

Architekturentwurf und Emulation eingebetteter Systeme

Dissertation

der Fakultät für Informatik

der Eberhard-Karls-Universität zu Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

vorgelegt von

Dipl.-Ing. Karlheinz Weiß

aus Winden

Tübingen

1999

Tag der mündlichen Qualifikation: 22. Dezember 1999

Dekan: Prof. Dr. Klaus-Jörn Lange

1. Berichterstatter: Prof. Dr. Wolfgang Rosenstiel

2. Berichterstatter: Prof. Dr. Ing. Klaus Bender

Diese Arbeit entstand an der Universität Tübingen und am Forschungszentrum Informatik (FZI) an der Universität Karlsruhe.

An erster Stelle gilt mein ganz besonderer Dank Herrn Prof. Dr. Wolfgang Rosenstiel für die hervorragende und stets motivierende Zusammenarbeit, für seine wertvollen Anregungen, Hinweise und die intensive Unterstützung der Arbeit. Herrn Prof. Dr. Klaus Bender danke ich für seine Bereitschaft, die Arbeit als 2. Berichterstatter zu unterstützen.

Für ihre fleißige und stets überdurchschnittliche Motivation bei der Implementierung und Erprobung danke ich meinen Kollegen, Diplomanden und Studenten Dipl.-Ing. Carsten Oetker, Dipl.Ing. Igor Katchan, Dipl.-Inform Carsten Nitsch, Dipl.-Ing. Ronny Kistner sowie Joachim Philipp, Hiwa Afandi, Jochen Heintz und Karl-Georg Esser.

Mein besonderer Dank gilt Dipl.-Inform. Thorsten Steckstor, dessen ausgezeichnetes Fachwissen, wertvolle Kommentare, Motivation und gute Zusammenarbeit bei der Implementierung des SPYDER-Systems über all die Jahre vorbildlich gewesen waren.

Mein herzlichster Dank gilt auch meiner Familie, insbesondere meiner Mutter Brigitte Weiß und meiner Großmutter Lydia Ulm. Sie haben mir durch ihre aufopfernde Art während den langen Jahren der Ausbildung, des Studiums und der Zeit bis zur Promotion den privaten Rückhalt und die Unterstützung gegeben, um diese Arbeit überhaupt vollbringen zu können.

Ebenfalls danken möchte ich auch meiner Lebensgefährtin Ilona Dettmar, die in der Zeit des Schreibens der Arbeit mich immer wieder motiviert hat und dabei auf viele Dinge im Privatleben verzichtet hat, um mir den nötigen Freiraum für die Arbeit zu geben.

Inhaltsverzeichnis

Kapitel 1	Einleitung	11
Kapitel 2	Komponenten eingebetteter Systeme	15
2.1	Hardware-Komponenten	15
2.1.1	Mikrocontroller	16
2.1.1.1	8-Bit-Mikrocontroller (Intel MCS51)	16
2.1.1.2	16-Bit-Mikrocontroller (Siemens C167)	18
2.1.1.3	32-Bit-Mikrocontroller (<i>Embedded PowerPC</i> 4xx, 5xx und 8xx) .	23
2.1.2	Halbleiterspeicher-Bausteine	36
2.1.2.1	Asynchrone Festwertspeicher (<i>ROM, EPROM, Flash</i>)	36
2.1.2.2	Statische Arbeitsspeicher (<i>SRAM</i> und <i>SSRAM</i>)	38
2.1.2.3	Dynamische Arbeitsspeicher (<i>DRAM</i> und <i>SDRAM</i>)	40
2.1.2.4	Zusammenfassung der Halbleiterspeicher	41
2.1.3	Programmierbare Logik-Bausteine	42
2.1.3.1	Komplexe programmierbare Logik-Bausteine	42
2.1.3.2	<i>SRAM</i> -basierte Anwender-programmierbare Schaltungen	44
2.1.3.3	Antifuse-basierte Anwender-programmierbare Schaltungen	48
2.1.3.4	Entwurfsmethoden für <i>FPGA</i>	49
2.1.4	Anwendungsspezifische integrierte Schaltungen	50
2.1.5	Technologie eingebetteter Systeme	50
2.2	Software-Komponenten	51
2.2.1	Echtzeitbetriebssystem <i>VxWorks</i>	52
2.2.1.1	<i>Tasks</i> und <i>Multitasking</i>	53
2.2.1.2	<i>Task</i> -Zustandsübergänge und <i>Scheduling</i> -Algorithmen	54
Kapitel 3	Motivation	57
3.1	Entwurfsprobleme	60
3.1.1	Eingebettete Systeme mit 4-Bit-Mikrocontrollern	61
3.1.2	Eingebettete Systeme mit 8-Bit-Mikrocontrollern	61
3.1.3	Eingebettete Systeme mit 16-Bit-Mikrocontrollern	63
3.1.4	Eingebettete Systeme mit 32-Bit-Mikrocontrollern	63
3.2	Bestehende Entwurfsmethoden	66
3.2.1	Die bisher in der Praxis eingesetzte Methode	66
3.2.1.1	Vor- und Nachteile dieser Methode	68
3.2.2	Stand der Forschung beim <i>Hardware/Software Co-Design</i>	69
3.2.2.1	Einheitliche Systembeschreibung	70
3.2.2.2	Hardware-Software Partitionierung	71
3.2.2.3	Diskussion der Vor- und Nachteile	72

3.3	Architekturentwurf und Emulation	74
Kapitel 4	Architekturentwurf durch Bewertung	79
4.1	Definition des Entwurfsraumes	80
4.1.1	Hardware	80
4.1.2	Software	81
4.1.3	Zusammenfassung und Begründung der Randbedingungen	82
4.2	Extrahierung des Auswahlalgorithmus	83
4.2.1	Entscheidungsfeld Funktionalität	85
4.2.2	Entscheidungsfeld Bus-Schnittstelle	86
4.2.3	Entscheidungsfeld Initialisierung	87
4.2.4	Entscheidungsfeld Technologie	87
4.2.5	Entscheidungsfeld Testbarkeit	88
4.3	Identifikation der Entscheidungsparameter	88
4.3.1	Entscheidungsparameter für die Funktionalität	89
4.3.2	Entscheidungsparameter für die Bus-Schnittstelle	91
4.3.3	Entscheidungsparameter für die Initialisierung	93
4.3.4	Entscheidungsparameter für die Technologie	95
4.3.5	Entscheidungsparameter für die Testbarkeit	98
4.4	Gesamtbewertung einer Komponente	101
Kapitel 5	Emulationsumgebung SPYDER	105
5.1	SPYDER-CORE-P1	107
5.1.1	Mikrocontroller-Kern	107
5.1.2	Schnittstellen- <i>FPGA</i>	110
5.1.3	Erweiterungs- und Logik-Analysator-Verbindungen	111
5.1.3.1	Erweiterung für <i>ASIC</i> -Komponenten	112
5.1.3.2	Erweiterung für <i>Co-Prozessoren</i>	114
5.1.3.3	Erweiterung für einfache Peripherie-Komponenten	114
5.1.3.4	Logik-Analysator-Unterstützung	114
5.1.4	Allgemeine Peripherie-Komponenten	115
5.1.4.1	Dual-Port- <i>RAM</i> und <i>AT-ISA</i> -Bus-Schnittstelle	115
5.1.4.2	Ethernet-Schnittstelle	115
5.1.4.3	<i>Controller Area Network (CAN)</i> - Schnittstelle	116
5.1.4.4	<i>Flash</i> -Festwertspeicher	116
5.2	Software-Umgebung des SPYDER-Systems	116
5.2.1	Echtzeitbetriebssystem <i>VxWorks</i>	117
5.2.2	Software-Monitor <i>SDS70</i>	118
5.2.3	Software-Umgebung für das Schnittstellen- <i>FPGA</i>	119

5.2.4	Das SPYDER-Betriebssoftware-Paket	119
5.3	SPYDER-ASIC-X2	121
5.4	Zusammenfassung der SPYDER-Emulationsumgebung	123
5.5	Wirtschaftliche Aspekte beim Einsatz von SPYDER	124
Kapitel 6	Ergebnisse der Methodik	125
6.1	<i>ATM</i> -Diagnose-Monitor	125
6.1.1	Allgemeine Funktionsbeschreibung	126
6.1.2	Initiale Hardware/Software-Partitionierung	127
6.1.3	Erste Stufe: Architekturentwurf durch Bewertung	128
6.1.3.1	Auswahl einer <i>ASIC</i> -Komponente	128
6.1.3.2	Bewertung verschiedener Entwicklungsmethoden für <i>FPGA</i>	132
6.1.4	Zweite Stufe: Überprüfung des <i>ALC-ASIC</i> durch Emulation	132
6.1.4.1	<i>ASIC</i> -Emulation mit SPYDER-ASIC-X2	132
6.1.4.2	<i>FPGA</i> -Emulation mit SPYDER-CORE-P1	137
6.1.4.3	Systemintegration auf der SPYDER-CORE-P1 Basis-Platine	142
6.1.5	Zusammenfassung des Beispiels <i>ATM</i> -Diagnose-Monitor	144
6.2	Aktuator-Sensor- <i>Interface</i> -Master (<i>ASI-Master</i>)	145
6.2.1	Allgemeine Funktionsbeschreibung	145
6.2.2	Initiale Hardware/Software-Partitionierung	146
6.2.3	Emulation der Kenngrößen des Echtzeitbetriebssystems	148
6.2.3.1	Analyse der Leistungsgrenzen durch Emulation	149
6.2.3.2	Emulationsergebnisse der initialen Hardware/Software- Partitionierung	151
6.2.4	Verbesserte Hardware/Software-Partitionierung	152
6.2.4.1	Ausnutzung von <i>on-Chip</i> -Eigenschaften bei neuen <i>FPGA</i>	154
6.2.5	<i>ASI</i> -Demonstrator	155
6.2.6	Zusammenfassung des Anwendungsbeispiels <i>ASI</i>	157
6.3	Vergleich mit der in der Praxis eingesetzten Methode	158
6.3.1	Industrielle Erfahrungen in der Kommunikationstechnik	158
6.3.1.1	Ergebnis des Architekturentwurfes durch Bewertung	159
6.3.2	Industrielle Erfahrungen in der Automation und im Automotive-Bereich..	161
Kapitel 7	Zusammenfassung und Ausblick	163
7.1	Zusammenfassung	163
7.2	Ausblick	165
Anhang A	Literatur-Referenzen	167
Anhang B	Veröffentlichungen, die im Rahmen dieser Arbeit entstanden sind	171

Abbildungsverzeichnis

Abbildung 1.1	Prognose des weltweiten Elektronik-Umsatzes (Quelle: Dataquest, Juni 1997)	12
Abbildung 2.1	Blockdiagramm des MC8051-Kerns (Quelle: [Intel94]).....	17
Abbildung 2.2	Blockdiagramm des SIEMENS C167 (Quelle: [Sie96])	19
Abbildung 2.3	Organisation des Befehls-Cache beim <i>MPC505</i> (Quelle: [Mot94]).....	28
Abbildung 2.4	Befehls-Warteschlange beim IBM <i>PowerPC PPC403</i> (Quelle: [IBM98]).....	30
Abbildung 2.5	Blockdiagramm des <i>PowerPC PPC403GCX</i> (Quelle: [IBM98]).....	33
Abbildung 2.6	Asynchroner Lesezugriff auf einen externen Festwertspeicher	37
Abbildung 2.7	Asyn. Schreibzugriff auf einen externen <i>SRAM</i> -Arbeitsspeicher (Quelle: [Par98])	38
Abbildung 2.8	Burst-Schreibzugriff eines <i>Pipeline-Sync-SRAM</i> (Quelle: [Mic98/1]) ...	39
Abbildung 2.9	<i>DRAM-Fast-Page-Mode</i> Lesezyklus (Quelle: [Mic98/2])	40
Abbildung 2.10	Burst-Lesezugriff auf einen synchronen <i>DRAM</i> (Quelle: [Mic98/3])	41
Abbildung 2.11	Xilinx <i>XC9500 CPLD</i> -Architektur (Quelle: [Xil98/1])	43
Abbildung 2.12	<i>Single-Double-Length-Lines</i> und programmierbare <i>Switch-Matrix</i> (Quelle: [Xil98/1])	44
Abbildung 2.13	Xilinx <i>XC4000</i> konfigurierbarer Logik-Block (Quelle: [Xil98/1]).....	45
Abbildung 2.14	<i>XC6000</i> -Architektur und Basis-Zelle (Quelle: [Xil96]).....	46
Abbildung 2.15	<i>Virtex</i> -Architektur (Quelle: [Xil99]).....	47
Abbildung 2.16	Xilinx <i>2-Slice Virtex CLB</i> - vereinfachte Darstellung (Quelle: [Xil99])	47
Abbildung 2.17	<i>Antifuse</i> -Halbleitertechnologie und <i>ACT1</i> -Logik-Modul (Quelle:[Act94]).....	48
Abbildung 2.18	<i>256 Ball Grid Array</i> -Gehäuse beim <i>MPC8xx</i> (Quelle: [Mot98/1]).....	51
Abbildung 2.19	Entwicklungsumgebung für Tornado für das Echtzeitbetriebssystem <i>VxWorks</i>	52
Abbildung 2.20	<i>Task</i> -Zustands-Übergangendiagramm bei <i>VxWorks</i>	54
Abbildung 2.21	Zeitdiagramm für <i>Priority Preemption Scheduling</i> beim <i>RTOS VxWorks</i> ..	54
Abbildung 2.22	Zeitdiagramm für <i>Round Robin Scheduling</i> beim <i>RTOS VxWorks</i>	55
Abbildung 3.1	Quantitativer Anstieg der Entwurfsproblematik bei eingebetteten Systemen	60
Abbildung 3.2	Entwurfsablauf bei der bisher in der Praxis eingesetzten Methode.....	67

Abbildung 3.3	Entwurfsablaufdiagramm beim <i>Hardware/Software Co-Design</i>	69
Abbildung 3.4	Allgemeine Architektur der betrachteten Klasse von eingebetteten Systemen	75
Abbildung 3.5	Entwurfsablauf: Architekturentwurf und Emulation von eingebetteten Systemen	77
Abbildung 4.1	Architektur der betrachteten Teilklasse	80
Abbildung 4.2	<i>SDS70</i> Software-Monitor <i>Single-Step</i>	81
Abbildung 4.3	Entscheidungsgebiete bei der Komponentenauswahl	83
Abbildung 4.4	Auswahlalgorithmus für das Entscheidungsfeld Funktionalität	85
Abbildung 4.5	Auswahlalgorithmus für das Entscheidungsfeld Bus-Schnittstelle	86
Abbildung 4.6	Auswahlalgorithmus für das Entscheidungsfeld Initialisierung	87
Abbildung 4.7	Auswahlalgorithmus für das Entscheidungsfeld Technologie	88
Abbildung 4.8	Auswahlalgorithmus für das Entscheidungsfeld Testbarkeit	88
Abbildung 4.9	Bewertung der Funktionalität	89
Abbildung 4.10	Allgemeines Modell der Menge verfügbarer <i>ASIC</i>	90
Abbildung 4.11	Bewertung der Bus-Schnittstelle durch Abgleich.....	91
Abbildung 4.12	Entscheidungsparameter bei der Initialisierung.....	93
Abbildung 4.13	Anstieg der Lagenanzahl in Abhängigkeit der Pin-Anzahl am Chip-Gehäuse	95
Abbildung 4.14	Entscheidungsparameter bei der Technologie-Bewertung.....	97
Abbildung 4.15	Bewertung des Entscheidungsparameters für die Testbarkeit	98
Abbildung 4.16	Gesamter Auswahlalgorithmus für die Komponenten-Bewertung.....	101
Abbildung 5.1	Unterstützung des Entwurfsablaufes durch das Emulationssystem SPYDER	106
Abbildung 5.2	Blockdiagramm und Bild von SPYDER-CORE-P1	108
Abbildung 5.3	Wirkungskette bei der <i>ASIC</i> -Bus-Schnittstellen-Emulation mit SPYDER-CORE-P1	110
Abbildung 5.4	Unterstützung der Entscheidungsfelder durch die Emulationsumgebung SPYDER	112
Abbildung 5.5	Zusammenführung verschiedener Software-Entwicklungswerkzeuge auf der Emulationsumgebung SPYDER-CORE-P1	117
Abbildung 5.6	Blockdiagramm und Bild von SPYDER-ASIC-X2	121
Abbildung 5.7	Wirkungskette bei der Schnittstellen- bzw. Hardware-Emulation mit SPYDER-ASIC-X2	122
Abbildung 6.1	Allgemeine Funktion des <i>ATM</i> -Diagnose-Monitors	126

Abbildung 6.2	Modell der Menge verfügbarer <i>ASIC</i> , verdeutlicht am Beispiel des <i>ALC</i> für <i>ATM</i>	129
Abbildung 6.3	Blockdiagramm des Emulationswerkzeuges <i>SPYDER-ASIC-X2</i> mit <i>ALC</i> -Trägerplatine	133
Abbildung 6.4	Vereinfachte Darstellung der Wirkungskette bei der Emulation	134
Abbildung 6.5	Transmit-Datenstruktur im gemeinsamen Pufferspeicher (Quelle: [Fuj96])	136
Abbildung 6.6	Fehlerzähler zur Anwendung der <i>CTR</i> -Methode bei <i>XC4000</i> und <i>XC6000-FPGA</i>	138
Abbildung 6.7	Fehlerzähler bei der Anwendung der lokalen <i>RTR</i> -Methode bei <i>XC6000-FPGA</i>	139
Abbildung 6.8	Chip-Aufteilung bei der lokalen <i>RTR</i> -Methode auf einem <i>XC6216</i>	141
Abbildung 6.9	Systemintegration mit <i>SPYDER-CORE-P1</i>	142
Abbildung 6.10	Chip-Layout des <i>XC6216</i>	143
Abbildung 6.11	<i>ASI</i> -Kommunikationssystem	146
Abbildung 6.12	Initiale <i>ASI</i> -Schnittstellen-Hardware im <i>FPGA</i>	147
Abbildung 6.13	Initiale Software-Architektur auf dem Mikrocontroller	147
Abbildung 6.14	Ausführungszeiten auf dem Mikrocontroller (Logik-Analysator - Aufzeichnung)	148
Abbildung 6.15	Verteilung der Rechenleistung des Mikrocontrollers	149
Abbildung 6.16	<i>ISR</i> -Verschiebung von Software nach Hardware	152
Abbildung 6.17	Erweiterte Hardware-Architektur nach Auslagerung der <i>ISR</i>	153
Abbildung 6.18	Architektur des <i>ASI</i> -Demonstrators	156
Abbildung 6.19	<i>ASI</i> -Demonstrator: Modell eines Hochregallagers	156
Abbildung 6.20	Eingebettete System-Architektur der Firma Hilan	158
Abbildung 6.21	Layout des <i>Advanced A2SI-ASIC</i>	161

Tabellenverzeichnis

Tabelle 1.1:	Anwendungsbeispiele für Mikrocontroller-basierte Systeme	12
Tabelle 2.1:	Sequentielle Befehls-Pipeline beim C167-Mikrocontroller	21
Tabelle 2.2:	Unterstützte Speichertechnologien bei den <i>Embedded PowerPC</i> -Familien ...	25
Tabelle 2.3:	Cache-Größen und Eigenschaften bei den <i>PowerPC</i> -Mikrocontrollern	30
Tabelle 2.4:	Halbleiterspeicher und deren Verwendung bei eingebetteten Systemen	42
Tabelle 4.1:	Bewertung von Entscheidungsparametern für die Bus-Schnittstelle	92
Tabelle 4.2:	Bewertung der Entscheidungsparameter für die Initialisierung	94
Tabelle 4.3:	Bewertung des Entscheidungsparameters „Gehäuse“ für die Technologie	98
Tabelle 4.4:	Bewertung des Entscheidungsparameters „ <i>on-Chip</i> -Testfunktionen“	99
Tabelle 4.5:	Gesamtübersicht bei der Komponenten-Bewertung	102
Tabelle 4.6:	Zusammenfassende Wertetabelle innerhalb eines Entscheidungspfades	102
Tabelle 6.1:	Gesamtübersicht der bewerteten <i>ALC-ASIC</i> -Komponenten	131
Tabelle 6.2:	<i>FPGA</i> -Ressourcen für die <i>CTR</i> -Methode bei der <i>XC4000E/EX</i> -Familie	140
Tabelle 6.3:	Leistungssteigerungsanalyse bei <i>PowerPC403GA</i> -Mikrocontrollern	150
Tabelle 6.4:	Leistungssteigerungsanalyse bei <i>PowerPC403GCX</i> -Mikrocontrollern	151
Tabelle 6.5:	Implementierungsergebnisse auf verschiedenen <i>FPGA</i> -Architekturen	155
Tabelle 6.6:	Bewertung der <i>ASIC</i> -Komponente im Hilan-Projekt	160

Kapitel 1

Einleitung

Der rasante Fortschritt im Bereich der Halbleitertechnologie hat die Entwicklung und Produktion von mikroelektronischen Komponenten wesentlich beeinflusst. Das Gesetz von Moore besagt, daß sich die Anzahl der Transistoren pro Chip jedes Jahr verdoppelt. Der Anstieg der funktionalen Leistungsfähigkeit und Zuverlässigkeit sowie die drastische Senkung der Produktionskosten dieser Komponenten hat das bewußte bzw. unbewußte Vordringen mikroelektronischer Systeme in viele Bereiche des menschlichen Lebens bewirkt.

Von vielen Menschen wird der Fortschritt in der allgemeinen Rechnertechnik in Form von „Workstations“ oder „Personal-Computern (PCs)“ bewußt wahrgenommen. Dieser Fortschritt wird im wesentlichen geprägt durch die drastische Leistungssteigerung der verwendeten Mikroprozessoren sowie die zur Verfügung stehende Speicherkapazität. Diese Steigerung kann bei den Mikroprozessoren im wesentlichen an ihrer Taktfrequenz und an der Wortbreite abgelesen werden. Dabei ist in den letzten Jahren ein enormer Konzentrationsprozeß auf wenige Mikroprozessor-Architekturen einzelner Hersteller zu beobachten, welche in immer kürzeren Abständen schnellere Varianten einer Mikroprozessor-Architektur auf den Markt bringen. Der momentane Stand ermöglicht Taktfrequenzen zwischen 400 MHz und 600 MHz und typischen Wortbreiten von 64 Bit. In den nächsten Jahren sind Taktfrequenzen von bis zu 1 GHz angekündigt. Im Bereich der Halbleiterspeicher sind Komponenten mit 64 MBit pro Chip auf dem Markt, die vorzugsweise zu Modulen mit 64 Bit Wortbreite zusammengefaßt werden. Typische Speichermodule stellen 128 MByte bei Zugriffszeiten von 20 ns bis 50 ns zur Verfügung. Für die nächsten Jahre werden Chips mit 256 MBit angekündigt. Bei den nicht-flüchtigen Massenspeichern liegt die typische Speicherkapazität im Bereich von 10-20 GByte.

Im Gegensatz dazu verlief der Einzug mikroelektronischer Komponenten in viele Geräte und technische Systeme im täglichen Umfeld des Menschen im wesentlichen unbewußt. So wurden z.B. die Steuereinheiten der Waschmaschinen von teurer und unflexibler Mechanik auf Mikroprozessor-basierte elektronische Systeme umgestellt. Der Anwender bemerkt diesen Wandel nur indirekt in Form von differenzierteren Programmabläufen und einer komfortableren Benutzerführung. Die dabei verwendeten Mikroprozessoren unterscheiden sich im Gegen-

satz zur allgemeinen Rechnertechnik im wesentlichen durch ihre kleinere Leistungsfähigkeit bezüglich der Taktfrequenz und Wortbreite. Im Bereich der integrierten Funktionseinheiten wurden allerdings bei den Mikroprozessoren für dedizierte Anwendungen die wichtigsten zum Betrieb unbedingt notwendigen Peripherieeinheiten mit auf dem selben Chip implementiert. Darunter versteht man z.B. Unterbrechungs-Controller, Zeitgeber, serielle Schnittstellen, Speicher-Controller und sogar kleinere flüchtige *SRAM*- bzw. nicht-flüchtige *ROM*-Speicher. Diese Funktionseinheiten bilden zusammen mit dem Mikroprozessor ein vollständiges Mikrorechner-system und werden deshalb im allgemeinen nicht mehr als Mikroprozessor, sondern als sogenannter „Mikrocontroller“ bezeichnet.

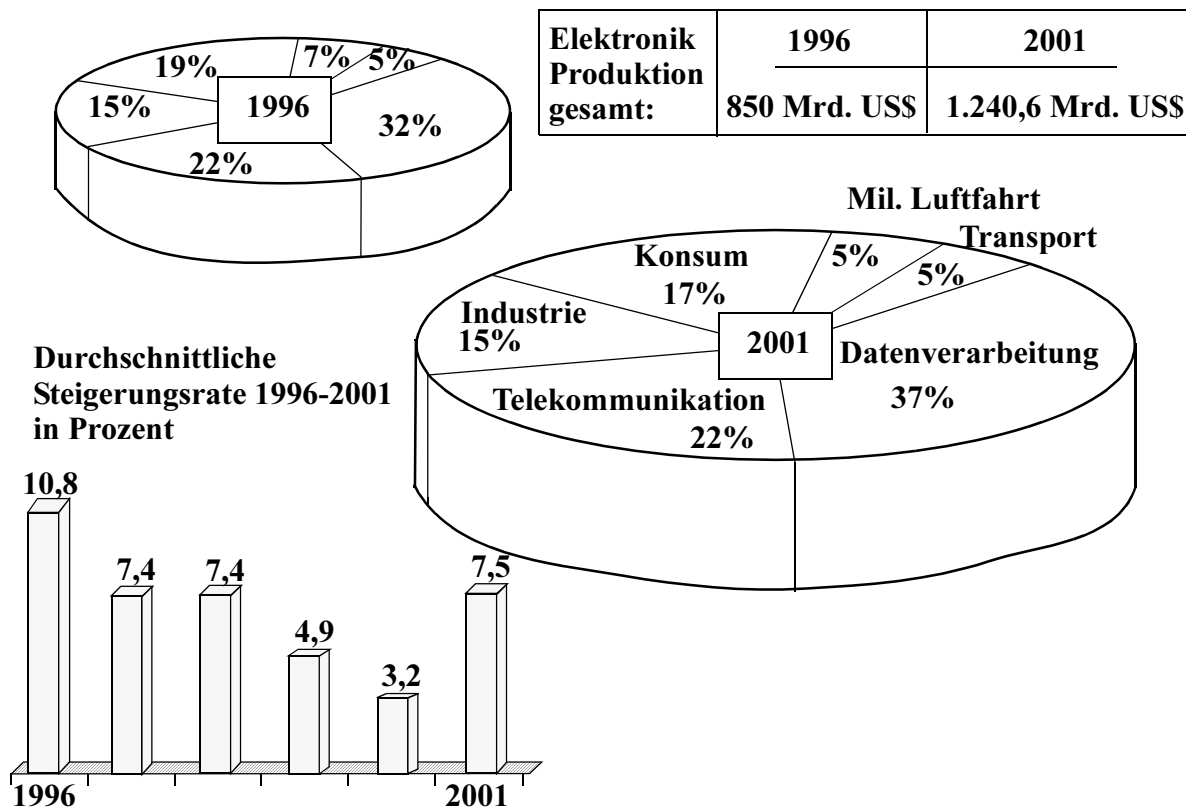


Abbildung 1.1 Prognose des weltweiten Elektronik-Umsatzes (Quelle: Dataquest, Juni 1997)

Diese Mikrocontroller entwickelten sich zu einer Basis-Komponente innerhalb der meisten elektronischen Produkte. Wie in Abbildung 1.1 dargestellt, wächst der Elektronik-Markt bis ins Jahr 2001 um durchschnittlich 7,9 % pro Jahr (siehe auch [Gies97]). Die Wachstumsrate für die speziell in dieser Arbeit betrachteten Anwendungsbereiche der industriellen Automation und Telekommunikation liegt hier bei durchschnittlich 7,4 % pro Jahr. Dabei entfallen auf die beiden Teilbereiche ca. 15 % (Industrie) bzw. 22 % (Telekommunikation) des Gesamtmarktes.

Industrielle Automation	Telekommunikation
Speicher-programmierbare-Steuerung (SPS)	Mobiltelefon
<i>PROFIBUS</i> - bzw. <i>ASI-Master</i> und <i>Slaves</i>	Intelligente Telephonanlagen
<i>INTERBUS-S</i>	ATM-Netzwerkkarten
Industrielle Roboter	Internet/Intranet-Anwendungen

Tabelle 1.1: Anwendungsbeispiele für Mikrocontroller-basierte Systeme

Tabelle 1.1 stellt beispielhaft einige Anwendungen zusammen. Die Beispiele zeigen bereits weitgehend am Markt erfolgreich eingeführte Produkte. Dabei unterliegen diese Systeme unterschiedlichsten Randbedingungen während des Betriebes. Diese Randbedingungen sind Faktoren wie Kosten, Bauform, Leistungsaufnahme, Benutzerführung und vor allem die Zuverlässigkeit.

- Bestimmte Anwendungen sind nur dann am Markt durchsetzbar, wenn ihre Anschaffungs- und Betriebskosten gering sind und dabei die Effizienz des Systems bzw. einer Dienstleistung deutlich steigern. Als Beispiel dienen viele Geräte des täglichen Bedarfs wie Telefonanlagen oder der kostengünstige Zugang zum Internet.
- Die unterschiedlichen Anwendungen erfordern für ihre mikroelektronischen Komponenten eine entsprechende Bauform, um sich den verschiedenen Umgebungsbedingungen anzupassen. Die Größen, die dabei eine Rolle spielen, sind der zur Verfügung stehende Raum und Umgebungsbedingungen wie Temperatur, Feuchtigkeit oder mechanische Belastungen. Als Beispiel dafür dienen die *PROFIBUS*- bzw. *ASI-Master* sowie deren *Slaves*, welche in der industriellen Automation unter extremen Umweltbedingungen eingesetzt werden.
- Die Leistungsaufnahme spielt bei allen mobilen Geräten, wie z. B. einem Mobiltelefon, eine entscheidende Rolle, um die Betriebsdauer zu maximieren und die Nachladezyklen zu minimieren.
- Die Benutzerführung differiert wesentlich. So gibt es Systeme, die über einen einfachen Tastendruck mit dem Benutzer kommunizieren, wie z. B. eine SPS-Maschinensteuerung. Außerdem gibt es Systeme, wie z.B. die aufwendige Steuerung eines Roboters, welche von einem Experten vor dem Produktionsbeginn auf ihre komplexe Tätigkeit programmiert werden müssen.
- Entscheidende Randbedingungen sind unterschiedliche Anforderungen an die Zuverlässigkeit von mikroelektronischen Systemen. Einige Systeme, wie z. B. eine Netzwerkkarte für den Internet-Zugang, dienen dem Komfort des Menschen. Bei einem Defekt wird sie repariert oder durch ein neueres Gerät ausgetauscht, ohne einen direkten Einfluß auf die Sicherheit des Menschen zu nehmen. Andere Systeme, wie z.B. die Heizungssteuerung, unterliegen schon weit höheren Sicherheitsanforderungen. Bei einem Ausfall können bei niedrigen Außentemperaturen beträchtliche Schäden an Gebäuden entstehen, wenn diese Fehlfunktion nicht unmittelbar bemerkt wird. Eine weitere Steigerung an die Anforderung nach unbedingter Zuverlässigkeit wird an Systeme gestellt, von deren korrekter Funktion das unmittelbare Leben von Menschen abhängt. Solche Systeme sind z.B. die SPS-Steuerungen in Atomkraftwerken oder auch die Auto-Pilot-Steuerung in Flugzeugen.

Die Darstellung der wichtigsten Randbedingung und deren Veranschaulichung an einigen Beispielen hat gezeigt, daß für den Entwurf solcher mikroelektronischer Systeme unterschiedlichste Anforderungen an den Entwickler gestellt werden. Diese Betrachtungen bilden die Grundlage für die wissenschaftliche Erforschung von entsprechenden Entwurfsmethoden.

In dieser Arbeit wird die Entwurfsmethodik „*Architekturwurf und Emulation von eingebetteten Systemen*“ vorgestellt, welche im wesentlichen auf einer zweistufigen Vorgehensweise basiert.

- In der ersten Stufe wird eine systematische Komponentenbewertung durchgeführt. Dabei werden alle mit einer Komponente beim Entwurf verbundenen Probleme analysiert und in entsprechende Entscheidungsfelder zusammengefaßt. Diese Entscheidungsfelder werden anschließend in einer Gesamtbewertung berücksichtigt. Durch diesen Schritt wird einer bestimmten Komponente aus der Menge von vielen möglichen Komponenten eine nach objektiven Kriterien, den sogenannten Entscheidungsparametern, beste Eignung für das zu entwerfende eingebettete System zugeordnet. Im Bereich der Hardware-Komponenten wird dieser Schritt hauptsächlich auf die Auswahl von komplexen *ASIC*-Bausteinen angewendet und optimiert. Diese erste Stufe wird in Kapitel 4 detailliert dargestellt.
- In der zweiten Stufe wird die Eignung der ausgewählten Komponente durch Emulation überprüft. Dabei wird die Komponente unter realistischen Umgebungsbedingungen betrieben. Ziel dieses Schrittes ist es, die im ersten Schritt für die Auswahl benutzten Entscheidungsparameter durch eine reale Betriebssituation zu bestätigen bzw. zu widerlegen.

Im Rahmen dieser Arbeit wurde das Emulationssystem SPYDER entworfen, welches zur Zeit aus zwei Werkzeugen besteht und in Kapitel 5 eingeführt wird. Das Werkzeug SPYDER-ASIC-X2 dient zur Emulation von *ASIC*-Komponenten bzw. von Synthese-fähigem *VHDL-Code* (engl.: *Intellectual Property Core - IP-Core*). Dieses Werkzeug basiert auf einer *PCI*-Einsteckkarte für einen PC und verbindet über einen speziellen *PCI*-Chip den *PCI*-Bus mit einem *SRAM*-basierten *FPGA* der neuesten Generation. Zusätzlich stehen externe synchrone *SRAM*-Bausteine sowie leistungsstarke Erweiterungs- und Testmöglichkeiten zur Verfügung.

Das zweite Werkzeug heißt SPYDER-CORE-P1 und dient zur Systemintegration bei der Emulation der gesamten Architektur des eingebetteten Systems. Es basiert im wesentlichen auf einem 32-Bit-Mikrocontroller, entsprechenden Speicher-Komponenten sowie einem *SRAM*-basierten *FPGA*. Beide Werkzeuge können über eine Verbindungsplatine sehr eng miteinander kombiniert werden, um unterschiedlichste Systemarchitekturen zu emulieren.

In Kapitel 6 wird die Entwurfsmethodik anhand zweier praxisrelevanter Anwendungsbeispiele demonstriert. Das erste Beispiel befaßt sich mit dem Entwurf eines ATM-Diagnosemonitors und zeigt insbesondere den in der ersten Stufe stattfindenden Auswahlprozeß eines komplexen *ASIC*-Bausteins zum Empfangen und Senden von ATM-Zellen. Das Werkzeug SPYDER-ASIC-X2 wird benutzt, um die einzelnen Entscheidungsparameter in der zweiten Stufe zu überprüfen, ohne dabei die gesamte Systemarchitektur zur Verfügung zu haben.

Das zweite Beispiel zeigt den Entwurf eines *ASI-Masters*. Bei diesem Beispiel steht zur Zeit nur eine *ASIC*-Komponente als Verbindungsglied zur Umwelt zur Verfügung, welche im wesentlichen eine serielle digital/analog-Wandlung durchführt. Für die Anbindung dieses Bausteins an einen Mikrocontroller ist keine *ASIC*-Komponente verfügbar. Deshalb kann in der ersten Stufe keine *ASIC*-Komponenten-Auswahl für diese benötigte Teilfunktion durchgeführt werden. Es muß zur Implementierung der gesuchten Schnittstellen-Funktionalität eine umfangreiche Eigenentwicklung durchgeführt werden. Dafür wird in der zweiten Stufe das Werkzeug SPYDER-ASIC-X2 benutzt, um diese komplexe Schnittstelle zwischen Mikrocontroller und dem *ASIC* zur Umgebung zu emulieren. Das Werkzeug SPYDER-CORE-P1 dient zur Emulation der gesamten Systemarchitektur, wobei insbesondere auch der Einfluß eines Echtzeitbetriebssystems analysiert wird.

Kapitel 2

Komponenten eingebetteter Systeme

In diesem Kapitel werden die wichtigsten Hardware- und Software-Komponenten eingebetteter Systeme vorgestellt. Die wesentliche Erkenntnis dabei ist, daß jede der verschiedenen Komponenten eigene Entwurfswerkzeuge besitzt. Diese Tatsache bedeutet, daß ein Entwickler auf Systemebene viele Werkzeuge gleichzeitig beherrschen muß. Bei der Definition einer entsprechenden Entwurfsmethode muß dieser Aspekt berücksichtigt werden. Um diesen Anforderungen gerecht zu werden, ist die Spezialisierung der Entwickler auf bestimmte Teilgebiete, vorzugsweise auf den Hardware- oder Software-Entwurf, in der Praxis nicht zu vermeiden. Ein Ansatz zur Lösung dieser Problematik ist die Parallelisierung des Entwurfsablaufes, der einen gleichzeitigen Hardware- und Software-Entwurf gestattet, begleitet von einem kontinuierlichen Informationsaustausch zwischen beiden Entwickler-Gruppen. Dadurch können sich die Entwickler auf ihre jeweilige Domäne konzentrieren und ihre Arbeit dennoch in kürzeren Entwurfszeiträumen durchführen.

2.1 Hardware-Komponenten

In diesem Abschnitt werden zunächst die bei digitalen eingebetteten Systemen eingesetzten Hardware-Komponenten beschrieben. Im Bereich der Mikrocontroller wird ein Überblick über die 8-Bit, 16-Bit und 32-Bit-Mikrocontroller gegeben. Dazu wurden die jeweiligen am Markt dominierenden Typen ausgewählt. Beschrieben werden neben der immer weiter steigenden Komplexität auch die damit verbundenen Entwurfs- und Testwerkzeuge. Zusätzlich wird analysiert, welche Anforderung der Einsatz bestimmter Mikrocontroller-Typen auf eine entsprechende Entwurfsmethode hat.

Eng mit den Mikrocontrollern verbunden sind die entsprechend benötigten Speicherbausteine. In Abschnitt 2.1.2 werden typische Standard-Speicherbauelemente bis zu den neuesten Speicherbausteinen mit extrem hohem Datendurchsatz beschrieben. Die enge Verbindung zu Mikrocontrollern wird durch die für den Zugriff benötigten unterschiedlichen Zugriffsverfahren unterstrichen, welche die Mikrocontroller unterstützen müssen.

Eine weitere wichtige Komponentengruppe stellen die programmierbaren Bausteine dar. Dazu wird in Abschnitt 2.1.3 ein Überblick gegeben, beginnend bei einfachen Bausteinen niedriger Komplexität bis zu hochintegrierten *FPGA* neuester Generation mit mehreren zehntausend Gatteräquivalenten. Zusätzlich wird für jede Bausteinklasse das dazugehörige typische Anwendungsspektrum beschrieben.

2.1.1 Mikrocontroller

2.1.1.1 8-Bit-Mikrocontroller (Intel MCS51)

Die 8-Bit-Mikrocontroller stellen noch heute die zahlenmäßig größte Anzahl bei der Implementierung von eingebetteten Systemen dar. Als Beispiel wird in diesem Abschnitt die MCS51-Familie [Intel94] von Intel beschrieben, die in einem sehr großen Anwendungsbereich eingesetzt wird. Die Gründe für die weite Verbreitung liegen sicherlich in ihrem geringen Preis von 2-5 US\$ und den vielen verschiedenen Reserve-Herstellern (engl.: *second source*), die dieser Mikrocontroller-Familie eine hohe Verfügbarkeit am Markt sichern. Dadurch werden diese Mikrocontroller in fast allen besonders kostensensitiven eingebetteten Systemen eingesetzt. Aber auch sehr teure und extrem sicherheitsrelevante Anwendungen, wie z.B. die Autopilot-Steuerung in Flugzeugen, Steuerungen in Atomkraftwerken und in vielen militärischen Systemen, basieren auf dieser Mikrocontroller-Familie. Der Hauptgrund für den Einsatz in solchen sehr teuren Systemen ist die extrem hohe Zuverlässigkeit. Durch die fast zwanzigjährige Verfügbarkeit am Markt und den Einsatz in vielen verschiedenen Systemen verfügen diese Bausteine über eine besonders hohe Fehlerabdeckung, die insbesondere auch sicherheitsrelevante Anwendungen erlaubt. In den folgenden Abschnitten wird im Gegensatz dazu gezeigt, daß moderne Hochleistungs-Mikrocontroller und *ASIC* auf der einen Seite zwar eine enorme Leistungsfähigkeit haben, auf der anderen Seite aber bei ihrer Markteinführung fast nie fehlerfrei sind. Ihr Einsatz in sicherheitsrelevanten Anwendungen ist daher nicht verantwortbar.

Die MCS51-Mikrocontroller-Familie besteht aus einer ganzen Serie von verschiedenen Derivaten, die alle die gleiche Architektur und den gleichen Befehlssatz (engl.: *Instruction Set Architecture -ISA*) haben. Damit kann ein einmal entwickelter und getesteter Programmcode auf alle Familienmitglieder portiert werden. In Abbildung 2.1 wird das Blockdiagramm des MC8051-Kerns dargestellt, an dem die wesentlichen Eigenschaften erklärt werden können.

- Die zentrale Verarbeitungseinheit (engl.: *Central Processing Unit - CPU*) verfügt über 8-Bit breite Datenpfade und wurde speziell für Steuerungsanwendungen optimiert. Sie verfügt dabei nur über einfache logische Operationen, kann aber zusätzlich über spezielle Befehle auf einzelne Bits zugreifen, die in einem speziellen Registersatz abgelegt sind. Dieser Registersatz wird in einem 128 Byte-RAM auf dem Mikrocontroller-Chip gespeichert. Die CPU wird von einer auf dem Chip integrierten Takterzeugungseinheit (engl.: *OSCillator unit - OSC*) gespeist, an dem extern nur ein einfacher und kostengünstiger Quarz angeschlossen werden muß.
- Zur Unterbrechungsanforderung (engl.: *interrupt request*) und der Verarbeitung ist ein eigener *Interrupt-Controller* implementiert. Er verarbeitet insgesamt sechs *Interrupt-Quellen* und unterscheidet zwischen zwei verschiedenen *Interrupt-Prioritäten*.
- Ein wesentliches Merkmal der MC8051-Familie ist der getrennte Adressraum zwischen den Daten und Befehlen.

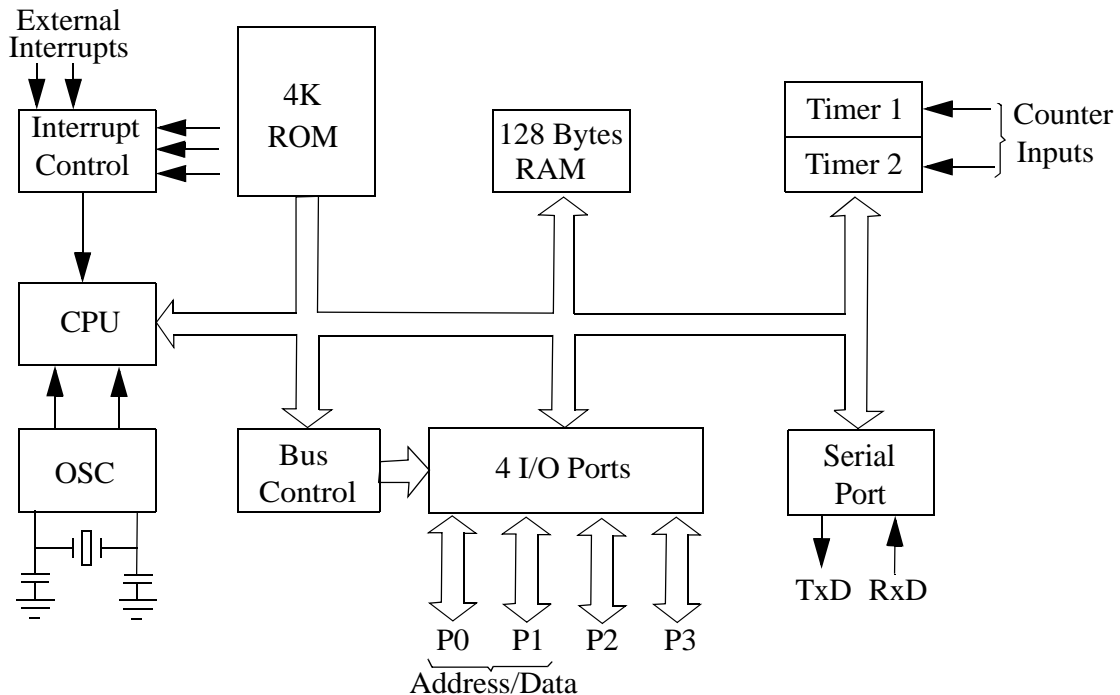


Abbildung 2.1 Blockdiagramm des MC8051-Kerns (Quelle: [Intel94])

Prinzipiell stehen für den Befehls- und Datenspeicher-Adressraum jeweils 64 KByte zur Verfügung. Innerhalb der MCS51-Familie gibt es Typen, die bereits auf dem Chip nicht-flüchtigen Speicher (*ROM* oder *EPROM*) zur Verfügung stellen, wobei die Größe zwischen 4 KByte, 8 KByte, 16 KByte, oder 32 KByte variiert. Im Datenbereich verfügen diese Mikrocontroller über bis zu 256 Byte flüchtigen Speicher, in dem zur Laufzeit wichtige Systemressourcen abgelegt werden. Dazu gehört der Registerstapel und die *Interrupt*-Vektoren.

Der eigentliche *CPU*-Kern besitzt einen Befehlssatz, der auch komplexe Befehle ausführen kann. Beispielsweise können Additionen auf externen Speicherzellen durchgeführt werden. Dieser komplexe Befehlssatz (engl.: *Complex Instruction Set Computer - CISC*) steht im Gegensatz zu den Befehlssätzen bei den 16-Bit- und 32-Bit-Mikrocontrollern, wie in den beiden folgenden Abschnitten gezeigt wird.

- Schon bei dieser Mikrocontroller-Generation ist eine serielle Schnittstelle vorhanden, welche bei der Entwicklung zum Laden des Programmcodes und im Anwendungsbetrieb zur Kommunikation mit anderen Systemen benutzt werden kann.
- Zwei Zeitgeber (engl.: *Timer*) und insgesamt vier 8-Bit parallele Ein/Ausgabe-Schnittstellen (engl.: *I/O Ports*) gehören ebenfalls zu den Standard-Einheiten dieser Mikrocontroller-Familie. Diese *I/O-Ports* können zur einfachen Ein- und Ausgabe von digitalen Signalen benutzt werden. Wird allerdings externer Speicher benötigt, dient der *Port P0* als bi-direktionaler Adress- und Datenport (die unteren 8-Bit Adressen und 8-Bit Daten werden durch einen 8-Bit breiten Multiplexer auf dem Chip umgeschaltet) und der *Port P2* liefert die oberen 8-Bit Adressen.

Die Aufbau-Technologie für den Systementwickler stellt bei diesen Mikrocontroller-Typen noch keine besonderen Anforderungen. In der Regel sind die Bausteine in einfacheren Ge-

häuseformen (z.B. DIP oder PLCC) verfügbar und haben je nach Derivat zwischen 40 und 84 Anschluß-Pins. Die Taktfrequenzen im Bereich zwischen 12 MHz und 20 MHz sind aus heutiger Sicht relativ unkritisch und erlauben im Entwicklungsbereich einfache Laboraufbauten.

Im Bereich der Software-Entwicklung basieren viele Anwendungen auf der direkten Programmierung in Assembler-Sprache. Dabei wird der Code direkt in einem nicht-flüchtigen EPROM gespeichert und im Zielsystem zur Ausführung gebracht. Die Programmausführung ist dabei völlig statisch. Ein Befehl wird mit minimal 12 Takten und maximal 24 Takten gelesen und ausgeführt. Erst wenn der Befehl gelesen, interpretiert, ausgeführt und die Ergebnisse in die entsprechenden Register zurückgeschrieben wurden, wird der nächste Befehl verarbeitet. Moderne Konzepte wie Pipeline-Verarbeitung oder *on-Chip-Caches* gibt es nicht. Die begrenzte Komplexität erlaubt die Benutzung von Simulatoren für die einzelnen Derivate, die auf einem Entwicklungsrechner laufen und das Mikrocontroller-Verhalten exakt nachbilden können. Diese Simulatoren sind relativ kostengünstig und werden von den Entwicklern zur Fehlersuche innerhalb des Programmcodes benutzt.

Bei Entwicklungsprojekten, bei denen der Programmcode größer wird, ist auch der Einsatz von Hochsprachen wie C üblich. Dabei verwendet man sehr häufig C-Code, welcher durch Assembler-Code an den lauffzeitkritischen Stellen optimiert wurde. Bei Entwicklungsprojekten, die unter harten Echtzeitbedingungen arbeiten müssen, haben sich sogenannte In-System-Emulatoren etabliert, die wesentlich teurer sind als die Simulatoren. Dabei werden die Mikrocontroller aus dem Zielsystem entfernt. Über mechanische Kontaktsockel werden spezielle Mikrocontroller-Typen eingesetzt, welche durch weitere Zusatz-Hardware eng mit einem Entwicklungsrechner gekoppelt sind. Diese speziellen Mikrocontroller unterscheiden sich in ihrem elektronischen und logischen Verhalten nicht von den eigentlichen Typen für die Produktion, stellen jedoch auf dem Chip noch weitere Einheiten zum Testen des Programmablaufes zur Verfügung. Dieser spezielle Betriebsmode (engl.: *ON-Circuit Emulation - ONCE*) wird von der Zusatz-Hardware angesprochen und vom Entwicklungsrechner, im allgemeinen einem PC, kontrolliert. Dadurch ist es möglich, den Programmcode über den *ONCE-Port* auf das Zielsystem zu laden und bestimmte Programmsequenzen während der Ausführung in Echtzeit zu überwachen. Bei bestimmten Fehlerzuständen, welche mit Hilfe von entsprechenden Trigger-Bedingungen definiert werden, kann die Programmausführung gestoppt und der gesamte interne Zustand des eingebetteten Systems ausgelesen und bewertet werden. Mit einem solchen Werkzeug ist die Entwicklung und der Test von Echtzeitsystemen möglich. Der Einsatz von Echtzeitbetriebssystemen ist allerdings wegen der relativ geringen Leistungsfähigkeit und des beschränkten Adressraumes dieser Mikrocontroller-Familie nicht möglich. Diese Einführung zeigt, daß bei der Entwicklung von Echtzeitsystemen die Entwurfsproblematik schon bei relativ einfachen 8-Bit-Mikrocontrollern sprunghaft ansteigt. Diese Tatsache entwickelt sich bei komplexen 32-Bit-Mikrocontrollern mit dynamischer Programmausführung zu schwierigen Entwurfsproblemen, deren Lösung insbesondere auch Gegenstand der in dieser Arbeit vorgestellten Methode ist.

2.1.1.2 16-Bit-Mikrocontroller (Siemens C167)

Das rapide Wachstum der eingebetteten Systeme im Bereich industrieller Steuerungsanwendungen hat den Übergang zu 16-Bit-Mikrocontrollern beschleunigt. Diese Anwendungen stehen häufig unter besonderen Randbedingungen, wie z.B. die Verarbeitung von vielen digitalen und analogen Eingangsvariablen unter harten Echtzeitbedingungen. Zusätzlich unterliegen diese Systeme auch Einschränkungen bezüglich der zur Verfügung stehenden Platinenfläche, des Leistungsverbrauches sowie niedriger Systemkosten. Die folgenden vier prinzipiellen Vor-

teile der 16-Bit-Mikrocontroller-Architekturen erzwingen deren Einsatz in komplexeren eingebetteten Systemen:

- Einer der wichtigsten Vorteile ist der größere lineare Adressraum im Gegensatz zu den 8-Bit-Mikrocontrollern, welcher bei der Programmierung von komplexen Algorithmen für die spezielle Anwendung benötigt wird.
- Die Verarbeitungsgeschwindigkeit des *CPU*-Kerns innerhalb des Mikrocontrollers kann nur durch die Erhöhung der Taktfrequenz nicht ausreichend gesteigert werden. Neue Architekturkonzepte, die bei Prozessoren der allgemeinen Rechnertechnik weitgehend zum Standard gehören, werden auch bei den Mikrocontrollern benötigt. Eine wichtige Technik dabei ist die Einführung der Pipeline-Verarbeitung, die eine parallele Verarbeitung von mehreren Befehlen gleichzeitig erlaubt.
- Um die Chip-Fläche optimal auszunutzen, muß der Befehlssatz entsprechend dimensioniert sein. Dazu werden Befehlssätze auf die wichtigsten Befehle reduziert (engl.: *Reduced Instruction Set Computer - RISC*), um diese effizient und mit möglichst wenig Hardware-Ressourcen implementieren zu können.

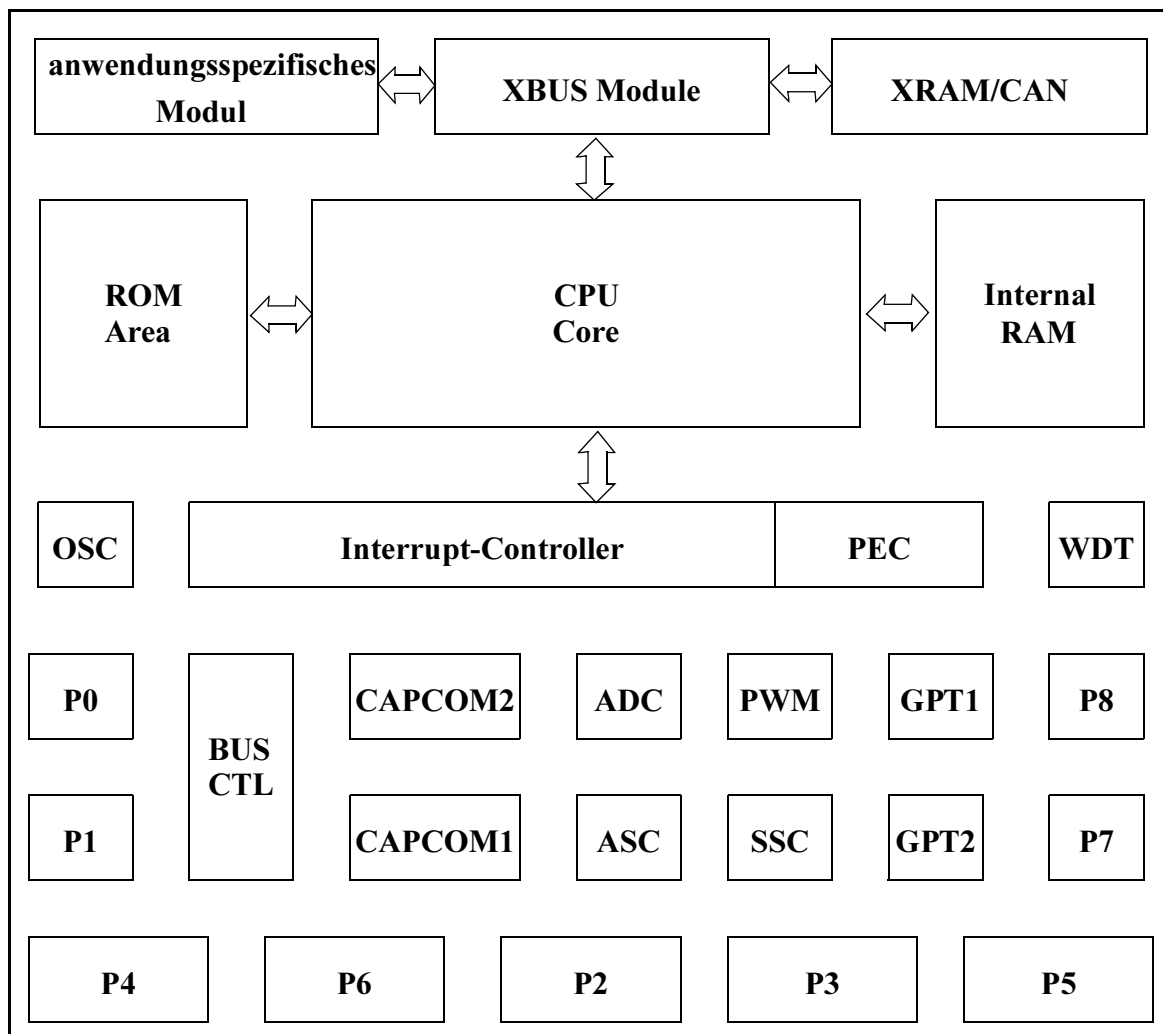


Abbildung 2.2 Blockdiagramm des SIEMENS C167 (Quelle: [Sie96])

- Um zusätzliche externe Bausteine zu eliminieren, werden weitere Peripheriekomponenten auf dem Chip gebraucht. Diese Forderung betrifft vor allem größere und flexiblere *on-Chip*-Speicher, aber auch komplexere digitale und analoge Schnittstellen zwischen dem Mikrocontroller und seiner Umgebung.

Betrachtet man die Gesamtfunktionalität dieses Mikrocontroller-Typs im Blockdiagramm der Abbildung 2.2, so sind neben dem eigentlichen CPU-Kern und den *on-Chip-ROM* und *RAM*-Bereichen eine Vielzahl von zusätzlichen Funktionseinheiten vorhanden. Die Ports P0 bis P8 bieten insgesamt 111 digitale Ein- und Ausgabelösungen, wobei allerdings die Ports P0, P1 und P4 beim Anschluß externer Speicher belegt sind. Zwei Überwachungseinheiten (engl.: *Capture Compare Unit - CAPCOM*) können zur Beobachtung von externen Signalverläufen frei programmiert werden und ein Analog-Digital-Konverter (*ADC*) wandelt analoge Signale in digitale Signale auf dem Chip um. Eine asynchrone bzw. synchrone serielle Schnittstelle (*ASC* und *SSC*) dienen zur Kommunikation mit externen Geräten. Zwei globale Zeitgeber (*GPT1* und *2*), eine Einheit zur Pulsweitenmodulation (*PWM*), ein *Interrupt-Controller* mit vorgeschaltetem Peripherie-Ereignis-Controller (engl.: *Peripherie Event Controller - PEC*), ein *on-Chip*-Oszillator (*OSC*) sowie ein *Watch-Dog-Timer (WDT)* stehen ebenfalls als programmierbare Einheiten auf dem Chip zur Verfügung. Eine genaue Beschreibung der einzelnen Peripheriekomponenten ist in [Sie96] zu finden. Das sogenannte *XBUS*-Modul definiert einen genormten Bus-Standard auf dem Chip, an dem verschiedene anwendungsspezifische Module angeschlossen werden können. Einige Derivate des C167 nutzen bereits diesen Bus, um z.B. ein zusätzliches *on-Chip-SRAM (XRAM)* sowie ein sogenanntes *Controller Area Network (CAN)* mit der *ALU* zu verbinden.

Würde man diese Leistungssteigerung allein durch die Erhöhung der Taktfrequenz gewinnen, würde die Leistungsaufnahme dieser *CMOS*-Schaltungen drastisch ansteigen. Deshalb werden bei dieser Mikrocontroller-Klasse die bereits am Anfang dieses Abschnittes erwähnten Konzepte der *RISC*-Architektur und Pipeline-Verarbeitung eingesetzt. Um diese Konzepte zu erklären, wird im folgenden der *CPU*-Kern des C167 näher betrachtet.

Pipeline-Verarbeitung

Der C167 zerlegt die Ausführung eines Befehles in insgesamt vier Stufen. In jeder Stufe befindet sich zu einem bestimmten Zeitpunkt ein entsprechender Befehl, jeweils in einem anderen Ausführungszustand. Diese überlappende Befehlsausführung läßt in jedem Maschinenzyklus einen Befehl in die Pipeline eintreten und einen Befehl austreten. Da gleichzeitig in der arithmetisch-logischen Einheit (engl.: *Arithmetic Logic Unit - ALU*) vier Befehle in der Ausführung sind, erscheint es einem externen Beobachter, als ob vier Befehle parallel oder simultan in Bearbeitung wären. Diese überlappende Bearbeitung bezeichnet man als Pipeline-Verarbeitung. Die vier Pipelinestufen führen dabei folgende Funktionen aus:

- **Erste Stufe: Befehl aus dem Speicher holen (engl.: *fetch stage*)**

In dieser Stufe wird der nächste auszuführende Befehl, welcher vom Befehlszähler (engl.: *Instruction Pointer - IP*) und dem Code-Segment-Zeiger (engl.: *Code Segment Pointer - CSP*) adressiert wird, aus dem internen *RAM*, dem internen *ROM* oder dem externen Speicher gelesen.

- **Zweite Stufe: Dekodieren des Befehls (engl.: *decode stage*)**

In dieser Stufe wird der Befehl dekodiert. Wenn der Befehl weitere Operanden benötigt, werden die entsprechenden Speicheradressen berechnet und gelesen. Bei Befehl-

len, die impliziten Zugriff auf den Stapel haben, wird der Stapel-Zeiger (engl.: *Stack Pointer SP*) entsprechend erhöht oder erniedrigt. Bei Sprung-Befehlen wird das Sprungziel im Befehlszähler-Register und im Code-Segment-Register eingetragen.

- **Dritte Stufe: Befehlsausführung (engl.: *execution stage*)**

In dieser Stufe wird der eigentliche Befehl in der *ALU* ausgeführt. Zusätzlich werden die entsprechenden Bedingungs-Flags im Prozessor-Status-Wort (PSW) aktualisiert.

- **Vierte Stufe: Zurückschreiben der Ergebnisse (engl.: *write back*)**

In dieser Stufe werden die Ergebnisse der Operationen in der *ALU* an die entsprechenden Speicher-Adressen zurückgeschrieben. Diese können sich im Speicher oder in den internen Registern befinden.

Zeit (t)	1. Maschi- nenzyklus $t=t_0$	2. Maschi- nenzyklus $t=t_1$	3. Maschi- nenzyklus $t=t_2$	4. Maschi- nenzyklus $t=t_3$	5. Maschi- nenzyklus $t=t_4$	6. Maschi- nenzyklus $t=t_5$
Fetch	I1	I2	I3	I4	I5	I6
Decode		I1	I2	I3	I4	I5
Execute			I1	I2	I3	I4
WriteBack				I1	I2	I3

Tabelle 2.1: Sequentielle Befehls-Pipeline beim C167-Mikrocontroller

Die Tabelle 2.1 zeigt den Eintritt eines Befehls I1 zum Zeitpunkt t_0 in die Befehls-Pipeline, der mit jedem Maschinenzyklus eine Stufe tiefer in der Pipeline fortschreitet. Die Zeit zwischen zwei Maschinenzyklen beim C167 schreitet dabei in Schritten von 100 ns fort. Zum Zeitpunkt t_1 tritt der nächste Befehl die sequentielle Bearbeitung in der Pipeline an.

Zum Zeitpunkt t_3 verläßt der Befehl I1 die Pipeline. Er brauchte für seine Bearbeitung also insgesamt vier Maschinenzyklen. An diesem Beispiel wird gezeigt, daß innerhalb einer bestimmten Zeitperiode die Zahl der bearbeitbaren Befehle drastisch ansteigt. Im idealen Fall liegt beim C167 eine Steigerung um den Faktor vier vor. Dies gilt aber nur für den ungestörten Befehlsdurchfluß. Treten Sprünge auf oder werden durch externe Ereignisse Unterbrechungen ausgelöst, wird die Befehlsabarbeitung an der Stelle unterbrochen und an einer anderen Stelle fortgesetzt. Dadurch entsteht eine Störung des Befehlsflusses, da bereits gelesene Befehle in der Pipeline ungültig werden und entfernt werden müssen. Die Befehle an der neuen Stelle müssen zuerst gelesen und in die Pipeline eingegeben werden. Daraus folgt, daß die Verluste bei Unterbrechungen um so größer werden, je tiefer die Pipeline ist. Bei reaktiven Systemen, wie bei eingebetteten Systemen in der industriellen Automation, treten solche Unterbrechungen sehr häufig auf. Die Pipeline-Tiefe darf deshalb nicht zu groß werden. Im realen Betrieb ist die durchschnittliche Steigerung deshalb kleiner als der ideale Faktor von vier, sicherlich aber deutlich größer als eins.

RISC-Architektur

Um die Leistung der Pipeline-Verarbeitung zu optimieren, werden häufig auch sogenannte „reduzierte Befehlssätze“ (engl.: *Reduced Instruction Set Computing - RISC*) verwendet. Der Befehlssatz des C167 ist ein Beispiel für eine solche *RISC*-Architektur und wurde mit folgender Zielsetzung entwickelt:

- Komplexere Befehle werden in eine Sequenz von einfacheren Befehlen, die häufig benötigt werden, zerlegt. Insbesondere werden Befehle zum Transfer von Operanden in bzw. von temporären Registern vermieden. Um auf externe *Interrupts* schnell reagieren zu können, wird der aktuelle Prozessorstatus vor Eintritt in die *Interrupt-Service-Routine (ISR)* automatisch gesichert.
- Die komplexe Kodierung von Befehlen wird vermieden. Dieses Ziel wird dadurch erreicht, daß die Operanden in gleiche Bit-Felder innerhalb des Befehlsformates eingetragen werden. Zusätzlich werden auch komplexere Adress-Dekodierverfahren vermieden. Diese beiden Maßnahmen verkürzen die Zeit für die Befehls-Dekodierung und vereinfachen die Entwicklung von Compilern und Assemblern.
- Die am meisten benutzten Befehle werden als sogenannte „Ein-Wort-Befehle“ implementiert. Alle anderen Befehle brauchen zwei Worte. Dadurch können die Befehle im Speicher an Wortgrenzen plaziert werden. Diese Eigenschaft vermeidet den Verbrauch von umfangreicher Hardware zur Justierung der Befehle vor ihrer Dekodierung.

Ein wesentliches Kennzeichen für die *RISC*-Architektur ist der relativ große und mehrfach vorhandene Registersatz. Die Befehle beziehen sich bei einer *RISC*-Architektur im wesentlichen immer auf einzelne Register, in denen die Operanden gehalten werden müssen. Beim C167 stehen für einen Kontext maximal 16 Register (engl.: *general purpose Register R0-R15*) zur Verfügung. Diese Registerbank wird zusammen mit den restlichen *CPU*-Registern physikalisch im *on-Chip-RAM* gespeichert. Der momentan gültige Kontext wird durch den Inhalt im sogenannten Kontext-Zeiger-Register (engl.: *Context Pointer - CP*) angezeigt. Bei einem Kontextwechsel, ausgelöst z.B. durch einen externen *Interrupt*, wird der gesamte momentane Prozessorzustand automatisch gespeichert. Dazu zählen neben dem Stapelzeiger (engl.: *Stack Pointer - SP*) und dem aktuellen Prozessor-Status-Wort PSW natürlich auch der aktive Registersatz R0-R15. Dieser Umschaltprozeß kann in einem einzigen Takt durchgeführt werden, in dem der Kontext-Zeiger auf einen neuen Wert gesetzt wird und somit eine neue Registerbank adressiert. Beim C167 ist die Anzahl der unterschiedlichen Registerbänke nur durch die Speicherkapazität des internen *RAM* begrenzt.

Im Bereich der Programmierung und des Tests werden prinzipiell die gleichen Verfahren wie bei den 8-Bit-Mikrocontrollern angewendet. Aufgrund der komplexeren Software ist ein stärkerer Trend zu Hochsprachen wie C, MODULA2 oder FORTH zu erkennen. Wenn die komplexen *on-Chip*-Peripherie-Einheiten ausgenutzt werden und der Datenfluß zwischen ihnen und dem *CPU*-Kern optimiert werden soll, ist der massive Einsatz von Assembler-Programmierung notwendig. Bei dieser Mikrocontroller-Klasse werden im Bereich der industriellen Kommunikation verstärkt standardisierte Protokolle wie *CAN* oder *PROFIBUS* eingesetzt, welche aus mehreren Kommunikationsschichten bestehen. Das z.B. bei *PROFIBUS* eingesetzte OSI-Schichtenmodell ist in [BeKa91] beschrieben. Die dazu notwendigen Hardware-Einheiten sind beim C167 über das *XBus*-Modul mit auf dem Chip integriert. Verschiedene Hersteller bieten spezielle Software an, die diese Einheiten entsprechend initialisieren und die darüber liegenden höheren Kommunikationsschichten in Software auf dem *CPU*-Kern behandeln. Diese sehr auf die *on-Chip*-Kommunikationseinheiten abgestimmte Software wird in der Literatur auch als Firmware bezeichnet.

Im Bereich der Technologie ist auch eine Erhöhung der Anforderungen zu erkennen. Durch die gesteigerte *on-Chip*-Peripherie und die breitere Architektur steigt die Anzahl der Ein/Ausgabesignale auf über 100 Pins an. Der C167 ist in einem 144 Pin *PQFP*-Gehäuse mit einem Pin-Abstand von 0,65 mm eingebaut. Das bedeutet, daß wegen der kleineren Pin-Abstände und

der durch die höhere Rechenleistung bedingten drastisch ansteigenden Umschaltvorgänge auf dem Chip die Anforderungen an den Prototyp-Aufbau bereits ansteigen. Einfache Laboraufbauten wie bei den 8-Bit-Mikrocontrollern sind hier nicht mehr möglich, sondern erfordern in der Regel Platinen mit mindestens vier elektrischen Lagen.

Um der steigenden Software-Komplexität zu begegnen, wird für den C167 auch ein Echtzeitbetriebssystem angeboten. Hier zeigt sich jedoch die Grenze der Anwendbarkeit von 16-Bit-Mikrocontrollern. Der C167 ist ein typischer Vertreter dieser Klasse und kann maximal einen Adressraum von 16 MByte bedienen, welcher allerdings in viele 64 KByte-Blöcke zerlegt ist. Ein modernes Echtzeitbetriebssystem braucht aber in der Regel einen größeren linearen Arbeitsspeicherbereich als die beim C167 zu Verfügung stehenden 64 KByte-Blöcke. Deswegen muß das Betriebssystem beim Speicherzugriff den gesamten Arbeitsspeicher in Form von mehreren 64 KByte-Blöcken verwalten, wodurch die Ausführungsgeschwindigkeit deutlich begrenzt wird. Anwendungen, deren komplexe Software-Architektur den Einsatz eines Echtzeitbetriebssystems erfordert, beschleunigen den Übergang zu 32-Bit-Mikrocontrollern. Diese 32-Bit-Mikrocontroller sind Gegenstand des nächsten Abschnittes und bieten durch ihren 32-Bit-Adressraum optimale Voraussetzungen zum Einsatz eines Betriebssystems.

2.1.1.3 32-Bit-Mikrocontroller (*Embedded PowerPC 4xx, 5xx und 8xx*)

Der rapide Fortschritt im Bereich der Mikroelektronik bei gleichzeitig sinkenden Herstellungskosten hat es ermöglicht, hochintegrierte VLSI-Chips auch in weiten Anwendungsbereichen eingebetteter Systeme zu etablieren. Diese positive Entwicklung bei den Herstellungskosten hat die Einführung von 32-Bit-Mikrocontrollern in den letzten fünf Jahren wesentlich beschleunigt. Hochleistungsfähige Mikrocontroller-Chips sind heute ab ca. 15 US\$ auf dem Markt verfügbar. Dieser Preisbereich bietet selbst in kostenintensiven Anwendungen Einsatzmöglichkeiten an.

Betrachtet man die Architektur dieser Mikrocontroller, so kann man erkennen, daß sie den allgemeinen Architektur-Merkmalen der Hochleistungsprozessoren in der allgemeinen Rechnertechnik folgen. Im vorangehenden Abschnitt wurden am Beispiel des C167 die beiden Architektur-Merkmale Pipeline-Verarbeitung und der *RISC*-Befehlssatz eingeführt. Diese beiden Merkmale werden auch bei den 32-Bit-Mikrocontrollern weitgehend übernommen. Zusätzlich kommen aber noch weitere Architektur-Merkmale hinzu, die im folgenden Text anhand der in dieser Leistungsklasse sehr weitverbreiteten *PowerPC*-Architektur eingeführt werden. Diese Architektur wurde ursprünglich für die allgemeine Rechnertechnik definiert. Durch die beiden führenden Mikrocontroller-Hersteller Motorola und IBM getrieben, setzte sich diese Architektur in den letzten Jahren als *CPU*-Kern innerhalb der 32-Bit-Mikrocontroller durch. Der Vorteil bei der *PowerPC*-Architektur liegt in der einheitlich festgelegten Software-Schnittstelle, die alle *PowerPC*-Prozessoren einhalten müssen. Damit ist ein Programm-Code von allen Prozessoren dieser Architekturreihe ausführbar, was die Anzahl der Compiler-Entwicklungen minimiert und den Entwicklungsaufwand auf einen *PowerPC*-Compiler konzentriert. Unterhalb dieser Software-Schnittstelle ist die Art und Weise der Hardware-Implementierung offen. Dadurch können für verschiedene Anwendungsspektren unterschiedliche Architekturkonzepte in Hardware implementiert werden. Diese *PowerPC*-Software-Schnittstelle ist in drei Büchern niedergelegt, die sich auf folgende Schwerpunkte beziehen:

- Buch 1: Benutzer-Befehlssatz (engl.: *User Instruction Set Architecture*)
- Buch 2: Virtuelle Umgebung (engl.: *Virtual Environment Architecture*)

- Buch 3: Betriebssystem-Umgebung (engl.: *Operating Environment Architecture*)

Die drei Bücher sind in einer gemeinsamen Ausgabe zusammengefaßt und in [IBM94] zu finden. In den letzten fünf Jahren sind spezielle Familien für eingebettete Systeme auf dem Markt erschienen. Diese sogenannten *Embedded PowerPC*-Familien PPC40x von IBM und MPC8xx bzw. MPC5xx von Motorola basieren alle auf der *PowerPC*-Architektur und besitzen zusätzlich eine Vielzahl von verschiedenen *on-Chip*-Peripherie-Komponenten. Ihre speziellen Anwendungsspektren liegen in eng aneinandergrenzenden Teilbereichen innerhalb der industriellen Automation und Kommunikation.

Für eingebettete Systeme mit extremen Anforderungen an die Rechenleistung ist zur Zeit auch der *PowerPC 740/750* verstärkt im Einsatz. Er ist aber im wesentlichen kein Mikrocontroller, sondern ein Mikroprozessor und wird im Ausblick zu diesem Abschnitt erläutert.

Dynamische Bus-Adaption

Moderne 32-Bit-Mikrocontroller besitzen auf dem Chip mehrere eigenständige Einheiten, die Befehle und Daten aus einem externen Speicher lesen bzw. schreiben müssen. Diese Einheiten sind z.B. der *CPU*-Kern, die Cache-Steuerung oder verschiedene *on-Chip*-Peripherie-Komponenten, welche in der Regel einen 32-Bit breiten Daten- und Adressbus besitzen.

Im Gegensatz dazu müssen vom Systementwickler außerhalb des Chips auf der Leiterplatte entsprechende Speicherkomponenten angeschlossen werden. Die speziellen Eigenschaften der unterschiedlichen Speichertechnologien werden in Abschnitt 2.1.2 beschrieben. Bei 32-Bit-Mikrocontrollern der ersten Generation, wie z.B. der Motorola 68K-Familie oder dem hyperstone E1, mußte diese Speicheranpassung durch zusätzliche externe Chips implementiert werden. Zum Einsatz kamen für die Adressdecoder und Bus-Umsetzer einfache programmierbare Bausteine und diskrete TTL-Bausteine. Die Nachteile einer solchen Lösung sind mehrere zusätzliche Bausteine (engl.: *glue logic*), hoher Platzbedarf und Kosten, mehr Leistungsverbrauch sowie hohe Fehleranfälligkeit. Zusätzlich liegen die Laufzeiten durch diese diskreten Bausteine im kritischen Pfad und verlängern die Zugriffszeiten der externen Speicher.

Bei 32-Bit-Mikrocontrollern der zweiten Generation, wie der hier vorgestellten *Embedded PowerPC*-Familie, ist diese Funktionalität auf dem Mikrocontroller-Chip integriert. Dabei bezeichnet man

- die Zerlegung des gesamten 32-Bit-Adressraumes in verschiedene Bereiche, die sogenannten Bänke (realisiert durch frei programmierbare Adressdecoder in entsprechenden Registern),
- die frei programmierbare Zuweisung einer unterschiedlichen Datenbreite (8-Bit, 16-Bit oder 32-Bit) zu jeder der definierten Bänke (in entsprechenden Registern),
- die frei programmierbare Zuweisung verschiedener Speichertechnologien zu jeder einzelnen Bank (in entsprechenden Registern) und
- die frei programmierbare Zuweisung von Zeitparametern für die einzelnen Speichertechnologien (in entsprechenden Registern)
- sowie die automatische und vollständige Generierung der aus diesen Werten abgeleiteten Ansteuersignale für die jeweiligen Speicherbänke mit dem richtigen logischen und zeitlichen Verhalten

als **dynamische Bus-Adaption (engl.: *dynamic bus sizing*)**. Dazu wurde ein Speicher-Controller (engl.: *memory controller*) integriert, der den wichtigsten Bestandteil einer übergeordneten Einheit, der sogenannten System-Schnittstelleneinheit (engl.: *System Interface Unit - SIU*), bildet. Diese *SIU* regelt den gesamten Datenaustausch mit der Außenwelt. Betrachtet man die *PowerPC*-Familien, so besitzt der Motorola MPC860 den am vielseitigsten programmierbaren Speicher-Controller, wie in [Mot98/1] dargestellt wird.

Durch die freie Konfigurierbarkeit verfügt der MPC860 bzw. der MPC850 über einen sehr flexiblen Speicher-Controller, um bestehende und zukünftige Speichermedien ohne externe Hardware zu bedienen, wie Tabelle 2.2 zeigt. Auf der anderen Seite liegt der Entwicklungsaufwand in der sehr aufwendigen Initialisierung der Zustandsmaschinen innerhalb der *SIU*. Diese sogenannte Firmware ist ein sehr kritischer Entwicklungsschritt. Bei einer falschen Programmierung arbeitet das System nicht fehlerfrei bzw. beendet nicht einmal den Initialisierungsvorgang (engl.: *boot-up sequence*). Damit kann auch keine Test-Software gestartet werden, wodurch die Fehlerdiagnose sehr schwierig und zeitaufwendig wird. Deshalb wird die Lösung dieser Problematik insbesondere auch in der Entwurfsmethode berücksichtigt (siehe Kapitel 4 und Kapitel 5), welche in dieser Arbeit vorgestellt wird.

Die beiden anderen *PowerPC*-Familien benötigen weniger Firmware-Aufwand bei der Initialisierung des Speicher-Controllers. Sie haben jedoch wesentliche Einschränkungen in der Anzahl der verschiedenen ansteuerbaren Speicherbausteine, die der Systementwickler bei dem Entwurf der Speicherarchitektur berücksichtigen muß. Diese Einschränkungen sind häufig auf den ersten Blick im Datenblatt nicht zu erkennen und bedürfen einer genauen Analyse und Interpretation der Bedeutung der Bits in den entsprechenden Konfigurationsregistern. Einen Überblick über die Einschränkungen der einzelnen Familien zeigt Tabelle 2.2.

Speichertechnologie	PPC 403GA	PPC 403GC X	MPC 505	MPC 509	MPC 555	MPC 801	MPC 850	MPC 860
Boot-EPROM	8/16/32	8/16/32	8/16/32	8/16/32	8/16/32	8/16/32	8/16/32	8/16/32
8 Bit asyn. SRAM	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
16 Bit asyn. SRAM	Nein	Ja	Ja	Ja	Ja	Ja	Ja	Ja
sync. SRAM	Nein	Ja	Ja	Ja	Ja	Ja	Ja	Ja
asyn. DRAM	Ja	Ja	Nein	Nein	Nein	Ja	Ja	Ja
sync. DRAM	Nein	Nein	Nein	Nein	Nein	Ja	Ja	Ja
Bus-Sig. frei konfigurierbar	Nein	Nein	Nein	Nein	Nein	Nein	Ja	Ja

Tabelle 2.2: Unterstützte Speichertechnologien bei den *Embedded PowerPC*-Familien

Speichertechnologie	PPC 403GA	PPC 403GC X	MPC 505	MPC 509	MPC 555	MPC 801	MPC 850	MPC 860
CPU/Bustakt [MHz]	40/40	80/40	25/25	44/44	40/40	40/40	66/66	66/66

Tabelle 2.2: Unterstützte Speichertechnologien bei den *Embedded PowerPC*-Familien

On-Chip Peripherie

Die Rechenleistung der *CPU* sowie die Anbindung des Mikrocontrollers an externe Speicher durch die *SIU* sind nur zwei der drei wesentlichen Leistungsmerkmale moderner 32-Bit-Mikrocontroller. Der dritte Punkt ist die Verfügbarkeit dedizierter *on-Chip*-Peripherie-Komponenten. Bei diesen Kriterien repräsentieren insbesondere die beiden Familien MPC8xx [Mot98/1] und MPC5xx [Mot98/2] den heutigen Stand der Technik und werden im folgenden Unterabschnitt näher betrachtet.

Der sogenannte MPC860-*PowerQUICC* ist ein Hochleistungs-Mikrocontroller, der im Bereich der industriellen Telekommunikation eingesetzt wird. Seine gesamte *on-Chip*-Peripherie wurde in einem gemeinsamen Kommunikation-Prozessor-Modul (engl.: *Communication Processor Module - CPM*) zusammengefaßt. Dabei verfügt auch dieses Modul über einen eigenen *RISC-CPU*-Kern, welcher über ein 5 KByte *Dual-Port-RAM* mit dem eigentlichen *PowerPC-CPU*-Kern kommuniziert. Der *RISC*-Controller kann spezifische Anwendungsalgorithmen ausführen, z.B. Berechnungen mit Hilfe der Multiplikation-Additions-Einheit (engl.: *Multiply Accumulate unit - MAC*) oder den Datenstrom über die zugeordneten seriellen Schnittstellen kontrollieren. Die Funktionsweise dieser verschiedenen seriellen Kommunikations-Controller (engl.: *Serial Communication Controller - SCC*) kann hier aus Platzgründen nicht beschrieben werden. Wichtig ist die Tatsache, daß auch diese, wie bei dem beschriebenen Speicher-Controller, auf unterschiedlichste Protokolle frei konfiguriert werden können. Damit können weite Bereiche der eingeführten seriellen Kommunikationsprotokolle, wie z. B. *PROFIBUS*, *HDLC* oder *Ethernet*, sowie auch zukünftige Protokolle physikalisch realisiert werden. Dazu ist eine sehr spezialisierte Firmware notwendig, deren Entwicklung nur von Spezialisten durchgeführt werden kann. Daraus folgt für die Entwurfsmethodik, daß vor allem die parallele Entwicklung der

- übergeordneten Software auf dem *PowerPC*-Prozessor-Kern,
- die Hardware-Entwicklung des gesamten eingebetteten Systems und
- die Firmware-Entwicklung für die *on-Chip*-Peripherie

ermöglicht werden muß. Auch hier zeigt sich, daß die Initialisierung eines der Hauptproblemfelder darstellt, welches möglichst getrennt von den anderen Problemen zu lösen ist. Diese Zielsetzung wird in dieser Arbeit verfolgt und in Kapitel 4 beschrieben. Als geeignetes Hilfsmittel wird in den folgenden Kapiteln die Emulation unter realen Umgebungsbedingungen benutzt.

