INITIATION AUX DSP

DESS Microélectronique

Université Paul Sabatier

Toulouse

Nicolas Nolhier 02/2001

PLAN du COURS

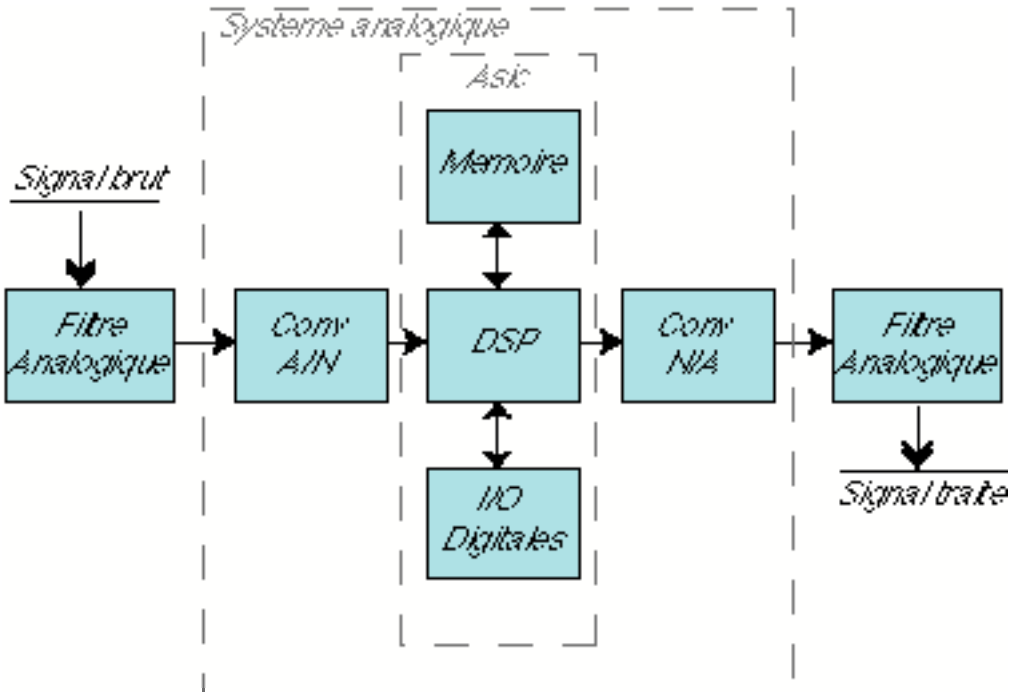
- Introduction
  - Spécificités d'un DSP
  - Formats des données
- Le DSP ADSP 21065L
  - Caractéristiques
  - Architecture
  - Unités de Calcul
  - Séquenceur
  - Générateurs d'adresse
  - Les interruptions
  - Le cache
  - Organisation de la mémoire
  - Le DMA
  - Le SPORT
  - Timer et I/O
- La Conversion Analogique ⇔ Digitale
  - Structures de CODEC
  - Communications avec le DSP
- Environnement de développement

Références

- Informations sur les DSP Analog Devices
  - http://www.analog.com/DSP/
- Informations sur les DSP Motorola
  - http://www.motorola.com/SPS/DSP/
- Cours de formation sur les DSP et le traitement numérique du signal
  - http://www.techonline.com/
- Introduction aux DSP
  - http://perso.wanadoo.fr/lapiste/dsp.htm

Le DSP

- DSP : *Digital Signal Processor*
- Cœur d'une chaîne de traitement numérique du signal

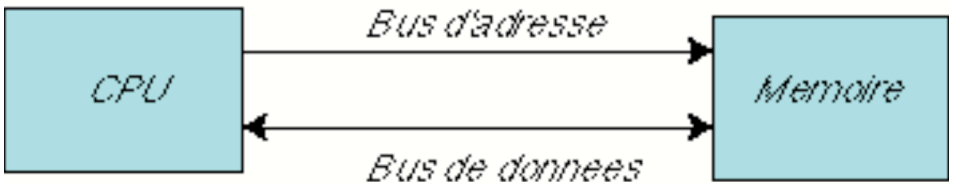


- Système numérique / système analogique :
  - Filtres numériques spécifiques
  - Stabilité / Reproductibilité
- DSP / Circuit Intégré dédié ASIC :
  - Reprogrammation
  - Algorithmes adaptatifs

Spécificités des DSP / Microprocesseur

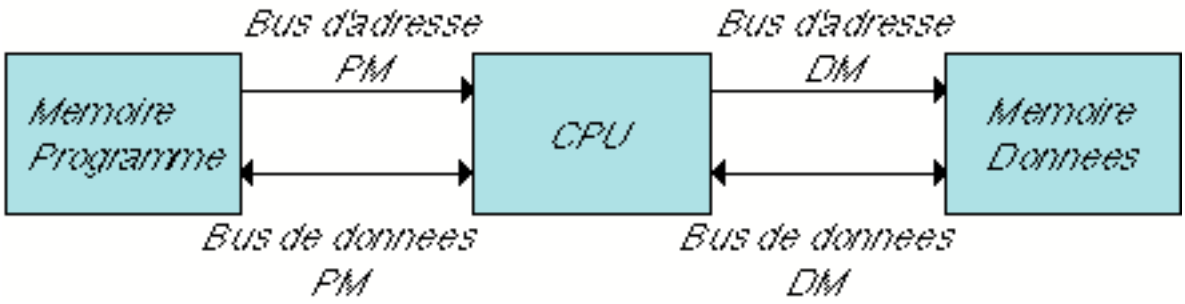
Architecture

- Cas d'un microprocesseur
  - Architecture Von Neuman



  - Mémoire globale

- Cas d'un DSP
  - Architecture Harvard



  - Mémoire séparée

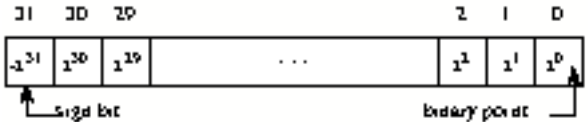
Spécificités des DSP / Microprocesseur

- Instructions adaptées et optimisées:
  - Ex le MAC : 1 cycle d'horloge

    A = B*C + D
- Modes d'adressage évolués
  - adressage circulaire post-incrémental
- RAM interne
- Gestion directe de la mémoire DMA
- Pipe-Line : gestion // des opérations
- "Multi-processing"
- Entrées / sorties numériques

Virgule fixe ou flottante ?

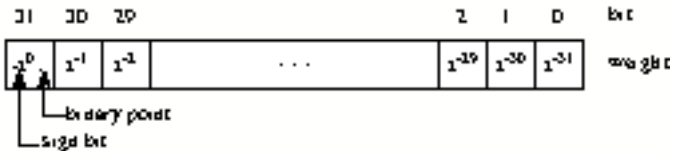
- DSP à virgule fixe
  - ex : sur 32 bits
    - entier



$$N = -b_{31}.2^{31} + \sum_{i=0}^{30} b_i 2^i$$

$$-2 \cdot 147 \cdot 483 \cdot 648 \leq N \leq 2 \cdot 147 \cdot 483 \cdot 647$$

    - fractionnaire

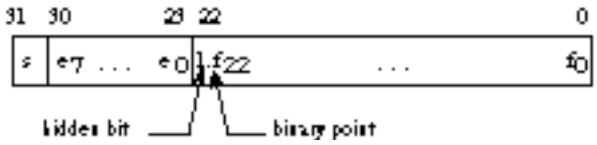


$$N = (-1)^{b_{31}} + \sum_{i=0}^{30} b_i 2^{i-31}$$

$$-1 \leq N \leq 0.999999977$$

  - Plus complexe à programmer - normalisation
  - Dynamique réduite
  - Coût plus faible

- DSP à virgule flottante (IEEE 754.1985)
  - ex : sur 32 bits



$$N = (-1)^S.(1, f).2^{e-127} \qquad e = \sum_{i=0}^{7} e_i 2^i \qquad f = \sum_{i=0}^{22} f_i 2^{i-23}$$

  - exemples :

0 00000111 11000000000000000000000 + 7 0.75	$+1.75 \times 2^{(7-127)} = +1.316554 \times 10^{-36}$

1 10000001 01100000000000000000000 - 129 0.375	$-1.375 \times 2^{(129-127)} = -5.500000$

  - Souplesse
  - Grande dynamique
  - Structure complexe - Prix élevé

Le DSP ADSP-21065L

- DSP 32 bits de chez ANALOG DEVICES
- Virgule flottante / fixe
- 180 MFLOPS
- Horloge interne 60 MHz - Un cycle par instruction !
- 544 kbits de SRAM intégrée
- 2 ports série haut débit (30MBits/s)
  - 32 canaux TDM
- 2 Timers (capture / PWM)
- 12 lignes In/Out digitales
- 10 canaux DMA
- 3.3 V
- $10 - pour 100.000 pièces !

Exemple d'application

- **<u>Processeur audio numérique AV32R</u>**

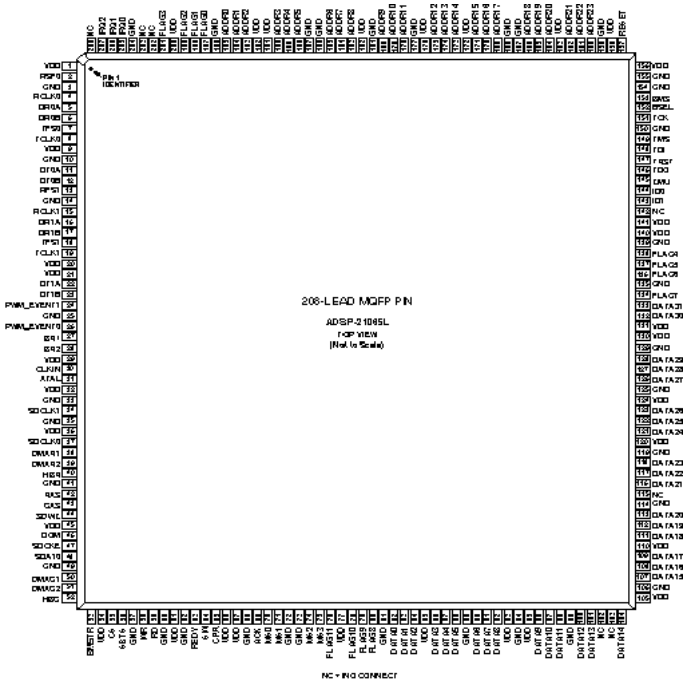
Décodage numérique pour Home Cinéma

Formats supportés :

- Dolby Digital™
- DTS™
- MPEG2™
- Dolby Pro Logic™
- TAG McLaren Surround
- Processing Format THX Cinema™

