Mise en place d'un calculateur

U.V T.X : coupe E=M6

Automne 2005

Projet réalisé par :
Johan MATHE

Dernière version : 21 Janvier 2006

Remerciements

Je tiens à remercier Messieurs Christophe Forgez, Emmanuel Foulon, et Vincent Lafranchi qui m'ont aidé tout au long du semestre à mener à bien ce travail. Je tiens à remercier tout particulièrement M. Stéphane Bonnet qui m'a aussi beaucoup apporté lors de la conception de mes prototypes.

Objectifs

L'objectif de cette U.V TX est de mette en place un calculateur fonctionnel pour la coupe E=M6. Notre choix a été porté sur le PC104 qui avait déjà été acheté par les anciennes sessions de la coupe E=M6. Nous expliquerons dans ce rapport comment fonctionne le PC104 et comment on l'utilise. Le système d'exploitation choisi a été linux car celui ci contenait déjà tous les drivers nécessaires dans son noyau pour faire fonctionner le pc104. De plus rtai a été installé sur la machine en vue de pouvoir faire du temps réel. Le tout a été mis dans une boite tupperware en vue de protéger les entrées sorties. Enfin, un driver pour bus I2C a été construit et sera documenté ici même. Ce dernier utilise les drivers déjà présents dans le noyau linux pour fonctionner.

Table des matières

Table des figures

Chapitre 1

Mise en place du matériel

Nous verrons tout au long de ce chapitre les raisons des choix technologiques, et le fonctionnement détaillé des modules du calculateur. Tout d'abord, nous nous attarderons sur le fonctionnement du PC104 lui même, de la carte d'entrées/sorties parallèles, et enfin de la carte de driver i2c.

1.1 Le PC104

Le PC104[1] est tout d'abord un pc d'architecture x86 (comme les PC de salon). La seule différence vient du standard de sa taille et de son connecteur. De plus, ces machines sont plus robustes, et mieux testées. La seule chose à reprocher sur un PC de ce type est sa puissance (il s'agit en l'occurence d'un modèle 486 DX 66Mhz) ce qui est assez vieux. Ce pc 104 est pour l'instant équipé d'une carte processeur (avec ports série, port parallèle, bus IDE etc.). De plus, une carte vidéo est montée sur la machine, permettant une utilisation confortable de la machine. Un port clavier est aussi disponible. Le PC104 a été monté dans un "tupperware" en vue de solidifier le tout. La documentation précise du PC104 est fournie en annexes.

1.2 La carte d'entrée/sorties

Le PC104 est équipé d'une carte de 32 entrées/sorties parallèles qui permettront de faire de la commande tout ou rien d'actionneurs au sein du robot. La carte a été testée et est très fragile au niveau des courants. Elle est à manier avec grandes précautions pour ce qui est des courants. L'utilisation de cette carte est très simple et décrite dans la partie logiciel.

[1] PC104 vient de picots sur le connecteur qui est de 104

1.3 La mise en place du port série

Une grande question lors de la réalisation de ce projet à été celle de la communication avec l'extérieur : pour cela, il était impossible de mettre en place une solution réseau classique. La solution retenue a donc été de mettre en place une liaison réseau au travers d'une liaison série pour qu'une machine de type WINDOWS ou UNIX puisse communiquer aisément grâce à un simple cordon RS232. Les tests effectués ont montré de réels résultats exploitables avec une vitesse de 10 $ko.s^{-1}$.

1.4 La carte driver I2C

La nécessité de l'utilisation d'un bus de terrain au sein de la coupe E=m6 devenait importante. En effet, pour assurer en terme de contraintes de sécurité de communication, d'économie de fils, etc. un bus de terrain comme l'i2c permet en robotique mobile de gagner énormément.

1.4.1 Le protocole I2C

I^2C (pour Inter Integrated Circuit Bus) est le nom du bus historique, développé par Philips pour les applications de domotique et d'électronique domestique au début des années 1980, notamment pour permettre de relier facilement à un microprocesseur les différents circuits dune télévision moderne.

Caractéristiques

Le bus I2C permet de faire communiquer entre eux des composants électroniques très divers grace à seulement trois fils : Un signal de donnée (SDA), un signal d'horloge (SCL), et un signal de référence électrique (Masse).

Ceci permet de réaliser des équipements ayants des fonctionnalités très puissantes (En apportant toute la puissance des systèmes microprogrammés) et conservant un circuit imprimé très simple, par rapport un schéma classique (8 bits de données, 16 bits d'adresse + les bits de contrôle).

Les données sont transmises en série à 100Kbits/s en mode standard et jusqu'à 400Kbits/s en mode rapide. Ce qui ouvre la porte de cette technologie à toutes les applications où la vitesse n'est pas primordiales.

De nombreux fabricants ayant adopté le système, la variété des circuits disponibles diposant d'un port I2C est énorme : Ports d'E/S bidirectionnels, Convertisseurs A/N et N/A, Memoires (RAM, EPREM, EEPROM, etc...), Circuits Audio (Egaliseur, Contrôle de volume, ...) et autre drivers (LED , LCD , ...)

Le nombre de composants qu'il est ainsi possible de relier est essentiellement limité par la charge capacitive des lignes SDA et SCL : 400 pF .

Principe

Afin de d'éviter les conflits électriques les Entrées/Sorties SDA et SCL sont de type "Collecteur Ouvert"

Voici un schéma de principe :

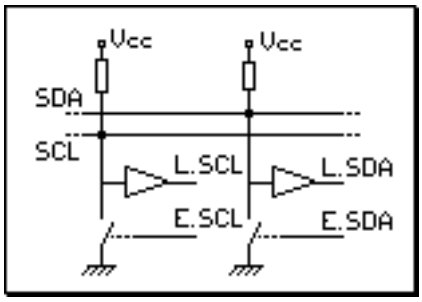

FIG. 1.1 – Structure d'entrée/sorties d'un module I2C

Le protocole I2C définit la succession des états logiques possibles sur SDA et SCL, et la façon dont doivent réagir les circuits en cas de conflits.

La prise de controle du bus

La prise controle du bus :

Pour prendre le controle du bus, il faut que celui-ci soit au repos (SDA et SCL à '1'). Pour transmettre des données sur le bus, il faut donc surveiller deux conditions particulières :

- La condition de départ. (SDA passe à '0' alors que SCL reste à '1')
- La condition d'arrèt. (SDA passe à '1' alors que SCL reste à '1')

Lorsqu'un circuit, aprés avoir vérifié que le bus est libre, prend le controle de celui-ci, il en devient le maître. C'est lui qui génère le signal d'horloge.

