Carte contrôleur et logiciel embarqué pour la commande de systèmes médicaux

David Guiraud, Bernard Gilbert

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Carte contrôleur et logiciel embarqué pour la commande de systèmes médicaux

Guiraud David, Gilbert Bernard

N° 0320

Mai 2005

THÈME BIO

INRIA Research Report model

Guiraud David[1], Gilbert Bernard[2]

Thème BIO – Systèmes biologiques
Projet DEMAR

Rapport technique n° 0320– Juin 2006 - 28 pages

Résumé: Une carte électronique et son logiciel ont été conçus afin de piloter divers systèmes électroniques prototypes ou industriels tels que des stimulateurs, des implants, et des plates-formes de mesures. La sécurité de fonctionnement, la facilité de programmation ont été obtenues par abstraction de l'accès aux couches bas niveaux. Par ailleurs l'optimisation du code en termes de compacité et de rapidité d'exécution ont été prises en compte, et enfin la consommation d 'énergie a fait l'objet d'une étude particulière.

Mots clefs: microcontrôleurs, logiciels embarqués, systèmes médicaux autonomes

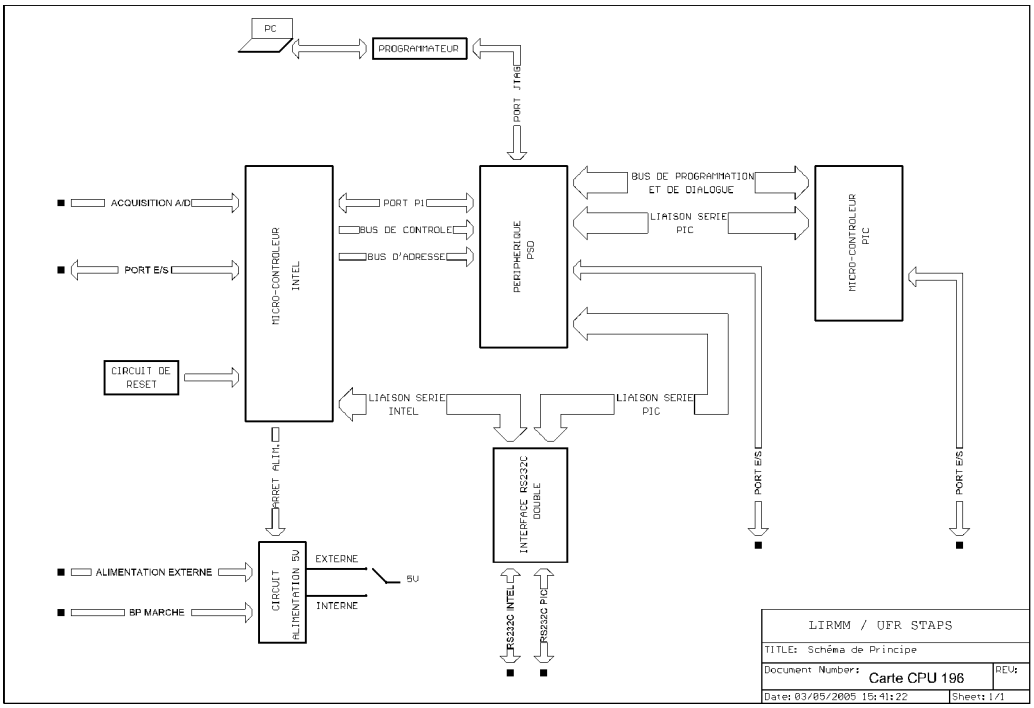
[1] Guiraud David Chargé de Recherches INRIA, projet DEMAR, UR Sophia Antipolis – David.Guiraud@inria.fr
[2] Gilbert Bernard Assistant Ingénieur, Université de Montpellier I, UFR STAPS – Bernard.Gilbert@lirmm.fr

Versions logicielles du 08/06/2006, version globale licenciée 1.0,

- CPU196 **5.00, graticiel non décrit dans ce document**
- 80C196 **3, licence APPIDDN.FR.001.090024.000.S.P.2005.000.10000**
- Pic **2, Graticiel non décrit dans ce document, logiciel développé en fonction de l'application. Seule l'interface C196 PIC est décrite.**

- **Matériel**

Cette carte CPU196 fonctionne sous le principe du Maître/Esclave. Le Maître (Microcontrôleur INTEL) gère entièrement la partie Esclave (Microcontrôleur PIC). Il peut programmer le PIC et ensuite donner des ordres à partir d'un Bus de dialogue. Cette carte est entièrement autonome. En effet, elle possède sa propre alimentation ainsi qu'un port de communication Série pour dialoguer avec un PC.



La carte CPU196 est composée de 6 ensembles (schéma de principe).

- Mode test : le cavalier JP1 place la CPU en mode test, elle force notamment le protocole RS232 CPU196.
- Le circuit d'alimentation
  1. Il est constitué d'un circuit L4931 de ST MICROELECTRONICS. Les cavaliers JP3, JP4 et JP5 permettent de sélectionner 2 modes de fonctionnement : Externe ou Interne.
  2. JP3/JP5 ouverts et JP4 fermé : La carte CPU196 est alimentée par une alimentation externe de 5V via la pin <+ALIM> du connecteur J1.
  3. JP3/JP5 fermés et JP4 ouvert : La carte CPU196 est alors alimentée par le circuit L4931. L'entrée de la broche <+ALIM> doit être comprise entre 6 et 8V. Le circuit L4931 peut être mis en veille par le Microcontrôleur Maître en utilisant le signal HSO.3. Dans ce cas, la broche <BPON> sert à mettre en marche le circuit L4931. Le signal <BPON> doit être mis à la masse temporairement.

Remarque : On peut forcer le signal <BPON> à la masse, mais dans ce cas on ne peut plus utiliser la mise en veille du circuit L4931 par le microcontrôleur.

- Le circuit Reset

Il est constitué du circuit DS1813 de DALLAS SC. Ce composant génère un signal </RESET> actif au niveau bas d'une durée de 150ms.
Remarque : On peut générer en externe un Reset en utilisant un bouton poussoir connecté entre le signal </RESET> et la Masse.

- Le Microcontrôleur INTEL 30C196KC30

Nous avons utilisé le Microcontrôleur 80C196KC de chez INTEL. C'est un Microcontrôleur fonctionnant à partir d'un Bus en 16 Bits.
Il dispose de :

1. 8 Entrées Analogiques de <ACHO> à <ACH7>.
2. 1 Bus multiplexé d'Adresses et de Données en 16 Bits
3. 1 Port de Contrôle.
4. 1 Liaison Série Asynchrone.
5. 1 Port P1 quasi-bidirectionnel.
6. 1 Port rapide d'entrées logiques <HSI.0>, <HSI.2> et <HSI.3>.
7. 1 Port rapide de sorties logiques connecté en interne.
8. 3 PWM .

Ce Microcontrôleur fonctionne avec un quartz de 20MHz et ne possède pas de mémoire de programme.

- Le périphérique de Microcontrôleur

Le 80C196KC est interfacé avec le PSD4235G2 de ST MICROELECTRONICS. Le PSD augmente les capacités de traitement du 80C196KC, il permet de ce fait les extensions suivantes :

1. 4 Mbits de mémoire Flash.
2. 256 Kbits pour le démarrage et le stockage de données.
3. 64 Kbits de SRAM.
4. 1 Bloc interne de logique combinatoire.
5. 1 Bloc interne et externe de logique séquentielle.
6. 7 Ports d'Entrées et de Sorties logiques.

