Enregistrement de trames GPS – développement sur microcontrôleur 8051/8052 sous GNU/Linux

Jean-Michel FRIEDT, Émile CARRY, 11 novembre 2005
friedtj@free.fr, milou@sequanux.org
Association Projet Aurore, Maison des Étudiants, 36A avenue de l'Observatoire, 25030 Besançon

Un certain nombre de projets visant à constituer des bases de données opensource de chemins et routes ont été lancés [1, 2]. Ces projets s'organisent autour de la contribution volontaire de traces GPS accumulées lors de trajets avec l'espoir que suffisamment de contributeurs finiront par constituer une cartographie dense et précise des routes.

Cependant, les récepteurs GPS sont encore chers et ne répondent pas nécessairement aux contraintes d'un tel projet : stocker continuellement un grand nombre de points au cours du trajet pour un traitement et une restitution ultérieurs. Nous proposons ici une solution faible coût pour palier à ces inconvénients : partant d'un récepteur GPS minimaliste, monter une interface de stockage pour mémoriser les traces.

Ce projet peut par ailleurs former le point de départ de montages plus ambitieux puisqu'il s'agit en fait d'un mode de stockage universel de trames RS232 mais aussi de données analogiques puisque déjà ici nous ajoutons la capacité de stocker la température en plus des trames GPS.

Nous allons donc présenter ici les développements matériels et logiciels sur un microcontrôleur peu coûteux répondant à nos exigences de faible consommation et encombrement, la connexion des périphériques nécessaires au positionnement (récepteur GPS OEM) et de stockage (MultiMediaCard) pour finalement aborder les méthodes de traitement et d'affichage des données acquises et leur comparaison aux cartes disponibles sur internet.

1 Le microcontrôleur – le cœur 8051/8052

Analog Devices propose depuis quelques années une gamme de microcontrôleurs appelés "convertisseurs intelligents" : la série des ADuC. Il s'agit en fait d'un cœur de microcontrôleur 8052 contrôlant des périphériques très performantes auxquelles Analog Devices nous a habitué. Il s'agit en pratique de convertisseurs analogiques-numériques rapides (jusqu'à 400 kéchantillons/s) ou précis (jusqu'à 24 bits) entourés de ports de communications pratiques (SPI/I^2C, RS232) et d'options selon les versions telles que dans le cas qui va nous intéresser ici des timers et des convertisseurs numériques-analogiques. Avec autant de fonctionnalités les boîtiers sont souvent difficiles à manipuler et nous avons ici choisi de décrire l'ADuC814 sélectionné pour son nombre restreint de broches qui le rend compatible avec un circuit imprimé simple face et son utilisation par un amateur sans équipement spécial, voir même un montage en l'air sans support de circuit imprimé pour le prototypage.

L'ADuC814 offre dans un boîtier CMS à 28 broches 8 kB de mémoire non-volatile (flash) pour la programmation, 256 octets de RAM, 6 convertisseurs analogiques-numériques (247 kéchantillons/s) et deux convertisseurs numériques-analogiques sur 12 bits de résolution dans tous les cas. 640 octets de mémoire non-volatile accessible depuis un programme en cours d'exécution vont nous permettre de stocker des paramètres d'exécution d'un programme sans réinitialisation en cas de perte de tension d'alimentation (5 V ou 3,3 V).

Nous utilisons un montage très classique (Fig. 1) tel que décrit dans la notice d'utilisation[1] avec connexion d'un quartz externe à 32,768 kHz (multiplié en interne par une PLL), d'un simple interrupteur pour la réinitialisation (*RESET*) puisque le circuit anti-rebond est fourni en interne par le microcontrôleur, et quelques composants passifs pour la gestion des tensions des convertisseurs. Pour une connexion à un PC (par exemple lors de la phase de programmation), un circuit classique de mise à niveau des tensions du port RS232 est nécessaire : classiquement un MAX232 alimenté en 5 V, ou un MAX3232 alimenté en 3,3 V si seule cette tension est disponible, voir le Dallas DS276 qui élimine même la nécessité des condensateurs si encombrants sur les composants

[1] http://www.analog.com/en/prod/0,,762_0_ADUC814,00.html

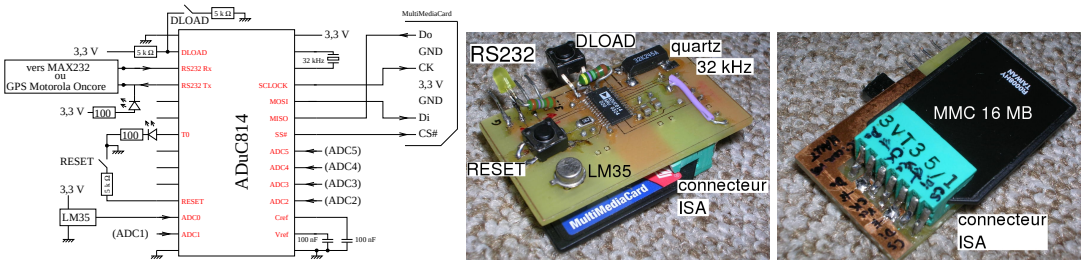


FIG. 1 – Gauche : schéma de principe du circuit basé sur un ADuC814, auquel sont connectés un capteur de température LM35 et une MultiMediaCard pour le stockage de masse non-volatile des données acquises. La communication RS232 se fait soit directement avec le récepteur GPS Motorola Oncore qui communique en 0/5 V, soit *via* un MAX232 pour communiquer avec un PC en ±12 V. La diode électroluminscente sur la broche de transmission RS232 nous sert d'indicateur pour vérifier le bon fonctionnement du programme exécuté (s'allume à chaque transmission par le port série de l'ADuC814). Toutes les valeurs de résistances sont données à titre indicatif mais il n'est pas nécessaire de les respecter scrupuleusement : toutes valeurs du même ordre de grandeur fera l'affaire. Milieu et droite : photos du montage utilisé au cours de ces expériences. Le circuit imprimé fait 45×27 mm^2. Nous utilisons un bout de connecteur ISA (pas 2,54 mm) comme support de la MMC.

cités auparavant. L'interrupteur DLOAD ouvert nous permet de charger un nouveau programme en mémoire non volatile du microcontrôleur selon la procédure décrit ci-dessous, ou d'exécuter le programme stocké en interne lorsque l'interrupteur est fermé au démarrage.

```
        .area code (ABS)
        .org 0h0000

        mov     A,#0h0a
begin:  cpl     0hB4    ; T0=P3.4=#22
        cpl     0h90    ; T2=P1.0=#1
        lcall   wait
        sjmp    begin
wait:   MOV     R0,a
lo0:    mov     r1,#065
lo2:    mov     r2,#0200
lo1:    djnz    r2,lo1
        djnz    r1,lo2
        djnz    r0,lo0
        ret
```

TAB. 1 – Premier exemple : faire clignoter une diode. Cet exemple classique permet de découvrir la syntaxe, les opcodes de base et la structure générale d'un programme [3] tel que flashé dans l'ADuC814. La syntaxe est celle utilisée pas `asxxxx` pour compiler une code à l'adresse de départ 0x0000. Une diode connectée à la broche 22 ou 1 clignote lorsque ce code est exécuté.

Nous avons utilisé dans notre montage le très classique 7805 pour réguler la tension de sortie de la batterie – variable entre 8,5 et 6,5 V – en un tension de sortie fixe de 5 V tel que requis par le récepteur GPS, tension qui est ensuite abaissée à 3,3 V pour alimenter le microcontrôleur et la MultiMediaCard nécessaire au stockage des données (section 3). Ce choix est peu judicieux car le 7805 nécessite une tension d'entée de 7.5 V, diminuant d'un facteur 2 l'autonomie de nos batteries. Nous expérimentons désormais avec le LM2940CT-5.0 [2] qui accepte une tension d'entrée de 6,5 V pour le même résultat et double donc l'autonomie du montage. Nous plaçons le régulateur fournissant la tension de 3,3 V (LM1086-3.3) en *série* avec le régulateur 5 V (et non directement sur la batterie) : nous protégeons ainsi notre accumulateur Lithium-Polymère (LiPo) contre une décharge excessive qui l'endommagerait. En effet, la régulateur 5 V voit sa tension de sortie chuter si la tension d'entrée passe sous les 6,5 V, et ainsi coupe le fonctionnement du récepteur GPS (qui nécessite 5 V pour fonctionner) et du régulateur de tension 3,3 V : la tension de la batterie

[2] Référence 412727 chez Farnell, 2,91 euros/pièce, le LM1086-5.0 anciennement à 0,67 euro/p n'étant plus disponible

ne passera donc jamais sous la valeur fatidique des 6,4 V qui ne doit pas être atteinte avec les accumulateurs LiPo.