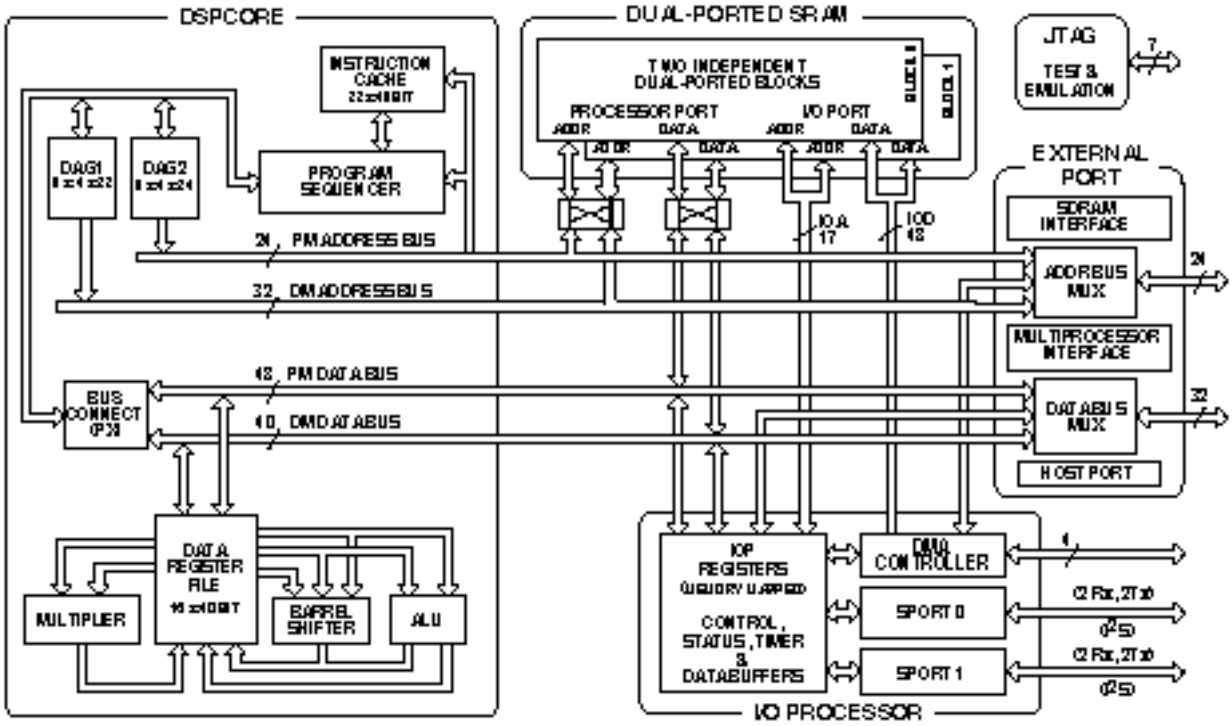
Brochage

- Boîtier 208 broches



Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
1	VDD	43	CAS	85	VDD	127	DATA26	169	ADDR17
2	RFS0	44	SDWE	86	DATA3	128	DATA29	170	ADDR16
3	GND	45	VDD	87	DATA4	129	GND	171	ADDR15
4	RCLK0	46	DQM	88	DATA5	130	VDD	172	VDD
5	DR0A	47	SDCKE	89	GND	131	VDD	173	ADDR14
6	DR0B	48	SDA10	90	DATA6	132	DATA30	174	ADDR13
7	TFS0	49	GND	91	DATA7	133	DATA31	175	ADDR12
8	TCLK0	50	DMAG 1	92	DATA8	134	FLAG7	176	VDD
9	VDD	51	DMAG 2	93	VDD	135	GND	177	GND
10	GND	52	HBG	94	GND	136	FLAG6	178	ADDR11
11	DT0A	53	BMSTR	95	VDD	137	FLAG5	179	ADDR10
12	DT0B	54	VDD	96	DATA9	138	FLAG4	180	ADDR9
13	RFS1	55	CS	97	DATA10	139	GND	181	GND
14	GND	56	SBTS	98	DATA11	140	VDD	182	VDD
15	RCLK1	57	GND	99	GND	141	VDD	183	ADDR8
16	DR1A	58	WR	100	DATA12	142	NC	184	ADDR7
17	DR1B	59	RD	101	DATA13	143	ID1	185	ADDR6
18	TFS1	60	GND	102	NC	144	ID0	186	GND
19	TCLK1	61	VDD	103	NC	145	EMU	187	GND
20	VDD	62	GND	104	DATA14	146	TDO	188	ADDR5
21	VDD	63	REDY	105	VDD	147	TRST	189	ADDR4
22	DT1A	64	SW	106	GND	148	TDI	190	ADDR3
23	DT1B	65	CPA	107	DATA15	149	TMS	191	VDD
24	PWM_EVENT1	66	VDD	108	DATA16	150	GND	192	VDD
25	GND	67	VDD	109	DATA17	151	TCK	193	ADDR2
26	PWM_EVENT0	68	GND	110	VDD	152	BSEL	194	ADDR1
27	BR1	69	ACK	111	DATA18	153	BMS	195	ADDR0
28	BR2	70	MS0	112	DATA19	154	GND	196	GND
29	VDD	71	MS1	113	DATA20	155	GND	197	FLAG0
30	CLKIN	72	GND	114	GND	156	VDD	198	FLAG1
31	XTAL	73	GND	115	NC	157	RESET	199	FLAG2
32	VDD	74	MS2	116	DATA21	158	VDD	200	VDD
33	GND	75	MS3	117	DATA22	159	GND	201	FLAG3
34	SDCLK1	76	FLAG11	118	DATA23	160	ADDR23	202	NC
35	GND	77	VDD	119	GND	161	ADDR22	203	NC
36	VDD	78	FLAG10	120	VDD	162	ADDR21	204	GND
37	SDCLK0	79	FLAG9	121	DATA24	163	VDD	205	IRQ0
38	DMAR1	80	FLAG8	122	DATA25	164	ADDR20	206	IRQ1
39	DMAR2	81	GND	123	DATA26	165	ADDR19	207	IRQ2
40	HBR	82	DATA0	124	VDD	166	ADDR18	208	NC
41	GND	83	DATA1	125	GND	167	GND		
42	RAS	84	DATA2	126	DATA27	168	GND		

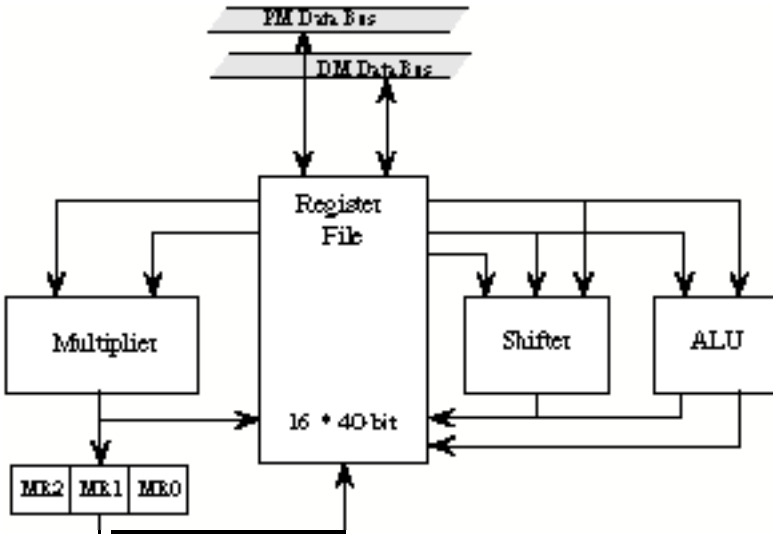
Schéma Bloc général



Les unités de calcul

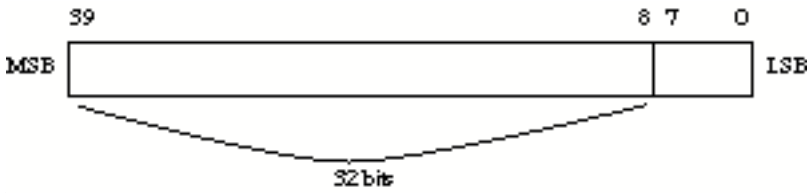
- 3 unités :
  - ALU
  - Multiplieur
  - Un registre à décalage

- Chaque unité : une opération par cycle



Le bloc des registres

- *Register File* : interface entre les bus de données et les unités de calcul
- Transfert de données et résultat de calcul
- 16 registres indépendants de 40 bits
  - si on travaille sur 32 bits b0 - b7 ignorés



- en assembleur :
  - Fx : registre x pour une opération en flottant
  - Rx : registre x pour une opération en fixe

  ex :

```
F0 = F1 * F2 ;
R0 = R1 * R2 ;
```

```
F15 = 12.254 ;
R15 = 13 ;
```

Unité arithmétique et Logique
ALU

- Opérations sur des nombres à virgule fixe et flottante.
- Addition/ soustraction/ moyenne
- Opérateurs logiques (fixes)
- Conversions
- Fonctions diverses

  – exemple d'instructions portant sur des flottants :

```
Fn = Fx + Fy ;
Fn = Fx - FY ;
Fn = ABS Fx ;
COMP(Fx,Fy) ;
Fn = (Fx + Fy)/2 ;
Fn = MAX (Fx,Fy) ;
Fn = MIN (Fx, Fy) ;
Fn = FLOAT Rx ;
Rn = FIX Fx ;
```

- Chaque instruction modifie des drapeaux dans les registres d'état

Drapeaux de l'ALU

- Mis à 1 ou à 0 après une opération
- Permettent les tests et branchements
- registre ASTAT

Bit	Nom	Description
0	AZ	ALU resultat nul ou sous-depassement de capacite en virgule flottante
1	AV	ALU depassement de capacite
2	AN	ALU resultat negatif
3	AC	ALU retenue issue d'un calcul en virgule fixe
4	AS	ALU signe de la donnee (ABS ....)
5	AI	ALU operation non valide en virgule flottante
10	AF	La derniere operation etait uneoperation non valide en virgule flottante
24-31	CACC	Accumulateur de comparaison(resultat des 8 dernieres comparaisons , le bit 31 donne la derniere comparaison)

- registre STKY (ces bits restent à 1 !)

Bit	Nom	Description
0	AUS	ALU sous-depassement de capacite en virgule flottante
1	AVS	ALU depassement de capacite en virgule flottante
2	AOS	ALU depassement de capacite en virgule fixe
5	AIS	ALU operation non valide en virgule flottante

Rem : - COMP (x,y); génère un 1 si x > y

- BIT CLR STKY AUS; retombe à 0.

Le multiplieur

- En virgule fixe :
  - capable d'effectuer des MAC
  - utilise un registre d'accumulation sur 80 bits MR
- En virgule flottante :
  - seulement la multiplication

```
Fn = Fx * Fy ;
```

- Drapeaux gérés par le multiplieur
  - registre ASTAT

Bit	Nom	Description
6	MN	MULTIPLIEUR resultat negatif
7	MV	MULTIPLIEUR depassement de capacite
8	MU	MULTIPLIEUR sous-depassement de capacite
9	MI	MULTIPLIEUR operation non valide en virgule flottante

  - registre STKY

Bit	Nom	Description
6	MOS	MULTIPLIEUR depassement de capacite en virgule fixe
7	MVS	MULTIPLIEUR depassement de capacite en virgule flottante
8	MUS	MULTIPLIEUR sous-depassement de capacite
9	MIS	MULTIPLIEUR operation non valide en virgule flottante

Unité de décalage
Shifter

- Travaille sur 32 bits sur des nombres à virgule fixe
  - porte sur Rn
- Décalages et rotations
- Mise à 1 ou 0 de bits, tests...
  - exemple d'instructions

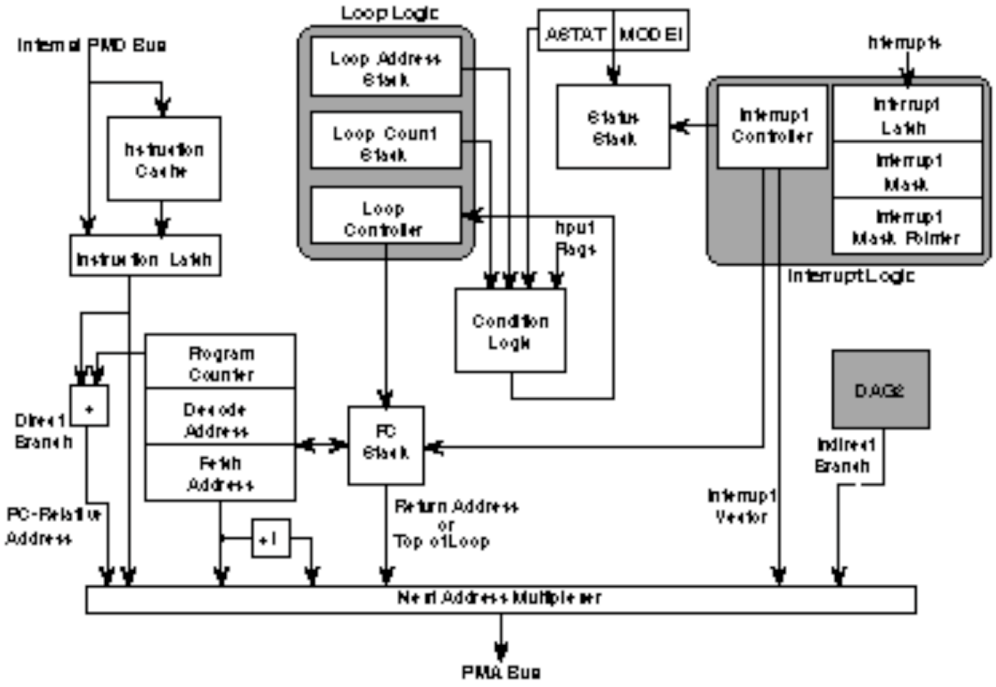
```
Rn = LSHIFT Rx BY Ry; /* signé >0 Left */
Rn = ROT Rx BY Ry ;
Rn = BCLR Rx BY Ry ;
Rn = BSET Rx BY Ry ;
BTST Rx BY Ry ;   /* set SZ si bit=0 */
```

- Drapeaux affectés :
  - dans le registre ASTAT

Bit	Nom	Description
11	SV	Depassement dans le decalage
12	SZ	Resultat nul dans le decalage
13	SS	Inutile

Séquenceur de programme

- Le séquenceur contrôle le déroulement du programme
  - linéaire
  - structures de boucles
  - sous-programmes
  - sauts
  - routines d'interruption
- Architecture :



Cycle d'instruction

- 3 cycles d'horloge pour gérer une instruction
  - lecture (mémoire ou cache)
  - décodage
  - exécution

- Chaque tâche est séparée
  - Structure "pseudo-parallèle" ou ***PipeLine***

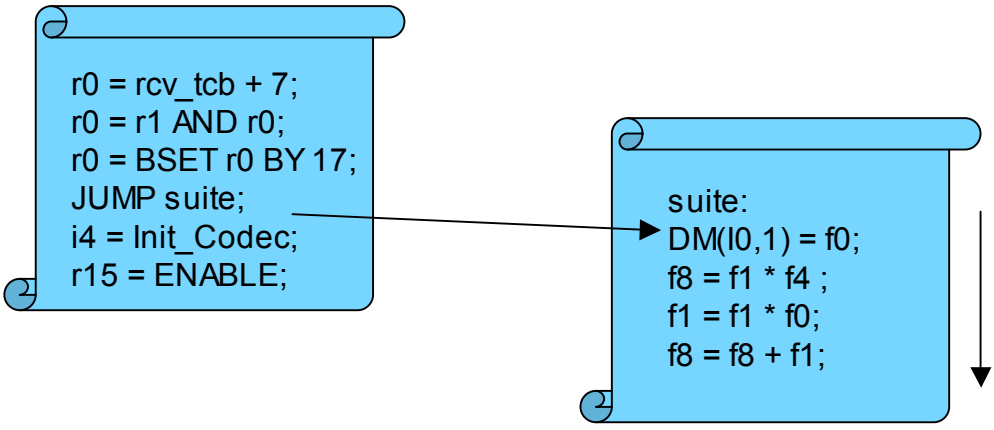
Temps (cycles)	Lecture	Décodage	Exécution
1	Instruction 1		
2	Instruction 2	Instruction 1	
3	Instruction 3	Instruction 2	Instruction 1
4	Instruction 4	Instruction 3	Instruction 2
….	….	….	….

  - Rupture du pipeline :
    conflit mémoire, sauts, boucles ....

Sauts

- Branchement :

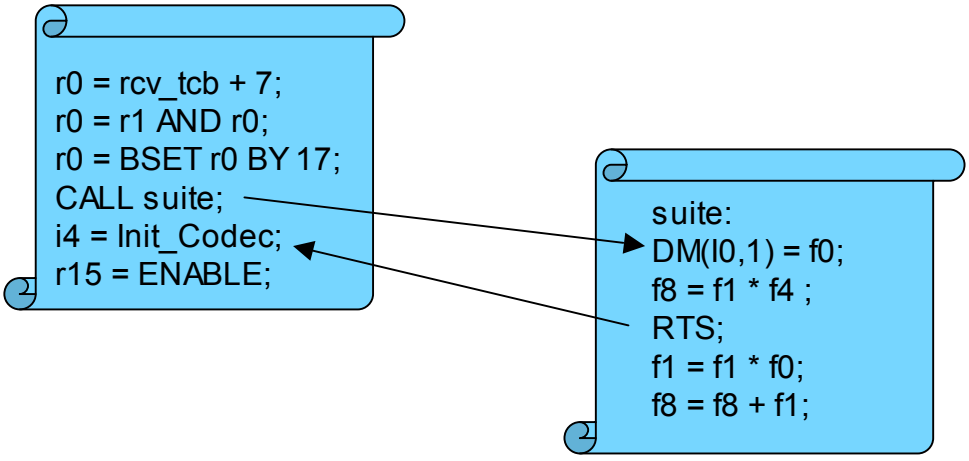
JUMP addr24 PC<- addr24



- Saut à un sous-programme :

CALL addr24 PC+1 -> PC Pile

PC <- addr24



Retour : RTS PC <- PC Pile

Branchements et sauts conditionnels

- JUMP et CALL peuvent être conditionnels :

```
IF condition JUMP addr24 ;
IF condition CALL addr24 ;
```

- Liste des conditions

Mnémonique	Description	Vrai si …
EQ	ALU :résultat = 0	AZ=1
LT	ALU :résultat < 0	AN=1 and AZ=0
LE	ALU :résultat ≤ 0	AZ=1 or AN=1
AC	ALU :retenue	AC=1
AV	ALU :dépassement	AV=1
MV	Multiplieur :dépassement	MV=1
MS	Multiplieur :résultat < 0	MN=1
SV	Shifter :dépassement	SV=1
SZ	Shifter :résultat = 0	SZ=1
TF	Test de Bit Vrai	BTF=1
LCE	Boucle terminée (Do Until)	CURLCNTR=1
NOT ICE	Boucle non terminée (If..)	CURLCNTR≠1
NE	ALU :résultat ≠0	AZ=0
GE	ALU :résultat ≥ 0	AN=0
GT	ALU :résultat > 0	AZ=0 and AN=0
NOT AC	ALU :pas de retenue	AC=0
NOT AV	ALU :pas de dépassement	AV=0
NOT MV	Multiplieur :pas de dépassement	MV=0
NOT MS	Multiplieur :résultat ≥ 0	MN=0
NOT SV	Shifter :pas de dépassement	SV=0
NOT SZ	Shifter :résultat non nul	SZ=0
NOT TF	Test de Bit Faux	BTF=0
FOREVER	Toujours Faux (Do Until)	
TRUE	Trajours Vrai (If…)	

- Tout ne s'applique pas au IF !!