Als weiteres markantes Beispiel eines solchen Mikrocontrollers, welcher in der Literatur mittlerweile auch als komplettes eingebettetes „System auf einem Chip“ (engl.: *System on a Chip - SoC*) bezeichnet wird, ist der MPC555. Er ist in der 32-Bit-Mikrocontroller-Klasse die Fortsetzung des im 16-Bit-Bereich eingeführten C167. Neben der eigentlichen *RISC-PowerPC-CPU (RCPU)* verfügt er über umfangreiche *on-Chip*-Speichermedien, nämlich über 448 KByte

Flash und 26 KByte schnelles *SRAM*. Ein weiterer wesentlicher Unterschied zu den *PowerPC-CPU*-Kernen in der *MPC8xx* und der *PPC40x*-Familie ist die integrierte Gleitkomma-Rechen-*einheit* (engl.: *Floating Point Unit - FPU*). Damit kann dieser Mikrocontroller einen umfangreichen Programmcode auf dem Mikrocontroller-Chip speichern. Seine *on-Chip*-Peripherie zeichnet sich im wesentlichen durch einen Analog-Digital-Wandler (*QADC*) sowie ein *CAN*-Modul aus. Diese beiden Einheiten bestimmen auch sein Haupteinsatzgebiet. Er ermöglicht die Digitalisierung und Filterung von analogen Prozeßsignalen durch Hardware-*FPU*-Unterstützung sowie die Kommunikation mehrerer Mikrocontroller über ein *CAN*-Netzwerk. Das gesamte eingebettete System kann in den meisten Anwendungsfällen als Einzelchip-Lösung realisiert werden. Diese Fähigkeiten werden hauptsächlich im Automotive-Bereich gefordert, wo hohe Zuverlässigkeit, niedriger Stromverbrauch und geringe Baugrößen als Randbedingungen vorgegeben werden.

***On-Chip*-Befehls- und Daten-Caches**

Mit der Einführung der Pipeline-Verarbeitung beim C167 stieg der Befehlsdurchsatz und damit der Bedarf an Bandbreite beim Speicherzugriff drastisch an. Während man beim C167 dieses Problem durch doppelt breite Busse zwischen den internen Speicherbereichen und der *ALU* noch lösen konnte, ist diese Methode bei den 32-Bit-Mikrocontrollern nicht mehr möglich. Obwohl bei diesen Mikrocontrollern ein 32-Bit breiter Daten- und Adressbus zu den schnellen externen Speicherbänken (z.B. *sync. SRAM* oder *sync. DRAM*) vorhanden sein kann, stellt die bereits eingeführte *SIU* einen Engpaß dar. Die benötigte Bandbreite der *on-Chip*-Peripherie und der *CPU* steigt überproportional stark an. Um die Rechenleistung der *CPU* dennoch optimal auszunutzen, werden bei den meisten heute auf dem Markt verfügbaren 32-Bit-Mikrocontrollern Caches mit auf den Chip integriert, um die Zugriffe der *CPU* auf die externen Speicher zu minimieren.

Die *PowerPC*-Mikrocontroller verfügen auf dem Chip über eine sogenannte *Harvard*-Architektur, d.h. getrennte Befehls- und Datenbusse. Abhängig vom konkreten Typ stehen verschiedene Befehls- und Daten-Cache-Größen zur Verfügung (engl.: *Instruction Cache - I-Cache*, *Data Cache - D-Cache*). Die Grundlagen über den Aufbau von Caches und deren Einsatz innerhalb der Speicherhierarchie können in dieser Arbeit nicht dargestellt werden und sind ausführlich in [HePa96] zu finden. An dieser Stelle wird nur auf die speziellen Eigenschaften von Caches bei Mikrocontrollern sowie deren Bedeutung bei der Entwicklung von eingebetteten Systemen eingegangen. Die umfangreichste Funktionalität bei den Befehls-Caches innerhalb der *Embedded PowerPC*-Mikrocontroller hat der I-Cache in der *MPC5xx* und *MPC8xx*-Familie, welcher in Abbildung 2.3 dargestellt ist. Bei der Implementierung in der *PPC40x*-Familie fehlt die Möglichkeit, bestimmte Cache-Zeilen mit Hilfe eines sogenannten Lock-Bits fest im Cache zu speichern. Dafür sind die Caches beim *PPC40x* bis zu achtmal größer als beim *MPC8xx* bzw. *MPC5xx*.

Der I-Cache ist als sogenannter Zwei-Wege-Assoziativ-Cache (engl.: *two way associative cache*) aufgebaut. Die wesentliche Grundeinheit bilden die sogenannten Cache-Zeilen (engl.: *cache lines*), welche in diesem Fall aus insgesamt vier unmittelbar hintereinander folgenden 32-Bit breiten Befehlswörtern ($4 \times 4 \text{ Bytes} = 16 \text{ Bytes}$) bestehen. Diese Cache-Zeilen bilden praktisch ein Speicherabbild des externen Arbeitsspeichers im Cache. Zu jeder einzelnen Cache-Zeile gehört noch eine Adressinformation, das sogenannte *TAG*. Es wird durch den oberen Teil der Befehlsadresse gebildet, welche als Basisadresse (in diesem Fall die Bits 0 bis 20) der 16 Byte Cache-Zeile aufgefaßt werden kann. Ein Satz (engl.: *SET*) besteht aus jeweils zwei kompletten Cache-Zeilen. Diese Anordnung bezeichnet man auch als eine sogenannte Zwei-Wege-

Organisation, physikalisch implementiert in *WAY 0* und *WAY 1*. Der mittlere Teil der Befehlsadresse, die Bits 21 bis 27, selektieren jeweils einen der insgesamt 128 *SETs*.

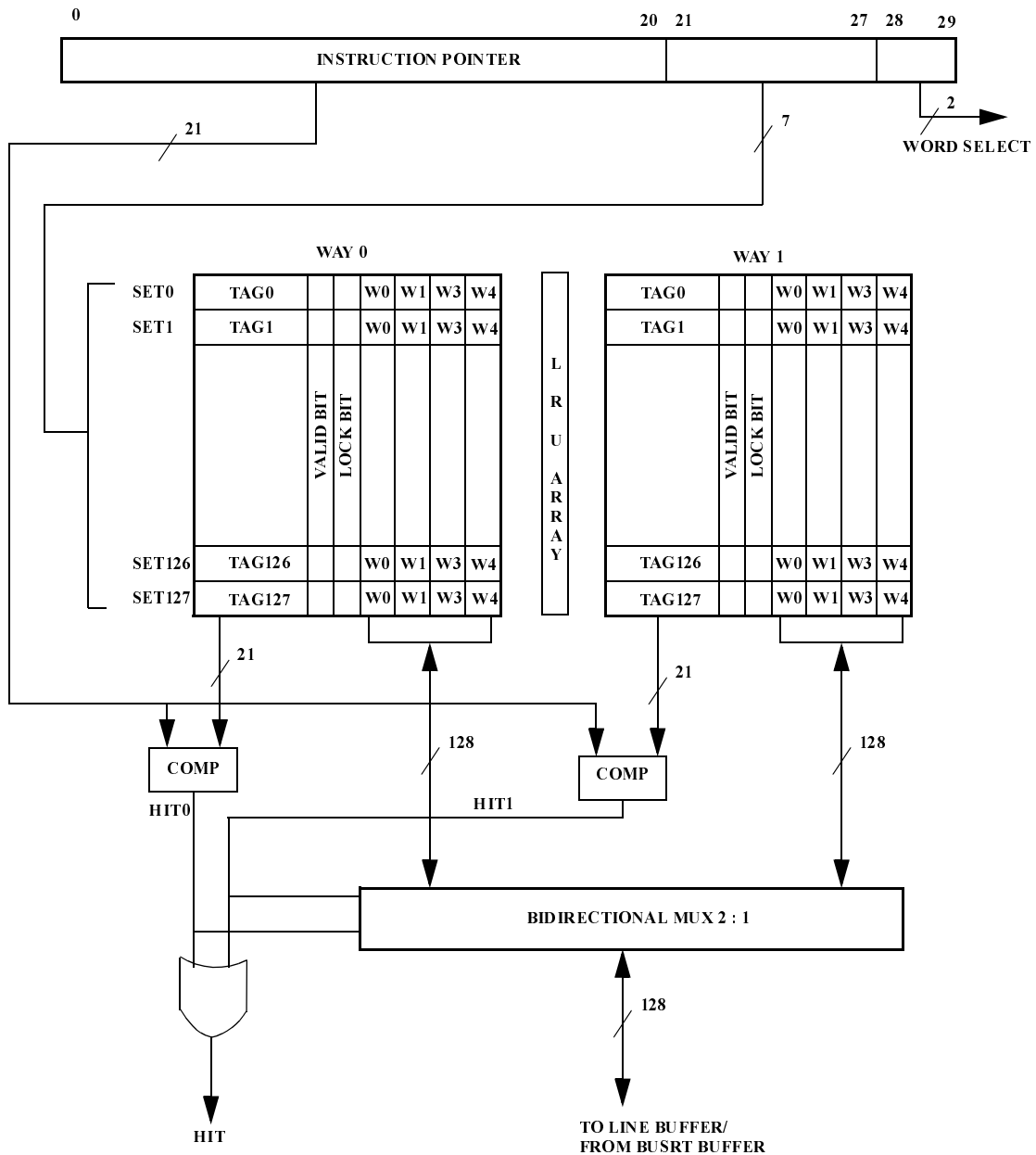


Abbildung 2.3 Organisation des Befehls-Cache beim MPC505 (Quelle: [Mot94])

Ergibt der Vergleich der Adresse des aktuell zu lesenden Befehls mit einem der beiden möglichen *TAGs* im selektierten *SET* eine Übereinstimmung, so liegt ein Cache-Treffer (engl.: *cache hit*) vor. Voraussetzung für den Treffer ist allerdings, daß das zu jeder Cache-Zeile gehörende Gültigkeits-Bit (engl.: *valid bit*) die Gültigkeit der gespeicherten Information bestätigt. In diesem Fall wählen dann die beiden weiteren Bits 28 und 29 eines der vier 32-Bit-Befehlsörter in der Cache-Zeile aus, welches dann im gleichen Takt zur Befehls-Ladeeinheit (engl.: *instruction fetch unit*) weiter transportiert wird.

Wenn keines der beiden *TAGs* eines selektierten *SETs* übereinstimmt oder das Gültigkeits-Bit den Inhalt der Cache-Zeile als „nicht gültig“ markiert, liegt kein Treffer vor (engl.: *cache miss*). In diesem Fall wird von der Cache-Steuerung eine komplette Cache-Zeile, bestehend aus

vier 32-Bit-Worten, vom externen Speicher nachgeladen. Dazu wird eine Leseanforderung an die *SIU* gesendet. Die 16 Bytes der Cache-Zeile werden zunächst im sogenannten *Burst*-Puffer (engl.: *burst buffer*) aufgesammelt. Dabei wird das gesuchte Zielwort zuerst geladen, anschließend folgen die restlichen drei Wörter. Erst wenn die gesamte Cache-Zeile vorhanden ist, wird sie zusammen mit dem entsprechenden *TAG* in den Cache eingetragen. Dabei versucht die Cache-Steuerung zuerst ungültige Cache-Zeilen zu füllen. Sind alle Zeilen belegt, wird die Cache-Zeile ersetzt, welche die längste Zeit nicht mehr benutzt wurde (engl.: *Least Recently Used - LRU*). Dieser Algorithmus basiert dabei auf der Korrelation der Lokalität des Programms, d.h. momentan benutzte Bereiche im Programm werden wahrscheinlich bald wieder benutzt. Daraus folgt, daß der am längsten nicht benutzte Programm-Code ersetzt werden kann.

Die Implementierung eines kompletten *LRU*-Algorithmus ist sehr aufwendig. Deshalb wird eine vereinfachte Realisierung benutzt. Bei jedem Ersetzen einer Cache-Zeile wird das *LRU*-Bit der gegenüberliegenden Cache-Zeile gesetzt. Dieses Bit bewirkt den Austausch der Zeile mit dem nächsten *cache miss*.

Auch hier wird von der Lokalität Gebrauch gemacht. Innerhalb eines *SETs* differieren die beiden *TAGs* der Cache-Zeilen um mindestens die halbe Cache-Größe, im Fall des MPC505 um 2 KByte. Das bedeutet, zwischen zwei unmittelbar hintereinander ersetzten Cache-Zeilen liegt mindestens eine Adress-Distanz von einer halben Cache-Größe. Damit wird verhindert, daß sequentiell angeordnete Befehle durch weiter entferntere Befehle ersetzt werden, weil die Wahrscheinlichkeit groß ist, daß dieser Bereich linear durchlaufen bzw. in einer Schleife mehrmals wiederholt wird.

Die Befehls-Caches der MPC50x und MPC80x zeichnen sich im Gegensatz zu den I-Caches in den PPC40x-Mikrocontrollern durch das sogenannte *Lock-Bit* aus, mit dem bestimmte Cache-Zeilen im Cache festgehalten werden können. Dadurch bekommt der Cache die Wirkung eines *on-Chip-SRAM*, in dem ein kritischer Programm-Code (z.B. eine *Interrupt-Service-Routine - ISR*) gespeichert werden kann.

Die Daten-Caches haben die gleiche Funktionsweise wie die Befehls-Caches. Bei Schreibzugriffen entsteht allerdings ein sogenanntes Kohärenz-Problem, d.h. wenn die Daten in der jeweiligen Cache-Zeile geändert werden. Damit sind sie nicht mehr identisch mit dem entsprechenden Datenbereich im externen Hauptspeicher. Die Steuerung der Daten-Caches bietet dafür zwei prinzipielle Lösungen an, nämlich den sogenannten „durchgreifenden“ Schreibzugriff (engl.: *write through*) und den „zurück kopierenden“ Zugriff (engl.: *copy back*). Im ersten Verfahren werden bei jedem Schreibzugriff die Daten in den Cache und gleichzeitig in den externen Hauptspeicher geschrieben. Der Nachteil dieser Methode besteht in der höheren Busbelastung, der Vorteil besteht im gleichen Datenabbild im Cache und im Hauptspeicher. Bei der zweiten Methode wird ein Schreibzugriff zunächst nur im Cache eingetragen und die Cache-Zeile mit Hilfe eines entsprechenden Bits als „verändert“ markiert (engl.: *dirty bit*). Erst beim Ersetzen der Cache-Zeile wird ihr Inhalt in den Hauptspeicher geschrieben. Damit sinkt die Busbelastung, wobei die Daten im Hauptspeicher nicht immer identisch mit dem Cache-Abbild sind. Diese Tatsache muß insbesondere bei Multi-Prozessor-Systemen beachtet werden.

In Tabelle 2.3 wird eine Übersicht über die aktuellen Cache-Größen der zur Zeit auf dem Markt verfügbaren *PowerPC*-Familien gegeben.

Alle Mikrocontroller-Typen sind als Zwei-Wege-Set-Assoziativ-Caches organisiert. Jede Cache-Zeile besteht dabei aus vier 32-Bit-Wörtern.

	PPC 403GA	PPC 403GC X	MPC 505	MPC 509	MPC 555	MPC 801	MPC 850	MPC 860
I-Cache Größe	2KB	16KB	4KB	4KB	-	2KB	2K	4K
I-Cache Lock-bar	Nein	Nein	Ja	Ja	-	Ja	Ja	Ja
D-Cache Größe	1KB	8KB	-	-	-	1KB	1K	4K
copy- back	Nein	Ja	-	-	-	Ja	Ja	Ja
write through	Ja	Ja	-	-	-	Ja	Ja	Ja
SRAM anstatt D-Cache	Nein	Nein	4KB	4KB	26K	Nein	Nein	Nein

Tabelle 2.3: Cache-Größen und Eigenschaften bei den *PowerPC*-Mikrocontrollern

Beim Einsatz von Mikrocontrollern in eingebetteten Systemen kann die Cache-Wirkung die Ausführungsgeschwindigkeit um den Faktor zehn steigern. Bei Systemen, die unter harten Echtzeitanforderungen stehen, kann diese Steigerung allerdings nicht in allen Fällen garantiert werden, sondern nur im Durchschnitt. Deshalb muß man den schlechtesten Fall ermitteln (engl.: *worst case analysis*), welcher ohne die Cache-Wirkung vorliegt. Eine solche Analyse ist auch Gegenstand dieser Arbeit und wird in Kapitel 6 am Anwendungsbeispiel *ASI-Master* durchgeführt. Zusätzlich wird dort gezeigt, welchen Einfluß bestimmte Cache-Größen und Konfigurationen auf wichtige Kenngrößen eines Echtzeitbetriebssystems haben. Insbesondere der PPC403GCX mit seinen extrem großen Caches wird in Abhängigkeit von verschiedenen Konfigurationen signifikante Leistungsunterschiede zeigen.

Sprung-Vorhersage und spekulatives Laden (engl.: *branch prediction, speculative fetches*)

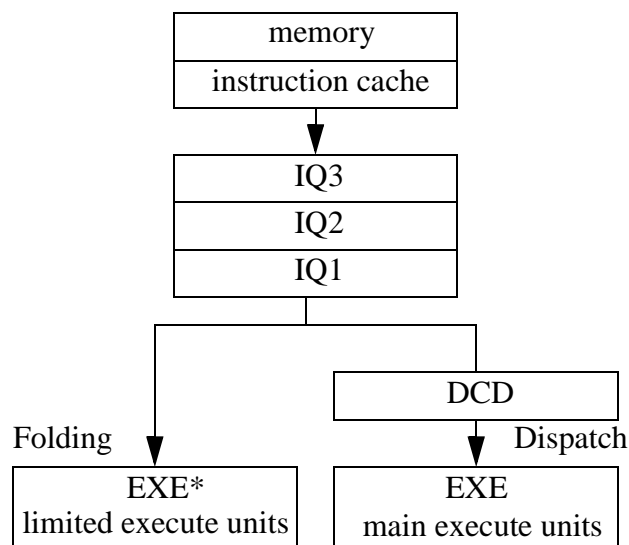


Abbildung 2.4 Befehls-Warteschlange beim IBM *PowerPC* PPC403 (Quelle: [IBM98])

Um den folgenden Unterabschnitt zu erläutern, wird in Abbildung 2.4 die Befehls-Warteschlange des PPC403 dargestellt. Im Gegensatz zu den 8-Bit- und 16-Bit-Mikrocontrollern bestehen die *RISC-CPU*-Kerne aus mehreren Ausführungseinheiten, die prinzipiell parallel arbeiten können. Die Hauptausführungseinheiten (engl.: *main execute units*) bestehen aus dem allgemeinen Registersatz (R0-R31), dem speziellen Registersatz (engl.: *special function registers*), der eigentlichen *ALU*, einer Multiplikations-Einheit sowie vier Zeitgebern.

Die Befehle werden in die Befehls-Warteschlange über den Befehls-Cache eingelesen. Auch wenn Befehle nicht im Cache gehalten werden dürfen, werden diese aus dem Speicher eingelesen und von der Befehls-Cache-Steuerung direkt in die Warteschlange weitergeleitet. Wenn die Schlange leer ist, werden die neuen Befehle direkt in die sogenannte Dekodier-Einheit (engl.: *DeCoDe stage - DCD*) weitergegeben. Die Weiterleitung von Befehlen aus der *DCD*-Einheit in die Ausführungseinheiten (*EXE*) bezeichnet man beim PPC403 als sogenanntes „*Dispatch*“.

Zusätzlich zu den Hauptausführungseinheiten gibt es noch in ihrer Funktionalität eingeschränkte Ausführungseinheiten (engl.: *limited execute units*). Beim PPC403 führen diese Einheiten im wesentlichen die bedingten und unbedingten Sprungbefehle aus. Wenn in der Befehls-Warteschlange mindestens zwei Befehle vorhanden sind, wird die zweite Operation von der Stufe *IQ1* direkt in die begrenzte Ausführungseinheit *EXE** übertragen. Dafür stehen auch physikalisch zwei getrennte Datenpfade zur Verfügung, die das gleichzeitige Ausgeben von zwei Befehlen (einer nach *DCD*, einer nach *EXE**) erlauben. Das Übertragen des Befehles von *IQ1* nach *EXE** bezeichnet man auch als „*Folding*“.

Im allgemeinen bezeichnet man die gleichzeitige Ausgabe von mehr als einem Befehl als Superskalarität (engl.: *superscalar operation*). In der Regel haben die meisten Befehle der *PowerPC*-Architektur eine Ausführungsdauer von einem Taktzyklus. Manche Befehle, wie zum Beispiel die Lade- und Speicherbefehle oder auch komplexere arithmetische Befehle, dauern länger als einen Taktzyklus. Dadurch könnte es passieren, daß Befehle mit kurzer Ausführungsdauer, die in der sequentiellen Ausführungsordnung hinter einem mehrere Takte dauernden Befehl angeordnet sind, diesen langen Befehl quasi „überholen“. Werden diese Befehle ausgeführt, spricht man von einer sogenannten „Ausführung außerhalb der Programmreihenfolge“ (engl.: *out-of-order execution*), welche bei der *MPC5xx*-Familie implementiert wurde. Mit diesem Verhalten sind prinzipielle Probleme verbunden, welche an dieser Stelle aus Platzgründen nicht weiter dargestellt und in [Mot94] beschrieben werden. Der hier betrachtete PPC403 besitzt zwar Superskalarität, erzwingt aber beim Ausführen der beiden gleichzeitig ausgegebenen Befehle, daß sie sich in der Ausführung nicht überholen. Man spricht deshalb von der sogenannten „Ausführung in der Programmreihenfolge“ (engl.: *in-order execution*).

Bei Sprungbefehlen wird die sequentielle Ausführung an einer bestimmten Stelle gestoppt und an einer anderen Stelle fortgesetzt. Bei bedingten Sprüngen besteht das Problem, daß beim Eintreffen des Befehles in der entsprechenden Ausführungseinheit noch nicht bestimmt werden kann, ob der Sprung genommen wird oder nicht. Um den Befehlsfluß mit möglichst wenig Stillstand ablaufen zu lassen, kann nicht gewartet werden, bis die entscheidende Sprungbedingung eingetroffen ist. Deswegen muß die Ausführungseinheit eine Vorhersage treffen, ob der Sprung genommen wird oder nicht (engl.: *branch prediction*).

Der Hauptgrund bei der Durchführung von Sprung-Vorhersagen besteht darin, möglichst frühzeitig neue Befehle aus einem langsamen externen Speicher zu laden. Dabei bezeichnet man das Laden von Befehlen nach einem noch nicht vollständig aufgelösten bedingten Sprung

als spekulatives Laden (engl.: *speculative fetch*). Dabei müssen folgende grundsätzliche Probleme berücksichtigt werden:

- Beim spekulativen Laden besteht lediglich eine große Wahrscheinlichkeit, daß der Programmablauf nach dem bedingten Sprung die vorhergesagte Richtung nimmt. Ist diese Richtung nicht richtig vorhergesagt, sind die gelesenen Befehle wertlos und müssen verworfen werden.
- Werden diese Befehle nicht nur gelesen, sondern anschließend auch ausgeführt, z. B. bei *out-of-order execution*, entstehen erhebliche Probleme, insbesondere in dem Fall, daß sich die Vorhersage als falsch herausstellt. In diesem Fall könnten die spekulativ gelesenen und ausgeführten Befehle den Prozessorzustand verändert haben (z.B. allgemeine Register, Statusregister usw.). Wenn die Vorhersage falsch war, muß der alte Prozessorzustand wieder hergestellt werden. Dazu sind umfangreiche Hardware-Einrichtungen notwendig, die ebenfalls in [Mot94] erklärt werden. Für den hier behandelten PPC403 gilt, daß er nur spekulatives Laden von Befehlen durchführt, um diese in der Befehls-Warteschlange bereit zu halten. Er führt diese Befehle aber erst aus, wenn der bedingte Sprung abgearbeitet ist, d.h. wegen seiner prinzipiellen „*in-order execution*“ vollständig aufgelöst ist. Damit entfällt hier das Problem, den im Falle einer falschen Vorhersage veränderten Prozessorzustand wieder herzustellen.
- Spekulative Zugriffe können auch zu Fehlern bei bestimmten I/O-Komponenten führen. Zur Verdeutlichung dient als Beispiel eine serielle Schnittstelle, bei der der Zugriff auf das Statusregister automatisch bestimmte Status-Bits zurücksetzt. Angenommen, eine Code-Sequenz, welche auf das Statusregister zugreifen soll, wird durch einen bedingten Sprung oder auch einen Interrupt vor dem eigentlichen Statusregisterzugriff unterbrochen. In diesem Fall werden nach dem Eintreffen des bedingten Sprunges weiter spekulative Ladezugriffe durchgeführt, unter anderem auch der Zugriffsbefehl auf das Statusregister. Liegt dieser Zugriffsbefehl in der vorhergesagten Richtung, welche aber dann tatsächlich nicht genommen wird, käme es zu einem Verlust der Informationen im Statusregister, wenn der Befehl zur Ausführung gebracht würde. Da der PPC403 keine *out-of-order execution* erlaubt, kann auch dieser Fall bei diesem Mikrocontroller kein Fehlerverhalten auslösen.

Die prinzipiellen Risiken solcher optimierten Ausführungseinheiten müssen dem Entwickler bekannt sein, um bei diesen Architekturen nicht einen fehlerhaften Code zu schreiben. Diese Fehler sind in der Praxis extrem schwer zu lokalisieren.

Test-Unterstützung auf dem Chip (engl.: *on-chip debugging*)

In Abbildung 2.5 ist das Blockdiagramm des Embedded PowerPC403GCX von IBM dargestellt. Er bietet zwei verschiedene *debug*-Möglichkeiten an, welche über die serielle *on-Chip*-Schnittstelle und den sogenannten *Test Access Port (JTAG)* der Außenwelt zugänglich gemacht wurden. Der *JTAG*-Port wurde in dem IEEE-Dokument 1149.1 genormt. Bei den *PowerPC*-Familien MPC5xx und MPC8xx existiert ebenfalls eine integrierte *debug*-Schnittstelle (engl.: *on-chip development port*), die allerdings ein firmenspezifisches Protokoll besitzt. Im folgenden Text werden diese *debug*-Methoden am Beispiel des PPC403 erläutert.

Im wesentlichen unterscheidet man zwischen der internen und externen *debug*-Methode. Die interne *debug*-Methode basiert auf sogenannten *debug*-Ausnahme-Situationen (engl.: *debug exceptions*). Solche *debug exceptions* kann man sich in ihrer Wirkung wie externe *Interrupts* vorstellen, die den normalen Programmfluß unterbrechen und an einer definierten Speicher-

adresse mit der Bearbeitung der *debug exception* fortfahren. Die Software-Routine, welche die *debug exception* behandelt, wird als *exception handler* bezeichnet. Prinzipiell kann diese Routine den gesamten internen Prozessor-Zustand, d.h. alle seine Register und interne Speicher bzw. Cache-Inhalte, über eine serielle Schnittstelle zu einem Entwicklungsrechner übertragen. Auf diesem Entwicklungsrechner muß eine spezielle Software laufen, welche mit dem *exception handler* kommuniziert und die Information übersichtlich darstellt. Auf diese Weise können in einem Speicherfenster auch externe Speicherbereiche angezeigt werden. Es ist zusätzlich möglich, die aktuellen Register und Speicherwerte nicht nur anzuzeigen, sondern sie von der Oberfläche des Entwicklungsrechners aus auch gezielt zu verändern. Ein Anwendungsprogramm kann entweder im Einzelschritt ausgeführt oder durch das Setzen von Unterbrechungspunkten (engl.: *break points*) gezielt an bestimmten Programmstellen angehalten werden. Dazu werden im Anwendungsprogramm-Code an den gewünschten Unterbrechungsstellen spezielle Sprungbefehle (engl.: *exection traps, instructions tw and twi*) eingefügt, welche den Programmfluß unterbrechen. Die CPU setzt die Ausführung sofort mit der *exception handler*-Routine fort und sendet den aktuellen Zustand des Prozessors zum Entwicklungsrechner bzw. erwartet von der Benutzeroberfläche weitere Kommandos.

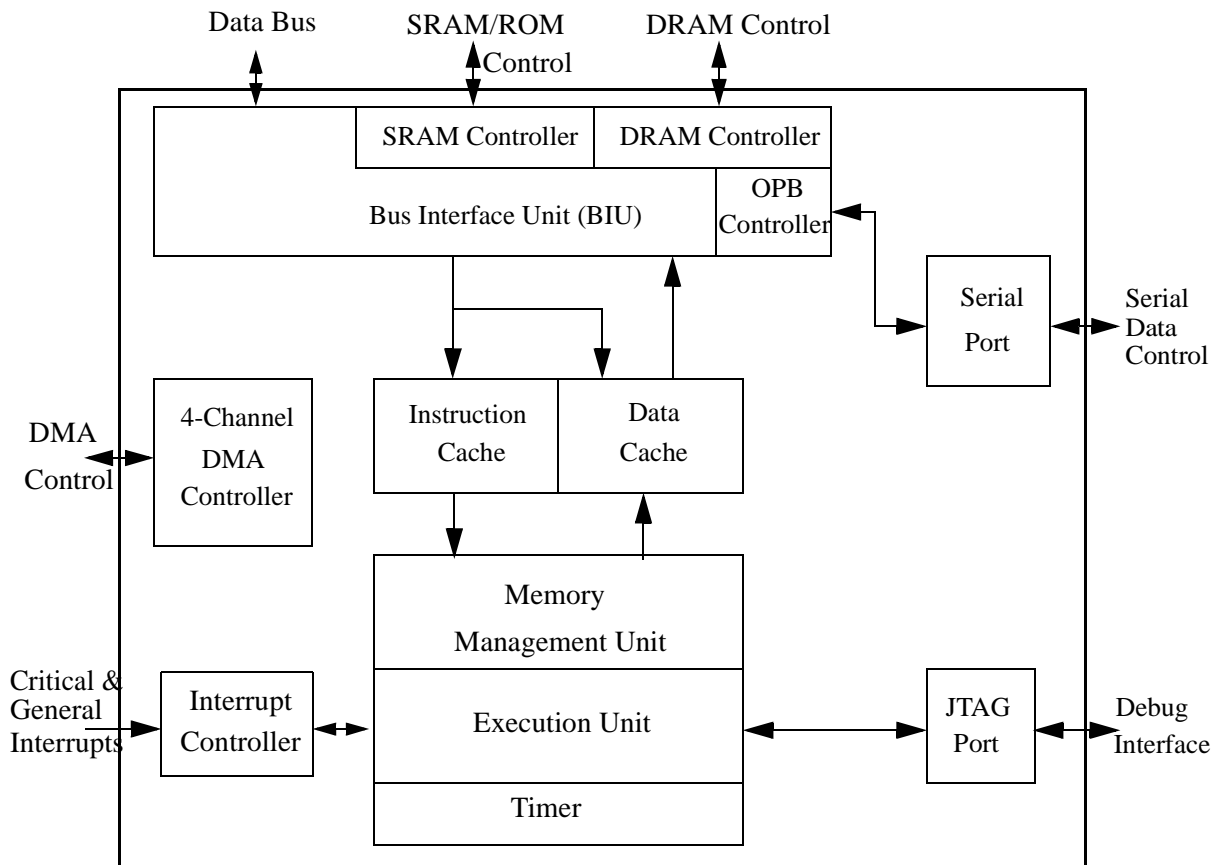


Abbildung 2.5 Blockdiagramm des *PowerPC* PPC403GCX (Quelle: [IBM98])

Der entscheidende Vorteil dieser *debug*-Methode basiert im wesentlichen darauf, daß sie nur aus Software besteht und keine besonderen externen Hardware-Einrichtungen benötigt. Sie nutzt von der Hardware-Architektur des Mikrocontrollers die entsprechende *debug exception*. Diese auf dem Mikrocontroller laufende Software verfügt über weitere Funktionalitäten, wie z.B. das Laden des Programmcodes vom Entwicklungsrechner auf das Zielsystem, und wird in seiner Gesamtheit auch als sogenannter Software-Monitor bezeichnet. Dazu wird als Kommunikationsschnittstelle beim PPC403 die serielle *on-Chip*-Schnittstelle genutzt. Ein Beispiel für

diese *debug*-Methode wird in Abschnitt 4.1.2 beschrieben. Der Nachteil liegt darin, daß zum Betrieb der externe Speicherbus funktionieren muß, um den Code des Monitors ausführen zu können. Bei den Prototyp-Platinen für komplexe 32-Bit-Mikrocontroller-Systeme ist diese Voraussetzung, wie in Abschnitt 2.1.5 gezeigt, aber keinesfalls selbstverständlich. Dadurch scheidet diese Methode für die ersten *debug*-Tests auf neuen Platinen häufig aus. Ein weiterer Nachteil liegt darin, daß diese Methode bei der Programmausführung in den Code des Anwendungsprogrammes eingreift und ihn so in seiner Ausführungsdauer verändert. Dadurch ist kein *debugging* möglich, welches auch das korrekte Testen des Echtzeitverhaltens erlaubt.

Die externe Methode eliminiert diese genannten Nachteile der internen Methode und basiert im wesentlichen auf der Nutzung von speziellen Hardware-Ressourcen für das *debugging* auf dem Mikrocontroller-Chip. Die externe Methode unterstützt dabei das Starten und Stoppen der CPU, den Zugriff auf alle internen Prozessor-Ressourcen sowie deren Veränderung, das Setzen von Hardware- und Software-Haltepunkten sowie die Beobachtung des aktuellen Prozessorstatus, ohne dabei das Echtzeitverhalten zu beeinflussen.

Diese Methode erlaubt die Definition von bestimmten *debug*-Ereignissen (engl.: *debug events*), deren Eintreffen von internen Hardware-Einheiten überwacht wird. Im Falle des Eintreffens eines *debug events* wird der gesamte Mikrocontroller angehalten. Diesen Zustand bezeichnet man auch als „eingefroren“, d.h. die Programmausführung wird gestoppt und alle Architektur-Ressourcen, wie Register und *on-Chip*-Speicher, halten ihren momentanen Zustand. Diese Informationen können dann über den *JTAG-Port* ausgelesen bzw. gezielt verändert werden.

Der PP403 befindet sich bei dieser Methode im Echtzeit-Beobachtungs-*debug*-Mode (engl.: *real-time trace debug mode*). Durch die Hardware-Unterstützung ändert er die Ausführungsgeschwindigkeit des Anwendungsprogrammes nicht. Mit Hilfe des Zugriffs über den *JTAG-Port* läßt sich der Prozessor zusätzlich kontrollieren. In den vorherigen Abschnitten wurde der Begriff der superskalaren Ausführung durch *Folding* eingeführt. Diese Funktionalität kann durch die *debug*-Register zum Test abschaltet werden.

Eine weitere wichtige Eigenschaft ist die Fähigkeit, Befehle über den *JTAG-Port* zu laden und auszuführen (engl.: *instruction stuff*). Damit können zum Test Programmsequenzen gestartet werden, auch wenn der externe Speicher nicht funktioniert. Diese Fähigkeit ist sehr wichtig bei Prototypen, bei denen über die Funktion der Platine oder der externen Speicher keine genaue Aussage getroffen werden kann.

Zusätzlich werden auch die Funktionen wie Einzelschritt-Ausführung, Haltepunkt-Setzen und verschiedene Reset-Funktionen über den *JTAG-Port* kontrolliert.

Prinzipiell sind Zugriffe auf interne Peripherie-Komponenten und Caches außerhalb des Chips nicht sichtbar. Einige Mikrocontroller besitzen eine Betriebsart, bei denen diese Zugriffe für das *Debugging* über die Schnittstelleneinheit (engl.: *Bus Interface Unit - BIU*) nach außen übertragen werden (engl.: *show internal bus cycle mode*). Diese Möglichkeit reduziert aber erheblich die Ausführungsgeschwindigkeit und verhindert so die Echtzeitfähigkeit. Der PPC403 besitzt dazu einen weiteren *Port*, mit dessen Hilfe er durch sechs Statussignale interne Ereignisse anzeigt. Diese Signale können direkt mit einem Logik Analysator (LA) verbunden werden, welcher diese Signale in Echtzeit aufzeichnet und interpretiert. Um diese Fähigkeit zu nutzen, muß der sogenannte Bus-Status-*debug*-Mode eingeschaltet werden.

Implementierungsanforderungen bei 32-Bit-Mikrocontrollern

Bei den 32-Bit-Mikrocontrollern erzwingen die Verbreiterung der Adress- und Datenbusse sowie die höheren Taktfrequenzen die Verwendung von komplexen Gehäuseformen. Häufig zu finden sind SMD-Gehäuse mit bis zu 300 Anschluß-Pins bei einem Pin-Abstand von 0,5 mm (z.B. bei den PPC40x und MPC5xx). Bei Mikrocontrollern mit einer extrem hohen Anzahl von *on-Chip*-Peripherie-Komponenten, welche ihrerseits mit der Außenwelt kommunizieren (z.B. bei den MPC8xx) müssen, haben sich sogenannte *Ball-Grid-Array*-Gehäuse mit bis zu 600 Anschluß-Pins auf ca. 5 cm² durchgesetzt. Diese hohe Signal-Pin-Dichte in Verbindung mit Busfrequenzen von bis zu 66 MHz erzwingen Leiterplatten mit bis zu 20 verschiedenen elektrischen Signallagen in Feinstleiteteknik. Diese Randbedingungen stellen enorme Anforderungen an die Fertigungstechnologie. Diese Eigenschaft wird in Abschnitt 2.1.5 näher beschrieben bzw. in der vorgestellten Entwurfsmethodik (siehe Kapitel 4) in Form eines eigenständigen Entscheidungsfeldes besonders berücksichtigt.

Ausblick bei den 32-Bit-Mikrocontrollern

Wie bereits in der Einleitung zu diesem Abschnitt erwähnt, wird in aktuellen Entwicklungen auch der *PowerPC 740/750* bei eingebetteten Systemen mit extremen Leistungsanforderungen in steigendem Maße eingesetzt. Seine Architektur ist typisch für einen Hochleistungsmikroprozessor, da er über keine anwendungsspezifische *on-Chip*-Peripherie wie ein Mikrocontroller verfügt. Im Gegensatz dazu besitzt er sehr große *on-Chip*-Caches (32 KByte Befehle und 32 KByte Daten) sowie viele parallele Ausführungseinheiten (zwei *Integer*-Einheiten, eine 64-Bit *FPU*, *Branch-Prediction*-Einheit, *Load/Store*-Einheit, System-Register-Einheit) und ist in einem Taktfrequenzbereich von 200 MHz bis 400 MHz verfügbar. Seine Bus-Schnittstelle hat einen 64-Bit breiten Datenbus und einen 32 Bit breiten Adressbus. Diese Busschnittstelle ist auf den Anschluß externer Cache-Speicher (engl.: *second level caches - L2 Cache*) optimiert.

Diese Architekturmerkmale ermöglichen eine Rechenleistung, die deutlich über der Rechenleistung momentan verfügbarer Mikrocontroller liegt und derjenigen der allgemeinen Rechnertechnik (*PC* oder *Workstation*) vergleichbar ist. Im Gegensatz dazu kann er keine langsameren externe Peripherie-Komponenten direkt anschließen, welche aber bei den meisten eingebetteten Systemen zusätzlich gebraucht werden. Deshalb wird er in vielen eingebetteten Systemen mit einem weiteren hochintegrierten *VLSI*-Chip (engl.: *party chip*) verbunden, welcher diese Anpassung des Busverhaltens zur Verfügung stellt und zusätzlich ein Vielzahl von Standard-Peripherie-Komponenten (*Interrupt*-Controller, serielle und parallele Schnittstellen, *Memory*-Controller usw.) integriert. Beide Chips, der *PowerPC740/750* und der *party chip*, bilden zusammen einen sogenannten Chip-Satz, welcher sehr viele Anschluß-Pins benötigt. Dadurch werden auch entsprechend hohe Anforderungen an die Fertigungstechnologie gestellt. Daraus folgt, daß solche Lösungen zur Zeit und in absehbarer Zukunft nur für relativ teure eingebettete Systeme der oberen Leistungsklasse eingesetzt werden können.

Von den Herstellern werden bereits 32-Bit-Mikrocontroller angekündigt (z.B. der PPC405 von IBM), die ab dem Jahr 2000 auf dem Markt eingeführt werden sollen. Das wichtigste Kennzeichen ist, daß durch eine Verkleinerung der Fertigungstechnologie auf Strukturbreiten von 0,25 µm die Taktfrequenz auf ca. 250 MHz ansteigen wird. Dieser Sprung bedeutet eine Steigerung von einem Faktor acht bis zehn gegenüber den heutigen Mikrocontrollern. Der im Bereich der allgemeinen Rechnertechnik bereits zum Standard gewordene sogenannte *Peripheral Component Interconnect (PCI) local bus* wird als weitere Systemschnittstelle direkt mit auf den Mikrocontroller-Chip integriert werden. Er zeichnet sich durch eine hohe Bandbreite sowie eine eindeutige Festlegung seines logischen und elektrischen Busverhaltens aus. Dadurch sollen die

Bus-Schnittstellen zwischen Mikrocontroller und externer Peripherie vereinheitlicht und so die Intergration weiterer *VLSI-ASIC* auf Systemebene erleichtert werden.

2.1.2 Halbleiterspeicher-Bausteine

Im vorangehenden Abschnitt über die Mikrocontroller wurden häufig die Fähigkeiten beim Anschluß externer Halbleiterspeicher im Zusammenhang mit der Einführung des Speicher-Controllers angesprochen. In diesem Abschnitt werden die auf dem Markt verfügbaren Halbleiterspeicher eingeführt, soweit sie für eingebettete Systeme von Bedeutung sind. Die wesentlichen Eigenschaften bezüglich ihrer Funktion, ihres Busverhaltens und ihrer Zugriffsgeschwindigkeiten werden erklärt sowie ihr Anwendungsfeld innerhalb der eingebetteten Systeme beschrieben.

2.1.2.1 Asynchrone Festwertspeicher (*ROM, EPROM, Flash*)

Festwertspeicher auf Basis der Halbleitertechnologie werden in eingebetteten Systemen zur Speicherung des gesamten Programm-Codes sowie aller festen Datenstrukturen benutzt. Ihre wesentliche Eigenschaft besteht in der Fähigkeit, die programmierte Information auch nach dem Abschalten der Betriebsspannung zu speichern. Im normalen Betrieb werden sie von einem Mikrocontroller mit einem asynchronen Zugriff nur ausgelesen, nicht aber beschrieben. Deshalb bezeichnet man sie auch als „nur-Lesespeicher“ (engl.: *Read Only Memory - ROM*). Die verschiedenen Typen unterscheiden sich in der Regel nur in der Art und Weise, wie die Information in dem Baustein gespeichert wird.

Read Only Memory (ROM)

ROM-Bausteine sind prinzipiell nicht löschar und werden während der Chip-Fertigung durch entsprechende Masken gleich mit der gewünschten Information produziert. Eine andere Möglichkeit bietet die Hilfe eines externen Programmiergerätes, welches die Speicherinformation in den Baustein „einbrennt“ (engl.: *One Time Programmable - OTP*). Die Speichermatrix besteht dabei aus einfachen Dioden, welche mit starken elektrischen Strömen durchgebrannt werden können. Damit wird die Diode praktisch zerstört und so in den hochohmigen Zustand gebracht. Dadurch kann ein komplementäres Informations-Bit gegenüber dem unzerstörten Zustand gespeichert werden. Die Informationsspeicherung kann deshalb auch nicht mehr gelöscht werden. Bei einer Programmänderung muß das *ROM* aus der Schaltung entfernt werden. Es ist danach nicht mehr zu verwenden.

UV-löschbares und programmierbares ROM (UV Eraseble Programmable ROM)

Bei eingebetteten Systemen sind durch ultraviolettes Licht löscharbare Festwertspeicher weit verbreitet und werden *UV-EPROM* genannt. Die Speicherzelle besteht bei diesen Bausteinen aus einer Transistorzelle, deren Basis durch Silizium-Oxid völlig isoliert ist. Die Programmierung wird mit Hilfe eines externen Programmiergerätes durchgeführt. Dabei wird durch eine Programmierspannung von ca. 14 V zwischen dem Kollektor des Speichertransistors und der isolierten Basis ein großes elektrisches Feld erzeugt. In diesem Feld nehmen die Elektronen eine große Energie auf, so daß sie die Isolierschicht durchdringen können und im Basisbereich eine Raumladung erzeugen, die den Transistor durchschaltet. Nach dem Abschalten der Programmierspannung bleibt die Raumladung erhalten, wodurch ein Informations-Bit gespeichert wird.

Der eigentliche Silizium-Chip ist durch eine durchsichtige Glasscheibe abgedeckt. Durch diese Scheibe kann ultraviolettes Licht den Chip bestrahlen. Aufgrund dieser Bestrahlung nehmen die in der Basiszone gespeicherten Elektronen wieder soviel Energie auf, daß sie das Isoliermaterial durchdringen können und von der Basis abwandern. Dadurch wird die Speicherinformation wieder gelöscht. Eine tiefergehende Beschreibung der physikalischen Vorgänge ist in [MöLu87] zu finden. Dieser Löschvorgang dauert je nach Intensität des UV-Lichtes ca. 15 Minuten, wobei die Glasscheibe im normalen Betrieb durch ein Schutzschild abgedeckt werden soll.

Der Speicherzugriff ist, wie in Abbildung 2.6 dargestellt, asynchron, d.h. es gibt keinen direkten Bezug zu einem Bustakt-Signal. Zur Beschreibung des zeitlichen Verlaufes eines asynchronen Speicherzugriffes sind drei Zeitwerte charakteristisch:

- **Zugriffszeit t_{acc} :** Die Zugriffszeit ist der Zeitraum vom Anlegen der Adressen bzw. des sogenannten *Chip-Select*-Signals (CS_- - *low aktiv*) durch den Mikrocontroller bis der Baustein die gespeicherten Daten am Datenausgang stabil zur Verfügung stellt. Bei *EPROM*-Bausteinen liegt dieser Wert im Bereich zwischen 70 ns und 150 ns.
- **Lesezeit t_{read} :** Die Lesezeit ist der Zeitraum vom Anlegen des Lesesignals (engl.: *Output Enable* - OE_- - *low aktiv*) durch den Mikrocontroller bis die Daten am Datenausgang stabil zur Verfügung stehen. Dieser Wert liegt im Bereich von 40 ns bis 50 ns.
- **Haltezeit t_{hold} :** Die Haltezeit ist der Zeitraum, welche der Baustein braucht, um nach dem Abschalten der Steuersignale (CS_- und OE_-) seine Datentreiber in den hochohmigen Zustand zu schalten und so den Mikrocontroller-Bus für den nächsten Zugriff freizugeben. Dieser Wert liegt im Bereich zwischen 35 ns und 40 ns und ist heute nur bei *EPROM*-Bausteinen größer als null. Bei asynchronen *SRAM*-Bausteinen ist dieser Wert in der Regel null.

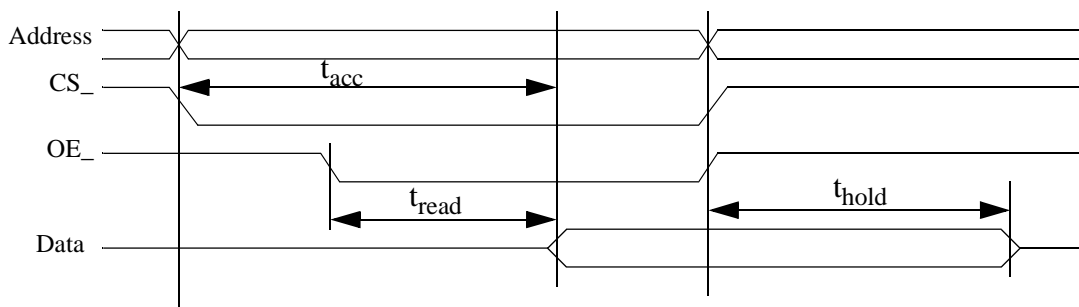


Abbildung 2.6 Asynchroner Lesezugriff auf einen externen Festwertspeicher

Diese Grundlagen wurden eingeführt, um damit eine Klassifizierung der Speicher nach Zugriffsgeschwindigkeit und Bandbreite durchführen zu können. Die Gesamtdauer eines Zugriffs ist somit die Summe aus $t_{acc} + t_{hold}$ und liegt bei heutigen *EPROM*-Bausteinen zwischen 105 ns und 190 ns. Die Speichergrößen liegen in Bereichen zwischen 16 KByte und 1 MByte, wobei die Speicherbreite in der Regel bei 8-Bit liegt. Daraus errechnet sich eine maximale Zugriffsbandbreite beim Lesen von ca. 10 MByte/sec. Die Programmierzeit für ein Byte in einem Programmiergerät liegt bei 100 μ s, d.h. ein 1 MByte großes *EPROM* kann in ca. 100 sec beschrieben werden.

Sektor-orientierter Festwertspeicher (engl.: *Flash-Memory*)

Die in eingebetteten Systemen eingesetzten Festwertspeicher werden zur Zeit durch sogenannte *Flash*-Speicher ersetzt. Diese verhalten sich beim Abschalten der Betriebsspannung ebenfalls als „nicht-flüchtige“ Festwertspeicher und basieren auf den gleichen physikalischen Grundlagen (Fowler-Nordheim-Tunneleffekt) während des Löschvorganges und der Elektronen-Injektion beim Schreibvorgang wie die *EPROM*-Bausteine.

Ihr Vorteil gegenüber einem *EPROM* liegt darin, daß sie kein externes Programmiergerät bzw. eine höhere Programmierspannung als die normale System-Betriebsspannung (3,3 V bzw. 5 V) benötigen und deshalb von einem Mikrocontroller mit speziellen Schreibzugriffssequenzen beschrieben werden können. Das Schreiben der *Flash*-Speicher ist allerdings nicht vergleichbar mit Schreiboperationen auf einem *SRAM*-Baustein, sondern unterliegt wesentlichen Einschränkungen, welche nur in seltenen Fällen der Programmierung angewendet werden können. Aus Platzgründen kann der Schreibvorgang nicht im Detail beschrieben werden und ist in [AMD98] zu finden. Durch die starken elektrischen Felder während der Schreib- bzw. Löschvorgänge an den Sperrschichten der Halbleiter garantieren die Hersteller nur 100.000 Programmiervorgänge und geben die mittlere Lebensdauer mit 1. Mio an. Die Löschdauer eines Chips der Größe von 512 KByte liegt im Bereich zwischen 8 sec und 64 sec.

Beim normalen Lesezugriff sind die *Flash*-Speicher wesentlich schneller. Die Werte für die bereits eingeführten Zugriffszeiten liegen etwas unter denen der *EPROM*-Bausteine ($t_{acc} = 55 \text{ ns} - 150 \text{ ns}$, $t_{read} = 30 \text{ ns} - 55 \text{ ns}$, $t_{hold} = 18 - 35 \text{ ns}$). Damit ergeben sich Lesezykluszeiten ($t_{acc} + t_{hold}$) zwischen 73 ns und 185 ns und somit eine Bandbreite von 13,7 MByte/sec und 5,4 MByte/sec.

2.1.2.2 Statische Arbeitsspeicher (*SRAM* und *SSRAM*)

Ein wesentlicher Bestandteil eines Mikrocontroller-Kerns sind neben den Festwertspeichern auch die Arbeitsspeicher. Diese können von einem Mikrocontroller im Gegensatz zu Festwertspeichern mit einer wesentlich schnelleren Zugriffsgeschwindigkeit geschrieben und gelesen werden. Deshalb bezeichnet man sie auch als Speicher mit wahlfreiem Zugriff (engl.: *Random Access Memory - RAM*). Sie dienen hauptsächlich zum Speichern von dynamischen Programmdateien.

Asynchrone statische Arbeitsspeicher (engl.: *SRAM*)

In Abbildung 2.7 ist ein asynchroner Schreibzugriff auf einen *SRAM*-Arbeitsspeicher dargestellt.

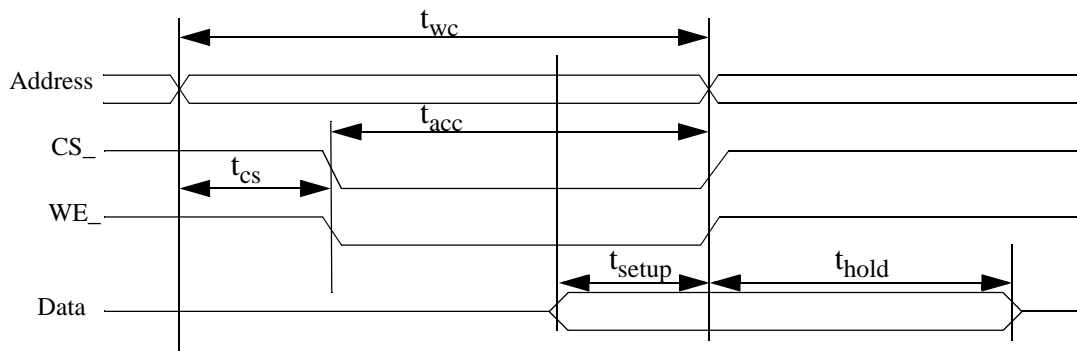


Abbildung 2.7 Asyn. Schreibzugriff auf einen externen *SRAM*-Arbeitsspeicher (Quelle: [Par98])

Der wichtigste Parameter ist die Schreibzykluszeit t_{wc} . Diese Zeit beschreibt die gesamte Dauer eines Zugriffs einer Schreiboperation und liegt bei *SRAM*-Bausteinen im Bereich von 10 ns bis 30 ns. Asynchrone *SRAM* sind in einer Wortbreite von 8-Bit bzw. 16-Bit auf dem Markt verfügbar. Betrachtet man ein 16-Bit breites *SRAM* (2 Byte) mit 1 Meg-Adressraum (2 MByte Speicherkapazität) und geht von der schnellsten Zugriffszeit von 10 ns (100 MHz) aus, so errechnet sich eine maximale Bandbreite von 2 Byte x 100 MHz = 200 MByte/sec. Die weiteren Parameter sind in [Par98] zu finden.

Synchrone statische Arbeitsspeicher (engl.: *SSRAM*)

Um die prinzipielle Funktionsweise eines synchronen *SRAM* zu erläutern, werden in Abbildung 2.8 in vereinfachter Form die wichtigsten Bus-Signale bei einem Schreibzugriff dargestellt. Der Lesezugriff hat ein ähnliches Verhalten und ist im Detail in [Mic98/1] zu finden.

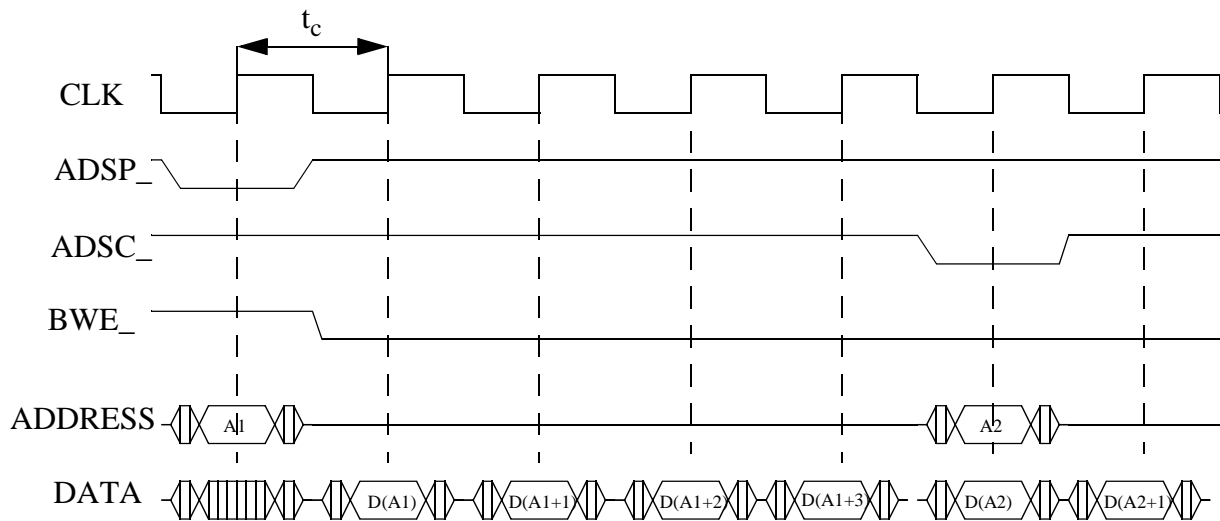


Abbildung 2.8 Burst-Schreibzugriff eines *Pipeline-Sync-SRAM* (Quelle: [Mic98/1])

Der wesentliche Unterschied liegt darin, daß alle Signale einen Bezug zu einem externen Bustakt (*CLK*) haben. Beim heutigen Stand der Technik liegen die Taktzeiten t_c im Bereich zwischen 5 ns (200 MHz) und 10 ns (100 MHz). Die maximal verfügbare Speichergröße beträgt dabei 8 MBit in einer Organisation von 256 K x 32-Bit. Prinzipiell ist in jedem externen Buszyklus eine Zugriffsoperation möglich, lediglich beim Umschalten von der Lese-Richtung in die Schreib-Richtung wird zum Umladen des Busses ein Zwischentakt notwendig. Deshalb ist die optimale Betriebsart der sogenannte *Burst-Zugriff*, bei dem mindestens vier Lese- bzw. Schreibzugriffe hintereinander erfolgen. Damit ergibt sich eine maximale Zugriffsbandbreite von 4 Byte Busbreite x 200 MHz = 800 MByte/sec. Damit stellen diese *SSRAM*-Bausteine die zur Zeit schnellsten externen Speicherbausteine dar. Sie werden in der Architektur eingebetteter Systeme als externe Cache-Speicher (engl.: *second level cache*) benutzt. Zur Zeit sind nur wenige Speichercontroller in der Lage, diese *SSRAM*-Chips direkt anzusprechen (siehe Tabelle 2.2). Die *Burst*-Tiefe von 4 x 32-Bit-Zugriffen entspricht dabei exakt der *Burst*-Tiefe der *on-Chip*-Cache-Controller, welche bei einem *on-Chip-cache-miss* eine ganze Cache-Zeile mit ebenfalls 4 x 32-Bit-Worte nachladen bzw. im Falle eines *on-Chip*-Daten-Cache eine Cache-Zeile mit 4 x 32-Bit-Worte schreiben. Diese *SSRAM*-Chips eignen sich daher besonders gut als sogenannte *second level-Cache* in der Zusammenarbeit mit den *first level-on-Chip-Caches*.

2.1.2.3 Dynamische Arbeitsspeicher (*DRAM* und *SDRAM*)

Dynamische Arbeitsspeicher basieren auf der sogenannten Ein-MOS-Transistorzelle und sind damit wesentlich höher integrierbar als die Sechs-MOS-Transistorzelle der *SRAM*-Bausteine. Der Nachteil dieser Lösung liegt in dem permanenten Ladungsverlust, welcher den Speicherkondensator mit der Zeit entlädt. Dieser Ladungs- und somit Informationsverlust muß in einem bestimmten Zeitfenster ersetzt werden. Dieser Prozeß besteht im wesentlichen durch ein zyklisches Auslesen und wieder neu Einschreiben der Information in die Speicherzelle (engl.: *refresh*). Wegen dieses permanenten Ladungsverlustes sowie der Auffrischung der Information bezeichnet man diese Art der Informationsspeicherung in einem Arbeitsspeicher auch als dynamischen Speicher mit wahlfreiem Zugriff (engl.: *Dynamic Random Access Memory - DRAM*).

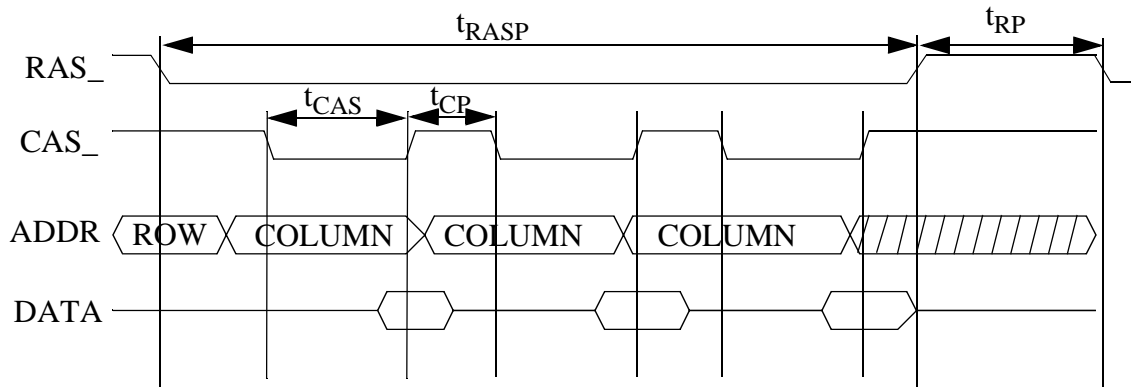


Abbildung 2.9 *DRAM-Fast-Page-Mode* Lesezyklus (Quelle: [Mic98/2])

Asynchrone dynamische Arbeitsspeicher mit wahlfreiem Zugriff (*Asyn. DRAM*)

Asynchrone *DRAM* besitzen eine Speicherkapazität von bis zu 16 MBit pro Chip. In Abbildung 2.9 wird in vereinfachter Form der Zugriff auf einen *DRAM*-Baustein dargestellt. Die Erklärung des Zugriffsprotokolles sowie der Zeitparameter sind in [Mic98/2] zu finden. Der ungünstigste Fall liegt vor, wenn in einer Zugriffssequenz die jeweilige Differenz zweier Zugriffe größer als die Seitengröße der *DRAM*-Bank ist. Dann liegt bei jedem Zugriff eine sogenannte Seitenunterbrechung (engl.: *page break*) vor, d.h. es muß jedesmal eine Übertragung der Reihen- und Spaltenadresse durchgeführt werden. Bei der Einhaltung der minimalen Zeitwerte für $t_{RASP} + t_{RP}$ errechnet sich eine Zugriffszeit von $50 \text{ ns} + 30 \text{ ns} = 80 \text{ ns}$. Bei einer typischen Bausteinbreite von 2 Byte ergibt sich eine Bandbreite von $1/80 \times 10^{-9} \text{ sec} \times 2 \text{ Byte} = 25 \text{ MByte/sec}$ pro 16-Bit breiten Baustein bzw. von 50 MByte/sec bei zwei parallelen Chips, um eine 32-Bit breite Bank zu bilden.

Betrachtet man den günstigsten Fall bei sequentiellen Zugriffen innerhalb einer einzigen Seitenadresse, so ergibt sich die Zeit für einen Zugriff aus der Addition der Zeiten t_{CA} (8 ns) und t_{CP} (8 ns) und liegt bei 16 ns. Die Bandbreite für einen 16-Bit breiten *DRAM*-Baustein errechnet sich aus $1/16 \times 10^{-9} \text{ sec} \times 2 \text{ Byte} = 125 \text{ MByte/sec}$ oder bei einer 32-Bit breiten Bank (bestehend aus zwei parallelen Chips) bei 250 MByte/sec.

Aus dieser Betrachtung ergibt sich ein Faktor von fünf für die Schwankung der Übertragungsbandbreite zwischen dem minimalen und maximalen Wert.

Synchrone dynamische Arbeitsspeicher mit wahlfreiem Zugriff (*SDRAM*)

Die maximale Speicherkapazität pro Chip liegt heute bei 64 MBit pro Chip. Beim Übergang von 16 MBit auf 64 MBit wurde dabei nicht nur die Kapazität um den Faktor vier gesteigert.

gert, sondern durch Änderung des asynchronen Busverhaltens in ein synchrones Busverhalten auch eine Geschwindigkeitserhöhung erreicht.

Um das prinzipielle Verhalten zu erklären, ist in Abbildung 2.10 ein vereinfachter *Burst*-Lesezugriff auf einen synchronen *DRAM*-Speicher dargestellt. Charakteristisch ist das Taktsignal (*CLK*) mit einer Zykluszeit zwischen 7 ns - 8 ns (143 MHz - 125 MHz). Alle anderen Bus-signale müssen synchron zu diesem Signal eine vorgegebene *Setup*- (2 ns) und *Hold*-Zeit (1 ns) einhalten.

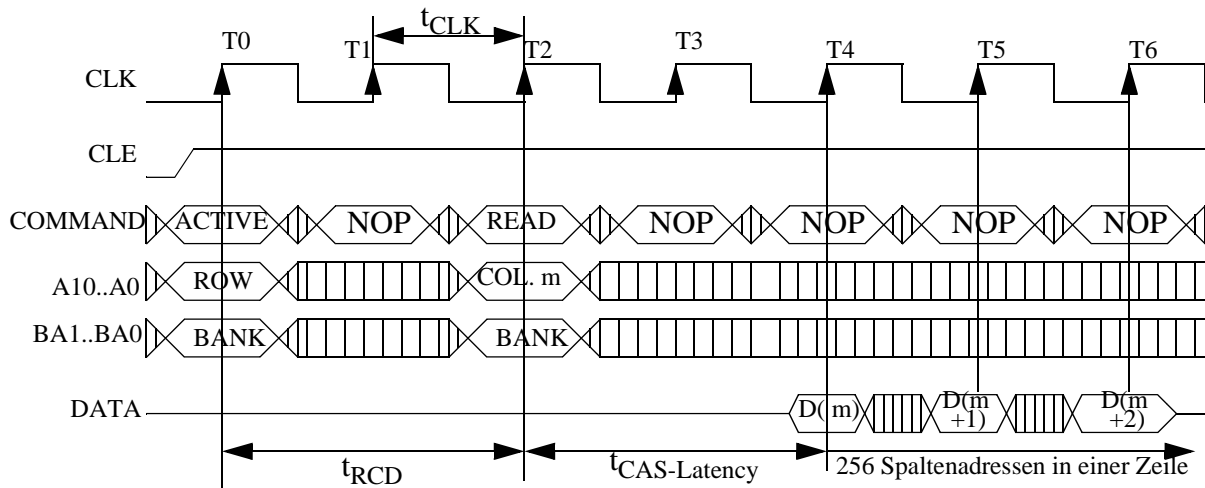


Abbildung 2.10 *Burst*-Lesezugriff auf einen synchronen *DRAM* (Quelle: [Mic98/3])

Das Zugriffsprotokoll und die charakteristischen Zeitparameter sind in [Mic98/3] erklärt. Prinzipiell kann innerhalb einer Seite in jedem Takt ein 32-Bit breites Datenwort gelesen bzw. geschrieben werden. Bei einer maximalen Taktfrequenz von 143 MHz (7 ns) und 4 Byte Datenbreite berechnet sich die Bandbreite aus $1/7 \times 10^{-9} \text{ sec} \times 4 \text{ Byte} = 571 \text{ MByte/sec}$. Diese Bandbreite liegt fast im Bereich der *SSRAM*-Bausteine und ermöglicht die Implementierung eines externen Cache-Speichers bzw. eines sehr schnellen Arbeitsspeichers. Die Entwicklung der *SDRAM* wurde vor allem im Bereich der allgemeinen Rechnertechnik vorangetrieben. Im Bereich der eingebetteten Systeme sind zur Zeit nur die Speicher-Controller der 32-Bit-Mikrocontroller MPC8xx-Familie in der Lage, diese Bausteine ohne externe Bus-Controller-Chips direkt anzusteuern. In den nächsten Jahren kann erwartet werden, daß die *SDRAM* die asynchronen *DRAM* auch im Bereich der eingebetteten Systeme ablösen werden.

2.1.2.4 Zusammenfassung der Halbleiterspeicher

Abschließend werden in Tabelle 2.4 die wichtigsten Kenndaten und Einsatzspektren der Halbleiterspeicher angegeben. Die Werte für die typischen Zugriffszeiten gelten bei den Festwertspeichern *ROM/EPROM* und *Flash* nur für Lesezugriffe. Die zweite Spalte zeigt die zur Zeit maximal verfügbaren Speichergrößen und Konfigurationen. Darunter ist jeweils die Anzahl der parallel zu schaltenden Bausteine angegeben, um auf eine Datenbreite von 32-Bit zu kommen.

Die nächste Spalte zeigt die maximale Bandbreite bei Zugriffen auf diese Speicherbausteine. In Klammern wird jeweils der normierte Wert bezogen auf eine 32-Bit breite Bank angegeben. Die letzte Spalte beschreibt die wichtigsten Anwendungen der verschiedenen Speicher-

technologien innerhalb von Architekturen eingebetteter Systeme. In diesem Zusammenhang wird auch auf die Tabelle 2.2 verwiesen, in der die Unterstützung der verschiedenen Speichertechnologien bei verschiedenen Mikrocontrollern angegeben ist.

	typische Zykluszeiten	maximale Speichergröße	maximale Bandbreite	typische Verwendung in eingebetteten Systemen
<i>ROM/EPROM</i>	105 ns - 190 ns	1 Meg x 8 32 Bit: 4 Chips	10 MB/sec (40 MB/sec)	Kritischer <i>boot-up</i> Code
<i>Flash</i>	73 ns - 185 ns	2 Meg x 8 32 Bit: 4 Chips	13,7 MB/sec (54,8 MB/sec)	Anwendungs-Code, <i>RTOS</i> -Code
Asyn. <i>SRAM</i>	10 ns - 30 ns	1 Meg x 16 32 Bit: 2 Chips	200 MB/sec (400 MB/sec)	schneller Arbeitsspeicher
Sync. <i>SRAM</i>	5 ns - 10 ns	256 K x 32 32 Bit: 1 Chip	800 MB/sec (800 MB/sec)	externer Cache
Asyn. <i>DRAM</i>	16 ns - 80 ns	1 Meg x 16 32 Bit: 2 Chips	100 MB/sec (250 MB/sec)	Arbeitsspeicher
Sync. <i>DRAM</i>	7 ns - 8 ns	2 Meg x 32 32 Bit: 1 Chip	572 MB/sec (572 MB/sec)	Arbeitsspeicher bzw. externer Cache

Tabelle 2.4: Halbleiterspeicher und deren Verwendung bei eingebetteten Systemen

2.1.3 Programmierbare Logik-Bausteine

Neben den Mikrocontrollern und Halbleiterspeichern werden beim Entwurf eingebetteter Systeme auch digitale Schaltungsteile benötigt, welche sehr spezifisch von der dedizierten Anwendung abhängen und deshalb als Standardkomponenten nicht verfügbar sind. Die Größe dieser Schaltungen variiert dabei von wenigen Logik-Gattern bis zu mehreren 100.000 Gatter-Äquivalenten. Für die Implementierung dieser Funktionen haben sich vom Anwender frei programmierbare Komponenten durchgesetzt. In diesem Abschnitt werden die wichtigsten Architekturen programmierbarer Logik-Bausteine vorgestellt. Anschließend werden die damit verbundenen Entwurfsmethoden beschrieben und ihre Bedeutung für den Architekturentwurf eingebetteter Systeme diskutiert. Dabei wird deutlich, daß diese Komponenten einen wesentlichen Beitrag leisten, um die Emulation von eingebetteten Systemen durchzuführen. Diese Komponenten sind wesentliche Bestandteile der in Kapitel 5 vorgestellten Emulationsplattform bzw. dienen als Zieltechnologien für die Anwendungsbeispiele in Kapitel 6.

2.1.3.1 Komplexe programmierbare Logik-Bausteine

In eingebetteten Systemen besteht häufig die Notwendigkeit, aus Bus-Signalen des Mikrocontrollers bzw. Peripheriekomponenten entsprechende Steuersignale abzuleiten. Typische Anwendungen sind die Generierung von Chip-Auswahlsignalen (engl.: *Chip Select* - *CS*) zur Aktivierung von Speicher- bzw. *ASIC*-Chips oder die Kontrolle von gemeinsamen Bussen durch eine entsprechende Verwaltungseinheit (engl.: *arbiter*). Die Schaltung besteht entweder aus kombinatorischer Logik oder aus Zustandsmaschinen. Zur Implementierung dieser Funktionen werden in der Regel sogenannte komplexe programmierbare Logik-Bausteine (engl.:

Complex Programmable Logic Devices - CPLD) eingesetzt. In Abbildung 2.11 ist die Architektur der XC9500-Familie dargestellt.

Diese CPLD-Architektur in Abbildung 2.11 besteht im wesentlichen aus bis zu 16 *Function Blocks (FB)* (je nach Bausteingröße), einer *FastCONNECT Switch Matrix* sowie bis zu 192 *I/O-Blocks (I/OB)*. In [Xil98/1] wird die Architektur im Detail erläutert.

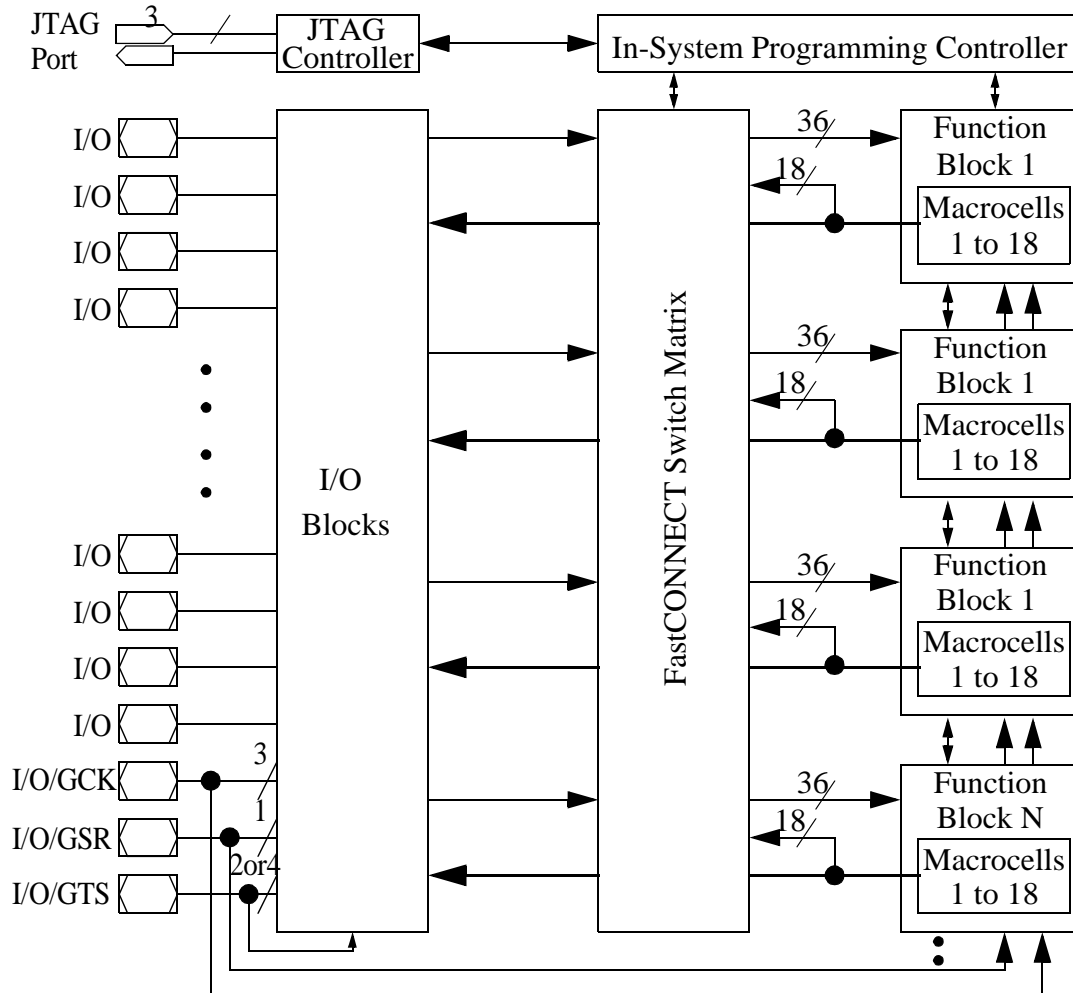


Abbildung 2.11 Xilinx XC9500 CPLD-Architektur (Quelle: [Xil98/1])

Das Einsatzspektrum der *CPLD* innerhalb der eingebetteten Systeme deckt hauptsächlich den Bereich der sehr schnellen Koppellogik (engl.: *glue logic*) ab. Dabei werden im Vergleich zu den in den nächsten beiden Abschnitten beschriebenen *FPGA* nur wenig Logikgatter gebraucht, allerdings mit relativ vielen *I/OB*. Typische Anwendungen sind schnelle und breite Adressdecoder, Zähler und Zustandsmaschinen. Der wesentliche Vorteil gegenüber den *FPGA* liegt in der schnellen Durchlaufzeit von Signalen durch das ganze *CPLD*-Chip mit Verzögerungszeiten von Pin-zu-Pin in 5 ns. Dieser Wert liegt etwa um den Faktor vier bis fünf unter dem Wert bei einem *FPGA*. Auf Grund der sehr regelmäßigen Architektur kann man zusätzlich die Verzögerungszeit relativ genau bestimmen. Im Gegensatz zu den *FPGA* ist die Verzögerungszeit deterministisch, weil sich die Durchlaufzeit aus festen Werten für die *I/OB*, *Switch Matrix*, und *FB* zusammensetzt.

2.1.3.2 SRAM-basierte Anwender-programmierbare Schaltungen

Um der Forderung nach höherer Integrationsdichte zu entsprechen, wurden in den letzten Jahren die Anwender-programmierbaren Schaltungen (engl.: *Field Programmable Gate Arrays - FPGA*) im Bezug auf Logik-Kapazität und Schaltgeschwindigkeit signifikant weiterentwickelt. Wesentlicher Vorteil der *FPGA* ist dabei die Eigenschaft, daß der Anwender die Funktionalität selbst bestimmen und implementieren kann. Diese Tatsache ermöglicht beim Architektorentwurf für eingebettete Systeme neue Möglichkeiten, z.B. zur Emulation von Schaltungsteilen, die insbesondere auch in dieser Arbeit genutzt werden. In den folgenden Abschnitten werden die wichtigsten Architekturen vorgestellt und anschließend die damit verbundenen Entwurfsmethoden diskutiert.

XC4000-Architektur (Xilinx)

Die prinzipielle Architektur der SRAM-basierten XC4000-*FPGA* zeigt Abbildung 2.12, eine genaue Beschreibung ist in [Xil98/1] zu finden. In einer *Switch-Matrix* können aneinanderstoßende Leitungssegmente in der gleichen Richtung verbunden oder eine Richtungsänderung durchgeführt werden. Dazu stehen Verdrahtungskanäle in vertikaler und horizontaler Richtung zur Verfügung (H/V-Verdrahtung).

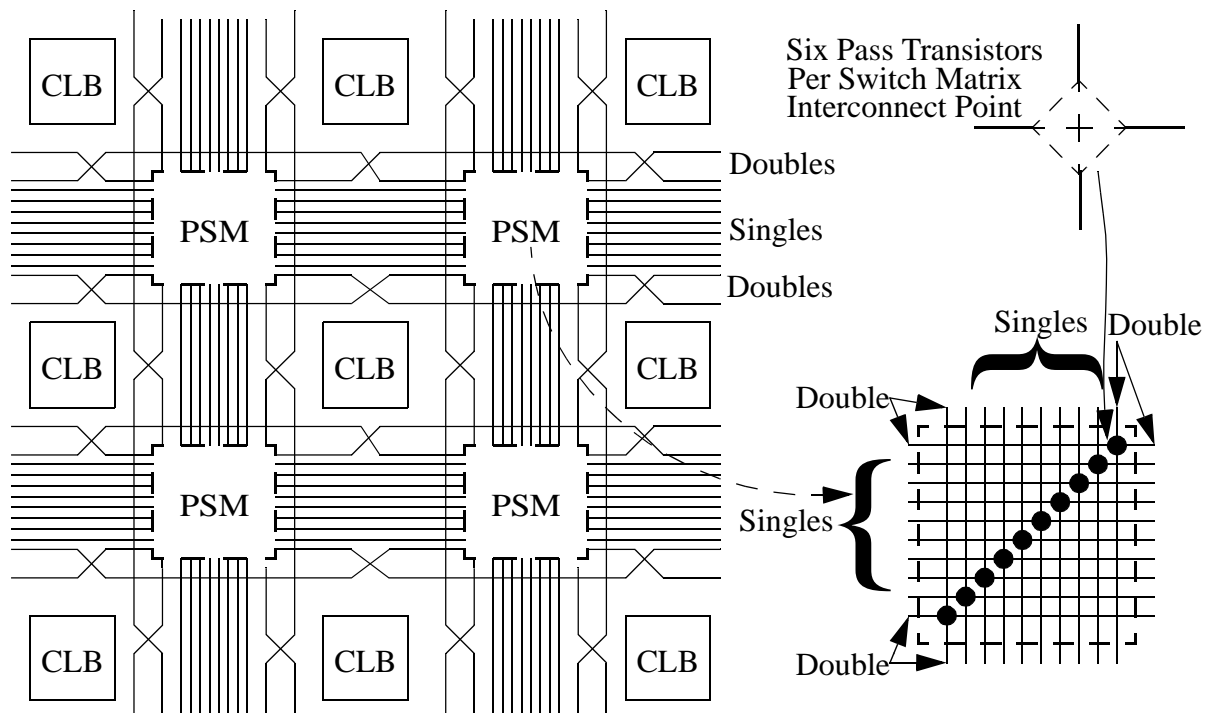


Abbildung 2.12 Single-Double-Length-Lines und programmierbare *Switch-Matrix* (Quelle: [Xil98/1])

In den sogenannten konfigurierbaren Logik-Blöcken (engl.: *Configurable Logic Blocks - CLB*) werden die eigentlichen Logik-Funktionen realisiert. Den vereinfachten Aufbau eines XC4000-*CLB* zeigt Abbildung 2.13, welcher im folgenden Text kurz erläutert wird.

- Zwei unabhängige Funktionsgeneratoren (F und G) können jede beliebige logische Funktion mit vier Eingangsvariablen abbilden. Ihre Ausgänge werden mit F' und G' bezeichnet. Die vier Eingangssignale bilden praktisch die Adressen eines *SRAM* der Konfiguration 16 x 1. Jede dieser 16 *SRAM*-Zellen wird mit einer Eins bzw. Null geladen. Dadurch wird die Funktion codiert, wobei man einen solchen Funktionsgenerator auch als eine sogenannte *Look-Up Table (LUT)* bezeichnet.

