

Page de garde

**DUBOIS** Lucien  **BRENET** Kévin

**DUT 2009**

**La maison domotique**

Projet Cartes à puce



Nous tenons à remercier ici plusieurs personnes qui nous ont aidées durant la réalisation de notre projet :

* Tout d’abord nos professeurs d’Etudes et Réalisations, nous avons cité Mr Hochmuth, Mr Gallien et notre professeur d’études et réalisations de première année pour leurs conseils avérés, leur patience et leurs diverses explications.
* Puis nous souhaitons remercier notre professeur de Culture et Communication qui nous à permis de réaliser un compte rendu digne de ce nom.
* Pour finir nous tenons à remercier tous les créateurs des sites internet que nous avons consultés et les auteurs des différentes sources, documentations utilisées pour l’avancement de notre projet...

Page 2 - Remerciements

1. **Introduction** …………………………………………..……….…………..................................... Page 4
2. **Cahier des charges** ........................................................................................................... Page 5
3. **Analyse fonctionnelle** ……………………….……………….…………………………….... Page 6 à 8

Diagramme FAST ………………………………………………….……………………………….... Page 6 à 7

Diagramme GANTT ………….………………………………………………………………………. Page 8

1. **Etude théorique** …………………………………………………………………........................ Page 9 à 24

Généralités ………………………………………………………………………………………………. Page 9 à 11

La carte GOLD …………………………………….………………………………………………….... Page 12 à 13

Dialogue avec la carte …………………………………………...…………………… ;;………….. Page 14 à 18

Lecteur phœnix ………………………………………………...……………………………………... Page 19 à 24

1. **Réalisation du Lecteur phœnix** …………………………………………………….. … Page 25 à 29

Réalisation du lecteur ………………………………………………………………………………. . Page 25 à 26

Routage ……………….…………………………………………………………………………………. Page 27

Tests…………………………………………………………………..…………………………………... Page 28 à 29

1. **Programmation carte GOLD** ………..……………………………………………..… Page 30 🡪 annexes
2. **Annexes**…..…………….…………………..………………………………………………………. Page suivantes

**Projet initial** : La maison domotique

**Groupe de travail** : LangRomain, Azaïs Jonathan, Kohler Pierre, Brenet Kévin, Dubois Lucien

**Objectif**: Améliorer une maquette de maison domotique et l’ensemble de ces fonctionnalités.

Nous avions pour mission de reprendre un projet déjà effectué lors des années précédentes et de l’améliorer fortement. Pour cela il nous a été proposé plusieurs sous projets que nous allons décrire brièvement ci-dessous :

* **Panneau solaire** : Nous devions implémenter un panneau solaire qui permettrait d’alimenter et d’économiser du courant pour la maison.
* **Capteur de température** : Le capteur de température devait analyser la température extérieure et gérer en fonction de celle-ci la fermeture automatique des volets de la maison, le chauffage ou autres.
* **Une éolienne** : De la même façon que le panneau solaire, l’éolienne devait permettre une économie de consommation de courant de la maison.
* **Un contrôle d’accès par carte à puce** : Nous devions implémenter un contrôle d’accès par carte à puce à la maison. Pour cela nous devions enregistrer des données sur une carte à puce, d’où le besoin de créer un lecteur de cartes à puce qui joue le rôle d’interface entre nous et la carte à puce. Un mot de passe pourrait alors être préenregistrer et la maison ne s’ouvrirait que si la carte à puce contient le mot de passe adéquat.

**Notre projet** 🡪 Le contrôle d’accès par carte à puce ?

Notre binôme, Kévin et Lucien avons finalement choisi de travailler sur le contrôle d’accès par carte à puce.

Ce projet nous a séduits dans un premier temps car chacun de nous utilise la carte à puce (notamment la carte bleue) régulièrement sans forcément en comprendre le fonctionnement et toutes les facettes exploitables de ce support.

Page 4 - Introduction

Nous avons réalisé notre analyse fonctionnelle à l’aide de deux précieux diagrammes très souvent utilisés voire indispensables dans les entreprises pour permettre de planifier efficacement les objectifs, les besoins, etc.… Les deux diagrammes présents ici sont les diagrammes F.A.S.T et GANTT.

* Un diagramme FAST (Functional Analysis System Technique) présente une traduction rigoureuse de chacune des fonctions de service en fonction(s) technique(s), puis matériellement en solution(s) constructive(s). Le diagramme FAST se construit de gauche à droite, dans une logique du pourquoi au comment.
* Le diagramme de Gantt est un outil très rependu pour la gestion d’un projet. Il permet de déterminer les dates de réalisation d’un projet, d’identifier les marges existantes sur certaines tâches, de visualiser le retard ou l’avancement des différents travaux. Il permet de représenter graphiquement la progression du projet. Cet outil répond à un objectif : planifier de façon optimale un planning établi.

**Diagramme FAST :**

*Fonctions Contraintes*

*Solutions techniques*

Lecteur et interface Phoenix

PIC ou Ordinateur et Programme en C

Décoder la mémoire de la carte

Utiliser un programme récurrent

Lire une carte à puce au format ISO

Analyse fonctionnelle - Diagramme FAST

Détecter la présence d’une carte

Contrôler l’accès à la maison

FP1

FP2

FP4

FP5

FP3

Page 6 – Diagramme FAST

**FP1** : Contrôler l’accès à la maison : Nous permettre un contrôle d’accès de la maison d’un ou plusieurs utilisateurs au préalablement prédéfinis en utilisant une ou plusieurs cartes à puce.

**FP2** : Détecter la présence d’une carte : Pour contrôler l’accès de la maison, la ou les cartes à puce employées devront bien évidemment être détectées d’une manière ou d’une autre…

**FP3** : Lire une carte à puce au format ISO : Dans un second temps, la carte à puce au préalablement détecté, il faudra alors interpréter la ou les données internes de la carte sachant que la carte employée sera au format ISO.

**FP4** : Utiliser un programme récurrent : Pour la détection de carte, nous utiliserons un programme récurrent qui permettra en temps réel de savoir si une carte est insérée ou non.

**FP5** : Décoder la mémoire de la carte : Pour permettre ou non l’accès à la maison, nous utiliserons une interface ‘’phœnix’’ qui permettra la lecture de la mémoire EEPROM contenue dans la puce de la carte.

Page 7 – Diagramme FAST

* **Qu’est ce qu’une carte à puce***?*

Une carte à puce est une carte en matière plastique, de quelques centimètres de coté et de moins d’un millimètre d’épaisseur, portant au moins un circuit intégré capable de contenir de l’information. Le circuit intégré (la puce) peut contenir un microprocesseur capable de traiter l’information, ou être limité à des circuits de mémoire non volatile et, éventuellement, un composant de sécurité (carte mémoire). Les cartes à puce sont utilisées dans les moyens de paiement, les preuves d’abonnement, les cartes SIM pour téléphones mobiles, ou bien encore comme moyens d’identification personnelle comme nous allons le voir ici.

* **Notions de normes***:*

Pour définir une carte à puce, il faut au moins normaliser trois types de paramètres. Cette caractérisation se fait par des normes internationales ISO.

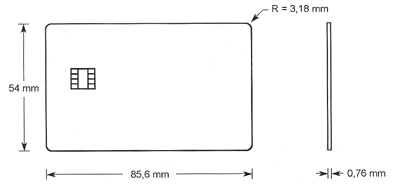
En ce qui concerne la carte à puce, quatre normes principales sont à extraire :

🡪 La norme ISO 7816-1 précisant les caractéristiques physiques, mécaniques de la carte.  
 🡪 La norme ISO 7816-2 définissant la position et le brochage des contacts de la carte à puce.  
 🡪 La norme ISO7816-3 définissant les niveaux électriques et les chronogrammes de bas niveau qui régissent le dialogue avec la carte à puce.   
 🡪 La norme ISO 7816-4 définissant elle les différentes commandes de base de la carte à puce. Elle permet de comprendre le dialogue entre une carte à puce et son lecteur.

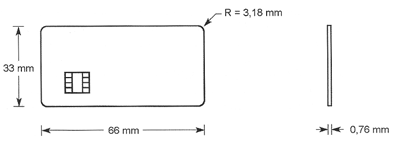
* **Dimensions***:*

Il existe exactement trois dimensions normalisées de cartes à puce appelées :

🡪 Le format ID 1   
🡪 Le format ID 00   
🡪 Le format ID 000

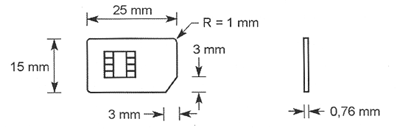


Format de cartes ID1



Page 9 - Généralités

Format de cartes ID00



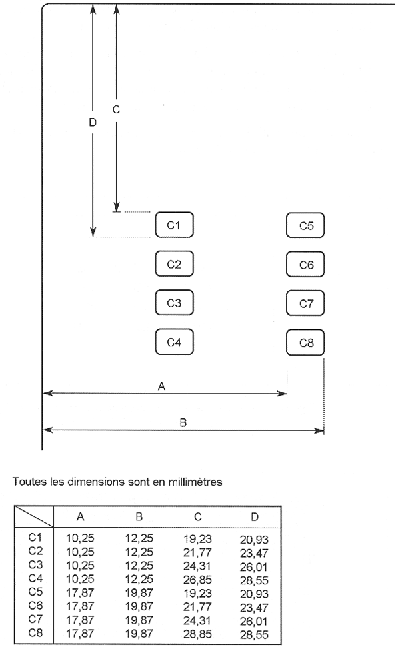
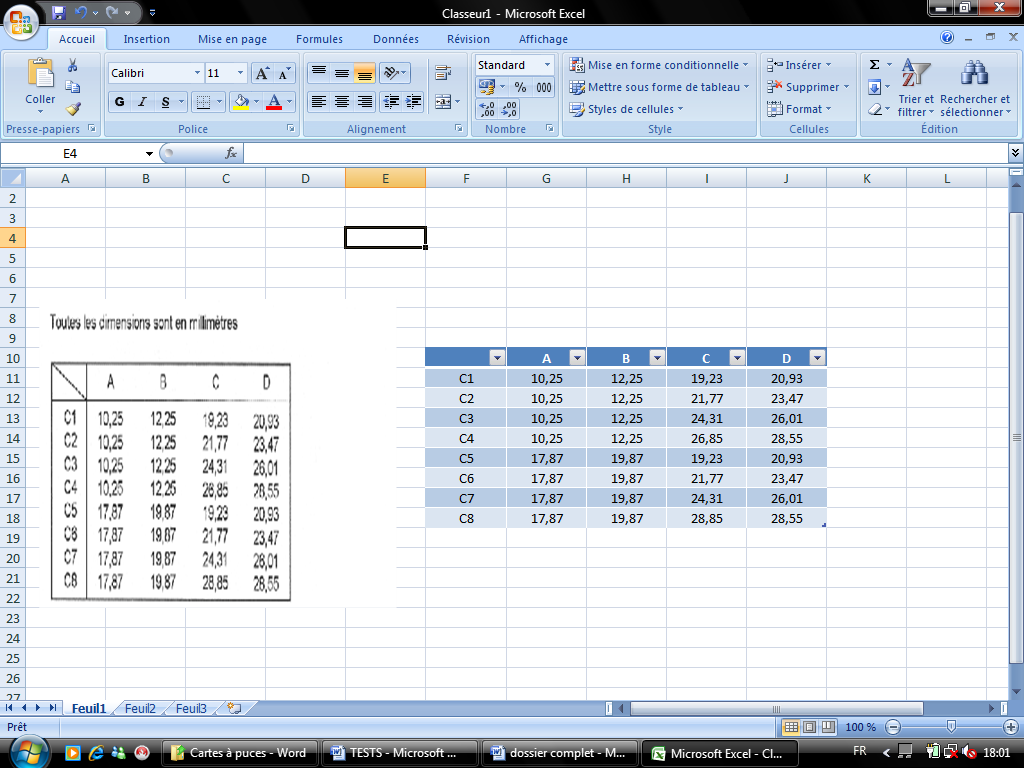
Format de cartes ID000

* **Position des contacts** :

La position des contacts est elle aussi normalisée ISO afin de pouvoir lire n’importe quelle carte dans n’importe quelle lecteur. La norme ISO donne la position précise de chaque connecteur électrique de la carte.