Structures de boucle

- Le contrôle de la boucle se fait en interne.
- Boucle avec condition de fin :

```
DO  addr24 UNTIL  condition;
```

```
DO suite UNTIL EQ;
f6 = max (f2,f7);
suite:
r3 = r3 - 1;
r3 = 0x120;
```

- Boucle avec compteur :

```
LCNTR= data16 , DO  addr24 UNTIL  LCE;
```

```
LCNTR=12, DO test UNTIL LCE;
f2 = DM(I0,M1);
f6 = max (f2,f7);
test: DM(I2,M1) = f6;
f6 = -5.0;
```

- Les boucles peuvent être imbriquées (6 MAX)
- Pas de branchement sur les 3 dernières instructions de la boucle

Registres spéciaux du séquenceur

- La pile du PC (PC Stack ou PCSTKP)
  - adresse de retour
    - sous-programmes
    - routines d'interruption
  - adresse de début du boucle
  - taille 30*24 bits
  - registre STKY : bit 21 PCFL PC stack full
    bit 22 PCEM PC stack empty
  - la pile pleine (29) génère une interruption

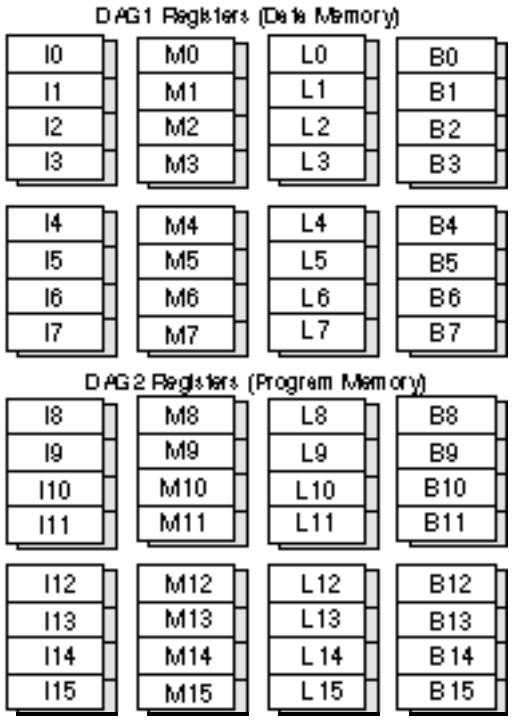
- Gestion des boucles
  - Pile d'adresse de fin de boucle LADDR
    - taille 6*32 bits (adresse + code de fin)
    - registre STKY : bit 25 LSOV pile pleine
      bit 26 LSEM pile vide
    - la pile pleine (6) génère une interruption
  - Pile de compteur de boucle associé à LADDR
    - taille 6*32 (nb boucle max 2^{32})
    - chaque case (imbrication de boucles) contient un compteur LCNTR qui est décrémenté à chaque boucle.

Génération d'adresse

- Accès direct
  - ex :

```
F1 = DM (0x0000C000) ;
R3 = PM (0x009400) ;
DM(0x0000C002) = R4;
```

- Accès indirect à la mémoire (pointeur)

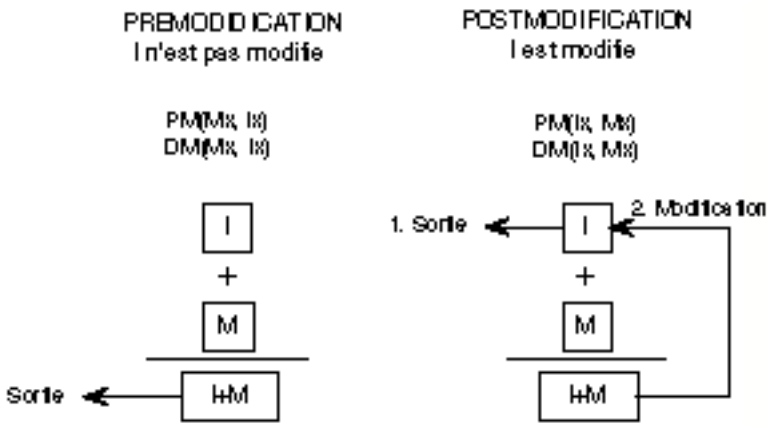


  - DAG 1 : 32 bits pour le bus DM
  - DAG2 : 24 bits pour le bus PM
  - Détail des registres
    - I : registre d'Index
    - M : registre d'incrément
    - B : registre d'adresse de Base
    - L : registre de Longueur

Buffer circulaire

Calcul d'adresse

- Calcul d'adresse




```
L0 = 0;
I0 =0x0000C010;
M0 = 1;
M1 = 2;
R2 = DM(M0,I0);
R3 = DM(M1,I0);
```

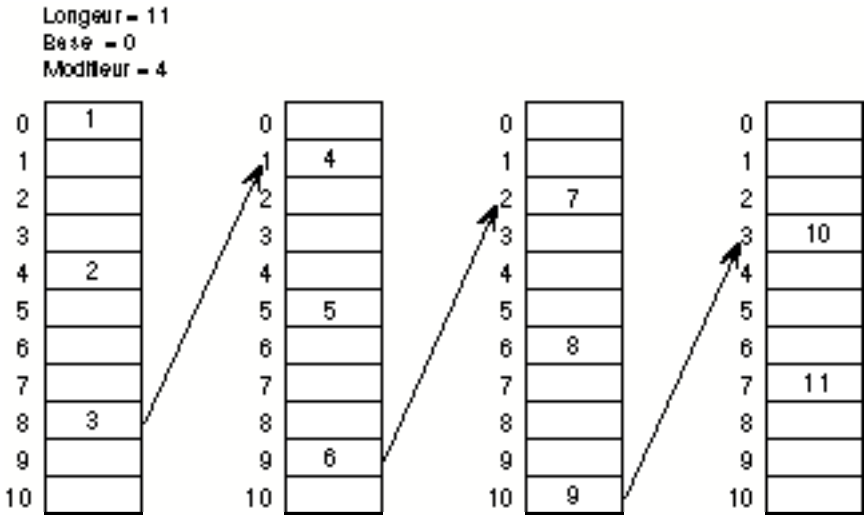
```
L2 = 0;
I2 =0x0000C010;
M2 = 2;
M3 = 0xFFFFFFFD
R2 = DM(I2,M2);
R3 = DM(I2,M2);
R4 = DM(I2,M3);
R5 = DM (I2,0);
```

- M peut être une valeur *immédiate*
- L doit avoir une valeur nulle

Buffer circulaire

- Table de valeurs avec historique fini
- Utilise que le mode post-modifié :

  B0 et L0 doivent être auparavant initialisés



- ex :

```
 B0 = 0;
L0 = 11;
.....
f1 = DM (I0,M0);
```

- Pour I7 et I15 : interruption quand modulo

Les interruptions

- Interruption ??
  - Déroutement du DSP vers une routine d'interruption
  - Vecteur = adresse routine (unique pour chaque type)
  - 3 interruptions externes : broches $\overline{IRQ}_{0\text{-}2}$ (état ou front)
  - internes : résultat mathématique, piles
    buffer circulaire, Timers
- Prise en compte de la demande d'interruption
  - pas de masquage
  - pas de demande de + haut niveau
  - interruption activées
- Séquencement

  - déroutement vers la routine d'interruption

    • sauvegarde du PC -> pile (ASTAT et MODE1 pour IRQ ou Timer)
    • Mise à 1 dans le regitre IRPTL
    • Mise à jour du masque (1)
    • branchement à la 1er instruction du vecteur

  - fin de la routine -> RTI;

    • Restitution du PC depuis la pile (ASTAT et MODE1 pour IRQ ou Timer)
    • Mise à 0 dans le regitre IRPTL
    • Mise à jour du masque (0)
    • branchement à l'adresse de retour

Vecteurs d'interruption

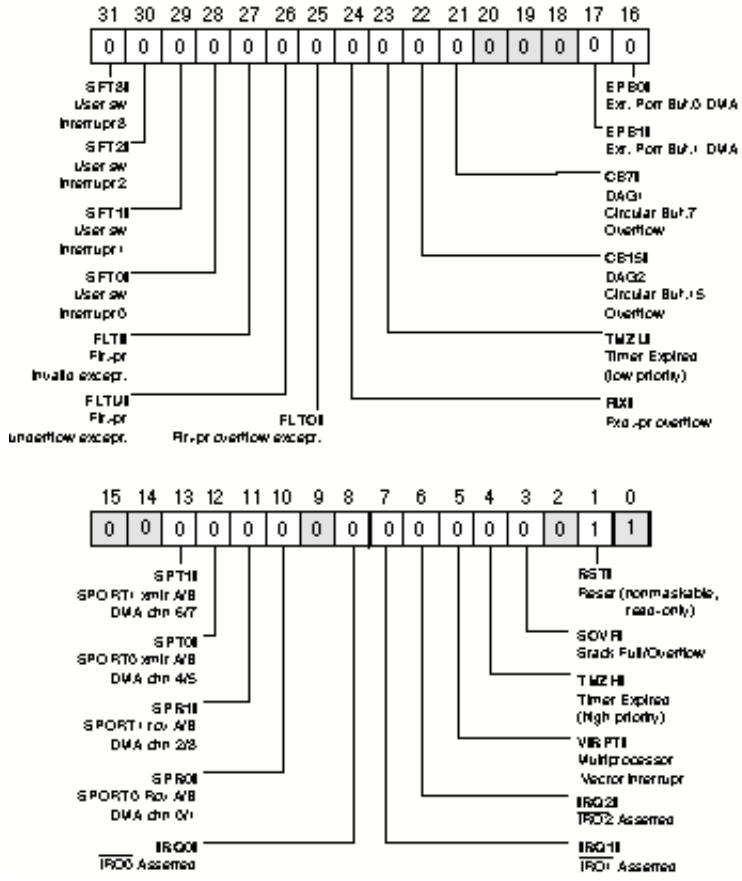
- Adresse de base : 0x00008000

Bit	Address	Name	Description	Priority
0	0x00	Reserved		
1	0x04	RSTI	Reset (read-only, non-maskable)	Highest
2	0x08	Reserved		
3	0x0C	SOVFI	Status stack or loop stack overflow or PC full	
4	0x10	TMZHI	Timer high priority option	
5	0x14	VIRPTI	Vector interrupt	
6	0x18	IRQ2I	$\overline{\text{IRQ2}}$ asserted	
7	0x1C	IRQ1I	$\overline{\text{IRQ1}}$ asserted	
8	0x20	IRQ0I	$\overline{\text{IRQ0}}$ asserted	
9	0x24	Reserved		
10	0x28	SPR0I	DMA channel 0/1; SPORT0 receive A&B	
11	0x2C	SPR1I	DMA channel 2/3; SPORT1 receive A&B	
12	0x30	SPT0I	DMA channel 4/5; SPORT0 transmit A&B	
13	0x34	SPT1I	DMA channel 6/7; SPORT1 transmit A&B	
14	0x38	Reserved		
15	0x3C	Reserved		
16	0x40	EP0I	DMA channel 8; Ext. port buffer 0	
17	0x44	EP1I	DMA channel 9; Ext. port buffer 1	
18	0x48	Reserved		
19	0x4C	Reserved		
20	0x50	Reserved		
21	0x54	CB7I	Circular buffer 7 overflow	
22	0x58	CB15I	Circular buffer 15 overflow	
23	0x5C	TMZLI	Timer low priority option	
24	0x60	FIXI	Fixed-point overflow	
25	0x64	FLT0I	Floating-point overflow exception	
26	0x68	FLTUI	Floating-point underflow exception	
27	0x6C	FLTII	Floating-point invalid exception	
28	0x70	SFT0I	User software interrupt 0	
29	0x74	SFT1I	User software interrupt 1	
30	0x78	SFT2I	User software interrupt 2	
31	0x7C	SFT3I	User software interrupt 3	Lowest

Registres de contrôle des interruptions

- IRPTL (32 bits)
  - indique par un flag les interruptions en cours ou en attente
  - on peut y écrire : interruption logicielle

    ex : `BIT SET IRPTL SFT1I;`



- IMASK (32 bits)
  - permet de masquer ou non une interruption

    ex : `BIT CLR IMASK IRQ2I;`

Interruptions (fin)

- Activation globale
  - bit 12 (IRPTEN) de MODE1
    - 1 : activée
    - 0 : désactivée
- Priorité
  - cas de 2 interruption simultanées
  - 0 la plus haute, 31 la plus basse
- Interruption multiples
  - mode actif : NESTM=1 dans MODE1
  - déroutement vers interruption de plus haut niveau
- Exceptions mathématiques
  - la routine doit gérer le registre STKY (mise à 0)

- Instructions IDLE et IDLE16
  - mis attente du DSP
  - réveil par IRQ, Timer ou transfert DMA (cas de IDLE)
  - IDLE16 ne permet pas la gestion du DMA ou SPORT mais Fclock / 16 !!
  - Gestion de l'énergie

