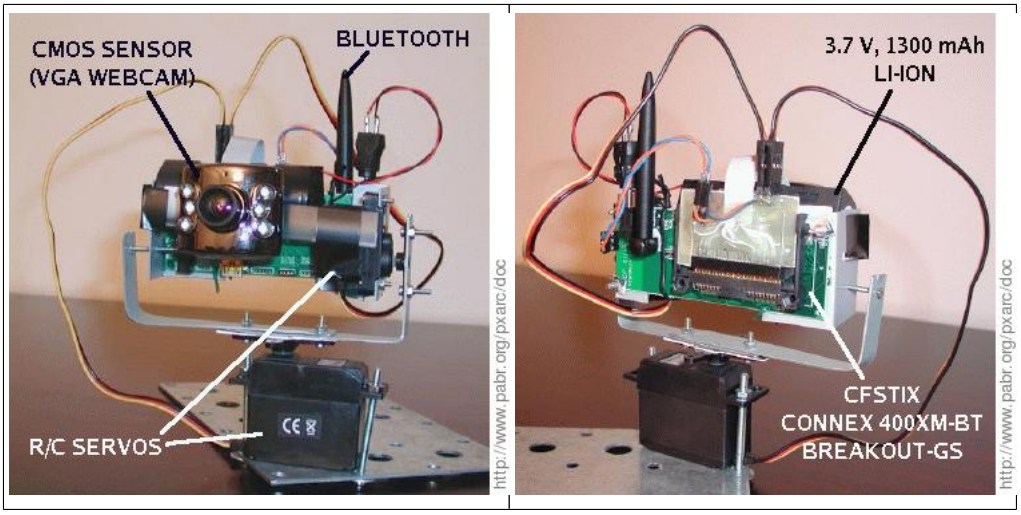
pxaRC - Logiciels pour le modélisme et la robotique sur Linux/PXA255/PXA270

pxaRC est un ensemble de drivers et d'outils pour applications de radio-commande et de robotique sur plate-formes de type Linux/PXA255/PXA270 telles que les cartes Gumstix basix, connex [http://docwiki.gumstix.org/Basix_and_connex] et verdex [http://docwiki.gumstix.org/Verdex].



READ THE HYPERTEXT VERSION HERE:
http://www.pabr.org/pxarc/doc/pxarc.fr.html

Historique des versions		
1.3	2008-09-07	Exemples d'applications. Joysticks Bluetooth.
1.2	2007-05-09	Convertisseurs ASCII/binaire. Compatibilité PXA270. Verrouillage de cache adapté pour GCC-4. cfcam optionnel.
1.1	2006-12-07	Séparé du projet ChRoMicro. Résolution 16 bits. Mise à jour pour linux-2.6.17. Support Buildroot. Support udev. Ajout infrastructure rtsched. Ajout décodeurs PWM, PPM, DCM. Ajout compteur d'impulsions. Ajout caméra CF. Renommage "bluerc" en "iprc" pour inclure Wi-Fi. Documentation en Français.
1.0	2005-06-02	Première publication dans le cadre de ChRoMicro

Table des matières

1. Introduction

Le tableau ci-dessous montre comment pxaRC se positionne entre le modélisme et la robotique.

Tableau 1. Aperçu

	Modélisme R/C	pxaRC	Robotique
Informatique embarquée	Récepteur R/C peu ou pas programmable	Carte à base d'ARM 200-400 MHz	PC/104, Mini-ITX et plate-formes propriétaires diverses
Système d'exploitation	Pas de systèmes d'exploitation ouverts	Linux avec extensions temps réel simples	Linux, vxWorks, Windows, WinCE
Canal de contrôle/ commande	Radio PPM ou PCM 41/72 MHz sans retour d'information	TCP/IP sur Bluetooth ou Wi-Fi	Protocoles divers sur 433 MHz, 868 MHz, 2.4 GHz ; TCP/IP
IHM de commande	Émetteur R/C 41/72 MHz avec manches analogiques	Joystick USB et **iprc_tx** sur PC portable ; joystick Bluetooth.	Console avec IHM dédiée
Réception	Récepteur R/C 41/72 MHz	**iprc_rx** et **pxa_opwm**	Systèmes propriétaires divers
Actuateurs	Servos et régulateurs de vitesse R/C	Servos et régulateurs de vitesse R/C ; servos I2C	Servos haut de gamme avec retour de position et d'effort ; plate-formes propriétaires diverses
Traitements vidéo	Caméra vidéo sans fil dans la bande 2.4 GHz	**pxa_cfcam** et capteur CMOS	Caméras USB ; caméras et cartes d'acquisition PC104 ou PCI
Entrées/ sorties	Sorties vers servos R/C ; peu d'entrées	GPIO, I2C, CF ; Audiostix, Robostix	Port parallèle/série/USB ; cartes d'interfaçage diverses
Capteurs	Gyros	Accéléromètres intégrés avec sorties I2C ou DCM	Divers

2. Avertissement

- Vous utilisez ces instructions et les logiciels associés à vos risques et périls.
- Il s'agit d'un projet expérimental qui n'offre aucune des fonctions de sécurité que l'on peut attendre d'un produit commercial.
- Les appareils Bluetooth classe 2 (les plus courants) ont une portée de 10 m, ce qui est insuffisant pour certaines applications.
- Contrairement aux émetteurs/récepteurs 41/72 MHz conçus pour le modélisme, les appareils Bluetooth et Wi-Fi utilisent des fréquences non réservées qui peuvent être brouillées par des sources diverses.

