



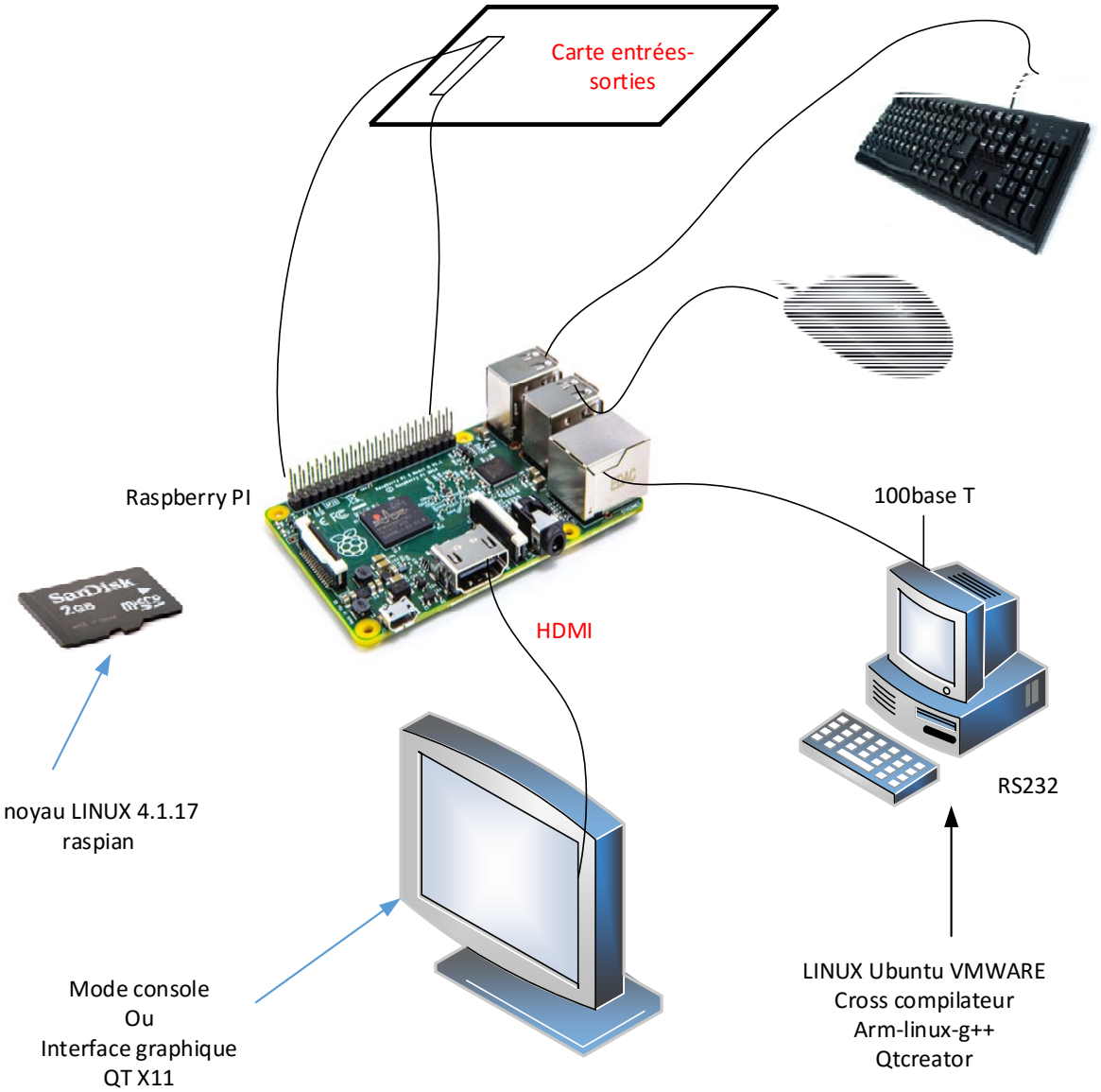
TP : INFORMATIQUE INDUSTRIELLE N°8
L'ordonnancement des taches sous LINUX

L'ordonnancement des taches sous LINUX

1) But

Découvrir mesurer et évaluer les caractéristiques multitâches multicoeur et temps réel du système d'exploitation LINUX.