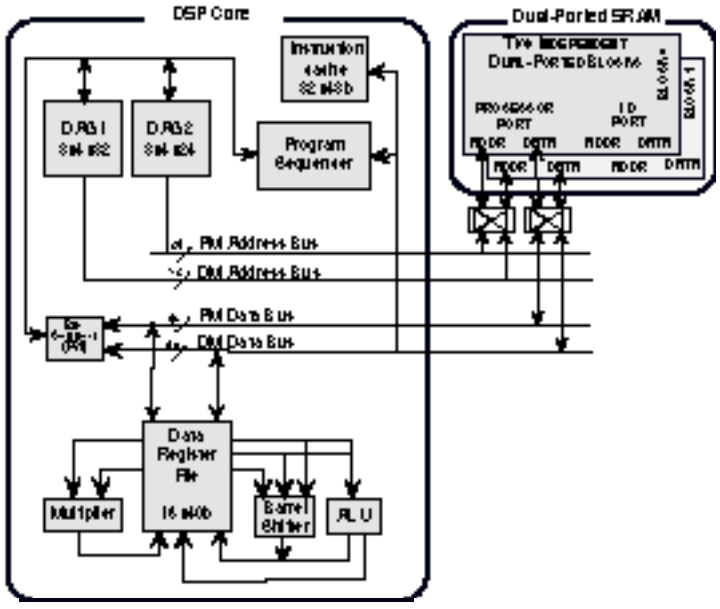
Le cache d'instruction

- un cache ? gestion des conflits d'accées mémoire
  - cache de 32 instructions (transparent)
  - utilisé que pour régler les conflits ≠ microP

- Exemple : l'instruction n accède à la mémoire programme

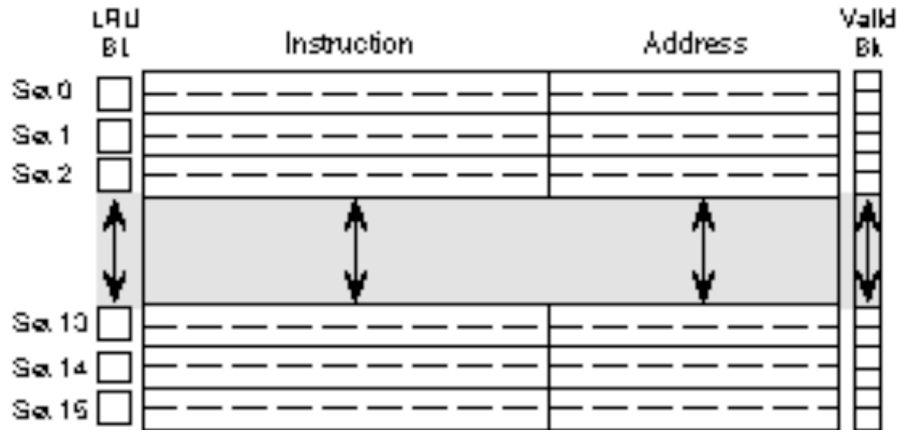
```
R8 = PM (0x00009000);
```

Exécution	Instruction n
Décodage	Instruction n+1
Lecture	Instruction n+2



- DM(0x00009000) ou instruction 2 ??

Structure du cache



- 16 entrées (ou *set*)
- chaque entrée possède 2 couples instruction/adresse
- Bit de validité
- numéro du *set* : 4 bits LSB de l'adresse
- LRU : instruction du *set* la moins recement utilisée

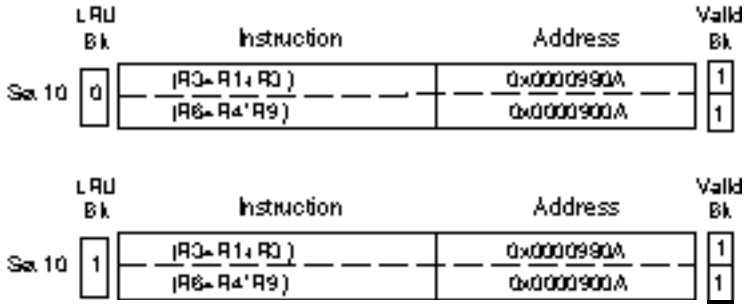
- Gestion :
  - 'cache hit' : instruction est dans le cache,
    le DSP récupère la donnée via PM Data Bus
  - 'cache miss' : instruction n'est pas dans le cache,
    le DSP prend un cycle de plus pour charger l'instruction dans le cache

Utilisation du cache

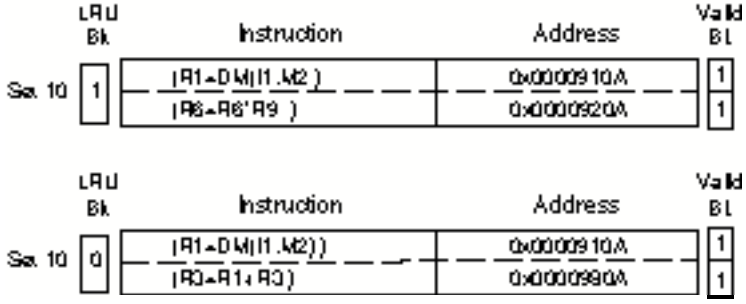
– exemple :

```
0x00009808:R1=PM(0x00009801);
0x00009809:R2=ROT R1 By 8;
0x0000980A:R3=R1 + R3;
0x0000980A: .....
```

– instruction présente dans le cache :



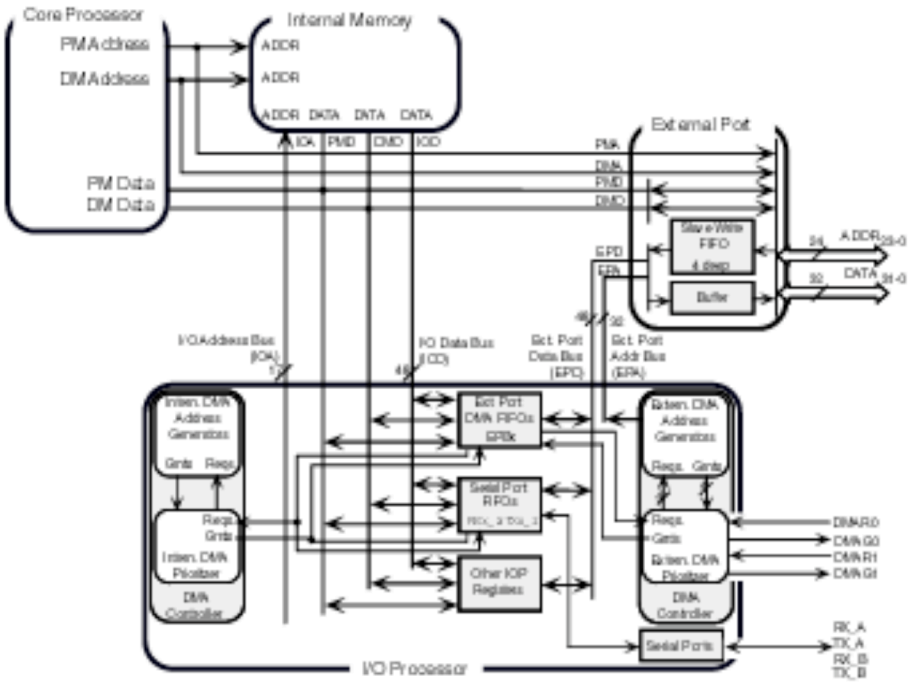
– instruction absente du cache :



- Rem : Désactivation du cache : CADIS dans MODE2

Le DMA

- *Direct Memory Access (module IØO)*
  - Transfert blocs de données
  - Soulage le cœur du DSP



Transferts :

mémoire interne <--> mémoire externe

mémoire interne <--> Ports séries SPORT

mémoire interne <--> Module I/O d'un DSP

Déclenchement externe : $\overline{DMAR_{2-1}}$ et $\overline{DMAG_{2-1}}$

Liste des canaux DMA

- 10 canaux DMA : 10 Buffers FIFO (internes I/O)

Chn	Data Buffer	Description
0	RxOA	Serial port 0 receive; A data
1	Rx1A	Serial port 1 receive; A data
2	RxOB	Serial port 0 receive; B data
3	Rx1B	Serial port 1 receive; B data
4	TxOA	Serial port 0 transmit; A data
5	Tx1A	Serial port 1 transmit; A data
6	TxOB	Serial port 0 transmit; B data
7	Tx1B	Serial port 1 transmit; B data
8	EPBO	External port FIFO buffer 0
9	EPB1	External port FIFO buffer 1

- Ports externes (2 canaux) :
  - transfert sur 48 bits max
  - registres de contrôle dédiés $DMAC_{0-1}$
  - bi-directionnels (configuartion)
- SPORTs (8 canaux):
  - transfert sur 32 bits entre buffer et mémoire
  - direction figée : réception SPORT -> mémoire
    émission mémoire -> SPORT
  - compression possible sur 16bits E/R en //
  - contrôle dans $SRCTL_{0-1}$

DMA : registres de paramètres

- Configuration des transferts
- "Mappés" en mémoire (accés simulaire à la mémoire)

DMA Chn	Registers	Description
0	IIR0A, IMR0A , CR0A, CPR0A, GPR0A	SPORT 0 receive; A data
1	IIR0B, IMR0B, CR0B, CPR0B, GPR0B	SPORT 0 receive; B data
2	IIR1A, IMR1A, CR1A, CPR1A, GPR0A	SPORT 1 receive; A data
3	IIR1B, IMR1B, CR1B, CPR1B, GPR1B	SPORT 1 receive; B data
4	IIT0A, IMT0A, CT0A, CTR0A, GPT0A	SPORT0 Transmit; A data
5	IIT0B, IMT0B, CT0B, CPT0B, GPT0B	SPORT0 Transmit; B data
6	IIT1A, IMT1A, CT1A, CTR1A, GPT1A	SPORT 1 Transmit; A data
7	IIT1B, IMT1B, CT1B, CPT1B, GPT1B	SPORT 1 Transmit; B data
8	IIEP0, IMEP0, CEP0, CPEP0, GPEP0, EIEP0, EMEP0, ECEP0	External Port Buffer 0
9	IIEP1, IMEP1, CEP1, CPEP1, GPEP1, EIEP1, EMEP1, ECEP1	External Port Buffer 1

DMA : registres de paramètres

- IIx : registre d'index sur 17 bits
  - offset de 0x000 8000 implicite
    - accés mémoire 0x0000 8000 à 0x0002 8000
- IMx : registre de modification sur 16 bits
  - nombre signé
    - après chaque transfert IIx=IIx + IMx
- Cx : compteur de transfert sur 16 bits
  - décrémenté à chaquetransfert de données
  - passage à 0 : fin de transfert du bloc. (Interruption)
  - si Cx initialisé à 0 : 2^{16} données à transférer !
- Cx et GPx : utilisés pour le chaînage
- Cas du port externe :
  - EIx, EMx, ECx : génère les adresses de la mémoire externe
- Exemple :

```
r0 = 0x00001000;
DM(IIR0A)= r0;
r0 = 0x0001;
DM(IMR0A)=r0;
r0 = 0x00FF;
DM(CR0A)=r0;
r0 = DM(SRCTL0);
r0 = BSET r0 by 18;
DM(SRCTL0) = r0;
```

pour relancer : DEN=0 / initialiser les registres / DEN=1

Contrôle DMA (suite)

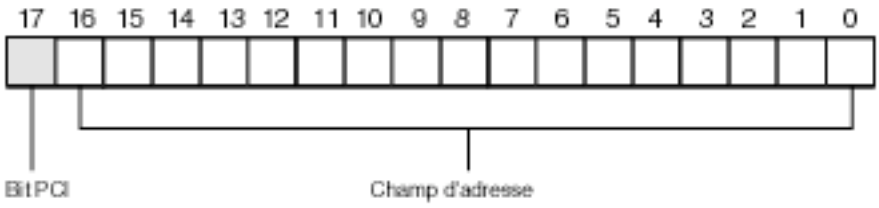
- Priorité des canaux
  - cas ou plusieurs transferts sur plusieurs canaux :

Priority	Core Access to I/O Registers	
	DMA Chn	Port/Buffer
Highest	0	Serial port 0 receive; Rx0_A
	1	Serial port 0 receive; Rx0_B
	2	Serial port 1 receive; Rx1_A
	3	Serial port 1 receive; Rx1_B
	4	Serial port 0 transmit; Tx0_A
	5	Serial port 0 transmit; Tx0_B
	6	Serial port 1 transmit; Tx1_A
	7	Serial port 1 transmit; Tx1_B
	NA	TCB loading requests
	8	External port buffer 0
Lowest	9	External port buffer 1

- Arrêt du transfert :
  - le registre arrive à 0
  - on force DEN=0 (sans chaînage)
    - le transfert reprend si DEN passe à 1

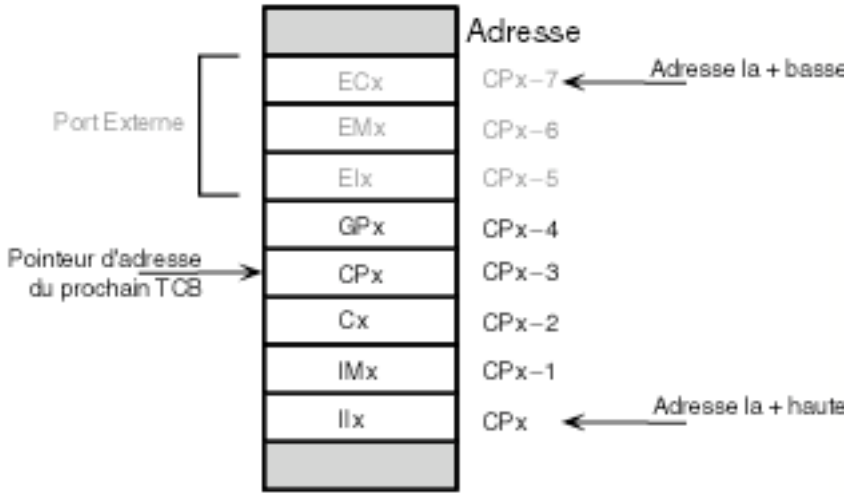
DMA "*chaining*"

- Enchaînement automatique de transfert de données
- CPx pointe sur un nouveau bloc de paramètre
  - TCB : Transfert Control Bloc (en mémoire)
- Structure du CPx :



