POLITECNICO DI TORINO

ESAMI DI STATO PER L'ABILITAZIONE ALLA PROFESSIONE DI INGEGNERE I
SESSIONE - ANNO 1998

RAMO INFORMATICA TEMA NO. 1

TORINO 5 MAGGIO 1998

Un sistema di analisi e classificazione di un segnale x(t) campionato e digitalizzato alla frequenza fc, deve essere realizzato tramite un microcomputer basato sull'8086 (o su un altro processore della famiglia 80x86), interfacciato tramite alcuni suoi periferici dedicati, fra i quali si possono considerare gli interrupt e DMA controller, 8259 e 8237, e le porte seriali e/o parallele, ad un acceleratore hardware **HWACC** che puó essere realizzato in due modi mutuamente esclusivi:

a) con l'uso del processore per l'elaborazione di segnali ADSP 2101 di cui sono riportate le specifiche nell'allegato 1,
b) oppure con un circuito progettato appositamente dal candidato che puó utilizzare a tal scopo i componenti standard SSI e MSI, tra cui ALU, registtri, contatori, memorie, come pure un moltiplicatore parallelo e combinatorio da 16x8 bit con risultato su 24 bit. Tali componenti possono essere scelti fra quelli commerciali, oppure specificati opportunamente dal candidato.

La fig. 1 indica la connessione dei componenti

La frequenza di campionamento sia indicata con fc.
Si supponga di avere disponibile come ingresso al sistema da progettare HWACC il segnale x(t) campionato e convertito in digitale alla frequenza fc ed indicato con x_n, rappresentato in complemento a 2 su 12 bit in forma parallela, oppure seriale (a scelta del candidato). Tale segnale digitale é fornito da un blocco di acquisizione dati ACQ, di cui non é richiesta la progettazione.
Le funzioni di elaborazione e classificazione possono essere modellate, nel modo semplificato richiesto per la soluzione di questo progetto, come indicato nel seguito (si faccia riferimento alla fig. 2).
Il segnale x_n di ingresso entra in 2 filtri, uno passa basso (HPB) ed uno passa alto (HPA), ambedue del tipo a poli e zero (Filtri IIR), con 4 poli e 3 zeri.
Le uscite corrispondenti y^B_n e y^A_n soddisfano all'equazioni alle differenze del tipo:

$$(1)\ y_n = a_1y_{n-1} + a_2y_{n-2} + a_3y_{n-3} + a_4y_{n-4} + b_0x_n + b_1x_{n-1} + b_2x_{n-2} + b_3x_{n-3}$$

dove y_n vale nei due casi y_n^A ed y_n^B
Le condizioni iniziali non sono importanti perché si puó considerare il funzionamento solo dopo il transitorio iniziale.
Si supponga inoltre che i massimi valori in modulo di a_i e b_i siano < 4, e che |ymax|/|xmax| < 8.
In alternativa alla 1) i filtri possono essere realizzati come filtri non recursivi FIR con 50 coefficienti. Questa soluzione é considerata di ripiego e sará valutata con peso minore rispetto a quella degli IIR.
Le variabili y_n considerate calcolate alla frequenza fc vanno ciascuna raddrizzate ed integrate su 100 valori, nei blocchi I_B ed I_A, ottenendo cosí le variabili z^A_p e z^B_p alla frequenza fc/100.
In formule i blocchi I_B ed I_A di fig. 2 devono realizzare la 2:

$$2)\ z_p = \sum_{k=0}^{99} y_{n+k} \qquad \text{per } n = 0, 100, 200, \ldots r*100, \ldots.$$

I valori di z^A_p e z^B_p sono valutati alla frequenza fc/100, con una risoluzione d'uscita paragonabile a quella dell'x_n di ingresso.

Le funzioni HPB e HPA devono essere realizzate obbligatoriamente nell'HWACC, il classificatore CL nel micro 80x86, mentre i blocchi I_B e I_A possono a scelta del candidato essere realizzati nell'HWACC o sull'80x86.
Il classificatore a ciascun frame ad indice p, per ciascuna coppia di valori z^A_p e z^B_p fornisce una prima uscita classificata nel modo seguente:

C_1 all'istante p, se z_p é prevalentemente di tipo B (l'ampiezza di z_p é molto maggiore di z^A_p)
C_2 all'istante p, se z_p é prevalentemente di tipo A (l'ampiezza di z_p é molto maggiore di z^B_p)
C_3 all'istante p, se z_p é circa uguale (l'ampiezza di z^B_p é circa uguale z^A_p)
C_4 all'istante p, se i due segnali z^A_p e z^B_p sono molto bassi.

Per tutti i valori di p da 0 fino ad un massimo corrispondente al funzionamento del sistema per 60 sec. si determinano i valori di C_i ottenendo una stringa di C_i $0<i<=4$.
Esempio: con fc = 1khz, si avrá fc/100 = 10Hz cioé si dovrá determinare una stringa di 600 valori di C_i.
Considerando infine che le sequenze di C_i dovrebbero contenere idealmente la ripetizione di un solo valore di C_i per un intervallo di tempo compreso tra 1 e 5 secondi, si determini un procedimento primitivo, ma intelligente per tale classificazione, che "filtri" le classificazioni linguisticamente errate, intermedie fra quelle supposte "corrette".
Si consideri per frequenza di campionamento i due casi fc = 1khz, 10Hz e per ambedue i casi si discutano le conseguenze che da esse derivano, fra le quali si indicano, anche se non in modo esaustivo: possibilitá di effettuare in tempo reale le elaborazioni su HWACC, sull'80x86, modalitá di trasferimento dei dati in interruzione, in DMA a secondo della scelta effettuata, possibilitá eventuale di trasferire parte di elaborazioni all'80x86 per alleggerire HWACC.

Il candidato per la soluzione completa del progetto dovrá svolgere le seguenti parti:

1. Progetto dell'HWACC in uno dei due modi mutuamente esclusivi (1a o 1b)
   1.a Progetto come circuito logico dedicato e secondo le tecniche apprese a Reti Logiche
      1.a.1 - Definizione della parte operativa (a blocchi) usando i blocchi indicati all'inizio del testo
      1.a.2 - Definizione in linguaggio tipo RTL (Register Transfer Language) delle operazioni richieste dalla 1 e 2
      1.a.3 - Progetto dell'automa a stati finiti realizzanti il controllo di cui ai punti 1.a.1 e 1.a.2
      L'eventuale formalizzazione di 1.a.2 ed 1.a.3 in un'unica descrizione formale é considerata valida.
   1.b Scrittura di un programma in assembler ADSP 2101 sostitutivo del progetto 1a.
      In questo caso le interfaccia di input ed output sono le seriali 1 e 2 del chip.
   Nei due casi i due filtri passa basso devono essere obbligatoriamente inglobati in HWACC, mentre i blocchi I_A ed I_B possono essere realizzati nel micro 80x86 oppure nel blocco HWACC.
2. Progettazione a blocchi dell'Interfaccia fra HWACC e micro 80x86, indicando il metodo di gestione dell'I/O (Interrupt o DMA o polling).
   Indicare comunque cosa occorre fare nel caso di gestione dell'interruzione HW con int 12H scrivendo le routine MASM86 che gestiscono il salvataggio dell'indirizzo dell'interrupt service routine (ISR) per 12H del BIOS e la sua sostituzione con l'indirizzo dell'ISR che il candidato assegnerá per la gestione dell'I/O progettato.
   Indicare a grandi linee le operazioni che devono essere svolte dalle procedure di I/O (driver).
3. Effettuare l'emulazione di HWACC sul processore 80x86 in linguaggio "C". E' richiesta inoltre la scrittura in MASM86 della funzione che effettua l'operazione di filtraggio, come subroutine, richiamabile "C", di cui si deve fornire il prototipo.

4. Si effettui il programma che implementi I_A ed I_B come funzione "C" se giá non implementato in HWACC.
5. Si progetti il software di classificazione in pseudocodice e si indichino i prototipi in linguaggio "C" di tutte le funzioni ritenute fondamentali (almeno 4).
   Si indichino la definizione delle strutture dati del main sia in memoria che su file (se richiesto).
   Si spieghi la sintassi e la semantica dei parametri e la semantica delle elaborazioni svolte da ciascuna funzione.
   Si implementi in "C" almeno una funzione significativa.

NOTA BENE

Al fine di ottenere un buon giudizio é richiesto svolgere in modo corretto e sufficientemente approfondito la parte 1+2 (progetto HW) oppure la parte 3+4+5 (sviluppo SW).
In ambedue i casi, la rimanente parte non approfondita nei dettagli deve essere tratteggiata almeno a livello generale (schema a blocchi della parte operativa oppure software di classificazione in pseudocodice).