2) Schéma de câblage



3) Cahier des charges

On souhaite découvrir et mesurer les caractéristiques multitâches multicoeur et temps réel du système d'exploitation LINUX. Pour ce faire nous allons utiliser des programmes simples permettant de piloter les LED du module entrées sorties. Les LED se trouvent sur la carte entrées sorties

4) Questions

Le fichier LED.c permet de faire clignoter la LED de votre choix

a) A partir de ce fichier source tempspartage/LED/LED.c, répondez à la question suivante : utilise-t-on les appels systèmes pour piloter le port GPIO et les LEDS

b) Décrire le rôle de chaque instruction du fichier source LED.c faisant appel à la fonction mmap()

5) Mesure

5 .1) Etude du mode temps partagé

a) Dans le répertoire tempspartage/LED Lancez votre applicatif LED pour faire clignoter la LED1 et mesurez la période de commutation de la LED N°1

✍

b) Lancez en utilisant une autre console le clignotement de la LED N°2 et la LED N°3 en même temps que la LED N°1.

Mesurez la période d'exécution d'un processus permettant de faire clignoter une LED?

✍

c) Observez les 2 signaux sur LED1 LED2 et LED3, conclure.

✍

d) Dans le répertoire tempspartage/LED Lancez votre applicatif LED pour faire clignoter la LED1, LED2 LED3 sur le même cœur de CPU, conclure

Sudo taskset –c 0 ./LED

e) Mesurez le changement de contexte entre deux processus (temps de latence)

✍

Le programme LED Noyau permet de mettre en place des appels systèmes vers le noyau via un driver
Afin de piloter le GPIO
Chargez le module gpiosen.ko dans le répertoire drivers afin de pouvoir faire appel aux appels systèmes dans le fichier de caractère spécial /dev/gpio

f) Fermez les programmes précédents, lancez le programme LEDNOYAU (tempspartage/LEDNOYAU) pour la LED N°1 et la LED 2 sur le même CPU observez la période de commutation d'une LED, conclure.

✍

La commande nice permet de choisir le niveau de courtoisie d'un processus

nice –n -1 ./LED cette commande permet de programmer un niveau de courtoisie de -1 pour le processus LED qui sera lancé par la suite. Le programme LED permet aussi de programmer le niveau de courtoisie

Sur le même CPU :

Programmez une courtoisie de -1 pour le processus LED (tempspartage/LED) qui pilote la LED N°1.
Programmez une courtoisie de 0 pour le processus LED qui pilote la LED N°2

sudo taskset -c 0 nice –n -1 ./LED
sudo taskset -c 0 ./LED // le programme nous demande le niveau de courtoisie

g) Lancez les 2 processus, conclure

Programmez une courtoisie de -3 pour le processus LED qui pilote la LED N°1
Programmez une courtoisie de 0 pour le processus LED qui pilote la LED N°2

h) Quel est le rapport entre le temps d'exécution du processus LED1 et LED2, Conclure.

✍

Programmez une courtoisie de -19 pour le processus LED qui pilote la LED N°1
Programmez une courtoisie de -3 pour le processus LED qui pilote la LED N°2

i) Observez et conclure.

✍

Faire un ping vers la carte embarqué Raspberry , y a t'il une réponse de la carte, conclure ?

Le programme pulsenoyau permet d'envoyer 2 impulsions sur la LED N°1. pulsenoyau fait appel aux appels systèmes pour piloter le GPIO grâce aux drivers.

j) Relevez la durée d'un appel système, conclure.

✍

k) Relancez le programme tempspartage/pulsenoyau avec une courtoisie de -19 et relevez la durée d'un appel système, conclure.

Sudo ./LED

5 .2) Etude du mode temps réel mou

On peut lancer un processus dans le mode temps réel mou en utilisant la commande chrt avec 2 algorithmes Round Robin et FIFO priorité de 1 à 99

La commande chrt permet de lancer un processus dans ne mode d'exécution temps réel mou.

chrt –r ou –f priorité ./executable

Exemple

chrt –r 10 ./LED

Dans le dossier tempspartage/LED Lancez dans une première console le programme chrt -r 25 ./led (prendre LED1)
dans une deuxième console le programme chrt -r 26 ./led (prendre LED3)

a) observez.

