- PROJET RTEL4I -

Conception détaillée du portage de Xenomai sur processeur NIOS2 (L2.4-c)

SOMMAIRE

1. Contexte du document**......4**

2. Mesures des performances**......5**

2.1. Mesures avec un oscilloscope......5

2.2. Mesures avec les outils Xenomai......7

3. Conclusion**......11**

4. Références documentaires**......11**

5. Références logicielles :......12

6. Annexe 2 : fichier source de génération d'un signal périodique processeur non chargé sous µClinux......13

7. Annexe 3 : fichier source de génération d'un signal périodique processeur chargé sous µClinux......15

8. Annexe 4 : fichier source de génération d'un signal périodique processeur non chargé sous Xenomai/µClinux......17

9. Annexe 5 : fichier source de génération d'un signal périodique processeur chargé sous Xenomai/µClinux......19

1. CONTEXTE DU DOCUMENT

Ce document présente les moyens de test utilisés pour la validation du portage de Xenomai sur architecture NIOS2.

2. MESURES DES PERFORMANCES

2.1. Mesures avec un oscilloscope

Les mesures de performances ont été effectuées sur la carte cible Stratix 1S10 d'Altera.

Il faut rappeler que le processeur a une fréquence de fonctionnement de 50 MHz. Les expériences menées ont consisté en la génération d'un signal carré périodique sur un port d'E/S parallèle de la carte cible grâce à un programme écrit en langage C.

La précision du signal de sortie est donc soumise aux contraintes de gestion du temps du système d'exploitation µClinux. Le but est de mettre en évidence à la fois l'intérêt du système Temps Réel vis-à-vis du système à temps partagé et la conservation du bornage du temps d'exécution que le processeur ait une forte charge de travail ou non.

Voici tout d'abord un créneau généré par le processeur sous Linux embarqué sans charge. La fréquence du créneau demandé est de 100 Hz.

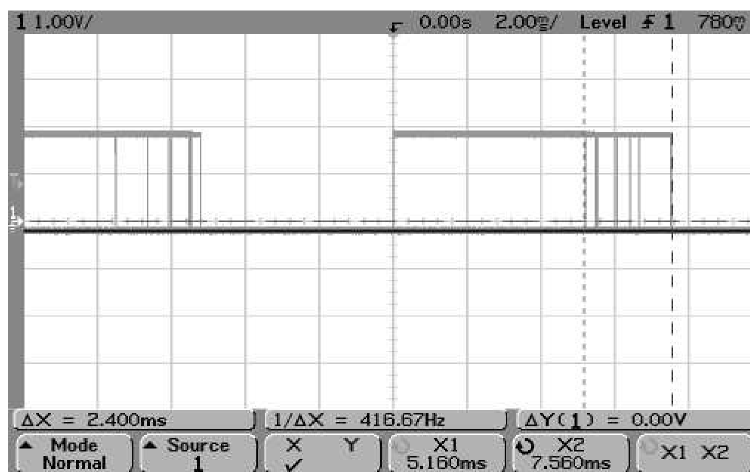

Figure 19 : Signal carré périodique généré sous Linux embarqué sans charge

Sans charge, la jigue générée est donc de 2,4 ms. La fréquence varie donc de 80 à 100Hz.

Par la suite, l'expérience est renouvelée en chargeant le processeur par exemple en écrivant et effaçant continuellement un gros fichier.

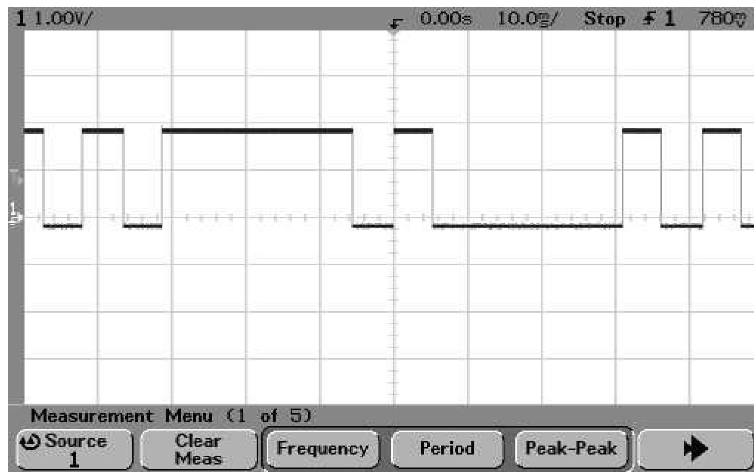

Figure 20 : Signal carré périodique généré sous Linux embarqué chargé

On voit que le signal généré n’est plus valide du tout.