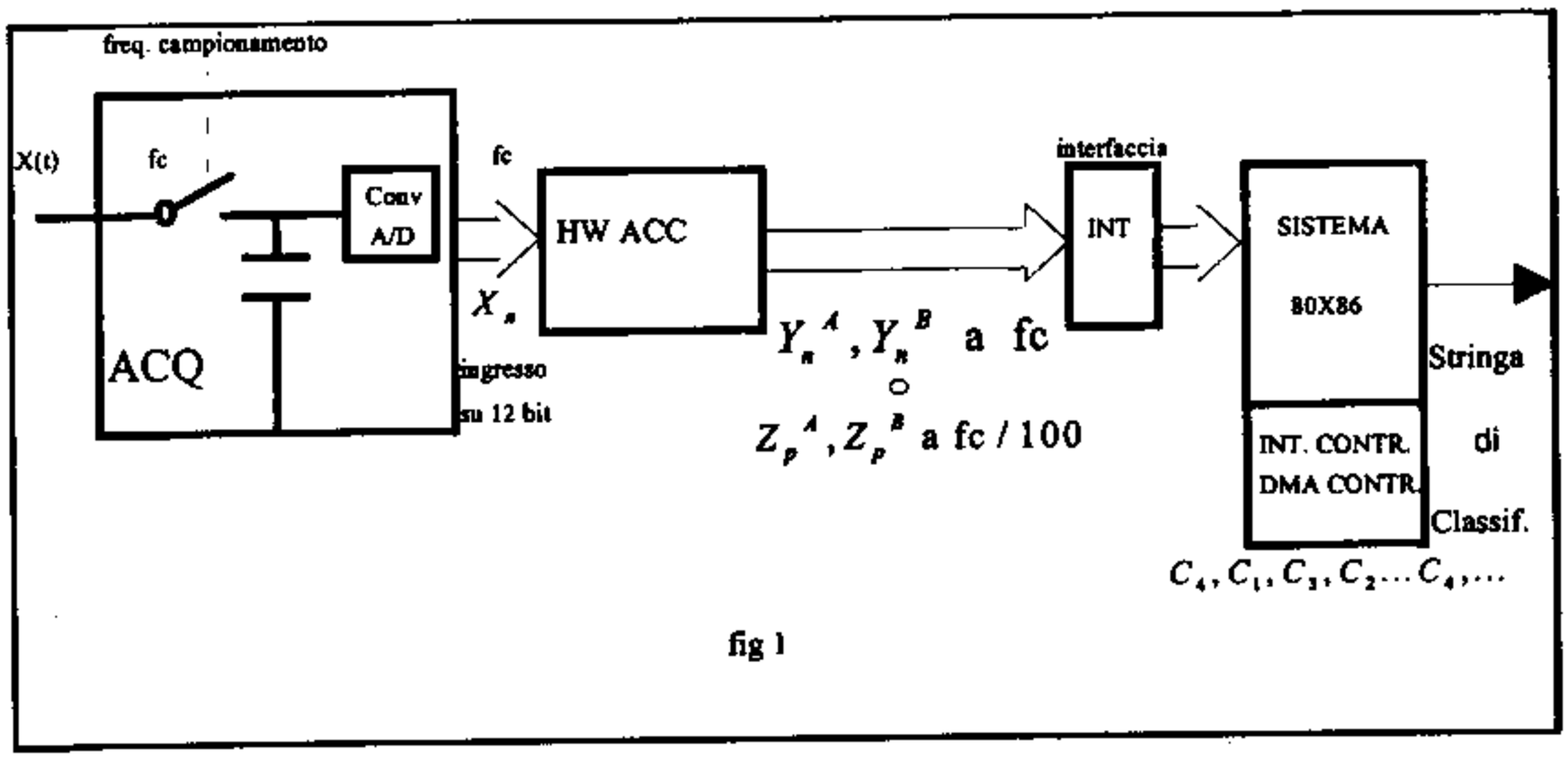

freq. campionamento
X(t)
fc
Conv A/D
ACQ
fc
X_n
ingresso su 12 bit
HW ACC
interfaccia
INT
Y_n^A, Y_n^B a fc
o
Z_p^A, Z_p^B a fc / 100
SISTEMA
80X86
INT. CONTR.
DMA CONTR.
Stringa di Classif.
$C_4, C_1, C_3, C_2 \ldots C_4, \ldots$

fig 1

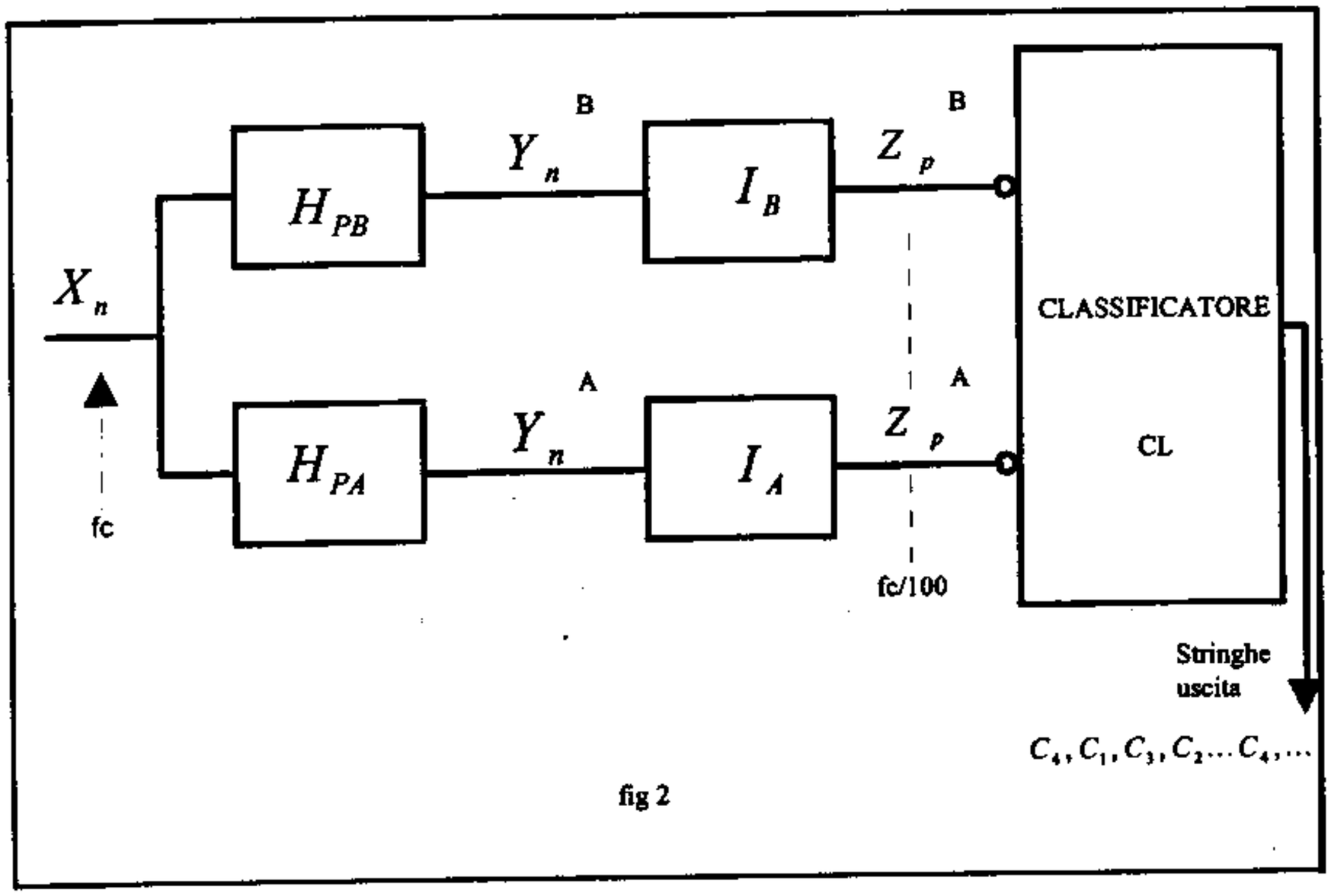

X_n
fc
H_{PB}
H_{PA}
Y_n^B
Y_n^A
I_B
I_A
Z_p^B
Z_p^A
fc/100
CLASSIFICATORE
CL
Stringhe uscita
$C_4, C_1, C_3, C_2 \ldots C_4, \ldots$

fig 2

ALLEGATO 1

C Compiler

```
g21 sourcefile [sourcefile ...]  [-switch ...]
or g21 @file_all  [-switch ...]
```

| source files listed in file_all |

Source (Input) File Filename Extension	File Contents and Translation Operation	Output File Filename Extension
.c	C source to be preprocessed	.i
.h	Macro or header file to be preprocessed	.i
.i	C source to be compiled	.s
.asm or .dsp or .s	Assembler source to be preprocessed	.is
.is	Assembler source to be assembled	.obj
.obj	Object file to be linked	.exe
.a	Library file of function definitions	–

Switches

-a *archfile*	.ACH architecture file read by compiler
-c	Compile (and/or assemble) only, without linking
-Dmacro[=defn]	Define macro
-g	Generate debugging information for simulator's C source debugger (CBUG)
-Idirectory	Include search directory (append *directory* to search path for include files)
-Ldirectory	Library search directory (linker appends *directory* to search path for library files)
-llibrary_name	Link library file (linker searches library file for functions)
-map	Direct linker to generate .map memory map file of symbols
-o *filename*	Select filename for final output files
-runhdr *filename*	Direct linker to use the specified runtime header file *(default: 2105_hdr.obj)*
-save-temps	Save temp files (.i, .s, .is)
-S	Stop after compiling, without assembling (.s file generated)
-Umacro	Undefine macro
-v	Verbose—compiler prints commands issued to execute each stage
-w	Warnings off—compiler inhibits all warning messages
-W	Warnings extra—compiler issues extra warning messages
-Wall	Warnings all—compiler issues all recommended warning messages

(Note: For a complete list of switch options, refer to the ADSP-2100 Family C Tools Manual)

PROM Splitter

```
spl21 exe_file promfile -pm [-switch ...]
or spl21 exe_file promfile -dm [-switch ...]
or spl21 exe_file promfile -bm [-switch ...]
or spl21 exe_file promfile -loader [-s] [-i]
```

Switches

-pm	Extract program memory ROM information
-dm	Extract data memory ROM information
-bm	Extract boot memory ROM information
-s	PROM file format: Motorola S record *(default)*
-i	PROM file format: Intel Hex record
-us	PROM file format: Motorola S record byte stream *(do not use with –bm)*
-us2	PROM file format: Motorola S2 record byte stream *(do not use with –bm)*
-ui	PROM file format: Intel Hex record byte stream *(do not use with –bm)*
-bs *pagesize*	Select boot page size *(default=2K)*
-bb *boundary*	Select boot page boundaries *(default=2K)*
-loader	Generate boot code with memory loading routines

HIP Splitter

```
hspl21 exe_file  [start_addr] [-i] [-boot]
```

Switches

-i	PROM file format: Intel Hex record *(default: –s)*
-boot	Ordering of 24-bit program memory words: High byte, Low byte, Middle byte *(default: High byte, Middle byte, Low byte)*