D'autre part le PSD4235G2 bénéficie de technologie JTAG. Il intègre la fonction ISP (In-System-Programming) permettant le téléchargement du noyau de 80C196. Les broches <TDO>, <TCK>, <TMS> et <TDI> de l'interface JTAG se connectent sur un Programmateur relié lui même à un PC.
Le PSD4235G2 permet la redirection du Port P1 du 80C196KC et de la RS du PIC. Il assure aussi le dialogue avec le PIC par l'intermédiaire du Bus de Programmation et de Dialogue.
Toute programmation des Ports (Direction) et de la PLD du PSD ne peut se faire que lors du téléchargement du noyau.

- Le Microcontrôleur PIC16F877A

Nous avons choisi le PIC 16F877A qui est en fait le Microcontrôleur Esclave associé au 80C196KC.
Le Microcontrôleur PIC 16F877A dispose de :

1. 8 Entrées analogiques.
2. 1 Port parallèle esclave.
3. 1 Port de 3 Bits d'Entrée et de Sortie bi-directionnels.
4. 2 PWM.

5. Des Entrées/Sorties qui peuvent être configurées au choix en :

1. Liaison Série Asynchrone.
2. Liaison Série Synchrone.
3. Liaison I2C.

La programmation du PIC 16F877A est réalisée à partir des signaux <HSO.0>, <HSO.1> , <PG6> et <PG7> associés respectivement à /MCLR (Reset), PGM , CLK et DATA du Microcontrôleur.
Remarque : Le cavalier JP2 doit être retiré lors de la programmation en 12V du PIC 16F877A. La broche /MCLR est alors reliée à un signal 12V externe.

- L'Interface Série RS232

Le circuit LT1180 de LINEAR TECHNOLOGY permet de convertir les signaux TTL en signaux conformes à la Norme RS232C. Les signaux <PTXD196> et <PRXD196> du 80C196KC ainsi que les signaux <PTXDPIC> et <PRXDPIC> du PIC 16F877 peuvent être directement connectés à tout périphérique comportant une interface série RS232C.
Remarque : Le signal <HSO.2> du 80C196 permet de mettre en veille le circuit LT1180.

Schéma électronique

- ## Affectation des ressources

1. Mémoires

1. Microcontrôleur 80C196 (version KC)

Mémoire disponible :
- RAM interne et registres 512k : 0h - 1FFh
- RAM externe 8k
- Flash principale externe 8x64k, les flashs sont accessibles à partir de 300h
- Flash boot externe 4x8k

L'espace mémoire adressable par le micro contrôleur est de 64k avec une distinction possible entre mémoire programme en lecture uniquement (code) et mémoire donnée. La pagination et le mapping est donc contrôlé non seulement par le bus d'adresse du microcontrôleur (16bits), et le signal inst (fetch d'une instruction), mais aussi par un registre externe **csiop.Page**. Ce dernier contient les bits :

Bit 7 **0**	Bit 6 **0**	Bit 5 **0**	Bit 4 **0**	Bit 3 **0**	Bit 2 **0**	Bit 1 **0**	Bit 0 **0**
NA	*NA*	*Data2*	*Data1*	*Data0*	*Pgr2*	*Pgr1*	*Pgr0*

On a ainsi le mapping mémoire :

Inst	0	1
0 – 1FFh	RAM interne et registre 196	Boot 0 (code)
200h – 2FFh	csiop	Boot 0 (code)
300h – 3FFh	Flash Pgr (réservée noyau)	Boot 0 (code)
400h – 1FFFh	Boot 3 (effaçable réservée noyau)	Boot 0 (code)
2000h – 27FFh	Boot 1	
2800h – 3FFFh	Flash Pgr (2800h-29FFh réservés noyau)	Boot 1 (code)

4000h – 5BFFh	RAM externe	Boot 2 (code)
5C00h – 5FFFh	RAM externe	
6000h – 7FFFh	Flash Pgr	
8000h – FFFFh	Flash Data	Flash Pgr

Remarque : la programmation de boot 3 nécessite la suspension des interruptions (programme en boot non accessible). Boot 1, 2 et 3 continennent le noyau et les constantes que l'utilisateur ne peut pas reprogrammer mais il peut faire appel à certaines procédures.

Le noyau utilise une partie de ces ressources de sorte que les zones mémoires laissées à l'utilisateur sont :

- En RAM interne **72h-1C7h**
- En RAM externe **4000h-5AFFh**. Attention cependant à la pile descendante dont la base est 5B00h, elle empiète par le haut sur la mémoire d'au moins **2Ch** octets nécessaires pour le fonctionnement du noyau.
- En flash **2A00h-3FFFh, 6000h-FFFFh**.

2. Configurations des registres 80C196

Les registres du 80196 sont normalement configurés par le noyau qui lui même extrait les informations stockées dans une zone de la boot 3 ; c'est le programme CPU196 qui va programmer, en fonction des configurations souhaitées par l'utilisateur, les valeurs correctes dans ces registres. Néanmoins, certaines valeurs sont figées et dans tous les cas, **aucun registre ne doit être modifié par une programmation directe de l'utilisateur**. Un x signifie que seul ce bit est programmable via le noyau.

ccr valeur 20CBh, 1 wait state, signaux wrh wrl, lu une seule fois au démarrage en flash protégée.

hsi_mode valeur 55h toutes les entrées hsi déclenchent sur un front montant (hsi_mode[2i..2i+1]=01).

ioc0 valeur 00h par défaut : tous les hsi sont dévalidés T2CLK(ioc0.7=0) et T2RST(ioc0.5=0) sont sur les pattes T2RST T2CLK (pas hsi.0 hsi.1) le reset externe est dévalidé (ioc0.3=0), ioc0.1 mis à 1 déclenche un reset Timer 2. les hsi.i sont validés si ioc0[2i] est mis à 1.

ioc1 valeur (0010 001x) T1 et T2 overflow interdits (ioc1[2..3]=00), TXD validé pour la RS (ioc1.5=1), hsi_int pour HoldinRegister loaded (ioc1.7=0), extint sur P0.7 (ioc1.1=1) ou P2.2, hso4 et hso5 dévalidés (ioc1.4=0 ioc1.6=0), PWM0 validé si ioc1.0=1.

ioc2 valeur (0101 0x00) T2 fast si ioc2.0=1, T2up_down (ioc2.1=0 up seulement), Slow PWM si ioc2.2=1, Adtime si ioc2.3=1, Adfast si ioc2.4=1, T2ovr sur échelle moitié si ioc2.5=1, LockCAM si ioc2.6=1, ClrCAM si ioc2.7=1.

ioc3 avec wsr=1, valeur (0000 xxx1), T2clk interne si ioc3.0=1, Clockout validée si ioc3.1=0, PWM1 et PWM2 on si ioc3.2 ioc3.3=1.

intmask (00xx xxx0) AD complete (intmask.1), software timer (intmask.5), int user (intmask.4), hsi pour timer (intmask.2), hso out (intmask.3).

intmask1 (0010 0011) TI & RI (intmask1[0..1]), extint1 (intmask1.5).

3. Configurations des registres PIC 16F877A

2. Ports d'entrées sorties

1. Ports du 80196

P0 AD[0..7]
Ports analogiques 0-5V 10 bits, ou entrées logiques dont l'état se trouve dans le registre **ioport0**. Ce port est directement accessible sur le connecteur.

P1

Bit 7 **1**	Bit 6 **1**	Bit 5 **1**	Bit 4 **1**	Bit 3 **1**	Bit 2 **1**	Bit 1 **1**	Bit 0 **1**
IO	*IO*	*2x4 En*	*PWM2/O*	*PWM1/O*	*RsPicRedir*	*Rs196Redir*	*Clock En*
PF.7	*PF.6*	*PF.5*	*PF.4*	*PF.3*	*PF.2*	*PF.1*	*PF.0*

Clock En=0 l'horloge à 10MHz est sortie sur la patte PA.7 sinon P1.7 est sorti sur PA.7.

Rs196Redir=0 redirection des données RS 196 vers PB.0 pour RX PB.1 pour TX, sinon T2Reset sur PB.0 PWM0 sur PB.1.
RsPICRedir=0 redirection des données RS PIC vers PB.2 pour RX PB.3 pour TX, sinon vers PB.6 pour RX sur PB.7 pour TX.