PCI : interuption à la fin du transfert du bloc courant

- Structure du TCB :



- Initialisation du transfert :
  - Ecriture des TCB's en mémoire
  - Mise à 1 de DEN et CHEN
  - Ecriture dans CPx de l'@ du 1er TCB

DMA : interruptions

- Cx passe à 0 : déroute le DSP en interruption
- Chaque canal a sa propre routine
- Utilisation de IRPTL et IMASK
  - PCI dans le cas d'un enchaînement
- Priorité des interruptions :

Interrupt	Description	Priority
SPR0I	DMA channels 0, 1; SPORT 0 receive	Highest
SPR1I	DMA channels 2, 3; SPORT 1 receive	
SPT0I	DMA channels 4, 5; SPORT 0 transmit	
SPT1I	DMA channels 6, 7; SPORT 1 transmit	
EP0I	DMA channel 8; Ext. port buffer 0	
EP1I	DMA channel 9; Ext. port buffer 1	Lowest

DMASTAT

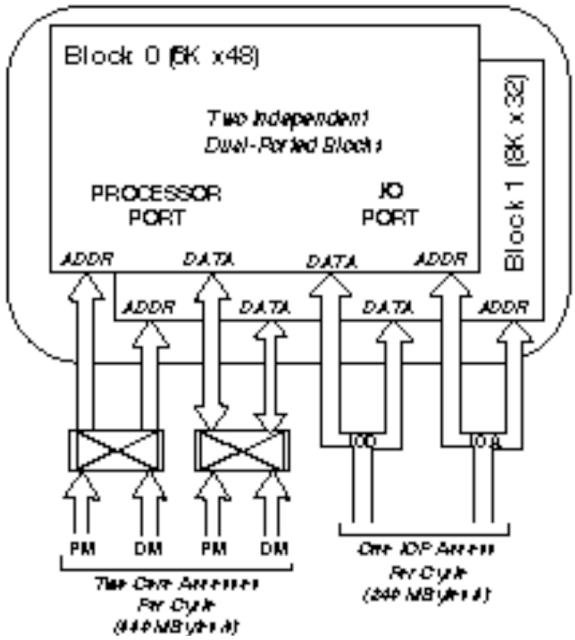
- Registre de status sur 32 bits (20 bits significatifs)
- Etat du canal bit $_{0\text{-}9}$: 1 actif / 0 inactif
- Etat de l'enchaînement bit $_{10\text{-}19}$: 1 transfert de TCB
  0 inactif

Bit	DMA Channel	Status for.....
0	0	Rx0_A
1	2	Rx1_A
2	4	Tx0_A
3	6	Tx1_A
4	1	Rx0_B
5	3	Rx1_B
6	8	EPB0
7	9	EPB1
8	5	Tx0_B
9	7	Tx1_B
10	0	Chaining on Rx0_A
11	2	Chaining on Rx1_A
12	4	Chaining on Tx0_A
13	6	Chaining on Tx1_A
14	1	Chaining on Rx0_B
15	3	Chaining on Rx1_B
16	8	Chaining on EPB0
17	9	Chaining on EPB1
18	5	Chaining on Tx0_B
19	7	Chaining on Tx1_B
20-31	Reserved	

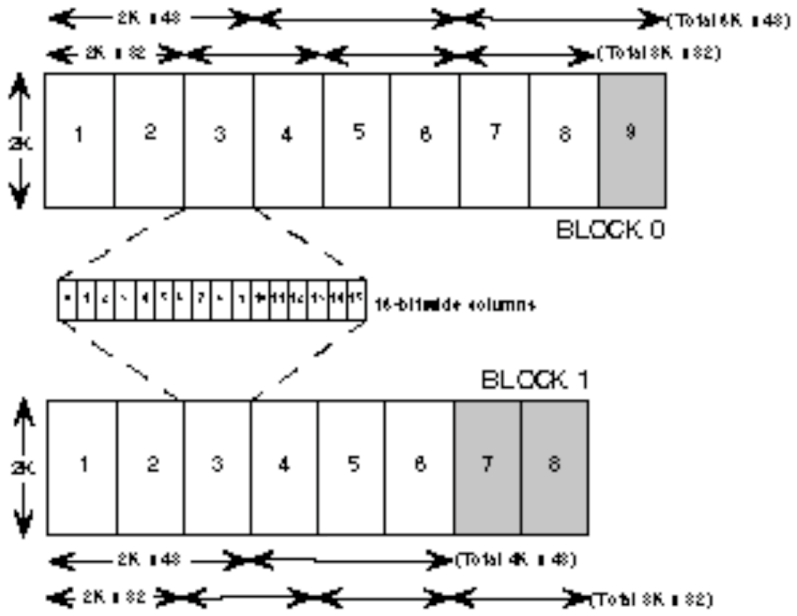
- Alternative aux interruption (Polling)

La mémoire (interne)

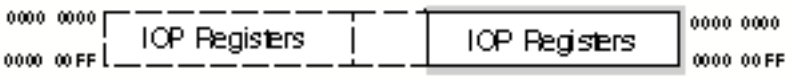
- Mémoire interne SRAM double port

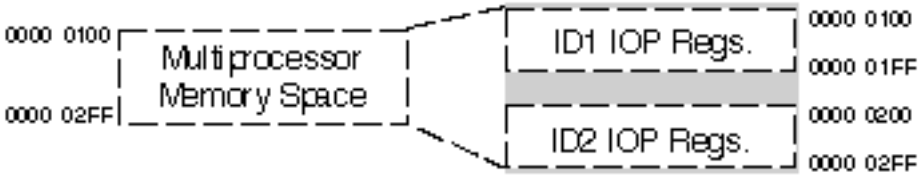


- Organisation de la mémoire



La mémoire interne (suite)



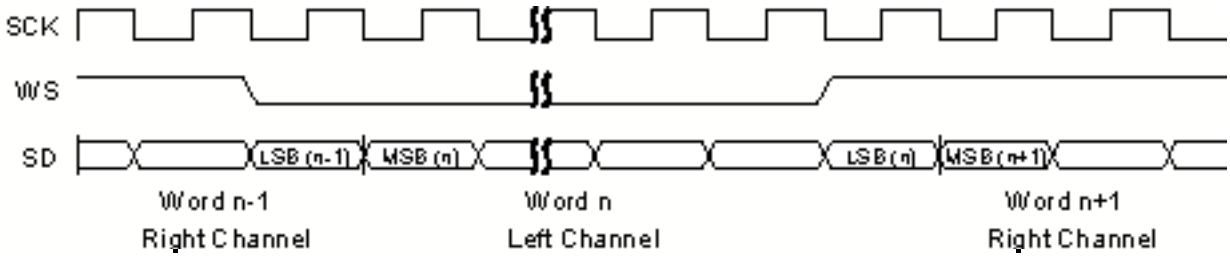


– cas de la platine de développement EZ-LAB

Adresse de debut	Adresse de fin	Contenu
0x0000 0000	0x0000 02FF	Registres d'entree/sortie
0x0000 8000	0x0000 9FFF	Block 0 Adressage normal; mots de 48 et 32 bits (SRAM)
0x0000 C000	0x0000 DFFF	Block 1 Adressage normal; mots de 48 et 32 bits (SRAM)
0x0001 0000	0x0001 3FFF	Block 0 Adressage pour mots de 16 bits (SRAM)
0x0001 8000	0x0001 BFFF	Block 1 Adressage pour mots de 16 bits (SRAM)
0x0002 0000	0x0002 FFFF	EPROM (systeme EZLAB)
0x0100 0000	0x0100 0000	BMAFE Adresse (systeme EZLAB)
0x0100 0001	0x0100 0001	BMAFE Donnee (systeme EZLAB)
0x0100 0004	0x0100 0007	UART (systeme EZLAB)
0x0300 0000	0x0310 0000	SDRAM (systeme EZLAB)

Ports séries : SPORTx

- 2 ports séries synchrones indépendants
    - SPORT0 et SPORT1
- 2 canaux par port (canal A et canal B)
- Emission / Réception simultanées (Full Duplex)
- Débit max : 30 Mbit/s
- Modes :
  - Standard : émission mono-données
  - I^2S :Inter IC System : émission voies droit et gauche avec TDM

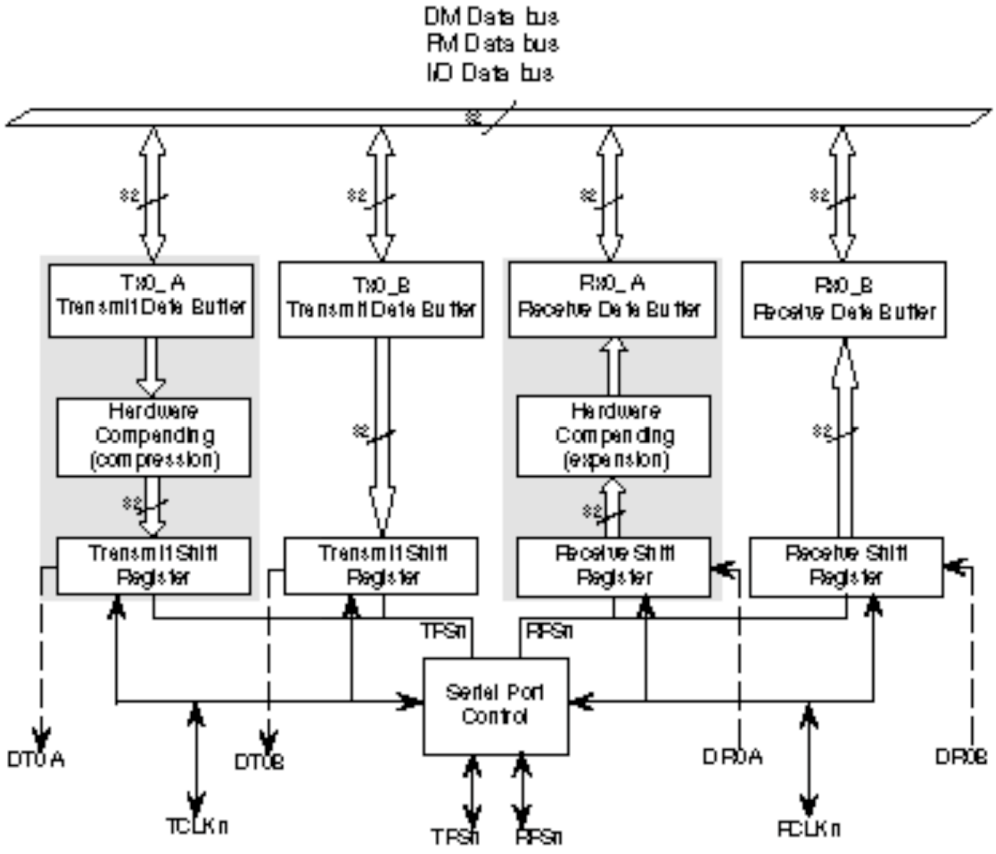


  - Multicanaux : 1 seul canal utilisé en TDM
- Données de longueur variable : 3 à 32 bits
- Contrôle DMA

SPORTx : Structure

- Communication synchrone sur CLK (interne ou externe)
- FS "frame synchro" : signaux de synchronisation de trame (début d'1 ou plusieurs données)
- Connections externes :

Function	SPORT0		SPORT1	
	A Chn	B Chn	A Chn	B Chn
Transmit data	DT0A	DT0B	DT1A	DT1B
Transmit clock	TCLK0		TCLK1	
Transmit frame sync/ word select	TFS0		TFS1	
Receive data	DR0A	DR0B	DR1A	DR1B
Receive clock	RCLK0		RCLK1	
Receive frame sync	RFS0		RFS1	



SPORTx : registres

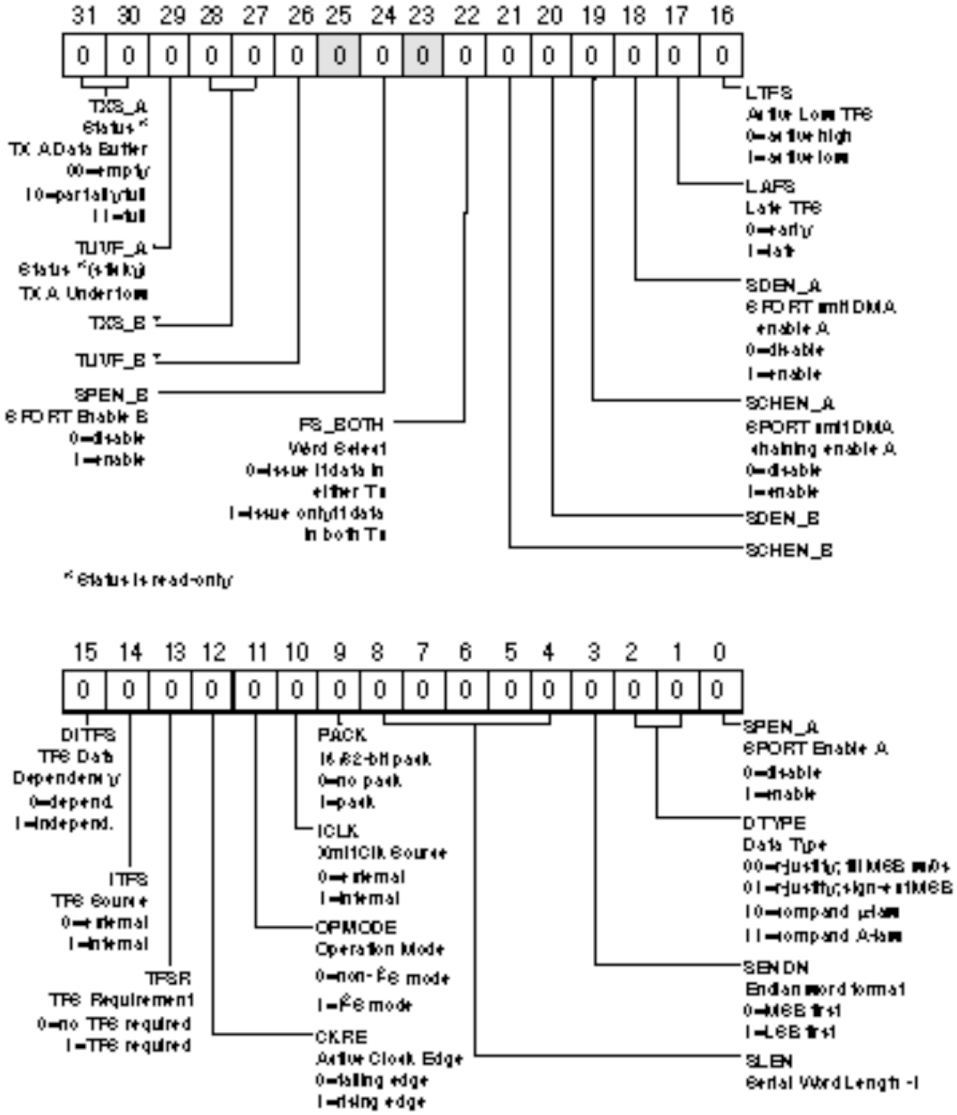
- 28 registres de 32 bits "mappés" en mémoire :

