1 Embedded Systeme

In der Literatur findet man unterschiedliche Umschreibungen zum Begriff *Embedded System (ES)*. Die hier vorgeschlagene Definition ist möglichst allgemein und gehalten:

> Ein *Embedded System* ist ein Computer System, das seinen Nutzen
> als Bestandteil eines technischen Systems erbringt.

Das umgebende technische System kann ein kleines Gerät sein, eine Maschine oder eine ganze Produktionsanlage. Wir werden die umgebenden Komponenten eines ES als *Technische Prozesse* bezeichnen. Dazu zählen wir der Einfachheit halber auch bediente Eingabegeräte und Anzeigen. Weiter können auch andere, bereits produktive embedded Systeme zur Umgebung des zu entwikkelnden ES gehören. Ein ES besteht aus einem oder mehreren Mikroprozessoren, auf denen systemspezifische Programme laufen. Ein Teil des ES kann auch aus eigens für das System entwickelten Hardware-Komponenten bestehen. Embedded Systeme finden wir heute fast überall, zum Beispiel in Haushalts- und Kommunikationsgeräten, in Automobilen, Schienenfahrzeugen und Flugzeugen, in allem mechatronischen Systemen und bei allen teilweise oder ganz automatisierten Produktionsanlagen. Früher wurden Embedded Systeme als Prozessrechner-Systeme oder Prozesssteuerungen bezeichnet.

Im folgenden wird auf die Unterschiede zwischen Embedded Systemen und reinen informationsverarbeitenden Anwendungen wie Softwarewerkzeuge, Administrationsprogramme oder Datenbankapplikationen eingegangen, und zwar bezüglich *Zweck* der Anwendung, *typischen Anforderungen* und *technischer Ausprägung*.

1.1 Der Zweck von Embedded Systemen

Die ersten Computer wurden entwickelt um Berechnungen zu automatisieren, daher auch der Name Computer (Rechner). Heute ist der Zweck einer Computeranwendung meistens die Verarbeitung von Daten, oder besser gesagt von *Information*. Unter Information verstehen wir hier binäre Daten (Codes), die eine bestimmte Bedeutung haben. In diesem Sinn liest eine Computerapplikation Informationsströme als Input, verarbeitet diese Information und erzeugt Informationsströme als Output. Diesen Prozess bezeichnen wir als Informationsverarbeitung oder *Information Processing*.

Ein Textverarbeitungsprogramm zum Beispiel empfängt vom Benutzer Schriftzeichen (ASCII Code) und bestimmte Formatierungsanweisungen (Formatierungscode) und erzeugt daraus einen formatierten Text als Output. In Text- oder Grafikwerkzeugen kann der Benutzer die Informationsverarbeitung jedoch kaum erkennen, da Ein- und Ausgaben an der benutzerfreundlichen Werkzeugschnittstelle die Codierung durch symbolische und grafische Werte ersetzt werden. Aber auch das Programm, das die Benutzeschnittstelle bedient, macht nichts anders als Informationsverarbeitung.

Embedded Systeme sind anders. Ein Embedded System wird nicht entwickelt, um Information zu verarbeiteten, sondern um zwischen den technischen Prozessen der Systemumgebung bestimmte physikalische Interaktionen zu erzeugen. Mit anderen Worten, der eingebettete Computer wird so programmiert, dass über den Rechner zwischen den Prozessen reale Interaktion entsteht. Der Rechner wirkt als physikalische Maschine, die ständig mit den Prozessen wechselwirkt. Im Unterschied zu interaktiven Informationssystemen sind die Prozesse, mit denen Information ausgetauscht wird, nicht kognitive Wesen, sondern technische Prozesse ohne Intelligenz. Genau dieser Umstand macht es aber erst möglich, das physikalische Verhalten der Prozesse gezielt über einen Computer zu beeinflussen. Für die Erzeugung des binärem Input und für die Umsetzung des binären Output in physika-

lische Wirkung sind aber systemspezifische Interaktionswandler notwendig. Das sind die Sensoren und Aktoren.

