Page de présentation

Département Génie électrique
et
informatique industrielle

IUT de l'université de Bordeaux I
33405 Talence

École Nationale Supérieure de radioélectricité
de Bordeaux

1, avenue du Docteur Albert
Schweitzer
Domaine Universitaire - B.P. 99
33402 TALENCE Cedex

Rapport de stage

Mise en place des outils de développement
68HC11
réalisation de cartes 68HC11

Yannick BENABEN
Groupe B3

Année 1998-99

Maître de stage
M. Patrice KADIONIK
Enseignant responsable
M. Christian CAZAUBON

Table des matières

Remerciements

Je tiens à remercier M. Kadionik pour m'avoir accueilli au sein de l'ENSERB et pour m'avoir fourni toute l'aide technique dont j'avais besoin. J'adresse également mes sincères remerciements à l'ensemble du personnel de l'ENSERB ainsi qu'à toutes les personnes m'ayant aidé à effectuer ce stage dans d'excellentes conditions.

Avant-propos

Ce rapport de stage est avant tout destiné à tous ceux qui désirent en savoir plus sur le développement d'applications 68HC11. Dans ce document nous verrons quelques moyens de développement qui sont l'assembleur, le langage C, l'utilisation de bibliothèques et la programmation multitâche avec un noyau temps réel. Chaque type de programmation est détaillé, les fichiers utilisés lors de la compilation et l'édition de liens sont détaillés dans le rapport et les listings commentés des applications sont donnés en annexe. Une partie de la documentation et certains logiciels sont fournis par les constructeurs du matériel utilisé et sont accessibles depuis leur site Internet respectif. La bibliographie retrace l'ensemble des documents papier et informatique utilisés pour l'élaboration des programmes ainsi que les adresses Internet où l'on peut se procurer une source considérable et fiable de documents fournis par les constructeurs.

Présentation de l'ENSERB

L'École Nationale Supérieure de Radioélectricité de Bordeaux (ENSERB) est située au sein de la Faculté des Sciences de Bordeaux. Créée en 1920, elle est habilitée à délivrer le titre d'ingénieur diplômé en 1934. Elle prend le statut d'école Nationale Supérieure d'Ingénieur en 1975.

L'ENSERB possède 2 pôles de formation :

- l'électronique avec les options Microélectronique, Automatique Robotique, Traitement du Signal, Informatique Industrielle, Technologie de l'Image et de la Communication
- l'informatique avec les options Algorithmique et Applications Massivement parallèles, Génie Logiciel, Réseaux et systèmes répartis, Technologies de l'Image et de la Communication.

L'ENSERB s'appuie sur des laboratoires de recherche de niveau international tels que le laboratoire de microélectronique IXL, le Laboratoire Bordelais de Recherche Informatique (LaBRI), le Laboratoire d'Automatique et de Productique (LAP) et l'Équipe Signal et Image : ESI, membre du GDR 'ISIS' du C.N.R.S.

Direction ENSERB
1, avenue du Docteur Albert Schweitzer
Domaine Universitaire - B.P. 99
33402 TALENCE Cedex
tél.: +33 05 56 84 65 00 - Fax : +33 05 56 37 20 23

I. Présentation du sujet du stage et du matériel utilisé.

Mise en place d'un atelier de développement intégré basé sur l'utilisation du microcontrôleur 68HC11.

Descriptif : Le sujet du stage est bâti autour du microcontrôleur 68HC11 de chez Motorola. Le stage a pour but de mettre en place un atelier de développement utilisant le microcontrôleur 68HC11 pour les projets de fin d'études àl'ENSERB. Lors de la mise en place de l'atelier, il faudra évaluer les différents équipements et logiciels (émulateur ICE, débugger, compilateur C). Une carte 68HC11 générique sera réalisée pour permettre aux étudiants d'avoir une base de travail afin d'accélérer leur réalisation finale.

Mots clefs : Émulateur / débuggers / moniteur 68HC11, compilateur C Cosmic, noyau multitâche temps réel µC/OS…

Le stage sera donc divisé en 2 parties : une partie hardware (réalisation de cartes 68HC11) et une partie software (réalisations de programmes de tests et noyau temps réel).

Diagramme de GANT :

Opérations à réaliser	Début de la réalisation	Fin de la Réalisation	N° de la semaine								
			18	19	20	21	22	23	24	25	26
Prise connaissance école et du sujet	26/04/99	28/04/99	■								
Etude de la carte mère	28/04/99	28/04/99	■								
Conception sous MENTOR GRAPHICS	28/04/99	30/04/99	■								
Réalisation de la plaquette carte mère	3/05/99	3/05/99		■							
Conception de la carte d'entrée - sortie	4/05/99	5/05/99		■							
Réalisation de la plaquette affichage	6/05/99	7/05/99		■							
Etude de la cartographie mémoire	7/05/99	10/05/99		■	■						
Etude des manuels HC11 & PCBUG11	10/05/99	10/05/99			■						
Conception + débuggage programmes tests	10/05/99	12/05/99			■						
Programmation en C de Cosmic Software	17/05/99	19/05/99				■					
Création d'une bibliothèque en C	18/05/99	19/05/99				■					
Etude et mise en place du moniteur	20/05/99	28/05/99				■	■				
Etude livre sur le temps réel et µC/OS II	31/05/99	31/05/99						■			
Adaptation µC/OS pour 68HC11Ax	1/06/99	2/06/99						■			
Mise en œuvre des sémaphores	03/06/99	08/06/99						■	■		
Etude et exemples de programmes µC/OS II	07/06/99	11/06/99							■		
Débuggage tâches µC/OS II	11/06/99	16/06/99							■	■	
Débuggage programmes & récapitulatif	16/06/99	18/06/99								■	
Rédaction rapport & page Web	16/06/99	25/06/99								■	■

II. Mise à jour de la carte mère.

A. *Caractéristiques du microcontrôleur 68HC11.*

Le processeur employé est un 68HC11A1 dont les caractéristiques essentielles sont résumées dans le schéma suivant :

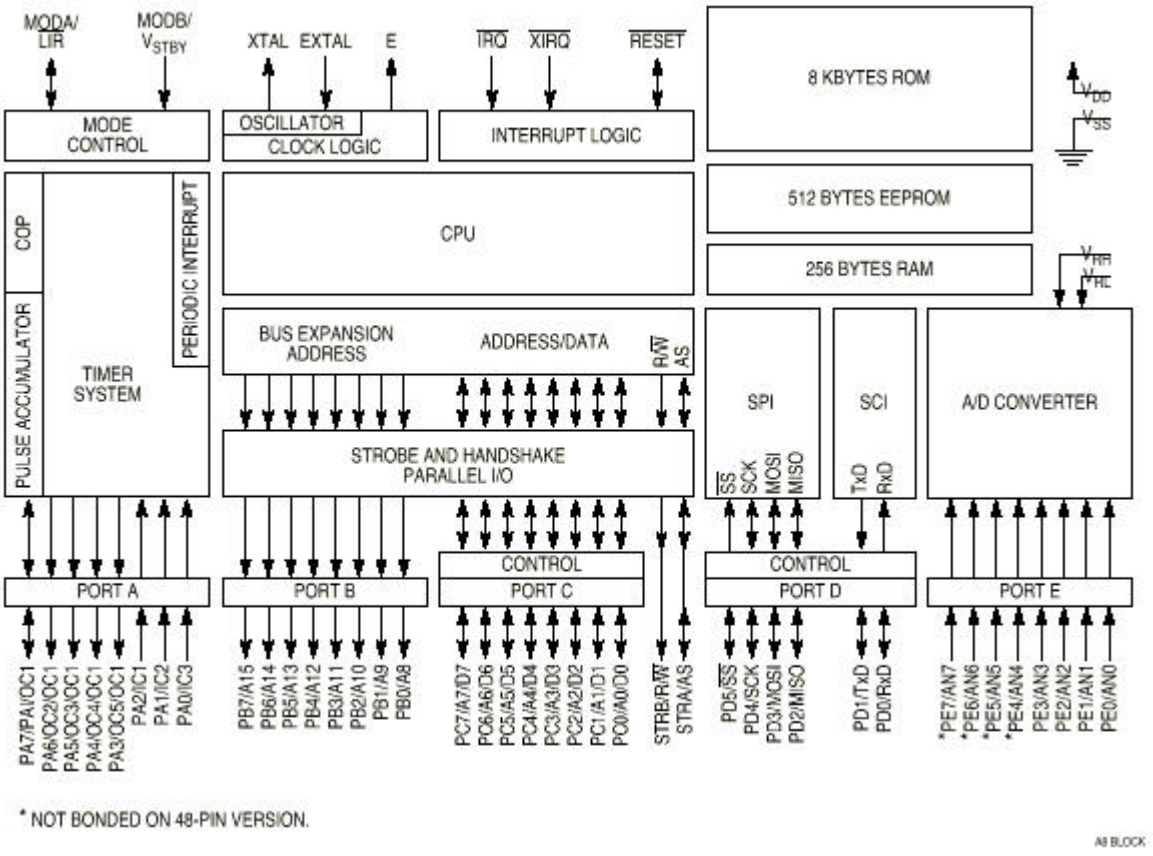

Figure II-1 - Architecture interne du 68HC11.

On peut souligner les éléments suivants :

- microcontrôleur technologie HCMOS
- 8Ko de ROM
- 512 octets de EEPROM
- 256 octets de RAM
- un timer 16 bits (3 entrées de captures, 5 sorties de comparaison)
- 2 accumulateurs 8 bits
- une liaison série asynchrone (SCI)
- une liaison série synchrone (SPI)
- un convertisseur analogique - numérique 8 bits, 8 entrées
- circuit d'interruption temps réel
- circuit oscillant externe (quartz 8 MHz dans notre application)

Les modes de fonctionnement du 68HC11 sont résumés dans le tableau suivant :

MODB	MODA	Mode sélectionné
1	0	Single Chip (Mode 0)
1	1	Expanded multiplexed (Mode 1)
0	0	Special bootstrap
0	1	Special test

La carte mère réalisée durant le stage permet l'utilisation du microcontrôleur 68HC11 dans tous les modes grâce à la présence de 2 jumpers qui correspondent à MODA et MODB vus précédemment.

B. Schéma électrique de la carte mère.

Voici le schéma électrique de la carte mère conçue par M. KADIONIK (voir en annexe le schéma imprimé en pleine page) :

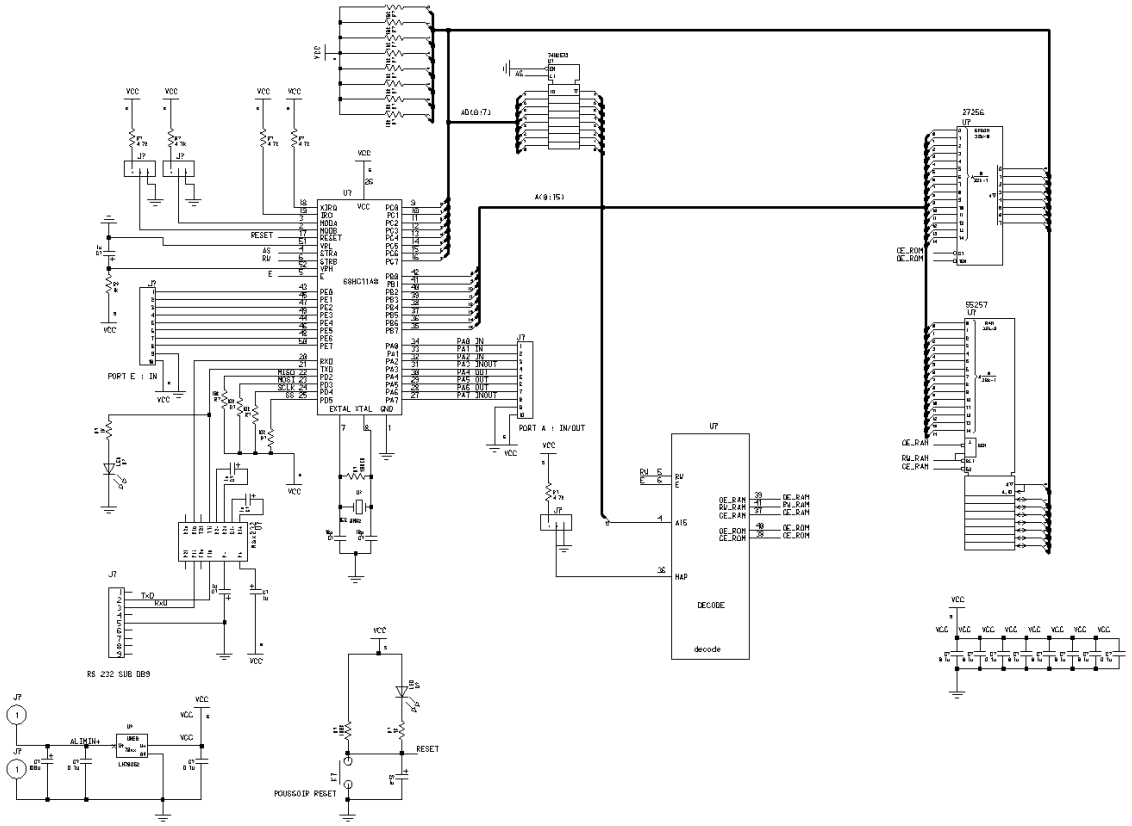

Figure II-2 – schéma électrique de la carte mère.

