République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

**Ecole Nationale Polytechnique**
Département d'Electronique


# Mémoire de Fin d'Etudes pour l'Obtention du Diplôme d'Ingénieur d'Etat en Electronique

# Mise en œuvre d'une carte DSP TMS320C30

*Proposé par :*

Pr. D. Berkani

*Réalisé par :*

Ahmed Saïd Aziz

1999/2000

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ECOLE NATIONALE POLYTECHNIQUE



Département de Génie Electrique et Informatique

PROJET DE FIN D'ETUDE

# Mise en oeuvre d'une carte DSP TMS320C30

Proposé et dirigé par :
Le Pr. D.BERKANI

Fait par :
Mr. Ahmed Said Aziz

Année universitaire 1999/2000

# Remerciements



Je remercie mes parents pour tout leur bien et toute leur patience de m'avoir permis d'arriver à ce niveau.

Je remercie mon promoteur qui m'a donné accès à cette connaissance.

Je remercie tous les enseignants qui m'ont appris ce que je sais.

Et je remercie tous mes amis qui m'ont aidé.

**ملخص :**

هذا العمل هدفه تشغيل بطاقة تقييم الحاسوب TMS320C30، تبيان طرق برمجتها وتسهيل استعمالها بتوفير مركبات قابلة للاستعمال مكتوبة بـــ C.

نبدأ هذا البحث بمقدمة قصيرة حول "حواسيب معالجة الإشارة" DSP (مأخوذة من المرجع [7])، ثم نتحدث حول الهندسة الداخلية وكيفيات العنونة لـــ TMS320C30 [1]، وفي الفصل الرابع نقدم البطاقة EVM [2]، [4] و [8]. في الفصل الخامس نتطرق إلى برامج التطوير التي توفرها Texas Instruments [3]، [5] و [6]، ثم في الفصل السادس نعرض تقنيات برمجة البطاقة.

في الأخير ، نتم بخاتمة عامة.

**RESUME :**

Ce travail a pour but de mettre en œuvre la carte d'évaluation du TMS320C30 (EVM), de montrer les méthodes de programmation de cette carte et de faciliter son utilisation grâce à des routines réutilisables en C.

Nous commençons notre exposé par une brève introduction sur les DSP (tirée de la référence [7]), puis nous entamons l'architecture interne et les modes d'adressage du TMS320C30 [1], ce n'est qu'au chapitre IV que nous présentons l'EVM [2], [4] et [8]. Au chapitre V nous parlons des logiciels de développement fournis par Texas Instruments [3], [5] et [6], et au chapitre VI nous montrons comment programmer la carte. En fin, nous terminons par une conclusion générale.

**SUMMARY :**

The target of this work is to operate the TMS320C30 evaluation module, to show the programmation methods of this card and to facilitate it's use with given C routines.

We start with a short introduction to DSPs (taken from reference [7]), then we talk about the internal architecture and addressing modes of the TMS320C30 [1], it's only at the chapter IV that we present the EVM board [2], [4] & [6]. In the chapter V we show the software development tools provide by Texas Instruments [3], [5] & [6], and in the chapter VI we explain how to program the EVM. Finally we finish with a general conclusion.

**MOTS CLES:**

DSP,TMS320C30,EVM,AIC,TLC32044.

# Sommaire

**I- INTRODUCTION** ..... 2
I-1-DOMAINES D'APPLICATION DES DSP ..... 2
*Dans les télécommunications* ..... 2
*Dans le domaine militaire* ..... 2
*Dans le domaine du multimédia et du grand public* ..... 2
*Dans le domaine médical* ..... 2
*Dans le domaine de l'automatique* ..... 3
*Dans le domaine de l'instrumentation* ..... 3
I-2-LES FAMILLES DE DSP DE TEXAS INSTRUMENTS ..... 3
*Les cinq premières générations de DSP* ..... 3
*Les processeurs pour applications spécifiques* ..... 4
*Les DSP sur mesure* ..... 4
*Les processeurs de traitement de signal audio* ..... 4
*Les multiprocesseurs MVP (Multimedia Video Processor)* ..... 4
**II- ARCHITECTURE DES DSP DE LA FAMILLE TMS320C3X** ..... 7
II-1-DIAGRAMME BLOC DU C30 ..... 8
II-2-LE CPU ..... 8
*II-2-1-Le Multiplieur* ..... 8
*II-2-2-L'Unité Arithmétique et Logique (ALU)* ..... 9
*II-2-3-L'Unité Arithmétique et Logique de registres auxiliaires(ARAUs :Auxilary Register Arithmetic Units)* ..... 9
*II-2-4-Les Bus* ..... 9
*II-2-5-Les Registres* ..... 9
II-3- LES PÉRIPHÉRIQUES ..... 15
*II-3-1-le timer* ..... 15
*II-3-2-Le port série* ..... 18
II-4-TMS320C30 MÉMORY MAPS ..... 26
**III- LES MODES D'ADRESSAGES** ..... 30
III-1-ADRESSAGE PAR REGISTRE ..... 30
III-2-ADRESSAGE DIRECT ..... 30
III-3-ADRESSAGE IMMÉDIAT ..... 30
III-4-ADRESSAGE RELATIF PC ..... 30
III-5-ADRESSAGE INDIRECT ..... 30
III-6-ADRESSAGE CIRCULAIRE ..... 32
*Principe* ..... 32
*Règles à respecter* ..... 32
*Application* ..... 32
III-7-ADRESSAGE PAR INVERSION DE BITS ..... 32
*Calcul d'adresse* ..... 33
**IV- LE MODULE D'ÉVALUATION EVM** ..... 35
IV-1- DESCRIPTION GENERALE ..... 35
IV-2- DIAGRAMME BLOC DE L'EVM ..... 35
IV-3-SWITCH ET CONFIGURATION MÉMOIRE ..... 35
IV-4-LES REGISTRES DE CONTRÔLE DE L'EVM ..... 36
IV-5-LE TEST BUS CONTROLER (TBC) ..... 37
*Les événements du TBC* ..... 37
IV-6-LA SYNCHRONISATION DE TRANSFERT DE DONNÉES ..... 37
*Etapes pour une ecriture synchrone* ..... 37
*Etapes pour une lecture synchrone* ..... 38
IV-7-ACCÈS DU DSP AU REGISTRE DE COMMUNICATION ..... 38
IV-8-L'INTÉRFACE ANALOGIQUE AIC ..... 38
*IV-8-1-Diagramme bloc de l'AIC* ..... 40
*IV-8-2-Brochage du circuit et description des signaux* ..... 40
*IV-8-3-principe de fonctionnement* ..... 42
*IV-8-4-programmation de l'AIC* ..... 43
**V-LES OUTILS DE DÉVELOPPEMENT** ..... 48
V-1-L'ASSEMBLEUR ..... 48
*Ligne de commande* ..... 48
*Les options les plus usuelles* ..... 48

*instructions assembleur les plus utilisées* ..... 48
V-2-LE LIEUR ..... 49
*Ligne de commande* ..... 49
*Les options les plus usuelles* ..... 49
V-3-LE COMPILATEUR C ..... 49
*Ligne de commande* ..... 49
*Les options les plus usuelles* ..... 49
*Construction de librairies* ..... 49

**VI-APPLICATIONS** ..... **52**

VI-1-PROGRAMMATION DU TIMER0 ..... 52
VI-2-PROGRAMMATION DU PORT SÉRIE ..... 52
VI-3-PROGRAMMATION DE L'AIC ..... 53
VI-4-GÉNÉRATION D'UN COSINUS ..... 54
VI-5-NUMÉRISATION DE LA PAROLE ..... 57
VI-5-EFFETS SPÉCIAUX 'TRUQUEUR DE VOIE' ..... 57
VI-6-LE PROJET''OSCILLOSCOPE NUMÉRIQUE' ..... 58
*VI-6-1-Partie DSP* ..... 58
*VI-6-2-Partie PC* ..... 58
VI-7-UTILISATION DU COMPILATEUR C ..... 61
*VI-7-1-routine d'initialisation des périphériques* ..... 61
*VI-7-2-routines d'accès AIC et PC* ..... 62

**VII-CONCLUSION** ..... **65**

**BIBLIOGRAPHIE** ..... **67**

# Introduction

# I- INTRODUCTION

Les processeurs de traitement du signal ont vu leur utilisation se développer considérablement depuis 1985 et ce, avec l'évolution des télécommunications (MIC, GSM, etc.) ; actuellement de nombreux produits comportent un Digital Signal Processor (DSP).

Ces processeurs particuliers sont assez semblables aux microprocesseurs et leur utilisation en est proche comme nous le verrons. Leur particularité essentielle est qu'ils sont conçus pour effectuer des calculs en temps réel et donc intègrent des opérateurs (multiplieur, ALU, registres, etc.), ils permettent un accès rapide aux données par des adressages particuliers et de nombreux bus d'accès. Ils se programment essentiellement en assembleur ou en langage C et leur jeu d'instructions est réduit.

## I-1-DOMAINES D'APPLICATION DES DSP

A l'origine, les deux principaux domaines d'application des DSP ont été les télécommunications et le secteur militaire. Aujourd'hui, les applications se sont diversifiées et s'orientent vers le multimédia, l'électronique grand public et l'informatique graphique mais aussi l'automatique, le domaine médical (traitement et archivage d'images, analyse de signaux électrocardiographiques, implants cochléaires), l'instrumentation, l'électronique automobile (détection de cliquetis, système ABS).

### DANS LES TELECOMMUNICATIONS

- La téléphonie filaire : les modems, les codeurs de parole de type ADPCM, les multiplexeurs, les récepteurs de numérotation DTMF, les annuleurs d'écho, les télécopieurs, les Minitels intelligents.
- La radiotéléphonie : téléphone cellulaire, téléphone sans fil (les codeurs de parole GSM, les modems radio).

### DANS LE DOMAINE MILITAIRE

Le guidage de missiles, la navigation, les modems radio, les communications cryptées, le traitement sonar, le traitement radar.

### DANS LE DOMAINE DU MULTIMEDIA ET DU GRAND PUBLIC

- En plus du traitement de la parole (compression, reconnaissance et synthèse), la compression de signaux audio de qualité HI-FI à des débits assez faibles pour leur transmission et leur stockage économique.
- La compression des images fixes ou animées.
- La radiodiffusion sonore numérique.
- Les cartes multimédia pour PC ou station de travail.
- La synthèse musicale.
- Les jeux.

### DANS LE DOMAINE MEDICAL

- La compression d'image médicale en vue de leur archivage.
- Le traitement de signaux biophysique ECG, EEG.
- Les implants cochléaires.
- Les équipements de *monitoring*.

### DANS LE DOMAINE DE L'AUTOMATIQUE

- La surveillance et la commande de machine.
- Le contrôle des moteurs.
- Les robots.
- Les servomécanismes.

### DANS LE DOMAINE DE L'INSTRUMENTATION

Les analyseurs de spectre.
Les générateurs de fonctions.
L'interprétation des signaux sismiques.

Le tableau suivant donne un ordre d'idée des temps nécessaires pour les algorithmes :

| Circuit | FIR 64 coefficients | IIR cellules biquadratiques | FFT, mémoire à mémoire 256/1024 pts |
|---|---|---|---|
| TMS320c1x<br>$t_c$ =114 ns | 133 $t_c$<br>$f_{emax}$=66 khz | 56 $t_c$<br>$f_{emax}$=157 khz | 4.73 ms<br>37.8 ms |
| TMS320c2x<br>$t_c$ =80 ns | 73 $t_c$<br>$f_{emax}$=172 khz | 43 $t_c$<br>$f_{emax}$=284 khz | 1.41 ms<br>9 ms |
| TMS320c3x<br>$t_c$ =50 ns | 69 $t_c$<br>$f_{emax}$=290 khz | 27 $t_c$<br>$f_{emax}$=741 khz | 0.45 ms<br>1.97 ms |
| TMS320c4x<br>$t_c$=40 ns | 68 $t_c$<br>$f_{emax}$=368 khz | 27 $t_c$<br>$f_{emax}$=929 khz | 0.30 ms<br>1.54 ms |
| TMS320c5x<br>$t_c$ =35 ns | 71 $t_c$<br>$f_{emax}$=391 khz | 43 $t_c$<br>$f_{emax}$=664 khz | 0.59 ms<br>2.89 ms |

## I-2-LES FAMILLES DE DSP DE TEXAS INSTRUMENTS

### LES CINQ PREMIERES GENERATIONS DE DSP

En 1982, TI a lancé son premier DSP : le TMS32010. Depuis, Texas s'est imposé comme le principal fabricant de processeurs de traitements de signal. La famille de DSP TMS320 est organisée aujourd'hui en cinq générations principales qui correspondent chacune à une classe de performances et d'applications. elles sont représentées par les sigles suivants :

- TMS320C1X.
- TMS320C2X.
- TMS320C3X.
- TMS320C4X.
- TMS320C5X.

Les processeurs de génération 1,2 et 5 travaillent en virgule fixe, ceux de la troisième et cinquième génération en virgule flottante. Une compatibilité logicielle (assembleur) ascendante est maintenue pour chaque famille. A ces cinq générations, s'ajoutent de nouveaux composants comme le TMS320C80, destiné aux applications multimédia et vidéo.

La technologie NMOS 2.4 µm utilisée à l'origine a été abandonnée pour une technologie CMOS inférieure à 1 µm, dont les performances s'accroissent régulièrement. Des versions militaires ou spatiales existent pour certains de ces composants.

Dans la dénomination de ces DSP :
- Le caractère C signifie CMOS,
- Le caractère E signifie EPROM (remplace la ROM sur la puce),
- Le caractère P indique une mémoire OTP,
- Un tiret suivi d'un nombre représente une fréquence d'horloge (par exemple, le TMS320c15-25 a une fréquence de 25 Mhz),
- Les caractères LC désignent une faible consommation (*low consumption*).

### LES PROCESSEURS POUR APPLICATIONS SPECIFIQUES

Le TMS320SA32 est un TMS320C10 masqué qui réalise un codeur de parole ADPCM à 32 Kb/s compatible ANSI et CCITT G721.

Le TMS320G722EC est un TMS320C5X masqué pour différentes applications de téléphonie :
- annulation d'écho,
- codage vocal (norme G722 ADPCM),
- générateur DTMF,
- générateur de sonnerie,
etc.

Le MSP58C80 est un composant contenant un cœur de TMS320C25. Ce composant s'interface au convertisseur analogique numérique MSP58C20 et à de la mémoire RAM pour former la base d'un répondeur enregistreur téléphonique.

### LES DSP SUR MESURE

TI a porté les processeurs TMS320C10, C15, C16, C25, C5X dans la bibliothèque de cellules standard TSC700 avec des cellules analogiques (convertisseurs analogiques numériques, numériques analogiques, VCO, amplificateurs opérationnels, etc.).
En 1993, Texas a introduit le TEC320C25A. Ce DSP comporte un C25, 1Kmots de RAM, une logique d'émulation, une PLL et 15 Ko de portes logiques. Son temps d'horloge est de 60 Mhz.
TI prévoit d'enrichir cette librairie en proposant d'autres cœurs de DSP (C5X, C3X) comportant plus de portes logiques ainsi que des convertisseurs analogiques numériques et des composants 3.3v.

### LES PROCESSEURS DE TRAITEMENT DE SIGNAL AUDIO

Texas propose une famille de DSP pour les applications audio (DASP, *Digital Audio Signal Processor*), la famille TMS5700XX.
Les applications audio nécessitent une grande dynamique, une grande précision et un faible coût.
La famille comprend trois processeurs 24 bits, un processeur 32 bits et un convertisseur numérique/analogique incluant un circuit de filtrage numérique.

### LES MULTIPROCESSEURS MVP (MULTIMEDIA VIDEO PROCESSOR)

Le TMS320C80 est le premier composant multiprocesseur. Il est destiné aux applications multimédia, particulièrement à la compression-décompression et au traitement vidéo. Il comprend quatre DSP 32 bits optimisés pour le traitement d'images et un processeur RISC maître 32 bits

virgule flottante. Une matrice de commutation permet le partage de la mémoire entre les cinq processeurs. En plus des cinq processeurs, le circuit dispose d'un contrôleur de transfert permettant un débit de 400 Mbytes par seconde, deux contrôleurs vidéo et une RAM statique interne de 50 Kbytes.
Le circuit effectue environ deux milliards d'opération élémentaires par seconde.

# Architecture des DSPs de la famille TMS320C3X

# II- ARCHITECTURE DES DSP LA FAMILLE TMS320C3X

L'architecture des C3X répond aux exigences de systèmes basés sur des algorithmes arithmétiques sophistiqués demandant des solutions aussi bien hardware que software.
De très hautes performances sont réalisées par les DSP de cette famille grâce à leur unité de calcul en virgule flottante, une large mémoire interne et un haut degré de parallélisme.

Diagramme bloc des DSP de la famille

Tms320C3x

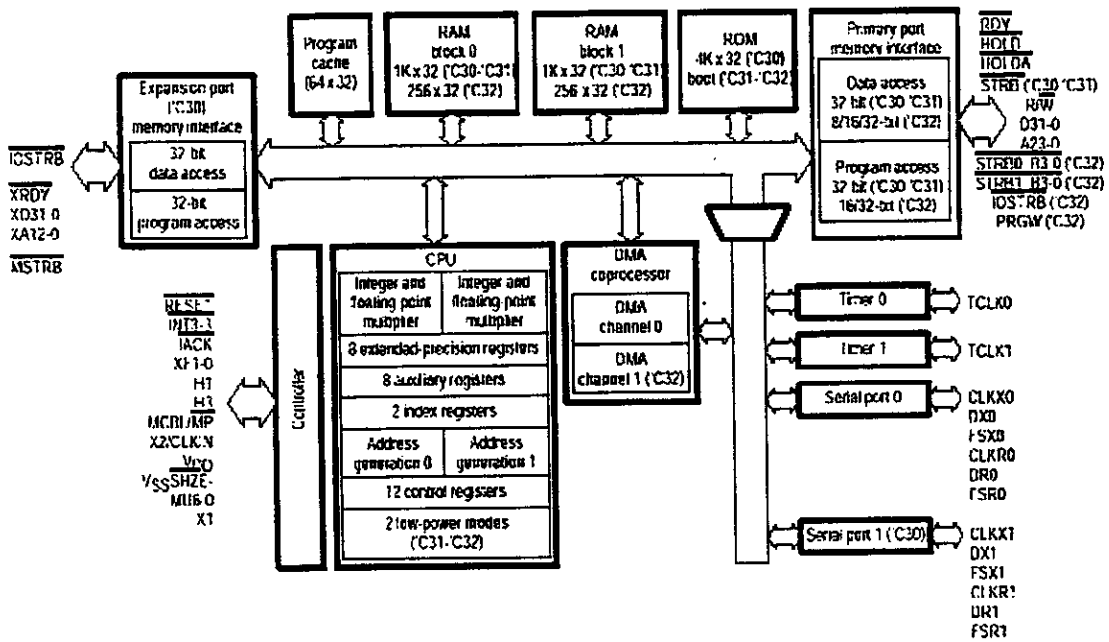



Figure II-1

Comme il n'y a pas beaucoup de différence entre les DPS de cette famille (figure II-1), nous n'allons étudier que le TMS320C30.

