I emissione	**Emessa:**	**Verificata:**	**Approvata:**
	ing Gabriele Filosofi	<Caio>	<Sempronio>

1 Indice

2 Premessa

L'IP Video Phone (IPvphone) di InteractiveMedia e' un dispositivo innovativo "combinato" costituito da un videotelefono IP (Modulo Digitale) e da un telefono POTS tradizionale (Modulo Analogico). Il dispositivo può essere utilizzato come un normale telefono fisso (POTS), anche in assenza di alimentazione locale, ma se fornito di alimentazione e di connettività IP permette di effettuare video chiamate IP ad alta qualità.

L'hardware (descritto nel documento IPvphone_Specifiche di Progetto - Part I), realizzato da ADFL Consulting, è basato sul TMS320DM644x "DaVinci Digital Media Processor", dispositivo dual core SoC di Texas Instruments, con core ARM9 e DSP TMS320C64x+. Il videotelefono è dotato di tastiera, monitor orientabile TFT a colori con risoluzione QVGA, videocamera con risoluzione VGA, ingressi e uscite audio e video, porta seriale di servizio, interfaccia USB 2.0 (host), slot per scheda SIM e slot SD/MMC/MS per lo storage.

Il core ARM9, il core DSP e le periferiche dedicate del DM644x formano una combinazione ideale per applicazioni multimediali incentrate sul video. Sul lato DSP esistono molte terze parti, oltre a TI stessa, in grado di fornire software di compressione audio/video (H.264, G.711, G.729, MPEG4, ecc). Inoltre il videotelefono implementa il self-view PIP (picture-in-picture), la cancellazione dell'eco, il VAD (voice activity detection), la generazione dei toni DTMF, un buffer anti-jitter adattativo, e tutti i protocolli di segnalazione e controllo necessari alla instaurazione di una videochiamata su IP (protocollo SIP o H323). E' prevista la configurazione e l'aggiornamento del FW in locale (UART) o da remoto (TFTP), la possibilità di integrare un WEB Server, di generare report di statistiche e Event Logging, di gestire SMS in entrata e in uscita, di gestire videoconferenze. Il prodotto è certificato FCC, CE.

Il presente documento contiene il Manuale di Installazione del SW di base per il Modulo Digitale e cioè il bootloader, il kernel Linux, il file system, i tools di cross compilazione, le utilities di programmazione della flash. Questi componenti, basati sul SW DVEVM/DVSDK distribuito da Texas Instruments e da Montavista insieme alla EVM del DM644x (DVEVM), sono stati modificati opportunamente da ADFL Consulting per funzionare sul IPvPhone.

Il modo di procedere è il seguente: si installa la versione originale di un componente, si prova la ricompilazione, sulla directory dei sorgenti si sovrascrive la directory dei sorgenti modificati per la IPvPhone, si prova la ricompilazione. Per ogni sviluppo futuro del LSP si consiglia di utilizzare il tool di gestione automatica delle patches **quilt**, incluso nel software Montavista.

Il presente documento è incluso in un DVD, insieme a tutti gli altri file necessari per l'installazione.

3 Contenuto del DVD

La seguente figura mostra l'albero principale delle cartelle contenute nel DVD

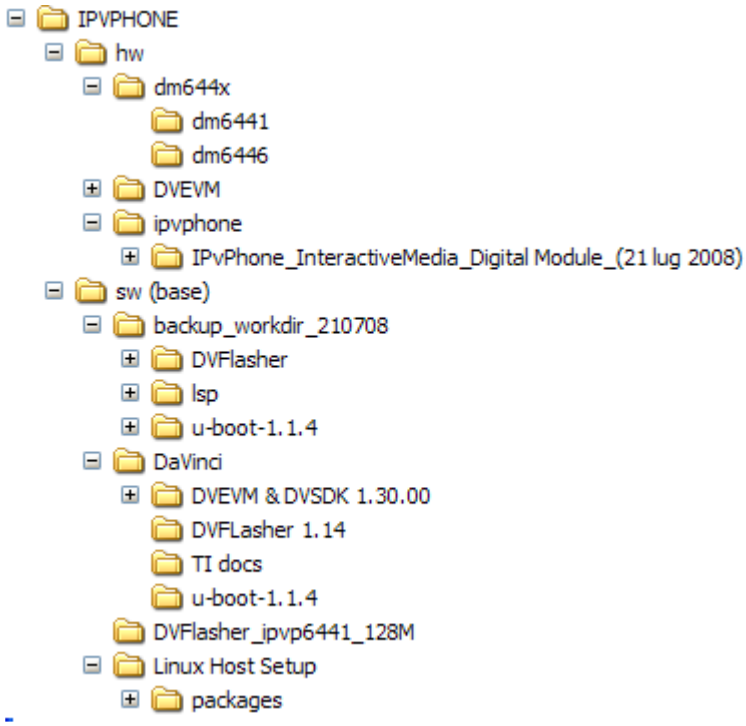


Figura 1. DVD – cartelle principali

La cartella

***D:/IPVPHONE*/hw/**

serve come utile riferimento al progetto hardware IPvPhone.

La cartella

***D:/IPVPHONE*/sw (base)/**

contiene tutti i file oggetto del presente documento.

4 Definizione degli ambienti di sviluppo

Nel seguito sono illustrati i principali tools SW e le opzioni esistenti per la definizione dell'ambiente di sviluppo. In rosso sono evidenziate le opzioni scelte dal sottoscritto, tra quelle disponibili.

4.1 Configurazione Host

- PC workstation con Windows OS[1]
- Partizione Linux su HD oppure VM[2] con distribuzione Linux
  - Red Hat Enterprise Linux v3 o v4, commerciale (http://redhat.com)
  - Fedora Core 9, free (http://fedoraproject.org). (kernel 2.6.25)
- Terminale seriale:
  - Opzione host Windows: HyperTerminal
  - Opzione host Linux: C-Kermit, Minicom, ecc.

4.2 Opzioni Target Kernel Linux

- MontaVista™ Pro Linux 4.0 per DaVinci (LSP1.20 con kernel 2.6.10 e drivers per DM644x)
- MV open source: git kernel 2.6[3] (kernel 2.6.25)
  http://source.mvista.com/git/?p=linux-davinci-2.6.git;a=summary

4.3 Target Bootloader

- TI U-Boot (www.ti.com/corp/docs/landing/davinci/faqs.html#14/)
- open source: U-Boot git (http://www.denx.de/cgi-bin/gitweb.cgi?p=u-boot.git;a=summary)

4.4 Tools e Ambienti di Sviluppo

4.4.1 Sviluppo applicazioni ARM

http://wiki.davincidsp.com/index.php?title=Linux_Toolchain

---

[1] Necessario per CCS

[2] La Virtual Machine (VM) è un software che gira su un macchina fisica (host) e che crea una macchina virtuale (guest). Nel nostro caso sull'host gira Windows e sul guest gira una distribuzione Linux. Un esempio di VM è VirtualBox della InnoTek. Il vantaggio della VM rispetto alla partizione HD è che non occorre riavviare il PC.

[3] L'open source git è stato sviluppato a partire da Montavista 2.6.10, ma non è supportato e ancora non ha alcuni driver (p.es. per il VPSS)

Per la cross-compilazione ARM del bootloader, del kernel, del file system, dei driver e dell'applicazione *user space* occorre la toolchain GNU per ARM926EJS (binutils, gcc, gdb, glibc). Ci sono le seguenti possibilità

- MV toolchain del Linux Professional Edition 4.0.1 per DaVinci (compreso nel DVSDK)
- CodeSourcery - Optimized free Linux GCC toolchain for ARM
- ELDK (Embedded Linux Development Kit) della DENX Software Engineering http://www.denx.de/en/News/WebHome
- RidgeRun free SDK per Davinci DM355 e DM6446 (utilizza uClibc, non glibc)

Per il debug lato ARM ci sono tre possibilità

- GNU DDD (Data Display Debugger)
- gdb (http://wiki.davincidsp.com/index.php?title=Debugging_remotely_on_DaVinci_using_gdb)
- MontaVista DevRocket IDE basato su Eclipse (compreso nel DVSDK)

4.4.2 Sviluppo applicazioni DSP

Per lo sviluppo delle applicazioni DSP occorre

- CCS Platinum v3.3 Development Tools Bundled with Annual S/W Subscription 3600 $
- XDS560 Blackhawk USB High-Performance JTAG Emulator 3000 $

Per il debug leggere http://wiki.davincidsp.com/index.php?title=Debugging_the_DSP_side_of_a_CE_application_on_DaVinci_using_CCS

4.5 DVEVM & DVSDK

- DVSDK (Digital Video Software Development Kit) con MontaVista™ Pro Linux 7000 $

Nota: Per scaricare i componenti SW aggiornati del DVEVM & DVSDK (versione dimostrativa) andare sul sito TI protetto

https://www-a.ti.com/extranet/cm/product/dvevmsw/dspswext/general/homepage.shtml

E' richiesta registrazione.

