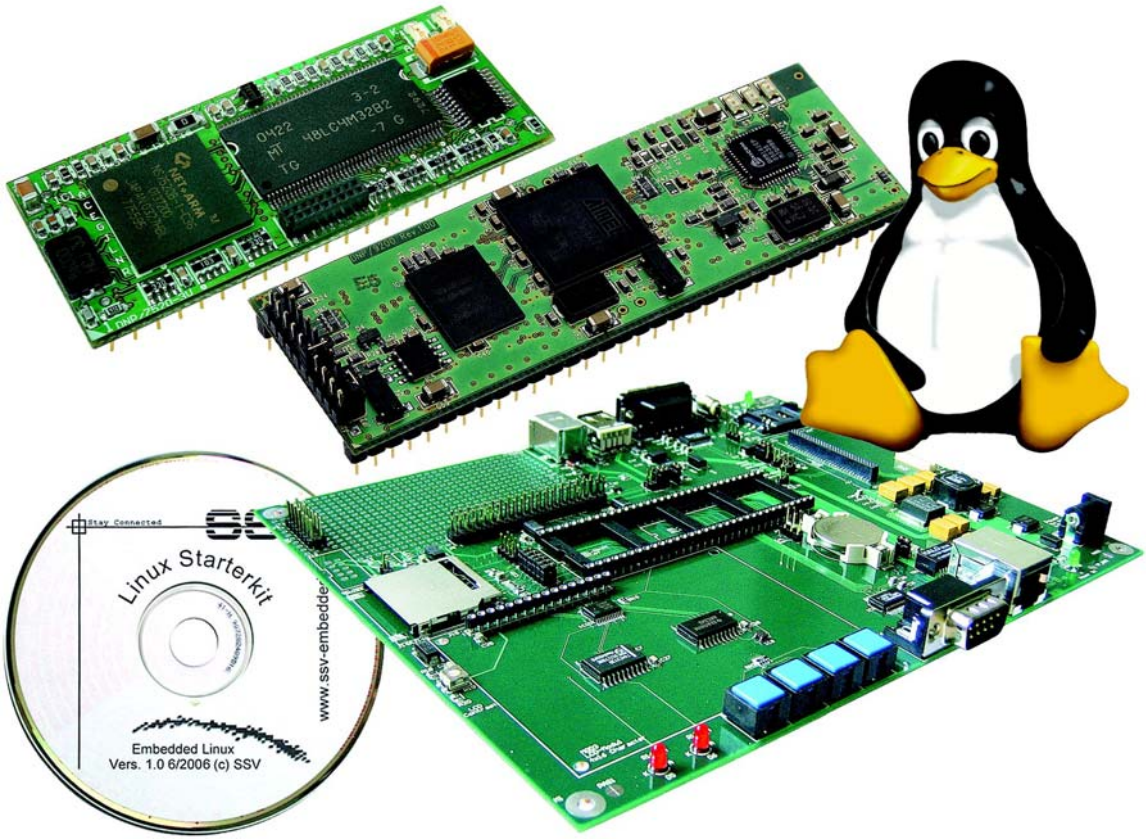

Linux Control Kochbuch

DIL/NetPCs DNP/7520 und DNP/9200

Inhalt

1. Einleitung 3
2. ARM-Architekturen 4
2.1 RISC und CISC 4
2.1.1 RISC-Entwurfziele 5
2.1.2 Pipelining 6
2.1.3 Cache-Speicher 7
2.2 Was ist ARM? 8
2.2.1 ARM-Architekturdetails 10
2.2.2 ARM-Register 11
2.2.3 ARM-Befehlssatz 13
2.3 Aufbau von ARM-MCUs und SoCs 14
2.3.1 Speicheraufteilung (Memory Map) 16
2.3.2 5V-Toleranz 18
3. Schaltungsbeschreibung DNP/7520 19
4. Schaltungsbeschreibung DNP/9200 22
A.1 Wichtige Hinweise 26

1. Einleitung

Sie halten die erste Version des Linux Control Kochbuchs zu den DIL/NetPCs DNP/7520 (Linux Control 1) und DNP/9200 (Linux Control 2) in den Händen. Zunächst einmal eine sehr wichtige Information vorab: **dieses Handbuch lebt**. Es wird laufend weiterentwickelt. Sie können durch spezielle Fragen zum Einsatz des DNP/7520 bzw. DNP/9200 zur weiteren Gestaltung des Inhalts beitragen. Wir werden die eingehenden Fragen und Supportanfragen nach entsprechender Eingangsprüfung als Anregungen für die Weiterentwicklung nutzen.

Die Linux Control DIL/NetPCs DNP/7520 und DNP/9200 basieren auf sogenannten ARM-Mikrocontrollern. Beim DNP/7520 kommt eine ARM7TDMI-MCU zum Einsatz, der DNP/9200 benutzt eine ARM9-MCU. Aus diesem Grund liefert das folgende Kapitel zunächst einmal einen Überblick zu den Grundlagen der ARM-Architekturen.

Die beiden weiteren Kapitel beinhalten eine Schaltungsbeschreibung der DIL/NetPCs DNP/7520 und DNP/9200 mit den wichtigsten Details. Innerhalb dieser Kapitel wird auch die jeweilige Pinbelegung der DIL-40- (DNP/7520) bzw. DIL-64- (DNP/9200) Steckverbinder vorgestellt.

Ihr DIL/NetPC-Team

www.dilnetpc.com
www.dilnetpc.com/linuxcontrol
support@ist1.de

2. ARM-Architekturen

Es ist noch nicht sehr lange her, da wurden die meisten Embedded-Systeme mit 32-bit-Wortbreite als Embedded-PCs – also auf Basis der IA-32-Prozessoren mit Intel-x86-Strukturen – konzipiert. Embedded-PC- und Büro-PC-Plattformen waren am Anfang in etwa identisch. Als Prozessoren wurden damals 386- oder 486/586-Lösungen eingesetzt. Es gibt immer noch einige MCUs mit IA-32-Prozessor-Core und PC-Architektur. Der Erweiterungsbus solcher Systeme basierte auf den 16-bit-ISA- und etwas später auf den 32-bit-PCI-Spezifikationen. Festplatten (jawohl, für Embedded-PCs wurden vielfach Festplatten mit rotierenden Teilen benutzt – man wollte ja Microsoft-PC-Betriebssysteme einsetzen) waren über das parallele ATA-Interface mit dem Mainboard verbunden. Damals reichten sogar 4-, 8- oder 16-Mbyte Arbeitsspeicher für den Einsatz eines Microsoft-Betriebssystems (zum Beispiel Windows 3.x, Windows 95) aus. Heute besitzt jede moderne Grafikkarte schon erheblich mehr Speicherplatz. Aktuelle PC-Prozessoren – wie zum Beispiel der Intel-Pentium 4 – verwenden *Hyperthreading* und arbeiten bereits mit einer Taktgeschwindigkeit von mehreren GHz.

Solche Prozessoren benötigen sehr viel Strom, der in Abwärme umgesetzt wird. Sie werden in den PCs mancher Hersteller bereits mit einer Wasserkühlung betrieben. Frontside-Bussysteme, also die Verbindung zwischen CPU und Arbeitsspeicher, arbeiten demnächst mit 800 MHz (FSB800 als Nachfolger für FSB400 und FSB533), und somit erheblich schneller als die 32-bit-Embedded-MCUs in den meisten Embedded-Control-Anwendungen. Relativ aufwendige serielle Erweiterungsbus-systeme (PCI Express) und Laufwerksschnittstellen (Serial ATA) lösen die parallelen Varianten ab. Mit anderen Worten: aktuelle PC-Plattformen sind als Basis für Embedded-Systeme nur noch bedingt geeignet.

Parallel zu den IA-32-Prozessoren hat sich in den letzten Jahren die ARM-Architektur nachhaltig verbreitet. Die Ursache dafür waren in erster Linie batteriebetriebene Geräte, bei denen der Stromverbrauch eine große Rolle spielt. ARM-CPUs kommen überwiegend in mobilen Telefonen, PDAs, tragbaren Spielkonsolen und Embedded-Systemen mit 32-bit Wortbreite zum Einsatz. Die größte Stückzahl eingebetteter ARM-Lösungen verteilt sich sicherlich auf Handys und auf andere Systeme im Konsumerbereich, wie zum Beispiel MP3-Player. ARM-Lösungen im Anwendungs-segment *Messen-Steuern-Regeln* verbreiten sich aber ebenfalls mit beachtlicher Geschwindigkeit. Das hat sicherlich auch damit zu tun, dass ARM wohl die derzeit preisgünstigere 32-bit-Mikrocontroller-Plattform ist.

2.1 RISC und CISC

Bevor wir in die Tiefen der ARM-Architekturen eintauchen, muss zunächst der Begriff *RISC* geklärt werden. Diese Abkürzung steht für *Reduced Instruction Set Computer*. RISC ist eine Prozessorarchitektur mit einem einfachen Befehlssatz, also das Gegenteil einer *CISC*-Architektur (CISC = Complex Instruction Set Computer, IA-32-Prozessoren sind typische CISC-CPUs). CISC war lange Zeit die dominante Architektur beim Entwurf von Prozessoren. Das Entwicklungsziel für CISC-CPUs war, mit einem einzigen Maschinenbefehl möglichst komplexe Operationen ausführen zu können. Das dabei auftretende Problem einer CISC-Architektur ist, dass zur Ausführung eines einzigen CISC-Befehls viele Taktzyklen benötigt werden.

CISC-Befehlssätze wurden vielfach durch Mikroprogramme in den CPUs umgesetzt. Mit anderen Worten: ein einziger CISC-Befehl wurde CPU-intern durch eine Folge von Mikrobefehlen

abgearbeitet. CISC-Befehle verwenden unterschiedliche Befehlsformate. Es gibt zum Beispiel Anweisungen mit einem 8-, 12-, 16- oder auch 24-bit-Operationscode. Die Ursprünge der CISC-Technologie liegen bei den Minicomputern der späten 60er und frühen 70er Jahre (PDP8 und PDP11 von DEC usw.). Die Prozessoren dieser Rechner wurden mit Hilfe vieler integrierter Schaltkreise aufgebaut (die erste PDP8-Variante bestand sogar nur aus einzelnen Transistoren). Damals war externer Speicher langsam und sehr teuer. Aus diesem Grund wurden CPUs mit Mikrocode-ROMs für die bereits angesprochenen Mikroprogramme ausgestattet, die dann die jeweiligen CISC-Maschinenbefehle implementierten. Mikrocode-ROMs waren deutlich schneller als der Hauptspeicher einer CPU. Mit der Verbreitung des Mikroprozessors wurde das Mikrocode-ROM zum Problem: Es benötigte relativ viel Silizium-Fläche, die für andere Funktionen fehlte. Die Befehlssätze vieler Mikroprozessoren wurden fest verdrahtet und damit – im Vergleich zu den Minicomputern – auch wieder etwas einfacher. Weiterhin machte die Speichertechnologie mit der Verbreitung komplexer integrierter Schaltkreise sehr große Fortschritte.

