GENIE ELECTRIQUE ET
INFORMATIQUE INDUSTRIELLE

PROJET : MODULE GPS

PROJET 2006/2007

Enseignants : P.Bouron
A.Chaussat

Etudiants: Umit Demir
Florian Abry

UNIVERSITÉ CLAUDE BERNARD - LYON 1

SOMMAIRE

Introduction

Lors de la seconde année d'IUT GEII (Génie Electrique et Informatique Industrielle), les élèves doivent réaliser un projet sur l'année, en abordant aussi bien sa conception que sa réalisation. Etant actuellement en seconde année de ce DUT, il nous a donc fallu choisir un projet. Parmi ceux proposés, celui ayant retenu notre attention fut le développement d'un module GPS détecteur de radars fixes. Ce module doit informer le conducteur de la voiture de la proximité d'un radar fixe par un jeu de leds. Il doit aussi faire état du nombre de satellites renseignant le GPS sur notre position.

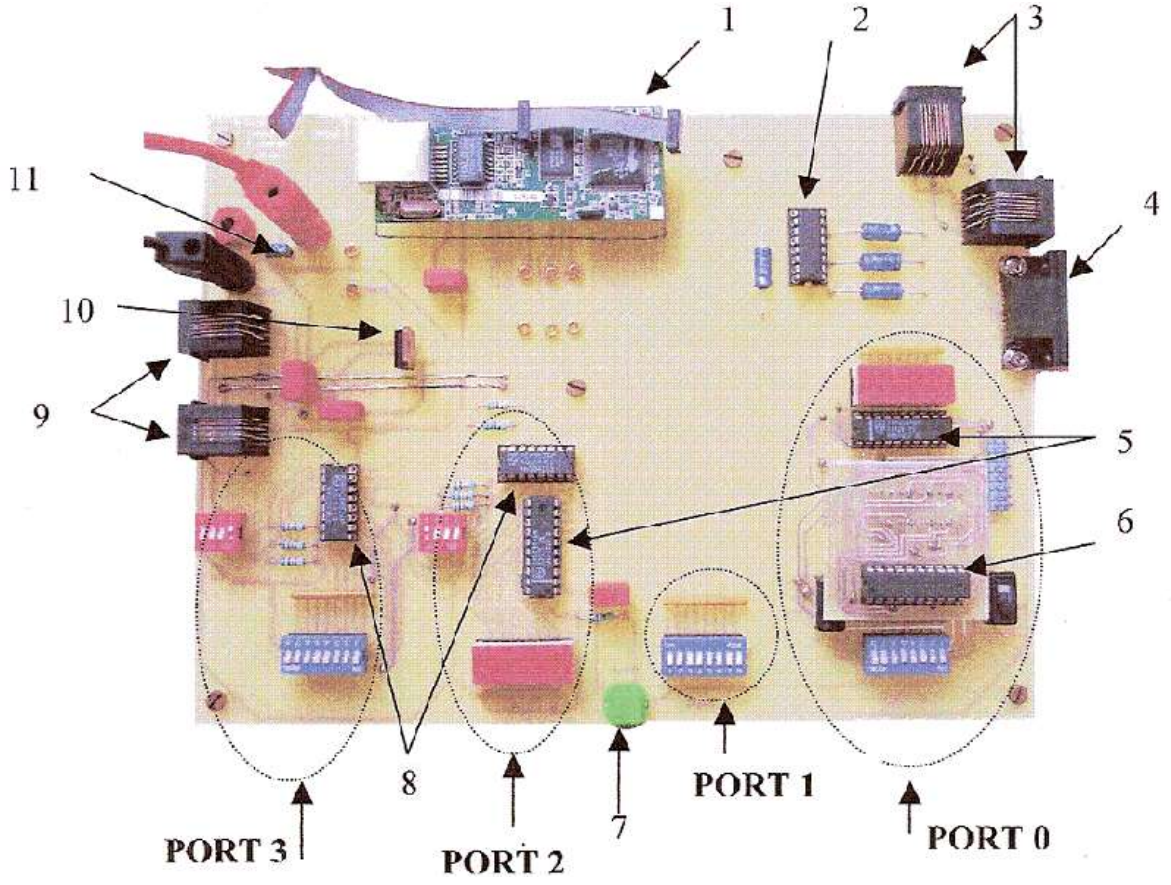
Tout d'abord, ce projet s'articule autour d'un microprocesseur de type Rabbit RMC 3700 qui se programme en C. Ce langage étant à la base de maintes applications, il peut apparaître comme important de le connaître et ce projet permets de l'appliquer de manière concrète à un système.
De plus, la technologie GPS à dernièrement perdu de son côté exotique. Plusieurs voitures en sont équipées de série. Or, elle n'est que peu étudiée. Ce projet offre donc la possibilité d'appréhender les bases de son fonctionnement et de son interface.
Enfin, outre ces spécificités, ce permets nous permet de valider nos connaissances, notamment dans le domaine de l'électronique (via le développement et le routage de la carte) et de l'informatique industrielle (via la gestion du microprocesseur, l'envoi de données par le port série et l'étude du bus I2C).

Ce compte rendu se veut donc l'état des lieux de ce projet à moins de 4 mois de la date butoir fixée pour le rendre. Il présente, dans un premier temps, le travail d'utilisation qu'il nous a été demandé de faire autour d'une carte pédagogique réalisée par les élèves de l'an dernier, dont le but était de faciliter la compréhension du microprocesseur de type Rabbit. Ensuite, il fait état de l'avancement de la conception du module et de la programmation du microprocesseur.

Carte électronique pédagogique 2006

A l'aide du compte rendu de l'année précédente nous pouvons décrire les fonctions de cette carte.

A) Détails de la carte



Légendes :

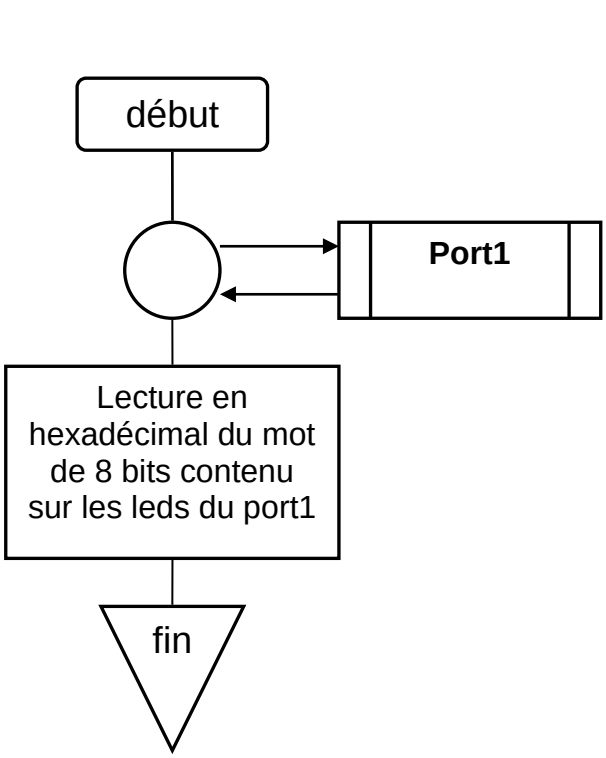
1 : Module RCM 3700
2 : MAX 232
3 : Connecteurs RJ45 (Ethernet)
5 : 74HCT540N (buffer inverseur)
4 : Prise liaison série femelle DB9
6 : 74HCT244 (buffer)
7 : Bouton poussoir (reset)
8 : PCF 8574
9 : Connecteur RJ11 (I2C)
10 : Régulateur 7805
11 : Cavalier (choix alimentation)

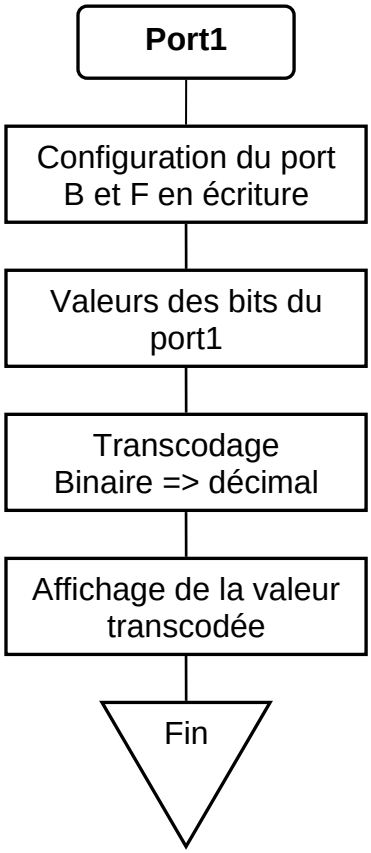
La description de certain composant sera reprise dans la deuxième partie du dossier, où nous donnerons quelques renseignements sur le bus I2C, le MAX 232, régulateur 7805 et il y aura aussi la documentation constructeur pour plus de renseignements.

B) Fonctionnement de la carte

Après avoir pris connaissance de la réalité physique de la carte, nous nous sommes donc attelés à la programmation afin d'essayer de comprendre les différences induites par l'utilisation du C dynamique appliquées à un microprocesseur de type Rabbit. Pour ce faire, nous avons, dans un premier temps, cherché à savoir comment lire l'état d'un port et comment rendre traitables les données ainsi récupérée, avant de chercher à les écrire sur un autre port et, par là même, imposer un état sur ce dernier.

1) Programme de lecture du Port 1 de la carte pédagogique.





