Systèmes embarqués 5/7

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

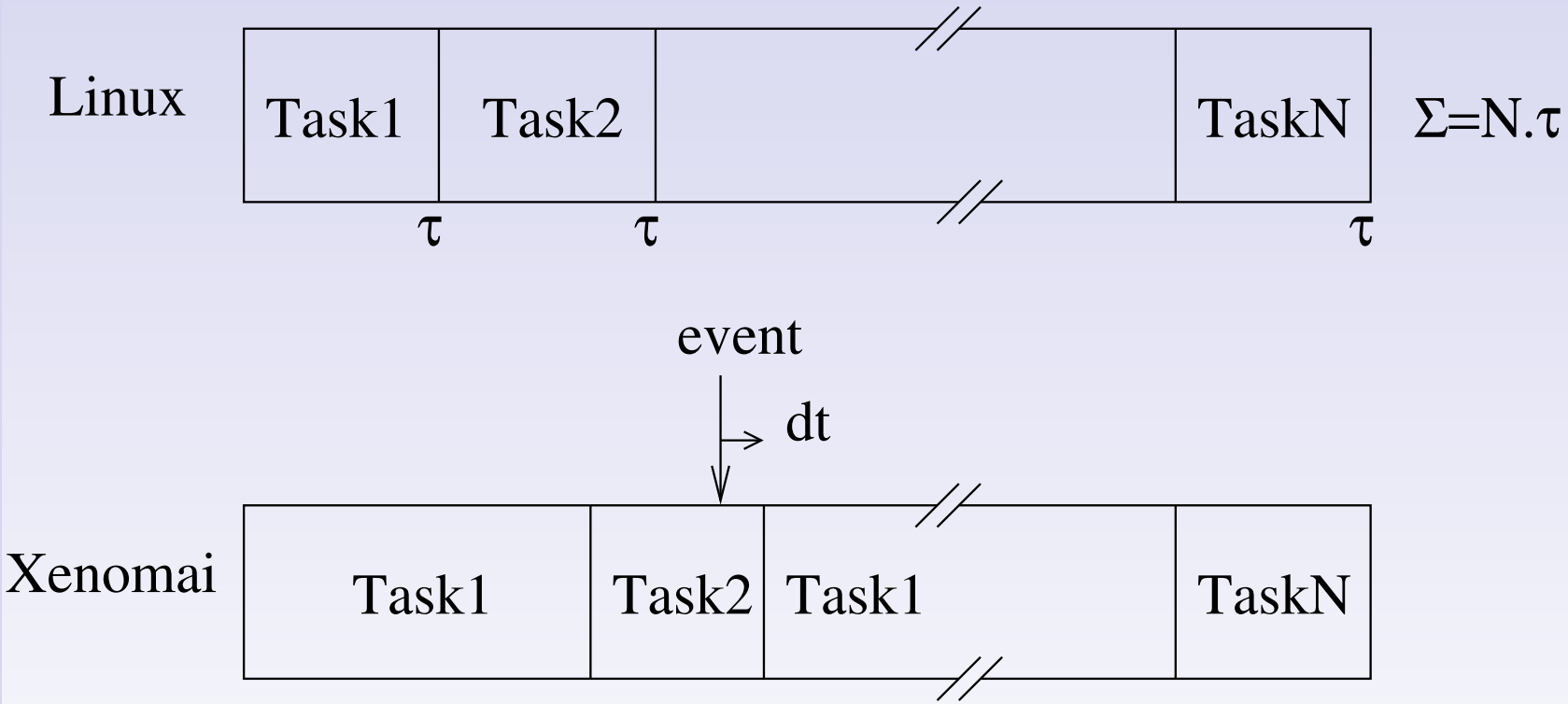
12 janvier 2019

Le temps réel : concepts

- Notion de temps réel : plus on ajoute de couches d'abstractions, plus on augmente la latence.
- Le temps réel garantit une latence bornée.
- Quelques "OS" (environnements exécutifs) libres temps réels : `http://www.freertos.org/`, `http://www.rtems.com/`
- Linux comme solution pour exporter vers un environnement embarqué [1] [2] les programmes développés sur PC
- Les interruptions sont source de latence (on fait autre chose que la tâche prioritaire en cours) $\Rightarrow$ processus atomique qui ne peut être interrompu (désactiver les interruptions matérielles en entrée et réactiver en sortie ... risque de pertes de données).
- Extension temps réel de Linux : xenomai [3].

