Introduction à la programmation sur PlayStation Portable

J.-M. Friedt
Association Projet Aurore, Besançon
27 janvier 2007

Nous proposons de présenter les outils de développement libres pour la console de jeux PlayStation Portable. Après l'installation de la chaîne de cross-compilation et un premier exemple simple d'affichage graphique, nous interfacerons un récepteur GPS à la console et afficherons les coordonnées GPS courantes ou le trajet parcouru. L'objectif est de géolocaliser les points d'accès wifi détectés au moyen de l'interface de communication sans fil dont est équipée la console et de réaliser ainsi un outil de *wardriving* peu encombrant et présentant une autonomie de quelques heures.

1 Introduction

Le développement sur console de jeu portable est un moyen pratique de se familiariser avec la plupart des concepts du développement logiciel pour systèmes embarqués tout en profitant d'une plate forme de développement peu coûteuse (par rapport aux systèmes industriels dédiés), avec de nombreuses interfaces fournies par défaut (touches, écran, périphériques de communication) et une puissance de calcul digne des ordinateurs actuels [1, 2]. Cependant, les principaux fabriquants de consoles de jeux n'ont jamais été favorables au développement amateur de logiciels pour leur matériel et le programmeur désireux d'appliquer ses compétences à ces plates formes a dû tant bien que mal trouver les ressources nécessaires.

Nous allons ici présenter les outils de développement disponibles pour la PlayStation Portable (PSP) de Sony : cette console fournit un écran couleur de résolution acceptable (480×272 pixels), un processeur central basé sur une architecture MIPS [1] cadencé à 333 MHz [3], de nombreux périphériques (ports USB, série asynchrone, wifi) et un support de stockage de masse peu coûteux et accessible depuis un PC fonctionnant sous GNU/Linux (le Sony Memory Stick). Nous allons tenter d'utiliser au mieux ce matériel dans un objectif d'acquisition et de visualisation de données. Il ne sera pas ici question de développement de jeux qui nécessite des méthodes de programmation très particulières que l'auteur ne maîtrise pas.

Nous supposerons dans la suite de ce document que nous disposons d'un Sony Memory Stick et d'un adaptateur permettant de connecter une telle carte mémoire sur un PC, et éventuellement d'un câble de liaison PSP-USB.

2 Installation des outils de développement

Plusieurs langages permettant le développement d'applications pour la PSP (dits "homebrew" dans la communauté de développeurs PSP) sont disponibles : mentionnons ici notamment les langages interprétés tels que Lua ou Python. Nous nous intéresserons ici cependant directement à la solution la plus efficace, à savoir l'utilisation de `gcc` pour compiler du code C ou assembleur à destination du processeur MIPS de la PSP.

2.1 Mise à jour du firmware de la console

La première étape pour rendre une PlayStation Portable (PSP) utilisable par un développeur est de revenir à un firmware permettant l'exécution depuis le support de stockage de masse Memory Stick (MS). En effet, historiquement Sony n'avait pas pris soin d'empêcher l'exécution d'un programme depuis ce support et cette fonctionnalité inattendue sembla poser un problème de sécurité. Il s'en est suivi une séquence de mises à jour du firmware par Sony interdisant l'exécution de programme depuis le MS, et la mise à disposition par les développeurs indépendants de méthodes pour

[1] architecture modulaire très utilisée dans les systèmes embarqués tels que les points d'accès Broadcom : `http://www.linux-mips.org/wiki/Systems`

ramener sa PSP à un firmware acceptant de telles fonctionnalités. Le firmware de prédilection pour le développeur a pour version 1.5. Notre premier objectif est donc de reflasher le firmware de la PSP de sa version disponible à l'achat vers la version 1.5 qui autorise l'exécution d'un programme depuis le support de masse MS [4].

> *Attention : la PSP deviendra inutilisable si cette opération échoue. Il est fondamental de bien se documenter sur les dangers de flasher sa PSP afin d'en remplacer le firmware courant par celui de la version 1.5. L'auteur ne saurait être tenu responsable pour tout dommage résultant d'un échec de cette mise à jour*

La version du firmware sur une PSP achetée en France en Décembre 2006 est 2.01. Le passage du firmware 2.01 au firmware 1.5 est bien documenté à `http://gueux-forum.net/index.php?showtopic=125133`. Un MS d'au moins 32 MB est pour cela nécessaire. Cette mise à jour efface bien entendu tous les logiciels ajoutés par Sony entre la version 1.5 et la version fournie au moment de l'achat de sa console – notamment le logiciel d'affichage de pages web. Ces fonctionnalités peuvent être retrouvées en repassant provisoirement sur un firmware plus récent grâce à `devhook`. Ce logicel permet en effet, depuis un firmware 1.50, de charger en mémoire volatile un firmware plus récent et les applications associées, et ce jusqu'à la réinitialisation de la console (par sa mise hors tension par exemple). Nous avons utilisé à ces fins la version de `devhook` incluant toutes les mises à jour, version 0.44 disponible à `http://3mul.free.fr/v1/module.php?page=programme_detail&type_programme=emulateurs&i=63` qui a le bon goût de déjà inclure tous les firmwares à utiliser. Il n'est pas claire que les versions plus récentes que 1.50 des firmwares amènent des améliorations notables, si ce n'est de brider l'utilisateur dans les applications possibles de sa PSP : l'accès aux firmwares plus récents ne semble donc pas d'un intérêt remarquable si ce n'est de se rassurer sur la disponibilité des quelques logiciels ajoutés récemment par Sony.