Ausgangsbasis für die RISC-Entwicklung war vor vielen Jahren eine IBM-Studie, aus der hervorging, dass bei einem CISC-Befehlssatz von ca. 200 einzelnen Befehlen eines IBM-Minicomputers ungefähr 10 Befehle 80% des Codes der meisten Programme, 21 Befehle 95% und 30 Befehle 99% ausmachten. Die verbleibenden 170 Befehle kommen gerade einmal für 1% des von einem Compiler erzeugten Codes zum Einsatz. Als Folge dieser Studie reifte die Erkenntnis, den Befehlssatz einer CPU auf ein Minimum zu reduzieren und die einzelnen Befehle in einem Schaltwerk fest zu verdrahten. Aus diesen Grundlagen sind RISC-Architekturen entstanden. Das Ziel beim Entwurf eines RISC-Prozessor ist es, die meisten Befehle mit einem einzigen Taktzyklus aufführen zu können. RISC bewirkt somit auch eine Leistungssteigerung für Prozessoren.

2.1.1 RISC-Entwurfziele

In den 80er Jahren versuchten verschiedene Einrichtungen RISC-CPUs zu entwickeln. Sie folgten beim Entwurf neu definierten Zielen. So sollte zunächst einmal vollständig auf die Mikroprogrammierung vergangener Tage verzichtet werden und statt dessen eine festverdrahtete Befehlslogik zum Einsatz kommen, die die meisten Befehle in einem Taktzyklus abarbeitet. Auch hinsichtlich der Befehlsformate wurden Änderungen entwickelt: Für leistungsfähige RISC-CPUs wird ein einheitliches Befehlsformat benötigt. Dadurch entfallen unnötige Fallentscheidungen, die bei einer CISC-CPU benötigt werden, um zum Beispiel zwischen 8-, 16- oder auch 24-bit-Befehlen zu differenzieren. Weiterhin sollten die Datenbewegungsbefehle optimiert werden.

Unterteilt man die Befehlstypen einer CPU in die fünf Schwerpunktgruppen: 1. Ablaufsteuerung, 2. Arithmetische Operationen, 3. Logische Operationen, 4. Vergleiche sowie 5. Datenbewegungsbefehle und wertet dann aus, mit welcher Häufigkeit die Befehle dieser Gruppen in typischen Programmen auftauchen, ergeben sich in etwa die Werte der Tabelle 2.1. Die Erkenntnis solcher Auswertungen ist in der Regel, dass eine CPU die Befehle zur Datenbewegung (Register-Register, Register-Speicher usw.) am häufigsten ausführt. Dabei ist zu berücksichtigen, dass Datenbewegung zwischen den internen CPU-Registern und dem Speicher nicht in einem einzigen Taktzyklus ausgeführt werden können. Für solche Operationen werden immer mehrere Takte benötigt. Um das RISC-Entwurfsziel zu erreichen, die meisten Befehle in einem Taktzyklus abzuarbeiten, werden RISC-Prozessoren mit besonders vielen internen Registern ausgestattet. Dadurch lässt sich die Anzahl der Datenbewegungen zwischen Registern und Speicheradressen minimieren. Für die Datenbewegung zwischen den Registern einer CPU reicht ein Taktzyklus aus. Man bezeichnet daher beispielsweise ARM-RISC-Prozessoren auch als *Load/Store-Architekturen*. Im Befehlssatz solcher RISC-CPUs findet man nur sehr wenige Befehle für den Datentransfer zwischen Registern und Speicheradressen. Genau

genommen reichen zwei Befehle – ein *Load*- und ein *Store*-Befehl – aus. Alle arithmetische und logische Operationen sowie Vergleiche beziehen sich auf die internen Register.

Befehlstyp	Dynamisches Aufkommen
Ablaufsteuerung	22%
Andere	1%
Arithmetische Operationen	15%
Datenbewegung	44%
Logische Operationen	5%
Vergleiche	13%

Tabelle 2.1: Auswertung des dynamischen Befehlsaufkommens

Ein weiterer Unterschied zwischen CISC- und RISC-Architekturen ist die Anzahl der Adressierungsarten (Direkt, Indirekt, Implizit, Indiziert, Relativ, Immediate, Stack-Adressierung, Kombinationen verschiedener Adressierungsarten usw.). Einige 16- und 32-bit-CISC-Mikroprozessoren der 80ger Jahre lieferten sich offensichtlich einen Wettbewerb mit Ziel „*wer hat die meisten und komplexesten Adressierungsarten zu bieten ?*“. Die Ausführung der Befehle mit hochkomplexer Adressierung erfordert sehr viele Taktzyklen. Teilweise ist die Anzahl der benötigten Taktzyklen noch nicht einmal vorhersehbar – zum Beispiel für intelligente String-Kopierbefehle. Da ein einzelner Maschinenbefehl (unabhängig davon, wie viele Taktzyklen zur Ausführung benötigt werden) nicht durch einen Interrupt unterbrochen werden kann, sind CISC-Prozessoren auch nur bedingt für Echtzeitanforderungen geeignet. Beim Entwurf von RISC-Architekturen verzichtet man weitestgehend auf komplexe Adressierungsarten. Dadurch sind RISC-CPUs auch für Echtzeitanwendungen bestens geeignet.

2.1.2 Pipelining

Die Befehlsausführung einer CPU lässt sich in drei Einzelschritte zerlegen: 1. Befehl aus dem Speicher holen (*Instruction Fetch*), 2. Befehl decodieren (*Instruction Decode*) und 3. Befehl ausführen (*Instruction Execute*). Diese drei Schritte sind in weitere Teilschritte unterteilbar. So sind zum Beispiel für die Befehlsausführung in der Regel Daten erforderlich, die zunächst aus einer Quelle geholt und nach einer Manipulation an ein Ziel transportiert werden müssen.

Um die Befehlsbearbeitung zu beschleunigen, findet man in RISC-Prozessoren eine Befehls-Pipeline. Es gibt zum Beispiel ARM-CPUs mit 3-stufigen- (ARM7TDMI) und 5-stufigen (ARM9TDMI) Pipelines. Die Funktion einer 3-stufigen Befehls-Pipeline bildet die drei Einzelschritte *Fetch*, *Decode* und *Execute* direkt ab. Abbildung 3.1 illustriert den Zusammenhang bei der Bearbeitung von drei Befehlen. Während in der Zeitspanne t3 durch die erste Stufe der Pipeline die Execute-Phase für den Befehl 1 bearbeitet wird, kann in der Pipeline-Stufe 2 bereits die Decode-Phase des Befehls 2 erfolgen. Zur gleichen Zeit wird die dritte Pipeline-Stufe für den Execute des 3. Befehls benutzt usw. Die Darstellung in der Abbildung 1.1 bezieht sich auf Befehle, die in einem Taktzyklus ausgeführt werden.

Eine 3-stufige Pipeline ist relativ einfach realisierbar. Sie benötigt auch vergleichsweise wenige Siliziumfläche. Wenn eine CPU einfache Befehle (zum Beispiel arithmetische oder logische Operationen) ausführt, bewirkt eine solche Pipeline, dass diese Befehle in einem einzigen Taktzyklus

ausgeführt werden (*Single Cycle Instructions* oder auch *One Cycle Instructions*). Auch Datenbewegungen zwischen Registern und Vergleiche sind innerhalb eines Taktzyklus ausführbar.

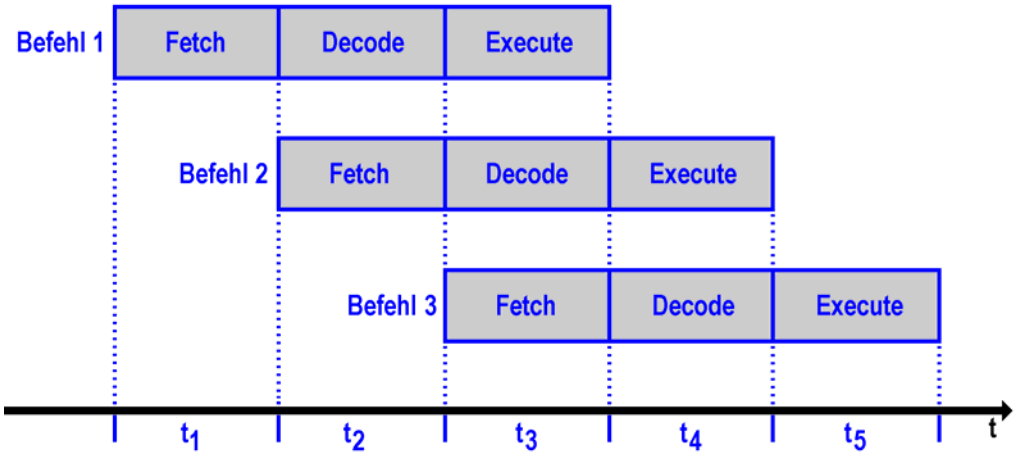

Abb. 2.1: 3-stufige Pipeline einer ARM-CPU MpA0145.tif

Deutlich mehr Zeit erfordern Datenbewegungen zwischen der CPU und dem externen Speicher (Register-Register-, Register-Speicher-Datenbewegungen). Daher ist es sinnvoll, eine CPU mit möglichst vielen internen Registern zu versehen. Durch sie reduziert sich die Anzahl der Datenbewegungen zwischen CPU und externem Speicher.

2.1.3 Cache-Speicher

Ein weiteres Konzept zur Leistungssteigerung einer CPU ist der *Cache*-Speicher. Dieser wird durch einen sehr schnellen Speicher realisiert und zwischen die CPU und den (langsameren) Hauptspeicher platziert (Abbildung 2.2). In einem solchen Cache werden Daten und Programmteile zwischengespeichert, auf die die CPU häufig zugreift. Die Speicherkapazität eines Cache ist deutlich geringer als die des Hauptspeichers. Man unterscheidet zwischen einem Daten-Cache (*D-Cache*), Programm-Cache (*I-Cache*) und einem gemeinsamen Cache (*Unified Cache*). Bei ARM-MCUs findet man alternativ alle drei Cache-Typen.