Comme la majorité des microcontrôleurs récents, l'ADuC814 est équipé d'un bootloader en ROM qui se charge des opérations de programmation des diverses mémoires disponibles autour du processeur. Ce bootloader communique au démarrage sur le port série : nous décrirons plus loin en détail le protocole de communication (section 2).

2 Outils de développement sous linux

Un avantage de travailler sur un processeur aussi commun que le 8052 est la disponibilité de plusieurs assembleurs opensource : par habitude, nous avons travaillé avec la version adaptée de `asxxxx` (`http://shop-pdp.kent.edu/ashtml/asxxxx.htm`). Une alternative est de travailler avec `asl` avec une syntaxe légèrement différente.

Le premier exemple que nous présentons table 1 vise à faire clignoter une diode connectée à une broche de l'ADuC814 et surtout à se familiariser avec la syntaxe de l'assembleur et l'architecture du microcontrôleur. Le programme sera stocké et exécuté depuis l'adresse de début de la mémoire flash : 0x0000. On identifie dans ce programme la syntaxe d'accès à l'accumulateur (*i.e.* le registre "brouillon" auquel accèdent presque toutes les commandes) A, les registres généraux Ri ($i \in [0:7]$) et l'accès à une broche d'un port d'entrée/sortie – T0 ou T2 – par une opération de complément (`cpl`) sur un bit unique. La pile est implicitement initialisée lors du démarrage du coœur de 8051.

Un second exemple présenté table 2 un peu plus complexe offre une application concrète de l'ADuC814 : lire une valeur sur un port du convertisseur analogique-numérique et la communiquer par RS232. Bien que les interruptions ne soient pas utilisées dans ce programme, nous prévoyons de libérer les 96 premiers octets qui contiennent les vecteurs d'interruption. Le vecteur de reset, situé en 0x0000, contient dans ce cas une instruction de saut vers le programme principal situé donc en 0x0060 (chaque vecteur d'interruption réserve 8 octets d'opcode pour soit permettre une instruction courte, soit un saut vers un routine de gestion d'interruption, qui se conclue par `RETI`). On trouve là une routine d'initialisation du port RS232 à 9600 bauds [4], la séquence d'initilisation du convertisseur analogique-numérique et la sélection du canal actif, et finalement le transfert en binaire de la valeur lue – octet de poids fort suivi de l'octet de poids faible – pour une transmission finale de l'ordre d'une valeur par seconde.

Une fois le programme assemblé et converti en une suite d'opcodes compréhensibles par le processeur (au moyen de `as8051` fourni dans `asxxxx` selon une procédure identique à celle présentée antérieurement [5] – pour rappel il s'agit d'extraire par `grep ^T prog.rel | cut -c9-80 > prog.out` les opcodes issus de l'assemblage du programme `prog.asm` en `prog.rel` par `as8051 -o prog.asm`), il nous reste à transférer le programme en mémoire non-volatile (flash) du microcontrôleur pour l'y exécuter. Nous avons à ces fins écrit un petit programme fonctionnant sous linux chargé de transférer par le port série (RS232) le programme en vérifiant l'intégrité des communications : l'ensemble de nos développements est disponible à `www.sequanux.org` ou `jmfriedt.free.fr`. Ce programme communique via le port RS232 avec le bootloader de l'ADuC814 dont l'exécution est automatique si la broche DLOAD est connectée à la tension d'alimentation. Une fois le programme chargé, il peut être directement exécuté depuis la mémoire flash : en connectant la broche DLOAD à la masse, le code utilisateur stocké en mémoire flash est directement exécuté au reset.

La communication avec le bootloader ne nécessite que d'implémenter quelques commandes mises à disposition par Analog Devices [6] :

- séquence d'initialisation 0x21 0x5A 0x00 0xA6 qui demande au microcontrôleur de s'identifier, ce à quoi le bootloader de l'ADuC814 répond par un entête de 25 octets contenant notamment la chaîne `ADI 814`
- toute commande se transfère par la séquence 0x07 0x0E suivie du nombre de caractères transmis excluant la commande (1 par exemple s'il n'y a pas d'argument mais seulement une commande à exécuter telle que 'A' pour effacer toute la mémoire) et finalement le checksum. Ce dernier ce calcule comme le complément à deux de la somme des octets transmis (excluant

```
; stty -F /dev/ttyS0 19200   to set to 19200 bauds
; ADC sur 12 bits => volts=(hi*256+lo)/4096*2.5

        LED     = 0hb3
        CHAN0   = 0             ; convert this ADC input channel (0 thru 6)

        .area code (ABS)
        .org 0h0000
        ljmp    MAIN

        .area code (ABS)
        .org 0h0060
MAIN:                           ; Main program
; change core freq: default is 0h03=PLL freq/8, so that 0h02 doubles the freq
; and 9600 bauds settings become 19200 bauds
;        MOV     0hD7,#0h02     ; double internal PLL freq (814) (D7=PLLCON)

        MOV     RCAP2H,#0hFF    ; config UART for 9830baud (~9600 bauds)
        MOV     RCAP2L,#-7      ; -7 for ADuC814, -5 for ADuC816
        MOV     TH2,#0hFF
        MOV     TL2,#-7         ; -7 for ADuC814, -5 for ADuC816
        MOV     SCON,#0h52
        MOV     T2CON,#0h34

       MOV     0hEF,#0h80   ; power up ADC -- ADCCON1=0hEF
       SETB    0hAF         ; enable interrupts -- EA=0hAF
       SETB    0hAE         ; enable ADC interrupt -- EADC=0hAE


START:  CLR     LED             ; temporary register
        MOV     0hD8,#CHAN0     ; ADC channel select
        SETB    0hDC            ; start ADC conversion
iciadc: JNB     LED,iciadc      ; wait until conversion completed
        MOV     A,0hDA          ; read hi byte
        LCALL   SENDCHAR
        MOV     A,0hD9          ; read lo byte
        LCALL   SENDCHAR

WAIT5S: MOV       A, #10        ; wait 5s
        LCALL     DELAY
        SJMP      START        ; start transmissions again
;-------------------------------------------------------------------
                                                        ; SENDCHAR
SENDCHAR:       ; sends ASCII value contained in A to UART

        JNB     TI,SENDCHAR             ; wait til present char gone
        CLR     TI              ; must clear TI
        MOV     SBUF,A
        RET
;-------------------------------------------------------------------
                                                            ; DELAY
DELAY:                  ; Delays by 100ms * A
                        ; 100mSec based on 1.573MHZ Core Clock
          MOV   R2,A     ; Acc holds delay variable
 DLY0:    MOV   R3,#50   ; Set up delay loop0
 DLY1:    MOV   R4,#131  ; Set up delay loop1
ici1:     DJNZ  R4,ici1  ; Dec R4 & Jump here until R4 is 0
                         ; wait here for 131*15.3us=2ms
          DJNZ  R3,DLY1  ; Dec R3 & Jump DLY1 until R3 is 0
                         ; Wait for 50*2ms
          DJNZ  R2,DLY0  ; Dec R2 & Jump DLY0 until R2 is 0
                         ; wait for ACC*100ms
          RET            ; Return from subroutine
;-------------------------------------------------------------------
```

Tab. 2 – Un exemple plus concret qui permet d'explorer la majorité des commandes mises à disposition par le cœur de 8051 : il s'agit ici d'un programme qui lit des valeurs de tension sur le port de conversion analogique-numérique pour les restituer sur le port RS232 selon le protocole de 9600 bauds, N81. Les fonctions de communication sont basées sur les exemples fournis par Analog Devices [4].

l'entête 0x07 0x0E).

