Université Paul Sabatier

Systèmes Temps Réel

Compte Rendu de TP Sujet -

Auteurs :
David TOCAVEN
Lucien RAKOTOMALALA

Encadrant :
Hamid DEMMOU

17 décembre 2017

Table des matières

Introduction

Chapitre 1

TP 1 : Initiation a un OS temps Réel basé sur Linux

1.1 Mesures sous Linux

1.1.1 Programme *carrelinux − comedi.c*

Ce premier programme est un générateur de signal carré. Il va nous permettre d'analyser les réponses temps réel de en étant basé sur Linux. Notre première analyse du programme donne :

- Fonction *Void out* : envoie un signal inverse celui qu'elle envoyait précédemment. La fréquence semble être défini ailleurs dans le programme. Le signal est envoyé vers le port 0 de la carte E/S initialisé dans la fonction *main*.
- *Carte E/S* : la librairie *Comedio* permet d'ouvrir la connexion série avec la carte d'entrée sortie puis d'initialiser le sens du port.
- *main* : Initialisation d'une structure de temps dans le main qui sera utilisé dans la boucle infinie du programme principal.

Le programme entre ensuite dans une boucle infinie dans laquelle il attend un temps *TIMER_ABSTIME* pour ensuite appeler la fonction *out*. Après, le programme calcule le *next shot*, i.e le prochain déclenchement, qu'il devra attendre lors du recommencement de la boucle.

Avec ce recueil d'informations, nous sommes capable de définir la fréquence du signal du signal carré que nous allons observer : elle est égale à 2 fois le *next shot* calculé dans la boucle infinie : ce calcul est : $next\_shot = 50000 + t$, la variable t appartient à la structure de temps et elle est défini en nanosecondes donc :

$$f = 2 \times 50000ns \Leftrightarrow f = 100\mu s \tag{1.1}$$

1.1.2 Mesures des période du signal

Pour mesurer les modifications de période, nous avons crée deux variables de type *timespec* : une qui mesure le temps précédent le sleep, une qui mesure a la fin de l'instance *while*(1). La mesure de la demi-période du signal carré δ est alors la différence entre le temps du début de la boucle et le temps en fin de boucle.

Pour permettre un affichage correct, nous avons introduit dans notre programme une fonction qui permet d'écrire dans un fichier les 5000 dernières δ mesurées et qui sera appelé dès que le signal **Ctrl+C** grâce à l'instruction :

```
signal(SIGINT, IntHandler)
```

Nous utilisons ensuite le fichier de mesure en .res pour un affiche avec un script *gnuplot*. Nous observons les résultats suivants :

