Option Informatique Industrielle Responsable du Projet **: Patrice KADIONIK**

Alexandre PHAO
Hamid CHOUKRI

PROJET DE FIN D'ETUDES :

Carte 68HC11 avec interface Ethernet

PROMO 2000

Ecole **N**ationale **S**upérieure d' **E**lectronique et de **R**adioélectricité de **B**ordeaux

Remerciements :

Notre formation à l'ENSERB touche à sa fin, et nous exprimons vivement toute notre reconnaissance pour le corps enseignant de l'ENSERB pour toutes les connaissances qui nous ont été transmises au cours de ces 3 années afin de mieux appréhender notre métier d'électronicien qui nous attend .

Concernant ce projet de fin d'études, nous tenons à remercier les personnes suivantes dont la collaboration nous a permis de mener au mieux le projet :

Monsieur Patrick LALANDE	du pole technique.
Monsieur Dominique MARSAN	du pole technique.
Madame LAVIGNE	de l'IXL.

Et particulièrement des remerciements à notre responsable de projet Monsieur Patrice KADIONIK pour nous avoir aidé et soutenu tout au long de ces 12 semaines dans chacune des phases du projet.

SOMMAIRE :

0) INTRODUCTION :

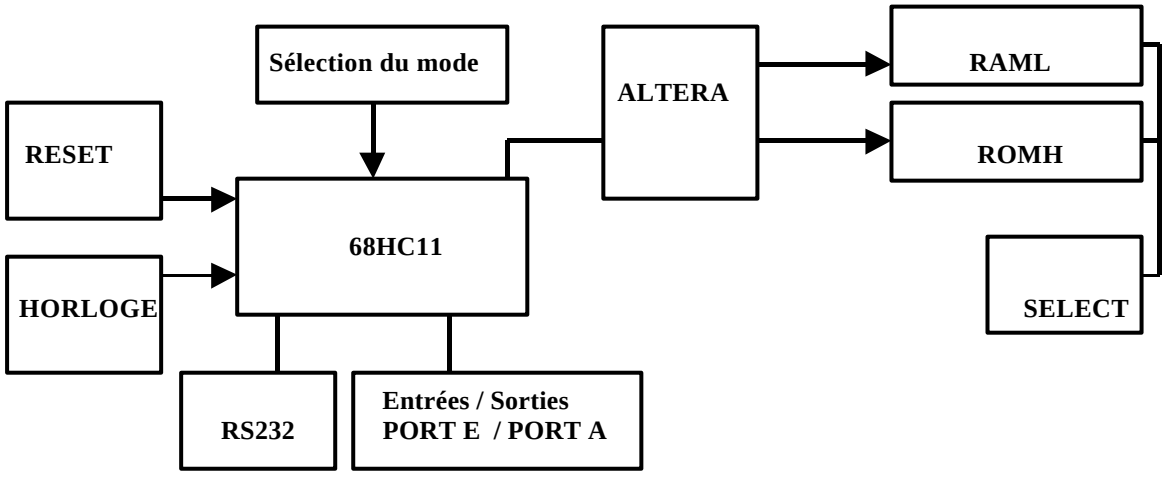
L'année précédente une carte basée sur le microcontrôleur 68HC11 de MOTOROLA a été développée dans le cadre de TP d'électronique en 1ère année de la filière informatique. Cette carte initiale comprend un microcontrôleur 68HC11A8 adressé à des ressources externes RAM / ROM, et dispose d'une liaison série de type RS232 pour la communication de données avec un PC offrant ainsi une multitude de possibilités d'application.

Pour étendre les fonctionnalités de cette carte, il est demandé d'y intégrer une interface Ethernet à l'aide du composant CS8900A de Cirrus Logic se comportant comme un périphérique 8 bits intégrant toutes les couches de protocoles Internet en Hardware, et de modifier quelques aspects de la carte. Puis, après développement et test de la carte, il est convenu de développer les logiciels couplés au noyau Temps Réel OS_2 afin de permettre un contrôle à distance par Sockets Internet dans un premier temps, puis par le Web dans un deuxième temps.

I) PARTIE HARDWARE

La partie Hardware constitue la partie la plus difficile à développer étant donné le nombre importants d'étapes avant l'obtention de la carte final.

1) Schéma de principe initial :



La carte initiale est articulée autour d'un microcontrôleur 68HC11A8 de MOTOROLA sur lequel est adressé jusqu'à 64K octets (16 Bits d'adresse : A0..A15) de ressources externes pour le mode étendu du processeur. Parmi ces ressources externes, la RAM de 32K est adressée initialement en $0000, puis pour les 32K octets d'adresses supérieures, un interrupteur permet de sélectionner soit cette RAM ou bien une ROM qui contient le moniteur BUFFALO.

Deux autres interrupteurs permettent de configurer le microcontrôleur dans l'un de ses 4 modes (Single, Extended, Bootstrap ou Test).

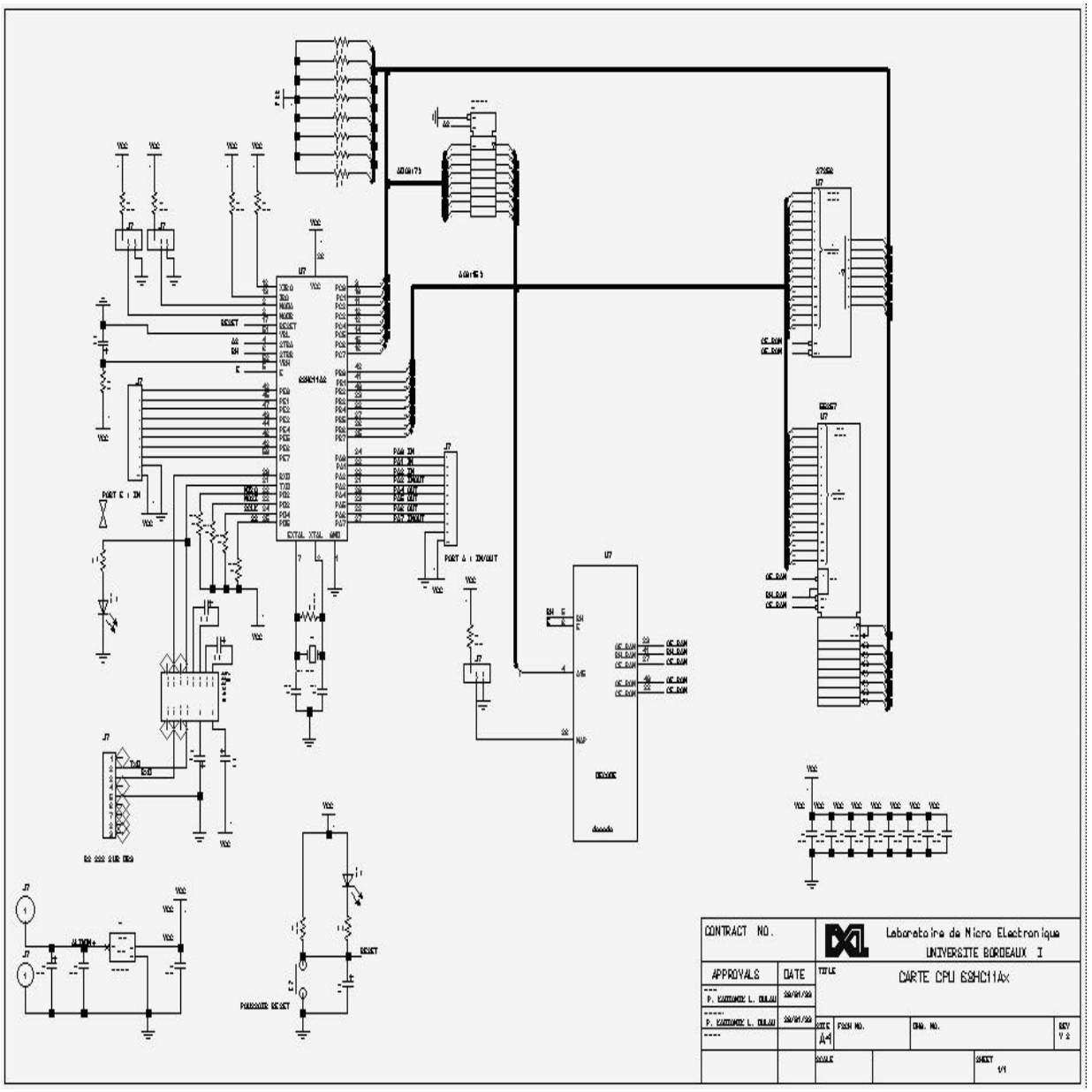
Le circuit 74HC573 est un latch assurant le démultiplexage du port C du 68HC11 sur lequel les adresses basses (A0..A7) et les données 8 Bits y sont véhiculées.

Le MAX232 assure la conversion de signaux polaire 0-5V en bipolaire –12V/+12V.

L'ALTERA permet de faire le décodage d'adresse, et suivant le cas, il génère les signaux de sélection permettant au microcontrôleur d'effectuer ses accès mémoire avec le bon composant.

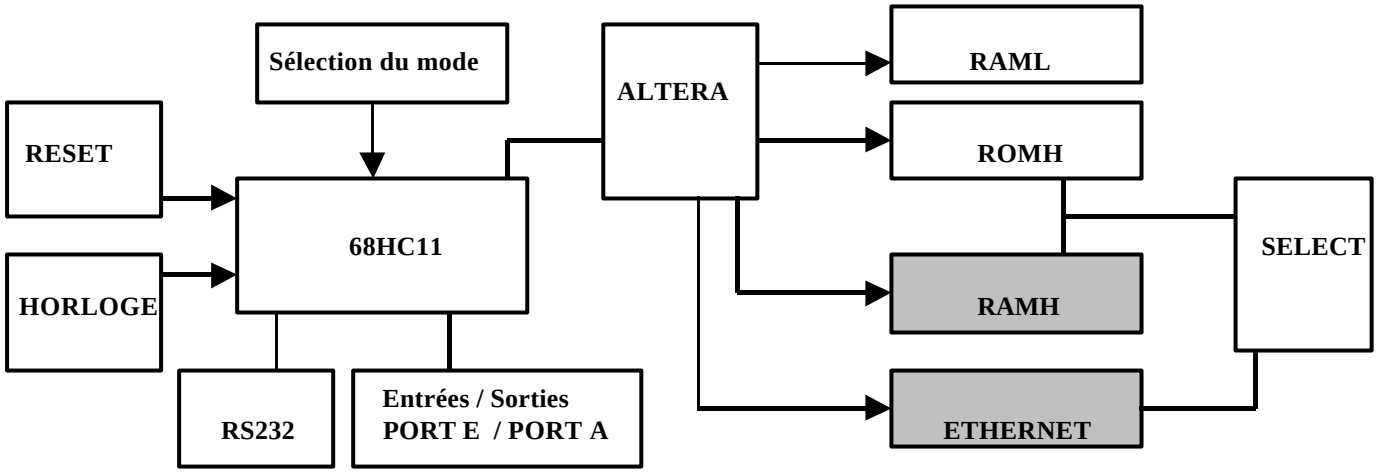
Quant aux entrées / sorties du microcontrôleur, le port E est un port d'entrée 8 bits (PE0….PE7), et concernant le port A, PA0…PA2 sont des entrées et PA3…PA7 sont des sorties.

2) Schéma électrique initial :



Le schéma électrique ne pose pas de problèmes, on retrouve toutes les unités citées dans le paragraphe précédent et dans le schéma de principe.

3) Travail à effectuer:



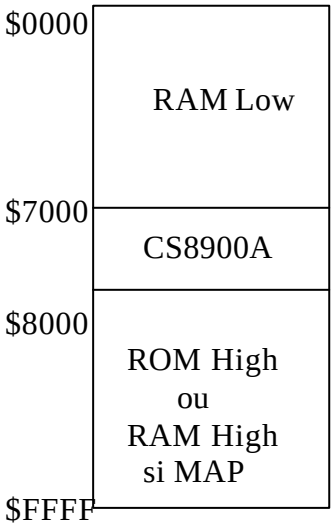
La grande partie du circuit à concevoir reprend le circuit initial avec en supplément le chip d'interface Ethernet CS8900A adressé comme un périphérique 8 bits, et une RAMH pour l'adressage des 32K supérieurs. On pourra donc avoir les configurations RAML /ROMH ou bien RAML / RAMH suivant que l'on utilise ou pas le moniteur BUFFALO, et dans les 2 cas l'interface Ethernet adressée volontairement dans la zone $7000-$7FFF.

De plus, il a été ajouté :

Des signaux supplémentaires pour le décodage des adresses et la génération des signaux de contrôle sur les ressources externes.

Une entrée d'interruption sur le 68HC11 pour le fonctionnement du CS8900A en mode MEM.

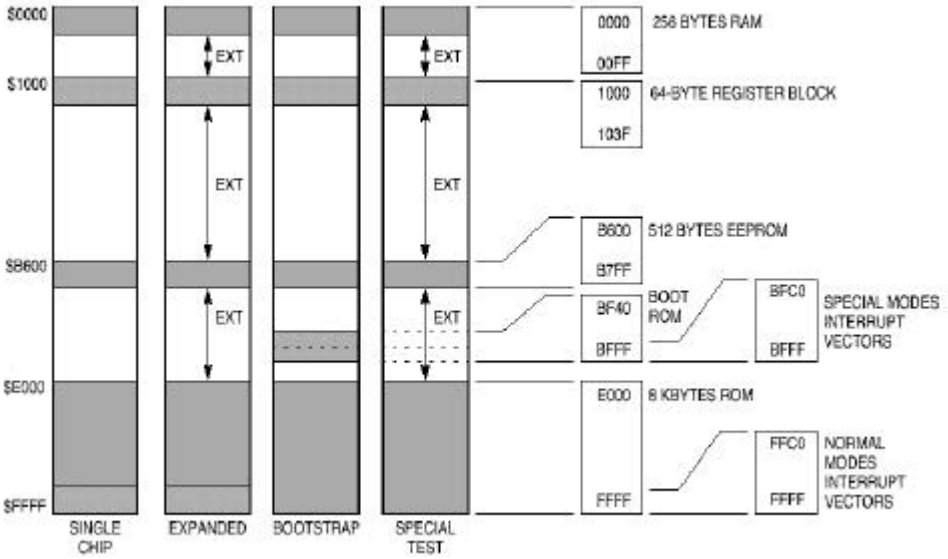
4) Cartographies Mémoires avec interface Ethernet:



Adressage mémoire des ressources externes

- Une RAM Basse RAMl est adressée entre $0000 et $6FFF.
- Ensuite vient la zone de l'interface Ethernet adressée entre $7000 et $7FFF.
- Pour les adresses hautes – les 32K supérieurs de $8000 jusqu'à $FFFF – on a le choix entre une ROMh ou une RAMh.

Sans oublier que suivant le mode de configuration du microcontrôleur on aura les mapping suivants :



5) Décodage d'adresses :

Interprétation :

- Lorsque A15 vaut 0 et A14.A13.A12 vaut 111, on accède dans ce cas au chip Ethernet adressé aux adresses $7XXX.
- Lorsque A15 vaut 1, c'est à dire pour les adresses débutant en $8000, ET si MAP vaut 0 c'est la ROMH qui est sélectionnée, sinon si MAP vaut 1, c'est la RAMH qui est sélectionnée.
- Pour le reste des cas avec A15 à 0 et A14.A13.A12 différent de 111, c'est la RAML qui est sélectionnée.

D'ou le tableau suivant :

MAP	A15	A14.A13.A12	RAML	Ether	ROMH	RAMH
X	0	0 0 0 à 1 1 0	1	0	0	0
X	0	1 1 1	0	1	0	0
0	1	X X X	0	0	1	0
1	1	X X X	0	0	0	1

Ou encore un tableau plus complet:

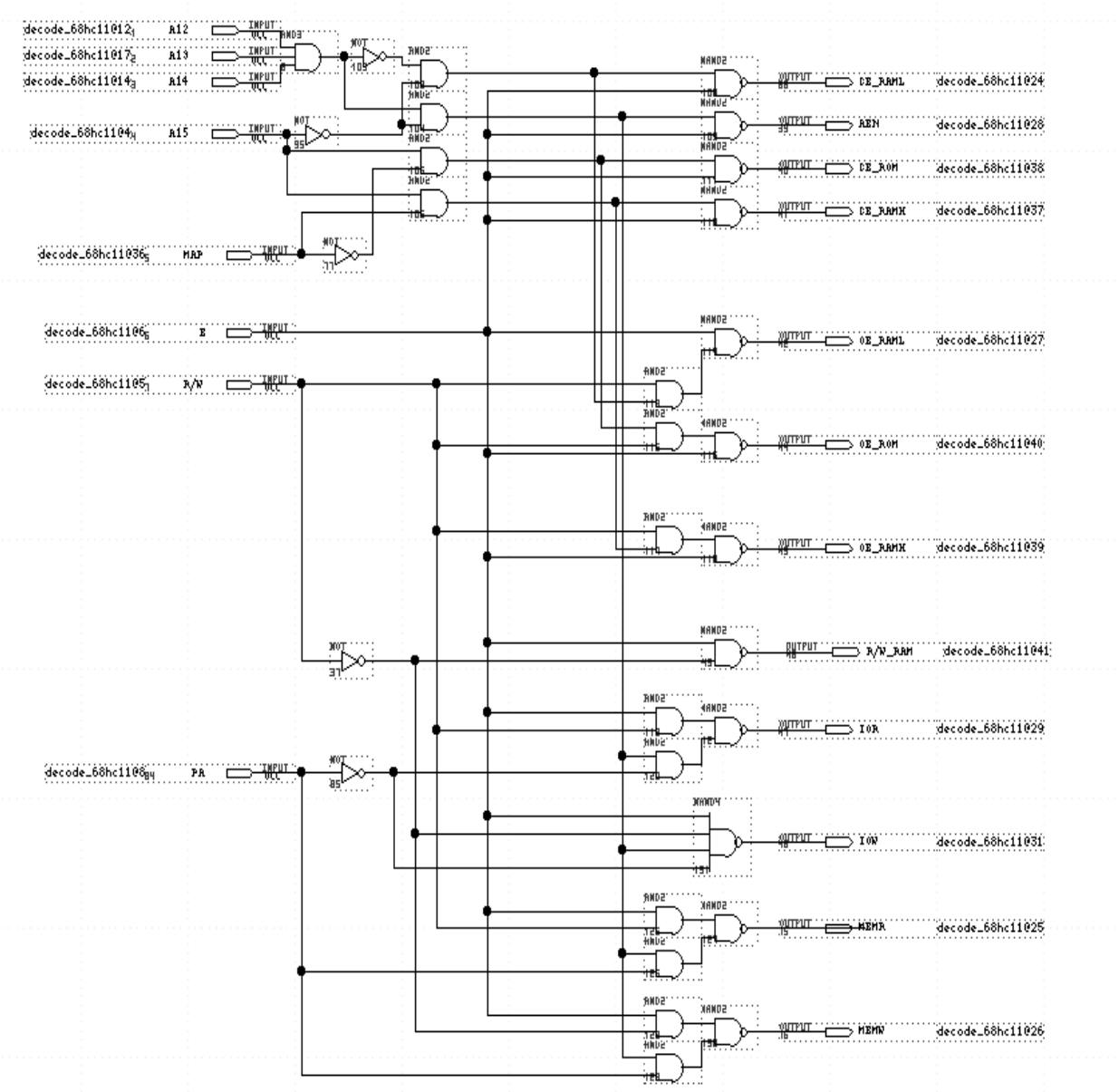
MAP	A15	A14.A13. A12	RAML	Ether	ROMH	RAMH	B	A
0	0	0	1	0	0	0	0	0
0	0	1	0	1	0	0	0	1
0	1	0	0	0	1	0	1	1
0	1	1	0	0	1	0	1	1
1	0	0	1	0	0	0	0	0
1	0	1	0	1	0	0	0	1
1	1	0	0	0	0	1	1	0
1	1	1	0	0	0	1	1	0

En posant G = A14.A13.A12, on obtient les équations suivantes pour la sélection des composants:

RAML = A15* . G*
CSEther = A15* . G
ROMH = MAP*. A15
RAMH = MAP . A15

Et pour un décodage démultiplexé on aura : A = A15*.G + MAP* . A15
B = A15

Quant au circuit logique câblé :

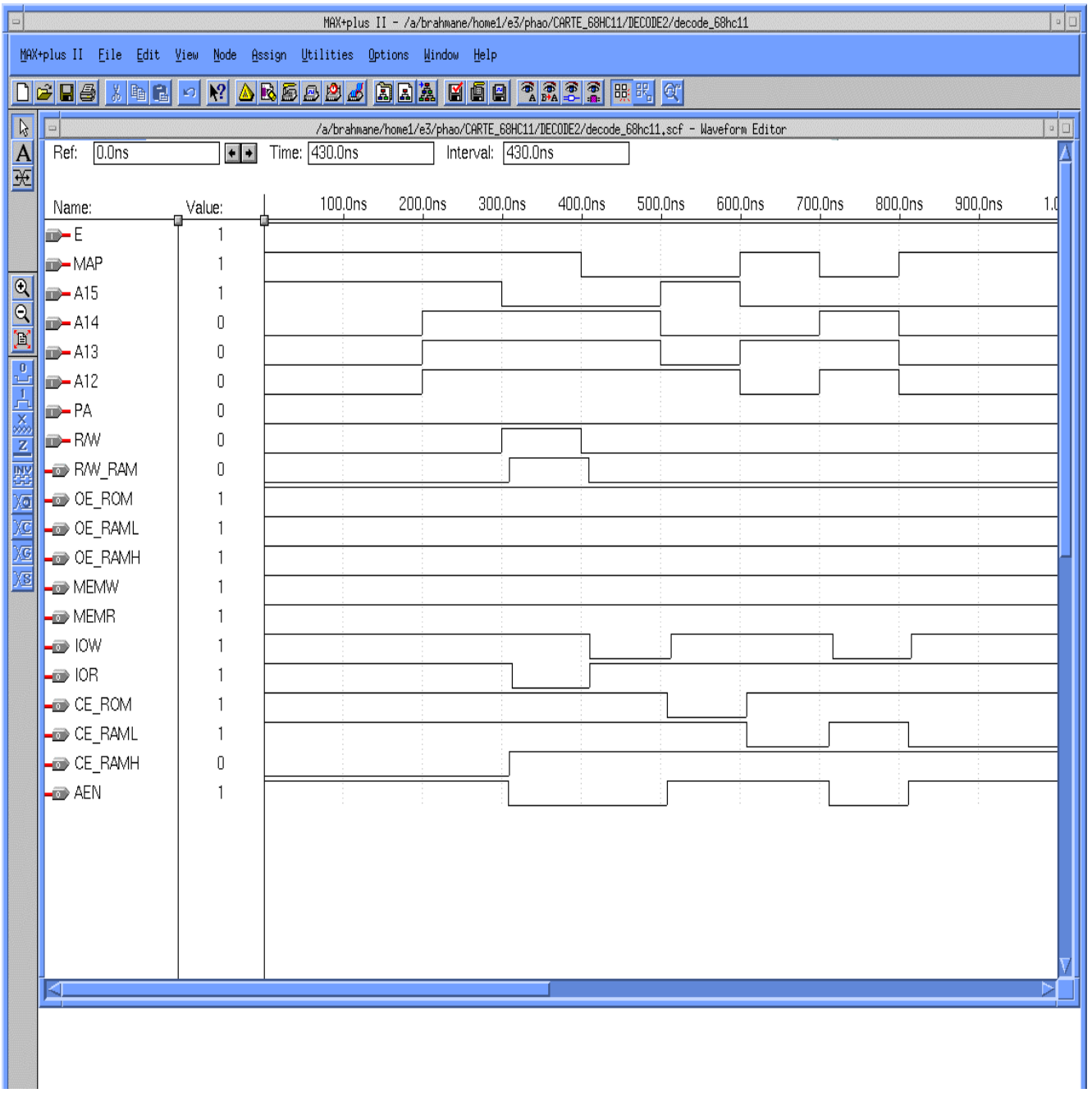


Les sorties CE (*Chip Enable*) sont les combinaisons des signaux d'adressage, de MAP et de E qui est un signal de synchronisation des échanges de données du 68HC11 en mode étendu.