3. Signaux RC

Ce paragraphe a pour but de préciser la terminologie des modulations utilisées en modélisme.

3.1. Signaux de commande de servos RC

Figure 1. Signal de servo RC

La position angulaire cible du servo est encodée par la durée des états "hauts" d'un signal en créneaux, entre 1 ms et 2 ms. La durée des états "bas" n'a pas de signification particulière. Il s'agit d'une modulation de largeur d'impulsion (PWM).

3.2. Signaux multiplexés sur voie radio

Figure 2. Signal "PPM" à six canaux

Les valeurs des canaux sont codées par la durée absolue entre deux transitions montantes successives, entre 1 ms et 2 ms. Les valeurs des différents canaux sont émises successivement. Un état "bas" excédant environ 5 ms indique le début d'une nouvelle trame. Techniquement il s'agit d'une modulation PPM différentielle multiplexée. Comme la durée des états "hauts" est généralement fixée à 0,5 ms, on peut également considérer un tel signal comme une modulation PWM multiplexée entre 0,5 et 1,5 ms.

3.3. Signal de commande de pont en H

Figure 3. Signal de commande de pont en H

Un pont en H est piloté par le rapport cyclique variable d'un signal en créneaux. Il s'agit donc d'une modulation de rapport cyclique (DCM). On l'appelle parfois "PWM", mais ceci n'est justifié que lorsque la fréquence des impulsions est constante (ce qui n'est généralement pas le cas pour un signal de commande de servo RC).

3.4. Signal en sortie d'un accéléromètre

Figure 4. Signal en sortie d'un accéléromètre

Certains accéléromètres intégrés produisent un signal en créneaux modulé en rapport cyclique. Notons qu'il serait incorrect de décoder un tel signal comme une modulation PWM, puisque la fréquence n'est pas régulée.

4. Logiciels

4.1. Pré-requis

Testé avec les versions de Buildroot Gumstix svn 1161 (linux-2.6.18gum) et 1321 (linux-2.6.18gum).

4.2. Principes de conception

- Pour des raisons de performance, les données sont codées de préférence en binaire. Des utilitaires permettent de les manipuler au format ASCII : voir Section 4.10, « **short2ascii, ascii2short, long2ascii, ascii2long** : Convertisseurs binaire vers ASCII ».

- Les interfaces des *devices* utilisent de préférence **read()** et **write()**, car ces appels système sont plus faciles à utiliser que **ioctl()** dans la plupart des langages de programmation.
- Les *devices* sont configurés à l'aide de paramètres de modules plutôt qu'avec une API dédiée. Ce choix tient compte du fait que les configurations sont généralement liées au hardware et n'ont donc pas besoin d'être changées en cours d'exécution.
- Le contrôle d'accès repose sur les permissions des *devices*.

4.3. Installation

Le code source peut être téléchargé à l'adresse http://www.pabr.org/pxarc/dist/.

4.3.1. Compilation dans l'arborescence Buildroot

```
$ tar zxvf pxarc-$VERSION.tar.gz
$ cp -r pxarc-$VERSION/package/buildroot $SOMEWHERE/gumstix-buildroot/package/pxarc
$ cd $SOMEWHERE/gumstix-buildroot/
$ echo 'source "package/pxarc/Config.in"' >> package/Config.in    # Optionnel (pour voi
$ make                                                            # Compile les outils
$ make TARGETS="linux pxarc"                                      # Compile pxaRC.
$ ls build_arm_nofpu/root/lib/modules/*/misc/                     # Vérifier la présenc
```

Vous devrez ensuite installer l'image flash puis configurer les modules à l'aide de `/etc/modprobe.conf` et/ou `/etc/modules`. Si **udev** est installé, les *devices* apparaîtront dans `/dev` à chaque boot. Sinon, vous devrez configurer **dev_major=...** pour chaque module et créer les *devices* avec **mknod**.

4.3.2. Compilation hors de Buildroot

```
$ tar zxvf pxarc-$VERSION.tar.gz
$ cd pxarc-$VERSION/src
$ vi Makefile    # Modifier $(BR) et $(LINUX_VERSION)
$ make
$ scp *.ko root@$IPADDRESS:
$ scp iprc_rx.target root@$IPADDRESS:iprc_rx
$ scp ppmrelay.target root@$IPADDRESS:ppmrelay
$ scp short2ascii.target root@$IPADDRESS:short2ascii
$ scp ascii2short.target root@$IPADDRESS:ascii2short
$ scp long2ascii.target root@$IPADDRESS:long2ascii
$ scp ascii2long.target root@$IPADDRESS:ascii2long
```

Vous devrez ensuite insérer les modules manuellement avec **insmod**.