Il existe exactement 8 contacts

NB*: Toutes les valeurs sont en millimètres…*

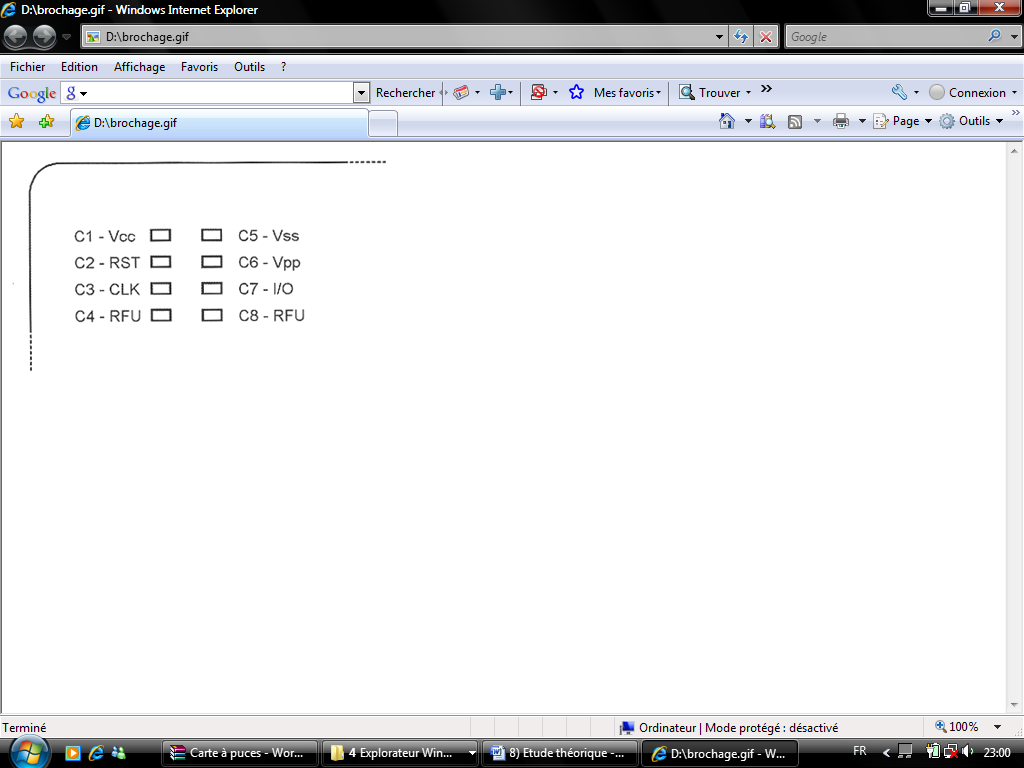


Page 10 - Généralités

* **Brochage** :

Les 8 contacts sont définis dans la norme 7816-3. Nous allons donner ici la fonction de chaque contact :

* Contact 1 : Tension d’alimentation positive de la carte normalisée **VCC**, fournie par le lecteur.
* Contact 2 : Commande de RESET de la carte normalisée **RST**. Entrée non obligatoire pour certaine carte à mémoire.
* Contact 3 : Horloge **CLK** fournie à la carte par le lecteur. Elle définit le rythme des échanges entre la carte et son lecteur.
* Contact 4 & 8: contacts normalisés **RFU** (Reserved for Future Use), c’est à dire que ce sont des contacts réservés pour de futures utilisations.
* Contact 5 : Masse électrique de la carte **VSS**.
* Contact 6 : contact qui n’est plus utilisé.
* Contact 7 : Correspond aux entrées et sorties de données en provenance ou à destination de la carte. Cette ligne est bidirectionnelle et s’appelle **I/O**.



Une carte à puce

Brochage d’une puce

Après cette courte étude sur les cartes à puce, nous allons nous intéressé à un type de carte particulier dénommé ‘’carte gold’’. Nous allons tenter d’expliquer le fonctionnement de la dite carte, ainsi que ces principales fonctionnalités.

L’étude d’une carte gold est très intéressante dans le sens ou la plupart des cartes à puce fonctionnent de la même manière.

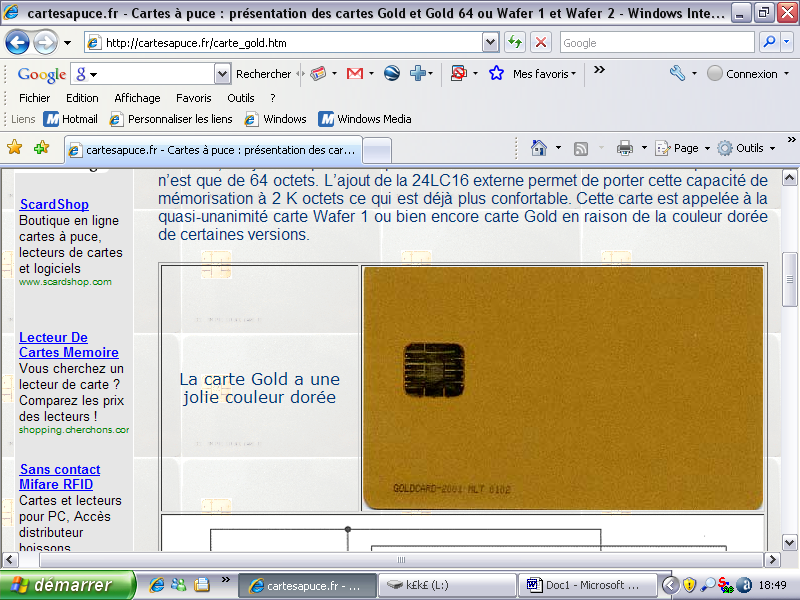
Page 11 - Généralités

Il existe deux différents types de cartes qui respectent la norme 7816 mais leurs possibilités sont bien distinctes :

* Les cartes à mémoire (synchrones) : C’est la famille la plus ancienne. Cette carte ne contient que de la mémoire, elles ne sont plus utilisées à l’heure actuelle.
* Les cartes à microcontrôleurs (asynchrones) : On les appelle aussi Smartcard. Ce sont des cartes asynchrones car leur protocole de dialogue se fait de manière asynchrone. Elles renferment un microcontrôleur complet (interface entrée/sortie série, mémoire EEPROM etc...).

Dans le cadre de notre projet, nous décidons d’utiliser une carte GOLD, qui fait partie de la seconde catégorie. Elle est caractérisée par sa couleur dorée.

Voici à quoi elle ressemble :



*Exemple d’une carte Gold ; Prix : 2,75 €*

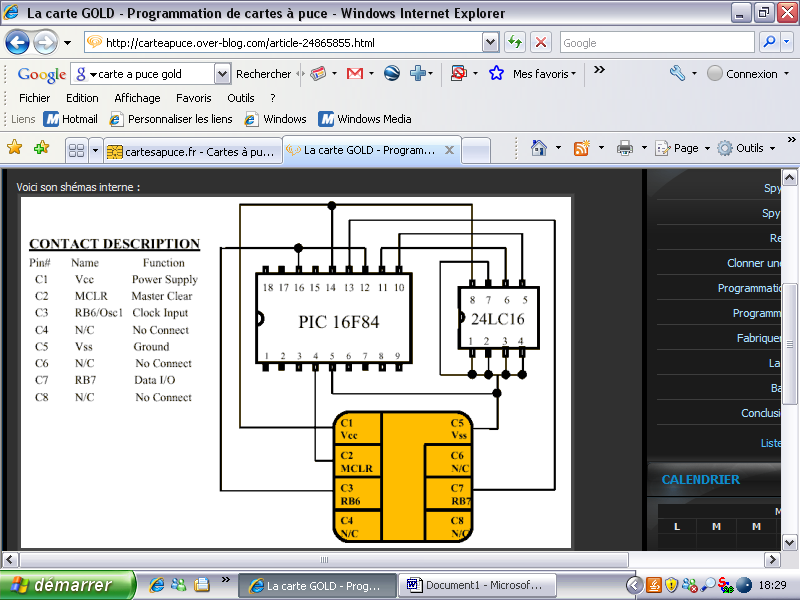
*Composants : PIC + EEPROM Durée de vie : Aléatoire (5 ans)*

Celle-ci est composée d’un microcontrôleur PIC16F84 et d’une mémoire EEPROM série 24LC16 pour compenser le manque de mémoire du PIC qui n’est que de 64 Octets. On a de ce fait une mémoire de 2KOctets.

Dans l’optique de pouvoir programmer plus tard notre propre carte gold, il est pour nous nécessaire de comprendre parfaitement le fonctionnement d’une carte et de savoir quels sont les composants qui la composent à l’aide du schéma interne.

Page 12 – Carte GOLD

Voici son schéma interne :



*Le pic 16F84 est un PIC générique très utilisé de nos jours car il est plutôt simple d’utilisation et performant pour un prix raisonnable.*

**PIC** est l'abréviation de Programmable Interrupt Controler, Contrôleur d'interruption programmable. Ce composant électronique numérique est utilisé dans les premiers microprocesseurs pour gérer les interruptions et renvoyer le numéro d'interruption correspondant.

Page 13 – Carte GOLD

* **Les différentes spécificités de la Norme ISO** :
* La carte ne dispose pas de sa propre horloge, elle est fournie par l’interface, ou maître, et est généralement sous la barre des 4MHz. Ici on utilise une horloge de 3,579MHz.
* Le temps séparant chaque bit est défini par la norme comme étant de 1 bit émis toutes les 372 impulsions de l’horloge du lecteur phœnix. Ceci donne un débit de l’ordre de 9600 bauds avec un quartz de 3,579 MHz.
* La transmission est du type série asynchrone, avec 1 Start-bit, 8 bits de datas, 1 bit de parité paire, et 2 Stop-bits.
* La communication est du type half-duplex, c’est à dire que les entrées et les sorties s’effectuent par alternance et en se servant de la même ligne physique (le lecteur et la carte ne peuvent pas « parler » en même temps). En effet, on utilise une ligne bidirectionnelle.

La carte ne prend jamais l’initiative de l’échange d’informations. Elle ne fait que répondre à des commandes de la forme :

* **CLASSE / INSTRUCTION / PARAMETRE1 / PARAMETRE2 / LONGUEUR**
* **CLASS** : détermine le groupe d’instructions concernées. Si le nombre d’instructions utilisé n’est pas élevé, rien n’interdit de n’utiliser qu’une seule classe. C’est ce que nous ferons dans cet exercice simplifié.
* **INS** : c’est l’instruction proprement dite, ou encore la commande. C’est cet octet qui détermine l’opération à effectuer par la carte.
* **P1** et **P2** : sont 2 paramètres que la carte reçoit avec la commande. Leur signification dépend de la commande envoyée. Ils peuvent être inutilisés mais doivent tout de même être présents.
* **LEN** : peut représenter 2 choses. Soit c’est le nombre d’octets qui suit la commande, dans ce cas la carte s’attendra à recevoir LEN octets supplémentaires avant de traiter la commande dans sa totalité. Soit ce sera la longueur de la chaîne de réponse que le maître s’attend à recevoir en provenance de la carte. C’est l’instruction qui permet d’établir le rôle de LEN.

Page 14 – Dialogue avec la carte

Voici comment se déroule un échange standard d’informations entre le lecteur phœnix et la carte ISO 7816 :

* A la mise en service, le lecteur génère un RESET sur la broche MCLR, la carte répond avec un certain nombre d’octets. Cette réponse s’appelle ATR, pour Answer To Reset.
* Le maître envoie la commande Class INS P1 P2 LEN
* La carte renvoie l’instruction comme accusé de réception INS
* Le maître envoie éventuellement les données complémentaires
* La carte envoie la réponse
* La carte envoie le status et repasse en mode d’attente de commande

Pour bien comprendre la suite, il faut comprendre ce qu’est une liaison série asynchrone.

Le mode série est défini pour signaler que tous les bits d’un octet vont être envoyés en série les uns derrière les autres. Ceci par opposition au mode parallèle pour lequel tous les bits sont envoyés en même temps, chacun sur un fil séparé.

Asynchrone est l’opposé de synchrone, c’est-à-dire qu’il s’agit d’une liaison qui ne fournit pas une horloge destinée à indiquer le début et la fin de chaque bit envoyé. Nous aurons donc besoin d’un mécanisme destiné à repérer la position de chaque bit. Notez qu’il s’agit ici d’un mode asynchrone particulier, étant donné que la carte utilise la même horloge que le maitre. La vitesse ne sera donc exceptionnellement pas donnée en bauds, mais en nombre d’impulsions d’horloge. Un bit toutes les 372 impulsions d’horloge.

Pour recevoir correctement les bits envoyés, il faut convenir d’un protocole de communication. Ce dernier doit comprendre les informations suivantes :

* La vitesse de transmission en bauds (ici 9600 bauds)
* Le format, c’est à dire le nombre de Start-bits, de stop-bits, de bits de données et le type de parité

Nous allons maintenant expliquer ces concepts et indiquer quelles sont les valeurs employées dans la norme ISO 7816.

**Le Start-bit**

Au repos, la ligne se trouve à l’état haut. L’émetteur fait alors passer la ligne à l’état bas : c’est le Start-bit. C’est ce changement de niveau qui va permettre de détecter le début de la réception des bits en série.

