7/99

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de
la Recherche Scientifique

ECOLE NATIONALE POLYTECHNIQUE
ENP

Département D'électronique

PROJET DE FIN D'ETUDES

Pour l'obtention du diplôme d'ingénieur d'état
En électronique

THEME

REALISATION D'UNE INTERFACE RECEPTEUR MICRO-ORDINATEUR

Présenté par :
- DALI M.
- HIMA A.

Devant le jury d'examen composé de :

- Président : M^r BELOUCHRANI A.
- Promoteur : M^r MEHENNI A.
- Examinatrice : M^me MOUSSAOUI A.

Année Universitaire
1998/1999

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de
la Recherche Scientifique

ECOLE NATIONALE POLYTECHNIQUE
ENP

Département D'électronique

PROJET DE FIN D'ETUDES

Pour l'obtention du diplôme d'ingénieur d'état
En électronique

THEME

REALISATION D'UNE INTERFACE RECEPTEUR MICRO-ORDINATEUR

Présenté par :
- DALI M.
- HIMA A.

Devant le jury d'examen composé de :

- Président : M^r BELOUCHRANI A.
- Promoteur : M^r MEHENNI A.
- Examinatrice : M^me MOUSSAOUI A.

Année Universitaire
1998/1999

إهداء

﴿ أهدي هذا البحث المتواضع إلى والدي و والدتي الكريمين ﴾

﴿ وإلى إخوتي الأعزاء و أخص بالذكر الخنساء و مجيب الرحمن ﴾

﴿ وإلى كل الأهل و الأقارب و الأصدقاء جملة وتفصيلا ﴾

﴿ وإلى كل من ساهم من قريب أو من بعيد في إنجاز هذا البحث. ﴾

هيمة عبد القادر

إهداء

﴿ أهدي هذا البحث المتواضع إلى والدي و والدتي الكريمين
و جدتي ﴾

﴿ وإلى إخوتي الأعزاء و جميع أفراد العائلة ﴾

﴿ وإلى كل الأهل و الأقارب و الأصدقاء ﴾

﴿ وإلى كل من ساهم في إنجاز هذا البحث المتواضع ﴾

دالي محمد

Remerciement :

Nous tenons à présenter nos vifs remerciements à notre promoteur monsieur M.MEHENNI qui a proposé et dirigé ce modeste travail et à monsieur A. BELOUCHRANI de vouloir présider le jury et à madame A.MOUSSAOUI de bien accepter d'être examinatrice de notre travail.

A tous ceux qui ont participé d'une manière ou d'une autre à l'aboutissement de ce travail trouvent ici toutes nos gratitudes et nos vifs remerciements.

INTRODUCTION.

CHAPITER I. NOTION DE LA TRANSMISSION DE DONNEES.

CHAPITRE II. LES ORGANES DE TRANSMISSION.

CHAPITRE III. ETUDE DE MICRO-CONTROLEUR 68HC11.

CHAPITRE IV. DEVELOPPEMENT D'UNE APPLICATION A BASE DE MICRO-CONTROLEUR.

CHAPITRE V. PRESENTATION DE LA REALISATION.

Table des figures :

Table des tableaux :

ملخص:

يهدف هذا البحث إلى إنجاز بطاقة تسمح بوصل جهاز استقبال بالحاسوب و يتم ذلك عن طريق تحويل الإشارات المستمرة إلى إشارات رقمية أولا ثم تحويل هذه الأخيرة إلى معلومة متسلسلة يفهمها الحاسوب.

كما تمكن هذه البطاقة من الإستقبال المباشر للمعلومات الرقمية مع تكييفها توتريا وفقا للرستور RS232C.

ABSTRACT:

Our project consists of the realization of an interface card between the PC and a receiver. That card converts analog signals, coming from a receiver, to digital form to be sent to the PC. It permits also level adjusting for digital informations coming from a numerical system.

RESUME:

Le but de ce modeste travaille consiste en la réalisation d'une carte permet d'établir une liaison entre le PC et un récepteur. Cela se fait par la conversion de signal analogique provenant d'un récepteur en format numérique, puis faire véhiculer cette information vers le micro-ordinateur pour les traitements convenables. La carte permet aussi l'adaptation de niveau des informations numériques provenant d'un système digital.

Les mots clés :

Transmission synchrone, Transmission asynchrone, UART, Conversion analogique numérique, Transfert de données, micro-contrôleur, RS232.

INTRODUCTION

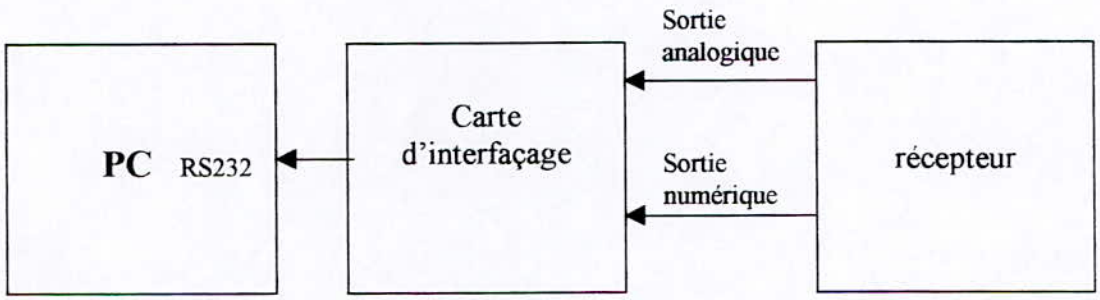
1-introduction :

La transmission des signaux, que se soit son type (analogique ou numérique), a fait l'objet de très nombreuses études. Ces études couvrent aujourd'hui un champ très vaste. Les résultats obtenus se sont rapidement concrétisés en pratique par l'amélioration de la conception des systèmes de transmission numérique.

Sur le plan matériel, le développement des transmissions numériques s'est appuyé sur les progrès rapides réalisés en matière technologique dans les circuits intégrés de traitement des signaux. Une synergie s'est ainsi établie entre la technologie disponible et les méthodes de traitement du signal numérique. Elle explique la généralisation de l'emploi des techniques numériques dans la mise en œuvre des nombreux services ainsi que dans la numérisation des signaux traditionnellement transmis en analogique.

De nos jours le micro ordinateur est un outil de traitement de données sophistiqué, mais comme ce dernier n'est pas prévu pour la transmission des données, il est indispensable de lui associer une carte d'interfaçage afin de lui permettre de bien recevoir et émettre les données en utilisant ces ports d'entrée/sortie (série ou parallèle).

Notre modeste travail entre dans le cadre d'étude puis de réalisation d'une interface entre un récepteur et un PC. Cette interface nous permet de faire adapter la sortie d'un récepteur (analogique ou numérique) à la norme RS232 dont le but est de permettre au PC la prise en charge du signal reçu : stockage, traitement, correction, affichage...



Emplacement de la carte d'interfaçage.

2-Présentation du projet :

Dans le cadre d'un projet qui à comme objectif la réalisation d'un système complet de communication (émetteur, récepteur et une interface pour PC) ; décomposé en deux parties :

- Partie réalisation d'un système de transmission standard : composé d'un émetteur et d'un récepteur, qui fait l'objet d'un projet de magister proposé au département d'électronique.
- Partie réalisation d'une interface entre un récepteur et un PC qui fait l'objet de notre travail.

Au premier chapitre nous présentons les notions de base de la transmission de données où ses différents modes sont détaillés. Nous décrivons de plus la norme utilisée par l'interface série, les protocoles de transfert et les méthodes de contrôle de flux.

La description du matériel nécessaire à la communication série présentée au second chapitre. Il s'agira de voir le rôle de chaque registre de l'interface série, et chaque broche du connecteur RS232.

Comme le micro contrôleur 68HC11 est le circuit moteur de notre application, et en outre c'est un circuit de comportement spécifique, cela nécessite une étude détaillée de ce dernier c'est pour cela qu'on a réservé un chapitre complet pour faire une représentation de ce circuit (68 HC11) c'est le troisième chapitre.

Le quatrième chapitre est consacré à décrire le chemin qu'on doit parcourir ainsi que les différents outils nécessaires au développement d'une application à base d'un micro contrôleur.

Le cinquième chapitre concerne la conception et la réalisation de l'application, qui est décomposé en deux parties :

- Côté carte d'application : concernant la conception et la réalisation matérielle de la carte et en outre, puisqu'on a utilisé un micro contrôleur, la réalisation logiciel du programme qui gère les taches effectuées par le MC68HC11.

- Côté liaison avec PC : concernant le choix du port d'entrées(port série) ; ainsi que l'écriture de programme qui assure la bonne réception par PC de données présentes sur son port série.

CHAPITRE I :

NOTION DE LA TRANSMISSION SERIE

I.1- INTRODUCTION :

On sait bien que les ordinateurs puissent communiquer entre eux et échanger des données à travers le monde entier. Ils utilisent généralement à cet effet le réseau téléphonique ordinaire, qui ne permet pas une transmission très rapide des données, mais qui présente l'avantage considérable de permettre d'atteindre pratiquement n'importe quel coin du globe. Les données sont transmises de façon série suivant un protocole de transmission bien précis. Comme le PC n'est pas prévu pour ce type de transmission des données dans sa configuration de base, cette transmission n'est possible qu'en implantant une carte RS232C.

I.2- LA NORME RS232C : [10] [2] [3] [8]

La norme RS232C définit 25 lignes entre le DCE (Data Communication Equipement) et le DTE (Data Terminal Equipement), et par suite un connecteur de 25 broches. La plupart d'entre elles sont réservées pour le transfert de données synchrones, les autres pour l'échange de données de manière asynchrone. Pour ce dernier 11 signaux du RS232C uniquement sont utilisés. IBM définit une connexion se faisant sur 9 lignes pour sont interface série asynchrone.
Pour le mode de transmission asynchrone, le tableau 1 montre, pour les deux types de connexion 25 et 9 broches, les affectations des broches ainsi que les significations des dénominations qui leurs sont associés.

25 PINS	9 PINS	Signal	Direction	Description
1	-	-	-	Masse de protection.
2	3	TD	DTE →DCE	Transmission de données.
3	2	RD	DTE ←DCE	Réception de données.
4	7	RTS	DTE →DCE	Demande d'émission.
5	8	CTS	DTE ←DCE	Prêt à émettre.
6	6	DSR	DTE ←DCE	Equipement de données prêt.
7	5	-	-	Masse logique.
8	1	DCD	DTE ←DCE	Détection porteuse.
20	4	DTR	DTE →DCE	Terminal de données prêt.
22	9	RI	DTE ←DCE	Détection sonnerie.
23	-	DSRD	DTE ←DCE	Sélection de cadence de données.

Tab.I.1- Affectations et significations des connections 9/25 broches.

Sur le connecteur 9 broches d'IBM, la masse de protection et le signal DSRD sont omises, mais les 9 signaux qui restent sont complètement suffisant pour l'échange de données serials asynchrones entre le DTE et le DCE, tout à fait en accord avec le standard RS232C. Le contrôle de transfert de données entre DCE et DTE s'effectue par les 5 lignes : RTS, CTS, DCD, DSR et le DTR dont on donne en ce qui suit le rôle de chacune d'entre elles ainsi que la manière de les utiliser:

- RTS (Request To Send): En permanence du DTE, ce signal avise le DCE de se préparer pour un transfert de données. En réponse le DCE active sa porteuse pour le transfert de données vers la destination.
- CTS (Clear To Send) : Issu du DCE et en réponse au DTE, ce signal indique au DTE qu'il est prêt pour le transfert de données en question.

On qualifie la paire (RTS , CTS) de signaux de poignée de main (HAND SHAKE SIGNALS).

• DCD (Data Carier Detect) : Le DCE active la ligne DCD quand il détecte le signal porteur que la destination active pour pouvoir transmettre.

• DSR (Data Set Ready) : Le DCE informe le DTE en activant le DSR pour dire que la connexion avec la destination est établie et qu'il lui est possible de commencer à communiquer.

• DTR (Data Terminal Ready) : Ce signal, quand il est activé, demande l'établissement de la connexion.

On appel la paire (DSR, DTR) signaux d'établissement de connexion. L'utilisation des cinq lignes de contrôle dépend du type de connexion. Dans ce qui suit, on va discuter l'utilisation de ces signaux dans les trois types de connexions susceptibles d'être rencontrer en pratique, à savoir : Simplex, Half-Duplex, Full-Duplex.

I.2.1- MODE SIMPLEX :

En principe ce mode peut avoir deux possibilités : un transfert du DTE vers le DCE ou vis versa. Le simplex est toujours unidirectionnel et la direction ne change jamais. La Fig. 1 illustre une liaison simplex.

Fig.I.1- Liaison simplex.

Dans cette liaison, le DTE et le DCE peuvent être émetteur et récepteur à la fois, mais une seul ligne de donnée est disponible (d'émission ou de réception) et qui est utilisé alternativement par le DTE et le DCE (figI.2).

I.2.2-. MODE HALF-DUPLEX :

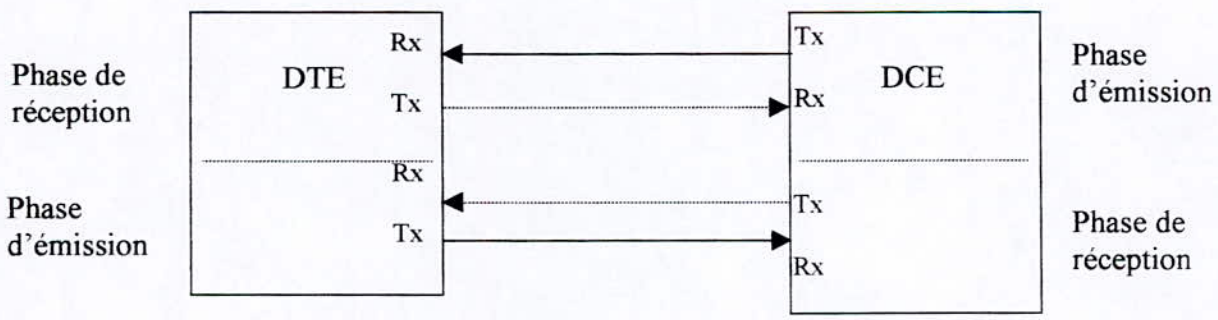


Fig.I.2- Mode half-duplex.

Note : les flèches en pointillé indiquent que la ligne est désactive.
Les flèches en continue indiquent que la ligne est activée.

I.2.3- MODE FULL-DUPLEX:

Dans ce type de liaison, le transfert de données se fait dans les deux sens, chacun affecté à une ligne. Autrement dit deux lignes séparées sont utilisées pour l'échange de données (figI.3).

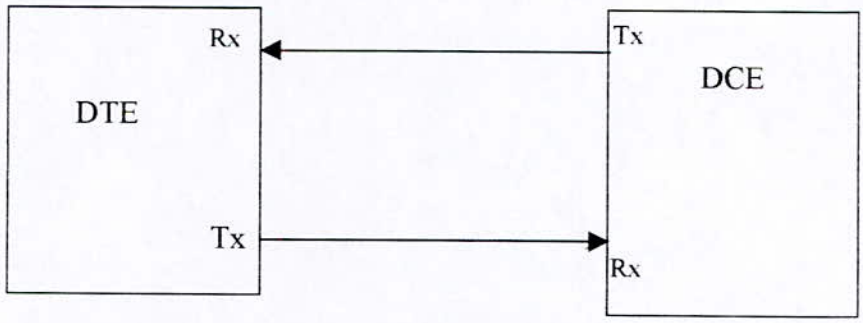


Fig.I.3- Mode Full-duplex.

I.3- TRANSMISSION ASYNCHRONE : [10] [2] [8] [3] [28]

En format asynchrone, chaque caractère transmis est précédé d'un bit de START. Ceci prévient le récepteur de ce qu'un caractère va arriver. Le caractère est suivit d'un ou de plusieurs bits de STOP qui laissent au récepteur une période repos avant la transmission du caractère suivant. Il n'y a pas d'horloge ou d'autre signal de synchronisation, mis avec les données. Le récepteur et l'émetteur ont des horloges internes et le bit START est utilisé pour les synchroniser.

La (Fig.I.4) montre la forme d'onde de la transmission du caractère ASCII du « H » en format asynchrone. Le code ASCII de « H » est « 0111000 « en binaire et il est transmis avec le bit le moins significatif en tête. La ligne de données est au repos à l'état de 1 logique.

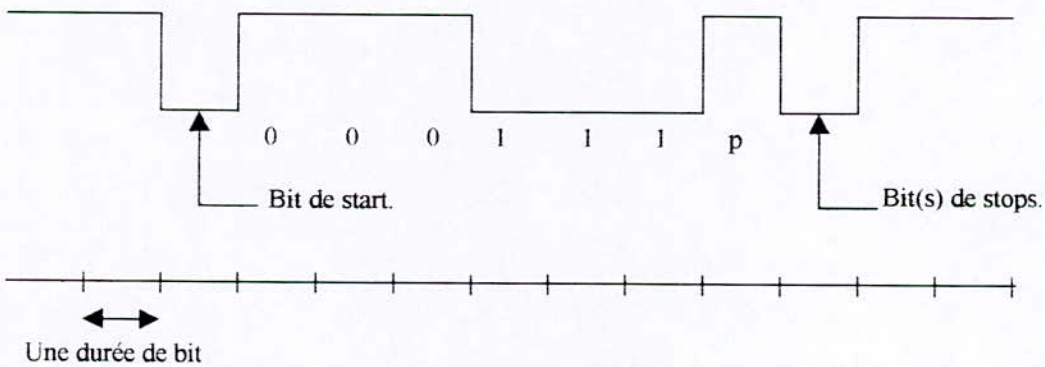


Fig.I.4. Forme d'onde série pour la lettre « H ».

Comme vous le montre la (Fig.I.4) , ce protocole ne prévoit que deux états significatifs, 0 et 1. Si aucun caractère n'est transmit, le canal est à 0 (qui correspond ... l'état électrique 5 volts). Si le signal passe à 1, le récepteur sait alors que des données vont maintenant être transmises. Suivant la convention appliquée, 5 à 8 bits sont envoyés à travers le canal. Le caractère peut être suivi d'un bit de parité qui sert à détecter des erreurs éventuelles lors de la transmission des données. On distingue à cet égard entre parité paire et parité impaire. La parité paire signifie que le mot de données transmis est complété par le bit de parité de telle façon que le nombre de bits soit toujours paire. Si le mot transmis comporte donc, par exemple, trois bits valent 1, le bit de parité vaudra

également 1 pour que le nombre de bits passe à 4 et soit bien un nombre pair. Si le mot transmis contenant par contre un nombre pair de bits, le bit de parité valait 0. Inversement si c'est le bit de parité impaire qui est convenue, le bit de parité sera fixé de telle façon que le nombre de bits 1 soit toujours impair.

Viennent en dernier lieu les bits de STOP, qui signalent la fin de la transmission de données correspondant à un caractère. Le protocole de transmission des données peut prévoir 1, 1.5 ou 2 durées de bits de STOP.

La durée de bit détermine la cadence maximale à laquelle les caractères peuvent être transmis et définie ainsi la fréquence de bit à laquelle fonctionne une interface déterminée. Les fréquences de bit normalisées pour les liaisons asynchrones sont : 50, 75, 110, 134.5, 150, 300, 600, 1200, 2400, 3600, 4800 et 9600 bits par seconde (bauds).

I.4- TRANSMISSION SYNCHRONE: [10] [2] [8] [3] [28]

La transmission synchrone est orientée bloc. Les messages transmis constitués de caractères multiples qui sont synchronisés en ajoutant des caractères au début du message. Un ou plusieurs caractères de synchronisation (SYNC) sont ajoutés à l'avant du message. Ces caractères SYNC servent à synchroniser le récepteur avec l'émetteur. Au contraire à ce qui se passe en transmission asynchrone, les émetteurs et récepteurs synchrones ont une horloge commune. Ou l'émetteur ou le récepteur doit envoyer un bit d'horloge à l'autre dispositif. Ainsi il n'est pas besoin d'avoir un bit de START car l'émetteur et le récepteur sont toujours synchronisés au niveau du bit. Les caractères SYNC sont utilisés pour permettre au récepteur et à l'émetteur de se synchroniser au niveau des frontières de caractère ou de message.

I.5- LE TRANSFERT DE FICHIERS :

I.5.1-INTRODUCTION : [4]

Lorsqu'un système a correctement été installé, une des activités les plus fréquemment entreprises par les utilisateurs des logiciels de communication est le transfert de fichiers. L'un des principaux problèmes avec les transferts de fichier est qu'il n'y a pas de norme précise sur la façon de faire les choses.

Par conséquent, lorsque l'on veut déplacer des fichiers d'une machine à une autre, on doit examiner le choix des protocoles qui sont disponibles sue les deux machines (entre deux ordinateurs ou un ordinateur et un autre périphérique). On cherche ensuite des protocoles communs aux deux machines et on tente de sélectionner le meilleur.

I.5.2- NECESSITE DU PROTOCOLE : [3]

Connecter un ordinateur pour transmettre des données en série est plutôt élémentaire ; presque tous les ordinateurs proposent en option la gestion d'une imprimante série et peuvent être programmés pour envoyer des données à partir d'un port série sans trop de difficulté.

Les concepteurs des méthodes de communication informatique actuelles, partent de l'hypothèse que l'information est transmise depuis un clavier, sous forme de texte et entrée à une vitesse de frappe humaine. Mais les transferts de fichier, d'autre part, ont a lieu à des vitesses bien supérieures à celle de la frappe humaine et contiennent parfois des caractères qui n'apparaissent pas au clavier.

Les problèmes qui en résultent sont abordés ci-dessous. La plupart d'entre eux étant résolue grâce aux protocoles qu'on va étudier.

I.5.2.1- LONGUEUR DE MOT :

Une grande partie des données contenues dans les fichiers du micro-ordinateur n'est pas constituée uniquement de texte, mais de programmes, de données graphiques ou autres informations non-ASCII ; ces données, souvent appelées données binaires, utilisent généralement les huit bits de chaque octet. La table ASCII standard, qui contient les caractères des claviers les plus courants, ne sert que de mots de sept (7) bits. De nombreux systèmes d'ordinateurs et la plus part des canaux de communication, élaborés uniquement pour des entrées ASCII, sont incapables d'accepter des mots de huit (8) bits. Ils ne reconnaissent que des mots de sept (7) bits et considèrent le huitième comme un bit de parité.

I.5.2.2-CARACTERE DE CONTROLE :

Un autre problème rencontré lors des transferts des fichiers concerne les caractères de contrôle. Les données transférées contiennent souvent des valeurs d'octets particulières, que l'ordinateur destinataire pourrait interpréter comme des signaux de contrôle ASCII. Quelques ordinateurs, d'autre part, manipulant des caractères spéciaux que les logiciels d'application ne peuvent pas traiter. De façon à surmonte ce problème, les données doivent être converties en caractères acceptables par l'ordinateur récepteur. Le protocole **KERMIT** est un exemple de protocole réalisant cette opération.

I.5.3-PROTOCOLE DE TRANSFERT DE FICHIERS : [5]

Il existe de nombreuses méthodes éprouvées pour faire face aux problèmes énoncés précédemment. Certaines d'entre elles sont abordées dans ce qui suit. Les protocoles **XMODEM, YMODEM, ZMODEM et KERMIT** , tout particulièrement sont l'objectif de notre étude.

I.5.3.1-LE PROTOCOLE XMODEM : [5] [3]

Le protocole de transfert le plus utilisé pour toutes sortes de fichiers, binaire ou ASCII est le protocole XMODEM. Il sert le plus souvent pour charger des fichiers à partir de petits systèmes informatiques, comme le PC.

I.5.3.1.1- FORMAT DES BLOCS :

Les données transférées par XMODEM sont en bloc sur un octet, chacun d'eux est constitué d'un caractère début texte($01), d'un numéro de bloc sur un octet, du complément à 1 de celui-ci, de 128 octets de données et d'une somme de contrôle sur un octet.

Le format est décrit dans le tableau suivant :

Numéro	Contenu
0	Caractère début d'entête
1	Numéro de bloc
2	Complément à 1 du numéro de bloc
3-130	128 octets de données
131	Somme de contrôle :somme des seuls octets de données retenues ignorés

Tab.I.2- Format des blocs du XMODEM.

Remarque : la somme de contrôle est calculée en additionnant les 128 octets de données.

I.5.3.1.2- FICHIER DE PROTOCOLE :

Avant que l'ordinateur émetteur puisse commencer à envoyer des données, il doit recevoir un caractère d'accusé de réception négatif (NAK)de la part du récepteur. Le programme récepteur est en effet supposé émettre le caractère après dix secondes sans réception de données, en marquant par cette temporisation l'ouverture du dialogue.

Dés que le programme récepteur reçoit un bloc, il rend compte d'une erreur chaque fois qu'un intervalle d'une seconde. Cependant, il doit attendre que la ligne de contrôle se libère avant d'envoyer le caractère NAK signalant l'erreur .

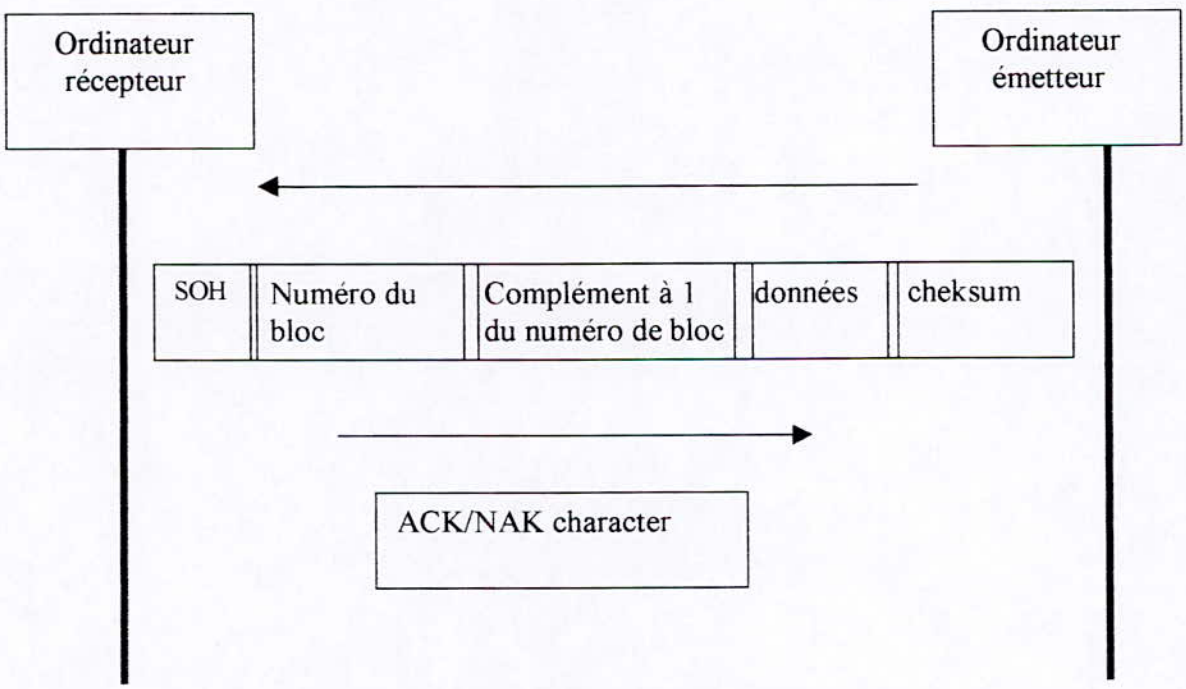


Fig.I.5- Structure d'un bloc dans XMODEM

Le récepteur vérifie le numéro du bloc et rend compte d'une erreur si celui-ci est hors de la série. Si le numéro du bloc est le même que le précédant, il signal une retransmission.

Après une réception de chaque bloc, le récepteur envoie le caractère ACK si le bloc est correctement reçu ou bien le caractère NAK. Dans ce dernier cas, l'émiteur réexpédie le bloc.
A la fin de transmission, l'émetteur' envoie le caractère fin de texte(EOT)puis attend un caractère ACK.

I.5.3.2- LE PROTOCOLE YMODEM : [5]

YMODEM est une extension du XMODEM apportant plusieurs améliorations. Les caractéristiques qu'il ajoute à XMODEM sont énumérées ci-dessous(Fig.I. 6) :

- Un test d'erreur CRC-16
- Des blocs de 1 KO en option.
- L'envoi du caractère début de texte (STX) au début de chaque bloc, au lieu du caractère début d'entête (SOH), signifie que le bloc qui suit contient 1024 octets au lieu de 128 octets. On peut mélanger des blocs de 1024 et de 128 octets dans une même transmission.
- Avortement avec CAN-CAN, deux caractères ANNULATION consécutifs indiquant qu'il faut avorter le transfert de fichier.
- Transmission de lots de fichier. Plusieurs fichiers peuvent être envoyés en même temps. Pour chaques fichiers on envoie un bloc numéro zéro, celui-ci contient le nom de fichier en minuscules terminé par le zéro ASCII.

Le bloc zéro peut également posséder l'un des champs supplémentaires suivants :

1. La taille du fichier sous forme d'une chaîne décimale suivie d'un espace. Si elle est envoyée, le programme récepteur saura quels caractères ignorer dans le dernier bloc.
2. La date courante, sous forme octale, exprimée en secondes à partir du premier janvier 1970,avec la valeur zéro si la date est inconnue. Elle doit être suivie d'un espace.

Le reste du bloc est mis à 0. D'autres fichiers sont envoyés, chacun possédant son propre bloc nominal. Un nom de fichier nul termine la transmission du lot.

STX	Numéro du bloc	Complément à 1 du numéro de	données	CRC high	CRC low

Fig.I.6 Structure du protocole YMODEM

I.5.3.3- LE PROTOCOLE ZMODEM : [4]

En 1986, le fournisseur de réseaux à communication de paquets de **Telnet** demanda à Chuck Forsberg d'Omen Technology de développer un nouveau programme de transfert de fichiers, qui pourrait être utilisé sur leur réseau. Le résultat fut ZMODEM, qui est à la fois un programme du domaine public et un nouveau protocole.

Le nom ZMODEM peut sous-entendre qu'il s'agit d'un descendant direct de XMODEM et de YMODEM, mais ce n'est pas le cas. ZMODEM est un nouveau protocole qui a très peu de chose en commun avec ces plus vieux systèmes.

ZMODEM a plusieurs caractéristiques importantes qui le rendent supérieur aux autres protocoles.

- Toutes les transactions sont protégées par des codes correcteurs d'erreurs(CRC) 16 ou 32 bits, qui réduisent beaucoup la possibilité de faux acquittements positifs ou négatifs.

- Le protocole fonctionne correctement sur des liaisons de communication absorbant les caractères de contrôle, bien qu'un canal 8 bits soit encore requis.
- Les données sont envoyées en un flux continu de paquets, sans attendre d'acquittement. Le protocole ne s'arrête d'envoyer des paquets que lorsque le récepteur l'interrompt pour signaler une erreur. Envoyer des données en un flot continu améliore énormément le débit de transmission sur des réseaux à communication de paquets ou avec des modems munis de zones tampons.

Grâce à ces caractéristiques supérieures, ZMODEM est devenu l'un des protocoles les plus utilisés aujourd'hui. ZMODEM est toutefois plus ambitieux à implémenter car c'est un protocole plus complexe, il est aussi disponible presque universellement. Beaucoup de machines UNIX et de machine à temps partagé n'offrant pas XMODEM sont capable de supporter ZMODEM.

I.5.3.4-LE PROTOCOLE KERMIT : [3] [5]

Ce protocole a été développé à l'université Colombia de New York dans le but de faciliter des transferts de fichiers entre micro ordinateurs (stockage, échange de fichiers). Des versions de ce logiciel tournent sur la plupart des gros systèmes.

I.5.3.4.1- MODE D'EMPLOI DE KERMIT :

La méthode habituelle d'utilisation de KERMIT pour les transferts en direction d'une unité centrale, est de partir d'un programme de communication tournant sur un micro-ordinateur équipé d'un émulateur de terminal et de protocole KERMIT, en faisant fonctionner le micro comme émulateur de terminal. L'utilisateur oblige l'unité centrale à faire tourner sa propre version de KERMIT. Le micro-ordinateur est ensuite commuté dans le mode KERMIT et le transfert de fichier peut commencer.

I.5.3.4.2- CODAGE DES CARACTERES:

A l'inverse de XMODEM, KERMIT transforme les caractères transmis en caractères imprimables standard (en ASCII, 32 ou 126). De ce fait, les caractères non imprimables peuvent être transmis sans obliger l'ordinateur à les manipuler d'une façon spéciale.

I.5.3.4.3- CARACTERES DE CONTROLE & D'INFORMATION :

Les caractères de contrôle sont les caractères de 0 à 31 et 127 ; lorsqu'ils sont détectés à l'intérieur des données à transmettre, KERMIT les convertit en caractères imprimables en les décalant de 6 positions à gauche et en les faisant précéder d'un caractère préfixe. Les caractères d'information de KERMIT , sont des caractères qui ne font pas partie des données transmises mais qui contiennent une information numérique à KERMIT, comme par exemple la longueur du paquet, sont codé en ajoutant 32.

I.5.3.4.4- PAQUETS:

L'unité de message principale utilisée par KERMIT est le paquet. Un paquet peut être constitué de données ou d'informations, tels un accusé de réception ou un message d'erreur.

Les types des paquets sont énumérés ci-dessous. Chaque type est désigné par une lettre de l'alphabet.

- D : paquet de données.
- Y : accusé de réception positif(ACK).
- N : accusé de réception négatif (NAK).

- S : début d'envoi (paramètres d'échange).
- B : fin de transmission.
- F : en tête de fichier.
- Z : fin de fichier.
- E : erreur .
- T : réservé à l'usage interne.
- I : initialisation (paramètres d'échange).
- A : attribut de fichier.
- R : début de réception.
- C : commande de serveur.
- K : commande de KERMIT.
- G : commande génétique de KERMIT.

I.5.3.4.5-FORMAT GENERAL DU PAQUET :

Le format général d'un paquet est illustré par le tableau suivant :

Numéro	Contenu	Signification
0	MARK	Début du paquet
1	LEN	Nombre de caractère après ce champ
2	SEQ	Numéro de séquence (modulo64)
3	TYPE	Type de paquet
4	DATA	Données
(fin)	CHECK	Somme de contrôle

Tab.I.3- Format d'un paquet KERMIT.

I.5.3.4.6-INITIATION DE LA SESSION :

Une session démarre lorsque l'appareil émetteur envoie un paquet d'initialisation contenant divers paramètres. Celui-ci fait l'objet d'un accusé de réception par l'appareil récepteur, qui indique en retour dans le paquet ACK quels paramètres et quelles options il peut satisfaire.

I.5.3.4.7- LE TRANSFERT DES DONNEES:

Une fois échangée les paquets d'initialisation, les données sont transmises en une série de paquets de données. L'appareil récepteur doit accuser la réception de chaque paquet. L'appareil émetteur doit attendre l'accusé de réception de chaque paquet avant d'en envoyer un autre.

Pour chaque fichier, KERMIT envoie un en-tête de fichier, un ou plusieurs paquets de données et un paquet de fin de fichier(Fig.I.7).

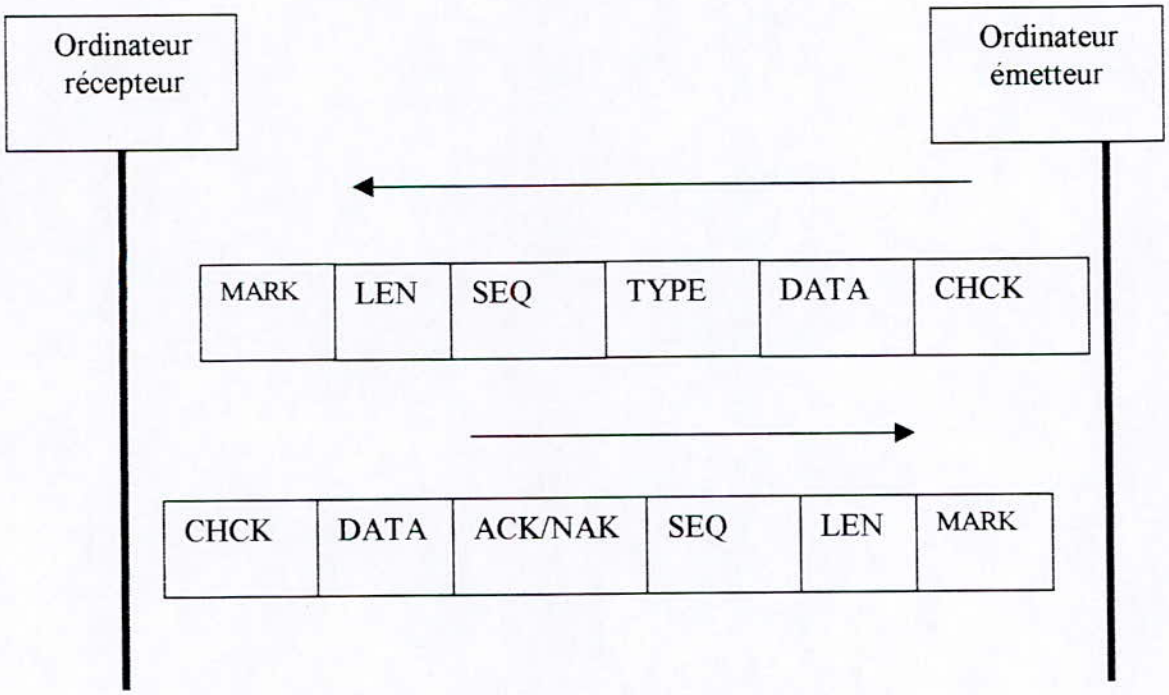


Fig.I.6- Format bloc dans KERMIT

I.5.3.5-AUTRES PROTOCOLES :

D'autres protocoles mis en lumière sont par exemple **CompusServe, Quick-B, Bimodem** et des protocoles propriétaires qui sont utilisés dans des programmes, tels que Blast et Crosstalk. A présent, il semblerait qu'il n'y ait pas de protocole offrant assez de caractéristiques nouvelles pour détrôner les leaders, mais un bon protocole peut rapidement prendre l'avantage dans nos environnements modernes.

I.6- LE CONTROLE DE FLUX SUR UNE INTERFACE SERIE : [6]

Quand on trouve entre deux ordinateurs avec un périphérique sur une interface série, il peut arriver au bout d'un certain temps que le récepteur ne puisse plus suivre le débit. Il faut alors signaler à l'émetteur qu'il doit interrompre la transmission. Cette signalisation peut s'effectuer :

- Soit par des fils de l'interface série : c'est la ligne DTR qui est plus souvent utilisée. L'ordinateur récepteur met la ligne DTR à l'état bas quand il n'est pas prêt à recevoir un nouveau caractère.
- Soit par des codes de contrôles inclus dans le circuit des données. Les principaux protocoles utilisés sont :

I.6.1- LE PROTOCOLE XTX-ACK :

Les données sont transmises par blocs et terminés par un code ETX. L'émetteur arrête alors la transmission. Le récepteur traite tous les caractères reçus et, s'il est prêt à recevoir un bloc, renvoie un code ETX. A la réception du code ACK, l'émetteur reprend la

transmission. Le code ACK n'étant transmis qu'après arrêt de la transmission dans l'autre sens, ce protocole peut fonctionner sur une liaison qui travaille à l'alternat.

I.6.2- LE PROTOCOLE X-ON/ X-OFF :

Plus performant que le protocole **ETX,ACK**, il implique la possibilité de transmission bidirectionnelle. Quand le buffer du récepteur est presque plein, il envoie un code **X-OFF** à l'émetteur -, qui s'arrête d'émettre. Quand le buffer redevient suffisamment vide, il envoie un code **X-ON**, qui autorise l'émetteur à transmettre de nouveau.