4.4. pxa_rtsched : ordonnanceur temps réel simple

4.4.1. Aperçu

`pxa_rtsched.ko` gère des tâches temps réel à l'aide des *fast interrupts* (FIQ) du PXA. Ses fonctionnalités principales sont :

- ordonnancement de tâches et de séquences de tâches, avec une gigue moyenne de l'ordre de la microseconde ;
- traitement d'interruptions GPIO sur transitions montantes et descendantes, avec une latence moyenne de l'ordre de la microseconde.

pxa_rtsched est conçu pour être utilisé par d'autres modules du noyau tels que **pxa_opwm** et **pxa_ipwm** via l'API `pxa_rtsched.h`. Il n'a aucune interface accessible par l'utilisateur, hormis l'affichage de statistiques dans **dmesg** lors du déchargement.

Le timer a une période de 271 ns pour le PXA255 et 308 ns pour le PXA370, ce qui correspond à une résolution d'environ 12 bits dans le contexte de signaux R/C. En pratique, pour une raison encore non identifiée, la gigue et la latence peuvent atteindre ±3 µs sur un processeur 200 MHz très chargé. Ceci reste acceptable pour de nombreuses applications (la plupart des servos R/C ont un hystérésis de plusieurs microsecondes).

Figure 5. Impulsions de 20 µs générées par pxa_rtsched, avec une gigue de ±2 µs

4.4.2. Détails de l'implémentation

pxa_rtsched utilise le registre OSMR1 du timer 1, qui est en principe disponible sur Linux ARM. OSMR0 est utilisé par le timer système, et OSMR3 par le *watchdog*.

Les astuces suivantes permettent de réduire la latence :

- utilisation des interruptions FIQ, qui sont prioritaires par rapport aux IRQs ordinaires utilisées par le noyau Linux ;
- utilisation des registres r8-r14 dédiés au mode FIQ ;
- verrouillage du *handler* FIQ dans le cache d'instructions ;
- verrouillage d'une partie du code du *handler* FIQ dans le cache de données également (utile car GCC stocke des constantes dans la section *.text* afin de les adresser relativement au PC) ;
- verrouillage de toutes les structures de données utilisées en mode FIQ dans le cache, y compris la pile ;
- verrouillage des adresses des périphériques internes dans le D-TLB ;
- utilisation des *Performance Monitor Count Registers* (PMN0, PMN1) du PXA pour vérifier que le code et les données sont réellement verrouillés dans les caches (plusieurs bugs ont été découverts de cette façon).

4.4.3. Problèmes identifiés

- Le temps d'exécution de l'algorithme d'ordonnancement n'est pas borné. Un comportement temps réel "dur" n'est pas garanti dans des scénarios excessivement complexes (par exemple : ajout d'une longue séquence de tâches alors que de nombreuses tâches sont déjà programmées). Un résumé des violations de contraintes temps réel peut être consulté avec **dmesg** lors du déchargement de `pxa_rtsched.ko`.
- Les interruptions du timer et celles des transitions GPIO se partagent la même ressource (les cycles CPU en mode FIQ). Si ces deux fonctionnalités sont activées simultanément, il est impossible d'assurer un comportement temps réel "dur". **Des transitions trop rapprochées sur une entrée GPIO peuvent même provoquer une paralysie complète du système.** Voir Section 6.2, « Anti-rebond logiciel sur les entrées GPIO ».

- En raison de subtilités dans la gestion de la mémoire par le noyau et de détails d'implémentation de **pxa_rtsched**, lorsqu'un module temps réel est déchargé, la précision de l'ordonnanceur est dégradée et le système devient en théorie vulnérable à une *race condition* qui peut aboutir à un crash. **Pour éviter ce risque, n'utilisez rmmod que pendant le développement, et déchargez tous les modules à la fois (y compris `pxa_rtsched.ko`).**

- Les tâches temps réel écrites en C doivent être conçues pour ne pas faire déborder la pile FIQ.

4.5. pxa_opwm : Générateur de signaux PWM/PPM

4.5.1. Aperçu

`pxa_opwm.ko` est un générateur de signaux PWM/PPM pour processeurs PXA255 et PXA270. Il se présente comme un module noyau Linux chargeable dynamiquement. Il peut être configuré pour générer un signal PPM multi-canal unique et/ou les signaux PWM correspondants. Les signaux peuvent être émis sur n'importe quelque broche GPIO.

pxa_opwm n'utilise pas les générateurs PWM hardware du PXA.

pxa_opwm remplace **pxa_mpwm**, version antérieure avec une résolution de 8 bits.

4.5.2. Utilisation

Figure 6. Signaux générés en mode PPM ; explication des réglages de timing

Les broches GPIO, canaux et temporisations sont configurés à l'aide des paramètres du module, comme illustré ci-dessous (Tableau 2, « Exemples de configurations pour le module **pxa_opwm** »).