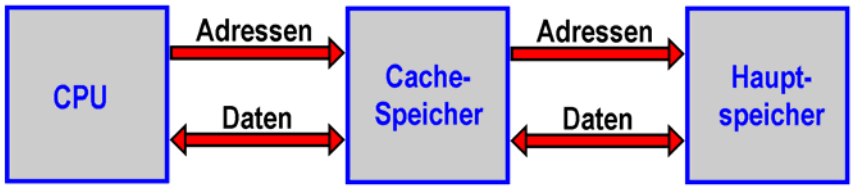

Abb. 2.2: Cache findet man zwischen CPU und Hauptspeicher MpA0146.tif

Ein Cache-Speicher ist in verschiedene Ebenen strukturiert. Es gibt L1- (Level 1), L2- (Level 2) und L3- (Level 3) Cache. Der *L1-Cache* ist in der Regel direkt auf dem CPU-Chip untergebracht. Er ist als D-Cache/I-Cache oder Unified-Cache ausgeführt. Die Speicherkapazität dieses primären Cache-Speichers ist sehr gering. Teilweise beträgt sie nur wenige KBytes. Der sekundäre *L2-Cache* kommt teilweise in hochwertigen PC-Prozessoren zum Einsatz (beispielsweise im *Transmeta Efficeon TM8300*, dessen Blockschaltung im Kapitel 1 zu finden ist). Er verfügt über deutlich mehr Speicherkapazität als der L1-Cache und wird als Unified-Cache genutzt. *L3-Cache* ist eine dritte

Ebene, die allerdings nur bei Hochleistungsrechnern zu finden ist. MCUs und SoCs besitzen bestenfalls einen L1-Cache.

Ist ein Cache vorhanden, prüft die CPU bei jedem Speicherzugriff, ob das adressierte Datum des Hauptspeichers sich im Cache-Speicher befindet. Ist das der Fall, wird es aus dem Cache gelesen, dadurch ergibt sich eine erhebliche Beschleunigung des CPU-Verhaltens. Ansonsten muss die CPU – ohne jeglichen Zeitvorteil – auf den externen Hauptspeicher zugreifen.

Ein Cache-Speicher wird zeilenweise genutzt. Es gibt verschiedene Cache-Organisationsformen. Man unterschiedet zwischen vollassoziativen, direkt abgebildeten und mengenassoziativen Caches. Im ersten Fall kann eine Speicherzelle des externen Speichers in einer beliebigen Cache-Zeile abgelegt sein. Bei einem direkt abgebildeten Cache gibt es eine feste Zuordnung, eine Speicherzelle des externen Speichers ist in genau einer Cache-Zeile abgespeichert. Ein mengenassoziativer Cache speichert eine Speicherzelle des Hauptspeichers in einer der wenigen Cache-Speicherzeilen.

2.2 Was ist ARM?

ARM ist die inzwischen inoffizielle Abkürzung für *Advanced RISC Machines*. *ARM* ist ein Warenzeichen der britischen Firma *ARM Limited*. Dieses Unternehmen ist 1990 aus der *Acorn Computer Limited* entstanden. Acorn baute am Anfang der 80ger Jahre den in England legendären *BBC Micro* – ein einfacher Mikrocomputer, der über die BBC (*British Broadcasting Corporation* – das öffentlich-rechtliche britische Radio- und TV-Unternehmen) im britischen Bildungswesen verbreitet wurde. Aus dem BBC Micro sind die Acorn-Archimedes-Rechner entstanden, die als (erfolgloses) Konkurrenzprodukt zu anderen damaligen Büro-PCs im Markt platziert wurden. Im Rahmen der Archimedes-Vermarktung entwickelte Acorn zwischen 1983 und 1985 die 32-bit-ARM-Architektur. Zunächst wollte man die daraus abgeleiteten 32-bit-CPUs nur in eigenen Rechnern einsetzen. Nach einiger Zeit erkannte man, dass eine Vermarktung als Architektur auf der Basis eines Lizenzmodells sinnvoller sein könnte. Aus diesem Grund wurde die ARM Limited gegründet.

Hinter der ARM Limited standen in der Anfangsphase – neben Acorn – die Firmen *Apple Computer* und *VLSI Technology*. Apple war in erster Linie an besonders stromsparenden Prozessorbausteinen für mobile Computer interessiert. Das US-Unternehmen VLSI wollte primär Prozessor- und Mikrocontroller-Chips fertigen und vermarkten (VLSI wurde inzwischen von Philips übernommen). Auf Basis dieser Zusammenarbeit kam 1993 der *Apple Newton* in den Markt, ein Vorläufer heutiger PDAs. Er basierte auf einem ARM-Prozessor und war wohl der Grund für den ARM-Erfolg. Stromsparende ARM-CPUs waren wie geschaffen für die damals aufkommenden Handys der digitalen Mobilfunknetze.

Die ARM Limited ist eine „Prozessorschmiede“, aber keine Halbleiterfirma. ARM besitzt weder Halbleiterfertigungsstätten (*Semiconductor Fabs* oder auch *Foundries* wie TSMC, UMC, u.a.), noch liefert die ARM Limited fertige integrierte Schaltkreise. Genau genommen handelt die ARM Limited lediglich mit CPU-Lizenzen, die einem ARM Limited-Kunden das Recht einräumen, ein IC mit einer integrierten ARM-CPU zu entwickeln, anzufertigen (oder anfertigen zu lassen) und zu verkaufen. Die ARM Limited bietet verschiedene CPU-Cores (CPU-Kerne), aus denen die ARM-Kunden völlig unterschiedliche MCU- und SoC-Bausteine herstellen können. Typische ARM-Kunden sind in erster Linie Halbleiterfirmen mit eigenen Fertigungsstätten, die ARM-basierte MCUs und SoCs anbieten wollen. Es gibt aber auch Firmen, die selbst über keine eigenen Halbleiterfertigungsstätten verfügen und trotzdem ARM-basierte MCU- und SoC-Bausteine vermarkten möchten. Diese sogenannten *Fabless Semiconductor Companys* besitzen ARM-Lizenzen und entwickeln eigene MCU- und SoC-

Bausteine, die dann aber von hochspezialisierten Fertigungsdienstleistern hergestellt werden. Die Vermarktung der Produkte erfolgt in der Regel wieder über die Fabless Semiconductor Company.

ARM-Core	Architektur
ARM1	V1
ARM2	V2
ARM2aS, ARM3	V2a
ARM6, ARM600; ARM610	V3
ARM7, ARM700, ARM710	V3
ARM7TDMI, ARM710T, ARM720T, ARM740T	V4T
StrongARM, ARMS, ARM810	V4
ARM9TDMI, ARM920T, ARM940T	V4T
ARM9E-S	V5TE
ARM10TDMI, ARM1020E	V5TE

Tabelle 2.2: Übersicht der ARM-CPU-Cores und deren Architektur

Die Vielfalt der unterschiedlichen ARM-CPU-Kerne (*CPU Core*), zu denen auch wieder verschiedenen Architekturen gehören, wirkt auf den ersten Blick recht unübersichtlich. Die Tabelle 2.2 liefert einen Überblick zum Zusammenhang zwischen den gängigen ARM-Cores und Architekturen. Diese Tabelle ist nicht vollständig. Es gibt mit Intel inzwischen einen ARM-Kunden, der das Recht besitzt, ARM-Architekturen zu verändern und der dieses Recht auch intensiv nutzt. Aus dieser Zusammenarbeit sind zum Beispiel inzwischen die Intel-Xscale-Bausteine entstanden.

Abkürzung	ARM-Funktionsdetail
D	Debug-Unterstützung
E	Enhanced DSP Instructions (Befehlssatz mit DSP-Erweiterungen)
I	ARM-EmbeddedICE
J	ARM-Jazelle (Jave-Unterstützung)
M	Enhanced Multiplier (Hardware-Multiplizierer)
S	Synthetisierbarer Core
T	Thumb-Befehlssatz

Tabelle 2.3: Erläuterung der Buchstaben in den ARM-Core-Bezeichnungen

Hinter den ARM-Core-Bezeichnungen in der Tabelle 2.2 findet man teilweise einzelne Buchstaben oder sogar Buchstabenkombinationen. Über diese Buchstaben sind teilweise wichtige Funktionsdetails gekennzeichnet. Tabelle 2.3 liefert weitere Einzelheiten zu diesen Abkürzungen. Eine ARM7**TDMI**-MCU beinhaltet somit (neben dem ARM7-Core) eine Option für den Thumb-Befehlssatz (**T**), die ARM-Debug-Unterstützung (**D**), den Enhanced Multiplier (**M**) sowie die ARM-EmbeddedICE (**I**) – das ist der eingebettete ARM-In-Circuit Emulator (ICE = In-Circuit Emulator). Eine solche Erweiterung bietet dem Entwickler vielfältige Testmöglichkeiten.

2.2.1 ARM-Architekturdetails

ARM-CPU-Architekturen sind – im Vergleich zu den derzeitigen PC-Prozessoren von AMD, Intel, Transmeta, VIA und anderen Anbietern – sehr übersichtlich gestaltet. Zur Realisierung wird eine relativ kleine Siliziumfläche benötigt, was letztendlich auch den geringen Stromverbrauch begründet. Die Abbildung 2.3 illustriert die Blockschaltung einer ARM-CPU-Architektur. Als Vorlage dient ein ARM7TDMI-Core.

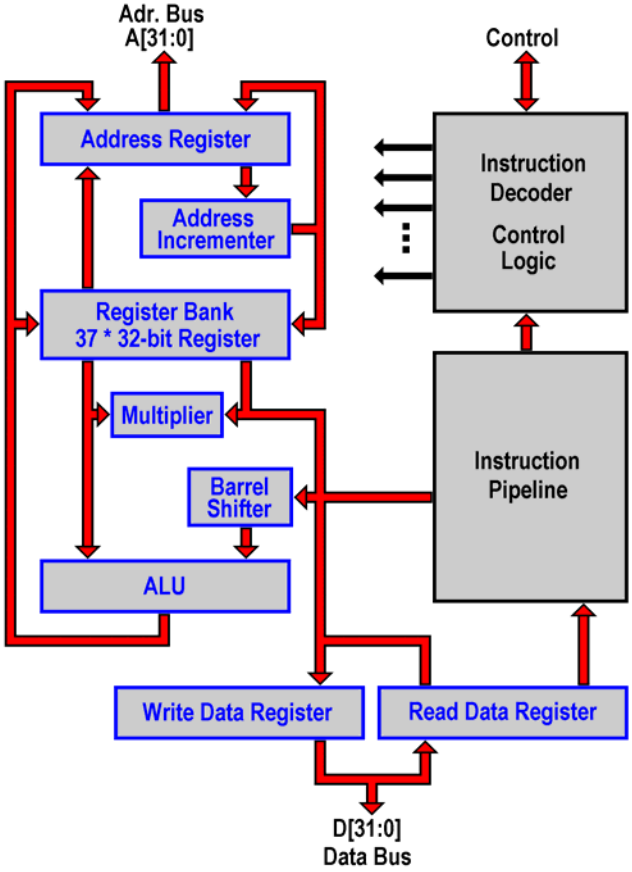

Abb. 2.3: Blockschaltung einer ARM-CPU (ARM Executing Unit) MpA0030.tif

Für die Verbindung zur Außenwelt besitzt ein ARM-CPU-Core einen 32-bit-Datenbus und einen 32-bit-Adressbus (Achtung: Nicht alle ARM-basierten Bausteine mit externem Bus-Interface bieten alle 32 Adressbits als separate Pins am IC-Gehäuse). Die wichtigsten internen Funktionseinheiten sind:

Address Register: Dieses CPU-interne 32-bit-Register enthält stets die Adresse des Speicherplatzes, auf den die CPU zugreifen soll.