4.5.1 TI eXpressDSP Configuration Kit

Lo *eXpress-DSP Configuration Kit* (compreso nel DVSDK) serve per combinare i codec desiderati in un "package" custom

- Include i codecs VISA di TI
- Supporta i codec proprietari rispondenti allo standard eXpressDSP Digital Media (xDM), il Codec Engine, il kernel DSP/BIOS, e il DSP/BIOS Link IPC (interprocess communication technology)
- Permette di integrare moduli software proprietari combinandoli in un singolo file eseguibile

4.5.2 TI TMS320DM644x SoC Analyzer

Tool basato su eXpressDSP Data Visualization Technology

- Acquisisce e visualizza graficamente l'attività del sistema, la distribuzione del carico, ecc.
- Visualizza i tasks del DSP e dell'ARM contemporaneamente
- Identifica i punti deboli dell'elaborazione

4.5.3 TI Codec Engine (CE)

Il Codec Engine di TI è un set di API che automatizza l'allocazione e la chiamata degli algoritmi rispondenti al eXpressDSP Algorithm Interface Standard (xDAIS). In particolare, usa le VISA API per i codec dei vari media (xDAIS-DM, o xDM) e le Core Engine API negli altri casi.
Il Codec Engine è fornito con il TMS320DM644x DVEVM (*Digital Video Evaluation Module*).
Il framework può lavorare in sistemi ARM-only, DSP-only o ARM+DSP. Dato che il DM644x è un sistema ARM+DSP il Codec Engine lavorerà utilizzando il DSP/BIOS Link.
Il Codec Engine non gestisce I/O. E' l'applicazione che preleva lo stream di dati dall'I/O e lo passa al Codec Engine tramite buffer in memoria, e viceversa.
Per ottenere il CE e altri componenti necessari allo sviluppo di applicazioni DSP consultare

Target Content: https://www-a.ti.com/downloads/sds_support/targetcontent/index.html/
CodeGen tools: https://www-a.ti.com/downloads/sds_support/CodeGenerationTools.htm/
Info: http://tiexpressdsp.com/wiki/index.php?title=Main_Page/
http://wiki.davincidsp.com/index.php?title=Main_Page/

4.5.4 Framework Components API Reference

E' formato da

- ti.sdo.fc.dskt2 - xDAIS utility library for instantiating xDAIS algorithms, and managing their memory resources
- ti.sdo.fc.dman3 - DMA Manager library for managing DMA hardware resources for xDAIS algorithms. This includes the IDMA3 interface, which xDAIS algorithms must implement to enable DMAN3 to manage their DMA resources
- ti.sdo.fc.acpy3 - High-Performance Functional DMA Interface
- ti.sdo.fc.utils - Utilities which frameworks can built upon, including DBC and DBG

5 Preparazione dell'host

In questo e nei paragrafi seguenti è data una lista di azioni, da applicare nell'ordine, per arrivare ad avere un ambiente di sviluppo minimale Linux per la scheda target IPvPhone (Modulo Digitale).

5.1.1 Installazione Fedora Core 9

Il sottoscritto utilizza per l'Host una distribuzione Fedora Core 9 su PC portatile HP. Linux è montato su una partizione dell'HD. Lo sviluppatore che non ritiene di necessitare di questi componenti può saltare direttamente il paragrafo.

1) Scaricare dal web il CD/DVD della release (http://fedoraproject.org/) e procedere all'installazione di Fedora Core 9 (ver. 2.6.25-14.fc9)

2) Check partizioni HD

```
host $ /sbin/fdisk -lu /dev/sda | grep NTFS

Device     Boot  Start      End        Blocks     Id    System
/dev/sda1  *     63         71682029   35840983+  7     HPFS/NTFS
/dev/sda2        71682030   205889039  67103505   7     HPFS/NTFS
/dev/sda3        205889040  488392064  141251512+ f     W95 Ext'd (LBA)
/dev/sda5        268398018  288880829  10241406   b     W95 FAT32
/dev/sda6        288880893  488392064  99755586   7     HPFS/NTFS
/dev/sda7        205889166  206306729  208782     83    Linux
/dev/sda8        206306793  268397954  31045581   8e    Linux LVM
```

Figura 2. Partizionamento sul disco host (esempio)

5.1.2 Aggiornamento della distribuzione e installazione di packages ausiliari

Le seguenti operazioni permettono di aggiornare tutti i pacchetti installati della distribuzione Fedora e di installarne di altri, alcuni opzionali. Lo sviluppatore che non ritiene di necessitare di questi componenti può saltare direttamente il paragrafo.
Si consiglia di seguire il documento

D:\IPVPHONE\sw\Linux Host Setup\Fedora Core 9 installation guide.pdf

3) Disponendo di un accesso alla rete internet, per l'aggiornamento di tutti i pacchetti installati eseguire

```
host $ sudo yum update
```

Fedora viene aggiornato alla versione più recente (p.es. 2.6.25.9-73.fc9) che apparirà per prima nella lista delle opzioni del bootloader grub al boot successivo.
Se si vuole modificare il menu di grub, ad esempio per cambiare la distribuzione che viene lanciata per default, basta modificare il file `/boot/grub/menu.lst`

4) Attivazione SW repository LIVNA for Fedora Core 9 (Base)

```
host $ sudo rpm -ivh http://rpm.livna.org/livna-release-9.rpm
host $ sudo rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-livna
```

per version successive alla 9 usare

```
host $ sudo rpm -ivh http://rpm.livna.org/livna-release.rpm
```

5) Installazione opzionale MPlayer

```
host $ yum install mplayer mplayer-gui gecko-mediaplayer mencoder
host $ sed -i 's/flip-hebrew/#flip-hebrew/' /etc/mplayer/mplayer.conf
```

6) Installazione opzionale Eclipse

```
host $ sudo yum –y install eclipse-cdt.i386
host $ sudo yum –y install eclipse-cdt-sdk.i386
```

7) Compatibilità gcc con versioni più vecchie

```
host $ sudo yum -y install compat-libstdc++-33 compat-libstdc++-296
```

8) Installazione opzionale gcc 3.4 (per applicazioni che non supportano gcc 4.1)

```
host $ sudo yum –y install compat-gcc-34 compat-gcc-34-c++
```

9) Installazione opzionale SAMBA

```
host $ sudo yum –y install samba samba-client
```

10) Installazione opzionale Minicom

```
host $ sudo yum –y install minicom
```

11) Installazione opzionale Xemacs

```
host $ sudo yum –y install xemacs.i386
```

12) Installazione opzionale network monitor

```
host $ sudo yum –y install iptraf.i386
```

13) Installazione opzionale GNU ddd debugger (3.3.11)

```
host $ sudo yum –y install ddd.i386
```

14) Installazione opzionale ffmpeg

```
host $ sudo yum –y install ffmpeg.i386
```

15) Installazione opzionale live555-tools

```
host $ sudo yum –y install live555-tools.i386
```

Nota: per cercare un pacchetto di cui si conosce solo una parola chiave usare

```
host $ sudo yum search parolachiave
```

6 Installazione SW DaVinci

L'installazione dei componenti SW DaVinci consiste nei seguenti passi. Tutti i file richiesti sono contenuti nel supporto DVD allegato alla documentazione. La cartella in questione è

D:\IPVPHONE\sw\DaVinci\DVEVM&DVSDK 1.30.00

La seguente figura mostra l'albero delle sotto-cartelle contenute nel DVD

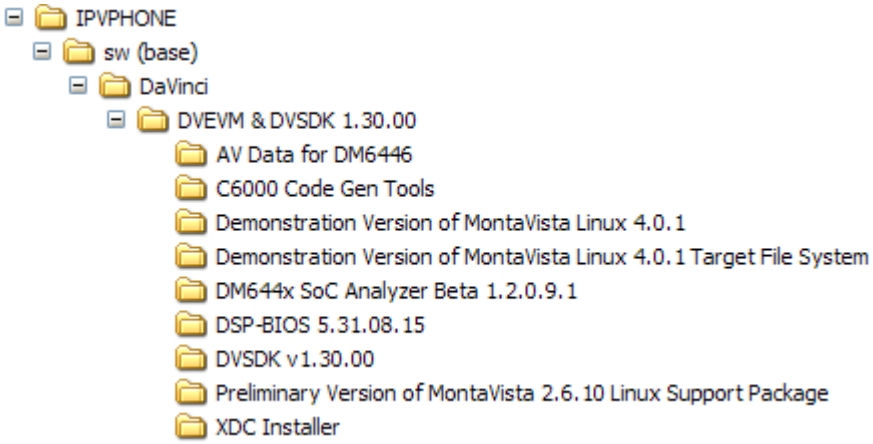


Figura 3. DVD – sottocartelle sw DVEVM&DVSDK

6.1 Installazione componenti DVEVM

Il pacchetto contiene:

- Codec Engine - Provides ARM-side API's for using codecs running on the DSP
- Codec Servers - pre-built DSP executables containing high performance A/V codecs.
- Demo Apps - Encode, Decode, and Encode / Decode A/V streams.
- Demonstration Version of Montavista Linux Pro v4.0.1 - MVL Tools, MVL Target File System, and LSP 1.20

6.1.1 Installare i componenti MV Linux

In ciò che segue **#_##_##_##** rappresenta la versione (es. 1_20_00_10), mentre "user" rappresenta il nome del profilo utente scelto dallo sviluppatore.

16) in una directory /tmp/dnld copiare i seguenti files, presenti nelle varie sottocartelle

- mvl_4_0_1_demo_sys_setuplinux.bin
- mvl_4_0_1_demo_target_setuplinux.bin
- mvl_4_0_1_demo_lsp_setuplinux_#_#_#_#.bin
- dvsdk_setuplinux_#_#_#_#.bin
- data.tar.gz
- xdc_setuplinux_#_#_#_#.bin
- bios_setuplinux_#_#_#_#.bin
- TI-C6x-CGT-v#.#.#.#.bin

17) Log in come root
18) da /tmp/dnld eseguire

```
host $ ./mvl_4_0_1_demo_sys_setuplinux.bin
host $ ./mvl_4_0_1_demo_target_setuplinux.bin
host $ ./mvl_4_0_1_demo_lsp_setuplinux.bin
```

Nota: Scegliere "/opt/dm644x/mv_pro_4.0.1" come directory di installazione
Nota: Se è necessario disinstallare qualcuno di questi componenti è sufficiente rimuovere la cartella.