Ces mêmes mesures sont effectuées avec l’extension Xenomai activé sous Linux embarqué.

La mesure est faite d’abord sans charge du processeur.

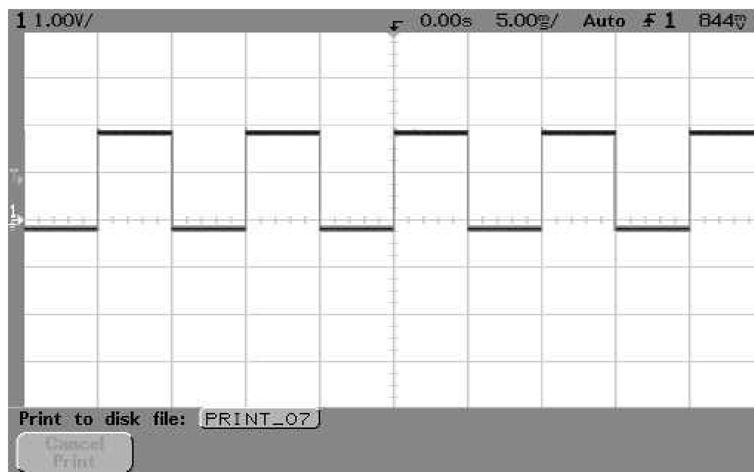

Figure 21 : Signal carré périodique généré sous Linux embarqué avec Xenomai sans charge

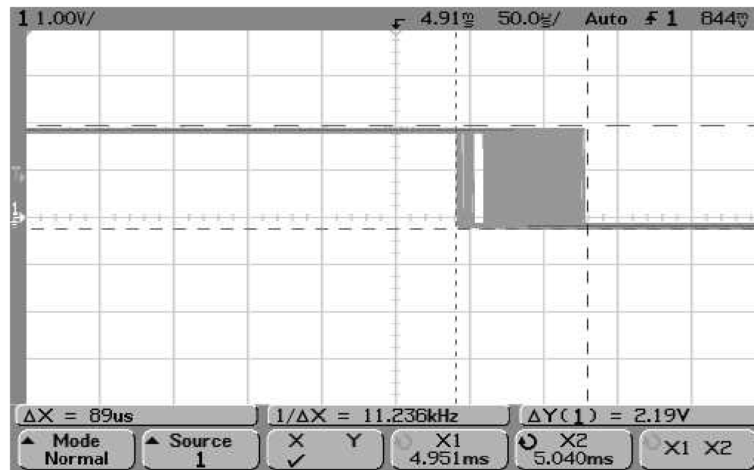

Figure 22 : Signal carré périodique généré sous Linux embarqué avec Xenomai sans charge (zoom)

La jigue maximale mesurée est de 89 µs.

La mesure est ensuite faite d'abord avec charge du processeur.

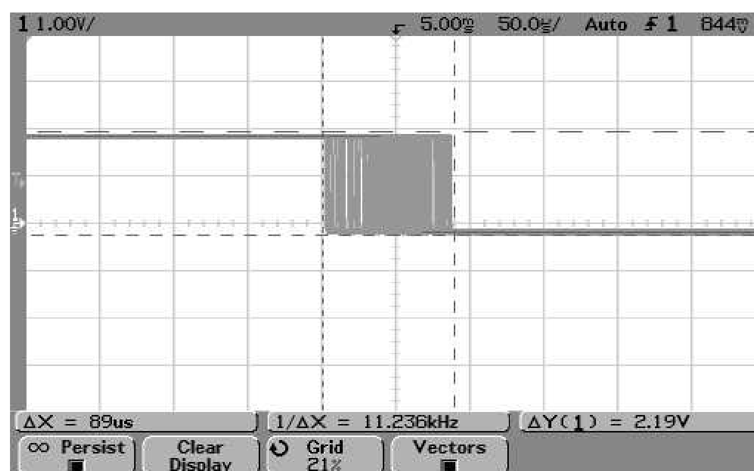

Figure 23 : Signal carré périodique généré sous Linux embarqué avec Xenomai avec charge (zoom)

La jigue maximale mesurée est également de 89 µs. La précision sur le signal périodique à générer est conservée avec Xenomai.