Ces deux configurations permettent de diriger (1) ou non (0) le signal des liaisons série vers le MAX232. Il est possible ainsi de configurer les liaisons séries pour qu'élles puissent s'interfacer avec des niveaux TTL ou RS232. Une configuration est cependant impossible : la RS du PIC en niveau RS232 et la RS du 196 en niveau TTL. Les trois autres configurations sont disponibles.

2x4 En=0 active le décodeur 2x4 dont les entrées sont PB.4 et PB.5 et les sorties PA[0..3]. Sinon PA.0 recopie PB.4 et PA.1 recopie PB.5.

Seules P1.7 P1.6, et éventuellement (si les PWM ne sont pas actifs) P1.4 et P1.3 peuvent être utilisées en sorties logiques avec précaution car il faut préserver l'état des autres sorties de ce port. Les autres sorties sont gérées par le noyau.

P2

Bit 7 **O**	Bit 6 **1**	Bit 5 **1**	Bit 4 **1**	Bit 3 **1**	Bit 2 **1**	Bit 1 **1**	Bit 0 **1**
LED	*Moniteur*	*PWM0/O*	*T2Rst / I*	*T2Clock /I*	*Int Pic*	*RX196*	*TX196*
NA	*Con.*	*PC.6*	*PC.5*	*PC.7*	*RA.4*	*PC.0*	*PC.1*

LED qui s'allume en cours de programmation flash.

Moniteur=1 mode normal. *Moniteur*=0 mode forcé moniteur. Dns ce dernier cas la configuration de base est forcée pour que le noyau prenne la main sur le logiciel utilisateur.

P2.3 et P2.4 peuvent servir d'entrées logiques et P2.5 de sortie logique si le PWM0 n'est pas utilisé et si a RS196 n'est pas redirigée. Les autres IO sont utilisées par le noyau.

HSIO

Hso.0 MCLR du pic
Hso.1 RB.3 PGM du pic
Hso.2 Activation circuit RS 12V
Hso.3 Switch off logiciel de toute la carte
Hsi.0 Entrée interruption externe utilisateur
Hsi.1 NA
Hsi.2 Entrée horloge timer 1 ou sortie horloge timer 1
Hsi.3 Entrée horloge timer 2 ou sortie horloge timer 2

Toutes ces entrées sorties sont utilisées et configurées par le noyau et **ne doivent pas être directement programmées par l'utilisateur**.

Sur l'ensemble des ports du 196 **il est déconseillé de les programmer directement**. Néanmoins, le port P0, P1.3, P1.4, P1.6, P1.7, P2.3, P2.4, P2.5 peuvent être utilisés sous certaines conditions en prenant soin de ne pas modifier les autres IO des mêmes ports.

2. Ports du PIC16F877A

RA

Bit 5	Bit 4	Bit 3	Bit 2 **1**	Bit 1 **1**	Bit 0 **1**
AD4	*Int 196*	*AD3*	*AD2*	*AD1*	*AD0*
Con.	*P2.2*	*Con.*	*Con.*	*Con.*	*Con.*

A part **RA.4** réservé pour le fonctionnement du noyau, chaque port peut être utilisé en entrée analogique, en émulation d'un port d'adresse, ou en entrée sortie logique. Les deux premiers cas sont gérés par la noyau, le troisième doit être géré par l'utilisateur et les autres fonctions désactivées.

RB

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Prog ISP/ Data3	*Prog ISP/ Data2*	*Data 1*	*Data0*	*PGM*	*Ctr2*	*Ctr1*	*Ctr0*
PG.7	*PG.6*	*PG.5*	*PG.4*	*Hso.1*	*PG.2*	*PG.1*	*PG.0*

Port réservé à la programmation du PIC et à l'échange de données avec le 80C196. Voir la partie logiciel.

RC

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RX	*TX*	*SDO*	*SDI/SDA*	*SCL*	*RC2 Sens RX*	*RC1/PWM3*	*Ctr3*
PB.6	*PB.7*	*Con.*	*Con.*	*Con.*	*PB.5*	*PB.4*	*PG.3*

Tous les ports sont utilisables par l'utilisateur sous réserve que le noyau ne les configure pas pour des fonctions spéciales, à l'exception de **RC.0**, réservé.

RD

8 bits accessibles sur le connecteur configurés soit en IO soit en port data maître µP.

RE

Bit 2 **1**	Bit 1 **1**	Bit 0 **1**
CS /AD7	*WR / AD6*	*RD / AD5*
Con.	*Con.*	*Con.*

Selon que l'on utilise le port D en IO ou en port maitre µP les ports sont soit configurés en entrées AD (port D en IO) soit en bus de contrôle.

3. Ports du PSD

PA

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ClockOut / P1.7	*P1.6 / T2C-lock*	*PWM2*	*PWM1*	*CS3*	*CS2*	*CS1*	*CS0*
Con.	*Con.*	*Con.*	*Con.*	*Con.*	*Con.*	*Con.*	*Con.*

Port non accessible à l'utilisateur il sort certains signaux redirigés des microcontrôleurs.

PA.0=(PB.4 + PB.5 + PF.5).(PB.4+!PF.5)
PA.1=(!PB.4 + PB.5 + PF.5).(PB.5+!PF.5)
PA.2=(PB.4 + !PB.5 + PF.5)
PA.3=(!PB.4 + !PB.5 + PF.5)
PA.4=PF.3
PA.5=PF.4
PA.6=PF.6
PA.7= (ClkIn + PF.0) & (PF.7 + !PF.0)

PB

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TXPIC	*RXPIC*	*RC2 / RX-Pic sens*	*RC1*	*TXCPIC*	*RXCPIC*	*TXC196 / PWM0*	*RXC196 / T2Rst*
RC.6	*RC.7*	*RC.2*	*RC.1*	*Con.*	*Con.*	*Con.*	*Con.*

Port non accessible par l'utilisateur.

PB.1=(PC.1 + PF.1).(PC.6 + !PF.1)
PB.2= PB.6 + PF.2
PB.2.OutpuEnable= !PB.5.!PF.2.PF.5
PB.3= PB.7 + PF.2
PB.6= (PB.2.!PF.2) + (PC.2.PF.2)
PC.6.OutputEnable= RC2 + PF.2 + !PF.5

PC

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T2Clock	*PWM0*	*T2Rst*	*NA*	*TXDPIC*	*RXDPIC*	*TX196*	*RX196*
P2.3	*P2.5*	*P2.4*	*NA*	*MAX232.*	*MAX232.*	*P2.0 / MAX232.*	*P2.1 / MAX232.*

Port non accessible par l'utilisateur.

PC.0= PB.0 + PF.1
PC.0.OutputEnable= !PF.1
PC.3= PB.6 + !PF.2
PC.3.OutputEnable=PF.2
PC.5= PB.0 + PF.1
PC.5.OutputEnable=PF.1
PC.7=PA.6

PF

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ICPLD	*ICPLD*	*ICPLD*	*ICPLD*	*ICPLD*	*ICPLD*	*ICPLD*	*ICPLD*
P1.7	*P1.6*	*P1.5*	*P1.4*	*P1.3*	*P1.2*	*P1.1*	*P1.0*

Port non accessible par l'utilisateur. Port uniquement en entrée, pas d'équation logique.

PG

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MCUIO	*MCUIO*	*MCUIO*	*MCUIO*	*MCUIO*	*MCUIO*	*MCUIO*	*MCUIO*
RB.7	*RB.6*	*RB.5*	*RB.4*	*RC.0*	*RB.2*	*RB.1*	*RB.0*

Port servant à l'échange des données entre 196 et PIC, pas d'équation logique, accès direct par les registres du PSD (csiop.IG, csiop.CG, csiop.OG, csiop.DG, csiop.SG voir documentation PSD PSD). **Il ne faut pas le programmer directement**.