System Builder Directives

```
.SYSTEM system_name;
.ADSP21xx;
.MMAPx;      (x=0,1)
.CONST constant_name=expression;
.SEG/qualifier/qualifier/... seg_name(length);
.PORT/qualifier/qualifier port_name;
.ENDSYS;
```

21xx= 2100, 2101, 2105, 2111, 2150,
2101P (2101 paged memory system),
2101MV (2101 memory variant—use for 2115, 216x)

.SEG qualifiers: PM, DM, BOOT=0,1,2,3,4,5,6,7
RAM, ROM
ABS=*address*
DATA, CODE, DATA/CODE
EMULATOR, TARGET
INTERNAL *(for ADSP2101MV)*

.PORT qualifiers: PM, DM
ABS=*address*

Assembler Directives

```
.MODULE/qualifier/qualifier/... module_name;
.PAGE;
.CONST constant_name=expression;
.VAR/qualifier/qualifier/... buffer_name(length),...;
.INIT buffer_name: init_values;
.GLOBAL buffer_name,...;
.ENTRY program_label,...;
.EXTERNAL external_symbol,...;
.PORT port_name;
.INCLUDE <filename>;
.DMSEG dmseg_name;
.MACRO macro_name (param1,param2,...);
.ENDMACRO;
.LOCAL macro_label,...;
.NEWPAGE;                Insert pagebreak in .LST file
.PAGELENGTH #lines;      Insert pagebreaks every #lines in .LST file
.LEFTMARGIN #columns;    Set left margin in .LST file
.INDENT #columns;        Indent code #columns in .LST file
.PAGEWIDTH #columns;     Set right margin in .LST file
.ENDMOD;
```

.MODULE qualifiers: RAM, ROM
ABS=*address* *(do not use with STATIC)*
SBG=*seg_name*
BOOT=0,1,2,3,4,5,6,7
STATIC

.VAR qualifiers: PM, DM,
RAM, ROM
ABS=*address* *(do not use with STATIC)*
SBG=*seg_name*
CIRC
STATIC

.INIT init_values: constant, constant, ...
<*filename*>
^*other_buffer*
%*other_buffer*

Assembler C Preprocessor Directives

```
#define macro_name (param1,...) expression   Define a macro
#undef macro_name                            Undefine a macro
#include "filename"                          Include text from another source file

#if expression      Conditionally include and assemble, depending on the value of
                    an expression that evaluates to a constant
#else               Include and assemble if the previous #if test failed
#endif

#ifdef macro_name   Conditionally include and assemble, if macro_name is defined (with #define or –d switch)
#ifndef macro_name  Conditionally include and assemble, if macro_name is not defined
#else               Include and assemble if the previous #ifdef or #ifndef test failed
#endif
```

Instruction Set Summary

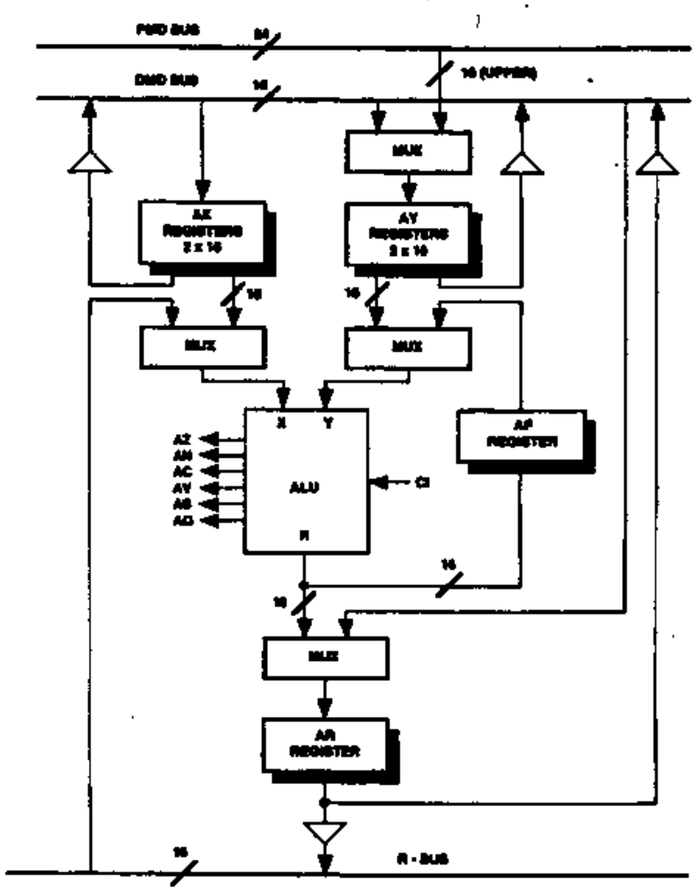
ALU Instructions

[IF cond] AR/AF = xop + yop ;		*Add/Add with Carry*
= xop + C ;		*Add/Add with Carry*
= xop + yop + C ;		*Add/Add with Carry*
= xop − yop ;		*Subtract X−Y/Subtract X−Y with Borrow*
= xop − yop + C − 1 ;		*Subtract X−Y/Subtract X−Y with Borrow*
= xop + C − 1 ;		*Subtract X–Y/Subtract X–Y with Borrow*
= yop − xop ;		*Subtract Y–X/Subtract Y–X with Borrow*
= yop − xop + C − 1 ;		*Subtract Y–X/Subtract Y–X with Borrow*
= −xop + C − 1 ;		*Subtract Y–X/Subtract Y–X with Borrow*
= xop AND yop ;		*AND, OR, XOR*
= xop OR yop ;		*AND, OR, XOR*
= xop XOR yop ;		*AND, OR, XOR*
= PASS xop ;		*Pass, Clear*
= PASS yop ;		*Pass, Clear*
= PASS −1 ;		*Pass, Clear*
= PASS 0 ;		*Pass, Clear*
= PASS 1 ;		*Pass, Clear*
= − xop ;		*Negate*
= − yop ;		*Negate*
= NOT xop ;		*NOT*
= NOT yop ;		*NOT*
= NOT 0 ;		*NOT*
= ABS xop ;		*Absolute Value*
= yop + 1 ;		*Increment*
= yop − 1 ;		*Decrement*
= DIVS yop, xop ;		*Divide*
= DIVQ xop ;		*Divide*

Notation Conventions

UPPERCASE	Explicit syntax—must be entered exactly as shown (either lowercase or uppercase may be used, however)
I0–I7	Index registers for indirect addressing
M0–M7	Modify registers for indirect addressing
L0–L7	Length registers for circular buffers (set to 0 for non-circular indirect addressing)
<data>	Immediate data value
<addr>	Immediate address value (absolute address or program label)
<exp>	Exponent (shift value) in shift immediate instructions (8-bit signed number)
cond	Condition code for conditional instruction
term	Termination code for DO UNTIL loop
dreg	Data register (of ALU, MAC, or Shifter)
reg	Any register (including dregs)
;	A semicolon terminates the instruction
,	Commas separate multiple operations of a single instruction
[ ]	Brackets enclose optional parts of instruction
[, ...]	Indicates multiple operations of an instruction that may be combined in any order, separated by commas.
option1 / option2 / option3	List of options; choose one.

ALU Block Diagram



IF Condition Codes

Cond	
EQ	Equal zero
NE	Not equal zero
LT	Less than zero
GE	Greater than or equal zero
LE	Less than or equal zero
GT	Greater than zero
AC	ALU carry
NOT AC	Not ALU carry
AV	ALU overflow
NOT AV	Not ALU overflow
MV	MAC overflow
NOT MV	Not MAC overflow
NEG	Xop input sign negative
POS	Xop input sign positive
NOT CE	Not counter expired
FLAG_IN *	FI pin=1
NOT FLAG_IN *	FI pin=0

* *Only for JUMP, CALL*

Allowed XOP, YOP Registers for ALU Instructions

xop	AX0, AX1 AR MR0, MR1, MR2 SR0, SR1
yop	AY0*, AY1 AF

* DIVS instruction may not use AY0 as YOP operand.

Instruction Set Summary

MAC Instructions

[IF cond]	MR/MF = xop * yop	(SS) (SU) (US) (UU) (RND) ;	*Multiply*
	= MR + xop * yop	(SS) (SU) (US) (UU) (RND) ;	*Multiply/Accumulate*
	= MR − xop * yop	(SS) (SU) (US) (UU) (RND) ;	*Multiply/Subtract*
	= MR [(RND)];		*Transfer MR*
	= 0;		*Clear*
IF MV SAT MR;			*Conditional MR Saturation*

(S) Signed input (xop, yop)
(U) Unsigned input (xop, yop)
(RND) Rounded output

IF Condition Codes

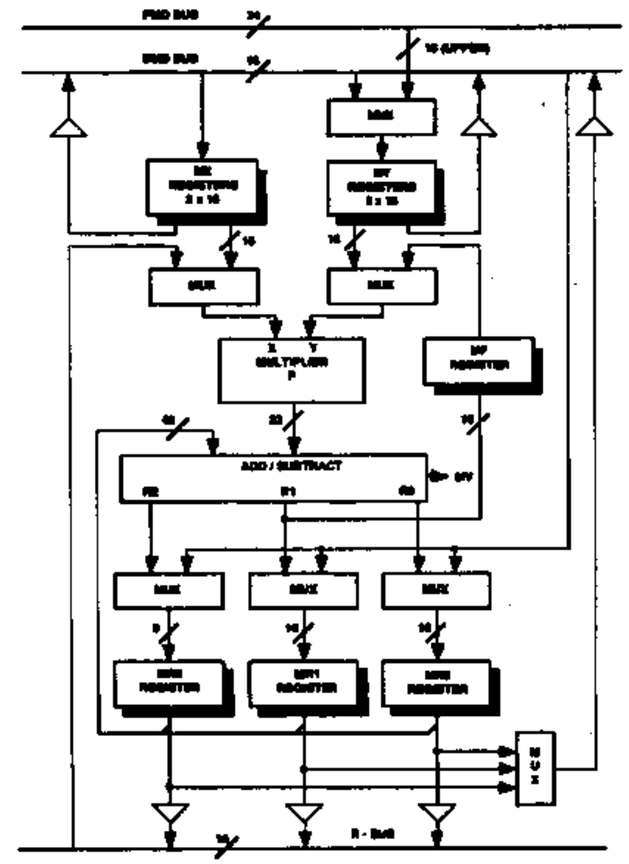
Cond	
EQ	Equal zero
NE	Not equal zero
LT	Less than zero
GE	Greater than or equal zero
LE	Less than or equal zero
GT	Greater than zero
AC	ALU carry
NOT AC	Not ALU carry
AV	ALU overflow
NOT AV	Not ALU overflow
MV	MAC overflow
NOT MV	Not MAC overflow
NEG	Xop input sign negative
POS	Xop input sign positive
NOT CE	Not counter expired
FLAG_IN *	FI pin=1
NOT FLAG_IN *	FI pin=0

