**Code implementation tutorial**

To simulate a transfer function (we call it here premier\_ordre), we use the two PCS of the model to play the role of supervisor and simulator.

* **On PC1 (supervisor)**

3 processes:

- The father creates an order and sends it through a pipe to the first son

- Son 1 receives the order from the father and sends to PC2 on CAN bus

- Son 2 receives the output sent by PC2 and then stores it in a file ‘Data.txt’

* **On PC2 (simulator)**

3 processes:

- The father reads the CAN and sends the order through a pipe to the son 1

- The first son simulates a calculation step of the model and sends the result through a second pipe to son 2

- The second son receives the output calculated by the first son and returns it back to PC1.

To make the simulation on the model, a **main.cpp** file must be created on each PC containing the main program that runs the father and son process. The **main.cpp** program PC2 (simulator) uses the programs automatically generated by Matlab and other programs:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* main simulateur \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "premier\_ordre.h" /\* fichiers header crées par Matlab \*/

#include "rtwtypes.h"

#include "ert\_main.h"

#include "recvFloat.h" /\* lire et envoyer par le bus CAN\*/

#include "sendFloat.h"

#include <stdio.h> /\* Bibliothèques standards \*/

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

The **premier\_ordre.cpp** file contains all calculations done to simulate the first order function, **ert\_main.cpp** file contains the initialization of the set u (block entry) and its size that require to be changed in the program as shown (in red)below:

/\*

\* File: ert\_main.cpp

\*

\* Code generated for Simulink model 'platform'.

\*

\* Model version : 1.18

\* Simulink Coder version : 8.5 (R2013b) 08-Aug-2013

\* C/C++ source code generated on : Tue Feb 17 11:51:33 2015

\*

\* Target selection: ert.tlc

\* Embedded hardware selection: 32-bit Generic

\* Code generation objectives: Unspecified

\* Validation result: Not run

\*/

#include <stdio.h> /\* This ert\_main.c example uses printf/fflush \*/

#include "premier\_ordre.h" /\* Model's header file \*/

#include "saveData.h"

#include "rtwtypes.h"

#include "ert\_main.h"

void rt\_OneStep(void)

{

static boolean\_T OverrunFlag = 0;

/\* Disable interrupts here \*/

/\* Check for overrun \*/

if (OverrunFlag) {

rtmSetErrorStatus(premier\_ordre\_M, "Overrun");

return;

}

OverrunFlag = TRUE;

/\* Save FPU context here (if necessary) \*/

/\* Re-enable timer or interrupt here \*/

/\* Step the model \*/

premier\_ordre\_step();

/\* Indicate task complete \*/

OverrunFlag = FALSE;

/\* Disable interrupts here \*/

/\* Restore FPU context here (if necessary) \*/

/\* Enable interrupts here \*/

}

/\*

\* The example "main" function illustrates what is required by your

\* application code to initialize, execute, and terminate the generated code.

\* Attaching rt\_OneStep to a real-time clock is target specific. This example

\* illustates how you do this relative to initializing the model.

\*/

int\_T ert\_main(int\_T argc, const char \*argv[])

{

/\* Unused arguments \*/

(void)(argc);

(void)(argv);

/\* Initialize model \*/

premier\_ordre\_initialize();

/\* Attach rt\_OneStep to a timer or interrupt service routine with

\* period 0.0001 seconds (the model's base sample time) here. The

\* call syntax for rt\_OneStep is

\*/

**const int taille = 150;**

**premier\_ordre\_U.u =1.0;**

**printf("Simulation du modele pour une entree = %f\n",premier\_ordre\_U.u);**

**double tab\_Y[taille];**

for(int i=0 ; i<taille ;i++){

rt\_OneStep();

/\* Get model outputs here \*/

**tab\_Y [i]=premier\_ordre\_Y.y;**

**printf("Modele\_1er\_ordre\_Y.ouput = %f\n", premier\_ordre\_Y.y);**

}

**saveData(taille,tab\_Y,"Sortie\_Y.txt");**

fflush((NULL));

premier\_ordre\_terminate();

return 0;

}

/\*

\* File trailer for generated code.