- effacer l'ensemble de la mémoire par la commande 0x43 ('C') qui efface à la fois le programme et la zone de données, ou 0x41 ('A') qui n'efface que le programme et conserve les données stockées en mémoire flash.
- écrire un octet de programme à l'adresse A (A étant un mot sur 24 bits) par 'W' suivi des 3 octets de A (octet de poids fort en premier) suivi des données à écrire. Pour cette commande, le nombre d'octets à transmettre (tel que transmis juste après la séquence d'initialisation de communication 0x07 0x0E) indique au microcontrôleur la taille du programme transmis
- écrire un octet de données à une adresse se fait comme précédemment mais avec la commande 'E'
- exécuter le programme depuis la mémoire flash par la commande 0x55 ('U').

Dans tous les cas le bootloader acquitte une commande en répondant par la valeur 0x06, ou informe d'une erreur au cours de l'exécution de la commande transmise par 0x07.

3 La MultiMediaCard : stockage de masse non-volatile

Nous avions déjà mentionné [7] l'intérêt de la MultiMédiaCard (MMC) comme support de stockage de masse non-volatile : peu coûteux, facilement disponible à l'unité, ne nécessitant que peu de fils pour la communication (3 fils selon un protocole SPI) et avec documentation gratuitement disponible sur internet [8, 9, 10, 11]. Nous allons ici profiter de la disponibilité d'un port SPI parmi les périphériques de l'ADuC814 pour communiquer avec la MMC. Nous allons expliciter son mode de fonctionnement, les protocoles d'initialisation et de communication, en nous basant dans un premier temps sur une interface entre la MMC et le port parallèle d'un PC qui nous donnera un environnement plus sympathique que le développement sur microcontrôleur. Cette interface pour être utilisée ultérieurement pour la lecture rapide du contenu d'une MMC.

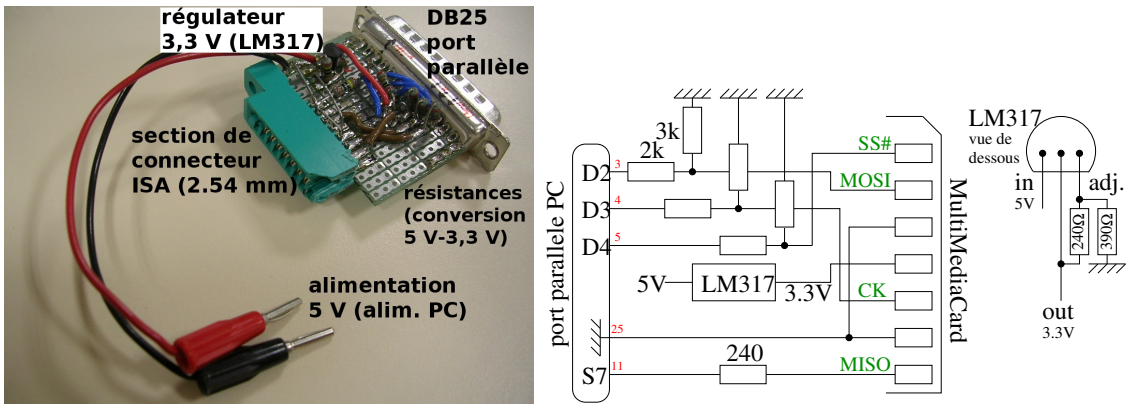


FIG. 2 – Gauche : exemple de montage réalisé sur circuit imprimé pré-perforé pour connecter une MMC au port parallèle d'un PC. Quelque composants passifs sont nécessaires pour la conversion des niveaux de tension du port parallèle du PC (TTL 5 V) à ceux de la MMC (3,3 V) alimentée par un régulateur de tension variable LM317 reglé pour fournir une tension proche de 3,3 V. Droite : schéma de principe du montage dans lequel une paire de résistances montées en diviseur de tension sur chaque ligne de donnée D2-D4 du port parallèle convertit la sortie 5 V en 3,3 V, tandis qu'une résistance sur la ligne de statut limite le courant pouvant circuler entre la MMC et le port parallèle.

Le protocole SPI est un protocole synchrone avec un fil pour l'horloge (CK), et deux fils de communication unidirectionnelle des données, l'un du maître vers l'esclave (MOSI : Master Out Slave In) et l'autre de l'esclave vers le maître (MISO). Ce protocole est particulièrement approprié pour être émulé sur des ports d'entrée-sortie généralistes tel que mis à disposition par le port parallèle. La seule astuce de connection consiste en la mise à niveau des tensions : le signal à 5 V

issu du port parallèle est abaissé à 3,3 v tel que requis par la MMC par un pont de résistances diviseur de tension (signaux SS# définissant l'activation de la carte par un niveau bas, aussi nommé parfois CS#, ainsi que CK et MOSI), tandis que le signal à 3,3 V issu de la MMC (MISO) est relevé à 5 V par une résistance de pull-up. Nous allons ici présenter une implémentation d'un émulateur de protocole SPI sur le port parallèle d'un PC afin de pouvoir écrire et lire des données sur une MultiMediaCard (testé sur des cartes de 16 et 128 MB) : les développements sont ainsi facilités par rapport à l'implémentation directe sur un microcontrôleur. L'implémentation de ce même protocole est d'autant plus simple sur microcontrôleur que le protocole y est supporté comme un périphérique matériel (et non émulé de façon logicielle comme dans le cas du port parallèle).

Le protocole SPI étant un protocole synchrone, son émulation logicielle est très simple : une transition sur le bus de données est synchronisée sur les transitions du signal d'horloge dont le maître impose la fréquence. Contrairement à un protocole asynchrone tel que RS232 (où chaque partenaire de la communication possède sa propre horloge, ces horloges n'étant pas synchronisées entre elles), nous n'avons donc ici aucune contrainte de temporisation : le port parallèle impose des créneaux sur le signal CK (clock) et émet ou lit la donnée à un moment prédéfini par le protocole (soit après le front montant, soit après le front descendant de l'horloge). Un système d'exploitation chargé par de nombreuses tâches effectuées en parallèle tel que GNU/Linux imposera donc une horloge plus lente mais ne risque en aucun cas de perdre la synchronisation de la communication comme cela peut arriver dans un protocole asynchrone. Une implémentation de ce protocole est présenté dans la table 3.

```
#define LPT1 0x378

char SendByte(char byte){
// CLOCK on pin 4, MISO=pin 11 (input), MOSI=pin 3 (output), SS=pin 5 (output)
  char input=0,bitin,bit,Data;
  int i;
  Data=InPort(LPT1);
  for(i=0;i<8;i++){                                    //get first bit to send
    bit=(byte & 0x80)>>6;    // checks what MSb is and moves it to 2nd position
    byte = byte << 1;
    Data=Data&~CLOCK;                                          //lower CLOCK
    OutPort(LPT1,Data);
    Data=((Data&~MOSI)|bit);                           //put DATA on MOSI line
    OutPort(LPT1,Data);
    Data=Data|CLOCK;                                           //raise CLOCK
    OutPort(LPT1,Data);
    bitin=(((InPort(LPT1+1)^MISO)&MISO)>>7);
    input=input<<1;
    input=input|bitin;
  }
  Data=Data|MOSI;                //Raise Data - acting like a pull-up resistor
  OutPort(LPT1,Data);
  return input;
}
```

Tab. 3 – Programme émulant la communication SPI sur le port parallèle d'un PC. Le transfert de 8 bits sert à la fois à l'émission d'un octet (bus MOSI) et simultanément la réception (bus MISO) : dans le cas d'une lecture seule, un octet quelconque est émis afin de générer les 8 impulsions d'horloge sur le bus CK.

Ayant maîtrisé la communication, il nous faut nous plonger dans le protocole d'intialisation et de communication spécifique à la MMC. L'intialisation se fait en envoyant 80 impulsions d'horloge avec le signal de sélection SS# au niveau *haut* (*i.e.* désactivé). Ce résultat s'obtient avec la routine de la table 3 en émettant 10 fois la valeur 0xFF après avoir pris soin de mettre le signal SS# au niveau haut. Une fois cette première séquence d'initalisation transmise (table 4), il nous faut comprendre la notion de commande, les arguments nécessaires et les séquences d'acquittement.