✍

Lancez dans une première console le programme chrt -f 25 ./led (prendre LED1)
dans une deuxième console le programme chrt -f 26 ./led (prendre LED3)

b) observez et conclure .

✍

On souhaite lancer les deux processus sur le même cœur de CPU

sudo chrt –r 25 taskset –c 0 ./LED

Dans le dossier tempspartage/LED Lancez dans une première console le programme chrt -r 25 ./led (prendre LED1)
dans une deuxième console le programme chrt -r 26 ./led (prendre LED2)

c) observez.

✍

Lancez dans une première console le programme chrt -f 25 ./led (prendre LED1)
dans une deuxième console le programme chrt -f 25 ./led (prendre LED2)

d) observez et conclure .

✍

Lancez dans une première console le programme chrt -f 25 ./led (prendre LED1)
dans une deuxième console le programme chrt -f 26 ./led (prendre LED2)

a) observez et conclure .

✍

Lancez dans une première console le programme chrt -r 25 ./led (prendre LED1) dans une deuxième console le programme chrt -r 25 ./led (prendre LED2)

b) observez et conclure

✍

La commande top permet de connaitre la priorité PR du processus en cours :

c) Pour une priorité de 99 qu'obtient-on pour PR ?

✍

d) Pour une priorité inférieure à 99 (98 par exemple) qu'obtient on pour PR ?

✍

e) Quelle relation lie PR aux paramètres prio et nice

✍

Le programme thrtled permet de lancer au sein d'un même processus deux threads qui vont permettre d'allumer respectivement la LED1 et la LED2.
Ces threads seront lancés en mode temps réel mou avec deux types d'algorithmes et une priorité que l'on pourra changer par la suite.
L'exécutable se situe dans le répertoire tempsreel/thrtled
On peut faire passer des paramètres en ligne de commande

sudo ./thrtled priorité tempsexécution algo

sudo ./thrtled 25 10 1 // priorité 25 temps d'exécution 10 algo 1 round robin

Pour un algo type FIFO prendre 0

f) Lancez la commande suivante observez et conclure

sudo ./thrtled 25 10 0

✍

Dans le programme thrtled.c :

Décommentez la ligne suivante :
param.sched_priority = (i + 1) * 10; // priorité variable
et commentez la ligne suivante
//param.sched_priority = atoi(argv[1]); // priorité constante

g) Compilez , Lancez la commande suivante observez et conclure

sudo ./thrtled 25 10 0

✍

h) Lancez la commande suivante observez et conclure

sudo ./thrtled 25 10 1

✍

Dans le programme thrtled.c :

commentez la ligne suivante :
param.sched_priority = (i + 1) * 10; // priorité variable
et décommentez la ligne suivante
//param.sched_priority = atoi(argv[1]); // priorité constante

i) Compilez , Lancez la commande suivante observez et conclure

sudo ./thrtled 25 10 1

✍SCR96.bmp

5.3) Influence des interruptions en mode temps réel mou

Chargez le module gpiosen.ko du répertoire drivers/moduleint afin de gérer les interruptions sur la raspberry. A chaque interruption (front montant sur DO1) la LED LD1 sera commuté de ON à OFF ou de OFF à ON.
Le temps système correspond au temps passé par LINUX dans le domaine noyau.

a) Appliquez un signal carré (0v 3.3V 10KHz) sur la broche DO1 de la carte entrées-sortie.

Grâce à la commande cat /proc/interrupts visualisez le nombre d'interruption géré par le pilote gpiosen

✍

b) Relevez le signal sur la LED LD1 et mesurez le temps de latence de l'interruption

✍

c) Lancez le processus occup en mode temps réel avec une priorité de 99

chrt -f 99 taskset –c 0 ./occup

Quelle influence le fonctionnement de ce programme a-t-il sur les interruptions.

