Signalprozessorfamilie ADSP21xx

Dieter Reiermann, N, TGM ADSPSIN.EXE

Die Architektur der ADSP21xx-Familie

Digitale Signalprozessoren (DSPs) müssen die Signalverarbeitungsalgorithmen der analogen Schaltungen durch ein Programm aus schnellen Befehlen nachbilden. Da dazu komplexe arithmetische Operationen erforderlich sind, müssen durch die DSP-Befehle komplexe Datenverarbeitungen in kurzer Zeit vorgenommen werden. Bei DSPs brauchen auch komplexe Befehle nur einen Maschinenzyklus. Dazu ist ein hohes Maß an Parallelität und ein großer Informationsgehalt im Befehlscode nötig. Parallelität ist nur durch ein weitgehendes Aufteilen der Daten- und Adreßströme über getrennte Busse möglich. Der Befehlscode muß der zweiten Forderung entsprechend breit sein. Bei der ADSP21xx-Familie gibt es zwei Datenbusse und zwei Adreßbusse (modifizierte Harvardarchitektur):

Datenbus	für	Daten	16 Bit
Adreßbus	für	Daten	14 Bit
Datenbus	für	Befehle	24 Bit
Adreßbus	für	Befehle	14 Bit

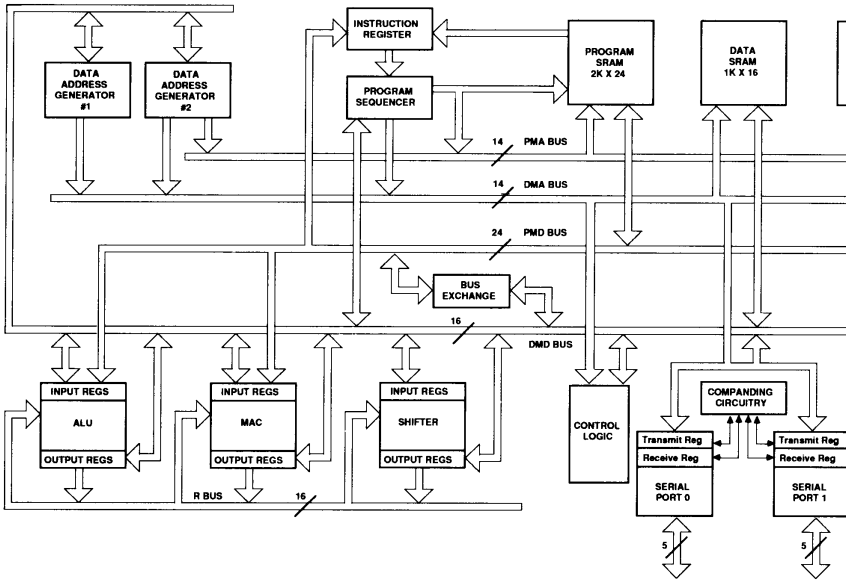


Abb.1: Architektur des ADSP2101

ALU Arithmetisches und logisches Rechenwerk

Die ALU ist das erste von 3 Rechenwerken des ADSP2101-Familie. Über die Eingangsregister AX0, AX1 und AY0, AY1 sowie über das Feedback-Register AF und den Result-Bus können 16 Bit Daten addiert, subtrahiert und (sequentiell) dividiert werden. Die Eingangsdaten können auch direkt vom Datenspeicher oder Programmspeicher bezogen werden. Das Ergebnis im AR-Register kann über den Result-Bus dem Daten-Daten-Bus zugeführt werden und im gleichen Prozessorzyklus im Datenspeicher abgelegt werden (siehe Befehlssatzüberblick- Multifunktionsbefehle). Im ASTAT-Statusregister werden arithmetische Flags aus den ALU-Operationen abgelegt.

Beispiele:

IF EQ AR=AX0+AY0;	Wird durchgeführt, wenn AR=0
IF GE AR=AX0- AY0+C-1;	Wird durchgeführt, wenn AR>=0, Borrow berücksichtigt
IF LT AR=PASS Y0;	AR wird von AY0 geladen, Flags werden modifiziert. Wird durchgeführt, wenn AR <0.
IF EQ AR=-AY0;	
IF NE AF=NOT AX0;	
IF NEG AF=ABS AX0;	Absolutbetrag von AX0 in AF
IF GT AF=AF+1;	
IF EQ AR=AY1-1;	