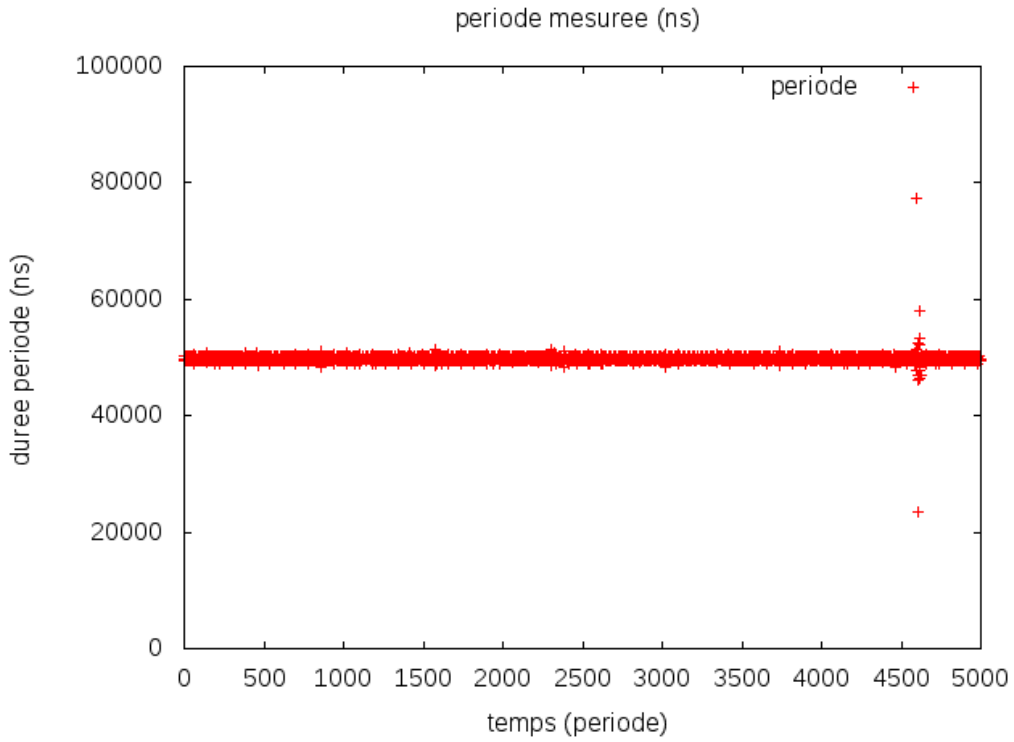

FIGURE 1.1 – Meure des périodes sans perturbation

Pour cette première observation, nous n'avons pas touché le système d'exploitation pendant la mesure des périodes et cependant, nous remarquons que celles ci ont eu out de même quelques légères perturbations. Nous allons maintenant effectué quelques perturbations pendant que le relevé s'effectue.

