Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Ingegneria Informatica

Driver di Radio FM per STLinux 2.6.20

Progetto del corso

"Laboratory of Operating Systems and Software Design"

Professore: William Fornaciari
Tutor: Patrick Bellassi

Lucas Martins De Marchi, 719958
Cristina Hohan Yu Chang, 722326

Politecnico di Milano
Anno Accademico 2007/2008

Indice

1 Obiettivo

Lo scopo di questo progetto è fare un driver per il sistema operativo Linux per il dispositivo STLC2590. Affinché questo sia raggiunto, i seguenti topici devono essere trattati:

- Sviluppo di un driver kernel Linux 2.6.x che consenta di pilotare il chip usando la API di STLinux[1] per l'interfacciamento con dispositivi I2C;
- Sistemare dentro il driver la connessione fra il chip di radio (STLC2590) e l'audiocoded (STW5095) in modo da ottenere la riproduzione del flusso audio. Questa connessione sarà fatta così soltanto per non dipendere dall'Alsa, giacché sarebbe il suo ruolo sistemarla;
- Interfaccia di controllo per lo spazio utente mediante l'integrazione con Video4Linux[2] (in modo tale che si possa controllarlo con applicativi user-space come fmio, fmtools, gnomeradio, kradio ecc).

2 Concetti base

Qui vengono presentati i concetti che bisogna sapere per capire questo documento. I posti dove trovare più informazioni vengono consigliate in ogni topico e tramite richiami alla bibliografia.

2.1 RDS

Il Radio Data System[3] è un protocollo di comunicazione standard dell' European Broadcasting Union. È un metodo per inviare piccole quantità di informazioni digitali insieme alla tradizionale trasmissione radio FM. Il sistema RDS permette di inviare diversi tipi di informazioni tra cui le più importante sono:

- Identificativo della stazione radio;
- Lista delle frequenze alternative per la stessa stazione;
- Informazione sul cantante e la traccia;
- Data e ora attuale;
- Tipo del programma (ad esempio: Rock, Pop, Notizie ecc.);
- Traffic message channel (TMC): informazione sul traffico stradale

La Tabella 2.1 mostra i campi che possono comparire nei pacchetti di dati RDS.

Simbolo	Nome	Descrizione
AF	Alternative Frequencies	Permette che il ricevitore si sintonizzi con un'altra frequenza che fornisce la stessa stazione quando il primo segnale diventa troppo debole
CT	Clock Time	Può sincronizzare l'orologio del ricevitore
EON	Enhanced Other Networks	Permette il ricevitore di monitorare altri reti o stazioni per programma di traffico, e automaticamente sintonizzare temporariamente in questa stazione
PI	Programme Identification	Codice unico che identifica la stazione
PS	Programme Service	Display statico di otto caratteri che in generale ha il nome della stazione
PTY	Programme Type	Tipo del programma che la stazione sintonizzata sta trasmettendo. Permette agli utenti di trovare programmazione simile per genero
REG	Regional	Usato per identificare la regione se il programma è specifico di una regione. Così l'utente è in grado di sintonizzare solo quelle stazioni della regione in cui si trova
RT	Radio Text	Questa funzione permette a una stazione radio di trasmettere informazione testuale
TA, TP	Traffic Announcement, Traffic Programme	Il ricevitore può essere configurato per automaticamente pausare il CD/ tape e sintonizzare con un notiziario di Traffico
TMC	Traffic Message Channel	Invia informazione sul traffico e viaggi agli utenti

Tabella 2.1: Campi di informazione RDS.

2.2 Chip STLC2590

Chip di STMicroeletronics che combina entrambi le funzionalità del Bluetooth (chip STLC2500C imbarcato) e del FM Tuner (FW15) in un solo prodotto. La configurazione e utilizzo del

Bluetooth è di tutto separato di quella di radio e non si tratterà qui di queste sue funzionalità.

Il sintonizzatore Radio FM supporta la banda FM a livello mondiale, cioè: è possibile configurare il chip per lavorare in un intervallo di frequenza d'accordo con la regione del mondo dove si trova. La sezione FM incorpora un processore digitale per l'European Radio Data System (RDS) e il North American Radio Broadcast Data System (RBDS) includendo tutti i requisiti di decodifica, sincronizzazione di blocchi, rilevamento e correzione d'errore.

La Tabella 2.2 seguente illustra i pin del chip rispetto alla Radio FM.

Nome	**Pin**	**Descrizione**	**Tipo**	**Stato durante il riavvio**	**Stato dopo il riavvio**
FM_GND	A3	Terra			
	B4				
	B6				
	B7				
	C4				
	C6				
	C7				
	D5				
FM_VA	A4	Tensione di alimentazione analogica			
FM_VD	B3	Tensione di alimentazione digitale			
FM_GPIO1	A7	Input/output multiuso	I/O	Input/Output	Input/Output
FM_GPIO2	A6				
FM_GPIO3	A5				
FM_FMIP	B8	RF input			
FM_ROUT	C5	Audio input a destra			
FM_LOUT	B5	Audio input a sinistra			
FM_RFGND	C8	Terra RF			
FM_RSTB	C9	Reinizializzare l'input	I	Input low	Input high
FM_VIO	D6	Tensione di alimentazione I/O			
FM_RCLK	D7	Input dell'oscillatore di referenza esterno	I	Input	Input
FM_SDIO	D8	Input/Output seriale di dati	I/O	Input/Output	Input/Output
FM_SENB	D9	Abilitare input seriale	I	Input	Input
FM_SCLK	E9	Input seriale dell'orologio	I	Input	Input

Tabella 2.2: Lista dei pin del chip STLC2590 relativi alla funzionalità di radio FM.

La Tabella 2.3 contiene l'elenco dei registri relativi alla configurazione della sezione di radio del chip.