Pour le premier programme, le cahier des charges que nous nous sommes fixé était des plus simple : il s'agissait d'afficher à l'écran du PC la valeur en hexadécimal du mot de huit bits écrit sur le Port 1 de la carte didactique réalisée l'an dernier par l'intermédiaire d'un jeu de commutateurs à deux positions (imposant l'état de chacune des pistes du Port 1 au zéro logique ou au un logique). Ce programme se trouve en Annexe 1. Chaque fois qu'il sera fait référence à une partie lambda de ce programme, cette partie sera isolée du programme en annexe (ceci est vrai pour tous les programmes présents dans ce compte rendu de projet). Les initialisations n'auront, quant à elles, pas de parties propres et seront traitées au fur et à mesure que le besoin de déclarer une nouvelle variable se fera sentir. Regardons maintenant sur le contenu de ce programme :

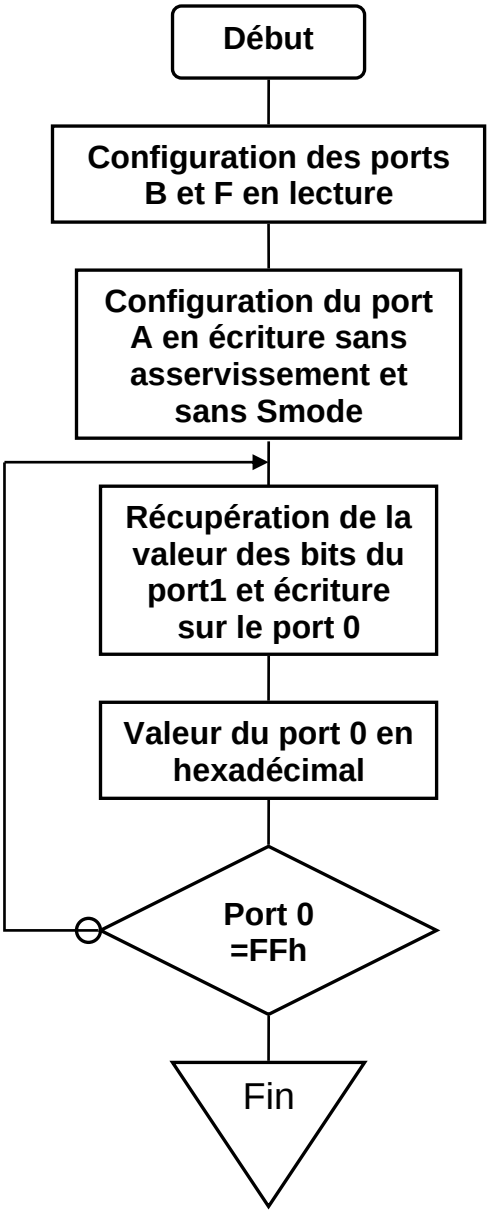
Partie A : cette partie est en fait le programme principal. Il s'occupe de l'interface avec le PC, en stockant dans une variable nommée *A* (la valeur retournée par le sous programme *port1()*) et en l'affichant à l'écran du PC à l'aide de la fonction *printf()*. *Port1()* retournant un caractère, *A* devra être déclaré comme tel. La valeur affichée devra être en hexadécimal. Déclarer l'affichage sous forme d'un mot hexadécimal dans un *printf()* se fait avec *%x*. C'est sur cet affichage que se termine la première partie du programme.

Partie B : il s'agit du sous programme *Port1()* visant à récupérer la valeur du mot de huit bits écrit sur le port 1. Avant d'entrer dans les détails du programme, il convient de préciser que ce qui est appelé Port 1 sur la platine didactique est en fait un mixe de plusieurs broches des ports B et F du Rabbit. Ce sous programme est déclaré comme : *char Port1(void)*. Le *char*, placé avant Port1, signifie que la valeur retournée par ce sous programme sera un caractère et le *void* qu'il ne nécessite aucune valeur venant du programme principal pour s'exécuter. *BF*, la variable qui va contenir la valeur retournée par le sous programme est donc déclarée comme un caractère, et ce, dés la première ligne du sous programme.

Ensuite, nous configurons les ports B et F du microprocesseur en lecture, à l'aide de l'instruction *WrPortI()*. Dans cette instruction, *PBDDR* et *PFDDR* configurent respectivement l'état du port B et du port F. *PBDDR* et *PFDDR* doivent être configurés à 0 pour que le port fonctionne en lecture et à 1 pour qu'il fonctionne en écriture. Nous le configurons donc à 0. Puis nous récupérons les valeurs contenues sur les ports du Rabbit composant le Port 1 grâce à l'instruction *BitRdPortI()* dans laquelle nous définissons la broche et le port lus. La valeur retournée est le complément de l'état du cette broche. Ce résultat est complémenté pour avoir la vraie valeur transmise sur cette broche. Une fois ce résultat obtenu, on le pondère de manière à avoir le la valeur décimale de ce mot de 8 bits, puis nous le mettons dan *BF* comme caractère. Enfin, le contenu de *BF* est déclaré comme valeur retournée par le sous-programme.

Bien que peu lourd, le développement de ce programme fut assez laborieux, tout particulièrement à cause de l'utilisation du C dynamique. En effet, le compte rendu de l'année dernière développait assez peu l'aspect programmation. Il se contentait de laisser en annexe des programmes non commentés qui ne fonctionnaient pas. Ces programmes ont néanmoins servis de bases à nos recherches concernant la configuration et la récupération d'informations sur les ports. Les données de leurs programmes croisées avec l'explication du jeu d'instructions propre au C dynamique nous ont permis, au final, de faire fonctionner ce programme.

2) Programme d'écriture sur le Port 0



Dans la continuité du premier programme, après avoir compris comment lire les états logiques des broches de la carte, il nous a maintenant fallu savoir comment faire pour les imposer. Ainsi, le cahier des charges que nous nous sommes fixés était de réaliser un programme transmettant l'état du port 1 de la carte au jeu de huit leds du port 0. Cette valeur devait être affichée sur l'écran du PC à chaque fois qu'elle différait de la précédente et stopper le programme lorsque cette valeur était *FFh*. Le code indexé de la même manière que précédemment du programme se trouve en annexe 2.

En ce qui concerne les initialisations (Partie A du programme), les ports B et F du microprocesseur (qui forment le port 1 de la platine) sont configurés en lecture comme précédemment. La particularité vient de la configuration du port A qui compose le port 0 de la platine. Ce dernier, outre la possibilité d'être configuré en Lecture / Ecriture, peut être asservi et sert à configurer le SMODE. Il faut donc, dans un premier temps, configurer le *SPCR* qui contrôle ces deux derniers points de la manière suivante :

Bit(s)	Value	Description
7	0	Obey SMODE pins (SMODE0 & SMODE1). This puts the slave into coldboot mode.
	1	Ignore SMODE pins.
6:5	Read	Reports the state of SMODE pins.
	Write	These bits are ignored.
4	x	This bit is ignored.
3:2 write-only	00	Disables the slave port. Parallel port A is a byte-wide input.
	01	Disables the slave port. Parallel port A is a byte-wide output.
	10	Enables the slave port, disabling parallel port A and various other port lines.
	11	Enable the auxilliary I/O bus (Rabbit 3000 only). Parallel port A is used for the data bus and parallel port B (PB7:PB2) is used for the address bus.
1:0	00	Disable slave interrupts.
	01	Enable slave interrupts and set to priority level 1.
	10	Enable slave interrupts and set to priority level 2.
	11	Enable slave interrupts and set to priority level 3.

Dans notre cas, ne voulant ni utiliser le SMODE, ni asservir le port, nous lui avons transmis la valeur *F4h* (soit *11110100b*) toujours pas l'intermédiaire de l'instruction *WrPortI()*. Il est à noter que pour lui transmettre cette valeur, nous avons écrit *0xF4*. Pour que le programme reconnaisse un nombre, ce dernier doit obligatoirement commencer pas un chiffre (*x* étant un caractère, il doit être précédé par un 0). Le *x* quand à lui signifie que le nombre qui va suivre sera en hexadécimal. Une fois cette initialisation faite, on peut configurer le port A en écriture en mettant *PADR* à un. Enfin, on met la valeur 0 dans un variable nommée *A*. Celle ci servira à tester si la valeur à afficher est la même que la valeur précédente. Elle sera amenée à ne manipuler que des entiers. Il faut donc la déclarer comme *integer*

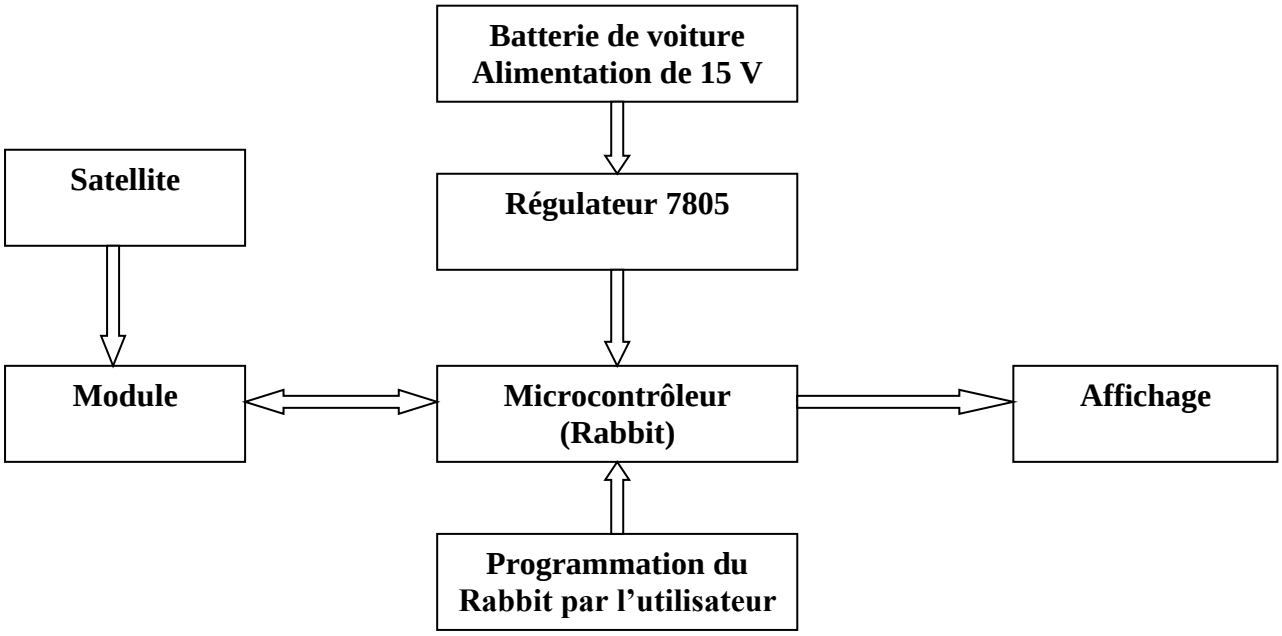
Vient ensuite la boucle à l'intérieur de laquelle l'état des bits du port 1 est transmis à ceux du port 0 (partie B du programme). Le port 1 étant un patchwork des ports B et F du Rabbit, nous avons opté pour une transmission bit à bit, même si d'autres solutions étaient possibles. L'instruction qui nous a servi est *BitWrPortI()*. Elle s'utilise de la manière suivante : *BitWrPortI(*nom_du_port, &nom_du_port*Shadow*, valeur_à_transmettre, numéro_de_la_broche_où_transmettre*)*. La valeur que nous voulions transmettre étant celle des bits du port 1, nous avons donc utilisé *BitRdPortI()* pour déterminer cette valeur. L'opération est répétée pour tous les bits composant le port 0.