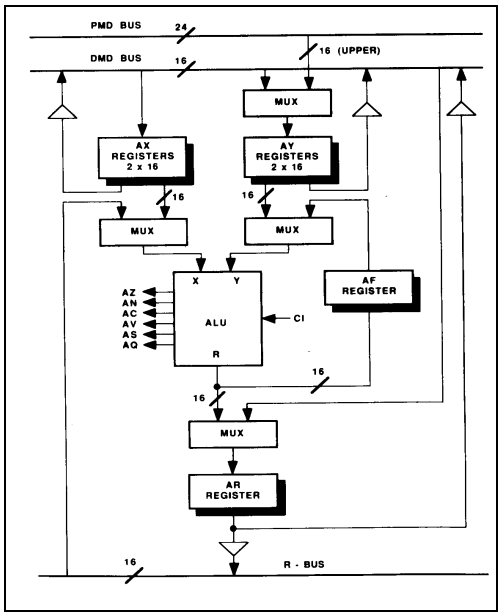


Abb.2: ALU

MAC Multiplizierendes und summierendes Rechenwerk

Dieses Rechenwerk ist für Multiplikationen und darauffolgende Additionen bzw. Subtraktionen zuständig. Die MAC-Rechenoperationen werden zur Programmierung von Differenzengleichungen, den Differentialgleichungen digitalen Signalverarbeitung, benötigt. MX0 und MX1 bzw. MY0 und MY1, sowie das MF-Register und der Result-Bus können für

die Operanden verwendet werden. Das 32 Bit Ergebnis kann mit dem Ergebnis der vorherigen Multiplikation addiert oder subtrahiert werden. Das MR-Ergebnisregister kann bis zu 40 Bit aufnehmen, besteht also eigentlich aus 2 16- und einem 8 Bit Register. Wie bei der ALU können die Eingangsdaten auch aus dem Daten- oder Programmspeicher bezogen werden oder in den Datenspeicher zurückgeliefert werden. Das ASTAT-Statusregister enthält das wichtigste Flag, das Overflow-Flag des MAC (siehe: Statusregister).

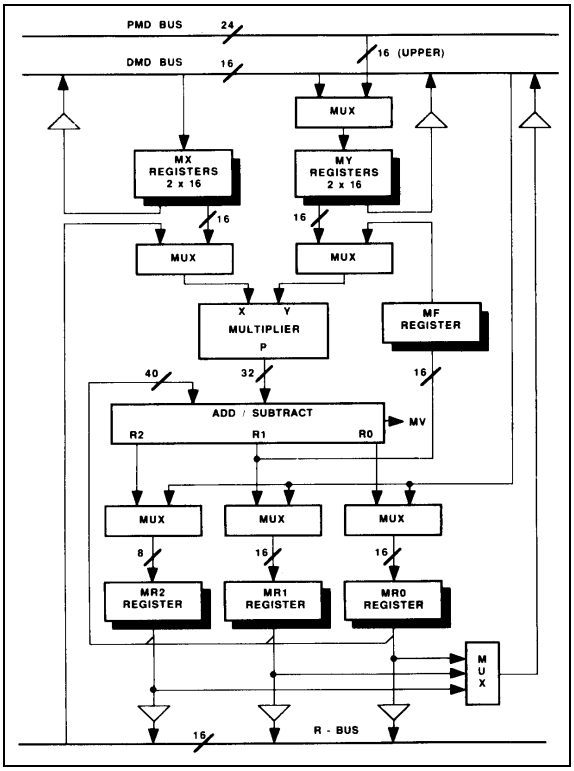


Abb.3 MAC

Beispiele:

*IF GE MR=MR+MX0*MY0(SS);*	Das Produkt aus MX0 und MY0 (beide Operanden Vorzeichen) wird zum Inhalt von MR addiert, wenn MR>= 0.

Besondere MAC-Befehle:

IF GT MR=0;	MR auf Null setzen, wenn sein Inhalt größer als Null.
IF EQ MF=MR;	MF wird aus MR geladen.
IF MV SAT MR;	Wenn Überlauf, dann wird MR auf die größte positive bzw. kleinste negative Zahl gebracht

SHIFTER Verschieben und Normieren

Der Shifter kann in einem Prozessorzyklus das 16-Bit Wort des Shifter-Inputregisters (SI) an irgendeine Stelle des 32 Bit Shifter Outputs plazieren. Die OR-PASS-Logik ermöglicht das Verschieben von 32 Bit Worten mit zwei Befehlen. Die Anzahl der Verschiebungen kann auch im Shifter-Exponent-Register (SE) variabel gehalten werden. Um die Dynamik von Eingangssignalen verarbeiten zu können, gibt es ein Block-Exponent-Register, in dem ein binärer Skalierwert für einen ganzen Datenblock abgelegt werden kann. Der dazugehörige NORM-Befehl führt dann die Skalierung für jeden Signalwert durch. Die Verschiebeergebnisse des Shifters können aus den beiden Ergebnisregistern SR0 und SR1 gelesen werden.