- Der Funktionsgenerator H kann jede beliebige logische Funktion mit drei Eingängen darstellen. Zwei dieser Eingänge sind mit den internen Signalen F' und G' verbunden.
- An zwei Ausgängen werden die kombinatorischen Ergebnisse dem externen Verbindungsnetzwerk zur Verfügung gestellt. F' oder H' werden mit dem Ausgang X und G' bzw. H' mit dem Ausgang Y verbunden.
- Zwei flankengesteuerte *D-Flip-Flops* können zur Speicherung der Funktionsergebnisse (F' , G' , H') oder eines externen Signals (D_{IN}) genutzt werden. Über die Ausgänge XQ und YQ werden ihre gespeicherten Werte dem Verbindungsnetzwerk zur Verfügung gestellt.

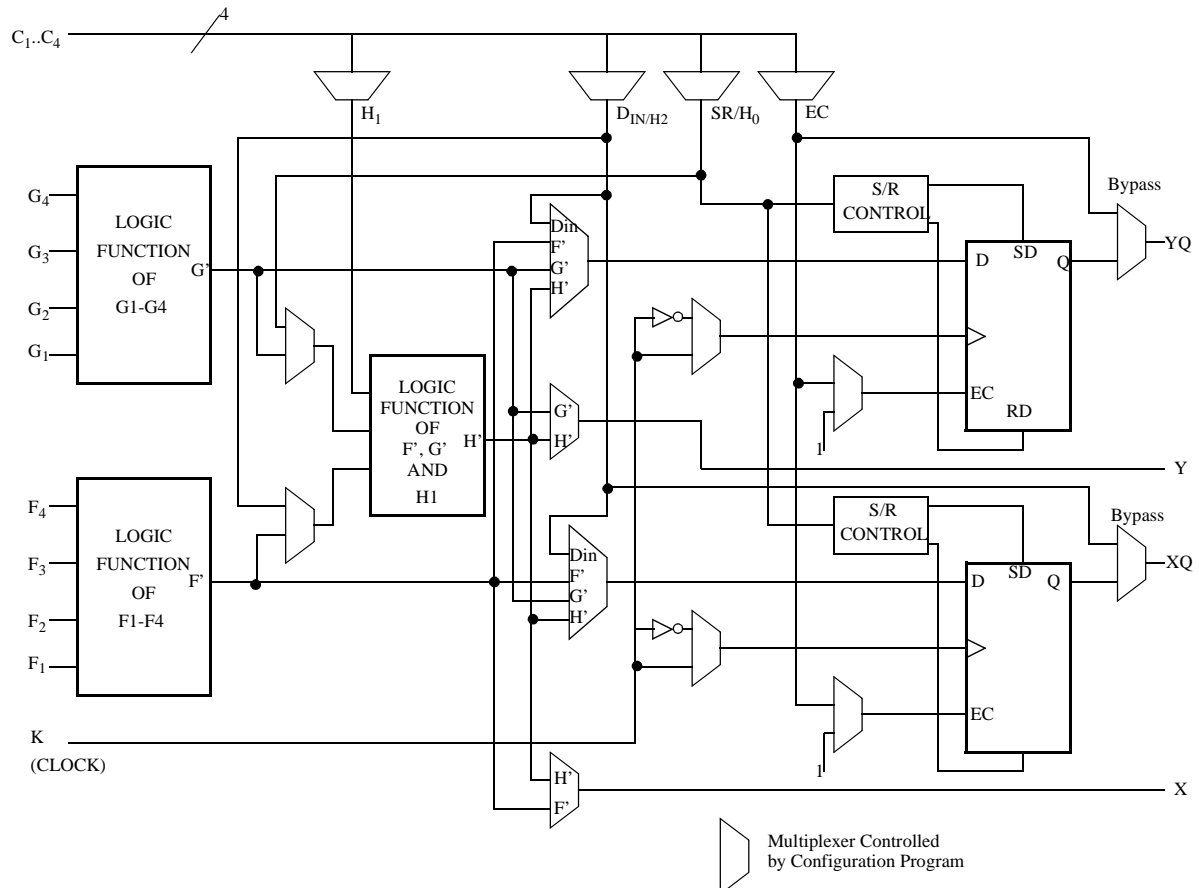


Abbildung 2.13 Xilinx XC4000 konfigurierbarer Logik-Block (Quelle: [Xil98/1])

Jeder *CLB* kann so konfiguriert werden, daß die Speicherzellen der *LUT* auch als *SRAM*-Blöcke verwendet werden können. Diese Fähigkeit man man feinkörniges *on-Chip-SRAM* (engl.: *fine-grained SRAM capability*). Sie zeichnet die XC4000-Architektur im Gegensatz zu allen anderen Architekturen aus.

Das kleinste *FPGA* innerhalb der XC4000-Familie verfügt über 100 *CLB* und 80 *IOB*, das größte *FPGA* hat zur Zeit 3136 *CLB* und 448 *IOB*. Die seriellen Konfigurations-Bits variieren dabei von 128 KBit bis 4 MBit.

Zusammenfassend besitzt die XC4000-Architektur folgende Eigenschaften:

- Eine digitale Schaltung kann während der Entwicklung leicht geändert werden, da die *FPGA*-Bausteine beliebig oft konfiguriert werden können.

- Bei einer neuen Konfiguration muß der gesamte *FPGA*-Chip geladen werden, eine teil- bzw. bereichsweise Konfiguration ist nicht möglich. Dieser Prozeß dauert je nach Bausteingröße zwischen 1 ms und 20 ms.
- Die XC4000-Architektur besitzt eine feinkörnige *on-Chip-SRAM*-Fähigkeit.

XC6200-Architektur (Xilinx)

Im Gegensatz zur weitverbreiteten XC4000-Architektur wurde von Xilinx die XC6000-Architektur vorgestellt, welche in Abbildung 2.14 dargestellt und in [Xil96] beschrieben ist. Die XC6000-Architektur ist zur Zeit in der wissenschaftlichen Forschung von großer Bedeutung. Die beiden entscheidenden Unterschiede zu den anderen *SRAM*-basierten Architekturen (z.B. XC4000, Altera 10K) sind die offene Konfigurationsstruktur und die 32-Bit breite Konfigurationsschnittstelle.

- Die Speicheradressen aller Verdrahtungsmultiplexer sowie die Multiplexer und *Flip-Flops* zur Implementierung von Logik-Funktionen innerhalb der *Function-Unit* sind offengelegt. Die Bedeutung der notwendigen Kodierungs-Bits, welche an den entsprechenden Adressen gespeichert werden müssen, ist ebenfalls offen. Sie wurden mit Sechs-Transistor-*SRAM*-Zellen implementiert. Diese Organisation bildet die Grundlage dafür, daß ein Mikrocontroller alle Register, die implementierte Logik und die Verdrahtung jederzeit durch Beschreiben und Lesen verändern kann. Diese Fähigkeit ermöglicht die Implementierung von virtueller Hardware, d.h. Schaltungsteile können während der Laufzeit ausgelesen und ihr aktueller Zustand gespeichert werden. Zusätzlich können neue Schaltungsteile eingeschrieben und aktiviert werden, ohne dabei andere Schaltungsteile zu stören.

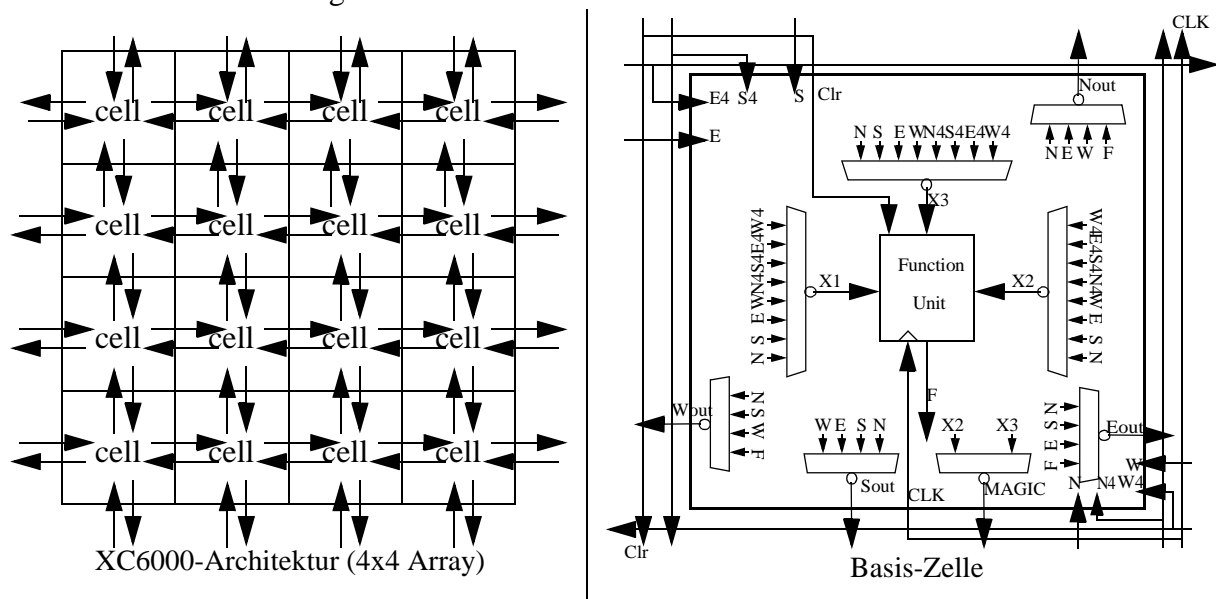


Abbildung 2.14 XC6000-Architektur und Basis-Zelle (Quelle: [Xil96])

- Um diese partielle Austauschmöglichkeit von Schaltungsteilen effizient durchführen zu können, wurden alle Konfigurations-Bits im Speicheradressraum einer parallelen Mikrocontroller-Busschnittstelle zusammengefaßt, welche mit einer Datenbreite von 8-Bit, 16-Bit oder 32-Bit gelesen und geschrieben werden kann. Die maximale Zugriffsbandbreite von ca. 150 MByte/sec erlaubt somit den Austausch von Schaltungsteilen im Bereich weniger Mikrosekunden.

Diese beiden Eigenschaften ermöglichen neue Entwurfsmethoden, welche in Abschnitt 2.1.3.4 diskutiert werden.

Virtex-Architektur (Xilinx)

Die Virtex-Architektur von Xilinx ist die neueste auf dem Markt verfügbare *SRAM*-basierte *FPGA*-Familie. Das entsprechende Blockdiagramm ist in Abbildung 2.15 dargestellt.

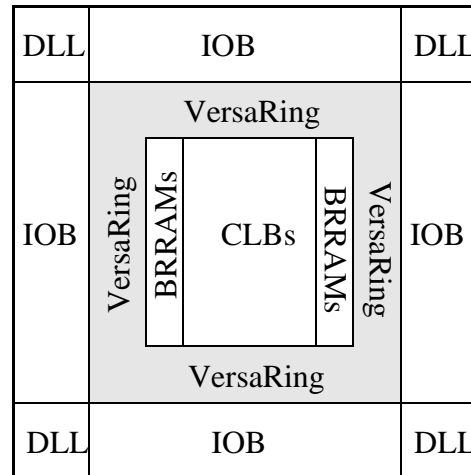


Abbildung 2.15 Virtex-Architektur (Quelle: [Xil99])

Diese Architektur basiert auf einer Weiterentwicklung der bereits vorgestellten XC4000-Architektur sowie einiger Eigenschaften der XC6000-Architektur. Die wesentlichen Erweiterungen sind in folgenden Punkten zusammengefasst:

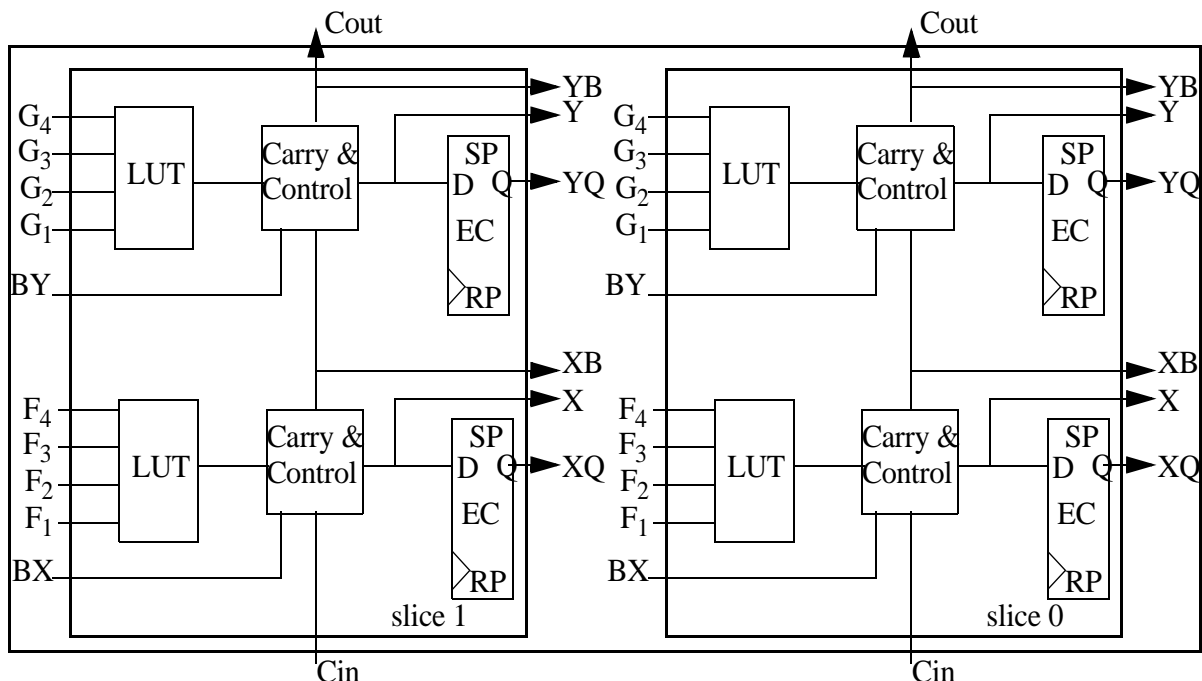


Abbildung 2.16 Xilinx 2-Slice Virtex CLB - vereinfachte Darstellung (Quelle: [Xil99])

- Die CLB bestehen aus zwei gleichen Teilen (engl.: *slice 1 and slice 0*), wie in Abbildung 2.16 dargestellt. Die LUT sind dabei sowohl zur Realisierung von kombinatori-

scher Logik als auch für feingranulare *RAM*-Blöcke (engl.: *select-RAM*) verwendbar. Der *Carry&Control-Block* faßt die Verbindungsstruktur zwischen *LUT* und *Flip-Flop* zusammen und stellt noch weitere Funktionen für schnelle Arithmetik zur Verfügung.

- Zusätzlich zu der feingranularen *SRAM*-Fähigkeit in den *LUT* stehen noch dedizierte eingebettete *SRAM*-Blöcke zur Verfügung, die mit *BRAM* (engl.: *Block RAM*) bezeichnet sind.
- Als Verbindungsmedium zwischen den *IOB* und den *CLB* ist ein sogenannter *Versa-Ring* vorhanden. Er verbindet alle *IOB* in einem Ring um die *CLB* und entkoppelt die interne Verdrahtung von Einschränkungen, welche durch eine feste Vorgabe der *IOB* entstehen.
- Alle *IOB* sind in insgesamt acht Gruppen zusammengefaßt. Jeder Gruppe kann eine andere Versorgungsspannung zugeordnet werden. Dadurch kann sich der *FPGA* an bis zu 16 verschiedene elektrische Schnittstellenverhalten (z.B. *PCI*, *AGP*, usw.) anpassen.
- Der *Virtex-FPGA* kann sowohl seriell als auch parallel geladen werden. Damit verfügt er über einen sehr schnellen Konfigurationsmodus. Zusätzlich können bestimmte *FPGA*-Bereiche während der Laufzeit neu geladen werden, während die anderen ungestört weiterarbeiten. Damit verfügt er über Eigenschaften, die bei der *XC6000*-Architektur eingeführt wurden.

Diese Bausteine werden für die Werkzeuge eingesetzt, welche im Rahmen dieser Arbeit implementiert wurden und in Kapitel 5 vorgestellt werden.

2.1.3.3 Antifuse-basierte Anwender-programmierbare Schaltungen

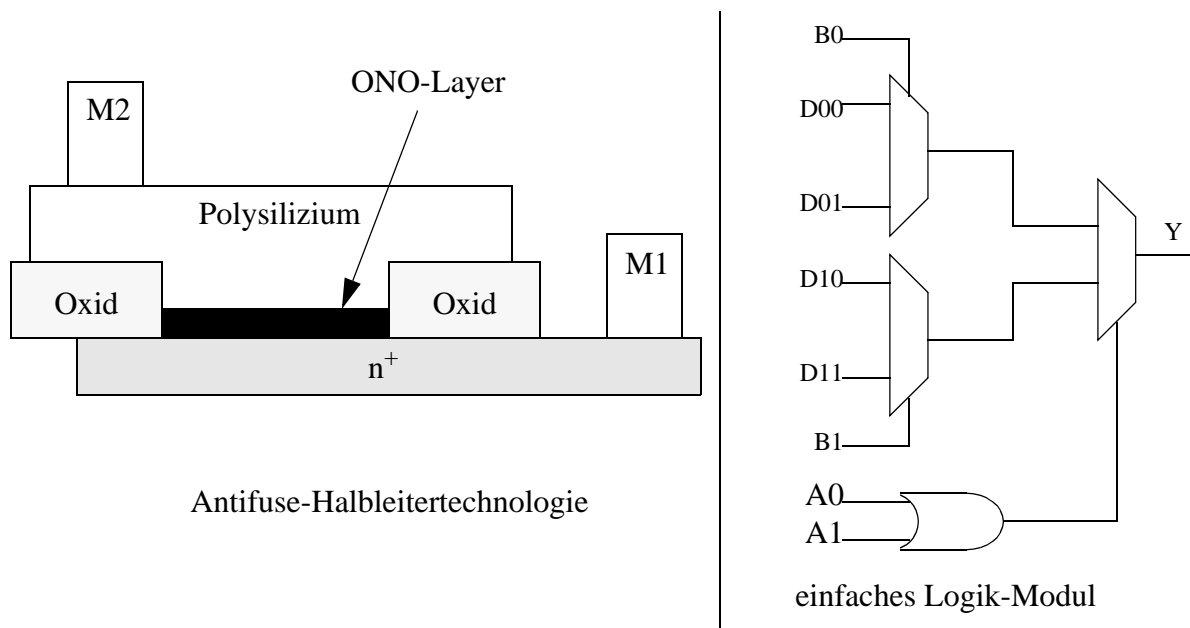


Abbildung 2.17 Antifuse-Halbleitertechnologie und ACT1-Logik-Modul (Quelle:[Act94])

Die *Antifuse-FPGA* von Actel werden vor der Bestückung auf der Leiterplatte in einem speziellen Programmiergerät gebrannt. Bei der *Antifuse*-Technologie, welche auf der linken Seite in Abbildung 2.17 dargestellt ist, existiert eine dünne Isolationsschicht zwischen zwei Leiterbahnen, die bei der Programmierung zerstört wird. Die so entstehende Verbindung ist auf Dauer „nicht-flüchtig“ im *FPGA* gespeichert. Es bedarf daher keiner Konfigurierung nach dem Einschalten der Betriebsspannung. Dieser Einbrennprozeß ist aber nicht umkehrbar, d.h. ein einmal programmierter Chip kann nie wieder geändert werden.

Die in Abbildung 2.17 dargestellte Isolation besteht aus einer Oxid-Nitrat-Oxid-Schichtung (ONO-Layer). Diese Schichten sind jeweils mit einer der Metallagen (M1, M2) verbunden, die den Leitungssegmenten entsprechen. Diese metallischen Leitungssegmente sind an Segmentgrenzen und an Kreuzungen durch sogenannte *Antifuses* getrennt, die nach der Produktion des Chips zunächst offen sind.

Auf der rechten Seite von Abbildung 2.17 ist ein einfaches Logik-Modul der ACT1-Familie dargestellt. Dieses Logik-Modul dient zur Implementierung der logischen Funktion und ist wesentlich einfacher aufgebaut als die *CLB* in den *SRAM*-basierten *FPGA*. Das Modul besitzt acht Eingänge und einen Ausgang. Es realisiert folgende Funktion:

$$Y = \bar{S}_1 \wedge \bar{B}_0 \wedge D_{00} \vee \bar{S}_1 \wedge B_0 \wedge D_{01} \vee S_1 \wedge \bar{B}_1 \wedge D_{10} \vee S_1 \wedge B_1 \wedge D_{11} \quad \text{mit } S_1 = A_0 \vee A_1$$

Die Daten- und Steuereingänge der Multiplexer können mit Signalen der Verdrahtungskanäle verbunden werden. Dazu zählen auch die Signale logisch Eins und Null, die immer im Verdrahtungskanal zur Verfügung stehen. Ein pegelgesteuertes Speicherelement läßt sich mit Hilfe eines Logikmoduls realisieren. Für ein flankengesteuertes Speicherelement werden zwei Logik-Module benötigt.

2.1.3.4 Entwurfsmethoden für *FPGA*

Nach der Einführung der wichtigsten *FPGA*-Architekturen werden in diesem Abschnitt die damit verbundenen Entwurfsmethoden diskutiert.

Compile Time Reconfiguration (CTR)-Methode

Die wichtigste Eigenschaft der *CTR*-Methode ist, daß die Konfigurationsdaten in das *FPGA* geladen werden, bevor die eigentliche Anwendung ausgeführt wird. Diese Konfiguration bleibt unverändert, solange die Anwendung ausgeführt wird. Diese Entwurfsmethode entspricht dem *ASIC*-Entwurf, für den leistungsstarke Entwurfswerkzeuge vorhanden sind. Diese Methode wird zur Zeit am häufigsten angewendet und ist nicht an eine bestimmte *FPGA*-Architektur gebunden. Das bedeutet, diese Methode kann für die weitverbreitete XC4000-, Virtex- und *Antifuse*-Architektur sowie für die partial rekonfigurierbare XC6000-Architektur eingesetzt werden.

Run Time Reconfiguration (RTR)-Methode

Die *RTR*-Methode allokiert und deallokiert *FPGA*-Ressourcen zur Laufzeit der Anwendung. Daraus folgt: Ein *FPGA*-Chip wird zur Implementierung von verschiedenen Hardware-Modulen benutzt. Diese Module sind allerdings ein Teil der Anwendung und werden während ihrer Laufzeit auf dem *FPGA*-Chip ausgetauscht. Hutchings und Wirtlin *et. al.* [HuWi95] unterteilen die *RTR*-Methode weiter in die globale *RTR* und lokale *RTR*-Methode.

- Die globale *RTR*-Methode kann für Anwendungen genutzt werden, bei denen Hardware-Module in nicht gleichzeitig ablaufende Teile zerlegt werden können. Diese Module werden in sequentieller Reihenfolge ausgeführt. In einem solchen Anwendungsfall startet die Anwendung mit einem Hardware-Modul, welches entweder alle *FPGA*-Ressourcen benötigt oder nur einen Teil davon. Während der Laufzeit werden andere Module geladen und ausgeführt. Immer nur ein Modul ist zu einem bestimmten Zeitpunkt aktiv. Wie in [ElHu94] beschrieben, könnte beispielsweise durch die Anwendung der globalen *RTR*-Methode die funktionale Dichte eines neuronalen Netzes um 500% gesteigert werden, im Gegensatz zur Anwendung der *CTR*-Methode. Die globale *RTR*-Methode ist prinzipiell nicht an eine spezielle *FPGA*-Architektur gebunden. Einzige Voraussetzung ist jedoch, daß die Architektur unbegrenzt neu programmierbar ist, wie z.B. die *XC4000*-Architektur.
- Die lokale *RTR*-Methode verändert dieses globale *RTR*-Konzept und ermöglicht zu beliebigen Zeitpunkten den Austausch von verschiedenen Hardware-Modulen, welche bestimmte Teile der *FPGA*-Chip-Fläche belegen. Dieser Ansatz führt zu einer feingranularen Rekonfigurationsarchitektur und zu einer effizienteren Nutzung der *FPGA*-Ressourcen. Diese Methode erfordert zwingend den Einsatz von partiell rekonfigurierbaren *FPGA*-Bausteinen wie die *XC6000-FPGA*. Die *Virtex-FPGA* verfügen ebenfalls über diese Eigenschaft, welche allerdings zur Zeit vom Hersteller nur angekündigt ist. Die genaue Funktionsweise ist noch nicht vollständig veröffentlicht.

2.1.4 Anwendungsspezifische integrierte Schaltungen

Im Rahmen dieser Arbeit werden unter anwendungsspezifischen integrierten Schaltungen (engl.: *Application Specific Integrated Circuits - ASIC*) Hardware-Komponenten verstanden, die auf dem Markt verfügbar sind und eine sehr spezielle, komplexe und anwendungsbezogene Teilfunktion innerhalb eines bestimmten Einsatzgebietes ausführen. Dabei handelt es sich in der Regel um mikroelektronische Komponenten mit einer sehr hohen Integrationsdichte (engl.: *Very Large Scale Integration - VLSI*)

Es wurde bereits festgestellt, daß eingebettete Systeme in dem hier betrachteten Anwendungsgebiet der industriellen Automation und Kommunikation aus wenigen, dafür aber hochintegrierten Bausteinen bestehen. Neben einem fast immer vorhandenen Mikrocontroller besteht die Architektur insbesondere auch aus *ASIC*-Bausteinen. Wenn ein *ASIC* für eine bestimmte Teilfunktion zur Verfügung steht, ist er einer eigenen Hardware-Entwicklung aus Zeit- und Kostengründen unbedingt vorzuziehen, da diese Bausteine auf einem enormen Entwicklungsaufwand basieren, der im Rahmen der Systementwicklung weiter genutzt werden muß. Die systematische Auswahl und Bewertung einer *ASIC*-Komponente ist ein zentraler Punkt bei dem in dieser Arbeit beschriebenen systematischen Architekturentwurf und wird ausführlich in Kapitel 4 dargestellt. In Kapitel 6 wird an einem Beispiel die Überprüfung der Eigenschaften eines *ASIC*-Bausteins durch Emulation gezeigt. Einen vollständigen Überblick über die Entwurfsprobleme, welche von *ASIC*-Komponenten beim Entwurf eingebetteter Systeme verursacht werden, wird in Abschnitt 3.1.4 dargestellt.

2.1.5 Technologie eingebetteter Systeme

Breitere Daten- und Adressbusse sowie zahlreiche *on-Chip*-Peripherie-Komponenten benötigen Gehäuse mit drastisch steigender Pinanzahl. Beim heutigen Stand der Technik werden

für hochintegrierte Chips sogenannte *Ball-Grid-Array*-Gehäuse benutzt. Als Beispiel ist in Abbildung 2.18 das Gehäuse eines 32-Bit-Mikrocontrollers aus der MPC8xx-Familie dargestellt.

Sie bieten externe Anschlußmöglichkeiten im Bereich zwischen 250 und 600 Gehäusepins, welche in einem quadratischen Feld angeordnet sind. Der Baustein liegt dabei mit kleinen Halbkugeln (engl.: *balls*) auf den Anschlußflächen (engl.: *pads*) der Platine auf. Diese *pads* sind in einem sehr engen Raster von 1,27 mm angeordnet. Im gezeigten Beispiel werden 256 Pins auf einer Fläche von nur 363 mm² (19,05 mm x 19,05 mm) mit der Platine kontaktiert. In Verbindung mit weiteren Randbedingungen, wie der externen Busfrequenz von zur Zeit 66 MHz und Chip-Temperaturen bis 95C^o, werden extreme Anforderungen an die Fertigungstechnologie gestellt. Einfache Laboraufbauten sind nicht mehr möglich und erfordern schon bei Prototypen hochentwickelte Multilayer-Platinen in Feinstleitetertechnik. Die damit verbundenen Probleme werden ebenfalls in Abschnitt 3.1.4 diskutiert. Dieser Faktor stellt auch ein eigenständiges Entscheidungsfeld bei der systematischen Bewertung von Hardware-Komponenten im Rahmen des Architekturentwurfs für eingebettete Systeme in Kapitel 4 dar.

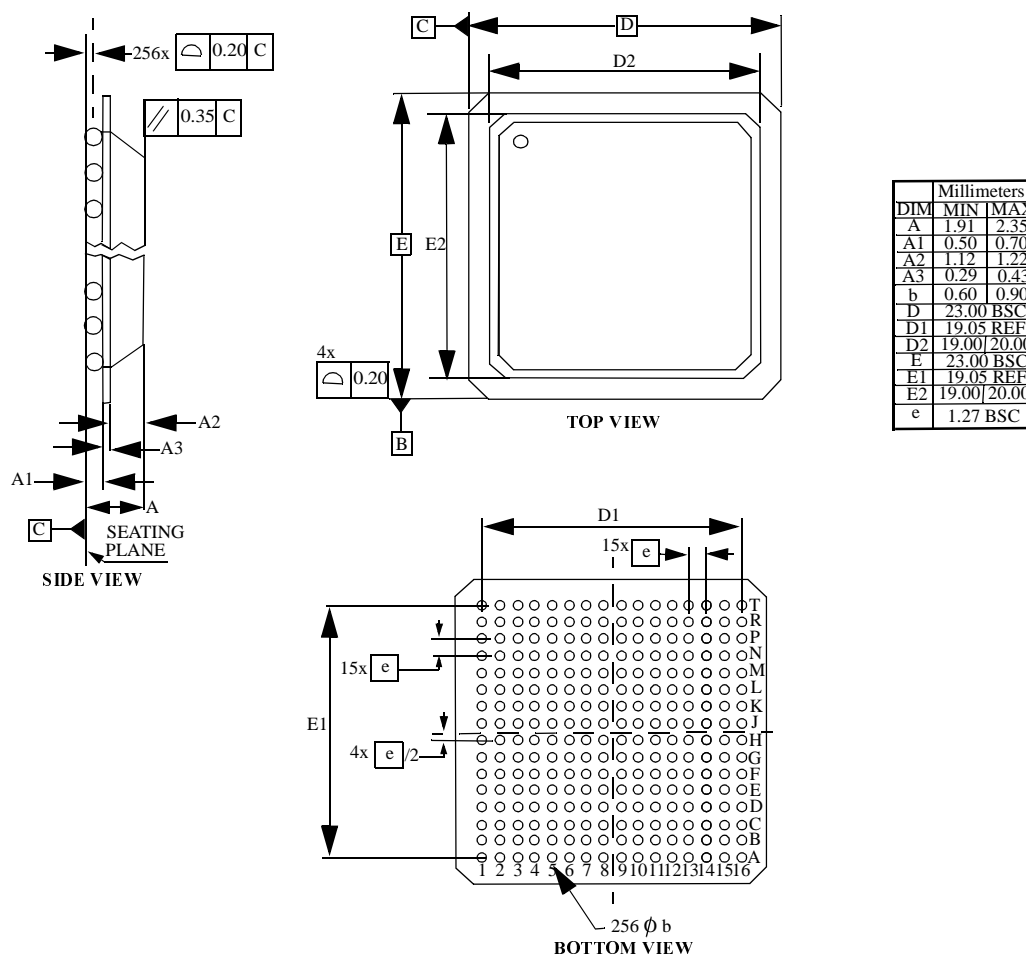


Abbildung 2.18 256 Ball-Grid-Array-Gehäuse beim MPC8xx (Quelle: [Mot98/1])

2.2 Software-Komponenten

Eingebettete Systeme mit 4-Bit- bzw. 8-Bit-Mikrocontrollern werden bis heute überwiegend ohne ein Betriebssystem eingesetzt. Diese Tatsache bedeutet, daß der Programmierer alle Funktionen, die für die Anwendung benötigt werden, selbst programmieren muß. Bei 8-Bit-Mikrocontrollern haben sich für nicht-zeitkritische Programmteile durchaus höhere Programmier-

sprachen wie z.B. C etabliert. Laufzeitkritische Teile werden dabei noch häufig im Assembler-Code optimiert.

Beim Übergang auf 16-Bit- bzw. 32-Bit-Mikrocontroller kommen immer häufiger Betriebssysteme zum Einsatz, welche im Bereich der allgemeinen Rechnertechnik seit vielen Jahren zum Standard gehören. Die damit verbundenen Vorteile liegen in der Verwaltung von angeschlossenen Peripheriegeräten sowie der erreichbaren Pseudo-Parallelität der Software. Die Rechenleistung des Mikrocontrollers wird dabei auf mehrere Prozesse verteilt. Die Implementierung und Optimierung komplexer Anwendungs-Software wird dadurch wesentlich vereinfacht. In diesem Zusammenhang spielen Begriffe wie ein Prozeß, ein *Scheduler* oder *Semaphore* eine Rolle, auf deren prinzipielle Bedeutung hier nicht eingegangen werden kann. Eine entsprechende Einführung in die Grundlagen wird in [Tan94] gegeben. In diesem Abschnitt wird zur Verdeutlichung ein kommerzielles Echtzeitbetriebssystem vorgestellt, welches den Stand der Technik gut widerspiegelt. Insbesondere sollen die mit der Einführung eines Betriebssystems verbundenen Entwurfsprobleme analysiert werden, welche bei der Entwicklung einer entsprechenden Entwurfsmethodik berücksichtigt werden müssen.

2.2.1 Echtzeitbetriebssystem *VxWorks*

Die Abbildung 2.19 zeigt die Entwicklungskonfiguration bei der Verwendung des Echtzeitbetriebssystems (engl.: *Real Time Operating System - RTOS*) *VxWorks*. Ein Entwicklungsrechner, in diesem Fall ein PC mit dem Betriebssystem *Windows-NT*, kommuniziert über ein Ethernet-basiertes Netzwerk mit dem eigentlichen Zielsystem. In dieser Arbeit wird das Entwicklungswerkzeug *SPYDER-CORE-P1* als Zielsystem verwendet. Der Programm-Code wird auf dem PC geschrieben (vorzugsweise in C oder C⁺⁺) und mit Hilfe eines Compilers in ein sogenanntes Objekt-Format übersetzt. Dieses Format beinhaltet einen ausführbaren Code für den jeweiligen Mikrocontroller, welchen das Echtzeitbetriebssystem *VxWorks* frei in seinem Arbeitsspeicher verschieben kann.

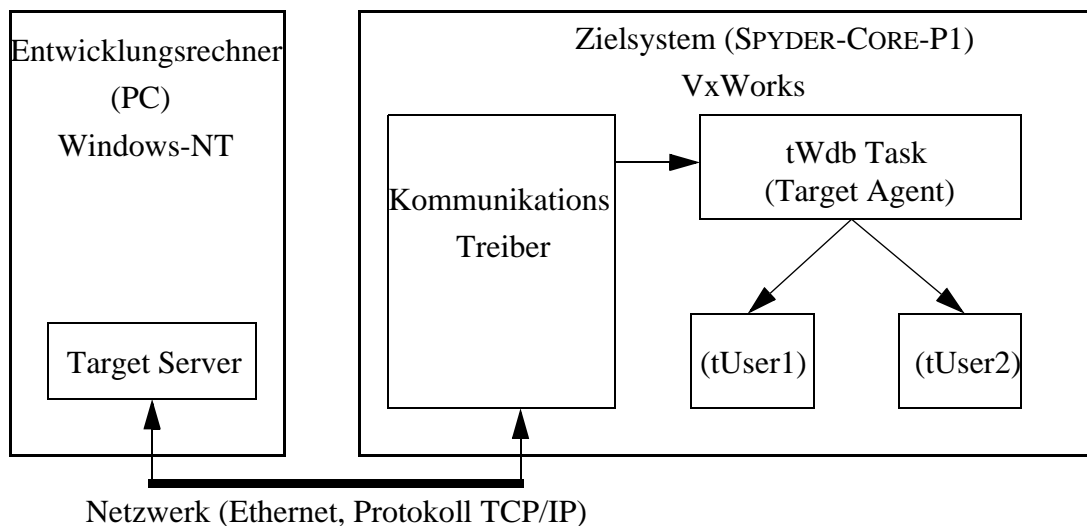


Abbildung 2.19 Entwicklungsumgebung für *Tornado* für das Echtzeitbetriebssystem *VxWorks*

Das Laden des ausführbaren Programmes auf das Zielsystem übernimmt der sogenannte *Target-Server*. Vom Hersteller wird als physikalisches Kommunikationsmedium eine Ethernet-Schnittstelle empfohlen. Dieser *Target Server* kommuniziert über einen speziellen Treiber auf dem Zielsystem mit dem sogenannten *Target Agent*. Als Protokoll wird das sogenannte *TCP/IP*

benutzt. Der *Target Agent* wird als eine eigene *Task* auf dem eigentlichen Echtzeitbetriebssystem *VxWorks* gestartet. Über diese Verbindung können nun beliebig viele eigenständige *Tasks* (*tUser1*, *tUser2*, ...) für die eigentliche Anwendung geladen werden. Zum Test können auch *debug*-Informationen von einem *Debugger* auf dem Entwicklungsrechner vom Zielsystem angefordert werden. Alle Software-Programme auf dem Entwicklungsrechner sind in einer gemeinsamen Benutzeroberfläche zusammengefaßt. Das gesamte Entwicklungssystem, bestehend aus den Werkzeugen auf dem Entwicklungsrechner und den Werkzeugen auf dem Zielsystem, welche das eigentlichen Echtzeitbetriebssystem *VxWorks* beinhalten, trägt den Namen *Tornado*.

2.2.1.1 *Tasks* und *Multitasking*

Bei steigender Programmkomplexität wird die Organisation der Gesamtfunktionalität der Software immer wichtiger. Eine leistungsstarke Methode zur Beherrschung dieser Komplexität ist die Partitionierung in unabhängige Programmteile. Diese Teile müssen zusätzlich in der Lage sein, miteinander gemeinsame Daten austauschen zu können. Jedes dieser alleine für sich ausführbare Teilprogramm wird als sogenannte *Task* bezeichnet. Der englische Begriff *Task* wird in diesem Zusammenhang im gleichen Sinne benutzt wie der Begriff „Prozeß“ in der deutschen Grundlagen-Literatur [Tan94]. Das Echtzeitbetriebssystem ermöglicht jeder *Task* den Zugriff auf die System-Ressourcen.

Multitasking ist der fundamentale Mechanismus für eine Anwendung, um auf externe diskrete Ereignisse zu reagieren. *Multitasking* sorgt dafür, daß mehrere *Tasks* „quasi gleichzeitig“ laufen. Tatsächlich bewirkt das *Multitasking*, daß die einzelnen *Tasks* zeitlich versetzt auf der *CPU* des Mikrocontrollers ablaufen. Die Art der Auswahl der einzelnen *Tasks* in dieser seriellen Reihenfolge kontrolliert ein sogenannter *Scheduling*-Algorithmus. Dabei hat jede *Task* ihren sogenannten Kontext. Darunter werden alle *CPU* und System-Ressourcen zusammengefaßt, die eine *Task* während ihrer aktiven Ausführungszeit benötigt. Bei *VxWorks* beinhaltet dieser Kontext folgende Variablen:

- Der aktuelle Programmzähler der *Task*,
- die belegten *CPU*-Register,
- ein externer *Stack*-Bereich für dynamische Variable und Funktionsaufrufe,
- Ein/Ausgabe-Zuweisungen für Standard-Ein/Ausgabe-Kommunikation,
- Zeitgeber,
- verschiedene Kernel-Kontrollstrukturen,
- *Debugging*-Information
- und der *Signal-Handler*.

Bei jedem *Task*-Wechsel muß der dazugehörige Kontext im Hauptspeicher abgelegt und der Kontext der neuen *Task* geladen werden. Aus diesem Prozeß resultieren die in einem eingebetteten System mit harten Echtzeitbedingungen wichtigen Kenngrößen der *Task*-Umschalt- bzw. *Interrupt*-Reaktionszeiten, welche im Abschnitt 6.2 am Beispiel des *ASI-Masters* untersucht werden.

2.2.1.2 Task-Zustandsübergänge und Scheduling-Algorithmen

Wie Abbildung 2.20 zeigt, kann eine *Task* vier verschiedene Zustände einnehmen, welche vom Betriebssystem kontrolliert werden:

- **ready-Zustand:** Der Zustand einer *Task*, wenn sie auf keine weiteren System-Ressourcen wartet außer auf die *CPU* des Mikrocontrollers.
- **pending-Zustand:** Die *Task* ist blockiert, weil bestimmte System-Ressourcen nicht verfügbar sind.
- **delay-Zustand:** Die *Task* soll für einen bestimmten Zeitraum nicht mehr aktiviert werden („schlafen gelegt“).
- **suspended-Zustand:** Diese *Task* kann nicht mehr ausgeführt werden (nur für *debug*-Zwecke).

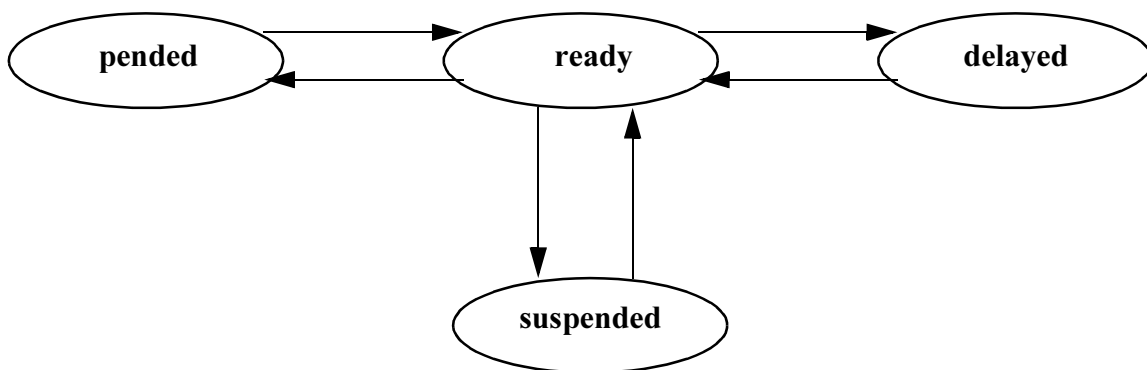


Abbildung 2.20 Task-Zustands-Übergangsdigramm bei *VxWorks*

Wenn aufgrund verschiedener externer Ereignisse, wegen Ablaufes einer bestimmten Zeitschranke oder wegen fehlender System-Ressourcen, eine *Task* nicht mehr weiter bearbeitet werden kann, führt das Betriebssystem einen *Task*-Wechsel durch. Bei der Existenz von vielen weiteren *Tasks* können prinzipiell nur solche *Tasks* gestartet werden, welche sich im aktuellen Zustand **ready** befinden. Es kann aber nur eine einzige *Task* weiter bearbeitet werden. Deshalb muß eine gewisse Auswahl unter den *Tasks* getroffen werden, welche sich im aktuellen Zustand **ready** befinden. Einen solchen Auswahl-Algorithmus nennt man *Task-Scheduling*. Das Echtzeitbetriebssystem *VxWorks* bietet zwei unterschiedliche Algorithmen an.

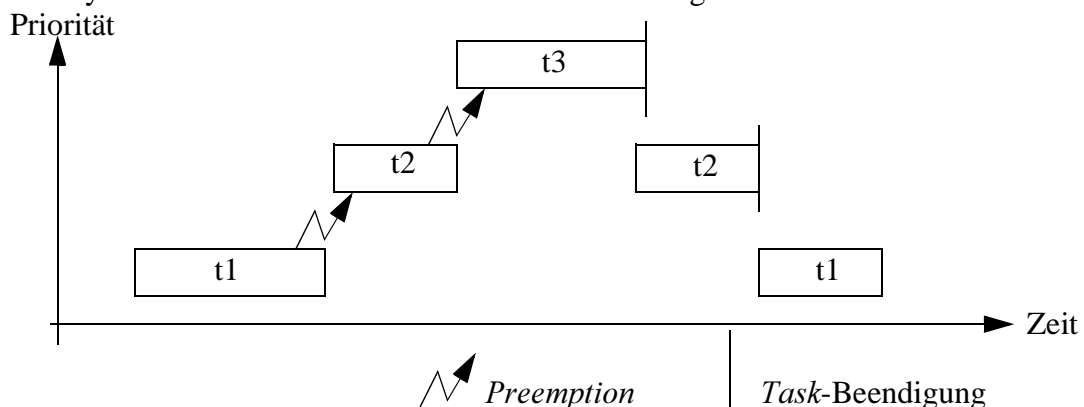


Abbildung 2.21 Zeitdiagramm für *Priority Preemption Scheduling* beim RTOS *VxWorks*

- **Priority Preemptive Scheduling:** Bei dieser Methode startet der Kernel die *Task*, welche die höchste Priorität hat. Dabei unterbricht der Kernel eine laufende *Task*, sichert ihren Kontext und lädt den Kontext der höher priorisierten *Task*. Das Verhalten ist in Abbildung 2.21 dargestellt. Die *Task* t1 wird unterbrochen von der *Task* t2. Diese wird zusätzlich unterbrochen von t3, welche die höchste Priorität hat. Nach der Beendigung von t3 wird t2 fortgesetzt, bis diese von sich aus ihre aktuelle Arbeit beendet hat und die *CPU* freigibt. Danach wird wieder die *Task* t1 ausgeführt.
- **Round Robin Scheduling:** Diese Methode versucht die Rechenzeit der *CPU* möglichst gleichmäßig auf die *ready-Tasks* einer gemeinsamen Prioritätsstufe zu verteilen. Abhängig von einem vorgegebenen Zeitfenster werden die *Tasks* bearbeitet und nach dem Ablauf des Zeitfensters zyklisch auf die nächste *Task* der gleichen Prioritätsebene weitergeschaltet. Erreicht eine *Task* mit höherer Priorität den *ready*-Zustand, wird diese unverzüglich gestartet und die aktuell laufende *Task* der niedrigeren Ebene unterbrochen (siehe t2). Nach der Beendigung von t4 wird die *Task* t2 an der unterbrochenen Stelle fortgesetzt.

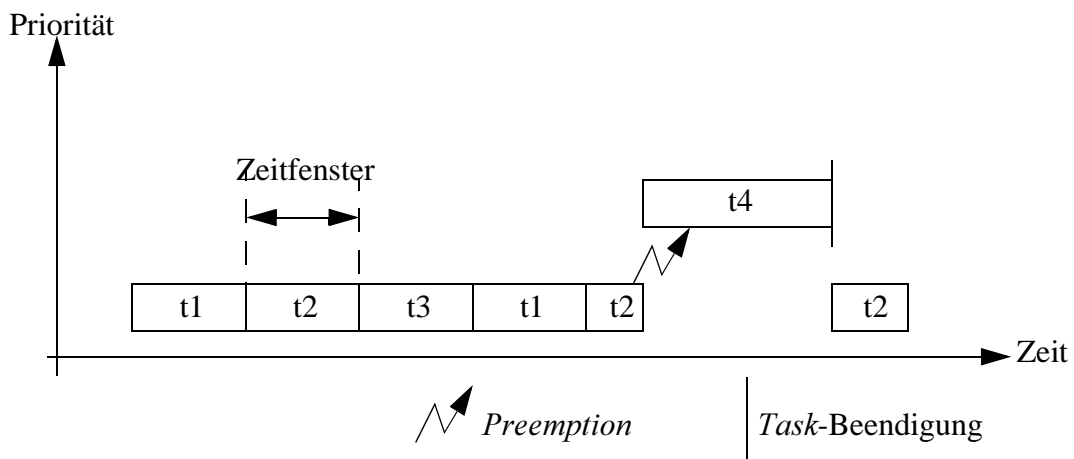


Abbildung 2.22 Zeitdiagramm für Round Robin Scheduling beim RTOS VxWorks

In dieser Arbeit wird ein Echtzeitbetriebssystem als komplexe Software-Komponente aufgefaßt. Beim Software-Architekturentwurf stellt sich für den Entwickler die Frage, ob die Komplexität der Anwendungssoftware den Einsatz eines Echtzeitbetriebssystems erfordert oder nicht. Um diese Frage zu beantworten, müssen die Kenngrößen, wie z.B. die *Task*-Umschalt- bzw. *Interrupt*-Reaktionszeiten und der zusätzliche Speicherplatzbedarf, bekannt sein. Diese Größen hängen aber wesentlich von der Hardware-Plattform, also dem Mikrocontroller-Kern, ab. Im Rahmen dieser Arbeit werden in Kapitel 6 am Beispiel des *ASI-Masters* diese Werte durch eine echtzeitfähige Emulation ermittelt. Sie bildet die Grundlage für die Auswahlentscheidung für oder gegen den Einsatz eines Echtzeitbetriebssystems.

Kapitel 3

Motivation

Dieses Kapitel greift die in Kapitel 1 dargestellten Systeme auf und führt den Begriff des sogenannten „eingebetteten Systems“ ein. Die Architektur einer bestimmten Klasse von eingebetteten Systemen, welche insbesondere im Bereich der industriellen Automation und Kommunikation zu finden ist, wird von anderen Architekturen abgegrenzt. Die in dieser Arbeit vorgestellte Entwurfsmethode bezieht sich insbesondere auf diese Klasse von eingebetteten Zielsystemen. Eine kurze Übersicht der entstehenden Entwurfsproblematik beim Übergang von 4-Bit, 8-Bit, 16-Bit bis zu modernen eingebetteten Systemen mit einem 32-Bit-Mikrocontroller wird gegeben. Dieses Kapitel steht in engem Zusammenhang mit Kapitel 2, in dem diese Komponenten im Detail erläutert wurden. Dieses Kapitel fokussiert auf die Darstellung der mit diesen Komponenten verbundenen Entwurfsproblematik und dient als Grundlage für die Diskussion der vorhandenen Entwurfsmethoden. Die bisher in der Praxis eingesetzte Methode wird mit aktuellen in der Forschung entwickelten Methoden verglichen. Anschließend werden aus dieser Gegenüberstellung die Vor- und Nachteile extrahiert. Diese Analyse wird die Notwendigkeit der in dieser Arbeit vorgestellten Methodik **„Architekturwurf und Emulation von eingebetteten Systemen“** verdeutlichen. Das Kapitel schließt mit einer Übersicht der weiteren Gliederung der Arbeit ab.

In der Fachliteratur hat sich für die in Kapitel 1 aufgeführten Systeme der Begriff des „eingebetteten Systems“ (engl.: *embedded system*) etabliert. Die meisten Leser würden sehr schnell darin übereinstimmen, daß es sich bei den in Kapitel 1 genannten Systemen um eingebettete Systeme handelt. Fragt man aber nach einer übergreifenden Definition eines eingebetteten Systems, so hat sich eine allgemein akzeptierte Definition, wie in [Wo94] dargestellt, noch nicht durchgesetzt. Um diesen Gegensatz zu verdeutlichen, werden zwei unterschiedliche Anwendungen gegenüber gestellt. Betrachtet man z.B. einen handelsüblichen PC wie in [Wo94], welcher mit spezieller Hardware für eine bestimmte Anwendung modifiziert wurde, so werden die Definitionsprobleme deutlich. Wahrscheinlich würden einige Leser dieses System nicht zu den eingebetteten Systemen zählen, sondern zu den typischen Anwendungen der allgemeinen Rechner-technik. Andere Leser würden, da es sich auch um dedizierte Hardware handelt, von einem eingebetteten System sprechen.

Ein anderes extremes Beispiel ist in [DeMi95] zu finden. Dort wird ein sogenannter „*Video Conference Processor (VCP)*“ vorgestellt. Die Besonderheit dabei ist, daß es sich um einen einzigen Chip handelt, auf dem mehrere Prozessoren (ein *MIPS-X* und ein digitaler Signal-Prozessor) zusammen mit ihren Speichern und Ein/Ausgabe-Einheiten implementiert wurden. Auch hier stellt sich die Frage, ob es sich um ein eingebettetes System handelt. Einige Leser würden diese Frage mit ja beantworten, vielleicht sogar noch einen Schritt weiter gehen und diese Anwendung mit einem weiteren Fachbegriff, nämlich als ein sogenanntes „*System on a Chip (SoC)*“, bezeichnen. Andere würden die Frage mit nein beantworten und diese Anwendung als einen hochintegrierten „*Application Specific Integrated Processor (ASIP)*“ bezeichnen. Diesen sogenannten *ASIP* würden sie dann eventuell als Komponente in einem übergeordneten, eingebetteten System auffassen.

Die funktionalen Eigenschaften, die ein eingebettetes System abgrenzen, sind sicher fließend und hängen wesentlich vom subjektiven Standpunkt des Betrachters ab. Ein wesentliches Kriterium von eingebetteten Systemen ist die Tatsache, daß sie für dedizierte Anwendungen entworfen werden und ganz bestimmten Randbedingungen unterliegen wie z.B. Zuverlässigkeit, Entwicklungszeit, Leistungsaufnahme, Bauform und vor allem Produktionskosten. Wie in [Wo94] dargestellt, ist sicherlich eine weitere Gemeinsamkeit von eingebetteten Systemen, daß viele Entwickler glauben, daß die Implementierung von einigen Systemfunktionen auf Mikroprozessoren bzw. Mikrocontrollern die Einhaltung der genannten Randbedingungen wesentlich erleichtert. Daraus resultiert die allgemeine Vorstellung, daß bei den meisten eingebetteten Systemen ein Mikrocontroller existiert, auf dem eine dedizierte Software ausgeführt wird, welche einen bestimmten Anteil der gesamten Systemfunktionen erfüllt. Die restlichen, von diesem Software-Teil nicht abgedeckten Systemfunktionen, müssen von einer speziellen Hardware erfüllt werden. Somit gilt für eingebettete Systeme, daß möglichst parallel ein zeitgleicher Entwurf für Hardware und Software durchgeführt werden muß, was in der Fachliteratur im allgemeinen mit dem Begriff des „parallelen Hardware/Software-Entwurfs“ (engl.: *Hardware/Software Co-Design*) bezeichnet wird.

Für die hier vorliegende Arbeit soll unter einem eingebetteten System folgendes verstanden werden:

Definition 3.1: Ein eingebettetes System besteht aus einem anwendungsspezifischen Software-Teil und aus einem anwendungsspezifischen Hardware-Teil. Beide Teile kommunizieren untereinander und mit der Umgebung. Insbesondere die Umgebung gibt bestimmte Randbedingungen vor, die von dem eingebetteten System eingehalten werden müssen.

Diese Definition beschreibt relativ allgemein ein eingebettetes System aus der Sicht seines Verhaltens und deckt ein weites Anwendungsspektrum ab. Verfeinert man die Gesamtfunktionalität in kleinere, untereinander kommunizierende Teile und bildet diese auf konkrete Funktionseinheiten ab, entsteht die sogenannte Zielarchitektur. Diese beschreibt aus struktureller Sicht ein eingebettetes System. Dabei zeigt sich, daß die Definition einer einheitlichen Architektur, welche möglichst alle eingebetteten Systeme beschreibt, nicht mehr möglich ist. Die Zielarchitektur hängt wesentlich von den Randbedingungen der Anwendung ab. Diese Einschränkungen können anhand zweier unterschiedlicher Anwendungen verdeutlicht werden:

- Der Entwurf eines eingebetteten Systems zur Erkennung eines handschriftlich geschriebenen Textes benötigt einen Mikrocontroller-Kern, auf dem ein Erkennungsalgorithmus abgearbeitet wird. Dieser Algorithmus ermittelt aus eingelesenen Bildpunkten (engl.: *bit map*) das dazugehörige Schriftzeichen. Dafür eignen sich bei-

spielsweise Architekturen mit mehreren parallel arbeitenden Mikrocontrollern. Dabei steigt die Systemleistung fast linear mit der Anzahl und Rechenleistung der einzelnen Mikrocontroller. Die Kommunikation der einzelnen Mikrocontroller untereinander oder mit einem übergeordneten Mikrocontroller ist im Verhältnis zur Laufzeit für den Erkennungsalgorithmus zu vernachlässigen. Diese Klasse von Anwendungen muß auf keine asynchronen Ereignisse seitens der Umgebung reagieren und zeichnet sich durch einen hohen Datendurchsatz mit hoher Prozessorauslastung aus.

- Betrachtet man im Gegensatz dazu ein eingebettetes System zur Steuerung eines Aufzuges in einem Hochhaus, so muß das System auf den asynchron eintreffenden Tastendruck der Fahrgäste in den einzelnen Etagen reagieren und daraus innerhalb bestimmter Zeitgrenzen die entsprechenden Stellbefehle für die Antriebsmotoren berechnen. Das eingebettete System ist somit ein reaktives System mit harten Echtzeitanforderungen. Die dazu benötigte Zielarchitektur basiert im wesentlichen auf einem einzelnen Mikrocontroller. Dieser muß genügend Rechenleistung bereitstellen, um in ungünstigen Situationen alle Anforderungen zu erfüllen. Seine mittlere Auslastung ist im Gegensatz zu dem oberen Beispiel relativ gering.

Die beiden Beispiele zeigen, daß eine einheitliche Architektur nur für bestimmte Anwendungsklassen definiert werden kann. Daraus folgt auch, daß keine einheitliche Entwurfsmethodik für alle eingebetteten Systeme angegeben werden kann. Sinnvolle Methoden können nur für eine bestimmte wohldefinierte Zielarchitektur entwickelt werden. Diese Arbeit beschäftigt sich im wesentlichen mit Architekturen für die Anwendungen in der industriellen Automation und Kommunikation. In diesem Anwendungsgebiet können die Architekturen durch folgende Eigenschaften gekennzeichnet werden:

- Der Software-Teil wird auf einem (Ein-)Mikrocontroller-Kern abgearbeitet. Dieser Kern besteht aus einem 4-Bit, 8-Bit, 16-Bit oder 32-Bit-Mikrocontroller und verschiedenen externen Halbleiterspeichern.
- Es handelt sich im wesentlichen um synchrone und/oder asynchrone reaktive eingebettete Systeme.
- Der Hardware-Teil wird auf dedizierte *FPGA* oder *ASIC* abgebildet.

Die genaue Erklärung dieser Begriffe wurde in Kapitel 2 gegeben, die detaillierte Begrenzung der betrachteten Zielarchitektur beschreibt Kapitel 4.

Im vorangegangenen Text wurde im wesentlichen der Begriff der eingebetteten Systeme eingeführt und deren Abbildung auf verschiedene, nicht einheitlich beschreibbare Zielarchitekturen erklärt. Die charakteristischen strukturellen Eigenschaften der in dieser Arbeit betrachteten Architekturen im Bereich der industriellen Automation und Kommunikation wurde beschrieben. An dieser Stelle ist für den Leser besonders wichtig, daß diese Klasse von Architekturen aus wenigen, dafür aber hochintegrierten *VLSI*-Halbleiterbausteinen besteht. Diese Tatsache ist ausschlaggebend für das Verständnis der dadurch entstehenden Entwurfsprobleme, welche im folgenden Abschnitt 3.1 anhand steigender Systemkomplexität dargestellt werden. Die Kenntnis der entstehenden Entwurfsprobleme sind bei der Bewertung von bestehenden Entwurfsmethoden, welche anschließend in Abschnitt 3.2 diskutiert werden, von großer Bedeutung. Diese Diskussion dient abschließend in Abschnitt 3.3 als Grundlage und Motivation für die in dieser Arbeit vorgestellte Entwurfsmethodik.

3.1 Entwurfsprobleme

Dieser Abschnitt beschreibt die auftretenden Entwurfsprobleme bei steigender Komplexität der eingebetteten Systeme. Diese können in erster Näherung als eine Funktion der Wortbreite der verwendeten Mikrocontroller-Architektur aufgefaßt werden. In Abbildung 3.1 wird der Anstieg der Entwurfsproblematik in einem dreidimensionalen Raum anhand eines 8-Bit-Mikrocontrollers (MC8051), eines 16-Bit-Mikrocontrollers (C167) sowie eines 32-Bit-Mikrocontrollers (MPC860) quantitativ dargestellt. Dieser Abschnitt steht in engem Zusammenhang mit Kapitel 2, in dem die Komponenten bereits im Detail eingeführt wurden.

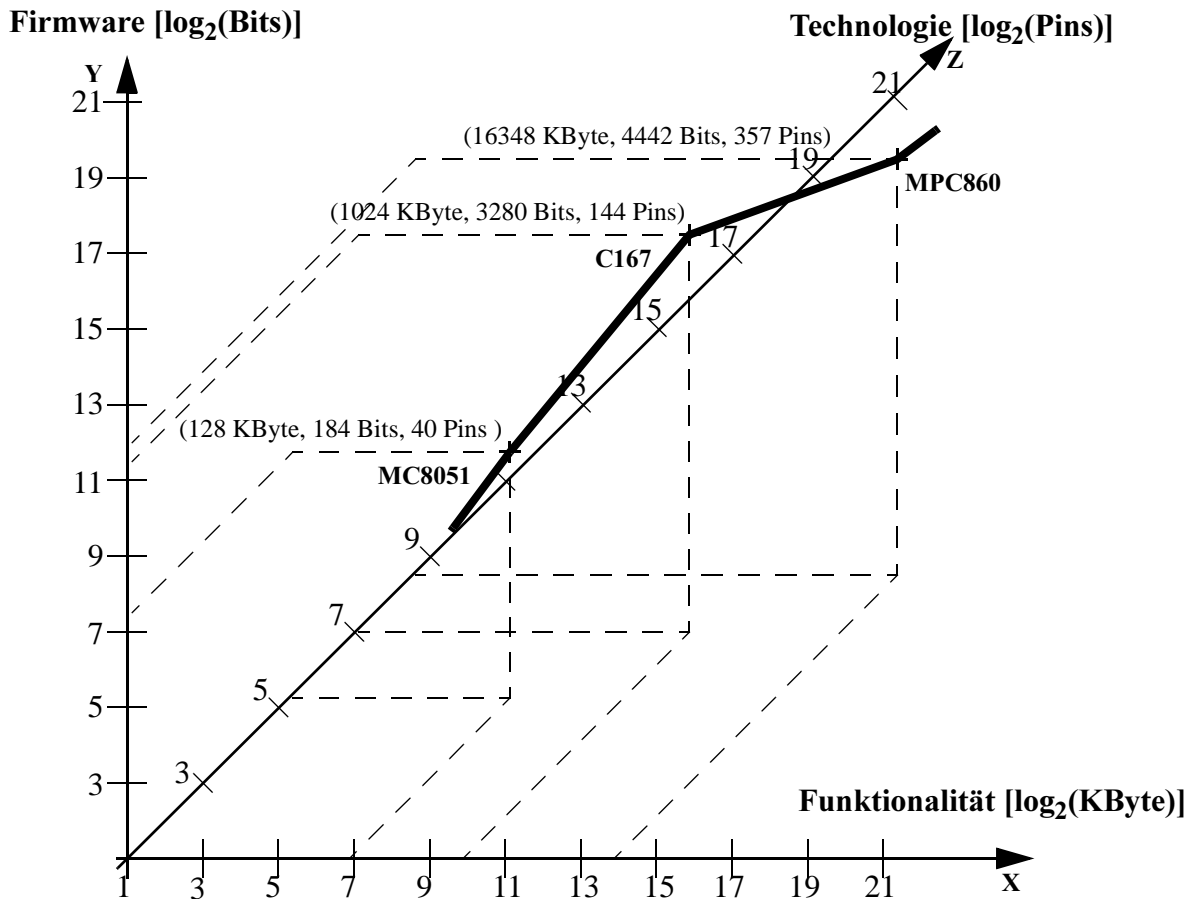


Abbildung 3.1 Quantitativer Anstieg der Entwurfsproblematik bei eingebetteten Systemen

Bei steigender Wortbreite des verwendeten Mikrocontrollers steigt zunächst in der ersten Dimension die implementierte Funktionalität an, welche man durch die Code-Größe im zur Verfügung stehenden Hauptspeicher beschreiben kann. In Abbildung 3.1 ist deshalb in X-Richtung die Hauptspeichergröße [log₂(KByte)] aufgetragen.

Die zweite Dimension besteht in der steigenden *on-Chip*-Funktionalität, welche auch als Firmware bezeichnet wird. In Abbildung 3.1 wird diese in Form der zu initialisierenden und zu kontrollierenden *on-Chip*-Register-Bits [log₂(Bits)] gemessen und auf der Y-Achse aufgetragen. Dabei handelt es sich ausschließlich um Register-Bits, die die Steuer- bzw. Statusinformationen für die *on-Chip*-Peripherie bzw. für die über den Bus-Controller extern angeschlossenen Speicher- bzw. ASIC-Bausteine betreffen. Die Bits in den Registersätzen sowie die Register mit Spezialfunktionen, welche zur eigentlichen CPU gehören, wurden dabei nicht berücksichtigt.

Die dritte Dimension wird durch die Technologie bestimmt. Als bestimmende Meßgröße kann dafür die Anzahl der Pins am Gehäuse benutzt werden. In Abbildung 3.1 wird deshalb die Pin-Anzahl $[\log_2(\text{Pins})]$ auf der Z-Achse aufgetragen.

Der gleichzeitige drastische Anstieg der Komplexität innerhalb der drei Dimensionen ist ein Indikator für die steigende Entwurfsproblematik, welche von den Entwicklern beherrscht werden muß. Um diese in einer gemeinsamen Graphik abbilden zu können, wurde für alle drei Achsen ein logarithmischer Maßstab gewählt. Die Werte für die drei Meßpunkte wurden aus [Intel94], [Sie96] und [Mot98/1] entnommen. In den folgenden Unterabschnitten werden die Entwurfsprobleme anhand der Beispiele in dieser Graphik noch detaillierter dargestellt.

3.1.1 Eingebettete Systeme mit 4-Bit-Mikrocontrollern

Die ersten verfügbaren Mikrocontroller basierten auf sehr einfachen 4-Bit breiten Architekturen. Diese Mikrocontroller haben auf dem Chip einen kleinen Programmspeicher mit einer Kapazität von wenigen hundert Bytes. Die Programmierung erfolgt in Assembler-Sprache, welche auf einem PC eingegeben und in ausführbaren Code übersetzt wird. Die Entwicklung basiert auf wiederverwendbaren, durch ultraviolettes Licht löschbaren Bausteinen. Dabei wird der binär codierte Programmcode durch ein entsprechendes Programmiergerät in dem *on-Chip-EPROM* gespeichert. Nach erfolgreichem Test wird der Programmcode in billigeren Bausteinen mit nicht-löschbaren *on-Chip-ROM* gespeichert. Die Bausteine sind auf Grund ihrer geringen Ein- und Ausgabe-Komplexität in einfachen Gehäuseformen (engl.: *Dual Inline Package - DIP*) verfügbar, die einfache Laboraufbauten gestatten. Dabei werden diese Bausteine häufig mit weiteren einfachen Logik-Gattern (sogenannte TTL-Gatter, z.B. der 74XXX-Serie) kombiniert. Diese Systeme bilden die einfachste Art von eingebetteten Systemen, die heute noch zahlenmäßig weit verbreitet sind. Viele Geräte des täglichen Bedarfs oder industrieller Anlagen basieren auf diesen Architekturen. Als Beispiel dient unter anderem die Waschmaschine oder der Elektroherd. Sie zeichnen sich durch sehr geringe Kosten aus und haben eine wesentlich höhere Zuverlässigkeit als mechanische Steuerungskomponenten.

Im allgemeinen stellen diese Systeme keine besonderen Anforderungen an die verwendeten Entwurfsmethoden, da die entstehenden Entwurfsprobleme in Bezug auf Funktionalität, Firmware und Fertigungstechnologie vom Entwickler ohne besonderen Einsatz von hochentwickelten Werkzeugen zu beherrschen sind.

3.1.2 Eingebettete Systeme mit 8-Bit-Mikrocontrollern

Wie Abbildung 3.1 zeigt, steigen die Entwurfsprobleme beim Einsatz eines 8-Bit-Mikrocontrollers (MC8051) bereits sichtbar an (siehe auch Abschnitt 2.1.1.1). Die eingebetteten Systeme basierend auf einem 8-Bit-Mikrocontroller sind, wie in [DeMi95] dargestellt, zahlenmäßig am weitesten verbreitet. Innerhalb dieser Anwendungsklasse ist der dargestellte MC8051-Baustein einer der am häufigsten eingesetzten Mikrocontroller-Typen. Der Hersteller beziffert seine jährliche Produktion auf ca. 100 Millionen Stück. Diese Verbreitung beruht auf der großen Anzahl von Derivaten, Entwicklungswerkzeugen und der Anzahl von verschiedenen Ersatzherstellern (engl.: *second source*) auf dem Markt. Kennzeichen dieser Mikrocontroller-Klasse ist die umfangreiche Peripherie, welche auf dem Chip implementiert ist. Darunter versteht man statische Halbleiterspeicher (*SRAM*), Festwertspeicher wie *EPROM* bzw. *ROM* sowie die wichtigsten Bausteine zur Systemverwaltung, wie Zeitgeber (engl.: *Timer*) und Zähler, sowie ein Controller zur Unterbrechungssteuerung. Im Bereich der Schnittstellen wird in den

Standard-Derivaten prinzipiell eine serielle (engl.: *Universal Asynchron Receiver Transmitter - UART*) und zwei parallele 8-Bit-Schnittstellen angeboten. In besonderen Ausführungen sind weitere Einheiten, wie ein Analog-Digital-Konverter, vorhanden. Um diese *on-Chip*-Peripherie zu initialisieren und während der Laufzeit zu kontrollieren, verfügt der MC8051 bereits über 23 *on-Chip*-Register mit insgesamt 184 Bits. Im Gegensatz zu den einfacheren 4-Bit-Mikrocontrollern wird beim MC8051 schon eine anspruchsvollere Firmware benötigt. Dadurch läßt sich aber weitgehend das gesamte eingebettete System auf einem einzigen Mikrocontroller-Chip implementieren, wodurch die Produktionskosten drastisch sinken. Die lange Verfügbarkeit der Bausteine über nun schon fast fünfzehn Jahre hat eine extrem hohe Zuverlässigkeit sowohl bei der Produktion der Chips als auch bei den darauf aufbauenden eingebetteten Systemen ermöglicht. Diese Tatsache ist der Grund für deren Einsatz einerseits im unkritischen Konsumbereich, andererseits bei sicherheitskritischen eingebetteten Systemen der Industrie, des Luft- und Raumfahrtbereiches oder im Bereich militärischer Systeme.

Die Programmierung läßt sich prinzipiell noch in Assembler-Code durchführen, doch erreichen die Programme mittlerweile die Größe von mehreren 10 KByte, wobei die maximale adressierbare Code-Größe im Hauptspeicher bei jeweils 64 KByte Programm-Code und 64 KByte für dynamische Daten liegt. Viele Anwendungen schöpfen diesen Speicher voll aus, weshalb in Abbildung 3.1 auf der X-Achse eine Gesamtgröße von 128 KByte eingetragen wurde. In dieser Größenordnung haben sich bereits höhere Programmiersprachen wie C etabliert. Typische Anwendungen benutzen für die meisten Programmteile entsprechende C-Compiler, nur die zeitkritischen Teile werden im abgeleiteten Assembler-Code noch zusätzlich optimiert. Da der gesamte Kern des eingebetteten Systems auf einem Chip integriert wird, steigen insbesondere die Anforderungen an die Testmethoden, weil die meisten Signale für Testmessungen von außen nicht zugänglich sind. Die Entwurfs- und Testmethoden basieren im wesentlichen auf Simulatoren und sogenannten *In-Circuit*-Emulatoren. Simulatoren werden auf einem Entwicklungsrechner ausgeführt und erlauben eine Taktzyklen-genaue Nachbildung der gesamten Chip-Architektur. Damit können funktionale Tests durchgeführt werden. Bei dieser Klasse von Anwendungen treten auch erste Echtzeitanforderungen auf. Die Testmethoden für solche Entwicklungsprobleme basieren auf dem Einsatz von Emulatoren. Dabei wird auf der Platine des eingebetteten Systems der eigentliche Mikrocontroller aus seinem Sockel genommen und durch einen speziellen Emulations-Typ ersetzt. Dieser erfüllt die gleichen funktionalen und elektrischen Eigenschaften wie der einfachere Produktionstyp. Zusätzlich ist er mit Hilfe von spezieller Hardware eng mit einem Entwicklungsrechner gekoppelt. Dadurch kann der Programm-Code in Echtzeit ausgeführt werden. Beim Auftreten von bestimmten Fehlerbedingungen kann der Programmablauf gestoppt und der innere Zustand des Mikrocontrollers analysiert werden.

Ein eingebettetes System besteht neben dem Mikrocontroller selbst noch aus zusätzlichen Standard-Bausteinen bzw. programmierbaren Logikbausteinen kleinerer Komplexität. Für die Fertigungstechnologie bedeutet dies, daß hier in der Regel noch keine besonderen Probleme entstehen. Der MC8051 selbst hat je nach Derivat zwischen 32 und 40 Anschluß-Pins und ist in einem einfachen *DIL* oder *PLCC*-Gehäuse verfügbar. Für erste Prototypen können durchaus einfache Laboraufbauten benutzt werden, da neben der Gehäuseform auch die Busfrequenzen im Bereich zwischen 12 MHz und 20 MHz relativ unkritisch sind.

Zusammenfassend ist festzustellen, daß die Entwurfsprobleme in Bezug auf die Programmierung der Anwendungs-Software sowie der Firmware des Mikrocontrollers bereits sichtbar ansteigen, die technologischen Probleme dagegen noch keine besonderen Anforderungen stellen.

3.1.3 Eingebettete Systeme mit 16-Bit-Mikrocontrollern

Der begrenzte Adressraum der 8-Bit-Mikrocontroller bei weiter steigender Programm-Komplexität und härteren Echtzeitanforderungen erfordern den Übergang zu 16-Bit-Mikrocontrollern (z.B. der SIEMENS C167, siehe Abschnitt 2.1.1.2). Wie Abbildung 3.1 zeigt, steigen die Entwurfsprobleme bei diesen eingebetteten Systemen bereits signifikant an. Die Code-Größe der Anwendungs-Software steigt dabei auf 1 MByte an. Im Gegensatz zu den eingebetteten Systemen mit 4-Bit- bzw. 8-Bit-Mikrocontrollern resultiert der Anstieg der Entwurfsproblematik nicht allein aus der steigenden Funktionalität der Anwendungs-Software des Mikrocontrollers.

Bei diesen Systemen beginnen sich zwei weitere Problemfelder signifikant bemerkbar zu machen. Um die umfangreiche *on-Chip*-Peripherie zu bedienen, steigt der Entwicklungsaufwand für die Firmware zur Initialisierung und Kontrolle der 3280 Bits auf dem Chip an, welche in insgesamt 217 vorwiegend 16-Bit breiten *on-Chip*-Registern implementiert wurden. Auch die Komplexität der peripheren Bausteine nimmt zu. Zum Einsatz kommen zusätzliche *ASIC*-Bausteine mittlerer Komplexität. Als Beispiel dienen *ASIC*-Komponenten für leistungsstärkere Schnittstellen (z.B. Ethernet) oder *ASIC*-Chips zur Ankoppelung an physikalische Prozesse. Im allgemeinen haben solche Peripherie-Komponenten eine intelligentere Eigenfunktion und benötigen auch dafür eine umfangreichere Initialisierungs- und Verwaltungs-Firmware, welche auf dem Mikrocontroller ausgeführt wird.

Ein weiteres Problemfeld erwächst aus der Tatsache, daß solche eingebetteten Systeme mit höheren Mikrocontroller- bzw. Bus-Taktfrequenzen betrieben werden, welche typischerweise im Frequenzbereich zwischen 20 MHz und 40 MHz liegen. Zusammen mit der Forderung nach mehr Signal-Pins sind diese Bausteine im allgemeinen nur noch in Gehäusen mit wesentlich feineren Pin-Abständen verfügbar. Der in Abbildung 3.1 dargestellte C167 ist in einem *TSOP*-Gehäuse (engl.: *Thin Small Outline Package - TSOP*) mit 144 Pins und einem Pin-Abstand von ca. 0,5 mm verfügbar. Das hat zur Folge, daß einfache Laboraufbauten zum Test der Prototypen nicht mehr möglich sind. Es wird schon für die ersten Prototypen eine Leiterplatte mit mehreren Signallagen benötigt, welche wegen den höheren Frequenzen und feineren Pin-Abstände besondere Anforderungen an ihre mechanischen und elektrischen Eigenschaften stellt. Typische Leiterplatten realisieren zwischen vier und sechs elektrische Signallagen, wobei zwei elektrische Lagen ausschließlich der stabilen Betriebsspannungsversorgung (+5 V und *GND*) dienen. Die Leiterbahnbreiten- und abstände liegen sich dabei im Bereich von 9 mil (ca. 228 µm).

3.1.4 Eingebettete Systeme mit 32-Bit-Mikrocontrollern

Wie in Abbildung 3.1 dargestellt, steigen die Entwurfsprobleme beim Einsatz von 32-Bit-Mikrocontrollern weiter an. In diesem Kapitel wurde bereits erwähnt, daß sich eingebettete Systeme in der industriellen Automation und Kommunikation durch wenige, dafür aber hochintegrierte *VLSI*-Bausteine in Form von Hochleistungs-Mikrocontrollern, *FPGA* und *ASIC* auszeichnen. Dabei steigen nicht nur die Entwurfsprobleme bezüglich der einzelnen Komponenten, sondern die Überlagerung der einzelnen Problemfelder kann sich zu einem signifikanten Entwicklungsrisiko akkumulieren. Diese Tatsache ist um so bedeutsamer im Hinblick darauf, daß Entwickler mit immer kürzeren Entwurfszeiträumen arbeiten müssen, um dem internationalen Innovationsdruck standzuhalten. Im folgenden werden die einzelnen Problemfelder getrennt aufgelistet, um einen möglichst vollständigen Überblick zu geben. Dabei muß auch Kapitel 2 berücksichtigt werden, in dem die Komponenten im Detail eingeführt wurden.

- **Problemfeld Mikrocontroller:** Wesentliches Kriterium bei dem Architekturentwurf für ein eingebettetes System ist die Auswahl des entsprechenden 32-Bit-Mikrocontrollers. Dabei ist nicht nur die reine Rechenleistung ausschlaggebend, sondern vielmehr die Gesamtfunktionalität aus *CPU*, Bus-Schnittstelle und komplexer *on-Chip*-Peripherie. Diese reicht von typischen Funktionen wie *Interrupt*-Controller, *Timer* und seriellen Schnittstellen bis hin zu *System-on-a-Chip*-Lösungen (z.B. beim MPC860 [Mot98/1]). Solche Bausteine verfügen über eine ausgeprägte Bus-Kontroll- und Schnittstellenfunktionalität. Dadurch können fast alle verfügbaren Speichertechnologien mit unterschiedlichen Busbreiten und Geschwindigkeiten ohne zusätzliche Hardware (siehe Abschnitt 2.1.1.3: *dynamic bus sizing*) an den Mikrocontroller angeschlossen werden. Externe Hardware muß dafür nicht mehr entworfen werden und die Komponentenzahl für das eingebettete System sinkt deutlich. Das Problem verlagert sich vom eigentlichen Hardware-Entwurf in die optimale Konfiguration und Ausnutzung der *on-Chip*-Ressourcen. Um eine Vorstellung für diese Problematik zu bekommen, wurde in Abbildung 3.1 der MPC860 eingezeichnet, welcher in 194 hauptsächlich 32-Bit breiten Registern insgesamt 4442 Bits besitzt, die von der Firmware initialisiert und zur Laufzeit kontrolliert werden müssen.

Zusätzlich folgt die Entwicklung bei diesen Mikrocontrollern der Entwicklung der Hochleistungsprozessoren in der allgemeinen Rechnertechnik. Das bedeutet, die Bausteine verfügen im Gegensatz zu den 4-Bit-, 8-Bit- oder 16-Bit-Mikrocontrollern über Cache-Speicher für Befehle und Daten auf dem Mikrocontroller-Chip, Pipeline-Strukturen in den Befehlsausführungseinheiten und Konzepte wie parallele Ausführungseinheiten, Superskalarität und *Out-of-order-Execution*. Diese Begriffe wurden ebenfalls in Abschnitt 2.1.1.3 eingeführt. An dieser Stelle wird bemerkt, daß diese Eigenschaften die Rechenleistung enorm erhöhen. Bei der Betrachtung unter harten Echtzeitbedingung allerdings sind diese Fähigkeiten die hauptsächliche Ursache für das stark unvorhersehbare Verhalten der Ausführungszeiten kritischer Programmbe-reiche. Diese Steigerung der Rechenleistung darf für die Abschätzung des schlechtesten Einsatzfalles (engl.: *worst case*) in ihrer Wirkung nicht berücksichtigt werden, da sie nicht zu jedem Zeitpunkt garantiert werden kann.

Insbesondere die Testverfahren verändern sich erheblich. Mußte der Systementwickler früher hauptsächlich Signale auf der Leiterplatten-Ebene beobachten, so muß er nun bei einem 32-Bit-Mikrocontroller die Signale auf dem Chip verfolgen. Dazu sind neue Testverfahren wie der sogenannte *Background Debug Mode (BDM)* notwendig. Diese nutzen im Chip integrierte Test-Einheiten und erfordern keine aufwendigen externen Emulatoren.

- **Problemfeld ASIC-Einbindung:** Mit Hilfe der *VLSI*-Technologie kann immer mehr Funktionalität in einem *ASIC* integriert werden. Funktionen, die in der Vergangenheit ganze Baugruppen füllten, stehen heute als preisgünstige Chips (z.B. *AAL*-Controller für B-ISDN) zur Verfügung. Sie können als in sich abgeschlossene Systeme betrachtet werden und kommunizieren mit ihrer Umgebung über gemeinsame Pufferspeicher. Die Beschreibung dieser Pufferspeicher (Anfangsadressen und Puffergröße, Zugriffsrechte, Kommunikationsparameter usw.) wird in sogenannten Deskriptoren festgelegt, welche ebenfalls in einem gemeinsamen Speicherbereich stehen. Zusätzlich können sie noch viele Initialisierungsparameter besitzen, die über eine Mikrocontroller-Bus-schnittstelle im *ASIC* programmiert werden müssen. Diese gemeinsamen Datenstrukturen und Parameter müssen von einem übergeordneten Rechner (engl.: *Host*), im allgemeinen dem Mikrocontroller, richtig initialisiert und während des Betriebes kon-

trolliert werden. Dazu ist zusätzlich zu der oben erwähnten Firmware für die *on-Chip*-Peripherie ein weiterer Firmware-Teil notwendig, welcher einen solchen *ASIC*-Baustein erst zur gewünschten Funktion bringt. Das bedeutet auch, daß ein *ASIC* nicht für sich alleine (z.B. beim Test) betrieben werden kann, sondern er benötigt immer die Anbindung an einen Mikrocontroller.

Die Testeigenschaften solcher *ASIC* sind aber größtenteils nicht so gut ausgeprägt (z.B. durch Rücklesemöglichkeiten der Initialisierungsparameter oder *BDM*). Sind alle *start-up*-Bedingungen erfüllt (Deskriptoren und Register initialisiert, gemeinsamer Speicher und Leiterplatte funktionsfähig), dann funktioniert der Chip, andernfalls ist seine Funktion nicht vorhersagbar. Diese „geht-nicht“ Mentalität ohne geeignete Analysemöglichkeiten erschwert die Einbindung und den Test in komplexen Systemen.

