Systèmes embarqués 1/7

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

7 septembre 2017

Rappels

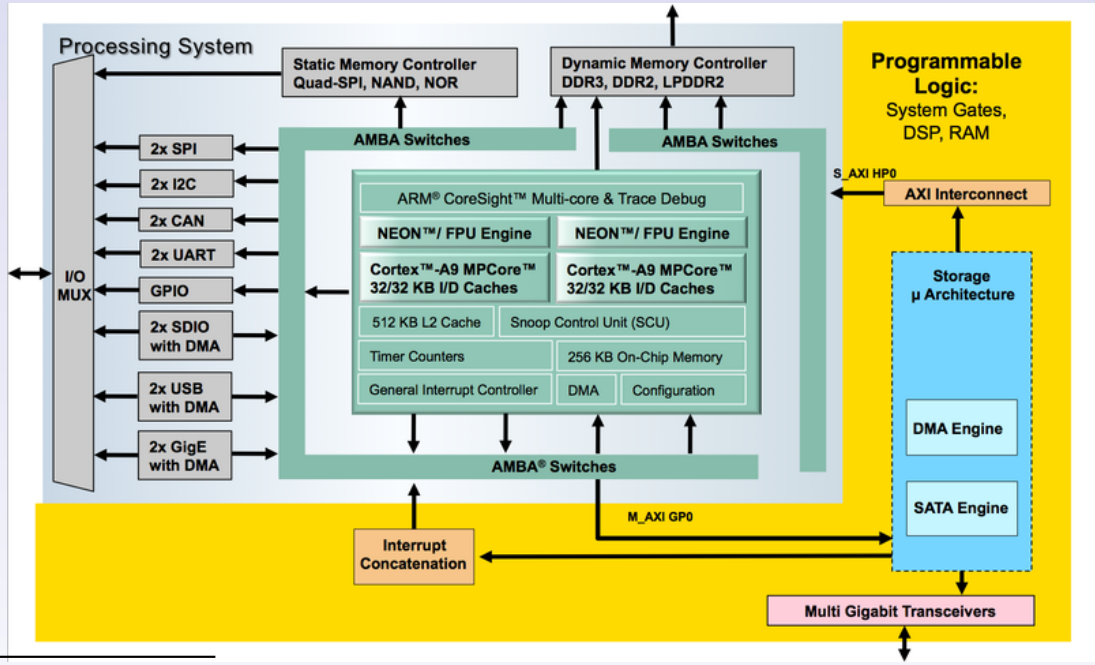
- Travail sur microcontrôleur : pas de processus, un programme en C converti en assembleur s'exécute sur le processeur
- Unique risque de perturber l'exécution séquentielle du programme : interruptions
- Aucune abstraction, écriture directe dans les registres de configuration du matériel
- Aucun mécanisme de partage des ressources ou de protection de la mémoire entre tâches
- $\Rightarrow$ environnements exécutifs (FreeRTOS, RTEMS, NuttX) donnent l'impression de ces mécanismes dans un exécutable statique
- Système d'exploitation (Linux) :
  - environnement portable avec interfaces standardisées (POSIX)
  - mécanismes d'abstraction du matériel (pilotes, `/dev` et `/sys/bus`)
  - notion de processus, chaque tâche possède sa mémoire
  - mécanisme d'abstraction des plages mémoires : MMU

Plan

1. Transition M1–M2 : rappels
   pourquoi GNU/Linux sur système embarqué ?
   Cross-compilation et architecture de buildroot. Réseaux IP. Contrôle du matériel depuis l'espace utilisateur (absence d'arbitration par le noyau), lien entre mémoire virtuelle et matérielle
2. Programmation en espace noyau – pilote dédié et module noyau chargé dynamiquement. Communication par l'interface `/dev` + `ioctl()`, mécanismes fournis par le noyau (timer)
3. Diverses interfaces utilisateur-noyau : `/dev` → plateformes `/sys/bus`, IIO
4. Description du matériel par le *devicetree*, accès aux informations depuis un module noyau, mécanismes fournis par le noyau (sémaphores, tâches dans l'ordonnanceur)
5. Accès à des ressources du FPGA depuis le processeur
6. sources de latences, applications temps-réel, extension Linux au temps réel : Xenomai
7. gestion des interruptions matérielles

Plateforme Redpitaya

- Xilinx Zynq [1] : architecture matérielle comprenant un processeur généraliste (PS) et un FPGA (PL)
- $\rightarrow$ tirer le meilleur parti des deux architectures (interface utilisateur, réseau, algorithmes complexes en PS, grande bande passante et calcul parallèle dans PL)
- bus AXI [2]
- 2 ADC 125 MS/s
- 2 DAC 125 MS/s
- ADCs lents, GPIO, USART, USB, ethernet ...