\*

\* [EOF]

\*/

The saveData function is used to store the simulation results in a text file and is given in appendix. Once **ert\_main.cpp** edited, the **main.cpp** file (simulator) must be completed as follows:

#define TAILLE\_MESSAGE 256 /\* Correspond à la taille de la chaîne à lire et à écrire \*/

int main(void)

{

pid\_t pid\_fils1;

pid\_t pid\_fils2;

int descripteurTube[2];

int descripteurTube1\_2[2];

char messageLire[TAILLE\_MESSAGE], messageEcrire[TAILLE\_MESSAGE];

printf("Création du tube.\n");

if(pipe(descripteurTube) != 0)

{

fprintf(stderr, "Erreur de création du tube.\n");

return EXIT\_FAILURE;

}

if(pipe(descripteurTube1\_2) != 0)

{

fprintf(stderr, "Erreur de création du tube1\_2.\n");

return EXIT\_FAILURE;

}

pid\_fils1 = fork();

if(pid\_fils1 == 0) //on est dans le fils 1 -> simuler un pas de calcul

{

printf("Fermeture de l'entrée tube dans le fils1 (pid = %d).\n\n",getpid());

close(descripteurTube[1]);

printf("Fermeture de la sortie tube 1\_2 dans le fils1 (pid = %d).\n\n",getpid());

close(descripteurTube1\_2[0]);

premier\_ordre\_initialize();

while(1){

read(descripteurTube[0], messageLire, TAILLE\_MESSAGE);

premier\_ordre\_U.u=atof(messageLire);

rt\_OneStep();

sprintf(messageEcrire, "%f",premier\_ordre\_Y.y);

write(descripteurTube1\_2[1], messageEcrire, TAILLE\_MESSAGE);

}

}

else //on est dans le fils 2, on envoie la vitesse au PC1 par le CAN

{

if(pid\_fils1 == -1) //Verification que le fils 1 est bien crée

{

fprintf(stderr, "Erreur de création du processus1.\n");

return 1;

}

pid\_fils2 = fork();

if (pid\_fils2 == 0)

{

printf("Fermeture de l'entree tube 1\_2 dans le fils2 (pid = %d) .\n\n",getpid());

close(descripteurTube1\_2[1]);

float speed;

while(1){

read(descripteurTube1\_2[0], messageLire, TAILLE\_MESSAGE);

speed = atof(messageLire);

printf("Envoie de la Vitesse du véhicule = %f au PC1\n",speed);

usleep(10000);//délai de 10ms entre 2 envois

sendFloat(speed);

}

}

else { // On est dans le père, la consigne est reçue par le bus CAN puis envoyée par le tube au fils 1

if(pid\_fils2 == -1) //Verification que le fils 2 est bien cree

{

fprintf(stderr, "Erreur de création du processus2.\n");

return 1;

}

printf("\nFermeture de la sortie dans le père (pid = %d).\n", getpid());

close(descripteurTube[0]);

float consigne;

char can[15]="canA";

for (int i=0;i<150;i++){

consigne = recvFloat(can);

sprintf(messageEcrire, "%f",consigne);

printf("Nous sommes dans le père (pid = %d).\nIl envoie le message suivant au fils : \"%s\".\n\n\n", getpid(), messageEcrire);

write(descripteurTube[1], messageEcrire, TAILLE\_MESSAGE);

}

wait(NULL);

}

}

return 0;

}

Communication between father and son (on the same PC) is established using pipes, the program above creates the son process and executes a specific code according to the role of each process. Reading and sending on the CAN bus are made using two *recvFloat* and *sendFloat* functions (cf. annexes).