La phase la plus dangereuse est passée : nous avons maintenant une PSP capable d'exécuter un logiciel depuis le support MS. Nous allons maintenant aborder les aspects d'installation des outils de compilation de nos programmes sous GNU/Linux à destination du processeur MIPS de la PSP (cross-compilation).

2.2 Installation de la chaîne de cross-compilation

La cross-compilation de code à destination du processeur MIPS de la PSP depuis l'ordinateur sur lequel tourne GNU/Linux (ici un processeur compatible Intel x86) nécessite la compilation du trio habituel `gcc` (version 4.0.2), `binutils` (version 2.16.1) et `newlib` (version 1.13.0). Les développeurs pour PSP ont considérablement simplifié le travail en fournissant un script, `toolchain.sh` dans l'archive `psptoolchain-20060120.tgz` disponible à `http://ps2dev.org/psp/Tools`, qui se charge de récupérer les bonnes versions des codes source, leur appliquer les patchs nécessaires pour supporter l'architecture PSP, et de les compiler avec les bonnes options – principalement `--target=psp`. Une fois `psp-gcc` installé dans son emplacement par défaut `/usr/local/pspdev/bin`, il ne nous reste plus qu'à compiler le PSPSDK (obtenu par CVS auprès de `svn://svn.pspdev.org/psp/trunk/pspsdk`) afin d'obtenir tous les outils nécessaires à la génération de binaires prêts à être exécutés depuis un firmware 1.5 sur la PSP. Cette opération est assez lourde puisqu'elle nécessite entre 32 et 64 MB de RAM et environ 1,5 GB d'espace disque.

L'accès à la console de jeu se fera soit après installation de `iRShell` disponible à `http://dl.qj.net/index.php?pg=12&fid=11531&catid=190` qui permet d'accéder à la PSP comme à une clé de stockage de masse sur port USB (Fig. 1), soit *via* le MS.

3 Premier exemple : la fractale de Mandelbrot

De nombreux tutoriaux sont disponibles sur le web, et ceux qui semblent les plus pertinents sont disponibles à `http://www.psp-programming.com/tutorials/`. En nous inspirant de ces présentations, nous allons commencer par développer une petite application simple affichant sur

FIG. 1 – Connexion de la PSP à un ordinateur portable sous Debian GNU/Linux sur lequel est installé le PSP SDK : la console apparaît comme un support de stockage de masse sur le bus USB.

l'écran de la PSP la fractale de Mandelbrot (accès au mode graphique) et sauvant sur MS le résultat du calcul pour exploitation ultérieure.

La fractale de Mandelbrot [5] est définie comme l'ensemble des points c dans le plan complexe pour lesquels l'itération de la suite $z_{n+1} = z_n^2 + c$ ($z_0 = c$) ne diverge pas. Nous affichons pour chaque point c le nombre d'itérations, borné à un maximum de 16, au bout duquel la suite a été identifée comme divergente (*i.e.* la couleur du point c est n lorsque $|z_n| > 2$, pour $n < 17$ – l'ensemble des points blancs sur la Fig. 2 présente donc les valeurs de c pour lesquelles la suite n'a pas divergé au bout de 16 itérations).

Le Makefile associé au programme (Tab. 1) est présenté dans le Tab. 2 : il s'appelle, pour générer un exécutable pour firmware 1.5, par la commande `make kxploit` qui génère deux sous répertoires, `mandel` et `mandel%` dans l'exemple qui nous intéresse ici (le nom de l'exécutable présenté par la PSP et des sous répertoires associés sont définis par la variable `PSP_EBOOT_TITLE` du Makefile). Ces deux répertoires sont placés sur le MS dans le sous répertoire `PSP/GAME` afin d'être reconnus lors de leur exécution sur la PSP. Si plusieurs projets sont placés dans le même sous répertoire, on prendra soin avant chaque nouvelle compilation d'effacer les objets et le fichier `PARAM.SFO` qui contient les paramètres de génération de l'exécutable `EBOOT.PBP` inclus dans ces répertoires.

Nous utilisons pour l'accès au mode graphique une librairie `graphics.h` disponible dans les exemples de `http://www.freewebs.com/jason867/psphomebrew.htm`. Le principal intérêt de ce code réside dans la mise en évidence de l'adresse de la mémoire vidéo et la procédure à suivre pour y écrire (fonction `PlotPixel()`) ou en lire le contenu pour effectuer une capture d'écran.

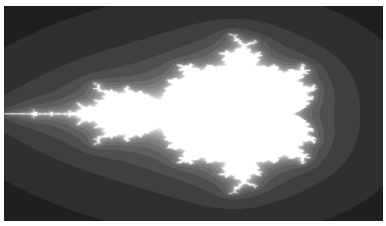

FIG. 2 – Capture d'écran du résultat de l'exécution du programme présenté au Tab. 1.