Les sorties OE *(Ouput Enable*) évitant les problèmes de contention sur le bus de données sont générées à partir des CE et du signal de lecture.

Suivant l'état du signal PA6 (port A6 du 68HC11), ceux sont soient les signaux MEM soient les signaux IO qui sont générés pour les accès en écriture ou en lecture du CS8900A.

La simulation donne :



Cette simulation suppose E toujours actif et PA toujours à 0 (c'est-à-dire que le mode IO du CS8900A est enclenché). On voit que pour pouvoir écrire ou lire correctement il faut obligatoirement que les Chips Enable ou les Output Enable soient déclenchés suivant le cas concerné.

Câblage :

Cette logique câblée est programmée dans un MAX7032 de chez ALTERA, voici les correspondances entre les signaux à générer et les pins du circuit ALTERA :

GND	1	VCC	35	CE_RAML	24	MEMR*	25
GND	2	A12	12	OE_RAML	27	MEMW*	26
GND	43	A13	17	CE_RAMH	37	AEN*	28
GND	44	A14	14	OE_RAMH	39	IOR*	29
GND	10	A15	4	RW_RAM	41	IOW*	31
GND	22	PA6	8	OE_ROM	40		16
GND	30	MAP	36	CE_ROM	38		18
GND	42	RW	5		9		19
VCC	3	E	6		11		20
VCC	15		7		13		21
VCC	23		33		34		32

6) Utilisation du CS8900A :

Le chip CS8900A est optimisé dans sa conception pour un câblage sur des bus de type ISA

Concernant notre cas de figure, le CS8900A doit pouvoir fonctionner en mode 8 bits (adressage des données sur le port C du 68HC11), et connecté à bus système qui n'est pas ISA. Il faut donc trier les signaux susceptibles d'interfacer correctement le chip avec le microcontrôleur 68HC11 :

- Les 8 bits de données du chip SD0…SD7 sont reliés au port C du 68HC11.
- SD8…SD15 sont à la masse.(puisqu'on est en données 8 bits)
- Les adresses SA0…SA11 (adressage jusqu'à 0FFF comprend facilement les 4Ko de données du chip Ethernet) connectées au bus d'adresse du 68HC11. (voir Packet Page Adresses)
- SA12…SA19 sont à la masse.
- ELCS* à la masse car LA non utilisé.
- EECS, EESK et EED Out non utilisés car pas de ROM.
- EEDI*, CHIPSEL* la masse.
- DMACK* en VCC car pas de DMA.
- CSOUT* non utilisé car pas de ROM de Boot.
- DO+/-, CI+/-, DI+/- non utilisés car l'AUI non utilisé.
- SLEEP à VCC car pas de Hardware Sleep ni Standby.
- RESET actif bas donc nécessité d'un inverseur à l'entrée.
- SBHE* à VCC car pas de transfert sur octet haut.
- IOCHRDY* non utilisé.
- INTRQ0 utilisé pour le mode MEM.
- AEN* sert de chip select du CS8900A. C'est le MAX7032 qui génère ce signal.
- MEMW*, MEMR*, IOR*, IOW* sont les signaux de lecture et d'écriture sur le chip suivant son mode de fonctionnement. C'est le MAX7032 qui génère ces signaux.
- LANLED* et LINKLED* sont reliés à des leds lumineuses.
- XTAL1,XTAL2 sont les broches pour relier le quartz de 20MHz.

- MEMCS16* et IOCS16* non utilisés car on est en mode 8bits.
- RES connecté à une résistance de 4.99K vers la masse.
- BSTATUS* non utilisé
- TEST* non utilisé car pas de Boundary Scan Test.
- DVDD à 5V numérique
- DVSS à la masse numérique.
- AVDD à 5V analogique.
- AVSS à la masse analogique.

7) Schéma électrique final avec interface Ethernet :

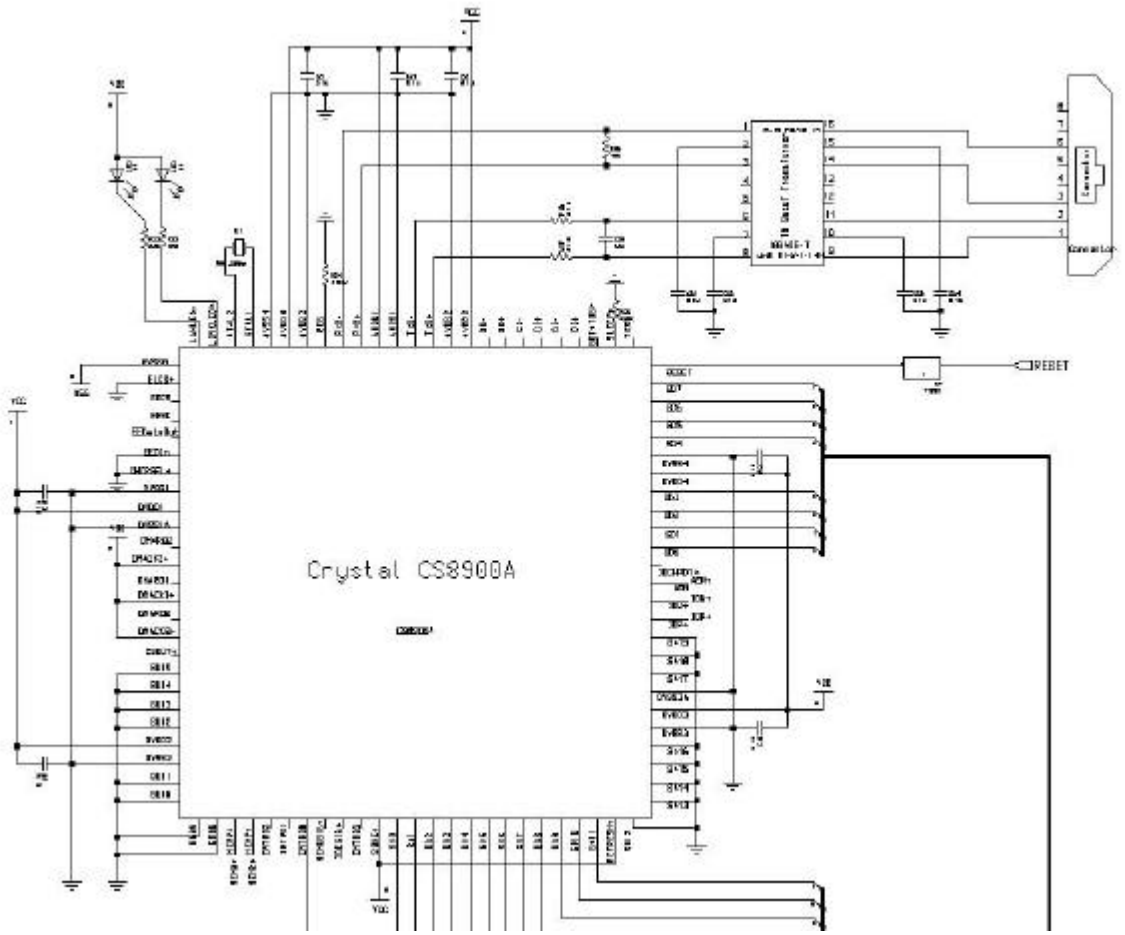


Schéma du circuit final avec gros plan sur le chip CS8900A.

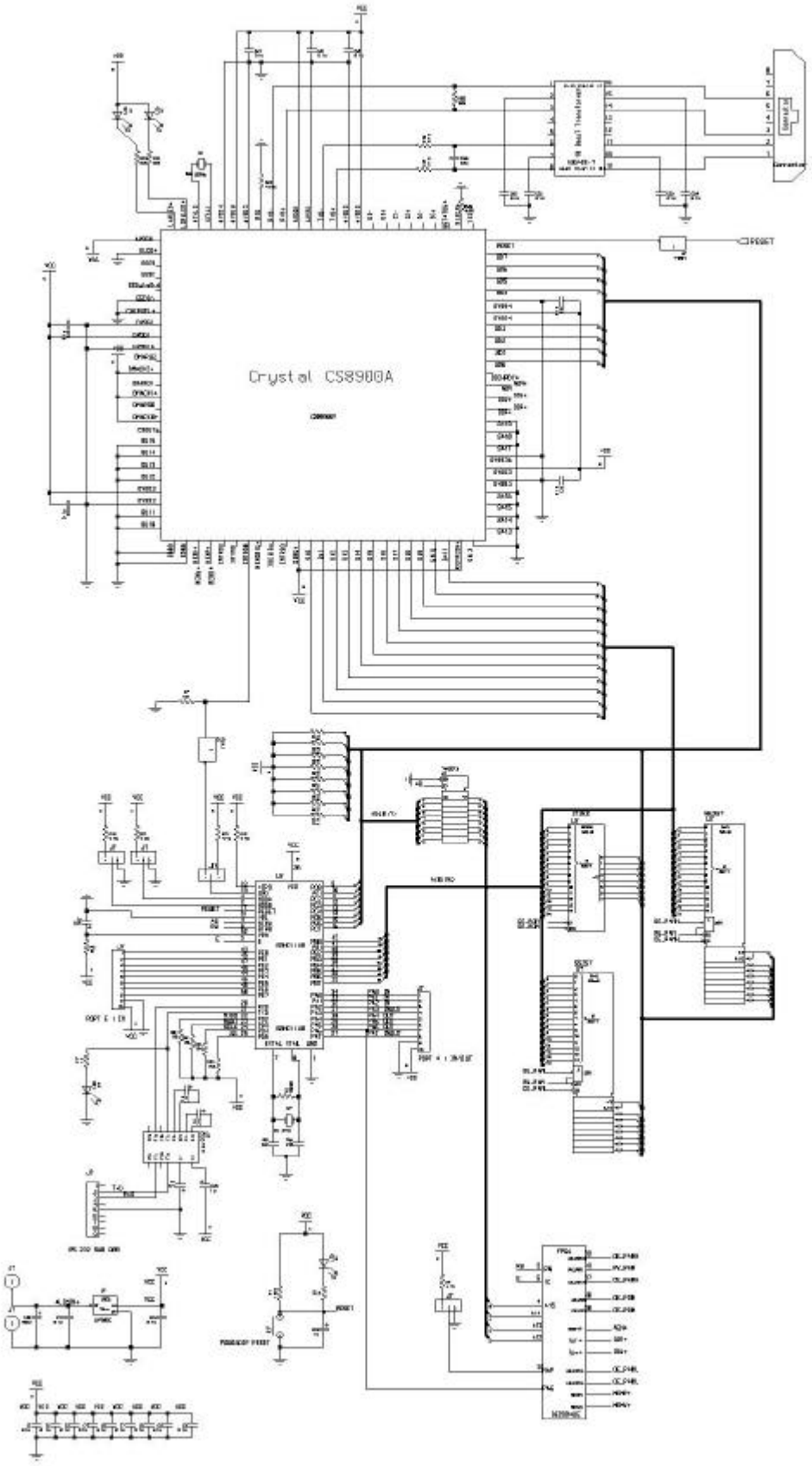


Schéma d'ensemble avec le microcontrôleur et toutes ces ressources.

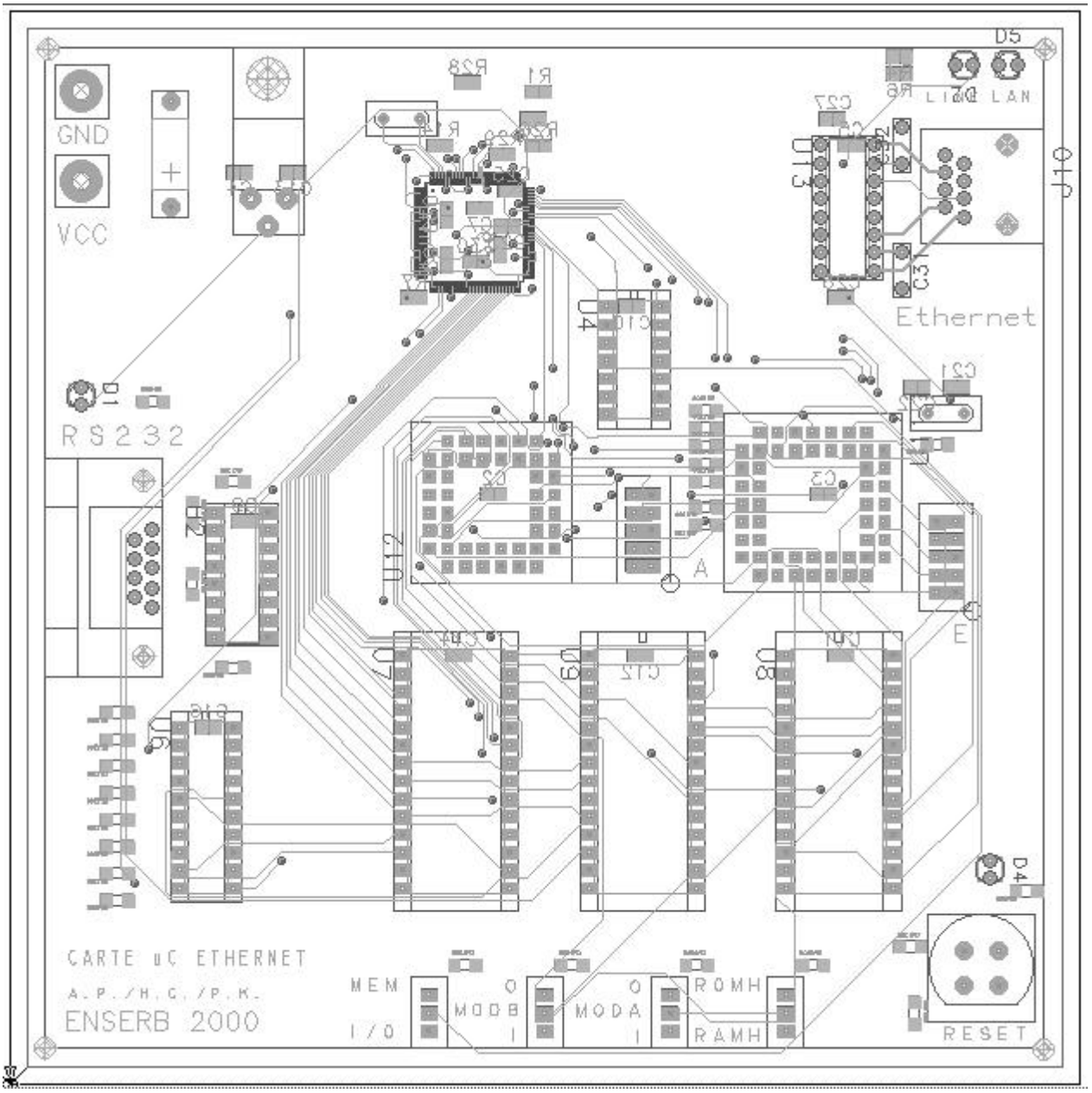
8) Routage / Nomenclature :

Les 2 schémas qui suivent sont les 2 couches de la carte à une échelle agrandie. On voit la complexité des tracés de piste et malgré les performances du routage automatique, une bonne partie des pistes a du être retravaillée manuellement.

Le routage a été effectué en classe 4, ce qui impose les valeurs théoriques de conception:

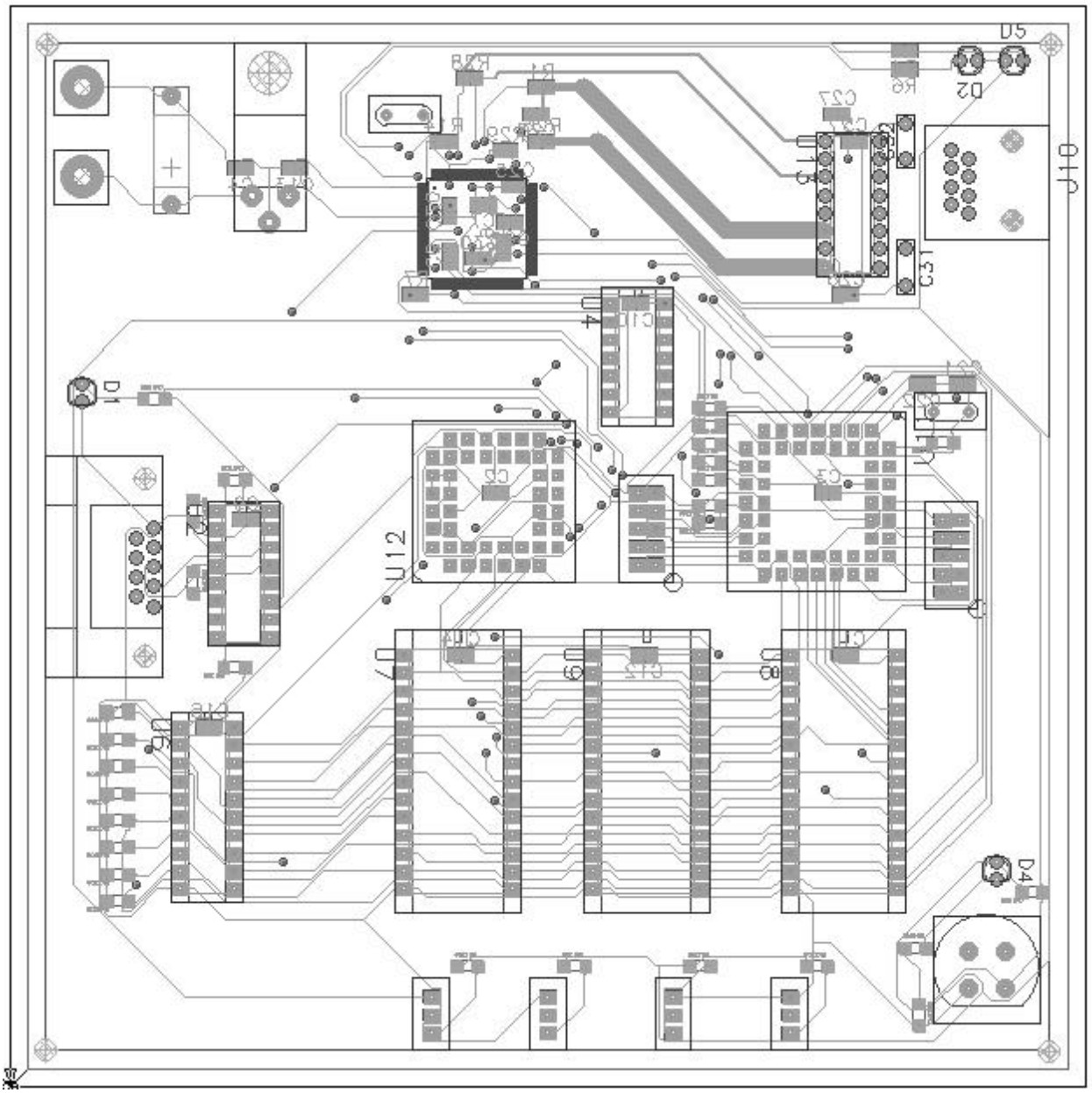
Pour des trous métallisés :

- Largeur des pistes : 0.005 inches
- Clearance : 0.008 inches



Coté composant – artwork1 -

Ecole **N**ationale **S**upérieure d' **E**lectronique et de **R**adioélectricité de **B**ordeaux



Coté cuivre – artwork2 -

Des géométries ont été crées pour les composants suivants : CS8900A, Connecteur RJ45.
Le 68HC11A8 est un plcc52, l'ALTERA est un plcc44 et le transformateur d'isolement est un dip16.