CHAPITRE II :
ORGANES
DE TRANSMISSION

II.1-INTRODUCTION : [3]

L'invention de l'ordinateur, a permis à l'homme d'atteindre le sommet dans diverses applications. La communication homme-machine étant connue, reste une chose pour élargir le domaine de communication c'est : la communication **machine-machine**.

Comme le PC n'est pas prévu pour ce type de transmission des données dans sa configuration de base, cette transmission n'est possible qu'en implantant une carte RS232. Cette carte est désignée, dans la terminologie officielle d'IBM, sous le nom **d'adaptateur pour la transmission asynchrone de données.**

Avec cette communication, il est possible de réaliser une transmission de données directe entre deux ordinateurs reliés par un câble ou bien une transmission indirecte à travers le câble téléphonique.

Le cerveau d'une carte **RS232** est constitue par un processeur appelé : **UART**(universal asynchronous reciever transmitter). Outre la gestion des fonctions de communication, **l'UART** permet de réaliser les tâches spécifiques à ce type de communication :

- Conversion parallèle-série de donnée à transmettre.
- Conversion série-parallèle de donnée reçue.

II.2-LE PC ET LA PROGRAMMATION SYSTEME : [7]

II.2.1-LE MODELE A TROIS COUCHES :

L'une des tâches fondamentales de la programmation système consiste à accéder au matériel constituant le PC. Cet accès ne se fait nécessairement de manière directe, la manière usuelle de procéder consiste à utiliser peut être le BIOS ou le DOS qui sont justement des interfaces logiciels créés pour gérer le matériel.

II.2.1.1-LE BIOS :

Le BIOS peut être considéré comme une couche située juste au-dessus du matériel, il permet d'accéder aux composants suivants :

- Carte vidéo
- Mémoire vive
- Disquette et d0isque dure
- Interfaces parallèles
- Clavier et horloge
- Interfaces séries

II.2.1.2-LE DOS :

Le DOS propose également des fonctions d'accès au matériel. Elles présentent cependant un autre caractère que les fonctions du BIOS, car elles considèrent le matériel non pas sous un angle physique mais comme un dispositif logique.

Exemple : pour l'accès aux disquettes et disque dur :

BIOS ⟶ En piste et en secteurs

DOS ⟶ Des fichiers et des sous répertoires

II.2.1.3- COTE MATERIEL :

Une compréhension minimale du matériel est obligatoire pour assimiler par la suite la programmation.

comme nous l'avant introduit, le PC est doté :

- D'une carte RS232qui l'adapte pour un type de transmission série.
- Le cerveau de cette carte constitue ce qu'on appelle l'UART.

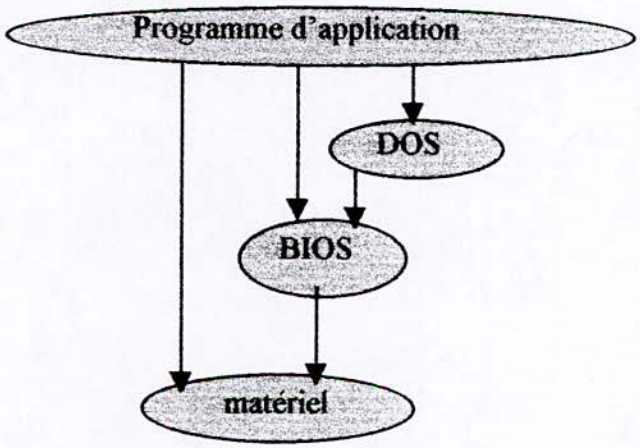


Fig.II.1 –Modèle de trois couches

II.3- ETUDE DE L'UART 8250 : [8] [4] [6] [9] [14]

Le processus d'écriture de données dans un flux de bits est un travail intensif pour un ordinateur. Chaque bit doit être décalé, placé dans un flux de sortie et ensuite ajusté à la longueur de bit correcte. Lire des bits sur la ligne série est encore plus difficile le flux d'entrée doit être échantillonné à une vitesse sensiblement plus élevée que la vitesse de la ligne série ; le processus doit rechercher des bits de démarrage écarter les bits de démarrage invalides causés par du bruit puis chronométrer chaque bit en échantillonnant environ la moitié supposée des bits.

De part la nature temps réel de cette tâche un processus sur lequel fonctionne un système d'exploitation avec une gestion ne peut pas le faire efficacement.

Ainsi à peu tous les systèmes Bureautique réalisent des transferts de données séries par l'intermédiaire d'une puce UART pour les utilisateurs.

L'UART est un processeur fait uniquement pour gérer les communications entre le PC et l'extérieur, elle exécute deux fonctions très importantes :

- Convertit les octets en données séries (bits) sur la ligne RS232 pendant la phase de transmission.
- Convertit les flux des données(bits) en octets pendant de réception.

II.3.1-STRUCTURE INTERNE DE L'UART :

L'UART (universal asynchron receiver transmitter) possède des registres programmables utilisés pour le contrôle et la gestion des ports séries.

Un grand nombre de ces registres sert pour l'initialisation et un petit nombre pour l'utilisation régulière, on peut accéder à tous les registres à partir de sept adresses des ports d'E/S, ces adresses sont calculées comme des adresses d'offset par rapport à une adresse de base, elle varie suivant le port de communication utilisé.

Les adresses des ports sont illustrées par le tableau suivant :

Port de communication	Adresse de base
COM1	$03F8
COM2	$02F8
COM3	$03E 8
COM4	$02 E 8

Tab.II.1- Adresse des ports de communication.

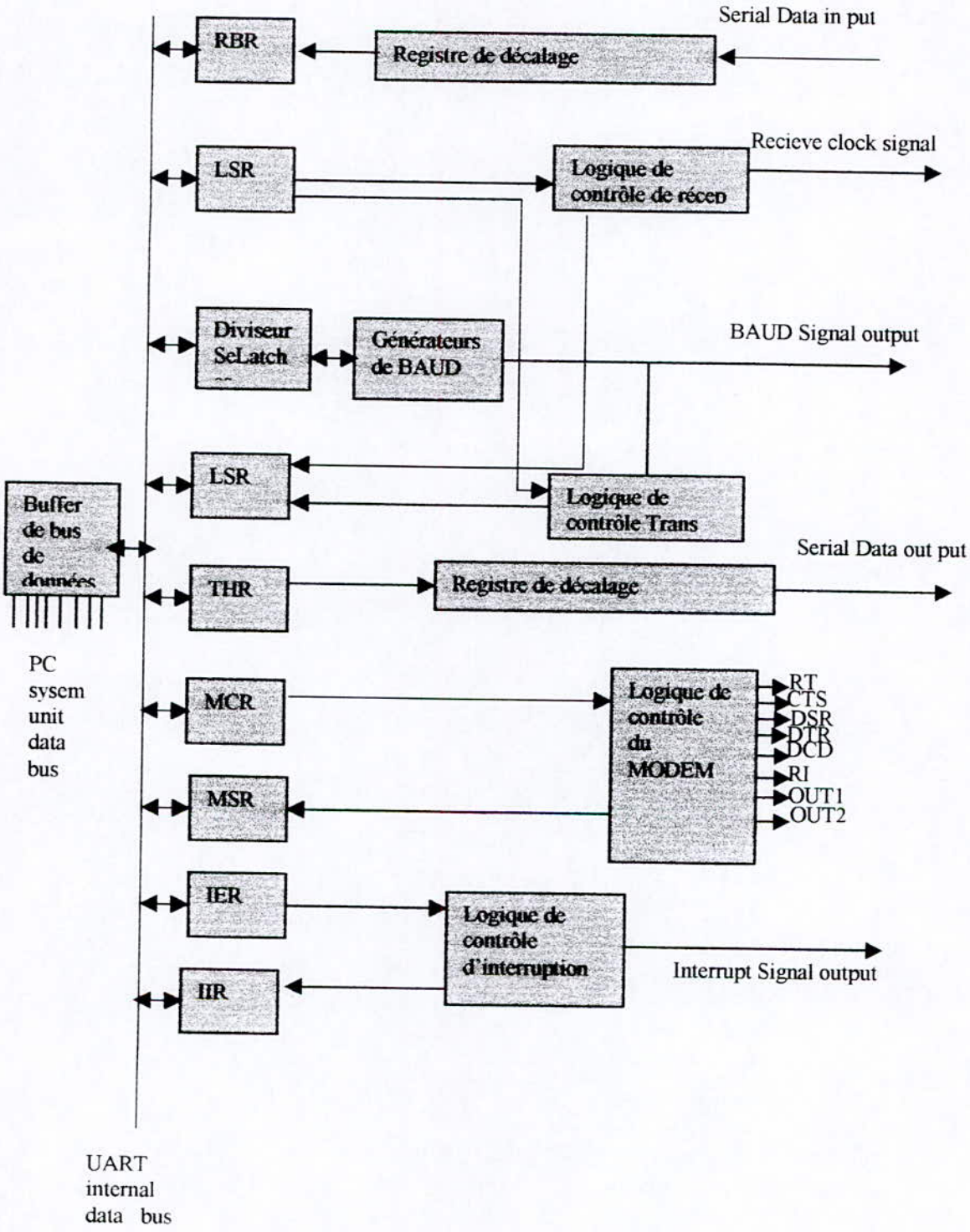


Fig.II.2- Architecture interne de l'UART

II. 3.2- DESCRIPTION DES REGISTRES DE L'UART :

L'UART possède trois sortes de registres :

- Les registres de contrôle qui reçoivent les commande de l'UC.
- Les registres d'état qui informent l'UC de ce qui arrive vers l'UART.
- Les registres tampons qui recueillent les caractères en instance d'émission ou de traitement.

La façon dont les registres sont manipulés dépend de l'architecture de l'ordinateur auquel l'UART est associée, dans le cas de l'IBM PC les vecteurs à placer dans les registres sont envoyés à une adresse d'E/S appropriée, au moyen d'une instruction 'out' adressé à l'UC. On accède aux registres à lire à l'aide d'une instruction 'in' accompagnée de l'adresse adéquate.

Adresse	DLAB	Registre sélectionné
$03F8	0	Registre de réception des données 'RDR'
$03F8	0	Registre d'attente d'émission 'THR'
$03F9	0	Registre de validation des interruptions 'IER'
$03FA	0	Registre d'identification des interruptions 'IIR'
$03FB	0	Registre de contrôle de la ligne 'LCR'
$03FC	0	Registre de contrôle de modem 'MCR'
$03FD	0	Registre d'état de la ligne 'LSR'
$03FE	0	Registre d'état du modem 'MSR'
$03FF	0	Pas de registre
$03F8	1	Registre diviseur de fréquences 'octet bas'
$03F9	1	Registre diviseur de fréquences 'octet haut'

Tab.II.2- Les adresses des différents registres de l'UART

Le **DLAB :** est le bit pointeur de registre indirect. C'est le plus significatif du registre de commande de ligne. Le processeur doit afficher ce bit de façon appropriée suivant la paire de registres à laquelle il désire pouvoir accéder.

Typiquement DLAB est mis à 1 pendant l'initialisation du boîtier pour l'introduction du diviseur de fréquence de bits, puis remis à 0 et laissé en permanence à cette valeur jusqu'à ce qu'une modification de la fréquence de bits soit nécessaire.

II.3.2.1- REGISTRE DE CONTROLE DE LA LIGNE (LCR : LINE CONTROL REGISTER) :

Le format de bits pour la liaison asynchrone est configuré à l'aide du registre de commande de ligne. Les bits de ce registre définissent le nombre de bits par caractère, le nombre de bit de stop et la parité. On trouve également dans ce registre un bit qui peut forcer le 8250 à envoyer le niveau de rupture (break) et un bit DLAB.

B7	B6	B5	B4	B3	B2	B1	B0
DLAB	BRK	STP	EPS	PEN	STB	WS1	WS0

Tab.II.3- Structure des bits du registre LCR.

B0 et B1 (WLS : world length select bits) : Spécifient le nombre de bits utiles dans chaque caractère émis ou reçus.

WLS1	WLS0	Longueur utile du caractère
0	0	5 bits
0	1	6 bits
1	0	7 bits
1	1	8 bits

Tab.II.4- Bits de sélection de longueur de mot du 8250.

STB : détermine le nombre de bits STOP
STB=1 : 1,5 bits STOP pour une longueur du caractère de 5 bits
2 bits STOP pour une longueur 6,7 ou 8 bits.
STB=0 : 1 bit STOP.
PEN : (parity enable) : validation de parité.
PEN=0 ne pas utiliser le bit de parité.
PEN=1 utiliser le bit de parité.
EPS : (even parity select) sélection de la parité.
EPS=0 parité impaire
EPS=1 parité paire
SP : (stick parity) blocage de la parité.
SP=0 sans effet.
SP=1 si les bits B3 et B5 valent 1, le bit de parité est transmis avec un état opposé à celui indiqué dans B4.
SB : (set break) générateur de break
SB=0 sans effet
SB=1 la sortie de l'UART est forcée au zéro logique.
DLAB :
DLAB=0 accès au registre émission/ réception et validation d'interruption.
DLAB=1 accès au diviseur.

II.3.2.2- GENERATEUR DE VITESSE DE TRANSMISSION :

La fréquence de bits (taux de baud) est déterminée par le registre diviseur de fréquences de bits : divisor latch LSB (DLL), divisor latch MSB (DLM).

Les deux octets des deux registres sont combinés en un nombre de 16 bits utilisé pour diviser la fréquence d'horloge fournie à l'UART. Le diviseur doit être positionné à 16 fois la cadence désirée.

$$\text{Taux de BAUD} = \frac{\text{Fréquence horloge}}{16 * \text{diviseur de Fréquence}} = \text{vitesse de transmission}$$

Le tableau suivant fournit les valeurs des registres diviseurs de fréquences pour les vitesses de transmission les plus utilisées.

Fréquence d'horloge=1,8432Mhz		Fréquence d'horloge=3,072 MHz	
Vitesse de transmission	**Registres diviseur**	**Vitesse de transmission**	**Registres diviseur**
50 bits/S	2304	50 bits/S	3840
110 bits/S	1047	110 bits/S	1745
1200 bits/S	96	1200 bits/S	160
2400 bits/S	48	2400 bits/S	80
9600 bits/S	12	9600 bits/S	20
56000 bits/S	2	56000 bits/S	3

Tab.II.5- Valeurs des registres diviseurs de fréquences.

II.3.2.3- REGISTRE D'ETAT DE LA LIGNE LSR (LINE STATUS EGISTER) :

Il contient l'état de la transmission et les types d'erreurs éventuelles.

B7	B6	B5	B4	B3	B2	B1	B0
RBE	TEM	THRE	BR	FE	PE	OE	DR

Tab.II.6- Structure des bits du registre LSR.

DR (Data ready) : donnée prête.

DR=1 : caractère disponible dans le registre de réception.

DR=0 : si non.

Ce bit est remis à 0 soit :

- Par la lecture de la donnée.
- En mettant ce bit à 0.

OE : (overun error) : erreur de recouvrement.

OE=1 : écrasement du caractère précédent.

OE=0 : si non.

PE : (parité error) : erreur de parité.

PE=1 : caractère reçu ne présente pas la bonne parité.

PE=0 : si non.

FE (framing error) : erreur de cardage.

FE=1 : caractère reçu ne présente pas la bonne configuration du bit STOP.

PE=0 : si non.

BI : (break interrupt) : rupture de séquencement.

BI=1 : détection du break.

BI=0 : si non.

THRE: (transmitter holding register empty) : indicateur du registre d'attente d'émission vide

THRE=1 : un caractère est transmis du registre d'attente vers le registre de décalage d'émission.

THRE=0 : un nouveau caractère est chargé dans le registre d'attente d'émission.

TE : (transmitter empty) : indicateur du registre de décalage d'émission vide.
TE=1 : registre de décalage d'émission vide.
TE=0 : si non.
RBE : utilisé pour signaler un time out.

II.3.2.4- REGISTRE DE VALIDATION DES INTERRUPTIONS IER (INTERRUPT ENABLE REGISTER). :

Ce registre est sur 8 bits, cependant quatre ne sont pas utilisés, il permet la validation de quatre types d'interruption du 8250 qui vont générer un signal de sortie. Pour déconnecter toutes les interruptions, mettre 0 dans ce registre.

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	MS	RLS	THRE	RDR

Tab.II.7- Structure des bits du registre IER.

RDR (receive data ready) : validation des interruptions pour données reçues.
THRE : (transmitter holding register empty) : validation des interruptions pour registre d'attente d'émission vide (prêt à émettre).
RLS : (receiver line status) validation des interruptions lors d'erreurs de réception.
MS : (modem status) : validation des interruptions du modem.
Bi=1 interruption validée.
Bi=0 interruption non validée.

II.3.2.5-REGISTRE D'IDENTIFICATION DES INTERRUPTIONS (IIR : INTERRUPT IDENTIFICATION REGISTER). :

Il permet au microprocesseur de déterminer rapidement la cause d'une interruption provenant de l'UART.

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	IID1	IID0	IP

Tab.II.8- Structure des bits du registre IIR.

IP : (interrupt pending) : indique s'il y a demande d'interruption de l'UART en attente.
IP=1 : interruption en attente.
IP=0 : pas d'interruption en attente.
IID1 et IID0 : (interrupt ID bits) : permettent l'identification de la source d'interruption.

IID1	IID0	Niveau de priorité	Source d'interruption	Remise à 0 de l'interruption
1	1	Plus haute	Erreur de réception (OV,PE,FE,BRK)	Lecture du registre d'état de la ligne
1	0	Seconde	Réception d'un caractère	Lecture du registre de réception
0	1	Troisième	Registre d'émission vide	Ecriture dans le registre d'émission
0	0	Plus basse	Etat du modem (CTS, DSR, RI, RLSD)	Lecture du registre d'état du modem

Tab.II.9- Signification des bits 0 et 1 de l'identification.

II.3.2.6- REGISTRE DE CONTROLE DE MODEM MCR (MODEM CONTROL REGISTER). :

Ce registre permet de piloter l'interface et la validation d'interruption. Il a les affectations de bits suivantes.

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	DM	OUT2	OUT1	RTS	DTR

Tab.II.10- Structure des bits du registre MCR.

DTR : (data terminal ready) commande l'état de la ligne $\overline{DTR}$

DTR=1 $\overline{DTR}$ est forcée à l'état actif.

DTR=0 $\overline{DTR}$ est mis à l'état inactif.

RTS : (request to send) : commande l'état de la ligne $\overline{RTS}$.

RTS=1 : $\overline{RTS}$ est forcée à l'état actif.

RTS=0 : $\overline{RTS}$ est forcée à l'état inactif.

OUT1 : (output 1 control) : commande la sortie auxiliaire OUT1 (non utilisée) ;

OUT2 : (output 2 control) : commande la sortie auxiliaire OUT2 utilisée pour contrôler les commandes d'interruption en provenance de IRQ3 et IRQ4.

OUT2=1 $\overline{OUT2}$ est forcée à l'état actif et autorise la génération d'interruption à travers les broches IRQ3 et IRQ4.

OUT2=0 : $\overline{OUT2}$ est mis à l'état inactif.

DM :(diagnostic mode) : contrôle la commande de blocage locale.

Lorsque ce bit est à 1, SOUT est mise à haute impédance, et le récepteur du 8250 est déconnecté de SIN. La sortie du registre de décalage d'émission est reliée d'une façon interne (looped back) au registre de décalage de réception.

II.3.2.7- REGISTRE D'ETAT DU MODEM MSR (MODEM STATUS REGISTER). :

B7	B6	B5	B4	B3	B2	B1	B0
DCD	RI	DSR	CTS	DDCD	DRI	DDSR	DCTS

Tab.II.11- Structure des bits du registre MSR.

L'état courant des signaux modem de la ligne est fourni par ce registre. Les bits de poids faibles, au nombre de quatre, indiquent une transition sur les signaux modem auxquels ils correspondent. A chaque fois que l'un des bits 0, 1, 2, 3 est à 1, une interruption modem est générée.

DCTS : (delta clear to send) : indique qu'il a eu changement dans l'état de la ligne $\overline{CTS}$.

DDSR : (delta data set ready) : indique qu'il a eu changement dans l'état de la ligne $\overline{CTS}$.

DRI : (delta ring indicator) : indique qu'il a eu changement dans l'état de la ligne $\overline{RI}$.

DDCD : (delta data carrier detect) : indique qu'il a eu changement dans l'état de la ligne $\overline{RLSD}$.

CTS : (clear to send) : reflète l'état complémentaire effectif de la ligne $\overline{CTS}$.

DSR : (data set ready) : reflète l'état complémentaire effectif de la ligne $\overline{DSR}$.

RI : (ring indicator) : : reflète l'état complémentaire effectif de la ligne $\overline{RI}$.

DCD : (data carrier detect) : reflète l'état complémentaire effectif de la ligne $\overline{RLSD}$.

NB : les bits B0...B3 indiquent un changement d'état du bit correspondant(B4...B7) depuis la dernière lecture du registre d'état du modem par le microprocesseur.

II.4- LA CONNEXION CROISEE DU RS232 : [3] [13]

Les modems et les lignes téléphoniques ne sont plus nécessaires pour connecter deux machines situées cote à cote ou à moins de 15 mètres de distance.

Si la mise en œuvre est bien faite, les broches du RS232, sont analogue quel que soit le DTE. Ceci implique également que ces machines soient prévues pour recevoir des signaux de la RS232 normalement émis par le DCE.

Lorqu'un port est conçu pour attendre le signal normalement fournis par le modem, ou dit qu'il émule un équipement terminal de données (DTE). Lorsqu'il reçoit des signaux, il produit lui aussi des signaux tels que (demande d'émission, terminal de données prêt). Si d'autre part, le port émule un équipement de communication, alors : données prêtes, prêt à émettre et détection de la porteuse sont des signaux en sortie par le port.

Lorsque le DTE ou le DCE sont émulé, les bons signaux de sorties sont produits et l'on peut attendre les signaux d'entrée correspondants. Nous divisons ces signaux en quatre catégories :

1. Terre.
2. Données.
3. Contrôle.

4. Synchronisation.

Avant d'aborder ce qui suit, consultant la (Fig.II.3.) (cas du câble avec modem) pour différencier les signaux du DTE et DCE.

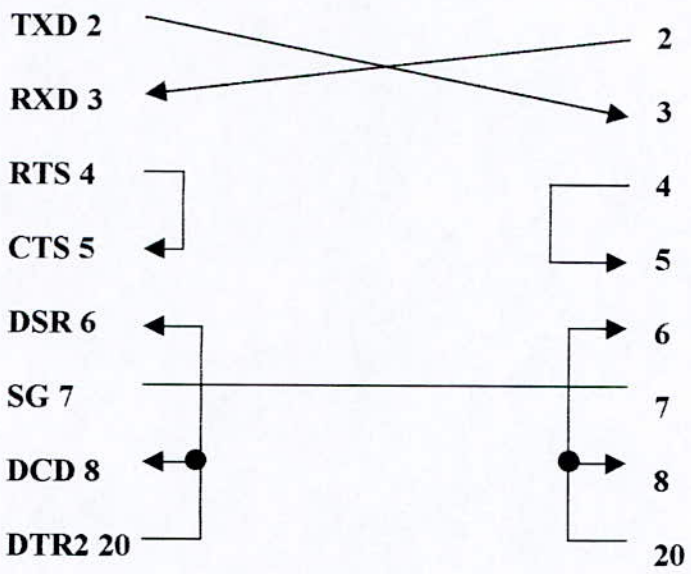


Fig.II.3 Interconnexion bifilaire DTE-DTE (casDB25).

II.4.1- PREMIERE CATEGORIE : TERRE :

La première catégorie, la plus simple est la terre. Comme ces signaux sont des signaux de protection et de référence, ils sont admis dans le câble sans modem.

II.4.2- DEUXIEME CATEGORIE : DONNEES :

Cette catégorie comprends les broches de données 2 et 3. La broche 2 sert à l'émission des données et la broche 3 sert à la réception des données.

Les données sont émises d'une machine par 2 et reçues par l'autre par 3. Pour permettre aux données d'être émises et reçues correctement sur les deux machines, il faut croiser la broche 2 à une extrémité avec la broche 3 de l'autre extrémité (Fig.II.4.).

Fig.II.4- Mode de connexion des broches 2 et 3.

II.4.3- TROISIEME CATEGORIE : BROCHES DE CONTROLE :

Le câble de connexion croisée doit permettre la connexion aux broches du DTE ainsi qu'aux broches du DCE. les signaux de contrôle sont :

1. RTS ;
2. DTR ;
3. DSR
4. CTS ;
5. DCD ;

Pour émuler le DSR aux deux extrémités il faut fixer le signal DTR de l'une des machines à la broche 6 de l'autre extrémité. Lorsque l'on relie la broche 20 à la broche 6 l'autre machine indiquera toujours que la ligne de transmission est disponible ; dans le cas contraire, l'autre machine ne recevra pas le signal DSR ce qui implique que le chemin de communication n'est pas établi.

Le signal RTS doit faire une boucle pour prévenir vers le DTE d'origine. Pour cela le RTS doit être recablé vers la broche 5 (CTS).

La machine réceptrice doit être prévenue de l'arrivée des données , il faut que le DCD (broche 8) provienne de la même source (RTS). Il faut donc connecter la broche 4 du DTE à la broche 8 de l'autre extrémité.

II.4.4- QUATRIEME CATEGORIE : BROCHES DE SYNCHRONISATION :

Il y a émulation des signaux suivants, vu le câble sans modem : DSR, CTS, DCD. Dans le cas de la (Fig.II.5), on peut avoir transmission dans les deux sens que les terminaux soient configurés pour la connexion en mode half-duplex ou full-duplex.

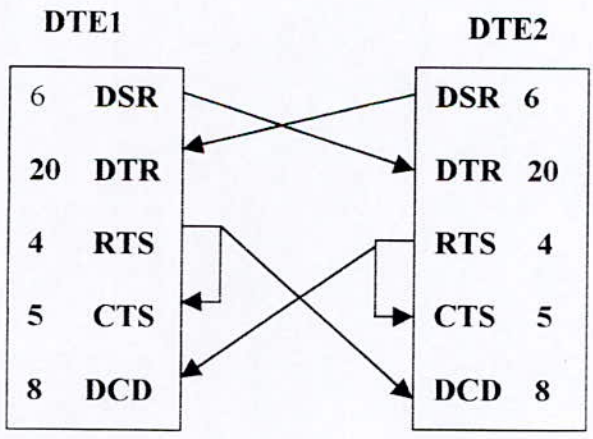


Fig.II.5- Dialogue entre deux DTE.

Dialogue entre deux DTE :

Le DTE1 met le RTS en position (demande d'instaurer le dialogue), le RTS active le CTS et il y a détection de porteuse par le DTE2 (activation du DCD) et ainsi DTE1 peut commencer à émettre.

Le DTE émetteur maintient le RTS en position jusqu'à ce que toutes les données soient émises, il éteint ensuite son RTS ce qui provoque l'abaissement du DCD du correspondant et du CTS local, libérant ainsi la ligne.

CHAPITRE III :

ETUDE DU MICRO CONTROLEUR 68HC11

III.1- INTRODUCTION : [15] [16] [17] [18] [21] [22]

Le 68HC11, est un micro contrôleur 8 bits de Motorola.
La majorité des 68HC11 adopte l'architecture interne qu'on peut découvrir dans son schéma synoptique présenté dans la page suivante.

Autour de l'unité centrale du 68HC11propement dite, qui est dérivée du vieux 6801, on trouve tout d'abord la mémoire.

Elle est subdivisée au maximum en trois blocs distincts dont la taille et la présence varient selon les références exactes du circuit.
La ROM qui peut être présente ou absente. Lorsqu'elle est présente elle peut être du types suivants :

- ROM programmable par masque (lors de la fabrication du circuit)
- EPROM : ROM programmable électriquement. Dans ce cas la ROM peut être non effaçable ; elle s'appelle alors OTPROM c'est-à-dire ROM programmable une seule fois. Elle peut aussi être du type UVPROM c'est-à-dire programmable électriquement et effaçable aux ultraviolets.

En plus de la ROM on trouve une EEPROM (mémoire programmable et effaçable électriquement) qui est destinée aux données sauf dans des cas rares tels que le test ou la mis au point de courts programmes.

Enfin une RAM est intégrée avec le système. Elle est réservée aux divers échanges de données.

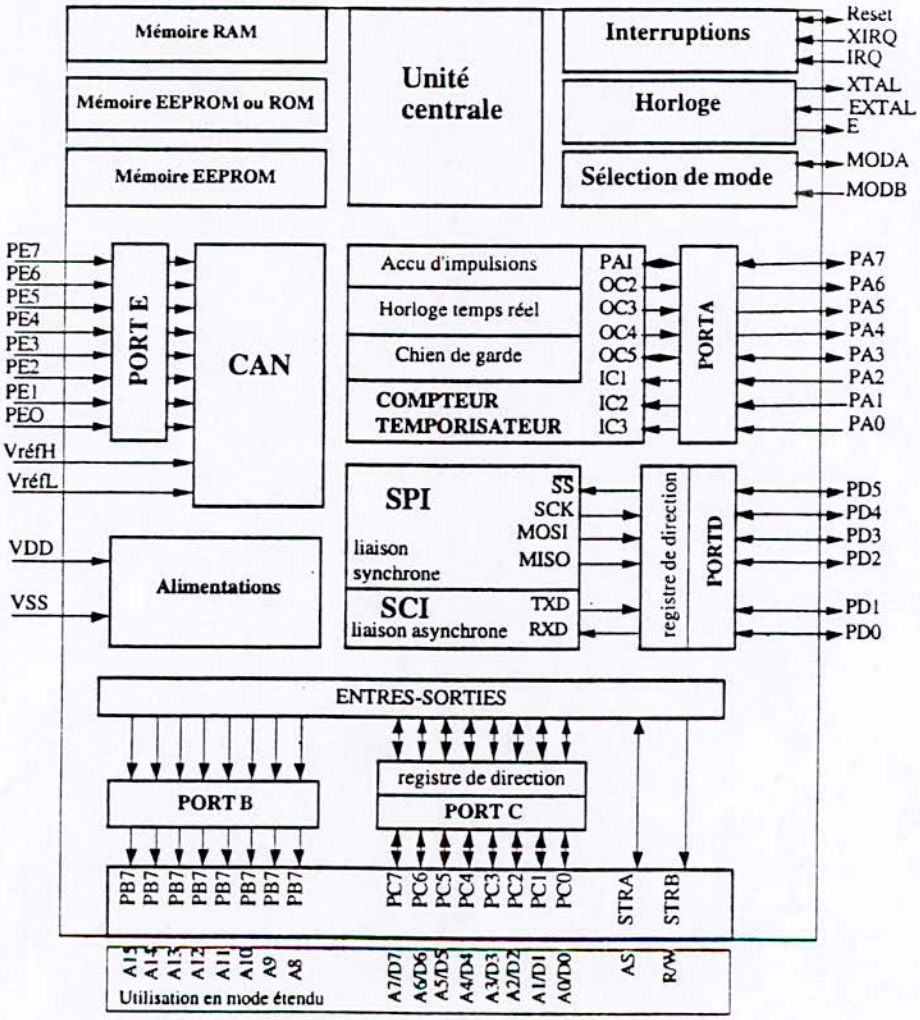
L'unité centrale est entourée d'un certain nombre de ports parallèles baptisés portA à portE qui peuvent être bidirectionnels ou unidirectionnels selon le cas. Certaines lignes de ces ports sont également partagées avec d'autres ressources internes et ne sont donc pas nécessairement accessibles directement en permanence. Ainsi par exemple le port E est-il commun avec le convertisseur analogique numérique.

Des entrées/soties séries sont aussi disponibles et peuvent fonctionner en mode synchrone ou asynchrone selon que l'on utilise la SPI ou la SCI.

Un timer est également disponible. Il comporte plusieurs timers très évolués ainsi qu'un accumulateur d'impulsions, une horloge temps réel et un 'chien de garde destiné à surveiller le fonctionnement du micro contrôle.

Un convertisseur analogique numérique à 8 entrées complète cette panoplie de ressources internes.

Toute la logique nécessaire, tant au traitement des interruptions qu'à la génération de l'horloge, est intégrée dans le 68HC11 dont la mise en oeuvre matérielle est ainsi fort simple.



Le schéma synoptique du MC68HC11.

III.1.1- FONCTIONS REALISEES PAR LE MC 68HC11 :

Transfert unidirectionnel ou bidirectionnel de données en parallèle (ports A, B, C et D) ;

- Transfert de données en série synchrone (SPI sur le port D) ;
- Transfert de données en série asynchrone (SCI sur le port D) ;
- Temporisation comptage (sur le port A) permettant :
  - de générer des intervalles de temps ;
  - de mesurer des intervalles de temps ;
  - de compter des impulsions ;
  - de générer des impulsions en temps réel (horloge temps réel) ;
  - de surveiller l'horloge et le bon fonctionnement du programme (COP ou chien de garde.) ;
- Conversion analogique digitale (le convertisseur dispose de 8 entrées multiplexées) ;

Les lignes d'entrées / sorties sont polyvalentes, elle peuvent être utilisées :

- Comme des entrées sorties parallèles unidirectionnelles ou bidirectionnelles (ports B et C) ;
- Comme des lignes d'adresses et de données (port B et C) quand on utilise le micro-controleur en mode étendu, c'est à dire avec des périphériques externes.

III.2- Les différents modes de fonctionnement : [15] [16] [17] [18] [21] [22]

III.2.1- LE MODE CIRCUIT SEUL (SINGLE SHIP):

C'est le mode micro-contrôleur proprement dit, le circuit remplit seul les fonctions données ci-dessus.

III.2.2- LE MODE ETENDU OU MULTIPLEXE (EXTENDED) :

Le micro-contrôleur utilise des ports d'entrées/sorties parallèles comme des bus d'adresses et de données, cela lui permet ainsi d'adresser 64 ko. Le bus d'adresses est constitué du port B pour les bits de poids fort et du port C pour les bits de poids faible, ce dernier étant multiplexé pour obtenir aussi le bus de données (figure 1)

Le signal read \ write est obtenu à partir de la ligne STRB et le signal de commande de multiplexage est obtenu à partir de la ligne STRA.

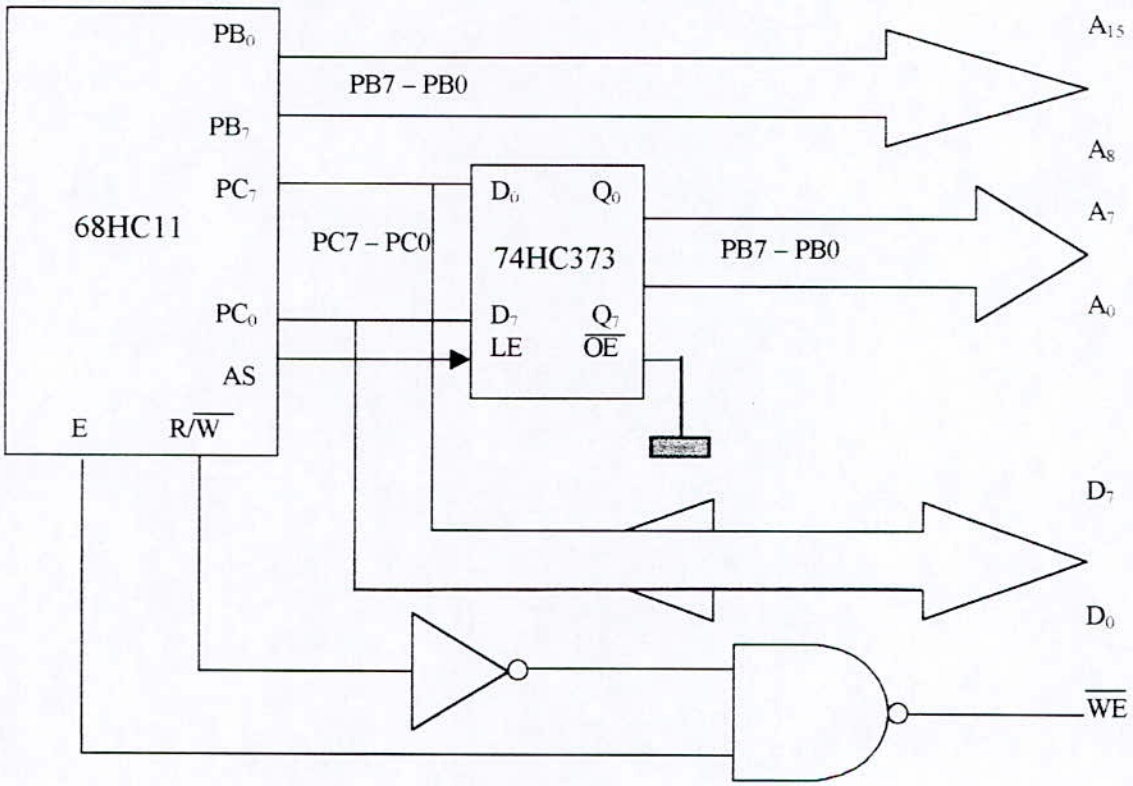


Fig III.1- Le mode multiplexé ou étendu.

III.2.3- LE MODE SPECIAL TEST

Ce mode est réservé au fabriquant pour tester le fonctionnement du micro-contrôleur en fin de chaîne de fabrication

III.2.4 .LE MODE SPECIAL BOOTSTRAP :

Ce mode charge en RAM un programme par la liaison série et lance sons exécution.

Il permet à l'utilisateur de brancher le micro-contrôleur avec la liaison série asynchrone de celui-ci sur un ordinateur et ainsi grâce au dialogue établi, de mettre au point et de charger un logiciel dans le micro-contrôleur. Dans celui-ci, le fabricant a placé un logiciel en ROM (de $BF40 à $BFFF) appelé chargeur (boot loader) et son vecteur RESET .

Au moment de RESET, le micro-contrôleur lit les deux connexions MODA et MODB, si celles-ci sont toutes les deux à zéro, il exécute le programme boot loader. Celui-ci initialise la liaison série pour établir la liaison avec l'ordinateur. Une fois cette liaison établie, l'ordinateur envoie le logiciel à tester dans la RAM du micro-contrôleur, puis lance celui-ci à partir de l'adresse zéro. Bien entendu, pour pouvoir fonctionner avec des branchements et des interruptions il est nécessaire de prévoir aussi dans la RAM un emplacement pour la PILE et

un emplacement pour les vecteurs d'interruption. Ceux-ci sont alors appelés pseudo vecteurs (TabIII.1) .

ADRESSE	ORIGINE
00C4 - C6	SCI
00C7 - C9	SPI
00CA - CC	ACCU. IMPULSIONS ENTREE
00CD - CF	DEBORDEMENT ACCU. IMPULSION
00D0 - D2	DEBORDEMENT TIMER
00D3 - D5	TOC 5
00D6 - D8	TOC 4
00D9 - DB	TOC 3
00DC - DE	TOC 2
00DF - E1	TOC 1
00E2 - E4	TIC 3
00E5 - E7	TIC 2
00E8 - EA	TIC 1
00EB - ED	INTERRUPTION TEMPS REEL
00EE - F0	IRQ
00F1 - F3	XIRQ
00F4 - F6	SWI
00F7 - F9	CODE OPERATION ILLIGAL
00FA - FC	COP
00FD - FF	SURVEILLANCE D'HORLOGE

Tab III.1- Les pseudo-vecteurs

III.3- Choix du mode de fonctionnement : [15] [16] [17] [18] [21] [22]

Le choix se fait pendant la période de RESET au démarrage de micro-contrôleur, celui-ci lit l'état des lignes MODA et MODB et déduit le mode de fonctionnement selon le tebleau (tab.III.2).