Si le microcontrôleur travaille en mode étendu, l'utilisateur dispose des ressources externes qui sont une RAM et une ROM externes de 32Ko. Des ports du 68HC11 sont alors attribués aux fonctions d'interfaçage des composants extérieurs :

➔ le PORT C est multiplexé : selon la valeur du signal AS, le PORT C est soit le bus bidirectionnel des données (D0…D7) soit le poids faible du bus d'adressage 16 bits (A0…A7). Le démultiplexage est ici confié au circuit 74LS573.

➔ le PORT B est le poids fort du bus d'adressage (A8…A15).

L'adressage sur 16 bits permet donc d'adresser directement 64Ko.

C. Le décodage d'adresse.

Le décodage d'adresse est réalisé par un circuit programmable d'ALTERA dont le schéma symbolique est le suivant :

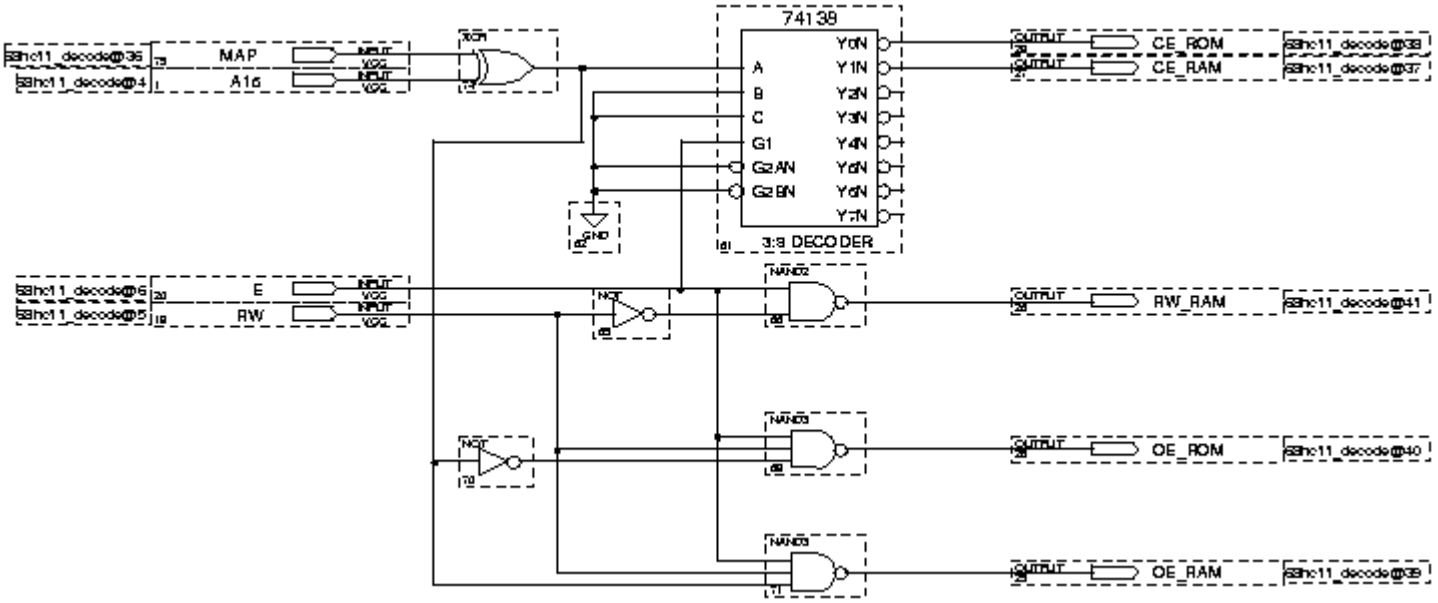

Figure II-3 – schéma symbolique du décodeur d'adresse.

On peut remarquer que le schéma comporte un décodeur d'adresse 74LS138 ainsi qu'un circuit logique destiné à activer les Chip Select des boîtiers mémoires et la broche R/W de la RAM. On remarque également qu'une entrée supplémentaire nommée « MAP » reliée à une porte XOR a été rajoutée afin de remplacer la mémoire RAM par la mémoire ROM et vice-versa selon le programme à implanter.

D. Cartographie mémoire de la carte mère.

La cartographie mémoire du 68HC11Ax est la suivante :

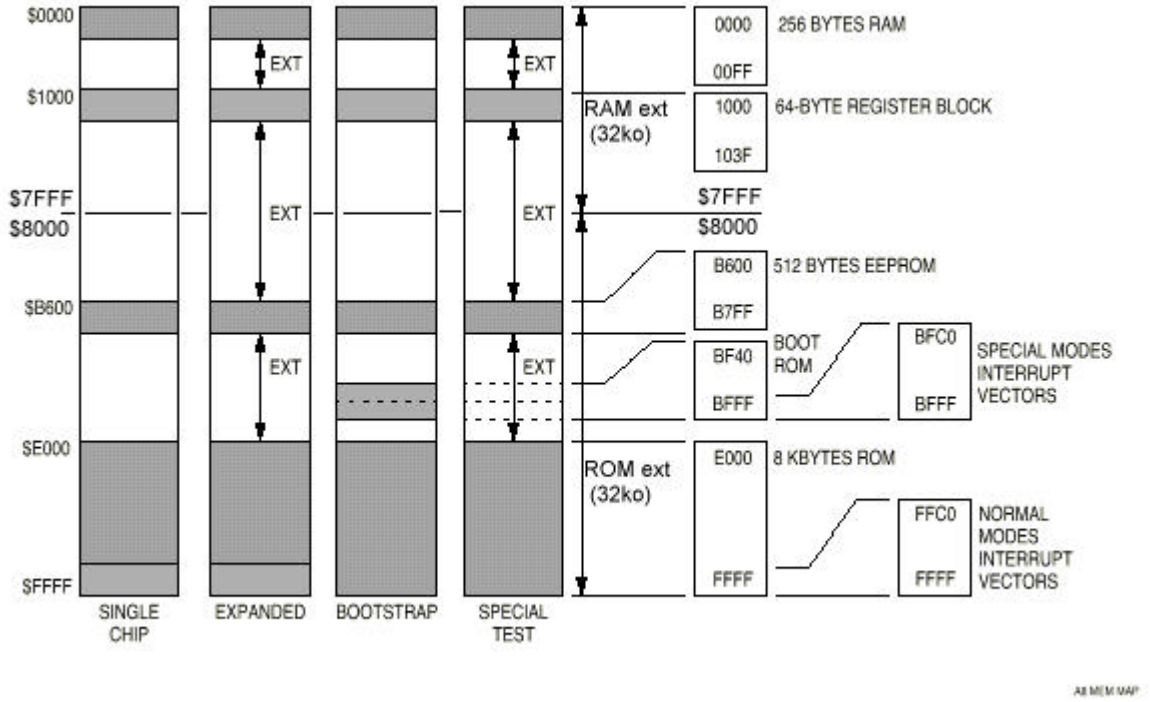

Figure II-4 – cartographie mémoire du 68HC11 dans tous les modes de fonctionnement.

Remarque : *il est important de noter que les accès aux ressources internes (RAM interne, EEPROM, registres...) sont prioritaires vis-à-vis des mémoires externes. De plus, la configuration d'origine du bit IRV du registre HPRIO permet de rendre "invisibles" les données internes vis - à - vis des données externes. En modes normaux, cette fonction permet donc de se passer d'un décodage logique complexe protégeant ainsi les circuits d'éventuels conflits accidentels sur le bus de données. (Cf. M68HC11 REFERENCE MANUAL p 3-3 chapitre 3.1.2 "Mode Control Bits in HPRIO register").*

Le 68HC11A1 est ici utilisé de 2 manières :

- avec PCBUG11 on travaille en mode bootstrap puis en mode special test. Ce n'est qu'après avoir débuggué le code du programme que nous configurons le processeur en mode étendu lui conférant ainsi une autonomie totale.
- avec le moniteur BUFFALO (Bit User's Fast Friendly Aid to Logical Operation) on travaille en mode étendu puisque le moniteur est programmé dans une EPROM externe. Le moniteur permet alors le débuggage grâce à un simple terminal ou un émulateur de terminal sur ordinateur.

Nous verrons plus en détail comment nous utilisons ces 2 logiciels de débuggage au chapitre "Conception de programmes de tests"

E. Conception sous MENTOR Graphics et gravure de la plaque.

La conception de la carte mère a été effectuée sous MENTOR GRAPHICS. Les modifications apportées ont été minimes : redisposition des jumpers pour un accès plus pratique, numérotation des ports d'extension (PORTA et PORTE), changement de géométrie de certains composants autour du MAX232.

Le logiciel *MENTOR GRAPHICS* se décompose en plusieurs sous - programmes accessibles depuis dmgr.

- La saisie du schéma est réalisée sous *Design Architect*. Le schéma de la carte mère était déjà saisi sous Design Architect par M Kadionik, je n'ai eu qu'à redisposer les composants et router la carte.
- L'édition des géométries se fait sous *Librarian* mais cette étape n'est pas obligatoire si l'on dispose de toutes les géométries nécessaires au schéma.
- Le lien entre le symbole électrique et la forme géométrique est réalisé par *Package* (possible également en utilisant Board Architect). Le choix des composants physiques est essentiellement réalisé grâce aux fonctions "Resolve geometrie", "build" et "check build" en plus des configurations préliminaires détaillées dans le manuel de MENTOR GRAPHICS (cf. bibliographie).
- *Layout* permet de placer les composants en mode manuel ou automatique (déconseillé) ainsi que le routage automatique (vivement conseillé pour une telle carte en double face). La méthode de routage est la suivante : on configure le routage sur les 2 faces et "by random choice" avec 4 passes en mode "pattern", 10 en mode "automatic" (le plus efficace) et 6 passes en mode manufacturing. On peut par la suite protéger une face (cuivre en général) grâce à la fonction "protect area" et configurer le routage en mode "by lengh" : on diminue alors le nombre de vias sur le montage...

- Lorsque le logiciel a correctement routé (unroutes(0)), on peut réaliser un fichier de perçage grâce au logiciel *Fablink*. Le fichier de perçage ainsi créé est envoyé à un robot muni d'une perceuse.