Toutes la partie autour du CS8900A a été routée manuellement selon le manuel d'application AN83 du composant.
Les bus d'adresses ont été routés manuellement pour facilité l'automatique.
Les composants CMS ont été placés sur la partie cuivre, notamment les condensateurs de découplage qui doivent être proches des circuits intégrés.

LISTE DES COMPOSANTS :

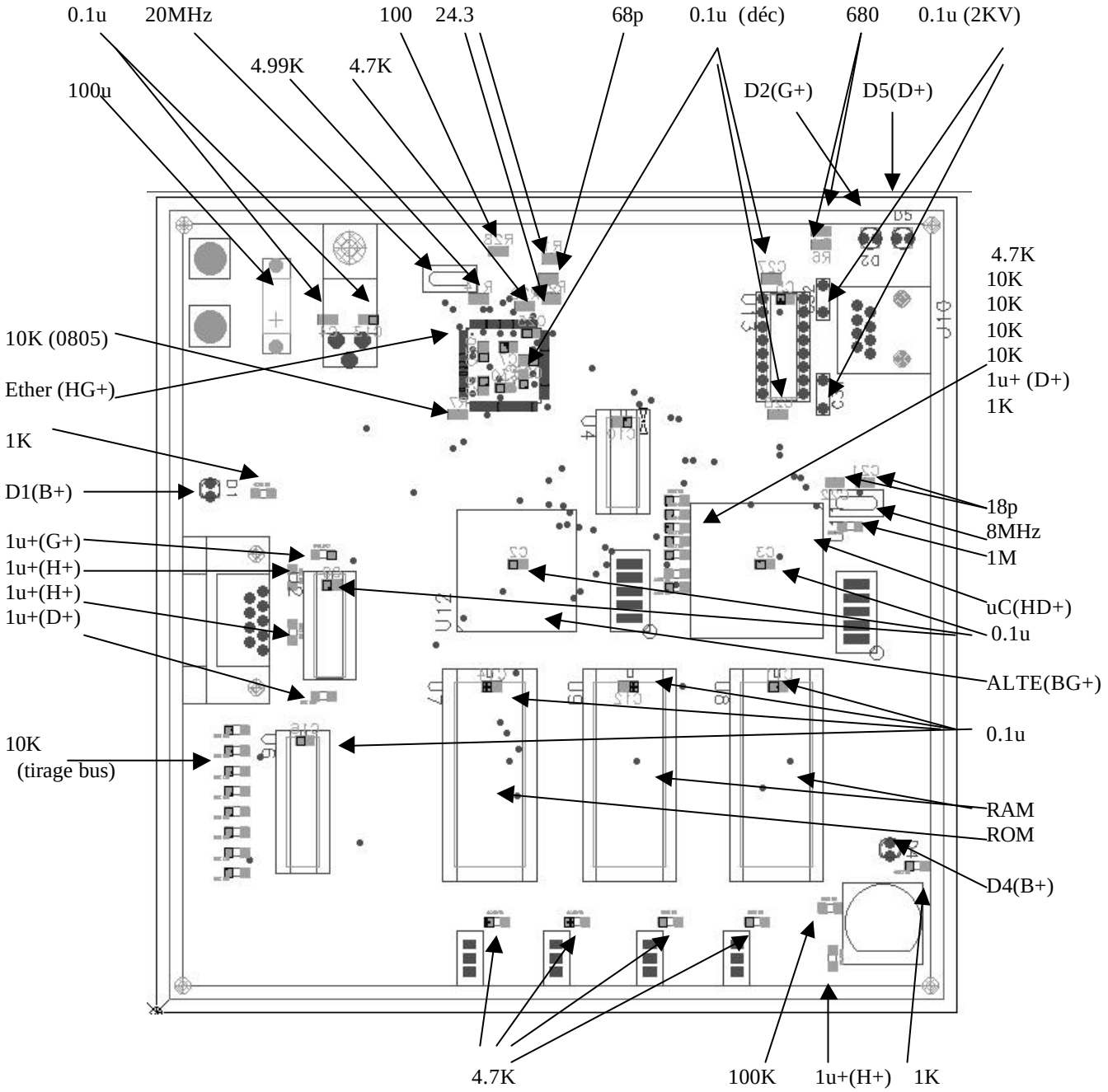
REFERENCE	ITEM_NUMBER	COMPANY PART NO.	GEOMETRY	DESCRIPTION
C1	16	pn-chim_11x5mm	c_chim_2	POL_CAPACITOR, 100u
C2	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C3	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C4	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C5	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C6	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C7	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C8	5	pn-1206pol	cr1206	POL_CAPACITOR, 1u
C9	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C10	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C11	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C12	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C13	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C14	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C15	5	pn-1206pol	cr1206	POL_CAPACITOR, 1u
C16	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C17	5	pn-1206pol	cr1206	POL_CAPACITOR, 1u
C18	5	pn-1206pol	cr1206	POL_CAPACITOR, 1u
C19	5	pn-1206pol	cr1206	POL_CAPACITOR, 1u
C20	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C21	10	pn-CC0805	CC0805	CAPACITOR, 18p
C22	10	pn-CC0805	CC0805	CAPACITOR, 18p
C23	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C24	5	pn-1206pol	cr1206	POL_CAPACITOR, 1u
C25	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C26	10	pn-CC0805	CC0805	CAPACITOR, 68p
C27	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C28	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C29	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C30	10	pn-CC0805	CC0805	CAPACITOR, 0.1u
C31	1	PN-CK05BX100K-CK	CK05	CAPACITOR, 10p
C32	1	PN-CK05BX100K-CK	CK05	CAPACITOR, 10p
D1	20	pn-led_01	LED_3X5.5MM	LED
D2	20	pn-led_01	LED_3X5.5MM	LED
D4	20	pn-led_01	LED_3X5.5MM	LED
D5	20	pn-led_01	LED_3X5.5MM	LED
J1	22	pn-plot_02	plk1x240	plot
J2	22	pn-plot_02	plk1x240	plot
J3	18	pn-conn3_01	conn1x3	conn3
J4	19	pn-conn9_01	subd9f_2.54	conn9
J5	17	pn-conn10_02	conn2x5	conn10
J6	18	pn-conn3_01	conn1x3	conn3
J7	18	pn-conn3_01	conn1x3	conn3
J8	17	pn-conn10_02	conn2x5	conn10
J9	18	pn-conn3_01	conn1x3	conn3
J10	12	pn-Connector.map	RJ45	Connector
K1	23	pn-poussoir_01	sw_bp12mm	poussoir
R1	14	pn-RC0805	CC0805	RESISTOR, 24.3
R2	15	pn-RC1206	cr1206	RESISTOR, 1K
R3	15	pn-RC1206	cr1206	RESISTOR, 4.7K
R4	14	pn-RC0805	CC0805	RESISTOR, 680
R5	15	pn-RC1206	cr1206	RESISTOR, 4.7K
R6	14	pn-RC0805	CC0805	RESISTOR, 680
R7	14	pn-RC0805	CC0805	RESISTOR, 10K
R8	15	pn-RC1206	cr1206	RESISTOR, 10K
R9	15	pn-RC1206	cr1206	RESISTOR, 10K
R10	15	pn-RC1206	cr1206	RESISTOR, 10K
R11	15	pn-RC1206	cr1206	RESISTOR, 4.7K
R12	15	pn-RC1206	cr1206	RESISTOR, 10K
R13	15	pn-RC1206	cr1206	RESISTOR, 4.7K
R14	14	pn-RC0805	CC0805	RESISTOR, 4.99k
R15	15	pn-RC1206	cr1206	RESISTOR, 10MEG
R16	15	pn-RC1206	cr1206	RESISTOR, 100K
R17	15	pn-RC1206	cr1206	RESISTOR, 1K
R18	15	pn-RC1206	cr1206	RESISTOR, 10K
R19	15	pn-RC1206	cr1206	RESISTOR, 10K
R20	15	pn-RC1206	cr1206	RESISTOR, 10K
R21	15	pn-RC1206	cr1206	RESISTOR, 10K
R22	15	pn-RC1206	cr1206	RESISTOR, 10K

R23	15	pn-RC1206	cr1206	RESISTOR, 10K
R24	15	pn-RC1206	cr1206	RESISTOR, 10K
R25	15	pn-RC1206	cr1206	RESISTOR, 10K
R26	14	pn-RC0805	CC0805	RESISTOR, 24.3
R27	15	pn-RC1206	cr1206	RESISTOR, 4.7K
R28	14	pn-RC0805	CC0805	RESISTOR, 100
R29	14	pn-RC0805	CC0805	RESISTOR, 4.7K
R30	15	pn-RC1206	cr1206	RESISTOR, 1K
U1	9	pn-78xx_hor	to220h	78xx
U2	21	pn-max232	DIP16_P	max232
U3	24	pn-quartz_01	hc18	quartz, 20MHz
U4	2	PN-SN74F04N-DIP	DIP14_P	74F04
U5	24	pn-quartz_01	hc18	quartz, 8MHz
U6	3	PN-SN74HC573N-DIP	DIP20_P	74HC573
U7	6	pn-27256	DIP28W_P	27256
U8	7	pn-55257	DIP28W_P	55257
U9	7	pn-55257	DIP28W_P	55257
U10	11	pn-CS8900A.map	tqfp100	CS8900A
U11	8	pn-68HC11A8.map	plcc52	68HC11A8
U12	13	pn-FPGA.map	plcc44	FPGA
U13	4	pn-10BASE-T.map	DIP16_P	10BASE-T

Liste des Composants sous une autre forme – Tri par quantité - :

Composant	**Valeur**	**Quantité**	**Géométrie**	**Référence**
R	4.99K	1	0805	215-3162
R	680	2	0805	347-9897
R	24.9	2	0805	203-9987
R	100	1	0805	347-9780
R	4.7K	1	0805	348-0005
C	68pF	1	0805	237-6876
R	10K	1	0805	
R	10K	12	1206	348-0730
R	4.7K	5	1206	348-0689
R	1K	3	1206	348-0588
R	1M	1	1206	348-0998
R	100K	1	1206	348-0853
Cpol	1uF	6	1206	334-9522
C	18pF	2	0805	237-6797
C	0.1uF	20	0805	
C	0.1uF(2kV)	2	CK05	
Cpol	100u	1	c_chim_2	
Quartz	20Mhz (0.01%)	1	hc18	
Quartz	8Mhz	1	hc18	
Max7032	LC44-12	1	plcc44	
UC 68HC11	A8/E1	1	plcc52	
RAM	32k x 8 bits	2	DIP28W_P	
ROM	32k x 8 bits	1	DIP28W_P	
CS8900A		1	tqfp100	
Trsf. Isol.	10Base T	1	DIP16_P	
Max232		1	DIP16_P	
Régulateur	7805	1	to220h	
Inverseur	7404N	1	DIP14_P	
Latch 8 bits	74HC573N	1	DIP20_P	
Poussoir		1	sw_bp12mm	
LED Lum.	3x5.5mm	4	LED_3x5.5MM	
Conn Alim.		2	p1k1x240	
Interrupteur		4		
Conn Série		1	subd9f_2.54	
Conn RJ45		1	RJ45	

Emplacement des Composants sur la carte :



Notes :

Les résistances sont en 1206 sauf la 10K près du CS qui est une 0805.
Pour les capacités polarisées la notation 1u+(D+) signifie 1 micro polarisé à droite – H : haut, B :bas et G : gauche – Idem pour les LEDS lumineuses indiquant le méplat ainsi que les gros chip . Les DIP sont tous H+.

9) Soudure/ Débuggage Hardware :

La détection des erreurs sur la carte commence par une vérification de toutes les soudures des composants CMS, des valeurs des composants implantés, des tests de continuité des pistes de la carte et la vérification d'un éventuel court-circuit.

La deuxième étape consiste à vérifier la validité du microcontrôleur dans ses divers mode de fonctionnement, en particulier les modes BootStrap et Extended. Pour cela on a recours aux moniteurs PCBUG11 et BUFFALO pour bien adresser directement le composant ou bien charger des fichiers S_Record .s19 programme vers la RAM du chip.

PCBUG11 et BUFFALO : Etapes pour exécuter un fichier avec la carte TP :

Commande :	**Notepad** pour composer un fichier source de type : .ASC (Assembleur 68HC11)	ou bien	.H et .C (langage C)
Commande :	asmhc11.exe		**cx6811.exe** (Compilation) .H .C
Résultat :	.LST et .S19		.O
Commande :			**clnk.exe** (linkage) .LKF
Résultat :			.H11 et .MAP
Commande :			**chex.exe** (conversion) .H11
Résultat :			S_Record .S19
Commande :			**cobj.exe** (inspection section) .H11
Résultat :			écran

Remarque : Pour l'aide sur la commande : > **Commande / ?**

Utilisation Rapide de PCBUG11 :

- UC. En mode BOOTSTRAP (modA = 0, modB = 0) et faire Reset poussoir avant de lancer PCBUG11.
- Validation des accès aux ressources externes : **BF 103C E5** (Valeur E5 dans le registre 103C)
- Basculer au mode DOS : Taper **DOS** au prompt ; puis EXIT pour sortir et revenir au display.
- Charger un fichier .s19 : Taper **LOADS** Nom_du_fichier_sans_l_extension_s19 .
- Vérifier le contenu de la mémoire : **MD** Adresse_Départ Adresse_Fin. (Memory Dump).
- Remplir une case Mémoire : **BF** Adresse_Départ <Adresse_Fin> Valeur. (Block Fill).
- Saut pour exécuter le programme : **G** Adresse. (Go).
- Redémarrage de PCBUG : **RESTART**. – Ne pas oublier la validation des accès externes si nécessaire ! ! ! -

Utilisation Rapide de BUFFALO:

- UC. En Mode Extended (modA = 1, modB = 1), lancer HYPERTERMINAL puis reset poussoir.
- Télécharger un fichier .s19 : Taper au prompt **LOAD T**, puis avec le menu >Transfert>Envoyer_fichier_Texte>
- Menu d'aide : **HELP**.
- Memory Dump : **MD** Adr1 Adr2.
- Block Fill : **BF** Adr1 Adr2 Valeur.
- Saut à l'instruction : **GO** Adresse.

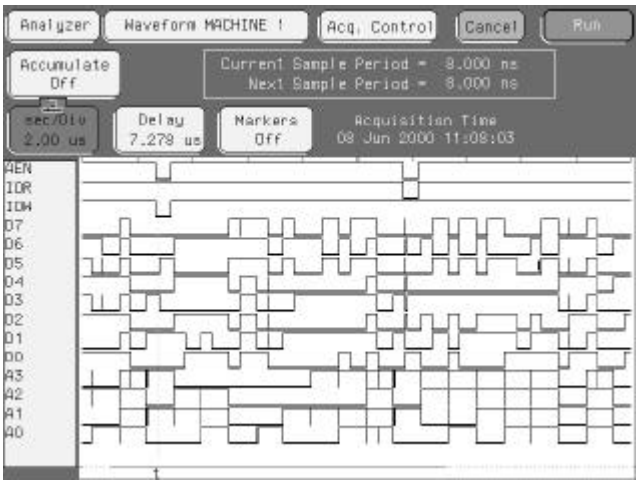
<<< Voir la partie Software pour l'exemple. >>>

La troisième étape consiste à écrire dans le CS8900A pour vérifier son fonctionnement conforme.

Impressions des premiers « Débuggages Hards » :

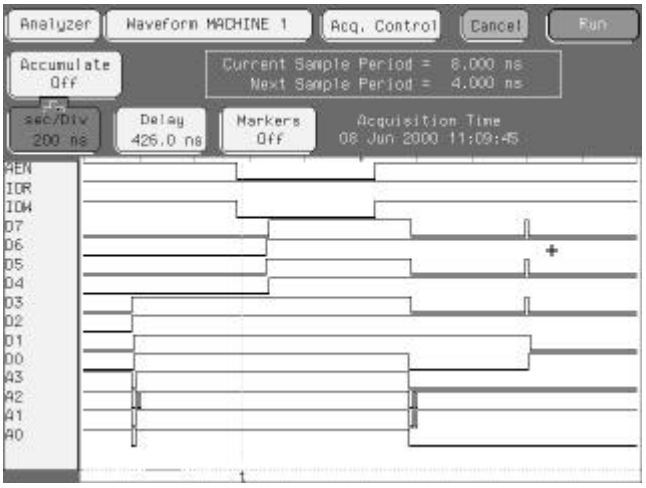
- La partie microcontrôleur a démarré avec succès aussi bien en BootStrap avec PCBUG11 qu'en Extended avec BUFFALO – Validation de la ROM -.
- Les écritures sur les RAM basses et hautes ne posent pas de problèmes. – validation des accès RAM –
- Analyse des signaux vers le CS8900A – validation des signaux vers le CS8900A, décodage d'ensemble correct -
- Précaution pour les accès au CS8900A, les données doivent être stable à chaque requête de lecture ou d'écriture.

Visualisation des signaux de contrôle, bus de données et quartet bas vers le CS8900A avec l'analyseur numérique :



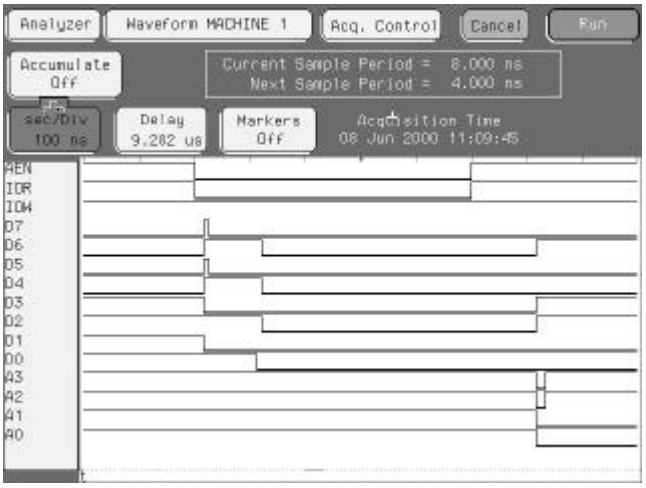
Ecriture de $22 en $730A avec BUFFALO::

On voit une écriture de $22 en $730A.
Et une lecture à cette adresse de $01.



Ecriture de $FF en $730F avec BUFFALO::

On voit une écriture de $FF en $730F.



Lecture de l'adresse $730F avec BUFFALO::

On voit une donnée égale à $00.

II) PARTIE SOFTWARE

1) Tests sous PCBUG11:

Au reset de la carte, le Dumping de la zone du chip Ethernet CS8900A situé en $7000 donne :

	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
7000	00	00	00	09	00	00	00	00	00	00	30	63	0E	63	0E

- On essaie ensuite d'écrire dans les registres du CS8900A afin de transmettre un trame quelconque :

7304 ← C9 On force les bits TxStart du registre 108h TxCMD. Poids faible.
7305 ← 00 Pods fort.

7306 ← 51 Longueur de la trame (81 decimal) dans le registre TxLENGTH. Poids faible.
7307 ← 00 Poids fort.

730A ← 38 Adresse de BusST. Poids faible.
730B ← 31 Poids fort.

Résultat :
730C → 18 Lecture de BusST. Poids faible.
730D → 00 ou 01 si Rdy4TxNow (Transnission possible visible sur bit 8 de BusST)

- Reset Software Manuel:

Il faut forcer le bit 6 de SelfCTL en 114h. Donc la procédure est:

730A ← 14 Poids fort de l'adresse de SelfCTL.
730B ← 31 Poids faible.

730C ← 55 Bit 6 forcé à 1 + séquence figée soit 0101 0101.

Immédiatement on retrouve le DUMP en $7300 qui est:

	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
7000	00	00	00	09	00	00	00	00	00	00	30	63	0E	63	0^E

2) Programmes de Test :

Fichier CRT.S : Assembleur 68HC11 : kit de démarrage.

```
;       C STARTUP FOR MC68HC11
;       Copyright (c) 1995 by COSMIC Software
;
        xdef   _exit, __stext
        xref   _main, __memory, __stack
;
        switch .bss
__sbss:
        switch .text
__stext:
        clra                ; reset the bss
        ldx    #__sbss      ; start of bss
        bra    loop         ; start loop
zbcl:
        staa   0,x          ; clear byte
        inx                 ; next byte
loop:
        cpx    #__memory    ; up to the end
        bne    zbcl         ; and loop
        lds    #__stack     ; initialize stack pointer
        jsr    _main        ; execute main
_exit:
        bra    _exit        ; and stay here
;
        end
```

Fichier .LKF : Linkage.

```
#       link command file for test program
#       Copyright (c) 1995 by COSMIC Software
#
+seg .text -b 0x2000 -n .text   # program start address
+seg .const -a .text            # constants follow program
+seg .data -b 0x4000            # data start address bss
crts.o
iotsttx.o                        # application program
libhc11.o                       # 68hc11 enserb lib
c:/logiciel/68hc11/cx32/lib/libi.h11            # C library
c:/logiciel/68hc11/cx32/lib/libm.h11            # machine library
+def __memory=@.bss             # symbol used by library
+def __stack=0x00ff             # stack pointer initial value
```

Fichier .BAT: Script exécutable.

```
@echo off
cx6811 -v -e crts.s iotsttx.c
if errorlevel 1 goto bad
:clink
echo.
echo Linking ...
clnk -o iotsttx.h11 -m iotsttx.map iotsttx.lkf
if errorlevel 1 goto bad
:chexa
echo.
echo Converting ...
chex -o iotsttx.s19 iotsttx.h11
if errorlevel 1 goto bad
:cobj
echo.
echo Section size ...
cobj -n iotsttx.h11
if errorlevel 1 goto bad
echo.
echo.
echo        OK.
goto sortie
:bad
echo.
echo        BAD.
:sortie
```

Fichier Source .C : Code programme.