Address Incrementer: Über diese ARM-interne Funktionseinheit wird die interne 32-bit-Adresse inkrementiert. Als typische 32-bit-CPU adressiert ARM jeweils 32-bit-Worte (also eine Gruppe aus vier Bytes). Da ein Speicher in der Regel eine Byte-Struktur besitzt, folgt auf die Byte-Adresse x die Adresse $x+4$.

Register Bank: Hier sind die internen 32-bit-Register einer ARM-CPU zusammengefasst, die bei der Befehlsbearbeitung eine wichtige Rolle spielen.

Multiplier: Diese Funktionseinheit führt sehr schnelle Multiplikationen mit Hilfe festverdrahteter Logik aus.

Barrel Shifter: Mit Hilfe dieses speziellen Schieberegisters kann eine ARM-CPU Schiebe- und Rotationsaufgaben um eine beliebige Bitanzahl innerhalb eines Taktes durchführen.

ALU: Die ALU ist das Rechenwerk einer CPU (ALU = Arithmetical Logical Unit). In der ALU einer CPU werden die wichtigsten arithmetischen Grundoperationen ausgeführt.

Write Data Register und **Read Data Register**: In diesen beiden 32-bit-Registern werden die Daten für den Schreib- und Lesezugriff auf externe Funktionseinheiten (Speicher- und Peripherie) zwischengespeichert.

Instruction Decoder / Control Logic: In dieser ARM-Funktionseinheit werden sämtliche Befehle durch eine festverdrahtete Logik dekodiert. Diese Komponente steuert auch die Ausführung der einzelnen Befehle in der CPU.

Instruction Pipeline: Diese Pipeline realisiert die bereits beschriebenen Funktionen zur Beschleunigung der Befehlsbearbeitung. Bei einem ARM7TDMI kommt eine 3-stufige Pipeline zum Einsatz.

Die einzelnen ARM-Core-Implementierungen unterschieden sich in sehr vielen Details. Interessierte sollten hier unbedingt die Website der ARM Limited (*www.arm.com*) als Informationsquelle benutzen, um die entsprechenden Architekturbeschreibungen zu studieren. Es gibt ARM-CPUs – wie zum Beispiel den ARM7TDMI – mit reinen Von-Neumann-Strukturen (intern und extern: jeweils ein gemeinsamer Bus für Daten und Befehle) und Kombinationen aus Von-Neumann- und Harvard-Konzepten (intern: getrennte Bussysteme für Daten und Befehle; extern: gemeinsamer Bus für Daten und Befehle). Manche ARM-CPUs besitzen eine MMU (Memory Management Unit), andere nur eine MPU (Memory Protection Unit). Durch die MMU ist virtueller Speicher und der vollständige Schutz einzelner Programmteile untereinander möglich. Eine MPU überwacht lediglich, dass auf bestimmte Speicherbereiche nur durch spezielle Programmteile zugegriffen wird. Auch hinsichtlich des Cache-Speichers findet man unterschiedliche Ansätze in den einzelnen ARM-CPUs.

Generell gilt die Faustregel: je höher die Architekturnummer, desto ausgefeilter die interne Struktur. So besitzt zum Beispiel ein ARM920T oder ARM940T eine interne Harvard-Architektur mit getrennten Bussen für Daten und Befehle, jeweils einen I-Cache und D-Cache sowie sogar zwei eigenständige MMUs (je eine für den Daten- und Befehlspfad). Die Pipeline dieser CPUs besitzt fünf einzelne Stufen.

2.2.2 ARM-Register

Das Programmiermodell für eine ARM-CPU enthält 37 interne (für den Programmierer sichtbare) 32-bit-Register. Die Abbildung 2.4 liefert einen ersten Überblick. 31 dieser 37 Register sind als Universal-Register (*General Purpose Register*) nutzbar. Darin ist der Programmzähler (PC = *Program Counter*) enthalten, über welchen der nächste auszuführende Befehl im Speicher adressiert wird. Sechs weitere 32-bit-Register dienen als Statusregister (CPSR = *Current Program Status Register*). In diesen Registern werden aber nur jeweils 12 einzelne Bits benutzt. Alle anderen Bits eines CPSR sollten als reserviert betrachtet werden und ungenutzt bleiben. Die Abbildung 2.5 zeigt die Zuordnung der Bits in einem ARM-Statusregister.

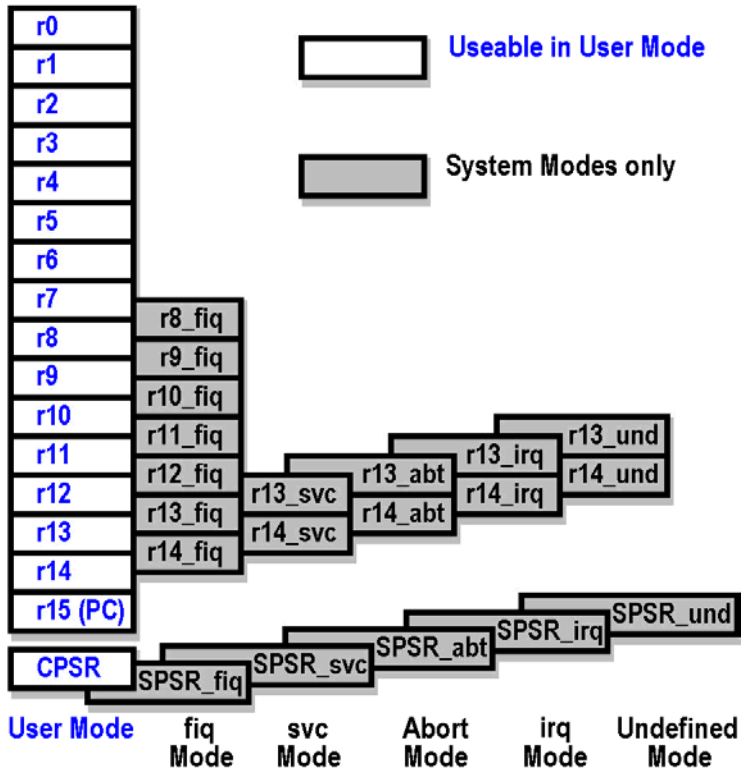

Abb. 2.4: Die Register einer ARM-CPU MpA0147.tif

Die ARM-Universal-Register tragen die Namen R0 bis R15. R15 ist der PC (Program Counter). Die Register R0 bis R14 lassen sich weiterhin in zwei Gruppen unterteilen: 1. die Gruppe der *Unbanked Registers* R0 bis R7 und 2. die Gruppe der *Banked Registers* R8 bis R14. Frei übersetzt bedeutet das: die Register R0 bis R7 (Unbanked Registers) haben in verschiedenen Betriebsarten einer ARM-CPU stets die gleiche Funktion. Die Funktion der Register R8 bis R14 (Banked Registers) hängt von der jeweiligen Betriebsart ab.

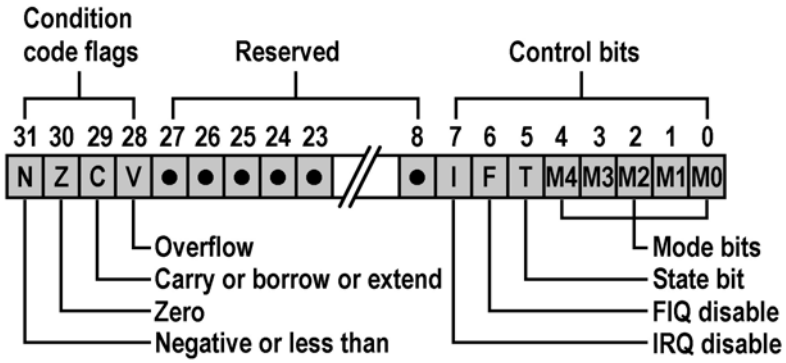

Abb. 2.5: Die Bits im ARM-CPSR (Current Program Status Register) MpA0148.tif

Eine ARM-CPU kennt bei der Programmausführung sieben verschiedene Zustände bzw. Betriebsarten: 1. *User Mode*, 2. *System Mode*, 3. *Supervisor Mode* (SVC), 4. *Abort Mode* (ABT), 5. *Undefined Mode* (UND), 6. *Interrupt Mode* (IRQ) und 7. *Fast Interrupt Mode* (FIQ). Die Betriebsarten 2 bis 7 in dieser Aufzählung gelten laut den ARM-Architekturbeschreibungen als privilegierte Betriebsarten. 3 bis 7 sind weiterhin sogenannte *Exception Modes* oder auch *System Modes*. Sie sind die Folge bestimmter

Ereignisse, zum Beispiel das Auftreten eines Interrupts. Für sechs dieser sieben Betriebarten ist jeweils ein eigenständiges CPSR mit den 12 Bits aus der Abbildung 2.5 vorhanden. So ist das *CPSR_fiq* in der Abbildung 2.4 zum Beispiel dem Fast Interrupt Mode (FIQ), *CPSR_svc* dem Supervisor Mode (SVC) zugeordnet usw.

2.2.3 ARM-Befehlssatz

Alle ARM-CPUs besitzen einen 32-bit-Befehlssatz. Bei den Architekturvarianten mit Thumb-Befehlssatz (zum Beispiel ARM7TDMI oder ARM9TDMI – der Buchstabe **T** im Namen zeigt die 16-bit-Thumb-Option an) ist zusätzlich noch ein 16-bit-Befehlssatz vorhanden. Es kann per Software zwischen beiden Befehlssätzen umgeschaltet werden. Eine ausführliche Beschreibung der ARM-Befehle würde den Rahmen dieses Buches bei weitem sprengen. Für die Assembler-Programmierung einer ARM-CPU und zur Klärung weiterer wichtiger Details (zum Beispiel: wann wird welches Bit in einem CPSR gesetzt) sollte in jedem Fall das *ARM Architecture Reference Manual* als Informationsquelle verwendet werden. Dieses Dokument mit über 800 Seiten steht als PDF-Datei über die Website der ARM Limited zur Verfügung (Abbildung 2.7).