Similarly, the main program of PC1 is made as follows:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* main superviseur \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "sendFloat.h"

#include "recvFloat.h"

#include "saveData.h"

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#define TAILLE\_MESSAGE 256 //Correspond à la taille de la chaîne à lire et à écrire

#define CONSIGNE 100 //Consigne a envoyer

int main(void)

{

pid\_t pid\_fils1;

pid\_t pid\_fils2;

int descripteurTube[2];

char messageLire[TAILLE\_MESSAGE], messageEcrire[TAILLE\_MESSAGE];

printf("Création du tube.\n");

if(pipe(descripteurTube) != 0)

{

fprintf(stderr, "Erreur de création du tube.\n");

return EXIT\_FAILURE;

}

int ii = 0; /// initialise le fichier de sauvegarde pour ii=0

pid\_fils1 = fork(); //Creation du premier fils

if(pid\_fils1 == 0) //On est dans le fils 1 envoie la consigne dans le canA

{

printf("Fermeture de l'entrée dans le fils1.\n\n");

close(descripteurTube[1]);

while(1)

{

read(descripteurTube[0], messageLire, TAILLE\_MESSAGE);

float consigne;

consigne = atof(messageLire);

printf("Début de lenvoie par le fils 1 (pid = %d).\n",getpid());

sendFloat(consigne);

printf("Fin de lenvoie par le fils 1 (pid = %d).\n\n",getpid());

}

}

else //On est dans le père

{

if(pid\_fils1 == -1) //Verification que le fils 1 est bien cree

{

fprintf(stderr, "Erreur de création du processus1.\n");

return 1;

}

pid\_fils2 = fork(); //Creation du deuxieme fils

if(pid\_fils2 == 0) // on est dans le fils 2

{

char can[15] = "canA";

float sortie\_modele ;

while(1)

{

sortie\_modele = recvFloat(can); //réception de la réponse du PC2

saveData(sortie\_modele,"Data.txt",ii); //Stockage dans Data.txt

printf("Ecriture dans le fichier texte terminée fils 2 (pid = %d) .\n",getpid());

ii=1;

}

}

else //On est dans le père

{

if(pid\_fils2 == -1) //Verification que le fils 2 est bien cree

{

fprintf(stderr, "Erreur de création du processus2.\n");

return 1;

}

printf("Fermeture de la sortie dans le père.\n");

close(descripteurTube[0]);

for (int i=0;i<150;i++) //150=taille de la consigne

{

sprintf(messageEcrire, "CONSIGNE"); // Création de la consigne

printf("Nous sommes dans le père (pid = %d).\nIl envoie au fils : \"%s\".\n", getpid(), messageEcrire);

write(descripteurTube[1], messageEcrire, TAILLE\_MESSAGE); //envoi de la consigne au fils 1 par le tube 1

usleep(10000);

}

wait(NULL);

}

}

return 0;

It is important to specify that this last program is the one responsible of creating the order value using the instruction *sprintf(messageEcrire, "CONSIGNE")*, CONSIGNE is a global variable set at 100 and can be changed depending on the use.

The **main.cpp** program PC2 must be simulated first to wait for the order of PC1 whose **main.cpp** program is executed just after.

If all goes well, the following messages should appear on both consoles:

We visualize on the pc1 console:

Début du père  
Nous sommes dans le père (pid = 2154).  
Il envoie au fils : "1".  
Début de l’envoie par le fils 1 (pid = 2155).  
Device Opened par sendFloat (/dev/canA)  
<0x001> [4] 33 30 30 30   
Device Closed par sendFloat   
Fin de l’envoie par le fils 1 (pid = 2155)

5 seconds later (we wait for the reception from the PC 2):

<0x001> [8] 30 2e 39 35 39 39 32 33   
Device Closed par recvFloat  
la chaine vaut : 0.959923  
Ecriture dans le fichier texte terminée fils 2 (pid = 2156).  
Device Opened (/dev/canA) par recvFloat  
<0x001> [7] 31 2e 32 33 33 38 31   
Device Closed par recvFloat  
la chaine vaut : 1.23381  
Ecriture dans le fichier texte terminée fils 2 (pid = 2156).  
Device Opened (/dev/canA) par recvFloat

