LINUX EMBEDDED

Ottavio Campana

LinuxDay 2008

Soluzioni per Linux Embedded

Esistono diverse soluzioni per sviluppare sistemi embedded basati su Linux, alcuni free, altri commerciali:

* Montavista
* Embedded Debian Project
* uClinux
* OpenEmbedded

Cosa serve per realizzare un sistema Linux Embedded?

* La cosa fondamentale è avere un cross compiler per l'architettura desiderata. Questo è fornito da tutte le soluzioni per Linux Embedded;
* Molte soluzioni forniscono tool per automatizzare la build dei sistemi;
* Sono disponibili, sebbene opzionali, sistemi di gestione del software (rpm, deb, ipk, opk, ...)

Montavista

Montavista vende *subscriptions*, ovvero:

* software;
* documentazione;
* supporto tecnico.

accessibili subordinatamente ad un canone.

Sono disponibili tre versioni: Professional Edition, Carrier Grade Edition e Mobilinux.

Montavista

Punti di forza di Montavista:

* hard real time
* DevRocket, un IDE basato su Eclipse
* supporto commerciale.

Embedded Debian Project

Come non citare la distribuzione più amata dal Lug Vicenza?

Embedded Debian Project ha come scopo l'estensione dei tool standard di installazione di Debian (apt, dpkg, ...) e dei pacchetti (debian/rules, CDBS, ...) per estendere il supporto ai sistemi Embedded.

Embedded Debian Project

Punti di forza di Embedded Debian Project:

* la maggior parte dei pacchetti Debian possono essere facilmente ricompilati ed installati con gli stessi tool di Debian;
* se la minimizzazione della dimensione del rootfs non è critica, gli aggiornamenti del software sono facilitati.

uClinux

* E’ stato uno dei primi progetti per l’applicazione su sistemi embedded di Linux.
* Nacque inizialmente come progetto per portare Linux su architetture non dotate di MMU, come per esempio Coldfire.
* Il supporto per architetture MMU-less è stato integrato nel tree ufficiale di Linux.

OpenEmbedded

Non è una distribuzione per sistemi embedded.

E' un framework per la generazione di sistemi embedded.

* E' composto da:
* bitbake, un esecutore di *ricette*;
* Le ricette di bitbake;
* La pre-configurazione per moltissime architetture;
* Ricette ad-hoc, dette *distribuzioni*, per generare rootfs diversi.

OpenEmbedded

Punti di forza di OpenEmbedded:

* sviluppata molto attivamente;
* ampio numero di architetture, software e distribuzioni, che dà origine ad una vastissima quantità di sistemi generabili;
* offre ipkg come strumento opzionale di gestione dei software installato;
* e' la base da cui è iniziato lo sviluppo di OpenMoko.

Bitbake

La compilazione di un sistema embedded richiede molti passaggi, soprattutto quando la sua generazione avviene *from scratch*, ovvero quando si ha a disposizione solo un compilare per la propria architettura. La generazione di un sistema ex novo richiede, tra gli atri passaggi:

* la compilazione di gcc (tre volte);
* la compilazione di glibc (tre volte) o di uclibc;
* la compilazione di tutti i comandi di base del sistema per il boot.

Bitbake

Per automatizzare tutte queste compilazioni, bitbake permette di definire delle ricette che gestiscono:

* le opzioni da passare al processo di compilazione affinché il software sia compatibile con il sistema per cui viene compilato;
* le informazioni di interdipendenza, per schedulare la compilazione dei pacchetti nell'ordine corretto;
* eventuali procedure di QA, per verificare la genuinità del software generato.

Supporto per architetture

Il numero di architetture è molto vasto. Esso include diversi sistemi basati sulle varie famiglie degli armi, power pc, x86, sh.

Ogni architettura è poi specializzata per le diverse schede disponibili: palmari, telefonini, schede di valutazione dei vari sistemi.