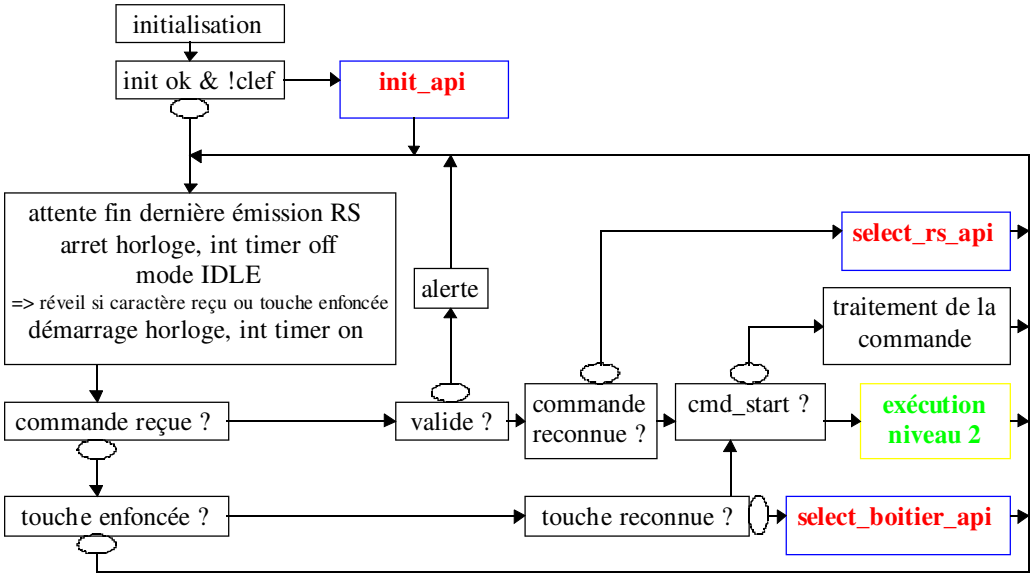
PD, **PE** réservés pour des signaux de contrôle.

En utilisation normale aucun registre du PSD ne doit être modifié par l'utilisateur.

- ## Logiciels embarqués

1. Noyau du 80C196

1. Structure

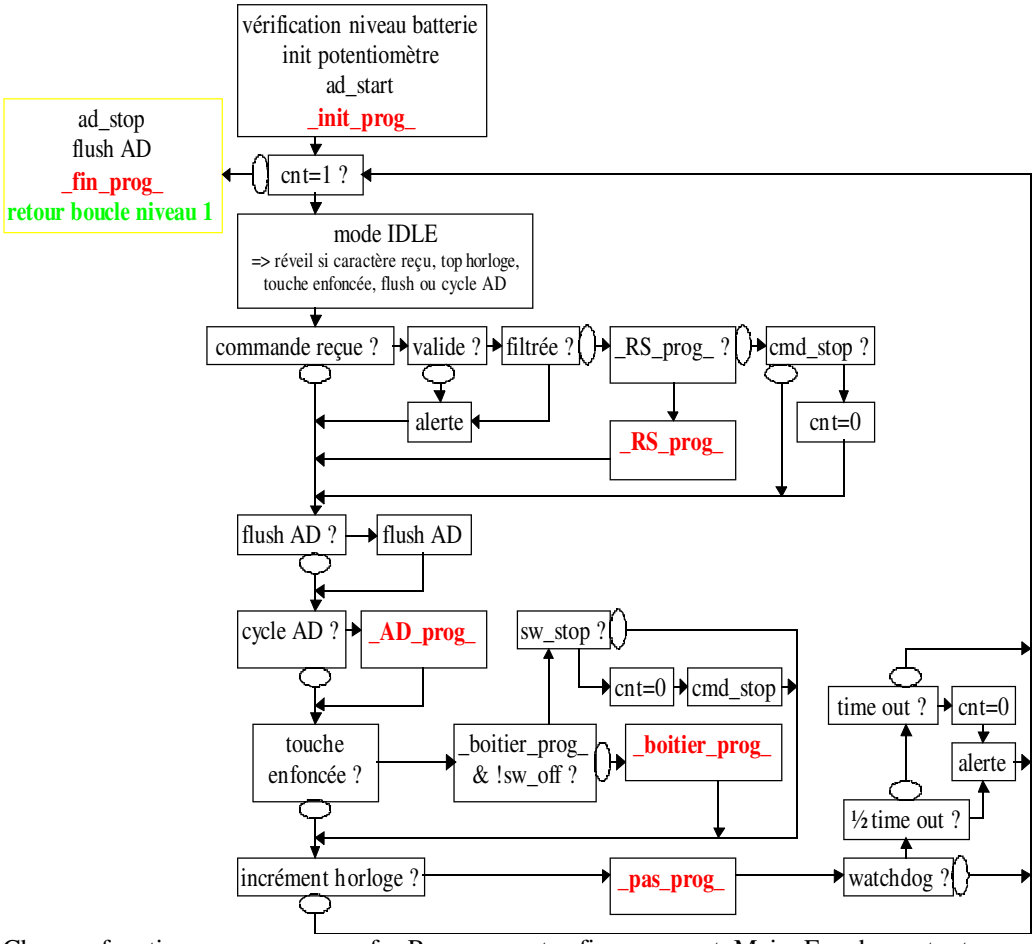


Les procédures en rouge sont les procédures que l'utilisateur peut programmer, par défaut elles sont vides. En fait, elles sont connues du noyau par leur adresse t appelées automatiqueent lorsque l'événement correspondant est déclanché. Toutes les fonctions sauf _Main_Fin _Main_RS et _Main_Fread, retournent un entier non signé sur 16 bits pouvant déclencher un Start. _NoStart_ (0x8000) ne déclenche rien, sinon il faut respecter le format de la commande _CMD_START_. A la suite de la procédure _Main_Fin, le processeur est soit ré initialisé, soit placé en powerdown. Dans les deux cas toutes les variables sont réinitialisées et le processeur redémarre au début du programme en mode reset..

_Main_Int est appelée pendant l'interruption. On ne doit utiliser cette procédure que si cette interruption est critique en temps ou priorité, mais il faut suivre une programmation qui tienne compte des particularités de la gestion des interruptions. Il vaut mieux utiliser –Main_User qui est appelée par le séquenceur au même niveau de priorité que les autres événements, sans perturber les autres procédures d'interruptions.

Un start fait entrer le programme dans une boucle de niveau inférieur qui présente les mêmes caractéristiques que la boucle de niveau supérieur avec les différences suivantes :

- le processeur n'est pas mis en veille
- certaines fonctions du noyau sont désactivées (cf. procédures)
- les AD du 196 sont démarrés
- les timers du 196 sont démarrés



Chaque fonction en rouge, sauf _Rs_prog_ et _fin_prog_ et_Main_Fread, peut stopper l'exécution de la boucle grâce à l'entier sur 8 bits retourné par la fonction (voir fichier moniteur.h *_ContExecXX_* ou *_StopExecXX_*). A noter que les Timers 1 et 2 ne sont activés que si les fonctions _pas_progX_ ont été programmées (même vide).

La procédure d'interruption _Main_Int reste active si elle est programmée.

Dans les deux boucles, si le protocole 196 n'est pas activé, aucun message n'est transmis ni compris par le 80196. L'utilisateur gère dans ce cas totalement l'émission et la réception RS.

2. Données

Ne sont pas décrits les registres propres au 80196 dont on peut trouver la définition dans la documentation du microcontrôleur et dans la première partie du document pour ceux dont la valeur a une importance particulière.

1. Constantes

Elles sont stockées en Flash EPROM de sorte qu'il est impossible de les écrire par affectation. L'adresse figure entre parenthèses pour les constantes localisées de manières absolue.

