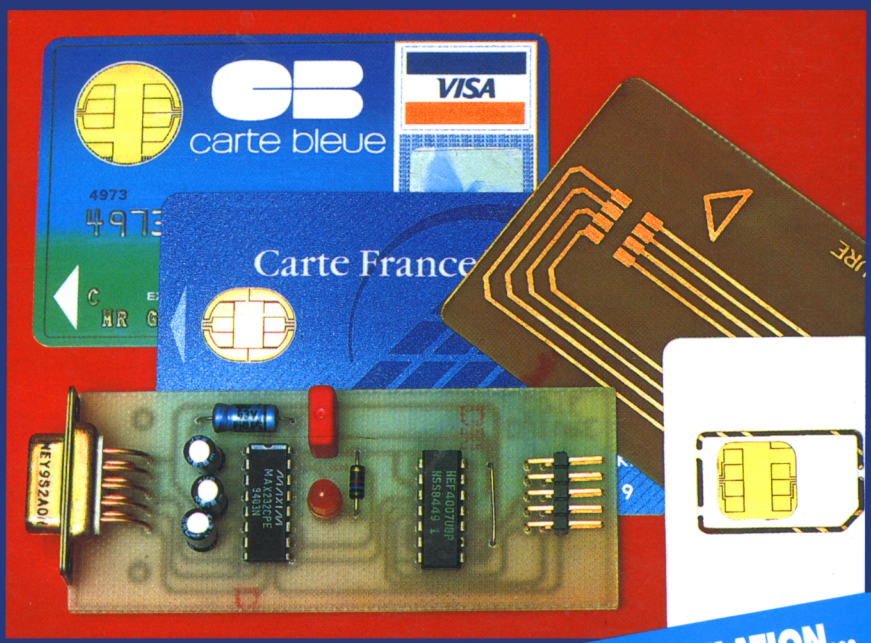
Patrick GUEULLE

PC et Cartes à Puce



LECTURE, ANALYSE, PROGRAMMATION, SIMULATION...

CE LIVRE CONTIENT UNE DISQUETTE

ETSF
EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES

PC
ET CARTES A PUCE

PATRICK GUEULLE
Ingénieur EFREI

PC
ET CARTES A PUCE

ETSF
EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES
2-12, rue de Bellevue, 75940 Paris Cedex 19
Tél. : 01 44 84 84 84

Les schémas et logiciels regroupés dans cet ouvrage et sa disquette d'accompagnement ne doivent être utilisés qu'à des fins expérimentales ou personnelles, à l'exclusion de toute exploitation commerciale ou industrielle.

Pour toute demande de licence des brevets INNOVATRON ou BULL CP8, il conviendrait de s'adresser à leurs titulaires respectifs.

© **E.T.S.F., Paris 1995**
ISBN 2 85535 239-8

TABLE DES MATIERES

AVANT-PROPOS

Quand nous avons attaqué, en 1992, la rédaction de notre ouvrage «CARTES À PUCE, Initiation et Applications», nous étions loin de nous douter que nos innocentes expériences de lecture et d'écriture dans des télécartes vides allaient nous conduire, trois ans plus tard, à percer les secrets des véritables «forteresses électroniques» que sont les cartes à microprocesseur et les nouvelles télécartes à mémoire réinscriptible.

Cela avec l'aimable complicité des fabricants et émetteurs de cartes eux-mêmes, dont la confiance est totale dans l'efficacité des mécanismes de protection de leurs applications.

Nous pouvons donc aujourd'hui montrer sans inconvénient à nos lecteurs comment transformer leur PC en un puissant outil d'exploration et même de simulation des cartes à puce les plus diverses.

Quelques montages électroniques fort simples et une disquette de logiciels très particuliers constituent tout le nécessaire pour aller vers de passionnantes découvertes. Bon voyage dans le monde fascinant des cartes à puce !

CHAPITRE PAGE

1 LES MICROPROCESSEURS DES CARTES A PUCE

MICROPROCESSEURS, MICROCALCULATEURS, OU MICROCONTROLEURS ?

C'est en 1981, autrement dit deux ans avant la mise en circulation des premières télécartes à puce, qu'est née chez BULL la première carte à microcalculateur monochip.

Il s'agissait à l'époque d'une grande première, réussie grâce à la collaboration de MOTOROLA, car depuis les premiers prototypes produits dès 1974 pour Roland Moreno, l'habitude avait été prise d'associer au moins deux puces distinctes (le microprocesseur et de la mémoire).

Entre temps, CII Honeywell-Bull prenait en 1976 la licence des brevets INNOVATRON et adoptait le terme de *carte à microcalculateur* pour désigner les produits qui, de nos jours, constituent la **galaxie CP8**.

Même si les concurrents de BULL CP8 parlent plus volontiers de *cartes à microprocesseur*, on nous permettra d'estimer que le terme le plus approprié serait *cartes à microcontrôleur*.

En effet, l'architecture des actuelles puces de cartes réunit une unité centrale 8 bits, de la mémoire ROM, RAM, EPROM et/ou EEPROM, et des ports d'entrée-sortie, exactement comme tout microcontrôleur qui se respecte.

Simplement, la tendance actuelle est d'y ajouter de plus en plus de ressources dites **sécuritaires**, toujours sur une même puce, et c'est là que pourrait bien se jouer la partie...

EXEMPLES DE RESSOURCES MATÉRIELLES

Si le marché des microcontrôleurs à *encarter* demeure assez nettement dominé par MOTOROLA, la concurrence est désormais très vive entre un nombre croissant de fabricants, conscients de l'enjeu qu'il représente : on commence en effet à produire des cartes en quantités astronomiques !

SGS-THOMSON est particulièrement bien placé, avec des produits basés sur un *cœur* compatible avec les processeurs MOTOROLA mais bénéficiant de solutions sécuritaires fort originales.

PHILIPS est de plus en plus présent à partir de son architecture 8051, avec des apports décisifs sur le plan des possibilités de calcul cryptographique.

TEXAS INSTRUMENTS, concurrent direct de SGS-THOMSON sur le marché des puces pour télécartes, propose pour sa part une gamme de microprocesseurs encartables dérivés de sa famille TMS370.

Et naturellement, les japonais s'intéressent de près à la question, à commencer par OKI avec ses produits OSCAR.

Il est intéressant de comparer sommairement ce que contiennent les microcontrôleurs de différentes marques, couramment rencontrés dans les cartes à puce.

La figure 1.1 reproduit ainsi le schéma synoptique commun à toutes les références de MOTOROLA, basées sur un très classique cœur 68HC05.

La version SC24 (3K ROM, 1K EEPROM, 128 octets RAM) équipe notamment les cartes bancaires françaises.

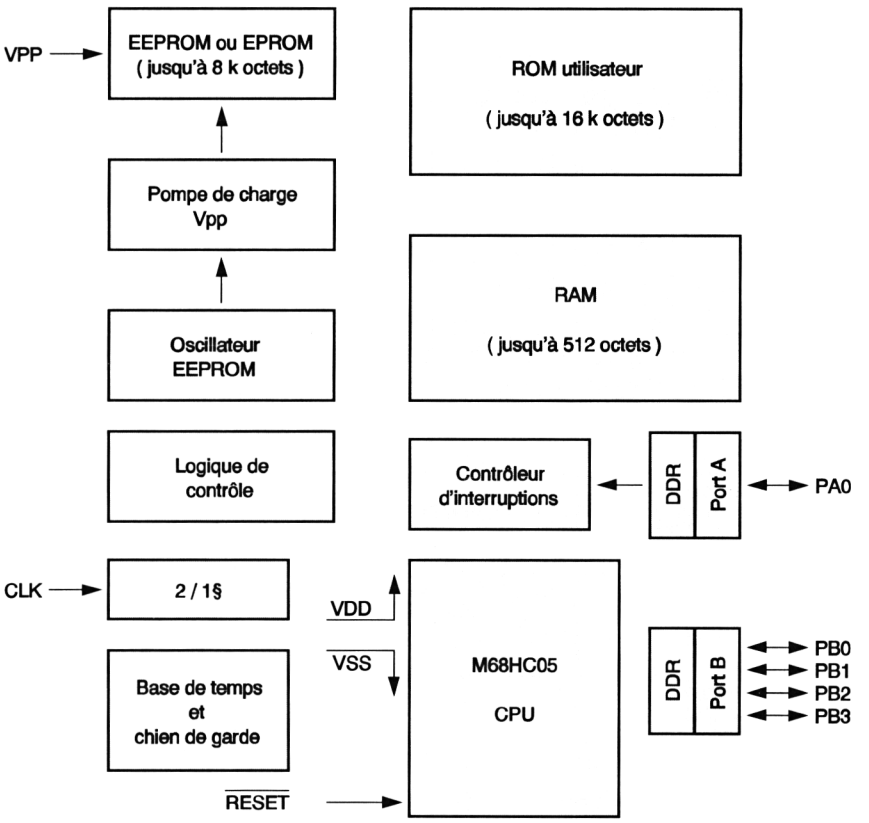


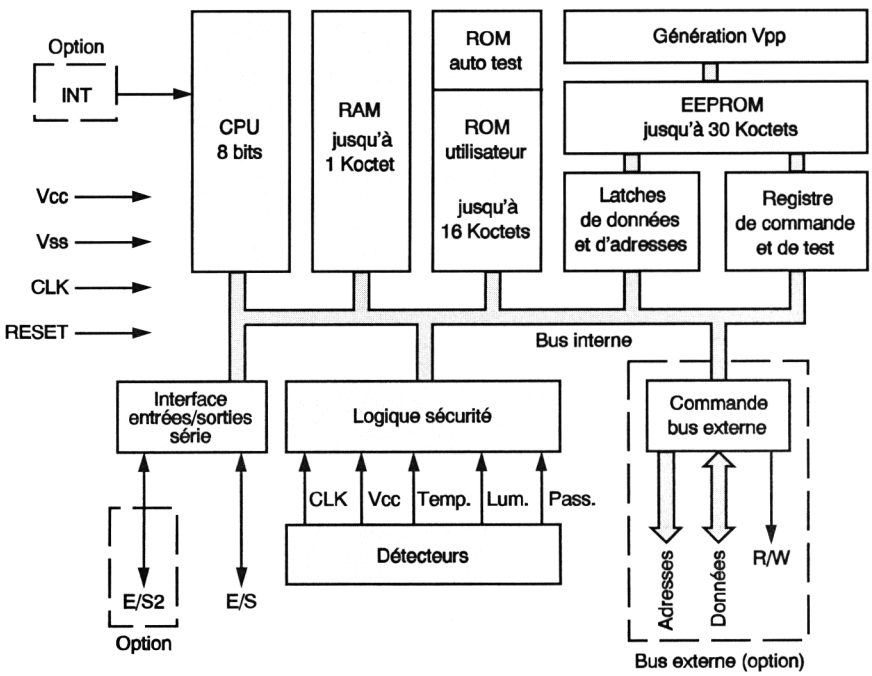
Figure 1.1. Architecture des M68HC05SC de MOTOROLA.

La figure 1.2 dévoile pour sa part l'architecture des puces ST16XYZ, cheval de bataille de SGS-THOMSON associant une unité centrale compatible 6805 à un ensemble de fonctions sécuritaires aussi bien matérielles que logicielles : logique de sécurité munie de *capteurs* détectant les tentatives d'accès frauduleux à la puce, ROM à bus d'adresses *brouillé*, système de masquage des variations de courant d'alimentation en fonction du contenu de la mémoire, matrice d'accès limitant les possibilités d'accès illicites à la mémoire, etc.

Notons que la version ST16301B équipe, en concurrence, les cartes bancaires françaises.

La figure 1.3 concerne un processeur particulièrement innovant de chez PHILIPS, le 83C852. Son unité centrale de type 8051 est secondé par une **unité de calcul** spécialisée, véritable coprocesseur rapide dédié aux opérations très particulières nécessaires aux chiffrements de données. Le tout, bien entendu, sur une seule et même puce.

Figure 1.2.
Architecture des ST16XYZ SGS-THOMSON.



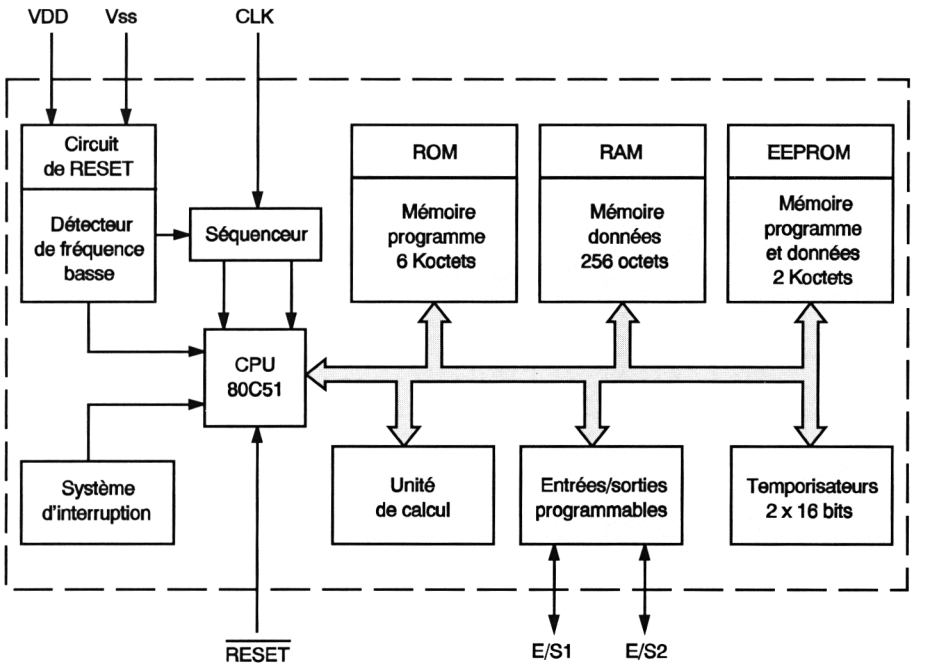


Figure 1.3.
Architecture du 83C852 de PHILIPS.

Enfin, la figure 1.4 révèle l'organisation interne d'une puce fort classique sinon très nouvelle, de TEXAS INSTRUMENTS : le TMS373C007.

C'est une version plus récente (TMS373C012) qui équipe certaines séries de cartes bancaires françaises.

On constate que, d'une marque à l'autre, la structure de base ne varie guère. Pratiquement tous les composants disposent, par exemple, de deux lignes d'entrée-sortie série alors que dans l'état actuel des choses on n'en utilise presque toujours qu'une seule (*half-duplex*).

Souvent modulables au gré du client, les capacités mémoire butent pour tout le monde sur les mêmes limites technologiques imposées par la surface de silicium pouvant être commodément et économiquement logée dans une carte (environ 4 x 6 mm), même si le recours à des filières *submicroniques* risque d'améliorer la situation à plus ou moins court terme.

Reste qu'on peut déjà loger beaucoup de choses dans 8 Ko d'EEPROM (réinscriptible) ou d'EPROM (consommable

sans possibilité de recyclage), et que 16 Ko de ROM peuvent héberger le code d'un *système d'exploitation* plus que performant.

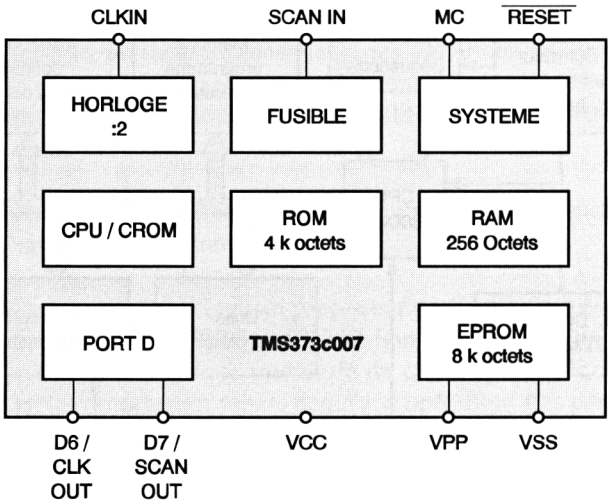


Figure 1.4.
Architecture du TMS373C007.

EXEMPLES DE RESSOURCES LOGICIELLES

Toutes les puces de cartes à microcalculateur dérivent plus ou moins directement de telle ou telle famille classique de microprocesseurs ou de microcontrôleurs.

Elles en héritent la majeure partie de leur jeu d'instructions, même s'il est habituel de l'amputer de quelques fonctions présentant peu d'utilité dans le contexte bien spécifique des cartes à puce, quitte à introduire en remplacement des instructions plus spéciales.

TEXAS INSTRUMENTS a ainsi inventé une instruction NOPV (NOP à longueur variable, très utile pour réaliser des temporisations) et une instruction de déplacement de bit, MOVB.

Nous reproduisons à la figure 1.5 le jeu d'instructions complet le plus répandu dans le monde des cartes à microprocesseur, puisque sensiblement commun aux puces SGS-THOMSON et MOTOROLA.

En principe, les manipulations courantes sur les cartes à microprocesseur ne nécessitent pas (et même ne permettent

Figure 1.5.
Le jeu d'instructions des ST16XYZ.

Register/Memory and Absolute Jump Group

Function	Mnemonic	Addressing Mode					
		Immediate	Direct	Extended	Index no offset	Index 8 bit offset	Index 16 bit offset
Load A with memory	LDA	2 A6 [2]	2 B6 [3]	3 C6 [4]	1 F6 [3]	2 E6 [4]	3 D6 [5]
Load X with memory	LDX	2 AE [2]	2 BE [3]	3 CE [4]	1 FE [3]	2 EE [4]	3 DE [5]
Load memory with A	STA	-	2 B7 [3]	3 C7 [4]	1 F7 [3]	2 E7 [4]	3 D7 [5]
Load memory with X	STX	-	2 BF [3]	3 CF [4]	1 FF [3]	2 EF [4]	3 DF [5]
Add memory to A	ADD	2 AB [2]	2 BB [3]	3 CB [4]	1 FB [3]	2 EB [4]	3 DB [5]
Add memory and carry to A	ADC	2 A9 [2]	2 B9 [3]	3 C9 [4]	1 F9 [3]	2 E9 [4]	3 D9 [5]
Subtract memory to A	SUB	2 A0 [2]	2 B0 [3]	3 C0 [4]	1 F0 [3]	2 E0 [4]	3 D0 [5]
Subtract memory with carry	SBC	2 A2 [2]	2 B2 [3]	3 C2 [4]	1 F2 [3]	2 E2 [4]	3 D2 [5]
And memory to A	AND	2 A4 [2]	2 B4 [3]	3 C4 [4]	1 F4 [3]	2 E4 [4]	3 D4 [5]
Or memory with A	ORA	2 AA [2]	2 BA [3]	3 CA [4]	1 FA [3]	2 EA [4]	3 DA [5]
Exclusive OR	EOR	2 A8 [2]	2 B8 [3]	3 C8 [4]	1 F8 [3]	2 E8 [4]	3 D8 [5]
Arithmetic Compare A	CMP	2 A1 [2]	2 B1 [3]	3 C1 [4]	1 F1 [3]	2 E1 [4]	3 D1 [5]
Arithmetic Compare X	CPX	2 A3 [2]	2 B3 [3]	3 C3 [4]	1 F3 [3]	2 E3 [4]	3 D3 [5]
Bit compare A and memory	BIT	2 A5 [2]	2 B5 [3]	3 C5 [4]	1 F5 [3]	2 E5 [4]	3 D5 [5]
Absolute Jump	JMP	-	2 BC [2]	3 CC [3]	1 FC [2]	2 EC [3]	3 DC [4]
Jump to subroutine	JSR	-	2 BD [5]	3 CD [6]	1 FD [5]	2 ED [6]	3 DD [7]

Bit manipulation and test Group

Function	Mnemonic	Addressing Mode
Bit Set	BSET b (b=0..7)	2 (10 + 2*b) [5]
Bit clear	BCLR b (b=0..7)	2 (11 + 2*b) [5]
Test bit b and branch if true	BRSET b (b=0..7)	3 (00 + 2*b) [5]
Test bit and branch if false	BRCLR b (b=0..7)	3 (01 + 2*b) [5]

Read/Modify/Write Group

Function	Mnemonic	Addressing Mode				
		Inherent A	Inherent X	Direct	Index no offset	Index 8 bit offset
Increment	INC	1 4C [2]	1 5C [2]	2 3C [5]	1 7C [5]	2 6C [6]
Decrement	DEC	1 4A [2]	1 5A [2]	2 3A [5]	1 7A [5]	2 6A [6]
Clear	CLR	1 4F [2]	1 5F [2]	2 3F [5]	1 7F [5]	2 6F [6]
One's Complement	COM	1 43 [2]	1 53 [2]	2 33 [5]	1 73 [5]	2 63 [6]
Negate (2's complement)	NEG	1 40 [2]	1 50 [2]	2 30 [5]	1 70 [5]	2 60 [6]
Rotate Left thru carry	ROL	1 49 [2]	1 59 [2]	2 39 [5]	1 79 [5]	2 69 [6]
Rotate Right thru carry	ROR	1 46 [2]	1 56 [2]	2 36 [5]	1 76 [5]	2 66 [6]
Logical shift left into Carry	LSL	1 48 [2]	1 58 [2]	2 38 [5]	1 78 [5]	2 68 [6]
Logical shift righ into Carry	LSR	1 44 [2]	1 54 [2]	2 34 [5]	1 74 [5]	2 64 [6]
Arithmetic shift right into Carry	ASR	1 47 [2]	1 57 [2]	2 37 [5]	1 77 [5]	2 67 [6]
Test for Negative or Zero	TST	1 4D [2]	1 5D [2]	2 3D [3]	1 7D [3]	2 6D [4]

Figure 1.5.
Le jeu d'instructions
des ST16XYZ.

Branch Group

Function	Mnemonic	Addressing Mode
		RELATIVE
Branch Always	BRA	2 20 [3]
Branch Never	BRN	2 21 [3]
Branch if Higher	BHI	2 22 [3]
Branch if Unsigned Lower or Same	BLS	2 23 [3]
Branch if Carry Clear	BCC	2 24 [3]
Branch if Unsigned Higher or Same	BHS	2 24 [3]
Branch if Carry Set	BCS	2 25 [3]
Branch if Unsigned Lower than	BLO	2 25 [3]
Branch if Not Equal	BNE	2 26 [3]
Branch if Equal	BEQ	2 27 [3]
Branch if Half Carry Clear	BHCC	2 28 [3]
Branch if Not Half Carry Set	BHCS	2 29 [3]
Branch if Plus	BPL	2 2A [3]
Branch if Minus	BMI	2 2B [3]
Branch if Not Interrupt Mask	BMC	2 2C [3]
Branch if Interrupt Mask	BMS	2 2D [3]
Branch if Interrupt Line Low	BIL	2 2E [3]
Branch if Interrupt Line High	BIH	2 2F [3]
Branch to Subroutine	BSR	2 AD [5]

Miscellaneous Group

Function	Mnemonic	Addressing Mode
		INHERENT
Multiply (X : A=X* A)	MUL	1 42 [10]
Transfer A to X	TAX	1 97 [2]
Transfer X to A	TXA	1 9F [2]
Transfer SP to A	TSA	1 9E [2]
Clear Carry Flag	CLC	1 98 [1]
Set Carry Flag	SEC	1 99 [1]
Clear Interrupt Mask bit	CLI	1 9A [2]
Set Interrupt Mask bit	SEI	1 9B [2]
Reset Stack Pointer	RSP	1 9C [2]
No Operation	NOP	1 9D [2]
Return from Interrupt	RTI	1 80 [2]
Return from Subroutine	RTS	1 81 [5]
Software Interrupt	SWI	1 83 [9]
Halt CPU/Enable INT	WAIT	1 8F [2]
Halt CPU/STOP Clocks/Enable INT	STOP	1 8E [2]

pas) la programmation directe de l'unité centrale. Pour des raisons de confort d'exploitation (mais aussi de sécurité !) le processeur est entièrement placé sous le contrôle d'un système d'exploitation résidant en ROM.

Mais, exception qui confirme la règle, certaines cartes dites COS possèdent une sorte de *porte d'entrée* permettant d'implanter un peu de code personnel en EPROM ou en EEPROM.

C'est là qu'une bonne connaissance du jeu d'instructions du processeur s'impose, et justement ces cartes sont souvent équipées d'un ST16CXYZ !

EXEMPLES DE RESSOURCES SÉCURITAIRES

Quelle que soit l'application envisagée, la vocation d'une carte à puce peut toujours se ramener à un stockage de données sous une forme plus ou moins protégée (ne parle-t-on pas d'ailleurs de *cartes à mémoire* ?)

La supériorité d'une carte à microprocesseur par rapport à une simple carte à mémoire, même sécurisée, tient d'une part dans son protocole de communication plus simple et normalisé, et d'autre part dans la possibilité qu'elle a de faire subir des traitements parfois complexes aux données qu'elle reçoit ou qu'elle émet.

Grâce aux ressources de la *cryptographie*, une carte à microprocesseur peut en effet fort bien ne jamais répondre deux fois de façon identique à une même sollicitation, du moins si on utilise correctement les possibilités qu'offre son système d'exploitation.

Un tel comportement complique singulièrement, c'est évident, toute tentative de *rejeu*, c'est-à-dire d'interception, puis de reproduction de la réponse de la carte.

Mais sans même aller jusque là, certaines données secrètes présentes en mémoire ne doivent en aucun cas pouvoir sortir de la carte : seul le système d'exploitation doit pouvoir y accéder, en tant qu'éléments de calculs dont il ne révélera que le résultat.

Il en va ainsi, notamment, des **codes confidentiels** attribués à l'utilisateur de la carte (*code porteur*) ou à son émetteur :

pas question de les lire comme sur une simple piste magnétique pour les comparer, à l'extérieur de la carte, au code tapé sur un clavier !

La procédure sécurisée consiste à *présenter* le code à la carte, qui va le vérifier, par l'intermédiaire de son système d'exploitation, de façon purement interne. Cette vérification effectuée, la carte peut ou bien dire tout simplement (de façon claire ou codée) si le code est bon ou faux, ou bien se contenter d'autoriser l'accès à une zone de sa mémoire qui était jusqu'à présent verrouillée.

C'est ce haut niveau de sécurité qui permet de se servir de cartes ou de clefs à microprocesseur pour des applications aussi *sensibles* que la monétique (carte bancaire, porte-monnaie électronique, etc.), la téléphonie cellulaire (GSM), ou la télévision à péage.

La figure 1.6 montre ainsi (selon SGS-THOMSON) comment sont construits les décodeurs de télévision compatibles avec les systèmes de cryptage les plus modernes.

Équipée d'une mémoire EEPROM au contenu modifiable de multiples fois, la carte contient toutes les données secrètes dont a besoin, avec celles reçues en même temps que les émissions, le microprocesseur du décodeur pour piloter son *désembrouilleur* digital.

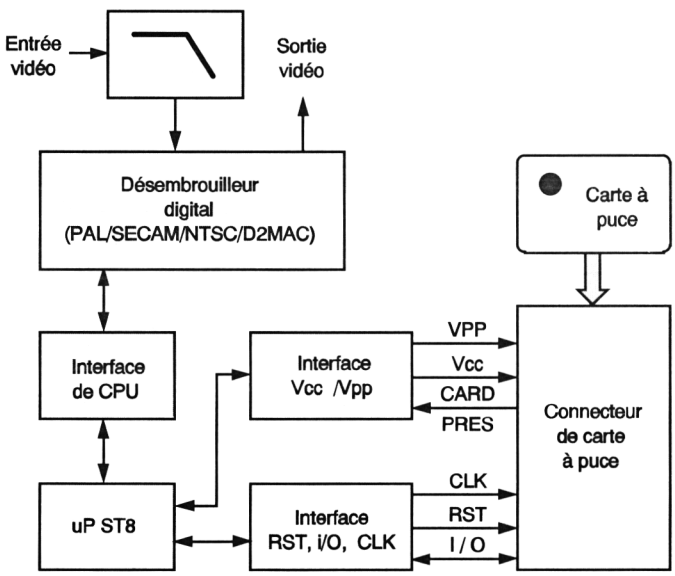


Figure 1.6.
Principe d'un décodeur TV à carte.

Le contenu de la carte peut fort bien être modifié à distance par le diffuseur de programmes, au moyen de commandes incorporées dans le signal vidéo, tandis qu'en cas de profonds changements dans le système, il est infiniment plus pratique d'envoyer de nouvelles cartes par courrier que de procéder à des échanges de décodeurs.

Il est d'ailleurs assez piquant de remarquer que les puces utilisées pour cette application sont très sensiblement plus *puissantes* que celles utilisées dans les cartes bancaires !

QUELQUES NOTIONS DE CRYPTOGRAPHIE

Le but de la cryptographie (science des *messages secrets*) est de permettre la transformation de données confidentielles en un message totalement dénué de signification pour quiconque parviendrait à s'en emparer.

On procède donc à un *chiffrage* des données à l'aide d'une *clef*, puis ultérieurement à un *déchiffrage* avec une clef qui peut être la même (algorithmes dits *symétriques*) ou une autre (algorithmes *asymétriques*).

Il est bien évident que si la clef de déchiffrage est identique à la clef de chiffrage, la protection de celle-ci pose les mêmes problèmes que celle des données elles-mêmes.

En effet, on peut évidemment décrypter un message codé si l'on connaît la clef et l'algorithme de codage, mais la chose est également réputée possible quand, ignorant tout de l'algorithme, on dispose à la fois de la clef et d'échantillons de messages en clair accompagnés de leur version codée.

L'un des procédés les plus simples de cryptage informatique consiste à effectuer un OU exclusif entre chaque octet des données à crypter et chaque octet de la clef.

Le grand intérêt de la méthode est que le même algorithme, appliqué avec la même clef, permet le décryptage.

Sous réserve d'une protection sans faille de la clef et qu'on n'utilise celle-ci qu'une seule et unique fois, même un procédé aussi simpliste apporte une protection absolue à condition que la longueur de la clef (son nombre d'octets) soit au moins égale à celle du message à coder.

Si par contre une même clef de faible longueur sert à de multiples reprises pour coder différents tronçons d'un message nettement plus long, alors toute la sûreté du cryptage est compromise.

```
10 REM ---- XOR.BAS ----
20 A$="(c)1995 Patrick GUEULLE"
30 FOR G=1 TO LEN(A$)
40 D$=MID$(A$,G,1):D=ASC(D$)
50 R=D XOR 47
55 REM clef unique = 47
70 PRINT D$,R,
80 K=R XOR 47
100 PRINT CHR$(K)
110 NEXT G
120 END
```

Le petit programme **XOR.BAS** permet de se familiariser avec cette méthode cryptographique élémentaire, et ce dans les pires conditions : la clef est en effet constituée d'un seul octet, ce qui veut dire qu'un même caractère sera toujours représenté par un même nombre (cela revient à un codage dit par *substitution*, analogue à ceux qu'utilisent les scouts).

Par contre, on conviendra que si la longueur du message se limite elle aussi à un octet, il est bel et bien impossible de se passer de la clef pour le décoder.

Le problème des clefs a été résolu de façon très élégante par le principe dit des *algorithmes à clefs publiques*.

Un tel système cryptographique utilise deux clefs distinctes : une pour le codage, et une pour le décodage.

En général, la clef de codage est *publique* et celle de décodage *secrète* : pour envoyer un message à un correspondant, on lui demande sa clef publique (à la limite on la cherche dans un *annuaire* spécial).

Même si les algorithmes de codage et de décodage sont eux aussi publics, seul le possesseur de la clef secrète pourra déchiffrer les messages chiffrés avec sa clef publique.

Mais le système peut fonctionner à l'envers : un message crypté avec la clef secrète pourra être décrypté par n'importe quel possesseur de la clef publique correspondante, qui

aura ainsi une preuve formelle de l'origine du message (on parle de *signature électronique*).

L'algorithme à clef publique le plus connu est le **RSA** (*Rivest Shamir Adleman*), qui utilise les propriétés mathématiques des *exponentiations modulo N*.

Sa mise en œuvre suppose que l'on choisisse deux nombres premiers (p et q) dont le produit (pq) servira de *modulo* pour les calculs de puissances à venir.

En termes simples, A puissance B modulo N est tout bonnement A multiplié B fois par A, moins autant de fois N qu'il faut pour arriver à un résultat inférieur à N mais encore positif.

L'algorithme *RSA* repose sur le fait que ;

$$\frac{(p-1)(q-1)}{x} = 1 \text{ modulo } pq$$

à condition toutefois que x ne soit divisible ni par p, ni par q.

En pratique, cette condition est automatiquement remplie si p et q sont supérieurs à x, ce qui est généralement le cas en cryptographie (on travaille le plus souvent sur des nombres de 512 bits !)

```
10 REM ---- MODULO.BAS ----
20 KEY OFF:CLS
30 INPUT"donnée ? ",D
40 INPUT"exposant ? ",E
50 INPUT"modulo ? ",M
60 R=D
70 FOR F=1 TO E-1
80 R=R*D
90 IF R<M THEN 110
100 R=R-M:GOTO 90
110 NEXT F
120 PRINT"RESULTAT: ";R
130 PRINT:GOTO 30
140 REM (c)1995 Patrick GUEULLE
```

Le petit programme **MODULO.BAS** est là pour permettre quelques expérimentations avec des opérandes librement choisis, et pour montrer le principe utilisé pour faire de l'exponentiation modulaire en BASIC sans trop de risques d'erreurs de dépassement (*overflow*).

Les deux clefs *RSA* (clef publique e et clef secrète d ou vice versa) doivent être choisies afin de répondre à la condition suivante ;

$$ed = 1 \text{ modulo } (p - 1)(q - 1)$$

Moyennant quoi, on a ;

$$\left(x^d\right)^e = x \text{ (modulo } pq)$$

tandis que rien ne permet de déduire d de e ou inversement.

```
10 REM ---- RSA.BAS ----
20 A$="(c)1995 Patrick GUEULLE"
30 FOR G=1 TO LEN(A$)
40 D$=MID$(A$,G,1):D=ASC(D$)
50 E=15:M=391
55 REM clef publique = 15 , modulo = 391
60 GOSUB 130
70 PRINT D$,R,
80 E=47:M=391
85 REM clef  secrète = 47 , modulo = 391
90 D=R:GOSUB 130
100 PRINT CHR$(R)
110 NEXT G
120 END
130 R=D
140 FOR F=1 TO E-1
150 R=R*D
160 IF R<M THEN 180
170 R=R-M:GOTO 160
180 NEXT F
190 RETURN
```

Le petit programme RSA.BAS applique cet algorithme dans des conditions exactement comparables aux précédentes, avec les clefs suivantes ;

- clef publique : 15, modulo 391 ;
- clef secrète : 47, modulo 391 ;

extraites des nombres premiers $p = 17$ et $q = 23$.

Bien entendu, la sûreté de l'opération n'est pas meilleure que précédemment, puisque nous opérons toujours octet par octet alors qu'il faudrait traiter des blocs de 512 bits.

Mais l'expérience a pour mérite de montrer à quel point les calculs sont lents, même sur des blocs de huit bits : c'est la raison pour laquelle on ne peut guère équiper des cartes à puce en RSA qu'avec l'assistance de coprocesseurs spécialisés, en attendant d'hypothétiques cartes à puce à cœur 32 bits.

Le 83C852, par exemple, arrive à faire le calcul en une seconde et demi, alors qu'il faudrait presque trois minutes pour l'exécuter à l'aide du jeu d'instructions de base d'un microprocesseur 8 bits...

Compte tenu de ce problème, la plupart des cartes à microcalculateur se contentent encore d'un algorithme symétrique dont la clef (ultra-secrète) est physiquement présente dans la carte, et doit par conséquent être sévèrement protégée.

Le plus courant est le DES (*Data Encryption Standard*) américain qui opère sur des blocs de 64 bits, mais on peut aussi citer le TELEPASS, algorithme développé par BULL CP8.

LA NOTION DE *MASQUE*

En micro-électronique, le terme de *masque* désigne couramment les clichés servant à réaliser les opérations de photolithographie nécessaires à la fabrication des circuits intégrés.

Dans le monde des cartes à puce, on appelle volontiers *masque* le logiciel du système d'exploitation qui, implanté en ROM, fixe le comportement des cartes à microprocesseur.

Rien de plus logique, en fin de compte, puisque cette ROM est de type *masquée*, c'est-à-dire programmée lors de la fabrication même des puces, précisément à l'aide d'un ou plusieurs masques de photogravure spécifiques.

Dans l'immense majorité des cas, une puce de carte à microprocesseur (et à plus forte raison une carte) sort de chez son fabricant irréversiblement équipée de son système d'exploitation. Pas question donc pour l'utilisateur d'y programmer son propre code écrit en assembleur, sauf *porte d'entrée* volontairement ouverte à cet effet et de toute façon placée sous le contrôle du système d'exploitation (points d'entrée, interruptions, etc.)

C'est en partie pour cette raison que nous vous expliquerons, dans un prochain chapitre, comment fabriquer vous-mêmes vos propres cartes à microcontrôleur, et comment écrire votre mini-système d'exploitation personnel.

LES MASQUES BULL CP8

La carte BULL CP8 la plus répandue est sans conteste la **M4**, dont la version **B0** n'est autre que la carte bancaire française à puce. Mais une nouvelle génération commence déjà à prendre le relais sous la forme du nouveau masque **B0'**, propriété de la communauté bancaire. Nous y reviendrons au prochain chapitre !