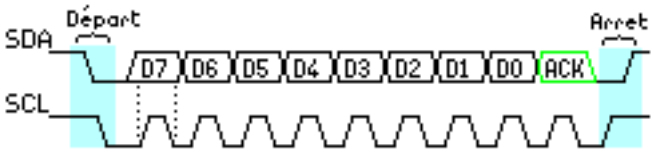

FIG. 1.2 – Exemple de condition de départ et d'arrêt

La transmission d'un octet

Après avoir imposé la condition de départ, le maître applique sur SDA le bit de poid fort D7. Il valide ensuite la donnée en appliquant pendant un instant un niveau '1' sur la ligne SCL. Lorsque SCL revient à '0', il recommence l'opération jusqu'à ce que l'octet complet soit transmis. Il envoie alors un bit ACK à '1' tout en scrutant l'état réel de SDA. L'escave doit alors imposer un niveau '0' pour signaler au maître que la transmission s'est effectuée correctement. Les sorties de chacuns étant à collecteurs ouverts, le maître voie le '0' et peut alors passer à la suite.

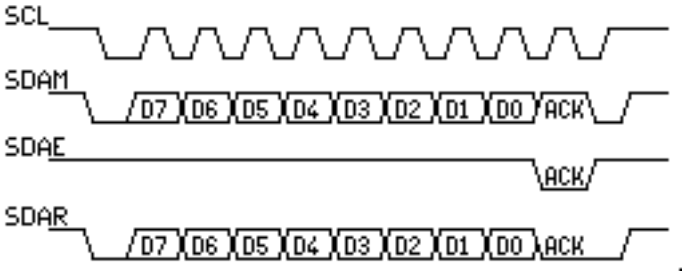

FIG. 1.3 – Exemple de transmission réussie.

Dans cet exemple :

- SCL : Horloge imposée par le maître.
- SDAM : Niveaux de SDA imposés par le maître.
- SDAE : Niveaux de SDA imposés par l'esclave.
- SDAR : Niveaux de SDA réels résultants.

1.4.2 La transmission de l'adresse

Le nombre de composants qu'il est possible de connecter sur un bus I2C étant largement supérieur à deux, il est nécessaire de définir pour chacun une adresse unique. L'adresse d'un circuit, codée sur sept bits, est défini d'une part par son type et d'autre part par l'état appliqué à un certain nombre de ces broches (Voir §9). Cette adresse est transmise sous la forme d'un octet au format particulier.

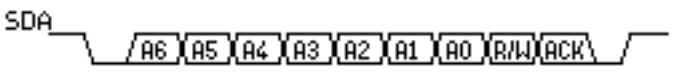

FIG. 1.4 – Exemple d'octet d'adresse.

On remarque ici que les bits D7 à D1 représentent les adresse A6 à A0, et que le bit D0 et remplacé par le bit de R/W qui permet au maître de signaler s'il veut lire ou écrire une donnée. Le bit d'acquitement ACK fonctionne comme pour une donnée, ceci permet au maître de vérifier si l'esclave est disponible.

Note 1: Cas perticulier des mémoires :

L'espace adressable d'un circuit de mémoire étant sensiblement plus grand que la plupart des autres types de circuits, l'adresse d'une information y est codée sur deux octets ou plus. Le premier représente toujours l'adresse du circuit, et les suivants l'adresse interne de la mémoire.

Note 2: Les adresses réservées.

Les adresses 00000XXX et 111111XX sont réservés à des modes de fonctionnement particuliers.

Ecriture d'une donnée

L'écriture d'une donnée par le maître ne pose pas de problème particulier :

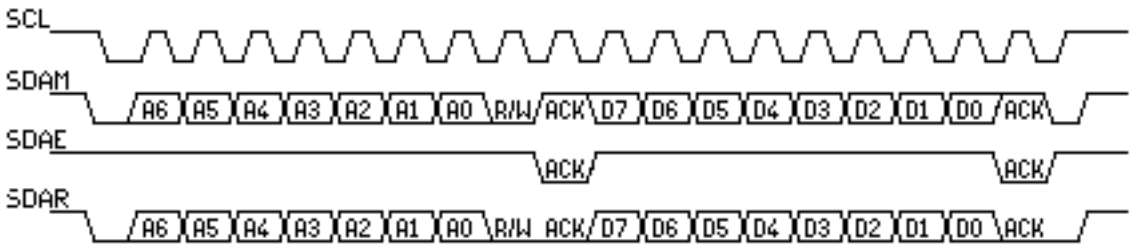

Fig. 1.5 – Exemple d'écriture d'une donnée

Note : Cas perticulier d'utilisation d'ACK :

L'écriture d'un octet dans certains composants (Mémoires, microcontrôleur, ...) peut prendre un certain temps. Il est donc possible que le maître soit obligé d'attendre l'acquitement ACK avant de passer à la suite.

Lecture d'une donnée

La lecture d'une donnée par le maître se caractèrise par l'utilisation spéciale qui faite du bit ACK. Après la lecture d'un octet, le maître positionne ACK à '0' s'il veut lire la donnée suivante (cas d'une mémoire par exemple) ou à '1' la cas échéant. Il envoie alors la condition d'arret.

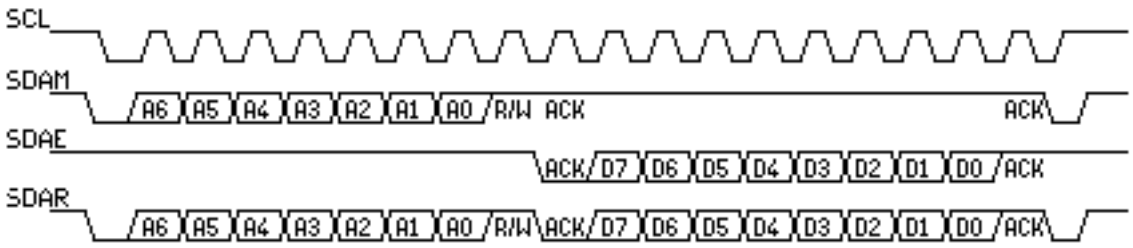

FIG. 1.6 – Exemple de lecture d'une donnée.