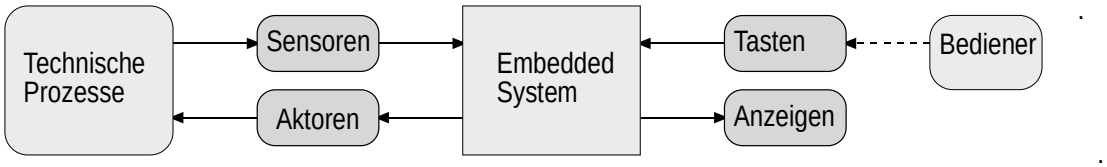


Abb. 1: Embedded System mit Umgebung

Abbildung 1 zeigt die '*Closed Loop*'-Verbindung eines Embedded Systems zu den überwachten und gesteuerten Prozessen sowie die Eingabegeräte und Anzeigen für die Bediener.

Die gewünschte Interaktion zwischen Prozessen könnte auch mittels mechanischer, elektrischer oder hydraulischer Maschinen erzeugt werden, was vor dem Computerzeitalter ja auch die einzigen Möglichkeiten waren. Im Unterschied zu klassischen Maschinen ist der Bau computer-basierter Maschinen jedoch viel "einfacher" und flexibler, da ein Computer als *General Purpose Machine* mittels eines Programms so konfiguriert werden kann, dass die gewünschten Interaktionen mit den Prozessen unterhalten werden.

Für die Entwickler eines Embedded Systems gibt es damit grundsätzlich zwei mögliche Sichten, aus denen die gestellte Aufgabe angegangen werden kann:

Die informationsorientierte Sicht. Der Entwickler sieht das zu bauende System als Informationsverarbeitungssystem, das die Werte von Sensorschnittellen als Information verarbeitet und den notwendigen Output an die Aktor-Schnittstellen übergibt. Mit andern Worten, es muss einfach ein Programm geschrieben werden, das ständig in wiederholten Zyklen Input liest, den Input verarbeitet und Output erzeugt.

Ist das so wirklich so einfach? Wir werden sehen, dass die rein informationsorientierte Sicht bei embedded Entwicklungen problemarisch ist, weil die Welt für den Entwickler bei den I/O-Geräten aufhört. Die Informationsverarbeitung erweist sich nämlich als komplexes Problem, weil für die Programmierung der Steuerfunktionen zuerst die Inputdaten interpretiert werden müssen (Was bedeuten die neuen Sensordaten in Bezug auf das aktuelle Verhalten der Prozesse?), und am Schluss noch Aktordaten aufzubereiten sind (Wie müssen die Outputaten sein, damit die Aktoren ein bestimmte Wirkung auf die Prozesse haben?). Wenn nicht explizit, so muss doch immer, zumindest im Kopf des Programmierers, ein Zusammenhang zwischen den Schnittstellendaten am Rechner und dem Verhalten der Technischen Prozesse hergestellt werden.

Die domänenenorientierte Sicht. Der Entwickler konstruiert ein Steuerprogramm, das über Sensoren und Aktoren mit den Technischen Prozessen interagiert. Neben der angeforderten Steuerung der Prozesse ist auch die Verbindung zu den Sensoren und Aktoren ein zentrales Problem. Ziel ist, die beiden Probleme möglichst getrennt zu lösen. Mehr dazu aber später in Kapitel 2.

1.2 Spezifische Merkmale und Anforderungen

Die umgebenden technischen Prozesse eine ES sind im Gegensatz zu interaktiven Anwendern nicht kognitiv, d. h. sie haben weder Intelligenz noch Verantwortung. Die Verhalten solcher Prozesse ist damit in erster Linie durch deren physikalisch dynamischen Eigenschaften bestimmt. Wir werden sehen, dass dieser Unterschied indirekt ein Hauptgrund für die meisten Erschwerungen bei der Entwicklung von Embedded Systemen ist.