A partir de ce produit de base, BULL CP8 a progressivement développé toute une famille de cartes à microcalculateur, dont la figure 1.7 retrace l'évolution.

La plupart des applications non bancaires doivent maintenant être envisagées dans le cadre de la famille SCOT, dont la figure 1.8 résume la composition : de 8 à 64 Kbit (soit 1 à 8 Ko) de mémoire EPROM ou EEPROM, avec algorithme cryptographique TELEPASS ou DES incorporé.

Fondamentalement, une carte CP8 est une mémoire non volatile, associée sur la même puce à un microprocesseur qui en contrôle totalement l'accès.

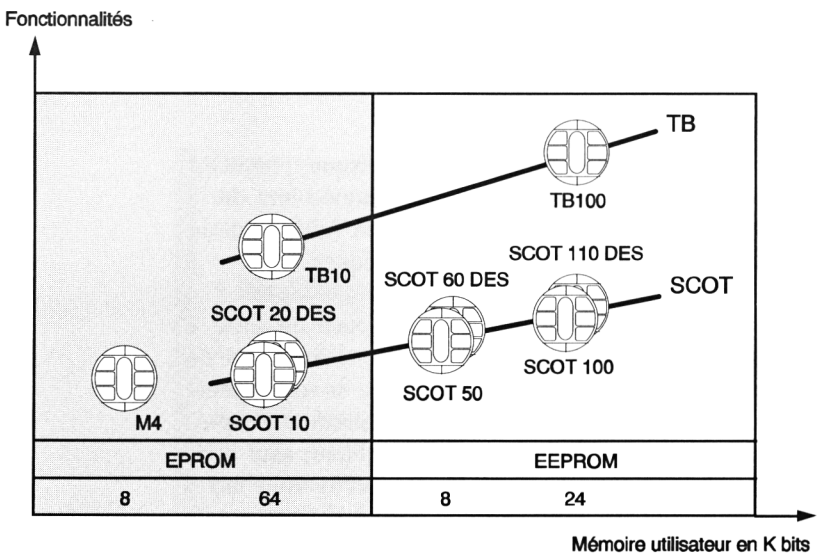
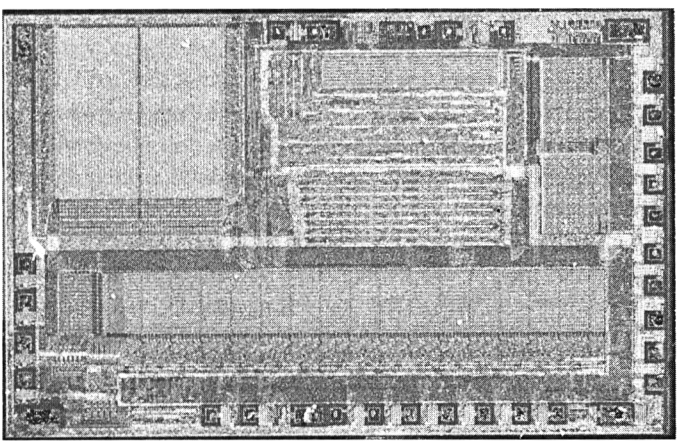


Figure 1.7.
Panorama
de la «galaxie CP8».

Quelques cartes BULL CP8 (à 6 et 8 contacts).



La puce MOTOROLA d'une carte BULL CP8 (environ 4 x 5 mm).

Ses zones les plus secrètes, en particulier, ne peuvent être lues, écrites, ou s'il y a lieu effacées que par le microprocesseur et en aucun cas directement depuis l'extérieur : on ne pourra exploiter leur contenu que de façon indirecte, par exemple en interprétant le résultat d'un calcul cryptographique dont il constitue seulement l'une des données.

SCOT 10	SCOT 20 DES	SCOT 50	SCOT 60 DES	SCOT 100	SCOT 110 DES
64 Kbits EPROM		8 Kbits EEPROM		24 Kbits EEPROM	
TELEPASS	DES	TELEPASS	DES	TELEPASS	DES
1	2	1	2	1	2

Figure 1.8.
Les caractéristiques de base des cartes SCOT. BULL CP8.

La mémoire des cartes SCOT ou M4 est organisée en un petit nombre de zones aux prérogatives bien déterminées, dont la figure 1.9 détaille l'agencement général, de moins en moins sécuritaire au fur et à mesure qu'on se déplace vers le haut de la mémoire.

Les droits d'accès à ces différentes zones dépendent du système d'exploitation interne de la carte, de l'application elle-même, et de la phase de vie de la carte (en usine, chez l'émetteur, ou dans la poche de l'utilisateur).

La figure 1.10 définit les conditions d'accès à chacune des zones, depuis la zone secrète accessible uniquement en interne par le microprocesseur, jusqu'aux zones de lecture et de fabrication qui peuvent être lues tout à fait librement par le premier venu.

C'est naturellement au développeur de l'application qu'il appartient de décider dans quelle zone il doit placer telle ou telle information, en fonction du degré de sécurité qu'il estime nécessaire.

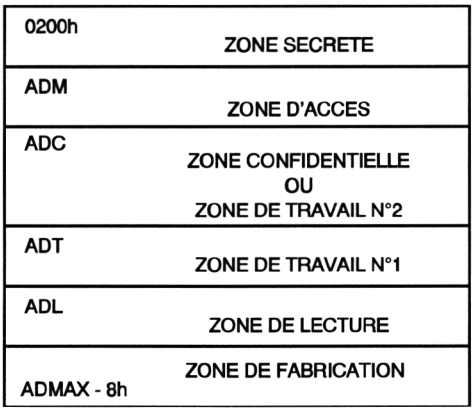


Figure 1.9.
Définition des zones mémoire des cartes BULL CP8.

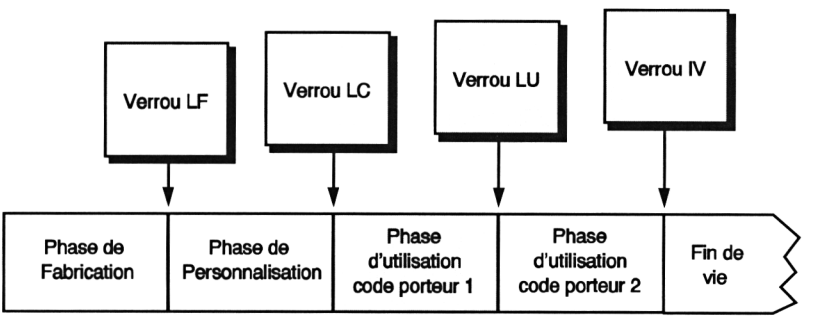
Droits d'accès aux différentes zones			
	Effacement	**Lecture**	**Ecriture**
Zone secrète	Interdit	Interdite	Interdite
Zone d'accès	Interdit	Protégée	Interdite
Zone confidentielle (option 1 zone de travail)	Interdit	Protégée	Interdite
Zone de travail 2 (option 2 zones de travail)	L'application définit les droits d'accès à cette zone		
Zone de travail 1	L'application définit les droits d'accès à cette zone		
Zone de lecture	Interdit	Libre	Interdit
Zone de fabrication	Interdit	Libre	Interdit (sauf verrous)

Figure 1.10.
Les droits d'accès aux zones mémoire des cartes BULL CP8.

Comme il faut bien pouvoir écrire dans les zones les plus secrètes lors de la fabrication des cartes puis lors de leur *personnalisation* chez l'émetteur, quatre *verrous* définis à la figure 1.11 peuvent être actionnés de façon irréversible à la fin de chaque phase de vie de la carte : le premier (LF) avant que la puce, numérotée individuellement, ne quitte l'usine, puis le second (LC) lorsque l'émetteur a terminé son travail de personnalisation.

Les deux autres verrous ne sont pas forcément utilisés au cours de la vie *normale* d'une carte : LU sert à neutraliser le code confidentiel d'origine après son remplacement par un autre (en principe à l'initiative du porteur de la carte), tandis que IV permet d'invalider définitivement une carte, par exemple en cas de détection d'une tentative de fraude.

Figure 1.11.
Les différents «verrous» des cartes BULL CP8.



Chaque zone de mémoire est organisée en mots de 32 bits (soit 4 octets) adressables au niveau de chaque quartet (chaque adresse *pointe* donc sur un mot de quatre bits et non sur un octet entier, et il faut, par conséquent, avancer de deux adresses pour passer à l'octet suivant).

Il est important de noter que ces adresses ne correspondent pas nécessairement (et en pratique quasiment jamais !) à celles directement traitées par le microprocesseur : elles sont en effet *vues* à travers le système d'exploitation, qui a toute latitude pour les transposer (ou *brouiller*) comme il l'entend.

L'adresse de début de chaque zone est contenue dans un pointeur spécifique, à l'exception de la zone secrète qui commence toujours à 0200h.

ADM marque ainsi le début de la zone d'accès, ADC le début de la zone confidentielle (ou de la seconde zone de travail), ADT le début de la zone de travail, et ADL le début de la zone de lecture.

La zone de fabrication, pour sa part, est placée juste en dessous du dernier mot de la mémoire. Elle prend donc fin à l'adresse ADMAX-8h, ADMAX étant la dernière adresse de la mémoire, variable d'un type de carte à l'autre.

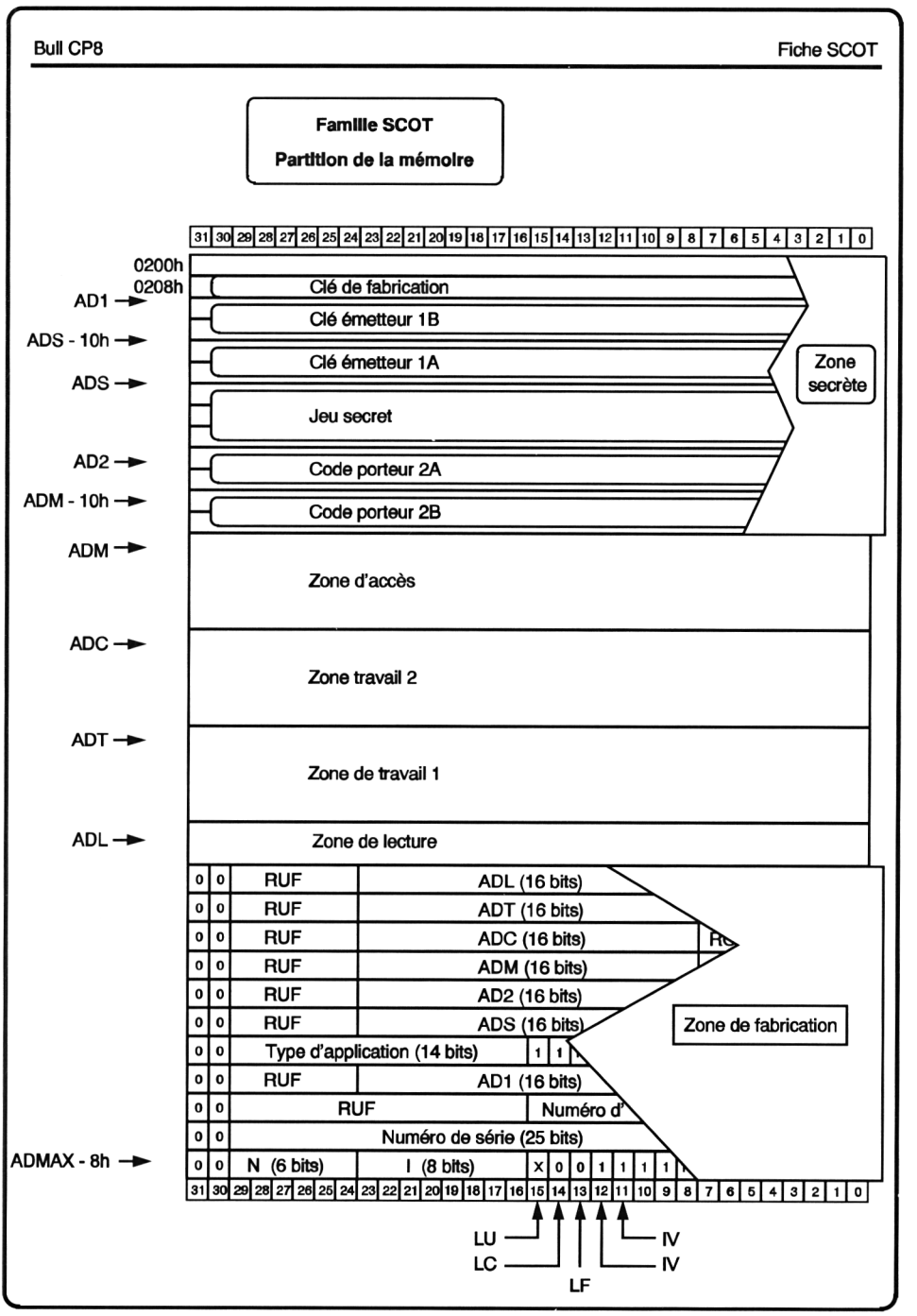
La figure 1.12, qui détaille de façon plus précise l'allocation de ces zones dans le cas général de la famille SCOT, montre que les pointeurs en question sont eux-mêmes stockés dans la zone de lecture : chacun peut donc en prendre librement connaissance, tandis que le développeur dispose ainsi de larges possibilités de fixation de la taille de telle ou telle zone.

On remarquera également que les clés et codes confidentiels résident obligatoirement dans la zone secrète, tout comme le *jeu secret* utilisé dans les calculs cryptographiques. Il ne sera donc jamais possible de les lire, même en usine.

L'accès à la mémoire se fait, sous le contrôle du microprocesseur, à partir d'un jeu d'instructions compatibles avec la norme 7816, dont la figure 1.13 donne la liste.

Conformément à cette norme, une commande envoyée à la carte par son lecteur se présentera sous la forme d'un bloc de cinq octets composé de la façon suivante ;

Figure 1.12.
Cartographie de la mémoire des cartes SCOT.



- un octet dit *classe*, égal à BCh dans le cas des cartes BULL CP8 ;
- un octet contenant le *code opération de l'instruction* ;
- deux octets précisant l'adresse à laquelle doit opérer l'instruction ;
- un octet indiquant la longueur du bloc de données devant être envoyé à la carte ou reçu de celle-ci (00h s'il n'y a pas échange de données).

Ordre disponible	INS
Lecture	B0h
Ecriture	D0h
Recherche d'un mot sur argument	A0h
Recherche du premier mot vierge	A0h
Recherche du premier non vierge	A8h
Lecture de résultat	C0h
Ecriture des verrous	50h
Effacement domaine	0Eh
Demande de nombre aléatoire	C4h
Présentation en clair de clé ou code	10h 30h 20h
Présentation clé de déblocage	20h 28h
Présentation chiffrée de clé	18h 38h
Validation de clé ou code	40h
Validation en écriture	70h
Calcul de certificat	80h
Changement code porteur	D2h

Figure 1.13.
Le jeu d'instructions des cartes SCOT.

LES MASQUES COS

Les cartes COS comptent parmi les cartes à microprocesseur les plus répandues, en raison de leur extrême souplesse de personnalisation. COS signifie *Chip Operating System* ou *Card Operating System*, par allusion au système d'exploitation chargé de la gestion de l'ensemble de la mémoire disponible, sous la protection de nombreux mécanismes de sécurité.

La figure 1.14 résume l'architecture du système interne à la carte, laquelle existe en versions EPROM et EEPROM.

Le principe EPROM, le plus ancien, ne permet pas l'effacement puis la réutilisation de zones mémoire : une *saturation* de la carte peut donc intervenir tôt ou tard.

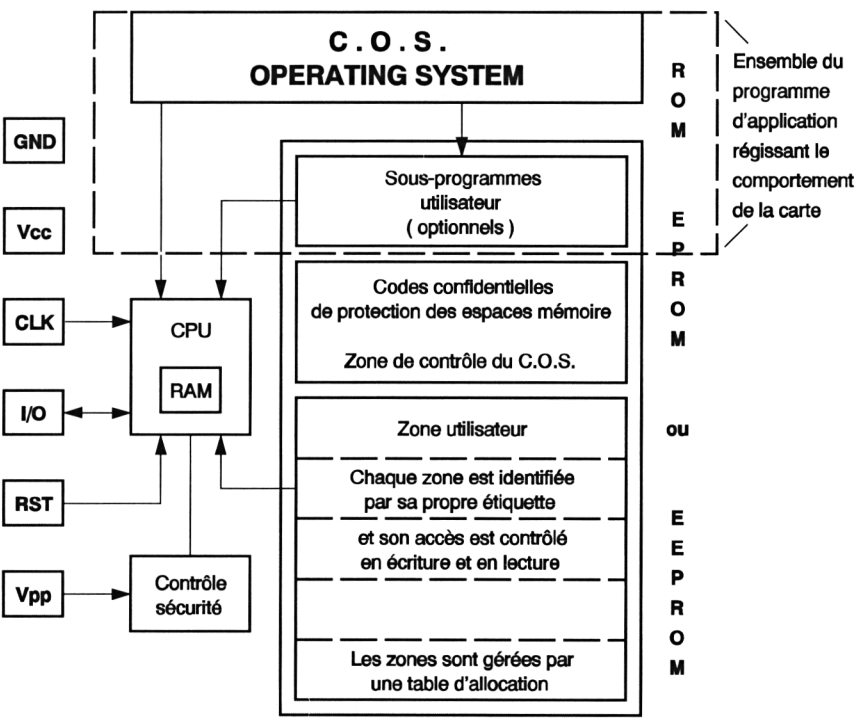
Les cartes COS EEPROM, pour leur part, sont *recyclables* presque à l'infini.

Cette technologie est appelée à se généraliser de plus en plus, car une carte à microprocesseur coûte tout de même relativement cher et doit être remplacée aussi peu souvent que possible.

Une variante de carte COS, dite MCOS, permet même d'héberger plusieurs applications totalement indépendantes dans une seule et unique carte : on parle alors de carte *multiprestataires*.

La figure 1.15 révèle le plan mémoire des cartes COS EEPROM **16 Kbit** de GEMPLUS CARD INTERNATIONAL, tandis que la figure 1.16 se rapporte au modèle 24 Kbit, deux exécutions particulièrement courantes.

Figure 1.14.
Le principe du masque «COS».



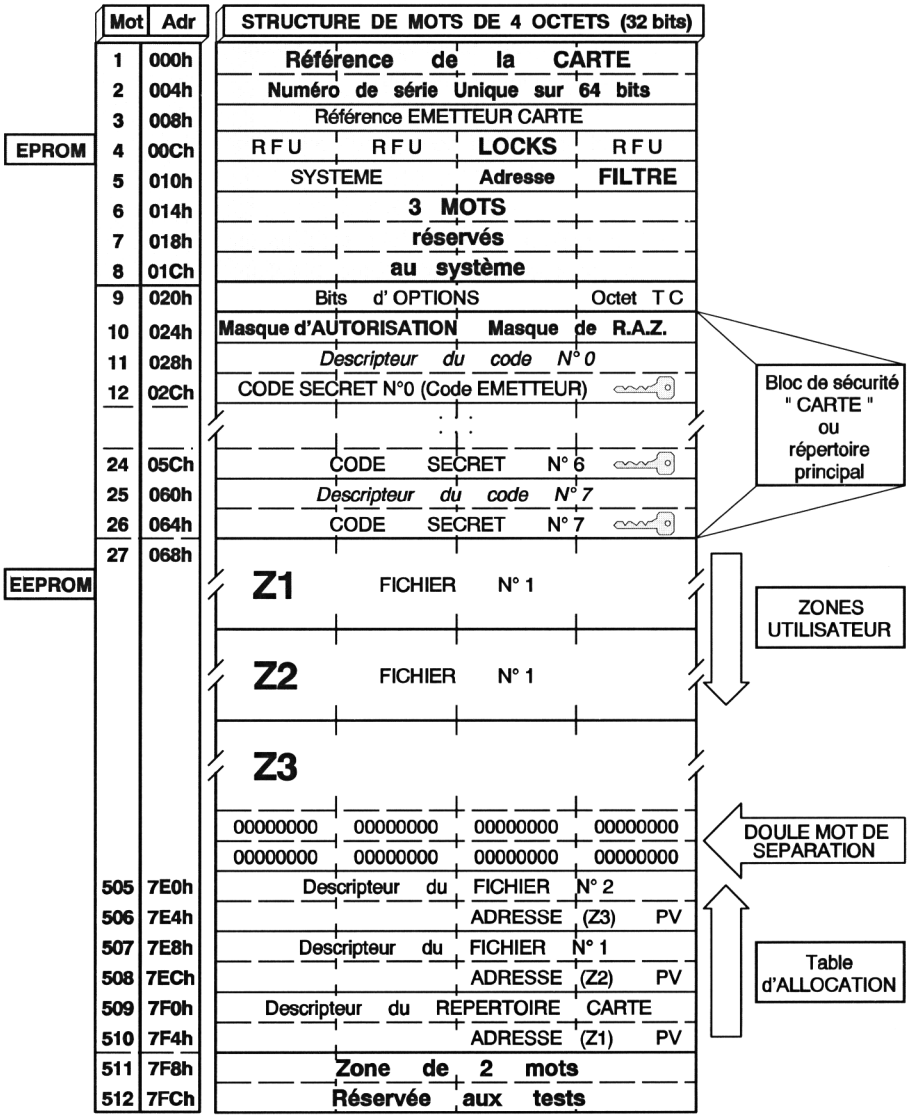


Figure 1.15.
Cartographie de la mémoire des cartes COS EEPROM 16 Kbits.

Bien que chaque octet de mémoire puisse éventuellement être lu ou écrit directement, le principe COS permet d'organiser la carte en un certain nombre de fichiers, un peu comme une disquette sous DOS (*Disk Operating System* ; la similitude est évidente...)

Bien entendu, chaque fichier peut bénéficier de multiples protections sécuritaires, précisées dans un *descripteur* approprié. En particulier, l'écriture et/ou la lecture peuvent être subordonnées à la présentation de codes confidentiels (chaque carte en supporte jusqu'à sept différents !)

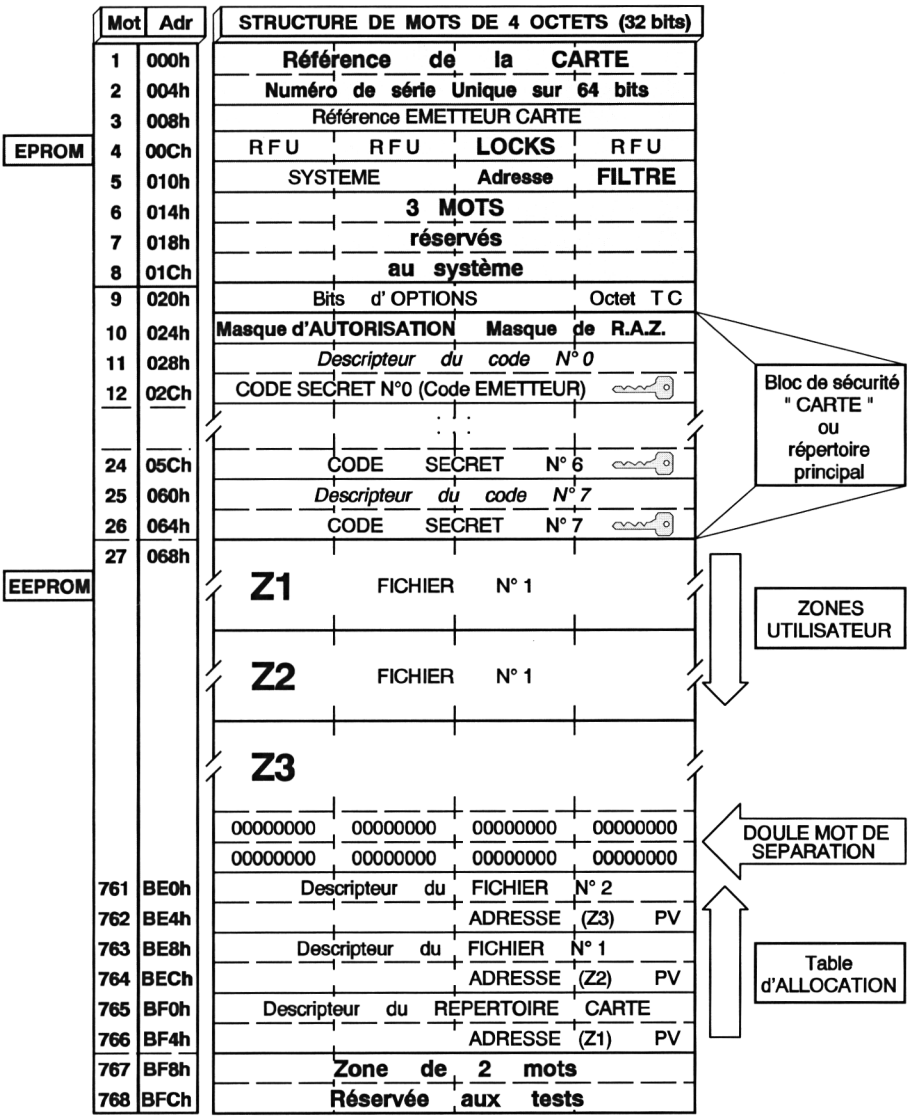


Figure 1.16.
Cartographie de la mémoire des cartes COS EEPROM 24 Kbits.

Certaines cartes COS intègrent l'algorithme de chiffrement **DES**, permettant de crypter les échanges de données entre la carte et l'extérieur : pas moyen par exemple de lire *au vol* un code confidentiel présenté ainsi à la carte !

D'un point de vue technologique, la mémoire est subdivisée en deux parties contiguës :

- une zone EPROM de 32 octets située au début de la mémoire, où seule l'écriture simple est possible. Une donnée dans cette partie de la mémoire ne peut jamais être effacée ou mise à jour, à commencer par le numéro de série individuel de chaque carte ;

- une deuxième partie qui comprend tout le reste de la mémoire, de type EEPROM. Ici, une donnée peut être effacée et réécrite à volonté.

L'ensemble de l'espace mémoire est constitué de mots de 32 bits, c'est-à-dire de 4 octets.

Dans la zone EPROM, on trouve d'abord la zone d'identification de la carte (le fameux numéro de série inscrit par le fabricant) et de l'émetteur (identifiant inscrit par l'organisme qui distribue les cartes). Il s'agit des mots 1 à 3.

La zone suivante est la **zone de contrôle du code ROM**. A cheval sur les zones EPROM et EEPROM, elle contient six mots (4 à 9) :

- le mot 4 héberge les **verrous** (*locks*) de la carte, et notamment les bits BFAB et BPERS décrits à la figure 1.17, dont la mise à 1 est irréversible.

Dans une carte en cours de fabrication (au sein de l'usine) les bits BFAB et BPERS sont tous deux à zéro, et le mot 4 vaudra par exemple 00h. Pour l'instant, la sécurité de la carte est pratiquement nulle.

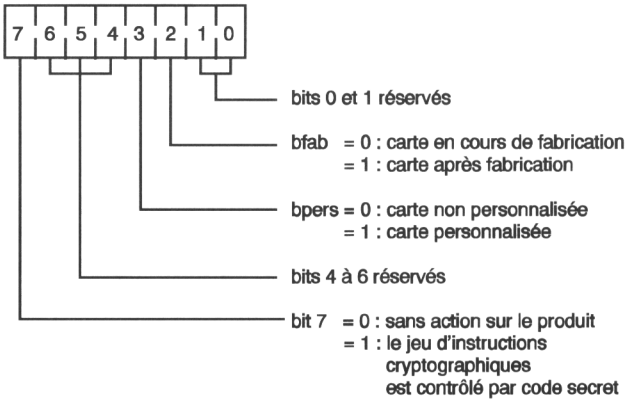


Figure 1.17.
Les différents «verrous» des cartes COS.

Dans toute carte quittant l'usine (avec un numéro de série) mais pas encore *personnalisée* par l'émetteur, BFAB est à 1 et BPERS à 0 : le mot 4 vaudra alors 04h. C'est notamment le cas des cartes échantillon.

Lorsque l'émetteur aura terminé son travail de personnalisation, il mettra le bit BPERS à 1 : le mot 4 vaudra alors 0Ch, valeur la plus probable à rencontrer lorsqu'on manipule avec des cartes mises en circulation dans le public ;

- le mot 5, appelé *filtre*, contient le point d'entrée d'un programme applicatif éventuellement implanté en mémoire EEPROM. Si les deux derniers octets du mot 5 sont 00h, alors le programme standard du COS est exécuté.

Il s'agit là d'une possibilité extrêmement puissante du COS, permettant en fait au développeur de conférer à la carte un comportement qui ne dépend pratiquement que du programme qu'il va écrire, à partir du jeu d'instructions du microprocesseur de la carte (en général un ST16XYZ de SGS-THOMSON) ;

- les mots 6 à 8 sont réservés au système ;

- le mot 9 contient toutes les options qui vont régir le comportement du code ROM de base : type de protocole de transmission, mode d'allocation des fichiers, classe d'application, certains éléments de la réponse au reset, etc.

Vient ensuite, à partir du mot 10 (adresse 024h), le *bloc de sécurité* de la carte, contenant notamment les codes confidentiels et leurs descripteurs.

Dès la fin du bloc de sécurité commencent les fichiers, contrôlés par la *table d'allocation physique* contenue en fin du même espace mémoire, et que l'on peut comparer à la FAT d'une disquette.

Enfin, deux *mots de test* sont placés tout à fait en haut de la mémoire : 64 bits que l'on peut venir lire et écrire tout à fait librement sans aucune répercussion par ailleurs.

2 A LA DÉCOUVERTE DE LA CARTE BANCAIRE

L'application *carte bancaire* constitue un bon terrain d'expérimentation autour des cartes à microprocesseur, surtout quand on a eu la (bonne ?) idée de ne pas restituer ni détruire ses cartes périmées.

Mais à défaut, il faut savoir que les manipulations qui vont être décrites peuvent fort bien être menées sur des cartes en cours de validité, puisque nous nous limiterons évidemment à des opérations de lecture.

Par ailleurs, il faut savoir que les cartes *FRANCE TELECOM* (anciennement PASTEL) sont très proches des cartes bancaires et peuvent aussi servir de matière première.

Précisons simplement qu'en tout état de cause, chacun devra assumer la pleine et entière responsabilité de ses propres tentatives, quelles qu'elles soient, compte tenu de leur caractère totalement expérimental.

UNE VARIANTE DU MASQUE BULL CP8 M4

L'important pour s'attaquer à la lecture des cartes bancaires est de se souvenir que la **classe ISO** de toutes les cartes BULL CP8 est BCh, et qu'en principe, le pointeur ADL vaut 08E0h pour toutes les cartes bancaires françaises et pour les cartes PASTEL.

C'est suffisant pour prendre connaissance, avec un PC équipé d'un lecteur approprié, des données dont la lecture est libre.

Mais à partir du moment où on possède le code confidentiel de la carte servant de cobaye (quoi de plus normal quand on en est le titulaire ?) il est fort tentant de s'en servir pour déverrouiller certaines zones déjà mieux protégées.

Toujours en principe, ADC et ADT valent 02E0h (il n'y aurait donc pas de *zone confidentielle*) et ADM est à 0260h sur les *vieilles* cartes bancaires à mémoire EPROM (masque dit B0).

Sur les nouvelles cartes à mémoire EEPROM (masque dit B0'), ADC et ADT semblent fixés à 0280h, ADM restant à 0260h. Cela trahit une zone d'accès (dite ici *zone d'état*) de taille sensiblement inférieure, conséquence de la technologie EEPROM employée qui, permettant le *recyclage* de cette zone, supprime le risque de saturation dont ont eu à se plaindre les gros utilisateurs de cartes dans les années passées.

Il est facile de distinguer les cartes B0 des cartes B0' par le fait que ces dernières portent normalement une **puce** en position ISO (dite *centrée*), et ne possédant que six contacts au lieu de huit.

Par contre, et même muni du bon code confidentiel, il ne faudra pas espérer pouvoir accéder à la *zone secrète* : en présence de tentatives de ce genre, la carte adoptera tout simplement un farouche mutisme, et c'est heureux !

Il va en effet de soi que si tel n'avait pas été le cas, nous n'aurions jamais écrit ce chapitre ni même probablement ce livre ...

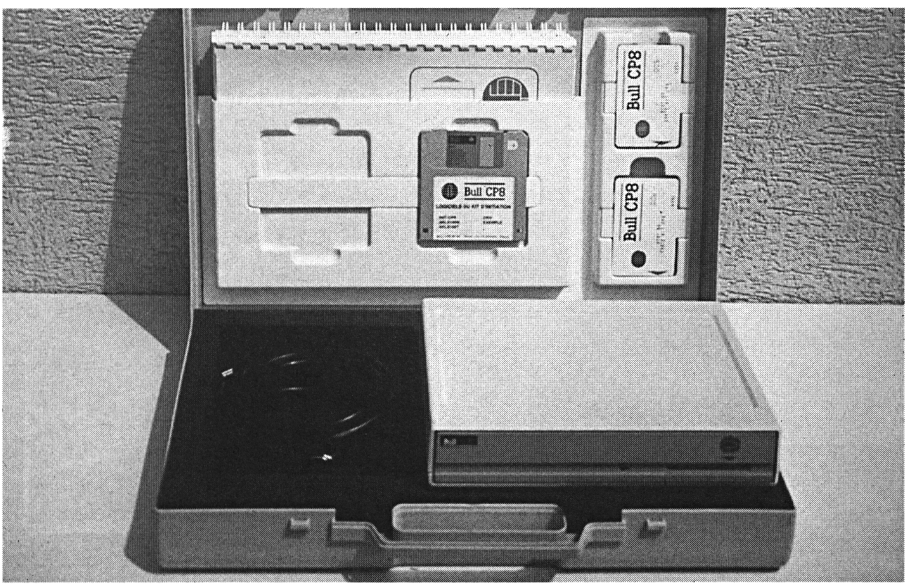
UN LECTEUR DE CARTES À MICROPROCESSEUR

Une solution commode pour s'attaquer aux cartes bancaires (et plus généralement à toutes les cartes CP8) est disponible sous la forme du kit d'initiation mis au point par BULL, et commercialisé par TEKELEC (5 rue Carle Vernet 92310 SEVRES).

Pour le prix d'un compatible PC moyen, cette mallette contient un lecteur TLP224, une vingtaine de cartes SCOT50, un jeu de manuels et toute une bibliothèque de logiciels.

Il y a donc là non seulement de quoi apprendre à maîtriser convenablement cette famille de cartes, mais aussi tous les éléments d'une plate-forme de développement suffisante pour prototyper de véritables applications dans les règles de l'art.

Le kit d'initiation de BULL CP8.



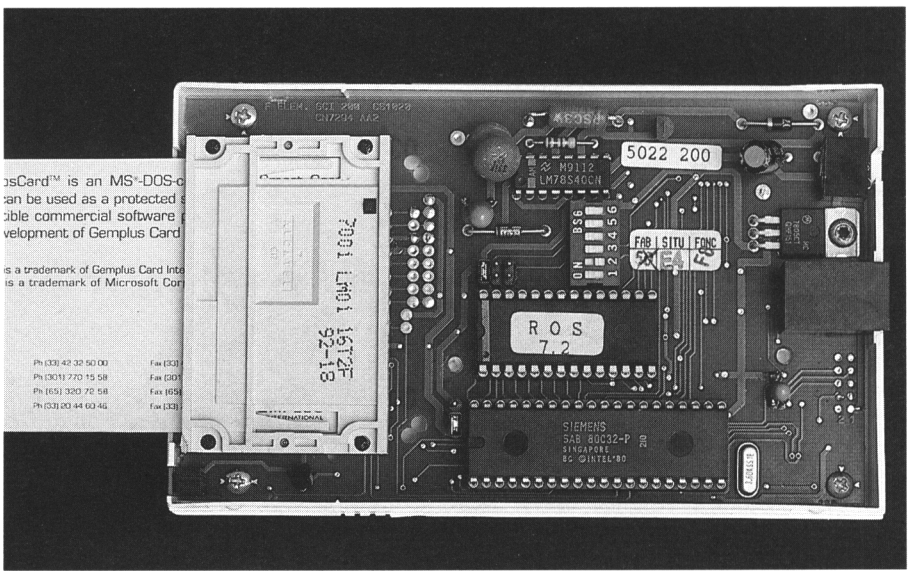
La documentation fournie s'articule autour du *manuel d'utilisation de la famille SCOT*, gros ouvrage de référence contenant tout ce qu'on peut souhaiter savoir sur ces cartes. Sa lecture nous a été extrêmement utile pour mener à bien les recherches dont nous présentons ici les résultats.

Mais une grande part de l'intérêt du kit réside dans son contenu logiciel : parallèlement à quelques programmes à caractère surtout démonstratif, on appréciera à leur juste valeur les drivers et bibliothèques en langage C qui faciliteront considérablement le développement d'applications réelles.

Vu sous cet angle, le prix du kit peut être considéré comme raisonnable, alors qu'une dépense de cet ordre ne saurait se justifier par le seul souci de satisfaire une simple curiosité.

L'approche que nous allons privilégier ici repose donc plutôt sur la construction d'un lecteur économique, en l'occurrence celui présenté dans notre livre *Cartes à puce, Initiation et Applications* paru dans la même collection.

Tous les logiciels pour cartes à microprocesseur réunis dans le présent ouvrage et sur sa disquette d'accompagnement ont été spécifiquement écrits pour lui, mais on pourrait songer à les transformer pour d'autres lecteurs (par exemple le GCR200 GEMPLUS, particulièrement courant).