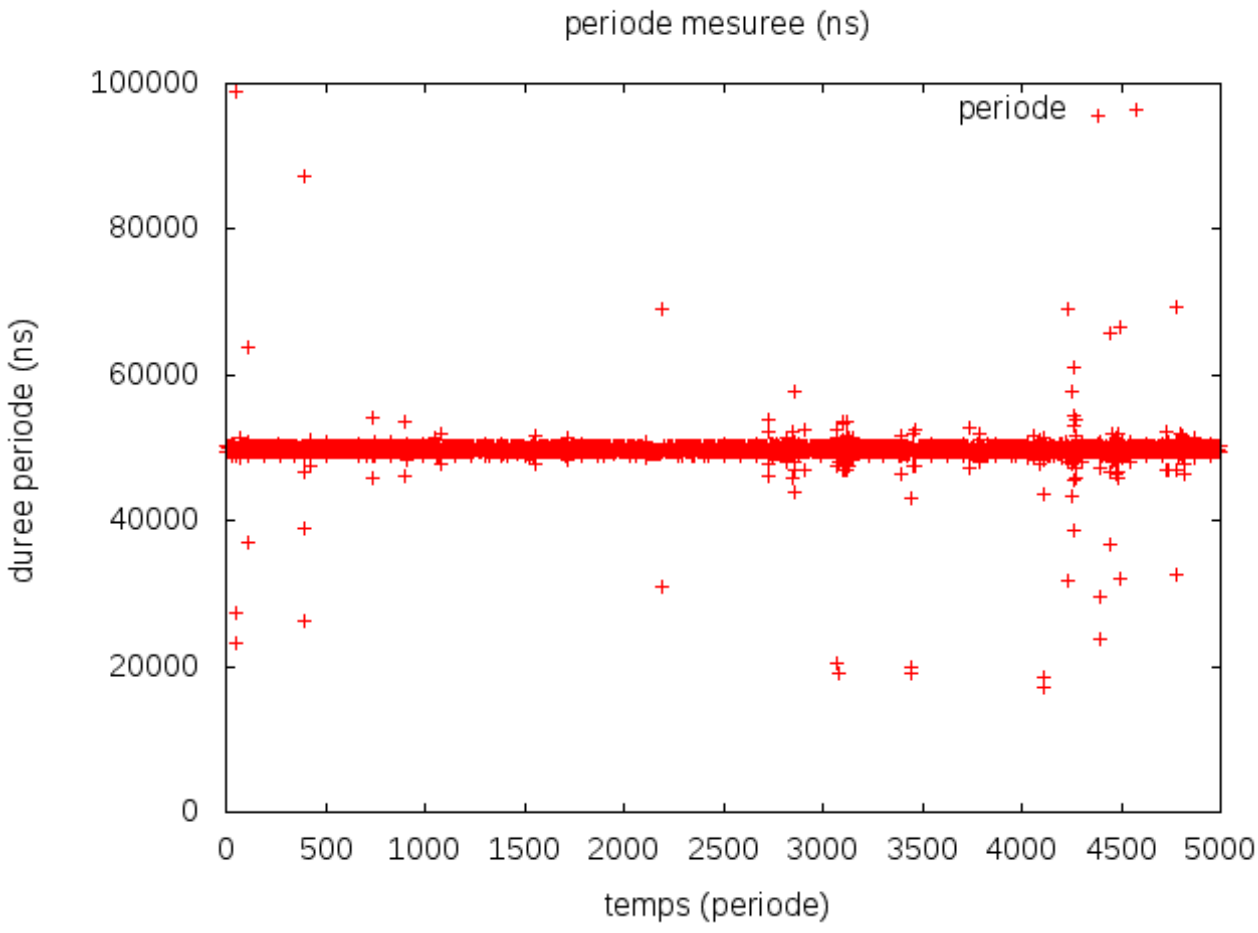

FIGURE 1.2 – Mesures des périodes avec quelques perturbations

Nous notons ici que les petites perturbations commencent à avoir de grosse conséquence sur la demi période du signal. Ces perturbations se notent aussi sur l'oscilloscope où nous observons des modifications du signal. Pour terminer cette étude, nous effectuons une dernière mesure des périodes dans laquelle nous demandons au système d'exploitation une concaténation de fichier volumineux.