Pour finir (partie C), il nous faut afficher la valeur transmise à l'écran si elle diffère de la précédente et recommencer tant que la valeur transmise n'est pas *FFh*. Dans un premier temps, grâce à *RdPortI(PADR),* on transmet la valeur à une variable nommée *A1* (déclarée en *integer*). Cette valeur est comparée avec celle de *A*. Si leurs valeurs différent, la valeur contenue dans *A1* est affichée. Suite à cela, la valeur de *A1* est sauvegardée dans *A* et testée. Si la valeur de *A* est égale à *FFh*. Si c'est le cas, nous sortons de la boucle, ce qui a pour effet de finir le programme. Sinon, on « re-boucle » sur le *do{}* pour recommencer la même opération.

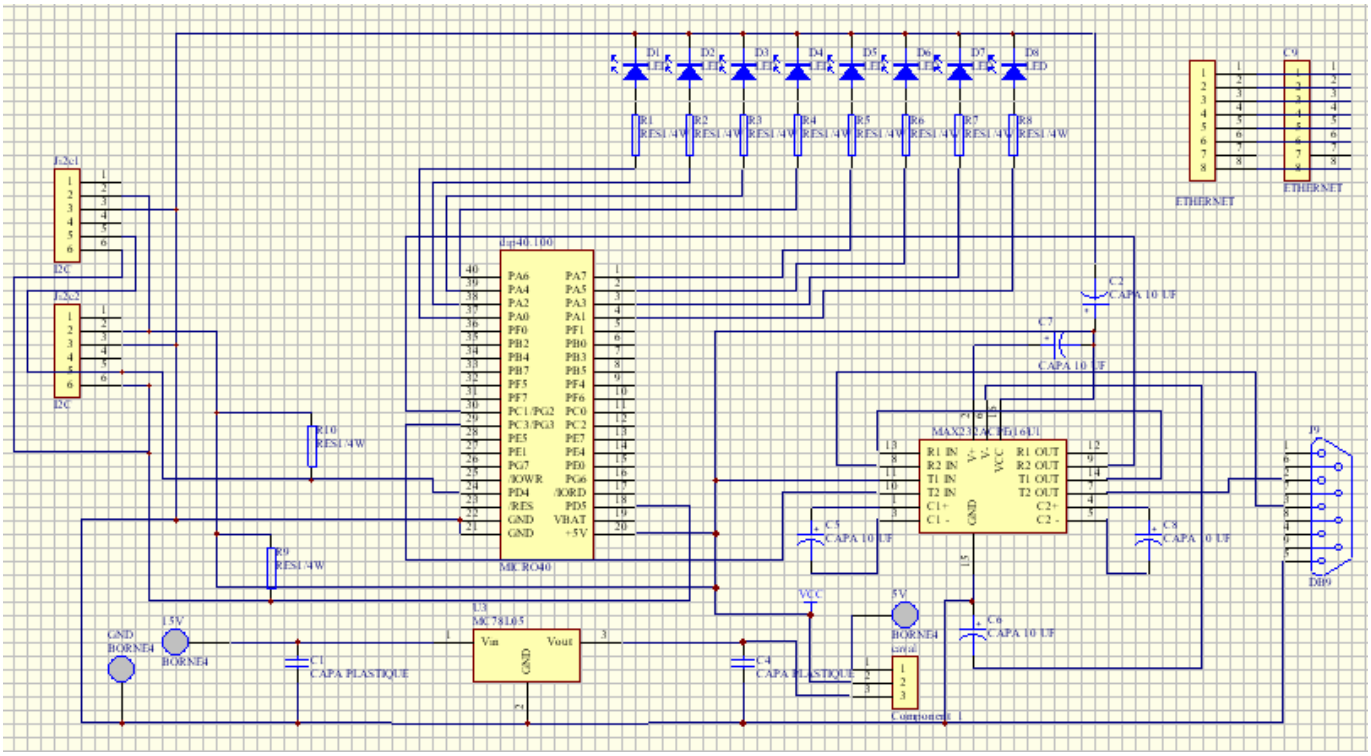
Suite à ces deux programmes nous avons décidés que les connaissances que nous avions emmagasinées sur le C dynamique étaient suffisantes pour nous attaquer au projet à proprement parler.

Présentation du projet

Synoptique général :



A) Fabrication et étude de la carte



Alimentation de la carte : LE REGULATEUR

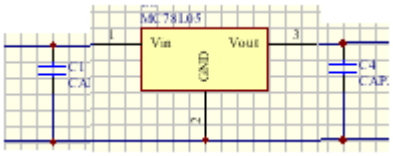
Comme nous allons alimenter notre carte par une batterie de voiture qui délivre environ 15V, nous devons effectuer une étude sur la dissipation thermique du régulateur qui doit délivrer 5V.

Donnée constructeur :

Température de jonction :125°C
Température de l'air :25°C
Résistance jonction => boîtier :3°C /W
Résistance boîtier=> dissipateur 1.4°C/W
Résistance jonction=>air :50°C/W

Nos données :

Ve=15V
Vs=5V
I=0.5A



1ère solution :

P =(Vs-Ve)*I

Tj-Ta =(Rthjb+Rthrad)P => Rthrad =(Tj-Ta)/P – Rthjb

P =(15-5)*0.5=5W

Rthrad =(125-25)/5 -3=17°C/W

Il nous faut un dissipateur thermique qui puisse dissiper 17°C/W.

Pour éviter d'utiliser un dissipateur thermique nous avons pensé à utiliser deux régulateurs en série pour dissiper cette même puissance ; un régulateur 7809 puis un régulateur 7805. Etudions cette solution.

2ème solution :

7805 :
P =(9-5)*0.5=2

Rthrad =(125-25)/ 2 -3=50°C/W

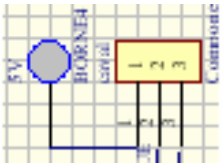
Dans ce cas nous n'avons pas besoin de dissipateur car le régulateur peut dissiper cette dissipation thermique. Mais voyons pour 7809.

7809
P =(15-9)*0.5=3

Rthrad =(125-25)/ 3 -3=30°C/W

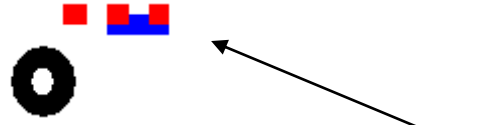
Ici nous voyons qu'il faudra un radiateur donc nous optons pour la première solution, utiliser qu'un seul régulateur le 7805 avec un radiateur qui dissipe 17°C/W.

Sur la carte nous pourrons aussi l'alimenter directement en 5V car il y a un système de cavalier qui permettra à la carte d'être alimentée en 5V sans passer par le régulateur.
Schéma :



5V 15V

Si nous possédons une alimentation de 5V stabilisée nous pourrons réaliser le branchement suivant :

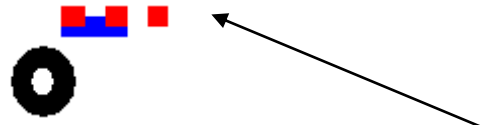


GND

Placer le cavalier comme sur l'image

Dans le cas où nous serons alimentés par la batterie nous devrons utiliser l'autre borne.

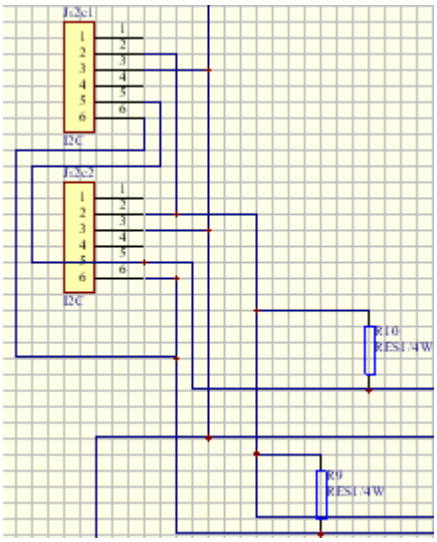
5V 15V



GND

Placer le cavalier comme sur l'image

Le bus I2C:



Nous n'avons pas tout de suite besoin de la liaison avec le bus I2C mais pour ne pas être coincé plus tard si notre projet avance bien nous pourrons l'améliorer avec cette liaison en visualisant directement la distance qui nous sépare du radar sur un afficheur I2C au lieu des diodes.
Nous rappelons le principe de fonctionnement du bus I2C :
<u>Présentation du bus I2C :</u>

Le bus I2C est un bus de liaison série synchrone (l'USB reprend le même principe) sur deux fils plus un fil servant de masse, de neutre ou de GND. Etant synchrone, un des fils (SDA, c'est la ligne de données) du bus transporte les données, et l'autre (SCL, la ligne d'horloge) "synchronise" la communication entre les éléments du bus.

SDA = Serial DAta, c'est la ligne de donnée bidirectionnelle sur laquelle sont véhiculées les données (0 ou 1), par paquets de 8 bits. De 0 à 1,5 volts, la ligne est considérée à l'état bas et de 3 à 5 volts, la ligne est considérée à l'état haut.