Register	Function
STCTLx	SPORT transmit control register
TXx_z[1]	Transmit data buffer
TDIVx	Transmit clock and frame sync divisors
MTCSx	Multichannel transmit select
MTCCSx	Multichannel transmit compand select
SRCTLx	SPORT receive control register
RXx_z[1]	Receive data buffer
RDIVx	Receive clock and frame sync divisors
MRCSx	Multichannel receive select
MRCCSx	Multichannel receive companding select
KEYWDx	SPORT receive comparison register
IMASKx	SPORT receive comparison mask

- Cas du mode standard :
  - TX et RX : écriture / lecture de 32 bits ou moins (justification à droite)
  - TX : on y écrit une donnée à sérialiser. Si rien dans ce buffer, interruption "TX Buff not full" (pas DMA)
  - RX : on y lit une donnée désérialisée. Quand une donnée est disponible, interruption "RX Buff not empty" (pas DMA)

Registres de contrôle

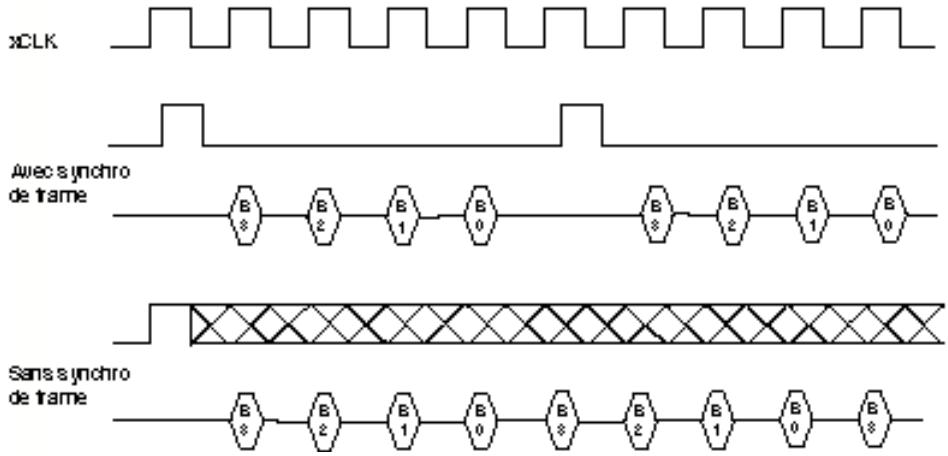
- STCTLx en mode standard :



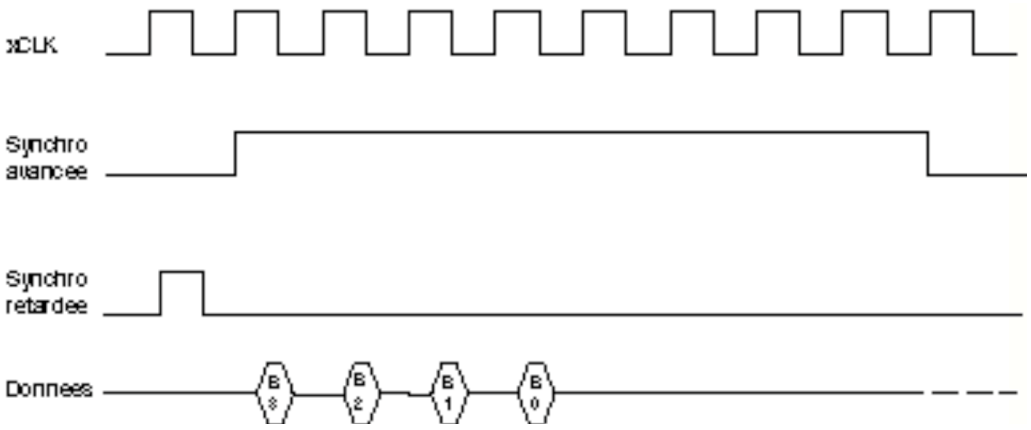
- DTYPE : - formatées à droite (extension signe?)
  - A-Law et mlaw compression GSM
- PACK : reçoit 2 valeurs sur 16bits -> une donnée sur 32 bits

Synchronisation de trame

- TFSR : active ou non synchro de trame(FS)



- ITFS : signal FS externe ou interne
- DITFS : signal indépendant ou non de l'état du buffer TX
- LTFS : signal FS actif haut ou bas
- LAFS : signal FS en retard ou en avance

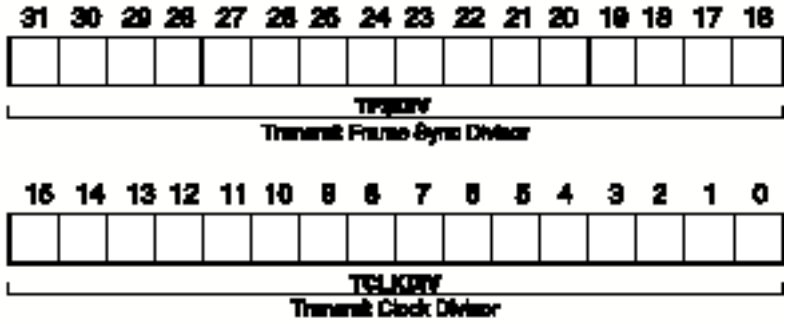


Fréquence de débit et synchronisation

- 2 registres fixes la fréquence du débit sériel et du signal de synchro :
  - TDIV pour transmission
  - RDIV pour réception

- Ex: TDIV

Bits	Name	Definition
15-0	TCLKDIV	Transmit clock divisor
31-16	TFSDIV	Transmit frame sync divisor



$$TCLKDIV = \frac{2 * f_{CLKIN}}{\text{fréquence du débit série}} - 1$$

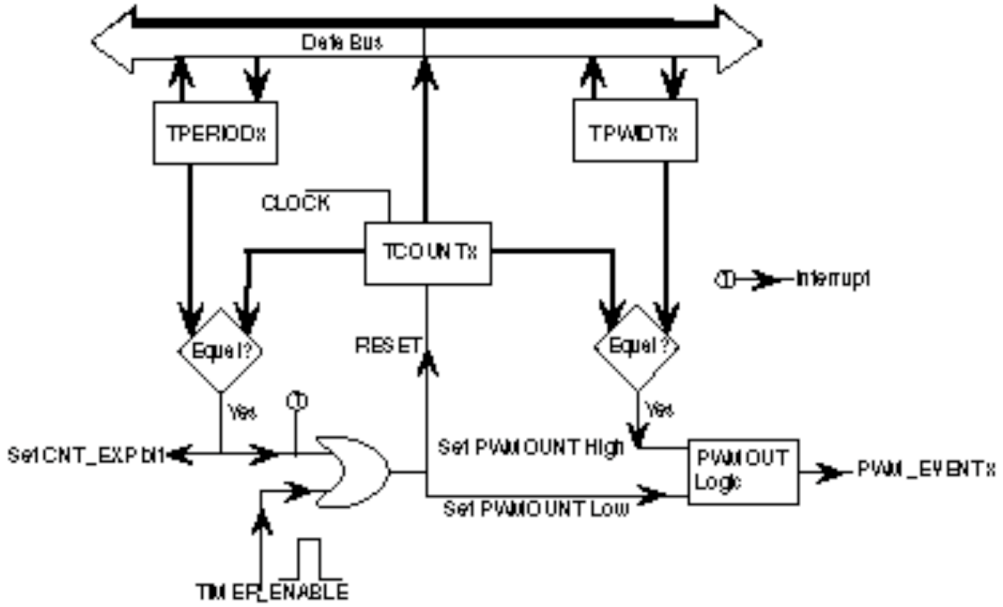
$$TFSDIV = \frac{\text{fréquence du débit série}}{\text{fréquence de la synchro.}} - 1$$

Timers

- 2 Timers internes indépendants
- 2 modes de fonctionnement :
  - Génération de signaux PWM (Pulse Width Modulation)
  - Capture d'un signal logique (période et rapport cyclique)
- 1 broche par Timer (PWM_EVENTx) In/Out
- 3 registres de 32 bits par Timer ("Mappés" en mémoire) :
  - TPERIODx
  - TPWIDTHx
  - TCOUNTx
- Timers utilisent l'horloge interne (2*CLKIN)
  - Tmax = $(2^{32}-1)$ * 16.67 ns = 71.5 s
- Activation du Timer_x :
  - TIMENx=1 dans le registre MODE2

Génération de signaux PWM

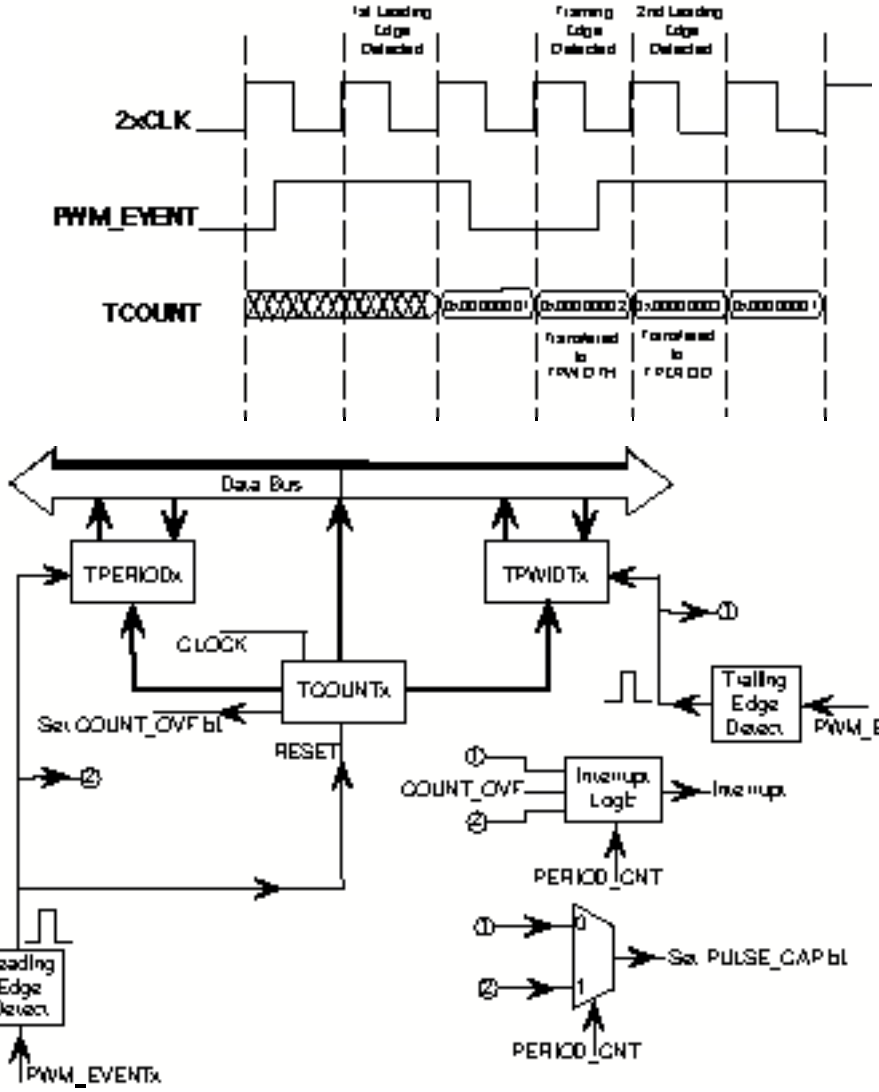
- PWMOUT Mode
  - sélectionné dans MODE 2
  - PWM_EVENTx est alors une sortie



- Comment ça fonctionne ?
  - à l'activation du Timer TCOUNTx=1
  - PWM_EVENTx = 1 quand TCOUNTx=TPWIDTHx
  - PWM_EVENTx = 0 quand TCOUNTx=TPERIODx
- génère une interruption quand TCOUNTx=TPERIODx
  - Mis à 1 du bit CNT_EXPx dans STKY

Mode Capture

- WIDTH_CNT Mode
  - sélectionné dans MODE 2
  - PWM_EVENTx est alors une entrée
- Fonctionnement :

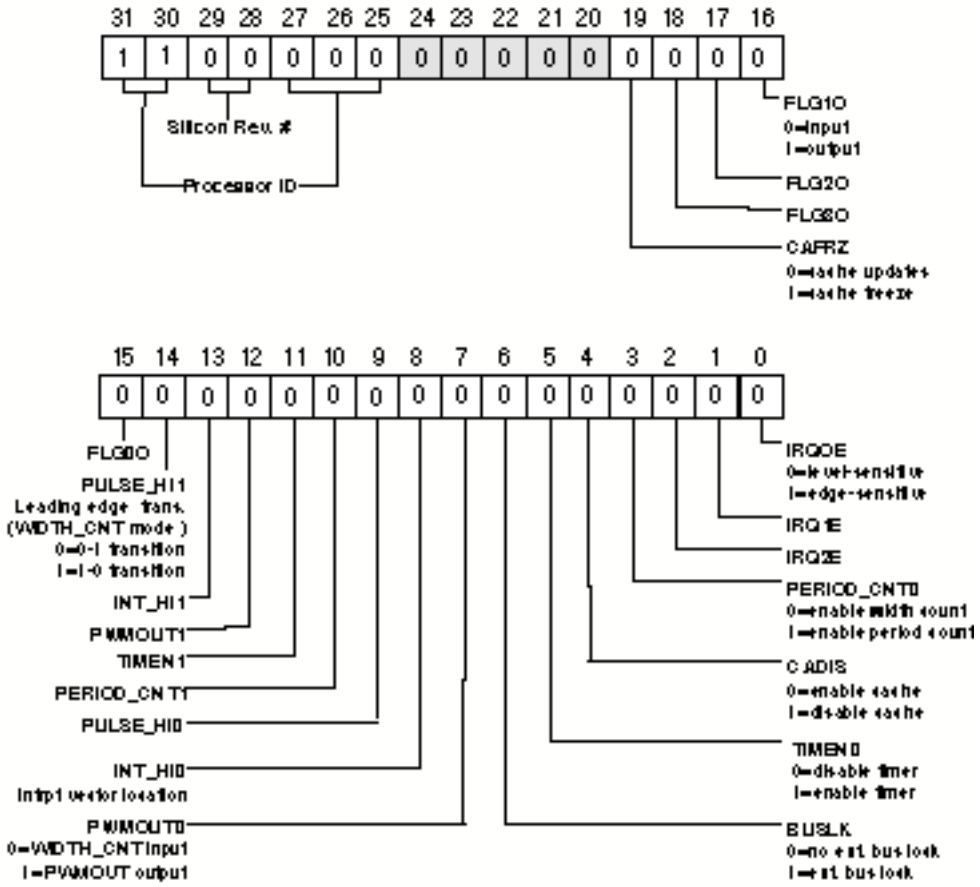


- Interruption générée :
  - au choix sur écriture de TPWIDTHx ou TPERIODx
  - quand il y a dépassement de TCOUNTx