Merkmale, die mit der Nicht-Intelligenz der Technischen Prozesse zusammenhängen

Anwendungsspezifische Kopplung an die technischen Prozesse

Die anwendungsspezifische Koppelungsproblematik mit den technischen Prozessen wird im Abschnitt 1.4 erläutert. Sie ist bei eingebetteten Rechnersystemen die Hauptursache für die anwendungsspezifische Computertechnologie.

Reaktive Funktionalität

Ein ES muss ständig auf externe Ereignisse reagieren und entsprechend auf die technischen Prozesse einwirken. Dabei besteht meistens eine so genannte *Closed-Loop*-Abhängigkeit zwischen technischen Prozessen und ES. Die Einwirkung des ES auf die technischen Prozesse bestimmt, welche Ereignisse auftreten können, und welche nicht! Dabei muss immer mit dem gleichzeitigen auftretenden verschiedener Ereignisse gerechnet werden, weil die Technischen Prozesse von Natur aus parallel sind.

Die reaktive Closed-Loop-Problematik bringt erhebliche Erschwerungen in der Softwareentwicklung, Die verständliche Strukturierung reaktiven Verhaltens gehört zu den anspruchsvolleren Softwareaufgaben.

Harte Echtzeitbedingungen

ES sind Echtzeitsysteme, die in der Regel *harte* Echtzeitbedingungen erfüllen muss. Darunter versteht man Bedingungen die verlangen, dass das System innerhalb vorgeschriebener Zeitspannen auf externe Ereignisse reagiert. Werden solche Bedingungen verletzt, funktioniert das System nicht richtig und es kann Schaden entstehen.

Die Echtzeitprobleme müssen in der SW-Entwicklung gelöst werden, die Grundlage dazu wird aber durch die Rechenleistung der Hardware geliefert. Zudem muss der Rechner Interruptdienste und geeignete Echtzeitfunktionen zur Verfügung stellen (periodische Abtastung, Wartezeiten, Zeitüberwachungen).

Hohe Zuverlässigkeitsanforderungen

Häufig müssen ES anspruchsvolle Zuverlässigkeitsanforderungen erfüllen. Dies gilt insbesondere für *sicherheitskritischen* Systeme, deren fehlerhaftes Verhalten sehr grosse Schäden und u. U. die Gefährdung von Menschenleben zur Folge haben kann.

Zuverlässigkeit kann z. B. durch fehlertolerantes Verhalten erhöht werden, indem kritische Funktionen redundant auf parallelen Prozessoren implementiert werden.

Eingeschränkte Testbarkeit

Die Natur eingebetteter System bringt es mit sich, dass das Testen auf dem Zielsystem schwierig und problematisch ist. Die gesteuerten Prozesse haben ihr eigenes aktives Verhalten, das heisst es können zum Beispiel nicht einfach bestimmte Ereignissequenzen zum Testen des Systemverhaltens erzeugt werden. Zudem können bei der laufenden Anlage die Folgen eines einzelnen Programmierfehlers gravierend sein. Das Testen von Embedded Systemen muss deshalb oft an Simulationsmodellen und auf geeigneten Prüfständen erfolgen.

Weitere besondere Merkmale

Implementierung auf mehreren Prozessoren

Die Steuerung räumlich ausgedehnter Systeme wird meistens entsprechend verteilt implementiert. Beispiele sind Produktionsanlagen oder die verschiedenen Komponenten eines Automobils (Motor, ABS, Airbag, Lüftung). Oft müssen auch mehrere Prozessoren eingesetzt werden, um gezielt die notwendige Rechnerleistung für bestimmter Funktionen zu erbringen. Die verteilte Implementierung eines ES hat auf den Hardware- und den Software-Entwurf stark erschwerende Auswirkungen (Kommunikationsschnittstellen und Protokolle).

