Électronique numérique

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

10 février 2017

Plan des interventions :

7 cours/TP (séances de 4 h) :

1. Exploitation d'un signal échantillonné en temps discret
   fréquence d'échantillonnage et filtrage
2. Quantification des coefficients, risques de dépassement de capacité.
   Conception et mise en œuvre d'un filtre FIR
3. Environnements exécutifs : FreeRTOS, RTEMS, Nuttx
4. FreeRTOS, RTEMS, Nuttx
   Multitâche et méthodes associées pour garantir l'intégrité des données partagées
5. Système embarqué sous GNU/Linux – système d'exploitation compatible POSIX
   Architecture d'un système d'exploitation, rôle du noyau
   Connectivité internet et configuration réseau
6. Accès aux ressources matérielles depuis l'espace utilisateur (MMU) - - `/dev/mem`
7. Accès aux ressources matérielles depuis un serveur

Pilote noyau de type caractère (*char device*) ?

Accès aux ressources matérielles en espace utilisateur

- Sur microcontrôleur : `*(type*)adresse=valeur;`
- Problème de l'accès au matériel par `/sys` : excessivement long (140 μs pour accéder à `/sys/class/gpio` !)
- Accès en C directement dans la mémoire : plus rapide ...
- mais nécessité de traduire adresse virtuelle en adresse rélle.
- *Memory Management Unit* : superviseur des accès mémoire, virtualisation de l'espace d'adressage (tous les processus ont les mêmes plages d'adresses virtuelles)
- mémoire accessible en espace utilisateur (`root`) par `/dev/mem`
- Le plus rapide : module noyau, mais même problème de MMU.

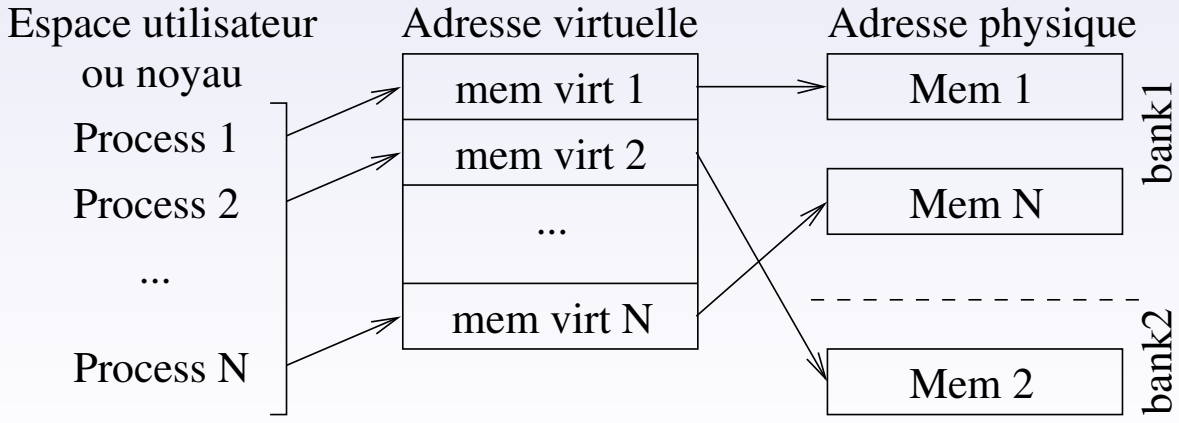
Mémoire virtuelle/mémoire matérielle

Mémoire matérielle

- mémoire matérielle : une adresse sur le bus d'adresse pour identifier un périphérique
- chaque périphérique décode le bus d'adresse pour savoir si le message lui est destiné
- un seul périphérique par adresse physique (sinon, conflit)

Mémoire virtuelle

- chaque processus a sa plage d'adresses
- organisation de la mémoire indépendante des contraintes physiques
- MMU : traduction entre mémoire matérielle et virtuelle