2.2. Mesures avec les outils Xenomai

On utilise maintenant les outils standards de mesure de performances fournis avec Xenomai.
Le principe de base est de générer une tâche Xenomai périodique et de mesurer la latence mesurée par rapport à la période théorique (usage du timer *hrclock*).

L'outil Xenomai utilisé est *latency*.

1. Tests latency en mode user

Dans un premier temps, le processeur n'est pas chargé en utilisant *latency* en mode *user* :

```
# ./ latency –t1
```

Les mesures donnent un temps de latence dans le pire cas de 314,26 µs.

```
== Sampling period: 10000 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
RTT|  00:00:01  (periodic user-mode task, 10000 us period, priority 99)
RTH|-----lat min|-----lat avg|-----lat max|-overrun|----lat best|---lat worst
RTD|     202.340|     246.560|     311.940|       0|     202.340|     311.940
RTD|     202.520|     262.020|     314.260|       0|     202.340|     314.260
RTD|     202.780|     257.840|     312.340|       0|     202.340|     314.260
RTD|     204.480|     257.980|     312.600|       0|     202.340|     314.260
RTD|     202.340|     257.980|     311.940|       0|     202.340|     314.260
RTD|     201.980|     258.000|     311.860|       0|     201.980|     314.260
RTD|     203.020|     258.260|     311.780|       0|     201.980|     314.260
RTD|     204.220|     257.060|     312.020|       0|     201.980|     314.260
RTD|     209.120|     260.620|     311.720|       0|     201.980|     314.260
RTD|     202.260|     261.000|     311.980|       0|     201.980|     314.260
RTD|     202.700|     252.980|     312.760|       0|     201.980|     314.260
RTD|     202.860|     257.220|     311.700|       0|     201.980|     314.260
RTD|     202.340|     260.460|     312.800|       0|     201.980|     314.260
RTD|     202.260|     261.760|     311.860|       0|     201.980|     314.260
RTD|     209.080|     260.100|     312.640|       0|     201.980|     314.260
RTD|     202.940|     262.280|     312.140|       0|     201.980|     314.260
RTD|     202.660|     260.360|     312.280|       0|     201.980|     314.260
RTD|     202.640|     258.660|     311.540|       0|     201.980|     314.260
RTD|     202.300|     258.960|     311.900|       0|     201.980|     314.260
---|------------|------------|------------|--------|-------------------------
RTS|     201.980|     258.420|     314.260|       0|    00:00:20/00:00:20
```

Figure 24 : Résultats des tests latency en mode *user* sans charge

On stresse le processeur et l'on réitère les mesures.

Les mesures donnent un temps de latence dans le pire cas de 314,72 µs.

```
== Sampling period: 10000 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
RTT|  00:00:01  (periodic user-mode task, 10000 us period, priority 99)
RTH|-----lat min|-----lat avg|-----lat max|-overrun|----lat best|---lat worst
RTD|     234.320|     248.680|     293.080|       0|     234.320|     293.080
RTD|     203.440|     246.320|     311.020|       0|     203.440|     311.020
RTD|     204.840|     246.340|     313.480|       0|     203.440|     313.480
RTD|     202.760|     247.740|     313.440|       0|     202.760|     313.480
RTD|     211.260|     248.440|     301.080|       0|     202.760|     313.480
RTD|     207.100|     246.460|     299.000|       0|     202.760|     313.480
RTD|     202.800|     249.800|     314.720|       0|     202.760|     314.720
RTD|     205.560|     256.060|     307.400|       0|     202.760|     314.720
RTD|     207.300|     248.800|     303.580|       0|     202.760|     314.720
RTD|     205.340|     251.380|     302.280|       0|     202.760|     314.720
RTD|     206.400|     255.140|     310.080|       0|     202.760|     314.720
RTD|     205.820|     255.140|     310.260|       0|     202.760|     314.720
RTD|     204.860|     255.620|     308.320|       0|     202.760|     314.720
RTD|     208.080|     248.980|     313.940|       0|     202.760|     314.720
RTD|     202.920|     250.380|     305.380|       0|     202.760|     314.720
RTD|     205.500|     254.600|     308.240|       0|     202.760|     314.720
RTD|     205.940|     255.320|     313.160|       0|     202.760|     314.720
RTD|     203.320|     256.780|     306.340|       0|     202.760|     314.720
RTD|     209.280|     246.420|     299.500|       0|     202.760|     314.720
---|------------|------------|------------|--------|-------------------------
RTS|     202.760|     250.960|     314.720|       0|    00:00:20/00:00:20
```