Le lecteur GCR200 de GEMPLUS CARD INTERNATIONAL.

Mais pour que cet ouvrage puisse se suffire à lui-même, nous reproduisons ici les plans de notre lecteur bâti, rappelons le, autour d'un *coupleur* développé par INNOVATRON et commercialisé (sous la référence CCU910) par COREL ELECTRONIQUE (37 avenue Raspail, 94253 GENTILLY CEDEX).

La figure 2.1 reprend donc le tracé du circuit imprimé à graver, et la figure 2.2 son plan de câblage.

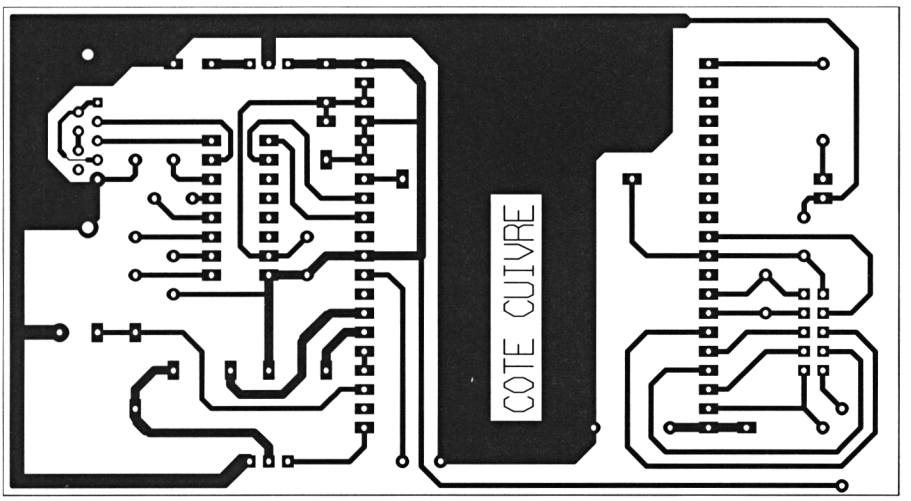


Figure 2.1.
Le circuit imprimé du lecteur universel.

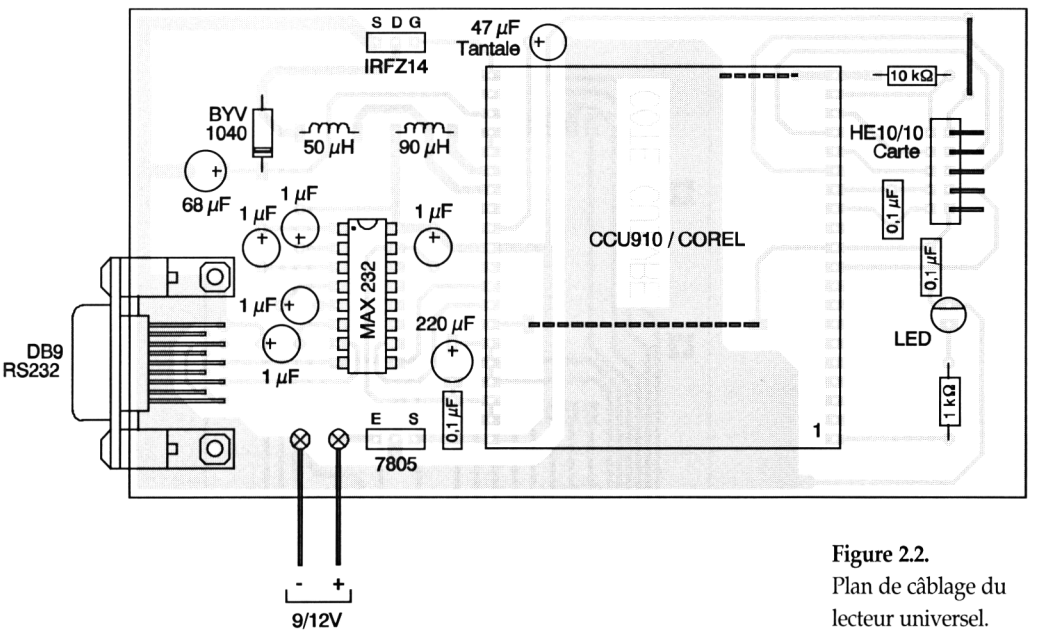


Figure 2.2.
Plan de câblage du lecteur universel.

Attention : l'embase DB9 (liaison RS232 avec le port COM1 : du PC) doit impérativement être du type *femelle*, et recevoir un câble du type *rallonge de moniteur* (mâle-femelle). Une embase mâle associée à un câble *double femelle* ne conviendrait pas !

Précisons que les deux selfs nécessaires (50 et 90 µH) peuvent être réalisées en bobinant respectivement 9 et 12 spires de fil émaillé 7/10 sur des tores ferrite de 650 nH/sp^2.

Reste maintenant à construire un connecteur de carte qui sera relié au circuit de lecture par un câble méplat équipé de deux fiches HE10 à dix contacts, tous reliés en parallèle fil à fil.

La figure 2.3 fournit le tracé du cuivre nécessaire, et la figure 2.4 le plan de câblage correspondant.

Notons que deux réceptacles pour fiches HE10 sont prévus (en l'occurrence deux tronçons de barrette sécable à double rangée de picots carrés coudés) : l'un correspond aux cartes à puce en position ISO (centrée) et l'autre aux puces en position AFNOR (excentrée). Il conviendra naturellement d'utiliser le bon connecteur pour chaque carte !

Notons que ce matériel ne servira pas qu'à lire les cartes bancaires, mais qu'il sera nécessaire pour beaucoup d'autres expériences qui vous attendent dans les prochains

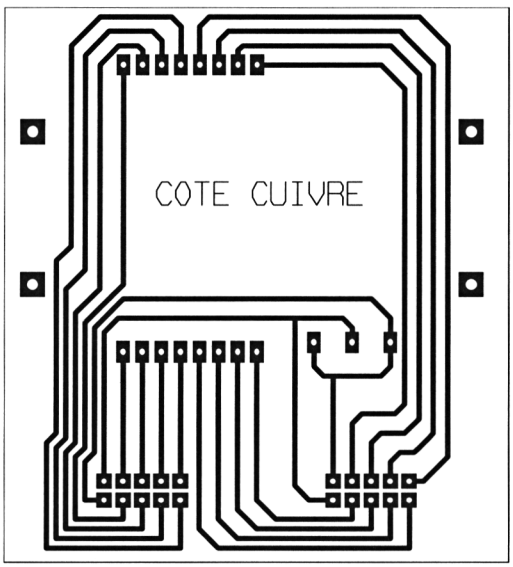
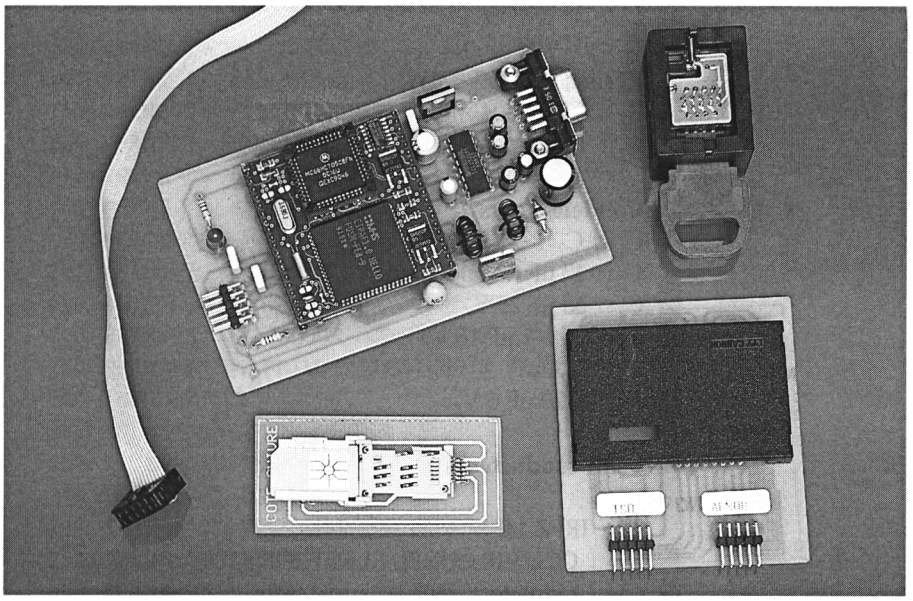


Figure 2.3.
Le circuit imprimé du connecteur de cartes ISO/AFNOR.



Notre lecteur universel et ses connecteurs adaptables.

chapitres. Il fait partie d'une *boîte à outils pour cartes à puce* décrite dans notre ouvrage déjà cité, et dont il est éminemment souhaitable de se doter pour travailler sérieusement.

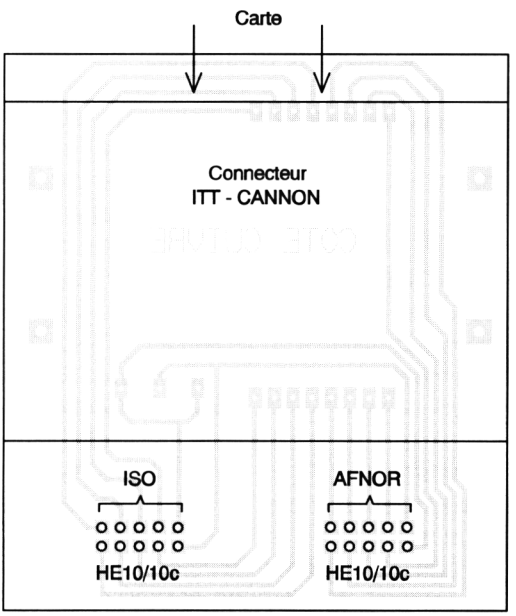


Figure 2.4.
Plan de câblage du connecteur de cartes.

Liste des composants

Résistances (5% 1/4W) :
R_1 : 1 kΩ,
R_2 : 10 kΩ.

Condensateurs :
C_1 et C_2 : 0,1 µF,
C_3 : 47 µF tantale goutte 10 V,
C_4 : 68 µF 25 V,
C_5 à C_9 : 1 µF 16 V,
C_{10} : 0,1 µF,
C_{11} : 220 µF 6 V.

Semi-conducteurs :

T_1 : IRFZ 14,
CI_1 : CCU 910 COREL ELECTRONIQUE,
CI_2 : régulateur 7805,
CI_3 : MAX 232,
D_1 : BYV 10-40,
D_2 : LED rouge.

1 self 90 µH (voir texte),
1 self 50 µH (voir texte),
1 embase coudée DB9 femelle pour circuit imprimé,
1 barrette sécable à doubles picots coudés,
1 barrette sécable tulipe,
1 connecteur de carte ITT-CANNON,
25 cm de câble méplat 10 conducteurs,
2 fiches HE10 à 10 contacts, à sertir,
1 bloc d'alimentation 9 à 12 V continu.

```
10 REM — COREL —
20 KEY OFF: CLS:BEEP
30 OPEN"com1:9600,n,8,1" AS #1
40 PRINT"COUPLEUR DE CARTES A MEMOIRE CCU910 COREL"
50 PRINT"========================================="
60 PRINT:PRINT:PRINT"INTRODUIRE UNE CARTE, puis presser ENTER"
70 INPUT Z$:CLS
80 PRINT:PRINT"CLASSE ISO DE LA CARTE ? (RETURN si inconnue)"
90 INPUT S$
100 IF LEN(S$)<2 THEN S$="0"+S$
110 IF LEN(S$)<2 THEN S$="0"+S$
```

```
120 CLS
130 PRINT "0 -> QUITTER"
140 PRINT "1 -> MISE SOUS TENSION et RESET"
150 PRINT "2 -> MISE HORS TENSION"
160 PRINT "3 -> MODE GPM256"
170 PRINT "4 -> LECTURE"
180 PRINT "5 -> ECRITURE"
190 PRINT "6 -> PRESENTATION DE CODE PORTEUR"
200 PRINT"7 -> EFFACEMENT DE MOT (2 octets)"
210 PRINT"8 -> PRESENTATION CODE EMETTEUR ET EFFACEMENT"
220 PRINT"9 -> AUTRES COMMANDES ISO"
230 PRINT:PRINT:PRINT"VOTRE CHOIX puis ENTER"
240 INPUT Z$:CLS
250 IF Z$="0" THEN SYSTEM
260 IF Z$="1" THEN 360
270 IF Z$="2" THEN 420
280 IF Z$="3" THEN 450
290 IF Z$="4" THEN 480
300 IF Z$="5" THEN 620
310 IF Z$="6" THEN 790
320 IF Z$="7" THEN 1010
330 IF Z$="8" THEN 1100
340 IF Z$="9" THEN 1260
350 GOTO 240
360 PRINT#1,CHR$(2)+CHR$(2)+CHR$(0)+CHR$(2)+CHR$(3);
370 GOSUB 1750
380 IF E$="03" THEN PRINT "CARTE ASYNCHRONE DETECTEE"
390 IF E$="04" THEN PRINT "CARTE SYNCHRONE DETECTEE"
400 PRINT
410 GOTO 130
420 PRINT#1,CHR$(2)+CHR$(3)+CHR$(0)+CHR$(3)+CHR$(3);
430 GOSUB 1750
440 GOTO 130
450 PRINT#1,CHR$(2)+CHR$(12)+CHR$(1)+CHR$(1)+ CHR$(12)+CHR$(3);
460 GOSUB 1750
470 GOTO 130
480 PRINT "NOMBRE D'OCTETS A LIRE ?"
490 INPUT Q
500 IF Q>255 THEN 490
510 Q$=HEX$(Q)
520 IF LEN(Q$)>1 THEN 540
530 Q$="0"+Q$:GOTO 520
540 PRINT"A PARTIR DE L'ADRESSE (hexa) ?"
550 INPUT A$:PRINT
560 IF LEN(A$)>3 THEN 580
570 A$="0"+A$:GOTO 560
580 X$="020005"+S$+"B0"+A$+Q$
```

```
590 GOSUB 1980
600 GOSUB 1750
610 GOTO 130
620 PRINT"ADRESSE (hexa) DE DEBUT D'ECRITURE ?"
630 INPUT A$
640 IF LEN(A$)>3 THEN 660
650 A$="0"+A$:GOTO 640
660 PRINT:PRINT"DONNEES (hexa) A ECRIRE ?"
670 INPUT H$
680 L=LEN(H$)/2
690 U$=HEX$(5+L)
700 IF LEN(U$)<2 THEN U$="0"+U$
710 IF (L-INT(L))<>0 THEN BEEP:GOTO 670
720 L$=HEX$(L)
730 IF LEN(L$)<2 THEN L$="0"+L$
740 X$="0201"+U$+S$+"D0"+A$+L$+H$
750 GOSUB 1980
760 FOR T=0 TO 5000:NEXT T
770 GOSUB 1750
780 GOTO 130
790 PRINT"ADRESSE (hexa) DU CODE A PRESENTER ?"
800 INPUT A$
810 IF LEN(A$)>3 THEN 830
820 A$="0"+A$:GOTO 810
830 PRINT:PRINT"CODE A PRESENTER ?"
840 INPUT H$
850 L=LEN(H$)/2
860 U$=HEX$(5+L)
870 IF LEN(U$)<2 THEN U$="0"+U$
880 IF (L-INT(L))<>0 THEN BEEP:GOTO 840
890 L$=HEX$(L)
900 IF LEN(L$)<2 THEN L$="0"+L$
910 X$="0201"+U$+S$+"20"+A$+L$+H$
920 GOSUB 1980
930 GOSUB 1750
940 IF M$="9000" THEN PRINT"CODE BON"
950 IF M$="9010" THEN PRINT"CODE FAUX: 1er essai":BEEP
960 IF M$="9020" THEN PRINT"CODE FAUX: 2ème essai":BEEP
970 IF M$="9040" THEN PRINT"CODE FAUX: 3ème essai":BEEP
980 IF M$="9080" THEN PRINT"CODE FAUX: CARTE BLOQUEE!":BEEP
990 PRINT:PRINT
1000 GOTO 130
1010 PRINT "EFFACEMENT DE 2 OCTETS"
1020 PRINT"AUTOUR DE L'ADRESSE (hexa) ?"
1030 INPUT A$:PRINT
1040 IF LEN(A$)>3 THEN 1060
1050 A$="0"+A$:GOTO 1040
```

```
1060 X$="020105"+S$+"D8"+A$+"00"
1070 GOSUB 1980
1080 GOSUB 1750
1090 GOTO 130
1100 PRINT"ADRESSE (hexa) DU CODE A PRESENTER ?"
1110 INPUT A$
1120 IF LEN(A$)>3 THEN 1140
1130 A$="0"+A$:GOTO 1120
1140 PRINT:PRINT"CODE A PRESENTER ?"
1150 INPUT H$
1160 L=LEN(H$)/2
1170 U$=HEX$(5+L)
1180 IF LEN(U$)<2 THEN U$="0"+U$
1190 IF (L-INT(L))<>0 THEN BEEP:GOTO 1150
1200 L$=HEX$(L)
1210 IF LEN(L$)<2 THEN L$="0"+L$
1220 X$="0201"+U$+S$+"10"+A$+L$+H$
1230 GOSUB 1980
1240 GOSUB 1750
1250 GOTO 130
1260 CLS:PRINT:PRINT:PRINT
1270 PRINT"0 -> ANNULATION"
1280 PRINT"1 -> COMMANDE ISO ENTRANTE (données -> carte)"
1290 PRINT"2 -> COMMANDE ISO SORTANTE (données <- carte)"
1300 PRINT:PRINT:PRINT"VOTRE CHOIX puis ENTER"
1310 INPUT Z$:CLS
1320 IF Z$="0" THEN CLS:GOTO 130
1330 IF Z$="1" THEN 1360
1340 IF Z$="2" THEN 1570
1350 GOTO 130
1360 PRINT"CODE ISO DE LA COMMANDE ?"
1370 INPUT I$
1380 IF LEN(I$)<>2 THEN 1370
1390 PRINT"ADRESSE (hexa) ?"
1400 INPUT A$
1410 IF LEN(A$)>3 THEN 1430
1420 A$="0"+A$:GOTO 1410
1430 PRINT:PRINT"DONNEES (hexa) ?"
1440 INPUT H$
1450 L=LEN(H$)/2
1460 U$=HEX$(5+L)
1470 IF LEN(U$)<2 THEN U$="0"+U$
1480 IF (L-INT(L))<>0 THEN BEEP:GOTO 1440
1490 L$=HEX$(L)
1500 IF LEN(L$)<2 THEN L$="0"+L$
1510 X$="0201"+U$+S$+I$+A$+L$+H$
1520 GOSUB 1980
```

```
1530 FOR T=0 TO 5000:NEXT T
1540 GOSUB 1750
1550 IF M$="9000" THEN PRINT"EXECUTION CORRECTE":PRINT
1560 GOTO 130
1570 PRINT"CODE ISO DE LA COMMANDE ?"
1580 INPUT I$
1590 IF LEN(I$)<>2 THEN 1580
1600 PRINT "LONGUEUR ?"
1610 INPUT Q
1620 IF Q>255 THEN 1580
1630 Q$=HEX$(Q)
1640 IF LEN(Q$)>1 THEN 1660
1650 Q$="0"+Q$:GOTO 1640
1660 PRINT"ADRESSE OU REFERENCE (hexa) ?"
1670 INPUT A$:PRINT
1680 IF LEN(A$)>3 THEN 1700
1690 A$="0"+A$:GOTO 1680
1700 X$="020005"+S$+I$+A$+Q$
1710 GOSUB 1980
1720 GOSUB 1750
1730 IF M$="9000" THEN PRINT"EXECUTION CORRECTE":PRINT
1740 GOTO 130
1750 FOR T=0 TO 5000:NEXT T
1760 R$=""
1770 IF LOC(1)=0 THEN PRINT"COUPLEUR MUET":GOTO 1970
1780 C$=INPUT$(LOC(1),#1)
1790 J=0:PRINT"REPONSE DU COUPLEUR:"
1800 FOR F=1 TO LEN(C$)
1810 D$=HEX$(ASC(MID$(C$,F,1)))
1820 IF LEN(D$)<2 THEN D$="0"+D$
1830 R$=R$+D$
1840 PRINT D$+CHR$(32);:J=J+1
1850 IF J>23 THEN J=0:PRINT
1860 NEXT F
1870 PRINT
1880 M$=MID$(R$,9,4)
1890 E$=MID$(R$,7,2)
1900 IF M$="9001"THEN PRINT"PROBLEME D'ECRITURE":BEEP
1910 IF M$="6E00"THEN PRINT"CLASSE ISO INCORRECTE":BEEP
1920 IF E$="02"THEN PRINT"CARTE NON TRAITEE":BEEP
1930 IF E$="FD"THEN PRINT"CARTE ARRACHEE":BEEP
1940 IF E$="FF"THEN PRINT"CARTE MUETTE":BEEP
1950 IF E$="FA"THEN PRINT"CARTE ABSENTE":BEEP
1960 IF E$="FB"THEN PRINT"COURT-CIRCUIT CARTE":BEEP
1970 PRINT:PRINT:RETURN
1980 L=0
1990 PRINT"TRAME ENVOYEE AU COUPLEUR:"
```

```
2000 FOR F=3 TO (LEN(X$)-1) STEP 2
2010 W$="&h"+MID$(X$,F,2)
2020 L=L XOR VAL(W$)
2030 L$=HEX$(L)
2040 IF LEN(L$)<2 THEN L$="0"+L$
2050 NEXT F
2060 X$=X$+L$+"03"
2070 FOR F=1 TO (LEN(X$)-1) STEP 2
2080 J=0
2090 J=J+1:PRINT MID$(X$,F,2)+CHR$(32);
2100 IF J>23 THEN J=0:PRINT
2110 W$="&h"+MID$(X$,F,2)
2120 PRINT#1,CHR$(VAL(W$));
2130 NEXT F
2140 PRINT:PRINT
2150 RETURN
2160 H$="(c)1993,1995 Patrick GUEULLE"
```

Sur le plan logiciel, le programme **COREL.BAS** a été développé de façon à permettre d'effectuer pratiquement n'importe quelle opération sur n'importe quelle carte, comme en témoigne son menu, reproduit ci-après:

0 QUITTER
1 MISE SOUS TENSION et RESET
2 MISE HORS TENSION
3 MODE GPM256
4 LECTURE
5 ECRITURE
6 PRESENTATION DE CODE PORTEUR
7 EFFACEMENT DE MOT (2 octets)
8 PRESENTATION CODE EMETTEUR ET EFFACEMENT
9 AUTRES COMMANDES ISO

VOTRE CHOIX puis ENTER

Les commandes les plus fréquemment utilisées sont accessibles *en direct*, tandis que l'option **9** permet d'émettre toute commande ISO normalisée, aussi bien **entrante** que **sortante**.

Rappelons qu'on appelle commande **entrante** une commande destinée à émettre un bloc de données vers la carte, et une commande **sortante** une commande destinée à recueillir des données en provenance de la carte.

Muni de la liste des instructions reconnues par les cartes bancaires, reproduite à la figure 2.5, ce logiciel permet donc de partir librement à la découverte de toute carte de ce type.

Il est logique de commencer par s'attaquer à la *zone de lecture*, dont une cartographie simplifiée est fournie à la figure 2.6.

On remarquera que chaque mot de 32 bits commence par une *clef de contrôle* souvent égale à 0011, mais fixée à 0010 lorsqu'elle marque le début d'un bloc dit prestataire.

Le prestataire le plus intéressant est le N°02, qui contient des données sensiblement équivalentes à celles qui sont gravées ou embossées dans le plastique de la carte et enregistrées sur ses pistes magnétiques: numéro de carte, identité de son porteur, dates de validité, etc.

INS	ORDRE
0E	Effacement
10	Présentation de la clé banque (CB)
20	Présentation du code confidentiel (CC) ou de la clé de déblocage de la carte
30	Présentation de la clé d'ouverture (CO)
40	Validation de lecture
50	Ecriture de verrou
70	Validation d'écriture
80 82 84 86 88	Certification avec - 1er jeu secret - 2ème jeu secret - 3ème jeu secret - 4ème jeu secret - 5ème jeu secret
A0	Recherche du premier mot vierge ou sur argument
A8	Recherche du premier mot non vierge
B0	Lecture d'octets
C0	Lecture de résultat
D0	Ecriture d'un mot

Figure 2.5.
Le jeu d'instructions des cartes bancaires M4 B0'.

ADL (08E0)	0010	1110	Prestataire 03		L=48	clef	CCE
08E8	0011	0000	0000	0000	0000		
08F0	0011	320 bits (valeur d'authentification)					
...							
(0948)	0010	1110	Prestataire 02		L=56	111	CCE
	0011	Code enreg. 00	N° carte (19 caractères BCD)				
...							
0968	0011	usage (3 car. BCD)		début validité (4 car. BCD)			
0970	0011	langue (3 car. BCD)		fin validité (4 car. BCD)			
0978	0011	devise (3 car. BCD)		exposant Bin. réf. (début)			
0980	0011	IDENTITE DU PORTEUR (26 caractères ASCII)					
...							
09B8	0011	IDENTITE DU PORTEUR		RUF	Bin. réf. (suite)		

09C0	00	AD1	CCE	RUF	
09C8	00	ADL	CCE	ADT	CCE
09D0	00	ADC	CCE	ADM	CCE
09D8	00	AD2	CCE	ADS	CCE
09E0	TYPE = 3FE5 (CB)				CCE
09E8	00	AD1	CCE	N° fabricant	CCE
09F0	N° de série				CCE
09F8				10011	

Figure 2.6.
Cartographie de la zone de lecture des cartes bancaires.

Le prestataire 03, quant à lui, joue déjà un rôle sécuritaire : il contient en effet une *valeur d'authentification* codée sur 320 bits, encore appelée *identité certifiée*.

Il semblerait que ce soit le résultat d'un calcul cryptographique mettant en jeu le contenu du prestataire 02 et le *jeu secret* dont on sait qu'il n'est en aucun cas lisible directement.

Un moyen comme un autre pour démontrer que la carte est authentique !

LE BON USAGE DU CODE CONFIDENTIEL

Dès qu'on tentera des opérations de lecture à des adresses inférieures à ADL, il faudra présenter à la carte le code confidentiel de son porteur.

On butera alors sur une particularité propre aux cartes CP8, qui exigent que tout code présenté soit ensuite *validé* (on dit aussi *ratifié*) par une instruction spéciale.

Il n'est donc pas suffisant d'utiliser la fonction *présentation de code* de COREL.BAS !

Pour mettre en œuvre l'instruction de validation de code (40h), il faut passer par sa fonction d'émission de commandes ISO quelconques, entrantes ou sortantes.

Mais nous ne sommes pas encore au bout de nos peines ! En effet, un code confidentiel normalement constitué se compose de quatre chiffres, alors que la carte attend un mot de quatre octets (huit chiffres hexa).

Le code 4950 devra par exemple être présenté sous la forme (hexadécimale) 12543FFF.

Chacun aura donc à *transcoder* les codes confidentiels de ses cartes avant de les présenter, en l'occurrence à l'aide du petit logiciel PIN2CB.BAS que nous avons écrit à cet effet.

```
10 REM —— PIN2CB.BAS ——
20 KEY OFF:CLS:INPUT "PIN: ",P$:K$=""
30 FOR F=1 TO LEN(P$)
40 M$="&h"+MID$(P$,F,1)
50 M=VAL(M$)
60 D=0
70 IF M>7 THEN D=1:M=M-8
80 GOSUB 280
90 IF M>3 THEN D=1:M=M-4
100 GOSUB 280
110 IF M>1 THEN D=1:M=M-2
120 GOSUB 280
130 D=M:GOSUB 280
140 NEXT F
150 K$="00"+K$+"11111111111111"
160 PRINT:PRINT"CODE HEXA A PRESENTER A LA CARTE: ";
170 FOR F=1 TO LEN(K$) STEP 4
180 D$=MID$(K$,F,4)
190 A=0
200 FOR G=1 TO 4
210 B$=MID$(D$,5-G,1)
220 IF B$="1" THEN A=A+2^(G-1)
230 NEXT G
```

```
240 A$=HEX$(A)
250 PRINT A$;
260 NEXT F:PRINT:PRINT
270 END
280 IF D=0 THEN K$=K$+"0"
290 IF D=1 THEN K$=K$+"1"
300 D=0:RETURN
310 REM (c)1994 Patrick GUEULLE
```

Le déblocage en lecture de la zone de travail nécessitera donc l'enchaînement des opérations suivantes :

- lancement de COREL.BAS ;
- déclaration de BC comme classe ISO de la carte ;
- mise sous tension de la carte (reset) ;
- présentation du code par la fonction réservée à cet effet, ou bien en tant que commande ISO entrante avec un code opération égal à 20h et une adresse fixée à 00h ;
- validation de ce code par une commande ISO entrante de code opération 40h, avec une adresse 00h et pas de données.

Il ne reste plus alors qu'à procéder, autant de fois qu'on le voudra, à la lecture d'un maximum de 128 octets à l'intérieur de cette zone, c'est-à-dire à partir de l'adresse 0260h.

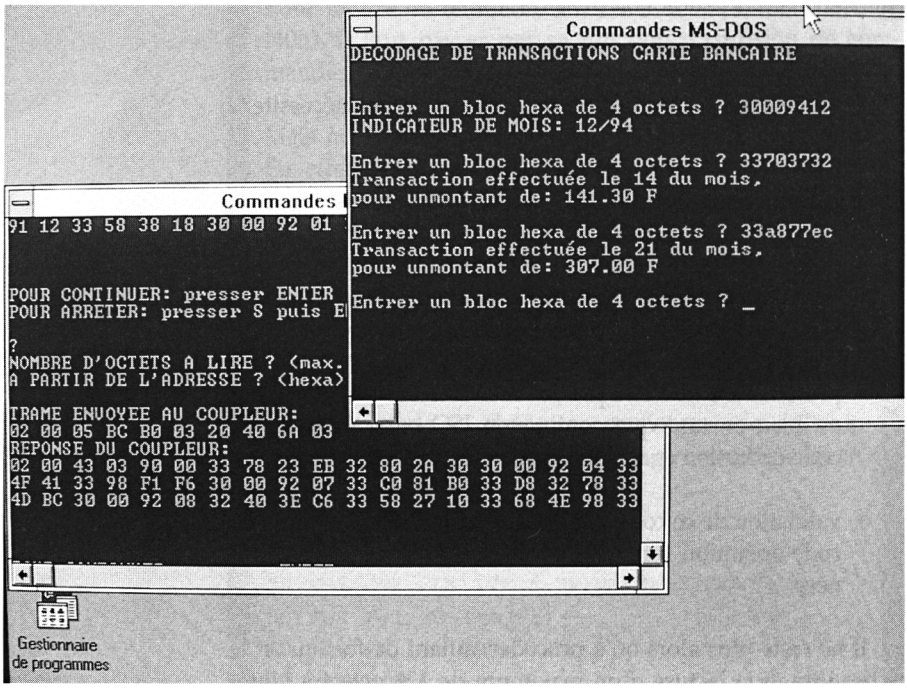
On devrait normalement observer une suite de 0 séparée d'une suite de F par un quartet pouvant être égal à 1, 3, ou 7.

Selon nos observations et nos informations, chaque présentation et validation d'un code confidentiel viendrait mettre un bit à zéro dans cette chaîne de bits à 1 : il y aurait donc ici un véritable compteur surveillant l'usage qui est fait de la carte et pouvant éventuellement, sur les cartes B0, se trouver saturé en cas d'utilisation particulièrement intensive.

Bien entendu, il y a ici une possibilité permettant de vérifier si une carte a été utilisée à l'insu de son porteur, mais attention : le contrôle nécessitant la présentation du code confidentiel, il *consomme* nécessairement un bit...

CONTROLEZ VOS RELEVÉS DE COMPTE

Les blocs de quatre octets qui suivent reproduisent (s'il y a lieu) l'historique des paiements effectués, mois après mois, par l'intermédiaire de la puce (et en aucun cas des pistes magnétiques) de la carte.



Exploration d'une carte bancaire sous WINDOWS 3.11.

Un mot **30 00 91 12** marquerait par exemple le début du mois de décembre 1991 tandis que les mots **33 58 38 18** ou **33 98 4C F4** trouvés juste après retraceraient deux achats effectués, avec présentation du code confidentiel à la puce, dans le courant de ce même mois.

Moyennant le décodage de ces mots de 32 bits, on arrive à retrouver trace, au centime près, de chaque achat et de sa date : de quoi reconstituer un véritable *relevé de compte* !

```
10 REM ---- TRANCB.BAS ----
20 KEY OFF:CLS
30 PRINT"DECODAGE DE TRANSACTIONS CARTE BANCAIRE":PRINT:PRINT
40 PRINT"Entrer un bloc hexa de 4 octets ";
50 INPUT B$:M=0:D=0:C$=""
60 IF B$="" THEN END
```

```
70 FOR F=1 TO LEN(B$)
80 K$=MID$(B$,F,1)
90 IF K$=CHR$(32) THEN 110
100 C$=C$+K$
110 NEXT F
120 B$=C$
130 IF LEN(B$)<>8 THEN BEEP:GOTO 50
140 IF LEFT$(B$,1)<>"3" THEN BEEP:GOTO 50
150 IF LEFT$(B$,4)="3000" THEN 280
160 D$="&h"+MID$(B$,3,1)
170 D=2*VAL(D$)
180 FOR F=1 TO 5
190 H$="&h"+MID$(B$,9-F,1)
200 H=VAL(H$)
210 M=M+(H*16^(F-1))
220 NEXT F
230 IF M>=2^19 THEN D=D+1:M=M-2^19
240 PRINT"Transaction effectuée le";D;"du mois, pour un montant de:";
250 PRINT USING"####.##";M/100;
260 PRINT" F":PRINT
270 GOTO 40
280 D$=RIGHT$(B$,4)
290 M$=RIGHT$(D$,2)
300 Y$=LEFT$(D$,2)
310 PRINT"INDICATEUR DE MOIS: ";M$;"/";Y$:PRINT
320 GOTO 40
330 REM (c)1995 Patrick GUEULLE
```

Le programme **TRANCB.BAS** se charge intégralement de ce travail, bien qu'il subsiste un doute quant à son comportement sur des transactions dont le montant excéderait quelques milliers de francs : à première vue, il y a risque de dépassement.

Notons que les utilisateurs de WINDOWS 3.1x pourront avantageusement exécuter COREL et TRANCB (de préférence compilés en fichiers EXE avec TURBO BASIC), dans deux fenêtres DOS distinctes : cela facilitera les allers-retours en cours d'exploration.

Bien entendu, la tenue dans la carte elle-même de cet historique détaillé de toutes les transactions explique les phénomènes de saturation parfois signalés avant la fin des deux années de validité des cartes bancaires à puce, car le masque B0 utilisant de la mémoire EPROM, aucun *recyclage* des zones utilisées n'est possible.

Réalisées en technologie EEPROM (réinscriptible), les nouvelles cartes B0′ supportent une procédure dans laquelle les transactions les plus anciennes sont effacées pour céder de la place aux plus récentes. Le problème peut donc être considéré comme réglé !

UN LOGICIEL SPÉCIAL CARTE BANCAIRE

Même si COREL.BAS permet de procéder à toutes les opérations imaginables sur les cartes bancaires, son utilisation peut se révéler fastidieuse lorsque les groupes d'octets lus nécessitent un décodage particulier.