19) da /opt/dm644x/mv_pro_4.0.1 eseguire

```
host $ tar zxf mvltools4.0.1-no-target.tar.gz
host $ tar zxf mvl4.0.1-target_path.tar.gz
host $ tar zxf DaVinciLSP#.#.#.tar.gz
```

ciò crea la directory /opt/dm644x/mv_pro_4.0.1/montavista/.

6.1.2 Configurare l'ambiente di sviluppo ARM

20) Log in con un user account (d'ora in avanti "user")
21) Aggiungere nel file `/home/user/.bashrc`

```
PATH="/opt/dm644x/mv_pro_4.0.1/montavista/pro/devkit/arm/v5t_le/bin:/opt
//dm644x/mv_pro_4.0.1/montavista/pro/bin:/opt/dm644x/mv_pro_4.0.1/montav
ista/common/bin:$PATH"
export PATH
```

6.1.3 Installare i componenti TI DVEVM/DVSDK

22) da /tmp/dnld eseguire come user

```
host $ ./dvsdk_setuplinux_#_##_##_##.bin  (installare in /home/user)
host $ ./xdc_setuplinux_#_#_#_#.bin (installare in /home/user/dvsdk_#_##)
host $ ./dsp_bios_setuplinux_#_#_#_#.bin (installare in /home/user/dvsdk_#_##)
host $ ./TI-C6x-CGT-v#.#.#.#.bin (installare in /home/user/dvsdk_#_##)
```

Se è necessario disinstallare qualcuno di questi componenti è sufficiente rimuovere la cartella.
P.es.

```
host $ rm –rf /home/user/dvsdk_#_##
```

6.1.4 Installare i files demo A/V

23) da /home/user/dvsdk**_#_##**

```
host $ cp /tmp/data.tar.gz data.tar.gz
host $ tar zxf data.tar.gz
```

24) Come root, in /tmp/dnld eliminare tutti gli eseguibili già installati.

7 Target Bootloader, Kernel e File System

7.1 Predisposizione dell'hardware e del collegamento seriale

Predisporre/collegare le seguenti parti hardware:

- Scheda target IPvPhone (Modulo Digitale)
- Alimentatore esterno 9-12V dalla rete alla scheda target (J11)
- Cavo ethernet di tipo Cross dall'host alla scheda target (J5)
- Cavo seriale RS232 con plug RJ-45 dall'host alla scheda target (J15)

7.1.1 Installare terminale Minicom

Minicom è il terminale seriale testuale scelto dal sottoscritto. Altre soluzioni sono possibili.
Il package è in

D:\IPVPHONE\sw\Linux Host Setup\packages\minicom

25) Come root, copiare in \tmp\dnld e unzippare i sorgenti
(http://alioth.debian.org/projects/minicom/)

```
host $ tar zxf minicom-2.3.tar.gz
host $ cd minicom-2.3
```

26) Ricompilare e installare

```
host $ ./configure
host $ make
host $ make install
```

27) Lanciare utilità di configurazione

```
host $ minicom -s
```

28) Settare Serial Port Setup

```
Serial device: /dev/ttyUSB0
Bps/Par/Bits: 115200 8N1
Flow Control: No
```

e salvare configurazioni

```
Save setup as dfl
```

Nota: lo sviluppatore deve impostare l'opzione "Serial device" in base al nodo di IO che intende utilizzare sul suo PC. Il sottoscritto ha dovuto usare un convertitore esterno USB/RS232.

29) Pulire

```
host $ cd /tmp
host $ rm –rf minicom-2.3
```

Note:

Per lanciare minicom

```
host $ minicom
```

Per poter usare minicom anche come user, sempre come root eseguire

```
host $ chmod 777 –R /var/lock
```

In caso non funzionasse, controllare sempre che il comando

```
host $ which minicom
```

da

```
/usr/local/bin/minicom
```

Infatti è possible che invece di questo venga intercettato

```
/opt/dm644x/mv_pro_4.0.1/montavista/common/bin/minicom
```

Una soluzione è rinominare il file

```
host $ mv /opt/dm644x/mv_pro_4.0.1/montavista/common/bin/minicom
/opt/dm644x/mv_pro_4.0.1/montavista/common/bin/_minicom
```

7.2 DVFlasher

DVFlasher è l'utility scelta per la programmazione della flash on-board. I sorgenti originali di TI sono in

D:\IPVPHONE\sw\DaVinci\DVFlasher 1.14

7.2.1 Installare e lanciare DVFlasher

30) creare la cartella /home/user/workdir/DVFlasher e copiarci dentro

```
dvflasher_#_#.tar.gz
```

31) da /home/user/workdir/DVFlasher

```
host $ tar zxf dvflasher_#_#.tar.gz
```

32) per lanciare l'applicazione sotto Linux deve essere installato il Mono Framework (http://www.mono-project.com/Downloads). Da /home/user/workdir/DVFlasher/exe

```
host $ mono DVFlasher_#_#.exe <options> <binfile>
```

Se necessario ridirezionare su una porta diversa da /dev/ttyS1 con

```
host $ mono dvflasher_#_#.exe -p "/dev/ttyUSB0" <options> <binfile>
```

Per non dover stare sempre a specificare mono e la porta si può creare un alias in /home/user/.bashrc

```
alias dvflasher='mono dvflasher_#_#.exe -p "/dev/ttyUSB0"'
```

per lanciare l'applicazione sotto Windows deve essere installato il .NET Framework 2.0 (http://www.microsoft.com/). Da home/user/workdir/DVFlasher/exe

```
> dvflasher_#_#.exe <options> <binfile>
```

Se necessario ridirezionare su una porta diversa da COM1

```
> dvflasher_#_#.exe -p "COM3" <options> <binfile>
```

Esempi:

Per cancellare la flash

```
> dvflasher_#_#.exe -p "COM3" -enor
```

Per scrivere in flash l'immagine del bootloader

```
> dvflasher_#_#.exe -p "COM3" -r u-boot.bin
```

7.2.2 Re-build di DVFlasher e UBL per IPvPhone

Per permettere a DVFlasher di funzionare sul target specifico IPvPhone è stato necessario apportare alcune modifche. Il DVD allegato alla documentazione contiene il progetto DVFlasher già modificato e ricompilato. La cartella in questione è

D:\IPVPHONE\sw\backup_workdir_210708\DVFlasher

33) Copiare la cartella suddetta in `/home/user/workdir/`

Nota: Se richiesto eseguire il Merge e il Replace dei files

34) Ricompilare per IPvPhone. I makefile del progetto sono stati modificati in modo tale da prevedere opzioni di compilazione adatte all'applicazione IPvPhone, in funzione del SoC (DM6441, DM6446) e del taglio della RAM (64MB,128MB,256MB).
In questo momento le opzioni previste sono le seguenti

- DM6441_128M

- DM6446_128M
- DM6446_256M

Se il target monta DM6441 e 128 MB di RAM allora

```
host $ make DM6441_128M
```

7.3 TFTP

Tramite il protocollo TFTP è possibile trasferire files dall'host al target. Il source package è in

D:\IPVPHONE\sw\Linux Host Setup\packages\tftp-server

7.3.1 Installare il server TFTP

35) per controllare se il TFTP è già installato

```
host $ rpm -q tftp-server
```

36) per installare il TFTP copiare in /tmp il pacchetto rpm (http://rpmfind.net/linux/rpm2html/search.php?query=tftp-server) e, da /tmp, eseguire

```
host $ rpm -ivh tftp-server-#.##-#.rpm
```

Se si dispone di yum e di un accesso alla rete usare

```
host $ sudo yum install tftp-server
```

37) Eseguire

```
host $ /sbin/chkconfig --list
```

Si dovrebbe vedere il seguente output (ultima riga):

```
xinetd based services:
...
tftp: off
...
```

38) per attivare il TFTP

```
host $ /sbin/chkconfig tftp on
```

La cartella di default per i file TFTP è /var/lib/tftpboot, ma si può cambiare in /etc/xinetd.d/tftp. Predisporre la cartella in /tftpboot.

39) per evitare che SELinux (policy di sicurezza su Fedora 9) blocchi le transazioni TFTP disabilitarlo scrivendo nel file `/etc/selinux/config` la riga

SELINUX=disabled

Altre distribuzioni possono avere altri sistemi di policy.

7.4 Network File System

Grazie al NFS il processore sul target monta il suo file system su directory fisicamente presenti e modificabili sull'host.

7.4.1 Esportare il NFS

Per creare ed esportare il file system dal server NFS

40) Log in come user

```
host $ cd /home/user
host $ mkdir -p workdir/filesys
host $ cd workdir/filesys
```

41) passare in root mode

```
host $ su root
```

42) creare una copia del target file system con permessi di scrittura nell'area <user>

```
host $ cp -a
/opt/dm644x/mv_pro_4.0.1/montavista/pro/devkit/arm/v5t_le/target/* .
host $ chown -R user opt
```

dove user è il nome del account

43) per esportare il file system creare ed editare /etc/exports

```
host $ nano /etc/exports
```

e aggiungere la riga

```
/home/user/workdir/filesys *(rw,no_root_squash,no_all_squash,sync)
```

44) sempre come root

```
host $ /usr/sbin/exportfs -a
host $ /sbin/service nfs restart
```

In seguito, per ri-esportare tutte le directory usare

```
host $ /usr/sbin/exportfs -rav
```

mentre per verificare che lo stato del NFS usare

```
host $ /etc/init.d/nfs status
```

45) Per comodità, in `/home/user/` si può creare il file **NFS_start** contenente

```
# .bashrc
```

```
sudo /usr/sbin/exportfs -a
sudo /sbin/service nfs restart
```

e il file **NFS_stop** contenente

```
# .bashrc
sudo /sbin/service nfs stop
```

e poi quando serve li si esegue con bash.