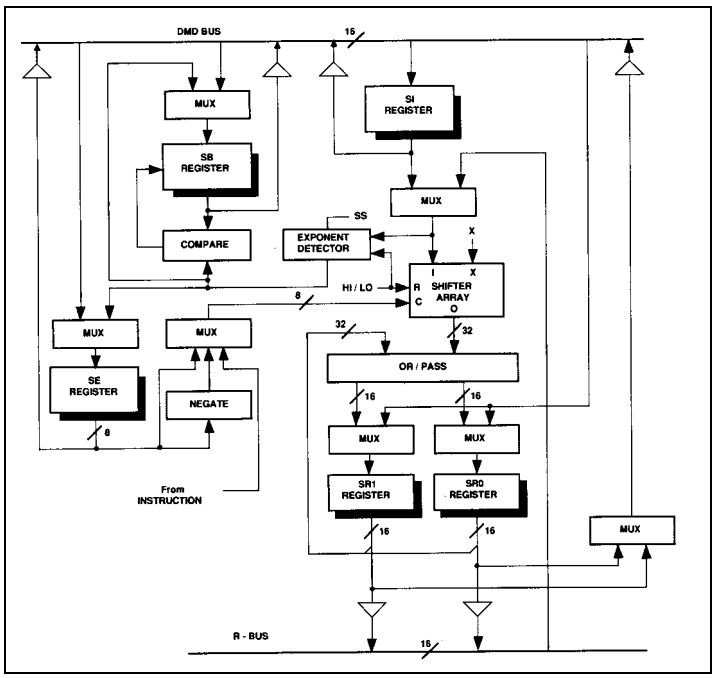


Abb.4 Der Shifter

Beispiele:

IF LT SR = SR OR ASHIFT SI (LO);	Verschieben von SR0 mit Beibehaltung des Vorzeichens und Verodern mit dem "alten" Inhalt von SR bei SR < 0.
IF GE SR = LSHIFT SI (HI);	Verschieben von SR1, wenn SR>=0 ist.
SE=EXP MR1 (HIX);	SE wird mit der Anzahl der führenden Vorzeichenbits geladen. Wenn vorher ein Überlauf (AV gesetzt) stattgefunden hat, wird SE auf +1 gesetzt
SR = NORM SI (HI);	Die Anzahl der Verschiebungen steht in SE. Es wird in SR1 verschoben.
SB = EXPADJ SI;	SB wird mit aktüllem Exponenten geladen, wenn dieser größer ist als der vorher in SB abgelegte.

DAG1, DAG2 Adreßgenerator für Daten

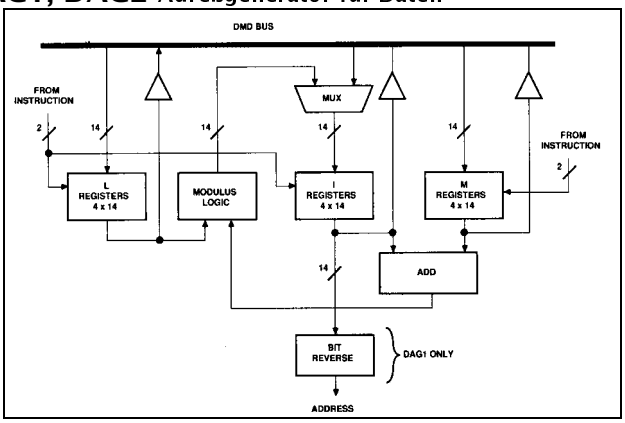


Abb.5 Die Adreßgeneratoren für Daten

Überaus komfortabel erfolgt die Adressierung von Daten. Zwei Adressierungswerke (DAG0, DAG1) enthalten je 4 Pointer, die über je 3 16 Bit Register beeinflußt werden können:

I0..I3 (DAG0)	bzw.	*I4..I7 (DAG1)*	Basisadressen
L0..L3 (DAG0)	bzw.	*L4..L7 (DAG1)*	Längen der Umlaufspeicherblöcke
M0..M3 (DAG0)	bzw.	*M4..M7 (DAG1)*	enthalten die Offsets zur nächsten Adresse in einem Abarbeitungszyklus (Veränderungswert)

Die nächste Adresse errechnet sich immer aus

(aktuelle Adresse + Veränderungswert - Basisadresse) MODULO (Länge) + Basisadresse

Das BIT REVERSE-Adreßrechenwerk des DAG1 kann die Reihenfolge der Bits einer Adresse umdrehen (wird für FFT-Algorithmen gebraucht).

Beispiele:

I7 = 0;	
I0 = ^data_buffer;	I0 wird Beginnadresse des Speicherbereichs
M0 = 1;	Adreßveränderungswert
L0 = %data_buffer;	L0 wird Länge des Speicherbereiches
M7 = M1;	Einfache Registeradressierung
PM(I6, M5) = AR;	
MX1 = PM(I6, M5);	MX1 wird mit dem Inhalt des Programmspeicherplatzes auf den I6 mit M5 zeigen, geladen.
DM(codec) = AR;	AR wird auf die Adresse *codec* kopiert
DM(I0, M1) = 0x3800;	

Die Adreßregister-Modifizierbefehle werden zur Veränderung der DAG-Zeiger eingesetzt:

MODIFY (I2 , M2)	I2 wird I2 + M2

Die leistungsstärksten Befehle des ADSP21xx werden mit Hilfe der DAGs abgewickelt. Diese Multifunktionsbefehle können in einem Maschinenzyklus eine ALU-,MAC- oder SHIFTER-Operation und einen Datentransport über die DAG-Pointer durchführen:

AR = AX0 + AY0, AX0 = DM(I0, M0);	Nach der Addition durch die ALU wird AR aus dem DM nachgeladen
*IF NOT MV MR = MR+MX0 *MY0, MX0 = DM(I1,M1),MY0=PM(I4,M4);*	Nach der MAC-Operation wierden die Operanden aus dem DM bzw. PM nachgeladen

PROGRAM SEQUENCER Programmsteuerwerk

Der Program Sequencer enthält 4-Level-Stacks für das Counter Register für verschachtelte Zählschleifen, die Statusregister, die vor dem Start einer Interruptserviceroutine automatisch abgespeichert werden, für verschachtelte Umlaufspeicher (LOOP-Stack) und natürlich für Rückkehradressen (PC-Stack). Für die von Bedingungen abhängigen ALU-, MAC- und SHIFTER-Befehle gibt es eine CONDITION LOGIC, Interrupts werden vom INTERRUPT-CONTROLLER abgewickelt. Das SYSTEM CONTROL REGISTER wird zur Einstellung von WAIT-States, BOOT-Page und SPORTs verwendet. Im DATA MEMORY WAIT STATE REGISTER werden Wait-States (bis zu 7) für die DatenRAM-Waitzonen festgehalten (siehe unten).

Zu den Programmsteuerbefehlen gehören Sprungbefehle, Unterprogrammaufrufe, RETURN von Unterprogrammen und Interruptsubroutinen. Sie können abhängig von einer Bedingung sein. Die Bedingungen sind prinzipiell dieselben wie die der Rechenbefehle. CALL und JUMP können auch von FLAG_IN abhängig sein. Der Schleifenbefehl DO...UNTIL ist abhängig vom Zählerstand im CTR-Register. Die Adressierung der Sprünge und Unterprogrammaufrufe erfolgt direkt oder über die I-Register (I4 .. I7). Der Befehl IDLE schaltet den ADSP2101 solange in den LOW-POWER-Zustand, bis eine Interruptanforderung eintrifft. SET, RESET und TOGGLE werden zur Veränderung des Ausgangspins FLAG_OUT des ADSP2101 verwendet.

Beispiele:

IF NOT CE JUMP (I7);	Springe, wenn Zähler noch nicht fertig
IF AV CALL Scale_Down;	Unterprogrammaufruf, wenn ALU-Überlauf
IF FLAG_IN JUMP Event;	Springe, wenn der FLAG_IN Input High ist

BUS EXCHANGE: Für Daten, die vom PM in das DM oder für Befehle, die vom DM in das PM übertragen werden.

Beispiel:

I0 = -1;	Datenadressierung dekrementierend
AR = DM(I0,M0); *PX = DM(I0,M0);* *PM(I4,M4) = AR;*	3 Byte sollen im PM aus dem DM abgelegt werden. Es werden die 2 MSBytes über AR in das PM gebracht, dabei wird das LSByte automatisch aus dem PX-Register gelesen.

Nur für den Single-Chip DSP ADSP 2101 kommt On-Chip-Programm- und Datenspeicher dazu:

PROGRAM RAM	2kWorte	24Bit / Wort
DATA RAM	1kWorte	16Bit/Wort

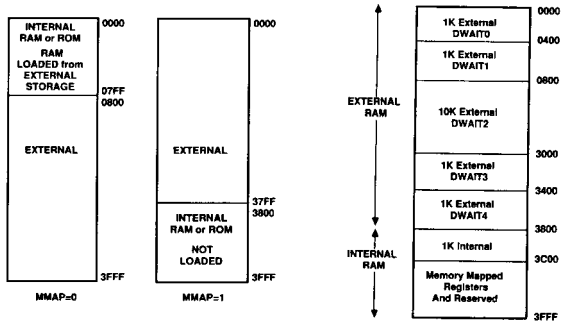


Abb.6 Speicheraufteilung beim ADSP2101

Der externe Datenspeicher ist in Zonen für unterschiedlich schnelle Speicherbausteine unterteilt. Dadurch wird es möglich, der Applikation entsprechend eine optimale Speicherbausteinauswahl zu treffen.

BOOT ADDRESS GENERATOR

Um auf einfache Weise Programme von einem externen Speicher in das schnelle interne RAM zu bekommen, muß das Boot-Programm von einem externen Programmspeicher (BOOT-EPROM) in Blöcken von 8kByte geladen bzw. nachgeladen werden. 3 Byte entsprechen immer einem Befehl, ein 4. Byte wird im BOOT-EPROM mit FFH belegt und hat nur Auffüllbedeutung. Daher können in einer Seite 2k Befehlsworte zu 24 Bit stehen. Das Laden aus dem BOOT-EPROM besorgt der Boot-Address-Generator. Selbstverständlich kann auch vom externen Programmspeicher aus gestartet werden. Dazu muß der Anschluß MMAP auf High geschaltet werden.