Les notices [11] concernant les mémoires de type MMC mentionnent systèmatiquement l'existence de commandes CMDx. Cette nomenclature doit se lire de la façon suivante :

- x est donné dans les notices en décimal
- le mot de commande à transmettre à la MMC s'obtient par un OU logique entre 0x40 et le mot x converti en hexadécimal. Par exemple la commande WriteBlock CMD24 s'exécute en pratique en envoyant la commande 0x58=0x40|0x18.
- la commande est suivie de 4 octets d'argument puis d'un checksum dont nous allons rapidement oublier l'existence (voir plus bas),

```
char MMCgetResponse(){                  //Response comes 1-8bytes after command
  int i=0;              //the first bit will be a 0, followed by an error code
  char response;                            //data will be 0xff until response
  while(i<=8){
    response=SendByte(0xff);
    if(response==0x00) break;
    if(response==0x01) break;
    i++;
  }
  return response;
}

void SendCmd(char command, int argument, char CRC)         // always MSB first
{char *cmd;
 cmd=&argument;  // break 1 int in 4 bytes: ASSUMES this system uses 4 byte int
 SendByte(command|0x40);
 SendByte(cmd[3]); SendByte(cmd[2]); SendByte(cmd[1]); SendByte(cmd[0]);
 SendByte(CRC);
}

int InitMMC(void){                     //raise SS and MOSI for 80 clock cycles:
  int i;                                // SendByte(0xff) 10 times with SS high
  char response=0x01;

  RaiseSS();                               //initialization sequence on PowerUp
  for(i=0;i<=9;i++) SendByte(0xff);
  LowerSS();
  SendCmd(0x00,0,0x95);                 //Send Command 0 to put MMC in SPI mode
  if(MMCgetResponse()!=0x01) return false;       //Now wait for READY RESPONSE
  while(response==0x01){
    debug("Snd Cmd1 -- ");                                  // then send CMD1
    RaiseSS();
    SendByte(0xff);
    LowerSS();
    SendCmd(0x01,0x00ffc000,0xff);
    response=MMCgetResponse();
  }
  if(response==0x00) debug("RESPONSE WAS GOOD\n");
  RaiseSS();
  SendByte(0xff);
  debug("MMC INITIALIZED AND SET TO SPI MODE PROPERLY.\n");
  return true;
}
```

Tab. 4 – Implémentation de la phase d'initialisation sur port parallèle d'une MMC en mode SPI.

– la réponse de la MMC est 0xFF tant que la carte travaille (*i.e.* une lecture de 0xFF signifie qu'il faut redemander à la carte son état) et renvoie généralement 0x00 ou 0x01 pour acquitter une commande.

Ainsi, la première commande que nous devons transmettre est le passage du protocole 3 fils MMC au SPI, qui désactive notamment le calcul des checksums. Nous devons envoyer pour effectuer la commande CMD0 la séquence d'octets : 0x40 (commande CMD0) suivie de 4 fois l'octet 0x00 suivis finalement du checksum 0x95 (ici le checksum n'est pas encore désactivé mais précalculé : ce n'est qu'aprés acqittement de cette commande CMD0 que les checksums ultérieurs seront tous égaux à 0xFF).

La MMC acquitte la commande CMD0 en répondant 0x01. Ainsi, tant que la réponse de la MMC est 0xFF nous redemandons une nouvelle réponse (envoi d'une valeur quelconque par la fonction `SendByte()` et test de la valeur résultante) jusqu'à obtenir une valeur autre que 0xFF : une valeur de 0x01 indique le passage de la carte en mode SPI, sinon l'initialisation doit être recommancée.

Suit la commande CMD1 qui doit être transmise autant de fois que nécessaire pour que la MMC nous réponde 0x00 : la séquence précalculée est ici 0x41=0x40|0x01 suivi de 4 octets quelconques et finalement un checksum quelconque pris à 0xFF. Noter que ici il faut *ré-émettre* la commande CMD1 tant que la réponse n'est pas 0x00 (et pas simplement se contenter de redemander la réponse de la carte sans renvoyer la commande). L'acquittement de la carte par 0x00 conclue la phase d'initialisation de la MMC.

Une fois la MMC initialisée, il nous reste à y écrire et lire des données. Toute transaction avec la MMC se fait par défaut par blocs de 512 octets. Les accès se font de façon aléatoire en fournissant une adresse d'accès à la mémoire multiple de 512. La commande d'écriture est CMD24, la commande de lecture est CMD17. Chaque commande est suivie de 4 octets définissant l'adresse d'écriture du bloc, octet de poids fort en premier, multiple de 512. Cela signifie que l'octet de poids le plus faible d'adresse (celui transmis en dernier) est toujours nul tandis que l'octet précédent est égal à 0x02$\times N$, $N \in [0:7]$. Finalement l'octet de checksum est toujours égal à 0xFF.

Dans le cas d'une écriture la commande CMD24 – suivie des 4 octets d'adresse et de l'octet

de checksum – est suivie après confirmation de la commande par la MMC (transmission d'une réponse égale à 0x00) de l'émission de l'octet 0xFE, puis de 512 valeurs à stocker et finalement 2 octets quelconques qui normalement correspondent au checksum (égaux tous deux à 0xFF dans notre cas). Après cette transmission de donnée la MMC répond 0x00 tant qu'elle écrit les données en mémoire non-volatile, et finalement confirme le stockage des données en répondant par un octet finissant par 0x05.

Dans le cas d'une lecture, la procédure est légèrement différente : on envoie 0x51 (CMD17) suivie d'une adresse sur 4 octets telle que définie précédemment et conclue par un checksum égal à 0xFF. Cette fois la MMC nous répond 0x00 (acquittement de la commande) suivi de 0xFE (cette fois la MMC nous envoie cette valeur confirmant le transfert des données). Nous lisons alors 512 valeurs qui sont les données extraites de la mémoire non-volatile, suivies de 2 valeurs dont nous ne tenons pas compte qui sont les checksums.

Nous fournissons ici un exemple d'implémentation d'écriture (table 5), la lecture s'en déduisant facilement.

```
int MMCwriteBlock(char *Block, int address){//WRITE a BLOCK starting at address
  unsigned char busy,dataResp;int count;      //Block size is 512 bytes exactly
  LowerSS();                                                  //First Lower SS
  SendCmd(24,address,0xff);                             //Then send write command
  if(MMCgetResponse()==00){             //command was a success - now send data
    SendByte(0xfe);                                 //start with DATA TOKEN = 0xFE
    for(count=0;count<BLOCK;count++){SendByte(Block[count]);}   //now send data
    SendByte(0xff);                           //data block sent - now send checksum
    SendByte(0xff);
    do {dataResp=SendByte(0xff);} while (dataResp==0xff); // read DATA RESPONSE
    debug("Write: response: %d\n",(unsigned char)dataResp);
    do {busy=SendByte(0xff);} while(busy==0);  // a 0 indicates the MMC is BUSY
    dataResp=dataResp&0x0f;     //mask the high byte of the DATA RESPONSE token
    RaiseSS();
    SendByte(0xff);
    if(dataResp==0x0b)
      {debug("DATA WAS NOT ACCEPTED BY CARD -- CRC ERROR\n");return false;}
    if(dataResp==0x05) {debug("Done writing %d bytes\n",BLOCK);return true;}
    debug("Invalid data Response token: %d.",dataResp);return false;
 }
  debug("Command 24 (Write) was not received by the MMC.\n");
  return false;
}
```

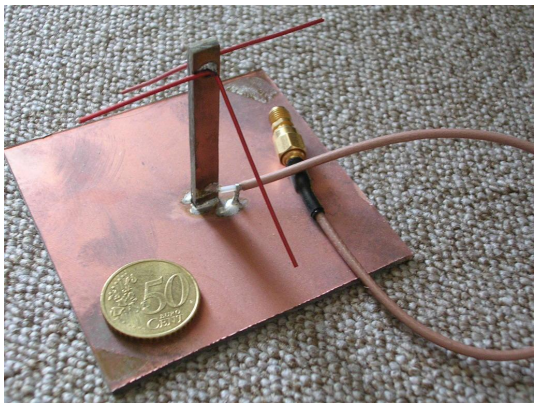
Tab. 5 – Implémentation de l'écriture dans une carte MMC.