SCL = Serial CLock, c'est la ligne d'horloge sur laquelle sont transmises les trains d'impulsions qui conditionnent la validité des bits transmis au même moment sur la ligne SDA. En effet, les bits valides (sur la ligne SDA donc) doivent avoir une durée au moins aussi longue que la durée

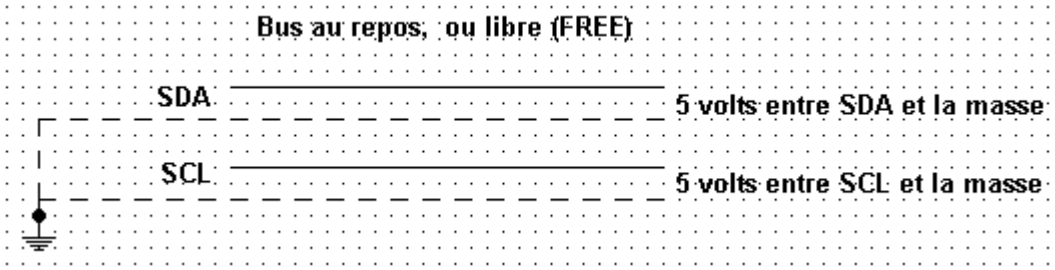
de l'impulsion SCL à l'état haut. Les changements de valeur de SDA se faisant quand SCL est à l'état bas et seulement à ce moment là.

Maître = Celui qui initialise la communication sur le bus, et par conséquent qui la termine. En général c'est un microcontrôleur ou un PC.

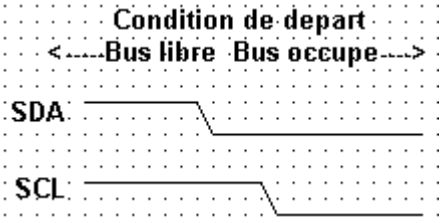
Esclave = Celui qui exécute les ordres du maître, en général un composant compatible avec le protocole I2C. L'esclave ne prend jamais l'initiative d'entamer la conversation avec d'autres éléments, il se contente d'activer ses entrées / sorties, d'accumuler et de restituer des données, de convertir du numérique en analogique et vice versa (PCF8591).

Son fonctionnement :

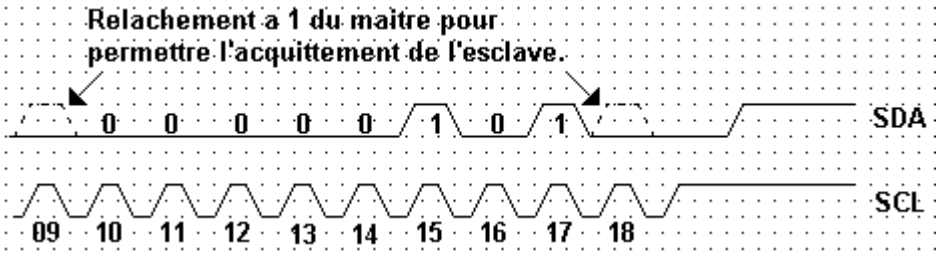
Quand aucune communication ne circule sur le bus, il est dit "au repos". Electriquement, les deux lignes du bus, SDA et SCL sont placées à l'état haut (au +5 Volts) par chaque élément du bus. Il faut savoir néanmoins que l'état haut est valide dès 3 volts, l'état bas étant situé entre 0 et 1,5 volt.



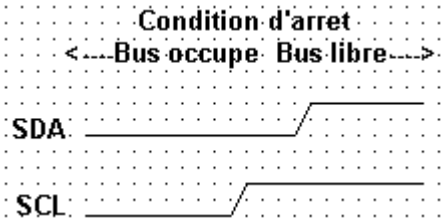
La condition de départ, ne peut avoir lieu que si le bus est au repos, ou libre. Le maître du bus surveille donc l'état du bus et commence par placer la ligne SDA (la ligne de données) l'état bas (au 0 volt) et juste après c'est au tour de la ligne SCL (la ligne d'horloge) de passer a l'état bas. A ce moment, le bus est dit "occupe"et aucun autre maître de bus ne prendra la main pour demander quoi que ce soit.



Apres avoir émis la condition de départ, le maître va "s'adresser" à son esclave en envoyant son adresse. Pendant le même temps, le maître envoie sur le bus les impulsions d'horloge sur la ligne SCL. Un bit de donnée "valide" (0 ou 1) doit être transmis pendant l'état haut de SCL.



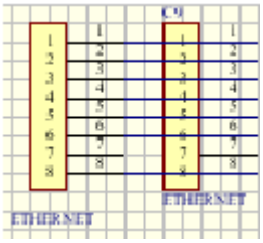
La condition de stop du bus(ou d'arrêt) est très similaire a la condition de départ, mais dans l'autre sens. Dans la condition de départ, c'est SDA qui passe à l'état bas en 1er, ici SDA repasse à l'état haut en dernier.



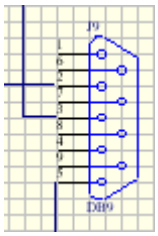
C'est une liaison série qui a beaucoup d'applications même aujourd'hui.

Liaison Ethernet :

Nous allons nous servir de cette liaison pour envoyer des données avec le microcontrôleur et créer une page html. Nous avions déjà accès à la liaison car le rabbit a une connexion au port Ethernet mais pour un souci de fiabilité nous avons préféré mettre le port Ethernet sur notre carte afin d'éviter de brancher et de débrancher et de ce fait abîmer le rabbit.



Liaison série DB9 :



Cette liaison permet de nous connecter avec le rabbit et ainsi de le programmer et de tester le module en recueillant les informations, sur notre PC, délivrer par les satellites.
Pour effectuer cet échange de renseignement nous nous servons d'un composant le MAX232 qui va transformer le TTL en CMOS et vis versa.

Le câblage du MAX 232 à été réaliser comme indiqué sur la documentation constructeur (voir annexe) mais nous avons rajouter un condensateur de découplage supplémentaire, comme indiqué sur la documentation, pour éviter les parasites.

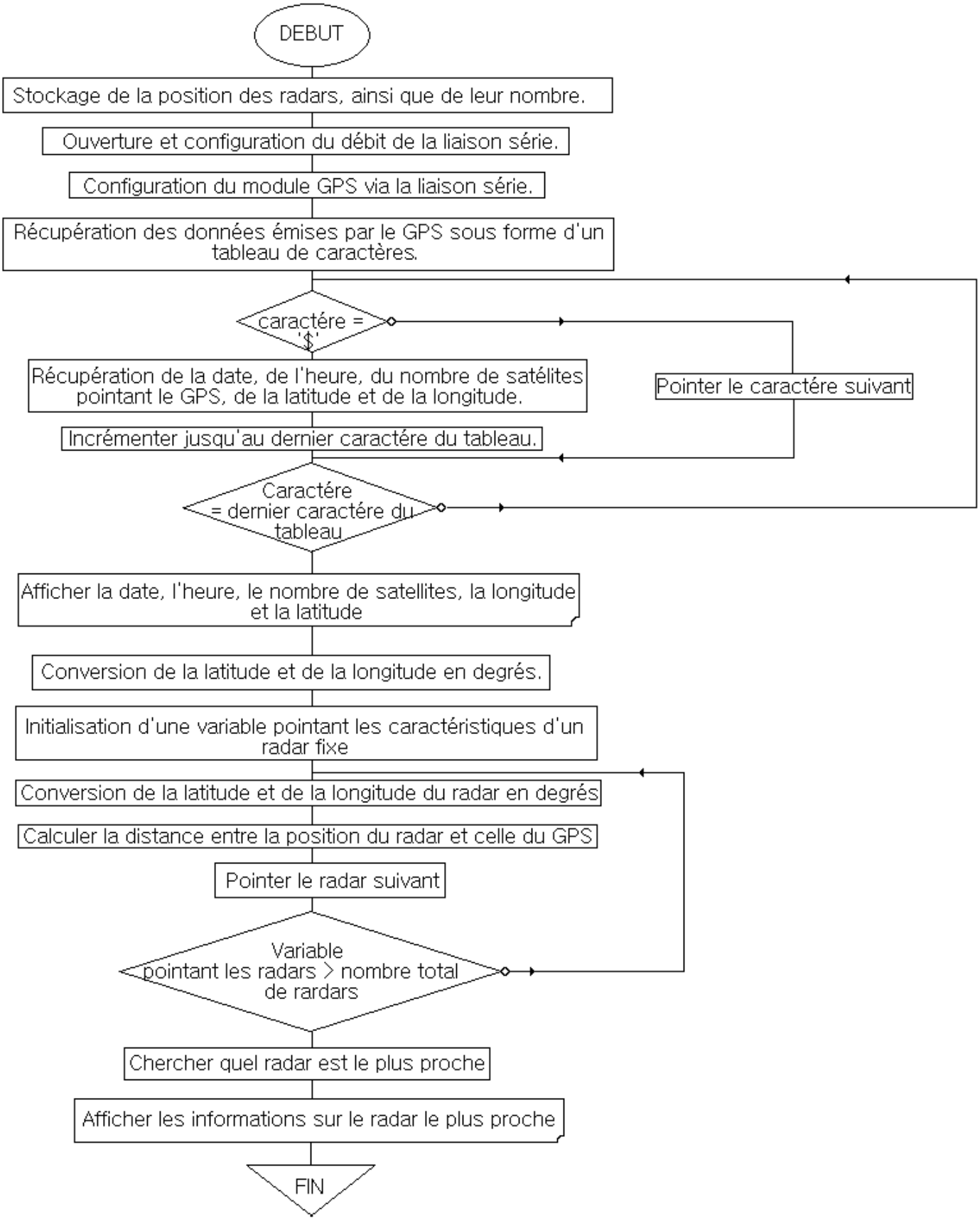
B) Nomenclature

Condensateurs			
Référence	Valeur	Type	Géométrie
C1	1µF	plastique	Radial
C2	47µF	Aluminium Electrolytic	Axial
C4	1µF	plastique	Radial
C5	10µF	Aluminium Electrolytic	Axial
C6	10µF	Aluminium Electrolytic	Axial
C7	10µF	Aluminium Electrolytic	Axial
C8	10µF	Aluminium Electrolytic	Axial

Résistance			
R1	220Ω		Axial
R2	220Ω		Axial
R3	220Ω		Axial
R4	220Ω		Axial
R5	220Ω		Axial
R6	220Ω		Axial
R7	220Ω		Axial
R8	220Ω		Axial
R9	1KΩ		Axial
R10	1KΩ		Axial
LED			
D1	Vert	Electroluminescente	
D2	Orange	Electroluminescente	
D3	Rouge	Electroluminescente	
D4	Vert	Electroluminescente	
D5	Orange	Electroluminescente	
D6	Rouge	Electroluminescente	
D7		Electroluminescente	
D8		Electroluminescente	
Circuits intégrés			
U1		7805	TO-220
U2		rabbit	DIP 40
U3		Max 232	DIP 16
Connecteurs			
Borne 4	GND	Douille de mesure	Ø4mm
Borne 4	15V	Douille de mesure	Ø4mm
Borne 4	5V	Douille de mesure	Ø4mm
J5		Embase Ethernet	RJ45
J6		Embase Ethernet	RJ45
J7		Embase I2C	RJ11
J8		Embase I2C	RJ11
J9		Sub-D	DB9

Du point de vue de la programmation, le programme répondant à la problématique du projet est relativement avancé. Il ne nous manque plus qu'à rentrer la liste complète des radars, puis à faire réagir des leds en fonction de la distance du radar le plus proche et du nombre de satellites accessibles au module pour déclarer le projet de base fini. Cependant, plusieurs pistes seront possibles pour l'améliorer par la suite. Mais nous verrons cela plus loin. Le programme actuel, présent en annexe 3 répond exactement à l'organigramme suivant :

C) Programmation du module par le microcontrôleur



Il convient maintenant de nous appesantir plus longuement sur son contenu afin de mieux cerner son fonctionnement. Tout d'abord, la partie A est la configuration du *Buffer* de la liaison série, limité dans notre cas à 255 caractères avant qu'il ne déborde et ce, en émission, tout comme en réception. Nous ne nous appesantirons pas sur les déclarations de variables pour l'instant, les expliquant au moment où elles interviendront dans le programme.