SPORT 0, 1 Serial Port

Die Übertragungsgeschwindigkeit geht bis zur halben Taktfrequenz des Chips, beide SPORTS sind doppelt gepuffert, können bis zu 16 Bit Worte übertragen und haben eine Hardware-Compander zur Komprimierung und Dekomprimierung der ankommenden und abgehenden Daten. Zusammen mit den DAGs kann ein Zirkularpuffer aufgebaut werden, wobei nur 1 Maschinenzyklus für Senden und Empfangen eines Datenwortes benötigt wird. Dementsprechend werden Interrupts nach Senden oder Empfangen von jedem Datenwort oder von einem ganzen Datenblock aus dem zirkularen Pufferspeicher angemeldet. SPORT 0 kann darüber hinaus als Mehrkanalschnittstelle für Multiprozessorschaltungen verwendet werden. Die Anschlüsse von SPORT 1 können auch als externe Interrupts (IRQ0 und IRQ1) und als allgemeine Ein- bzw. Ausgänge (FLAG-IN und FLAG-) eingesetzt werden.

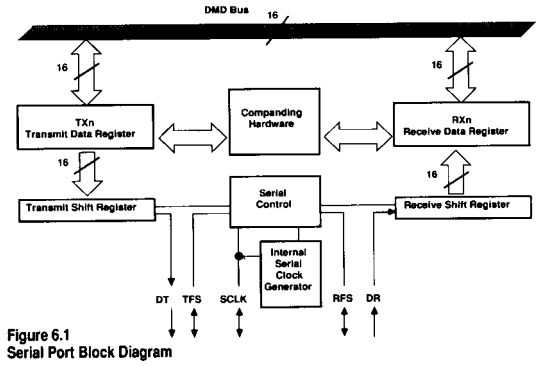


Abb. 7 Die seriellen Ports

Zur Steuerung der SPORTs dienen die folgenden memory-mapped Register:

SPORT0-CONTROL	Framing, Serial Word Length, Framing Signal Inversion Companding mode (A oder µ-Law), Multichannelsteuerung
SPORT0-SERIAL CLOCK DIVIDE MODULUS	Hält die Teilerzahl für den Takt der synchronen Übertragung
SPORT0-RECEIVE FRAME SYNC DIVIDE MODULUS	Hält die Teilerzahl für die Einfügung von Sync-Impulsen
SPORT0 AUTOBUFFER CONTROL	Festlegung der Umlaufspeicher für automat. Puffern der einlangenden oder abgehenden Daten
4 SPORT0 MULTICHANNEL WORD ENABLE REGISTERS	zur Auswahl der zu sendenden oder zu empfangenden Kanäle aus einer Auswahl von 24 oder 32
SPORT 1-CONTROL	Wie oben
SPORT1-SERIAL CLOCK DIVIDE	Wie oben
SPORT1-RECEIVE FRAME SYNC	Wie oben
SPORT1-AUTOBUFFER CONTROL	Wie oben

TIMER

Der Timer des ADSP2101 (bzw. ADSP2105) ist als 16 Bit Intervallgeber mit automatischer Wiederholung zur Einstellung von Zeitintervallen. ausgelegt. Dazu werden das TCOUNT als eigentliche Zählregister und das TPERIOD-Register, das den Nachladewert hält, verwendet. Mit dem 8 Bit TSCALE-Register wird der Prozessortakt für den Timer geteilt. Es wird immer jeder N+1-te Taktimpuls verwendet, wenn N die in TSCALE abgelegte 8-Bit-Zahl ist. Daraus ergibt sich für 12.5MHz Taktfreqünz ein maximales Zeitintervall von 1.34s. Der Timer kann selbstverständlich einen Interrupt hervorrufen (siehe dort).

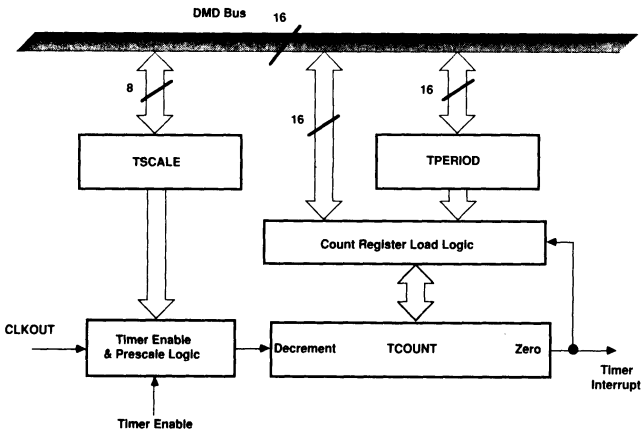


Abb.8 Timer

INTERRUPT

Die Interrupts der ADSP21xx-Familie sind festen Adressen zugeordnet. Beim ADSP2101 sind das:

IRQ2	0004H	höchste Priorität
SPORT0-TX	0008H	
SPORT0-RX	000CH	
SPORT1-TX oder IRQ1	0010H	
SPORT1-RX oder IRQ0	0014H	
TIMER	0018H	niedrigste Priorität

Zur Bearbeitung dienen 3 Register:

IMASK	6 Bit	Interrupt-Enable Register zum Einschalten der Interruptquellen
ICNTL	5 Bit	IRQ-Pegel- oder Flankensteuerung Interrupt-Nesting-Enable
IFC	12Bit	Je ein Bit zur Aktivierung und je ein Bit zur Abschaltung einer Interruptanforderung vom Programm aus

STATUS-REGISTER

Der ADSP2101 (ADSP2105) enthält 3 Status-Register:

ASTAT	6 Bit	arithmetische Flags: AZ, AN, AV, AC, AS, AQ, MV, SS (ALU-Zero, -Negativ, -Overflow, -Carry, -X-Input-Vorz., -Quotient, MAC-Overflow, SHIFTER-Input-Vorzeichen
SSTAT	8 Bit	Stack-Flags: PC-,Count-,Status-,Loopstack: EMPTY oder OVERFL
MSTAT	7 Bit	Verschiedene Steuerflags: (Register Bank Select, Bit Reverse Mode, ALU-OV-Latch Mode, AR Saturation Mode, MAC Result Format,Timer Enable, Go Mode)

Die meisten Bedingungen der arithmetischen Befehle (IF EQ.....) werden aus den Flags von ASTAT berechnet. Zur Veränderung des MSTAT Registers gibt es zwei eigene Befehle:

Beispiele:

```
ENA M_Mode;
DIS Bit_Rev;
```

Der ADSP2101 und seine Peripherie

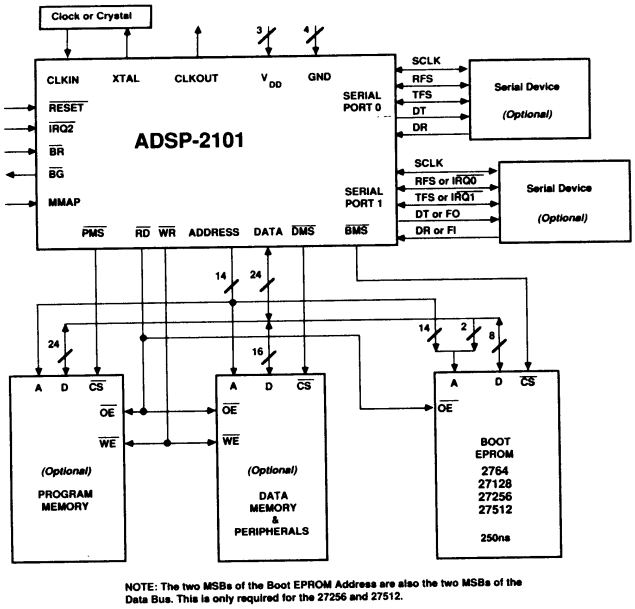


Abb.9 ADSP2101 -Peripherie

DMS	Data Memory Select	
PMS	Program Memory Select	
BMS	Boot Memory Select	Die 2 MSBs der Boot-Eprom-Adresse werden vom Datenbus bezogen (Bit 14 und Bit 15)
BR	Bus Reqüst	
BG	Bus Grant	Während der Bus abgegeben ist, kann intern weiter gearbeitet werden (Go-Mode), bis ein Zugriff auf das externe Memory erfolgt.
MMAP		Steürt den Bootvorgang: MMAP=0 ... Laden von Boot-Page 0
RESET		Rücksetzen der Stackpointer beschalten aller Interruptqüllen (IMASK) STAT wird Null gesetzt

Externe I/O-Peripherie wird memory-mapped angesprochen.

Die Demonstrationsprogramme im Einzelnen:

DTMF	Über DAC0 und Analog Output	Telefonnr. von Analog Devices
ECHO	Analog Input, Analog Output	Echoeinfügung und -unterdrückung
FFT	Analog Input, DAC0	Es wird eine 512 Punkt FFT durchgeführt und als y(t) Signal, mit Triggerimpulsen versehen, ausgegeben. Darstellung des Spektrums mit linearem und logarithmischen Rechteckfenster und mit Hanningfenster (logarithmisch).
FIR	Analog Input, Analog Output	4 FIR-Bandpässe mit Analog Input, 4 mit internem Pseudorauschen als Eingangssignal
GSM	Sprache über Analog Input, Analog Output	Die für die GSM-Mobiltelefontechnik notwendige "Comfort-Noise-Insertion" wird demonstriert.
GRAPHIK	DAC0, DAC2	3D-Rotation des Schriftzuges "ADSP2101" für Oszilloskop im x-y-Betrieb.
MUSIK	Analog Out	Synthesizer-Demoprogramm.