L'ensemble de ces fonctions a été testé avec succès sur le port parallèle de PC ainsi que sur plusieurs microcontrôleurs supportant ou non le protocole SPI (ADuC814, 68HC908) sur une carte MMC de 16 MB. Nous avons tenté de maîtriser une carte SD dont la connectique est très similaire mais la phase d'initialisation est légèrement différente : ce travail est encore en cours à la date de rédaction de ce document. Nous n'avons pas encore eu l'opportunité de tester une MMCplus : ces cartes sont désormais disponibles pour des mémoire allant jusqu'à 1 GB de données, mais nécessitent l'achat d'une notice excessivement chère pour connaître l'ensemble du protocole de programmation avec notamment une bande passante considérable par un accès en parallèle aux 8 bits de données. Pour le moment nous nous contenterons d'écrire les données à enregistrer dans des blocs séquentiels, mais il est parfaitement envisageable d'implémenter un système de fichier tel que proposé dans les appareils photos numériques par exemple.

4 La réception et stockage des trames GPS

Tous les récepteurs GPS dont nous envisageons l'utilisation transmettent leurs données à faible débit par RS232. Notre première approche pour nous familiariser avec les trames transmises est donc de connecter le récepteur GPS sélectionné au port série d'un ordinateur fonctionnant sous GNU/Linux *via* un convertisseur de tension pour tranformer le signal asymétrique issu du récepteur GPS (3,3 ou 5 V) en signal symétrique (± 3 à ± 12 V). Les composants cités auparavant (MAX232 pour 5 V, MAX3232 pour 3,3 V, ou DS276) sont utilisés comme interface entre le récepteur GPS et le PC. Les terminaux tel que `minicom` permettent non seulement d'ajuster les paramètres de communication (4800 ou 9600 bauds, N81) mais aussi d'afficher et enregistrer dans un fichier les trames transmises. Nous avons tenté de réaliser quelques antennes de réception passives [12,

13] (Fig. 3) avant de conclure – malgré leur bon fonctionnement – que l'achat d'une antenne commerciale s'impose par souci d'encombrement et de robustesse (Fig. 4).



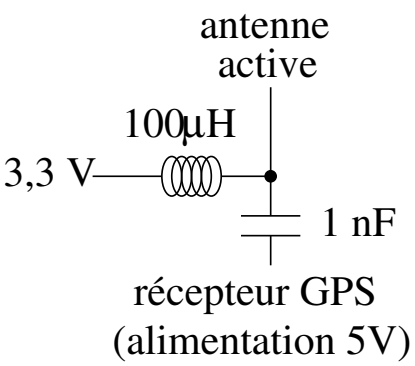


FIG. 3 – Gauche : photographie de l'antenne réalisée avec du matériel de récupération permettant de recevoir des signaux GPS, selon les plans founis à [12]. Une telle géométrie n'est cependant pas assez robuste pour nos applications mobiles, et n'inclut pas de préamplification pour une réception optimale des signaux. Droite : schéma du montage électronique permettant d'adapter une antenne prévue pour être alimentée en 3,3 V sur un récepteur fonctionnant sous 5 V. Le condensateur coupe le potentiel continu (DC) issu du récepteur GPS et ne laisse passer que le signal radiofréquence, tandis que l'inductance laisse passer la tension DC de polarisation tout en coupant la liaison entre l'alimentation et le signal radiofréquence. Les valeurs des composants sont données à titre indicatif.

Nos premiers tests avec divers récepteurs GPS nous donnent ainsi les résultats suivants :

- avec le Motorola Oncore [3], on obtient des suites de valeurs hexadécimales commençant par `@@` et contenant soit la position de l'antenne réceptrice, soit l'identification des satellites visibles et la qualité du signal qui en est reçu [14],
- avec le Laipac UV40 [4] des trames ASCII au format NMEA bien documentées [15] de la forme

```
$GPGSV,2,1,07,03,26,277,40,06,30,083,00,10,05,024,00,15,65,183,47*7E
$GPGSV,2,2,07,16,67,291,47,21,65,077,47,22,05,161,00,,,,*41
$GPVTG,285.2,T,,M,001.6,N,003.0,K*69
$GPGGA,184759,4714.9052,N,00559.3972,E,1,04,02.8,0294,M,,M,000,0000*7B
$GPGLL,4714.9052,N,00559.3972,E,184759,A*22
$GPRMC,184759,A,4714.9052,N,00559.3972,E,001.6,285.2,031005,,*18
$GPZDA,184759,03,10,2005,,*4B
$GPGSA,A,3,21,03,16,15,,,,,,,,,05.2,02.9,04.3*0A
```

Une différence majeure entre les deux types de sorties est que le NMEA nous fournit une trame en ASCII avec des coordonnées de la forme `degrés.minutes` (ici 47 degrés et 14.9052 minutes nord, 005 degrés et 59.3972 minutes est) tandis que le Motorola Oncore nous fournit une valeur binaire entre ± 324000000 en latitude et ± 648000000 en longitude (à convertir dans la plage $\pm 90^o$ et $\pm 180^o$ respectivement). Dans le cas de la trame NMEA, les positions sont fournies sous forme de degrés suivis de minutes d'angles et de leurs décimales : le tracé des courbes nécessite la conversion en degrés suivi de décimales (au lieu de degrés suivi de minutes d'angle), résultat que nous obtenons par exemple par les commandes Matlab/octave suivantes (en supposant le tableau à deux colonnes `gps` composé d'une première colonne des latitudes et une seconde colonne des longitudes tel que extrait

[3] acquis comme vente de surplus pour 36$/pièce auprès de Synergy (http://www.synergy-gps.com/) au 05 Juillet 2003

[4] acquis pour 104 euros/pièce auprès de Lextronic (http://www.lextronic.fr/laipac/uv40.htm) au mois d'Août 2005

de la trame GPGLL par `grep GPGLL fichier.gps | cut -d, -f2,4 | sed 's/,/ /g'`) : `deg=floor(g/100) ; minutes=(g-deg*100) ; g=deg+minutes/60 ; plot(g( :,2),g( :,1),'.') ;`.

FIG. 4 – Exemples de montage pour l'acquisition de données GPS lors de randonnées : le montage est installé dans une petite boite fermée protégeant l'électronique des chocs extérieurs (notamment risque d'appui sur l'interrupteur de RESET du microcontrôleur) tandis que l'antenne à support magnétique est placée en vue directe du ciel.

Ainsi, le récepteur GPS Motorola Oncore fournit non seulement la date et sa position une fois par seconde (trame `@@Ea`) mais en plus les éphémérides des satellites visibles depuis le sol (trames `@@Bb`). En plus d'analyser l'évolution de la trajectoire suivie par le récepteur, nous pouvons donc connaître le nombre de satellites visibles, leur élévation au dessus de l'horizon et le décalage Doppler de fréquence résultant. Le Motorola Oncore VP possède de plus une sortie 1 PPS (1 pulse par seconde) qui permet un asservissement excessivement précis d'un oscillateur local (résonateur à quartz stable en fréquence à court terme) sur cette référence de temps [19, 20].

La consommation totale de notre circuit, incluant un ADuC814, un MAX232, le Motorola GPS Oncore VP et une carte MMC de 16 MB est de 210 mA, soit une autonomie de l'ordre de 8 h avec nos batteries LiPo de 2000 mA.h.

5 Résultats : validation et dissémination des traces

Le montage constitué d'un GPS Oncore VP, un microcontrôleur ADuC814, une MMC de 16 MB alimenté par une batterie LiPo 2 Ah [16] (T2M Powerhouse [5] – disponible chez Euromodel à Paris, pour 89 euros/p) a été utilisé dans la plupart des configurations imaginables entre l'été 2005 et la date de parution de cet article : en voiture avec l'antenne fixée sur le toit (car ne fonctionnant pas derrière une vitre), en marche à pieds ou en jogging, ou fixé sur la façade d'un train (antenne verticale sur une surface métallique).

L'ensemble des traces ainsi acquises est présenté sur les figures 5, 6, 7, et 8. Cette dernière ne présente, à cette échelle, que les traces acquises en voiture et train tandis que les premières somment des traces acquises à pieds et en véhicules motorisés.