✍

d) Quelle méthode préconisez-vous pour que notre processus utilisateur puisse prendre la main sur les processus du noyau LINUX

✍

5.4) Etude du mode temps réel dur sous XENOMAI

Le module rtdmirq.ko a été développé dans l'environnement Xenomai
Il permet de prendre en compte une interruption sur la broche gpio4 (D01 sur la carte) et de commuter la LED N°1 ON vers OFF ou OFF vers ON
On insère le module irq-gpio-rtdm.ko
Les interruptions sont alors prises en compte par xenomai :

Le module gpiosen est dans un premier temps déchargé (aucune gestion des interruptions par LINUX)

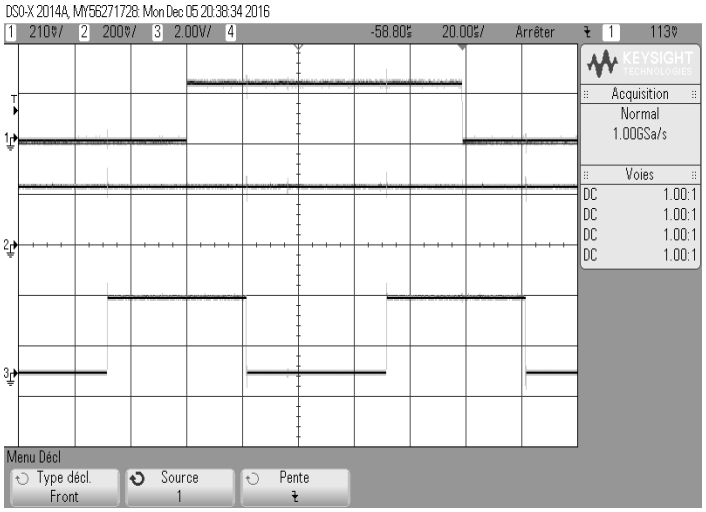
Les interruptions (signal DO1 10KHZ) sont bien prises en compte par xenomai

```
root@beagleboard:/home/xeno1/rtdmirq# rmmod rtdmirq
root@beagleboard:/home/xeno1/rtdmirq# insmod rtdmirq.ko
root@beagleboard:/home/xeno1/rtdmirq# cat /proc/xenomai/irq
IRQ          CPU0
 38:       601463         [timer]
298:       561335         Exemple
418:            0         [virtual]
root@beagleboard:/home/xeno1/rtdmirq#
```

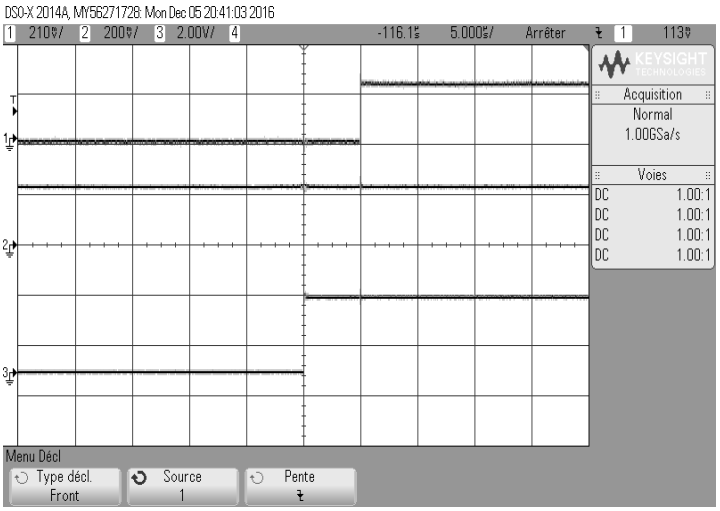
16:08:18 2014
04:36:41 2014
15:16:49 2014
20:40:04 2013
19:42:07 2014
19:37:08 2014
41 items, Free space: 301.2 MB

a) On donne les résultats observés entre la génération de l'interruption et la prise en compte de cette interruption par la commutation de la LED, Relevez le temps de latence de l'interruption et conclure avec les résultats obtenus précédemment sur LINUX non patché.