BYTE **Version_moniteur (2016h)** version du programme dans le noyau
BYTE **Quartz (2014h)** vitesse du Quartz, toujours 20MHz les autres valeurs sont obsolètes
WORD **reset_196 (201Ah)** contient le code instruction reset
WORD **tab_int2 (2000h), tab_int3[2] (200Ch), tab_int4[2] (2010h)**, **tab_int5[3] (2034h), tab_int6[2] (203Ch)** tableaux d'adresses pointant sur l'instruction reset pour les interruptions non utilisées.
BYTE **_commande_invalide_[9]** liste des commandes invalides dans la boucle inférieure.
BYTE **filtre_a[9]** valeur des coefficients du filtre FII butterworth passe bas premier ordre.

Toutes les constantes suivantes sont uniquement reprogrammables par le noyau.

ARRAY **_bloc_[256] (300h)**, répertorie les blocs de 256 octets encore vierges en flash EPROM utiliBYTE **numero_serie[16] (400h)** numéro de série sur 8 caractères direct et codé.
ARRAY **_compression_[8] (410h)**, contient le paramètre de la compression des AD.
WORD **tab_lin_log16_8[256] (600h)**, contient les valeurs pour la compression 16 vers 8 bits.
STRUCTURE **csboot[256bytes] (500h)**
{
unsigned info_valide; AA55h si les infos de cette structure sont valides
struct {
unsigned Protocole196:2; MIDI 0, CPU196 1, aucun 2, OFF 3
unsigned ProtocolePIC:2; MIDI 0, CPU196 1, aucun 2, OFF 3
unsigned Hardware:1; 1 RS232 en marche, 0 TTL uniquement
unsigned Watchdog196:1; transmission du watchdog ou MIDI sensing
unsigned WatchdogPIC:1; transmission du watchdog ou MIDI sensing
unsigned Bauds196:3; vitesse 1200 2400 9600 19200 31250 38400 57600 115200 250000
unsigned Adresse:6; adresse MIDI (0-16) ou pipe
} rs;
unsigned boot;
sefm thhh pabbbnnn, meme signification que la commande _CMD_START_ avec en plus
s=1pas de start s=0 start
e=1pas d'ad Pic maintenu e=0 ad Pic maintenu
f=1pas de timer pic maintenu, f=0 timer pic maintenu
struct { unsigned char pwm[4],p1masque; } c96inf;
c96inf.pwm[0..2], valeurs initiales, c96inf.pwm[3].4 vitesse (rapide si 0), PWM1/2 on si c96inf.pwm[0].(2..3)=1, PWM0 on si c96inf.pwm[3].0=1, ClockOut on si c96inf.pwm[3].1=0, les autres bits sont à 0
c96inf.p1masque, masque du port P1, voir définition port P1
struct { unsigned char pwm[4],cfg[2],cycle[2],adon[2];
unsigned char adrs,seuil[3],adr,vref; } picinf;
picinf.pwm[2] valeurs initiales du PWM1 du pic sur 10 bits, les deux bits de poids faibles sont en picinf.pwm[1] aux 2 bits de poids fort, tous les autres bits sont à 0.
picinf.cycle, voir commande wr_pic_startT dans le protocole C196-Pic
picinf.adon, masque des ad PIC en marche pour poids faible adon, et poids fort adon donne la fréquence d'échantillonnage voir protocole C196-Pic commande wr_pic_startAD
picinf.adrs, masque des ad PIC transmis par rs

picinf.seuil[3] définit les trois seuils de détection batterie par ordre croissant 0xFF si on ne veut pas utiliser cette fonctionnalité, voie 0 Pic uniquement.

picinf.cfg, voir protocole PIC. Si tous les bit à 1 valeur invalide, pic non configuré.

unsigned char info_user[231] ; utilisation libre

}

STRUCTURE **prginf[256bytes] (2800h)**

{

unsigned info_valide; AA55h si les infos de cette structure sont valides

unsigned char hso[8]

8 fois la même signification, génération d'un top horloge synchrone avec le software timer 1 (resp. 2) si hso[n].4 =1 (resp hso.6=1) tous les autres bits à 0.

unsigned char ad_init[8][18] 8 fois la même structure dont la signification est la suivante

unsigned char ad_init[n][0..7]

séquencement inverse (il faut décrémenter) des 16 canaux codés sur poids faible puis poids fort exceptée la première valeur qui est effectivement la première valeur (dernier chargement CAM) on a en fait le séquencement (0 15) (14 13) (12 11) (10 9) (8 7) (6 5) (4 3) (2 1)

unsigned char ad_init[n][8..15]

Suite décrémentée avec pour chaque octet (donc pour les 8 voies en sens inverse)

(0scc ffff) : s=1 donnée transmise, cc format (00) 16 bits, (01) 10 bits, (10) 8 bits, (11) 8 bits log centré si par ailleurs r=0 et s=0 (qui sera remis à 1 dans le programme cpu)

si 0≤ffff≤8 filtrage passe bas de rang ffff, si 9≤ffff≤15 filtrage moyenneur sur 2 puissance ffff-9 échantillons (de 1 à 32)

unsigned char ad_init[n][16] base de temps T qui donne finalement une période d'échantillonage de (75+5*T)x0.8µs

char ad_init[n][17] puissance de deux du rapport entre l'interruption et le cycle pts donc de 1:1 à 1:16 (cas normal)

unsigned char adon[8][2], cf csboot

unsigned char adrs[8], cf csboot

unsigned char cycle[8][2]; cf csboot

unsigned cycle1[8],

8 fois la même signification, attention ne fonctionne que si la procédure user existe, temps de cycle en pas timer 1250 pas par ms. Si cycle1 vaut 0 le timer déclenche sur hsi.2.

unsigned cycle2[8]; idem cycle1 mais pour le deuxième software timer

}

STRUCTURE **ptrprg[256bytes] (2900h)**

{

unsigned (*init_api)(void), (*select_user_api)(void), (*select_rs_api)(void), (*select_pic_api)(void);

void (*fin_api)(void), (*int_api)(void), (*fread_api)(void);

struct {

unsigned char (*_init_prog_)(void), (*_user_prog_)(void), (*_RS_prog_)(void);

unsigned char (*_pic_prog_)(void), (*_pas_prog1_)(void), (*_pas_prog2_)(void), (*_AD_prog_)(void);

void (*_fin_prog_)(void);

} user[8];

unsigned reserved[57];
}
Cette structure contient les jeux de pointeurs vers les fonctions utilisateurs que le noyau appelle selon l'événement déclencheur.

2. Variables

Les adresses sont précisées lorsque les variables sont placées de manière absolue.

STRUCTURE **csiop[≡256bytes] (200h)**
{
 unsigned char IA,IB;
 unsigned char reserved0203[2];
 unsigned char OA,OB,DA,DB,SA,SB,IMCA,IMCB,EA,EB;
 unsigned char reserved0E0F[2];
 unsigned char IC,ID;
 unsigned char reserved1213[2];
 unsigned char OC,OD,DC,DD,SC,SD,IMCC;
 unsigned char reserved1B;
 unsigned char EC;
 unsigned char reserved1D_1F[3];
 unsigned char OMCA,OMCB,MMCA,MMCB;
 unsigned char reserved24_2F[12];
 unsigned char IE;
 unsigned char reserved31;
 unsigned char CE;
 unsigned char reserved33;
 unsigned char OE;
 unsigned char reserved35;
 unsigned char DE;
 unsigned char reserved37;
 unsigned char SE;
 unsigned char reserved39_3F[7];
 unsigned char IF,IG;
 unsigned char CF,CG;
 unsigned char OF,OG;
 unsigned char DF,DG;
 unsigned char SF,SG;
 unsigned char reserved4A4B[2];
 unsigned char EF;
 unsigned char reserved4D_AF[99];
 unsigned char PMR0;
 unsigned char reservedB1_B3[3];
 unsigned char PMR1;
 unsigned char reservedB5_BF[11];
 unsigned char Protect;
 unsigned char reservedC1;
 unsigned char BProtect;
 unsigned char reservedC3_C6[4];

```
 unsigned char JTAG;
 unsigned char reservedC8_DF[24];
 unsigned char Page;
 unsigned char reservedE1;
 unsigned char VM;
 unsigned char reservedE3_EF[13];
 unsigned char ID0,ID1;
 unsigned char reservedF2_FF[14];
}
```

La signification des différents champs est disponible dans la documentation du PSD mais ces registres ne doivent pas être programmés directement.