1.4.3 La gestion des conflits

Mise en situation

La structure même du bus I2C a été conçu pour pouvoir y accueillir plusieurs maîtres. Se pose alors le problème commun à tout les réseaux utilisant un canal de communication unique : la prise de parole. En effet, chaque maître pouvant prendre possession du bus dès que celui-ci est libre, il existe la possibilité de que deux maîtres prennent la parole en même temps. Si cela ne pose pas de problème sur le plan électrique grace l'utilisation de collecteurs ouverts, il faut pouvoir détecter cet état de fait pour éviter la corruption des données transmises.

Principe

Comme nous l'avons vu précedement, pour prendre le contrôle du bus, un maître potentiel doit d'abord vérifier que celui-ci soit libre, et qu'une condition d'arrèt ait bien été envoyée depuis au moins $4,7\mu s$. Mais il reste la possibilité que plusieurs maîtres prennent le contrôle du bus simultanement.

Chaque circuit vérifie en permanence l'état des lignes SDA et SCL, y compris lorsqu'ils sont eux même en train d'envoyer des données. On distingue alors plusieurs cas :

1. Les différents maîtres envoyent les mêmes données au même moment : Les données ne sont pas corrompues, la transmission s'effectue normalement, comme si un seul maître avait parlé. Ce cas est rare.

2. Un maître impose un '0' sur le bus : Il relira forcément '0' et continuera à transmettre. Il ne peut pas alors détecter un eventuel conflit.

3. Un maitre cherche à appliquer un '1' sur le bus : Si il ne relit pas un niveau '1', c'est qu'un autre maître a pris la parole en même temps. Le premier perd alors immédiatement le controle du bus, pour ne pas perturber la transmission du second. Il continue néanmoins à lire les données au cas celles-ci lui auraient été destinées.

Exemple

Soit le chronogramme suivant :

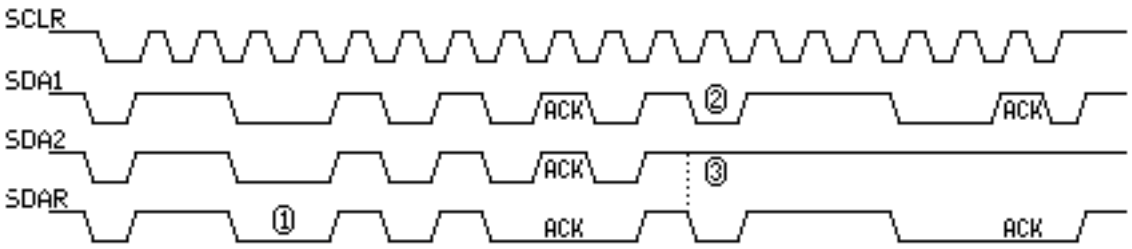

Fig. 1.7 – Chronogramme illustrant le conflit I2C.

Dans cet exemple :

- SCLR : Horloge résultante.
- SDA1 : Niveaux de SDA imposés par le maître n°1.
- SDA2 : Niveaux de SDA imposés par le maître n°2.
- SDAR : Niveaux de SDA réels résultants lus par les deux maîtres.

Analyse : Le premier octet est transmis normalement car les deux maîtres imposent les même données. (Cas n°1). Le bit ACK est mis à '0' par l'esclave. Lors du deuxième octet, le maître n°2 cherche à imposer un '1' (SDA2) , mais relit un '0' (SDAR), il perd alors le contrôle du bus et devient esclave (Cas n°3) . Il reprendra le contrôle du bus, lorsque celui-ci sera de nouveau libre.

Le maître n°1 ne voit pas le conflit et continue à transmettre normalement. (Cas n°2)

Au total, l'esclave à reçu les données du maître n°1 sans erreurs et le conflit est passé inaperçu.

1.4.4 Les adresses réservées

Les addresses 0000 0xxx ne sont pas utilisées pour l'adressage de composants. Ils ont été résevés par Phillips pour effectuer certaines fonctions spéciales.

Adresse d'appel general

Adresse :

```
0000 0000
```

Après l'émission d'un appel général, les circuits ayant la capacité de traiter ce genre d'appel émettent un acquitement.

Le deuxième octet permet definir le contenu de l'appel :

`0000 0110`

RESET. Remet tout les registres de circuits connectés dans leur état initial (Mise sous tension). Les circuits qui le permettent rechargent leur adresse d'esclave.

`0000 0010`

Les circuits qui le permettent rechargent leur adresse d'esclave.

`0000 0100`

Les circuits définissant leur adresse de façon matériel reinitialisent leur adresse d'esclave.

`0000 0000`

Interdit

`xxxx xxx1`

Cette commande joue le rôle d'interruption. xxxx xxx peut être l'adresse du circuit qui a généré l'interruption.

Octet de Start

Adresse :

`0000 0001`

Cet octet est utilisé pour synchroniser les périphériques lents avec les périphériques rapides.

Debut d'adressage CBus

Adresse :

`0000 001x`

L'emission de cet octet permet de rendre sourd tout les circuits I2C présent sur le bus. A partir de ce momont, on peut transmettre ce que l'on desire sur le bus, en utilisant par exemple un autre protocole. Le bus repasse en mode normal lors de la réception d'une condition d'arrèt.

Autre

Adresses :

`0000 0110 à 0000 1111`

Ces octets ne sont pas définits et sont ignoré par les circuits I2C. Il peuvent être utilisé pour débugger un reseau multimaster.

1.4.5 Fonctionnement global de la carte

Le fonctionnement global de la carte est le suivant : un composant décode l'adresse du bus ISA et quand l'adresse de base est détectée (0x330 dans notre cas) un canal CE (*Chip Enable*) est activé ce qui commande la mise en marche du composant permettant de piloter le bus I2C. La sortie I2C est ensuite reliée aux deux pics 16F819 qui permettent la conversion en signaux PPM (*Pulse Position Modulation*) qui commandent les servos.

