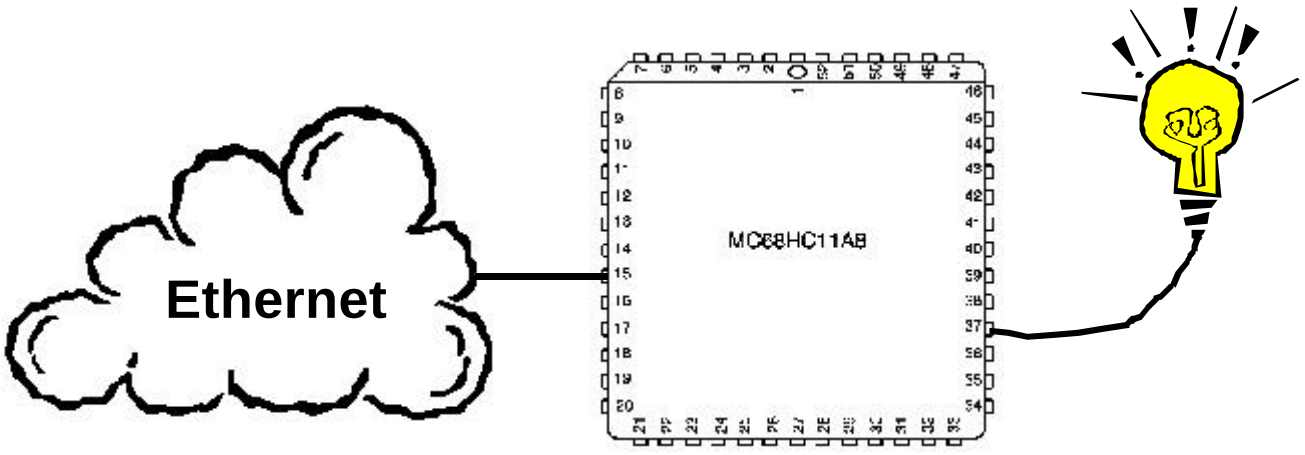
Rapport de Projet de 3ème année
présenté à
Ecole **N**ationale **S**upérieure
d'**E**lectronique **I**nformatique et **R**adiocommunication
de **B**ordeaux

Filière Electronique

Implantation de protocoles réseaux IP (UDP, TCP, ICMP) sur microcontrôleur 68HC11 muni d'un noyau temps réel

Auteurs : **Xilban KRECKELBERGH**
Eric GUIONNEAU



Responsable de Projet : **Monsieur Patrice Kadionik**

Année Universitaire : 2000 - 2001

Sommaire

Introduction

Le projet que nous avons réalisé, nous a fait côtoyer et mettre en œuvre différents domaines. Le premier domaine concerne les réseaux, et plus précisément ceux permettants la communication sur Internet (IP, etc.). Le second nous a permis de mettre en œuvre la programmation avec un noyau temps réel, et, ainsi, d'aborder les concepts et les difficultés de la programmation en temps réel que nous avions préalablement aperçus en cours.

Notre projet s'inscrit dans le sens de l'évolution industrielle, par le fait que la tendance actuelle est d'intégrer l'échange de donnée via Internet sur des systèmes embarqués, et de leur fournir une autonomie à ce niveau. En effet, ces applications connaissent un développement très significatif, notamment dans le domaine automobile pour lequel les constructeurs développent activement des autoradios intégrant la majorité des fonctionnalités multimédias, et entre autres, Internet, mail, etc.

A travers ce document, nous allons, tout d'abord, vous présenter la configuration matérielle et logicielle avec laquelle nous avons réalisé ce projet.

Ensuite, nous développerons les différents niveaux de protocoles que nous avons mis en œuvre lors de ce projet.

Enfin, nous décrirons l'organisation de notre programme et expliciterons les différents choix que nous avons effectué pour mener à bien ce projet.

1. Le sujet de projet

1.1 Présentation

Notre projet consiste à l'implantation des protocoles Internet et, à un niveau supérieur, des protocoles UDP, TCP et ICMP. Ces différents protocoles sont à intégrer sur un microcontrôleur 68HC11 de la famille Motorola, munis du noyau temps réel µC/OS2, qui est embarqué sur une carte reliée au réseau Ethernet.

1.2 Cahier des charges

La mise en place des différents protocoles sera réalisée en ne considérant qu'une seule connexion entre l'utilisateur distant et l'application fonctionnant sur la carte 68HC11.

La programmation doit être réalisée avec un noyau temps réel.

Les protocoles TCP, UDP, ICMP et ARP doivent être implantés.

Etant donné la configuration Hardware utilisée, la maquette doit recevoir et émettre des trames de type Ethernet compatible avec la norme 802.3 10-base T.

La pile IP a créer doit avoir un rôle de récepteur, elle ne transmet pas de données. Les trames transmises sont des acquittements ou des trames concernant la gestion de la liaison entre les deux équipements distants

Les données contenues dans les trames reçues par la carte 68HC11 pourront être utilisées par une application programmée sur le microcontrôleur.

1.3 Objectifs du projet

Le but de ce projet est de pouvoir commander une application via un réseau Ethernet. Les commandes sont réalisées par l'émission de trames IP de la part d'une station distante. L'application est connectée à la maquette qui réalise l'analyse des trames. La figure 1 donne une vision globale du projet.

A moyen terme, l'objectif est de réaliser cette commande à distance grâce au réseau Internet et ainsi permettre, à d'autres universités, de faire fonctionner certaines applications quelles ne possèdent pas. Ce concept a déjà été exploité par le projet RETWINE qui permet de faire de l'instrumentation à distance, grâce à un partenariat entre différentes universités européennes.

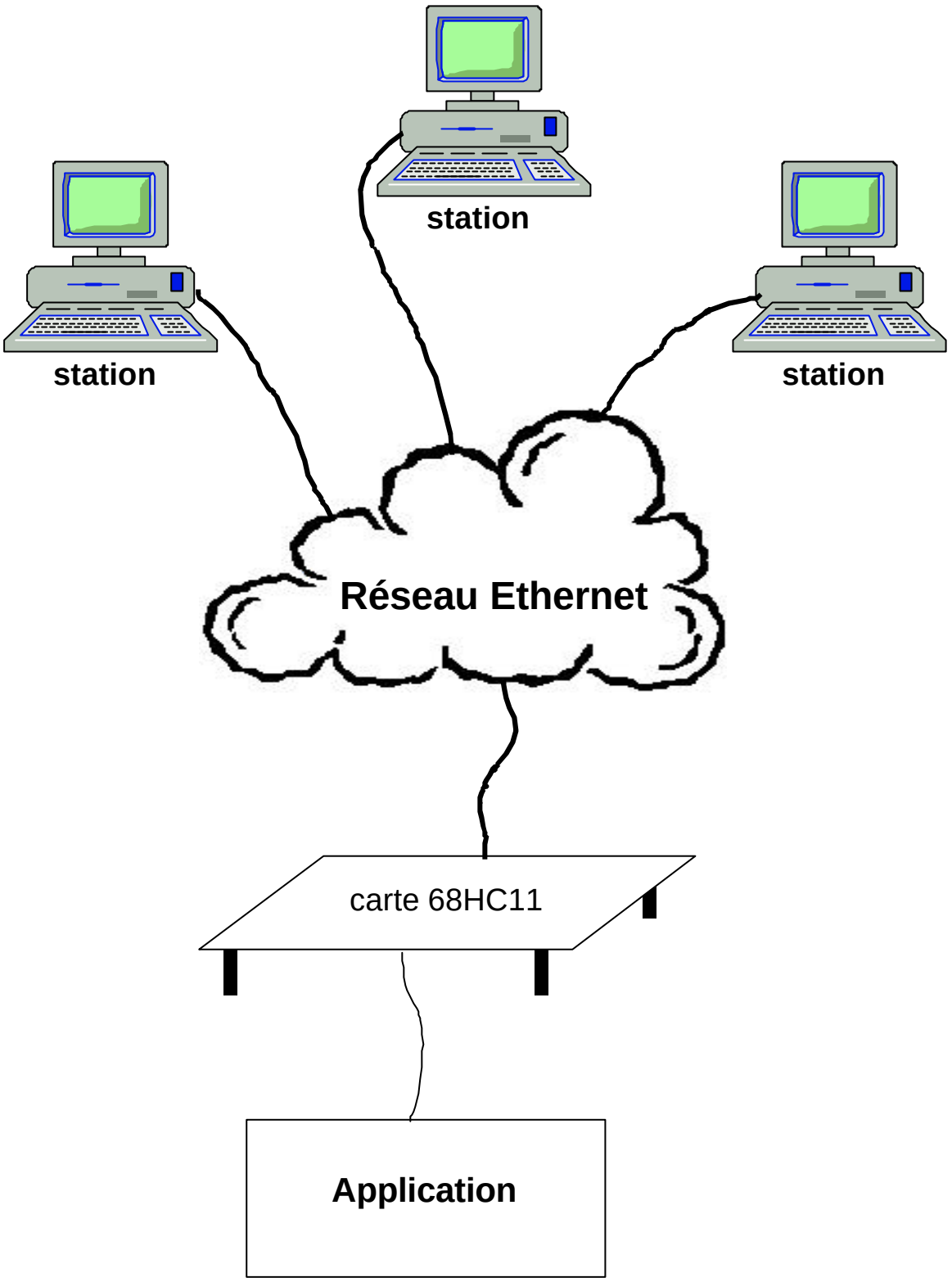


Figure 1 : schéma du projet

2. Configuration

2.1 Configuration matérielle

Toute la partie matérielle a déjà été réalisée par une autre équipe lors d'un projet de fin d'étude de l'année 1999-2000. Nous ne ferons pas une étude détaillée du matériel, mais nous nous focaliserons sur les éléments ayants un rapport direct avec la mise en œuvre de notre projet.

2.1.1 Description de la carte

La carte avec laquelle nous travaillons est composée des éléments suivants :

- Le microcontrôleur 68HC11A8 : ce microcontrôleur possède un bus d'adresse de 16 bits (soit une capacité d'adressage de 64 Koctets) afin d'adresser des mémoires en mode étendu. Le bus de données est de 8 bits.
- Une RAM externe de 32 Koctets correspondant aux adresses basses.
- Les adresses supérieures sont, soit dans une seconde RAM externe de 32 Koctets, soit dans une ROM, en fonction de la position d'un interrupteur.
- Une sélection du mode de fonctionnement du 68HC11 via deux interrupteurs.
- L'interface Ethernet CS8900, qui permettra l'émission et la réception de trames Ethernet. Cette interface se comporte comme un périphérique 8 bits.
- L'interface RS232, MAX232, réalisant l'adaptation en tension des signaux circulant sur la liaison série RS232, qui permet le chargement du programme dans le microcontrôleur 68HC11 et la communication avec un ordinateur.

2.1.2 Cartographie mémoire

La figure suivante représente l'adressage mémoire externe sur la carte.

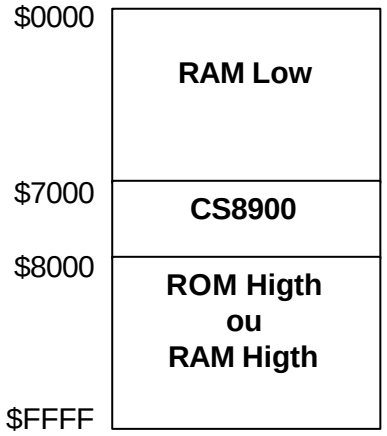


Figure 2 : Cartographie mémoire

2.1.3 Décodage d'adresse

L'accès à chacune des ressources externes, est réalisée par un décodage d'adresse définit par la position de l'interrupteur MAP, et des quatre bits de plus fort poids du bus d'adresse (A15, A14, A13, A12). La table de vérité caractérisant ce décodage est représenté par la figure 3.

MAP	A15	A14, A13, A12	RAML	CS8900	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

Figure 3 : Table de décodage

2.1.4 Le microcontrôleur 68HC11

Comme nous l'avons indiqué précédemment, nous travaillons avec un 68HC11 de chez Motorola. Ce microcontrôleur possède un bus de données de 8 bits et un bus d'adresse de 16 bits.

Il est issu de la technologie HCMOS. Il contient 8 Ko de ROM interne, 512 octets de EEPROM, 256 octets de RAM et un timer de 16 bits.

Le 68HC11 possède cinq ports d'entrée/sorties et deux accumulateurs de 8 bits. La communication avec certains périphériques peut être assurée par une liaison série asynchrone SCI (Serial Communication Interface) et par une liaison série synchrone SPI (Serial Peripherical Interface).

Il est également muni d'un convertisseur Analogique / Numérique de 8 bits, ayant huit entrées multiplexées.

Pour notre application, nous utiliserons le 68HC11 en mode Etendu (Expanded), les autres modes possibles étant Single chip, Bootstrap et Special test. En effet, nous utiliserons les RAM L et RAM H, pour implanter le code nécessaire à la mise en place des protocoles.

2.1.5 L'interface Ethernet CS8900

Le CS8900 est une interface Ethernet permettant la réception et l'émission de trames Ethernet sur le réseau. Ce composant est connecté au réseau via un connecteur RJ45. Le bus de données du CS8900 est de 16 bits. Or le bus de données du microcontrôleur n'étant que de 8 bits, il nous faudra travailler sur 8 bits, et ainsi, configurer les différents registres nécessaires à ce mode de fonctionnement.

Ce composant est particulièrement intéressant. En effet, il effectue une première analyse des trames Ethernet reçues en vérifiant la valeur du CRC (les éléments de la trame seront expliqués ultérieurement), ainsi que la taille de ces mêmes trames. Il peut également différents signaux vers le microcontrôleur, donnant des informations sur l'état de la transmission envoyée et sur le type de trame reçue.

2.1.6 Le FPGA ALTERA

Ce composant participe au décodage d'adresse. Il a été programmé pour sélectionner le composant CS8900, la RAML ou ROMH/RAMH (la sélection entre RAMH et ROM H se faisant grâce à un interrupteur). Il permet au microcontrôleur de communiquer avec le composant adéquat.