** Only for JUMP, CALL*

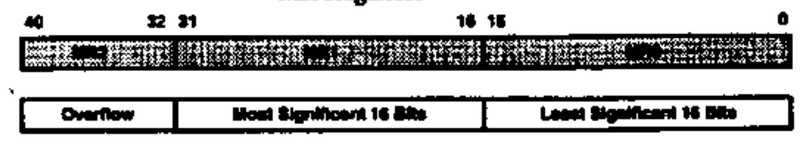
Allowed XOP, YOP Registers for MAC Instructions

xop	MX0, MX1 MR0, MR1, MR2 AR SR0, SR1
yop	MY0, MY1 MF

MAC Block Diagram



MR Registers



Instruction Set Summary

Shifter Instructions

[IF cond] SR = [SR OR] ASHIFT xop	(HI) / (LO) ;	*Arithmetic Shift*
[IF cond] SR = [SR OR] LSHIFT xop	(HI) / (LO) ;	*Logical Shift*
[IF cond] SR = [SR OR] NORM xop	(HI) / (LO) ;	*Normalize*
[IF cond] SE = EXP xop	(HI) / (LO) / (HIX) ;	*Derive Exponent*
[IF cond] SB = EXPADJ xop ;		*Block Exponent Adjust*
SR = [SR OR] ASHIFT xop BY <exp>	(HI) / (LO) ;	*Arithmetic Shift Immediate*
SR = [SR OR] LSHIFT xop BY <exp>	(HI) / (LO) ;	*Logical Shift Immediate*

(HI) Shift is referenced to SR1 (most significant 16 bits)
(LO) Shift is referenced to SR0 (least significant 16 bits)
(HIX) HI extend (AV overflow bit read by exponent detector)

IF Condition Codes

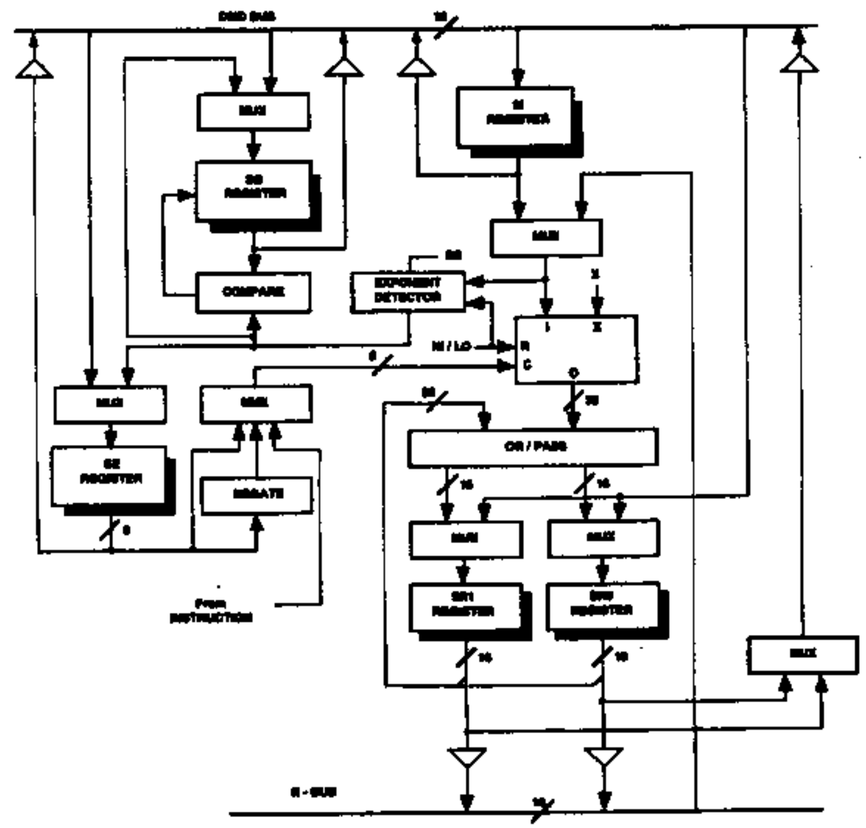
Cond	
EQ	Equal zero
NE	Not equal zero
LT	Less than zero
GE	Greater than or equal zero
LE	Less than or equal zero
GT	Greater than zero
AC	ALU carry
NOT AC	Not ALU carry
AV	ALU overflow
NOT AV	Not ALU overflow
MV	MAC overflow
NOT MV	Not MAC overflow
NEG	Xop input sign negative
POS	Xop input sign positive
NOT CE	Not counter expired
FLAG_IN *	FI pin=1
NOT FLAG_IN *	FI pin=0

** Only for JUMP, CALL*

Allowed XOP Registers for Shifter Instructions

xop	SI, SR0, SR1 AR MR0, MR1, MR2

Shifter Block Diagram



Instruction Set Summary

Data Move Instructions

```
reg  = reg;                                  Register-to-Register Move

reg  = <data>;                               Load Register Immediate
dreg = <data>;

reg  = DM (<addr>);                          Data Memory Read (Direct Address)

dreg = DM ( I0 , M0 );                       Data Memory Read (Indirect Address)
           I1   M1
           I2   M2
           I3   M3
           ========
           I4   M4
           I5   M5
           I6   M6
           I7   M7

dreg = PM ( I4 , M4 );                       Program Memory Read (Indirect Address)
           I5   M5
           I6   M6
           I7   M7

DM (<addr>) = reg;                           Data Memory Write (Direct Address)

DM ( I0 , M0 ) = dreg;                       Data Memory Write (Indirect Address)
     I1   M1
     I2   M2
     I3   M3
     ========
     I4   M4
     I5   M5
     I6   M6
     I7   M7

PM ( I4 , M4 ) = dreg;                       Program Memory Write (Indirect Address)
     I5   M5
     I6   M6
     I7   M7
```

Allowed Registers for Data Move & Multifunction Instructions

- **dreg** (data registers): AX0, AX1; AY0, AY1; AR; MX0, MX1; MY0, MY1; MR0, MR1, MR2; SI, SE, SR0, SR1
- **reg**: I0, I1, I2, I3, I4, I5, I6, I7; M0, M1, M2, M3, M4, M5, M6, M7; L0, L1, L2, L3, L4, L5, L6, L7; TX0, TX1, RX0, RX1; SB, PX; ASTAT, MSTAT; SSTAT *(read-only)*; IMASK, ICNTL; IFC *(write-only)*; CNTR; OWRCNTR *(write-only)*

Multifunction Instructions

```
<ALU>
<MAC>  ,  dreg = dreg;                       Computation with Register-to-Register Move
<SHIFT>

<ALU>
<MAC>  ,  dreg = DM ( I0 , M0 ) ;            Computation with Memory Read
<SHIFT>               I1   M1
                      I2   M2
                      I3   M3
                      ========
                      I4   M4
                      I5   M5
                      I6   M6
                      I7   M7

                 PM ( I4 , M4 ) ;
                      I5   M5
                      I6   M6
                      I7   M7

DM ( I0 , M0 ) = dreg ,  <ALU>  ;            Computation with Memory Write
     I1   M1                <MAC>
     I2   M2                <SHIFT>
     I3   M3
     ========
     I4   M4
     I5   M5
     I6   M6
     I7   M7

PM ( I4 , M4 )
     I5   M5
     I6   M6
     I7   M7

AX0 = DM ( I0 , M0 )  ,  AY0 = PM ( I4 , M4 ) ;   Data & Program Memory Read
AX1        I1   M1               AY1        I5   M5
MX0        I2   M2               MY0        I6   M6
MX1        I3   M3               MY1        I7   M7

<ALU>  ,  AX0 = DM ( I0 , M0 )  ,  AY0 = PM ( I4 , M4 ) ;   ALU/MAC with Data & Program
<MAC>     AX1      I1   M1          AY1      I5   M5        Memory Read
          MX0      I2   M2          MY0      I6   M6
          MX1      I3   M3          MY1      I7   M7
```

<ALU>*† Any ALU instruction (except DIVS, DIVQ)
<MAC>*† Any multiply/accumulate instruction
<SHIFT>* Any shifter instruction (except Shift Immediate)

* *May not be conditional instructions*
† *AR, MR result registers must be used—not AF, MF feedback registers*

Instruction Set Summary

Program Flow Instructions

DO <addr> [UNTIL term];	*Do Until*
[IF cond] JUMP | (I4) (I5) (I6) (I7) <addr> |;	*Jump*
[IF cond] CALL | (I4) (I5) (I6) (I7) <addr> |;	*Call Subroutine*
IF | FLAG_IN NOT FLAG_IN | | JUMP CALL | <addr>;	*Jump/Call on Flag In Pin*
[IF cond] | SET RESET TOGGLE | | FLAG_OUT FL0 FL1 FL2 | [, ...];	*Modify Flag Out Pin*
[IF cond] RTS;	*Return from Subroutine*
[IF cond] RTI;	*Return from Interrupt Service Routine*
IDLE [(n)]; *n=16, 32, 64, or 128*	*Idle*