7.4.2 Testare il NFS

46) vedere l'ind. IP dell'host usando

```
host $ /sbin/ifconfig
```

Sia `192.168.0.78`

47) connettere al target e aprire un terminale RS-232 (p.es. Minicom). Accendere il target e abortire la sequenza di boot automatica premendo un tasto nella finestra di comando u-boot. Impostare le seguenti variabili di ambiente

```
target # setenv ipaddr 192.168.0.244
target # setenv serverip 192.168.0.78
target # setenv nfshost 192.168.0.78
target # setenv bootaddr 0x2060000
target # setenv rootpath /home/user/workdir/filesys
target # setenv bootdir boot
target # setenv bootfile uImage

target # setenv bootcmd 'cp.b $(bootaddr) 0x80700000 0x140000; bootm
$(bootaddr)

target # setenv bootargs console=ttyS0,115200n8 noinitrd rw ip=$(ipaddr)
root=/dev/nfs nfsroot=$(nfshost):$(rootpath),nolock mem=120M
```

Nota: se si intende usare l'assegnazione automatica dell'indirizzo IP, in bootargs usare `ip=dhcp`

48) salvare

```
target # saveenv
```

49) controllare che tutto sia a posto

```
target # printenv
```

50) eseguire il boot usando il NFS

```
target # boot
```

7.4.3 Caricare ed eseguire un semplice programma tramite NFS

51) Log in con un user account sul NFS

```
host $ mkdir /home/user/workdir/filesys/opt/hello
```

```
host $ cd /home/user/workdir/filesys/opt/hello
```

52) Creare un file hello.c tipo

```
#include <stdio.h>
int main() {
printf("Buongiorno DaVinci!\n");
return 0;
}
```

53) compilare

```
host $ arm_v5t_le-gcc hello.c -o hello
```

54) dalla target window di linux (o da telnet)

```
target $ cd /opt/hello
target $ ./hello
```

55) Si dovrebbe leggere “Buongiorno DaVinci!”

7.5 Bootloader

Il bootloader inizializza l’HW del target al power-up e dirige il caricamento del sistema operativo.

7.5.1 Re-build di u-boot

I sorgenti originali di TI (ver. 1.1.4) sono in

D:\IPVPHONE\sw\DaVinci\u-boot-1.1.4

56) Log in come user
57) copiare la cartella dei sorgenti u-boot-1.1.4 in /home/user/workdir. Quindi

```
host $ cd /home/user/workdir/u-boot-1.1.4
```

58) ricompilare per la DVEVM

```
host $ make distclean
host $ make dvevm_config
host $ make
```

7.5.2 Re-build di u-boot per IPvPhone

Per permettere a u-boot di funzionare sul target specifico IPvPhone è stato necessario apportare alcune modifche. Il DVD allegato alla documentazione contiene il progetto u-boot già modificato e ricompilato. La cartella in questione è

D:\IPVPHONE\sw\backup_workdir_210708\u-boot-1.1.4

La seguente figura mostra l’albero delle sub-directory contenute nel DVD

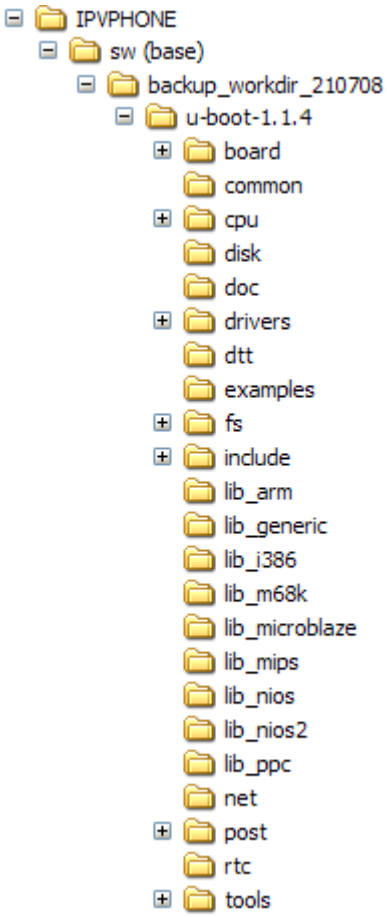


Figura 4. DVD – sottocartelle u-boot

59) Copiare la cartella suddetta in `/home/user/workdir/`

Nota: Se richiesto eseguire il Merge e il Replace dei files

60) Ricompilare per IPvPhone

```
host $ make distclean
host $ make ipvp_config
host $ make
```

7.5.3 Flash download di u-boot

il file binario u-boot.bin, generato in `/home/user/workdir/u-boot-1.1.4` può essere caricato in flash tramite l'applicazione DVFlasher_1_14.

61) Sul target posizionare lo switch SW1-4 su ON (boot da UART)
62) Premere il pulsante RESET
63) Sul terminale seriale si deve ricevere "… BOOTME BOOTME …"
64) P.es. dopo aver copiato u-boot.bin in /DVFlasher/exe, da command window

```
> DVFlasher_1_14.exe -r u-boot.bin
```

Il bootloader verrà scritto in flash a partire dall'indirizzo 0x02000000.

```
-----------------------------------------------------
   TI DVFlasher Host Program for DM644x
   (C) 2007, Texas Instruments, Inc.
-----------------------------------------------------

Platform is Windows.
Restoring NOR flash with u-boot.bin.

Attempting to connect to device COM1...
Press any key to end this program at any time.


Waiting for DVEVM...
BOOTME commmand received. Returning ACK and header...
ACK command sent. Waiting for BEGIN command...
        IPVP644x:         BEGIN
BEGIN commmand received. Sending CRC table...
CRC table sent.  Waiting for DONE...
DONE received.  Sending the UART UBL file...
DONE received.  UART UBL file was accepted.
UART UBL Transmitted successfully.

        IPVP644x:        0x12345679 0x1234567A 0x1234567B
        IPVP644x:        0xE59FF014 0xE59FF014 0xE59FF014
        IPVP644x:       TI UBL Version: 1.14, Flash type: NOR
        IPVP644x:       Booting PSP Boot Loader
        IPVP644x:       PSPBootMode = UART

Waiting for UBL on DVEVM...
UBL's BOOTPSP commmand received. Returning CMD and command...
CMD value sent.
SENDAPP received. Returning ACK and header for application data...
ACK command sent. Waiting for BEGIN command...
UBL's BEGIN commmand received. Sending the application code...
Application code sent.  Waiting for DONE...
DONE received.  All bytes of application code received...
DONE received.  Application S-record decoded correctly.
        IPVP644x:       NOR Initialization:
        IPVP644x:               Command Set: Intel
        IPVP644x:               Manufacturer: INTEL
        IPVP644x:               Size (in bytes): 0x02000000
        IPVP644x:       Erasing the NOR Flash
        IPVP644x:       Erased through 0x02008000
        IPVP644x:       Erased through 0x02010000
        IPVP644x:       Erased through 0x02018000
        IPVP644x:       Erased through 0x02020000
        IPVP644x:       Erase Completed
        IPVP644x:       Writing the NOR Flash
        IPVP644x:       NOR Write OK through 0x02000800
        IPVP644x:       NOR Write OK through 0x02001000
        IPVP644x:       NOR Write OK through 0x02001800
        IPVP644x:       NOR Write OK through 0x02002000
        IPVP644x:       NOR Write OK through 0x02002800
        IPVP644x:       NOR Write OK through 0x02003000
        IPVP644x:       NOR Write OK through 0x02003800
        IPVP644x:       NOR Write OK through 0x02004000
```

```
        IPVP644x:       NOR Write OK through 0x02004800
        IPVP644x:       NOR Write OK through 0x02005000
        IPVP644x:       NOR Write OK through 0x02005800
        IPVP644x:       NOR Write OK through 0x02006000
        IPVP644x:       NOR Write OK through 0x02006800
        IPVP644x:       NOR Write OK through 0x02007000
        IPVP644x:       NOR Write OK through 0x02007800
        IPVP644x:       NOR Write OK through 0x02008000
        IPVP644x:       NOR Write OK through 0x02008800
        IPVP644x:       NOR Write OK through 0x02009000
        IPVP644x:       NOR Write OK through 0x02009800
        IPVP644x:       NOR Write OK through 0x0200A000
        IPVP644x:       NOR Write OK through 0x0200A800
        IPVP644x:       NOR Write OK through 0x0200B000
        IPVP644x:       NOR Write OK through 0x0200B800
        IPVP644x:       NOR Write OK through 0x0200C000
        IPVP644x:       NOR Write OK through 0x0200C800
        IPVP644x:       NOR Write OK through 0x0200D000
        IPVP644x:       NOR Write OK through 0x0200D800
        IPVP644x:       NOR Write OK through 0x0200E000
        IPVP644x:       NOR Write OK through 0x0200E800
        IPVP644x:       NOR Write OK through 0x0200F000
        IPVP644x:       NOR Write OK through 0x0200F800
        IPVP644x:       NOR Write OK through 0x02010000
        IPVP644x:       NOR Write OK through 0x02010800
        IPVP644x:       NOR Write OK through 0x02011000
        IPVP644x:       NOR Write OK through 0x02011800
        IPVP644x:       NOR Write OK through 0x02012000
        IPVP644x:       NOR Write OK through 0x02012800
        IPVP644x:       NOR Write OK through 0x02013000
        IPVP644x:       NOR Write OK through 0x02013800
        IPVP644x:       NOR Write OK through 0x02014000
        IPVP644x:       NOR Write OK through 0x02014800
        IPVP644x:       NOR Write OK through 0x02015000
        IPVP644x:       NOR Write OK through 0x02015800
        IPVP644x:       NOR Write OK through 0x02016000
        IPVP644x:       NOR Write OK through 0x02016800
        IPVP644x:       NOR Write OK through 0x02017000
        IPVP644x:       NOR Write OK through 0x02017800
        IPVP644x:       NOR Write OK through 0x02018000
        IPVP644x:       NOR Write OK through 0x02018800
        IPVP644x:       NOR Write OK through 0x02019000
        IPVP644x:       NOR Write OK through 0x02019800
        IPVP644x:       NOR Write OK through 0x0201A000
        IPVP644x:       NOR Write OK through 0x0201A800
        IPVP644x:       NOR Write OK through 0x0201B000
        IPVP644x:       NOR Write OK through 0x0201B800
        IPVP644x:       NOR Write OK through 0x0201C000
        IPVP644x:       NOR Write OK through 0x0201C800
        IPVP644x:       NOR Write OK through 0x0201D000
        IPVP644x:       NOR Write OK through 0x0201D7B8
        IPVP644x:          DONE

Operation completed successfully.

C:\ipvphone\DVFlasher_ipvp6441_128M>pause
Premere un tasto per continuare . . .
```