Accès au matériel depuis l'espace utilisateur

Au travers de `/dev/mem`

- avantage : ne pas passer par le noyau (rapide)
- inconvénient : ne pas passer par le noyau (pas de gestion de l'accès aux ressources)

```
#include <fcntl.h>
#include <sys/mman.h>
#define MAP_SIZE 4096UL
#define MAP_MASK (MAP_SIZE - 1)

int main(int argc, char **argv) {
 int fd; void *map_base, *virt_addr; unsigned long read_result, writeval;
 off_t target;
 int access_type = 'w'; // Args : { address } [ type [ data ] ]
 target = strtoul(argv[1], 0, 0);
 if(argc > 2) access_type = tolower(argv[2][0]);
 fd = open("/dev/mem", O_RDWR | O_SYNC);
 map_base = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, target & ~MAP_MASK);
 virt_addr = map_base + (target & MAP_MASK);
 switch(access_type) { case 'b': read_result = *((unsigned char *) virt_addr); break;
                       case 'h': read_result = *((unsigned short *)virt_addr); break;
                       case 'w': read_result = *((unsigned long *) virt_addr); break;
 }
 printf("Value at address 0x%X (%p): 0x%X\n", target, virt_addr, read_result);
 if(argc > 3) {
  writeval = strtoul(argv[3], 0, 0);
  switch(access_type) { case 'b': *((unsigned char *) virt_addr) = writeval; break;
                        case 'h': *((unsigned short *)virt_addr) = writeval; break;
                        case 'w': *((unsigned long *) virt_addr) = writeval; break;
               }       }
 munmap(map_base, MAP_SIZE); close(fd); return 0;
}
```

Mémoire virtuelle/mémoire matérielle

Problème identique au niveau du noyau : fonction `ioremap` après avoir réservé une plage d'adresses.

```
#include <linux/io.h>                  // ioremap
#define IO_BASE 0x01c20800
```

et dans la fonction d'initialisation

```
if (request_mem_region(IO_BASE,36*9,"GPIO test")==NULL)
   printk(KERN_ALERT "mem request failed");
jmf_gpio=(u32)ioremap(IO_BASE, 36*9); // 36 octets/port, A-I
writel(0x10,jmf_gpio+36*6+0x04);
writel(0x0200,(void*)(jmf_gpio+36*6+0x10));
```

Pour libérer la ressource :

```
release_mem_region(IO_BASE, 36*9);
```

Adresses réelle-virtuelle

Équivalent de /dev/mem et mmap en espace utilisateur : ici aussi il faut demander à la MMU de nous fournir le bloc qui pointera vers l'adresse physique :

```
request_mem_region (IO_BASE, 36*9, "GPIO test" );// PG9
jmf_gpio = ( u32 )ioremap(IO_BASE, 36*9);      // 36 bytes/port, ports A-I
writel (0x10, jmf_gpio+36*6+0x04 );            // GPIO PG9 en sortie
```

30.2. Port Register List

Module Name	Base Address
PIO	0x01C20800

Register Name	Offset	Description
Pn_CFG0	n*0x24+0x00	Port n Configure Register 0 (n from 0 to 9)
Pn_CFG1	n*0x24+0x04	Port n Configure Register 1 (n from 0 to 9)
Pn_CFG2	n*0x24+0x08	Port n Configure Register 2 (n from 0 to 9)
Pn_CFG3	n*0x24+0x0C	Port n Configure Register 3 (n from 0 to 9)
Pn_DAT	n*0x24+0x10	Port n Data Register (n from 0 to 9)
Pn_DRV0	n*0x24+0x14	Port n Multi-Driving Register 0 (n from 0 to 9)
Pn_DRV1	n*0x24+0x18	Port n Multi-Driving Register 1 (n from 0 to 9)
Pn_PUL0	n*0x24+0x1C	Port n Pull Register 0 (n from 0 to

Base address, espacement des registres ... : A10 User Manual à https://dl.linux-sunxi.org/A10/

Correspondance hard-soft

1. Identifier la broche sur le connecteur GPIO
2. Identifier sa nomenclature carte (Olimex)
3. Identifier sa nomenclature CPU (A13)
4. Trouver les adresses des registres de configuration de la broche dans la datasheet CPU

