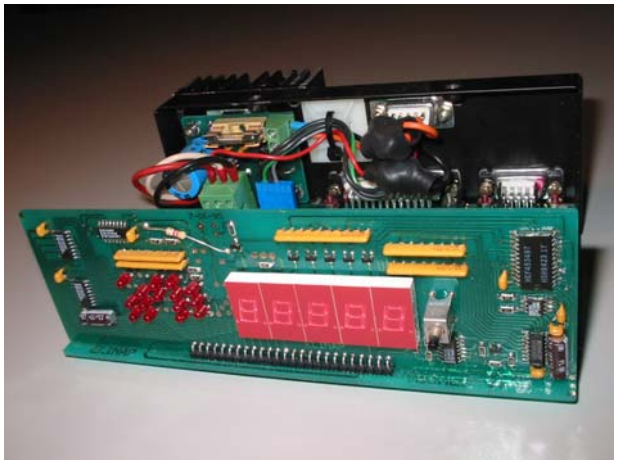
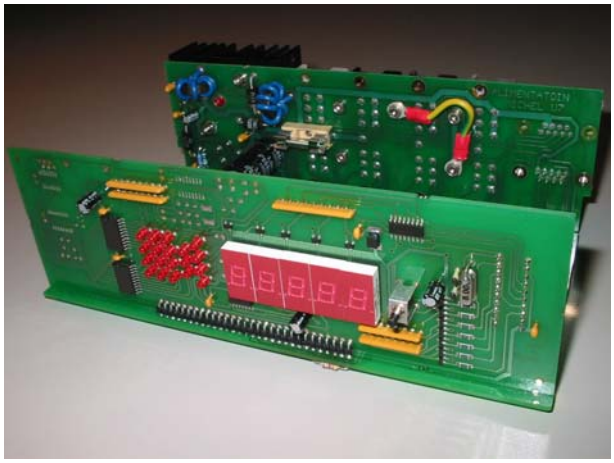
BOUVRY Colin **IUT GEII Belfort**

Etude du remplacement du microcontrôleur sécurisé de marque DALLAS





Stage dans l'entreprise ELINAP **Année 2003**

SOMMAIRE

REMERCIEMENTS

Je tiens à remercier Madame JANUSIEWICZ ainsi que Monsieur Patrick AINE pour la confiance qu'ils m'ont accordée et pour m'avoir permis d'effectuer un stage de dix semaines au sein de l'entreprise ELINAP.

J'adresse également mes remerciements à toute l'équipe de l'entreprise :

- Madame Janina JANUSIEWICZ

- Monsieur Patrick AINE

- Monsieur Olivier SAVONET

- Monsieur Thibault FOURE

- Monsieur Tadeusz SLIWA

DESCRIPTION DE L'ENTREPRISE

Fiche signalétique de l'entreprise

NOM : ELINAP – Electronique et Informatique Appliquée

DATE DE CREATION : 1988

STATUT JURIDIQUE : SARL au capital de 112 500 euros

ACTIVITES DE L'ENTREPRISE :

- Fabrication et commercialisation de systèmes électroniques de comptage de voyageurs et de systèmes d'informations pour des sociétés de transport.
- Elaboration de logiciels informatiques adaptés au matériel.
- Installation et maintenance du matériel vendu.

Organisation de l'entreprise

Cette société comprend 5 salariés :

- Madame Janina JANUSIEWICZ, Gérante (actionnaire majoritaire).
- Monsieur Patrick AINE, Ingénieur Responsable technique.
- Monsieur Tadeusz SLIWA, Ingénieur Recherche et Développement.
- Monsieur Olivier SAVONET et Thibault FOURE, Techniciens en électronique.

Description de la clientèle.

Les principaux clients de la société ELINAP sont :

- La ville de BESANCON.
- IRIS bus
- Des constructeurs de bus comme :
    - Van Hool
    - Heuliez bus
    - EVO bus
    - Bombardier Transport

- Des réseaux de transport comme le réseau de :
    - Nîmes
    - Carcasonne
    - Narbonne

SUJET DE STAGE : Etude du remplacement du microcontrôleur sécurisé de marque DALLAS.

Diplôme préparé :

La société ELINAP utilise depuis de nombreuses années des microcontrôleurs sécurisés de marque DALLAS, pour des raisons d'approvisionnement, de prix et d'évolution technologique, elle souhaite remplacer ces produits par une ou des cartes assurant les mêmes fonctions.

Objectif 1 : remplacer le microcontrôleur DS 5000 lorsqu'il n'y a pas de fonction de mémorisation par un produit compatible broche à broche, offrant une bonne protection du programme et définir les procédures d'adaptation des programmes.

Objectif 2 : remplacer le microcontrôleur DS 5000 avec les caractéristiques suivantes :

- assurer la compatibilité matérielle, brochage et support ;
- assurer l'adaptation logicielle des programmes existants ;
- améliorer les performances et la capacité mémoire pour les nouveaux projets ;
- assurer la gestion d'un périphérique de mémoire de stockage amovible.

Objectif 3 : interfacer deux capteurs spécialisés :

- capteur d'accélération ;
- capteur de comptage d'impulsion lumineuse.

Objectif 4 : respecter la réglementation CEM

DEROULEMENT DU STAGE :

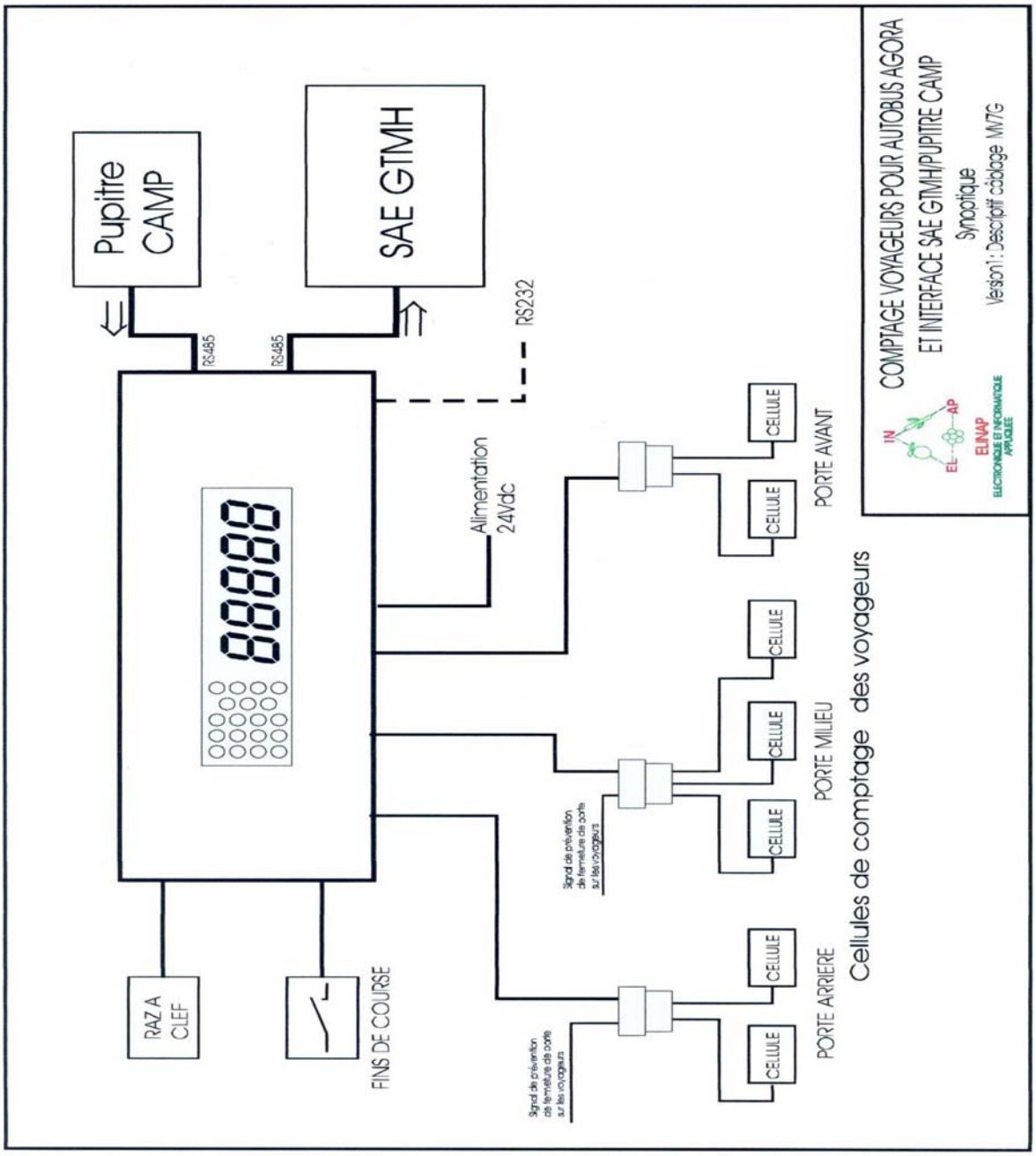
Objectif 1 : J'ai tout d'abord remplacé le microcontrôleur DS 5000 sur un système de comptage simple sans mémorisation par un produit compatible broche à broche, offrant une bonne protection du programme et j'ai défini les procédures d'adaptation des programmes.

Objectif 2 : J'ai remplacé le microcontrôleur DS 5000 avec les caractéristiques suivantes :

- assurer la compatibilité matérielle, brochage et support ;
- assurer l'adaptation logicielle des programmes existants ;
- améliorer les performances et la capacité mémoire pour les nouveaux projets

PRESENTATION DES SYSTEMES DE COMPTAGE

Schéma fonctionnel d'une installation dans un autobus

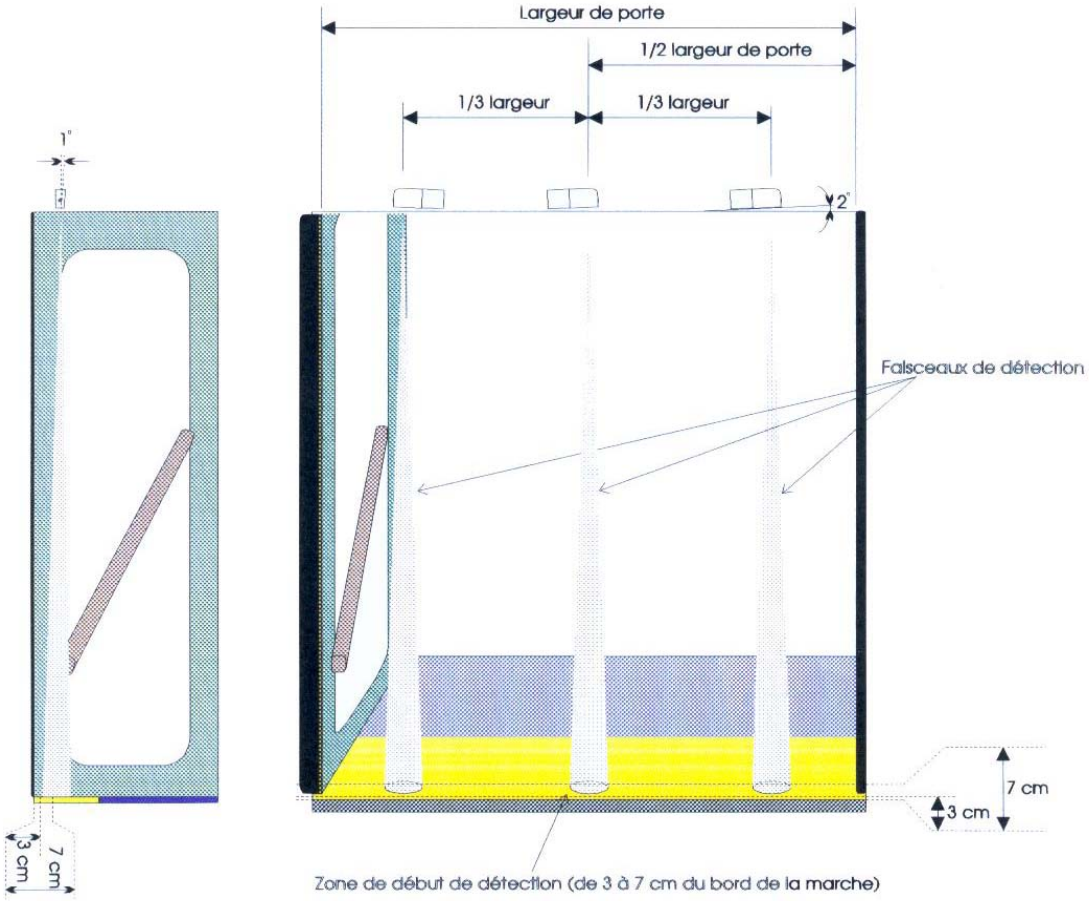


Pupitre CAMP : Son rôle est de commander des validateurs de ticket de transport. Il met à l'heure, compte le nombre de ticket et éventuellement possède un afficheur pour l'affichage du compteur.

SAE GTMH : Système d'aide à l'exploitation. Il gère la communication radio, l'indication au chauffeur du retard, de l'avance, de la régularité des passages du véhicule …

Détections

La détection des voyageurs se fait par des cellules placées bien précisément au dessus de la porte. Il existe deux types de porte, la porte sans séparation et la porte avec séparation.
Pour la porte sans séparation, il y a 3 cellules de détection et pour la porte avec séparation (par exemple une barrière), deux cellules de détection sont utilisées.



Historique des réalisations

1988 : MICHEL V4
Système de comptage de voyageurs qui comptabilise les flux de passagers (sans reconnaissance du sens de passage) et qui présente le résultat sur un afficheur à LEDs.

1989 : MICHEL V5
Boîtier MICHEL V4 amélioré.

1990 : TADE V2
Premier système de comptage avec reconnaissance du sens de passage. Il ne gère pas les oblitérations, les distances et les temps d'ouverture des portes mais les données de comptage peuvent être récupérées par un système extérieur.

1992 : TBA (tade bus autonome)
Le TBA est un système de régulation embarquée pouvant fonctionner en mode centralisé (SAE) ou autonome.
La récupération des données se fait automatiquement (par balise Hyperfréquence). Il dispose d'un pupitre de commande dans le bus et permet au centre de régulation de connaître l'avancement des bus sur leurs trajets.