Das EZ-LAB Entwicklungsboard

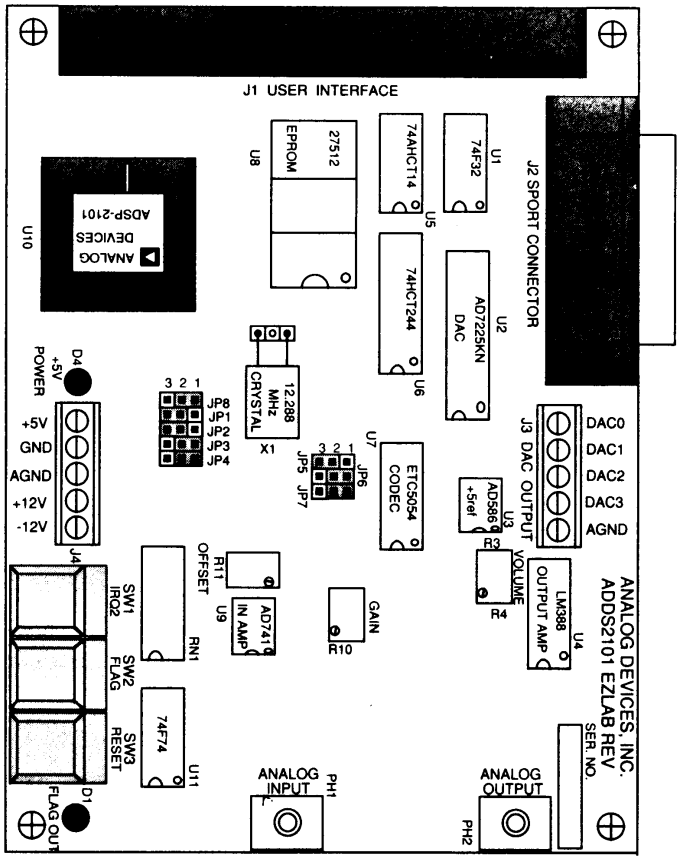


Abb.10 Das EZ-LAB-Board

Mit dem EZ-LAB kann bei Verwendung eines EPROM-Simulators zur Simulation des 256k Boot-EPROMs auf kostengünstige Weise Programmentwicklung für den ADSP2101 betrieben werden. Es besteht aus

ANALOG INPUT	für Standard-Mikrofone (ca. 10mV)
ANALOG OUTPUT	für Standard-Kopfhörer
CODEC	kompandierender ADC-DAC für den Frequnzbereich ca. 300Hz bis 3400Hz
DAC	linear, 4 Kanäle (0 ... 4.9805V (255/256 x 5V), Vorzeichen in Offset-Darstellung)
FLAG IN Taste	
FLAG OUT Led	
IRQ2 Taste	
RESET Taste	
8 JUMPER	Umschaltung SPORT /externer INTERRUPT, BOOT EPROM ENABLE, ...
USER INTERFACE	60 poliger Stecker mit Adress-,Daten und Steürbus
SPORT Connector	25 polige SUB-D-Steckbuchse für SPORT0 und SPORT 1

Starten der eingebauten Demoprogramme:

- Versorgung einschalten
- Mikrofon und Kopfhörer anschließen
- Oszilloskop an DAC0 und DAC2 anschließen
- eventuell Funktionsgenerator für FFT bereithalten
- RESET -Taste

Mit der FLAG_IN Taste werden die 7 Demoprogramme (7 Boot-Eprom-Seiten) durchgetastet. Innerhalb der einzelnen Demos wird mit der IRQ 2 -Taste weitergeschaltet.

Programmbeispiel Sinusgenerator:

Ein interessantes Beispiel aus dem Bereich Signalerzeugung ist der nachfolgend beschriebene Quadratur-Oszillator zur Erzeugung von Sinusschwingungen. Ein komplexer Zeiger $r = e^{-j\alpha i}$, i=0 ... 127, wird 128 mal um den Winkel α = 360/128 ° in Gegenuhrzeigerrichtung weitergedreht. Sein neuer komplexer Wert wird durch

$$r_{i+1} = (Re(r_i) + jIm(r_i))\ (\cos\alpha + j\sin\alpha)$$

bestimmt. Da für kleine α cos(α) etwa 1 und sinα α wird, kann eine sehr einfache Rekursionsformel angegeben werden:

$$r_{i+1} = Re(r_i) - \alpha * Im(r_i) + j*(\alpha Re(r_i) + Im(r_i))$$

Daraus kann leicht eine Sinus- bzw. Cosinusschwingung abgeleitet werden.