*Side note on the function rt\_OneStep:*

*When the function rt\_OneStep is called there is no output and no arguments involved. But you can retrieve the parameters of the simulation (the step that you have executed).*

*For that you just have to call the variable (like defined IN the fuction rt\_OneStep). We think that is because all the variables in this function are globally defined so you can access them from anywhere, the call of the function rt\_OneStep is just here to do one step of the simulation like a TIC of a clock.*

*Example:*

a = premier\_ordre\_U.u; //a: input at t

b = premier\_ordre\_Y.y; //b: Output at t

rt\_OneStep(); //Do a step of the simulation and work (and change) the variables in memory (global)

a = premier\_ordre\_U.u; //a: input at t+dt

b = premier\_ordre\_Y.y; //b: Output at t+dt

**Annexes**

***recvFloat.cpp***

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* recvFloat.cpp \*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*/

Fontionnement de recv float

Ouvre le port canA ou canB (argument en entrée de la fonction)

Rentre dans la boucle while(running)

Attend que quelqu'un parle le bus can

Dès qu'une trame est recu sauvegarde la trame dans data avec ioctl

Convertie la trame en float

Renvoie le float

-Attention cout renvoie une avec tronquée du float

le float vaut avec cout : 1.23457e+07

Le float vaut avec printf : 12345678.000000

L'utilisation de printf est recommandée

/\*/

#include "recvFloat.h"

#include <string>

using namespace std;

extern int optind, opterr, optopt;

static int s = -1;

static int running = 1;

static void sigterm(int signo) {

running = 0;

}

float recvFloat(char \*port){

#define BUF\_SIZ (255)

FILE \*out = stdout;

char \*optout\_recv = NULL;

char \*ptr;

char buf[BUF\_SIZ];

int n = 0, err;

int nbytes, i\_recv, dlc\_recv;

int opt\_recv, optdaemon = 0;

uint32\_t id\_recv, mask;

short appRetVal\_recv;

int appWRetVal\_recv;

char appDeviceName\_recv[5];

int appDevHandle\_recv;

unsigned long data[10];

int fileflag = 0;

FILE \*RX;

unsigned long fcount\_recv = 1;

struct can\_frame \*frame\_recv;

sprintf(appDeviceName\_recv,"/dev/%s",port);

//printf("appDeviceName\_recv : %s\n",appDeviceName\_recv);

signal(SIGPIPE, SIG\_IGN);

struct option long\_options[] = {

{ "help", no\_argument, 0, 'h' },

{ 0, 0, 0, 0},

};

if (optdaemon)

daemon(1, 0);

else {

signal(SIGTERM, sigterm);

signal(SIGHUP, sigterm);

}

appDevHandle\_recv = open(appDeviceName\_recv , O\_RDWR);

if(appDevHandle\_recv < 0) {

printf("Device Open Error (%s)\n", appDeviceName\_recv, appDevHandle\_recv);

exit (0);

} else {

printf("Device Opened (%s) par recvFloat\n", appDeviceName\_recv, appDevHandle\_recv);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\* Recupération des données \*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

running = 1;

while (running) {

appWRetVal\_recv = ioctl(appDevHandle\_recv, SJA1000\_IOCRECV, (unsigned long) data);

if(appWRetVal\_recv != 0)

{

//Permet d'écrire dans la console ce qui est dans la trame recue

i\_recv=0;

if (data[i\_recv] & CAN\_EFF\_FLAG){

printf("<0x%08x> ", data[i\_recv++] & CAN\_EFF\_MASK);

}

else

{

printf("<0x%03x> ", data[i\_recv++] & CAN\_SFF\_MASK);

}

//(unsigned long) DATA\_RECV = data[i\_recv]

dlc\_recv = data[i\_recv++];

printf("[%d] ", dlc\_recv);

running=0;

while(i\_recv <= dlc\_recv + 1) {

printf("%02x ", data[i\_recv++]);

