Einfache Beschleunigungsschaltungen

Die meisten Beschleunigungseinrichtungen sind vergleichsweise elementar. Die Hersteller der FPGAs bieten Schnittstellen an, über die man universelle Prozessorkerne mit Akzeleratoren und Coprozessoren erweitern kann. Die Abbildungen 2.95 und 2.96 veranschaulichen zwei Einfachlösungen.

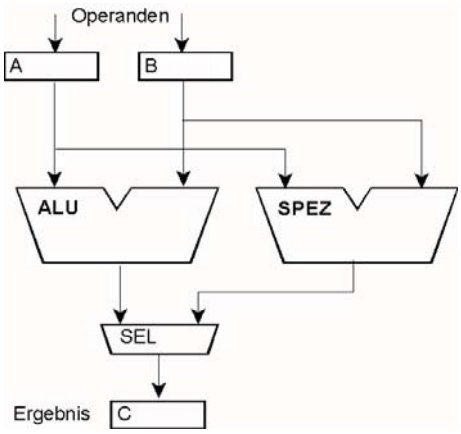

Abb. 2.95 Der Verarbeitungseinheit des Prozessors (ALU) wird ein spezielles Verarbeitungswerk beigeordnet, das die vorhandenen Operanden- und Ergebnisregister mitnutzt.

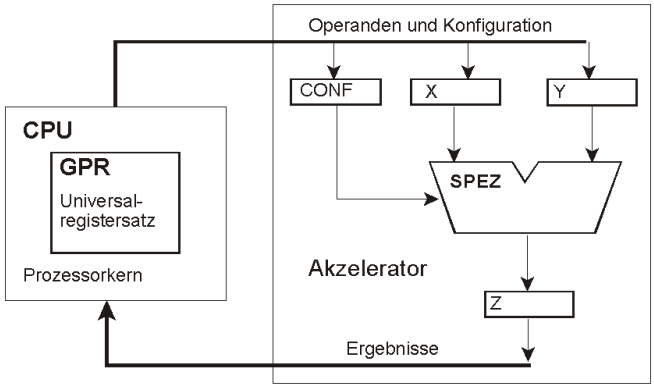

Abb. 2.96 Anschluss eines Spezialrechenwerks an einen Prozessorkern (nach [2.43]).

Die Lösung gemäß Abbildung 2.95 ist im Grunde eine Erweiterung der Verarbeitungseinheit mit zusätzlichen Verknüpfungsnetzwerken und Auswahlschaltungen* (Abbildung 3.4*** zeigt, dass universelle Verarbeitungswerke grundsätzlich aus Verknüpfungsnetzwerken für die einzelnen Operationen und Auswahlschaltungen aufgebaut werden können). Man kann nur solche Zusatzfunktionen implementieren, die mit den Operanden- und Ergebnisregistern der Verarbeitungseinheit auskommen. Manche Prozessorkerne, die für FPGAs vorgesehen sind, haben eigens Erweiterungsbefehle. Ansonsten müssen einige bisher reservierte Operationscodes decodiert werden, um die Auswahlsignale anzusteuern. Das Einfügen von Auswahlschaltungen in den Verarbeitungsweg vergrößert aber die Schaltungstiefe.

*: Praxistipp: Manchmal ist es möglich, Prozessorkerne, die als Schaltungsentwürfe (Soft Cores) vorliegen, intern umzubauen anstatt sie zu erweitern (indem man Funktionen, die nicht benötigt werden, durch eigene ersetzt).

Um diese Probleme zu vermeiden, werden gemäß Abbildung 2.96 die Zusatzfunktionen aus den Datenwegen der ALU herausgehalten. Die Operanden (und ggf. zusätzliche Operationscodes, Konfigurationseinstellungen usw.) werden mit Ladebefehlen in die Register der Zusatzeinrichtung eingetragen. Dann laufen dort die Verarbeitungsfunktionen an. Sind die Ergebnisse fertig, werden sie mit Speicherbefehlen abgeholt. Die Schnittstellen enthalten Signale zum Anzeigen von Wartezuständen, zum Auslösen von Unterbrechungen usw. Die Verarbeitungsabläufe dürfen komplizierter sein und länger dauern. Wenn die Kommunikation mit der Zusatzeinrichtung n Taktzyklen erfordert, muss die Schaltung offensichtlich mehr leisten, als sich in dieser Zeit mit üblichen Maschinenbefehlen erledigen ließe.