Schéma de principe de la carte

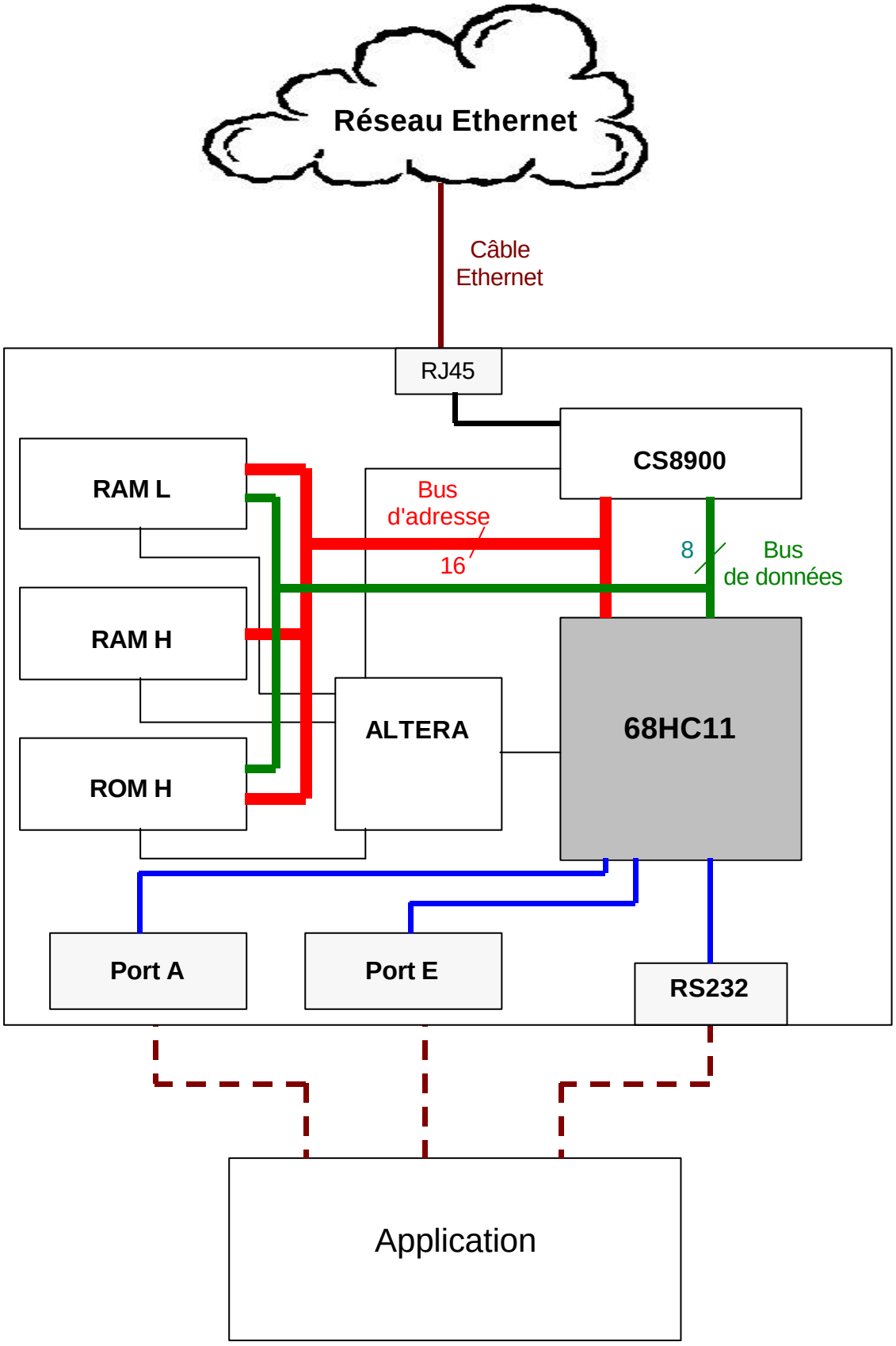


Figure 4 : Schéma synoptique

2.2 Configuration logicielle

2.2.1 Cosmic Software version 3.60

Cosmic est le compilateur que nous utilisons pour le développement de notre projet. L'écriture du programme est réalisée en langage C. Ce compilateur est un compilateur croisé.

L'avantage de ce type de compilateur est qu'il permet un développement des programmes dans un langage évolué et non nécessairement en langage assembleur.

2.2.2 WINBUG11

Winbug est un moniteur permettant de visualiser le contenu de la mémoire sur la carte et d'en modifier le contenu. Cet outil, nous permet de visualiser les données reçues, et les données correspondant aux variables globales.

2.2.3 HCLOAD

Cet outil permet le chargement du programme (.S19) à partir du PC, sur la carte, via une liaison RS232. Ce logiciel est d'une utilisation très simple, en effet, il suffit d'appuyer sur le bouton RESET de la carte, et de sélectionner le fichier .S19 désirer, ainsi que l'adresse de départ du code et d'appuyer sur OK.

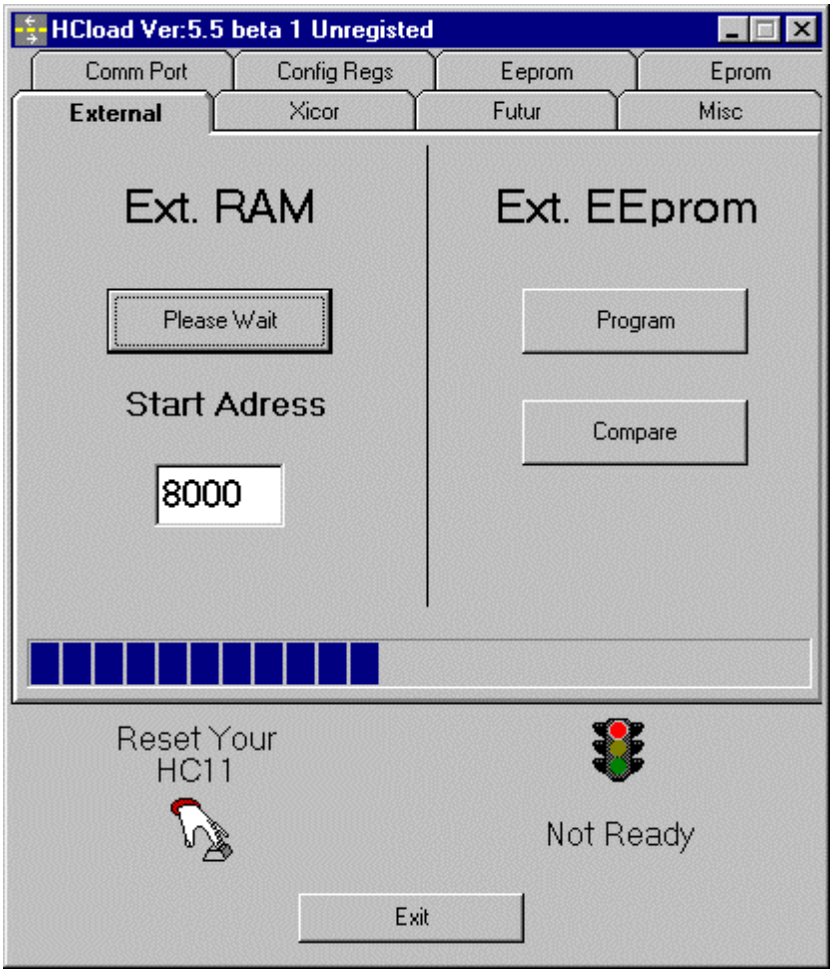


Figure 5 : Interface de HCLOAD

2.2.4 Hyperterminal 68HC11

Ce logiciel permet de dialoguer avec la carte 68HC11 via la liaison série RS232 lorsque le programme est en cours d'exécution, en effet cela est très utile pour afficher des informations à l'écran qui ont deux buts :

- Le debuggage du programme en marquant les différentes fonctions qui sont traversées.
- Une fonction d'application, qui intervient au dessus des protocoles comme le montre la figure 20.

L'Hyperterminal 68HC11 offre, également, la possibilité d'envoyer des données vers la carte microprocesseur. Nous n'utiliserons pas cette possibilité car notre projet a un rôle de récepteur.

2.3 Développement avec l'émulateur

Un émulateur est un appareil permettant de simuler le fonctionnement d'un microcontrôleur. A la place de ce dernier, nous posons une sonde (voir figure 6) et grâce à un logiciel spécifique, **Pathfinder11**, nous faisons fonctionner le programme sur l'ordinateur et pouvons, ainsi, visualiser tous les paramètres du programme.

Ceci présente l'avantage de ne pas avoir besoin de télécharger le programme dans le microcontrôleur à chaque modification (cette opération prend plusieurs minutes) et offre une très grande visibilité du programme, tout en faisant fonctionner la carte dans les conditions réelles d'utilisation, c'est-à-dire que les stimulis d'entrées/sorties sont réels.

PathFinder11 présente la possibilité de positionner des points d'arrêt dans le programme, mais également de visualiser le routage d'exécution du programme.

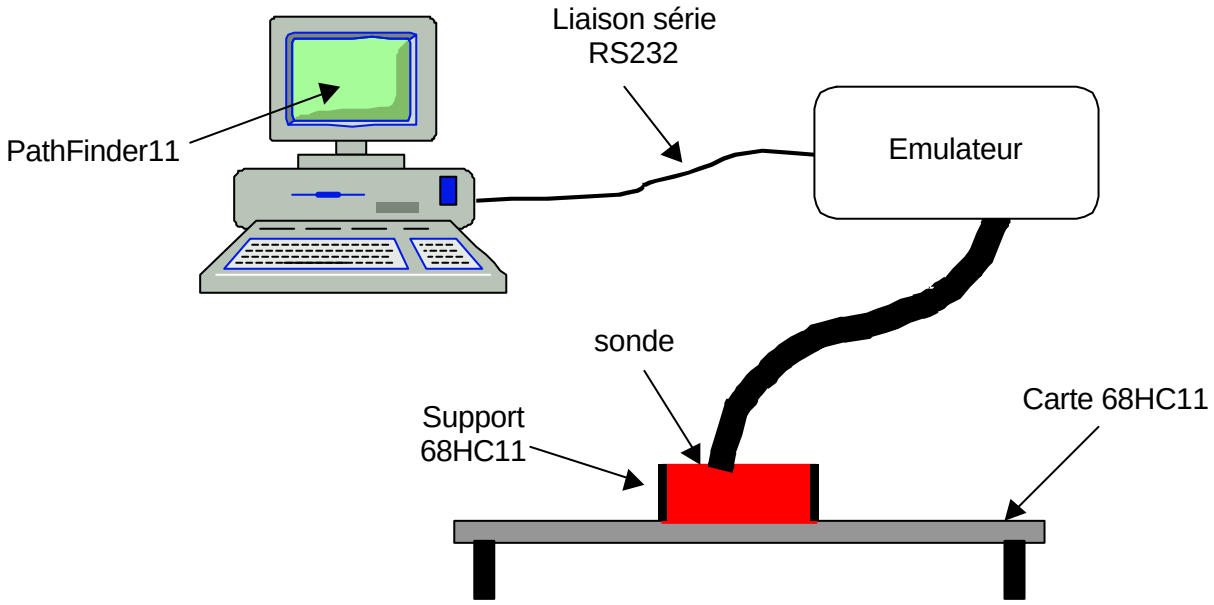


Figure 6 : Configuration matérielle avec l'émulateur

2.4 Le logiciel Surveyor

Ce logiciel est un outil fondamental pour notre projet, en effet il permet de visualiser les différentes trames circulant sur un réseau. Il s'agit d'un analyseur de réseau.

Surveyor offre la possibilité de visualiser les différents champs d'une trame reçue en décomposant cette dernière. Cela permet de vérifier la nature des trames que nous émettons en fonction des trames reçues, notamment en ce qui concerne la coordination des numéros de séquence dans le cas d'une connexion TCP.

Avec le logiciel **Surveyor** on peut également émettre des trames sur le réseau pour tester toutes les configurations et réaliser nos propres scénarios de test.

La décomposition d'une trame par **Surveyor** est représentée par la figure 7.

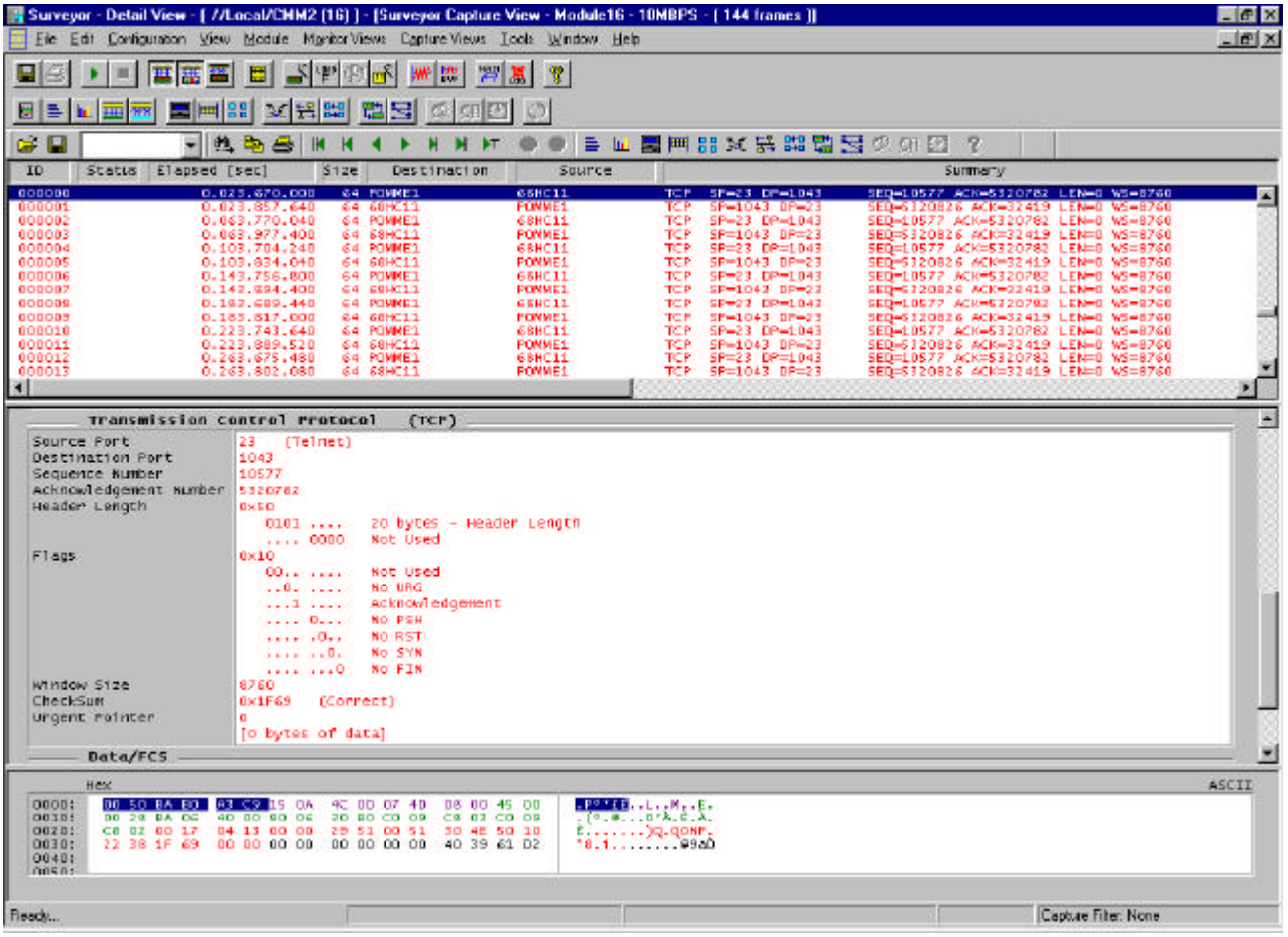


Figure 7 : Interface du logiciel Surveyor

3. Les protocoles Internet

La mise en place des protocoles Internet appelle l'étude des protocoles reliés à Internet. Les différents protocoles utilisés sur le réseau Internet sont : le protocole IP, le protocole ARP, le protocole ICMP, le protocole UDP, le protocole TCP, ainsi que le protocole HTTP.

Un protocole est une méthode standard qui permet la communication entre deux machines, c'est-à-dire un ensemble de règles et de procédures à respecter pour émettre et recevoir des données sur un réseau. Il en existe plusieurs selon ce que l'on attend de la communication. Le protocole ICMP sert à gérer simplement l'état de la transmission et des erreurs de la communication IP.

Sur Internet, les protocoles utilisés font partie d'une suite de protocoles, c'est-à-dire un ensemble de protocoles reliés entre-eux.

La réalisation de notre projet nous a fait travailler tous les protocoles énoncés précédemment. Nous allons décrire ces différents protocoles dans les paragraphes qui suivent. Les protocoles Internet seront transportés par des trames Ethernet compatible MAC 802.3.

3.1 Les trames Ethernet MAC 802.3 10-Base T

3.1.1 Présentation de la trame Ethernet

La trame Ethernet du standard IEEE 802.3 est représentée dans la figure 8.