1993 : IDB (information dans le bus)
Borne d'information placée dans le bus et permet au centre de régulation de connaître l'avancement des bus sur leurs trajets.

1994 : ELIN V2C
Système de comptage évolué permettant d'obtenir de nombreuses informations supplémentaires : montées, descentes, distance, charge, oblitérations, temps d'ouvertures des portes.

1994 : MICHEL V7
Amélioration par rapport au michel V5 : le système peut maintenant stocker les informations de comptage et faire l'objet d'une récupération de données.

1995 : ELIN V2C/12
C'est une version limitée du modèle ELIN v2C qui à l'avantage d'être moins onéreuse.

1996 : MICHEL V7B
Amélioration par rapport au michel V7. Configuration automatique du système en fonction du type de véhicule (ajustement automatique du nombre de portes).

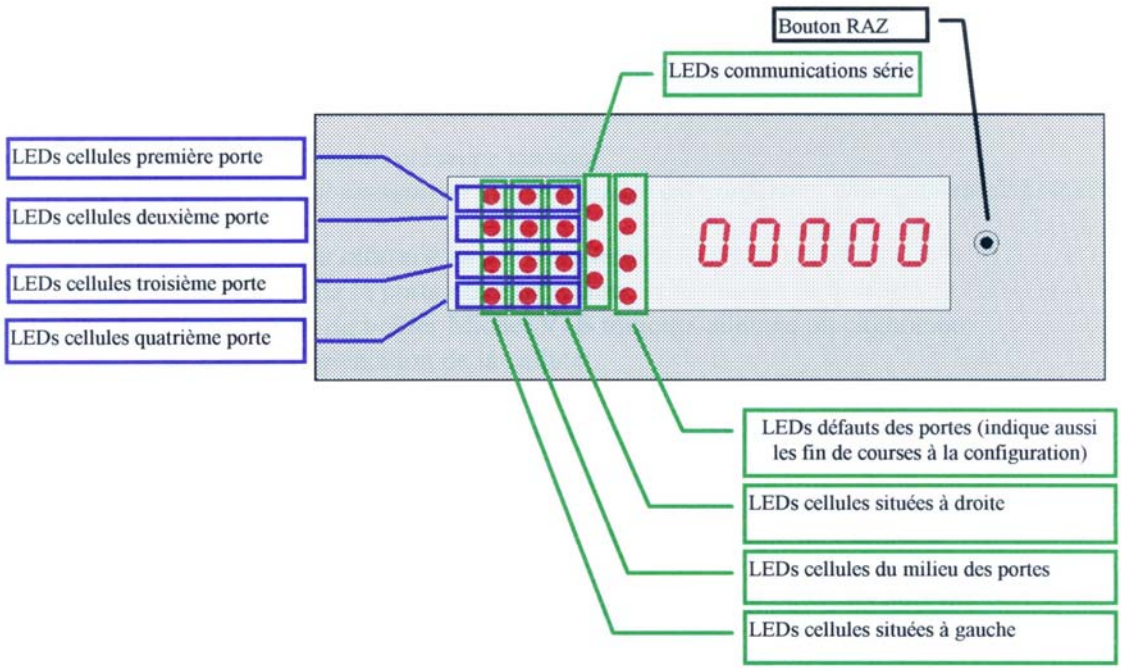
Les différents systèmes de comptage sur lesquels j'ai travaillé

Il existe plusieurs versions :

- Le système de comptage simple sans mémorisation sur MICHEL V7
- Le système de comptage avec mémorisation avant la coupure de l'alimentation sur MICHEL V7B
- Le système de comptage avec mémorisation avant la coupure de l'alimentation et archivage sur au moins quinze jours du comptage sur MICHEL V7B

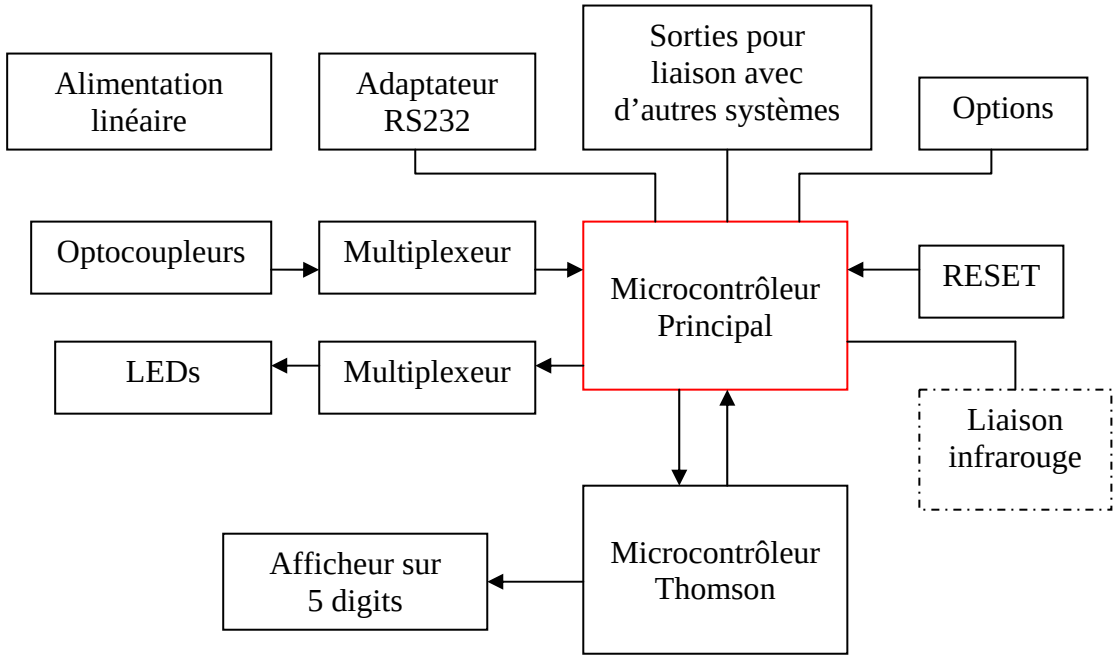
Utilisation du système de comptage MICHEL V7B

Le système de comptage MICHEL V7 B se configure automatiquement en fonction des caractéristiques du véhicule (1,2 ou 3 cellules par porte, 1 à 4 portes). Pour que la configuration se réalise, il convient de procéder à quelques manipulations lors de l'installation dans le véhicule.



Les LEDs servent à indiquer le bon fonctionnement des différentes cellules.
Le bouton sert à la remise à zéro, et aussi à la mise sous tension, au passage au mode de configuration.
La remise à zéro s'effectue un bref instant après que l'on ait appuyé sur le bouton de R.A.Z.

Schéma fonctionnel du système de comptage MICHEL V7B



RAPPEL SUR LES MEMOIRES

Il existe deux Acronymes pour définir les Mémoires

ROM : Read Only Memory (Mémoire Lecture Seule) - Mémoire Morte

RAM : Random Access Memory (Mémoire à accès Aléatoire) - Mémoire Vive

Les mémoires RAM

Technologie SRAM Bilan

Rapide
Cellule Statique : Rétention infinie tant qu'alimentation
Faible Consommation (CMOS)
Cellule volumineuse
Chère
Volatile

Technologie DRAM Bilan

Compact
Faible Consommation (CMOS)
Lecture destructrice : nécessité de réécriture
Rafraîchissement régulier
Volatile

Les mémoires ROM

Nécessité de mémoire non- volatile
Bios de micro- ordinateur
Stockage de programmes dans des systèmes embarqués
Reconfiguration automatique des FPGA

Les mémoires ROM - Technologie

Mask - ROM : réalisé lors de la fabrication du circuit
PROM : Fusible - One Time Programming
EPROM : Transistors à Grille Flottante Reprogrammabilité
NOVRAM : RAM + Pile
FRAM : technologie Ferroélectrique

Mask - ROM

L'écriture des données dans les emplacements mémoires est effectuée une fois pour tout par le fabricant.
Aucune Souplesse
Faible Coût

PROM

Programmable qu'une fois par l'utilisateur
Technologie à Fusibles et Antifusibles
Très faible Utilisation

La Grille Flottante

Piège des électrons dans la grille flottante (Tension de l'ordre de 13 V)

EPROM - Technologie

UV- EPROM : Effacement aux ultra- violet
EEPROM : Effacement électrique par mot mémoire
FLASH : Effacement électrique par bloc

UV- EPROM

Programmation hors système
Effacement hors système
Temps d'effacement long (15 minutes pour les UV- EPROM)

E- EPROM

Programmation ISP (In Situ Programming ou In Serial Programming)
Effacement ISP
Temps d'effacement rapide
Limité en nombre d'écriture environ 100 000
Coût élevé car effacement par mot mémoire

FLASH

Programmation ISP
Effacement ISP
Temps d'effacement rapide (quelques secondes)
Coût limité car effacement par bloc
Temps d'écriture long
Accès par bloc
Nombre d'écritures finies
Pas bonne RAM

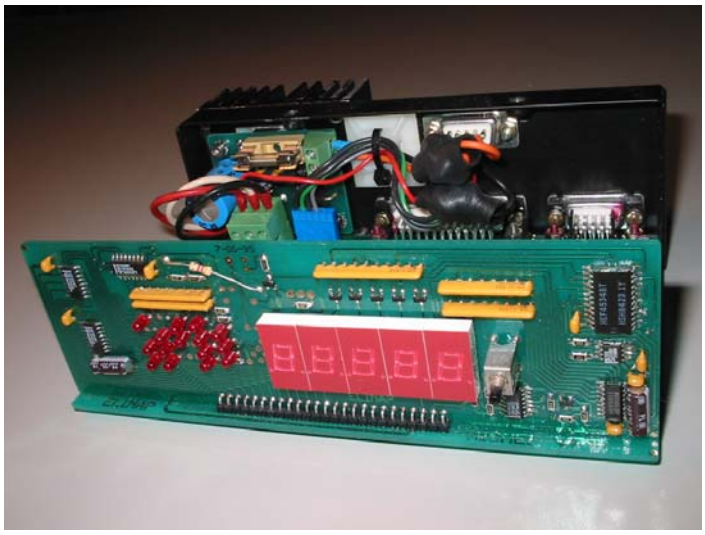
FRAM

Non volatile
Temps d'écriture très court
Durée de vie de 10 milliards d'écriture
Coût élevé

OBJECTIF N° 1

Introduction

Le but de cet objectif est de remplacer le microcontrôleur DS 5000 sur le système de comptage MICHELV7 avec le microcontrôleur P89C51RD2 de PHILIPS. Ce microcontrôleur est compatible broche à broche avec le DS5000, et le système de comptage MICHELV7 ne doit pas posséder de fonction de mémorisation donc il ne possèdera pas de modifications hardware.



MICHELV7

Déroulement

J'ai tout d'abord utilisé un compilateur et un programmateur pour le microcontrôleur PHILIPS P89C51RD2. Je me suis ensuite familiarisé avec le langage PLM et le programme MNP2ACA qui servait pour le microcontrôleur DS5000 de ce système de comptage.
Par la suite, j'ai défini les procédures d'adaptation logicielle.

Adaptation logicielle

Il faut déjà adapter les déclarations de variables

Les variables de type byte ou word sont réparties soit dans la RAM interne ou soit dans la RAM auxiliaire de 1 Koctets.
Il est possible de déclarer de deux manières différentes :

```
DCL variable        byte    AUX ;
DCL variable        byte;
```

Problème du pré diviseur d'horloge

Les produits P89C51RD2 et DS5000 ont tous les deux une architecture 8051 compatible. Mais ils n'ont pas forcément les mêmes registres de contrôle selon les besoins du client. Par exemple, le microcontrôleur P89C51RD2 possède un registre CKCON (contrôle horloge) que n'a pas le microcontrôleur DS5000.

Le produit PHILIPS possède deux diviseurs d'horloge, diviseur par 6 et diviseur par 12 contrôlable par le registre CKCON. J'ai donc testé la vitesse d'exécution du programme.

Programme permettant de savoir la vitesse d'exécution du programme :

Ce programme permet d'allumer et d'éteindre une LED. Avec un chronomètre on peut ainsi savoir si la durée est la bonne.
Les morceaux de programme sont rajoutés au programme « Mnp2aca »

```
/*************************************************************/
/* Dans le programme principal */

if test >= 250 then do ;
        def4 = not(def4);
        test=0;
end;

/* Dans l'interruption périodique toutes les 10 ms */

test = test + 1;
/*************************************************************/
```

Une impulsion lumineuse ou non lumineuse doit durer normalement 250 *10ms soit 2,5s cela veut dire que la LED s'allume toutes les 5 s.
En chronométrant sur 60 s, on trouve 24 impulsions lumineuses, soit une impulsion toutes les 60/24 soit 2,5 s. En comptant, cela signifie qu'on a une interruption toutes les 5 ms.
Le microcontrôleur DALLAS possède un diviseur d'horloge par 12.
Alors que le microcontrôleur PHILIPS a deux diviseurs d'horloge contrôlable par le registre CKCON. Il se met par défaut en diviseur par 6.

Deux solutions existent pour remédier au problème du diviseur :

- On configure le registre CKCON en mode diviseur par 12.
- On modifie les valeurs de rechargement automatique du timer 0 pour obtenir le temps d'interruption adéquate.

<u>Première solution :</u>
Il faut mettre CKCON.1 (T0X2) à 1 (voir annexe)
« 0 » pour diviseur par 6
« 1 » pour diviseur par 12

/* on rajoute une ligne de programme dans la procédure config */
CKCON = CKCON or 02h;
/*met à « 1 » le deuxième bit de CKCON sans modifier les autres */

CKCON étant protégé par des sécurités, j'ai donc essayé de contourner celles-ci.
J'ai essayé toutes sortes de chose sans résultat. Cette solution est donc abandonnée.

Deuxième solution :
Il faut recharger de nouvelles valeurs dans TH0 et TL0 du Timer 0.
Le timer 0 joue le rôle d'envoi de trame pour l'affichage du compteur sur 5 digits.

