Signalprozessoren

Robert Dörfel
Daniel Mentz

05.06.2003

Inhaltsverzeichnis

I Allgemeiner Teil **3**

1 Einleitung **3**

2 Die von-Neumann Architektur **3**

3 Die Harvard Architektur **3**

4 Pipelining auf dem Datenpfad **4**

5 Pipelining auf Befehlsebene **5**

6 Single Instruction Multiple Data **6**

7 Very Long Instruction Word **6**

8 Super skalare Architektur **7**

9 Adressierungsarten **8**

10 Ausblick **9**

II Der Digitale Signalprozessor TMS320C50 **10**

1 Allgemeine Eigenschaften **10**

2 Architektur **10**
2.1 On-Chip Speicher 10
2.2 Central Processing Unit (CPU) 11
2.2.1 Systemkontrolle 11
2.2.2 Central Arithmetic Logic Unit (CALU) 11
2.2.3 Parallel Logic Unit (PLU) 12

3 Adressierung **12**
3.1 Short/Long Immediate Adressierung 12
3.2 Direkte Adressierung 13
3.3 Memory-Mapped Adressierung 14
3.4 Indirekte Adressierung 14
3.5 Ringpuffer 15

4 Befehlssatz **16**
4.1 Befehlssatz für die ALU 16
4.2 Befehlssatz für den Multiplizierer 16
4.3 Befehlssatz für die PLU 16
4.4 Befehlssatz für die Systemkontrolle 17

5 Das DSP Starter Kit für den TMS320C50 **17**
5.1 Software 17
5.2 Das DSK-Board 17

6 Beispielprogramme **18**

Teil I
Allgemeiner Teil

1 Einleitung

Digitale Signalprozessoren - auch DSPs genannt - sind auf die Anforderungen der Digitalen Signalverarbeitung optimierte Mikroprozessoren. Ihre besondere Leistungsfähigkeit entfalten sie bei der Abarbeitung von berechnungsintensiven und zeitkritischen Algorithmen. Im Vergleich zu universellen bzw. so genannten general-purpose Mikroprozessoren, dessen Aufgabe häufig darin besteht Daten zu verschieben bzw. zu organisieren, steht bei DSPs die möglichst schnelle Ausführung von arithmetischen Operationen im Vordergrund. Im Folgenden wird versucht die besonderen Architekturmerkmale solcher Signalprozessoren zu beschreiben, mit Hilfe derer diese Leistungsfähigkeit möglich wird.

2 Die von-Neumann Architektur

Rechnerarchitekturen nach dem von-Neumann Konzept zeichnen sich durch einen gemeinsamen Hauptspeicher für Programme und Daten aus. Abbildung 1 soll dies veranschaulichen. Die Architektur besteht vereinfacht gesprochen aus drei Einheiten. Der bereits erwähnte Hauptspeicher fasst den Datenspeicher (engl. Data Memory, DM) und den Programmspeicher (engl. Program Memory, PM) zusammen. Wichtig an dieser Stelle ist, dass pro Takt höchstens ein Zugriff auf den Hauptspeicher möglich ist. Somit kann in einem Takt entweder ein neuer Maschinenbefehl oder ein Operand geladen werden. Der Instruction Processor (IP) ist dafür zuständig, die Maschinenbefehle aus dem Speicher zu laden und sie zu dekodieren. Die letztendliche Ausführung der Befehle wird durch den Data Processor (DP) übernommen, der für diese Aufgabe u.a. eine Arithmetic-Logic-Unit und einen Multiplizierer enthält. Um die Geschwindigkeit von DSPs zu erhöhen, möchte man in der Zeitspanne von einem Takt einen Befehl ausführen und gleichzeitig den nächsten Befehl laden. Hier kommt es allerdings bei einer von-Neumann-Architektur zum Konflikt, da das Laden eines Befehls, sowie das Ausführen eines Befehls einen, möglicherweise zwei Speicherzugriffe erfordert. Somit ist auf einem von-Neumann-Rechner das Laden sowie das Ausführen von Befehlen nur nacheinander und nicht parallel möglich.

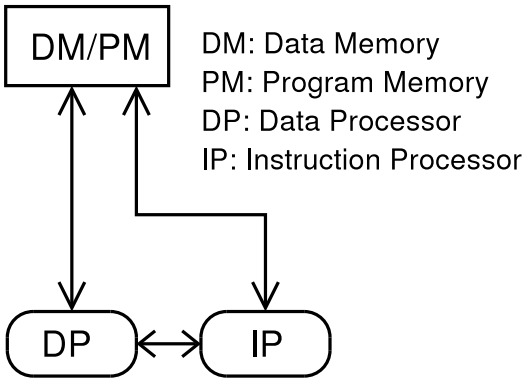


Abbildung 1: Die von-Neumann-Architektur

3 Die Harvard Architektur

Um die genannten Beschränkungen der von-Neumann-Architektur zu umgehen, bedient man sich der Harvard-Architektur. Wie in Abbildung 2 ersichtlich sieht die Harvard-Architektur zwei getrennte Speicher für Daten und Programm vor. Im Gegensatz zur klassischen Harvard-Architektur besteht bei DSPs meist noch die Möglichkeit, dass auch der Data Processor mit Hilfe des Multiplexers auf den Programmspeicher zugreifen kann, um von dort ebenfalls Operanden wie z.B. Filterkoeffizienten zu laden. Eine

um diese Fähigkeit erweiterte Harvard-Architektur nennt man auch modifizierte Harvard-Architektur. Der Name Programmspeicher ist unter dem Gesichtspunkt, dass er ja auch Daten enthält offensichtlich etwas unglücklich gewählt. Da Daten- und Programmspeicher nun unabhängig von einander sind, können Instruction Processor sowie Data Processor gleichzeitig einen Befehl und einen Operand laden. Dadurch ist es jetzt möglich einen Maschinenbefehl, der lediglich einen Operand benötigt auszuführen und gleichzeitig den nächsten Befehl aus dem Speicher zu laden. Um den Durchsatz an Instruktionen

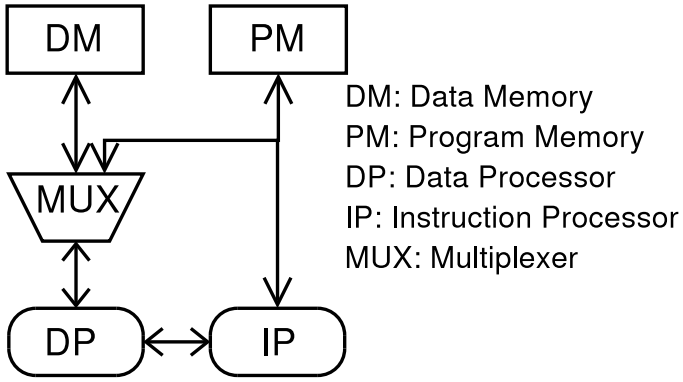


Abbildung 2: Die modifizierte Harvard-Architektur

noch weiter zu steigern, haben sich die Entwickler von DSPs noch eine weitere Methode zur Optimierung einfallen lassen. Damit der Data Processor während einem Takt gleichzeitig einen Operanden aus dem Datenspeicher und einen aus dem Programmspeicher laden kann, hat man zwischen dem Programmspeicher und dem Instruction Processor einen Cache-Speicher eingefügt, der Befehle die schon einmal geladen wurden zwischenspeichert. (Abbildung 3) Taucht im Maschinenprogramm des DSPs eine Schleife auf, so liegen nach dem ersten Schleifendurchlauf alle Maschinenbefehle der Schleife im Cache. Bei den darauf folgenden Schleifendurchläufen müssen die Befehle lediglich aus dem Cache geladen werden und nicht mehr aus dem Programmspeicher. Wichtig ist an dieser Stelle noch zu erwähnen, dass der Hauptspeicher