# II-1-DIAGRAMME BLOC DU C30

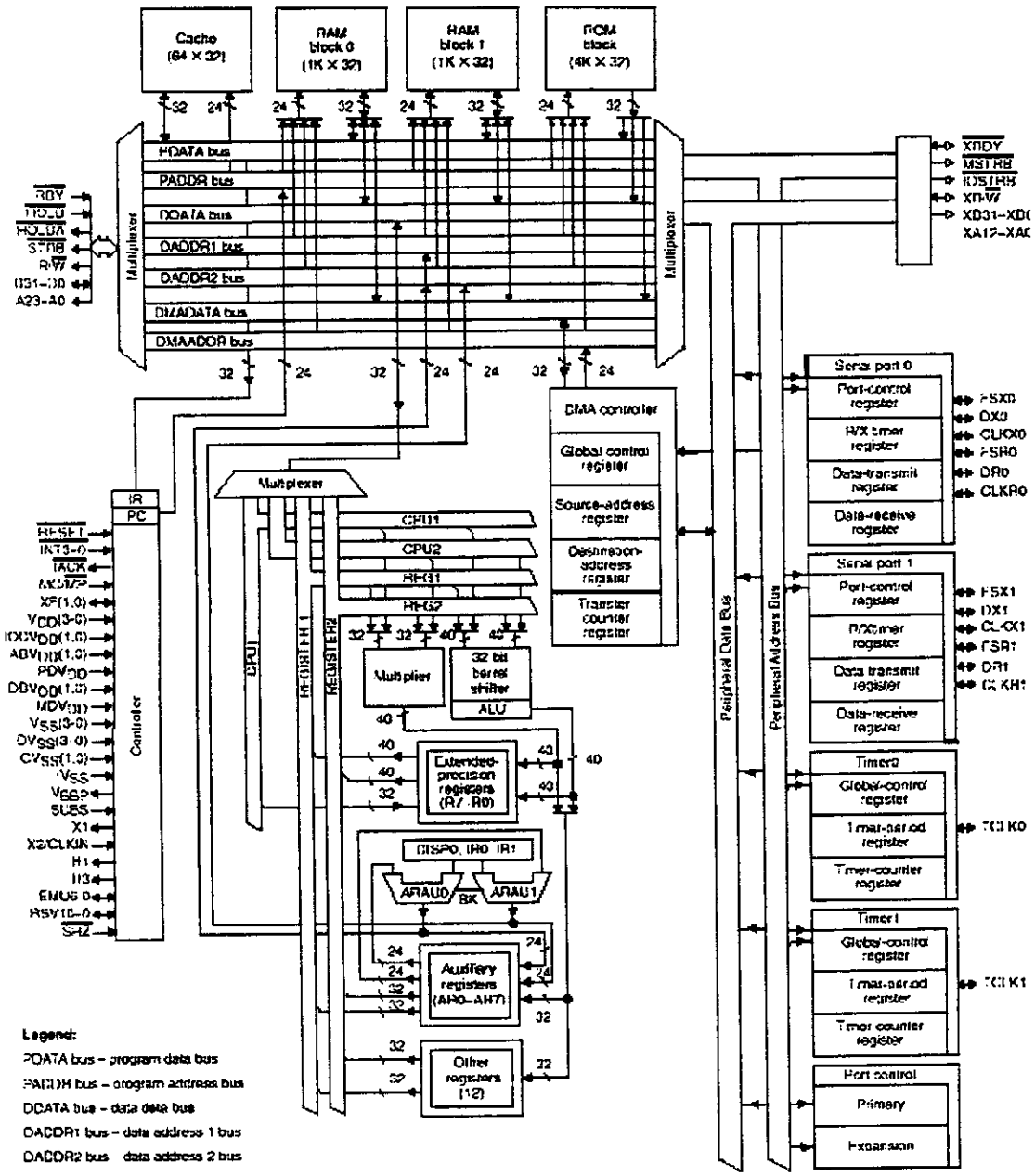



Figure II-2

# II-2-LA CPU

La CPU consiste en (Figure II-2) :

## II-2-1-LE MULTIPLIEUR

Le multiplieur réalise une multiplication en virgule fixe ou flottante en un cycle machine. En virgule fixe l'entrée est deux opérandes sur 24 bits et la sortie est un entier sur 32 bits. En virgule flottante l'entrée est deux opérandes en virgule flottante sur 32 bits et la sortie est un nombre en virgule flottante sur 40 bits. Le multiplieur peut travailler en parallèle avec l'ALU pour faire une multiplication et une opération ALU en même temps.

### II-2-2-L'UNITE ARITHMETIQUE ET LOGIQUE (ALU)

L'ALU peut faire des opérations arithmétiques sur des entiers de 32 bits ou des nombres en virgule flottante de 40 bits, des opérations logiques sur des entiers de 32 bits et des conversions de format entre les données entières et en virgule flottante. Le résultat est toujours maintenu sur 32 ou 40 bits. Toutes les opérations se font en un cycle machine.

### II-2-3-L'UNITE ARITHMETIQUE ET LOGIQUE DE REGISTRES AUXILIAIRES(ARAUS :*AUXILARY REGISTER ARITHMETIC UNITS)*

Deux ARAUs sont utilisées pour générer deux adresses en un cycle machine. Les ARAUs travaillent en parallèle avec l'ALU et le multiplieur, elles supportent plusieurs modes d'adressage dont un spécialement dédié au filtrage (*circular addressing mode*) et un autre à l'FFT (*bit reversed addressing mode*).

### II-2-4-LES BUS

De multiples bus permettent la réalisation d'opérations en parallèles et des accès simultanés aux données:
- Trois bus de données;
- Deux bus programme;
- Deux bus DMA ;
- Deux bus pour les périphériques.

Le CPU a quatre bus: CPU1 ,CPU2 ,REG1 ,REG2 .

### II-2-5-LES REGISTRES

Le C30 à vingt-huit registres (Table II-1) de 32 bits (sauf pour les registres Rn).

Table II-1

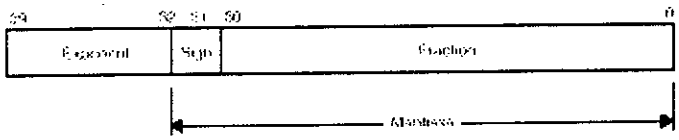
| Symbol | Value (hex) | Assigned Function Name |
|---|---|---|
| R0 | 00 | Extended-precision register 0 |
| R1 | 01 | Extended-precision register 1 |
| R2 | 02 | Extended-precision register 2 |
| R3 | 03 | Extended-precision register 3 |
| R4 | 04 | Extended-precision register 4 |
| R5 | 05 | Extended-precision register 5 |
| R6 | 06 | Extended-precision register 6 |
| R7 | 07 | Extended-precision register 7 |
| AR0 | 08 | Auxiliary register 0 |
| AR1 | 09 | Auxiliary register 1 |
| AR2 | 0A | Auxiliary register 2 |
| AR3 | 0B | Auxiliary register 3 |
| AR4 | 0C | Auxiliary register 4 |
| AR5 | 0D | Auxiliary register 5 |
| AR6 | 0E | Auxiliary register 6 |
| AR7 | 0F | Auxiliary register 7 |
| DP | 10 | Data page pointer |
| IR0 | 11 | Index register 0 |
| IR1 | 12 | Index register 1 |
| BK | 13 | Block-size register |

Suite de la table II-1 :

| | | |
|---|---|---|
| SP | 14 | System-stack pointer |
| ST | 15 | Status register |
| IE | 16 | CPU/DMA interrupt-enable |
| IF | 17 | CPU interrupt flags |
| IOF | 18 | I/O flags |
| RS | 19 | Repeat start-address |
| RE | 1A | Repeat end-address |
| RC | 1B | Repeat counter |

**Les registres à précision étendue (R0-R7)**
Ce sont des registres 40 bits (les seuls), Ils sont dédiés aux calculs arithmétiques. Utilisent 32 bits pour les entiers, et 40 pour le format virgule flottante (Figure II-3).

Format virgule flottante



Format virgule fixe

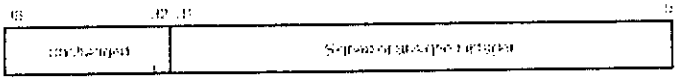


Figure II-3

**Les registres auxiliaires (AR0-AR7)**
Leur fonction principale est la génération d'adresse 24 bits. Ils peuvent être utilisés comme registres généraux 32 bits que l'ALU ou le multiplieur peuvent modifier.

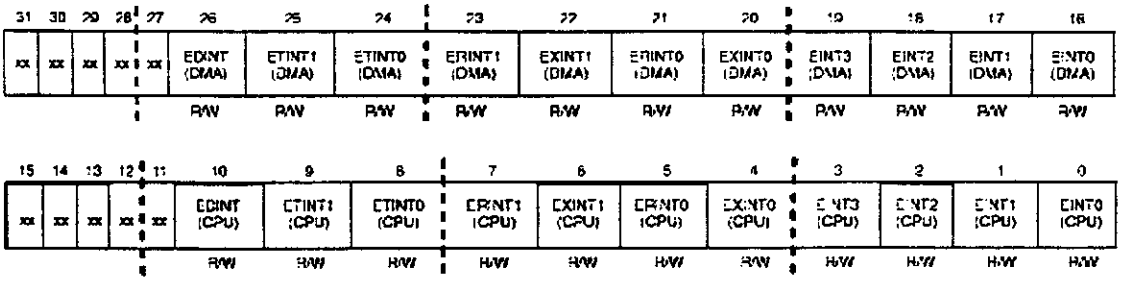
**Le pointeur de pages de données (DP)**
Les bits de 8 à 31 sont réservés. Les bits de 0 à 7 sélectionnent la page d'adresses mémoire (256 pages de 64 kmots).

**Les registres d'index (IR0-IR1)**
Utilisés par les ARAUs pour l'indexage d'adresses (voir modes d'adressages chapitre III).

**Registre taille de bloc (BK)**
Utilisé par les ARAUs pour l'adressage circulaire.

**Pointeur de pile système (SP)**
Pointe toujours sur le dernier élément mis en pile. Il est utilisé dans les interruptions et appel de sous programme.

**Le registre d'interruption CPU/DMA (IE)**
Ce registre permet le contrôle de toutes les interruptions CPU et DMA soit en les activant (mettre un 1 dans le bit qu'il faut) ou en les désactivant (par un zéro).




**Notes:** 1) xx = reserved bit, read as 0
2) R = read, W = write

Figure II-4

La table II-2 montre la signification de chaque bit :

Table II-2

| Abbreviation | Reset Value | Description |
|---|---|---|
| EINT0 (CPU) | 0 | CPU external interrupt 0 enable |
| EINT1 (CPU) | 0 | CPU external interrupt 1 enable |
| EINT2 (CPU) | 0 | CPU external interrupt 2 enable |
| EINT3 (CPU) | 0 | CPU external interrupt 3 enable |
| EXINT0 (CPU) | 0 | CPU serial port 0 transmit interrupt enable |
| ERINT0 (CPU) | 0 | CPU serial port 0 receive interrupt enable |
| EXINT1 (CPU) | 0 | CPU serial port 1 transmit interrupt enable |
| ERINT1 (CPU) | 0 | CPU serial port 1 receive interrupt enable |
| ETINT0 (CPU) | 0 | CPU timer0 interrupt enable |
| ETINT1 (CPU) | 0 | CPU timer1 interrupt enable |
| EDINT (CPU) | 0 | CPU DMA controller interrupt enable |
| EINT0 (DMA) | 0 | DMA external interrupt 0 enable |
| EINT1 (DMA) | 0 | DMA external interrupt 1 enable |
| EINT2 (DMA) | 0 | DMA external interrupt 2 enable |
| EINT3 (DMA) | 0 | DMA external interrupt 3 enable |
| EXINT0 (DMA) | 0 | DMA serial port 0 transmit interrupt enable |
| ERINT0 (DMA) | 0 | DMA serial port 0 receive interrupt enable |
| EXINT1 (DMA) | 0 | DMA serial port 1 transmit interrupt enable |
| ERINT1 (DMA) | 0 | DMA serial port 1 receive interrupt enable |
| ETINT0 (DMA) | 0 | DMA timer0 interrupt enable |
| ETINT1 (DMA) | 0 | DMA timer1 interrupt enable |
| EDINT (DMA) | 0 | DMA controller interrupt enable |

**Le registre indicateur d'interruption CPU (IF)**
Les bits de ce registre montre l'état des interruptions correspondantes.

| 31–16 | 15–12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| xx | yy | yy | DINT | TINT1 | TINT0 | RINT1 | XINT1 | RINT0 | XINT0 | INT3 | INT2 | INT1 | INT0 |
| | | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**Notes:** 1) xx = reserved bit, read as 0
2) yy = reserved bit, set to 0 at reset; can store value
3) R = read, W = write

Figure II-5

La table II-3 montre la signification de chaque bit :

Table II-3

| Bit Name | Reset Value | Function |
|---|---|---|
| INT0 | 0 | External interrupt 0 flag |
| INT1 | 0 | External interrupt 1 flag |
| INT2 | 0 | External interrupt 2 flag |
| INT3 | 0 | External interrupt 3 flag |
| XINT0 | 0 | Serial port 0 transmit flag |
| RINT0 | 0 | Serial port 0 receive flag |
| XINT1 | 0 | Serial port 1 transmit flag |
| RINT1 | 0 | Serial port 1 receive interrupt flag |
| TINT0 | 0 | Timer 0 interrupt flag |
| TINT1 | 0 | Timer 1 interrupt flag |
| DINT | 0 | DMA channel interrupt flag |

**Le registre indicateur d'E/S (IOF)**
Ce registre contrôle la fonction des deux broches d'E/S XF0 et XF1.

| 31–16 | 15–12 | 11–8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| xx | x | xx | INXF1 | OUTXF1 | $\overline{I/O}$XF1 | xx | INXF0 | OUTXF0 | $\overline{I/O}$XF0 | xx |
| | | | R | R/W | R/W | | R | R/W | R/W | |

**Notes:** 1) xx = reserved bit, read as 0
2) R = read, W = write

Figure II-6

La table II-4 montre la signification de chaque bit :

Table II-4

| Bit Name | Reset Value | Function |
|---|---|---|
| $\overline{I}$/OXF0 | 0 | If 0, XF0 is configured a general-purpose input pin.<br>If 1, XF0 is configured a general-purpose output pin. |
| OUTXF0 | 0 | Data output on XF0. |
| INXF0 | 0 | Data input on XF0. A write has no effect. |
| $\overline{I}$/OXF1 | 0 | If 0, XF1 is configured a general-purpose input pin.<br>If 1, XF1 is configured a general-purpose output pin. |
| OUTXF1 | 0 | Data output on XF1. |
| INXF1 | 0 | Data input on XF1. A write has no effect. |

**Le registre d'état (ST)**

Le registre d'état est l'un des registres les plus importants, il contient des informations globales sur l'état de la CPU. Il est modifié par les opérations arithmétiques ou logiques et peut être chargé par n'importe quelle valeur quel que soit l'état de ses bits.

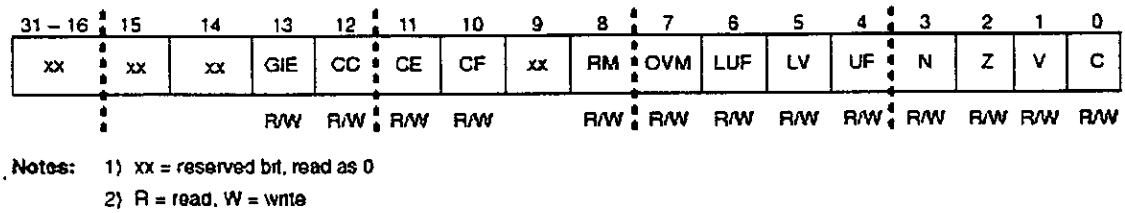



Figure II-7

La table II-5 montre la signification de chaque bit :

Table II-5

| Bit Name | Reset Value | Name | Description |
|---|---|---|---|
| C | 0 | Carry flag | Carry condition flag |
| V | 0 | Overflow flag | Overflow condition flag |
| Z | 0 | Zero flag | Zero condition flag |
| N | 0 | Negative flag | Negative condition flag |
| UF | 0 | Floating-point under-flow flag | Floating-point underflow condition flag |
| LV | 0 | Latched overflow flag | Latched overflow condition flag |
| LUF | 0 | Latched floating-point underflow flag | Latched floating-point underflow condition flag |

Suite de la table II-5:

| | | | |
|---|---|---|---|
| OVM | 0 | Overflow mode flag | Overflow mode flag<br>The overflow mode flag affects only integer operations.<br>If OVM = 0, the overflow mode is turned off and integer results that overflow are treated in no special way.<br>If OVM = 1, integer results overflowing in the positive direction are set to the most positive, 2s-complement number (7FFF FFFFh), and integer results overflowing in the negative direction are set to the most negative 32-bit, 2s complement number (8000 0000h). |
| RM | 0 | Repeat mode flag | Repeat mode flag<br>If RM = 1, the PC is modified in either the repeat block or repeat single mode. |
| CE | 0 | Cache enable | CE enables or disables the instruction cache.<br>Set CE = 1 to enable the cache, allowing the cache to be used according to the least recently used (LRU) stack manipulation.<br>Set CE = 0 to disable the cache, preventing cache updates or modifications (no cache fetches can be made). Cache clearing (CC = 1) is allowed when CE = 0. |
| CF | 0 | Cache freeze | Enables or disables the instruction cache<br>Set CF = 1 to freeze the cache (cache is not updated), including LRU stack manipulation. If the cache is enabled (CE = 1), fetches from the cache are allowed, but modification of the cache contents is not allowed. Cache clearing (CC = 1) is allowed. At reset, this bit is cleared to 0, but it is set to 1 after reset.<br>When CF = 0, the cache is automatically updated by instruction fetches from external memory. Also, when CF = 0, cache clearing (CC = 1) is allowed.<br>The following table summarizes the CE and CF bits:<br>**CE** **CF** **Effect**<br>0 0 Cache not enabled<br>0 1 Cache not enabled<br>1 0 Cache enabled and not frozen<br>1 1 Cache enabled but frozen (cache read only) |
| CC | 0 | Cache clear | CC = 1 invalidates all entries in the cache. This bit is always cleared after it is written to, and is always read as 0. At reset, 0 is written to this bit. |
| GIE | 0 | Global interrupt enable | If GIE = 1, the CPU responds to an enabled interrupt.<br>If GIE = 0, the CPU does not respond to an enabled interrupt. |

**Les registres de répétition de bloc (RC-RS-RE)**

Ces registres sont utilisés quand le CPU travaille dans le *repeat mode*. Le registre RC contient le nombre de fois qu'un bloc doit être répété, le registre RS contient l'adresse de début du bloc et le registre RE contient l'adresse de fin du bloc.

# II-3- LES PERIPHERIQUES

Les DSP de la famille TMS320C3X possèdent trois types de périphériques le Timer, le port série et le contrôleur DMA (voir Figure II-1). Le C30 a deux Timers, deux ports série et un contrôleur DMA.