Pour LINUX non patché



Pour Xenomai



On lance le module qui gère les interruptions par LINUX (insertion de gpiosen.ko) et on observe les interruptions dans les 2 domaines LINUX et xenomai :

Terminal

File Edit View Terminal Help

```
Err:          0
root@beagleboard:/home/GPIO_beagleint# cat /proc/interrupts
           CPU0
  7:          0        INTC  TWL4030-PIH
 11:          0        INTC  prcm
 12:      18769        INTC  DMA
 25:          3        INTC  OMAP DSS
 38:     237717        INTC  gp timer
 56:        311        INTC  omap_i2c
 61:          0        INTC  omap_i2c
 72:          0        INTC  serial idle
 73:          0        INTC  serial idle
 74:       1194        INTC  serial idle, OMAP UART2
 77:      45826        INTC  ehci_hcd:usb1
 80:          0        INTC  serial idle
 83:      20773        INTC  mmc0
167:          0        GPIO  user
298:     414478        GPIO  gpiosen
376:          0     twl4030  twl4030_pwrbutton
378:          0     twl4030  twl4030_usb
379:          0     twl4030  rtc0
384:          0     twl4030  mmc0
Err:          0
root@beagleboard:/home/GPIO_beagleint#
```

Domaine LINUX

Terminal

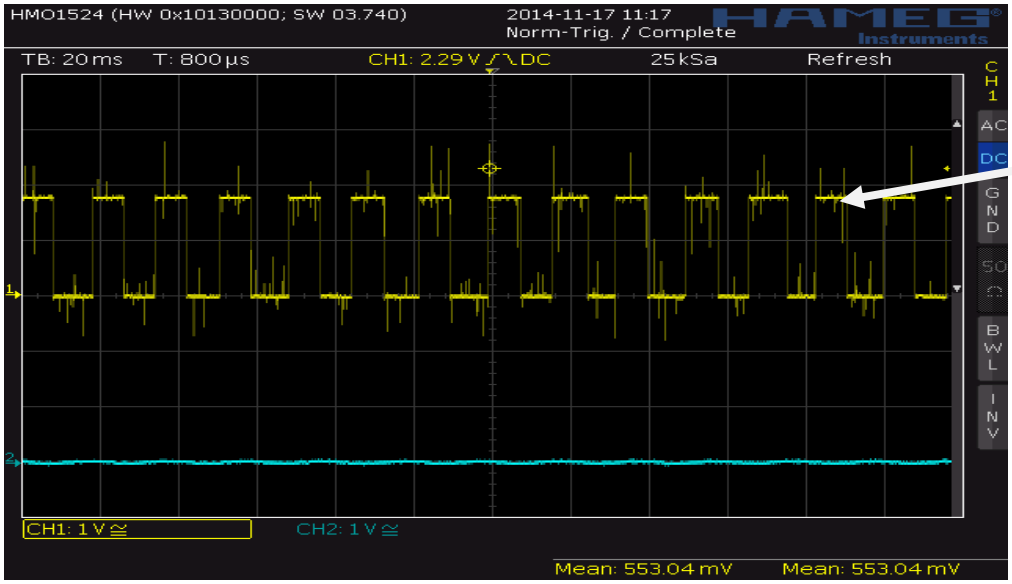
Edit View Terminal Help

```
:      228015         [timer]
:      283391         Exemple
:           0         [virtual]
t@beagleboard:/home/xeno1/rtdmirq# cat /proc/xenomai/irq
           CPU0
:      228352         [timer]
:      284680         Exemple
:           0         [virtual]
t@beagleboard:/home/xeno1/rtdmirq# cat /proc/xenomai/irq
           CPU0
:      228660         [timer]
:      285830         Exemple
:           0         [virtual]
t@beagleboard:/home/xeno1/rtdmirq# cat /proc/xenomai/irq
           CPU0
:      232637         [timer]
:      302442         Exemple
:           0         [virtual]
t@beagleboard:/home/xeno1/rtdmirq# cat /proc/xenomai/irq
           CPU0
:      233155         [timer]
:      304620         Exemple
:           0         [virtual]
t@beagleboard:/home/xeno1/rtdmirq#
```