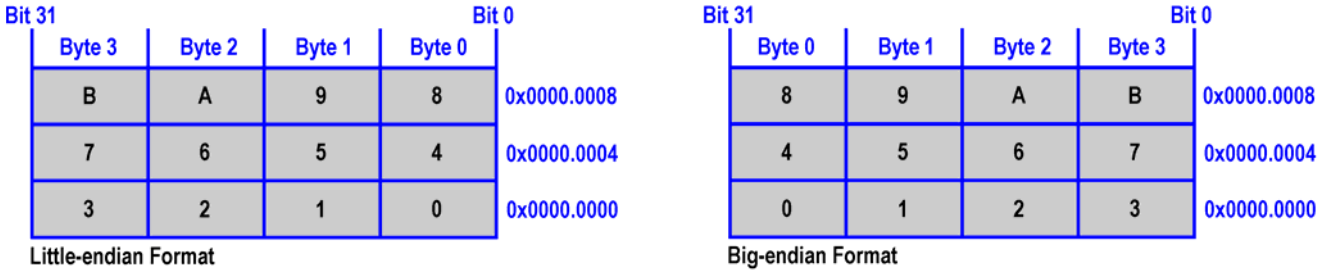

Abb. 2.6: ARM unterstützt Little-endian- und Big-endian-Formate MpA0150.tif

ARM-Prozessoren sind sogenannte *Load-and-Store-Architekturen*. Bei solchen CPUs beziehen sich praktisch alle Datenverarbeitungsbefehle auf die CPU-internen Register. Mit anderen Worten: werden zum Beispiel zwei Operanten miteinander verknüpft, so müssen diese vor der Befehlsausführung in den CPU-Registern vorliegen. Das Ergebnis wird grundsätzlich wieder in einem CPU-internen Register abgelegt. Das Load-and-Store-Konzept spart sehr viele Speicherzugriffe und beschleunigt somit die Programmausführung. Nur bei Bedarf wird der Inhalt eines CPU-Registers über einen *Store-Befehl* in den Speicher transferiert bzw. über einen *Load-Befehl* aus dem Speicher geladen. Die Load-and-Store-Technik ist ein typisches RISC-Konzept.

Die gesamte ARM-Speicherorganisation basiert ebenfalls auf einer 32-bit-Struktur. Ein einziger Load- oder Store-Befehl kann jeweils ein 32-bit-Wort zwischen einem CPU-Register und dem Speicher hin und her transferieren (im ARM-Thumb-Modus bewegen die Load- und Store-Befehle jeweils 16-bit-Wörter). Die CPU sieht den Speicher als Folge von Bytes, die als 32-bit-Worte (Thumb-Modus: 16-bit-Wörter) organisiert sind. ARM unterstützt grundsätzlich die *Little-endian-* und *Big-endian-Speicherorganisation* (Abbildung 2.6). Im ersten Fall (*Little-endian*) beinhaltet die niederwertige Byte-Adresse eines Speichers jeweils das niederwertige Byte eines Wortes. Beim Big-endian-Format ist es genau umgekehrt: Das höherwertige Byte eines Wortes ist in der niederwertige Byte-Adresse gespeichert. Bei vielen ARM-MCUs und SoCs kann zwischen Little-endian- und Big-endian-Format umgeschaltet werden. Einige ARM-basierte Varianten unterstützen allerdings nur das Little-endian-Format. Diese Speicherorganisationsform sollte daher für ARM-Anwendungen als Favorit betrachtet werden.

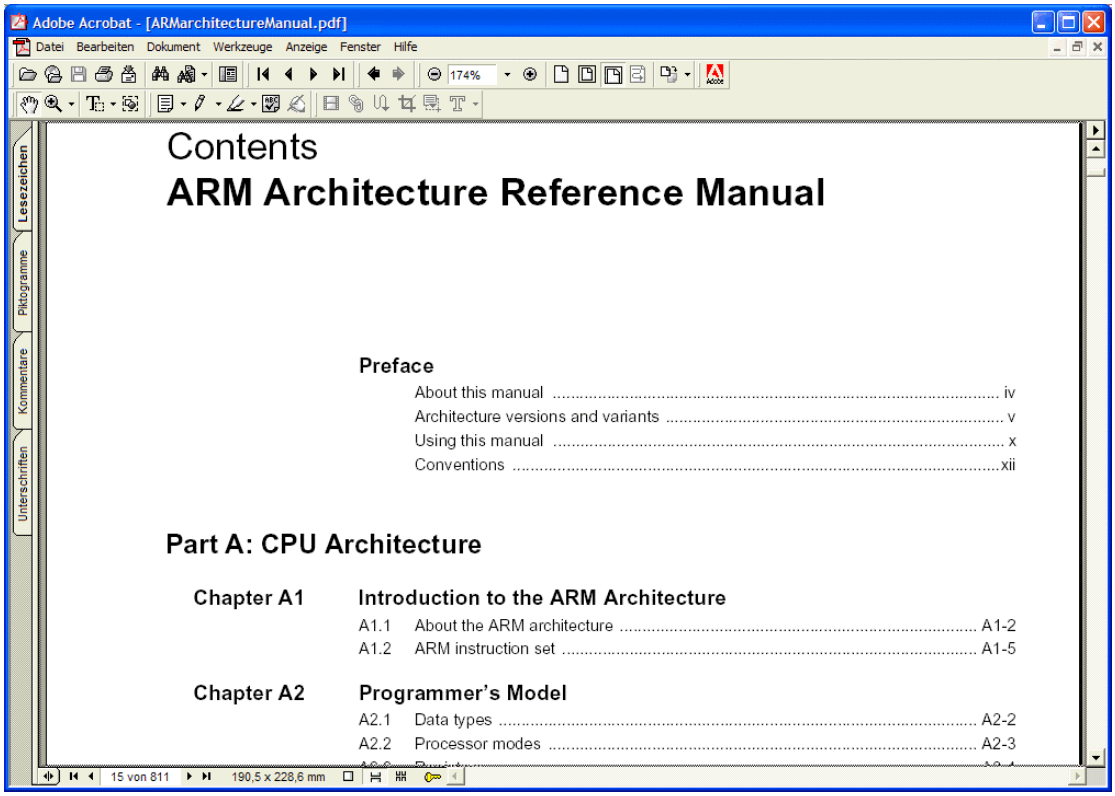

Abb. 2.7: Wichtige Details findet man im ARM Architecture Reference Manual MpA0149.tif

Für die ARM-C-Programmierung sind Detailkenntnisse des Befehlssatzes nicht unbedingt von Bedeutung. Eine Ausnahme bildet häufig das Programmieren eines Interrupt-Treibers für eine MSR-Anwendung. Solche Softwarekomponenten werden in der Regel in Assembler erstellt. Hier führt dann kein Weg am Studium des ARM-Befehlssatzes und dem bereits angesprochenen ARM Architecture Reference Manual vorbei. Wenn möglich, sollte ein Beispielcode als Vorlage verwendet werden. Zum Teil reicht dann schon ein einfaches Grundverständnis der ARM-Befehle, um die Vorlage an die eigenen Bedürfnisse anzupassen.

2.3 Aufbau von ARM-MCUs und SoCs

ARM-basierte Mikrocontroller- (MCU-) und System-on-Chip- (SoC-) Bausteine besitzen im Großen und Ganzen die im Kapitel 1 beschriebenen Strukturen typischer MCUs und SoCs. Das wichtigste Unterscheidungsmerkmal ist, das sie grundsätzlich über einen 32-bit-ARM-CPU-Core verfügen. Hinsichtlich der Peripheriefunktionen sind die Unterschiede zu anderen MCU- und SoC-Familien nur in den Implementierungsdetails zu finden. Bestimmte Funktionen, wie zum Beispiel die 16450- oder 16550-kompatible UART-Komponente (UART = Universal Asynchronous Receiver Transmitter = serielle Schnittstelle für die asynchrone Datenübertragung), findet man sowohl in vielen ARM-basierten Bausteinen als auch in anderen Mikrocontrollern. Die Chip-Entwickler benutzen teilweise die gleichen Peripherie-IPs (Intellectual Property) und sogar die gleichen IP-Quellen. Die Abbildung 2.8 zeigt die Blockschaltung eines typischen Mikrocontrollers bzw. SoCs auf ARM-Basis.

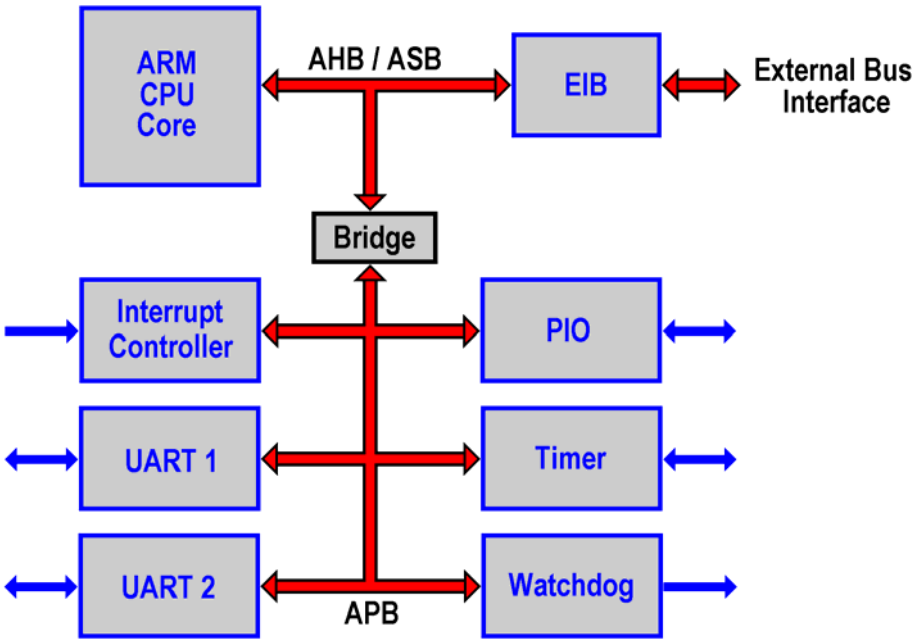

Abb. 2.8: Blockschaltung eines ARM-basierten SoC MpA0048.tif

In diesem Beispiel ist ein ARM-CPU-Core über einen internen SoC-Bus mit einer EIB-Komponente (EIB = External Bus Interface), einem Interrupt-Controller, zwei UARTs für die serielle Datenübertragung, einem PIO (Parallel-I/O), einem Timer (Zeitgeber) und einem Watchdog-Zeitgeber verbunden. Interne Speicherfunktionen sind im Beispiel der Abbildung 2.8 nicht vorhanden. Die benötigten Speicherbereiche für Programm und Daten müssen extern über das EIB mit dem ARM-Beispielbaustein verbunden werden. Hinsichtlich des EIB gibt es verschiedene Varianten. Man findet auf dem Markt auch ARM-Bausteine mit internem Speicher und externem Bus-Interface. Es gibt allerdings auch ARM-basierte MCUs und SoCs, die zwar interne Speicher für Programm und Daten besitzen, denen aber das externe Bus-Interface fehlt. Zwei solcher Bausteine werden in diesem Kapitel noch vorgestellt. Ein weiteres wichtiges Unterscheidungsmerkmal ARM-basierter MCU- und SoC-Bausteine sind die internen SoC-Bussysteme (die grundsätzlichen Aufgaben solcher internen Busse werden im Kapitel 1 beschrieben). Bei der Auswahl eines ARM-Bausteins für eine bestimmte MSR-Anwendung ist unbedingt zu berücksichtigen, dass externe Peripherie-Bausteine mit Bus-Interface ebenfalls über das EIB angekoppelt werden. Mit anderen Worten: eine ARM-MCU ohne externes Bus-Interface ist auch nicht über busfähige Bausteine (zum Beispiel durch einen Ethernet-LAN-Controller) erweiterbar.