## II-3-1-LE TIMER

**Diagramme bloc du Timer**

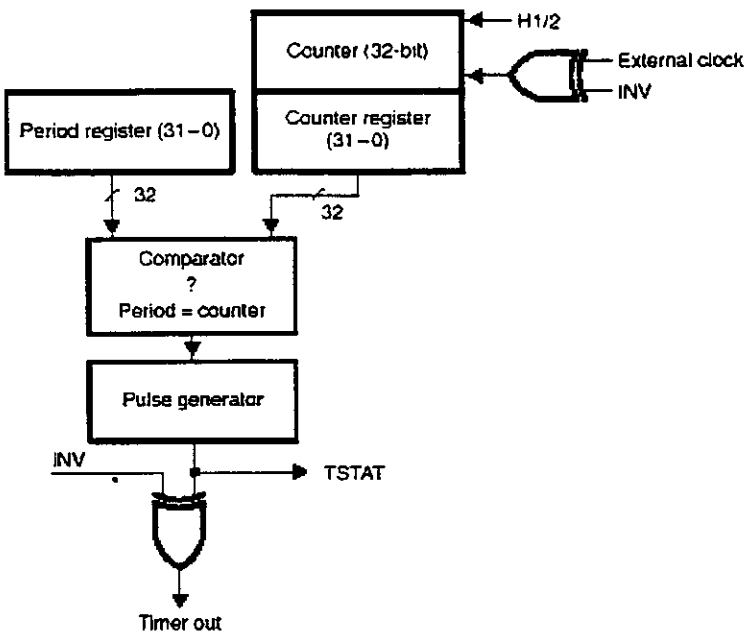



Figure II-8

Le Timer a une seule broche TCLK. Elle peut être utilisée comme une E/S d'horloge ou d'impulsion, ou comme une broche d'E/S générale. Le Timer est contrôlé grâce à trois registres qui déterminent son mode de fonctionnement.

### Les registres du Timer

**Global-control register**
Il détermine le mode de fonctionnement du Timer, contrôle la fonction de TCLK et renseigne sur l'état du Timer.

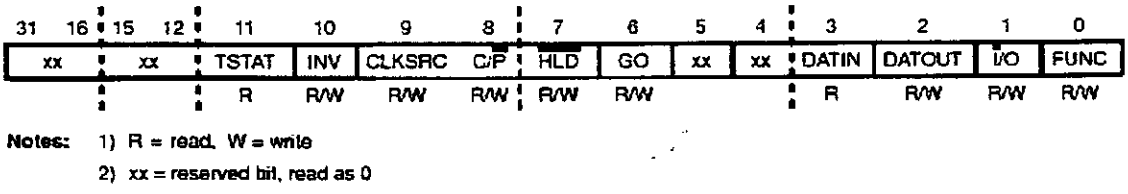



Figure II-9

La table II-6 montre la signification de chaque bit :

Table II-6

| Abbreviation | Reset Value | Name | Description |
|---|---|---|---|
| FUNC | 0 | Function | Controls the function of TCLK.<br>If FUNC = 0, TCLK is configured as a general-purpose digital I/O port.<br>If FUNC = 1, TCLK is configured as a timer pin. |
| $\overline{I}$/O | 0 | Input/output | If FUNC = 0 and CLKSRC = 0, TCLK is configured as a general-purpose I/O pin.<br>If $\overline{I}$/O = 0, TCLK is configured as a general-purpose input pin.<br>If $\overline{I}$/O = 1, TCLK is configured as a general-purpose output pin. |
| DATOUT | 0 | Data output | Drives TCLK when the 'C3x is in I/O port mode. You can use DATOUT as an input to the timer. |
| DATIN | x† | Data input | Data input on TCLK or DATOUT. A write has no effect. |
| GO | 0 | Go | Resets and starts the timer counter.<br>When GO = 1 and the timer is not held, the counter is zeroed and begins incrementing on the next rising edge of the timer input clock. The GO bit is cleared on the same rising edge.<br>GO = 0 has no effect on the timer. |
| $\overline{HLD}$ | 0 | Counter hold signal | When this bit is 0, the counter is disabled and held in its current state. If the timer is driving TCLK, the state of TCLK is also held.<br>The internal divide-by-2 counter is also held so that the counter can continue where it left off when $\overline{HLD}$ is set to 1.<br>You can read and modify the timer registers while the timer is being held. $\overline{RESET}$ has priority over $\overline{HLD}$. The effect of writing to GO and HOLD is shown below.<br>**GO** **$\overline{HLD}$** **Result**<br>0 0 All timer operations are held. No reset is performed (reset value).<br>0 1 Timer proceeds from state before write.<br>1 0 All timer operations are held, including zeroing of the counter. The GO bit is not cleared until the timer is taken out of hold.<br>1 1 Timer resets and starts. |
| C/$\overline{P}$ | 0 | Clock/pulse mode control | When C/$\overline{P}$ = 1, clock mode is chosen, and the signaling of the TSTAT flag and external output has a 50% duty cycle.<br>When C/$\overline{P}$ = 0, the status flag and external output will be active for one H1 cycle during each timer period |
| CLKSRC | 0 | Clock source | This bit specifies the source of the timer clock.<br>When CLKSRC = 1, an internal clock with a frequency equal to one-half of the H1 frequency is used to increment the counter. The INV bit has no effect on the internal clock source.<br>When CLKSRC = 0, you can use an external signal from the TCLK pin to increment the counter. The external clock is synchronized internally, thus allowing external asynchronous clock sources that do not exceed the specified maximum allowable external clock frequency. This is less than f(H1)/2. |

Suite de la table II-6

| | | | |
|---|---|---|---|
| **INV** | 0 | Inverter control bit | If an external clock source is used and INV = 1, the external clock is inverted as it goes into the counter.<br>If the output of the pulse generator is routed to TCLK and INV = 1, the output is inverted before it goes to TCLK<br>If INV = 0, no inversion is performed on the input or output of the timer. The INV bit has no effect, regardless of its value, when TCLK is used in I/O port mode. |
| **TSTAT** | 0 | Timer status bit | This bit indicates the status of the timer. It tracks the output of the uninverted TCLK pin. This flag sets a CPU interrupt on a transition from 0 to 1. A write has no effect. |

† x = 0 or 1 (set to value read on TCLK)

Ainsi le bit FUNC détermine si TCLK est une E/S (FUNC=0) ou une broche Timer (FUNC=1), si FUNC=1 (mode Timer) le bit CLKSRC détermine la source de l'horloge du Timer qui peut être interne (CLKSRC=1, $f = f\mu p/2$) ou externe (CLKSRC=0, $f \leq f\mu p/2$) et dans ce même mode le Timer peut délivrer des impulsions ($C/\overline{P} = 0$) ou bien des tops d'horloge ($C/\overline{P} = 1$). L'horloge µp et disponible sur la broche $H_1$ du DSP (voir Figure II-2).

**Period register**
Permet de spécifier (en mode timer) la fréquence de sortie (de TCLK) du Timer comme on va le voir plus bas.
**Counter register**
Il est incrémenté à chaque top d'horloge (horloge source du Timer) jusqu'à atteindre la valeur du *Periode register* là,une impulsion est générée (mode Timer) via TCLK (voir Figure II-8).

**Timing du Timer**

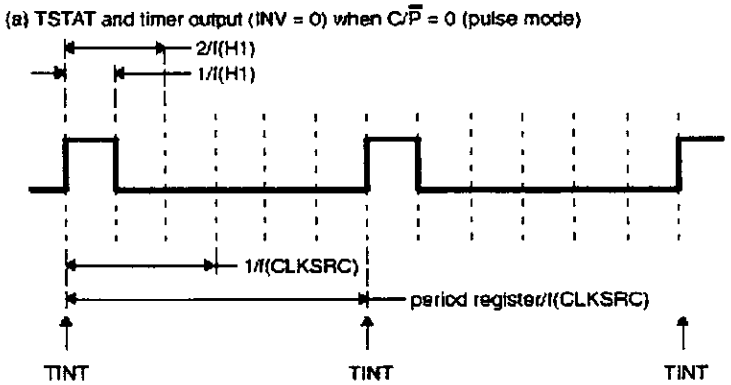



Figure II-10-a

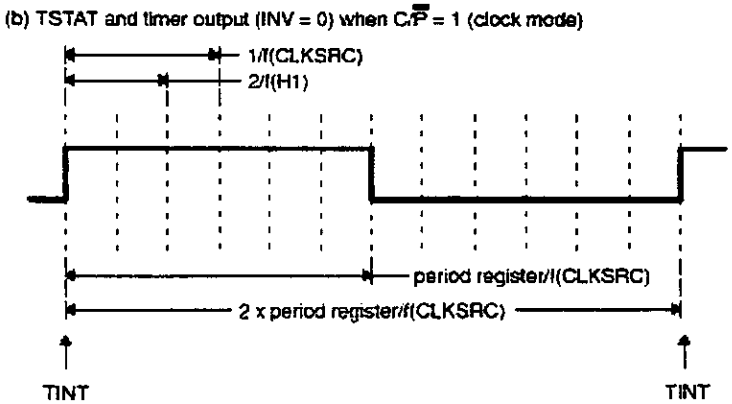



Figure II-10-b

*f(pulse mode) = f(timer clock) / period register*

*f(clock mode) = f(timer clock) / (2 x period register)*

## II-3-2-LE PORT SÉRIE

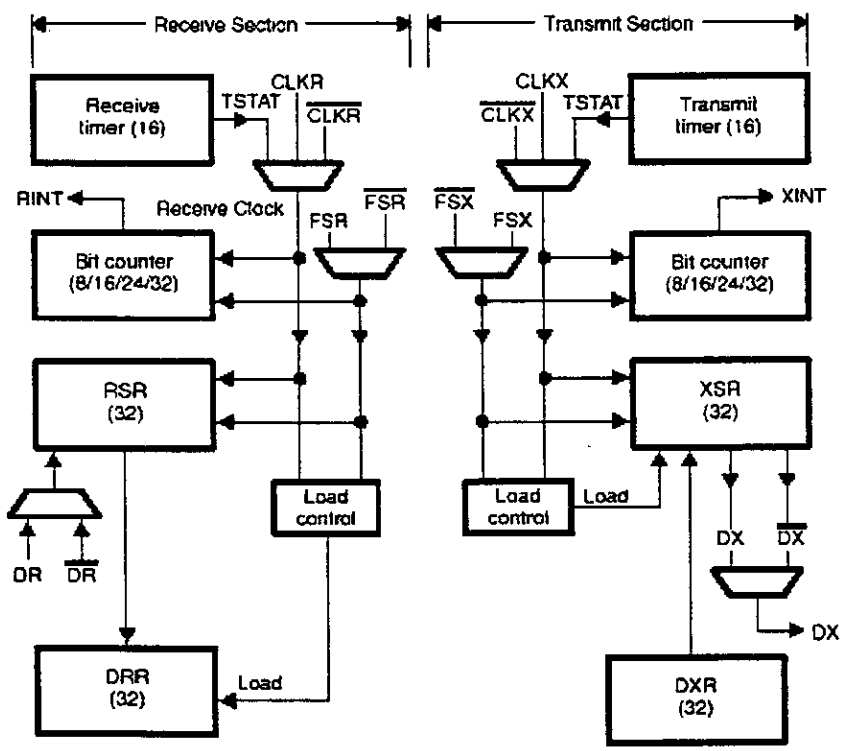



Figure II-11

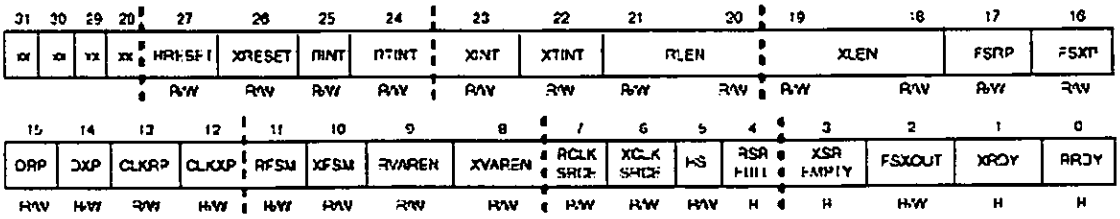
Le port série un périphérique qui permet la communication avec des circuits externes tels que des convertisseurs analogique/numérique ou avec d'autres processeurs. Le port série peut transmettre ou recevoir des mots de 8,16,24,32 ou 32 bits en mode synchrone (standard mode) ou asynchrone (continuous mode). L'horloge du port série peut être interne grâce à son timer (voir Figure II-11) ou externe via les broches CLKX en transmission et CLKR en réception.

**Les registres du Port série**

Il possède huit registres : le Global-control register, deux registres de contrôle pour les six broches d'E/S, trois registres timer (transmission/réception), le registre de transmission de donnée et le registre de réception de donnée.

**Global-control register**

Le mode de fonctionnement du port série est fixé par ce registre.




Notes: 1) R = read, W = write
2) xx = reserved bit, read as 0

Figure II-12

Les bits du registre de contrôle global (table II-7) :

Table II-7

| Abbreviation | Reset Value | Name | Description |
|---|---|---|---|
| RRDY | 0 | Receive ready flag | If RRDY = 1, the receive buffer has new data and is ready to be read. A three H1/H3 cycle delay occurs from the loading of DRR to RRDY = 1. The rising edge of this signal sets RINT.<br>If RRDY = 0, the receive buffer does not have new data since the last read. RRDY = 0 at reset and after the receive buffer is read. |
| XRDY | 1 | Transmit ready flag | If XRDY = 1, the transmit buffer has written the last bit of data to the shifter and is ready for a new word. A three H1/H3 cycle delay occurs from the loading of the transmit shifter until XRDY is set to 1. The rising edge of this signal sets XINT.<br>If XRDY = 0, the transmit buffer has not written the last bit of data to the transmit shifter and is not ready for a new word. |
| FSXOUT | | Transmit frame sync configuration | FSXOUT = 0 configures the FSX pin as an input.<br>FSXOUT = 1 configures the FSX pin as an output. |
| XSREMPTY | 0 | Transmit-shift register empty flag | If XSREMPTY = 0, the transmit-shift register is empty.<br>If XSREMPTY = 1, the transmit-shift register is not empty.<br>Reset or XRESET causes this bit to = 0. |

Suite de la table II-7 :

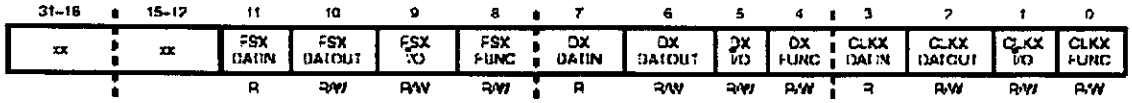
| | | | |
|---|---|---|---|
| **RSRFULL** | 0 | Receive-shift register full flag | If RSRFULL = 1, an overrun of the receiver has occurred. In continuous mode, RSRFULL is set to 1 when both RSR and DRR are full. In noncontinuous mode, RSRFULL is set to 1 when RSR and DRR are full and a new FSR is received. A read causes this bit to be set to 0. This bit can be set to 0 only by a system reset, a serial-port receive reset (RRESET = 1), or a read. When the receiver tries to set RSRFULL to 1 at the same time that the global register is read, the receiver dominates, and RSRFULL is set to 1.<br>If RSRFULL = 0, no overrun of the receiver has occurred. |
| **HS** | 0 | Handshake | If HS = 1, the handshake mode is enabled.<br>If HS = 0, the handshake mode is disabled. |
| **XCLK SRCE** | 0 | Transmit clock source | If XCLK SRCE = 1, the internal transmit clock is used.<br>If XCLK SRCE = 0, the external transmit clock is used. |
| **RCLK SRCE** | 0 | Receive clock source | If RCLK SRCE = 1, the internal receive clock is used.<br>If RCLK SRCE = 0, the external receive clock is used. |
| **XVAREN** | 0 | Transmit data rate mode | Specifies a fixed or variable data rate mode when transmitting.<br>With a fixed data rate, FSX is active for at least one XCLK cycle and then goes inactive before transmission begins.<br>With variable data rate, FSX is active while all bits are being transmitted. When you use an external FSX and variable data rate signaling, the DX pin is driven by the transmitter when FSX is held active or when a word is being shifted out. |
| **RVAREN** | 0 | Receive data rate mode | Specifies a fixed or variable data rate mode when receiving.<br>If RVAREN = 0 (fixed data rate), FSX is active for at least one RCLK cycle and then goes inactive before reception begins.<br>If RVAREN = 1 (controlled data rate), FSX is active while all bits are being received. |
| **XFSM** | 0 | Transmit frame sync mode | Configures the port for continuous mode operation or standard mode operation.<br>If XFSM = 1 (continuous mode), only the first word of a block generates a sync pulse, and the rest are transmitted continuously to the end of the block.<br>If XFSM = 0 (standard mode), each word has an associated sync pulse. |
| **RFSM** | 0 | Receive frame sync mode | Configures the port for continuous mode operation or standard mode operation.<br>If RFSM = 1 (continuous mode), only the first word of a block generates a sync pulse, and the rest are received continuously to the end of the block.<br>If RFSM = 0 (standard mode), each word received has an associated sync pulse. |
| **CLKXP** | | CLKX polarity | If CLKXP = 0, CLKX is active high.<br>If CLKXP = 1, CLKX is active low. |
| **CLKRP** | 0 | CLKR polarity | If CLKRP = 0, CLKR is active (high).<br>If CLKRP = 1, CLKR is active (low). |
| **DXP** | 0 | DX polarity | If DXP = 0, DX is active (high).<br>If DXP = 1, DX is active (low). |
| **DRP** | 0 | DR polarity | If DRP = 0, DR is active (high).<br>If DRP = 1, DR is active (low). |

Suite de la table II-7 :

| | | | |
|---|---|---|---|
| **FSXP** | 0 | FSX polarity | If FSXP = 0, FSX is active (high).<br>If FSXP = 1, FSX is active (low). |
| **FSRP** | 0 | FSR polarity | If FSRP = 0, FSR is active (high).<br>If FSRP = 1, FSR is active (low). |
| **XLEN** | 00 | Transmit word length | These two bits define the word length of serial data transmitted. All data is assumed to be right justified in the transmit buffer when fewer than 32 bits are specified.<br>0 0 — 8 bits  1 0 — 24 bits<br>0 1 — 16 bits  1 1 — 32 bits |
| **RLEN** | 00 | Receive word length | These two bits define the word length of serial data received. All data is right justified in the receive buffer.<br>0 0 — 8 bits  1 0 — 24 bits<br>0 1 — 16 bits  1 1 — 32 bits |
| **XTINT** | 0 | Transmit timer interrupt enable | If XTINT = 0, the transmit timer interrupt is disabled.<br>If XTINT = 1, the transmit timer interrupt is enabled. |
| **XINT** | 0 | Transmit interrupt enable | If XINT = 0, the transmit interrupt is disabled.<br>If XINT = 1, the transmit interrupt is enabled.<br>**Note:** The CPU receive flag XINT and the serial-port-to-DMA interrupt (EXINT0 in the IE register) is the OR of the enabled transmit timer interrupt and the enabled transmit interrupt. |
| **RTINT** | 0 | Receive timer interrupt enable | If RTINT = 0, the receive timer interrupt is disabled.<br>If RTINT = 1, the receive timer interrupt is enabled. |
| **RINT** | 0 | Receive interrupt enable | If RINT = 0, the receive interrupt is disabled.<br>If RINT = 1, the receive interrupt is enabled.<br>**Note:** The CPU receive interrupt flag RINT and the serial-port-to-DMA interrupt (ERINT0 in the IE register) are the OR of the enabled receive timer interrupt and the enabled receive interrupt. |
| **XRESET** | 0 | Transmit reset | If XRESET = 0, the transmit side of the serial port is reset.<br>To take the transmit side of the serial port out of reset, set XRESET to 1.<br>Do not set XRESET to 1 until at least three cycles after $\overline{\text{RESET}}$ goes inactive. This applies only to system reset. Setting XRESET to 0 does not change the contents of any of the serial-port control registers. It places the transmitter in a state corresponding to the beginning of a frame of data. Resetting the transmitter generates a transmit interrupt. Reset this bit during the time the mode of the transmitter is set. You can toggle XFSM without resetting the global-control register. |
| **RRESET** | 0 | Receive reset | If RRESET = 0, the receive side of the serial port is reset.<br>To take the receive side of the serial port out of reset, set RRESET to 1.<br>Do not set RRESET to 1 until at least three cycles after RESET goes inactive. This applies only to system reset. Setting RRESET to 0 does not change the contents of any of the serial-port control registers. It places the receiver in a state corresponding to the beginning of a frame of data. Reset this bit at the same time that the mode of the receiver is set. You can toggle without resetting the global-control register. |

**FSX/DX/CLKX Port-control register**

Contrôle la fonction des broches FSX,DX et CLKX.