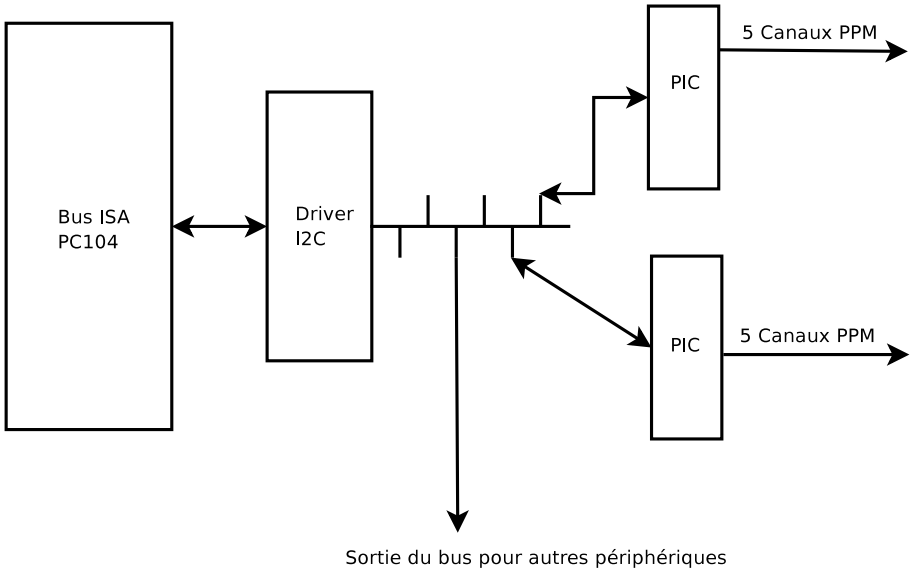

FIG. 1.8 – Schéma explicatif fu fonctionnement de la carte

1.4.6 Le driver I2C : le PCA9564

Ce composant fourni par phillips permet de convertir un bus parallèle en bus I2C. Ce composant est assez standard et remplace avantageusement l'ancienne version de ce type de composant aussi développée par phillips (le PCA85*). Celui ci ne nécessite pas d'horloge externe pour fonctionner. La documentation est fournie en annexe.

1.4.7 Le PAL22V10

Ce composant est un composant programmable qui permet de décoder l'adresse de commande du composant demandée par le BUS ISA. Ce dernier a été programmé en VHDL pour sortir la valeur de commande du PCA9564. Une optimisation future serait intéressante car celui ci consomme beaucoup (environs 200 mA)

1.4.8 Les PICS 16f819 : drivers I2C/PPM

Ces composants de chez microchip sont des microcontrolleurs ayant la propriété d'être orienté I2C. En effet, ils ont été choisis parce qu'ils décodent facilement le bus I2C d'une facon matérielle (ce qui est beaucoup plus rapide). Ces composants permettent de commander jusqu'à 5 servomoteurs[2] de modélisme en commande PPM.

Le fonctionnement

Il s'agit d'un driver permettant de faire de la commande i2c de quelque périphérique quel qu'il soit tant qu'il est commandé par des signaux PPM (Pulse Position Modulation - généralement des servomoteurs).

Particularités

Ce module permet de commander jusqu'à 6 périphériques simultanément. Son grand avantage vient surtout de sa précision (environ 5000 pas). En effet, certaines applications nécessitent une grande précision de commande pour ce qui est des servomoteurs (asservissement...). Les modes de fonctionnement

Le driver a trois modes de fonctionnement. Ceux ci sont spécifiés par l'intermédiaire de l'en tête de trame/

- Mode de *transmission de données numéro 1* :
  Le premier mode de fonctionnement est le suivant : l'en tête est composé du numéro du servo à piloter (entre 0 et 5), l'octet de poids faible et l'octet de poids fort.

FIG. 1.9 – Trame mode 1

- Mode de *transmission de données numéro 2* :

  Le deuxiememe mode de transmission consiste à passer tout d'abord l'en tête (06) puis les informations relatives aux 6 servos une par une : on commence par transmettre l'octet de poids faible du périphérique numéro 1, puis celui de poids fort, et on continue.

FIG. 1.10 – Trame mode 2

[2]micro-actionneurs

1.5 Brochage et détails quant à la carte d'E/S

Sur cette carte, on peut trouver deux connecteurs sortant le bus I2C, 10 canaux sortant des signaux PPM. Les adresses I2C des canaux de commande sont, pour les canaux de 1 à 5 0x08 et pour les canaux de 6 à 10 0x09. Se référer au schéma en annexes et au scéma d'implantation des broches pour voir le brochage des composants.
Il faut alimenter les servomoteurs sur un circuit d'alimentation secondaire. Le choix d'alimentation externe a été fait délibérément pour ne pas avoir de boucles de courant dans le calculateur.

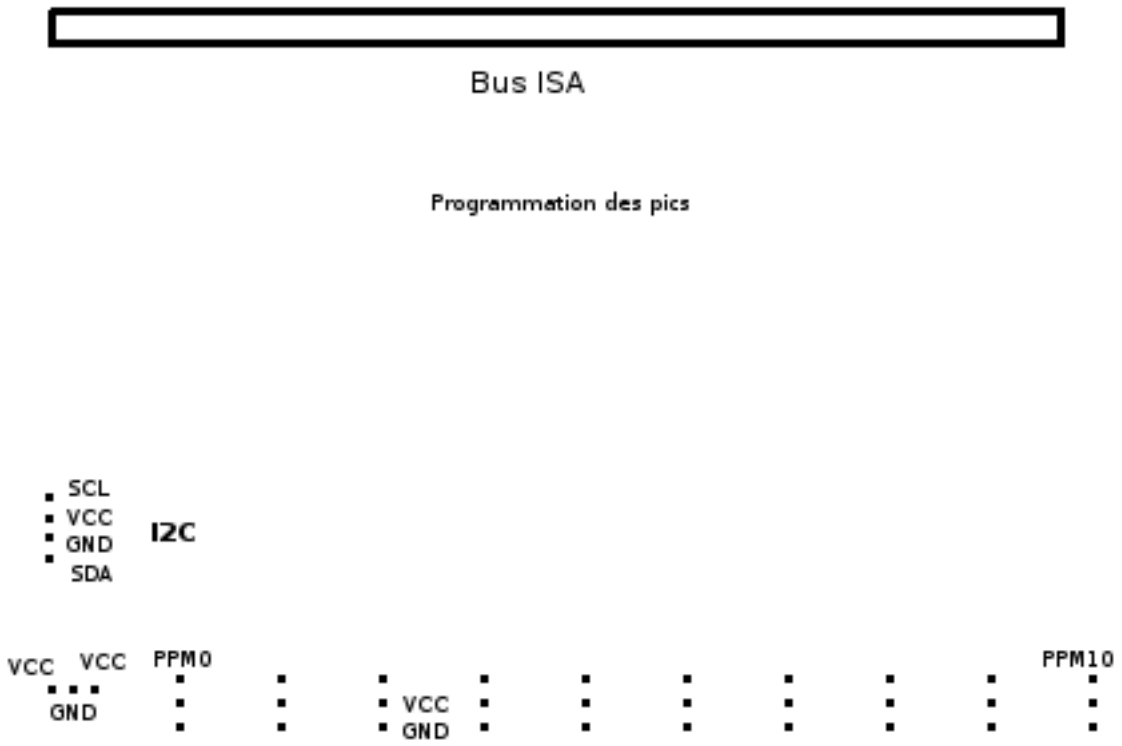

Fig. 1.11 – Schema d'implantation des broches

Chapitre 2

Mise en place du logiciel

Nous allons maintenant nous attarder sur la facon dont le logiciel a été mis en place.