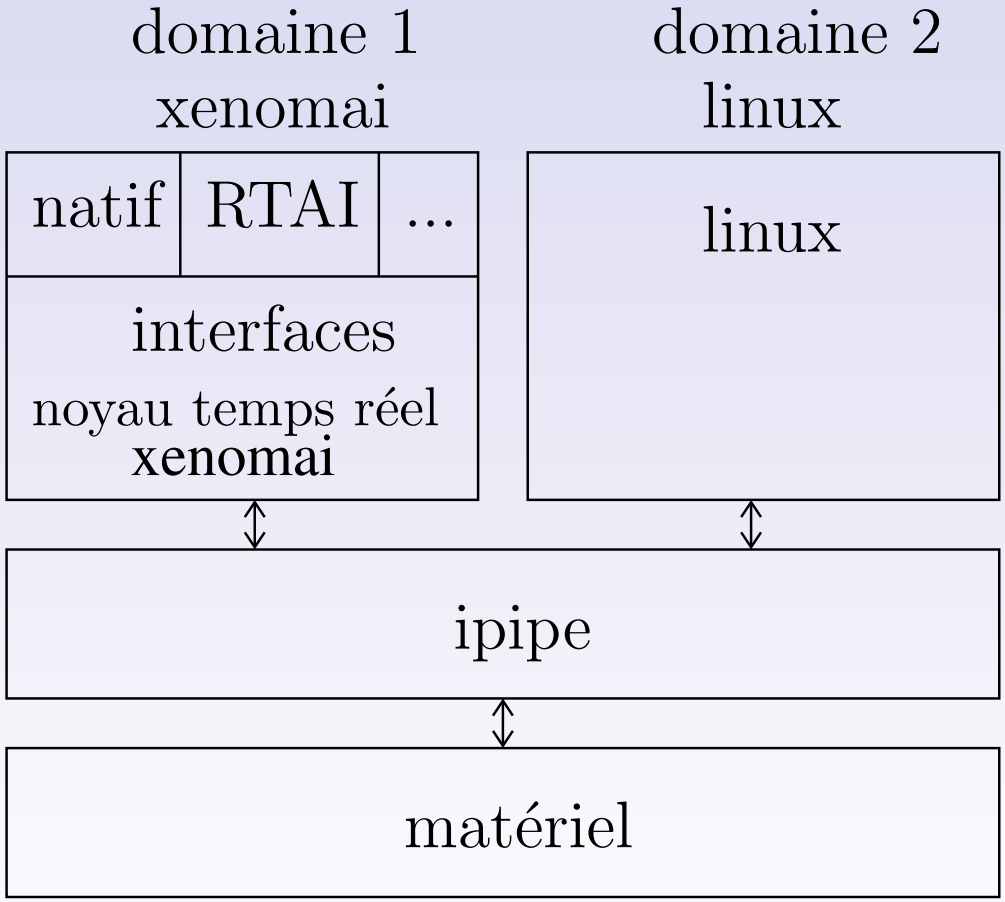
1. P. Ficheux & É. Bénard, *Linux Embarqué 4ème édition*, Eyrolles (2012)
2. K. Yaghmour, J. Masters, G. Ben-Yossef, P. Gerum, *Building Embedded Linux Systems, 2nd Ed.*, O'Reilly (2008)
3. `http://www.xenomai.org/index.php/Main_Page`

Ordonnanceurs



Xenomai

- Xenomai est un ensemble de patchs au noyau linux.
- Domaines $\simeq$ noyaux. Accès concurrents aux ressources.
- Applications Xenomai sont exécutées en espace utilisateur.



Skins = diverses API vers le noyau temps-réel xenomai

Le temps réel : exemple

Système “classique”

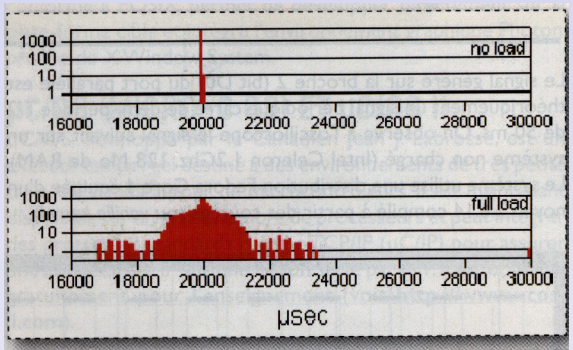
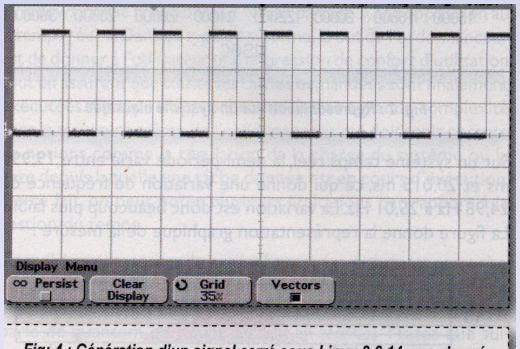