}

if (data[0] & CAN\_RTR\_FLAG)

printf("remote request");

printf("\n");

running=0;

}

}

}

/\*/for (int indice = 0 ; indice<10 ; indice++){

// printf("<0x%03x> ", data[indice] & CAN\_EFF\_MASK);

cout << "data[indice] = " << data[indice] << endl ;

}/\*/

appRetVal\_recv = close(appDevHandle\_recv);

if(appRetVal\_recv == 0) {

printf("Device Closed par recvFloat\n", appRetVal\_recv);

} else {

printf("Device Close Error - %d\n", appRetVal\_recv);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\* Conversion ASCII float \*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int taille=dlc\_recv;

int i[taille-1];

char b[taille];

for (int ind=0 ; ind<taille ; ind++)

{

i[ind]=data[ind+2]; //valeur en ascii

b[ind]=(char)i[ind];//valeur du caractere

}

b[taille]='\0'; //caractère qui indique la fin de la chaine

char chaine[taille];

sprintf(chaine,"%s",b);

cout << "la chaine vaut : " << chaine << endl;

float nombre=atof(chaine);

// cout << "le float vaut avec cout : " << nombre << endl;

// printf("Le float vaut avec printf : %f\n",nombre);

return nombre;

}

***recvFloat.h***

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* recvFloat.h \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef RECVDATA\_H

#define RECVDATA\_H

#include <string>

#include <errno.h>

#include <iostream> //Pour afficher les messages (cout)

#include <fstream> //Pour la lecture des fichier en C++

#include <getopt.h>

#include <libgen.h>

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <limits.h>

#include <stdint.h>

#include <sys/uio.h>

#include <sys/ioctl.h>

#include <fcntl.h>

#include "can.h"

#include "sja1000\_ioctl.h"

using namespace std;

static void sigterm(int signo);

float recvFloat(char \*port);

#endif

***sendFloat.cpp***

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Send Float.cpp \*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//permet d'envoyer 1 trame sur le canA

//recupere un float

//attention le float doit avoir au maximum 8 caractères (sinon modifié la fonction pour envoyer plusieurs trames).

//le transforme en char

//envoie

#include "sendFloat.h"

#include <string.h>

using namespace std;

extern int optind, opterr, optopt;

float \*sendFloat(float nombre) {

struct can\_frame frame = {

//can\_id = 1,

};

int loopcount = 1, infinite = 0;

int s, opt, ret, i, dlc = 0, rtr = 0, extended = 0;

int verbose = 0,pattern = 0;

short appRetVal, appWRetVal;

//char appDeviceName[15];

char appDeviceName[15]="/dev/canA";

char \*optout = NULL;

//char \*optout = TRUE;

FILE \*TX;

int appDevHandle;

char ch;

int count = 0, id = 1;

unsigned long fcount = 1;

void handler\_fin (int unused);

struct can\_btr btr;

struct sigaction sa\_fin; //Declaration pour terminaison (ctrl-C)

pid\_t pid;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\* Conversion float ascii \*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ostringstream os;

os << nombre;

string st=os.str();

char CHAINE[st.size()];

sprintf(CHAINE , "%f",nombre);

//cout << "Size = " << s.size() <<endl;

/\*/for (int i1=0 ; i1<s.size() ; i1++){

int x = CHAINE[i1];

cout << "The character '" << CHAINE[i1] << "' has an ASCII code of " << std::hex << x << endl;

}/\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\* Mise en mémoire dans data \*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*/

for (int i2=0 ; i2<s.size() ; i2+=2){

int x1 = CHAINE[i2];

int x2 = CHAINE[i2+1];

ostringstream os2;

os2 << std::hex << x1 << std::hex << x2;

string caracteres\_s = os2.str();

char const \*caracteres\_c=caracteres\_s.c\_str();

cout << "caracteres\_c = " << caracteres\_c << endl;

frame.data[dlc] = strtoul(caracteres\_c, NULL, 0);

cout << "frame.data[dlc] = " << frame.data[dlc] << endl;

dlc++;

if (dlc == 8)

break;