DA	SA	Length Field	LLC data	PAD	CRC

Figure 8 Trames Ethernet 802.3

- La zone **DA** définit l'adresse de destination de la trame Ethernet. Elle est codée sur 6 octets.
- La zone **SA** définit l'adresse source de la trame Ethernet. Elle est codée sur 6 octets.
- La zone **Length Field** définit la longueur des données contenue dans la zone **LLC data**. Elle est codée sur 2 octets. Le nombre définit correspond aux nombres d'octets contenues dans la trame Ethernet.
- La zone **LLC data** correspond à la zone où sont stockées les données provenant de la couche supérieure (UDP, TCP, …).
- La zone **PAD** permet de remplir la zone de données pour atteindre la valeur de 46 octets qui est la longueur minimale pour que la trame totale atteigne 64 octets avec les octets de préambule et de délimitation.
- La zone **CRC** définit une détection d'erreurs codée sur 4 octets. La detection des erreurs est assurée par un polynôme cyclique g(x) tel que :

$$g(x)=x^{32}+x^{26}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}$$

3.1.2 Implantation sur la carte 68HC11

La carte 68HC11 possède une adresse Ethernet définit dans le fichier *trame.h*. Il est donc possible de la modifier à tout moment.

Actuellement l'adresse Ethernet de la carte 68HC11 est :

ADRESSE_Ethernet_1 21

ADRESSE_Ethernet_2 10
ADRESSE_Ethernet_3 76
ADRESSE_Ethernet_4 13
ADRESSE_Ethernet_5 07
ADRESSE_Ethernet_6 77

3.2 Le protocole ARP (Address Resolution Protocol)

3.2.1 Présentation du protocole ARP

Le protocole ARP a un rôle phare parmi les protocoles de la couche Internet de la suite TCP/IP, car il permet de connaître l'adresse physique d'une carte réseau correspondant à une adresse IP, c'est pour cela qu'il s'appelle *Protocole de résolution d'adresse* (en anglais ARP signifie *Address Resolution Protocol*).

La figure 9 représente la trame ARP.

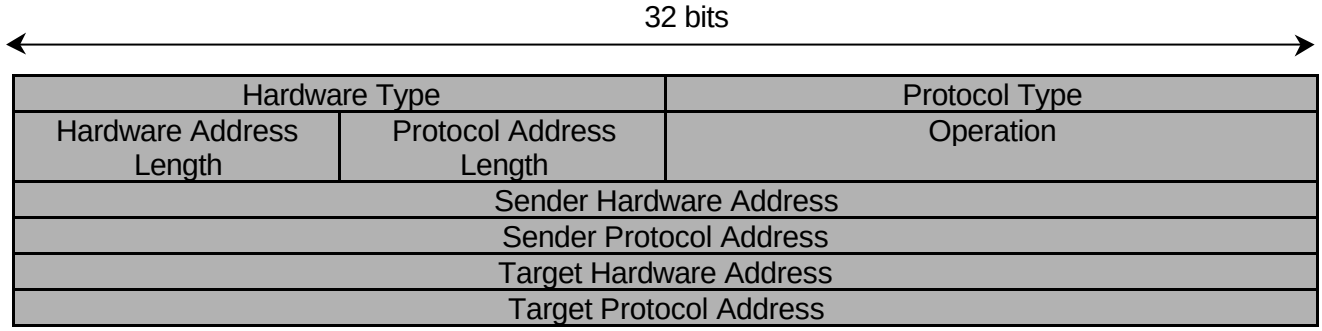


Figure 9 : Datagramme ARP

- **Hardware Type** : il s'agit du type du matériel sur lequel sont transmises les trames du protocole choisit. Ce champ est codé sur 2 octets.
- **Protocol Type** : il s'agit du type de protocole que nous utiliserons pour la communication. Ce champ est codé sur 2 octets.
- **Hardware Address Length** : nos définissons la longueur des adresses physiques en octets. Ce champ est codé sur 1 octet.
- **Protocol Address Length** : nous définissons la longueur des adresses correspondant au protocole choisi. Ce champ est codé sur 1 octet.
- **Operation** : il s'agit de définir le type d'opération de la trame ARP envoyé. Ce champ est codé sur 1 octet.
- **Sender Hardware Address** : adresse physique de l'émetteur de la trame. La longueur de ce champ dépend de la valeur inscrite dans le champ **Hardware Address Length**.
- **Sender Protocol Address** : adresse de l'émetteur de la trame. La longueur de ce champ dépend de la valeur inscrite dans le champ **Protocol Address Length**.
- **Target Hardware Address** : adresse physique du récepteur de la trame. La longueur de ce champ dépend de la valeur inscrite dans le champ **Hardware Address Length**.
- **Target Hardware Address** : adresse du récepteur de la trame. La longueur de ce champ dépend de la valeur inscrite dans le champ **Protocol Address Length**.

Chaque machine connectée au réseau possède un numéro d'identification de 48 bits. Ce numéro est un numéro unique qui est fixé dès la fabrication de la carte en usine. Toutefois la communication sur Internet ne se fait pas directement à partir de ce numéro (car il faudrait modifier l'adressage des ordinateurs à chaque fois que l'on change une carte réseau) mais à partir d'une adresse dite logique attribuée par un organisme: l'adresse IP.

Ainsi, pour faire correspondre les adresses IP aux adresses logiques, le protocole ARP interroge les machines du réseau pour connaître leur adresse physique.

Dans notre cas le protocole ARP émet une requête sur le réseau de type *broadcast*, suite à quoi notre carte 68HC11 consulte son adresse Ethernet qui correspond à son adresse IP.
Toutes les applications implémentées sur la carte 68HC11 (ping, Telnet et HTTP) utilisent le protocole ARP afin de déterminer l'adresse Ethernet de la carte 68HC11. En effet, la caractéristique de ces applications est de travailler directement sur l'adresse IP de la carte 68HC11.

3.2.2 Utilisation pour le protocole IP

Dans notre cas les champs définis ci-dessus auront les valeurs suivantes :
- Hardware Type : 1 (10 Mbps Ethernet).
- Protocol Type : 0x0800 (IP).
- Hardware Address Length : 6.
- Protocol Address Length : 4.

Les champs correspondants aux adresses physiques des différentes cartes seront codés sur 6 octets, et les champs correspondants aux adresses IP des différentes cartes seront codés sur 4 octets.

3.3 Le protocole IP (Internet Protocol)

3.3.1 Présentation du protocole IP

Le datagramme IP est décrit dans la figure 10.

32 bits

Version	Longueur d'en tête	Type de service	Longueur totale
Identification		Drapeau	Décalage fragment
Durée de vie	Protocole	Checksum d'en-tête	
Adresse IP source			
Adresse IP destination			
Données			

Figure 10 : Datagramme IP

- **Version** : il s'agit de la version du protocole IP que nous utilisons. Actuellement la version 4 est utilisée (*IPv4*). Ce champ est codé sur 4 bits.
- **Longueur d'en-tête** : il s'agit du nombre de mots de 32 bits sur lesquels sont répartis les différents champs de l'en-tête. Ce champ est codé sur 1 octet.
- **Type de service** : le type de service donne une indication sur la qualité de service souhaitée, qui reste cependant un paramètre "abstrait". Ce paramètre est utilisé pour "guider" le choix des paramètres des services actuels lorsqu'un datagramme transite dans un réseau particulier. Certains réseaux offrent un mécanisme de priorité, traitant préférentiellement un tel trafic par rapport à un trafic moins prioritaire. Principalement, le choix offert est une négociation entre les trois contraintes suivantes : faible retard, faible taux d'erreur, et haut débit. Ce champ est codé sur 1 octet.
- **Longueur totale** : il s'agit de la longueur du datagramme entier y compris en-tête et données, mesurée en octets. Ce champ ne permet de coder qu'une longueur de datagramme d'au plus 65,535 octets. Ce champ est codé sur 1 octet.
- **Identification** : une valeur d'identification assignée par l'émetteur pour identifier les fragments d'un même datagramme. Ce champ est codé sur 16 bits.
- **Drapeaux (flags)** : Divers commutateurs de contrôle codés sur 3 bits.

- **Déplacement de fragment** : ce champ indique le décalage du premier octet du fragment par rapport au datagramme complet. Cette position relative est mesurée en blocs de 8 octets (64 bits). Le décalage du premier fragment vaut zéro. Ce champ est codé sur 13 bits.
- **Durée de vie**: (TTL: *Time To Live*) ce champ permet de limiter le temps pendant lequel un datagramme reste dans le réseau. Si ce champ prend la valeur zéro, le datagramme doit être détruit. Ce champ est modifié pendant le traitement de l'en-tête Internet. La durée de vie est mesurée en secondes. Chaque module Internet doit retirer au moins une unité de temps à ce champ, même si le traitement complet du datagramme par le module est effectué en moins d'une seconde. Ce champ est codé sur 1 octet.
- **Protocole**: ce champ indique quel protocole de niveau supérieur est utilisé dans la section donnée du datagramme Internet. Ce champ est codé sur 8 bits.
- **Checksum d'en-tête** : ce champ contient une valeur codée sur 16 bits qui permet de contrôler l'intégrité de l'en-tête afin de déterminé si celui-ci n'a pas été altéré pendant la transmission.
- **Adresse IP Source**: ce champ représente l'adresse IP de la machine émettrice, il permet au destinataire de répondre. Ce champ est codé sur 32 bits.
- **Adresse IP destination**: adresse IP du destinataire du message. Ce champ est codé sur 32 bits.

3.3.2 Implantation sur la carte 68HC11

Nous travaillons sur la version 4 du protocole IP.

Nous ne spécifions pas les bits correspondants au type de service souhaitait.

Les protocoles sur lesquels nous travaillons sont : ICMP, TCP et UDP. Les valeurs indiquées dans le champ protocole sont :

- ICMP : 1.
- TCP : 6.
- UDP : 17.

L'algorithme utilisé pour le Checksum est le suivant :

nous calculons le complément à un sur 16 bits de la somme des compléments à un de tous les octets de l'en-tête pris par paires (mots de 16 bits). Lorsque nous calculons le checksum, nous considérons un en-tête dont le champ réservé pour ce même checksum vaut zéro.

La carte 68HC11 possède une adresse IP définit dans le fichier *trame.h*. Il est donc possible de la modifier à tout moment. Actuellement l'adresse IP de la carte 68HC11 est :

ADRESSE_IP_1 192
ADRESSE_IP_2 9
ADRESSE_IP_3 200
ADRESSE_IP_4 3

3.4 Le protocole ICMP (Internet Control Message Protocol)

3.4.1 Présentation du protocole ICMP

Le protocole ICMP est un protocole qui permet de gérer les informations relatives aux erreurs dues aux machines connectées. Etant donné le peu de contrôles que le protocole IP réalise, il permet de faire-part de ces erreurs aux protocoles des couches voisines. Tous les routeurs l'utilisent pour reporter une erreur (*Delivery Problem*).

Les messages ICMP reportent principalement des erreurs concernant le traitement d'un datagramme dans un module IP. Pour éviter de ne pas entrer dans un cercle vicieux de réémission de message de contrôle en réponse à un autre message de contrôle et ce sans fin, aucun message ICMP ne sera réémis en réponse à un message ICMP. De même les messages ICMP ne

seront transmis qu'en réponse à un traitement erroné du fragment zéro dans le cas d'un datagramme fragmenté. (Le fragment zéro est celui dont l'offset vaut zéro).

Les messages ICMP sont émis en utilisant l'en-tête IP de base. Le premier octet de la section de données du datagramme est le champ de type ICMP; Sa valeur détermine le format du reste des données dans le datagramme ICMP.

Sauf mention particulière contraire signalée dans chaque description de message spécifique, les valeurs des champs d'en-tête Internet auront la signification suivante:

- Version : 4.
- IHL : Longueur d'en-tête Internet en mots de 32-bits.
- Type de Service : 0.
- Longueur Totale : longueur totale du datagramme en octets.
- Identification :
- Bits Contrôles et Fragment Offset : Utilisés par le mécanisme de fragmentation.
- Durée de vie : durée de vie du datagramme en secondes.
- Protocole : ICMP = 1.
- Checksum : Le complément à un sur 16 bits de la somme des compléments à un de l'en-tête Internet pris par mots de 16 bits.
- Adresse source : L'adresse de l'hôte qui compose le message ICMP.
- Adresse destinataire : L'adresse de l'hôte à qui le message doit être envoyé.

Le datagramme ICMP est décrit dans la figure 11.

32 bits

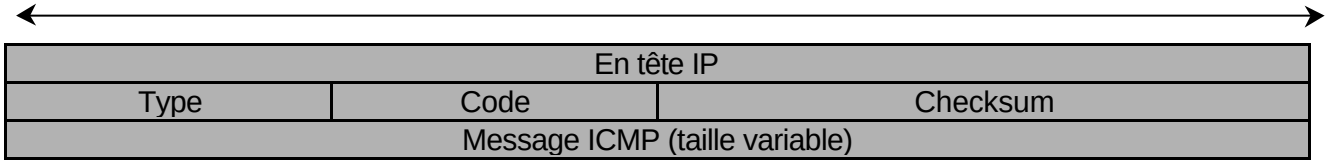


Figure 11 : Datagramme ICMP

- **Type** : sa valeur détermine le format du reste des données dans le datagramme ICMP.
- **Code** : sa valeur détermine le message implémentée dans le datagramme ICMP.
- **Checksum** : le complément à un sur 16 bits de la somme des compléments à un du message ICMP.
- **Message ICMP** : sa valeur varie avec le type et le code du datagramme ICMP.

3.4.2 Implantation sur la carte 68HC11

Nous avons implémenté les requêtes correspondant à une commande *ping*. Les messages ICMP correspondants sont : *echo* et *echo reply*.

Le datagramme décrivant les messages ICMP *echo* et *echo reply* se trouve dans la figure 12.

32 bits

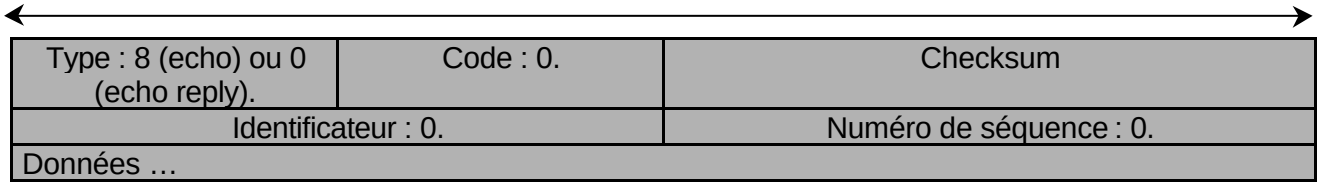


Figure 12 : Datagramme du message ICMP

Les données reçues dans un message d'écho doivent être réémises dans la réponse à l'écho.

3.5 Le protocole UDP (User Datagram Protocol)

3.5.1 Présentation du protocole UDP

Le protocole UDP est défini dans le but de fournir une communication par paquet unique entre deux processus dans un environnement réseau étendu. Ce protocole suppose l'utilisation du protocole IP comme support de base à la communication.

Ce protocole définit une procédure permettant à une application d'envoyer un message court à une autre application, selon un mécanisme minimaliste. Ce protocole est transactionnel, et ne garantit ni la délivrance du message, ni son éventuelle duplication.

Le format de l'en-tête UDP est décrit dans la figure 13.

