Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association suisse des électriciens, de l'Association des entreprises électriques suisses

Herausgeber: Schweizerischer Elektrotechnischer Verein ; Verband Schweizerischer Elektrizitätsunternehmen

Band: 88 (1997)

Heft: 3

Artikel: Hardware/Software-Codesign : Massgeschneiderte elektronische Systeme : Teil 2 : HW/SW-Synthese

Autor: Teich, Jürgen

DOI: https://doi.org/10.5169/seals-902179

Nachdem im ersten Teil dieses Beitrages (Bulletin SEV/VSE 25/1996) die Problemstellung des Hardware/Software-Codesigns, die in Frage kommenden Architekturen und Realisierungsformen sowie Berechnungsmodelle, Spezifikationssprachen und Entwurfspraktiken vorgestellt wurden, hat der vorliegende zweite Teil den eigentlichen Entwurfsablauf zum Thema. Ein grosser Teil der nachstehenden Ausführungen befasst sich mit der Partitionierung und Optimierung des Entwurfs. Diese Entwurfstätigkeiten sind von grösster Bedeutung, bestimmen sie doch ganz direkt die Fähigkeiten und den Preis der zukünftigen Produkte.

Hardware/Software-Codesign: Massgeschneiderte elektronische Systeme

Teil 2: HW/SW-Synthese

■ Jürgen Teich

4. HW/SW-Synthese

4.1 Entwurfsablauf

Der typische Entwurfsablauf bei der Entwicklung eines HW/SW-Systems ist in Bild 10 dargestellt. Ausgehend von der Spezifikation des Systemverhaltens erfolgt die Aufteilung der zu implementierenden Funktionalität in Hardware- und Softwarefunktionalität. Man spricht in diesem Zusammenhang auch von der *Hardware/Software-Partitionierung*.

Man sollte sich spätestens an dieser Stelle den Unterschied zwischen Hardware- und Softwarerealisierung vergegenwärtigen. Zum einen läuft zwar keine Software ohne Hardware (der Prozessor realisiert eine gewünschte Funktionalität durch Ausführung eines Programms), zum anderen aber wird die Unterscheidung zwischen Hard- und Softwarerealisierung schwammig, wenn man zum Beispiel an programmierbare Hardwarebausteine (z. B. PLA, FPGA) denkt. Dem Baustein wird die Funktionalität der Schaltung, die er realisieren soll, durch Programmierung (Software) aufgeprägt und lässt sich sogar häufig während des Betriebs umprogrammieren. In dieser Hinsicht ist die Funktionalität ebenfalls in Software realisiert (als Code, der die Gatterschaltung konfiguriert). Man kann also von einer Programmierung auf Gatterebene sprechen (siehe auch Bild 6 in Teil 1). Ausserdem werden selbst komplexe Asic heute mit Hilfe von Programmiersprachen (Hardwarebeschreibungssprachen, zum Beispiel VHDL, Verilog) beschrieben und synthetisiert. Wir treffen daher die übliche, jedoch nicht fest definierte Konvention, dass man von einer Implementierung einer Funktionalität in Hardware genau dann redet, wenn der Entwerfer bildlich eine Schaltung vor Auge hat, welche die Funktionalität (und nur diese) implementiert, und von Software, wenn das beabsichtigte Endstadium der Synthese eine Beschreibung ist, die als Programm von einer Maschine ausgeführt wird.

Im Verlauf der Hardware/Software-Partitionierung wird nun die Entscheidung getroffen, welche Funktionalität in Software und welche in eine Schaltung verfeinert wird. Später werden wir die wesentlichen Anforderungen und Aufgaben der Hardware/Software-Partitionierung näher betrachten.

Ausgehend von dieser Partitionierung, werden die Teilspezifikationen dann durch Synthesewerkzeuge verfeinert. Für

Adresse des Autors
PD Dr.-Ing. *Jürgen Teich*, Institut für Technische Informatik und Kommunikationsnetze (TIK), ETH Zürich, 8092 Zürich