Un processus utilisateur contrôle **pxa_opwm** en écrivant des entiers non signés sur 16 bits en orientation native (*host endianness*) dans un *device* (voir Tableau 3, « *Devices* **pxa_opwm** »). Les signaux sont générés dès que le programme a ouvert le *device* et initialisé les valeurs de tous les canaux.

Lorsque le programme libère le *device*, les sorties repassent au niveau logique 0. Ceci provoque généralement l'arrêt des variateurs de vitesse et la mise en veille des servos. Cette fonction de sécurité peut être désactivée en ajoutant **persistent=1** dans le paramétrage du module.

Tableau 2. Exemples de configurations pour le module pxa_opwm

Application	Paramétrage du module
Un signal PPM RC contenant 6 canaux	**modprobe pxa_opwm gpio=61 nchans=6 tmin=1000 tmax=2000 tpause=500 tsync=12000**
Idem avec les signaux PWM pour servos correspondants, sauf CH5	**modprobe pxa_opwm gpio=61 nchans=6 servo=58,59,60,62,-1,63 tmin=1000 tmax=2000 tpause=500 tsync=12000**
Sorties PWM pour servos uniquement	**modprobe pxa_opwm nchans=6 servo=58,59,60,62,-1,63 tmin=1000 tmax=2000 tpause=500 tsync=12000**

Tableau 3. *Devices* pxa_opwm

Device	Mineur	Affectation
/dev/opwm0-0	0	Canal n°1 du signal PPM n°1
/dev/opwm0-1	1	Canal n°2 du signal PPM n°1
/dev/opwm0-2	2	Canal n°3 du signal PPM n°1
...	...	...
/dev/opwm0	15	Signal PPM n°1 (canaux configurés simultanément)
/dev/opwm1-0	16	Canal n°1 du signal PPM n°2
...	...	...
/dev/opwm1	31	Signal PPM n°2 (canaux configurés simultanément)

4.5.3. Détails de l'implémentation

pxa_opwm est une application directe de **pxa_rtsched**. Pour générer une trame PPM de N impulsions, **pxa_opwm** enregistre une séquence temporisée de 2*N tâches dans l'ordonnanceur (N pour les transitions montantes, N pour les transitions descendantes). Les signaux PWM optionnels correspondants sont générés par les même tâches.

4.5.4. Problèmes identifiés

- Les sorties GPIO ont une amplitude de 3,3 V (VOL=0,4; VOH=3,2 - voir [PXA255_ELEC], [PXA270_ELEC]). La plupart des servos RC se contentent de signaux PWM d'amplitude 2 V, mais certains à base de circuits CMOS pourraient de ne pas reconnaître un "1" logique à 3,2 V.
- C'est généralement dans le but de réduire les coûts que l'on choisit de générer des signaux PWM par logiciel. Pour des applications nécessitant des signaux très précis, un générateur PWM hardware serait plus approprié.
- **pxa_opwm** autorise **pxa_rtsched** à prolonger les impulsions de synchronisation au delà de `tsync` si cela permet de résoudre des conflits d'ordonnancement. Ceci se produira notamment si l'on génère plusieurs signaux PPM simultanément (mais pas lorsqu'on génère plusieurs signaux PWM à partir d'un seul signal PPM "virtuel" en configurant exclusivement des broches **servo=...**).
- Le paragraphe ci-dessus implique également que **pxa_opwm** ne doit pas être utilisé pour générer des signaux modulés en rapport cyclique ou en fréquence, sauf dans des cas très simples où l'ordonnancement est entièrement déterministe (par exemple : signal unique et aucune autre tâche temps réel).
- Les avertissements relatifs à **pxa_rtsched** s'appliquent également à **pxa_opwm**.

Il existe au moins deux autres solutions pour générer des signaux PWM sur une plate-forme Gumstix:

- utiliser le générateur PWM hardware du PXA (deux à quatre signaux PWM 3.3 V indépendants ; PPM possible avec un driver approprié) ;
- ajouter une carte périphérique (voir [ROBOSTIX_SIMPLE_SERVO]).

4.6. pxa_ipwm : Compteur d'impulsions et décodeur de signaux PWM/PPM/DCM

4.6.1. Aperçu

`pxa_ipwm.ko` est un décodeur de signaux PWM/PPM/DCM et compteur d'impulsions à base d'interruptions GPIO. Il se présente comme un module noyau Linux chargeable dynamiquement.

4.6.2. Utilisation

Les broches GPIO, formats des signaux et temporisations sont configurés à l'aide des paramètres du module comme illustré ci-dessous (Tableau 4, « Exemples de configurations pour le module **pxa_ipwm** »).