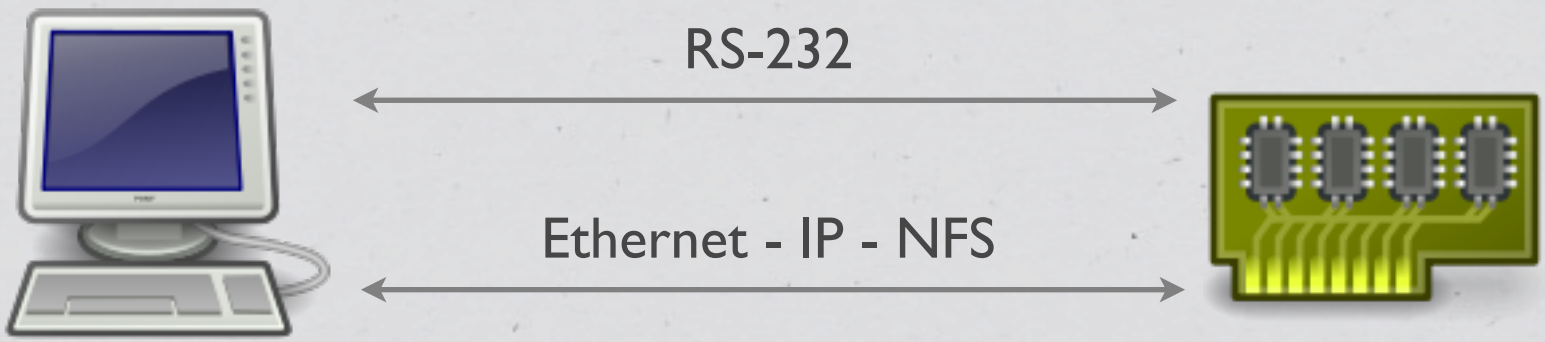
Estendere il supporto ad una board è semplice: basta specificare i kernel e le patch necessarie ed allegare il .config del kernel .

Distribuzioni disponibili

Sono disponibili oltre una trentina di distribuzioni, ovvero di combinazioni note di librerie e software noti per funzionare correttamente assieme.

La più famosa tra le distribuzioni è angstrom.

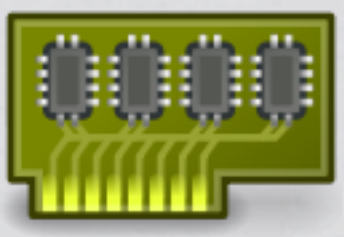
Come organizzare lo sviluppo



Durante tutta la fase di prototipizzazione non è conveniente salvare il rootfs nella memoria flash del dispositivo. E' più conveniente:

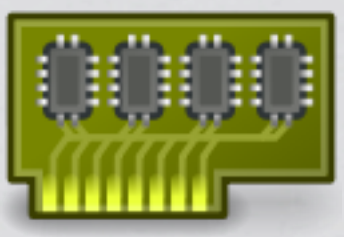
* collegarsi in seriale per controllare bootloader e console;
* usare NFS per montare il rootfs da remoto.