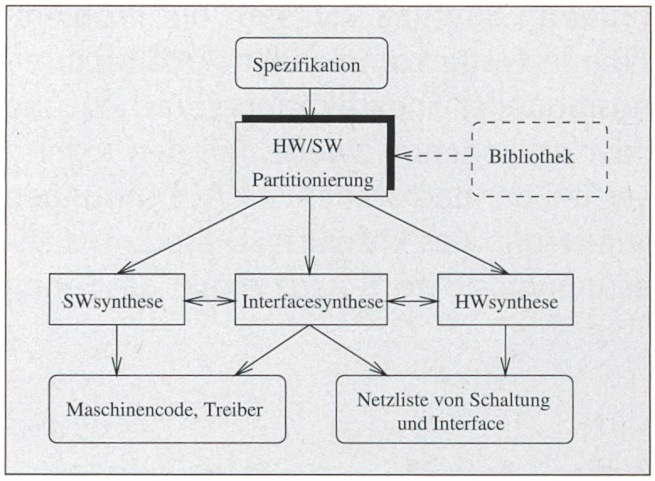

Bild 10 Typischer Entwurfsablauf von HW/SW-Systemen

die Verhaltensbeschreibung der Software bedeutet dies beispielsweise die Generierung eines Hochsprachenprogramms (z. B. in C, Pascal) und die Compilierung dieses Codes in Maschinencode mit Hilfe eines Compilers für den Zielprozessor. Analog wird für die Verhaltensbeschreibung der Hardwarekomponenten eine Verhaltensbeschreibung typischerweise in einer Hardwarebeschreibungssprache generiert (z. B. VHDL, Verilog). Mit Hilfe von CAD-Werkzeugen lassen sich auch diese Hardwarebeschreibungen nun verfeinern. Ist beispielsweise ein FPGA die Zielarchitektur zur Implementierung der Hardware, so wird die VHDL-Beschreibung verfeinert bis auf eine strukturelle Netzlistenbeschreibung beziehungsweise eine Beschreibung der Plazierung und Verdrahtung von Gattern aus einer Zellbibliothek beim Entwurf eines Asic. Für diese im allgemeinen komplexen Syntheseschritte gibt es – ähnlich wie bei den Compilern zur Softwaresynthese – automatische Werkzeuge, welche die Zwischenschritte der High-Level-Synthese, Logiksynthese und gegebenenfalls der Plazierung und Verdrahtung übernehmen. Zu den Werkzeuganbietern zählen beispielsweise die CAD-Systemhäuser Cadence, Synopsys und Mentor.

Nach der Hardware/Software-Partitionierung muss die Spezifikation so verfeinert werden, dass die Komponenten miteinander fehlerfrei Daten auszutauschen in der Lage sind. Die Kommunikation eines Mikroprozessors mit einer Hardwarekomponente (z. B. Asic, FPGA) kann zum Beispiel über einen Prozessorbus (memory-mapped I/O, DMA, Interrupt), eine serielle Schnittstelle oder über einen Port erfolgen. Man kann sich vorstellen, dass zum einen in der Software Treiberroutinen zur Realisierung der Kommunikation synthetisiert und zum anderen in der Hardware eine Steuereinheit zur Adressierung der Hardwarekomponenten, zum Generieren der Prozessorsignale und des Kommunikationsprotokolls realisiert werden müssen. Den gesamten Schritt bezeichnet man als *Interfacesynthese*. Eine wichtige Anforderung betrifft die Korrektheit des Zusammenspiels zwischen Hardware und Software. Zur Validierung müssen Simulationswerkzeuge miteinander gekoppelt werden und die Beschreibung auf unterschiedlichen Verfeinerungsstufen der Synthese – Verhalten (funktional), Register-Transfer (taktzyklenecht), Gatter (inkl. Gatterverzögerungszeiten) – simuliert werden. Bei Verilog ist dies beispielsweise auf Verhaltensebene durch Einbindung von in C geschriebenen Funktionen und Prozeduren möglich. Auf Register-Transfer-Ebene ist eine taktzyklengetreue Modellierung der Prozessorsignale nötig. Die benötigte Simulationszeit steigt im allgemeinen stark mit dem Grad der Verfeinerung.

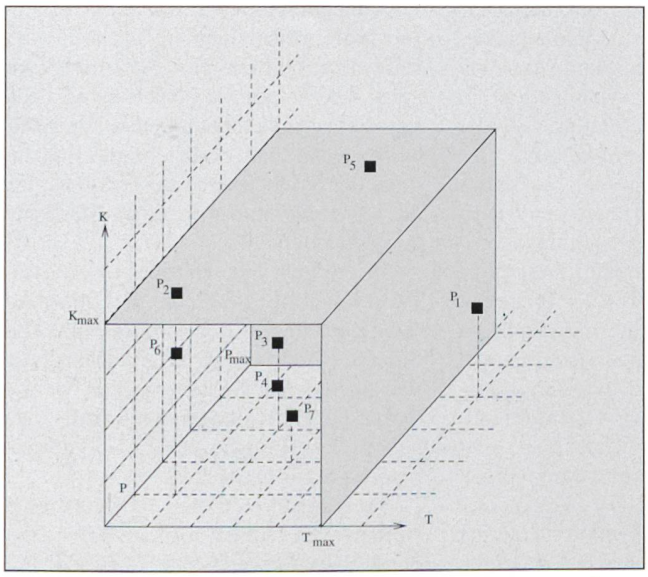

Bild 4 Verschiedene Realisierungsvarianten eines Netzwerk-Controllers (Wiederholung)