(PULSE_CAP et CNT_OVF dans STKY)

Timer : Contrôle et interruptions

- Configuration dans MODE2 :



- INT_HIx : configuration du niveau d'interruption:

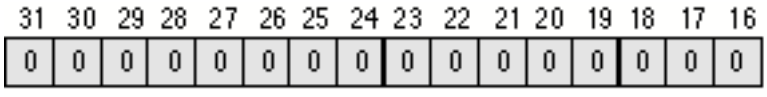
INT_HI1	INT_HI0	Status
0	0	Both timers latch to TMZLI
0	1	timer1 => TMZLI, timer0 => TMZHI
1	0	timer1 => TMZHI, timer0 => TMZLI
1	1	Both timers latch to TMZHI

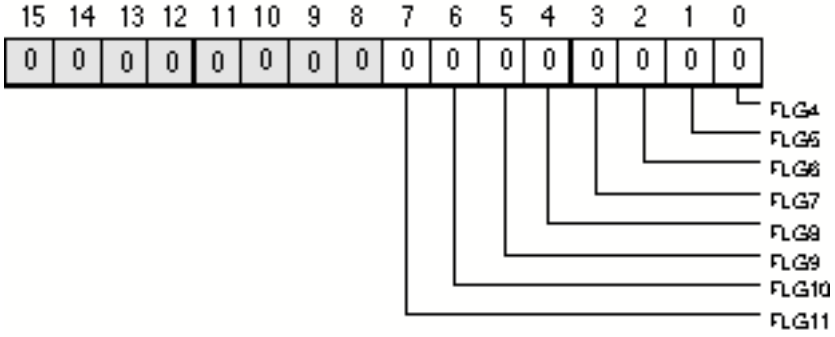
Port d'entrées/sorties numériques

- 12 lignes d'entrées/sorties $FLAG_{11-0}$
- Cas de $FLAG_{3-0}$:
  - configuration dans MODE2 : 0 entrée, 1 sortie
  - état de la ligne fixée dans ASTAT

```
BIT SET MODE2 FLG1O;
BIT SET ASTAT FLG1;
```

- Cas de $FLAG_{11-4}$:
  - On utilise 2 registes "mappés" en mémoire :
    - IOCTL: configuration
    - IOSTAT : état

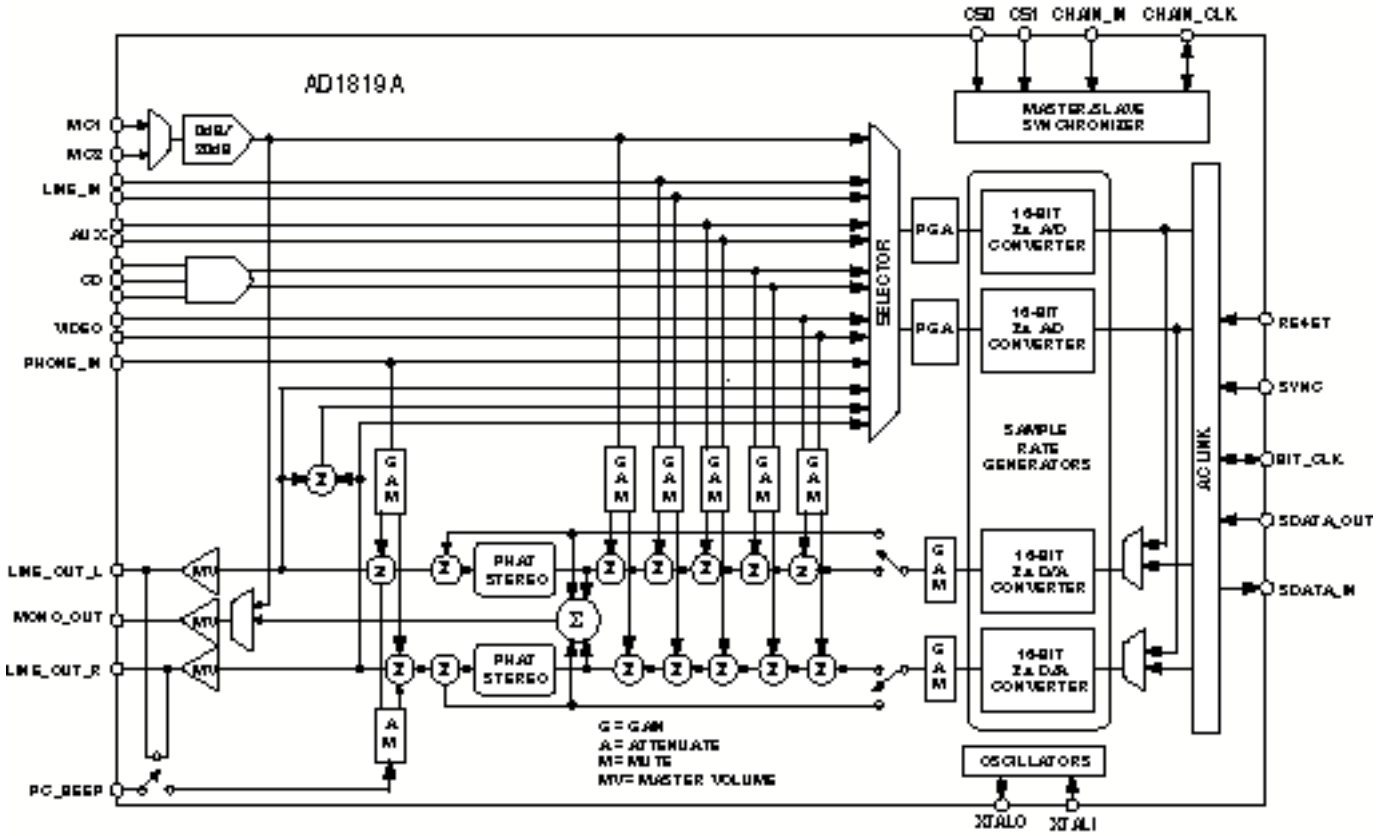




  - On en peut pas utiliser l'instruction BIT .....
    - Passage par registres Rx

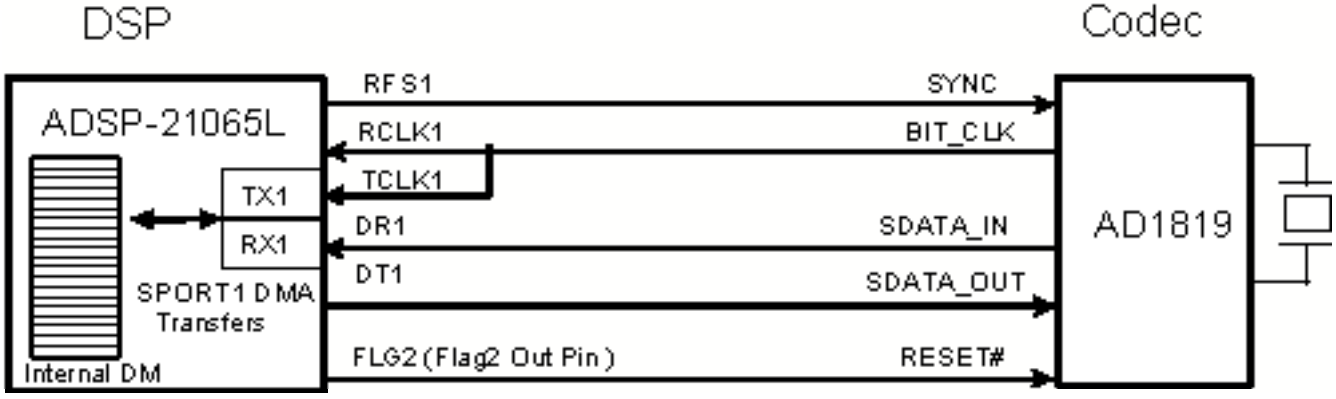
La conversion analogique-digitale

- Utilise le **Codec AD1819A**
  - Conversion sur 16 bits stéréo
  - Plusieurs entrées et sorties analogiques (multiplexage)
  - Fréquence d'échantillonnage de 7kHz à 48kHz
  - Bande passante en sortie de 20Hz à 20KHz
  - Amplis et atténuateurs internes programmables



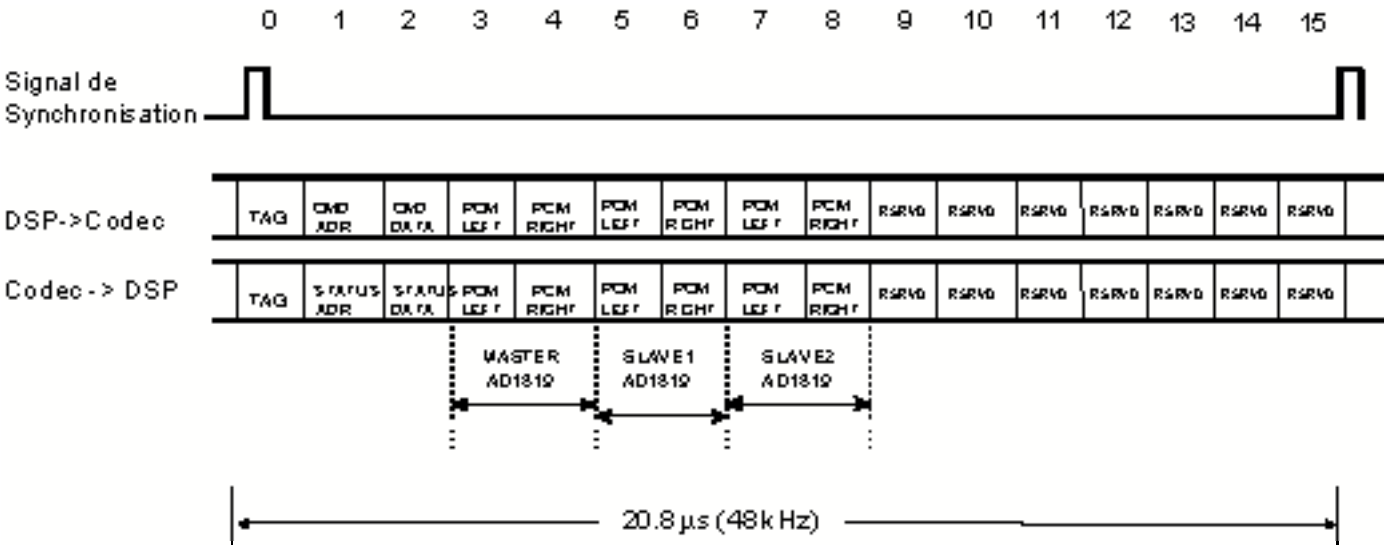
Connections avec le DSP

- Communication série haut débit utilisant le SPORT1



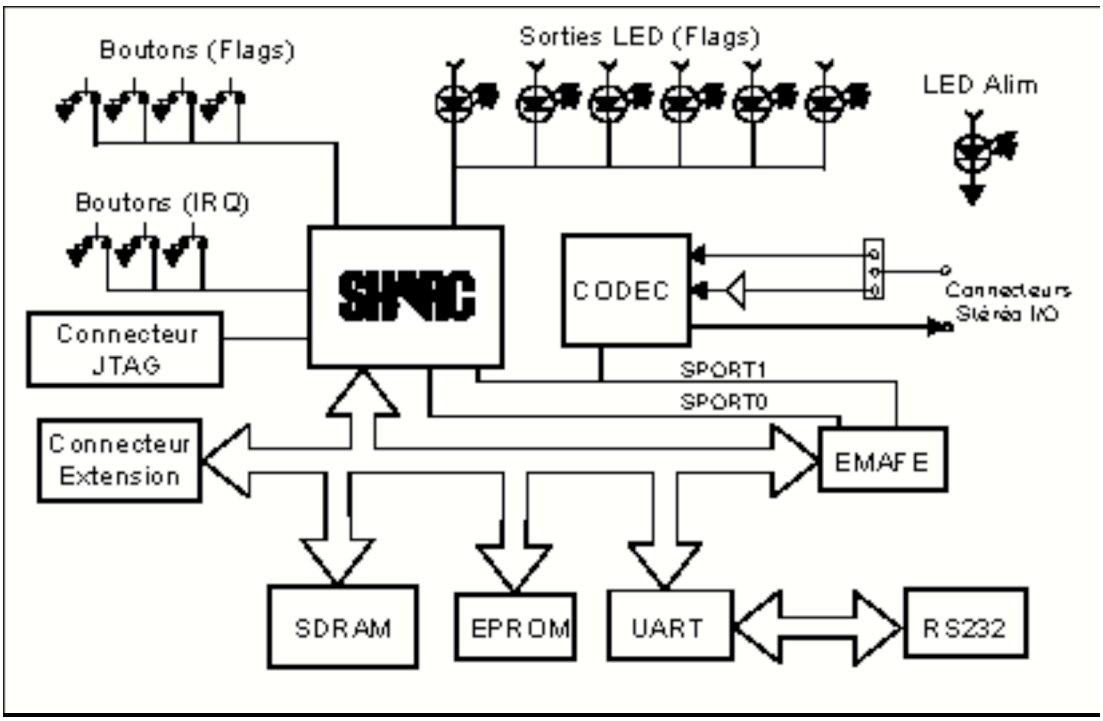
  - Initialisation du DMA et SPORT
  - Initialisation du Codec
  - Gestion des échantillons : routine d'interruption