Notes: 1) R = read, W = write.
2) xx = reserved bit, read as 0.

Figure II-13

Les bits de ce registre :

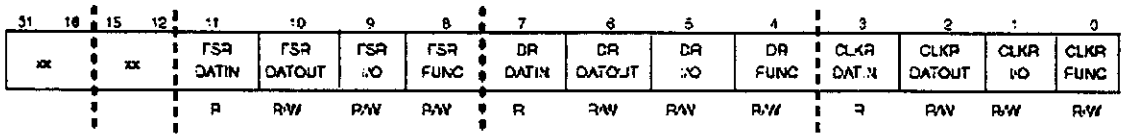
Table II-8

| Abbreviation | Reset Value | Name | Description |
|---|---|---|---|
| **CLKX FUNC** | 0 | Clock transmit function | Controls the function of CLKX.<br>If CLKX FUNC = 0, CLKX is configured as a general-purpose digital I/O port.<br>If CLKX FUNC = 1, CLKX is configured as a serial port pin. |
| **CLKX $\overline{I}$/O** | 0 | Clock transmit input/output mode | If CLKX $\overline{I}$/O = 0, CLKX is configured as a general-purpose input pin.<br>If CLKX $\overline{I}$/O = 1, CLKX is configured as a general-purpose output pin. |
| **CLKX DATOUT** | 0 | Clock transmit data ouput | Data output on CLKX when configured as general-purpose output. |
| **CLKX DATIN** | x† | Clock transmit data input | Data input on CLKX when configured as general-purpose input. A write has no effect. |
| **DX FUNC** | 0 | DX function | DXFUNC controls the function of DX.<br>If DXFUNC = 0, DX is configured as a general-purpose digital I/O port.<br>If DXFUNC = 1, DX is configured as a serial port pin. |
| **DX $\overline{I}$/O** | 0 | DX input/output mode | If DX $\overline{I}$/O = 0, DX is configured as a general-purpose input pin.<br>If DX $\overline{I}$/O = 1, DX is configured as a general-purpose output pin. |
| **DX DATOUT** | 0 | DX data output | Data output on DX when configured as general-purpose output. |
| **DX DATIN** | x† | DX data input | Data input on DX when configured as general-purpose input. A write has no effect. |
| **FSX FUNC** | 0 | FSX function | Controls the function of FSX.<br>If FSX FUNC = 0, FSX is configured as a general-purpose digital I/O port.<br>If FSX FUNC = 1, FSX is configured as a serial port pin. |
| **FSX $\overline{I}$/O** | 0 | FSX input/output mode | If FSX $\overline{I}$/O = 0, FSX is configured as a general-purpose input pin.<br>If FSX $\overline{I}$/O = 1, FSX is configured as a general-purpose output pin. |
| **FSX DATOUT** | 0 | FSX data output | Data output on FSX when configured as general-purpose output. |
| **FSX DATIN** | x† | FSX data input | Data input on FSX when configured as general-purpose input. A write has no effect. |

† x = 0 or 1

**FSR/DR/CLKR Port-control register**

Contrôle la fonction des broches FSR,DR et CLKR.




**Notes:** 1) R = read, W = write

2) xx = reserved bit, read as 0

Figure II-14

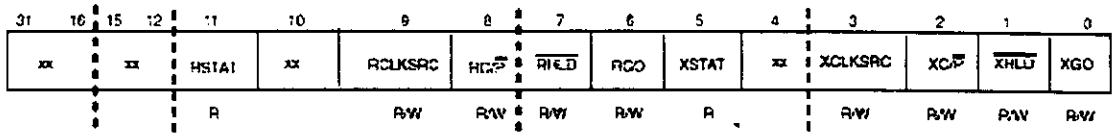
Les bits de ce registre :

Table II-9

| Abbreviation | Reset Value | Name | Description |
|---|---|---|---|
| **CLKR FUNC** | 0 | Clock receive function | Controls the function of CLKR.<br>If CLKR FUNC = 0, CLKR is configured as a general-purpose digital I/O port.<br>If CLKR FUNC = 1, CLKR is configured as a serial port pin. |
| **CLKR $\overline{I/O}$** | 0 | Clock receive input/output mode | If CLKR $\overline{I/O}$ = 0, CLKR is configured as a general-purpose input pin.<br>If CLKR $\overline{I/O}$ = 1, CLKR is configured as a general-purpose output pin. |
| **CLKR DATOUT** | 0 | Clock receive data output | Data output on CLKR when configured as general-purpose output. |
| **CLKR DATIN** | x† | Clock receive data input | Data input on CLKR when configured as general-purpose input. A write has no effect. |
| **DR FUNC** | 0 | DR function | Controls the function of DR.<br>If DR FUNC = 0, DR is configured as a general-purpose digital I/O port.<br>If DR FUNC = 1, DR is configured as a serial port pin. |
| **DR $\overline{I/O}$** | 0 | DR input/output mode | If DR $\overline{I/O}$ = 0, DR is configured as a general-purpose input pin.<br>If DR $\overline{I/O}$ = 1, DR is configured as a general-purpose output pin. |
| **DR DATOUT** | 0 | DR data output | Data output on DR when configured as general-purpose output. |
| **DR DATIN** | x† | DR data input | Data input on DR when configured as general-purpose input. A write has no effect. |
| **FSR FUNC** | 0 | FSR function | FSR FUNC controls the function of FSR.<br>If FSR FUNC = 0, FSR is configured as a general-purpose digital I/O port.<br>If FSR FUNC = 1, FSR is configured as a serial port pin. |
| **FSR $\overline{I/O}$** | 0 | FSR input/output mode | If FSR $\overline{I/O}$ = 0, FSR is configured as a general-purpose input pin.<br>If FSR $\overline{I/O}$ = 1, FSR is configured as a general-purpose output pin. |
| **FSR DATOUT** | 0 | FSR data output | Data output on FSR when configured as general-purpose output. |
| **FSR DATIN** | x† | FSR data input | Data input on FSR when configured as general-purpose input. A write has no effect. |

**Receive/Transmit Timer-control register**

Le Timer du port série est similaire au Timer étudié précédemment, il peut être considéré comme un Timer 16 bits doublé pour la réception et la transmission.




Notes: 1) R = read, W = write
2) xx = reserved bit, read as 0

Figure II-15

La table II-10 explique les bits de ce registre :

Table II-10

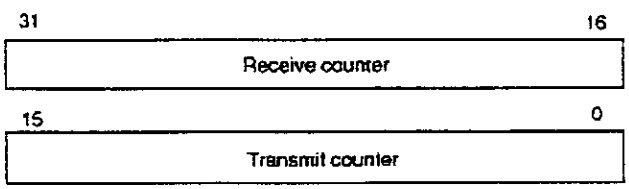
| Abbreviation | Reset Value | Name | Function |
|---|---|---|---|
| XGO | 0 | Transmit timer counter restart | Resets and restarts the transmit timer counter.<br>If XGO = 1 and the timer is not held, the counter is zeroed and begins incrementing on the next rising edge of the timer input clock.<br>The XGO bit is cleared on the same rising edge. Writing 0 to XGO has no effect on the transmit timer. |
| $\overline{XHLD}$ | 0 | Transmit counter hold signal | If $\overline{XHLD}$ = 0, the counter is disabled and held in its current state.<br>If $\overline{XHLD}$ = 1, the internal divide-by-two counter is also held so that the counter continues where it left off. |
| XC/$\overline{P}$ | 0 | Transmit clock/pulse mode control | When XC/$\overline{P}$ = 1, the clock mode is chosen. The signaling of the status flag and external output has a 50 percent duty cycle.<br>When XC/$\overline{P}$ = 0, the status flag and external output are active for one CLKOUT cycle during each timer period. |
| XCLKSRC | 0 | Transmit clock source | Specifies the source of the transmit timer clock.<br>When XCLKSRC = 1, an internal clock with frequency equal to one-half the CLKOUT frequency is used to increment the counter.<br>When XCLKSRC = 0, you can use an external signal from the CLKX pin to increment the counter.<br>The external clock source is synchronized internally, thus allowing for external asynchronous clock sources that do not exceed the specified maximum allowable external clock frequency, that is, less than f(H1)/2.6. |
| XTSTAT | 0 | Transmit timer status | Indicates the status of the transmit timer. It tracks what would be the output of the uninverted CLKX pin.<br>This flag sets a CPU interrupt on a transition from 0 to 1. A write has no effect. |
| RGO | 0 | Receive timer counter restart | Resets and starts the receive timer counter.<br>When RGO is set to 1 and the timer is not held, the counter is zeroed and begins incrementing on the next rising edge of the timer input clock.<br>The RGO bit is cleared on the same rising edge. Writing 0 to RGO has no effect on the receive timer. |

Suite de la table II-10 :

| | | | |
|---|---|---|---|
| $\overline{RHLD}$ | 0 | Receive counter hold signal | If $\overline{RHLD}$ = 0, the counter is disabled and held in its current state.<br>If $\overline{RHLD}$ = 1, the internal divide-by-2 counter is also held so that the counter will continue where it left off.<br>You can read and modify the timer registers while the timer is being held. $\overline{RESET}$ has priority over $\overline{RHLD}$. |
| RC/$\overline{P}$ | 0 | Rclock/pulse mode control | When RC/$\overline{P}$ = 1, the clock mode is chosen. The signaling of the status flag and external output has a 50% duty cycle.<br>When RC/$\overline{P}$ = 0, the status flag and external output are active for one CLKOUT cycle during each timer period. |
| RCLKSRC | 0 | Receive timer clock source | Specifies the source of the receive timer clock.<br>When RCLKSRC = 1, an internal clock with frequency equal to one-half the CLKOUT frequency is used to increment the counter.<br>When RCLKSRC = 0, you can use an external signal from the CLKR pin to increment the counter. The external clock source is synchronized internally, allowing for external asynchronous clock sources that do not exceed the specified maximum allowable external clock frequency (that is, less than f(H1)/2.6). |
| RTSTAT | 0 | Receive timer status | Indicates the status of the receive timer. It tracks what would be the output of the uninverted CLKR pin.<br>This flag sets a CPU interrupt on a transition from 0 to 1. A write has no effect. |

**Receive/Transmit Timer-counter register**

Il est incrémenté jusqu'à la valeur du registre de période. Ce registre est divisé en deux parties : l'une pour transmission et l'autres pour la réception (Figure II-16).

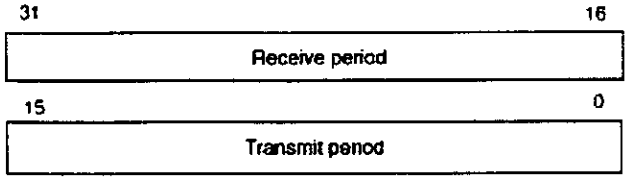



Note: All bits are read/write.

Figure II-16

**Receive/Transmit Timer-period register**

Permet de spécifier la fréquence timer de transmission et de réception (Figure II-17).




Note: All bits are read/write.

Figure II-17

**Data-transmit register (DXR)**
C'est le registre qui va contenir la donnée à transmettre. Dés que DXR est prêt (XRDY=1), la donnée (8/16/24/32 bits) peut y être chargée, la XRDY passe à 0 et la donnée est shiftée vers l'extérieure (Figure II-18) à travers le transmit-shift register (XSR). Pour une nouvelle transmission on doit attendre que XRDY soit à 1.

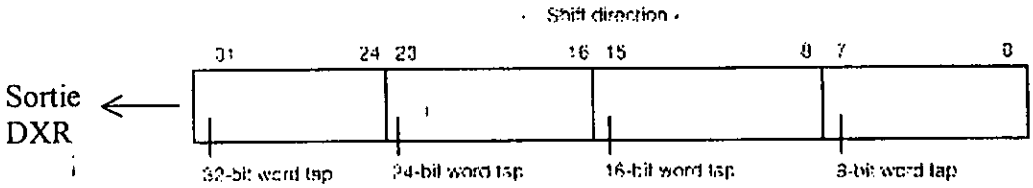



Figure II-18

Le test de XRDY n'est pas vraiment nécessaire si l'on configure le port série pour générer une interruption lors de la transmission (XINT). Là, il suffit de mettre le code dans la routine d'interruption associé à XINT (voir la table II-7 pour la configuration).

**Data-receive register**
C'est le registre qui va contenir la donnée reçue de l'extérieur. Dés qu'une donnée est shiftée dans le receive-shift register (RSR), le data-receive register (DRR) est chargé, là, RRDY passe à 1 et on peut alors lire DRR. La lecture de DRR remet RRDY à 0.
Comme pour la transmission on peut travailler en mode interruptible, la réception d'une donnée générant une interruption RINT.

Dans nos applications nous n'allons pas utiliser le contrôleur DMA, nous n'allons pas l'étudier.

## II-4-TMS320C30 MEMORY MAPS

Dans la partie périphériques nous avons vu que le TIMER et le port série ont plusieurs registres de configurations, ces registres sont dits *memory mapped registers* c-a-d qu'on y accède comme à des positions mémoires.

TMS320C30 memory map

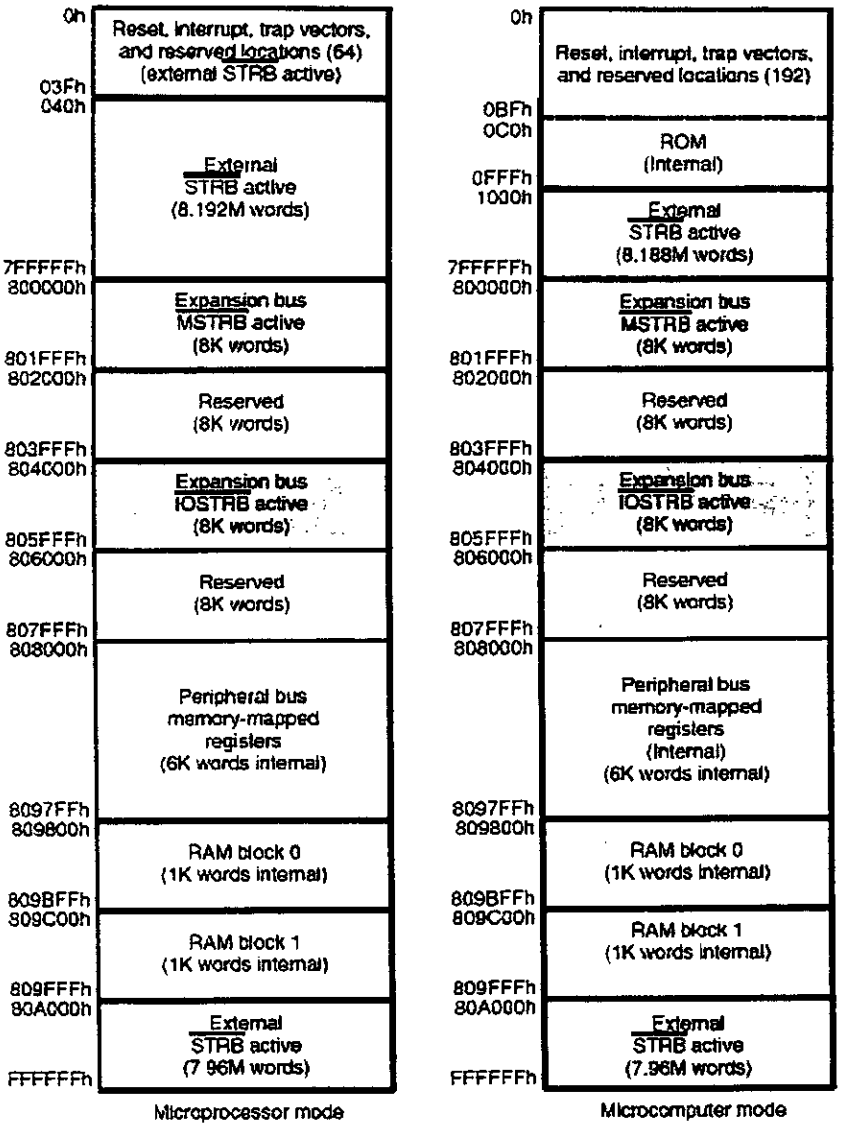



Figure II-19

Memory mapped TIMER locations

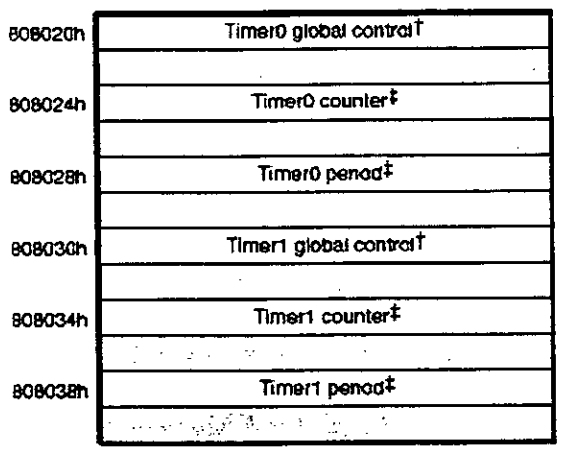



Figure II-19

Memory mapped locations for serial ports

| | |
|---|---|
| 808040h | Serial-port 0 global control† |
| | |
| 808042h | Serial port 0 FSX/DX/CLKX control‡ |
| 808043h | Serial port 0 FSR/DR/CLKR control§ |
| 808044h | Serial port 0 R/X timer control¶ |
| 808045h | Serial port 0 R/X timer counter# |
| 808046h | Serial port 0 R/X timer period‖ |
| | |
| 808048h | Serial port 0 data transmit★ |
| | |
| 80804Ch | Serial port 0 data receive❑ |
| | |
| 808050h | Serial-port 1 global control† |
| | |
| 808052h | Serial port 1 FSX/DX/CLKX control‡ |
| 808053h | Serial port 1 FSR/DR/CLKR control§ |
| 808054h | Serial port 1 R/X timer control¶ |
| 808055h | Serial port 1 R/X timer counter# |
| 808056h | Serial port 1 R/X timer period‖ |
| | |
| 808058h | Serial port 1 data transmit★ |
| | |
| 80805Ch | Serial port 1 data receive❑ |
| | |

Figure II-21

# Les Modes d'Adressage

# III- LES MODES D'ADRESSAGES

Il y a six types d'adressages qu'on peut utiliser dans quatre groupes d'adressage :

*General addressing modes (G).*
*3-operand addressing modes (T).*
*Parallel addressing modes (P).*
*Conditional-branch addressing modes (B).*

## III-1-ADRESSAGE PAR REGISTRE

Un registre CPU contient l'opérande.