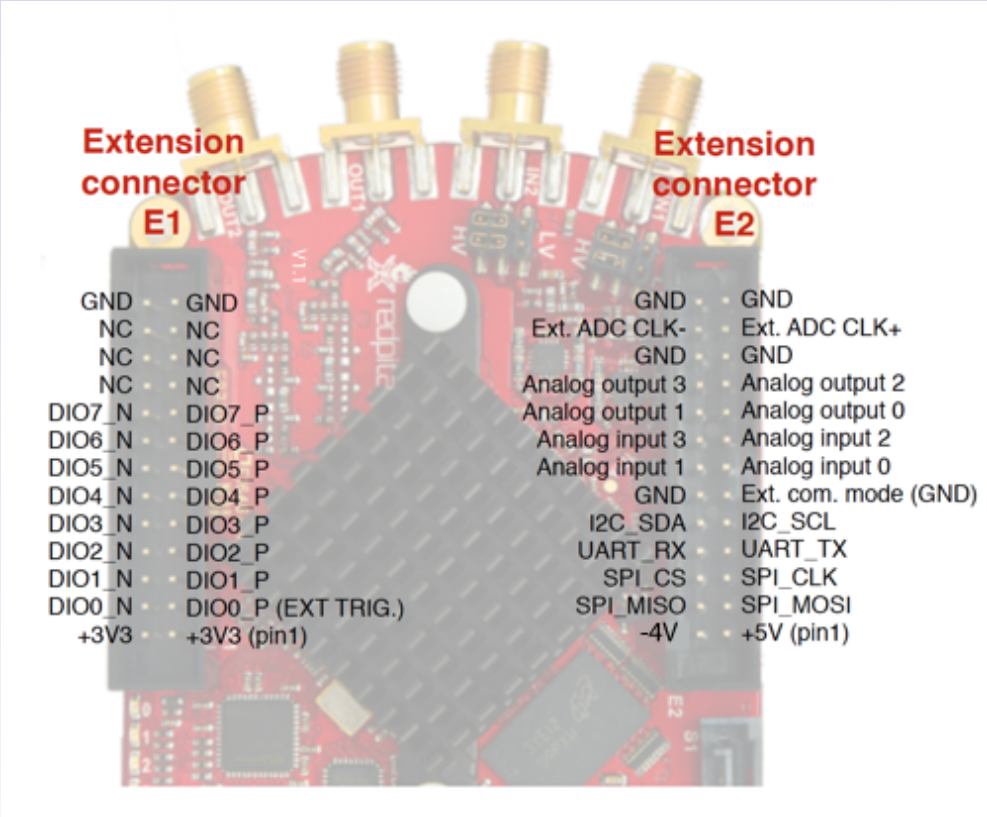


1. `www.xilinx.com/products/silicon-devices/soc/zynq-7000.html`, concurrent de Intel/Atera SOC `www.altera.com/products/soc/overview.html` ou Microsemi/Actel SmartFusion liant FPGA et Cortex-M3 `www.microsemi.com/products/fpga-soc/soc-fpgas`
2. 32 bits @ 150 MHz = 1.2 GB/s

Plateforme Redpitaya

Connecteur E2 (1 bord de carte coté SATA) [3]

Pin	Description
1	+5V
2	-3.4V (50mA)1
3	SPI(MOSI) PS_MIO10
4	SPI(MISO) PS_MIO11
5	SPI(SCK) PS_MIO12
6	SPI(CS#) PS_MIO13
7	UART(TX) PS_MIO08
8	UART(RX) PS_MIO09
9	I2C(SCL) PS_MIO50
10	I2C(SDA) PS_MIO51
11	Ext com.mode GND (default)
12	GND
13	Analog Input 0 0-3.5V
14	Analog Input 1 0-3.5V
15	Analog Input 2 0-3.5V
16	Analog Input 3 0-3.5V
17	Analog Output 0 0-1.8V
18	Analog Output 1 0-1.8V
19	Analog Output 2 0-1.8V
20	Analog Output 3 0-1.8V
21	GND
22	GND
23	Ext Adc CLK+
24	Ext Adc CLK-
25	GND
26	GND