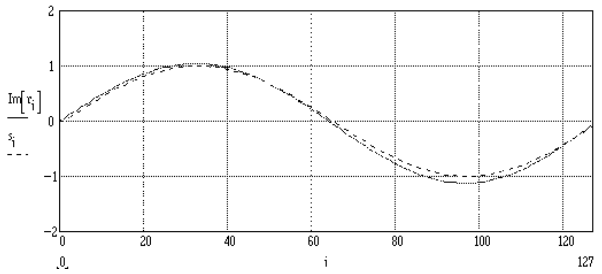
Zunächst die Simulation mit Mathcad (zum Vergleich wird eine "reine" Sinusschwingung s_i dargestellt):

$N := 128 \qquad \alpha := 2\cdot\frac{\pi}{N} \qquad i := 0..2\cdot N-1 \qquad r_0 := 1+j\cdot 0$

$s_0 := 0$

$r_{i+1} := Re(r_i) - Im(r_i)\cdot\alpha + j\cdot Re(r_i)\cdot\alpha + j\cdot Im(r_i) \qquad r_{112} = 0.806 - 0.813j$

$s_{i+1} := \sin(i\cdot\alpha) \qquad s_{112} = -0.741$



Das Programm für den ADSP2101 wurde ohne Timer-Interrupt geschrieben. Damit ist die maximale Frequenz erzielbar. Die Simulation im ADSP2101-Simulator SIM21 ist damit auch einfacher.

Anmerkung: Die Datei ADSPSIN.EXE auf Diskette PCN-DSK-407 enthält ein selbstentpackendes Archiv mit allen für die Simulation erforderlichen Dateien.

```
{Erzeugung von Sinusschwingungen mit einem Quadraturoszillator}

.module/boot=0/abs=0            sinus;

{Deklarieren des DAC}
.port write_dac0_;
.port write_dac2_;
.port load_dac_;

next2sin:   ena m_mode;         {1.15-Fixed Point Format}
            my0=0;              {erster Im-Teil}
            ar=0x0648;          {2*Pi/256}
            mx0=0x3fff;         {erster Realteil, Vorsicht:
                                 wenn zu groß, Überlauf
                                 möglich}
            mr0=0;
            mr1=ar;             {mf hält auch alpha }
            mf=mr(rnd);         {mf = ar = alpha}
            mr1=0;
            cntr=64;            {N=128, N/2, da 2 Outputs/Loop}
            do sin until ce;
            mr1=mx0;
            mr=mr-ar*my0(ss);   {mx1=mx0-alpha*my0....Realteil}
            mx1=mr1;
            mr1=my0;
            mr=mr+mx0*mf(ss);   {my1=mx0*alpha+my0....Im-Teil}
            my1=mr1;
            call output;
{ 2. Sin-Wert wird berechnet; aktüller Zeiger: mx1 + j*my1  }
            mr1=mx1;
            mr=mr-ar*my1(ss);
            mx0=mr1;
            mr1=my1;
            mr=mr+mx1*mf(ss );
            my0=mr1;
            call output;
sin:                        nop;

{Ausgabe für Mathcad (Zweierkomplementdarstell.),
 für EZLAB Binary-Offset notwendig}

output:     dm(write_dac0_)=sr1;    {Aktivieren der DAC-Ausgabe
            dm(load_dac_)=sr1;      {Ausgabe}
            rts;
.endmod;
```

Nach Assemblieren und Linken mit `a.bat sin3t`:

```
asm21 %1 -l
ld21 %1 -a ezlab -e %1 -g -x
```

Wenn mit dem EZLAB getestet werden soll, kommt in `a.bat` noch dazu:

```
spl21 %1 %1 -bm -i >splout.asc    Erzeugen des Ladefiles für
                                  den Epromsimulator
rem dl %1.bnm                     Laden in den Epromsimulator
```

Simulation mit SIM21

```
Rem batch file to set ADSP2101 env. var.
rem @ECHO OFF
set adi_dsp=c:\adi_dsp\
set ADII=c:\adi_dsp\include\
set ADIL=c:\adi_dsp\lib
set ADIRTH=c:\adi_dsp\lib
set ADI_PATH=c:\adi_dsp\misc\sim_2101
set adi_doc=c:\adi_dsp\doc\sim_2101
c:\
cd \
cd adi_dsp
sim2101 -a c:\adi_dsp\ezlab
```

mit den Eingaben im Command-Fenster:

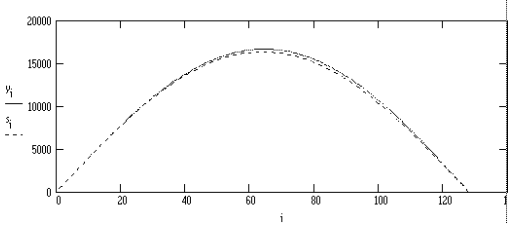
```
l  'sin3t'
cr
o dm[0x2000] > 'sin3sim.dat'
g
o dm[0x2000]
quit
```

können die Sinuswerte nach Umwandeln von hexadezimaler Darstellung in dezimale Darstellung mit

```
hexdec.exe
```

in Mathcad importiert werden und überprüft werden:

$i := 1..128 \qquad y_i := READ(sin3mcd) \qquad s_i := \sin\left[i\cdot 2\cdot\frac{\pi}{256}\right]\cdot 16384$



□