Figure 25 : Résultats des tests latency en mode *user* avec charge

2. Tests latency en mode kernel

Dans un deuxième temps, le processeur n'est pas chargé en utilisant *latency* en mode *kernel* :

```
# ./ latency -t2
```

```
== Sampling period: 10000 us
== Test mode: in-kernel timer handler
== All results in microseconds
warming up...
RTT|  00:00:01  (in-kernel timer handler, 10000 us period, priority 99)
RTH|-----lat min|-----lat avg|-----lat max|-overrun|----lat best|---lat worst
RTD|      22.920|      48.282|      66.780|       0|      22.920|      66.780
RTD|      17.100|      58.571|     106.360|       0|      17.100|     106.360
RTD|       7.360|      54.721|      94.040|       0|       7.360|     106.360
RTD|      26.700|      56.096|      92.400|       0|       7.360|     106.360
RTD|      12.680|      53.571|      92.120|       0|       7.360|     106.360
RTD|      18.000|      51.837|      91.300|       0|       7.360|     106.360
RTD|      16.760|      54.180|      87.700|       0|       7.360|     106.360
RTD|      21.580|      55.595|     102.940|       0|       7.360|     106.360
RTD|      27.120|      55.062|      97.060|       0|       7.360|     106.360
RTD|      25.920|      53.633|      95.880|       0|       7.360|     106.360
RTD|       3.580|      52.902|      86.620|       0|       3.580|     106.360
RTD|      21.000|      54.332|     102.120|       0|       3.580|     106.360
RTD|      24.960|      55.215|      93.280|       0|       3.580|     106.360
RTD|      38.520|      56.561|      84.960|       0|       3.580|     106.360
RTD|      19.000|      56.026|     103.660|       0|       3.580|     106.360
RTD|      40.320|      55.980|      93.560|       0|       3.580|     106.360
RTD|      16.940|      51.754|     105.900|       0|       3.580|     106.360
RTD|       1.160|      52.571|      90.000|       0|       1.160|     106.360
RTD|       4.200|      56.019|      97.200|       0|       1.160|     106.360
---|------------|------------|------------|--------|-------------------------
RTS|       1.160|      54.363|     106.360|       0|    00:00:20/00:00:20
```

Figure 26 : Résultats des tests latency en mode *kernel* sans charge

Les mesures donnent un temps de latence dans le pire cas de 106,36 µs.

On stresse le processeur et l'on réitère les mesures.

Les mesures donnent un temps de latence dans le pire cas de 109,4 µs.

```
== Sampling period: 10000 us
== Test mode: in-kernel timer handler
== All results in microseconds
warming up...
RTT|  00:00:01  (in-kernel timer handler, 10000 us period, priority 99)
RTH|-----lat min|-----lat avg|-----lat max|-overrun|----lat best|---lat worst
RTD|       1.580|      51.823|     108.180|       0|       1.580|     108.180
RTD|       0.060|      56.703|     100.980|       0|       0.060|     108.180
RTD|       1.540|      56.243|     107.700|       0|       0.060|     108.180
RTD|       0.900|      54.928|      97.900|       0|       0.060|     108.180
RTD|       2.580|      54.569|     102.520|       0|       0.060|     108.180
RTD|       1.940|      56.734|     101.580|       0|       0.060|     108.180
RTD|       1.540|      56.515|     104.140|       0|       0.060|     108.180
RTD|       1.900|      55.250|      94.560|       0|       0.060|     108.180
RTD|       1.520|      55.008|     106.620|       0|       0.060|     108.180
RTD|       1.940|      54.149|      95.140|       0|       0.060|     108.180
RTD|       1.540|      56.936|     109.400|       0|       0.060|     109.400
RTD|       1.900|      55.798|     107.740|       0|       0.060|     109.400
RTD|       2.260|      57.401|     102.240|       0|       0.060|     109.400
RTD|       1.620|      56.371|      92.540|       0|       0.060|     109.400
RTD|       1.660|      54.902|     107.820|       0|       0.060|     109.400
RTD|       1.580|      55.699|     107.740|       0|       0.060|     109.400
RTD|       1.500|      56.673|     107.940|       0|       0.060|     109.400
RTD|       2.620|      55.029|      94.180|       0|       0.060|     109.400
RTD|       1.660|      56.646|     108.040|       0|       0.060|     109.400
---|------------|------------|------------|--------|-------------------------
RTS|       0.060|      55.651|     109.400|       0|    00:00:20/00:00:20
```

Figure 27 : Résultats des tests latency en mode *kernel* avec charge

On constate que la charge du processeur NIOS II n'influence pas les temps de latence mesurés avec latency en mode *user* et en mode *kernel*.