Nous constatons dans cet exemple que l'appel aux fonctions du système d'exploitation exécuté par la PSP implique un certain nombre de changements de nomenclature par rapport à l'ANSI C classique. Par exemple l'ouverture de fichier habituelle par `fopen()` est remplacée par `sceIoOpen()`

```
// 29/12/2006
// http://psp.jim.sh/pspsdk-doc/

#include <pspkernel.h>
#include <pspctrl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "graphics.h"

PSP_MODULE_INFO("mandel JMF", 0x1000, 1, 1);
PSP_MAIN_THREAD_ATTR(0);

#define xmin -2.0
#define xmax 1.0
#define ymin -1.5
#define ymax 1.5

void nextname(char *base)
{char s[52];
 int fcpt=0;
 SceUID f;
 do {sprintf(s,"ms0:/%s%d.dat",base,fcpt);
     f=sceIoOpen(s,PSP_O_RDONLY,0777);
     if (f>0) {sceIoClose (f);fcpt++;}         // f>=3 if file exists
 } while (f>0);
 sprintf(base,"%s",s);
}

int main(void)
{
  SceCtrlData pad;
  int cpt=0,i,ix,iy;
  double x,y,xt,yt,tmp;
  SceUID f;
  char s[42];

  SetupCallbacks();                          // ajout anti-freeze (cf web)
  InitializeGraphics(1); //  2 bytes/pixel && 512 instead of 480
//  memset((char*)0x44000000,0,512*272*2);   // blank screen
  sprintf(s,"mandel");nextname(s);
  f=sceIoOpen(s,PSP_O_WRONLY|PSP_O_CREAT,0777);
  for (iy=0;iy<272;iy++)
    for (ix=0;ix<480;ix++) {
      cpt=0;
      x=(xmax-xmin)/480.*(double)ix+xmin;
      y=(ymax-ymin)/272.*(double)iy+ymin;
      xt=x;yt=y;
      do {tmp=xt*xt-yt*yt+x;yt=2.*xt*yt+y;xt=tmp;cpt++;}   // z -> z^2+c
         while (((xt*xt+yt*yt)<4)&&(cpt<16));    // R=sup(|c|,2)
      if (cpt==0) cpt=1;
      PlotPixel(ix,iy,cpt*16-1,cpt*16-1,cpt*16-1);
      sceIoWrite(f,&cpt,1);
  }
  PlotPixel(1,1,255,255,255);
  sceIoClose(f);
  do {sceCtrlReadBufferPositive(&pad, 1); sceDisplayWaitVblankStart();}
     while (!(pad.Buttons & PSP_CTRL_CROSS));
  sceKernelExitGame();
  return 0;
}

////////////////////////////////////////////////////////
// http://www.psp-programming.com/tutorials/c/lesson02.htm
////////////////////////////////////////////////////////
/* Exit callback */
int exit_callback(int arg1, int arg2, void *common) {
 sceKernelExitGame();return 0;
}

/* Callback thread */
int CallbackThread(SceSize args, void *argp) {
 int cbid;

 cbid = sceKernelCreateCallback("Exit Callback", exit_callback, NULL);
 sceKernelRegisterExitCallback(cbid);
 sceKernelSleepThreadCB();
 return 0;
}

/* Sets up the callback thread and returns its thread id */
int SetupCallbacks(void) {
 int thid = 0;

 thid=sceKernelCreateThread("update_thread",CallbackThread,0x11,0xFA0,0,0);
 if(thid >= 0) {sceKernelStartThread(thid, 0, 0);}
 return thid;
}
```

Tab. 1 – Exemple de programme affichant la fractale de Mandelbrot et stockant sur MS les résultats du cacul. Aucune optimisation n'a été incluse dans ce code – qu'il s'agisse de l'affichage ou de l'algorithme de calcul – afin de le garder aussi lisible que possible : la durée d'exécution du calcul est de 37 secondes.

```
# make -f simple_rs232.Makefile kxploit

TARGET = mandel
OBJS = mandel.o

USE_PSPSDK_LIBC = 1

INCDIR =
CFLAGS = -O2 -G0 -Wall
CXXFLAGS = $(CFLAGS) -fno-exceptions -fno-rtti
ASFLAGS = $(CFLAGS)

LIBDIR =
LDFLAGS =


EXTRA_TARGETS = EBOOT.PBP
PSP_EBOOT_TITLE = mandel_JMF

PSPSDK=$(shell psp-config --pspsdk-path)
include $(PSPSDK)/lib/build.mak

LIBS += -lpsphprm_driver -lpspkernel -lpsppower -lm -lpng -lz -lc -lpspkernel
# -lc *doit* etre avant -lpspkernel
```

Tab. 2 – Fichier de configuration Makefile associé au programme présenté au Tab. 1.

avec les mêmes arguments, et `fclose()` est remplacé par `sceIoClose()`. Le reste du code suit une syntaxe tout a fait classique par ailleurs.

Une alternative pour l'affichage en mode texte d'informations sur la PSP est l'utilisation du mode Debug : initialisé par la procédure `pspDebugScreenInit();`, il permet d'afficher des caractères ASCII au moyen de la fonction `pspDebugScreenPrintf()` qui accepte une liste d'arguments identique à celle dont nous avons l'habitude avec `printf()`. Nous utiliserons plus bas ce mode pour afficher les trames NMEA issues d'un récepteur GPS connecté à la PSP.

4 Communication par le port série asynchrone

Nous allons dépasser le cadre des tutoriaux sus-cités pour aborder les aspects d'accès aux ports de communication de la PSP, aspect le plus intéressant pour faire interagir la console avec le monde qui l'entoure.

Nous avons présenté à plusieurs reprises dans ces pages [6, 7] des développements autour de récepteurs GPS OEM de coût raisonnable au vu de leurs performances. Cependant, un point délicat avec l'enregistrement en aveugle de trames GPS au cours d'un trajet est l'incapacité de vérifier en temps réel le bon fonctionnement du logiciel d'acquisition : rien n'est plus frustrant que de rentrer d'une longue randonnée et de se rendre compte qu'un élément du circuit d'acquisition n'a pas fonctionné correctement. Nous allons ici utiliser la PSP comme écran d'affichage des trames acquises, comme support de stockage de masse non volatile de secours pour conserver les traces du trajet, pour finalement combiner ces fonctionnalités avec l'acquisition géolocalisée de quantités physiques (dans le cas qui nous intéressera plus bas, la puissance radiofréquence reçue depuis des points d'accès wifi).