Domaine xenomai

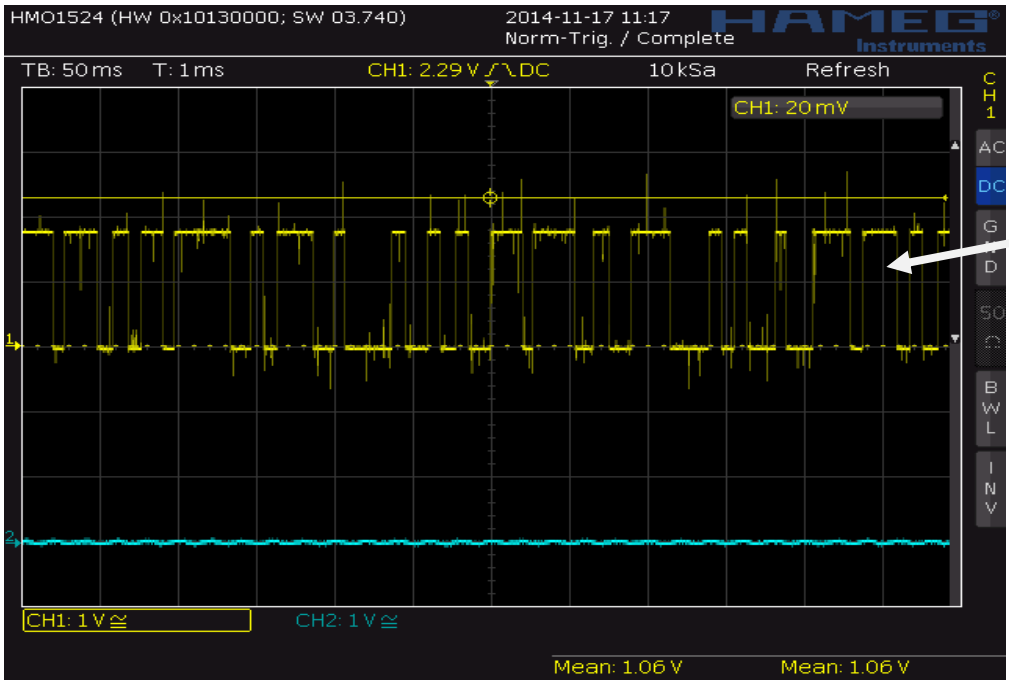
b) Les interruptions n'évoluent pas sous LINUX, expliquez pourquoi ?

Soit le programme xeno1 sous xenomai qui permet d'afficher à l'écran une chaine de caractère et qui s'endort pendant 100ms soit le programme LEDNOYAU1 sous le domaine utilisateur LINUX qui permet de faire clignoter indéfiniment la LED N°1.
On lance au départ le programme LEDNOYAU1 on obtient l'image ci-dessous :



Clignotement de La LED N°1

On lance par la suite xeno1 le programme LEDNOYAU1 est toujours actif on obtient l'image suivante :



c) Commentez les résultats.

6) Préparation

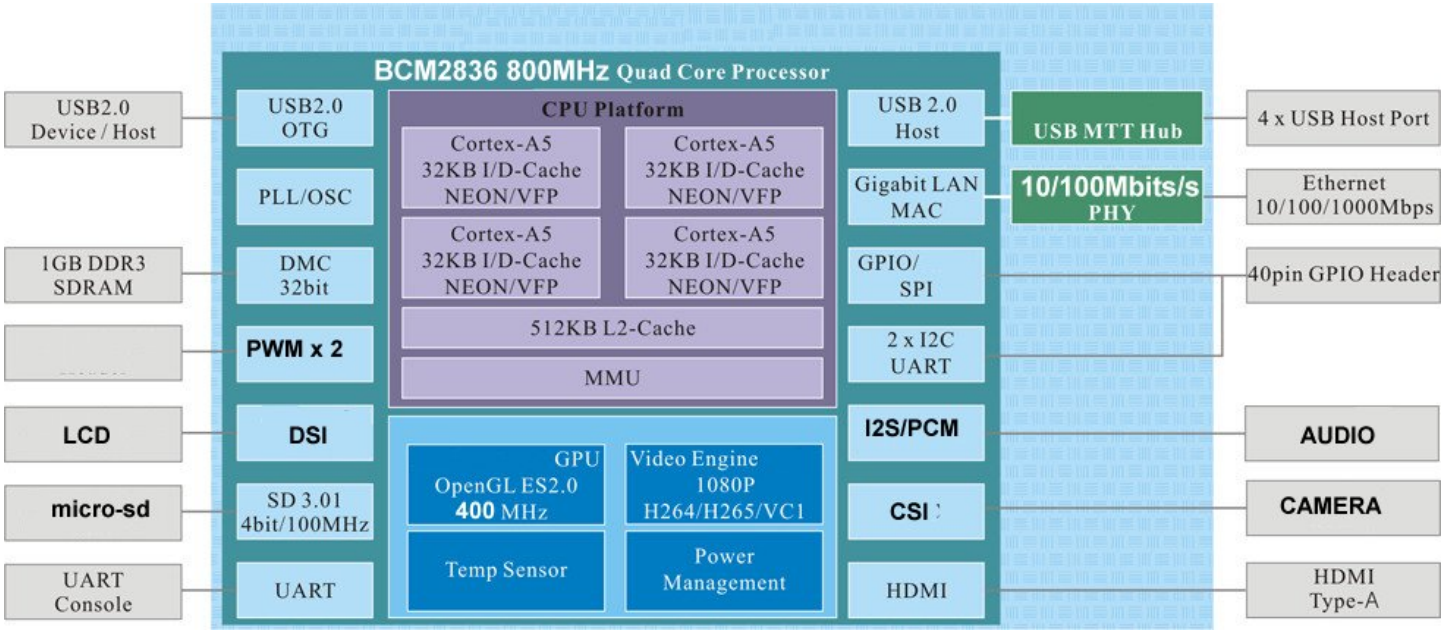
Répondre aux questions 5.4)