}/\*/

for (int i2=0 ; i2<st.size() ; i2++){

int x1 = CHAINE[i2];

//int x2 =CHAINE[i2+1];

ostringstream os2;

os2 << x1;

//os2 << std::hex << CHAINE[i2] << CHAINE[i2+1];

string caracteres\_s = os2.str();

char const \*caracteres\_c=caracteres\_s.c\_str();

//cout << "caracteres\_c = " << caracteres\_c << endl;

frame.data[dlc] = strtoul(caracteres\_c, NULL, 0);

//cout << "frame.data[dlc] = " << frame.data[dlc] << endl;

dlc++;

//cout << "dlc = " << dlc << endl;

if (dlc == 8)

break;

}

frame.can\_dlc = dlc;

frame.can\_id=1;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\* Ouverture du port can \*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

appDevHandle = open(appDeviceName , O\_RDWR);

if(appDevHandle < 0) {

printf("Device Open Error (%s)\n", appDeviceName, appDevHandle);

exit(0);

} else {

printf("Device Opened par sendFloat (%s)\n", appDeviceName, appDevHandle);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\* Envoie direct des données \*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

while (infinite || loopcount--) {

if(pattern) {

for (i = 0; i < frame.can\_dlc; i++)

frame.data[i] = frame.data[i] + 1;

appWRetVal = ioctl(appDevHandle, SJA1000\_IOCTRANS, (unsigned long)&frame);

} else {

appWRetVal = ioctl(appDevHandle, SJA1000\_IOCTRANS, (unsigned long)&frame);

}

if (frame.can\_id & CAN\_EFF\_FLAG) {

printf("<0x%08x> ", frame.can\_id & CAN\_EFF\_MASK);

}

else

printf("<0x%03x> ", frame.can\_id & CAN\_SFF\_MASK);

dlc = frame.can\_dlc;

printf("[%d] ", dlc);

for (i = 0; i < frame.can\_dlc; i++) {

printf("%02x ", frame.data[i]);

}

if (frame.can\_id & CAN\_RTR\_FLAG)

printf("remote request");

printf("\n");

usleep(2000);

}

}

appRetVal = close(appDevHandle);

if(appRetVal == 0) {

printf("Device Closed par sendFloat \n", appRetVal);

} else {

printf("Device Close Error\n", appRetVal);

}

return 0;

}

***sendFloat.h***

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Send Float.h \*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef SENDDATA\_H

#define SENDDATA\_H

#include <string>

#include <iostream> //Pour afficher les messages (cout)

#include <fstream> //Pour la lecture des fichier en C++

#include <getopt.h>

#include <libgen.h>

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <limits.h>

#include <sys/uio.h>

#include <sys/ioctl.h>

#include <fcntl.h>

#include <sstream> //Pour la convertion float Ascii

#include "can.h"

#include "sja1000\_ioctl.h"

using namespace std;

float \*sendFloat(float char\_input);

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* saveData.cpp \*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "saveData.h"

#include "clearData.h"

#include <iostream> //Pour afficher les messages (cout)

#include <fstream> //Pour la lecture des fichier en C++

using namespace std;

float \*saveData(float sum\_save, string filename\_save,int i) {

/// Initialise le fichier

if (i == 0){clearData(filename\_save);}

//// sauvegarde des points

ofstream fichier(filename\_save.c\_str(),ios::app);

if(fichier) {

fichier << sum\_save << endl;

fichier.close(); // on referme le fichier

}

else {cerr << "La sauvegarde de points n'a pas abouti" << endl;}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* saveData.h \*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef SAVEDATA\_H

#define SAVEDATA\_H

#include <string>

#include <iostream> //Pour afficher les messages (cout)

#include <fstream> //Pour la lecture des fichier en C++

using namespace std;

float \*saveData(float sum\_save, string filename\_save,int i) ;

#endif