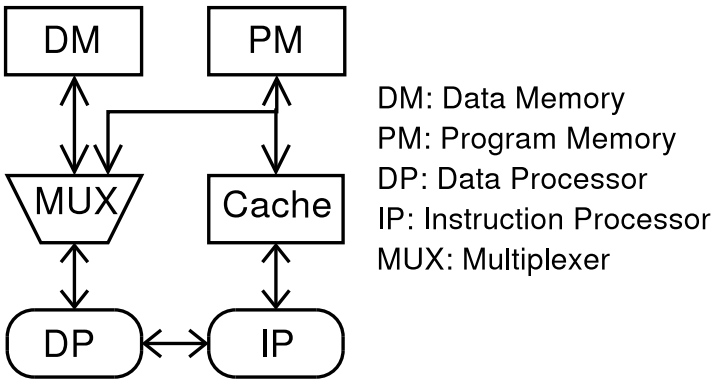


Abbildung 3: Die modifizierte Harvard-Architektur mit zusätzlichem Cache-Speicher

also Programm- und Datenspeicher meist auf dem DSP-Chip mitintegriert ist. Dies bedeutet eine höhere Geschwindigkeit, sowie niedrigere Kosten, da weniger Bussysteme nach außen geführt werden müssen und der DSP-Chip somit weniger Anschlüsse besitzt.

4 Pipelining auf dem Datenpfad

Eine weitere architektonische Besonderheit an DSPs ist das Pipelining auf dem Datenpfad. Dies soll an der für Signalverarbeitungsalgorithmen typischen MAC-Operation erläutert werden. MAC steht für Multiply-and-Accumulate und bezeichnet die Hintereinanderausführung einer Multiplikation und einer

Addition. In der Digitalen Signalverarbeitung müssen häufig Summen von Produkten (engl. product of sums) berechnet werden. Z. B. müssen bei FIR-Filtern (Finite Impulse Response Filtern) die letzten n Abtastwerte mit n verschiedenen Koeffizienten multipliziert werden und die Ergebnisse aufaddiert werden. Dies wird in der Regel durch eine Schleife realisiert, die in einem Schleifendurchlauf ein Abtastwert mit dem entsprechenden Koeffizient multipliziert und das Ergebnis auf das Akkumulator-Register aufaddiert. Zusammenfassend kann man also sagen, dass in einem Schleifendurchlauf die folgenden Operationen durchgeführt werden müssen:

- zwei Operanden laden (Abtastwert und Filterkoeffizient)
- beide Operanden miteinander multiplizieren
- Ergebnis auf den Akkumulator aufaddieren.

Die Arithmetikeinheiten der meisten DSPs haben zwei getrennte Einheiten zum Multiplizieren und zum Addieren, was bedeutet, dass eine Multiplikation sowie eine Addition parallel durchgeführt werden können. Werden die einzelnen Schritte der beschriebenen Schleife sequentiell ausgeführt liegt immer mindestens eine Recheneinheit brach. Um das zu vermeiden bedient man sich eines Tricks, der an die Fließbandverarbeitung erinnert. Sind zwei Operanden geladen und werden gerade multipliziert, so können zur selben Zeit die beiden nächsten Operatoren geladen werden. Im darauf folgenden Takt wird das Ergebnis der Multiplikation auf den Akkumulator aufaddiert. Während dessen wird das nächste Produkt berechnet und die übernächsten Operanden werden geladen. Es existiert also eine 3-stufige Pipelineverarbeitung. Abbildung 4 versucht das zu veranschaulichen.

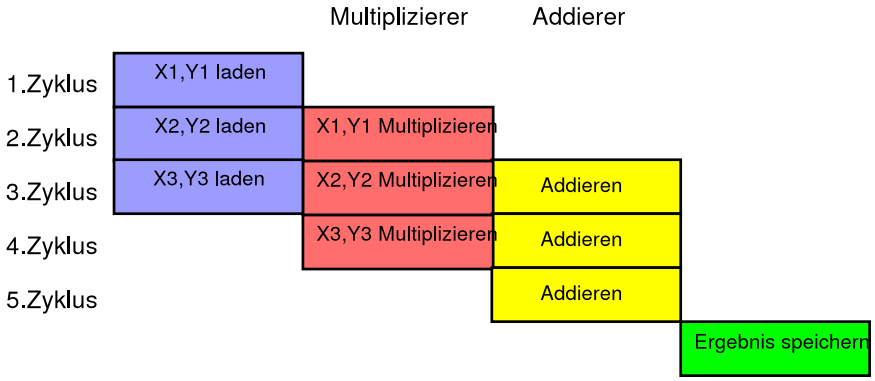


Abbildung 4: Pipelining auf dem Datenpfad

5 Pipelining auf Befehlsebene

Wie auf dem Datenpfad wird auch auf der Befehlsebene das Pipeline-Konzept verwendet. Die drei Stufen, die eine einzelne Maschinenanweisung durchläuft könnten z.B. die folgenden sein (siehe dazu auch Abbildung 5):

1. Stufe: Befehl laden

2. Stufe: Befehl dekodieren

3. Stufe: Befehl ausführen

Bei gefüllter Pipeline erreicht man auf diese Weise einen Durchsatz von bis zu einer Anweisung pro Taktzyklus. Ein Problem entsteht jedoch, sobald Verzweigungen im Programmcode auftauchen, wie das z.B. bei Schleifen der Fall ist. Normalerweise wird immer der Befehl an der nächst höheren Speicheradresse in die Befehlspipeline geladen, da darauf spekuliert wird, dass dieser auch als nächstes ausgeführt wird. Verzweigt der Prozessor aber an eine andere Adresse, so ist der Inhalt der Pipeline plötzlich obsolet. Um eine solche Situation möglichst zu vermeiden, versucht man eine Sprungvorhersage zu entwickeln, die mit

möglichst großer Wahrscheinlichkeit die richtige Instruktion in die Pipeline lädt. Ein gutes Beispiel dafür sind Schleifen, bei denen die Schleifenbedingung am Ende abgefragt wird. Man kann hier davon ausgehen, dass in den meisten Fällen an den Schleifenanfang gesprungen wird. Ein weiterer wichtiger Begriff in diesem Zusammenhang ist das „zero overhead looping“. Gemeint ist damit, dass die Schleifenbedingung schon abgefragt wird, bevor das Ende das Schleifenrumpfes erreicht wird. Somit geht keine Prozessorzeit durch Abfragen von Schleifenbedingungen oder Verzweigungen verloren. Die Ausführungszeit einer Schleife besteht somit nur aus der Ausführungszeit des Schleifenrumpfes multipliziert mit der Anzahl der Schleifendurchläufe.

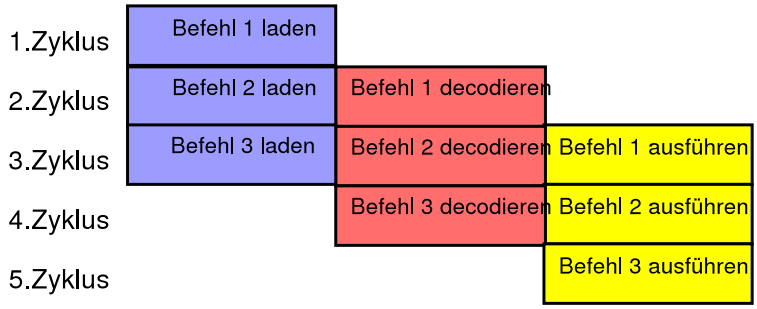


Abbildung 5: Pipelining auf Befehlsebene

6 Single Instruction Multiple Data

Viele Signalprozessoren sind in der Lage mit einer Instruktion mehrere Paare von Operanden in einem Takt zu verknüpfen. Diese Technik nennt man Single Instruction Multiple Data oder kurz SIMD. Ein gutes Beispiel ist die Addition von Vektoren. Anstatt die einzelnen Komponenten hintereinander zu addieren, kann man durch einen einzigen Befehl alle Komponenten parallel addieren. (siehe Abbildung 6) Häufig wird das so umgesetzt, dass ein Register in z.B. vier gleich große Teile zerlegt wird. Sollen nun zwei Register addiert werden, so werden die Register nicht als ganzes addiert, sondern die einzelnen Registerteile werden paarweise miteinander verknüpft.

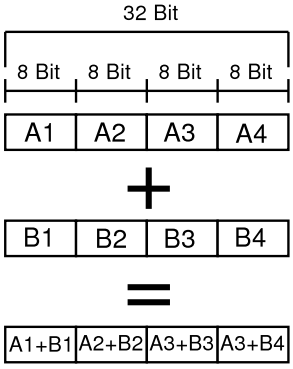


Abbildung 6: Single Instruction Multiple Data

7 Very Long Instruction Word

Um die Leistungsfähigkeit von DSPs noch weiter zu steigern, bringen die Hersteller von DSPs mehrere parallele Funktionseinheiten auf einem Chip unter. Der TI TMS320C62x verfügt z.B. über acht „functional units“ (2 Multiplizierer, 6 ALUs). Dieses Konzept verlangt natürlich auch, dass die einzelnen Einheiten parallel mit Instruktionen versorgt werden. Ein Befehlsstrom von einzelnen Anweisungen reicht hierfür

nicht mehr aus, da sonst immer nur eine Einheit pro Taktzyklus mit einem Befehl versorgt werden könnte. Alle anderen Einheiten blieben somit ungenutzt und der DSP wäre schlecht ausgelastet. Um das zu vermeiden, fasst man die einzelnen Befehle für die verschiedenen Funktionseinheiten zusammen und bildet ein sog. langes Befehlswort oder auch Very Long Instruction Word. Der Programmierer kann also entscheiden, welche Befehle parallel ausgeführt werden, um damit ein deterministisches Verhalten zu gewährleisten. Ein solches Befehlswort für den TMS320C62x könnte z.B. wie folgt aussehen:

```
     [A1]   B        .S2    LOOP         ; branch to loop
||   [A1]   SUB      .S1    A1,1,A1      ;* decrement loop counter
||          ZERO     .L1    A8           ; zero out sum0 accumulator
||          ZERO     .L2    B8           ; zero out sum0 accumulator
```

Die zwei Striche am Zeilenanfang weisen daraufhin, dass dieser Befehl parallel mit den vorhergehenden ausgeführt werden soll und somit in dasselbe Befehlswort gepackt werden muss. Die dritte Spalte legt die Funktionseinheit fest, auf die der Befehl zur Laufzeit zugewiesen werden soll. Diese Zuweisung von Befehlen auf Funktionseinheiten wird auch Dispatching genannt und soll durch Abbildung 8 verdeutlicht werden. Einige Nachteile der VLIW-Architektur dürfen hier natürlich nicht verheimlicht werden. Auf Grund von Abhängigkeiten zwischen einzelnen Operationen ist es häufig nicht möglich die Berechnungsschritte eines Algorithmus zu parallelisieren. Deshalb muss das Befehlswort mit sog. NOP-Befehlen (No Operation) aufgefüllt werden, was dazu führt, dass Speicher verschwendet wird und die Auslastung der Funktionseinheiten sinkt. Weitere Nachteile sind u.a. die schwierige Programmierung und die Tatsache, dass solche DSPs meist nicht aufwärtskompatibel sind.

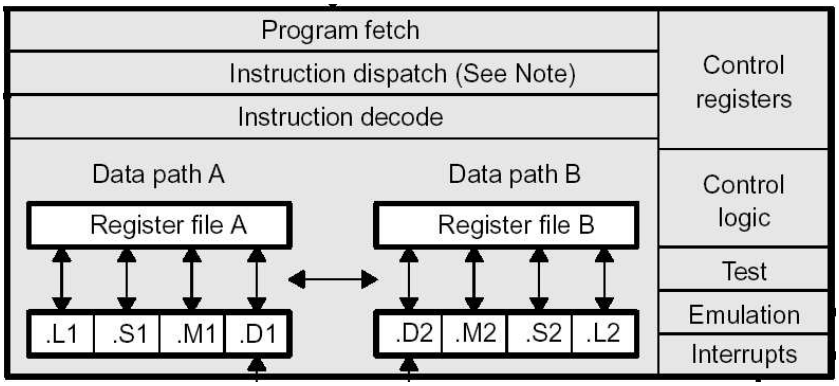


Abbildung 7: Der TI TMS320C62x von Texas Instruments mit seinen acht functional units

8 Super skalare Architektur

Einen etwas anderen Weg, wie bei VLIW-Architekturen geht man bei super skalaren DSPs. Die Parallelisierung wird bei diesem Konzept nicht vom Programmierer durchgeführt, sondern von der Hardware während der Laufzeit. Der Prozessor liest also wieder einen sequentiellen Befehlsstrom in seine Befehlspipeline ein und versucht dann die einzelnen Befehle in der Pipeline für eine parallele Ausführung zu gruppieren (Instruction Grouping). Diesen Vorgang nennt man auch dynamisches Instruktionsscheduling. Wichtig bei der Entwicklung eines solchen DSPs ist, dass den Abhängigkeiten zwischen den einzelnen Instruktionen Rechnung getragen wird. Ist z. B. die zweite Anweisung von zwei aufeinander folgenden Anweisungen von dem Ergebnis der Ersten abhängig, so können diese nicht parallel ausgeführt werden. Ein Nachteil von super skalaren DSPs ist, dass die Vorhersagbarkeit von Ausführungszeiten erschwert wird und sie somit ihre eigene DSP-Tauglichkeit gefährden. Was bei VLIW-Architekturen ein Nachteil ist und hier zum Vorteil wird, ist die einfachere Programmierung von super skalaren DSPs. Als Beispiel für super skalare Architekturen soll die ZSP400 Architektur von LSI Logic dienen. Abbildung 9 zeigt die fünfstufige Befehlspipeline.

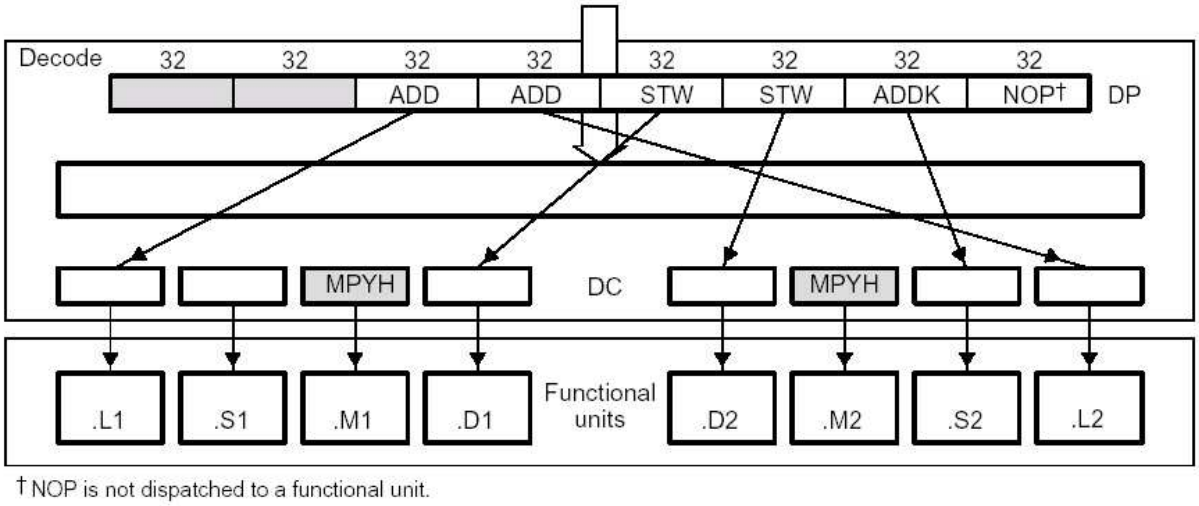


Abbildung 8: Die einzelnen Instruktion werden beim Dispatching auf die Funktionseinheiten verteilt.

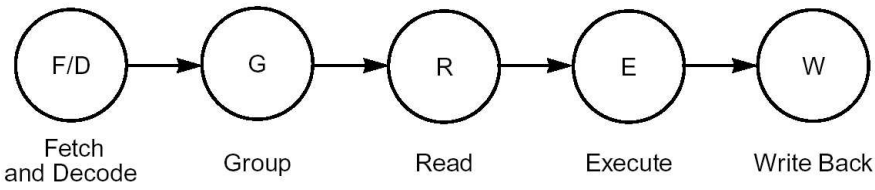


Abbildung 9: Befehlspipeline des LSI403LP

9 Adressierungsarten

DSPs verfügen häufig über separate Adressgeneratoren, die die notwendigen arithmetischen Operationen zur Berechnung von Speicheradressen durchführen. Durch diese zusätzlichen Einheiten wird der Arithmetikeinheit einiges an Arbeit abgenommen, was zur Folge hat, dass die Ausführung von DSP-Programmen noch weiter beschleunigt wird. Bei einem FIR-Filter z.B. müssen die Abtastwerte mit einer Reihe von Filterkoeffizienten verrechnet werden, die alle hintereinander im Speicher abgelegt sind. In jedem Schleifendurchlauf der Berechnung muss also die Speicheradresse des nächsten Koeffizienten berechnet werden, indem auf die alte Adresse eine Konstante aufaddiert wird. Diese Funktion nennt man pre- bzw. post-modify, da vor bzw. nach einem Speicherzugriff, der Wert des Registers, das die Speicheradresse enthält erhöht wird. Eine andere für die Signalverarbeitung wichtige Adressierungsart ist die sog.

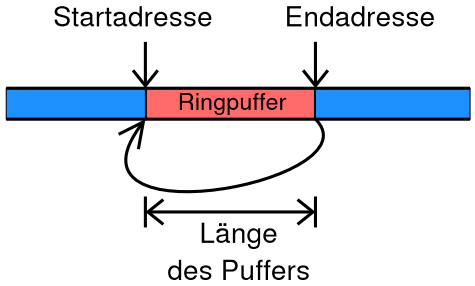


Abbildung 10: Der Ringpuffer

Modulo-Adressierung. Sie ermöglicht die Verwendung von Ringpuffern, die notwendig sind um z.B. ein Signal zu verzögern. Ein Ringerpuffer besteht aus den Speicherzellen zwischen einer festgelegten Start- und Endadresse. Sobald die Speicheradresse bei fortlaufenden Leseoperationen die Endadresse erreicht hat, muss diese wieder auf die Anfangsadresse gesetzt werden. Genauso verhält es sich bei Schreiboperationen. (siehe dazu auch Abbildung 10). Diese Überprüfungen und das Zurücksetzen auf die Anfangsadresse wird von den Adressgeneratoren übernommen, die somit die Arithmetikeinheit entlasten.

10 Ausblick

Wie auch bei general-purpose Mikroprozessoren schreitet der technologische Fortschritt bei DSPs schnell voran. Einige Ziele aktueller Entwicklungen sind eine geringere Integrationsdichte sowie ein geringer Leistungsverbrauch, damit der Einsatz in mobilen Anwendung ermöglicht wird. Für sehr rechenintensive Anwendung wie z.B. Funkortungssysteme o.ä. werden vermehrt Multiprozessorsystem eingesetzt, was bedeutet, dass bis zu 16 oder mehr Prozessoren parallel an einer Aufgabe arbeiten.

Teil II
Der Digitale Signalprozessor TMS320C50

1 Allgemeine Eigenschaften

Der 1992 erschienene DSP TMS320C50 von Texas Instruments soll als praktisches Beispiel für die Signalprozessoren dienen. Er besitzt folgende allgemeine Eigenschaften:

- *Fixpunkt-DSP*
  Der TMS320C50 ist ein Fixpunkt-DSP, der als Zahldarstellung die 2-er Komplementdarstellung mit einer Breite von 16 Bit benutzt.

- *Harvard-Architektur*
  Er ist nach der Harvard-Architektur gebaut, was bedeutet, dass es zwei Busse gibt - einen für das Programm und den anderen für die Daten. Allerdings werden nach außen hin nur 16 Pins für die Adresse und 16 Pins für die Daten gelegt, was Kosten bei der Produktion erspart. Diese Zusammenlegung der beiden Busse nach außen hin, nennt Texas Instruments die „modifizierte Harvard Architektur".

- *4-Level-Pipelining auf dem Datenpfad*
  Dies bedeutet, dass er in einem Takt den Befehl aus dem Programm-Speicher holt und im nächsten Takt dekodiert, und zusätzlich schon den nächsten Befehl einliest. Im darauffolgenden Takt holt er den Operanden um im folgenden Takt die Operation auszuführen, während er schon den nächsten Befehl dekodiert und den übernächsten Befehl aus dem Speicher holt. Das gesamte Schema ist in Abbildung 11 dargestellt.

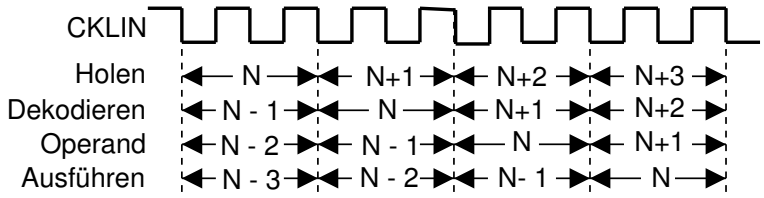


Abbildung 11: Pipelining

2 Architektur

Der DSP TMS320C50 besitzt auf dem Chip folgende zwei Hauptbestandteile: den On-Chip Speicher und die Central Processing Unit (CPU), die in den nächsten Abschnitten weiter erklärt werden.

2.1 On-Chip Speicher

Der TMS320C50 besitzt im Chip selbst Speicher, auf den er in einem Takt zugreifen kann, und somit einen schnellen Zugriff auf die Daten und Programme hat. Der Speicher teilt sich in folgende drei Teile auf:

- 1056 × 16 Bit dual-access Daten-RAM
  Auf diesen Speicherteil kann der DSP in einem Takt gleichzeitig lesend und schreibend zugreifen. Deshalb ist er besonders für Daten geeignet.