Name	Aufgabe
AHB	Advanced High Performance Bus. Schneller SoC-Bus für ARM-Bausteine.
APB	Advanced Peripheral Bus. Langsamer Peripherie-Bus für ARM-Bausteine.
ASB	Advanced System Bus. Schneller SoC-Bus für ARM-Bausteine.

Tabelle 2.4: SoC-Bussysteme gemäß den ARM-AMBA-Spezifikationen

Die SoC-Bussysteme für ARM-MCUs und SoCs sind in den AMBA-Spezifikationen (AMBA = Advanced Microcontroller Bus Architecture) der ARM Limited beschrieben. Die Tabelle 2.4 liefert einen Überblick zu den wichtigsten Buskonzepten dieser Entwicklervorgaben. Man unterscheidet grundsätzlich zwischen schnellen synchronen Bussen, welche zum Beispiel für Speicher- und externe

Busschnittstellen dienen, und langsameren asynchronen Peripherie-Bussen. Zur Kopplung einzelner Busse dienen Bus-Brücken (Bridges). In der Abbildung 2.2 ist die schnelle AHB/ASB-Busverbindung über eine solche Brücke mit dem APB verbunden.

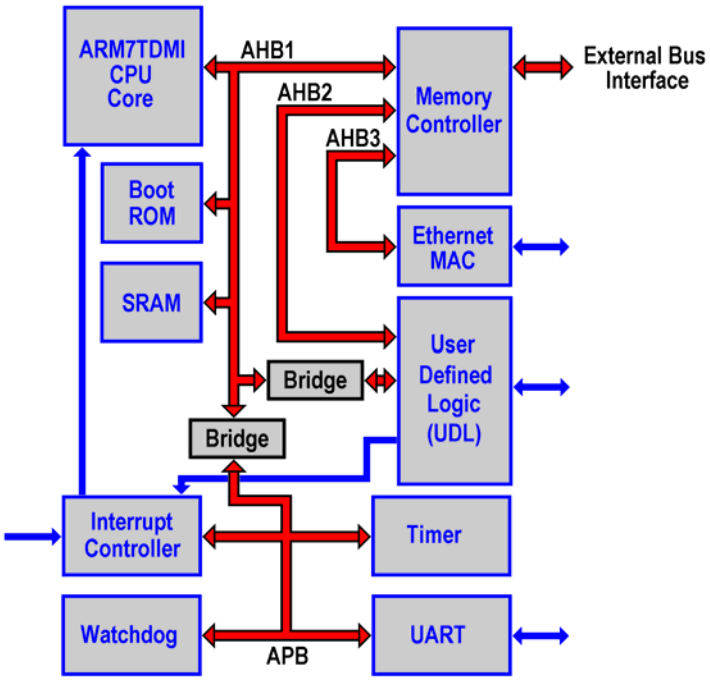

Abb. 2.9: SoCLite+, ein ARM7TDMI mit Gate Array MpA0073.tif

Es gibt auch ARM-basierte Bausteine mit mehrere Brücken. Die Abbildung 2.9 zeigt die Blockschaltung eines SoC-Bausteins mit dem Namen *SoCLite+* der Firma NEC. Dieser Baustein basiert auf einem ARM7TDMI-Core. SoCLite+ besitzt mehrere AHB-Bussysteme. *AHB1* dient in der Abbildung 2.9 als Verbindung zwischen der CPU, dem Memory Controller, dem Boot ROM, dem SRAM sowie – über zwei voneinander unabhängige Brücken – zu einem Funktionsblock mit benutzerspezifischer Logik und dem Peripherie-Bus APB. *AHB2* bildet einen schnellen Verbindungsbus zwischen dem Memory Controller und dem Funktionsblock mit der benutzerspezifischen Logik. Der dritte Hochgeschwindigkeitsbus *AHB3* koppelt den SoCLite+ Memory Controller mit der MAC-Funktionseinheit eines internen Ethernet-LAN-Controllers. Er ermöglicht schnelle DMA-Operationen (DMA = Direct Memory Access) zwischen dem LAN-Controller und externen Speicherbausteinen. Der Funktionsblock mit der benutzerspezifischen Logik ist beim NEC-SoCLite+ als *Gate Array* ausgeführt. Die Funktionen dieses Blocks werden durch Benutzerlogik in einem relativ preisgünstigen Produktionsschritt in der Halbleiterfertigung fest verdrahtet. Standardmäßig besitzt der SoCLite+ nur sehr wenige Peripheriefunktionen. NEC geht davon aus, dass der Kunde sich die tatsächlich benötigten Funktionen mit Hilfe der Gate Array-Logik definiert.

2.3.1 Speicheraufteilung (Memory Map)

Die ARM-Speicherorganisation basiert – wie bereits angesprochen – auf einer 32-bit-Struktur. Das gilt auch für die Adressierung. Dadurch ist ein Adressraum von insgesamt 4 GByte möglich. Die einzelnen Bytes in diesem Adressraum werden über eine 32-bit-Adresse abgesprochen. Schreib- und Lesezugriffe erfolgen allerdings auf der Basis von 32-bit- oder 16-bit-Worten (32-bit-ARM- oder 16-

bit-Thumb-Modus). Mit einer 32-bit-Adresse ist ein Adressraum von 0x0000.0000 bis 0xFFFF.FFFF möglich.

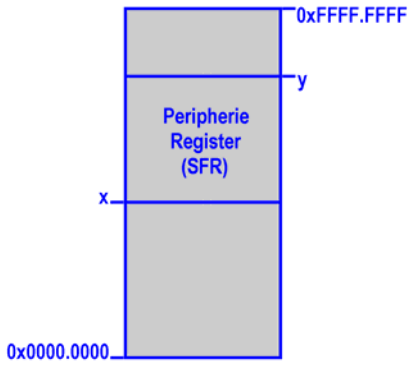

Abb. 2.10: Eine ARM-CPU besitzt einen 4 GByte-Adressraum MpA0154.tif

Die Abbildung 2.10 zeigt den möglichen Adressraum (*Memory Map*) einer fiktiven ARM-MCU. Innerhalb dieser Memory Map findet man einen Bereich mit den Adressen der Peripheriekomponenten (*Peripherie Register* oder auch *SFR* = *Special Function Registers* = Sammelbegriff für die speziellen Register eines Mikrocontrollers, über die bestimmte Funktionen zur Verfügung stehen). In der Abbildung 2.10 belegt dieser Adressbereich die *Adresse x* bis *Adresse y*. Die jeweiligen Details (Adressbereiche, Adressen und Strukturen der Register usw.) variieren von MCU zu MCU. Sie müssen dem jeweiligen Benutzerhandbuch entnommen werden.

```
#define AT91C_TTGR    0xFFFC0028  // Transmitter Time-guard Register
#define AT91C_BRGR    0xFFFC0020  // Baud Rate Generator Register
#define AT91C_RHR     0xFFFC0018  // Receiver Holding Register
#define AT91C_IMR     0xFFFC0010  // Interrupt Mask Register

#define AT91C_IER     0xFFFC0008  // Interrupt Enable Register
#define AT91C_CR      0xFFFC0000  // Control Register
#define AT91C_RTOR    0xFFFC0024  // Receiver Time-out Register
#define AT91C_THR     0xFFFC001C  // Transmitter Holding Register
#define AT91C_CSR     0xFFFC0014  // Channel Status Register
#define AT91C_IDR     0xFFFC000C  // Interrupt Disable Register
#define AT91C_MR      0xFFFC0004  // Mode Register
```

Listing 2.1: Auszug aus den SFR-Registeradressen einer Atmel-AT91M42800A-MCU

Das Listing 2.1 zeigt einen kleinen Auszug aus den Adressen der Peripheriefunktionen für eine Atmel-AT91M42800A-ARM-MCU. Es handelt sich in diesem Beispiel um die Register eines UARTs. Jedes einzelne UART-Register besitzt eine 32-bit-Adresse. Die Funktionsbits in einem solchen Register belegen in der Regel nicht alle 32 möglichen Bits. Es kann durchaus sein, dass ein bestimmtes SFR nur die acht Bits eines einzelnen Bytes in einem solchen 32-bit-Wort benutzt.

Exception	Modus	Vektoradresse
Reset	SVC	0x0000.0000
Undefinierter Befehl	UND	0x0000.0004
Software-Interrupt	SVC	0x0000.0008

Prefetch Abort (Speicherfehler beim Befehlszugriff)	ABT	0x0000.000C
Data Abort (Speicherfehler beim Datenzugriff)	ABT	0x0000.0010
IRQ (Normaler Interrupt)	IRQ	0x0000.0018
FIQ (Schneller Interrupt)	FIQ	0x0000.001c

Tabelle 2.5: Speicheradressen der Exception-Vektoren einer ARM-CPU

Der gesamte 32-bit-Adressraum einer ARM-CPU steht für die Nutzung durch Speicher- und Peripherieadressen zur freien Verfügung. Die einzigen festen Vorgaben sind die Adressen der *Exception-Vektoren* ab der Adresse 0x0000.0000 (Exceptions = Ausnahmezustände). Die Tabelle 2.5 zeigt die Zuordnung der wichtigsten Adressen. Zu diesen Vektoradressen verzweigt eine ARM-CPU immer beim Auftreten des jeweiligen Ausnahmezustands. Ein typisches Beispiel ist der CPU-Reset: Nach einem solchen Ereignis wird die weitere Programmausführung über die Adresse 0x0000.0000 festgelegt. Hier kann zum Beispiel der Sprungbefehl in das eigentliche Programm stehen. Gleiches gilt für das Einbinden von Interrupt-Treiberprogrammen.

Die festen Adressen der Vektoren aus der Tabelle 2.5 bewirken, dass ein ARM-basiertes System ab der Adresse 0x0000.0000 stets Programmspeicher besitzen muss. Bei vielen MCUs und SoCs mit internen Speicherfunktionen ist der Flash-Speicherbereich ab dieser Adresse im Adressraum zu finden.