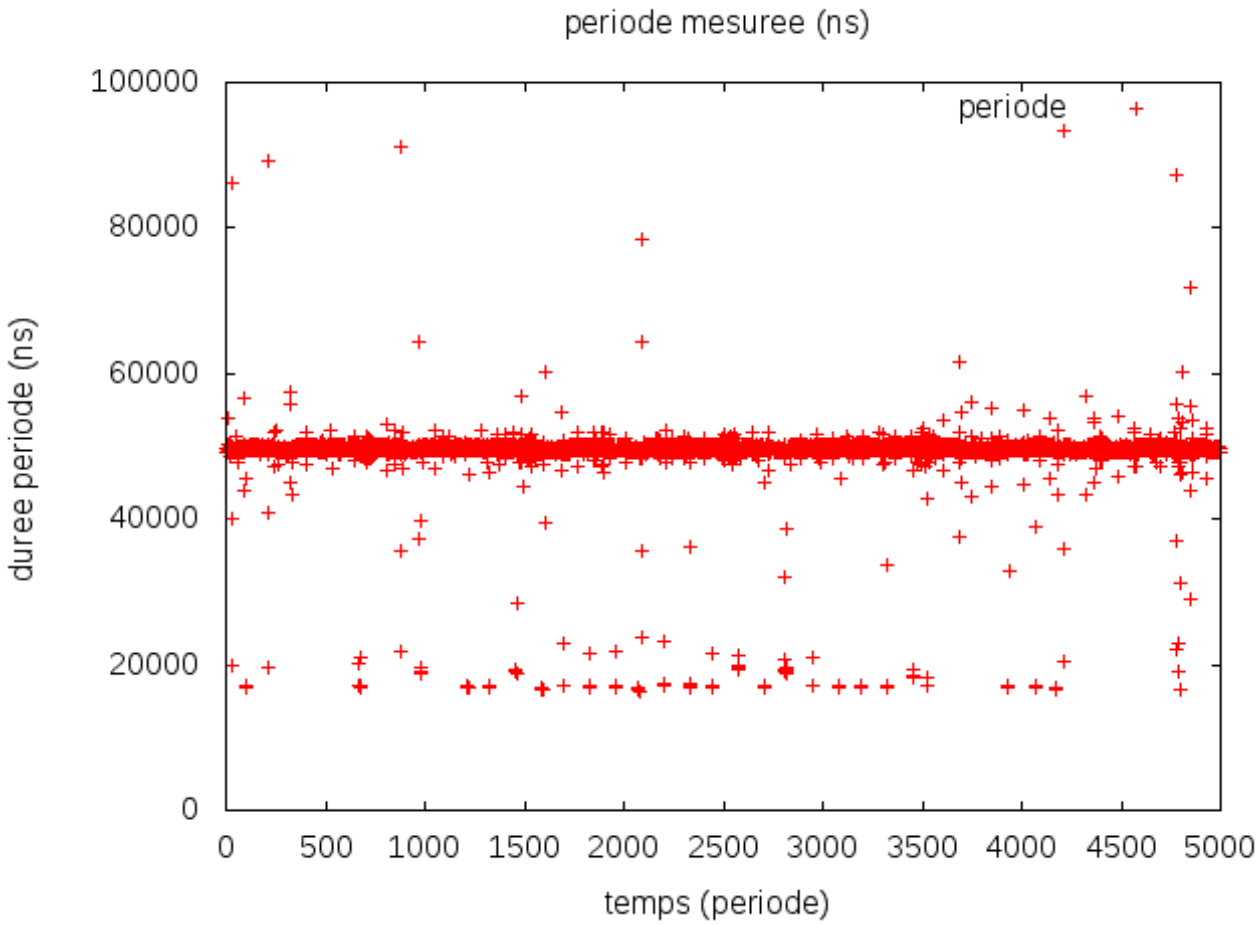

FIGURE 1.3 – Mesures des périodes avec la commande "cat /dev/zer > file2delete" en parallèle

Nous observons que la dispersion des périodes est beaucoup plus importante. Le système d'exploitation n'a pas réussi à ordonnancer notre programme avec l'instruction que nous lui avons demandé. De telles différences ne sont pas acceptables pour des applications ou des systèmes Temps Réel. Le système d'exploitation Linux seul ne peut pas être utilisé tel quel pour le Temps réel, nous devons ajouter au noyau la capacité à préempter des tâches avec l'architecture logicielle RTAI.

1.2 Mesures sous RTAI

1.2.1 L'environnement RTAI Kernel

Voir en annexe les fichiers des différents fichiers

Dans cette partie, nous avons refait la fonctionnalité précédente, c'est-à-dire un générateur de signal carré à fréquence fixe de 100 microsecondes, sous la forme d'un processus temps réel.

Dans un premier temps, nous avons modifié les fichiers *go*, *stop* et *Makefile* afin qu'ils correspondent avec le nom du fichier a compiler. En plus de cela, dans le *Makefile*, nous avons ajouté une règle *clean* afin de détruire les fichiers objets de la précédente compilation afin d'être sur que les compilations soient bien effectuées (le décalage temporel des ordinateurs causant parfois problèmes en salle de TP). Nous avons aussi, dans la règle *default*, ajouté une commande afin de rendre exécutable l'exécutable généré.

Dans un second temps, nous avons complété le processus RTAI (fichier *squelet.c*).

Nous avons complété la fonction *init_module* afin qu'elle crée et initialise la fonction *Tache1*. Cette configuration fait d'elle une tâche périodique, de fréquence $50\mu s$. De cette façon, la *Tache1* est appelée toute les demi-périodes du signal à générer. Nous avons aussi créé un compteur *now* qui permet relever le temps présent et de lancer la tâche *Tache1* en même temps que celle-ci est rendue périodique. Il y est également effectué l'ouverture de la carte d'entrée/sortie et l'initialisation d'un timer nécessaire à la mesure du temps dans la tâche *Tache1*.

La tâche *Tache1*, sert a générer le signal carré et a générer la mesure et l'écriture dans une FIFO de chaque période. Elle est organisée en deux parties :

Une initialisation où sont créé 3 entiers : *voie*, *composant* et *delta_i* servants respectivement a désigner le numéro de la voie du FIFO où nous écrivons, a désigner le numéro de son composant et a stocker la valeur de la période du signal généré. Dans une seconde partie, contenue dans une boucle infinie afin que celle-ci se répète indéfiniment. Au début de cette partie, nous relevons le temps présent grâce au timer global décrit précédemment. Il est stocké dans *delta_i*. Ensuite, nous générons la valeur haute de sortie sur le port 0. Nous attendons la fin de la période de la tâche (la moitié de celle du signal) et ensuite nous faisons de même pour la valeur basse du signal et nous récupérons le temps courant et calculons la période du signal. La dernière étape est d'écrire dans une fifo la valeur.
Une troisième tâche est décrite dans ce fichier, *cleanup_module* qui est lancée en fin d'exécution permet d'arrêter les timers et de détruire les tâches.