- 9K × 16 Bit single-access Daten- oder Programm-RAM
  Dieser Speicherteil dient für die Programme und bietet zusätzlich noch Platz für größere Daten.

- 2K × 16 Bit Programm-ROM
  Das ROM ist für Programme gedacht. Da der Speicher nicht flüchtig ist, kann der DSP sich mit diesem Speicherteil initialisieren. Allerdings müssen die fertigen Programme Texas Instruments geschickt werden, die sie anschließend auf das ROM brennen.

2.2 Central Processing Unit (CPU)

Die CPU läßt sich wiederum in drei Teile unterteilen: die Systemkontrolle, die Central Arithmetic Logic Unit (CALU) und die Parallel Logic Unit (PLU).

2.2.1 Systemkontrolle

Die Systemkontrolle sorgt dafür, dass der Befehl aus dem Speicher geholt wird, um anschließend dekodiert und ausgeführt zu werden. Ebenfalls kümmert sie sich um Programmverzweigungen und Unterprogrammaufrufe.

2.2.2 Central Arithmetic Logic Unit (CALU)

Die CALU ist die Recheneinheit, in der die wichtigsten arithmetischen Funktionen ausgeführt werden. Die superskalare Architektur ist hier besonders auffällig. Denn die CALU unterteilt sich in die drei unterschiedlichen Einheiten: die Schieber, der Multiplizierer und die Arithmetic/Logic Unit (ALU).

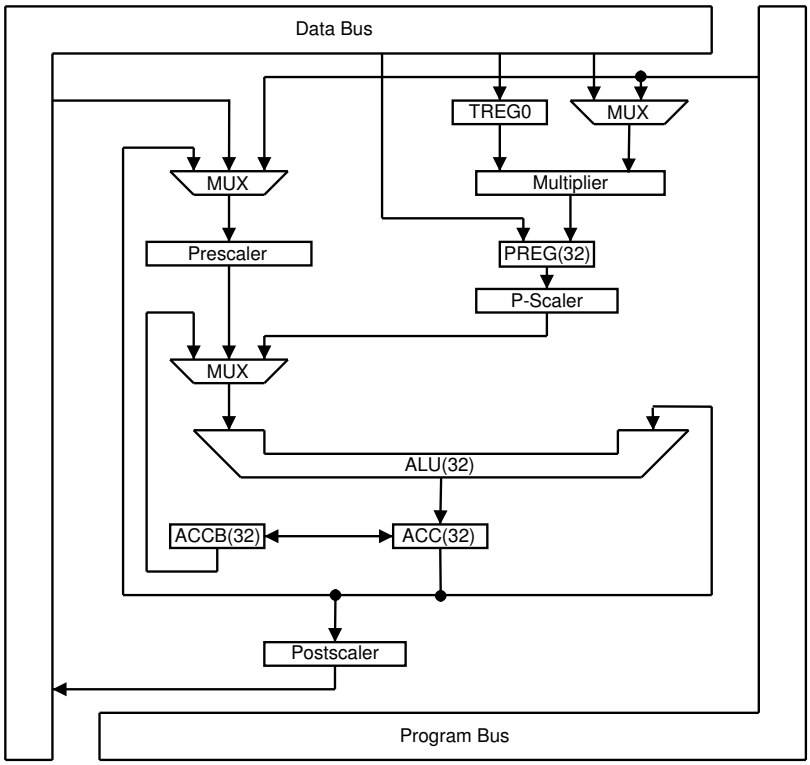


Abbildung 12: Central Arithmetic Logic Unit (CALU)

Schiebe-Einheiten Die Schiebe-Einheiten sind dafür zuständig, dass die Operanden, bevor sie von der ALU verarbeitet werden, nach links oder rechts (ohne die ALU zu benutzen) geschoben werden können. Wie aus der Abbildung 12 ersichtlich befinden sich eine der Schiebe-Einheiten vor der ALU. Diese sorgt dafür, dass der erste Operand, der aus dem Speicher kommt oder der Akkumulator selbst ist, um gewisse Bits nach links oder rechts geschoben werden. Ebenfalls kann das Produkt-Register, bevor es

auf den Akkumulator aufaddiert wird, geschoben werden. Wenn der Akkumulator zurück in den Speicher geschrieben wird, besteht ebenfalls die Möglichkeit, das Ergebnis nach links oder rechts zu verschieben.

Multiplizierer Der Multiplizierer führt eine 16 Bit $\times$ 16 Bit Multiplikation in einem Takt aus und speichert das Ergebnis in dem 32 Bit breiten Product Register (PREG). Der erste Faktor ist immer das 16 Bit breite Temporary Register 0 (TREG0). Der zweite Operand kommt entweder vom Daten Bus oder vom Programm Bus.

Arithmetic/Logic Unit (ALU) Die 32 Bit ALU ist für die verschiedensten Operationen, von denen sie die meisten in einem Takt ausführt, zuständig. Sie addiert, subtrahiert, führt logische Operationen wie die UND-/ODER-Operation, Schiebe- und Rotier-Operationen aus. Der erste Operand ist immer der 32 Bit breite Akkumulator. Der zweite Operand ist entweder der Akkumulator-Puffer, das Produkt Register oder ein Operand vom Daten oder vom Programm Bus.

2.2.3 Parallel Logic Unit (PLU)

Die Parallel Logic Unit ist eine zur ALU parallele Logik-Einheit, die Und-, Oder- und Exklusiv-Oder-Operationen neben der ALU ausführen kann. Der Vorteil dabei ist, dass der Akkumulator den aktuellen Wert beibehalten kann, während die Logik-Operation in der PLU ausgeführt wird. Der erste Operand ist ein Wert aus einer beliebigen Speicherzelle, die entweder direkt oder indirekt[1] adressiert wird. Der zweite Operand ist entweder das DBMR-Register oder ein Immediate Operand[1].

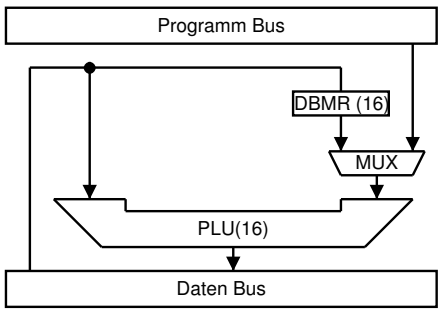


Abbildung 13: Parallel Logic Unit

3 Adressierung

Der Adressraum geht von der Adresse 0h bis FFFFh (16 Bit breite Adressen). Allerdings hat der TMS320C50 keine 64K Wörter Speicher. Deshalb ergibt sich folgende Belegung des Speichers, die in Abbildung 14 dargestellt ist. Dabei ist zu beachten, dass RAM, CNF und OVL Statusbits sind, die während der Laufzeit in den Statusregistern PMST und ST1 gesetzt werden können und sich dann entsprechend der Adressraum ändert.