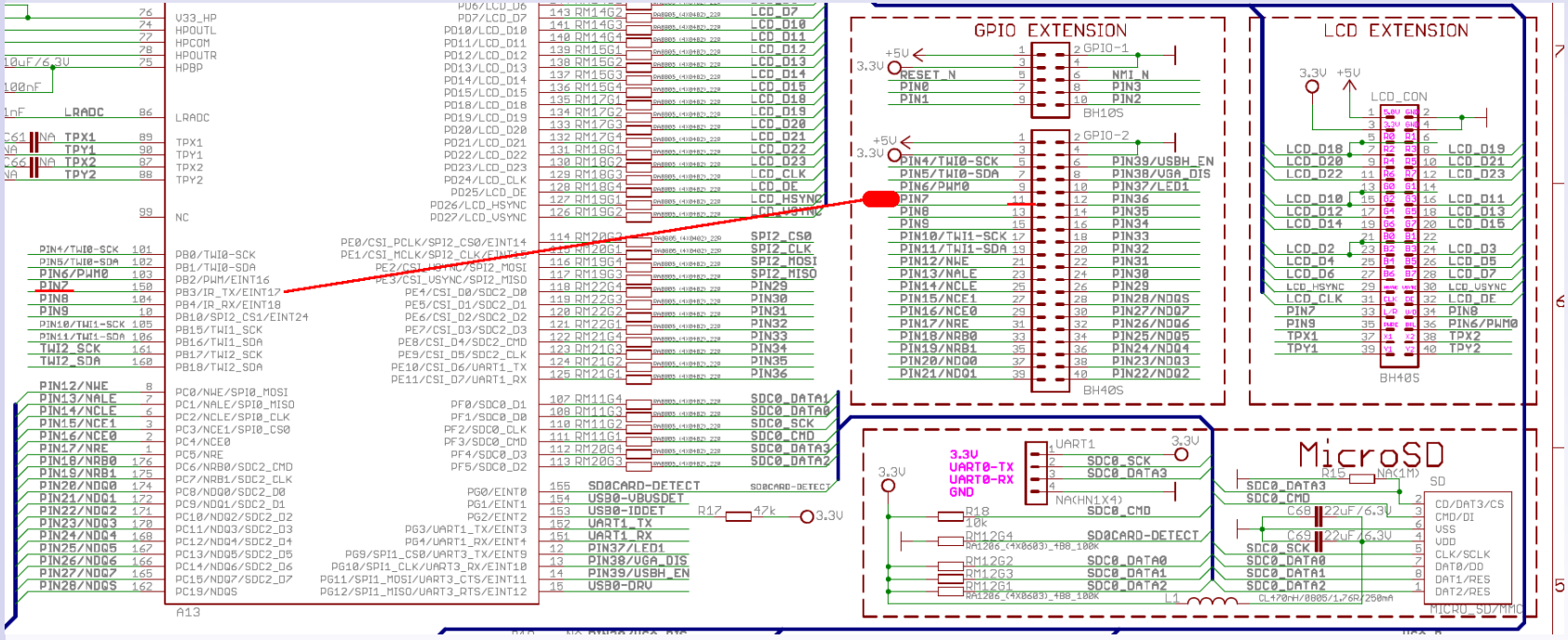


Schéma de la carte Olinuxino A13 :

https://github.com/OLIMEX/OLINUXINO/tree/master/HARDWARE/A13-OLinuXino

Adresse GPIO

- Cross-compilation à partir du compilateur fourni par buildroot,
- Description du matériel dans le User Manual[1] du processeur, et en particulier section 30 pour les GPIOs.
- Adresse de base des GPIOs : 0x01C20800
- 0x24=36 : 36 registres par port
- la fonction de chaque port est décrite pp.288–
- implémentation dans `gpio_lib` en espace utilisateur à `http://dl.cubieboard.org/software/libs/gpio.tar`

```
int sunxi_gpio_init(void) {
...
    fd = open("/dev/mem", O_RDWR);
    PageSize = sysconf(_SC_PAGESIZE);
    PageMask = (~(PageSize-1));
    addr_start = SW_PORTC_IO_BASE & PageMask;
    addr_offset = SW_PORTC_IO_BASE & ~PageMask;
    gpio_map = (void *)mmap(0, PageSize*2, PROT_READ|→
        ↪PROT_WRITE, MAP_SHARED, fd, addr_start);
}
```

1. http://dl.linux-sunxi.org/A10/A10%20User%20Manual%20-%20v1.20%20(2012-04-09,%20DECRYPTED).pdf

Adresse GPIO

- Cross-compilation à partir du compilateur fourni par buildroot,
- Description du matériel dans le User Manual du processeur, et en particulier section 30 pour les GPIOs.
- Adresse de base des GPIOs : 0x01C20800
- 0x24=36 : 36 registres par port
- la fonction de chaque port est décrite pp.288–
- implémentation dans `gpio_lib` en espace utilisateur à `http://dl.cubieboard.org/software/libs/gpio.tar`

```
struct sunxi_gpio {
    unsigned int cfg[4];
    unsigned int dat;
    unsigned int drv[2];
    unsigned int pull[2];
};
```

(4+2+2+1)*sizeof(int)=36 octets/GPIO

```
struct sunxi_gpio_reg {
    struct sunxi_gpio gpio_bank[9];
    unsigned char res[0xbc];
    struct sunxi_gpio_int gpio_int;
};
struct sunxi_gpio *pio =&((struct sunxi_gpio_reg *)→
    ↪SUNXI_PIO_BASE)->gpio_bank[bank];
```

Conclusion

Bien que nous soyons sur un système d'exploitation, nous pouvons toujours accéder au matériel, sous réserve de maîtriser la MMU.

Mise en pratique :

1. cross-compilation en exploitant le compilateur fournit par buildroot
2. manipuler un GPIO par `/dev/mem`
3. serveur TCP/IP permettant la commande à distance

`http://jmfriedt.free.fr/a13_userspace.pdf`