2.6.2 Maschinenbefehle erweitern

Die folgenden Anregungen betreffen Einfachlösungen auf Grundlage fertiger Prozessoren. Um den Overhead der Parameterübergabe, Funktionsauslösung usw. zu vermeiden, wird der Speicher des Prozessors um einen zusätzlichen Steuerspeicher erweitert (Abbildung 2.97). Der Steuerspeicher wird genau so adressiert wie der Speicher des Prozessors. Aus Sicht der Adressierung werden die Speicherworte einfach verlängert. Die Verlängerung wirkt aber nicht im Innern des Prozessors, sondern auf zusätzliche Steuerschaltungen. Der Prozessorkern bleibt unverändert.

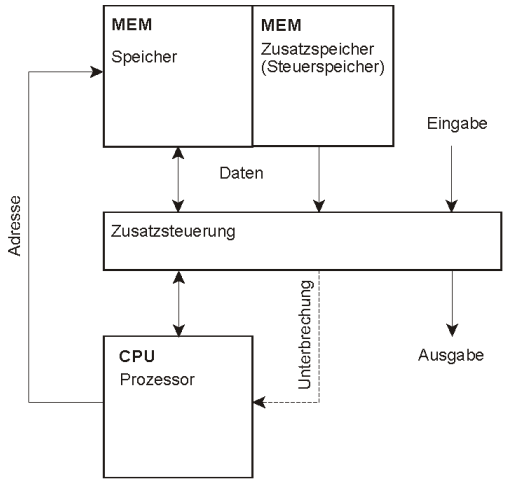

Abb. 2.97 Ein Prozessor mit zusätzlichen Befehlswirkungen durch Speichererweiterung.

Befehls- und Datenspeicher

Welche Speicher zu ergänzen sind, hängt davon ab, welche Wirkungen ausgelöst werden sollen. Die meisten der nachfolgend skizzierten Wirkungen betreffen den Befehlsspeicher.

ROM und RAM
Der Zusatzspeicher wird während des normalen Betriebs nur gelesen, auch bei Schreibzugriffen. Somit liegt es nahe, ihn als ROM auszuführen. Gelegentlich ist aber ein RAM zweckmäßiger.

Betriebsarten
Die jeweilige Betriebsart wird in einem programmseitig ladbaren Register eingestellt. Zweckmäßige Betriebsarten sind:

- eine allgemeine Erlaubnissteuerung (Sonderwirkungen ein/aus),
- prorammseitige Lesezugriffe zum Steuerspeicher (zu Diagnosezwecken),
- programmseitige Schreibzugriffe (zum Laden, wenn der Steuerspeicher ein RAM ist).

Adressbereiche
Der Zusatzspeicher kann im gesamten Adressraum oder in einzelnen Bereichen angeordnet sein. Beispiel: in einem Adressraum von 64k belegt der Steuerspeicher einen Bereich von 8k.

Anwendungsvoraussetzungen:

1. Man muss an die Speicherzugriffswege des Prozessors herankommen. Das Prinzip kann man dann implementieren, wenn der Prozessor als Soft Core vorliegt oder externe Speicherzugriffswege aufweist.
2. Der Prozessor muss die Befehle so ausführen, wie er sie aus dem Speicher holt. Prozessorkerne mit Befehlspufferung (wie 8086) und Caches scheiden aus.

Anwendungsbereiche
Es geht um die Erweiterung von Mikrocontrollern und anderen einfachen Prozessorkernen. Oftmals reichen solche Prozessoren für viele Teilaufgaben des Anwendungsproblems vollauf aus. An manchen Stellen wünscht man sich aber eine gewisse Beschleunigung oder funktionelle Ergänzung. Die Schaltungen sind einfach*, und die volle programmseitige Flexibilität bleibt erhalten. Der erweiterte Befehl wird im Grunde zu einem Mikrobefehl, der selektive Wirkungen in der Umgebung des Prozessors auslösen und auf den Prozessor selbst einwirken kann. Die Beschleunigung (Speedup) liegt im Bereich von 2:1 bis 20:1 (Richtwerte). Im Folgenden soll eine Auswahl an Zusatzfunktionen kurz vorgestellt werden.

*: Wenn man die Speicherschnittstelle des Prozessors kennt und weiß, was man außen anstellen will, kann man sie ohne Schwierigkeiten durch eine Verhaltensbeschreibung erfassen. Deshalb können wir uns hier auf einige knappe Skizzen beschränken.