Name	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
DEVICEID	PN[3:0]				MFGID[11:0]											
CHIPID	REV[5:0]						DEV	FIRMWARE[8:0]								
POWERCFG	DSMUTE	DMUTE	MONO	0	RDSM	SKMODE	SEEKUP	SEEK	0	DISABLE	0	0	0	0	0	ENABLE
CHANNEL	TUNE	0	0	0	0	0	CHAN[9:0]									
SYSCONFIG1	RDSIEN	STCIEN	0	RDS	DE	AGCD	0	0	BLNDAD[1:0]		GPIO3[1:0]		GPIO2[1:0]		GPIO1[1:0]	
SYSCONFIG2	SEEKTH[7:0]								BAND[1:0]		SPACE[1:0]		VOLUME[3:0]			
SYSCONFIG3	STMUTER[1:0]		STMUTEA[1:0]						SKSNR[3:0]				SKCNT[3:0]			
TEST1	0	AHIZEN														
STATUSRSSI	RDSR	STC	SF/BL	AFCRL	RDSS	BLERA[1:0]		ST	RSSI[7:0]							
READCCHAN	BLERB[1:0]		BLERC[1:0]		BLERD[1:0]		READCHAN[9:0]									
RDSA	RDSA[15:0]															
RDSB	RDSB[15:0]															
RDSC	RDSC[15:0]															
RDSD	RDSD[15:0]															

Tabella 2.3: Elenco dei registri

2.3 SysFS

Sysfs è un filesystem virtuale che è montato su /sysfs con cui è possibile ai drivers esportare dei variabili. L'utente può leggere i valori di queste variabili ed anche, se permesso per il driver, modificare i suoi valori tramite semplice chiamate di operazione con i file sotto il /sysfs: ogni volta che l'utente fa una lettura o scritta in un file sotto il sysfs il kernel gira questa chiamata al driver corrispondente che deve essere in grado di risponderla inviando una stringa allo spazio utente (se è stata fatta una lettura) oppure usando il valore passato dallo spazio utente, quest'ultimo caso se fosse una scritta.

Così l'interfaccia con lo spazio utente si rende molto più facile di fare e può essere anche usata per esportare delle variabili di debugging, perciò serve di grande aiuto nello sviluppo del driver. Maggior informazioni possono essere trovate in [4].

2.4 Video4Linux

È una interfaccia kernel per dispositivi di media come radio, scheda di cattura di video, camera ecc. L'API usata in questo driver è la seconda, conosciuta come V4L2 (Video4Linux 2) ed è stata progettata per supportare una grande varietà di dispositivi. Ci concentreremo in spiegare l'interfaccia di V4L con i dispositivi di radio. L'intera API si trova su [5].

L'interfaccia di V4L2 con lo spazio utente viene fatta tramite gli archivi di character device (il default per i dispositivi di radio sono i /dev/radio0, /dev/radio1, ecc). La configurazione del chip deve essere fatta con chiamate di ioctl su questi archivi. V4L2 implementa la funzione video_ioctl2 il cui ruolo è mappare le chiamate di ioctl dallo spazio utente sulle funzioni implementate nel driver, come vedremo nella Figura 2 in “Dettagli dell'implementazione“. Nella Tabella 2.4 c'è l'elenco di tutte le funzione di ioctl che devono essere implementate nei drivers di radio e una breve spiegazione di ognuna.

IOCTL	Descrizione
vidioc_querycap	Riporta i nomi del chip e del driver, il bus usato nella sua configurazione e le funzionalità di V4L2 che il driver implementa. Nel caso di radio che supporta RDS le flag abilitate sono quelle di sintonizzatore e la “read” (per l'RDS).
vidioc_queryctrl	Riporta i controllori disponibile nel driver che non sono davvero necessari al funzionamento del driver: volume e mute.
vidioc_g_ctrl vidioc_s_ctrl	Metodi “set” e “get” da chiamare per configurare i controllori riportati dalla funzione di sopra.
vidioc_g_tuner vidioc_s_tuner	Il metodo “get” riporta l'intervallo di frequenza disponibile, se la radio è capace di ricevere segnalo stereo ed anche se quella attualmente sintonizzata è stereo o meno e il livello del segnalo. Come i dispositivi radio hanno soltanto 1 sintonizzatore il metodo “set” non deve fare niente altro che non permettere che il tuner sia cambiato.
vidioc_g_frequency vidioc_s_frequency	Metodo per cambiare o ottenere la frequenza su cui il chip della radio è sintonizzato.
vidioc_g_audio vidioc_s_audio	Poiché i drivers di radio non possono avere nessuna entrata o uscita d'audio questi metodi devono soltanto non permettere che i programmi nello spazio utente configurino questi parametri.
vidioc_g_input vidioc_s_input	Poiché i drivers di radio non possono avere nessuna entrata o uscita di video questi metodi devono soltanto non permettere che i programmi nello spazio utente configurino questi parametri.

Tabella 2.4: IOCTLs dell'API V4L2

3 Cross-compiling e tools utilizzate

Il target del driver è la board NHK-15 che ha un processore ARM. Per potere sviluppare il driver sui PC con processore x86 è stata usata la toolchain di cross-compiling di Poky Linux[6]. Inoltre abbiamo usato il kernel linux 2.6.20 con dei patches di ST per la board NHK-15 e per le API del STLinux.

Tutte le configurazione di cross-compiling sono state messi in un archivio e l'abbiamo chiamato init_env. Così, per compilare il driver basta fare un “make” sotto la cartella con la sorgente.

Per le prove sono stati usati:

i) il programma “fm” di fmtools[7] per pilotare il chip cambiando frequenza sintonizzata, volume, mute ecc;

ii) il daemon rdsd per interrogare il chip e prendere dei dati RDS;

iii) telnet per collegarsi al daemon e chiedere le informazione di RDS.

4 La board NHK-15

La board NHK-15 contiene un processore Nomadik, che appartiene all'architettura dei processori ARM. Per quanto riguarda la riproduzione di radio FM le parti più importante della board sono:

- Radio FM STLC2590;
- Audiocodec STW5095;
- Port expander STMPE2401;
- Altoparlanti;

In Figura 1 sono rappresentati i collegamenti fra i vari chip.

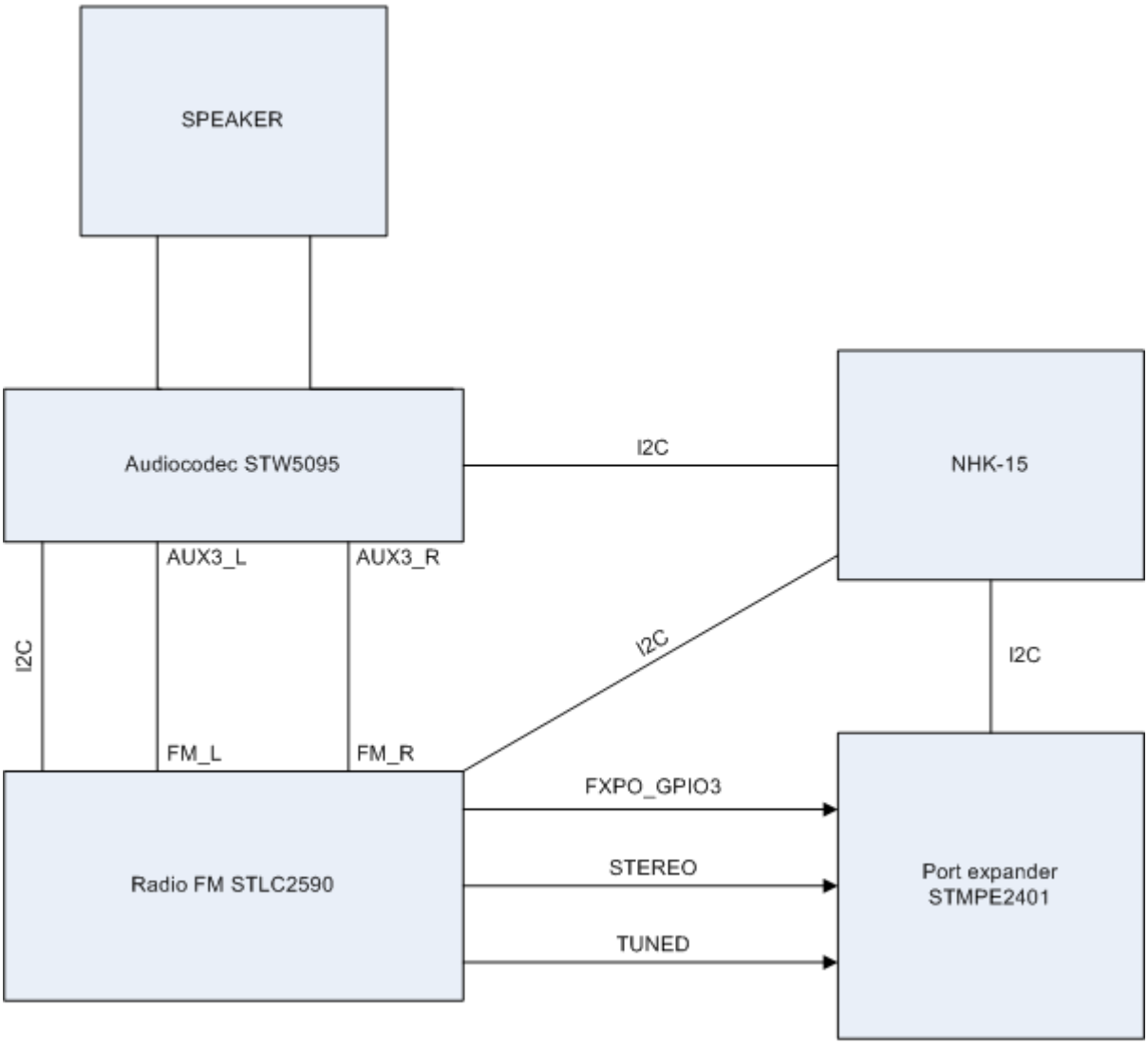

Figura 1: I collegamenti fra i chip nella board NHK-15

5 Dettagli dell'implementazione

Nella implementazione del driver abbiamo cercato di lasciare separate le funzione che implementano l'interfaccia esportata da Video4Linux dalle funzione i cui ruoli è quel di pilotare il chip: mentre le prime fanno le conversioni di unità, restituiscono i valori delle variabile e hanno una logica limitata, le ultime gestiscono gli scambi dati con il chip.

Per miglior comprendere ogni parte abbiamo suddiviso il codice nelle cinque principale funzione del driver che sono, in ordine: inizializzazione, sincronizzazione di una data frequenza, sistematizzazione dei controllori audio, finalizzazione del driver e in fine la parte relativa al RDS. Ognuna di queste verrà spiegata nelle sue sottosezioni più avanti. Prima di farlo ci conviene spiegare le funzione `refresh_read` e `refresh_write` che sono usate in queste sottosezioni.

Il chip STLC2590 non supporta la lettura o scritta di un determinato registro. Invece ogni volta facciamo una lettura o scritta il chip comincia a leggere dal registro 0AH in poi e a scrivere dal 02H in poi rispettivamente. Le funzione `refresh_read` e `refresh_write` servono per astrarre questo fatto. Sotto le sue descrizioni:

- `stlc2590_refresh_read`: invia al chip via I2C un comando per leggere tutti i registri e salva i suoi valori nella variabile `regs`, dentro la struttura `dev`. Questa variabile serve come un cache dei registri presenti nel chip. Così, ogni volta si vuole leggere un registro in cui non è certo che il suo valore nel cache sia il più recente, si deve fare prima una chiamata a questa funzione per aggiornarlo.
- `stlc2590_refresh_write`: invia al chip via I2C un comando per scrivere tutti i registri conforme i valori presente nel cache. Così, ogni volta si vuole scrivere un registro, si aggiorna prima il suo valore nel cache e poi si fa una chiamata a questa funzione che è in grado di aggiornare i valori presenti nel chip.

Nella Figura 2 c'è un schematico di tutte le interfacce tra spazio utente, Video4Linux, funzioni del driver, sysfs e i chip. Essa è usata in tutta questa sezione per spiegare l'implementazione del driver.