DO UNTIL Termination Codes

Term	
CE	Counter expired
EQ	Equal zero
NE	Not equal zero
LT	Less than zero
GE	Greater than or equal zero
LE	Less than or equal zero
GT	Greater than zero
AC	ALU carry
NOT AC	Not ALU carry
AV	ALU overflow
NOT AV	Not ALU overflow
MV	MAC overflow
NOT MV	Not MAC overflow
NEG	Xop input sign negative
POS	Xop input sign positive
FOREVER	Always

IF Condition Codes

Cond	
EQ	Equal zero
NE	Not equal zero
LT	Less than zero
GE	Greater than or equal zero
LE	Less than or equal zero
GT	Greater than zero
AC	ALU carry
NOT AC	Not ALU carry
AV	ALU overflow
NOT AV	Not ALU overflow
MV	MAC overflow
NOT MV	Not MAC overflow
NEG	Xop input sign negative
POS	Xop input sign positive
NOT CE	Not counter expired
FLAG_IN *	FI pin=1
NOT FLAG_IN *	FI pin=0

** Only for JUMP, CALL*

Miscellaneous Instructions

NOP;	*NOP*
MODIFY (| I0 I1 I2 I3 I4 I5 I6 I7 | , | M0 M1 M2 M3 M4 M5 M6 M7 |);	*Modify Address Register*
| PUSH POP | STS] [, POP CNTR] [, POP PC] [, POP LOOP];	*Stack Control*
| ENA DIS | | SEC_REG BIT_REV AV_LATCH AR_SAT M_MODE TIMER G_MODE | [, ...];	*Mode Control*

Modes	
SEC_REG	Secondary register set
BIT_REV	Bit-reverse addressing in DAG1
AV_LATCH	ALU overflow (AV) status latch
AR_SAT	AR register saturation
M_MODE	MAC result placement mode
TIMER	Timer enable
G_MODE	Go mode enable

Circular Buffer Addressing

Next Address = (I + M − B) modulo(L) + B

I=current address M=modify value (signed) M≤L
B=base address L=buffer length

(Set L=0 for standard, non-circular indirect addressing: I+M=modified address)

Buffer Length & Base Address Operators

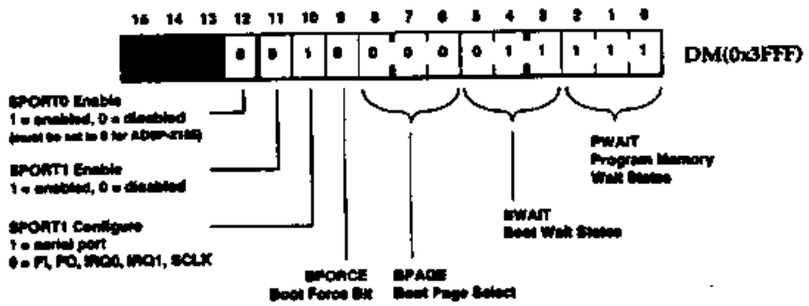
^buffer_name	*Base address of buffer_name*
%buffer_name	*Length (number of locations) of buffer_name*

Example: Setting Up DAG Registers for Circular Buffer & DO UNTIL Loop

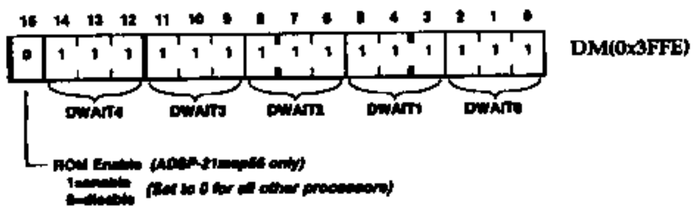
```
        .VAR/DM/RAM/CIRC real_data[n];     {n=number of input samples}
         I5=^real_data;                    {buffer base address}
         L5=%real_data;                    {buffer length}
         M5=1;                             {post-modify I5 by 1}
         CNTR=%real_data;                  {loop counter = buffer length}
         DO loop UNTIL CE;
            AX0=DM(I5,M5);                 {get next sample}
            ...
            {now process sample stored in AX0}
loop:       ...
```

Control/Status Registers

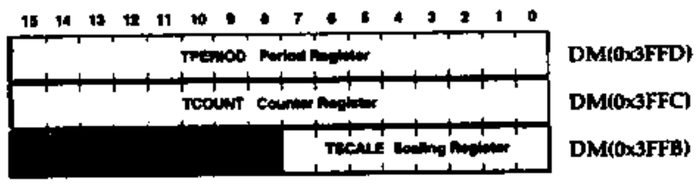
System Control Register



Data Memory Waitstate Register



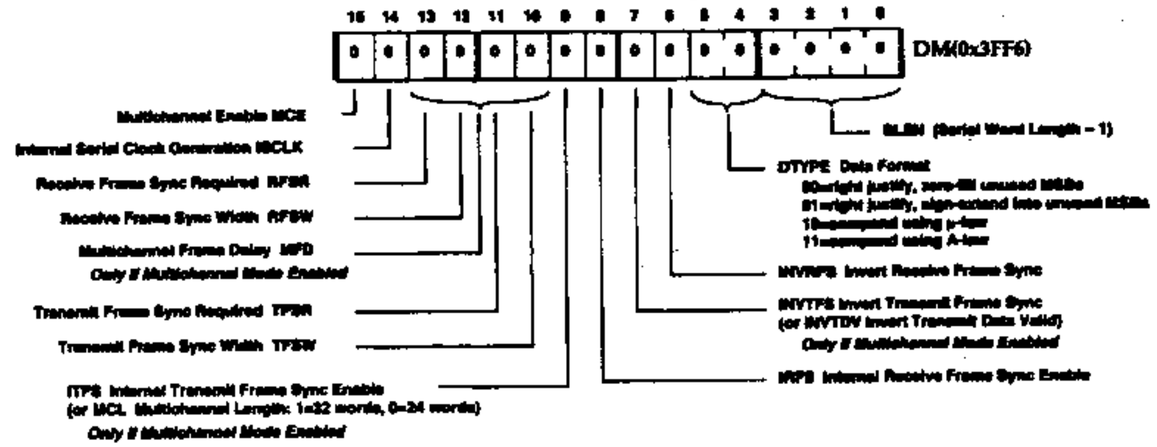
Timer Registers



Control register default bit values at reset are as shown; if no value is shown, the bit is undefined after reset. Reserved bits are shown on a gray field—these bits should always be written with zeros.

SPORT0 Control Register

Not on ADSP-2105



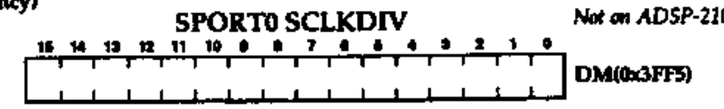
SPORT0 Autobuffer Control

Not on ADSP-2105

$$\text{SCLKDIV} = \frac{\text{CLKOUT frequency}}{2 \cdot (\text{SCLK frequency})} - 1$$