C'est notamment le cas des données stockées dans la zone de lecture, aussi avons nous écrit un programme infiniment plus convivial mais strictement réservé aux applications *carte bancaire* ou PASTEL. Il se nomme CABA.BAS, et facilite de surcroît la présentation et la ratification du code confidentiel : il suffit de composer le code tel que le mémorise son porteur (quatre chiffres), et sans aucun transcodage intermédiaire, d'accepter la ratification proposée.

```
10 REM —- CABA.BAS —-
20 KEY OFF:CLS
30 OPEN "COM1:9600,n,8,1" AS #1
40 DATA 2,2,0,2,3,2,0,5,188,176,9,72,60,116,3,2,3,0,3,3
50 X$="":FOR F=1 TO 20
60 READ D:X$=X$+ CHR$(D)
70 NEXT F
80 R$=LEFT$(X$,5)
90 L$=MID$(X$,6,10)
100 S$=RIGHT$(X$,5)
110 PRINT:PRINT"INTRODUIRE UNE CARTE BANCAIRE OU PASTEL"
120 BEEP:PRINT"           puis presser ENTER"
130 INPUT Z$
140 PRINT#1,R$;
150 FOR T=0 TO 2000:NEXT T
160 IF LOC(1)=0 THEN PRINT"COUPLEUR MUET":CLS:GOTO 110
170 C$=INPUT$(LOC(1),#1)
180 CLS:PRINT"RESET (réponse du coupleur):"
190 FOR F=1 TO LEN(C$)
200 D$=HEX$(ASC(MID$(C$,F,1)))+" "
210 IF LEN(D$)<3 THEN D$="0"+D$
220 PRINT D$;
230 NEXT F:PRINT
```

```
240 PRINT#1,L$;
250 N$=""
260 FOR T=0 TO 2000:NEXT T
270 IF LOC(1)=0 THEN PRINT"COUPLEUR MUET":CLS:GOTO 110
280 C$=INPUT$(LOC(1),#1)
290 IF MID$(C$,5,1)<>CHR$(144) THEN 110
300 PRINT:PRINT"LECTURE ZONE LIBRE (réponse du coupleur):":PRINT
310 FOR F=1 TO LEN(C$)
320 D$=HEX$(ASC(MID$(C$,F,1)))+" "
330 IF LEN(D$)<3 THEN D$="0"+D$
340 N$=N$+LEFT$(D$,2)
350 PRINT D$;
360 NEXT F:PRINT:PRINT
370 PRINT"No CARTE:     ";
380 PRINT MID$(N$,24,5);
390 PRINT MID$(N$,30,7);
400 FOR F=38 TO 44
410 Y$=MID$(N$,F,1)
420 Y=ASC(Y$)
430 IF Y<48 OR Y>57 THEN 450
440 PRINT Y$;
450 NEXT F
460 PRINT:PRINT
470 PRINT"VALIDITE DE: ";MID$(N$,51,2);"/";MID$(N$,49,2);
480 PRINT" Ö: ";MID$(N$,59,2);"/";MID$(N$,57,2)
490 PRINT:PRINT"PORTEUR:      ";
500 I$=""
510 FOR F=70 TO 118 STEP 8
520 I$=I$+MID$(N$,F,7)
530 NEXT F
540 I$=I$+MID$(N$,126,3)
550 FOR F=1 TO 51 STEP 2
560 K$=MID$(I$,F,2)
570 K$="&h"+K$
580 PRINT CHR$(VAL(K$));
590 NEXT F
600 PRINT:PRINT:PRINT
610 FOR T=0 TO 2000:NEXT T
620 PRINT"COMPOSER LE CODE CONFIDENTIEL"
630 PRINT"ET VALIDER en pressant ENTER"
640 INPUT P$:K$=""
650 FOR F=1 TO LEN(P$)
660 M$="&h"+MID$(P$,F,1)
670 M=VAL(M$)
680 D=0
690 IF M>7 THEN D=1:M=M-8
700 GOSUB 910
```

```
710 IF M>3 THEN D=1:M=M-4
720 GOSUB 910
730 IF M>1 THEN D=1:M=M-2
740 GOSUB 910
750 D=M:GOSUB 910
760 NEXT F
770 K$="00"+K$+"11111111111111":P$=""
780 PRINT:PRINT"PRESENTATION DU CODE HEXA: ";
790 FOR F=1 TO LEN(K$) STEP 4
800 D$=MID$(K$,F,4)
810 A=0
820 FOR G=1 TO 4
830 B$=MID$(D$,5-G,1)
840 IF B$="1" THEN A=A+2^(G-1)
850 NEXT G
860 A$=HEX$(A)
870 P$=P$+A$
880 NEXT F
890 PRINT P$
900 GOTO 940
910 IF D=0 THEN K$=K$+"0"
920 IF D=1 THEN K$=K$+"1"
930 D=0:RETURN
940 A$="0000"
950 H$=P$
960 L=LEN(H$)/2
970 U$=HEX$(5+L)
980 IF LEN(U$)<2 THEN U$="0"+U$
990 IF (L-INT(L))<>0 THEN BEEP:GOTO 950
1000 B$=HEX$(L)
1010 IF LEN(B$)<2 THEN B$="0"+B$
1020 X$="0201"+U$+"BC"+"20"+A$+B$+H$
1030 GOSUB 1680
1040 GOSUB 1450
1050 IF M$="9000" THEN PRINT"PRESENTATION EFFECTUEE"
1060 IF M$="9010" THEN PRINT"PRECEDENT CODE FAUX: 1er essai":BEEP
1070 IF M$="9020" THEN PRINT"PRECEDENT CODE FAUX: 2ème essai":BEEP
1080 IF M$="9040" THEN PRINT"PRECEDENT CODE FAUX: DERNIER essai":BEEP
1090 IF M$="9080" THEN PRINT"CARTE BLOQUEE!":BEEP
1100 BEEP:PRINT"RATIFICATION DU CODE: O/N puis ENTER":PRINT
1110 INPUT Z$: IF Z$<>"o" AND Z$<>"O" THEN 1420
1120 X$="020105BC40000000"
1130 GOSUB 1680
1140 IF LOC(1)=0 THEN 1140
1150 GOSUB 1450
1160 IF M$="9000" THEN PRINT"CODE BON":GOTO 1220
1170 IF M$="9014" THEN PRINT"CODE FAUX: 1er essai":BEEP
```

```
1180 IF M$="9024" THEN PRINT"CODE FAUX: 2ème essai":BEEP
1190 IF M$="9044" THEN PRINT"CODE FAUX: 3ème essai":BEEP
1200 IF M$="9080" THEN PRINT"CARTE SATUREE!":BEEP
1210 PRINT:GOTO 1370
1220 PRINT:PRINT"LECTURE POSSIBLE EN ZONE PROTEGEE:"
1230 PRINT"NOMBRE D'OCTETS A LIRE ? (max. 64) ";
1240 INPUT Q
1250 IF Q>100 THEN 1240
1260 Q$=HEX$(Q)
1270 IF LEN(Q$)>1 THEN 1290
1280 Q$="0"+Q$:GOTO 1270
1290 PRINT"A PARTIR DE L'ADRESSE ? (hexa) ";
1300 INPUT A$:PRINT
1310 IF LEN(A$)>3 THEN 1330
1320 A$="0"+A$:GOTO 1310
1330 X$="020005BCB0"+A$+Q$
1340 GOSUB 1680
1350 IF LOC(1)=0 THEN 1350
1360 GOSUB 1450
1370 PRINT"POUR CONTINUER: presser ENTER"
1380 PRINT"POUR ARRETER: presser S puis ENTER"
1390 PRINT:INPUT Z$
1400 IF Z$<>"" THEN 1420
1410 GOTO 1230
1420 PRINT#1,S$;
1430 END
1440 Z$="(c)1993,1995 Patrick GUEULLE"
1450 FOR T=0 TO 2000:NEXT T
1460 R$=""
1470 IF LOC(1)=0 THEN PRINT"COUPLEUR MUET":GOTO 1670
1480 C$=INPUT$(LOC(1),#1)
1490 J=0:PRINT"REPONSE DU COUPLEUR:"
1500 FOR F=1 TO LEN(C$)
1510 D$=HEX$(ASC(MID$(C$,F,1)))
1520 IF LEN(D$)<2 THEN D$="0"+D$
1530 R$=R$+D$
1540 PRINT D$+CHR$(32);:J=J+1
1550 IF J>23 THEN J=0:PRINT
1560 NEXT F
1570 PRINT:PRINT
1580 M$=MID$(R$,9,4)
1590 E$=MID$(R$,7,2)
1600 IF M$="9001"THEN PRINT"PROBLEME D'ECRITURE":BEEP
1610 IF M$="6E00"THEN PRINT"CLASSE ISO INCORRECTE":BEEP
1620 IF E$="02"THEN PRINT"CARTE NON TRAITEE":BEEP
1630 IF E$="FD"THEN PRINT"CARTE ARRACHEE":BEEP
1640 IF E$="FF"THEN PRINT"CARTE MUETTE":BEEP
```

```
1650 IF E$="FA"THEN PRINT"CARTE ABSENTE":BEEP
1660 IF E$="FB"THEN PRINT"COURT-CIRCUIT CARTE":BEEP
1670 PRINT:RETURN
1680 L=0
1690 PRINT"TRAME ENVOYEE AU COUPLEUR:"
1700 FOR F=3 TO (LEN(X$)-1) STEP 2
1710 W$="&h"+MID$(X$,F,2)
1720 L=L XOR VAL(W$)
1730 L$=HEX$(L)
1740 IF LEN(L$)<2 THEN L$="0"+L$
1750 NEXT F
1760 X$=X$+L$+"03"
1770 FOR F=1 TO (LEN(X$)-1) STEP 2
1780 J=0
1790 J=J+1:PRINT MID$(X$,F,2)+CHR$(32);
1800 IF J>23 THEN J=0:PRINT
1810 W$="&h"+MID$(X$,F,2)
1820 PRINT#1,CHR$(VAL(W$));
1830 NEXT F
1840 PRINT
1850 RETURN
```

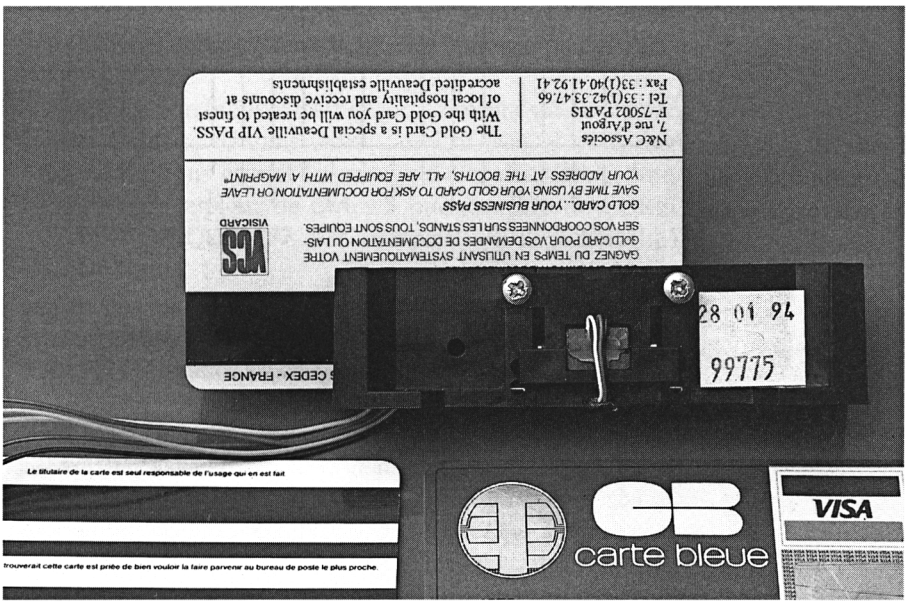
Ce logiciel gère aussi les éventuelles présentations de codes erronés qui, même dans ce contexte expérimental, invalident bel et bien la carte au bout de trois tentatives et laissent en tout état de cause des traces durables dans la **zone d'état**.

COMMENT LIRE LES PISTES MAGNÉTIQUES

Même si cela affaiblit considérablement leur sécurité, certaines cartes à microprocesseur sont munies, en plus, de pistes magnétiques reproduisant au moins en partie le contenu de la puce.

Tel est notamment le cas des cartes bancaires internationales, situation qui se prolongera fatalement tant que la puce ne sera pas adoptée à l'échelon mondial.

Mais les cartes *nationales* sont également concernées, puisque les distributeurs de billets français doivent bien accepter les cartes purement magnétiques de nos visiteurs étrangers.



La figure 2.7 montre comment sont placées les trois pistes internationales dites ISO 1, ISO 2, et ISO 3. Les pistes supplémentaires T2 et T3, que l'on trouvait jusqu'à présent surtout au dos des cartes bancaires VISA, sont en voie de disparition.

Les pistes magnétiques sont également intéressantes à lire.

Chacune de ces pistes peut être encodée selon un principe fort simple, dont la figure 2.8 donne le détail.

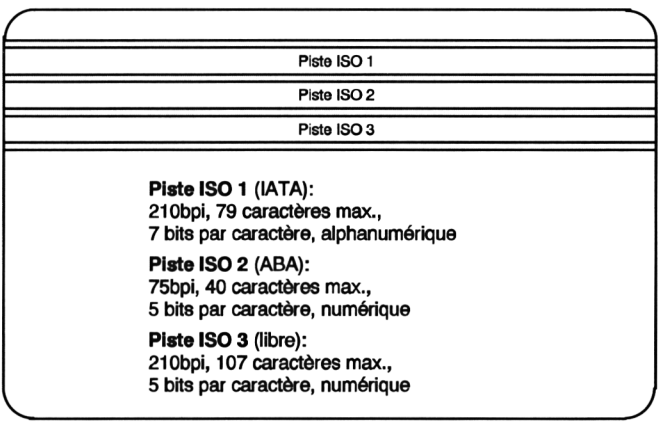


Figure 2.7.
Les caractéristiques des pistes magnétiques.

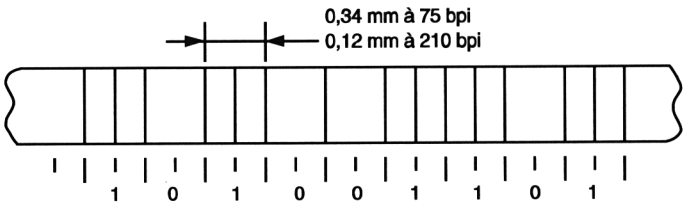


Figure 2.8.
Le codage des bits sur les pistes magnétiques.

Chaque trait vertical correspond à une étroite zone magnétisée, parfaitement visible à la loupe si on pulvérise sur la piste un produit *révélateur magnétique* comme le TRANSCODE de JELT-CM.

La densité d'enregistrement peut être de 75 ou 210 bpi (bits par pouce), et on peut inscrire soit des caractères alphanumériques à 7 bits (ASCII) soit des données numériques codées sur 5 bits.

Compte tenu de la longueur de la carte (environ 85 mm), on arrive ainsi à une limite de 79 caractères pour la piste ISO 1 (dite aussi IATA car utilisée couramment par les compagnies aériennes), 40 caractères pour la piste ISO 2 ou ABA (largement employée par les banques) et 107 caractères pour la piste ISO 3 dont l'affectation est libre. C'est évidemment bien peu par rapport à une carte à microprocesseur moderne, mais pas ridicule pour autant en comparaison des 256 bits (et non pas caractères !) d'une télécarte à puce.

Bien qu'il soit à la rigueur possible de décoder visuellement (surtout à 75 bpi) le contenu d'une piste préalablement révélée, il est infiniment plus pratique de se servir d'un lecteur spécialisé.

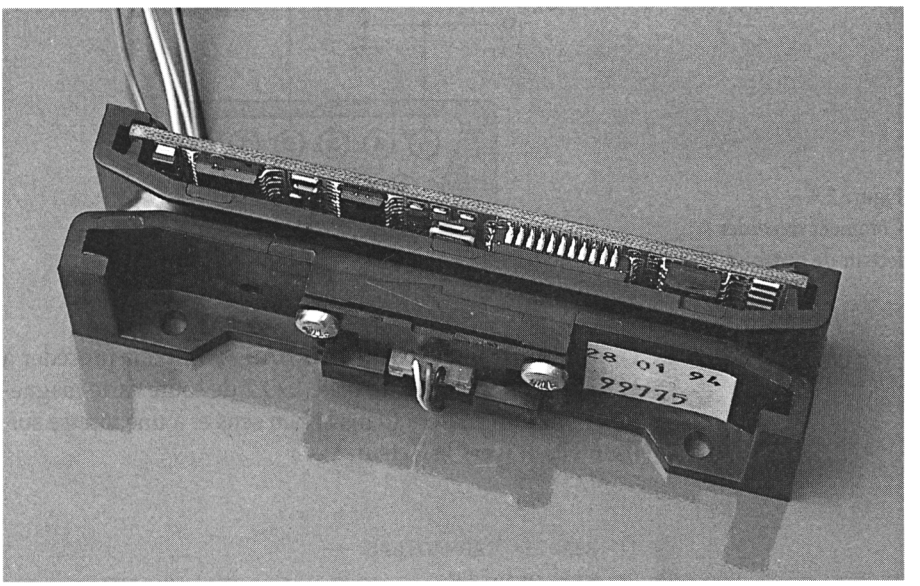
Un peu comme un connecteur pour cartes à puce, un lecteur de cartes magnétiques est un organe de précision qu'on ne peut guère se permettre de *bricoler* soi-même. Il existe heureusement des modèles à défilement manuel de la carte, relativement économiques et faciles à interfacer à un PC.

Il faut cependant distinguer les lecteurs simples, dont l'électronique incorporée se contente de mettre en forme les signaux de la tête de lecture, des lecteurs **RS232** dans lesquels un microcontrôleur assure le décodage de ces informations brutes.

C'est dans cette seconde catégorie qu'il convient de choisir les lecteurs destinés à être connectés à un compatible PC.

Parmi les nombreux constructeurs présents sur le marché, nous avons sélectionné le français THOMSON-LCC, dont les fabrications offrent l'avantage d'être disponibles, tout comme le coupleur de cartes à puce, auprès de COREL ELECTRONIQUE (37 Av. Raspail, 94253 GENTILLY CEDEX).

Ces lecteurs existent en version monopiste (ISO 1, 2, ou 3) ou bipiste (ISO 1 et 2 ou ISO 2 et 3).



Le lecteur magnétique RS 232 de THOMSON-LCC.

La transmission des données lues se fait, après vérification de l'intégrité du message et transcodage en ASCII, par une liaison RS232 à 2400 bauds.

L'alimentation est prévue en 5 V, avec une consommation inférieure à 40 mA.

Pour la plupart des manipulations intéressantes à mener sur un compatible PC avec des cartes magnétiques courantes, le lecteur le plus indiqué est le modèle LR 200 L2, prévu pour la piste ISO 2.

La figure 2.9 montre comment utiliser son petit connecteur MOLEX, équipé de cinq fils repérables d'après leur couleur.

La prise DB9 viendra se brancher sur le port série (COM1 :) du PC, au besoin par l'intermédiaire d'un adaptateur 25 broches ou d'un cordon si l'ordinateur est équipé d'une prise RS232 **DB 25**.

L'alimentation 5 V pourra provenir d'une source externe, ou être prélevée sur un connecteur qui en dispose (par exemple une prise pour manettes de jeu).

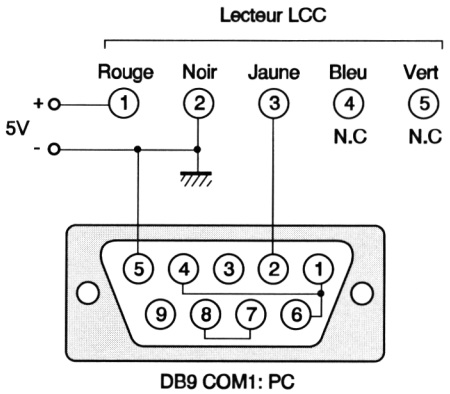


Figure 2.9.
Comment raccorder le lecteur de cartes au PC.

Le petit programme **CARMAG.BAS** suffit pour procéder à la lecture complète de la piste ISO 2 de toute carte magnétique que l'on passera (dans le bon sens et à une vitesse suffisante) à travers le lecteur.

```
10 REM —— CARMAG.BAS ——
20 KEY OFF:CLS
30 OPEN"COM1:2400,n,8,1" AS #1
40 IF LOC(1)<>0 THEN GOSUB 60
50 GOTO 40
60 FOR T=1 TO 5000:NEXT T
70 IF LOC(1)=0 THEN RETURN
80 C$=INPUT$(LOC(1),#1)
90 PRINT C$:PRINT
100 RETURN
110 REM (c)1994 Patrick GUEULLE
```

La suite de chiffres qui s'affiche représente le contenu utile de la piste, débarrassé des caractères d'encadrement qui le précèdent et le suivent en général, et se termine par un *retour chariot*.

On peut avoir toute confiance dans ce qui s'affiche, car en cas d'erreur ou de piste vierge le lecteur ne transmet rien.

Si l'on tente de lire une carte bancaire de cette façon, on obtient une ligne de chiffres contenant, entre autres, le numéro de la carte et sa date limite de validité, données identiques à celles embossées dans le plastique et enregistrées dans la puce.

Une carte à microprocesseur asynchrone n'est finalement rien d'autre qu'un petit système micro-informatique, avec mémoire et unité centrale, capable de communiquer avec son lecteur par une liaison série bidirectionnelle.

A condition de *faire l'impasse* sur une bonne partie des subtilités de la norme ISO 7816-3, celles-là mêmes que ne peut en aucune façon se permettre d'ignorer un développeur d'applications *sensibles*, on peut considérer que le fonctionnement d'une carte asynchrone se résume la plupart du temps à des échanges d'octets à 9 600 bauds.

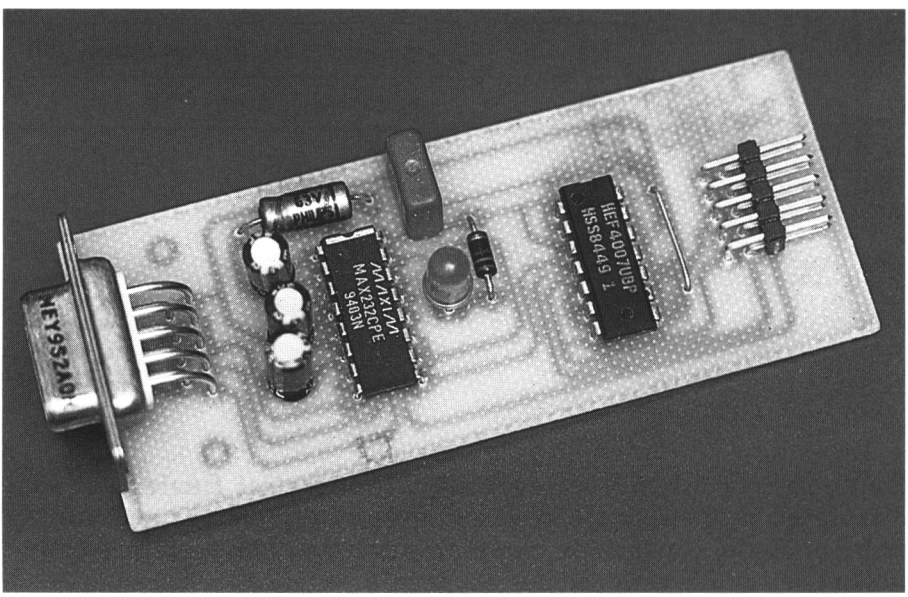
Il est évidemment fort tentant de chercher à interfacer cartes à microprocesseur, lecteurs, et ports RS232 de PC pour faire de l'analyse de protocole ou même, il suffit d'oser, de la simulation.

UN ADAPTATEUR RS232 POUR CARTE ASYNCHRONE

On pourra s'étonner du fait que le petit montage dont la figure 3.1 dévoile le schéma ne supporte que quatre des six à huit contacts dont dispose normalement un connecteur de carte à puce.

En réalité, les cartes asynchrones n'utilisent dans leur immense majorité que six contacts, dont un affecté à une tension de programmation V_{pp} que nous n'avons pas besoin de gérer dans notre contexte.

L'adaptateur RS232 achevé.



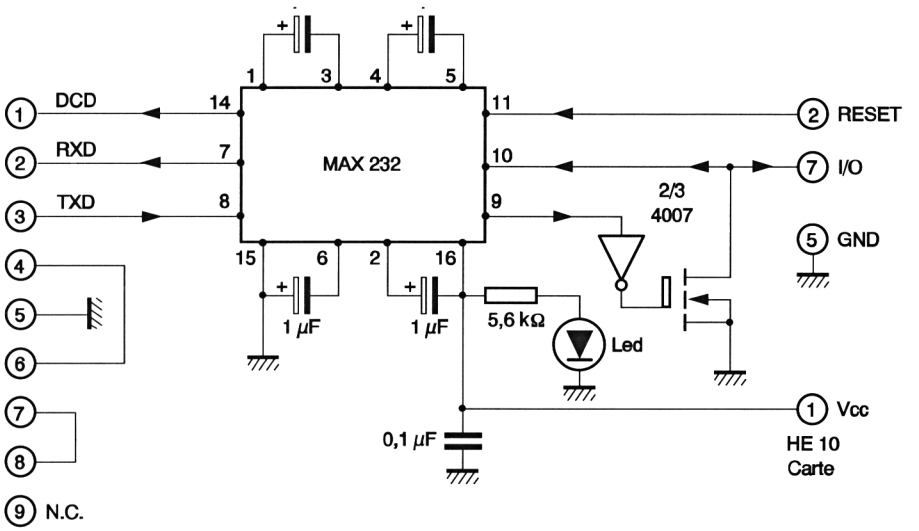


Figure 3.1.
Le schéma de l'adaptateur RS232.

Plus surprenante est peut-être l'absence de la connexion d'horloge, mais elle s'explique fort bien : le PC cadence son port série à 9 600 bauds à partir de sa propre horloge interne, ce qui lui permet d'ignorer superbement le signal **CLK** provenant du lecteur.

Ce n'est évidemment guère orthodoxe vis-à-vis de la norme ISO (qui prévoit des possibilités de changement de fréquence d'horloge et de rythme de modulation), mais l'expérience montre que c'est acceptable dans la grande majorité des cas susceptibles de nous intéresser.

La pièce maîtresse du montage est un circuit intégré MAX232 qui, alimenté par le + 5 volts venant du lecteur de cartes, reconstitue les niveaux + 12 V et – 12 V compatibles avec le port RS232.

Le MAX232 est secondé par un CD4007, chargé de multiplexer sur l'unique ligne d'entrée-sortie (bidirectionnelle) de la carte, les données circulant séparément sur les lignes TXD et RXD de la RS232 : cela laisse supposer qu'après avoir servi passivement d'analyseur de protocole, le même montage pourra, le moment venu, simuler au moins partiellement une carte à microprocesseur !

Cette configuration matérielle reconstitue en effet aussi fidèlement que possible la structure à *drain ouvert* des cartes à puce normalisées.

Converti lui aussi en niveaux RS232, le signal de remise à zéro de la carte (RESET) est relayé vers le PC par la ligne DCD (*Carrier Detect*) du port série : cela permettra au logiciel de détecter les demandes de remise à zéro formulées par le lecteur de cartes, voire même d'y répondre s'il n'y a pas de vraie carte pour le faire.

Assemblé sur un petit circuit imprimé dont la figure 3.2 fournit le tracé, le montage est muni de deux connecteurs à câbler selon la figure 3.3 :

- une embase DB9 femelle destinée à venir s'enficher, directement ou par l'intermédiaire d'un adaptateur DB9-DB25, dans le port COM1 : du PC (attention, n'utiliser en aucun cas une embase mâle et un cordon double femelle : employer plutôt, si nécessaire, un cordon mâle-femelle dit *rallonge de moniteur*) ;

- un connecteur HE10 femelle compatible avec l'ensemble des éléments de la *boîte à outils pour cartes à puce* décrite dans notre ouvrage «CARTES A PUCE, Initiation et Applications» paru dans cette même collection : connecteurs de cartes, *fausses cartes* en circuit imprimé, et bien entendu lecteur de cartes.

Il est naturellement vital que la correspondance de brochage soit assurée entre tous ces éléments. Ce sera automatiquement le cas si on sertit des fiches HE10 mâles à dix contacts dans le même sens sur un morceau de câble en nappe à dix conducteurs, et si on utilise en tant que réceptacles des tronçons de barrettes sécables à picots carrés coudés.

En effet, l'ergot de détrompage des fiches HE10 viendra alors buter sur le circuit imprimé en cas de tentative d'insertion dans le mauvais sens.

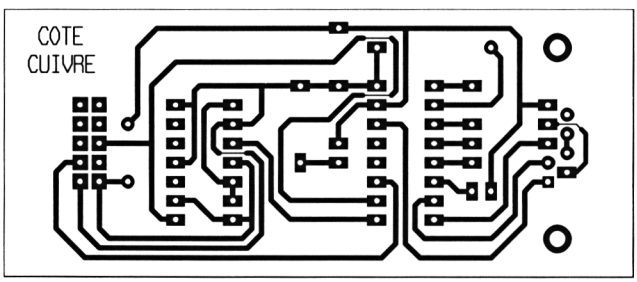


Figure 3.2.
Le circuit imprimé de l'adaptateur RS232.

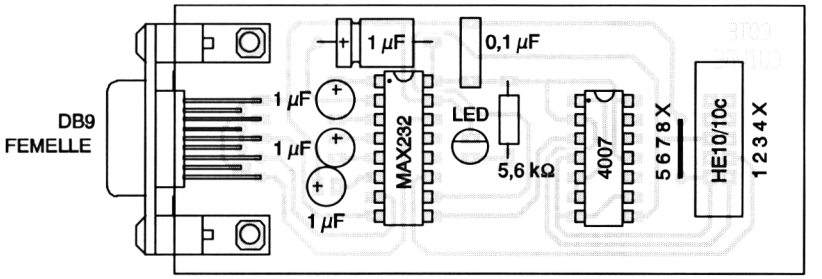


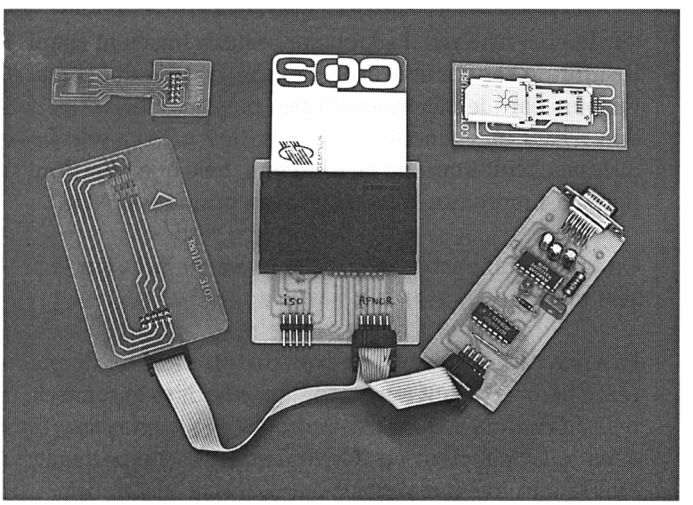
Figure 3.3.
Plan de câblage de l'adaptateur RS232.

Liste des composants

1 circuit intégré MAX 232,
1 circuit intégré CD4007,
1 LED rouge 5 mm,
4 condensateurs chimiques 1 µF 25 V (1 axial, 3 radiaux),
1 condensateur 0,1 µF 63 V,
1 résistance 5,6 kΩ 1/4 W,
1 embase DB9 femelle coudée pour circuit imprimé,
1 barrette sécable à double rangée de picots carrés coudés,
1 fiche HE10 à 10 contacts supplémentaire pour le cordon.

UN PETIT ANALYSEUR DE PROTOCOLE

Que ne donnerait-on pas, parfois, pour savoir ce que peuvent bien se dire les cartes à puce et les lecteurs dans lesquels on les introduit ?



L'analyseur de protocole et ses accessoires.

Aussi surprenant que cela puisse paraître, ce petit montage et quelques lignes de BASIC suffisent pour *espionner* une carte bancaire, de téléphone (PASTEL ou GSM) ou bien, et c'est plus amusant encore, de télévision à péage.

Toujours plus fort, on peut même songer à simuler, dans des limites raisonnables, le dialogue ainsi intercepté !

Bien entendu, nous ne prenons la liberté d'écrire ces lignes que parce que nous avons pu vérifier qu'une telle démarche ne remet nullement en cause la sécurité des applications carte correctement protégées.

L'un des buts de ce livre est précisément de permettre à nos lecteurs de se faire une idée personnelle sur la question : après tout, on croit ce qu'on voit, n'est-ce pas ?

Une bonne plate-forme d'expérimentation de cartes asynchrones comprendra, outre le présent montage, un lecteur de cartes à puce (figure 2.2) connecté à son propre compatible PC.

Il sera en effet prudent de ne songer que dans un deuxième temps à s'attaquer à des équipements disponibles à domicile tels que MINITEL à lecteur de cartes, téléphone portable GSM, ou décodeur de télévision.

Il faut donc en principe deux micro-ordinateurs pour manipuler confortablement, mais l'un d'entre eux (de préférence celui affecté à l'analyseur) pourra sans problème être un vieil XT à 8086 ou encore un portable.

Sur le plan matériel, l'adaptation est extrêmement simple : il suffit de sertir une troisième fiche HE10 à dix contacts, toujours dans le même sens, à peu près au milieu d'un cordon reliant un connecteur de carte (figure 2.4) à une *fausse carte* en circuit imprimé de 8/10 mm, telle que celle dont la figure 3.4 reproduit le tracé (puce en position AFNOR).

On branchera, c'est évident, cette troisième fiche sur le présent montage.

La vraie carte à puce (carte bancaire ou PASTEL de préférence périmée) étant maintenant insérée dans le connecteur relié au cordon, il suffit d'introduire à sa place la *fausse carte* dans le lecteur dont on désire étudier le comportement : tout se retrouve fort simplement connecté en parallèle.

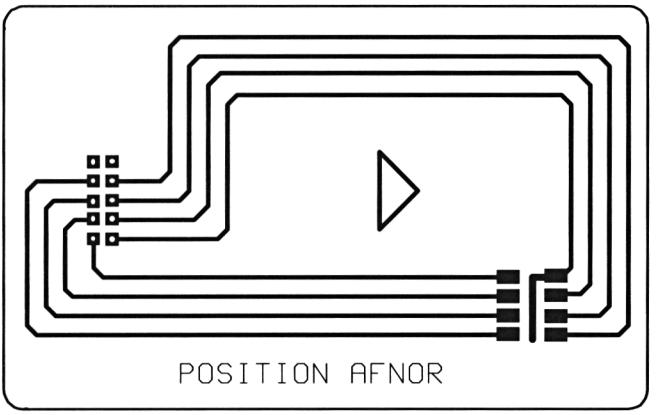


Figure 3.4.
Le tracé du cuivre de la «fausse carte» AFNOR.

A condition d'avoir été lancé avant toute émission de demande de reset de la carte par le lecteur (donc en premier), le très simple logiciel **ESPION.BAS** va alors afficher (en hexadécimal) tous les octets circulant entre le lecteur et la carte.

```
10 REM --- ESPION.BAS ---
20 KEY OFF
30 OPEN "COM1:9600,n,8,2" AS #1
40 E = INP(&H3FE) AND 128
50 IF E<>128 THEN 40
60 GOSUB 80
70 END
80 IF LOC(1)<>0 THEN GOSUB 100
90 GOTO 80
100 IF LOC(1)=0 THEN RETURN
110 ON ERROR GOTO 310
120 C$=INPUT$(LOC(1),#1): PRINT
130 FOR K=1 TO LEN(C$)
140 N=ASC(MID$(C$,K,1))
150 M=255
160 IF N>127 THEN N=N-128:M=M-1
170 IF N>63 THEN N=N-64:M=M-2
180 IF N>31 THEN N=N-32:M=M-4
190 IF N>15 THEN N=N-16:M=M-8
200 IF N>7 THEN N=N-8:M=M-16
210 IF N>3 THEN N=N-4:M=M-32
220 IF N>1 THEN N=N-2:M=M-64
230 IF N>0 THEN M=M-128
240 D$=HEX$(M)+" "
250 IF LEN(D$)<3 THEN D$="0"+D$
```

```
260 PRINT D$;
270 NEXT K
280 IF LOC(1)<>0 THEN C$=INPUT$(LOC(1),#1):GOTO 130
290 PRINT
300 RETURN
310 RESUME
320 REM (c)1995 Patrick GUEULLE
```

On remarquera que le port série du PC est ouvert en mode **N**, c'est-à-dire sans bit de parité, mais avec deux bits de stop.

Les cartes à puce ISO 7816 utilisent pourtant un protocole à parité paire (*E comme even*).

La raison de cette entorse à la norme est que le GWBASIC n'admet pas de bit de parité lorsqu'il est programmé avec des mots de huit bits. Or, nous tenions à utiliser ce langage particulièrement populaire chez nos lecteurs.

Pour que notre version GWBASIC puisse tout de même fonctionner correctement, il nous a fallu y inclure une routine de traitement des erreurs de parité, qui se contente d'ailleurs de les contourner sans autre forme de procès (ON ERROR, RESUME).

On pourra toutefois avantageusement compiler ce programme sous TURBO-BASIC, langage moins regardant sur la programmation du port série, après avoir modifié la ligne 30 avec les réglages 9600,E,8,2.

On remarquera aussi toute une routine de transcodage d'octets qui permute, à la réception, les bits de poids fort avec ceux de poids faible, tout en les complémentant. C'est ce qu'on appelle une conversion en *convention ISO inverse*, les cartes à puce communiquant en général bit de poids fort en tête et **1** logique représenté par un niveau bas.

Déclenchant un retour à la ligne dès que le flot de données s'interrompt, même brièvement, ce programme permet dans une certaine mesure de distinguer la provenance de tel ou tel groupe d'octets (venant du lecteur ou de la carte).

A ce stade de nos investigations, il est commode de piloter le lecteur par un logiciel mettant à contribution un maximum de fonctions des cartes bancaires, le plus indiqué

étant naturellement notre programme **CABA.BAS** (voir chapitre 2).