Remarque : *des captures d'écran des différentes étapes sont disponibles en annexe ainsi que les fichiers générés à chaque étape (schéma électrique, schéma d'implantation, typons double face, nomenclature et décodage d'adresse).*

La réalisation de la plaque doit être faite de la manière suivante :

- perçage automatique de la plaque
- mise en place du calque (typon) en face des trous
- insolation aux UV de 3 minutes en double face
- révélation rapide du circuit (4 secs !)
- gravure de la plaquette (vitesse environ 3mm/s)
- étamage (pellicule d'étain sur les pistes pour faciliter la soudure et éviter l'altération du cuivre avec le temps)

La carte mère est ensuite testée seule avec le logiciel PCBUG11 (voir chapitre conception de programmes de tests)

III. Conception d'une carte d'entrées - sorties pour 68HC11.

A. *Conception sous MENTOR GRAPHICS.*

La conception de cette carte a été faite après l'étude de la carte mère précédemment décrite. En effet le 68HC11Ax en mode étendu ne possède que 2 ports libres : le PORT A et le PORT E. Le PORT E ne pose pas de problème particulier puisque tous les bits du PORT E sont accessibles en entrée. Le PORT A nécessite une attention plus particulière notamment au niveau du registre PACTL qui configure le TIMER partagé avec le PORT A. Nous choisissons alors d'utiliser toutes les broches possibles en sorties. La carte d'affichage sera configurée de la manière suivante :

- PORT E : 8 bits en entrées
- PORT A : entrées = PA0, PA1, PA2
  sorties = PA3, PA4, PA5, PA6, PA7

Le schéma électrique est donné ci-dessous : on retrouve les 2 connecteurs du PORT A et PORT E. Le PORT E est interfacé avec un boîtier DIL de 8 micro-switchs dont les sorties sont reliées à des résistances de pull down de 10Kohms. Le PORT A est partagé en 2 : Les 3 bits de poids faible sont reliés de la même manière que le PORT E grâce à un boîtier micro-switch de 4 interrupteurs, les 5 bits de poids fort sont reliés à un circuit buffer 74HC245 qui alimente 5 LEDs de visualisation.

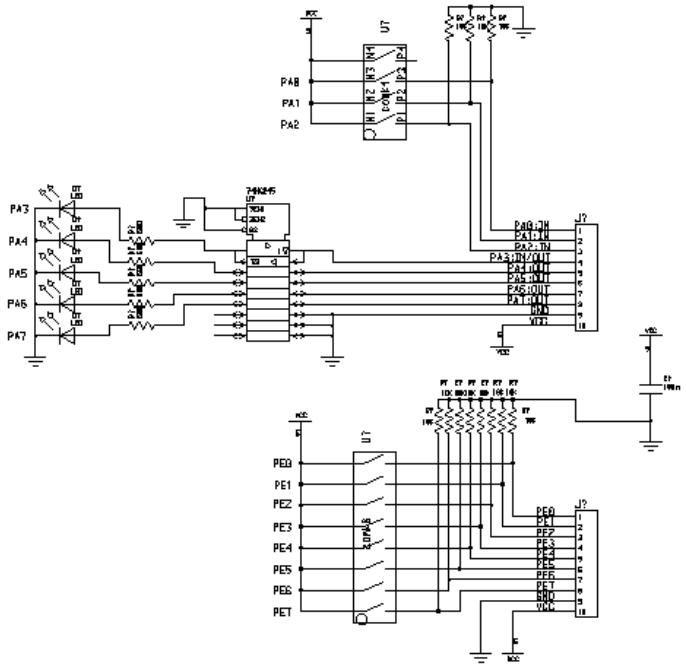

Figure III-1 – schéma électrique de la carte d'affichage.

La conception de la carte sous MENTOR GRAPHICS est identique à celle de la carte mère : on réalise le montage en double face.

Remarque : *des captures d'écran des différentes étapes sont disponibles en annexe ainsi que les fichiers générés à chaque étape (schéma électrique, schéma d'implantation, typons double face).*

IV.Conception de programmes de tests

A. *Fonctionnement du 68HC11 en mode bootstrap avec PCBUG11 3.42.*

Avant d'aborder la programmation du 68HC11 nous allons voir un peu plus en détail le fonctionnement du logiciel PCBUG11.

PCBUG11 est un petit logiciel qui permet de télécharger et débugger des programmes que l'utilisateur désire faire exécuter sur le 68HC11. Mais comment procède - t - on pour télécharger son programme dans les mémoires du 68HC11 ?

Tout d'abord, lors de l'écriture du programme, une directive de compilation (ORG + adresse en hexa) permet de définir l'adresse de départ du programme dans la mémoire. Il faut donc placer son programme dans une zone libre et inscriptible, c'est le cas de la RAM et de l'EEPROM. L'EEPROM nécessite un protocole de programmation un peu plus complexe que la RAM mais l'avantage est que le code du programme sera conservé même si l'alimentation du microcontrôleur est coupée.

Pour la suite, il faut aller examiner le fonctionnement du microcontrôleur. En mode bootstrap, il apparaît une "boot ROM " en $BF40-$BFFF, zone de mémoire morte à laquelle le processeur saute après le Reset. Cette mémoire de boot, présente dans la cartographie uniquement en mode bootstrap et special test, contient une petite routine qui autorise le téléchargement de 256 octets depuis le port série vers la RAM à partir de l'adresse $0000. En effet, ce programme en boot ROM configure la liaison série à 9600 bps, 1 bit de start, 8 bits de données, 1 bit de stop (pas de contrôle ni de parité ni de flux). Le programme de boot offre également la possibilité de télécharger les 256 octets à une vitesse de 1200 bps par l'intermédiaire d'une routine interne de la ROM appelée autobaud qui détecte la vitesse du téléchargement grâce à la réception du caractère $FF. C'est donc à ce moment là que le logiciel PCBUG11 envoie un petit programme de 192 octets environ qui est appelé TALKER. Comme son nom l'indique, ce petit programme va permettre de communiquer avec l'ordinateur hôte et ainsi d'exécuter les commandes de PCBUG11.

Le logiciel PCBUG11 se présente de la manière suivante :

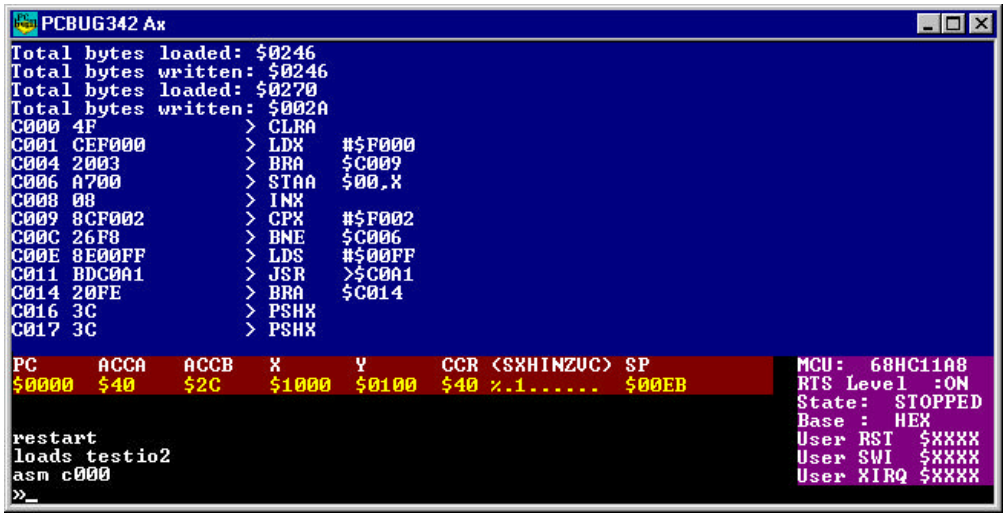

Figure IV-1 – le logiciel PCBUG11 en cours d'utilisation.

La zone supérieure est la zone de dialogue où s'affiche le résultat des opérations lancées par l'utilisateur. La zone centrale au milieu montre l'état des registres du 68HC11 et à

droite le type de processeur ainsi que le mode de fonctionnement de PCBUG11 (Running, Stopped, Trace). La partie inférieure est réservée à l'utilisateur pour rentrer les commandes de PCBUG11.

Voici une brève description des commandes essentielles de PCBUG11 :

- ASM addr : permet le désassemblage / assemblage en ligne du code téléchargé en mémoire à l'adresse addr.
- BF addr1 addr2 byte|word : permet de remplir un bloc de mémoire commençant à addr1 et terminant par addr2 avec la valeur byte / word.
- BR addr [macroname] : Cette fonction permet de placer des points d'arrêt permettant ainsi à l'utilisateur de débugger son programme. Cette fonction est utilisable seulement si on a répondu No à la question "Do you wish use the XIRQ interrupt ?" Au lancement de PCBUG11. Le programme doit être également implanté dans une zone de mémoire accessible en écriture puisque PCBUG11 place une instruction SWI (interruption logicielle) à l'adresse du point d'arrêt décalant ainsi tout le reste du programme en mémoire. Le paramètre [macroname] est facultatif, il permet d'exécuter une macro préalablement chargée lors d'un point d'arrêt.
- EEPROM 0|addr1 addr2 : permet de supprimer une plage EEPROM (paramètre 0) ou bien de la configurer en spécifiant l'adresse de départ et d'arrivée.
- EEPROM ERASE [bulk] : permet d'effacer partiellement ou dans son intégralité la EEPROM dont l'étendue est définie grâce à EEPROM addr1 addr2.
- G addr : exécute le programme à l'adresse addr.
- HELP : affiche toutes les fonctions de PCBUG11 ainsi qu'un commentaire.
- LOADS filename : permet de charger un fichier S19 en mémoire.
- MD startaddr [endaddr] : visualise la plage de mémoire s'étendant de startaddr à endaddr.
- MM addr : modifie le contenu de la case mémoire à l'adresse addr.
- NOBR addr : supprime le point d'arrêt placé à l'adresse addr.
- QUIT Y : quitte le logiciel PCBUG11.
- RD : rafraîchit l'affichage des registres dans la fenêtre centrale.
- RESTART : permet de relancer PCBUG11 si une erreur de communication est apparue.

La liste complète de toutes les instructions ainsi que leur mode d'emploi est disponible dans le manuel de PCBUG11 (cf. bibliographie).

B. Chargement d'un programme avec PCBUG11.

Lorsque l'utilisateur désire charger un programme en EEPROM il doit procéder comme suit :

- effectuer un RESET sur la carte cible.
- lancer PCBUG11.

***Remarque** : le fonctionnement de PCBUG11 sous DOS ne nécessite pas de configuration particulière de l'ordinateur . En revanche l'utilisation sous Windows 95 / 98 nécessite les configurations suivantes : **pas de contrôle de flux matériel** (pour le port de communication COM1 ou COM2) et **sensibilité d'attente basse** (dans l'onglet propriété « divers » du programme PCBUG11). Il est nécessaire d'avoir le patch pour PCBUG11 qui*

permet de faire fonctionner le logiciel dans le cas des processeurs de vitesse supérieure à 200MHz – sinon génération de l'erreur « runtime error ».

➔ taper EEPROM $B600 $B7FF si l'EEPROM est située de l'adresse $B600 à $B7FF.

➔ taper EEPROM ERASE BULK qui permet d'effacer l'EEPROM. Nous pouvons ainsi vérifier si l'opération a été réussie en tapant md $B600 $B7FF : on ne doit voir que des octets $FF.

***Remarque importante** : si la programmation de la mémoire EEPROM n'est pas correcte, il est possible soit qu'elle ne se trouve pas aux adresses $B600 à $B7FF soit que le registre de protection en écriture BPROT (présent sur les 68HC11Ex, F1...) ait été configuré pour protéger la EEPROM. Pour résoudre ce problème, il faut, après avoir lancé PCBUG11, taper mm $1035 00 où $1035 est l'adresse du registre BPROT.*

Si la EEPROM n'est pas présente dans la cartographie mémoire, c'est que le "CONFIG register" est mal configuré : attention, ce registre se programme comme une case de mémoire EEPROM... voir le manuel "M68HC11 REFERENCE MANUAL", page 3-5, paragraphe 3.2.2 The config register. Le bit concerné est nommé EEON et permet d'activer ou non la EEPROM.

➔ taper ensuite loads filename où filename est le nom du fichier compilé (sans l'extension) au format S19 de Motorola (S19-record file) qui se trouve alors dans le même répertoire que PCBUG11. Les fichiers S19 sont créés par exemple par le compilateur assembleur (asmhc11.exe) ou bien par un compilateur croisé C (utilisation d'un fichier **go.bat** fourni en annexe).

➔ il ne reste plus qu'à vérifier que le programme est bien implanté en mémoire grâce à l'instruction asm $B600 si le programme démarre en $B600.

➔ si le programme est bien chargé (pas de message d'erreurs) alors l'exécution du programme se fait par G $B600 et l'arrêt par S. Lorsqu'on stoppe le programme, le 68HC11 revient au TALKER.

Nous verrons ensuite que le logiciel PCBUG11 est pratique pour la mise au point de programmes mais comporte néanmoins des inconvénients tels que l'occupation importante de la RAM interne ainsi que la nécessité d'être en mode bootstrap.