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>

#define IO_BASE 0x7300
#define PP_POINTER 0x000a
#define PP_DATA 0x000c
#define NO_FRAMES_TO_SEND 1000
#define SIZE_OF_FRAME    100
#define WORD_DATA 0x1234
#define TX_CMD 0x0004
#define TX_LENGTH 0x0006
#define SUCCESS 0
#define FAILURE 1
#define SKIP 0040



main();
void reset();
inport();
outport();
delay();

main()
{

       int i,j,receive_frame_size, temp;

/*printf("Debut tests...\n");
*/
   /* Initialize the part */

        /* Configure RxCTL for Promiscious mode, RxOK */
        /*(1) Write 0x0104 to IOBase+0xA (PacketPage Pointer)
        (2) Write 0x0180h to IOBase+0xC (PacketPage Data Port)

        /* Configure TestCTL (FDX, DisableLT) */
        /*(3) Write 0x0118 to IOBase+0xA (PacketPage Pointer)
        (4) Write 0x4080 to IOBase+0xC (PacketPage Data Port)

        /* Set 10BaseT, SerRxOn, SerTxOn in LineCTL */
        /*(5) Write 0x0112 to IOBase+0xA (PacketPage Pointer)
        (6) Write 0x00C0 to IOBase+0xC (PacketPage Data Port)
        */
/*   printf("Initialisation...");
*/   outport( (IO_BASE + PP_POINTER ), 0x0104 );
   outport( (IO_BASE + PP_DATA ), 0x0180 );
   outport( (IO_BASE + PP_POINTER ), 0x0118 );
   outport( (IO_BASE + PP_DATA ), 0x4080 );
   outport( (IO_BASE + PP_POINTER ), 0x0112 );
   outport( (IO_BASE + PP_DATA ), 0x00c0 );
/*   printf("OK\n");
*/
   for (i = 0; i < NO_FRAMES_TO_SEND ; i++)
   {
      /* Write the TX command */
               /*(7) Write 00C0h to IOBase+0x4 (PacketPage Pointer)

               /* Write the frame length  (XX = number of bytes to TX) */
               /*(8) Write 00XXh to IOBase+0x6 (PacketPage Pointer)

               /* Read BusST to verify it is set as 0x0118 (Rdy4TxNow is set) */
               /*(9) Write 0x0138 to IOBase+0xA (PacketPage Pointer)
               (10) Read IOBase+0xC  Should read 0x0118
      */

/*      printf("Emission trame %d...",i);
 */     outport( (IO_BASE + TX_CMD ), 0x00c0 );
      outport( ( IO_BASE + TX_LENGTH ), SIZE_OF_FRAME * 2 ); // bid for bytes
      outport( (IO_BASE + PP_POINTER), 0x0138 );
      delay(25);
      if (inport( ( IO_BASE + PP_DATA ) ) !=  0x0118)
```

```
      {
   /*         printf("\n\n Bid for transmit failed");
     */        return(FAILURE);
      }

      /* Copy the TX frame to the on-chip buffer */
      /*(11) Write XX/2 words to IOBase+0x0 (TX Data Port) */

      for (j = 0; j < SIZE_OF_FRAME; j++)
      {
       outport( IO_BASE , WORD_DATA );
      }
/*   printf("OK\n");
 */  }
   reset();

/*   printf("\n\n Tx Test Complete");
 */  return(SUCCESS);
}


void reset()
{
   outport( ( IO_BASE + PP_POINTER ), 0x0114 );
   outport( ( IO_BASE + PP_DATA ), 0x0040 );
   delay(25);
}

delay(int time)
{
int i;
for(i=0;i<0x1000*time;i++)
       ;
}

outport(int commande, int data)
{
char datal, datah;
char *ptrl, *ptrh;

datal = data & 0xff;
datah = (data >> 8) & 0xff;

ptrl = (char *)commande;
ptrh = (char *)(commande+1);
*ptrl = datal;
*ptrh = datah;
}


inport(int commande)
{
char datal, datah;
char *ptrl, *ptrh;
int data;

ptrl = (char *)commande;
ptrh = (char *)(commande+1);
datal = *ptrl;
datah = *ptrh;

data = ((datah << 8) + datal)& 0xffff;

return(data);
}
```

3) Programmes plus élaborés:

Transmission d'une trame

Avant tout fonctionnement du CS8900A, on effectue une initialisation des registres de contrôle de transmission RxCTL, TestCTL et LineCTL.
Une fois tout ces registres configurés, on écrit dans le registre TxCMD la valeur 0x00C0, ce qui permet la transmission après que tout le bits soient écrit dans le mémoire tampon du CS8900A, on écrit ensuite la longueur de la trame à transmettre dans le registre TxLENGTH dont l'adresse est 0x7306. On configure le registre BusST pour permettre la transmission, et enfin on effectue une série d'écriture sur les registres 0x7300 puis 0x7301 de la trame à transmettre. Une fois finie, le composant effectue la transmission automatique de trame.

Algorithme de transmission

```
#define IO_Base_Low     0x7300      Registre Réception/Transmission de données
#define IO_Base_High    0x7301      Registre Réception/Transmission de données

#define TxCMD           0x7304      Registre commande de Transmission
#define TxLength        0x7306      Registre de la longueur de la trame
#define RxCTL           0x7404      Registre contrôle de Réception/Transmission
#define LineCTL         0x7412      Registre configuration de circuit de configuration
#define TestCTL         0x7418      Registre configuration du 10Base-T
#define BusST           0x7438      Registre statut de la Transmission

#define Data            0x6500      Adresse des données à transmettre


Reception{

        int i,Size_Of_Frame=4;
        int tab={0,1,2,3,4,5,6,7,8,9}

        Write(RxCTL,0x0183);
        Write(TestCTL,0x4080);
        Write(LineCTL,0x00c0);

        Write(TxCMD,0x00c0);
        Write(TxLENGTH,Size_Of_Frame*2);
        While(Valeur_Bit_8(BusST) != 1){
                ;}

        For(i=0;i<Size_Of_Frame;i++){
                Write(IO_Base_Low,Data+2i);
                Write(IO_Base_High,Data+2i+1);
                }
        }
end
```

NB : la fonction Write(adresse, données) permet d'effectuer une écriture.

Réception d'une trame

Avant tout fonctionnement du CS8900A, on effectue une initialisation des registres de contrôle de transmission RxCTL, TestCTL, LineCTL et BufCFG.
Une fois tout ces registres configurés, on scrute le registre RxEvent indiquant l'état du CS8900A (attente/réception), puis ensuite on lit le statut/longueur_trame/données par une suite de lecture 8bits des registres d'adresses 0x7300 et 0x7301.

Algorithme de réception

```
#define IO_Base_Low     0x7300      Registre Réception/Transmission de données
#define IO_Base_High    0x7301      Registre Réception/Transmission de données

#define RxCTL           0x7404      Registre contrôle de Réception/Transmission

#define BufCFG          0x740a      Registre configuration du buffer
#define LineCTL         0x7412      Registre configuration de circuit de configuration
#define TestCTL         0x7418      Registre configuration du 10Base-T
#define RxEvent         0x7424      Registre statut de la réception

#define Memory          0x6000      Adresse de stockage de la trame reçue

reception{
        int i,Size_Of_Frame,RxStatusRegister ;
        char Data_High,Data_Low ;

        Write(RxCTL,0x0183);
        Write(TestCTL,0x0099);
        Write(LineCTL,0x0053);
        Write(BufCFG,0x000B);

        While (RxEventRegister != 0x0104){
                        ;}

        Read(RxStatusRegister_High,IO_Base_High);
        Read(RxStatusRegister_Low,IO_Base_Low);

        Read(Size_Of_Frame_High,IO_Base_High);
        Read(Size_Of_Frame_Low,IO_Base_Low);

        For(i=0;i<Size_Of_Frame;i++){
                Read(Data_High,IO_Base_L);
                Read(Data_Low,IO_Base_High);
                Write((Memory+(2*i)), Data_High, Data_Low);
                }
        }
end
```

NB : RxEventRegister est le contenu du registre RxEvent
La fonction Read(registre, adresse) effectue une lecture du contenu de l'adresse et le stocke dans registre.
La fonction Write(adresse, donnée) effectue une écriture de donnée à l'adresse.

Transmission/Réception en utilisant l'interruption du microcontrôleur 68HC11

Le principe consiste à se mettre dans une première phase dans un mode de transmission pendant lequel le CS8900A envoie une trame dans le réseau Ethernet, puis ensuite une deuxième phase dans un mode de réception. Cette dernière phase est la plus intéressante puisqu'elle fonctionne sous interruption. En effet une fois le CS800A, ayant reçu une trame il envoie un signal d'interruption vers le microcontrôleur pour ainsi exécuter le programme associé au mode de réception.

Le Code source en C :

```
/*************************************/
/* Definition des ports I/O du CS8900A*/
/*************************************/
#define IO_BASE             0x7300
#define PP_RxTx             IO_BASE + 0x0000
#define PP_TxCMD            IO_BASE + 0x0004
#define PP_TxLength         IO_BASE + 0x0006
#define PP_ISQ              IO_BASE + 0x0008
#define PP_POINTER          IO_BASE + 0x000a
#define PP_DATA             IO_BASE + 0x000c

/*************************************/
/* Definition des Registres du CS8900A       */
/*************************************/

/* Controle et Configuration  */
#define RxCFG               0x0102
#define RxCTL               0x0104
#define TxCFG               0x0106
#define TxCMD               0x0108
#define BufCFG              0x010a
#define LineCTL             0x0112
#define SelfCTL             0x0114
#define BusCTL              0x0116
#define TestCTL             0x0118

/* Status et Evenements  */
#define ISQ                 0x0120
#define RxEvent             0x0124
#define TxEvent             0x0128
#define BufEvent            0x012c
#define RxMISS              0x0130
#define TxCOL               0x0132
#define LineST              0x0134
#define SelfST              0x0136
#define BusST               0x0138
#define TDR                 0x013c

/* Zone Memoire pour recuperer/envoyer */
#define RxBuffer            0x6000
#define TxBuffer            0x6500

#define SUCCESS             0
#define FAILURE             1

/*          authorize interrupts for 68HC11 */
#define cli()           _asm("cli\n")

/* Variables Globales */
int NB_FRAMES_TO_SEND = 50;
int SIZE_OF_FRAME_TX;
int SIZE_OF_FRAME_RX;
```

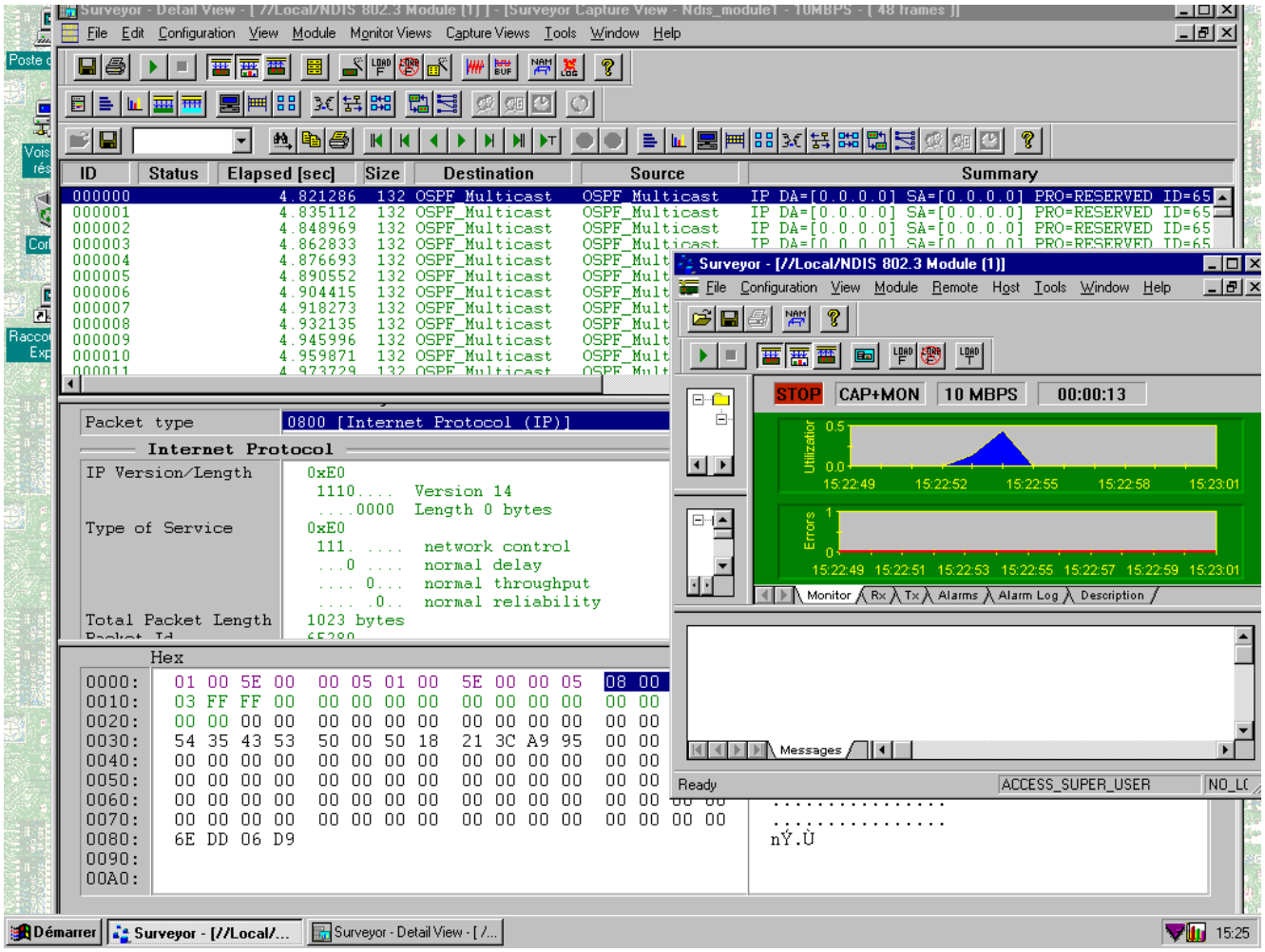
```
/*******************************/
/* Les fonctions de ce driver */
/*******************************/
void delay(int time);                           /* Temporisation  */
void initialisation(void);                      /* Initialisation de la ligne */
void reset(void);                               /* Reset Software Global */
void out16(int adresse,int donnee);             /* Ecriture 16Bits en $ */
int in16(int adresse);                          /* Lecture 16Bits en $  */
void out16Intel(int adresse,int donnee);        /* Ecriture 16Bits en $ */
int in16Intel(int adresse);                     /* Lecture 16Bits en $  */
void outReg16(int Registre, int Donnee);        /* Ecriture 16Bits en REG */
int inReg16(int Registre);                      /* Lecture 16Bits en REG  */

void IRQproc(void);                             /* Traitement de l IRQ    */
int receive_frame(int *data);                   /* Reception trame en RxBuffer    */
int transmit_frame(int Length,int *data);       /* Transmission trame de TxBuffer */


void delay(int time)
{
        int i;

        for(i=0;i<0x1000*time;i++)
                ;
}


void initialisation(void)
{
        /* Configuration de RxCTL (CRC, toutes $, filtre Hash OK)  */
        outReg16( RxCTL, 0x0183);

        /* Configuration du registre TestCTL : ( pas de LoopBack)  */
        outReg16(TestCTL, 0x0099);

        /* Configuration LineCTL (10BaseT,SerTxOn,SerRxOn)        */
        outReg16( LineCTL, 0x00D3);
        delay(25);

        /* Configuration de BufCFG */
        outReg16( BufCFG, 0x000B);
        delay(25);

        SIZE_OF_FRAME_TX = SIZE_OF_FRAME_RX = 0;
}


void reset(void)
{
        /* Bit 6 a 1 du registre SelfCTL en 114h */
        outReg16(SelfCTL, 0x0040);
        delay(25);
}


void init4IRQ(void) {

        /* Configuration BufCFG: (RxDestiE bit F) 800B*/
        /* outReg16( BufCFG, 0x800B); */

        /* Configuration RxCFG: (RxOiE)     */
        outReg16( RxCFG, 0x0103);

        /* Selection de l'interruption IRQ0 */
        outReg16(0x0022,0x0000);

        /* Configuration de BusCTL: (EnableIRQ)  */
        outReg16( BusCTL, 0x8017);

        delay(25);
}
```

```
void out16(int adresse,int donnee)
{
        char donnee_l,donnee_h;             /* poids faible, fort */
        char *ptrl = (char *)adresse;       /* pointeur sur poids faible */
        char *ptrh = (char *)(adresse+1);   /* pointeur sur poids fort   */

        donnee_l = donnee & 0xff;           /* Recup PFb */
        donnee_h = (donnee >> 8) & 0xff;    /* Recup PFt */

        /* ATTENTION a l ordre de lecture */
        *ptrl = donnee_l;                   /*Octet faible --> $ faible */
        *ptrh = donnee_h;                   /*Octet fort   --> $ fort   */
}


void out16Intel(int adresse,int donnee)
{
        char donnee_l,donnee_h;             /* poids faible, fort */
        char *ptrl = (char *)adresse;       /* pointeur sur poids faible */
        char *ptrh = (char *)(adresse+1);   /* pointeur sur poids fort   */

        donnee_l = donnee & 0xff;           /* Recup PFb */
        donnee_h = (donnee >> 8) & 0xff;    /* Recup PFt */
        *ptrl = donnee_h;                   /*Octet fort --> $ faible      */
        *ptrh = donnee_l;                   /*Octet faible   --> $ fort    */
}


int in16(int adresse)
{
        char donnee_l,donnee_h;             /* poids faible, fort */
        char *ptrl = (char *)adresse;       /* pointeur sur poids faible */
        char *ptrh = (char *)(adresse+1);   /* pointeur sur poids fort   */
        int donnee;

        donnee_l = *ptrl;                   /* Recup PFb de $ faible */
        donnee_h = *ptrh;                   /* Recup PFt de $ fort   */
        donnee = ( ( donnee_h << 8 ) + donnee_l ) & 0xffff;
        return(donnee);
}


int in16Intel(int adresse)
{
        char donnee_l,donnee_h;             /* poids faible, fort */
        char *ptrh = (char *)(adresse+1);   /* pointeur sur poids fort   */
        char *ptrl = (char *)adresse;       /* pointeur sur poids faible */
        int donnee;

        donnee_h = *ptrh;           /* Recup PFt de $ fort  */
        donnee_l = *ptrl;           /* Recup PFb de $ faible*/
        donnee = ( ( donnee_h << 8 ) + donnee_l ) & 0xffff;
        return(donnee);
}


void outReg16(int Registre, int Donnee)
{
        out16( PP_POINTER, Registre);
        out16( PP_DATA   , Donnee );
}


int inReg16(int Registre)
{
        int Donnee;

        out16( PP_POINTER, Registre);
        Donnee = in16(PP_DATA);
        return(Donnee);
}
```

```
int receive_frame(int *data)
{
        int i,var;

        /* (a) Lecture de RxEvent ou RxStatus */
        var = in16Intel(PP_RxTx);

        /* (b) Lecture de Length of Frame */
        SIZE_OF_FRAME_RX = in16Intel(PP_RxTx);

        /* Ecriture de la Length dans une zone mem */
        out16(0x4100,SIZE_OF_FRAME_RX);
        out16(0x4104,0xEA);

        /* (c) Recuperation de la trame entiere en RxBuffer */
        for (i=0; i<SIZE_OF_FRAME_RX; i++)
                {
                var = in16(PP_RxTx);            /* !!! Sens de Lecture   */
                out16( (int)&data[i],var);/* sauvegarde en zone data  */
                }

        return(SUCCESS);
}


int transmit_frame(int Length, int *data)
{
        int j,var;

        SIZE_OF_FRAME_TX = Length;

        /* (a) Ecriture de la commande TxCMD */
        out16(PP_TxCMD , 0x00c0 );

        /* (b) Ecriture de la longueur de la trame en OCTECT */
        out16(PP_TxLength , SIZE_OF_FRAME_TX * 2 ); // bid for bytes

        /* (c) Lecture de BusST  */
        var = inReg16(BusST);
        delay(25);

        /* (d) Test du Bit Rdy4TxNow: On devrait lire BusST=118h  */
        if ( var!=  0x0118)
                {
                return(FAILURE);
                }

         /* (e) Copie de la trame entiere de TxBuffer dans le chip */
        /* (f) Ecriture de XX/2 words en IOBase+0x0 (TX Data Port) */

         for (j = 0; j < SIZE_OF_FRAME_TX; j++)
                {
                out16Intel( PP_RxTx, *(data++) );
                }

        return(SUCCESS);

}


static int OK = 0;

@interrupt void  IRQproc(void)
{
        int *ZoneRx,i,etat;
        if ( i = inReg16(ISQ) == 0x0000)  /* Si evenement non CS sortie */
                return;
        ZoneRx = (int *)0x3000;
        etat =  receive_frame(ZoneRx);
        out16(0x410e, 0x06);
}
```

```
/*****************************/
/* Proc pour prog test      */
/*****************************/

/* Test de Transmission */
void  TestTx(void)
{
        int *ZoneTx;                    /* Zone a transmettre */
        int i, etat;

        ZoneTx = (int *)0x4000;
        for(i =0; i < NB_FRAMES_TO_SEND; i++)
                etat =  transmit_frame(64,ZoneTx);
}


/* Test de Reception */
void TestRxPoll(void)
{
        int *ZoneRx;
        int etat;