Un point fondamental dans l'exploitation de ces résultats, et éventuellement leur dissémination ultérieure dans des bases de données visant à mettre en commun de telles traces, est la vérification de la validité des points acquis, leur précision et leur reproductibilité. Nous nous sommes donc efforcés de vérifier que plusieurs traces prises à intervalles de temps allant de la journée à la semaine se recouvrent bien, et avons pris soin de comparer nos données aux cartes propriétaires disponibles. Par exemple, Fig. 10 démontre que toutes nos traces correspondent bien à des rues répertoriées dans les cartes fournies par `www.mapquest.com`. Ce site fournit, dans sa rubrique `Maps`, l'option très pratique ici d'afficher une carte routière localisée par ses coordonnées GPS (`Map by`

[5] Attention : ces batteries peuvent exploser ou prendre feu en cas de manipulation inappropriée. Lire attentivement la notice et les conditions d'utilisation, et ne recharger qu'avec un chargeur dédié aux accumulateurs LiPo. N'utiliser *en aucun cas* un autre chargeur, notamment NiCd, pour recharger ces batteries.

`Lat/Long`). Notre travail consiste alors à trouver un moyen simple et fiable de superposer nos traces à ces cartes obtenues sur le web.

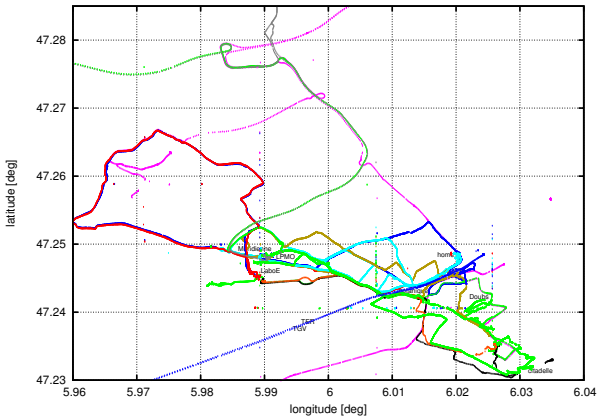


FIG. 5 – Tracés obtenus autour de Besançon.

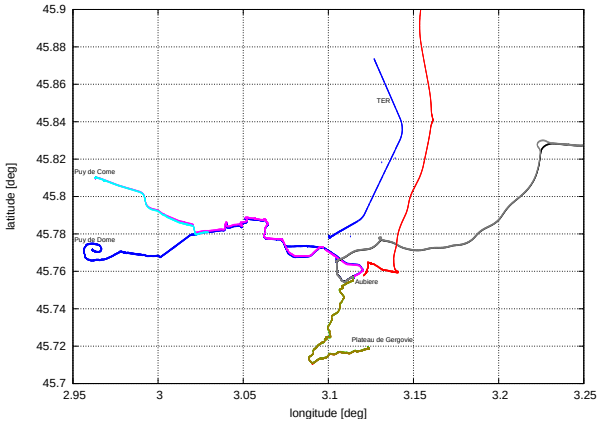


FIG. 6 – Tracés obtenus autour de Clermont-Ferrand.

Le PostScript fournit une méthode parfaite pour une telle application : ce langage simple et bien documenté [17] travaille sur des données vectorielles donc sans perte de précision quel que soit le taux d'agrandissement utilisé. Ainsi, les traces issues du GPS sont sauvées au format PostScript au moyen de `gnuplot`, `octave` ou `matlab`. D'autre part, les images au format GIF fournies par `mapquest` sont elles aussi converties au format PostScript par `giftopnm fichier.gif | ppmtopgm | pnmtops > fichier.ps`. Les étapes successives correspondent à une conversion de l'image GIF en image bitmap en tons de gris suivie d'une conversion de l'image en tons de gris en un format lisible par une interpréteur PostScript (`gs` dans notre cas). La concaténation des images se fait simplement par concaténation des fichiers puis effacement de la commande de conclusion du premier fichier `showpage` ainsi que de l'entête du second fichier (toutes les lignes d'entête commençant par %%). Ayant concaténé les images, il nous faut encore les translater afin de les superposer correctement : la commande `X Y translate` insérée entre les deux fichiers concaténés permet de translater la seconde image de `X Y` point par rapport à la première.

Ayant concaténé les images bitmap pour former une carte suffisamment précise et couvrant une surface suffisamment vaste, il nous reste à ajouter nos traces GPS : là encore nous nous contentons de concaténer le fichier obtenu par `gnuplot/octave/matlab` sans son entête en fin du fichier précédent. Le résultat est de dessiner les traces GPS au dessus des cartes issues de `mapquest`.

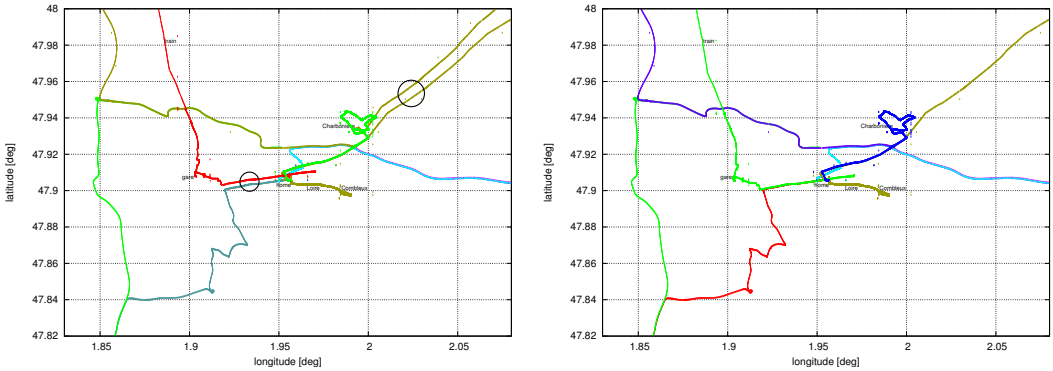


FIG. 7 – Tracés obtenus autour d'Orléans : à gauche avant corrections d'offsets systématiques sur des acquisitions complètes (les erreurs sont indiquées par des cercles en gras), à droite après correction manuelle (voir section 5).

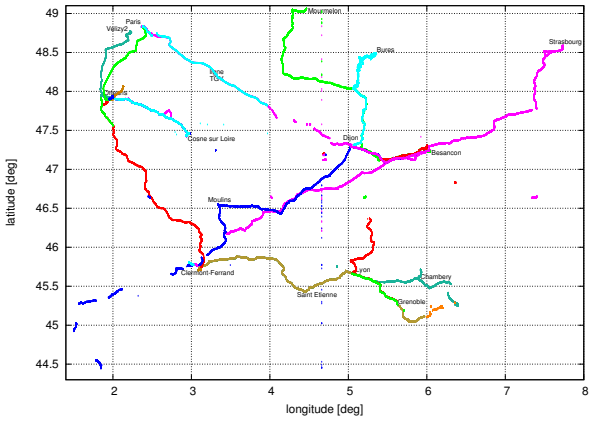


FIG. 8 – Tracés obtenus lors de parcours en France en automobile et en train.

Ayant constaté que ces deux ensembles de points sont bien orientés de la même façon (le nord vers le haut, donc pas de rotation nécessaire), la seule commande en plus du `X Y translate` que nous avons déjà cité plus haut et la commande d'homothétie : `U V scale` pour effectuer une homothétie d'un facteur `U` selon l'axe des abscisses et `V` selon les ordonnées. Muni de ces deux commandes, `scale` et `translate`, il nous reste à trouver le bon ensemble de paramètres pour superposer au mieux les traces GPS aux cartes bitmaps et obtenir une figure du type de celle présentée en Fig. 10.