PTR **_user_prog_ (1E8h), _init_prog_ (1EEh), _pic_prog_ (1F0h), _pas_prog1_ (1F4h), _pas_prog2_ (1F6h), _AD_prog_ (1EAh), _RS_prog_ (1F2h), _fin_prog_ (1ECh)**
Pointeurs 16 bits vers les functions user de la boucle inférieure. Ils sont initialisés par le noyau.

UNION **_AX_ (6Ch), _BX_ (6Eh), _CX_ (68h), _DX_ (6Ah)**

```
{
 unsigned X;
 struct { unsigned char L,H } O;
}
```

Registres essentiellement utilisés par le noyau, mais dans certains cas ils contiennent des retours de fonctions (voir les fonctions).

BYTE **_page_ (2Ah), _pic_commande_ (2Bh), ad_C1, _cfg_, _pic_version_**
Variables utilisées par le noyau. Il ne faut pas les modifier. Néanmoins elles peuvent être lues si nécessaire. Elles renseignent alors respectivement sur la page flash en cours, la dernière commande PIC envoyée, le nombre de paquet ad restants, la configuration courante, et enfin les versions logicielle et matérielle du PIC.

UNION **_EX_**

```
{
 unsigned char C[4];
 unsigned short S[2];
 unsigned long L;
}
```

Variable utilisée par le noyau. A ne pas modifier.

UNION **_flag_**

```
{
    struct {
     unsigned EvenementUser:1;
     unsigned EvenementWT:1;
     unsigned EvenementWTOVR:1;
     unsigned EvenementT1:1;
     unsigned EvenementT2:1;
     unsigned EvenementAD:1;
     unsigned EvenementPic:1;
     unsigned Cont:1;
```

```
        } bit;
    unsigned char octet;
 }
```

Variable utilisée par le noyau. A ne pas modifier.

```
STRUCTURE _flag1_
{
 unsigned Batterie:2; // 0 dechargee, 1 faible, 2 ok, 3 secteur
 unsigned TI_flag:1;
 unsigned ADStop:1;
 unsigned ADOn:1;
 unsigned WTCount:3;
 }
```

Variable utilisée par le noyau. A ne pas modifier.

```
STRUCTURE ri_pts
{
 unsigned p_wr_in,p_rd_in;
}
```

Pointeur de la FIFO de la RS C196 en entrée. A ne pas modifier.

WORD **pic_buf, _periode_, _periode1_, _periode2_, ad_n_FIFO, _fichier_ouvert_ (70h)**
Registres utilisés par le noyau pour les communications avec le PIC, pour la gestion des horloges logicielles et de la FIFO AD196. A ne pas modifier.

PTR **p_rd_out, p_wr_out**
Pointeurs pour la FIFO de la liaison série C196 en sortie. A ne pas modifier.

```
STRUCTURE ad
{
 unsigned n_echantillon,wr_in,rd_in;
 unsigned char *p,count,cpl,base_de_temps;
}
```

Variable utilisée par le noyau. A ne pas modifier.

```
STRUCTURE ad_pts (1F8h)
{
 unsigned char count,control;
 unsigned data,org;
 unsigned char C0,i_cpl;
}
```

Variable utilisée par le noyau. A ne pas modifier.

```
STRUCTURE ad_pic
{
 unsigned donnee[8];unsigned char masque;
}
```

Contient les données acquises sur les AD du PIC. Mis à jour automatiquement par le noyau.

STRUCTURE **ad_voie[8]**
{
 unsigned donnee;
 unsigned char filtre,n;
}
Variable dont le champ donnee, contient la donnée acquise sur les AD du C196 mise à jour automatiquement par le noyau.

BYTE **_commande_**
Contient la commande courante.

BYTE **_car_rs_**
Contient le caractère reçu ou à émettre sur l'une des liaisons série.

BYTE **_time_out_**
S'incrémente une fois par milliseconde.

WORD **flash_1000 (7000h), flash_XAAA (6AAAh), flash_X554 (6554h)**
Registres utilises pour la programmation de la flash, à ne pas modifier.

3. Procédures du noyau non appelables

void int_pic(void)
procédure de gestion de l'interruption PIC.

void int_api_user(void)
procédure de gestion de l'interruption externe.

void increment_horloge_hsi(void)
procédure de gestion de l'interruption horloge logicielle sur horloge externe.

void increment_horloge_hso(void)
procédure de gestion de l'interruption horloge logicielle avec génération d'une horloge externe.

void increment_horloge(void)
procédure de gestion de l'interruption horloge logicielle.

void transmit(void)
procédure de gestion de l'interruption de émission RS 196.

void receive(void)
procédure de gestion de l'interruption de réception RS196.

void ad_lecture(void)
procedure de gestion de l'acquisition des AD196.

cstart (2080h)
Adresse de départ du processeur appelle directement main.

void main(void), main196(void), mainSansProtocole(void)
main est une procédure du noyau qui initialise l'ensemble des registres et variables. Elle appelle ensuite l'une des deux autres procédures selon que l'on utilise ou pas le protocole RS 196.

void execute_programme196(void), execute_programmeSansProtocole(void)
Ces procédures gèrent la boucle de niveau inférieur en utilisant ou non le protocole de communication entre PC et C196.

void PWM(void), I2CSPI(void)
Ces deux procédures gèrent respectivement les 4 PWM et les liaisons série du PIC en utilisant le protocole de communication entre PC et C196.

char testNS(void)
retour
-1 si le numéro de série est correct, sinon 0.

char test_numero_serie_crypte(void)
retour
0 si le numéro de série crypté envoyé par le PC est bon, sinon renvoie 1 (cryptage non valide), 2 (numéro de série non valide), 3 (numéro de série et cryptage non valides), 4 (erreur de communication).

void call_ram(void) (5F00h)
appel direct en RAM de différentes fonctions de gestion de la flash boot chargées selon les besoins (effacement ou programmation)

void programme_boot(void)
procédure de programmation d'un mot de la zone flash boot. _CX_.X doit contenir le mot à programmer, _DX_.X l'adresse paire de destination.

void efface_boot(void)
procédure d'effacement de la zone flash boot.

void transfert_RS_flash(void)
programme ou lit 256 octets de la flash en utilisant le protocole de communication entre PC et C196.

void pic_version(void)
initialise les variables _pic_version_ _pic_revision_.

void pic_reset(void)
reset le PIC et le positionne en mode run.

void pic_prg(void)
reset le PIC er le positionne en mode programmation ICSP.

void pic_serial_send(char)

envoie les données série de programmation ICSP.

void pic_serial_receive(void)
réceptionne les données série de la programmation ICSP.

void pic_prog_erase(void), pic_data_erase(void)
efface respectivement les zones flash et EEPROM du PIC par programmation ICSP.

void pic_erase(void)
efface les deux zones par programmation ICSP.

void pic_unlock(void)
efface les deux zones et reinitialise le mot de config du PIC par programmation ICSP.

unsigned char pic_event196(void), pic_eventSansProtocole(void)
teste si le PIC à un événement à traiter et gère cet événement. La fonction retourne 0 si aucun événement est traité, 1 sinon. Les deux versions diffèrent par l'envoi ou non d'information sur la RS196 en utilisant le protocole de communication entre PC et C196.