La note d'application "AS-67 application snapshot" montre comment on peut accéder aux ressources externes tout en utilisant PCBUG11. En effet, PCBUG11 ne marche qu'en mode «bootstrap » en revanche il est possible de basculer en mode « special test » sous PCBUG11 en écrivant $E5 à l'adresse $103C (adresse du registre HPRIO). On positionne ainsi les bits SMOD et MODA à 1 (special test) et le bit IRV reste à 0 évitant ainsi les conflits sur le bus de donnée comme nous l'avons vu au chapitre "Mise à jour de la carte mère".

A partir de ce moment, il est possible d'accéder aux ressources externes du 68HC11 (cf. cartographie mémoire figure II-4) et donc de télécharger des programmes en RAM externes depuis PCBUG11.

C. Programmation en assembleur Motorola.

Le premier système de développement étant vu, nous abordons la partie programmation proprement dite. Le jeu d'instructions est disponible dans le manuel "M68HC11 REFERENCE MANUAL - APPENDIX A". Le premier programme conçu

permet de tester le fonctionnement correct de la carte d'affichage. Le programme a directement été écrit en langage assembleur et permet selon l'état de PA0 de recopier le PORTE sur le PORTA (PA0=1) ou bien de faire un "chenillard" sur les 5 bits disponibles du PORTA. Le fichier source est disponible en annexe sous le nom de **testio.** compilé avec asmhc11.exe.

D. Programmation en langage C avec Cosmic Software.

1. Développement de programmes en C.

Après le débuggage relativement simple de ce programme, le programme testio.asc a été réécrit en langage C pour HC11. Le système de développement utilise toujours PCBUG11 mais le compilateur provient de COSMIC Software. Ce langage de haut niveau permet d'écrire des programmes de manière plus intuitive qu'en assembleur. Une version du programme de test est disponible en annexe sous le nom de **testio.c**.

On remarque que le programme généré à partir du C est beaucoup plus gros que celui généré à partir de l'assembleur cependant le compilateur C possède des paramètres permettant l'optimisation des programmes en taille ou en vitesse.

2. Création d'une bibliothèque de fonctions en C.

Le programme précédent est amélioré en incluant une bibliothèque de fonctions ainsi qu'une gestion du port série en langage C.

Une bibliothèque de fonction a donc permis une programmation plus claire et mieux structurée. La source de la bibliothèque est donnée en annexe sous le nom de **hc11lib.c**. Des fonctions supplémentaires ont été créées permettant ainsi d'allumer ou d'éteindre une seule LED ou bien de recevoir et d'émettre un caractère sur la liaison série du 68HC11. Ces fonctions sont illustrées dans le programme intitulé **testio2.c**.

La compilation des programmes en C mérite une attention particulière : en effet la compilation en C exige le lien avec des programmes externes tels que crts.s, vector.c et hc11lib.c qui sont respectivement le fichier de démarrage, la table des vecteurs d'interruption et la bibliothèque de fonctions utilisées par le programme principal. La génération d'un fichier S19 se fait de la manière suivante :

- compilation des sources avec le compilateur cx6811.exe.
- édition des liens avec clnk.exe.

Lors de l'édition des liens, un fichier de configuration d'édition de liens .LKF permet de spécifier les adresses des différentes sections du programme (code programme, données, pile, vecteurs…).

L'automatisation du processus est faite grâce au fichier **go.bat** que l'on trouvera en annexe.

Le logiciel PCBUG11 permet le téléchargement des fichiers S19 en mémoire de manière simple. En revanche PCBUG11 a quelques inconvénients tels que l'occupation d'une grande partie de la RAM interne ainsi que l'obligation de démarrer le 68HC11 en mode bootstrap. De plus l'utilisation de la liaison série par le programme utilisateur rompt la liaison entre le TALKER et PCBUG11. Nous allons voir dans la partie "Mise en place du moniteur BUFFALO" comment on améliore le développement de programmes plus complexes.

V. Mise en place du moniteur Buffalo rev 3.4 sur la carte mère.

A. *Qu'est – ce qu'un moniteur ?*

Nous avons vu que le logiciel PCBUG11 télécharge un petit programme de 192 octets environ en mémoire RAM, un moniteur est un programme beaucoup plus gros (8 Ko environ) que l'on programme dans une EPROM externe et qui permet donc de remplacer le TALKER de PCBUG11. Les avantages sont multiples puisque le moniteur occupe très peu de RAM (uniquement pour quelques variables), il permet de débugguer le programme utilisateur grâce à des commandes intégrées que nous verrons plus loin et le moniteur dialogue avec l'utilisateur par l'intermédiaire d'un simple terminal (l'émulateur « hyperterminal » par exemple).

L'exécution du moniteur se fait lors du RESET en mode ETENDU et rend libre l'utilisation du port série après le lancement du programme utilisateur. Pour reprendre la main, il suffit de réinitialiser le microcontrôleur (cette manipulation est expliquée au chapitre "Utilisation de l'émulateur CT68HC11 et du moniteur BUFFALO").

B. *Cartographie mémoire avec moniteur BUFFALO 3.4 pour 68HC11 séries E et A.*

Voici la cartographie mémoire de la carte mère avec moniteur intégré :

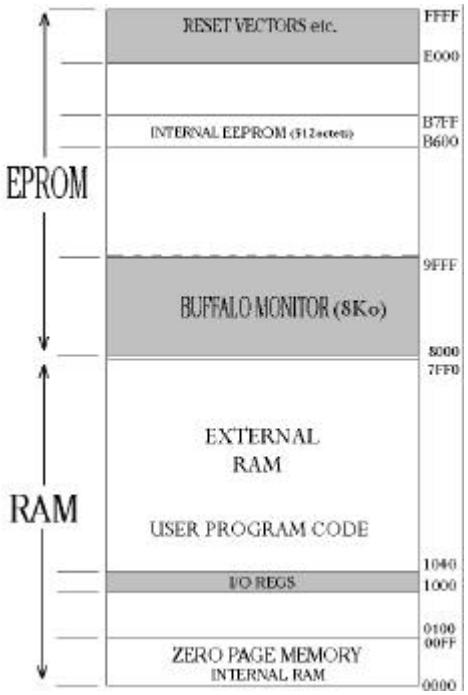

Figure V-1 – cartographie mémoire en mode étendu avec le moniteur BUFFALO 3.4.

Elle est identique à celle que nous avons déjà vue figure II-4, seulement une zone de 8Ko est réservée au code du moniteur. En revanche l'EPROM utilisée est de 32Ko permettant ainsi de mettre d'autres programmes en EPROM. L'initialisation de la table de vecteur doit se faire lors de la programmation de l'EPROM. Nous verrons au chapitre 9 « Mise en place d'un noyau multitâche temps réel µC/OS » que la table vecteur doit initialiser les vecteurs Timer Output Compare 1 ($FFE8-E9), SWI ($FFF6-F7) et RESET ($FFFE-FF).

C. Utilisation du moniteur BUFFALO 3.4.

Les commandes du moniteur BUFFALO 3.4 sont similaires à celles utilisées par PCBUG11 et sont accessibles dès la mise en route du moniteur en tapant entrée ou help : la capture d'écran suivante montre l'ensemble des commandes dont le rôle est assez intuitif.

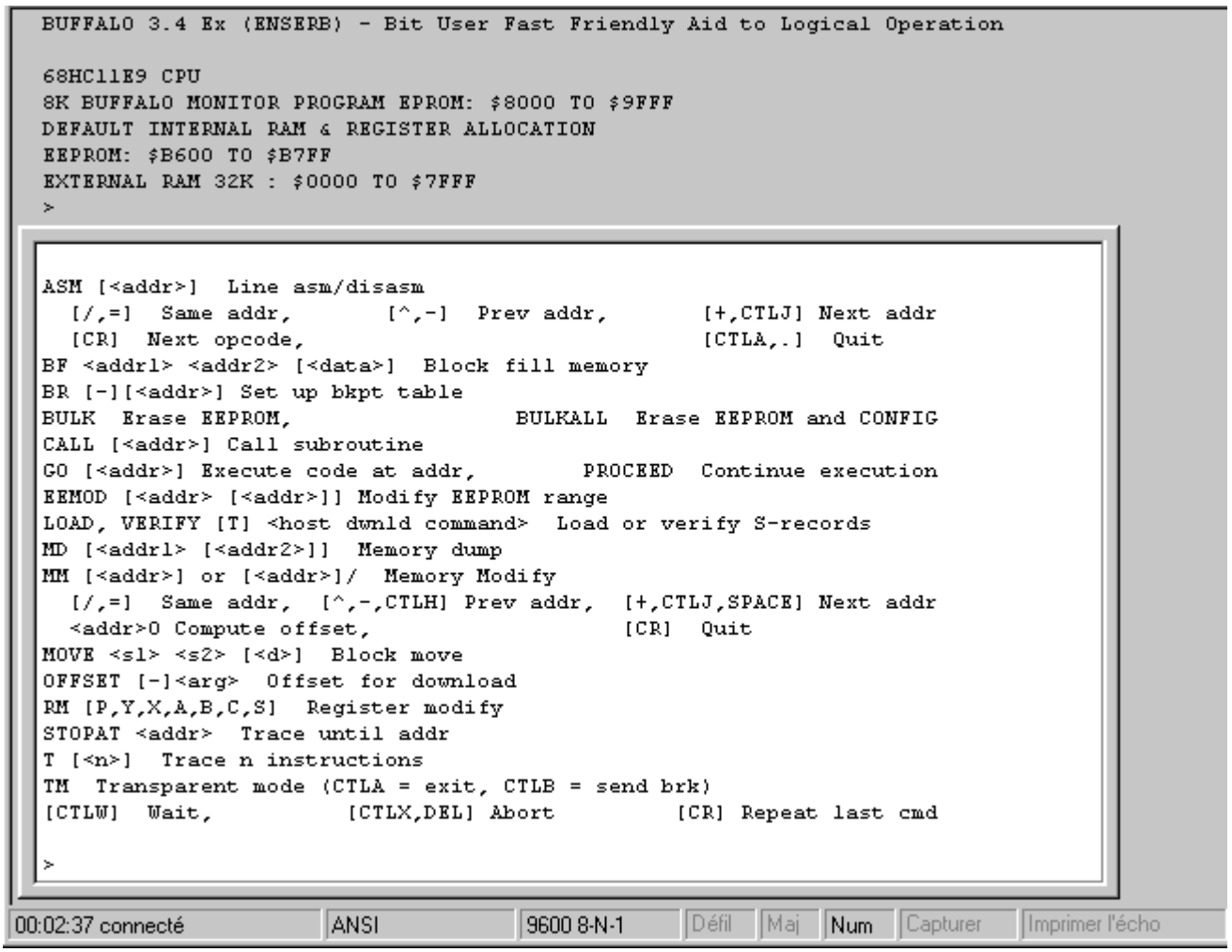

Figure V-2 – le moniteur BUFFALO en cours d'utilisation.

La source du moniteur BUFFALO (fichier buf34.asm de 134Ko environ) est disponible chez MOTOROLA pour différentes versions du 68HC11. (voir bibliographie). La configuration du moniteur consiste à spécifier le début de la zone où le moniteur sera implanté (EPROM en général), le début de la RAM interne, la EEPROM, les registres... puis on compile le code grâce au logiciel as11.exe.

La mise en place du moniteur sur la carte mère a été faite d'une manière différente de celle avec PCBUG11. D'une part le moniteur a été placé en RAM afin de faciliter la mise au point et d'autre part le 68HC11 a été remplacé par l'émulateur "CT68HC11 emulator" de l'entreprise ASHLING Microsystems Ltd. Nous allons voir comment fonctionnent l'émulateur et le logiciel Pathfinder et nous verrons alors par la suite comment utiliser le moniteur BUFFALO.