Calcul des nouvelles valeurs :

TH0 = 0DBh
TL0 = 0FFh
0FFFFh – 0DBFFh = 2400h
0FFFFh – 2*2400h = 0B7FFh ; il faut que la durée soit 2 fois plus longue
TH0 = 0B7h
TL0 = 0FF h

Il faut remplacer TH0 = 0DBh et TL0 = 0FFh; par TH0 = 0B7h; et TL0 = 0FFh;

/* Dans l'interruption périodique toutes les 10 ms */

TH0 = 0B7h;
TL0 = 0FFh;

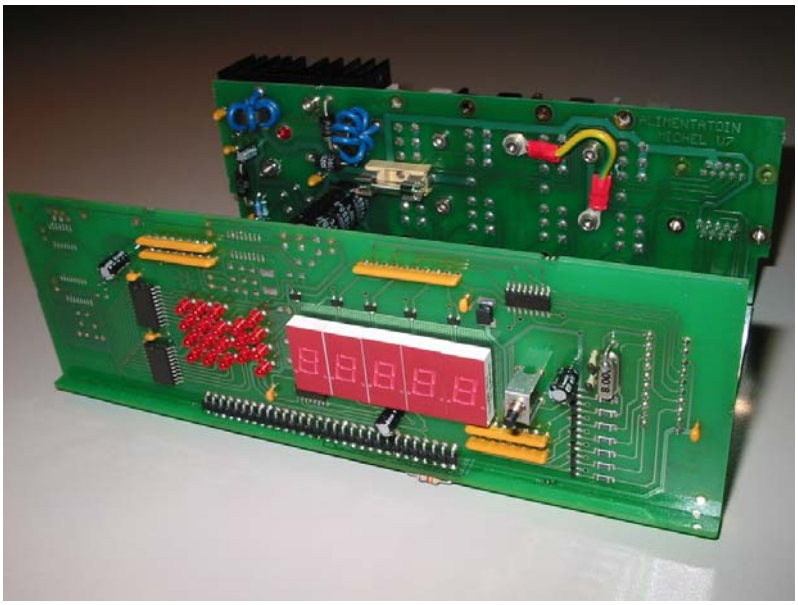
En testant, on a bien cette fois ci 12 impulsions lumineuse en 60s ce qui donne une interruption périodique de 10 ms.

OBJECTIF N° 2

Introduction

Le but de cet objectif est de remplacer le microcontrôleur DS 5000 sur le système de comptage MICHELV7B avec les microcontrôleurs T89C51RD2 de ATMEL et P89C51RD2 de PHILIPS . Ces deux microcontrôleurs sont compatibles broche à broche avec le DS5000. Le système de comptage MICHELV7B doit pouvoir mémoriser si possible sans modification hardware.
Il existe deux versions de système de comptage :

- Le système de comptage avec mémorisation avant la coupure de l'alimentation sur MICHEL V7B
- Le système de comptage avec mémorisation avant la coupure de l'alimentation et archivage sur au moins quinze jours du comptage sur MICHEL V7B



MICHELV7B

AMELIORATION DE LA CAPACITE MEMOIRE

Introduction

Le microcontrôleur DS 5000 de DALLAS a de la mémoire FLASH, de la mémoire RAM sauvegardée, un calendrier et une horloge. De plus ces mémoires sont assez volumineuses pour de grosses sauvegardes.
Les microcontrôleurs ATMEL ou PHILIPS ne possédant pas tous ces avantages, il faut soit les interfacer avec des composants externes ou utiliser leur mémoire interne.
Cette partie consiste déjà à voir tout ce qui est possible de réaliser par la théorie et par la pratique pour enfin choisir la meilleure solution.

Contraintes

Pour interfacer des composants, les microcontrôleurs ne possèdent que de 3 broches disponibles.

- **Pour le système de comptage avec mémorisation avant la coupure de l'alimentation :**

Au rallumage, il faut que le système puisse retrouver les anciennes valeurs utilisées comme la variable du compteur, la configuration…

Pour cette mémorisation, plusieurs méthodes ont été étudiées :

Pour le P89C51RD2 de PHILIPS, la mémorisation dans la FLASH EEPROM interne au microcontrôleur, ou la mémorisation par un composant externe au microcontrôleur est possible théoriquement.

Pour le AT89C51RD2 de ATMEL, la mémorisation dans la FLASH EEPROM ou EEPROM interne au microcontrôleur, ou la mémorisation par un composant externe au microcontrôleur est possible théoriquement.

- **Pour le système de comptage avec mémorisation à la coupure de l'alimentation et archivage sur au moins quinze jours du comptage :**

Il faut interfacer, en plus que précédemment, une RAM sauvegardée pour des variables qui doivent être mémorisées souvent, un calendrier et une horloge, et de la mémoire de type EEPROM assez importante pour des sauvegardes étalées sur au moins 2 semaines.

Déroulement et explications

Tout d'abord, pour remédier à ces contraintes, le choix s'est porté sur le protocole de communication bus I2C. Ce bus I2C est synchrone et tient sur 2 broches. Ce protocole est très utilisé, et de nombreux composants se contrôlent par ce bus. Ce qui assure sa pérennité.
Ensuite, le choix du langage de programmation pour construire le bus I2C s'est posé entre le langage assembleur 8051 et le langage PLM. Pour des raisons de rapidité, le choix s'est fait sur le langage assembleur « très proche de la machine » plutôt que le langage évolué PLM moins rapide. Une fois le programme assembleur du bus I2C réalisé, il faut utiliser un linkeur qui permet de faire le lien entre le programme assembleur et le programme PLM. Bref. Je me suis retrouvé dans une impasse dans l'incapacité de résoudre des erreurs de linkage.
On a décidé d'abandonner le programme en assembleur pour le remplacer par un programme entièrement en PLM qui a l'inconvénient d'être moins rapide.
Par la suite, j'ai défini les procédures pour la communication I2C et réalisé des tests avec différentes mémoires. J'ai donc réalisé des procédures pour gérer des mémoires EEPROM SRAM et FRAM sachant qu'elles peuvent avoir soit 8 ou 16 bits d'adresse, et pour gérer un calendrier et une horloge.

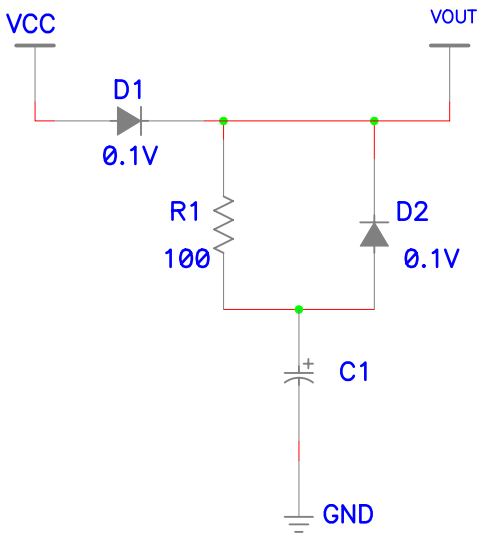
Ensuite, un problème s'est posé : A quel moment, on mémorise les données à conserver avant l'extinction du système de comptage ? Sachant que l'EEPROM est limité en écriture à 100000 fois dans une même case mémoire. Pour cela deux solutions :

Soit on mémorise à chaque fermeture des portes du véhicule ou soit on mémorise à la coupure de l'alimentation.

Pour la mémorisation à chaque fermeture des portes, c'est à dire après chaque arrêt, il faut écrire plus de 800 fois par jour, ce qui fait un peu près sur dix ans 800*365*10 = 3 millions. Ce qui dépasse largement les 100000 fois ! Il faut alors écrire à des endroits différents de la mémoire EEPROM pour ne pas utiliser toujours les mêmes cases mémoire. Il s'agit ici de gestion de la détérioration de mémoire ! Il faut donc aussi mémoriser le pointeur du dernier enregistrement afin de retrouver à l'allumage du système les dernières données mémorisées. Pour ce dernier, il faut obligatoirement de la SRAM sauvegardée qui a les avantages d'une réécriture beaucoup plus importante que l'EEPROM et d'une sauvegarde quand le système n'est plus alimenté.

Pour la mémorisation à la coupure de l'alimentation, c'est à dire à la baisse de tension de l'alimentation générale. On détecte la coupure de l'alimentation par une entrée d'interruption du microcontrôleur relié au circuit électronique RESET (Voir modifications matérielles). Par la suite, il est possible de faire des opérations de sauvegarde pendant le cours temps de décharge des capacités de l'alimentation du système de comptage.

J'ai décidé de poursuivre plutôt la mémorisation à la coupure de l'alimentation car il y a moins de contraintes.
J'ai donc essayé de mémoriser dans la mémoire EEPROM interne au microcontrôleur ATMEL. J'ai testé les procédures d'écriture et de lecture dans l'EEPROM ce qui fonctionnait très bien. Mais le problème était à la coupure de l'alimentation, l'écriture dans l'EEPROM n'était pas fiable dû à l'instabilité de la tension ce qui amenaient des valeurs aléatoires dans les cases mémoires de l'EEPROM. Alors pour remédier à ce problème, j'ai interfacé un petit montage permettant de prolonger la tension d'alimentation du microcontrôleur. (Voir montage ci-dessous) Ce petit montage est interfacé entre le +5V de l'alimentation et l'alimentation du microcontrôleur.



Une fois interfacé ce petit montage, le temps d'alimentation était beaucoup plus important et le problème n'était pas résolu. J'en ai conclu que cela ne marchait pas avec une alimentation non stabilisée. Ce qui implique de faire de plus lourdes modifications sur l'alimentation. J'ai donc choisi une mémorisation par accès bus I2C externe au microcontrôleur. La mémoire externe peut

être soit une SRAM sauvegardée ou soit une FRAM. La SRAM sauvegardée a un nombre infini en écriture et un accès rapide, tandis que la FRAM possède une durée de vie de 10 milliards d'écriture, un accès rapide et elle est en plus non volatile. On est dans les cas plus limités en nombre d'écriture on peut alors mémoriser soit à la coupure de l'alimentation ou soit à la fermeture des portes.

Ce qui fait quatre possibilités de mémorisations :

- Mémorisation dans la SRAM sauvegardée à la coupure de l'alimentation
- Mémorisation dans la SRAM sauvegardée à chaque fermeture des portes
- Mémorisation dans la FRAM à la coupure de l'alimentation
- Mémorisation dans la FRAM à chaque fermeture des portes

J'ai défini l'adaptation logicielle pour chacun de ces cas.

Une autre solution est encore possible mais délicate, l'écriture dans la mémoire FLASH.
J'ai essayé de mémoriser de cette façon mais sans succès.

Enfin, j'ai travaillé sur la compréhension du programme de comptage avec mémorisation avant la coupure de l'alimentation et archivage des données.

Explication :
L'archivage est réalisé en fonction de la configuration choisie sur le logiciel de gestion de l'archivage. La capacité mémoire est en fonction de la configuration du choix des données à enregistrer.
Pour améliorer la rapidité de l'écriture sur l'EEPROM, il est possible d'y écrire en mode page sur le bus I2C (Voir annexe). C'est-à-dire on envoie les octets à écrire les uns après les autres sans envoyer l'adresse. L'adresse s'incrémente tout seul sur la mémoire EEPROM jusqu'au maximum 128 octets. On définit ainsi, une page est égale à 128 octets.

La plus grande capacité mémoire EEPROM à accès par bus I2C trouvée est de 512 kbits.
Le maximum de mémoire EEPROM à 512 kbits qu'on peut interfacer ensemble est de 4. Car il possède chacun 2 bits d'adresse.
Ce qui fait au total, 4 fois 512 kbits d'espace mémoire disponible. C'est-à-dire 2048 pages de 128 octets.
Dans la configuration où le nombre de données à enregistrer est maximal, le nombre de la taille mémoire que cela occupe par jour est de 20 pages à 160 pages selon la durée d'utilisation du véhicule.
Soit environ 80 pages en moyenne par jour.
Ce qui fait en moyenne 2048/80 soit environ 25 jours d'archivage possible.

Choix

Protocole de communication I2C.

Pour le système de comptage avec mémorisation avant la coupure de l'alimentation.

Pour la mémorisation dans la SRAM :
Le choix s'est porté sur une mémoire RAM M41T11. Il possède 56 octets de RAM, un calendrier et une horloge, et un contrôle par bus I2C. L'horloge et le calendrier ne seront pas utilisés dans cette mémorisation.

Pour la mémorisation dans la FRAM :
Le choix s'est porté sur une mémoire FRAM FM24CXX du constructeur RAMTRON, XX définit la taille de la mémoire, on n'a pas besoin de grand espace mémoire. Il faut environ une bonne cinquantaine d'octets. On peut utiliser une mémoire très faible. Les essais se sont portés sur une FRAM FM24C64 de 64 Kbits soit 8 Koctets.

Pour le système de comptage avec mémorisation avant la coupure de l'alimentation et archivage sur au moins quinze jours du comptage

Le choix s'est porté sur une mémoire RAM M41T11 pour sa RAM, horloge et calendrier.
Pour ce qui est de l' EEPROM, le choix s'est porté sur les mémoires AT24C512 de ATMEL de 64 koctets. Avec quatre de ces mémoires (le maximum), lui fait une capacité mémoire au total de 256 koctets.
Une telle capacité est nécessaire pour enregistrer des données pendant au moins quinze jours.

Description du Bus I2C

Les signaux SDA et SCL.

Le bus I2C véhicule les données à l'aide de deux signaux synchrones :

SDA (Signal DAta)
SCL (Signal CLock).

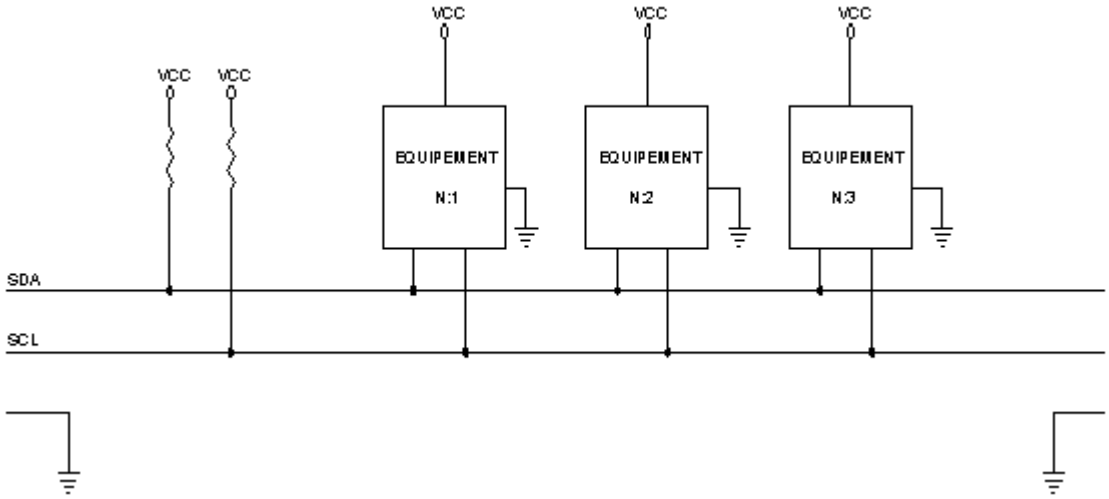


Figure 1

Les dispositifs qui doivent se raccorder au bus I2C viennent se connecter en parallèle sur les lignes SDA et SCL. Ceci est possible grâce à la structure des sorties qui sont de type 'collecteur ouvert'. Des résistances de rappel à VCC permettent de garantir l'état haut des signaux SDA et SCL lorsque tous les nœuds du bus ont leur sortie en haute impédance. Les tensions associées aux niveaux logiques des lignes SDA et SCL dépendent de la technologie des circuits en présence sur le bus (CMOS, TTL).

Le bus I2C fonctionne en Maître – esclave, c'est-à-dire le maître engage toujours la parole et l’esclave renvoie ce qu’on lui dit de faire.

La rapidité du bus I2C à un maximum de 100 kHz, ou 400 khz et même 1000 khz selon les circuits intégrés, et n’a pas de minimum.

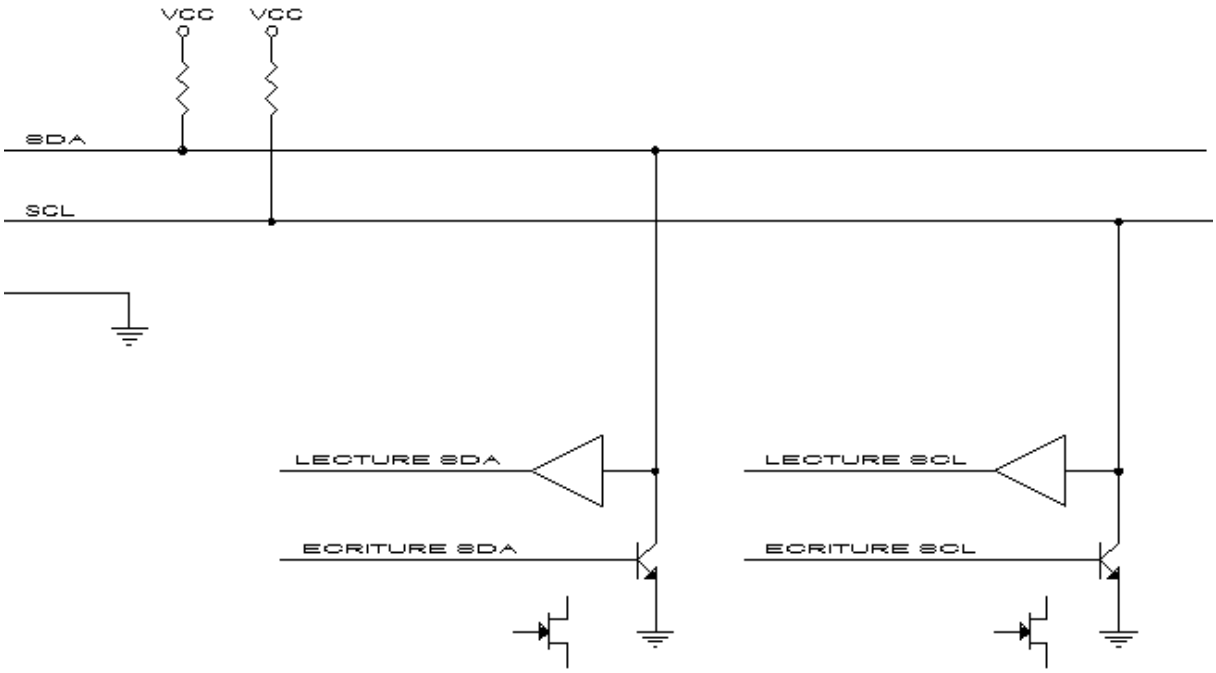


Figure 2

Le protocole I2C. (Pour plus de détails voir annexe)

Le protocole du bus I2C définit la succession des états possibles pour les signaux SDA et SCL. Le protocole définit aussi comment doivent réagir les nœuds raccordés au bus en cas de conflit. Avant d'initier un dialogue sur le bus, un nœud doit s'assurer que le bus I2C est libre. Pour se faire le nœud vérifiera si les lignes SDA et SCL sont au repos (à l'état haut) pendant un intervalle de temps suffisant. Pour transmettre des données sur un bus I2C, un nœud doit surveiller deux conditions particulières: la condition de départ " START " et la condition d'arrêt " STOP "

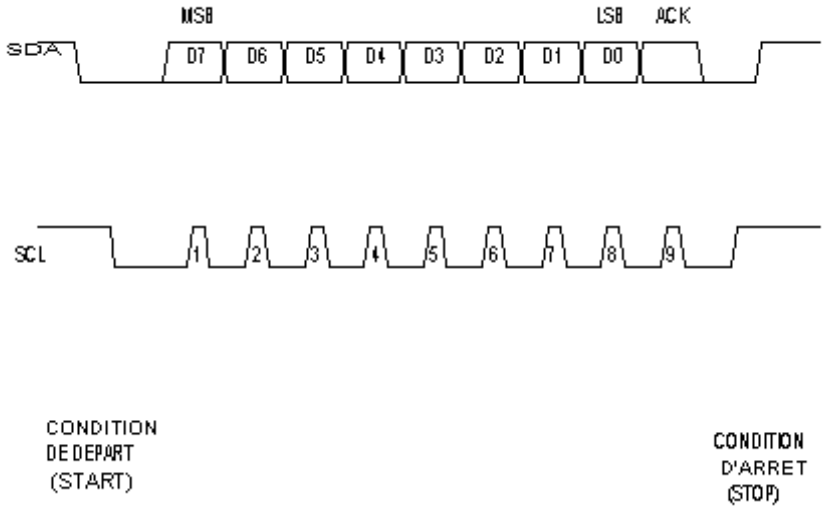


Figure 3

ADAPTATION MATERIELLE

Introduction

A l'extinction de l'alimentation générale, on perd toutes les données.
Pour y remédier, il y a quatre façons de mémoriser les données :

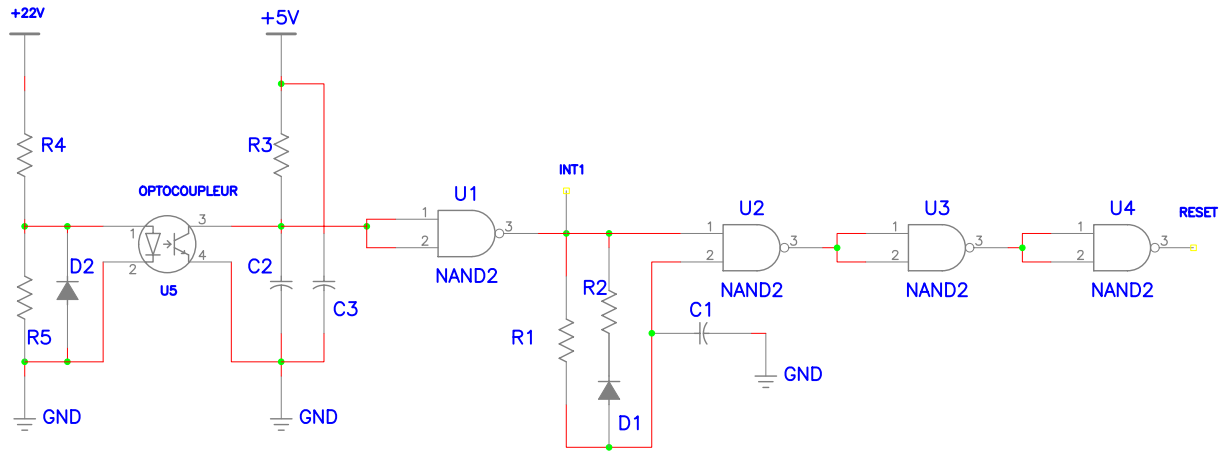
- Mémorisation dans la SRAM sauvegardée à la coupure de l'alimentation
- Mémorisation dans la SRAM sauvegardée à chaque fermeture des portes
- Mémorisation dans la FRAM à la coupure de l'alimentation
- Mémorisation dans la FRAM à chaque fermeture des portes

Modification du système de comptage MICHEL V7B pour la mémorisation à la coupure de l'alimentation

Pour mémoriser à la coupure de l'alimentation, il faut détecter à quel moment on doit mémoriser. Cette détection va générer une interruption sur le microcontrôleur pour déclencher un enregistrement des variables.
Pour réaliser cette détection, on utilise le circuit RESET du système de comptage.

INT1 est actif à l'état bas ou sur front descendant selon la configuration d'un registre. (Voir annexe sur les microcontrôleurs)
RESET est actif à l'état haut

Le circuit RESET :



U2 (3) = U4 (3)

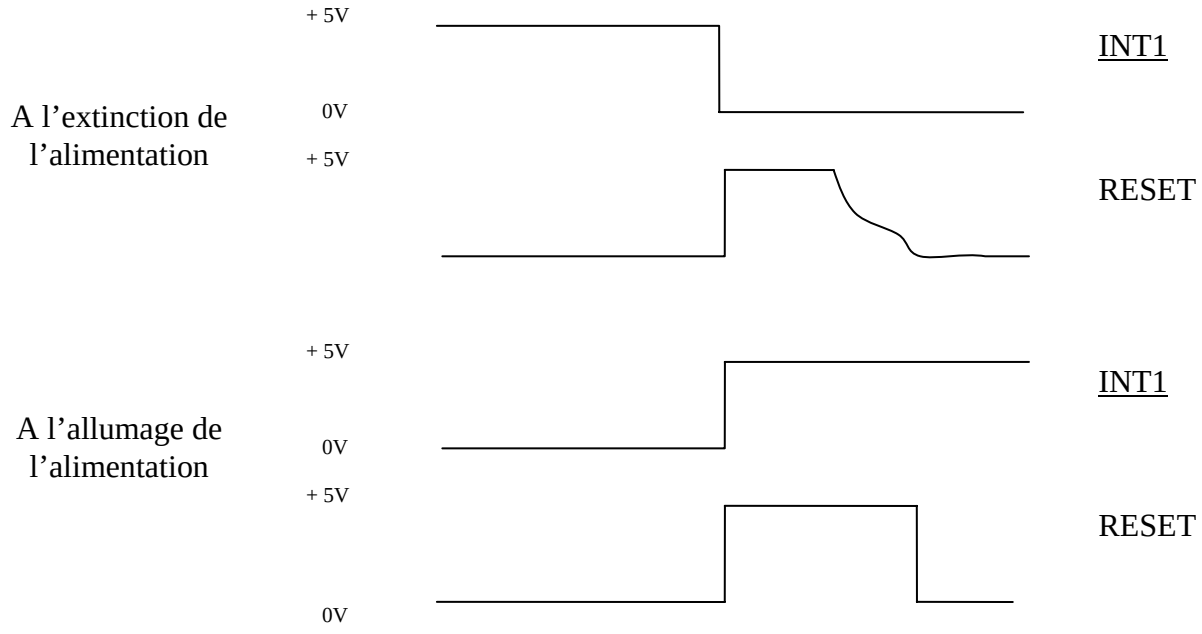
Entre le 22V et le 5V, il y a une alimentation avec des capacités. Il y a donc un temps de charge et de décharge des capacités, ce qui nous laisse le temps en jouant avec cette différence entre le 22V et le 5V de faire des opérations après la coupure de l'alimentation.

Explications :

A l'allumage, on a 22V à l'entrée et l'optocoupleur est passant donc on a un « 0 » logique aux entrées de la porte U1, ce qui entraîne un « 1 » logique à l'entrée 1 de la porte U2. Le condensateur C1 étant déchargé, un « 0 » logique est à l'entrée 2 de la porte U2. Ce qui fait un « 1 » logique sur la sortie de la porte U2 et donc sur le RESET car U3 et U4 sont montées en inverseuse. Ensuite, le condensateur se charge à travers R1 et procure à partir de VCC/2 un « 1 » logique sur l'entrée 2 de la porte U2, ce qui entraîne un « 0 » logique sur RESET. Le programme pourra tourner normalement sur le front descendant de RESET.

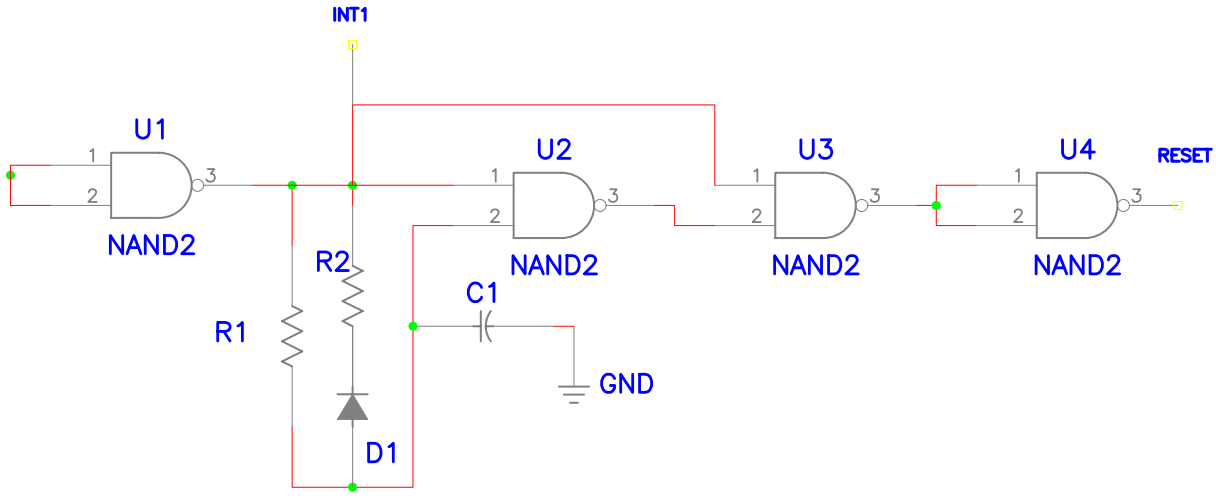
A l'extinction, on a 0V à l'entrée et l'optocoupleur est bloqué donc on a un « 1 » logique aux entrées de la porte U1, ce qui entraîne un « 0 » logique à l'entrée 1 de la porte U2. Le condensateur C1 étant chargé, un « 1 » logique est à l'entrée 2 de la porte U2. Ce qui fait un « 1 » logique sur la sortie de la porte U2 et donc sur le RESET car U3 et U4 sont montées en inverseuse. Ensuite, le condensateur se décharge à travers R1 et R2 et procure à partir de VCC/2 un « 0 » logique sur l'entrée 2 de la porte U2, ce qui entraîne un « 0 » logique sur RESET. En réalité, le RESET ne passe pas à « 0 » logique à cause des portes mais à cause de la baisse de l'alimentation.

Chronogrammes :



Problème :
A la coupure de la tension d'alimentation, un RESET est généré sur le microcontrôleur et l'interruption ne peut se produire car le RESET est incontestablement prioritaire sur les interruptions externes.
Pour résoudre ce problème une petite modification est nécessaire.
Le but de cette modification est de supprimer le RESET à l'extinction de l'alimentation.

Le circuit RESET avec modification :



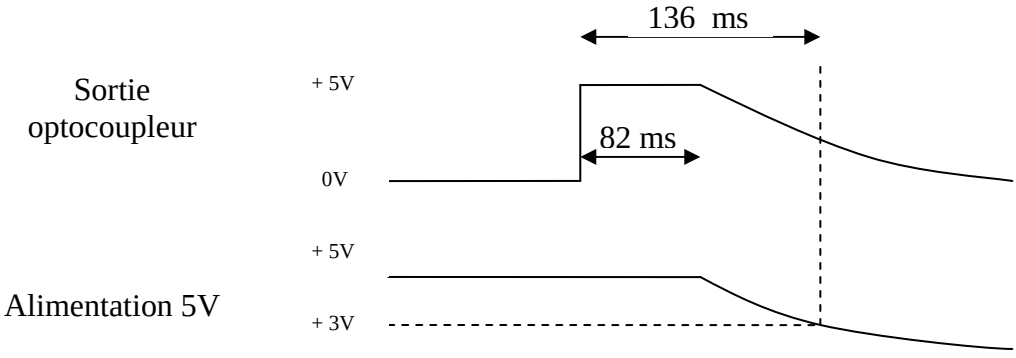
Explications :

	U2 (1)	U2 (2)	U2 (3)	RESET
A l'allumage	1	0	1	1
	1	1	0	0
A l'extinction	0	1	1	0
	0	0	1	0

U2 (3) est l'ancien RESET (sans la modification).
RESET = U2 (1) ET U2 (3)
Soit U3 (3) = U2 (1) ET U2 (3)

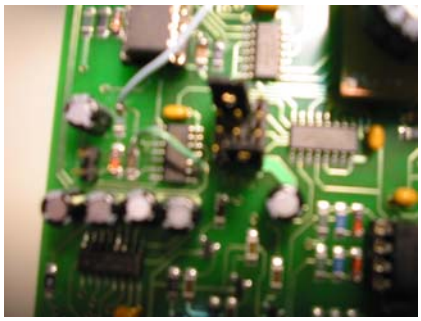
Le temps de sauvegarde à la coupure de l'alimentation :

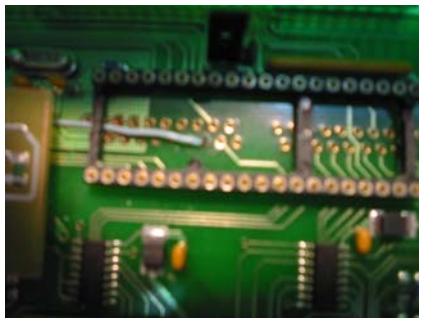
Les microcontrôleurs ATMEL et PHILIPS fonctionnent avec une alimentation entre 3V et 5,5V.



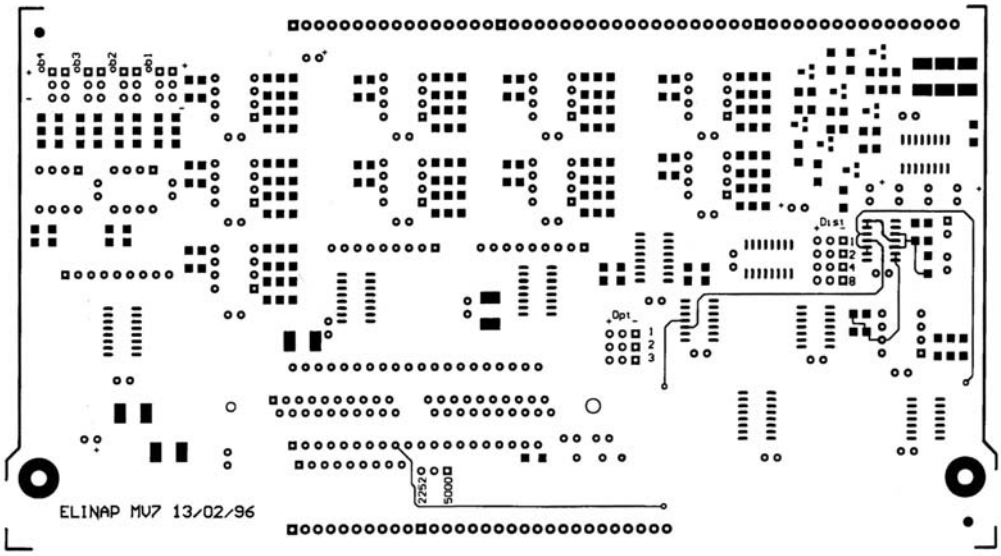
Modifications hardware :

Photos des modifications :

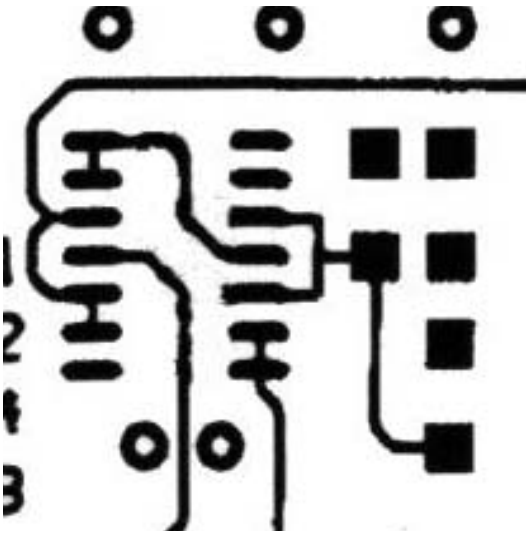




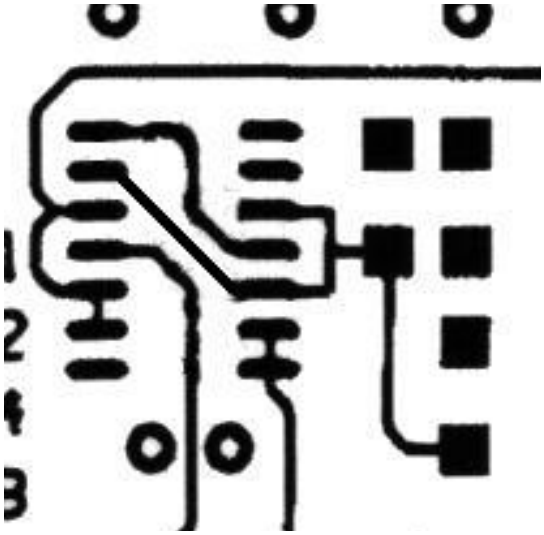
Typon global du RESET sans modifications :



RESET sans modifications :

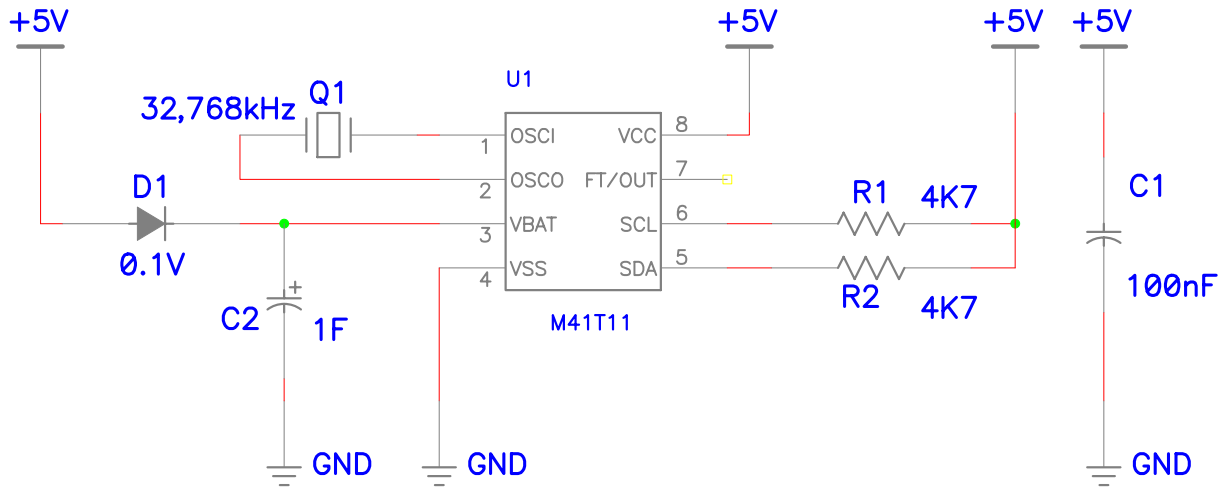


RESET avec modifications :



Mémorisation dans la mémoire RAM sauvegardée

Schéma structurel



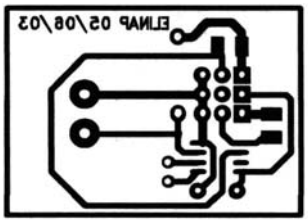
Le circuit intégré M41T11 comporte 8 pattes :

OSCI : entrée pour un quartz
OSCO : sortie pour un quartz
VBAT : entrée d'alimentation pour le mode économie
VCC et VSS : alimentation en +5V et 0V
SDA : Donnée sériel du bus I2C
SCL : Horloge sériel du bus I2C
FT/OUT : permet de tester la fréquence interne pour un éventuel réglage de l'horloge

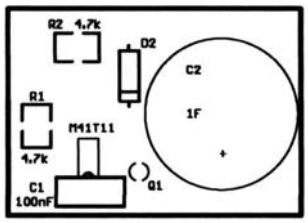
Explications :
Le circuit intégré M41T11 possède une horloge, un calendrier et une mémoire RAM de 56 octets.
Le quartz permet une oscillation très précise qui ne varie pas en fonction de la température, utilisé pour l'horloge et le calendrier au circuit M41T11.
Le condensateur C1 sert de filtrage pour l'alimentation du circuit intégré.
Le condensateur C2 sert de relai en cas de coupure d'alimentation. Quand l'alimentation est en marche, C2 se charge et à la coupure C2 alimente le circuit qui se met en mode économie d'énergie afin de sauvegarder les données en mémoire RAM.
La diode D1 permet au condensateur de se décharger que dans le circuit intégré.
Les résistances R1 et R2 servent de résistances de tirage pour fixer le potentiel à « 1 » lorsque aucune information ne circule sur SDA ou SCL.

Typon

Face cuivre



Implantation



La face cuivre est du même côté que l'implantation.

Nomenclature

Noms :	Référence :	Propriétés :
Résistances : R1, R2	4K7	CMS, ¼ W, boîtier 1210
Condensateurs : C1 C2	100nF 1F	Traversant, 2 pas Traversant, 2 pas, chimique
Diode : D1	1N5819	Traversant, 0,1V chotkky
Quartz : Q1	32,768 kHz	Traversant, axial
Circuit intégré : U1	M41T11	CMS, boîtier SO8
Divers : 2 oeillets		court

Perçage

Les deux trous en dessous du condensateur C2 sont percés en foret de diamètre 1,80 mm.
Les deux trous en dessous du quartz Q1 sont percés en foret de diamètre 0,8 mm.
Le reste est percé à 1,50 mm de diamètre.

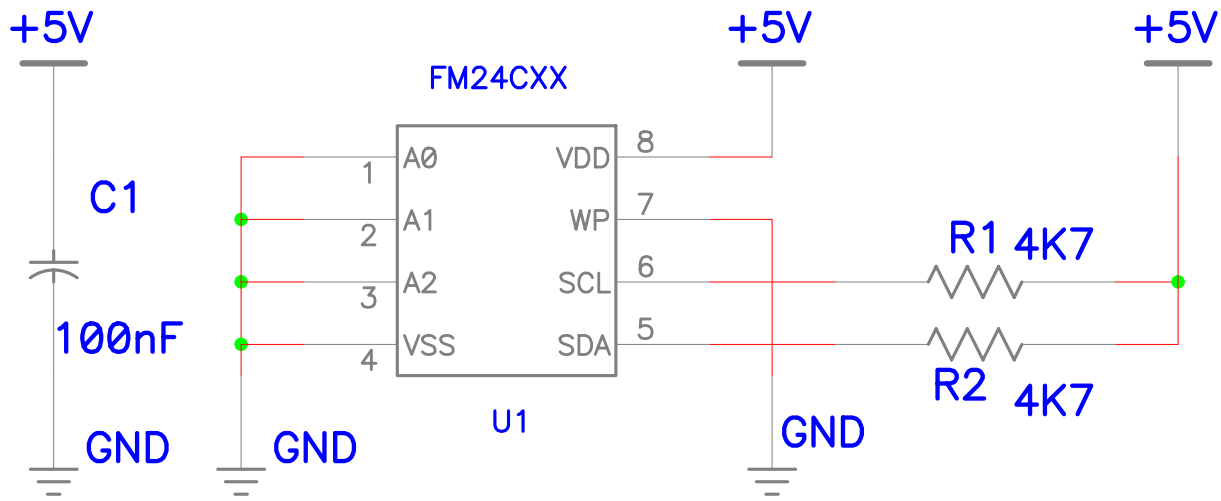
Montage

Une fois le circuit imprimé réalisé et percé, commencer par mettre les œillets dans les trous en dessous du condensateur C2 du côté cuivre. Puis avec un poinçon et un marteau taper de l'autre côté pour refermer les œillets. Ensuite souder sur le côté cuivre les œillets. Par la suite souder tous les autres composants. Une fois la plaque réalisée et testée, finir par l'assembler sur le système de comptage (voir photo) en la mettant sur les neuf broches avec le condensateur vers la droite en regardant par devant.

Mémorisation dans la mémoire FRAM

Aucune modification hardware n'est nécessaire sur le système de comptage si ce n'est l'ajout du circuit ci-dessous sur un connecteur.

Schéma structurel



Le circuit intégré FM24CXX comporte 8 pattes :

A0-A2 : Adresse du circuit intégré
VCC et GND : alimentation en +5V et 0V
SDA : Donnée sériel du bus I2C
SCL : Horloge sériel du bus I2C
WP : Protection en écriture permet de verrouiller l'opération en écriture

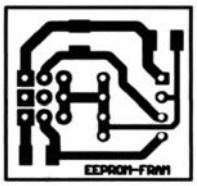
Explications :

Le circuit intégré FM24CXX est une mémoire non volatile à accès par bus I2C de type FRAM. La mémoire FRAM est de technologie ferroélectrique ce qui lui permet des écritures jusqu'à 10 milliards de fois, beaucoup plus conséquente que l'EEPROM. Cette mémoire FRAM peut donc être utilisée pour la mémorisation à l'extinction de l'alimentation ou à chaque fermeture de porte.

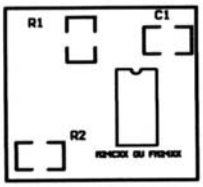
Le circuit intégré FM24CXX a l'adresse 0 sur A0, A1 et A2.
Le condensateur C1 permet le filtrage de l'alimentation de la mémoire.
Les résistances R1 et R2 servent de résistances de tirage pour fixer le potentiel à « 1 » lorsque aucune information ne circule sur SDA ou SCL.

Typons :

Face cuivre



Implantation



La face cuivre est du même côté que l'implantation.

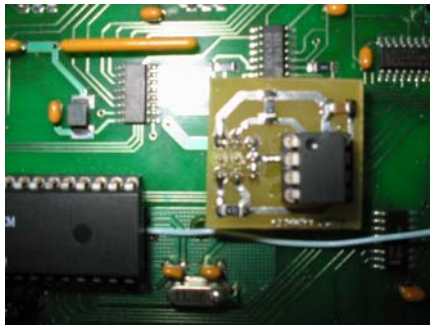
Nomenclature :

Noms :	Références :	Propriétés :
Résistances : R1, R2	4K7	CMS, ¼ W, boîtier 1210
Condensateurs : C1	100 nF	CMS, boîtier 1210
Circuit intégré : FRAM	FM24C64	Traversant, boîtier DIP8
Porte CI		Traversant, boîtier DIP8

Perçage

Le reste est percé à 1,50 mm de diamètre.

Montage



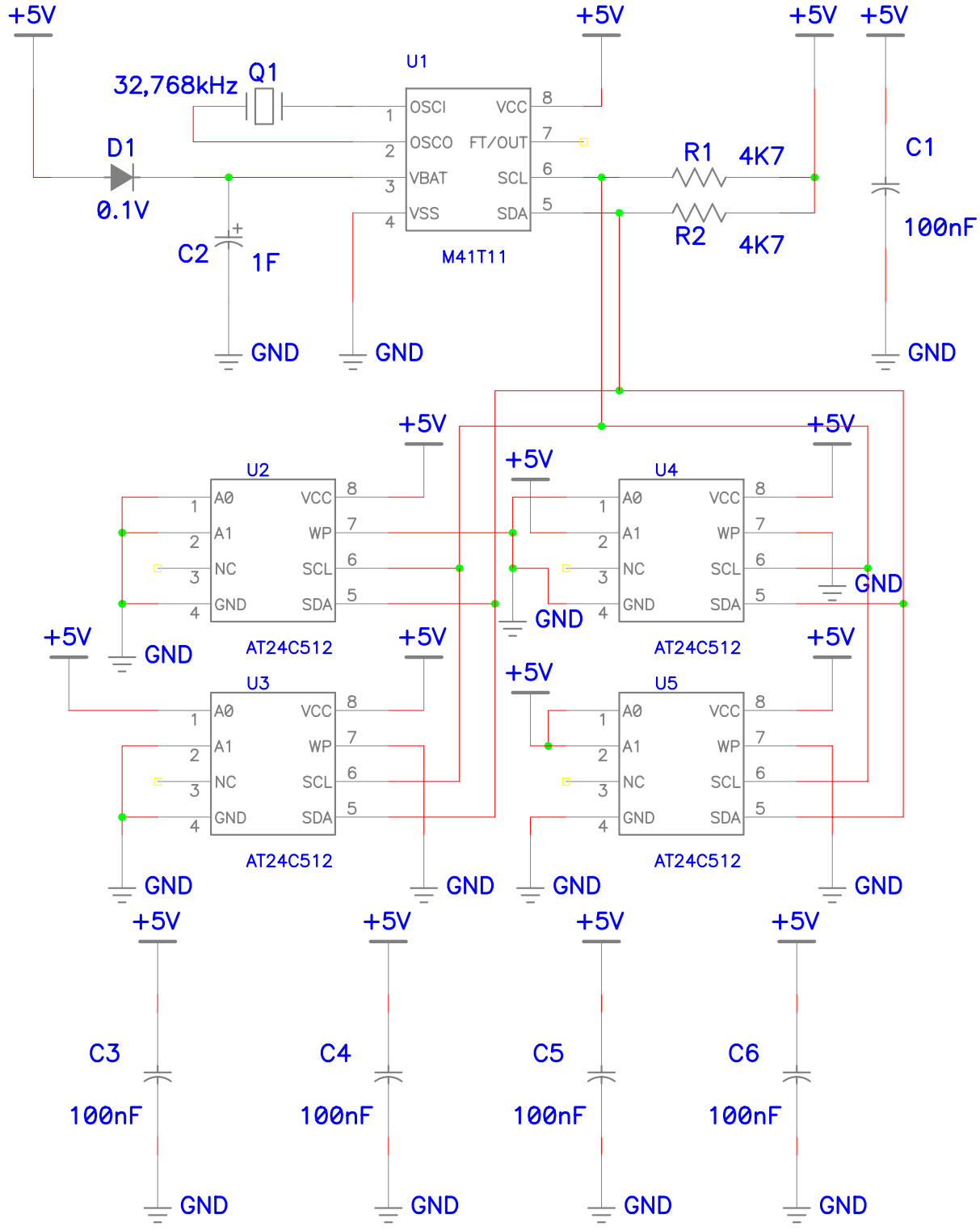
Conclusion

La mémorisation à chaque fermeture des portes, il n'est pas utile de faire des modifications hardwares sur le RESET du système de comptage.
Le système de mémorisation à la coupure de l'alimentation est plus volumineux à cause du condensateur de 1F et légèrement plus coûteux.
Il n'y a qu'un constructeur qui fait de la FRAM, s'il arrête de les fabriquer ou si les grossistes arrêtent de les approvisionner, on sera obligé de changer de technologie d'où l'importance d'avoir différentes méthodes de mémorisation.
La FRAM est non volatile alors que la mémoire RAM est sauvegardée pendant un certain temps.
Il sera donc préférable de choisir la mémorisation à chaque fermeture de porte avec le système qui utilise la mémoire FRAM.

MICHEL V7B avec mémorisation avant coupure de l'alimentation et archivage

Pour archiver des données, il faut impérativement augmenter la capacité mémoire et avoir une horloge et un calendrier.

Schéma structurel



Le circuit intégré AT24C512 comporte 8 pattes :

A0-A1 : Adresse du circuit intégré
NC : Non connecté
VCC et GND : alimentation en +5V et 0V
SDA : Donnée sériel du bus I2C
SCL : Horloge sériel du bus I2C
WP : Protection en écriture permet de verrouiller l'opération en écriture

Explications :

Les circuits intégrés AT24C512, U2 à U5 sont des mémoires EEPROM de 512 kbits chacun à accès série par bus I2C. Ce qui fait en tout 2048 kbits soit 256 koctets de mémoire.
Les circuits intégrés AT24C512 ont toutes des adresses différentes sur A0 et A1.
Les condensateurs C3 à C6 permettent le filtrage de l'alimentation des mémoires.

Conclusion :

Il sera préférable de changer le circuit RESET par un circuit intégré 7705 plus simple, mieux contrôlable et plus flexible.
Le typon devra être refait entièrement car il devra intégrer tous les éléments précédents (Mémoires, horloge, calendrier, RESET) et en plus intégrer une nouvelle alimentation à découpage et un aiguilleur pour liaisons séries RS232 et RS485 et éventuellement un capteur d'accélération et un capteur de comptage d'impulsion lumineuses.

ADAPTATION LOGICIELLE DES PROGRAMMES EXISTANTS

L'adaptation logicielle se porte sur le système de comptage avec mémorisation avant la coupure de l'alimentation. L'adaptation logicielle avec mémorisation et archivage des données nécessite énormément de modifications logicielles, cela remet en cause tout le programme existant. De plus il faut modifier le programme sur ordinateur entièrement réalisé en langage C, qui gère la communication entre le PC et le système de comptage afin de recueillir l'archivage des données.

Adaptation du programme général

Il faut déjà adapter les déclarations de variables :

Les variables de type byte ou word sont réparties soit dans la RAM interne ou soit dans la RAM auxiliaire.
Il faut déclarer de cette façon :
DCL variable byte AUX ;
DCL variable byte;

Erreurs dans le programme :

Au départ, une fois les déclarations adaptées et le programme compilé sur le microcontrôleur ArMEL.
Un problème est survenu sur le figeage de la partie afficheur. En effet, la valeur de départ était affichée et ensuite figée autrement dit, l'affichage ne marchait qu'une seule fois.

Pour résoudre ce problème un petit changement est nécessaire :
Dans l'interruption périodique toutes les 10 ms
Il faut remplacer la ligne suivante
"do cc1 = 0 to longtrame;"
par "do cc1 = 0 to (longtrame - 1);"

Comment j'ai trouvé cette erreur ? :

Au départ, j'ai pensé que cette erreur venait de la communication entre les deux microcontrôleurs. J'ai donc étudié ce protocole par l'intermédiaire de deux programmes, celui du microcontrôleur THOMSON (langage assembleur THOMSON sans commentaire) et celui du microcontrôleur principal (langage PLM sans commentaires). Le protocole de communication fonctionne de la manière suivante, elle marche tout d'abord sur deux fils, une ligne de données bidirectionnelle et une ligne de synchronisation. La ligne de synchronisation est branchée sur l'interruption NMI du microcontrôleur THOMSON, cette interruption est prioritaire et actif sur front descendant. A chaque interruption NMI, le programme du THOMSON lit le bit de la ligne de données et ainsi de suite.

Tout d'abord, dans le cas le plus courant le microcontrôleur principal envoie un octet pour définir le type de données et cinq octets à la suite qui définissent chacun la donnée à afficher de chaque digits et par la suite il réceptionne le checksum (somme des données) venant du microcontrôleur THOMSON. Si le checksum réceptionné est bon, le microcontrôleur principal continue à envoyer les nouvelles données sinon il continue à envoyer les mêmes données. C'est pour ces raisons que j'ai porté mon attention sur le checksum. J'ai donc modifié le programme

pour qu'il affiche le checksum calculé et le checksum reçu en fonction de la commutation d'un interrupteur. J'ai tout de suite vu qu'il y avait un décalage entre les deux checksums. J'ai regardé à l'oscilloscope la communication pour faire la somme moi-même pour trouver la vrai valeur du checksum ensuite j'ai comparé cette vraie valeur avec les checksums affichés.
J'ai tout de suite vu que c'était le checksum calculé qui était mauvais.

Identification de la partie du programme suivante se trouvant vers la fin de la procédure d'interruption toute les 10 ms :

```
do cc1 = 0 to longtrame ;
     checksumctrl = oct(cc1) + checksumctrl;
end;
```

La variable longtrame détermine la longueur de la trame envoyée c'est à dire le nombre d'octet envoyé. Dans le cas le plus courant, longtrame = 6
La variable Checksumctrl est le checksum calculé.
Oct est un tableau où sont stockés les données à envoyer

Dans le cas le plus courant, il y a 6 octets à envoyer : oct(0) à oct(5)
Or dans la partie du programme ci-dessus, cc1 change de 0 à 6 ce qui fait 7 octets.
Il suffit de mettre "do cc1 = 0 to (longtrame - 1);" pour que le programme affiche correctement.

Ce qu'il faut rajouter pour tous les programmes :

```
/* Dans les déclarations */
DCL totall          byte;
DCL totalh          byte;
DCL totalbisl       byte;
DCL totalbish       byte;
```

Les variables de type byte ou word sont réparties soit dans la RAM interne ou soit dans la RAM auxiliaire.
Déclaration de cette façon :

```
DCL variable        byte    AUX ;
DCL variable        byte;
```

```
/* Dans CONFIG: procedure;*/
AUXR=0000$0100b;
TCON = 0101$0101b;
IE = 1000$0110b;
IP = 0000$0110b;
IPH = 0000$0100b;
EA = 1;
EX1 = 1;
IT1 = 0;
```

Il faut rajouter les procédures d'écriture et de lecture dans l'EEPROM appartenant à Monsieur Daniel GUENON (Il travaille à la CTB et nous a fait grâce de ces deux procédures).

De**larations et procédures du bus I2C :**

/* le bus I2C doit être capable de gérer toutes sortes de mémoires, de gérer l'adressage sur 8 ou 16 bits, de gérer le mode page pour augmenter la rapidité et de gérer une horloge et un calendrier. */

```
/*************************************************************************/
/*                    déclaration des constantes du programme            */
/*************************************************************************/


DCL i2c_wr          byte constant (0000$0000b);
DCL i2c_rd          byte constant (0000$0001b);
DCL i2c_eeprom      byte constant (1010$0000b);
DCL i2c_ram         byte constant (1101$0000b);
DCL M24C02          byte constant (1);
DCL FM24C64             byte constant (0);
DCL M24C64          byte constant (0);
DCL M41T11          byte constant (2);

/*************************************************************************/
/* I2C                 déclaration des variables pour le programme       */
/*************************************************************************/

DCL i2c_data_r      byte AUX;
DCL i2c_data_w      byte AUX;
DCL chipselect      byte AUX; /* 0 pour le M24C02, 1 pour le M24C64 et 2 pour le M41T11
*/
DCL i2c_adrhigh     byte AUX;
DCL i2c_adrlow      byte AUX;
DCL var1            byte AUX;
DCL var2            byte AUX;
DCL var3            byte AUX;
DCL var4            byte AUX;
DCL a               byte AUX;


/*************************************************************************/
/* I2C                   déclaration des variables de type bit           */
/*************************************************************************/

DCL i2cack          bit  ;
DCL i                bit  ;
```

```
/*****************************************************************************/
/* I2C                déclaration des adresses des bits E/S                  */
/*****************************************************************************/


DCL sda                 bit   at  (0A3h)    REG; /* P2.3 */
DCL scl                 bit   at  (0A2h)    REG; /* P2.2 */
/*************************/
/*   tempo                 */
/*************************/
/* tempo de 6.36 us */

tempo6us: procedure;
i = 1;
end tempo6us;

/*************************************************/
/* Envoie une séquence 'START' au périphérique I2C          */
/*************************************************/

i2cstart: procedure;

scl = 1;            /* Au début, SCL */
sda = 1;            /* et SDA sont a '1' */
call    tempo6us;   /* on attend 6 us */
sda = 0;            /* puis SDA passe a '0' */
call    tempo6us;   /* puis on attend 6us */
scl = 0;            /* puis SCL passe a '0' */
call    tempo6us;   /* puis on attend 6 us */
sda = 1;            /* SDA doit être remis a '1' pour pouvoir */
                    /* voir les données de la mémoire. */
end i2cstart;

/************************************************/
/*  Envoie une séquence STOP au périphérique I2C          */
/************************************************/

i2cstop: procedure;

scl = 0;            /* Au début SCL /*
sda = 0;            /* et SDA sont a '0' */
call    tempo6us;   /* on attend 6 us */
scl = 1;            /* puis SCL passe a '1' */
call    tempo6us;   /* puis on attend 6 us */
sda = 1;            /* puis SDA passe a '1' */
call    tempo6us;   /* on attend 6 us ... */

end i2cstop;
```

```
/*************************************************************************/
/* Vérifie si le périphérique I2C renvoie un 'Acknowledge' ; si celui-ci */
/* est présent, le bit I2CACK est mis a '1', sinon I2CACK est mis a '0'. */
/*************************************************************************/



i2cgetack: procedure;

sda = 1;                    /* On s'assure que SDA est a '1' */
scl = 0;                    /* On génère l'impulsion positive sur SCL */
call    tempo6us;           /* SCl a '0', attente de 6us */
scl = 1;                    /* SCL passe a '1', attente de 6us */
call    tempo6us;           /* La on va lire la valeur de SDA : */
if sda = 1 then i2cack = 0; /* Si SDA = '1' on met i2cack a '0' */
if sda = 0 then i2cack = 1; /* Si SDA = '0' on met i2cack a '1' */
scl = 0;                    /* puis SCL repasse a '0' */

end i2cgetack;

/*****************************************************/
/*   Envoie un bit 'Acknowledge' au périphérique I2C.  */
/*****************************************************/

i2csetack: procedure;

scl = 0;            /* Au début SCL est a '0' */
sda = 0;            /* SDA est mis a '0' (on applique ACK) */
call tempo6us;      /* on attend 6 us */
scl = 1;            /* SCL passe a '1' */
call tempo6us;      /* pendant 6 us */
scl = 0;            /* puis repasse a '0' */
sda = 1;            /* il faut penser remettre SDA a '1' ! */

end i2csetack;

/*****************************************************/
/ *   Envoie un 'No Acknowledge' au peripherique I2C.  */
/*****************************************************/

i2csetnoack: procedure;

scl = 0;            /* Au debut, SCL est a '0' */
sda = 1;            /* et SDA a '1' */
call    tempo6us;   /* on attend 6 us */
scl = 1;            /* impulsion positive sur SCL */
call    tempo6us;   /* SCL passe a '1' pendant 6 us */
scl = 0;            /* puis repasse a '0' */

end i2csetnoack;
```

```
/*************************************************************/
/*      Lit un octet depuis le périphérique I2C et le met dans a :          */
/*      (Attention, cela modifie les variables VAR1 et VAR2)                */
/*************************************************************/

/* Cette routine ressemble beaucoup a la précédente ; mais au lieu
 de recopier le bit B7 sur SDA et de décaler l'octet vers la gauche
 et ce huit fois de suite, on commence par décaler l'octet, puis on
 recopie SDA sur B0, toujours huit fois de suite. Les bits ne sont
 plus retirés un a un, mais au contraire empilés les uns après les
 autres dans VAR1 */


i2ctoa: procedure;

var2 = 8;/* on met 8 dans var2, variable compteur qui va comptabiliser les 8 décalages */
var1 = 0;                                          /* var1 initialisee a '0' */
do while var2 >0;                                  /* On execute 8 fois */
        scl = 0;                                   /* SCL mis a '0' */
        sda = 1;                                   /* SDA mis a '1' pour être lisible */
        call    tempo6us;                          /* Attente de 6 us */
        scl = 1;                                   /* SCL passe a '1' */
        call    tempo6us;                          /* Attente de 6us */
        var1 = ROL(var1,1);                        /* Décalage de 1 coup vers la gauche */
        var1 = var1 and 1111$1110b;                /* On met le bit 0 de var1 a '0' */
        if sda = 1 then var1 = var1 or 0000$0001b; /* Si SDA est a '1' le bit 0 de var1 a '1' */
        var2 = var2 - 1;                           /* On décrémente le compteur de 1 */
end;
scl = 0;                                           /* après le dernier cycle : SCL a '0' */
a = var1;                                          /* VAR1 (l'octet lu) est copie dans a */

end i2ctoa;

/*************************************************************/
/* Envoie un octet contenu dans a au peripherique I2C en 'serie' :     */
/* (attention, cela modifie les variables VAR1 et VAR2)                */
/*************************************************************/

/* L'octet à envoyer est mis dans VAR1 ; on va alors faire subir 8 fois
 à cet octet un décalage vers la gauche, en recopiant a chaque fois au
 préalable le bit B7 sur la broche SDA (avant d'appliquer une impulsion
 d'horloge sur SCL). La variable VAR2 sert de compteur : elle part de 8
 et est décrémentée 8 fois jusqu'a 0 afin de compter le nombre de
 décalages :*/

atoi2c: procédure;

var1 = a;                                  /* Octet a envoyer mis dans var1 */
var2 = 8;                                  /* 8 (nb de décalages) dans var2 */
```

```
do while var2 >0;                           /* On execute 8 fois */
        scl = 0;                            /* SCL mis a '0' */
        var3 = var1 and 1000$0000b;         /* bit 7 de var1 mis a 1 dans var3 */
        if var3 = 0000$0000b then sda = 0;  /* si le bit 7 de var1 est a '0', SDA est a '0'*/
        if var3 = 1000$0000b then sda = 1;  /* si le bit 7 de var1 est a '1', SDA est a '1'*/
        call    tempo6us;                   /* On applique l'impulsion sur SCL : */
        scl = 1;                            /* attente de 6us, SCL passe a '1' */
        call    tempo6us;                   /* attente de 6us, SCL passe a '0' */
        scl = 0;
        var1 = ROL(var1,1);                 /* on décale un coup var1 a gauche */
        var2 = var2 - 1;                    /* on décrémente le compteur */
end;
sda = 1;        /* On remet SDA a '1' (ce qui permet de scruter SDA comme une entrée) */

end atoi2c;

/*****************************************************************/
/* Test si l'eeprom est disponible                               */
/* On boucle tant qu'on ne reçoit pas un ACK  s'est a dire       */
/* on boucle tant que i2cack = 0                                 */
/*****************************************************************/

test_eeprom: procedure;

do while i2cack = 0;
        call    i2cstart;
if chipselect = M41T11 then
        a = i2c_ram or i2c_wr;
else
        a = (ROL (chipselect,1) and 00001110b) or i2c_eeprom or i2c_wr;
        a = i2c_ram or i2c_wr;
        call    atoi2c;
        call    i2cgetack;
end;

end test_eeprom;

/*****************************************************************************/
/* Dans l'EEPROM I2C, pointe l'adresse mémoire voulue :                      */
/* (Attention, cela modifie les variables VAR1 et VAR2)                      */
/*****************************************************************************/

set_adr_i2c: procedure;

call    i2cstart;                           /* 'start' : Debut de sequence */
if chipselect = M41T11 then                 /* si on selectionne la RAM M41T11 : */
        a = i2c_ram or i2c_wr;              /* a = 1101$0000b */
else    /* sinon a = 1010$xxx0b (xxx étant les adresses des eeproms définit par chipselect) */
        a = (ROL (chipselect,1) and 00001110b) or i2c_eeprom or i2c_wr;
```

```
call atoi2c;                                    /* on envoie a sur le bus i2c */
call i2cgetack;                                 /* on vérifie si l'octet a bien été reçu */
if i2cack = 1 then do;                          /* si oui, on continue */
        a = i2c_adrhigh;                        /* on envoie l'adresse de poids fort */
        call    atoi2c;                         /* sur le bus i2c */
        call    i2cgetack;                      /* on vérifie si l'octet a bien été reçu */
        if i2cack = 1 then do;                  /* si oui on continu */
                if chipselect = (M24C64 or FM24C64) then do;
                                /* si on sélectionne l'eeprom M24C64 (16 bits d'adresse) */
                        a = i2c_adrlow;         /* on envoie l'adresse de poids faible */
                        call    atoi2c;         /* sur le bus i2c */
                        call    i2cgetack;      /* on verifie si l'octet a bien ete recu */
                end;
        end;
end;

end set_adr_i2c;




/*************************************************************************/
/* Lit l'octet situe dans l'EEPROM I2C et met le contenu dans [I2C_data_r].  */
/* (Attention, cela modifie les variables VAR1, VAR2)                        */
/*************************************************************************/


readmemoire: procedure;

i2cack = 0;     /* on met i2cack a '0' pour rentrer au moins une fois dans la boucle do while */
var4 = 3;                                       /* on definit a 3 fois maximum le bouclage quand on
ne recoit pas d'ACK */
do while  (var4 > 0) and (i2cack = 0) ;         /* on boucle 3 fois au maximum et tant que i2cack =
'0' */
        call    set_adr_i2c;                    /* pointer l'adresse */
        if i2cack = 1 then do;                  /* si i2ack = 1 on continue */
                call    i2cstart;               /* envoyer un ordre de lecture */
                if chipselect = M41T11 then /* si on selectionne la RAM M41T11 : */
                        a = i2c_ram or i2c_rd;/* a = 1101$0000b */
                else
/* sinon a = 1010$xxx0b (xxx étant les adresses des eeproms définit par chipselect) */
                        a = (ROL (chipselect,1) and 00001110b) or i2c_eeprom or i2c_rd;
                call    atoi2c;                 /* on envoie a sur le bus i2c */
                call    i2cgetack;              /* on vérifie si l'octet a bien été reçu */
                if i2cack = 1 then do;          /* si oui, on continue */
                        call i2ctoa;            /* on lit l'octet reçu */
                        i2c_data_r = a;                 /* on le met dans i2c_data_r */
                        call i2csetnoack;       /* on envoie un noack (sda a '1'), pour dire qu'on a
finie la lecture */
                end;
```

```
        end;
        call    i2cstop;                    /* Stop */
        var4 = var4 -1;                         /* On decremente var4 de 1 */
end;

end readmemoire;

/*****************************************************************************/
/* Ecrit l'octet situe dans [I2C_data_w] dans l'EEPROM I2C.                  */
/* (Attention, cela modifie les variables VAR1, VAR2)                        */
/*****************************************************************************/


writememoire: procedure;

i2cack = 0;     /* on met i2cack a '0' pour rentrer au moins une fois dans la boucle do while */
var4 = 3;                                   /* on définit a 3 fois maximum le bouclage quand on
ne recoit pas d'ACK */
do while  (var4 > 0) and (i2cack = 0) ;     /* on boucle 3 fois au maximum et tant que i2cack =
'0' */
        call    set_adr_i2c;                /* pointer l'adresse */
        if i2cack = 1 then do;              /* si i2ack = 1 on continue */
                a = i2c_data_w;             /* on ecrit la donnee i2c_data_w */
                call    atoi2c;             /* sur le bus i2c */
                call    i2cgetack;          /* on verifie si l'octet a bien ete recu */
        end;
        call    i2cstop;                    /* Stop */
        var4 = var4 -1;                     /* On decremente var4 de 1 */
        if i2cack = 0 then call time(100);  /* on attend 10ms avant de réécrire */
end;

end writememoire;


/*****************************************************************************/
/* Ce sous-programme permet d'initialiser l'horloge et le calendrier de la ram M41T11    */
/*****************************************************************************/

init_ram_clk: procedure;

chipselect = M41T11;        /* on n'oublie pas de sélectionner la RAM M41T11 */
i2c_adrhigh = 0;            /* on commence par l'adresse 0 (premier registre) */

/*i2c_data_w = 1000$0000b;
call writememoire;*/
/*i2c_data_w = 0000$0000b;
call writememoire;*/


call set_adr_i2c;           /* on pointe l'adresse */
```

```
a = 0000$0011b;             /* adresse 0 -> ST, 10 seconds $ seconds */
call    atoi2c;
call    i2cgetack;

a = 09h;                    /* adresse 1 -> 10 minutes $ minutes*/
call    atoi2c;
call    i2cgetack;

a = 16h;                    /* adresse 2 -> CEB, CB, 10 hours $ hours */
call    atoi2c;
call    i2cgetack;

a = 0000$0001b;             /* adresse 3 -> x $ day */
call    atoi2c;
call    i2cgetack;

a = 05h;                    /* adresse 4 -> 10 date $ date */
call    atoi2c;
call    i2cgetack;

a = 05h;                    /* adresse 5 -> 10 month $ month */
call    atoi2c;
call    i2cgetack;

a = 03h;                    /* adresse 6 -> 10 years $ years */
call    atoi2c;
call    i2cgetack;

a = 0000$0000b;             /* adresse 7 -> OUT, FT, S, Calibration */
call    atoi2c;
call    i2cgetack;

call    i2cstop;

end init_ram_clk;


/* procédure de lecture en mode page */

read_page: procedure byte;
if i2cack = 1 then do;
        call i2ctoa;
        call i2csetack;
end;
return a;
end read_page;


/* procédure de lecture en mode page pour le dernier (noack) */
```

```
read_page_end: procedure byte;
if i2cack = 1 then do;
        call i2ctoa;
        call i2csetnoack;
end;
return a;
end read_page_end;


/* procédure d'écriture en mode page */

write_page: procedure(variable);
DCL variable byte;
if i2cack = 1 then do;
        a = variable;
        call    atoi2c;
        call    i2cgetack;
end;
end write_page;
```