Le petit logiciel **CB2PIN.BAS** permettrait même, le cas échéant, de *remonter* au code confidentiel en partant des huit octets interceptés lors de leur transmission : à n'utiliser donc qu'à bon escient...

```
10 REM —- CB2PIN.BAS —-
20 KEY OFF:CLS
30 INPUT "CODE HEXA présenté à la carte: ",P$:K$=""
40 FOR F=1 TO LEN(P$)
50 M$="&h"+MID$(P$,F,1)
60 M=VAL(M$)
70 D=0
80 IF M>7 THEN D=1:M=M-8
90 GOSUB 300
100 IF M>3 THEN D=1:M=M-4
110 GOSUB 300
120 IF M>1 THEN D=1:M=M-2
130 GOSUB 300
140 D=M:GOSUB 300
150 NEXT F
160 K$=MID$(K$,3,LEN(K$)-16)
170 PRINT:PRINT"PIN = ";
180 FOR F=1 TO LEN(K$) STEP 4
190 D$=MID$(K$,F,4)
200 A=0
210 FOR G=1 TO 4
220 B$=MID$(D$,5-G,1)
230 IF B$="1" THEN A=A+2^(G-1)
240 NEXT G
250 A$=HEX$(A)
260 PRINT A$;
270 NEXT F
280 PRINT:PRINT:PRINT
290 END
300 IF D=0 THEN K$=K$+"0"
310 IF D=1 THEN K$=K$+"1"
320 D=0:RETURN
330 REM (c)1994 Patrick GUEULLE
```

UN PETIT SIMULATEUR DE CARTE

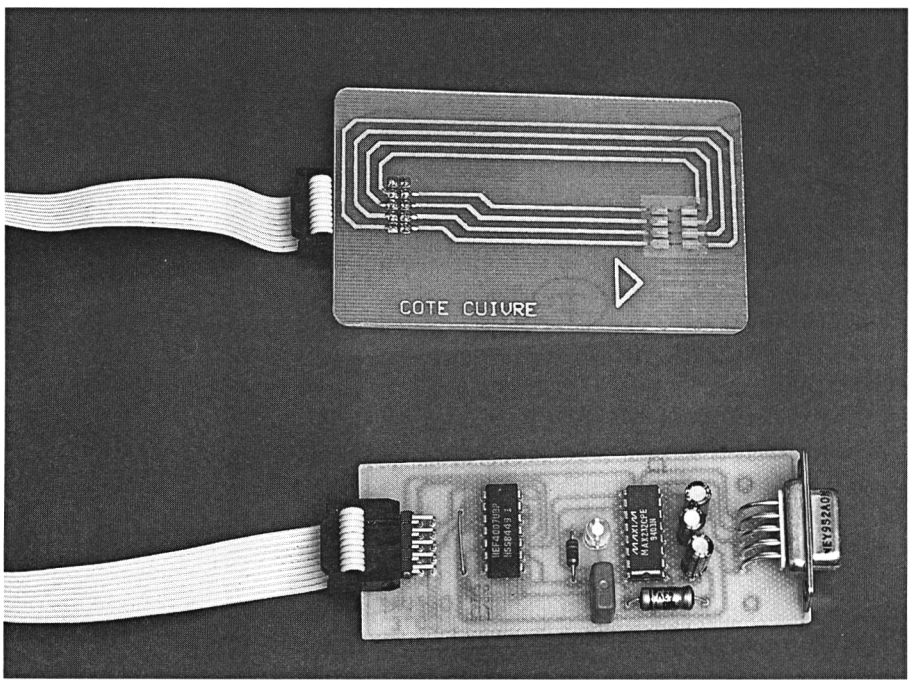
Parmi toute la panoplie d'outils dont disposent les développeurs d'applications *cartes à puce*, l'émulateur de cartes figure en bonne place.

Qu'il soit autonome ou connecté à un PC, cet instrument équipé d'une *fausse carte* en circuit imprimé est en principe capable de se faire passer pour n'importe quelle carte à puce *asynchrone*, autrement dit à microprocesseur.

Nous allons nous pencher ici sur un projet d'émulateur très simplifié que nous baptiserons plutôt *simulateur*.

Piloté lui aussi par un compatible PC, il partagera avec l'analyseur de protocole qui vient d'être décrit le même adaptateur RS232 dont nous avons pu apprécier la déroutante simplicité.

Sur le plan matériel, un simulateur de cartes à puce peut en effet être fort compliqué ou étonnamment simple, mais dans les deux cas du logiciel spécifique doit naturellement être écrit pour chaque carte qu'il s'agit de simuler. Les spécialistes appellent cela *programmer un masque*.



Le simulateur de carte prêt à être raccordé au PC.

Faut-il donc en déduire qu'il suffit de réaliser un petit montage électronique et d'écrire un peu de code pour se fabriquer une *vraie-fausse carte* bancaire ou de télévision à péage ? Nous n'irons évidemment pas jusque là !

En effet, toute carte à microprocesseur destinée à des applications *sensibles* doit en principe intégrer des fonctions *cryptographiques* basées sur des clefs secrètes qui ne peuvent en aucun cas être lues depuis l'extérieur de la carte (voir chapitre 1).

Sauf faute très lourde de l'émetteur d'une telle carte (dont il porterait évidemment l'entière responsabilité), on considère donc communément comme impossible de percer le mystère de la partie sécurisée des échanges de données entre la carte et son lecteur.

Émulateur de carte simplifié à l'extrême, le présent montage va néanmoins vous permettre de reproduire, grâce à votre PC, les parties pas trop secrètes des dialogues carte-lecteur que vous aurez préalablement eu la bonne idée d'intercepter : une façon comme une autre d'évaluer jusqu'où on peut se permettre d'*aller trop loin*...

Électriquement, la mise en œuvre du simulateur est on ne peut plus simple : réunir, grâce au câble ad-hoc, une *fausse carte* en circuit imprimé de 8/10 au module RS232 que vous avez en principe déjà réalisé.

Sans intercaler, cette fois, un quelconque connecteur de carte, branchez le tout sur le port série **COM1** : d'un compatible PC. C'est tout !

On pourra avantageusement commencer les manipulations avec le court logiciel **CARS232.BAS**, dont le rôle se limite à simuler une *réponse* au *reset* plausible, puis à afficher en hexadécimal ce que le lecteur voudra bien transmettre ensuite à la carte. Le lecteur se croira bel et bien en présence d'une carte asynchrone, et le premier *barrage* sera franchi.

```
10 REM —- CARS232.BAS —-
20 KEY OFF
30 A$=CHR$(&H3)+CHR$(&HC9)+CHR$(255)+CHR$(255)
40 B$="AAAAAAAAAAAA"
50 OPEN "COM1:9600,n,8,2" AS #1
60 E = INP(&H3FE) AND 128
```

```
70 IF E<>128 THEN 60
80 PRINT#1,A$+B$;
90 GOSUB 110
100 END
110 IF LOC(1)<>0 THEN GOSUB 130
120 GOTO 110
130 IF LOC(1)=0 THEN RETURN
140 ON ERROR GOTO 340
150 C$=INPUT$(LOC(1),#1): PRINT
160 FOR K=1 TO LEN(C$)
170 N=ASC(MID$(C$,K,1))
180 M=255
190 IF N>127 THEN N=N-128:M=M-1
200 IF N>63 THEN N=N-64:M=M-2
210 IF N>31 THEN N=N-32:M=M-4
220 IF N>15 THEN N=N-16:M=M-8
230 IF N>7 THEN N=N-8:M=M-16
240 IF N>3 THEN N=N-4:M=M-32
250 IF N>1 THEN N=N-2:M=M-64
260 IF N>0 THEN M=M-128
270 D$=HEX$(M)+" "
280 IF LEN(D$)<3 THEN D$="0"+D$
290 PRINT D$;
300 NEXT K
310 IF LOC(1)<>0 THEN C$=INPUT$(LOC(1),#1):GOTO 160
320 PRINT
330 RETURN
340 RESUME
350 REM (c)1995 Patrick GUEULLE
```

Nos lecteurs pourront alors pousser plus loin leurs investigations, en cherchant à simuler telle ou telle carte dont ils connaissent plus ou moins bien le fonctionnement.

Quelques commentaires s'imposent quant à ce programme, dont plusieurs éléments serviront de base à tous ceux qu'on pourra être amené à écrire par la suite, en BASIC ou dans un autre langage plus performant.

Ce sont essentiellement les mêmes que ceux formulés à propos de notre logiciel **ESPION.BAS** : ouverture du port série du PC en mode **N** (c'est-à-dire sans bit de parité) mais avec deux bits de stop.

Là encore, la compilation sous TURBO-BASIC est intéressante, après avoir modifié les réglages en 9600,E,8,2.

L'astuce utilisée pour que la version GWBASIC puisse tout de même fonctionner en réception (ON ERROR, RESUME) est inapplicable en sens inverse car les lecteurs de cartes conformes à la norme ISO 7816 sont plus pointilleux : lorsqu'ils détectent une erreur de parité, ils envoient à la carte un signal qui consiste en un niveau bas prolongé, puis attendent une répétition du caractère considéré comme mal reçu. Dans notre cas, il y aurait blocage !

Il faudra donc se limiter, dans la composition du message de réponse au reset, à l'utilisation de caractères comportant un nombre pair de bits à 1. C'est le cas, notamment, de la lettre **A**.

La routine des lignes 60 et 70 sert pour sa part à attendre l'arrivée d'une demande de reset. On pourra la réutiliser ailleurs dans les programmes, afin que la *fausse carte* réagisse correctement à des demandes de reset reçues en cours de session et pas seulement au début. Cela peut arriver !

On remarquera aussi, comme dans le logiciel **ESPION.BAS**, une routine de conversion en *convention ISO inverse* utilisée seulement en réception.

A l'émission, nous avons directement opéré la conversion lors de la construction du message de réponse au reset : rien de plus simple grâce à la table de la figure 3.5.

La réaction normale d'un lecteur de cartes recevant une réponse au reset qui lui donne satisfaction est en général

	Convention directe	Convention inverse	
0	0 0 0 0	1 1 1 1	F
1	0 0 0 1	0 1 1 1	7
2	0 0 1 0	1 0 1 1	B
3	0 0 1 1	0 0 1 1	3
4	0 1 0 0	1 1 0 1	D
5	0 1 0 1	0 1 0 1	5
6	0 1 1 0	1 0 0 1	9
7	0 1 1 1	0 0 0 1	1
8	1 0 0 0	1 1 1 0	E
9	1 0 0 1	0 1 1 0	6
A	1 0 1 0	1 0 1 0	A
B	1 0 1 1	0 0 1 0	2
C	1 1 0 0	1 1 0 0	C
D	1 1 0 1	0 1 0 0	4
E	1 1 1 0	1 0 0 0	8
F	1 1 1 1	0 0 0 0	0

Figure 3.5.
Une table de conversion en «convention inverse».

d'émettre une commande à destination de la carte, à charge pour celle-ci d'y répondre.

Nous arrivons bien évidemment là aux limites de ce premier petit programme : il va certes afficher l'ordre émanant du coupleur, mais ne pourra pas y répondre.

Grâce à notre *analyseur de protocole*, nous savons pourtant ce que doit répondre la carte dans un certain nombre de situations courantes, et c'est précisément cela qu'il est intéressant de reproduire.

A peine plus compliqué que les précédents, le programme **SIMU.BAS** permet de reproduire, en GWBASIC, quelques fonctions de base d'une carte genre **COS** : reset, lecture de deux octets DDDDh à l'adresse AAAAh, et présentation du code confidentiel CCCCh à l'adresse BBBBh. Cela avec une classe ISO égale à 00h.

```
10 REM —— SIMU.BAS ——
20 KEY OFF
30 A$=CHR$(&H3)+CHR$(&HC9)+CHR$(255)+CHR$(255)
40 B$="AAAAAAAAAAAA"
50 J$=CHR$(&HF6)+CHR$(&HFF)
60 OPEN "COM1:9600,N,8,2" AS #1
70 E=INP(&H3FE) AND 128
80 IF E<>128 THEN 70
90 PRINT#1,A$+B$;
100 GOSUB 160
110 IF M$="00 B0 AA AA 02 " THEN GOSUB 390
120 IF M$="00 20 BB BB 02 " THEN GOSUB 400
130 IF M$="21 CC CC " THEN GOSUB 410
140 GOTO 100
150 END
160 IF LOC(1)=0 THEN 160
170 ON ERROR GOTO 370
180 M$=""
190 C$=INPUT$(LOC(1),#1): PRINT
200 FOR K=1 TO LEN(C$)
210 N=ASC(MID$(C$,K,1))
220 M=255
230 IF N>127 THEN N=N-128:M=M-1
240 IF N>63 THEN N=N-64:M=M-2
250 IF N>31 THEN N=N-32:M=M-4
260 IF N>15 THEN N=N-16:M=M-8
270 IF N>7 THEN N=N-8:M=M-16
```

```
280 IF N>3 THEN N=N-4:M=M-32
290 IF N>1 THEN N=N-2:M=M-64
300 IF N>0 THEN M=M-128
310 D$=HEX$(M)+" "
320 IF LEN(D$)<3 THEN D$="0"+D$
330 M$=M$+D$
340 NEXT K
350 PRINT M$
360 RETURN
370 RESUME
380 REM (c)1995 Patrick GUEULLE
390 PRINT#1,CHR$(&H72)+"DD"+J$;:RETURN
400 PRINT#1,CHR$(&H7B);:RETURN
410 PRINT#1,J$;:RETURN
```

Notons que les premiers octets de la réponse au reset simulée informent le lecteur que la carte fonctionne sous une tension unique de 5 V (pas de V_{pp} externe) et en *convention inverse*.

Typiquement, on fera tourner ce petit programme sur un premier PC équipé du simulateur, tandis qu'un second PC sera muni du lecteur construit selon la figure 2.2.

Bien entendu, SIMU.BAS sera lancé avant le logiciel de pilotage du lecteur, ou tout au moins avant l'envoi à celui-ci d'un quelconque ordre de mise sous tension de la carte.

Il importera que l'un des deux PC ne soit pas exagérément rapide par rapport à l'autre : si l'émulateur fonctionne par exemple sur un vieil XT à 4,77 MHz, alors il est souhaitable que le PC muni du lecteur ne *tourne* pas à plus de 8 MHz (XT à 8 MHz, ou AT386SX25 sans *turbo*).

De nombreux *hors-temps* sont en effet prévus dans les échanges de données entre carte et lecteur, dont tout dépassement risquerait fort de bloquer la situation (compte-rendu *carte muette* ou *coupleur muet*).

En tout état de cause, un logiciel aussi simplifié ne peut prétendre que simuler bien grossièrement, et surtout très partiellement, le fonctionnement fort complexe d'une carte à microprocesseur.

Il ne peut que se borner à montrer à nos lecteurs la voie à suivre pour mettre au point leurs propres *masques* écrits

dans le langage de leur choix, à commencer par l'assembleur si on souhaite se fabriquer une *fausse carte* autonome à partir d'un microcontrôleur courant.

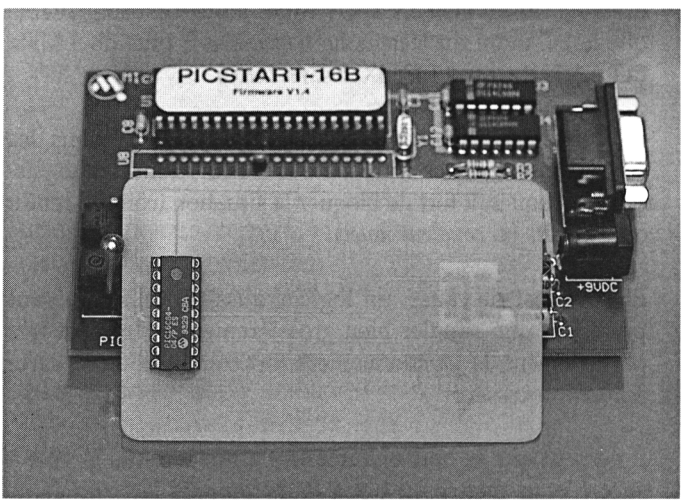
UNE CARTE À PUCE EXPÉRIMENTALE À PIC16CXX

Une carte à puce asynchrone (autrement dit à microprocesseur) n'est finalement rien d'autre qu'un microcontrôleur spécial monté dans ce boîtier bien particulier qu'est une carte plastique munie de contacts conformes à la norme ISO 7816.

Aussi surprenant que cela puisse paraître de prime abord, il suffit de câbler un microcontrôleur DIP ou CMS sur un circuit imprimé de 8/10 mm d'épaisseur pour fabriquer une *fausse carte à puce* permettant déjà de fort intéressantes manipulations, naturellement en tout bien tout honneur.

Les PIC16C71 et PIC16C84 de MICROCHIP se prêtent admirablement à ce genre d'exercice, car nous allons constater que l'émulation des fonctionnalités de base des cartes asynchrones ne nécessite finalement que fort peu de ressources, aussi bien matérielles que logicielles.

Le terme *carte asynchrone* couramment employé dit bien ce qu'il veut dire, à savoir qu'une telle carte se contente de recevoir et d'émettre tour à tour (en *half-duplex*) des octets transmis en mode série asynchrone.



Le côté composants de la carte à PIC16C84.

Le côté cuivre de la carte à PIC16C84.

Le cadencement de ces transmissions est confié à un signal d'horloge fourni à la carte par le lecteur dans lequel on l'introduit, de telle façon que pour une fréquence d'horloge de 3,58 MHz environ le rythme de modulation soit de 9600 bits par seconde.

Une seule et même ligne (le contact «I/O» de la carte) servant à la fois au transfert d'octets en provenance de la carte ou destinés à celle-ci, une configuration «OU câblé» est nécessaire pour éviter tout risque de conflit matériel.

La figure 3.6 montre ainsi comment lecteur et carte disposent tous deux d'étages de sortie à collecteur ouvert (ou à drain ouvert), une résistance de rappel à V_{cc} devant être prévue au moins du côté du lecteur.

Lorsque la ligne de transmission est au niveau haut (son état de repos), le lecteur ou la carte peut imposer un niveau bas (bit de start) pour indiquer qu'il commence à transmettre. Il est alors de bon ton que l'autre protagoniste s'abstienne de transmettre en même temps, bien que ce genre de conflit ne puisse en aucun cas avoir d'effets destructifs ni d'un côté ni de l'autre.

Les paramètres de communication sont fixés par la norme ISO 7816 : un bit de start, huit bits de données, un bit de

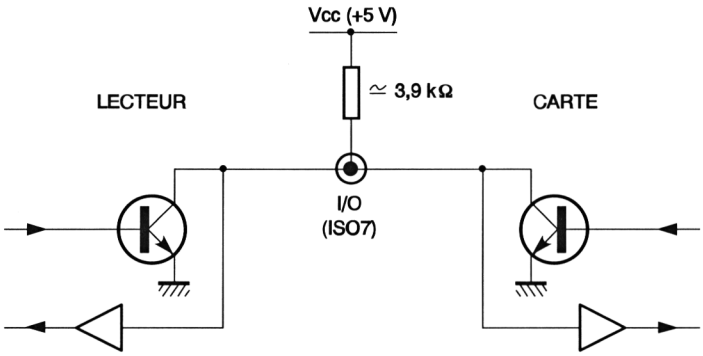


Figure 3.6.
L'agencement des entrées-sorties en «OU câblé».

parité paire, et un bit de stop servant également d'indicateur d'erreur de transmission (un niveau bas recouvrant le bit de stop constitue une demande de réémission d'un octet mal reçu).

Nous savons toutefois que deux variantes sont admises : la *convention ISO inverse* (bit de poids fort en tête et **1** logique représenté par un niveau bas), ou plus rarement la *convention ISO directe* (poids faible en tête et **1** logique au niveau haut).

C'est la carte qui impose au lecteur l'une de ces deux conventions, au début de l'émission de sa *réponse au reset*.

Ce groupe d'octets, émis spontanément par la carte lors de sa mise sous tension et/ou à réception d'un signal sur son contact de *reset*, contient de multiples informations sur les caractéristiques de la carte et permet ainsi au lecteur de s'y adapter.

Bien entendu, la signification de chaque octet de la réponse au reset est prévue dans la norme ISO 7816, à l'exception toutefois des derniers caractères (dits *historiques*) dont l'usage est laissé à la discrétion du développeur de l'application.

Cinq contacts suffisent donc pour relier la carte à son lecteur :

- la masse (GND), ou «ISO 5» ;
- l'alimentation +5 V (Vcc) ou «ISO 1» ;

- l'horloge (CLK) ou «ISO 3» ;
- la ligne de données (I/O) ou «ISO 7» ;
- l'entrée de reset (RST) ou «ISO 2».

Sur certaines cartes de technologie un peu ancienne (à mémoire EPROM), un sixième contact peut être nécessaire pour la tension de programmation (Vpp) ou «ISO 6», tandis que deux contacts (ISO 4 et ISO 8) restent en réserve pour de futurs usages (RFU).

Si le numérotage de ces contacts est imposé sans équivoque par la norme ISO 7816, en revanche deux positionnements sont possibles (ISO et AFNOR).

La figure 3.7 résume donc ce qu'il est vital de connaître pour assurer la correspondance entre les contacts de la carte et le connecteur du lecteur.

Sur le plan logiciel, rien n'empêche théoriquement de développer un *masque* de carte asynchrone à partir du jeu d'instructions de n'importe quel microcontrôleur.

Pour notre part, nous avons choisi de bénéficier de l'architecture RISC rapide des PIC16CXX.

Bien sûr, les 1024 mots d'EPROM, 36 octets de RAM, et 64 octets d'EEPROM d'un PIC16C84 paraîtront quelque peu étriqués devant les 3 Ko de ROM, 128 octets de RAM, et 1 Ko d'EEPROM du 68HC055SC24 qui équipe, par exemple, la dernière génération de Cartes Bancaires (masque M4 B0').

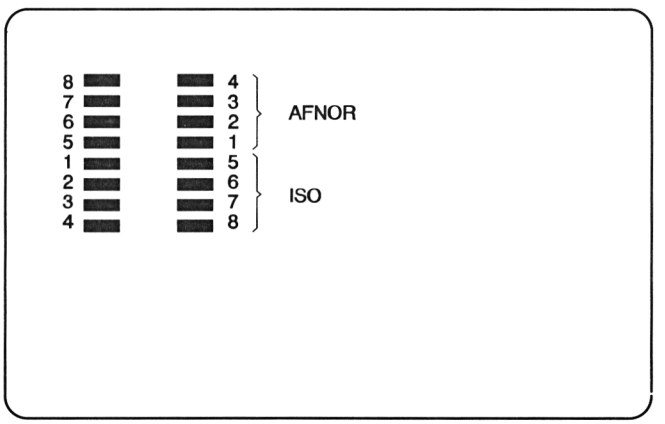


Figure 3.7. L'affectation des contacts selon la norme ISO7816.

Il convient toutefois de remarquer que le code écrit pour les PIC est sensiblement plus compact que celui destiné aux processeurs conventionnels (dits «CISC»), et que rien n'interdit d'associer une EEPROM série externe à n'importe quel PIC, 16C71 ou même 16C5X, voire de faire travailler deux PIC en parallèle dans une même carte.

Mais de toute façon, bien des applications pas trop *sensibles* se contentent d'une fraction seulement des possibilités des cartes à microcalculateur conventionnelles, tandis qu'il n'entre nullement dans nos intentions d'expliquer ici comment fabriquer une fausse carte bancaire ou de télévision à péage !

Cela étant précisé, le mini-système d'exploitation que nous avons développé pour les PIC16C71 et PIC16C84 (il tient dans une centaine de mots d'EPROM) suppose la réalisation du schéma de la figure 3.8.

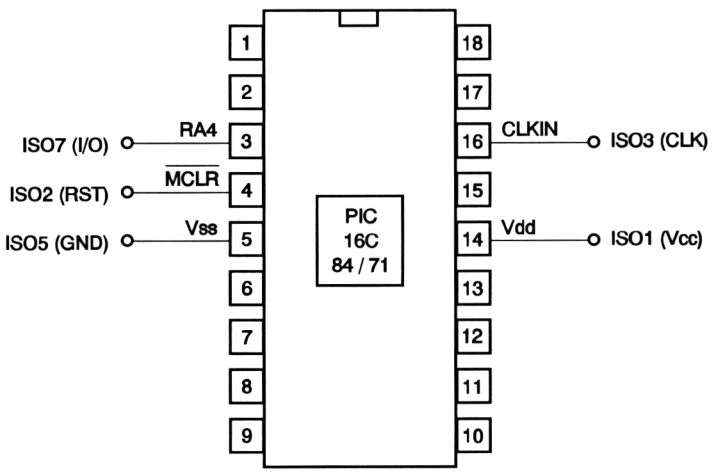


Figure 3.8.
Le très simple schéma de la carte à PIC16CXX.

Pourrait-on imaginer plus simple ?

Les connexions de masse et d'alimentation rejoignent bien entendu les contacts ISO correspondants, sans condensateur de découplage (il n'y en a pas dans les *vraies* cartes à puce !), et c'est l'entrée CLKIN qui reçoit directement le signal d'horloge fourni par le lecteur : pas besoin de quartz ni même de réseau RC !

Le signal de reset attaque fort logiquement la broche/MCLR (*master clear*) du PIC, tandis que nous avons retenu la broche RA4 pour l'entrée-sortie série.

Cela en raison de ses caractéristiques électriques bien particulières qui, rappelées à la figure 3.9, se prêtent tout particulièrement à la réalisation d'un robuste OU câblé avec l'entrée-sortie du lecteur.

Reste maintenant à construire pour de bon la *fausse carte* destinée à nos expérimentations.

Deux tracés de circuit imprimé ont été étudiés : celui de la figure 3.10 est porteur d'une *puce* de contact en position ISO (européenne et donc la seule qui subsistera à terme) et celui de la figure 3.12 d'une puce AFNOR (condamnée à disparaître progressivement car franco-française).

Attention, le sens de montage du PIC est inverse d'une version à l'autre : on positionnera donc attentivement le support *tulipe* que nous ne saurions trop recommander d'intercaler entre la carte et le PIC.

Bien entendu, la gravure de ces *fausses cartes* doit se faire sur de l'époxy présensibilisé de 8/10 (c'est un produit standard du CIRCUIT IMPRIME FRANÇAIS), parfaitement compatible avec les connecteurs prévus pour des cartes épaisses de 0,76 mm.

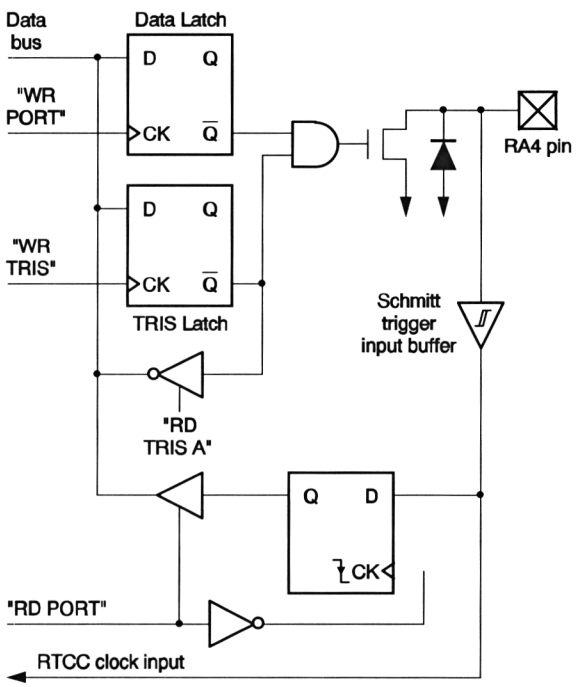


Figure 3.9.
Schéma équivalent de la broche RA4 des PIC16C84/71.

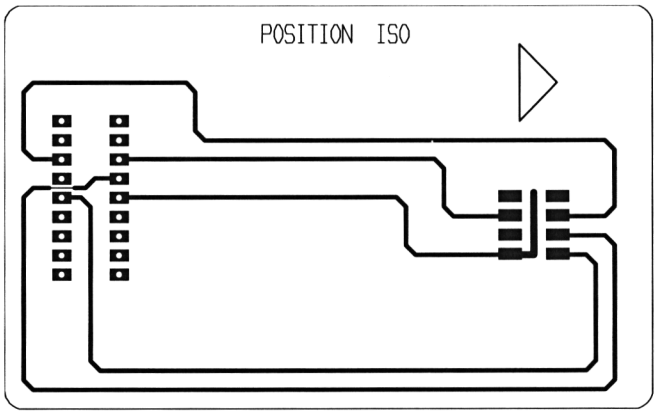


Figure 3.10.
Le tracé du cuivre
de la «fausse carte» ISO.

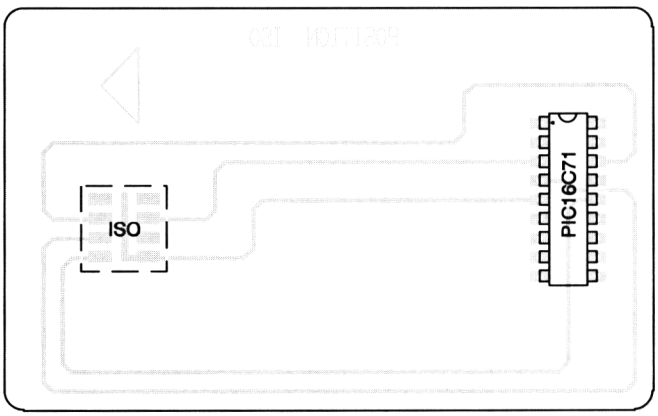


Figure 3.11.
Plan de câblage
de la «fausse carte» ISO.

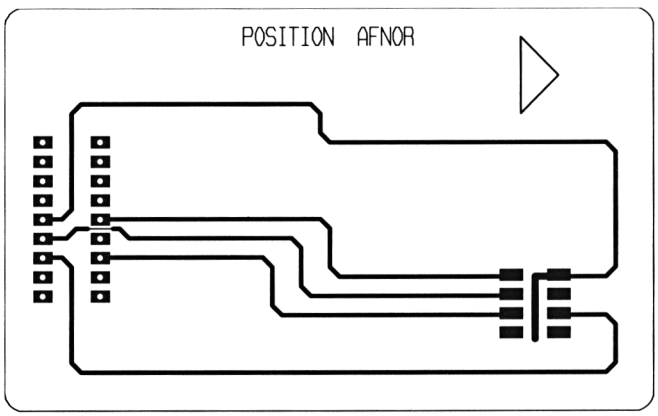


Figure 3.12.
Le tracé du cuivre
de la «fausse carte»
AFNOR.

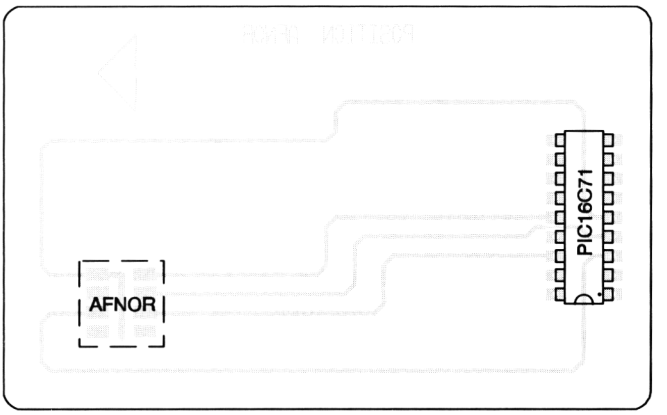


Figure 3.13.
Plan de câblage de la "fausse carte" AFNOR.

En principe, les cartes ne pénètrent pas dans les lecteurs assez loin pour que le PIC et son support puissent arriver en butée, mais en cas de doute on peut toujours rallonger un peu les cartes.

De toute façon, ne comptez pas sur nous pour vous faire réaliser une carte pouvant être *avalée* par un distributeur de billets...

Ceux de nos lecteurs ayant déjà réalisé la *boîte à outils pour cartes à puce* décrite dans notre livre «Cartes à puce Initiation et Applications» auront tout intérêt à graver plutôt (sur de l'époxy d'épaisseur normale) le tout petit circuit imprimé de la figure 3.14.

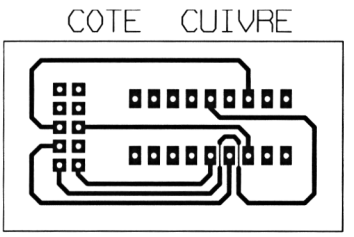


Figure 3.14.
Le tracé du cuivre et le plan de câblage de l'adaptateur pour PIC16CXX.

Équipé selon la figure 3.15 de l'habituel tronçon de barrette sécable à double rangée de picots carrés coudés, il pourra être relié, par le non moins habituel cordon à deux connecteurs HE10 à dix contacts, à toutes les *fausses cartes*

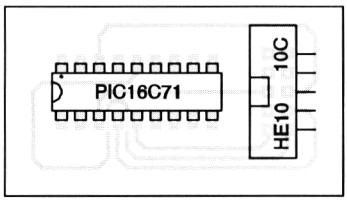


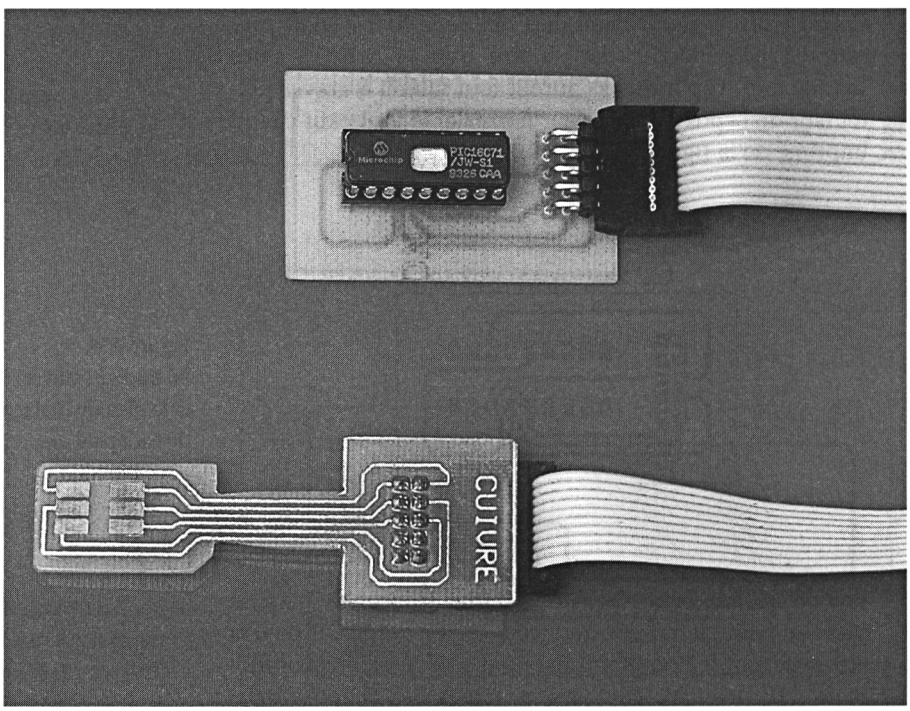
Figure 3.15.
Le tracé du cuivre et le plan de câblage de l'adaptateur pour PIC16CXX.

universelles déjà développées : ISO et AFNOR, bien sûr, mais également SIM compatible avec certains téléphones portables «GSM».

Mais passons maintenant à la partie logicielle de cette étude :

Ce que nous pourrions appeler un peu pompeusement le *masque* de notre *fausse carte* à puce a bien entendu été développé en langage assembleur, en l'occurrence à l'aide des outils présents dans le kit «PICSTART 16B» de MICROCHIP.

Le listing est en fait le résultat de l'assemblage (les adresses de branchement sont calculées) d'un fichier-source nommé PICPUCE.ASM (fourni sur la disquette).