Vergleichsstopp
Eine Bitposition im Zusatzspeicher bewirkt, dass im Prozessor eine Unterbrechung ausgelöst wird. Auf dieser Grundlage kann man eine Vergleichsstopp- oder Breakpoint-Funktion implementieren. Die Speichererweiterung muss als RAM ausgeführt sein. Wenn man den

gesamten Speicher entsprechend erweitert, kann man beliebig viele Breakpoints setzen, bis hin zum befehlsweisen Betrieb (Schrittbetrieb) in ausgewählten Adressbereichen. Dem gegenüber unterstützen die typischen eingebauten Breakpointvorkehrungen der Mikrocontroller nur wenige Breakpoints (beispielsweise vier).

Unterbrechungsverhinderung

Impulse und Signalspiele werden oftmals programmseitig implementiert. Eine Befehlsfolge der Art Bit setzen – Zeit abwarten – Bit löschen erzeugt einen Impuls, dessen Dauer sich aus den Ausführungszeiten der Befehle ergibt. Werden nun zwischen diesen Befehlen Unterbrechungen wirksam, kann sich die Impulsdauer unvorhersagbar verlängern; aus wenigen hundert Nanosekunden können viele Millisekunden werden. Eine Lösung besteht darin, die Unterbrechungen zu Beginn der Befehlsfolge zu verhindern (DI-Befehl) und am Ende wieder zuzulassen (EI-Befehl). Wird der Ablauf aber (als Unterprogramm) in einem Programm ausgeführt, das in sich nicht unterbrechbar sein darf, so ist der Erlaubnisbefehl (EI) falsch. Es können dann Unterbrechungen zur unrechten Zeit wirksam werden und zu Fehlern im Programmablauf führen (Absturz). Eine Bitposition im Zusatzspeicher kann genutzt werden, um die Unterbrechungsauslösung im Prozessor zeitweilig zu verhindern (Abbildung 2.98). Diese Sperre muss kombinatorisch – und mit Vorrang – wirken, darf also die eingebaute Erlaubnissteuerung (beispielsweise ein Erlaubnisbit im Bedingungsregister) nicht beeinflussen.

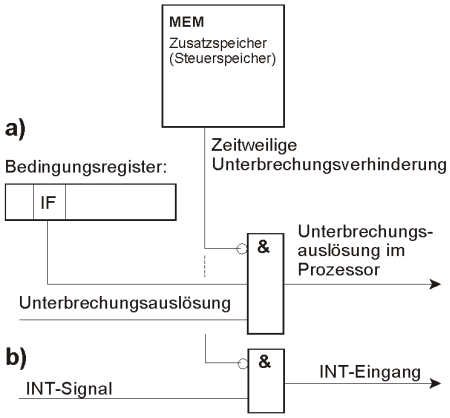

Abb. 2.98 Unterbrechungsverhinderung. a) kombinatorische Wirkung in Zusammenhang mit dem Statusbit im Bedingungsregister des Prozessors (Interrupt Flag IF). b) ist der Prozessor als fertiger Schaltkreis gegeben (Mikrocontroller auf Leiterplatte), wird dessen INT-Eingang so beschaltet.

Bedingte Befehlsausführung

Die Befehle werden durch Bedingungsauswahlsignale ergänzt. Es ergeben sich somit erweitere Befehlsformate, die dem Prinzip nach denen der ARM-Prozessoren entsprechen (vergleiche Abbildung 2.56a). Die Bedingungen werden aber der Umgebung des Prozessors oder der Außenwelt entnommen (Eingangssignale). Hierbei wird der Befehlsleseweg des Prozessors beeinflusst. Ist die Bedingung erfüllt, wird der aus dem Speicher gelesene Befehl zum Prozessor weitergeleitet. Ist sie nicht erfüllt, wird ein NOP aufgeschaltet. Soll eine Reihe

von Befehlen übergangen werden, ist es wichtig, dass die Bedingung nur am Anfang erfasst und dann konstant gehalten wird. In Abbildung 2.99 werden alle Bedingungen auf einmal abgetastet und gehalten (Prinzip des Abbildungsregisters).

Eine bedingte Verzweigung auf eine solche Bedingung ist ein unbedinger Verzweigungsbefehl, der bei Nichterfüllung übergangen wird. Ein Warten auf eine solche Bedingung ist eine unbedingte Verzweigung auf sich selbst, die bei Erfüllung übergangen und bei Nichterfüllung ausgeführt wird.

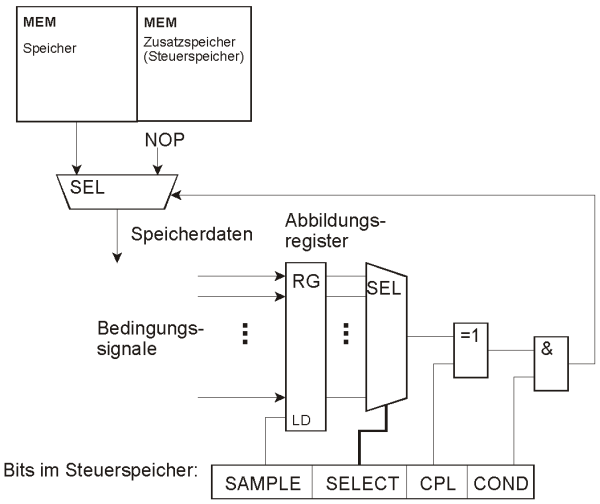

Abb. 2.99 Bedingte Befehlsausführung. SAMPLE = Bedingungen abtasten, SELECT = Bedingungsauswahl; CPL = Invertierung; COND = Funktionssteuerung.

Zusatzeingabe

In Lade- oder Eingabebefehlen wird anstelle des adressierten Inhalts ein Bitmuster gelesen, dessen Quelle über den Zusatzspeicher adressiert wird. Das Bitmuster wird beim Datenzugriff in den Lesedatenweg zum Prozessor eingespeist (Abbildung 2.100). Die vom Befehl gelieferte Zugriffsadresse kann ignoriert oder zur Auswahl der Lesedaten oder zur Datenausgabe genutzt werden.

Funktionsverzweigung

Wird die Zusatzeingabe während des Befehlslesens wirksam, kann man beliebige Befehle von außen einspeisen. Auch wäre es möglich, bei der bedingten Befehlsausführung dem Prozessor anstelle des NOP einen beliebigen anderen Befehl anzubieten. Das läuft aber auf wirklich hanebüchene Tricklösungen hinaus. Eine reguläre, oftmals brauchbare Abwandlung ist die Funktionsverzweigung. Hierbei werden beim Befehlslesen bestimmte Bitpositionen des Befehlsformats vom Datenweg des Speichers auf ausgewählte Signale umgeschaltet (Abbildung 2.101). Die typische Nutzung besteht darin, die Verzweigungsadresse von Verzweigungs- oder Unterprogrammrufbefehlen abzwandeln. Ein solcher Befehl wird so zu einer Art Mikrobefehl (vergleiche Abschnitt 4.3), der zu einem von beispielsweise acht oder 16 Nachfolgern verzweigen kann. Ebenso wäre es möglich, ein ganzes Kommandobyte

einzuspeisen, um das zugehörige Behandlungsprogramm auf schnellstem Wege zu erreichen.

EXECUTE-Befehl

Dass ein solcher Befehl manchmal nützlich wäre, wurde bereits erläutert (Seite ****). Diese Funktion kann implementiert werden, indem man die gemäß Abbildung 2.99 im Datenweg angeordnete Auswahlschaltung an programmseitig ladbare Register anschließt.

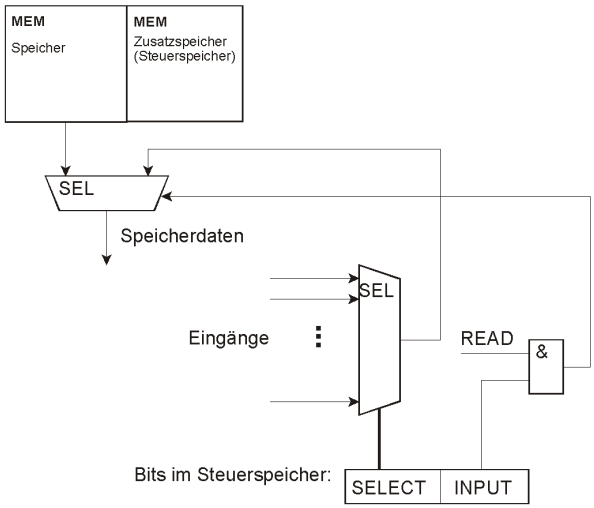

Abb. 2.100 Zusatzeingabe durch Einspeisen in den Lesedatenweg.SELECT = Auswahl der Eingangssignale (falls erforderlich); INPUT = Funktionssteuerung; READ = Steuersignal zum Datenlesen.