Les valeurs admissibles sont 1 ou 2 Start-bit(s). La norme ISO 7816 nécessite un Start-bit.

Page 15 – Dialogue avec la carte

**Les bits de données**

Après avoir reçu le Start-bit, on trouve les bits de données, en commençant par le bit 0. Les normes usuelles utilisent 7 ou 8 bits de data. Pour la norme ISO 7816, nous aurons 8 bits de donnée, ce qui nous donne des valeurs admissibles de 0 à 0xFF pour chaque octet reçu.

**Le bit de parité**

Le bit de parité est une vérification du bon déroulement du transfert. Lors de l’émission, on comptabilise chaque bit de donnée qui vaut 1. A la fin du comptage, on ajoute un bit à 1 ou à 0 de façon à obtenir un nombre de bits total impair ou pair.

On dira qu’on utilise une parité paire si le nombre de bits à 1 dans les bits de données et le bit de parité est un nombre pair.

De même, une parité impaire donnera un nombre impair de bits à 1.

Notez que le bit de parité n’est pas indispensable dans une liaison série asynchrone. Nous avons donc 3 possibilités, à savoir pas de parité, parité paire, ou parité impaire. Dans le cas de la norme ISO 7816, nous devrons utiliser une parité paire.

**Le Stop-bit**

Après la réception des bits précédents, il est impératif de remettre la ligne à l’état haut pour pouvoir détecter le Start-bit de l’octet suivant. C’est le rôle du Stop-bit. Les valeurs admissibles sont de 1 ou 2 Stop-bits.

Dans la norme ISO 7816, nous utiliserons 2 stop-bits, c’est à dire tout simplement un stop-bit d’une durée équivalente à la durée de 2 bits.

**Vitesse et débit**

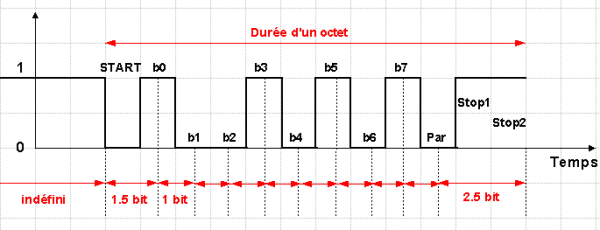
La durée de chaque bit est une constante et dépend de la vitesse de transmission.   
Par exemple :  
  
 Pour une vitesse de 9600 bauds 🡪 c’est à dire 9600 bits par seconde  
Chaque bit durera :  
   
 1s/9600 = 104,17 µS.

Page 16 – Dialogue avec la carte

Le temps nécessaire pour recevoir un octet entier est la somme du temps nécessaire pour recevoir chacun des bits de cet octet**.** Dans le cas de la norme ISO 7816, nous utiliserons 1 Start-bit + 8 bits de données + 1 bit de parité + 2 Stop-bits = 12 bits.   
Le temps total pour recevoir un octet est donc de 1250 µS.   
  
Il faut 12 bits pour envoyer un octet, ce qui correspond à un temps de 1250 µS.  
On sait que la vitesse de transmission est de 9600 bits/sec, on en déduit donc le nombre d’octets transmis en une seconde :  
  
 9600/12=800 octets  
  
Le débit maximum en octets par seconde est donc de 800 Octets par seconde pour la norme ISO 7816.

NB : *Nous parlons ici de débit maximum car ceci est le cas dans lequel un octet commence dès la fin du précédent, ce qui n’est pas obligatoire pour une liaison asynchrone.*

**Acquisition des bits** :



Chaque carré représente ici la durée d’un bit.

La ligne est à l’état haut depuis un temps indéterminé. Survient le Start-bit, qui annonce la transmission de 8 bits de données. Nous voyons que le meilleur moment pour lire le bit 0 est de le mesurer au milieu de sa largeur afin d’obtenir le moins de risque d’erreur possible.

Souvenez-vous que pour la norme ISO 7816, la largeur d’un bit vaut 372 impulsions d’horloge. Comme notre PIC divise déjà la fréquence d’horloge par 4, cela nous fera une largeur de bit équivalent à 372/4 = 93 impulsions d’horloge.

Page 17 – Dialogue avec la carte

Le bit 0 sera donc lu 93 + 46 cycles d’horloge après le début du Start-bit. Ensuite nous pourrons lire tous les bits à un intervalle de temps équivalent à 93 instructions.

Une fois le bit de parité lu, nous avons 2 possibilités :

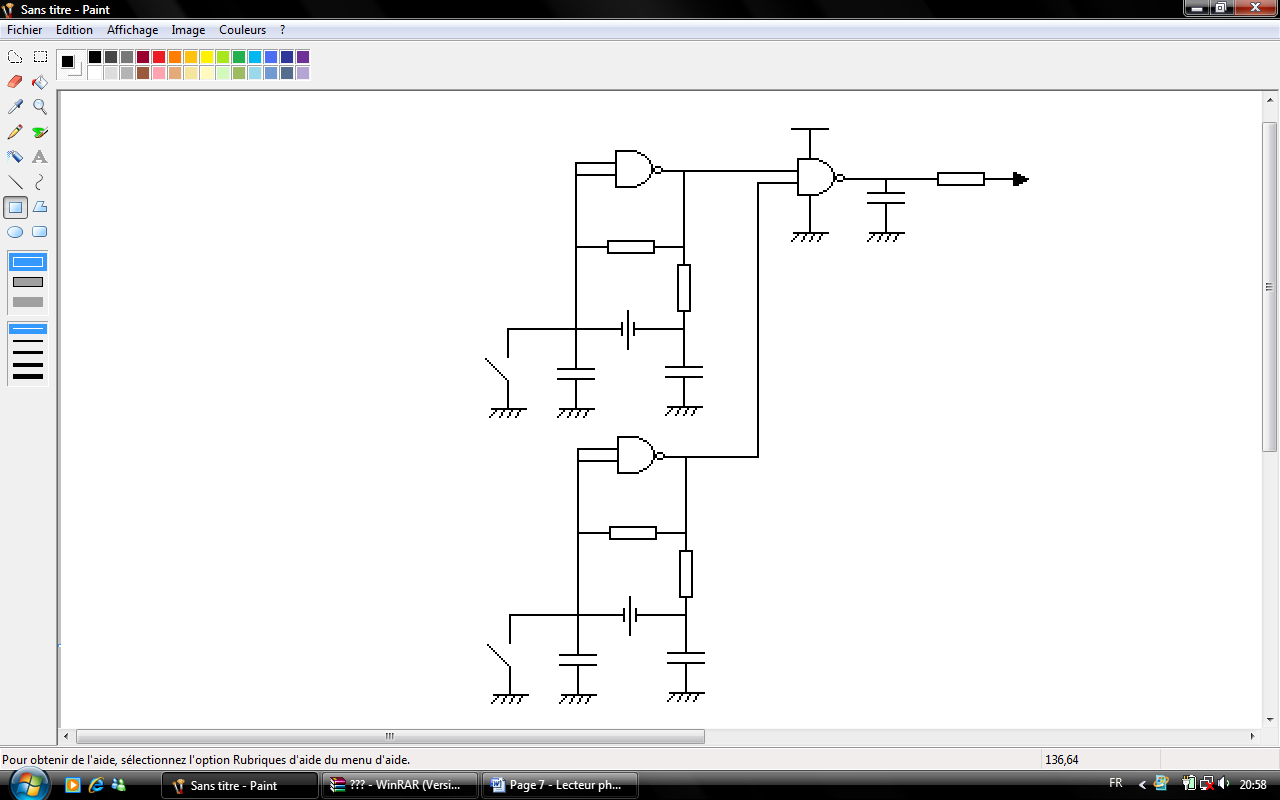
* Si nous souhaitons faire suivre cette lecture par la lecture d’un autre octet, nous devons nous positionner quelque part dans les 2 stop-bits afin d’être prêt à détecter le Start-bit suivant. Nous devons donc attendre entre 0.5 et 2.5 bits.
* Si nous souhaitons envoyer un octet après cette lecture, nous ne pourrons pas le faire avant la fin du second stop-bit, soit au minimum après 2.5 cycles.

Si nous voulons savoir l’Octet reçu lors de cet exemple, on trouve tout simplement :

B0 = 1 B5=1  
 B1 = B2 = 0 B6=0  
 B3 = 1 B7=1  
 B4 = 0 Parité=0

L’octet reçu est donc B’10101001’, soit 0xA9. Profitons-en pour vérifier la parité. Nombre de bits à 1 reçus = 4, donc parité paire, c’est donc conforme à la norme ISO 7816.

Page 18 – Dialogue avec la carte



+5V

IC1A

R3

R5

IC1C

CLK

C5

R1

R2

Schéma de l’horloge

QZ1A

C1

S4

C2

IC1B

L’horloge utilisée par la carte est générée par un oscillateur à Quartz. Cette horloge permet de commander la vitesse des échanges entre la carte et l’utilisateur.

Nous pouvons réaliser une horloge autour du composant IC1A pour fonctionner à 3,579 MHz ou autour de IC1B pour fonctionner à 6 MHz.  
  
 Pour utiliser l’horloge de 3,579Mhz qui est très largement suffisante pour notre utilisation il suffit de mettre un strap à l’emplacement S5 pour bloquer le Quartz de 6Mhz que nous ne souhaitons pas utiliser.

R4

QZ2

C3

C4

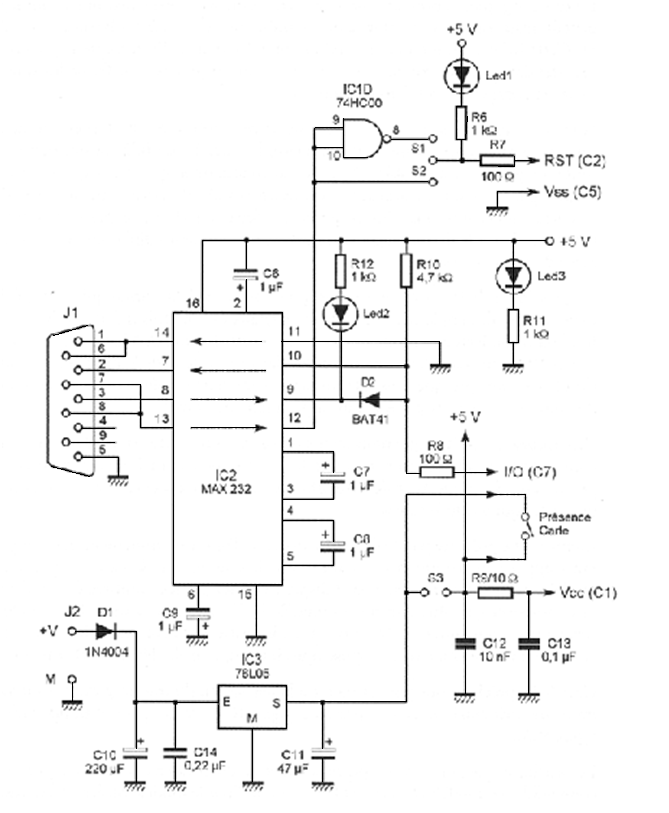
S5

Les condensateurs en parallèles autour du quartz permettent sont bon fonctionnement. Les NAND sont configurés en ‘’inverseur’’ en reliant les 2 broches d’entrée de ceux-ci. Le rebouclage permet à la base d’avoir un circuit oscillant.

Donc un circuit oscillant avec quartz permet d’avoir une fréquence d’utilisation fixe.