2.3.2 5V-Toleranz

Viele ARM-MCUs und SoCs arbeiten mit 3,3 VDC als I/O-Spannung. Die Spannung an den Eingängen darf diesen Wert nur unwesentlich übersteigen. Dadurch ergibt sich in vielen Anwendungen ein Problem mit externer 5 V-Logik. Eingänge einer 3,3 VDC-MCU sind nicht immer 5 V-tolerant. Einzelheiten sollten über das Datenblatt der jeweiligen MCU geklärt werden. Zur Lösung dieses Problems sind externe Schaltkreise erforderlich. Die Abbildung 2.11 zeigt einen Schaltungsvorschlag. Zwischen der ARM-MCU im linken Teil der Abbildung und einem 5 V-Logikbaustein im rechten Abbildungsteil wird ein Baustein aus der Philips-AHC-Familie (zum Beispiel der *74AHC244*) geschaltet.

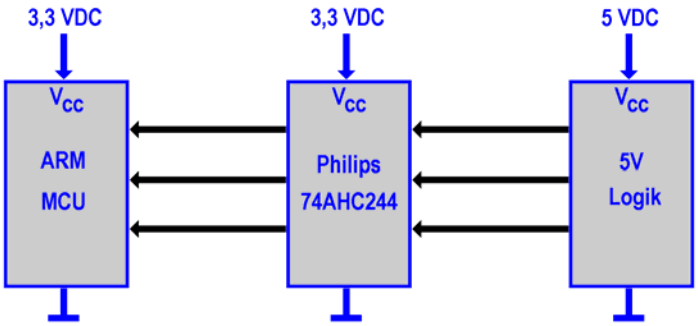

Abb. 2.11: Ein Baustein aus der Philips-AHC-Familie als Pegelwandler MpA0053.tif

Die ARM-MCU und der Philips-Baustein in der Abbildung 2.11 werden jeweils mit 3,3 VDC versorgt. Im Gegensatz zu einer nicht 5V-toleranten ARM-MCU darf den Eingängen eines Philips-AHC-Bausteins die Ausgangsspannung einer 5 V-Logik zugeführt werden, ohne dass diese Bausteine beschädigt werden. Laut dem Philips-Datenblatt kann ein 74AHC244 mit einer Versorgungsspannung von 2,0 bis 5,5 VDC betrieben werden. An den Eingängen sind zwischen 0 und 5,5 VDC erlaubt.

3. Schaltungsbeschreibung DNP/7520

Die Abbildung 3.1 zeigt die Blockschaltung des DIL/NetPCs DNP/7520 (Linux Control 1). Die zentrale Funktionseinheit dieses Moduls bildet ein 32-bit ARM7TDMI-Mikrocontroller mit der Bezeichnung *NS7520*. Dieser Baustein wird auf dem DNP/7520 mit einer Taktgeschwindigkeit von 55 MHz betrieben.

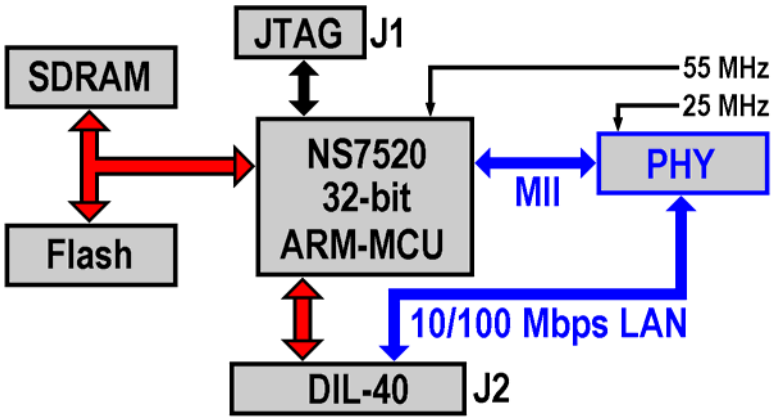

Abb. 3.1: Blockschaltung des Linux Control DIL/NetPC DNP/7520 7520-blk1.tif

Der NS7520 ist ein hochkomplexer ARM-SoC-Baustein des US-Herstellers *NetSilicon*. Er beinhaltet – neben dem ARM7TDMI-Prozessor-Core – ein vielfältig nutzbares Bus-Interface, einen DMA-Controller, zwei universelle UART-/HDLC- und SPI-Funktionseinheiten, einen 10/100 Mbps-Ethernet-MAC-Controller, vielseitig nutzbare GPIO- und Timer-Funktionen sowie einen Interrupt-Controller. Mechanisch ist der NS7520 in einem kleinen BGA-Gehäuse mit 13 mm x 13 mm Kantenlänge und insgesamt 177 Pin untergebracht.

Pin	Name	Funktion	Gruppe
1	PA0	Parallel I/O, Port A, Bit 0	GPIO
2	PA1	Parallel I/O, Port A, Bit 1	GPIO
3	PA2	Parallel I/O, Port A, Bit 2	GPIO
4	PA3	Parallel I/O, Port A, Bit 3	GPIO
5	PA4	Parallel I/O, Port A, Bit 4	GPIO
6	PA5	Parallel I/O, Port A, Bit 5	GPIO
7	PA6	Parallel I/O, Port A, Bit 6	GPIO
8	PA7	Parallel I/O, Port A, Bit 7	GPIO
9	SD0	Expansion Bus, Data Bit 0	GPE
10	SD1	Expansion Bus, Data Bit 1	GPE
11	SD2	Expansion Bus, Data Bit 2	GPE
12	SD3	Expansion Bus, Data Bit 3	GPE
13	SD5	Expansion Bus, Data Bit 4	GPE
14	SD5	Expansion Bus, Data Bit 5	GPE
15	SD6	Expansion Bus, Data Bit 6	GPE

16	SD7	Expansion Bus, Data Bit 7	GPE
17	RESIN#	Reset Input (Low Active)	---
18	CS1#	Expansion Bus, Chip Select Output 1 (Low Active)	GPE
19	CS2#	Expansion Bus, Chip Select Output 2 (Low Active)	GPE
20	GND	Ground	---
21	RCM	RCM (Remote Console Mode) Input	GPIO
22	TX+	10/100 Mbps Ethernet LAN Interface, TX+ Pin	LAN
23	TX-	10/100 Mbps Ethernet LAN Interface, TX- Pin	LAN
24	RX+	10/100 Mbps Ethernet LAN Interface, RX+ Pin	LAN
25	RX-	10/100 Mbps Ethernet LAN Interface, RX- Pin	LAN
26	TXD2	COM2 Serial Port, TXD Output Pin	SIO
27	RXD2	COM2 Serial Port, RXD Input Pin	SIO
28	NC	Not Connected	SIO
29	VccOUT	3.3 VDC Output	SIO
30	DSR1	COM1 Serial Port, DSR Input Pin	SIO
31	DCD1	COM1 Serial Port, DCD Input Pin	SIO
32	RTS1	COM1 Serial Port, RTS Output Pin	SIO
33	CTS1	COM1 Serial Port, CTS Input Pin	SIO
34	TXD1	COM1 Serial Port, TXD Output Pin	SIO
35	RXD1	COM1 Serial Port, RXD Input Pin	SIO
36	SA0	Expansion Bus, Address Bit 0	GPE
37	SA1	Expansion Bus, Address Bit 1	GPE
38	WR#	Expansion Bus, Write Signal (Low Active)	GPE
39	RD#	Expansion Bus, Read Signal (Low Active)	GPE
40	Vcc	3.3 VDC Power Input	---

Tabelle 3.1: DIL-40-Pinbelegung des Linux Control DIL/NetPC DNP/7520

Die Speicherausstattung des DIL/NetPC DNP/7520 besteht aus je einem 16 MByte-SDRAM- und 8 MByte-Flash-Speicherbaustein. Das SDRAM dient dem Embedded-Linux-Betriebssystem als Arbeitsspeicher und RAM-Disk. Der nichtflüchtige Flash-Speicherbereich beherbergt den Boot Loader (U-Boot) und das Linux-Root-Dateisystem auf Basis eines JFFS2 (Journaling Flash File System, Version 2).

Der interne 10/100 Mbps-Ethernet-MAC-Controller des NS7520 ist in der Schaltung des DNP/7520 mit einem externen PHY-Baustein verbunden. MAC und PHY zusammen bilden das DIL/NetPC DNP/7520-Ethernet-LAN-Interface.

Für Hardware-Test- und Software-Debugging-Aufgaben verfügt der NS7520 über das ARM-typische EmbeddedICE-Interface. Diese JTAG-konforme Schnittstelle ist auf der Leiterplattenoberfläche des DIL/NetPC DNP/7520 über einen kleinen Spezialsteckverbinder zugänglich (J1 in der Blockschaltung, siehe Abbildung 3.1).

Zur Schaltungserweiterung und für alle anderen Verbindungen dient der 40-polige Steckverbinder J2. Über diesen Stecker erfolgt zugleich die Spannungsversorgung des DIL/NetPC DNP/7520 mit einer 3,3 VDC-Versorgungsspannung. Auch die Ethernet-LAN-Signale des DIL/NetPC DNP/7520 sind auf J2 geführt.

Tabelle 3.1 liefert eine vollständige Auflistung aller 40 Pins. Die Einteilung der Signale in Gruppen innerhalb der Tabelle erfolgt aus Kompatibilitätsgründen. Andere Module aus der DIL/NetPC-Familie mit DIL-40-Pinout sind bei Beachtung der entsprechenden Applikationshinweise vollständig oder bedingt pinkompatibel zum DIL/NetPC DNP/7520.

4. Schaltungsbeschreibung DNP/9200

Die Abbildung 4.1 zeigt die Blockschaltung des DIL/NetPCs DNP/9200 (Linux Control 2). Die zentrale Funktionseinheit dieses Moduls bildet ein sehr leistungsfähiger 32-bit ARM9-Mikrocontroller mit der Bezeichnung *AT91RM9200*. Dieser Baustein wird auf dem DNP/9200 mit einer Taktgeschwindigkeit von 180 MHz betrieben.

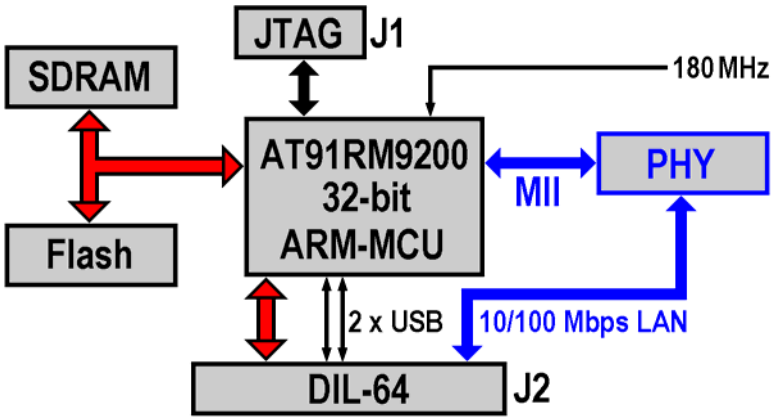

Abb. 4.1: Blockschaltung des Linux Control DIL/NetPC DNP/9200 9200-blk1.tif

Der AT91RM9200 ist ein hochkomplexer ARM-SoC-Baustein des US-Herstellers *Atmel*. Er besteht aus einem ARM9-Prozessor-Core, Bus-Interface, DMA-Controller, verschiedenen UART- und SPI-Funktionseinheiten, einen 10/100 Mbps-Ethernet-MAC-Controller, vielseitig nutzbare GPIO- und Timer-Funktionen, Interrupt-Controller, je einen USB-Host- und Device-Port, ein MultimediaCard-(MMC) Interface und zahlreiche weitere ARM-SoC-typische Funktionseinheiten. Mechanisch ist der AT91RM9200 in einem kleinen BGA-Gehäuse mit 15 mm x 15 mm Kantenlänge und insgesamt 256 Pin untergebracht.