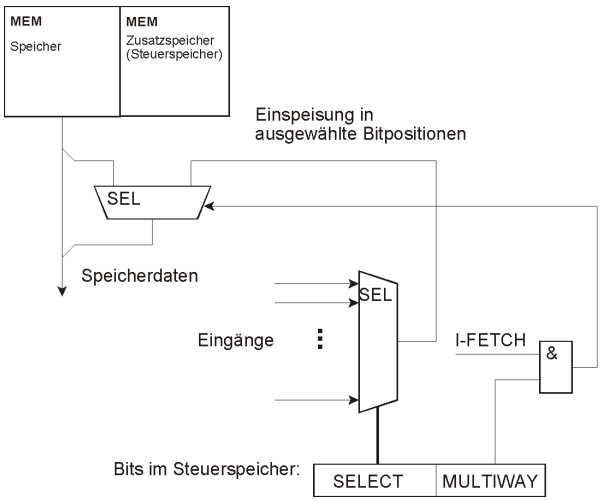

Abb. 2.101 Funktionsverzweigung durch Befehlsmodifikation. SELECT = Auswahl der Eingangssignale (falls erforderlich); MULTIWAY = Funktionssteuerung; I-FETCH = Steuersignal zum Befehlslesen.

Adressausgabe

Die während der Befehlsausführung gelieferte Datenadresse wird zur Datenausgabe genutzt. Die Adressbits werden als Datenbits interpretiert und beispielsweise einem Ausgabeport zugeführt, der vom Zusatzspeicher adressiert wird (Abbildung 2.102). Auf diese Weise kann eine 8-Bit-Maschine mit einem einzigen Befehl 16 Bits ausgeben; wenn man das Datenbyte hinzunimmt, sogar 24 Bits (Erweiterung eines Speicherbefehls STORE Adresse, Register). Es ist dafür zu sorgen, dass Zugriffe mit beliebigen Adressen keine Nebenwirkungen haben. Bei Lesezugriffen ist das von Hause aus der Fall. Ein Lesebefehl bewirkt hier nur eine Adressausgabe. Die gelesenen Daten sind gleichgültig. Bei Schreibzugriffen müssen aber – wie in Abbildung 2.102 veranschaulicht – die Schreibwirkungen im Speicher verhindert werden.

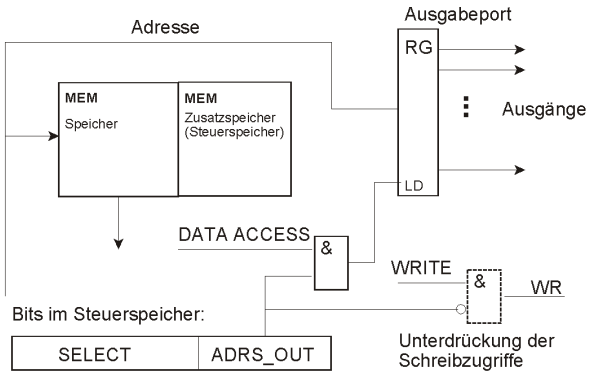

Abb. 2.102 Adressausgabe. Die Adresse als Datenwort. SELECT = Auswahl des Ausgabeports (falls erforderlich); ADRS_OUT = Funktionssteuerung; DATA ACCESS = Steuersignal für Datenzugriff (wahlweise READ oder WRITE allein oder in disjunktiver Verknüpfung); WRITE = Schreibsteuersignal des Prozessors; WR = Schreibsteuersignal am Speicherbus.

Zusatzausgabe

Parallel zur jeweiligen Befehlswirkung im Prozessor wird ein im Zusatzspeicher befindliches Bitmuster ausgegeben. Es versteht sich von selbst, dass man Bitpositionen des Zusatzspeichers verwenden kann, um beliebige weitere Funktionen auszulösen. Ein typisches Beispiel ist ein Zustandsautomat, der auf diesem Wege veranlasst wird, die Handshaking-Signalspiele eines Datenübertragungsablaufs zu steuern.

Befehlsausgabe

Die Bitmuster der Befehle werden ausgegeben. Dem Prozessor werden stattdessen NOP-Bitmuster zugeführt (Abbildung 2.103). Diese Funktion ermöglicht es, die Zugriffsbreite des Programmspeichers und den Befehlsleseablauf zu Ausgabezwecken zu nutzen (das lückenlose Befehlslesen ist oftmals der schnellste Leseablauf des Prozessors).

Nebenläufige Ausgabe

Die Ausgabe ist mit Ladebefehlen verbunden. Die gelesenen Daten werden gleichsam abgezweigt (Abbildung 2.104). Eine Folge der Art LOAD Register, Adresse – OUT E-A-Port, Register verkürzt sich damit auf den Ladebefehl.