Figure 7. Décodage PWM ; explication des paramètres

Figure 8. Décodage PPM ; explication des paramètres

Figure 9. Décodage DCM ; explication des paramètres

Tableau 4. Exemples de configurations pour le module pxa_ipwm

Application	**Paramétrage du module**
Signal PWM de servo	**modprobe pxa_ipwm mode=0 gpio=61 tmin=1000 tmax=2000 tsync=4000 timeout=100000**
Un signal PPM RC contenant 6 canaux multiplexés	**modprobe pxa_ipwm mode=1 gpio=61 nchans=6 tmin=1000 tmax=2000 tsync=4000 timeout=100000**
Signal DCM à 2 kHz ±10%	**modprobe pxa_ipwm mode=2 gpio=61 tmin=455 tmax=556 timeout=100000**
Comptage d'impulsions, lecture retardée de 3 ms	**modprobe pxa_ipwm mode=3 gpio=61 timeout=3000**
Deux compteur d'impulsions, lecture retardée de 3 ms	**modprobe pxa_ipwm mode=3,3 gpio=61,62 timeout=3000,3000**

Un processus utilisateur interroge **pxa_ipwm** par l'intermédiaire de *devices* (voir Tableau 5, « *Devices* **pxa_ipwm** »).

En mode PWM/PPM/DCM, **read()** renvoie l'erreur -ETIMEDOUT si aucune mesure datant de moins de `timeout` n'est disponible. Si le signal ne respecte pas les temporisations configurées, **read()** renvoie -EIO (cf. figures). Sinon il renvoie la mesure la plus récente (sans attendre l'impulsion suivante). Le format des valeurs de canaux est le même que pour **pxa_opwm**.

En mode compteur, **read()** renvoie un entier non signé de 32 bits en orientation native (*host endianness*). Le premier appel à **read()** renvoie immédiatement la valeur courante du compteur. Chaque appel suivant attend que le compteur s'incrémente. Il peut y avoir une latence (configurable par **timeout=...**) ; ceci est dû au fait qu'il est difficile de faire réveiller un process utilisateur par une tâche FIQ.

Un compteur d'impulsions peut être réinitialisé à une valeur arbitraire en écrivant dans le *device*. Cependant, il n'est pas prévu de pouvoir lire et effacer un compteur de façon atomique.

Tableau 5. *Devices* pxa_ipwm

Device	Mineur	Affectation
/dev/ipwm0-0	0	Canal n°1 du signal n°1 (mode PPM uniquement)
/dev/ipwm0-1	1	Canal n°2 du signal n°1 (mode PPM uniquement)
/dev/ipwm0-2	2	Canal n°3 du signal n°1 (mode PPM uniquement)
...	...	...
/dev/ipwm0	15	Signal n°1, canaux lus simultanément (modes PPM/PWM/DCM uniquement) ; ou compteur n°1
/dev/ipwm1-0	16	Canal n°1 du signal n°2 (mode PPM uniquement)
...	...	...
/dev/ipwm1	31	Signal n°2, canaux lus simultanément (modes PPM/PWM/DCM uniquement) ; ou compteur n°2

4.6.3. Détails de l'implémentation

pxa_ipwm s'enregistre auprès de **pxa_rtsched** pour recevoir les interruptions sur transitions GPIO. Le *handler* FIQ enregistre un horodatage de chaque transition dans une file FIFO (implémentée par un buffer cyclique), et ces données sont converties en valeurs PWM/PPM/DCM lorsque **read()** est appelé. Le *handler* FIQ incrémente aussi les compteurs d'impulsions.

4.6.4. Problèmes identifiés

- Les entrées GPIO du PXA sont prévues pour des niveaux logiques 3,3 V. Ne connectez pas un signal 5 V directement.
- Les broches GPIO0 et GPIO1 ne peuvent pas être utilisées car des interruptions spécifiques leur sont affectées.
- Toutes les transitions sont horodatées dans une même file. En conséquence, si l'on décode plusieurs signaux dont l'un a une fréquence beaucoup plus élevée que les autres, **pxa_ipwm** ne pourra pas décoder les signaux lents. Dans certains cas, on peut remédier à ce problème en augmentant `MAX_EV` lors de la compilation. Voir aussi Section 6.3, « Files multiples dans **pxa_ipwm** ».
- **pxa_ipwm** prend le contrôle de toutes les interruptions sur transitions GPIO (sauf sur GPIO0 et GPIO1). Sur les cartes Gumstix ceci perturbe notamment le driver ethernet. Le driver USB (**pxa2xx_udc**) devrait être affecté également car il détecte les IRQs de la broche USB_GPIOn; en pratique, les connexions USB pré-existantes continuent à fonctionner.
- Pour des raisons de performance, toutes les broches GPIO utilisées par **pxa_ipwm** doivent être dans le même groupe de 32 bits.
- La précision des mesures décroît lorsque le nombre de broches à observer et la probabilité de transitions "simultanées" augmentent.
- En mode compteur, les impulsions doivent durer plus de 1 µs (d'après [PXA255_DEVEL]). La fréquence de comptage maximale dépend de la latence des interruptions et de la durée de la tâche temps réel la plus longue. Si une impulsion survient alors que le processeur n'a pas encore acquitté l'interruption d'une impulsion précédente, elle ne sera pas comptée. En l'absence d'autres tâches temps réel, **pxa_ipwm** peut compter des impulsions à 100 kHz.