        /* Scrutation jusqu a reception : POLLING */
        while ( (etat = inReg16(RxEvent)) != 0x0104)
                ;
        ZoneRx = (int *)0x4200;
        etat =  receive_frame(ZoneRx);
}


/* Test de l interruption en Rx */
void TestRxIRQ(void)
{
        init4IRQ();             /* Validation du CS8900 en IRQ */
        cli();                  /* active l IRQ sur le uC */
        while(1)
                {
                out16(0x4102,0x07);
                delay(25);
                }
}


void main()
{
         reset();
        delay(25);
        initialisation();
        TestTx();
        TestRxIRQ();
}
```

Exemple de capture de trames avec l'analyseur de réseaux Surveyor



Transmission/Réception de trames en utilisant le noyau temps réel µC/OS II et des sémaphores binaires

On utilise les fonctions définies précédemment pour effectuer la transmission et la réception de trames.

Mode d'utilisation du noyau :

- Tâche AppStartTask : création de 2 sémaphores binaire, lancement des tâches AppTask1 et AppTask2 bloqués sur sémaphore. Libération de sémaphores un à un pour activer chaque tâche.
- Tâche AppTask1 :bloqué sur le sémaphore binaire sem1. A sa libération par AppStartTask, transmission d'une trame.
- Tâche AppTask2 :bloqué sur le sémaphore binaire sem2. A sa libération par AppStartTask, réception d'une trame.

Programme

```
#include "INCLUDES.H"
#include "libhc11.h"
#include <io.h>

extern int transmit_frame(void);
extern int receive_frame(void);
extern void delay(void) ;
extern void reset(void) ;

OS_STK  AppStartTaskStk[256];
OS_STK  AppTask1Stk[256];
OS_STK  AppTask2Stk[256];

INT16U  Task1Ctr;
INT16U  Task2Ctr;

OS_EVENT *Sem1;
OS_EVENT *Sem2;

static void  AppStartTask(void *pdata);
static void  AppTask1(void *pdata);
static void  AppTask2(void *pdata);

void main (void)
        {
        OSInit();
        OSTaskCreate(AppStartTask, (void *)0, (void *)&AppStartTaskStk[255], 0);
        OSStart();
        }

static void  AppStartTask (void *pdata)
        {
        char *car,etat;

        pdata = pdata;
        AppTickInit();
        Sem1 = OSSemCreate(5);
        Sem2 = OSSemCreate(4);
        OSTaskCreate(AppTask1, (void *)0, (void *)&AppTask1Stk[255], 15);
        OSTaskCreate(AppTask2, (void *)0, (void *)&AppTask2Stk[255], 10);
        baudrate(9600);
        while (TRUE) {
                etat = read_com(car);
                if (etat) {
                        switch(*car){

                                case '1' : /* Envoi d'une trame */
                                        OSSemPost(Sem1);
                                        break;

                                case '2' : /* Reception d'une trame */
                                        OSSemPost(Sem2);
                                        break;
```

```
                                        default :
                                                break;
                                        }
                                }
                        OSTimeDly(1);
                        }
}

static void AppTask1 (void *pdata)
{
        INT8U err;
        int etat,i;

        pdata=pdata ;
        baudrate(9600);
        delay(25);
        while (TRUE) {
                OSSemPend(Sem1,0,&err);
                Task1Ctr++;
                reset();
                initialisation();
                etat = receive_frame() ;
                OSTimeDly(2);
                 }
}

static void AppTask2 (void *pdata)
{
         INT8U err, etat;