- **Problemfeld Technologie:** Weitere Probleme erwachsen auch aus der Technologie für die Systemfertigung. Mit den *on-Chip*-Funktionen steigt auch die Zahl der *I/O*-Pins der Bausteine. *SMD*-Gehäuse mit bis zu 300 Pins (engl.: 0,5 mm *Finepitch*) oder *Ball-Grid-Arrays* mit bis zu 600 Pins werden bereits eingesetzt. In Abbildung 3.1 ist der *MPC860* eingezeichnet, welcher in einem *Ball-Grid-Array* mit 357 Pins verfügbar ist. Durch hohe Taktfrequenzen entstehen weitere, häufig schwer zu lokalisierende und nicht-deterministische Fehler. Dabei erreichen moderne 32-Bit-Mikrocontroller zur Zeit externe Busfrequenzen bis 66 MHz. Bei den Herstellern sind bereits erste Prototypen verfügbar, welche externe Busfrequenzen bis 133 MHz erlauben.

Diese Bausteine werden auf Leiterplatten mit zwischen 8 und 16 Signallagen integriert. Dabei sinken die Leiterbahnbreiten- und abstände auf 5 mil (ca. 127 μm). Bei den Durchkontaktierungen (engl.: *vias*) kommen sogenannte partielle *vias* zum Einsatz (engl.: *buried vias*), welche nicht alle Signallagen durchdringen, sondern nur bestimmte Teile der elektrischen Lagerschichten. Damit steigt die Packungsdichte der Bauteile auf der Leiterplatte signifikant an. Diese Randbedingungen führen zu technologischen Fehlern in der Systemfertigung, die nur schwer erkannt und selbst bei Prototypen kaum behoben werden können. Flexible Probeaufbauten, wie bei der Klasse der 4-Bit-, 8-Bit-, oder teilweise noch der 16-Bit-Mikrocontroller üblich, sind bei diesen hochintegrierten eingebetteten Systemen nicht mehr möglich.

- **Problemfeld Echtzeitbetriebssysteme:** Bei dieser Klasse von eingebetteten Systemen werden immer häufiger Echtzeitbetriebssysteme eingesetzt. Die Voraussetzung dafür ist die entsprechende Rechenleistung des 32-Bit-Mikrocontrollers sowie der zur Verfügung stehende Adressraum und Speicherplatz. Typische Hauptspeichergößen liegen im Bereich von 16 MByte, wie in Abbildung 3.1 für den *MPC860* dargestellt. Dadurch wird die Implementierung von komplexen Software-Architekturen erleichtert. Die Probleme entstehen bei harten Echtzeitbedingungen, bei denen diese Echtzeitbetriebssysteme aufgrund der dynamischen Mikrocontroller-Architektur nicht mehr exakt voraussagbare Umschaltzeiten für die einzelnen *Tasks* erlauben. Auch die Portierung dieser Systeme auf das eingebettete System zeigt sich als nicht zu unterschätzender Zeitfaktor für die Entwicklung.
- **Problemfeld Entwickler:** Um die beschriebenen Komponenten zu nutzen, muß der Entwickler bereit sein, in jedem Projekt neue Bausteine einzusetzen, die eine geforderte Gesamtfunktion möglichst optimal erfüllen. Zu beobachten ist jedoch, daß diese Freiheitsgrade heute noch nicht ausgenutzt werden. Entwickler bleiben lange Zeit bei

einer Komponentengeneration, was häufig dazu führt, daß vorhandene bessere Lösungen nicht eingesetzt werden.

Jedes der genannten Problemfelder ist für sich eine wesentliche Gefahrenquelle beim Entwurf von komplexen eingebetteten Systemen, wie in [WeHeRo97] veröffentlicht wurde. Diese Publikation ist im Rahmen dieser Arbeit entstanden. Wenn diese Problemfelder sich beim Entwurf auch noch überlagern, ist das Scheitern eines Entwicklungsprojektes in vielen Fällen vorprogrammiert. Die Kenntnis dieser Problematik dient in den folgenden Abschnitten zur Diskussion bestehender Entwurfsmethoden, um deren Vor- und Nachteile zu bewerten. Diese Analyse dient als Grundlage für die anschließend vom Autor vorgestellte Methodik „*Architekturentwurf und Emulation von eingebetteten Systemen*“.

3.2 Bestehende Entwurfsmethoden

Im Abschnitt 3.1 wurden die Entwurfsprobleme aufgezeigt, welche sich mit steigender Komplexität eingebetteter Systeme nicht nur addieren, sondern durch Überlagerung zu einem unkalkulierbaren Entwicklungsrisiko steigern können. Prinzipiell ist zu erkennen, das sich die Entwurfsmethoden im wesentlichen immer auf die einzelnen Komponenten beziehen. Da eingebettete Systeme aus mehreren verschiedenen Komponenten sowohl in der Hardware als auch in der Software bestehen, findet man häufig beim Systementwurf die gleichzeitige Anwendung vieler verschiedener Methoden nebeneinander. Jede für sich bezieht sich dabei auf einen Teilaspekt einer bestimmten Komponente innerhalb des eingebetteten Systems. Diese auf die Komponenten bezogenen Methoden wurden in Kapitel 2 im Detail vorgestellt und sind zur Zeit relativ gut entwickelt.

Betrachtet man dagegen Methoden, die den Entwurf eines eingebetteten Systems auf Systemebene behandeln, ist ihre Anzahl sehr begrenzt. Nach Einschätzung vieler Wissenschaftler sind diese Methoden zusätzlich noch relativ unerforscht [Wo94] [KuAy96] [Gupta95], was eine breite Anwendung beim Entwurf eingebetteter Systeme verhindert. Versucht man die existierenden Methoden zu beschreiben, so kann man prinzipiell zwei Ansätze finden, die im folgenden Text dargestellt werden. Die erste Methode hat sich aus der täglichen Praxis beim Entwurf eingebetteter Systeme entwickelt und wird deshalb als die in der Praxis bisher eingesetzte Methode bezeichnet. Die zweite Methode wird aus den theoretischen Arbeiten im Bereich *Hardware/Software Co-Design* abgeleitet. Aktuelle Arbeiten befassen sich damit, diese Erkenntnisse auf den speziellen Anwendungsbereich der eingebetteten Systeme zu übertragen.

3.2.1 Die bisher in der Praxis eingesetzte Methode

Ausgehend von den Anforderungen der relativ einfachen eingebetteten Systeme mit 4-Bit- und 8-Bit-Mikrocontrollern entwickelte sich eine in der Praxis eingesetzte Entwurfsmethode, die selbst bei hochintegrierten eingebetteten Systemen mit einem 32-Bit-Mikrocontroller von vielen Entwicklern auch heute noch angewendet wird. In der Literatur [Gupta95] wird sie häufig auch als eine entwurfsorientierte Methode (engl.: *design-oriented approach*) bezeichnet.

Diese Entwurfsmethode ist durch einen sequentiellen Entwurfsablauf gekennzeichnet, welcher in Abbildung 3.2 dargestellt ist. Zunächst beginnt der Entwurfsprozeß mit einer Spezifikation des zu entwerfenden eingebetteten Systems. Dabei werden alle funktionalen und nicht-funktionalen Eigenschaften wie z.B. Baugröße, Leistungsaufnahme und Kosten festgelegt.

Danach findet eine Partitionierung in zwei Teile statt. Der eine Teil ist die Hardware, der andere Teil die Software, welche in Form eines entsprechenden Programm-Codes auf einem Mikrocontroller abgearbeitet wird. Dieser Partitionierungs-Schritt wird vom Entwickler bzw. von der am ganzen Projekt beteiligten Arbeitsgruppe durchgeführt und basiert im wesentlichen auf deren Erfahrung und Einschätzung, wie das gesamte System sich bei dieser Verteilung der Funktionalität und unter Einhaltung der nicht-funktionalen Randbedingungen verhalten wird.

Nach der Festlegung der beiden Teile wird zunächst der Hardware-Teil bearbeitet. In der Literatur [IySh89] [KuAy96] wird dieser Ansatz deshalb auch der „Hardware-zuerst“-Ansatz (engl.: *hardware first approach*) genannt. Dabei wird der gesamte in Hardware zu implementierende Teil auf konkrete Komponenten abgebildet. Diese Komponenten und ihre Verbindungsstruktur untereinander wird als die Hardware-Architektur bezeichnet. Die Auswahl der Komponenten ist ebenfalls stark von den Erfahrungen der Entwickler beeinflusst. Häufig werden nicht die neuesten und geeignetsten Komponenten benutzt. Die Auswahl wird auf solche Komponenten beschränkt, die in der Vergangenheit schon einmal eingesetzt wurden. Diese Vorgehensweise wird durch die Absicht motiviert, bestehendes Wissen und Erfahrungen wiederzuverwenden. Diese Beobachtung gilt insbesondere für die Auswahl bestimmter Mikrocontroller-Typen. In Abschnitt 2.1.1 wurde dieses Vorgehen anhand konkreter Komponenten näher erläutert, welches in der Vergangenheit bereits zu einem Konzentrationsprozeß auf wenige Mikrocontroller-Typen geführt hat. Nach der Architektur-Definition wird die Implementierung der Komponenten auf einer Leiterplatte durchgeführt. Während dieses Entwurfsschrittes, welcher mit einem getesteten Hardware-Prototyp endet, können die Arbeiten im Bereich der Software nicht fortgesetzt werden (dargestellt durch die gestrichelte Linie zwischen Partitionierung und Software). Diese Tatsache liegt in der Eigenschaft begründet, daß im Gegensatz zur allgemeinen Rechnertechnik bei den eingebetteten Systemen das Entwicklungssystem mit dem eigentlichen Zielsystem nicht übereinstimmt. Das bedeutet, daß zum Test der eingebetteten Software die dazugehörige eingebettete Hardware zuerst existieren muß.

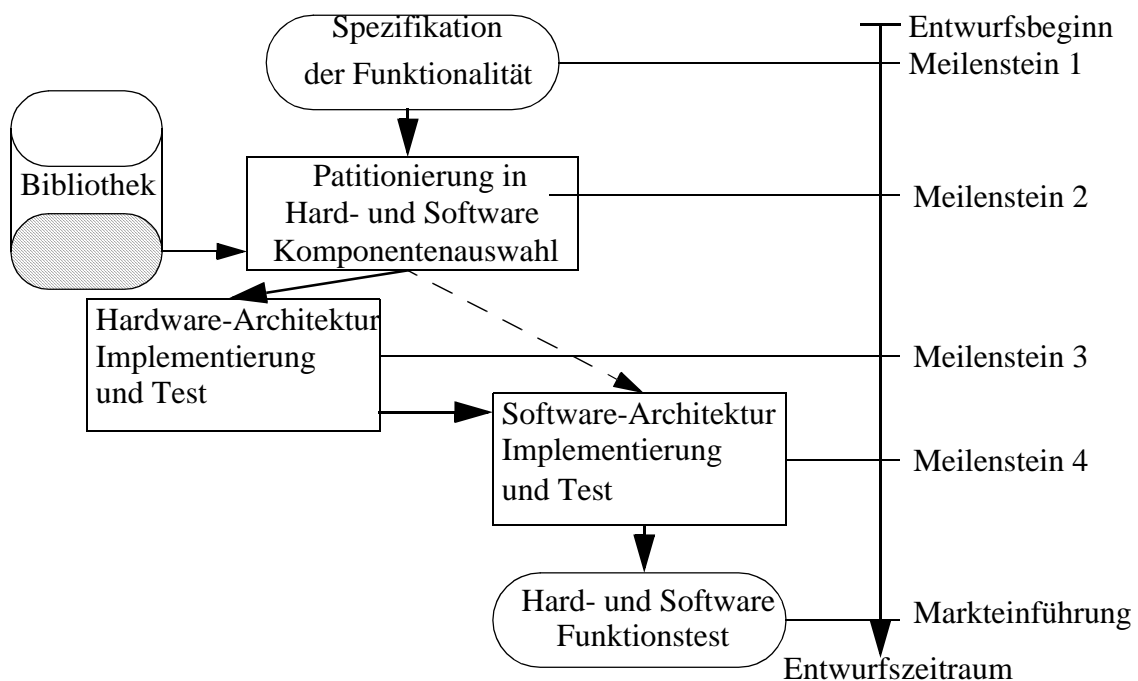


Abbildung 3.2 Entwurfsablauf bei der bisher in der Praxis eingesetzten Methode

Im nächsten Schritt wird auf der entwickelten Ziel-Hardware die dazugehörige Software implementiert und getestet. Die dazu benutzten Methoden wurden in Abschnitt 2.2 im Detail

beschrieben. Auch dieser Arbeitsschritt ist gekennzeichnet von dem Versuch, bereits in vorhergehenden Projekten geschriebene Programme wiederzuverwenden. In der Regel können die Entwickler erst in diesem Projektstadium, welches bereits relativ weit fortgeschritten ist, erkennen, ob die in der Spezifikation gemachten Forderungen eingehalten werden können.

Den letzten Schritt bildet der Funktionstest. Dabei werden alle funktionalen und nicht-funktionalen Randbedingungen der Spezifikation überprüft. Insbesondere werden alle Schnittstellen zur Umgebung des Systems angeschlossen. Hier muß das System in einem realen Einsatzfeld unter typischen und extremen Umgebungsbedingungen alle Anforderungen einhalten.

3.2.1.1 Vor- und Nachteile dieser Methode

Die Entstehung dieser Methode ist geprägt vom Entwurf der ersten eingebetteten Systeme niedriger Komplexität im Bereich der 4-Bit- und 8-Bit-Systeme. Bei der Diskussion der Vorteile ist zu nennen, daß diese Methode über lange Zeit die einzig anwendbare war, da andere Methoden gar nicht zur Verfügung standen. Sie zeichnet sich dadurch aus, daß sie sehr schnell zu ersten Implementierungsschritten führt, an denen konkrete Erkenntnisse über die Funktion gewonnen werden können. Diese ergebnisorientierte Vorgehensweise erscheint vielen Entwicklern, die in immer kürzeren Zeiträumen ihre eingebetteten Systeme entwickeln müssen, als das wesentlichste Argument. Diese Methode ist am weitesten verbreitet und wird von den meisten Entwicklern angewendet. Der Beweis, daß sie auch zu durchaus positiven Resultaten führt, kann schon an der großen Anzahl von eingebetteten Systemen unterschiedlichster Komplexität abgelesen werden, welche auf dem Markt funktionsfähig eingeführt wurden.

Bei den eingebetteten Systemen niedriger und mittlerer Komplexität wird es auch in Zukunft kaum Gründe geben, das Vorgehen wesentlich zu ändern. Der Hauptgrund liegt darin, wie der nächste Abschnitt zeigen wird, daß andere Methoden einen zu großen Aufwand im Vorfeld der eigentlichen Implementierung benötigen. Dadurch wird die Entwicklungszeit zu stark verlängert, welche anschließend durch die eventuell entstehenden Vorteile (z.B. Effizienz und Kosten) neuer Methoden nicht mehr ausgeglichen werden kann. Ein weiterer Grund liegt in den bei diesen Systemen auftretenden Entwurfsproblemen, die von den Entwicklern im Vorfeld noch relativ gut eingeschätzt werden können. Ihre Lösung kann während des Entwurfs in jedem Stadium mit überschaubarem Aufwand bewältigt werden. Als Beispiel dafür dient die bei solchen Systemen angewendete Fertigungstechnik, die jederzeit den einfachen Einsatz von Meßtechnik erlaubt, um Fehler in der Hardware oder Software zu erkennen. Durch Eingriffe in die Laboraufbauten können diese relativ einfach im Prototyp beseitigt werden. Dadurch bleibt das Entwicklungsrisiko während des gesamten Ablaufes vorhersehbar und damit beherrschbar.

Die Nachteile dieser Methode werden insbesondere beim Übergang auf komplexere eingebettete Systeme, basierend auf 32-Bit-Mikrocontrollern, sichtbar. Um diese Problematik zu erkennen, wird an dieser Stelle noch einmal auf die Darstellung der bei diesen Systemen auftretenden Problemfelder in Abschnitt 3.1.4 verwiesen. Diese Problemfelder wurden zusätzlich an konkreten Beispielen in Kapitel 2 weiter vertieft. An dieser Stelle ist die Erkenntnis besonders wichtig, daß solche Systeme aus wenigen, dafür aber hochintegrierten Komponenten bestehen. Insbesondere die Überlagerung der damit verbundenen Probleme beim Entwurf können das Entwicklungsrisiko unberechenbar machen. Um diese Gefahren beim Entwurf zu reduzieren, schränken die Entwickler die Freiheitsgrade bei der Partitionierung und Komponentenauswahl drastisch ein. Sie versuchen nur Komponenten einzusetzen, die sie schon seit längerer Zeit kennen. Dieses Verhalten führt dazu, daß nicht die innovativsten Komponenten benutzt werden, sondern diejenigen Bausteine, mit denen die Entwickler die meiste Erfahrung haben.

In [Gupta95] wird dieser Effekt ebenfalls beschrieben. Der Autor zeigt, daß neue Generationen elektronischer Komponenten (z.B. Mikrocontroller, *FPGA* und *ASIC*) alle sechs Monate neu auf dem Markt erscheinen und dabei ihre Leistung jedes Jahr um ca. 50% steigern. Diese Komponenten werden jedoch erst nach mehreren Jahren in eingebetteten Systemen implementiert. Daraus schließt der Autor auf eine Lücke in der Entwurfsmethodik für komplexe eingebettete Systeme. Die Entwurfsmethodik auf Systemebene kann mit der rasanten Weiterentwicklung im Bereich der einzelnen mikroelektronischen Komponenten nicht Schritt halten.

Die Entwickler begegnen also den schwer zu kalkulierenden Entwurfsrisiken mit der starken Einschränkung bei der Komponentenauswahl. Die Ursache dafür liegt bei dieser Entwurfsmethode in der starken sequentiellen Ausrichtung, bei der zuerst ein kompletter Hardware-Prototyp mit allen Komponenten der Hardware-Architektur erstellt wird. Dadurch können sich die Probleme der einzelnen Komponenten überlagern. Zusätzlich werden sie im Software-Entwurfsprozeß erst spät entdeckt. Die Analyse der Entwurfsfehler des Prototyps ist dann nur noch mit großem Aufwand möglich, in einigen Fällen sogar unmöglich. Häufig wird dann ein weiterer Durchlauf durch den gesamten Entwicklungsablauf notwendig, der aus Kosten- oder Zeitgründen das gesamte Projekt gefährden kann.

Um dem ständig steigenden internationalen Innovationsdruck zu begegnen, ist aber diese Zurückhaltung bei der Komponentenauswahl in der Zukunft nicht mehr akzeptabel. Bei der Entwicklung neuer und verbesserter Entwurfsmethoden ist deshalb die genaue Kenntnis der entstehenden Problemfelder, welche mit diesen hochintegrierten Komponenten zusammenhängen, besonders wichtig. Die Analyse, Bewertung und Lösung dieser Problemfelder mit Hilfe geeigneter Werkzeuge ist deswegen wesentliche Motivation der in dieser Arbeit vorgestellten Entwurfsmethodik für komplexe eingebettete Systeme (siehe Abschnitt 3.3).

3.2.2 Stand der Forschung beim *Hardware/Software Co-Design*

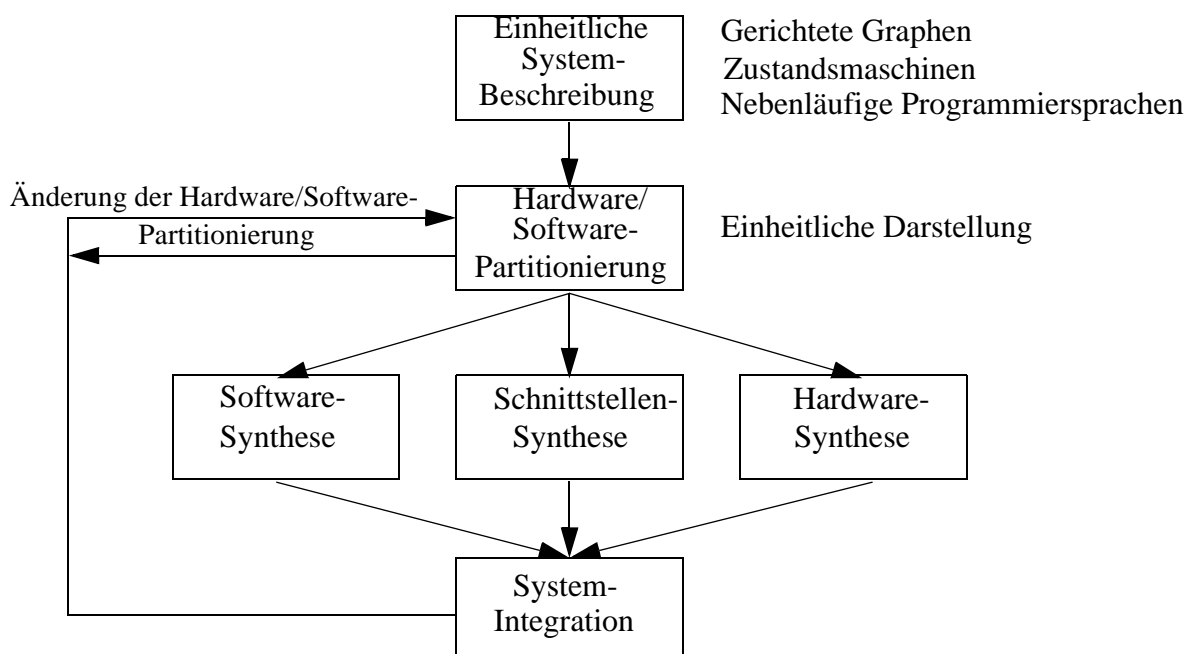


Abbildung 3.3 Entwurfsablaufdiagramm beim *Hardware/Software Co-Design*

Unter *Hardware/Software Co-Design* wird ein einheitlicher, kooperativer Ansatz für den Entwurf von eingebetteten Systemen verstanden, bei dem verschiedene Hardware- und Software-Optionen unter bestimmten Randbedingungen betrachtet werden. Eine detaillierte Einführung wird u.a. in [KuAy96] und [Gupta95] gegeben. Der Ablauf eines typischen Entwurfes ist in Abbildung 3.3 dargestellt.

3.2.2.1 Einheitliche Systembeschreibung

Ein wesentlicher Punkt bei diesem Ansatz ist eine einheitliche Systembeschreibung für Hardware und Software. Eine solche Beschreibungsform erlaubt zunächst eine von der Hardware- und Software-Implementierung unabhängige Betrachtung des Systems und ermöglicht so die Erkennung der vorhandenen Freiheitsgrade innerhalb des Systems, bevor die eigentliche Partitionierung durchgeführt wird. Dabei ist es zur Zeit nicht möglich, mit einer einzigen Beschreibungssprache alle Aspekte eines Systems zu erfassen. Deshalb müssen mehrere Beschreibungsverfahren gleichzeitig eingesetzt werden.

Graphen-basierte Darstellungsformen wie der Datenfluß-Graph oder Petri-Netze können zur unabhängigen Modellierung von Hardware und Software benutzt werden. Dabei können beispielsweise die Knoten dieser Graphen eine Operation in der Software oder verschiedene Funktionseinheiten in der Hardware modellieren. Diese Darstellungsform eignet sich damit zur Modellierung des Daten- und Kontrollflusses, insbesondere bei parallelen Funktionsabläufen.

Zustandsautomaten (engl.: *Finite State Machines - FSM*) eignen sich besonders zur Modellierung des Systemverhaltens in Bezug auf seine Umwelt. Diese *FSM* werden deshalb zur Modellierung von reaktiven Echtzeitsystemen benutzt. Eine weitere Methode zur Darstellung bieten nebenläufige Programmiersprachen (z. B. *Verilog* oder *VHDL*), welche die parallele Modellierung von Funktionsabläufen ermöglichen. Sie haben den großen Vorteil, daß sie direkt synthetisiert und dadurch sehr effizient implementiert werden können.

Mit Hilfe von diesen verschiedenen Systembeschreibungsmethoden wird es möglich, bestehende Freiheitsgrade frühzeitig zu erkennen und mehrere Implementierungsalternativen in einem sehr frühen Stadium innerhalb des Entwurfsablaufes zu bewerten. Ein wesentlicher Ansatzpunkt dabei ist, sehr Rechenzeit-intensive Teilfunktionen zu erkennen, um sie dann bei der Implementierung auf möglichst geeigneten Hardware-Komponenten abzubilden. Eine weitere Möglichkeit besteht darin, die weniger Rechenzeit-intensiven Teilfunktionen auf eventuell vorhandene und versteckte Systemressourcen abzubilden. Dadurch erhält man eine möglichst gute Ausnutzung aller Ressourcen des Gesamtsystems (z.B. die Rechenleistung des Mikrocontrollers bei der Software bzw. freie Logik-Gatter auf einem *ASIC* oder *FPGA* bei der Hardware).

Ein weiterer wichtiger Aspekt ist das Bestreben, den Entwurfsablauf formal zu beschreiben, um daraus geeignete Werkzeuge ableiten zu können. Dieser erste Teilschritt wird in der Literatur [KuAy93] mit folgenden Begriffen beschrieben:

- **Begriff „Funktion“:** Eine Abbildung von Eingangssignale auf Ausgangssignale, welche auch auf Zuständen basieren kann.

Dieser Begriff beschreibt die Abbildung eines Eingangsvektors und eines aktuellen Zustandsvektors auf einen Ausgangsvektor und einen neuen Zustandsvektor. Ausgehend von dieser Systemfunktion wird anschließend eine Dekomposition durchgeführt.

- **Begriff „funktionale Dekomposition“:** Prozeß der Entwicklung einer Implementati-on einer Funktion, ausgedrückt in einem Algorithmus, welcher einen Satz von einfacheren Teilfunktionen verbindet.

Funktionale Dekomposition ist ein von oben nach unten (engl.: *top down*) durchgeführter Verfeinerungsprozeß. Dabei entsteht aus einer komplexen Funktion in einer baumartigen Struktur ein Satz von einfacheren Teilfunktionen. Diese Teilfunktionen beschreiben dabei sehr intime Kenntnisse über die zugrundeliegenden Datenstrukturen.

3.2.2.2 Hardware-Software Partitionierung

- **Begriff „Hardware/Software-Partitionierung“:** Ein Prozeß, in dem bestimmt wird, welche Teilfunktionen in Hardware und welche in Software implementiert (bzw. durchgeführt) werden sollen.
- **Begriff „Hardware/Software-Partition“:** Eine Beschreibung, welche die Teilfunktionen spezifiziert, die in Hardware oder Software implementiert werden sollen.

Die Hardware/Software-Partition ist das Resultat des Partitionierungsprozesses und bildet somit das Ergebnis der Partitionierungsstufe (siehe Abbildung 3.3).

Bis zu diesem Punkt wurde dargestellt, daß man eine Gesamtfunktion in viele verschiedene Hardware/Software-Partitionen zerlegen kann. Daraus resultieren verschiedene Implementierungsmöglichkeiten, die im folgenden Text Hardware/Software-Alternativen genannt werden.

- **Begriff „Hardware/Software-Alternative“:** Eine mögliche Hardware/Software Implementierung einer Systemfunktion.

Eine Hardware/Software-Alternative für die Systemfunktion besteht aus einer Menge von Software-Einheiten, Hardware-Einheiten und ihre Kommunikation untereinander. Diese Software- und Hardware-Einheiten bilden somit die funktionalen Grundbausteine, welche zur Implementierung der Systemfunktion benutzt werden. Das wesentlichste Ziel bei der Evaluierung liegt in der Aufgabe, aus der Menge der Alternativen die geeignetste Alternative auszuwählen. Diese Auswahl wird bestimmt durch die einzuhaltenden Randbedingungen, welche in sogenannten *trade-off*-Funktionen beschrieben werden.

- **Begriff „Hardware/Software trade-off“:** Eine Entscheidung bezüglich der Allokierung von Funktionen in Hardware und Software. Diese Entscheidung versucht eine Menge von Kriterien zu erfüllen. Viele dieser Kriterien führen bei dem Versuch ihrer Einhaltung zu Konflikten untereinander und sind somit nicht miteinander verträglich.

Die Ausnutzung von Hardware/Software *trade-offs* basiert auf dem Prozeß der Bildung von verschiedenen Hardware/Software-Alternativen sowie deren Bewertung bei der Implementierung. Entscheidend bei der Bewertung ist dabei die Einhaltung der Randbedingungen (z. B. Leistungsaufnahmen, Kosten, Ausführungszeiten, Formfaktoren).

Die Bewertung der verschiedenen Alternativen erfordert zwei Schritte. Der erste Schritt basiert auf der Bewertung der einzelnen Alternativen untereinander, der zweite Schritt bewertet die Verwendung einer Alternative im Gesamtsystem. Dieser Bewertungsprozeß wird qualitativ an einem Bewertungsmodell durchgeführt, welches aus folgenden fünf Teilkomponenten besteht:

- Die erste Komponente ist eine Menge von Metrik-Funktionen, welche für jede interessierende Meßgröße (z.B. Ausführungszeit, Kosten, Zuverlässigkeit usw.) einen Wert m_j bestimmt. Als Eingangsgröße wird jeweils eine Alternative benutzt, wobei die Funktion der jeweiligen eingesetzten Alternative einen quantitativen Wert m_j zuweist. Man kann damit m_j interpretieren als einen Wert, der alle Alternativen bezüglich einer bestimmten Eigenschaft relativ zueinander in Beziehung setzt.
- Die zweite Komponente repräsentiert Gütefunktionen, die den Wert m_j benutzen und daraus einen Gütewert g_j berechnen. Dabei variiert g_j im Wertebereich zwischen null und eins.
- Die dritte Komponente faßt die Gütewerte bezüglich einer bestimmten Metrik zusammen. Wurden beispielsweise fünf Alternativen bezüglich Ausführungszeit und Kosten bewertet, beinhaltet diese Komponente zwei Elemente, nämlich g_T und g_C .
- Die vierte Komponente besteht aus Gewichtswerten, welche die Bedeutung der einzelnen Metriken bestimmen. Jedes Gewicht w_j bestimmt die Gewichtung einer bestimmten Metrik j . Die Summe der einzelnen Gewichte addieren sich dabei zu einem Gesamtwert von eins.
- Die fünfte Komponente bestimmt eine Menge von Qualitätswerten für die einzelnen Alternativen.

Wenn mit Hilfe dieses Formalismus eine bestimmte Alternative als die beste bezüglich der Einhaltung der Randbedingungen ermittelt wurde, schließt sich, wie in Abbildung 3.3 dargestellt, die eigentliche Implementierung an. Dabei werden die Software-Funktionen in ausführbaren Code übersetzt. Typische Werkzeuge dabei sind Hochsprachen-Compiler für C oder C⁺⁺. Die Hardware-Funktionen, welche im Partitionierungsschritt in Form von Synthese-fähigem *VHDL*-Code entstanden sind, werden in entsprechende Netzlisten überführt, welche dann auf die gewünschte Zieltechnologie abgebildet werden. Software- und Hardware-Synthese werden mittlerweile durch entsprechende Werkzeuge gut unterstützt. Die automatische Generierung der entsprechenden Kommunikation in Form von Schnittstellen zwischen der Software (ausgeführt auf einem Mikrocontroller) und der Hardware (implementiert in einem *FPGA* oder *ASIC*) ist zur Zeit noch nicht möglich und benötigt die massive Unterstützung des Entwicklers.

Die Systemintegration führt alle Teilfunktionen zusammen und überprüft, ob alle Anforderungen der Spezifikation eingehalten werden. Bei der Verletzung von bestimmten Anforderungen oder zur Optimierung des Gesamtsystems kann eine Änderung der Partitionierung vorgenommen werden. Danach wird der Zyklus erneut durchlaufen. Durch diese Iterationen werden systematisch die Anforderung erfüllt bzw. optimiert.

3.2.2.3 Diskussion der Vor- und Nachteile

Dieser Ansatz versucht die Auswahlentscheidung für bestimmte Komponenten anhand streng formal beschreibbarer Kriterien abzuleiten und steht somit im direkten Gegensatz zur praxisorientierten Methode, die im wesentlichen auf den subjektiven Erfahrungen der Entwickler beruht. Durch die Bewertung von verschiedenen Alternativen im Vorfeld der eigentlichen Implementierung sollen die wichtigsten Probleme, insbesondere Probleme bezüglich der Einhaltung von verschiedenen Randbedingungen, sichergestellt werden. Dadurch wird versucht,

eine teure und langwierige Wiederholung des gesamten Entwurfsablaufes zu verhindern, wenn bei der Systemintegration die Spezifikation nicht eingehalten werden kann.

Ein weiterer Vorteil liegt im optimalen Entwurf des Gesamtsystems. Diese Methode setzt dabei zwei Annahmen voraus:

- Die erste Annahme beruht darauf, daß bei der Bewertung von verschiedenen Alternativen für die Implementierung von Teilfunktionen sich die beste Alternative bezüglich der Ausnutzung aller Systemressourcen durchsetzt.
- Die zweite Annahme beruht darauf, daß die Optimierung aller Teilfunktionen die unmittelbare Optimierung der Gesamtfunktion des Systems zur Folge hat.

Aufgrund dieser Annahmen geht man davon aus, daß am Ende des Entwurfszyklus ein System steht, daß weder unterdimensioniert noch überdimensioniert ist und dennoch alle Anforderungen der Spezifikation einhält. Die formale Darstellungsweise aller Schritte bewirkt die mögliche Umsetzung in Rechner-basierte Entwurfswerkzeuge, die zusätzlich die Entwurfszeit drastisch verkürzen.

Bei der Betrachtung der Nachteile muß sicherlich bemerkt werden, daß diese Methode noch weitgehend Gegenstand der Forschung ist und nicht wie die in der Praxis bisher eingesetzte Methode schon über viele Jahrzehnte eingeführt ist. Ein erster Kritikpunkt liegt in der Tatsache, daß der betrachtete Entwurfsraum nicht eindeutig beschrieben wird. Man motiviert die Einführung dieser Methode mit der immer weiter steigenden Komplexität von eingebetteten Systemen, welche in immer kürzeren Zeiträumen entworfen werden müssen. Dabei werden im allgemeinen keine Einschränkungen gemacht, was die Zuordnung der Methode zum Entwurf von konkreten Systemen deutlich erschwert. In der Literatur, z. B. [KuAy96], werden die bestehenden Ansätze dagegen immer an Beispielen kleinerer Komplexität evaluiert, wie z.B. der Implementierung einer Fourier-Transformation. Über den Entwurf von komplexen eingebetteten Systemen, wie z.B. eines Autopiloten im Flugzeug, wurde dagegen noch nicht berichtet.

Wesentlich wichtiger bei dieser Diskussion sind jedoch die grundlegenden Aspekte bei der Durchführung dieser Methode. Sie baut auf zwei Ansatzpunkten auf, die im folgenden Text näher betrachtet und analysiert werden.

- Der erste Ansatzpunkt ist die geforderte einheitliche Systembeschreibung. Die Tatsache, daß es zur Zeit keine einheitliche Darstellungsmethode gibt, die alle Eigenschaften eines eingebetteten Systems beschreiben kann und deshalb mehrere verschiedene Methoden gleichzeitig benutzt werden müssen, ist dabei noch nicht einmal das größte Problem. Wesentlich schwieriger ist die Tatsache, daß ein einzelner Entwickler oder ein Entwickler-Team sehr detaillierte Kenntnisse über das interne Systemverhalten in die Beschreibung einbringen muß, um vernünftige Ergebnisse bei der Bewertung zu bekommen. Betrachtet man komplexere Systeme, wie z.B. das Segmentieren von Übertragungszellen bei *ATM* (siehe Kapitel 6), so hat ein Entwickler dieses Wissen auf Systemebene nicht mehr. Dies hängt damit zusammen, daß die Entwickler aus Gründen der Komplexität auf Systemebene keinen Einblick mehr in das Verhalten von wesentlichen Teilfunktionen haben, sondern diese als fertige Komponenten übernehmen müssen. Auch das Verhalten dieser Komponenten innerhalb eines übergeordneten Systems läßt sich nur sehr schwer nachbilden und beschreiben. Der Freiheitsgrad, bestimmte Funktionen von der Software in die Hardware bzw. umgekehrt zu verschieben, besteht auch nicht mehr, da die Funktionalität dieser Komponenten fest vorgegeben ist. Das Beispiel zeigt, daß eingebettete Systeme, welche aus wenigen, da-

für aber hochintegrierten Komponenten bestehen, mit dieser Methode nur sehr schwer zu entwerfen sind, da die Detail-Informationen auf übergeordneter Systemebene nicht mehr zur Verfügung stehen. Auch die Freiheitsgrade, Teilfunktionen zwischen den Komponenten bei der Bewertung des Systems zu verschieben bestehen nicht mehr.

- Der zweite Ansatzpunkt baut auf der Tatsache auf, daß die aus der einheitlichen Systembeschreibung abgeleiteten Hardware-Software-Partitionierungen anhand verschiedener Implementierungsalternativen bewertet werden müssen. Dieses Vorgehen setzt voraus, daß man Wissen über das Verhalten der betrachteten Teilfunktion auf den verschiedenen Ziel-Architekturen, aus denen die verschiedenen Alternativen bestehen, gewinnt und dieses Wissen dann in die Bewertung einführt. Dieser Sachverhalt spiegelt sich in dem Bewertungsmodell wieder, bei dem für die verschiedenen Alternativen für jede Metrik die entsprechenden Güterwerte gefunden werden müssen. Für nicht-funktionale Eigenschaften, wie z.B. die Kosten, ist dieses vielleicht noch möglich. Bei der Bestimmung funktionaler Eigenschaften, wie z.B. der Ausführungszeit, ist dieses für komplexe Komponenten nicht mehr möglich, da auch hier das interne Verhalten auf Systemebene im allgemeinen nicht bekannt ist.

Betrachtet man mögliche Anwendungsfelder für diese Methode, so muß man von Entwicklungen ausgehen, bei denen der vollständige Entwurf des Systems in bezug auf Hardware und Software durchgeführt wird. Als Beispiel dient der Entwurf eines *RISC*-Mikrocontollers, bei dem die Hardware in Form des Chips und die Software durch Generierung entsprechender Ausnahmesprünge (engl.: *expection traps*) bewertet werden soll. Hier haben die Entwickler alle internen Informationen und die Freiheitsgrade, die Hardware-Software-Grenzen frei zu verschieben. Ob man den Entwurf eines solchen Chips schon als eingebettetes System bezeichnen kann, ist sicherlich von dem Standpunkt des Betrachters abhängig.

Faßt man die Ausführungen zusammen, so läßt sich zeigen, daß diese Methode für eingebettete Systeme, bestehend aus wenigen hochkomplexen Komponenten, nicht geeignet ist. Die wesentlichen Probleme bestehen bei der Anwendung der Methode, da die benötigten internen Systemkenntnisse bei der Bewertung nicht zur Verfügung stehen bzw. nur sehr schwer zu gewinnen sind. Ein weiterer Punkt liegt in dem fehlenden Freiheitsgrad der offenen Hardware-Software-Grenzen, die hier im allgemeinen auch nicht vorhanden sind. Da eingebettete Systeme mit komplexen hochintegrierten Komponenten insbesondere auch Gegenstand dieser Arbeit sind, bilden die bis zu diesem Punkt gemachten Einführungen und Erkenntnisse die Motivation zur Definition einer speziellen Methode für diese Systeme, die im nächsten Abschnitt eingeführt wird.

3.3 Architekturentwurf und Emulation

Vergleicht man die in den beiden letzten Abschnitten vorgestellten Methoden miteinander, so kann man sagen, daß die Benutzung der in der Praxis bisher eingesetzten Methode zu einem eingebetteten System führt, ohne dabei das genaue interne Systemverhalten zu kennen. Die Architektur des zu entwerfenden eingebetteten Systems basiert im wesentlichen auf der Erfahrung und der subjektiven Einschätzung des Verhaltens der einzelnen Komponenten durch die Entwickler. Die Methode des *Hardware/Software Co-Designs* erkennt dieses Problem, leidet aber selbst unter der Tatsache, daß es keine geeigneten Werkzeuge gibt, um diese Methode effizient durchzuführen. Ein wesentliches Problem dabei ist, geeignete und sinnvolle Informationen über

das jeweilige interne Systemverhalten zu bekommen, um die Bewertungs-Algorithmen mit geeigneten Eingabewerten zu bedienen.

In diesem Abschnitt wird zunächst die entsprechende Zielarchitektur der in dieser Arbeit betrachteten eingebetteten Systeme eingeführt. Anhand dieser konkreten Klasse von eingebetteten Systemen wird dann eine dafür geeignete problemorientierte Entwurfsmethode abgeleitet. Diese stellt dabei durchaus eine Synthese der beiden bereits vorgestellten Methoden dar, indem die wichtigsten Erkenntnisse dieser Methoden analysiert werden und je nach Eignung auf die hier zu definierende anwendungsorientierte Methode übertragen werden.

In Abbildung 3.4 wird eine allgemeine Architektur eines eingebetteten Systems dargestellt. Dabei handelt es sich um einen Mikrocontroller mit entsprechenden Speicherbausteinen. Mit dem Mikrocontroller-Bus sind in der Regel *ASIC*-Bausteine verbunden, die für die Anwendung dedizierte Funktionen ausführen. An dieser Stelle ist es wichtig festzustellen, daß der Einsatz dieser *ASIC*-Komponenten unverzichtbar ist und jeder eigenen Hardware-Entwicklung aus Gründen der Entwicklungszeit und der Kosten unbedingt vorzuziehen ist. Diese *ASIC*-Bausteine stellen in der Regel auch die Schnittstelle zur Umwelt dar. Schraffiert dargestellt sind *FPGA*-Bausteine, welche immer dann zum Einsatz kommen, wenn bestimmte Funktionen nicht als fertige Komponenten zur Verfügung stehen. Sie bilden eine geeignete Zieltechnologie für die Emulation eigener Hardware-Entwicklungen. In Kapitel 2 wurde auf die Bedeutung der einzelnen Hardware-Komponenten genauer eingegangen.

Zur Zeit ist ein weiterer Trend erkennbar. Der in der allgemeinen Rechnertechnik bereits eingeführte *PCI*-Bus setzt sich auch im Bereich der eingebetteten Systeme als Standard-Bus zur Verbindung komplexer *ASIC*-Komponenten mit dem Mikrocontroller durch. Zukünftige Mikrocontroller werden dafür bereits eine *PCI*-Schnittstelle mit auf dem Chip integriert haben. Momentane Systeme benötigen dafür noch einen weiteren Chip, eine sogenannte *PCI-Bridge*.

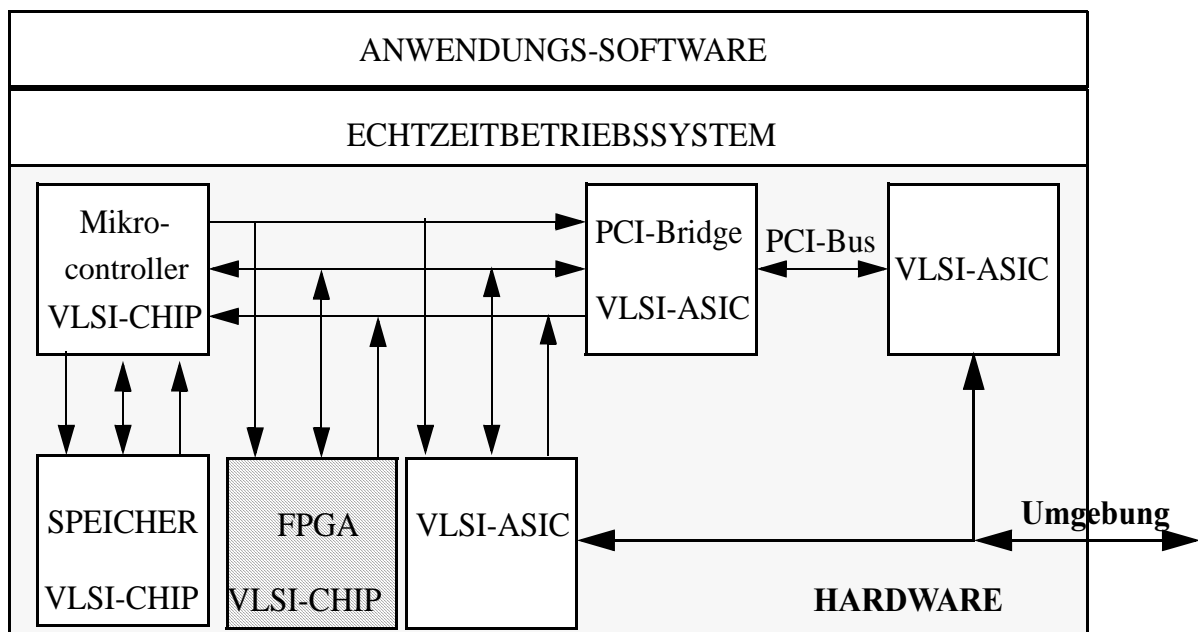


Abbildung 3.4 Allgemeine Architektur der betrachteten Klasse von eingebetteten Systemen

Die drastischen Steigerungen der Rechenleistung von 32-Bit-Mikrocontrollern forcierte auch den Einsatz von Echtzeitbetriebssystemen, welche als unmittelbare Software-Schicht über

der Hardware liegen. Sie ermöglicht die effiziente Implementierung komplexer Anwendungs-Software, welche in Form von verschiedenen *Tasks* auf dem Echtzeitbetriebssystem läuft.

Betrachtet man die in der Praxis bisher eingesetzte Methode in Abschnitt 3.2.1, so zeichnet sie sich durch ein sehr zielgerichtetes Vorgehen aus. Dieser Ansatz, durch konkrete Implementierungen möglichst schnell zu Ergebnissen zu gelangen, wird auch in der hier vorgestellten Methode prinzipiell weiter angewendet. Der wesentliche Nachteil, daß Entwickler aus Gründen der Minimierung des Entwurfsrisikos die Komponentenauswahl drastisch einschränken, ist in Zukunft nicht mehr länger vertretbar. Dieser Aspekt wird in der ersten der insgesamt zweistufigen Vorgehensweise in der hier vorgestellten Methode besonders berücksichtigt:

- **Erste Stufe: Architekturentwurf durch Bewertung**

Wesentlicher Zweck ist die Bewertung der mit komplexen Komponenten verbundenen Problemfelder. Daraus wird die Qualität der Eignung einer Komponente für das eingebettete System bestimmt. Dabei wird die Komponentenauswahl nicht mehr auf bestimmte Komponenten eingeschränkt, sondern es werden alle zur Entwicklungszeit zur Verfügung stehenden Komponenten betrachtet. Eingangsfaktoren für diese Bewertung sind zum einen die Funktionalität einer Komponente, zum anderen das Entwicklungsrisiko, das mit ihrer Verwendung verbunden ist. Die besondere Bedeutung der dafür verantwortlichen Problemfelder wurde in Kapitel 2 und Abschnitt 3.1 anhand konkreter Hardware-Komponenten ausführlich dargestellt. In Kapitel 4 wird anhand dieser Vertiefung der Bewertungsprozeß im Detail beschrieben.

- **Zweite Stufe: Emulation des eingebetteten Systems**

In dieser Stufe wird die Auswahlentscheidung aus der ersten Stufe überprüft. Als geeignete Methode wird die Emulation angewendet. Dabei wird jede Komponente zunächst einzeln betrachtet, wobei reale Echtzeitbedingungen bereits in diesem Entwurfsschritt zugelassen werden. Wesentliches Merkmal in diesem Schritt ist die dabei durchgeführte Trennung der einzelnen Problemfelder, da eine Überlagerung zu einem unkalkulierbaren Risiko führen würde. Erst wenn diese Emulation die Aussagen aus dem Datenblatt bezüglich funktionaler und nicht-funktionaler Eigenschaften bestätigt, wird die Komponente in die Systemarchitektur übernommen.

Der zweistufige Ablauf der Entwurfsmethode ist in Abbildung 3.5 dargestellt. Nach der Spezifikation der Funktionalität wird durch den Entwickler eine initiale Partitionierung in Hardware und Software durchgeführt. Die Hardware-Funktionen werden auf entsprechende Komponenten aus einer Hardware-Bibliothek abgebildet, z.B. Mikrocontroller, Speicherbausteine, *ASIC*- und *FPGA*-Chips. Kennzeichen der initialen Komponentenauswahl ist, daß die entstehende Architektur aus wenigen, dafür aber hochintegrierten Komponenten besteht. Die Software- bzw. Firmware-Funktionen werden entweder direkt auf den Instruktionssatz der zugrunde liegenden Mikrocontroller-Hardware abgebildet oder auf ein dazwischenliegendes Echtzeitbetriebssystem.

Anschließend werden die bei der initialen Komponentenauswahl erkannten Komponenten bewertet. Die bei der Hardware-Software-Partitionierung entstandenen Funktionen können prinzipiell auf mehrere alternative Komponenten abgebildet werden. Der Bewertungsprozeß liefert für die einzelnen Funktionen eine Komponentenauswahl. Die Eigenschaften der Komponente mit der besten Eignung wird anschließend in der zweiten Stufe durch Emulation im Detail überprüft. Dieser Entwurfsschritt verfolgt dabei die gleiche Absicht, wie der in Abschnitt 3.2.2 vorgestellte *Hardware/Software Co-Design*-Ansatz, nämlich die frühzeitige Bewertung von

verschiedenen Entwurfsalternativen. Allerdings wird hier nicht versucht, Teilfunktionen über Komponentengrenzen zu verschieben, sondern es werden die mit einer Komponente verbundenen funktionalen Eigenschaften und Entwurfsrisiken bewertet. Die Informationen für diese Bewertung können alle der entsprechenden Beschreibung im dazugehörigen Datenblatt oder Benutzerhandbuch entnommen werden.

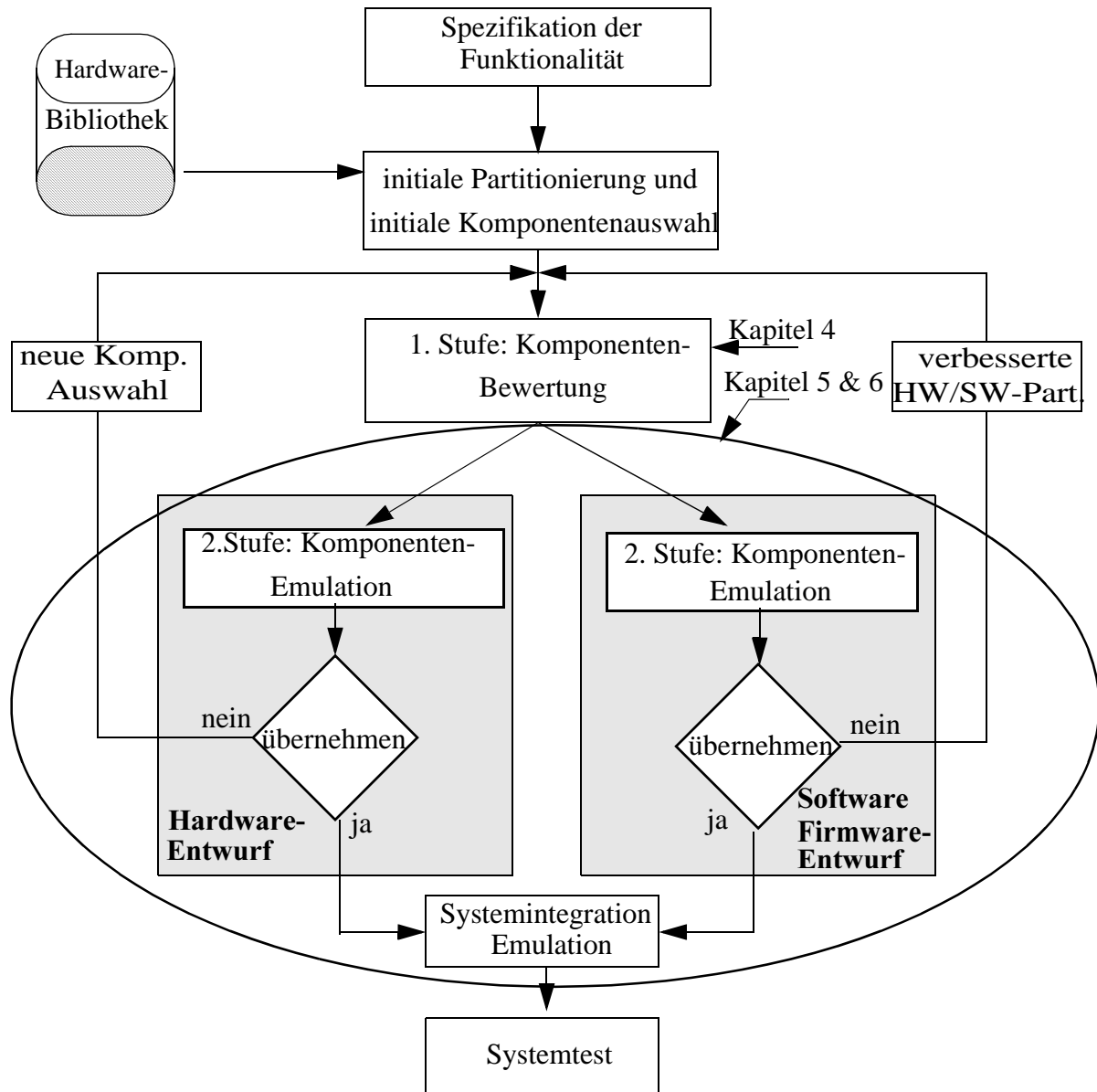


Abbildung 3.5 Entwurfsablauf: Architekturentwurf und Emulation von eingebetteten Systemen

Der zweite Schritt der Methode besteht aus der parallelen Emulation der einzelnen Hardware-Komponenten sowie der mit ihnen verbundenen Software- bzw. Firmware-Komponenten. Dabei werden die einzelnen Problemfelder möglichst getrennt bearbeitet. Erst wenn eine Komponente alle beschriebenen Eigenschaften auch in der Emulation unter Echtzeitbedingungen einhält, wird sie als Komponente für die eigentlich zu entwickelnde Systemarchitektur übernommen. Dabei wird sie während der Systemintegration mit den anderen Komponenten gemeinsam betrachtet. Diese Überprüfung wird auch in diesem Schritt durch Emulation

durchgeführt. Der abschließende Systemtest überprüft die Einhaltung der in der Spezifikation festgelegten Anforderungen.

Um die in dieser Methode durchgeführte Vorgehensweise zu präzisieren, wird zunächst auf Kapitel 2 verwiesen. Im Abschnitt 3.1 wurde ebenfalls deutlich, warum in dieser Arbeit soviel Wert auf den Architekturentwurf unter spezieller Berücksichtigung der Problemfelder der einzelnen Komponenten gelegt wird, welcher in Kapitel 4 im Detail eingeführt wird. Um insbesondere die in dieser Arbeit benötigte Emulation durchführen zu können, werden spezielle Werkzeuge benötigt. In Kapitel 5 wird dazu die Emulations-Plattform SPYDER vorgestellt. Als Ergebnis der Anwendung dieser Methode werden in Kapitel 6 zwei Beispiele beschrieben, die zeigen werden, wie mit Hilfe der Werkzeuge konkrete Entwicklungsprojekte erfolgreich durchgeführt werden können. Kapitel 7 faßt die Ergebnisse zusammen und gibt einen Ausblick über die weitere Arbeit.

Kapitel 4

Architekturentwurf durch Bewertung

Dieses Kapitel beschreibt detailliert die erste Stufe der in Abschnitt 3.3 formulierten Entwurfsmethodik. Da der Entwurfsraum für eingebettete Systeme sehr groß ist, wird in Abschnitt 4.1 zunächst eine bestimmte Teilklasse von eingebetteten Systemen definiert. Die Einschränkungen dieser Teilklasse geben Randbedingungen vor, welche die Struktur und Anzahl möglicher Zielarchitekturen begrenzen. Diese Randbedingungen sind notwendig, um einen wohldefinierten Ansatz für eine systematische Entwurfsmethodik für eingebettete Systeme dieses Teilraumes zu formulieren.

Um eine Systemarchitektur zu finden, müssen zuerst geeignete Komponenten ausgewählt werden. In Abschnitt 4.2 wird die im wesentlichen auf Erfahrung basierende Vorgehensweise eines Entwicklers bei dieser Auswahl analysiert. Die bereits in Abschnitt 3.1 motivierten Problemfelder werden bei dieser Auswahl in Form von entsprechenden Entscheidungsfeldern berücksichtigt. Für jedes dieser Entscheidungsfelder wird ein dazugehöriger (Teil-) Auswahlalgorithmus abgeleitet, der den Bewertungsprozeß innerhalb des Entscheidungsgebietes beschreibt. Dabei muß dieser Algorithmus möglichst auf alle in der betrachteten Klasse vorkommenden Komponenten anwendbar sein und bildet gleichzeitig den Ansatz für einen systematischen Architekturentwurf durch Bewertung von Komponenten, welche vorwiegend aus *ASIC*-Bausteinen bestehen.

Um einen Algorithmus zu durchlaufen, müssen Entscheidungsparameter unter Einhaltung bestimmter Randbedingungen entsprechend bewertet werden. Die Randbedingungen werden von der Spezifikation oder dem selektierten Mikrocontroller vorgegeben. In Abschnitt 4.3 werden alle dazu notwendigen Entscheidungsparameter für die einzelnen Entscheidungsfelder identifiziert und bewertet.

Alle Teilentscheidungen innerhalb der einzelnen Entscheidungsfelder führen zu einer Gesamtentscheidung, welche als eine Art „Güte“ einer Komponente aufgefaßt werden kann. Dieser gesamte Bewertungsprozeß wird abschließend in Abschnitt 4.4 dargestellt.

4.1 Definition des Entwurfsraumes

Das weite Einsatzspektrum eingebetteter Systeme und eine unüberschaubare Anzahl von vorhandenen Komponenten macht eine geschlossene Entwurfsmethodik für alle eingebetteten Systeme nur sehr schwer möglich. Diese Arbeit macht deshalb Einschränkungen und begrenzt die untersuchten eingebetteten Systeme auf eine Teilklasse. Diese Teilklasse repräsentiert jedoch eine weitverbreitete Architekturform, die in der industriellen Automations- und Kommunikationstechnik implementiert wird.

4.1.1 Hardware

Im Mittelpunkt des Architekturentwurfs steht der verwendete Mikrocontroller. Grundsätzlich wird wie in Abbildung 4.1 vom Einsatz eines einzigen Mikrocontrollers ausgegangen. Mehrprozessorsysteme werden somit nicht betrachtet.

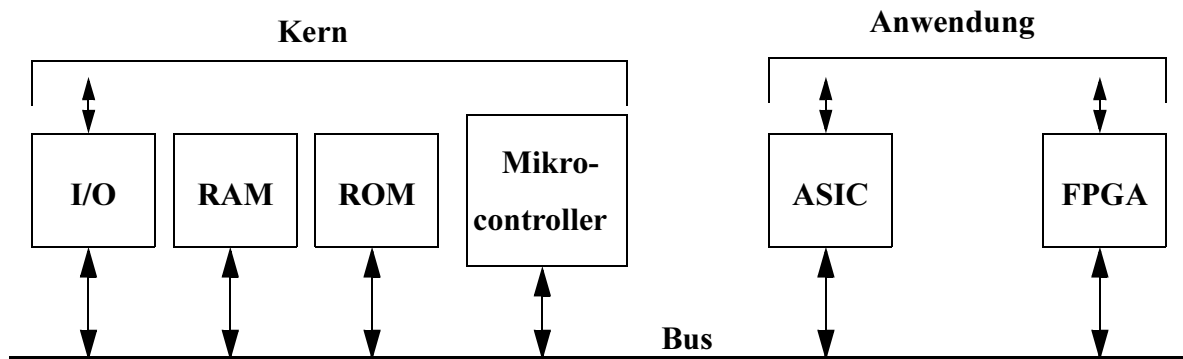


Abbildung 4.1 Architektur der betrachteten Teilklasse

Der Mikrocontroller benötigt für seine Funktion Speicherbausteine, die an seinem Bus angeschlossen werden, um ein ausführbares Programm zu speichern. Programmteile in einem nicht-flüchtigen *ROM* werden dabei nach einem Systemstart zuerst ausgeführt, das *RAM* dient als Arbeitsspeicher. Zusätzlich werden Ein- und Ausgabe-Bausteine implementiert. Häufig werden serielle Schnittstellen eingesetzt, in leistungsstärkeren Systemen findet man auch Ethernet-Schnittstellen. Hauptaufgabe dieser Funktionsgruppe ist die Verbindung zu einem Entwicklungsrechner, über die während der Entwicklungsphase ausführbare Programme in den Arbeitsspeicher geladen werden oder Analysedaten während der Fehlersuche abgerufen werden können. Die Funktionsbaugruppen *I/O*, *RAM*, *ROM* sind in allen Architekturen als Basiskomponenten zu finden und werden im folgenden zusammen mit dem Mikrocontroller als (Mikrocontroller-)Kern bezeichnet. Insbesondere die Schnittstelle des Mikrocontrollers zur Außenwelt, der Mikrocontroller-Bus, gibt wesentliche Randbedingungen bei der Auswahlentscheidung von Komponenten vor und beeinflusst die Architekturfindung wesentlich. Dem logischen und elektrischen Zeitverhalten dieses Busses müssen sich alle anderen Komponenten unterordnen bzw. müssen mit Hilfe von weiteren digitalen Schaltungen angepaßt werden.

Eine Architektur, welche die Anforderungen einer bestimmten Anwendung erfüllt, benötigt die richtige Auswahl von auf dem Markt verfügbaren *ASIC*-Komponenten. Erst wenn eine speziell geforderte Funktionalität nicht verfügbar, die Implementierung nicht effizient durchführbar ist oder die Leistung der Komponente nicht ausreicht, wird ein Eigenentwurf durchge-

führt. Eine geeignete Zieltechnologie zur Emulation eines solchen Eigenentwurfs bilden dabei *FPGA*-Bausteine.

4.1.2 Software

Die in dieser Teilklasse betrachteten Systeme können prinzipiell ohne Betriebssystem arbeiten. Der Programmcode wird in einer höheren Programmiersprache (z.B. C) geschrieben und mit Hilfe eines Compilers, Assemblers und Linkers in einen ausführbaren Code für die Hardware des Zielsystems übersetzt. Um die Entwicklung zu unterstützen, kommen sogenannte Software-Monitore zum Einsatz. Darunter versteht man ein Programmpaket, welches im wesentlichen aus zwei Teilen besteht. Der erste Teil wird auf dem Prozessor eines Entwicklungsrechners ausgeführt und kommuniziert über eine serielle Verbindung mit dem zweiten Teil, welcher auf dem Mikrocontroller des Zielsystems läuft. Dadurch werden Funktionen ermöglicht wie das Laden von ausführbarem Programmcode auf das Zielsystem, einfache Testfunktionen, das Ausführen von Programmcode im Einzelschritt oder das Anzeigen von Speicher- und Registerinhalten. Anwendungsprogramme, die mit Hilfe dieser Werkzeuge entwickelt werden, bestehen aus einem Hauptprogramm und aus einem oder auch mehreren Unterbrechungsprogrammen. Die Benutzeroberfläche eines im Rahmen dieser Arbeit eingesetzten Software-Monitors zeigt Abbildung 4.2.

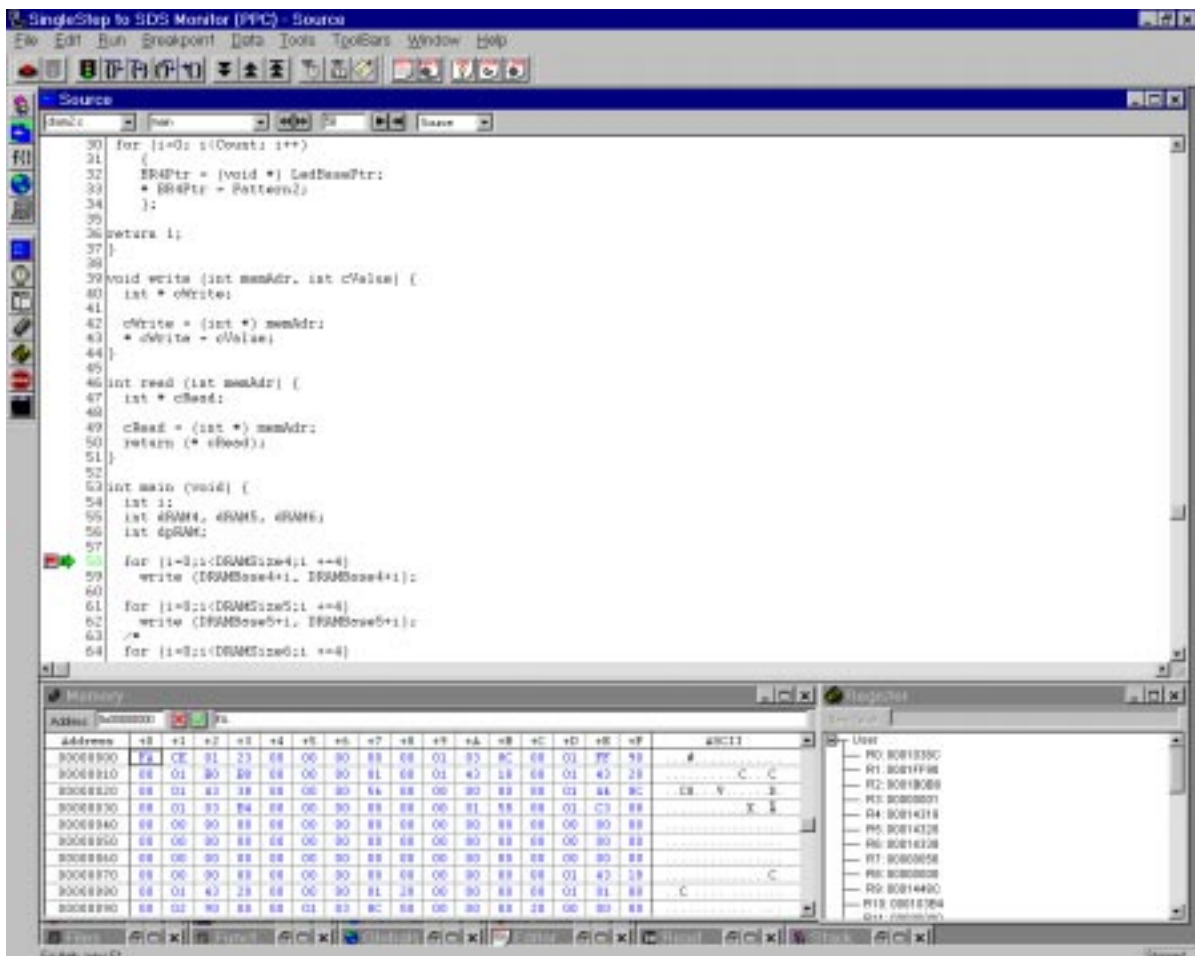


Abbildung 4.2 SDS70 Software-Monitor Single-Step

Komplexe Programmstrukturen, welche zusätzlich unter Echtzeitbedingungen ausgeführt werden müssen, erfordern den Einsatz eines Echtzeitbetriebssystems. Solche Anwendungsprogramme bestehen aus mehreren Programmteilen (*Multi-Tasks*) und mehreren Unterbrechnungsprogrammen (*Interrupt-Tasks*) und gehören ebenfalls zu der Klasse der betrachteten Zielsysteme. Im Rahmen dieser Arbeit wird das Echtzeitbetriebssystem *VxWorks* (siehe Abschnitt 2.2.1) eingesetzt.

4.1.3 Zusammenfassung und Begründung der Randbedingungen

Die Architektur der betrachteten Teilklasse wird wie folgt beschrieben:

- Im Bereich der Hardware bilden die Basis-Komponenten Mikrocontroller, *ROM* und *RAM* den Mikrocontroller-Kern,
- es werden nur Ein-Mikrocontroller-Systeme betrachtet,
- der Mikrocontroller gibt das Bus-Verhalten vor,
- andere Komponenten müssen sich diesem Verhalten anpassen,
- das vorgestellte Auswahlverfahren bezieht sich auf Hardware-Komponenten,
- wobei im Vordergrund der Einsatz von vorhandenen *ASIC*-Komponenten steht,
- diese *ASIC* haben in der Regel eine digitale Bus-Schnittstelle zum Mikrocontroller und analoge oder digitale Schnittstellen zur Umgebung,
- in Sonderfällen findet auch eine spezifikationsbedingte Eigenentwicklung (Zieltechnologien *FPGA* und *CPLD*) statt,
- im Bereich der Software existieren einfache Programmstrukturen mit einem Hauptprogramm ohne Kontextwechsel und verschiedenen *Interrupt*-Programmen
- sowie auch komplexe Programmstrukturen mit mehreren Programmteilen und Kontextwechsel, basierend auf einem Echtzeitbetriebssystem.

Ausgangspunkt für den Architekturentwurf ist der einzusetzende Mikrocontroller. Moderne 32-Bit-Mikrocontroller (siehe Abschnitt 2.1.1.3) ermöglichen die freie Skalierung ihrer Rechenleistung über weite Bereiche, so daß viele verschiedene eingebettete Systeme mit unterschiedlichen Leistungsmerkmalen mit demselben Mikrocontroller effizient implementiert werden können. Mit einem Mikrocontroller sind wesentliche initiale Vorarbeiten und Kosten verbunden, die einen Austausch der einzelnen Typen bei unterschiedlichen Projekten nicht einfach ermöglichen. Die wichtigsten Punkte, die gegen ein beliebiges Wechseln des Mikrocontrollers sprechen, sind:

- Die Beschaffung und Portierung umfangreicher Entwicklungswerkzeuge (Betriebssystem, Compiler, Testwerkzeuge),
- diese Werkzeuge erfordern erhebliche Einarbeitungszeiten und Entwurfserfahrung, wodurch die Wiederverwendung von Wissen unbedingt notwendig wird.

Diese Randbedingungen unterstreichen die zentrale Stellung der Komponente Mikrocontroller und den dadurch begründeten „Mikrocontroller-getriebenen Ansatz“ bei dem Architekturentwurf für eingebettete Systeme.

4.2 Extrahierung des Auswahlalgorithmus

Dieses Teilkapitel basiert auf der Analyse des Vorgehens eines Entwicklers bei der Komponentenauswahl. Die prozedurale Darstellung dieses Prozesses ist Grundlage für die weitere systematische Beschreibung eines Auswahlverfahrens für Hardware-Komponenten. In Abschnitt 3.1 wurde eine umfassende Darstellung der Problemfelder beim Entwurf von eingebetteten Systemen gegeben, welche wesentlich von der Beschreibung des Standes der Technik in Kapitel 2 begründet werden.

Diese Problemfelder müssen bei der Auswahl von Komponenten entsprechend berücksichtigt werden. Dazu werden in dieser Arbeit einzelne Entscheidungsfelder eingeführt, welche die einzelnen Problemfelder berücksichtigen und eine systematische Bewertung einer Komponente erlauben. Dieser Schritt wird in Abbildung 4.3 dargestellt.

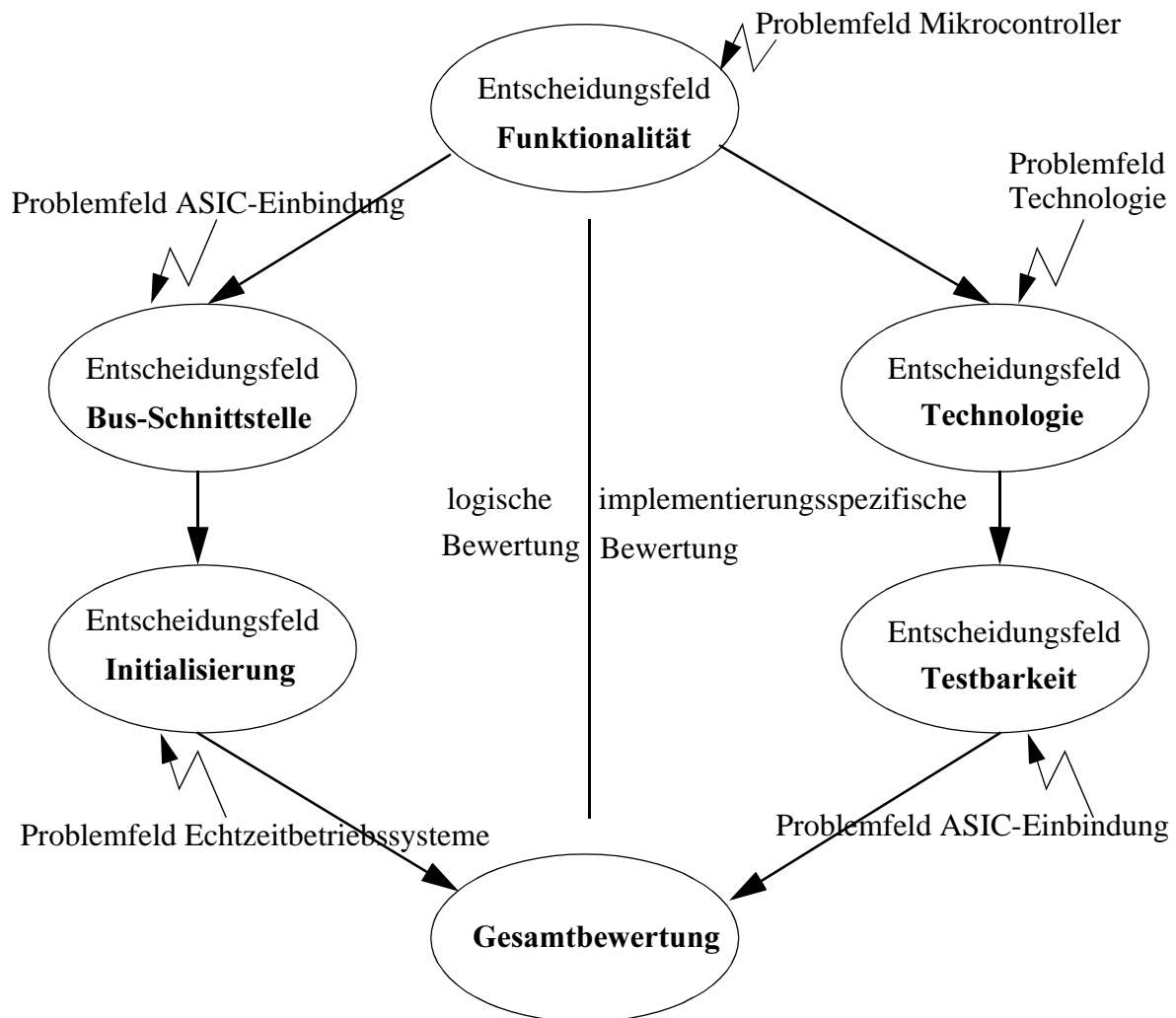


Abbildung 4.3 Entscheidungsgebiete bei der Komponentenauswahl

- **Entscheidungsfeld Funktionalität:** Bei der Hardware-Auswahl ist die wichtigste Frage, welcher Mikrocontroller ausgewählt wird. Er bestimmt die Architektur des eingebetteten Systems in zwei wesentlichen Punkten. Der erste Aspekt ist seine Rechenleistung in Verbindung mit seiner *on-Chip*-Peripherie. Dieser beschreibt, wieviel von der gesamten Systemfunktionalität bereits von ihm selbst abgedeckt werden kann. Die Freiheitsgrade bei der Mikrocontroller-Auswahl sind allerdings relativ begrenzt. Wie bereits in Abschnitt 2.1.1 motiviert, haben sich insbesondere im Bereich der industriellen Automation und Kommunikation wenige in diesem Anwendungsbereich hochspezialisierte Mikrocontroller-Familien auf dem Markt etabliert.

Der zweite Aspekt ist die Bus-Schnittstelle des Mikrocontrollers, welche wesentlich die Auswahl von weiteren Komponenten bestimmt. Diese müssen die restlichen vom Mikrocontroller nicht bereitgestellten Systemfunktionen übernehmen. Deshalb wurde bereits in Abschnitt 3.3 von einem „Mikrocontroller-basierten“ Ansatz gesprochen. Diese Funktionen werden in erster Linie auf *ASIC*-Bausteine abgebildet. Erst wenn keine geeigneten *ASIC*-Bausteine gefunden werden können, wird eine eigene Entwicklung durchgeführt.

Dieser zweite Aspekt hat dabei wesentlich mehr Freiheitsgrade als der erste, weil die Anzahl der am Markt verfügbaren *ASIC*-Bausteine, welche prinzipiell als Architektur-Komponenten zur Auswahl stehen, sehr groß ist. Die Aufgabenstellung der Bewertung von *ASIC*-Komponenten im Rahmen des Architekturentwurfs ist deshalb auch die wichtigste Zielsetzung dieses Kapitels.

Voraussetzung für die Betrachtung einer Komponente im Bewertungsprozeß ist, daß sie eine gesuchte Funktionalität erfüllen kann. Wie man diese Frage beantworten kann, wird in Abschnitt 4.3.1 detaillierter gezeigt. Außer der Frage nach der Funktionalität müssen aber noch weitere Entscheidungsfelder betrachtet werden, welche man, wie in Abbildung 4.3 dargestellt, prinzipiell in die zwei parallelen Pfade einer logischen sowie einer implementierungsspezifischen Bewertung einteilen kann.

- **Logische Bewertung - Entscheidungsfeld Bus-Schnittstelle:** Im logischen Bewertungspfad muß zunächst für eine *ASIC*-Komponente geklärt werden, welchen Aufwand der Anschluß an den Mikrocontroller darstellt, welcher ein bestimmtes Bus-Verhalten vorgibt. Dieses Entscheidungsfeld trägt den logischen Problemen bei der Einbindung eines *ASIC* in die Architektur eines eingebetteten Systems Rechnung. Motiviert wird dieses Entscheidungsfeld durch die Ausführungen in Abschnitt 2.1.4 sowie in Abschnitt 3.1.4. Die zu der Bewertung dieses Entscheidungsfeldes notwendige Vorgehensweise wird in Abschnitt 4.2.2 dargestellt. Die Bewertung dieses Entscheidungsfeldes, d.h die prinzipielle technische Möglichkeit des Anschlusses an den Mikrocontroller hat Priorität vor dem nachfolgenden Entscheidungsfeld Initialisierung, da die Kommunikation über die Bus-Schnittstelle die Grundvoraussetzung für einen Einsatz der Komponente darstellt.
- **Logische Bewertung - Entscheidungsfeld Initialisierung:** Das Entscheidungsfeld Initialisierung faßt alle Fragen zusammen, die mit der richtigen Inbetriebnahme der Komponente verbunden sind. Im Falle des Einsatzes eines Echtzeitbetriebssystems muß eine Firmware (oder auch Treiber) entwickelt und zusammen mit dem Betriebssystem-Kern implementiert werden, welche die Komponente für die Anwendungs-Software in einer von Initialisierungsdetails befreiten Art und Weise zur Verfügung stellt. Die Bewertung dieses Entscheidungsfeldes wird in Abschnitt 4.2.3 dargestellt.

- **Implementierungsspezifische Bewertung - Entscheidungsfeld Technologie:** Das Teilgebiet Technologie faßt alle Fragen zusammen, die mit dem Einsatz der Komponente in einer bestimmten Umgebung verbunden sind. Insbesondere werden die Anforderungen der Komponente hinsichtlich der Auslegung der Leiterplatte berücksichtigt. Zur Verdeutlichung wird auf die Grundlagen in Abschnitt 2.1.5 sowie die Beschreibung der Entwurfsproblematik in Abschnitt 3.1.4 verwiesen. In Abschnitt 4.2.4 wird gezeigt, wie die Bewertung dieses Entscheidungsfeldes durchgeführt werden kann. Dieses Entscheidungsfeld hat im implementierungsspezifischen Pfad Priorität gegenüber dem nachfolgenden Entscheidungsfeld der Testbarkeit, weil die Beherrschung der technologischen Anforderungen einer Komponente eine Grundvoraussetzung für deren Einsatz ist.
- **Implementierungsspezifische Bewertung - Entscheidungsfeld Testbarkeit:** Das Teilgebiet Testbarkeit befaßt sich mit den notwendigen Einrichtungen auf dem Chip oder der Leiterplatte, um die korrekte Funktion einer Komponente zu überprüfen bzw. die richtige Zusammenarbeit mit anderen Komponenten des Systems sicherzustellen. Die Motivation für dieses eigenständige Entscheidungsfeld wurde bei der Beschreibung der Probleme der *ASIC*-Einbindung in Abschnitt 3.1.4 gegeben. Im Abschnitt 4.2.5 wird die für die Bewertung notwendige Vorgehensweise dargestellt.

Beide Pfade führen in einen Gesamtbewertungsprozeß, in dem aus einer Menge von potentiell in Frage kommenden Komponenten diejenige Komponente mit der besten Eignung für das eingebettete System ausgewählt wird. Dieser abschließende Prozeß wird in Abschnitt 4.4 vorgestellt.

4.2.1 Entscheidungsfeld Funktionalität

Eine wesentliche Aufgabe beim Architekturentwurf für eingebettete Systeme in dem beschriebenen Anwendungsbereich ist die Erkennung von bestimmten Teilfunktionen, welche auf dedizierte Hardware-Komponenten abgebildet werden können. Abbildung 4.4 stellt dafür die Vorgehensweise in Form eines Ablaufdiagrammes dar.

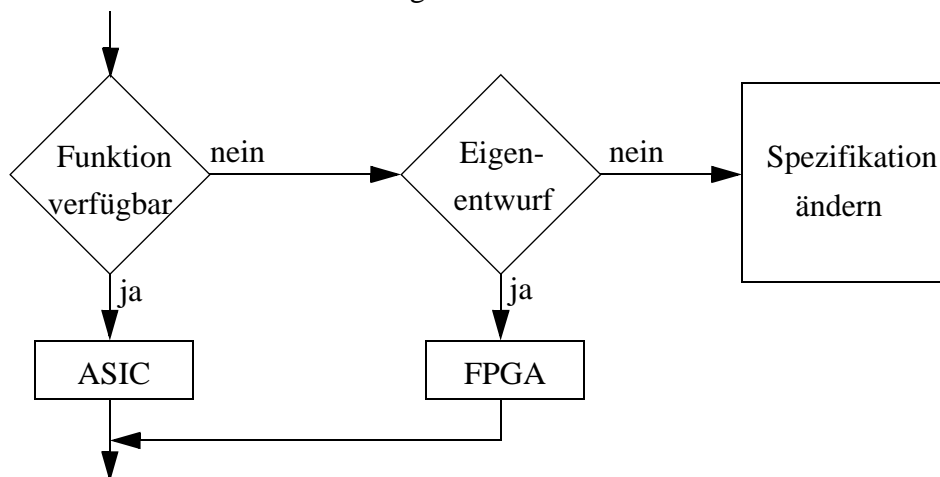


Abbildung 4.4 Auswahlalgorithmus für das Entscheidungsfeld Funktionalität

Dabei wird zuerst versucht, die gewünschte Funktion durch den Einsatz einer vorhandenen *ASIC*-Komponente zu erfüllen. Dieser Schritt führt zur Einführung eines *ASIC* in die Architektur des eingebetteten Systems. Erst wenn keine Komponente gefunden werden kann, die den

Anforderungen entspricht, muß geprüft werden, ob eine eigene Entwicklung durchgeführt werden kann. Als geeignete Zieltechnologien sowohl für die Entwicklung als auch für die Serienfertigung wird hier der Einsatz von *FPGA*-Bausteinen betrachtet. Ist auch der Eigenentwurf unter den vorgegebenen Randbedingungen nicht möglich, ist der Entwurf nicht durchführbar. Es muß frühzeitig eine entsprechende Änderung der Spezifikation durchgeführt werden. Im Auswahlalgorithmus in Abbildung 4.4 sind die beiden Entscheidungsparameter „Funktion verfügbar“ und „Eigenentwurf“ enthalten. In Abschnitt 4.3.1 wird dargestellt, wie diese Entscheidungsparameter bewertet werden.