VI. Utilisation de l'émulateur CT68HC11 de ASHLING.

A. *Présentation de l'émulateur CT68HC11.*

Le logiciel embarqué sur l'émulateur CT68HC11 est conçu pour émuler la série de microcontrôleurs 68HC11A0, A1, A8. Toutes les fonctions d'émulation se font en temps réel et on peut configurer l'émulateur dans tous les modes : bootstrap, special test, single chip et extended. L'utilisateur peut également choisir d'utiliser l'horloge de la carte cible (target) ou bien l'horloge interne de l'émulateur. Voici ci-dessous l'ensemble des configurations que l'on peut effectuer sur le type de processeur :

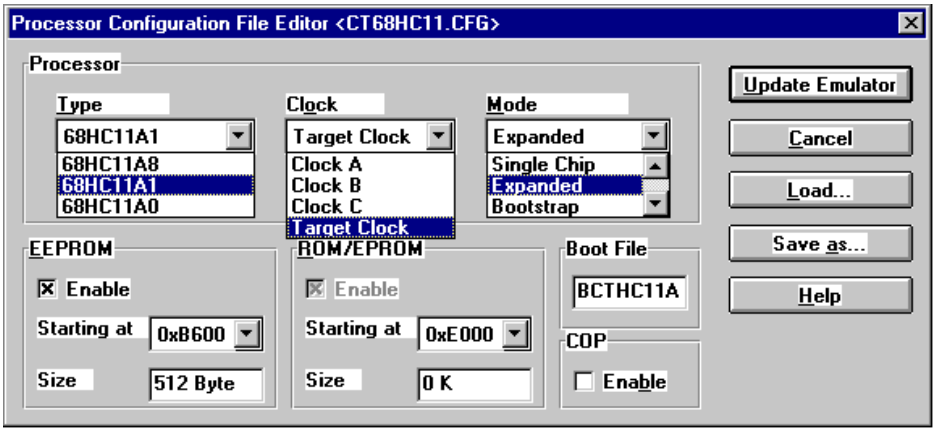

Figure VI-1 – écran de configuration de l'émulateur CT68HC11.

L'émulateur 68HC11 possède 64 Ko de mémoire programme et supporte une fréquence d'horloge jusqu'à 24 MHz (soit 6 MHz interne).

L'émulateur permet grâce au logiciel Pathfinder de visualiser simultanément dans des fenêtres séparées l'état du microprocesseur, la mémoire, les registres internes, le code désassemblé, la pile (pointeur + contenu) et la ligne de commande. L'émulateur permet également de visualiser le code source, les variables du programme ainsi que l'exécution pas à pas en mode Trace.

B. *Utilisation de l'émulateur CT68HC11 avec Pathfinder.*

L'utilisation de l'émulateur se fait entièrement à l'aide de la souris et les commandes sont tapées dans la fenêtre prévue à cet effet. La liste des commandes est disponible dans le guide "Emulator Command Reference" et voici les fonctions les plus courantes lors de la prise en main de l'émulateur :

- ASS: il s'agit de l'assembleur en ligne qui fonctionne de la même façon que PCBUG11 ou BUFFALO.

  Exemple d'utilisation : ASS 8000h ou bien ASS 0F000h pour assembler en ligne aux adresses 8000 ou F000 en hexa.
- C : efface les points d'arrêt.
- D : affiche les points d'arrêt.

remarque: *la mise en place et l'effacement d'un point d'arrêt peuvent se faire de manière pratique avec la souris : il suffit de double-cliquer sur l'adresse pour placer ou enlever un point d'arrêt à l'adresse pointée par la souris.*

➔ G : permet l'exécution du programme chargé en mémoire. L'utilisation du bouton droit de la souris permet d'exécuter le code en mode Trace, chaque clic droit exécute l'instruction suivante.

➔ LOAD/LOADH/LOADS : permettent de télécharger respectivement un fichier objet + symboles, un fichier objet, un fichier symbole.

remarque : *le chargement d'un programme peut se faire à l'aide de la barre menu en sélectionnant File>LOAD>fichier objet ASHLING. En effet, l'émulateur ne peut pas lire directement les fichiers .S19, il est nécessaire de les convertir à l'aide du programme Srec2ash.exe pour récupérer la table des symboles. Un fichier .OBJ est alors généré et peut être chargé dans l'émulateur.*

➔ R : lit un octet à l'adresse spécifiée. La lecture d'une plage d'adresse de addrstart à addrend se fait par la syntaxe suivante : R addrstarth T addrendh.

➔ REG : permet de visualiser l'état des registres du 68HC11 : PC, A, B, DSP, X, Y et le Code Condition register (CCR).

➔ S : permet de définir des points d'arrêt à l'adresse spécifiée.

➔ W : permet d'écrire un octet à une adresse précise. On peut également utiliser la syntaxe W byteh addrstarth T addrendh qui est similaire à BF $addrstart $addrend $byte de PCBUG11 et du moniteur BUFFALO.

Une manipulation intéressante consiste à émuler un 68HC11Ax en mode bootstrap (mémoire émulateur) avec le contrôle de PCBUG11 sur le port série. On visualise ainsi sur la capture d'écran ci-dessous le programme du TALKER qui a été téléchargé en RAM par PCBUG11.

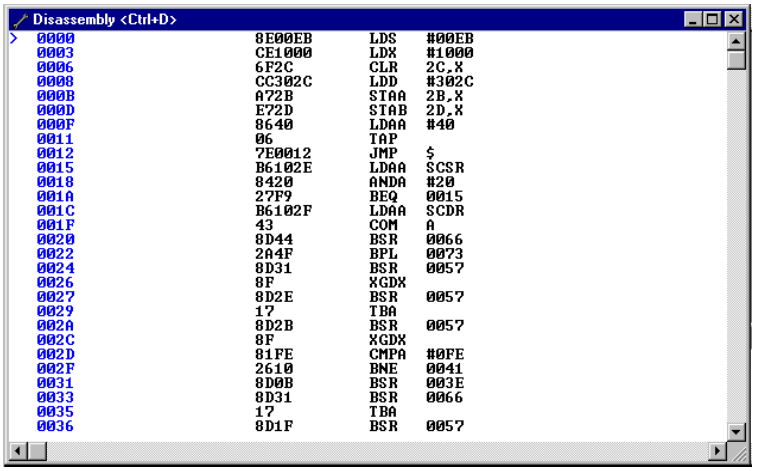

Figure VI-2 – le TALKER de PCBUG11 visualisé depuis l'émulateur.

Le programme PCBUG11 permet de charger et débugger de la même manière des programmes comme s'il s'agissait d'un 68HC11 implanté sur circuit imprimé. En revanche, l'utilisation du port série par le programme utilisateur oblige à relancer PCBUG11 avec la commande Restart et l'utilisation de la mémoire en mode étendu sur la carte mère semble impossible (passage du mode bootstrap en mode special test impossible).

C. Mise en place du moniteur sous Pathfinder.

La mise en place du moniteur BUFFALO a été réalisée grâce à l'émulateur : on charge en RAM (sur la carte cible) le code du moniteur BUFALLO pour 68HC11Ex (compatible avec le processeur 68HC11A1). On lance ensuite le programme terminal sur un autre PC relié à la carte mère par le port série. On peut ainsi à l'aide de Pathfinder vérifier le fonctionnement correct du moniteur ainsi que les espaces mémoires occupés en RAM interne et en RAM externe : le moniteur occupe 8 Ko de l'adresse $8000 à l'adresse $9FFF. Des variables propres au moniteur occupent un faible espace en RAM à partir de l'adresse $2D. Un espace de 20 octets est laissé libre pour la pile utilisateur. Pour plus de renseignement sur la place disponible en RAM interne, on pourra se référer au fichier source de BUFFALO (buf34.asm pour HC11Ex).

Le chargement d'un programme utilisateur par le moniteur BUFFALO se fait grâce à la commande LOAD T. Après avoir tapé la commande LOAD T, le moniteur attend le téléchargement d'un fichier S19. On utilise pour cela la fonction de transfert de fichiers du Terminal disponible dans le menu Transferts>Envoyer un fichier texte>toto.s19.

***Remarque importante** : si on charge un programme en EEPROM (à partir de l'adresse $B600), il faut ralentir le transfert de caractères en attribuant par exemple une durée de 1/10s par octet.*

Le téléchargement terminé, le moniteur renvoie alors le message "done". L'exécution du programme se fait alors grâce à l'instruction G addr. Lors des tests, le programme testio.S19 a été chargé en $C000 (RAM) puis en $B600 (EEPROM). L'exécution s'est faite par G C000 (ou B600 selon le cas) depuis la ligne de commande du moniteur. Le retour à la ligne de commande du moniteur se fait par pression sur le bouton RESET de la carte cible (à condition d'avoir bien spécifié en $FFFE-FF l'adresse de départ du moniteur qui est ici $8000).

***remarque** : lors de la compilation en C, il faut spécifier dans le fichier vector.c (table des vecteurs d'interruption) l'adresse du vecteur de RESET afin que le microcontrôleur boote correctement sur le moniteur.*
La ligne à modifier est la suivante : _stext, / RESET */*
*pour devenir : (void *)((int)0x8000),/* RESET où 0x8000 est l'adresse en hexadécimal du début du code du moniteur.*

On recompile le programme utilisateur avec le nouveau fichier vector.c, on charge le fichier S19 généré avec le moniteur grâce à la commande LOAD T précédemment décrite. On peut vérifier grâce à l'émulateur si l'adresse de RESET est bien chargée : pour cela on interrompt l'émulation et on positionne la fenêtre memory ou disassembly à l'adresse $FFFE (on lit 80) et en $FFFF (on lit 00). Si cela est correctement configuré, on relance l'émulateur là où on l'avait interrompu et on tape G C000 sur le terminal pour exécuter le programme utilisateur. Pour revenir au moniteur on appuie sur le bouton RESET qui a pour effet de diriger le microcontrôleur à l'adresse $FFFE-$FFFF puis $8000.

Pour des raisons de fiabilité, le moniteur peut être implanté en EPROM comme c 'est le cas de toutes les cartes avec moniteur disponible chez Motorola ou sur Internet.

VII. Mise en place du moniteur en EPROM : utilisation du programmateur A.R.T..

La programmation de l'EPROM s'est faite grâce au programmateur EPP2 de la société ART (Advanced Research Technology). On trouvera sur le site web du constructeur les dernières mises à jour des logiciels sous DOS et sous Windows pour piloter le programmateur. Le fichier source (S19) est transféré dans la mémoire tampon du logiciel. Le logiciel lit le fichier S19 et place le code du moniteur à partir de l'adresse 8000h. Il suffit d'utiliser la fonction copier / dupliquer (Move / copy sur la capture ci-dessous) pour placer le code situé en 8000h - 9FFFh en 0000h. Les 2 derniers octets reçoivent les valeurs 80h et 00h afin de configurer le vecteur de RESET de la carte HC11 comme la capture d'écran ci-dessous :

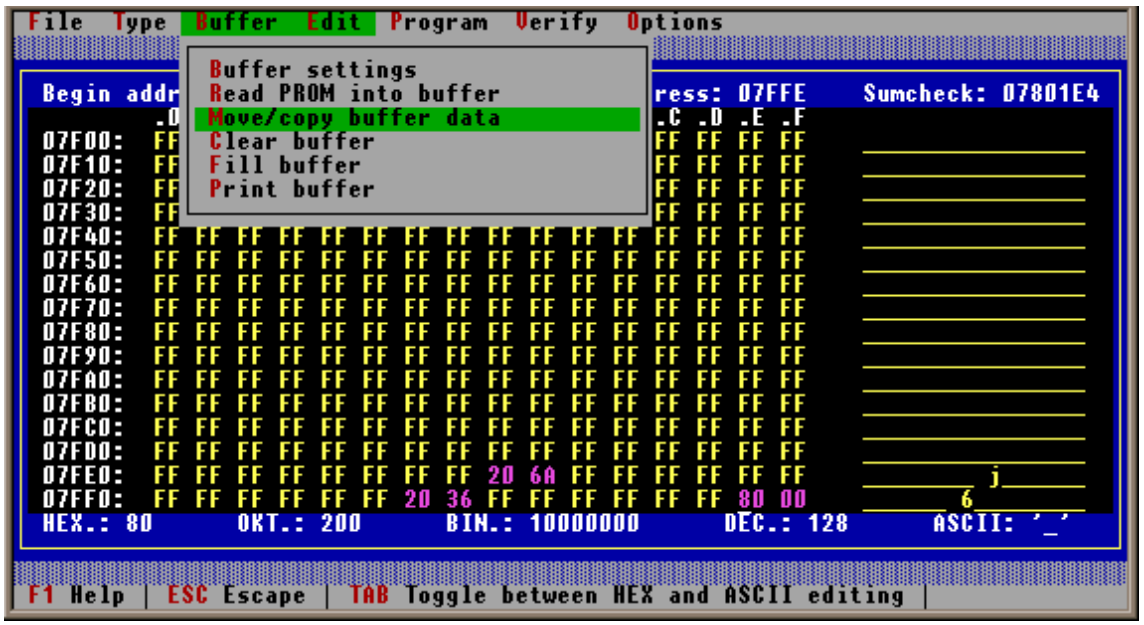

Figure VII-1 – programmation de l'EPROM (table des vecteurs d'interruption).

Nous remarquons que les octets modifiés aux adresses $FFE8 et $FFF6 (ici représentés en violet) servent à la configuration du noyau temps réel µC/OS que nous aborderons au chapitre suivant.

Enfin après avoir effacé l'EPROM aux UV et configuré le logiciel pour le bon type d'EPROM, on lance la programmation.

L'EPROM est placée sur la carte mère : on change le décodage d'adresse (programme ALTERA). On vérifie à l'aide de l'émulateur que le code du moniteur est bien placé à l'adresse 8000h du 68HC11 et la RAM s'étend maintenant de 0000h à 7FFFh (rappel :les ressources internes sont prioritaires).

VIII. Mise en place d'un noyau multitâche temps réel µC/OS.

A. *Introduction : qu'est ce qu'un système temps réel ?*

Pour expliquer brièvement le rôle d'un système multitâche, je vais présenter le logiciel µC/OS-II (version 68HC11) conçu et mis au point par Labrosse. Le logiciel µC/os-II est un noyau temps réel qui permet d'effectuer une exécution de plusieurs tâches sur un microcontrôleur 68HC11 ou un microprocesseur (680x0 par exemple). On peut trouver sur le site Internet de µC/OS II (voir bibliographie) les différentes versions de µC/OS II portées sur des systèmes différents (Motorola famille 680x0, 68HC11/16, Power PC 860, Intel 80x86, Philips XA...). Ce noyau multitâche est fourni gratuitement sur le site Internet et un manuel écrit par l'auteur peut être commandé (voir bibliographie).

B. *Le rôle du µC/OS sur le 68HC11.*

Le multitâche / monoprocesseur permet d'effectuer à l'échelle humaine plusieurs tâches simultanément. En effet l'utilisateur peut alors diviser son projet en plusieurs tâches indépendantes. Au niveau du processeur, une seule tâche est effectuée à la fois en revanche le multitâche permet d'éliminer les temps machines inutilisés (boucle d'attente...).

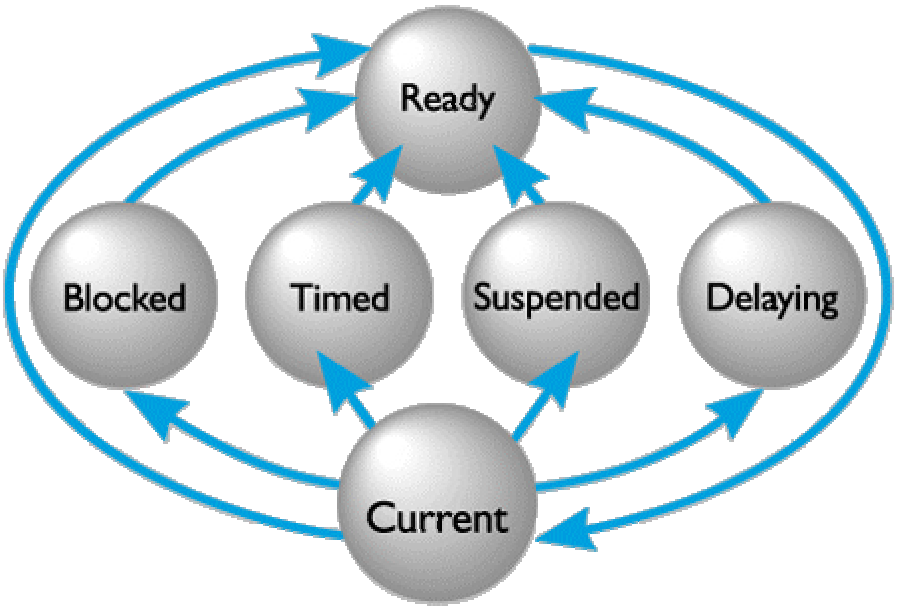

Figure VIII-1 – les différents états des tâches

Le fonctionnement du noyau temps réel sur le 68HC11 est le suivant :

➔ lors de l'initialisation du programme µC/OS-II, les différents programmes de l'utilisateur sont considérés comme des tâches qui sont toutes créées pendant cette période d'initialisation. Le programmeur doit alors spécifier le point d'entrée de la tâche, l'emplacement des données pour cette tâche, l'adresse haute de la pile de la tâche et le degré de priorité de la tâche. Ainsi la tâche de plus haut degré de priorité et prête est exécutée par le microcontrôleur (on dit que le noyau est **préemptif**). La figure VIII-1 montre les différents états des tâches lors du fonctionnement du noyau multitâche.

Remarque : *dans le noyau µC/OS, l'utilisateur dispose de 64 tâches (56 réellement disponibles) où chaque degré de priorité correspond à une tâche, c'est-à-dire que 2 tâches ne peuvent pas avoir le même degré de priorité (pas de "round-robin" ou "méthode du tourniquet").*

Les caractéristiques essentielles du noyau temps réel µC/OS sont les suivantes :

- création et gestion de 63 tâches maximum.
- création et gestion de sémaphores binaires et comptés.
- fonction d'attente de tâche.
- changement de priorité des tâches.
- effacement de tâches.
- suspension et continuation de tâches.
- envoi de messages depuis une routine d'interruption (ISR) ou d'une tâche vers une autre tâche.

Les tâches peuvent ainsi communiquer avec d'autres tâches (grâce aux sémaphores, boîtes aux lettres, files d'attentes…) ou bien avec des périphériques grâce aux ISR (**Interrupt Service Routine)** dont la liste complète est disponible dans la note d'application AN711 de chez Philips.

C. Notion de temps réel.

La notion de temps réel correspond à la façon dont les tâches sont exécutées dans le temps : le temps d'exécution des tâches étant déterminant pour la commutation des tâches, le noyau temps réel exécute en premier les tâches dont le temps d'exécution est critique.

Le fonctionnement du noyau préemptif est illustré à la figure suivante :

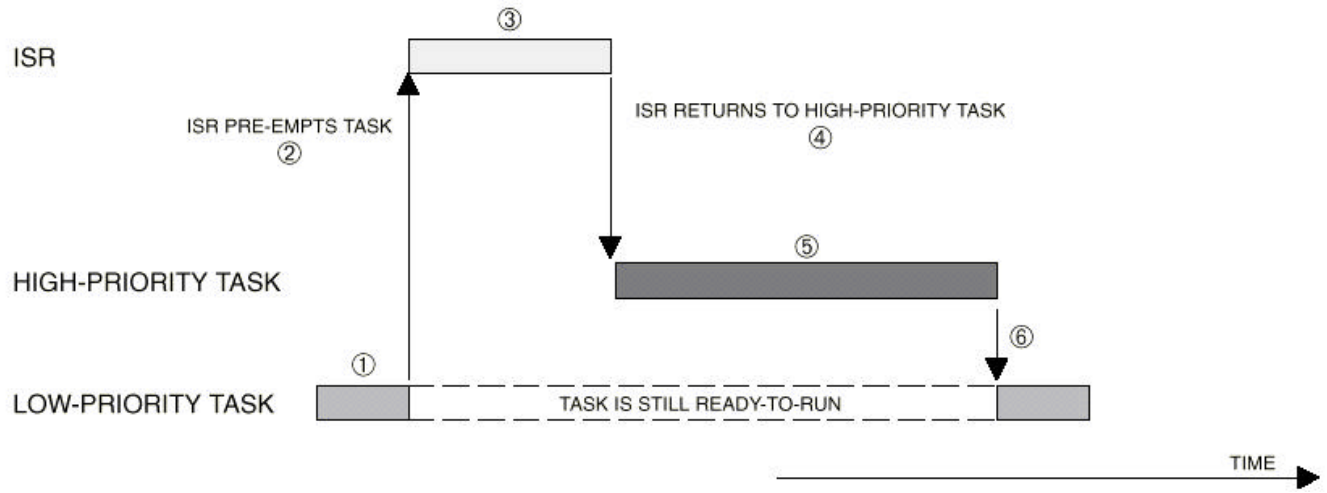

Figure VIII-2 – illustration d'un noyau préemptif.

Comme nous le voyons, une tâche de faible priorité est exécutée en (1) alors qu'une interruption asynchrone intervient en (2). Le microprocesseur saute donc au vecteur d'interruption exécutant alors une tâche prête de degré plus important que la tâche précédente (3). A la fin de la routine d'interruption, l'ISR (Interrupt Service Routine) invoque un service

du noyau (4) qui décide de retourner à la tâche de plus haute priorité et non à la tâche précédente de plus basse priorité. La tâche de plus haut degré s'exécute donc (5) de manière normale à moins qu'une nouvelle interruption n'arrive. A la fin de la tâche de haute priorité, le noyau reprend l'exécution de la tâche de plus faible priorité qui est prête depuis son interruption (6). Comme on peut le voir, le noyau garantit que les tâches dont le temps d'exécution est critique sont effectuées en premier.

Un noyau temps réel assure principalement 2 fonctions : **l'ordonnancement** et le **changement de contexte**. L'ordonnancement permet de déterminer quelle tâche prête a le plus haut degré de priorité. Quand une tâche de plus haute priorité doit être exécutée, le noyau doit sauvegarder toutes les informations nécessaires de la tâche en cours d'exécution afin de permettre une éventuelle continuation. Les informations sauvegardées (appelées contexte de la tâche) sont généralement les registres (A, B, X, Y, CCR pour le HC11), la pile et le pointeur de pile de la tâche qui va être suspendue. Lors de l'exécution d'une nouvelle tâche à un plus haut degré de priorité, le contexte de la nouvelle tâche est chargé puis l'exécution reprend là où elle s'était interrompue.

Le changement de contexte est illustré à la figure suivante :

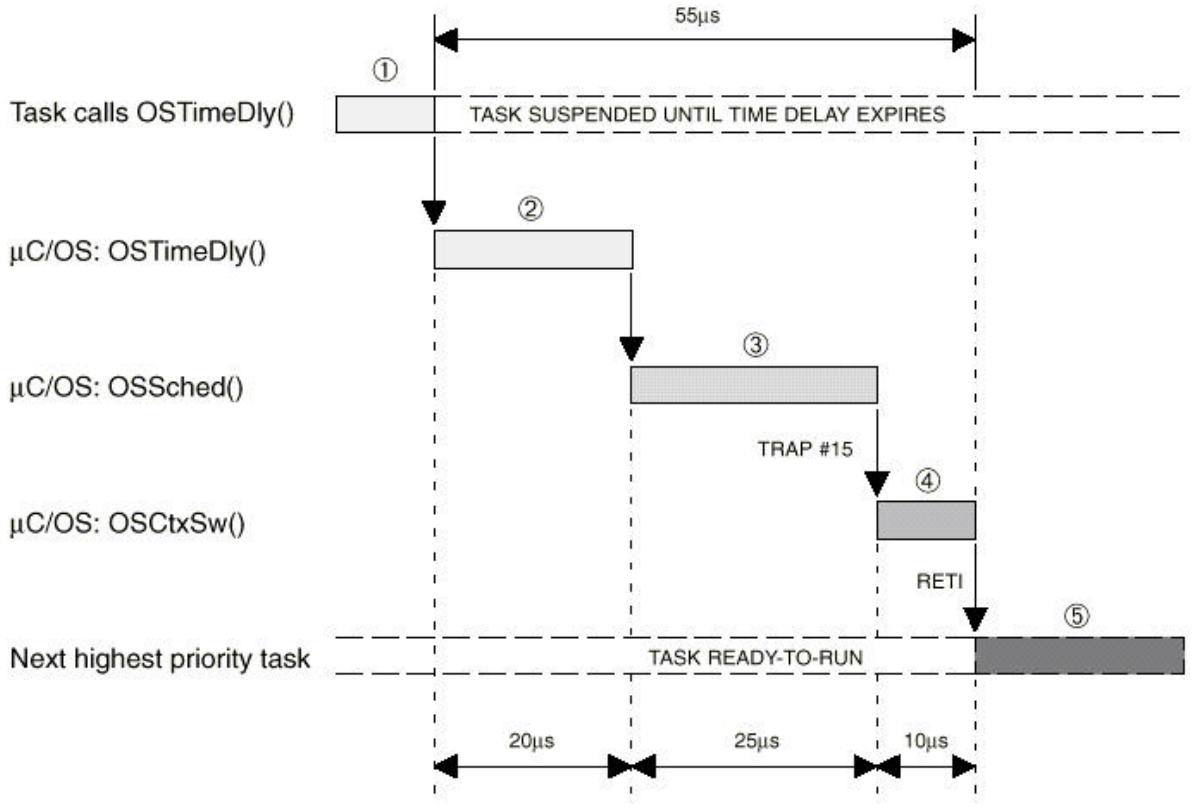

Figure VIII-3 – illustration du changement de contexte de tâches.