2.1 Le système d'exploitation : Linux

A la base, le système d'exploitation MSDOS était installé sur le PC104. Deux choix se présentaient à nous : utiliser MSDOS qui commencait à devenir obsolete, mais adapté à l'architecture, ou bien utiliser et optimiser un Linux adapté au 486. C'est la deuxieme solution qui a été choisie.

2.1.1 Buildroot

Buildroot[1] est le logiciel qui a été utilisé pour générer le linux présent au sein de la machine. En effet, basé sur une microlibC et sur busybox, buildroot permet d'obtenir un système minimal et fonctionnel. Celui ci génère un fichier image qu'il suffit de copier directement sur le disque dur. Pour monter ce fichier :

```
mount -t ext2 -o loop img_file.ext2 /foo_path
```

Il faut créer une partition de swap et une partition principale grace à fdisk :

```
fdisk /dev/hdX
```

Où X est la lettre du lecteur.

Il faut ensuite lancer lilo sur une autre machine pour gérer le démarrage :

```
lilo -b /dev/hdb -r /path_to_chroot -v
```

[1] http://buildroot.uclibc.org/

2.2 Le noyau

Pour ce qui est du noyau, la version 2.6.9 a été choisie car ce dernier était entièrement compatible avec RTAI, le système temps réel. Les paramètres du noyau sont définis en annexe.

2.3 Le temps réel

Pour ce qui est du temps-réel, c'est RTAI qui a été choisi et implémenté dans le noyau. En l'occurence, il n'est pas nécessaire de l'utiliser. Pour l'utiliser il faudra compiler des modules qui seront eux memes les applications.

2.4 La liaison série

La liaison série a été utilisée sur le PC104 en vue de gérer une couche réseau. Grâce au démon ppp il est aisé d'émuler une couche réseau :

```
pppd /dev/ttyS1 115200 10.0.0.1:10.0.0.2 local
netmask 255.255.255.0 persist passive noauth -detach asyncmap 0
```

En effet, le périphérique série est disponible sur le port ttyS1. Il suffit de faire la même chose de l'autre côté sur une machine unix pour avoir deux interfaces réseau de chaque côté entièrement exploitables.

2.5 Chaine de compilation

Une chaine de compilation a été mise en place sur le PC104. En effet, un gcc est présent et permet de compiler les application in-situ. Le processus est relativement lent mais peut dépanner et éviter d'avoir à faire de la cross compilation. Il s'agit de Gcc 3.4 et ce dernier est présent sur buildroot[2].

2.6 Le driver I2C

Pour utiliser le driver I2C, c'est très simple :

2.6.1 Initialisation

Pour initialiser le driver les étapes suivantes sont nécessaires :

1. Charger les modules :

[2]La documentation présente sur le site web ne nécessite pas ici de détailler son utilisation

```
# modprobe i2c-pca-isa
# modprobe i2c-dev
```

D'expérience, la compilation en dur dans un noyau monolithique de ces drivers n'est pas très fructueuse et fonctionne très mal. En effet, au démarrage, le système plante. Il faut aussi créer la device qui va bien dans le repertoire /dev/. C'est ce périphérique qui sera ouvert en mode caracère par les applications.

2. Dans le programme lui même, initialiser un descripteur de fichier correspondant au périphérique :

```
/*
 * Fichier d'exploitaiton du bus i2c
 *
 *
 */
#include "/usr/src/modules/i2c/kernel/i2c.h"
#include "/usr/src/modules/i2c/kernel/i2c-dev.h"

#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdarg.h>
#include <sys/types.h>

#include <errno.h>


int open_i2c(char *dev, int addr)
{
               int fd;

               if ((fd = open(dev, O_RDWR)) >= 0)
                {
                       printf("Ouverture du port i2c ok\n");
                }
               else
               {
                  fprintf(stderr, "Impossible d'ouvrir %s : %s.\n", dev,
                      strerror(errno));
               }

               /*Paramétrage de l'adresse de l'esclave.... */
                if (ioctl(fd,I2C_SLAVE,addr) < 0) {
```

```
34                printf("Impossible de paramétrer l'adresse de l'esclave !\n"
                      );
35                return(-1);
36                 }
37
38  return(fd);
39
40 }
```

3. Il est ensuite facile de manipuler les données tout simplement en allant écrire ou lire des données avec les fonctions d'entrées/sorties standard unix.

2.6.2 Manipulation des données

Des extraits de la page de manuel seront parlants :

NOM

read - Lire le contenu d'un fichier.

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
```

DESCRIPTION

read lit jusqu'à *count* octets depuis le descripteur de fichier *fd* dans le buffer pointé par *buf*.

Si *count* vaut zéro, *read* renvoie zéro et n'a pas d'autres effets. Si count est supérieur à SSIZE_MAX, le résultat est indéfini.

VALEUR RENVOYÉE

read renvoie -1 s'il échoue, auquel cas *errno* contient le code d'erreur, et la position de la tête de lecture est indéfinie. Sinon, read renvoie le nombre d'octets lus (0 en fin de fichier), et avance la tête de lecture de ce nombre. Le fait que le nombre renvoyé soit plus petit que le nombre demandé n'est pas une erreur. Ceci se produit à la fin du fichier, ou si on lit depuis un tube ou un terminal, ou encore si read a été interrompu par un signal.

NOM

write - Écrire dans un descripteur de fichier.

SYNOPSIS

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

DESCRIPTION

write écrit jusqu'à *count* octets dans le fichier associé au descripteur *fd* depuis le buffer pointé par *buf*. POSIX réclame qu'une lecture avec read() effectuée après le retour d'une écriture avec write(), renvoie les nouvelles données. Notez que tous les systèmes de fichiers ne sont pas compatibles avec POSIX.

VALEUR RENVOYÉE

write renvoie le nombre d'octets écrits (0 signifiant aucune écriture), ou -1 s'il échoue, auquel cas errno con- tient le code d'erreur. Si count vaut zéro, et si le descripteur est associé

à un fichier normal, 0 sera renvoyé sans effets de bord. Pour un fichier spécial, les résultats ne sont pas portables.

2.6.3 Exemple

Pour manipuler des données, l'exemple suivant sera parlant : Cet exemple envoie une trame composée de 3 octets sur le bus I2C permettant d'adresser le PIC d'adresse 0x08 et d'envoyer la valeur 2560 (10×256) au servo numéro 3.