Fig. 2 : Représentation sur un système classique



Fig; 4 : Génération d'un signal carré sous Linux 2.6.14 non chargé

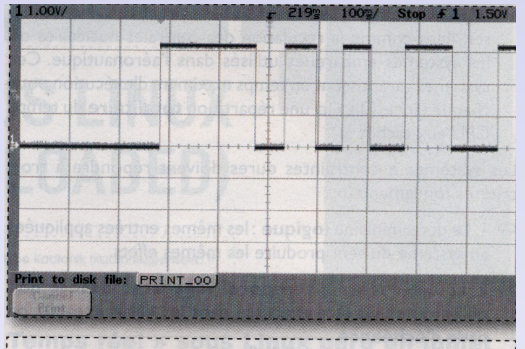


Fig. 5 : Génération d'un signal carré sous Linux 2.6.14 chargé

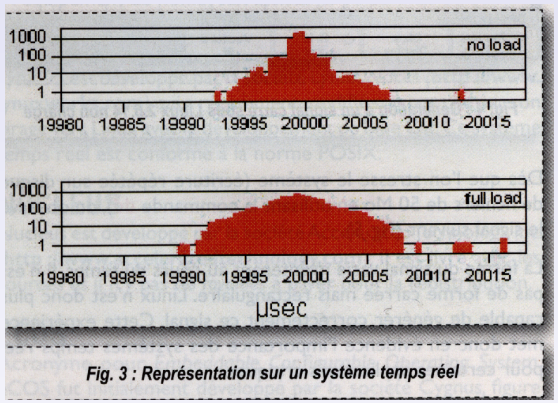


Fig. 3 : Représentation sur un système temps réel

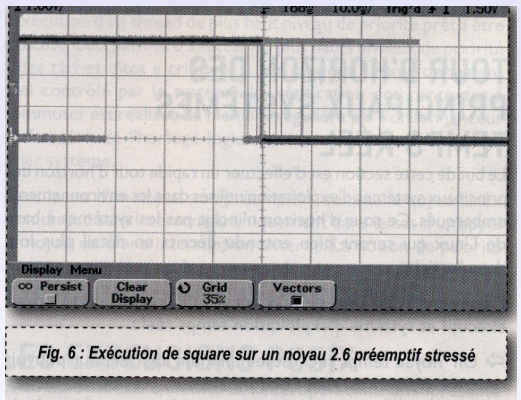


Fig. 6 : Exécution de square sur un noyau 2.6 préemptif stressé

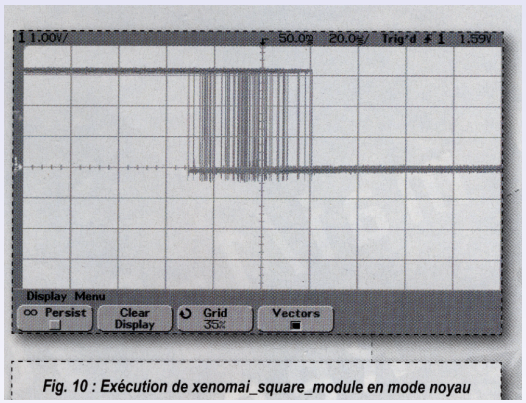


Fig. 10 : Exécution de xenomai_square_module en mode noyau

Extension `xenomai` temps-réel de Linux[4]

4. Toutes le figures sont extraites de P. Ficheux &. P. Kadionik, *Temps réel sous Linux (reloaded)*, GNU/Linux Magazine France, Hors Série 24 (Février-Mars 2006), pp.72-81

Xenomai : extension temps-réel de Linux

Qualification des latences : accès direct au matériel sans passer au travers des couches du noyau

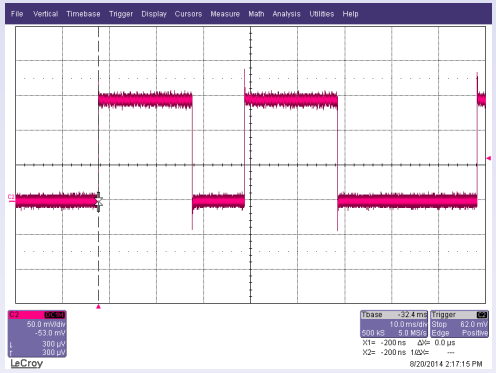
```
unsigned char* red_gpio_init()
{const int base_addr2= 0xe000a000;
 unsigned char *h     = NULL;
 unsigned int page_addr,page_offset;
 unsigned page_size=sysconf(_SC_PAGESIZE);
[...]
 page_addr  = base_addr2 & (~(page_size-1));
 page_offset= base_addr2-page_addr;
 map_file = open("/dev/mem", O_RDWR | O_SYNC);
 h=mmap(NULL,page_size,PROT_READ|PROT_WRITE,MAP_SHARED,→
     ↪map_file,page_addr);
 return(h);
}

void red_gpio_output(unsigned char *h,int num,int value)
{if (value!=0)
    *(unsigned int*)(h+0x040)|=(1<<num);   // output value
 else
    *(unsigned int*)(h+0x040)&=(~(1<<num));
}
```