- Les avertissements relatifs à **pxa_rtsched** s'appliquent également à **pxa_ipwm**.

Il existe des alternatives à **pxa_ipwm** pour la plate-forme Gumstix ; voir par exemple [ROBOSTIX_RC_INPUT].

4.7. iprc_rx : Émulateur de récepteur RC

```
iprc_rx [-p <port>] [-t <timeout>] [-c <OPWM device>]
```

Exemple 1. iprc_rx

```
iprc_rx -p 9001 -t 2000 -c /dev/opwm0
```

iprc_rx est un programme exécutable sur la carte embarquée. Il reçoit des ordres de radio-commande sous forme de paquets UDP sur une connexion Bluetooth BNEP ou Wi-Fi, et les transmet à **pxa_opwm**.

Le programme s'arrête lorsqu'il n'a reçu aucun paquet UDP pendant un intervalle de temps configurable.

4.8. iprc_tx : Émulateur d'émetteur RC

```
iprc_tx [-c <joystick>] [-r <refresh rate>] [-m <mixer file>] [-d <dest IP>] [-p <dest
```

Exemple 2. iprc_tx

```
iprc_tx -c /dev/js0 -r 50 -m xpad_linear6.mix -d 192.168.10.1 -p 9001
```

iprc_tx est un programme exécutable sur une station Linux équipée d'une interface Bluetooth ou Wi-Fi et d'un joystick USB. Il lit les positions des boutons et des manettes analogiques, les mélange linéairement pour produire 6 canaux RC, et transmet les valeurs des canaux dans des paquets UDP sur une connexion IPv4 sans fil.

La correspondance et le mixage entre les axes du joystick et les canaux RC sont spécifiés par une matrice dans un fichier de configuration. Voir par exemple `pxarc/examples/xpad_linear6.mix`.

Procédure 1. Trimming

1. amener les manettes à la position souhaitée pour le "neutre" ;
2. presser et maintenir enfoncé le bouton "back" ;
3. laisser les manettes revenir à leur position centrale ;
4. relâcher le bouton "back".

4.9. ppmrelay : Copie un flux PPM

ppmrelay est un programme exécutable sur la carte embarquée. Il reçoit des commandes PPM décodées par **pxa_ipwm**, affiche les valeurs des canaux sur la sortie standard, et les transmet à **pxa_opwm**.

Il peut servir de canevas pour des traitements plus complexes tels que :

- détecter et pallier les pertes de signal radio ;
- interpoler les commandes à une fréquence supérieure (la plupart des servos tolèrent un rafraîchissement plus rapide que les 50 Hz habituels) ;
- exécuter des séquences de commandes pré-enregistrées ;

- injecter des données de capteurs embarqués dans la chaîne de commande.

4.10. short2ascii, ascii2short, long2ascii, ascii2long : Convertisseurs binaire vers ASCII

Ces utilitaires peuvent être utilisés pour manipuler des *devices* **pxa_opwm** et **pxa_ipwm** en format ASCII (plutôt que binaire), par exemple à partir d'une console ou d'un script shell.

Avertissement

ascii2short et **ascii2long** interprètent en octal les nombres commençant par 0.

Exemple 3. Lire les valeurs PWM

```
short2ascii  < /dev/ipwm0
```

Exemple 4. Lire les valeurs PWM toutes les 20 ms environ

```
short2ascii -d 20000  < /dev/ipwm0
```

Exemple 5. Lire un compteur d'impulsions

```
long2ascii -d 0  < /dev/ipwm0
```

Exemple 6. Écrire des valeurs PPM

```
echo "32768 32768"  |  ascii2short  > /dev/opwm0
```

Exemple 7. Écrire une séquence de valeurs PPM

```
(echo "16384 49152";
 sleep 1;
 echo "49152 16384";
 sleep 1;
 echo "32768 32768"
)  |  ascii2short  > /dev/opwm0
```

Exemple 8. Manipuler les canaux d'un signal PPM

```
short2ascii -d 10000  < /dev/ipwm0 \
| while read v1 v2 v3 v4 v5 v6; do
    echo "Échange de $v1 et $v2, inversion de $v3"  1>&2
    echo "$v2 $v1 $((65535-v3)) $v4 $v5 $v6"
  done \
| ascii2short  > /dev/opwm0
```

4.11. pxa_cfcam : capteur vidéo CMOS sur port PCM-CIA/CF

4.11.1. Aperçu

`pxa_cfcam.ko` capture des images en provenance d'un capteur CMOS connecté directement sur le port PCMCIA/CF du PXA255. L'interfaçage est décrit dans [CFCAM].

Fonctionnalités :

- capture d'images fixes et de flux vidéo ;
- interface pour *plugins* permettant de traiter les données brutes à la volée ;
- émulation I2C sur GPIO (utile car les signaux I2C du PXA255 ne sont pas accessibles sur le connecteur CF).