MIO connecté au PS, EMIO au PL

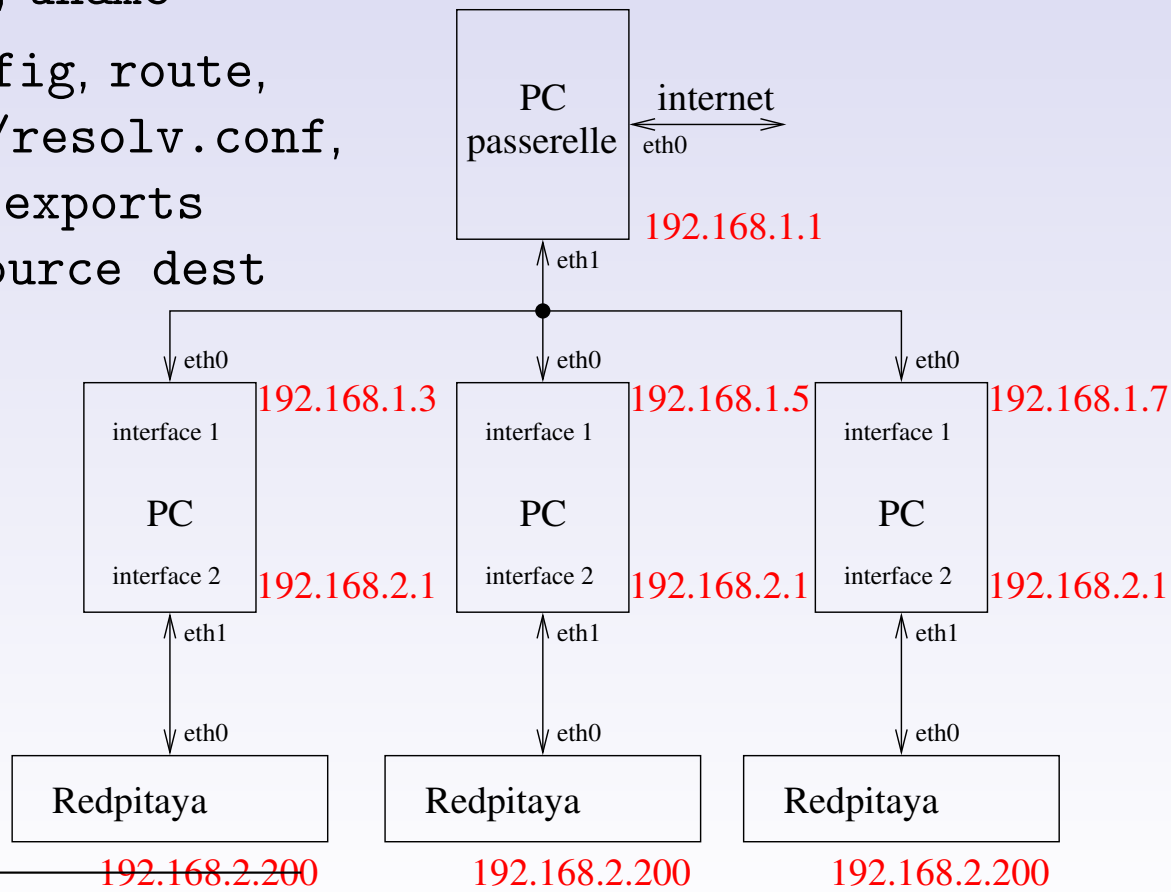
3. redpitaya.readthedocs.io/en/latest/developerGuide/125-14/extent.html

Objectif du cours

- voir ensemble comment accéder depuis Linux aux ressources du FPGA (LEDs)
- nécessite de
  - cross-compiler,
  - accéder aux ressources matérielles depuis Linux (on commence par les LEDs connectées au processeur),
  - configurer le FPGA depuis GNU/Linux et informer le noyau des ressources requises
  - maîtriser les divers interfaces entre espaces noyau et utilisateur de Linux
- vous refaites sur le convertisseur analogique-numérique (XADC)

Rappel des commandes utiles et architecture réseau

- Modules noyau : `lsmod`, `insmod`, `rmmod`, `modprobe`, modules dans `/lib/modules`
- système : `dmesg`, `uname`
- réseau [4] : `ifconfig`, `route`, DNS dans `/etc/resolv.conf`, NFS dans `/etc/exports`
  ⇒ `mount IP:source dest`