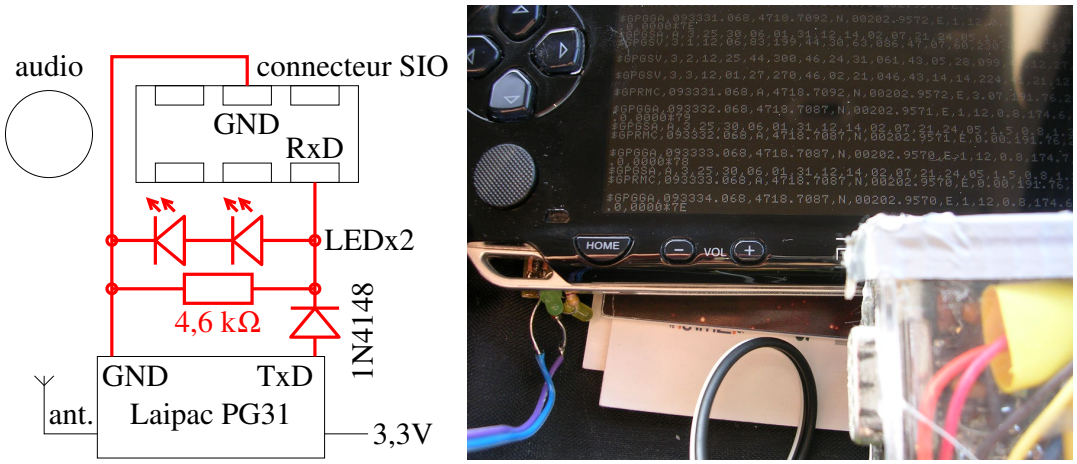

FIG. 3 – Gauche : montage connectant un GPS à une PSP. Les composants passifs ont pour vocation de protéger la PSP contre tout signal indésirable (la diode 1N4148 élimine tout risque d'appliquer un potentiel négatif sur la broche RxD, tandis que les deux LEDs en série limitent le potentiel appliqué à RxD à environ 3,4 V. La résistance en parallèle avec les deux LEDs permet alors au potentiel de RxD de redescendre lors du passage d'un bit 1 à un bit 0). Nous avons pu connecter un récepteur GPS Laipac PG31 (alimenté en 3,3 V et communiquant donc sur un signal haut à cette tension) sans composant passif de protection. Noter qu'un MAX(3)232 comporte non seulement une pompe de charge générant la tension négative nécessaire au protocole RS232 mais aussi des inverseurs : il ne suffit donc pas de connecter une diode pour faire communiquer un PC avec la PSP mais en plus un inverseur. Droite : utilisation de ce montage pour afficher la position au cours d'un trajet en voiture.

Le connecteur à 6 broches situé à côté de la prise de l'écouteur audio (en bas à gauche de la PSP) contient tous les signaux pour permettre une communication asynchrone compatible avec la norme RS232 [8]. Les niveaux des signaux sont entre 0 V et 2,5 V nominaux, compatibles avec un signal haut à 3,3 V. Nous allons connecter un récepteur GPS Laipac PG-31 [2]. La connection ne nécessite aucune précaution particulière bien que nous ayons choisi de placer deux LEDs afin de protéger le port série de la PSP de tout risque de surtension (une diode zener ferait le même effet).

Une fiche capable de s'insérer dans l'emplacement prévu à côté du connecteur audio doit être réalisée : nous gravons pour ceci un circuit imprimé double face (epoxy FR4 de 1,6 mm d'épaisseur) avec 3 pistes de chaque côté, espacées de 1,5 mm et de 1 mm de largeur. Le lecteur n'ayant pas accès au matériel nécessaire pour graver un circuit imprimé pourra détruire une carte PCI pour en récupérer trois broches du connecteur prévu pour s'enficher dans une carte mère de PC (Fig. 5).