- Format des échanges de données
  - *AC-Link* norme audio

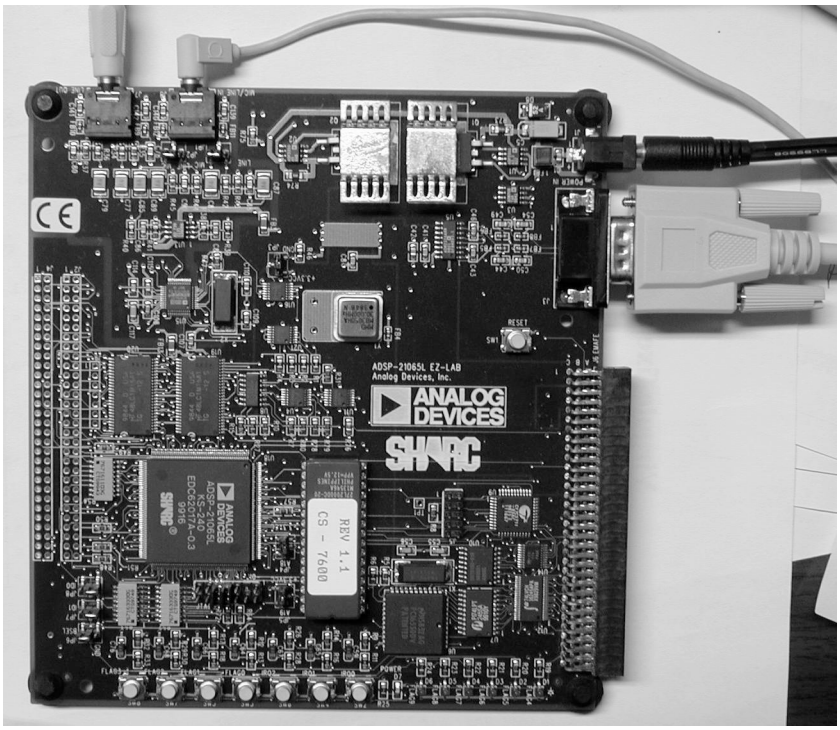


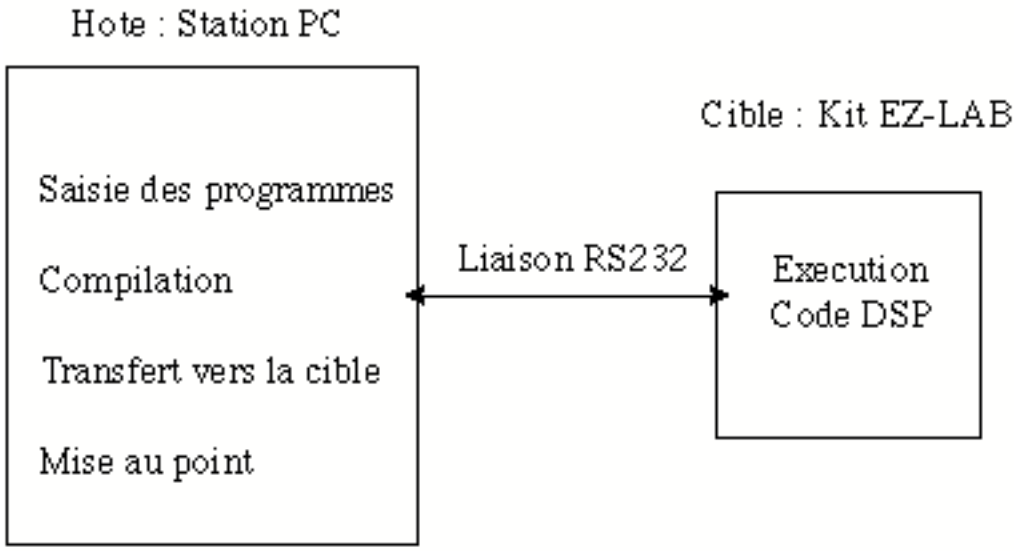
Kit de développement EZLAB

- Carte EZ-LAB
  - DSP ADSP-21065L
  - Codec AD1819A
  - Interface RS-232 (vers PC)
  - EPROM 1M * 8 bits
  - SDRAM 1M * 32 bits



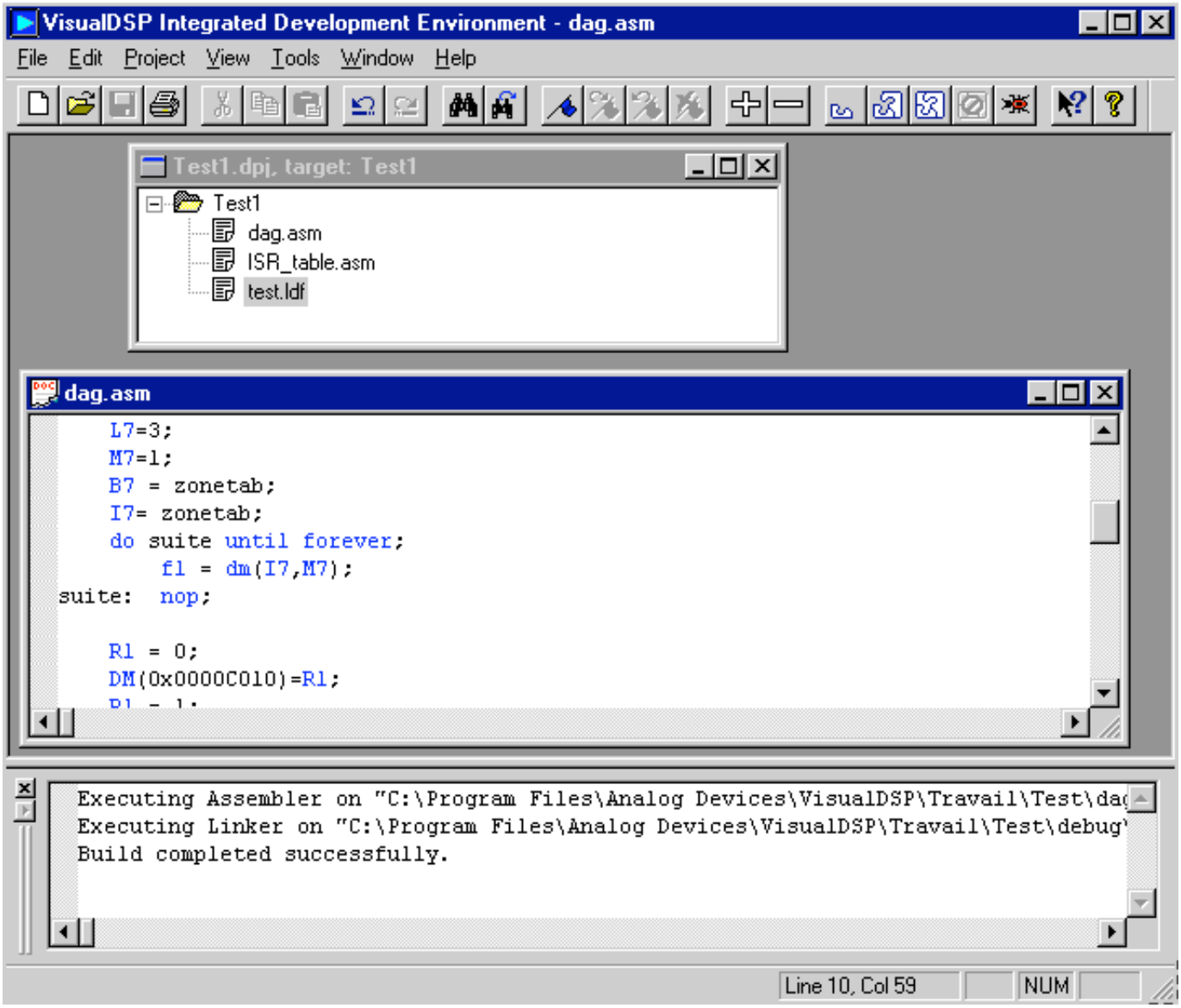
Environnement de développement





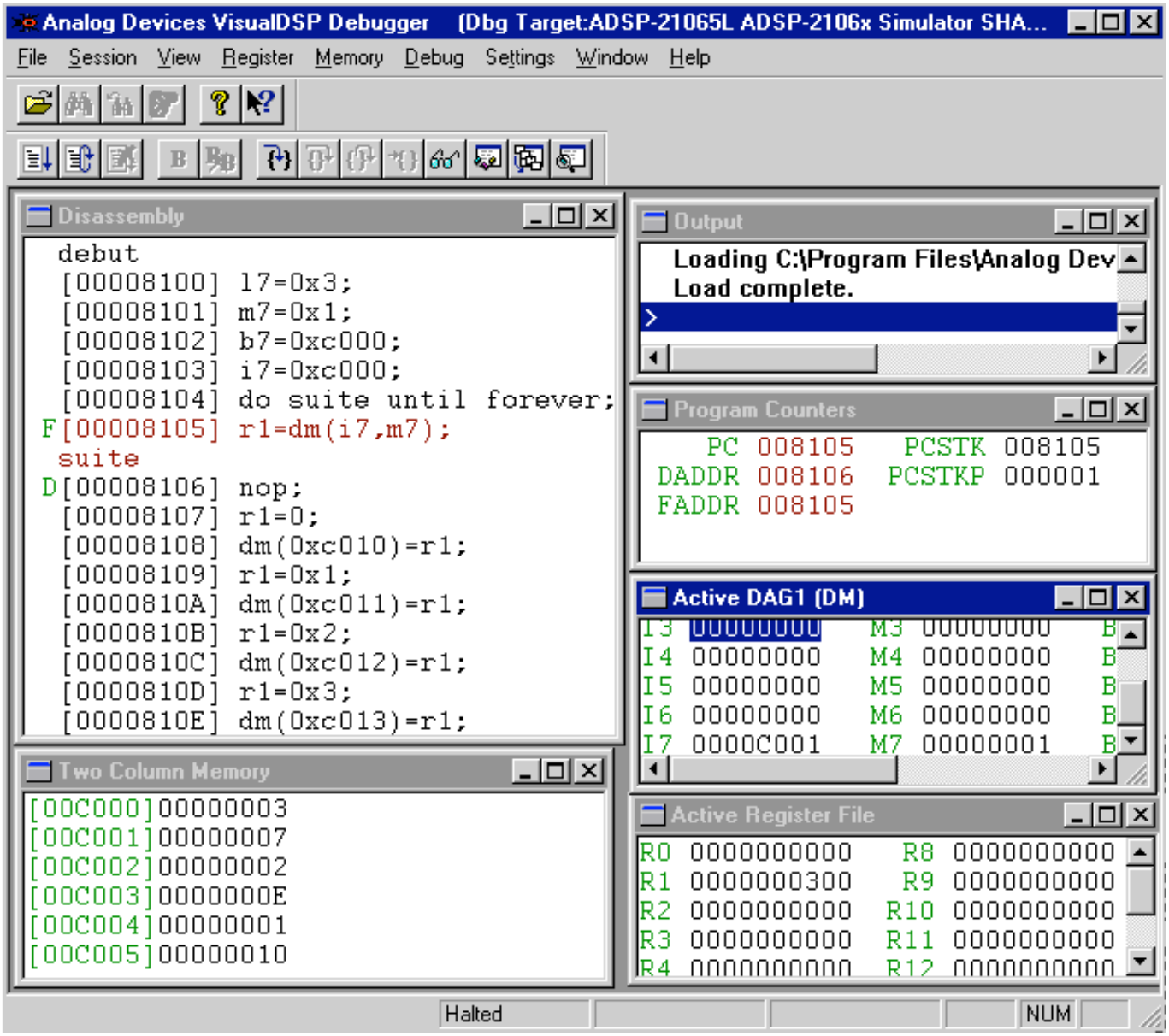
Gestion du projet

- Edition des programmes sources en assembleur
- Compilation
- Edition de liens



Debugger

- Permet la mise au point du programme
- Simulation ou contrôle de la platine EZ-Lab



Exemple de source assembleur

```
/* commentaire */
#include "def21065l.h"
#define N 10

.SEGMENT/DM     zonetab;
/* table de donnees a traiter */
.VAR table[N] = 3,7,2,14,1,16,18,12,28,22;
.ENDSEG;

.SEGMENT/PM mon_code;
debut: M1=0x01;          /* increment */

Do fin until sz ;
                r0 = bclr r0 by 0x01;     /* r0 flag = 0*/
                I1 = table;                                  /* initialise a case 1 */
                LCNTR=N-1, Do suite until LCE;
                r1=DM(I1,M1);
     r2=DM(0x00,I1);
                COMP(R2,R1);              /* compare la case I et I+1 */
                If GE jump suite;         /* si <= on saute le swap */
                DM(0xFFFFFFFF,I1)=r2;
                DM(0x00,I1)=r1;           /* on permutte */
                r0 = bset r0 by 0x01;     /* flag = 1 */
                suite: nop;
fin: Btst r0 by 0x01;       /* test de flag */
                nop;
                /* c'est simple non ??? */
.ENDSEG;
```

Exemple de fichier de liens

```
ARCHITECTURE(ADSP-21065L)

SEARCH_DIR( $ADI_DSP\21k\lib )

$LIBS = lib060.dlb;

// Libraries from the command line are included in COMMAND_LINE_OBJECTS.
$OBJECTS =  $COMMAND_LINE_OBJECTS;


MEMORY
{
        mon_code { TYPE(PM RAM) START(0x00008100)
                                END(0x000081ff) WIDTH(48) }
        zonetab  { TYPE(DM RAM) START(0x0000C000)
                                END(0x0000Dfff) WIDTH(32) }
        isr_tabl       { TYPE(PM RAM) START(0x00008005)
                                END(0x0000807f) WIDTH(48) }
}
PROCESSOR p0
{
   LINK_AGAINST( $COMMAND_LINE_LINK_AGAINST )
        OUTPUT ( $COMMAND_LINE_OUTPUT_FILE )

   SECTIONS
   {
        .mon_code { INPUT_SECTIONS( $OBJECTS(mon_code)
                                $LIBS(mon_code) )} >mon_code
        .ma_tab { INPUT_SECTIONS( $OBJECTS(zonetab)
                                $LIBS(zonetab) )} >zonetab
        .isr_tabl{ INPUT_SECTIONS( $OBJECTS(isr_tbl)
                                $LIBS(isr_tbl) )} >isr_tabl

   }
}
```