SPORT0 SCLKDIV

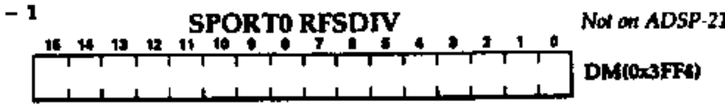
Not on ADSP-2105



$$\text{RFSDIV} = \frac{\text{SCLK frequency}}{\text{RFS frequency}} - 1$$

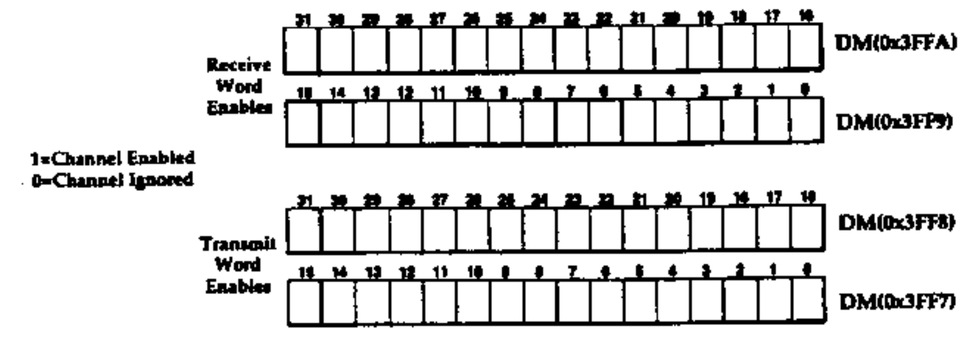
SPORT0 RFSDIV

Not on ADSP-2105

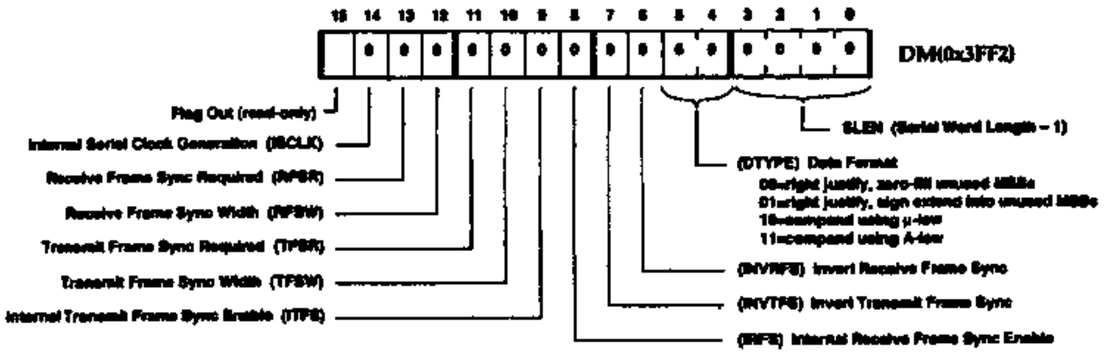


SPORT0 Multichannel Word Enables

Not on ADSP-2105

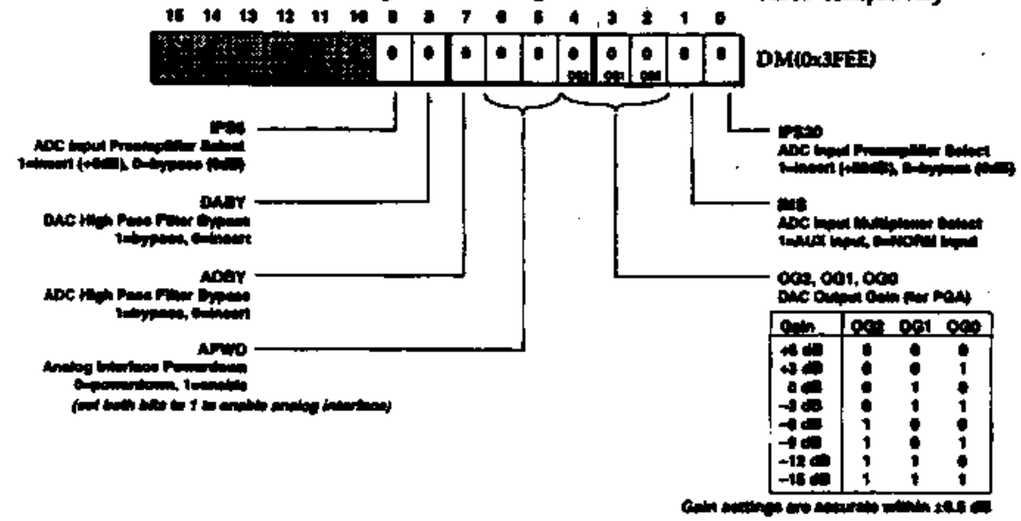


SPORT1 Control Register



Analog Control Register

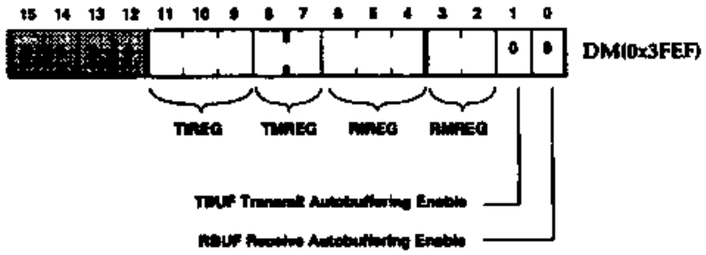
ADSP-21msp50 only



Gain	OG2	OG1	OG0
+6 dB	0	0	0
+3 dB	0	0	1
0 dB	0	1	0
–3 dB	0	1	1
–6 dB	1	0	0
–9 dB	1	0	1
–12 dB	1	1	0
–15 dB	1	1	1

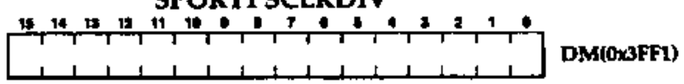
SPORT1 Autobuffer Control

Not on ADSP-21msp50



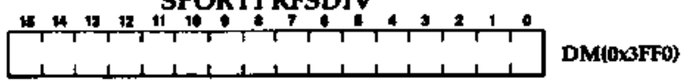
$$\text{SCLKDIV} = \frac{\text{CLKOUT frequency}}{2 \cdot (\text{SCLK frequency})} - 1$$

SPORT1 SCLKDIV



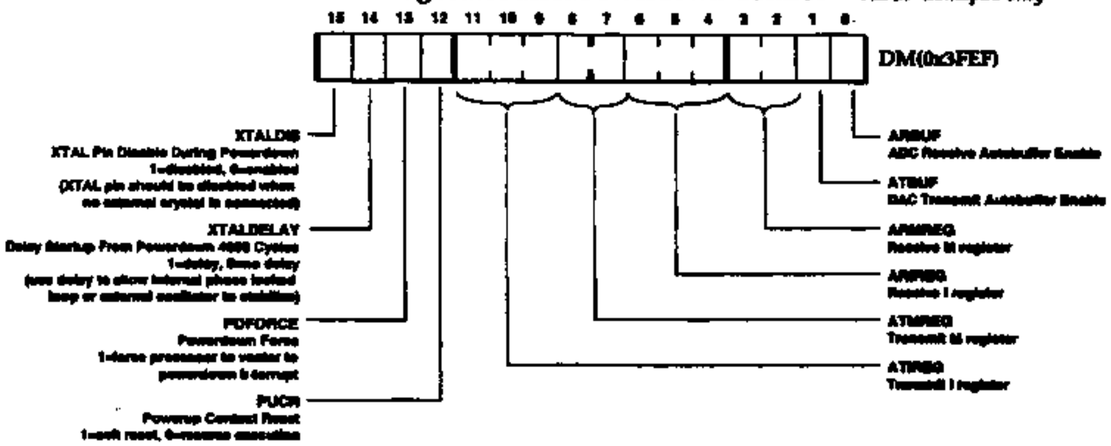
$$\text{RFSDIV} = \frac{\text{SCLK frequency}}{\text{RFS frequency}} - 1$$

SPORT1 RFSDIV



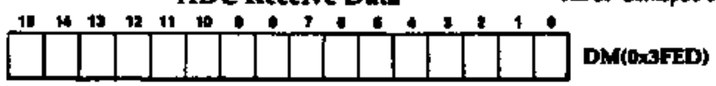
Analog Autobuffer/Powerdown Control

ADSP-21msp50 only



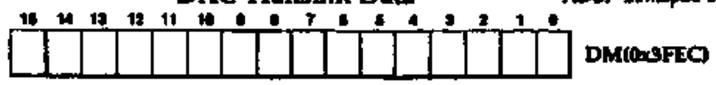
ADC Receive Data

ADSP-21msp50 only



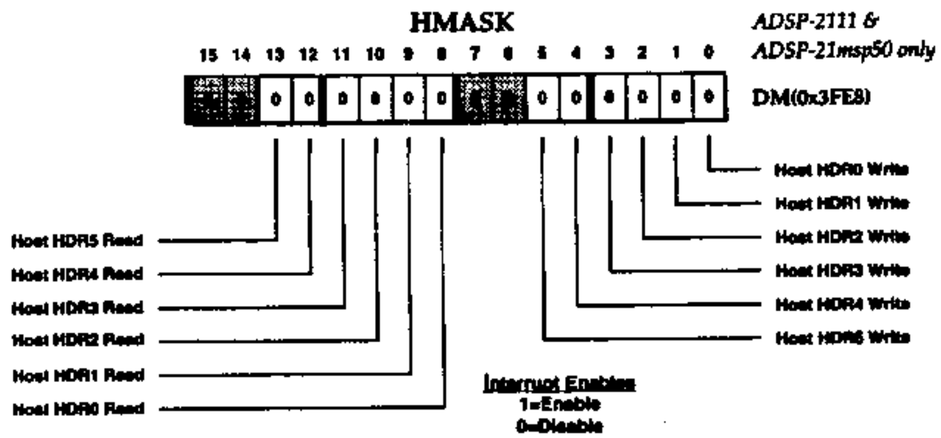
DAC Transmit Data

ADSP-21msp50 only

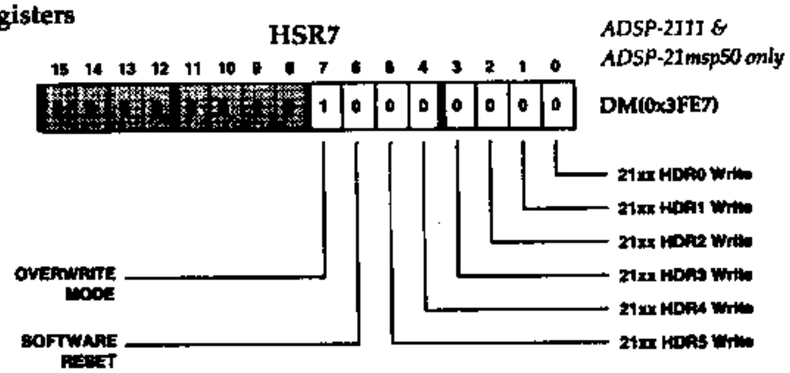


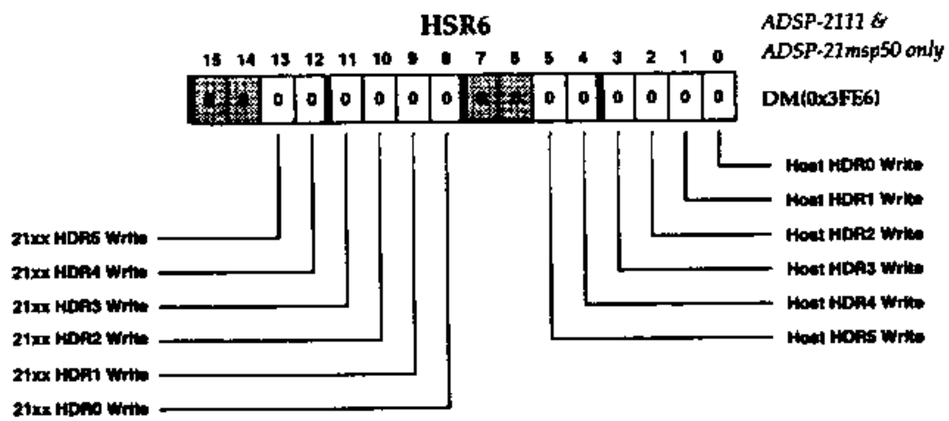
No registers are located between DM(0x3FEC) and DM(0x3FE8)