4.2.2 Entscheidungsfeld Bus-Schnittstelle

Beim Anschluß der Komponente muß das logische Anschlußverhalten an das Verhalten der Bus-Schnittstelle des Mikrocontrollers angepaßt werden. Abbildung 4.5 zeigt die Vorgehensweise bei diesem Entscheidungsfeld.

Wenn dieses Verhalten nicht eingehalten werden kann, müssen zusätzliche digitale Schaltungen zwischen Komponente und Mikrocontroller eingefügt werden. Diese Koppellogik muß in einem weiteren Schritt, nämlich der Prüfung des zeitlichen Verhaltens, berücksichtigt werden. Durchlaufzeiten durch die Koppellogik addieren sich zu der Zugriffszeit und verlangsamen dadurch die Zugriffsgeschwindigkeit der Bus-Schnittstelle einer Komponente.

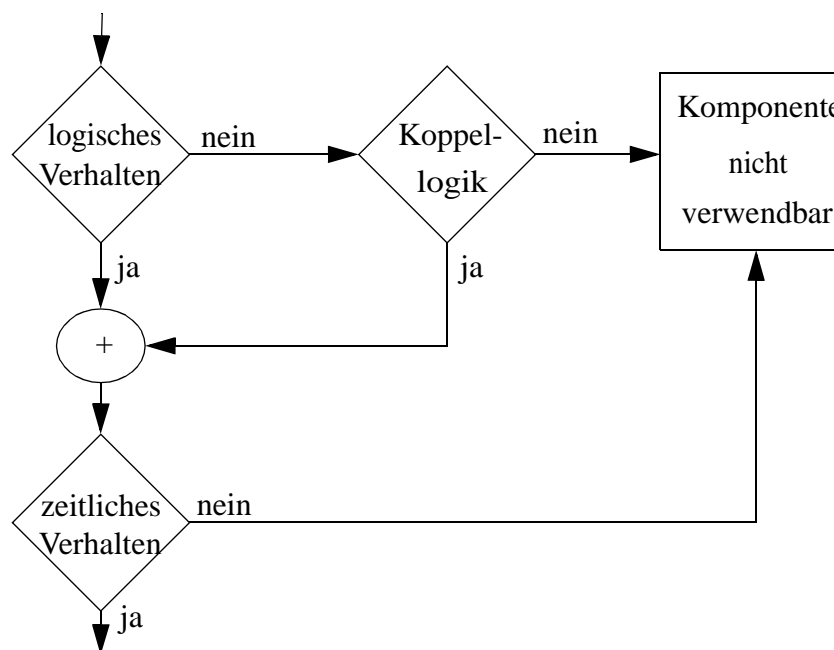


Abbildung 4.5 Auswahlalgorithmus für das Entscheidungsfeld Bus-Schnittstelle

Wenn der technische Aufwand zu groß wird, um das logische Verhalten anzupassen, oder wenn das elektrische Verhalten aufgrund von bestimmten Zeitbedingungen nicht eingehalten werden kann, muß diese Komponente als nicht verwendbar bei der weiteren Bewertung ausgeschlossen werden. In diesem Entscheidungsfeld sind insgesamt die drei Entscheidungsparameter „logisches Verhalten“, „Koppellogik“ und „zeitliches Verhalten“ zu bewerten, welche in Abschnitt 4.3.2 detailliert eingeführt werden.

4.2.3 Entscheidungsfeld Initialisierung

Dieses Entscheidungsfeld befaßt sich mit der Aufgabe, eine Komponente in einen von mehreren möglichen Betriebszuständen zu versetzen und diesen Zustand während der Laufzeit auch zu kontrollieren. Dazu muß der Mikrocontroller bestimmte Steuerregister in der Komponente entsprechend beschreiben und Statusregister lesen und richtig interpretieren. Bestimmte Komponenten benötigen auch den Aufbau und die Kontrolle von Datenstrukturen in einem gemeinsamen Speicherbereich. Dieses Entscheidungsfeld bewertet im wesentlichen die sehr komponentenabhängigen Programme, die im allgemeinen als Firmware oder auch Treiber bezeichnet werden.

Die Vorgehensweise bei diesem Entscheidungsfeld ist in Abbildung 4.6 dargestellt. Ein wesentlicher Entscheidungsfaktor für den Einsatz einer Komponente ist, wenn bereits ein Treiber entwickelt und getestet wurde.

Liegt kein Treiber vor, muß der Hersteller eine detaillierte Beschreibung in Form einer Vorlage zur Verfügung stellen. Kann auch dieses nicht gewährleistet werden, ist der Einsatz einer solchen Komponente sehr riskant.

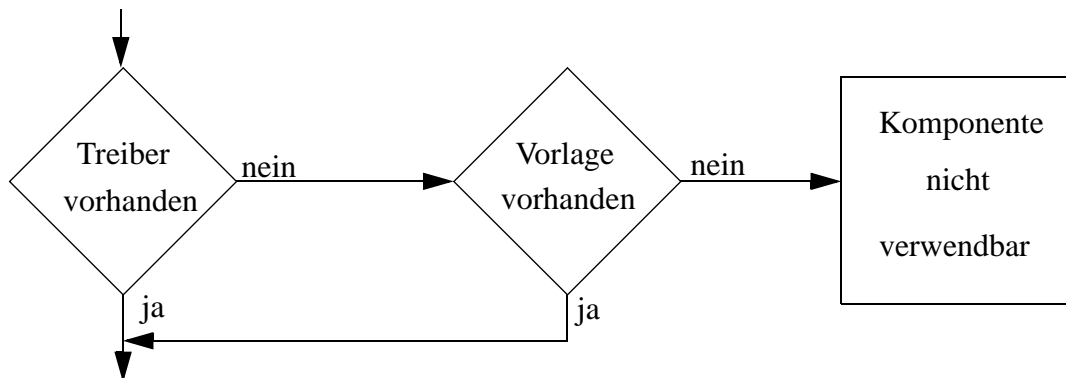


Abbildung 4.6 Auswahlalgorithmus für das Entscheidungsfeld Initialisierung

Für einen Systementwickler, der nicht den genauen internen Aufbau und die Funktionsweise einer Komponente kennt, ist es sehr schwierig, basierend auf einer reinen Registerbelegung diese Treiberfunktionen zu entwickeln. In einem solchen Fall ist die Komponente unter realistischen Bedingungen nicht verwendbar und muß von der weiteren Bewertung ausgeschlossen werden. In diesem Entscheidungsfeld sind die beiden Entscheidungsparameter „Treiber vorhanden“ und „Vorlage vorhanden“ zu bewerten, welche in Abschnitt 4.4 beschrieben werden.

4.2.4 Entscheidungsfeld Technologie

Bei diesem Entscheidungsfeld werden alle Eigenschaften betrachtet, die den Einsatz der Komponente in der Zielumgebung und auf der Leiterplatte betreffen. Abbildung 4.7 zeigt den Auswahlalgorithmus für dieses Entscheidungsfeld.

Eine Komponente steht in einem bestimmten Gehäuse zur Verfügung. Dieses besitzt Eigenschaften, die in Abhängigkeit von der Spezifikation bewertet werden müssen. Kann eine Gehäuseform die Anforderungen nicht erfüllen, muß geprüft werden, ob diese Komponente in einem alternativen Gehäuse zur Verfügung steht. Kann keine Gehäuseform gefunden werden, die den Randbedingungen der Spezifikation genügt, kann die Komponente nicht eingesetzt wer-

den. In diesem Entscheidungsfeld sind die beiden Entscheidungsparameter „Gehäuseform und Pins“ sowie „Alternativ-Gehäuse“ zu bewerten, welche in Abschnitt 4.3.4 beschrieben werden.

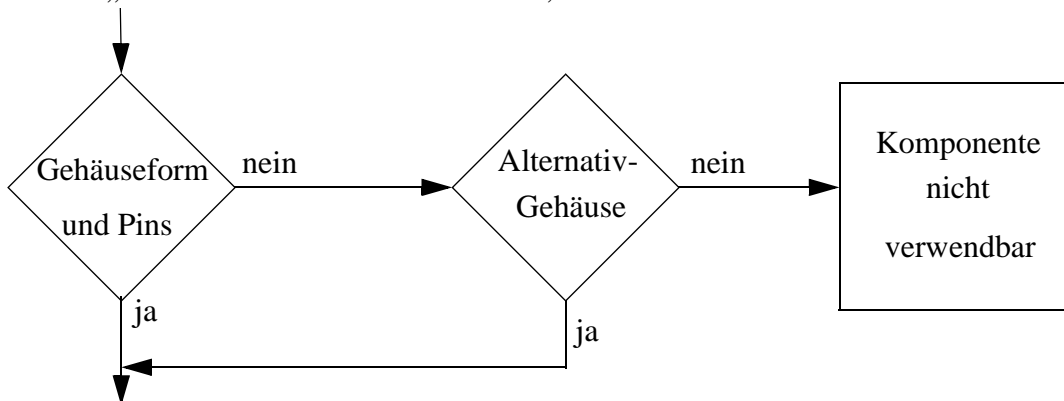


Abbildung 4.7 Auswahlalgorithmus für das Entscheidungsfeld Technologie

4.2.5 Entscheidungsfeld Testbarkeit

Dieses Entscheidungsfeld beschreibt alle Maßnahmen, die der Überprüfung der wichtigsten Grundfunktionen einer Komponente dienen. Wie in Abbildung 4.8 dargestellt, werden Einrichtungen benötigt, die nach der Integration auf einer Leiterplatte die systematische Inbetriebnahme einer Komponente unterstützen. Sind solche Einrichtungen nicht auf dem Chip selbst vorgesehen, müssen diese auf der Leiterplatte zusätzlich implementiert werden.

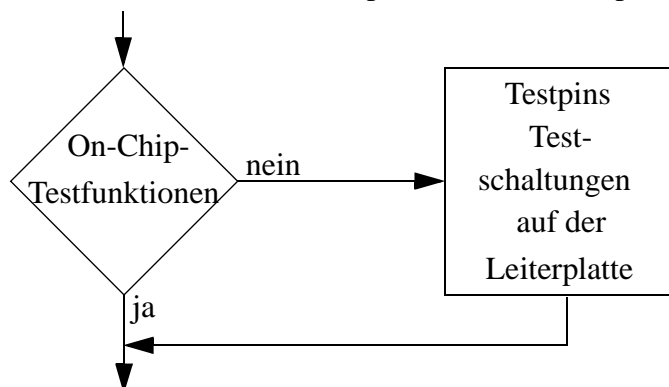


Abbildung 4.8 Auswahlalgorithmus für das Entscheidungsfeld Testbarkeit

Wichtig dabei ist, daß die Möglichkeit besteht, die Komponente bei einer Fehlfunktion vom restlichen System zu isolieren und gegebenenfalls durch eine andere Komponente zu ersetzen. Dadurch wird erreicht, daß das restliche Systemverhalten ohne die Komponente geprüft werden kann. Der Ausfall einer Komponente darf nicht das Scheitern des gesamten Entwicklungsprojektes eines eingebetteten Systems zur Folge haben. Bei diesem Entscheidungsfeld ist der Entscheidungsparameter „on-Chip-Testfunktionen“ zu bewerten, welcher in Abschnitt 4.3.5 eingeführt wird.

4.3 Identifikation der Entscheidungsparameter

Der im Teilkapitel 4.2 abgeleitete Auswahlalgorithmus hat verschiedene Pfade, die abhängig von entsprechenden Entscheidungsparametern durchlaufen werden müssen. Dieser Ab-

schnitt beschreibt für die einzelnen Entscheidungsfelder die entsprechenden Parameter und zeigt die Bewertung innerhalb des Entscheidungsfeldes.

4.3.1 Entscheidungsparameter für die Funktionalität

In Abbildung 4.4 muß zuerst der Entscheidungsparameter „Funktion verfügbar“ beantwortet werden. Bei der Erarbeitung einer Methode zur Festlegung dieses Parameters wird als Ausgangspunkt ebenfalls die Vorgehensweise eines Entwicklers analysiert. Diese Analyse führt zur Bildung eines Modells für den Entwurfsraum in der beschriebenen Teilklasse.

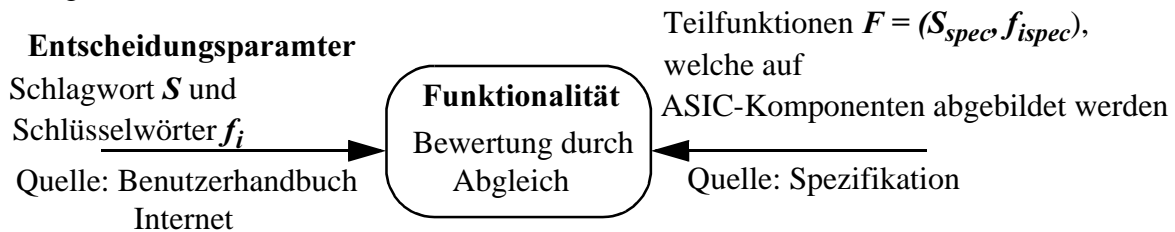


Abbildung 4.9 Bewertung der Funktionalität

Die Vorgehensweise eines Entwicklers ist im wesentlichen in zwei Schritte gegliedert. Der Ausgangspunkt bei dieser Bewertung ist in Abbildung 4.9 dargestellt. Im ersten Schritt bestimmt er zunächst anhand der Spezifikation einzelne Teilfunktionen (F), welche in der Praxis mit einem funktionsbeschreibenden Schlagwort (S_{spec} , z.B. *ATM*) und einem Satz von Schlüsselwörtern (f_{ispec} , z.B. *ALL5*) beschrieben werden können. Dieses Schlagwort bezeichnet die wesentlichste funktionale Eigenschaft, die für die Realisierung einer Teilfunktion unbedingt gefordert wird und dient als Eingabe für eine erste Vergleichsphase. In dieser Phase wird innerhalb jeder Hauptgruppe versucht, eine *ASIC*-Komponentengruppe zu finden, die diese funktionale Haupteigenschaft (S) ebenfalls besitzt. Der Vergleich des Schlagwortes aus der Spezifikation (S_{spec}) mit allen vorhandenen Schlagwörtern (S) innerhalb der verschiedenen *ASIC*-Hauptgruppen ergibt entweder die Übereinstimmung mit einer einzigen Untergruppe oder mit keiner Untergruppe. Im letzteren Fall wird die Funktion schon hier als „nicht verfügbar“ erkannt.

Wenn eine geeignete *ASIC*-Komponenten-Untergruppe in diesem Auswahlprozeß gefunden wurde, schließt sich ein zweiter, verfeinerter Vergleichsprozess an. Dazu müssen aus der Spezifikation für eine zu implementierende Teilfunktion Schlüsselwörter (f_{ispec}) bestimmt werden und diese mit den funktionsbeschreibenden Schlüsselwörtern (f_i) der Komponenten innerhalb der gefundenen Komponenten-Untergruppe verglichen werden. Diese Schlüsselwörter haben innerhalb eines bestimmten funktionalen Teilgebietes, welches durch das Schlagwort (z.B. *ATM*) bestimmt ist, eine wohldefinierte Bedeutung. Existieren *ASIC*-Komponenten, sogenannte Alternativen (A), die alle geforderten Schlüsselwörter in ihrem funktionalen Verhalten erfüllen, kann der Entscheidungsparameter „Funktion verfügbar“ mit <ja> beantwortet werden.

Dazu wird zunächst ein Modell der Menge aller *ASIC* eingeführt, welches in Abbildung 4.10 dargestellt ist. Anschließend wird dieses benutzt, um daran die oben beschriebene Vorgehensweise zu demonstrieren.

Die Menge aller verfügbaren *ASIC* kann zunächst eingeteilt werden in n verschiedene Hauptgruppen (z.B. Kommunikation-*ASIC*, Graphik-*ASIC*, Arithmetik-*ASIC* usw.) Jede Hauptgruppe besteht aus unterschiedlich vielen Untergruppen (bezeichnet mit der Laufvariabel m und t). Sie werden mit einem funktionsbeschreibenden Schlagwort (S) bezeichnet und repräsentieren verschiedene Einsatzspektren von *ASIC* innerhalb der Hauptgruppe, z.B. bei den Kommu-

nikations-ASIC-Chips für ATM, Ethernet, PROFIBUS, ASI usw.. Der Name der Untergruppe selbst beschreibt dabei schlagwortartig (S) das betreffende Einsatzspektrum.

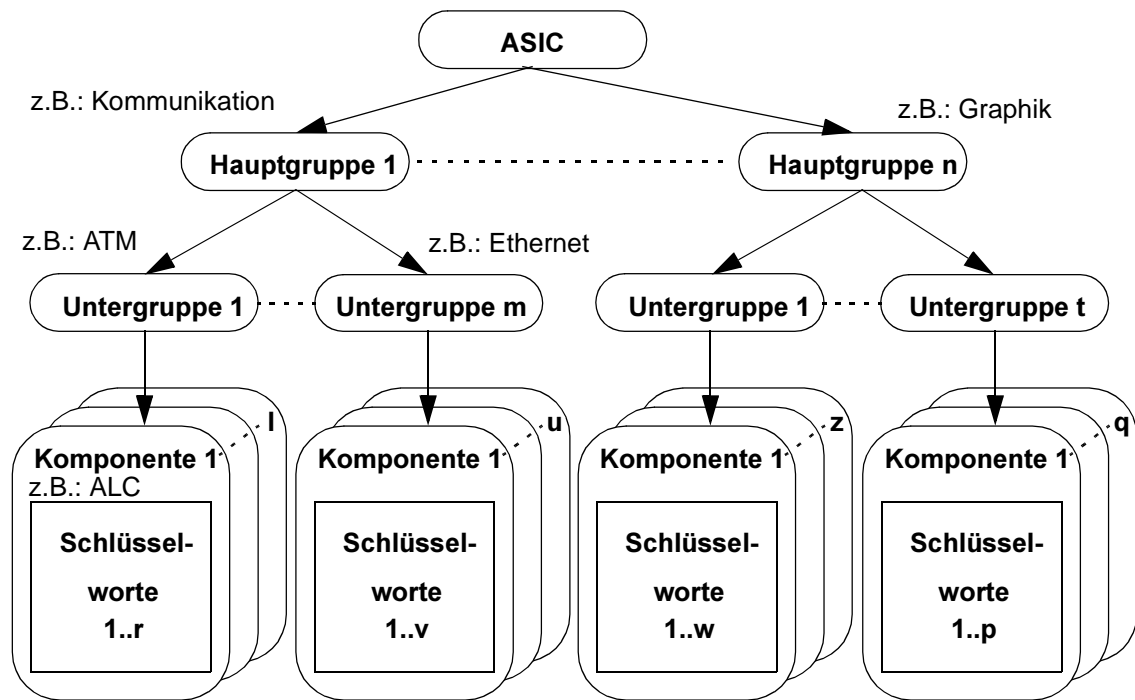


Abbildung 4.10 Allgemeines Modell der Menge verfügbarer ASIC

Jede Untergruppe beinhaltet unterschiedlich viele ASIC-Komponenten (bezeichnet mit l, u, z, q), welche in diesem Einsatzspektrum zur Verfügung stehen, z.B. bei *ATM* verschiedene „Adaption Layer Controller - ALC“ - oder auch „physical Layer - PHL“ - Chips. Diese ASIC-Komponenten werden in den dazugehörigen Benutzerhandbüchern mit einem Satz von Schlüsselwörtern f_i beschrieben, die die einzelnen funktionalen Eigenschaften der Komponenten beschreiben und in diesem Einsatzspektrum eine wohldefinierte Bedeutung haben (z.B.: das Schlüsselwort f_1 = „segmentation and reassembly“ für das logisch richtige Zerteilen von Nutzdaten in entsprechende ATM-Zellen vor der Übertragung). Jede Komponente kann dabei eine unterschiedliche Anzahl (bezeichnet mit r, v, w, p) von Schlüsselwörtern f_i besitzen.

Für die Beantwortung des Entscheidungsparameters „Funktion verfügbar“ muß also die entsprechende Hauptgruppe durchsucht werden, ob zunächst eine Untergruppe existiert, für die gilt $S = S_{spec}$

Wenn diese Forderung zutrifft, muß weiter geprüft werden, ob für alle $S_{spec}(f_{ispec}) = S(f_i)$ gilt. Jede durch diesen Vergleich gefundene ASIC-Komponente wird in die Menge der Implementierungsalternativen (A) aufgenommen. Für jede Komponente aus der Alternativ-Menge (A) schließt sich der Bewertungsprozeß an, welcher in Abbildung 4.3 dargestellt ist.

Bleibt diese Menge leer, ist die gesuchte Teilfunktion $F = (S_{spec}, f_{ispec})$ als ASIC-Komponente nicht verfügbar, der Entscheidungsparameter „Funktion verfügbar“ muß mit <nein> beantwortet werden.

Wurden keine Implementierungsalternativen unter denen am Markt verfügbaren ASIC-Komponenten gefunden, muß der Aufwand einer eigenen Hardware-Entwicklung abgeschätzt werden. Entscheidungsparameter sind bei diesem Teilschritt die Effizienz der Entwicklungs-

werkzeuge, die Entwicklungszeit und die Kosten der entsprechenden Zieltechnologie. Dabei eignen sich für die Emulation und Produktion in kleineren Stückzahlen *FPGA*. In Abschnitt 2.1.3 wurden die wichtigsten Architekturen vorgestellt. Im Rahmen dieser Arbeit wurden speziell zur Unterstützung dieses Schrittes Werkzeuge entwickelt, die in Kapitel 5 beschrieben und in Kapitel 6 demonstriert werden.

4.3.2 Entscheidungsparameter für die Bus-Schnittstelle

Bei der Untersuchung des Anschlusses einer *ASIC*-Komponente an die Bus-Schnittstelle eines Mikrocontrollers müssen die Bus-Eigenschaften der selektierten *ASIC*-Komponente mit den Bus-Eigenschaften des Mikrocontrollers abgeglichen werden, wie in Abbildung 4.11 schematisch dargestellt.

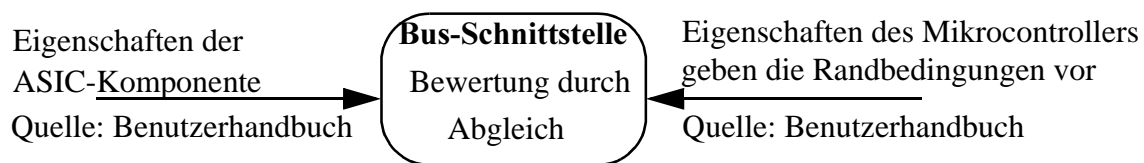


Abbildung 4.11 Bewertung der Bus-Schnittstelle durch Abgleich

Bei diesem Bewertungsprozeß gibt der Mikrocontroller die Randbedingungen vor. Moderne 32-Bit-Mikrocontroller (siehe Abschnitt 2.1.1.3) sind in der Lage, sich an viele unterschiedliche logische und elektrische Bus-Parameter einer externen *ASIC*-Komponente anzupassen. Die Bewertung dieses Anpassungsprozesses ist allerdings ein sehr schwieriger und fehleranfälliger Prozeß und hängt im wesentlichen von den Detail-Kenntnissen des Entwicklers bezüglich der Hardware-Eigenschaften von Bus-Schnittstellen ab. Der in dieser Arbeit verfolgte Ansatz basiert auf einer Art „Check-Liste“, welche in Tabelle 4.1 angegeben ist. Das Ziel dabei ist, den Entwickler in einer systematischen Art und Weise durch diesen Themenkomplex zu führen, wobei für die einzelnen Anpassungsfälle eine Bewertung des notwendigen Aufwandes durchgeführt wird.

In der linken Spalte wurde der Entscheidungsparameter „logisches Verhalten“ aufgelistet. Diese Liste kann sicherlich nicht jeden Spezialfall abdecken, erhebt aber durchaus den Anspruch, für *ASIC*-Komponenten im Anwendungsbereich der industriellen Automation und Kommunikation eine breite Abdeckung verschiedener Bus-Eigenschaften zu berücksichtigen.

Der beste Anpassungsfall liegt dann vor, wenn die Flexibilität der Bus-Schnittstelle des Mikrocontrollers ausreicht, sich ohne weitere elektronische Koppellogik an die Bus-Schnittstelle der *ASIC*-Komponente anzupassen. Wenn diese ideale Anpassung (bezeichnet mit I) nicht möglich ist, wird zusätzliche Koppellogik gebraucht, welche auch noch das zeitliche Verhalten beeinflusst. Diese Tatsache wird in der zweiten Spalte beim Entscheidungsparameter „Koppellogik“ berücksichtigt. Hier werden die Maßnahmen aufgelistet, die notwendig werden, wenn eine bestimmte Bus-Eigenschaft der anzupassenden *ASIC*-Komponente in der linken Spalte vom Mikrocontroller **nicht** direkt erfüllt werden kann.

In der dritten Spalte des Entscheidungsparameters „zeitliches Verhalten“ wird der zusätzliche Einfluß der Koppellogik bei der Anpassung beschrieben.

Entscheidungsparameter „logisches Verhalten“	Entscheidungsparameter „Koppellogik“	Entscheidungsparameter „zeitliches Verhalten“	Bewertung
serieller Bus	serial/parallel-Wandlung	kein Einfluß	M
paralleler Standardbus (PCI, VME, SBUS usw.)	Koppel-Chips mit vielen Pins (<i>Bridge-Chips</i>)	kein Einfluß	G
paralleler synchroner Bus	nicht realisierbar	nicht realisierbar	NV
paralleler asynchroner Bus (gemultiplext/nicht gemultiplext)	Adress-Latches für die gesamte Breite des Adressbusses	Zugriffszeit auf die Komponente wird um die Laufzeit durch die <i>Latches</i> verzögert	M
Wortbreite/Ordnung der Daten-Bytes (<i>Big/Little Endian-Mode</i>)	Multiplexer für die gesamte Breite des Datenbusses	Durchlaufzeit durch Multiplexer erhöht <i>setup-Zeit</i> für die Daten	M
Semantik der Steuersignale (Intel/Motorola-Mode, usw.)	Anpassungs-Logik für jedes nicht passende Steuersignal	Zugriffszeit auf die Komponente wird um die Laufzeit durch die Logik-Gatter verzögert	K
Kommunikationsmechanismus <i>Direct Memory Access (DMA)</i>	Koppel-Chip mit vielen Pins	kein Einfluß	G
Kommunikationsmechanismus <i>ASIC-Komponente als Bus-Master</i>	nicht realisierbar	-	NV
Kommunikationsmechanismus <i>Ready/Wait</i>	nicht realisierbar	-	NV

Tabelle 4.1: Bewertung von Entscheidungsparametern für die Bus-Schnittstelle

In der rechten Spalte wird eine Bewertung der Maßnahmen zur Ankoppelung gegeben. Dabei haben die Abkürzungen folgende Bedeutung:

- **Nicht verwendbar (NV):** In diesem Fall existiert keine technische Möglichkeit, die beiden Busverhalten der *ASIC*-Komponente und des Mikrocontrollers miteinander in Einklang zu bringen. In diesem Fall scheidet die Komponente aus dem Auswahlprozeß aus, selbst wenn alle anderen Entscheidungsfelder einen Einsatz zulassen würden. Die Komponente ist nicht anpassbar.
- **Großer Aufwand (G):** Diese Anpassung benötigt die Einführung eines weiteren *ASIC* zwischen der anzupassenden Komponente und dem Mikrocontroller. Solche Chips werden im allgemeinen als *Bridge-Chips* bezeichnet (z.B. *PCI-Bridge*, *VME-Bridge*, externer *DMA*-Controller) und setzen das Busverhalten der einen Seite vollständig in das der anderen Seite um, wobei sie in der Regel die beiden Busse vollständig entkoppeln. Der Aufwand für diese Lösung wird mit „groß“ bewertet, da sie zusätzlich einen signifikanten Flächen- und Energieaufwand haben. Sie müssen praktisch wie eine zusätzlich zu implementierende Teilfunktion den gesamten Bewertungsprozeß durchlaufen.

- **Mittlerer Aufwand (M):** Anpassungen mit mittlerem Aufwand sind Lösungen, bei denen komplexere Zustandsmaschinen (z.B. bei seriell-parallel-Wandlungen) bzw. viele kombinatorische Gleichungen implementiert werden müssen. Als Zieltechnologien kommen *CPLD* oder *FPGA* mit bis zu 1 K-Gattern zum Einsatz.
- **Kleiner Aufwand (K):** Anpassungen mit kleinem Aufwand sind Schaltungen mit einfachen speichernden bzw. kombinatorischen Gattern zur Adaption von Steuersignalen. Zur Implementierung werden kleine *CPLD* mit wenigen *I/O*-Pins benutzt. Dieser Fall kommt in der Regel in eingebetteten Systemen relativ oft vor.
- **Idealer Fall (I):** Dieser Fall liegt vor, wenn der Mikrocontroller ohne zusätzlichen Aufwand alle Eigenschaften der *ASIC*-Komponente direkt bedienen kann.

4.3.3 Entscheidungsparameter für die Initialisierung

Komplexe *ASIC*-Komponenten benötigen eine aufwendige Initialisierungs-Firmware (oder auch Treiber genannt), welche für die korrekte Funktion des Bausteines verantwortlich ist. Mit steigender *on-Chip*-Funktionalität des *ASIC* steigen auch die Anforderungen an diese Firmware, wie bereits in Abschnitt 2.1.4 und Abschnitt 3.1.4 ausführlich motiviert wurde.

In Abbildung 4.12 sind auf der linken Seite die beiden Entscheidungsparameter „Treiber vorhanden“ bzw. „Vorlage vorhanden“ als Eingangsgrößen in diesem Bewertungsprozeß dargestellt. Auf der rechten Seite steht der Mikrocontroller mit seinen Entwicklungswerkzeugen, der auch in diesem Entscheidungsfeld die Randbedingungen für die Entwicklung vorgibt.

In Tabelle 4.2 werden verschiedene Abstufungen der beiden Entscheidungsparameter gegeben. Wie auch in Abbildung 4.6 dargestellt, hat dabei der Entscheidungsparameter „Treiber vorhanden“ eine gewisse Priorität gegenüber dem Entscheidungsparameter „Vorlage vorhanden“, weil in diesem Fall ein bereits entwickelter Treiber-Code zur Verfügung steht.



Abbildung 4.12 Entscheidungsparameter bei der Initialisierung

In der linken Spalte werden zunächst verschiedene Abstufungen des Entscheidungsparameters „Treiber vorhanden“ angegeben. Wenn dieser Entscheidungsparameter erschöpft ist, wird in der mittleren Spalte der Entscheidungsparameter „Vorlage vorhanden“ mit seinen Abstufungen weiter bewertet. Auch in diesem Entscheidungsfeld kann eventuell nicht jeder Sonderfall berücksichtigt werden, es erhebt aber durchaus den Anspruch, ein breites in der Praxis vorkommendes Szenario im Entwicklungsbereich der industriellen Automation und Kommunikation abzudecken.

Entscheidungsparameter „Treiber vorhanden“	Entscheidungsparameter „Vorlage vorhanden“	Bewertung
Treiber für ausgewählten Mikrocontroller vorhanden, gewünschte Betriebsart wird unterstützt und wurde auf einem Zielsystem bereits getestet	---	I
Treiber für ausgewählten Mikrocontroller vorhanden, gewünschte Betriebsart wurde noch nicht implementiert bzw. getestet	----	K
Treiber vorhanden, allerdings nicht für den ausgewählten Mikrocontroller, gewünschte Betriebsart wurde bereits getestet	---	M
Treiber vorhanden, allerdings nicht für den ausgewählten Mikrocontroller, gewünschte Betriebsart wurde noch nicht getestet	---	M
Kein Treiber vorhanden	Vorlage vorhanden, Beschreibung eindeutig und gut dokumentiert, Unterstützung des Herstellers gegeben	M
Kein Treiber vorhanden	Vorlage vorhanden, Beschreibung ohne ausreichende Kommentierung, keine Herstellerunterstützung	G
Kein Treiber vorhanden	Vorlage nicht vorhanden bzw. mangelhaft dokumentiert, keine Herstellerunterstützung	NV

Tabelle 4.2: Bewertung der Entscheidungsparameter für die Initialisierung

In der dritten Spalte wird eine Bewertung angegeben, die in diesem Entscheidungsfeld folgende Interpretation voraussetzt:

- **Idealer Fall (I):** Dieser Anwendungsfall liegt vor, wenn die Hardware und die Firmware ohne Änderung bereits aus früheren Projekten des Entwicklers selbst oder im Rahmen von Pilotprojekten des Herstellers bereits zur Verfügung steht. Hier wird praktisch bereits erworbenes Wissen und Erfahrung wiederverwendet (engl.: *design re-use*).
- **Kleiner Aufwand (K):** Dieser Fall klassifiziert ein Szenario, bei dem ein Treiber bereits im Quell-Code zur Verfügung steht, dieser sich aber in bestimmten Umgebungsbedingungen der Hardware oder der Firmware von den Anforderungen des aktuellen Entwicklungsprojekts unterscheidet.
- **Mittlerer Aufwand (M):** Dieser Fall beschreibt ein Szenario, bei dem ein ähnlicher Anwendungsfall bereits entwickelt und dokumentiert wurde, sich aber von den Anforderungen des aktuellen Entwicklungsprojekts im Bereich der Hardware und der Firmware signifikant unterscheidet. Die Komponente wurde in dem zu untersuchenden

Anwendungsfall noch nicht eingesetzt, wodurch insbesondere das Entwicklungsrisiko ansteigt und bereits bei der Bewertung berücksichtigt wird.

- **Großer Aufwand (G):** Bei diesem Fall wurden noch keine Prototypen entwickelt und dokumentiert. Daraus folgt eine relativ begrenzte Unterstützung für den Entwickler. Die geringe Erfahrung beim Einsatz dieser Komponente bewirkt, daß das Entwicklungsrisiko signifikant ansteigt und zum dominierenden Faktor bei der Bewertung wird.
- **Nicht verwendbar (NV):** Dieser Fall liegt dann vor, wenn für die Komponente keinerlei Unterstützung, weder in textlicher noch in mündlicher Form, durch den Hersteller angeboten wird. Das Entwicklungsrisiko ist in diesem Fall unkalkulierbar. Die Komponente wird vom weiteren Bewertungsprozeß ausgeschlossen.

4.3.4 Entscheidungsparameter für die Technologie

In Abschnitt 3.1.4 wurde bereits motiviert, daß bei steigender Komplexität der *ASIC*-Chips auch deren Verwendung auf Systemebene zu einem signifikanten Entwurfsproblem anwächst. In Abschnitt 2.1.5 wurde als Grundlage der Stand der Technik im Bereich der Höchstintegration dargestellt.

Bei der Auswahl einer *ASIC*-Komponente erfordert die Pin-Anzahl und damit verbunden die Gehäuseform automatische bestimmte Anforderungen an die Fertigungstechnologie, welche wesentliche technologisch bedingte Randbedingungen wie Fertigungskosten, Ausbeute oder Zuverlässigkeit eines eingebetteten Systems mitbestimmen. Dieser Zusammenhang ist in Abbildung 4.13 qualitativ angegeben. Im folgenden Text wird eine Einteilung in verschiedene Technologie-Klassen eingeführt, welche als Grundlage der anschließenden Technologie-Bewertung dient:

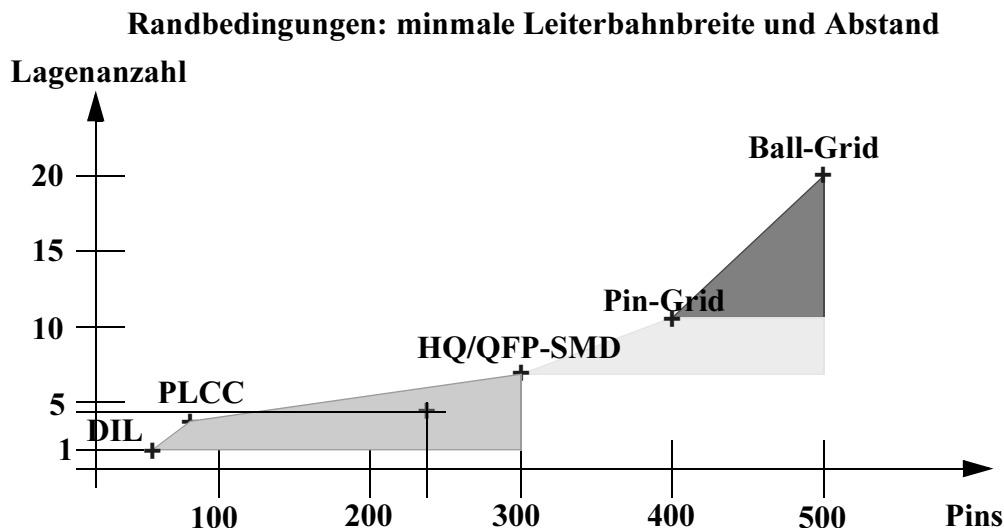


Abbildung 4.13 Anstieg der Lagenanzahl in Abhängigkeit der Pin-Anzahl am Chip-Gehäuse

- **Einfache technologische Anforderungen - Kategorie E:** Bausteine in einfacher *Dual-In-Line*-Technik (*DIL*) können auf einseitigen oder doppelseitigen Leiterplatten verdrahtet werden. Diese Technologie ist sehr kostengünstig und wird in der Massenproduktion eingesetzt. Die Anzahl der Gehäuse-Pins reicht von 8 bis 50 Pins pro Bau-

stein mit einem Pin-Abstand von 100 mil (2,54 mm). Die Leiterbahnbreiten- und abstände liegen im Bereich von 16 mil (0,4 mm) bis 25 mil (0,635 mm). Diese Werte ermöglichen beim heutigen Stand der Fertigungstechnologie eine große Ausbeute und Zuverlässigkeit. Besondere Anforderungen für die Testbarkeit bestehen nicht, da alle Pins und Verbindungsleitungen beim Test frei zugänglich sind.

- **Standard-Technologie - Kategorie S:** Im Bereich zwischen 16 und 84 Pins werden sogenannte *PLCC*-Gehäuse eingesetzt, wobei der Pin-Abstand mit 100 mil relativ unproblematisch ist. Die Lagenanzahl steigt dabei auf vier Lagen an, wobei jeweils zwei Lagen für eine stabile Betriebsspannungsversorgung (*VCC* und *GND*) verwendet werden. Die Fertigungskosten sind auch bei dieser Technik relativ niedrig. Diese Leiterplatten sind weitgehend in einfacheren Systemen der industriellen Automation zu finden und garantieren eine hohe Zuverlässigkeit. Dabei sinken die Leiterbahnbreiten- und abstände auf 10 mil (0,25 mm). Alle Signalleitungen sind auch bei dieser Technik frei zugänglich, weshalb bei dieser Kategorie keine besonderen Anforderungen an die Testbarkeit gestellt werden.
- **Feinleitertechnik - Kategorie F:** Eine der effizientesten Gehäuseformen für Bausteine mit Gehäuse-Pins im Bereich zwischen 100 und 300 Pins sind die sogenannten *Quad-Flat-Packages (QFP)*. Sie haben einen Pin-Abstand im Bereich von 25 mil (0,635 mm) bis 20 mil (0,5 mm). Die Leiterbahnbreiten- und abstände sinken auf 7 mil (0,18 mm) bis 9 mil (0,23 mm). Aufgrund steigender Packungsdichte der einzelnen Bauteile steigt für die notwendige Verdrahtung die Lagenanzahl auf sechs bis acht Lagen, wobei auch hier zwei Lagen für die Betriebsspannungsversorgung benutzt werden. In diesem Technologiebereich steigen die Kosten bereits stark an. Aufgrund der kleineren Strukturbreiten kommt es zu elektrischen Problemen, wie z.B. das Übersprechen, welche meßtechnisch relativ schwer zu beherrschen sind. Dadurch kommt es zu Problemen bei der Zuverlässigkeit und Ausbeute, welche durch verstärkten Eingriff des Entwicklers in die Auslegung der Verdrahtung (engl.: *layout*) begrenzt werden müssen. Aufgrund der Tatsache, daß die Signallagen im Inneren der Leiterplatte während des Tests nicht mehr zugänglich sind, werden verstärkte Anforderungen an die Testbarkeit gestellt.
- **Feinstleitertechnik (engl.: Advanced) - Kategorie A:** Das obere Ende der technologischen Entwicklung stellen zur Zeit die *Pin-Grid-* bzw. *Ball-Grid-*Gehäuse dar. Sie ermöglichen Gehäuse im Bereich zwischen 300 und 700 Pins. Die *Pin-Grid-Arrays* haben Pin-Abstände von 100 mil (2,54 mm). Beim Einsatz von *Ball-Grid-Arrays* sinken die Pin-Abstände auf 50 mil (1,27 mm) (siehe Abschnitt 2.1.5) und die Leiterbahnbreiten- und abstände auf 5 mil (0,127mm). Dieser Technologie-Standard kann bei Gehäusen bis zu 500 Pins eingesetzt werden, wobei zwischen acht und zehn Signallagen benötigt werden. Dabei werden höchste Anforderungen an die Testbarkeit gestellt, da bei der Inbetriebnahme die überwiegende Anzahl der Signalleitungen auf inneren Lagen nicht mehr zugänglich sind.
- **Höchstintegration - Kategorie H:** Bei *Ball-Grid-Array* im Bereich zwischen 500 und 700 Pins steigt die notwendige Lagenanzahl für die Verdrahtung auf bis zu 20 Lagen an, wobei auch die Leiterbahnbreiten- und abstände auf bis zu 3 mil (0,07 mm) sinken. Diese hohe Lagenanzahl erfordert, daß Durchkontaktierungen nicht durch die ganze Platine kontaktiert werden, sondern daß nur über bestimmte Lagenbereiche elektrische Verbindungen hergestellt werden (engl.: *partial vias*), um nicht alle Lagen für weitere Signalleitungen zu sperren. Diese Technologie ist zur Zeit extrem kostenintensiv und

wird nur in speziellen militärischen Anwendungen eingesetzt. Die weiter steigende Komplexität der *ASIC*-Bausteine wird aber in absehbarer Zeit den Einsatz dieser Technologie auch in weiten Anwendungsbereichen der industriellen Automation und Kommunikation erzwingen.

Diese Einteilung zeigt, daß der technologische Entscheidungsparameter „Gehäuseform“ wesentlichen Einfluß auf die Technologie eines eingebetteten Systems hat. Dabei wird die Gehäuseform als primärer Parameter betrachtet, der wie die oben beschriebene Einteilung weitere Technologieparameter, wie Lagenanzahl und Leiterbahnbreite, bestimmt. Die entsprechende Bewertung dieses Entscheidungsparameters unter Berücksichtigung der in der Spezifikation vorgegebenen Randbedingungen, wie Fertigungskosten, Ausbeute und Anforderungen an die Zuverlässigkeit, ist wesentlicher Bestandteil einer Auswahlentscheidung für eine *ASIC*-Komponente. Abbildung 4.14 stellt die Ausgangssituation bei der Technologie-Bewertung dar.

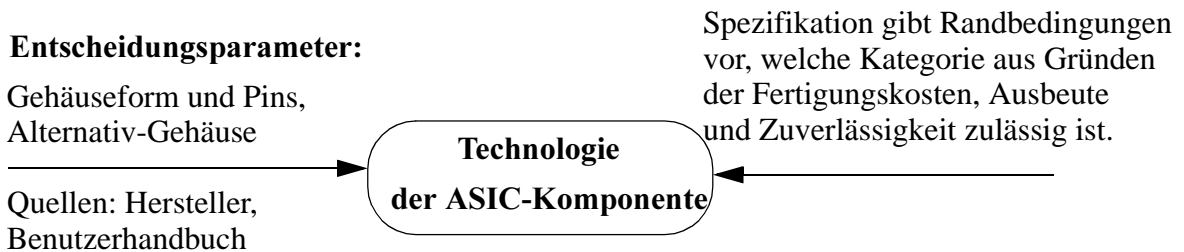


Abbildung 4.14 Entscheidungsparameter bei der Technologie-Bewertung

In Tabelle 4.3 werden die verschiedenen Abstufungen des Entscheidungsparameters „Gehäuseform“ den verschiedenen Abstufungen der zur Verfügung stehenden Fertigungskategorien gegenübergestellt. Die Bewertung wird auch bei diesem Entscheidungsparameter mit folgenden Kenngrößen durchgeführt:

- **Idealer Fall (I):** Diese Bewertung liegt vor, wenn eine Komponente unter den gegebenen Randbedingung der Fertigungskategorie keine Probleme bei der Integration auf der Leiterplatte darstellt. Dies bedeutet, daß keine besonderen Eingriffe seitens des Entwicklers notwendig sind und die Komponente vollständig von automatischen Werkzeugen verdrahtet werden kann. Aufgrund der Gehäuseform sind alle Signale auf der Leiterplatte meßtechnisch zugänglich, so daß keine speziellen Anforderungen an die Testbarkeit bei der Inbetriebnahme gestellt werden.
- **Kleiner Aufwand (K):** Diese Bewertung liegt vor, wenn es aufgrund der vorgegebenen Fertigungskategorie an bestimmten Signal-Pins (*VCC* und *GND*-Versorgung, kritische Signalnetze, wie Taktleitungen usw.) zu Engpässen kommt, die automatische Werkzeuge zur Verdrahtung nicht lösen können. Dabei sind alle Signal-Pins auf der Leiterplatte frei zugänglich, so daß keine speziellen Anforderungen an die Testbarkeit bei der Inbetriebnahme gestellt werden.
- **Mittlerer Aufwand (M):** Bei dieser Bewertung steigen die Anforderungen an den Leiterplatten-Entwurf signifikant an. Automatische Werkzeuge können nur begrenzt eingesetzt werden bzw. brauchen die massive Unterstützung des Entwicklers, insbesondere bei der optimalen Platzierung der Bauteile auf der Leiterplatte. Es werden zusätzliche Maßnahmen zur Testbarkeit (siehe Abschnitt 4.3.5) notwendig, da einige Signal-Pins auf der Leiterplatte nicht zugänglich sind.

- **Großer Aufwand (G):** Bei dieser Bewertung ist aufgrund der komplexen Gehäuseform und unter der Einhaltung der vorgegebenen Fertigungskategorie kein Einsatz automatischer Werkzeuge für Platzierung und Verdrahtung möglich. Eine Implementierung erfordert eine weitgehend vom Entwickler für die zu entwerfende Leiterplatte optimierte Lösung. Diese Anforderung erhöht signifikant die Entwicklungszeit und somit die Kosten. Es werden erhebliche Maßnahmen zur Testbarkeit (siehe Abschnitt 4.3.5) notwendig, da nur wenige Signal-Pins auf der Leiterplatte zugänglich sind.
- **Nicht verwendbar (NV):** Bei dieser Bewertung besteht keine Möglichkeit, unter Einhaltung der vorgegebenen Fertigungskategorie die Komponente auf der Leiterplatte zu implementieren. Die Komponente ist nicht verwendbar und scheidet aus dem Bewertungsprozeß aus.

Kategorie	DIL	PLCC	QFP	Pin-Grid	Ball-Grid
E	K	M	NV	NV	NV
S	I	K	G	NV	NV
F	I	I	M	G	NV
A	I	I	K	M	G
H	I	I	I	K	M

Tabelle 4.3: Bewertung des Entscheidungsparameters „Gehäuse“ für die Technologie

4.3.5 Entscheidungsparameter für die Testbarkeit

Bei der Bewertung der Entscheidungsparameter für die Testbarkeit muß man zwischen den Fähigkeiten auf dem Chip bzw. auf der Platine unterscheiden. In dem Maße, wie die *on-Chip*-Testfunktionalität abnimmt, müssen externe Einrichtungen auf der Leiterplatte vorgesehen werden, um dieses Defizit auszugleichen. In Abbildung 4.15 ist die Ausgangssituation bei der Bewertung der Testeigenschaften graphisch dargestellt.



Abbildung 4.15 Bewertung des Entscheidungsparameters für die Testbarkeit

Auch in diesem Entscheidungsfeld lassen sich unterschiedlich leistungsstarke Testeigenschaften definieren, die in folgende Kategorien mit absteigenden Testeigenschaften eingeteilt werden können:

- **Echtzeitfähige Hardware-Testunterstützung - Kategorie E:** Diese Kategorie bietet die leistungsfähigste Testunterstützung. Dabei können alle internen und externen Funktionszustände überwacht werden, ohne daß die Ausführungsgeschwindigkeit in irgend einer Weise beeinflusst wird. Zur Durchführung des Tests werden auf der Leiterplatte keine weiteren funktionsfähigen Komponenten benötigt. Die beste *on-Chip*-Testunterstützung bietet dafür der *BDM*-Mode. Dieser Mode basiert auf dedizierten Testeinheiten, die in Hardware mit auf dem Chip realisiert werden. Die Fähigkeiten

wurden bereits in Abschnitt 3.1.4 beschrieben. Dieser Mode ist heute hauptsächlich bei Mikrocontrollern zu finden. Bei den insbesondere in diesem Kapitel betrachteten Auswahlentscheidungen für *ASIC*-Komponenten steht dieser Mode in der Regel nicht zur Verfügung und muß durch aufwendige Testeinrichtungen auf der Leiterplatte ersetzt werden.

- **Nicht-echtzeitfähige Hardware-Testunterstützung - Kategorie NE:** Diese Kategorie ermöglicht auch die Überwachung aller internen und externen Funktionszustände, greift aber in die Ausführungsgeschwindigkeit signifikant ein, so daß keine Echtzeitfähigkeit mehr gegeben ist. Zur Durchführung des Tests werden auf der Leiterplatte ebenfalls keine weiteren funktionsfähigen Komponenten benötigt. Diese Test-Eigenschaft stellt z.B. der *JTAG-Port* dar, dessen Funktionsweise ebenfalls in Abschnitt 3.1.4 beschrieben wurde.
- **Schnittstellen-Testunterstützung - Kategorie - ST:** Diese Kategorie bietet keinen kompletten Einblick mehr in die internen bzw. externen Funktionszustände einer Komponente. Es wird lediglich geprüft, ob die Komponente in einem eingebetteten System physikalisch vorhanden ist. Dabei wird über die Mikrocontroller-Bus-Schnittstelle mit der *ASIC*-Komponente eine Testsequenz ausgetauscht. Voraussetzung ist, daß die Konfigurationsregister der Komponente schreib- bzw. lesbar sind und zusätzlich der Mikrocontroller-Kern funktionsfähig ist. Eine weitere Testmöglichkeit in dieser Kategorie bietet insbesondere bei *ASIC* für Kommunikationsanwendungen der sogenannte *loop-back*-Mode, bei dem die Sendedaten im Chip auf den Empfangseingang geschaltet werden. Dadurch ist ein Test des *ASIC* auf der Platine möglich, bei dem ein Mikrocontroller einen Testdatensatz in die Komponente schreibt und die gleichen Daten anschließend wieder zurück liest. Diese Kategorie kann lediglich benutzt werden, um die Aussage „Komponente betriebsbereit vorhanden“ zu untersuchen. Tiefer gehende Analysen, z.B. bei der Treiberentwicklung, sind nicht möglich, insbesondere auch in Hinblick auf das Echtzeitverhalten.
- **Ohne Hardware-Testunterstützung - Kategorie - OT:** Diese Kategorie stellt überhaupt keine Anforderungen an die Testbarkeit. Bei der Inbetriebnahme auf der Leiterplatte kann lediglich eine „geht-geht nicht“ - Aussage getroffen werden.

Kategorie	BDM	JTAG	Register lesbar/schreibbar	keine on-Chip Unterstützung
E	I	K	M	G
NE	I	I	K	G
ST	I	I	I	M
OT	I	I	I	I

Tabelle 4.4: Bewertung des Entscheidungsparameters „on-Chip-Testfunktionen“

Unter Einhaltung der Randbedingung einer geforderten Testunterstützung (z.B. vom Entwickler durch Angabe einer bestimmten Test-Kategorie E, NE, ST, OT vorgegeben) läßt sich der Testaufwand für eine *ASIC*-Komponente in Abhängigkeit von ihrer zur Verfügung stehenden *on-Chip*-Testfunktionalität durch die in Tabelle 4.4 dargestellten Abstufungen angeben. Die Abkürzungen haben dabei folgende Bedeutung:

- **Idealer Fall (I):** In diesem Fall werden die vorgegebenen Testanforderungen des Entwicklers von der Komponente in vollem Umfang durch die zur Verfügung stehenden

on-Chip-Testfunktionen der zu bewertenden Komponente erfüllt. Es werden für den Test keine weiteren Maßnahmen extern auf der Leiterplatte benötigt.

- **Kleiner Aufwand (K):** Um die vorgegebenen Testanforderungen des Entwicklers zu erfüllen, werden auf der Leiterplatte zusätzliche Testeinrichtungen benötigt. Darunter fallen bestimmte Test-Pins (z.B. Steuersignale, Taktsignale, Jumper zum Umschalten von bestimmten Modes usw.) zum Anschluß von meßtechnischen Geräten, um dem Entwickler wichtige Zusatzinformationen zu liefern, die durch die verfügbare *on-Chip*-Testunterstützung nicht gegeben werden.
- **Mittlerer Aufwand (M):** Um die Testanforderungen des Entwicklers zu erfüllen, wird ein signifikanter Bruchteil der *I/O*-Pins der Komponente auf externen Test-Pins benötigt, um mit meßtechnischen Mitteln das Defizit der *on-Chip*-Testfunktionen auszugleichen. Ein typischer Anwendungsfall ist die externe Überwachung der kompletten Bus-Schnittstelle, um die Kommunikation der Komponente mit den restlichen Systemkomponenten zu kontrollieren. Diese Bewertung des Aufwandes hat bereits eine signifikante Rückwirkung auf die Technologie-Anforderungen, da auf der Leiterplatte bereits ein großer Flächenbedarf für die Test-Pins benötigt wird.
- **Großer Aufwand (G):** In diesem Fall sind die Testanforderungen des Entwicklers hoch angesetzt und die *on-Chip*-Testfunktionen so schwach ausgeprägt, daß auf der Leiterplatte ein extremer Aufwand notwendig wird, um die geforderte Testunterstützung zur Verfügung zu stellen. Typische Implementierungen verbinden alle *I/O*-Signale der Komponente mit Test-Pins, um die komplette Funktionalität der Komponente mit einem Logik-Analysator überwachen zu können. Diese Bewertung des Aufwandes hat maßgebliche Rückwirkung auf die Technologie-Anforderungen, da auf der Leiterplatte ein Flächenbedarf für die Test-Pins benötigt wird, welcher die Fläche der eigentlichen Komponente um ein Vielfaches übertreffen kann.

4.4 Gesamtbewertung einer Komponente

Die Abbildung 4.16 zeigt den gesamten Auswahlalgorithmus, wie er in der ersten Stufe der in dieser Arbeit vorgestellten Entwurfsmethodik durchlaufen wird.

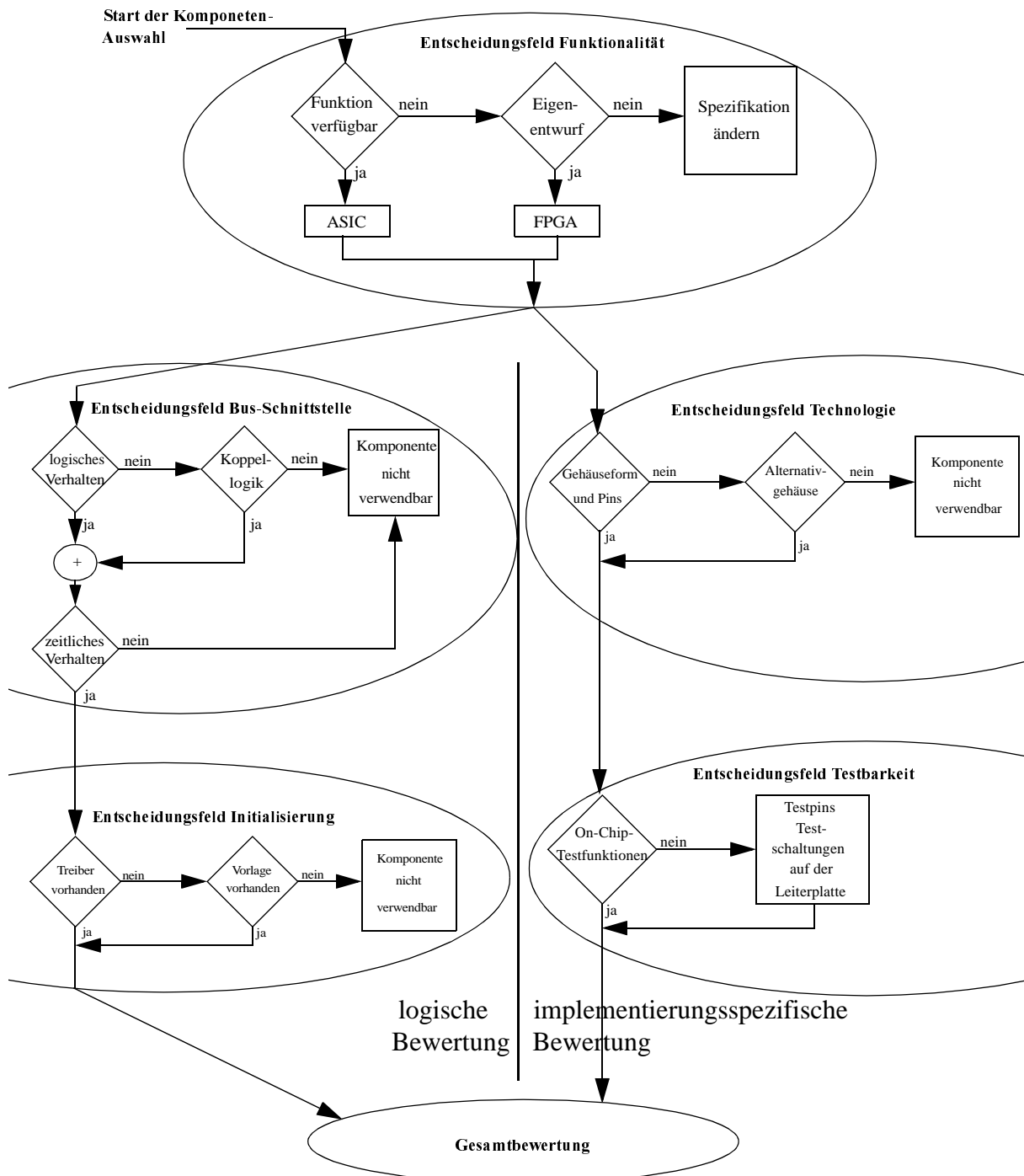


Abbildung 4.16 Gesamter Auswahlalgorithmus für die Komponenten-Bewertung

Für jede zu untersuchende Komponenten-Alternative wird der Algorithmus vollständig durchlaufen. Dabei entsteht die in Tabelle 4.5 dargestellte Gesamtübersicht, die für jede Alternative die Bewertung für die einzelnen Entscheidungsfelder angibt. Dabei werden die beiden

Teilpfade in Form eines gemeinsamen Wertes für die logische (Log = Bus + Init) und implementierungsspezifische Bewertung (Imp = Tech + Test) zusammengefaßt.

Alternative ASIC-Komponenten	Alternative 1	Alternative 2	Alternative n
Bus-Schnittstelle	Bus ₁	Bus ₂	Bus _{n}
Initialisierung	Init ₁	Init ₂	Init _{n}
Bewertung: Logischer Pfad	Log ₁	Log ₂	Log _{n}
Technologie	Tech ₁	Tech ₂	Tech _{n}
Testbarkeit	Test ₁	Test ₂	Test _{n}
Bewertung: Implementierungsspezifischer Pfad	Imp ₁	Imp ₂	Imp _{n}

Tabelle 4.5: Gesamtübersicht bei der Komponenten-Bewertung

Die Art und Weise dieser Zusammenfassung wird durch die Tabelle 4.6 ausgedrückt. Dabei wird von folgenden Annahmen ausgegangen, die im folgenden Text begründet werden:

- Wird mindestens ein Entscheidungsfeld innerhalb eines Pfades mit **I** bewertet, wird die Gesamtbewertung des Pfades von dem Wert des anderen Entscheidungsfeldes bestimmt. Darin liegt die Annahme, daß der ideale Fall bei einem Entscheidungsfeld die Gesamtbeurteilung zwar nicht verbessert, aber auch nicht verschlechtert.
- Wenn in irgendeinem Entscheidungsfeld eine Bewertung mit **NV** vorliegt, ist die Komponente prinzipiell nicht verwendbar und wird aus der Bewertung ausgeschlossen.
- Werden die zwei zusammengehörenden Entscheidungsfelder mit **K** bzw. **M** bewertet, so akkumuliert sich der Gesamtaufwand zu **M** bzw. **G**.
- Wird ein Entscheidungsfeld einmal mit **G** bewertet, kann das zweite Entscheidungsfeld die Gesamtbewertung nicht mehr verbessern.

Bus/Init oder Tech/Test	I	K	M	G	NV
I	I	K	M	G	NV
K	K	M	M	G	NV
M	M	M	G	G	NV
G	G	G	G	G	NV
NV	NV	NV	NV	NV	NV

Tabelle 4.6: Zusammenfassende Wertetabelle innerhalb eines Entscheidungspfades

Das Endergebnis dieses Prozesses liefert für jede Alternative jeweils eine Bewertung ihrer logischen Eigenschaften bzw. ihrer implementierungsspezifischen Eigenschaften. Damit erhält der Entwickler anhand eines Satzes objektiver Kriterien eine wesentliche Entscheidungshilfe, welche Komponente sich besonders gut für den Einsatz in dem zu entwerfenden eingebetteten

System eignet. Als Randbedingungen dienen dabei zum einen der selektierte Mikrocontroller, zum anderen bestimmte Vorgaben der Spezifikation bzw. des Entwicklers.

Die beiden Bewertungen bezüglich der logischen und implementierungsspezifischen Eigenschaften werden an dieser Stelle bewußt nicht weiter zusammengefaßt. Sie beschreiben praktisch zwei völlig verschiedene Bewertungsbereiche, die in einer allgemeingültigen Formel nicht miteinander in Einklang zu bringen sind. Diese Abwägung bleibt dabei weiter der subjektiven Entscheidung des Entwicklers überlassen, der an dieser Stelle weitere Randbedingungen wie die zur Verfügung stehende Entwicklungszeit oder seine eigene Entwicklungserfahrung benutzen muß, um damit einer bestimmten Komponente aus der Menge der zur Verfügung stehenden Alternativen eine Priorität zu geben, welche dann unmittelbar in der zweiten Stufe dieser Methodik durch Emulation überprüft wird.

Diese zweite Stufe ist die Überprüfung der in der ersten Stufe gefundenen Bewertung. Dazu wurden Werkzeuge entwickelt, die in Kapitel 5 vorgestellt und in Kapitel 6 anhand zweier praxisrelevanter Beispiele demonstriert werden.

Betrachtet man die tägliche Arbeit eines Entwicklers, so ist das Finden und Beurteilen der in diesem Kapitel eingeführten Entscheidungsparameter eine sehr zeitaufwendige Arbeit und besteht in einer sehr intensiven Analyse des Benutzerhandbuches bzw. einer aufwendigen Kommunikation mit den verschiedenen Herstellern der Komponenten. Darin liegt die Motivation, diesen Vorgang weiter zu systematisieren und später auch zu automatisieren. Diese Arbeit kann daher auch als Grundlage für weiterführende Arbeiten verstanden werden, bei denen die vollständige Auswahl von Komponenten durch bewertende Methoden in Datenbanken ermöglicht werden soll. Hauptschwierigkeit dabei ist, funktionale und nicht-funktionale Eigenschaften, die jeweils unterschiedliche Rollen innerhalb der Auswahlentscheidung spielen, miteinander in Beziehung zu setzen. Für die funktionalen Eigenschaften wurde in diesem Kapitel bereits ein Satz objektiver Kriterien und deren Bewertung vorgeschlagen.

Eine weitere Voraussetzung ist, die angegebenen Entscheidungsparameter in Form von informationstechnisch lesbaren Informationen abzulegen. Erste Bestrebungen der Halbleiterindustrie gibt es dazu bereits unter dem Stichwort „elektronisches Datenblatt“. Die Implementierung des vorgestellten Auswahlalgorithmus in Verbindung mit den definierten und elektronisch verwertbaren Entscheidungsparametern, welche in entsprechenden Datenbanken abgelegt werden müssen, ermöglicht so die Entwicklung von *CAD*-Werkzeugen zum automatischen Architekturentwurf durch Bewertung von Komponenten für eingebettete Systeme, welcher dem menschlichen Vorgehen dabei sehr ähnlich ist.

Kapitel 5

Emulationsumgebung SPYDER

In Kapitel 4 wurde ein Auswahlverfahren für *ASIC*-Hardware-Komponenten eingeführt, welches, wie in Abbildung 4.3 dargestellt, von fünf Entscheidungsfeldern abhängig ist. Wenn im wichtigsten Entscheidungsfeld, der geforderten Funktionalität, eine oder sogar mehrere *ASIC*-Komponenten gefunden werden, entscheiden die restlichen vier über die Einsetzbarkeit einer *ASIC*-Komponente. Die Emulationsumgebung muß Einrichtungen zur Verfügung stellen, die eine Überprüfung der Realisierbarkeit der getroffenen Auswahlentscheidung erlauben. Das bedeutet, daß die verbleibenden vier Entscheidungsfelder, die im wesentlichen das logische Verhalten und die effektive Implementierung einer Komponente im Gesamtsystem beschreiben, auf der Emulationsumgebung überprüft werden müssen. Die Unterstützung des in Abschnitt 3.3 bereits eingeführten Entwurfsablaufs wird durch die Emulationsumgebung SPYDER ermöglicht und ist in Abbildung 5.1 im Detail dargestellt.

Für den Fall, daß die geforderte Funktionalität als auf dem Markt verfügbare *ASIC*-Komponente nicht zur Verfügung steht, muß eine Eigenentwicklung durchgeführt werden. Dabei muß es möglich sein, den notwendigen Hardware-Entwurf zunächst vom Entwurf des restlichen Systems zu trennen, um einen parallelen Entwurfsablauf zu ermöglichen. Für diesen Schritt müssen die Emulationswerkzeuge verschiedene *FPGA*-Architekturen zur Verfügung stellen, um die entwickelte Hardware zunächst getrennt zu emulieren und anschließend ins eingebettete Gesamtsystem zu integrieren. Ein weiterer Punkt ist deshalb die Evaluierung verschiedener *FPGA*-Architekturen für die effiziente Implementierung der Hardware in der Serienproduktion.

Zur Durchführung der genannten Forderungen wurde im Rahmen dieser Arbeit die Emulationsumgebung SPYDER entwickelt. Diese besteht aus zwei Teilwerkzeugen. Das erste Werkzeug heißt SPYDER-CORE-P1 [WeSt98] und dient im wesentlichen zur Emulation von einzelnen *ASIC*-Komponenten sowie der gesamten eingebetteten Systemarchitektur während der Systemintegration und des Systemtests. Im Mittelpunkt steht die bereits angesprochene Überprüfung der Realisierbarkeit einer Auswahlentscheidung. Das zweite Werkzeug SPYDER-ASIC-X2 [WeOe98] unterstützt hauptsächlich die Emulation eines Eigenentwurfs von digitaler, anwendungsspezifischer Hardware.

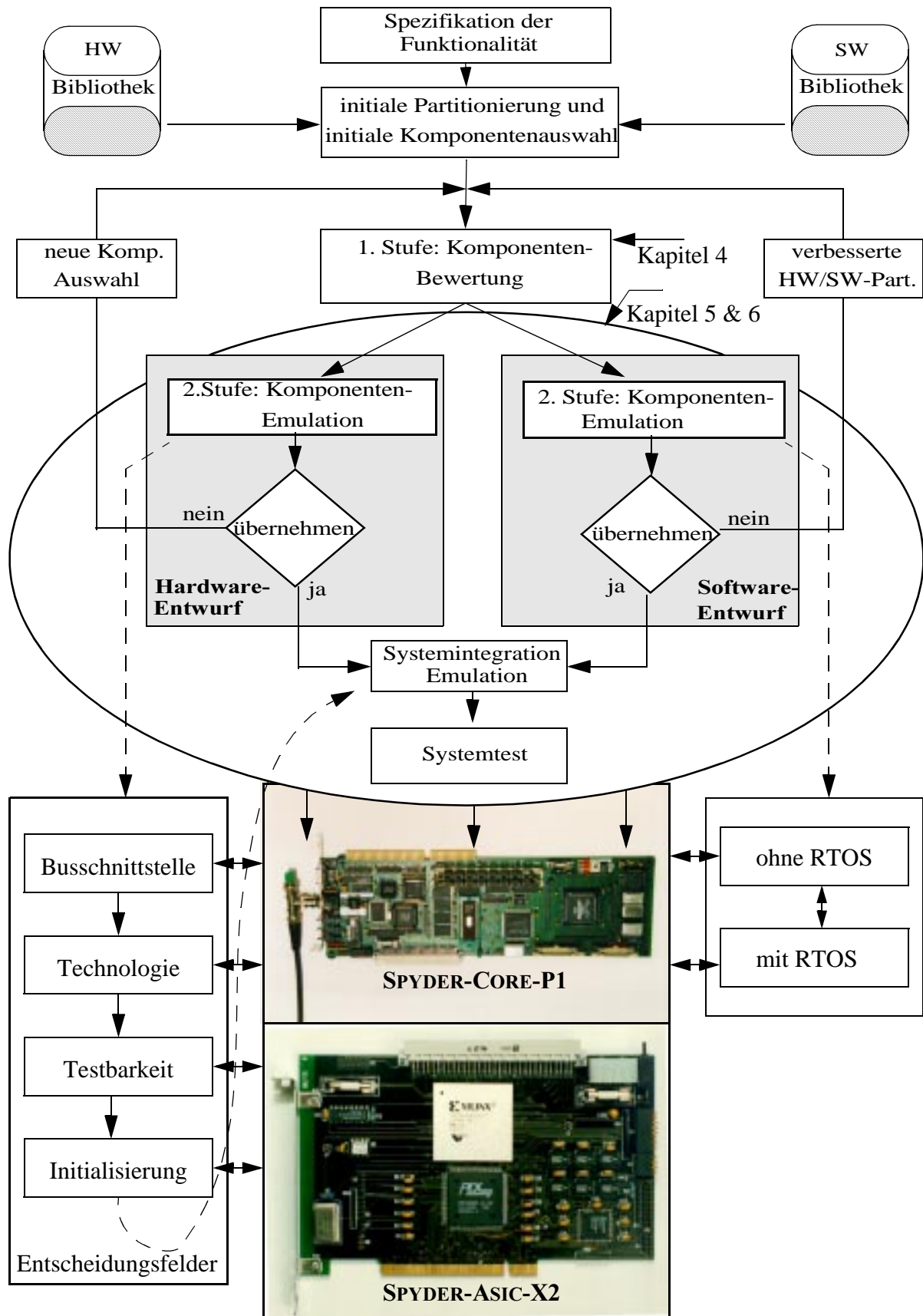


Abbildung 5.1 Unterstützung des Entwurfsablaufs durch das Emulationssystem SPYDER

In Abschnitt 5.1 wird das Werkzeug SPYDER-CORE-P1 eingeführt und gezeigt, wie die in Kapitel 4 dargestellten verschiedenen Entscheidungsfelder durch Emulation eindeutig bewertet werden können. In Abschnitt 5.2 wird die Software-Umgebung des SPYDER-Systems beschrieben, insbesondere das portierte Echtzeitbetriebssystem *VxWorks*, die Unterstützungs-Software und der Software-Monitor *SDS70*.

Abschnitt 5.3 beschreibt das Werkzeug SPYDER-ASIC-X2 und erläutert die Aufgaben, welche im Rahmen der in dieser Arbeit vorgestellten Entwurfsmethodik von diesem Werkzeug abgedeckt werden. In Abschnitt 5.4 werden die wichtigsten Eigenschaften der gesamten Emulationsumgebung zusammengefaßt, und Abschnitt 5.5 diskutiert einige wirtschaftliche Aspekte beim Einsatz der Emulationsumgebung SPYDER.

5.1 SPYDER-CORE-P1

Das Einsatzspektrum von SPYDER-CORE-P1 kann in zwei Hauptaufgaben beschrieben werden. Die erste Aufgabe ist die Emulation einer ausgewählten Hardware-Komponente. Dabei werden die vier Entscheidungsfelder Bus-Schnittstelle, Initialisierung, Technologie und Testbarkeit einer bestimmten Hardware-Komponente an einem real existierenden Objekt überprüft. Nach Möglichkeit nehmen an dieser Emulation nur die Funktionsbaugruppen teil, die zur Emulation der zu überprüfenden Komponente unbedingt gebraucht werden.

Die zweite Aufgabe ist die Emulation des Gesamtsystems vor dessen Überführung in die Serienfertigung. Dabei müssen alle benötigten Hardware-Komponenten zusammen mit der vollständigen Anwendungssoftware emuliert werden.

Wie in Kapitel 4 beschrieben, basiert die hier vorgestellte Entwurfsmethodik auf einem Mikrocontroller-getriebenen Ansatz. SPYDER-CORE-P1 stellt dazu im wesentlichen alle Hardware-Komponenten zur Verfügung, die den eingebetteten Systemen der betrachteten Teilklasse in Abschnitt 4.1 gemeinsam sind und in vielen verschiedenen Projektentwicklungen immer wieder gebraucht werden. Im Bereich der Software bietet SPYDER-CORE-P1 zur Emulationsunterstützung einen Software-Monitor und für bestimmte Projekte den Einsatz des Echtzeitbetriebssystems *VxWorks* (siehe Abschnitt 5.2) an. Flexible und leistungsstarke Schnittstellen erlauben die Adaption von speziellen Komponenten, die nur für ein bestimmtes Projekt benötigt werden.

Die Beschreibung der Funktionalität der Emulationsumgebung ist wie folgt strukturiert: Zunächst werden alle Funktionseinheiten, die im Blockdiagramm in Abbildung 5.2 dargestellt sind, funktional beschrieben. Anschließend wird ihre Bedeutung für die Emulation der in Kapitel 4 eingeführten Entscheidungsfelder dargestellt. Gleichzeitig werden Verweise auf das Kapitel 6 gegeben, wo diese Funktionseinheiten zur Emulation zweier Beispielapplikationen eingesetzt werden.

5.1.1 Mikrocontroller-Kern

Die wichtigste Komponente ist der eingesetzte Mikrocontroller. Er gibt ein bestimmtes Bus-Verhalten vor, an das sich alle anderen Hardware-Komponenten anpassen müssen. Auf der Basis-Platine in Abbildung 5.2 steht ein *Embedded PowerPC 403GA/GCX* zur Verfügung, der zusammen mit den beiden *DRAM*-Speicherbänken den eigentlichen Mikrocontroller-Kern bil-

det. Dieser Mikrocontroller-Typ wurde aus folgenden Gründen für die in dieser Arbeit betrachteten eingebetteten Systeme ausgewählt:

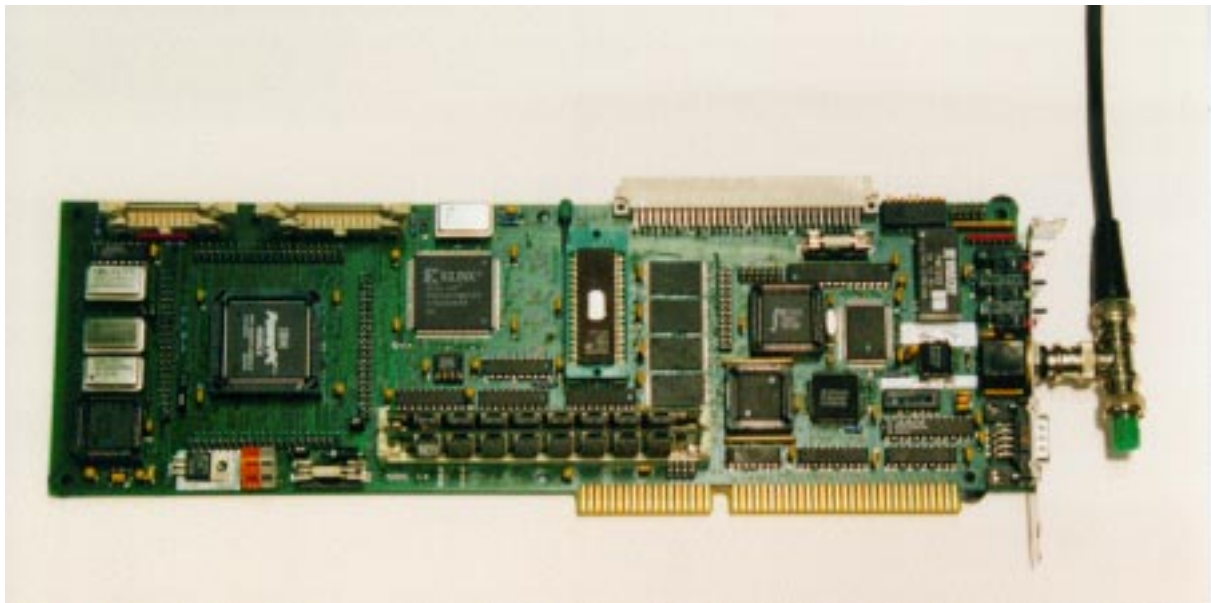
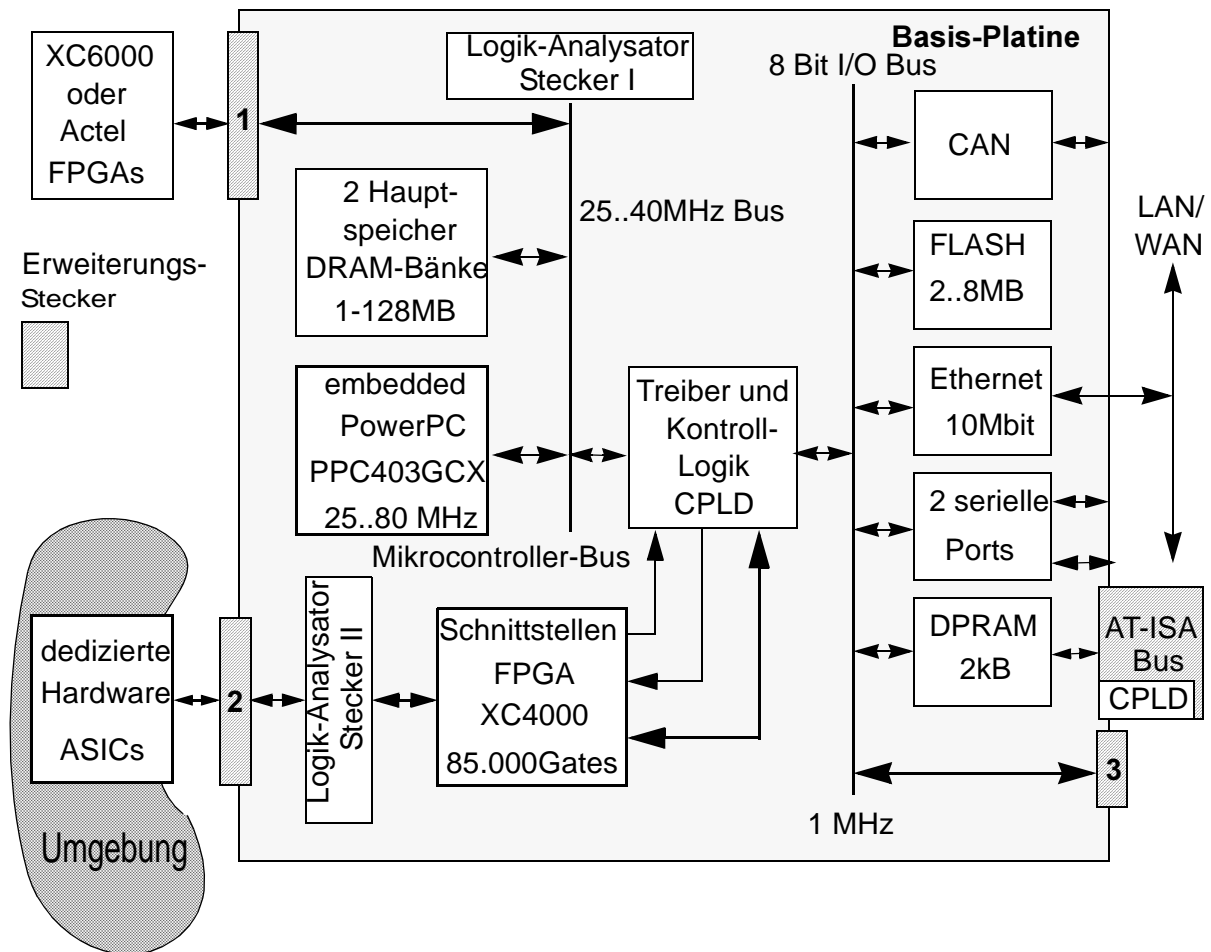


Abbildung 5.2 Blockdiagramm und Bild von SPYDER-CORE-P1

- Die Prozessorfrequenz muß über einen möglichst großen Bereich skalierbar sein, um die Leistungsfähigkeit an verschiedene Anwendungen anzupassen. Der *PPC403GA/GCX* ist im Bereich zwischen 25 MHz bis 80 MHz zur Zeit verfügbar, wobei Werte bis 200 MHz angekündigt sind. Dadurch wird die Wiederverwendung einer Mikrocontroller-Architektur in mehreren verschiedenen Anwendungen wesentlich erleichtert. Durch Emulation kann der optimale Wert für die Taktfrequenz ermittelt werden, bei dem alle Echtzeitbedingungen der speziellen Anwendung gerade noch erfüllt werden, um ein über- bzw. unterdimensioniertes eingebettetes System zu vermeiden.
- Die Bus-Schnittstelle muß sehr flexibel sein, um unterschiedlichste Speicher und *ASIC*-Komponenten anzuschließen. Darunter versteht man Komponenten mit 8-Bit, 16-Bit oder 32-Bit breiten Datenbussen und unterschiedlichem zeitlichen Anschlußverhalten. Die Fähigkeit dieses Mikrocontrollers, sich auf unterschiedliche Eigenschaften dieser Komponenten einzustellen, wurde in Abschnitt 2.1.1.3 bereits als “*dynamic bus sizing*” eingeführt und erläutert. Damit können einfache bzw. auch komplexe eingebettete Systeme mit relativ wenig Koppellogik implementiert werden.
- Die wichtigsten Peripherie-Komponenten, die in den meisten eingebetteten Systemen benötigt werden, sind auf dem Mikrocontroller-Chip verfügbar. Darunter versteht man Einheiten wie einen *Interrupt*-Controller, *DMA*-Controller, *DRAM*-Controller, Adressdecoder und eine serielle Kommunikationsschnittstelle.
- Diese Mikrocontroller-Architektur bietet verschiedene Typen von *on-Chip-Caches* (Befehls- und Datencache) an. Zusätzlich bieten verschiedene Mitglieder der *PPC403*-Familie unterschiedlich große Caches (z.B. *PPC403GA* 2K-I-Cache/1K-D-Cache oder der *PPC403GCX* 16K-I-Cache/8K-D-Cache) an.