[2] disponible auprès de Lextronic, `www.lextronic.fr`, pour 83 euros TTC : `http://www.lextronic.fr/laipac/`

```
// 26/12/2006
// http://psp.jim.sh/pspsdk-doc/

#include <pspkernel.h>
#include <pspdebug.h>
#include <pspdisplay.h>
#include <pspsdk.h>
#include <pspctrl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <psppower.h>
#include <math.h>
#include <png.h>
#include "graphics.h"

PSP_MODULE_INFO("GPS JMF", 0x1000, 1, 1); // 1000 = set kernel mode
PSP_MAIN_THREAD_ATTR(0);

#define printf pspDebugScreenPrintf

void nextname(char *base)
{char s[52];
 int fcpt=0;
 SceUID f;
 do {sprintf(s,"ms0:/%s%d.dat",base,fcpt);
     f=sceIoOpen(s,PSP_O_RDONLY,0777);
     if (f>0) {sceIoClose (f);fcpt++;}  // f>=3 if file exists
 } while (f>0);
 sprintf(base,"%s",s);
}

int main(void)
{
  SceCtrlData pad;
  int bauds[7]={1200,2400,4800,9600,19200,38400,57600};
  int cpt=2,ch,fcpt=0,flag=0,ilat,ilon,i;
  double flat,flon,oflat=0.,oflon=0.,mag=128.,degla,deglo;
  char sauv=0,s[52],lat[16],lon[16],txt=1;
  SceUID f,scr;

  SetupCallbacks();                         // ajout anti-freeze (cf web)
  pspDebugScreenInit();                     // init text mode
  pspDebugScreenSetTextColor (0xffffffff);  // White
  pspDebugScreenPrintf("[] and 0 to change baud rate, ^ to toggle save\n");
  pspDebugScreenPrintf("UP to toggle txt/graphic mode, X to validate and quit");
  pspDebugScreenSetXY(1,6);
  pspDebugScreenPrintf("Select: clear screen\n");
  pspDebugScreenPrintf("Start: dump screen\n");
  pspDebugSioInit();                        // init RS232
  while (1) {
     pspDebugScreenSetXY(1,3);
     pspDebugScreenPrintf("baud=%d  \tsave=%d \ttxt=%d ",bauds[cpt],sauv,txt);
     sceCtrlPeekBufferPositive(&pad, 1);
     if (pad.Buttons & PSP_CTRL_SQUARE) {cpt++;if (cpt>6) cpt=0;}
     if (pad.Buttons & PSP_CTRL_CIRCLE) {cpt--;if (cpt<0) cpt=6;}
     if (pad.Buttons & PSP_CTRL_TRIANGLE) {sauv=1-sauv;}
     if (pad.Buttons & PSP_CTRL_UP) {txt=1-txt;}
     if (pad.Buttons & PSP_CTRL_CROSS) {break;}
     sceKernelDelayThread(200000); // delay 200 ms = debounce
  }
  printf("\n");
/*
// write dummy file
  f=sceIoOpen("ms0:/gps0.dat",PSP_O_WRONLY|PSP_O_CREAT,0777);
  for (i=32;i<127;i++) {sceIoWrite(f,&i,1);printf("%c",i);}
  sceIoClose(f); printf("\n");
// read dummy file
  ch=0;
  f=sceIoOpen("ms0:/gps0.dat",PSP_O_RDONLY,0777);
  do {res=sceIoRead(f,&i,1);printf("%c",i);ch++;} while ((res!=0)&&(ch<100));
  sceIoClose(f); printf("\n");
*/
// let's start
  if (sauv==1) {sprintf(s,"gps"); nextname(s); // returns in s the new filename
                f=sceIoOpen(s,PSP_O_WRONLY|PSP_O_CREAT|PSP_O_APPEND,0777);}

  pspDebugSioSetBaud(bauds[cpt]);
  printf("\nPSP GPS JMF test");
  if (sauv==1) pspDebugScreenPrintf(": saving in %s\n",s); else
               pspDebugScreenPrintf("\n");
  sceKernelDelayThread(1000000);  // delay 1 s

  if (txt==0) {
    InitializeGraphics(1); //  2 bytes/pixel && 512 instead of 480
    memset((char*)0x44000000,0,512*272*2);   // blank screen
    }
  cpt=0;fcpt=0;
  while(1) {
           ch = pspDebugSioGetchar(); // get new RS232 char
           if ((ch >= 0) && (ch != '\r')) {
         if (sauv==1) {sceIoWrite(f,&ch,1);} // save in File
                if (txt==1) {pspDebugScreenPrintf("%c", ch);} // print
                if (((ch=='G')||(ch=='P'))||(ch=='A')||(flag>99))) flag++;
                                                          else flag=0;
                if (flag==5) {flag=100;fcpt=0;}                // on a eu GPGGA
                if ((flag>112) && (flag<122)) {lat[flag-113]=ch;}
                if ((flag>124) && (flag<135)) {lon[flag-125]=ch;}
                if (flag==135) {                          // fini parsing $GPGGA
                        lat[9]=0;lon[10]=0; // end of string
                        flat=atof(lat);flon=atof(lon);
degla=floor(flat/100.)*100.;
deglo=floor(flon/100.)*100.;
flat=degla+(flat-degla)/60.*100.; // 0..60 -> 0..100 lat
flon=deglo+(flon-deglo)/60.*100.; // 0..60 -> 0..100 lon
                        if ((oflat==0.) && (oflon==0.))
                           {oflat=flat;oflon=flon;}   // init pos
ilat=(int)(floor((flat-oflat)*mag))+136;
ilon=(int)(floor((flon-oflon)*mag))+240;
                        if (txt==1)
         {printf("\n -> %lf/%lf %lf/%lf mag=%lf\n",flat,oflat,flon,oflon,mag);}
                        else
                          if ((ilat>0) && (ilat<272) && (ilon>0) && (ilon<480))
                            PlotPixel(ilon,271-ilat,255,255,255);
                        flag=0;}
           }
           sceCtrlPeekBufferPositive(&pad, 1);   // DON'T wait VSync
           if (pad.Buttons & PSP_CTRL_CROSS) {break;    }
           if (fcpt==0) {
               if (pad.Buttons & PSP_CTRL_LEFT)   {oflon-=10.;fcpt=1;}
               if (pad.Buttons & PSP_CTRL_RIGHT)  {oflon+=10.;fcpt=1;}
               if (pad.Buttons & PSP_CTRL_UP)     {oflat+=10.;fcpt=1;}
               if (pad.Buttons & PSP_CTRL_DOWN)   {oflat-=10.;fcpt=1;}
               if (pad.Buttons & PSP_CTRL_SQUARE) {mag*=2.;fcpt=1;}
               if (pad.Buttons & PSP_CTRL_CIRCLE) {if (mag>1) mag/=2.;fcpt=1;}
               if (pad.Buttons & PSP_CTRL_SELECT) {
                   memset((char*)0x44000000,0,512*272*2); // blank screen
                   oflat=0.;oflon=0.;}
               if (pad.Buttons & PSP_CTRL_START) {
                   sprintf(s,"dump"); nextname(s);  // returns the new filename
                   scr=sceIoOpen(s,PSP_O_WRONLY|PSP_O_CREAT|PSP_O_APPEND,0777);
                   sceIoWrite(scr,lat,9);                    // dump screen
                   sceIoWrite(scr,lon,10);
                   for (i=0;i<272;i++) {
                      sceIoWrite(scr,(char*)(0x44000000+2*i*512),480*2);// line
                      PlotPixel(1,i,155,155,155);}
                   sceIoClose(scr);
                   // screenshot("ms0:/d.png");
               }
           }
  }
  if (sauv==1) sceIoClose(f);
  if (txt==1) pspDebugScreenPrintf("Good bye");
  sceKernelExitGame();
  return 0;
}

////////////////////////////////////////////////////////
// http://www.psp-programming.com/tutorials/c/lesson02.htm
////////////////////////////////////////////////////////
/* Exit callback */
int exit_callback(int arg1, int arg2, void *common) {
 sceKernelExitGame();return 0;
}

/* Callback thread */
int CallbackThread(SceSize args, void *argp) {
 int cbid;

 cbid = sceKernelCreateCallback("Exit Callback", exit_callback, NULL);
 sceKernelRegisterExitCallback(cbid);
 sceKernelSleepThreadCB();
 return 0;
}

/* Sets up the callback thread and returns its thread id */
int SetupCallbacks(void) {
 int thid = 0;

 thid=sceKernelCreateThread("update_thread",CallbackThread,0x11,0xFA0,0,0);
 if(thid >= 0) {sceKernelStartThread(thid, 0, 0);}
 return thid;
}
```