Ex : ABSF R1 ; R1=|R1|

## III-2-ADRESSAGE DIRECT

Dans cet adressage on donne l'adresse de l'opérande. Cette adresse sera formée par la concaténation des 8 LSBs du registre de page DP avec 16 LSBs spécifiés par le mot d'instruction (expression). L'espace mémoire adressable maximale sans la modification du DP est de 64 Kmots (16 bits).

Adr=(8 LSBs de DP)∪(16 LSBs de expression)

Syntaxe assembleur : @expression.
ex : ADDI @ 0BCDEh,R7 ; si DP=8A alors Adr=8ABCDEh (h pour chiffre en hexa).

## III-3-ADRESSAGE IMMEDIAT

Dans cet adressage on donne directement l'opérande qui est sur 16 ou 24 bits. Selon l'instruction cette opérande peut être un entier (signé ou non) ou un nombre en virgule flottante.

ex : LDF 5.35,R0 ; 5.35 dans R0.
LDI 5,R0 ; 5 dans R0.

## III-4-ADRESSAGE RELATIF PC

Cet adressage est utilisé dans les branchements (étiquette ou adresse). La valeur du PC est modifié par un déplacement pour avoir l'adresse de branchement :
PC=PC+déplacement+1
ex : BU label ; si PC=1000h,label=1005h alors dépalcement=4.

## III-5-ADRESSAGE INDIRECT

L'adresse est le contenu d'un registre ARn.
On peut utiliser des pré ou post incrémentations/déplacements sur 8 bits.
On peut utiliser des registres d'index IR0/IR1.
Les différentes possibilités et le calcul d'adresses sont résumées dans les tables suivantes :
Tables III :

(a) Adressage indirect avec déplacement

| Syntaxe | Opération | Description |
|---|---|---|
| * + ARn(disp) | Adr=ARn+disp | Avec ajout d'un pré-déplacement |
| * - ARn(disp) | Adr=ARn-disp | Avec soustraction d'un pré-déplacement |
| * ++ ARn(disp) | Adr=ARn+disp<br>ARn=ARn+disp | Avec ajout d'un pré-déplacement et modification |
| * -- ARn(disp) | Adr=ARn-disp<br>ARn=ARn-disp | Avec soustraction d'un pré-déplacement et modification |
| * ARn ++ (disp) | Adr=ARn<br>ARn=ARn+disp | Avec ajout d'un post-déplacement et modification |
| * ARn -- (disp) | Adr=ARn<br>ARn=ARn-disp | Avec soustraction d'un post-déplacement et modification |
| * ARn ++ (disp)% | Adr=ARn<br>ARn=circ(ARn+disp) | Avec ajout d'un post-déplacement et modification circulaire |
| * ARn -- (disp)% | Adr=ARn<br>ARn=circ(ARn-disp) | Avec soustraction d'un post-déplacement et modification circulaire |

(b) Adressage indirect avec registres d'index

| Syntaxe | Opération | Description |
|---|---|---|
| * + ARn(IR) | Adr=ARn+IR | Avec ajout d'un pré-index |
| * - ARn(IR) | Adr=ARn-IR | Avec soustraction d'un pré-index |
| * ++ ARn(IR) | Adr=ARn+IR<br>ARn=ARn+IR | Avec ajout d'un pré-index et modification |
| * -- ARn(IR) | Adr=ARn-IR<br>ARn=ARn-IR | Avec soustraction d'un pré-index et modification |
| * ARn ++ (IR) | Adr=ARn<br>ARn=ARn+IR | Avec ajout d'un post-index et modification |
| * ARn -- (IR) | Adr=ARn<br>ARn=ARn-IR | Avec soustraction d'un post-index et modification |
| * ARn ++ (IR)% | Adr=ARn<br>ARn=circ(ARn+IR) | Avec ajout d'un post-index et modification circulaire |
| * ARn -- (IR)% | Adr=ARn<br>ARn=circ(ARn-IR) | Avec soustraction d'un post-index et modification circulaire |

(c) Adressage indirect (cas spéciaux)

| Syntaxe | Opération | Description |
|---|---|---|
| * ARn | Adr=ARn | |
| * ARn ++ (IR0)B | Adr=ARn<br>ARn=B(ARn+IR0) | Avec ajout d'un post-index (IR0) et modification par inversion de bits |

Légende :

Adr adresse mémoire
disp déplacement
ARn registres auxiliaires AR0-AR7
IR registres d'index IR0-IR1
B() adressage par inversion de bits
circ() adressage circulaire
B utilisation de l'adressage par inversion de bits
% utilisation de l'adressage circulaire

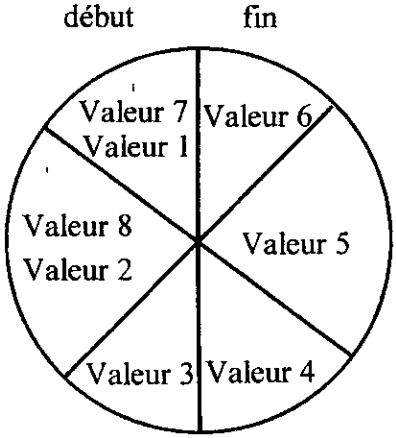
## III-6-ADRESSAGE CIRCULAIRE

L'adressage circulaire fait partie de l'adressage indirect sauf que le calcul d'adresse se d'une façon particulière :
(Voir les tables III-a et b)
$Adr = ARn$

$ARn = circ(ARn \pm disp / IR)$

avec :
$circ(A) = A \bmod [BK]$ ou BK est le registre taille de bloc.

**PRINCIPE**
Supposons un bloc de six positions mémoire (BK=6).
On veut mettre 8 valeurs dans ce bloc par un adressage circulaire :




**REGLES A RESPECTER**

On doit spécifier la taille du buffer (bloc) R dans le registre BK ( $R \leq 64K$ (une page)).
Soit K le nombre de bits variant dans les adresses des positions mémoire du buffer, l'adresse de début du buffer doit contenir K zéros :

$Adrdebut = xxxxx \underbrace{00\ldots0}_{Kzéros}$

Il faut que : $disp \leq R$ et $IR \leq BK$.
remarque : on peut implémenter plusieurs buffers circulaires tant qu'ils sont de même taille.

**APPLICATION**
Filtrage numérique.

## III-7-ADRESSAGE PAR INVERSION DE BITS

Ce mode d'adressage est spécialement dédié à la FFT, il suit les même règle que l'adressage circulaire ( $R \leq 64K$ ,l'adresse de début du buffer doit contenir K zéros) en plus de quelque règles supplémentaires :
la table doit contenir $2^n$ valeur (FFT).
Si on utilise deux tables différentes pour les parties réelles et imaginaires :
$R = 2^n; IR0 = 2^{n-1}; K = n$

Si les parties réelles et imaginaires sont mises dans la même table consécutivement (re0,im0,re1,im1,..etc) :
$R = 2^{n+1}; IR0 = 2^n; K = n+1$

**CALCUL D'ADRESSE**
(Voir table III-d)

$Adr = ARn$
$ARn = B(ARn + IR0)$

$B(ARn + IR0) \Leftrightarrow KLSBsARn = KLSBsARn + IR0$ avec propagation de retenu en inverse (de gauche à droite)

ex : 8 positions mémoire
R=8,K=3 ;IR0=$100_2$ (en base 2) ;
AR0=xx..x000 ;

* AR0 ++ (IR0)B ; AR0=xx..x000 (valeur 0)
* AR0 ++ (IR0)B ; AR0=xx..x100 (valeur 4)
* AR0 ++ (IR0)B ; AR0=xx..x010 (valeur 2)
* AR0 ++ (IR0)B ; AR0=xx..x110 (valeur 6)
* AR0 ++ (IR0)B ; AR0=xx..x001 (valeur 1)
* AR0 ++ (IR0)B ; AR0=xx..x101 (valeur 5)
* AR0 ++ (IR0)B ; AR0=xx..x011 (valeur 3)
* AR0 ++ (IR0)B ; AR0=xx..x111 (valeur 7)
* AR0 ++ (IR0)B ; AR0=xx..x000 (valeur 0)

...etc.
remarque : ce mode d'adressage permet de se passer de routine d'inversion de bits lors de la programmation de la FFT en assembleur.

# Le Module d'Evaluation EVM

# IV- LE MODULE D'EVALUATION EVM

## IV-1- DESCRIPTION GENERALE

L'EVM est une carte 8 bits compatible IBM PC/AT configurable sur 4 positions d'E/S pour l'adresse de base. Elle contient les circuits suivants :

- Un TMS320C30 cadencé à 30MHZ.
- 16K mots (de 32 bits) de SRAM *zero wait-state* (c-à-d quelles marchent à la fréquence du DSP).
- Un convertisseur analogique/numérique TLC32044 (l'AIC).
- Un port de communication PC bidirectionnel 16 bits (TBC *host port* + C30 *host port*).
- Des prises jacks RCA pour les E/S analogiques.

## IV-2- DIAGRAMME BLOC DE L'EVM

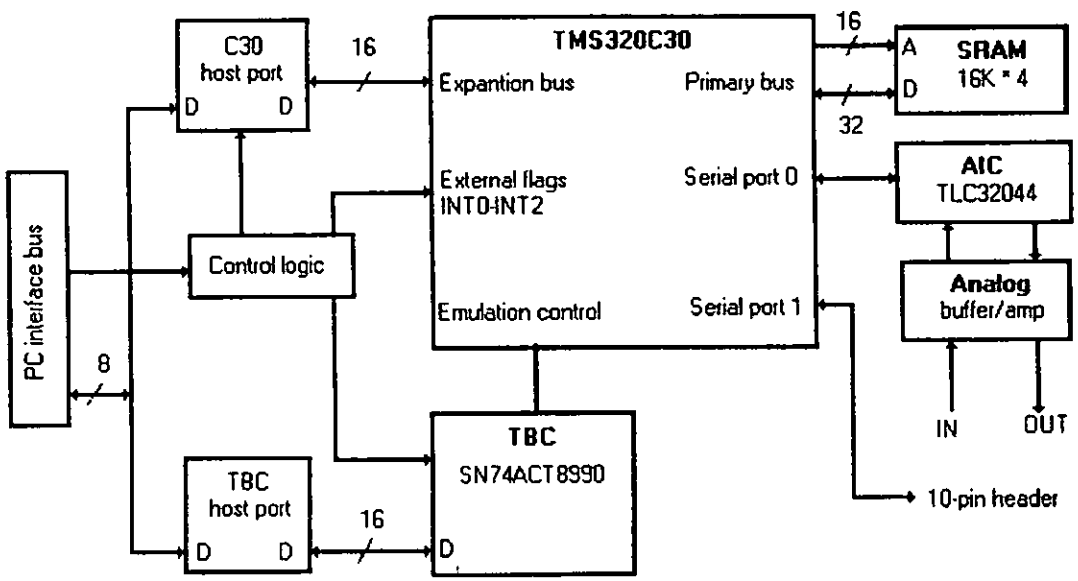



Figure IV-1

## IV-3-SWITCH ET CONFIGURATION MEMOIRE

Comme tout matériel installé sur un ordinateur, l'EVM va occuper une plage d'entrées /sorties dans le système (on appelles ca des ressources). Cette plage d'E/S peut être sélectionnée grâce à un switch 4 positions présent sur l'EVM (voir Figure IV-2).

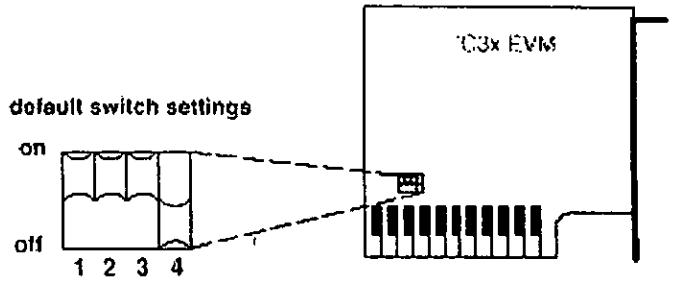



Figure IV-2

La table IV-1 montre les adresses d'E/S correspondant à chaque position du switch :

Table IV-1

| | Switches | | | |
|---|---|---|---|---|
| **Adresses d'E/S** | Sw1 | Sw2 | Sw3 | Sw4 |
| 0x0240-0x25f | On | On | On | Off |
| 0x0280-0x29f | On | Off | On | Off |
| 0x0320-0x033f | Off | On | On | Off |
| 0x340-0x035f | Off | Off | On | Off |

Les positions 3 et 4 ne doivent pas être modifiées.

Pour la configuration correcte de ce switch il faut regarder dans le panneau de configuration de Windows (dans système) les plages prises par les autres périphériques et choisir celle qui est libre pour ne pas avoir de conflit matériel. Généralement la position par défaut (la position une : on-on-on-off) ne cause pas de problème, il n'est donc pas nécessaire de la changer.

## IV-4-LES REGISTRES DE CONTROLE DE L'EVM

Les concepteurs de la carte ont défini six registres qui permettent l'envoi et la réception de données et l'envoi de commandes à l'EVM. Ces registres sont :

Table IV-2

| Registre | Offset | Taille | Accès |
|---|---|---|---|
| COM_DATA | 0x0808 | 16 | R/W |
| MINOR_CMD | 0x0014 | 16 | R/W |
| STATUS0 | 0x0400 | 16 | R |
| COM_CMD | 0x0800 | 8 | R/W |
| CONTROL5 | 0x000A | 16 | R/W |
| SOFT_RESET | 0x0818 | 0 | w |

- le registre COM_DATA est utilisé pour la transmission/réception de données de et vers l'EVM.
- le registre MINOR_CMD est utilisé dans la synchronisation (on le verra plus bas).

- le registre STATUS0 indique la disponibilité de l'EVM pour la réception ou la transmission (utilisé dans la synchronisation).
- le registre COM_CMD est utilisé pour l'envoi de commandes (instructions au DSP).
- le registre CONTROL5 est utilisé pour initialiser (reset) le TMS320C30.
- le registre SOFT_RESET est utilisé pour l'initialisation de l'EMV. Ce registre n'existe pas physiquement, c'est un décodage d'adresse qui permet cela.

Le PC accède à ces registres en écrivant (ou lisant) à l'adresse formée par l'addition de l'adresse de base de l'EVM avec l'Offset du registre :
Par exemple, si l'EVM se trouve à l'adresse 0x0240 (adresse par défaut) l'adresse du COM_DATA sera ADR=0x0240+0x0808 (voir tableau IV-2).

## IV-5-LE TEST BUS CONTROLER (TBC)

Le TBC est un circuit SN74ACT8990 *test bus controler*. Son rôle est de permettre un accès facile au TMS, de permettre le debuggage et le chargement des programmes.
Le TBC a quatre événements externes EVT0-EVT3. EVT0 sert à l'émulation et ne doit pas être modifié, EVT1 et EVT2 sont utilisés pour la synchronisation de transfert de données et EVT3 pour initialiser et maintenir en reset le TMS. Seuls les événements EVT1 et EVT2 sont importants et voici leurs descriptions :

### LES EVENEMENTS DU TBC

**EVT1-host read akcnowledge**

EVT1 est un *flag* de communication qui est mis à 1 lorsque le TMS320C30 écrit dans le registre de communication (COM_DATA). Quand EVT1 passe à 1 le PC (l'hôte) peut lire une donnée valide. L'hôte doit remettre à zéro EVT1 avant la lecture de la donnée.

**EVT2-host write akcnowledge**

EVT2 est un *flag* de communication qui est mis à 1 lorsque le TMS320C30 lit le registre de communication (COM_DATA). Quand EVT2 passe à 1 l'hôte peut écrire une nouvelle donnée dans COM_DATA. L'hôte doit remettre à zéro EVT2 avant chaque écriture de donnée.

## IV-6-LA SYNCHRONISATION DE TRANSFERT DE DONNEES

Grâce à EVT1 et EVT2 on peut synchroniser les communications entre le DSP et le PC qui doit sonder les événements EVT1 et EVT2.

TI recommande le sondage pour ne pas avoir de perte de donnée, mais il n'est pas nécessaire (cela dépend de l'application et de la rapidité du PC). Le débit de transfère avec la méthode de sondage est approximativement de 200 KB par seconde.

### ETAPES POUR UNE ECRITURE SYNCHRONE

1- mettre EVT2 à zéro.
2- écrire la donnée dans le registre COM_DATA.
3- réactualiser le registre STATUS0.
4- si EVT2=1 aller à 1 sinon refaire 3.

La table IV-3 résume les paramètres de chaque étape :

Table IV-3

| Etape | Registre | Valeur | Offset | Accès |
|---|---|---|---|---|
| 1 | MINOR_CMD | 0x0004 | 0x0014 | W |
| 2 | COM_DATA | la donnée | 0x0808 | W |
| 3 | MINOR_CMD | 0x6044 | 0x0014 | W |
| 4 | STATUS0 | 0x0004(masque) | 0x0400 | R |

**ETAPES POUR UNE LECTURE SYNCHRONE**

1- réactualiser le registre STATUS0.
2- si EVT1=1 aller à 3 sinon refaire 1.
3- mettre EVT1 à zéro.
4- lire la donnée dans le registre COM_DATA.

La table IV-4 résume les paramètres de chaque étape :

Table IV-4

| Etape | Registre | Valeur | Offset | Accès |
|---|---|---|---|---|
| 1 | MINOR_CMD | 0x6044 | 0x0014 | W |
| 2 | STATUS0 | 0x0002(masque) | 0x0400 | R |
| 3 | MINOR_CMD | 0x0002 | 0x0014 | W |
| 4 | COM_DATA | la donnée | 0x0808 | R |

## IV-7-ACCES DU DSP AU REGISTRE DE COMMUNICATION

On a vu que la communication entre le PC et le DSP se fait à travers les registres de l'EVM, principalement par le registre de communication COM_DATA. On a vu que le PC pouvait y accéder grasse à son adresse d'E/S dans le système (comme il accède à une carte son ou une imprimante).
De même, le TMS320C30 peut lire ou modifier le contenu de ce registre à l'adresse 0x804000 (c-a-d que pour le DSP, COM_DATA se trouve à l'adresse 0x804000).