Wirtschaftlichkeit: grosse Stückzahl, kleiner Preis

Bei Produkten, die in grosser Stückzahl angefertigt werden, wie z. B. Haushaltgeräte oder Steuerungskomponenten von Automobilen wirken sich die Entwicklungskosten relativ gering auf den Preis des Einzelproduktes aus. Was zählt sind die Kosten der Hardware. Aus diesem Grund wird für die Implementation die kostengünstigste Hardware verwendet (4- und 8-bit Mikrokontroller mit nur einigen KByte ROM)). Die Erschwerungen, die bei der Softwareentwicklung aufgrund der knappen Hardware-Ressourcen entstehen (z. B. Optimierung auf Assemblerebene), werden dabei in Kauf genommen. Ob die gesamte Kostenrechnung stimmt, bleibt aber zu oft ein ungelöster Diskussionspunkt.

Extreme Umgebungsbedingungen

Im Gegensatz zu den meisten Rechenanlagen im kommerziellen oder wissenschaftlichen Anwendungsbereich müssen ES oft mit extremen Umgebungsbedingungen auskommen (grosse Temperaturbereiche, Erschütterungen, EMV, Auflagen bezüglich Volumen oder Gewicht, usw.). Solche Bedingungen schränken die Wahl der Hardware entsprechend ein (Low-Power Mikrokontroller, keine Gebläse, keine Massenspeicher, keine Standardtastaturen).

1.3 Embedded Systeme - Technische Ausprägungen

Embedded Systeme unterscheiden sich auch hardwaremässig von Computersystemen aus dem Desktop-Bereich. Im Unterschied zu typischen Desktop-Anwendungen läuft eine embedded Anwendung in der Regel als einzige Anwendung auf einem spezifischen Computer. Sowohl die Auswahl der Rechnerarchitektur (Embedded-PC, Mikrokontroller, etc.) als auch die verwendeten Input/Output-Module und Bussysteme sind fast immer auf das einzelne Embedded System zugeschnitten.

Systemspezifische Ein- und Ausgabegeräte (Sensoren und Aktoren)

Bei Computeranwendungen im kommerziellen Bereich können Systemlösungen rein auf Software-Ebene gefunden werden. Standardisierte Ein- und Ausgabefunktionen (Tastatur, Bildschirm, Files) werden durch das Betriebssystem zur Verfügung gestellt. Embedded Systeme hingegen können nicht als reine Softwarelösungen entwickelt werden. Vor allem im Bereich der Ankoppelung an die technische Umgebung wird die Verwendung von anwendungsspezifischer Hardware notwendig.

Die funktionalen Anforderungen an das ES beziehen sich, wie gesagt, auf eine Umgebung, die aus technischen Prozessen besteht. Wenn z. B. in einem Lift ein Etageknopf gedrückt wird, muss die Lifttüre schliessen und anschliessend der Lift zum gewählten Stockwerk fahren. Damit es möglich ist, den Lift über einen programmierten Rechner zu steuern, muss eine physikalische Koppelung zwischen Rechner und Lift bestehen. Die Realisierung solcher Kopplungen ist neben der Entwicklung der embedded Software eine zweite Hauptaufgabe bei der Realisierung eines ES, für die System-Ingenieure zuständig sind.

Sensoren sind Interaktionswandler, die physikalisch mit den technischen Prozessen interagieren und bestimmte Prozessphänomene in digitale oder analoge elektronische Signale umsetzen. Umgekehrt erzeugen Aktoren aus elektronischen Signalen bestimmte physikalische Wirkungen bei den techni-

schen Prozessen. Zudem sind meistens spezifische elektronische Bausteine notwendig für die Vor- und Nachbearbeitung der vom Rechner empfangenen und erzeugten Signale (A/D-Wandler, Filterfunktionen, Pegelanpassung). Die Anbindung des embedded Rechners an ein heterogenes Arsenal von I/O-Geräten ist damit eher die Regel, und hat auch einen entsprechenden Einfluss auf die Komplexität der Ein- und Ausgabesoftware.