Nous avons vu précédemment que le noyau est préemptif, c'est-à-dire qu'il exécute la tâche de plus haut degré de priorité qui soit prête. Un exemple de changement de contexte peut être provoqué grâce à la fonction OSTimeDly(n); où n représente le nombre de coups d'horloge (ticks, interruption TIMER) que la tâche va laisser à d'autres tâches (1). L'appel de ce service du noyau place la tâche en suspens (2) et appelle l'**ordonnanceur** (**scheduler**).

L'ordonnanceur (à l'aide du **dispatcher**) va chercher le processus à exécuter (choix effectué selon le degré de priorité ou bien en fonction du temps d'exécution) (3). Un changement de contexte est effectué par la fonction OStxSw(); qui est spécialement écrite pour le processeur employé. En effet, le changement de contexte consiste d'une part en la sauvegarde de l'adresse de retour (PC), la pile, les registres du 68HC11, et l'adresse du haut de la pile, d'autre part au rétablissement du contexte de la nouvelle tâche c'est-à-dire le chargement du pointeur de pile, la pile, le Program Counter et enfin le reste des registres du HC11.

Les temps reportés sur la figure ci-dessus correspondent à un microcontrôleur Philips XA avec une horloge de 24MHz. On remarque que le changement de contexte ne prend que 10µs. Le "basculement" d'une tâche à l'autre a nécessité 55 µs entre le moment de la suspension de la tâche et l'exécution de la nouvelle tâche.

µC/OS nécessite une source d'interruption périodique générée ici par le TICK TIMER du 68HC11. µC/OS utilise le registre OC1 pour générer une interruption et configure le TIMER à 250KHz. Le taux d'interruption est fixé à 2500, donc le temps imparti à une tâche est de 10 ms (100 Hz). Ce paramètre (OS_TICK_OC_CNTS) est configurable selon les critères de temps des tâches mises en jeu et permet d'accroître l'efficacité du noyau (il ne faut pas passer plus de temps à commuter les tâches qu'à les exécuter !).

D. La création des tâches.

La création d'une tâche se fait grâce à la fonction suivante :

```
OSTaskCreate(AppTask1, (void *)0, (void *)&AppTask1Stk[255], 10);
```

Le premier paramètre est le point d'entrée du programme utilisateur (nom de l'étiquette), le second paramètre indique l'adresse des données, le troisième paramètre indique l'adresse du haut de la pile de la tâche et le dernier paramètre indique le degré de priorité de la tâche (ici 10). Pour plus de précision, on pourra consulter le fichier OS_CPU_C.C (contenu dans c68hc11.zip), le fichier 68hc11.c (contenu dans 68hc11_1.zip) ou bien dans le manuel adéquat de µC/OS.

Le corps de la tâche est constitué de la manière suivante :

- une zone d'initialisation de la tâche.
- une zone permettant d'initialiser les variables du programme utilisateur.
- une zone constituée d'une boucle infinie où l'utilisateur place le code de son programme.
- une instruction OSTimeDly(x) permettant de céder x coups d'horloge aux autres tâches.

***remarque**: cette instruction est obligatoire dans le contenu du programme de la tâche afin de basculer vers d'autres tâches.*

E. Les fonctions de µC/OS.

Les fonctions de base du noyau temps réel employées dans le programme **test.c** (disponible en annexe) sont les suivantes :

- OSinit(); initialisation globale du noyau
- OSTaskcreate (pointeur sur le programme utilisateur, pointeur données, adresse haut de la pile, priorité); permet de créer une tâche. Le pointeur "données" permet de

faire passer des données à une tâche lors de son initialisation. La priorité est définie telle que plus le nombre est faible, plus le degré de priorité est élevé.

- OSStart(); permet de créer au moins une tâche en appelant OSTaskCreate();

D'après la remarque précédente, le fonctionnement correct de µC/OS exige qu'une tâche doive obligatoirement appeler une fonction (un service du noyau) afin de :

- faire attendre la tâche d'un délai de n coups d'horloge (OSTimeDly).
- attendre un sémaphore.
- attendre un message d'une autre tâche ou d'une ISR (Interrupt Service Routine : il peut s'agir d'une fonction du noyau qui permet de modifier le statut d'une tâche comme le degré de priorité d'une tâche).
- suspendre l'exécution de cette tâche.

Les fonctions du noyau commencent toutes par les lettres « OS » (Operating System), les lettres suivantes OS définissent les familles de fonctions de µC/OS II. La liste des fonctions de µC/OS II est la suivante :

- initialisation (OSinit, OSStart).
- la gestion des tâches (OSTaskCreate, OSTaskDel, OSTaskDelReq, OSTaskChangePrio, OSTaskSuspend, OSTaskResume, OSSchedlock, OSSchedUnlock).
- la gestion du temps (OSTimeDly, OSTimeDlyResume, OSTimeSet, OSTimeGet).
- la gestion des sémaphores (OSSemCreate, OSSemAccept, OSSemPost, OSSemPend, OSSemInit).
- la gestion des boîtes aux lettres (OSMboxcreate, OSMboxAccept, OSMboxPost, OSMboxPend).
- la gestion des files d'attente (OSQCreate, OSQAccept, OSQPost, OSQPend).
- la gestion d'interruption (OSIntEnter, OSIntExit).

La syntaxe de toutes ces fonctions est donné dans la note d'application AN711 de chez Philips et dans le fichier postscript ucosman.ps (version ucos-arm). Les explications complètes et les exemples figurent dans l'ouvrage "µC/OS-II book The real Time Kernel" de Jean J. Labrosse (voir bibliographie).

F. La communication entre tâches.

Voici un bref descriptif des moyens qu'offre µC/OS pour la communication entre tâches.

1. Les sémaphores.

Un sémaphore est un objet qui permet de résoudre les problèmes de synchronisation entre plusieurs tâches concurrentes.

Un sémaphore est constitué d'un compteur à valeur entière (valeur du sémaphore) et d'une file d'attente. Sur le compteur sont définies 2 opérations qui sont Prendre (P) et Vendre (V). Une valeur positive du sémaphore désigne le nombre d'accès disponibles à un instant donné ; une valeur négative représente par la valeur absolue le nombre de processus en attente de libération de la ressource. En effet, avant d'accéder à une donnée, un processus doit utiliser l'opération P. Si la valeur du sémaphore est 0, alors le processus doit attendre que cette valeur

devienne supérieure à 0. Si la valeur est supérieure à 0, alors l'opération P décrémente la valeur du sémaphore et le processus continue son exécution. Après l'accès aux données, le processus doit appeler l'opération V qui incrémente le sémaphore autorisant ainsi aux autres processus l'accès aux données.

Les sémaphores permettent donc soit de synchroniser des processus (échanges de données), d'exclure des processus concurrents (accès à une même variable en même temps). Un exemple d'utilisation de sémaphores est disponible en annexe sous le nom de **test2.c**).

2. *Les boîtes aux lettres.*

Une boîte à lettre est une zone d'échange entre 2 processus et se compose de 2 files d'attente : une file d'attente de messages et une file d'attente de tâches. Les boîtes aux lettres permettent de synchroniser la communication entre des tâches asynchrones.

3. *Les files d'attente.*

Une file d'attente constitue une suite de processus en attente d'un évènement, d'un sémaphore…On trouve alors des files d'attente d'évènements (les processus attendent un certain type d'évènement), des files d'attente sur sémaphore (les processus sont gérés par ordre d'arrivée : une opération Prendre non passante amène la tâche à la fin de la file et pour chaque opération Vendre c'est la première tâche en attente qui est activée) et enfin des files d'attente sur boîtes aux lettres.

G. *Mise en place du noyau µC/OS.*

1. *Modification des fichiers de compilation et d'édition de liens.*

Le noyau temps réel µC/OS est écrit en langage C (COSMIC SOFTWARE 4.5). Seuls les portages spécifiques au microcontrôleur (ici le 68HC11F1) sont écrits en langage d'assemblage. Le travail a consisté en premier lieu à adapter le noyau µC/OS au 68HC11Ax et Ex. Les lignes à modifier sont dans les fichiers start.s et build.bat.

Le fichier start.s contient des initialisations propres au 68HC11F1 telles que :

- les bits de protection de la EEPROM (4 bits de poids fort du registre CONFIG).
- l'activation de la logique de décodage (rattachée au PORT G) des boîtiers de RAM et de ROM externes (pas besoin de 74HC138).
- le pointeur de pile et l'initialisation des 1024 octets de RAM.

Les modifications à apporter ont consisté à enlever la configuration des bits de protection de la EEPROM et de la logique de décodage interne, l'initialisation des 256 octets de RAM interne et le pointeur de pile en $0FF.

Remarque: *le mode étendu de la carte mère permet de prolonger la mémoire RAM interne jusqu'à l'adresse $0FFF soit 4Ko de RAM disponible. Il faudra donc prendre soin de ne pas faire "interférer" la pile du moniteur BUFFALO et celle de µC/OS.*

Le fichier build.bat doit être configuré pour appeler le fichier io.h (et non pas iof1.h) qui est situé dans le répertoire de Cosmic Software (ex : C:\logiciel\68HC11\cx32\H6811\IO.H).

Remarque importante : *lors de la compilation, l'archive de µC/OS I (c68hc11c.zip) ne contient pas les fichiers os_core.c, os_mbox.c... nécessaires lors de la compilation de Ucos_II.c mais le fichier objet est heureusement fourni sous le nom de Ucos_ii.o situé dans le répertoire UCOS-II/M68hc11/Cosmic/obj. Il faut donc "commenter" les lignes suivantes du fichier build.bat :*

rem DEL ..\OBJ\uCOS_II.O

rem C:\logiciel\68HC11\cx32\CX6811 -cl..\LST -co..\OBJ -e -gv -l -ocv -v +nowiden +sprec +debug uCOS_II.C

2. *Programmes pour µC/OS II.*

a) Compilation et éditions de liens.

La mise en place du noyau temps réel pour 68HC11Ax ou Ex s'est faite grâce à l'émulateur CT68HC11. Le programme µC/OS sera donc installé dans la mémoire interne de 64 Ko de l'émulateur ou bien en RAM externe après avoir programmé correctement l'EPROM aux adresses $7FE8 (avec $206A) et $7FF6 (avec $2036) ainsi que le vecteur de RESET (pour plus de précisions se reporter au fichier **vector.c** en annexe IV-C-2). µC/OS-II est fourni avec un programme **test.c** (disponible en annexes IV-C-1) qui permet de créer 3 tâches : une tâche de démarrage (AppStartTask) qui va créer 2 autres tâches (task#1 et task#2) lesquelles vont être mises en compétition avec des degrés de priorité différentes (Noter que la tâche de démarrage a le plus haut degré de priorité).

La figure suivante illustre simplement l'ordonnancement des 3 tâches. L'ordonnanceur peut continuer ou suspendre une tâche grâce aux signaux « continue » et « suspend » représentés ici par des flèches en pointillés. Lorsqu'une tâche est suspendue, la suivante et continuée. Le schéma montre comment sont ordonnancées les tâches mais ne précise pas comment fonctionne la communication entre les tâches ou avec les périphériques.