4.2 Hardware/Software-Partitionierung

Heutzutage hat man bereits ein gutes Verständnis für die getrennte Optimierung von Software (Ablaufplanung, Befehlsauswahl, Registerallokation) und Hardware (High-Level-Synthese, Logikoptimierung). Hingegen zeigt es sich, dass das mangelnde Erwägen von HW/SW-Alternativen häufig zu Entwürfen führt, die entweder zu teuer (überdimensioniert), zu langsam (unterdimensioniert) oder gemäss den Anforderungen nicht flexibel für spätere Änderungen sind (siehe z. B. Bild 4 in Teil 1). Typischerweise werden Entwurfsbeschränkungen (z. B. maximale Kosten, minimale Performanz, maximaler Leistungsverbrauch) vorgegeben, die gültige Entwurfsräume von ungültigen Entwurfsgebieten abgrenzen.

Man erkennt, dass der Raum von gemischten Hardware/Software-Realisierungen eine breite Basis von Zwischenlösungen bietet. Aus Effizienzgründen lohnt es sich, diesen Raum, der im allgemeinen mehr als zweidimensional ist, näher zu untersuchen. Die Abklärung von Alternativen ist daher Teil der Aufgabe der Hardware/Software-Partitionierung. Dabei sind zwei Wege denkbar, um Entwurfspunkte zu charakterisieren und zu bewerten:

- Synthese (synthesebasierter Ansatz)
- Abschätzung (bibliotheksbasierter Ansatz)

Der erste Weg ist oft zu zeitaufwendig; insbesondere bei komplexen Systementwürfen können nur wenige Entwurfspunkte betrachtet werden. Der wesentliche Vorteil dieses Weges besteht allerdings darin, dass das Verhalten eines Entwurfspunktes sehr genau dem Verhalten des gebauten Systems entspricht (Simulation bzw. Profiling des Codes).

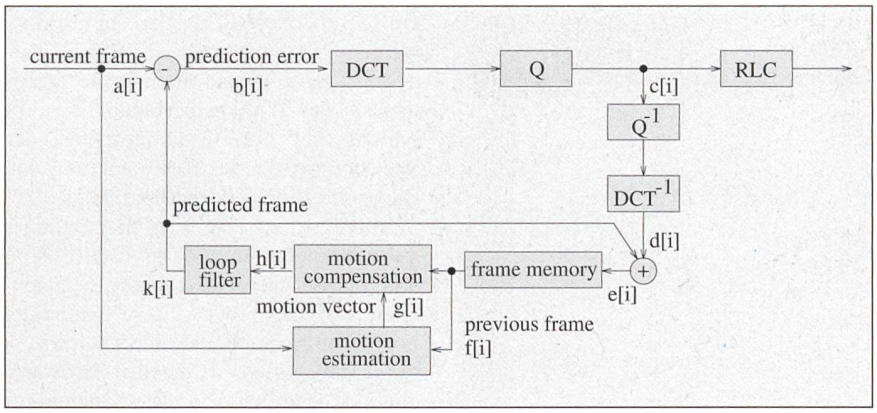

Bild 11 Darstellung eines Hybrid-Codierers für Bildsequenzen

Beim zweiten Ansatz können im allgemeinen wesentlich mehr Entwurfspunkte iteriert werden, da eine Abschätzung mit Hilfe von optimierten Modulbibliotheken im allgemeinen schneller als durch Synthese erfolgen kann. Nachteil dieser Methode ist allerdings, dass – je nach Granularität der Bibliotheksmodule – die Schätzung ungenau ist und man bezüglich der wahren Kosten, Performanz usw. nie ganz sicher sein kann, ob sich das entworfene System nach der Realisierung wie abgeschätzt verhält: Trifft man eine zu konservative Abschätzung, dann findet man eventuell nicht alle optimalen Lösungen, trifft man eine zu vage Abschätzung, so kann es sein, dass der Entwurfspunkt nicht alle Entwurfsbeschränkungen erfüllt. Der erste Pfad wird daher häufiger bei Systemen gegangen, die ein hohes Mass an Steuerung beziehungsweise ein hohes Mass an Nichtdeterminismus aufweisen, sogenannte Systeme mit Steuerungsdominanz (z. B. eingebettete Steuerungssysteme, Embedded Control). In diesen Fällen wäre eine bibliotheksbasierte Schätzung meist viel zu ungenau. Exakte Werte der Entwurfsparameter müssen über eine Simulation nach der Synthese gewonnen werden. Bei sogenannten *Systemen mit Datenflussdominanz* (z. B. signal- und bildverarbeitende Systeme) lassen sich die Entwurfsparameter hingegen recht gut aus Bibliotheken abschätzen, da solche Systeme im allgemeinen einen hohen Grad an Determinismus und statischer Parallelität aufweisen.