3.1 Short/Long Immediate Adressierung

Bei der Immediate Adressierung steht der Operand direkt hinter dem Befehl im Programmspeicher. In dem Assemblercode zeigt das Symbol # vor dem Operand an, dass es sich um einen Immediate-Operanden handelt. Dabei gibt es zwei Arten von der Immediate Adressierung:

[1] Die Adressierungsarten sind in Kaptitel 3 erklärt

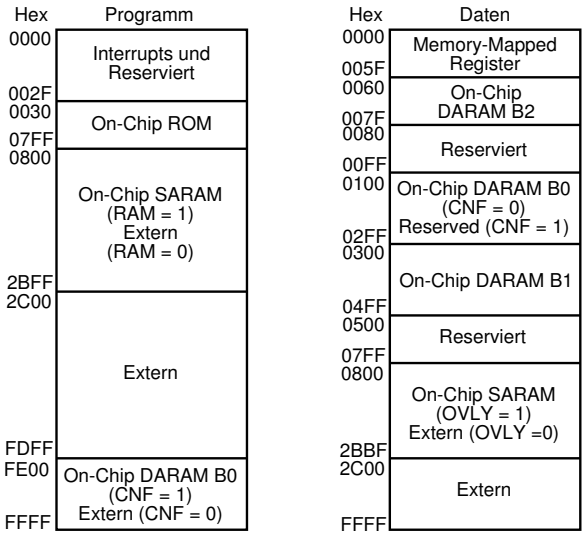
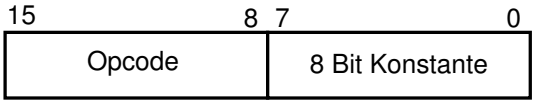
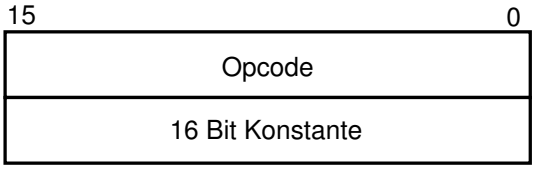


Abbildung 14: Adressraum

- *Short Immediate Adressierung*
  Hier handelt es sich um ein 8 Bit Wert, der in den unteren 8 Bit des Befehlswortes steht.



- *Long Immediate Adressierung*
  Dabei ist der Immediate-Operand ein 16 Bit Wert der hinter dem Befehlswort in der nächsten Speicherzelle im Programmspeicher steht.



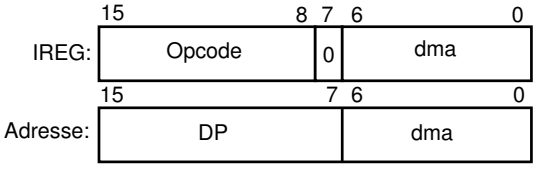
Welche der beiden Adressierungen verwendet wird, entscheidet der Assembler während der Assemblierung: Ist der Immediate-Operand ein Wert der sich durch eine 8 Bit Zahl darstellen läßt, dann wird die Short Immediate Adressierung verwendet, ansonsten die Long Immediate Adressierung.
Beispiele:

- `ADD #32h`
  Hier wird der Wert 32h auf den Akkumulator addiert, wobei die Short Immediate Adressierung verwendet wird, weil der Wert 32h noch mit 8 Bit dargestellt werden kann.

- `ADD #FFFh`
  Hier wird der Wert FFFh auf den Akkumulator addiert. Allerdings wird die Long Immediate Adressierung verwendet, da der Wert FFFh nicht mit 8 Bit dargestellt werden kann.

3.2 Direkte Adressierung

Der gesamte Adressraum ist in 512 Seiten, die jeweils 128 Wörter haben, eingeteilt. Mit dem 9 Bit breiten Data Memory Page Pointer (DP) werden die Seiten angesteuert. Die Offset-Adresse steht in den 7 niedrigsten Bits des Befehlwortes. Diese Bits werden auch Data Memory Address (dma) genannt. Somit ergibt sich für die direkte Adressierung folgendes Bild:



Beispiele:
`ADD 56h; DP = 30h`
Die Adresse lässt sich wie folgt errechnen: $30h \cdot 80h + 56h = 0f56h$. Somit wird der Wert mit Adresse 0f56h auf den Akkumulator aufaddiert.

3.3 Memory-Mapped Adressierung

Wie in der Abbildung 14 über den Adressraum zu erkennen ist, werden im Datenraum in den ersten Wörtern die Register des DSP eingeblendet. Auf diesem Weg können den Registern Werte zugewiesen werden, indem der Data Page Pointer auf 0 gesetzt wird, und mit der entsprechenden Offset-Adresse das Register angesprochen wird. Für diesen Zugriff auf die Register bietet der DSP zusätzlich die Befehle `LAMM` und `SAMM` an, mit denen die Register gelesen beziehungsweise den Registern Werte zugewiesen werden können, ohne dass man den DP auf null setzen muss. Denn dies wird automatisch von den beiden Befehlen erledigt. Der Vorteil ist wiederum, dass man sich einen Befehl für das Setzen des DP spart und den DP somit auf seinem alten Wert beläßt.

3.4 Indirekte Adressierung

Die indirekte Adressierung geschieht über die so genannten acht Auxiliary Register (AR0 - AR7), die jeweils 16 Bit breite haben, und eine Adresse enthalten. Zusätzlich gibt es noch den 3 Bit breiten Auxiliary Register Pointer (ARP), der angibt, welche der acht AR's gerade verwendet wird. Für die Manipulation dieser Register gibt es zusätzlich die Auxiliary Register Arithmetic Unit (ARAU), die der ALU wiederum Arbeit abnimmt, indem sie die Manipulation der Auxiliary Register übernimmt. Die ARAU kann folgende Operationen ausführen:

- Den ARP auf einen neuen Wert setzen.
- Das aktuelle AR inkrementieren oder dekrementieren.
- Das aktuelle AR um das Index Register (INDX) erhöhen oder erniedrigen.
- Das aktuelle AR um das Index Register (INDX) mit einer umgekehrten Übertragsrichtung erhöhen oder erniedrigen.

Im Assemblercode bedeutet * als Operand, dass der Operand indirekt adressiert wird. Somit wird mit dem Wert an der Adresse, auf die das aktuelle AR zeigt, gerechnet. Ein Zusätzliches + oder - bedeutet, das das aktuelle AR nach der Operation inkrementiert oder dekrementiert wird. Falls hinter dem * ein 0+ oder 0- steht, wird das Indexregister nach der Operation aufaddiert oder abgezogen. Ein BR vor der 0 bedeutet, dass das Indexregister mit einer umgekehrten bertragsrichtung aufaddiert beziehungsweise subtrahiert wird. Nach dieser Kombination der indirekten Adressierung kann noch abgegrenzt durch ein Komma der neue Wert des ARP übergeben werden.

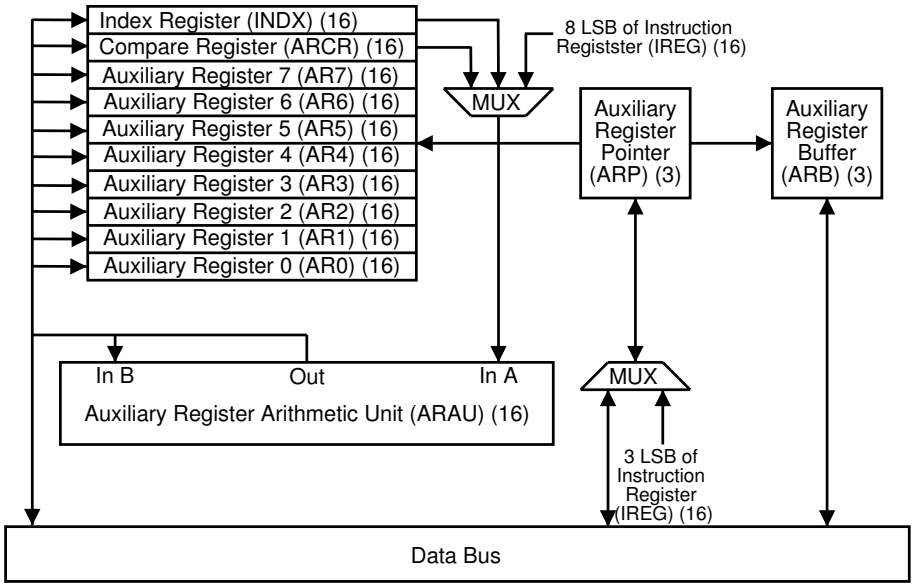


Abbildung 15: Die Auxiliary Register

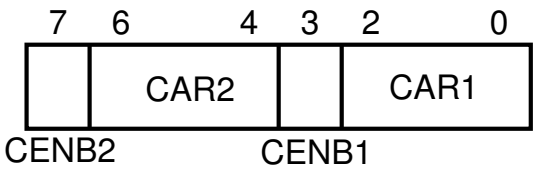
Beispiele:

- `ADD *`
  Addiert den Speicherinhalt, auf den das aktuelle Auxiliary Register zeigt, auf den Akkumulator.
- `ADD *+, AR5`
  Addiert den Speicherinhalt, auf den das aktuelle Auxiliary Regsiter zeigt, auf den Akkumulator und erhöht den Wert vom aktuellen AR um eins und setzt den ARP auf 5.
- `ADD *0-`
  Addiert den Speicherinhalt, auf den das aktuelle Auxiliary Register zeigt, auf den Akkumulator und erniedrigt das aktuelle AR um den Wert des Index-Register.
- `ADD *BR0+`
  Addiert den entsprechenden Speicherinhalt auf den Akkumulator und erhöht das aktuelle AR um den Wert des Indexregister mit umgekehrter Übertragsrichtung.

3.5 Ringpuffer

Der TMS320C50 besitzt zwei Ringpuffer, die mit Hilfe der indirekten Adressierung und folgenden Registern gelöst sind:

- *CBSR1* enthält die Startadresse des ersten Ringpuffers.
- *CBER1* enthält die Endadresse des ersten Ringpuffers.
- *CBSR2* enthält die Startadresse des zweiten Ringpuffers.
- *CBER2* enthält die Endadresse des zweiten Ringpuffers.
- *CBCR* (Ringpuffer-Statusregister) hat folgenden Aufbau:



- *CAR1* gibt an, mit welchem Auxiliary Register der erste Ringpuffer verbunden ist
- *CENB1* zeigt an, ob der erste Ringpuffer aktiviert (=1) oder deaktiviert (=0) ist
- *CAR2* gibt an, mit welchem Auxiliary Register der zweite Ringpuffer verbunden ist
- *CENB2* zeigt an, ob der zweite Ringpuffer aktiviert(=1) oder deaktiviert (=1) ist

Falls einer der zwei Ringpuffer aktiviert ist, so überprüft die ARAU nach einer indirekten Adressierung, ob das in CAR1 beziehungsweise CAR2 angegebene Auxiliary Register immer noch zwischen der Start- und Endadresse des Ringpuffers ist. Falls die Adresse über der Endadresse ist, so wird das Auxiliary Register auf die Startadresse gesetzt. Ebenso wird es auf die Endadresse gesetzt, wenn es die Startadresse durch Dekrementieren unterläuft.

4 Befehlssatz

Der folgende Auszug aus dem Befehlssatz des TMS320C50 ist nur eine kleine Auswahl und dient bloß der Übersicht über die Möglichkeiten des Signalprozessors. Sie ist keinesfalls vollständig. Allerdings sieht man, wie die Befehle für die digitale Signalverarbeitung vor allem für die Filterberechnung zugeschnitten sind.

4.1 Befehlssatz für die ALU

Bei den Befehlen für die ALU ist der erste Operand immer der Akkumulator. Der zweite wird mit den vorher genannten Adressierungsarten angesprochen oder als Immediate-Operand übergeben.

Befehl	*Beschreibung*
ADD	Addiert einen Wert auf den Akkumulator
ADDB	Addiert den Akkumulator-Puffer auf den Akkumulator
CMPL	Komplementiert den Akkumulator
AND	Und-Operation
OR	Oder-Operation
ROL	Rotiert den Akkumulator nach links
ROR	Rotiert den Akkumulator nach rechts
SFL	Schiebt den Akkumulator nach links
SFR	Schiebt den Akkumulator nach rechts

4.2 Befehlssatz für den Multiplizierer

Bei Befehlen für den Multiplizierer wird zum Teil auch die ALU angesprochen: Wenn ein Multiplikation mit einer Addition ausgeführt wird, so wird das TREG0 mit dem Operanden multipliziert und das letzte Ergebnis der Multiplikation von der ALU auf den Akkumulator aufaddiert. Diese Befehle werden vor allem bei der Filterberechnung gebraucht.

Befehl	*Beschreibung*
LT	Lädt das TREG0-Register mit einem Wert
LTA	Lädt das TREG0-Register mit einem Wert und addiert das vorhergehende Produkt
MPY	Multipliziert einen Wert mit dem TREG0-Register
MPYA	Mulitpliziert wie MPY und addiert das vorhergehende Produkt auf den Akkumulator
MAC	Multipliziert und addiert Werte
MACD	Multipliziert und addiert Werte mit einem zusätzlichen Verschieben der Daten
SQRA	Quadriert einen Wert und addiert das vorhergehende Produkt

4.3 Befehlssatz für die PLU

Die PLU ist eine zu der ALU parallele Logik-Einheit, die folgende Befehle ausführen kann:

Befehl	*Beschreibung*
APL	Und-Operation
OPL	Oder-Operation
SPLK	Speichert einen long immediate Wert in den Daten-Speicher
XPL	Exklusiv-Oder-Operation

4.4 Befehlssatz für die Systemkontrolle

Hier sind neben bedingten und unbedingten Programmverzweigungen und Unterprogrammaufrufe vor allem die Wiederholungsanweisungen auffällig. Sie spielen in Kombination mit den Befehlen der ALU und des Multiplizerers eine wichtige Rolle bei der Filterbrechnung.

Befehl	*Beschreibung*
B	unbedingter Sprung
BCND	bedingter Sprung
CALL	Unterprogramm-Aufruf
CC	bedingter Unterprogramm-Aufruf
RET	Unterprogramm-Rücksprung
RETC	bedingter Unterprogramm-Rücksprung
RPT	Wiederholung der nächsten Anweisung
RPTB	Wiederholung eines bestimmten Blocks

5 Das DSP Starter Kit für den TMS320C50

Das DSP Starter Kit (DSK) von Texas Instruments besteht zum einen aus dem DSK-Board, das eine Platine mit dem DSP TMS320C50 ist. Zum anderen enthält das Paktet MS-DOS Programme, die zur Erstellung und zur Ausführung von DSP-Programmen auf der Testplatine gedacht sind. Dieses Kit wird von Texas Instruments angeboten, um Entwickler die Möglichkeit zu geben, DSP-Programme in einer Testumgebung zu entwickeln.

5.1 Software

Die Software besteht erstens aus den drei Hauptprogrammen:

- Der Assembler „dsk5a“
  Der Assembler erzeugt aus einem in Maschinensprache geschriebenem DSP-Programm eine so genannte DSK-Datei, die dann mit Hilfe des Loaders oder des Debuggers auf den DSP gespielt wird.
- Der Debugger „dsk5d“
  Der Debugger bietet die Möglichkeit, ein DSP-Programm auf den DSP zu spielen und es anschließend zu debuggen, um entsprechend Fehler zu suchen oder das Programm nachzuverfolgen.
- Der Loader „dsk5l“
  Der Loader spielt über die serielle Schnittstelle eine DSK-Datei auf den DSP und startet die Ausführung des Programms auf dem DSP.