TAB. 3 – Exemple de programme permettant de sélectionner la vitesse de communication du GPS sur son port RS232 (4800 bauds par défaut pour le Laipac PG-31) puis d'afficher les trames NMEA brutes issues du récepteur, ou de traiter ces trames pour en extraire latitude et longitude pour restituer les informations de façon graphique.

Un exemple de code minimaliste permettant de capturer les trames sur le port série et de

`tf30.htm`

les afficher est disponible à `http://forums.ps2dev.org/viewtopic.php?t=6294`. On y trouve l'initialisation de la vitesse de transfert par la commande `pspDebugSioSetBaud(38400);` que nous adaptons à nos propres besoins.

Nous observons cependant que ce code contient une boucle de taille prédéfinie qui ne saurait convenir pour une application générale. Nous allons donc remplacer cette boucle par une condition de boucle infinie (`while (1) { ...}`) et conditionner la sortie de cette boucle par l'appui d'un bouton sur la PSP. Cependant, l'utilisation de la fonction bloquante d'interrogation des touches `sceCtrlReadBufferPositive()` pour détecter l'appui d'une touche que nous associerions à l'arrêt du programme est trop lente et nous fait perdre des caractères. La solution qui nous a semblé appropriée est l'appel à la fonction non-bloquante `sceCtrlPeekBufferPositive()` qui – n'étant pas bloquante jusqu'au prochain rafraichissement de l'écran – nous permet d'interrompre proprement le programme (par appel à la fonction `sceKernelExitGame();`) sans perdre de caractère transmis par le port série. Nous profiter de la gestion de l'appui des touches pour proposer la translation de l'origine de la carte affichée à l'écran, l'effacement ou la capture d'écran (Fig. 4).

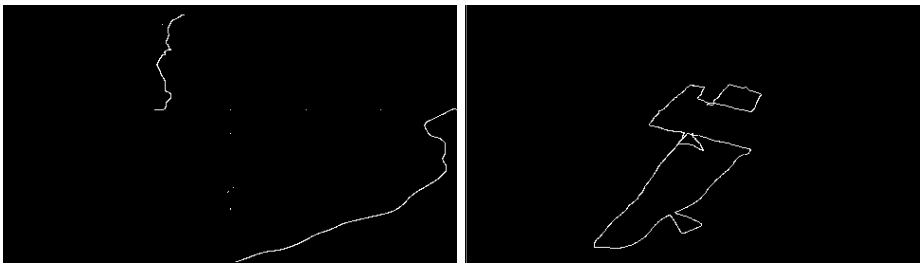

FIG. 4 – Captures d'écrans obtenues lors de l'acquisition de traces en voiture (gauche : du péage de Saint Arnould au sud de Paris au bois de Boulogne à Paris) et à pieds (droite : Jardin de Bagatelle, Paris). L'image de gauche a été recentrée en cours d'acquisition au moyen des flèches de direction à mi-hauteur alors que la trace allait sortie de l'écran : les points blancs indiquent le pas de translation lors du recentrage de la trace.

Il apparaît à l'usage que les fichiers créés pour les sauvegardes sur MS ne sont créés qu'au moment de leur fermeture. Cela signifie que les données accumulées ne sont réellement stockées que lors de la fermeture du fichier, *i.e.* à la fin d'une séance d'acquisition de données. Aucune solution satisfaisante n'a pour le moment été identifiée pour conserver les données en cours d'utilisation du programme, avant de quitter proprement et de fermer le fichier (`sceIoClose()` qui se comporte exactement comme l'habituel `fclose()`). Il ne semble pas exister de fonction pour vider les tampons (fonction `fflush()` classiquement) sur PSP. L'ensemble des fonctions spécifiques au PSP SDK est décrit à `http://psp.jim.sh/pspsdk-doc/`.

Nous sommes donc capables, à ce stade d'avancement, de capturer des informations sur le port série et de les afficher soit en mode texte, soit en mode graphique (Fig. 4), sur l'écran de la PSP. Il nous reste maintenant à encoder dans la couleur de chaque point l'évolution spatiale d'une quantité physique mesurée au cours du trajet.