Programme pour enregistrer à la coupure de l'alimentation :

SRAM

```
SAUVEGARDE: procedure interrupt 2;


chipselect = M41T11;
i2c_adrhigh = 10;


i2cack = 0;
var4 = 3;
do while  (var4 > 0) and (i2cack = 0) ;
        call set_adr_i2c;
        if i2cack = 1 then do;
                call write_page (low(total));
                call write_page (high(total));
                call write_page (low(totalbis));
                call write_page (high(totalbis));
                call write_page (pointenvmd);
                call write_page (envmd(0));
                call write_page (envmd(1));
                call write_page (envmd(2));
                call write_page (envmd(3));
                call write_page (envmd(4));
                call write_page (envmd(5));
                call write_page (envmd(6));
                call write_page (envmd(7));
```

```
                call write_page (envmd(8));
                call write_page (envmd(9));
                call write_page (envmd(10));
                call write_page (envmd(11));
                call write_page (envmd(12));
                call write_page (envmd(13));
                call write_page (envmd(14));
                call write_page (envmd(15));
                call write_page (envmd(16));
                call write_page (envmd(17));
                call write_page (envmd(18));
                call write_page (envmd(19));
                call write_page (envmd(20));
                call write_page (envmd(21));
                call write_page (envmd(22));
                call write_page (envmd(23));
                call write_page (envmd(24));
                call write_page (envmd(25));
                call write_page (envmd(26));
                call write_page (envmd(27));
                call write_page (envmd(28));
                call write_page (envmd(29));
                call write_page (envmd(30));
                call write_page (envmd(31));
                call write_page (envmd(32));
        end;
        call    i2cstop;
        var4 = var4 -1;
        if i2cack = 0 then call time(100); /* on attend 10ms avant de réécrire */
end;

END SAUVEGARDE;



LECTURE: procedure;

chipselect = M41T11;
i2c_adrhigh = 10;

i2cack = 0;
var4 = 3;
do while  (var4 > 0) and (i2cack = 0) ;
        call    set_adr_i2c;
        if i2cack = 1 then do;
                call    i2cstart;
                a = i2c_ram or i2c_rd;
                call    atoi2c;
                call    i2cgetack;
                totall = read_page;
```

```
                totalh = read_page;
                totalbisl = read_page;
                totalbish = read_page;
                pointenvmd = read_page;
                envmd(0) = read_page;
                envmd(1) = read_page;
                envmd(2) = read_page;
                envmd(3) = read_page;
                envmd(4) = read_page;
                envmd(5) = read_page;
                envmd(6) = read_page;
                envmd(7) = read_page;
                envmd(8) = read_page;
                envmd(9) = read_page;
                envmd(10) = read_page;
                envmd(11) = read_page;
                envmd(12) = read_page;
                envmd(13) = read_page;
                envmd(14) = read_page;
                envmd(15) = read_page;
                envmd(16) = read_page;
                envmd(17) = read_page;
                envmd(18) = read_page;
                envmd(19) = read_page;
                envmd(20) = read_page;
                envmd(21) = read_page;
                envmd(22) = read_page;
                envmd(23) = read_page;
                envmd(24) = read_page;
                envmd(25) = read_page;
                envmd(26) = read_page;
                envmd(27) = read_page;
                envmd(28) = read_page;
                envmd(29) = read_page;
                envmd(30) = read_page;
                envmd(31) = read_page;
                envmd(32) = read_page_end;

        end;
        call    i2cstop;
        var4 = var4 -1;
end;

conf(0) = READ_EE (0);
conf(1) = READ_EE (1);
conf(2) = READ_EE (2);
conf(3) = READ_EE (3);
conf(4) = READ_EE (4);
conf(5) = READ_EE (5);
configh = READ_EE (6);
```

```
configl = READ_EE (7);
divaffimp = READ_EE (8);
olddiv = READ_EE (9);
typecel = READ_EE (10);
oldtypecel = READ_EE (11);

total = totalh*256 + totall;
totalbis = totalbish*256 + totalbisl;

END LECTURE;
```

FRAM

Mêmes procédures que précédemment en changeant seulement les parties de programmes en rouge comme ci-dessous.

```
SAUVEGARDE: procedure interrupt 2;

chipselect = FM24C64;
i2c_adrhigh = 0;
i2c_adrlow = 0;

END SAUVEGARDE;



LECTURE: procedure;

chipselect = FM24C64;
i2c_adrhigh = 0;
i2c_adrlow = 0;

i2cack = 0;
var4 = 3;
do while  (var4 > 0) and (i2cack = 0) ;
        call      set_adr_i2c;
        if i2cack = 1 then do;
                call      i2cstart;
                a = i2c_eeprom or i2c_rd;

END LECTURE;
```

Programme pour enregistrer à chaque fermeture de porte :

L'enregistrement à chaque fermeture des portes se fait en fonction de la configuration automatique du système installé.

Procédure commune :
gestionsauv: procedure;

```
if (conf(1) and 128) = 128 then do;   /* FC1 present */
        if fc1 = 1   then do;
                if (vuouverture and 1) = 0 then vuouverture = vuouverture or 1;
        end;
        if fc1 = 0  then do;
                if (vuouverture and 1) = 1 then do;
                        vuouverture  = vuouverture and not(1);
                        enregistrement = 1;
                end;
        end;
end;

if (conf(1) and 16) = 16 then  do;      /* FC2 present */
        if fc2 = 1   then do;
                if (vuouverture and 2) = 0 then vuouverture = vuouverture or 2;
        end;
        if fc2 = 0  then do;
                if (vuouverture and 2) = 2 then do;
                        vuouverture  = vuouverture and not(2);
                        enregistrement = 1;
                end;
        end;
end;


if (conf(2) and 128) = 128 then do;             /* FC3 present */
        if fc3 = 1   then do;
                if (vuouverture and 4) = 0 then vuouverture = vuouverture or 4;
        end;
        if fc3 = 0  then do;
                if (vuouverture and 4) = 4 then do;
                        vuouverture  = vuouverture and not(4);
                        enregistrement = 1;
                end;
        end;
end;

if (conf(3) and 128) = 128 then do;             /* FC4 present */
        if fc4 = 1   then do;
                if (vuouverture and 8) = 0 then vuouverture = vuouverture or 8;
```

```
        end;
        if fc4 = 0  then do;
                if (vuouverture and 8) = 8 then do;
                        vuouverture  = vuouverture and not(8);
                        enregistrement = 1;
                end;
        end;
end;

if (conf(4) and 128) = 128 then do;             /* FC5 present */
        if fc5 = 1   then do;
                if (vuouverture and 16) = 0 then vuouverture = vuouverture or 16;
        end;
        if fc5 = 0  then do;
                if (vuouverture and 16) = 16 then do;
                        vuouverture  = vuouverture and not(16);
                        enregistrement = 1;
                end;
        end;
end;


if enregistrement = 1 then do; /* on enregistre 10 s après une fermeture des portes*/
        if dixsec > 1000 then do;
                dixsec = 0;
                call sauvegarde;
                enregistrement = 0;
        end;
end;

end gestionsauv;
```

SRAM

```
SAUVEGARDE: procedure;


chipselect = M41T11;
i2c_adrhigh = 10;


i2cack = 0;
var4 = 3;
do while  (var4 > 0) and (i2cack = 0) ;
        call set_adr_i2c;
        if i2cack = 1 then do;
                call write_page (low(total));
                call write_page (high(total));
                call write_page (low(totalbis));
```

```
                call write_page (high(totalbis));
                call write_page (pointenvmd);
                call write_page (envmd(0));
                call write_page (envmd(1));
                call write_page (envmd(2));
                call write_page (envmd(3));
                call write_page (envmd(4));
                call write_page (envmd(5));
                call write_page (envmd(6));
                call write_page (envmd(7));
                call write_page (envmd(8));
                call write_page (envmd(9));
                call write_page (envmd(10));
                call write_page (envmd(11));
                call write_page (envmd(12));
                call write_page (envmd(13));
                call write_page (envmd(14));
                call write_page (envmd(15));
                call write_page (envmd(16));
                call write_page (envmd(17));
                call write_page (envmd(18));
                call write_page (envmd(19));
                call write_page (envmd(20));
                call write_page (envmd(21));
                call write_page (envmd(22));
                call write_page (envmd(23));
                call write_page (envmd(24));
                call write_page (envmd(25));
                call write_page (envmd(26));
                call write_page (envmd(27));
                call write_page (envmd(28));
                call write_page (envmd(29));
                call write_page (envmd(30));
                call write_page (envmd(31));
                call write_page (envmd(32));
        end;
        call    i2cstop;
        var4 = var4 -1;
        if i2cack = 0 then call time(100); /* on attend 10ms avant de réécrire */
end;

END SAUVEGARDE;

LECTURE: procedure;

chipselect = M41T11;
i2c_adrhigh = 10;

i2cack = 0;
var4 = 3;
```

```
do while  (var4 > 0) and (i2cack = 0) ;
        call     set_adr_i2c;
        if i2cack = 1 then do;
                call     i2cstart;
                a = i2c_ram or i2c_rd;
                call     atoi2c;
                call     i2cgetack;
                totall = read_page;
                totalh = read_page;
                totalbisl = read_page;
                totalbish = read_page;
                pointenvmd = read_page;
                envmd(0) = read_page;
                envmd(1) = read_page;
                envmd(2) = read_page;
                envmd(3) = read_page;
                envmd(4) = read_page;
                envmd(5) = read_page;
                envmd(6) = read_page;
                envmd(7) = read_page;
                envmd(8) = read_page;
                envmd(9) = read_page;
                envmd(10) = read_page;
                envmd(11) = read_page;
                envmd(12) = read_page;
                envmd(13) = read_page;
                envmd(14) = read_page;
                envmd(15) = read_page;
                envmd(16) = read_page;
                envmd(17) = read_page;
                envmd(18) = read_page;
                envmd(19) = read_page;
                envmd(20) = read_page;
                envmd(21) = read_page;
                envmd(22) = read_page;
                envmd(23) = read_page;
                envmd(24) = read_page;
                envmd(25) = read_page;
                envmd(26) = read_page;
                envmd(27) = read_page;
                envmd(28) = read_page;
                envmd(29) = read_page;
                envmd(30) = read_page;
                envmd(31) = read_page;
                envmd(32) = read_page_end;

        end;
        call     i2cstop;
        var4 = var4 -1;
end;
```

```
conf(0) = READ_EE (0);
conf(1) = READ_EE (1);
conf(2) = READ_EE (2);
conf(3) = READ_EE (3);
conf(4) = READ_EE (4);
conf(5) = READ_EE (5);
configh = READ_EE (6);
configl = READ_EE (7);
divaffimp = READ_EE (8);
olddiv = READ_EE (9);
typecel = READ_EE (10);
oldtypecel = READ_EE (11);

total = totalh*256 + totall;
totalbis = totalbish*256 + totalbisl;

END LECTURE;
```

FRAM

Mêmes procédures que précédemment en changeant seulement les parties de programmes en rouge comme ci-dessous.

```
SAUVEGARDE: procedure;

chipselect = FM24C64;
i2c_adrhigh = 0;
i2c_adrlow = 0;

END SAUVEGARDE;

LECTURE: procedure;

chipselect = FM24C64;
i2c_adrhigh = 0;
i2c_adrlow = 0;

i2cack = 0;
var4 = 3;
do while  (var4 > 0) and (i2cack = 0) ;
        call     set_adr_i2c;
        if i2cack = 1 then do;
                call     i2cstart;
                a = i2c_eeprom or i2c_rd;
END LECTURE;
```

Ce qu'il reste à faire.

Une petite erreur d'affichage est à identifier dans la procédure de configuration automatique.
Adapter les programmes pour le microcontrôleur P89C51RD2 de PHILIPS.
Ecrire dans la mémoire flash.
L'adaptation logicielle pour le système de comptage avec mémorisation avant la coupure de l'alimentation et archivage des données.
Interfacer deux capteurs spécialisés :
- capteur d'accélération ;
- capteur de comptage d'impulsion lumineuse

CONCLUSION

Ce stage m'a permis de me confronter au monde du travail, d'appliquer mes connaissances et d'apprendre d'avantage.
J'ai pris conscience de l'importance de la documentation, commentaires des programmes, de l'organisation du travail, de l'anglais et de la communication entre collègues de travail.

Ce stage m'a apporté des connaissances solides et sérieuses

- en gestion de mémoire
- en débuggage de programme
- en recherche d'explications
- en recherche sur les documentations rédigées en Anglais
- en capacité d'analyse
- en langage de programmation PLM
- en compréhension de programmes sans commentaires
- en programmation pour tester les nouvelles procédures et pour analyser les programmes

De part les difficultés rencontrées, ce stage était principalement de la recherche, de la compréhension et de la programmation ; En jonglant sur la compréhension de programmes et la confrontation de la théorie à la pratique.