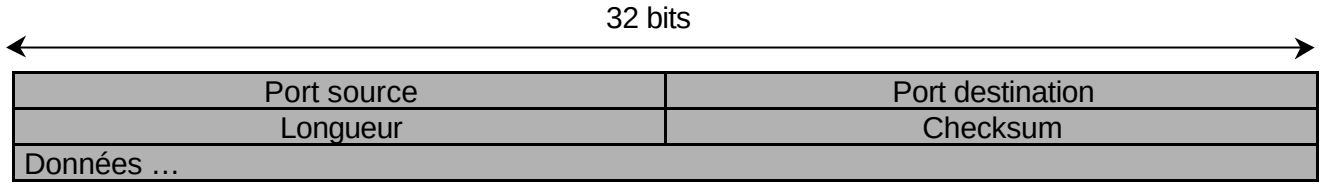


Figure 13 : Datagramme UDP

- **Port source** : c'est un champ optionnel. Lorsqu'il est significatif, il indique le numéro de port du processus émetteur.
- **Port Destinataire** : il indique le numéro de port de l'application réceptrice.
- **Longueur** : compte le nombre d'octets dans le datagramme entier y compris le présent en-tête. (Et par conséquent la longueur minimale mentionnée dans ce champ vaut huit, si le datagramme ne transporte aucune donnée).
- **Checksum** : il se calcule en prenant le complément à un de la somme sur 16 bits des compléments à un calculé sur un pseudo en-tête constitué de l'information typique d'un en-tête IP, l'en-tête UDP elle-même, et les données, le tout additionné d'un octet nul éventuel afin que le nombre total d'octets soit pair.

Le pré en-tête ajouté avant l'en-tête UDP contient l'adresse IP source, l'adresse IP destinataire, le code de protocole, et la longueur du segment UDP. Cette information permet d'augmenter l'immunité du réseau aux erreurs de routage de datagrammes.

Le datagramme du pré en-tête est décrit dans la figure 14.

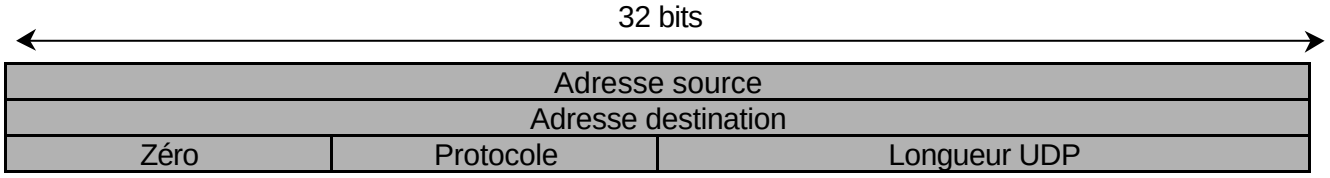


Figure 14 : Datagramme du pré en-tête UDP

3.5.2 Implantation sur la carte 68HC11

Le module UDP implanté dans la carte 68HC11 est utilisé pour afficher sur l'écran de l'*hyperterminal* toutes les données contenues dans la trame UDP.
De nouvelles applications pourront être mises en place pour gérer au mieux les données contenues dans chaque trame.

3.6 Le protocole TCP (Transmission Control Protocol)

Le protocole TCP est défini dans le but de fournir un service de transfert de données de haute fiabilité entre deux ordinateurs "maîtres" raccordés sur un réseau de type "paquets commutés", et sur tout système résultant de l'interconnexion de ce type de réseaux.

TCP est un protocole orienté connexion, c'est-à-dire qu'il permet à deux machines qui communiquent de contrôler l'état de la transmission.

Les caractéristiques principales du protocole TCP sont les suivantes:

- TCP permet de remettre en ordre les datagrammes en provenance du protocole IP.
- TCP permet de vérifier le flot de données afin d'éviter une saturation du réseau.
- TCP permet de formater les données en segments de longueur variable afin de les "remettre" au protocole IP.
- TCP permet de multiplexer les données, c'est-à-dire de faire circuler simultanément des informations provenant de sources (applications par exemple) distinctes sur une même ligne.
- TCP permet enfin l'initialisation et la fin d'une communication de manière courtoise.

Grâce au protocole TCP, les applications peuvent communiquer de façon sûre (grâce au système d'accusés de réception du protocole TCP), indépendamment des couches inférieures.

Lors d'une communication à travers le protocole TCP, les deux machines doivent établir une connexion. La machine émettrice (celle qui demande la connexion) est appelée client, tandis que la machine réceptrice est appelée serveur. On dit qu'on est alors dans un environnement client-serveur.

Les machines dans un tel environnement communiquent en full-duplex, c'est-à-dire que la communication se fait dans les deux sens.

Pour permettre le bon déroulement de la communication et de tous les contrôles qui l'accompagnent, les données sont encapsulées, c'est-à-dire qu'on ajoute aux paquets de données un en-tête qui va permettre de synchroniser les transmissions et d'assurer leur réception.

Une autre particularité de TCP est de pouvoir réguler le débit des données grâce à sa capacité à émettre des messages de taille variable, ces messages sont appelés *segments*.

TCP permet d'effectuer une tâche importante: le multiplexage/démultiplexage. Nous sommes capables de faire transiter sur une même ligne des données provenant d'applications diverses. Ces opérations sont réalisées grâce au concept de ports (ou sockets), c'est-à-dire un numéro associé à un type d'application, qui, combiné à une adresse IP, permet de déterminer de façon unique une application qui tourne sur une machine donnée.

La constitution d'une trame TCP est décrite dans la figure 15.

32 bits

<table>
<tr><td colspan="5">Port source</td><td colspan="4">Port destinataire</td></tr>
<tr><td colspan="9">Numéro de séquence</td></tr>
<tr><td colspan="9">Accusé de réception</td></tr>
<tr><td>Data offset</td><td>Réservé</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="2">Fenêtre</td></tr>
<tr><td colspan="5">Checksum</td><td colspan="4">Pointeur de données urgentes</td></tr>
<tr><td colspan="7">Options</td><td colspan="2">Remplissage</td></tr>
<tr><td colspan="9">Données …</td></tr>
</table>

Figure 15 : Datagramme de l'en-tête TCP

- **Port Source** (16 bits): port relatif à l'application en cours sur la machine source.

- **Port Destination** (16 bits): port relatif à l'application en cours sur la machine de destination
- **Numéro de séquence** (16 bits): lorsque le drapeau SYN est à 0, le numéro d'ordre est celui du premier mot du segment en cours. Lorsque SYN est à 1, le numéro de séquence est le numéro de séquence initial utilisé pour synchroniser les numéros de séquence (ISN).
- **Accusé de réception** (32 bits): dernier segment reçu par le récepteur.
- **Data offset** (4 bit): il permet de repérer le début des données dans le paquet. Le décalage est ici essentiel car le champ d'options est de taille variable.
- **Réservé** (6 bits): champ inutilisé actuellement mais prévu pour l'avenir.
- **Drapeaux (flags)** (6x1 bit): Les drapeaux représentent des informations supplémentaires:
  - **URG**: si ce drapeau est à 1 le paquet doit être traité de façon urgente.
  - **ACK**: si ce drapeau est à 1 le paquet est un accusé de réception.
  - **PSH** (PUSH): si ce drapeau est à 1, le paquet fonctionne suivant la méthode PUSH.
  - **RST**: si ce drapeau est à 1, la connexion est réinitialisée.
  - **SYN**: si ce drapeau est à 1, les numéros d'ordre sont synchronisés.
  - **FIN**: si ce drapeau est à 1 la connexion s'interrompt.
- **Fenêtre** (16 bits): champ permettant de connaître le nombre d'octets que le récepteur souhaite recevoir sans accusé de réception.
- **Checksum** : le checksum est constitué en calculant le complément à 1 sur 16 bits de la somme des compléments à 1 des octets de l'en-tête et des données pris deux par deux (mots de 16 bits). Si le message entier contient un nombre impair d'octets, un 0 est ajouté à la fin du message pour terminer le calcul du checksum. Cet octet supplémentaire n'est pas transmis. Lors du calcul du checksum, les positions des bits attribués à celui-ci sont marquées à 0.
  Le checksum couvre de plus une pseudo en-tête de 96 bits préfixée à l'en-tête TCP. Cette pseudo en-tête comporte les adresses Internet source et destinataires, le type de protocole et la longueur du message TCP. Ceci protège TCP contre les erreurs de routage. Cette information sera véhiculée par IP, et est donnée comme argument par l'interface TCP/Réseau lors des appels d'IP par TCP.
  La longueur TCP compte le nombre d'octets de l'en-tête TCP et des données du message, en excluant les 12 octets de la pseudo en-tête.

  Le datagramme du pré en-tête TCP est décrit dans la figure 16.

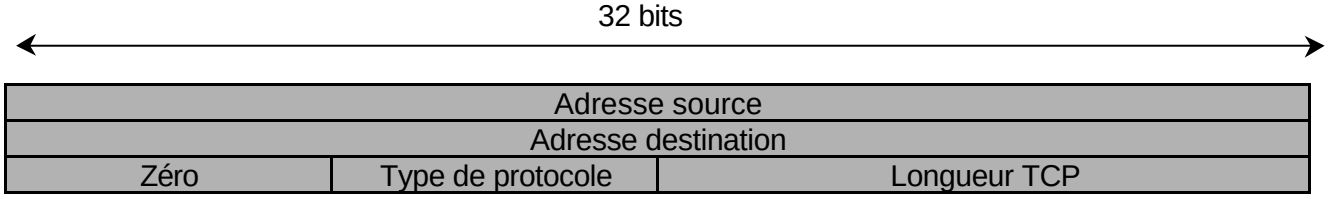


Figure 16: Datagramme du pré en-tête TCP

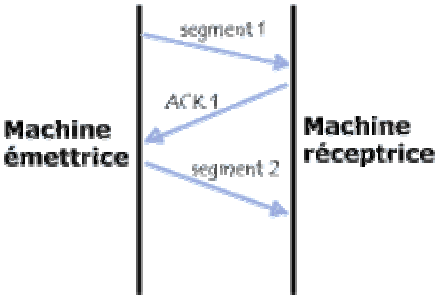
- **Pointeur de données urgentes** (16 bits): indique le numéro d'ordre à partir duquel l'information devient urgente.
- **Options** (Taille variable): des options diverses.
- **Remplissage**: on remplit l'espace restant après les options avec des zéros pour avoir une longueur de 32 bits.

3.6.1 Fiabilité des transferts

Le protocole TCP permet d'assurer le transfert des données de façon fiable, bien qu'il utilise le protocole IP, qui n'intègre aucun contrôle de livraison de datagramme.

En réalité, le protocole TCP possède un système d'accusé de réception, grâce au drapeau de contrôle ACK.

Lors de l'émission d'un segment, un numéro de séquence lui est associé. A la réception d'un segment de donnée, la machine réceptrice va retourner un accusé de réception dont le drapeau ACK est à 1 accompagné d'un numéro d'accusé de réception égal au numéro de séquence du prochain segment émit par l'émetteur.



3.6.2 Etablissement d'une connexion

Etant donné que ce processus de communication qui se fait grâce à une émission de données et d'un accusé de réception, est basé sur un numéro de séquence, il faut que les machines émettrices et réceptrices (client et serveur) connaissent le numéro de séquence initial de l'autre machine.

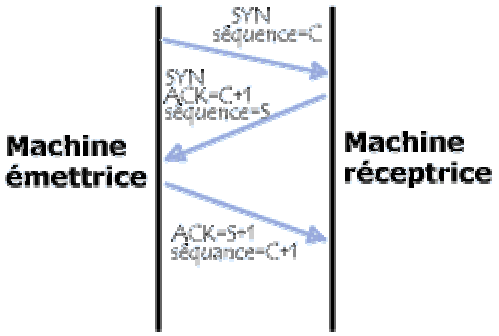
L'établissement de la connexion entre deux applications se fait souvent selon le schème suivant:

- Les ports TCP doivent être ouverts.
- L'application sur le serveur est passive, c'est-à-dire que l'application est à l'écoute, en attente d'une connexion.
- L'application sur le client fait une requête de connexion sur le serveur dont l'application est en ouverture passive. L'application du client est dite "en ouverture passive".

Les deux machines doivent donc synchroniser leurs séquences grâce à un mécanisme communément appelé *three-ways-handshake* (*poignée de main en trois temps*), que l'on retrouve aussi lors de la clôture de session.

Ce dialogue permet d'initier la communication, il se déroule en trois temps, comme sa dénomination l'indique:

- Dans un premier temps la machine émettrice (le client) transmet un segment dont le drapeau SYN est à 1 (pour signaler qu'il s'agit d'un segment de synchronisation), avec un numéro de séquence N, que l'on appelle numéro d'ordre initial du client
- Dans un second temps la machine réceptrice (le serveur) reçoit le segment initial provenant du client, puis lui envoie un accusé de réception, c'est-à-dire un segment dont le drapeau ACK est à 1 et le drapeau SYN est à 1 (car il s'agit là encore d'une synchronisation). Ce segment contient le numéro de séquence de cette machine (du serveur) qui est le numéro de séquence initial du client. Le champ le plus important de ce segment est le champ accusé de réception qui contient le numéro d'ordre initial du client, incrémenté de 1.
- Enfin, le client transmet au serveur un accusé de réception, c'est-à-dire un segment dont le drapeau ACK est à 1, dont le drapeau SYN est à zéro (il ne s'agit plus d'un segment de synchronisation). Son numéro d'ordre est incrémenté et le numéro d'accusé de réception représente le numéro de séquence initial du serveur incrémenté de 1.



3.6.3 La machine d'état d'une connexion TCP

Une connexion connaît plusieurs états durant sa durée de vie. Les états définis sont: LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, et CLOSED.
CLOSED est dit fictif car il correspond à une situation où la connexion elle-même n'existe plus.

Nous donnons ci-dessous un descriptif rapide des états cités:

- **LISTEN** : la connexion reste en attente d'une requête de connexion externe par un TCP distant. Cet état est atteint après une demande de connexion passive.
- **SYN-SENT** : la connexion se met en attente d'une requête de connexion, après avoir envoyé elle-même une requête à un destinataire.
- **SYN-RECEIVED** : les deux requêtes de connexion se sont croisées. La connexion attend confirmation de son établissement.
- **ESTABLISHED** : la connexion a été confirmée de part et d'autre et les données peuvent transiter sur la voie de communication. C'est l'état stable actif de la connexion.
- **FIN-WAIT-1** : sur requête de déconnexion émise par l'application, la connexion demande la confirmation d'une requête de déconnexion qu'elle a elle-même émise vers le distant.
- **FIN-WAIT-2** : la connexion se met en attente d'une requête de déconnexion par le distant, une fois reçue la confirmation de sa propre requête.
- **CLOSE-WAIT** : la connexion se met en attente d'une requête de déconnexion émise par l'application.
- **CLOSING** : la connexion attend la confirmation de sa requête de déconnexion par le TCP distant, lequel avait auparavant émis sa propre requête de déconnexion.
- **LAST-ACK** : la connexion attend la confirmation de sa requête de déconnexion, émise suite à une requête similaire à l'initiative du distant.
- **TIME-WAIT** : un temps de latence avant fermeture complète du canal, pour s'assurer que toutes les confirmations ont bien été reçues.
- **CLOSED** : la connexion n'existe plus. C'est un pseudo-état.

La figure 17 montre l'enchaînement des états et les différentes trames émises. Il occulte par contre le traitement des fautes, ainsi que tous les autres événements qui ne sont pas en relation avec les changements d'état.