Ce choix peut aussi se faire grâce aux bits de registre HPRIO en mode spécial uniquement.

MODA	MODB	MODE DE FONCTIONNEMENT
1	0	CIRCUIT SEUL
1	1	ETENDU
0	0	BOOT-STRAP
0	1	SPECIAL TEST

Tab.III.2- Choix du mode.

III.4- Description des connexions : [15] [16] [17] [18] [21] [22]

V_{DD} et V_{SS} : Ce sont des connexions d'alimentation du circuit. Ce circuit est réalisé en technologie HCMOS, il est alimenté avec une tension de 5 volts ce qui le rend compatible avec le circuit CMOS et TTL. Il est conseillé de bien découpler cette alimentation (commutation rapide).

MODA et MODB : Ces deux signaux permettent de choisir (pendant la durée du RESET) le mode de fonctionnement du micro-contrôleur.

En dehors de cette phase de RESET la ligne MODA devient LIR (loader instruction registre :chargement du registre d'instruction) et peut être employée pour synchroniser un appareil de mesure durant la mise au point d'un programme. LIR passe au niveau logique zéro pendant le premier cycle d'horloge E de l'exécution d'une instruction.

MODB peut servir à alimenter la RAM interne dés que la tension V_{DD} devient inférieur à -0,7 V (Fig.III.2). La commutation se fait automatiquement grâce à une logique interne

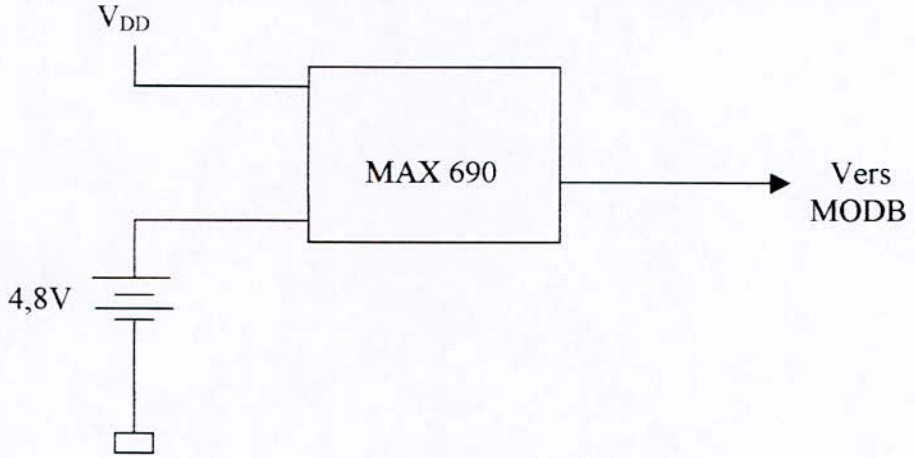


Fig.III.2 Alimentation de la RAM interne

EXTAL et XTAL : Remettent la connexion de quartz de l'horloge ou d'un horloge externe.

Généralement on utilise un quartz de 8 Mhz mais on peut faire descendre cette fréquence bien en dessous à condition de faire attention à la limite fixée par le surveillant de l'horloge. la fréquence l'horloge de bus est égale au quart de fréquence de quartz. ces deux connexions peuvent aussi être utilisées comme entrée d'horloge externe :EXTAL, ou sortie d'horloge : XTAL afin par exemple de synchroniser deux micro-contrôleur 68HC11 entre eux.

Signal E : sortie de l'horloge bus, cette ligne est surtout utilisée en mode étendu.

VREFL et VREFH : ces deux connexions permettent d'appliquer les tensions de référence utiles au convertisseur analogique/numérique. Ces tensions ne doivent pas dépasser les tensions d'alimentation du 68HC11 pour ne pas endommager celui-ci et l'écart entre VREFH et VREFL ne doit pas descendre en dessous de 2,5 V pour garder une bonne précision au convertisseur. Ces entrées doivent être soigneusement découpler .

III.5- Configuration de la mémoire et des registre par l'utilisateur : [15] [16] [17] [18] [21] [22]

Les emplacement des registres et des zones mémoire peuvent être modifiés par programmation de bits dans certaines registres. Plusieurs registres de contrôle sont protégés en écriture sauf en certaines circonstances (modes spéciaux). Cette protection inclut la possibilité d'écrire ces bits une fois et une seule pendant les 64 cycles d'horloge E qui suivent un RESET.

III.5 .1- LE REGISTRE CONFIG $103F :

Ce registre compte en réalité deux registres distincts, l'un en EEPROM qui conserve les informations en permanence et l'autre est un registre de travail normal en RAM. Après un RESET le registre en EEPROM est recopié dans le registre de travail. Une fois le micro-contrôleur en route il devient possible de modifier le registre en EEPROM par une opération particulière (programmation).

Le registre CONFIG est protégé par la mise à un du bit PTCON du registre BPROT sauf pour les modèles A_{XX}. la modification nécessite la même procédure d'effacement et de programmation que l'EEPROM. ce bit n'existe pas sur tous les modèles.

CONFIG	EE3	EE2	EE1	EE0	NOSEC	NOCOP	1	EEON
Circuit seul	1	1	1	1	P	P	1	1
Bootstrap	1	1	1	1	P	P	1	1
Etendu	P	P	P	P	1	P	1	P
Test	P	P	P	P	1	P	1	0

Tab.III.3-Les différents bits du registre CONFIG selon le mode d'utilisation.

EE3..... EE0 déterminent l'emplacement de L'EEPROM , celle-ci peut être placée n'importe où sur les 64 KO de l'espace adressable et ceci par secteur de 4 KO. Par exemple : si on a EE3...EE0=(0000) l'EEPROM va de l'adresse $0800 jusqu'à $0FFF, et si EE3...EE0=(1111)l'EEPROM va de $F800 à $FFFF.

NOSEC : (EEPROM securety disable) ce bit est normalement à un lors de la fabrication du circuit, ce qui veut dire la sécurité est enlevée, toutefois on peut demander au fabricant de mettre cette sécurité en service. Sécurité veut dire que le circuit ne peut être employé qu'en mode circuit seul ou en mode boot strap, ceci interdit la relecture de logiciel qui ne peut se faire qu'en mode étendu.

NOCOP : Ce bit permet d'enlever le système de surveillance de fonctionnement du logiciel(COP) appelé chien de garde.

EEON : Veut dire que l'EEPROM enable, ce bit est normalement forcé à un. Si ce bit est à zéro la zone mémoire EEPROM est retirée de la carte mémoire.

III.5.2- LE REGISTRE INIT $103D :

Ce registre permet la modification par pas de 4KO les emplacements des registres et de la mémoire RAM. Il est protégé car il ne peut être écrit que pendant les 64 cycles d'horloge E qui suivent un RESET.

INIT	RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0
Au reset	0	0	0	0	0	0	0	1

Tab.III.4-les affectations de déférents bits du registre INIT au moment de RESET.

RAM3...RAM0 : donnent l'emplacement de la RAM par pas de 4 KO dans l'espace adressable.

REG3...REG0 : donnent l'emplacement des registres par pas de 4 KO dans l'espace adressable

Remarque :

On peut remarquer qu'il y a un risque de chevauchement des registres sur la zone mémoire RAM. Les registres sont prioritaires sur la RAM et les octets communs sont dévalidés de la zone RAM.

III.5.3- LE REGISTRE OPTION $1039 :

Ce registre permet la mise en service d'un certain nombre de fonction. Certains de ses bits sont protégés (IRQE, DLY, CR1et CR0) et ne peuvent être modifiés que pendant les 64 premiers cycles d'horloge E qui suivent un RESET.

OPTION	ADPU	CSEL	IRQE	DLY	CME	0	CR1	CR0
Au reset	0	0	0	1	0	0	0	0

Tab.III.5-les affectations des différents bits du registre OPTION au moment de RESET.

ADPU (A /D power up) : Ce bit mis à un permet la mise en service de la pompe de charge du convertisseur analogique numérique (CAN) . Il faut attendre 100 µs avant d'utiliser le CAN.

CSEL (clock sélect) : Permet la sélection de l'horloge de la pompe de charge du convertisseur et du système d'écriture en EEPROM. Ce bit doit être à 1 si la fréquence d'horloge est trop faible pour la pompe de charge.

Si CSEL=1 :l'horloge est le signal E.

Si CSEL=0 : l'horloge est fixé par un circuit RC interne.

IRQE (IRQ enable) : Ce bit détermine le signal actif de l'entrée de demande d'interruption ITQ.

IRQE=0, l'entrée est sensible au niveau logique 0

IRQE=1, l'entrée est sensible à un front(descendant),

DLY (enable oscillator startup DELY) délai de mise en route de l'oscillateur.

DLY=1, le temps d'attente est de 4064 cycles d'horloge.

DLY=0, le temps d'attente est de 4 cycles d'horloge.

CME (clock monitor enable) mise en service de la surveillance d'horloge.

CME=1, surveillance en service.

CME=0, surveillance hors service.

CR_0 ,CR_1 (cop timer rate select) programmation de la période d'horloge du chien de garde.

CR_1	CR_0	$E/2^{15}$ divisée par :
0	0	1
0	1	4
1	0	16
1	1	64

Tab.III.6-Facteur de division de $E/2^{15}$ suivant les bits CR_1,CR_0 .

III.5-4- LE REGISTRE BPROT $1035 :

Ce registre permet la protection du registre CONFIG et la protection par bloc de 512 Octets de l'EEPROM.

BPROT	0	0	0	PCTON	BPRT3	BPRT2	BPRT1	BPRT0
Au reset	0	0	0	1	1	1	1	1

Tab.III.7- Les affectations des différents bits du registre BPROT au moment du RESET.

PTCON :

PTCON=1, le registre CONFIG ne peut être effacé ou programmé.

PTCON=0, le registre CONFIG peut être effacé et programmé normalement.

BPRT3...BPRT0 : ces quatre bits sélectionnent le paquet de 512 octets qui doivent être protégés, si le bit est à 1 la protection est en service.

III.5.5- LE REGISTRE TMSK2 $1024 :

Les deux bits de poids faible de ce registre ne peuvent être modifiés que pendant les 64 cycles d'horloge E qui suivent le RESET

Ils déterminent le rapport de pré-division appliqué à l'horloge E pour la commande du compteur libre TCNT.

PR1	PR0	FACTEUR DE PREDIVISION
0	0	1
0	1	4
1	0	8
1	1	16

Tab.III.8 Facteur de pré-division appliqué à l'horloge E suivant les bits PR1,PR0.

Remarque :
au RESET les deux bits PR1 et PR0 sont mis à zéro.

III.5.6- CONFIGURATION DE LA MEMOIRE :

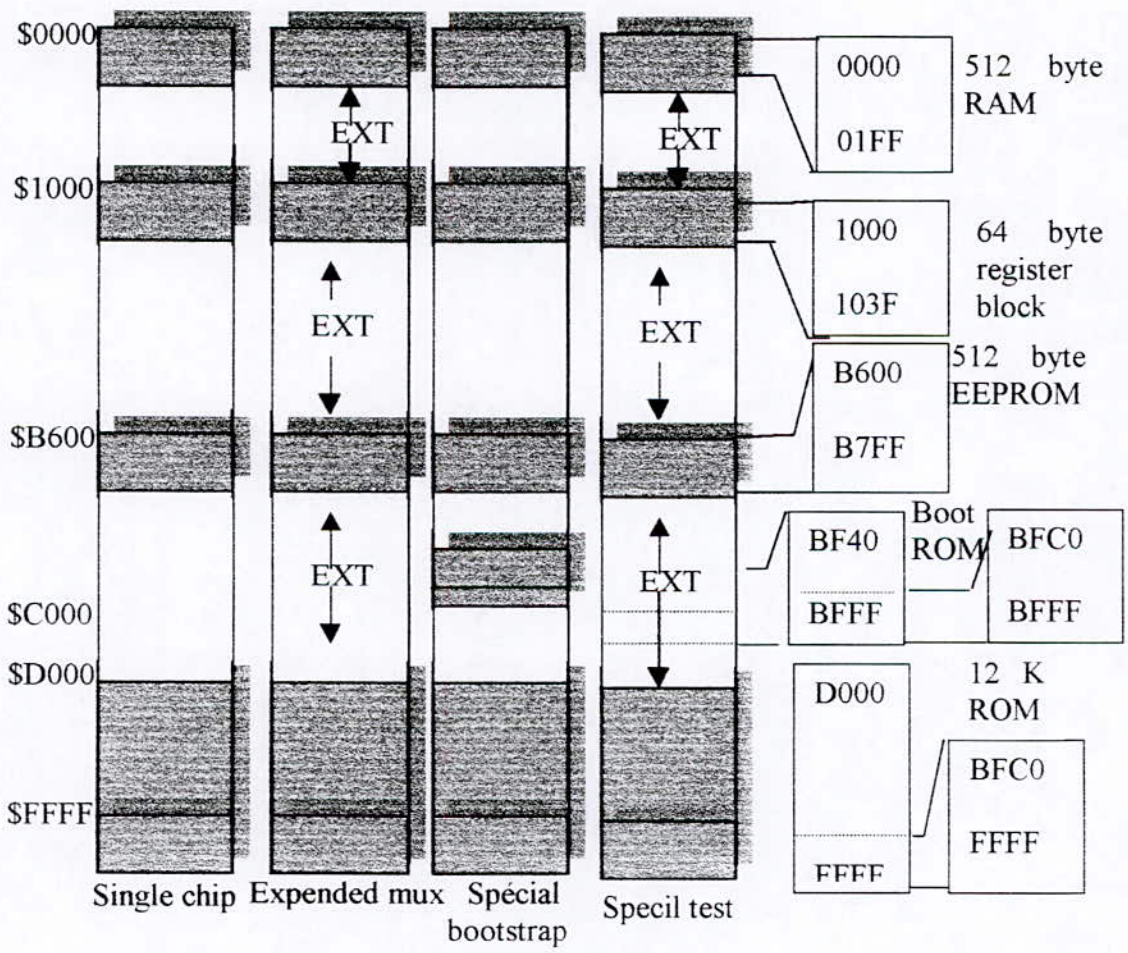


Fig.III.3- Cartographie mémoire.

III.6- Utilisation de l'EEPROM : [15] [16] [17] [18] [21] [22] [29] [30]

l'EEPROM du 68HC11 peut être :

- Effacée octet par octet, ligne par ligne ou entièrement en une seule fois.
- Ecrite et programmée par une procédure particulière.
- Protégée.

La haute tension nécessaire est générée par une pompe de charge interne.

III.6.1- MODES DE PROGRAMMATION DE L'EEPROM :

la programmation de l'EEPROM est contrôlée par le registre PPROG.

PROG	ODD	EVEN	0	BYTE	ROW	ERASE	EELAT	EEPGM
Au reset	0	0	0	0	0	0	0	0

Tab.III.9 Les affectations des bits du registre PPROG au moment de RESET.

ODD EVEN : réservés au fabriquant.
BYTE ROW :

BYTE	ROW	Effacement
0	0	Effacement complet
0	1	Effacement d'une ligne
1	0	Effacement d'un octet
1	1	Effacement d'un octet

Tab.III.10-Types d'effacement selon les bits BYTE et ROW.

ERASE :autorisation d'effacement.
ERASE=1 effacement
ERASE=0 écriture et lecture de l'EEPROM.

EELAT (EEPROM latch control) contrôle de verrouillage de l'EEPROM. Ce bit, quant il est à 1 permet la programmation de l'EEPROM en provoquant la configuration des données et des adresses et le temps nécessaire à l'écriture ou à l'effacement.

Si ce bit est à 0 la mémoire peut être lue.

EEPGM :(EEPROM programming voltage enable) mise en service de pompe de charge.
EEPGM=1 permet la création de la tension nécessaire à la programmation.

Remarque :

Le registre CONFIG est lui même en EEPROM il faut donc utilisé la même procédure pour l'effacer ou le modifier. Par sécurité il ne peut être effacé avec le mode effacement total.

III.6.2- PROTECTION DU CONTENU DE LA MEMOIRE :

Afin d'éviter le piratage du logiciel contenu dans la mémoire il est possible d'empêcher la relecture des mémoire RAM et EEPROM. Si l'option sécurité est en place grâce à la mise en place du logiciel approprié à la fabrication le bit NOSEC du registre OPTION, mis à zéro interdit la lecture et la copie des données en mémoire en interdisant l'utilisation du mode étendu. Le bootloader est utilisé pour supprimer l'option sécurité.

III.6.3- ORGANIGRAMME D'EFFACEMENT ET DE PROGRAMMATION DE L'EEPROM :

Pour effacer ou programmer l'EEPROM il faut la déprotéger par la mise à zéro des bits BPROT3...BPROT0 appropriés du registre BPROT, ensuite on positionne le bit EELAT du registre BPROG à 1 ou bien le bit ERASE à un, puis on choisit le mode d'effacement désiré en positionnant les bits ROW et BYTE du registre PPROG aux niveaux appropriés, ensuite on applique la tension de programmation (pompe de charge)avec le bit EEPGM du registre PPROG à un en faisant attention à la fréquence d'horloge, on attend 20 ms pour que la pompe de charge ait le temps de fournir la haute tension, finalement on annule la haute tension (EPGM=1). La donnée doit maintenant être écrite en mémoire.

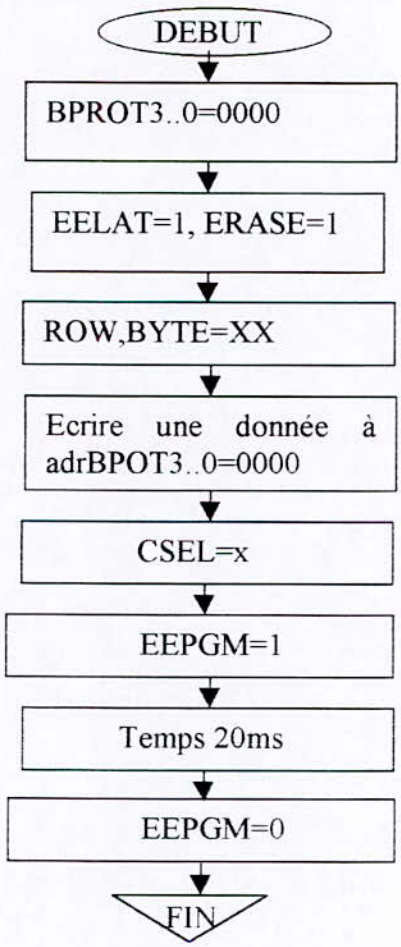


Fig.III.4- Organigramme d'éffacement et de programmation de l'EEPROM.

III.6.4- ORGANIGRAMME DE MODIFICATION DU REGISTRE CONFIG

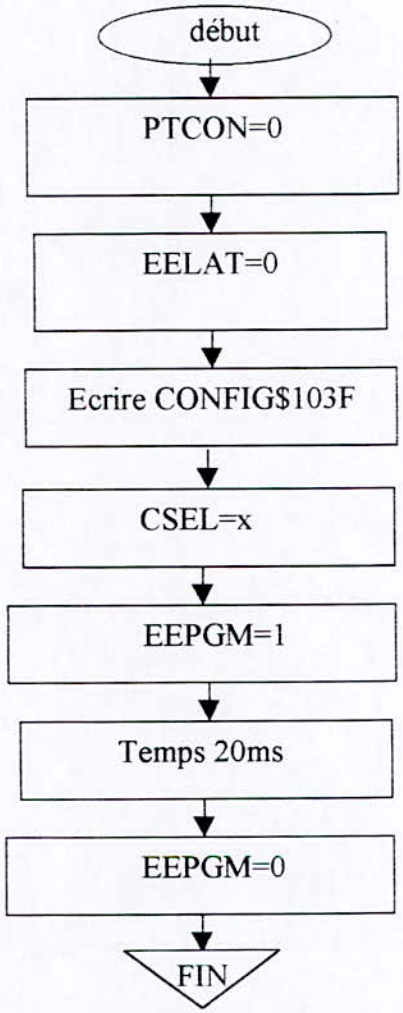


Fig.III.5-Organigramme de modification du registre GONFIG.

Remarque :

Après modification du registre CONFIG la lecture de ce registre en RAM ne donne pas la nouvelle valeur mais l'ancienne car le registre est copié à cette adresse($103F) au moment de RESET. Pour retrouver la nouvelle valeur il faut donc effectuer un RESET.

III.7- Le reset : [15] [16] [17] [18] [21] [22]

Après avoir placé le programme en mémoire on peut se demander comment faire pour le lancer et ainsi mettre en route l'application. l'opération RESET permet ce démarrage. Elle consiste à placer un niveau logique zéro sur la connexion appelée RESET du boîtier du micro-contrôleur. Une fois cette entrée détectée par l'unité centrale, une procédure d'initialisation

interne est déclenchée. celle-ci va aller chercher dans la case mémoire $FFFE et $FFFF l'adresse de départ de logiciel et va la placer dans le compteur programme, qui déclenche l'exécution du logiciel.

La connexion RESET est une ligne bidirectionnel ; elle initialise un certain nombre de registres afin que rien de dangereux ne puisse arriver aux interfaces avant que le programme de l'utilisateur ne vienne les configurer ; elle provoque à son déclenchement la mise à l'état bas (en sortie) la ligne RESET permettant ainsi le déclenchement de RESET sur les périphériques externes.

III.7.1- LES QUATRE SOURCES DE RESET :

III.7.1.1- POR (POWER ON RESET) : A LA MISE SOUS TENSION :

Ce type de RESET est destiné à initialiser le micro-contrôleur, il est généré lors d'une transition positive de V_{DD} et utilisé seulement pour l'initialisation de démarrage.

III.7.1.2- RESET EXTERIEUR :

c'est un RESET qui est d'origine extérieur du micro-contrôleur (généré par un circuit externe).

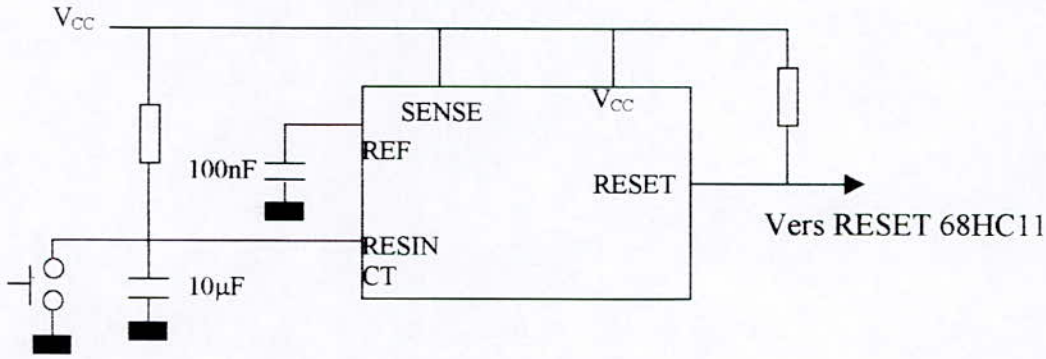


Fig.III.6.Le circuit de RESET .

III.7.1.3- SURVEILLANCE DE FONCTIONNEMENT DU LOGICIEL LE COP (COMPUTER OPERATING PROPRLY) :

Si le logiciel est perturbé le rechargement d'un compteur interne n'a plus lieu en temps voulu et un niveau zéro est placé pendant quatre cycles d'horloge sur la connexion RESET.

Le bit NOCOP de registre CONFIG permet la mise en service du COP le délai de rechargement est programmable avec les bits CR0 et CR1 du registre option.

Le vecteur associé se trouve en $FFFA et FFFB.

III.7.1.4-SURVEILLANCE DE L'HORLOGE :CM (CLOCK MONITOR) :

La surveillance de l'horloge CM déclenche un RESET si la période du signal d'horloge devient supérieur à une valeur déterminée par la constante de temps d'un circuit RC interne (5µS). ceci interdit de faire descendre la fréquence en dessous de 200Khz. Cette surveillance complète le COP car celui-ci ne peut détecter l'erreur d'horloge. Le vecteur associé se trouve en $FFFC et $FFFD.

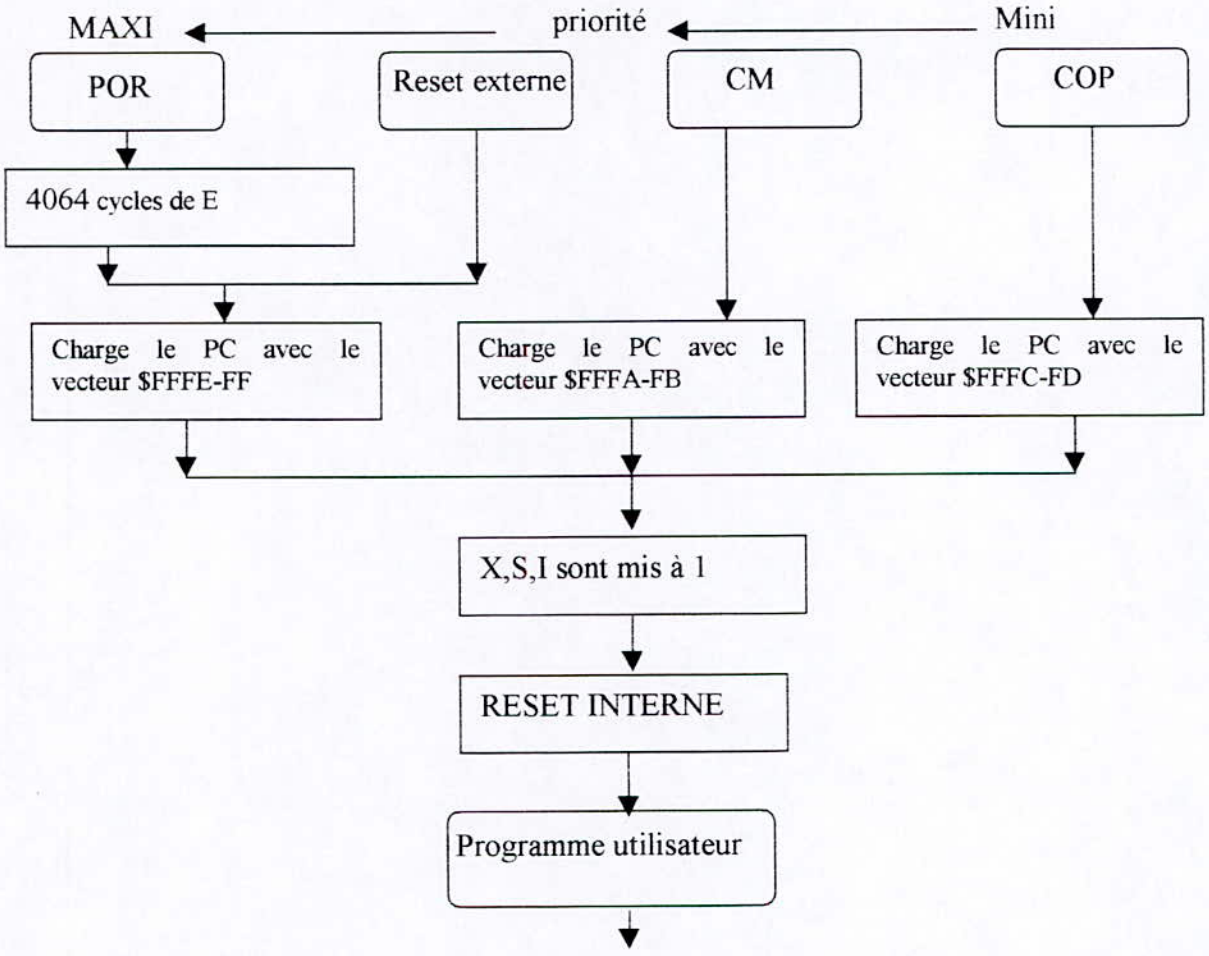


Fig.III.7.Les priorités de RESET.

III.8- Les interruptions : [15] [16] [17] [18] [21] [22]

Sur le boîtier du micro-contrôleur il existe des connexions d'entrées actives au niveau logique zéro et qui ont la possibilité de déclencher des interruptions dites externes('IRQ XIRQ). Dans un micro-contrôleur les interfaces qui sont intègres ont la possibilité d'effectuer des demande d'interruption.

Il existe aussi une instruction (SWI) qui déclenche une demande d'interruption. les circuits externes doivent maintenir le niveau logique bas pendant un temps suffisant pour que le micro-contrôleur puisse prendre en compte l'interruption.

III.8.1- OPERATIONS EFFECTUEES LORS D'UNE INTERRUPTION :

A la réception d'une demande d'interruption le micro-contrôleur termine l'instruction en cours puis :

- Sauvegarde tous les registres dans la pile dans un ordre précis.
- Décrimante le pointeur de pile du nombre d'octets chargé.
- Positionne le masque d'interruption correspondant du registre à l'état 1
- Le contenu du vecteur d'interruption correspondant est chargé dans le compteur programme (PC) et déclenche le sous programme d'interruption.

Une fois l'interruption est terminée l'instruction RTI permet :

- La récupération de tous les registres à partir de la pile.
- Décrémente le pointeur de pile du nombre d'octets correspondant.
- Charge le PC avec l'adresse de l'instruction du programme principal.

III.8.2 HIERARCHIE DES INTERRUPTION :

Les priorités de demande d'interruption sont définies par le constructeur selon l'ordre ci-dessous :

III.8.2.1- INTERRUPTIONS NON MASQUABLES :

- POR ou RESET externe
- CM
- COP
- XIRQ
- Interruption pour code illégal
- Interruption par programme

III.8.2.2- INTERRUPTIONS MASQUABLES :

- Débordement timer
- Accumulateur d'impulsion plein
- Front d'entrée sur l'accumulateur d'impulsion
- Fin de transmission série synchrone
- Fin de transmission série asynchrone
- IRQ
- Interruption par l'horloge temps réel
- Entrées de capture(1,2,3,)
- Sorties de comparaison (1,2,3,4)
- Entrée de capture 4 ou sortie de comparaison 5.

Le registre HPRIO permet la modification de cette hiérarchie grâce aux bits PSEL0...PSEL3 qui déterminent la source que l'on veut la plus prioritaire après XIRQ :

HPRIO	RBOOT	SMOD	MDA	IRV	PSEL3	PSEL2	PSEL1	PSEL0
Au reset	0	0	0	0	0	1	0	1

Tab.III.11-Affectation des divers bits du registre HPRIO au moment de RESET.

RBOOT : ce bit ne peut être écrit que si SMOD=1 il est mis à 1 pendant le RESET en mode spécial bootstrap.

RBOOT=1 la ROM de téléchargement se trouve en $BF40.

RBOOT=0 la ROM est retirée de la mémoire

SMOD :(special mode select) : peut être mis à 0 mais ne peut être remis à 1

SMOD=0 le mode normal est en service.

IRV(internal read visibility) : ce bit ne peut être modifié que si SMOD=1 il est utilisé pendant la fabrication (visibilité externe des accés internes en lecture)

III.8.3- ETUDE DES INTERRUPTIONS XIRQ, IRQ ET SWI :

XIRQ est une entrée de demande d'interruption non masquable, car pendant le déroulement d'un sous-programme d'interruption de type IRQ si le bit X est différent de zéro XIRQ peut toujours déclencher une autre interruption imbriquée dans la première et interdit toute nouvelle prise en compte .

IRQ est une entrée d'interruption masquable par le bit I du registre d'état elle est sensible par défaut à un niveau logique zéro mais peut être sensible à un front descendant en plaçant à 1 le bit IRQE du registre OPTION.

SWI est une instruction et n'est pas inhibée par les bits du registre d'état. l'exécution du SWI met à 1 les bits I et X.

III.9- Les interfaces parallèles: [15] [16] [17] [18] [21] [22]

III.9.1- LE PORT A $1000:

Ce port est normalement destiné aux interfaces compteur-temporisateur et au compteur d'impulsions mais il peut servir de port parallèle

PORT A	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
Au reset	HZ	0	0	0	HZ	HZ	HZ	HZ

Tab.III.12- Etat de différentes lignes du port A au reset.

PA0...PA2 sont des lignes d'entrée de données.

PA4...PA6 sont des lignes de sortie de données.

PA3 et PA7 peuvent être en sortie ou en entrée suivant DDR3 et DDR7 du registre PACTL($1026)

DDRx=0 PAx en entrée

DDRx=1 PA x en sortie (x=3 ou 7)

III.9.2- LE PORT D $1008:

Ce port est relié aux interfaces séries synchrone et asynchrone, on peut l'utiliser en port parallèle bi-directionnel. Si les interfaces série ne sont pas utilisées le sens de transfert des lignes est déterminé par un registre de direction DDRD ($1009).

Les connexions du port D peuvent être configurées en OU câblé, la sortie se trouvant alors en drain ouvert. Ceci permet de les brancher sur d'autres sorties de circuits logiques. Ce mode est obtenu en mettant à 1 le bit DWOM du registre de contrôle SPCR ($1028).

PORTD	-	-	PD5	PD4	PD3	PD2	PD1	PD0
Au reset	-	-	0	0	0	0	0	0

Tab.III.13-État du port D au RESET .

DDRD	-	-	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
Au reset	-	-	0	0	0	0	0	0

Tab.III.14-état du DDRD au RESET .

SPCR	-	-	DWON	-	-	-	-	-
Au reset	-	-	0	-	-	-	-	-

Tab.III.15-État du registre SPCR au RESET .

III.9.3- LE PORT B $1004 :

le port B est uni-directionnel (toujours en sortie) il peut être utilisé en port parallèle en mode monochip ou encore en bus d'adresse pour le mode étendu. Au port B est associé une ligne de commande STRB, celle-ci permet un nombre de fonctionnement automatique.

PORTB		PB7	PB5	PB4	PB3	PB2	PB1	PB0
Au reset	0	0	0	0	0	0	0	0

Tab.III.16-État du PORTB au RESET .

III.9.3.1- UTILISATION DE LA LIGNE STRB : LE MODE IMPULSIONNEL :

Ce mode de fonctionnement est sélectionné par la mise à zéro du bit HNDS du registre PIOC ($1002). La ligne STRB génére une impulsion d'une durée de 2 cycles d'horloge à chaque écriture sur le port B.

Le bit INVB du registre PIOC détermine la polarité de l'impulsion :

INVB=0 : impulsion négative (niveau logique actif 0) ;

INVB=1 : impulsion positive (niveau logique actif 1) ;

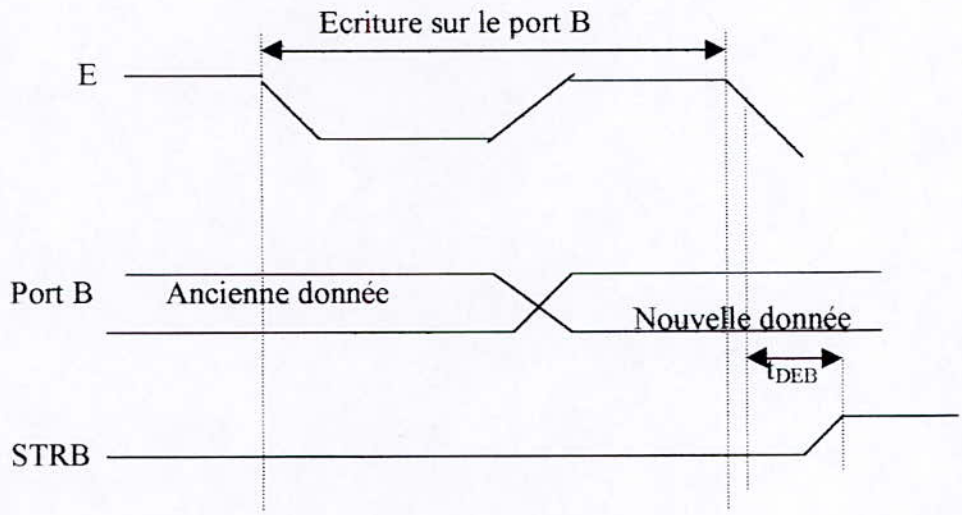


Fig.III.8. Mode impulsionnel sur le port B.

III.9.4- LE PORT E $100A :

Ce port est normalement destiné au convertisseur analogique/numérique si ce dernier n'est pas utilisé, les ligne de ce port sont utilisés en entrées.

PORT E	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0
Au reset	U	U	U	U	U	U	U	U

Tab.III.17- Etat du port E au moment de RESET.

NOTE : le signe U signifie que le registre n'est pas affecté par le RESET

III.9.5- LE PORT C $100D :

Ce port peut être utilisé en port parallèle bidirectionnel, ou encore en bus d'adresses ou de données en mode étendu.

Le sens de transfert est déterminé par le registre de direction DDRC($1007) qui est mis à zéro après chaque RESET.

Comme pour le port D, les lignes peuvent être commutées en drains ouverts pour la réalisation de la fonction OU câblée. Ceci permet de les relier à plusieurs sorties de fonctions logiques, cette disposition est déterminée par la mise à 1 du bit CWOM du registre PIOC.

9.6-ETUDE DU REGISTRE PIOC $1002 :

PIOC	STAF	STAI	CWOM	HNDS	OIN	PLS	EGA	INVB
Au reset	0	0	0	0	0	X	1	1

Tab.III.18- Etat du registre PIOC au RESET.

STAF(strob a flag) : indicateur associé à STRA, il est mis à 1 par une transition de celle-ci, sa remise à 0 se fait en deux étapes :

- La lecture du PIOC prépare la remise à 0 ;
- La lecture du PORTCTL (port tampon associé au port C)en mode impulsionnel et en mode dialogue en entrée, remet ce bit à 0, en mode dialogue en sortie l'écriture dans le PORTCTL remet STAF à 1.

STAI(strobe a interupt flag) : indicateur d'interruption associé à STRA

STAI=1 le passage à 1de STAF déclenche une demande d'interruption.

STAI=0 interruption masquée.

CWOM(port C wire or mode) : mode drain ouvert(ou cablé)

CWOM=0 le port est en CMOS normal.

CWOM=1 le port est mode drain ouvert.

HNDS(hand shake mode) :mode dialogue.

HNDS=0 STRA est une entrée de strobe qui verrouille les données dans le PORTCTL, STRB est une sortie de strobe après écriture dans le port B.

HNDS=1 on obtiens le mode dialogue du port C avec STRA et STRB.

OIN (output or input handshaking) :

Il détermine le sens du mode dialogue (en entrée ou en sortie).

PLS(pulse inter locked hand shake opération) : mode impulsionnel.

PLS=0 STRB activé reste active jusqu'à ce que le signal STRA soit détecté.

PLS=1 STRB forme une impulsion d'une durée de 2 périodes d'horloge E.

EGA (active adge of STRA) : détermine le front actif du STRA.

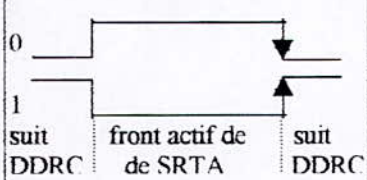
EGA=0 front descendant : le port C en mode dialogue suit le DDRC tant que STRA est à l'état bas, mais est forcé en sortie quand STRA passe à 1.

EGA=1 : front montant : si STRA=1 le port C suit le DDRC, mais le port est forcé en sotie dès que STRA passe à 0.

INVB(invert strobe B) :niveau actif du signal STRB.

INVB=0 STRB est une impulsion active au niveau 0.

INVB=1 STRB est active au niveau haut.