Pin	Name	Funktion	Gruppe
1	PA0	Parallel I/O, Port A, Bit 0 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
2	PA1	Parallel I/O, Port A, Bit 1 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
3	PA2	Parallel I/O, Port A, Bit 2 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
4	PA3	Parallel I/O, Port A, Bit 3 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
5	PA4	Parallel I/O, Port A, Bit 4 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
6	PA5	Parallel I/O, Port A, Bit 5 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
7	PA6	Parallel I/O, Port A, Bit 6	PIO
8	PA7	Parallel I/O, Port A, Bit 7	PIO
9	PB0	Parallel I/O, Port B, Bit 0 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
10	PB1	Parallel I/O, Port B, Bit 1 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
11	PB2	Parallel I/O, Port B, Bit 2 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
12	PB3	Parallel I/O, Port B, Bit 3 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
13	PB4	Parallel I/O, Port B, Bit 4 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
14	PB5	Parallel I/O, Port B, Bit 5 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
15	PB6	Parallel I/O, Port B, Bit 6 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO

16	PB7	Parallel I/O, Port B, Bit 7 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
17	PC0	Parallel I/O, Port C, Bit 0 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
18	PC1	Parallel I/O, Port C, Bit 1 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
19	PC2	Parallel I/O, Port C, Bit 2 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
20	PC3	Parallel I/O, Port C, Bit 3 (siehe auch: Alternative Funktionen – Tab. 4.2)	PIO
21	RXD1	COM1 Serial Port, RXD Pin	SIO
22	TXD1	COM1 Serial Port, TXD Pin	SIO
23	CTS1	COM1 Serial Port, CTS Pin	SIO
24	RTS1	COM1 Serial Port, RTS Pin	SIO
25	DCD1	COM1 Serial Port, DCD Pin	SIO
26	DSR1	COM1 Serial Port, DSR Pin	SIO
27	DTR1	COM1 Serial Port, DTR Pin	SIO
28	RI1	COM1 Serial Port, RI Pin	SIO
29	RESIN	RESET Input	RESET
30	TX+	10BASE-T/100BASE-TX Ethernet Interface, TX+ Pin	LAN
31	TX-	10BASE-T/100BASE-TX Ethernet Interface, TX- Pin	LAN
32	GND	Ground	---
33	RX+	10BASE-T/100BASE-TX Ethernet Interface, RX+ Pin	LAN
34	RX-	10BASE-T/100BASE-TX Ethernet Interface, RX- Pin	LAN
35	RESOUT	RESET Output	RESET
36	VBAT	Real Time Clock Battery Input	PSP
37	CLKOUT	Clock Output	PSP
38	TXD2	COM2 Serial Port, TXD Pin	PSP
39	RXD2	COM2 Serial Port, RXD Pin	PSP
40	HDMA	USB Host Port- (Interrupt Input on other DIL/NetPCs)	PSP
41	HDPA	USB Host Port+ (Interrupt Input on other DIL/NetPCs)	PSP
42	DDM	USB Device Port- (Interrupt Input on other DIL/NetPCs)	PSP
43	DDP	USB Device Port+ (Interrupt Input on other DIL/NetPCs)	PSP
44	INT1	Interrupt Input 1	PSP
45	CS4	Chip Select Output 4. Physical Addr. Range 0x7000.0000-0x7FFF.FFFF	PSP
46	CS3	Chip Select Output 3. Physical Addr. Range 0x6000.0000-0x6FFF.FFFF	PSP
47	CS2	Chip Select Output 2. Physical Addr. Range 0x5000.0000-0x5FFF.FFFF	PSP
48	CS1	Chip Select Output 1. Physical Addr. Range 0x4000.0000-0x4FFF.FFFF	PSP
49	RDY	External Ready Input	PSP
50	RD#	Read Signal, Expansion Bus (Low active)	PSP
51	WR#	Write Signal, Expansion Bus (Low active)	PSP
52	SA3	Expansion Bus, Address Bit 3	PSP
53	SA2	Expansion Bus, Address Bit 2	PSP
54	SA1	Expansion Bus, Address Bit 1	PSP

55	SA0	Expansion Bus, Address Bit 0	PSP
56	SD7	Expansion Bus, Data Bit 7	PSP
57	SD6	Expansion Bus, Data Bit 6	PSP
58	SD5	Expansion Bus, Data Bit 5	PSP
59	SD4	Expansion Bus, Data Bit 4	PSP
60	SD3	Expansion Bus, Data Bit 3	PSP
61	SD2	Expansion Bus, Data Bit 2	PSP
62	SD1	Expansion Bus, Data Bit 1	PSP
63	SD0	Expansion Bus, Data Bit 0	PSP
64	Vcc	3.3 Volt Power Input	---

Tabelle 4.1: DIL-64-Pinbelegung des Linux Control DIL/NetPC DNP/9200

Die Speicherausstattung des DIL/NetPC DNP/9200 besteht aus je einem 32 MByte-SDRAM- und 16 MByte-Flash-Speicherbaustein. Das SDRAM dient dem Embedded-Linux-Betriebssystem als Arbeitsspeicher und RAM-Disk. Die SRAM-Anbindung an den AT91RM9200 erfolgt mit 32-bit Wortbreite. Der nichtflüchtige Flash-Speicherbereich beherbergt den Boot Loader (U-Boot) und das Linux-Root-Dateisystem auf Basis eines JFFS2 (Journaling Flash File System, Version 2). Für die Verbindung zwischen Flash und AT91RM9200-MCU wird ein 16-bit Datenpfad benutzt.

Der interne 10/100 Mbps-Ethernet-MAC-Controller des AT91RM9200 ist in der Schaltung des DNP/9200 mit einem externen PHY-Baustein verbunden. MAC und PHY zusammen bilden das DIL/NetPC DNP/9200-Ethernet-LAN-Interface.

Pin	Name	Alternate Funktion	Direction
1	TF1	SSI/I2S Interface: Transmitter Frame Sync	I/O
2	TK1	SSI/I2S Interface: Transmitter Clock	I/O
3	TD1	SSI/I2S Interface: Transmitter Data	Out
4	RD1	SSI/I2S Interface: Receiver Data	In
5	RK1	SSI/I2S Interface: Receiver Clock	I/O
6	RF1	SSI/I2S Interface: Receiver Frame Sync	I/O

9	MCCK	SD/MMC Interface: Multimedia Card Clock	Out
10	MCCDA	SD/MMC Interface: Multimedia Card A Command	I/O
11	MCDA0	SD/MMC Interface: Multimedia Card A Data 0	I/O
12	MCDA1	SD/MMC Interface: Multimedia Card A Data 1	I/O
13	MCDA2	SD/MMC Interface: Multimedia Card A Data 2	I/O
14	MCDA3	SD/MMC Interface: Multimedia Card A Data 3	I/O

15	SPICS2	SPI Function Pins: SPI Chip Select CS2	Out
16	SPICS3	SPI Function Pins: SPI Chip Select CS3	Out
17	MOSI	SPI Function Pins: Master Out Slave In (SPI Data Output)	Out

18	MISO	SPI Function Pins: Master In Slave Out (SPI Data Input)	In
19	SPICLK	SPI Function Pins: SPI Clock	Out
20	SPICS0	SPI Function Pins: SPI Chip Select CS0	Out

Tabelle 4.2: Alternative Funktionen am DIL-64-Steckverbinder

Für Hardware-Test- und Software-Debugging-Aufgaben verfügt der AT91RM9200 über das ARM-typische EmbeddedICE-Interface. Diese JTAG-konforme Schnittstelle ist auf der Leiterplatten-oberfläche des DIL/NetPC DNP/9200 über einen Spezialsteckverbinder zugänglich (J1 in der Block-schaltung, siehe Abbildung 4.1).

Zur Schaltungserweiterung und für alle anderen Verbindungen dient der 64-polige Steckverbinder J2. Über diesen Stecker erfolgt zugleich die Spannungsversorgung des DIL/NetPC DNP/9200 mit einer 3,3 VDC-Versorgungsspannung. Auch die Ethernet-LAN- und USB-Signale des DIL/NetPC DNP/9200 sind auf J2 geführt.

Tabelle 4.1 liefert eine vollständige Auflistung aller 64 Pins. Die Einteilung der Signale in Gruppen innerhalb der Tabelle erfolgt aus Kompatibilitätsgründen. Andere Module aus der DIL/NetPC-Familie mit DIL-64-Pinout sind bei Beachtung der entsprechenden Applikationshinweise vollständig oder bedingt pinkompatibel zum DIL/NetPC DNP/9200.

A.1 Wichtige Hinweise

Einzelne Punkte in diesem Manuskript wurden meinem Buch „**Messen, Steuern und Regeln mit ARM-Mikrocontrollern**. Franzis' 2004, 3-7723-4017-2" und „**Embedded Internet in der Industrie-automation**. Hüthig 2003, ISBN: 3-7785-2899-8" entnommen. © 2002 - 2006 Klaus-Dieter Walter. Alle Rechte vorbehalten.

Namen, die zugleich eingetragene Warenzeichen sind, wurden in diesem Manuskript nicht besonders kenntlich gemacht. Es kann also aus dem Fehlen der entsprechenden Markierung nicht geschlossen werden, dass die Bezeichnung ein freier Warenname ist. Ebenso wenig sind Rückschlüsse auf aktuelle Patente oder weitere Rechte – wie zum Beispiel ein Gebrauchsmusterschutz – möglich.

Der Autor hat alle Texte und Abbildungen mit großer Sorgfalt erarbeitet. Dennoch können Fehler nicht ausgeschlossen werden. Deshalb übernimmt der Autor keinerlei Garantien für die in diesem Manuskript gegebenen Informationen. Der Autor haftet auch in keinem Fall für irgendwelche direkten oder indirekten Schäden, die sich aus der Anwendung dieser Informationen ergeben können.

Rev.	**Datum**	**Aktion**	**Name**
0.1	30.01.2006	Erste Version aus Buchprojekten abgeleitet	KDW