Dans *writeToFile.c*, qui est lancée dans *stop*, nous lisons les 5000 valeurs de la FIFO précédente et les stockons dans un fichier *"erreur.res"*.

Nous n'avons pas réussi a lire les données dans la FIFO mais les résultats attendus sont que l'application lancée sur le noyau RTAI présente une meilleure robustesse aux actions exécutées depuis le système d'exploitation et, donc, que la période du signal généré présente une plus faible variation autour de 100 microsecondes.

Chapitre 2

Chapitre 3

Chapitre 4

Conclusion

Annexes

Annexe 1 - TP 1

Code partie 1

```
/* compile using  "gcc -o swave swave.c -lrt -Wall"  */
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <sched.h>
#include <signal.h>
#include <sys/io.h>
#include </usr/local/src/comedilib/include/comedilib.h>

#define NSEC_PER_SEC    1000000000
#define DIO 2
#define SIZETAB 5000

comedi_t *cf;


/* array who contain all outputs periodes */
   unsigned int deltat[SIZETAB];

/* the struct timespec consists of nanoseconds
 * and seconds. if the nanoseconds are getting
 * bigger than 1000000000 (= 1 second) the
 * variable containing seconds has to be
 * incremented and the nanoseconds decremented
 * by 1000000000.
 */
static inline void tsnorm(struct timespec *ts)
{
   while (ts->tv_nsec >= NSEC_PER_SEC) {
      ts->tv_nsec -= NSEC_PER_SEC;
      ts->tv_sec++;
   }
}

/* increment counter and write to parallelport */
void out()
{
   static unsigned char state=0;
   comedi_dio_write(cf,DIO,0,state);
  state=!state;
}

/* for print the datas into a file */
void IntHandler(int sig)
{
  FILE *file;
  int i;
```

```
  file = fopen("/home/m2istr_13/Documents/TP_RTAI/M2ISTR_RTAI/tp1_mesures_TR/I/
      delta.res","w");
  if(file==NULL)
  {
    fprintf(stderr,"Erreur de creation du fichier\n");
  }
  for(i=0;i<SIZETAB;i++)
  {
    fprintf(file,"%d %d\n",i,deltat[i]);
  }
  fclose(file);
  exit(0);
}

/*****************************  MAIN
   ****************************************/
int main()
{
   struct timespec t, /* for output signal */
        t1,        /* for get the time at the beginning of the loop */
          t2;          /* for get the time at the end of the loop */

   /* default interval = 50000ns = 50us
    * cycle duration = 100us
    */
   int interval=50000,
       i = 0; /* for moving into deltat */

   /*    attatch the Ctr+C action to print file  */
   signal(SIGINT,IntHandler);
   cf=comedi_open("/dev/comedi0");
   if(cf==NULL)
   {
      comedi_perror("Comedi fails to open");
      return -1;
   }

   // Configure le device ANALOG_OUTPUT pour envoyer les donnees signaux
   comedi_dio_config(cf,DIO,0,COMEDI_OUTPUT);
   comedi_dio_config(cf,DIO,1,COMEDI_OUTPUT);

   /* get current time */
   clock_gettime(0,&t);

   /* start after one second */
   t.tv_sec++;

   while(1){
      /* wait untill next shot */
      clock_gettime(0,&t1);
      clock_nanosleep(0, TIMER_ABSTIME, &t, NULL);
      /* do the stuff */
      out();
      /* calculate next shot */
      t.tv_nsec+=interval;
      tsnorm(&t);
      clock_gettime(0,&t2);
      deltat[i] = t2.tv_nsec - t1.tv_nsec + t2.tv_sec - t1.tv_sec;
      i=(i+1)%SIZETAB;
   }
   return 0;
```

|| }

Annexe 2 - TITRE