In questo caso il bootloader occupa 0x1D7B8 bytes. Quindi un settore della Flash (128 KB) è stato sufficiente.
Nota: Per eseguire questa operazione Minicom può non funzionare correttamente. In tal caso cambiare terminale oppure utilizzare Windows, come fatto nell'esempio.
La cartella

D:\IPVPHONE\sw\DVFlasher_ipvp6441_128M

contenente già tutti i file necessari, compreso un batchfile per lanciare direttamente l'utility sotto Windows (dnld_u-boot_com1.bat).

7.5.4 Shell di u-boot

65) Sul target posizionare lo switch SW1-4 su OFF (boot da Flash)
66) Premere il pulsante RESET
67) Sul terminale seriale (19200,N,8) compare il bunner ADFL Consulting s.r.l.

Quando si interrompe il countdown che precede il caricamento del sistema operativo si entra nella shell di U-Boot

```
target #
```

Nel caso della scheda IPvPhone il prompt è

```
IPvPhone #
```

Invocando l'help viene visualizzato il menu delle operazioni che è possibile fare.

Go	start application at address 'addr'
Run	run commands in an environment variable
bootm	boot application image from memory
bootp	boot image via network using BootP/TFTP protocol
tftpboot	boot image via network using TFTP protocol and env variables "ipaddr" and "serverip"
rarpboot	boot image via network using RARP/TFTP protocol
loads	load S Record file over serial line
loadb	load binary file over serial line
Md	memory display
mm	memory modify with auto-incrementing
Nm	memory modify at constant address
mw	memory write/fill
Cp	memory copy
cmp	memory compare
crc32	checksum calculation
imd	i2c memory display
imm	i2c memory modify with auto-incrementing
inm	i2c memory modify at constant address
imw	i2c memory write/fill
icrc32	i2c checksum calculation
iprobe	probe to discover valid I2C chip addresses
iloop	infinite loop on address range
isdram	print SDRAM configuration information
sspi	SPI utility commands

base	print or set address offset
printenv	print environment variables
setenv	set environment variables
saveenv	save environment variables to persistent storage
protect	enable or disable FLASH write protection
erase	erase FLASH memory
flinfo	print FLASH memory information
fsinfo	print FLASH filesystem information
bdinfo	print Board Info structure
iminfo	print header information for application image
coninfo	print console devices and informations
loop	infinite loop on address range
loopw	infinite write loop on address range
mtest	simple RAM test
icache	enable or disable instruction cache
dcache	enable or disable data cache
reset	Perform RESET of the CPU
echo	echo args to console
version	print monitor version
help	print online help

Tabella 1. u-boot - comandi

Il bootloader passa delle variabili al kernel. Sono le variabili di ambiente bootargs.
Per visualizzare queste variabili, dalla shell usare il comando

```
target # printenv
```

Per modificare queste variabili usare il comando

```
target # setenv bootargs "…"
```

Esempio. Per fare in modo che il kernel al boot aggiorni automaticamente la variabile `videostd` in base alla posizione del jumper sulla EVM

```
target # setenv bootargs "mem=120M console=ttyS0,115200n8 root=/dev/hda1
rw noinitrd ip=dhcp"

target # setenv bootcmd "setenv setboot setenv bootargs \$(bootargs)
video=dm64xxfb:output=\$(videostd);run setboot;bootm 0x2060000"
```

Per salvare le modifiche in Flash

```
target # saveenv
```

Le variabili verranno salvate nel primo settore vuoto dopo l'area occupata dal bootloader.

7.5.5 Trasferimento di file da shell u-boot tramite TFTP

Qualora in Flash sia già presente U-Boot e occorra aggiornarlo si può utilizzare il TFTP dalla directory del server tftp (/tftpboot) mediante i seguenti comandi:

1. Trasferimento via rete del u-boot in DDR2

```
target # tftp 0x80700000 u-boot.bin
```

Nota: questa operazione visualizza la dimensione del file `<size in hex>` e lo assegna alla variabile di ambiente `filesize`

2. Sprotezione settori della Flash

```
target # protect off 0x02000000 +<size in hex>
```

oppure

```
target # protect off 0x02000000 +0x$(filesize)
```

3. Cancellazione settori della Flash

```
target # erase 0x02000000 +0x$(filesize)
```

4. Scrittura in Flash

```
target # cp.b 0x80700000 0x02000000 0x$(filesize)
```

Analogamente, per salvare in Flash l'immagine del kernel:

```
target # tftp 0x80700000 uImage
```

Si dovrebbe vedere

```
TFTP from server 192.168.0.78; our IP address is 192.168.0.244
Filename 'uImage'.
Load address: 0x80700000
Loading:
*#################################################################################
#################################################################################
#################################################################################
##############################################
done
Bytes transferred = 1334800 (145e10 hex)
```

Quindi, per scrivere in Flash

```
target # protect off 0x02060000 +0x145e10
target # erase 0x02060000 +0x145e10
target # cp.b 0x80700000 0x02060000 0x145e10
```

Nota: per sapere `<size in hex>` si può anche leggere l'header dell'immagine presente in ram, usando

```
target # imi 0x80700000
```

e aggiungere a questa 64 (la dimensione dell'header)

7.5.6 Variabili di ambiente settate in HW

La scheda DVEVM prevede la possibilità di inizializzare una variabile di ambiente direttamente in HW, in base allo stato di uno switch. Questa variabile di ambiente è chiamata `videostd` e comunica al kernel la modalità di funzionamento dell'interfaccia video analogica (PAL/NTSC). Questo concetto è stato mantenuto ed esteso nella scheda IPvPhone. Oltre alla variabile `videostd`, anche le variabili `usersw1` e `usersw2` possono essere definite dallo stato dello switch multiplo SW1.

SW1 switch	Nome variabile	Stato ON – valore	Stato OFF – valore
1	usersw1	On	off
2	usersw2	On	off
3	videostd	ntsc	pal

Tabella 2. Variabili di boot impostate mediante HW Switch SW1

Le variabili ausiliarie `usersw1` e `usersw1` possono essere utilizzate dall'applicazione in svariati modi.
Al momento, in base allo stato degli switch, il bootloader precarica le variabili di ambiente secondo il seguente schema

SW1	ON	OFF
1	display COMPOSITO	display LCD
2	kernel e FS in FLASH	TFTP boot e NFS
3	formato video NTSC	formato video PAL
4	boot da UART0	Boot da FLASH

7.6 Kernel

Partiamo dal LSP 1.20 di Montavista contenuto nel pacchetto TI

7.6.1 Re-build del kernel

68) Log in come user
69) copiare il MV LSP (linux 2.6.10 e drivers) in /home/user/workdir

```
host $ cp -R /opt/mv_pro_4.0.1/montavista/pro/devkit/lsp/ti-davinci .
host $ cd ti-davinci
```

70) Per configurare il kernel, cioè selezionare i driver e le features richieste, con i defaults della DVEVM

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- davinci_dm644x_defconfig
```

con opzioni custom

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- menuconfig
```

Il file che invece viene usato dal MAKE per generare l'immagine del kernel e che viene aperto con menuconfig è

```
/home/user/workdir/lsp/ti-davinci/.config
```