Diese Cache-Eigenschaften bieten die Möglichkeit, die Analyse der Leistungssteigerung in bezug auf unterschiedliche Cache-Konfigurationen in schnellen Echtzeitsystemen, welche unter der Kontrolle eines *RTOS* ablaufen, durchzuführen. Diese Arbeiten stellen zur Zeit ein interessantes Forschungsgebiet dar und werden auch in dieser Arbeit betrachtet.

- Der *CPU*-Kern stellt hochentwickelte Ausführungseinheiten mit Pipeline-Verarbeitung, Superskalarität und Sprung-Vorhersage zur Verfügung. Diese *PowerPC*-Architektur besitzt somit alle Eigenschaften moderner Mikrocontroller-Architekturen, welche die Ausführungsgeschwindigkeit massiv erhöhen. Andererseits bewirken diese Fähigkeiten, daß das Systemverhalten nicht mehr exakt vorhersagbar ist, insbesondere bei eingebetteten Systemen mit harten Echtzeitbedingungen. Eine Methode, dennoch eine genaue Kenntnis des internen Systemverhaltens zu bekommen, besteht in der *worst-case*-Analyse, welche während der Emulation durchgeführt wird. Dieser Ansatz wird an dem Anwendungsbeispiel in Abschnitt 6.2 demonstriert.

Der gesamte Mikrocontroller-Kern wird aus dem Mikrocontroller und seinem Hauptspeicher gebildet. Dazu sind zwei *Simm*-Steckplätze vorhanden, in die jeweils nach Bedarf der zu emulierenden Anwendung *DRAM*-Module mit einer Speicherkapazität zwischen 1 MByte bis 64 MByte eingesetzt werden können. Der *DRAM*-Controller auf dem *PPC403*-Chip stellt alle notwendigen Signale zur Verfügung, um diese Speicherbausteine ohne zusätzliche Schaltungen anzusteuern.

Da jeder weitere Anschluß einer Komponente an den Mikrocontroller-Bus eine kapazitive Belastung darstellt, welche die maximale Zugriffsgeschwindigkeit herabsetzt, werden alle an-

deren Komponenten durch ein *CPLD* abgekoppelt. Für diese Komponenten ist aus Sicht der Ausführungsgeschwindigkeit ein extrem schneller Zugriff nicht notwendig. Deshalb sind diese Komponenten vom direkten Mikrocontroller-Bus durch bidirektionale (für Daten) und unidirektionale (für Adress- und Steuerleitung) Treiber getrennt. Die maximale Anzahl der kapazitiv belasteten Eingänge, welche mit dem Mikrocontroller direkt verbunden sind, werden damit auf zwei (*DRAM* und Treiber selbst) begrenzt. Diese Maßnahme ermöglicht eine hohe Speicher-Zugriffsgeschwindigkeit. Bei der Verwendung eines *PPC403GCX* kann dadurch die Frequenz auf dem Chip mit 80 MHz und der externe Bus mit 40 MHz betrieben werden, ohne daß bei dieser Bus-Geschwindigkeit Probleme mit der kapazitiven Busbelastung auftreten.

Zum eigentlichen Mikrocontroller-Kern zählt zusätzlich noch eine der beiden seriellen Schnittstellen, welche bei Verwendung des *PPC403* auf dem Chip mit integriert ist. Dieser Mikrocontroller-Kern kann in dieser Form, wie in Abschnitt 4.1 eingeführt, für alle eingebetteten Systeme der betrachteten Teilklasse verwendet werden. In Abschnitt 6.1 und Abschnitt 6.2 werden zwei unterschiedliche Anwendungen gezeigt, die diesen Mikrocontroller-Kern verwenden.

5.1.2 Schnittstellen-FPGA

Die zweite zentrale Komponente neben dem Mikrocontroller-Kern ist das Schnittstellen-*FPGA*. In der aktuellen Version von *SPYDER-CORE-P1* wird ein *SRAM*-basiertes *Xilinx-FPGA* der *XC4000*-Architektur (siehe Abschnitt 2.1.3.2) eingesetzt. Die Gehäuseform (*HQ/PQFP208*) erlaubt die Bestückung eines Chips mit einer maximalen Anzahl von 85.000 Logik-Gattern. Die Hauptfunktion liegt in der Anpassung der Schnittstelle des Mikrocontrollers an das Verhalten der Bus-Schnittstelle des zu untersuchenden *ASIC*, welcher auf Systemebene in seiner Zusammenarbeit mit dem Mikrocontroller emuliert werden soll. Die strukturelle Wirkungskette bei dieser Emulation zeigt Abbildung 5.3.

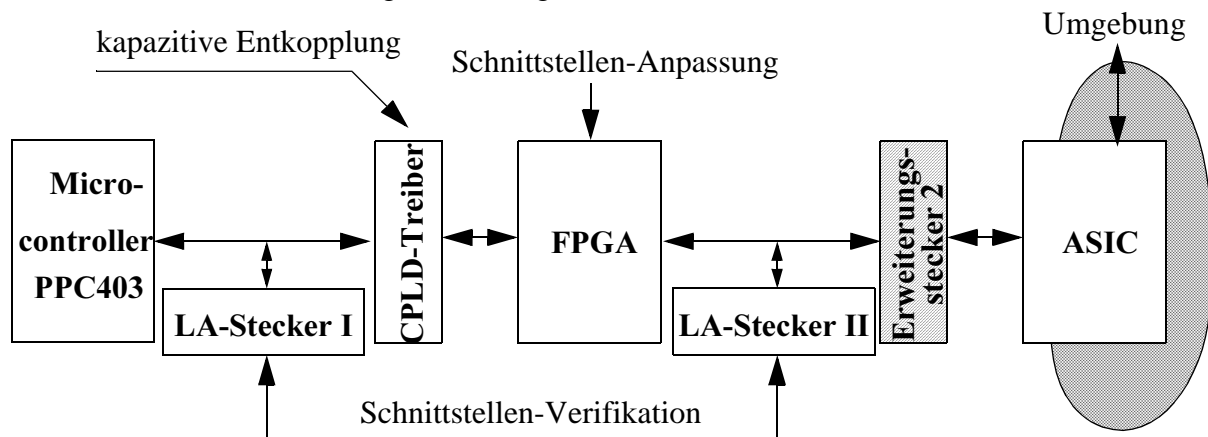


Abbildung 5.3 Wirkungskette bei der ASIC-Bus-Schnittstellen-Emulation mit SPYDER-CORE-P1

Die in Abbildung 5.3 dargestellte Wirkungskette dient zur Bearbeitung der beiden Entscheidungsfelder Bus-Schnittstelle und Initialisierung aus Abschnitt 4.2.

- **Entscheidungsfeld Bus-Schnittstelle:** Alle Fragen, die mit der Ankopplung der *ASIC*-Komponente an den Mikrocontroller verbunden sind, können mit diesem Emulationsaufbau bearbeitet werden. Die eigentliche Anpassung des logischen und zeitlichen Verhaltens der Bus-Schnittstelle an das Verhalten der *ASIC*-Komponente wird im Schnittstellen-*FPGA* implementiert. Die Treiber im *CPLD* koppeln den Mikrocontrol-

ler-Bus von einer weiteren kapazitiven Belastung ab. Funktional greift dieser Baustein aber nicht in das Verhalten ein, lediglich die Durchlaufzeit von 7,5 ns -10 ns muß beim zeitlichen Verhalten berücksichtigt werden. Die Verifikation dieser Bus-Ankopplung kann durch die beiden Logik-Analysator-Stecker **I** und **II** vor und nach dem *FPGA* meßtechnisch überprüft und gegebenenfalls entsprechend modifiziert werden.

Die Komplexität der im Schnittstellen-*FPGA* zu implementierenden Logik kann in bezug auf die Anzahl benötigter Logik-Gatter von relativ einfachen kombinatorischen Schaltungen bis zu aufwendigen anwendungsspezifischen Anpassungsschaltungen reichen. Betrachtet man den einfachen Fall, bei dem die *ASIC*-Komponente im wesentlichen von den Bussignalen des Mikrocontrollers direkt bedient werden kann, müssen die Bussignale nur durch das *FPGA* durchgeschaltet werden. In diesem Fall muß allerdings die typische Durchlaufzeit von ca. 20 ns beim zeitlichen Verhalten berücksichtigt werden, wie in Abbildung 4.5 graphisch dargestellt.

In der Regel werden aber kompliziertere Anpassungen benötigt. In Abschnitt 6.1 wird an einem Beispiel aus der Kommunikationstechnik gezeigt, daß für den Datenaustausch zwischen einer *ASIC*-Komponente und dem Mikrocontroller ein gemeinsamer Pufferspeicher gebraucht wird. Der gleichzeitige Zugriff von beiden Seiten auf diesen Puffer muß durch einen *Arbiter* gesteuert werden, welcher in Form einer Zustandsmaschine im Schnittstellen-*FPGA* implementiert und verifiziert werden kann.

Im Abschnitt 6.2 wird anhand eines Beispiels aus der industriellen Automation gezeigt, daß eine solche Anpassung aus einer anspruchsvollen digitalen parallel-seriell-Umwandlungs- und Fehlerüberwachungs-Hardware bestehen kann, die zwischen 4.000 und 16.000 Gatteräquivalenten benötigt.

- **Entscheidungsfeld Initialisierung einer *ASIC*-Komponente:** Wenn eine *ASIC*-Komponente mit einem Mikrocontroller über eine funktionsfähige Busschnittstelle verbunden ist, können die Routinen zur richtigen Initialisierung des *ASIC* sowie die entsprechenden darauf aufbauenden Treiber angepaßt beziehungsweise entwickelt werden. Dabei kann die *ASIC*-Komponente zunächst losgelöst von den restlichen Teilen des eingebetteten Systems für sich alleine emuliert werden. Betrachtet man den Entwurfsablauf in Abbildung 5.1, wird bei der parallelen Emulation der linke Pfad des Hardware-Entwurfs in der zweiten Stufe durchlaufen. Insbesondere wird hier für die zu emulierende *ASIC*-Komponente die Frage beantwortet, ob sie in die Architektur des eingebetteten Systems aufgenommen werden kann (*<ja>*) oder ob der Rückführungspfad (*<nein>*) zur Komponentenauswahl in der ersten Stufe genommen wird. Bei auftretenden Problemen während der Emulation dieser als Firmware bezeichneten Routinen bieten die Logik-Analysator-Stecker **I** und **II** eine effektive Unterstützung bei der Analyse des logischen bzw. implementierungsspezifischen Verhaltens an.

5.1.3 Erweiterungs- und Logik-Analysator-Verbindungen

Auf der SPYDER-CORE-P1 Basis-Platine befinden sich insgesamt drei Erweiterungsstecker und zwei Stecker zum Anschluß eines Logik-Analysators. Diese Steckerverbindungen bieten eine leistungsstarke Möglichkeit, dedizierte Komponenten gezielt mit dem Mikrocontroller-Kern zu verbinden und so das eigentliche eingebettete System zu konfigurieren.

5.1.3.1 Erweiterung für ASIC-Komponenten

Diese Erweiterung bietet die Möglichkeit, insbesondere die beiden Entscheidungsfelder Testbarkeit und Technologie bei einem ASIC zu bearbeiten und die damit verbundenen Probleme vom restlichen eingebetteten System auf der Basis-Platine zu trennen. Die Methodik für die beiden Entscheidungsfelder wird in Abbildung 5.4 dargestellt.

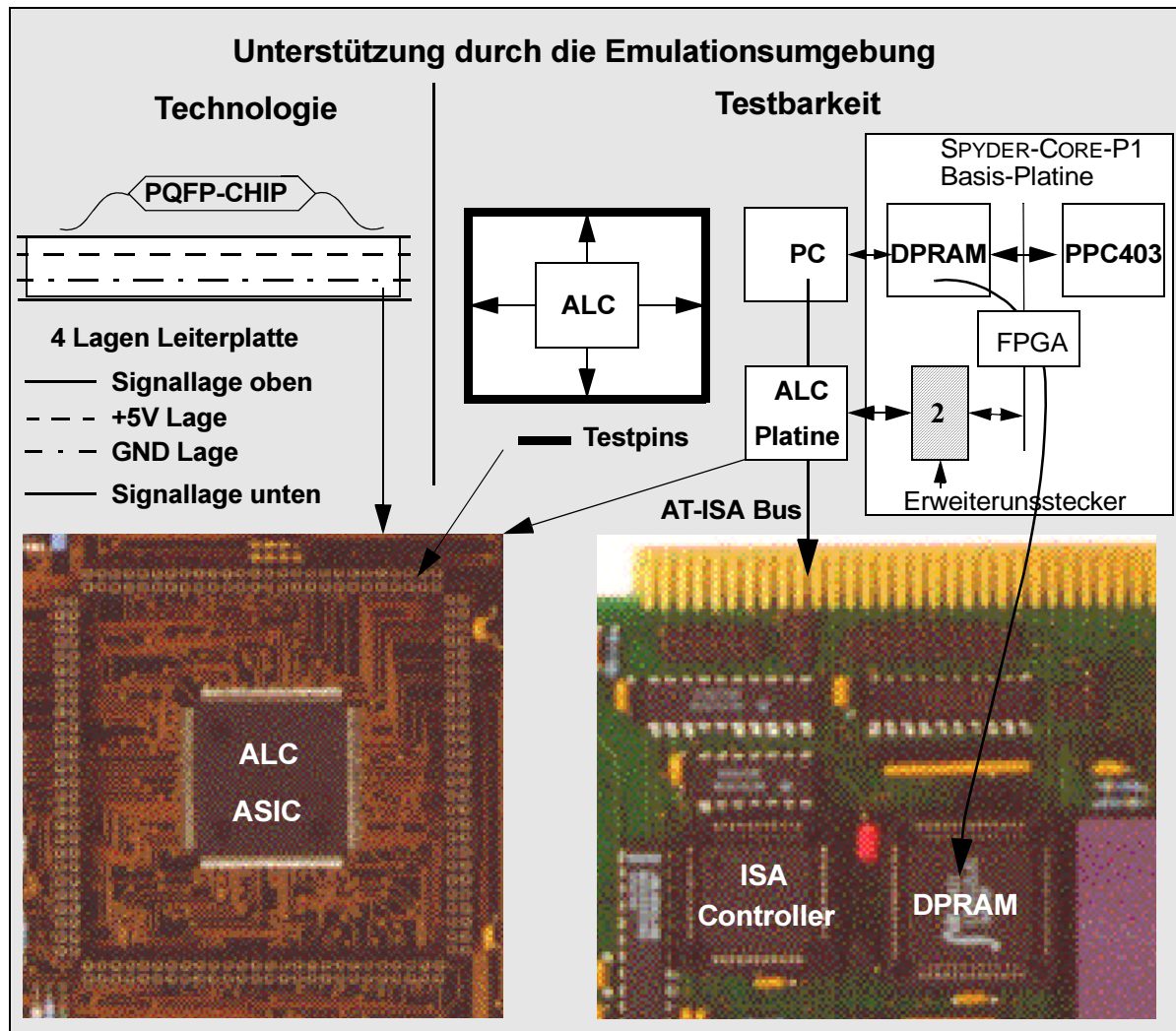


Abbildung 5.4 Unterstützung der Entscheidungsfelder durch die Emulationsumgebung SPYDER

- **Entscheidungsfeld Testbarkeit:** In Abschnitt 3.1.4 wurde bereits eingeführt, daß die Unterstützung des Funktionstests auf dem Chip bei einer ASIC-Komponente im Gegensatz zu einem Mikrocontroller sehr schlecht ausgeprägt ist. Als typisches Beispiel dafür wird in Abschnitt 6.1 ein sogenannter *Adaption Layer Controller-(ALC)-ASIC* emuliert, welcher keine Testmöglichkeiten auf dem Chip bietet. Deshalb müssen dem Systementwickler auf der Platine Möglichkeiten geboten werden, um diesen Test effizient durchführen zu können.

Diese Testmöglichkeit wird durch eine spezielle Ansteckplatine erzeugt, die den zu emulierenden ASIC trägt und zusätzlich eine auf seine spezifischen Eigenschaften optimierte Testunterstützung zur Verfügung stellt. Auf der linken Seite der Abbildung 5.4 ist eine solche Emulationsplatine dargestellt, die alle Signalleitungen des ALC-

ASIC auf Test-Pins zur Verfügung stellt. Dadurch erhält der Systementwickler die Möglichkeit, die gesamte Aktivität des *ASIC*-Bausteins während der Emulation durch entsprechende Analysewerkzeuge (z.B. Logik-Analysator, Oszilloskop) unter Echtzeitbedingungen zu verfolgen. Diese Maßnahme stellt die leistungsfähigste Methode dar, um das Fehlverhalten einer Komponente auf Systemebene zu lokalisieren.

Auf der rechten Seite der Abbildung 5.4 ist die Architektur beim Test auf der SPYDER-CORE-P1 Basis-Platine dargestellt. Hier wird der Test im wesentlichen durch ein *Dual-Port-Ram (DPRAM)* zwischen dem Mikrocontroller und einem Entwicklungsrechner (PC) unterstützt. Bei elementaren Hardware-Tests kommt es häufig vor, daß der Entwickler beim Fehlverhalten einer Komponente die Kontrolle über den korrekten Programmablauf auf dem Mikrocontroller verliert. Eine Teststrategie, welche den Datenaustausch zwischen Mikrocontroller und *ASIC* in dem *DPRAM* zwischenspeichert, ermöglicht in diesem Fall, daß der PC die Programmsequenzen bis zum Fehlverhalten nachträglich auslesen und dem Entwickler anzeigen kann. Dieses Vorgehen ist eine einfache, aber sehr wirkungsvolle Methode beim Testen von Hardware-Komponenten.

- **Entscheidungsfeld Technologie:** Moderne *ASIC*-Komponenten mit bis zu 500 Gehäuse-Pins benötigen ein aufwendiges technologisches Verfahren für die Prototypen-Produktion. Wenn der für die Implementierung notwendige Aufwand in Form von feinsten Leiterbahnbreiten auf hochintegrierten Multilayer-Platinen mit den Kostenanforderungen des Projektes vereinbar ist, wird in dieser Arbeit eine Methode zur effizienten und in bezug auf das Entwicklungsrisiko kalkulierbaren Emulation durchgeführt. Auf die Problematik bei der Überlagerung von technologischen und funktionalen Fehlern wurde bereits in Abschnitt 3.1.4 hingewiesen.

Die Abtrennung dieser technologischen Probleme von anderen Entwurfsproblemen wird im wesentlichen dadurch erreicht, daß speziell für die zu emulierende *ASIC*-Komponente eine eigene Trägerplatine entwickelt wird, welche der SPYDER-CORE-P1-Emulationsplattform am Erweiterungsstecker zugesteckt wird. Dieses Vorgehen basiert im wesentlichen darauf, daß die SPYDER-CORE-P1-Hardware als eine erprobte und fehlerfreie Basis-Hardware wiederverwendbar zur Verfügung steht und dadurch keine technologischen Fehler aufweist. Die möglichen technologischen Fehler werden auf die Trägerplatine begrenzt und gefährden somit nicht die Funktion des gesamten zu emulierenden eingebetteten Systems. Die Analyse dieser möglichen Fehlfunktionen ist eng verbunden mit dem oben dargestellten Entscheidungsfeld Testbarkeit.

An dieser Stelle wird festgestellt, daß die Entwicklung und Produktion einer auf die jeweilige Komponente zugeschnittenen Trägerplatine zunächst einen Mehraufwand bedeuten. Dieser Punkt steht damit im wesentlichen Gegensatz zu der in Abschnitt 3.2.1 beschriebenen praxisorientierten Methode, in der prinzipiell in einem ersten Implementierungsschritt die gesamte Systemarchitektur auf einer Platine zusammenfaßt wird, um sofort zur endgültigen Version für die Serienproduktion zu kommen.

Dieser Mehraufwand ist aber relativ gering im Gegensatz zu den Nachteilen, welche während der Testphase von eingebetteten Systemen entstehen, die aus hochkomplexen *ASIC*-Komponenten bestehen, ohne dabei leistungsfähige Analysemöglichkeiten zu besitzen. Insbesondere die Überlagerung der Entwurfsprobleme kann sich zu einem unkalkulierbaren Entwicklungsrisiko aufsummieren, welches das gesamte Projekt gefährden kann.

5.1.3.2 Erweiterung für Co-Prozessoren

Das Blockdiagramm in Abbildung 5.2 zeigt den Erweiterungsstecker 1, an dem auf der Basis-Platine der gesamte Mikrocontroller-Bus zur Verfügung steht. Dadurch kann die Emulationsumgebung mit Komponenten erweitert werden, welche sich ohne zusätzliche digitale Schaltungen an das logische wie zeitliche Verhalten des Mikrocontrollers anpassen können. Diese Erweiterungsmöglichkeit stellt somit den schnellsten Datenaustausch zwischen externen Komponenten um dem Mikrocontroller sicher, ohne die Durchlaufzeiten durch *CPLD* oder *FPGA* zu verbrauchen. In der Regel wird diese Möglichkeit für Co-Prozessoren benutzt, welche direkt mit dem Mikrocontroller zusammenarbeiten. Im Rahmen dieser Arbeit wurde dieser *Port* verwendet, um ein partiell rekonfigurierbares *FPGA* (siehe Abschnitt 2.1.3.2) mit der Emulationsumgebung zu kontaktieren. Im Beispiel in Abschnitt 6.1 wird dieses *FPGA* und die damit verbunden Entwurfsmethoden bezüglich der Rekonfiguration zur Laufzeit evaluiert, und es wird gezeigt, welchen Nutzen diese *FPGA*-Bausteine für eine Anwendung haben können.

5.1.3.3 Erweiterung für einfache Peripherie-Komponenten

Ebenfalls im Blockdiagramm in Abbildung 5.2 ist der Erweiterungsstecker 3 dargestellt. Physikalisch bietet er eine 8-Bit breite Mikrocontroller-Schnittstelle, welche durch Treiberbausteine vom eigentlicher Mikrocontroller-Bus kapazitiv entkoppelt ist. Dieser *Port* dient zum Anschluß von einfacheren Peripherie-Komponenten (z.B. Feldbus-Controller für *PROFIBUS* oder *Interbus*, einfache *I/O*-Komponenten), welche für eine spezielle Emulation eines eingebetteten Systems innerhalb der betrachteten Anwendungsklasse gebraucht werden. Der Erweiterungsstecker wird nur wegen der Vollständigkeit bei den Erweiterungssteckern beschrieben. Aus der Sicht des Einsatzkonzepts für die Emulationsumgebung gehört er im wesentlichen zu den allgemeinen Peripherie-Komponenten in Abschnitt 5.1.4.

5.1.3.4 Logik-Analysator-Unterstützung

Im Blockdiagramm in Abbildung 5.2 sind den Erweiterungssteckern 1 und 2 auch direkt Anschlußstecker für Logik-Analysatoren zugeordnet. Diese speziellen *SMD*-Steckverbindungen (engl.: *mictor high density connectors*) sind sehr kompakt und ermöglichen auf kleinster Platinenfläche den einfachen und kompakten Anschluß der entsprechenden Meßleitungen.

Durch die Anordnung des LA-Steckers **I** direkt am Mikrocontroller-Bus und des LA-Steckers **II** am Erweiterungsstecker 2 zwischen externer *ASIC*-Komponente und dem Schnittstellen-*FPGA* können die wichtigsten Knotenpunkte elektronischer Signalverläufe auf der Basis-Platine überwacht werden. Diese Fähigkeit ist die Grundvoraussetzung, um die Emulation effizient durchführen zu können und um einen detaillierten Einblick in das innere Systemverhalten zu bekommen.

Die Aufzeichnung des gesamten Mikrocontroller-Busses (LA-Stecker **I**) ist von besonderer Bedeutung. Moderne Logik-Analysatoren, deren Fähigkeiten bei der Definition der Architektur der Emulationsumgebung *SPYDER* berücksichtigt wurden, können die Signalverläufe auf dem Bus aufzeichnen und mit Hilfe einer speziellen Software (engl.: *disassembler*) in Maschinenbefehle zurück übersetzen, um sie anschließend mit Hilfe eines sogenannten Korrelationswerkzeuges dem Quellcode gegenüberzustellen. Durch diese Unterstützung bekommt der Entwickler einen direkten Vergleich zwischen auszuführendem Soll-Code und ausgeführtem Ist-Code. Diese Einrichtung ist eine zusätzliche Maßnahme, um die beiden Entscheidungsfelder

Testbarkeit und Initialisierung effizient mit Hilfe der Emulationsumgebung SPYDER zu bearbeiten.

5.1.4 Allgemeine Peripherie-Komponenten

Auf der Basis-Platine befinden sich einige Hardware-Komponenten, die für eingebettete Systeme im Bereich der industriellen Automation und Kommunikation häufig gebraucht werden. Zusätzlich dienen sie der Anbindung der Emulationsumgebung an den Entwicklungsrechner, auf dem eine Vielzahl von Entwicklungswerkzeugen laufen. Die wesentlichen Komponenten und ihre Bedeutung für die Emulationsunterstützung werden im folgenden Text beschrieben. Diese Komponenten benötigen für die Kommunikation mit dem Mikrocontroller eine 8-Bit-Bus-Schnittstelle, die über Treiber in einem *CPLD* entkoppelt werden, wie im Blockdiagramm in Abbildung 5.2 dargestellt.

5.1.4.1 Dual-Port-RAM und AT-ISA-Bus-Schnittstelle

Die Hardware des Emulationssystems SPYDER besteht prinzipiell aus Einsteckkarten für den *AT-ISA*-Bus des PC. Diese Betriebsart wird im wesentlichen während der ersten Entwicklungsphase benutzt, in der digitale Hardware-Schaltungen für die Implementierung im *FPGA* bzw. Code für die Ausführung auf dem Mikrocontroller generiert werden. Nach dieser Generierung wird durch entsprechende Software-Unterstützungswerkzeuge der auf dem Mikrocontroller ausführbare Code bzw. die *Bit-Files* für das *SRAM*-basierte *FPGA* über die *AT-ISA*-Bus-Schnittstelle auf die Emulationsumgebung geladen. Diese Daten (engl.: *bit image*) können entweder direkt zur Ausführung gebracht werden oder in Festwertspeichern gespeichert werden.

Diese Speicherung in Festwertspeichern ist die Voraussetzung für den sogenannten *Stand-Alone-Mode*, beim dem die Basis-Platine aus dem Entwicklungsrechner entfernt werden kann. Dadurch ist es möglich, die Emulationsplattform mit der eigentlichen Umgebung in Verbindung zu bringen, in der das zu emulierende eingebettete System betrieben werden soll. Dieser Betriebsmodus wird im wesentlichen in einer fortgeschritteneren Phase während der Emulation benutzt.

Als Austauschmedium ist ein 2 K x 8-Bit (2 KByte) großes *DPRAM* verfügbar, auf das von zwei Seiten sowohl völlig asynchron als auch gleichzeitig zugegriffen werden kann. Auf der einen Seite greift der PC über den *AT-ISA*-Bus zu, auf der anderen Seite greift der Mikrocontroller mit einem 8-Bit-Speicherbuszugriff (engl.: *memory mapped I/O*) zu. Die notwendige Zugriffsllogik seitens des *AT-ISA*-Busses sowie das Laden des *FPGA* wird von einem speziellen *ISA*-Bus-Controller gesteuert, der in einem für den Benutzer unsichtbaren *CPLD* implementiert ist.

Zusätzlich unterstützt dieses *DPRAM* auch die Testbarkeit von *ASIC*-Komponenten, wie in Abschnitt 5.1.3.4 beschrieben und in Abbildung 5.4 dargestellt.

5.1.4.2 Ethernet-Schnittstelle

Die Ethernet-Schnittstelle verbindet die Emulationsplattform zunächst mit einem sogenannten *Local Area Network (LAN)*. Die untere physikalische Übertragungsschicht realisiert ein serielles Protokoll mit einer Baudrate von 10 MBit/sec.

Um diese Schnittstelle effizient nutzen zu können, werden zusätzlich höhere Software-basierte Protokolle wie *TCP/IP* benötigt, welche von dem Echtzeitbetriebssystem *VxWorks* zur Verfügung gestellt und in Abschnitt 5.2 beschrieben werden.

5.1.4.3 *Controller Area Network (CAN) - Schnittstelle*

Die *CAN*-Schnittstelle ist insbesondere im Automotive- und industriellen Automationsbereich zu finden und basiert ebenfalls auf einer seriellen Datenübertragung. Um die Emulation von solchen Anwendungen zu unterstützen, wurde ein kompletter *CAN*-Controller in Form eines dedizierten *ASIC* auf der Basis-Platine implementiert. Das Echtzeitbetriebssystem *VxWorks* wurde mit einem speziellen Treiber erweitert, welcher diesen *ASIC* in höhere Übertragungsschichten einbindet.

5.1.4.4 *Flash-Festwertspeicher*

Die SPYDER-CORE-P1 Basis-Platine verfügt über insgesamt vier 16 MBit-*Flash*-Chips mit einer Organisation von 2 Meg x 8-Bit und einer Gesamtspeicherkapazität von 8 MByte. Die *Flash*-Bausteine dienen zur nicht-flüchtigen Speicherung des *Boot-Codes* und des Echtzeitbetriebssystems.

Insbesondere bei der Verwendung von *VxWorks* steht ein *Dos-File-System* zur Verfügung, welches dem Anwender die Benutzung der *Flash*-Bausteine wie eine Festplatte gestattet. Neben dem Anwendungsprogramm können auf dieser *Flash*-Festplatte auch verschiedene *Bit-Files* für das Schnittstellen-*FPGA* gespeichert werden. Zusätzlich wurde *VxWorks* um Treiber-Funktionen erweitert, welche die Installation dieser *Bit-Files* im *FPGA* durchführen. Damit kann der Anwender durch einfache Systemaufrufe das *FPGA* beschreiben und so eine einfache und effiziente globale Rekonfiguration zur Laufzeit durchführen.

5.2 Software-Umgebung des SPYDER-Systems

Das Grundkonzept des SPYDER-Systems besteht in der Verbindung vorhandener Entwurfswerkzeuge für Hardware und Software auf einer gemeinsamen Zielplattform. Die speziell für SPYDER entwickelte Software-Unterstützung hat im wesentlichen die Aufgabe, die Ausgangsprodukte der einzelnen Entwurfswerkzeuge für Mikrocontroller und *FPGA* auf die Emulationsplattform zu übertragen, bzw. die Plattform während des Betriebes im Entwicklungsrechner zu kontrollieren.

In Abbildung 5.5 werden die wichtigsten Entwurfswerkzeuge auf dem Entwicklungsrechner (engl.: *cross development PC*) dargestellt. Die leistungsfähigste Unterstützung bei der Emulation von eingebetteten Systemen erhält der Entwickler beim Einsatz eines Echtzeitbetriebssystems, z.B. *VxWorks*. Dies gilt sowohl für die Unterstützung auf dem PC als auch auf der Emulationsplattform SPYDER selbst. Für einfachere System-Emulationen kann auch der Software-Monitor *SDS70* (siehe Abschnitt 4.1.2) benutzt werden. In den beiden folgenden Unterabschnitten wird der Emulationsablauf mit *VxWorks* und *SDS70* anhand der in Abbildung 5.5 dargestellten Graphik beschrieben, wobei alle speziell für SPYDER entwickelten Werkzeuge im Programmpaket SPYDER-Software zusammengefaßt wurden und in Abschnitt 5.2.4 beschrieben werden.

5.2.1 Echtzeitbetriebssystem *VxWorks*

Auf der linken Seite der Abbildung 5.5 wird der Entwurfsablauf zur Erzeugung von ausführbarem Code für *VxWorks* dargestellt. Die Grundlagen des Echtzeitbetriebssystems wurden in Abschnitt 2.2.1 eingeführt. An dieser Stelle wird nur die PC-seitige Entwurfsumgebung von *VxWorks* vorgestellt, welche im Programmpaket *Tornado* zusammengefaßt ist, sowie die Anbindung von *Tornado* an die Emulationsplattform SPYDER-CORE-P1.

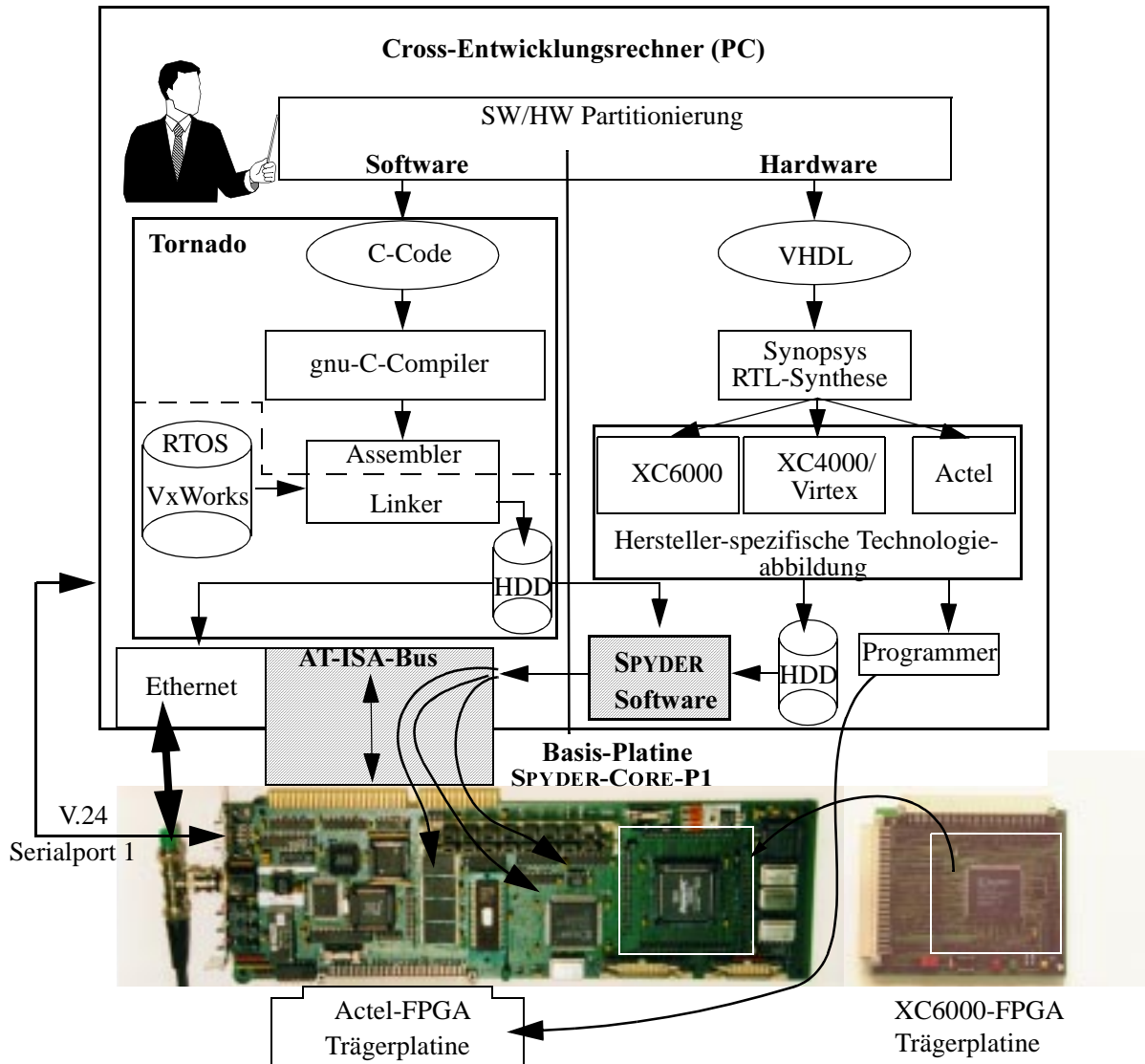


Abbildung 5.5 Zusammenführung verschiedener Software-Entwicklungswerkzeuge auf der Emulationsumgebung SPYDER-CORE-P1

Der Entwickler bestimmt im Partitionierungsschritt die benötigten Software- und Hardware-Teile. Die Software wird in C geschrieben und mit Hilfe des *gnu-C-Compilers* in Assembler-Code übersetzt. Im C-Code können verschiedene Betriebssystemaufrufe von *VxWorks* enthalten sein, welche vom Linker aus der *RTOS*-Bibliothek mit eingebunden werden. Der Linker und die *RTOS*-Bibliothek bleiben dabei dem Benutzer verborgen. Das Ausgangsprodukt der Entwicklungsumgebung ist ein ausführbarer Programmcode, welcher zunächst in einem *Objekt-File* auf der Festplatte des PC gespeichert wird. Es existieren zwei Möglichkeiten, diesen Programmcode auf der Emulationsplattform zur Ausführung zu bringen.

Auf der SPYDER-CORE-P1 Basis-Platine läuft das eigentliche Echtzeitbetriebssystem *VxWorks*, welches zusätzlich das Kommunikationsprotokoll *TCP/IP* unterstützt und über Ethernet mit dem Entwicklungsrechner kommuniziert. Über diese Verbindung kann der Code geladen und auf dem Mikrocontroller gestartet werden. Das Anwendungsprogramm im Objekt-Format ist dabei im Arbeitsspeicher frei verschiebbar und wird vom Laufzeitsystem entsprechend beim Programmstart ausgewählt. Diese als erste Möglichkeit bezeichnete Anbindung an die Emulationsplattform dient im wesentlichen zum schnellen Test von Programmsegmenten.

Die zweite Möglichkeit besteht in der Erzeugung eines kompletten Programmpakets, bestehend aus Initialisierungssequenzen, *VxWorks*-Kernel und Anwendungsprogrammen in Form eines Speicherabbildes zur Speicherung in einem Festwertspeicher. Ein Software-Werkzeug im SPYDER-Betriebssoftware-Paket kann dieses Festwertspeicher-Programm über den *AT-ISA*-Bus in den *Flash*-Bereich der Basis-Platine übertragen. Diese als zweite Möglichkeit bezeichnete Vorgehensweise wird insbesondere vor einer Herausnahme der Emulationsplattform aus dem PC benutzt, um die Konfiguration für den Einsatz in der realen Umgebung des zu emulierenden eingebetteten Systems im *Stand-Alone-Mode* durchzuführen.

Wesentliche Grundlage für die Entscheidung des Einsatzes eines Betriebssystems als Software-Komponente sind Kenngrößen wie *Task*-Umschaltzeiten, *Interrupt*-Reaktionszeiten und zusätzlicher Speicherplatzbedarf. In Abschnitt 6.2 wird der beschriebene Entwurfsablauf unter *VxWorks* an einem Beispiel demonstriert und die entsprechenden Kenngrößen während der Emulation bestimmt. Beim Einsatz von *VxWorks* auf der SPYDER-CORE-P1-Basis-Platine verwaltet das Betriebssystem alle Hardware-Ressourcen, welche aus dem Arbeitsspeicher, Festwertspeicher, *FPGA* und allen Peripherie-Komponenten bestehen. Um auf diese Komponenten zugreifen zu können, benötigt das Betriebssystem dedizierte Treiber, welche man bei *VxWorks* als Firmware-*BSP* bezeichnet (engl.: *Board Support Package - BSP*). Dieses *BSP* wurde im Rahmen dieser Arbeit entwickelt und ist Bestandteil des SPYDER-Betriebssoftware-Pakets auf der Emulationsplattform. Damit wird der Entwickler von allen Hardware-spezifischen Programmier-Aufgaben der Basis-Platine bei der Emulation seiner Anwendung entlastet.

Setzt man voraus, daß SPYDER-CORE-P1 als wiederverwendbare Emulationsplattform schon zu Beginn eines Projektes zur Verfügung steht und *VxWorks* die Hardware-Komponenten der Basis-Platine durch Integration im Betriebssystem in einfacher Weise zur Benutzung anbietet, so können bereits Software-Komponenten entwickelt und emuliert werden, bevor die gesamte Architektur der Ziel-Hardware des eingebetteten Systems definiert ist. Betrachtet man den dargestellten Entwurfsablauf der in dieser Arbeit vorgeschlagenen Methodik in Abbildung 5.1, so ermöglicht die Emulationsplattform die Parallelisierung des Entwurfsablaufs für Hardware und Software. Dabei wird der rechte Pfad des Software-Entwurfs durchlaufen, bei dem bereits Software-Teile emuliert werden können, während gleichzeitig auf einer weiteren Basis-Platine z.B. Hardware-Komponenten (linker Pfad) ausgewählt und durch Emulation bewertet werden können.

5.2.2 Software-Monitor *SDS70*

Die Testunterstützung durch die sogenannte *Debug Exception* wurde bereits in Abschnitt 2.1.1.3 eingeführt. Sie bildet die technische Grundlage für den Betrieb eines Software-Monitors. Er besteht im wesentlichen aus zwei miteinander kommunizierenden Programmteilen, welche auf dem Mikrocontroller der Emulationsplattform und auf dem PC ausgeführt werden. Die wesentliche Funktionalität besteht im Laden des ausführbaren Programmcodes vom PC auf die Emulationsplattform, im Setzen und Kontrollieren von Programm-Unterbrechungen sowie

in der Analyse der Register und Speicherbereiche des Mikrocontroller-Kerns. Die Oberfläche des PC-seitigen Teils ist in Abbildung 4.2 dargestellt. Der Programmteil, welcher auf dem Mikrocontroller läuft, ist Bestandteil des SPYDER-Betriebssoftware-Pakets auf der Emulationsplattform und wird in einem EPROM gespeichert. Durch einen Schalter auf der SPYDER-CORE-P1 Basis-Platine kann ausgewählt werden, ob die Emulationsplattform nach dem *Reset* aus dem *Flash* mit der Programmausführung startet (engl.: *boot code*) (beinhaltet *VxWorks*) oder aus dem EPROM (beinhaltet *SDS-70 Monitor*).

Diese Betriebsart mit *SDS-70 Monitor* eignet sich zur Emulation einer Software-Architektur, die aus nur einem Hauptprogramm (einer *Single-Task*) und einer oder mehrerer *Interrupt Service Routinen (ISR)* besteht. Dabei muß der Entwickler alle Komponenten auf der Basis-Platine selbst programmieren. Der wesentliche Vorteil besteht aber in der extrem schnellen Reaktion auf einen *Interrupt* von ca. 3 µs. Dieser Wert ist etwa um den Faktor 30 schneller als beim Einsatz eines *RTOS* wie z.B. *VxWorks*.

5.2.3 Software-Umgebung für das Schnittstellen-FPGA

Auf der rechten Seite der Abbildung 5.5 ist ein typischer Entwurfsablauf für Hardware abgebildet, welche in einem *FPGA* implementiert wird. Die digitale Schaltung wird dabei in *VHDL* spezifiziert und mit Hilfe eines Synthese-Compilers in eine Netzliste übersetzt. Hersteller-spezifische Software-Werkzeuge plazieren und verdrahten die Schaltung auf der entsprechenden *FPGA*-Zieltechnologie. Auf der SPYDER-CORE-P1 Basis-Platine befindet sich ein Schnittstellen-FPGA der XC4000-Familie. Zusätzlich existieren Trägerplatinen mit XC6000 und Actel-FPGA-Chips. Die speziellen Eigenschaften der verschiedenen Architekturen sind in Abschnitt 2.1.3 dargestellt. Das Ergebnis beim Durchlauf dieses Entwurfsablaufs sind *Bit-Files*, die in das entsprechende *FPGA* geladen werden müssen. Das SPYDER-Betriebssoftware-Paket auf dem PC enthält dafür Werkzeuge, die diese Übertragung von der Festplatte des PC auf die SPYDER-Emulationsplattform durchführen. Ihre genaue Funktion wird im nächsten Abschnitt beschrieben.

5.2.4 Das SPYDER-Betriebssoftware-Paket

Neben der Entwicklung der Hardware für das SPYDER-System wurde auch ein Software-Paket entwickelt, welches die Bedienung der Emulationsumgebung innerhalb des PC und im *Stand-Alone-Mode* unterstützt. Es besteht aus einem PC-seitigen Teil und einem Teil, welcher auf der SPYDER-CORE-P1 Basis-Platine nicht-flüchtig gespeichert und ausgeführt wird.

Der Teil auf der Emulationsumgebung besteht zum einen aus dem beschriebenen Software-Monitor *SDS70* und aus einem *Boot*-Kernel für *VxWorks*. Die Auswahl zwischen beiden Betriebsmodi wird durch einen Schalter getroffen, welcher den entsprechenden Festwertspeicherbereich nach dem *Reset* in den *Boot*-Speicherbereich des Mikrocontrollers einblendet. Der *SDS70*-Monitor initialisiert die Basis-Platine und kommuniziert über eine serielle Schnittstelle mit dem Teil, welcher auf dem PC ausgeführt wird. Der *Boot*-Kernel für *VxWorks* initialisiert ebenfalls die Basis-Platine, lädt anschließend über die Ethernet-Schnittstelle vom PC das eigentliche Betriebssystem-File (*VxWorks.o*) und bringt es zur Ausführung. Danach übernimmt *VxWorks* die Kontrolle über alle Hardware-Ressourcen auf der Basis-Platine.

Der PC-seitige Teil des Betriebssoftware-Pakets lädt Programm- und *Bit-Files* für den Mikrocontroller und das *FPGA* in die Emulationsumgebung. Im einzelnen existiert folgende Unterstützung:

- **Software-Reset:** Dieses Programm versetzt das SPYDER-System im PC in den *Reset*-Zustand.
- **Software-Ready:** Dieses Programm versetzt das SPYDER-System im PC in den *Arbeits*-Zustand.
- **Xload:** Dieses Programm lädt ein *Bit-File* von der Festplatte des PC direkt in das *SRAM*-basierte *XC4000-FPGA* auf der Basis-Platine.
- **Xload6000:** Dieses Programm lädt ein *Bit-File* von der Festplatte des PC zunächst über das *DPRAM* in den Hauptspeicher (*DRAM*) des Mikrocontrollers. Der Mikrocontroller schreibt bei Bedarf dieses *Bit-File* über die parallele 32-Bit-Mikrocontroller-Schnittstelle ins *SRAM*-basierte *XC6000-FPGA*. Bei diesem Vorgang nimmt der Mikrocontroller aktiv am Ladeprozeß teil. Die Trägerplatine des *XC6000* muß zuvor auf dem Erweiterungsstecker 1 (siehe Abbildung 5.5) installiert werden.
- **Eload:** Dieses Programm überträgt ein *Bit-File* für das *XC4000-FPGA* von der Festplatte in ein spezielles serielles *EEPROM* auf der Basis-Platine. Nach dem Entfernen der Basis-Platine aus dem PC lädt der *FPGA* automatisch diesen Datensatz nach dem Einschalten der externen Betriebsspannung im sogenannten *Master-Mode* aus dem *EEPROM* und ist nach ca. 10 ms betriebsbereit.
- **Status:** Liefert Status-Informationen über die Basis-Platine.
- **ISA:** Steuert verschiedene Optionen des *AT-ISA*-Buses während des PC-Betriebs.
- **FLASHLoad:** Lädt ein Programm-File mit ausführbarem Code für den Mikrocontroller über das *DPRAM* in den *Flash*-Festwertspeicher. An diesem Vorgang nimmt der Mikrocontroller aktiv teil. Nach dem Entfernen der Basis-Platine aus dem PC *bootet* der Mikrocontroller aus diesem *Flash* und führt das nicht-flüchtig gespeicherte Programm aus. Diese Funktion realisiert somit einen *ROM*-Emulator.

5.3 SPYDER-ASIC-X2

In Abschnitt 5.1.2 wurde die Schnittstellen-Anpassung eines *ASIC* an einen Mikrocontroller-Bus beschrieben. Um diesen kritischen Schritt während der Vorbereitung zur Emulation effizient durchführen zu können, wurde das Werkzeug SPYDER-ASIC-X2 entwickelt, dessen Architektur in Abbildung 5.6 dargestellt ist.

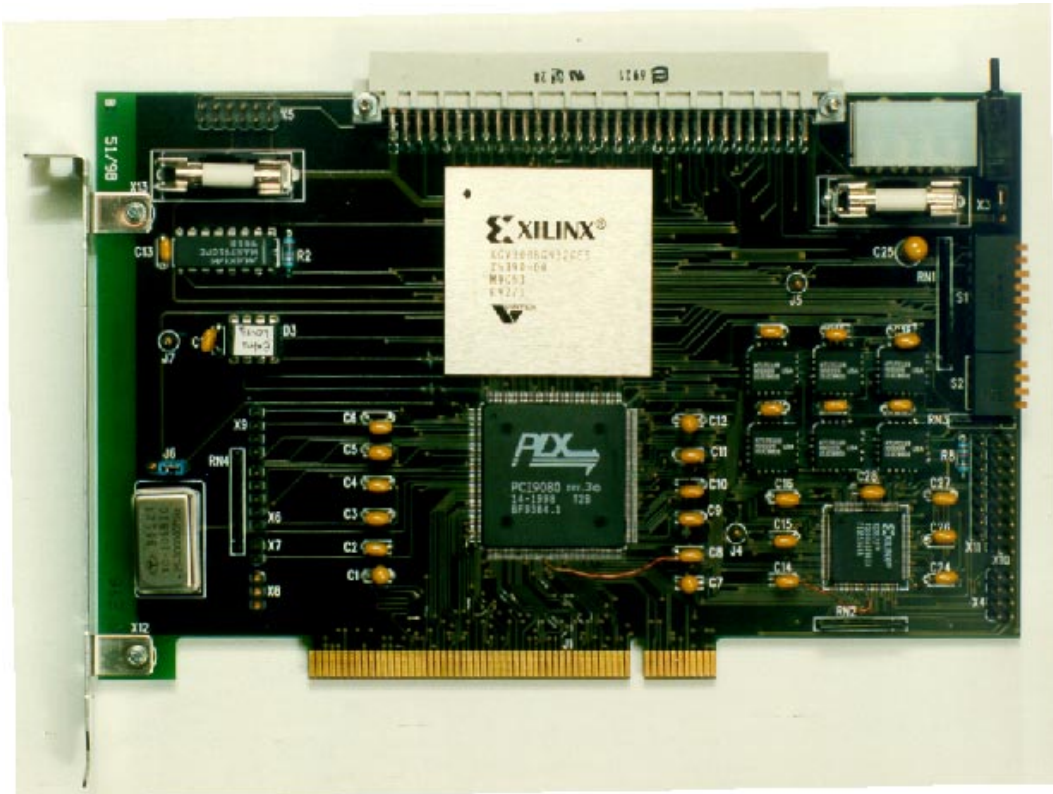
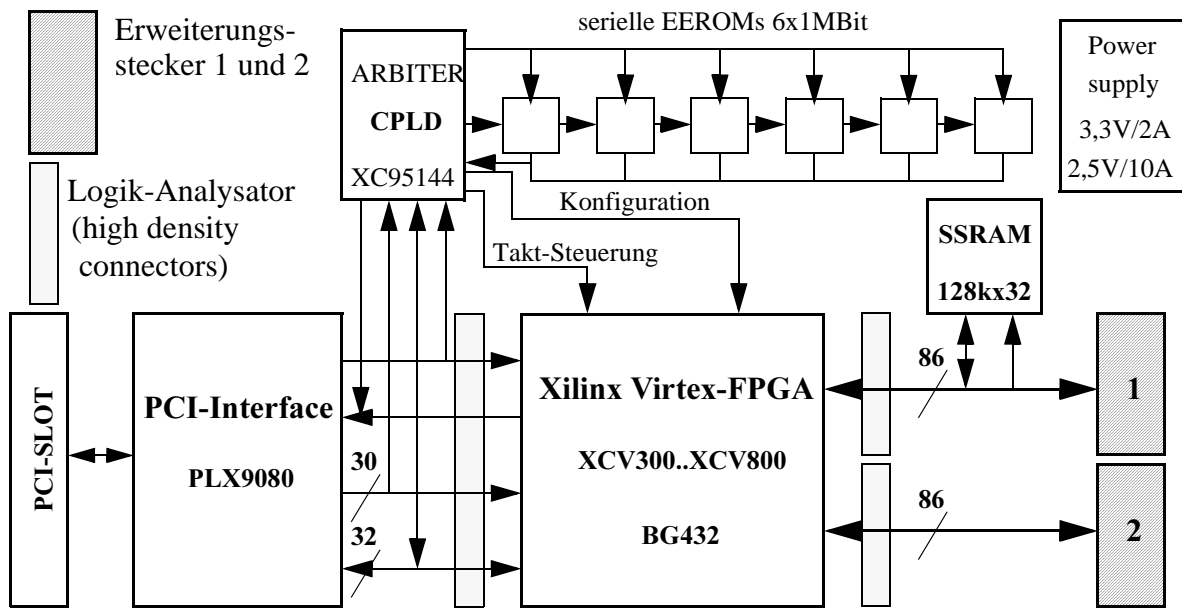


Abbildung 5.6 Blockdiagramm und Bild von SPYDER-ASIC-X2

Ergibt diese Emulation, daß die Komponente in die Architektur des gesamten eingebetteten Systems aufgenommen werden soll, kann die entwickelte Schnittstellen-Hardware zusammen mit der Trägerplatine während der Systemintegration auf der SPYDER-CORE-P1 Basis-Platine aufgrund der Kompatibilität beider Werkzeuge ohne großen Aufwand integriert werden.

- **Zusätzliche Unterstützung des Entscheidungsfeldes Initialisierung:** Es wurde bereits festgestellt, daß die Initialisierung und Treiberentwicklung bei einem komplexen *ASIC* einen signifikanten Entwicklungs- und Risikofaktor darstellt. Durch die Einblendung des *ASIC* in den Adressraum des PC kann der Entwickler die gesamten Ressourcen eines Entwicklungsrechners nutzen, wie z.B. Bildschirm, Tastatur, Festplatte, Compiler sowie leistungsstarke Software-Testwerkzeuge. Die Trennung zwischen Entwicklungssystem und Zielsystem entfällt. Dadurch wird dieses kritische Entscheidungsfeld während der Emulation wesentlich unterstützt.

Der Einsatz und die Vorteile von SPYDER-ASIC-X2 während der Emulation werden in Abschnitt 6.1 anhand einer Schnittstellen-Anpassung für eine komplexe *ASIC*-Komponente und in Abschnitt 6.2 anhand einer digitalen Hardware-Emulation demonstriert.

5.4 Zusammenfassung der SPYDER-Emulationsumgebung

Die Analyse der zur Zeit durchgeführten Entwurfsmethoden in Kapitel 3 bildet die Grundlage zur Definition der in dieser Arbeit vorgestellten zweistufigen Entwurfsmethodik, bestehend aus einem systematischen Architekturentwurf durch Bewertung sowie der anschließenden Überprüfung der auf die Bewertung gestützte Auswahlentscheidung durch Emulation. Damit diese Emulation eine hohe Aussagekraft und Akzeptanz bei den Entwicklern erhält, sind leistungsstarke Werkzeuge notwendig.

Mit dieser Zielsetzung wurde die Emulationsumgebung SPYDER definiert und implementiert. Wie im Eingangsteil von Kapitel 4 motiviert, existieren zur Zeit für die einzelnen Teilkomponenten eingebetteter Systeme verschiedene Entwicklungswerkzeuge und die damit verbundenen Entwurfsmethoden nebeneinander. Ein wesentlicher Ansatzpunkt bei der Definition von SPYDER liegt deshalb darin, diese singulären Methoden auf einer gemeinsamen Emulationsplattform zusammenzuführen und dabei möglichst effizient anwendbar zu machen. Eng damit verbunden ist auch das Bestreben, die verschiedenen Teile des Entwurfsablaufs, welcher in Abbildung 5.1 dargestellt ist, möglichst voneinander zu entkoppeln und zu parallelisieren.

Das SPYDER-Emulationssystem besteht dabei aus zwei Werkzeugen, welche von einem gemeinsamen Betriebssoftware-Paket in eine PC-basierte Entwicklungsumgebung eingebunden werden. Im wesentlichen greift die SPYDER-Betriebssoftware die Ausgangsprodukte von *State-of-the-Art* Entwicklungswerkzeugen für *FPGA* und Mikrocontroller auf und überträgt sie auf eine gemeinsame Emulationsplattform.

Das Werkzeug SPYDER-CORE-P1 ist die zentrale Basis-Platine und dient der Emulation von einzelnen *ASIC*-Komponenten, der Emulation von Software-Komponenten sowie der abschließenden Systemintegration aller Teilkomponenten eines eingebetteten Systems.

Das Werkzeug SPYDER-ASIC-X2 ist ein Hilfswerkzeug für SPYDER-CORE-P1, welches insbesondere den Mikrocontroller durch den Mikroprozessor des Entwicklungsrechners ersetzt.

5.5 Wirtschaftliche Aspekte beim Einsatz von SPYDER

Betrachtet man die Gesamtkosten für ein eingebettetes System, so muß man zwischen den Produktionskosten in der Serie und den Entwicklungskosten unterscheiden. In vielen Anwendungen in der industriellen Automation und Kommunikation liegen die Stückzahlen im Bereich von wenigen 100 bis 1000 Stück pro Jahr. Hier spielen die Entwicklungskosten und die Entwicklungszeit eine wesentliche Rolle.

Um diese beiden Faktoren zu begrenzen, beeinflussen die angewendete Methodik und die damit verbundenen Entwicklungswerkzeuge den Erfolg einer Entwicklung wesentlich. Analysiert man die Gründe, warum bei Entwicklungsprojekten das Budget für diese beiden Faktoren nicht eingehalten werden kann, so kommt man immer wieder zu folgenden Problemstellungen, die sogar eine gewisse zeitliche Reihenfolge während des Entwurfsablaufs besitzen:

- **Technologische Probleme:** Platinen, die aus der Fertigung kommen, haben Produktionsfehler, deren Lokalisierung und Beseitigung einen sehr zeitaufwendigen Prozeß darstellen.
- **Fehlverhalten von Komponenten:** Die Entwickler haben die Funktionsweise einer oder mehrerer Komponenten falsch interpretiert bzw. die Angaben im Datenblatt sind nicht richtig. Daraus ergeben sich schwierige Entwurfsänderungen, die mit erheblichem Zeitaufwand verbunden sind.
- **Software-technische Probleme:** Bei komplexen Software-Strukturen ergeben sich erhebliche Probleme beim Test. Unvorhersehbares Verhalten des Betriebssystems bzw. der Zielhardware führen zu erheblichen Verzögerungen.
- **Probleme beim Einhalten der Spezifikation:** Wenn die gesamte Anwendung implementiert ist, machen sich weitere Probleme bemerkbar, z.B wenn bestimmte Randbedingungen der Spezifikation wie Reaktionszeiten, Leistungsaufnahme, Wärmeentwicklung usw. nicht mehr eingehalten werden können.

Der Innovationsdruck erzwingt, daß eingebettete Systeme von der Idee bis zum fertigen Produkt in immer kürzeren Zeiträumen entwickelt werden müssen. Rückschläge in der Entwicklung verlängern den Entwicklungszeitraum und gefährden somit den wirtschaftlichen Erfolg. Beachtet man weiter, daß ein Personen-Monat in der Industrie mit ca. DM 20.000 kalkuliert wird, können Probleme, die erst spät im Entwicklungsablauf entdeckt werden, nicht nur das einzelne Projekt gefährden, sondern in extremen Fällen auch die Existenz eines Unternehmens.

Durch Einsatz der in dieser Arbeit vorgeschlagenen Methodik und der Emulationsumgebung SPYDER werden diese Entwicklungsrisiken beherrschbar. Wesentlicher Ansatzpunkt dafür ist die Begrenzung von technologischen Fehlern auf kleine Trägerplatinen für die jeweilige Komponente und die frühzeitige Emulation von Teilkomponenten. Die parallele Emulation von Hardware- und Software-Teilen reduziert die Totzeit, in der z.B. Software-Entwickler aufgrund fehlender Zielhardware nicht effektiv am Entwicklungsprozeß mitarbeiten können. Da das Emulationssystem SPYDER wesentliche Teile des eingebetteten Systems in getesteter und wiederverwendbarer Art und Weise zur Verfügung stellt, kann in wesentlich kürzerer Zeit eine funktionsfähige Zielhardware emuliert werden. Dadurch wird sehr frühzeitig ein tiefer Einblick in die Funktionsweise des eingebetteten Systems gewonnen, wodurch eine enorme Reduktion der Entwicklungskosten und der Entwicklungszeit erreicht wird (siehe auch Kapitel 6).

Kapitel 6

Ergebnisse der Methodik

In diesem Kapitel wird der in Abbildung 5.1 dargestellte Entwurfsablauf anhand von zwei praxisrelevanten eingebetteten Systemen aus dem Bereich der Kommunikation (Abschnitt 6.1) und der industriellen Automation (Abschnitt 6.2) demonstriert. Diese beiden Beispiele zeigen zum einen die Notwendigkeit eines systematischen Architekturentwurfs durch Bewertung und die Effizienz der Emulationsumgebung SPYDER bei der Emulation einzelner Komponenten sowie des gesamten eingebetteten Systems. In Abschnitt 6.3 wird ein Vergleich der in dieser Arbeit vorgestellten Methode mit der bisher in der Praxis eingesetzten Methode dargestellt.

6.1 ATM-Diagnose-Monitor

Dieses Anwendungsbeispiel zeigt die Auswahl einer komplexen *ASIC*-Komponente anhand des in Kapitel 4 eingeführten Auswahlalgorithmus. Die Entscheidungsfelder werden anhand eines sogenannten *Adaption Layer Controller-ALC-ASIC* für den Asynchronen Transfer Modus (ATM) bewertet. Dadurch wird die Komponente systematisch für die Architektur des eingesetzten Systems ausgewählt. Anschließend wird gezeigt, wie die einzelnen Entscheidungsfelder der *ASIC*-Komponente separat durch Einsatz des SPYDER-ASIC-X2 Werkzeugs überprüft werden können.

Zur Emulation des Gesamtsystems wird das Werkzeug SYPDER-CORE-P1 eingesetzt. Dabei wird ein Teil der Gesamtfunktionalität, welche nicht in Software auf dem Mikrocontroller bzw. als eine auf dem Markt verfügbare *ASIC*-Komponente implementiert werden kann, durch einen eigenen Hardware-Entwurf vervollständigt. Zusätzlich werden verschiedene *FPGA*-Zieltechnologien durch Emulation unter Echtzeitbedingungen bewertet. Auch diese Analyse wird durch die SYPDER-CORE-P1 Basis-Platine effizient unterstützt.

6.1.1 Allgemeine Funktionsbeschreibung

In diesem Unterabschnitt wird zunächst eine allgemeine Funktionsbeschreibung des ATM-Diagnose-Monitors als zu entwerfendes eingebettetes System gegeben. Eine Einführung in die umfangreichen Details des ATM-Kommunikationsprotokolls kann hier nicht dargestellt werden und ist in [Gor95] zu finden. In Abbildung 6.1 wird der wesentliche Teil eines ATM-Netzwerks dargestellt.

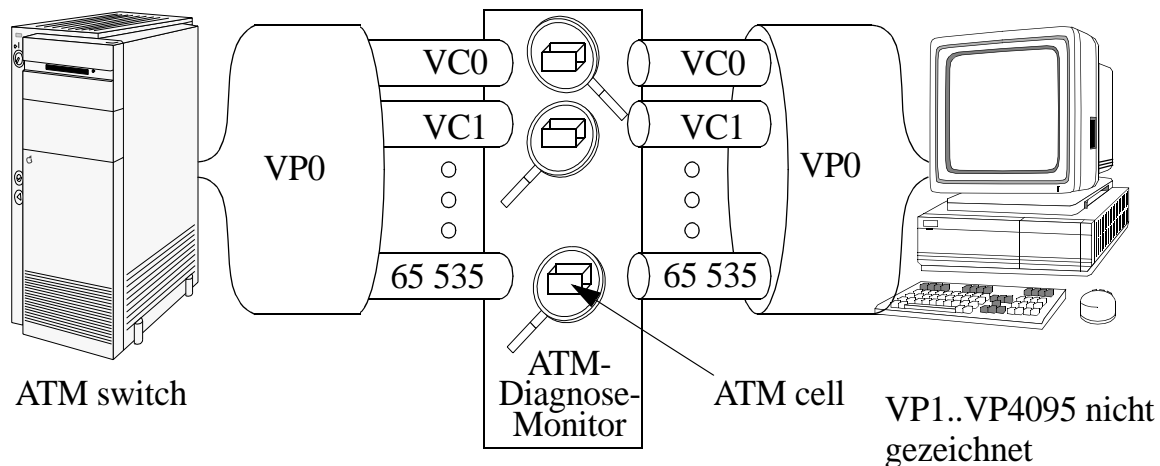


Abbildung 6.1 Allgemeine Funktion des ATM-Diagnose-Monitors

Ein PC, ausgerüstet mit einer ATM-Netzwerkkarte, ist über ein physikalisches Medium mit einem ATM-Switch verbunden. Dieses Medium kann entweder ein Coaxial-Kabel oder eine Glasfaserader sein. Die gesamte Kommunikationsbandbreite ist unterteilt in bis zu 4096 sogenannte virtuelle Pfade (engl.: *Virtual Path* - $VP0..VP4095$). Jeder VP ist seinerseits wieder unterteilt in bis zu 4096 sogenannte virtuelle Kanäle (engl.: *Virtual Channels* - $VC0..VC4095$). Die Übertragung von Information ist Bit-seriell. Die heutige Baudrate liegt bei 155 MBit/sec und wird in naher Zukunft auf 622 MBit/sec steigen. Eine ATM-Zelle besteht aus 53 Bytes, wobei 5 Bytes für den Zellkopf reserviert sind. Die restlichen 48 Bytes beinhalten die Nutzinformationen. Jede Zelle hat einen bestimmten VP und VC , welcher mit jeweils 12 Bit zusammen mit weiteren Verkehrsinformations-Bits im Zellkopf codiert ist.

Die Vergrößerungsgläser in Abbildung 6.1 symbolisieren die Hauptaufgabe des ATM-Diagnose-Monitors. In laufenden Netzwerken müssen Entwickler fehlerhafte Zellen erkennen und analysieren. Der erste Schritt ist die Beobachtung aller VP und ihrer untergeordneten VC , um dabei fehlerhafte Zellen zu erkennen. Wenn fehlerhafte ATM-Zellen erkannt worden sind, muß deren Anzahl in Abhängigkeit des jeweiligen VP/VC gezählt werden. Fehlerhafte ATM-Zellen erscheinen nur sporadisch auf zufälligen VP/VC -Nummern. Daraus folgt, der ATM-Diagnose-Monitor muß den gesamten ATM-Informationsverkehr auf allen VP/VC gleichzeitig über einen längeren Zeitraum überwachen und analysieren.

Der zweite Schritt ist dann die Analyse der fehlerhaften ATM-Zellen, um einen Hinweis auf die Ursache der Fehler zu bekommen, z.B. mögliche *CRC*-Fehler oder Paket-Längenfehler.

Diese allgemeine funktionale Beschreibung des zu entwickelnden eingebetteten Systems gibt einen Überblick über die Funktionalität und soll als Kurzbeschreibung einer Spezifikation dienen. Eine tiefergehende Darstellung ist in der Diplomarbeit [Kist97] zu finden, welche im Rahmen dieser Arbeit angefertigt wurde. Diese Arbeit wurde zusätzlich benutzt, um die vorge-

stellte Entwurfsmethodik in Kapitel 4 und die dafür entwickelten Werkzeuge in Kapitel 5 zu validieren.

6.1.2 Initiale Hardware/Software-Partitionierung

Um die im letzten Abschnitt beschriebene Spezifikation in eine eingebettete System-Architektur umzusetzen, muß eine initiale Partitionierung der Gesamtfunktionalität in entsprechende Teilfunktionen durchgeführt werden. Diese Architektur besteht dabei in Übereinstimmung mit dem beschriebenen Entwurfsraum in Abschnitt 4.1 aus wenigen, dafür aber hochintegrierten Komponenten. Dieser Schritt wird in der vorgestellten Entwurfsmethode vom Entwickler durchgeführt. Im wesentlichen werden dabei drei Grundkomponenten benötigt:

- **ASIC (engl.: *Adaption Layer Controller - ALC*):** Ein auf die dedizierte Aufgabe des Empfangens und Sendens von ATM-Zellen (engl.: *Segmentation And Reassembly - SAR*) spezialisierter *ALC-ASIC* soll eingesetzt werden, um den technisch sehr anspruchsvollen Empfangsprozess von ATM-Zellen bei einer seriellen Baudrate von 155 MBit/sec durchzuführen. Dieser Chip empfängt ATM-Zellen auf allen *VP*, konvertiert sie in Benutzerdaten zur Weiterverarbeitung und führt eine Fehlerüberprüfung in Echtzeit durch. Diese *ASIC*-Komponente speichert die empfangenen und in einem kontinuierlichen Datensatz zusammengesetzten (engl.: *reassemble*) ATM-Zellen in einem gemeinsamen (engl.: *shared memory*) Pufferspeicher zwischen dem *ALC* und dem Mikrocontroller. Zusätzlich speichert der *ALC* alle zell-spezifischen Informationen wie *VP* und *VC* und signalisiert mögliche Fehlerzustände einer Zelle, damit eine weitere Funktionseinheit diese Fehlerinformation zählen und weiter analysieren kann.

Zur Zeit sind viele komplexe *ALC-ASIC*-Komponenten auf dem Markt verfügbar. Ein wesentlicher Grundsatz der in dieser Arbeit vorgestellten Methodik besteht darin, solche Komponenten vorrangig zu benutzen, da in ihnen bereits erhebliches Spezialwissen investiert wurde, welches unbedingt weiter ausgenutzt werden soll. Wesentlicher Punkt dabei ist die Auswahl einer solchen Komponente, um einen systematischen Architekturentwurf durchführen zu können. Dieses Anwendungsbeispiel ist gut geeignet, um den Auswahlprozeß, basierend auf der Bewertung in Kapitel 4, zu demonstrieren (siehe Abschnitt 6.1.3.1).

- **Mikrocontroller:** Der Mikrocontroller ist in seiner Funktion sehr flexibel, aber relativ langsam im Vergleich zu spezialisierter Hardware. Er wird benutzt, um eine spezielle Initialisierungs- und Steuerungsaufgabe zu realisieren. Zum Einsatz kommt der auf der Basis-Platine vorhandene *PPC403GA/GCX*. Er initialisiert nach dem System-Reset den *FPGA* und den *ALC-ASIC*. Zusätzlich initialisiert und kontrolliert er gemeinsame Datenstrukturen mit dem *ALC-ASIC* in dem gemeinsamen Pufferspeicher-Bereich.

Während der Laufzeit der Anwendung wird auf Grund extremer Echtzeitanforderungen seine gesamte Rechenleistung zur Kontrolle des permanenten Empfangsprozesses des *ALC-ASIC* benötigt.

- **FPGA:** Wenn eine fehlerhafte ATM-Zelle erkannt wurde, muß sie in Abhängigkeit ihrer virtuellen Pfad- und Kanalnummer (*VP/VC*) analysiert und gezählt werden. Dafür muß der ATM-Diagnose-Monitor entsprechende Fehlerzähler bereitstellen, wie in Abbildung 6.1 durch die Vergrößerungsgläser symbolisch dargestellt. Wenn der *ALC* fehlerhafte ATM-Zellen bei einer Empfangsgeschwindigkeit von 155 MBit/sec erkennt,

speichert er die *VP/VC*-Nummer sowie eine 4-Bit-Fehlererkennung im Pufferspeicher und signalisiert diesen Fehlerfall durch einen *Interrupt* an den Mikrocontroller. In extremen Fällen, wenn mehrere fehlerhafte Zellen direkt hintereinander empfangen werden, wird alle $2,73 \mu\text{s}$ ein *Interrupt* ausgelöst. Dieser Wert ist eine extreme Echtzeitbedingung (er dauert nur ca. 90 *PPC403GA* Taktzyklen bei 33 MHz). Dadurch ist auch der Einsatz eines *RTOS* unmöglich, da die *Interrupt*-Reaktionszeit z.B. von *VxWorks* um den Faktor zehn größer ist.

Diese Echtzeitbedingung ist zu anspruchsvoll für die vollständige Realisierung der benötigten Fehlerzähler-Funktionen in der Software auf dem Mikrocontroller, zusätzlich zu seinen bereits erwähnten Aufgaben während der Laufzeit. Beim Eintreffen eines *Interrupts* löst der Mikrocontroller während der Kontrolle des Empfangsprozesses nur einen *Trigger*-Zugriff auf eine spezielle Hardware aus, welche die entsprechenden Fehlerzähler in Hardware realisiert. Als Zieltechnologie wird für diese Hardware, welche nicht als verfügbare *ASIC*-Komponente auf dem Markt verfügbar ist, ein *FPGA* verwendet. Für diesen Schritt muß bewertet werden, welche *FPGA*-Architektur für diese Art der Anwendung am besten geeignet ist. Auch diese Frage wird durch Einsatz der Emulationsumgebung *SPYDER-CORE-P1* beantwortet (siehe Abschnitt 6.1.4.2.).

6.1.3 Erste Stufe: Architekturentwurf durch Bewertung

Auf dem Markt sind zur Zeit viele *ASIC*-Bausteine von verschiedenen Herstellern verfügbar, welche prinzipiell die Funktionalität eines *ALC* bieten. Die erste Anforderung an den Entwickler ist zu entscheiden, ob eine oder mehrere *ASIC*-Komponenten die rein funktionalen Anforderungen der zu realisierenden Teilfunktion erfüllt. Wenn mehrere *ASIC*-Komponenten zur Auswahl stehen, ist zunächst eine zu bestimmen, welche sich auf Grund objektiver Kriterien besonders gut eignet. Die Frage nach der besten Eignung kann dabei anhand der Bewertung der bereits eingeführten Entscheidungsparameter für die einzelnen Entscheidungsfelder getroffen werden.

6.1.3.1 Auswahl einer *ASIC*-Komponente

Dieser Schritt ist in der heutigen Praxis ein zeitaufwendiger Prozeß, da die Entwickler im wesentlichen eine manuelle Recherche bzg. der auf dem Markt verfügbaren *ASIC*-Komponenten durchführen müssen. In der Regel werden im Internet die Benutzerhandbücher der einzelnen *ASIC*-Komponenten gesucht und analysiert. Anhand dieses Beispiels kann die Vorgehensweise des in dieser Arbeit vorgestellten Architekturentwurfs durch Bewertung demonstriert werden.

Zunächst kann man aus der Spezifikation eine gesuchte Teilfunktion F ableiten:

- $F = \{\text{ATM; AAL3/4+5, 155 MBit/sec, Full duplex, 53 Byte transparent, min. 1K offene VC}\}$

Dabei besteht (F) aus dem Schlagwort $S_{spec} = \text{ATM}$ und aus einem nachfolgenden Satz an funktionsbeschreibenden Schlüsselwörtern f_{ispec} , welche alle zur Erfüllung von (F) gefordert werden und im jeweiligen Einsatzspektrum in ihrer Bedeutung wohl definiert sind (siehe [Gor95]).

In Abbildung 6.2 wird das in Abbildung 4.10 eingeführte allgemeine Modell verfügbarer *ASIC* am Beispiel von *ALC*-Bausteinen für ATM verdeutlicht. Wendet man den Vergleich aus

Abschnitt 4.3.1 auf dieses Modell an, so ergibt sich zunächst in der Hauptgruppe h_i = Kommunikation die Übereinstimmung der Schlagwörter für die Untergruppe $S = S_{spec} = \text{ATM}$. Der Vergleich der Schlüsselwörter f_{ispec} mit den f_i der in dieser Untergruppe ATM vorhandenen ASIC-Komponenten führt auf insgesamt drei ALC-ASIC-Komponenten. Diese Untergruppe enthält einen MB86687A-ALC von Fujitsu, einen IDT77252-ALC von Integrated Device Technology und einen TNETA1560-ALC von Texas Instruments. Dabei wurden die Schlüsselwörter f_i in Abbildung 6.2 aus den jeweiligen Benutzerhandbüchern entnommen.

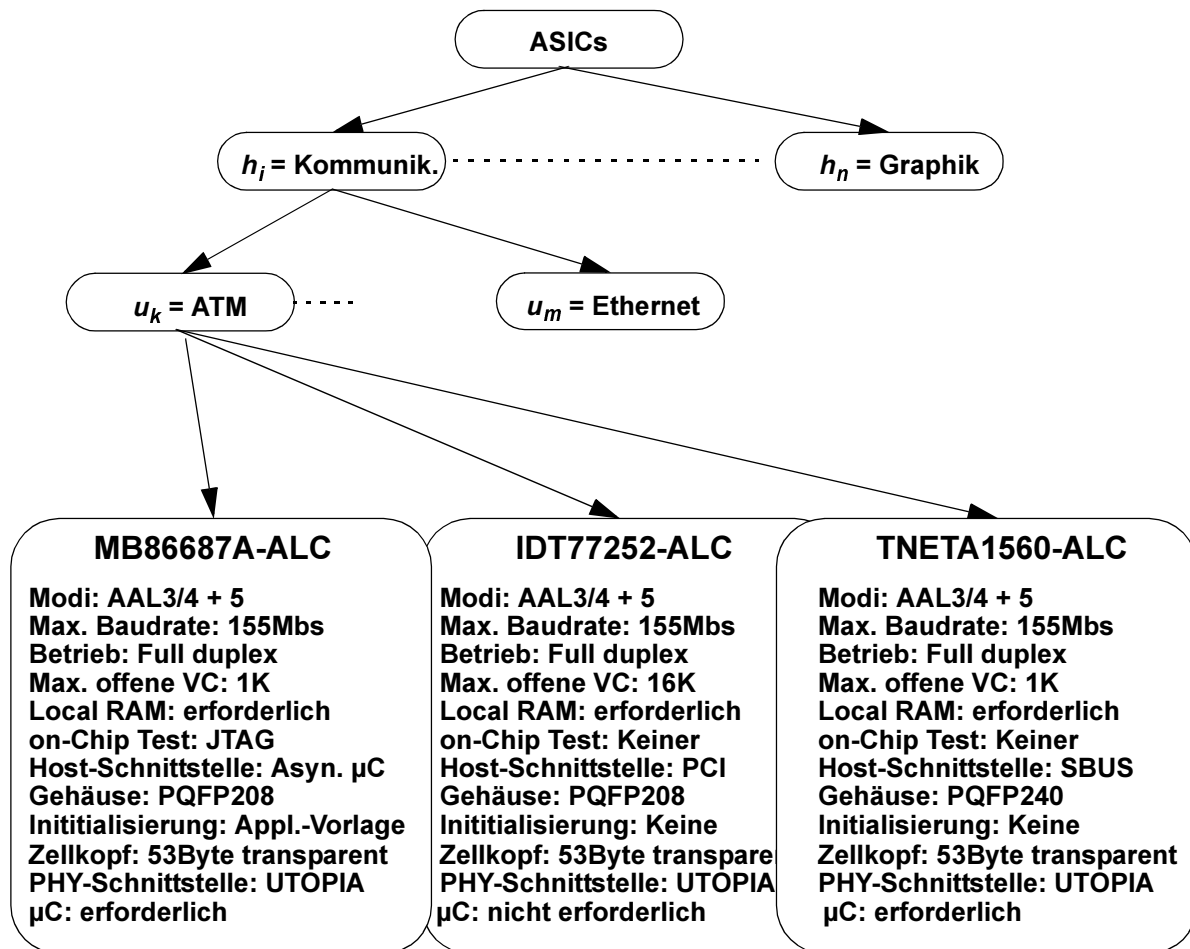


Abbildung 6.2 Modell der Menge verfügbarer ASIC, verdeutlicht am Beispiel des ALC für ATM

Durch den direkten Vergleich der geforderten Schlüsselwörter f_{ispec} aus F mit den Schlüsselwörtern der Komponenten f_i ergibt sich die Menge der Alternativen A . Eine Komponente ist dann eine Alternative A , wenn sie alle geforderten Schlüsselwörter f_{ispec} enthält. Im obigen Beispiel besteht die Menge der Alternativen deshalb aus $A = \{\text{MB86687A}, \text{TNETA1560}\}$. Der IDT77252-ALC kann nicht aufgenommen werden, da das Schlüsselwort „53 Byte transparent“ nicht vorhanden ist, d.h. die VP/VC-Nummer der empfangenen ATM-Zelle ist außerhalb der Komponente nicht mehr sichtbar. Diese Information wird aber gebraucht, um die Fehleranalyse bezüglich der verschiedenen VP/VC selektiv durchführen zu können.

Da die Menge A nicht aus der leeren Menge besteht, kann zunächst der Entscheidungsparameter im Auswahlalgorithmus in Abbildung 4.4 mit $\langle ja \rangle$ beantwortet werden. Welche der beiden Alternativen in A nun in die endgültige Architektur des eingebetteten Systems aufgenommen wird, hängt von der Bewertung der in Abschnitt 4.3 eingeführten Entscheidungsfelder

ab. Im folgenden Text wird diese Auswahlentscheidung, basierend auf der Bewertung der Entscheidungsparameter, zwischen dem MB86687A und dem TNETA1560 beschrieben.

- **Entscheidungsfeld Bus-Schnittstelle:** Wie in Abschnitt 4.3.2 dargestellt, hängt dieses Entscheidungsfeld von vielen einzelnen logischen und zeitlichen Entscheidungsparametern ab, die hier nicht alle diskutiert werden können. Es werden daher nur die wichtigsten Parameter betrachtet. In dieser Arbeit wird von einem Mikrocontrollergetriebenen Ansatz ausgegangen, d.h. er gibt das Schnittstellenverhalten vor. In diesem Fall wird ein PPC403GA eingesetzt. Er verfügt über einen asynchronen Mikrocontroller-Bus mit der Fähigkeit „*dynamic bus sizing*“ (siehe Abschnitt 2.1.1.3).

Der MB86687A besitzt zwei Schnittstellen. Eine 16-Bit breite Registerschnittstelle für seine 60 *on-Chip*-Register und eine 32-Bit breite Speicherschnittstelle als Verbindung zu einem ebenfalls noch benötigten Pufferspeicher. Das asynchrone Busverhalten kann vom *PPC403* auf Grund seiner programmierbaren Flexibilität direkt ohne Koppellogik angepaßt werden. Für die beiderseitige Zugriffsregelung auf den Pufferspeicher ist für die Steuersignale zusätzlich ein einfacher *Arbiter* in Form einer Zustandsmaschine in einem *FPGA* oder *CPLD* notwendig. Benutzt man die Bewertung in Tabelle 4.1, so liegt der Fall „Semantik der Steuersignale anpassen“ vor, welcher mit einem kleinen (K) Aufwand bewertet wird.