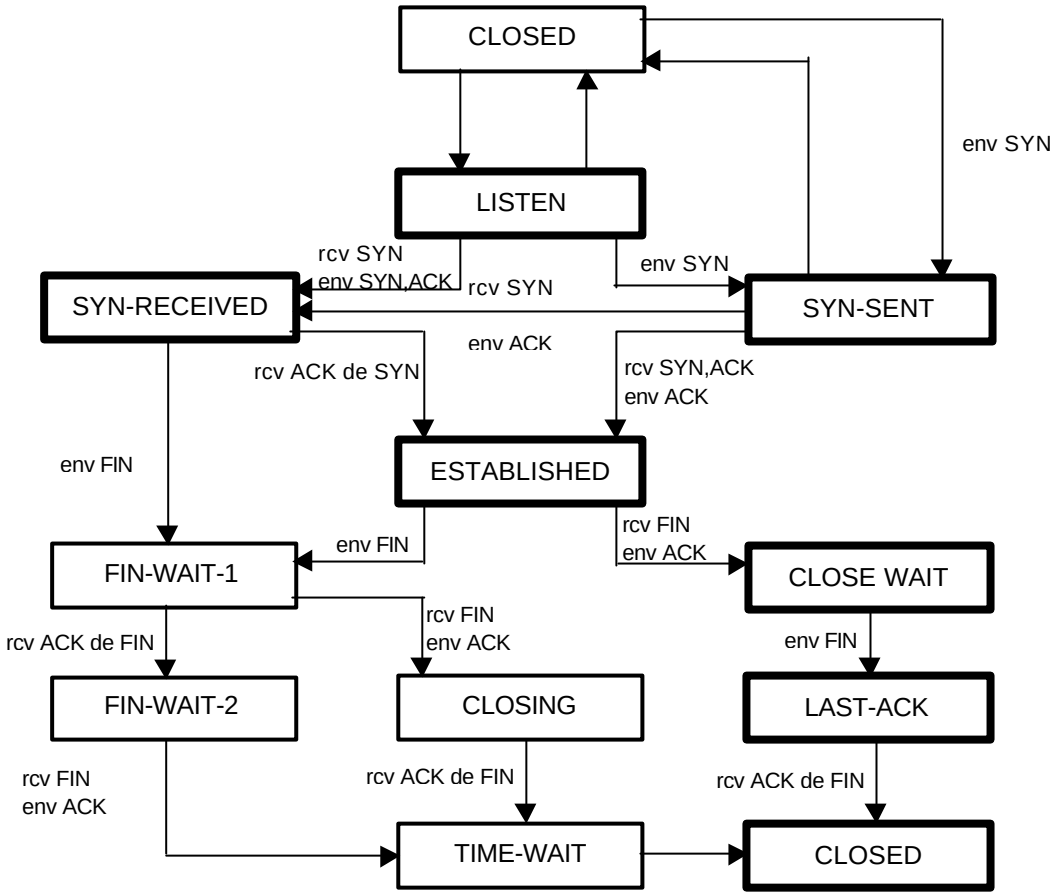


Figure 17: Diagramme d'état d'une connexion TCP

Le diagramme décrit ci-dessus est un résumé et ne doit pas être compris comme une spécification complète.

3.6.4 Implantation sur la carte 68HC11

Le protocole TCP implanté sur la carte 68HC11 traite principalement les événements en relation directe avec la réception des données. En effet, nous ne gérons pas la partie de la machine d'état mettant la carte 68HC11 dans l'état d'émetteur d'information: nous ignorons les états FIN-WAIT-1, FIN-WAIT-2, CLOSING, TIME-WAIT.

Nous traitons les trames reçues en partant de l'état de départ LISTEN. Etant donné que la carte se comporte comme un récepteur de données nous n'avons pas mis en place les procédures nécessaires au passage de l'état CLOSED à l'état LISTEN. Ainsi, nous avons ignorer l'état CLOSED.

Les ports de communication fonctionnant grâce au protocole TCP sont les ports:

- 23: application TELNET.
- 80 : application HTTP.

3.7 Le protocole http (Hyper Text Transfer Protocol)

Le protocole HTTP est le protocole le plus utilisé sur Internet depuis 1990. Il permet désormais de transférer des messages avec des en-têtes décrivant le contenu du message en utilisant un codage de type MIME (Multipurpose Internet Mail Extensions).

Le but du protocole HTTP est de permettre un transfert de fichiers (essentiellement au format HTML) localisé grâce à une chaîne de caractères appelée URL (Uniform Resource Locator) entre un navigateur (le client) et un serveur Web.

3.7.1 Communication entre navigateur et serveur

La communication entre le navigateur et le serveur se déroule en deux temps:

- Le navigateur effectue une **requête http**.
- Le serveur traite la requête puis envoie une **réponse http**.

La figure 18 décrit le processus correspondant.

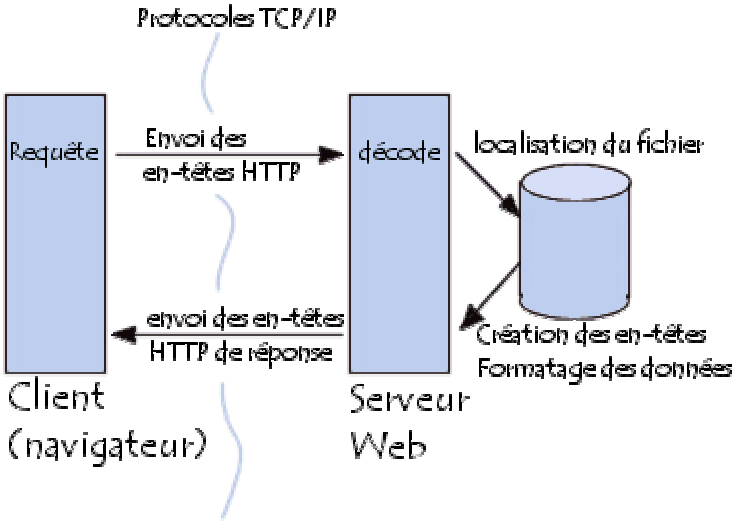


Figure 18 : Communication entre le serveur et le navigateur

3.7.1.1 Une requête HTTP

Une requête HTTP est un ensemble de lignes envoyé au serveur par le navigateur. Elle comprend:

- **Une ligne de requête**: permet de préciser le type de document demandé, la méthode qui doit être appliqué, et la version du protocole utilisée. La ligne comprend trois éléments devant être séparé par un espace:
  - La méthode.
  - L'URL.
  - La version du protocole utilisé par le client (généralement *HTTP/1.0*).
- **Les champs d'en-tête de la requête**: il s'agit d'un ensemble de lignes facultatives permettant de donner des informations supplémentaires sur la requête et/ou le client (Navigateur, système d'exploitation,...). Chacune de ces lignes est composé d'un nom qualifiant le type d'en-tête, suivi de deux points (:) et de la valeur de l'en-tête.

- **Le corps de la requête**: C'est un ensemble de ligne optionnel devant être séparé des lignes précédentes par une ligne vide et permettant par exemple un envoi de données par une commande POST lors de l'envoi de données au serveur par un formulaire.

Une requête HTTP a donc la syntaxe suivante (<*crlf*> signifie retour chariot ou saut de ligne):

METHODE URL VERSION<*crlf*>
EN-TETE : Valeur<*crlf*>
.
.
.
EN-TETE : Valeur<*crlf*>
Ligne vide<*crlf*>
CORPS DE LA REQUETE

Un exemple de requête HTTP:

GET http://www.192.9.200.3 HTTP/1.0
Accept : text/html
If-Modified-Since : Saturday, 15-January-2000 14:37:11 GMT
User-Agent : Mozilla/4.0 (compatible; MSIE 5.0; Windows 95)

3.7.1.2 Une réponse HTTP

Une réponse HTTP est un ensemble de lignes envoyé au navigateur par le serveur. Elle comprend:

- **Une ligne de statut**: c'est une ligne précisant la version du protocole utilisé et l'état du traitement de la requête à l'aide d'un code et d'un texte explicatif. La ligne comprend trois éléments devant être séparé par un espace:
  - La version du protocole utilisé.
  - Le code de statut.
  - La signification du code.
- **Les champs d'en-tête de la requête**: il s'agit d'un ensemble de lignes facultatives permettant de donner des informations supplémentaires sur la réponse et/ou le serveur. Chacune de ces lignes est composé d'un nom qualifiant le type d'en-tête, suivi de deux points (:) et de la valeur de l'en-tête.
- **Le corps de la réponse**: contient le document demandé.

Une réponse HTTP a donc la syntaxe suivante (<*crlf*> signifie retour chariot ou saut de ligne):
VERSION-HTTP CODE EXPLICATION<*crlf*>
EN-TETE : Valeur<*crlf*>
.
.
.
EN-TETE : Valeur<*crlf*>
Ligne vide<*crlf*>
CORPS DE LA REPONSE

Un exemple de réponse HTTP:

HTTP/1.0 200 OK
Date : Sat, 15 Jan 2000 14:37:12 GMT
Server : Microsoft-IIS/2.0
Content-Type : text/HTML

Content-Lentgh : 1245
Last-Modified : Fri, 14 Jan 2000 08:25:13 GMT

3.7.2 Implantation sur la carte 68HC11

Nous avons mis en place une application http permettant de communiquer avec un navigateur Internet (Netscape Navigator, Internet Explorer). En effet, en réponse à une requête http nous envoyons les données nécessaires à l'affichage d'une page web sur le navigateur du client. La page web se trouve en annexe 4.

4. Structure du programme

4.1 Le noyau temps réel µC/OS II

Comme nous l'avons précisé précédemment, pour réaliser l'implantation des différentes couches réseau, nous avons utilisé un noyau temps réel. Dans le paragraphe qui suit, nous allons essayer d'expliquer les particularités de ce noyau, et son fonctionnement.

Le noyau temps réel µC/OS II (Micro Controller Operating System version 2) à la particularité d'être distribué gratuitement. Il est portable sur un très grand nombre de microprocesseurs, de microcontrôleur et de DSP, et notamment le 68HC11. Ce noyau temps réel est programmé en C ANSI.

Il est prévu pour être implanté dans une ROM, et donc parfaitement adapté à notre application. µC/OS est un noyau préemptif, c'est-à-dire qu'il est possible d'interrompre une tâche par une interruption et de reprendre son cours une fois que l'exécution de l'interruption est terminée. Il est, de plus, déterministe parce que les différentes tâches ont des temps d'exécution déterminés.

A chaque fois qu'une tâche est créée, une priorité par rapport aux autres tâches lui est affectée. Dans le cas du noyau µC/OS II, plus le nombre correspondant à la priorité est faible, plus la tâche est prioritaire. Lors de l'exécution du programme, la priorité d'une tâche peut être modifiée; la priorité des tâches peut varier dynamiquement. Ce noyau autorise la création d'au plus 62 tâches.

Le scheduler ou l'ordonnanceur garantit à chaque tâche un temps d'allocation donné, il gère la priorité d'intervention des différentes tâches.

Il faut également noter que nous sommes dans une configuration multitâche / monoprocesseur, il est ainsi possible d'effectuer plusieurs tâches indépendantes simultanément à l'échelle humaine. Au niveau du processeur, une seule tâche est effectuée à la fois, mais le multitâche permet d'éliminer les temps machines inutilisés (boucle d'attente...).

La figure 19 représente les différents états que peut prendre une tâche :

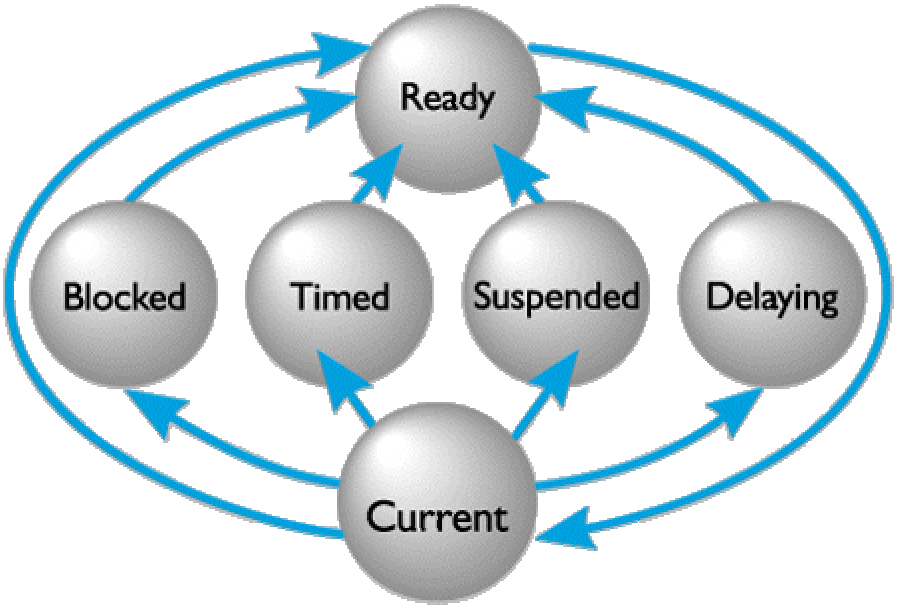


Figure 19 : Changement d'état d'une tâche

4.2 Décomposition du projet

- Analyse du travail effectué l'année précédente.
- Etude du noyau temps réel µC/OS II.
- Analyse du travail à réaliser.
- Réalisation de TP pour se familiariser avec la carte et prendre en main le noyau temps réel.
- Programmation des initialisations.
- Essais d'émission et de réception de trames.
- Programmation du protocole IP.
- Installation du PING : installation du protocole ARP, implantation de *l'Echo Reply* au niveau ICMP.
- Programmation du protocole UDP.
- Programmation de la tâche d'application UDP.
- Programmation du protocole TCP.
- Essai avec le TELNET.
- Essai du HTTP.
- Ecriture de la fonction de traitement HTTP.
- Ecriture de la fonction de traitement TELNET.
- Mise au propre du logiciel.

4.3 Analyse du projet

Bien évidemment, nous n'avons pas programmé tous les protocoles en une seule fois, mais nous les avons implantés les uns après les autres. Le développement de ce projet a été réalisé en C. Le langage C que nous avons utilisé n'est pas réellement du C ANSI. En effet, il est adapté au compilateur Cosmic, qui permet de générer de l'assembleur pour le 68HC11. Nous avons pu noter quelques différences par rapport à une programmation en C classique.

Par exemple, il ne faut pas initialiser les tableaux de données au moment de leur déclaration. De même, la gestion d'une chaîne de caractère avec un pointeur est très délicate.

Pour implanter les protocoles que nous avons décris dans le chapitre précédent, nous avons décidé d'utiliser une tâche pour chaque niveau de protocole, découpé comme le montre la figure 20.