	STAF (R.A.Z.)	HNDS	OIN	PLS	EGA	Port C	Port B
Mode impulsion	Lire PIOC puis lire le PORTCTL	0	X	X	0 : front descendant. 1 : front montant.	Les entrées sont verrouillé-es dans le PORCTL sur chaque front actif de STRA	L'écriture sur le PORTB provoque une impulsion sur STRB
Mode dialogue en entrée	Lire PIOC puis lire le PORTCTL	1	0	0 :STRB actif sur niveau 1 :STRB actif sur pulse.	0 : front descendant. 1 : front montant.	Les entrées sont verrouillé-es dans le PORCTL sur chaque front actif de STRA	Sortie de port non affectée par le mode dialogue
Mode dialogue en sortie	Lire PIOC puis lire le PORTCTL	1	1	0 :STRB actif sur niveau 1 :STRB actif sur pulse.		Les entrées sont verrouillé es dans le PORCTL sur chaque front actif de STRA	Sortie de port non affectée par le mode dialogue

Tab.III.19- Résumé du fonctionnement du port C.

III.10- La transmission série d'information : [15] [16] [17] [18] [21] [22]

III.10.1- CARACRERISTIQUES DE LA TRANSMISSION SERIE ASYNCHRONE DE 68HC11 :

- La ligne en attente est à l'état haut ;
- Le bit de start est au niveau bas ;
- Le format de données est de 8 ou 9 bits ;
- Le bit de poids faible est transmit le premier ;
- 1 bit de STOP au niveau haut indique la fin de transmission ;
- signal de BREAK sépare les trames(données)

- l'émetteur et le récepteur sont indépendant donc permettant la transmission en FULL duplex
- L'interface permet le réveil automatique dés qu'elle reçoit des signaux valides.

10.2-UTILISATION DE LA LIAISON SERIE ASYNCHRONE (SCI :SERIAL COMMUNICATION INTERFACE) :

Entrées/sorties l'interface série asynchrone utilise la première et la deuxième ligne du port D : $Rx=PD_0$

$Tx=PD_1$

Adresses de port utilisés :

- $102B BAUD : registre de débit.
- $102C SCCR1 : registre de contrôle 1
- $102D SCCR2 : registre de contrôle 2
- $102E SCSR : registre d'état
- $102f SCDR : registre de données(DTR ou RDR)

III.10.2.1- L'EMISSION :

Il suffit d'écrire dans le registre SCDR , l'octet est ensuite transféré dans le registre à décalage, qui lui transmet sur la ligne, les bits les uns après les autres. Ce transfert met à 1 le bit DTRE du registre d'état SCSR, ceci peut alors générer une demande d'interruption si elle est autorisée, à la fin de transmission le bit de transmission complète(TC) passe à 1 et peut aussi générer une demande d'interruption si elle est autorisée.

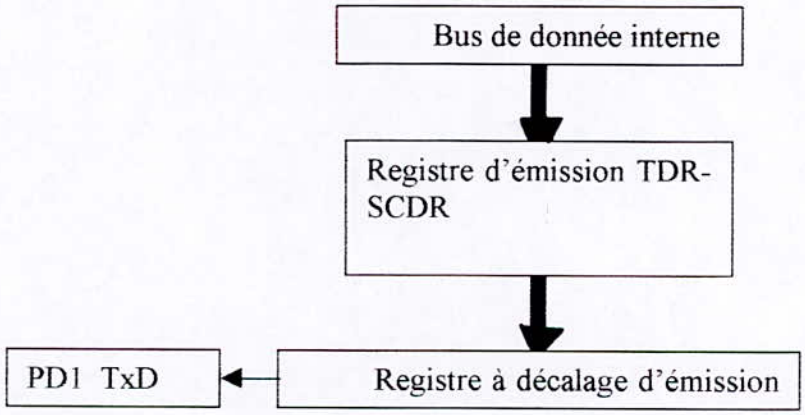


Fig.III.9- Schéma de l'interface d'émission

III.10.2.2-LA RECEPTION :

La lecture du registre SCDR suffit car celui-ci contient la dernière donnée reçue si le récepteur est validé. Le bit RDRF du registre d'état SCSR est mis à 1 pour indiquer que la données a été transférée dans le registre à décalage SCDR. Une interruption peut être générée(donnée reçue) si elle est autorisée.

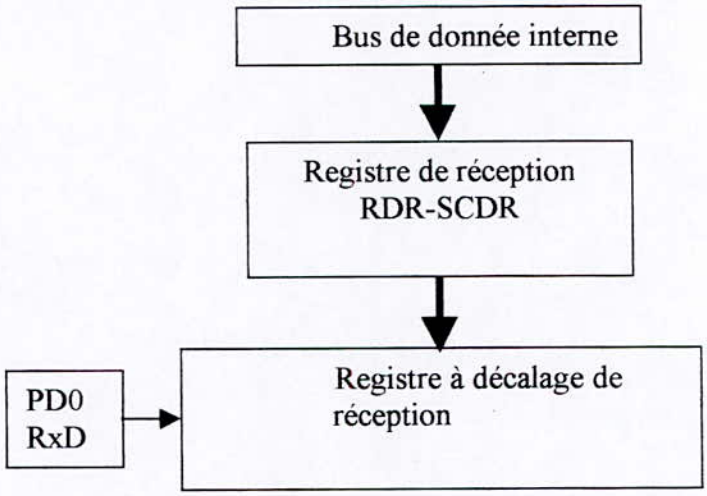


Fig.III.10-Schéma de l'interface de réception.

III.10.2.3- LE PRINCIPE DU REVEIL :

Il est possible de monter plusieurs récepteurs en réseau, le récepteur doit donc pouvoir détecter qu'un message va lui être envoyé, il est en sommeil jusqu'à ce que un message le réveille. Le réveil consiste à détecter le début du message qui contient l'adresse du destinataire, le logiciel dans ce type de récepteur évalue le premier caractère d'un message, si le message est prévu pour un autre récepteur il se remet en sommeil. Il existe deux possibilités de réveil, choisies par le bit WAKE du registre de contrôle 2 (SCCR2).

III.10.2.3.1- LE MODE ATTENTE :

Le mode attente est caractérisé par la mise à l'état 1 de la ligne RxD pendant au moins la durée d'un caractère (10 ou 11 bits).

Le récepteur se réveille à chaque fois qu'un caractère est reçu, quand la ligne RxD est en attente. Le système doit donc prévoir une telle durée entre 2 messages mais pas entre les caractères d'un message.

III.10.2.3.2- LE MODE ADRESS MARK :

Le récepteur se réveille quand il reçoit une information d'adresse. le bit le plus significatif du mot transmis est utilisé pour indiquer si le caractère est une adresse(MSB=1) ou une donnée (MSB=0), ensuite chaque récepteur doit déterminer si le message lui est destiné.

III.10.2.4- ETUDE DES REGISTRES :

III.10.2.4.1- LE REGISTRE DE CONTROLE 1 :SCCR $102C :

SCCR1	R8	T8	0	M	WAKE	0	0	0
Au reset	-	-	0	0	0	0	0	0

Tab.III.20 – Etat du registre SCCR1 au RESET.

R8 : si M=1 le neuvième bit du mot reçu est mis dans R8.
T8 : si M=1 T8 contient le neuvième bit du caractère à transmettre.
M : longueur de caractère

M=0 8 bits de donnée ;
M=1 9 bits de donnée ;

WAKE : sélection du mode de réveil

WAKE=0 mode attente ;
WAKE=1 mode ADRESS MARK ;

III.10.2.4.2-LE REGISTRE DE CONTROLE 2 SCCR2 $102D :

SCCR2	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Au reset	0	0	0	0	0	0	0	0

Tab.III.21–Etat du registre SCCR2 au RESET.

TIE : interruption autorisée pour le transmetteur vide :
TIE=0 : interruption inhibée ;
TIE=1 :interruption autorisée ;
TCIE : interruption autorisée pour transmission terminée :
TCIE=0 : interruption inhibée ;
TCIE=1 : interruption validée ;
RIE : interruption autorisée pour récepteur plein :
RIE=0 : interruption inhibée ;
RIE=1 :interruption autorisée
ILIE : : interruption autorisée en mode attente :
ILIE=0 : interruption inhibée ;
ILIE=1 : interruption validée ;
TE : Transmetteur prêt :
TE=1 la sortie du registre à décalage est reliée à TxD pour l'émission ;
TE=0 la ligne dépend du DDRD1 ;
RE : récepteur prêt :
RE=1 la ligne PD0=RxD est forcée en entrée ;

RE=0 la ligne dépend du DDRD0 ;

RWU :mis en fonction du réveil .

SBK : envoi d'un signal BREAK : si se bit est à 1 puis à 0 le transmetteur émet 10 ou 11 bits au niveau 0.

Si SBK reste à 1 : le émetteur envoie continuellement des blocs de 0(signal BREAK) jusqu'à ce que SBK repasse à 0.

III.10.2.4.3- LE REGISTRE D'ETAT SCSR($102E) :

SCSR	TDRE	TC	RDRF	IDLE	OR	NF	FE	0
Au reset	1	1	0	0	0	0	0	0

Tab.III.22- Etat du registre d'état au RESET.

TDRE :registre de transmission vide :

Mis à 1quand le registre de transmission a été transféré dans le registre à décalage. Et remis à 0 par une lecture de registre SCSR suivie d'une écriture dans le registre SCDR

TC transmission terminée : mis à 1 à la fin d'une transmission et remis à 0 par une lecture du registre d'état suivie d'une écriture dans le registre de transmission.

RDRF : registre de réception plein : ce bit est mis à 1 quand le registre à décalage a été transféré dans le SCDR il est remis à 0 par une lecture de registre d'état suivie d'une lecture de registre de réception(SCDR)

IDLE : détection de mode attente : mis à 1 par le détection du mode attente sur la ligne (niveau haut pendant une durée d'au moins un caractère : 10 ou 11 bits) ;il est remis à 0 par la lecture du registre d'état suivie par la lecture du SCDR.

OR : (over run error) erreur d'écrasement : dans ce cas la donnée qui arrive dans e registre est perdue mais la donnée dans le SCDR qui n'a pas encore été lue n'est pas modifiée.

FE :erreur de format : mis à 1 si le bit STOP n'est pas détecté ,ce bit est remis à 0 par une double lecture.

III.10.2.4.4- LE REGISTRE DE DEBIT : BAUD ($102B) :

Le registre BAUD permet de programmer la vitesse de transfert des informations.

BAUD	TCLR	0	SCP1	SCP0	RCBK	SCR2	SCR1	SCR0
Au reset	0	0	0	0	0	U	U	U

Tab.III.23- Etat du registre BAUD au RESET.

Note : U signifie que le bit pas affecté par leRESET

TCLR:(clear baud rate counter): utilisé uniquement en mode test par le fabriquant pour remettre à 0le compteur de gamme de débit.

RSKB :utilisé en mode test pour établir un ou exclusif entre les horloges de réception et d'émission.

TCLR :utilisé en mode test pour remettre à 0 les bits de sélection de la vitesse de transmission.

SCP1-SCP0 : pré-diviseur (à partir de l'horloge $\phi 2$: même signal que E mais déphasé de 90^0).

SCP1	SCP	TAUX	Vitesse maxi avec un quartz de 8MHZ
0	0	1	125000 bauds
0	1	3	41667 bauds
1	0	4	31250 bauds
1	1	3	9600 bauds

Tab.III.24- Pré-division de $\phi 2$.

SCR2-SCR1-SCR0 : sélection de débit : Ces bits ne sont pas affectés par le RESET. Le tableau suivant donne les vitesses pour une prédivision maximale(9600 bauds)

SCR2	SCR1	SCR0	TAUX	Vitesse
0	0	0	1	9600
0	0	1	2	4800
0	1	0	4	2400
0	1	1	8	1200
1	0	0	16	600
1	0	1	32	300
1	1	0	64	150
1	1	1	128	075

Tab.III.25 Sélection de débit

III.11- La conversion analogique numérique : [15] [16] [17] [18] [21] [22]

III.11.1- INTRODUCTION :

Le convertisseur analogique numérique du micro-contrôleur 68HC11 fonctionne sur le principe de l'approximation successive obtenue par répartition de charges dans des capacités. La répartition des charges se fait grâce à des commutateurs analogiques. Pour qu'ils puissent commuter des tensions jusqu'à VDD (5V), il est nécessaire de les alimenter avec une tension d'au moins 8V. Ceci est rendu possible par l'utilisation de la pompe de charge .

L'utilisateur devra donc penser à mettre en route cette pompe de charge en plaçant le bit ADPU du registre OPTION ($1039) à 1, il devra aussi laisser le temps (100µs) à cette pompe de charge pour établir la 'haute tension'.

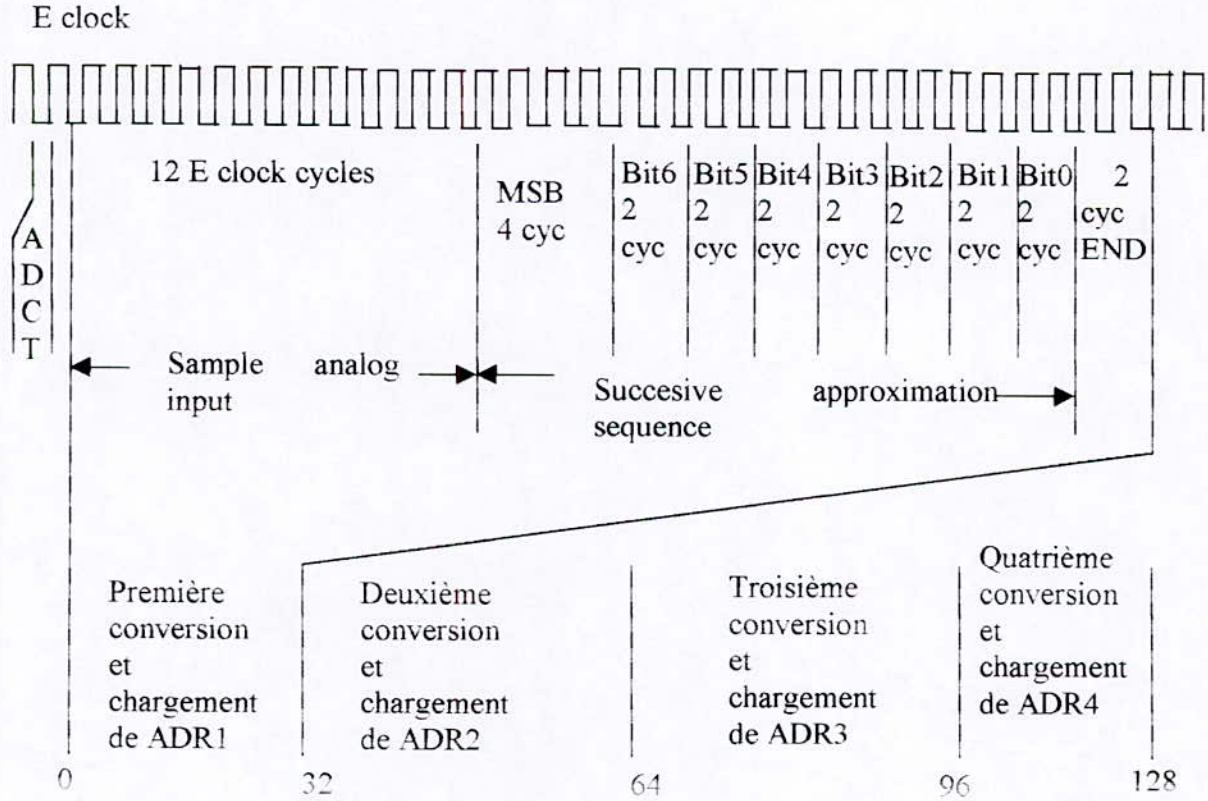
III.11.2- UTILISATION DU CONVERTISSEUR :

III.11.2.1- CARACTERISTIQUES :

Le convertisseur possède 8 entrées analogiques multiplexées. Chaque entrée peut recevoir une tension analogique ne dépassant pas 6 volts. Les entrées sont protégées contre les tensions inverses et la surtension par un dispositif de protection composé en partie d'une diode ZENER. Afin d'éviter le claquage de la diode il est bon de placer une résistance de quelques KΩ en série pour limiter le courant. Si la tension à convertir Vx est égale à la tension de référence haute (VRH) le convertisseur donnera la valeur $FF, et inversement si elle est égale à la tension de référence basse le convertisseur donnera la valeur $00.

III.11.2.2- CHRONOGRAMME DE CONVERSION:

Les conversions ont lieu successivement puis sont rangées dans les registres correspondants. Trente deux cycles d'horloge sont nécessaires pour réaliser une conversion. Compte tenu du fonctionnement par transfert de charges dans des capacités, il ne faut pas descendre la fréquence d'horloge du microcontrôleur en dessous de 750 KHZ. Le résultat de la conversion est placé dans le SAR puis est transféré dans le « registre ADRx correspondant pendant le cycle FIN. L'indicateur CCF est mis à 1 pendant le cycle FIN.



FigIII 11-Séquence de conversion analogique numérique

Le multiplexeur interne est capable de sélectionner 16 canaux (grâce aux 4 bits du registre de contrôle CA à CD). Huit canaux sont branchés sur les entrées du port E, 5 canaux sont réservés et 3 autres sont connectés sur les références internes VRH, VRL VRH/2 .

III.11.3-MODES DE FONCTIONNEMENT :

III.11.3.1-MODE CANAL UNIQUE: MULT=0 :

Dans ce mode de fonctionnement deux possibilités sont offertes :

SCAN=0 : il y a acquisition du canal quatres fois de suite, chacun des résultats étant mis en mémoire respectivement dans les registres ADR1 à ADR4 ;

SCAN=1 : il y a acquisition continue du canal dans les quatre registres, le cinquième résultat écrasant le premier dans ADR1. Ceci permet de suivre l'évolution du signale mesuré.

III.11.3.2-MODE CANAUX MULTIPLES : MULT=1 :

SCAN=0: il y a acquisition unique des quatre canaux sélectionnés et mise en mémoire dans les registres ADR1 àADR4.

SCAN=1 : il y a acquisition permanente des quatre canaux et mise en mémoire dans les registres la nouvelle donnée écrasant celle qui s'y trouve.

Canal	D	C	B	A	Signal traité	Mise en mémoire si MULT=1
1	0	0	0	0	AN0-E0	ADR1 :$1031
2	0	0	0	1	AN1-E1	ADR2 :$1032
3	0	0	1	0	AN2-E2	ADR3 :$1033
4	0	0	1	1	AN3-E3	ADR4 :$1034
5	0	1	0	0	AN4-E4	ADR1
6	0	1	0	1	AN5-E5	ADR2
7	0	1	1	0	AN6-E6	ADR3
8	0	1	1	1	AN7-E7	ADR4
9 à 12	x	x	x	x	Réservés	
13	1	1	0	0	VRH	ADR1
14	1	1	0	1	VRL	ADR2 } utilisé par le fabriquant
15	1	1	1	0	VRH/2	ADR3
16	1	1	1	1	Réservé	ADR4

Tab.III.26- Affectation des canaux.

III.11.4- UTILISATION DES REGISTRES :

III.11.4.1 –LE REGISTRE OPTION $1039 :

OPTION	ADPU	CSEL	-	-	-	-	-	-
Au reset	0	0	-	-	-	-	-	-

Tab.III.27- Etat du registre OPTION au RESET.

CSEL (clock select) : choix de l'horloge
Ce bit permet de faire le choix de la source d'horloge
CSEL=0 l'horloge E est utilisé ;
CSEL=1 une source interne d'horloge est utilisée ;

III.11.4.2- REGISTRE DE CONTROLE ADCTL($1030) :

ADCTL	CCF	0	SCAN	MULT	CD	CC	CB	CA
Au reset	0	0	-	-	-	-	-	-

Tab.III.28- Etat du registre ADCTL au RESET.

Les bits 0 à3 (CA à CD) servent à sélectionner grâce à un multiplexeur analogique l'origine des signaux à mesurer.
MULT : choix des canaux.
SCAN : permet de définir la succession de conversions.
CCF : (conversion complete flag) : conversion terminée.
L'indicateur de fin de conversion indique que les 4 conversions ont eu lieu et que les résultats dans les registres sont valides. Si SCAN est à 1 les registres sont mis à jour même si CCF est à 1. Pour le mettre à 0 il suffit d'écrire dans ADCTL.

III.12- AUTRES :

il existe d'autres registre et d'autre interfaces qu'on n'a pas cité dans ce chapitre tels que :

- L'interface de communication série synchrone (SPI : SERIAL PERIPHERAL INTERFACE).
- le timer programmable : utilisés pour mesurer la période d'un signal (fréquences) ou la durée d'une impulsion, présenté sur le port A, comme on peut l'utiliser pour générer des signaux rectangulaires ou carrés en sortie du port A.

Cela peut être justifié par la non utilisation de ces interfaces dans notre application.

CHAPITRE IV :
DEVELOPPEMENT D'UNE APPLICATION A BASE DU MICRO-CONTROLEUR

IV.1- INTRODUCTION :

Le développement d'une application à base de micro contrôleur, quels que soient sont type et sa marque d'ailleurs, diffère quelque peu de celui d'une application à base du micro contrôleur à vocation plus généraliste.

En effet, compte tenu de l'interpénétration très étroite entre matériel et logiciel qu'on ne le rencontre que dans un micro contrôleur, il est indispensable d'appréhender les deux aspects du problème avant de décider de quoi que ce soit.

IV.2-CHOIX DU MICRO-CONTROLEUR : [22]

Ce choix est évidemment dicté en tout premier lieu par l'adaptation de son architecture interne aux besoins de notre application ; ce choix est aussi conditionné par le fait qu'on possède ou non un système de développement et qu'on décide de travailler avec des moyens industriels ou avec des moyens minima.

En effet, mieux vaut parfois de sélectionner un micro contrôleur ne contenant pas exactement tous les éléments dont on a besoin, quitte à lui ajouter un ou deux boîtiers externes, mais pour lequel on dispose de tous les moyens de développement nécessaires : plutôt qu'un super micro contrôleur pour lequel aucun outil de mise au point n'est disponible ou pour lequel investissement nécessaire pour posséder un outil est hors de proportion compte tenu de l'application envisagée.

IV.3-NOTION DE SYSTEME DE DEVELOPPEMENT : [22]

IV.3.1-LANGAGE MACHINE ET LANGAGE EVOLUE :

Le développement de tout programme informatique peut être fait en langage machine improprement appelé « assembleur » ou en langage évolué. En ce qui concerne les microcontrôleurs que sont les 68HC11,le langage machine est de loin la meilleure solution même si elle semble plus lourde à mettre en œuvre.

Si le programme est écrit en langage machine, il peut déjà réaliser des fonctions interdisantes. Si par contre, il est écrit en langage évolué, seules quelques centaines d'instructions du langage évolué vont trouver place dans la mémoire. En effet le propre d'un langage que qu'il soit, est de présenter un taux d'expansion, certes variable selon que l'on travaille en C, en Basic ou en pascal, mais toujours important. Ainsi, une instruction élémentaire en langage évolué peut-elle générer après compilation plusieurs centaines de lignes de langage machine.

IV.3.2-LES CONSTITUANTS DE BASE D'UN SYSTEME DE DEVELOPPEMENT :

Un système de développement type comporte en premier lieu un éditeur ou un éditeur de texte dont le type exact importe peu. Il permet en effet seulement d'écrire avec un maximum de confort le programme, qu'il soit en langage machine ou en langage évolué.

Le deuxième programme important est évidemment l'assembleur si l'on travaille en langage machine ou le compilateur au langage évolué que l'on souhaite utiliser pour programmer si l'on décide, malgré l'avertissement vu précédemment de travailler de la sorte.

L'assembleur traduit les instructions en utilisant les mnémoniques du langage machine en code binaire exécutable par le microcontrôleur. La suite des mnémoniques s'appelle le listing ou code source du programme alors que le code binaire s'appelle l'objet ou l'exécutable.

Le compilateur, quant à lui, traduit les instructions écrites en langage évolué qui constituent aussi ce que l'on appelle le listing ou code source, en code binaire exécutable par le microcontrôleur.

Assembleur ou compilateur doivent nécessairement « tourner » sur un micro-ordinateur appelé machine hôte. Cette machine peut être n'importe quoi : systèmes spécifique du fabriquant du microcontrôleur, minicalculateur de forte puissance(VAX, station de travaille SUN, Apollo ou HP...) ou encore, et c'est une solution en passe de se généraliser, sur compatible PC. Cette dernière approche permet de minimiser l'investissement de départ surtout lorsque on possède déjà d'un PC utilisé ailleurs.

Une fois le programme de l'application écrit et assemblé ou compilé sur la machine hôte, on est en possession d'un fichier binaire exécutable. Sauf à être particulièrement inconscient, il est évident que l'on peut lancer la fabrication de plusieurs milliers de pièces d'une version du circuit programmé par masque par exemple sans plus de vérification. Il est donc quasiment indispensable de contrôler ce programme en le faisant fonctionner dans des conditions aussi proches que possible de son utilisation réelle future. Pour ce faire plusieurs solutions existent et nous allons les développer ici rapidement.

IV.3.3-EMULATEURS, SIMULATEURS ET MAQUETTES DE TEST :

La première solution pour valider une application, qui est de loin la plus performante et quasiment la seule concevable industriellement est aussi, hélas, la plus coûteuse à mettre en œuvre. Elle consiste à utiliser un émulateur comme schématisé sur la figure suivante :

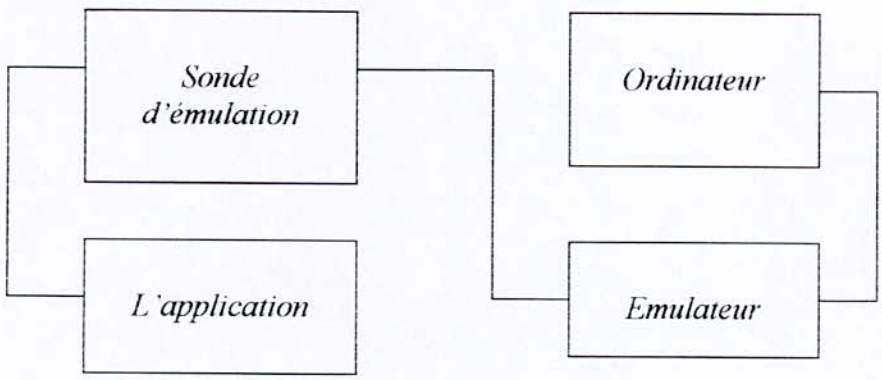


Fig.IV.1- Mise en œuvre d'un émulateur.

Cet émulateur, qui le cœur de ce système, est en effet un montage particulier, qui peut être très complexe, et qui se comporte exactement comme le microcontrôleur qu'il remplace. Mais c'est une version « éclatée » de ce dernier, c'est- à –dire en fait une version dans laquelle toutes les conditions internes du microcontrôleur ont été séparés. Cet émulateur est

muni d'un cordon de connexion spécial, appelé la sonde d'émulateur, à l'extrémité duquel est monté un connecteur analogue au boîtier du microcontrôleur. Ce connecteur vient donc se brancher sur la maquette de l'application en lieu et place du vrai microcontrôleur.

Par ailleurs, l'émulateur est relié généralement par une liaison série RS232, au système de développement de façon à pouvoir être téléchargé par le logiciel de l'application. En outre, c'est souvent également ce système de développement qui pilote l'émulateur grâce à un logiciel aussi convivial que possible.

Comme l'émulateur est une version éclatée du microcontrôleur qu'il remplace, on a accès aux divers signaux internes de celui-ci et, en particulier, on peut savoir à quelles adresses passe le programme, ce qui se passe au niveau des divers registres des périphériques internes.

En outre, comme l'émulateur est connecté sur l'application, il est possible de faire fonctionner celle-ci en temps réel. En effet, malgré l'éclatement du microcontrôleur qu'il réalise, le principe même de tout bon émulateur est de fonctionner aussi vite que le microcontrôleur qu'il remplace.

Tous les contrôles nécessaires peuvent donc être réalisés sur le programme comme s'il était réellement mis dans la ROM du microcontrôleur de l 'application.

La deuxième solution reste relativement confortable à l'usage mais peut donner des résultats très variables selon le type d'application à développer. Elle consiste à faire appel à un simulateur. Ce simulateur est un programme, écrit spécialement pour le microcontrôleur qu'il est sensé simuler. Il fonctionne généralement sur la même machine que celle sur laquelle on a fait la compilation ou l'assemblage. on lui fournit en entrée le code objet à exécuter et il se comporte alors comme se comporterait le microcontrôleur qu'il simule.

Comme un microcontrôleur comporte un nombre non négligeable d'entrées/sorties et que cela ne peut être le cas de la machine sur laquelle tourne le simulateur ; celui-ci simule les entrées/sorties sous forme de mémoires particulières. Ainsi, si l'on dispose d'un port parallèle 8 bits, il va être représenté par un octet mémoire particulier. Il suffira donc d'aller le lire pour savoir à tout instant de l'exécution du programme dans quel état sont les lignes de sortie. l'écriture en mémoire dans cet octet quant à elle simulera une entrée de données.

Il est évident que cette phase de simulation des entrées/sorties est un peu longue et lourde mais que, si elle est bien conduite, elle permet de vérifier 80% des fonctionnalités d'un programme. Ou la situation se corse, c'est quand les entrées/sorties sont un peu particulières et font intervenir des notions de temps. En effet, le simulateur n'est jamais qu'un programme qui tourne sur une machine pour reproduire le fonctionnement de l'unité centrale du microcontrôleur. Même si les éléments utilisés sont performants, le simulateur tourne à peu près de 10 à 100 fois moins vite que ne tournera le même programme directement exécuté par le microcontrôleur. Un certain nombre d'opérations faisant intervenir des notions de temps précises ou critiques ne sont donc pas accessibles à la simulation.

La troisième solution consiste à faire appel à une maquette de l'application sur laquelle on va monter une version UVPROM du microcontrôleur. Toutes les erreurs seront permises grâce à la possibilité d'effacement du circuit. La famille 68HC11 se prête toutefois assez mal à ce genre de « sport », tout au moins directement, car les versions de circuits équipées de mémoire UVPROM sont assez peu nombreuses.

Une solution intermédiaire existe toutefois si l'application n'est pas très gourmande en entrées/ sorties parallèles. Elle consiste à faire la mise au point du programme en utilisant le 68HC11 en mode étendu ; son programme étant alors placé dans une UVPROM externe. On est alors ramené au cas précédent puisque cette mémoire peut être effacée souvent que nécessaire.

Que l'on utilise un microcontrôleur avec UVPROM intégré ou un montage avec UVPROM externe, cette approche est plus longue que celle faisant appel à l'émulateur ou au simulateur puisque en évidence des erreurs peut parfois prendre du temps ce qui nécessite l'utilisations des instructions placées spécialement dans le programme pour ce faire, mais il est parfaitement possible de développer des applications de la sorte.

IV.4-LE DEVELOPPEMENT LOGICIEL AU NIVEAU INDUSTRIEL : [22]

Pour être bien conduit, le développement d'une application à base d'un microcontrôleur en environnement industriel doit se dérouler de la façon suivante :

Il faut tout d'abord définir les choix matériels, de façon aussi précise que possible car cela influe directement sur le volume de logiciel à développer. Certains de ces choix vont être faciles à faire, d'autres vont être plus délicats car faisant intervenir tout à la fois les concepteurs matériel et logiciel.

Une fois ces choix déterminés, on doit bâtir l'organigramme du logiciel. Cette partie du travail peut amener à des modifications matériels qui doivent encore être rendues possible à ce niveau.

Ensuite, il importe de choisir un langage ou des langages de travail. On peut utiliser un langage évolué ou un langage machine.

Lorsque ce choix est fait on passe à l'écriture du programme proprement dit. Pour faciliter le test du logiciel, il est prudent de scinder celui-ci en un certain nombre de modules aux fonctions parfaitement bien définies. Ces modules seront alors écrits les uns après les autres et testés les uns après les autres. Ceci est particulièrement vrai pour les ressources internes. Chaque type de ressource doit être piloté par un module logiciel qui lui est propre.

Une fois le logiciel développé, il faut passer à la phase de test de celui-ci sur l'application. Selon le cas on fera appel à un simulateur ou à un émulateur comme expliqué précédemment.

Lorsque la mise au point du logiciel est terminée, il faut ensuite réaliser une disquette une bande perforée ou une mémoire programmée avec celui-ci et la fournir au fabriquant du microcontrôleur selon des modalités définies par ses soins de façon à ce qu'il puisse en faire la programmation par masque.

En principe le fabriquant fournit, dans un premier temps, un petit lot de circuits programmes. Il importe d'essayer ceux ci très sérieusement sur l'application pour vérifier que le microcontrôleur se comporte bien comme prévu. En effet c'est à la suite du test de ces quelques pièces qu'on va devoir donner le feu vert au fabricant pour la production de série. Il est donc fondamental de ne pas donner la réponse à la légère.

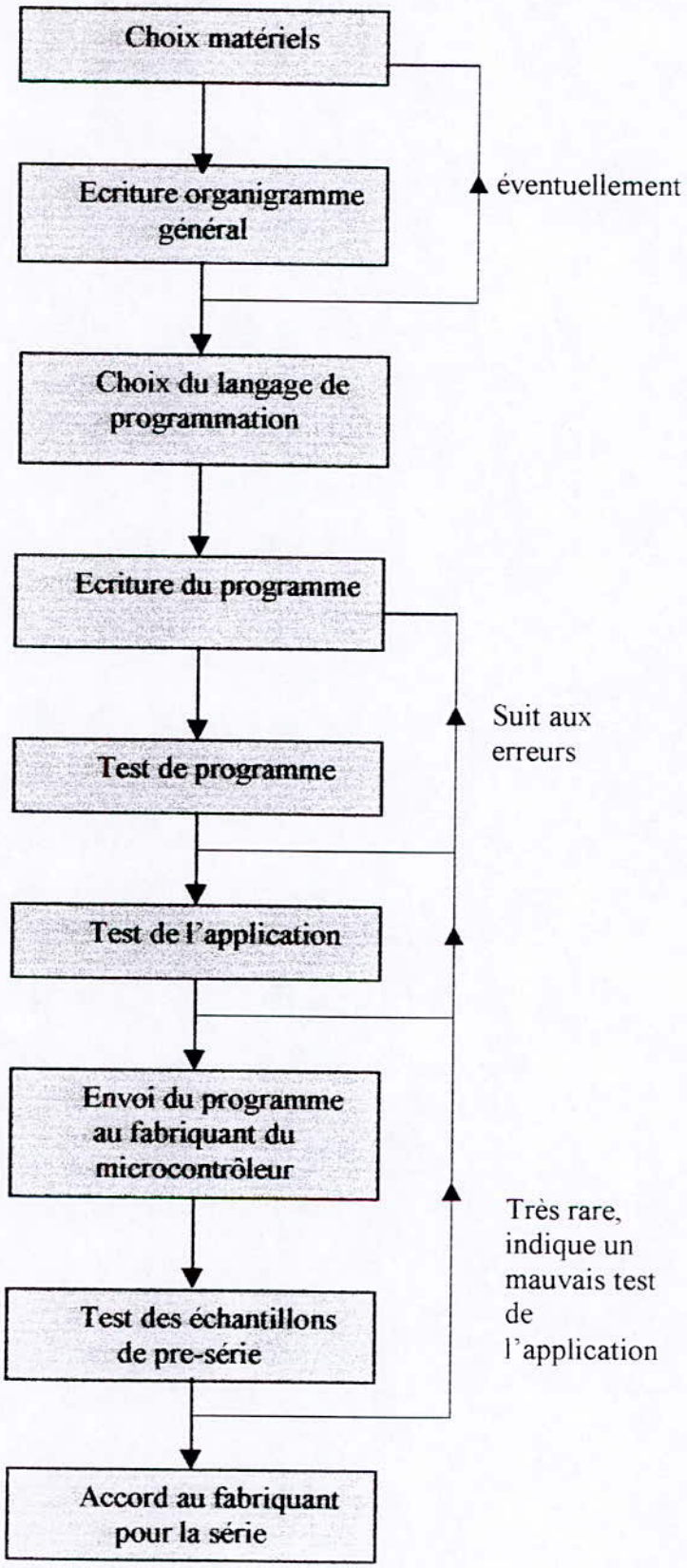


Fig.IV.2- Organigramme de développement complet d'une application au plan logiciel.

IV.5- DEVELOPPEMENT LOGICIEL AVEC DES MOYENS MINIMAUX : [22]

La première condition à respecter pour développer une application avec des moyens minima est de choisir un microcontrôleur dont une version avec UVPROM ou OPTROM existe. En effet c'est cette version qui va nous servir de moyen de test et remplacer simulateur et émulateur.

Il est tout aussi indispensable de posséder un assembleur pour le microcontrôleur considéré.

Les étapes du développement sont analogues à celles vues dans le paragraphe précédent, mais il faut s'arrêter au stade du début du test logiciel qui va devoir être conduit de façon différente. En effet, comme nous ne disposons ni de simulateur, ni d'émulateur, la seule solution va être de programmer le microcontrôleur à UVPROM, de le mettre sur l'application et de regarder ce qui ce passe. Il faut, lors de l'écriture du logiciel, prévoir un certain nombre d'instructions inutiles qui vont avoir seulement pour but de nous indiquer si notre programme les a exécutées ou non. Ces instructions non utiles doivent disposer d'un moyen de signalisation. Pour ce faire l'idéal est de disposer de quelques lignes d'entrées/sorties parallèles de trop. On connecte alors des LED sur ces dernières et chaque instruction inutile va allumer ou éteindre une combinaison de LED permettant ainsi de suivre, même grossièrement, le déroulement du programme depuis l'extérieur.

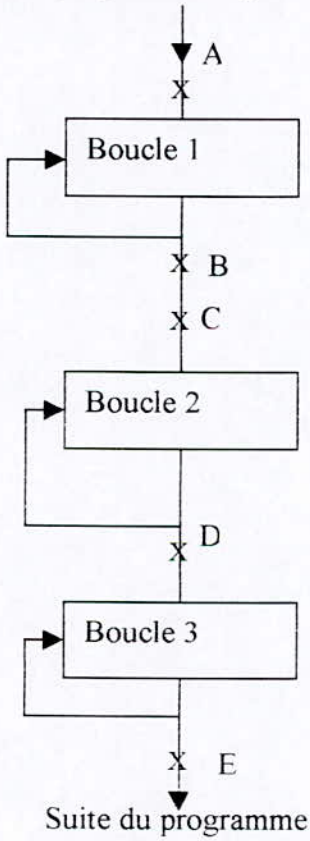


Fig.IV.3- Exemple de mise au point de programme avec des instructions « inutiles ».

Note : Les points A, B, C, D et E indiquent l'emplacement de points de test (instructions inutile).

CHAPITRE V :

PRESENTATION DE LA REALISATION

V.1-COTE PC : [7] [24] [25] [27] [28]

Selon notre besoin qui se résume à établir une communication série entre le PC et la carte à base du micro-contrôleur 68HC11, nous allons passer par une étude de communication au niveau PC.

V.1.1- TECHNIQUES DE COMMUNICATION SERIE AU NIVEAU DOS ET BIOS :

Le DOS et le BIOS fournissent un certain nombres de fonctions internes qui peuvent être appelées par programme ; plusieurs d'entre elles concernent la communication série.

La communication série peut avoir recours à deux fonctions DOS et quatre fonctions BIOS. Elles sont appelées par l'intermédiaire d'interruptions programmées.

V.1.1.1- FONCTIONS DOS :

La première fonction DOS de communication sert aux entrées séries ; celles ci sont réalisées en mettant le registre AH à 3 et exécutant une instruction int 21H. Le DOS attend de recevoir un caractère de COM1 et le retourne dans AL.

L'autre fonction DOS concerne les sorties séries ; elles sont réalisées en mettant AH à 4 et en plaçant le caractère à émettre dans DL et en exécutant int 21H le caractère sera envoyé à COM1.