## IV-8-L'INTERFACE ANALOGIQUE AIC

L'interface analogique AIC (Analog Interface Controller) est un circuit TLC32044 de Texas Instruments de la famille TLC320C4X des convertisseurs analogiques/numériques de qualité audio fait pour interfacer directement les TMS via le port série.

Ce circuit est particulièrement important car c'est lui qui va nous permettre l'acquisition de signaux physiques réels à traiter, chose pour laquelle les DSP ont été conçus.

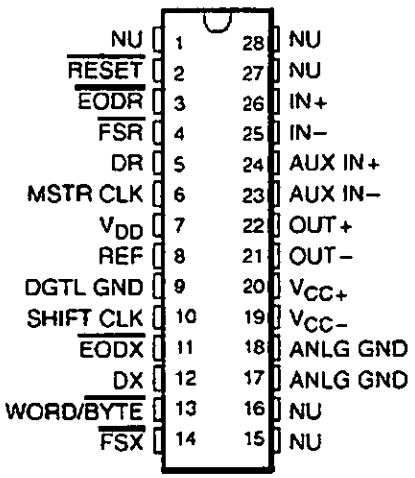
La description de ces circuits est présenter dans le figure IV-3 ci-dessous : (data sheet de TI)

# TLC32044C, TLC32044E, TLC32044I, TLC32044M, TLC32045C, TLC32045
# VOICE-BAND ANALOG INTERFACE CIRCUITS

SLAS017F – MARCH 1988 – REVISED MAY 199

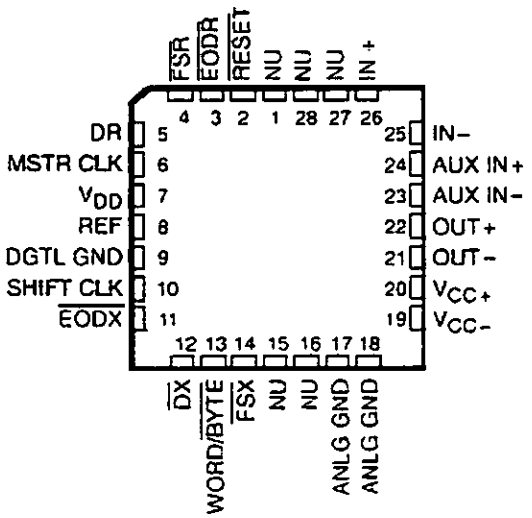
- 14-Bit Dynamic Range ADC and DAC
- 2's Complement Format
- Variable ADC and DAC Sampling Rate Up to 19,200 Samples per Second
- Switched-Capacitor Antialiasing Input Filter and Output-Reconstruction Filter
- Serial Port for Direct Interface to TMS(SMJ)320C17, TMS(SMJ)32020, TMS(SMJ)320C25, and TMS320C30 Digital Signal Processors
- Synchronous or Asynchronous ADC and DAC Conversion Rates With Programmable Incremental ADC and DAC Conversion Timing Adjustments
- Serial Port Interface to SN74(54)299 Serial-to-Parallel Shift Register for Parallel Interface to TMS(SMJ)32010, TMS(SMJ)320C15, or Other Digital Processors
- Internal Reference for Normal Operation and External Purposes, or Can Be Overridden by External Reference
- CMOS Technology

JT OR N PACKAGE
(TOP VIEW)




† Refer to the mechanical data for the JT package.

FK OR FN PACKAGE
(TOP VIEW)




## description

The TLC32044 and TLC32045 are complete analog-to-digital and digital-to-analog input and output systems on single monolithic CMOS chips. The TLC32044 and TLC32045 integrate a bandpass switched-capacitor antialiasing input filter, a 14-bit-resolution A/D converter, four microprocessor-compatible serial port modes, a 14-bit-resolution D/A converter, and a low-pass switched-capacitor output-reconstruction filter. The devices offer numerous combinations of master clock input frequencies and conversion/sampling rates, which can be changed via digital processor control.

Figure IV-3

## IV-8-1-DIAGRAMME BLOC DE L'AIC

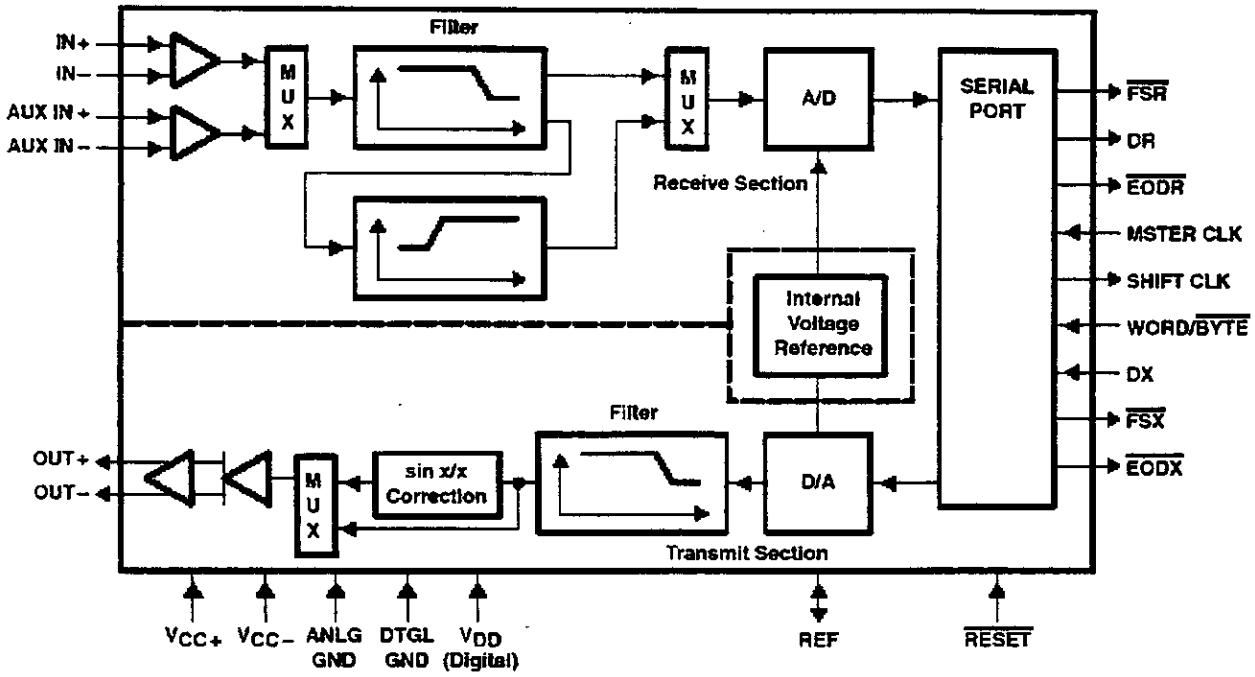



Figure IV-4

## IV-8-2-BROCHAGE DU CIRCUIT ET DESCRIPTION DES SIGNAUX

Table IV-5

| TERMINAL NAME | TERMINAL NO. | I/O | DESCRIPTION |
|---|---|---|---|
| ANLG GND | 17,18 | | Analog ground return for all internal analog circuits. Not internally connected to DGTL GND. |
| AUX IN+ | 24 | I | Noninverting auxiliary analog input stage. AUX IN+ can be switched into the bandpass filter and A/D converter path via software control. If the appropriate bit in the control register is a 1, the auxiliary inputs will replace the IN+ and IN– inputs. If the bit is a 0, the IN+ and IN– inputs will be used (see the AIC DX data word format section). |
| AUX IN– | 23 | I | Inverting auxiliary analog input (see the above AUX IN+ description). |
| DGTL GND | 9 | | Digital ground for all internal logic circuits. Not internally connected to ANLG GND. |
| DR | 5 | O | Data receive. DR is used to transmit the ADC output bits from the AIC to the TMS320 (SMJ320) serial port. This transmission of bits from the AIC to the TMS320 (SMJ320) serial port is synchronized with the SHIFT CLK signal. |
| DX | 12 | I | Data transmit. DX is used to receive the DAC input bits and timing and control information from the TMS320 (SMJ320). This serial transmission from the TMS320 (SMJ320) serial port to the AIC is synchronized with the SHIFT CLK signal. |
| $\overline{\text{EODR}}$ | 3 | O | End of data receive. (See the WORD/$\overline{\text{BYTE}}$ description and Serial Port Timing diagram.) During the word-mode timing, $\overline{\text{EODR}}$ is a low-going pulse that occurs immediately after the 16 bits of A/D information have been transmitted from the AIC to the TMS320 (SMJ320) serial port. $\overline{\text{EODR}}$ can be used to interrupt a microprocessor upon completion of serial communications. Also, $\overline{\text{EODR}}$ can be used to strobe and enable external serial-to-parallel shift registers, latches, or external FIFO RAM, and to facilitate parallel data bus communications between the AIC and the serial-to-parallel shift registers. During the byte-mode timing, $\overline{\text{EODR}}$ goes low after the first byte has been transmitted from the AIC to the TMS320 (SMJ320) serial port and is kept low until the second byte has been transmitted. The DSP can use this low-going signal to differentiate between the two bytes as to which is first and which is second. $\overline{\text{EODR}}$ does not occur after secondary communication. |

Suite de la table IV-5 :

| TERMINAL NAME | TERMINAL NO. | I/O | DESCRIPTION |
|---|---|---|---|
| $\overline{EODX}$ | 11 | O | End of data transmit. (See the WORD/$\overline{BYTE}$ description and Serial Port Timing diagram.) During the word-mode timing, $\overline{EODX}$ is a low-going pulse that occurs immediately after the 16 bits of D/A converter and control or register information have been transmitted from the TMS320 (SMJ320) serial port to the AIC. $\overline{EODX}$ can be used to interrupt a microprocessor upon the completion of serial communications. Also, $\overline{EODX}$ can be used to strobe and enable external serial-to-parallel shift registers, latches, or an external FIFO RAM, and to facilitate parallel data-bus communications between the AIC and the serial-to-parallel shift registers. During the byte-mode timing, $\overline{EODX}$ goes low after the first byte has been transmitted from the TMS320 (SMJ320) serial port to the AIC and is kept low until the second byte has been transmitted. The DSP can use this low-going signal to differentiate between the two bytes as to which is first and which is second. |
| $\overline{FSR}$ | 4 | O | Frame sync receive. In the serial transmission modes, which are described in the WORD/$\overline{BYTE}$ description, $\overline{FSR}$ is held low during bit transmission. When $\overline{FSR}$ goes low, the TMS320 (SMJ320) serial port begins receiving bits from the AIC via DR of the AIC. The most significant DR bit is present on DR before $\overline{FSR}$ goes low. (See Serial Port Timing and Internal Timing Configuration diagrams.) $\overline{FSR}$ does not occur after secondary communications. |
| $\overline{FSX}$ | 14 | O | Frame sync transmit. When $\overline{FSX}$ goes low, the TMS320 (SMJ320) serial port begins transmitting bits to the AIC via DX of the AIC. In all serial transmission modes, which are described in the WORD/$\overline{BYTE}$ description, $\overline{FSX}$ is held low during bit transmission (see Serial Port Timing and Internal Timing Configuration diagrams). |
| IN+ | 26 | I | Noninverting input to analog input amplifier stage |
| IN− | 25 | I | Inverting input to analog input amplifier stage |
| MSTR CLK | 6 | I | Master clock. MSTR CLK is used to derive all the key logic signals of the AIC, such as the shift clock, the switched-capacitor filter clocks, and the A/D and D/A timing signals. The Internal Timing Configuration diagram shows how these key signals are derived. The frequencies of these key signals are synchronous submultiples of the master clock frequency to eliminate unwanted aliasing when the sampled analog signals are transferred between the switched-capacitor filters and the A/D and D/A converters (see the Internal Timing Configuration diagram). |
| OUT+ | 22 | O | Noninverting output of analog output power amplifier. OUT+ can drive transformer hybrids or high-impedance loads directly in either a differential or a single-ended configuration. |
| OUT− | 21 | O | Inverting output of analog output power amplifier. OUT− is functionally identical with and complementary to OUT+. |
| REF | 8 | I/O | Internal voltage reference. An internal reference voltage is brought out on REF. An external voltage reference can also be applied to REF. |
| $\overline{RESET}$ | 2 | I | Reset function. $\overline{RESET}$ is provided to initialize the TA, TA', TB, RA, RA', RB, and control registers. A reset initiates serial communications between the AIC and DSP. A reset initializes all AIC registers including the control register. After a negative-going pulse on $\overline{RESET}$, the AIC registers are initialized to provide an 8-kHz data conversion rate for a 5.184-MHz master clock input signal. The conversion rate adjust registers, TA' and RA', are reset to 1. The control register bits are reset as follows (see AIC DX data word format section):<br>d9 = 1, d7 = 1, d6 = 1, d5 = 1, d4 = 0, d3 = 0, d2 = 1.<br>This initialization allows normal serial-port communication to occur between the AIC and DSP. |
| SHIFT CLK | 10 | O | Shift clock. SHIFT CLK is obtained by dividing the master clock signal frequency by four. SHIFT CLK is used to clock the serial data transfers of the AIC, described in the WORD/$\overline{BYTE}$ description below (see the Serial Port Timing and Internal Timing Configuration diagrams). |
| $V_{DD}$ | 7 | | Digital supply voltage, 5 V ±5% |
| $V_{CC+}$ | 20 | | Positive analog supply voltage, 5 V ±5% |
| $V_{CC-}$ | 19 | | Negative analog supply voltage, −5 V ±5% |

Suite de la table IV-5 :

| TERMINAL NAME | NO. | I/O | DESCRIPTION |
|---|---|---|---|
| WORD/BYTE | 13 | I | Used in conjunction with a bit in the control register, WORD/BYTE is used to establish one of four serial modes. These four serial modes are described below.<br>*AIC transmit and receive sections are operated asynchronously.*<br>The following description applies when the AIC is configured to have asynchronous transmit and receive sections. If the appropriate data bit in the control register is a 0 (see the AIC DX data word format section), the transmit and receive sections are asynchronous.<br>L Serial port directly interfaces with the serial port of the DSP and communicates in two 8-bit bytes. The operation sequence is as follows (see Serial Port Timing diagrams).<br>1. FSX or FSR is brought low.<br>2. One 8-bit byte is transmitted or one 8-bit byte is received.<br>3. EODX or EODR is brought low.<br>4. FSX or FSR emits a positive frame-sync pulse that is four shift clock cycles wide.<br>5. One 8-bit byte is transmitted or one 8-bit byte is received.<br>6. EODX or EODR is brought high.<br>7. FSX or FSR is brought high.<br>H Serial port directly interfaces with the serial ports of the TMS(SMJ)32020, TMS(SMJ)320C25, or TMS(SMJ)320C30, and communicates in one 16-bit word. The operation sequence is as follows (see Serial Port Timing diagrams):<br>1. FSX or FSR is brought low.<br>2. One 16-bit word is transmitted or one 16-bit word is received.<br>3. FSX or FSR is brought high.<br>4. EODX or EODR emits a low-going pulse.<br>*AIC transmit and receive sections are operated synchronously.*<br>If the appropriate data bit in the control register is 1, the transmit and receive sections are configured to be synchronous. In this case, the bandpass switched-capacitor filter and the A/D conversion timing are derived from the TX counter A, TX counter B, and TA, TA', and TB registers, rather than the RX counter A, RX counter B, and RA, RA', and RB registers. In this case, the AIC FSX and FSR timing are identical during primary data communication; however, FSR is not asserted during secondary data communication since there is no new A/D conversion result. The synchronous operation sequences are as follows (see Serial Port Timing diagrams).<br>L Serial port directly interfaces with the serial port of the DSP and communicates in two 8-bit bytes. The operation sequence is as follows (see Serial Port Timing diagrams):<br>1. FSX and FSR are brought low.<br>2. One 8-bit byte is transmitted and one 8-bit byte is received.<br>3. EODX and EODR are brought low.<br>4. FSX and FSR emit positive frame-sync pulses that are four shift clock cycles wide.<br>5. One 8-bit byte is transmitted and one 8-bit byte is received.<br>6. EODX and EODR are brought high.<br>7. FSX and FSR are brought high.<br>H Serial port directly interfaces with the serial port of the TMS(SJM)32020, TMS(SMJ)320C25, or TMS320C30, and communicates in one 16-bit word. The operation sequence is as follows (see Serial Port Timing diagrams):<br>1. FSX and FSR are brought low.<br>2. One 16-bit word is transmitted and one 16-bit word is received<br>3. FSX and FSR are brought high.<br>4. EODX or EODR emit low-going pulses.<br>Since the transmit and receive sections of the AIC are now synchronous, the AIC serial port with additional NOR and AND gates interface to two SN74(54)299 serial-to-parallel shift registers. Interfacing the AIC to the SN74(54)299 shift register allows the AIC to interface to an external FIFO RAM and facilitates parallel, data bus communications between the AIC and the digital signal processor. The operation sequence is the same as the above sequence (see Serial Port Timing diagrams). |

## IV-8-3-PRICIPE DE FONCTIONNEMENT

### Les entrées analogiques :

Le circuit a deux paires d'entrées analogiques (figure IV-4) IN-,IN+ et AUXIN+,AUXIN-. Sur l'EVM seul IN+ est utilisée et IN- est reliée à la masse analogique AGND.

**Les filtres numériques**

Ils sont du type *switched-capacitor filter*.

On a deux filtres à l'entrée, un filtre passe haut qui élimine la composante continue du signal et un filtre passe bas anti-repliement, à la sortie il y a un filtre passe bas. La fonction de transfert de ces filtres dépend de la fréquence d'échantillonnage et de l'horloge (*switched-capacitor filter clock)* choisies de telle sorte que les fréquences de coupure changent (automatiquement) de façon à respecter le critère de SHANON.

En plus de ces trois filtres il y a un filtre continu de correction (sinx)/x qui permet l'élimination des composantes périodiques dues aux horloges des filtres numériques et à l'échantillonnage.

**Les sorties analogiques :**

Deux sorties analogiques sont disponibles OUT+ et OUT- à travers un étage de puissance qui peut supporter des charges de faible impédance (sur l'EVM seul OUT+ est disponible).

**Le port série**

Le TLC32044 a un port série qui permet une liaison directe avec le DSP à travers son port série0 (figure IV-5). Le TIMER0 fournit (par sa broche TCLK0) l'horloge de base du TLC MCLK (on programme le TIMER pour donner une fréquence $f = f_{\mu p}/4$) qui à son tour génère les horloges de transmission et de réception pour le port série0 ainsi que les signaux de synchronisation FSX0 et FSR0. Le port série du DSP reçoit les données par DR0 et peut les transmettre par DX0. La broche XF0 est une broche d'entrée/sortie du DSP (voir II-2 et II-6), elle est reliée au reset du TLC et va nous servir lors de la programmation de celui-ci.