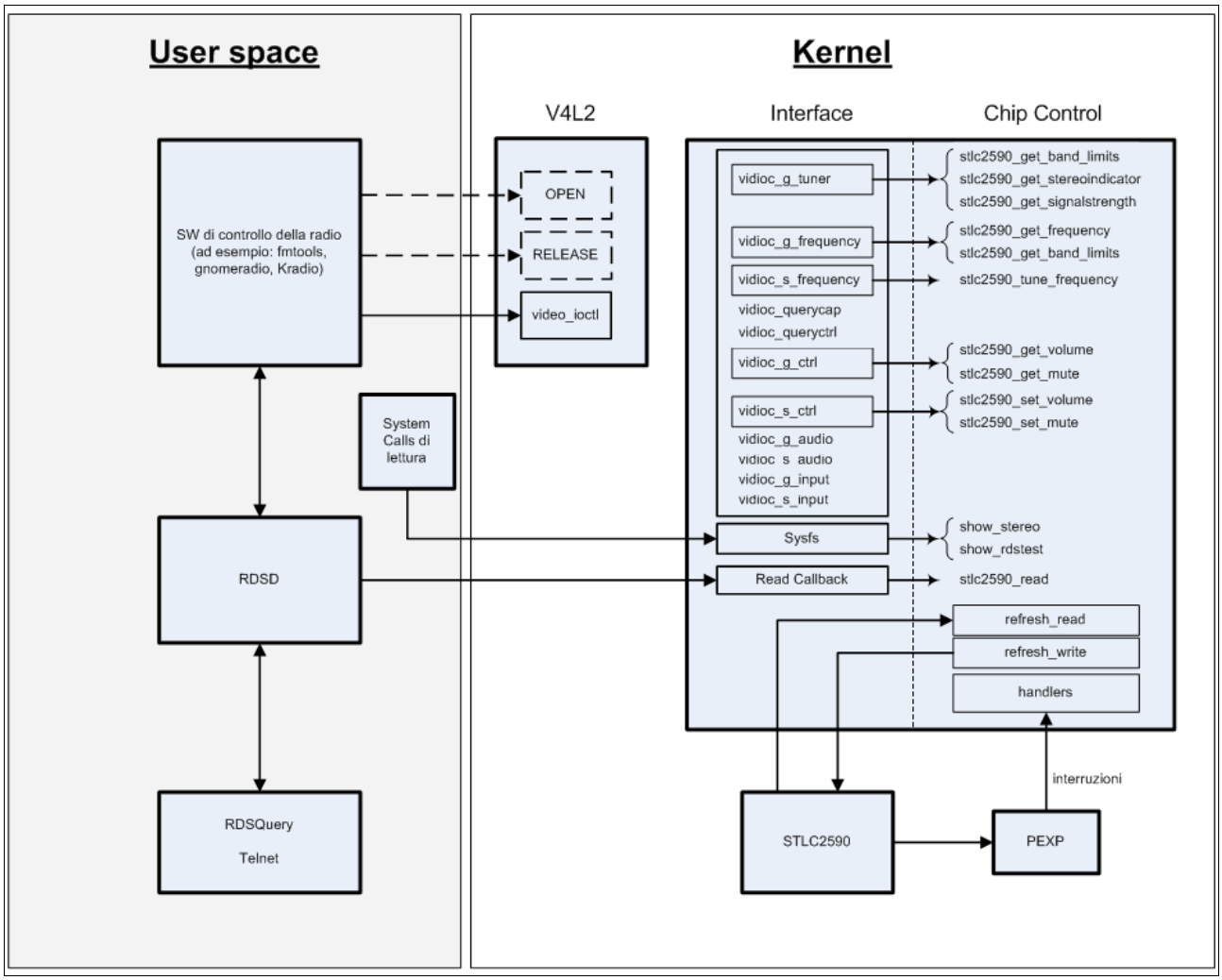

Figura 2: Interfacce tra spazio utente, kernel e dispositivi

5.1 Inizializzazione

La funzione `stlc2590_init` è responsabile per fare l'inizializzazione del driver di radio che avviene nel momento in cui il modulo è inserito nel kernel. La Figura 3 mostra la sequenza delle chiamate fatte in questa funzione. Un riassunto di queste chiamate potrebbe essere, in ordine:

i. Configurare l'instradamento dell'audio dentro l'audiocodec e i registri per avviare il chip conforme dettagliato nel suo datasheet;

ii. Settare i valori default dei registri;

iii. Sintonizzare una stazione qualsiasi;

iv. Creare le entrate nel sysfs;

v. Inizializzare le irq relativi alle interruzioni di RDS.