Il n'y a pas de moyen d'initialiser les paramètres de communications au moyen des fonctions DOS, il faut que le programmeur utilise les fonctions BIOS ou bien que l'utilisateur initialise les paramètres à l'aide de la commande mode du DOS. Le DOS ne retourne aucune information sur l'état de la ligne ou du modem.

V.1.1.2-FONCTIONS BIOS :

Pour faire appelle aux quatre fonctions du BIOS concernant la communication série, on doit accéder à l'interruption 14H , adresse 050-053 dans la table des vecteurs d'interruptions.

Il faut placer un nombre de 0à 3 dans AH pour indiquer laquelle des quatre fonctions on désire appeler ; puis on indique ensuite un numéro de port dans DX, 0 pour COM1 et 1 pour COM2.

1. AH=00 : initialisation du port série.

   Entrée :
   - AL : paramètres d'initialisation.
   - DX : numéro du port (0 ou 1).

   Sortie :
   - AL : état du modem.
   - AH : état de la ligne.

2. AH=01 : émission d'un caractère.

   Entrée :
   - AL : caractère à envoyer.
   - DX : numéro du port (0 ou 1).

   Sortie :
   - bit 7 du AH est à 1 si le caractère n'a pu être envoyé.

3. AH=02: réception d'un caractère.
   Entrée :
   DX : numéro du port (0 ou 1).
   Sortie :
   AL : caractère reçu.
   AH : 0 si le caractère est correctement reçu.

4. AH=03 : lecture de l'état de la communication.
   Entrée :
   DX : numéro du port (0 ou 1).
   Sortie :
   AL : état du modem.
   AH : état de la ligne.

V.1.2- MODE D'EXPLOITATION DE L'UART :

Avant de récupérer la donnée reçue du registre tampon, il faut s'informer de sa présence. Nous citons deux modes différents pour cet effet :

- **Méthode de polling :**

Si l'on désire programmer le 8250 par la méthode de polling, on débute par initialiser à zéro le registre de validation des interruptions.

Si on est en train d'émettre, la routine d'émission doit tourner dans une boucle tant qu'il reste des caractères à envoyer ; on lit à chaque passage le registre d'état de lignes (LSR) et le registre d'état de modem (MSR). Lorsque le registre tampon est vide et que les signaux de contrôle sont à l'état haut, il envoie le nouveau caractère.

La routine de réception est analogue à la routine d'émission. On doit d'abord initialiser les signaux de contrôle exigés par l'autre appareil.

- **Méthode d'interruption :**

Beaucoup d'ordinateurs ont ce qu'on appelle un service d'interruption. Sur réception d'un signal le processeur termine l'instruction courante et passe à un module de gestion des interruptions (interrupt handler), situé à une adresse connue de la mémoire.

L'interface UART peut être programmée pour envoyer un signal d'interruption lors d'un événement précis à travers la sortie INTRPT (broche 30 du boîtier UART 8250). L'autorisation des interruptions, par le contrôleur, est accompagnée par précision de sa nature par celui-ci dans le registre IIR.

Un des avantages de ce système est de permettre un traitement systématique des événements aléatoire, sans que la partie principale du programme soit compliquée par des tests continuels de l'état des périphériques.

En général, avec une gestion des interruptions programmées, un programme principal tourne en même temps que le programme de gestion d'interruptions. Ceux-ci peuvent communiquer avec le programme principal en s'adressant des caractères, par l'intermédiaire des tampons de réception et d'émission en mémoire.

V.1.3-IMPLEMENTATION DE LA COMMUNICATION AVEC LE 8250 :

Exploitant les fonctions vues précédemment nous avons conçu, dans le programme de communication au niveau PC, les fonctions suivantes (en langage C++3.00) :

ser_UARType : détermine le type du chip UART.

Entrée :

iSerPort : adresse de base de l'interface à tester.

Sortie :

0 : chip UART introuvable.
1 : chip INS8250 ou INS8250-B chip.
2 : INS8250A, INS82C50A, NS16450, NS16C450.
3 : chip NS16550A.
4 : chip NS16C55.

ser_Init : initialise l'interface série.

Entrées :

iSerPort : adresse de base de l'interface à utiliser.

IBaud : vitesse de transmission (vont de 1-115200).

Sortie :

TRUE : initialisation de l'interface réussie.
FALSE : interface introuvable.

ser_IsDataAvaiable : les données sont-elles prêtes à la lecture ?

Entrée :

iSerPort : adresse de base de l'interface à utiliser.

Sortie :

== 0 : aucun octet n'existe pour la lecture.
!=0 : un octet est disponible.

ser_IsWratingPossible : l'interface peut-elle envoyer encore un octet ?

Entrée :

iSerPort : adresse de base de l'interface à utiliser .

Sortie :

== 0 : il ne faut pas envoyer d'octet.
!=0 : l'interface est prête à émettre.

Note : il ne faut pas utiliser une interface série pour envoyer des octets dans les cas suivants :

- Un octet reçu n'a pas encore été appelé par l'interface.
- Une demande d'émission n'a pas encore été exécutée.

ser_WriteByte : envoi d'un octet de donnée.

Entrées :

lSerPort : adresse de base de l'interface à utiliser.

bData : l'octet à envoyer.

uTimeOut : nombre de passages effectués dans la boucle avant que l'échec d'une émission ne soit signalé par une erreur timeout.

bSigMask : masque de bits des lignes de signal à tester (RTS, CTS, CD, RI).

bSigVals : état des lignes de signal après avoir démasqué les masque ci-dessus.

Sortie :

==0 : l'octet a été envoyé.

!=0 : erreurs d'émission.

ser_ReadByte : réception d'un octet de donnée.

Entrées :

iSerPort : adresse de base de l'interface à utiliser.

pData : la variable byte accueille l'octet reçu.

uTimeOut : nombre de passages effectués dans la boucle avant que l'échec d'une réception ne soit signalé par une erreur time-out.

bSigMask : masque de bits des lignes de signal à tester (RTS, CTS, CD, RI).

bSigVals : état des lignes de signal après avoir démasqué le masque ci-dessus.

Sortie :

==0 : l'octet a été reçu.

!=0 : erreurs de réception.

trans_WriteAck : envoi acknowledge positif par l'interface.

Entrée :

iSerPort : adresse de base de l'interface à tester.

Sortie :

Niveau d'erreur.

trans_RecievePaket : réception d'un paquet de données par l'interface.

Entrées :

iSerPort : adresse de base de l'interface à tester.

pPaket : adresse du buffer d'entrée.

iPaketSise : taille du buffer.

Sortie :

Niveau d'erreur.

trans_receiveFile : réception d'un fichier entier par l'interface.

Entrée :

iSerPort : adresse de base de l'interface à Utiliser .

Sortie :

Niveau d'erreur.

trans_pError : émission d'un message d'erreur.

Entrée :

iError : erreur survenue.

GetArg : retourne paramètre de la ligne de commande.

Entrées :

argc :compteur d'arguments de la ligne de commande.

Argv : valeur de l'argument de la ligne de commande.

pPrefix : paramètre préfixe.

iType : type des paramètres(_char, _int, _long, _string, _none= vérifié la présence d'un paramètre).

pVar : adresse de la variable qui va contenir les valeurs des paramètres de la ligne de commande.

iNumElements : s'il y a plusieurs valeurs séparés par des virgules ','enregistre les paramètres jusqu'à l'élément iNumEléments dans l'array indiqué dans le pVar.

Sortie :

Nombre de valeurs calculées ou 0 s'il n'y a pas de paramètres de commande.

GetNArg : retourne paramètre de la ligne de commande sans préfixe.

Entrées :

argc :compteur d'arguments de la ligne de commande.

Argv : valeur de l'argument de la ligne de commande.

pPrefix : préfixe des paramètres.

pString : adresse du pointeur de String arrays qui recueille les paramètres.

iNumElements : s'il y a plusieurs paramètres contenus dans la ligne de commande, tout au plus max paramètre sont préserver dans iNumEléments.

Sortie :

Nombre de paramètres communiquer ou 0 s'il n'y a pas de paramètres sur la ligne de commande.

FindString : recherche d'un string dans un string-array et envoi de la position de string qui a été trouvé dans l'array.

Entrées :

pArray :string-array à examiner.

pFind : string à rechercher.

uNum : nombre de strings dans le string-array.

Sortie :

Position + 1 du string trouver dans l'array ou 0 si le string n'est pas contenu dans l'array.

V.1.4- ORGANIGRAMME DU PROGRAMME PRINCIPAL :

Le programme lit en premier la ligne de commande et teste la présence de trois paramètres : le nom de fichier, le port à utiliser et la vitesse de transmission. S'il détecte la présence d'un nom de fichier sur la ligne de commande il se branche vers la routine de réception d'un bloc de données de 128 octets. S'il ne détecte pas de nom de fichier, il se branche vers la routine de réception de fichier.

V.1.4.1- ROUTINE DE RECEPTION DE BLOC DE DONNEES :

La routine teste en premier lieu si la réception du bloc de données est faite avec succès. Si oui elle ouvre le fichier indiqué dans la ligne de commande puis écrit les données dedans et enfin la ferme. Sinon la routine fait 10 boucles en attendant la présence de données.

V.1.4.2- ROUTINE DE RECEPTION DE FICHIER :

Le fichier à recevoir doit être décomposé en blocs, chaque bloc contient deux champs :champ de commande et champ de données. Le premier englobe des informations concernant le type, la taille et le rang du bloc.

- Si le bloc est de type **PAKET_FILEOPEN**, le deuxième champ représente le nom de fichier qui doit être créé parle programme.
- Si le bloc est de type **PAKET_FILEDATA**, le deuxième champ représente un bloc de données, de taille spécifiée dans le champ de commande, à stocker dans le fichier ouvert précédemment.
- Si le bloc est de type **PAKET_FILECLOSE**, le programme ferme le fichier en ignorant le deuxième champ.

A la réception de chaque bloc le programme lit le rang et teste son séquencement.

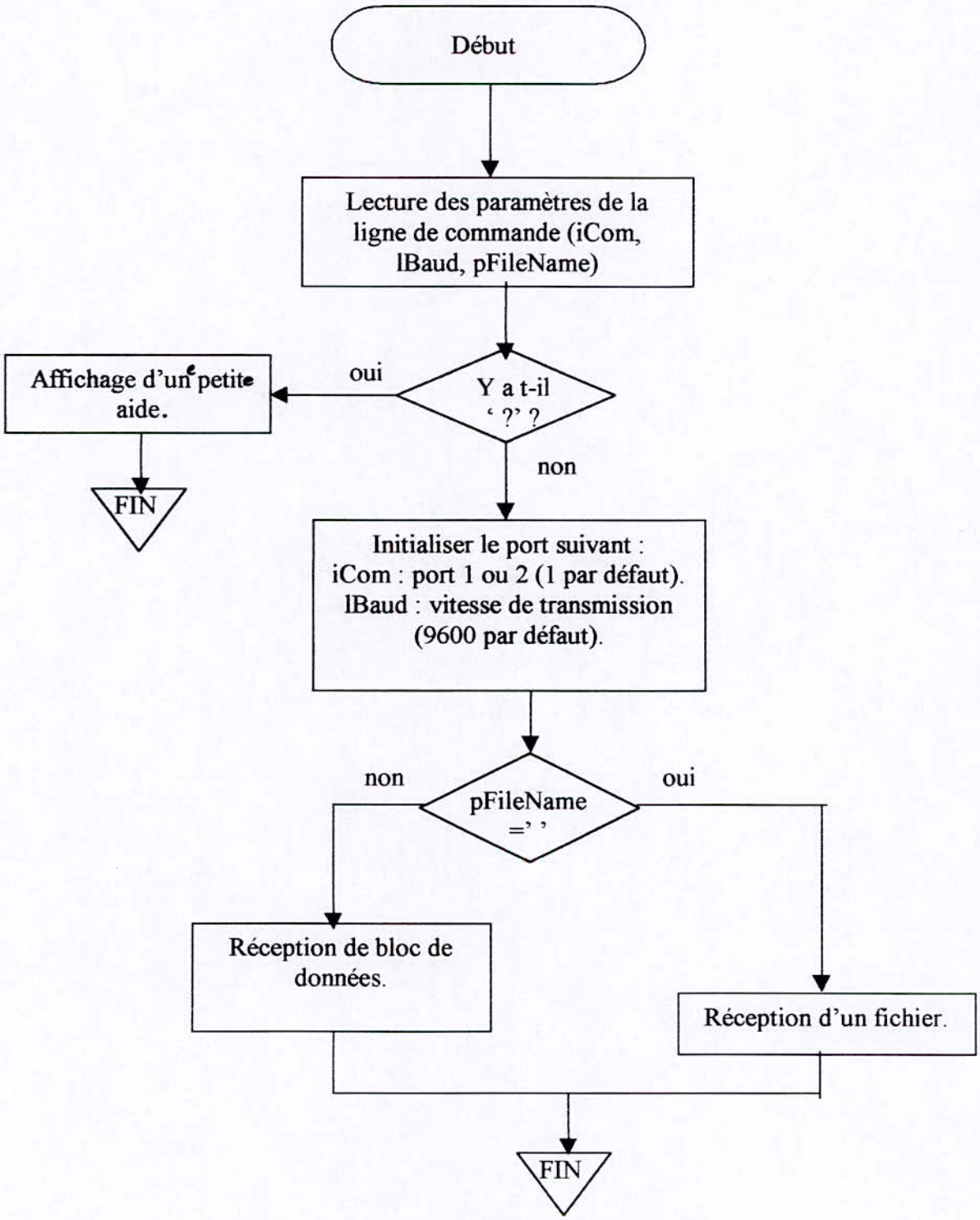


Fig.V.1- Organigramme du programme principal.

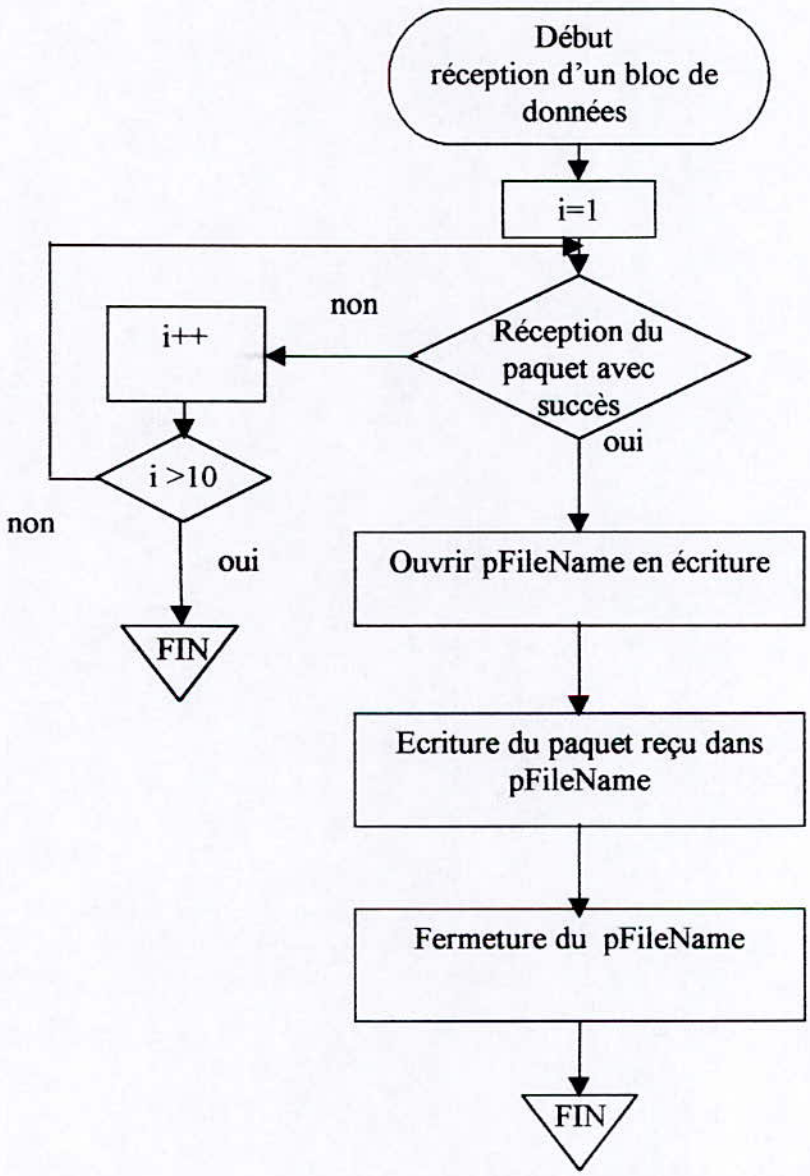
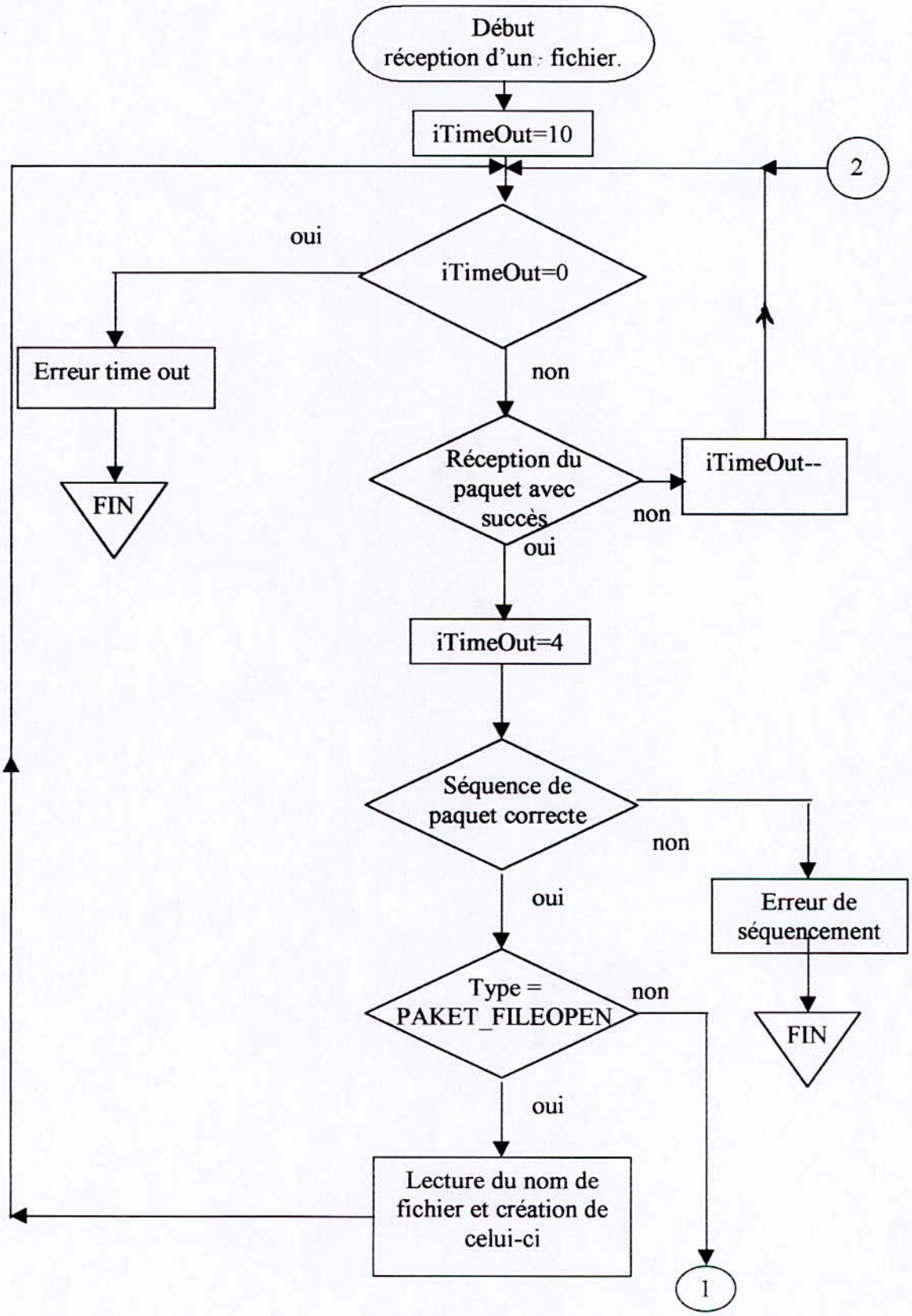


Fig.V.2- Organigramme de routine de réception d'un bloc de données.


Début
réception d'un fichier.
iTimeOut=10
2
oui
iTimeOut=0
Erreur time out
FIN
non
Réception du
paquet avec
succès
non
iTimeOut--
oui
iTimeOut=4
Séquence de
paquet correcte
non
Erreur de
séquencement
FIN
oui
Type =
PAKET_FILEOPEN
non
oui
Lecture du nom de
fichier et création de
celui-ci
1

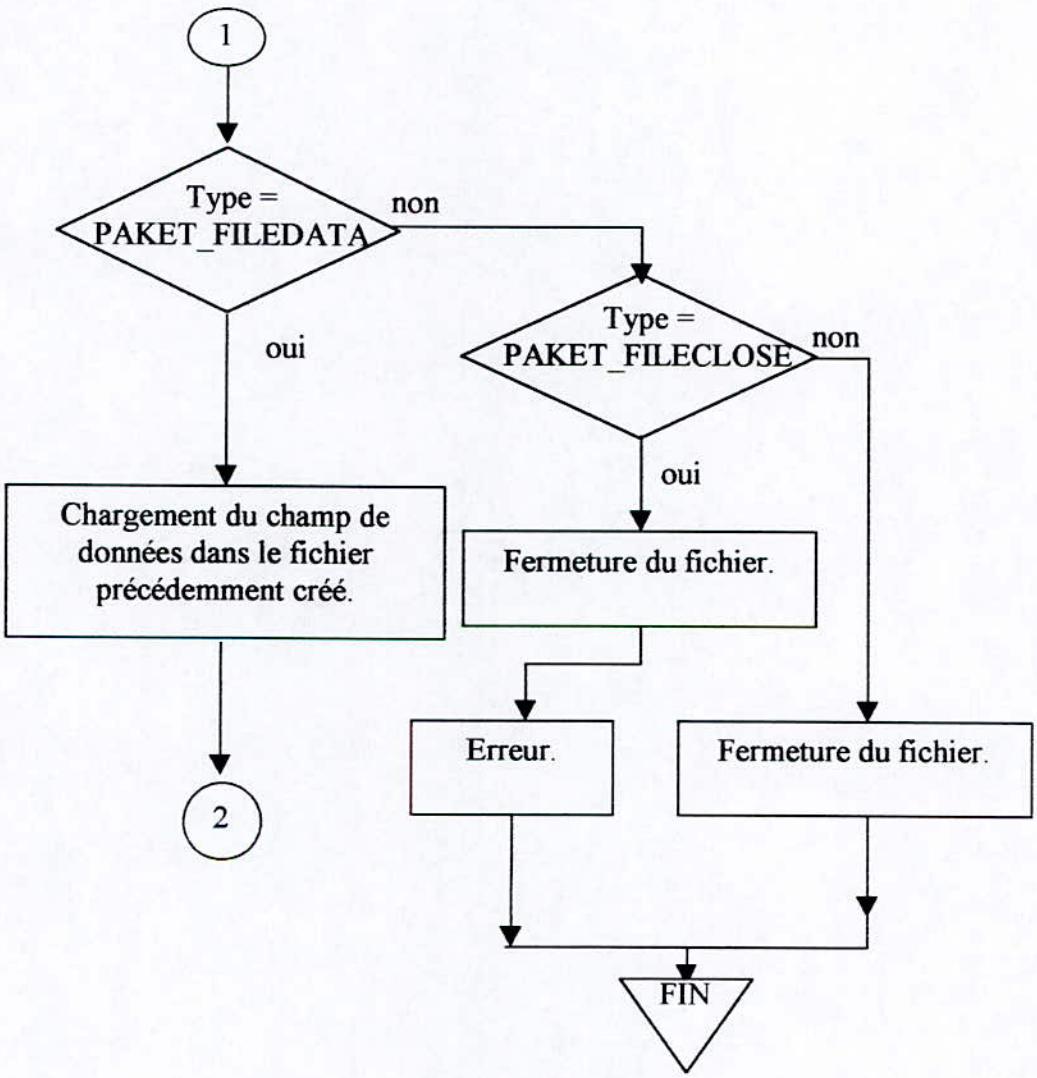
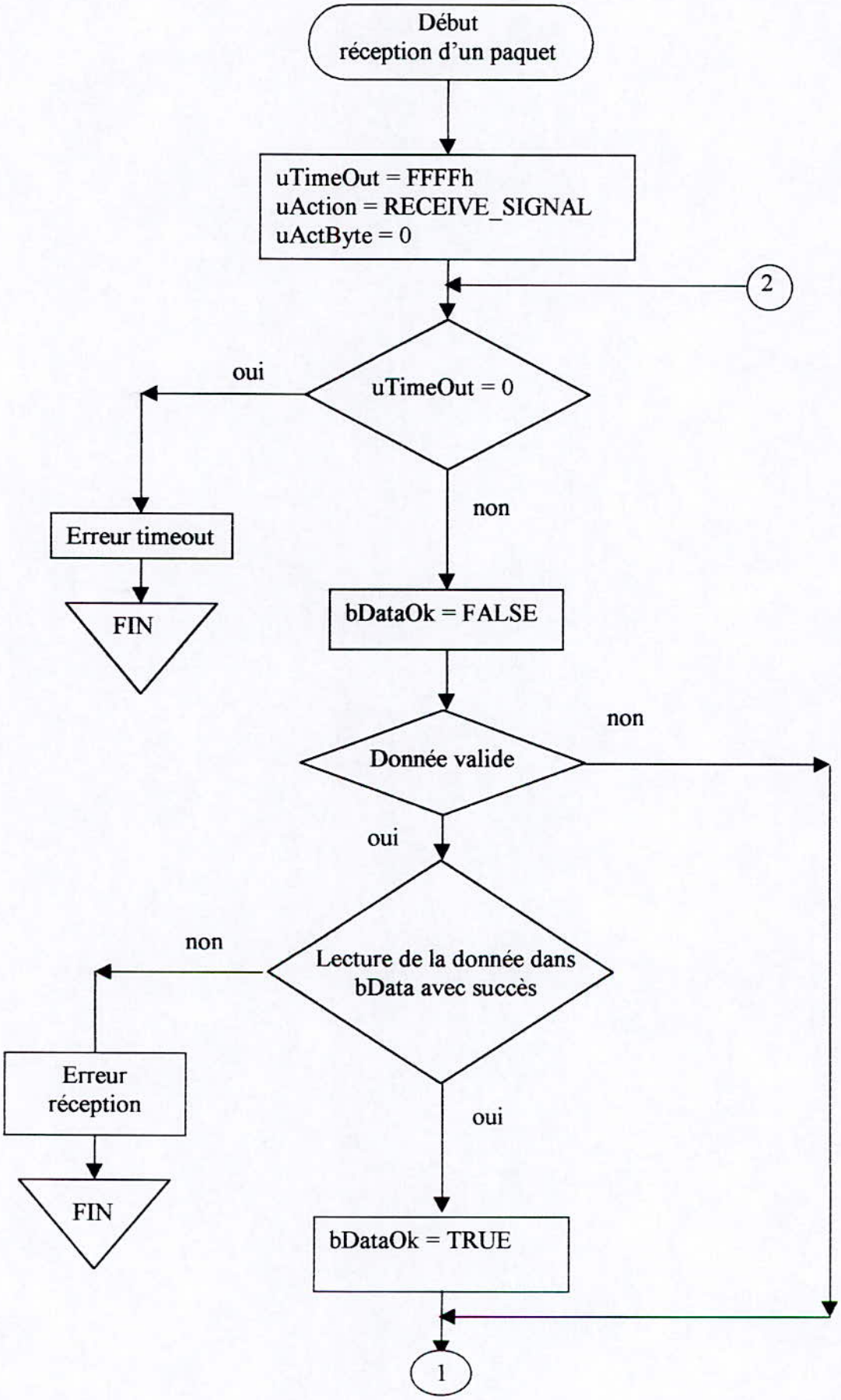
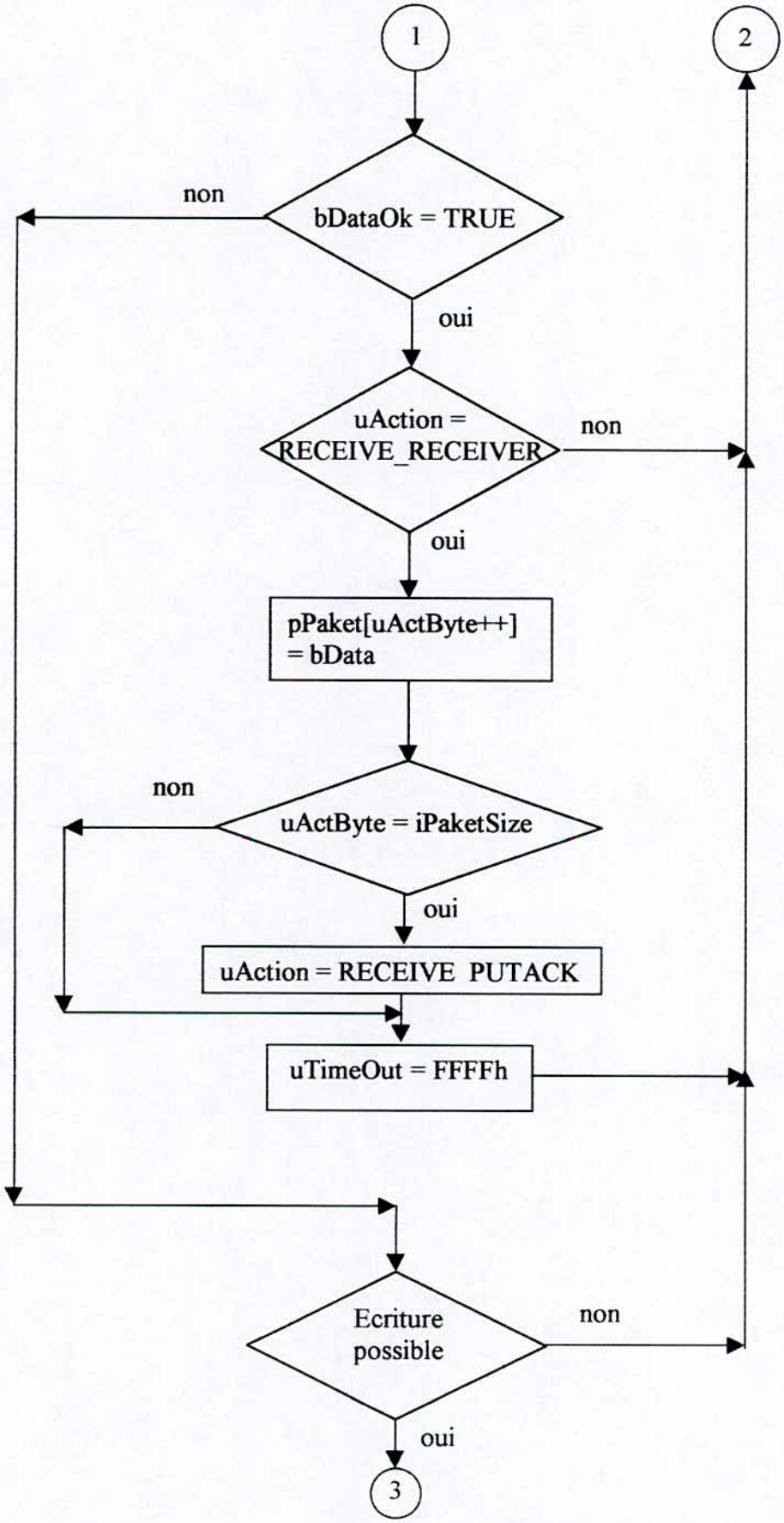


Fig.V.3- Organigramme de routine de réception d'une fichier.


Début
réception d'un paquet
uTimeOut = FFFFh
uAction = RECEIVE_SIGNAL
uActByte = 0
2
uTimeOut = 0
oui
non
Erreur timeout
FIN
bDataOk = FALSE
Donnée valide
non
oui
Lecture de la donnée dans
bData avec succès
non
oui
Erreur
réception
FIN
bDataOk = TRUE
1


1
2
bDataOk = TRUE
non
oui
uAction = RECEIVE_RECEIVER
non
oui
pPaket[uActByte++] = bData
uActByte = iPaketSize
non
oui
uAction = RECEIVE_PUTACK
uTimeOut = FFFFh
Ecriture possible
non
oui
3

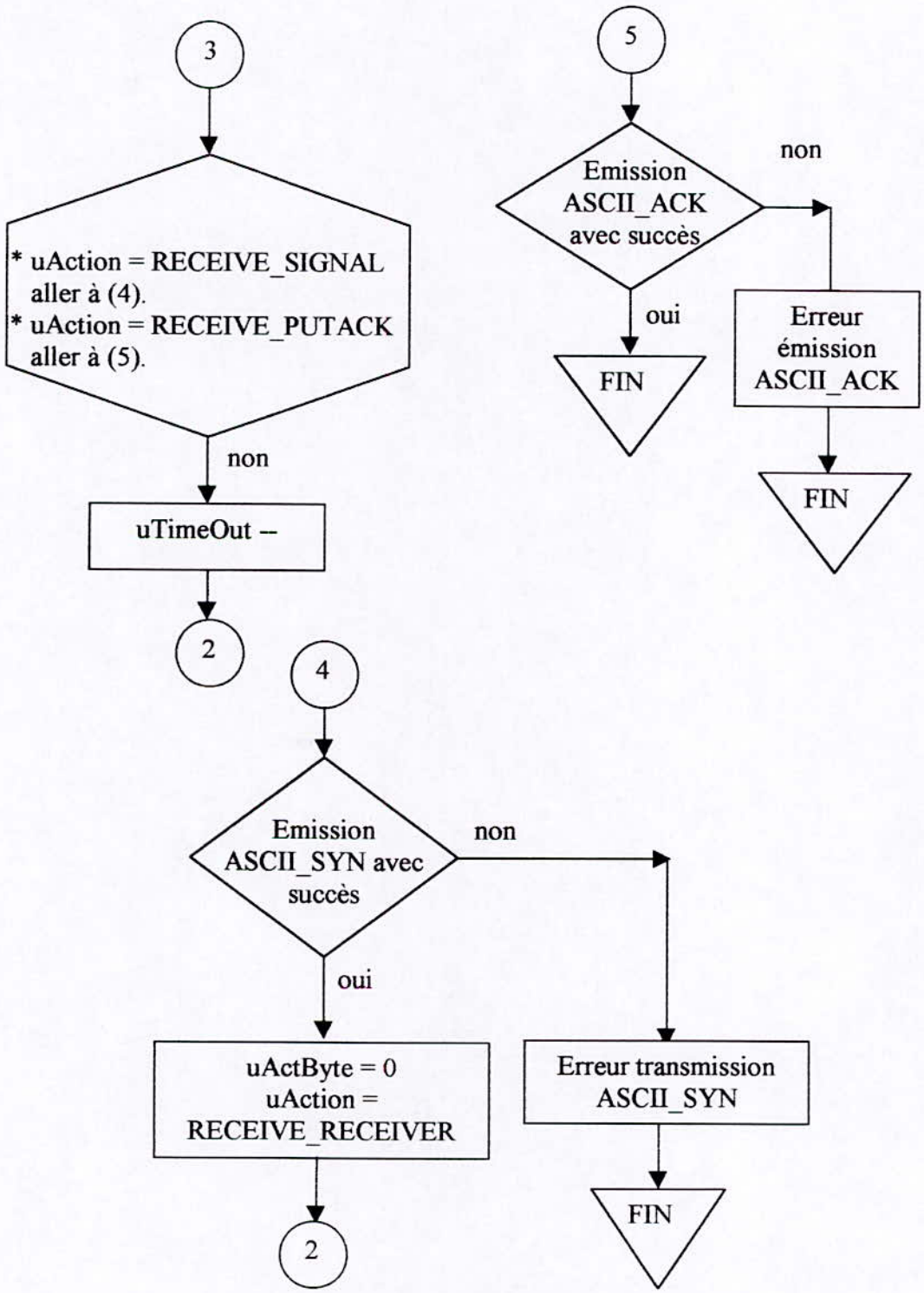


Fig.V.4- Organigramme de routine de réception d'un paquet de données.

V.2-CÔTE CARTE : [17] [18] [19] [20] [21] [22] [29 [30] [31]

V.2.1-INTRODUCTION :

Cette carte est destinée à travailler comme une interface entre un récepteur, fournissant soit un signal analogique soit une information numérique série, et le PC à travers le port série de celui-ci.

Avec la carte décrite ici, on peut utiliser le micro-contrôleur en mode bootstrap, en mode single chip (circuit seul), en mode spécial test ou en mode étendu en positionnant le micro-switch K4 à l'état convenable voir (Tab.III.2).

V.2.2-SCHEMA SYNOPTIQUE :

Notre carte contient les sous-ensembles cité ci-dessous (Fig.V.5.) :

- un connecteur DB25 dont le rôle est de permettre la liaison matérielle entre le PC et la carte,
- un convertisseur de niveau dont le rôle est de faire l'adaptation TTL – RS232,
- un micro-contrôleur 68HC11A1 qui est le pilote de notre application,

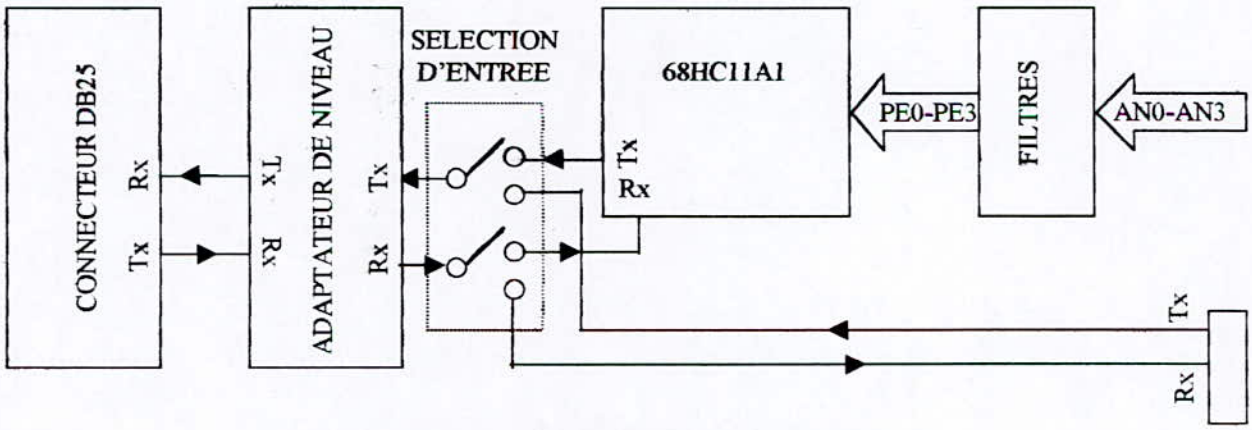


Fig. V.5– Schéma synoptique de la carte.

V.2.3-DETAIL DES SCHEMAS PARTIELS :

V.2.3.1- LES TENSIONS DE REFERENCE DU CONVERTISSEUR (VREFL, VREFH) :

Afin de conserver une bonne précision au convertisseur analogique numérique l'écart entre VREFL et VREFH doit être au moins de 2.5 volts. Par ailleurs ces tensions ne doivent pas devenir supérieures (pour VREFH) et inférieur (pour VREFL) aux alimentations du circuit

encore que le convertisseur continue de fonctionner correctement même pour VREFH=6 volts. Si l'alimentation de 5 volts du 68HC11 est correctement stabilisée, il est possible de mettre en œuvre le schéma de (Fig.V.2.) et simplifier ainsi la génération de VREFL et VREFH.