Passons directement au programme principal (partie B). Toute la première partie de ce programme est le chargement dans des variables de la longitude et de la latitude des radars fixes en Jusqu'à. Ayant toute cette liste au format texte, nous pensions pouvoir la charger automatiquement dans le programme lors de son lancement. Malheureusement, cette méthode n'est pas faisable, car nous ne pouvons pas copier le fichier texte dans le microprocesseur. Il nous a donc fallu nous résoudre à rentrer ces valeurs à la main. Ayant réussi à répertorier 94 radars fixes (la liste exhaustive des radars fixes répertoriés est en Annexe 4), nous enregistrons ces valeurs dans un tableau pour la longitude et un pour la latitude de 95 lignes et d'une colonne (respectivement *Lo[]* et *La[]*, tous deux déclarés comme *Float* étant donné que la longitude et la latitude sont des nombres à virgules). Nous pointons chaque ligne du tableau avec un compteur *i* (déclaré en *integer*). Lorsque les dernières coordonnées ont été rentrées, nous sauvegardons sa valeur dans une variable nommée *nbrad* (déclarée en *integer*), nous renseignant pour la suite du programme sur le nombre de radars enregistrés.

Vient ensuite (partie C) la configuration de la liaison série et du GPS. Tout d'abord, avec l'instruction *serFopen(4800)*, nous déclarons ouvert les ports séries du Rabbit. Ils sont configurés pour une vitesse de transfert de 4800 bauds. Suite à cela, une chaîne de caractères est envoyée. Elle vise à configurer le fonctionnement de la liaison série du GPS, ainsi que de paramétrer les informations qu'il retournera au programme, en accord avec le synoptique ci dessous :

Table 5-11 Set Serial Port Data Format		
Name	**Example**	**Description**
Message ID	$PSRF100	PSRF 100 protocol header
Protocol	1	0=SiRF Binary, 1=NMEA
Baud	4800	4800, 9600, 19200, 3840
DataBits	8	8, 7
StopBits	1	0, 1
Parity	0	0=None, 1=Odd, 2=Even
Checksum	*0C	End of message termination
<CR> <LF>		

Table 5-15 Query/Rate Control Data Format (See example 1.)		
Name	**Example**	**Description**
Message ID	$PSRF103	PSRF103 protocol header
Msg	00	See Table 5-16
Mode	00	0=SetRate, 1=Query
Rate	00	In seconds. Output-off=0, max=255
CksumEnable	00	0=Disable Checksum, 1=Enable Checksum
Checksum	*25	
<CR> <LF>		

Table 5-16 Messages	
0	GGA
1	GLL
2	GSA
3	GSV
4	RMC
5	VTG

Il en résulte que le GPS envoie ses informations à une vitesse de 4800 bauds et que toute autre information que celles répondant à la norme GGA ne sont pas envoyées. Quand aux informations GGA, elles le sont à la cadence d'un envoi par seconde. Il est à noter que *\n\r* est un moyen d'envoyer un retour chariot et un saut de ligne par la liaison série. L'instruction permettant de réaliser cet envoi est s*erFwrite()*.

Vient ensuite la récupération des informations envoyées par le GPS (partie D). Pour ce faire, après avoir remit notre compteur *i* à 0, nous ouvrons une boucle *while()* dans laquelle le programme tournera 151 fois. Ce nombre a été choisi car les informations envoyées par le GPS le sont sous forme d'une chaîne de 76 caractères. Pour être sûr d'avoir une chaîne complète, il faut au minimum récupérer deux fois le nombre de caractères moins un, soit 151. Ensuite, pour récupérer un caractère, il faut utiliser l'instruction *serFread()*.Les données ainsi récupérées sont envoyées dans *s[]* (déclaré en un tableau de *char* à 152 lignes). Une des particularités de cette instruction, c'est qu'elle renvoie le nombre de caractères correctement transmis. Nous utilisons donc une variable nommée *n* (déclarée en *integer*) pour récupérer ce nombre. Enfin, le 1500 à la fin de l'instruction signifie qu'il faut attendre 1500 millisecondes un caractère avant de passer au suivant. Cela à notamment de l'intérêt sur le temps q'il y a entre deux chaînes d'informations envoyées par le GPS. En effet, sur un temps en dessous de la seconde, le tableau *s[]* risque de se remplir de cases vide ou avec des caractères parasites dus au non envoi d'informations sur la liaison série. Cela est d'autant plus dommage qu'on cherche à optimiser la taille des tableaux et le temps d'exécution des programmes.

La partie E concerne principalement le traitement de la chaîne de caractère envoyé par le GPS pour en retirer sa latitude, sa longitude, le nombre de satellites qui lui sont accessibles et l'heure, selon le tableau suivant :

Table 5-2 GA Data Format			
Name	**Example**	**Units**	**Description**
Message ID	$GPGGA		GGA protocol header
UTC Time	161229.487		hhmmss.ssss
Latitude	3723.2475		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12158.3416		dddmm.mmmm
E/W Indicator	W		E=east or W=west
Position Fix Indicator	1		See Table 5-3
Satellites Used	07		Range 0 to 12
HDOP	1.0		Horizontal Dilution of Precision
MSL Altitude	9.0	Meters	
Units	M	Meters	
Geoid Separation		Meters	
Units	M	Meters	
Age of Diff. Corr.		Second	Nul if DGPS id not used

Diff. Ref. Station ID	0000		
Checksum	*18		
<CR> <LF>			End of message termnation

Table 5-3 Position Fix Indicator

Value	**Description**
0	Fix not available or invalid
1	GPS SPS Mode, fix valid
2	Differential GPS, SPS Mode, fix valid
3	GPS PPS Mode, fix valid

Deux variables sont initialisées à 0 : notre compteur *i*, et une autre variable nommée *depart* (déclarée en *integer*). Avec *i*, on scrute *s[i]* jusqu'à trouver le caractère *$*. Une fois ce caractère trouvé, la valeur de *i* est sauvegardée dans *depart*. A partir de là, après avoir remis à zéro la variable *n*, nous isolons certaines parties de la chaine de caractère de manière à récupérer données. Par exemple, pour récupérer l'heure, les caractères situés entre le septième et le treizième de la chaîne sont récupérés dans le tableau *heu[]* (déclaré comme char). Puis, grâce à l'instruction *strtod()*, cette suite de caractères est enregistrée comme nombre dans la variable *heure* (déclarée en *long*). La valeur contenue dans cette variable est ensuite traitée pour en isoler la valeur de l'heure, celle des minutes, et celle des secondes. La méthode mise en œuvre pour ce traitement n'est pas spécialement compliquée à mettre en œuvre mais doit être expliquée car elle est utilisée assez fréquemment dans le reste du programme. Elle utilise la particularité des variables déclarées en *integer* qui fait que, lorsqu'un nombre à virgule est entré dans un variable de ce type, seule sa partie entière est enregistrée. On peut donc, par ce procédé, isoler certaines parties d'un nombre (dans notre cas, la valeur de l'heure, des minutes et des secondes).

L'intégralité du processus ci-dessus pour récupérer l'heure est répétée pour récupérer la valeur de la latitude, de la longitude et du nombre de satellites pointant le module GPS. Une fois toutes ces données récupérées, nous les affichons à l'écran du PC ainsi que la chaîne de caractère. Ceci est particulièrement utile pour voir si le programme a bien fonctionné et, si non, à quel niveau se situe de dysfonctionnement.

Une fois tout ceci affiché, nous devons faire face à un autre problème : les coordonnées récupérées sont en degrés/heure/minute/secondes, ce qui n'est pas sans poser quelques problèmes pour les opérations trigonométriques dont nous aurons besoin plus loin. Il faut donc convertir la latitude et la longitude en degrés (Partie F). Pour arriver à notre but, nous utiliserons la même propriété sur les *integer* que précédemment, isolant consécutivement les degrés, les heures, les minutes et les secondes pour les convertir en degrés et réinjecter dans la variable correspondante (*longitude* ou *latitude*) la valeur convertie.

Ensuite, il faut calculer la distance séparant chaque radar de la position du GPS (partie G du programme). Pour avoir un maximum de précision, il convient de prendre en compte la distance parcourue pour une variation d'un degré de latitude. 360 degrés de latitude représentent 40000kms. Un degré représente donc 111.11km. Pour la distance d'un degré de longitude, elle dépend de la latitude (le périmètre de la terre n'est pas le même le long de l'équateur que suivant la latitude passant en Islande). Le cosinus de la latitude à laquelle se trouve le module doit donc intervenir pour minimiser les erreurs de calcul de la distance. Le calcul à réaliser est le suivant :

$$\sqrt{[(\text{Latitude} - \text{Latitude\_Radar}) \cdot 111.11]^2 + \left[(\text{Longitude} - \text{Longitude\_Radar}) \cdot 111.11 \cdot \cos\left(\frac{\text{Latitude} \cdot 6.283}{360}\right)\right]^2}$$

On remarqueras que la latitude dans le *cos()* est multipliée par 6.283 (environ égal à 2*pi) et divisé par 360, car l'instruction du C converti les grandeurs en radiant. Or, nous manipulons des

grandeurs en degrés. Il nous faut donc les convertir. Toutes ces distances sont enregistrées dans le tableau *dist[]* (déclaré en *float*).