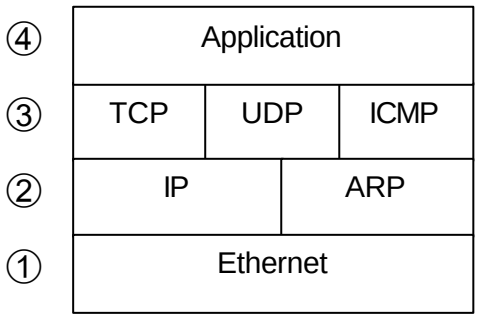


Figure 20 : Représentation des différents niveaux protocolaires

4.3.1 Diagramme de contexte de données (DCD)

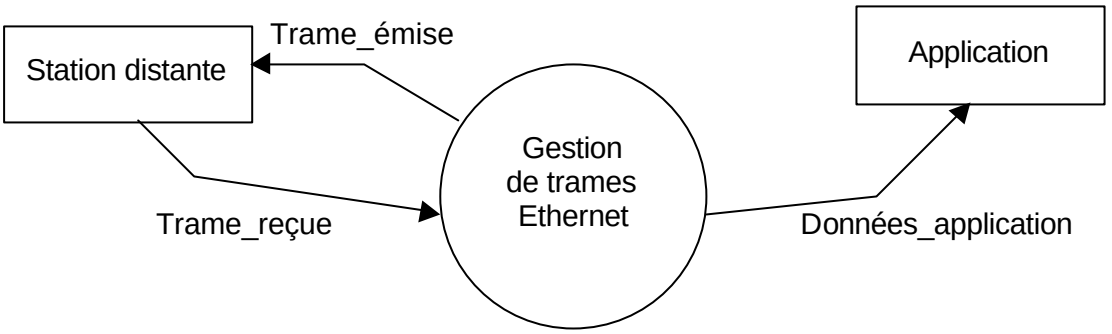


Figure 21 : Diagramme de contexte de données

4.3.2 Diagramme de flots de données contextuel (DFDC)

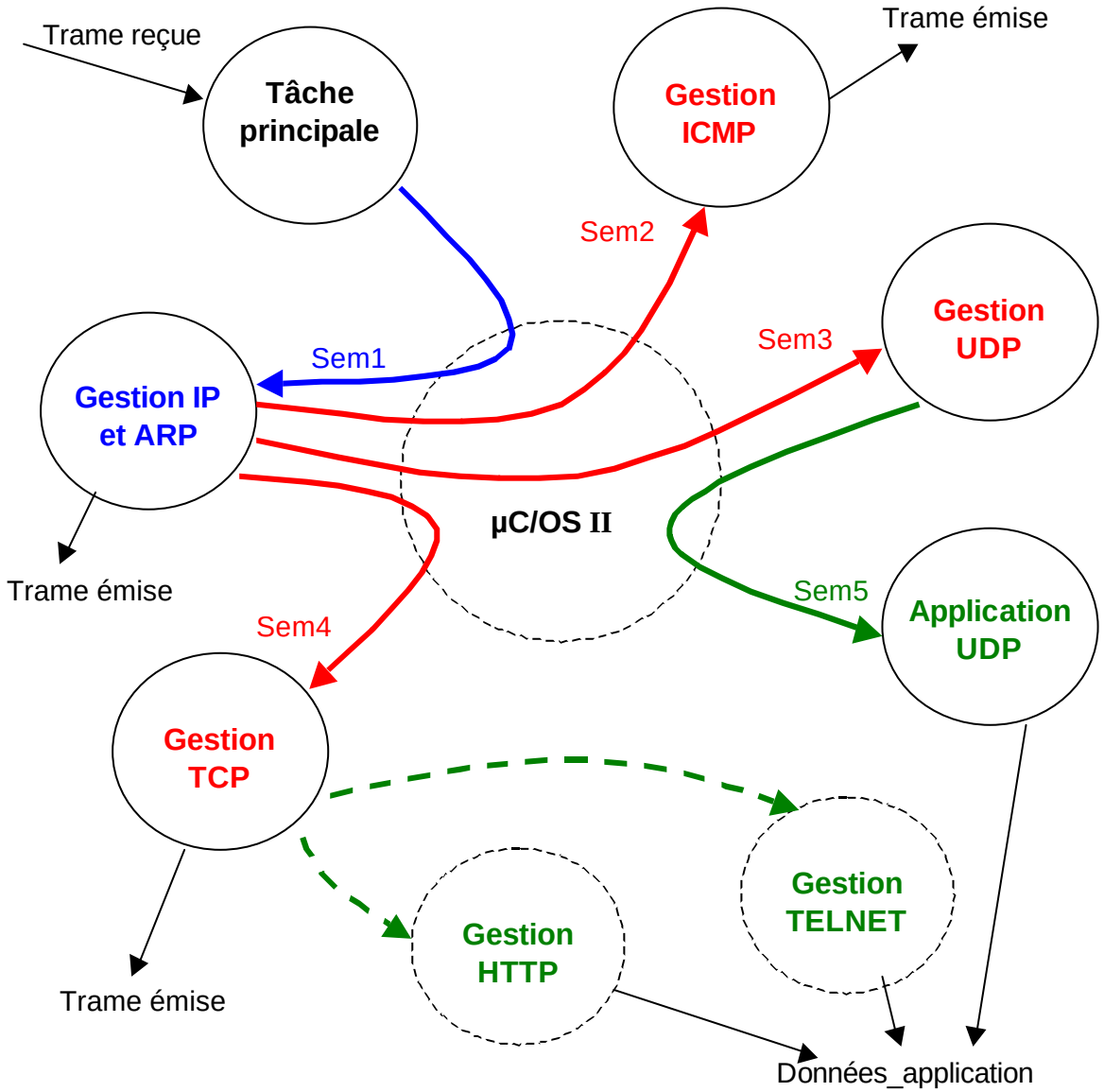


Figure 22 : Diagramme de flot de données

Le diagramme de flot de données représenté par la figure 22 représente la coordination entre les tâches. Nous avons volontairement mis des couleurs différentes, pour identifier le niveau de protocole pour lequel chaque tâche intervient.

Certaines ressources sont accessibles en tous points du programme. C'est le cas pour la trame reçue (ZoneRx) et la trame à émettre (ZoneTx). Ainsi, nous évitons le passage d'un grand nombre de paramètres entre les fonctions, mais la ressource est rendue plus vulnérable. Il faut manipuler ces paramètres avec précaution.

4.3.3 Description des tâches

4.3.3.1 La tâche principale : TachePrincipale( )

Cette tâche a pour but de scruter l'arrivée d'une trame. Une fois qu'une trame est arrivée, elle positionne le sémaphore Sem1 de manière à activer la tache Gestion IP et ARP. Comme vous pouvez le constater nous effectuons directement le traitement au niveau IP (ou ARP) et non au niveau Ethernet. Il ne s'agit pas d'un oubli, mais les opérations de validité de la trame Ethernet et de vérification du CRC sont réalisées par le composant CS8900. Dans le cas où la trame reçue comporte une erreur à ce niveau, le CS8900 peut automatiquement la rejeter.

La trame à laquelle nous avons accès est déjà dépourvue de son CRC. De même, lorsque nous envoyons une trame sur le réseau, le CRC est automatiquement calculé et ajouté à la trame transmise par l'interface Ethernet.

La scrutation de l'arrivée des trames est réalisée par la fonction : **TestRxPoll( )**

4.3.3.2 La tâche Gestion IP et ARP : TacheIP( )

Comme nous l'avons précisé dans le paragraphe précédent, cette tâche est appelée à chaque fois qu'une trame, valide au niveau Ethernet, est reçue. A ce niveau, nous allons vérifier la cohérence des informations de niveau 2 sur la figure 20, c'est à dire au niveau protocolaire IP ou ARP. Plusieurs étapes de vérification sont réalisées avant de faire un traitement au niveau supérieur (niveau 3). Si un problème est détecté ici, la trame est ignorée.

Cette tâche présente deux cas de traitement en fonction du protocole utilisé. La détection du type de protocole est réalisée par les fonctions **VerifTrameIP( )** et **VerifTrameARP( )** :

1) Cas du protocole IP

Dans un premier temps, nous calculons le checksum du niveau IP, grâce à la fonction **VerifChecksumIP( )**. La valeur calculée est ensuite comparée à celle inscrite dans le champs IPHeadChecksum de la trame. Si les deux valeurs sont identiques, nous passons à l'étape suivante.

La seconde étape consiste à vérifier si l'adresse IP de destination est bien destinée à notre application. Pour cela nous comparons cette valeur reçue avec une adresse IP que nous avons définit arbitrairement. Ceci est réalisé par la fonction **VerifAdresseIP( )**. Dans le cas où les deux adresses sont identiques, on atteint la troisième étape.

Cette dernière étape consiste à définir le protocole de niveau 3 (ICMP, UDP ou TCP) qui est concerné par la trame reçue et à positionner le sémaphore (Sem2, Sem3, Sem4) qui activera la

tâche de niveau 3 correspondante. Si la valeur indiquée dans le champ protocole de la trame ne correspond à aucun des trois cités précédemment, la trame est rejetée.

2) Cas du protocole ARP

Dans ce cas, nous effectuons une vérification de la commande ARP envoyée. Cette commande est un *broadcast* : la trame est envoyée à toutes les machines du réseau et celle dont l'adresse IP correspond à l'adresse IP contenue dans la trame ARP renvoie son adresse Ethernet.

Nous vérifions que la trame reçue correspond à un *Echo* et que la requête nous est bien adressée grâce à la fonction **VerifPingARP( )**.

Une fois tous ces paramètres corrects, nous construisons la trame réponse à ce *broadcast* en envoyant cette fois la réponse *Reply*, ainsi que les adresses nécessaires, grâce à la fonction **PingARP( )**.

Enfin, lorsque la trame réponse est construite, nous l'envoyons à l'interface CS8900 grâce à la fonction **cs8900TxFrame( )** à laquelle nous devons fournir l'adresse de départ de la trame et le nombre de mots de 16 bits la composant. Ce sera cette même fonction que nous utiliserons dans le reste du programme pour émettre des trames sur le réseau.

4.3.3.3 La tâche Gestion ICMP : TacheICMP( )

Cette tâche intervient au niveau 3 (figure 20). Elle est activée lorsque tous les paramètres IP sont corrects et que le protocole est ICMP.

Tout d'abord, la fonction **VerifChecksumICMP( )** effectue le calcul de la checksum ICMP et la compare avec la valeur contenue dans le champ correspondant de la trame.

Etant donné que nous avons implanté ce protocole, pour vérifier si la carte est "vivante" sur le réseau grâce à la commande PING, nous avons uniquement implanté le dialogue *Echo* et *Echo reply* au niveau ICMP.

Ainsi, nous vérifions les champs type et code, d'une commande *Echo*. Dans le cas contraire, la trame est ignorée.

Cette vérification étant faite, nous réalisons la trame réponse *Echo-reply*, grâce à la fonction **EchoReply( )**.

Enfin cette trame est envoyée sur le réseau de la même manière que précédemment.

4.3.3.4 La tâche Gestion UDP : TacheUDP ( )

Cette tâche est la plus simple car le protocole, de part son manque de sécurité, est très simple.

Nous effectuons tout de même la vérification du checksum UDP, grâce à la fonction **VerifChecksumUDP( )**. Si cette valeur correspond à celle contenue dans le champ, le sémaphore Sem5 est positionné afin de passer à un traitement de niveau supérieur (niveau 4).

4.3.3.5 La tâche Gestion TCP : TacheTCP ( )

Le protocole TCP est, de loin, le plus compliqué. Cette tâche vérifie, premièrement, le checksum TCP de la même manière que dans les tâches précédentes grâce à la fonction **VerifChecksumTCP( )**.

Si le checksum est correcte, alors grâce à la fonction **SequenceurTCP( )**, nous accédons au séquenceur de la gestion du protocole.

Nous ne décrirons pas ici le séquenceur du traitement TCP, cela a déjà été réalisé dans le chapitre présentant les différents protocoles.

A l'intérieur de ce séquenceur nous avons testé deux protocoles de plus haut niveau, c'est à dire le TELNET et l'HTTP. Pour cela, nous testons la valeur du port destination (23 : TELNET et 80 : HTTP), et le cas échéant appelons les fonctions traitant les applications correspondantes.

Pour l'instant, les applications des autres valeurs de port ne sont pas implantées, car le besoin n'en a pas été exprimé, mais si cela est nécessaire, il suffit de tester une autre valeur de port et d'appeler les fonctions correspondantes dans la fonction **SequenceurTCP( )** et de créer ces fonction de niveau 4.

4.3.3.6 La tâche Application UDP : ApplicationUDP( )

Cette tâche applicative (niveau 4) a pour but de tester la communication UDP. Pour cela, lorsqu'une trame de ce type est reçue, le champ de données est envoyé sur la liaison série RS232 de la carte vers l'Hyperterminal de l'ordinateur. Ainsi, les données contenues dans la trame apparaissent à l'écran de l'ordinateur.

4.3.3.7 La fonction Gestion TELNET : ApplicationTELNET( )

Cette fonction permet de tester la fonction TELNET. Pour cela, on ouvre une connexion TELNET sur une station distante, et le texte qui est taper sur la station distante s'affiche, grâce à cette fonction, sur l'écran de l'ordinateur.

Pour cela, lorsque la connexion est établie, pour chaque trame TELNET reçue, le champs de données est envoyer sur la liaison série RS232 vers l'Hyperterminal. On aperçoit, l'affichage des caractères au fur et à mesure de la frappe.

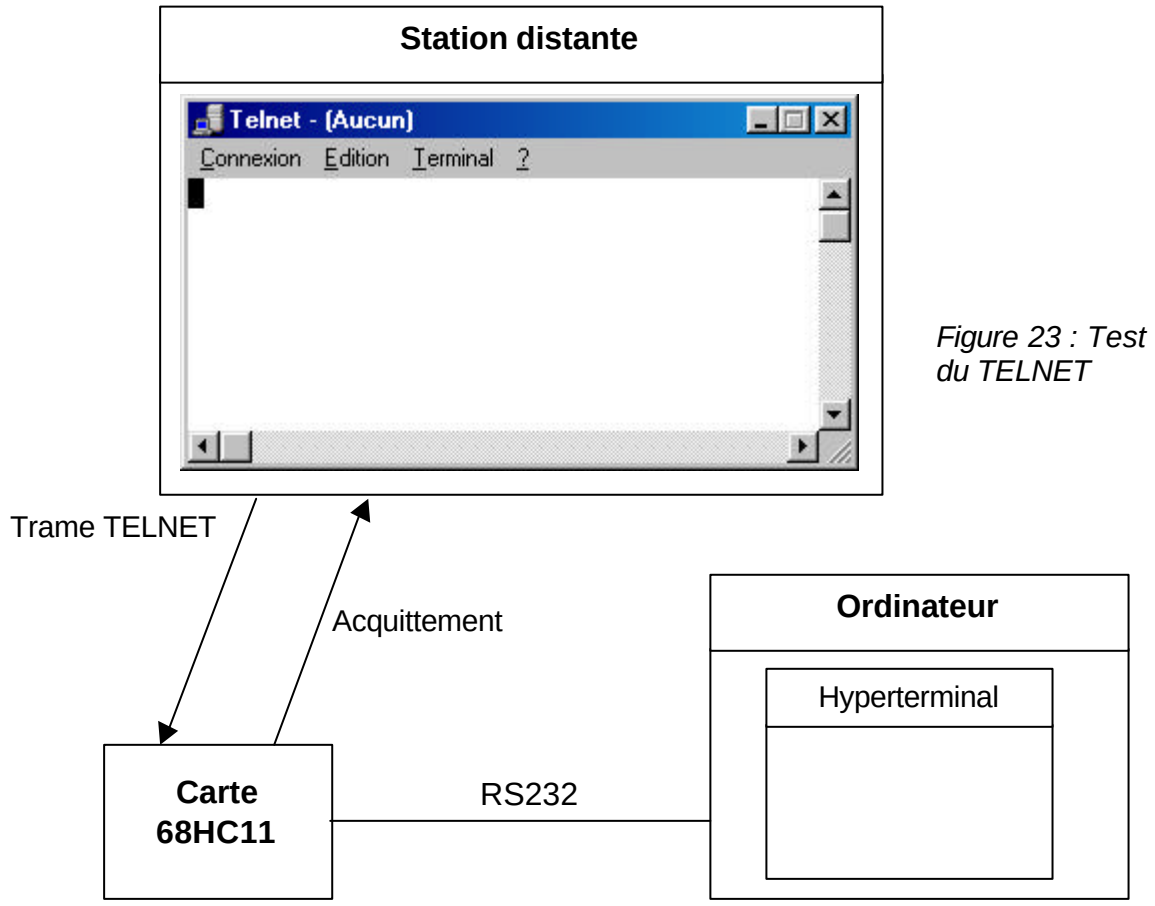


Figure 23 : Test du TELNET

4.3.3.8 La fonction Gestion HTTP : ApplicationHTTP( )

Cette fonction permet de tester le protocole HTTP. Elle permet de construire une trame HTTP, contenant, dans son champ de données, du code HTML qui a été préalablement stocké dans une chaîne de caractère sur la carte 68HC11.