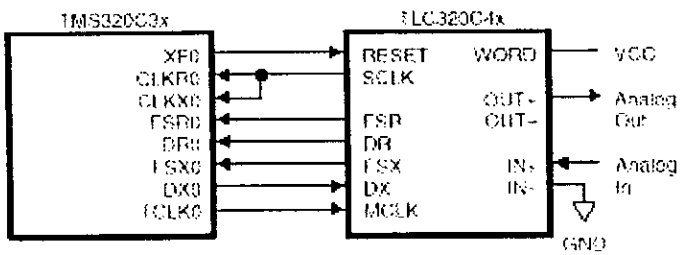



Figure IV-5

## IV-8-4-PROGRAMMATION DE L'AIC

L'AIC a six registres qui permettent de choisir la fréquence d'horloge des filtres et la fréquence d'échantillonnage. Ces registres sont TA, TA` et TB pour la partie transmission, et RA, RA` et RB pour la partie réception (figure IV-6). L'AIC peut marcher en mode synchrone (une impulsion de synchronisation avec chaque mot ) on a alors :

**Horloge des filtres numériques en mode synchrone**

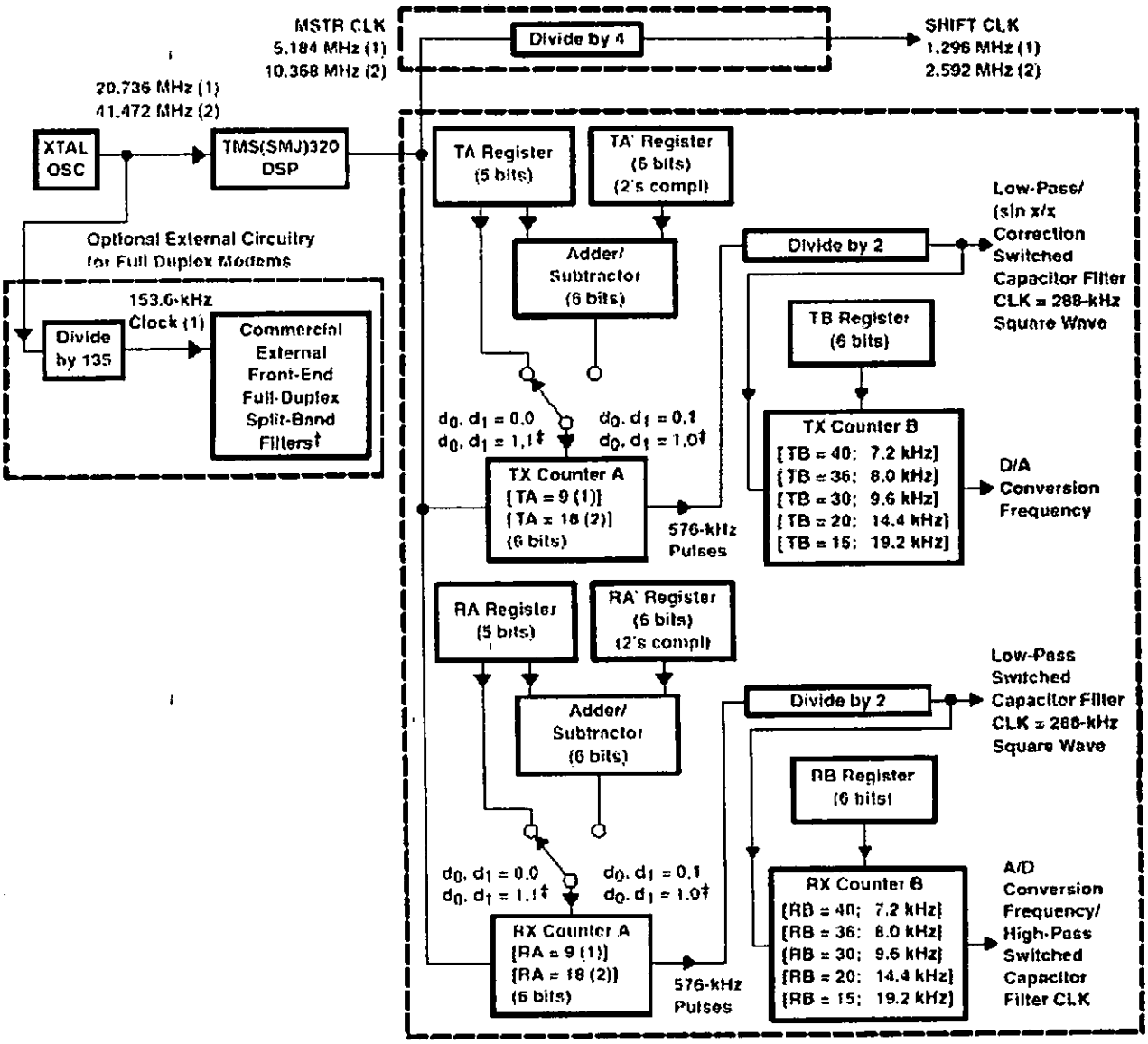
$$f(A/D) = \frac{MCLK}{2 \cdot TA} \quad \text{et} \quad f(D/A) = \frac{MCLK}{2 \cdot RA}$$

**Fréquence d'échantillonnage en mode synchrone**

$$f(A/D) = \frac{MCLK}{2 \cdot TA \cdot TB} \quad \text{et} \quad f(D/A) = \frac{MCLK}{2 \cdot RA \cdot RB}$$

En mode asynchrone (un groupe de mot produit une impulsion de synchronisation), les registres de la partie réception ne sont pas utilisés et on a les même fréquences pour les deux parties fixées par TA,TB et TA' (selon les même formules).

La figure IV-6 montre sur deux exemples comment les fréquences sont générées.



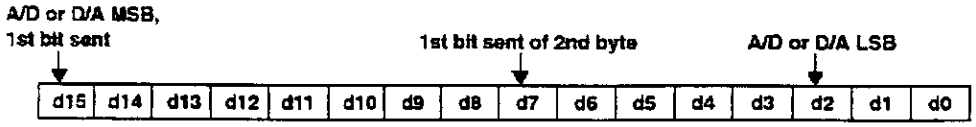

† Split-band filtering can alternatively be performed after the analog input function via software in the TMS/SMJ320.
‡ These control bits are described in the AIC DX data word format section.
NOTE: Frequency 1 (20.736 MHz) is used to show how 153.6 kHz (for a commercially available modem split-band filter clock), popular speech and modem sampling signal frequencies, and an internal 288-kHz switched-capacitor filter clock can be derived synchronously and as submultiples of the crystal oscillator frequency. Since these derived frequencies are synchronous submultiples of the crystal frequency, aliasing does not occur as the sampled analog signal passes between the analog converter and switched-capacitor filter stages. Frequency 2 (41.472 MHz) is used to show that the AIC can work with high frequency signals, which are used by high speed digital signal processors.

Figure IV-6

Les registres TA' et RA' sont utilisés pour modifier les périodes de conversion A/D et D/A en chargeant les compteurs TX et RX par les valeurs de $TA \pm TA'$ et $RA \pm RA'$ (table IV-6)

Table IV-6

**AIC DR or DX word bit pattern**




**AIC DX data word format section**

| d15 | d14 | d13 | d12 | d11 | d10 | d9 | d8 | d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 | Comments |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **primary DX serial communication protocol** | | | | | | | | | | | | | | | | |
| ←d15 (MSB) through d2 go to the D/A converter register | | | | | | | | | | | | | → | 0 | 0 | The TX and RX counter As are loaded with the TA and RA register values. The TX and RX counter Bs are loaded with TB and RB register values. |
| ←d15 (MSB) through d2 go to the D/A converter register | | | | | | | | | | | | | → | 0 | 1 | The TX and RX counter As are loaded with the TA + TA' and RA + RA' register values. The TX and RX counter Bs are loaded with the TB and RB register values LSBs d1 = 0 and d0 =1 cause the next D/A and A/D conversion periods to be changed by the addition of TA' and RA' master clock cycles, in which TA' and RA' can be positive or negative or zero (refer to Table 1). |
| ←d15 (MSB) through d2 go to the D/A converter register | | | | | | | | | | | | | → | 1 | 0 | The TX and RX counter As are loaded with the TA – TA' and RA – RA' register values. The TX and RX counter Bs are loaded with the TB and RB register values. LSBs d1 = 1 and d0 = 0 cause the next D/A and A/D conversion periods to be changed by the subtraction of TA' and RA' master clock cycles, in which TA' and RA' can be positive or negative or zero (refer to Table 1). |
| ←d15 (MSB) through d2 go to the D/A converter register | | | | | | | | | | | | | → | 1 | 1 | The TX and RX counter As are loaded with the TA and RA register converter register values. The TX and RX counter Bs are loaded with the TB and RB register values. After a delay of four shift clock cycles, a secondary transmission immediately follows to program the AIC to operate in the desired configuration. |

NOTE: Setting the two least significant bits to 1 in the normal transmission of DAC information (primary communications) to the AIC initiates secondary communications upon completion of the primary communications. Upon completion of the primary communication, $\overline{FSX}$ remains high for four shift clock cycles and then goes low and initiates the secondary communication. The timing specifications for the primary and secondary communications are identical. In this manner, the secondary communication, if initiated, is interleaved between successive primary communications. This interleaving prevents the secondary communication from interfering with the primary communications and DAC timing, thus preventing the AIC from skipping a DAC output. In the synchronous mode, $\overline{FSR}$ is not asserted during secondary communications.

L'envoi des mots 0,1 ou 2 au TLC (par le port série0) fixe la façon dont les conteurs seront chargés. L'envoi du mot 3 permet d'initier une séquence de communication secondaire (*setup secondary communication*) pour donner les valeurs aux registres TA, TA', TB, RA, RA' et RB (table IV-7).

Table IV-7

**secondary DX serial communication protocol**

| | | |
|---|---|---|
| x x I ← to TA register → I x x I ← to RA register → I | 0 0 | d13 and d6 are MSBs (unsigned binary) |
| x I ← to TA' register → I x I ← to RA' register → I | 0 1 | d14 and d7 are 2's complement sign bits |
| x I ← to TB register → I x I ← to RB register → I | 1 0 | d14 and d7 are MSBs (unsigned binary) |
| x x x x x x d9 x d7 d6 d5 d4 d3 d2 1 1<br>← Control Register → | | d2 = 0/1 deletes/inserts the A/D high-pass filter<br>d3 = 0/1 disables/enables the loopback function<br>d4 = 0/1 disables/enables the AUX IN+ and AUX IN−<br>d5 = 0/1 asynchronous/synchronous transmit and receive sections<br>d6 = 0/1 gain control bits (see gain control section)<br>d7 = 0/1 gain control bits (see gain control section)<br>d9 = 0/1 delete/insert on-board second-order (sin x)/x correction filter |

Table IV-8

**Gain Control Table (Analog Input Signal Required for Full-Scale A/D Conversion)**

| INPUT CONFIGURATIONS | CONTROL REGISTER BITS | | ANALOG INPUT‡ | A/D CONVERSION RESULT |
|---|---|---|---|---|
| | d6 | d7 | | |
| Differential configuration<br>Analog input = IN+ − IN−<br>= AUX IN+ − AUX IN− | 1 | 1 | ±6 V | Full-scale |
| | 0 | 0 | | |
| | 1 | 0 | ±3 V | Full-scale |
| | 0 | 1 | +1.5 V | Full-scale |
| Single-ended configuration<br>Analog input = IN+ − ANLG GND<br>= AUX IN+ − ANLG GND | 1 | 1 | +3 V | Half-scale |
| | 0 | 0 | | |
| | 1 | 0 | −3 V | Full-sale |
| | 0 | 1 | ±1.5 V | Full-scale |

‡ In this example, $V_{ref}$ is assumed to be 3 V. In order to minimize distortion, it is recommended that the analog input not exceed 0.1 dB below full scale.

Après avoir initié la séquence de communication secondaire on envoi un des quatre mots possibles pour modifier soit TA et RA ou TA' et RA' ou TB ou RB' ou bien pour les autres contrôles.

Ahmed Saïd Aziz juin 2000

# Les Outils de Développement

# V-LES OUTILS DE DEVELOPPEMENT

Les logiciels de développement fournis pas TI sont les outils d'assemblage (assembleur, lieur), un compilateur C ANSI et un debugger(nous n'allons pas en parler car il est d'un emploi facile, il suffit de lire le *Tutorial* dans la référence [5]).

## V-1-L'ASSEMBLEUR

### LIGNE DE COMMANDE

**asm30** [nom-fichier[nom-objet[nom-listing]]] [- option]

nom-fichier : nom du fichier à assembler (extension par défaut *.asm)
nom-objet :nom du fichier objet que l'assembleur va générer (par défaut nom-fichier.obj)
nom-listing :nom du fichier listing si l'option –l est utilisée

### LES OPTIONS LES PLUS USUELLES

-v :version de DSP : -v30 pour le C3X, -v40 pour les C4X (par défaut –v30)
-l :produit un listing
-c :ne fait pas de distinction entre les minuscules et les majuscules
-i :spécifie un répertoire ou l'assembleur peut trouver les fichiers

ex : asm30 fichier.asm

### INSTRUCTIONS ASSEMBLEUR LES PLUS UTILISEES

LDI :chargement d'un registre Rn ou ARn par entier
Ex : LDI 3,R0

LDF :chargement d'un registre Rn par nombre en virgule flottante
Ex : LDF 3.14,R4

MPYI :multiplication d'un registre Rn par un entier
Ex : MPYI 5,R3

MPYF : multiplication d'un registre Rn par un nombre en virgule flottante
Ex : MPYF 5.203,R3

FIX :conversion d'un nombre en virgule flottante en un entier
Ex : FIX R1,R1 (si R1 contenait la valeur 1.02, sa nouvelle valeur sera 1)

AND : fait un AND entre deux opérandes
Ex : AND R1,02h,R2 (R1&02 dans R2)

STI :sauvegarde en mémoire d'un entier
Ex : STI R2,@ABC2 (met le contenu de R2 dans l'adresse donnée par ABC2+DP (voir modes d'adressages)

Bcond :branchement conditionnel
Ex : BZ label (branchement à 'label' si le résultat de l'opération précédantes est nul).

## V-2-LE LIEUR

### LIGNE DE COMMANDE

**LNK30** [- option] nom-fichier1 nom-fichier2 ...

nom-fichierN: nom des fichiers à lier pour créer un exécutable DSP (extension par défaut *.obj)

### LES OPTIONS LES PLUS USUELLES

-m :produit un listing
-o : spécifie le nom du fichier exécutable (par défaut a.out)
-i :spécifie un répertoire où le lieur peut trouver les fichiers
-l :spécifie une librairie

ex : LNK30 fichier.OBJ -o fichier .out (link le fichier 'fichier.obj' et produit en sortie un fichier nommé 'fichier.out')

## V-3-LE COMPILATEUR C

### LIGNE DE COMMANDE

**Cl30** [- option] [nom-fichier]

nom-fichier : nom du fichier à assembler (extension par défaut *.c)

### LES OPTIONS LES PLUS USUELLES

- pk :compatibilité avec la définition K&R du C
-k :garde le fichier assembleur
-g :génère des directives de debuggage (indispensable pour l'utilisation du debugger C)
- mb :utilise un modèle mémoire large
-o :invoque l'optimiseur (opt30)

ex : cl30 -o - pk -g source.c

Le compilateur C est fourni avec des librairies (propres aux TMS) contenant les fonctions mathématiques du C et les fonctions de gestion dynamique de la mémoire. Le code source de ces librairies est nommé 'Rts.src' qu'on peut reconstruire selon nos exigences.

### CONSTRUCTION DE LIBRAIRIES

C'est le programme **mk30.exe** fourni par TI qui permet cela.

**Ligne de commande**

**mk30** [options] Rts.src -l nomlib.lib

nomlib.lib :nom qu'on veut donner à la librairie
**Les options les plus usuelles**

- pk :compatibilité avec la définition K&R du C

- g :génère des directives de debuggage (indispensable pour l'utilisation du debugger C)
- mb :utilise un modèle mémoire large
-- h :pour la génération des fichiers .h du C (comme math.h)
-- u :pour garder les fichiers .h existants
- o :invoque l'optimiseur (opt30)
- x :force la relecture des librairies

ex : mk30 --h -o -x - pk Rts.src -l rts30k.lib

# Applications

# VI-APPLICATIONS

## VI-1-PROGRAMMATION DU TIMER0

Comme nous l'avons dit (au chapitre IV-8-3) c'est le TIMER0 qui va fournir l'horloge de base à l'AIC, pour cela il faut programmer le timer pour fonctionner en mode impulsion avec le *period register* égale à 1, TCLK0 en sortie et sélectionner une horloge interne. Cela nous donne une fréquence AIC MCLK=7.5Mhz.

```
**** programmation timer ******
        .data
timer0  .word   808020h
tgcv    .word   2c1h
tcv     .word   0h
tpv     .word   1h

        .text
        ldp     timer0
        ldi     @timer0,ar0
        ldi     0,r0
        ldi     @tcv,r1
        ldi     @tpv,r2
        ldi     @tgcv,r3
        sti     r0,*ar0
        sti     r1,*+ar0(4)
        sti     r2,*+ar0(8)
        sti     r3,*ar0

*****************************
```

## VI-2-PROGRAMMATION DU PORT SERIE

Les broches CLKX/R, DX/R et FSX/R doivent être programmées comme *serial port pins*, FSX comme une entrée, sélectionner une horloge de transmission/réception externe, sélectionner le mode variable de transmission/réception, sélectionner le mode standard pour la *frame sync* avec FSX/FSR actives au front bas, transfert de 16 bits.

```
***** programmation port série *******

        .data
serial0 .word   808040h
comdata .word   804000h
pgcv    .word   0c170300h
xpcv    .word   0111h
rpcv    .word   0111h

        .text
        ldi     @serial0,ar0
        ldi     @comdata,ar1
        ldi     @xpcv,r0
        ldi     @rpcv,r1
```

```
        ldi   @pgcv,r2
        sti   r0,*+ar0(2)
        sti   r1,*+ar0(3)
        sti   r2,*ar0
```

**********

## VI-3-PROGRAMMATION DE L'AIC

La programmation de l'AIC consiste à choisir la fréquence d'horloge des filtres numériques, la fréquence d'échantillonnage et les différents contrôles. Dans la pratique la fréquence d'échantillonnage est typiquement 8 Khz pour la parole, comme MCLK=7.5Mhz on prend TA=13 et TB=36 (f=8012,82hz).

```
********** programmation AIC ********************
        ldi   2,r2
*** setup secondary communications:

        ldi   3h,r7
waitx11 and   *ar0,r2,r0
        bz    waitx11
        sti   r7,*+ar0(8)

*** select 288khz for the scfclk:

        ldi   1a34h,r7
waitx12 and   *ar0,r2,r0
        bz    waitx12
        sti   r7,*+ar0(8)

*** setup secondary communications:

        ldi   3h,r7
waitx21 and   *ar0,r2,r0
        bz    waitx21
        sti   r7,*+ar0(8)

*** select the sampling rate:

        ldi   4892h,r7
waitx22 and   *ar0,r2,r0
        bz    waitx22
        sti   r7,*+ar0(8)

*** setup secondary communications:

        ldi   3h,r7
waitx31 and   *ar0,r2,r0
        bz    waitx31
        sti   r7,*+ar0(8)
```

*** select controls:

```
            ldi   2a7h,r7
waitx32     and   *ar0,r2,r0
            bz    waitx32
            sti   r7,*+ar0(8)
************************************************
```

## VI-4-GENERATION D'UN COSINUS

Nous allons générer un cosinus par la formule de récurrence :

$$\cos(\omega \cdot n \cdot T_e) = 2\cos(\omega \cdot T_e) \cdot \cos(\omega \cdot (n-1)T_e) - \cos(\omega \cdot (n-2)T_e)$$

avec $\omega$ la pulsation du cosinus et $T_e$ la période d'échantillonnage.
qui devient en remplaçant $\cos(\omega \cdot m \cdot T_e)$ par y(m) :

$$y(n) = a \cdot y(n-1) - y(n-2)$$

avec $a = 2\cos(\omega \cdot T_e)$ et
$y(0) = 1$

$$y(1) = \cos(\omega \cdot T_e)$$

| $T_e$ | $\omega$ | $y(0)$ | $y(1)$ | a |
|---|---|---|---|---|
| 0,12 ms | $2\pi \cdot 500$ | 1 | 0,93 | 1,96 |

Le cosinus généré sera sorti à travers l'AIC (pour entendre le son dans un HP).