Integrierte Hardware-Funktionseinheiten

Neben der grundlegenden Rechnerfunktionalität muss die Hardware einer eingebetteten Komponente wie gesagt fast immer auch Funktionen unterstützen, die eine elektronische Koppelung zu den Sensoren und Aktoren ermöglichen. Dazu kommen Bausteine für Zeitfunktionen und für die Kommunikation mit anderen Komponenten des eingebetteten Systems. Um die Bausteine effizient benutzen zu können ist zudem eine geeignetes Interrupt System notwendig. *Abbildung 2* zeigt die Basiskomponenten (CPU, ROM, RAM, Ports) und spezifische Funktionseinheiten eines eingebetteten Prozessorsystems.

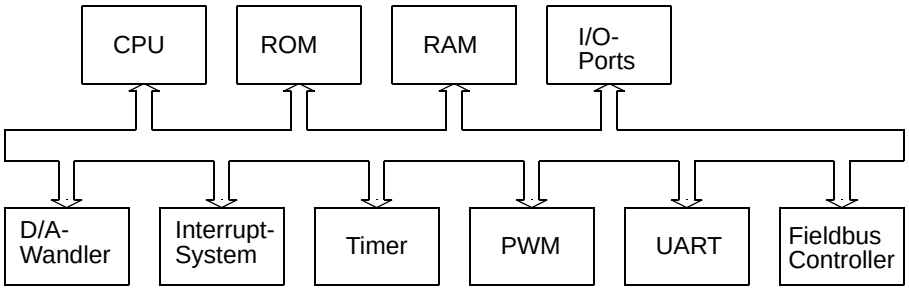


Abb. 2: HW Basiskomponenten und Funktionseinheiten eines eingebetteten Systems

Koppelungsfunktionen

Funktionseinheiten für die elektronische Koppelung zur Systemumgebung werden oft mit dem Basisrechner zu einem Embedded Rechner integriert. Harte Echtzeitanforderungen verlangen, dass die Koppelungselektronik möglichst direkt mit dem Rechner verbunden ist. Zum Beispiel muss der Rechner einen A/D-Wandler zu ganz bestimmten Zeitpunkten (Sampling) aktivieren und die gewandelten Daten möglichst schnell auswerten können.

Zeitfunktionen

Für Regelungen, zeitgetriebene Steuerfunktionen und Zeitüberwachungen stellt die Hardware geeignete Zeitfunktionen zur Verfügung. Im Prinzip könnten solche Funktionen auf der Basis des Systemtaktes programmiert werden, was aber bei hoher Zeitauflösung unnötig viel Rechenleistung beansprucht. Typische Beispiele von Zeitfunktionen sind Timer, Watchdog. PWM (Pulse Width Modulation).

Kommunikationsfunktionen

Bei verteilten Implementationen ermöglichen Steuereinheiten für Busschnittstellen die Kommunikation mit anderen Systemkomponenten. Für die Komponenten eines lokalen Systemkomplexes (Rack) werden in der Regel Parallelbusse eingesetzt (z. B. VME Bus), für räumlich verteilte Komponenten arbeitet man mit seriellen Bussystemen, die als Feldbusse bezeichnet werden (z. B. CAN-Bus-Systeme in Automobilen). Dazu kommen oft auch universelle serielle Schnittstellen (UART), an die z. B. Peripheriegeräte angeschlossen werden.

Ausprägungen von Hardwarekonfigurationen

Der Embedded-Markt bietet vollständige Hardware-Konfigurationen an, typischerweise als Processor Boards oder als Single-Chip-Rechnersystem (Mikrokontroller). Gekaufte Hardware-Konfigura-

tionen haben den Vorteil, dass sie bereits getestet und in der Regel schon in anderen Projekten eingesetzt worden sind, selbstentwickelte bringen grössere Flexibilität in der Entwicklung und im Unterhalt eines ES.