Le choix de R = 1kΩ et C = 1μF.

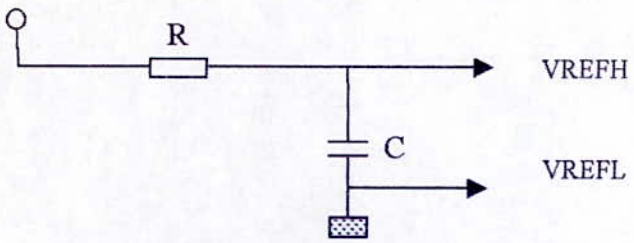


Fig.V.6- Circuit de stabilisation des tensions de référence.

V.2.3.2- LES ENTREES DU CONVERTISSEUR A/D :

PE0-PE7 sont les entrées du convertisseur A/D. Un filtre passe bas (Fig.V.7.) est indispensable pour permettre le filtrage d'éventuels bruits HF.

Le choix des valeurs de la résistance R et le condensateur C dépend de la fréquence minimale de la bande de bruit qu'on doit éliminer. Nous avons choisi R = 1 KΩ et C = 0.01μF ce qui permet le filtrage des fréquences au-delà de f_{MIN} = 1 / (R * C) = 100 KHz.

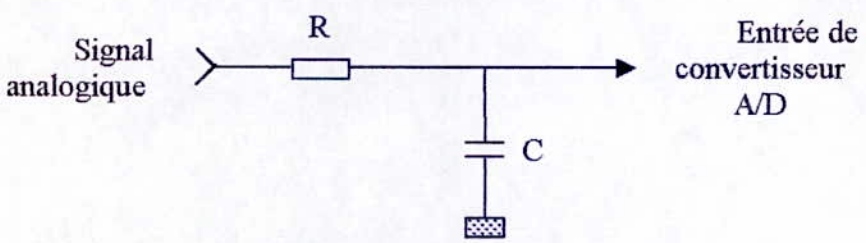


Fig.V.7- Circuit de filtrage des signaux analogiques.

V.2.3.3- CIRCUIT D'HORLOGE :

La (Fig.V.8.) montre le mode de connexion d'un quartz aux broches XTAL et EXTAL selon que sa fréquence est supérieure à 1 MHz ou inférieur à cette valeur. La fréquence du quartz est égale à 4 fois la fréquence réelle de l'horloge du bus interne du 68HC11. Il est possible d'utiliser la broche XTAL comme sortie, à condition de l'amplifier par un circuit à faible courant d'entrée. Il est également possible d'utiliser comme entrée d'horloge la broche EXTAL, pour synchroniser deux 68HC11 par exemple. La (Fig. V.9.) résume ces deux possibilités.

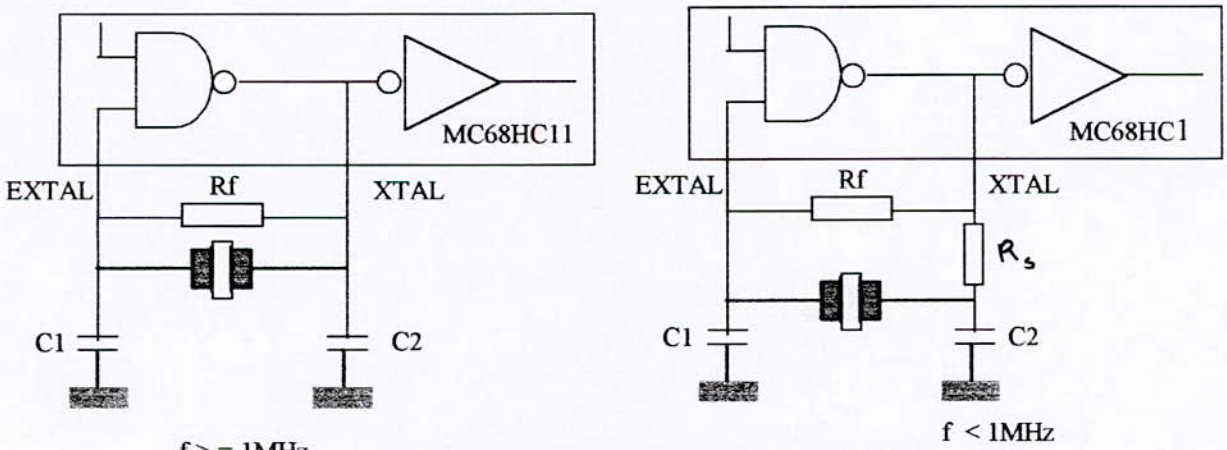


Fig. V.8– Connexion du Quartz d'horloge en fonction de sa fréquence.

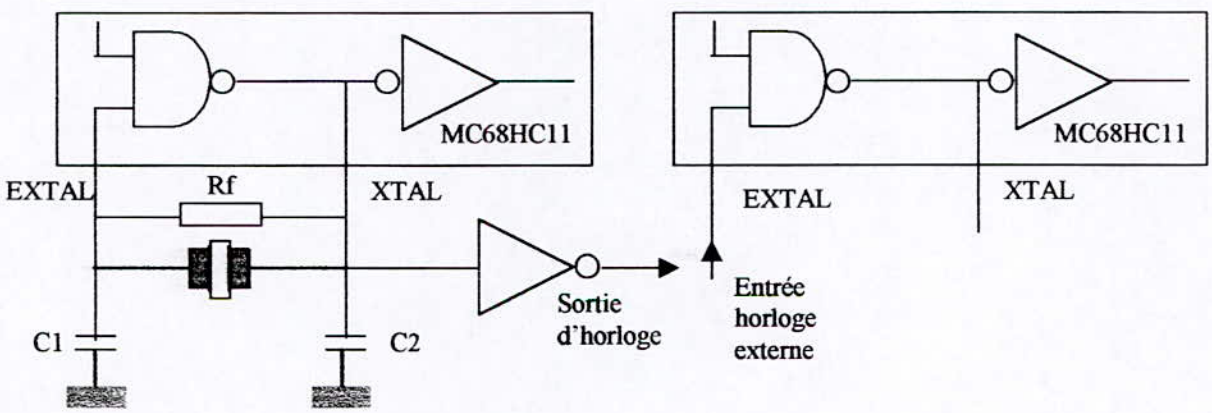


Fig. V.9- Comment réaliser une entrée ou une sortie d'horloge.

La valeur de la résistance Rf de la (Fig. V.8) dépend du type de quartz utilisé et de sa fréquence. Elle varie habituellement de 1 MΩ à 20 MΩ. Rs quant à elle varie de quelque centaine d'ohms à quelque Kohms.

Les condensateurs C1 et C2 dépendent eux aussi des caractéristiques précise du quartz mais aussi du type de 68HC11 utilisé, du dessin du circuit imprimé, etc. ils varient habituellement de 4.7 à 27 pF.

V.2.3.4- CIRCUIT DE RESET :

Afin d'éviter l'altération de contenu de l'EEPROM mais aussi de certains registres réalisés selon la même technologie, il est fortement conseillé de piloter l'entrée RESET du 68HC11 par un circuit détectant automatiquement toute baisse de la tension de l'alimentation en dessous de la valeur limite autorisée et générant alors un RESET dans ce cas.

La (Fig. V.10.) montre un exemple de ce type. Ce schéma permet deux possibilités : un RESET automatique (partie droite) et un RESET manuelle (parti gauche).
Dans notre application nous n'avons utilisé que le RESET manuel.

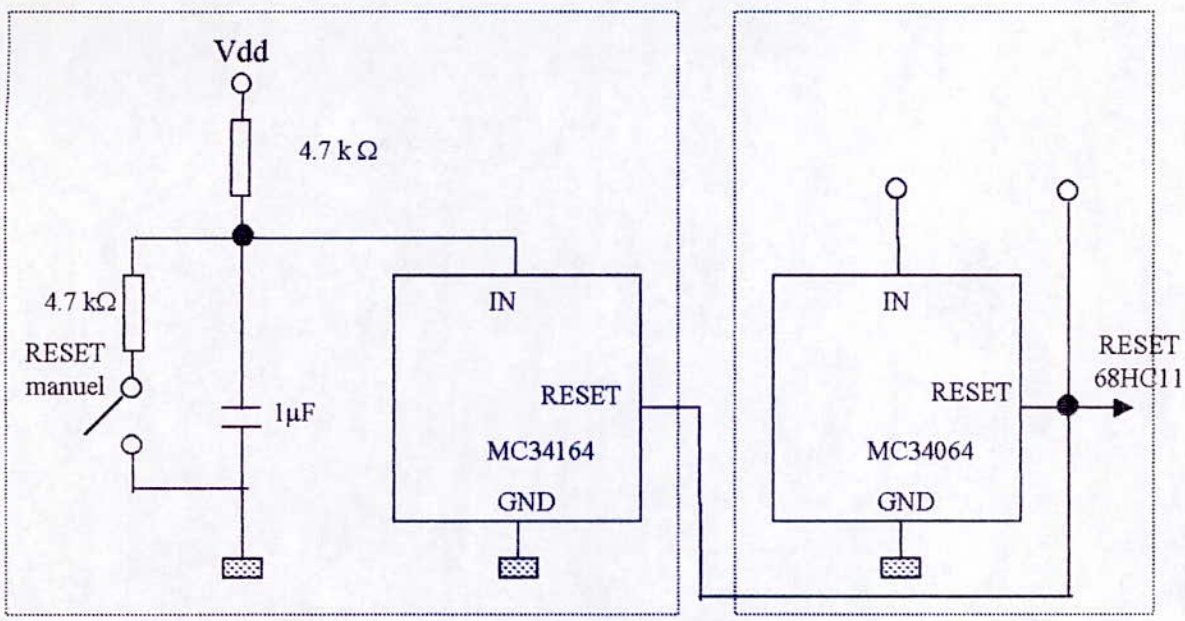


Fig. V.10- Le circuit de RESET automatique (cas de baisse de tension de l'alimentation) et manuel.

V.2.3.5- CIRCUIT D'ADAPTATION DE NIVEAU :

L'entrée Rx et la sortie Tx du micro-contrôleur présente des niveaux TTL mais la norme RS232 demande des niveaux de tension de ± 12 volts, c'est pour cela qu'on est besoin d'un convertisseur de niveau.

Afin de ne pas avoir besoin de traditionnelles alimentations symétriques ± 9 volts et ± 12 volts, il est logique de faire appel aux circuits spécialisés. Le plus connu est le MAX232 dont le schéma de mise en œuvre est présenté (Fig. V.11). Alimenté sous une tension unique de 5 volts, ce circuit est capable de recevoir et de produire de véritables niveaux RS232 grâce à des convertisseurs de tension statiques internes à capacités commutées. C'est la raison d'être des quatre condensateurs chimiques qui l'entourent.

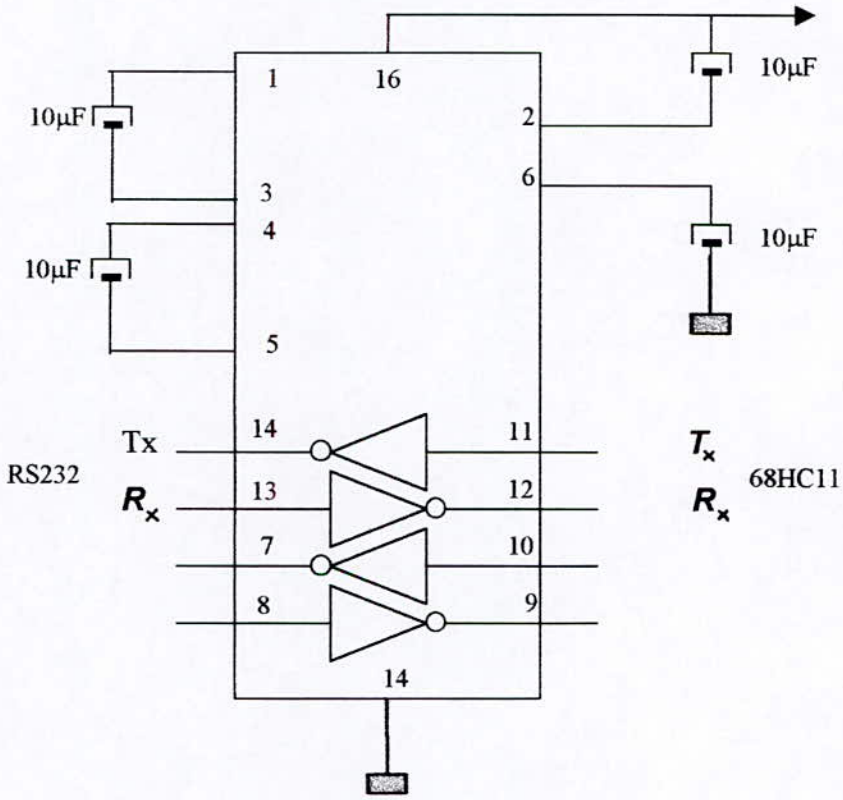


Fig. V.11- Schéma de mise en œuvre du MAX232.

V.2.4-DESCRIPTION DU SCHEMA GLOBAL :

Au cœur du schéma (Fig.V.12) se trouve le 68HC11A1. Quelques résistances de rappel (R9, R11, R13 - 26) sont connectées aux ports pouvant fonctionner en entrée pour ne pas perturber le fonctionnement du micro-contrôleur par d'éventuel bruit. Un quartz de 8 MHz permet le cadencement du circuit à la fréquence nominale. Rappelons qu'en interne cette fréquence se trouve divisée par quatre ce qui donne dans ce cas 2 MHz. Un cycle machine aura donc une durée T = 0,5 µs (valeur à retenir lorsque l'on doit calculer la fréquence d'échantillonnage).

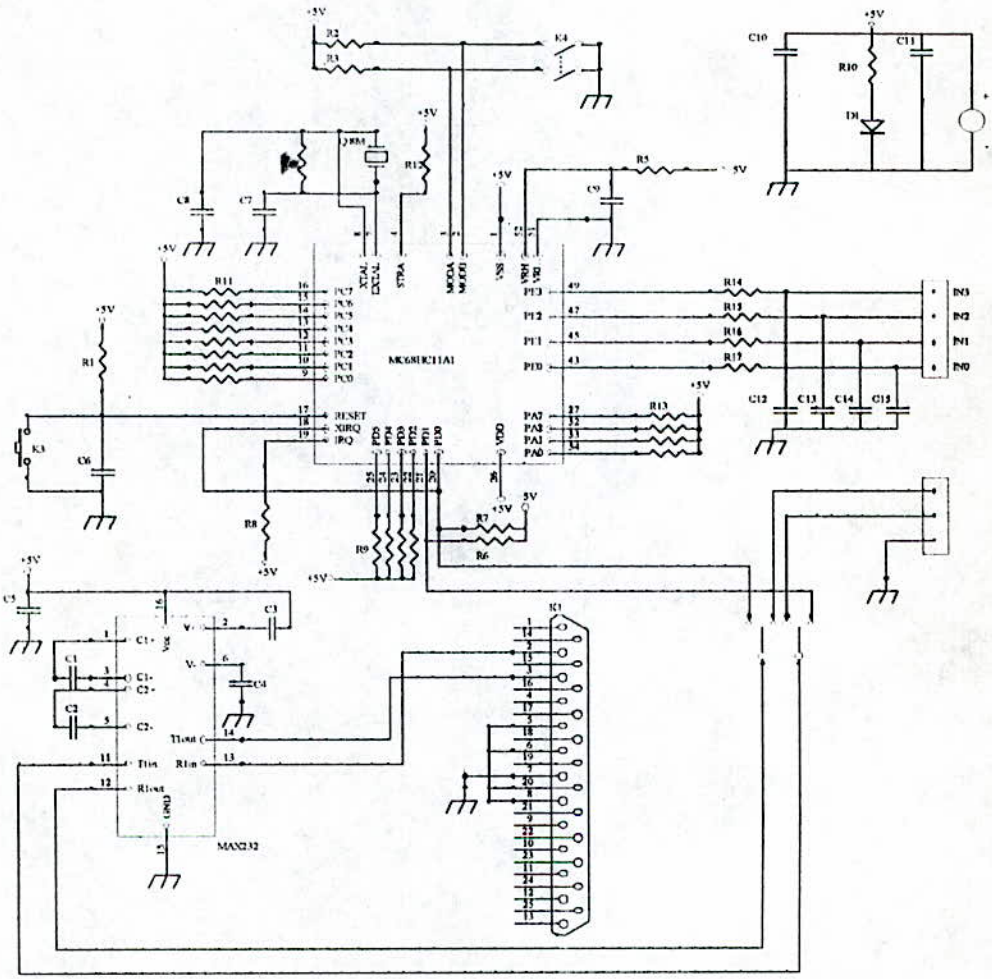


Fig V.12 – Schéma électrique de la carte.

Le montage doit être alimenté sous une tension continue de 5V. Une LED rouge permet de vérifier la présence de la tension sur la platine. L'interrupteur K2 sert à commander l'alimentation du micro-contrôleur : on le ferme en cas de sélection de l'entrée analogique et on l'ouvre en cas d'entrée de l'information série. Cette sélection se fait à l'aide du micro-switch K5. Notons que le MAX232 doit être alimenté en permanence.

A la mise sous tension, les niveaux présents sur les pattes MODA et MODB définissent le mode de fonctionnement du 68HC11. Pour notre application nous utilisons le micro-contrôleur en mode circuit seul (single chip) en phase de mise en œuvre et nous utilisons le mode bootstrap en phase de programmation de l'EEPROM interne. On a bien sûr prévu cette possibilité ici : le micro-switch K4 permet la sélection du mode convenable.

Chaque signal analogique issu de l'une des entrées K7 passe par un filtre passe bas (R27-C13, R28-C12, R29-C15, R30-C14) pour éliminer le bruit HF qui peut engendrer des mesures erronées.

La tension de référence haute Vrh du convertisseur est connecté à Vcc à travers un filtre passe bas pour assurer plus de stabilité.

Le circuit de RESET est réduit en minimum la résistance R1, le condensateur C6 et le bouton poussoir K3 qui permet un RESET manuel. Remarquons que cette configuration, bien que largement répandue, n'est pas celle qui est recommandé par MOTOROLA.

Enfin la communication avec le PC se fait par une liaison série par l'intermédiaire d'un circuit intégré convertisseur de niveau devenu bien classique : le MAX232.

V.2.5-LE LOGICIEL ET LE MATERIEL DE MISE AU POINT :

Pour la mise au point des programmes à télécharger dan l'EEPROM du micro-contrôleur nous avons utilisé les logiciels et les matériaux suivants :

- un éditeur de textes (EDIT du DOS) pour l'édition des fichiers sources,
- un assembleur AS11.EXE ou ASMHC11.EXE pour transformer le fichier source en fichier compréhensible par le 68HC11 (c'est le format S19),
- un logiciel de communication avec la carte distribuer gratuitement par MOTOROLA : PCBUG.
- un émulateur 68HC11EVB (68HC11 Evaluation Board).
- un programmateur d'EEPROM interne du micro-contrôleur 68HC11EVM (68HC11 Evaluation Module).
- un logiciel de communication entre le PC et le 68HC11EVB et/ou le 68HC11EVM : KERMIT.
- un logiciel d'édition des schémas : PFSCH (PROTEL SCHEMATEC).
- Un logiciel de routage de circuits imprimés : EAGLE2.

V.2.6-EMULATION :

V.2.6.1- SCHEMA SYNOPTIQUE DE L'EMULATEUR 68HC11EVB :

Le 68HC11EVB (Fig. V.13.)est un émulateur de 68HC11A1 ou 68HC811E2 qu'on a utilisé en phase de mise au point des programmes assembleurs.

Comme l'émulateur est une version éclatée du micro-contrôleur il contient, en plus de micro-contrôleur, les organes suivants :

une RAM,
une EEPROM,
une ACIA (adaptateur d'interface série),
une PRU (adaptateur d'interface parallèle),
deux connecteurs RS232,
une sonde d'émulation 52 pins.

V.2.6.2- TRAVAIL EFFECTUE AU NIVEAU DE L'EMULATEUR 68HC11EVB :

Avant de programmer l'EEPROM interne du 68HC11A1 il faut s'assurer de bon fonctionnement du programme par l'utilisation de l'émulateur. A cet effet, nous avons procédé comme suit :

- édition du programme en langage assembleur 6811 de MOTOROLA par l'éditeur de texte de DOS,
- conversion du programme, précédemment édité, en format S19 en utilisant le logiciel AS11.EXE,
- établissement de la liaison entre le PC et l'émulateur à l'aide de logiciel KERMIT.EXE,
- téléchargement du programme en format S19 de PC vers l'EEPROM du 68HC11EVB à travers KERMIT,
- vérification du contenu de l'EEPROM pour s'assurer que le programme est correctement téléchargé,
- lancement de l'exécution du programme et vérification de son bon fonctionnement en utilisant les outils disponibles a cet effet :
    - visualisation du signal issu des pins convenables de la sonde,
    - accès aux registres internes de l'émulateur,
    - lecture de la RAM.
- Une fois que le programme fonctionne correctement on effectue des essais sur l'application en intercalant la sonde de l'émulateur dans le support du micro-contrôleur de l'application.

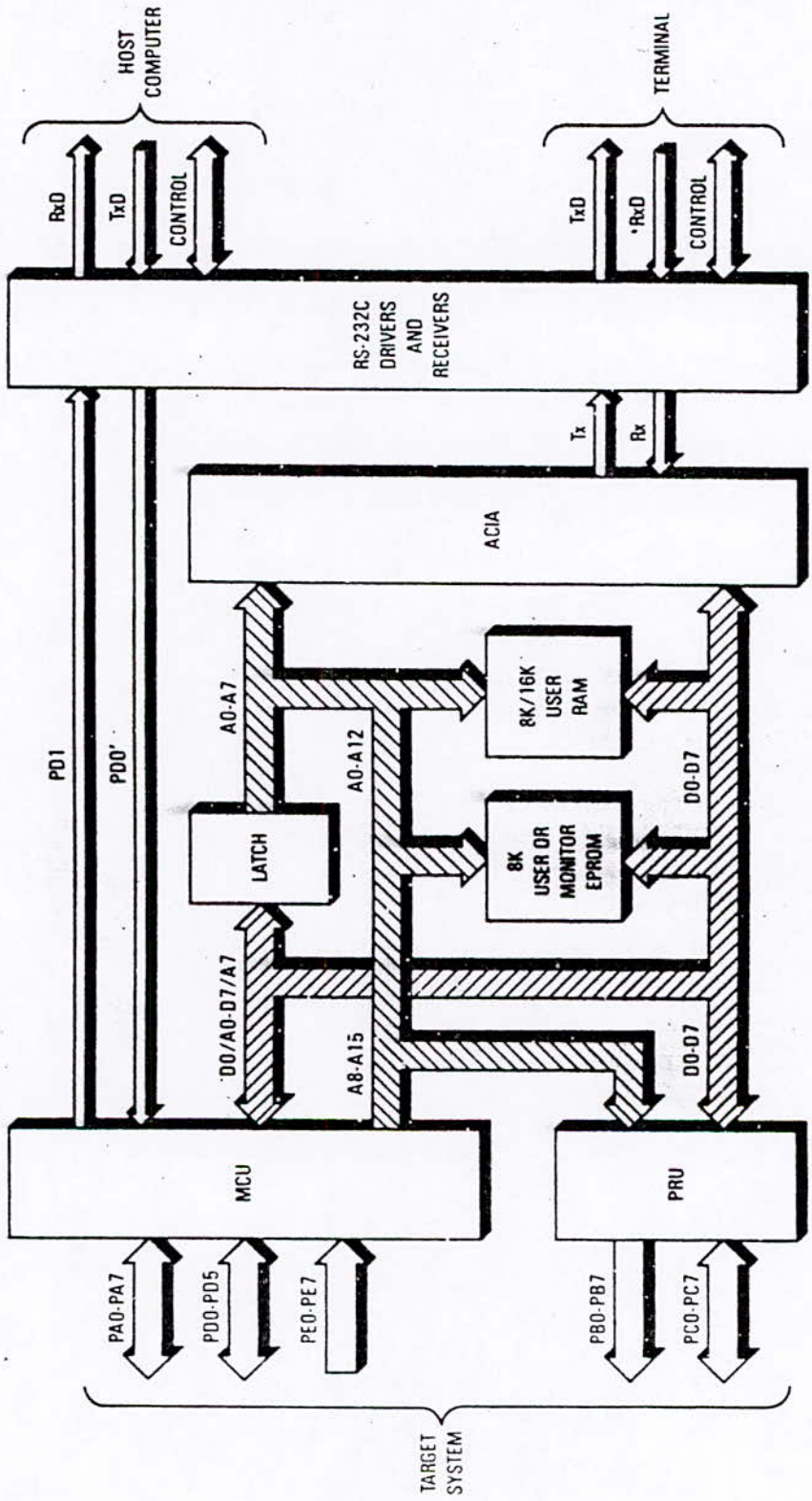
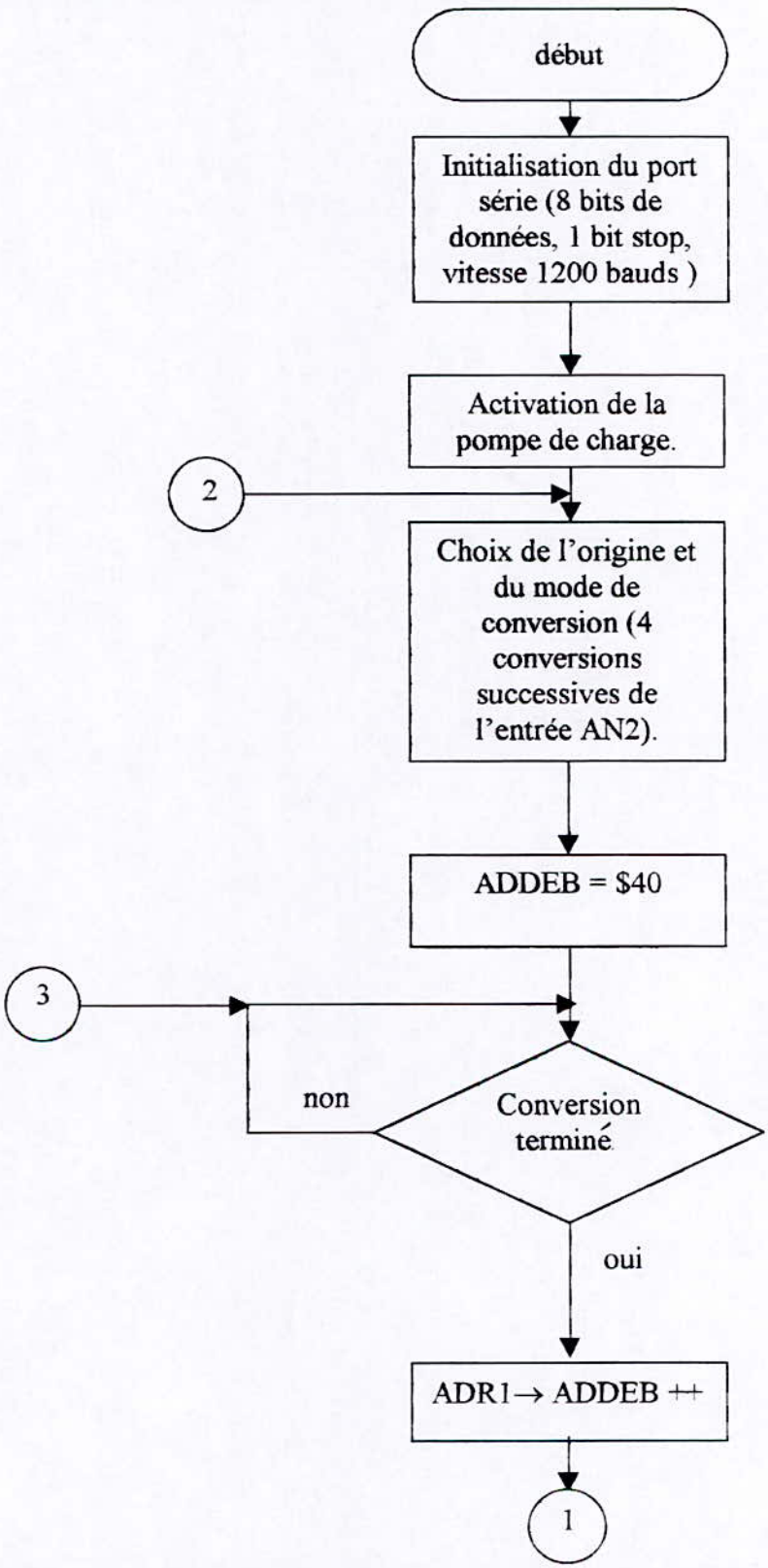


Fig. V.5 Schéma bloc du 68HC11EVB.

V.2.7-organigramme du Programme moniteur :



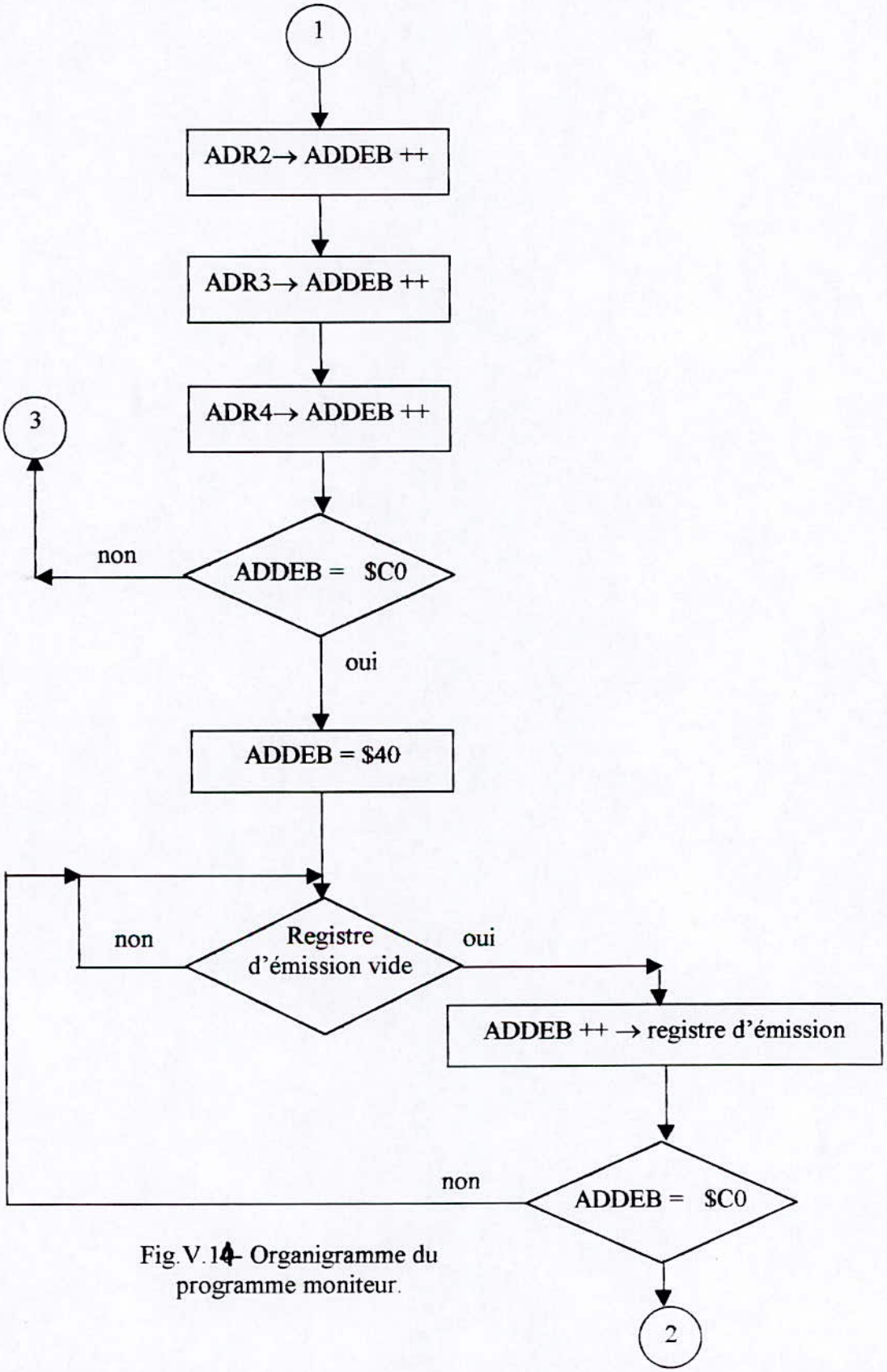


Fig.V.14- Organigramme du programme moniteur.

V.2.8-PROGRAMMATION :

V.2.8.1- SCHEMA SYNOPTIQUE DU PROGRAMMATEUR 68HC11EVM :

En plus du micro-contrôleur 68HC11A1 ou 68HC811E2 le programmateur 68HC11EVM (Fig. V.15.) contient :

- une pseudo EEPROM utilisateur,
- une pseudo ROM utilisateur,
- une EPROM utilisateur,
- une RAM utilisateur,
- deux PRU,
- une DUART (double UART),
- deux supports du micro-contrôleur à programmer l'un de type PLCC52 l'autre de type DIP48,

V.2.8.2- TRAVAIL EFFECTUE AU NIVEAU DU POGRAMMATEUR 68HC11EVM :

Après la mise au point du programme on doit le charger dans la mémoire du micro-contrôleur, c'est là où intervient le 68HC11EVB. Pour programmer le micro-contrôleur on doit passer par les étapes suivantes :

- placer le micro-contrôleur à programmer à sont emplacement sur le programmateur,
- établir la connexion entre le PC et le programmateur en utilisant le logiciel KERMIT,
- téléchargement du programme (sous format S19) dans l'EEPROM du micro-contrôleur résident,
- effacement de l'EEPROM du micro-contrôleur cible,
- transfert du programme de l'EEPROM du micro-contrôleur résident vers celle du micro-contrôleur cible,
- vérification du bon transfert du programme.

V.2.9-UTILISATION DU LOGICIEL PCBUG11 :

Nous avons conçu notre carte d'application de façon à permettre de programmer le micro-contrôleur directement par le PC en utilisant le logiciel PCBUG11 qui exploite à cet effet le mode BOOTSTRAP.
Pour cela on effectue les étapes suivantes :

- établissement d'une connexion matériel entre la carte d'application et le PC,
- positionnement du micro-contrôleur en mode BOOTSTRAP (MODA = MODB = 0),
- établissement d'une connexion logiciel entre le PC et la carte,
- enlever la protection de l'EEPROM du micro-contrôleur,
- téléchargement du programme (sous format S19) dans l'EEPROM du micro-contrôleur,

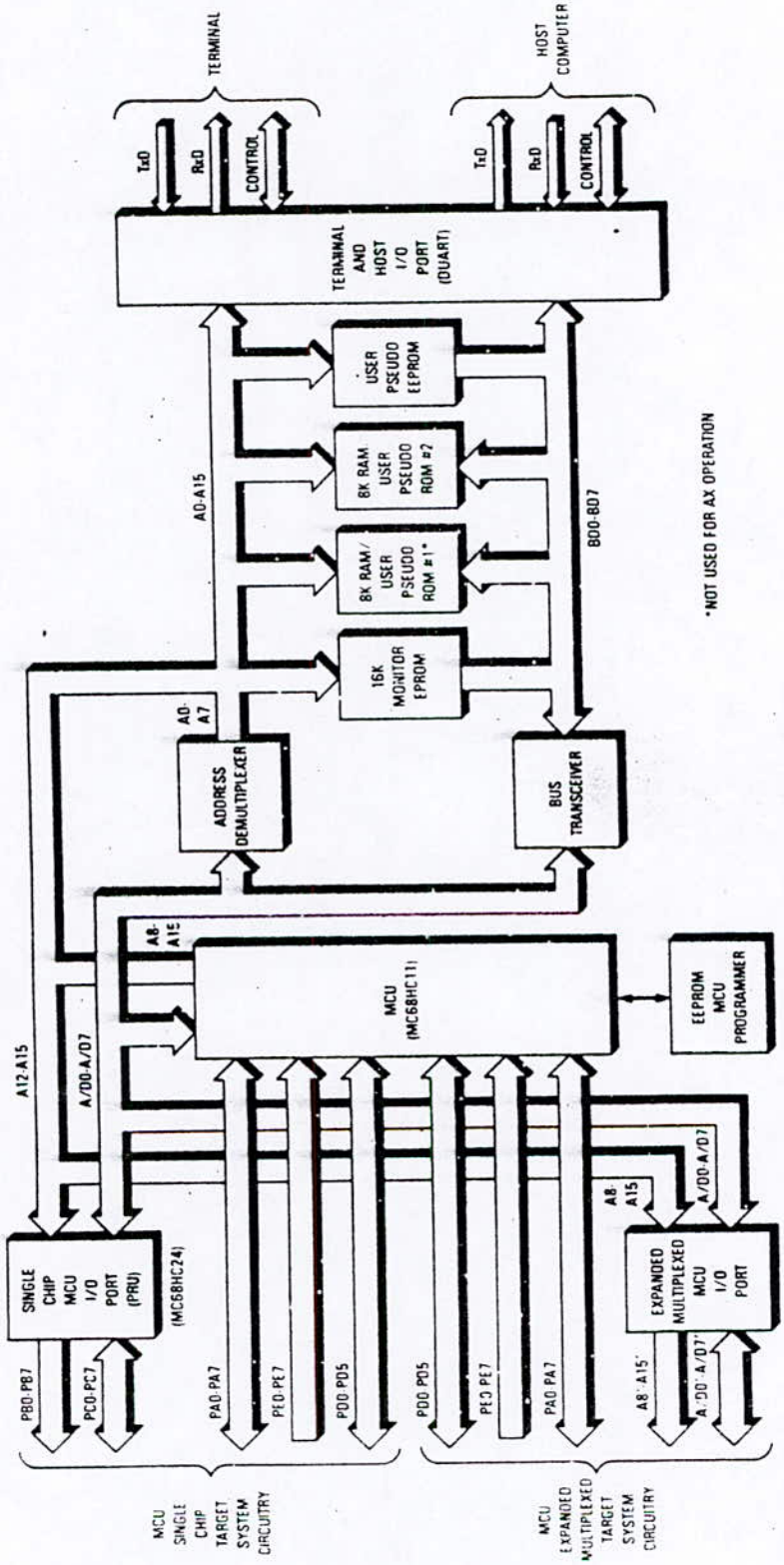


Fig. V.15 Schéma bloc du 68HC11EVM.

vérification du bon téléchargement du programme,

- lancement de l'exécution du programme et vérification du bon fonctionnement de celui-ci,
- correction d'éventuelles erreurs.

V.2.10-MISE EN ROUTE DE LA CARTE :

- raccorder la carte à la prise RS232 du PC (COM1 ou COM2),
- sélectionner le mode circuit seul (single chip) en utilisant le micro-switch K4,
- mettre la carte sous tension (5V),
- sélectionner l'entrée désirée (numérique ou analogique) en utilisant le micro-switch K5,
- lancer le programme RECEP.EXE en spécifiant les paramètres de la ligne de commande convenables,

- Faire un Reset matériel en appuyant sur le bouton poussoir K3 prévu à cet effet,

V.2.11- CARACTERISTIQUES DE LA CARTE :

V.2.11.1- FREQUENCE D'ECHANTILLONNAGE :

Le convertisseur A/D intégré du micro-contrôleur permet de prendre au maximum un échantillon chaques 32 cycles de l'horloge E. Sachant que la fréquence de l'horloge E est égale à 2 MHz, la fréquence maximale d'échantillonnage sera femax = 62.5 KHz.