void transfert_RS_picinterne(char)
programmation ICSP de la flash du PIC en utilisant le protocole de communication entre PC et C196.

void transfert_picinterne_RS(char)
lecture ICSP de la flash du PIC en utilisant le protocole de communication entre PC et C196.

void transfert_RS_piceeprom(char)
programmation ou lecture ICSP de l'EEPROM du PIC en utilisant le protocole de communication entre PC et C196.

void horloge_stop(void)
arrête toutes les horloges logicielles.

void horloge_start_execute(void)
démarre les horloges logicielles 1 et 2.

void horloge_start(void)
démarre l'horloge logicielle pour le time out.

void rs_init(void)
Initialise la RS196 en function de la programmation faite via le logiciel PC.

unsigned char reception_commande(void)
retour
msg_time_out, pas de commande complète (commande et complément) reçue dans le temps imparti.
msg_reception_error, erruer sur la complément la commande est non valide.
msg_no_error, pas d'erreur

procédure d'attente d'une commande. Elle vérifie la validité de la commande (réception de la commande et de son complément) et n'est pas bloquante (time out). Elle stocke la commande dans la variable _commande_, et retourne le message renvoyé en se servant du protocole entre PC et 196 (sauf s'il n'y a aucune erreur). Elle gère la commande CMD_WATCHDOG.

void ad_stop(void)
arrête les AD 196 en fin de boucle inférieure.

void ad_flush(void)
vide régulièrement le buffer AD et transmet les données en utilisant le protocole entre PC et 196.

void config_in196(void), config_inSansProtocole(void)
configure les AD 196 en gérant les erreurs (fréquence d'échantillonage trop importante par exemple). L'erreur est renvoyée en utilisant le protocole entre PC et 196 dans la première fonction.

unsigned char ad_config(void)
configure les registres et variables pour les AD 196. LA fonction retourne 0 si il n'y a pas de problème, 1 sinon (AD non configurés et qui ne seront pas mis en marche).

void ad_start(void)
démarre les AD 196 en début de boucle inférieure.

4. Procédures du noyau appelables par l'utilisateur et macros

unsigned char transfert_sram_flash(unsigned char bank, unsigned char base)
retour
msg_invalid_parameter_value, mauvaise zone flash, impossible d'écrire ou lire
msg_not_blank_error, zone flash non efface, impossible d'écrire
msg_no_error, pas d'erreur
base, contient les 8 bits de poids fort de l'adresse de base en flash.
bank, contient le numéro de la bank (0-7), le bit 3 doit être à 1 pour une écriture, à 0 sinon.
transfert des données entre flash et mémoire RAM (à partir de l'adresse _flash_buffer_) par paquets de 256 octets.

void efface_flash(void)
efface la bank flash (0 à 7) don't le numéro est stocké dans préalablement _CX_.O.L.

void affiche (unsigned char n,unsigned char format,unsigned char *tab)
envoie n caractère(s) ou mot(s) stocké(s) dans *tab dans le format spécifié (décrit dans PRO196.h). Le moniteur PC affiche dans la fenêtre d'application les octets envoyés. **Ne fonctionne que si le protocole entre PC et C196 est activé.**

void fwrite(unsigned char f,unsigned char n,unsigned char *tab)
écrit dans le fichier f n caractère(s) ou mot(s) stocké(s) dans *tab. **Ne fonctionne que si le protocole entre PC et C196 est activé.**

unsigned char fopen(unsigned char n ,char *nom,unsigned char mode)
ouvre un fichier nommé nom de longueur n en lecture (mode=0) ou écriture (mode=1). **Ne fonctionne que si le protocole entre PC et C196 est activé.**

void fclose(unsigned char f)
ferme le fichier f. **Ne fonctionne que si le protocole entre PC et C196 est activé.**

void pic_wrc(unsigned char commande)
envoie un octet de commande au PIC. La procédure est bloquante si le PIC plante.

void pic_wrb(void)
envoie un octet de donnée stocké préalablement dans pic_buf. La procédure est bloquante si le PIC plante.

MACRO associée
pic_wr(c), idem mais place c dans pic_buf.

unsigned char pic_rd(void)
Renvoie un caractère venant du PIC Renvoie 0 en cas d'erreur. La procédure est bloquante si le PIC plante.

void rs_FIFO_wr(void)
envoie le caractère stocké dans _car_rs_ dans la pile d'émission. La fonction est bloquante.

MACROS associées
emission_caractere(c), émet le caractère c.
emission_commande(c), émet la commande c et son complément. **Ne fonctionne que si le protocole entre PC et C196 est activé.**

unsigned char rs_FIFO_rd(void)
renvoie 0 si un caractère est disponible, 1 sinon. Le caractère reçu est placé dans _car_rs_.

unsigned char reception_caractere(void)
retour
msg_time_out, pas de caractère reçu dans le temps imparti.
msg_no_error, caractère reçu sans erreur.
La fonction attend un caractère sur la RS mais avec un time out. Le caractère reçu est dans _car_rs_. **Ne fonctionne que si le protocole entre PC et C196 est activé.**

MACROS qui ne fonctionnent que si le protocole entre PC et C196 est activé.

emission_canal_virtuel_16bits(n,d), utilise le protocole entre PC et 196 pour envoyer une valeur 16 bits d, sur un canal n reconnu par le moniteur.
emission_canal_virtuel_12bits(n,d), idem pour une valeur 12 bits.
emission_canal_virtuel_8bits(n,d), idem pour une valeur 8 bits.

fread(f,n), demande une lecture de n octet dans le fichier f. Les octets reçus sont récupérés au travers de la fonction définie dans Main_Fread.

Autres MACROS

pwm0(c), pwm1(c), pwm2(c), pwm12(c1,c2), commande des 3 PWM du 196 sur 8 bits.

2. Protocole RS232, 80196 PC

La commande est envoyée suivie de son complément. Les commandes comprises entre 00 et EF sont disponibles pour les applications user. Dans la mesure du possible elles doivent avoir un format fixe.

Codes		Paramètres	Signification
F0 _CMD_THREAD_	2+n	00 n texte	Affiche du texte (max 256 caractères)
	2+2n	03 n data[2][1..2n]	Affiche une suite de nombre entier sur 16 bits en hexa
	2+2n	05 n data[2][1..2n]	Affiche une suite de nombre entier sur 16 bits non signé
	2+2n	07 n data[2][1..2n]	Affiche une suite de nombre entier sur 16 bits signé
	2+n	02 n data[1..n]	Affiche des octets en hexa
	2+n	04 n data[1..n]	Affiche des octets non signés
	2+n	06 n data[1..n]	Affiche des octets signés
			Le 3ième bit à 1 provoque un retour à la ligne sur les commandes d'affichage
	3+n	40 n nom[1..n] f	Ouverture d'un fichier en lecture
	3+n	41 n nom[1..n] f	Ouverture d'un fichier en écriture
	2	18 f	Fermeture d'un fichier f
	3	10 n f	Lecture dans le fichier f
	3+n	20 n data[1..n] f	Ecriture dans le fichier f
	3+n	30 n data[1..n] f	n octets lus dans f
F1 _CMD_CPUINF_	1	00	Demande
	23	01 Q V196 VPIC PICREV NS[0..7] Adr Valim Compression	réponse : Quartz, version noyau 196 & Pic, révision Pic, numéro de série, adresse, tension d'alim, valeur double de la compression log
F2 _CMD_START_	2	(pabb bnnn) (s00m thhh)	AD on a=1 / off a=0 (idem p pour le Pic), bank b[0..2], config n[0..2], bank haute h[0..2], timer Pic on t=1 / off t=0, s=1 pas de start, m=1 changement de page moniteur sans start
F3 _CMD_STOP_	0	Aucun	sortie du noyau de niveau 2
F4 _CMD_POWERDOWN_	1	00	PowerDown et arrêt alim
		01	Reset software
		02	la carte se met en PowerDown et a arrêté son alim
		03	la carte vient d'effectuer un reset