A la réception d'une requête HTTP à destination de l'adresse IP de notre application, la trame construite précédemment est envoyée vers la station distante permettant l'affichage d'une page HTML sur cette dernière.

Dans ce cas, notre carte joue le rôle d'un "mini serveur" WEB. Sur la station distante, la page s'affiche correctement, mais nous constatons que le navigateur attend quelque chose. En effet, ce dernier attend la fermeture de la connexion de la part du serveur, même si c'est lui qui est à l'origine de cette même connexion. Seulement, comme nous l'avons spécifié au début de ce document, notre carte a surtout un rôle de récepteur et elle n'est pas programmée pour être à l'origine d'établissement et de fermeture de connexion.

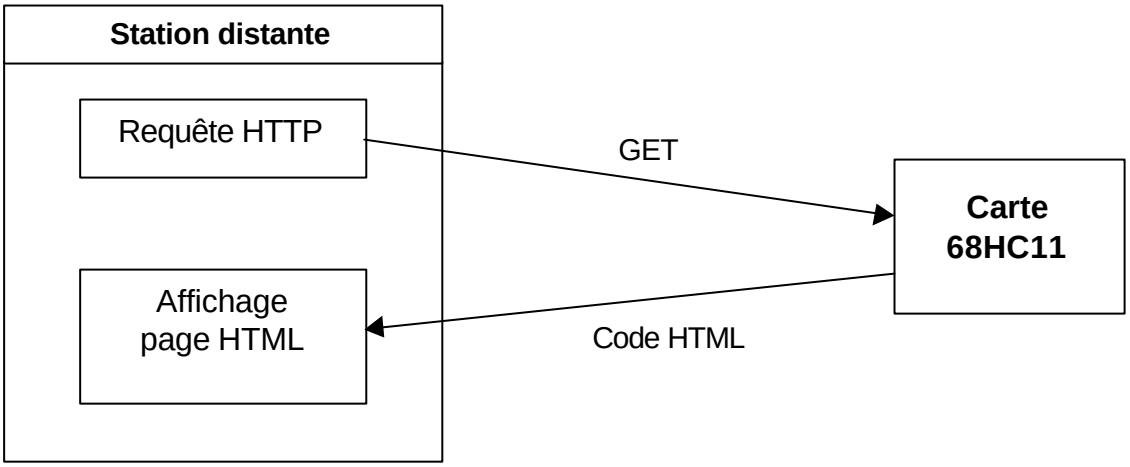


Figure 24 : test du HTTP

Attention : En ce qui concerne les applications liées au protocole TCP, nous n'avons pas pus les implanter dans des tâches gérées par l'ordonnanceur du noyau, car l'appel de l'application est réalisé à partir de la tâche **TacheTCP( )**. Une fois que le traitement applicatif est réalisé le programme doit revenir où il avait quitté **TacheTCP( )**. Cependant le noyau µC/OS II n'offre pas cette possibilité. Nous avons décidé de coder le niveau applicatif dans des fonctions traditionnelles qui permettent de revenir à l'endroit de leur appel après leur exécution.

Cela explique le fait que les flèches indiquant la relation entre la tache Gestion TCP et les deux applications, soient en pointillés et ne passent pas par le noyau temps réel sur la figure 22.

4.4 Organisation mémoire du programme

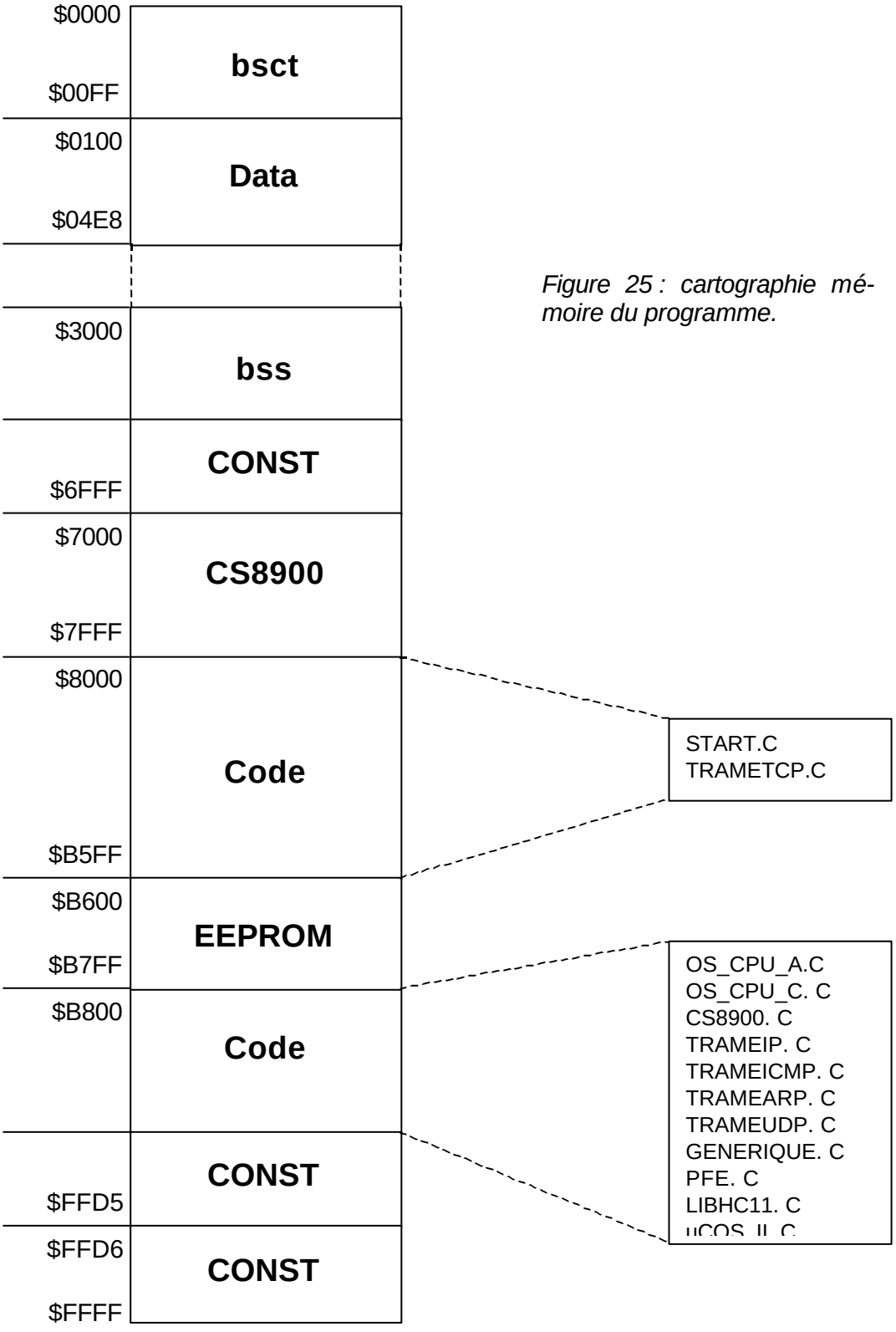


Figure 25 : cartographie mémoire du programme.

La figure 25 représente l'organisation mémoire du code et des différents types de variables. Cette cartographie est définie dans le fichier d'édition de liens *pfe.lkf*.

Nous pouvons constater qu'il y a différents types de zones :

- bsct : contient toutes les données des 256 premiers octets, cette zone de données est appelée, également, la page zéro.
- data : contient toutes les données initialisées déclarées en static.
- Bss : contient toutes les variables externes non initialisées.
- Const : contient toutes les variables initialisées.
- Code : contient tout le code composant le programme.

Lorsque nous observons le fichier *pfe.map*, nous constatons que certaines zones sont vides, comme la zone bsct, ou bien, encore, la zone data, où il n'y a qu'une seule donnée.

4.5 Répartition des fonctions

Fichier *pfe. C* :

- static void TachePrincipale(void *pdata)
- static void TacheIP(void *pdata)
- static void TacheICMP(void *pdata)
- static void TacheUDP(void *pdata)
- static void TacheTCP(void *pdata)
- static void ApplicationUDP(void *pdata)
- **void ApplicationHTTP( )**
- **void ApplicationTELNET( )**

Fichier *trameIP. C* :

- **int VerifTrameIP(char *ZoneRx)**
- **char VerifChecksumIP(char *Zone,char VerifOK)** : dans cette fonction l'argument VerifOK indique s'il s'agit simplement d'une vérification de checksum reçue ou bien du calcul d'une checksum pour remplir le champs correspondant dans la trame à émettre.
- **char VerifAdresseIP(char *ZoneRx)** : permet de vérifier si l'adresse IP de la trame reçue correspond avec celle de notre application, inscrite dans le fichier *trame.h*.
- **char VerifProtocoleIP(char *ZoneRx)** : vérifie le protocole de niveau supérieur à IP utilisé.

Fichier *trameUDP. C* :

- **char VerifChecksumUDP(char *Zone)**

Fichier *trameICMP. C* :

- **char VerifChecksumICMP(char *Zone,char VerifOK)**
- **int EchoReply(char *ZoneRx,char *ZoneTx)** : permet de fabriquer la trame IP/ICMP du type *Echo Reply* qui sera émise.

Fichier *trameARP. C* :

- **int VerifTrameARP(char *ZoneRx)** : test s'il s'agit d'une trame ARP.
- **char VerifPingARP(char *ZoneRx)** : test si la trame ARP reçue correspond à la commande PING.
- **void PingARP(char *ZoneRx,char *ZoneTx)** : permet de construire la trame ARP de réponse à la commande PING.

Fichier *trameTCP. C* :

- **char VerifChecksumTCP(char *Zone,char VerifOK)**
- **void SequenceurTCP(char *ZoneRx, char *ZoneTx)** : correspond à la machine d'état du protocole TCP.
- **void InversePort(char *ZoneTx)** : permet d'inverser les numéros de port de la trame reçue et de la trame à émettre.

Fichier *generique. C* :

Ce fichier contient des fonctions qui peuvent être appelées à différents endroits du programme.

- **int EthernetLengthCalcul(int *ZONE)** : permet le calcul de la longueur d'une trame Ethernet, sans le CRC, en octets.
- **void InverseAdresse(char *ZoneRx,char* ZoneTx)** : permet l'inversion des adresses Ethernet et IP d'une trame reçue vers une trame à émettre.

Fichier *cs8900. C* :

- **void cs8900init(void)** : permet l'initialisation de l'interface Ethernet CS8900.
- **void cs8900reset(void)** : permet la remise à zéro l'interface Ethernet CS8900.
- **int cs8900TxFrame(int lengthword, int *data)** : permet l'émission d'une trame Ethernet, sans le CRC vers l'interface Ethernet CS8900 qui l'envoie ensuite sur le réseau.
- **int TestRxPoll(void)** : permet la scrutation du registre de réception du CS8900, et stocke la trame reçue en mémoire par l'intermédiaire de la fonction **int cs8900RxFrame(int *data)**.

Fichier *vector. C* :

Ce fichier contient la table des vecteurs d'interruption, quand ce mode de fonctionnement est utilisé.

Conclusion

Ce projet s'est révélé très enrichissant. En effet, il nous a permit de découvrir et surtout de côtoyer le fonctionnement des protocoles Internet dans le détail. Cela nous a permit d'acquérir beaucoup de connaissances dans le domaine des réseaux qui viennent s'ajouter à l'enseignement traditionnel qui nous a été dispensé.

Au terme de ce projet, nous avons abouti à des résultats très intéressants, c'est-à-dire que le cahier des charges est totalement rempli. Nous sommes parvenu à implanter la couche IP, le niveau UDP, le niveau ICMP, ainsi que le niveau TCP. Il faut noter que dans le cas du protocole TCP, nous l'avons implanté uniquement en mode récepteur, c'est-à-dire que les requêtes de demandes de début et de fin de connexion proviennent de la station distante. Les essais que nous avons réalisés en transmettant des trames à la carte, grâce au logiciel **Surveyor**, nous ont montré que les réponses envoyées par la carte sont tout à fait cohérentes avec celles attendues. Nous avons, également, pu observer une bonne dynamique dans le fonctionnement de notre programme.

Nous sommes également allés au-delà des spécifications du cahier des charges. Effectivement, nous avons implanté le niveau applicatif. Par exemple, dans le cadre du protocole UDP nous avons réalisé l'affichage des données. Nous avons également traité le cas du TELNET avec le protocole TCP, ainsi que le cas du HTTP pour lequel nous avons réalisé un mini serveur WEB.

Comme nous l'avons dit au début de cet exposé, ce projet s'inscrit parfaitement dans le contexte industriel actuel et dans la tendance qui consiste à exploiter au maximum les possibilités offertes par le réseau Internet. En effet, nous avons, notamment, découvert qu'une société proposait un composant comprenant les protocoles Internet, plus d'autres particularités, qui correspond tout à fait au type d'application pour lequel nous avons travaillé.

En comparaison, notre travail est, certes, moins complet et moins performant, mais en termes de coût, le prix de revient est bien moindre pour des petites applications fonctionnant en réception.