5 Utilisation de l'interface wifi

Nous avons donc démontré notre capacité à interfacer un récepteur GPS à une PSP pour afficher le trajet parcouru, et éventuellement de conserver en mémoire non volatile les latitudes et longitudes échantillonnées toutes les secondes. La PSP est par ailleurs équipée d'une interface wifi : quoi de plus naturel donc que de combiner la fonction de géolocalisation avec la capacité à écouter et identifier tous les points d'accès wifi à proximité (*wardriving* et *warwalking*) [9].

La mise en œuvre de l'interface wifi de la PSP n'est pas aisée et nécessite l'appel à des fonctions du firmware fourni par Sony et identifiées par *reverse engineering* [10]. La structure de données

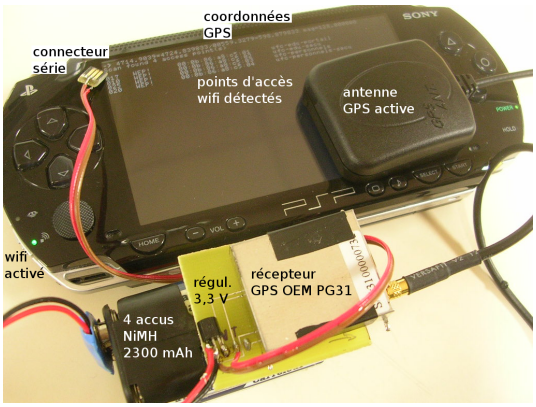

FIG. 5 – Montage minimal pour géopositionner les points d'accès wifi au moyen de la PSP. Dans ce cas le connecteur entre le GPS et la PSP a été réalisé au moyen d'un bout de carte PCI (connecteur s'enfichant normalement sur la carte mère d'un PC).

retournée par ces fonctions contient – pour un maximum de 32 points d'accès – des informations telles que les identifiants (BSSID et nom du point d'accès), la puissance reçue (RSSI, *Received Signal Strength Indication*) ou le mode d'encryption des trames. L'excellent exemple mis à disposition avec le PSPSDK dans le répertoire `samples/wlanscan_elf` nous fournit en fait pratiquement toutes les fonctionnalités dont nous avons besoin, et il nous suffit de joindre ce programme avec celui présenté précédemment pour réaliser une application de cartographie des points d'accès échantillonnés toutes les secondes (Fig. 5).

Nous avons choisi de présenter en temps réel le trajet parcouru et le nombre de point(s) d'accès vus est codé dans la couleur du point affiché (mode graphique), ou alternativement les coordonnées de la position et les identifiants des points d'accès vus (Tab. 4).

```
 4852.6325=4887.720833 00215.2192=225.365333
015 00:03:c9:a9:9a:2e
037 00:03:c9:54:ef:e3 Wanadoo_0ffc
020 00:03:c9:55:7a:ee Wanadoo_dcb4
010 00:0f:ea:5a:7c:6f 04BTBU01
020 16:5b:a5:1e:9c:6c azur
010 00:02:2d:0b:2b:1e Airport Maison
022 00:0f:b5:ea:d4:06 MARKUS
 4852.6338=4887.723000 00215.2190=225.365000
025 00:03:c9:a9:9a:2e
010 00:14:7f:0d:36:17 SpeedTouch712E8B
027 00:03:c9:54:ef:e3 Wanadoo_0ffc
015 00:03:c9:55:7a:ee Wanadoo_dcb4
027 16:5b:a5:1e:9c:6c azur
010 00:14:bf:a5:b6:6c linksys
010 00:02:2d:0b:2b:1e Airport Maison
045 00:0f:b5:ea:d4:06 MARKUS
 4852.6341=4887.723500 00215.2198=225.366333
010 00:03:c9:54:ef:e3 Wanadoo_0ffc
032 00:03:c9:55:7a:ee Wanadoo_dcb4
017 00:0f:ea:5a:7c:6f 04BTBU01
037 16:5b:a5:1e:9c:6c azur
```

```
020 00:14:bf:a5:b6:6c linksys
012 00:02:2d:0b:2b:1e Airport Maison
 4852.6336=4887.722667 00215.2224=225.370667
005 00:03:c9:a9:9a:2e
015 00:16:cf:45:ad:b4 Livebox-9BB8
001 00:14:7f:0d:36:17 SpeedTouch712E8B
015 00:16:ce:47:dd:f2 Livebox-48B8
007 fe:64:36:5d:7f:0b freephonie
027 00:03:c9:55:7a:ee Wanadoo_dcb4
017 00:0f:ea:5a:7c:6f 04BTBU01
067 00:0f:b5:ea:d4:06 MARKUS
 4852.6336=4887.722667 00215.2260=225.376667
015 00:16:ce:47:dd:f2 Livebox-48B8
032 00:03:c9:a9:9a:2e
010 00:14:a4:28:a8:fd WANADOO-C5DA
012 00:14:7f:0d:36:17 SpeedTouch712E8B
010 fe:64:36:5d:7f:0b freephonie
032 16:5b:a5:1e:9c:6c azur
015 00:0f:ea:5a:7c:6f 04BTBU01
037 00:0f:b5:ea:d4:06 MARKUS
...
```

TAB. 4 – Exemple de fichier acquis à pieds avec un échantillonnage toutes les secondes, présentant la position GPS – sous forme de trames ASCII brutes extraites de la transmission RS232 (format degrés, minutes et fractions de minutes) et après conversion au format degrés + fractions de degrés (validant ainsi le bon fonctionnement de la librairie mathématique et du calcul flottant) – associée aux points d'accès wifi visibles identifiés dans l'ordre par la puissance reçue, le BSSID et le nom.

