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
```