Bibliographie

Ouvrages :

Maucourant R., C facile, Marabout, 1988

Jamsa K., Lars Klander .Ph.D, C/C++ La bible du programmeur, Eyrolles, 1999

Clot du Hecquet J.-N., Escoffier-Gentile Ph., Aide-mémoire de C, Marabout, 1990

Zhang T., Le langage C, collection Le tout en poche, Campus Press, 1999

Labrosse J.J., MicroC/OS-II The Real-Time Kernel, R&D Books, 1999

Pujolle G., Les Réseaux, Eyrolles, 1998

Documentation :

C Cross Compiler User's Guide for MC68HC11, Cosmic Software, 1996

Carte 68HC11 avec interface Ethernet, Rapport de projet de fin d'étude (1999/2000)

Documentation technique du CRISTAL CS8900

Internet :

www.enseirb.fr/~kadionik

http://www.eisti.fr

Annexes

Annexe 1 : Le 68HC11

Annexe 2 : Le CS8900

Annexe 3 : Le fichier .S19

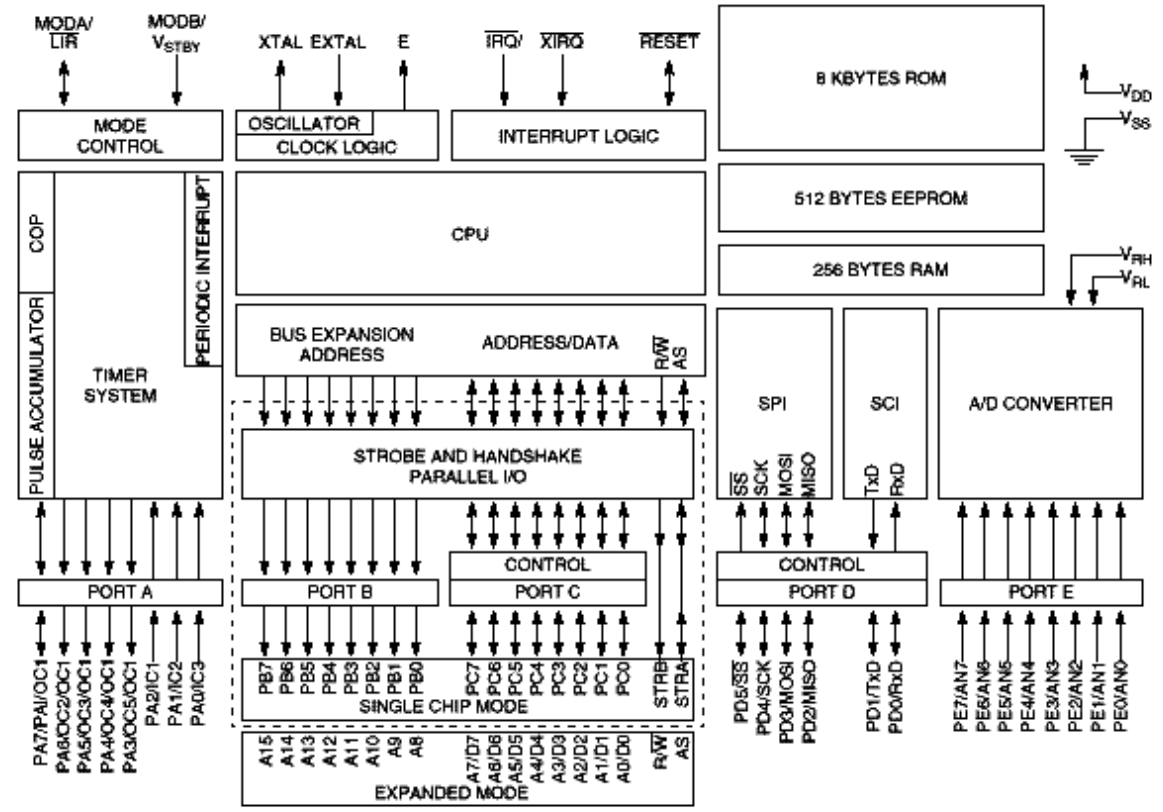
Annexe 4 : La page HTML

Annexe 5 : Le fichier d'édition des liens

Annexe 6 : Programme du projet

Annexe 1 : Le 68HC11A8

Architecture :



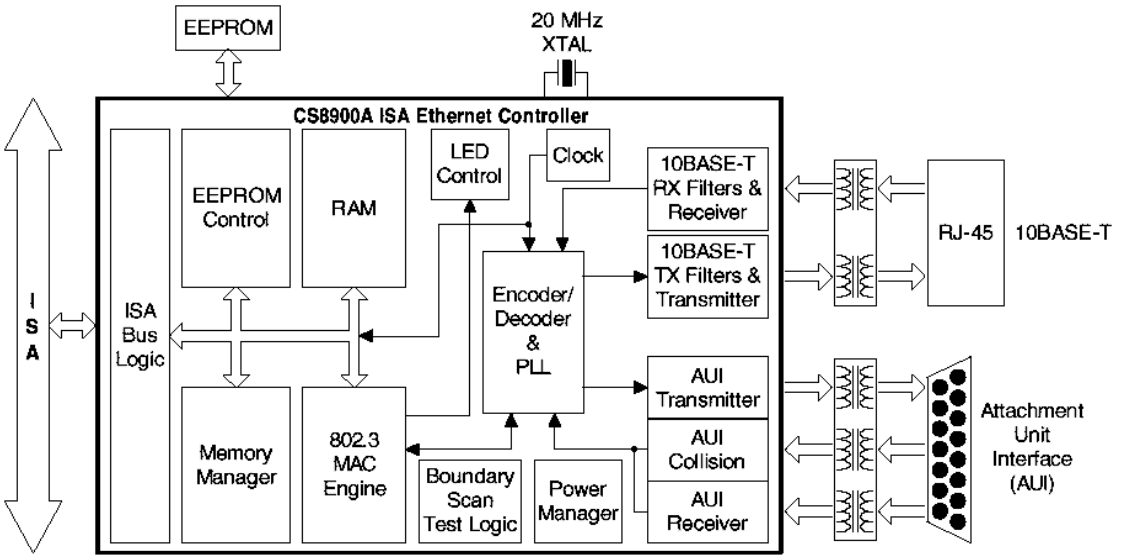
CIRCUITRY ENCLOSED BY DOTTED LINE IS EQUIVALENT TO MC68HC24.

Brochage :

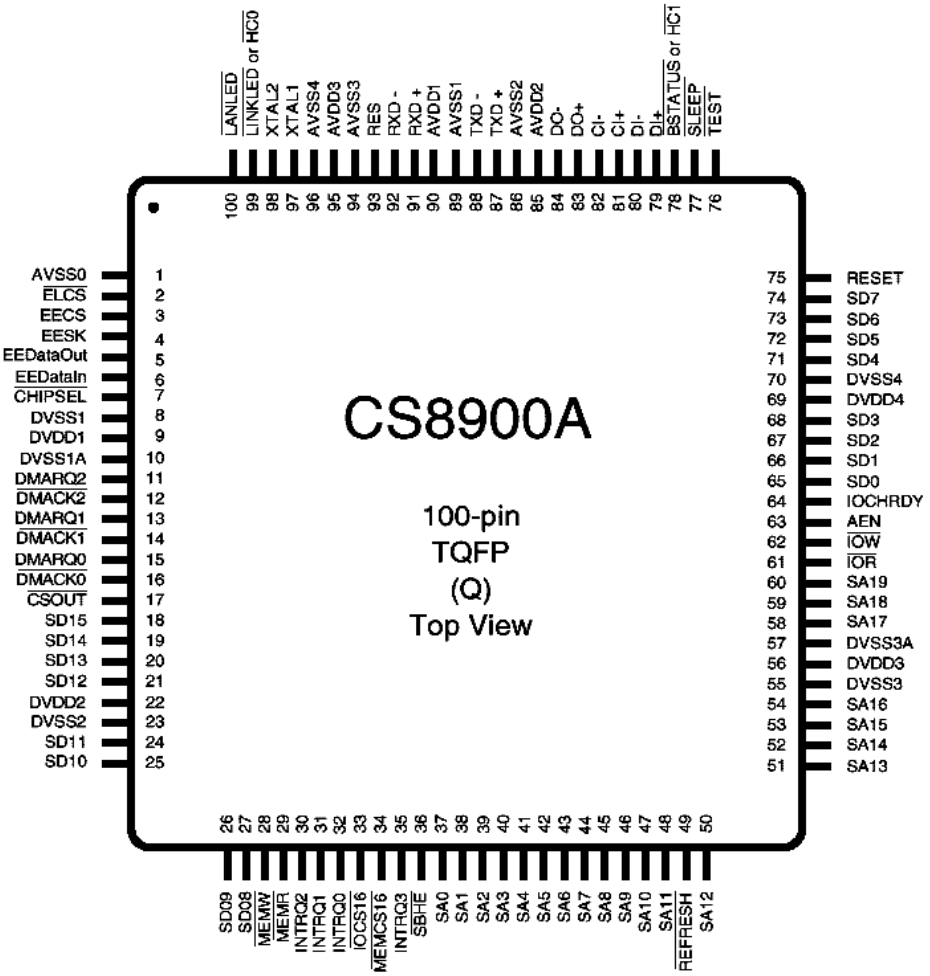
1	GND	14	PC5	27	PA7	40	PB2
2	MODB	15	PC6	28	PA6	41	PB1
3	MODA	16	PC7	29	PA5	42	PB0
4	STRA AS	17	RESET	30	PA4	43	PE0
5	E	18	XIRQ	31	PA3	44	PE4
6	STRB RW*	19	IRQ	32	PA2	45	PE1
7	EXTAL	20	RXD	33	PA1	46	PE5
8		21	TXD	34	PA0	47	PE2
9	PC0	22	PD2	35	PB7	48	PE6
10	PC1	23	PD3	36	PB6	49	PE3
11	PC2	24	PD4	37	PB5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

Annexe 2 : Le CS8900

Architecture :



Brochage :

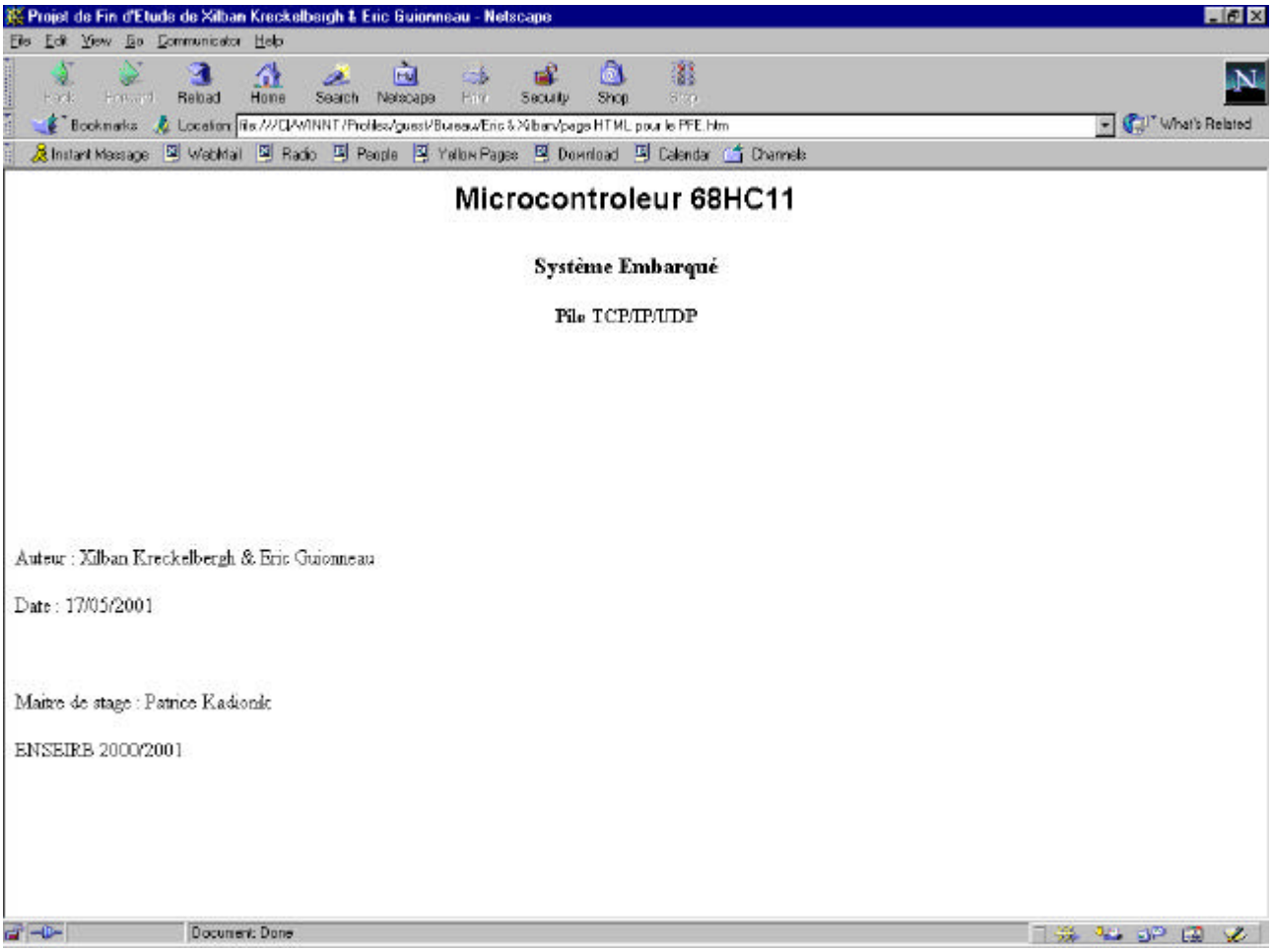


Annexe 3 : Le fichier .S19

Le fichier ci-dessous représente le flot de données qui est transmis au microcontrôleur lors de la phase de téléchargement du programme. Il ne s'agit pas de donner le fichier complet, mais sa structure.

S00A00007066652E483131E2
S12397F7485454502F312E3020323030204F4B0D0A436F6E74656E742D54797065203A20A9
S1239817746578742F48544D4C0D0A0D0A3C68746D6C3E3C686561643E3C7469746C653EFF
S123983750726F6A65742064652046696E20642745747564652064652058696C62616E204A
S12398574B7265636B656C626572676820262045726963204775696F6E6E6561753C2F7491
S123987769746C653E3C6D65746120687474702D65717569763D22436F6E74656E742D540C
S1239897797065222063 6F6E74656E743D22746578742F68746D6C3B2063686172736574D5
S12398B73D69736F2D383835392D31223E3C2F686561643E3C626F6479206267636F6C6F81
S12398D7723D22234646464646223E3C7020616C69676E3D2263656E746572223E3C66BC
S12398F76F6E7420666163653D2256657264616E612C20417269616C2C2048656C76657444
S12399176963612C2073616E732D736572696622073697A653D222B32223E3C623E4D690D
S123993763726F636F6E74726F6C657572202020363848433131 3C2F623E3C2F666F6E7493
S12399573E3C2F703E3C7020616C69676E3D2263656E746572223E3C666F6E742066616341
S1239977653D2254696D6573204E657720526F6D616E2C2054696D65732C207365726966EC
S1239997223E3C623E3C666F6E742073697A653D222B31223E537973742665677261766534
S12399B73B6D65202020456D626172717526656163757465 3B3C2F666F6E743E3C2F623E0F
S12399D73C2F666F6E743E3C2F703E3C7020616C69676E3D2263656E746572223E3C666FCB
S12399F76E7420666163653D2254696D6573204E657720526F6D616E2C2054696D65732C79
S1239A1720736572696223E3C623E50696C65205443502F49502F5544503C2F623E3C2FCF
S1239A37666F6E743E3C2F703E3C7020616C69676E3D226C656674223E266E6273703B3C6C
S1239A572F703E3C7020616C69676E3D226C656674223E266E6273703B3C2F703E3C7020D4
S1239A77616C69676E3D226C656674223E266E6273703B3C2F703E3C7020616C69676E3D15
S1239A97226C656674223E266E6273703B3C2F703E3C7020616C69676E3D226C656674224E
.
.
.
.
.
.
.
.
.
S121E5408F18EB0089008FED00323335388F186E013CEE02CDAC002603CDA302383989
S123FFD6000000000000000000000000000000000000B840000000000000000000000000F
S10DFFF6B80C0000000000008000B9
S903FFFFFE

Annexe 4 : La page HTML



Annexe 5 : Le fichier d'édition des liens *pfe.lkf*

```
#################################################################################
#                    uC/OS-II, The Real-Time Kernel
#
#                          Motorola 68HC11
#
#
# Filename  : pfe.lkf
#
# Note(s)   : 1) This linker command file assumes the V4.x compiler tool chain
#             by COSMIC SOFTWARE.
#################################################################################
+seg .bsct   -b 0x0000 -m 256
+seg .data   -b 0x0100 -m 1000
+seg .bss    -b 0x3000 -n .ram
+seg .const  -a .ram
+seg .text   -b 0x8000 -m 13824 -n .text
START.O
TRAMETCP.O
+seg .text   -b 0xB800 -n .text
+seg .const  -a .text
OS_CPU_A.O
OS_CPU_C.O
CS8900.O
TRAMEIP.O
TRAMEICMP.O
TRAMEARP.O
TRAMEUDP.O
GENERIQUE.O
PFE.O
LIBHC11.O
uCOS_II.O
c:/logiciel/68hc11/cx32/lib/libf.h11          # C library
c:/logiciel/68hc11/cx32/lib/libi.h11          # C library
c:/logiciel/68hc11/cx32/lib/libm.h11          # machine library
#+seg .eeprom -b 0xB600 -m 512 -n .eeprom
+seg .const  -b 0xFFD6 -m42
VECTORS.O
```