Die Auswahl eines Mikroprozessors oder einer Prozessorkarte ist in jedem Fall ein wichtiger Entwicklungsentscheid. Das zentrale Auswahlkriterium, vor allem bei Mikrokontrollern, muss die Funktionalität sein. Von Vorteil ist ein Rechnersystem, das einer Familie von Rechnersystemen angehört, die variable Funktionalität unterstützt. Weitere wichtige Auswahlkriterien sind die Qualität der Entwicklungsumgebung, Dokumentation und der Support.

Processor Boards, Industrie PC

Bei Processor Boards sind die Basiskomponenten des Rechnersystems und die spezifischen Funktionseinheiten auf einer Platine integriert. Dazu kommen Steckplätze für zusätzliche anwendungsspezifische Erweiterungen. Als Prozessoren werden General-Purpose Prozessoren oder sog. Embedded Prozessoren (einzelnen Zusatzfunktion on Chip) mit hoher Leistung verwendet.

Processor Boards werden typisch bei komplexen Anlagen mit hoher Anforderung an Leistung und Zuverlässigkeit eingesetzt. Das Programm läuft in einem im RAM-Speicher, der bei jedem Systemstart neu aus einem Flash ROM geladen wird.

Mikrokontroller

Bei einem Mikrokontroller sind der Prozessor, der Speicher und die spezifischen Funktionsbausteine auf einem Chip integriert. Das Programm läuft in der Regel im einem ROM-Speicher.

Mikrocontroller werden in erster Linie für die Steuerung von Geräten eingesetzt, vor allem für sog. Low-Cost Produkte, die in hoher Stückzahl vermarktet werden (z. B. Haushaltgeräte oder verteilte Komponenten bei Automobilen wie Motorsteuerung, ABS, Airbag, ...).

Speicherprogrammierbare Steuerungen - SPS

Viele Automatisierungsaufgaben werden mit speicherprogrammierbaren Steuerungen (SPS) gelöst. Geschichtlich haben die speicherprogrammierbaren Steuerungen die klassischen Relaissteuerungen abgelöst. SPS werden in getakteten Zyklen programmiert: *Eingabe, Verarbeitung, Ausgabe*. SPS-Systeme basieren heute auf Standard-Mikroprozessoren und werden als modular konfigurierbare Systeme mit Speicherkomponenten und Kommunikationsschnittstellen angeboten.

Bemerkung:
In der Praxis hat sich folgende Unterscheidung in der Verwendung der Begriffe *'Embedded System'-Bereich* und *Automatisierungstechnik* eingebürgert. Im *'Embedded System'-Bereich* werden Processor Boards oder Mikrokontroller eingesetzt, und es wird vorwiegend in C (C++) programmiert. In der *Automatisierungstechnik* wird SPS-Technologie angewendet. Bezüglich der grundlegenden Problemstellung, nämlich der Automatisierung technischer Prozesse, ist diese Unterscheidung jedoch nicht von zentraler Bedeutung.

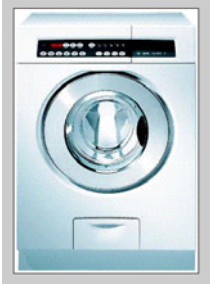
HW/SW-Codesign

Wir sind in dieser Einführung auf prozessor-basierte eingebettete System eingegangen. Bei diesen Systemen steckt die spezifische Anwendungsfunktionalität in einem Programm, das durch einen Prozessor abgearbeitet wird. Eingebettete Komponenten können aber auch als reine Hardware Systeme entwickelt werden (ASIC, FPGA). Als vielversprechender Ansatz bietet sich damit die Entwicklung von Systemen an, die sowohl aus Hardware- und Software-Komponenten bestehen. Der Entwurf solcher Systeme wird als HW/SW-Codesign bezeichnet.

1.4 Zwei Beispiele von Embedded Systemen

Die beiden vorgestellten Embedded-Architekturen entsprechen nicht mehr dem neuesten Stand der entsprechenden Produktentwicklungen.