71) per ricompilare, sempre come user,

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
```

La ricompilazione genererà un kernel chiamato

Linux-2.6.10_mvl401

L'immagine generata di questo kernel è un file binario compresso u-boot compatible chiamato uImage, e si trova nella directory `/home/user/workdir/ti-davinci/arch/arm/boot`. Ovviamente questa immagine è relativa alla scheda DVEVM e non funzionerà sulla IPvPhone.

7.6.2 Re-build del kernel per IPvPhone

Per permettere al kernel di funzionare sul target specifico IPvPhone è stato necessario apportare alcune modifche al MV LSP. Il DVD allegato alla documentazione contiene il progetto già modificato e ricompilato. La cartella in questione è

D:\IPVPHONE\sw\backup_workdir_210708\lsp

La seguente figura mostra l'albero delle sub-directory contenute nel DVD

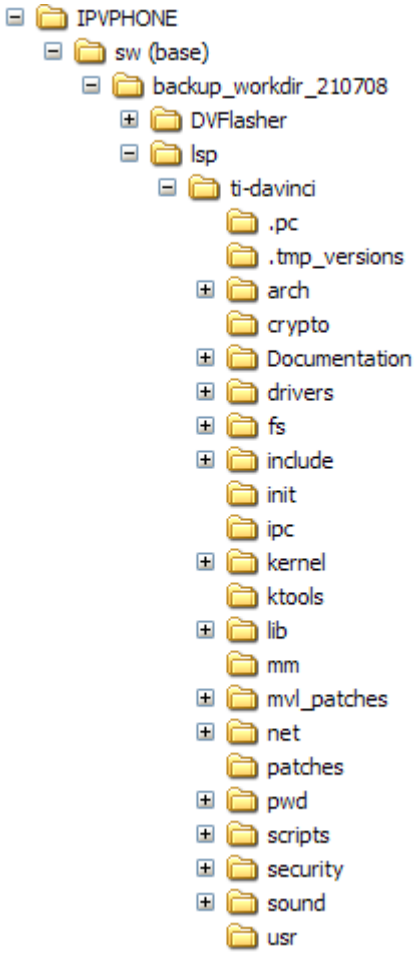


Figura 5. DVD – sottocartelle LSP

72) Copiare la cartella suddetta in `/home/user/workdir/`

Nota: Se richiesto eseguire il Merge e il Replace dei files

La seguente tabella indica i file del LSP che sono stati effettivamente modificati/creati.

File	Azione
localversion-mvl	Modificato
include/asm/mach-types.h	Modificato
include/asm/arch/i2c-client.h	Modificato
include/asm/arch-davinci/i2c-client.h	Modificato
include/config/mach/davinci/ipvp.h	Creato
include/config/ipvp/module.h	Creato
arch/arm/tools/mach-types	Modificato
arch/arm/mach-davinci/Makefile	Modificato
arch/arm/mach-davinci/gpio-ipvp.c	Creato
arch/arm/mach-davinci/gpio-ipvp.h	Creato
arch/arm/mach-davinci/i2c-client.c	Modificato
arch/arm/mach-davinci/board-ipvp.c	Creato
arch/arm/mach-davinci/lateinit-ipvp.c	Creato
arch/arm/mach-davinci/clock.c	Modificato
arch/arm/mach-davinci/leds-ipvp.c	Creato
arch/arm/mach-davinci/Kconfig	Modificato
arch/arm/configs/ipvp_defconfig	Creato
arch/arm/Kconfig	Modificato
drivers/net/Kconfig	Modificato
drivers/char/Kconfig	Modificato
drivers/usb/musb/davinci.c	Modificato

Tabella 3. MV LSP – file modificati/creati per generare il BSP IPvPhone

73) Ricompilare per IPvPhone

```
host $ make clean
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- ipvp_defconfig
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
```

Nota: i file di configurazione di default sono in /home/user/workdir/lsp/ti-davinci/arch/arm/config/
Il file di configurazione di default per la scheda IPvPhone è

```
/home/user/workdir/lsp/ti-davinci/arch/arm/config/ipvp_defconfig
```

La ricompilazione genererà un kernel chiamato

Linux-2.6.10_mvl401-davinci_ipvp

L'immagine generata è un file binario compresso u-boot compatible chiamato uImage, e si trova nella directory `/home/user/workdir/ti-davinci/arch/arm/boot`.

74) se si é configurato il kernel con almeno un modulo, compilare i moduli

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- modules
```

e trasferirli nel filesystem

```
host $ sudo make ARCH=arm CROSS_COMPILE=arm_v5t_le-
INSTALL_MOD_PATH=/home/user/workdir/filesys modules_install
```

75) copiare uImage nella cartella /tftpboot, in modo da renderlo disponibile a U-Boot per il download via TFTP

```
host $ cp /home/user/workdir/lsp/ti_davinci/arch/arm/boot/Image
/tftpboot/uImage
host $ chmod a+r /tftpboot/uImage
```

7.7 Metodi di boot

Per default la DVEVM esegue il boot da Flash con target file system su HDD (on board).
La IPvPhone non ha hard disk. Ci sono i seguenti altri modi possibili che devono essere testati nell'ordine

1. TFTP boot with NFS filesystem
2. Flash boot with NFS filesystem
3. Flash boot with Flash filesystem

7.7.1 Boot via TFTP usando il NFS

Questa modalità è utile quando si deve modificare e provare il kernel.

76) Assicurarsi di aver trasferito nella cartella tftp l'immagine del kernel che deve essere provata, a partire dalla cartella di creazione. Per esempio

```
host $ cd /home/user/workdir/lsp/ti-davinci
host $ cp arch/arm/boot/uImage /tftpboot/uImage
```

I passi seguenti dovranno essere eseguiti solo una volta. Non serviranno nei successivi boot.

77) accendere il target e abortire la sequenza di boot automatica premendo un tasto nella finestra di comando u-boot. Impostare le seguenti variabili di ambiente

```
target # setenv bootcmd 'tftp 0x80700000; bootm 0x80700000'
target # setenv serverip <TFTP SERVER IP ADDRESS>
target # setenv bootfile uImage
target # setenv rootpath /home/user/workdir/filesys
target # setenv nfshost <NFS HOST IP ADDRESS>
target # setenv ipaddr 192.168.0.244
target # setenv bootargs console=ttyS0,115200n8 noinitrd rw ip=dhcp
root=/dev/nfs nfsroot=$(nfshost):$(rootpath),nolock mem=120M
```

Possiamo ipotizzare
`<TFTP SERVER IP ADDRESS> = <NFS HOST IP ADDRESS> = 192.168.0.243`
Eventualmente si possono aggiungere le seguenti altre variabili relative all'interfaccia video

```
target # setenv bootargs \$(bootargs) video=davincifb:vid0=720x576x16,
2500K:vid1=720x576x16,2500K:osd0=720x576x16,2025K
davinci_enc_mngr.ch0_output=COMPOSITE
```

```
davinci_enc_mngr.ch0_mode=$(videostd)
```

Nota: i comandi setenv bootargs .. vanno editati su una singola linea
Nota: se si intende usare l'assegnazione automatica dell'indirizzo IP, in bootargs usare `ip=dhcp`, altrimenti usare `ip=$(ipaddr)`