Zusätzlich sind auch viele Beispiel DSP-Programme enthalten, die einerseits die Leistungsfähigkeit des DSP zeigen und andererseits den Entwicklern als Anschaungsmaterial dienen sollen.

5.2 Das DSK-Board

Das DSK-Board ist wie folgt aufgebaut und besitzt folgende Komponenten:

DSP TMS320C50 ist der DSP des Boards.

Boot Rom ist ein 32K × 16 Bit ROM, das das Initialisierungsprogramm enthält. Da der Speicher des DSP bis auf das interne ROM flüchtig ist, ist es nötig den DSP nach dem Einschalten zu initialisieren. Dies geschieht mit Hilfe des Boot Rom, das entsprechende Programme für die serielle Schnittstelle

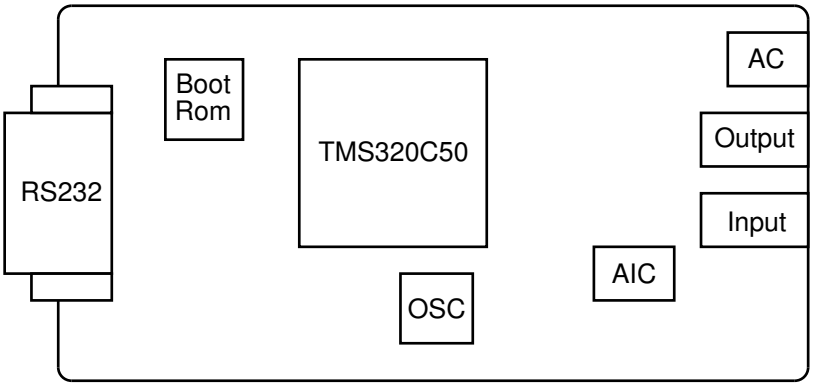


Abbildung 16: Das DSK-Board

enthält, damit der DSP anschließend neue Programme von dem Computer erhalten kann. Nach dem Initialisieren wird das Boot Rom wieder aus dem Adressraum genommen, und anschließend besteht kein Zugriff mehr auf das ROM.

OSC ist der Taktgeber für den DSP. Er läuft mit einer Frequenz von 40 MHz.

AIC ist der Analog/Digital und Digital/Analog Wandler, der benötigt wird, um die analogen Eingangssignale in digitale Werte umzuwandeln und die vom DSP gerechneten digitalen Werte wieder in analoge Ausgangssignale umzuwandeln. Dieser AD/DA-Wandler arbeitet mit einem Takt von 19kHz.

RS232 ist eine serielle Schnittstelle, mit der der DSP an einen Computer angeschlossen werden kann. Über sie können dann die DSP-Programme vom Computer dem DSP übertragen werden. Aber es können auch während der Laufzeit Daten zwischen DSP und Computer ausgetauscht werden.

Input ist der analoger Chinch-Eingang, von dem der DSP seine Signale erhält.

Output ist der analoge Chinch-Ausgang, an dem die vom DSP gerechneten Signale ausgegeben werden.

AC ist die 9 Volt Stromversorgung.

6 Beispielprogramme

Das erste Beispielprogramm verdeutlicht die Mächtigkeit eines MACD-Befehls (**M**ultiply and **Ac**cumulate with **D**ata move) in Kombination mit einer RPT-Anweisung. Dabei wird mit der Befehlsfolge

```
RPT #3
MACD #h0, *-
```

jeweils vier mal zwei verschiedene Werte miteinander multipliziert und die Produkte auf dem Akkumulator aufaddiert. Und zwar ist der erste Operand ab der Adresse h0 zu finden und der zweite ab der Adresse des aktuellen Auxiliary Register. Zusätzlich wird nach jeder Multiplikation die Adresse des ersten Operanden um eins erhöht und die des zweiten um eins erniedrigt. Außerdem werden die Daten, die indirekt adressiert werden, jeweils in die nächst höhere Speicherzelle kopiert.
Für die nächsten Beispiele werden auf dem Computer per MP3-Player Musikstücke abgespielt, die über den Line-Out des Rechners in den analogen Eingang des DSK-Boards gelangen. Das DSK-Board verarbeitet diese Signale, und gibt die veränderten Musikdaten über den analogen Ausgang wieder aus. Der Ausgang ist mit einem Lautsprecher verbunden, der die verarbeiteten Musikstücke hörbar macht. Als Beispielfilter werden ein Tiefpass und ein Hochpass verwendet, der entsprechend die hohen beziehungsweise tiefen Töne herausfiltert. Ebenso verleiht eine Hallfilter dem Musikstück ein Echo.

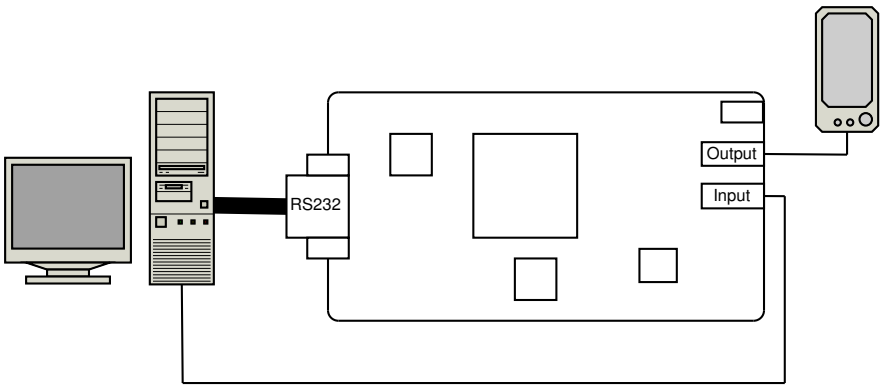


Abbildung 17: Dsk-Board-Konfiguration für die Beispielprogramme

Literatur

[1] Gerhard Doblinger: *Signalprozessoren - Architekturen, Algorithmen, Anwendungen;* J. Schlembach Fachverlag, 2000

[2] Martin Jaiser und Dr. Finbarr Moynihan: *Zur Anatomie von Mikrocontroller und DSP;* Elektronik 1/2002

[3] Jeff Bier und Cornelius Kellerhoff: *Qual der Wahl - der richtige DSP;* Elektronik 10/2000

[4] Cornelius Kellerhoff: *Durchblick im DSP-Dschungel;* Elektronik 13/2000

[5] Cornelius Kellerhoff: *Viele Wege führen zum Ziel - Hochleistungs-DSPs für die Kommunikation;* Elektronik 9/2000

[6] Axel Sikora: *Der DSP-Report 2000;* 1. und 2. Teil, Elektronik 9/2000

[7] Axel Sikora: *Der DSP-Report 2001;* Elektronik 23/2001

[8] Craig Marven und Gillian Ewers: *a simple approach to digital signal processing;* Texas Instruments, 1993

[9] Texas Instruments: *TMS320C5x DSK - Applications Guide;* Texas Instruments, 1997

[10] Texas Instruments: *TMS320C5x DSP Starter Kit - User's Guide;* Texas Instruments, 1994

[11] Texas Instruments: *TMS320C5x - User's Guide;* Texas Instruments, 1993