Il faut maintenant trouver le radar le plus proche de notre position. Pour cela, il suffit de trier les distances et d'isoler la plus proche de la notre. La solution mise en œuvre pour cela et de déclarer arbitrairement une distance comme la plus petite et de la soustraire à toutes les autres. Si jamais le résultat d'une soustraction est négatif, nous déclarons la distance ayant été soustraite comme la plus petite et on continue les soustractions jusqu'à ce que toutes les valeurs aient été testées. La fin de ce test isole dans la variable *petit* (déclarée en *integer*) le numéro du radar fixe le plus proche. Il ne reste plus qu'à afficher la distance (en kilomètres à laquelle il se trouve de nous.

Conclusion

Au final, il semble que la problématique initiale d'information par leds soit presque finie. En effet, la conception de la carte électronique touche à sa fin et sa réalisation n nécessite que peu de temps. Il en va de même pour la programmation. A l'heure actuelle, il ne reste plus qu'a commander l'allumage des leds pour considérer le programme fini.

Cependant, la carte électronique a été réalisée de telle manière que des améliorations sont possibles. Tout d'abords, un afficheur LCD commandé par un bus I2C pourras compléter l'information par leds notamment en affichant la distance et la route sur laquelle se trouve le radar le plus proche. Ensuite, il est envisageable de brancher un clavier I2C pour commander les informations qui doivent être affichées (une touche commanderas par exemple l'affichage de l'heure, une autre le nombre de satellites ayant accès au GPS…). Enfin, il est envisageable de rendre ces données visibles sur un ordinateur. Elles seraient transmises par un câble Ethernet et seraient afficher sur une page HTML.

ANNEXE 1

Programme de lecture du mot de huit bit écrit sur le Port 1.

```
#class auto

char A;
```

A:

```
void main(void)
{
A=port1();
printf("le port 1 a pour valeur %x", A);
}
```

B:

```
char port1 (void)
{
char BF;
WrPortI(PBDDR, &PBDDRShadow, 0);
WrPortI(PFDDR, &PFDDRShadow, 0);
BF=(1-BitRdPortI(PFDR, 5))*128+(1-BitRdPortI (PFDR, 4))*64+(1-BitRdPortI
(PBDR, 7))*32+(1-BitRdPortI (PBDR, 5))*16+(1-BitRdPortI(PBDR, 4))*8+(1-
BitRdPortI (PBDR, 3))*4+(1-BitRdPortI (PBDR, 2))*2+(1- BitRdPortI (PBDR, 0));
return BF;
}
```

ANNEXE 2

Programme de dialogue entre le port 1 et le port 0.

```
#class auto


int A, A1;


A:  void main(void)
    {
    WrPortI(PBDDR, &PBDDRShadow, 0);
    WrPortI(PFDDR, &PFDDRShadow, 0);
    WrPortI(SPCR, &SPCRShadow, 0xF4);
    WrPortI(PADR, &PADRShadow, 0);
    A=0;
B :  do
        {
        BitWrPortI(PADR, &PADRShadow , BitRdPortI(PFDR, 5), 0);
        BitWrPortI(PADR, &PADRShadow , BitRdPortI(PFDR, 4), 1);
        BitWrPortI(PADR, &PADRShadow , BitRdPortI(PBDR, 7), 2);
        BitWrPortI(PADR, &PADRShadow , BitRdPortI(PBDR, 5), 3);
        BitWrPortI(PADR, &PADRShadow , BitRdPortI(PBDR, 4), 4);
        BitWrPortI(PADR, &PADRShadow , BitRdPortI(PBDR, 3), 5);
        BitWrPortI(PADR, &PADRShadow , BitRdPortI(PBDR, 2), 6);
        BitWrPortI(PADR, &PADRShadow , BitRdPortI(PBDR, 0), 7);
C :     A1=RdPortI(PADR) ;
        if(A!=A1)
            {
            printf("la valeur est %x\n", A1);
            }
        A=A1;
        }
    while(A!=0xff);
    }
```

ANNEXE 3

Programme de traitement des données envoyées par le GPS.

```
#class auto
```

A:

```
#define FINBUFSIZE 255
#define FOUTBUFSIZE 255

int i,n,depart, nbrad, petit;
char lat[9],lon[10],heu[7],sat[3], s[151],*pEnd, Inf[94][100];
float longitude,latitude, Lo[94], La[94], dist[94], soust;
long heure;
int hh,mm,ss,satellite, ld, lh, lm, ls;
```

B:

```
main()
{
i=0;
Lo[i]=0.10107;
La[i]=45.61760 ;
i++;
Lo[i]=0.14841;
La[i]=45.67828 ;
i++;
Lo[i]=0.28317;
La[i]=46.55447;
i++;
Lo[i]=-0.31145;
La[i]=49.17480;
i++;
Lo[i]=4.80349;
La[i]=45.76310;
i++;
Lo[i]=4.81326;
La[i]=45.77453;
i++;
Lo[i]=4.81674;
La[i]=45.22421;
nbrad=i;
```

C:

```
serFopen(4800);
serFwrite("$PSRF100,1,4800,8,1,0*0C\r\n$PSRF103,00,00,01,00*25\r\n$PSRF103,01,00
,00,00*25\r\n$PSRF103,02,00,00,00*25\r\n$PSRF103,03,00,00,00*25\r\n$PSRF103,04,00,00,00
*25\r\n$PSRF103,05,00,00,00*25\r\n", 177);
```

D:

```
i=0;
while(i<151)
        {
        n=serFread(s,151,1500);
        i++;
        }
```

E:

```
depart=0;
i=0;
```

```
do
        {
        if(s[i]=='$')
                {
                depart=i;
                n=0;
                for(i=depart+7;i<depart+7+7;i++)
                        {
                        heu[n]=s[i];
                        n++;
                        }
                heure=strtol(heu,&pEnd,10);
                hh=(heure/10000);
                mm=(heure/100)-(100*hh);
                ss=(heure)-(100*mm)-(10000*hh);
                n=0;
                for(i=depart+18;i<depart+18+9;i++)
                        {
                        lat[n]=s[i]
                        ;n++;
                        }
                latitude=strtod(lat,&pEnd)/100;
                n=0;
                for(i=depart+30;i<depart+30+10;i++)
                        {
                        lon[n]=s[i];
                        n++;
                        }
                longitude=strtod(lon,&pEnd)/100;
                n=0;
                for(i=depart+46;i<depart+46+3;i++)
                        {
                        sat[n]=s[i];
                        n++;
                        }
                satellite=strtol(sat, &pEnd, 10);
                i=76;
                }
        else
                {
                i++;
                }
        }
while(i<76);
printf("\n\n");
printf("h:%d min:%d sec:%d\t|lat:%f\t|long:%f\t|nbre sat:%d\n\n", hh, mm, ss, latitude,
longitude, satellite);
for(i=depart; i<depart+75; i++)
        {
        printf("%c", s[i]);
        }
printf("\n\n%f", latitude);
```

F : `latitude=1000000*latitude;`

```
ld=latitude/1000000;
lh=(latitude/10000-(ld*100));
lm=(latitude/100-(ld*10000)-(lh*100));
ls=(latitude-(ld*1000000)-(lh*10000)-(lm*100));
printf("\n\n%d\t%d\t%d\t%d", ld, lh, lm, ls);
latitude=((ld*1.0)+(lh/60.0)+(lm/(60.0*100.0))+(ls/(60.0*10000.0)));
longitude=1000000*longitude;
ld=longitude/1000000;
lh=(longitude/10000-(ld*100));
lm=(longitude/100-(ld*10000)-(lh*100));
ls=(longitude-(ld*1000000)-(lh*10000)-(lm*100));
printf("\n\n%d\t%d\t%d\t%d", ld, lh, lm, ls);
longitude=((ld*1.0)+(lh/60.0)+(lm/(60.0*100.0))+(ls/(60.0*10000.0)));
printf("\n\n%f", latitude);
for(i=0; i<=nbrad; i++)
        {
```

G:

```
        dist[i]=sqrt(((latitude-La[i])*111.11)*((latitude-La[i])*111.11)+((longitude-
Lo[i])*111.11*cos(latitude*6.283/360))*((longitude-
Lo[i])*111.11*cos(latitude*6.283/360)));
        }
```

H:

```
i=nbrad;
petit=i;
do
        {
        soust=dist[i-1]-dist[petit];
        if(soust>=0)
                {
                i--;
                }
         else
                {
                petit=i-1;
                i--;
                }
        }
 while(i!=0);
 printf("\n\n%f\t%d", dist[petit], petit);
 serFclose();
}
```

ANNEXE 4

Liste des radars fixes qui seront entrés dans le programme.