Waschmaschine (V-Zug AG)

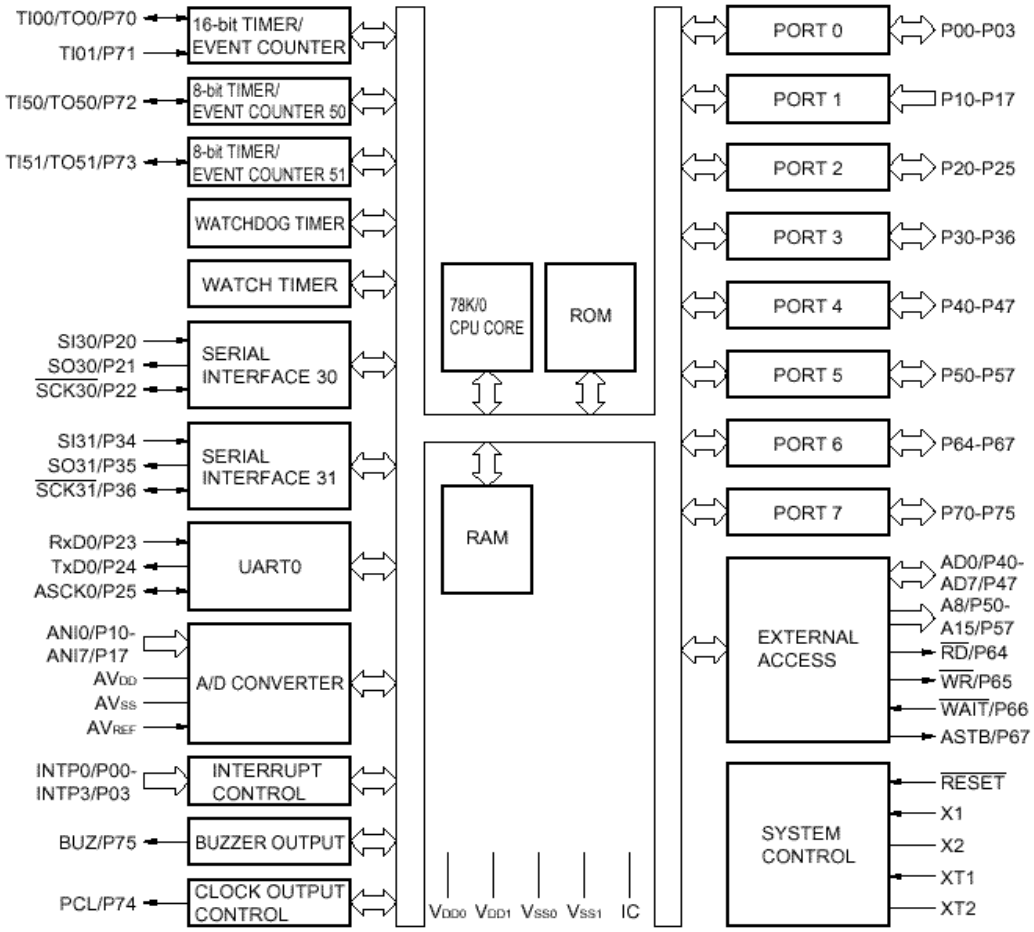


Anforderungen

Standardfunktionalität einer Waschmaschine

Minimale Hardwarekosten (grosse Stückzahl)

Implementation



NEC Mikrokontroller uPD780024

64-pin, 8,38 MHz, 32kB ROM, 1kB RAM, 8 A/D-Wandlerkanäle zu 8 Bit

1 UART für Diagnose-Schnittstelle

Tasten und LCD-Anzeige: 4-Bit Datenports

Eigenes, sehr rudimentäres Betriebssystem.

Fast alles in C programmiert, wenige zeitkritische Teile in Assembler.

Gleiskorrekturmaschine (J.Müller AG, Effretikon)

Funktionalität

Daten für die Solllage des Geleises auf Diskette verfügbar.

Wegmessung, Lasersspiegelung, Dynamischer Kreisel zur Bestimmung der Gleisabweichung.

Ansteuerung der Korrekturmaschine (heben, senken, stopfen).