Cette étape de validation des données nous a permis de constater (et corriger) un certain nombre d'artéfacts dans les données obtenues par le récepteur Motorola Oncore :

- une inexactitude des données au démarrage du récepteur (phase d'initialisation), généralement négligeable mais dans un cas si importante que les points ont dû être censurés (voir plus bas),
- des sauts soudains au cours d'une acquisition qui mettent quelques minutes à converger à nouveau vers une valeur raisonnable – ces effets peuvent être identifiés de façon automatique en calculant la dérivée des coordonnées acquises et seuiller. Nous avons notamment constaté un tel effet sur la mesure d'altitude,
- des sauts soudains de position qui se corrigent tout aussi soudainement (*i.e.* sans convergence lente comme dans le cas précédent) : ici un segment de trace se trouve simplement décalé par rapport aux traces adjacentes. Un cas extrême de cette situation a été l'acquisition depuis le

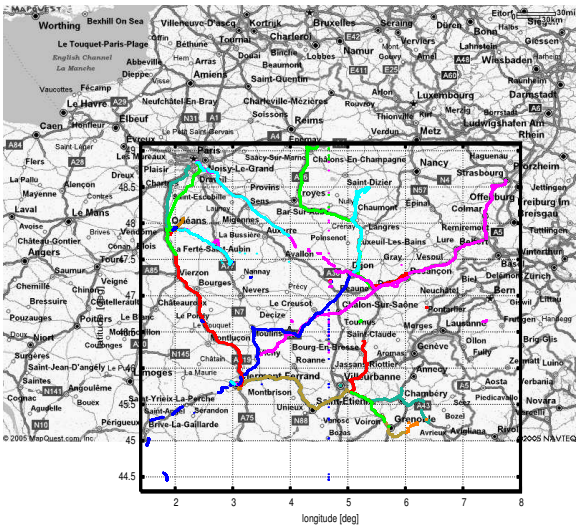


FIG. 9 – Superposition de cartes de la France obtenues sur www.mapquest.com et de traces GPS obtenues avec notre montage. L'accord semble généralement excellent, mais montre surtout le peu d'intérêt de superposer des traces aussi précises que celles issues d'un récepteur GPS avec une résolution aussi mauvaise. Les cartes avec un grossissement plus important telles que les Figs 6 et 5 semblent ainsi beaucoup plus intéressantes et font ressortir tout l'intérêt d'une méthode de localisation précise qui n'est pas exploitée ici.

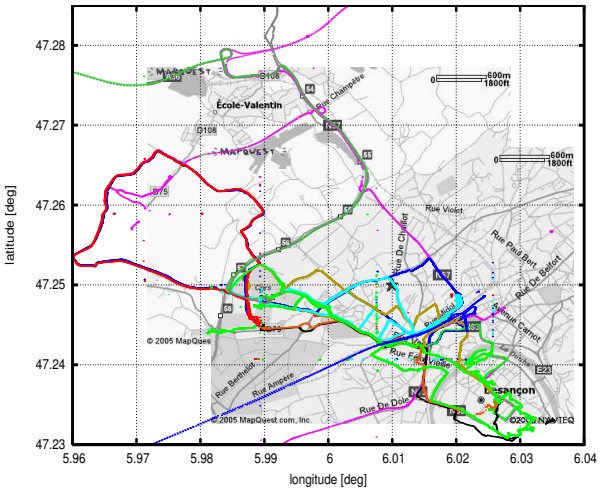


FIG. 10 – Superposition de cartes de Besançon obtenues sur www.mapquest.com et de traces GPS obtenues avec notre montage. L'accord semble généralement excellent, les quelques traces ne correspondant pas à des routes étant dues à des parcours à pieds sur des chemins non-carossables.

train Paris-Orléans tel que présentée sur la Fig. 7 dont les deux coordonnées étaient décalées de 0.0002^o et 0.0025^o en longitude et latitude respectivement sur *l'ensemble du parcours* (soit environ 11 m et 278 m respectivement, la seconde erreur étant clairement inacceptable). Cette erreur n'a été décelable qu'en comparaison des traces acquises antérieurement et de la superposition sur carte `mapquest`. Nous ne pouvons par expliquer cette erreur ni imaginer une méthode systématique de correction. C'est peut être ainsi que le Pôle Minatec (http://www.minatec.com/) – à notre connaissance situé à Grenoble – se retrouve en Suisse sur

www.upct.org.

La seule façon de corriger ces incertitudes sur les traces est l'accumulation de suffisamment de parcours *a priori* identiques pour lentement faire converger les coordonnées vers des valeurs représentatives.

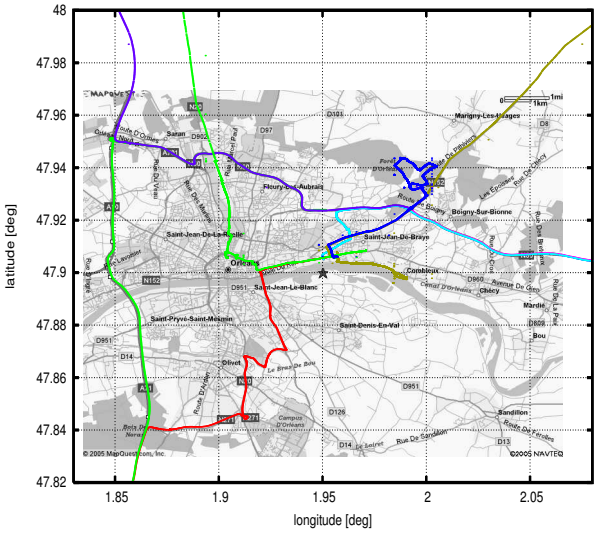


FIG. 11 – Tracés obtenus autour d'Orléans, superposés à une carte obtenue sur Mapquest : là encore l'accord entre les deux ensembles de données est excellent.

Un tel résultat présente deux intérêts :

- valider les résultats acquis, éventuellement compléter une carte propriétaire avec des traces inexistantes (mise à jour) ou qui n'ont pas lieu d'être (chemins piétons n'apparaissant pas sur une carte routière). Si le cas se présente, il peut apparaître nécessaire d'éliminer des traces incohérentes. En plus de 3 mois d'acquisitions, il nous a fallu censurer seulement les deux premières minutes d'une trace qui étaient évidemment fausses, probablement du fait d'une initialisation erronée du GPS qui a mis plus de temps que d'habitude pour faire converger son algorithme de localisation.
- estimer la quantité de travail restant pour obtenir une cartographie dense et pertinente d'une région.

Ces deux points nous semblent essentiels, le premier pour des questions évidentes de validité des points disséminés, la seconde car un travail bénévole sera d'autant plus efficace qu'il sera bien planifié par rapport aux données existantes et disponibles pour un usage personnel.

La dissémination des points acquis sur les bases de données des projets de cartographie opensource est le point qui nous a posé le plus de problème. Afin de communiquer avec les bases de données existantes, il nous faut adopter un format de données compréhensible par ces logiciels et suffisamment simple pour nous pour pouvoir écrire un convertisseur depuis les données brutes {date, latitude, longitude}. Nous avons choisi le format utilisé par `gpsman` (http://www.ncc.up.pt/gpsman/) pour des raisons de disponibilité du code source de ce logiciel opensource et du format de données qu'il attend (http://www.ncc.up.pt/gpsman/gpsmanhtml/manual/html/GPSMandoc_9.html), sa documentation sur le web et sa capacité à générer ensuite en principe un grand nombre de formats de sortie. En pratique, nous avons été capables de faire lire nos données par `gpsman`, mais le transfert d'un tel fichier vers www.upct.org s'est soldé par un échec. www.openstreetmaps.org quant à lui utilise un format XML (nommé GPX) que nous avons tenté de générer au moyen de `gpsbabel` (http://www.gpsbabel.org/) mais apparemment sans succès puisque là encore les fichiers soumis ont été rejetés. Il n'est cependant pas clair si le problème vient de nos formats de fichiers ou des serveurs des sites web. Une description plus complète des logiciels permettant une exploitation efficace des points acquis est disponible à la Ref. [18].

6 Applications et perspectives

Beaucoup d'applications quantitatives des données de position obtenues par GPS nécessitent une conversion d'une information angulaire en distance. Cette opération s'obtient en considérant que pendant longtemps la définition du mètre était donnée par la circonférence terrestre (ici supposée comme sphérique de circonférence 40000 km), et que la longueur d'un parallèle s'obtient par homothétie du cercle de l'équateur par $\sin(latitude)$. Nous déduisons donc la distance d_{lat} parcourue "parallèlement" à un méridien par $d_{lat} = 40000/360 \times \Delta_{lat}$ avec Δ_{lat} la variation angulaire de latitude entre deux points en degrés. Pour ce qui est de la direction selon un parallèle, la distance parcourue est $d_{lon} = 40000/360 \times \Delta_{lon} \sin(lat)$ avec lat la valeur moyenne de la latitude et Δ_{lon} la variation angulaire de longitude. Finalement la distance totale d parcourue s'obtient en considérant que les méridiens et parallèles sont orthogonaux : $d = \sqrt{d_{lat}^2 + d_{lon}^2}$ et sachant que notre GPS fournit un point par seconde, la vitesse en km/s se déduit par dérivation de d (fonction `diff()` sous Matlab/Octave). Le résultat de tels calculs appliqués aux trajets Paris-Orléans et Besançon-Paris en train sont présentés sur la figure 12.