4.11.2. pxa_cfcam_pnm : conversion au format PNM

`pxa_cfcam_pnm.ko` est un *plugin* pour **pxa_cfcam** qui transforme les données du capteur CMOS au format Bayer en images non compressées au format PNM (couleur 24 bits ou niveaux de gris 8 bits). Le header PNM peut être supprimé avec l'option **pnm_header=0**.

Exemple 9. Capture d'images fixes

```
# modprobe pxa_cfcam
# modprobe pxa_cfcam_pnm
# dd if=/dev/cfcam0 of=/tmp/image.ppm bs=2M count=1
```

Exemple 10. Capture d'images fixes (niveaux de gris)

```
# modprobe pxa_cfcam
# modprobe pxa_cfcam_pnm bpp=1
# dd if=/dev/cfcam0 of=/tmp/image.pgm bs=2M count=1
```

Exemple 11. Capture vidéo asynchrone

```
# modprobe pxa_cfcam
# modprobe pxa_cfcam_pnm pnm_header=0

/* Exemple simplifié sans traitement des erreurs. */
int fd = open("/dev/cfcam0", O_RDWR, 0);
unsigned char *mem = mmap(NULL, 320*240*3*2, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
while ( 1 ) {
  u32 offset;
  read(fd, &offset, sizeof(offset));
  unsigned char *rgb = mem + offset;
  /* 'rgb' pointe vers une image RVB de 320*240*3 octets */
}
```

Ce mode permet de capturer des images consécutives. Lorsqu'un buffer est plein, **read()** renvoie sa position par rapport au début de la zone *mmap*ée, et le driver commence à capturer l'image suivante dans un autre buffer. Le programme doit finir son traitement et appeler à nouveau **read()** avant que le second buffer ne soit plein à son tour.

4.11.3. Capteurs compatibles

L'implémentation est encore très rudimentaire. La liste des configurations supportées est figée dans `pxa_cfcam.c`. La configuration à utiliser est sélectionnée à l'aide du paramètre de module **sensor=N**. Pour gérer un nouveau capteur ou un nouveau format d'image, il faut modifier le code source.

Pour déterminer les caractéristiques d'un nouveau capteur, on peut commencer par enregistrer une image brute de la façon suivante :

Exemple 12. Enregistrement des données brutes d'un capteur inconnu

```
# modprobe pxa_cfcam sensor=0
# modprobe pxa_cfcam_pnm raw=1 bpp=1
# dd if=/dev/cfcam0 of=/tmp/image.pgm bs=2M count=1
```

4.11.4. Détails de l'implémentation

pxa_cfcam commence par actionner l'entrée *reset* du capteur, et envoie optionnellement des commandes I2C pour le configurer. Il émet également le nombre d'impulsions d'horloge requis pour que le capteur achève son initialisation et commence à produire des images.

Ensuite, une liste circulaire de descripteurs DMA lit en permanence les données brutes du capteur CMOS, et le cadence par la même occasion. Les données reçues pendant les intervalles de synchronisation horizontale et verticale sont ignorées.

Des interruptions DMA sont générées à la fin de chaque ligne. En mode synchrone (sans utiliser **mmap()**), **pxa_cfcam** transfère et convertit les données depuis le buffer DMA vers le buffer référencé dans l'appel système **read()**. En mode asynchrone (avec **mmap()**), **pxa_cfcam** transfère et convertit les données depuis le buffer DMA vers la moitié d'un buffer *mmap*é au préalable, pendant que le process utilisateur travaille sur l'autre moitié. Les deux moitiés sont échangées lors de chaque appel système **read()**.

pxa_cfcam n'utilise pas les sorties HSYNC and VSYNC du capteur. Ceci nécessite que les nombres de cycles d'horloge de chaque image et de chaque intervalle de synchronisation soient connus à l'avance.

4.11.5. Problèmes identifiés

- **pxa_cfcam** utilise certains signaux du connecteur CF d'une façon qui n'est pas compatible avec la norme CF. Ne l'installez pas alors qu'une carte CF est insérée.
- Les sorties d'un capteur CMOS ne sont pas trois-états et seraient donc en conflit avec celles de tout autre périphérique connecté au bus d'E/S. **N'utilisez pas pxa_cfcam avec un netcf ou un wifistix-cf.** Ceci nécessiterait un *driver* capable de couper l'alimentation du capteur pendant les accès au périphérique, et réciproquement.
- **read()** comporte une boucle d'attente active, ce qui peut ralentir considérablement les autres processes. Pour des applications temps réel, il est préférable d'utiliser l'API asynchrone (**mmap()**).
- Les versions de `rootfs` antérieures à 1161 peuvent booter Linux avec GPIO54 configuré en mode "AF2 in" (à vérifier par **cat /proc/gpio/GPIO54**). Ceci peut provoquer des conflit sur le bus de données du PXA255, d'où des *segmentation faults* aléatoires. Pour réduire ce risque, il suffit d'ajouter **"pinit on"** au début de la chaîne **bootcmd** de **u-boot**.