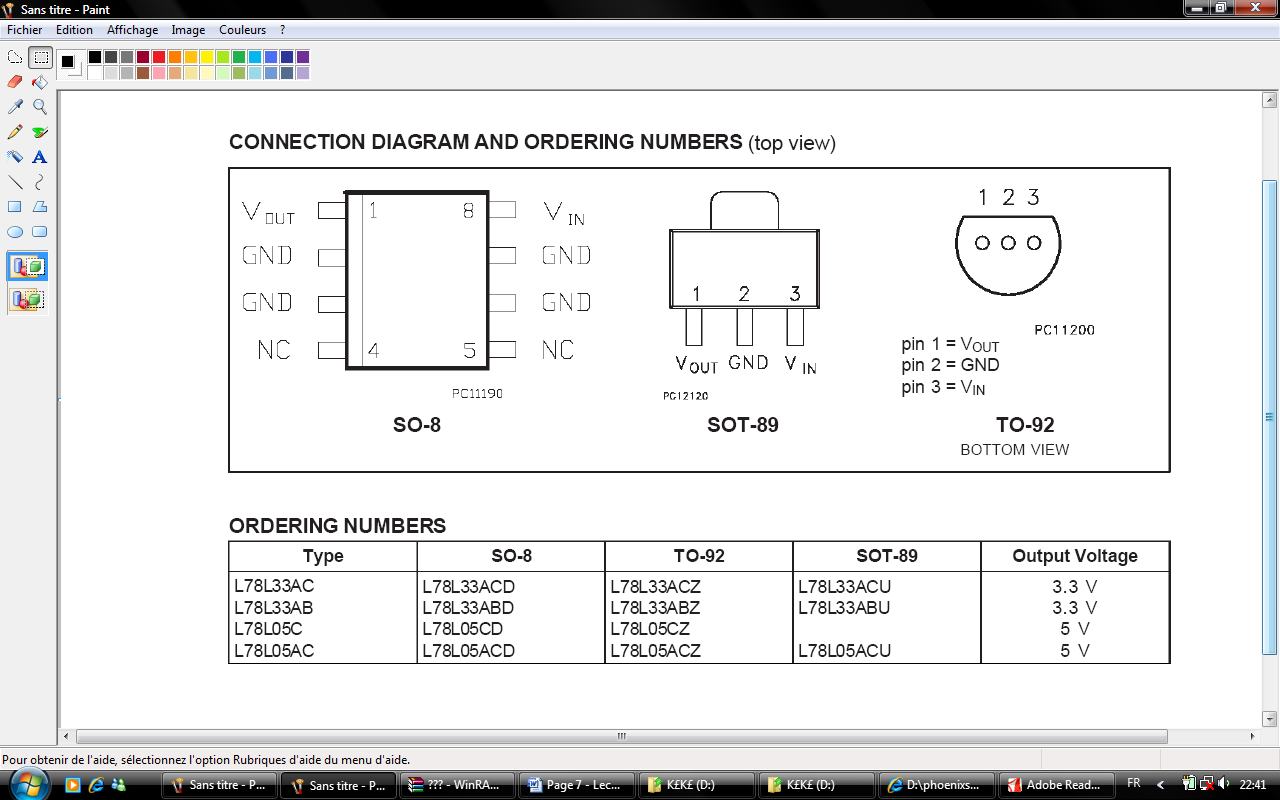
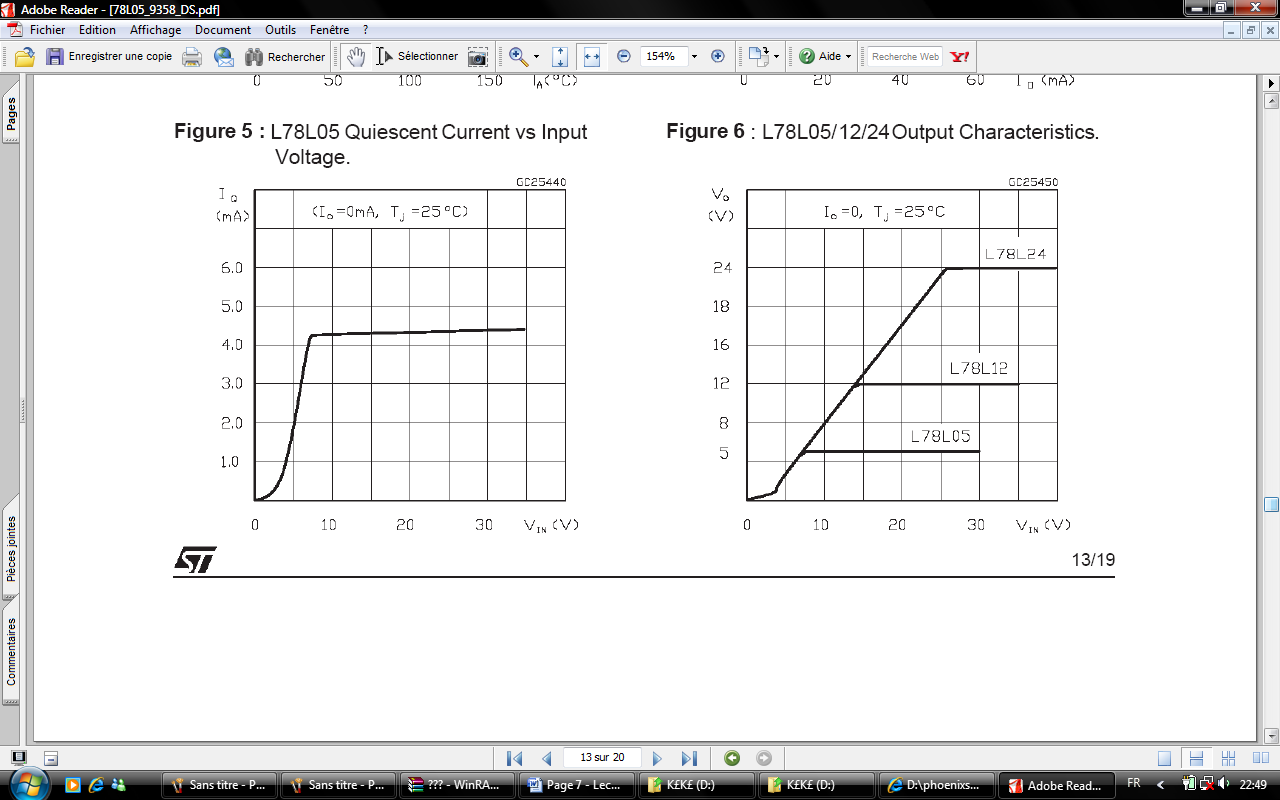
Page 19 – Lecteur phœnix

Nous étudions à présent le coté ENTREE/SORTIE du lecteur de cartes à puce.  
  
Voici le schéma de l’interface du lecteur de carte phœnix :

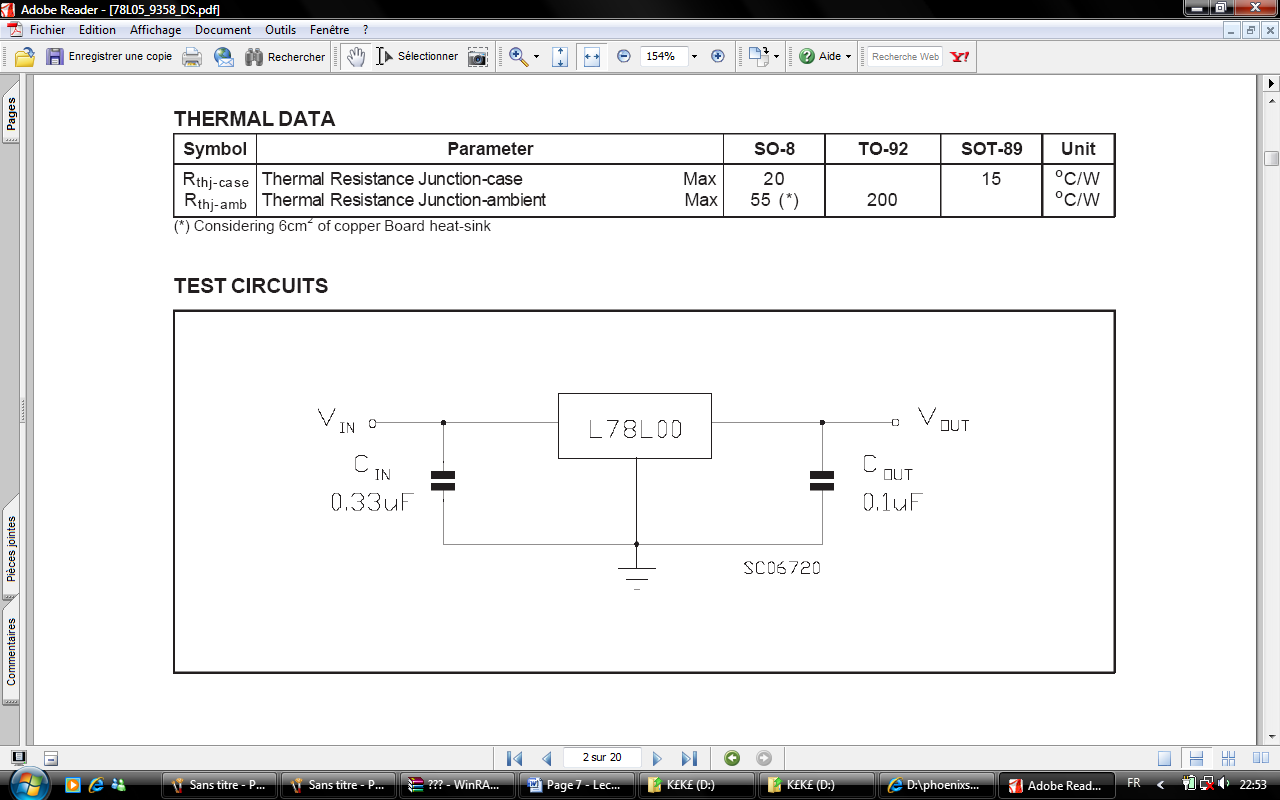


🡪 La prise J2 est la prise jack qui permet l’alimentation du montage. On alimente avec une tension de 9V environ pour les tests avec un courant d’une centaine de mA.  
  
🡪 La diode D1 est une diode qui protège le montage de toute inversion de polarité possible. Elle bloque le courant indésirable.  
  
🡪 L’alimentation est ensuite régulée par l’intermédiaire de IC3. Ce composant est un régulateur à 5V. Il délivre un courant de sortie de 100mA.

Page 20 – Lecteur phœnix



**Utilisation**

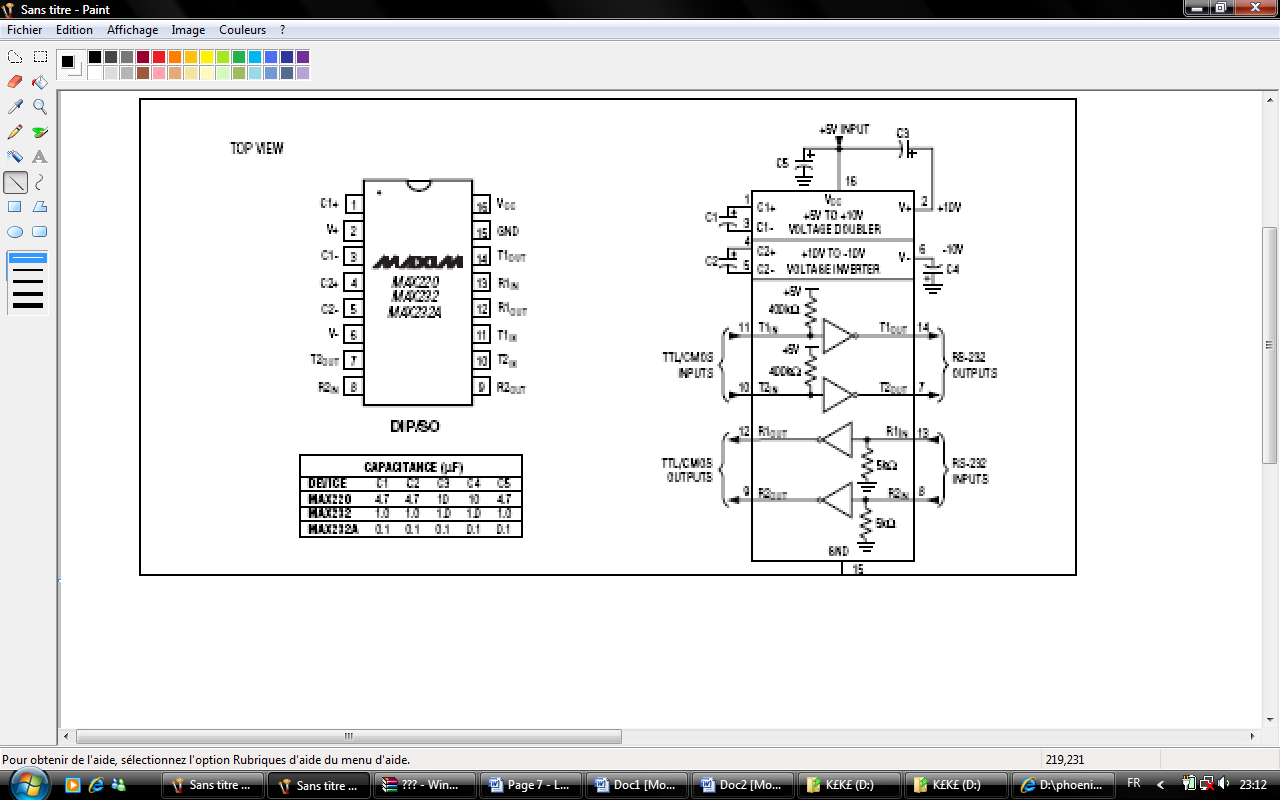


🡪Les condensateurs placés en amont et aval du régulateur permettent d’éliminer les fluctuations résiduelles sur une tension continue. Ces fluctuations peuvent être de deux natures différentes :  
  
 \_ Parasites (hautes fréquences) générés par des éléments externes.  
\_ Fluctuations de la tension suite à des variations importantes de l’intensité consommée.