4. `ssh root@192.168.2.200` : pas de compte utilisateur, `nobody` pour serveurs

Accès au matériel depuis l'espace utilisateur

Au travers de `/sys/class`

- échange d'informations en trames ASCII
- configuration au travers du fichier approprié (pas de `ioctl`)
- particulièrement approprié pour une interaction avec l'utilisateur (programmation *shell*)

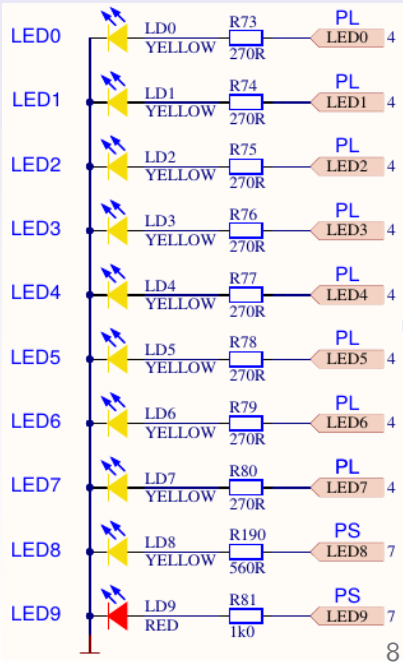
```
# echo "1" > /sys/class/gpio/gpio906/value
```

Attention en C : `fseek` à l'offset 0 en `SEEK_SET` pour plusieurs accès sans refaire `fopen` et `fclose`

Sur Redpitaya, indice du MIOx est 906+x

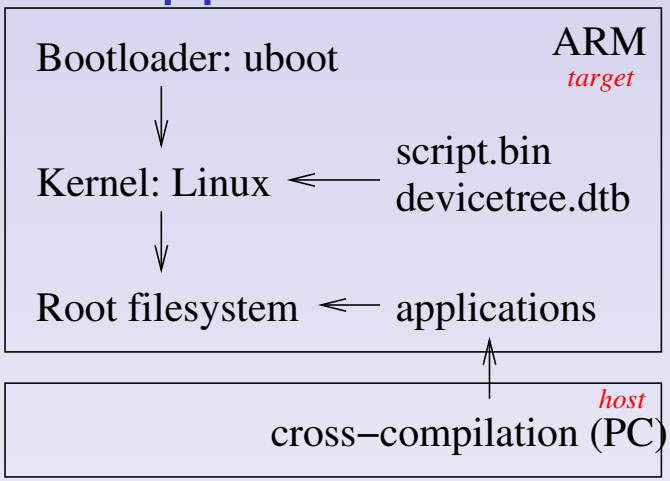
The default pin assignment for GPIO is described in the next table.

FPGA	con-nector	GPIO	MIO/EMIO index	`sysfs` index	color, dedicated meaning
		`exp_p_io [7:0]`	`EMIO[15: 8]`	`906+54+[15: 8]=[975:968]`	
		`exp_n_io [7:0]`	`EMIO[23:16]`	`906+54+[23:16]=[983:976]`	
		LED `[7:0]`	`EMIO[ 7: 0]`	`906+54+[ 7: 0]=[967:960]`	yellow
		LED " [8]"	`MIO[ 0]`	`906+ [ 0] = 906`	yellow = CPU heartbeat (user defined)
		LED " [9]"	`MIO[ 7]`	`906+ [ 7] = 913`	red = SD card access (user defined)



Rappels sur l'environnement de développement

- Buildroot = environnement homogène pour fournir chaîne de compilation croisée, bibliothèques, noyau et exécutables
- host=PC x86, target=Redpitaya ARM
- `output/host/usr/bin` contient les outils pour travailler sur PC



- `output/build` contient les sources des exécutables disponibles sur la cible
- dans `output/build`, le répertoire `linux-3f3c7b60919d56119a68813998d3005bca501a40` contient le noyau
- principe du `BR2_EXTERNAL` pour compléter l'archive officielle de buildroot par le support de la Reditaya
- support Redpitaya décrit dans `jmfriedt.free.fr/redpitaya.pdf` qui s'appuie sur `github.com/trabucayre/redpitaya.git`