78) salvare

```
target # saveenv
```

79) eseguire il boot usando il NFS

```
target # boot
```

80) Segue la sequenza di boot

```
*******************************************
*        ADFL Consulting S.r.l. 2008      *
*             DaVinci IPvPhone            *
*******************************************
U-Boot 1.1.4 (Jul 18 2008 - 17:57:23)

U-Boot code: 81080000 -> 8109D7B8  BSS: -> 810A6260
RAM Configuration:
   Bank #0: 80000000 128 MB
INTEL Flash Configuration: 32 MB
In:    serial
Out:   serial
Err:   serial
ARM Clock : 256MHz
DDR Clock : 162MHz
SW1.1=ON : usersw1 = on
SW1.2=OFF : usersw2 = off
SW1.3=OFF : videostd = pal
Hit any key to stop autoboot: 10..0
TFTP from server 192.168.0.78; our IP address is 192.168.0.244
Filename 'uImage'.
Load address: 0x80700000
Loading:
*#################################################################
#################################################################
#################################################################
##############################
done
Bytes transferred = 1252860 (131dfc hex)
## Booting image at 80700000 ...
   Image Name:   Linux-2.6.10_mvl401-davinci_ipvp
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1252796 Bytes =  1.2 MB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
OK

Starting kernel ...
```

```
Uncompressing
Linux.....................................................................
.............. done, booting thenLinux version 2.6.10_mvl401-
davinci_ipvp (gabriele@localhost.localdomain) (gcc version 3.4.3
(MontaVista 3.4.3-25.0.104.0600975 2006-07-06)) #34 Thu Jul 17 23:20:29
CEST 2008
CPU: ARM926EJ-S [41069265] revision 5 (ARMv5TEJ)
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 4, 32 byte lines, 128 sets
CPU0: D cache: 8192 bytes, associativity 4, 32 byte lines, 64 sets
Machine: DaVinci IPvPhone
Memory policy: ECC disabled, Data cache buffered
Built 1 zonelists
Kernel command line: console=ttyS0,115200n8 noinitrd rw ip=192.168.0.244
root=/dev/nfs nfsroot=192.168.0.78:/home/gabriele/workdir/filesys,nolock
mem=120M
PID hash table entries: 512 (order: 9, 8192 bytes)
Console: colour dummy device 80x30
Dentry cache hash table entries: 16384 (order: 4, 65536 bytes)
Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)
Memory: 120MB = 120MB total
Memory: 118912KB available (2174K code, 426K data, 136K init)
Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
CPU: Testing write buffer coherency: ok
spawn_desched_task(00000000)
desched cpu_callback 3/00000000
ksoftirqd started up.
desched cpu_callback 2/00000000
desched thread 0 started up.
NET: Registered protocol family 16
Registering platform device 'musb_hdrc'. Parent at platform
DaVinci I2C DEBUG: 15:34:51 Jul 17 2008
Registering platform device 'i2c'. Parent at platform
usbcore: registered new driver usbfs
usbcore: registered new driver hub
JFFS2 version 2.2. (NAND) (C) 2001-2003 Red Hat, Inc.
yaffs Jul 17 2008 15:34:32 Installing.
Registering platform device 'davincifb.0'. Parent at platform
Setting Up Clocks for DM420 OSD
Console: switching to colour frame buffer device 90x30
fb0: dm_osd0_fb frame buffer device
fb1: dm_vid0_fb frame buffer device
fb2: dm_osd1_fb frame buffer device
fb3: dm_vid1_fb frame buffer device
Serial: 8250/16550 driver $Revision: 1.90 $ 2 ports, IRQ sharing
disabled
Registering platform device 'serial8250'. Parent at platform
ÿttyS0 at MMIO 0x1c20000 (irq = 40) is a 16550A
io scheduler noop registered
io scheduler anticipatory registered
RAMDISK driver initialized: 1 RAM disks of 32768K size 1024 blocksize
Registering platform device 'ti_davinci_emac'. Parent at platform
TI DaVinci EMAC: MAC address is 44:be:29:c0:01:00
TI DaVinci EMAC Linux version updated 4.0
TI DaVinci EMAC: Installed 1 instances.
netconsole: not configured, aborting
i2c /dev entries driver
```

```
Linux video capture interface: v1.00
Registering platform device 'vpfe.1'. Parent at platform
DaVinci v4l2 capture driver V1.0 loaded
elevator: using anticipatory as default io scheduler
musb_hdrc: version 2.2a/db-0.4.8 [pio] [host] [debug=0]
musb_hdrc: USB Host mode controller at c8060000 using PIO, IRQ 12
musb_hdrc musb_hdrc: new USB bus registered, assigned bus number 1
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 1 port detected
Registering platform device 'davinci-audio.0'. Parent at platform
NET: Registered protocol family 2
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 8192 bind 16384)
NET: Registered protocol family 1
NET: Registered protocol family 17
IP-Config: Guessing netmask 255.255.255.0
IP-Config: Complete:
      device=eth0, addr=192.168.0.244, mask=255.255.255.0,
gw=255.255.255.255,
     host=192.168.0.244, domain=, nis-domain=(none),
     bootserver=255.255.255.255, rootserver=192.168.0.78, rootpath=
Looking up port of RPC 100003/2 on 192.168.0.243
Looking up port of RPC 100005/1 on 192.168.0.243
VFS: Mounted root (nfs filesystem).
Freeing init memory: 136K
INIT: version 2.85 booting
Mounting a tmpfs over /dev...done.
Creating initial device nodes...done.
Activating swap...done.
Remounting root filesystem...done.
Calculating module dependencies
Loading modules:
Checking all file systems: fsck
fsck 1.35 (28-Feb-2004)
Mounting local filesystems: mount nothing was mounted
Cleaning: /tmp /var/lock /var/run done.
Setting up networking (ifupdown) ..
Cleaning: /etc/network/run/ifstate done.
Starting network interfaces: done.
Starting hotplug subsystem:
   pci
   pci      [success]
   usb
   usb      [success]
   isapnp
   isapnp   [success]
   ide
   ide      [success]
   input
   input    [success]
   scsi
   scsi     [success]
done.
Starting portmap daemon: portmap.
done.
Setting pseudo-terminal access permissions...done.
Updating /etc/motd...done.
```

```
INIT: Entering runlevel: 3
Starting NFS common utilities: statd lockd.
Starting internet superserver: inetd.
Starting MontaVista target tools daemon: mvltdmvltd version 2.1
MontaVista Software,Inc.
mvltd[961]: can't send broadcast message: Network is unreachable
mvltd[961]: can't send broadcast message: Network is unreachable
. mvltd[962]: started on port 34577


MontaVista(R) Linux(R) Professional Edition 4.0.1 (0600980)

192.168.0.244 login: root
Last login: Thu Mar 16 14:29:43 1972 on console
Linux 192.168.0.244 2.6.10_mvl401-davinci_ipvp #34 Thu Jul 17 23:20:29
CEST 2008 armv5tejl GNU/Linux

Welcome to MontaVista(R) Linux(R) Professional Edition 4.0.1 (0600980).

root@192.168.0.244:/#
```

7.7.2 Boot da Flash usando il NFS

Questa modalità è utile quando il kernel è stato messo a punto e si devono sviluppare moduli e drivers.

81) accendere il target e abortire la sequenza di boot automatica premendo un tasto nella finestra di comando u-boot. Impostare le seguenti variabili di ambiente

```
target # setenv nfshost 192.168.0.243
target # setenv bootaddr 0x2060000
target # setenv ipaddr 192.168.0.244
target # setenv rootpath /home/user/workdir/filesys
target # setenv bootargs console=ttyS0,115200n8 noinitrd rw ip=dhcp
root=/dev/nfs nfsroot=$(nfshost):$(rootpath),nolock mem=120M
```

Si noti che il boot viene eseguito da flash, quindi il check del crc32 dell'immagine del kernel viene eseguito in flash. Per accelerare il boot è preferibile che questa oparazione venga eseguita in ram. Se si fissa a 0x140000 la dimensione massima di uImage (ad esempio), una soluzione è la seguente

```
target # setenv bootcmd 'cp.b 0x2060000 0x80700000 0x140000; bootm
0x80700000'
```

Nota: se si intende usare l'assegnazione automatica dell'indirizzo IP, in bootargs usare `ip=dhcp`, altrimenti usare `ip=$(ipaddr)`

82) salvare

```
target # saveenv
```

83) reboot

```
target # boot
```

7.7.3 Boot da Flash con target file system in Flash

Questa modalità è quella definitiva.
In questo paragrafo indichiamo la procedura per creare un JFFS2 file system in flash. Quanto segue è stato implementato sulla versione git del kernel (2.6.25).
Diversamente dai file system residenti in flash, ma che vengono copiati in ram (p.es. ramdisk), il JFFS2 resta in flash, in modo tale che le modifiche apportate tra un reboot e l'altro restano permanenti. Per la scrittura dell'immagine in flash utilizziamo i comandi di u-boot, che deve essere già presente in Flash (questo ci permette di non necessitare di un ram file system intermedio tipo NFS).

84) modificare e configurare il kernel secondo le proprie esigenze

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- menuconfig
```

85) assicurarsi che le seguenti opzioni siano abilitate in .config

```
CONFIG_MTD=y
CONFIG_MTD_PARTITIONS=y
CONFIG_MTD_CHAR=y
CONFIG_MTD_BLKDEVS=y
CONFIG_MTD_BLOCK=y
CONFIG_MTD_CFI=y
CONFIG_MTD_GEN_PROBE=y
CONFIG_MTD_MAP_BANK_WIDTH_1=y
CONFIG_MTD_MAP_BANK_WIDTH_2=y
CONFIG_MTD_MAP_BANK_WIDTH_4=y
CONFIG_MTD_CFI_I1=y
CONFIG_MTD_CFI_I2=y
CONFIG_MTD_CFI_AMDSTD=y
CONFIG_MTD_CFI_UTIL=y
CONFIG_MTD_PHYSMAP=y
CONFIG_MTD_PHYSMAP_START=0x2000000
CONFIG_MTD_PHYSMAP_LEN=0x1000000
CONFIG_MTD_PHYSMAP_BANKWIDTH=2

CONFIG_JFFS2_FS=y
CONFIG_JFFS2_FS_DEBUG=0
CONFIG_JFFS2_FS_WRITEBUFFER=y
CONFIG_JFFS2_ZLIB=y
CONFIG_JFFS2_RTIME=y

CONFIG_PROC_FS=y
CONFIG_PROC_SYSCTL =y
CONFIG_SYSFS=y
CONFIG_TMPFS=y
```

86) assicurarsi che le seguenti opzioni siano disabilitate in .config

```
CONFIG_MTD_NAND
CONFIG_MTD_NAND_VERIFY_WRITE
CONFIG_MTD_NAND_IDS
CONFIG_MTD_NAND_DAVINCI
```

```
CONFIG_PROC_SYSFS
```

87) qualora si decida di non lavorare più con NFS e/o ramdisk, disabilitare queste opzioni in .config, per rendere il kernel più leggero
88) compilare

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
```

89) copiare l'immagine prodotta in /tftpboot

```
host $ cp arch/arm/boot/uImage /tftpboot
```

90) compilare i moduli

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- modules
```

91) assicurarsi di avere installato sull'host le utility MTD. In caso contrario eseguire i seguenti step
92) Ottenere i sorgenti del pacchetto liblzo dal sito http://www.oberhumer.com/opensource/lzo/download/
93) Copiare in /home/user/, estrarre e ricompilare

```
host $ tar zxf lzo-#.##.tar.gz
host $ tar rm lzo-#.##.tar.gz
host $ cd lzo-#.##
host $ ./configure
host $ sudo make
host $ sudo make install
```

94) Ottenere i sorgenti git delle utilities MTD dal sito http://git.infradead.org/mtd-utils.git
95) Copiare in /home/user/, estrarre, ricompilare e copiare gli eseguibili in /usr/bin/ (in questo caso a noi serve solo l'utility mkfs.jffs2)

```
host $ tar zxf mtd-utils-487550498f70455f083cdc82b65442596fe7308e.tar.gz
host $ tar rm mtd-utils-487550498f70455f083cdc82b65442596fe7308e.tar.gz
host $ cd mtd-utils
host $ make clean
host $ sudo make
host $ sudo make install
host $ sudo cp mkfs.jffs2 /usr/bin/
```

96) preparare e montare il file system di esempio ramdisk (fornito nel pacchetto DVSDK) su una directory temporanea rootfs

```
host $ cd ~/workdir
host $ cp ~/dvsdk_1_30_00_40/PSP_01_20_00_014/bin/ramdisk.gz .
host $ gunzip ramdisk.gz
host $ mkdir -p rootfs
host $ sudo mount -o loop ramdisk rootfs
```

Nota: E' possible creare un nuovo RAM disk seguendo la procedura data in appendice.

97) installare i moduli in rootfs