Page 21 – Lecteur phœnix

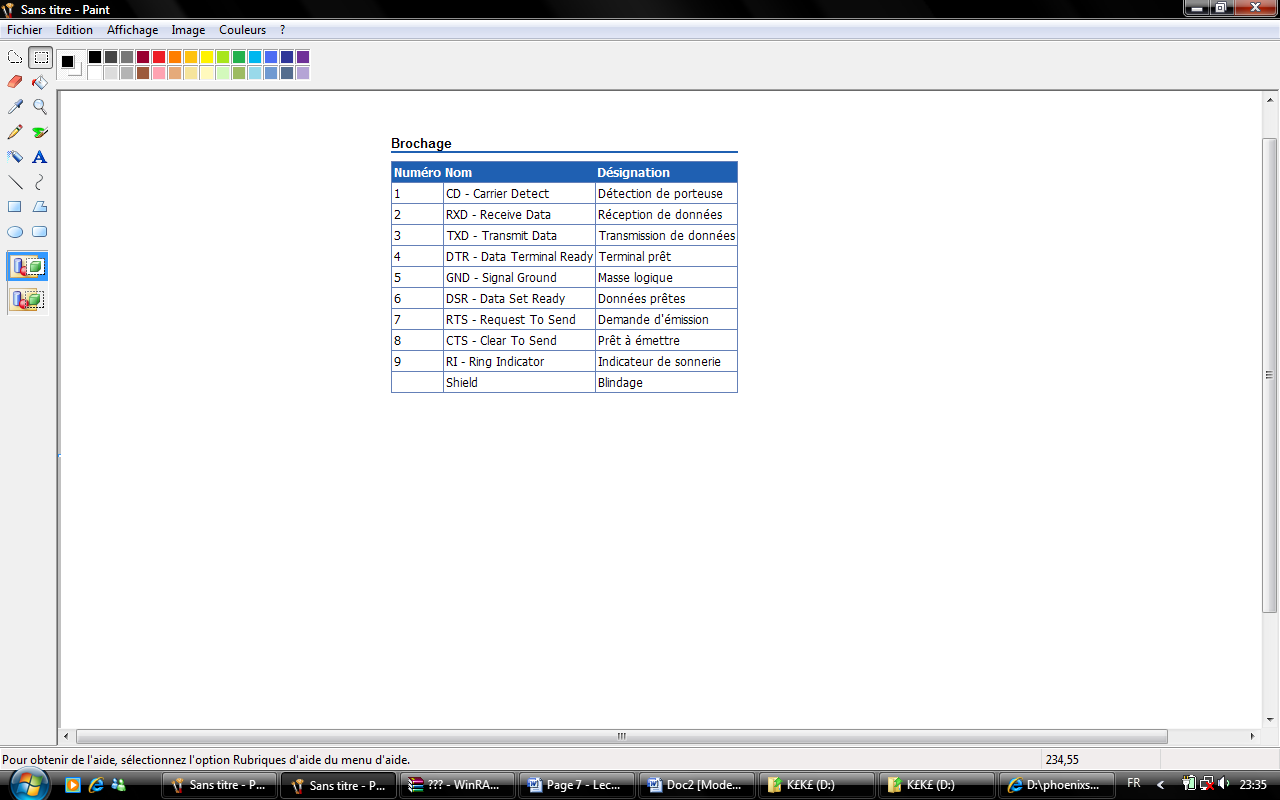
De manière plus symbolique, ils permettent de lisser la tension continue au maximum.  
Les condensateurs chimiques ont une grande capacité pour filtrer les fluctuations importantes et on met donc en amont un condensateur céramique en parallèle pour filtrer les hautes fréquences, donc un condensateur de petite valeur (environ 100nF).  
  
🡪 Max RS232 possède deux alimentations internes qui convertissent en ±10V à partir du +5V. Le premier convertisseur utilise la capacité C1 pour doubler l’entrée +5V (+10V) sur la capacité C3 à la sortie V+. Le second convertisseur utilises la capacité C2 pour inverser le +10V (-10V) sur la capacité C4 à la sortie V-. Ce composant utilise uniquement des condensateurs chimiques de 1μF.

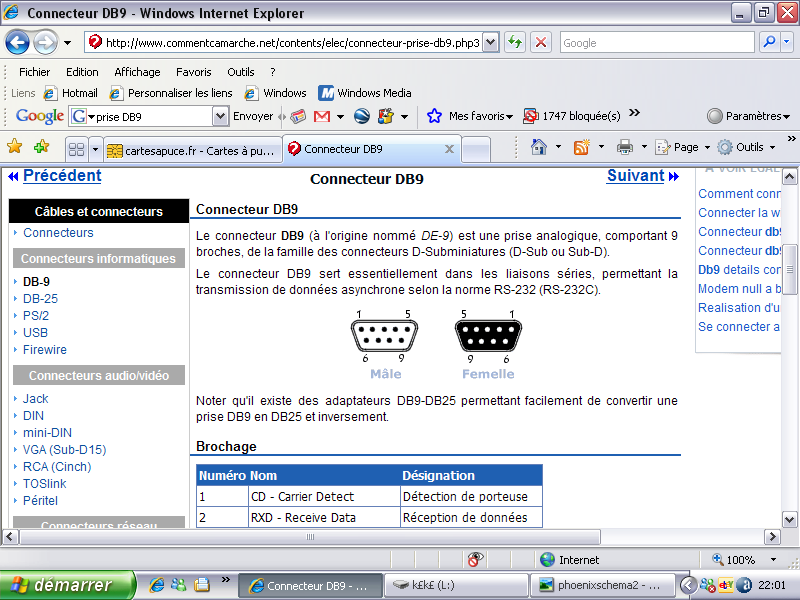
Voici le brochage du composant :



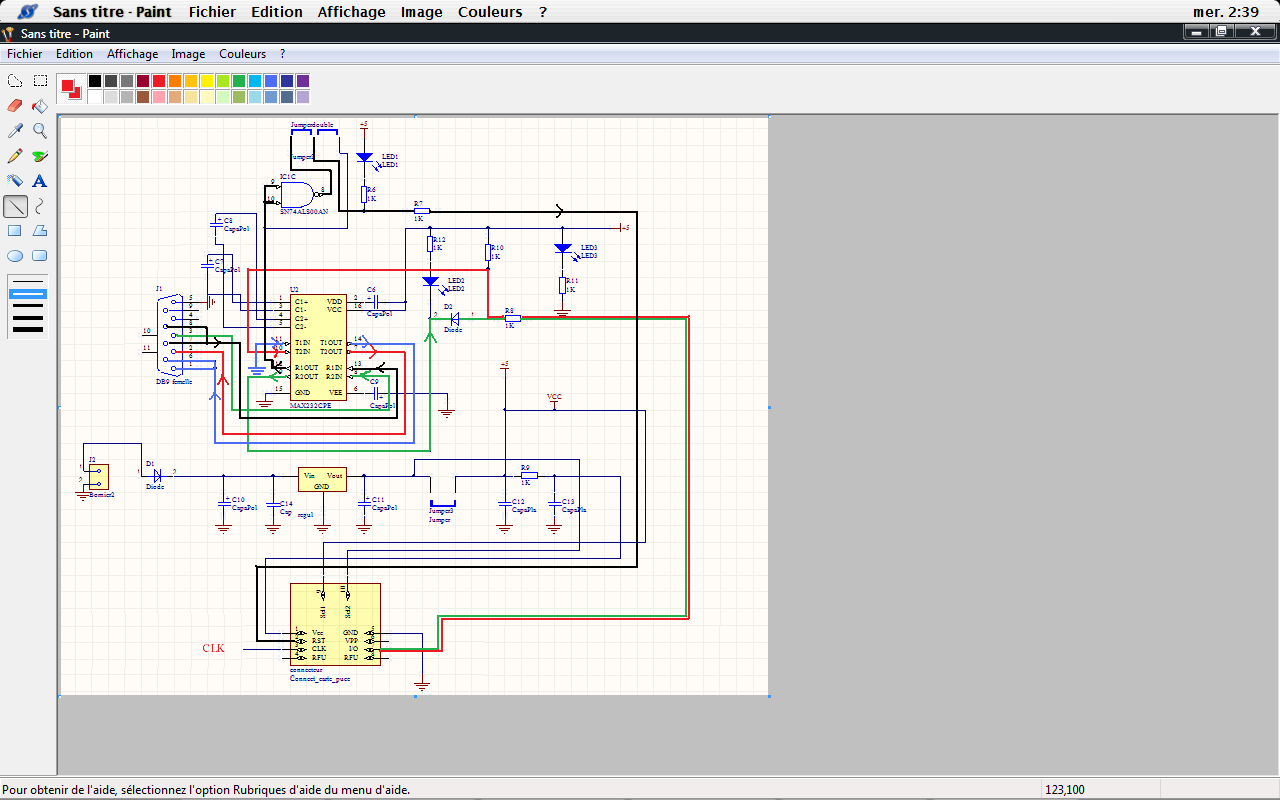
🡪La prise Jack femelle J1 est un connecteur DB9. Le connecteur DB9 est une prise analogique, comportant 9 broches, de la famille des connecteurs D-Subminiatures. Il sert essentiellement dans les liaisons séries, permettant la transmission de données asynchrones selon la norme RS232.

Page 22 – Lecteur phœnix





**COMMUNICATION ENTRE LA CARTE ET LE DB9** (liaison série asynchrone)



Page 23 – Lecteur phœnix

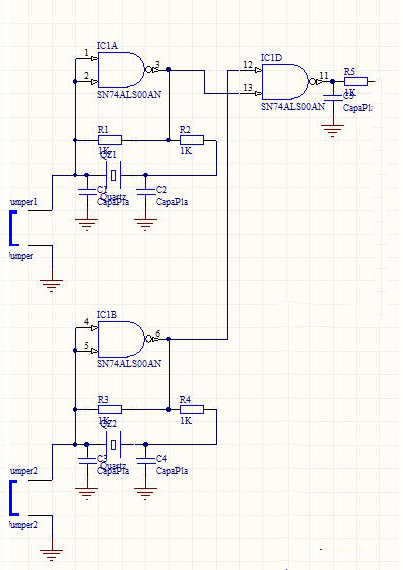
🡪Les broches 1 et 6 nous informe de la présence carte, nous décidons de mettre la broche 11 du Max RS232 à la masse afin d’avoir une présence de carte en permanence. En effet, en mettant à la masse la broche 11 du Max RS232, comme il y a un inverseur dans le Max RS232, on se retrouve avec un état haut sur la broche 14 qui nous indique que la carte est détectée, et prête à l’utilisation. Le logiciel ne pourra de toute façon dialoguer uniquement si la carte est présente. (Voir schéma ci-dessus en **bleu**).

🡪L’interrupteur de présence de carte permet d’alimenter la carte uniquement si celle-ci est enfoncée dans son logement.  
  
🡪Les broches 4 et 9 ne sont pas utilisées.  
  
🡪La broche 5 représente la masse logique, c’est pourquoi on la relie à la masse.  
  
🡪Le Reset de la carte est lui commandé par l’ordinateur ou par nous même grâce au double jumper. Le Reset de la carte est relié aux broches 7 et 8 qui représentent les broches RTS et CTS. Si l’on regarde le chemin en **noir** sur le schéma ci-dessus, on pilote le Reset par ordinateur et si on place le cavalier sur l’autre jumper on fait un Reset permanent.  
  
🡪En ce qui concerne la ligne de bidirectionnelle qui permet une réelle communication entre la carte et le PC, c’est un peu plus complexe. A noter également que le 0 logique correspond au +12V et le 1 logique au -12V selon la norme RS232.  
  
- La broche 2 du DB9 correspond à la réception de données provenant de la carte, le chemin emprunter par les données qui transitent de la carte vers le PC est celui représenté en **rouge** sur le schéma ci-dessus. Cette réception de données à lieu uniquement si la diode D2 est bloquée, c'est-à-dire qu’il faut un niveau logique haut à ses bornes. Il faut qu’un niveau logique bas arrive sur R2IN pour avoir un niveau logique haut sur R2OUT. On doit programmer un état de repos à un niveau logique haut.  
  
-La broche 3 du DB9 permet, elle, la transmission de données vers la carte. Les données transitent par le chemin en **vert.** Les données transitent de façon asynchrone 

Page 24 – Lecteur phœnix

Après une longue étude théorique, nous avons entamé la création de notre lecteur de carte phœnix. Tout d’abord nous avons du créer le schéma de notre lecteur à l’aide du logiciel ‘’Altium Designer’’. Il nous a fallu rentrer le schéma complet comportant deux grandes parties :

* La partie horloge du montage : Composée de 2 quartz permettant de travailler à 2 fréquences d’horloge différentes.
* La partie traitement de la carte en elle-même avec le lecteur de carte (connecteur pour carte à puces iso au format ID1) et pour principale composant un max 232 de la compagnie ‘’Maxim’’.



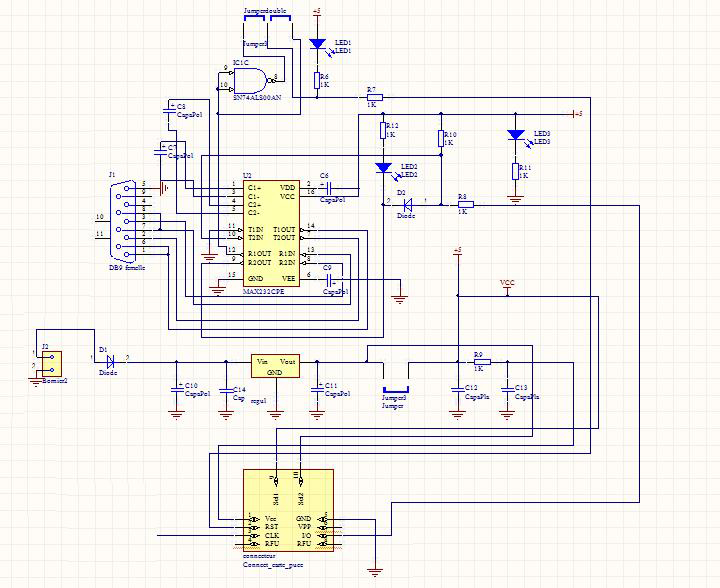
*Figure 1 : Partie horloge du montage*

Comme dans la partie théorique (Page 10), nous utilisons la partie supérieure de notre montage avec le quartz de 3,579 Mhz.

Pour cela, on utilise un cavalier sur un jumper pour pouvoir choisir le mode de fonctionnement (3Mhz ou 6Mhz).

Nous avons dû utiliser les bons composants avec les empreintes appropriées pour le perçage et le soudage de ceux-ci ultérieurement. Pour cela nous devions les chercher dans toutes les librairies qui étaient à notre disposition, autant dire que cela nous a pris énormément de temps.

Page 25 - Réalisations



Comme dans la partie théorique (Page 10), nous avons réalisé la partie ‘’utilisation et traitement de la carte’’.

Pour cela, on utilise un Max 232, des portes logiques et des composants analogiques avec les bonnes empreintes au préalablement cherchées dans les documentations constructeurs.

La broche ‘CLK’ du ‘’connecteur carte à puce’’ est relié à notre premier montage de ‘’l’horloge’’.

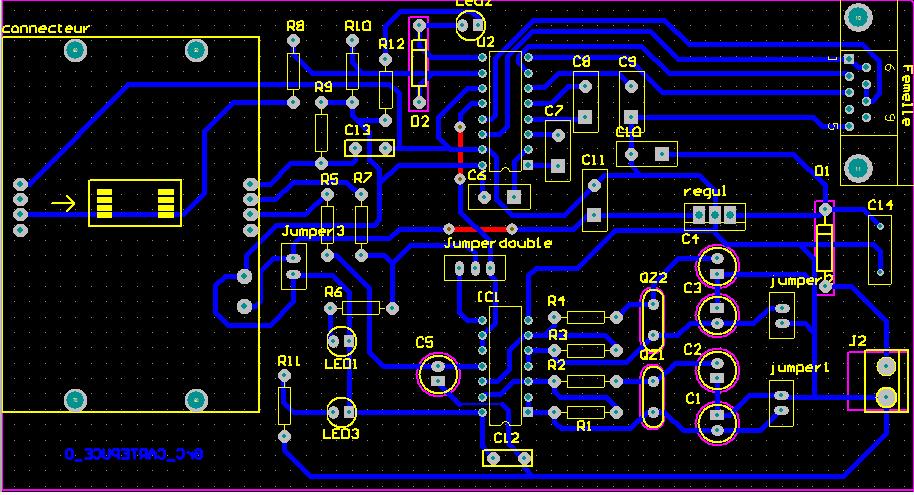
Nous avons eu pas mal de petites difficultés pour finir le schéma intégralement. Notamment le fait que certaines empreintes n’étaient pas les bonnes et qu’il a fallu crée l’empreinte du connecteur carte à puce. Finalement nous avons réussi à terminer le schéma pour pouvoir enfin commencer le routage de la carte.

*Figure 2 : Partie traitement de la carte*

Page 26 - Réalisations

Après avoir réalisé notre schéma avec ‘’Altium Designer’’, nous avons routé notre carte électronique du lecteur phœnix.

Le routage de la carte s’est déroulé sans encombre, nous n’avons en effet pas eu de réelles difficultés à router notre carte car nous avions bien été préparés à cette tâche. Voici notre carte entièrement routé avec ‘’Altium Designer’’ (figure ci-dessous) :



*Figure 3 : Carte du lecteur phœnix* route

Nous avons du procéder à une petite astuce pour passer une piste du circuit : Nous avons réduit la taille d’une piste qui passe sous le composant IC1.

Nous avons ensuite utilisé 2 ‘’straps ‘’ sur la couche extérieure de notre carte (visibles en rouge sur la figure ci-dessus).

NB : Vous pouvez vous référer à la nomenclature en annexe.

Page 27 – Réalisations - Routage

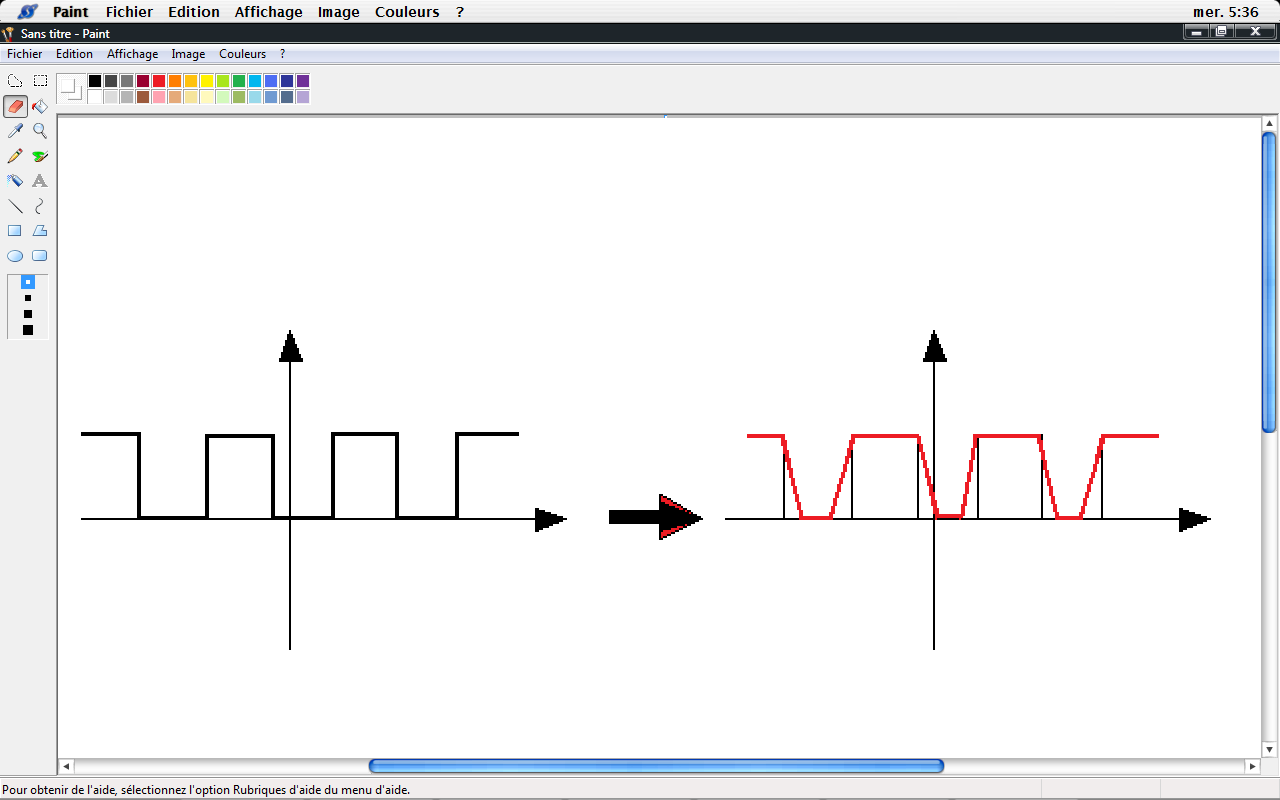
En effet, cette partie est consacrée aux différents tests et contrôles réalisés tout au long de ce projet.  
  
Le premier contrôle effectué a été de vérifier une fois la carte créer, les différentes connections pour être certain qu’il n y est aucune interruption de piste.  
Par la suite, nous avons placé les différents composants sur notre carte et avons procédé à une série de test pour vérifier le bon fonctionnement de celle-ci.

Pour procéder à une vérification de notre lecteur phœnix, nous l’alimentons en 9V sous une centaine de milliampères à l’aide du bornier.  
  
  
Pour commencer, on relève toutes les tensions pour les comparer à la théorie. On regarde avant tout si la LED3 s’allume car elle nous indique la présence du +5V sur notre lecteur.

On vérifie que le régulateur 78L05 stabilise bien la tension de sortie à 5V et qu’il délivre un courant suffisant. C’est le cas le régulateur fonctionne parfaitement.

On regarde à présent si les tensions aux différentes bornes du composant MAX232 sont exactes. On observe bien :   
  
 🡪 La masse sur la patte 15.  
 🡪Une tension de +10V sur la patte 2.  
 🡪Une tension de -10V sur la patte 6.  
 🡪Une tension de +5V sur la patte 16.  
  
  
On s’assure après cela que le jumper du Reset est bien placé, en effet si la LED1 reste allumée cela nous signale qu’il y a un Reset permanent sur le lecteur. On se doit alors de changer le jumper de place. On vérifie également le Reset à l’aide du logiciel HyperTerminal.  
  
  
Ensuite, une fois que les tensions ont été vérifiées, nous devons nous assurer que le Quartz fonctionne. Pour cela, nous observons le signal délivré par le Quartz qui nous intéresse (3,59 MHz) à l’oscilloscope. On observe bien le signal souhaité.  
  
  
  
Seulement nous avons observé le signal d’horloge qui entrait dans la borne CLK, et on observe une déformation du signal due à la capacité C5 qui intégrer notre signal, nous l’avons donc supprimée :

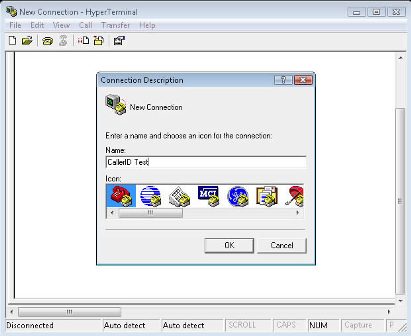
Page 28 - Tests



Ensuite, nous avons vérifié toutes les connectivités du connecteur afin de vérifier que son emprunte utilisé auparavant était exacte. Nous avons pour cela observé chaque connexion et avons vu qu’elles étaient toutes bien placées : VCC, VSS, RST, CLK …

Nous avons vérifié de la même façon les 9 connexions du DB9 femelle.

Nous avons également testé la ligne bidirectionnelle par laquelle transitent les données, avec le logiciel HyperTerminal et le câble série relié à l’ordinateur (comme pour le Reset).



*Logiciel hyperterminal*

Page 29 - Tests

Page 29 - Tests

Page 29 - Tests

Page 29 - Tests

Dans l’optique de programmer notre carte GOLD, nous avons recherché sur internet des utilitaires permettant de programmer le PIC contenu dans une carte GOLD.

Nous avons cherchés dans les documentations constructeurs ou ‘’datasheet’’ et nous avons trouvé quelques sources très intéressantes.

Vous trouverez des pages intéressantes de ces datasheet en annexes mais vous trouverez ci-dessous quelques lignes de codes commentés très utiles pour la programmation du PIC16F84 contenu dans notre carte. Ce fut un choix de notre part de mettre ces quelques lignes de codes dans notre projet et non pas dans les annexes car celles-ci sont très bien commentés…

|  |
| --- |
| **Réception d’un octet** |

|  |  |  |  |
| --- | --- | --- | --- |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ; Réception d'un octet provenant du maître  ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ;--------------------------------------------------------------------- ; Caractère lu dans W. La parité pas n'est pas vérifiée ;--------------------------------------------------------------------- | | | |
| Receive |  |  |  |
|  |  | ; attendre début start-bit ; ------------------------ | |
|  | btfsc | SERIAL | ; Tester si start bit arrivé |
|  | goto | Receive | ; non, attendre |
|  |  | ; se positionner sur le milieu du 1er bit utile ; --------------------------------------------- | |
|  | call | temp\_1bd | ; attendre 1bit et demi |
|  |  | ; réception du caractère ; ---------------------- | |
|  | movlw | 0x8 | ; pour 8 bits |
|  | movwf | cmptbts | ; dans compteur de bits |
| Recloop |  |  |  |
|  | bcf | STATUS , C | ; Carry = 0 |
|  | btfsc | SERIAL | ; tester si bit = 0 |
|  | bsf | STATUS , C | ; Carry = bit reçu |
|  | rrf | caract , f | ; faire entrer le bit par la gauche |
|  | call | temp\_1b | ; attendre milieu caractère suivant |
|  | decfsz | cmptbts , f | ; décrémenter compteur de bits |
|  | goto | Recloop | ; pas dernier, suivant |
|  |  | ; on pointe actuellement sur le centre du bit de parité ; reste donc à attendre +- 1.5 bits pour être dans le second stop-bit ; ------------------------------------------------------------------------------ | |
|  | call | temp\_1bd | ; Attendre 1,5 bit |
|  | movf | caract , w | ; charger caractère lu |
|  | return |  | ; et retour |

Ce sous-programme ne contient pas de difficultés particulières. Remarquez que plutôt que de positionner chaque bit reçu directement à la bonne position, on le fait entrer dans b7 en se servant d’une instruction " rrf ". Le bit précédent est de fait reculé en b6 et ainsi de suite. Le premier bit lu se retrouve donc en b0, le dernier restant en b7.

Concernant la dernière temporisation, nous pointons à ce moment sur le centre du 9ème bit, c’est à dire le bit de parité. Comme nous ne le traitons pas, inutile de le lire. Nous devons alors nous positionner sur un endroit où la ligne est repassée au niveau " 1 ", c’est à dire un des stop-bits, afin de pouvoir éventuellement commencer une attente d’un nouveau caractère.

Nous devons donc nous attendre entre 0.5 et 2.5 bits. La sous-routine 1.5 bit est en plein dans cette zone et est directement utilisable.

Le présent sous-programme utilise des variables que nous déclarerons plus loin.

|  |
| --- |
| **L’émission d’un caractère** |

N’oublions pas que notre carte n’émet qu’en réponse à une interrogation du " maître ". Donc, notre sous-programme d’émission sera appelé après le sous-programme de réception d’un octet.

Il est important également de se souvenir que nous travaillons en mode half-duplex, c’est à dire que la même ligne sert pour les entrées et les sorties. Comme chacun des interlocuteurs parle à tour de rôle, il faut laisser à chacun le temps de repasser en lecture après l’envoi de son dernier message. Ceci s’appelle " temps de retournement ". Il faut également se rappeler que notre routine de réception s’achèvera quelque part au milieu des stop-bits, il faudra également laisser à l’émetteur le temps de finir d’envoyer la fin de ses stop-bits.

Un bon compromis et la facilité d’écriture du programme nous permet de choisir une attente de 1.5 bit avant de commencer à émettre. J’ai choisi cette valeur car cette temporisation permet d’initialiser le timer. Cette valeur n’est cependant pas critique. Il faut simplement répondre après que le " maître " soit placé en mode de réception, et avant qu’il ne considère que vote carte n’a pas répondu.

Notre programme va donc effectuer les opérations suivantes :

* Attendre temps choisi avant émission
* Passer en émission et envoyer le start-bit
* Envoyer les 8 bits en commençant par b0
* Pour chaque bit " 1 " envoyé, inverser la parité, pour obtenir une parité paire
* Envoyer la parité
* Envoyer les 2 stop-bits
* Repasser en réception

Voici donc notre sous-programme :

|  |  |  |  |
| --- | --- | --- | --- |
| ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ; Envoi d'un octet vers le lecteur de carte ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ;------------------------------------------------------------------------------ ; envoie l'octet contenu dans le registre w vers le lecteur de carte ;------------------------------------------------------------------------------ | | | |
| Send |  |  |  |
|  | movwf | caract | ; Sauver caractère à envoyer |
|  | call | temp\_1bd | ; attendre 1 bit et demi |
|  | BANK1 |  | ; passer banque1 |
|  | bcf | SERIAL | ; port série en sortie |
|  | BANK0 |  | ; repasser banque0 |
|  |  | ; envoyer start-bit ; -------------------- | |
|  | bcf | SERIAL | ; envoyer 0 : start-bit |
|  | clrf | parite | ; effacer bit de parité |
|  | movlw | 8 | ; pour 8 bits à envoyer |
|  | movwf | cmptbts | ; dans compteur de bits |
|  | call | temp\_1b | ; attente entre 2 bits |
|  |  | ; envoyer 8 bits de data ; --------------------------- | |
| Send\_loop |  |  |  |
|  | rrf | caract , f | ; décaler caractère, b0 dans carry |
|  | rrf | caract , w | ; carry dans b7 de w |
|  | andlw | 0x80 | ; garder bit à envoyer en position b7 |
|  | xorwf | parite , f | ; positionner parité |
|  | xorwf | PORTB , w | ; Garder 1 si changement sur SERIAL |
|  | xorwf | PORTB , f | ; si oui, inverser RB7 |
|  | call | temp\_1b | ; attente entre 2 bits |
|  | decfsz | cmptbts , f | ; décrémenter compteur de bits |
|  | goto | Send\_loop | ; pas dernier, suivant |
|  |  | ; envoyer parité ; ------------------ | |
|  | movf | parite , w | ; charger parité paire calculée |
|  | xorwf | PORTB , w | ; Si serial différent de bit à envoyer |
|  | xorwf | PORTB , f | ; alors inverser RB7 |
|  | call | temp\_1b | ; attendre fin de parité |
|  |  | ; envoyer 2 stop-bits ; ------------------- | |
|  | BANK1 |  | ; passer banque1 |
|  | bsf | SERIAL | ; repasser en entrée (et niveau haut) |
|  | BANK0 |  | ; passer banque0 |
|  | call | temp\_1b | ; attendre temps entre 2 bits |
|  | call | temp\_1b | ; attendre temps entre 2 bits |
|  | return |  | ; et retour |

Si vous êtes attentifs, vous avez remarquer une légère inversion en ce qui concerne la fin du protocole. En effet, plutôt que d’envoyer un niveau 1 (stop-bit) puis d’attendre 2 bits et enfin de repasser en entrée, nous sommes passé en entrée puis avons attendu 2bits.

Ceci est strictement identique, car la résistance de rappel au +5V du PORTB, que nous avons activée se charge d’imposer un niveau haut sur RB7 dès que cette pin est remise en entrée.

La routine servant à envoyer les 8 bits utilise l’instruction " xorwf " au lieu de tester si le bit à émettre vaut 1 ou 0. La routine commence par placer le bit à émettre en position b7. Pourquoi b7 ? Et bien tout simplement parce que c’est également b7 dans le PORTB que nous devrons modifier. Nous utilisons en effet RB7. La procédure utilisée permet de positionner en même temps le bit de parité. Effectuez l’opération manuellement sur papier pour vous en convaincre. Tentez d’écrire une routine utilisant " btfss " et " btfsc " et comparez les résultats obtenus.

Un petit mot concernant l’envoi du bit proprement dit, c’est à dire les 2 instructions :

|  |  |  |  |
| --- | --- | --- | --- |
|  | xorwf | PORTB , w | ; Garder 1 si changement sur SERIAL |
|  | xorwf | PORTB , f | ; si oui, inverser RB7 |

Nous commençons ici par lire le PORTB et nous effectuons un " xorlw " avec le bit à envoyer contenu dans " W ". Comme "W" ne contient que ce bit, RB0 à RB6 ne seront pas modifiés par les opérations suivantes.

Si le bit7 contenu dans "W" est différent de celui présent sur RB7, nous obtenons b7 = 1 dans W. Dans le cas où b7 de W est identique à RB7, nous obtenons 0. N’oublions pas en effet que le ",W" permet de placer le résultat dans "W".

Si nous appliquons " W " sur le PORTB en effectuant un " xorwf ", nous inverserons RB7 uniquement si b7 de " W " vaut 1, c’est à dire, en d’autres mots : Nous inverserons RB7 uniquement si son niveau actuel est différent du niveau que nous devons envoyer.

Ceci paraît un peu " tordu ", mais si vous essayez d’écrire cette routine autrement, vous verrez qu’en effectuant plusieurs essais, vous arriverez à un résultat identique, tout ceci à cause de la parité à gérer.

Remarquez que vous pouvez ignorer la vérification du bit de parité en réception, c’est votre problème. Par contre, vous êtes obligé de positionner correctement celle-ci à l’émission, car il y a de fortes chances pour que le maître vérifie cette parité.

|  |
| --- |
| **Initialisation** |

Nous allons maintenant étudier le corps de notre programme principal. Comme tout programme qui se respecte, nous commencerons par l’initialisation. Celle-ci va être très simple, elle se limite à initialiser le registre OPTION.

|  |  |  |  |
| --- | --- | --- | --- |
| ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ; INITIALISATIONS ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | | | |
|  | org | 0x000 | ; Adresse de départ après reset |
| init |  |  |  |
|  | BANK1 |  | ; passer banque1 |
|  | movlw | OPTIONVAL | ; charger masque |
|  | movwf | OPTION\_REG | ; initialiser registre option |
|  | BANK0 |  | ; passer banque0 |

|  |
| --- |
| **Envoi de l’ATR** |

ATR signifie Answer To Reset, c’est à dire réponse à un reset. C’est une commande envoyée par la carte lors d’une mise sous tension ou lors d’un reset généré par le maître via la broche " MCLR ".

Tout d’abord, nous allons attendre un peu que le maître soit prêt à recevoir notre ATR. Il peut être nécessaire d’ajuster ce temps en fonction des caractéristiques du maître.

|  |  |  |  |
| --- | --- | --- | --- |
| ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ; PROGRAMME PRINCIPAL ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | | | |
| start |  |  |  |
|  |  | ; on commence par attendre un peu ; ------------------------------- | |
|  | call | temp\_1bd | ; attendre 1 bit et demi |

Ensuite, nous pouvons envoyer l’ATR. Pour des raisons de facilité, nous avons écrit notre ATR dans la zone eeprom interne. J’ai choisi un ATR de 5 caractères que j’ai inventé. Consultez les caractéristiques du maître pour connaître les ATR valides de votre application.

|  |  |  |  |
| --- | --- | --- | --- |
| ;=============================================================== ; ENVOI DE L'ATR ;=============================================================== ;--------------------------------------------------------------------------- ; Envoi d'un ATR fictif : l'ATR est dans les 5 octets de 0x04 à ; 0x00 de l'eeprom interne. L'ATR est écris en sens inverse ;--------------------------------------------------------------------------- | | | |
|  | movlw | 0x5 | ; pour 5 octets |
|  | movwf | cmpt1 | ; dans compteur de boucles = adresse |
| ATR\_loop |  |  |  |
|  | decf | cmpt1 , w | ; adresse à lire = compteur de boucles-1 |
|  | call | Rd\_eeprom | ; Lire un octet eeprom interne |
|  | call | Send | ; Envoyer sur le décodeur |
|  | decfsz | cmpt1 , f | ; décrémenter compteur |
|  | goto | ATR\_loop | ; pas fini, suivant |

**Remarques**

L’utilisation de la commande " decfsz " facilite l’écriture des boucles, mais, comme le compteur de boucles est en même temps l’offset de la position en eeprom, l’ATR sera écrit à l’envers dans l’eeprom, c’est à dire du dernier vers le premier octet.

A l’endroit de l’appel de la sous-routine " Rd\_eeprom ", le compteur de boucles variera de 5 à 1. Or, notre adresse eeprom variera de 4 à 0. Donc, l’opération " decf " permet de charger dans " W " la valeur du compteur de boucles – 1.