Come organizzare lo sviluppo



Come organizzare lo sviluppo

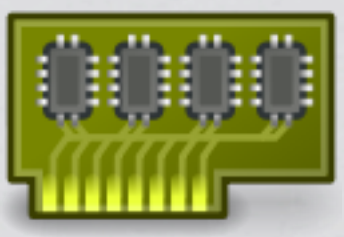
Avvio del terminale seriale



Come organizzare lo sviluppo

Avvio del terminale seriale

Prompt bootloader

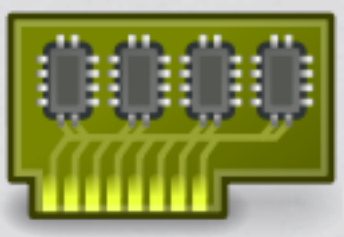


Come organizzare lo sviluppo

Avvio del terminale seriale

Prompt bootloader

Download kernel



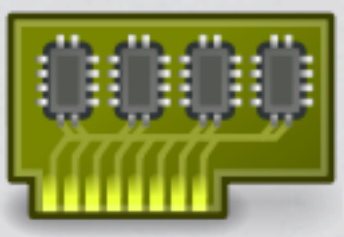
Come organizzare lo sviluppo

Avvio del terminale seriale

Prompt bootloader

Download kernel

Parametri di boot



Come organizzare lo sviluppo

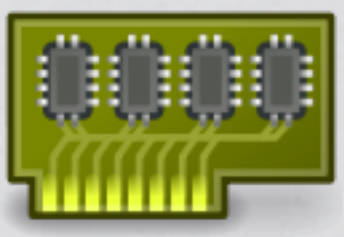
Avvio del terminale seriale

Prompt bootloader

Download kernel

Parametri di boot

Mount rootfs remoto



Come organizzare lo sviluppo

Avvio del terminale seriale

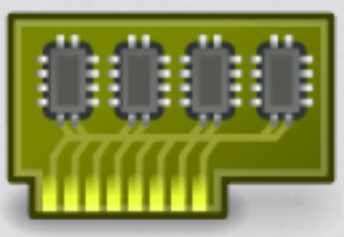
Prompt bootloader

Download kernel

Parametri di boot

Mount rootfs remoto

Prompt della console



Scaricare OpenEmbedded

Il primo passo per sviluppare un sistema con OpenEmbedded è scaricare il download del framework.

Per fare questo è necessario:

* scaricare bitbake;
* scaricare il repository di OpenEmbedded.

Scaricare OpenEmbedded

Non è necessario installare bitbake nel sistema, basta scaricarlo dal suo repository in una cartella locale, perché è scritto in python

dsp@pc:~ $ mkdir stuff

dsp@pc:~ $ cd stuff

dsp@pc:stuff $ svn co svn://svn.berlios.de/bitbake/branches/bitbake-1.8/ bitbake

Scaricare OpenEmbedded

OpenEmbedded è scaricabile in due modi: monotone o wget:

dsp@pc:stuff $ mtn --db=OE.mtn pull monotone.openembedded.org org.openembedded.dev

E' possibile scaricare una copia del repository OE.mtn con wget

dsp@pc:stuff $ wget http://openembedded.org/snapshots/OE.mtn.bz2

e successivamente usare monotone solo per sincronizzare le differenze, dopo aver usare bzip2 per decomprimere il file.

Scaricare OpenEmbedded

Una volta scaricato il repository, si deve estrarre con il seguente comando:

dsp@pc:stuff $ mtn --db=OE.mtn co -b org.openembedded.dev

e tutto il framework di OpenEmbedded sarà disponibile nella directory org.openembedded.dev

Directory layout

stuff
- bitbake
- org.openembedded.dev
  - _MTN
  - classes
  - conf
  - contrib
  - file
  - packages
  - site

Configurazione del framwork

dsp@pc:stuff $ cd org.openembedded.dev

dsp@pc:org.openembedded.dev $ cd conf

dsp@pc:conf $ mv local.conf.sample local.conf

I minimi parametri da impostare sono:

BBFILES := "/home/dsp/stuff/org.openembedded.dev/packages/*/*.bb"

MACHINE = "davinci-sffsdr"
DISTRO = "angstrom-2008.1"

Variabili d'ambiente

Per terminare la configurazione di Openembedded ed iniziare la compilazione è necessario impostare delle variabili d'ambiente:

```
export PATH=$PATH:${HOME}/stuff/bitbake/bin/

export BBPATH=${HOME}/stuff/build:${HOME}/stuff/
org.openembedded.dev

export BBFILES=${HOME}/stuff/org.openembedded.dev/
packages/*/*.bb
```

Compilazione

Per ricompilare un sistema minimale completo:

dsp@pc:org.openembedded.dev $ bitbake minimal-image

Questo comando compilerà tutto il framework, includendo tutti i tool necessari per scaricare il software, patcharlo, i compilare, la libreria C, i programmi necessari ad avere un sistema minimale e tutte le dipende.

Può richiedere diverse ore.

Il risultato della compilazione

Al termine del processo di compilazione si saranno ottenuti:

* il kernel
* il rootfs

L'immagine del kernel dovrà essere messa in una directory accessibile dal servizio TFTP mentre il rootfs, disponibile nella directory /home/dsp/stuff/org.openembedded.dev/tmp/rootfs dovrà essere copiato nella directory del servizio NFS.

Prima di effettuare il boot

Ci sono due *common issues* che possono far fallire il boot del sistema generato:

* non effettuare un chown -R 0.0 del rootfs copiato nella directory del server NFS
* configuare erroneamente eth0 nel rootfs nel file /etc/network/interfaces (lo stesso di Debian). In particolar modo, se si vuole usare il client dhcp del kernel, è necessario commentare la riga con scritto auto eth0 .

Boot della board

Il bootloader più usato nei sistemi embedded è U-Boot. In generale per avviare una scheda possono essere necessari diversi comandi:

```
U-Boot > setenv ipaddr 192.168.0.100
U-Boot > setenv serverip 192.168.0.1
U-Boot > setenv ethaddr 00:d0:cc:05:00:96
U-Boot > tftp 0x80700000 uImage-davinci-sffsdr.bin
U-Boot > set bootargs console=ttyS0,115200n8 root=/dev/nfs
rw noinitrd ip=dhcp nfsroot=192.168.0.1:/srv/nfs/lyrtech

U-Boot > boot 0x80700000
```

Boot senza ethernet

Nell'esempio precedente il boot è stato fatto utilizzando il protocollo TFTP. Tuttavia è possibile trasferire più lentamente il kernel anche direttamente con la seriale:

U-Boot > loadb 0x80700000

premere CTRL-\ C

(/home/dsp/) C-Kermit$>$ send /srv/tftp/uImage-davinci-sffsdr.bin

Kermit è più robusto di minicom, quindi preferibile.

Termine del boot

Al termine della procedura di boot, /dev/console verrà aperta sulla seriale e sarà possibile interagire con il sistema.

Essendo il sistema di networking completo, sarà possibile interagire via rete con il sistema, per esempio utilizzando dropbear per gli accessi remoti sicuri.

Sviluppare un programma

Bitbake possiede tutti gli automatismi per la gestione dei software gestiti tramite autotools.

La strategia più semplice è quindi basare lo sviluppo della propria applicazione direttamente con gli autotools.

Per integrare l'applicazione nel sistema embedded, basterà creare una ricetta per bitbake affinché compili il software e modificare la minimal-image perché include anche il software sviluppato.

Memorizzare il sistema

L'ultimo passo nello sviluppo di un sistema embedded prevede il salvataggio del rootfs servito via NFS nella scheda.

OpenEmbedded provvede già in automatico a generare una immagine jffs2 per poter memorizzare il filesystem in una memoria flash.

Questo è una soluzione molto comune nei dispositivi, tuttavia alcune volte è necessario non montare la memoria flash. In questo caso si può utilizzare initrd o initramfs.

JFFS2

E’ un filesystem dedicato alle memore flash di tipo NAND e NOR.

Integra la possibilità di comprimere i dati nel filesystem.

Prevede di utilizzare “in modo circolare” i settori della flash, per allungarne il tempo di vita.

Purtroppo tutti i blocchi di JFFS2 devono essere controllati durante il mount. Questo rende il boot lento in memorie ampie. Per questo motivo LogFS e UbiFS sono stati proposti come nuovi filesystem.

Uso dei ramdisk

Esistono due tipi di ramdisk per linux: initrd e initramfs.

* initrd prevede che tutto il filesystem venga incluso nell'immagine del kernel durante la sua compilazione. Quindi tutto il sistema consisterà di un solo binario

* initramfs prevede un file per il kernel ed un'immagine del filesystem. Questo permette di modificare il rootfs senza ogni volta dover ricompilare il kernel.