Xenomai : extension temps-réel de Linux

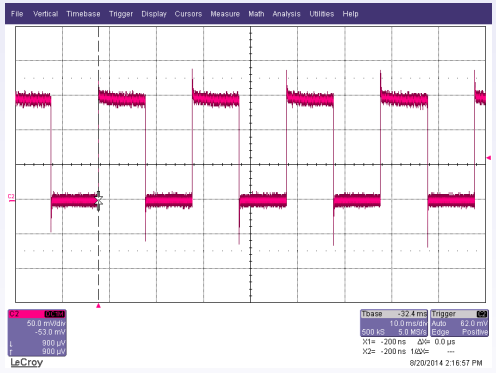
- Le noyau Linux devient une tâche de Xenomai [5] (non-prioritaire)
- Les tâches ont un ordre de priorité ⇒ tâches temps-réel présentent une **latence bornée**

```
int status=0; // declenche' a chaque appel du timer
unsigned char *h;

void test(int signum) {
    red_gpio_output(h,0,status);
    status^=0xff;
}

int main()
{struct sigaction sa;
 struct itimerval timer;

 h=red_gpio_init();
 red_gpio_set_cfgpin(h,0); // LED0

 memset(&sa, 0,sizeof(sa));
 sa.sa_handler = &test;
 sigaction(SIGVTALRM,&sa, NULL);
 timer.it_value.tv_sec = 0; // timer expires after TIMESLEEP us
 timer.it_value.tv_usec = TIMESLEEP;
 timer.it_interval.tv_sec = 0; // and every TIMESLEEP us after
 timer.it_interval.tv_usec = TIMESLEEP;
 setitimer(ITIMER_VIRTUAL, &timer, NULL);
 while(1);
 red_gpio_cleanup();
 return 0;
}
```

En charge



Sans charge



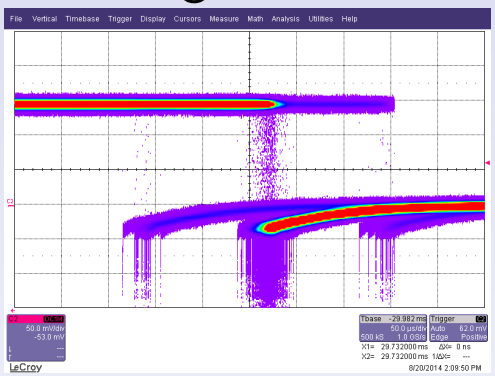
5. P.Ficheux, *Les distributions "embarquées" pour Rasberry PI*, Opensilicium 7 (2013)

Xenomai : extension temps-réel de Linux

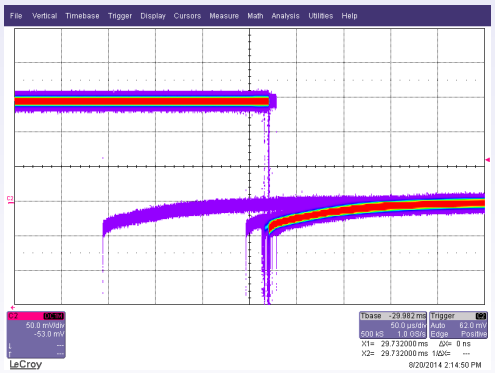
- Le noyau Linux devient une tâche de Xenomai [6] (non-prioritaire)
- Les tâches ont un ordre de priorité ⇒ tâches temps-réel présentent une **latence bornée**

```
int main()
{int status=0;
 unsigned char *h;

 h=red_gpio_init();
 red_gpio_set_cfgpin(h,0); // LED0

 while (1) {
    red_gpio_output(h,0,status); usleep(TIMESLEEP);
    status^=0xff;
 }
 red_gpio_cleanup();
 return 0;
}
```

En charge



Sans charge



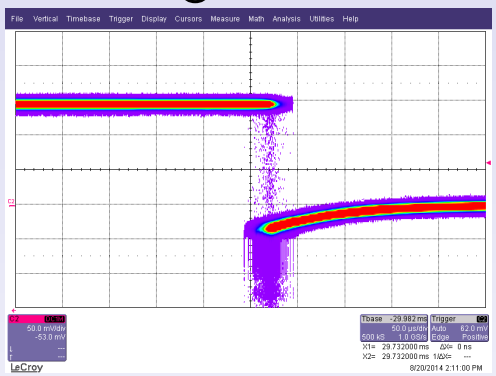
6. P.Ficheux, *Les distributions "embarquées" pour Rasberry PI*, Opensilicium 7 (2013)

Xenomai : extension temps-réel de Linux

- Le noyau Linux devient une tâche de Xenomai [7] (non-prioritaire)
- Les tâches ont un ordre de priorité ⇒ tâches temps-réel présentent une **latence bornée**

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

RT_TASK blink_task;
unsigned char *h;

void blink(void *arg){
 int status=0;
 rt_printf("blink task\n");
 rt_task_set_periodic(NULL, TM_NOW, TIMESLEEP*1000);
 while(1)
   {rt_task_wait_period(NULL);
    red_gpio_output(h,0,status);
    status^=0xff;
    nanosleep(&tim,NULL);
   }
}

int main(int argc, char *argv[])
{h=red_gpio_init();
 red_gpio_set_cfgpin(h,0); // LED0
 mlockall(MCL_CURRENT|MCL_FUTURE); // Avoid memory swapping
 rt_task_create(&blink_task, "blinkLed", 0, 99, 0);
 rt_task_start(&blink_task, &blink, NULL);
 pause();
 rt_task_delete(&blink_task);
 red_gpio_cleanup();
 return 0;
}
```

En charge



Sans charge

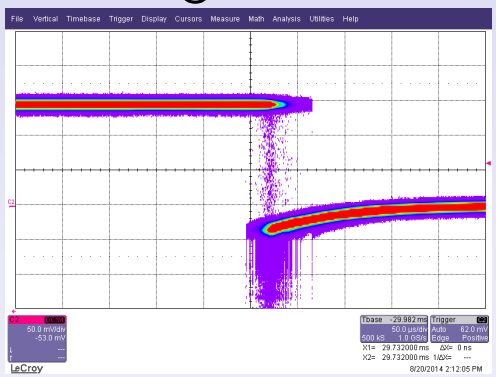
7. P.Ficheux, *Les distributions "embarquées" pour Rasberry PI*, Opensilicium 7

Xenomai : extension temps-réel de Linux

- Le noyau Linux devient une tâche de Xenomai [8] (non-prioritaire)
- Les tâches ont un ordre de priorité ⇒ tâches temps-réel présentent une **latence bornée**

```
#include <native/task.h>
RT_TASK blink_task;
unsigned char *h;

void blink(void *arg){
 int status=0;
 struct timespec tim = {0,TIMESLEEP*1000};
 while(1)
   {red_gpio_output(h,0,status);
    status^=0xff;
    if (nanosleep(&tim,NULL) != 0)
       {printf("erreur usleep\n");return;}
   }
}

int main(int argc, char *argv[])
{h=red_gpio_init();
 red_gpio_set_cfgpin(h,0); // LED0
 mlockall(MCL_CURRENT|MCL_FUTURE); // Avoid memory swapping
 rt_printf("hello RT world\n");
 rt_task_create(&blink_task, "blinkLed", 0, 99, 0);
 rt_task_start(&blink_task, &blink, NULL);
 pause();
 rt_task_delete(&blink_task);
 red_gpio_cleanup();
 return 0;
}
```

En charge

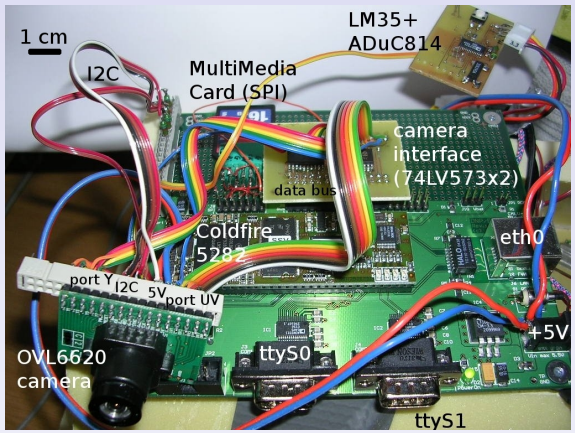


Sans charge

8. P.Ficheux, *Les distributions "embarquées" pour Rasberry PI*, Opensilicium 7 (2013)

Le temps réel : application

Exemple de l'acquisition d'une image :
sur un OS multitâches, il faut couper toutes les interruptions pendant la capture des pixels ⇒ perte du scheduler (monotâche) et de la communication réseau.



Exemple d'une carte son implémentée de façon logicielle
→ exploitation de la Queued ADC (tampon) [9]

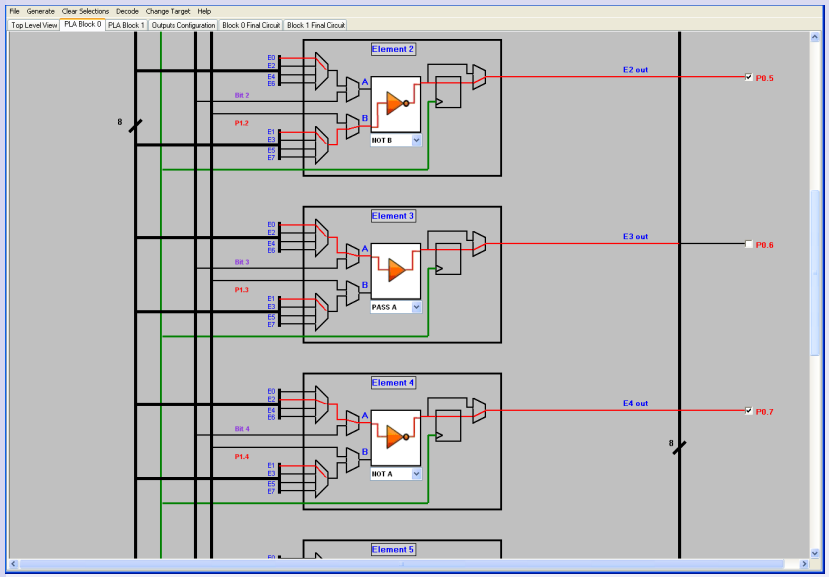
9. S. Guinot, J.-M Friedt, *La réception d'images météorologiques issues de satellites : utilisation d'un système embarqué*, GNU/Linux Magazine France, Hors Série 24 (Février 2006)

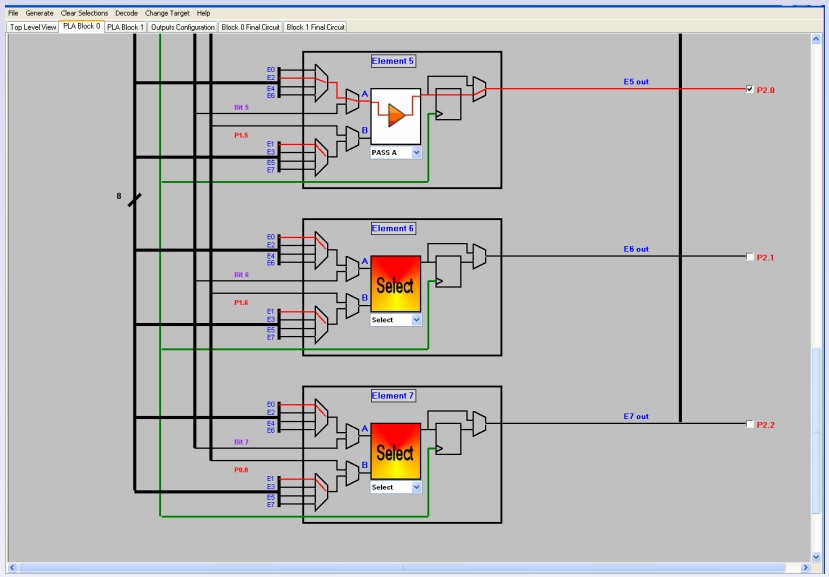
Au dela des processeurs ...

- Ajouter des périphériques pour décharger les processeur : coprocesseurs dédiés pour décharger le processeur de tâches spécifiques complexes (codec audio, ethernet).
- Alternative au temps réel : le matériel reconfigurable se charge des opérations rapides et décharge le processeur de tâches immédiates (UART, FIFO, ADC du coldfire par exemple).
- De plus en plus d'architectures embarquent de la logique programmable
  - Xilinx Zynq,
  - Atera/Intel SoC FPGA,
  - Lattice/Microsemi SmartFusion (ARM),
  - Microsemi PolarFire (RISC-V)

Coprocesseur matériel dans les microcontroleurs

Analog Devices ADuC7026 : quelques cellules de portes logiques





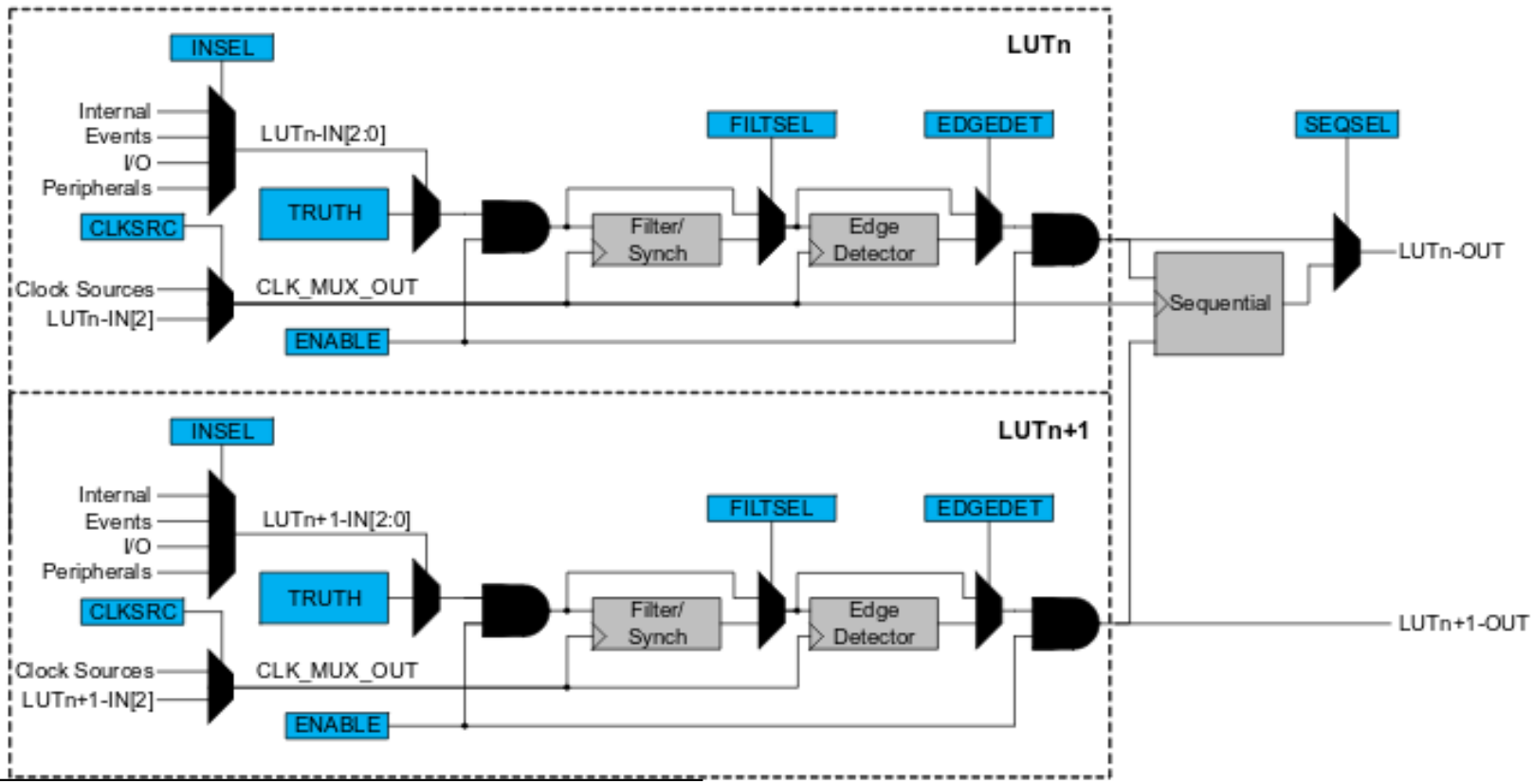
(Re)configurables depuis le firmware

```
        PLACLK   = 0x0003;        // Clk Source configuration
        PLAELM0 = 0x0018;           // output = P0.4, P0.5
        PLAELM1 = 0x00CC;
        PLAELM2 = 0x000A;
        PLAELM3 = 0x0058;
        PLAELM4 = 0x0246;
        PLAELM5 = 0x0258;
...
        PLADIN=0xffff;
```

Coprocesseur matériel dans les microcontroleurs

megaAVR 0-Series CCL (Configurable Custom Logic) [10] [11]

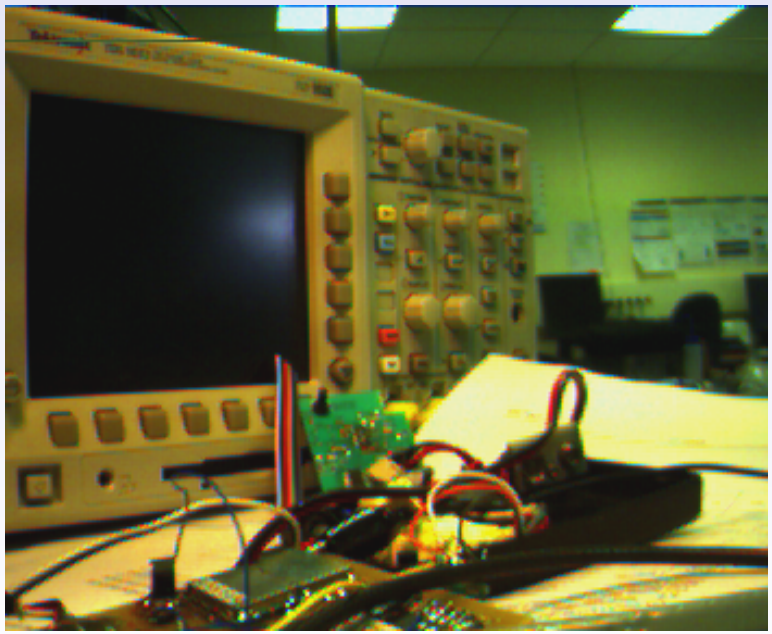
Figure 26-1. Configurable Custom Logic

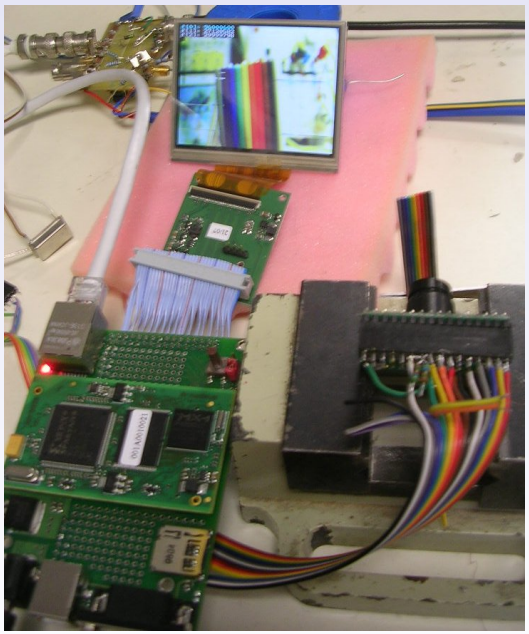


10. ww1.microchip.com/downloads/en/DeviceDoc/40002015A.pdf
11. hackaday.com/2018/03/02/new-part-day-atmegas-with-programmable-logic/

Matériel reconfigurable : le FPGA

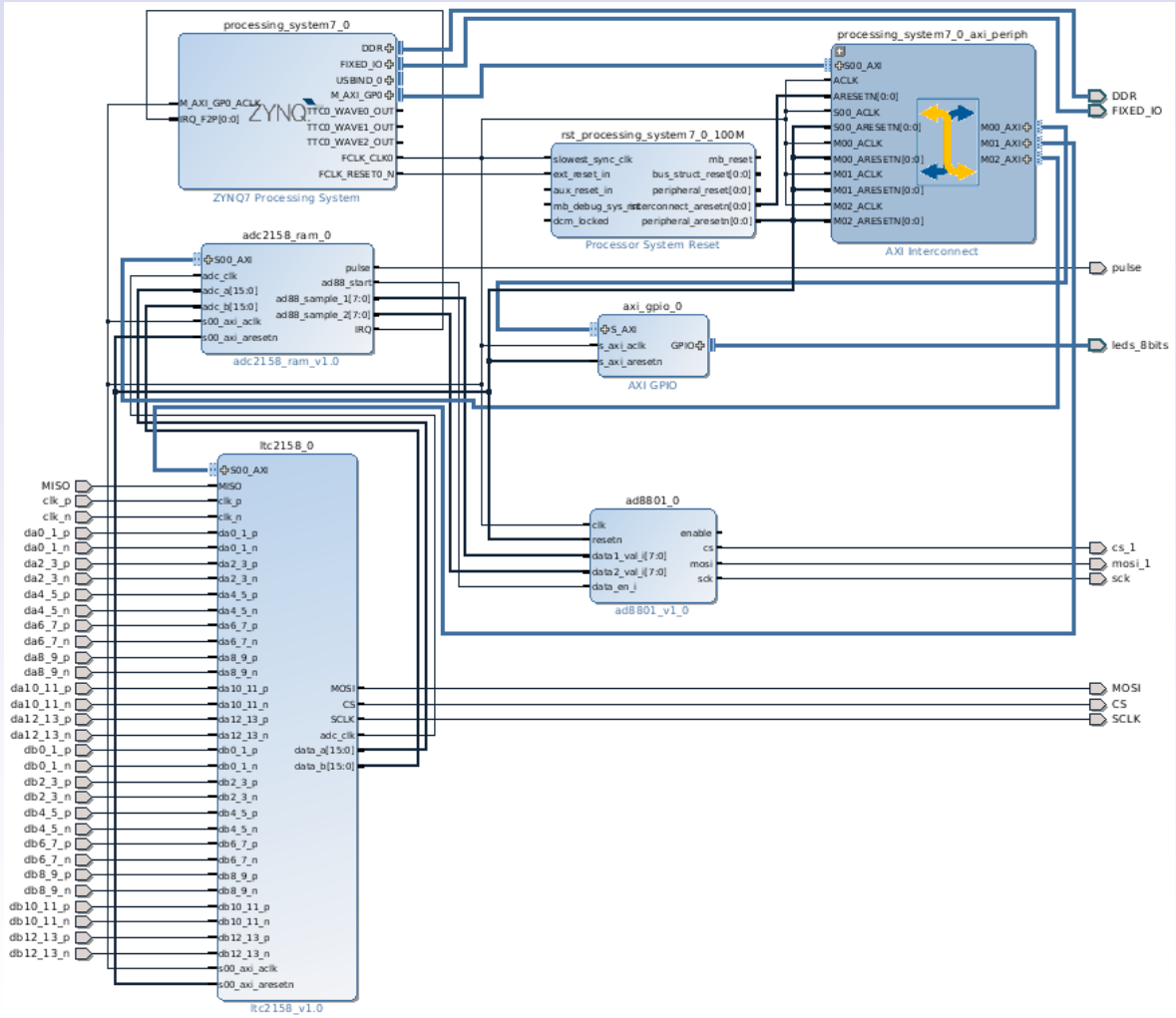
- Le FPGA permet de s'adapter à chaque nouvelle application sans nécessiter d'ajouter du matériel (théorique : mise en forme des signaux).
- Caméra : le flux vidéo impose la cadence des données
- Une approche CPU seul avec OS nécessite de couper toutes les interruptions (et donc d'arrêter l'OS)
- ici, le FPGA stocke l'image, puis la transfère au CPU





FPGA+CPU

- le CPU est en charge des tâches non-temps réel (réseau, communication homme-machine, pré et post-traitements)
- le FPGA est en charge des tâches respecant un temps réel dur (ici, remplir une FIFO de données d'un ADC rapide) : Xilinx Zynq



Mise en pratique

- Mise en œuvre sur Redpitaya puisque Zynq est supporté par Xenomai
- Compiler son environnement de travail (`buildroot`) en fournissant une image de noyau adaptée à Xenomai : `https://github.com/trabucayre/redpitaya` configuré par `make redpitaya_xenomai_defconfig` sur une archive 2018.08.1 de `buildroot`
- Compiler ses applications pour tester le concept de temps-réel

Gestion des GPIO : exploiter `jmfriedt.free.fr/ledlib.c` et `ledlib.h`

- `unsigned char* red_gpio_init();` retourne le pointeur sur la zone mémoire du contrôleur GPIO
- `void red_gpio_set_cfgpin(unsigned char *, int );` définit la direction d'un GPIO
- `void red_gpio_output(unsigned char *,int ,int );` définit l'état (3ème arg) d'une broche (2nd arg) en sortie

Temporisation :

- Sous GNU/Linux : `usleep(int);` ou `sigalarm`
- sous Xenomai : `int nanosleep(const struct timespec*,NULL);`
- Création tâche temps-réel : `int rt_task_spawn(RT_TASK*, const char*,int,int,int, void(*)(*cookie),void*);` avec en argument les entiers que sont la tâche, son nom (NULL), taille de la pile (0 pour automatique), priorité (élevée pour plus prioritaire, 99), mode (`T_JOIGNABLE`), pointeur sur la fonction, argument éventuel à la fonction (NULL)
- Lancer l'ordonnaceur Xenomai : `int rt_task_join (RT_TASK *);`