Le sous-routine " Rd\_eeprom " n’est rien d’autre que notre macro de lecture de mémoire eeprom transformée en sous-programme. La voici :

|  |  |  |  |
| --- | --- | --- | --- |
| ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ; Lecture d'un octet en eeprom interne \* ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ;------------------------------------------------------------------------------------------------------- ; Lecture d'un octet de l'eeprom interne. L'adresse est passée dans w.octet lu est dans W ;-------------------------------------------------------------------------------------------------------- | | | |
| Rd\_eeprom |  |  |  |
|  | movwf | EEADR | ; adresse à lire dans registre EEADR |
|  | bsf | STATUS , RP0 | ; passer en banque1 |
|  | bsf | EECON1 , RD | ; lancer la lecture EEPROM |
|  | bcf | STATUS , RP0 | ; repasser en banque0 |
|  | movf | EEDATA , w | ; charger valeur lue dans W |
|  | return |  | ; retour |

Pensons également à inscrire notre ATR dans la zone eeprom :

|  |  |  |  |
| --- | --- | --- | --- |
| ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ; DECLARATIONS DE LA ZONE EEPROM \* ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | | | |
|  | org | 0x2100 | ; adresse début zone eeprom |
| ATR | DE | 0x07 | ; Réponse à l'ATR |
|  | DE | 0xAB | ; B7 01 BB AB 07 |
|  | DE | 0xBB |  |
|  | DE | 0x01 |  |
|  | DE | 0xB7 |  |
|  |  |  |  |

|  |
| --- |
| **L’envoi du status** |

La norme ISO 7816 demande que chaque émission d’une réponse de la carte soit suivi de 2 octets de status qui indiquent la manière dont a été interprétée la commande.

J’ai inventé des status pour cet exercice. Le status " 80 00 " indiquera que la commande a été exécutée correctement. J’utiliserai également le status " 60 40 " pour indiquer que la commande n’existe pas.

Nous allons donc créer 2 sous-programme. Un qui envoie le status standard, l’autre qui envoie n’importe quel status.

|  |  |  |  |
| --- | --- | --- | --- |
| ;=============================================================== ; ENVOI DU STATUS STANDARD ;=============================================================== ;----------------------------------------------------------------------- ; Envoie le status standard, dans ce cas on a pris 0x80 0X00 ;------------------------------------------------------------------------ | | | |
| Statstd |  |  |  |
|  | movlw | 0x80 | ; prendre 1er octet status |
|  | call | Send | ; l'envoyer |
|  | clrw |  | ; effacer w |
|  | call | Send | ; envoyer 00 |
|  | goto | classe | ; et traiter classe |
| ;================================================================ ; ENVOI D'UN STATUS SPECIFIQUE ;================================================================ ;---------------------------------------------------------------------------------------- ; Envoie d'abord l'octet contenu dans w, puis l'octet contenu dans status2 ;---------------------------------------------------------------------------------------- | | | |
| Statxx |  |  |  |
|  | call | Send | ; on envoie valeur |
|  | movf | status2 , w | ; charger byte à envoyer |
|  | call | Send | ; on envoie 2ème octet du status |

|  |
| --- |
| **Réception de la classe** |

Maintenant, notre carte passe en mode réception et attend sa première commande. Notre programme, pour des raisons de facilité ne gère qu’une seule classe. Nous nous contentons donc de lire l’octet, sans le vérifier ni le traiter. Il s’agit en effet d’un exercice didactique.

|  |  |  |  |
| --- | --- | --- | --- |
| ;================================================================= ; LECTURE DE LA CLASSE ;================================================================= ;------------------------------------------------------------------------------------------------------------- ; on considère dans cet exemple qu'il n'y a qu'une seule classe valide. ; on attend l'arrivée de la classe et on ne la traite pas ;------------------------------------------------------------------------------------------------------------- | | | |
| classe |  |  |  |
|  | call | Receive | ; Lire le byte venant du maître |

|  |
| --- |
| **Réception de INS, P1, P2, et LEN** |

|  |  |  |  |
| --- | --- | --- | --- |
| ;=============================================================== ; LECTURE DE INS, P1, P2, LEN ;=============================================================== ;----------------------------------------------------------------------------------------------------------- ; INS sera placé dans la variable Ser\_ins P1 sera placé dans Ser\_P1 et P2 dans Ser\_P2 ; La longueur du champs de data sera dans Ser\_len ;----------------------------------------------------------------------------------------------------------- | | | |
|  | movlw | Ser\_Ins | ; pointer sur emplacement instruction |
|  | movwf | FSR | ; initialiser pointeur indirection |
| read\_loop |  |  |  |
|  | call | Receive | ; Lire un octet |
|  | movwf | INDF | ; sauver dans emplacement prévu |
|  | incf | FSR , f | ; pointer sur suivant |
|  | btfss | FSR , 0x4 | ; Tester si adresse 0x10 atteinte |
|  | goto | read\_loop | ; non, octet suivant |

Vous pouvez constater l’utilisation de l’adressage indirect pour sauvegarder les octets reçus dans 4 emplacements consécutifs. Nous choisirons les adresses 0x0C à 0x0F, ce qui nous permet facilement de détecter la fin de la commande. En effet, une fois le 4ème octet sauvé, FSR pointe sur 0x10, il suffit donc de tester son bit 4 pour tester la fin de la boucle, sans avoir besoin d’un compteur de boucles supplémentaire.

|  |
| --- |
| **Contrôle de l’instruction reçue** |

Une fois la commande reçue, nous devons traiter les différentes commandes reçues. Dans notre exemple didactique, j’ai implémenté une seule instruction. La seule instruction valide est l’instruction 0x25.

Cette instruction calcule la somme de P1 et P2, et renvoie le résultat. Comme LEN contient la longueur de la chaîne de réponse, si LEN est supérieur à 1, la réponse sera complétée par des 0xFF. Toute autre instruction sera considérée comme incorrecte.

Voici notre test :

|  |  |  |  |
| --- | --- | --- | --- |
| ;============================================================== ; SWITCH SUIVANT INSTRUCTION RECUE ;============================================================== ;--------------------------------------------------------------------------------------------------------- ; Nous allons imaginer que nous allons réagir à une instruction 0x25 ; Toute autre instruction sera considérée comme incorrecte ;--------------------------------------------------------------------------------------------------------- | | | |
|  | ; tester instruction reçue ; ------------------------ | | |
|  | movf | Ser\_Ins , w | ; charger instruction reçue |
|  | sublw | 0x25 | ; comparer avec 0x25 |
|  | btfsc | STATUS , Z | ; tester si identique |
|  | goto | Ins25 | ; oui, traiter instruction 25 |
|  | ; traiter instruction incorrecte ; ------------------------------ | | |
|  | movlw | 0x40 | ; charger octet2 status à envoyer |
|  | movwf | status2 | ; placer dans variable |
|  | movlw | 0x60 | ; charger octet 1 status |
|  | goto | Statxx | ; envoyer status |

Nous voyons ici que si l’instruction reçue est 0x25, nous sautons au traitement de l’instruction. Dans le cas contraire, nous envoyons le status " 60 40 " qui signifie dans notre cas " instruction incorrecte ".

|  |
| --- |
| **Traitement d’une instruction** |

Nous en arrivons maintenant au traitement de notre instruction proprement dite.  
  
On va traiter cette instruction de la manière suivante :

* Comme dans toute instruction, on renvoie l’instruction reçue
* La carte renvoie la somme de P1 et de P2.
* La trame d'envoi est complété par des 0xFF pour atteindre une longueur totale de data identique à Ser\_Len
* Ensuite le status standard est envoyé " 80 00 "

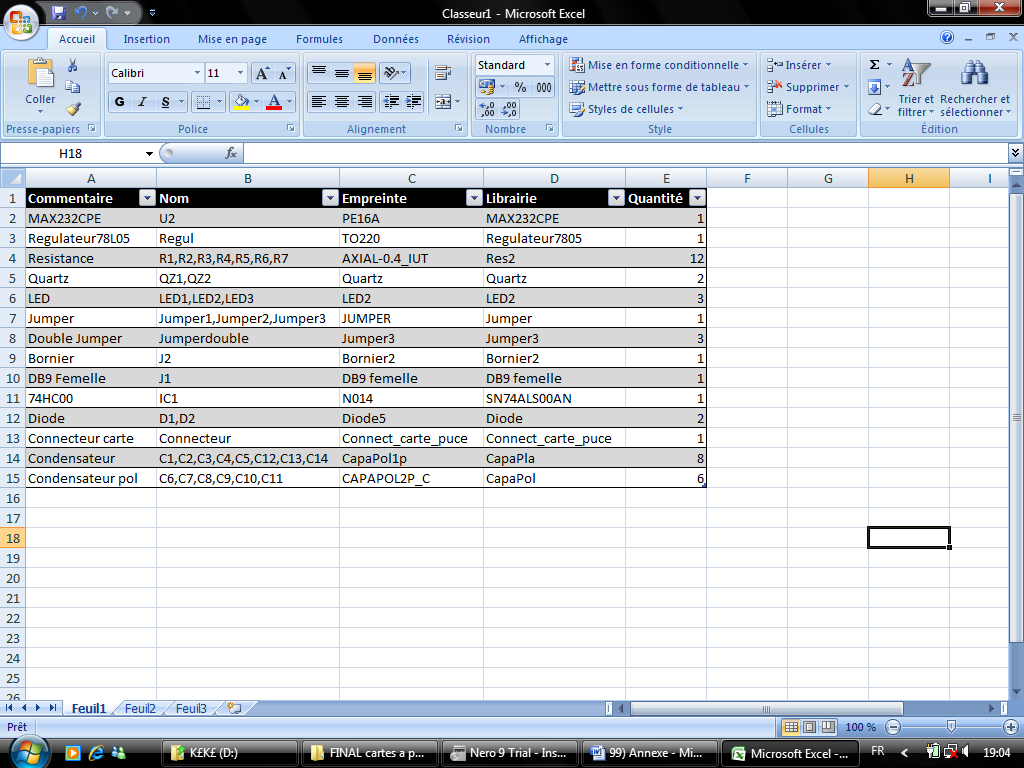
|  |  |  |  |
| --- | --- | --- | --- |
| ;================================================================ ; TRAITER INSTRUCTION 25 ;================================================================ | | | |
| Ins25 |  |  |  |
|  |  | ; envoyer écho de la commande ; ------------------------------------- | |
|  | movf | Ser\_Ins , w | ; charger commande reçue |
|  | call | Send | ; renvoyer en écho |
|  |  | ; renvoyer P1 + P2 ; --------------------- | |
|  | movf | Ser\_P1 , w | ; charger P1 |
|  | addwf | Ser\_P2 , w | ; + P2 |
|  | call | Send | ; envoyer résultat |
|  |  | ; Tester longueur de la réponse ; ----------------------------------- | |
|  | decf | Ser\_Len , f | ; car déjà résultat envoyé |
|  | btfsc | STATUS , Z | ; tester si complet |
|  | goto | Statstd | ; oui, envoyer status standard |
|  |  | ; compléter avec des 0xFF | |
|  |  | ; ------------------------------- | |
| Insloop |  |  |  |
|  | movlw | 0xFF | ; valeur à envoyer |
|  | call | Send | ; envoyer 0xFF |
|  | decfsz | Ser\_Len , f | ; décrémenter compteur de boucles |
|  | goto | Insloop | ; pas fini, suivant |
|  |  | ; envoyer status standard ; ----------------------------- | |
|  | goto | Statstd | ; envoyer status standard |

|  |
| --- |
| **Les variables** |

Il nous reste maintenant à déclarer les variables utilisées. Nous avons décidé ici d’utiliser des variables locales lorsque c’était possible :

|  |  |  |  |
| --- | --- | --- | --- |
| ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ; DECLARATIONS DE VARIABLES ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | | | |
|  | CBLOCK | 0x00C | ; début de la zone variables |
|  |  | Ser\_Ins : 1 | ; instruction ISO7816 |
|  |  | Ser\_P1 : 1 | ; paramètre 1 ISO7816 |
|  |  | Ser\_P2 : 1 | ; paramètre 2 ISO7816 |
|  |  | Ser\_Len : 1 | ; longueur data ISO7816 |
|  |  | local1 : 1 | ; variable locale 1 |
|  |  | local2 : 1 | ; variable locale 2 |
|  |  | local3 : 1 | ; variable locale 3 |
|  |  | local4 : 1 | ; variable locale 4 |
|  |  | temp\_sauvw : 1 | ; sauvegarde de W pour temp |
|  | ENDC |  | ; Fin de la zone |
|  |  | ; routine ATR ; ----------- | |
| #DEFINE | cmpt1 | local1 | ; compteur d'octets pour ATR |
|  |  | ; sous-routine send et receive ; ---------------------------- | |
| #DEFINE | caract | local2 | ; caractère à envoyer |
| #DEFINE | parite | local3 | ; bit de parité |
| #DEFINE | cmptbts | local4 | ; compteur de bits |
|  |  | ; pour STATUS ; ----------- | |
| #DEFINE | status2 | local1 | ; octet 2 du status |
|  |  | ; pour instruction 25 ; ------------------- | |
| #DEFINE | cmpt2 | local1 | ; compteur d'octets |

*Voici la nomenclature des différents composants utilisés sous le logiciel altium designer :*



*Vous trouverez le schéma complet du lecteur de carte à puce ‘’phoenix’’ à la page suivante…*