```
1  /*
2   * Fichier d'exploitaiton du bus i2c
3   *
4   *
5   */
6  #include "/usr/src/modules/i2c/kernel/i2c.h"
7  #include "/usr/src/modules/i2c/kernel/i2c-dev.h"
8
9  #include <sys/stat.h>
10 #include <fcntl.h>
11 #include <stdio.h>
12 #include <unistd.h>
13 #include <stdarg.h>
14 #include <sys/types.h>
15
16 #include <errno.h>
17
18
19 int open_i2c(char *dev, int addr)
20 {
21                 int fd;
22
23                 if ((fd = open(dev, O_RDWR)) >= 0)
24                  {
25                         printf("Ouverture du port i2c ok\n");
26                  }
27                 else
28                 {
29                    fprintf(stderr, "Impossible d'ouvrir %s : %s.\n", dev, strerror
                       (errno));
30                 }
31
32                 /*Paramétrage de l'adresse de l'esclave.... */
33                  if (ioctl(fd,I2C_SLAVE,addr) < 0) {
34                 printf("Impossible de paramétrer l'adresse de l'esclave !\n");
35                 return(-1);
```

```
36                 }
37
38  return(fd);
39
40 }
41
42
43 int main(){
44
45 /* Exemple de code pour envoyer la valeur
46 2560 sur le servo numéro 3 */
47 char trame[3];
48 int fd;
49 trame[0]=3;
50 trame[1]=0x0A;
51 trame[2]=0x00;
52
53
54 if((fd=open_i2c("/dev/i2c",0x08))==-1)
55 {
56   printf("Erreur lors de l'ouverture du périphérique");