3. CONCLUSION

Le portage de Xenomai pour le processeur NIOS II dans l'environnement µClinux (sans MMU) est opérationnel.
Nous avons défini un design matériel original qui a facilité le portage de Xenomai.

Le portage a été testé sur une carte cible d'Altera (Stratix 1S10) fonctionnant à 50 MHz. Avec cette fréquence relativement basse pour un processeur embarqué, nous avons des temps de latence de l'ordre d'une centaine de µs, ce qui est très honorable avec ce processeur.

Nous tenons à remercier Philippe Gerum, concepteur et mainteneur du projet Xenomai, pour toute l'aide qu'il a pu nous apporter pour mener à bien ce portage.

4. RÉFÉRENCES DOCUMENTAIRES

[1] Projet Xenomai. http://www.xenomai.org

[2] Projet µClinux pour processeur NIOS II. http://www.nioswiki.com/

[3] Page InstallNios2Linux. http://www.nioswiki.com/InstallNios2Linux

[4] Page QuartusforLinux. http://www.nioswiki.com/OperatingSystems/UClinux/QuartusforLinux

[5] Page des ressources ENSEIRB-MATMECA de Linux embarqué et du portage de Xenomai sur processeur NIOS II. http://www.enseirb.fr/~kadionik/nios-xenomai

5. RÉFÉRENCES LOGICIELLES :

[1] Outils Altera sous Linux : ftp://ftp.altera.com/outgoing/release

[2] Portage µClinux pour le processeur NIOS II : ftp://ftp.altera.com/outgoing/nios2-linux-20080619.tar

[3] Compilateur croisé gcc pour processeur NIOS II : http://www.niosftp.com/pub/gnutools/nios2gcc-20080203.tar.bz2

[4] Portage Xenomai pour le processeur NIOS II : http://git.xenomai.org/

[5] Ces briques logicielles sont en secours sur la page Xenomai pour le processeur NIOS II de l’ENSEIRB-MATEMCA : http://www.enseirb.fr/~kadionik/nios-xenomai/

6. ANNEXE 2 : FICHIER SOURCE DE GÉNÉRATION D'UN SIGNAL PÉRIODIQUE PROCESSEUR NON CHARGÉ SOUS MCLINUX

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <sys/mman.h>

#include <native/task.h>
#include <native/timer.h>

#define TIMESLEEP 10000
#define LED 0x00810880

static void * work_thread (void  *vptr_args){

FILE * file;
int i = 500;
int j = 100;
while(j--){

  file = fopen ("work_file", "w");
  while(i--){
      fputs ("Hello", file);
      }
  fclose (file);

  }
pthread_exit(NULL);
}

static void * led_thread (void  *vptr_args){

  int iomask = 0x00;
  char * leds;
  int inc ;
  leds = LED;

  while(1){
      inc = TIMESLEEP ;
      while(inc --){};
      *leds = iomask;
        iomask^=1;
  }

pthread_exit(NULL);
}

int main(int argc, char **argv) {

  pthread_t thread;
```

```
  pthread_t thread_led;

  /* Hardwork task creation */
   if (pthread_create(&thread, NULL, work_thread, NULL) != 0)
    {
        return EXIT_FAILURE;
    }


  /* Blink leds task creation */
   if (pthread_create(&thread_led, NULL, led_thread, NULL) != 0)
    {
        return EXIT_FAILURE;
    }

 getchar();
 pthread_exit(NULL);

  return 0;
}
```

7. ANNEXE 3 : FICHIER SOURCE DE GÉNÉRATION D'UN SIGNAL PÉRIODIQUE PROCESSEUR CHARGÉ SOUS MCLINUX

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <sys/mman.h>

#include <native/task.h>
#include <native/timer.h>

#define TIMESLEEP 10000
#define LED 0x00810880

static void * work_thread (void  *vptr_args){

FILE * file;
int i = 500;
int j = 100;
while(j--){

  file = fopen ("work_file", "w");
  while(i--){
      fputs ("Hello", file);
      }
  fclose (file);

  }
pthread_exit(NULL);
}

static void * led_thread (void  *vptr_args){

  int iomask = 0x00;
  char * leds;
  int inc ;
  leds = LED;

  while(1){
      inc = TIMESLEEP ;
      while(inc --){};
      *leds = iomask;
        iomask^=1;
  }

pthread_exit(NULL);
}

int main(int argc, char **argv) {

  pthread_t thread;
```

```
  pthread_t thread_led;

  /* Hardwork task creation */
   if (pthread_create(&thread, NULL, work_thread, NULL) != 0)
    {
        return EXIT_FAILURE;
    }


  /* Blink leds task creation */
   if (pthread_create(&thread_led, NULL, led_thread, NULL) != 0)
    {
        return EXIT_FAILURE;
    }

 getchar();
 pthread_exit(NULL);

  return 0;
}
```

8. ANNEXE 4 : FICHIER SOURCE DE GÉNÉRATION D'UN SIGNAL PÉRIODIQUE PROCESSEUR NON CHARGÉ SOUS XENOMAI/MCLINUX

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <sys/mman.h>

#include <native/task.h>
#include <native/timer.h>

#define TIMESLEEP 5000000
#define LED 0x00810880

RT_TASK blink_task;
int terminate=0;

void blink(void *arg){
  int iomask = 0x00;
  char * leds;
  leds = LED;
  rt_task_set_periodic(NULL, TM_NOW, TIMESLEEP);

  while(1){
      rt_task_wait_period(NULL);
      *leds = iomask;
        iomask^=1;
  }
}

static void * work_thread (void  *vptr_args){

FILE * file;
int i = 350;
int j = 100;
while(j--){

  file = fopen ("work_file", "w");
  while(i--){
      fputs ("Hello", file);
      }
  fclose (file);

  }
pthread_exit(NULL);
}

void catch_signal() {}

int main(int argc, char **argv) {
```

```
  signal(SIGTERM, catch_signal);
  signal(SIGINT, catch_signal);
  pthread_t thread;

  /* Avoids memory swapping for this program */
  mlockall(MCL_CURRENT|MCL_FUTURE);


  /* RT Task Creation */
  rt_task_create(&blink_task, "blinkLed", 0, 99, 0);
  rt_task_start(&blink_task, &blink, NULL);

  /* Hardwork task creation */
   if (pthread_create(&thread, NULL, work_thread, NULL) != 0)
    {
        return EXIT_FAILURE;
    }

 getchar();
  rt_task_delete(&blink_task);
 getchar();
 pthread_exit(NULL);

  return 0;
}
```

9. ANNEXE 5 : FICHIER SOURCE DE GÉNÉRATION D'UN SIGNAL PÉRIODIQUE PROCESSEUR CHARGÉ SOUS XENOMAI/MCLINUX

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <sys/mman.h>

#include <native/task.h>
#include <native/timer.h>

#define TIMESLEEP 5000000
#define LED 0x00810880

RT_TASK blink_task;
int terminate=0;

void blink(void *arg){
  int iomask = 0x00;
  char * leds;
  leds = LED;
  rt_task_set_periodic(NULL, TM_NOW, TIMESLEEP);

  while(1){
      rt_task_wait_period(NULL);
      *leds = iomask;
        iomask^=1;
  }
}

static void * work_thread (void  *vptr_args){

FILE * file;
int i = 350;
int j = 100;
while(j--){

  file = fopen ("work_file", "w");
  while(i--){
      fputs ("Hello", file);
      }
  fclose (file);

  }
pthread_exit(NULL);
}

void catch_signal() {}

int main(int argc, char **argv) {
```

```
  signal(SIGTERM, catch_signal);
  signal(SIGINT, catch_signal);
  pthread_t thread;

  /* Avoids memory swapping for this program */
  mlockall(MCL_CURRENT|MCL_FUTURE);


  /* RT Task Creation */
  rt_task_create(&blink_task, "blinkLed", 0, 99, 0);
  rt_task_start(&blink_task, &blink, NULL);

  /* Hardwork task creation */
   if (pthread_create(&thread, NULL, work_thread, NULL) != 0)
    {
        return EXIT_FAILURE;
    }

 getchar();
  rt_task_delete(&blink_task);
 getchar();
 pthread_exit(NULL);

  return 0;
}
```