HIP Interrupt Mask Register

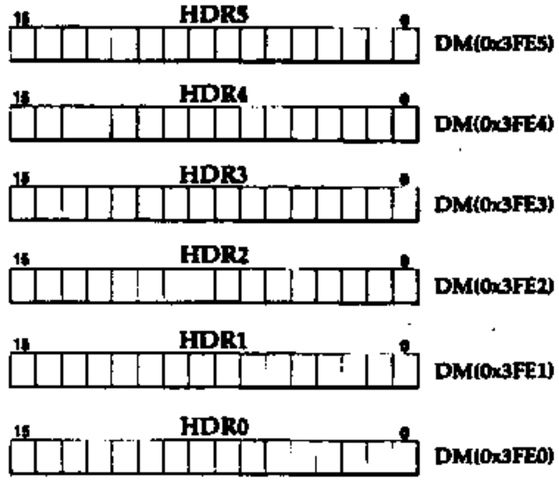


HIP Status Registers



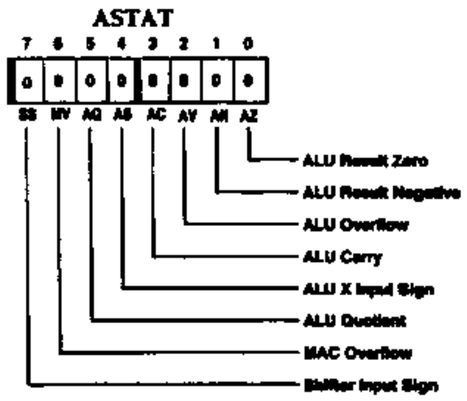


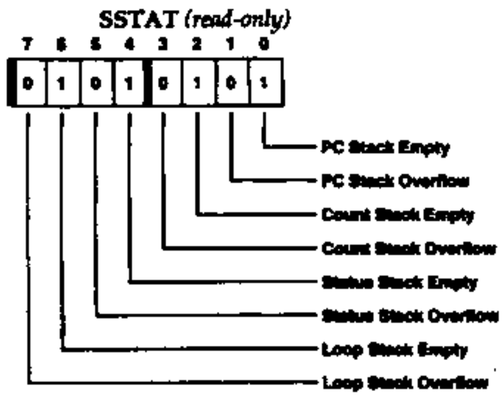
HIP Data Registers

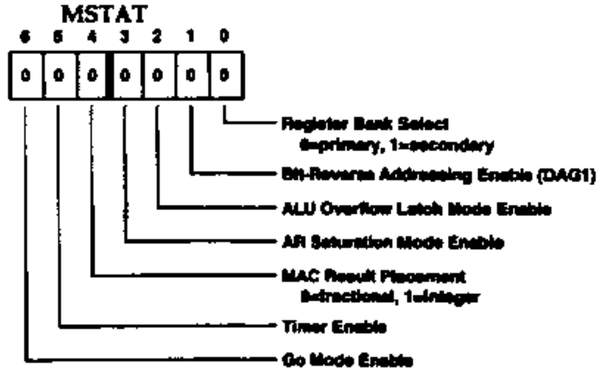


ADSP-2111 &
ADSP-21msp50 only

Status Registers
(Non-Memory-Mapped)



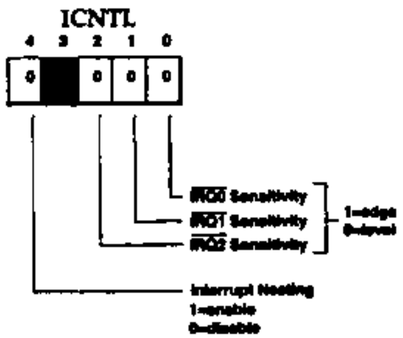


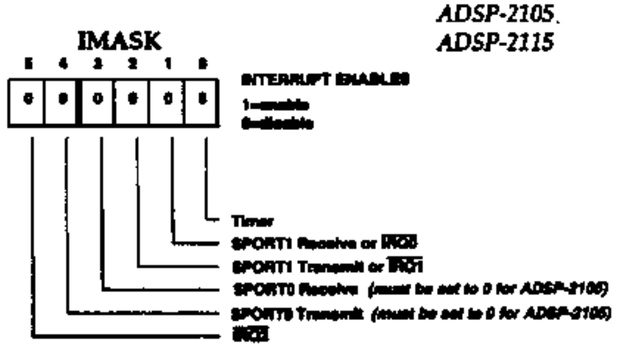


Mode Names for Mode Control Instruction
(see Miscellaneous Instructions on page 13)

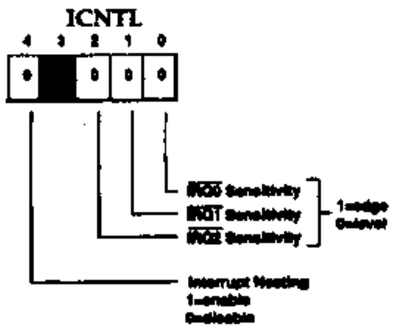
Bit	*Mode Name*	
0	SEC_REG	Secondary register set
1	BIT_REV	Bit-reverse addressing in DAG1
2	AV_LATCH	ALU overflow (AV) status latch
3	AR_SAT	AR register saturation
4	M_MODE	MAC result placement mode
5	TIMER	Timer enable
6	G_MODE	Go mode enable

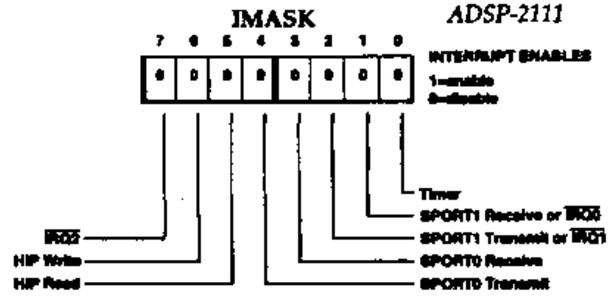
Interrupt Registers
(Non-Memory-Mapped)