Dossier ressource

Lecture et écriture dans les plages d'adresses des périphériques

Un driver de périphérique est un point d'entrée dans le système. Comme les modules des pilotes sont chargés dynamiquement est lié à un noyau existant, une erreur du driver peut engendrer un crash système.

Pour directement accéder aux adresses mémoires des périphériques ou de la mémoire du microprocesseur, il faut connaitre son architecture avec le rôle de tous les registres. La Raspberry dispose d'un SOC BCM2836. Elle est basée sur une architecture ARM v7 à faible consommation orientée notamment vers les applications multimédias pour systèmes embarquées portables et mobiles.
On donne ci-dessous une vue éclatée de l'architecture du BCM2836



Les périphériques sont très nombreux (encadré rouge hormis l'USB) et il faut se référer à la documentation constructeur afin de connaitre le rôle de chacun des registres.

Accès au registre dans l'espace utilisateur

Pour accéder au registre des périphériques du microprocesseur GPP (processeur à usage général) à partir du domaine utilisateur, LINUX met à notre disposition un système de fichier à caractère spécial (/dev/mem). L'ouverture de ce système de fichier nous permettra grâce à la fonction mmap de passer du domaine d'adresse virtuelle au domaine d'adresse réelle et d'atteindre les registres du processeur.

La fonction MMAP

```
void *mmap(void *addr, size_t length, int prot, int flags,int fd, off_t offset);
```

mmap créé un nouveau mapping dans l'espace d'adresse virtuelle du processus appellant.
L'adresse de base pour le nouveau mapping est spécifiée par le paramètre addr. Si l'adresse addr est nulle alors le noyau choisit de lui-même l'adresse de l'espace virtuelle.
Pour atteindre l'espace de mémoire réelle il faut spécifier par le paramètre lenght la longueur de cet espace et par le paramètre offset, l'adresse dont le point de départ est référencé par le descripteur de fichier fd (offset correspondra donc à une adresse physique dans notre GPP).

Cette fonction renvoie un pointeur dont l'adresse correspond à une adresse physique de GPP définit par l'offset.

Le paramètre prot défini la protection en mémoire désirée du mapping (il ne doit pas être en conflit avec la fonction fopen), on donne ci-dessous les indicateurs les plus connus:

```
PROT_EXEC      la plage d'adresse peut être accédée en execution.
PROT_READ      la plage d'adresse peut être accédée en lecture.
PROT_WRITE     La plage d'adresse peut être accédée en écriture.
```

Exemple d'utilisation :

```
volatile ulong *pinconf;
pinconf = (ulong*) mmap(NULL, 0x10000, PROT_READ | PROT_WRITE, MAP_SHARED, fp,
0X48000000);
```

pinconf[0] pointera sur l'adresse 0X48000000.

Chaque case mémoire pointée à pour format 4 octets soit 32 bits (ulong)