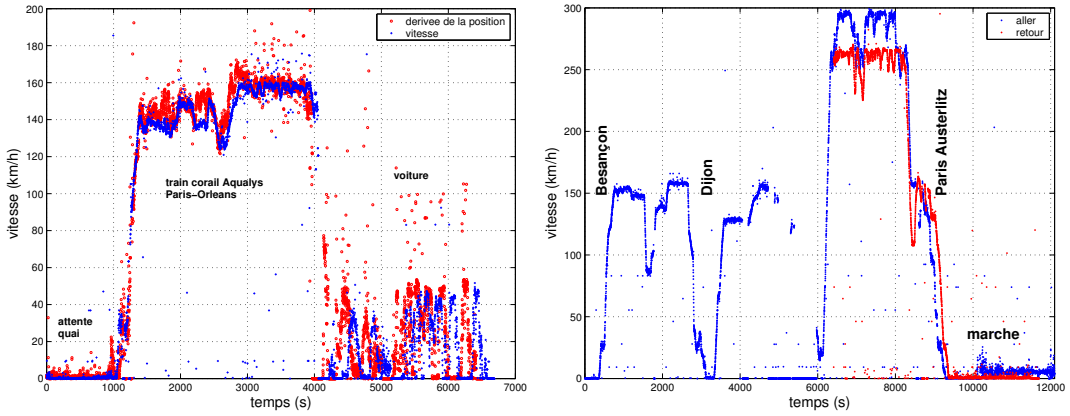


FIG. 12 – Gauche : mesure de vitesse d'un train Corail et comparaison entre la dérivée des positions acquises par le GPS et de l'information de vitesse fournie par le récepteur. Un bon accord est observé entre les deux quantités. Droite : mesure de vitesse d'un TGV sur un parcours rapide. Attention : les wagons de TGV étant conçus pour être totalement étanches, la seule façon de sortir une antenne GPS est par la porte. Ceci résulte en une probabilité de 50% de perdre l'antenne si la voie de débarquement se situe du mauvais côté à l'arrivée du train : il n'existe apparemment aucune autre façon que l'ouverture d'urgence pour ouvrir une porte de TGV côté voie.

Dans cette perspective nous imaginons un certain nombre d'autres applications de ce circuit tel que le suivi d'animaux domestiques suffisamment gros pour porter les batteries tels que des bovins. Le circuit – notamment la source d'énergie – n'est pas encore suffisamment petit pour être compatible avec des animaux aussi petits que les chats mais avec la baisse de consommation des récepteurs GPS (utilisation dans les téléphones portables notamment) une telle application sera probablement bientôt envisageable. Un nouveau récepteur GPS disponible chez Lextronic [6], à peine plus grand qu'une pièce d'un euro, nous approche de ce type d'application mais sa consommation relativement élevée ne résout par le problème de l'encombrement de la source d'énergie.

7 Conclusion

Nous avons présenté dans ce document la mise en œuvre d'un microcontrôleur basé sur un cœur de 8051, incluant les outils d'assemblage et de programmation fonctionnant sous GNU/Linux. Nous

[6] http://www.lextronic.fr/laipac/tf30.htm

avons par ailleurs décrit le protocole de programmation des MultiMediaCard afin d'acquérir la maîtrise d'une méthode de stockage de masse compatible et terme d'encombrement et de consommation avec une application embarquée.

Nous avons mis en pratique les connaissances acquises sur le microcontrôleur et la carte mémoire pour réaliser un système d'enregistrement de trames GPS permettant le calcul de la trajectoire suivie par le porteur du dispositif, et ce avec une résolution temporelle de la seconde. Nous avons appliqué ce montage à la cartographie visant à contribuer aux projets de base de données libres à des fins de navigation.

Le coût total de ce montage – excluant les batteries et le chargeur associé – est inférieur à 100 euros [7], et est principalement lié au prix du récepteur GPS et de son antenne. Il vise donc à étendre la population de contributeurs potentiels aux projets de cartographie en abaissant la somme à investir en terme de matériel. D'autre part, du fait de son architecture ouverte, ce projet offre de nombreuses autres perspectives, associées ou non au GPS dans le cadre de l'enregistrement de télémétrie plus générale ou de synchronisation d'horloges distantes à des fins d'interférométrie à grande base par exemple.

Le GPS en particulier offre un domaine riche en explorations supplémentaires : nous avons ici omis de décrire toutes les trames sur l'état des satellites qui donnent une information expérimentale sur des notions telles que le décalage Doppler associé au mouvement des satellites, la résolution de la position en fonction du nombre de satellites vus ou l'évolution du rapport signal sur bruit en fonction de l'élévation du satellite sur l'horizon.

Références

[1] www.openstreetmaps.org pour un projet anglais décrit par ses auteurs à http://wiki.whatthehack.org/index.php/OpenStreetMap

[2] www.upct.org pour un projet français décrit dans Linux Pratique **30** (Juillet/Août 2005), pp.6-7 et pp.8-9

[3] http://www.8052.com/tutorial.phtml est un exemple parmi d'autre de site web présentant ce cœur de microcontrôleur très largement utilisé.

[4] ftp://ftp.analog.com/pub/www/technology/dataConverters/microconverter/ADuC_code_V35.zip

[5] J.-M Friedt & S. Guinot, *Programmation et interfaçage d'un microcontrôleur par USB sous linux : le 68HC908JB8*, GNU/Linux Magazine Hors Série 23 (Nov./Déc 2005)

[6] "Micronconverter Technical Note uC004 – Understanding the Serial Download Protocol", disponible dans les Application Notes sur le site web de Analog Devices http://www.analog.com/en/prod/0,,762_0_ADUC814,00.html

[7] "Introduction au Coldfire 5282", GNU/Linux Magazine France **75** (Sept. 2005)

[8] Datasheet *HB288016MM1 MultiMediaCard 16 MByte* Rev 0.1 (Nov. 24, 1999), Hitachi, ou *HB28E016MM2* disponible à http://www.ulrichradig.de/site/atmel/avr_mmcsd/pdf/hitachi_hb28b128mm2.pdf

[9] http://www.ulrichradig.de/site/atmel/avr_mmcsd/pdf/MMCSDTimming.pdf

[10] *Application Note AN0501* disponible à www.mmca.org/compliance/buy_spec/AN_MMCA050419.pdf

[11] *SanDisk MultiMediaCard and Reduced-Size MultiMediaCard, Product Manual* disponible à http://www.sandisk.com/pdf/oem/manual-rs-mmcv1.0.pdf

[12] http://www.arrl.org/tis/info/pdf/0210036.pdf

[13] http://home.iae.nl/users/plundahl/antenne/patchant.htm

[7] 36$ de récepteur GPS Motorola Oncore, 31 euros d'antenne, 12 euros pour la carte MMC 128 MB et une dizaine d'euros de régulateurs de tension et connecteurs

[14] les documentations des Motorola Oncore se trouvent à http://www.synergy-gps.com/TR_Manual.html et plus particulièrement http://www.synergy-gps.com/TRM_ch6.pdf

[15] de nombreux sites web présentent le format des trames NMEA, par exemple http://www.commlinx.com.au/NMEA_sentences.htm ou http://www.gpsinformation.org/dale/nmea.htm

[16] G. Brocard, *La propulsion électrique des modèles réduits d'avions et de planeurs. Tome 1*, New Power Modélisme Édition (2004), pp.249-269

[17] http://jeff.cs.mcgill.ca/~luc/PSgeneral.html et en particulier le lien vers le Blue Book à http://www-cdf.fnal.gov/offline/PostScript/BLUEBOOK.PDF

[18] "Géographie libre", Linux Pratique **29** (Mai-Juin 2005), pp. 48-53

[19] W. Lewandowski, P. Moussay, P. Guerin, F. Meyer & M. Vincent, "Testing Motorola Oncore GPS receiver for time metrology", 11th European Frequency and Time Forum (1997), pp.493-497

[20] O.E. Rudnev, Y.S. Shmaliy, E.G. Sokolinskiy, A.Y. Shmaliy & O.I. Kharchenko, "Kalman filtering of a frequency instability based on Motorola Oncore UT GPS timing signals", Joint Meeting EFTF-IEEE IFCS (1999), pp. 251-254