dd.ddddd°	dd°mm.mmm'	dd°mm'ss.sss'	
0.10107 45.61760	0°06.064' 45°37.056'	0°06'03.852" 45°37'03.360"	N10 - Km 56.4 La Couronne - 110Km/h
0.14841 45.67828	0°08.905' 45°40.697'	0°08'54.276" 45°40'41.808"	N10 - Km 48.7 Saint-Yrieix - 110Km/h
0.28317 46.55447	0°16.990' 46°33.268'	0°16'59.412" 46°33'16.092"	A10 - Km 310.86 Vounneuil-sous-biard - 130Km/h
-0.31145 49.17480	-0°18.687' 49°10.488'	-0°18'41.220" 49°10'29.280"	N814 - Km 1 Mondeville - Péripherique - 90Km/h
0.31572 47.80293	0°18.943' 48°48.176'	0°18'56.592" 48°48'10.548"	N138 - Km 23 Marigné-Laillé - 70Km/h
0.35583 46.62916	0°21.350' 46°37.750'	0°21'20.988" 46°37'44.976"	N10 - Km 49.25 Migné-Auxances - 70Km/h
0.37138 48.00480	0°22.282' 48°00.288'	0°22'16.932" 48°00'17.280"	N157 - Km 35 Champagné - 90Km/h
-0.38631 44.66545	-0°23.179' 44°39.927'	-0°23'10.716" 44°39'55.620"	N113 - Km 44.578 Virelade - 50Km/h
-0.54874 47.48415	-0°32.924' 47°29.049'	-0°32'55.464" 47°29'02.940"	N23 - Km 34.3 Angers - Voie des berges - 70Km/h
-0.55250 44.88278	-0°33.150' 44°52.967'	-0°33'09.000" 44°52'58.008"	A630 - Km 2.8 Bordeaux Lac - Lormont - Rocade Nord - pont d'aquitaine - 70Km/h
-0.56562 47.46796	-0°33.937' 47°28.078'	-0°33'56.232" 47°28'04.656"	N23 - Km 37 Angers - Voie des berges - 70Km/h
-0.57205 49.23032	-0°34.323' 49°13.819'	-0°34'19.380" 49°13'49.152"	N13 - Km 81.4 Loucelles - 70Km/h
-0.67558 44.83897	-0°40.535' 44°50.338'	-0°40'32.088" 44°50'20.292"	A630 - Km 16.5 Mérignac - Rocade Extérieure - 110Km/h
-0.92550 44.21544	-0°55.530' 44°12.927'	-0°55'31.800" 44°12'55.616"	N10 - Km 24.45 Labouheyre - 90Km/h
-0.92638 44.21257	-0°55.583' 44°12.754'	-0°55'34.968" 44°12'45.252"	N10 - Km 24.75 Labouheyre - 90Km/h
-1.10736 46.11833	-1°06.442' 46°07.100'	-1°06'26.496" 46°07'05.988"	N137 - Km 111.4 La Rochelle - Rocade sud - 90Km/h
-1.13972 46.18153	-1°08.383' 46°10.892'	-1°08'22.992" 46°10'53.508"	N237 - Km 1.9 Lagord - Rocade nord - 90Km/h
1.14520 49.45130	1°08.712' 49°27.078'	1°08'42.720" 49°27'04.680"	N28 Rouen - Rocade Est - 90Km/h
1.46040 43.64670	1°27.624' 43°38.802'	1°27'37.440" 43°38'48.120"	A62 - Km 228 Toulouse - Périphérique Intérieur - 110Km/h
-1.46796 43.51279	-1°28.078' 43°30.767'	-1°28'04.656" 43°30'46.044"	N10 - Km 0.3 Bayonne - 90Km/h
-1.50417 48.10556	-1°30.250' 48°06.334'	-1°30'15.012" 48°06'20.016"	N157 - Km 36.2 Noyal-sur-Vilaine - 110Km/h
1.51933 45.10012	1°31.160' 45°06.007'	1°31'09.588" 45°06'00.432"	A20 - Km 280 Noailles - Tunnel de Noailles - 110Km/h
-1.62366 47.23970	-1°37.420' 47°14.382'	-1°37'25.176" 47°14'22.920"	N844 Nantes - Périphérique Ouest - 90Km/h
-1.85410 47.19496	-1°51.246' 47°11.698'	-1°51'14.760" 47°11'41.856"	D723 Rouans - Route de Paimboeuf - 70Km/h

-1.96244 48.18693	-1°57.746' 48°11.216'	-1°57'44.784" 48°11'12.948"	N12 - Km 84/85 Bédée - 110Km/h
2.06075 48.81462	2°03.645' 48°48.877'	2°03'38.700" 48°48'52.632"	A12 - Km 4.2 St Cyr l'Ecole - 130Km/h
2.08504 48.83310	2°05.068' 48°49.958'	2°05'04.092" 48°49'57.504"	A12 - Km 1/2 Bailly - 110Km/h
2.19011 48.70773	2°11.407' 48°42.464'	2°11'24.396" 48°42'27.828"	N118 - Km 10.8 Orsay - 90Km/h
2.24083 48.88685	2°14.450' 48°53.211'	2°14'26.988" 48°53'12.660"	N13 - Km 2.9 Puteaux - Boulevard Circulaire - 70Km/h
2.25164 48.88751	2°15.049' 48°53.260'	2°15'02.952" 48°53'15.612"	A14 - Km 0.1 La Défense - 70Km/h
2.25442 48.85441	2°15.265' 48°51.265'	2°15'15.912" 48°51'15.876"	Km 12.95 Paris - Périph - Porte de Passy - 80Km/h
2.26472 48.88333	2°15.883' 48°53.000'	2°15'52.992" 48°52'59.988"	N13 - Km 7 Neuilly-sur-Seine - Avenue Charles de Gaulle - 70Km/h
2.27980 48.67046	2°16.788' 48°40.228'	2°16'47.280" 48°40'13.656"	N20 - Km 7.6 La Ville-du-bois - 70Km/h
2.30960 48.89655	2°18.575' 48°53.793'	2°18'34.524" 48°53'47.580"	Km 19.4 Paris - Périph - Porte de Clichy - 80Km/h
2.31000 48.82348	2°18.600' 48°49.409'	2°18'36.000" 48°49'24.528"	Km 6.45 Paris - Périph - Porte de Brançion - 80Km/h
2.32847 49.92363	2°19.708' 49°55.418'	2°19'42.492" 49°55'25.068"	N25 - Km 8 Amiens - Rocade Nord - 110Km/h
2.33206 49.04773	2°19.924' 49°02.864'	2°19'55.416" 49°02'51.828"	N1 Moisselles - 110Km/h
2.36368 48.92995	2°21.821' 48°55.797'	2°21'49.248" 48°55'47.820"	A1 - Km 3.4 Saint Denis - 90Km/h
2.37950 48.82181	2°22.770' 48°49.309'	2°22'46.200" 48°49'18.516"	Km 1 Paris - Périph - Porte d'Ivry - 80Km/h
2.39624 48.88981	2°23.792' 48°53.320'	2°23'47.508" 48°53'19.176"	Km 26.6 Paris - Périph - Porte de Pantin - 80Km/h
2.40663 48.63202	2°24.398' 48°37.921'	2°24'23.868" 48°37'55.272"	A6 - Km 24.435 Courcouronnes - 110Km/h
2.40803 48.77407	2°24.482' 48°46.444'	2°24'28.908" 48°46'26.652"	A86 - Km 31 Vitry-sur-Seine - 90Km/h
2.41464 50.41852	2°24.878' 50°25.111'	2°24'52.704" 50°25'06.672"	N41 - Km 22.8 La Thieuloye - 90Km/h
2.41888 48.76247	2°25.133' 48°45.748'	2°25'07.968" 48°45'44.892"	D38 Choisy-le-Roi - Avenue Villeneuve St Georges - 30Km/h
2.44882 48.81664	2°26.902' 48°49.010'	2°26'54.096" 48°49'00.624"	A4 - Km 4.3 Saint-Maurice - 90Km/h
2.50780 48.82752	2°30.468' 48°49.651'	2°30'28.080" 48°49'39.072"	A4 - Km 9.35 Champigny-sur-Marne - 110Km/h
2.60282 50.53434	2°36.169' 50°32.060'	2°36'10.152" 50°32'03.624"	N43 - Km 30.8 Annezin - 90Km/h
2.60660 48.45506	2°36.396' 48°27.304'	2°36'23.760" 48°27'18.216"	N37 - Km 7.7 Barbizon - 90Km/h
2.60685 48.45612	2°36.411' 48°27.367'	2°36'24.660" 48°27'22.032"	N37 - Km 7.7 Barbizon - 90Km/h
2.68662 49.31897	2°41.197' 49°19.138'	2°41'11.832" 49°19'08.292"	A1 - Km 57.4 Chevrières - 130Km/h
2.85491 49.95796	2°51.295' 49°57.478'	2°51'17.676" 49°57'28.656"	A1 - Km 130.5 Hem Monacu - 130Km/h
3.08972 50.58213	3°05.383' 50°34.928'	3°05'22.992" 50°34'55.668"	A1 - Km 206.1 Lesquin - 110Km/h