        pdata = pdata;
         baudrate(9600);
         while (TRUE) {
                OSSemPend(Sem2,0,&err);
                Task2Ctr++;
                reset();
                initialisation();
                SIZE_OF_ FRAME_TX = 64;
                etat = transmit_frame();
                OSTimeDly(2);
                }
}
```

III) DETAILS TECHNIQUES CONCERNANT LA MANIPULATION DE LA CARTE

Cas1: Programme en assembleur

Assembler avec asm6811 pour générer un .s19.
Charger dans la RAM avec PCBUG ou BUFFA LO.

Cas2: Programme en C

1er cas : Sans interruption ni modification de la table de vecteurs.

Avoir les fichiers crts.s, un fichier d'édition des liens .LKF, un script .BAT et le programme .c.
Téléchargement identique avec PCBUG ou BUFFALO.
Démarrage avec "go adresse".

2ème cas : Sans interruption mais avec modification de la table des vecteurs.

Avoir les fichiers crts.s, un fichier d'édition des liens .LKF, un script .BAT et le programme .c.
Avoir le fichier table de vecteur VECTOR.c
Téléchargement identique avec PCBUG ou BUFFALO.
Sélection RAMH, mode IO(pas d'entrée d'interruption).
Démarrage avec reset (Hard) en mode étendu.

3ème cas : Avec interruption et modification de la table des vecteurs.

Avoir les fichiers crts.s, un fichier d'édition des liens .LKF, un script .BAT et le programme .c.
Avoir le fichier table de vecteur VECTOR.c
Téléchargement identique avec PCBUG ou BUFFALO.
Sélection RAMH, Mode MEM.
Démarrage avec reset (Hard) en mode étendu.

III) CONCLUSION

Le sujet proposé pour ce projet de fin d'études est des plus intéressant puisqu'il met en application aussi bien le matériel que le logiciel. Par ailleurs, le thème étant d'actualité et le fait de concevoir autour un système autonome nous a permis d'aborder le fameux microcontrôleur 68HC11 et les interfaçages autour de ce processeur. Ce projet nous a permis de réviser nos classiques – Interfaçage, routage, décodage, programmation … - et de découvrir d'autres domaines intéressant – réseaux Ethernet , Temps Réel, soudure de CMS … - , et nous sommes plus que satisfaits de la richesse de ce sujet de projet qui s'adapte merveilleusement bien dans l'optique de notre spécialité qu'est l'informatique industrielle.

D'autant plus que nous sommes parvenus à valider l'interface Ethernet de la carte dans son mode de fonctionnement IO (Transmission/Réception de trames avec ou sans interruption), cependant pour manque de temps, nous n'avons pas pu développer le soft associé à la partie communication WEB, seul un driver bas niveau est disponible. Cette dernière partie logicielle pourra constituer un bon projet pour la nouvelle filière télécommunications.

ANNEXES :

Présentation rapide du microcontrôleur 68HC11A8 :

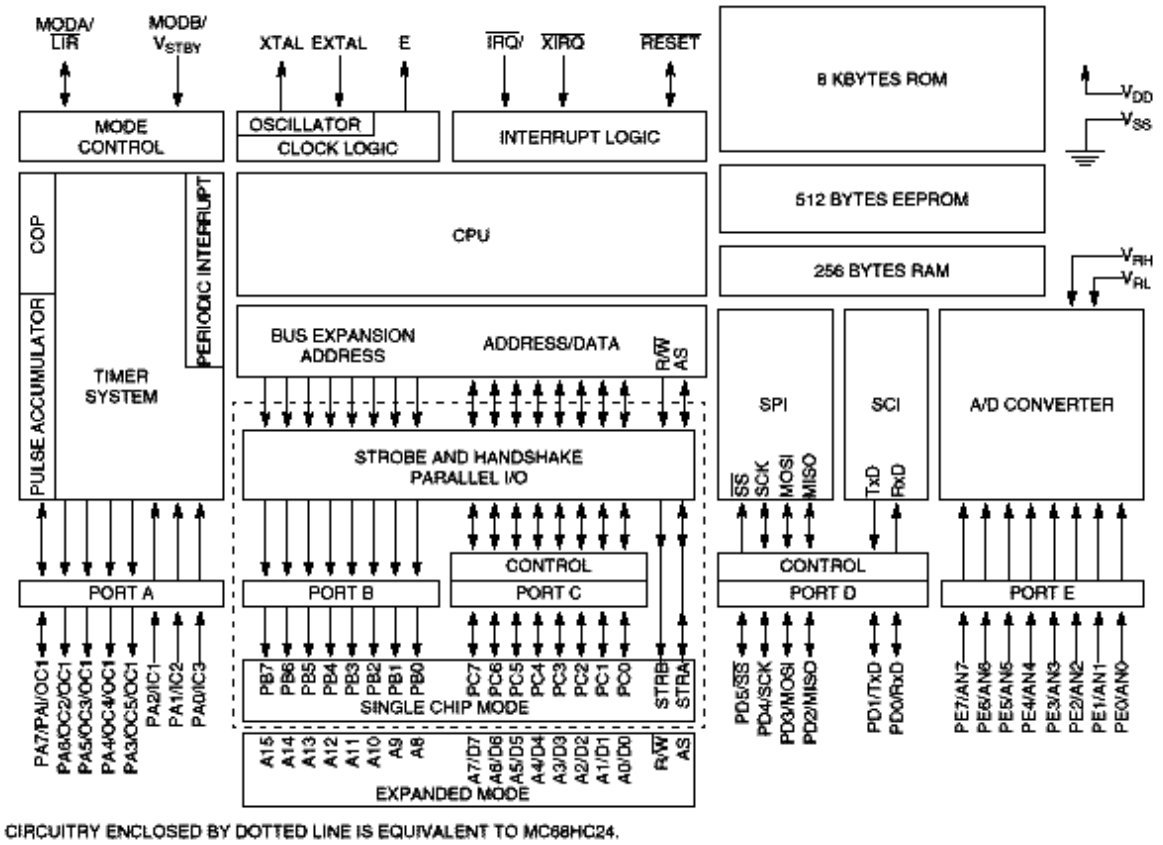


Schéma bloc du microcontrôleur 68HC11

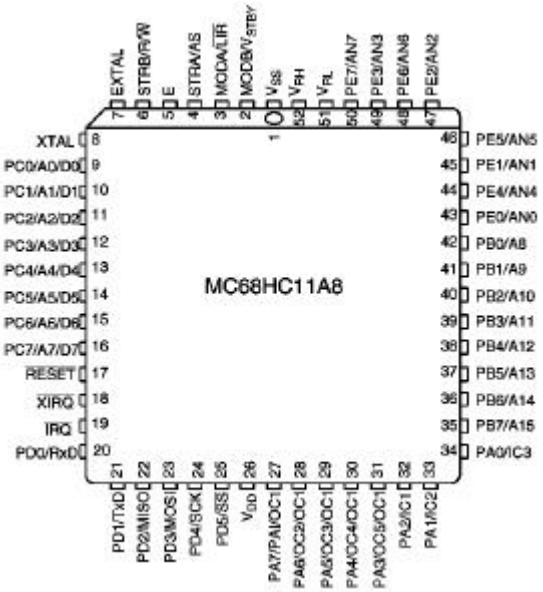
Généralités :

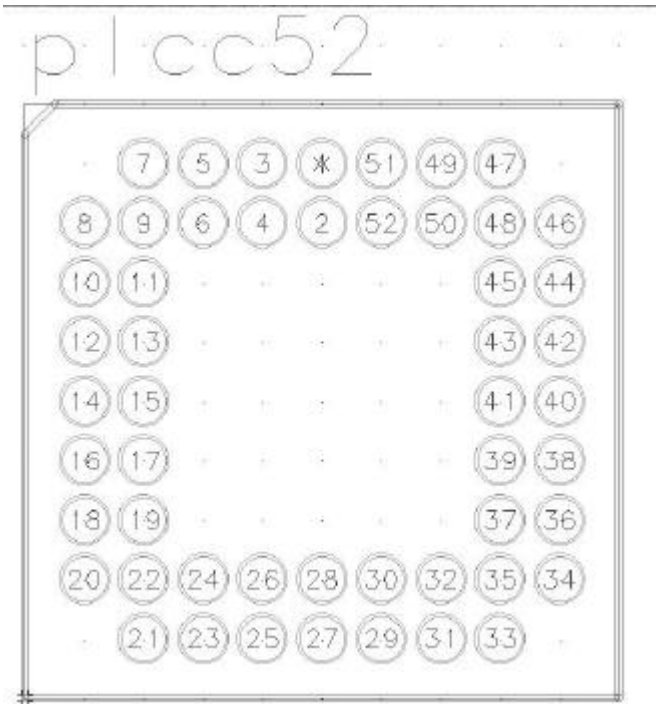
Microcontrôleur 8 bits de MOTOROLA avec les caractéristiques suivantes :

- Technologies HCMOS
- 8 Ko de ROM
- 512 octets de EEPROM
- 256 octets de RAM
- un Timer 16 bits
- 5 ports d'E / S parallèles : ports A, B, C, D et E.
- 2 accumulateurs 8 bits
- Une liaison série asynchrone SCI *(Serial Communication Interface*)
- Une liaison série synchrone SPI *(Serial Peripherical Interface*)
- Un CAN 8 bits, 8 entrées multiplexées
- Un circuit d'interruption temps réel
- Circuit oscillant et de Reset externes

Brochage :

Brochage du 68HC11A8 en boitier PLCC52 :





Mapping :

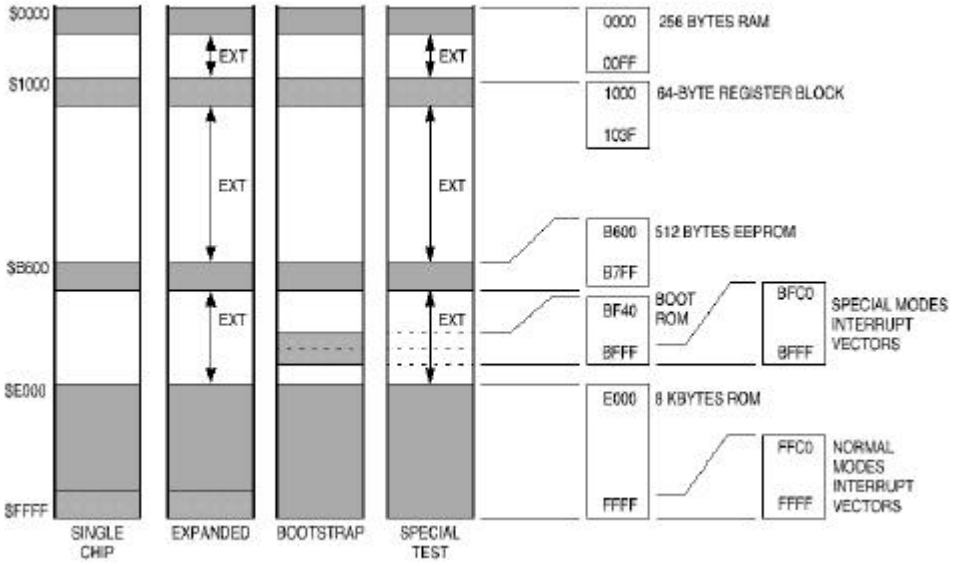
1	GND	14	PC5	27	PA7	40	PB2
2	MODB	15	PC6	28	PA6	41	PB1
3	MODA	16	PC7	29	PA5	42	PB0
4	STRA AS	17	RESET	30	PA4	43	PE0
5	E	18	XIRQ	31	PA3	44	PE4
6	STRB RW*	19	IRQ	32	PA2	45	PE1
7	EXTAL	20	RXD	33	PA1	46	PE5
8		21	TXD	34	PA0	47	PE2
9	PC0	22	PD2	35	PB7	48	PE6
10	PC1	23	PD3	36	PB6	49	PE3
11	PC2	24	PD4	37	PB5	50	PE7
12	PC3	25	PD5	38	PB4	51	VRL
13	PC4	26	VCC	39	PB3	52	VRH

Vdd et Vss : Alimentation du circuit.

MOD A et MOD B : Les niveaux sur ces deux broches permettent d'initialiser le mode de fonctionnement du uC.

MOD B	MOD A	Fonctionnement	Description
1	**0**	SingleChip (Mode 0)	Utilise ses ressources internes
1	**1**	Expanded (Mode 1)	• Utilise les ressources internes et externes • Table des vecteurs en $FFF0 • $FFFE-$FFFF indique le point d'entrée du programme.
0	**0**	Special BootStrap	• Utilise les ressources internes. • Une Boot ROM en $B040-$BFFF où le processeur saute après un RESET contenant une routine chargeant 256 octets depuis la liaison RS232 en RAM à $0000.
0	**1**	Special Test	NON UTILISE, NON RECOMMANDE

Selon les modes de fonctionnement, on a les cartographies mémoire suivantes :



PA0…PA7 : Ce sont des lignes de port lorsque le Timer n'est pas utilisé.
PA0…PA2 sont des entrées.
PA3…PA6 sont des sorties.
PA7 configurable en entrée ou en sortie.

PB0…PB7 : Ce sont des sorties lorsque le uC fonctionne en SingleChip.
En Etendu , ces broches véhiculent le poids fort du bus d'adresses (A8…A15).

PC0…PC7 : En SingleChip, c'est un bus d'E/S bidirectionnel.
En Etendu, ces broches véhiculent le poids faible (A0…A7) du bus d'adresses et les 8 bits de données (D0…D7) du bus de données. (Technique de multiplexage ; le démultiplexage est confié au circuit Latch 74LS573)

PD0…PD5 : Ces broches sont des E/S bidirectionnelles si elles ne sont pas utilisées par le registre de l'interface série.

PE0…PE7 : Ce port comporte 8 entrées dont la particularité est de pouvoir véhiculer des signaux analogiques ou numériques suivant que le CAN soit utilisé ou pas.

IRQ* : Entrée d'interruptions matérielles masquables.
XIRQ* : Entrée d'interruptions matérielles non masquables.

RESET* : Reset sur un niveau bas du microprocesseur.

E, XTAL et EXTAL : Broches concernant l'horloge.
E est une sortie horloge (fréquence du Quartz / 4) permettant la synchronisation des échanges avec des composants extérieurs en mode Etendu.

RxD : Broche de Réception lorsque l'interface SCI est utilisée.

TxD : Broche de Transmission lorsque l'interface SCI est utilisée.

MISO : Lorsque l'interface SPI est utilisée, cette broche est configurée soit en entrée mode maître ou sortie mode esclave.

MOSI: Lorsque l'interface SPI est utilisée, cette broche est configurée soit en sortie mode maître ou entrée mode esclave.

SS* : Lorsque l'interface SPI est utilisée, cette broche permet de sélectionner les esclaves.

SCK : Lorsque l'interface SPI est utilisée, cette broche véhicule le signal d'horloge fournie par le maître. Cette broche devient une entrée sur les esclaves.

VrH et VrL : Entrées de référence de tension (entre Vdd et Vss) utilisées par le CAN.
Vrl est le niveau bas de référence.
Vrh est le niveau haut de référence et doit etre de 3V supérieur à Vrl

STRB : En Mode Etendu c'est le signal R/W sur le bus de données.
En Single Chip c'est une entrée de ligne d'échange permettant la synchronisation des échanges sur le port C ou comme une entrée de détection d'évènements sensible sur front générant une requête d'interruption.

STRA : En Mode Etendu, cette broche véhicule le signal AS pour démultiplexer les adresses basses et les données.
En Single Chip c'est une entrée de ligne d'échange permettant la synchronisation des échanges sur le port C ou comme une entrée de d'etection d'évènements, sensible aux fronts et pouvant générer une requete d'interruption.

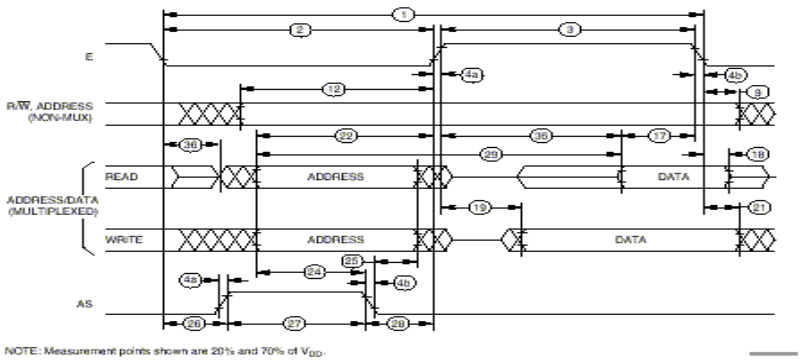


Figure A-14 Multiplexed Expansion Bus Timing Diagram

Table A-7a Expansion Bus Timing (MC68L11A8)

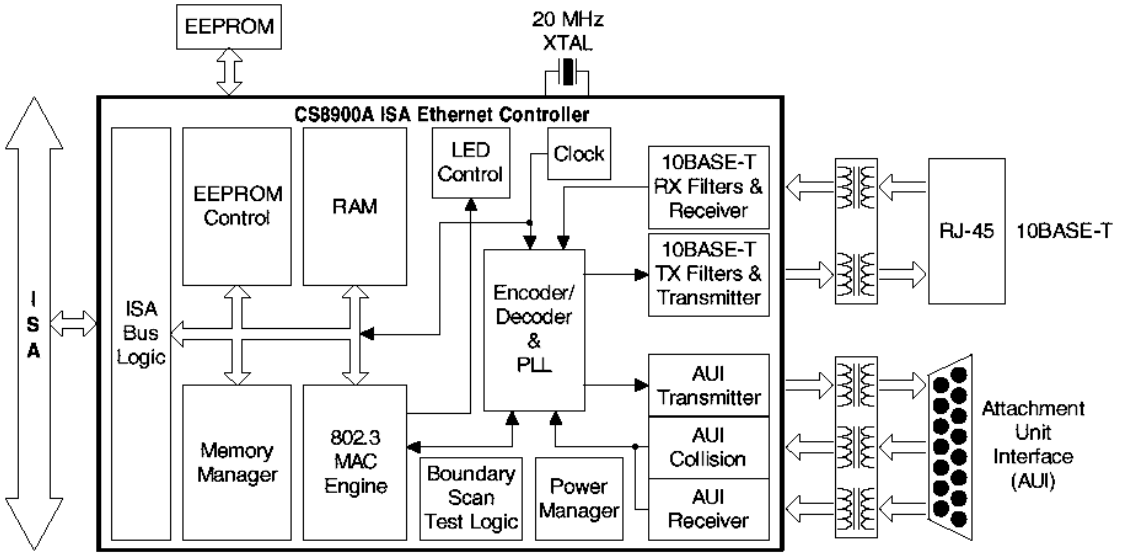
V_{DD} = 3.0 Vdc to 5.5 Vdc, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H

Num	Characteristic	Symbol	1.0 MHz		2.0 MHz		Unit
			Min	Max	Min	Max	
	Frequency of Operation (E-Clock Frequency)	f_o	dc	1.0	dc	2.0	MHz
1	Cycle Time	t_{cyc}	1000	—	500	—	ns
2	Pulse Width, E Low $PW_{EL} = 1/2\ t_{cyc} - 23$ ns (Note 1)	PW_{EL}	475	—	225	—	ns
3	Pulse Width, E High $PW_{EH} = 1/2\ t_{cyc} - 28$ ns (Note 1)	PW_{EH}	470	—	220	—	ns
4a, b	E and AS Rise and Fall Time	t_r t_f	—	25 25	—	25 25	ns
9	Address Hold Time $t_{AH} = 1/8\ t_{cyc} - 29.5$ ns (Note 1, 2a)	t_{AH}	95	—	33	—	ns
12	Non-Muxed Address Valid Time to E Rise $t_{AV} = PW_{EL} - (t_{ASD} + 80$ ns) (Note 1, 2a)	t_{AV}	275	—	88	—	ns
17	Read Data Setup Time	t_{DSR}	30	—	30	—	ns
18	Read Data Hold Time (Max = t_{MAD})	t_{DHR}	0	150	0	88	ns
19	Write Data Delay Time $t_{DDW} = 1/8\ t_{cyc} + 65.5$ ns (Note 1, 2a)	t_{DDW}	—	195	—	133	ns
21	Write Data Hold Time $t_{DHW} = 1/8\ t_{cyc} - 29.5$ ns (Note 1, 2a)	t_{DHW}	95	—	33	—	ns
22	Muxed Address Valid Time to E Rise $t_{AVM} = PW_{EL} - (t_{ASD} + 90$ ns) (Note 1, 2a)	t_{AVM}	265	—	78	—	ns
24	Muxed Address Valid Time to AS Fall $t_{ASL} = PW_{ASH} - 70$ ns (Note 1)	t_{ASL}	150	—	25	—	ns
25	Muxed Address Hold Time $t_{AHL} = 1/8\ t_{cyc} - 29.5$ ns (Note 1, 2b)	t_{AHL}	95	—	33	—	ns
26	Delay Time, E to AS Rise $t_{ASD} = 1/8\ t_{cyc} - 9.5$ ns (Note 1, 2a)	t_{ASD}	120	—	58	—	ns
27	Pulse Width, AS High $PW_{ASH} = 1/4\ t_{cyc} - 29$ ns (Note 1)	PW_{ASH}	220	—	95	—	ns
28	Delay Time, AS to E Rise $t_{ASED} = 1/8\ t_{cyc} - 9.5$ ns (Note 1, 2b)	t_{ASED}	120	—	58	—	ns
29	MPU Address Access Time (Note 2a) $t_{ACCA} = t_{cyc} - (PW_{EL} - t_{AVM}) - t_{DSR} - t_f$	t_{ACCA}	735	—	298	—	ns
35	MPU Access Time $t_{ACCE} = PW_{EH} - t_{DSR}$	t_{ACCE}	—	440	—	190	ns
36	Muxed Address Delay (Previous Cycle MPU Read) $t_{MAD} = t_{ASD} + 30$ ns (Note 1, 2a)	t_{MAD}	150	—	88	—	ns

NOTES:

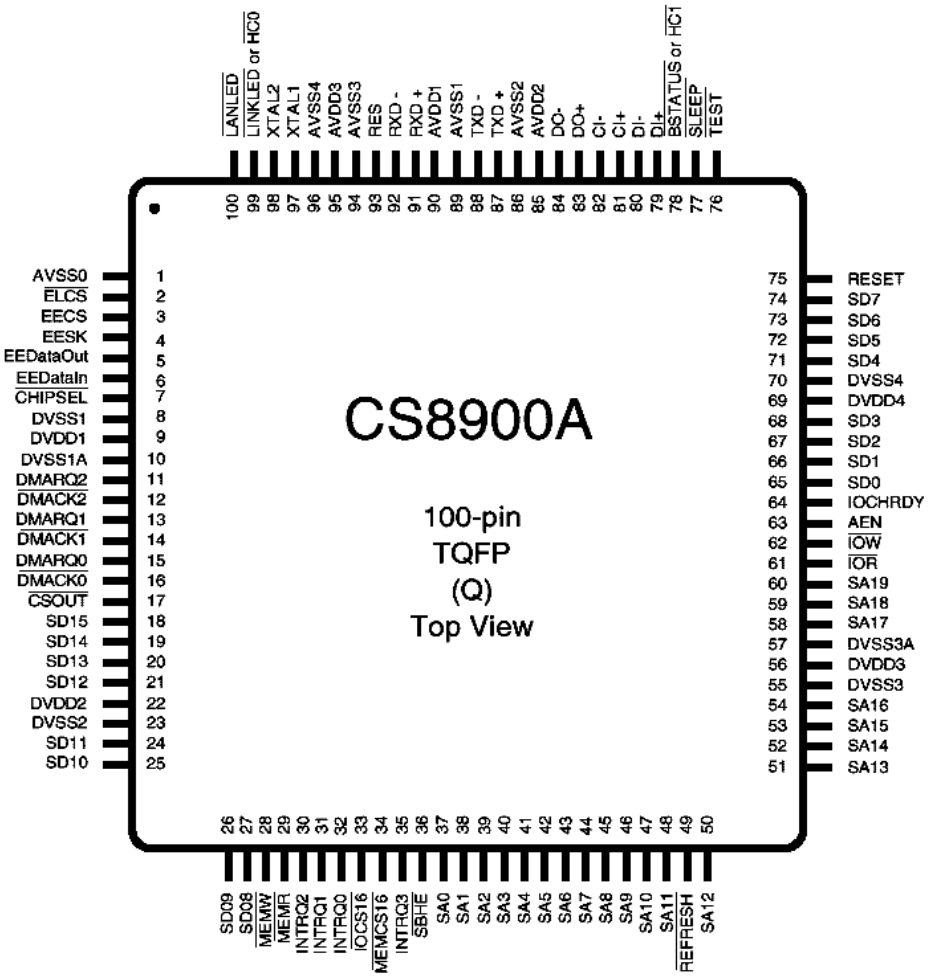
1. Formula only for dc to 2 MHz.
2. Input clocks with duty cycles other than 50% affect bus performance. Timing parameters affected by input clock duty cycle are identified by (a) and (b). To recalculate the approximate bus timing values, substitute the following expressions in place of 1/8 t_{cyc} in the above formulas, where applicable:
   (a) (1-DC) x 1/4 t_{cyc}
   (b) DC x 1/4 t_{cyc}
   Where:
   DC is the decimal value of duty cycle percentage (high time).
3. All timing is shown with respect to 20% V_{DD} and 70% V_{DD}, unless otherwise noted.

Présentation rapide du CS8900A :



Généralités :

- Interface complète pour bus ISA (Industry Standard Architecture).
- 4 lignes d'interruption et 3 canaux DMA .
- Contrôleur Ethernet 16 bits.
- Modes de fonctionnement en espace I-O ou PacketPage.
- 4Ko de RAM intégré dans le Chip.
- Unité MAC (Media Access Control) incorporée gérant les transmissions et réceptions de trames (collision, erreurs).
- Interface EEPROM permettant une configuration personnalisée au démarrage sur EEPROM externe.
- Interface bout de ligne entièrement analogique avec 10Base-T et AUI (Attachment Unit Interface).
- Package TQFP permettant le design de cartes occupant une surface inférieure à 10cm carré.
- Design possible du chip sur carte mère ou carte d'adaptation.



Le point supérieur gauche du composant réel avec l'angle cassé permet d'orienter le chip.

Brochage :

Description rapide des Broches du CS8900A pour la conception Hardware:

- *Interface du Bus ISA :*

SA0…SA19 : Bus d'adresse du système pour décodage des accès aux espaces I/O ou Memory.

SD0…SD15 : Bus bidirectionnel de données format 16 bits.

RESET : Broche d'initialisation asynchrone actif sur état HAUT. – ma intien minimal de 400ns –

AEN : Entrée Adresse Enable.
Si Test* est HAUT, cette entrée indique au CS8900Aque le contrôleur du système DMA prend le contrôle du bus ISA .

MEMR* : Memory Read. Entrée active sur état BAS indique que l'on exécute une opération de lecture.

MEMW* : Memory Write. Entrée active sur état BAS indique que l'on exécute une opération d'écriture.

MEMCS16* : Sortie pour le mode MEM 16bits

REFRESH* : Entrée pour indiquer qu'un cycle de rafraîchissement DRAM est en progression.
Lorsque REFRESH* est BAS, MEMR*, MEMW*, IOR*, IOW*, DMACK1* et DMACK2* sont ignorées.

IOR* : I/O Read. Indique une sortie du contenu du registre I/O 16 bits sur le bus de données du système lorsqu'une adresse valide est détectée.
Lorsque REFRESH* est BAS, IOR* est ignorée.

IOW* : I/O Write. Le CS8900 écrit les données du bus de données vers le registre I/O 16 bits lorsqu'une adresse valide est détectée.
Lorsque REFRESH* est BAS, IOW* est ignorée.

IOCS16* : Le CS8900A génère cette sortie BAS lorsqu'il reconnaît une adresse du bus ISA qui correspond à l'espace I/O assigné sinon sortie en 3^{e} état.

SBHE* : Entrée indiquant un transfert sur l'octet fort du bus de données (SD8…SD15)

INTRQ0…INTRQ2 : Ces sorties indiquent la présence d'un événement d'interruption.

DMARQ0…DMARQ2 : Sortie HAUT indiquant que le CS8900A demande un transfert DMA. Seule une sortie est utilisée à la fois, les autres sont en 3^{e} état.

DMACK0…DMACK2 : Entrée indiquant la reconnaissance par l'hôte de la correspondance de la sortie de demande DMA

CHIPSEL* : Entrée générée par un décodeur logique d'adresse latchable externe lorsque une adresse mémoire valide est présente sur le bus ISA.
Si le Mode Memory n'est pas utilisé, CHIPSEL* doit être tirée BAS.
CHIPSEL* est ignorée en mode I/O et DMA du CS8900A.

- *Interface EEPROM et PROM de Boot :*

EESK : Horloge Série utilisée pour les transferts de données vers l'EEPROM.

EECS : Sortie pour sélectionner l'EEPROM

EEDataIN : Entrée Série pour la réception des données de l'EEPROM connectée à la broche DO de l'EEPROM.
Permet également de sonder la présence de l'EEPROM.

ELCS* : Signal bidirectionnel pour configurer un décodeur logique d'adresse externe latchable.
Si le LA n'est pas utilisé, ELCS* doit être tiré vers BAS.

EEDataOut : Sortie série pour l'envoie de données vers l'EEPROM, connectée à la broche DI de l'EEPROM.
Si TEST* est BAS, elle devient la sortie pour le Boundary Scan Test.

CSOUT* : Sortie pour sélectionner une PROM de Boot externe lorsque le CS8900A décode une adresse mémoire valide de la PROM de Boot.

- *Interface 10 Base-T :*

TXD+, TXD- : Paire différentielle de sortie 10Mb/s pour la transmission de données codées en Manchester vers le 10Base-T.

RXD+, RXD- : Paire différentielle d'entrée 10Mb/s pour la réception de données codées en Manchester vers le 10Base-T.

- *InterfaceAUI (Attachment Unit Interface)*

DO+, DO- : Paire différentielle de sortie 10Mb/s pour la transmission de données codées en Manchester vers l'AUI.

DI+, DI- : Paire différentielle d'entrée 10Mb/s pour la réception de données codées en Manchester vers l'AUI.

CI+, CI- : Paire différentielle d'entrée connectée àla paire de collision de l'AUI.

- *Pins Généraux :*

XTAL1…XTAL2 : Pour connexion d'un quartz de fréquence 20MHz.
Si un signal de 20MHz est utilisé au lieu du quartz, il est relié à XTAL1 et XTAL2 libre.

SLEEP* : Entrée permettant l'activation des modes de mise en veille hardware (Suspend et Hardware Standby).

LINKLED* ou HC0* : Sortie pour la détection d'impulsions de connexion valide.

BSTATUS* ou HC1* : Sortie indiquant une activité sur le bus ISA.

LANLED* : Sortie indiquant l'arrivée, la transmission de paquets pour une collision.

TEST* : Entrée utilisée pour mettre le CS8900A en mode Boundary Scan Test.
Pour une utilisation normale, cette pin est laissée HAUT.

RES : Cette entrée est reliée à une résistance de 4.99K +/- 1% pour

DVDD1…DVDD4 : Fournit le 5V +/- 5% aux circuits numériques du CS8900A.

DVSS1…DVSS4 : Masses numériques.

AVDD1…AVDD3 : Fournit le 5V +/- 5% aux circuits analogiques du CS8900A.

AVSS0…AVSS4 : Masses analogiques.

Notes concernant le fonctionnement du chip CS8900A en mode 8 bits :

- Ports d'entrées / Sorties :

  En Mode 8 bits, les accès sur le CS8900A s'effectuent à partir des 8 ports E/S 16 bits.

Offset	Type	Description
0000h	Lecture / Ecriture	Rx/ Tx de la Donnée (Port 0)
0002h	Lecture / Ecriture	Rx/ Tx de la Donnée (Port 1)
0004h	Ecriture uniquement	TxCMD (Commande de Transfert)
0006h	Ecriture uniquement	TxLength (Taille de Transmission)
0008h	Lecture uniquement	ISQ Interrupt Status Queue
000Ah	Lecture / Ecriture	PacketPage Pointer
000Ch	Lecture / Ecriture	Donnée PacketPage (Port 0)
000Eh	Lecture / Ecriture	Donnée PacketPage (Port 1)

- Transmission d'une Trame :

  1. Requête de l'espace de transmission dans le buffer : Ecriture de la commande sur TxCMD et de la longueur en octets de la donnée à transmettre dans TxLength, puis vérification du registre BusSt (bit 8).
  2. Si l'espace dans le buffer est disponible, écrire la donnée, un octet à la fois, dans le port 0 de Rx/Tx de données.

  Exemple :

  En supposant l'espace I/O du chip soit en 300h.
  Transmission d'une trame de 81 octets (soit 51h) de long.
  1. TxCMD reçoit 00C0h (C0 en 304h et 00 en 305h) et 51h dans TxLength. (51h en 306h et 00h en 307h)
  2. Sondage du bit 8 du registre BusSt pour vérifier si l'espace est disponible. Pour cela il faut utiliser le Packet Page Pointer et le port PacketPage : Ecriture de 0138h en PPPointer puis lecture de PPData depuis 30Ch. Si le bit 8 (Rdy4TxNow) est actif, alors la transmission peut commencer en port 0.
  3. Ecriture du premier octet à transmettre dans 300h, puis le second en 301h, puis le 3^{e} en 300h, puis 301h…jusqu'à ce que toute la trame soit transmise. Le chip se charge de la transmettre une fois le dernier octet écrit.

- Réception d'une Trame :

  (En supposant la base de l'espace d'adressage I/O du chip soit en 300h)
  1. Polling sur le registre RxEvent pour connaître l'arrivée d'une trame.
  2. Lecture de RxStatus à partir du port 0. Lecture de l'octet haut 301h puis du bas 300h.
  3. Lecture de RxLength à partir du port 0. Lecture de l'octet haut 301h puis du bas 300h.
  4. Début de la lecture des données ; d'abord 300h puis 301h, puis 300h, 301h… jusqu'à la fin de la trame.

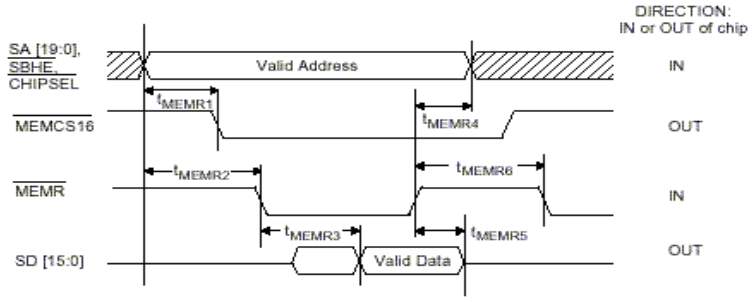
Remarques :

Pas de mode d'interruption en 8 bits, donc un Polling s'impose.
Pas d'EEPROM connectable dans ce mode.

Contraintes Temporelles :

SWITCHING CHARACTERISTICS (Continued)

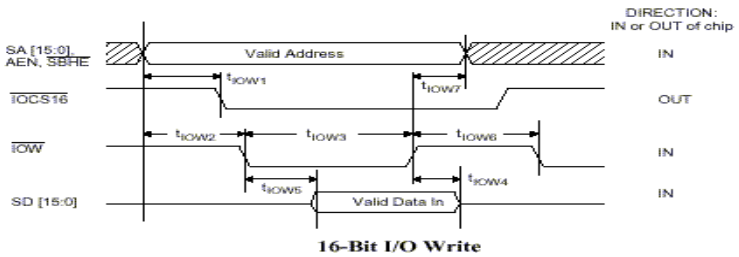
Parameter	Symbol	Min	Typ	Max	Unit
16-Bit Memory Read, IOCHRDY Not Used					
SA [19:0], SBHE, CHIPSEL, active to MEMCS16 low	t_{MEMR1}	-	-	30	ns
Address, SBHE, CHIPSEL active to MEMR active	t_{MEMR2}	10	-	-	ns
MEMR low to SD valid	t_{MEMR3}	-	-	135	ns
Address, SBHE, CHIPSEL hold after MEMR inactive	t_{MEMR4}	0	-	-	ns
MEMR inactive to SD 3-state	t_{MEMR5}	-	30	-	ns
MEMR inactive to active	t_{MEMR6}	35	-	-	ns
16-Bit Memory Read, With IOCHRDY					
MEMR low to IOCHRDY inactive	t_{MEMR7}	-	35	-	ns
IOCHRDY low pulse width	t_{MEMR8}	125	-	175	ns
IOCHRDY active to SD valid	t_{MEMR9}	-	-	0	ns



16-Bit Memory Read, IOCHRDY not used

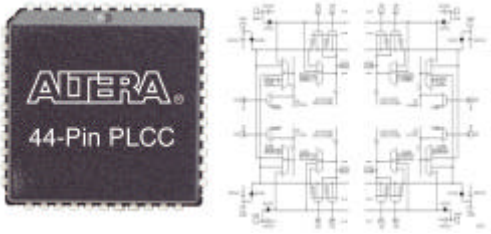
SWITCHING CHARACTERISTICS (Continued)

Parameter	Symbol	Min	Typ	Max	Unit
DMA Read					
DMACKx active to IOR active	t_{DMAR1}	60	-	-	ns
AEN active to IOR active	t_{DMAR2}	10	-	-	ns
IOR active to Data Valid	t_{DMAR3}	-	-	135	ns
IOR inactive to SD 3-state	t_{DMAR4}	-	30	-	ns
IOR n-1 high to DMARQx inactive	t_{DMAR5}	-	-	20	ns
DMACKx, AEN hold after IOR high	t_{DMAR6}	20			ns
16-Bit I/O Write					
Address, AEN, SBHE valid to IOCS16 low	t_{IOW1}	-	-	35	ns
Address, AEN, SBHE valid to IOW low	t_{IOW2}	20	-	-	ns
IOW pulse width	t_{IOW3}	110	-	-	ns
SD hold after IOW high	t_{IOW4}	0	-	-	ns
IOW low to SD valid	t_{IOW5}	-	-	10	ns
IOW inactive to active	t_{IOW6}	35	-	-	ns
Address hold after IOW high	t_{IOW7}	0	-	-	ns



16-Bit I/O Write

Les FPGA

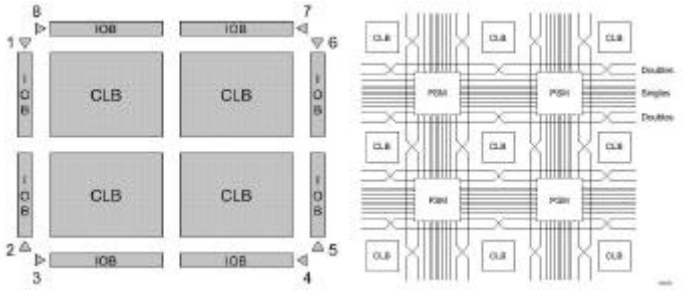


Les FPGA (Field Programmable Gate Array) sont des circuits programmables pouvant réaliser des fonctions logiques combinatoires ou séquencielles. On distingue deux types de FPGA, d'abord les FPGA à SRAM puis les FPGA à anti-fusibles.
Dans notre projet, on utilise un composant de chez ALTERA et qui est de type anti-fusibles, mais cela n'empêche pas de parler des FPGA à SRAM.

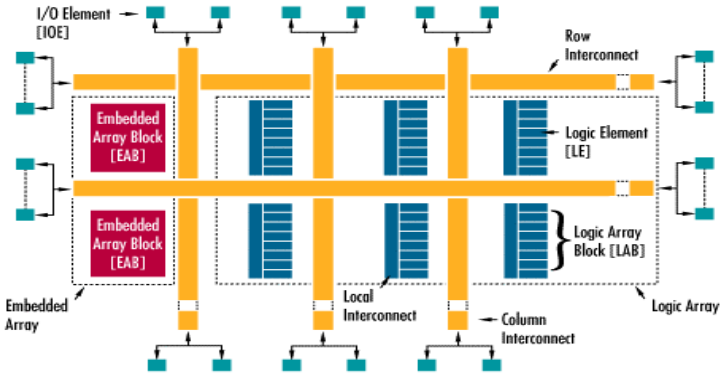
FPGA à SRAM

Ce type de FPGA a été tout d'abord développé par la firme XILINX et commercialisés pour la première fois en 1984.
Ces composants sont reprogrammables indéfiniment, en effet ils présentent des matrices de mémoires RAM qui sont accessibles en lecture et en écriture mais à chaque coupure de courant le circuit perd tout les informations. Pour remédier à ce problème on peut mettre en série une mémoire PROM qui permettra de charger notre programme à chaque utilisation.

Ces FPGA ont une architecture LCA, où on retrouve la puce des blocs logiques ceinturés par des blocs d'entrées/sorties programmables. D'autre part, il existe dans les FPGA



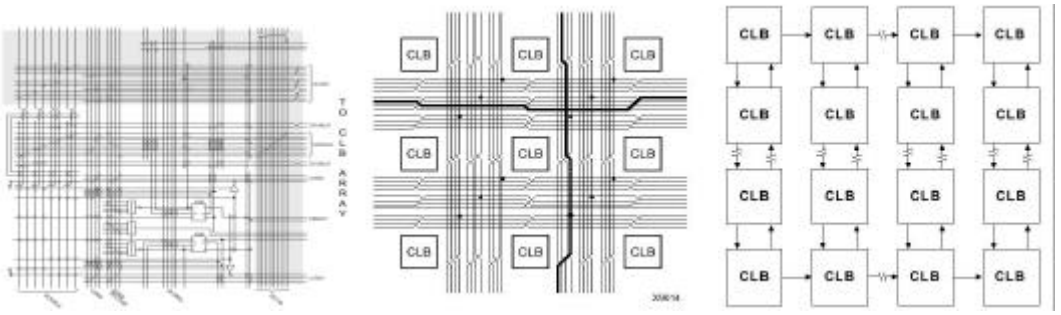
des matrices de lignes d'interconnexions qui courent horizontalement et verticalement entre les divers CLB (Configurable Logic Bloc). Ces lignes d'interconnexions sont effectués par des transistors de types MOS dont l'état est contrôlé par des cellules de mémoire vive ou RAM.



On distingue trois types d'interconnexions :

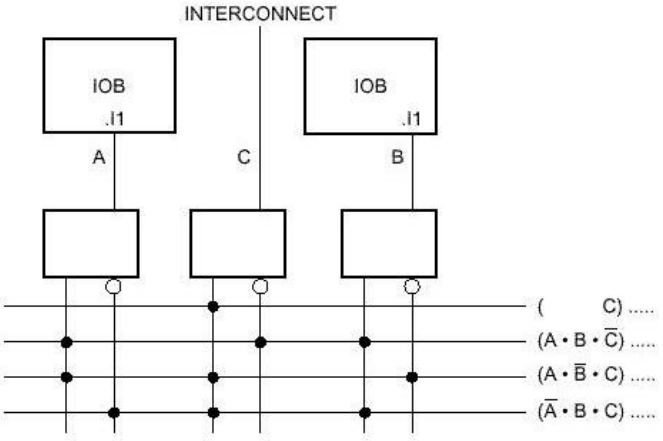
- Les interconnexions à usage général ;
- Les interconnexions directes ;
- Les longues lignes ou long lines.

Les interconnexions à usage général consistent en une grille de cinq segments métalliques verticaux et de cinq segments métalliques horizontaux placés entre les rangées et les colonnes de CLB et d'IOB. Chaque segment fait la hauteur ou la largeur d'un bloc logique. A chaque



intersection de rangée et de colonne se trouve placée une matrice de commutation qui permet de raccorder les segments entre eux selon diverses configurations présentées figure.

Les interconnexions directes dont le principes est visible figure, permettent l'établissement de liaisons entre les CLB et les IOB avec un maximum d'efficacité en terme de vitesse et d'occupation de la puce. Elles résultent du fait que, en raison de la géométrie des



cellules, il est possible de relier directement certaines entrées de l'une aux sorties de l'autre sans faire appel aux autres ressources d'interconnexions. Les liaisons qui peuvent être établies de cette façon sont les suivante :

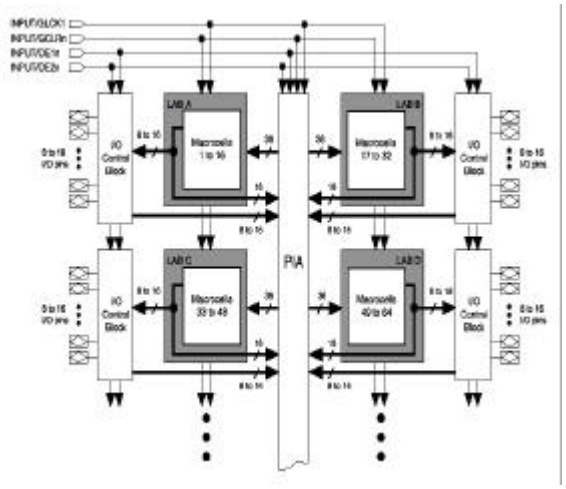
-pour chaque CLB, la sortie X peut être reliée directement à l'entrée B du CLB situé immédiatement sur sa droite et à l'entrée C du CLB situé sur sa gauche. La sortie Y quant à elle peut être reliée directement de la même façon à l'entrée D du CLB du dessus ou à l'entrée A du CLB de dessous ;

-pour chaque CLB adjacent à une IOB, les connexions directes sont possibles tour à tour avec les entrées I ou les sorties O des IOB selon leur positionnement sur la puce.

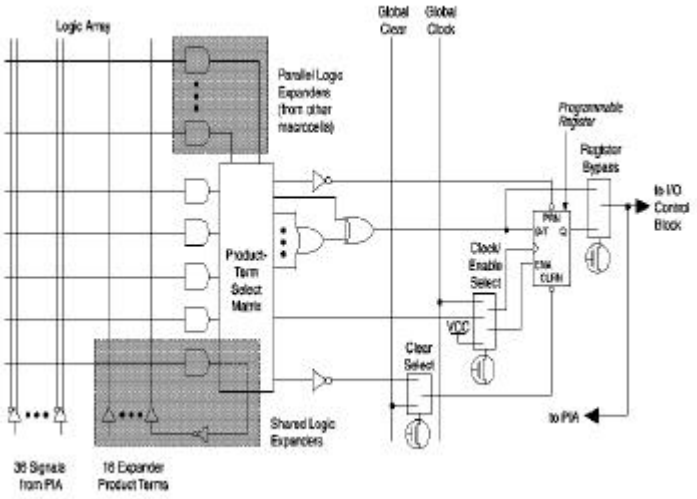
Les longues lignes ou long lines sont des moyens d'interconnexions un peu différents de ce que nous venons de voir en ce sens qu'elles ne passent par les matrices de commutations. Ce sont des lignes métalliques qui traversent la puce de part en part de haut en bas et de droite à gauche. Elles sont utilisées en priorité lorsque des signaux doivent être transmis avec un minimum de retard entre différents éléments afin d'assurer un synchronisme d'action aussi parfait que possible.

FPGA à Anti-Fusibles

Ces circuits sont encore plus récents et visent à optimiser l'utilisation des ressources placés dans une puces par une extension des possibilités d'interconnexions entre les différents blocs logiques. Cependant ce type de FPGA présente le désavantage d'être programmable qu'une seule fois comme les PROM, ceci est en accord avec la technologies anti-fusibles qui physiquement occupe très peu de place sur la puce



Ce type de FPGA a une architecture similaire à celles des LCA, à part qu'entre les cellules du FPGA, il existe un certain nombre de segments métalliques, isolés les uns des autres par les anti-fusibles vierges situés à leurs points d'intersection. En outre, sur une longueur de puce, qu'elle soit horizontale ou verticale, ces segments métalliques sont régulièrement

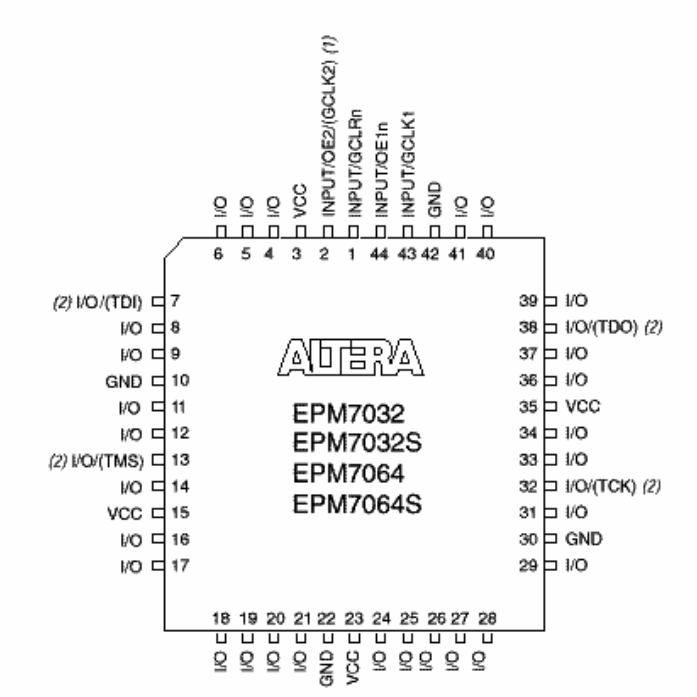


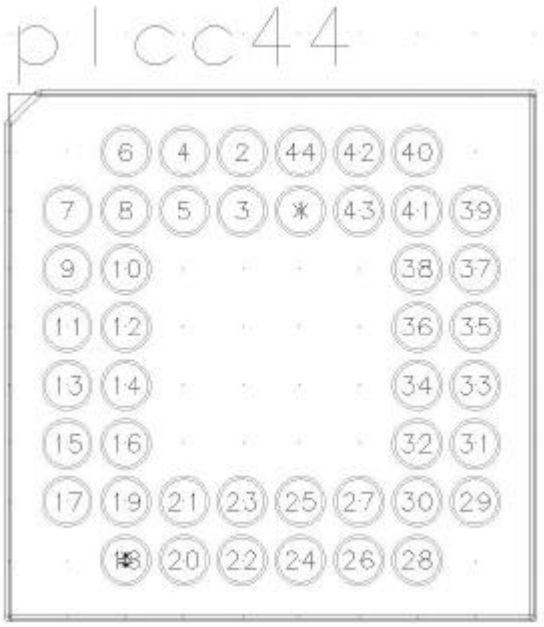
interrompus par des transistors MOS qu'il est possible de rendre conducteurs ou non. De ce fait une grande souplesse d'interconnexions est permise, surtout en raison du nombre très élevé d'anti-fusibles placés sur la puce.

Conclusion

Les FPGA ont connu de nos jours de forts développements, voir qu'ils sont devenus de vrai concurrents des ASIC (Application Specific Integrated Circuit) puisqu'ils fonctionnent dorénavant à des fréquences de l'ordre de la centaine de MegaHertz et peuvent intégrer une un nombre de porte qui avoisine le million de portes. Cependant leur prix présente encore obstacle et promet encore de beaux jours pour les ASIC.

Brochage du MAX7032 :





Interface RS232

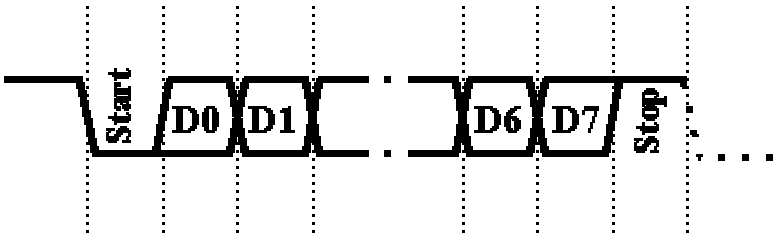
Introduction

La liaison série RS232 est utilisée pour la communication de deux appareils, en général un ordinateur et un périphérique, pour cela elle utilise pour la transmission des signaux en +/-12 Volts :

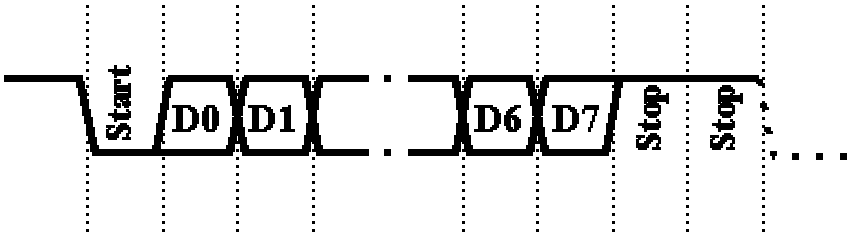
- +12 V : niveau logique 0.
- - 12 V : niveau logique 1.

Les trames

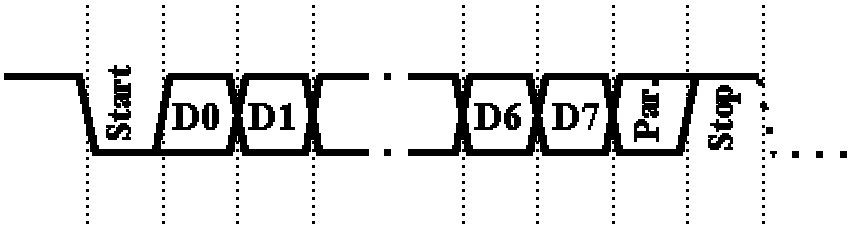
La transmission peut prendre plusieurs format (7 ou 8 bits) avec ou sans contrôle de parité ; une trame commence par 1 bit de start (0 logique) et se termine par 1 ou 2 bits de stop (1 logique).



RS232 8,n,1 : 8 bits de données, pas de parité, 1 bit de stop



RS232 8,n,2 : 8 bits de données, pas de parité, 2 bits de stop



RS232 8,p,1 : 8 bits de données, avec parité, 1 bit de stop

(Parité paire : Par. = 1 si D(7:0) a un nombre pair de "1").
(Parité impaire Par = 0 si D(7:0) a un nombre impair de ″1″).

Le port série (Coté PC)

a) Géométrie

A l'origine, tous les compatibles PC possèdent 2 ports séries: COM1 et COM2. L'un d'entre eux se présente sous la forme d'une prise DB9 mâle et le deuxième, sous la forme d'une DB25 mâle.

DB9 Mâle (vue de devant)