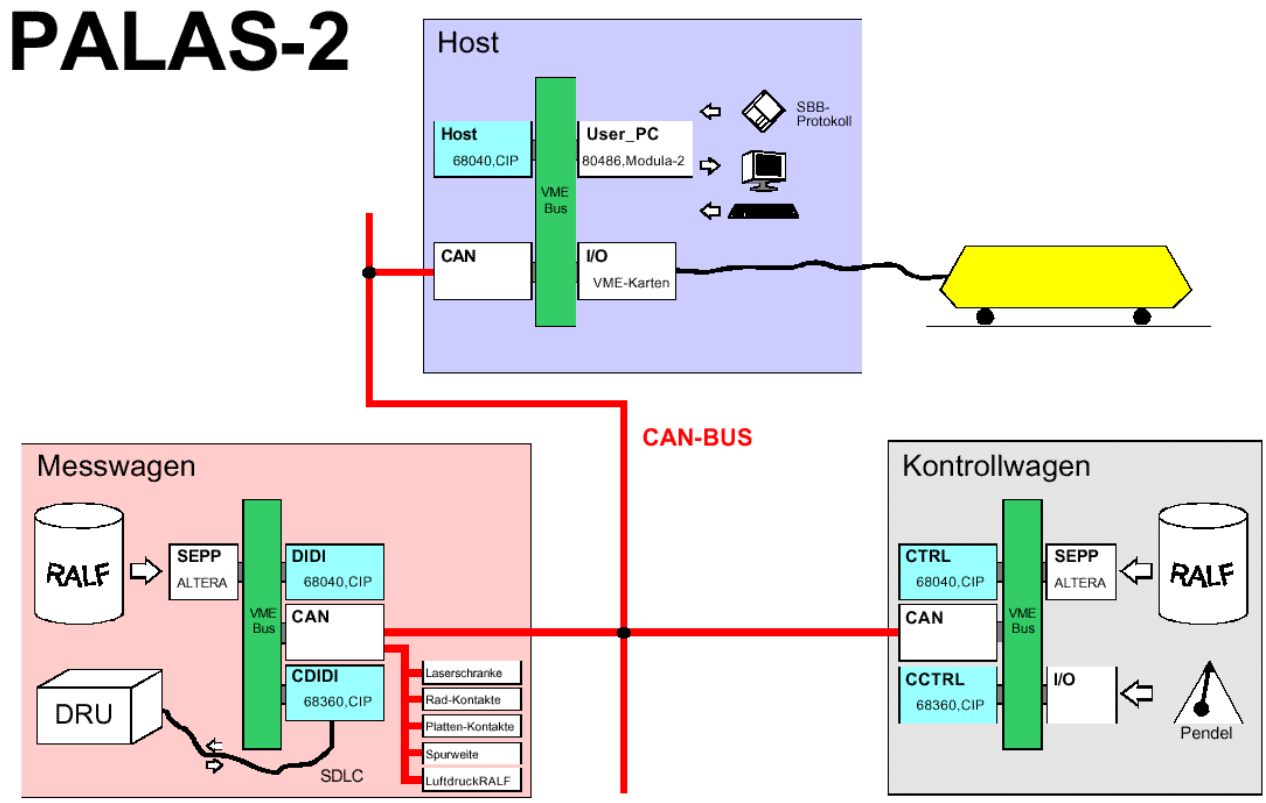
Kontrolle der Gleiskorrektur.

Die Maschine fährt mit 1 m/s.
Die Abweichung des korrigierten Geleise vom Sollwert ist kleiner als 1 mm.

Implementation

Drei verteilte Prozessorkomplexe, verbunden mit Can-Bus.

Pro Komplex (Rack) verschiedene Processor Boards (MC68030, MC68040, 80486, ALTERA FPGA), verbunden mit VME-Bus.



RALF-Laserscanner: **R**otating **A**ngle Measuerement by **L**aser Re**f**lection

DRU-Kreiselplattform: **D**ynamic **R**eference **U**nit