Der TNETA1560 besitzt ein sogenanntes *SBUS-Interface*, welches in Workstations benutzt wird. Die Anpassung an den *PPC403GA* ist grundsätzlich möglich, erfordert allerdings umfangreiche Koppellogik in Form eines *Bridge-Chips*. Dieser Fall benötigt den Einsatz eines speziellen Schnittstellen-*ASIC* aus dem Bereich der allgemeinen Rechnertechnik, welcher einen asynchronen Mikrocontroller-Bus auf den *SBUS*-Standard umsetzt. Legt man auch hier die Bewertungskriterien der Tabelle 4.1 an, liegt der Fall „paralleler Standard-Bus“ vor, welcher vom Mikrocontroller nicht unterstützt wird. Der Aufwand für diesen Fall wird mit „groß“ (G) klassifiziert.

Betrachtet man dieses Entscheidungsfeld für sich, so ist eindeutig der MB86687A zu bevorzugen, da der zusätzliche Aufwand wesentlich geringer ist als beim TNETA1560.

- **Entscheidungsfeld Initialisierung:** Für beide *ASIC* sind keine Initialisierungsroutinen (z.B. in C) für einen Mikrocontroller bzw. eine entsprechende Entwicklungsumgebung für die Firmware verfügbar.

Für den MB86687A werden vom Hersteller eine Text-Vorlage zur Entwicklung einer Initialisierungsroutine sowie ein Beispiel für die Organisation der gemeinsamen Datenstruktur im Pufferspeicher zum Datenaustausch zwischen dem *ALC* und dem Mikrocontroller zur Verfügung gestellt. Applikations-Spezialisten stellt er aber nicht zur Verfügung. Legt man die Bewertung von Tabelle 4.2 an, so wird der Initialisierungsaufwand für diesen *ASIC* mit „groß“ (G) klassifiziert.

Für den TNETA1560 werden zur Zeit keine entsprechenden Hilfen angeboten. Benutzt man die Bewertung nach Tabelle 4.2, so muß für diesen Baustein sogar die Einstufung „nicht verwendbar“ (NV) gegeben werden, da das Entwicklungsrisiko für diesen Baustein als unkalkulierbar angesehen wird.

Betrachtet man dieses Entscheidungsfeld für sich, so bietet auch hier der MB86678A die bessere Unterstützung an. Der TNETA1560 ist nicht verwendbar.

- **Entscheidungsfeld Technologie:** Beide *ALC* sind in einem *PQFP*-Gehäuse mit 208 Pins beim *MB86687A-ALC* und 240 Pins beim *TNETA1560-ALC* mit Pin-Abständen von 0,5 mm. Beide Chips stellen damit fast identische Anforderungen an das Platinen-*Layout*. Für das Projekt wird für die Technologie-Anforderung in der Spezifikation die Kategorie (**F** = Feinleitertechnik) vorgegeben. Aus dem Diagramm in Abbildung 4.13 ist ersichtlich, daß zwischen vier und sechs Lagen gebraucht werden, um die Bausteine effizient zu verdrahten. Diese Technik gehört heute zum Standard und stellt keine besonderen Anforderungen an die Fertigung. Nach Tabelle 4.3 werden die beiden Bausteine mit einem mittleren (**M**) Implementierungsaufwand klassifiziert

Betrachtet man dieses Entscheidungsfeld für sich, so sind beide *ALC-ASIC* gleichmäßig gut zu beurteilen.

- **Entscheidungsfeld Testbarkeit:** Da es sich um eine erste Prototyp-Entwicklung handelt, wird von den Entwicklern die höchste Testbarkeitsstufe in der Kategorie **E** verlangt. Der *MB86687A* sowie der *TNETA1560* verfügen über keine Testunterstützung auf dem Chip. Daher muß bei seinem Einsatz während der Entwicklung eine externe Testunterstützung geschaffen werden, z.B. durch Test-Pins für alle Signale des Chips. Benutzt man die Bewertung nach Tabelle 4.4, so ist auch der Aufwand in diesem Entscheidungsfeld für beide Bausteine gleich mit „groß“ (**G**) zu bewerten.

Um eine Auswahlentscheidung für eine der beiden möglichen Alternativen zu treffen, müssen die einzelnen Entscheidungsfelder in einer Gesamtbeurteilung zusammen genommen werden. Dieser Schritt wird in Tabelle 6.1 durchgeführt. Der *IDT77252* scheidet schon bei der

Alternative <i>ALC-ASIC</i>	<i>MB86687A</i>	<i>TNETA1560</i>	<i>IDT77252</i>
Funktionalität	ja	ja	nein (NV)
Bus-Schnittstelle	K	G	---
Initialisierung	G	NV	---
Bewertung: Logischer Pfad	G	NV(Ausschluß)	---
Technologie	M	M	---
Testbarkeit	G	G	---
Bewertung: Implementierungsspezifischer Pfad	G	G	---

Tabelle 6.1: Gesamtübersicht der bewerteten *ALC-ASIC*-Komponenten

funktionalen Betrachtung aus. Der *TNETA1560* scheidet auf Grund seiner unkalkulierbaren Risiken bei der Initialisierung aus. Die beiden Pfade der logischen und implementierungsspezifischen Bewertung können nur beim *MB86687A* ohne Ausschlußkriterium durchlaufen werden. Dabei wurde die Gesamtbewertung der beiden Pfade mit Hilfe der Tabelle 4.6 durchgeführt.

Obwohl die Bewertung der einzelnen Entscheidungsfelder von durchaus objektiven Entscheidungsparametern abhängt, ist die Gesamtbeurteilung sicher von subjektiven Einschätzungen des Entwicklers abhängig, je nach der Gewichtung der einzelnen Entscheidungsfelder. In diesem hier diskutierten Beispiel ist die Auswahl allerdings relativ eindeutig und fällt auf den *MB86687A*. Das bedeutet jedoch noch nicht, daß er endgültig in die Architektur des eingebetteten Systems aufgenommen wird. Dieser Schritt wird erst durchgeführt, wenn alle Bewertungen, welche zu seiner Auswahl geführt haben, im zweiten Schritt durch Emulation mit dem *SPYDER*-System überprüft wurden (siehe Abschnitt 6.1.4.1). Insbesondere das Entscheidungs-

feld Initialisierung sowie die implementierungsspezifischen Aspekte stellen durchaus noch ein signifikantes Entwicklungsrisiko dar. Insbesondere die dazu notwendige Firmware eines komplexen *ASIC* benötigt einen enormen Entwicklungsaufwand. Dennoch wird die Nutzung dieser Komponente durch Emulation überprüft. Erst wenn diese Überprüfung negativ endet, würde eine eigene Hardware-Entwicklung in Erwägung gezogen, da sie einen noch deutlich höheren Entwicklungsaufwand bedeuten würde.

6.1.3.2 Bewertung verschiedener Entwicklungsmethoden für *FPGA*

Im Abschnitt 6.1.2 wurde bereits eingeführt, daß zur Analyse der fehlerhaften ATM-Zellen eine spezielle Hardware zusätzlich zum *ALC* im *FPGA* benötigt wird, weil diese Funktionalität als *ASIC* nicht verfügbar ist. Als Zieltechnologie für diesen Eigenentwurf eignen sich *FPGA*. Während dieses Entwurfsschrittes werden verschiedene *FPGA*-Architekturen bezüglich ihrer Eignung für diese spezielle Anwendung untersucht.

Dieses Anwendungsbeispiel wird benutzt, um die in der Industrie weitverbreitete XC4000 Architektur mit der in der wissenschaftlichen Diskussion stehenden XC6000 Architektur zu vergleichen, welche beide in Abschnitt 2.1.3.2 beschrieben wurden. Eng mit den verschiedenen *SRAM*-basierten *FPGA*-Architekturen sind unterschiedliche Entwurfsmethoden (siehe Abschnitt 2.1.3.4) verbunden, deren spezieller Einfluß ebenfalls analysiert wird.

Um diese Abhängigkeiten aufzuzeigen, wird eine Analyse in zwei Schritten durchgeführt. Im ersten Schritt wird die am weitesten verbreitete *CTR*-Methode benutzt und auf die beiden *FPGA*-Zielarchitekturen XC4000 und XC6000 angewendet.

Im zweiten Schritt bleibt dann die Zielarchitektur XC6000 gleich, wobei die Entwurfsmethode von *CTR* auf lokale-*LTR* geändert wird. Diese Untersuchungen werden ebenfalls durch Emulation auf dem SPYDER-System durchgeführt und sind in Abschnitt 6.1.4.2 dargestellt.

6.1.4 Zweite Stufe: Überprüfung des *ALC-ASIC* durch Emulation

Dieser Abschnitt befaßt sich mit der Durchführung der Emulation. Wesentliches Ziel dabei ist es, das eingebettete System unter möglichst identischen Echtzeitbedingungen im Vergleich zum eigentlichen Zielsystem zu emulieren, ohne wesentliche Änderungen. Ein weiterer Aspekt ist das Bestreben, auch die reale Umgebung mit in die Emulation einzubeziehen, um einen detaillierten Einblick in das interne Systemverhalten zu bekommen. Diese Zielsetzung wird ermöglicht durch den Einsatz der Emulationsumgebung SPYDER, welche in Kapitel 5 bereits eingeführt wurde.

6.1.4.1 *ASIC*-Emulation mit SPYDER-*ASIC-X2*

Ziel dieses zweiten Schrittes ist die Überprüfung der vorausgesetzten Annahmen bei der Bewertung der einzelnen Entscheidungsparameter innerhalb der Entscheidungsfelder durch Emulation. Voraussetzung dafür ist die Fertigung einer entsprechenden Trägerplatine, welche den *ALC-ASIC* mit dem Emulationswerkzeug SPYDER-*ASIC-X2* verbindet, wie in Abbildung 6.3 und Abbildung 6.4 dargestellt. Diese Trägerplatine stellt zunächst einen gewissen Aufwand dar, welcher aber insbesondere durch die erreichbare Effizienz und Geschwindigkeit im Bereich der Entwicklungszeit und der Minimierung des Entwicklungsrisikos gerechtfertigt ist. Dieser Aspekt wird am Ende dieses Abschnittes verdeutlicht.

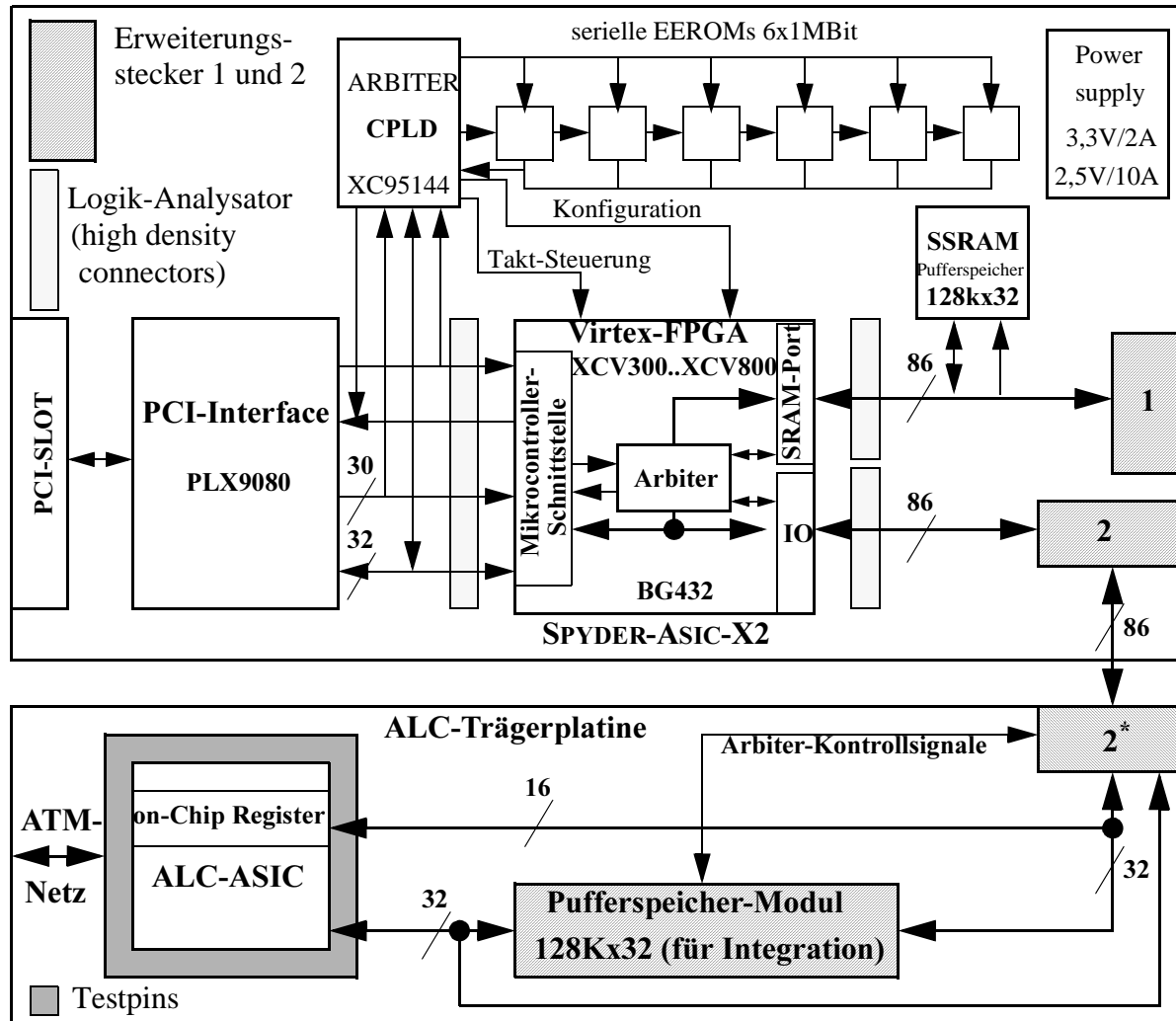


Abbildung 6.3 Blockdiagramm des Emulationswerkzeugs SPYDER-ASIC-X2 mit ALC-Trägerplatine

Die Trägerplatine hat die hauptsächliche Aufgabe, den *ALC-ASIC* der Emulationsumgebung physikalisch zugänglich zu machen. Zusätzlich ist noch, wie in Abbildung 6.3 dargestellt, der Pufferspeicher als gemeinsames Medium zum Datenaustausch mit dem Mikrocontroller implementiert, welcher ein *ALC-ASIC* zum effizienten Betrieb unbedingt braucht. Diese Maßnahme dient bereits der späteren Systemintegration (siehe Abschnitt 6.1.4.3) mit dem Werkzeug SPYDER-CORE-P1. Für die reine Emulation des *ALC-ASIC* mit SPYDER-ASIC-X2 könnte auch ersatzweise der *SSRAM*-Speicherblock auf dem Emulationswerkzeug selbst verwendet werden.

In Abbildung 6.4 wird der Emulationsaufbau des Blockdiagramms in Abbildung 6.3 vereinfacht dargestellt, insbesondere um die Struktur der Wirkungskette während der Emulation des *ALC-ASIC* hervorzuheben. Verfolgt man diese Wirkungskette von links nach rechts, so wird zunächst ein Entwicklungsrechner (PC) über den *PCI*-Bus, welcher bereits zum Standard im Bereich der allgemeinen Rechnertechnik geworden ist, mit dem Emulationswerkzeug SPYDER-ASIC-X2 verbunden. Der *PCI*-Chip, ein dedizierter *PCI-ASIC*, setzt den *PCI*-Bus in einen asynchronen Mikrocontroller-Bus (*PCI-2- μ C*) um, welcher dem logischen und zeitlichen Verhalten eines realen Mikrocontrollers weitestgehend entspricht. Im *FPGA* wird die Schnittstellen-Anpassung zwischen dem Mikrocontroller und dem *ALC-ASIC* durchgeführt. Das Werkzeug

schließt mit dem Erweiterungsstecker 2 ab, welcher die Verbindung mit der Trägerplatine des *ALC-ASIC* herstellt. Dieser Aufbau wird benutzt, um die einzelnen Entscheidungsfelder durch Emulation zu überprüfen, wie im folgenden Text beschrieben wird:

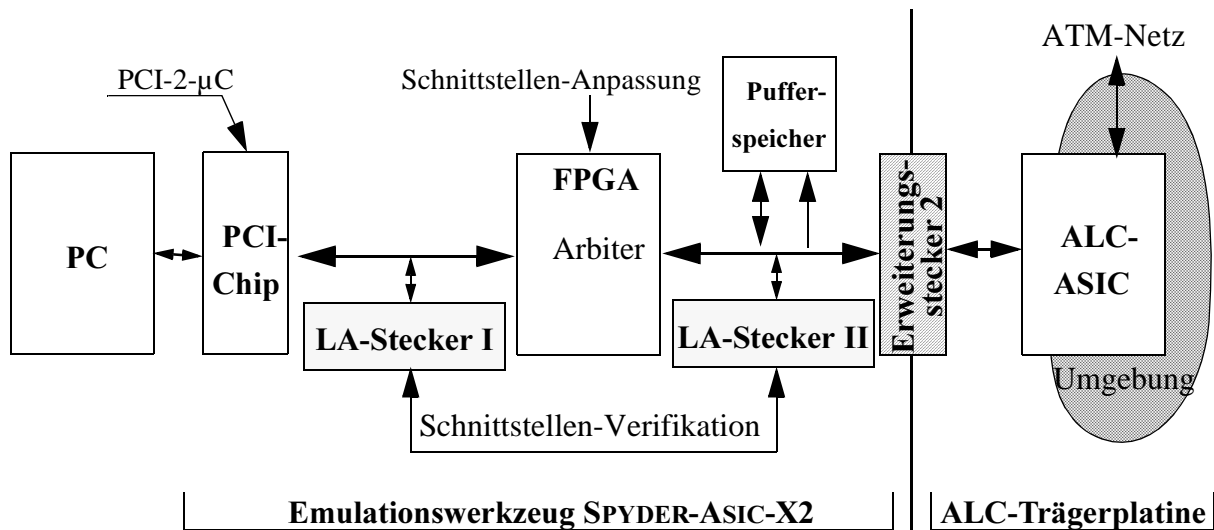


Abbildung 6.4 Vereinfachte Darstellung der Wirkungskette bei der Emulation

- **Entscheidungsfeld Bus-Schnittstelle:** Zur Bus-Schnittstelle gehört ein 16-Bit-Datenpfad zwischen Mikrocontroller und *ALC-ASIC*, welcher für den Schreib- und Lesezugriff für die 60 *on-Chip*-Register des *ALC* benutzt wird. Dieser Datenpfad wird im wesentlichen durch das Schnittstellen-*FPGA* ohne besondere Logik verdrahtet.

Zusätzlich wird ein 32-Bit-Datenpfad mit einem 128 K-Adressraum vom Mikrocontroller zum Pufferspeicher bzw. vom *ALC-ASIC* zum Pufferspeicher benötigt. Der beiderseitige Zugriff über den *SRAM-Port* (engl.: *single port read/write*) muß über einen *Arbitrier* gesteuert werden. Dieser *Arbitrier* besteht im wesentlichen aus einer Zustandsmaschine, welche die Zugriffsanforderungen von beiden Seiten als Eingangssignale entgegennimmt und die entsprechenden Aktivierungssignale für die sogenannten *Tri-state*-Bustreiber generiert, um den Speicherbus der jeweils selektierten Seite auf den *SRAM-Port* aufzuschalten, damit ein Lese- bzw. Schreibzugriff durchgeführt werden kann.

Entscheidend dabei ist, daß die gesamte oben beschriebene Bus-Schnittstellen-Funktionalität zunächst vollständig im Schnittstellen-*FPGA* mit SPYDER-ASIC-X2 emuliert werden kann (siehe Blockdiagramm Abbildung 6.3). Durch die LA-Stecker I und II kann das gesamte Schnittstellenverhalten meßtechnisch unter Echtzeitbedingungen verifiziert werden (siehe Abbildung 6.4). Auf Grund der sehr hohen Flexibilität des *SRAM*-basierten Schnittstellen-*FPGA* kann ein evtl. festgestelltes Fehlverhalten in kurzer Zeit beseitigt werden.

Nach der fehlerfreien Emulation des kompletten Schnittstellen-Verhaltens wird der eigentliche Pufferspeicher auf der Trägerplatine ausgelagert. Der mit Pufferspeicher-Modul gekennzeichnete Teil des Blockdiagramms enthält einen 128 K x 32-Bit-*SRAM*-Speicher und die notwendigen Bustreiber zur Entkoppelung. Der *SRAM*-Speicher und die Bus-Entkoppelung können dadurch nach erfolgreichem Test von dem Emulationswerkzeug ebenfalls auf die Trägerplatine verlagert werden. Damit steht der *ALC* zusammen mit allen notwendigen Datenpfaden auf der Trägerplatine zur Verfü-

gung. Die *Arbiter*-Zustandsmaschine bleibt im Schnittstellen-*FPGA*. Dieser Auslagevorgang ist nur für die Systemintegration und den anschließenden Systemtest notwendig, welche in Abbildung 6.1.4.3 beschrieben werden.

- **Entscheidungsfeld Technologie:** Die Emulationsumgebung SPYDER steht als getestete und wiederverwendbare Komponente zur Verfügung, bei der die Wahrscheinlichkeit technologiebedingter Fehler seitens des Werkzeugs sehr gering ist.

Die einzige technologiebedingte Fehlerquelle besteht in der anwendungsspezifischen Trägerplatine. Damit wird erreicht, daß technologische Fehler auf den eigentlichen Probanden während der Emulation begrenzt werden. In diesem Beispiel handelt es sich um eine vier-Lagen-Platine (Kategorie **F**) mit einem *QFP208-Chip* (Pin-Abstand 0,5 mm). Diese Anforderungen sind auf einer Trägerplatine, bei der die Bauteildichte nicht besonders hoch sein muß und die Lagenanzahl niedrig (<6) ist, sicher zu beherrschen.

Bei der in diesem Anwendungsbeispiel eingesetzten Trägerplatine muß eigentlich nur die fehlerfreie Verdrahtung des *ALC-ASIC* mit dem Erweiterungsstecker 2 gewährleistet sein, um die Emulation durchführen zu können, da das Pufferspeicher-Modul bei der singulären Komponenten-bezogenen Emulation nicht gebraucht wird.

Diese Vorgehensweise wird um so wichtiger bei der Emulation von Komponenten, die ein *Ball-Grid*-Gehäuse mit bis zu 600 Pins und über zehn Lagen brauchen. Hier wird deutlich, wie wichtig es ist, durch technologische Verfahren bedingte Fehlerquellen während des Entwicklungsablaufs auf kleine Trägerplatinen zu begrenzen, die sich nicht auf das komplette zu emulierende System auswirken können.

- **Entscheidungsfeld Testbarkeit:** Im Gegensatz zu den anderen Alternativen verfügt der zu emulierende *ALC MB86687A* über keine *on-Chip*-Testunterstützung. Bei der Entwicklung der Komponenten-spezifischen Trägerplatine wird die Testbarkeit durch Test-Pins (siehe auch Abbildung 5.4) zur Verfügung gestellt, die einen meßtechnischen Zugang zu allen Signal-Pins des *ALC-ASIC* erlauben. Damit hat der Entwickler auch die Möglichkeit, Analysen am laufenden System unter Echtzeitbedingungen durchzuführen. Diese Testunterstützung ist besonders bei der Systemintegration und beim Systemtest sehr wirkungsvoll, um versteckte Fehlerquellen im Gesamtsystem zu finden.
- **Entscheidungsfeld Initialisierung:** Die Abbildung 6.5 zeigt z.B. die zum Senden benötigte Datenstruktur, auf die in dem gemeinsamen Pufferspeicher von beiden Seiten zugegriffen wird. Vor Beginn des Betriebes muß diese Datenstruktur von einem *Host* initialisiert werden. Während des Betriebes regelt diese Datenstruktur die Kommunikation zwischen dem *Host* und dem *ALC*, wobei der *Host* diese Struktur permanent überwachen und entsprechend kontrollieren muß. Der *Transmit Descriptor* besteht aus einem 8 * 32-Bit-Feld und beinhaltet neben Zeigern auf die Nutzdaten und *Circuit Reference Tables* viele Einzel-Bit-Einträge. Diese steuern die Verarbeitung (engl.: *segmentation*) der Nutzdaten und die Fehlerbehandlung. Die *Circuit Reference Table* ist ein 4 * 32-Bit-Feld und enthält Verkehrsinformationen und Parameter, die im Zellkopf der ATM-Zelle weitergegeben werden. Die *Transmit Pending Queue* steuert die Verarbeitung mehrerer Pakete in einer Warteschlange und die *Transmit Buffer Release Queue* gibt bearbeitete Speicherbereiche wieder frei. Eine ähnliche Datenstruktur gibt es auch für den Empfangspfad. Zusätzlich müssen bis zu 60 Register vom *Host* direkt im *ALC-ASIC* programmiert werden. Eine tiefer gehende Erklärung kann hier aus

Platzgründen nicht gegeben werden und ist in [Fuj96] zu finden. An diesem Beispiel soll gezeigt werden, daß zum Betrieb des *ALC* eine sehr spezielle Firmware, bestehend aus komplexen Bit-Operationen, benötigt wird.

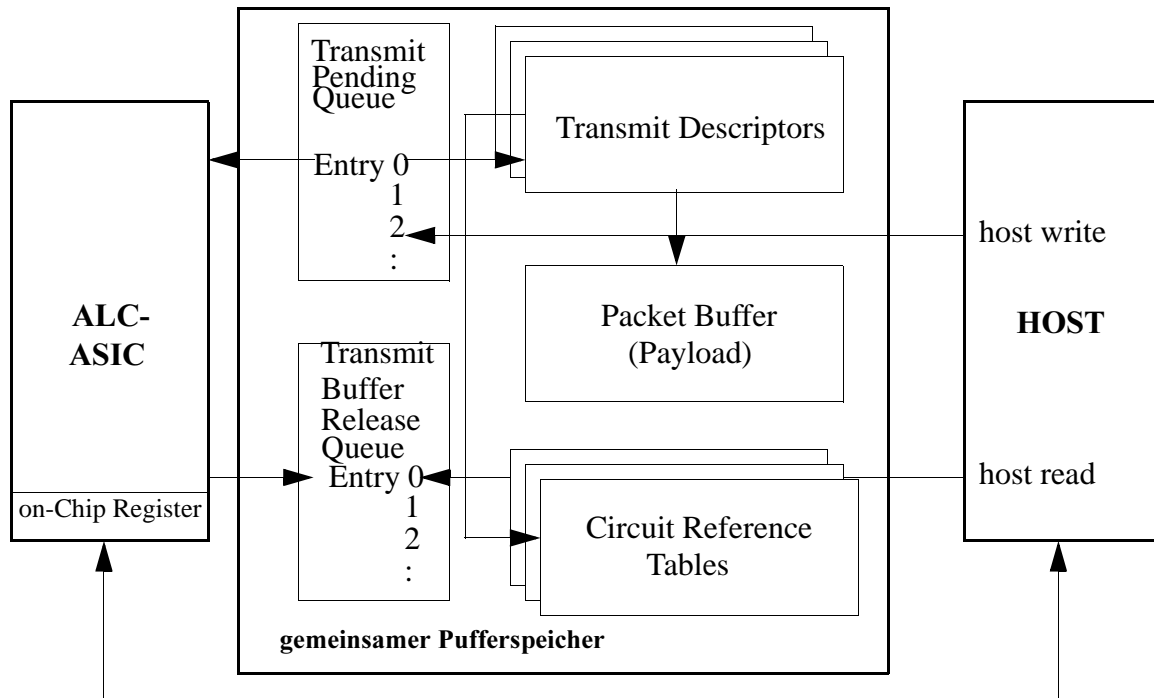


Abbildung 6.5 Transmit-Datenstruktur im gemeinsamen Pufferspeicher (Quelle: [Fuj96])

Der *ALC* liefert dabei keine geeignete Testunterstützung. Die vorhandene Vorlage des Herstellers muß in eine funktionsfähige Firmware umgesetzt werden.

Dabei kommt die fundamentale Eigenschaft des SPYDER-ASIC-X2-Werkzeugs zum Tragen. Der PC kann während der Emulation als *Host* arbeiten und den eigentlichen Mikrocontroller ersetzen. Diese Methode führt zu Vorteilen, die die Emulation effizient beschleunigen. Durch die Zusammenlegung von Entwicklungsrechner (PC) und Zielsystem entfallen alle Probleme, die eine Übertragung und Überwachung des auszuführenden Firmware-Codes auf einen Mikrocontroller in einem eingebetteten System beinhalten. Zusätzlich können die leistungsstarken Entwicklungswerkzeuge für den Entwicklungsrechner (PC) wie z.B. Compiler und *Debugger* genutzt werden. Zusätzlich kann die gesamte Standard-Peripherie des PC (Bildschirm, Tastatur, Festplatte) zur Visualisierung, Verifikation und Kontrolle verwendet werden.

In Kapitel 3 wurde bereits motiviert, daß die Implementierung dieser Firmware ein enormes Entwicklungsrisiko bei komplexen *ASIC* darstellt sowie mit einem signifikanten Entwicklungs- und Testaufwand verbunden ist. Die Adaption des Mikrocontrollers durch den PC während der Emulation stellt somit Software-Entwicklern eine Methode zur Verfügung, eine *ASIC*-Initialisierung zu bewerten, ohne daß sie über das restliche eingebettete System verfügen. Damit wird auch eine Bewertung von neuesten *ASIC*-Komponenten in einem sehr frühen Entwicklungsstadium eines Projektes ermöglicht, welche bei der zur Zeit in der Praxis eingesetzten Methode (siehe Abschnitt 3.2.1 und Abschnitt 6.3) nicht berücksichtigt werden können, da ihre Entwicklungsrisiken für die Entwickler unkalkulierbar sind.

Die Validierung der Methodik und des Werkzeugs wurde im Rahmen der Diplomarbeit [Kist97] durchgeführt. Die Ergebnisse wurden in [WeHeRo97] veröffentlicht. Dabei wurde der erste Prototyp von SPYDER-ASIC mit dem Suffix X0 eingesetzt. Das aktuelle Werkzeug SPYDER-ASIC-X2 ist nach X1 bereits die dritte Generation. Das Einsatzspektrum des Werkzeugs innerhalb des SPYDER-Emulationssystems ist dabei immer gleich geblieben, wobei die Qualität und Effizienz der Emulation drastisch verbessert wurde. Im wesentlichen existieren zwei technische Modifikationen zur ersten Version:

- Die erste Prototypen-Version basierte auf dem 8-Bit-*AT-ISA*-Bus als Verbindung zum PC, welcher durch Koppellogik im Schnittstellen-*FPGA* auf einem 32-Bit-Mikrocontroller-Bus umgesetzt wurde. Damit konnte nur das logische Verhalten während der Emulation nachgebildet werden, nicht aber das zeitliche Verhalten. Durch Übergang auf den *PCI*-Bus und dessen Umsetzung in einen lokalen Mikrocontroller-Bus durch den *PCI*-Chip (PLX9080) in Verbindung mit dem lokalen *Arbiter* (XC9572-*CPLD*, siehe Abbildung 5.6) kann auch das zeitliche Verhalten eines 32-Bit-Mikrocontrollers emuliert werden.
- Im Zuge der Weiterentwicklung wurden immer leistungsstärkere Schnittstellen-*FPGA* implementiert. So stieg die Gatter-Kapazität von der ersten Version mit 10.000 Gatter (XC4010E) auf ca. 800.000 Gatter (XCV800) bei der neuesten Virtex-Architektur von Xilinx (siehe Abschnitt 2.1.3.2). Die Durchlaufzeiten reduzierten sich dabei drastisch, was sich insbesondere bei der notwendigen Addition in Abbildung 4.5 positiv auswirkt. Damit profitierte das Werkzeug SPYDER-ASIC-X2 von der enormen Leistungssteigerung im Bereich der *FPGA*, was die universelle Einsetzbarkeit für viele verschiedene Anwendungen deutlich erhöhte (siehe Abschnitt 6.3). Zusätzliche Einsatzfelder sind beispielsweise die Emulation von großen synthetisierbaren digitalen Schaltungskernen (engl.: *Intellectual Property-IP-Core*) in einem Entwicklungsrechner. So befassen sich zur Zeit einige wissenschaftliche Arbeiten mit der Emulation von *JAVA*-Prozessoren mit Hilfe dieses Werkzeugs.

6.1.4.2 *FPGA*-Emulation mit SPYDER-CORE-P1

Um die fehlerhaften ATM-Zellen in Abhängigkeit ihrer *VP/VC*-Nummer in Echtzeit zu registrieren, werden sogenannte Fehlerzähler (FZ) benötigt. Sie bestehen aus einer 28-Bit-breiten Vergleichseinheit, einem 32-Bit-Inkrementierer, Koppellogik und verschiedenen *on-Chip*-Bussen. Diese Funktionalität dient als Benchmark zur Evaluierung verschiedener *FPGA*-Architekturen und den dazugehörigen Methoden, wie in Abschnitt 6.1.4.2 vorgestellt. Zur Emulation der einzelnen Benchmark-Schaltungen wird die SPYDER-CORE-P1 Basis-Platine benutzt, welche die beiden *FPGA*-Familien unterstützt.

Compile-Time Reconfiguration (CTR) - Methode

- ***CTR* angewendet auf XC4000E/EX-*FPGA*:** Der Mikrocontroller liest bei der Ausführung des Steuerungsalgorithmus die *VP/VC*-Nummer aus dem Pufferspeicher (daher die Forderung bei der *ALC*-Auswahl „53 Byte transparent“). Er signalisiert das Erscheinen von fehlerhaften ATM-Zellen durch gezielte Lesezugriffe auf eine bestimmte Adresse, welche von der zentralen Logik im *FPGA* überwacht wird. Die Architektur der digitalen Hardware im *FPGA* bei der Anwendung der *CTR*-Methode ist in Abbildung 6.6 dargestellt.

Um alle *VP/VC*-Nummern gleichzeitig zu überwachen, würden 2^{28} -Fehlerzähler benötigt werden. Dieser Aufwand ist mit verfügbaren Hardware-Ressourcen nicht realisierbar. Deswegen muß ein Mechanismus implementiert werden, welcher sich selektiv nur auf *VP/VC*-Nummern bezieht, welche fehlerhafte ATM-Zellen empfangen. Wenn zum ersten Mal eine fehlerhafte ATM-Zelle auf einer *VP/VC*-Nummer erkannt wird (kein aktives *feedback*-Signal), wird die zugehörige *VP/VC*-Nummer in das *VP+VC*-Register eingetragen. Dadurch wird ein Fehlerzähler zum Zählen aller weiteren Fehler auf dieser Nummer aktiviert. Die Vergleichseinheit vergleicht die aktuell anliegenden *VP/VC*-Nummern mit dem festgelegten Registerwert und generiert ein entsprechendes Freigabe-Signal für den 32-Bit-Inkrementierer.

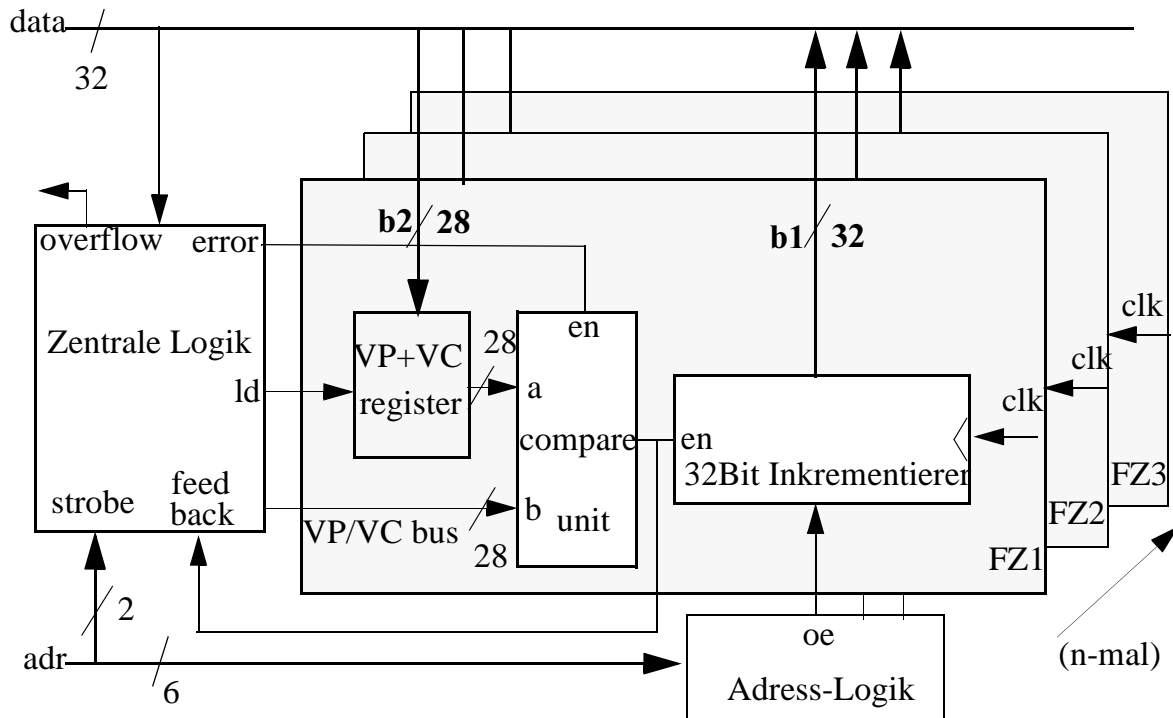


Abbildung 6.6 Fehlerzähler zur Anwendung der *CTR*-Methode bei *XC4000* und *XC6000-FPGA*

Eine spezielle Adress-Logik muß implementiert werden, um die Anzahl der erkannten Fehlerzellen auf einer *VP/VC*-Nummer nach einer bestimmten Überwachungszeit auszulesen. Jede beobachtete *VP/VC*-Nummer benötigt einen eigenen Fehlerzähler. Die zentrale Logik setzt ein *overflow*-Bit, wenn mehr *VP/VC*-Nummern mit fehlerhaften ATM-Zellen zu überwachen sind als freie Fehlerzähler zur Verfügung stehen.

- ***CTR* angewendet auf *XC6000-FPGA*:** Die *XC6000*-Architektur verfügt über eine Eigenschaft, welche die Architektur in Abbildung 6.6 signifikant vereinfacht. Auf Grund der Tatsache, daß alle Speicherelemente (engl.: *Flip-Flops* - *FF*) in dem Adressraum eines Mikrocontrollers eingebettet sind und dadurch frei gelesen und geschrieben werden können, werden die beiden *on-Chip*-Busse **b1** und **b2** nicht benötigt. Auch die Adress-Logik wird nicht notwendig. Wie am Ende dieses Abschnittes gezeigt wird, trägt diese Eigenschaft einen wesentlichen Anteil zur sehr guten Effizienz dieser *FPGA*-Architektur für diese Anwendung bei.

Die notwendige Vergleichseinheit besteht aus 28 *XOR*-Gattern, jedes verbunden mit einem *FF* des *VP-VC*-Registers und dem zugehörigen Bussignal, sowie 28 nachge-

schalteten *2-Input-AND*-Gattern. Wie im nächsten Unterabschnitt gezeigt, kann bei Änderung der Entwurfsmethode von *CTR* nach lokaler *RTR* eine weitere Vereinfachung erreicht werden.

Local Run Time Reconfiguration (Local RTR)-Methode

Diese Methode ist zwingend an eine partiell rekonfigurierbare *FPGA*-Architektur wie z.B. die *XC6000*-Familie gebunden. In Abbildung 6.7 ist dafür eine geeignete Architektur dargestellt.

- Diese Methode profitiert von der partiellen Rekonfiguration von kleineren Schaltungsteilen zur Laufzeit. Die Schaltungsvereinfachung basiert auf dem Effekt der Fortpflanzung von Konstanten innerhalb logischer Gleichungen (engl.: *constant propagation*), welche auch in [WirHut97] beschrieben ist.

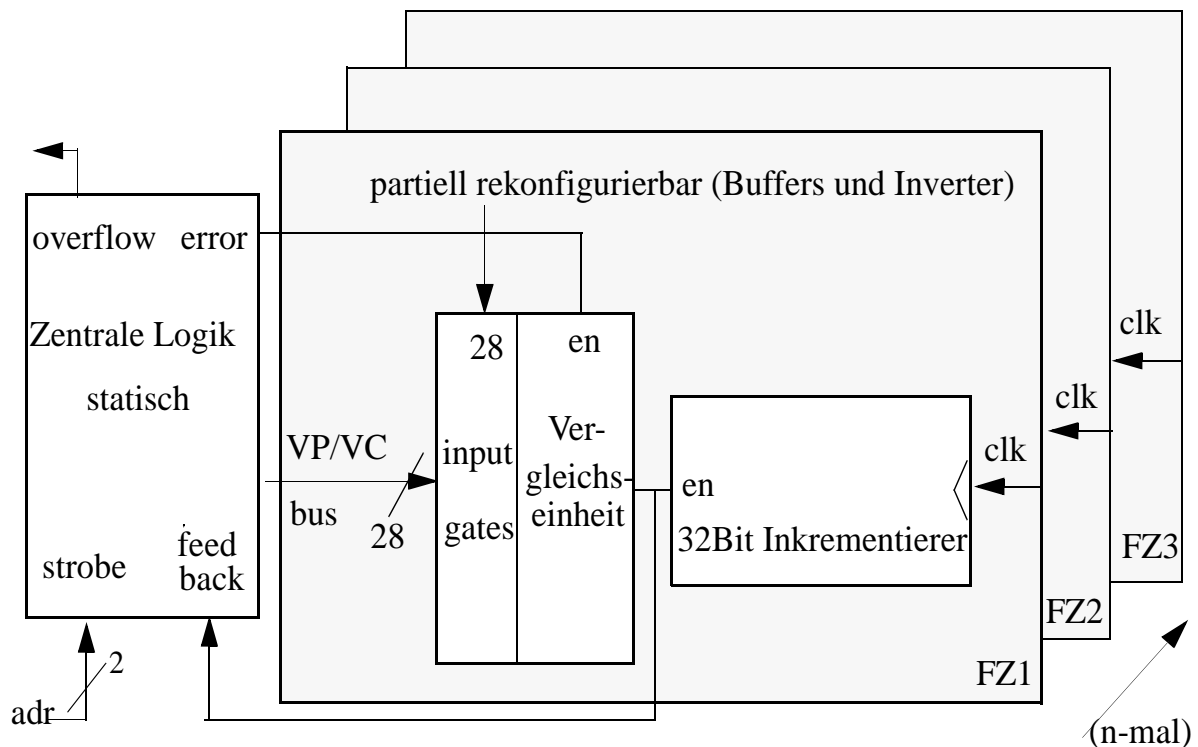


Abbildung 6.7 Fehlerzähler bei der Anwendung der lokalen *RTR*-Methode bei *XC6000-FPGA*

Dabei wird die aktuelle *VP/VC*-Nummer mit einer Konstanten verglichen, wobei sich die Vergleichseinheit weiter drastisch vereinfacht. Sie besteht aus 28 partiell rekonfigurierbaren *Input-Gates*, welche durch ihre festen Positionen innerhalb der Vergleichseinheit bestimmt sind. Dadurch bilden sie die definierte Schnittstelle zur Rekonfiguration zwischen dem globalen *VP/VC*-Bus mit der aktuellen Nummer und der Vergleichseinheit. Während der Rekonfiguration bestimmt der Mikrocontroller die logische Funktion der *Input-Gates* anhand der einzustellenden *VP/VC*-Nummer (Buffer für eine logische Eins, Inverter für eine Null) für alle 28 Bits des *VP/VC*-Busses. Diese Gatter-Information wird in die dedizierten Positionen der rekonfigurierbaren *Input-Gates* geschrieben.

Die Anzahl der Gatter in der vereinfachten Vergleichseinheit reduziert sich. Das 28-Bit breite *VC-VP*-Register in jedem Fehlerzähler sowie die lokalen Verbindungen die-

ses Registers mit den *XOR*-Gattern entfallen. Der 28-Bit-Bus mit der aktuellen *VP/VC*-Nummer wird von der zentralen Logik an alle Vergleichseinheiten weitergeleitet. Wenn keine aktive *feedback*-Leitung anzeigt, daß bereits ein Fehlerzähler für diese Nummer vorhanden ist, wird von der zentralen Logik ein Rekonfigurations-*Interrupt* zum Mikrocontroller gesendet, welcher unverzüglich für die benötigte *VC/VP*-Nummer in einen zur Zeit unbenutzten Fehlerzähler durch partielle Rekonfiguration der entsprechenden *Input-Gates* aktiviert.

Ergebnisse bei Anwendung der *CTR*-Methode bei *XC4000E/EX* und *XC6000*

Bei dieser Anwendung ist das Entwicklungsziel, so viele aktive Fehlerzähler wie möglich auf einem bestimmten *FPGA* zu betreiben. Daraus resultiert der Begriff der Hardware-Funktionalität:

- **Hardware-Funktionalität:** Die Eignung einer *FPGA*-Architektur für diese spezielle Anwendung, welche gemessen wird anhand der maximalen Anzahl von simultan laufenden Fehlerzählern, jeweils bezogen auf einen bestimmten Aufwand an *FPGA*-Ressourcen.

Dieser Zusammenhang wird hier als **Hardware-Funktionalität** bezeichnet.

#FZ	10	12	14	16	18	20	24
CLBs	505	611	724	817	898	1023	1219
FPGA	4020E	4025E	4028EX	4028EX	4028EX	4028EX	4036EX
CLB Ausnutzung	64%	59%	70%	79%	87%	99%	94%
FF Belegung	29%	27%	31%	35%	39%	44%	43%

Tabelle 6.2: FPGA-Ressourcen für die *CTR*-Methode bei der *XC4000E/EX*-Familie

In Tabelle 6.2 sind die Implementierungsergebnisse für die Anzahl der implementierbaren Fehlerzähler auf verschiedenen *XC4000-FPGA* unterschiedlicher Gatter-Kapazität zusammengefaßt.

- Bei der Benutzung eines *XC4020E* (10 FZ) und *XC4025E-FPGA* (12 FZ) ist die Ausnutzung der *CLB* bei ca. 65% begrenzt. Mehr Funktionalität kann nicht mehr implementiert werden, weil die Verdrahtungsressourcen auf dem Chip erschöpft sind.

Die *XC4000EX*-Chips haben wesentlich mehr Verdrahtungskanäle. Dadurch können auf einem *XC4028EX* bis zu 20 FZ implementiert werden, und die *CLB*-Auslastung steigt dabei auf 99% an, wobei die max. *FF*-Auslastung bei nur 44% liegt. Daraus folgt, daß der begrenzende Faktor nicht die verfügbaren Speicherelemente sind, sondern die Verdrahtungsressourcen. Diese werden hauptsächlich von den breiten globalen *on-Chip*-Bussen **b1** und **b2** (siehe Abbildung 6.6) verbraucht.

- Bei der Benutzung der *XC6000-FPGA* ermöglicht die *CTR*-Methode die Implementierung von maximal 20 FZ auf einem *XC6216*-Chip. Diese Architektur profitiert von der Eigenschaft, daß alle Speicherelemente in dem Adressraum eines Mikrocontrollers eingebettet sind und ohne Verdrahtungsressourcen auf dem Chip gelesen bzw. ge-

geschrieben werden können. Daraus folgt, daß die Busse **b1** und **b2** zusammen mit der Adress-Logik entfallen.

Die gleiche Anzahl von 20 FZ, welche einen XC4028EX voll auslasten, können damit auch auf einem XC6216-Chip implementiert werden. In den Datenblättern [Xil98/1] [Xil96] beschreibt der Hersteller den typischen Gatter-Kapazitätsbereich eines XC4028EX-Chips mit 18.000 bis 50.000 Gattern und den Bereich für den XC6216-Chip mit 16.000 bis 24.000 Gattern. Rechnet man mit den Durchschnittswerten von 32.000 Gattern für den XC4028EX und 20.000 Gattern für den XC6216, so erreicht der XC6216 eine um 60% höhere für die Anwendung nutzbare Hardware-Funktionalität. Dabei liegt für beide *FPGA* die Auslastung der jeweiligen *on-Chip*-Ressourcen bei ca. 100%.

Ergebnisse bei Änderung der *CTR*-Methode in die lokale *RTR*-Methode bei XC6000

In Abbildung 6.8 wird die Chip-Aufteilung bei einem XC6216 dargestellt. Jeder FZ benötigt eine Chip-Fläche von 10 x 16 Funktionseinheiten (bei XC6000 werden diese *cells* genannt). Dadurch steigt die Anzahl der mit lokaler *RTR*-Methode implementierbaren FZ von 20 bei *CTR* auf 24 FZ.

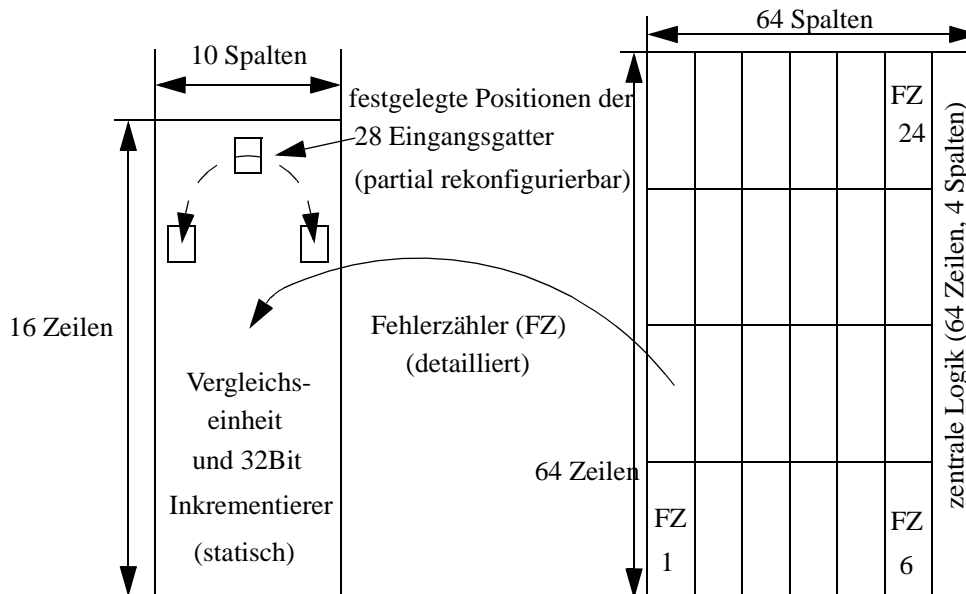


Abbildung 6.8 Chip-Aufteilung bei der lokalen RTR-Methode auf einem XC6216

Die Hardware-Funktionalität steigt im Gegensatz zur Anwendung der *CTR*-Methode auf dem gleichen Chip um weitere 20% an. Auf Grund der relativ schlechten Unterstützung der lokalen *RTR*-Methode durch entsprechende Entwurfswerkzeuge ist die Implementierung allerdings sehr schwierig. Im Betriebssoftware-Paket von SPYDER-CORE-P1 wurden bereits Treiber integriert, die die Rekonfiguration von XC6000-*FPGA* unterstützen, wie im nächsten Abschnitt dargestellt wird. Die detaillierte Darstellung dieser Bewertung von *FPGA*-Architekturen für den ATM-Diagnose-Monitor wurde in [WeKiRo98] veröffentlicht.

6.1.4.3 Systemintegration auf der SPYDER-CORE-P1 Basis-Platine

Das Blockdiagramm in Abbildung 6.9 zeigt die Integration aller Einzelkomponenten für das Anwendungsbeispiel ATM-Diagnose-Monitor beim abschließenden Systemtest.

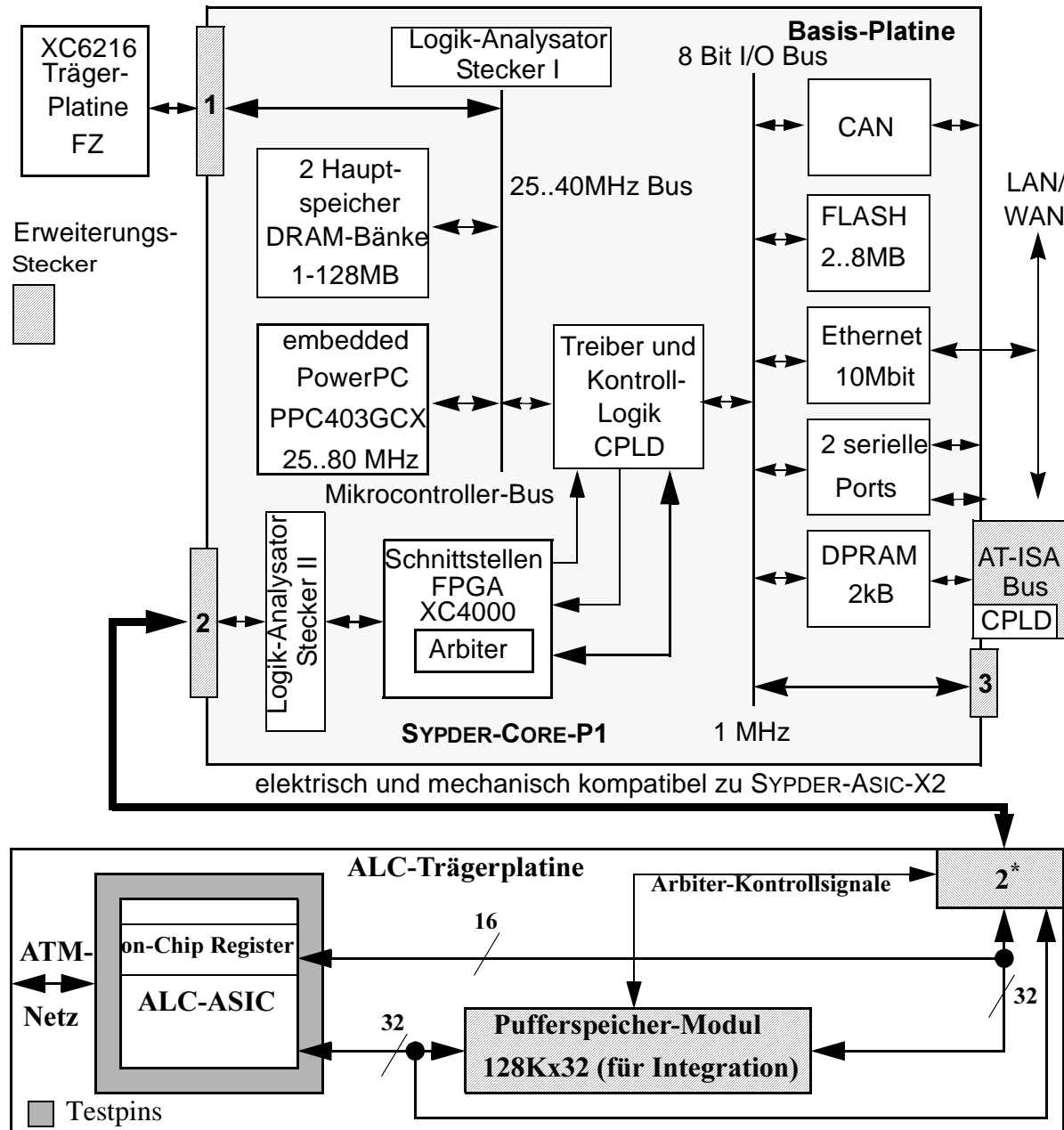


Abbildung 6.9 Systemintegration mit SYPDER-CORE-P1

Voraussetzung der effizienten Systemintegration ist die Kompatibilität beider Teilwerkzeuge. Die ALC-Träger-Platine wird über den Erweiterungsstecker 2, welcher bei beiden Werkzeugen gleiche elektrische und mechanische Eigenschaften besitzt, mit der Basis-Platine SYPDER-CORE-P1 verbunden. Die Ergebnisse der einzelnen Teilemulationen werden wie folgt zum Gesamtsystem zusammengesetzt:

- Der Pufferspeicher wird vom Werkzeug SPYDER-ASIC-X2 in das Pufferspeicher-Modul auf der *ALC*-Trägerplatine verlagert. Der *Arbiter* wird vom Schnittstellen-*FPGA* des Werkzeugs SPYDER-ASIC-X2 in das Schnittstellen-*FPGA* der Basis-Platine SPYDER-CORE-P1 übernommen.

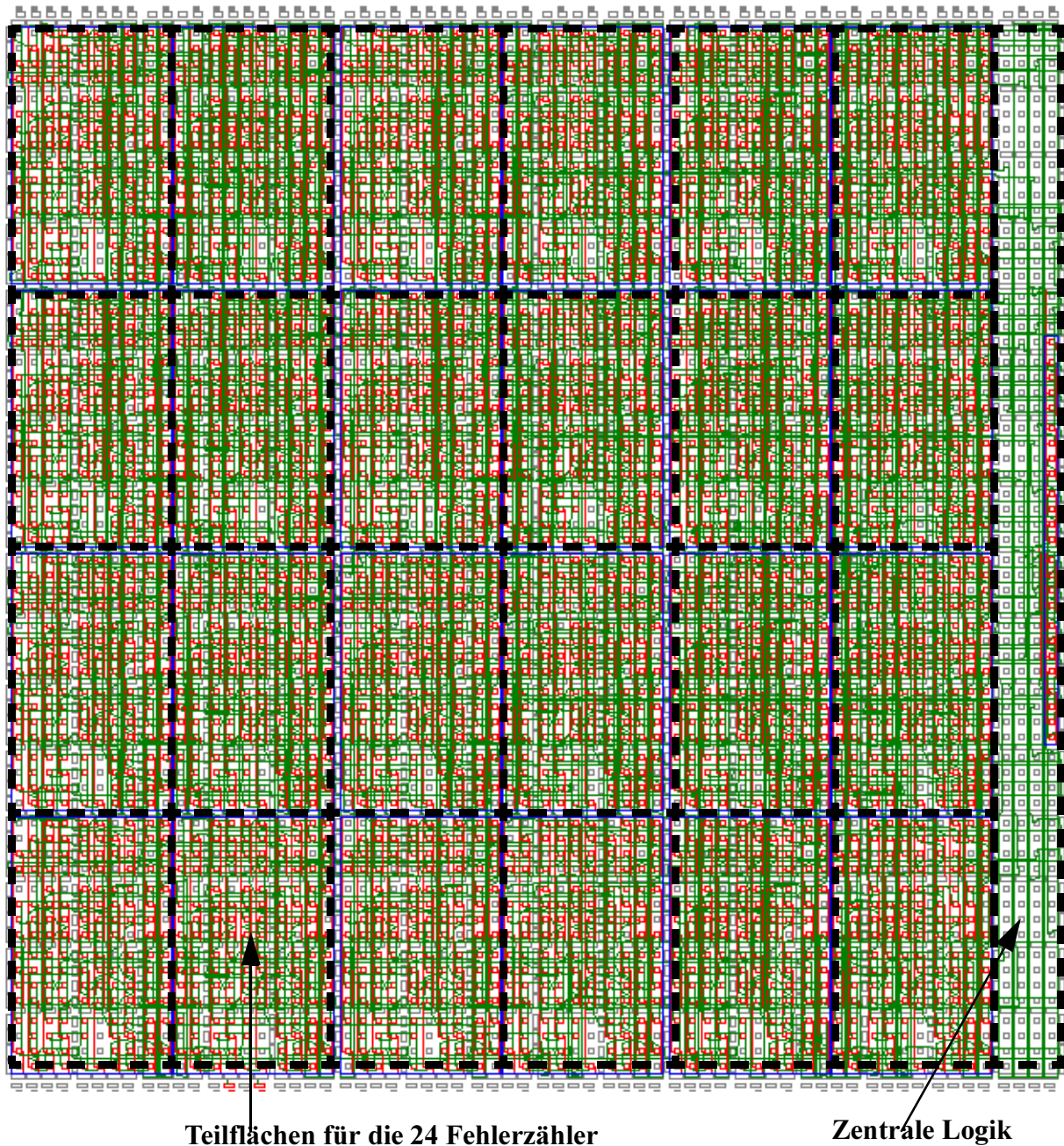


Abbildung 6.10 Chip-Layout des XC6216

- Die auf dem PC in C entwickelte Firmware wird mit einem C-Compiler auf den eigentlichen Mikrocontroller abgebildet. Wesentlich dabei ist, daß diese Firmware bereits funktionsfähig ist und nur an wenigen Stellen (z.B. Basis-Adressen, *Interrupt*-Nummern) modifiziert werden muß.

- Die Trägerplatine mit dem *XC6216-FPGA* wird über den Erweiterungsstecker direkt mit dem Mikrocontroller-Bus verbunden. Dieses *FPGA* dient zur Implementierung der 24 Fehlerzähler. Das *Layout* des *XC6216-Chips* ist in Abbildung 6.10 dargestellt.

Für das Laden und Testen der Anwendungssoftware während der Systemintegration auf dem Mikrocontroller wurde der *SDS70* Software-Monitor, welcher in Abschnitt 5.2.2 vorgestellt wurde, benutzt. Entscheidend dabei ist, daß alle funktionalen Eigenschaften der drei wichtigen Einzelkomponenten des eingebetteten Systems, nämlich die Software bzw. Firmware auf dem Mikrocontroller, die entwickelte *FPGA*-Hardware und der *ALC-ASIC*, durch parallel ablaufende Emulationen als funktionierende Architektur-Komponenten überprüft und bestätigt wurden.

6.1.5 Zusammenfassung des Beispiels ATM-Diagnose-Monitor

Zur Einordnung dieses Beispiels wird noch einmal auf den in dieser Arbeit vorgestellten Entwurfsablauf in Abbildung 5.1 verwiesen. Im wesentlichen wurden dabei der linke Pfad des Hardware-Entwurfs sowie der rechte Pfad der Software- bzw. Firmware-Entwicklung demonstriert. Es wurde gezeigt, wie eine umfangreiche Teilfunktion aus der Spezifikation vorwiegend auf verfügbare *ASIC*-Bausteine abzubilden ist, um dieses enorme Entwicklungspotential wiederzuverwenden. Beide Stufen der Entwurfsmethodik wurden demonstriert (siehe Abbildung 4.4):

- **Erste Stufe:** Die theoretischen Ansätze des Kapitel 4 zur Bewertung von *ASIC*-Komponenten wurden am konkreten Beispiel von *ALC*-Chips für *ATM*-Anwendungen demonstriert. Dabei wurden im ersten Schritt mehrere Alternativen anhand der eingeführten Entscheidungsparameter bewertet und daraus eine systematische Auswahlentscheidung für einen *ASIC* getroffen, der anhand objektiver Entscheidungsparameter die beste Auswahl darstellt.
- **Zweite Stufe:** Die getroffene Auswahl wurde in einem zweiten Schritt mit Hilfe des Emulationssystems *SPYDER* durch eine echtzeitfähige Emulation überprüft. Diese Ergebnisse wurden in [WeHeRo97] veröffentlicht.

Für den zusätzlich notwendigen Eigenentwurf einer weiteren Teilfunktion wurden zwei *SRAM*-basierte *FPGA*-Architekturen und ihre dazugehörigen Entwurfsmethoden untersucht. Die Ergebnisse wurden in [WeKiRo98] veröffentlicht und können wie folgt zusammengefaßt werden:

- Bei der Benutzung der gleichen *CTR*-Methode und Änderung der *FPGA*-Architektur von *XC4000* auf *XC6000* zeigte sich, daß die *XC6000*-Architektur eine um 60% höhere Hardware-Funktionalität der Anwendung zur Verfügung stellen kann. Die Vorteile der *XC6000*-Architektur im Gegensatz zur weitverbreiteten *XC4000*-Architektur liegen hauptsächlich in der Eigenschaft, daß alle Speicherelemente in den Adressraum eines Mikrocontrollers eingebettet sind und ohne *on-Chip*-Ressourcen gelesen bzw. geschrieben werden können. Damit eignet sich diese Architektur für Anwendungen, bei denen viele Zwischenwerte (hier die 32-Bit-FZ-Werte) zwischen einer aktiven Hardware in einem *FPGA* und einem Mikrocontroller zur Laufzeit ausgetauscht werden müssen.
- Bei einer Änderung der *CTR*-Methode zur lokalen *RTR*-Methode auf der gleichen *XC6000*-Architektur steigt die Hardware-Funktionalität um weitere 20%.

6.2 Aktuator-Sensor-Interface-Master (ASI-Master)

Im Gegensatz zu dem Anwendungsbeispiel in Abschnitt 6.1 demonstriert dieser Abschnitt den Fall, in dem kein *ASIC*-Baustein am Markt zur Implementierung einer benötigten umfangreichen Teilfunktion zur Verfügung steht. In diesem Fall muß eine Eigenentwicklung durchgeführt werden. Die Machbarkeit dieses Schrittes hängt wesentlich von den zur Verfügung stehenden Werkzeugen und der angewendeten Methode ab. Für das Anwendungsbeispiel „*ASI-Master*“ entfällt daher die erste Stufe der Bewertung von in Frage kommenden *ASIC*-Komponenten. Die zweite Stufe Emulation kann allerdings für die Eigenentwicklung der Hardware und Software in vollem Umfang angewendet werden. Dabei dient das Beispiel „*ASI-Master*“ aus drei wesentlichen Gründen als Benchmark, um die Leistungsfähigkeit der Werkzeuge und die Methodik zu validieren:

- Es besitzt eine anspruchsvolle Software-Architektur, basierend auf einem Echtzeitbetriebssystem.
- Es erlaubt die Evaluierung der Einflüsse von modernen Mikrocontrollern mit *on-Chip* Caches.
- Die Spezifikation verlangt kurze Reaktionszeiten auf externe Ereignisse, wodurch die *Task*-Umschaltzeiten und *Interrupt*-Reaktionszeiten des Echtzeitbetriebssystems einen begrenzenden Leistungsfaktor darstellen.

Diese Eigenschaften führen zu einem komplexen internen Systemverhalten, welches mit Hilfe der Emulationsumgebung SPYDER-CORE-P1 analysiert werden kann. Die Emulation muß dabei Antworten auf vier wichtige Fragestellungen geben, die ein Systementwickler braucht, um die Entscheidung für oder gegen den Einsatz eines Echtzeitbetriebssystems treffen zu können:

- Wo liegt die minimale Taktgeschwindigkeit des eingebetteten Systems, um alle Echtzeitbedingungen zu erfüllen?
- Wieviel Rechenleistung des Mikrocontrollers verbraucht das Echtzeitbetriebssystem?
- Wie groß ist die Leistungssteigerung der *on-Chip*-Caches, und welche Funktionen können damit realisiert werden?
- Was ist der Effekt verschieden großer Caches, und wie wirkt er sich auf wichtige Kenngrößen des Echtzeitbetriebssystems (z.B. *Task*-Umschaltzeit, *Interrupt*-Reaktionszeit) aus?

Der *ASI-Master* wird im folgenden Text benutzt, um daran diese Fragestellungen an einem praxisrelevanten Beispiel zu beantworten. Es wird gezeigt, daß diese Fragen von generellem Interesse für viele eingebettete Systeme im Bereich der industriellen Automation und Kommunikation sind. Insbesondere die Emulationswerkzeuge erlauben einen schnellen und genauen Einblick in das interne Systemverhalten.

6.2.1 Allgemeine Funktionsbeschreibung

Das *Aktuator Sensor Interface (ASI)* verbindet bis zu 32 *ASI-Slave*-Chips mit Hilfe eines einfachen zweiadrigen Kabels mit der sogenannten *ASI-Master*-Einheit, wie in Abbildung 6.11 dargestellt. Das Verbindungskabel hat dabei zwei wesentliche Aufgaben. Die erste Aufgabe ist

die Energieversorgung der *Slave-Chips*, welche zur Zeit als *Single-Chip-Lösungen (ASIC)* auf dem Markt verfügbar sind. Die Energie wird dabei von einem speziellen *ASI-Netzteil* eingespeist. Die zweite Aufgabe ist die bidirektionale Informationsübertragung zwischen *ASI-Master* und *Slave-Chips*.

Der *ASI-Master* ruft mit einem seriell codierten Masteraufruf in einem *Polling-Zyklus* alle mit dem *ASI-Kabel* verbundenen *Slaves* auf. Im *Master-Aufruf* ist die Adresse des *Slave* (A4..A0) codiert, gefolgt vom Ausgangsdatenabbild (I4..I0). Ein *Slave* antwortet mit der *Slave-Antwort*, wenn die Adresse im Masteraufruf mit seiner eigenen Adresse, welche in einem *EEPROM* gespeichert ist, übereinstimmt, und überträgt das Eingangsdatenabbild zurück zum *ASI-Master*. Jeder *Slave* ist in der Lage, maximal bis zu vier binäre Sensoren und Aktuatoren zu kontaktieren. Damit sind bei 32 *Slave-Chips* pro *ASI-Kabel* bis zu 128 Sensoren und 128 Aktuatoren bedienbar. Wenn ein *ASI-Kabel* an einer *ASI-Master-Einheit* angeschlossen ist, spricht man von einem *Einkanal-ASI-Master*.

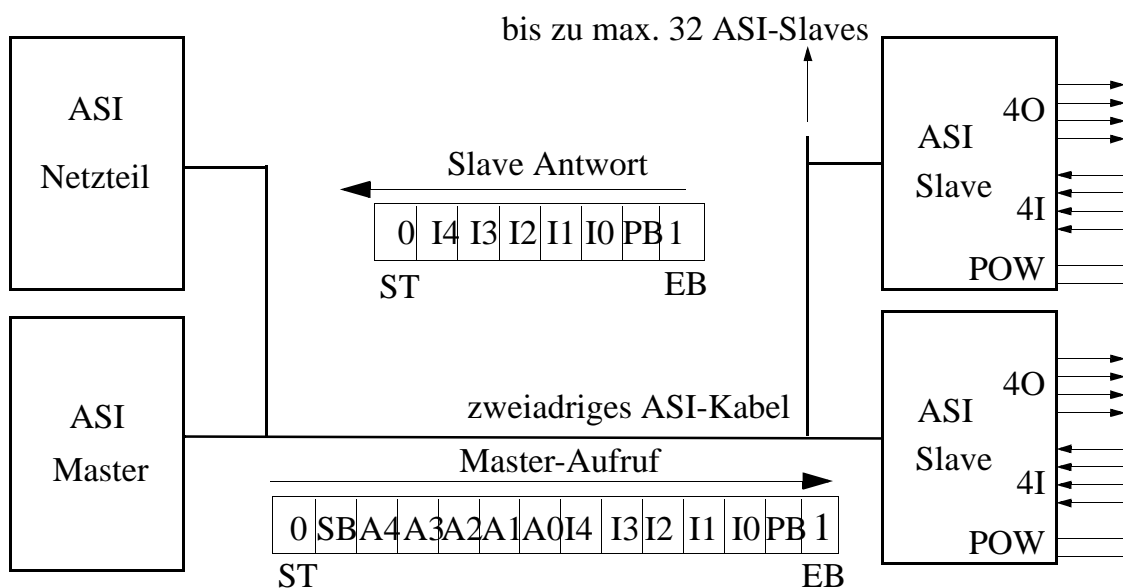


Abbildung 6.11 ASI-Kommunikationssystem

Das serielle Protokoll wird mit Hilfe eines speziellen *Analog-Digital-ASIC* auf das *ASI-Kabel* aufmoduliert. Eine genaue Beschreibung des kompletten *ASI-Standards* ist in [ASI98] zu finden.

6.2.2 Initiale Hardware/Software-Partitionierung

Der *ASI-Master* wird auf der *SPYDER-CORE-P1* Basis-Platine emuliert, ohne signifikante Änderungen zum eigentlichen finalen Zielsystem. Die Verbindung des Mikrocontrollers zum *Analog-Digital-ASIC*, welcher das serielle Protokoll auf das *ASI-Kabel* aufmoduliert und im Empfangsbetrieb demoduliert, wird durch eine digitale Hardware im Schnittstellen-*FPGA* implementiert. Die Architektur dieser *ASI-spezifischen* Hardware ist in Abbildung 6.12 dargestellt und repräsentiert den durchzuführenden Eigenentwurf, da diese Funktionalität zur Zeit als *ASIC* am Markt nicht verfügbar ist.

Der Mikrocontroller schreibt das Ausgangsdatenabbild für einen adressierten *Slave* in ein Schreibregister (engl.: *write_reg_low*, *write_reg_high*) bzw. liest das Eingangsdatenabbild des *Slave* (n+1) unmittelbar nach dem Schreiben zurück. Der sogenannte *ASI-UART* liest die Sen-

dedaten ein, führt eine seriell-parallel-Wandlung durch und leitet den seriellen Bit-Strom *Manchester*-codiert zum Analog-Digital-ASIC weiter.

Im Empfangsbetrieb empfängt die Schnittstellen-Hardware ein serielles Eingangssignal vom Analog-Digital-ASIC, welches zunächst *Manchester*-decodiert wird. Danach wird es seriell-parallel gewandelt und in die beiden 8-Bit-Leseregister eingetragen. Nach dem vollständigen Empfang der *Slave*-Antwort wird automatisch ein *Interrupt* zum Mikrocontroller ausgelöst. Dieser *Interrupt* wiederholt sich alle 220 μ s und ist praktisch eine durch das Zeitverhalten des *ASI*-Protokolls festgelegte Konstante, die während der Analyse noch eine wichtige Rolle spielen wird.

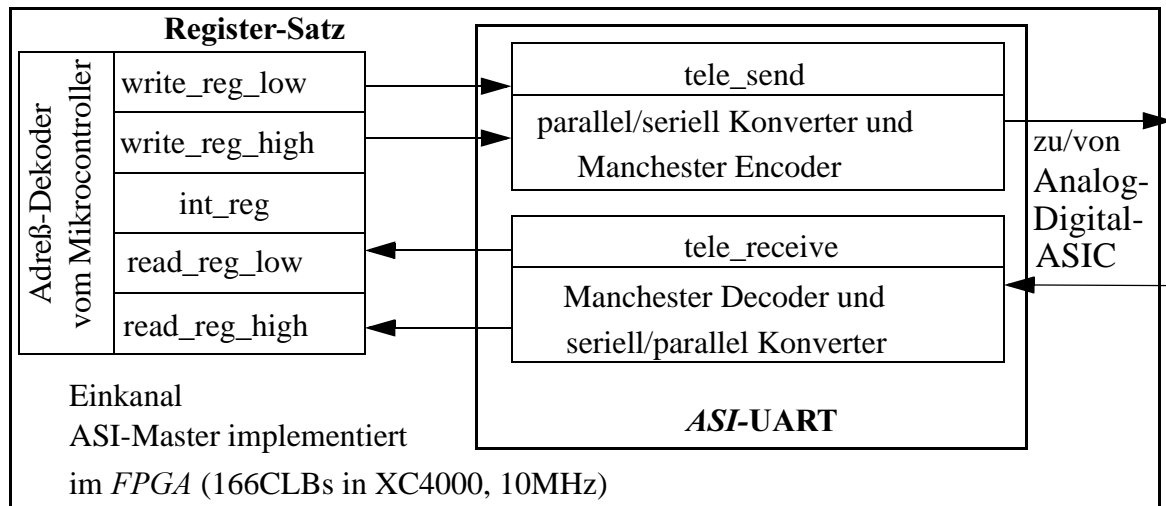


Abbildung 6.12 Initiale ASI-Schnittstellen-Hardware im *FPGA*

Auf der SPYDER-CORE-P1 Basis-Platine wurde das Echtzeitbetriebssystem *VxWorks* [Wind97] portiert. Zusätzlich zu der bereits in Abschnitt 2.2.1 beschriebenen Funktionalität stellt es einen *TCP/IP-Stack* und einen *http-Server* für die Kommunikation über das Internet zur Verfügung. Die Architektur der gesamten initialen Software-Architektur auf dem Mikrocontroller ist in Abbildung 6.13 dargestellt.

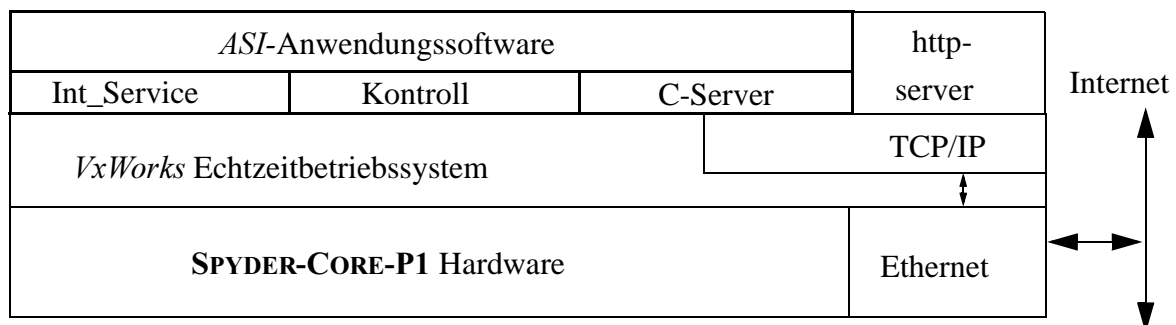


Abbildung 6.13 Initiale Software-Architektur auf dem Mikrocontroller

Insgesamt laufen auf dem Echtzeitbetriebssystem vier *Tasks*. Zwei *Tasks* unterliegen harten Echtzeit-Bedingungen, die beiden anderen *Tasks* haben keine Echtzeit-Einschränkungen.

- Die *Int_Service Task* unterliegt harten Echtzeit-Bedingungen und ist verantwortlich für den Datenaustausch mit den *Slaves*. Sie generiert das aktuelle Prozeßdatenabbild.

- Die *Kontroll-Task* unterliegt ebenfalls harten Echtzeit-Bedingungen und benutzt das Prozeßdatenabbild zur Ableitung von entsprechenden Steuerkommandos.
- Die *C-Server-Task* hat keine Echtzeit-Anforderungen und ist verantwortlich für den Daten- und Kommando-Austausch über das Internet.

Diese drei *Tasks* wurden speziell für den *ASI-Master* entwickelt und bilden die eigentliche Anwendungssoftware. Die vierte *Task* wird vom Echtzeitbetriebssystem bereitgestellt.

- Die *embedded http-Server Task* hat ebenfalls keine Echtzeit-Einschränkungen und transferiert ein *JAVA-Applet* zu einem aufrufenden *Client-Computer*. Eine detaillierte Darstellung dieses Mechanismus ist in [HeWWRo98] und [Wind97] zu finden.

6.2.3 Emulation der Kenngrößen des Echtzeitbetriebssystems

In Abbildung 6.14 ist das Zeitverhalten des *ASI-Masters* dargestellt. Dafür wurden die Anfangspunkte und Endpunkte der einzelnen *Tasks* mit *I/O*-Zugriffen auf ein einfaches externes Gerät (auf der Basis-Platine stehen dafür *Leuchtdioden* zur Verfügung) markiert. Diese Zeitabschnitte können mit einem Logik-Analysator aufgezeichnet und ausgewertet werden.

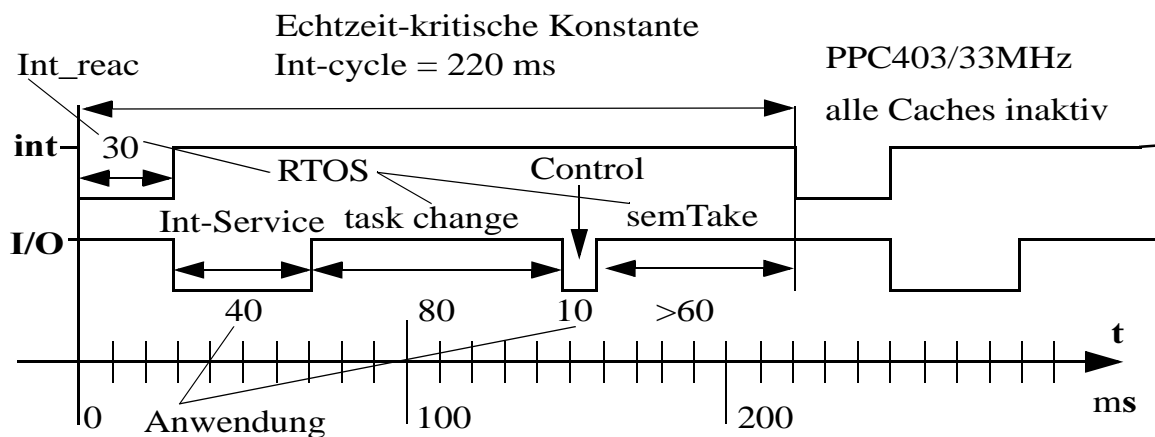


Abbildung 6.14 Ausführungszeiten auf dem Mikrocontroller (Logik-Analysator-Aufzeichnung)

Das obere *int*-Signal zeigt die *Interrupt*-Kommunikation zwischen der *ASI*-spezifischen Hardware im Schnittstellen-*FPGA* und dem Mikrocontroller. Die *Int_Service* und *Kontroll-Task* kennzeichnen ihren Beginn und ihr Ende mit einem Zugriff auf das *I/O*-Gerät, dargestellt durch das untere *I/O*-Signal. Die *ASI*-Spezifikation definiert die Zeit zwischen zwei sequentiellen Bits auf dem *ASI*-Kabel mit $6 \mu\text{s}$. Zusammen mit *Master*-Aufruf, *Slave*-Antwort und Pausenzeiten muß alle $220 \mu\text{s}$ ein Datenaustausch mit einem *Slave* durchgeführt werden. Dieser Wert von $220 \mu\text{s}$ wird an dieser Stelle als *ASI*-spezifische Echtzeit-kritische Konstante eingeführt.

Während dieser Zeit muß der Mikrocontroller die folgende Echtzeit-kritische *Task*-Sequenz ausführen: *Interrupt*-Reaktion (*Int_reac*), *Int_Service*, *Task*-Umschaltung zur *Kontroll-Task*, Ausführen der *Kontroll-Task* und abschließend die Ausführung der Betriebssystem-Funktion *SemTake*, wie in Abbildung 6.14 dargestellt.

Um die minimale Taktfrequenz für den Mikrocontroller zu bestimmen, bei der er noch alle Echtzeit-Anforderungen einhalten kann, muß eine *worst-case*-Analyse durchgeführt werden, d.h. alle Caches müssen deaktiviert werden. Die beschriebene *Task*-Sequenz muß innerhalb

(oder kleiner-gleich) des Zeitraums für die Echtzeit-kritische Konstante ausgeführt werden. In Abbildung 6.14 ist der Fall dargestellt, bei dem die Echtzeit-Anforderungen gerade noch erfüllt werden. Der Wert der Mikrocontroller-Taktfrequenz liegt dabei bei 33 MHz.

6.2.3.1 Analyse der Leistungsgrenzen durch Emulation

Betrachtet man nun das Systemverhalten über den vollen Taktfrequenz-Bereich des eingesetzten Mikrocontrollers *PPC403GA/GCX* und in Abhängigkeit von verschiedenen Cache-Parametern, so erhält man die Verteilung der Rechenleistung in Abbildung 6.15.

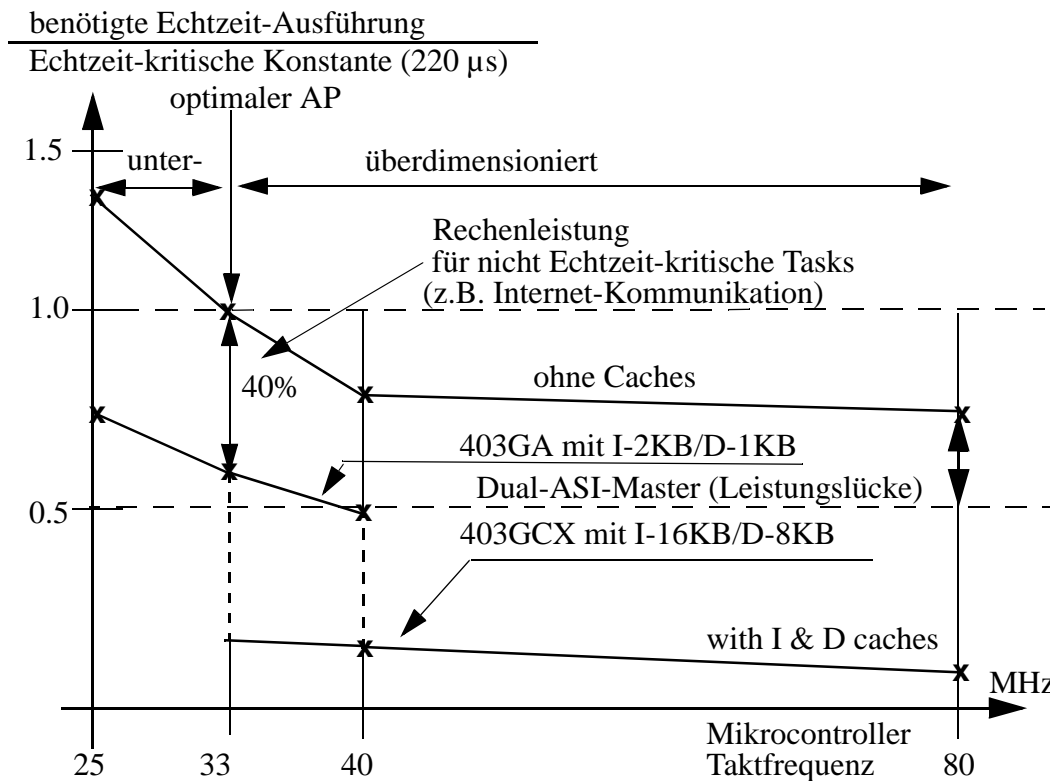


Abbildung 6.15 Verteilung der Rechenleistung des Mikrocontrollers

Dabei variiert die Taktfrequenz im Bereich zwischen 25 MHz und 80 MHz auf der X-Achse. Die Y-Achse zeigt das Verhältnis der tatsächlich benötigten Echtzeit-Ausführungszeit des Mikrocontrollers für die beschriebene kritische *Task*-Sequenz, bezogen auf die Echtzeit-kritische Konstante von 220 µs.

Die obere Kurve zeigt den *worst-case*-Fall ohne Cache-Wirkung. Bei 33 MHz ist der Mikrocontroller gerade noch in der Lage (beim Verbrauch seiner gesamten Rechenleistung) die Echtzeit-Anforderungen einzuhalten. In diesem Fall wäre keine Rechenleistung mehr frei, um die nicht Echtzeit-kritischen Aufgaben zu bearbeiten. Wird die Taktfrequenz kleiner, steigt das Verhältnis über den Wert von 1.0, d.h. das eingebettete System verletzt die Echtzeit-Anforderung und ist unterdimensioniert. Werte über 33 MHz bedeuten, daß das System überdimensioniert ist. Der Betriebspunkt bei 33 MHz wird deshalb optimaler Arbeitspunkt (AP) genannt.

Die unteren Kurven zeigen das Verhalten mit eingeschalteten Caches. Diese Betriebsart entspricht dem normalen Einsatz des Mikrocontrollers. Die Kurve zeigt am AP einen Leistungsgewinn von ca. 40% bei einem *PPC403GA*, welcher einen 2 KByte-I-Cache und einen 1

KByte-D-Cache besitzt. Diese Leistungssteigerung kann zwar über die meiste Zeit während der Laufzeit erwartet werden, ist aber nicht immer garantiert. Deshalb kann diese Leistungssteigerung nur für nicht Echtzeit-kritische Systemaufgaben benutzt werden. In diesem Anwendungsbeispiel sind dies die beiden bereits beschriebenen *Tasks* zur Internet-Kommunikation.

Im weiteren Verlauf der Kurve bis zum maximal-Wert von 80 MHz kann eine zusätzliche Leistungssteigerung gemessen werden. In diesem Bereich wird der *PPC403GCX* eingesetzt, welcher intern seine Taktfrequenz verdoppeln kann und über eine achtfache Cache-Größe (16 KByte-I-Cache, 8 KByte-D-Cache) verfügt.

PPC403GA 33MHz (AP) 2KB I-Cache 1KB D-Cache	ohne I-Cache ohne D-Cache	ohne I-Cache mit D-Cache	mit I-Cache ohne D-Cache	mit I-Cache mit D-Cache
Task Umschaltzeit	100% (87 μ s)	-1%	+46%	+60%
Interrupt-Reaktionszeit	100% (27 μ s)	-4%	+50%	+43%
Dhrystone	100% (6211)	+10%	+187%	+455%

Tabelle 6.3: Leistungssteigerungsanalyse bei PowerPC403GA-Mikrocontrollern

In Tabelle 6.3 wird die Leistungssteigerung durch die Cache-Wirkung am Arbeitspunkt bei 33 MHz weiter untersucht. Sie zeigt die Zusammenwirkung unterschiedlicher Cache-Konfigurationen beim *PPC403GA*. Aus dem Datenblatt des Mikrocontroller-Herstellers [IBM98] kann als eine typische Kenngröße für die Leistung der *Dhrystone*-Wert entnommen werden, der eine Auswahl bestimmter Integer-Algorithmen zusammenfaßt.

Wenn ein Echtzeitbetriebssystem eingesetzt wird, reicht dieser Wert zur Bewertung der Rechenleistung nicht aus. In diesem Zusammenhang benötigen die Entwickler als Entscheidungshilfe Aussagen über die *Task*-Umschaltzeiten und *Interrupt*-Reaktionszeiten. Insbesondere der Einfluß der Cache-Wirkung auf die Verbesserung dieser Werte ist dabei von Interesse.

Die Hauptaussage der Tabelle 6.3 ist die Tatsache, daß diese beiden Schlüsselwerte nur um 60% bzw. 43% durch das Zuschalten der Caches verbessert werden. Diese Erkenntnis steht im Gegensatz zu dem weitverbreiteten *Dhrystone*-Wert, welcher eine Verbesserung von 455% verspricht. Diese Verbesserung ist bei den wichtigen Kenngrößen in einem eingebetteten System in realistischen Anwendungen nicht erreichbar und liegt um den Faktor zehn niedriger. Ein weiterer bemerkenswerter Effekt kann in der Spalte ohne I-Cache und mit D-Cache beobachtet werden. Wenn der D-Cache sehr klein ist (wie beim *PPC403GA* nur 1 KB), dann kann keine Steigerung der Ausführungsgeschwindigkeit erreicht werden. Im Gegenteil, es ist sogar mit einer kleinen Verschlechterung zu rechnen.

Die Tabelle 6.4 zeigt den Einfluß bei der Vergrößerung der Caches um den Faktor acht durch Einsatz eines *PPC403GCX*, welcher in der gleichen Umgebung im Arbeitspunkt bei 33 MHz betrieben wird. Die Tabelle zeigt, daß sich die Werte für die beiden Kenngrößen stark verbessern, wenn die Caches signifikant vergrößert werden. Bei eingeschalteten Caches verbessern sich die *Task*-Umschalt- und *Interrupt*-Reaktionszeiten um 340% bzw. 377% und erreichen etwa die gleiche Größenordnung, welche auch vom *Dhrystone*-Wert (529%) ausgedrückt wird.

PPC403GCX 33 MHz (AP) 16 KB-I-Cache 8 KB-D-Cache	ohne I-Cache ohne D-Cache	ohne I-Cache mit D-Cache	mit I-Cache ohne D-Cache	mit I-Cache mit D-Cache
<i>Task</i> -Umschaltzeit	100% (87 μ s)	+10%	+152%	+340%
Interrupt-Reaktionszeit	100% (27 μ s)	+12%	+205%	+377%
<i>Dhrystone</i> s	100% (6211)	+11%	+207%	+529%

Tabelle 6.4: Leistungssteigerungsanalyse bei PowerPC403GCX-Mikrocontrollern

Das zentrale Ergebnis dieser Untersuchung ist, daß beim Einsatz eines Echtzeitbetriebssystems nur relativ große Caches (16 KB/8 KB) einen signifikanten Gewinn für die *Task*-Umschaltzeit sowie der Interrupt-Reaktionszeit erbringen. Diese Tatsache ist noch wichtiger, wenn man bedenkt, daß die meisten heute auf dem Markt verfügbaren Mikrocontroller nur kleine Caches im Bereich von 2 KB/1 KB haben, wie z.B. die MPC8xx oder die MPC5xx-Familien (siehe auch Tabelle 2.3).

6.2.3.2 Emulationsergebnisse der initialen Hardware/Software-Partitionierung

Die Emulation der initialen Hardware/Software-Partitionierung des *ASI-Masters* auf SPYDER-CORE-P1 hat bis zu diesem Punkt zunächst einen detaillierten Einblick in das komplexe interne Systemverhalten ermöglicht. Insbesondere können Antworten auf die am Anfang dieses Anwendungsbeispiels extrahierten Fragestellungen gegeben werden, die in [WeSNRo99] veröffentlicht wurden:

- Der Arbeitspunkt, an dem alle Echtzeitbedingungen eingehalten werden können, liegt bei dieser initialen Hardware/Software-Partitionierung bei 33 MHz.
- Das Echtzeitbetriebssystem braucht ca. 80 μ s für die *Task*-Umschaltung und ca. 30 μ s für die Reaktion auf einen externen *Interrupt*, jeweils unter Annahme des schlechtesten Falles. Wie aus Abbildung 6.14 zu erkennen ist, benötigt in diesem Fall das Echtzeitbetriebssystem 170 μ s von der maximal zur Verfügung stehenden Zeit (220 μ s) und die eigentliche Anwendung nur 50 μ s. Daraus folgt, daß das Echtzeitbetriebssystem ca. 77% der Ausführungszeit des Mikrocontrollers verbraucht.
- Wenn die Caches aktiviert sind, kann bei einem *PPC403GA* (2K-I, 1K-D) ein Leistungsgewinn von ca. 40% gemessen werden. Dieser Gewinn kann für nicht Echtzeit-kritische Systemfunktionen wie die Internet-Kommunikation verwendet werden.
- Erst wenn die *on-Chip*-Caches drastisch vergrößert werden (Faktor fünf bis zehn), verbessern sich auch die Werte für die *Task*-Umschalt und *Interrupt*-Reaktionszeiten des Echtzeitbetriebssystems signifikant und erreichen Werte, die auch in *Dhrystone*-Messungen ausgedrückt werden.

Um die gemessenen Ergebnisse zu verifizieren, wurde als zweites Echtzeitbetriebssystem *RTEMS* [Army96] auf die SPYDER-CORE-P1 Basis-Platine portiert und die Messung wiederholt. In dieser Untersuchung blieben alle Umgebungsbedingungen gleich, d.h. gleiche Hardware,

gleiche Anwendung und gleiche Code-Generierung (*gmu-C-Compiler*, gleicher opt. Level). Diese Untersuchung ergab, daß *RTEMS* das gleiche Verhalten zeigt wie *VxWorks*, wobei die Differenzen lediglich bei weniger als 5% lagen. Diese Ergebnisse zeigen, daß das beobachtete Verhalten nicht an ein spezielles Echtzeitbetriebssystem gebunden ist.

6.2.4 Verbesserte Hardware/Software-Partitionierung

Die Emulation der initialen Hardware/Software-Partitionierung hat ergeben, daß die *Interrupt*-Reaktionszeit zusammen mit der *Task*-Umschaltzeit etwa 110 μ s dauert. Dieser Wert liegt in der gleichen Größenordnung wie die externen Anforderungen an das System. Auf Grund dieser sehr ungünstigen Verhältnisse steigt der *Overhead* des Betriebssystems drastisch an. In diesem Beispiel wird vom Echtzeitbetriebssystem 77% der Rechenleistung im Arbeitspunkt bei 33 MHz verbraucht. Daraus resultieren zwei wesentliche Motivationen, die initiale Hardware/Software-Partitionierung nicht zu übernehmen, sondern in einer weiteren Iteration der Schleife in Abbildung 5.1 eine verbesserte Hardware/Software-Partitionierung durch Emulation zu erarbeiten:

- Eine Motivation liegt darin, an dem *ASI-Master* ein zweites Kabel mit weiteren 32 *Slaves* zu betreiben (*Dual-ASI-Master*). In diesem Fall muß die beschriebene *Task*-Sequenz in der Hälfte der Zeit bearbeitet werden. Wie Abbildung 6.15 zeigt, wäre diese Forderung bei der initialen Hardware/Software-Partitionierung selbst bei einer max. Taktfrequenz von 80 MHz unmöglich. Bei der *worst-case*-Betrachtung existiert eine Leistungslücke.

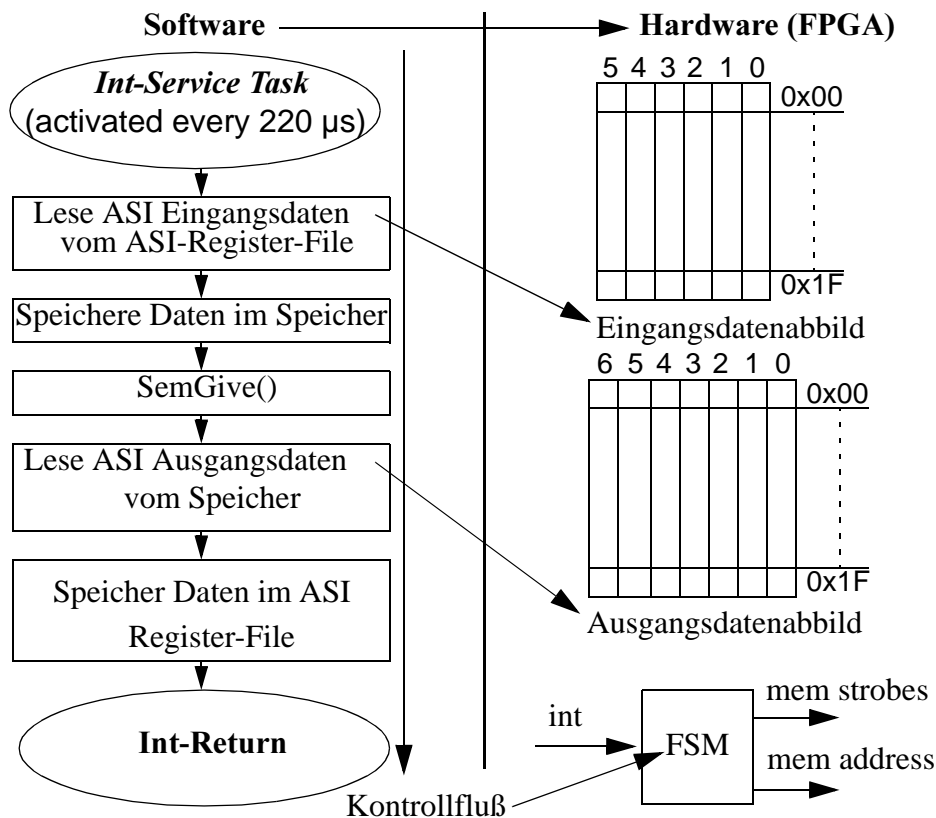


Abbildung 6.16 ISR-Verschiebung von Software nach Hardware

- Die zweite Motivation liegt in dem Bestreben, die Systemkosten für die Serienproduktion weiter zu reduzieren, indem der externe Mikrocontroller-Bus von 32-Bit auf 8-Bit reduziert wird und langsamere Speicher eingesetzt werden können.

Diese Ziele können nur dann realisiert werden, wenn die *Int_Service-Task* als Software-Funktion auf dem Mikrocontroller in die Hardware auf dem *FPGA* verschoben wird, damit die extrem häufigen *Task*-Wechsel eliminiert werden, welche einen großen Teil der Rechenleistung des Mikrocontrollers verbrauchen. Diese Verschiebung ist in Abbildung 6.16 dargestellt.

Die linke Seite des Diagramms zeigt den Programmablauf der *Int_Service-Task*. Der Mikrocontroller empfängt alle 220 μ s einen Interrupt, wobei er die aktuellen Eingangsdaten von einem *Slave* einliest und neue Ausgangsdaten zu einem *Slave* schickt. Wenn an einem *ASI*-Kabel 32 *Slaves* betrieben werden, wird in ca. 7 ms ein komplettes Prozeßabbild erzeugt. Wenn diese Funktionalität in Hardware ausgelagert werden soll, werden im *FPGA* ein Speicher für das Eingangsdatenabbild (6-Bit x 32 = 192 Bits), ein Speicher für das Ausgangsdatenabbild (7-Bit x 32 = 224 Bits) und eine Zustandsmaschine benötigt.

Wie im verbesserten Blockdiagramm in Abbildung 6.17 dargestellt, werden zusätzlich zur bereits eingeführten initialen *ASI*-Schnittstellen-Hardware (siehe Abbildung 6.12) zwei *on-Chip*-Speicher und eine Zustandsmaschine (FSM) im *FPGA* implementiert.

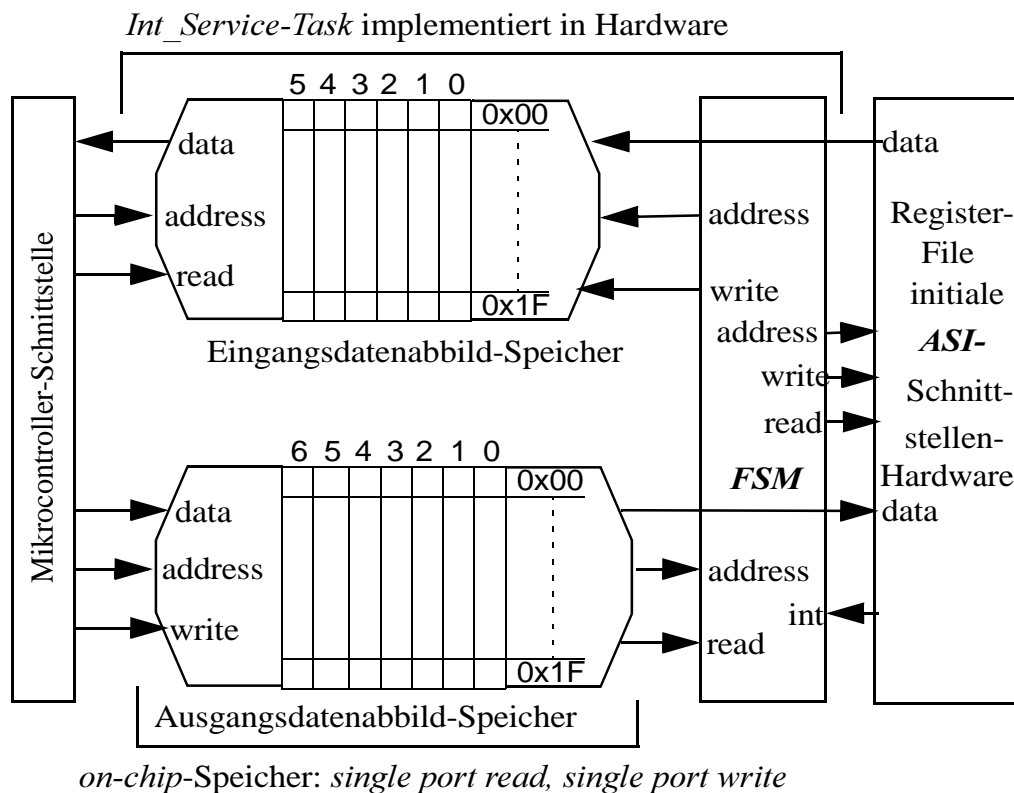


Abbildung 6.17 Erweiterte Hardware-Architektur nach Auslagerung der *ISR*

Diese erweiterte Hardware-Architektur im *FPGA* befreit den Mikrocontroller von der Reaktion auf die externen *Interrupts* alle 220 μ s mit Hilfe einer *ISR* in Software. Die meisten Systeme im Bereich der industriellen Automation erlauben die Erzeugung von neuen Prozeßabbildern im Bereich von 10 ms. Dadurch sinkt die Anzahl der *Task*-Umschaltungen um den Faktor 45 (10 ms/220 μ s). Diese Tatsache reduziert den Ressourcen-Verbrauch des Echtzeitbetriebssystems drastisch.

6.2.4.1 Ausnutzung von *on-Chip*-Eigenschaften bei neuen *FPGA*

Die Effizienz der Hardware-Implementierung hängt wesentlich von der Ausnutzung der *on-Chip*-Eigenschaften ab, welche neue *FPGA*-Architekturen besitzen. Die Implementierung der Architektur von Abbildung 6.17 kann als Benchmark benutzt werden, um die verschiedenen *FPGA* für Anwendungen zu evaluieren, welche neben kombinatorischer und sequentieller Logik auch *on-Chip-SRAM*-Speicher erfordern. Im wesentlichen müssen drei Architekturkomponenten implementiert werden:

- Die initiale *ASI*-Schnittstellen-Hardware, welche im wesentlichen aus Zustandsmaschinen in Verbindung mit vielen Zählern besteht.
- Die Zustandsmaschine für die Implementierung des Kontrollflusses der *Interrupt-Routine*. Diese Komponente kommt bei der verbesserten Partitionierung zusätzlich dazu.
- Zwei *on-Chip-SRAM*-Speicherbereiche für das Eingangsdatenabbild (192 Bits) und das Ausgangsdatenabbild (224 Bits), welche gleichzeitig lesbar und schreibbar sein müssen (engl.: *single port read, single port write*).

Die SPYDER-CORE-P1 Emulationsumgebung bietet drei *FPGA*-Architekturen zur Bewertung an. Die Antifuse-*FPGA* von Actel und die XC4000-Familie von Xilinx sind in diesem Anwendungsgebiet weit verbreitet und können direkt miteinander in einen Vergleich gesetzt werden, während die XC6000-Architektur zur Zeit mehr im Bereich der Forschung angesiedelt ist und für kommerzielle Produkte nicht eingesetzt wird. Die wesentlichsten Vorteile dieser Architektur allerdings werden in neuesten Architekturen wie z.B. der Virtex-Serie übernommen. Dabei wurden folgende Ergebnisse ermittelt, welche auch in [WeStRo99] veröffentlicht wurden:

- Bei Verwendung der XC4000-FPGA werden für die initiale *ASI*-Schnittstellen-Hardware 166 *CLB* belegt. Die *Int_Service-FSM* benötigt weitere 20 *CLB*. Die *LUT* der *CLB* können zur Implementierung der *SRAM*-Speicherfunktion (engl.: *fine grained SRAM feature*) benutzt werden. Bei der *single-port-read/single-port-write-Option* können jeweils 16 Bits in einem *CLB* gespeichert werden. Für die gesamte Speichergröße werden insgesamt 26 *CLB* gebraucht. Daraus folgt, daß für den zusätzlichen Aufwand bei der erweiterten Architektur nur 28% ($46\text{CLB}/166\text{CLB}$) mehr *FPGA*-Ressourcen benötigt werden.
- Bei der Verwendung der Antifuse-*FPGA* A1200XL werden für die initiale *ASI*-Schnittstellen Hardware 436 *LM* (engl.: *logic module, Actel area unit*) und für die *Int_Service-FSM* 51 *LM* benötigt. Die Actel-*FPGA* stellen dediziertes *on-Chip-SRAM* erst in Chips ab dem A32100DX mit 1362 *LM* oder größer zur Verfügung. Für diese Anwendung, die weit weniger *LM* benötigt, können in den kleineren Chips nur FF zur Speicherung der Ein- und Ausgangsdatenabbilder benutzt werden. Dafür werden 763 *LM* gebraucht. Das Verhältnis zwischen dem zusätzlichen Aufwand der verbesserten Architektur zur initialen Architektur ($814\text{LM}/436\text{LM}$) liegt deshalb bei 187%.
- Bei der Verwendung der XC6000-Architektur werden für die initiale *ASI*-Schnittstellen Hardware 841 XC6000-Zellen (engl.: *cells*) und für die *Int_Service-FSM* 112 *cells* verbraucht. Die Speicherbereiche müssen mit *FF* implementiert werden, welche in der Chip-Architektur bereits eingebettet sind. Das bedeutet, keine Verdrahtungsressourcen werden für den Zugriff durch einen Mikrocontroller benötigt. Die benötigte Speicherfunktionalität kann mit 957 *cells* realisiert werden. Daraus errechnet sich ein Verhält-

nis zwischen zusätzlichem Aufwand der erweiterten Architektur zu der ursprünglichen initialen Architektur von 127% (1069 *cells*/841 *cells*).

Die Ergebnisse basieren alle auf dem gleichen *VHDL*-Code, welcher und mit Hilfe des Entwicklungsablaufs in Abbildung 5.5 auf die Emulationsumgebung SPYDER-CORE-P1 abgebildet wurde. Sie sind in Tabelle 6.5 zusammengefaßt.

	XC4000 (CLBs)	XC6000 (cells)	Actel (logic modules)
initiale <i>ASI</i> -Schnittstellen-HW	166	841	436
<i>Int_Service FSM</i>	20	112	51
<i>Int_Service on-Chip</i> -Speicher	26	957	763
zusätzliche Ressourcen für <i>Int_Service (FSM, Speicher)</i>	28%	127%	187%

Tabelle 6.5: Implementierungsergebnisse auf verschiedenen *FPGA*-Architekturen

Die Spalte mit der XC4000-Architektur zeigt, daß die Ausnutzung der feingranularen *SRAM*-Fähigkeit zu einer sehr effizienten Implementierung der zusätzlichen Komponenten der erweiterten *ASI*-Hardware-Architektur führt. Diese Architektur ist für Anwendungen im Bereich bis 10.000 Gatter und der Forderung nach *on-Chip-SRAM* im Bereich einiger hundert Bits sehr gut geeignet. Diese Art der Anforderung wird bei den in dieser Arbeit betrachteten eingebetteten Systemen in der industriellen Automation häufig angetroffen.

Die XC6000-Architektur ist weniger geeignet. Zwar profitiert diese Architektur auch wie beim ATM-Diagnose-Monitor von der Tatsache, daß die *FF* im Adressraum des Mikrocontrollers eingebettet sind, allerdings ist die Anzahl der Speicherelemente zu gering, damit diese Fähigkeit signifikant zum Tragen kommt.

Die Einsetzbarkeit der *Actel-FPGA* ist nur bei Standard-Anwendungen ohne *on-Chip SRAM* gegeben. Diese *FPGA*-Familie hat die wesentliche Einschränkung, daß im Gatter-Bereich unter 10.000 keine *on-Chip-SRAM*-Fähigkeit besteht.

6.2.5 *ASI*-Demonstrator

Im Rahmen dieser Arbeit wurde für dieses Anwendungsbeispiel ein *ASI*-Demonstrator entwickelt. Die Abbildung 6.18 zeigt die Architektur des gesamten *ASI-Masters* nach der Systemintegration und dem Test auf der SPYDER-CORE-P1 Basis-Platine.

Ein *JAVA-Applet* wird auf einem *Client-Computer* ausgeführt. Es ist über *TCP/IP* mit dem *C-Server* verbunden, welcher unter *VxWorks* auf dem Mikrocontroller des eingebetteten Systems läuft. Diese Verbindung wird benutzt, um Kommandos für die Kontroll-*Task* zu übertragen, welche ein funktionsfähiges Modell eines Hochregals steuert.

Dieses Modell besitzt 28 Sensoren (binäre Positionsschalter) und 11 Aktuatoren (Relais für die Antriebsmotoren), welche durch insgesamt sieben *Slaves* über das *ASI*-Kabel mit dem *ASI-Master* verbunden sind. Der *ASI-Master* wird vollständig auf der SPYDER-CORE-P1 Basis-Platine emuliert.

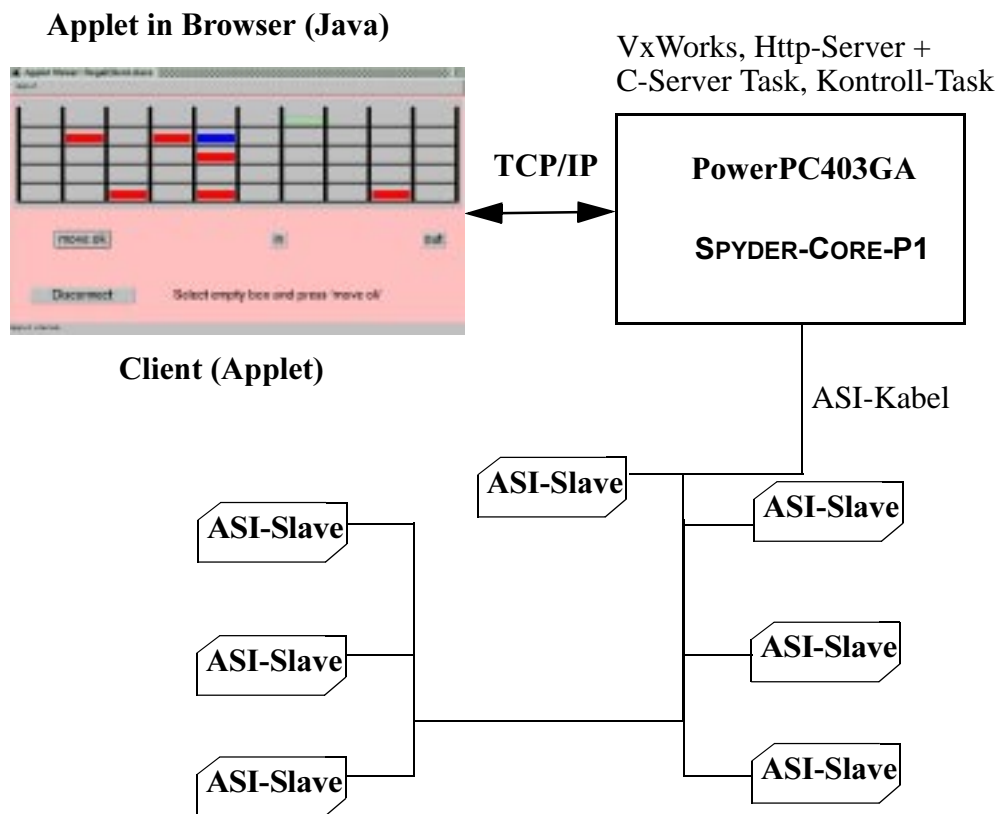


Abbildung 6.18 Architektur des ASI-Demonstrators

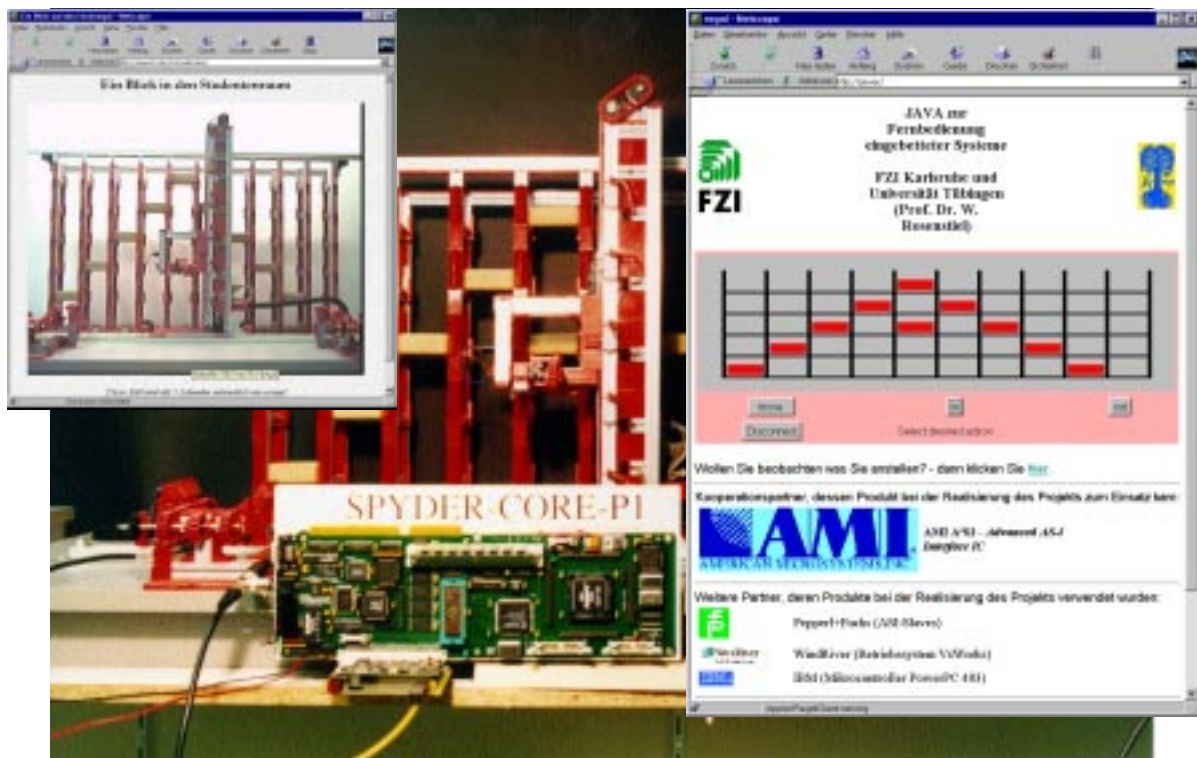


Abbildung 6.19 ASI-Demonstrator: Modell eines Hochregallagers

Ein Benutzer kann sich beim Internet-fähigen *ASI-Master* mit Hilfe eines Browsers anmelden. Dazu braucht er die entsprechende *Web-Adresse* (<http://www.fzi.de/jever>), eine Benutzer-Kennung und das Passwort. Nach dem *Login* überträgt der *http-Server* von *VxWorks* das *JAVA-Applet* zur Bedienung, welches in dem *Flash-Speicher* der Basis-Platine zusammen mit dem Betriebssystem und der Anwendungssoftware gespeichert ist.

Die Abbildung 6.19 zeigt im Hintergrund das reale Modell der Anlage mit der Emulationsumgebung SPYDER-CORE-P1. Im rechten Vordergrund ist die *Web-Seite* dargestellt, welche der Benutzer nach dem *Login* zur Bedienung erhält. Im linken oberen Bereich ist eine weitere *Web-Seite* dargestellt, in der das Modell zu sehen ist, welches mit einer *Web-Kamera* aufgenommen wurde. Dadurch kann der Benutzer die Ausführung seiner Kommandos über das *Internet* verfolgen. Diese Funktionalität ist die Grundlage zur Implementierung weiterer Eigenschaften wie z.B. Ferndiagnose und Fernwartung, welche für die industrielle Automation in der Zukunft wesentliche Bedeutung erlangen werden. Diese Ansätze wurden insbesondere in [HeWWRo98] veröffentlicht.

6.2.6 Zusammenfassung des Anwendungsbeispiels *ASI*

In diesem Anwendungsbeispiel wurde im wesentlichen die Leistungsfähigkeit der Emulationsumgebung SPYDER demonstriert. Da im Bereich der Hardware wie auch der Software ein Eigenentwurf durchzuführen war, kam nur die zweite Stufe der Methodik zur Anwendung. Betrachtet man den Entwurfsablauf in Abbildung 5.1, so wurde der rechte Pfad „Software-Entwurf“ durchlaufen. Parallel dazu wurde aber auch ein umfangreicher „Hardware-Entwurf“ in Form der *ASI-Schnittstellen-Hardware* durchgeführt, wobei gleichzeitig der linke Pfad durchlaufen wurde. Damit konnte gezeigt werden, daß die parallele Emulation von Hardware und Software durchführbar ist.

Die SPYDER-CORE-P1 Basis-Platine wurde eingesetzt, um das Verhalten des weitverbreiteten Echtzeitbetriebssystems *VxWorks* an einem praxisrelevanten Beispiel eines *ASI-Masters* zu untersuchen.

Begonnen wurde mit einer initialen Hardware/Software-Partitionierung, welche in einem ersten Schritt vom Entwickler festgelegt wurde. Im zweiten Schritt wurde das eingebettete System unter Echtzeitbedingungen emuliert. Dabei konnte sehr frühzeitig im Entwurfsablauf ein sehr detaillierter Einblick in das komplexe interne Systemverhalten gewonnen werden. Die wichtigsten Erkenntnisse waren:

- Die minimale Taktfrequenz des Mikrocontrollers, bei der alle Echtzeitbedingungen gerade noch eingehalten werden, konnte mit 33 MHz ermittelt werden.
- Bei diesem Arbeitspunkt verbraucht das Echtzeitbetriebssystem ca. 77% der Rechenleistung des Mikrocontrollers unter *worst-case*-Bedingungen (ohne Caches).
- Beim Zuschalten der Caches wird eine Leistungssteigerung von ca. 40% erreicht, welche für nicht Echtzeit-kritische Systemfunktionen eingesetzt werden kann (z.B. *Internet-Kommunikation*).

Die bis zu diesem Punkt erarbeiteten Erkenntnisse wurden unter Federführung des Autors in [WeSNRo99] veröffentlicht. Es wurde weiter motiviert, warum diese initiale Hardware/Software-Partitionierung durch eine zweite Iteration optimiert werden muß. Für diese verbesserte Hardware/Software-Partitionierung wurde die *Int_Service-Task* von der Software in die Hard-

ware verschoben. Die neue *ASI*-Schnittstellen Hardware-Architektur basiert auf *SRAM*-Speicher. Zur effizienten Implementierung wurden verschiedene *FPGA*-Architekturen evaluiert. Für diese Anwendung im Bereich bis 10.000 Gatter mit einigen 100 Bit *on-Chip-SRAM* wurde gezeigt, daß die *XC4000*-Architektur aufgrund ihrer feingranularen *on-Chip-SRAM*-Fähigkeit signifikante Vorteile gegenüber der weitverbreiteten *Antifuse*-Architektur bzw. der *XC6000*-Architektur besitzt. Dieser zweite Teil der Arbeit wurde ebenfalls unter Federführung des Autors in [WeStRo99] veröffentlicht.

6.3 Vergleich mit der in der Praxis eingesetzten Methode

Im Rahmen der Validierung der Entwurfsmethodik und der Emulationsumgebung *SPYDER* wurde intensiv mit Entwicklern in der Industrie zusammengearbeitet. Im folgenden Text werden industrielle Anwendungen diskutiert, die sehr ähnliche Aufgabenstellungen wie in den beiden Beispielen in Abschnitt 6.1 und Abschnitt 6.2 verfolgen. Es wird gezeigt, welchen Einfluß die in dieser Arbeit vorgestellte Methodik auf die Ergebnisse dieser Arbeiten hat.

6.3.1 Industrielle Erfahrungen in der Kommunikationstechnik

Während der Entwicklung des in Abschnitt 6.1 vorgestellten *ATM*-Diagnose-Monitors wurde mit der Firma Hilan GmbH in Karlsruhe ein intensiver Erfahrungsaustausch durchgeführt. Deren Entwickler arbeiteten zur gleichen Zeit an einem eingebetteten System, dessen Architektur in Abbildung 6.20 dargestellt ist.

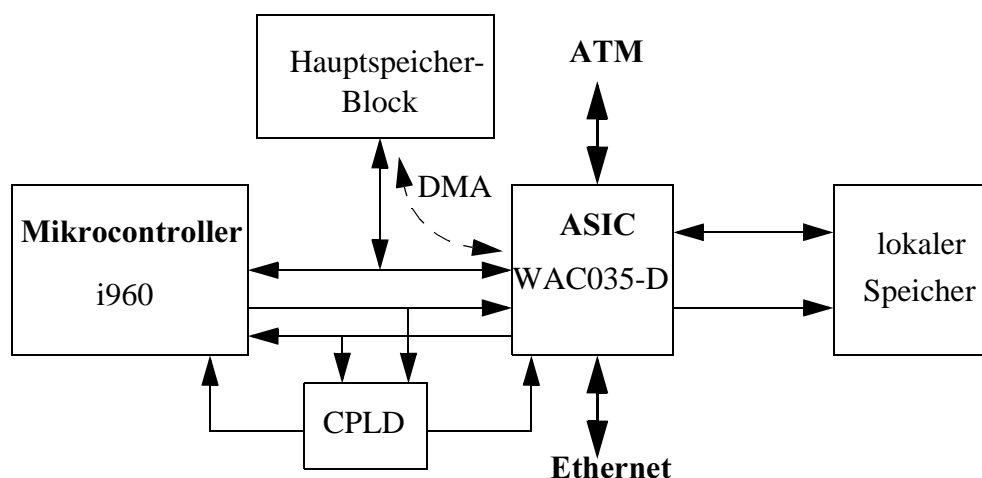


Abbildung 6.20 Eingebettete System-Architektur der Firma Hilan

Das eingebettete System hat die Aufgabe, ein *ATM*-Netz auf ein *Ethernet*-Netz umzusetzen. Dazu benutzen die Entwickler einen speziellen *ASIC* (*WAC035*, Version D), der laut Benutzerhandbuch in der Lage ist, mit mehreren Mikrocontrollern direkt zu kommunizieren. Dabei wurde explizit der *MPC860* sowie der *i960* angegeben.

Aus einem Vorgänger-Projekt hatten die Entwickler den *WAC035* in der älteren *C*-Version in Verbindung mit dem *MPC860* erfolgreich eingesetzt und damit wertvolle Erfahrung gesammelt. Deswegen beurteilten sie den hier vorliegenden Fall mit einem *i960*-Mikrocontroller als relativ problemlos und gingen praktisch von einem Wiederverwendungsfall aus. Aufgrund der bisherigen Erfahrung legten die Entwickler keine zusätzlichen Anforderungen an die Testbar-

keit und akzeptierten die Test-Möglichkeiten, welche der *ASIC on-Chip* zur Verfügung stellt. Nach der Klassifikation von Abschnitt 4.3.5 ist diese Testbarkeit mit „Schnittstellen-Testunterstützung - Kategorie ST“ bezeichnet.

Der Entwurfsablauf wurde, wie in der Praxis weitverbreitet, nach dem Diagramm in Abbildung 3.2 durchgeführt, d.h. es wurde zuerst ein kompletter Hardware-Prototyp entwickelt und produziert, anschließend die Firmware bzw. Software-Entwicklung durchgeführt.

Bei der Systemintegration konnten *Burst-Transfers* im *DMA-Mode* zwischen dem *WAC035* und dem Hauptspeicherblock nicht zur Funktion gebracht werden. Dieser *Burst-DMA-Mode* ist allerdings besonders wichtig, um die Leistungsanforderungen der Spezifikation zu erfüllen.

Die Fehleranalyse unter Echtzeitbedingungen war relativ schwierig, da die Entwickler dafür keine entsprechende Testbarkeit auf der Platine vorgesehen hatten. Nach ca. einem Mann-Monat für die Fehleranalyse konnte die Ursache lokalisiert werden. Der *i960* hat im Gegensatz zum *MPC860* einen externen Bus, welcher im Multiplex-Verfahren zwischen den Adressen und Daten umgeschaltet wird. Damit hat sich neben dem Mikrocontroller auch die Art der Buschnittstelle zwischen Mikrocontroller und *ASIC* geändert. Wenn der *DMA-Controller* des *WAC035* als *Master* den Mikrocontroller-Bus übernimmt, muß er am Ende des *Burst-Zugriffes* zwischen letzter Datenphase und erster neuer Adressphase einen Wartezyklus einhalten, damit die Bustreiber beim Multiplex-Verfahren genügend Zeit haben, um die Richtung der Treiber umzuschalten. Dieser Wartezyklus wird auch laut Benutzerhandbuch zugesichert, wobei sich während der Analyse mit dem Logik-Analysator dieser Wartezyklus nicht nachweisen ließ.

Der Hersteller erklärt dazu, daß ihm während der Entwicklungsphase der neueren Version D gegenüber der älteren Version C ein Entwurfsfehler unterlaufen ist, der diesen Wartezyklus nicht einhält. Dadurch kommt es zu elektrischen Buskonflikten, die die Daten auf dem Bus zerstören. Er empfiehlt weiter, den *Burst-DMA-Mode* deswegen nicht zu verwenden.

Diese Problemlösung war für die Entwickler nicht ausreichend. Sie arbeiteten an einer eigenen Lösung, bei der sie nach jedem *Burst-Zugriff* dem *DMA-Controller* im *WAC035* die Buskontrolle entzogen, um sie ihm anschließend sofort in einer neuen Arbitrierungsphase wieder zuzuteilen. Dadurch könnte der Wartezyklus praktisch erzwungen werden. Diese Steuerung implementierten sie in einem externen *CPLD*.

Der Erfolg dieser Maßnahme war allerdings nicht befriedigend. Der Zeitverlust bei der Arbitrierung war so groß, daß die Übertragungsbandbreite gegenüber dem fehlerlosen Fall immer noch signifikant unter dem spezifizierten Wert lag. Den gesamten zusätzlichen Testaufwand bezifferten die Entwickler mit ca. 2 Mann-Monaten. Entscheidend für die abschließende Beurteilung aber war, daß das eingebettete System im Endprodukt nicht in der Lage war, die in der Spezifikation festgelegte Leistungsfähigkeit zu erbringen. Aufgrund der Tatsache, daß die Entwickler diesen Fehler erst sehr spät im Entwurfszyklus bemerkten, konnten sie nicht rechtzeitig auf diesen Fall reagieren. Diese Problemstellung ist eine der Hauptgefahrenquellen bei der in der Praxis bisher eingesetzten Methode, wie bereits in Abschnitt 3.2.1 dargestellt.

6.3.1.1 Ergebnis des Architekturentwurfs durch Bewertung

Bei der Anwendung des in Kapitel 4 vorgestellten Architekturentwurfs durch Bewertung würde die erste Stufe der Methodik folgende Ergebnisse für den *WAC035* liefern:

- Die funktionale Bewertung würde ebenfalls wie bei der Firma Hilan die Aufnahme als Implementierungsalternative bestätigen.
- Die Bus-Schnittstelle würde mit der Bewertung „idealer Fall (I)“ belegt, da der Hersteller die volle Kompatibilität zum selektierten Mikrocontroller i960, welcher die Randbedingung für die Bus-Schnittstelle vorgibt, im Benutzerhandbuch publiziert (siehe Abschnitt 4.3.2).
- Das Entscheidungsfeld Initialisierung würde mit der Bewertung „mittlerer Aufwand (M)“ belegt (siehe Tabelle 4.2). Beim hier vorliegenden Anwendungsfall liegt die getestete Firmware zwar vor, aber für einen anderen Mikrocontroller. Zusätzlich hat sich auch der Betriebsmode des *WAC035* sowie seine Versions-Bezeichnung geändert.

Die Gesamtbewertung des logischen Pfades in Tabelle 6.6 ergibt einen mittleren Aufwand. Darin ist berücksichtigt, daß alle Änderungen zusammen als „Gefahrenpotential“ erkannt werden und mit einem bereits signifikant gestiegenen Entwurfsrisiko eingestuft werden.

Alternative <i>ALC</i> -ASIC	VAC035-D
Funktionalität	ja
Busschnittstelle	I
Initialisierung	M
Bewertung: Logischer Pfad	M
Technologie	M
Testbarkeit	M
Bewertung: Implementierungsspezifischer Pfad	G

Tabelle 6.6: Bewertung der *ASIC*-Komponente im Hilan-Projekt

Die Bewertung des logischen Pfades ergibt bei der Einstufung des implementierungsspezifischen Pfades eine gewisse Rückwirkung, die wie folgt interpretiert werden kann:

- Bei der Bewertung der Technologie muß der *WAC035* als *Ball-Grid-Array* mit über 500 Pins implementiert werden. Die Spezifikation ermöglicht dazu die Verwendung der Höchstintegration (Kategorie **H**). Der Implementierungsaufwand für den *WAC035* wird laut Tabelle 4.3 mit „mittlerem Aufwand (M)“ bewertet.
- Die Bewertung des logischen Pfades in Verbindung mit der technologischen Kategorie **H** sind für den Entwickler ein Hinweis, auch bei der Testbarkeit eine höhere Anforderung zu stellen, um diesem Entwicklungsrisiko zu begegnen. Es liegt nahe, bei solchen komplexen *ASIC*-Komponenten eine echtzeitfähige Testbarkeit zu fordern, welche in Abschnitt 4.3.5 mit der Kategorie **E** bezeichnet wird. Wenn man diese Testbarkeit als Randbedingung vorgibt, steigt der Aufwand für die Testbarkeit nach Tabelle 4.4 auf „mittlerer Aufwand“, da die *on-Chip*-Testeigenschaften der Komponente nur das Lesen und Schreiben der Register-Schnittstelle erlaubt.

Mit dieser Vorgehensweise ergibt sich für die Gesamtbewertung des implementierungsspezifischen Pfades die Bewertung „großer Aufwand (G)“ (siehe auch Tabelle 4.6).

Entscheidend bei der in dieser Arbeit vorgestellten Methodik ist, daß in der zweiten Stufe diese Bewertung durch Emulation überprüft wird. Damit kann das Entwicklungsrisiko in einer

sehr frühen Entwurfsphase geklärt werden. Zunächst muß eingeräumt werden, daß für den *WAC035* auch eine kleine Trägerplatine produziert werden müßte, um ihn der Emulation zugänglich zu machen. Dieser Mehraufwand ist aber akzeptabel, wenn man bedenkt, welche Auswirkungen ein unerkanntes Entwurfsproblem bewirken kann.

Zur Emulation eignet sich hier insbesondere das Werkzeug SPYDER-ASIC-X2. Der *PCI-Chip* ist in der Lage, auch einen im Multiplex-Verfahren betriebenen Mikrocontroller-Bus, wie beim *i960*, nachzubilden. Der benötigte Hauptspeicher-Block ist ebenfalls auf dem Werkzeug vorhanden. Der *WAC035* muß mit Hilfe einer Trägerplatine dem Werkzeug zugeführt werden. Mit diesem Emulationsaufbau kann auf dem Entwicklungsrechner die benötigte Firmware abgeändert und getestet werden, ohne daß dafür das restliche eingebettete System vorhanden sein muß.

6.3.2 Industrielle Erfahrungen in der Automation und im Automotive-Bereich

Anwendung in der industriellen Automation

Eine sehr fruchtbare Zusammenarbeit entwickelte sich mit der Firma *American Microsystems, INC (AMI)* mit Sitz in Dresden. Diese Firma ist spezialisiert auf den Entwurf und die Produktion von sogenannten *Mixed-Signal-ASIC*. Die Firma erhielt von einem Firmenkonsortium, bestehend aus mehreren mittelständischen Unternehmen, den Auftrag, einen *ASIC* für das bereits vorgestellte *ASI-Master-System* zu entwerfen. Nach konstruktiven Gesprächen entschied sich die Firma, den in *VHDL* beschriebenen *ASIC* nicht nur zu simulieren, sondern mit Hilfe der Emulationsumgebung SPYDER zu emulieren. Das Werkzeug SPYDER-ASIC-X1 wurde benutzt, um den digitalen Teil des *ASIC*, bestehend aus ca. 3.000 Gattern, zu emulieren. Der analoge Teil wurde in Form einer dedizierten elektronischen Schaltung an das Werkzeug adaptiert. Dadurch konnte der gesamte Chip auf dem Emulator nachgebildet werden. Die Abbildung 6.21 zeigt das *Layout* des Chips nach dem ersten Durchlauf durch die Fertigung.

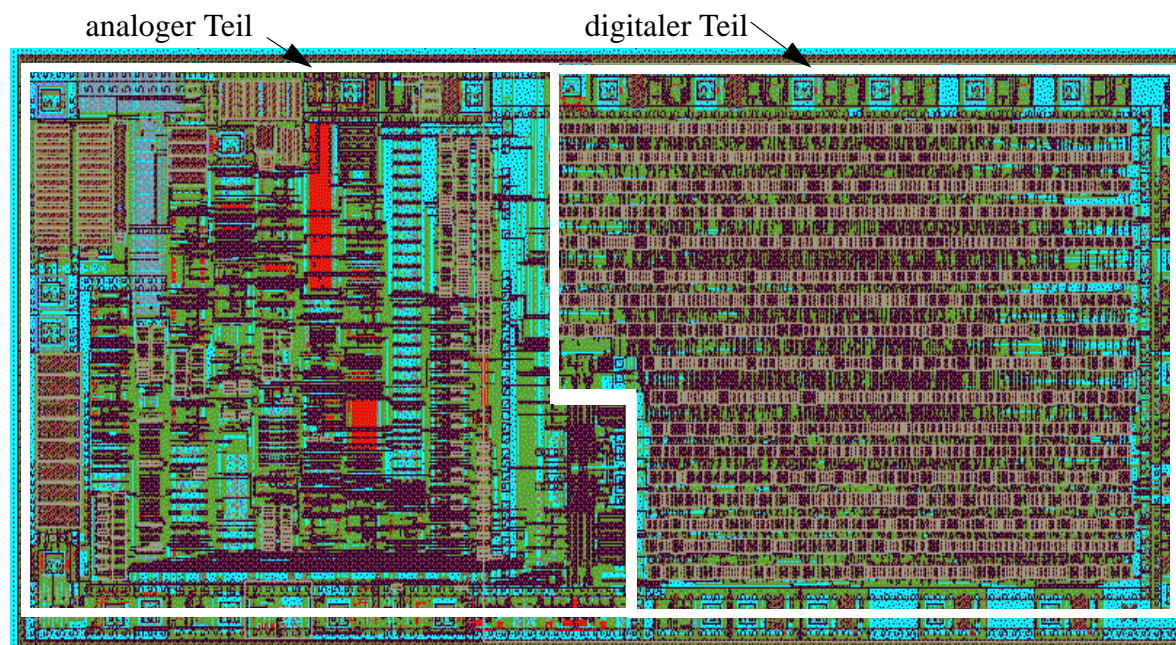


Abbildung 6.21 Layout des Advanced A²SI-ASIC

Die Analyse der Ergebnisse des sogenannten „*first silicon*“ erbrachte die folgende Bewertung bezüglich der angewandten Entwurfsmethode und der Werkzeuge durch die Firma *AMI*, welche in Form eines schriftlichen Berichtes vorliegt und hier zitiert wird (Dr. Thierfelder):

„Durch die Anwendung der in dieser Arbeit formulierten Entwurfsmethode konnten Fehler im Vorfeld erkannt werden, die mit Hilfe der Simulation nicht entdeckt wurden. Dadurch wurde mindestens eine Entwurfsiteration eingespart. *AMI* veranschlagt dafür ca. \$50.000. Zusätzlich wurden noch 2-3 Mann-Monate Entwicklungszeit eingespart, welche *AMI* nochmals mit ca. \$20.000 angibt. Zusammen konnten Kosten in Höhe von ca. \$70.000 oder 120.000DM eingespart werden.“

Der bei der Kooperation entstandene Erfahrungsaustausch hat bereits beiden Seiten einen wesentlichen Nutzen erbracht. Dabei wurde der *ASI*-Chip in der *ASI*-Demonstrationsanlage (siehe Abbildung 6.18) eingebaut und evaluiert. *AMI* hat im Gegenzug als industrieller Anwender die Entwurfsmethodik benutzt und die Werkzeuge des *SPYDER*-Systems im industriellen Umfeld getestet. Der in Kapitel 4 publizierte Ansatz wurde dabei als sehr hilfreich eingestuft. Die Firma *AMI* machte zusätzlich wesentliche Verbesserungsvorschläge bei der industriellen Handhabung der Werkzeuge, welche bei der Weiterentwicklung in Kapitel 5 berücksichtigt wurden. Darüber hinaus ist *AMI* auch zukünftig bereit, die entstehenden Methoden und Werkzeuge der in dieser Arbeit publizierten Entwurfsmethode im industriellen Umfeld zu evaluieren.

Der aktuelle Stand der Zusammenarbeit war die Präsentation der Entwurfsmethodik und der Werkzeuge am Beispiel dieses *ASI-ASIC* auf der *electronica*-Messe vom 11. bis 13. November 1998 in München. Die Firma ermöglichte dem Autor eine Präsentation der Ergebnisse auf dem *AMI*-eigenen Stand. Dadurch bestand eine weitere Möglichkeit, die wissenschaftliche Arbeit einem breiten Fachpublikum aus Industrie und Hochschulen zu präsentieren.

Die Ergebnisse bezüglich den Untersuchungen des Echtzeitbetriebssystems *VxWorks* sowie der Ankoppelung des *ASI-Masters* an das Internet fanden insbesondere bei der Herstellerfirma *Windriver Systems* großes Interesse. Sie ermöglichte dem Autor die Präsentation der Ergebnisse auf der Messe „*Embedded Systems*“ im Februar 1999 in Nürnberg, wobei sie ebenfalls ihren eigenen Ausstellungsstand zur Verfügung stellte.

Anwendung im Automotive-Bereich

Diese Präsentation war Ausgangspunkt für eine weitere intensive Zusammenarbeit zwischen der Firma *Windriver* und der Firma *Becker Automotive Systems*. Hier wird zur Zeit die Methodik in einem weiteren großen Industrieprojekt evaluiert. Es wird ein eingebettetes System mit einem 32-Bit-Mikrocontroller entwickelt, welches alle Service-Geräte im Kraftfahrzeug wie z.B. Radio, Navigationssystem (engl.: *Global Positioning System - GPS*) und Mobiltelefon zentral verwaltet und dem Fahrer in übersichtlicher Art und Weise auf einem Display zur Bedienung darstellt. Im Rahmen dieses Projektes wurde das *SPYDER*-System bereits um eine weitere Basis-Platine erweitert, welche einen 32-Bit Mikrocontroller von *Hitachi (SH3-7709A oder SH3-7729-DSP)* besitzt. Sie trägt den Namen *SPYDER-CORE-P2/SH3* und ist mit den bereits eingeführten Werkzeugen *SPYDER-CORE-P1* und *SPYDER-ASIC-X2* kompatibel.

Wie die Firma *Becker Automotive Systems* bestätigte, hat das Werkzeug *SPYDER-CORE-P2/SH3* und die damit verbundene Entwurfsmethodik wesentlich zum Erfolg der Entwicklung eines ersten Vorserien-Produktes innerhalb der gesetzten Meilensteine beigetragen. Die zeitliche Einhaltung der vorgegebenen Meilensteine hätte bei der Anwendung der bisher in der Praxis eingesetzten Methodik aus Abschnitt 3.2.1 nicht erreicht werden können.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

In Kapitel 3 wurden zwei Entwurfsmethoden für eingebettete Systeme eingeführt. Die erste Methode wurde als die in der Praxis eingesetzte Methode bezeichnet, da sie die zur Zeit wohl die am häufigsten angewandte Entwurfsmethode darstellt. Die zweite Methode kommt aus dem Bereich des *Hardware/Software Co-Designs* und steht momentan noch in der wissenschaftlichen Diskussion. Beide Methoden wurden beschrieben und ihre Vor- und Nachteile analysiert.

Dabei wurde gezeigt, daß die in der Praxis eingesetzte Methode sehr zielorientiert ist, wobei die Entwickler aber erst sehr spät im Entwicklungsablauf Informationen über das interne Systemverhalten erhalten. In dieser Tatsache liegt ein signifikantes Entwicklungsrisiko begründet, welches die Entwickler mit einer Einschränkung bei der Bausteinauswahl zu begrenzen versuchen. Es werden dabei primär nur Komponenten eingesetzt, bei denen die Entwickler bereits über eine bestimmte Entwurfserfahrung verfügen.

Die *Hardware/Software Co-Design*-Methode erkennt diese Probleme bezüglich der sehr späten Einsicht in das interne Systemverhalten. Sie versucht, durch eine einheitliche Systembeschreibung bereits frühzeitig Aussagen über das Systemverhalten zu treffen. Gleichzeitig liegen aber auch zwei wesentliche Probleme darin. Zum einen gibt es zur Zeit keine Systembeschreibungssprache, die alle Aspekte eines eingebetteten Systems beschreiben kann. Es müssen viele Beschreibungsmethoden parallel benutzt werden. Die zweite Aufgabe besteht darin, gesicherte und aussagefähige Bewertungskriterien für Hardware- bzw. Software-Komponenten zu gewinnen, um diese für den Entwurf komplexer eingebetteter Systeme ausnutzen zu können. Diese Methode gestaltet sich in ihrer Anwendung deshalb sehr schwierig.

In der im Rahmen dieser Arbeit vorgestellten Methodik „Architekturentwurf und Emulation eingebetteter Systeme“ wird ein zweistufiger Ansatz beschrieben. Dieser Ansatz greift dabei die positiven Eigenschaften der oben diskutierten Methoden auf. Von der zur Zeit in der Praxis eingesetzten Methode wird das zielgerichtete Vorgehen weiter benutzt. Von der *Hardware-*

Software Co-Design-Methode wird das Ziel weiter verfolgt, frühzeitig einen tiefen Einblick in das reale Systemverhalten zu erlangen.

Für die vorgestellte Methodik wurde in Kapitel 4 zunächst der Entwurfsraum festgelegt, welcher den Bereich der industriellen Automation und Kommunikation abdeckt. In der ersten Stufe wird versucht, anhand objektiver Bewertungskriterien die Eignung einer Komponente für das spezifizierete eingebettete System auszuwählen. Dabei wurde die Vorgehensweise eines Entwicklers analysiert und in einen Auswahlalgorithmus überführt.

Es wird von einem Mikrocontroller-getriebenen Ansatz ausgegangen. Das bedeutet, für das festgelegte Anwendungsgebiet existieren Mikrocontroller, die für diese Anwendungen optimiert sind und deswegen mit Priorität eingesetzt werden müssen. Sie geben insbesondere ein bestimmtes Busverhalten vor, an das sich alle anderen Komponenten anpassen müssen. Ferner wurde gezeigt, daß die Architekturen dieser eingebetteten Systeme aus wenigen, dafür aber hochintegrierten Bausteinen bestehen, in der Regel aus Mikrocontroller, *ASIC* und *FPGA*. Bei der Komponentenauswahl sind insbesondere existierende *ASIC* einem eigenen Entwurf zu bevorzugen, um vom enormen Entwicklungsaufwand dieser *ASIC*-Komponenten weiter zu profitieren und Entwicklungszeit zu sparen.

Insbesondere die *ASIC*-Auswahl wird durch den Auswahlalgorithmus unterstützt. Dabei wurden in Kapitel 4 alle Aspekte, die ein Entwickler bei der Bewertung eines *ASIC* zu berücksichtigen hat, in einzelne Entscheidungsfelder zusammengefaßt. Dabei kristallisierten sich ein logischer und ein implementierungsspezifischer Pfad heraus. Jedes Entscheidungsfeld besitzt dazu bestimmte Entscheidungsparameter, deren sequentielle Betrachtung eine Entscheidung erlaubt, ob das Entscheidungsfeld mit der Spezifikation unter Einhaltung der Randbedingungen in Übereinstimmung gebracht werden kann. Als Grundlage der Bewertung wird der relative Aufwand angegeben, den die Verwendung der Komponente während des Entwicklungsablaufes mit sich bringt. Abschließend werden alle Entscheidungsfelder eines Pfades in einer Gesamtbewertung zusammengefaßt. Durch eine systematische Anwendung des Algorithmus kann eine *ASIC*-Komponente aus einer Menge von Alternativen bestimmt werden, die sich besonders für einen Einsatz eignet.

In einem zweiten Schritt werden die einzelnen Entscheidungsfelder durch Emulation überprüft. Der gesamte Entwurfsablauf der vorgestellten Methode wurde in Abbildung 5.1 graphisch dargestellt. Um diesen Entwurfsablauf effizient anwenden zu können, wurde das Emulationssystem SPYDER entwickelt, welches in Kapitel 5 eingeführt wurde. Es besteht aus zwei Werkzeugen. Die SPYDER-CORE-P1 Basis-Platine unterstützt im wesentlichen den parallelen Entwurfsablauf von Software und Hardware, sowie die Systemintegration und den Systemtest. SPYDER-ASIC-X2 ist als untergeordnetes Werkzeug zur singulären Emulation eines *ASIC* zu betrachten. Hauptziel von SPYDER ist die Emulation unter Echtzeit-Anforderungen unter möglichst realistischen Umweltbedingungen.

In Kapitel 6 wurde die vorgestellte Methodik sowie die Benutzung der Werkzeuge anhand zweier praxisrelevanter Beispiele demonstriert. Beim ersten Beispiel ATM-Diagnose-Monitor steht die erste Stufe der Methodik des systematischen Architekturentwurfes durch Bewertung von *ASIC*-Komponenten im Vordergrund. Zusätzlich wurden verschiedene *FPGA*-Architekturen bewertet, um eine weitere Teilfunktion des Anwendungsbeispiels, welche als *ASIC* nicht zur Verfügung steht, effizient implementieren zu können.

Im zweiten Anwendungsbeispiel *ASI-Master* steht keine *ASIC*-Komponente zur Verfügung, weshalb hier die erste Stufe nicht zur Anwendung kommt. Hauptaufgabe dieses Anwen-

dungsbeispiels ist es, die zweite Stufe der Methodik, die Emulation und die Emulationsumgebung SPYDER zu demonstrieren. Dafür wurde ein umfangreicher Hardware- und Software-Entwurf durchgeführt. Zum Einsatz kommt ein Echtzeitbetriebssystem, dessen Verhalten intensiv mit Hilfe der Emulation untersucht wurde. Es wurde gezeigt, wie durch den Einsatz von SPYDER ein frühzeitiger und tiefer Einblick in das interne Systemverhalten zu erreichen ist. Zusätzlich konnte durch eine iterative Verfeinerung der Hardware-Software-Partitionierung eine wesentliche Leistungssteigerung und Optimierung des eingebetteten Systems erzielt werden.

7.2 Ausblick

Betrachtet man den Fortschritt bei mikroelektronischen Komponenten in den letzten Jahren, so zeigt sich ein enormer Innovationsschub. Neue Generationen elektronischer Komponenten erscheinen alle sechs Monate neu auf dem Markt, wobei ihre Leistung jedes Jahr um ca. 50% steigt. Der Grund dafür liegt in den hochentwickelten Entwurfsmethoden, die bei der *High-Level*-Synthese beginnen und bis hinunter zu den einzelnen Strukturen für die Transistoren auf dem Chip reichen. Dabei ist auch zu erkennen, daß exakt definierte Entwurfsmethoden die Grundlage für die Entwicklung von entsprechenden *CAD*-Werkzeugen bilden, deren Einsatz diesen Innovationsschub ermöglichen. Eine weitere Erkenntnis liegt auch darin, daß hier zwei Ebenen, die unterste Transistorebene und die darüberliegende Architekturentwurfsebene für die Chips, eng miteinander gekoppelt sind.

Verläßt man die Ebene der einzelnen mikroelektronischen Komponenten und geht noch eine Stufe höher zur Systemebene, auf der eingebettete Systeme entworfen werden, so ist keine eindeutige, wohldefinierte und vor allem nach unten durchgängige Entwurfsmethodik zu erkennen. Es existieren zwar viele einzelne Methoden, die bestimmte Teilaspekte des Entwurfes eingebetteter Systeme adressieren, aber keine Methode, die den Architekturentwurf in ähnlich effizienter Weise unterstützt wie die Methoden einer Stufe tiefer beim Entwurf mikroelektronischer Komponenten. Die Folge dieser „methodischen Lücke“ einer exakt definierten und durchgängigen Methodik auf Systemebene ist, daß der oben erwähnte Innovationsschub sich hier nicht in gleichem Tempo fortsetzt. Als Beiweiß dafür wird in [Gupta95] gezeigt, daß die oben angesprochenen mikroelektronischen Komponenten erst nach Jahren in eingebetteten Systemen eingesetzt werden.

Ein wesentlicher Grund für diese Verzögerungen des Innovationsschubes beim Übergang von der Komponenten auf die Systemebene ist der „Informationsstau“. Betrachtet man die Art und Weise, in der der Systementwickler seine Informationen vom Chip-Entwickler übergeben bekommt, wird die Problematik deutlich. Die Funktionalität von *ASIC*-Komponenten, in denen mehrere Dutzend Mann-Jahre Entwicklungszeit und Intelligenz investiert wurden, werden auf bis zu 2000 Seiten Text (z.B. MPC860-Manual) beschrieben und als Benutzerhandbuch oder Datenblatt im Internet veröffentlicht. Um diesen beschriebenen „Informationsstau“ beim Übergang auf die Systemebene aufzulösen, muß es auch hier Methoden geben, die dem Systementwickler die notwendigen Informationen für die Systementwicklung geben, ohne ihn mit Details der internen Funktion der Komponente zu belasten.

An dieser Stelle setzt der in Kapitel 4 beschriebene Ansatz des Architekturentwurfes durch Bewertung von Komponenten an. Die dort eingeführten Entscheidungsparameter können als ein Satz objektiver Bewertungskriterien aufgefaßt werden, welche ein Systementwickler kennen muß, um über eine Komponente und das mit ihr verbundene Entwicklungsrisiko zu entscheiden. Um heutzutage diese Information zu gewinnen, ist für jede Komponente eine

langwierige Analyse der umfangreichen Benutzerhandbücher notwendig, die in einigen Fällen bereits mehr Entwicklungszeit in Anspruch nimmt, als der eigentliche Systementwurf danach. Das Ergebnis der Analyse ist zusätzlich stark von der Erfahrung des Systementwicklers abhängig sowie von seiner Fähigkeit, aus der Vielzahl von Informationen die für den Architektorentwurf wichtigen Informationen herauszufiltern. Die hier vorgestellte Arbeit faßt alle notwendigen Informationen bezüglich der Einsatzfähigkeit einer Komponente systematisch zusammen und führt einen Bewertungsprozeß durch. Zukünftig muß erreicht werden, daß diese Bewertung noch effizienter durchgeführt werden kann. Hauptziel dabei ist es, den in dieser Arbeit vorgestellten Bewertungsprozeß auch als Entscheidungsgrundlage bei der Implementierung entsprechender *CAD*-Werkzeuge zu benutzen.

Auch im Bereich der Emulation werden weitere Arbeiten folgen. Diese Arbeit hat gezeigt, daß die Emulation von eingebetteten Systemen unter Echtzeitbedingungen einen frühen und tiefen Einblick in das interne Systemverhalten erlaubt. Im Bereich der Mikrocontroller hat sich bereits durchgesetzt, daß fast alle Hersteller zu ihrem Chip ein sogenanntes *Evaluation-Board* anbieten, auf dem der Systementwickler erste Erfahrungen mit der Komponente machen kann.

Im wesentlich größeren Markt der *ASIC*-Komponenten ist dieser Zustand noch nicht erreicht. Diese Arbeit hat dazu bereits eine Emulationsumgebung mit definierten Schnittstellen zum Anschluß von *ASIC*-Komponenten präsentiert. Ziel weiterer Arbeiten ist das Bestreben zur Definition eines einheitlichen Standards. Wünschenswert wäre ein Verfahren, bei dem der Hersteller seine Komponente bereits auf einer Trägerplatine basierend auf einem solchen Standard zur Verfügung stellt, ohne daß sich jeder Systementwickler eine eigene Trägerplatine für den zu emulierenden *ASIC*-Baustein entwickeln muß.

Anhang A

Referenzen

- [Act94] ____: *FPGA-Data Book and Design Guide*. Actel, Inc. 1994
- [ASI98] ____: *Aktuator Sensor Interface (ASI) Complete Specification*. Beuth Verlag, Berlin, EN50295, Version 2.1, 1998
- [AMD98] ____: *AMD CMOS 5.0V Uniform Sector Flash Memory*. Publication #21445 <http://www.amd.com>, January 1998
- [Army96] ____: *Real Time Executive for Multiprocessor Systems (RTMS) Development Environment Guide*. U.S. Army Missile Command, Redstone Arsenal, Alabama 35898-5254, Release 3.5.1, January 1996
- [BeKa91] K. Bender, M. Katz: *PROFIBUS - der Feldbus für die Automation*, Carl Hanser Verlag, München, 1991
- [DeMi95] G. De Micheli: *Hardware/Software Co-Design: Application Domains and Design Technologies*. Proceedings of the NATO Advanced Study Institute on Hardware/Software Co-Design, Tremezzo Italy, Kluwer Academic Publishers, 1995
- [ElHu94] J. Eldrege, B. Hutchings: *RRANN: The Run-Time Reconfiguration Artificial Neural Network*. IEEE Custom Integrated Circuits Conference, San Diego, pp 77-80, 1994
- [Fuj96] ____: *MB86687A - Adaption Layer Controller (ALC)*. Data Sheet, Edition 2.0, FUJITSU, 1996
- [Gies97] B. Giese: *Die Basis für die Märkte der Zukunft in der Wissensgesellschaft*. Informations- und Kommunikationstechnik, aktualisierte Übersicht, DLR, IT-GE, November 1997
- [HePa96] J. Hennessy, D. Patterson: *Computer Architecture - A Quantitative Approach*. Second Edition, Morgan Kaufmann Publishers, Inc. SF-CA-USA, 1996
- [HeWWRo98] A. Hergenhan, C. Weiler, K. Weiß, W. Rosenstiel: *Value-Added Services in the Industrial Automation*. ACoS'98, Lisabon Portugal, April 1998
- [HuWi95] B. Hutchings, M. Wirthlin: *Implementation Approaches for Reconfigurable Logic Applications*. Proceedings of 5th International Workshop of Field-Programmable Logic and Applications, Oxford 1995

- [Gor95] W. Goralski: *Introduction to ATM-Networking*. McGraw-Hill Series on Computer Communications, 1995.
- [Gupta95] R. Gupta: *Co-Synthesis of Hardware and Software for Digital Embedded Systems*. Kluwer Academic Publishers Group, 1995
- [IBM94] ___: *The PowerPC Architecture: A Specification for a new Family of RISC Processors*. Morgan Kaufmann Publishers, Inc., San Fransisco, CA, IBM 1994
- [IBM98] ___: *PowerPC 403GCX Embedded Controller User Manual*. <http://www.chips.ibm.com/techlib/products/embedded/manual/>, IBM 1998
- [Intel94] ___: *MCS51 Microcontroller Family User's Manual*. Order Nr.: 272383-002, <http://www.intel.com>, Intel 1994
- [IySh89] V. Iyer, H. Sholl: *Software Partitioning for Distributed, Sequential, Pipelined Applications*. IEEE Transactions on Software Engineering, Vol. 15, No. 10, pp.1270-1279, 1989
- [Kist97] R. Kistner: *Entwurf eines Testgenerators/-Monitors für ATM-Hochleistungsnetze*. Diplomarbeit am Institut für theoretische Elektrotechnik und Meßtechnik, Universität Karlsruhe, 1997
- [KuAy96] S. Kumar, J. Aylor, B. Johnson, W. Wolf: *Embedded Systems - A Unified Hardware/Software Representation*. Kluwer Academic Publishers Group, 1996
- [KuAy93] S. Kumar, J. Aylor, B. Johnson, W. Wolf: *A Framework for Hardware/Software Codesign*. IEEE Computer, Vol. 26, No. 12, pp. 39-45, 1993
- [Mic98/1] ___: *Advance 8Mb pipelined syncburst SRAM 256K x32/36*. Micron, Inc. Datasheet, www.micron.com/mti/msp/html/datasheet.html, 1998
- [Mic98/2] ___: *MT4LC1M16C3 FPM DRAM 1Meg x 16*. Micron, Inc. Datasheet, www.micron.com/mti/msp/html/datasheet.html, 1998
- [Mic98/3] ___: *MT48LC2M32B1TG Advance Synchronous DRAM 64Mb x 32*. Micron, Inc. Datasheet, www.micron.com/mti/msp/html/datasheet.html, 1998
- [Mot98/1] ___: *MPC860 PowerQUICC - User's Manual*. Motorola Inc., Revision 1.0, <http://www.motorola.com>, July 1998
- [Mot98/2] ___: *Power Risc Control - MPC555 User's Manual*. Motorola Inc., <http://www.motorola.com>, September 1998

- [Mot94] ____: *Power Risc Control - MPC500 Family - RCPU Reference Manual*. Motorola Inc., <http://www.motorola.com>, 1994
- [MöLu87] A. Möschwitzer, K. Lunze: *Halbleiterelektronik*. 7. Auflage, Hüthig Verlag, Heidelberg, 1987
- [Par98] ____: *PDM41024 - 1 Megabit Static RAM 128k x 8 Bit*. PARADIGM Datasheet Rev. 3.3, 1998
- [Sie96] ____: *C167 16 Bit CMOS Single-Chip Microcontroller - User's Manual*. Siemens AG, Version 2.0, 1996
- [Tan94] A. Tanenbaum: *Moderne Betriebssysteme*. Hanser Verlag München, 1994
- [WeHeRo97] K. Weiß, A. Hergenhan, W. Rosenstiel: *Systematischer Entwurf und Test eingebetteter Systeme am Beispiel eines ATM-Diagnosesystems*. Workshops zur Architektur von Rechnersystemen, ARCS, Rostock 1997
- [WeKiRo98] K. Weiß, R. Kistner, W. Rosenstiel: *Analysis of the XC6000 Architecture for Embedded System Design*. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa Valley CA, April 1998
- [WeOe98] K. Weiß, C. Oetker: *SPYDER-ASIC-XI User Manual*. <http://www.fzi.de/weiss.html>, FZI Karlsruhe 1998
- [WeSt98] K. Weiß, T. Steckstor: *SPYDER-CORE-P1 User Manual*. <http://www.fzi.de/weiss.html>, FZI Karlsruhe 1998
- [WeSNRo99] K. Weiß, T. Steckstor, C. Nitsch, W. Rosenstiel: *Performance Analysis of Real-Time-Operation Systems by Emulation of an Embedded System*. 10th IEEE International Workshop on Rapid System Prototyping (RSP), Clearwater, Florida, USA, 1999
- [WeStRo99] K. Weiss, T. Steckstor, W. Rosenstiel: *Exploiting FPGA-Features during the Emulation of a Fast Reactive Embedded System*. ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA), Monterey, CA, USA, 1999
- [Wind97] ____: *Tornado User's and Programmer's Guide*. Windriver Systems, Inc. 1997
- [WirHut97] M. Wirthlin, B. Hutchings: *Improving Functional Density Through Run-Time Constant Propagation*. ACM-Symposium on Field Programmable Gate Arrays, 1997.

- [Wo94] W. Wolf: *Hardware-Software Co-Design of Embedded Systems*. Proceedings of the IEEE, Vol. 82, No. 7, July 1994
- [Xil99] ____: *Virtex 2,5V Field Programmable Gate Arrays*. Xilinx, Inc. Datasheet, <http://www.xilinx.com>, 1999
- [Xil98/1] ____: *The Programmable Logic Data Book*. Xilinx, Inc. 1998
- [Xil96] ____: *XC6000 Field Programmable Gate Array*. Xilinx, Inc. Datasheet, <http://www.xilinx.com>, 1996

Anhang B

Veröffentlichungen, die im Rahmen dieser Arbeit entstanden sind

Nationale Veröffentlichungen

[WeHeRo97] K. Weiß, A. Hergenhan, W. Rosenstiel: *Systematischer Entwurf und Test eingebetteter Systeme am Beispiel eines ATM-Diagnosesystems*. Workshops zur Architektur von Rechnersystemen, ARCS, Rostock 1997

Internationale Veröffentlichungen

[WeKiRo98] K. Weiß, R. Kistner, W. Rosenstiel: *Analysis of the XC6000 Architecture for Embedded System Design*. IEEE Symposium on Field Programmable Custom Computing Machines (FCCM), Napa Valley CA, April 1998

Diese Arbeit wurde auch als Poster-Präsentation auf der ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA), Monterey, CA, USA, 1998 gezeigt.

[WeSNRo99] K. Weiß, T. Steckstor, C. Nitsch, W. Rosenstiel: *Performance Analysis of Real-Time-Operation Systems by Emulation of an Embedded System*. 10th IEEE International Workshop on Rapid System Prototyping (RSP), Clearwater, Florida, USA, 1999

Diese Arbeit wurde auch als Poster-Präsentation auf der DATE 99 in München gezeigt.

[WeStRo99] K. Weiss, T. Steckstor, W. Rosenstiel: *Exploiting FPGA-Features during the Emulation of a Fast Reactive Embedded System*. ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA), Monterey, CA, USA, 1999

Weitere Präsentationen

Diese Arbeit wurde teilweise unterstützt mit Mitteln der Deutschen Forschungsgemeinschaft unter der Bezugsnummer 3221040 innerhalb des Schwerpunktprogramms „Entwurf und Entwurfsmethoden für eingebettete Systeme“

Im Rahmen des Zwischenseminars im Oktober 1998 in München wurde die Emulationsumgebung SPYDER von den Gutachtern der DFG mit dem ersten Preis „*Best application potential Award*“ ausgezeichnet.

Lebenslauf

Name: Karlheinz Weiß
geboren am: 30.05.1964
Geburtsort: Kandel, Rheinland-Pfalz

Schulausbildung
01.08.1970-30.06.1974 Grundschule in Minfeld, Rheinland-Pfalz
01.08.1974-31.05.1980 Staatliche Realschule in Kandel, Rheinland-Pfalz
01.09.1980-15.06.1982 Berufsfachschule Elektrotechnik in Bad Bergzabern, Rheinland-Pfalz, Abschluß: Mittlere Reife

Berufsausbildung
01.09.1982-12.07.1983 Ausbildung beim Heeresinstandsetzungswerk 870 der Bundeswehr, Rheinland-Pfalz, Abschluß: Nachrichtengerätetechniker
13.07.1983-05.07.1984 Ausbildung beim Heeresinstandsetzungswerk 870 der Bundeswehr, Rheinland-Pfalz, Abschluß: Funkelektroniker

Hochschulreife
01.09.1984-24.06.1986 Carl-Engler-Schule Karlsruhe, Technische Oberschule, Abschluß: Fachgebundene Hochschulreife

Universitätsausbildung
15.10.1986-09.09.1992 Technische Hochschule Karlsruhe, Studiengang Elektrotechnik, Studienrichtung: Integrierte Schaltungen
Abschluß: Diplom-Ingenieur

Industrietätigkeit
28.09.1992-13.11.1992 Deutsche Aerospace, Ottobrunn b. München, Bayern, Werkstudent
Entwicklung 8051-basierter eingebetteter Systeme
01.03.1993-30.07.1995 Mitarbeit als Diplomingenieur bei Hyperstone electronics GmbH in Konstanz, Aufgabengebiet: Logikentwurf für die ASIC-Entwicklung E1-32
Entwicklung von hyperstone-E1-32-Mikrocontroller-basierten eingebetteten Systemen

Wissenschaftliche Arbeit
seit dem 01.08.1995 Wissenschaftlicher Mitarbeiter am Wilhelm-Schickard-Institut für Informatik der Universität Tübingen und am Forschungszentrum Informatik (FZI) in Karlsruhe