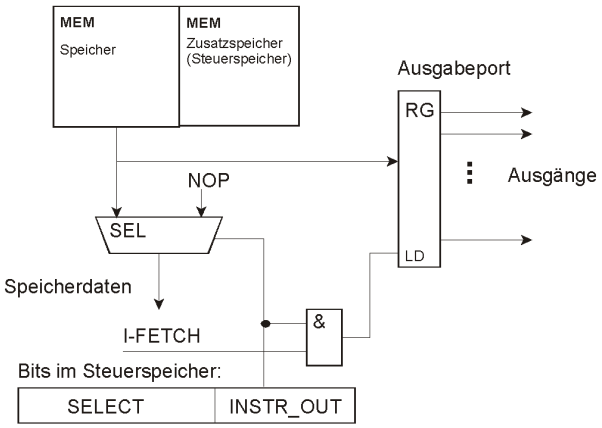

Abb. 2.103 Befehlsausgabe. SELECT = Auswahl des Ausgabeports (falls erforderlich); INSTR_OUT = Funktionssteuerung; I-FETCH = Steuersignal zum Befehlslesen.

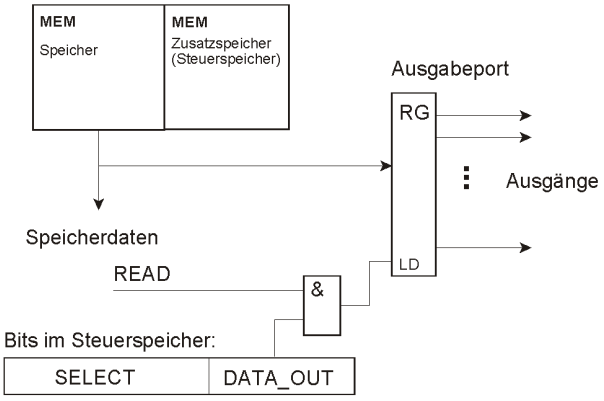

Abb. 2.104 Nebenläufige Ausgabe. SELECT = Auswahl des Ausgabeports (falls erforderlich); DATA_OUT = Funktionssteuerung; READ = Steuersignal zum Datenlesen.

Abbildung 2.105 veranschaulicht, wie das Spezialrechenwerk von Abbildung 2.96 an einen erweiterten Prozessor angeschlossen werden kann. Die Register werden mit nebenläufigen Ausgaben geladen. Der Prozessor muss nur die Operanden aus dem Speicher lesen. Dabei gelangen sie gleichsam nebenher in das angeschlossene Rechenwerk. Die Ergebnisse werden über Zusatzeingaben abgeholt. Der Rechenablauf kann durch weitere Bitpositionen im Steuerspeicher gestartet werden. Der Operationscode kann auch aus dem Steuerspeicher stammen, so dass mit dem Eintragen des letzten Operanden der Rechenvorgang automatisch beginnt. Es ist zudem nicht schwierig, den Prozessor so lange im Wartezustand zu halten, bis das Ergebnis vorliegt. Das Spezialrechenwerk verhält sich dann nicht wie ein nachträglicher Zusatz, dessen Nutzung mit einem Verwaltungs-Overhead verbunden ist, sondern so, als ob es von Grund auf zum Prozessor gehören würde.

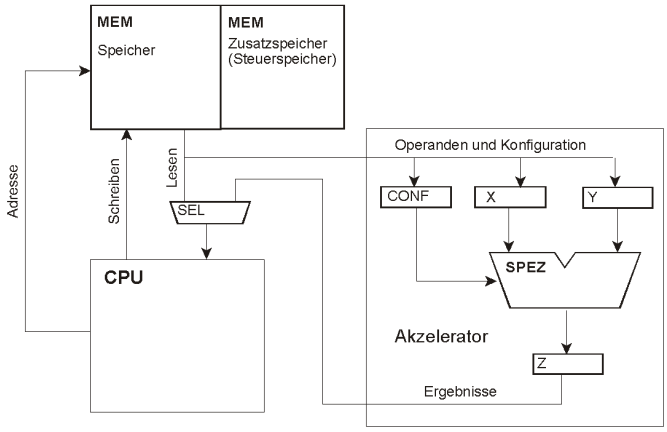

Abb. 2.105 Anschluss eines Spezialrechenwerkes an einen erweiterten Prozessor.