Cette présentation de l'interfaçage d'un GPS avec la PSP ne saurait être complète sans une mention au travail de développement associé à l'application `Map This !` disponible à `http://deniska.dcemu.co.uk/`. Il ne nous a malheureusement pas encore été possible de faire fonctionner ce programme avec le PG31.

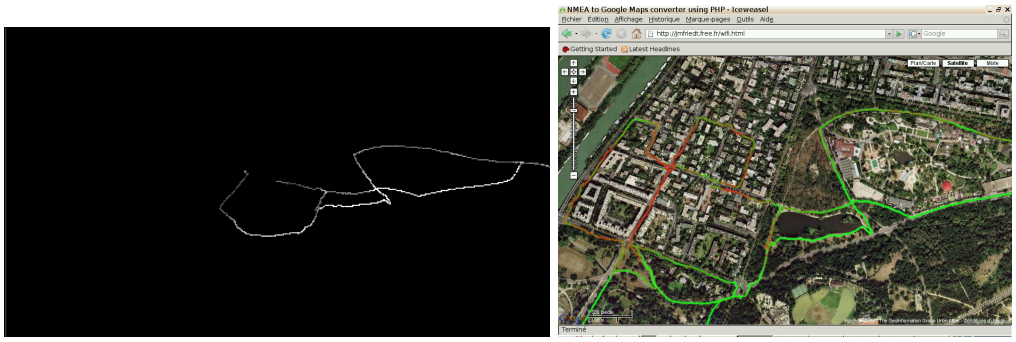

FIG. 6 – Gauche : copie d'écran d'une trace acquise au cours d'un trajet à pieds entre le bois de Boulogne et le Jardin d'Acclimatation (Paris). L'intensité de gris indique le nombre de point d'accès wifi identifiés pour chaque point GPS de la trace (échantillonnage toutes les secondes). Plus le point est clair, moins il y a de points d'accès wifi détectés. Nous constatons clairement que le nombre de point d'accès augmente lorsqu'on s'approche des quartiers résidentiels (nord) pour s'annuler dans le parc (sud). Droite : vue aérienne de la même zone. Dans cette seconde vue, l'absence de point d'accès wifi se traduit par un point vert clair qui devient d'autant plus rouge que le nombre de points d'accès détectés est élevé.

6 Conclusion

Nous avons présenté la console de jeu Sony PlayStation Portable comme une plate forme de développement d'applications embarquées aliant une puissance de calcul intéressante, une bonne autonomie pour un prix inférieur aux systèmes industriels équivalents. Nous avons présenté les modifications logicielles à effectuer sur la console pour la rendre apte à exécuter nos programmes (flasher un nouveau firmware), et la méthode pour installer les outils de cross-compilation sous GNU/Linux. Nous avons ensuite illustré notre capacité à écrire un programme fonctionnel en mode graphique par l'affichage de la fractale de Mandelbrot, pour ensuite aborder les aspects d'interfaçage de matériel sur le port de communication asynchrone compatible RS232. L'application finale est un outil de géolocalisation des points d'accès wifi.

Nous avons vu que le PSP SDK, basé sur `gcc` compilé à destination d'une architecture MIPS, nous fournit une interface de développement très proche du C classiquement utilisé pour développer des applications sous GNU/Linux. Cependant, l'utilisation intensive des fonctions fournies par le firmware propriétaire de Sony pour accéder aux fonctions bas niveau et au matériel induisent certaines différences et signifient que toutes les fonctions habituellement accessibles à un programme C ne sont pas (encore) nécessairement implémentées sur PSP.

Références

[1] GBECG – M. Cremmel, *Électrocardioscope pour la Game Boy*, Elektor (Octobre 2006), pp.32-41 qui utilise la console de Nintendo en émulation Z80.

[2] `http://friedtj.free.fr/gb_eng.pdf` pour quelque exemples sur Gameboy Pocket/Gameboy Classic (processeur Z80).

[3] `http://media.psp.ign.com/articles/542/542182/imgs_1.html`

[4] A. Rahimzadeh, *Hacking the PSP : Cool Hacks, Mods, and Customizations for the Sony Playstation Portable*, Wiley (2006), qui n'a que peu d'inérêt par ailleurs puisque ne consacrant que peu de place au développement sur cette plateforme.

[5] H.-O. Peitgen, H. Jürgens & D. Saupe, *Chaos and Fractals : New Frontiers of Science*, Springer (1993)

[6] J.-M Friedt & É. Carry, *Acquisition et dissémination de trames GPS à des fins de cartographie libre*, GNU/Linux Magazine France, Hors Série **27** (Octobre 2006)

[7] J.-M Friedt & É. Carry, *Enregistrement de trames GPS – développement sur microcontroleur 8051/8052 sous GNU/Linux*, GNU/Linux Magazine France, **81** (Février 2006)

[8] `http://mc.pp.se/psp/phones.xhtml`, `http://nil.rpc1.org/psp/remote.html` ou `http://www.dcemu.co.uk/vbulletin/showthread.php?t=30035`

[9] `http://en.wikipedia.org/wiki/Wardriving`. Notez que je cite ici la page anglophone afin d'éviter la page francophone qui associe la recherche de points d'accès wifi à un objectif de piratage informatique que je ne cautionne pas.

[10] `http://hitmen.c02.at/files/releases/psp/syscalls.txt`