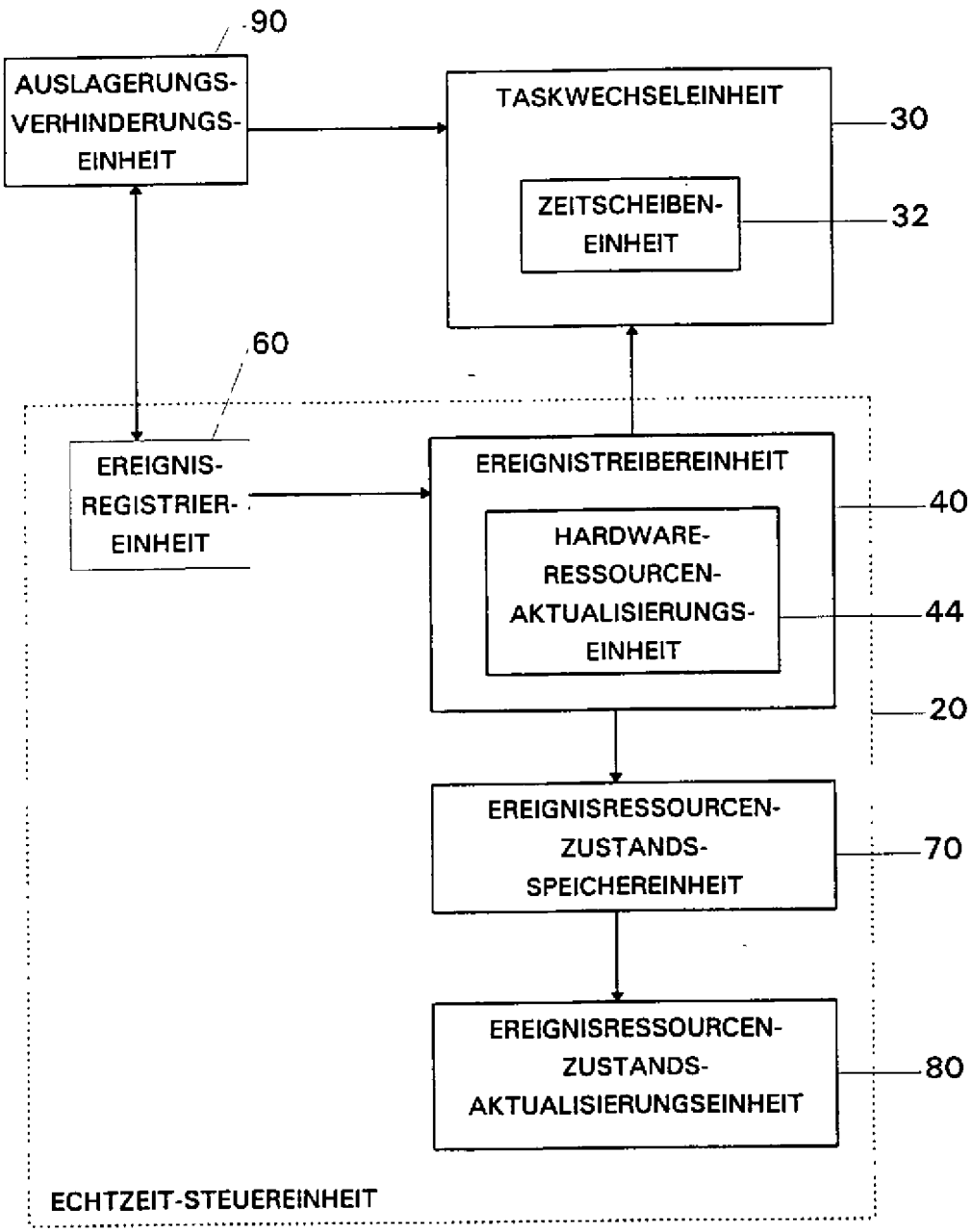

START
VERARBEITUNGSSTUFE 1
EREIGNISERWARTUNGSZUSTAND
ERZEUGT
S110
EREIGNISREGISTRIERUNGS-
ANFORDERUNG
S120
S60
BETRIEBSSYSTEM
AKTIVIERT
ENTSPRECHENDEN TASK
WARTEN AUF
EREIGNIS ?
JA
NEIN
VERARBEITUNGSSTUFE 2
S140
ENDE

FIG. 6


90
AUSLAGERUNGS-VERHINDERUNGS-EINHEIT
TASKWECHSELEINHEIT
30
ZEITSCHEIBEN-EINHEIT
32
60
EREIGNIS-REGISTRIER-EINHEIT
EREIGNISTREIBEREINHEIT
40
HARDWARE-RESSOURCEN-AKTUALISIERUNGS-EINHEIT
44
20
EREIGNISRESSOURCEN-ZUSTANDS-SPEICHEREINHEIT
70
EREIGNISRESSOURCEN-ZUSTANDS-AKTUALISIERUNGSEINHEIT
80
ECHTZEIT-STEUEREINHEIT