5. Exemples d'applications

5.1. Télécommande avec joystick USB connecté à une station fixe

Figure 10. Télécommande avec joystick USB connecté à une station fixe

Dans ce scénario (Figure 10, « Télécommande avec joystick USB connecté à une station fixe »), **iprc_tx** tournant sur un PC fixe transmet les commandes d'un joystick USB à **iprc_rx** par Bluetooth (ou Wi-Fi).

5.2. Télécommande avec joystick Bluetooth

Figure 11. Radio-commande avec un joystick Bluetooth

Ce scénario (Figure 11, « Radio-commande avec un joystick Bluetooth ») repose sur le fait que les joysticks sans fil modernes sont compatibles avec le profil Bluetooth HID, ce qui permet de les connecter directement à une carte Linux embarquée équipée d'un module Bluetooth. C'est le cas notamment du contrôleur "SIXAXIS" de la console PS3, comme expliqué dans [SIXLINUX].

6. Développements futurs

6.1. Qualité de service sur Bluetooth

Objectif : Réduire la latence (actuellement de l'ordre de 20 ms).

Bluetooth offre un service appelé SCO (*Synchronous Connection-Oriented*). Ceci pourrait permettre d'obtenir une latence plus faible qu'avec BNEP.

6.2. Anti-rebond logiciel sur les entrées GPIO

Objectif : Éviter que des transitions trop rapprochées sur une entrée GPIO puissent surcharger le système.

Une solution possible consiste à masquer les interruptions pendant un délai configurable après chaque événement.

6.3. Files multiples dans pxa_ipwm

Objectif : Éviter qu'un signal haute fréquence puisse perturber le décodage de signaux plus lents.

Une solution possible consiste à gérer une file par broche. Cependant, le *handler* FIQ serait alors plus lent, ce qui dégraderait les performances temps réel.

6.4. Compatibilité Video4Linux

pxa_cfcam implémente des fonctions équivalentes à celles de V4L (avec une API plus légère). Il pourrait être utile de le transformer en un *device* compatible V4L.

Bibliographie

[CFCAM] *CFcam - Interfaçage d'une caméra CMOS avec une carte Gumstix Connex* . http://www.pabr.org/cfcam/doc/cfcam.fr.html .

[PXA255_DEVEL] *Intel PXA255 Processor*. Developer's manual. 27869302.pdf.

[PXA255_USER] *Intel XScale Microarchitecture for the PXA255 Processor*. User's Manual. 27879601.pdf.

[PXA255_ELEC] *Intel PXA255 Processor*. Electrical, Mechanical, and Thermal Specification. 27878002.pdf.

[PXA27X_DEVEL] *Intel PXA27x Processor Family*. Developer's manual. 2800002.pdf.

[PXA270_ELEC] *Intel PXA270 Processor*. Electrical, Mechanical, and Thermal Specification. 28000205.pdf.

[SIXLINUX] *Utilisation du joystick de la PlayStation 3 en mode Bluetooth avec Linux* . http://www.pabr.org/sixlinux/sixlinux.fr.html .

[ROBOSTIX_SIMPLE_SERVO] *Robostix simple servo*. Dave Hylands. http://docwiki.gumstix.org/Robostix_simple_servo.

[ROBOSTIX_RC_INPUT] *Robostix RC input*. Dave Hylands. http://docwiki.gumstix.org/Robostix_RC_input.

Glossaire

Bluetooth Network Encapsulation Protocol	Fournit une interface semblable à Ethernet (typiquement **bnep0**) à chaque extrémité d'une connexion Bluetooth.
CompactFlash	Une variante de PCMCIA.
Duty Cycle Modulation	Modulation de rapport cyclique. Une modulation qui code un signal analogique par la valeur moyenne d'un signal en créneaux. Utilisée couramment pour varier la vitesse d'un moteur électrique avec un pont en H. Voir aussi Pulse Width Modulation.
Electronic Speed Controller	Un variateur de vitesse pour moteur, typiquement asservi par un signal PWM dans un modèle réduit.
General-Purpose I/O	Le PXA255 comporte 85 broches qui peuvent être configurées chacune comme entrée ou comme sortie, ou connectées aux périphériques intégrés (par exemple les ports série et le contrôleur LCD). Le PXA270 en a 119.
Pulse Code Modulation	Désigne une transmission numérique, par opposition à PWM ou PPM. Voir aussi Pulse Width Modulation, Pulse Position Modulation.
Pulse Position Modulation	Désigne une modulation PPM différentielle à plusieurs canaux, dans la terminologie du modélisme. Utilisé couramment sur la voie radio dans les systèmes RC commerciaux. Voir aussi Pulse Width Modulation, Pulse Code Modulation.
Pulse Width Modulation	Une modulation qui code un signal analogique par des impulsions binaires de durée variable. Utilisée couramment pour contrôler des servos RC. Voir aussi Pulse Position Modulation, Duty Cycle Modulation.