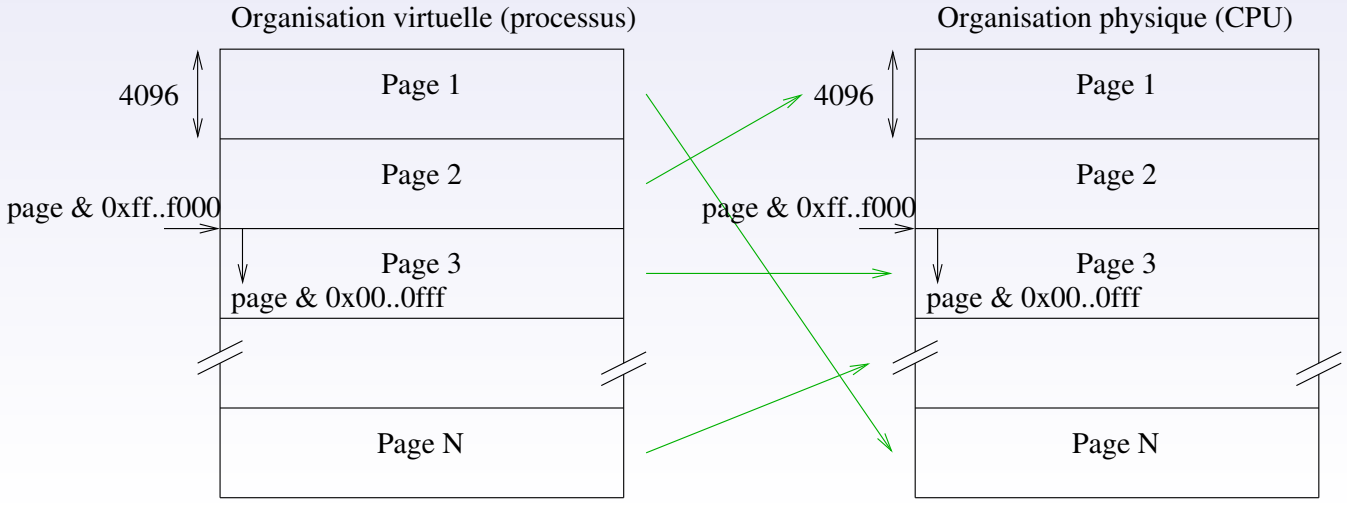
Mémoire virtuelle/mémoire matérielle

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
- chargement dynamique des exécutables et bibliothèques associées
- 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          // MMU page size
#define MAP_MASK (MAP_SIZE-1)    // mask

int main(int argc, char **argv) {
 int fd; void *map_base, *virt_addr;
 unsigned long read_result, writeval;
 off_t target=0x12345678;        // physical @
 fd = open("/dev/mem", O_RDWR | O_SYNC); // MMU access
 map_base=mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, \
                    MAP_SHARED, fd, target & ~MAP_MASK);
 virt_addr=map_base+(target & MAP_MASK); // virt. @
 read_result=*((unsigned long *)virt_addr); // read mem
 printf("0x%X (%p): 0x%X\n",target,virt_addr,read_result);
 // *((unsigned long *) virt_addr) = writeval; // write
 munmap(map_base, MAP_SIZE); close(fd); return 0;
}
```

Outil devmem

- Prototypage rapide de l'accès aux registres du processeur
- Mis à disposition par busybox [5]
- Utilisation

```
Read/write from physical address
        ADDRESS Address to act upon
        WIDTH   Width (8/16/...)
        VALUE   Data to be written
```

5. $BUILDROOT/output/build/busybox-1.27.1/miscutils/devmem.c

Mise en pratique

- Deux LEDs sont connectées à MIO0 et MIO7 : les faire clignoter depuis le shell au travers de `/sys/class/gpio`
- Compiler un programme affichant "Hello World" et l'exécuter sur Redpitaya (connexion `ssh` et montage `nfs`)
  Buildroot se trouve dans
  `/home/jmfriedt/buildroot-2017.05.2` : exporter le `PATH`
- Identifier dans la documentation technique [6] les registres nécessaires à configurer ces GPIOs, et proposer un programme en C pour faire clignoter ces LEDs. Quelles opérations effectuer sur les GPIOs pour les rendre fonctionnels ?
- Commander l'allumage/extinction des LEDs par une page web (serveur `boa` sur Redpitaya).
- Intégrer la commande des LEDs dans un serveur TCP/IP (`socket` → `bind` → `listen` → { `accept` → `recv`/`send` → `close`})

6. `www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf`