Kernel Hacking: aggiungere il supporto per un dispositivo ARM embedded

Claudio Imbrenda

Linux Day 2009 PISA

24 ottobre 2009

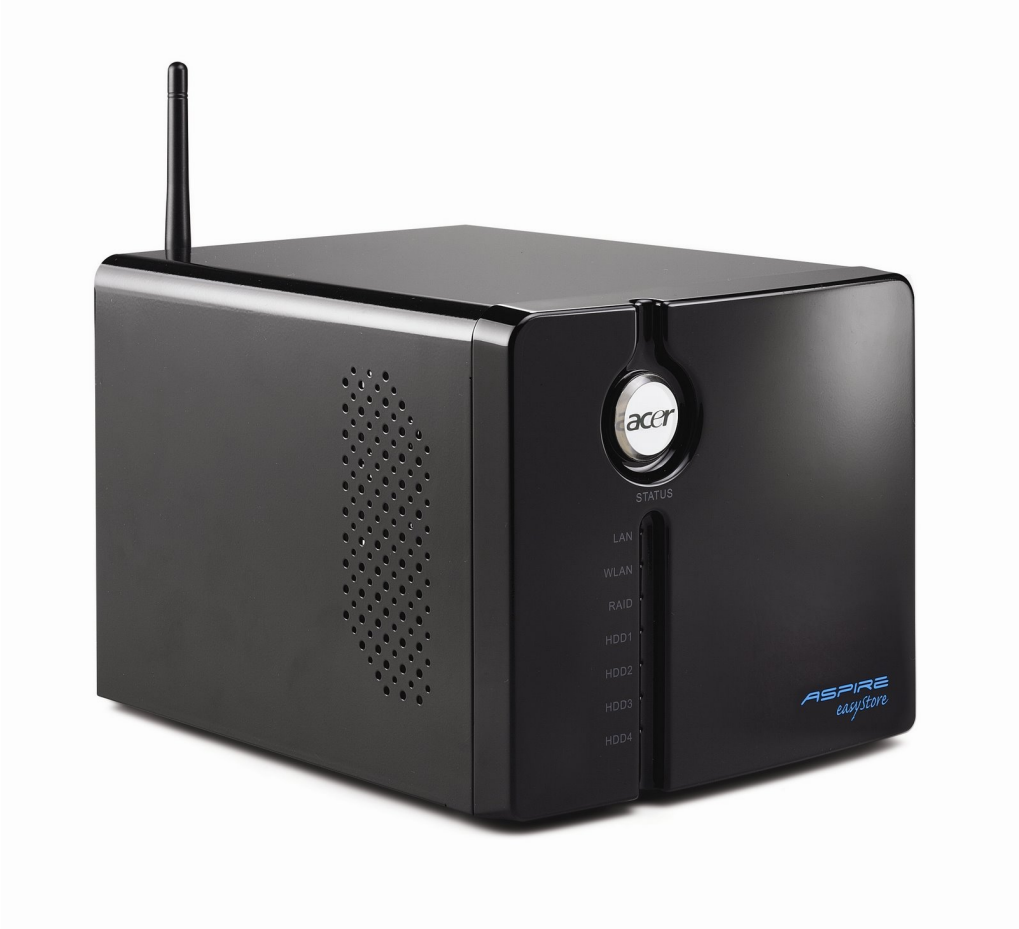
Cosa?

Cosa è un dispositivo ARM embedded?

Cosa?

Cosa è un dispositivo ARM embedded?
È un dispositivo integrato, basato su processore ARM.

Questo è un dispositivo ARM embedded:



Acer ASPIRE easyStore

Come l'ho avuto

Un mio amico ha avuto problemi col firmware originale (a un certo punto gli impediva totalmente l'accesso ai propri file)

Il mio amico ha allora aperto il dispositivo e tolto i dischi.
I dischi li ha messi in un altro computer.
Il giochino (vuoto) lo ha regalato a me!

Come l'ho avuto

Un mio amico ha avuto problemi col firmware originale (a un certo punto gli impediva totalmente l'accesso ai propri file)

Il mio amico ha allora aperto il dispositivo e tolto i dischi.
I dischi li ha messi in un altro computer.
Il giochino (vuoto) lo ha regalato a me!

Grazie Vincenzo!

Cosa contiene?

Hardware:

Processore	Marvell Orion2 MV88F5281-D0 (ARM) 500MHz
RAM	128MB DDR
Flash	Spansion S29GL128N 11TFI01 (16 MB)
SATA	Marvell 88SX7042 PCI-e 4-port SATA-II
WiFi	RaLink RT2561/RT61 802.11g PCI
Ethernet	Gigabit MV-643xx
Controller Ventola	BOH? (Controllata via GPIO)
Sensore temperatura	Compatibile LM75
RTC	ST M41T80
USB	non c'è, ma il controller integrato va :)
Seriale	RS232 115200 8N1 TTL (standard 16550)

Cosa contiene?

software:

Bootloader	Das U-Boot (v. 1.11)
OS	Linux! (kernel 2.6.12)

Cosa contiene?

software:

Bootloader	Das U-Boot (v. 1.11)
OS	Linux! (kernel 2.6.12)

Aspetta.. Linux c'era già?!?

Non mainline

Effettivamente a bordo c'era già Linux. E non solo. Sul sito della Acer, un po' (MOLTO) nascosto, c'erano pure i sorgenti di tutto il software libero usato nel firmware originale.

Non mainline

Effettivamente a bordo c'era già Linux. E non solo. Sul sito della Acer, un po' (MOLTO) nascosto, c'erano pure i sorgenti di tutto il software libero usato nel firmware originale.
Facile la vita eh allora?

Non mainline

Effettivamente a bordo c'era già Linux. E non solo. Sul sito della Acer, un po' (MOLTO) nascosto, c'erano pure i sorgenti di tutto il software libero usato nel firmware originale.
Facile la vita eh allora?
Beh, no:

- il supporto nel kernel era fatto in modo ad-hoc dalla Marvell
- non c'è un modo umano per installare una qualunque distribuzione
- qualunque altra versione del kernel non supporta quel dispositivo
- non tutto il software del firmware è libero! (in particolare il controller della ventola)

Facilitazioni

Però il fatto che altri dispositivi basati su chip Orion2 fossero supportati mi ha aiutato notevolmente.

Facilitazioni

Però il fatto che altri dispositivi basati su chip Orion2 fossero supportati mi ha aiutato notevolmente.

Vediamo ora tutti i passaggi che ho compiuto per aggiungere il supporto.

ATTENZIONE!

La strada che illustrerò è quella che ho seguito io, che sicuramente non è la più breve, né la più facile!

Seriale

Il bootloader è su console seriale ed è capace di scrivere sulla flash, supporta inoltre la modalità “kermit” per il trasferimento dei file via seriale.
Questo è indispensabile nel caso si facciano errori nello scrivere sulla flash. (e SI FANNO!)
O se non si ha idea di come sia mappata la flash!

Seriale

Per prima cosa ho quindi costruito un adattatore seriale da RS232 standart a RS232 TTL.
Lo standard RS232 ha i voltaggi definiti nel seguente modo:

bit	voltaggio
1	$-5\text{v} \rightarrow -25\text{v}$
0	$5\text{v} \rightarrow 25\text{v}$

La porta TTL invece vuole:

bit	voltaggio
1	0v
0	3.3v

MAX232 e MAX3232

Si trovano su internet guide[1] su come fare un semplice adattatore da RS232 a TTL. Quelle guide indicano di usare un circuito integrato MAX3232, che è RARO. Ho quindi usato un semplice MAX232, mettendo un partitore di tensione all'uscita verso il dispositivo, e niente in ingresso, dal momento che il voltaggio del dispositivo rientra nella tolleranza del MAX232 e viene quindi riconosciuto!
L'unico problema è che il mio adattatore, oltre al connettore della seriale standard e a quello da attaccare al dispositivo, ha bisogno anche di un'alimentazione esterna da 5v.

[1] Tra cui i circuiti di esempio nella documentazione del MAX3232

Scoprire quale è TX e quale è RX

Una volta trovata fisicamente la seriale (facile! c'è scritto accanto), bisognava determinare quale pin è TX e quale è RX.

Per farlo ho semplicemente messo un tester (analogico!) e ho misurato il voltaggio dei vari pin durante il boot, ovvero quando sono sicuro che il kernel manda cose sulla seriale.

L'unico pin il cui voltaggio variava durante il boot era chiaramente TX, uno era fisso a 0 (GND), uno fisso a 3.3v, e uno a $2,4$v (RX).

Ho quindi costruito il connettore usando componenti di recupero.

Scoprire quale è TX e quale è RX

Una volta trovata fisicamente la seriale (facile! c'è scritto accanto), bisognava determinare quale pin è TX e quale è RX.

Per farlo ho semplicemente messo un tester (analogico!) e ho misurato il voltaggio dei vari pin durante il boot, ovvero quando sono sicuro che il kernel manda cose sulla seriale.

L'unico pin il cui voltaggio variava durante il boot era chiaramente TX, uno era fisso a 0 (GND), uno fisso a 3.3v, e uno a $2,4$v (RX).

Ho quindi costruito il connettore usando componenti di recupero.

Nel frattempo ho imparato a:

- saldare
- dissaldare
- usare una breadboard

Gli hard disk

Beh, non ci si fa molto con uno storage di rete senza hard disk.

Ho comprato dei normali dischi SATA interni da 1TB e li ho messi dentro.

Installare una distribuzione

Il passo successivo è stato cercare di installare una distribuzione di Linux. Ovviamente ho installato Debian.

Ho usato il nuovo port `armel`, che sfrutta la nuova e più efficiente ABI ARM.

Delle uova e delle galline

Beh.. come faccio ad installare una distribuzione di Linux se il dispositivo non è supportato?

Delle uova e delle galline

Beh.. come faccio ad installare una distribuzione di Linux se il dispositivo non è supportato?

Qui entra in gioco il supporto preesistente di altri dispositivi simili.

Ho infatti usato l'installer per un altro dispositivo simile: il `DNS323`.

Delle uova e delle galline

Beh.. come faccio ad installare una distribuzione di Linux se il dispositivo non è supportato?

Qui entra in gioco il supporto preesistente di altri dispositivi simili.

Ho infatti usato l'installer per un altro dispositivo simile: il `DNS323`.
Ovviamente non poteva andare tutto bene. In particolare:

- le "partizioni" della flash non tornano. Come conseguenza il kernel non può essere scritto sulla flash.
- la ventola non gira e non c'è modo di avviarla. Ho quindi dovuto usare l'easyStore aperto, per evitare surriscaldamenti

Delle uova e delle galline

Ho più fortuna con il kernel per un altro dispositivo, il `QNAP TS-409`.

Le partizioni vengono sempre viste in modo errato, ma almeno la procedura di flashing avviene in modo corretto.

Ad ogni boot devo dare a mano il comando tramite seriale, perché kernel e initrd vengono messi in posizioni in cui il bootloader non le cerca.

Sistema installato, e ora?

Ora che ho un sistema Debian installato, posso iniziare lo sviluppo vero e proprio.

Capire cosa c’è e cosa non c’è

Innanzitutto vedo un po’ cosa ho a disposizione. Con questi comandi ho visto cosa veniva riconosciuto:

```
dmesg
lspci
lsusb
```

Cose che vanno e cose che non vanno

Questo è un sommario delle varie cose che vanno e che non vanno con i due kernel che ho provato:

	Ventola	WiFi	SATA	Sens.temp.	Flash	Pulsanti	LED
DNS323	NO	NO	Sì	Sì	NO	Sì	alcuni
QNAP TS-409	Sì, fissa	NO	Sì	NO	quasi	NO	alcuni

GPIO

Alcuni processori (soprattutto i SoC integrati come l'Orion2) hanno dei piedini "generici" direttamente controllabili da software.

Possono essere usati per vari scopi:

- Controllare LED
- Ricevere eventi di pulsanti
- Porta seriale
- Segnalazione interrupt
- …

Provare i LED

Alcuni pin GPIO vengono mappati su LED, è quindi facile per me provare ad accenderli e spegnerli per vedere l'effetto che fa.

Successivamente, quando già avevo iniziato a scrivere il supporto, ho mappato tutti i pin GPIO come LED e li ho provati uno per uno.

GPIO

Andando per tentativi (e facendo delle prove, ovvero tante ore di compilazione) ho scoperto almeno i seguenti collegamenti:

pin	cosa fa
0	Interrupt PCIe
1	Power LED (rosso)
4	Power LED (blu)
7	RTC (?)
8	Spegnimento
9	Pulsante accensione
10	Pulsante reset
11	Interrupt Slot 0 PCI
13	Ventola (lenta)
14	Ventola (veloce)
15	LED WiFi (attivo basso)

Le “partizioni” della flash

Analizzando le informazioni ricavate dal boot loader e quelle ricavate dal firmware originale (ahh.. se avessi salvato quelle informazioni PRIMA, invece di dover rimettere il firmware originale solo per controllare), ho determinato il seguente mapping delle partizioni della flash:

Partiz.	Inizio	Dimens.	Nome	Descrizione
0	`0x000000`	128k	MTD1	Dati NVRAM del firmware originale
1	`0x020000`	128k	MTD2	Dati NVRAM del firmware originale
2	`0x040000`	1536k	Linux Kernel	Il kernel
3	`0x1C0000`	14080k	File System	Initrd
4	`0xF80000`	512k	u-boot	Das U-Boot bootloader

Il boot loader si aspetta delle immagini u-boot valide sia per il kernel che per l’initrd.

La partizione del boot-loader è read-only. Non c’è modo di scriverci sopra, neanche tentando di sproteggerla da scrittura tramite la console seriale del boot-loader.

La ventola

Gli altri dispositivi che montano chip Soc Orion2 interagiscono con la ventola che hanno (se la hanno) in uno di due modi:

- Controller di ventola su smbus, come sul DNS323
- Controller di ventola su seriale, come sul QNAP TS-409

Ovviamente l'easyStore si comporta in modo differente. La ventola è controllata tramite due pin GPIO:

Pin 13	Pin 14	Ventola
0	0	Ferma
1	0	Lenta
—	1	Veloce

Da notare che anche durante l'estate la ventola lenta ha tenuto la temperatura interna dei dischi e della scheda madre sotto i 50°C, anche durante uso intensivo del processore.

Il bus PCI

Il bus PCI non è usato né dal DNS323, né dal QNAP TS-409.

Per fortuna altri dispositivi (schede di sviluppo) lo supportano, e quindi ho fatto un copia e incolla delle (poche) linee di codice necessario a farlo funzionare.
Ci avrei messo moltissimo tempo di più altrimenti.

Anche in questo caso mi sono state utili le informazioni che ho reperito dal firmware originale. (in particolare il mapping degli interrupt)

La wireless

La wireless è l'unica cosa sul bus PCI, una volta fatto funzionare il bus PCI, funziona anche la wireless.

Tuttavia una volta che la scheda wireless viene spenta, quando poi viene fatta ripartire successivamente non ha più la radio attiva.

Sospetto che ci sia un pin GPIO che controlla la radio.

La ethernet

La scheda ethernet funziona felicemente coi driver del kernel.

L'unico problema è che il MAC-address non è rilevato.

La ethernet

La scheda ethernet funziona felicemente coi driver del kernel.

L'unico problema è che il MAC-address non è rilevato.

Come in altri dispositivi simili, il MAC-address è memorizzato come `stringa` sulla flash, nella partizione del boot-loader.

La ethernet

La scheda ethernet funziona felicemente coi driver del kernel.

L'unico problema è che il MAC-address non è rilevato.

Come in altri dispositivi simili, il MAC-address è memorizzato come `stringa` sulla flash, nella partizione del boot-loader.

Da qualche parte...

La ethernet

La scheda ethernet funziona felicemente coi driver del kernel.

L'unico problema è che il MAC-address non è rilevato.

Come in altri dispositivi simili, il MAC-address è memorizzato come `stringa` sulla flash, nella partizione del boot-loader.

Da qualche parte... ma basta cercare con `hexdump`.

Il registro delle sottoarchitetture

Ogni sottoarchitettura o dispositivo integrato ARM ha un numero che li identifica (`mach-type`). In questo modo è possibile avere una stessa immagine del kernel che supporti dispositivi integrati diversi, le cui differenze sono sufficienti da giustificare una (sotto-)sottoarchitettura differente, ma altrimenti molto simili (come nel nostro caso).

Il registry è sul sito:
`http://www.arm.linux.org.uk/developer/machines/`

Ho registrato l'easyStore: ha `mach-type` 2379.

mach-type

Il kernel riceve il parametro `mach-type` dal boot-loader in un registro della CPU. Si suppone infatti che il boot loader sappia esattamente quale sia la sottoarchitettura su cui sta facendo bootstrap.