F5 _CMD_PRGFLASH_	3	00 Bank Bloc	demande du contenu de la flash N° bank (0 à 7) et bloc (0 à 255)
	257	01 Data[0..255]	réponse à la commande précédente
	259	02 Bank Bloc Data[0..255]	écriture de la flash N° bank N° bloc, réponse d'un message msg_completed
	2	03 Bank	effacement N° bank, réponse d'un message msg_completed
F6 _CMD_PRGPIC_	2	00 Bloc	demande du contenu du N° bloc (1 à 3)
	4097	01 Data[0..4095]	réponse à la commande précédente 1k
	4098	02 Bloc Data[0..4095]	écriture du N° bloc de 1k, réponse d'un message msg_completed tous les 256 octets (16 fois)
	1	03	demande lecture EEPROM
	257	04 Data[0..255]	réponse à la commande précédente
	257	05 Data[0..255]	écriture EEPROM, réponse d'un message msg_completed
F7 _CMD_CPUFORMAT_	9	00 NSC[0..7]	demande noyau Pic, numéro de série crypté, clef test valide
	4097	01 Data[0..4095]	réponse à la commande précédente
	4105	02 NSC[0..7] Data[0..4095]	écriture noyau Pic , numéro de série crypté, clef test valide
	17	03 NSC[0..7] New[0..7]	écriture numéro de série et effacement général flash et Pic, clef test valide
	2	04 Bloc	demande N° bloc (5 à 31) CS0
	257	05 Data[0..255]	réponse à la commande
	257	06 Data[0..255]	écriture CS0 bloc 5 uniquement
	1	07	effacement général flash boot
	521	08 Data[0..511] Comp[0..7]	programmation table de compression
	3	55 AA xy	uniquement après reset (<100ms) protocole 196 autorisé ou test_moniteur à 0 ou info non valide. si y=3 effacement flash, si y=C, effacement boot, et si x=F effacement des deux, si x=C effacement Pic, si x=A effacement Pic+prog haute tension et mise en veille (procédure exceptionnelle)
F8 _CMD_PWM_	2	(vv00 00nn) Valeur	Numéro de PWM (n=3 Pic 196 sinon) et bits de poids faible, bits de poids fort
		(0000 0100) Valeur	Valeur sur le DA PIC sur 5 bits

F9 _CMD_I2CSPI_	2	00 Octet	octet reçu SPI/I2C
		01 Octet	octet à envoyer SPI/I2C
		02 Octet	octet reçu SCI
		03 Octet	octet à envoyer SCI
		04 Octet	octet lu sur bus parallèle
		05 Octet	octet à envoyer sur bus parallèle
		06 Octet	adresse ou masque bus parallèle
	1	07	Demande lecture bus parallèle
FA _CMD_ADCONT_	3 à 257	Data[0..] Checksum	Data selon format, checksum (XOR des data)
FB _CMD_ADSAMPLE_	2	(000x xnnn) data	Voie n Pic, data est le poids fort, donnée complète sur 10 bits
		(1xxx xnnn) data	Voie n virtuelle, data est le poids fort, donnée complète sur 12 bits
FC _CMD_CFGAD_	2	00 (00bb bnnn)	Demande de config AD flash N° b[0..2] config N° n[0..2]
	22	01 196[0..17] Pic[0..2]	réponse à la commande précédente
FD _CMD_ALERTE_	1	Alerte	numéro du message de l'alerte (voir PRO196.h)
FE _CMD_MSG_	1	Message	numéro du message (voir PRO196.h)
FF _CMD_WATCHDOG_	0	Aucun	envoyé par le PC ou la carte CPU196.

3. Protocole PIC 80196

Les commandes acceptent de 0 à 2 paramètres, le bit 4 de la commande indique s'il s'agit d'une lecture ou d'une écriture. Les commandes 1x imposent une exécution immédiate par le PIC.

Principe du dialogue sur le port PG[0..7] / RB[0..2], RC.0, RB[4..7]
Data sur PG.2 PG[4..7] / RB.2 RB[4..7]
Pic occupé PG.3 / RC.0
PG.0 / RB.0 Requête CPU196 et poids fort / faible
PG.1 / RB.1 R/W

Codes		Paramètres	Signification
10 rd_pic_event		Event data	repérage sur les 3 bits de poids fort de l'événement
	1		pic_no_event 00 00
	2		pic_error_event (001x xxxx) xx numéro d'erreur
	2		pic_sci_event 40 data reçue
	2		pic_spi_event 60 data reçue
	2		pic_ad_event (100v vnnn) xx data sur 10 bits canal n
	1		pic_tx_event A0 le bus sci a fini d'émettre
	1		pic_comparator_event (110x xxc1c0) changement d'état des deux comparateurs
	2		pic_user_event (111x xxxx) xx, user
11 rd_pic_io	1	data	lecture du port D en mode µP ou IO
12 rd_pic_eprom	1	data	lecture de l'eeprom et autoincrément de l'adresse
02 wr_pic_reference	1	data	Écriture du Vref sur 5 bits

03 wr_pic_cfg	2	(ppio ovvv) (smmm rccc)	sci i on (1)-off (0) vvv vitesse, spi s vitesse, p[0..1] mode spi desc, spi mont, i2c, off, pwm o[0..1]=0 (0 rapide 1 moyen 2 lent 3 off) masque m[0..2] port D config ports ACDE 00x portD IO classique + 8 AD 01x portD µP + 5 AD RE(CS) en adresse x=1 RC1&2 = CS 10x portD µP + 1 AD RE(CS) RA(4) = adresse x=1 RC1&2 = CS 11x portD µP + 0 AD RE(CS) RA(5) = adresse x=1 RC1&2 = CS r=1 Vref redirigé vers RA2 c[0..2] 1xx comparateurs off 00x, comparateurs sur entrées 0 et 1 et ref sur 3 et 2 si x=1 3 sinon 01x, comparateurs sur entrées 0 et 1 ou 3 et 2 et ref interne
04 wr_pic_pwm	2	Num data	valeur sur 10 bits num=(dd00 0000) data est le poids fort
15 wr_pic_stopT	0		arrête le timer
16 wr_pic_stopAD	0		arrête les ad
07 wr_pic_startT	2	high (llll llpp)	cycle sur 16 bits pas 800ns*1 2 4 8 (selon pp[0..1])
08 wr_pic_startAD	2	(0000 pfff) masque	masque + fréquence, si p=1 Fosc/2048, sinon Fosc/(4096*2^f) pour l'ensemble des canaux soit divisé par 8 pour un canal
09 wr_pic_io_adr	1	adresse	adresse de l'esclave ou masque IO (0 en sortie, 1 en entrée)
0A wr_pic_eprom_adr	1	adresse	positionnement du pointeur d'adresse
0B wr_pic_sci	1	data	écriture sur le port sci
0C wr_pic_spi	2	Commande data	lecture / écriture sur le port spi / i2c
0D wr_pic_io	1	data	écriture sur le port D en mode maitre µP ou IO
0E wr_pic_eprom	1	data	écriture dans l'eeprom et autoincrément de l'adresse
0F wr_pic_user	1	data	écriture d'une donnée pour les programmes user

• Remarques

- Bug du compilateur / linker Tasking : passage d'un **char** (ou **unsigned char**) dans une procédure à partir d'un tableau de **char** à plusieurs dimensions déclaré dans une structure **const**, placée de manière absolue et contenant au moins un entier 16 bits. Le code généré est faux.
- Ne pas utiliser les tabulations de **switch** si **inst** est décodé.

• Références

[1] PSD. mcu.st.com/mcu

[2] PIC. www.microchip.com

[3] 80C196. www.intel.com/design/mcs96/