En appliquant la condition de d'échantillonnage de ***SHANNON***, qui dit que la fréquence d'échantillonnage doit être d'au moins deux fois la fréquence maximale du signal à échantillonner, la fréquence maximale du signal susceptible d'être converti est de 31.25 KHz.

V.2.11.2- LIMITE EN TENSION :

La tension du signal analogique à convertir doit être comprise entre la tension de référence basse Vrefl et la tension de référence haute Vrefh.

Nous avons choisi Vrefl = 0 volt et Vrefh = 5 volts.

V.2.11.3- RESOLUTION :

Si on a un quantificateur sur N bits on aura :

- 2^N niveaux,
- un pas de quantification de $\Delta V_{REF} / 2^N$, ΔV_{REF} étant l'écart entre la tension de référence haute et la tension de référence basse.

Le convertisseur du micro-contrôleur est de type 8 bits ayant 5 volts comme tension de référence haute ce qui donne :

- 256 niveaux,
- pas de quantification de 19.5 mvolts.

CONCLUSION

CONCLUSION :

Nous avons réalisé dans notre travail, et compte tenu des moyens d'investigation, pour ne pas dire pratiquement inexistant, une carte d'interface pour PC. Ce projet nous a permis de s'approcher de très près des architectures des systèmes intelligents et par conséquent d'en concevoir et de pouvoir en proposer des nouvelles.

La carte que nous avons conçue, apporte une certaine originalité résidant dans l'utilisation du micro-contrôleur 68HC11 très convivial et très interactif avec lequel elle interagit avec l'utilisateur, ce qui la rend pour ainsi dire, très facile à utiliser que d'autre carte utilisant d'autre type de micro-contrôleur qui nécessite du matériel spécial pour l'implémentation.

Il reste cependant, que notre carte peut faire l'objet d'amélioration sur deux plans :

- *Au niveau matériel, et moyennant quelques modifications, peut devenir plus complète par des extensions mémoires et l'utilisation des autres ports d'entrées/sorties.*
- *Au niveau logiciel, notre application a besoin d'en être enrichi étant donnée qui nous est accordée. Néanmoins,*

le temps

notre réalisation est très satisfaisante et pourra constituer un prélude à un travail se faisant dans le même sens.

GLOSSAIRE :

ACK : Accuse de réception positif.

CAN : Convertisseur analogique numérique.

CM : Contrôle d'horloge.

COP : Contrôle de programme.

CRC : Contrôle de redondance cyclique.

CTS : Prêt a émettre.

DCD : Détection de porteuse.

DCE : Equipement de communication de données.

DSR : Poste de données prêt.

DTE : Equipement terminale de données.

DTR : Terminal de données prêt.

EOT : Fin de transmission.

IRQ : Interruption masquable.

NAK : Accusé de réception négatif.

PC : Ordinateur Personnel.

POR : Reset a la mise sous tension.

RI : Indicateur d'appel.

RS232 : Norme de communication série.

RTS : Demande a émettre.

RxD : Ligne de réception de données.

SCI : Interface série asynchrone.

SOH : Début d'en-tête.

SPI : Interface série synchrone.

STX : Début du message.

TxD : Ligne d'émission de données.

UART : Emetteur récepteur asynchrone universel.

XIRQ : Interruption non masquable.

BIBLIOGRAPHIE

Bibliographie :

[1]: les reseaux locaux d'entreprise
Auteur : F. Hoste
Edition : Editests –1984

[2]: IBM PC du laboratoire à l'industrie – interface- liaison- adaptation
Auteurs : G. Apruze & L. Frauty
Edition : Dunod –1986

[3]: Techniques de communication série sur PC et compatibles
Auteur : P.W. Goftan
Edition : Sybex –1987

[4]: Communication série
Auteur : Mark Nelson
Edition : Dunod –1994

[5]: Reseaux et micro-ordinateur
Auteur : P. H. Jesty
Edition : Masson –1987

[6]: Reseaux et télématique
Auteurs : G. Pugolle D. Seret D. Dronard E. Horlait
Edition : Eyrolles –1986

[7]: La bible du PC
Auteur : Michel Tischer
Edition : Micro application –1994

[8]: Manuel des interfaces
Auteur : Steve Laybson
Edition : Mc Grow Hill –1984

[9]: DOS programmer's reference
Auteur : Terry Dettman
Edition : QUE –1989

[10]: revue: Toute l'electronique
numéro 96
Aout-septembre –1984

[11]: TP liaison RS232: Etude des transmissions asynchrones séries et analyse de l'interface RS232
Edition : LSTI ENSERG (grenoble)

[12]: Electronique numérique: comprendre les micro-processeurs
Auteurs : Marcel Ginde & Denis Roux
Edition : Mc Grow Hill -1990

[13]: Interfaces pour micro-processeurs et micro-ordinateur
Auteur : H. Hilen
Edition : Radio –1983

[14]: Revue micro système
Octobre –1989

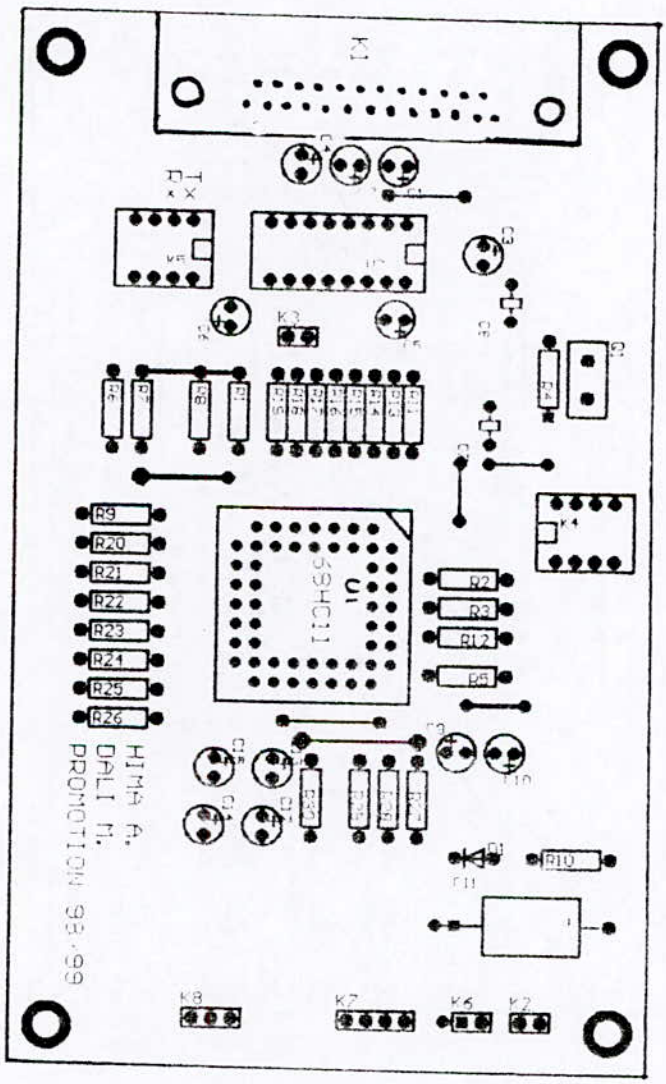
[15]: HCMOS micro-controller MC68HC11E9
Motorola –1988

[16]: HCMOS micro-controller MC68HC11A8
Motorola –1988

[17]: Micro-processor micro-controller and peripheral data
Motorola –1988
[18]: Reference manual
Motorola –1991
[19]: MC68HC11EVB: evaluation board user's manual
Motorola –1990
[20]: MC68HC11EVM: evaluation module user's manual
Motorola –1990
[21]: Le MC 68HC11
Auteur : Bernard Beyhyn
Edition : Hermes –1997
[22]: Le MC 68HC11
Auteur : C. Tavernier
Edition : Dunod –1997
[23]: Interface RS232
Auteur : M.P Seyer
Edition : Masson –1988
[24]: La liaison RS232
Auteur : Joe Combell
Edition : Sybex –1984
[25]: communication série en C++ guide de developpeur
Auteur : M. Nelson
Edition : Dunod–1994
[26]: nterfaces pour micro-processeurs et micro-ordinateur
Auteur : H. Hilen
Edition : Radio –1983
[27]: Conception et réalisation d'un logiciel dédié à la commande à distance d'une chaine d'acquisition sysmologique à base de DSP
Auteur : M. Bougesso & F. Hadiby
Thèse de PFE. INI – 96/1997
[28]: Reseau de PC à l'aide de courants porteurs : carte de communication.
Thèse de PFE . ENP –1996.
[29]: AN 1060: MC 68HC11 Bootstrap mode
Motorola
[30]: AN 1010: MC 68HC11 EEPROM programming from PC
Motorola
[31]: Revue Elecronique pratique numéro 215
Année : 1997

ANNEXES

ANNEXE 5. Schema de la carte réalisée.



- Placement des composants.

Liste des composants:

C1 a C6, C9, C10 : 1uF.

C7, C8 : 15pF.

C11 : 220 uF/25.

C12 a C15 : 0.01uF.

D1 : LED rouge.

U1 : MC68HC11A1.

U2 : MAX232.

Q1 : Quartz 8 MHz.

R1 a R3, R6 a R8 : 4,7 KHz.

R4 : 10 Mohms.

R5, R10, R27 a R30 : 1Kohms.

R9, R11 a R26 : 10Kohms.

K1 : connecteur DB25 male.

K2 : interrupteur.

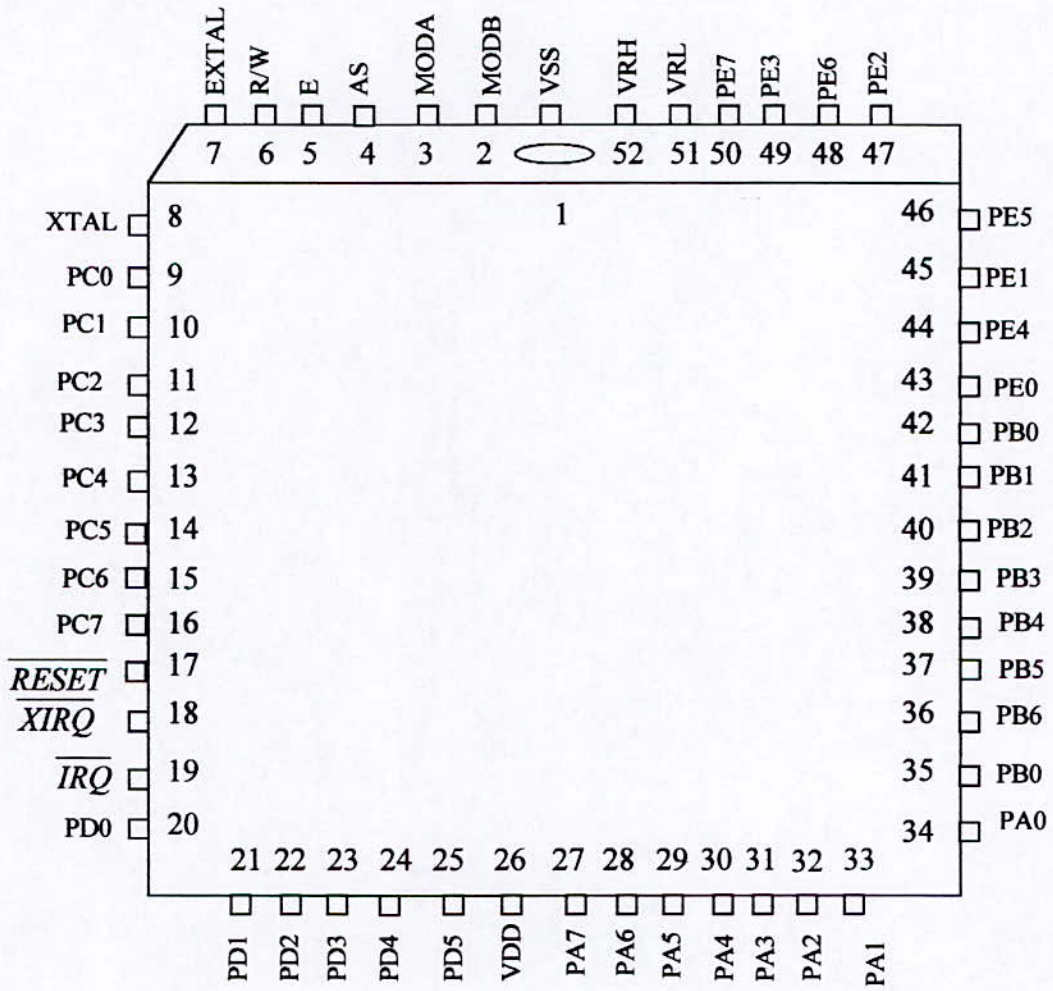
K3 : bouton poussoir.

K4, K5 : micro-switch a 4 boutons.

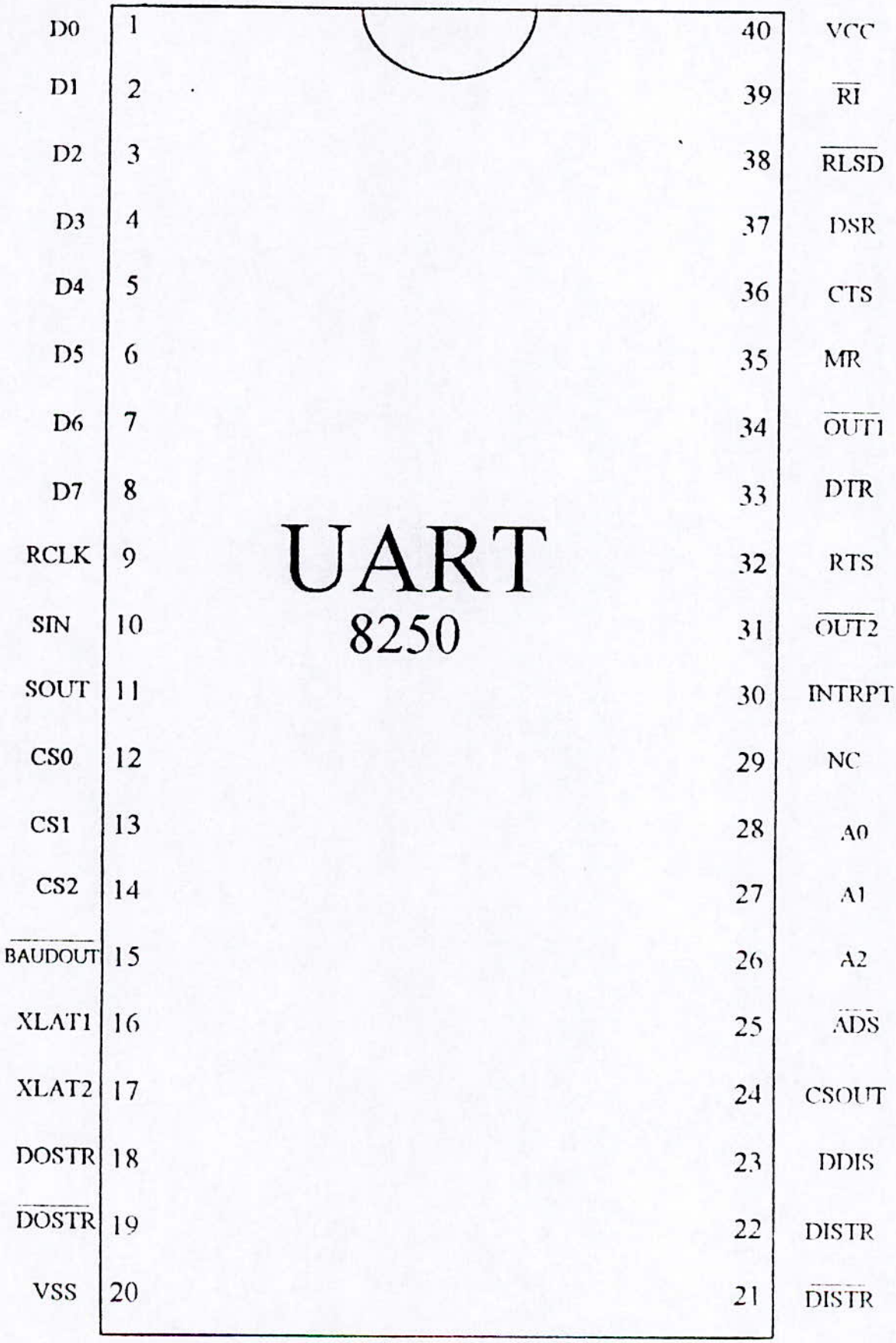
K6 : barette 2 broches.

K7 : barette 4 broches.

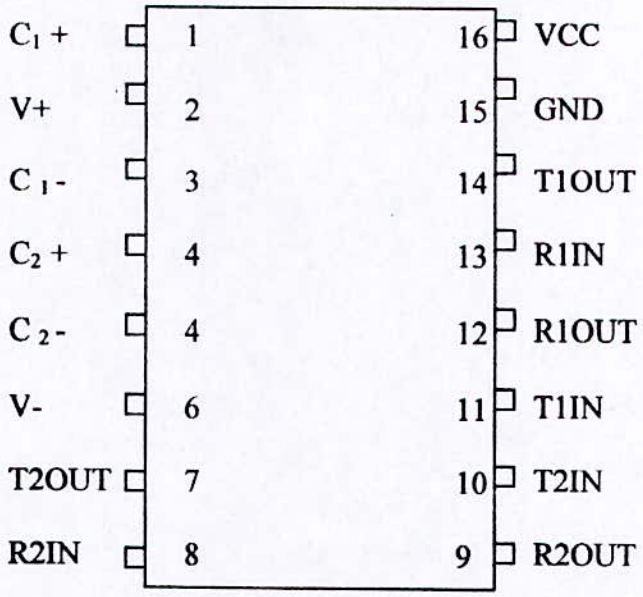
K8 : barette 3 broches.



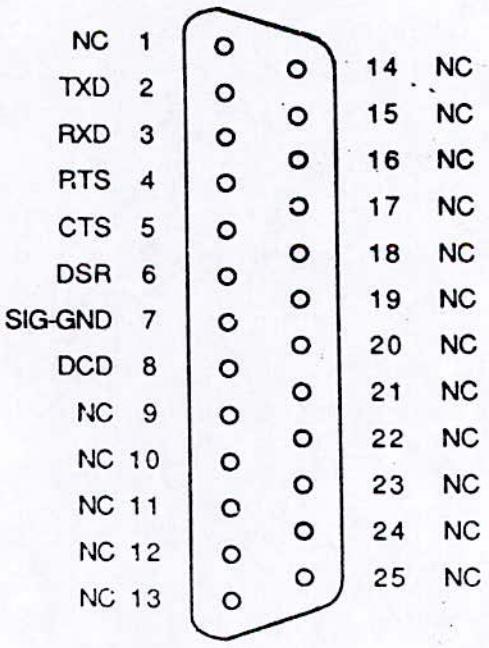
Brochage du micro-contrôleur 68HC11A1.



- Brochage de l'UART.



Brochage du MAX232.

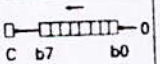
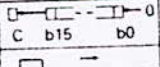
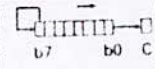


Brochage de connecteur DB25.

Table 3-1. Register and Control Bit Assignments (Sheet 1 of 2)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		
$1000	Bit 7	—	—	—	—	—	—	Bit 0	PORTA	I/O Port A
$1001									Reserved	
$1002	STAF	STAI	CWOM	HNDS	OIN	PLS	EGA	INVB	PIOC	Parallel I/O Control Register
$1003	Bit 7	—	—	—	—	—	—	Bit 0	PORTC	I/O Port C
$1004	Bit 7	—	—	—	—	—	—	Bit 0	PORTB	Output Port B
$1005	Bit 7	—	—	—	—	—	—	Bit 0	PORTCL	Alternate Latched Port C
$1006									Reserved	
$1007	Bit 7	—	—	—	—	—	—	Bit 0	DDRC	Data Direction for Port C
$1008			Bit 5	—	—	—	—	Bit 0	PORTD	I/O Port D
$1009			Bit 5	—	—	—	—	Bit 0	DDRD	Data Direction for Port D
$100A	Bit 7	—	—	—	—	—	—	Bit 0	PORTE	Input Port E
$100B	FOC1	FOC2	FOC3	FOC4	FOC5				CFORC	Compare Force Register
$100C	OC1M7	OC1M6	OC1M5	OC1M4	OC1M3				OC1M	OC1 Action Mask Register
$100D	OC1D7	OC1D6	OC1D5	OC1D4	OC1D3				OC1D	OC1 Action Data Register
$100E	Bit 15	—	—	—	—	—	—	Bit 8	TCNT	Timer Counter Register
$100F	Bit 7	—	—	—	—	—	—	Bit 0		
$1010	Bit 15	—	—	—	—	—	—	Bit 8	TIC1	Input Capture 1 Register
$1011	Bit 7	—	—	—	—	—	—	Bit 0		
$1012	Bit 15	—	—	—	—	—	—	Bit 8	TIC2	Input Capture 2 Register
$1013	Bit 7	—	—	—	—	—	—	Bit 0		
$1014	Bit 15	—	—	—	—	—	—	Bit 8	TIC3	Input Capture 3 Register
$1015	Bit 7	—	—	—	—	—	—	Bit 0		
$1016	Bit 15	—	—	—	—	—	—	Bit 8	TOC1	Output Compare 1 Register
$1017	Bit 7	—	—	—	—	—	—	Bit 0		
$1018	Bit 15	—	—	—	—	—	—	Bit 8	TOC2	Output Compare 2 Register
$1019	Bit 7	—	—	—	—	—	—	Bit 0		
$101A	Bit 15	—	—	—	—	—	—	Bit 8	TOC3	Output Compare 3 Register
$101B	Bit 7	—	—	—	—	—	—	Bit 0		
$101C	Bit 15	—	—	—	—	—	—	Bit 8	TOC4	Output Compare 4 Register
$101D	Bit 7	—	—	—	—	—	—	Bit 0		
$101E	Bit 15	—	—	—	—	—	—	Bit 8	TOC5	Output Compare 5 Register
$101F	Bit 7	—	—	—	—	—	—	Bit 0		

Table 10-1. MC68HC11A8 Instructions, Addressing Modes, and Execution Times (Sheet 1 of 7)

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal) Opcode	Operand(s)	Bytes	Cycle	Cycle by Cycle*	Condition Codes S	X	H	I	N	Z	V	C
ABA	Add Accumulators	A + B → A	INH	1B		1	2	2-1	-	-	↕	-	↕	↕	↕	↕
ABX	Add B to X	IX + 00:B → IX	INH	3A		1	3	2-2	-	-	-	-	-	-	-	-
ABY	Add B to Y	IY + 00:B → IY	INH	18 3A		2	4	2-4	-	-	-	-	-	-	-	-
ADCA (opr)	Add with Carry to A	A + M + C → A	A IMM	89	ii	2	2	3-1	-	-	↕	-	↕	↕	↕	↕
			A DIR	99	dd	2	3	4-1								
			A EXT	B9	hh ll	3	4	5-2								
			A IND,X	A9	ff	2	4	6-2								
			A IND,Y	18 A9	ff	3	5	7-2								
ADCB (opr)	Add with Carry to B	B + M + C → B	B IMM	C9	ii	2	2	3-1	-	-	↕	-	↕	↕	↕	↕
			B DIR	D9	dd	2	3	4-1								
			B EXT	F9	hh ll	3	4	5-2								
			B IND,X	E9	ff	2	4	6-2								
			B IND,Y	18 E9	ff	3	5	7-2								
ADDA (opr)	Add Memory to A	A + M → A	A IMM	8B	ii	2	2	3-1	-	-	↕	-	↕	↕	↕	↕
			A DIR	9B	dd	2	3	4-1								
			A EXT	BB	hh ll	3	4	5-2								
			A IND,X	AB	ff	2	4	6-2								
			A IND,Y	18 AB	ff	3	5	7-2								
ADDB (opr)	Add Memory to B	B + M → B	B IMM	CB	ii	2	2	3-1	-	-	↕	-	↕	↕	↕	↕
			B DIR	DB	dd	2	3	4-1								
			B EXT	FB	hh ll	3	4	5-2								
			B IND,X	EB	ff	2	4	6-2								
			B IND,Y	18 EB	ff	3	5	7-2								
ADDD (opr)	Add 16-Bit to D	D + M:M + 1 → D	IMM	C3	ll kk	3	4	3-3	-	-	-	-	↕	↕	↕	↕
			DIR	D3	dd	2	5	4-7								
			EXT	F3	hh ll	3	6	5-10								
			IND,X	E3	ff	2	6	6-10								
			IND, Y	18 E3	ff	3	7	7-8								
ANDA (opr)	AND A with Memory	A•M → A	A IMM	84	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			A DIR	94	dd	2	3	4-1								
			A EXT	B4	hh ll	3	4	5-2								
			A IND,X	A4	ff	2	4	6-2								
			A IND,Y	18 A4	ff	3	5	7-2								
ANDB (opr)	AND B with Memory	B•M → B	B IMM	C4	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			B DIR	D4	dd	2	3	4-1								
			B EXT	F4	hh ll	3	4	5-2								
			B IND,X	E4	ff	2	4	6-2								
			B IND,Y	18 E4	ff	3	5	7-2								
ASL (opr)	Arithmetic Shift Left	 C b7 b0	EXT	78	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	↕
			IND,X	68	ff	2	6	6-3								
			IND,Y	18 68	ff	3	7	7-3								
ASLA			A INH	48		1	2	2-1								
ASLB			B INH	58		1	2	2-1								
ASLD	Arithmetic Shift Left Double	 C b15 b0	INH	05		1	3	2-2	-	-	-	-	↕	↕	↕	↕
ASR (opr)	Arithmetic Shift Right	 b7 b0 C	EXT	77	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	↕
			IND,X	67	ff	2	6	6-3								
			IND,Y	18 67	ff	3	7	7-3								
ASRA			A INH	47		1	2	2-1								
ASRB			B INH	57		1	2	2-1								
BCC (rel)	Branch if Carry Clear	? C = 0	REL	24	rr	2	3	8-1	-	-	-	-	-	-	-	-
BCLR (opr) (msk)	Clear Bit(s)	M•(mm) → M	DIR	15	dd mm	3	6	4-10	-	-	-	-	↕	↕	0	-
			IND,X	1D	ff mm	3	7	6-13								
			IND,Y	18 1D	ff mm	4	8	7-10								
BCS (rel)	Branch if Carry Set	? C = 1	REL	25	rr	2	3	8-1	-	-	-	-	-	-	-	-
BEQ (rel)	Branch if = Zero	? Z = 1	REL	27	rr	2	3	8-1	-	-	-	-	-	-	-	-

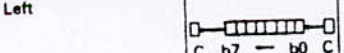
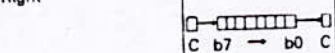
* Cycle-by-cycle number provides a reference to Tables 10-2 through 10-8 which detail cycle-by-cycle operation.
Example: Table 10-1 Cycle-by-Cycle column reference number 2-4 equals Table 10-2 line item 2-4.

Table 10-1. MC68HC11A8 Instructions, Addressing Modes, and Execution Times (Sheet 3 of 7)

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal)		Bytes	Cycle	Cycle by Cycle*	Condition Codes						
				Opcode	Operand(s)				S	X	H	I	N	Z	V	C
CMPB (opr)	Compare B to Memory	B – M	B IMM	C1	ii	2	2	3-1	-	-	-	-	↕	↕	↕	↕
			B DIR	D1	dd	2	3	4-1								
			B EXT	F1	hh ll	3	4	5-2								
			B IND,X	E1	ff	2	4	6-2								
			B IND,Y	18 E1	ff	3	5	7-2								
COM (opr)	1's Complement Memory Byte	$FF – M → M	EXT	73	hh ll	3	6	5-8	-	-	-	-	↕	↕	0	1
			IND,X	63	ff	2	6	6-3								
			IND,Y	18 63	ff	3	7	7-3								
COMA	1's Complement A	$FF – A → A	A INH	43		1	2	2-1	-	-	-	-	↕	↕	0	1
COMB	1's Complement B	$FF – B → B	B INH	53		1	2	2-1	-	-	-	-	↕	↕	0	1
CPD (opr)	Compare D to Memory 16-Bit	D – M:M + 1	IMM	1A 83	jj kk	4	5	3-5	-	-	-	-	↕	↕	↕	↕
			DIR	1A 93	dd	3	6	4-9								
			EXT	1A B3	hh ll	4	7	5-11								
			IND,X	1A A3	ff	3	7	6-11								
			IND,Y	CD A3	ff	3	7	7-8								
CPX (opr)	Compare X to Memory 16-Bit	IX – M:M + 1	IMM	8C	jj kk	3	4	3-3	-	-	-	-	↕	↕	↕	↕
			DIR	9C	dd	2	5	4-7								
			EXT	BC	hh ll	3	6	5-10								
			IND,X	AC	ff	2	6	6-10								
			IND,Y	CD AC	ff	3	7	7-8								
CPY (opr)	Compare Y to Memory 16-Bit	IY – M:M + 1	IMM	18 8C	jj kk	4	5	3-5	-	-	-	-	↕	↕	↕	↕
			DIR	18 9C	dd	3	6	4-9								
			EXT	18 BC	hh ll	4	7	5-11								
			IND,X	1A AC	ff	3	7	6-11								
			IND,Y	18 AC	ff	3	7	7-8								
DAA	Decimal Adjust A	Adjust Sum to BCD	INH	19		1	2	2-1	-	-	-	-	↕	↕	↕	↕
DEC (opr)	Decrement Memory Byte	M – 1 → M	EXT	7A	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	-
			IND,X	6A	ff	2	6	6-3								
			IND,Y	18 6A	ff	3	7	7-3								
DECA	Decrement Accumulator A	A – 1 → A	A INH	4A		1	2	2-1	-	-	-	-	↕	↕	↕	-
DECB	Decrement Accumulator B	B – 1 → B	B INH	5A		1	2	2-1	-	-	-	-	↕	↕	↕	-
DES	Decrement Stack Pointer	SP – 1 → SP	INH	34		1	3	2-3	-	-	-	-	-	-	-	-
DEX	Decrement Index Register X	IX – 1 → IX	INH	09		1	3	2-2	-	-	-	-	-	↕	-	-
DEY	Decrement Index Register Y	IY – 1 → IY	INH	18 09		2	4	2-4	-	-	-	-	-	↕	-	-
EORA (opr)	Exclusive OR A with Memory	A ⊕ M → A	A IMM	88	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			A DIR	98	dd	2	3	4-1								
			A EXT	B8	hh ll	3	4	5-2								
			A IND,X	A8	ff	2	4	6-2								
			A IND,Y	18 A8	ff	3	5	7-2								
EORB (opr)	Exclusive OR B with Memory	B ⊕ M → B	B IMM	C8	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			B DIR	D8	dd	2	3	4-1								
			B EXT	F8	hh ll	3	4	5-2								
			B IND,X	E8	ff	2	4	6-2								
			B IND,Y	18 E8	ff	3	5	7-2								
FDIV	Fractional Divide 16 by 16	D/IX → IX; r → D	INH	03		1	41	2-17	-	-	-	-	-	↕	↕	↕
IDIV	Integer Divide 16 by 16	D/IX → IX; r → D	INH	02		1	41	2-17	-	-	-	-	-	↕	0	↕
INC (opr)	Increment Memory Byte	M + 1 → M	EXT	7C	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	-
			IND,X	6C	ff	2	6	6-3								
			IND,Y	18 6C	ff	3	7	7-3								
INCA	Increment Accumulator A	A + 1 → A	A INH	4C		1	2	2-1	-	-	-	-	↕	↕	↕	-
INCB	Increment Accumulator B	B + 1 → B	B INH	5C		1	2	2-1	-	-	-	-	↕	↕	↕	-
INS	Increment Stack Pointer	SP + 1 → SP	INH	31		1	3	2-3	-	-	-	-	-	-	-	-

* Cycle-by-cycle number provides a reference to Tables 10-2 through 10-8 which detail cycle-by-cycle operation.
Example: Table 10-1 Cycle-by-Cycle column reference number 2-4 equals Table 10-2 line item 2-4.

Table 10-1. MC68HC11A8 Instructions, Addressing Modes, and Execution Times (Sheet 5 of 7)

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal) Opcode	Operand(s)	Bytes	Cycle	Cycle by Cycle*	Condition Codes S	X	H	I	N	Z	V	C
NEG (opr)	2's Complement Memory Byte	0 − M → M	EXT	70	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	↕
			IND,X	60	ff	2	6	6-3								
			IND,Y	18 60	ff	3	7	7-3								
NEGA	2's Complement A	0 − A → A	A INH	40		1	2	2-1	-	-	-	-	↕	↕	↕	↕
NEGB	2's Complement B	0 − B → B	B INH	50		1	2	2-1	-	-	-	-	↕	↕	↕	↕
NOP	No Operation	No Operation	INH	01		1	2	2-1	-	-	-	-	-	-	-	-
ORAA (opr)	OR Accumulator A (Inclusive)	A + M → A	A IMM	8A	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			A DIR	9A	dd	2	3	4-1								
			A EXT	BA	hh ll	3	4	5-2								
			A IND,X	AA	ff	2	4	6-2								
			A IND,Y	18 AA	ff	3	5	7-2								
ORAB (opr)	OR Accumulator B (Inclusive)	B + M → B	B IMM	CA	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			B DIR	DA	dd	2	3	4-1								
			B EXT	FA	hh ll	3	4	5-2								
			B IND,X	EA	ff	2	4	6-2								
			B IND,Y	18 EA	ff	3	5	7-2								
PSHA	Push A onto Stack	A → Stk, SP = SP − 1	A INH	36		1	3	2-6	-	-	-	-	-	-	-	-
PSHB	Push B onto Stack	B → Stk, SP = SP − 1	B INH	37		1	3	2-6	-	-	-	-	-	-	-	-
PSHX	Push X onto Stack (Lo First)	IX → Stk, SP = SP − 2	INH	3C		1	4	2-7	-	-	-	-	-	-	-	-
PSHY	Push Y onto Stack (Lo First)	IY → Stk, SP = SP − 2	INH	18 3C		2	5	2-8	-	-	-	-	-	-	-	-
PULA	Pull A from Stack	SP = SP + 1, A ← Stk	A INH	32		1	4	2-9	-	-	-	-	-	-	-	-
PULB	Pull B from Stack	SP = SP + 1, B ← Stk	B INH	33		1	4	2-9	-	-	-	-	-	-	-	-
PULX	Pull X from Stack (Hi First)	SP = SP + 2, IX ← Stk	INH	38		1	5	2-10	-	-	-	-	-	-	-	-
PULY	Pull Y from Stack (Hi First)	SP = SP + 2, IY ← Stk	INH	18 38		2	6	2-11	-	-	-	-	-	-	-	-
ROL (opr)	Rotate Left	 C b7 — b0 C	EXT	79	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	↕
			IND,X	69	ff	2	6	6-3								
			IND,Y	18 69	ff	3	7	7-3								
ROLA			A INH	49		1	2	2-1								
ROLB			B INH	59		1	2	2-1								
ROR (opr)	Rotate Right	 C b7 → b0 C	EXT	76	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	↕
			IND,X	66	ff	2	6	6-3								
			IND,Y	18 66	ff	3	7	7-3								
RORA			A INH	46		1	2	2-1								
RORB			B INH	56		1	2	2-1								
RTI	Return from Interrupt	See Special Ops	INH	3B		1	12	2-14	↕	↕	↕	↕	↕	↕	↕	↕
RTS	Return from Subroutine	See Special Ops	INH	39		1	5	2-12	-	-	-	-	-	-	-	-
SBA	Subtract B from A	A − B → A	INH	10		1	2	2-1	-	-	-	-	↕	↕	↕	↕
SBCA (opr)	Subtract with Carry from A	A − M − C → A	A IMM	82	ii	2	2	3-1	-	-	-	-	↕	↕	↕	↕
			A DIR	92	dd	2	3	4-1								
			A EXT	B2	hh ll	3	4	5-2								
			A IND,X	A2	ff	2	4	6-2								
			A IND,Y	18 A2	ff	3	5	7-2								
SBCB (opr)	Subtract with Carry from B	B − M − C → B	B IMM	C2	ii	2	2	3-1	-	-	-	-	↕	↕	↕	↕
			B DIR	D2	dd	2	3	4-1								
			B EXT	F2	hh ll	3	4	5-2								
			B IND,X	E2	ff	2	4	6-2								
			B IND,Y	18 E2	ff	3	5	7-2								
SEC	Set Carry	1 → C	INH	OD		1	2	2-1	-	-	-	-	-	-	-	1
SEI	Set Interrupt Mask	1 → I	INH	OF		1	2	2-1	-	-	-	1	-	-	-	-
SEV	Set Overflow Flag	1 → V	INH	OB		1	2	2-1	-	-	-	-	-	-	1	-

* Cycle by-cycle number provides a reference to Tables 10-2 through 10-8 which detail cycle-by-cycle operation.
Example: Table 10-1 Cycle-by-Cycle column reference number 2-4 equals Table 10-2 line item 2-4.

Table 10-1. MC68HC11A8 Instructions, Addressing Modes, and Execution Times (Sheet 7 of 7)

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal)		# Bytes	Cycle	Cycle by Cycle*	Condition Codes							
				Opcode	Operand(s)				S	X	H	I	N	Z	V	C
TXS	Transfer X to Stack Pointer	IX – 1 → SP	INH	35		1	3	2-2	-	-	-	-	-	-	-	-
TYS	Transfer Y to Stack Pointer	IY – 1 → SP	INH	18 35		2	4	2-4	-	-	-	-	-	-	-	-
WAI	Wait for Interrupt	Stack Regs & WAIT	INH	3E		1	***	2-16	-	-	-	-	-	-	-	-
XGDX	Exchange D with X	IX → D, D → IX	INH	8F		1	3	2-2	-	-	-	-	-	-	-	-
XGDY	Exchange D with Y	IY → D, D → IY	INH	18 8F		2	4	2-4	-	-	-	-	-	-	-	-

*Cycle-by-cycle number provides a reference to Tables 10-2 through 10-8 which detail cycle-by-cycle operation.
Example: Table 10-1 Cycle-by-Cycle column reference number 2-4 equals Table 10-2 line item 2-4.