Ovviamente il boot-loader che c'è non usa il `mach-type` giusto.

E, come già detto prima, non c'è la possibilità di cambiare il boot-loader.

Si ricorre quindi a un barbatrucco: una volta generata l'immagine del kernel, si aggiungono 8 byte all'inizio: due istruzioni in codice macchina in cui si carica a mano il valore giusto nel registro, rimediando all'incapacità del boot-loader. Questa operazione verrà poi fatta in automatico dallo script che si occupa di flashare i kernel.

easysote-setup.c

Tutto il supporto per una dispositivo come nel caso è contenuto generalmente in un unico file, in cui sono inizializzate delle strutture e dichiarate delle funzioni che verranno poi usate durante la fase di inizializzazione del kernel per inizializzare l'hardware essenziale.

Io ho copiato da quelli del DNS323 e del QNAP TS-409, apportando le opportune modifiche, ovviamente.

Cross-compilare il kernel

Cross-compilare il kernel è senz'altro possibile, e anzi, se non si ha esigenza di creare un pacchetto debian col kernel, è desiderabile.

Cross-compilare il kernel

Cross-compilare il kernel è senz'altro possibile, e anzi, se non si ha esigenza di creare un pacchetto debian col kernel, è desiderabile.

Compilare su un ARM a 500MHz

Cross-compilare il kernel

Cross-compilare il kernel è senz'altro possibile, e anzi, se non si ha esigenza di creare un pacchetto debian col kernel, è desiderabile.

Compilare su un ARM a 500MHz è

Cross-compilare il kernel

Cross-compilare il kernel è senz'altro possibile, e anzi, se non si ha esigenza di creare un pacchetto debian col kernel, è desiderabile.

Compilare su un ARM a 500MHz è una

Cross-compilare il kernel

Cross-compilare il kernel è senz'altro possibile, e anzi, se non si ha esigenza di creare un pacchetto debian col kernel, è desiderabile.

Compilare su un ARM a 500MHz è una cosa

Cross-compilare il kernel

Cross-compilare il kernel è senz'altro possibile, e anzi, se non si ha esigenza di creare un pacchetto debian col kernel, è desiderabile.

Compilare su un ARM a 500MHz è una cosa moooltooo

Cross-compilare il kernel

Cross-compilare il kernel è senz'altro possibile, e anzi, se non si ha esigenza di creare un pacchetto debian col kernel, è desiderabile.

Compilare su un ARM a 500MHz è una cosa moooltooo luuungaaaa

Cross-compilare il kernel

Cross-compilare il kernel è senz'altro possibile, e anzi, se non si ha esigenza di creare un pacchetto debian col kernel, è desiderabile.

Compilare su un ARM a 500MHz è una cosa moooltooo luuungaaaa …

Cross-compilare il kernel

Cross-compilare il kernel è senz'altro possibile, e anzi, se non si ha esigenza di creare un pacchetto debian col kernel, è desiderabile.

Compilare su un ARM a 500MHz è una cosa moooltooo luuungaaaa ……

Perché non funziona?

Peccato che io non sia riuscito a far funzionare `make-kpkg`[2] col cross-compiler.

Non ho ancora capito perché.

[2] Lo strumento Debian per creare pacchetti `.deb` del kernel personalizzati, partendo dai sorgenti

Pacchetto Debian

Il pacchetto Debian l'ho quindi creato sull'easyStore stesso. In questo modo non ho avuto alcun problema di cross-compilazione.

Pacchetto Debian

Il pacchetto Debian l'ho quindi creato sull'easyStore stesso. In questo modo non ho avuto alcun problema di cross-compilazione.

Basta non aver fretta.

Non c’entra!

La partizione del kernel, come già mostrato, è grande esattamente 1.5MB. Il kernel da me compilato, con praticamente le stesse opzioni dei kernel Debian che avevo usato sugli altri dispositivi, viene grande circa 1.7MB.

Ho quindi dovuto eliminare un po’ di cose inutili.

Conclusione

Sono quindi riuscito a far funzionare il mio Acer Aspire easyStore in modo corretto. Appena ho un po' di tempo proporrò la patch agli sviluppatori del kernel.

Fine

Happy hacking!