L'adaptateur de PIC16C71 branché sur une fausse carte SIM.

```
16c5x/xx Cross-Assembler V4.12.01 Intermediate  Fri Mar 24 11:13:44 1995  Page 1

Line  PC    Opcode

0001  ; ---------- PICPUCE.ASM ----------
0002  ; émulation carte à puce asynchrone avec un PIC16C71 ou PIC16C84
0003  ; copyright (c)1995 Patrick GUEULLE
0004  ;
0005  ; fusibles de configuration: WDT et PuT inactifs, oscillateur en mode XT
0006  ;
0007        0000  org 0
0008  0000  2804         goto init
0009        0000  org 4
0010  0004  1683  init   bsf 3,5       ;initialisation du port A
0011  0005  1408         bsf 88,0      ;mode entrées-sorties logiques
0012  0006  1488         bsf 88,1      ;(CAN déconnecté)
0013  0007  1283         bcf 3,5
0014  0008  1606         bsf 6,4
0015  0009  2066         call tx       ;broche 3 en mode sortie
0016  000A  3032         movlw .50
0017  000B  0090         movwf 10
0018  000C  20A2  tempo  call delay1   ;retard avant réponse au reset
0019  000D  0B90         decfsz 10,1
0020  000E  280C         goto tempo
0021
0022  000F  303F         movlw 3F      ;émission de la réponse au reset
0023  0010  208F         call even     ;(simulation carte bancaire BO')
0024  0011  3065         movlw 65
0025  0012  208F         call even     ;émission d'un octet de parité paire
0026  0013  3025         movlw 25
0027  0014  2095         call odd      ;émission d'un octet de parité impaire
0028  0015  3008         movlw 08
0029  0016  2095         call odd
0030  0017  3031         movlw 31
0031  0018  2095         call odd
0032  0019  3004         movlw 04
0033  001A  2095         call odd
0034  001B  306C         movlw 6C
0035  001C  208F         call even
0036  001D  3090         movlw 90
0037  001E  208F         call even
0038  001F  3000         movlw 00
0039  0020  208F         call even
0040
0041  ;routine d'attente du code confidentiel ABCDh à une adresse quelconque
0042
0043  0021  206A         call rx       ;broche 3 en mode entrée
```

```
0044  0022  207E  clas    call recv     ;attente de la réception d'un octet
0045  0023  30BC          movlw 0BC     ;l'octet reçu est-il BCh ? (classe ISO BULL CP8)
0046  0024  0691          xorwf 11,1
0047  0025  1D03          btfss 3,2
0048  0026  28A7          goto error    ;si non mutisme de la carte
0049  0027  207E  pin     call recv

16c5x/xx Cross-Assembler V4.12.01 Intermediate  Fri Mar 24 11:13:44 1995  Page 2

Line  PC    Opcode

0050  0028  3020          movlw 20      ;l'octet reçu est-il 20h ? (présentation code)
0051  0029  0691          xorwf 11,1
0052  002A  1D03          btfss 3,2
0053  002B  28A7          goto error    ;si non mutisme de la carte
0054  002C  207E          call recv     ;ignorer le premier octet d'adresse
0055  002D  207E          call recv     ;ignorer le second octet d'adresse
0056  002E  207E          call recv     ;attente de réception d'un octet
0057  002F  3002          movlw 02      ;longueur du code = 2 octets ?
0058  0030  0691          xorwf 11,1
0059  0031  1D03          btfss 3,2
0060  0032  28A7          goto error    ;si non mutisme de la carte
0061  0033  2066          call tx
0062  0034  20A2          call delay1
0063  0035  20A2          call delay1
0064  0036  3020          movlw 20      ;émission de l'octet de procédure "sans Vpp"
0065  0037  2095          call odd
0066  0038  206A          call rx
0067  0039  207E  cla     call recv
0068  003A  30AB          movlw 0AB     ;premier octet du PIN présenté = ABh ?
0069  003B  0691          xorwf 11,1
0070  003C  1D03          btfss 3,2
0071  003D  28A7          goto error    ;si non mutisme de la carte
0072  003E  207E          call recv
0073  003F  30CD          movlw 0CD     ;second octet du PIN présenté = CDh ?
0074  0040  0691          xorwf 11,1
0075  0041  1D03          btfss 3,2
0076  0042  28A7          goto error    ;si non mutisme de la carte
0077  0043  2066          call tx
0078  0044  20A2          call delay1
0079  0045  20A2          call delay1
0080  0046  3090          movlw 90      ;émission du compte-rendu "code bon"
0081  0047  208F          call even
0082  0048  3000          movlw 00
0083  0049  208F          call even
0084  004A  206A          call rx       ;broche 3 en mode entrée
0085
```

```
0086  ;routine de réponse à un ordre de lecture de 2 octets (adresse quelconque)
0087
0088  004B  207E  class   call recv
0089  004C  30BC          movlw 0BC
0090  004D  0691          xorwf 11,1
0091  004E  1D03          btfss 3,2
0092  004F  28A7          goto error
0093  0050  207E  read    call recv
0094  0051  30B0          movlw 0B0      ;l'octet reçu est-il B0h ? (lecture)
0095  0052  0691          xorwf 11,1
0096  0053  1D03          btfss 3,2
0097  0054  28A7          goto error
0098  0055  207E  adr1    call recv      ;ignorer le premier octet d'adresse
0099  0056  207E  adr2    call recv      ;ignorer le second octet d'adresse
0100  0057  207E  len     call recv      ;ignorer l'octet de longueur
```

```
16c5x/xx Cross-Assembler V4.12.01 Intermediate  Fri Mar 24 11:13:44 1995  Page 3

Line  PC    Opcode
```

```
0101  0058  20A2  ack     call delay1
0102  0059  20A2          call delay1
0103  005A  2066          call tx        ;broche 3 en mode sortie
0104  005B  30B0          movlw 0B0      ;réponse de la carte
0105  005C  2095          call odd
0106  005D  30CD  data1   movlw 0CD      ;premier octet de données
0107  005E  2095          call odd       ;(parité impaire)
0108  005F  30EF  data2   movlw 0EF      ;second octet de données
0109  0060  2095          call odd
0110  0061  3090  me1     movlw 90       ;compte-rendu "exécution correcte"
0111  0062  208F          call even      ;(parité paire)
0112  0063  3000  me2     movlw 00
0113  0064  208F          call even
0114  0065  284B  loop    goto class
0115
0116  0066  1683  tx      bsf 3,5        ;routine de mise en mode sortie
0117  0067  1205          bcf 85,4
0118  0068  1283          bcf 3,5
0119  0069  0008          return
0120  006A  1683  rx      bsf 3,5        ;routine de mise en mode entrée
0121  006B  1605          bsf 85,4
0122  006C  1283          bcf 3,5
0123  006D  0008          return
0124  006E  008D  send    movwf 0D       ;routine UART émission
0125  006F  098D          comf 0D,1
0126  0070  3008          movlw 8
0127  0071  008E          movwf 0E
```

```
0128  0072  1205          bcf 5,4
0129  0073  20A2          call delay1
0130  0074  1003  next    bcf 3,0
0131  0075  0D8D          rlf 0D,1
0132  0076  1803          btfsc 3,0
0133  0077  1605          bsf 5,4
0134  0078  1C03          btfss 3,0
0135  0079  1205          bcf 5,4
0136  007A  20A0          call delay2
0137  007B  0B8E          decfsz 0E,1
0138  007C  2874          goto next
0139  007D  0008          return
0140  007E  0191  recv    clrf 11       ;routine UART réception
0141  007F  1A05          btfsc 5,4
0142  0080  289D          goto delay3
0143  0081  209B          call delay4
0144  0082  3008          movlw 8
0145  0083  0090          movwf 10
0146  0084  1003  rnext    bcf 3,0
0147  0085  0D91          rlf 11,1
0148  0086  1A05          btfsc 5,4
0149  0087  1411          bsf 11,0

16c5x/xx Cross-Assembler V4.12.01 Intermediate  Fri Mar 24 11:13:44 1995  Page 4

Line  PC    Opcode

0150  0088  20A2          call delay1
0151  0089  0B90          decfsz 10,1
0152  008A  2884          goto rnext
0153  008B  209B  parity  call delay4   ;ignorer le bit de parité reçu
0154  008C  0991          comf 11,1     ;bit à 1 = niveau bas (convention inverse)
0155  008D  0008          return
0156  008E  20A2          call delay1
0157  008F  206E  even    call send     ;émission d'un octet en parité paire
0158  0090  1605          bsf 5,4
0159  0091  20A2          call delay1
0160  0092  1605          bsf 5,4
0161  0093  20A2          call delay1
0162  0094  0008          return
0163  0095  206E  odd     call send     ;émission d'un octet en parité impaire
0164  0096  1205          bcf 5,4
0165  0097  20A2          call delay1
0166  0098  1605          bsf 5,4
0167  0099  20A2          call delay1
0168  009A  0008          return
0169  009B  3022  delay4  movlw .34     ;temporisation 1,25 bit
```

```
0170  009C  28A3          goto time
0171  009D  300E  delay3   movlw .14      ;temporisation 1/2 bit
0172  009E  20A3          call time
0173  009F  287E          goto recv
0174  00A0  301B  delay2   movlw .27      ;temporisation 1 bit (104 ÊS à 9600 bauds)
0175  00A1  28A3          goto time
0176  00A2  301C  delay1   movlw .28      ;durée d'un bit de start/stop
0177  00A3  008F  time     movwf 0F       ;boucle de temporisation
0178  00A4  0B8F  redo     decfsz 0F,1
0179  00A5  28A4           goto redo
0180  00A6  3400           retlw 0
0181  00A7  28A7  error    goto error     ;boucle sans fin (mutisme et blocage de la carte)
0182  0000          end                   ;(il faudra faire un RESET pour repartir)
```

L'opération d'assemblage avec MPALC produit également un fichier INTELHEX nommé PICPUCE.OBJ qui, reproduit à la figure 3.17 et disponible sur la disquette, est directement compatible avec le programmateur PICSTART de MICROCHIP.

Figure 3.16. L'assemblage du logiciel PICPUCE.ASM.

```
:020000000428D2
:1000080083160814881483120616662032309006E
:10001800A220900B0C283F308F2065308F20253090
:1000280095200830952031309520043095206C308B
:100038008F2090308F2000308F206A207E20BC30A7
:10004800910603 1DA7287E2020309106031DA728AE
:100058007E207E207E2002309106031DA728662080
:10006800A220A220203095206A207E20AB30910665
:10007800031DA7287E20CD309106031DA7286620E2
:10008800A220A22090308F2000308F206A207E206E
:10009800BC309106031DA7287E20B0309106031DB1
:1000A800A7287E207E207E20A220A2206620B030B5
:1000B8009520CD309520EF30952090308F2000305E
:1000C8008F204B2883160512831208008316051605
:1000D800831208008D008D0908308E000512A220B9
:1000E80003108D0D03180516031C0512A0208E0B96
:1000F8007428080091010 51A9D289B20083090005B
:10010800031091 0D051A1114A220900B84289B202E
:1001180091090800A2206E200516A2200516A2202B
:1001280008006E200512A2200516A2200800223021
:10013800A3280E30A3207E281B30A3281C308F0054
:080148008F0BA4280034A72846
:00000001FF
```

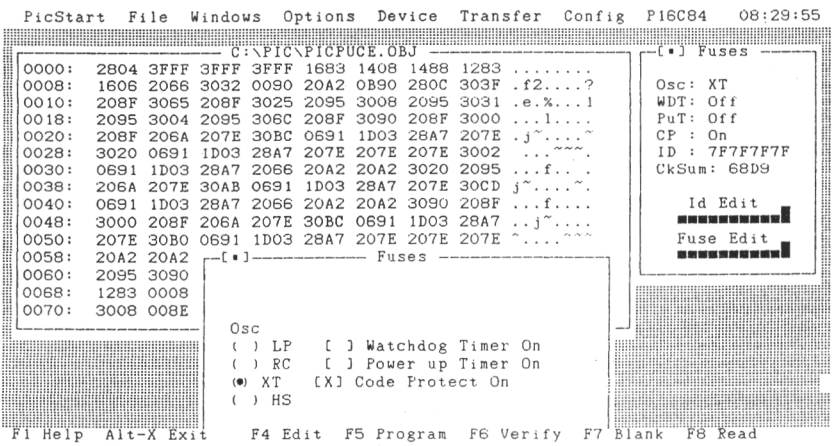
Figure 3.17. Le fichier INTEL-HEX de programmation du PIC.

Il peut indifféremment être *brûlé* dans un PIC16C71 (version effaçable aux UV) ou dans un PIC16C84 (à EEPROM), car nos routines de démonstration ne font pas encore appel à la mémoire EEPROM de ce dernier.

Par contre, il est impératif de neutraliser le *chien de garde* (WDT) et le temporisateur de mise sous tension (PuT), et de mettre l'oscillateur en mode «XT».

Cela en programmant les *fusibles* appropriés dans le menu du logiciel de programmation MPSTART, conformément à la recopie d'écran de la figure 3.18.

Figure 3.18.
Préparation de la programmation du PIC sous MPSTART.



Bien entendu, nos lecteurs désireux de pousser plus loin leurs expérimentations (sous leur seule responsabilité !) en écrivant leurs propres routines applicatives, auront tout intérêt à se pencher sur la lecture et l'écriture dans cette mémoire non volatile, qui conservera son contenu lorsque la carte sera retirée de son lecteur.

Tel qu'il est publié, ce logiciel comprend toutes les routines de base, soigneusement réglées souvent à un cycle d'horloge près, permettant d'émettre une réponse au reset fixe, de recevoir et émettre des octets, et de placer la carte dans un mode de *mutisme* dont seule une mise hors tension ou un reset volontaire pourront la faire sortir (c'est une fonction de sécurité courante sur les vraies cartes à microprocesseur).

Bien entendu, ce *masque* comprend aussi toute une série de sous-programmes chargés d'opérations plus élémentaires, à

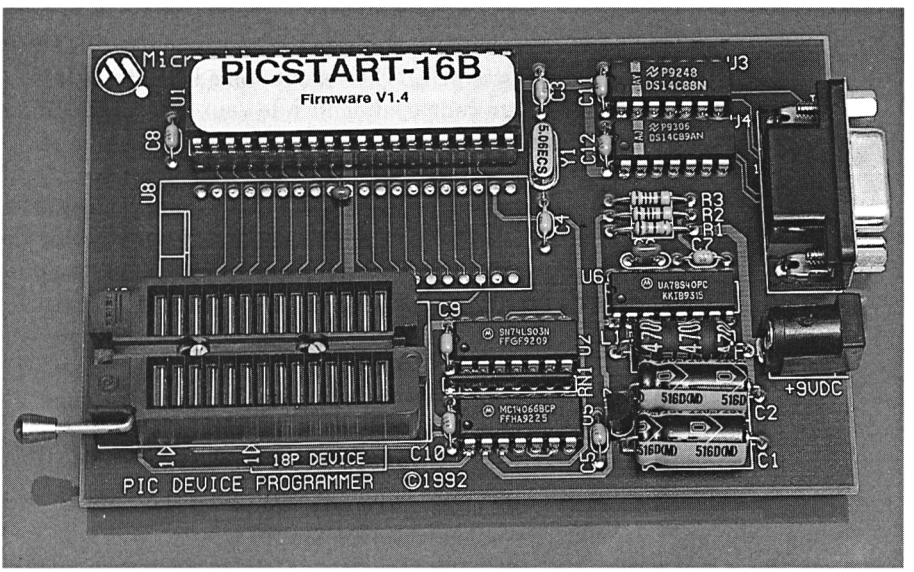
commencer par les deux routines qui reconstituent logiciellement l'UART de communication en série.

Loin de devoir être considéré comme une solution au *rabais* par rapport à un UART matériel, ce choix d'un UART logiciel est courant dans le monde des cartes à puce car il présente l'avantage d'économiser du silicium : on estime en effet que la surface occupée par un UART sur la puce serait comparable à celle pouvant accueillir 1500 octets de ROM, soit pratiquement la totalité de la mémoire du PIC !

Mais un problème s'est naturellement posé quant à la gestion du bit de parité prévu par la norme : le jeu d'instructions des PIC ne permettant pas d'assurer celle-ci de façon simple, nous avons pris le parti (comme en GWBASIC !) d'ignorer carrément, à la réception, le bit de parité ainsi que l'éventuel signal d'erreur dont l'apparition est en pratique exceptionnelle. Dans le programme, cela revient tout simplement à appeler une routine de temporisation déjà disponible, d'une durée égale à celle du bit à *sauter*.

En émission, il n'était par contre pas question de faire ainsi l'impasse sur un bit que contrôlent tous les lecteurs sérieux.

Partant du principe que dans le cadre de nos manipulations essentiellement expérimentales on connaît d'avance les octets à transmettre, nous avons imaginé de prévoir deux routines d'émission distinctes.



Le programmateur PICSTART fourni par Microchip.

L'une (label *even*) incorpore d'autorité un bit de parité à 1, et l'autre (label *odd*) un bit de parité à 0.

On appellera donc soit l'une soit l'autre, selon que le nombre de bits de l'octet à transmettre sera pair ou impair.

Naturellement, un bon exercice consisterait à élaborer une routine de calcul automatique du bit de parité, mais il y a probablement mieux à faire avec les ressources mémoire restantes.

Intéressons nous maintenant aux exemples que nous avons programmés à partir de ces routines de base, et qui devront être exécutés dans l'ordre lors des essais de la *fausse carte* sur le lecteur de la figure 2.2 piloté par le logiciel COREL.BAS.

Le premier exemple fait tout simplement émettre à la carte une réponse au reset en tout point identique à celle des cartes bancaires les plus récentes (CP8 M4 B0' à mémoire EEPROM) :

3F 65 25 08 31 04 6C 90 00

Il ne s'agit nullement d'un quelconque défi, mais d'un moyen commode pour signaler au lecteur que notre carte travaille en «convention inverse» et se contente d'un V_{pp} de 5 V, sans pour autant nous fourvoyer dans les méandres de la programmation d'une réponse au reset *sur mesures*.

Le second exemple, qui affecte à la carte une *classe ISO* BCh (celle des cartes BULL CP8) programme l'attente de la présentation d'un code confidentiel de deux octets (ABCDh) à une adresse quelconque.

On remarquera à cette occasion que, conformément à la norme ISO, le lecteur commence par émettre un bloc de cinq octets appelé *en-tête*, indiquant seulement qu'il se propose de présenter un code :

- BCh (classe ISO de la carte) ;
- 20h (code opération «présentation de code») ;
- un premier octet d'adresse (poids fort) ;
- un second octet d'adresse (poids faible) ;
- un octet précisant la longueur du code (ici 2 octets, soit 02h).

La carte doit immédiatement répondre par un *octet de procédure* identique au code opération de l'instruction *présentation de code* (20h).

Notons que si elle répondait 21h (bit de poids faible à 1), cela signifierait qu'elle réclame l'application de la tension V_{pp}.

Ce n'est qu'après avoir reçu cet octet que le lecteur envoie les deux octets du code confidentiel, auxquels la carte répond une dernière fois par les deux octets 90h et 00h, compte-rendu caractéristique d'une opération exécutée avec succès.

On remarquera qu'à la moindre erreur (classe ISO incorrecte, code opération différent de 20h, code faux, etc.) le programme va *boucler* sur une routine (label *error*) de blocage de la carte dans un farouche mutisme.

Une *vraie* carte émettrait plutôt un compte-rendu précisant la nature de l'incident survenu, comportement que nos lecteurs pourront essayer de programmer à titre d'exercice (c'est très facile !)

On remarquera comment l'appel pur et simple de la routine *recv* sans exploitation de son octet de retour permet de *sauter* un octet auquel on ne s'intéresse pas.

Dans le cas présent, il s'agit des deux octets précisant l'adresse à laquelle le code confidentiel doit être présenté.

La plupart du temps, on présente le code *porteur* (ou PIN) à l'adresse 0000h, mais on pourra vérifier que n'importe quelle adresse fait ici l'affaire.

Si et seulement si le bon code a bien été présenté, la routine suivante peut alors être exécutée, un nombre illimité de fois tant que ne survient pas une irrégularité déviant le programme vers la boucle de mutisme.

Ce troisième exemple programme la carte pour qu'elle émette les deux octets CDh et EFh (suivis d'un compte-rendu 90 00) en réponse à toute demande de lecture de deux octets (code opération B0h) à une adresse quelconque :

```
BCh ; B0h ; ADR1 ; ADR2 ; 02h
```

Bien entendu, un bon exercice serait de spécifier une unique adresse à laquelle ces données pourraient être lues (huit instructions seulement à ajouter !).

Les microcontrôleurs PIC pouvant être protégés contre la relecture de leur programme, on arrive déjà avec très peu de moyens à une fort bonne sécurité des données ainsi incluses dans le corps même du logiciel (et donc dans de la mémoire EPROM).

Pour lire ces deux octets, il faut en effet présenter un code confidentiel de deux octets (256 x 256 soit 65536 combinaisons possibles), et éventuellement spécifier une adresse exprimée elle aussi sur deux octets : une aiguille dans une botte de foin, même si rien n'est encore prévu pour mémoriser (dans un octet d'EEPROM) le nombre de tentatives de présentation d'un code faux et bloquer définitivement la carte après trois essais malheureux. Oui, avec un PIC16C84 ce serait étonnamment simple à mettre en œuvre !

Et puisque nous en sommes au chapitre de la sécurité, il est temps de se livrer à une comparaison objective entre notre carte expérimentale et les *vraies* cartes à microprocesseur.

Il doit être parfaitement clair que notre minisystème d'exploitation, même s'il peut bel et bien reproduire les comportements de base de n'importe quelle carte asynchrone, n'arrive pas à la cheville d'un masque de carte COS ou CP8.

Une *vraie* carte à microprocesseur dispose, parallèlement à une capacité mémoire nettement plus importante, de ressources sécuritaires basées, entre autres, sur des algorithmes de cryptage permettant d'empêcher qu'on puisse intercepter puis reproduire un dialogue entre la carte et son lecteur.

Dépourvue de telles possibilités, notre carte à PIC est donc relativement *piratable* mais ne devrait en aucune façon pouvoir servir à pirater une application correctement protégée.

Bien que la plupart de nos lecteurs sachent fort probablement lire et écrire dans les télécartes depuis des années (nous avons en effet révélé comment procéder dès avant 1991 !), il n'est assurément pas inutile de revenir sur la question dans cet ouvrage.
Bien des choses ont en effet changé, à commencer par l'emplacement de la puce des télécartes françaises, dorénavant en position ISO au lieu d'AFNOR.

Parallèlement, de nombreux pays étrangers se sont *mis à la puce*, et tout porte à croire que ce n'est que le début d'une généralisation mondiale.

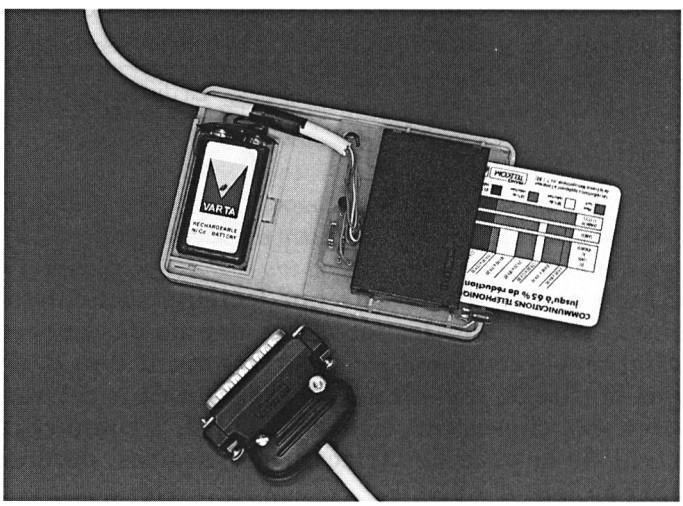
Mais la micro-électronique a aussi évolué, et il nous faut compter maintenant avec les T2G (télécartes françaises de seconde génération) et les cartes EUROCHIP dérivées de la TELEFONKARTE allemande.

A côté de la "vieille" technologie EPROM NMOS des premières télécartes françaises (qu'il faut désormais appeler T1G), toujours fidèle au poste, au moins deux technologies EEPROM attendent donc que nous nous intéressions à elles...

UN MINI-LECTEUR ISO / AFNOR

Même si écrire dans les télécartes (de préférence vides d'unités !) est toujours une expérience passionnante, bien des applications fort intéressantes ne font appel qu'à leur lecture.

Débarrassé des circuits servant à gérer la tension de programmation Vpp nécessaire pour écrire dans les cartes à mémoire EPROM, un lecteur de cartes à puce synchrones pour PC devient un accessoire extrêmement simple, pouvant même être alimenté par pile.



Le lecteur de cartes synchrones achevé.

A l'occasion, ce genre de lecteur pourrait d'ailleurs servir aussi à écrire dans les cartes de seconde génération, dont la mémoire EEPROM se contente d'une tension d'alimentation unique de 5 volts.

Si le schéma de la figure 4.1 est bien évidemment compatible avec les logiciels déjà publiés dans nos précédents ouvrages, il n'en comporte pas moins un certain nombre d'améliorations non négligeables.

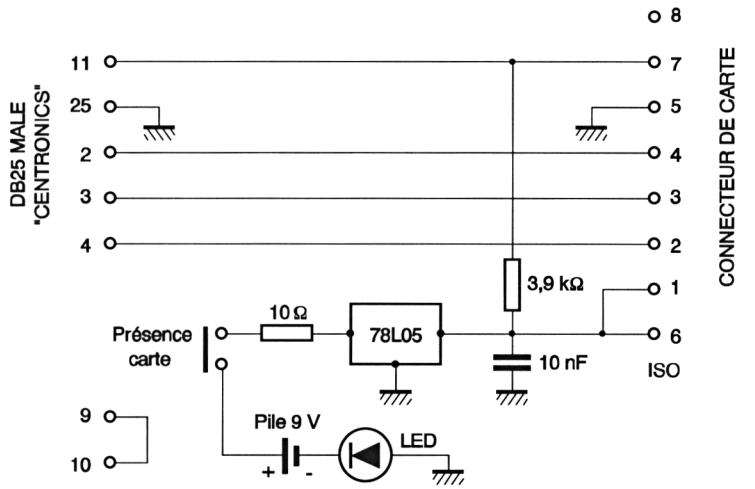


Figure 4.1.
Le schéma du lecteur de cartes synchrones.

Aisément réalisable après-coup sur nos lecteurs déjà décrits, l'interconnexion des broches 9 et 10 de la fiche DB25 mâle permet à nos logiciels les plus perfectionnés (présents sur la disquette de cet ouvrage) de reconnaître automatiquement le port parallèle sur lequel est branché le lecteur (LPT1: ou LPT2:).

Facile à ajouter pour une dépense minime à n'importe quel PC possédant encore un slot libre, un second port Centronics permet en effet de profiter à la fois des services du lecteur de cartes et de ceux de l'imprimante, sans *gymnastique* dangereuse avec les cordons.

Naturellement, le tracé du circuit imprimé de la figure 4.2 exploite les deux jeux de contacts du connecteur (balais ISO et AFNOR montés en parallèle), tandis que les composants de l'alimentation 5 V, câblés selon le plan de la figure 4.3 , ont été optimisés pour assurer une longue autonomie à une simple pile 9 V alcaline (une centaine d'heures environ de présence effective de cartes dans le lecteur, ce qui est considérable).

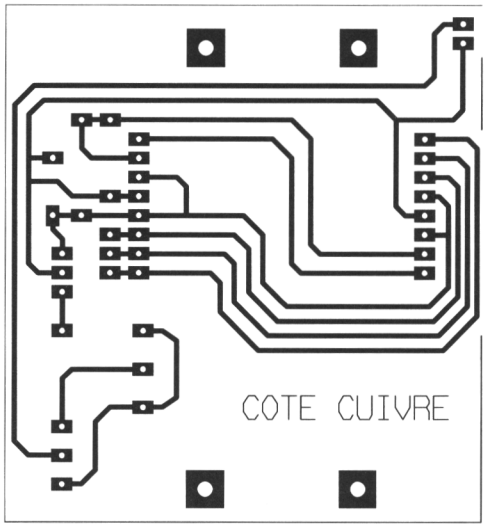


Figure 4.2.
Le circuit imprimé du lecteur de cartes synchrones.

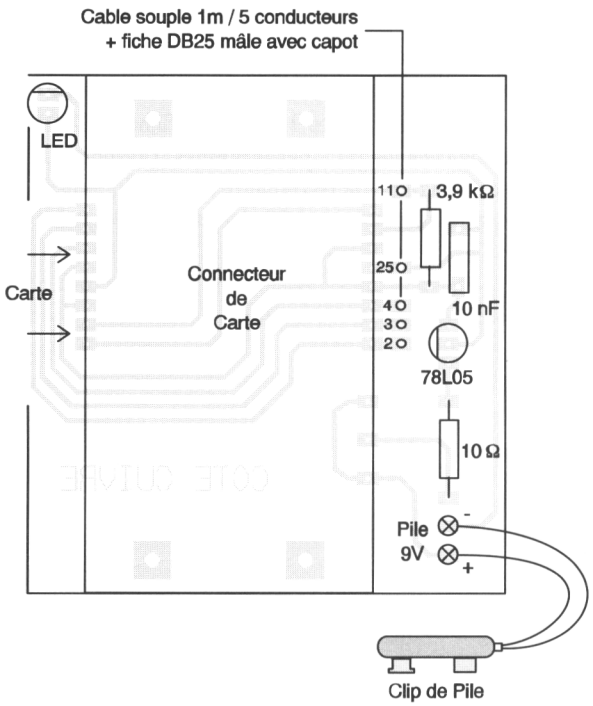


Figure 4.3.
Plan de câblage du lecteur de cartes synchrones.

Les dimensions de ce petit montage permettent une "mise en boîte" commode dans la plupart des coffrets plastique de type calculatrice avec logement de pile.

On y pratiquera simplement une fente d'introduction des cartes en face du connecteur, un trou pour la LED témoin de mise sous tension, et un autre pour le câble de liaison au PC.

Liste des composants

1 connecteur de cartes ITT-CANNON,
1 régulateur 78L05,
1 LED rouge 5 mm,
1 résistance 10 Ω 1/4 W,
1 résistance 3,9 kΩ 1/4 W,
1 pile 9 V et son clip,
1 fiche DB25 mâle,
1 m de câble souple 5 conducteurs,
1 boîtier *calculatrice* avec logement de pile 9 V.

La disquette accompagnant cet ouvrage contient un puissant logiciel intégré supportant la plupart des cartes connues, et évidemment compatible avec ce lecteur : CARTES.EXE, à installer directement sur sa disquette avec INSTALL.EXE et la télécarte de votre choix en guise de *clef logicielle*.

Mais nous avons prévu également un court programme pouvant être considéré comme un *minimum vital*, dont le principal avantage est une modularité qui ménage au maximum les possibilités d'évolution future du système.

```
10 REM —— MINILECT.BAS ——
20 KEY OFF:CLS:DEF SEG=0
30 S2=PEEK(&H40A)+256*PEEK(&H40B) 'pour LPT2:
40 S1=PEEK(&H408)+256*PEEK(&H409) 'pour LPT1:
50 OUT S2,0:E2=S2+1
60 IF (INP(E2) AND 64) <> 0 THEN S=S1:GOTO 100
70 OUT S2,128
80 IF (INP(E2) AND 64) <> 64 THEN S=S1:GOTO 100
90 S=S2
100 E=S+1
110 DIM B$(256)
120 KEY OFF
130 CLS
```

```
140 OUT S,0
150 PRINT"Insérer carte, puis presser ENTER"
160 INPUT Z$
170 CLS
180 OUT S,250:OUT S,248
190 FOR I=1 TO 256
200 OUT S,249
210 D=INP(E):D= (D AND 128)
220 IF D=128 THEN B$(I)="0"
230 IF D<>128 THEN B$(I)="1"
240 OUT S,251
250 NEXT I
260 N=1
270 FOR F=1 TO 8
280 FOR G=1 TO 8
290 FOR H=1 TO 4
300 PRINT B$(N);:N=N+1
310 NEXT H
320 PRINT" ";:NEXT G
330 PRINT:NEXT F
340 PRINT:PRINT:PRINT"Pour sauvegarder, taper un nom de fichier puis ENTER"
350 PRINT:PRINT"Pas de sauvegarde: changer carte et presser ENTER"
360 PRINT:PRINT:INPUT Z$
370 IF Z$="" THEN 170
380 FOR F=1 TO LEN(Z$)
390 IF MID$(Z$,F,1)="." THEN  420
400 NEXT F
410 Z$=Z$+".CAR"
420 OPEN Z$ FOR OUTPUT AS #1
430 N=1
440 FOR F=1 TO 8
450 FOR G=1 TO 8
460 FOR H=1 TO 4
470 PRINT#1, B$(N)+" ";:N=N+1
480 NEXT H
490 PRINT#1," ";:NEXT G
500 PRINT#1,:NEXT F
510 CLOSE#1:GOTO 130
520 REM (c)1995 Patrick GUEULLE
```

En effet, MINILECT.BAS se contente de lire 256 bits consécutifs dans la carte préalablement remise à zéro (reset), mais avec une possibilité de sauvegarde sur disque, dans le format .**CAR** que nous avons défini et utilisé dans nos précédents ouvrages sur la question. Là encore, la compatibilité est assurée !

```
10 REM —— T1G.BAS ——
20 DIM N(256):DIM M(256)
30 KEY OFF:CLS
40 PRINT:GOTO 350
50 CLS:I=0
60 FOR F=1 TO 8
70 FOR G=1 TO 8
80 FOR H=1 TO 4
90 I=I+1
100 N$="1":IF N(I)=0 THEN N$="0"
110 PRINT N$;
120 NEXT H
130 PRINT" ";:NEXT G
140 PRINT
150 IF F=3 THEN PRINT
160 NEXT F:PRINT
170 PRINT"CHOIX + ENTER:":PRINT
180 PRINT"0  ->  Retour au DOS (quitter)"
190 PRINT"1  ->  Chargement"
200 PRINT"2  ->  Analyse"
210 PRINT"3  ->  Affichage HEXA"
220 PRINT"4  ->  Affichage BINAIRE"
230 PRINT"5  ->  Contrôle parité"
240 PRINT"6  ->  Appel du DOS (shell)"
250 INPUT Z$
260 IF Z$="0" THEN SYSTEM
270 IF Z$="6" THEN SHELL:CLS:GOTO 170
280 IF Z$="1" THEN 340
290 IF Z$="2" THEN 480
300 IF Z$="4" THEN 50
310 IF Z$="5" THEN 900
320 IF Z$="3" THEN 1220
330 GOTO 250
340 CLS
350 PRINT" Nom du fichier à charger ?"
360 INPUT S$
370 IF S$="" THEN 340
380 FOR F=1 TO LEN(S$)
390 IF MID$(S$,F,1)="." THEN 420
400 NEXT F
410 S$=S$+".CAR"
420 OPEN S$ FOR INPUT AS #1
430 CLS:PRINT"—— CHARGEMENT EN COURS ——"
440 FOR F=1 TO 256
450 INPUT#1,Q: N(F)=Q
460 NEXT F
470 CLOSE#1:GOTO 50
```

```
480 CLS:PRINT"CODE FAMILLE:                    ";
490 F$="":A=9
500 GOSUB 1130:F$=F$+K$
510 A=13:GOSUB 1130:F$=F$+K$
520 IF N(9)=0 THEN 550
530 PRINT:PRINT"(TELECARTE NON RECONNUE)"
540 PRINT:GOTO 170
550 PRINT"   (TELECARTE)"
560 PRINT"NUMERO DE SERIE:                 ";
570 FOR A=17 TO 29 STEP 4
580 GOSUB 1130:NEXT A
590 FOR A=41 TO 53 STEP 4
600 GOSUB 1130:NEXT A
610 PRINT:PRINT"MESSAGE D'AUTHENTICITE:       ";
620 A=57:GOSUB 1130
630 A=61:GOSUB 1130
640 A=73:GOSUB 1130
650 A=77:GOSUB 1130
660 PRINT:PRINT"PARAMETRES DE PROGRAMMATION: ";
670 A=81:GOSUB 1130
680 IF K$="0" THEN PRINT"   (50 ms / 25 V)"
690 IF K$="1" THEN PRINT"   (50 ms / 21 V)"
700 PRINT"CODE SERVICE:                     ";
710 A=85:GOSUB 1130
720 IF K$="0" THEN PRINT"   (carte jetable)";
730 PRINT:PRINT"POUVOIR FINANCIER:            ";
740 P$=""
750 A=89:GOSUB 1130:P$=P$+K$
760 A=93:GOSUB 1130:P$=P$+K$
770 P=VAL(P$):P=(10*P)-10
780 PRINT"   (";P;" unités)"
790 PRINT"CONSOMMATION:                     ";
800 C=0
810 FOR F=97 TO 248
820 IF N(F)=1 THEN C=C+1
830 NEXT F:C=C-10
840 PRINT C;" UTC"
850 IF C<P THEN PRINT"IL RESTE:       ";P-C;" UTC":BEEP
860 IF C>P THEN PRINT"CARTE ANORMALE":BEEP
870 IF C=P THEN PRINT"CREDIT EPUISE"
880 PRINT:PRINT:GOTO 170
890 GOTO 170
900 CLS:PRINT"CONTROLE DE PARITE:":PRINT
910 IF N(9)=0 THEN 950
920 PRINT"SANS SIGNIFICATION!"
930 PRINT"(TELECARTES seulement)":PRINT
940 GOTO 170
```

```
950 W=1:GOSUB 990
960 W=2:GOSUB 990
970 W=3:GOSUB 990
980 PRINT:GOTO 170
990 PRINT"BLOC N°";W;": ";
1000 K=0
1010 X=(32*W)-31
1020 IF N(X)=0 THEN 1110
1030 FOR F=0 TO 4
1040 X=X+1:IF N(X)=1 THEN K=K+2^(4-F)
1050 NEXT F
1060 Z=0
1070 FOR F=X+1 TO X+26
1080 IF N(F)=0 THEN Z=Z+1
1090 NEXT F
1100 IF Z=K THEN PRINT"parité correcte":RETURN
1110 PRINT"PARITE INCORRECTE":RETURN
1120 END
1130 K=0
1140 FOR J=0 TO 3
1150 B=N(A+J)
1160 IF B=1 THEN K=K+2^(3-J)
1170 NEXT J
1180 IF K<10 THEN K$=CHR$(48+K)
1190 IF K>=10 THEN K$=CHR$(55+K)
1200 PRINT K$;
1210 RETURN
1220 CLS
1230 FOR L=1 TO 225 STEP 32
1240 FOR N=0 TO 28 STEP 4
1250 A=L+N:GOSUB 1130
1260 NEXT N
1270 PRINT:NEXT L
1280 PRINT
1290 GOTO 170
1300 END
1310 C$="(c)1994 Patrick GUEULLE"
```

Strictement réservé aux télécartes françaises de première génération (code famille 03 ou, c'est nouveau, 04), le logiciel T1G.BAS peut venir décrypter les fichiers ainsi extraits des cartes lues par MINILECT.BAS, pour en tirer un maximum d'informations dont la figure 4.4 montre un échantillonnage.