3.09636 50.58901	3°05.782' 50°35.341'	3°05'46.896" 50°35'20.436"	A1 - Km 207 Faches-Thumesnil - 110Km/h
3.31571 43.82710	3°18.943' 43°49.626'	3°18'56.556" 43°49'37.560"	A75 - Km 262.6 Pegairolles de l'Escalette - 80Km/h
4.37972 45.46527	4°22.783' 45°27.916'	4°22'46.992" 45°27'54.972"	A72 - Km 6 Saint-Etienne - 70Km/h
-4.40137 48.39042	-4°24.082' 48°23.425'	-4°24'04.932" 48°23'25.512"	N165 Relecq-Kerhuon - Pénétrante Sud de Brest - 90Km/h
4.77083 44.63389	4°46.250' 44°38.033'	4°46'14.988" 44°38'02.004"	N7 - Km 81 La Coucourde - 90Km/h
4.79927 44.12304	4°47.956' 44°07.382'	4°47'57.372" 44°07'22.944"	A7 - Km 168.4 Orange - 130Km/h
4.80349 45.76310	4°48.209' 45°45.786'	4°48'12.564" 45°45'47.160"	A6 - Km 453 Lyon - Tunnel de Fourvière - 90Km/h
4.81326 45.77453	4°48.796' 45°46.472'	4°48'47.736" 45°46'28.308"	N6 Lyon - Tunnel de la Croix-Rousse - 50Km/h
4.81674 45.22421	4°49.004' 45°13.453'	4°49'00.264" 45°13'27.156"	N7 - Km 8.4 Laveyron - 90Km/h
4.83194 45.69596	4°49.916' 45°41.758'	4°49'54.984" 45°41'45.456"	A450 - Km 0.3 Pierre-Bénite - 90Km/h
4.83214 45.69577	4°49.928' 45°41.746'	4°49'55.704" 45°41'44.772"	A450 - Km 0.3 Pierre-Bénite - 90Km/h
4.83812 46.31085	4°50.287' 46°18.651'	4°50'17.232" 46°18'39.060"	N6 - Km 74 Mâcon - Av. du Maréchal-de-Lattre-de-Tassigny - 70Km/h
4.84265 46.78969	4°50.559' 46°47.381'	4°50'33.540" 46°47'22.884"	N6 - Km 16 Chalon-sur-Saône - Avenue du 8 Mai 1945 - 70Km/h
4.84937 45.67996	4°50.962' 45°40.798'	4°50'57.732" 45°40'47.856"	D301 - Km 0.5 Feyzin - Boulevard Urbain Sud - 50Km/h
4.88397 44.03829	4°53.038' 44°02.297'	4°53'02.292" 44°02'17.844"	A7 - Km 182 Bédarrides - 110Km/h
4.88609 44.04282	4°53.165' 44°02.569'	4°53'09.924" 44°02'34.152"	A7 - Km 181.4 Bédarrides - 110Km/h
4.90681 45.76511	4°54.409' 45°45.907'	4°54'24.516" 45°45'54.396"	N383 - Km 6 Villeurbanne - Boulevard Laurent Bonnevay - 90Km/h
4.90691 45.76501	4°54.415' 45°45.901'	4°54'24.876" 45°45'54.036"	N383 - Km 6 Villeurbanne - Boulevard Laurent Bonnevay - 90Km/h
5.35371 43.37993	5°21.224' 43°22.795'	5°21'13.464" 43°22'47.676"	A7 - Km 273.2 Septème-les-Vallons - 110Km/h
5.73178 45.55753	5°43.907' 45°33.452'	5°43'54.408" 45°33'27.108"	A43 - Km 73 Dullin - Tunnel de Dullin - 110Km/h
5.82611 45.57805	5°49.567' 45°34.683'	5°49'33.996" 45°34'40.980"	A43 - Km 81.7 La Motte Servolex - Tunnel de l'Epine - 110Km/h
5.93383 45.57345	5°56.030' 45°34.407'	5°56'01.788" 45°34'24.420"	N201 - Km 2 Chambéry - Voie Rapide Urbaine - 90Km/h
5.95160 43.12068	5°57.096' 43°07.241'	5°57'05.760" 43°07'14.448"	A57 Toulon - Tunnel de Toulon - 70Km/h
6.08228 45.91465	6°04.937' 45°54.879'	6°04'56.208" 45°54'52.740"	N508 Annecy - Voie rapide - 90Km/h
6.14639 49.10639	6°08.783' 49°06.383'	6°08'47.004" 49°06'23.004"	A31 - Km 301.6 Montigny-lès-Metz - 90Km/h
6.16173 49.12735	6°09.704' 49°07.641'	6°09'42.228" 49°07'38.460"	A31 - Km 304.4 Metz - 90Km/h
6.17456 49.27269	6°10.474' 49°16.361'	6°10'28.416" 49°16'21.684"	A31 - Km 321.4 Richemont - 110Km/h
6.25431 48.61852	6°15.259' 48°37.111'	6°15'15.516" 48°37'06.672"	A33 - Km 15.8 Nancy - 130Km/h

6.36361 46.81250	6°21.817' 46°48.750'	6°21'49.000" 46°48'45.000"	N57 - Km 81 Les-Hopitaux-Neufs - 90Km/h
6.36365 46.81255	6°21.817' 46°48.750'	6°21'49.000" 46°48'45.000"	N57 - Km 81 Les-Hopitaux-Neufs - 90Km/h
6.51916 45.49333	6°31.150' 45°29.600'	6°31'08.976" 45°29'35.988"	N90 - Km 48 Moûtiers - Tunnel de Ponserand - 80Km/h
6.68232 45.16552	6°40.939' 45°09.931'	6°40'56.352" 45°09'55.872"	A43 Modane - Tunnel de Fréjus - 70Km/h
6.82666 43.54555	6°49.600' 43°32.733'	6°49'35.976" 43°32'43.980"	A8 - Km 147 110Km/h
6.83720 43.54003	2°36.411' 48°27.367'	2°36'24.660" 48°27'22.032"	A8 - Km 149.3 110Km/h
6.84777 43.53861	6°50.866' 43°32.317'	6°50'51.972" 43°32'18.996"	A8 - Km 148 110Km/h
6.86304 45.90038	6°51.782' 45°54.023'	6°51'46.944" 45°54'01.368"	N205 Chamonix - Tunnel du Mont-Blanc - 70Km/h
7.20728 43.66579	7°12.437' 43°39.947'	7°12'26.208" 43°39'56.844"	N98 - Km 31/32 Nice - Promenade des Anglais - 70Km/h
7.25313 43.73417	7°15.188' 43°44.050'	7°15'11.268" 43°44'03.012"	A8 - Km 196 Nice - Tunnel de Las Planas - 90Km/h
7.55726 48.49228	7°33.436' 48°29.537'	7°33'26.136" 48°29'32.208"	A35 - Km 12.7 Innenheim - Voie rapide du piémont des vosges - 130Km/h
7.70984 48.53605	7°42.590' 48°32.163'	7°42'35.424" 48°32'09.780"	A35 Ostwald - Rocade de Strasbourg - 90Km/h
7.73861 48.57055	7°44.317' 48°34.233'	7°44'18.996" 48°34'13.980"	N4 - Km 45.2 Strasbourg - Pénétrante de l'Etoile - 90Km/h
7.77506 48.67762	7°46.504' 48°40.657'	7°46'30.216" 48°40'39.432"	A35 - Km 245.6 Hoerdt - 130Km/h

http://focus.ti.com/lit/ds/symlink/max232.pdf

http://www.commentcamarche.net/elec/connecteur-prise-db9.php3

http://www.ortodoxism.ro/datasheets/fairchild/LM7805.pdf

MC78XX/LM78XX/MC78XXA

3-Terminal 1A Positive Voltage Regulator

Features

- Output Current up to 1A
- Output Voltages of 5, 6, 8, 9, 10, 12, 15, 18, 24V
- Thermal Overload Protection
- Short Circuit Protection
- Output Transistor Safe Operating Area Protection

Description

The MC78XX/LM78XX/MC78XXA series of three terminal positive regulators are available in the TO-220/D-PAK package and with several fixed output voltages, making them useful in a wide range of applications. Each type employs internal current limiting, thermal shut down and safe operating area protection, making it essentially indestructible. If adequate heat sinking is provided, they can deliver over 1A output current. Although designed primarily as fixed voltage regulators, these devices can be used with external components to obtain adjustable voltages and currents.

TO-220

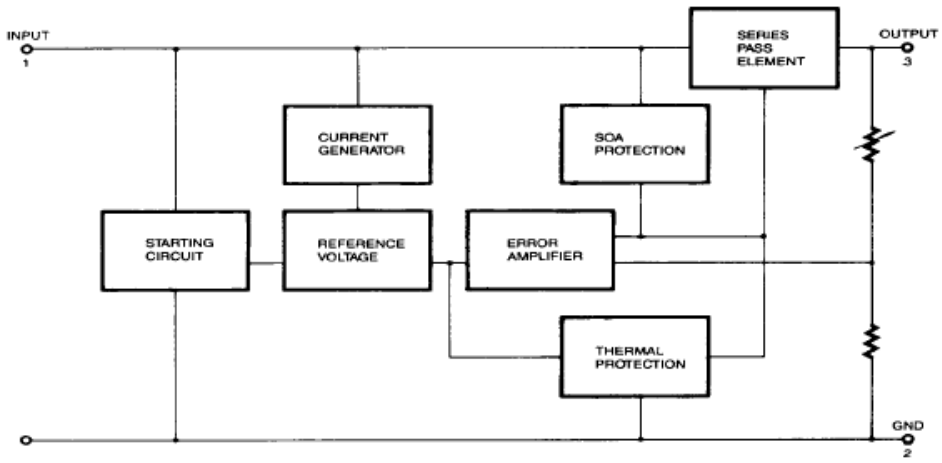
1

D-PAK

1

1. Input 2. GND 3. Output

Internal Block Digram



Rev. 1.0.1

Absolute Maximum Ratings

Parameter	Symbol	Value	Unit
Input Voltage (for V_O = 5V to 18V) (for V_O = 24V)	V_I V_I	35 40	V V
Thermal Resistance Junction-Cases (TO-220)	$R_{\theta JC}$	5	°C/W
Thermal Resistance Junction-Air (TO-220)	$R_{\theta JA}$	65	°C/W
Operating Temperature Range	T_{OPR}	0 ~ +125	°C
Storage Temperature Range	T_{STG}	-65 ~ +150	°C

Electrical Characteristics (MC7805/LM7805)

(Refer to test circuit ,0°C < T_J < 125°C, I_O = 500mA, V_I = 10V, C_I= 0.33µF, C_O= 0.1µF, unless otherwise specified)

Parameter	Symbol	Conditions		MC7805/LM7805 Min.	Typ.	Max.	Unit
Output Voltage	V_O	T_J =+25 °C		4.8	5.0	5.2	V
		5.0mA ≤ Io ≤ 1.0A, P_O ≤ 15W V_I = 7V to 20V		4.75	5.0	5.25	
Line Regulation (Note1)	Regline	T_J=+25 °C	V_O = 7V to 25V	-	4.0	100	mV
			V_I = 8V to 12V	-	1.6	50	
Load Regulation (Note1)	Regload	T_J=+25 °C	I_O = 5.0mA to1.5A	-	9	100	mV
			I_O =250mA to 750mA	-	4	50	
Quiescent Current	I_Q	T_J =+25 °C		-	5.0	8.0	mA
Quiescent Current Change	ΔI_Q	I_O = 5mA to 1.0A		-	0.03	0.5	mA
		V_I= 7V to 25V		-	0.3	1.3	
Output Voltage Drift	$\Delta V_O/\Delta T$	I_O= 5mA		-	-0.8	-	mV/ °C
Output Noise Voltage	V_N	f = 10Hz to 100KHz, T_A=+25 °C		-	42	-	µV/Vo
Ripple Rejection	RR	f = 120Hz V_O = 8V to 18V		62	73	-	dB
Dropout Voltage	V_{Drop}	I_O = 1A, T_J =+25 °C		-	2	-	V
Output Resistance	r_O	f = 1KHz		-	15	-	mΩ
Short Circuit Current	I_{SC}	V_I = 35V, T_A =+25 °C		-	230	-	mA
Peak Current	I_{PK}	T_J =+25 °C		-	2.2	-	A

Note:

1. Load and line regulation are specified at constant junction temperature. Changes in V_O due to heating effects must be taken into account separately. Pulse testing with low duty is used.