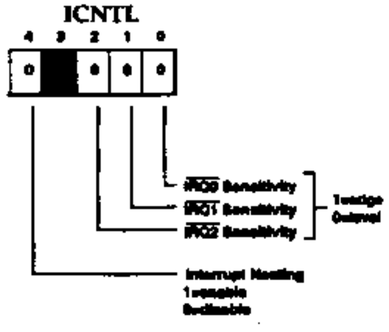


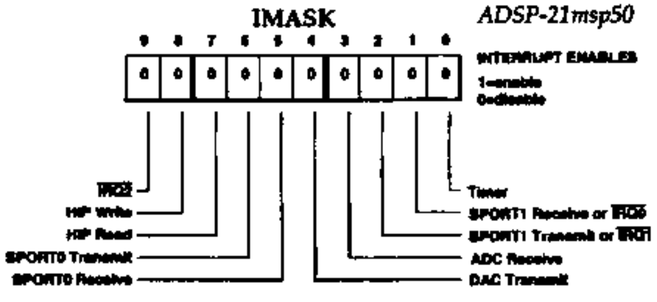
ADSP-2101
ADSP-2105
ADSP-2115



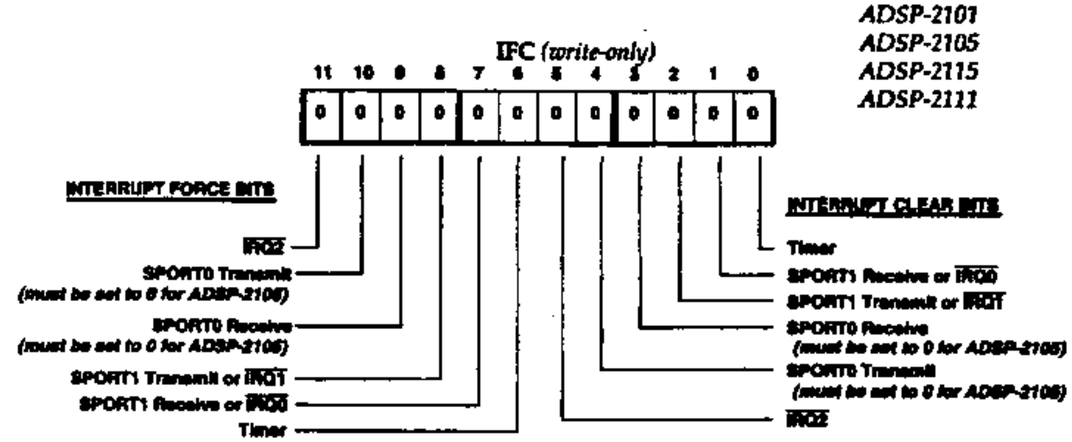


ADSP-2111

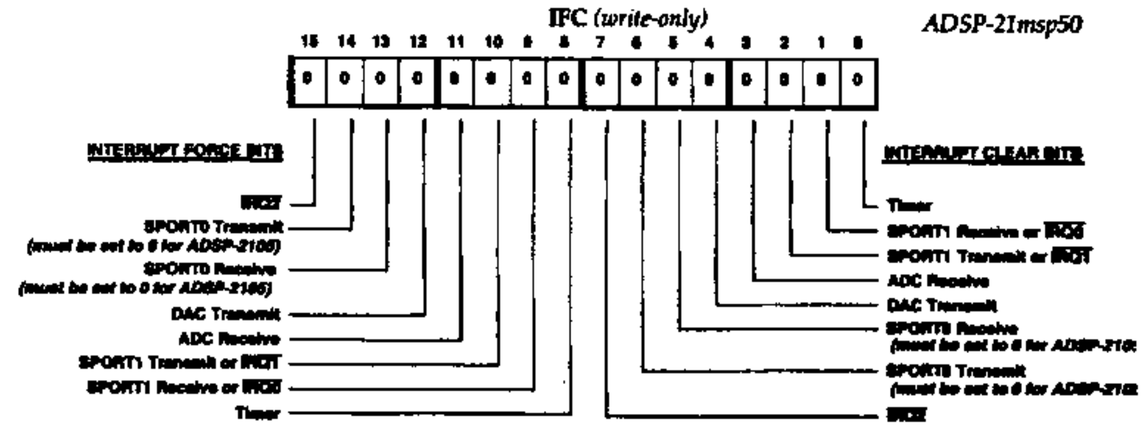




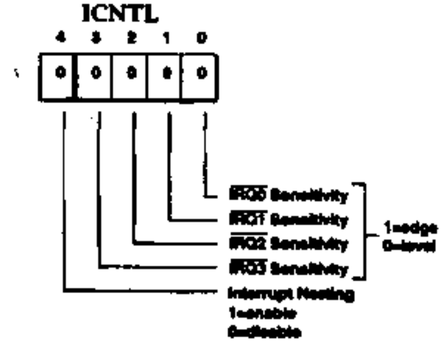
ADSP-21msp50

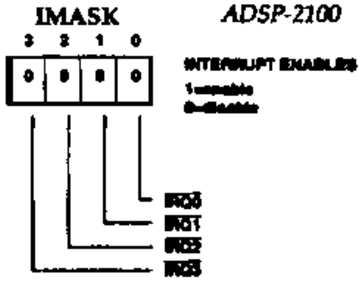


ADSP-2101
ADSP-2105
ADSP-2115
ADSP-2111



ADSP-21msp50

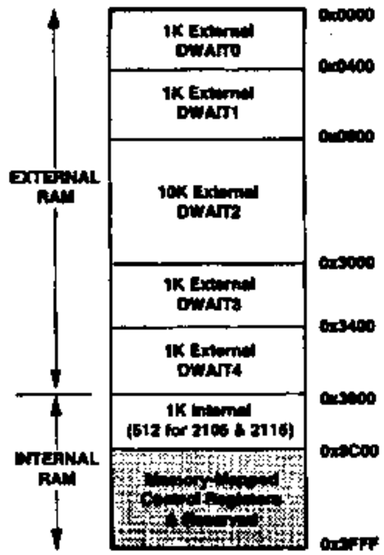




ADSP-2100

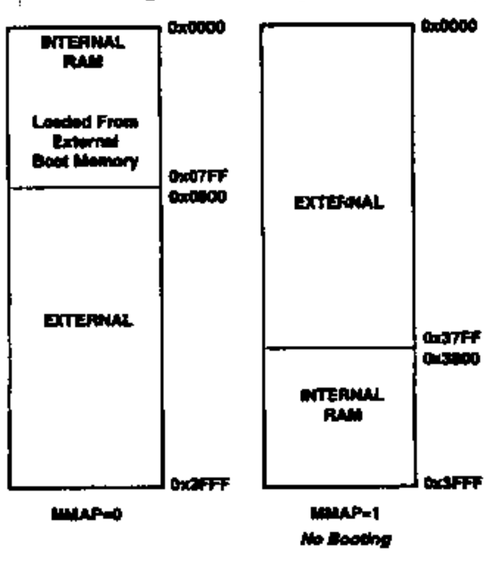
Memory Maps

Data Memory



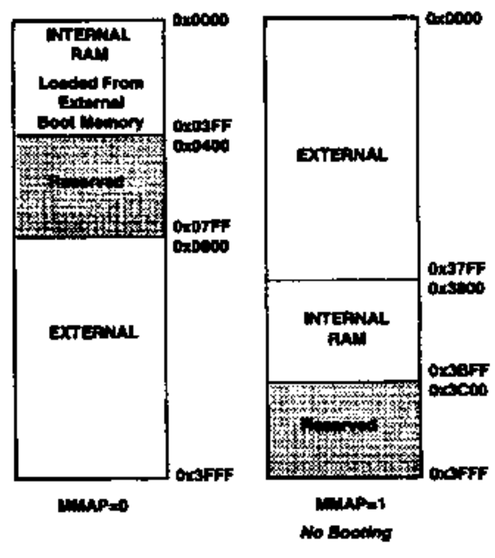
Program Memory

ADSP-2101
ADSP-2111
ADSP-21msp50



Program Memory

ADSP-2105
ADSP-2115



Interrupt Vector Tables

ADSP-2100

Interrupt Source	*Interrupt Vector Address*
IRQ0	0x0000
IRQ1	0x0001
IRQ2	0x0002
IRQ3	0x0003
RESET startup	0x0004

ADSP-2101/2115

Interrupt Source	*Interrupt Vector Address*
RESET startup	0x0000
IRQ2	0x0004 *(high priority)*
SPORT0 Transmit	0x0008
SPORT0 Receive	0x000C
SPORT1 Transmit/IRQ1	0x0010
SPORT1 Receive/IRQ0	0x0014
Timer	0x0018 *(low priority)*

ADSP-2105

Interrupt Source	*Interrupt Vector Address*
RESET startup	0x0000
IRQ2	0x0004 *(high priority)*
SPORT1 Transmit/IRQ1	0x0010
SPORT1 Receive/IRQ0	0x0014
Timer	0x0018 *(low priority)*

ADSP-2111

Interrupt Source	*Interrupt Vector Address*
RESET startup	0x0000
IRQ2	0x0004 *(high priority)*
HIP Write from Host	0x0008
HIP Read to Host	0x000C
SPORT0 Transmit	0x0010
SPORT0 Receive	0x0014
SPORT1 Transmit/IRQ1	0x0018
SPORT1 Receive/IRQ0	0x001C
Timer	0x0020 *(low priority)*

ADSP-21msp50

Interrupt Source	*Interrupt Vector Address*
RESET *(or powerup w/PUCR=1)*	0x0000
Powerdown *(non-maskable)*	0x002C *(high priority)*
IRQ2	0x0004
HIP Write from Host	0x0008
HIP Read to Host	0x000C
SPORT0 Transmit	0x0010
SPORT0 Receive	0x0014
Analog Transmit (DAC)	0x0018
Analog Receive (ADC)	0x001C
SPORT1 Transmit/IRQ1	0x0020
SPORT1 Receive/IRQ0	0x0024
Timer	0x0028 *(low priority)*

Control/Status Registers

Symbolic names for the memory-mapped control and status registers are provided in four files included with the development software. The symbols are defined as constants equal to the register addresses, and can be used for direct addressing. To use these symbols, include the appropriate file in the your source code files with the assembler's .INCLUDE directive:

Filename	*Include Directive To Use:*	
DEF2101.H	.INCLUDE <DEF2101.H>;	*(use also for 2115)*
DEF2105.H	.INCLUDE <DEF2105.H>;	
DEF2111.H	.INCLUDE <DEF2111.H>;	
DEF2150.H	.INCLUDE <DEF2150.H>;	

Control/Status Register	*Data Memory Address*	*Assembly Code Symbol*	
System Control Register	0x3FFF	Sys_Ctrl_Reg	
Data Memory Wait State Control Register	0x3FFE	Dm_Wait_Reg	
Timer Period	0x3FFD	Tperiod_Reg	
Timer Count	0x3FFC	Tcount_Reg	
Timer Scaling Factor	0x3FFB	Tscale_Reg	
SPORT0 Multichannel Receive Word Enable Register (32-bit)	0x3FFA	Sport0_Rx_Words1	*Not on 2105*
	0x3FF9	Sport0_Rx_Words0	*Not on 2105*
SPORT0 Multichannel Transmit Word Enable Register (32-bit)	0x3FF8	Sport0_Tx_Words1	*Not on 2105*
	0x3FF7	Sport0_Tx_Words0	*Not on 2105*
SPORT0 Control Register	0x3FF6	Sport0_Ctrl_Reg	*Not on 2105*
SPORT0 Serial Clock Divide Modulus	0x3FF5	Sport0_Sclkdiv	*Not on 2105*
SPORT0 Rcv Frame Sync Divide Modulus	0x3FF4	Sport0_Rfsdiv	*Not on 2105*
SPORT0 Autobuffer Control Register	0x3FF3	Sport0_Autobuf_Ctrl	*Not on 2105*
SPORT1 Control Register	0x3FF2	Sport1_Ctrl_Reg	
SPORT1 Serial Clock Divide Modulus	0x3FF1	Sport1_Sclkdiv	
SPORT1 Rcv Frame Sync Divide Modulus	0x3FF0	Sport1_Rfsdiv	
SPORT1 Autobuffer Control Register	0x3FEF	Sport1_Autobuf_Ctrl	*Not on 21msp50*
Analog Autobuffer/Powerdown Ctrl Reg	0x3FEF	Codec_Autobuf_Ctrl	*21msp50 only*
Analog Control Register	0x3FEE	Codec_Ctrl_Reg	*21msp50 only*
ADC Receive Data Register	0x3FED	Codec_Rx_Data	*21msp50 only*
DAC Transmit Data Register	0x3FEC	Codec_Tx_Data	*21msp50 only*
HIP Interrupt Mask Register	0x3FE8	Hmask_Reg	*2111, 21msp50 only*
HIP Status Register 7	0x3FE7	HSR7_Reg	*2111, 21msp50 only*
HIP Status Register 6	0x3FE6	HSR6_Reg	*2111, 21msp50 only*
HIP Data Register 5	0x3FE5	HSR5_Reg	*2111, 21msp50 only*
HIP Data Register 4	0x3FE4	HSR4_Reg	*2111, 21msp50 only*
HIP Data Register 3	0x3FE3	HSR3_Reg	*2111, 21msp50 only*
HIP Data Register 2	0x3FE2	HSR2_Reg	*2111, 21msp50 only*
HIP Data Register 1	0x3FE1	HSR1_Reg	*2111, 21msp50 only*
HIP Data Register 0	0x3FE0	HSR0_Reg	*2111, 21msp50 only*

	2100	2101	2105	2115	2111	21msp50/55/56
Data Memory (RAM)	–	1K	½K	½K	1K	1K
Program Memory (RAM)	–	2K	1K	1K	2K	2K
Timer	–	✔	✔	✔	✔	✔
Serial Port 0 (multichannel)	–	✔	–	✔	✔	✔
Serial Port 1	–	✔	✔	✔	✔	✔
Host Interface Port	–	–	–	–	✔	✔
Analog Interface	–	–	–	–	–	✔