On notera que la fonction *contrôle de parité*, inédite jusqu'à présent, permet de vérifier que le contenu de la *zone de fabrication* (les 96 premiers bits protégés par le fameux *fusible*) est vraisemblable : un seul bit non conforme ferait échouer le test.

```
1101 0011 0000 0100 0000 0000 0001 0110
1100 1111 0010 0000 0110 1000 0001 0000
1100 0111 1000 1011 0001 0000 0000 0110

1111 1111 1111 1111 1111 1111 1111 1111
1111 1111 1111 1111 1111 1111 1111 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 1111 1111

CHOIX + ENTER:

0  -->  Retour au DOS (quitter)
1  -->  Chargement
2  -->  Analyse
3  -->  Affichage HEXA
4  -->  Affichage BINAIRE
5  -->  Contrôle parité
6  -->  Appel du DOS (shell)
?


CODE FAMILLE:                    04  (TELECARTE)
NUMERO DE SERIE:                 00162068
MESSAGE D'AUTHENTICITE:          108B
PARAMETRES DE PROGRAMMATION:     1  (50 ms / 21 V)
CODE SERVICE:                    0  (carte jetable)
POUVOIR FINANCIER:               06  ( 50  unités)
CONSOMMATION:                    50  UTC
CREDIT EPUISE


CONTROLE DE PARITE:

BLOC N° 1 : parité correcte
BLOC N° 2 : parité correcte
BLOC N° 3 : parité correcte


D3040016
CF206810
C78B1006
FFFFFFFF
FFFFFFF0
00000000
00000000
000000FF
```

Figure 4.4.
Les possibilités du logiciel T1G.BAS.

Nouveau lui aussi, le décodage du *message d'authenticité* permettrait de différencier deux cartes portant éventuellement le même numéro de série.

L'affichage hexadécimal offre pour sa part une *image* particulièrement compacte, et plus parlante pour certains que de simples blocs de quatre bits.

Nous découvrirons plus loin dans ce chapitre l'équivalent de ce logiciel pour les T2G et pour les cartes EUROCHIP, tandis qu'il ne serait pas bien compliqué de développer des versions spécifiques pour d'autres familles de cartes (par exemple des télécartes étrangères à mémoire EPROM).

La disquette d'accompagnement de cet ouvrage contient d'ailleurs, dans son répertoire **IMAGES**, une sélection de véritables fichiers **.CAR** permettant de commencer à expérimenter même sans lecteur ni cartes !

RECONNAISSANCE LOGICIELLE DES TYPES DE PUCES

Bien que les collectionneurs de télécartes appellent volontiers *puce* le *micromodule*, ce n'est pas cela, la puce électronique proprement dite, cette fameuse mémoire dans laquelle sont emmagasinées les unités !

Petit carré de silicium d'environ un millimètre de côté seulement, le véritable coeur des cartes à puce est en réalité collé au dos de ce groupe de huit (bientôt six) contacts.

Noyée dans l'épaisseur du plastique de la carte, la puce mémoire reste donc le plus souvent invisible.

On la devine plus ou moins au verso des cartes à *pastilles* transparentes, marron ou blanches, mais pas sur les cartes à pastille noire ou sans pastille du tout.

Sur certains specimens, notamment de fabrication Schlumberger, la résine transparente est suffisamment limpide pour qu'on arrive à voir plus ou moins correctement la puce.

Il apparaît ainsi que toutes les télécartes ne sont pas équipées du même modèle de puce silicium, dont de patientes recherches nous ont permis de mettre en évidence au moins trois versions.

La plus ancienne est facilement reconnaissable aux grandes lettres «ET» (cinq centièmes de millimètre de haut...) inscrites dans l'angle d'une vaste surface bleutée pratiquement visible à l'œil nu (voir photo page 145).

Il faut par contre un microscope pour arriver à lire la date du masque (1983) gravée vers le centre de la puce : les chiffres sont hauts d'à peine 20 µm, c'est-à-dire de même pas deux centièmes de millimètre!

Plus récente et concurrente, la puce TMS3561 de TEXAS INSTRUMENTS porte la date de masque 1986. Un peu plus petite que la puce ET1001, elle est facilement identifiable (au microscope) par le logo de la marque (un T et un I imbriqués) gravé dans un coin.

On la rencontre couramment sur certaines séries de télécartes de fabrication SCHLUMBERGER ou SOLAIC, mais nous n'en avons par contre jamais trouvé encore dans des cartes GEMPLUS. Cela pourrait être une conséquence des liens historiques de la firme avec SGS-THOMSON.

Mais la puce la plus commune, celle qui équipe la plupart des télécartes fabriquées aujourd'hui et qui ne sera sans doute détrônée que par la T2G, est la TS1001, précisément de marque SGS-THOMSON (date de masque 1987).

C'est la plus petite des trois (et donc sans doute la plus économique à fabriquer car la surface coûte cher), et elle ressemble fort à une ET1001 de laquelle on aurait éliminé toute la place perdue.

Il est intéressant de remarquer que les puces TEXAS et SGS-THOMSON ne réagissent pas tout à fait de la même manière aux opérations de lecture : les puces TEXAS présentent le bit lu sur le contact ISO7 juste après le front montant de l'impulsion d'horloge, tandis que les puces SGS-THOMSON attendent pour ce faire le retour à zéro du signal d'horloge.

Il n'en fallait pas davantage pour que nous imaginions un petit algorithme de reconnaissance de puces, qui se traduit en pratique par le programme **TEXAS.BAS**.

```
10 REM ---- TEXAS ----
20 DEF SEG=0:KEY OFF:CLS
30 S2=PEEK(&H40A)+256*PEEK(&H40B)  'pour LPT2:
40 S1=PEEK(&H408)+256*PEEK(&H409)  'pour LPT1:
50 OUT S2,0:E2=S2+1
60 IF (INP(E2) AND 64) <> 0 THEN S=S1:GOTO 100
70 OUT S2,128
80 IF (INP(E2) AND 64) <> 64 THEN S=S1:GOTO 100
90 S=S2
100 E=S+1
110 UC=0:UD=0:ZC=0:ZD=0:OUT S,0
120 PRINT"Insérer carte, puis presser ENTER"
130 INPUT Z$:CLS
140 OUT S,250:OUT S,248
150 FOR F=1 TO 16
160 D=INP(E):D= (D AND 128)
170 IF D=128 THEN ZD=ZD+1
180 IF D<>128 THEN UD=UD+1
190 OUT S,249
200 C=INP(E):C= (C AND 128)
210 IF C=128 THEN ZC=ZC+1
220 IF C<>128 THEN UC=UC+1
230 OUT S,251
240 NEXT F:PRINT
250 IF ZC=0 OR UC=0 THEN PRINT "TELECARTE ANORMALE":GOTO 280
260 IF ZC=ZD THEN PRINT "PUCE TMS3561 (TEXAS) ou EEPROM"
270 IF UD=16 AND UC<16 THEN PRINT "PUCE ET1001 ou TS1001 (THOMSON)"
280 PRINT:PRINT:GOTO 110
290 REM (c)1995 Patrick GUEULLE
```

Mais nous l'avons également inclus dans notre logiciel intégré **CARTES.EXE**, car son intérêt est évident pour les collectionneurs : la présence d'une puce plutôt que d'une autre à l'intérieur de certaines variétés de télécartes peut en effet dans certains cas constituer une rareté de grande valeur !

TÉLÉCARTES ET PROTECTION DE LOGICIELS

Si notre logiciel **CARTES.EXE** est muni d'une protection contre la copie par télécarte-clef, c'est surtout à titre de démonstration de ce que l'on peut obtenir par des moyens somme toute fort simples.

A vrai dire, toute personne disposant de l'utilitaire **INSTALL.EXE** peut créer autant de copies qu'elle possède de télécartes pouvant servir de clefs.

En revanche, **CARTES.EXE** installé sur une disquette (ou un disque dur) ne contenant pas (ou plus) INSTALL.EXE ne pourra être lançé qu'en présence de la seule télécarte-clef ayant servi à son installation.

Vous pourrez protéger selon le même principe (mais avec un algorithme légèrement différent) vos propres programmes compilés sous TURBO-BASIC grâce à l'utilitaire PROTECT.BAS.

```
10 REM ---- PROTECT.BAS ----
20 DEF SEG=0:KEY OFF
30 S1=PEEK(&H408)+256*PEEK(&H409)
40 S2=PEEK(&H40A)+256*PEEK(&H40B)
50 OUT S2,0:E2=S2+1
60 IF (INP(E2) AND 64) <> 0 THEN S=S1:GOTO 100
70 OUT S2,128
80 IF (INP(E2) AND 64) <> 64 THEN S=S1:GOTO 100
90 S=S2
100 E=S+1
110 DIM N(256):DIM M(256)
120 CLS:PRINT"PROTECTION DE LOGICIEL PAR TELECARTE"
130 PRINT"=================================="
140 PRINT:PRINT
150 IF S=S2 THEN PRINT"     (lecteur sur LPT2:)"
160 IF S=S1 THEN PRINT"     (lecteur sur LPT1:)"
170 PRINT:PRINT"Insérer la carte-clef, puis presser ENTER"
180 INPUT Z$:CLS:KK$=""
190 OUT S,250:OUT S,248
200 FOR F=1 TO 12
210 K=0
220 FOR G=0 TO 7
230 OUT S,249
240 D=INP(E):D=(D AND 128)
250 IF D<>128 THEN K=K+2^(7-G)
260 OUT S,251:NEXT G
270 K=K XOR 55
280 KK$=KK$+CHR$(K):NEXT F
290 OUT S,0
300 OUT S,250:OUT S,248
310 OPEN "1995.(c)" FOR OUTPUT AS #3
320 KK$=KK$+" (c)1995 Patrick GUEULLE"
330 PRINT #3,KK$
340 CLOSE #3
350 PRINT KK$:PRINT:PRINT
360 END
370 REM (c)1995 Patrick GUEULLE
380 RETURN
```

La marche à suivre est simple : il faut incorporer le module de protection PROTEGE.BAS dans le code source du logiciel à protéger (écrit en GWBASIC), en faisant en sorte qu'il soit exécuté en tout premier lieu lors d'un RUN.

```
1 REM ---- PROTEGE.BAS ----
2 DEF SEG=0:KEY OFF
3 S1=PEEK(&H408)+256*PEEK(&H409)
4 S2=PEEK(&H40A)+256*PEEK(&H40B)
5 OUT S2,0:E2=S2+1
6 IF (INP(E2) AND 64) <> 0 THEN S=S1:GOTO 15
7 OUT S2,128
8 IF (INP(E2) AND 64) <> 64 THEN S=S1:GOTO 15
9 S=S2
15 E=S+1
20 DIM N(256):DIM M(256)
30 CLS:PRINT"LOGICIEL PROTEGE PAR TELECARTE"
40 PRINT"============================="
41 PRINT" (c)1992,1995 Patrick GUEULLE":PRINT
45 IF S=S2 THEN PRINT"     (lecteur sur LPT2:)"
46 IF S=S1 THEN PRINT"     (lecteur sur LPT1:)"
47 GOSUB 5000
50 CLS:FOR F=1 TO 10
55 PRINT"CARTE-CLEF RECONNUE, EXECUTION AUTORISEE"
60 PRINT:NEXT F
4999 END
5000 PRINT:PRINT"Insérer la carte-clef, puis presser ENTER"
5010 BEEP:INPUT Z$:CLS:KK$=""
5020 OUT S,250:OUT S,248
5030 FOR F=1 TO 15
5032 K=0
5035 FOR G=0 TO 7
5060 OUT S,249
5070 D=INP(E):D=(D AND 128)
5080 IF D<>128 THEN K=K+2^(7-G)
5090 OUT S,251:NEXT G
5091 K=K XOR 55
5099 KK$=KK$+CHR$(K):NEXT F
5100 OUT S,0
5150 OUT S,250:OUT S,248
5200 OPEN "1995.(c)" FOR INPUT AS #3
5210 INPUT#3,XX$
5220 IF LEFT$(XX$,12)=LEFT$(KK$,12) THEN RETURN
5230 CLS:PRINT:PRINT"CARTE-CLEF NON RECONNUE":PRINT:END
5250 CLOSE #3
5300 REM(c)1995 Patrick GUEULLE
5999 RETURN
```

Cela fait, on compilera le tout sous TURBO-BASIC de façon à obtenir un fichier .EXE, dont il n'est d'ailleurs pas interdit de diminuer la taille à l'aide de LZEXE ou d'un compacteur équivalent.

Ce même exécutable pourra être copié directement sur toutes les disquettes que l'on se propose de distribuer, mais il ne fonctionnera pas sans un *fichier secret* élaboré par **PROTECT.BAS**.

Ledit fichier sera préparé à partir de la télécarte qu'on fournira en tant que clef (accompagnée du lecteur ad-hoc), ou dont on montera la puce dans un *dongle* maison (voir notre ouvrage *Montages à composants programmables* dans cette même collection).

Imaginons que le logiciel à protéger, déjà compilé (par exemple PROTEGE.EXE) soit stocké avec PROTECT.EXE (version compilée de PROTECT.BAS) sur un disque placé dans le lecteur B.

Pour produire une version protégée sur une disquette placée dans le lecteur A, on fera successivement les opérations suivantes à partir de A :

1. COPY B:PROTEGE.EXE

2. B:PROTECT

Bien entendu, on insèrera une télécarte dans le lecteur approprié lorsque PROTECT.EXE le demandera, puis on rangera celle-ci avec la disquette qui lui est maintenant liée, sur laquelle un fichier supplémentaire nommé «1995.(c)» s'est trouvé créé.

Lors de chacun de ses lancements, le logiciel protégé demandera l'introduction de la télécarte-clef dans son lecteur, la lira, et comparera son contenu à son *image* cryptée contenue dans le fichier secret. S'il y a concordance, alors le logiciel s'exécutera normalement, mais dans le cas contraire seul un message d'erreur s'affichera.

Bien entendu, chacun pourra apporter des modifications de son cru à l'algorithme utilisé par PROTECT.BAS, à condition de faire exactement de même dans le module de protection du logiciel à protéger.

Le plus simple est d'intervenir au niveau du OU exclusif, mais on pourrait aussi songer à utiliser un fichier caché, ou toute autre astuce personnelle.

UN LECTEUR VRAIMENT UNIVERSEL

Bâti autour d'un **coupleur** universel d'origine INNOVATRON, le lecteur de cartes à microprocesseur décrit au chapitre 2 est aussi capable de lire et d'écrire dans virtuellement n'importe quelle carte à puce synchrone.

Ceux de nos lecteurs qui l'ont déjà réalisé souhaiteront probablement s'en servir à l'occasion avec des télécartes.

Le logiciel COREL256.BAS est précisément prévu à cet effet, et peut donc rendre exactement les mêmes services que MINILECT.BAS et son lecteur simplifié.

```
10 REM — COREL256 —
20 DIM M(256):KEY OFF:CLS
30 OPEN"com1:9600,n,8,1" AS #1
40 PRINT"COUPLEUR DE CARTES A MEMOIRE CCU910 COREL"
50 PRINT"========================================="
60 PRINT:PRINT:PRINT"INTRODUIRE UNE GPM256, puis presser ENTER"
70 INPUT Z$:CLS
80 S$="00"
90 CLS:PRINT"INITIALISATION DU COUPLEUR":PRINT
100 GOSUB 700
110 PRINT"MISE SOUS TENSION ET RESET":PRINT
120 GOSUB 610
130 IF E$<>"04" THEN 580
140 PRINT"LECTURE DE LA CARTE:    presser L puis ENTER"
150 PRINT"PROGRAMMATION DE LA CARTE: presser P puis ENTER":PRINT
160 INPUT Z$:PRINT
170 IF Z$<>"p" AND Z$<>"P" AND Z$<>"l" AND Z$<>"L" THEN CLS:GOTO 140
180 IF Z$="p" OR Z$="P" THEN 790
190 IF Z$="l" OR Z$="L" THEN GOSUB 730
200 PRINT"Nom du fichier .CAR à créer ";
210 INPUT F$:IF F$="" THEN F$="nul"
220 FOR F=1 TO LEN(F$)
230 IF MID$(F$,F,1)="." THEN 260
240 NEXT F
250 F$=F$+".CAR":PRINT:PRINT
260 OPEN F$ FOR OUTPUT AS #2
```

```
270 C$=MID$(C$,7,32):N=0
280 FOR F=1 TO 32
290 CC$=MID$(C$,F,1):CC=ASC(CC$)
300 N=N+1:IF CC>127 THEN M(N)=1 ELSE M(N)=0
310 IF CC>127 THEN CC=CC-128
320 N=N+1:IF CC>63 THEN M(N)=1 ELSE M(N)=0
330 IF CC>63 THEN CC=CC-64
340 N=N+1:IF CC>31 THEN M(N)=1 ELSE M(N)=0
350 IF CC>31 THEN CC=CC-32
360 N=N+1:IF CC>15 THEN M(N)=1 ELSE M(N)=0
370 IF CC>15 THEN CC=CC-16
380 N=N+1:IF CC>7 THEN M(N)=1 ELSE M(N)=0
390 IF CC>7 THEN CC=CC-8
400 N=N+1:IF CC>3 THEN M(N)=1 ELSE M(N)=0
410 IF CC>3 THEN CC=CC-4
420 N=N+1:IF CC>1 THEN M(N)=1 ELSE M(N)=0
430 IF CC>1 THEN CC=CC-2
440 N=N+1:IF CC>0 THEN M(N)=1 ELSE M(N)=0
450 NEXT F
460 I=0
470 FOR F=1 TO 8
480 FOR G=1 TO 8
490 FOR H=1 TO 4
500 I=I+1
510 NN$="1":IF M(I)=0 THEN NN$="0"
520 PRINT NN$;:PRINT#2,NN$+" ";
530 NEXT H
540 PRINT" ";:PRINT#2," ";:NEXT G
550 PRINT:PRINT#2,
560 IF F=3 THEN PRINT
570 NEXT F:PRINT:CLOSE#2
580 PRINT"MISE HORS TENSION DE LA CARTE":PRINT
590 GOSUB 670
600 END
610 PRINT#1,CHR$(2)+CHR$(2)+CHR$(0)+CHR$(2)+CHR$(3);
620 GOSUB 1270
630 IF E$="03" THEN BEEP:PRINT "CARTE ASYNCHRONE (à microprocesseur) DETECTEE"
640 IF E$="04" THEN PRINT "CARTE SYNCHRONE (à logique câblée) DETECTEE"
650 PRINT
660 RETURN
670 PRINT#1,CHR$(2)+CHR$(3)+CHR$(0)+CHR$(3)+CHR$(3);
680 GOSUB 1270
690 RETURN
700 PRINT#1,CHR$(2)+CHR$(12)+CHR$(1)+CHR$(1)+ CHR$(12)+CHR$(3);
710 GOSUB 1270
720 RETURN
730 REM lecture
```

```
740 A$="0000":Q$="20"
750 X$="020005"+S$+"B0"+A$+Q$
760 GOSUB 1510
770 GOSUB 1270
780 RETURN
790 PRINT"Nom du fichier .CAR à programmer ";
800 INPUT P$
810 IF P$="" THEN CLS:GOTO 580
820 FOR F=1 TO LEN(P$)
830 IF MID$(F$,F,1)="." THEN 860
840 NEXT F
850 P$=P$+".CAR":PRINT
860 OPEN P$ FOR INPUT AS #3
870 A$="0000":H$=""
880 FOR F=1 TO 32
890 C=0
900 FOR G=1 TO 8
910 INPUT#3,Q
920 IF Q=1 THEN C=C+2^(8-G)
930 NEXT G
940 C$=HEX$(C):IF LEN(C$)<2 THEN C$="0"+C$
950 H$=H$+C$
960 NEXT F
970 CLOSE#3
980 U$="25"
990 L$="20"
1000 X$="0201"+U$+S$+"D0"+A$+L$+H$
1010 GOSUB 1510
1020 FOR T=0 TO 5000:NEXT T
1030 GOSUB 1270
1040 IF M$<>"9001" THEN 1260
1050 PRINT"EVITER LA ZONE PROTEGEE ? O/N + ENTER"
1060 INPUT Z$:IF Z$="o" OR Z$="O" THEN 1070 ELSE 580
1070 OPEN P$ FOR INPUT AS #3:H$=""
1080 PRINT:FOR F=1 TO 96
1090 INPUT#3,Q:NEXT F
1100 FOR F=1 TO 20
1110 C=0
1120 FOR G=1 TO 8
1130 INPUT#3,Q
1140 IF Q=1 THEN C=C+2^(8-G)
1150 NEXT G
1160 C$=HEX$(C):IF LEN(C$)<2 THEN C$="0"+C$
1170 H$=H$+C$
1180 NEXT F
1190 CLOSE#3
1200 U$="19"
```

```
1210 L$="14"
1220 X$="0201"+U$+S$+"D0"+"000C"+L$+H$
1230 GOSUB 1510
1240 FOR T=0 TO 5000:NEXT T
1250 GOSUB 1270
1260 GOTO 580
1270 FOR T=0 TO 2000:NEXT T
1280 R$=""
1290 IF LOC(1)=0 THEN PRINT"COUPLEUR MUET":GOTO 1500
1300 C$=INPUT$(LOC(1),#1)
1310 J=0:PRINT"REPONSE DU COUPLEUR:"
1320 FOR F=1 TO LEN(C$)
1330 D$=HEX$(ASC(MID$(C$,F,1)))
1340 IF LEN(D$)<2 THEN D$="0"+D$
1350 R$=R$+D$
1360 PRINT D$+CHR$(32);:J=J+1
1370 IF J>23 THEN J=0:PRINT
1380 NEXT F
1390 PRINT
1400 M$=MID$(R$,9,4)
1410 E$=MID$(R$,7,2)
1420 IF M$="9001"THEN PRINT"PROBLEME D'ECRITURE":BEEP
1430 IF M$="6E00"THEN PRINT"CLASSE ISO INCORRECTE":BEEP
1440 IF M$="9000"THEN PRINT:PRINT"BONNE EXECUTION"
1450 IF E$="02"THEN PRINT"CARTE NON TRAITEE":BEEP
1460 IF E$="FD"THEN PRINT"CARTE ARRACHEE":BEEP
1470 IF E$="FF"THEN PRINT"CARTE MUETTE":BEEP
1480 IF E$="FA"THEN PRINT"CARTE ABSENTE":BEEP
1490 IF E$="FB"THEN PRINT"COURT-CIRCUIT CARTE":BEEP
1500 PRINT:PRINT:RETURN
1510 L=0
1520 PRINT"TRAME ENVOYEE AU COUPLEUR:"
1530 FOR F=3 TO (LEN(X$)-1) STEP 2
1540 W$="&h"+MID$(X$,F,2)
1550 L=L XOR VAL(W$)
1560 L$=HEX$(L)
1570 IF LEN(L$)<2 THEN L$="0"+L$
1580 NEXT F
1590 X$=X$+L$+"03"
1600 FOR F=1 TO (LEN(X$)-1) STEP 2
1610 J=0
1620 J=J+1:PRINT MID$(X$,F,2)+CHR$(32);
1630 IF J>23 THEN J=0:PRINT
1640 W$="&h"+MID$(X$,F,2)
1650 PRINT#1,CHR$(VAL(W$));
1660 NEXT F
1670 PRINT:PRINT
```

```
1680 RETURN
1690 H$="(c)1995 Patrick GUEULLE"
```

Les recopies d'écran de la figure 4.5 montrent toutefois qu'il faut transiter par le format hexadécimal, puisque le coupleur opère exclusivement sur des octets complets.

Qu'à cela ne tienne, notre logiciel se charge de la conversion, et produit donc des fichiers **.CAR** parfaitement compatibles avec nos autres utilitaires, à commencer par **T1G.BAS**.

Notons que si on se contente de presser **ENTER** sans donner de nom de fichier, le contenu binaire de la carte s'affiche seulement à l'écran, sans sauvegarde sur disque.

Mais COREL256.BAS est aussi capable d'écrire dans les cartes à mémoire EPROM 256 bits que sont les télécartes de première génération et les GPM256, en vérifiant d'ailleurs bit après bit le succès de l'opération.

```
REPONSE DU COUPLEUR:
02 02 0D 04 C3 03 38 32 B7 69 17 51 B7 DF 10 13 32 03


CARTE SYNCHRONE (à logique câblée) DETECTEE

LECTURE DE LA CARTE:       presser L puis ENTER
PROGRAMMATION DE LA CARTE: presser P puis ENTER

? l

TRAME ENVOYEE AU COUPLEUR:
02 00 05 00 B0 00 00 20 95 03

REPONSE DU COUPLEUR:
02 00 23 04 90 00 C3 03 38 32 B7 69 17 51 B7 DF 10 13 FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF C0 00 00 FF B1 03

BONNE EXECUTION


Nom du fichier .CAR à créer ?



1100 0011 0000 0011 0011 1000 0011 0010
1011 0111 0110 1001 0001 0111 0101 0001
1011 0111 1101 1111 0001 0000 0001 0011

1111 1111 1111 1111 1111 1111 1111 1111
1111 1111 1111 1111 1111 1111 1111 1111
1111 1111 1111 1111 1111 1111 1111 1111
1111 1111 1111 1111 1111 1111 1111 1111
1100 0000 0000 0000 0000 0000 1111 1111

MISE HORS TENSION DE LA CARTE

REPONSE DU COUPLEUR:
02 03 01 04 06 03
```

Figure 4.5.
Lecture d'une télécarte avec COREL256.BAS.

Ce logiciel est par conséquent capable de *brûler* un fichier .CAR dans une GPM256 vierge ou non, mais attention : on ne peut toujours que transformer des **0** en **1** et jamais le contraire, tandis qu'aucune modification n'est possible parmi les 96 premiers bits si le fusible de protection est détruit.

C'est pour cette raison que nous avons prévu une option permettant, en cas d'échec d'une programmation dû à une tentative de modification illicite dans cette zone, de renouveler l'opération en *sautant* purement et simplement les 96 premiers bits.

LA T2G OU TÉLÉCARTE DE SECONDE GÉNÉRATION

Depuis 1983, l'industrie des semiconducteurs a considérablement évolué : la technologie NMOS, qui sert encore à produire les puces des actuelles télécartes (T1G), est arrivée au seuil de l'obsolescence et devrait être progressivement abandonnée par les fabricants.

FRANCE TELECOM a donc imaginé dès 1989 de passer en CMOS, quitte à profiter de l'occasion pour améliorer les fonctionnalités du produit sans pour autant en augmenter le prix de revient.

Après une phase d'expérimentation *terrain* sur 100 000 cartes fin 1993, l'adaptation de 120 000 publiphones à carte a en principe été opérée courant 1994.

Parfaitement transparente pour l'usager, cette opération se traduira cependant un jour ou l'autre par l'impossibilité d'utiliser les T1G restant en circulation.

Le remplacement de l'actuelle mémoire EPROM par une EEPROM spécifique développée en partenariat avec SGS-THOMSON permet de concilier une simplification du matériel (six contacts au lieu de huit, pas de V_{pp} externe), avec une plus grande souplesse du produit.

Il serait par exemple imaginable de programmer une T2G pour qu'elle ne puisse servir à appeler qu'un seul et unique numéro.

Réinscriptible (et donc peut-être un jour rechargeable ?), une T2G pourrait contenir des milliers d'unités : c'est un

atout pour l'exportation, de nombreux pays appliquant une taxation beaucoup plus fine qu'en France (parfois à la seconde près !)

Grâce à un mécanisme d'authentification par calcul de signature à partir de clefs secrètes internes, la T2G est également plus sûre que la T1G bien que FRANCE TELECOM, qui dispose de puissants moyens de dépistage de toute forme de fraude, affirme catégoriquement que ce problème est aujourd'hui réglé.

Mais en la matière, il est vital d'avoir toujours quelques longueurs d'avance sur l'adversaire potentiel !

Rien d'étonnant donc à ce que les caractéristiques techniques détaillées des T2G soient tenues secrètes. Aimablement fournies par SGS-THOMSON, les informations que nous allons néanmoins dévoiler ici se rapportent donc au composant ST1333, celui équipant les T2G semblant s'appeler plutôt ST1332.

Fort heureusement, les différences apparaissent suffisamment minimes pour ne pas nous gêner dans nos tentatives de lecture.

La figure 4.6 reproduit donc la cartographie mémoire supposée des T2G, à partir de laquelle nous avons mené nos investigations avec, il faut l'avouer, un certain succès.

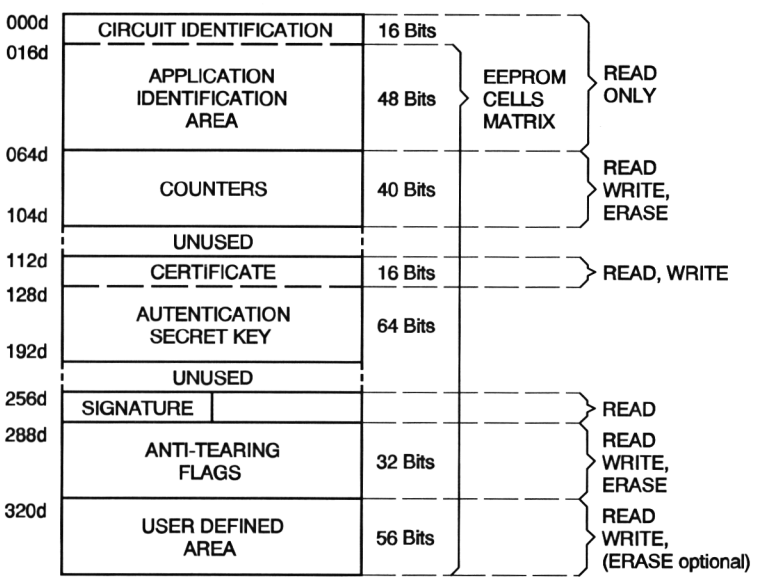


Figure 4.6. Cartographie mémoire supposée des T2G.

Il apparaît immédiatement que l'espace adressable s'étend sur 376 bits, c'est à dire 120 de plus que dans le cas de la T1G.

La zone intéressante à lire coïncide toutefois avec les 256 premiers bits, ce qui signifie qu'on pourra se servir du même format de fichiers .CAR, et dans une certaine mesure des mêmes lecteurs et des mêmes logiciels de lecture (MINILECT.BAS, COREL256.BAS, et naturellement CARTES.EXE).

La figure 4.7 montre en effet que le protocole de communication employé (dit *à 6 contacts*) est compatible avec celui des T1G. Simplement, une micro-instruction de comparaison de bit fait son apparition, peut-être pour supporter la procédure de sécurisation cryptographique au sujet de laquelle nous n'avons pu apprendre que fort peu de choses : elle s'apparenterait à la fois au DES et au RSA, par le biais d'une *signature* de 4 bits, d'un *certificat* de 16 bits, et d'une *clef secrète d'authentification* (illisible de l'extérieur) de 64 bits. Rien de moins !

ISO2	ISO4	ISO3	MICRO-INSTRUCTION
0	0		RESET
1	0		sans effet
0	1		READ (UP)
0	1		COMPARE
1	1		PROGRAM "1" (ISO7 à Vcc)

PROTOCOLE "6 CONTACTS" : T2G, GPM256

Figure 4.7.
Le jeu de «micro-instructions» supposé des T2G.

D'un autre côté, des logiciels d'analyse très spécifiques vont devoir être mis en oeuvre, car les unités sont comptabilisées d'une façon radicalement différente.

Après le *numéro de silicium* à 16 bits qui paraît pour l'instant être commun à toutes les T2G (8140h), on trouve 48 bits abritant notamment le numéro de série de la carte et son *pouvoir financier* (indication codée du nombre d'unités dont elle a été chargée, 05h pour 50 unités).

Suivent 40 bits organisés en cinq *compteurs* de huit bits chacun.

Grâce à un *plan de comptage* inspiré du principe des bouliers, ces 40 bits permettent de comptabiliser jusqu'à 32768 unités !

En effet, la technologie EEPROM présente cette particularité remarquable d'être *réinscriptible* : un **0** peut être transformé en **1**, mais un **1** peut tout aussi bien être transformé en **0**.

Dans le cas de la T2G, on peut librement transformer, de l'extérieur, des **0** en **1** pour consommer des unités. Mais c'est la carte elle-même, sous le contrôle de sa logique interne, qui a le privilège de remettre à **0** des bits qui étaient à **1**.

Bien entendu, cette manoeuvre n'étant pas, dans l'état actuel des choses, destinée à recharger la carte, il faut débiter ici ce que l'on crédite là.

Chaque bit d'un compteur donné ayant une valeur, exprimée en unités, huit fois plus forte que celle d'un bit du compteur qui le suit dans l'ordre des adresses, l'astuce consiste à mettre simultanément un bit à **1** dans un compteur et huit bits à **0** dans le compteur suivant.

En pratique, on ne remet toutefois que sept bits à **0** puisque l'opération s'effectue à l'occasion de la consommation d'une unité : chaque compteur actif contient par conséquent toujours au moins un bit à **1**.

Notre logiciel T2G.BAS est capable de déterminer le crédit d'une carte à partir du fichier .CAR lu dans celle-ci.

```
10 REM —- T2G.BAS —-
20 KEY OFF:CLS
30 PRINT"NOM DU FICHIER .CAR à ANALYSER  ";
40 INPUT N$
50 IF N$="" THEN END
60 FOR F=1 TO LEN(N$)
70 IF MID$(N$,F,1)="." THEN 100
80 NEXT F
90 N$=N$+".CAR"
100 OPEN N$ FOR INPUT AS #1
```

```
110 A=0
120 FOR F=0 TO 15
130 INPUT#1,Q
140 IF Q=1 THEN A=A+2^(15-F)
150 NEXT F
160 IF A<>2^15+2^8+2^6 THEN 500
170 FOR F=16 TO 59
180 INPUT#1,Q
190 NEXT F
200 A=0
210 FOR F=60 TO 63
220 INPUT#1,Q
230 IF Q=1 THEN A=A+2^(63-F)
240 NEXT F
250 IF A<>5 THEN 500
260 PRINT"T2G de";A*10;"unités,    ";
270 U=0
280 FOR F=64 TO 71
290 INPUT#1,Q
300 NEXT F
310 FOR F=72 TO 78
320 INPUT#1,Q
330 IF Q=1 THEN U=U+64
340 NEXT F
350 INPUT#1,Q
360 FOR F=80 TO 86
370 INPUT#1,Q
380 IF Q=1 THEN U=U+8
390 NEXT F
400 INPUT#1,Q
410 FOR F=88 TO 94
420 INPUT#1,Q
430 IF Q=1 THEN U=U+1
440 NEXT F
450 U=(10*A)-U
460 PRINT"CREDIT: ";U;" UTC"
470 IF U=0 THEN PRINT:PRINT"(CREDIT EPUISE)"
480 IF U<>0 THEN BEEP
490 PRINT:CLOSE#1:GOTO 30
500 BEEP:PRINT"CETTE CARTE N'EST PAS UNE T2G
50 !"
510 GOTO 490
520 REM (c)1995 Patrick GUEULLE
```

Bien entendu, nous en avons également incorporé les fonctionnalités (en mieux !) dans CARTES.EXE.

Rien ne garantit toutefois la compatibilité de ces programmes avec les T2G de plus de 50 unités qui, à l'heure où nous écrivons ces lignes, n'existent pas encore. Une mise à niveau sera peut-être à prévoir à moyen terme.

Ceux de nos lecteurs qui ne possèderaient pas encore de T2G trouveront quelques fichiers .CAR intéressants dans le répertoire IMAGES de la disquette jointe à cet ouvrage.

On peut suivre le processus de comptage sur les deux *images-carte* de la figure 4.8 , qui correspondent respectivement à une T2G de 50 unités neuve, et à une T2G50 épuisée.

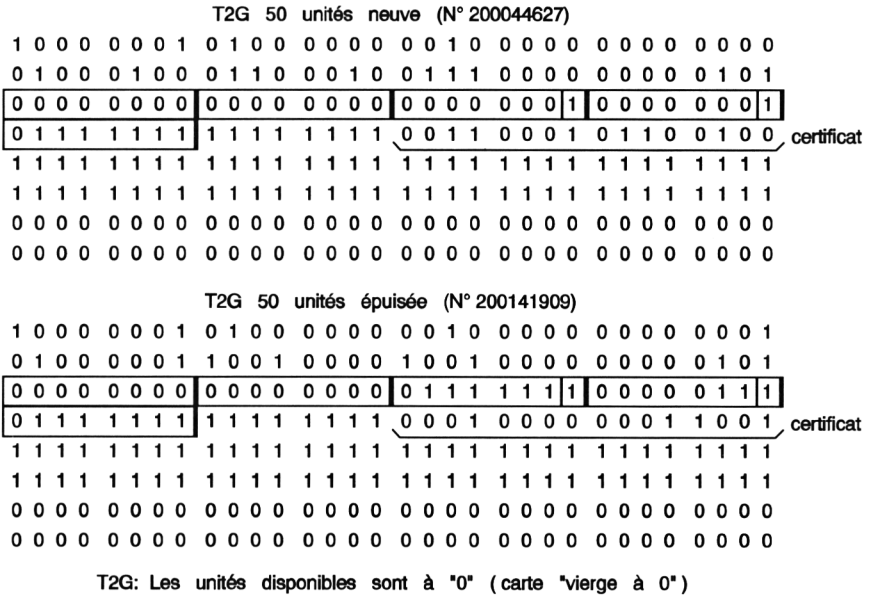


Figure 4.8. Deux «images-carte» de T2G lues avec MINILECT.BAS.

Pour plus de clarté, nous avons repéré les différents compteurs, dont trois demeurent pour l'instant au repos. Gageons qu'ils pourraient servir tôt ou tard (nous préfèrerions tôt...) à instaurer d'une diminution massive du montant de l'unité de taxation, associée à une périodicité de perception accrue en proportion. Il est en effet manifestement injuste de facturer, et ce depuis des temps immémoriaux, le même prix pour une communication locale de trois minutes ou de trois secondes : le perdant est toujours le client, jamais l'exploitant !

Mais nous allons constater que le bon exemple commence à nous arriver de l'étranger.