```
------------
\ 1 2 3 4 5 /
 \ 6 7 8 9 /
  ---------
```

DB25 Mâle (vue de devant)

```
-----------------------------
\ 1  2  3  4  5  7  8 ... 13 /
 \ 14 15 16 17 18 .......25 /
  -------------------------
```

Broche DB9	Broche DB25	Nom
1	8	DCD
2	3	RX
3	2	TX
4	20	DTR
5	7	GND
6	6	DSR
7	4	RTS
8	5	CTS
9	22	RI

Description et attribution des signaux

- DCD (Data Carrier Detect): cette ligne est une entrée active haute. Elle signale à l'ordinateur qu'une liaison a été établie avec un correspondant.
- RX (Receive Data): cette ligne est une entrée. C'est ici que transitent les informations du correspondant vers l'ordinateur.
- TX (Transmit Data): cette ligne est une sortie. Les données de l'ordinateur vers le correspondant sont véhiculées par son intermédiaire.
- DTR (Data Terminal Ready): cette ligne est une sortie active haute. Elle permet à l'ordinateur de signaler au correspondant que le port série a été libéré et qu'il peut être utilisé s'il le souhaite.
- GND (GrouND): c'est la masse.
- DSR (Data Set Ready). Cette ligne est une entrée active haute. Elle permet au correspondant de signaler qu'une donnée est prête.

- RTS (Request To Send): cette ligne est une sortie active haute. Elle indique au correspondant que l'ordinateur veut lui transmettre des données.

- CTS (Clear To Send): cette ligne est une entrée active haute. Elle indique à l'ordinateur que le correspondant est prêt à recevoir des données.

- RI (Ring Indicator): cette ligne est une entrée active haute. Elle permet à l'ordinateur de savoir qu'un correspondant veut initier une communication avec lui.

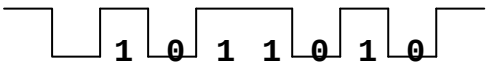
D'un point de vue électronique, les signaux TX et RX en sortie des prises répondent aux normes RS232, c'est à dire:1 logique compris entre -3 et -25V0 logique compris entre +3 et +25V**Programmation**

La communication série nécessite trois fils au minimum: une masse pour référencer les signaux, un fil émetteur et un fil récepteur. Notre liaison série est en effet full-duplex, c'est à dire que l'on peut émettre et recevoir en même temps (comme le téléphone par exemple).

La différence principale entre le port parallèle et le port série est que les informations ne sont pas transmises simultanément sur des fils séparés (D0 à D7) mais les unes après les autres sur un même fil. Cela amène une économie de câble (un fil au lieu de 8) mais un montage décodeur devient nécessaire pour retransformer les données sérialisées.

L'exemple ci-dessous montre comment l'octet 10110101 est transformé pour être transmis sur un seul fil. Vous voyez qu'en plus de l'information utile (10110101) se greffent d'autres bits comme le bit de start. Ces bits sont utiles pour la synchronisation de l'émetteur et du récepteur.