57 }
58
59 printf("Nombre d'octets envoyés avec succès : %d \n",write(fd, trame, 3));
60
61 exit(0);
62 }
```

Chapitre 3

Annexes

3.1 Photos du travail réalisé et typons

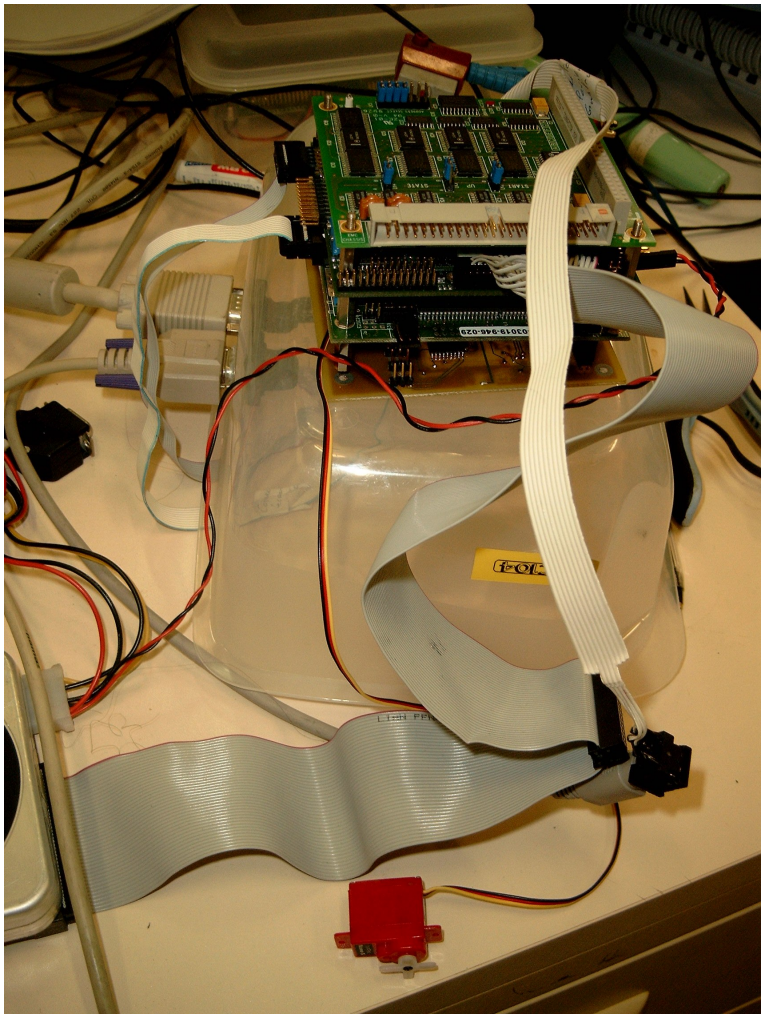

Fig. 3.1 – Le PC104 dans son ensemble

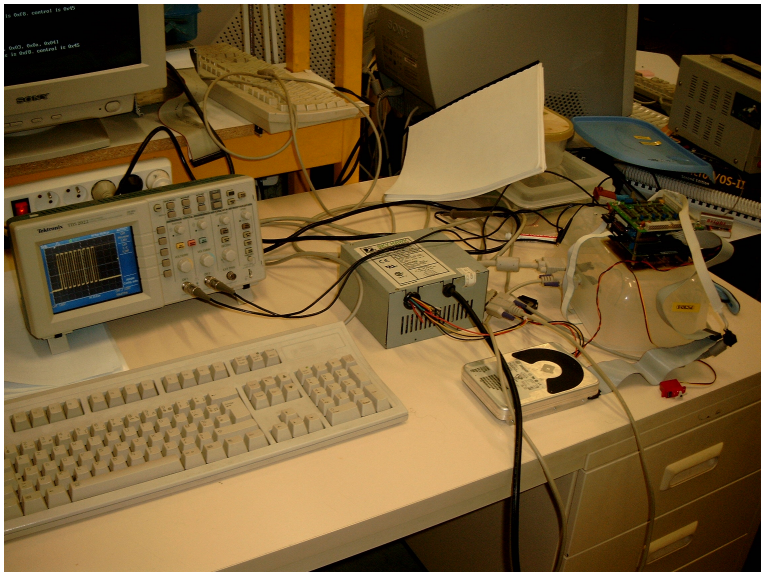

FIG. 3.2 – Le tout en fonctionnement avec le bus I2C

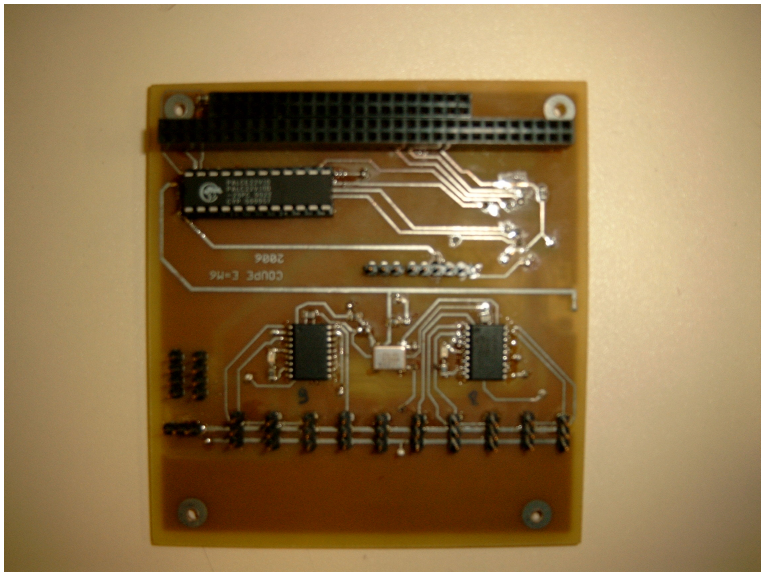

FIG. 3.3 – Vue de dessus de la carte I2C

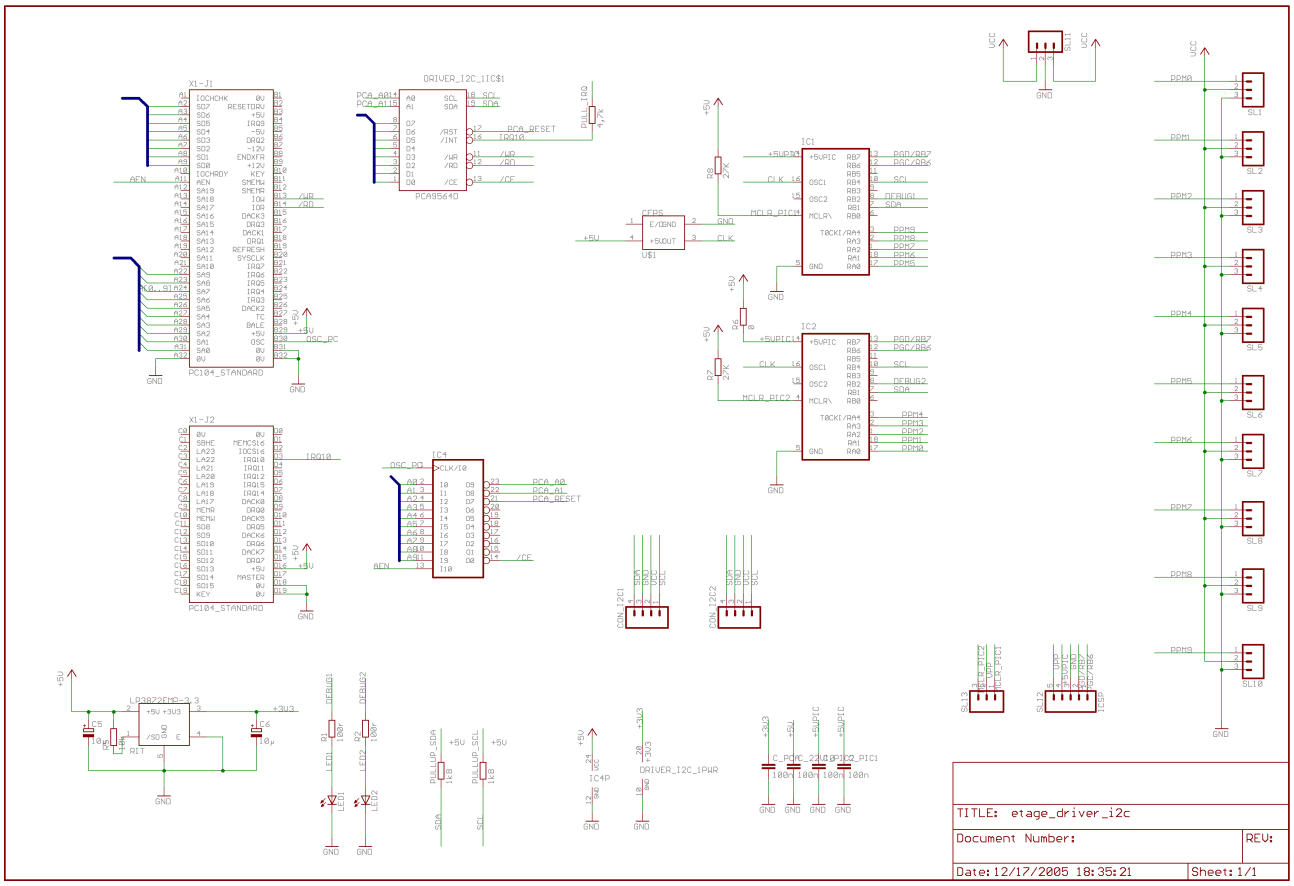

FIG. 3.4 – Schema de fonctionnement

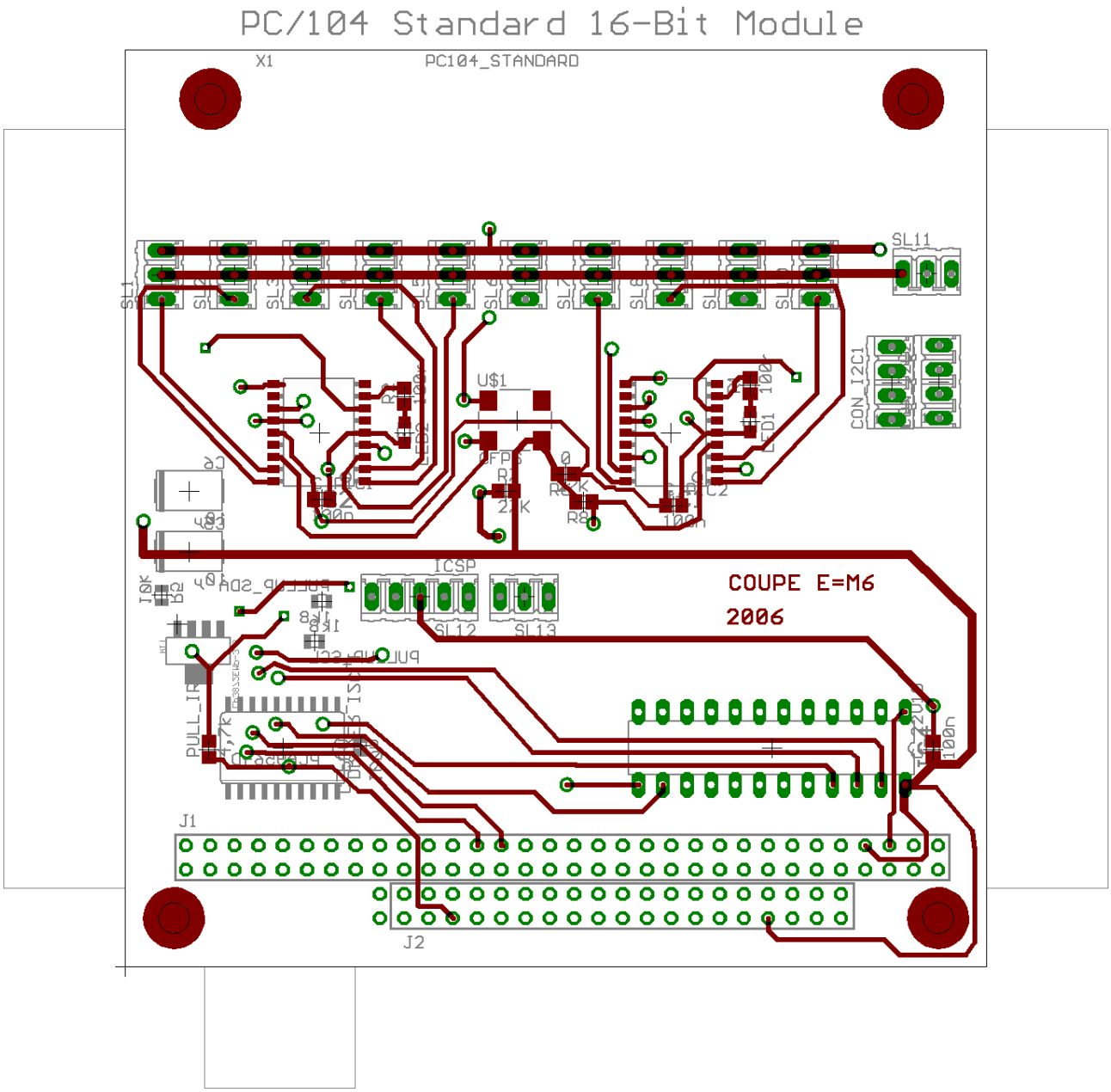

FIG. 3.5 – Typon couche dessus

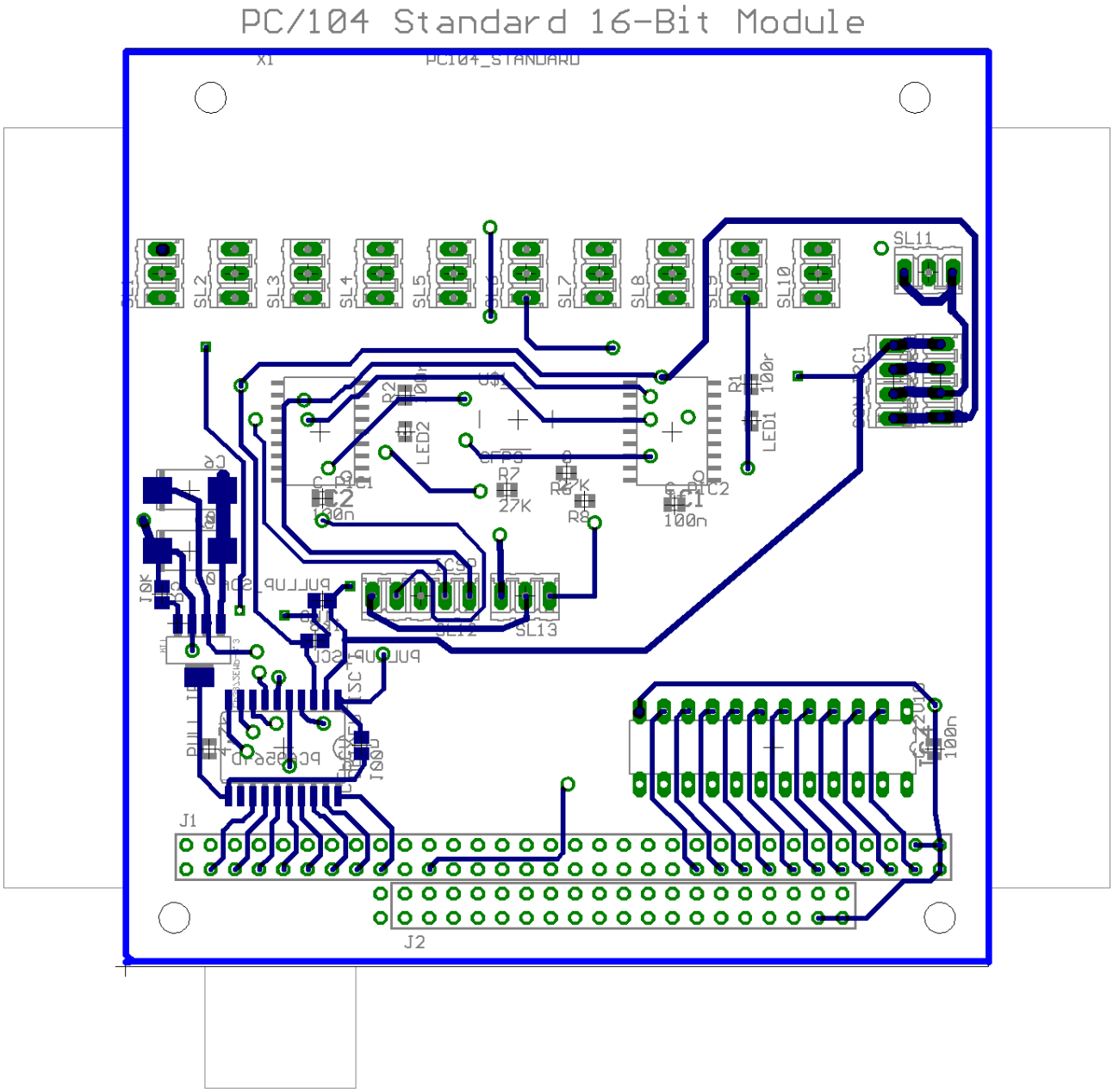

FIG. 3.6 – Typon couche dessous