LES CARTES EUROCHIP

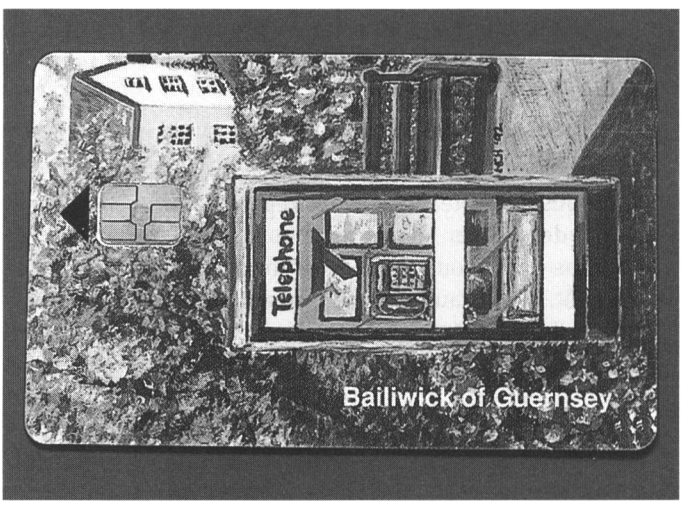
Venue à la carte à puce plusieurs années après la France, l'Allemagne a pu bénéficier d'une technologie plus moderne pour sa TELEFONKARTE.

Déjà munie d'une mémoire EEPROM, sa puce SIEMENS qui possède maintenant de nombreux équivalents, préfigurait déjà notre T2G (mais sans sécurisation cryptographique).

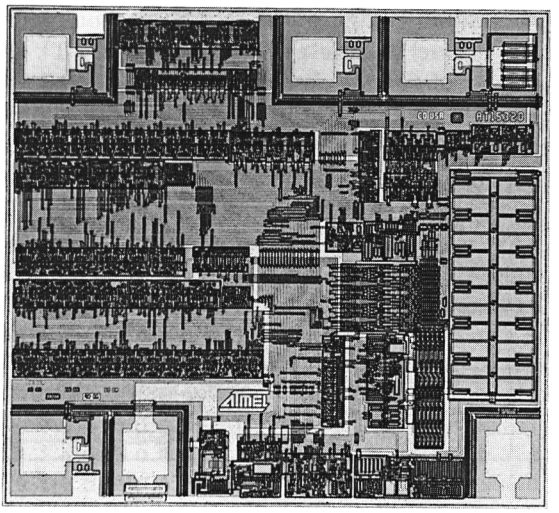
Elle est adoptée, sous l'appellation EUROCHIP, par un nombre croissant de pays européens dont l'île de Guernesey en attendant, bientôt, l'ensemble du Royaume-Uni (British Telecom).

La similitude avec la T2G est évidente au niveau de la cartographie de la figure 4.9 , tout au moins au niveau de ses 104 premiers bits puisque l'EUROCHIP de base n'en contient pas davantage.

Simplement, le compteur de *poids fort* (D5) ne possède que cinq bits, ce qui limite les possibilités de comptage à 20480 unités.



Une carte EUROCHIP à 6 contacts en position ISO.



La puce ATMEL d'une carte EUROCHIP.

Les trois bits ainsi exclus de la zone de comptage sont deux bits de test (b65 et b66), et un bit servant de *verrou* en fin de personnalisation (b64) : en quelque sorte un fusible logiciel.

Le protocole de communication (et notamment de reset) de l'EUROCHIP est par contre sensiblement différent de celui de la T2G et, il faut bien le reconnaître, plus conforme aux normes ISO.

Celles-ci n'en étaient en effet qu'à leurs balbutiements lors du lancement des premières télécartes françaises, avec lesquelles la T2G se doit de maintenir un certain degré de compatibilité matérielle.

	Function	Address	Bits
	Chip Data (CHD)	0 - 23	24
	Issuer Data (ID)	24 - 63	40
	Personalization Complete (PC)	64	1
Count Data	Test Bits (TB)	65 - 66	2
Count Data	Digit 5 (D5)	67 - 71	5
Count Data	Digit 4 (D4)	72 - 79	8
Count Data	Digit 3 (D3)	80 - 87	8
Count Data	Digit 2 (D2)	88 - 95	8
Count Data	Digit 1 (D1)	96 - 103	8
			104

Figure 4.9. Cartographie mémoire des cartes EUROCHIP.

On constate sur la figure 4.10 qu'il s'agit d'un protocole *5 contacts* et que la carte est du type *vierge à 1* : pour y consommer des unités, on y transforme des **1** en 0.

Attention : ce protocole est incompatible avec les modalités de *reset* du mode GPM256 du coupleur INNOVATRON. On ne pourra donc pas lire les EUROCHIP avec COREL 256.BAS, mais il n'y a par contre aucun problème avec MINILECT.BAS ou CARTES.EXE.

ISO2	ISO3	MICRO-INSTRUCTION
1		RESET
0		READ (UP)
0		COMPARE
 suivi de : 0	0 	PROGRAM "0" (ISO7 à GND)

PROTOCOLE "5 CONTACTS" : EUROCHIP

Figure 4.10.
Le jeu de «micro-instructions» des cartes EUROCHIP.

Le principe de comptage, toujours basé sur la technique du *boulier*, fait appel à une pré-programmation des compteurs lors de l'émission de la carte, de telle façon que tous leurs bits (sauf un !) se retrouvent à **0** une fois le crédit de la carte épuisé.

En l'absence (provisoire !) de toute sécurisation cryptographique, c'était encore le moyen le plus sûr pour éviter les rechargements frauduleux...

Les deux *images-carte* de la figure 4.11 se composent toujours des 256 bits que nous avons pris l'habitude de lire, ce qui signifie que le contenu de la carte y figure en double lorsque le compteur d'adresses des EUROCHIP *fait le tour* au bout de 128 impulsions d'horloge. Ce n'est nullement un inconvénient !

On remarquera une mise à contribution de quatre compteurs sur les cinq disponibles, contre deux sur cinq dans le cas de la T2G.

La raison est simple : on compte des unités monétaires (pfennigs en Allemagne, pence à Guernesey, etc.) et non plus des unités téléphoniques.

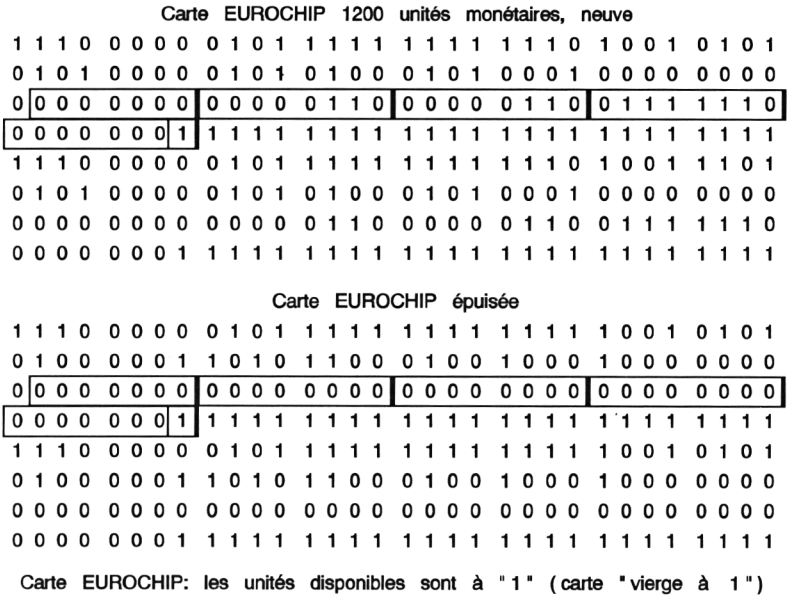


Figure 4.11. Deux «images-carte» d'EUROCHIP lues avec MINILECT.BAS.

Ainsi, une TELEFONKARTE de 40 unités contient en réalité 1200 Pf, c'est à dire 12 DM : la *finesse* de taxation peut donc être rendue incomparablement meilleure, et par conséquent plus juste pour l'usager.

Notre petit logiciel EURO.BAS est capable de déterminer le crédit d'une carte EUROCHIP à partir de son *image* lue sous la forme d'un fichier **.CAR**.

```
10 REM —- EURO.BAS —-
20 KEY OFF:CLS
30 PRINT"NOM DU FICHIER .CAR à ANALYSER  ";
40 INPUT N$
50 IF N$="" THEN END
60 FOR F=1 TO LEN(N$)
70 IF MID$(N$,F,1)="." THEN 100
80 NEXT F
90 N$=N$+".CAR"
100 OPEN N$ FOR INPUT AS #1
110 FOR F=0 TO 66
120 INPUT#1,Q
130 NEXT F
140 U=-1
150 FOR F=67 TO 71
160 INPUT#1,Q
```

```
170 IF Q=1 THEN U=U+4096
180 NEXT F
190 FOR F=72 TO 79
200 INPUT#1,Q
210 IF Q=1 THEN U=U+512
220 NEXT F
230 FOR F=80 TO 87
240 INPUT#1,Q
250 IF Q=1 THEN U=U+64
260 NEXT F
270 FOR F=88 TO 95
280 INPUT#1,Q
290 IF Q=1 THEN U=U+8
300 NEXT F
310 FOR F=96 TO 103
320 INPUT#1,Q
330 IF Q=1 THEN U=U+1
340 NEXT F:PRINT
350 PRINT"CREDIT: ";U;" Unités monétaires"
360 IF U=0 THEN PRINT:PRINT"(CREDIT EPUISE)"
370 IF U<>0 THEN BEEP
380 PRINT:CLOSE#1:GOTO 30
390 REM (c)1995 Patrick GUEULLE
```

CARTES.EXE fait naturellement de même, tout en déterminant automatiquement la nationalité de la carte et l'unité monétaire correspondante (du moins pour l'Allemagne et pour Guernesey).

5 LA DISQUETTE DU LIVRE

La disquette accompagnant cet ouvrage est le fruit de plus de cinq années de patientes recherches menées avec passion autour de ces objets fascinants que sont les cartes à puce.
Cette bibliothèque de logiciels unique en son genre regroupe non seulement les 23 programmes du présent ouvrage, mais aussi les 23 listings GWBASIC de notre livre *Cartes à puce, Initiation et Applications*, E.T.S.F. éditeur.

Nous ne craignons pas d'affirmer qu'une telle somme d'informations et d'outils donne au possesseur de ces deux ouvrages les moyens d'aller, moyennant un peu d'imagination, jusqu'au bout de ce que l'amateur peut raisonnablement espérer apprendre et entreprendre sur les cartes à puce en tous genres.

ORGANISATION DE LA DISQUETTE

Cette disquette a été composée de façon à être utilisable sur un maximum de configurations PC, même si quelques-uns de ses logiciels risquent de fonctionner au ralenti sur les matériels les plus anciens (8088 à 4,77 MHz).

Un formatage spécial permet à cette disquette *trois pouces et demi* d'être éventuellement copiable sur une *cinq pouces un quart* double densité (360 Ko) par un simple DISKCOPY qui en respectera donc intégralement la structure.

L'opération ne réussira pas forcément sur tous les PC à double lecteur, auquel cas on recommencera tout simplement la tentative sur un système plus complaisant.

Tous les programmes du présent ouvrage sont enregistrés dans le répertoire-racine, selon l'ordre alphabétique indiqué sur la figure 5-1.

Une copie de sauvegarde est prévue dans le répertoire LIVRE, accessible par la commande CD LIVRE.

Les répertoires CARPUCE, IMAGES, et PHOTO contiennent respectivement les 23 programmes de notre ouvrage CARTES A PUCE, Initiation et Applications, une sélection de fichiers .CAR extraits de cartes pas forcément faciles à se procurer (cartes étrangères, T2G, etc.), et enfin une microphotographie en couleurs de la puce d'une télécarte, directement affichable sur un écran VGA.

INSTALLATION DE LA DISQUETTE

Bien que tous les programmes BASIC de la disquette soient utilisables immédiatement, une procédure d'installation très simple est nécessaire pour rendre utilisable le logiciel CARTES.EXE.

```
 Volume in drive B is PC_PUCE
 Directory of  B:\

LIVRE          <DIR>        2/01/80    0:03
PHOTO          <DIR>        2/01/80    0:03
IMAGES         <DIR>        2/01/80    0:03
CARPUCE        <DIR>        2/01/80    0:04
CABA       BAS      5303    5/03/95   19:16
CARMAG     BAS       265    5/03/95   18:54
CARS232    BAS       873    5/03/95   18:56
CB2PIN     BAS       706    5/03/95   18:51
COREL      BAS      6294   28/03/95   12:37
COREL256   BAS      4842    5/03/95   18:15
ESPION     BAS       778    5/03/95   18:57
EURO       BAS       831    5/03/95   19:07
MINILECT   BAS      1262    8/04/95    9:22
MODULO     BAS       284   26/03/95   11:16
PIN2CB     BAS       695    5/03/95   18:50
PROTECT    BAS      1054   27/03/95    9:41
PROTEGE    BAS      1222   27/03/95    9:37
RSA        BAS       422   26/03/95   17:24
SIMU       BAS      1044    5/03/95   18:57
T1G        BAS      3328    5/03/95   18:43
T2G        BAS      1087   10/03/95   17:02
TEXAS      BAS       905    8/04/95    9:35
TRANCB     BAS       898    5/03/95   19:17
XOR        BAS       226   26/03/95   11:08
PICPUCE    ASM      5565   24/03/95    8:11
PICPUCE    OBJ       960   24/03/95   11:13
CARTES     EXE     35117    6/04/95   12:14
INSTALL    EXE     30733    5/04/95   16:28
                      38    6/04/95   18:04
       29 File(s)              0 bytes free
```

Figure 5.1.
Les 23 logiciels du présent ouvrage.

Cette manoeuvre nécessite la réalisation préalable d'un lecteur de télécartes (par exemple celui décrit au chapitre 4), pour *lier* la disquette à une télécarte *clef* librement choisie dans votre collection personnelle.

Nous vous demandons de considérer cette installation comme une démarche volontaire, revêtant une signification juridique précise: l'engagement formel de n'en faire des copies que pour votre usage personnel.

Toute reproduction de la disquette originale non accompagnée de la télécarte-clef ayant servi à son installation sera donc réputée être une copie illégale, exposant son détenteur aux poursuites prévues par la loi.

L'installation doit être exécutée de la façon suivante:

Un lecteur de télécartes approprié étant connecté à un port parallèle du PC (LPT1: ou LPT2:), exécuter l'utilitaire INSTALL.EXE en frappant INSTALL à partir de la disquette originale NON PROTEGEE EN ECRITURE.

Introduire une télécarte quelconque dans le lecteur dès que le logiciel le demandera, et presser ENTER.

Au bout de quelques secondes, un message de copyright doit apparaître sur l'écran. Sachez que sa réplique est maintenant inscrite sur la disquette, que vous pouvez à nouveau protéger en écriture et dont vous pouvez effectuer les copies de sauvegarde que vous jugez nécessaires pour votre usage personnel.

Essayez maintenant de lancer CARTES.EXE en frappant CARTES, et introduisez votre télécarte-clef à la demande du logiciel.

En cas de problème (message CARTE-CLEF NON RECONNUE ou autre message d'erreur), vérifiez soigneusement le lecteur en exécutant, sous GWBASIC, MINILECT.BAS (ou GPM256.BAS à partir du répertoire CARPUCE).

Si le problème ne concerne que CARTES.EXE, recommencez l'installation avec une autre télécarte.

Une fois l'installation réussie, il est souhaitable de copier tout le répertoire-racine de la disquette (et éventuellement les autres) sur un éventuel disque dur, ou au moins sur une disquette de travail.

Attention: ne tentez pas de copier CARTES.EXE seul, il ne fonctionnerait pas.

MODE D'EMPLOI DE CARTES.EXE

Ce puissant logiciel concrétise toute l'expérience qu'il nous a été possible de réunir, au moment où nous écrivions ces lignes, en matière de lecture de télécartes françaises et étrangères. Il n'a donc rien de définitif !

Il doit être associé à un lecteur de cartes synchrones tel que celui décrit au chapitre 4, et installé selon la procédure qui vient d'être décrite. Bien entendu, la télécarte-clef devra être présentée lors de chacun de ses lancements (on la rangera donc de préférence avec le livre).

Surtout destiné aux collectionneurs et aux curieux de tout poil, ce programme n'est pas prévu pour écrire dans les télécartes: ce genre d'opération nécessite l'usage d'un lec-

teur plus complexe et d'un logiciel tel que GPM256.BAS (répertoire CARPUCE) ou COREL256.BAS.

Dans son état actuel, CARTES.EXE supporte les familles de cartes synchrones suivantes:

- cartes EPROM 256 bits GPM256: télécartes françaises T1G, télécartes de nombreux pays étrangers, cartes diverses (lavage de voitures, stationnement, cinéma, etc.)

- T2G françaises de 50 unités;

- cartes EUROCHIP (Allemagne, Guernesey, etc.).

Chaque fois que les informations en notre possession l'ont permis, nous avons prévu l'identification automatique du pays et la lecture du nombre d'unités disponibles.

Lorsque cela n'a pas été possible (les cartes d'Espagne et de Croatie, notamment, gardent pour l'instant leurs secrets...) nous avons pris le parti d'afficher ce qui pouvait tout de même l'être.

En tout état de cause, l'option Affichage binaire et hexa permet d'examiner une *image-carte* brute, non décodée, dont chacun pourra tenter de percer les éventuels mystères.

Très utiles, les options *Sauvegarde sur disque* et *Chargement du disque* permettent d'enregistrer sur disque dur ou sur disquette n'importe quelle image-carte pour la relire et la décrypter de nouveau ultérieurement, même si on ne dispose plus de la carte originale.

Si une imprimante est connectée en même temps que le lecteur (sur un autre port parallèle), la fonction *sauvegarde* permet d'imprimer une image-carte binaire en spécifiant PRN en tant que nom de fichier (rien ne sera alors enregistré sur le disque).

Le cas échéant, les images-carte sauvegardées sur disque peuvent être modifiées à l'aide d'un éditeur de texte puis rechargées: cela permet notamment de simuler des consommations d'unités ou de reconstituer des images de cartes neuves à partir de cartes épuisées, pour la plus grande satisfaction des collectionneurs.

Les options *Contrôle de parité* (T1G françaises seulement) et *Vérification d'une carte* sont utiles pour détecter les cartes

dont le contenu a changé par rapport à leur image précédente: consommation d'unités, bien sûr, mais éventuellement altération du contenu de leur mémoire.

On chargera évidemment, depuis le disque, l'image de référence avant d'appeler la fonction de vérification et d'introduire la carte à vérifier.

UNE SÉLECTION D'IMAGES-CARTE

Nos lecteurs ne disposant pas nécessairement de tous les specimens de cartes nécessaires à leurs manipulations, le répertoire IMAGES contient un choix de 13 images-carte dont la figure 5-2 donne la liste.

On y accèdera par la commande : CD IMAGES.

Le chargement de ces fichiers .CAR dans CARTES.EXE se traduira par un décryptage immédiat de leur contenu, après lequel il sera bon d'examiner un affichage binaire pour chercher à comprendre la façon dont sont comptabilisées les unités.

ALM1.CAR à ALM4.CAR proviennent de quatre télécartes allemandes (EUROCHIP) neuves, vides, et entamées, GUERN1.CAR et GUERN2.CAR de deux cartes EUROCHIP de Guernesey (une vide et une entamée), tandis que MALTE.CAR et IRLANDE.CAR correspondent à deux cartes de ces pays respectifs (même technologie que les T1G françaises, mais avec un codage différent).

```
 Volume in drive B is PC_PUCE
 Directory of  B:\IMAGES

.            <DIR>       2/01/80   0:03
..           <DIR>       2/01/80   0:03
ALM1     CAR        592  24/11/94  17:43
ALM2     CAR        592  24/11/94  17:44
ALM3     CAR        592  28/03/95  13:15
ALM4     CAR        592  28/03/95  13:15
GUERN1   CAR        592  16/08/94  12:03
GUERN2   CAR        592  16/08/94  12:03
HOLA     CAR        592  10/03/95  18:07
IRLANDE  CAR        592  10/03/95  18:06
MALTE    CAR        592  30/08/94  11:09
MOBIL    CAR        592  28/03/95  13:13
PASVIDE  CAR        592  23/10/94  15:59
T2G5     CAR        592  28/03/95  13:13
T2GA     CAR        592  10/03/95  18:05
       15 File(s)          0 bytes free
```

Figure 5.2.
Une sélection de 13 «images-carte».

HOLA.CAR et PASVIDE.CAR ont été lus dans deux télécartes françaises dotées d'un *code famille* différent : l'habituel 03h pour PASVIDE.CAR, et 04h pour HOLA.CAR. Ce nouveau code semble avoir été mis en service pour pallier à l'épuisement imminent des numéros de cartes encore inutilisés, et va donc fort probablement se généraliser jusqu'à l'avènement définitif des T2G.

T2G5.CAR et T2GA.CAR proviennent précisément de deux télécartes de seconde génération ayant participé à l'expérimentation *terrain* de 1993.

L'une est neuve, tandis que l'autre contient encore 5 unités sur les 50 dont elle avait été chargée en usine.

En principe, des T2G de 120 unités et peut-être davantage devraient faire leur apparition sous peu. Bien qu'il ne nous ait pas encore été donné d'en introduire dans nos lecteurs, nous avons d'ores et déjà prévu leur lecture sous réserve toutefois de surprises toujours possibles...

Enfin, MOBIL.CAR est l'image d'une carte de lavage de voitures telle qu'on en utilise dans les stations service de ce pétrolier, et qui existe en 12 et 24 unités (autrefois 10 et 20). La technologie utilisée (SOLAIC) est celle des T1G.

```
ALM1.CAR:
0000 0010 0011 1111 1111 1110 1010 1010
1000 1011 0010 0101 0011 0011 0001 1000
0000 0000 0000 0000 0000 0000 0000 0100
0000 1111 1111 1111 1111 1111 1111 1111
0000 0010 0011 1111 1111 1110 1010 1010
1000 1011 0010 0101 0011 0011 0001 1000
0000 0000 0000 0000 0000 0000 0000 0100
0000 1111 1111 1111 1111 1111 1111 1111
ALM2.CAR:
1101 0000 1011 1111 1111 1110 0010 1010
1001 0011 1010 0010 0101 1001 1010 0000
0000 0000 0000 0000 0011 1100 0000 0100
1111 1111 1111 1111 1111 1111 1111 1111
1101 0000 1011 1111 1111 1110 0010 1010
1001 0011 1010 0010 0101 1001 1010 0000
0000 0000 0000 0000 0011 1100 0000 0100
1111 1111 1111 1111 1111 1111 1111 1111
```

```
ALM3.CAR:
1110 0000 0101 1111 1111 1110 1001 0101
0101 0000 0101 0100 0101 0001 0000 0000
0000 0000 0000 0110 0000 0110 0111 1110
0000 0001 1111 1111 1111 1111 1111 1111
1110 0000 0101 1111 1111 1110 1001 0101
0101 0000 0101 0100 0101 0001 0000 0000
0000 0000 0000 0110 0000 0110 0111 1110
0000 0001 1111 1111 1111 1111 1111 1111
ALM4.CAR
1110 0000 0101 1111 1111 1111 1001 0101
0101 0001 1000 1001 1101 1001 1000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0001 1111 1111 1111 1111 1111 1111
1110 0000 0101 1111 1111 1111 1001 0101
0101 0001 1000 1001 1101 1001 1000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0001 1111 1111 1111 1111 1111 1111
GUERN1.CAR:
0010 1000 0111 0111 1111 1110 0000 1100
1110 0000 0000 0000 0000 0011 1011 1100
0000 0000 0000 0000 0000 0000 0001 1110
0001 1111 1111 1111 1111 1111 1111 1111
0010 1000 0111 0111 1111 1110 0000 1100
1110 0000 0000 0000 0000 0011 1011 1100
0000 0000 0000 0000 0000 0000 0001 1110
0001 1111 1111 1111 1111 1111 1111 1111
GUERN2.CAR:
0010 1000 0111 0111 1111 1110 0000 1100
1110 0010 0000 0011 0101 1000 1100 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0001 1111 1111 1111 1111 1111 1111
0010 1000 0111 0111 1111 1110 0000 1100
1110 0010 0000 0011 0101 1000 1100 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0001 1111 1111 1111 1111 1111 1111
HOLA.CAR:
1101 0011 0000 0100 0000 0000 0001 0110
1100 1111 0010 0000 0110 1000 0001 0000
1100 0111 1000 1011 0001 0000 0000 0110
1111 1111 1111 1111 1111 1111 1111 1111
1111 1111 1111 1111 1111 1111 1111 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 1111 1111
```

```
IRLANDE.CAR:
1011 1010 1000 0011 0001 0001 0000 0010
0100 0000 0000 0001 1100 1101 0100 1010
0001 1001 1010 0111 0001 0001 0011 1100
1110 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
MALTE.CAR:
1011 1101 1000 0011 0001 0000 0110 0010
0000 0000 0001 0101 1010 0100 0101 1010
1000 1000 1001 0100 0001 0001 0101 0100
1111 1111 1111 1111 1111 1111 1111 1111
1111 1111 1111 1111 1111 1111 1111 1100
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
MOBIL.CAR:
1000 1000 1000 0000 0010 0000 0000 0010
0011 1100 0111 0101 1000 0010 0010 0100
1010 0001 0000 0000 0000 0000 0000 0001
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
PASVIDE.CAR:
1100 1011 0000 0011 1001 0000 0100 0001
1011 1011 0101 0110 0110 0111 0000 0010
1100 0011 0100 1111 0001 0000 0000 0110
1111 1111 1111 1111 1111 1111 1111 1111
1111 1111 1111 1111 1111 1111 1110 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
T2G5.CAR:
1000 0001 0100 0000 0010 0000 0000 0000
1000 0010 0100 0101 0110 0000 0000 0101
0000 0000 0000 0000 0011 1111 0011 1111
0111 1111 1111 1111 0101 0011 1000 0010
1111 1111 1111 1111 1111 1111 1111 1111
1111 1111 1111 1111 1111 1111 1111 1111
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

```
T2GA.CAR:
1000 0001 0100 0000 0010 0000 0000 0001
0100 0010 0101 0010 1000 0000 0000 0101
0000 0000 0000 0000 0000 0001 0000 0001
0111 1111 1111 1111 0111 1000 1011 1010
1111 1111 1111 1111 1111 1111 1111 1111
1111 1111 1111 1111 1111 1111 1111 1111
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

ENCORE 23 PROGRAMMES BASIC !

Accessible par la commande CD CARPUCE, le répertoire CARPUCE, dont la figure 5-3 récapitule le contenu, réunit la totalité des logiciels publiés dans notre ouvrage CARTES A PUCE, Initiation et Applications.

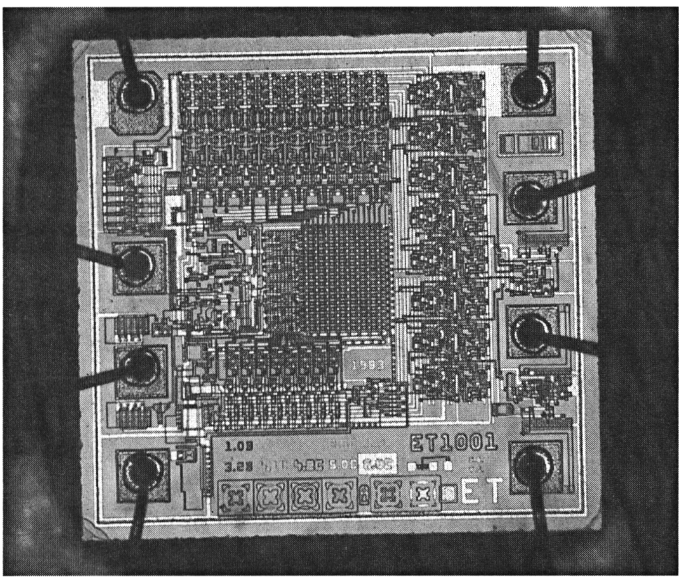
```
 Volume in drive B is PC_PUCE
 Directory of  B:\CARPUCE

.            <DIR>       2/01/80   0:04
..           <DIR>       2/01/80   0:04
BINTOROM BAS      394    1/01/80   0:07
CARTOROM BAS      447    1/01/80   0:46
COPYIIC  BAS     1026    1/01/80   0:18
CREDIT12 BAS      637   18/01/93  14:06
CREDIT50 BAS      638   18/01/93  14:06
GPM256   BAS     3936   29/12/92  12:09
KARTE    BAS      409   24/12/92  14:03
LECAIIC  BAS      321    1/01/80   0:36
LECI2C   BAS     1459    1/01/80   0:17
LECIIC   BAS      316    1/01/80   0:14
LRC      BAS      378    1/01/80   0:26
NOVCHECK BAS     1144    1/01/80   0:05
NOVERASE BAS     1217    1/01/80   0:08
NOVWRITE BAS     1326    1/01/80   0:25
PROGI2C  BAS     1494    1/01/80   0:23
PROGIIC  BAS      356    1/01/80   0:16
RESET    BAS      610    1/01/80   0:05
RESETD   BAS      358   10/12/93  14:43
ROMTOBIN BAS      426    1/01/80   0:03
ROMTOCAR BAS      817    1/01/80   0:09
SERRURE  BAS     1125   19/01/93  14:45
TESTCOUP BAS      770    1/01/80   0:24
VERI2C   BAS     1578    1/01/80   0:38
       25 File(s)          0 bytes free
```

Figure 5.3.
Les 23 logiciels de «CARTES A PUCE, Initiation et Applications».

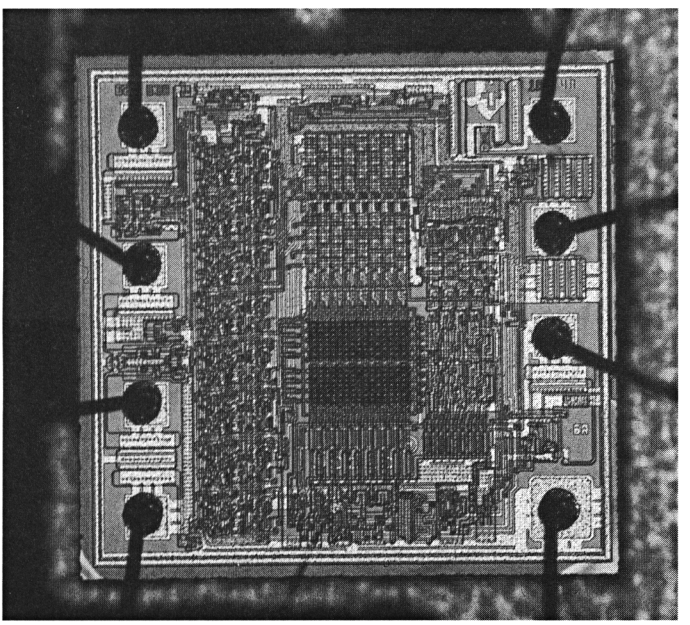
Ceux de nos lecteurs qui ne le possèderaient pas encore pourront ainsi se le procurer sans craindre d'avoir à saisir manuellement ses listings, avec tous les risques d'erreurs que cela suppose.

Ils y découvriront toutes les notions de base de vocabulaire, normalisation, et mise en oeuvre des cartes à puce sur les-

quelles il n'était évidemment pas question de s'attarder à nouveau ici, et également tout un choix de montages d'application originaux: serrure de sûreté à télécarte, testeur de poche pour télécartes, et lecteur-enregistreur de télécartes.



La puce de télécarte ET1001.



La puce Texas Instruments pour télécartes.

UNE PHOTO AUTO-EXECUTABLE

Accessible par la commande CD PHOTO, le répertoire PHOTO de la figure 5-4 contient un seul et unique fichier nommé PUCE.EXE.

Il s'agit d'une véritable photographie d'une puce de télécarte (modèle ET1001 de 1983) digitalisée, mise en forme, et transformée en un fichier auto-exécutable sous DOS.

Pour peu que le PC utilisé dispose d'un écran VGA couleur, il suffit donc de taper PUCE pour déclencher l'affichage de cette image étonnante (dimensions réelles 1 mm x 1 mm !)

```
 Volume in drive B is PC_PUCE
 Directory of  B:\PHOTO

.            <DIR>       2/01/80   0:03
..           <DIR>       2/01/80   0:03
PUCE     EXE    71023    4/12/94  23:07
        3 File(s)          0 bytes free
```

Figure 5.4.
Ce fichier exécutable affichera une étonnante image sur votre écran VGA.

Imprimerie Arts Graphiques du Perche 28240 Meaucé
Dépôt légal : janvier 1997 — N° d'Imprimeur 97359
Dépôt légal de la 1re édition : 3e trimestre 1995
Imprimé en France

PC
et Cartes à Puce

Même si vous savez déjà lire et écrire dans les cartes à puce, ce livre va vous expliquer comment aller beaucoup plus loin !

Avec l'aide de votre PC, vous allez par exemple pouvoir lire le "relevé de compte" de votre carte bancaire ou savoir si on l'a utilisée à votre insu, et même décrypter les échanges de données entre les cartes et leurs lecteurs.

Vous apprendrez également à fabriquer vos propres cartes et à les programmer depuis le PC.

Et lorsque vous aurez réalisé les montages décrits et utilisé les logiciels réunis sur la disquette, les nouvelles télécartes françaises ou étrangères n'auront plus guère de secrets pour vous, que vous soyez ou non collectionneur.

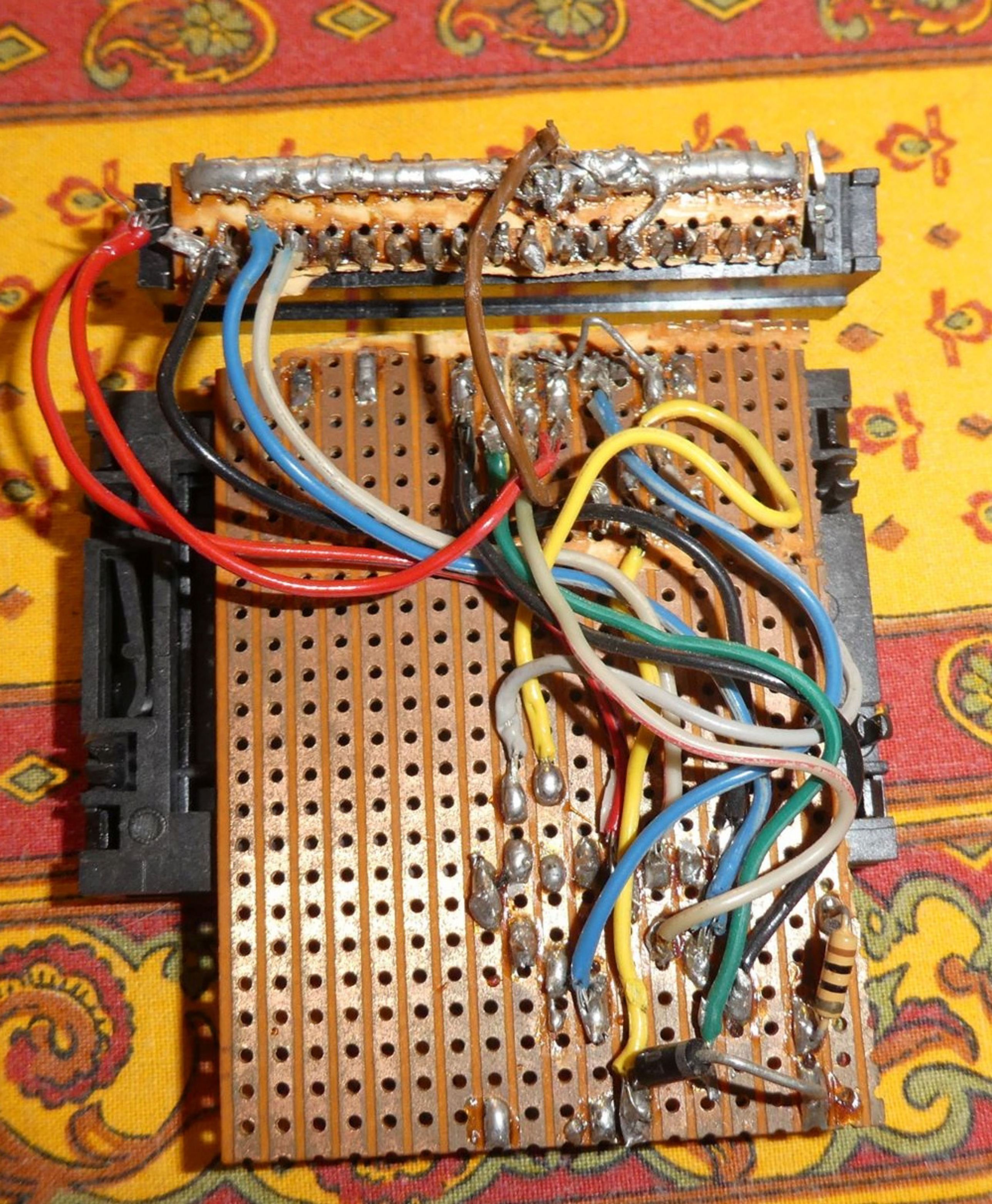
EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES

9 782855 352398 Code 023963

ISBN 2-85535-239-8

Patrick GUEULLE

ETSF



Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE

https://acpc.me/