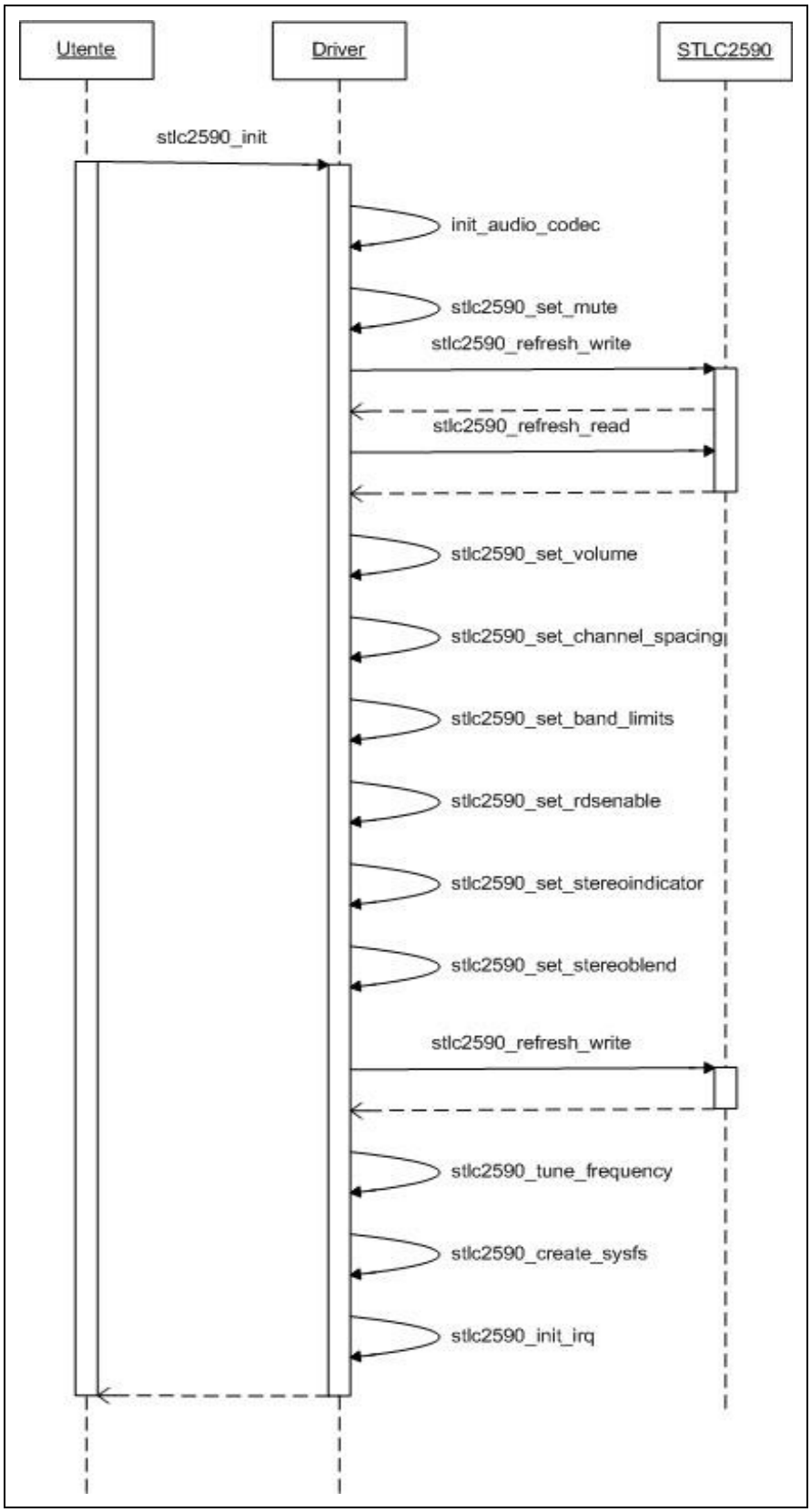

Figura 3: La funzione `stlc2590_init`

5.2 Sintonizzazione

Le chiamate di sintonizzazione avvengono ogni volta l'utente vuol cambiare la stazione attualmente sintonizzata e la funzione che la fa è `stlc2590_tune_frequency`. Come si vede in Figura 3 è proprio la stessa funzione chiamata all'inizializzazione per sintonizzare su una stazione qualsiasi (scelta come parametro del modulo).

L'operazione di sintonizzazione è iniziata configurando il bit TUNE con livello basso, dopodiché il channel desiderato è messo nei bit CHAN[9:0]. Per provare se l'operazione è stata fatta con successo, il bit TUNE è configurato al livello alto, e il driver verifica per un intervallo di tempo se il bit STC è configurato al livello alto. Quando se ne verifica il livello alto l'operazione è conclusa configurando il bit TUNE al livello basso un'altra volta. Come il valore del channel dipende dai valori di *channel spacing*[1] e di *band limits*[2], la prima cosa a fare è leggere i suoi valori dopodiché si può procedere con la sintonizzazione della stazione.

La Figura 4 mostra la sequenza delle sue chiamate.

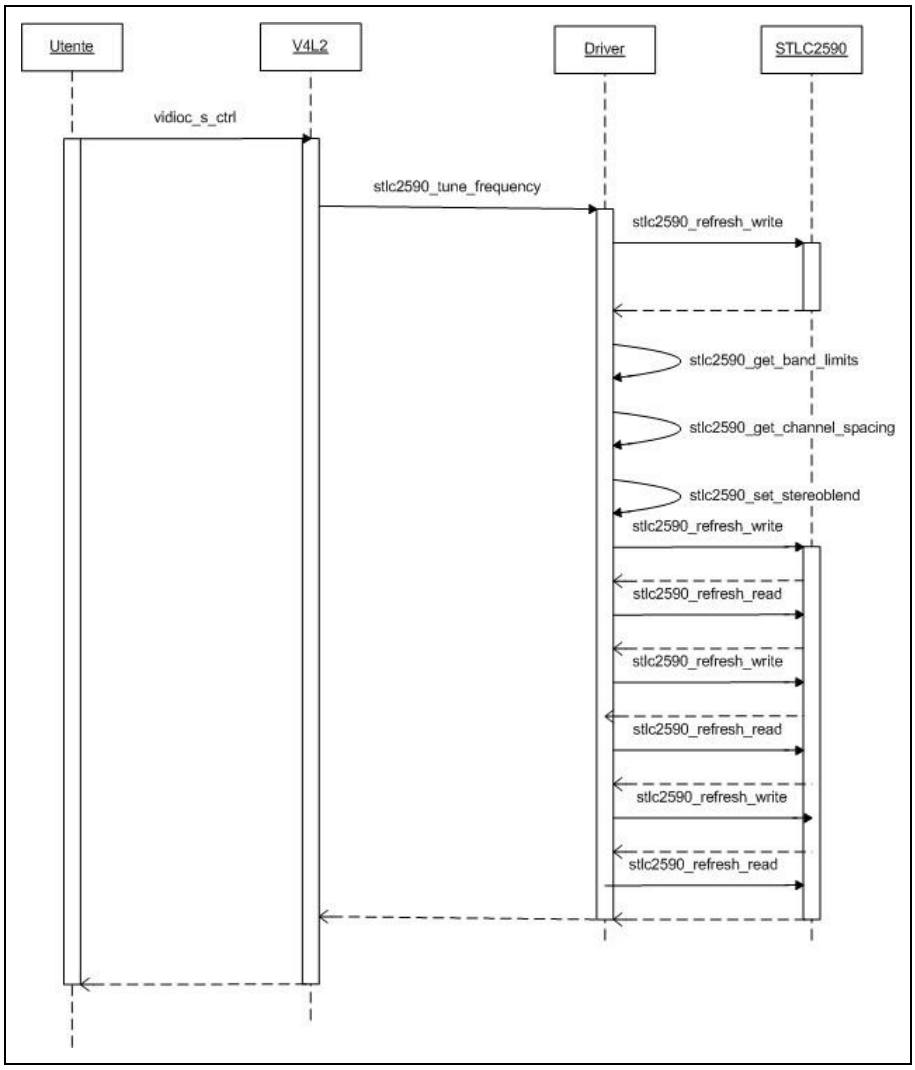

Figura 4: La sintonizzazione di una stazione radio

1 Il minimo spazio in frequenza che può esistere fra due stazione

2 I limiti di frequenza dentro cui si deve cercare le stazione radio.

5.3 Controllori audio

L'Interfaccia di V4L2 permette che due controllori di audio siano configurati: mute e volume. Essi vengono sistemati tramite le IOCTLs `vidioc_g_ctrl` e `vidioc_s_ctrl` (come visto nella Tabella 2.4) in cui uno dei suoi parametri riporta il controllore che si vuol configurare.

Nel chip STLC2590, l'uscita dell'audio può essere mutata configurando i registri MUTE e SMUTE (*soft mute*). La caratteristica *soft mute* è disponibile per attenuare l'uscita dell'audio e minimizzare rumori in condizione di basso segnalo. Per controllare il volume della radio, si configura i registri VOLUME[3:0]. La Figura 5 mostra la configurazione di mute e soft mute mentre nella Figura 6 c'è il controllo di volume.

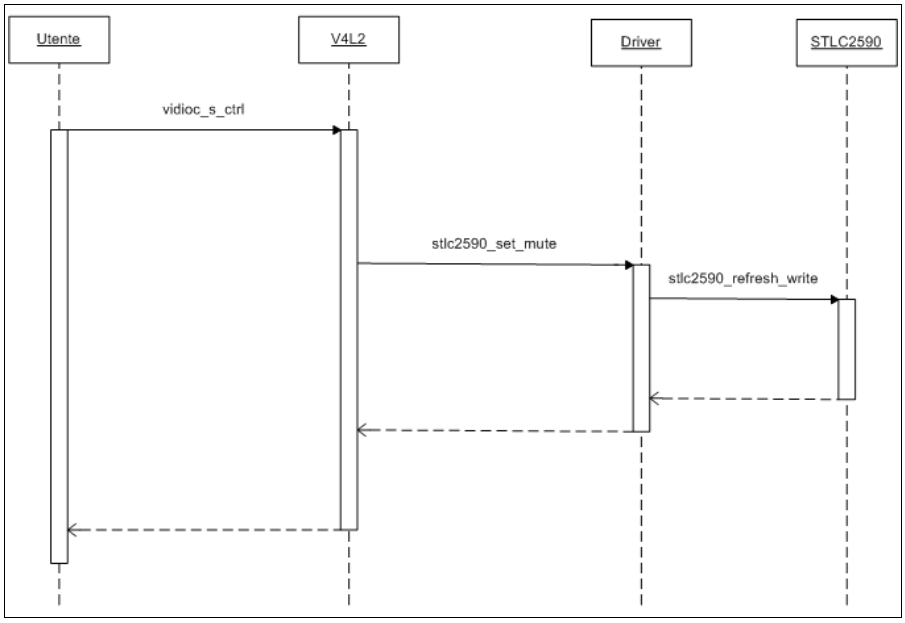

Figura 5: Configurazione dei controllori Mute e Soft Mute

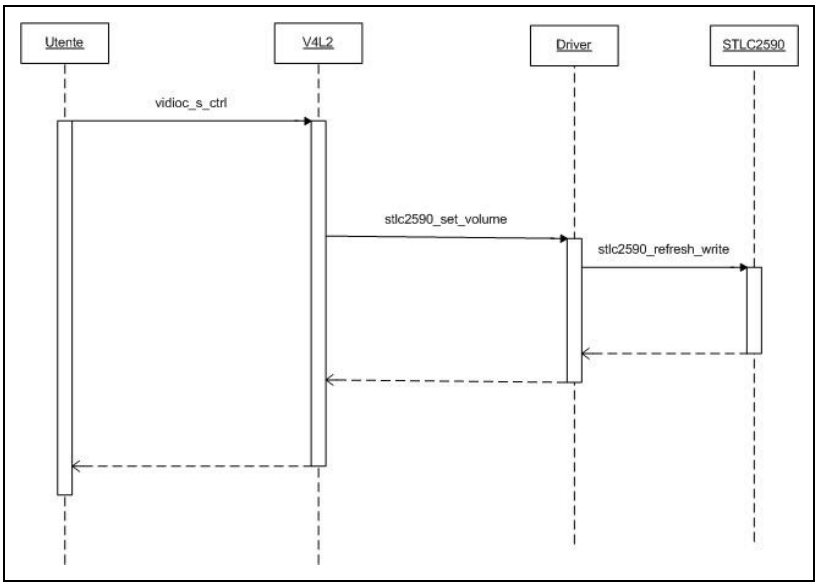

Figura 6: Controllore di volume

5.4 Finalizzazione

Per finalizzare e rimuovere il modulo del kernel, prima si deve configurare la rimozione delle interruzioni (affinché il kernel non passi più il controllo della CPU al driver quando esse arrivano) nonché rimuovere il driver del sysfs che sono le due interfacce con lo spazio utente. Poi si configura il chip per chiudere silenziosamente e infine si rimuove il driver del kernel. La finalizzazione è illustrata nella .

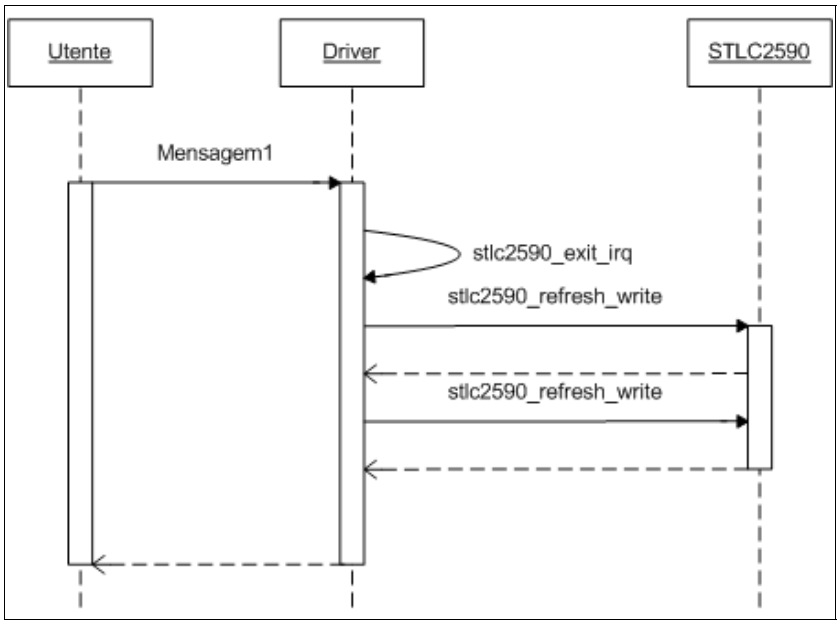

Figura 7: Finalizzazione del driver

5.5 RDS

La funzionalità RDS/RBDS è abilitata configurando il registro RDS. Il chip offre due modalità RDS: una modalità standard e l'altra verbose controllabile attraverso il registro RDSM. La differenza tra le due modalità è che nella verbose c'è una visibilità incrementata degli errori nei blocchi di dati e della qualità del segnalo. Siccome non vogliamo ricevere i dati RDS tramite *pooling*, dobbiamo abilitare le interruzioni nel registro RDSIEN e configurarle perché arrivino nel driver. La sua configurazione è fatta appena il modulo è inserito nel kernel chiamando la funzione `stlc2590_init_irq` come può essere visto nella Figura 3.

Lo schematico nella mostra come il driver legge i dati RDS provenienti dal chip usando le chiamate di interruzione e come l'utente può leggerli indipendentemente dal momento in cui sono arrivati.

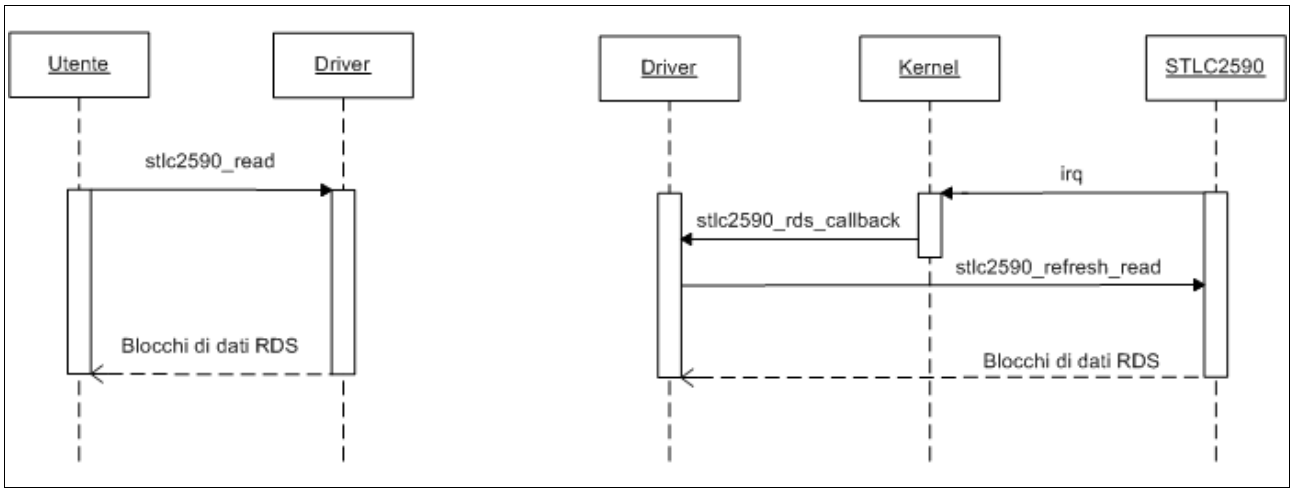

Figura 8: Gli scambi di dati RDS tra utente e driver (sinistra) e tra il driver e il chip (destra)

6 Test cases

I test cases vengono fatte per provare il funzionamento della radio e saranno spiegati dal punto di vista dell'utente. Come dettagliato su [Cross-compiling e tools utilizzate] useremo fmtools per pilotare il driver ed il rdsd per ricevere i dati RDS.

L'opzione "--help" del programma "fm" che è fornito dentro il pacchetto di fmtools ci dà tutte le possibile comandi per pilotare il driver:

```
$ ./fm -h

fmtools fm version 1.0.2

usage: ./fm [-h] [-o] [-q] [-d <dev>] [-t <tuner>] [-T none|forever|
time] <freq>|on|off [<volume>]

A small controller for Video for Linux radio devices.

  -h         display this help
  -o         override frequency range limits of card
  -q         quiet mode
  -d <dev>   select device (default: /dev/radio0)
  -t <tuner> select tuner (default: 0)
  -T <time>  after setting frequency, sleep for some time
             (default: none; -=forever)
  <freq>     frequency in MHz (i.e. 94.3)
  on         turn radio on
```

```
off         turn radio off (mute)
+           increase volume
-           decrease volume
<volume>    intensity (0-65535)
```

La board NHK-15 usata nelle prove ha soltanto uno dispositivo di radio e perciò tutte le volte che il driver è inserito nel kernel si registra con gli stessi “minor number” e “major number” cioè quelli che corrisponde al device file /dev/radio0. Siccome il default è che fmtools sistemi il driver corrispondente a /dev/radio0 non c'è bisogno di specificare tutte le volte qual è il device file che deve essere usato. Nei test case non ci preoccuperemo dunque con questo parametro.

6.1 Sintonizzazione

Questa è l'operazione più importante e ci permette di cambiare la stazione sintonizzata. Se l'operazione è completata con successo sarà possibile sentire la nuova radio sintonizzata e inoltre fmtools ci dà un messaggio che tra l'altro ci dice . Per esempio, se volessimo sintonizzare la radio “One o One”, 101.2 MHz, dovremmo fare:

```
$ ./fm 101.2
Radio tuned to 101.2 MHz at 100.00% volume
```

Nel caso in cui provassimo di dare una frequenza fuori dell'intervallo permesso (configurato all'inizializzazione), ci ritornerebbe il messaggio di errore:

```
$ ./fm 74.0
Frequency 74.0 out of range (87.5 – 108.0 MHz)
```

6.2 Controllori dell'audio

Alcuni chip di sintonizzazione permettono anche di sistemare qualche parametro di audio. Quelli più comuni, che sono pure quelli presenti nel chip STLC2590, sono il controllore di volume e il di mute/unmute. In generale il controllore di volume è poco usato perché sono pochi precisi e si può sistemare il volume del sistema traverso l'API di Alsa. Però, se si vuole usarlo fmtools lo sa pilotare.

Per rendere il chip in modo silenzioso, c'è bisogno di eseguire il seguente comando:

```
$ ./fm off
Radio muted
```

e per attivarlo:

```
$ ./fm on
Radio on at 100.00% volume
```

La configurazione del volume che si desidera può essere fatta di due forme: i) come parametro di “fm” in una scala da 0 a 65535; ii) usando i simboli “+” e “-” per aumentare e

abbassare il volume da uno passo (il numero di passi è configurato dal driver).

```
$ ./fm 101.2 65535
Radio tuned to 101.2 MHz at 100.00% volume
$
$ ./fm -
Setting volume to 87.50%
$ ./fm +
Setting volume to 100.00%
```

6.3 Inserimento del modulo nel kernel

Quando il driver è inserito nel kernel il chip è avviato in modo silenzioso, i parametri del modulo sono configurati e il chip è pronto per fare suonare una stazione di radio qualsiasi. I possibili parametri del modulo sono: `channel_spacing`, `band_limits`, `rds_enable`, `stereo_blend`, `external_amp_enable`.

Per inserire il modulo nel kernel con tutte le opzione default e essendo sotto la cartella dove c'è il modulo compilato, basta il seguente comando*:

```
# insmod stl2590.ko
```

Per configurare il chip per usare i limiti dell'intervallo di frequenza come quelli usati in Giappone:

```
# insmod stl2590.ko band_limits=0x01
```

Per configurare inoltre lo spaziamento tra le stazione:

```
# insmod stl2590.ko band_limits=0x01 channel_spacing=0x01
```

E così via per tutti gli altri parametri.

6.4 Rimozione del modulo dal kernel

È possibile rimuovere il modulo dal kernel senza avere bisogno di riavviare la board:

```
# rmmod stl2590
```

6.5 RDS

Il controllo del funzionamento di RDS è un po' diverso dai precedenti perché fmtools non è in grado di gestire i dati che arrivano via RDS. Anzi è necessario un altro programma chiamato “rdsd” che interroga il driver sui dati RDS, fa il parsing delle informazioni ottenute e lasciale disponibile agli altri programmi nello spazio utente via TCP/IP.

La prima cosa che rdsd fa quando acceso è aprire il suo archivio di configurazione (default

* Attenzione: questa operazione si fa come root (indicata dal “#” prima del comando)

in /etc/rdsd.conf) in cui legge tutte le sue configurazioni, apre un socket per ascoltare ad una porta predefinita (la default è 4321), poi interroga continuamente il chip sui dati RDS che arrivano (aggiornando le sue variabile interne) e accetta connessioni dei programmi clienti nella stessa porta.

L'interroga del daemon rdsd può essere fatta di vari forme: i) usando il programma "rdsquery" che è presente nel pacchetto di rdsd; ii) usando un programma più sofisticato dei fmtools con supporto a RDS; iii) usando una connessione TCP via telnet e facendo i comandi a mano. Siccome i comandi non sono difficile scegliamo di usare telnet. Nella Tabella 6.1 c'è l'elenco dei comandi disponibile e le loro relazioni con i campi RDS.

Comando[3]	**Descrizione**
`0:rxfre`	Frequenza sintonizzata attualmente
`0:rxsig`	Potenza di segnale
`0:rflags`	Alcuni flags
`0:picode`	Identificatore del programma
`0:ptype`	Identificatore del tipo di programma
`0:locdt`	Data e orario
`0:utcdt`	Data e orario in formato UTC
`0:rtxt`	Buffer di testo attuale
`0:lrtxt`	Ultima stringa di "radio text" che si è ricevuta completamente
`0:tmc`	Lista di messaggi sul traffico
`0:Aflist`	Lista di frequenze alternative
`0:gstat`	Statistica sui gruppi RDS

Tabella 6.1: Elenco dei comandi disponibile nel rdsd

Il comando per avviare rdsd e un esempio di uso:

```
$ ./rdsd &
[1] 17785
$ telnet localhost 4321

0:pname

R 101
```

In questo caso di sopra siamo connessi con il rdsd via TCP/IP, che per default ascolta sulla porta 4321, e poi abbiamo chiesto qual è il nome della statistica attualmente sintonizzata. Come si può vedere il driver ci ha risposto "Radio one o one": R 101.

3 Il "0" prima del comando significa "il dispositivo 0" configurabile su rdsd.conf.

6.6 SysFS

Come spiegato in [2.3-SysFS] il driver esporta alcune variabile nel sysfs. L'elenco delle variabile è: stereo, rdstest. Per leggere qualsiasi variabile basta leggere i corrispondenti archivi che sono sotto /sys/class/video4linux/radio0. Esempio:

```
$ cat /sys/class/video4linux/radio0/stereo
mono
$ cat /sys/class/video4linux/radio0/rdstest
11540
$ ./fm 103.1
Radio tuned to 103.1 MHz at 100.00% volume
$ cat /sys/class/video4linux/radio0/stereo
mono
$ cat /sys/class/video4linux/radio0/stereo
```

Bibliografia

1: STMicroeletronics, STLinux, 2008, http://www.stlinux.com/

2: Deucher A., Walls A. et al, Video4Linux, 2008, http://linuxtv.org/v4lwiki/

3: RDS, Radio Data System, 2008, http://www.rds.org.uk/

4: Mochel Patrick, The sysfs Filesystem, 2005,
http://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf

5: Deucher A., Walls A. et al, V4L2 API, 2008, http://v4l2spec.bytesex.org/

6: OpenedHand Ltd., Poky Linux, 2008, http://www.pokylinux.org/

7: Kroll Russel, Pfaff Ben, Fmtools, 2006, http://www.stanford.edu/~blp/fmtools/