4.2.1 Optimierung

Betrachten wir den Entwurfsraum gültiger HW/SW-Lösungen näher, so stellen wir fest, dass nicht nur ein Punkt optimal sein kann; es gibt zum Beispiel Lösungen (z. B. Punkt P_7 in Bild 4), die kostenoptimal sind, und Lösungen, die eine optimale Performanz (z. B. Punkt P_6 in Bild 4) aufweisen. Im allgemeinen sind alle diejenigen Entwurfspunkte interessant, die dahingehend optimal sind, dass sie von keinem anderen Punkt in allen Entwurfskriterien geschlagen werden. Solche Punkte nennt man auch *Pareto-Punkte*.

Beispiel 4.1

Punkt P_4 in Bild 4.1 ist neben P_6 und P_7 ein Pareto-Punkt. P_4 und P_7 sind zwar gleich gut in den Kosten, jedoch weist P_4 eine niedrigere Datenübertragungszeit auf, während P_7 einen niedrigeren Leistungsverbrauch besitzt. Auch von anderen Punkten wird keiner der beiden Punkte in allen Eigenschaften dominiert. Folglich sind beide Punkte Pareto-Punkte.

Geometrisch lassen sich Pareto-Punkte so verdeutlichen: Seien die Eigenschaften aller Entwurfsmetriken zu minimieren (z. B. Kosten, Datenberechnungszeit, Leistungsverbrauch usw.), dann liegt ein Pareto-Punkt genau dann vor, wenn es keinen Entwurfspunkt innerhalb des von ihm mit dem Nullpunkt eingeschlossenen Quadranten gibt.

Wenn man die Möglichkeit hat, Entwurfsalternativen zu untersuchen, so besteht das Problem der *Entwurfsraumexploration* darin, dem Entwicklungsingenieur alle beziehungsweise möglichst viele Pareto-Punkte als Alternativen anzubieten.

Bild 4 stellt nur die Eigenschaften von Entwurfspunkten, also möglichen Systemlösungen dar; es sagt nichts darüber aus, wie man zu solchen Systemlösungen kommt.

4.2.2 Allokation, Bindung und Ablaufplanung

Zur Berechnung eines Entwurfspunktes muss das Werkzeug zur Hardware/Software-Partitionierung drei Aufgaben lösen, die wir als nächstes beschreiben möchten. Dabei wählen wir das Beispiel eines Video-Codecs zur Kompression von Bildsequenzen.

Beispiel 4.2

In digitalen Videoanwendungen müssen die Anforderungen an die Übertragungsbandbreiten oft durch eine geeignete Datenkompression reduziert werden. Das folgende Beispiel ist ein Hybrid-Codierer, der Transformationscodierung und prädiktive Codierung kombiniert. Der Kompressionsfaktor einer reinen Bildcodierung wird durch ein prädiktives Schema für Bildfolgen verbessert. Ein Block innerhalb eines Bildes wird aus den Daten eines entsprechenden Blocks des vorangegangenen Bildes geschätzt. Bild 11 zeigt eine Darstellung eines solchen Hybrid-Codierers auf der Verhaltensebene.

Aus der nun folgenden Beschreibung wird deutlich, dass die einzelnen Blöcke der Darstellung komplexe Teiloperationen beschreiben und dass die Kommunikation mittels komplexer Datentypen erfolgt (hier Bildsequenzen, wobei die Bilder ihrerseits wieder aus Blöcken, Makroblöcken und einzelnen Pixeln zusammengesetzt sind). Die zweidimensionale diskrete Kosinustransformation (DCT) wird auf nichtüberlappende Blöcke des Prädiktionsfehlerbildes $b[i]$ angewendet. Die transformierten Blöcke repräsentieren den räumlichen Frequenzinhalt des

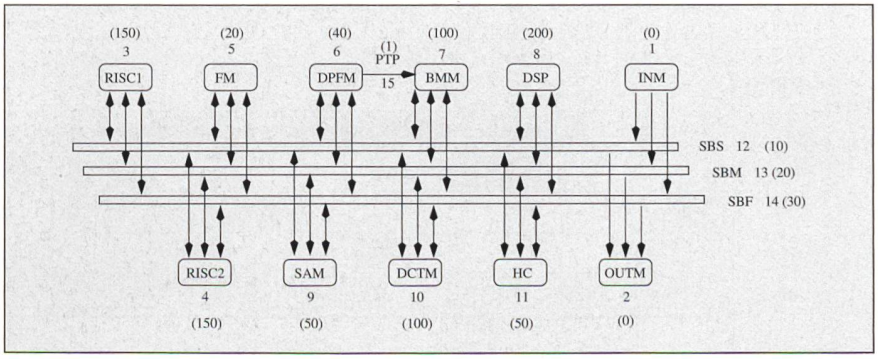

Bild 12 Beschreibung der Möglichkeiten von Zielarchitekturen und Komponenten zur Realisierung des Video-Codecs. Die Werte in Klammern bedeuten die Kosten der Ressourcen im Falle einer Verwendung.

Knoten #	Operation	Ressource/ Berechnungsdauer	Ressource/ Berechnungsdauer	Ressource/ Berechnungsdauer
1	IN	INM/0		
31	IND	INM/0		
14	OUT	OUTM/0		
36	OUTD	OUTM/0		
2	BM	BMM/22	DSP/60	RISC/88
3	RF	FM/0	DPFM/0	
37	RFD	FM/0	DPFM/0	
12	SF	FM/0	DPFM/0	
39	SFD	FM/0	DPFM/0	
4	LF	HC/2	DSP/3	RISC/9
38	LFD	HC/2	DSP/3	RISC/9
5	DIFF	SAM/1	DSP/2	RISC/2
6	DCT	DCTM/2	DSP/4	RISC/8
10	IDCT	DCTM/2	DSP/4	RISC/8
34	IDCTD	DCTM/2	DSP/4	RISC/8
7	TH	HC/2	DSP/8	RISC/8
8	Q	HC/1	DSP/2	RISC/2
9	IQ	HC/1	DSP/2	RISC/2
33	IQD	HC/1	DSP/2	RISC/2
11	REC	SAM/1	DSP/2	RISC/2
35	RECD	SAM/1	DSP/2	RISC/2
13	RLC	HC/2	DSP/8	RISC/8
32	RLD	HC/2	DSP/8	RISC/8

Tabelle I Abbildungsmöglichkeiten von Operationen auf Komponenten

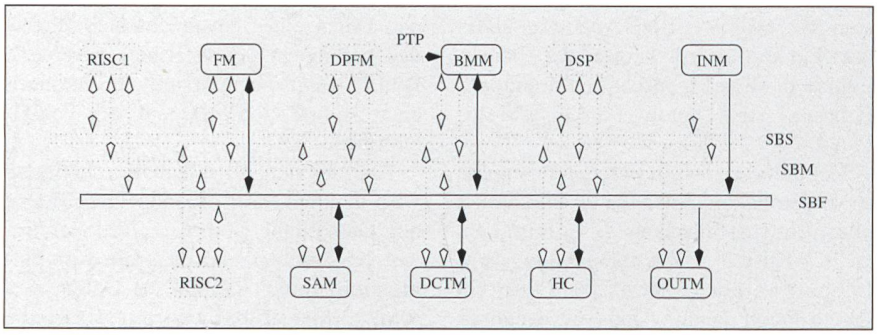

Bild 13 Architektur einer der schnellsten Implementierungen des Video-Codecs

entsprechenden Blocks. Die nachfolgende Quantisierung (Q) benutzt die räumliche Redundanz innerhalb eines Bildes und die abschliessende Codierung (RLC) die Dynamik der zu übertragenden Werte. Die Bewegungsschätzung und Bewegungskompensation werden für die Codierung zwischen aufeinanderfolgenden Bildern einer Sequenz benutzt. Ein Block im Bild $a[i]$ wird verglichen mit Nachbarblöcken des vorangegangenen Bildes $f[i]$, und hieraus wird ein Bewegungsvektor $g[i]$ bestimmt. Als Resultat der Bewegungskompensation erhält man ein geschätztes Bild $k[i]$. In Bild 11 werden Teilalgorithmen als Blöcke dargestellt. Es gibt also noch keine Spezifikation des zeitlichen Ablaufs, der Abbildung auf eine Zielarchitektur, der Speichergrössen, der Partitionierung von Bildern in Blöcke oder Makroblöcke.

Bild 12 zeigt eine Vielfalt möglicher Systemarchitekturen. Dargestellt sind drei Busse, die unterschiedliche Datenraten besitzen, zwei Speichermodule, zwei Risc-Prozessoren, ein Signalprozessor (DSP), einige dedizierte Hardwaremodule, nämlich ein Blockmatchingmodul (BMM), ein Modul zur Berechnung von DCT/IDCT-Transformationen (DCTM), ein Subtrahier/Addier-Modul (SAM), ein Huffman-Coder (HC) und I/O-Module (INM und OUTM). Die Risc-Prozessoren (RISC1 und RISC2) sowie der DSP können jeden der Blöcke der Spezifikation implementieren, allerdings ist der DSP schneller für spezielle Aufgaben, jedoch teuer als die Risc. Die anderen Hardwaremodule sind nur in der Lage, spezielle Aufgaben zu implementieren. Beispielsweise kann das DCTM-Modul nur die Operationen DCT und IDCT implementieren. Die Möglichkeiten und Eigenschaften der Abbildung von Operationen auf Module sind in Tabelle I dargestellt. Dabei wurden die in Bild 11 dargestellten Blöcke in elementare Operationen verfeinert und in die Kanten Kommunikationsknoten eingefügt, die hier nicht dargestellt sind. Diese Kommunikationsknoten können intern auf den Modulen oder auf den Bussen realisiert werden.

Zunächst soll das Partitionierungs-Tool eine Zielarchitektur aus der Vielfalt von Komponenten auswählen. Diese Aufgabe bezeichnet man als *Allokation* der Ressourcen. Beispielsweise soll nur einer der drei in Bild 12 dargestellten Busse ausgewählt werden, da vorausgesetzt wird, dass jedes Modul (mit Ausnahme des Blockmatchingmoduls BMM) nur einen Busanschluss hat. Die Kosten eines Entwurfspunktes werden dann häufig einfach aus der Summe der

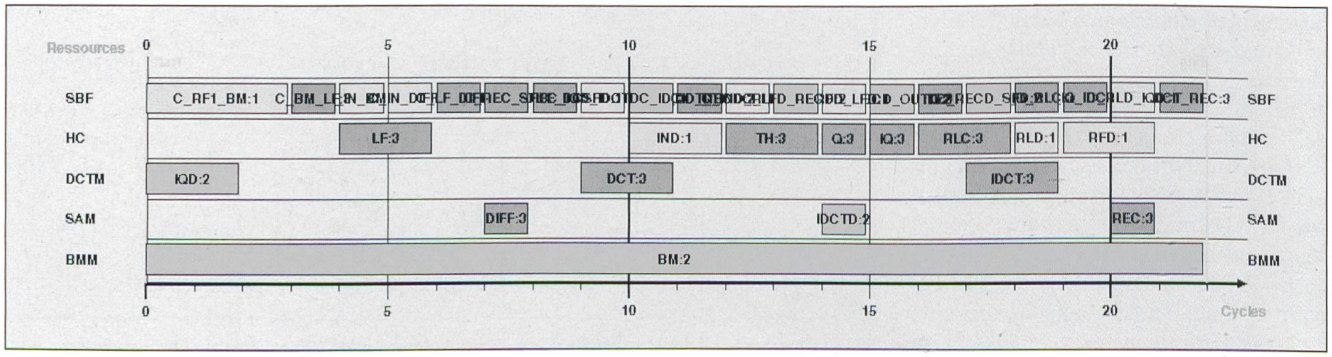

Bild 14 Gantt-Chart einer der schnellsten gefundenen Implementierungen

	Pa_1	Pa_2	Pa_3	Pa_4	Pa_5	Pa_6
Bildperiode P	22	42	54	78	114	166
Kosten c	350	340	330	280	230	180

Tabelle II Pareto-Punkte in einem einzigen Lauf des evolutionären Algorithmus

Kosten der allozierten Komponenten gebildet.

Beispiel 4.3

Bild 13 zeigt die Allokation für einen speziellen Entwurfspunkt. Alle nicht gestrichelten Komponenten sind alloziert, das heisst, sie gehören zur Implementierung, alle gestrichelten Komponenten gehören nicht zur Implementierung.

Im weiteren muss festgelegt werden, welche Funktionalität auf welche Komponente abgebildet wird (Bindung) und wann die Funktionalität (Reihenfolge oder absolute Zeit) auf den Komponenten berechnet wird (Ablaufplanung).

Dabei muss bekannt sein, auf welche Komponenten eine Funktion abgebildet werden kann und welche Eigenschaften eine solche Bindung mit sich bringt (Kosten, Berechnungsdauer usw., Tabelle I). Üblicherweise gibt man diese Abbildungsmöglichkeiten in Bibliotheken an. Zum Beispiel kann ein Modul zur Realisierung eines Filters in verschiedenen Softwarerealisierungen für verschiedene Zielprozessoren sowie in Form verschiedener Zellbibliotheken für Hardwarekomponenten (z. B. FPGA, Makroblock für Asic usw.) vorliegen, jeweils parametrisiert in Kosten, Ausführungsdauer, Leistungsaufnahme usw.

Auf der Systemebene ist besonders wichtig, auch Kommunikations- (Busse, Verbindungsleitungen) und Speicherressourcen (RAM, ROM, Register usw.) zu modellieren und als Ressourcen zu betrachten. Beispielsweise kann die Entscheidung HW oder SW oft davon abhängen, wie das Kommunikationsmedium realisiert ist. Sollte nämlich ein unterdimensionierter Bus zum Flaschenhals für die Performanz werden, wäre es besser, eine reine Softwarerealisierung zu entwerfen; ein Geschwindigkeitsgewinn durch eine dedizierte Hardwarekomponente würde durch den Kommunikationsflaschenhals ja wieder zunichte gemacht.

Für eine gegebene Problemstellung sind also

- die Funktionalität an funktionale Einheiten (Prozessoren, Multiplizierer, ALU usw.),
- Kommunikationen an Busressourcen und
- Variablen an Speicherressourcen zu binden.

Die Bindung und Ablaufplanung stellt man üblicherweise in einem *Gantt-Chart* dar (Bild 14). Dargestellt sind auf der *X*-Achse die Zeit und auf der *Y*-Achse die allozierten Ressourcen (Bild 13). Die Bindung der Operationen und die Berechnungsintervalle der Operationen sind als Balken dargestellt. Für die Bestimmung eines Ablaufplans müssen Datenabhängigkeiten und Ressourcenbeschränkungen berücksichtigt werden. Aus Platzgründen können wir hier nicht näher auf die dabei zu lösenden Probleme eingehen.

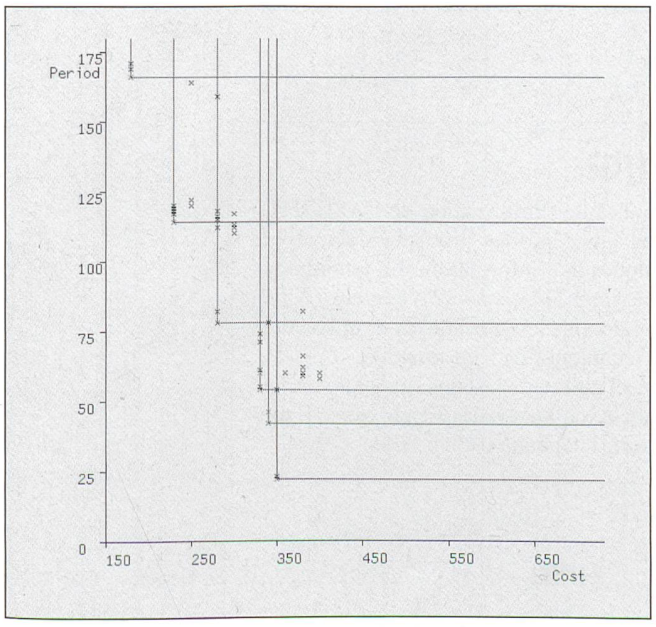

Bild 15 Entwurfsraumexploration (Video-Codec) in CodeSign mit den in Tabelle II dargestellten Pareto-Punkten

4.2.3 HW/SW-Partitionierung in CodeSign

Ziel der Entwurfsraumexploration ist, möglichst viele Implementierungen zu finden, die Pareto-Punkte sind. Im Projekt CodeSign der ETH Zürich wurde dazu ein Verfahren eingesetzt, das auf dem Prinzip der evolutionären Algorithmen beruht [20]. Eine Übersicht über die Prinzipien der Optimierung mittels evolutionärer Algorithmen findet man zum Beispiel in der Ausgabe 25/95 des Bulletins SEVVSE [21]. Das Verfahren zeigt sich als sehr gut geeignet zur Entwurfsraumexploration, da die Komplexität der zu untersuchenden Lösungen ($1.9 \cdot 10^{27}$ mögliche Bindungen im Beispiel) sowohl exakte Methoden als auch enumerative Suchmethoden in den Schatten stellt.

Beispiel 4.4

Die für das Beispiel des Video-Codecs gefundenen Pareto-Punkte sind in Tabelle II und in Bild 15 für einen zweidimensionalen Entwurfsraum (Kosten und Bildperiode) dargestellt. Die Berechnungszeit des Verfahrens zur Ermittlung aller Pareto-Punkte betrug weniger als 10 Minuten.

Bild 14 zeigt den Gantt-Chart der schnellsten gefundenen Implementierung und Bild 13 die Architektur des entsprechenden Entwurfspunktes. Offensichtlich ist die minimale Periode durch die Berechnungsdauer des Blockmatching-moduls (BMM) bestimmt.

4.3 Schätzung

Das Problem der Schätzung ist bereits auf den Ebenen der Software-Codegenerierung und Hardwaresynthese bekannt. Will man beispielsweise garantieren können, dass ein Prozessor in einem gewissen maximalen Zeitintervall (sog. Deadline) auf ein Ereignis (z. B. Interrupt) reagiert, so muss man die Ausführungszeit des auf dem Prozessor laufenden Programms im schlimmsten angenommenen Fall kennen. Man weiss, dass für ein gegebenes Programm im allgemeinen nur unter zahlreichen Einschränkungen (Verzicht auf Zeigeroperationen, beschränkte Schleifengrenzen, keine Interrupts) überhaupt entscheidbar ist, ob das Programm in endlicher Zeit beendet wird. Zusätzlich machen es die heutigen Prozessoren durch komplizierte Cache-Mechanismen und intensives Befehlspipelining sowie Ausnahmebehandlungen schwer, die Ausführungszeit eines Code-Segments zu berechnen. Zum anderen ist die Abschätzung der Ausführungszeit schwierig wegen der Existenz von datenabhängigen Befehlen, insbesondere Programmkontrollstrukturen (Schleifen, Verzweigungen).

Heutige Arbeiten zur Schätzung versuchen nun beispielsweise, möglichst gute Schranken für die Worst-Case-Ausführungszeiten eines Code-Segments zu bestimmen. Aus sicherheitskritischen Gesichtspunkten (z. B. bei Echtzeit-Systemen) ist man an der Worst-Case-Ausführungszeit interessiert, da beim Verpassen einer Deadline mit katastrophalen Auswirkungen zu rechnen ist.

Analogien lassen sich auch im Bereich der Hardwaresynthese finden. Eine gute Abschätzung der benötigten Chipfläche bedarf beispielsweise einer vorausgegangenen Plazierung und Verdrahtung. Da die Synthese bis auf diese physikalische Ebene im allgemeinen zu lange dauert, muss man auf höheren Entwurfsebenen (z. B. High-Level-Synthese) auch eine konservative Schätzung des Flächenaufwands und der Verzögerungen vornehmen, damit man garantieren kann, dass bestimmte Entwurfsbeschränkungen (z. B. Chipfläche, Taktrate) eingehalten werden.

Ohne auf Schätzungsverfahren näher einzugehen, lässt sich die Erfahrungsregel aufstellen, dass sich die Dauer zur Berechnung einer Schätzung und die Güte der Schätzung entgegengesetzt zueinander verhalten.

5. Zusammenfassung

Im vorliegenden Beitrag wurde versucht, eine Übersicht über die wichtigsten Ausprägungsformen und Problemstellungen zu geben, mit denen sich der Bereich des Hardware/Software-Codesigns beschäftigt, wobei der einführende Charakter des Beitrages zu betonen ist. In einigen Punkten (Spezifikation, HW/SW-Partitionierung) wurden Ansätze des Projekts CodeSign der ETH Zürich vorgestellt.

Als Fazit des Beitrages soll stehen, dass die Möglichkeiten der Entwurfsautomatisierung elektronischer Systeme noch bei weitem nicht erschöpft sind und dass in Zukunft noch einige neue Entwurfswerkzeuge auf den Markt kommen werden. Dies mag den Entwicklungsingenieur beunruhigen, der sich daran gewöhnt hat, mit ständig neuen Werkzeugen arbeiten zu müssen. Die neuen Werkzeuge sehen langfristig gesehen vielversprechend aus; sie werden sicherlich keine blosse Modeerscheinung sein.

Ich möchte an dieser Stelle Herrn Prof. Lothar Thiele und meinen Kollegen im Projekt CodeSign der ETH Zürich danken für die Unterstützung und nützliche Kommentare. Insbesondere gilt mein Dank Rob Esser und Tobias Blickle für die Beiträge und Abbildungen aus dem CodeSign-Projekt. Ebenfalls möchte ich Herrn Markus Pilz von der Universität Zürich für aufmerksame Kommentare danken.

Literatur

[20] *T. Blickle, J. Teich and L. Thiele:* System-level synthesis using Evolutionary Algorithms. Technical Report 16, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich, 8092 Zurich, March 1996. Presented at Workshop Dagstuhl Seminar No. 9613, Evolutionary Algorithms and their Application, Schloss Dagstuhl, Germany, March 1996.

[21] *T. Blickle:* Optimieren nach dem Vorbild der Natur – Evolutionäre Algorithmen. Bulletin SEV/VSE 86(1995)25, S. 21–26.

Hardware/Software-Codesign

Partie 2: Synthèse matériel/logiciel

Après avoir présenté dans la première partie de cet article (Bulletin ASE/UCS 25/1996) l'énoncé du problème du codesign hardware/software, les architectures et formes de réalisation ainsi que les modèles de calcul, les langages de spécification et les pratiques de projet qui entrent en ligne de compte, cette deuxième partie a pour thème le déroulement proprement dit du projet. Une grande partie de cet article traite de la partition et optimisation du projet. Ces activités revêtent une très grande importance, car elles déterminent donc tout directement les capacités et le prix des futurs produits.