```
LSB     1011010 en série      MSB
```



En effet, la liaison série est totalement asynchrone. Aucune horloge n'est transmise. Il faut donc se mettre d'accord sur la vitesse de transfert des bits et rajouter des bits de synchronisation.

Voici un petit résumé des différents paramètres rentrant en jeu lors d'une communication série:

- **longueur de mot:** sur le PC, le BIOS ne permet une longueur de mot que de 7 ou 8 bits.

- **Parité:** le mot transmis peut être suivi d'un bit de parité qui sert à détecter les erreurs éventuelles de transmission. Il existe deux parités: la parité paire et la parité impaire. Dans le cas de la parité paire, et pour le mot 10110101 contenant 5 états à 1, le bit de parité sera 1 amenant ainsi le nombre total de 1 à un nombre pair (6). Dans le cas de la parité impaire, le bit de parité aurait été 0 car le nombre total de 1 est déjà impair. L'intérêt de ce rajout est le suivant: si jamais lors de la transmission un état 1 est transformé en état 0 (perturbation du canal par des parasites par exemple) le nombre total de 1 change et donc le bit de parité recalculé par le récepteur ne correspond plus à celui reçu. L'erreur est donc détectée. Evidemment, si deux états à 1 passent à 0, l'erreur ne sera pas détectée mais la probabilité pour que cela arrive est très faible.

- **Bit de start:** lorsque rien ne circule sur la ligne, celle-ci est à l'état haut. Pour indiquer qu'un mot va être transmis, la ligne passe à bas avant de commencer le transfert. Cette précaution permet de resynchroniser le récepteur.

- **Bits de stop:** ces bits signalent la fin de la transmission. Selon le protocole utilisé, il peut y avoir 1, 1.5, ou 2 bits de stop (ces bits sont toujours à 1).

Vitesse de transmission: la plupart des cartes série permettent de choisir une vitesse entre 300 et 9600 bauds (par exemple à 300 bauds, un bit est transmis tout les un trois-centième de seconde). Les cartes récentes proposent des vitesses jusqu'à 115200 bauds. Ces vitesses ne vous paraissent peut-être pas énormes mais il faut garder à l'esprit que la liaison série est avant tout pensée pour les liaisons téléphoniques par modems, dont la bande passante est très limitée.

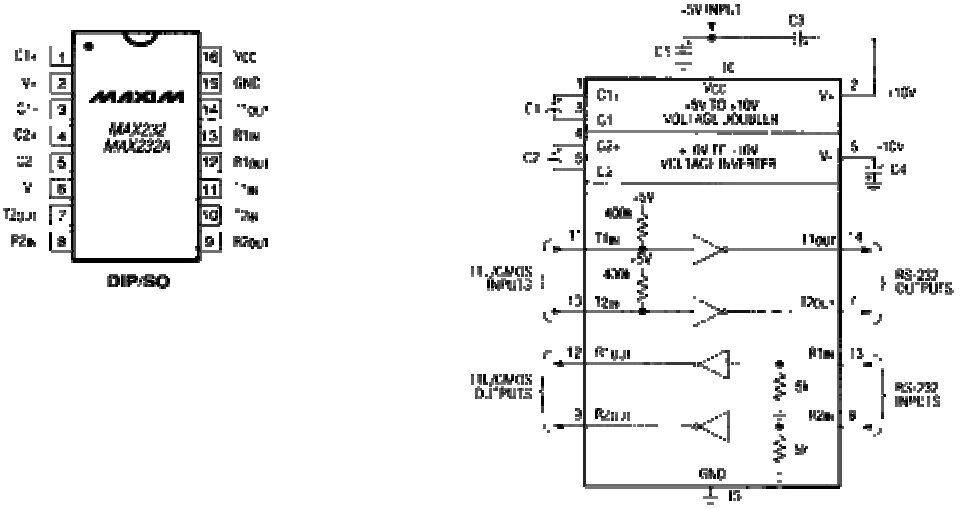
Les protocoles de transmission

On ne peut réussir une transmission qu'à partir du moment où l'émetteur et le récepteur se sont entendu sur la vitesse, le nombre de bit de stop, la longueur du mot et la parité. A ce niveau là, savoir à quel voltage correspond un état haut n'a aucune importance.

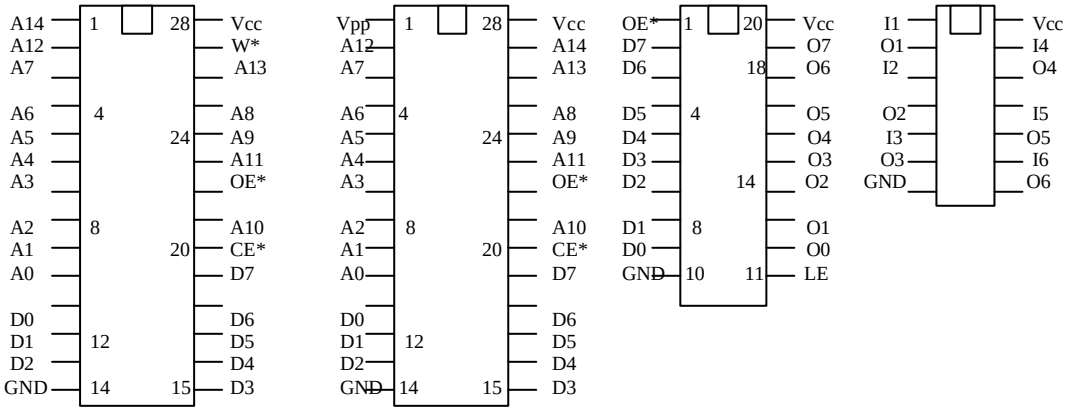
D'une manière générale, la parité est toujours présente car elle permet de détecter la plus grande partie des erreurs de transmission. **Le port série (Coté carte)**

En général les cartes ont une logique qui fonctionne en 0/5Volts. Ainsi avant la transmission il faut faire une translation des niveaux vers (-12V/+12V), pour cela on utilise le composant MAX232.(figure çi-dessous)

+5V-Powered Multi-Channel RS-232 Drivers/Receivers



Brochages des autres circuits intégrés intervenant dans la carte:



RAM
32K x 8bits

ROM
32K x 8 bits

74HC573
Latch 8 bits

Inverseur
7404

µC/OS II

(**M**icro **C**ontroller **O**perating **S**ystem version :2)

Le µC/OS II

L'OS II est un système d'exploitation écrit en langage ANSI C et avec des fonctions en assembleur. L'utilisation de l'assembleur a été optimisé au minimum possible pour permettre la compatibilité avec la grande partie des processeurs. Le µC/OS II peut fonctionner pour la majorité des microprocesseurs, micro-contrôlleur ou DSP de 8-, 16-, 32- ou 64-bits.

Une autre particularité du µC/OS II est qu'il est ROMable, en effet il est possible de le mettre en ROM et ainsi il permet de fonctionner pour toutes les applications embarquées.

Le µC/OS est 'scalable' autrement il est possible d'utiliser que les services nécessaire pour notre application et ceci grâce à la possibilité d'utiliser la compilation conditionnelle.

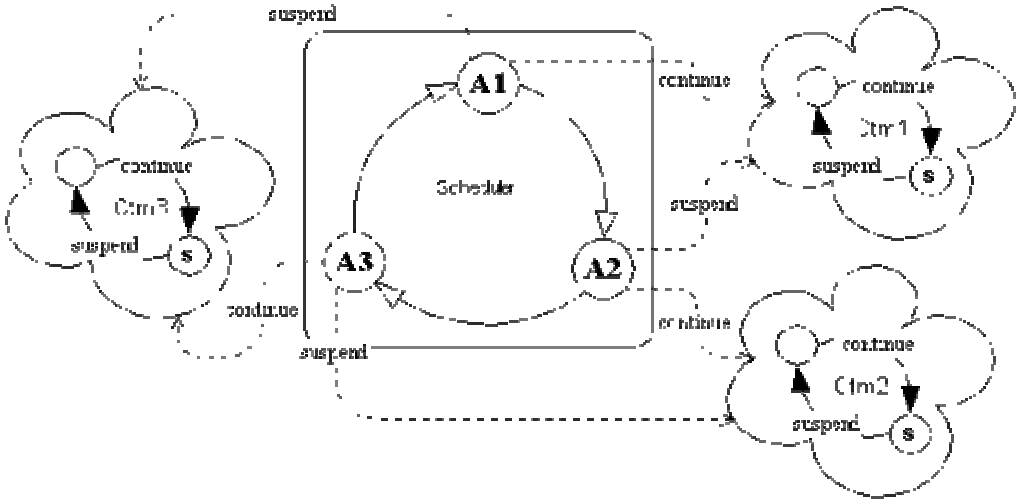
Cet OS est Préemptif et déterministe puisque toutes les fonctions et services du OS ont une durée d'exécution déterminée.

Autre caractéristique du µC/OS II est qu'il est multitâches dans un environnement monoprocesseur. En effet l'utilisateur peut 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, de scrutation ou «polling»…). Une tâche peut être vue comme un programme en cours d'exécution avec son propre contexte (état de la tâche, valeur courante des registres du processeur..). Il est alors possible d'avoir 2 tâches dans ces conditions qui exécutent le même programme.

Fonctionnement du noyau temps réel µC/OS II

- Après l'initialisation des ressources internes du noyau µC/OS II, les différentes tâches de l'utilisateur sont toutes créées pendant cette phase d'initialisation. Le programmeur doit donc fixer le point d'entrée de la tâche, l'emplacement des données pour cette tâche, l'adresse haute de la pile de la tâche (dans le cas du 68HC11, on empile vers les adresses décroissantes et dépilent vers les adresses croissantes) et le degré de priorité de la tâche.
- Lancement de l'ordonnanceur ou scheduler.

Le scheduler ou l'ordonnanceur est le chef d'orchestre d'un système temps réel. Son importance est d'autant plus grande qu'il est invoqué au moins à chaque fois qu'une modification intervient sur l'ensemble des tâches actives.



L'ordonnaceur des processus consiste à effectuer le choix d'un processus parmi N processus éligibles et de lui allouer le processeur. Le processus concerné est dit processus élu. L'ordonnanceur a deux rôles essentiels :

- Assurer la gestion des commutations des tâches de l'état bloqué à l'état éligibles.
- Effectuer le choix d'une tâche dans l'ensemble des tâches éligibles.

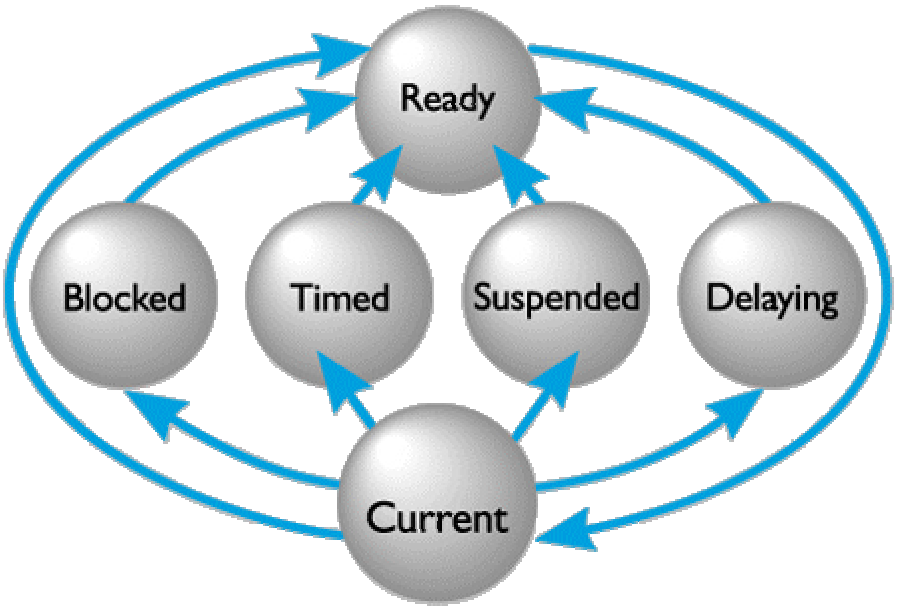
Le principe du fonctionnement de l'ordonnanceur est présenté sur la figure çi-dessus.

A toute ordonnanceur est associé une technique de gestion des différentes tâches du système repérée par l'algorithme de choix. Ce dernier respecte généralement nombre de contraintes :

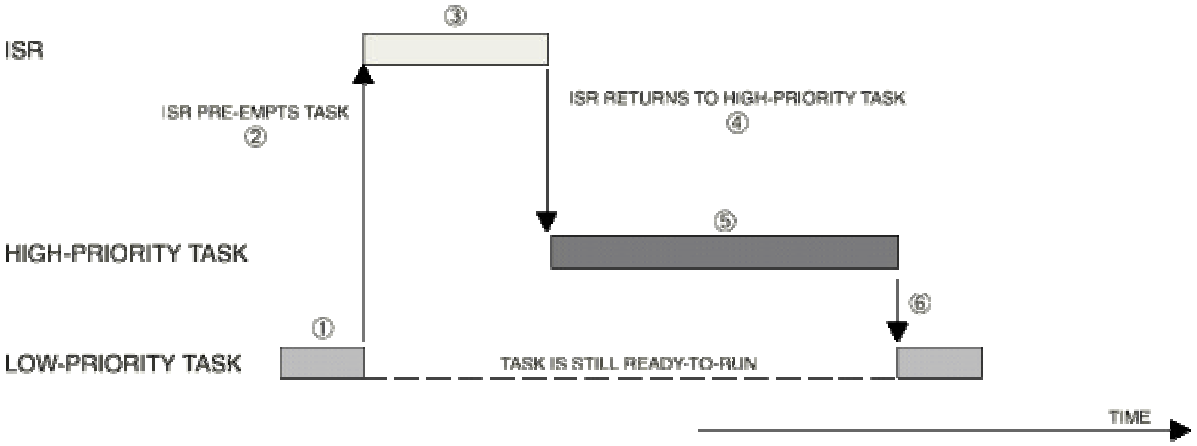
- Garantir à chaque tâche un temps d'allocation donné.
- Respecter un ordre de priorité entre tâches.
- Respecter un temps de réponse donné.
- Annuler de manière préemptive un tâche qui monopolise le processeur.

Caractéristiques du noyau temps réel µC/OS II

- Création et gestion de 62 tâches au maximum.
- Création et gestion de sémaphores binaires et à compter.
- Changement de priorité des tâches.
- Destruction de tâches.
- Suspension et réveil de tâches.
- Envoi de messages depuis une routine d'interruption (ISR) ou d'une tâche vers une autre tâche.



Une tâche est fonction du temps, elle peut être créé, exécuté ou encore détruite. elles ont la possibilité de communiquer entre elles grâces aux sémaphores, les boîtes aux lettres et aux files d'attentes.
Un périphérique peut communiquer avec une tâches grâce aux ISR (Interrupt Service Routine).



Le temps réel

Une des caractéristiques du temps réel est qu'il permet d'assurer un certain déterminisme entre deux utilisations successives d'un processeur par un processus. Il serait inacceptable pour un processus ayant encore besoin de temps machine après l'expiration de son quantum de temps, de disposer à nouveau du processeur après un délai incompatible avec le traitement qu'il a à effectuer.

La prise en compte de ce type de contraintes conduit à associer une priorité au processus. La file d'attente des processus éligibles est alors triée par ordre de priorité croissante ou décroissante et le processus de plus forte priorité est élu. Ainsi un processus pourra s'assurer de la disponibilité du processeur tant qu'aucun processus de priorité supérieur ne le réclamera.

Généralement, les tâches les plus prioritaires sont associées aux traitements de défauts du système (panne, reconfiguration, alarme, etc..). Ensuite sont considérées les tâches associées aux événements matériels tels que les exceptions ou les interruptions. Enfin restent les tâches associées à la gestion des périphériques (tâches de service) et les tâches «utilisateur». Parmi ces dernières une plus forte priorité est généralement associé aux tâches les plus courtes. Par ailleurs, une règles de base selon cette approche est de minimiser les temps de traitements des tâches les plus prioritaires.

A la notion de priorité est attachée celle de rang de processus. Le seul aspect important est généralement la priorité relative des processus et non leur priorité absolue. Exécuter deux processus P1 et P2 de priorités 10 puis les exécuter de nouveau avec des priorités 100 conduit au même fonctionnement (si ces processus sont les seuls existant sur le système). Deux grandes méthodes sont universellement employées en environnement temps réel pour le partage du temps CPU entre processus :

- La première méthode est basée sur la modification dynamique des priorités. Le principe est d'assurer que tout processus, quelle que soit sa priorité, finira à un moment ou à un autre par s'exécuter. Un circuit horloge est dans ce cas souvent nécessaire pour la gestion des incrément de priorités. L'avantage d'une telle approche est de procurer une certaine souplesse dans la gestion des tâches. Elle permet d'affiner cette gestion de telle sorte que l'on puisse par exemple exécuter une tâche trois fois plus souvent qu'une autre. En environnement multi-utilisateur, une telle méthode a encore l'avantage de moduler facilement les différents traitement des utilisateurs. Son inconvénient majeur réside dans le fait que l'exécution d'une tâche est fonction de la charge du système, c'est à dire du nombre et de la priorité des processus évoluant simultanément avec cette même tâche. Ceci nuit au critère déterministe du système. Il convient alors d'utiliser des palliatifs tels que les techniques de préemption (réquisition du processeur) pour répondre à ce critère.
- La deuxième méthode, plus simple dans sa philosophie, met justement l'accent sur l'aspect déterministe en stipulant que c'est toujours la tâche de plus haute priorité qui est exécutée. Ceci impose une gestion plus suivie et plus contraignante des tâches. Par définition, le déterminisme implique l'exécution d'un traitement dans un temps donné. Par conséquent, une tâche de priorité élevée, effectuant un long travail, gardera le processeur aussi longtemps qu'elle désire, bloquant ainsi toutes les autres tâches actives de priorité inférieure.

Réseaux Ethernet

Avant-propos

Au début des années 70, la société Xerox travaillait sur les systèmes bureautiques ouvert et des embryons de réseaux locaux. Elle conçut une version expérimentale d'Ethernet fonctionnant à 3 Mbits/s sur du câble coaxial de 75 Ohms et pouvant couvrir jusqu'à un kilomètre.

Ethernet aujourd'hui domine le marché réseau local, sa diffusion couvre plusieurs dizaines de millions de personnes et il évolue encore pour répondre aux nouvelles exigences de l'informatique moderne.

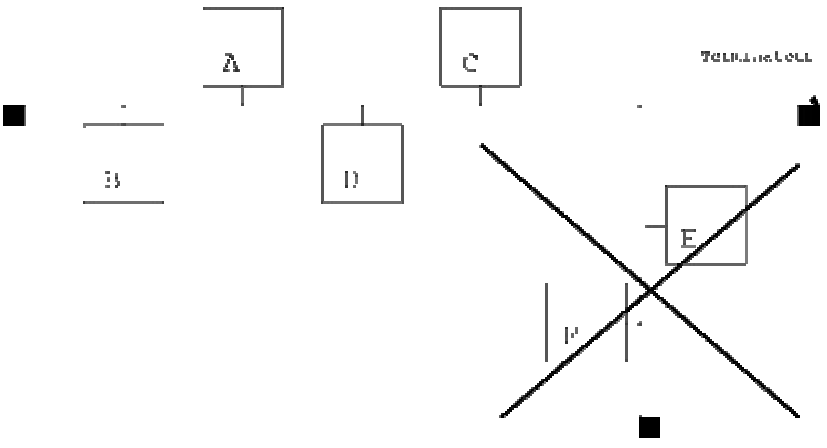
La popularité d'Ethernet est due à l'universalité de ces interfaces et à son faible coût. Ces deux atout sont renforcés par les technologies complémentaires qui permettent à Ethernet de démultiplier ses performances, telles que 100Base-T, 100VG-AnyLAN, IsaEthernet, la commutation dynamique de paquets, le full duplex et les Wireless LAN.

Introduction

Ethernet est un réseau à diffusion en bus avec un contrôle décentralisé et qui fonctionne à 10 ou 100 Mbits/s. Les machines connectées sur Ethernet peuvent transmettre lorsqu'elles le désirent, si deux paquets (ou plus) entre en collision, chaque machine attend un temps aléatoire et re-émet son paquet.

Principes

La structure doit être linéaire. Il n'est donc pas permis d'y ajouter des machines déjà reliées entre-elles et de créer ainsi un embranchement. Le résultat est non prévisible. Il se peut que tout fonctionne correctement pour ces machines mais que des problèmes apparaissent sur le reste du câble.



Lorsqu'une machine émet, le signal parcourt tout le câble puis est absorbé par les terminateurs. La machine à qui il est adressé le lit lorsqu'il passe.

Types de câblage Ethernet

Nom	Type de câble	Longueur Maxi d'un segment	Nbre Max stations/segment	Remarques
10Base5	Coaxial épais	500 m	100	Adapté aux réseaux fédérateurs
10Base2	Coaxial fin	200 m	30	Système le moins cher
10Base-T	Paires torsadées	100 m	1024	Maintenance facile
10Base-F	Fibre optique	2000 m	1024	Le plus adapté entre plusieurs immeuble

Types de codage

Aucun réseau Ethernet n'utilise un codage binaire direct (0=0Volts et 1=5Volts) sur le canal de transmission, en effet cela pourra créer des problèmes d'interprétation de l'information obtenue ;

Exemple : une station qui émet une trame 0001000, les autres stations sur le réseau pourraient interpréter cette séquence binaire comme 1000000 ou 0100000, et ceci pour cause qu'il n'y a pas de différence entre l'état de repos et un bit 0.

Ainsi il y a deux types possible de codage :

- Le codage Manchester ; où chaque intervalle de temps représentatif d'un bit est divisé en deux sous-intervalles égaux. Deux niveaux de tension V1 et V0 représentent les signaux codés. Un bit 1 est représenté par la tension V1 pendant le premier sous-intervalle temporel et par la tension V0 pendant le second sous-intervalle. La représentation du bit 0 est exactement l'inverse.
- Le codage Manchester différentiel est un variante du codage Manchester. Un bit à 1 est représenté par absence de transition(V0→V1 ou V1→V0) au début de l'intervalle de temps d'un bit. de temps d'un bit. En revanche, un bit à 0 est représenté lui, par une transition au début de l'intervalle de temps du bit.

La trame Ethernet

Préambule	Délimiteur de début	Adresse de destination	Adresse source	Longueur du champ Données	Données	Remplissage	CRC
7 octets	1 octet	2ou 6octets	2ou 6 octets	2 octets	0 à 1500 octets	0 à 46 octets	4 octets

Chaque trame contient deux champs d'adresses, Adresse de destination et Adresse source. La norme définit des adresses sur 2 ou 6 octets (16 ou 48 bits) mais pour les systèmes de types bande de base à 10 Mbits/s, elle spécifie uniquement des adresses sur 6 octets. Le bit de poids fort du champ Adresse de destination précise s'il s'agit d'une adresse individuelle, 0, ou d'une adresse de groupe, 1. une adresse de groupe permet à plusieurs stations de reconnaître une même adresse. Lorsqu'une trame est transmise avec une adresse de groupe, toutes les stations appartenant au groupe la reçoivent. Ce type de transmission est appelé transmission multi-destinataire (multicast). Si l'adresse de destination ne comporte que des 1, la transmission est générale, il s'agit d'une diffusion (broadcast). Une trame transmise en diffusion est délivrée à toute les stations du réseau. Une autre caractéristique intéressante de l'adressage est le bit 46 du champ Adresse de destination. Il permet de distinguer des adresses locales et des adresses globales. Les adresses locales sont attribuées par l'administrateur du réseau et n'ont de signification que pour le réseau local concerné. En revanche, les adresses globales sont attribuées par l'IEEE et permettent à toutes les stations de disposer d'une adresse absolument unique au niveau mondial.

Le champ longueur définit le nombre d'octets dans le champ Données (entre 0 et 1500 octets).

Le dernier champ est le champ Total de contrôle ; il contient quatre octets et permet d'effectuer un contrôle d'erreur de transmission.

Bibliographie / Sources Internet :

- Datasheets Crystal concernant le CS8900A : Product DataSheet, AN181, AN83 (Technical Reference Manual).
- Datasheet du 68HC11 de MOTOROLA
- Datasheet du Max7000 d'ALTERA.
- µC/OS II de P.LABROSSE.

Sites Internet :

http://www-elec.enserb.fr/~kadionik
http://www.cirrus.com
http://www.motorola.com
http://www.altera.com
http://www.xilinx.com
ftp://ftp.cirrus.com

Outils de travail pour mener le projet :

- Mentor Graphics (Design de la carte, PCB).
- MaxPlus II (programmation et simulation de la logique câblée pour ALTERA).
- COSMIC (Compilateur C).
- Programmateur de FPGA DATA I/O
- Carte 68HC11 + OS_2 (noyau temps réel) + PCBUG11, Buffalo (Moniteurs de communication avec le 68HC11).
- Analyseur de signaux numériques HP.
- Analyseur de réseaux SURVEYOR.
- Word (Pour taper ce rapport).
- Autres : Netscape, Acroread, fer à souder et outils.

Alexandre PHAO
Ingénieur Info-Indus (ENSERB 2000)
60 Square des Bauves
95140 Garges-lès-Gonesse
tél :01 39 93 39 85

Hamid CHOUKRI
Ingénieur Info-Indus (ENSERB 2000)
Lot Messaoudia
Rue2 n 4 CIL HaySalam
20200 Casablanca
Maroc
Tél : 06.16.90.06.07

Patrice KADIONIK
Professeur ENSERB
+----------+
+"Tout doit être aussi simple que possible, pas seulement plus simple"+
+----------+
+ Patrice KADIONIK http://www-elec.enserb.fr/~kadionik +
+ ENSERB - IXL fax : +33 5.56.37.20.23 +
+ Laboratoire de Microélectroniques tél : +33 5.56.84.23.47 +
+ 351, Cours de la Libération http://www.enserb.fr +
+ 33405 TALENCE Cedex mailto:kadionik@enserb.fr +
+ FRANCE +
+----------+

Ecole **N**ationale **S**upérieure d' **E**lectronique et de **R**adioélectricité de **B**ordeaux