********** génération d'un cosinus **********

```
**** programmation timer ******
        .data
timer0  .word  808020h
tgcv    .word  2c1h
tcv     .word  0h
tpv     .word  1h

        .text
        ldp    timer0
        ldi    @timer0,ar0
        ldi    0,r0
        ldi    @tcv,r1
        ldi    @tpv,r2
        ldi    @tgcv,r3
        sti    r0,*ar0
        sti    r1,*+ar0(4)
        sti    r2,*+ar0(8)
        sti    r3,*ar0
```

```
************ reset AIC ***************************

        ldi    2,iof

************ programmation port série **************

          .data
serial0   .word  808040h
comdata   .word  804000h
pgcv      .word  0c170300h
xpcv      .word  0111h
rpcv      .word  0111h

          .text
          ldi    @serial0,ar0
          ldi    @comdata,ar1
          ldi    @xpcv,r0
          ldi    @rpcv,r1
          ldi    @pgcv,r2
          sti    r0,*+ar0(2)
          sti    r1,*+ar0(3)
          sti    r2,*ar0


********* mise or reset de l'AIC *****************

     ldi    6,iof

*********************************************
********** programmation AIC *****************
            ldi    2,r2
*** setup secondary communications:

            ldi    3h,r7
waitx11     and    *ar0,r2,r0
            bz     waitx11
            sti    r7,*+ar0(8)

*** select 288khz for the scfclk:

            ldi    1a34h,r7
waitx12     and    *ar0,r2,r0
            bz     waitx12
            sti    r7,*+ar0(8)

*** setup secondary communications:

            ldi    3h,r7
waitx21     and    *ar0,r2,r0
            bz     waitx21
            sti    r7,*+ar0(8)
```

```
*** select the sampling rate:

            ldi   4892h,r7
waitx22     and   *ar0,r2,r0
            bz    waitx22
            sti   r7,*+ar0(8)

*** setup secondary communications:

            ldi   3h,r7
waitx31     and   *ar0,r2,r0
            bz    waitx31
            sti   r7,*+ar0(8)

*** select controls:

            ldi   2a7h,r7
waitx32     and   *ar0,r2,r0
            bz    waitx32
            sti   r7,*+ar0(8)
************************************************

************* génération d'un cosinus ***********
*formule utilisée: y(n)=a y(n-1) - y(n-2).
************************************************
*r0=y(n);r1=y(n-1);r2=y(n-2).
************************************************
y0          .set  1.0
y1          .set  0.96
a           .set  1.93
            ldi   2,r7
            ldf   y0,r2
            ldf   y1,r1
suite       ldf   r2,r0       ;r0=y(n-2)
            mpyf  -1.0,r0     ;r0=-y(n-2)
            ldf   a,r4
            mpyf3 r4,r1,r3    ;r3=a*y(n-1)
            addf  r3,r0       ;r0=a*y(n-1) - y(n-2)
            ldf   r0,r3
            mpyf  250.0,r3    ;pour ne pas avoir de problème avec fix
            fix   r3,r3       ;r3=y(n) mais en entier pour la transmission
            mpyi  20,r3
waitx       and   *ar0,r7,r6
            bz    waitx
            sti   r3,*+ar0(8)
            ldf   r1,r2       ;r2=y(n-1)=y(n-2) futur
            ldf   r0,r1       ;r1=y(n)=y(n-1) futur
            bu    suite

********** fin de programme *********************
```

## VI-5-NUMERISATION DE LA PAROLE

Dans ce programme nous allons échantillonner un signal de parole (on branche un microphone sur la carte) puis lui faire un traitement quelconque (au minimum une amplification) puis le ressortir à travers un HP.

La partie programmation des périphériques ne change pas, nous donnant ici le code qui n suit.

```
************ réception et transmission de données *****
            ldi    1,r1
            ldi    2,r2
waitx       and    *ar0,r2,r0
            bz     waitx

********** Sortie des données ******************
            sti    r3,*+ar0(8)   ;vers Aic (sur HP)
************************************************

waitr       and    *ar0,r1,r0
            bz     waitr
            ldi    *+ar0(12),r3
            mpyi   10,r3          ;amplification (échantillon * 10)
            bu     waitx

********** fin de programme *********************
```

## VI-5-EFFETS SPECIAUX 'TRUQUEUR DE VOIE'

On pourrait utiliser les deux programmes précédents pour moduler la parole par notre cosinus et obtenir ainsi une voie de robot. L'effet de truquage sera différent selon la fréquence du cosinus.

```
************ truquage de voie ***************
y0          .set   1.0
y1          .set   0.93
a           .set   1.96
            ldi    1,r6
            ldi    2,r7
            ldf    y0,r2
            ldf    y1,r1
************* transmission du signal truqué ******
suite:
waitx       and    *ar0,r7,r5
            bz     waitx
            sti    r3,*+ar0(8)
******** génération du cosinus ******************

            ldf    r1,r2        ;r2=y(n-1)=y(n-2) futur
            ldf    r0,r1        ;r1=y(n)=y(n-1) futur

            ldf    r2,r0        ;r0=y(n-2)
            mpyf   -1.0,r0      ;r0=-y(n-2)
            ldf    a,r4
```

```
            mpyf3  r4,r1,r3     ;r3=a*y(n-1)
            addf   r3,r0        ;r0=a*y(n-1) - y(n-2)
            ldf    r0,r3
            mpyf   250.0,r3     ;pour ne pas avoir prob avec fix
            fix    r3,r3        ;r3=y(n) mais en entier pour la transmit
*           mpyi   50,r3        ;amplification

************ réception du signal d'entrée *******

waitr       and    *ar0,r6,r5
            bz     waitr
            ldi    *+ar0(12),r4

************ modulation par le cosinus **********

            mpyi r4,r3

            bu     suite

************ fin de programme **********************
```

## VI-6-LE PROJET 'OSCILLOSCOPE NUMERIQUE'

Nous proposons maintenant, de faire la communication entre le DSP et le PC en illustrant cela par un projet (développement de deux programmes, une partie DSP et une partie PC sous MS visuel C++ 4.0) qui consiste en l'échantillonnage de la parole et visualisation en 'temps réel' sur l'écran d'un ordinateur (temps réel signifie un retard d'une trame).

### VI-6-1-PARTIE DSP

Dans la partie DSP, il suffit de rajouter trois lignes pour la transmission au PC dans le programme de numérisation de la parole.

Au début du bloc 'transmission et réception de données' :

```
comdata         .word  804000h      ;adresse du registre de communication
                ldi    @comdata,ar1 ;AR1 va contenir cette adresse
```

et dans le bloc 'sortie des données' :

```
                sti    r3,*ar1      ;vers comdata (PC)
```

### VI-6-2-PARTIE PC

Le problème qui se pose pour la réalisation d'un tel projet est comment faire pour accéder à l'EVM (ses registres) sous Windows ?. En effet, sous DOS tous les compilateurs C offraient un couple de fonctions d'accès aux ports 16 bits d'E/S :

**inpw** et **outpw** pour les compilateurs Microsoft.
**inportw** et **outportw** pour les compilateurs BORLAND.

Mais sous Windows BORLAND a carrément enlevé ces fonctions, et Microsoft ne les définit que pour des applications console. Ceci nous a conduit à les réécrire en assembleur (Intel).

**Fonctions d'accès à l'EVM**

```
void evmwrite(unsigned short adr, unsigned short data)
{
__asm
{
mov dx,adr;
mov ax,data;
out dx,ax;
}

}


short evmread(unsigned short adr)
{
short data;
__asm
{
mov dx,adr;
in  ax,dx;
mov data,ax;
}
return data;
}
```

**Réception de donnée**

La réception d'une donnée est faite selon les étapes décrites dans le chapitre IV-6 :

```
////////////////////////////////
while(evt1==0)
{
//-1)Update the tbc status reg

evmwrite(evmbase+0x14,0x6044);

//-2)Test evt1:
evt1=evmread(evmbase+0x0400);
evt1=evt1 & 0x0002;
}
////////////////////////////////

//-3)Clear evt1
evt1=0;
evmwrite(evmbase+0x14,0x0002);
```

```
 //-4)Read the data //*/
data=evmread(evmbase+0x0808);

 ////////////////////////////////////////
```

'data' est la variable (short) destinée à recevoir l'échantillon et 'evmbase' l'adresse e base de l'EVM (=0x0240 si le switch n'a pas été modifié).

Avec un composant timer (un timer de Windows) on peut afficher périodiquement nos échantillons et obtenir notre oscilloscope (figure VI-1).

Oscilloscope numérique

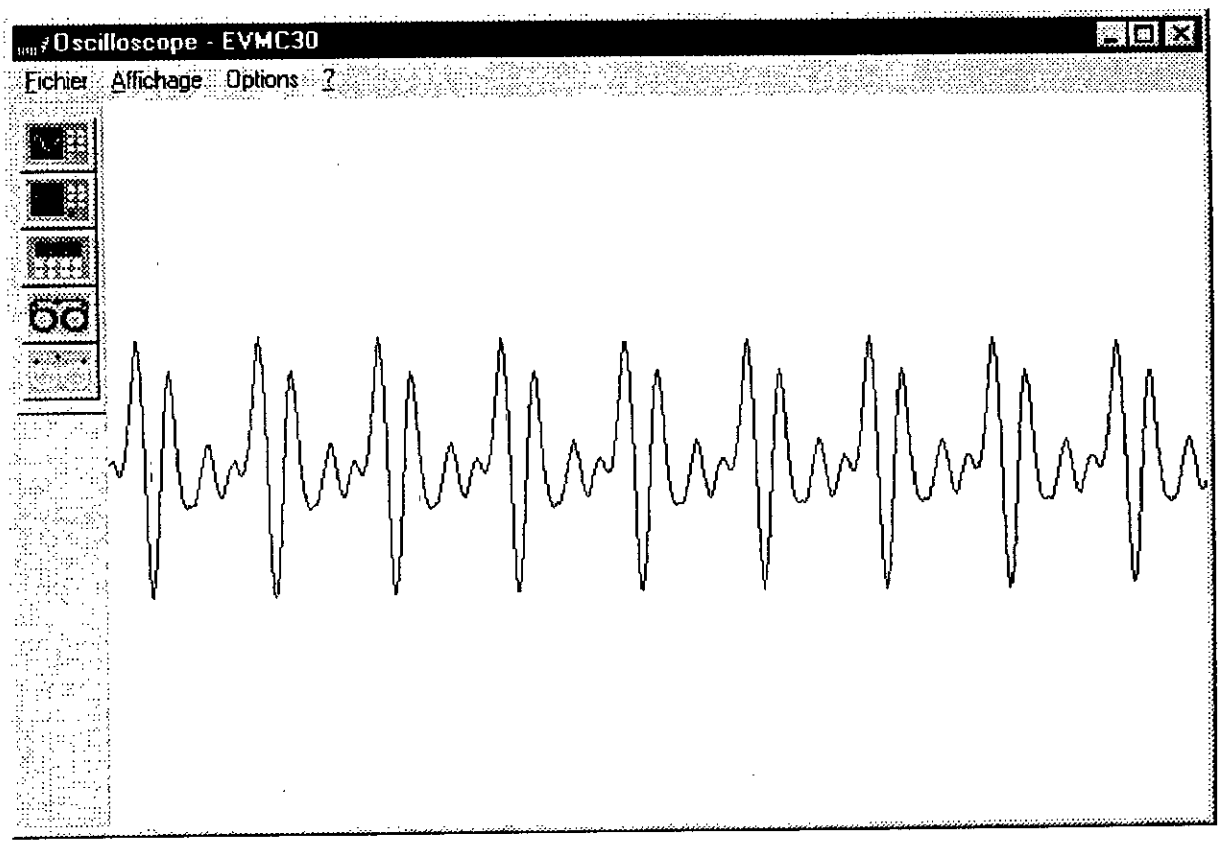



Figure VI-1

# VI-7-UTILISATION DU COMPILATEUR C

Dans tous les programmes C que nous présenterons, nous utilisons un modèle mémoire large (nécessaire pour nos pointeurs). A la compilation il faut utiliser l'option '– mb'.

### VI-7-1-ROUTINE D'INITIALISATION DES PERIPHERIQUES

Nous avons réécrit tous les programme assembleur en C, et nous les avons rassemblés dans une routine 'initperiph' dont le fichier source est 'initp.c'.

```
void initperiph(void)
{
/*===== routine d'initialisation des périphériques =====*/

/*********** programmation timer ************/

*(unsigned int *)0x808020=0x00;
*(unsigned int *)0x808024=0x00;
*(unsigned int *)0x808028=0x01;
*(unsigned int *)0x808020=0x2c1;

/************ reset AIC ************************/

asm("      ldi 2,iof      ");


/************ programmation port serie *********/

*(unsigned int *)0x808042=0x0111;
*(unsigned int *)0x808043=0x0111;
*(unsigned int *)0x808040=0x0c170300;

/*********** reset de xdr ************************/

*(unsigned int *)0x808048=0x00;

/********** mise or reset de l'AIC **************/

asm("          ldi 6,iof      ");


/****** désactivation des interruptions cpu *******/

asm("      ldi 0,IE       ");


/*********** programmation AIC ******************/
/*** setup secondary communications :*/

while((*(unsigned int *)0x808040 && 0x02)==0) ;
```

```
*(unsigned int *)0x808048=0x03;

/*** select 288khz for the scfclk :*/

while((*(unsigned int *)0x808040 && 0x02)==0) ;
*(unsigned int *)0x808048=0x1a34;

/*** setup secondary communications :*/

while((*(unsigned int *)0x808040 && 0x02)==0) ;
*(unsigned int *)0x808048=0x03;


/*** select the sampling rate :*/

while((*(unsigned int *)0x808040 && 0x02)==0) ;
*(unsigned int *)0x808048=0x4892;

/*** setup secondary communications :*/

while((*(unsigned int *)0x808040 && 0x02)==0) ;
*(unsigned int *)0x808048=0x03;

/*** select controls :*/

while((*(unsigned int *)0x808040 && 0x02)==0) ;
*(unsigned int *)0x808048=0x02a7;

/*========= fin de routine ============*/
}
```

### VI-7-2-ROUTINES D'ACCES AIC ET PC

Ces routines sont disponibles dans le programme 'Iorts.c'.

```
unsigned short AIC_read(void)
{
unsigned short data;

while ((*(unsigned int *)0x808040 && 0x01)==0) { ;}
data=*(unsigned int *)0x80804c;
return data;
}


void AIC_write(unsigned short data)
{

while ((*(unsigned int *)0x808040 && 0x02)==0) { ;}
*(unsigned int *)(0x808048)=data;
}
```

```
unsigned short PCread(void)
{
unsigned short data;
data=*(unsigned int *)0x804000;
return data;
}
void PCwrite(unsigned short data)
{
*(unsigned int *)0x804000=data;
}
```

Grâce à ces routines, la programmation de l'EVM devient très facile, avec un code très réduit. Si l'on réécrivait la partie DSP du projet 'Oscilloscope' le programme serait :

```
#include"iorts.h"
#include"math.h"

void main(void)
{

int i;
unsigned short sample;

initperiph();

for(i=0;;i++)  /* boucle sans fin */
{

sample=10*AIC_read(); /* le 10 est une amplification */
AIC_write(sample);
PCwrite(sample);

}


}
```

# Conclusion

# VII-CONCLUSION

Bien que l'EVM soit une carte d'évaluation, elle est quant même préformante et est destinée à faire des applications telles que le codage de la parole, l'annulation d'écho, modem téléphonique. Il ne faut pas oublier qu'elle contient un circuit d'interface analogique de haute qualité et que son DSP est plus puissant qu'un 'Pentium' dans les calculs arithmétiques (malgré que le TMS320C30 soit assez vieux). Le seul inconvénient de l'EVM est sa mémoire très limitée qui cause rapidement des problèmes de saturation lors de la programmation en C.

La pratique des DSP fut pour nous comme une sorte de stage final qui nous a ouvert le monde des systèmes numériques, il nous a permit de voir ce que c'était ces processeurs spéciaux très utilisés dans l'industrie de haute technologie que sont les DSP, et il nous a appris la manipulation de différentes sortes de circuits numériques comme les convertisseurs analogiques/numériques, les ports série, les timers et de faire la coordination entre eux.

Le travail que nous avons fait a été de faciliter l'étude des DSP de la famille TMS320C3X en rassemblant en un minimum de page toutes les connaissances indispensables à leur utilisation, en donnant des exemples de programmation et en fournissant des routines en C pour l'exploitation de la carte d'évaluation. Les futurs programmeurs n'auront qu'à étudier notre thèse (comme première approche), parfaire leurs connaissances de la documentation de TI et n'écrire que le code de traitement des données dans leurs applications.

Cette étude manque d'applications professionnelles (car nous ne le sommes pas), nous n'avons pas le bagage d'en faire une. Un des aspects les plus importants de la programmation des DSP n'a pas été exploité, il s'agit du travail en mode intérruptible et de l'utilisation du contrôleur DMA. Nous avons négligé volontairement ces possibilités car leur emploi en C demande une bonne documentation que nous n'avions pas [5] (le C de ces DSP à été étendu pour pouvoir utiliser les interruptions).

Nous espérons que cette thèse sera d'une grande utilité à ceux qui utiliseront ce matériel après nous et que les futurs utilisateurs la compléteront et l'enrichiront.

# Bibliographie

# BIBLIOGRAPHIE

1- TMS320C3X User's Guide (spru031E.pdf)

2- TMS320C30 Evaluation Module Technical Reference (spru069.pdf)

3- TMS320 Floating-Point DSP Assembly Language Tools User's Guide (spru035B.pdf)

4- TLC32044 data sheet (slas017f.pdf)

5- TMS320C3X/C4X Optimizing C Compiler User's Guide (spru0034H.pdf)

6- TMS320C3X C Source Debugger User's Guide (spru053B.pdf)

7- Les processeurs de traitement du signal famille 320C5X –G.Baudoin et F.Viroleau-Dunod

8- TMS320C3X Evaluation Module Installation Guide (spru120.pdf)

9- TMS320C3X/C4X Code Generation Tools Getting Started Guide (spru119B.pdf)

Toute la documentation que nous avons citée (sauf la référence [7]) se trouve sur le site Internet **TI.com**. A droite de chaque référence vous avez le nom du fichier à télécharger.

Les références [4] et [5] ne sont pas disponibles à l'école.