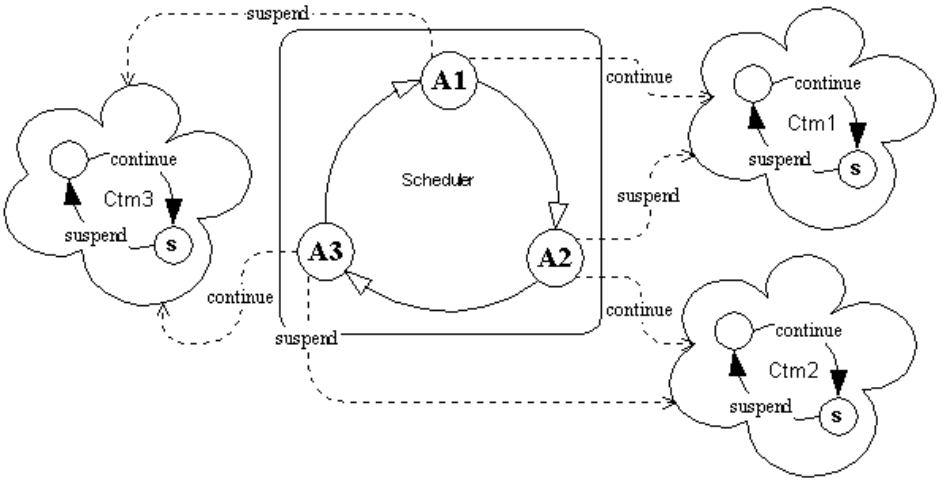

Figure VIII-4- l'ordonnancement des tâches.

Le programme mis au point pour la carte mère est basé sur l'exemple **test.c** fourni dans l'archive c68hc11.zip. Le programme crée une première tâche qui consiste à faire un écho sur la liaison série RS232 configurée à 9600bps, puis une deuxième tâche qui consiste à exécuter un chenillard 4/5 LEDs (PA7 est configurée par PACTL et PA3 est condamnée si on utilise le moniteur BUFFALO) enfin une troisième tâche qui recopie les interrupteurs du PORT E sur le PORT A.

Le fichier test.lnk permet de configurer la mise en mémoire du noyau µC/OS :

```
+seg .bsct   -b 0x0000 -m256                    #zéro page
+seg .bss    -b 0x7000 -m8192                   #adresse de la pile
+seg .text   -b 0x2000 -m32726  -n .text        #adresse début du programme
+seg .const  -a .text                           #début des données
../OBJ/START.O                          # routine de démarrage : initialisation de la RAM,
../OBJ/OS_CPU_A.O                               #du pointeur de pile...
../OBJ/OS_CPU_C.O
../OBJ/TEST.O
../OBJ/uCOS_II.O      #fichier fourni avec l'archive c68hc11.zip à ne pas recompiler
../OBJ/hc11lib.o                                #bibliothèque de fonctions externes.
C:/logiciel/68HC11/cx32/LIB/LIBF.H11    #bibliothèques Cosmic software
C:/logiciel/68HC11/cx32/LIB/LIBI.H11
C:/logiciel/68HC11/cx32/LIB/LIBM.H11
#+seg .const  -b 0xFFD6 -m42                   #table des vecteurs d'interruption.
#../OBJ/VECTORS.O
```

Les 2 dernières lignes ne sont à commenter que si µC/OS marche avec le moniteur BUFFALO en EPROM. En effet, les vecteurs d'interruptions doivent être programmés dans l'EPROM comme cela est expliqué au chapitre "Mise en place du moniteur en EPROM". Si les lignes ne sont pas commentées, l'émulateur ou BUFFALO renverra un message d'erreur d'écriture.

La mise en place du programme multitâche se fait grâce à la commande LOAD de l'émulateur (utiliser hex2ash.exe pour créer test.obj) ou bien par la commande LOAD T du moniteur BUFFALO après avoir converti le fichier.hex en .S19.

L'exécution du programme se fait par la commande G 2000. On observe alors le fonctionnement correct des tâches. L'utilisation de la fonction OSTimeDly permet donc de faire des temporisations dont l'unité est le tick, laissant ainsi ces cycles d'attente à d'autres tâches.

b) Quelques explications concernant le fonctionnement des sémaphores.

Un deuxième programme a permis de mettre en évidence le fonctionnement des sémaphores avec µC/OS II. Le programme **test2.c** (cf. annexe C-3) permet de synchroniser 3 tâches. La tâche #1 attend le sémaphore sem1 et lorsque celui - ci est validé, la tâche allume la LED PA5. De même pour la tâche #2 qui attend le sémaphore 2 (sem2) et allume la LED PA6. La tâche 3 attend le sémaphore 3 (sem3) et allume la LED PA7. La tâche principale (celle qui a la plus forte priorité) permet de valider les 3 sémaphores en « même temps » après avoir exécuté une temporisation (fonction OSTimeDLy).

Il est intéressant de se reporter au document ucosman.ps (voir bibliographie) qui explique le fonctionnement des sémaphores. Lors de la création d'un sémaphore (initialisation), on spécifie la valeur du sémaphore qui est nulle dans l'exemple **test2.c**.

En effet une tâche peut être mise en attente avec la fonction OSSemPend qui décrémente la valeur du sémaphore si elle était nulle. Lorsque le sémaphore est négatif, la fonction OSSEmPend attend alors que la valeur du sémaphore redevienne nulle (elle attend l'incrémentation du sémaphore). Ceci est effectué par la tâche principale grâce à la fonction OSSemPost qui incrémente le sémaphore si sa valeur était nulle ou négative et donne la main à la tâche de plus haute priorité qui est en attente de sémaphore.

Par la suite, les tâches poursuivent leur exécution normale. Pour réutiliser les sémaphores, il faut les réinitialiser grâce à la fonction OSSemInit().

Le noyau µC/OS II permet donc de développer des applications 68HC11 multitâches en langage C de Cosmic Software. Les fonctions de communications entre tâches et les ISR permettent de concevoir des programmes performants. Le portage du noyau µC/OS II a divers processeurs et microcontrôleurs le rend populaire pour le développement d'applications.

CONCLUSION

Le stage proposé par M. Kadionik m'a permis d'approfondir ma connaissance du 68HC11 principalement dans la programmation. En effet, le long de ce stage j'ai pu programmer en langage d'assembleur, en langage C et j'ai pu concevoir des programmes sous un noyau temps réel. Le stage m'a permis d'utiliser plusieurs logiciels tels que MENTOR GRAPHICS sous UNIX, le logiciel ALTERA sous UNIX et sous Windows NT, l'utilisation de l'émulateur 68HC11 de ASHLING, l'utilisation de PCBUG11 et du logiciel embarqué BUFFALO, la programmation avec le logiciel A.R.T.et enfin la recherche de documentation sur le réseau Internet.

IX.Bibliographie

MENTOR GRAPHICS.

Lors du stage, la conception des cartes pour 68HC11 a été développée sous MENTOR GRAPHICS sur Station SUN. La documentation sur l'utilisation de MENTOR GRAPHICS a été donnée par l'ENSERB (CAO MENTOR GRAPHICS, simulations et PCB, de M.G. MORIZET). Un condensé des étapes à suivre pour réaliser un PCB sur MENTOR GRAPHICS est disponible à l'adresse suivante :

http://www-elec.enserb.u-bordeaux.fr/~kadionik

La documentation technique sur le 68HC11 a été fournie gratuitement par MOTOROLA dont le site Web (http://www.mot-sps.com/home/lit_ord.html) permet de commander les manuels de référence utilisés le long du stage :

Hardware et software MOTOROLA :

- le manuel papier "M68HC11 REFERENCE MANUAL" ref. :M68HC11RM/AD et le guide condensé "M68HC11A8 PROGRAMMING REFERENCE GUIDE" ref:MC68HC11A8RG/AD

- l'ouvrage « Les microcontrôleurs HC11 et leur programmation » de C. Cazaubon.

  Edition Masson.
  Enseignement de l'Electronique
  ISBN 2-225-85527-7

- le manuel au format PDF "Technical Summary 8-bit Microcontrollers MC68HC11A8, A1, A0" (11a8bk.pdf).

- les notes d'application suivantes : "an1060" pour le fonctionnement en mode bootstrap, AS-67 Application Snapshot pour le fonctionnement en mode spécial test sous PCBUG11 que l'on peut trouver aux adresses suivantes :

  http://www.mcu.motsps.com/lit/app_notes/an1060.htm
  et http://www.mcu.motsps.com:80/lit/faq/as-67.htm

- le logiciel PCBUG11 v3.42 (pcbug342.exe) et son manuel complet "M68HC11 PCBUG11 USER'S MANUAL" ref. : M68PCBUG11/D (pcbug11.pdf) téléchargeables à l'adresse suivante : http://www.mcu.motsps.com/freeweb/amcu_ndx.html#mcu11

➡ le manuel fourni avec la carte d'évaluation pour 68HC11F1 "HC11 EVALUATION SYSTEM OPERATIONS MANUAL" qui donne la liste des commandes du moniteur Buffalo.

➡ le programme moniteur (buf34E.asm) disponible pour toutes les versions du 68HC11 téléchargeable à l'adresse suivante :

http://www.mcu.motsps.com/freeweb/amcu_ndx.html#mon

Software langage C :

➡La version d'évaluation (x68hc11.zip) et le manuel complet "C Cross Compiler User's Guide for Motorola MC68HC11" de COSMIC Software téléchargeables à l'adresse suivante : http://www.std.com/cosmic ou http://www.cosmic-us.com

➡la note d'application "COSMIC V4.1x 68HC11 C Compiler Package" (eval-6811.pdf).

Hardware et software émulateur Pathfinder-11 :

➡ Le manuel complet "Embedded development systems USER MANUAL" de ASHLING Microsystems Ltd, l'adresse du site internet est http://www.ashling.com (rubrique CT68HC11 Microprocessor Development System for windows).

➡ Le guide abrégé des commandes de l'émulateur "Emulator Command reference".

Programmateur EPROM d'Advanced Research technology :

➡ Logiciels disponibles sur le site http://www.artbv.nl

Développement du noyau temps réel sur 68HC11Ax :

➡ "Le temps réel en milieu industriel" de A DORSEUIL, P. PILLOT
Edition Dunod Informatique industrielle.
BORDAS 1991, ISBN : 2-04-019875-X

➡ Ouvrage "MicroC/OS-II, The real-Time Kernel" de J. Labrosse.
Category: Embedded systems Programming
Product code : rd3005
ISBN 0-87930-543-6

Cet ouvrage peut être commandé sur les sites internet suivants :
http://www.ucos-ii.com/
ou
http://www.rdbooks.com/cgi-bin/shopsspecific/store/products/rd3005.htm

- Voir également la note d'application de Philips "AN711 : µC/OS for the Philips XA" :

  http://www.ucos-ii.com/ports.htm (liens vers la note d'application AN711)
  ou http://www.semiconductors.philips.com
  et ftp://ftp.ibsystems.com/pub/philips-mcu/bbs/xa/rtos

- Manuel ucosman.ps : ftp://ftp.cygnus.com/pub/embedded/ucos/ucos-arm/

X. Sources iconographiques

- Figure II-1 : "M68HC11 REFERENCE MANUAL" – MOTOROLA.
- Figure II-2 : MENTOR GRAPHICS sur Station SUN.
- Figure II-3 : Logiciel MAX+ Plus ALTERA.
- Figure II-4 : "M68HC11 REFERENCE MANUAL"- MOTOROLA.
- Figure III-1 : MENTOR GRAPHICS sur Station SUN.
- Figure IV-1 : Logiciel PCBUG11 v3.42 de MOTOROLA.
- Figure V-2 : Hyperterminal WindowsNT.
- Figure VI-1 : Logiciel PathFinder de ASHLING Microsystems Ltd.
- Figure VII-1 : Logiciel Promprog sous DOS d'Advanced Reseach Technology.
- Figure VIII-1 : États des tâches (origine : http://www.on-time.com/index.html)
- Figure VIII-2 : Note d'application AN711 de Philips Semiconductors.
- Figure VIII-3 : Note d'application AN711 de Philips Semiconductors.
- Figure VIII-4 : A premptive kernel (http://www.ec3.om/Uperized/PREEMPTI.HTM)

XI. Annexes

Annexes
du rapport de stage