**Infinity or Until Reset Occurs

***12 Cycles are used beginning with the opcode fetch. A wait state is entered which remains in effect for an integer number of MPU E-clock cycles (n) until an interrupt is recognized. Finally, two additional cycles are used to fetch the appropriate interrupt vector (14 + n total).

dd = 8-Bit Direct Address ($0000 – $00FF) (High Byte Assumed to be $00)
ff = 8-Bit Positive Offset $00 (0) to $FF (255) (Is Added to Index)
hh = High Order Byte of 16-Bit Extended Address
ii = One Byte of Immediate Data
jj = High Order Byte of 16-Bit Immediate Data
kk = Low Order Byte of 16-Bit Immediate Data
ll = Low Order Byte of 16-Bit Extended Address
mm = 8-Bit Bit Mask (Set Bits to be Affected)
rr = Signed Relative Offset $80 (– 128) to $7F (+ 127)
(Offset Relative to the Address Following the Machine Code Offset Byte)

ANNEXE III. Programmes développés.

```
/*****************************************************************
*****/
/*                              R E C E P . C P P
    */
/**-------------------------------------------------------------
---**/
/*  Tache        : Reception de fichier ou de bloc de donnees
    */
/*                 par l'interface serie.
    */
/**-------------------------------------------------------------
-  **/
/*  Auteurs              : HIMA A. ; DALI M.
    */
/*****************************************************************
*****/
#include <stdio.h>
#include <dos.h>
#include <process.h>
#include <string.h>

#include "RECEP.H" // il faut que le fichier include recep.h soit
dans
                   // le meme repertoir que recep.cpp

/*****************************************************************
*****/
/* ser_UARTType : Determine le type du chip UART
    */
/**-------------------------------------------------------------
----*/
/* Entree : iSerPort      - Adresse de base de l'interface a teste
r   */
/* Sortie : 0 (NOSER)     - Chip UART introuvable
    */
/*          1 (INS8250)   - Chip INS8250 ou INS8250-B Chip
    */
/*          2 (NS16450)   - INS8250A, INS82C50A, NS16450,NS16C450
    */
/*          3 (NS16550A)  - Chip NS16550A
    */
/*          4 (NS16C552)  - Chip NS16C55
    */
/*****************************************************************
*****/
INT ser_UARTType( INT iSerPort )
{
  /*- Tester fonctionnalites de base ----------------------------
--- */

  outportb( iSerPort + SER_LINE_CONTROL, 0xAA );
                        /* Diviseur-Latch positionne */
```

```
  if( inp( iSerPort + SER_LINE_CONTROL ) != 0xAA ) return NOSER;

  outportb( iSerPort + SER_DIVISOR_MSB, 0x55 );
                          /* Description du diviseur */
  if( inp( iSerPort + SER_DIVISOR_MSB ) != 0x55 ) return NOSER;

  outportb( iSerPort + SER_LINE_CONTROL, 0x55 );
                            /* Efface le diviseur */
  if( inp( iSerPort + SER_LINE_CONTROL ) != 0x55 ) return NOSER;

  outportb( iSerPort + SER_IRQ_ENABLE, 0x55 );
  if( inp( iSerPort + SER_IRQ_ENABLE ) != 0x05 ) return NOSER;

  outportb( iSerPort + SER_FIFO, 0 );  /* Efface FIFO et IRQ */
  outportb( iSerPort + SER_IRQ_ENABLE, 0 );
  if( inp( iSerPort + SER_IRQ_ID ) != 1 ) return NOSER;

  outportb( iSerPort + SER_MODEM_CONTROL, 0xF5 );
  if( inp( iSerPort + SER_MODEM_CONTROL ) != 0x15 ) return
NOSER;

  outportb( iSerPort + SER_MODEM_CONTROL, SER_MCR_LOOP );  /* Bouc
le */
  inp( iSerPort + SER_MODEM_STATUS );
  if( ( inp( iSerPort + SER_MODEM_STATUS ) & 0xF0 ) != 0 )
return NOSER;

  outportb( iSerPort + SER_MODEM_CONTROL, 0x1F );
  if( ( inp( iSerPort + SER_MODEM_STATUS ) & 0xF0 ) != 0xF0 )
      return NOSER;

  outportb( iSerPort + SER_MODEM_CONTROL, SER_MCR_DTR | SER_MCR_RT
S
);

  outportb( iSerPort + SER_SCRATCH, 0x55 ); /* le registre Scratch
 existe?*/
  if( inp( iSerPort + SER_SCRATCH ) != 0x55 ) return INS8250;
  outportb( iSerPort + SER_SCRATCH, 0 );

  outportb( iSerPort + SER_FIFO, 0xCF );          /* le FIFO est pr
esent?  */
  if( ( inp( iSerPort + SER_IRQ_ID ) & 0xC0 ) != 0xC0 ) return NS1
6450;
  outportb( iSerPort + SER_FIFO, 0 );

                /* le registre Alternate-Function est present? */

  outportb( iSerPort + SER_LINE_CONTROL, SER_LCR_SETDIVISOR );
  outportb( iSerPort + SER_2FUNCTION, 0x07 );
  if( inp( iSerPort + SER_2FUNCTION ) != 0x07 )
  {
```

```
    outportb( iSerPort + SER_LINE_CONTROL, 0 );
    return NS16550A;
  }
  outportb( iSerPort + SER_LINE_CONTROL, 0 );   /* Reinitialiser *
/
  outportb( iSerPort + SER_2FUNCTION, 0 );
  return NS16C552;
}

/*****************************************************************
****/
/* ser_Init : Initialiser l'interface serie
   */
/*
   */
/*---------------------------------------------------------------
--**/
/* Entree : iSerPort - Adresse de base de l'interface serie
   */
/*                     qu'il faut initialiser.
   */
/*           lBaud    - Vitesse  ( von 1 - 115200 )
   */
/*           bParams  - Masque de bits des autres parametres
   */
/*                       (a voir Bits SER_LCR_...-Bits)
   */
/* Sortie :   TRUE  - initialisation de l'interface reussie
   */
/*            FALSE - Interface introuvable
   */
/*****************************************************************
****/
INT ser_Init( INT  iSerPort, LONG lBaudRate, BYTE bParams )
{ WORD uDivisor;  if( ser_UARTType( iSerPort ) != NOSER )
 {                                      /* Calcul du diviseur de ba
ud */
    uDivisor = ( WORD )( SER_MAXBAUD / lBaudRate );
    outportb( iSerPort + SER_LINE_CONTROL, /* Permet l'acces du di
viseur */
         inportb( SER_LINE_CONTROL ) | SER_LCR_SETDIVISOR );
                     /* Positionne le diviseur de baud */
    outportb( iSerPort + SER_DIVISOR_LSB, LOBYTE( uDivisor ) );
    outportb( iSerPort + SER_DIVISOR_MSB, HIBYTE( uDivisor ) );
                          /* Empeche son acces */
    outportb( iSerPort + SER_LINE_CONTROL,
         inportb( SER_LINE_CONTROL ) & ~SER_LCR_SETDIVISOR);
 /*- Pour fixer les autres parametres attendre le positionnement e
n  */
 /* arriere des taux de baud Latch parce que l'operation efface to
us */
 /*                                     les parametres de l'interfa
```

```
ce */
      /* Fixer les parametres de transmission - sauf pour la vites
se */
    outportb( iSerPort + SER_LINE_CONTROL, bParams );
       /* Lecture d'un octet, pour placer les erreurs eventuelleme
nt */
       /*                                              presentes en arrie
re */
    inp( iSerPort + SER_TXBUFFER );
    return TRUE;
  }
  return FALSE;
}
/*****************************************************************
*****/
/*  ser_IsDataAvaiable : Les donnees sont-elles pretes a la lectur
e ? */
/**-------------------------------------------------------------------
---**/
/* Entree : iSerPort - Port de base de l'interface a tester
    */
/* Sortie : == 0 : Aucun octet n'existe pour une lecture
    */
/*          != 0 : L'octet est disponible
    */
/**-------------------------------------------------------------------
---**/
/* Info : L'octet se transmet bit par bit. Il est de nouveau
    */
/*        complet au niveau de l'interface receptrice qui l'a
    */
/*        recompose. L'action se verifie par cette fonction .
    */
/*****************************************************************
*****/
INT ser_IsDataAvaiable( INT iSerPort )
{
  return inp( iSerPort + SER_LINE_STATUS ) & SER_LSR_DATARECEIVED;
}

/*****************************************************************
*****/
/* ser_IsWritingPossible :
    */
/*                    L'interface, peut-elle envoyer encore un oct
et ?*/
/**-------------------------------------------------------------------
---**/
/* Entree : iSerPort - Port de base de l'interface a tester
    */
/* Sortie : == 0 : Il ne faut pas envoyer l'octet.
    */
```

```
/*           != 0 : l'interface est prete a emettre
    */
/**-------------------------------------------------------------
-- **/
/*Info : Il ne faut pas utiliser une interface serie pour envoyer
    */
/*       des octets dans les cas suivants :
    */
/*       1. Un octet recu n'a pas encore ete appele par l'interfac
e   */
/*       2. Une demande d'emission n'a pas encore ete executee
    */
/*************************************************************
*****/
INT ser_IsWritingPossible( INT iSerPort )
{
  return ( inp( iSerPort + SER_LINE_STATUS ) & SER_LSR_TSREMPTY);}

/*************************************************************
*****/
/* ser_WriteByte : Envoi d'un octet
    */
/**-------------------------------------------------------------
---**/
/*Entree : iSerPort - Port de base de l'interface pour
    */
/*                    envoyer un octet.
    */
/*          bData     - octet a envoyer
    */
/*          uTimeOut - Nombre de passages effectues dans la boucle
    */
/*                     avant que l'echec d'une emission n'est sign
ale */
/*                     par une erreur TimeOut.
    */
/*                     (Si iTimeOut = 0 l'attente continue
    */
/*          bSigMask - Masque de bits des lignes de signal a teste
r   */
/*                     (RTS, CTS, CD, RI)
    */
/*          bSigVals - Etat des lignes de signal apres avoir
    */
/*                     demasque le masque ci-dessus.
    */
/*Sortie :  = 0 - octets ont ete envoyes
    */
/*          <> 0 - Erreurs
    */
/*************************************************************
*****/
```

```
INT ser_WriteByte( INT iSerPort, BYTE bData, UINT uTimeOut,
           BYTE bSigMask, BYTE bSigVals )
{ if( uTimeOut )                                        /* Boucle time-
out */
  {
    while( !ser_IsWritingPossible( iSerPort ) && uTimeOut )
uTimeOut--;
    if( !uTimeOut ) return SER_ERRTIMEOUT;
  }
  else while( !ser_IsWritingPossible( iSerPort ) );        /* Atten
te! */

                          /* Test des lignes de signal */
  if( ( ( BYTE ) inp( iSerPort + SER_MODEM_STATUS ) & bSigMask
) ==
          bSigVals )
  {
         /* Transmission des octets a emettre vers l'interface */
    outportb( iSerPort + SER_TXBUFFER, bData );
                          /* Retourne erreur interface */
    return inp( iSerPort + SER_LINE_STATUS ) &
SER_LSR_ERRORMSK;
  }
  else return SER_ERRSIGNALS;
}

/*************************************************************************
*****/
/* ser_ReadByte : Reception d'un octet
    */
/* Entree : iSerPort - Port de base de l'interface utilisee
    */
/*                     pour recevoir un octet
    */
/*           Data    - la variable byte accueille l'octet
    */
/*                     recu.
    */
/*          uTimeOut - Nombre de passages dans la boucle
    */
/*                     avant qu'une erreur TimeOut ne signale
    */
/*                     l'echec de la reception.
    */
/*                      (Si iTimeOut = 0 l'attente continue )
    */
/*          bSigMask - Masque de bits des lignes de signal a
    */
/*                     tester
    */
/*                      (RTS, CTS, CD, RI)
```

```
    */
/*          bSigVals - Etat des lignes de signal apres
    */
/*                     avoir demasque avec le masque ci-dessus.
    */
/*          Sortie :  = 0 - octets ont ete envoyes
    */
/*          != 0 - Erreurs
    */
/*****************************************************************
*****/
INT ser_ReadByte( INT iSerPort, PBYTE pData, UINT uTimeOut,
          BYTE bSigMask, BYTE bSigVals )
{ if( uTimeOut )                                    /* Boucle TimeOu
t */
  {
    while( !ser_IsDataAvaiable( iSerPort ) && uTimeOut )
uTimeOut--;
    if( !uTimeOut ) return SER_ERRTIMEOUT;
  }
  else while( !ser_IsDataAvaiable( iSerPort ) );          /* Atten
te! */

                        /* Test des lignes de signaux */
  if( ( ( BYTE ) inp( iSerPort + SER_MODEM_STATUS ) & bSigMask)
       ==  bSigVals )

  {                          /* Lecture de l'octet recu par l'interf
ace */
    *pData = ( BYTE )inp( iSerPort + SER_RXBUFFER );
    return inp( iSerPort + SER_LINE_STATUS ) & SER_LSR_ERRORMSK;
  }
  else return SER_ERRSIGNALS;
}


/*****************************************************************
*****/
/* trans_WriteAck : Envoi acknowledge positif par l'interface
    */
/**-------------------------------------------------------------
---**/
/* Entree : iSerPort - Adresse de base de l'interface
    */
/* Sortie : Niveau d'erreur
    */
/*****************************************************************
*****/
INT trans_SendAck( INT iSerPort )
{
  if( ser_WriteByte( iSerPort, ASCII_ACK, 0x8000,0,0) == SER_SUCCE
SS )
```

```
    return TRANS_SUCCESS;
  return TRANS_TRANSERROR;
}


/*********************************************************************
*****/
/* trans_ReceivePaket : Recoit paket de donnees de l'interface
    */
/**-----------------------------------------------------------------
---**/
/* Entree : iSerPort   - Adresse de base de l'interface a utiliser
    */
/*          pPaket     - Adresse du buffer d'entree
    */
/*          iPaketSize - Taille du buffer
    */
/* Sortie : Niveau d'erreur
    */
/*********************************************************************
*****/
INT trans_ReceivePaket( INT iSerPort, PBYTE pPaket, UINT iPaketSiz
e )
{
  UINT  uAction, uTimeOut, uActByte;
  BYTE bData, bDataOk;
  INT  iAckError;

  uTimeOut = 0xFFFF;
  uAction = RECEIVE_SIGNAL;

  uActByte = 0;

  while( uTimeOut )
  {
    bDataOk = FALSE;                          /* il n'y a pas de donnee
s */
    if( ser_IsDataAvaiable( iSerPort ) )
      if( ser_ReadByte( iSerPort, &bData, 1, 0, 0 ) != SER_SUCCESS
 )
                return TRANS_TRANSERROR;
      else bDataOk = TRUE;                          /* ou quand meme ?
*/
    if( bDataOk )
    {
      switch( uAction )
      {
        case RECEIVE_RECEIVE:
             pPaket[ uActByte++ ] = bData;
             if( uActByte == iPaketSize )  uAction = RECEIVE_PUTAC
K;
```

```
            uTimeOut = 0xFFFF;
          break;
        }
      }
      else
      if( ser_IsWritingPossible( iSerPort ) )
      {
        switch( uAction )
        {
          case RECEIVE_SIGNAL:
               if( ser_WriteByte( iSerPort,
                                  ASCII_SYN,
                                  1,
                                  0,
                                  0 ) == SER_SUCCESS )
               {
                  uActByte = 0;
                  uAction = RECEIVE_RECEIVE;
               }
               else {printf("Erreur transmission ascii_syn ")  ;retu
rn TRANS_TRANSERROR; }
          break;
          case RECEIVE_PUTACK:
               if( ser_WriteByte( iSerPort,
                                  ASCII_ACK,
                                  1,
                                  0,
                                  0 ) == SER_SUCCESS )
                 { printf("emission ascii_ack avec succe");return
TRANS_SUCCESS;}
               else {printf("erreur transmission ascii_ack");return
TRANS_TRANSERROR;}

          default:  uTimeOut--;
        }
      }
    }
    return TRANS_TIMEOUT;
}

/*****************************************************************
*****/
/* trans_ReceiveFile : Recoit un fichier entier par l'interface
    */
/**-------------------------------------------------------------
---**/
/* Entree : iSerPort - Adresse de base de l'interface a utiliser
    */
/* Sortie : Niveau d'erreur
    */
/*****************************************************************
*****/
```

```
INT trans_ReceiveFile( INT iSerPort )
{

BYTE  recep[PAKET_DATASIZE];
FILE *fHandle;
INT iTimeOut=10, uNr, i, uDataSize;
PBYTE pFileName;

while(iTimeOut)
 {
   INT err;

   err=trans_ReceivePaket( iSerPort, recep, sizeof(recep )) ;


   if (err == TRANS_SUCCESS)
   {
   iTimeOut=4;

   if(uNr!=recep[2])
    {
      if (fHandle) fclose(fHandle);
      return PAKET_SEQUENCE;
    }
   else
    {
       uNr++;
       printf(" \r Reception de bloc %d \n",uNr);

       switch (recep[0])
          {
            case PAKET_FILEOPEN :
                 for (i=1;i<=recep[1];i++) pFileName[i-1]=recep[i+
2];
                 fHandle = fopen( ( PCHAR )pFileName, "wt" );
                 if( !fHandle )
                    {
                      fclose(fHandle);
                      return TRANS_CANTCREATFILE;
                    }
            break;

            case PAKET_FILEDATA :
                 uDataSize=recep[1];
                 for(i=0;i<=uDataSize;i++) recep[i]=recep[i+3];
                 if(fwrite(recep ,1,uDataSize,fHandle) != uDataSiz
e )
                    {
                      fclose(fHandle);
                      return TRANS_CANTWRITEINFILE;
                    }
            break;
```

```
            case PAKET_FILECLOSE :
                 fclose(fHandle);
                 return TRANS_SUCCESS;
            default:
                 return PAKET_UNKNOWN;
          }
     }
    }
    else
    iTimeOut--;
  }
return TRANS_TIMEOUT;
}


/*****************************************************************
*****/
/* trans_PError : Emission d'un message d'erreur
    */
/**-------------------------------------------------------------
---**/
/* Entree : iError - Erreur survenue
    */
/*****************************************************************
*****/
VOID trans_PError( INT iError )
{
  switch( iError )
  {
    case TRANS_SUCCESS:
         printf("Transmission OK!\n");                    break;

    case TRANS_TIMEOUT:
         printf("ERREUR: Byte-Timeout");                  break;

    case PAKET_SEQUENCE:
         printf("ERREUR: Suite de blocs interdit !\n"); break;

    case PAKET_UNKNOWN:
         printf("ERREUR : Blocs non permis !\n");         break;

    case TRANS_CANTCREATFILE:
         printf("ERREUR : Impossible d'ouvrir le fichier");break;

    case TRANS_CANTWRITEINFILE:
         printf("ERREUR : Impossible d'ecrire dans le fichier");
                                                          break;

  }

}
```

```
/*****************************************************************
*****/
/* GetArg : Retourne parametre de la ligne de commande
    */
/* Entree : argc - Argument-Count (a voir main(INT argc, CHAR *arg
v[]))*/
/*          argv - Argument-Values (a voir main(INT argc, CHAR *ar
gv[]))*/
/*          pPrefix - Parameter-Prefi
    */
/*          iType - type du parametre (_char, _int, _long, _string
,   */
/*                  _none = verifie la presence d'un parametre)
    */
/*           pVar - adresse de la variable, qui va contenir les
    */
/*                  valeurs des parametres de la ligne de commande
    */
/*           iNumElements - S'il y a plusieurs valeurs separes par
 ','*/
/*                          Enregistre les param. jusqu'a l'elemen
t   */
/*                          iNumElements dans l'array indique dans
 le */
/*                          pVar
    */
/* Sortie : Nombre de valeurs calcules ou  0 s'il n'y a pas de
    */
/*          parametre de commande
    */
/*****************************************************************
*****/
INT GetArg( INT argc, PCHAR argv[], PCHAR pPrefix, INT iType,
            PVOID pVar, INT iNumElements )
{
  INT i, j;
  INT iLen;

  PCHAR  cPtr;
  PINT   iPtr;
  PLONG  Ptr;
  PCHAR *sPtr;

  iLen = _fstrlen( pPrefix );

  for( i = 1; i < argc; i++ )
    if( _fstrnicmp( pPrefix, ( LPCHAR )argv[ i ], iLen ) == 0 )
      switch( iType )
      {
        case _int:
          iPtr = (PINT)pVar;
```

```
        for( j = 0; ( j < iNumElements ) && argv[ i ][ iLen ]; j
++ )
        {
          *iPtr = atoi( ( PCHAR )&argv[ i ][ iLen ] );
          while( argv[ i ][ iLen ] && ( argv[ i ][ iLen ] != ','
 ) )
            iLen++;
          if( argv[ i ][ iLen ] == ',' ) iLen++;
          iPtr++;
        }
        return j;

      case _char:
        cPtr = (PCHAR)pVar;
        for( j = 0; ( j < iNumElements ) && argv[ i ][ iLen ]; j
++ )
        {
          *cPtr = argv[ i ][ iLen ];
          while( argv[ i ][ iLen ] && ( argv[ i ][ iLen ] != ','
 ) )
            iLen++;
          if( argv[ i ][ iLen ] == ',' ) iLen++;
          cPtr++;
        }
        return j;

      case _long:
        Ptr = (PLONG)pVar;
        for( j = 0; ( j < iNumElements ) && argv[ i ][ iLen ]; j
++ )
        {
          *Ptr = atol( ( PCHAR )&argv[ i ][ iLen ] );
          while( argv[ i ][ iLen ] && ( argv[ i ][ iLen ] != ','
 ) )
            iLen++;
          if( argv[ i ][ iLen ] == ',' ) iLen++;
          Ptr++;
        }
        return j;

      case _string:
        sPtr = ( PCHAR *)pVar;
        for( j = 0; ( j < iNumElements ) && argv[ i ][ iLen ]; j
++ )
        {
          *sPtr = &argv[ i ][ iLen ];
          while( argv[ i ][ iLen ] && ( argv[ i ][ iLen ] != ','
 ) )
            iLen++;
          if( argv[ i ][ iLen ] == ',' ) argv[ i ][ iLen++ ] = '
\0';
          sPtr++;
```

```
        }
        return j;

      case _none:
        return TRUE;
    }
  return FALSE;
}

/*****************************************************************
*****/
/* GetNArg:Retourne le parametre de la ligne de commande sans pref
ixe */
/**---------------------------------------------------------------
----*/
/* Entree : argc - Argument-Count (a voir main(INT argc, CHAR *arg
v[]))*/
/*          argv - Argument-Values (a voir main(INT argc, CHAR *ar
gv[]))*/
/*          pPrefix - prefixe des parametres (par exemple : "-o" p
our */
/*                    output) qu'il faut ecarter des parametres
    */
/*          pString - adresse  du  pointeur de String- Arrays qui
    */
/*                    recueille les parametres
    */
/*          iNumElements - S'il y a plusieurs parametres contenus
    */
/*                         dans la ligne de commande, tout au plus
 max*/
/*                         parametres sont preserves dans iNumElem
ents*/
/* Sortie : Nombre de parametres communiques, ou 0 s'il n' y a pas
 de */
/*          parametres sur la ligne de commande
    */
/*****************************************************************
*****/
INT GetNArg( INT argc, PCHAR argv[], PCHAR pPrefix,
             PCHAR *pString, INT iNumElements )
{
  INT i, j;
  INT iLen;

  iLen = _fstrlen( pPrefix );

  for( i = 1, j = 0; ( i < argc ) && ( j < iNumElements ); i++ )
    if( _fstrnicmp( pPrefix, ( PCHAR )argv[ i ], iLen ) != 0 )
      pString[ j++ ] = argv[ i ];

  return j;
```

```
}

/*****************************************************************
*****/
/* FindString : Recherche d'un string dans un String-Array et renv
oi  */
/*              de la position du string qui a ete trouve dans l'a
rray*/
/**---------------------------------------------------------------
---**/
/* Entree : pArray  - String-Array a examiner
    */
/*          pFind   - String a rechercher
    */
/*          iNum    - nombre de Strings dans le String-Array
    */
/* Sortie : Position + 1 du string trouve dans le array ou 0
    */
/*           si le string n'est pas contenu dans l'array
    */
/*****************************************************************
*****/
INT FindString( PCHAR pArray[], PCHAR pFind, INT iNum )
{ INT i;
  for( i = 0; i < iNum; i++ )
   if( _fstricmp( pArray[ i ], pFind ) == 0 ) return i + 1;
  return 0;
}


/*****************************************************************
*****/
/* P R O G R A M M E   P R I N C I P A L
    */
/*****************************************************************
*****/

VOID main(INT argc, PCHAR argv[] )
{
  INT iSerPort, iCom, i;
  LONG  lBaud;
  PCHAR pFileName;
  BYTE  recep[PAKET_DATASIZE];
  FILE *fHandle = NULL;

  if( FindString( argv, "?", argc ) )
  {
    printf("Appel:\n" );
    printf("RECEP [Nom de fichier] [-COM:port] [-BAUD:vitesse]\n")
;
    printf("nom de fichier a creer \n");
    printf("port    = 1 ou 2       ( defaut: 1 )\n");
```

```
    printf("vitesse = 50 - 115200   ( defaut: 9600 )\n");
    exit(0);
  }

  if( GetArg( argc, argv, "-COM:", _int, &iCom, 1 ) )
  {
    if( iCom == 1 )
      iSerPort = SER_COM1;    /* Seulement COM1 et COM2 sont "stand
ard"" */
    else
    if( iCom == 2 )
      iSerPort = SER_COM2;    /* Seulement COM1 et COM2 sont "stand
ard"" */
    else
    {
      printf("Port Com incompatible !\n");
      printf("Reception abandonnee ");
      exit(0);
    }
  }
  else
    iSerPort = SER_COM1;    /* Seulement COM1 et COM2 sont "standar
d"" */

  if( !GetArg( argc, argv, "-BAUD:", _long, &lBaud, 1 ) )
      lBaud = 9600L;
  if( lBaud > SER_MAXBAUD )   /* Taux maximal de baud de l'UART : 8
450A */
  {
    printf("Vitesse trop elevee !\n");
    printf("Maximum: %ld Bd\n", SER_MAXBAUD );
    printf("Reception abandonnee ");
    exit(0);
  }


  if( !ser_Init( iSerPort, lBaud,
          SER_LCR_8BITS | SER_LCR_1STOPBIT | SER_LCR_NOPARITY ) )
  {
    printf("Il n'y a pas d'interface !\n");
    printf("Reception abandonnee ");
    exit( 0 );
  }


  if( !GetNArg( argc, argv, "-", &pFileName, 1 ) )
  {
    printf("Reception d'un fichier...\n" );
    trans_PError( trans_ReceiveFile( iSerPort ) );
  }
  else
```

```
  {
    printf("Reception de bloc de donnees dans %s ...\n",(PCHAR)pFi
leName);

   for( i=1 ; i <= 10 ; i++ )
   {
    if( trans_ReceivePaket( iSerPort , recep , sizeof(recep)) == S
ER_SUCCESS )
    {
                  fHandle = fopen( (PCHAR) pFileName, "wt" );
                  if( !fHandle )
                     {
                       printf(" Erreur ouverture fichier \n ");
                       fclose(fHandle);
                       return ;
                     }
                  if(fwrite(recep ,1,PAKET_DATASIZE,fHandle) != PAK
ET_DATASIZE )
                     {
                       printf("Erreur ecriture de la donnee dans le
 fichier\n");
                       fclose(fHandle);
                       return;
                     }

                  fclose(fHandle);
                  printf("Tres bien reception avec succe ");
                  return;
    }
    printf("%d tentatives de reception echouer \n",i);
  }
 }

}
```

```
/*****************************************************************
*****/
/*                        R E C E P . H
    */
/**---------------------------------------------------------------
---**/
/*  Tache          : Fichier include pour RECEP.CPP
    */
/**---------------------------------------------------------------
---**/
/*  Auteurs        : HIMA A. / DALI M.
    */
/*****************************************************************
*****/
#ifndef _INC_RECEP_H
#define _INC_RECEP_H

#ifndef _FP
#define _FP far
#endif

#ifndef _NP
#define _NP near
#endif

typedef unsigned char BYTE;
typedef BYTE _FP   *LPBYTE;
typedef BYTE _NP   *NPBYTE;
typedef BYTE        *PBYTE;

typedef unsigned int  UINT;
typedef UINT _FP   *LPUINT;
typedef UINT _NP   *NPUINT;
typedef UINT        *PUINT;

typedef unsigned int  WORD;
typedef WORD _FP   *LPWORD;
typedef WORD _NP   *NPWORD;
typedef WORD        *PWORD;

typedef long           LONG;
typedef LONG _FP   *LPLONG;
typedef LONG _NP   *NPLONG;
typedef LONG        *PLONG;

typedef int             INT;
typedef int  _FP    *LPINT;
typedef int  _NP    *NPINT;
typedef int          *PINT;


typedef char          CHAR;
```

```
typedef char _FP   *LPCHAR;
typedef char _NP   *NPCHAR;
typedef char        *PCHAR;

typedef void          VOID;
typedef void _FP   *LPVOID;
typedef void _NP   *NPVOID;
typedef void        *PVOID;

#define LOBYTE( w ) ( ( BYTE ) ( ( w ) & 0xFF ) )
#define HIBYTE( w ) ( ( BYTE ) ( ( ( w ) >> 8 ) & 0xFF ) )
#define MAKEWORD( h, l ) ( ( ( WORD )( h ) << 8 ) | ( WORD )( l )
)

#define LOWORD( l ) ( ( WORD )( ( l ) & 0x0000FFFF ) )
#define HIWORD( l ) ( ( WORD )( ( l ) >> 16 ) )
#define MAKELONG( h, l ) ( ( ( LONG )( h ) << 16 ) | ( LONG )( l )
 )

#define TRUE  (0 == 0)
#define FALSE (0 == 1)

#define SER_COM1 0x3F8                          /* Adresse de base C
OM1 */
#define SER_COM2 0x2F8                          /* Adresse de base C
OM2 */

                                              /* Les registres de l'U
ART */

#define SER_TXBUFFER       0x00                   /* Transmit Regis
ter */
#define SER_RXBUFFER       0x00                    /* Receive Regis
ter */
#define SER_DIVISOR_LSB    0x00                 /* Diviseur de vitesse
LSB */
#define SER_DIVISOR_MSB    0x01                 /* Diviseur de vitesse
MSB */
#define SER_IRQ_ENABLE     0x01               /* Registre Interrupt-Ena
ble */
#define SER_IRQ_ID         0x02              /* Registre d'interruption
 ID */
#define SER_FIFO           0x02                        /* Registre F
IFO */
#define SER_2FUNCTION      0x02             /* Registre Alternate-Funct
ion */
#define SER_LINE_CONTROL   0x03                   /* Contr"le des lig
nes */
#define SER_MODEM_CONTROL  0x04                    /* Contr"le du mo
dem */
#define SER_LINE_STATUS    0x05                   /* Contr"le des lig
nes */
```

```
#define SER_MODEM_STATUS    0x06                    /* Etat du mo
dem */
#define SER_SCRATCH         0x07                  /* Registre scra
tch */

        /*Registre Line-Control- - Bits (parametre de transmissi
on) */
#define SER_LCR_WORDLEN     0x03      /* Nombre de bits a transmet
tre */
#define SER_LCR_5BITS       0x00
#define SER_LCR_6BITS       0x01
#define SER_LCR_7BITS       0x02
#define SER_LCR_8BITS       0x03
#define SER_LCR_2STOPBITS   0x04            /* 2 / 1.5 bits de s
top */
#define SER_LCR_1STOPBIT    0x00                    /* 1 bit de s
top */
#define SER_LCR_NOPARITY    0x00              /* Test de parite
OFF */
#define SER_LCR_ODDPARITY   0x08                  /* Parite impa
ire */
#define SER_LCR_EVENPARITY  0x18                   /* Parite pa
ire */
#define SER_LCR_PARITYSET   0x28 /*Le bit de parite est tjs positio
nne */
#define SER_LCR_PARITYCLR   0x38    /* Le bit de parite est tjs abs
ent */
#define SER_LCR_PARITYMSK   0x38
#define SER_LCR_SETDIVISOR  0x80           /* pour acces au divis
eur */
                                                /* des vites
ses */

      /*Bits du registre de contr"le du modem (contr"le des signau
x ) */
#define SER_MCR_DTR         0x01           /* positionner signal
DTR */
#define SER_MCR_RTS         0x02           /* positionner signal
RTS */
#define SER_MCR_UNUSED      0x04
#define SER_MCR_IRQENABLED  0x08   /*communiquer IRQ au contr"leur
IRQ */
#define SER_MCR_LOOP        0x10                       /* Autot
est */

            /* Bits du registre Line-Status (Erreur de transmissi
on) */
#define SER_LSR_DATARECEIVED 0x01 /* Mot de donnees recu (5 - 8 Bi
ts) */
#define SER_LSR_OVERRUNERROR 0x02 /* Mot de donnees anterieur perd
u   */
#define SER_LSR_PARITYERROR  0x04                /* Erreur de par
```

```
ite */
#define SER_LSR_FRAMINGERROR 0x08      /* Erreur de bit de Start/S
top */
#define SER_LSR_BREAKDETECT  0x10                  /* Break detect
e   */
#define SER_LSR_ERRORMSK (SER_LSR_OVERRUNERROR|SER_LSR_PARITYERROR
|\
                          SER_LSR_FRAMINGERROR|SER_LSR_BREAKDETECT
)
#define SER_LSR_THREMPTY     0x20
#define SER_LSR_TSREMPTY     0x40

#define NOSER    0
#define INS8250  1                    /*UART  National Semiconduc
tor */
#define NS16450  2
#define NS16550A 3
#define NS16C552 4

#define SER_MAXBAUD 115200L                    /* vitesse maxim
ale */

                                               /* Messages d'err
eur */
#define SER_SUCCESS    0
#define SER_ERRSIGNALS 0x0300
#define SER_ERRTIMEOUT 0x0400

                            /* Types d'arguments de ligne de comma
nde */
#define _char   0
#define _int    1
#define _long   2
#define _string 3
#define _none   4

#define ASCII_ACK        0x06                  /* Positive Acknowle
dge */
#define ASCII_SYN        0x16                  /* Synchronus Idle
    */

#define PAKET_DATASIZE 11

#define PAKET_FILEOPEN   0                  /* Blocs aleatoires
    */
#define PAKET_FILECLOSE  1
#define PAKET_FILEDATA   2

typedef struct tagPaket
{
  BYTE bType;                  /* Type du bloc (Def. par utilisate
ur) */
```

```
  UINT uDataSize;                    /* Donnees autorises dans un bloc
    */
  UINT uNr;                                  /* nø de sequence du blo
c    */
  BYTE bData[ PAKET_DATASIZE ];                                  /* Donne
es   */
} PAKET;
typedef PAKET *PPAKET;
                                                    /* Messages d'erreur
    */
#define TRANS_SUCCESS           0
#define TRANS_TIMEOUT          -1
#define TRANS_TRANSERROR       -2
#define TRANS_CANTCREATFILE    -3
#define TRANS_CANTWRITEINFILE -4


                             /* Code d'action pour  trans_ReceivePake
t() */
#define RECEIVE_RECEIVE     0
#define RECEIVE_SIGNAL      1
#define RECEIVE_PUTACK      2

#define PAKET_SUCCESS    0
#define PAKET_SEQUENCE -10
#define PAKET_UNKNOWN  -11
#define PAKET_TIMEOUT  -12


                                                          /* Prototy
pes */
INT  ser_UARTType          ( INT iSerPort );
INT  ser_Init              ( INT iSerPort, LONG lBaudRate, BYTE bPar
ams );
INT  ser_IsWritingPossible( INT iSerPort );
INT  ser_WriteByte       ( INT iSerPort, BYTE bData, UINT uTimeOut,
                           BYTE bSigMask, BYTE bSigVals );
INT  ser_IsDataAvaiable    ( INT iSerPort );
INT  ser_ReadByte        ( INT iSerPort, PBYTE lpData, UINT uTimeOu
t,
                           BYTE   bSigMask, BYTE   bSigVals );
INT trans_SendAck( INT iSerPort );
INT trans_ReceivePaket( INT iSerPort, PBYTE pPaket, UINT iPaketSiz
e );
INT trans_ReceiveFile( INT iSerPort );
VOID trans_PError( INT iError );

INT    GetArg( INT argc, PCHAR argv[],
               PCHAR lpPrefix, INT iType,
               PVOID lpVar, INT iNumElements );
INT    GetNArg( INT argc, PCHAR argv[ ],
                PCHAR lpPrefix,
```

```
                    PCHAR lpString[], INT iNumElements );
INT     FindString( PCHAR lpArray[], PCHAR lpFind, INT iNum );


#endif
```

```
********** LES REGISTRES

ADCTL     EQU  $30          ;REGISTRE DE CONTROLE
ADR1      EQU  $31          ;LE 1er REG QUI RECOIT LA DONNEE CONVERT
IT
ADR2      EQU  $32          ;LE 2e  REG QUI RECOIT LA DONNEE CONVERT
IT
ADR3      EQU  $33          ;LE 3e  REG QUI RECOIT LA DONNEE CONVERT
IT
ADR4      EQU  $34          ;LE 4e  REG QUI RECOIT LA DONNEE CONVERT
IT
OPTION    EQU  $39          ;LE REG OPTION
PPCHG     EQU  $93          ;ACTIVE LA POMPE DE CHARGE ET L'E CLOCK
DSPPCHG   EQU  $80          ;DESACTIVE LA POMPE DE CHARGE
OMCV      EQU  $02          ;ORIGINE ET MODE DE CONVERTION

SCSR      EQU  $2E          ;REGISTRE D'ETAT
SCSR1     EQU  $2C          ;REGISTER DE CONTROLE 2
SCSR2     EQU  $2D          ;REGISTRE DE CONTROLE 2
SCDR      EQU  $2F          ;REGISTRE DE DONNEES
BAUD      EQU  $2B          ;REGISTRE DE DEBIT
VITTE     EQU  $330C        ;CHOIX DE VITESSE ET ACTIVATION DE TE/RE

DEBDAT    EQU  $40          ;ADRESSE DE DEBUT DES DONNEES
VALMAX    EQU  $BF          ;LE NOMBRE D'OCTETS A CHARGER
REGBAS    EQU  $00          ;REGISTRE DE BASE
ORIGINE   EQU  $B600        ;L'ORIGINE DE PROGRAMME

********** DEBUT DE PROGRAMME

          ORG     ORIGINE
          LDX     #REGBAS           ;ADRESSE DE BASE  DES REGISTRES
          CLR     SCSR1,X           ;SCI EN MODE 8 BITS DE DONNEES,
                                     1 BIT  START,   1   BIT   STOP
          LDD     #VITTE            ;1200 BAUDS

          STAA    BAUD,X
          STAB    SCSR2,X

          LDAA    #PPCHG            ;ACTIVATION  DE  POMPE  DECHARGE

                                     ET UTILISATION  DE  L'HORLOGE E
          STAA    OPTION,X          ;ACTIVATION DE POMPE DECHARGE
          LDY     #DEBDAT
DEBUT     LDAA    #OMCV             ;CHOIX  DE  L'ORIGINE DE CONV ET

                                     LE    MODE:4   COVNVERTIONS  DE

                                     L'ENTREE AN2
          STAA    ADCTL,X           ;CHARGEMENT DE CETTE VAL DANS
                                     LE REG DE CONTROLE
```

```
TSTCCF   LDAA    ADCTL,X           ;LECTURE DE REG DE CONTROLE
         ASLA
         BCC     TSTCCF            ;SI CCF A UN CONTINUER


         LDAA    ADR1,X            ;LECTURE  DE LA 1er DONNEE
         STAA    ,Y
         INY

         LDAA    ADR2,X            ;LECTURE  DE LA 2e  DONNEE
         STAA    ,Y
         INY

         LDAA    ADR3,X            ;LECTURE  DE LA 3e  DONNEE
         STAA    ,Y
         INY

         LDAA    ADR4,X            ;LECTURE  DE LA 4e  DONNEE
         STAA    ,Y
         INY

         CPY     #VALMAX           ;TOUTES LES DONNEES SONT RECUS ?
         BLE     DEBUT             ;OUI .. CONTINUER
         BCLR    OPTION,X DSPPCHG ;DESACTIVER POMPE DE CHARGE



         LDY     #DEBDAT           ;CHARGEMENT DE L'ADRESSE DE DEBU
T

DEB      LDAA    ,Y                ;LECTURE  DE  LA  DONNEE  PRESEN
T
                                    DANS LA RAM
         BSR     ENVOI             ;ECRITURE POSSIBLE ?
         STAA    SCDR,X            ;ECRITURE DANS LE REG DE TRANS
         INY
         CPY     #VALMAX           ;TOUTES LES DONNEES SONT EMISES
         BNE     DEB               ;SI OUI CONTINUER

         RTS

********* CE SUBROUTINE TESTE SI L'EMISSION EST POSSIBLE
ENVOI    LDAB    SCSR,X
         ASLB
         BCC     ENVOI
         RTS
```