```
host $ cd lsp/linux-davinci-2.6.25
```

```
host $ sudo make ARCH=arm CROSS_COMPILE=arm_v5t_le-
INSTALL_MOD_PATH=/home/gabriele/workdir/rootfs/ modules_install
```

98) applicare eventuali altre modifiche a rootfs

```
example:
host $ cp arch/arm/boot/uImage ~/workdir/rootfs/tmp
```

99) creare l'immagine JFFS2 di rootfs usando le utility MTD

```
host $ cd ~/workdir
host $ sudo mkfs.jffs2 -d rootfs -o rootfs.jffs2 -e 0x20000
```

100) smontare rootfs (ed eventualmente cancellarlo)

```
host $ sudo umount rootfs
```

101) copiare l'immagine creata in /tftpboot

```
host $ cp rootfs.jffs2 /tftpboot
```

102) accendere il target e abortire la sequenza di boot automatica premendo un tasto nella finestra di comando u-boot

103) usando i comandi di u-boot trasferire in ram e copiare in flash l'immagine del file system

```
IPvPhone# tftp 0x80700000 rootfs.jffs2
IPvPhone# protect off 0x2260000 +$(filesize)
IPvPhone# erase 0x2260000 +$(filesize)
IPvPhone# cp.b 0x80700000 0x2260000 $(filesize)
```

104) resettare il target e abortire la sequenza di boot automatica premendo un tasto nella finestra di comando u-boot

105) assicurarsi che i parametri di boot siano corretti

```
IPvPhone# printenv
```

Ad esempio, per avere il root fs montato su una partizione in flash dovrà essere

```
bootargs=console=ttyS0,115200n8 noinitrd rw ip=192.168.0.244
root=/dev/mtdblock4 noatime rootfstype=jffs2 mem=120M
```

Per avere il boot del kernel tramite tftp dovrà essere

```
bootcmd=tftp 0x80700000 uImage; bootm 0x80700000
```

Questa opzione non è necessaria se uImage è già presente in Flash nella sua partizione (all'indirizzo 0x2060000). In tal caso si può partire direttamente dalla Flash

```
bootcmd= cp.b 0x2060000 0x80007FC0 0x200000; bootm 0x80007FC0
```

106) reboot

```
IPvPhone# boot
```

107) Al termine della sequnza di boot, eseguire il login come root

```
192.168.0.244 login: root
target $ cd /
```

108) Verificare che le partizioni MTD siano ben definite

```
target $ cat /proc/mtd
 dev:    size   erasesize  name
 mtd0: 00020000 00008000 "bootloader"
 mtd1: 00020000 00020000 "params"
 mtd2: 00020000 00020000 "spare"
 mtd3: 00200000 00020000 "kernel"
 mtd4: 01da0000 00020000 "filesystem"

target $ cat /proc/partitions
 major minor  #blocks  name
   1     0      32768 ram0
  31     0        128 mtdblock0
  31     1        128 mtdblock1
  31     2        128 mtdblock2
  31     3       2048 mtdblock3
  31     4      30336 mtdblock4
```

109) Creare i nodi MTD sui quali poter poi montare le partizioni. Ogni partizione richiede due nodi, uno di tipo c (mtd*n*) e uno di tipo b (mtdblock*n*).

```
target $ mknod /dev/mtd0 c 90 0
target $ mknod /dev/mtd1 c 90 2
target $ mknod /dev/mtd2 c 90 4
target $ mknod /dev/mtd3 c 90 6
target $ mknod /dev/mtd4 c 90 8
target $ mknod /dev/mtdblock0 b 31 0
target $ mknod /dev/mtdblock1 b 31 1
target $ mknod /dev/mtdblock2 b 31 2
target $ mknod /dev/mtdblock3 b 31 3
target $ mknod /dev/mtdblock4 b 31 4
```

110) Ora è possibile operare sulle partizioni della flash. Ad esempio, avendo messo una copia di uImage in /tmp, è possibile aggiornarla nella partizione dedicata usando le utility MTD (già rese disponibili sul target da MV)

```
target $ flash_eraseall -j /dev/mtd3
target $ flashcp -v /tmp/uImage /dev/mtd3
```

La seguente tabella mostra la mappa delle partizioni della memoria Flash del Modulo Digitale. In caso si montasse una Flash da 16 Mbytes la partizione riservata al file system sarebbe diminuita di 16 Mbytes rispetto a quanto riportato in figura.

Sector	Start Address	File	Descrizione	Name
0	0x02000000	u-boot.bin (max 0x20000)	Bootloader	mtd0 / mtdblock0
1	0x02020000	Params (max 0x20000)	Parametri di u-boot	mtd1 / mtdblock1
2	0x02040000	Spare sector (max 0x20000)	Not used	mtd2 / mtdblock2

3	0x02060000	uImage (max 0x200000)	Kernel	mtd3 / mtdblock3
…	….			
18	0x02240000			
19	0x02260000	rootfs.jffs2 (max 0x1DA0000)	File system	mtd4 / mtdblock4
…	…			
255				

Tabella 24. Partizionamento della Flash NOR (size 32MB)

Per quanto riguarda le dimensioni del file system, è possibile sapere l'occupazione analitica di tutte le cartelle principali con il comando

```
target $ cd /
target $ du –h
```

Ad esempio, un file (compresso) rootfs.jffs2 di 8 MB, una volta decompresso in RAM produce il seguente filesystem

```
772.5k     ./bin
452.0k     ./dev
290.5k     ./etc
5.9M       ./lib
0          ./mnt
26.0M      ./opt
0          ./tmp
0          ./sys
291.0k     ./var
2.6M       ./usr
59.0k      ./proc
571.0k     ./sbin
1.0k       ./root

Totale: 36.9M
```

La stessa misura fatta sul NFS dà

```
612K  ./dev
7.3M  ./etc
642M  ./usr
16K   ./root
0     ./tmp
5.4M  ./bin
1016M ./opt
4.0K  ./media
```

```
44M   ./var
0     ./sys
14M   ./lib
4.0K  ./home
4.0K  ./srv
117K  ./proc
6.0M  ./sbin
0     ./mnt
```

7.8 Creare e provare un semplice modulo

Un driver può essere inserito nel kernel in modo *built-in*, oppure può essere un modulo che, una volta copiato nel file system, viene caricato run-time e gira in kernel mode.
Perché un driver sia gestito come modulo, la corrispondente costante di opzione nel file

```
/home/user/workdir/lsp/ti-davinci/.config
```

deve essere “=m”.

111) dopo aver modificato un modulo occorre compilarlo

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le modules
```

112) e trasferirlo nel NFS filesystem

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le-
INSTALL_MOD_PATH=/home/user/workdir/filesys modules_install
```

Questa operazione avrà aggiornato tutti i file presenti nella cartella

```
/home/user/workdir/filesys/lib/modules/2.6.10_mvl401-davinci_ipvp
```

in particolare i moduli driver presenti in

```
/home/user/workdir/filesys/lib/modules/2.6.10_mvl401-
davinci_ipvp/kernel/drivers
```

Dalla shell del target si può quindi caricare il modulo desiderato utilizzando il comando *insmod*. Ad esempio

```
root@192.168.0.244:/# insmod lib/modules/2.6.10_mvl401-
davinci_ipvp/kernel/drivers/mmc/davinci_mmc.ko
```

Lo stesso si sarebbe ottenuto più semplicemente con il comando *modprobe*

```
root@192.168.0.244:/# modprobe -a davinci_mmc
```

Con *lsmod* possiamo verificare che il modulo è stato effettivamente caricato

```
root@192.168.0.244:/# lsmod

Module                  Size  Used by
davinci_mmc            12920  0
```

Il comando *lsmod* legge le informazioni presenti nel file /proc/modules.
Per fare in modo che un modulo venga caricato automaticamente al boot aggiungerlo nel file

```
/etc/modules
```

Il caricamento di alcuni moduli può richiedere il caricamento di altri moduli. Le dipendenze tra moduli sono descritte nel file

```
/lib/modules/2.6.10_mvl401-davinci_ipvp/modules.dep
```

Per creare un nuovo driver, chiamato ad esempio ipvp_core.ko:

a) Creare la cartella `/home/user/workdir/lsp/ti-davinci/drivers/ipvp/`

   con i file `ipvp.c, ipvp.h, Kconfig, Makefile`. Conviene partire da una copia di un driver preesistente.

b) Modificare conseguentemente il file
   `/home/user/workdir/lsp/ti-davinci/arch/arm/Kconfig`

   aggiungendo la riga

   ```
   Source "drivers/ipvp/Kconfig"
   ```

   Andando a eseguire il `make menuconfig` si troverà il nuovo modulo incluso nel menù, e si potrà selezionare la modalità <M>. Questo farà sì che nel file
   `/home/user/workdir/lsp/ti-davinci/.config` venga aggiunta la riga

   ```
   CONFIG_IPVP=m
   ```

c) Compilare

   ```
   host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- modules
   ```

d) Trasferire nel filesystem

   ```
   host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le-
   INSTALL_MOD_PATH=/home/user/workdir/filesys modules_install
   ```

La seguente tabella indica i file creati/modificati nella directory

```
/home/user/workdir/lsp/ti-davinci/
```

drivers/Kconfig	Modificato
drivers/Makefile	Modificato
drivers/ipvp/Kconfig	Creato
drivers/ipvp/Makefile	Creato
drivers/ipvp/ipvp.c	Creato
drivers/ipvp/ipvp.h	Creato

Tabella 4. MV LSP – file modificati/creati per generare un modulo di esempio ipvp

8 Device Drivers

8.1 Driver per il codec audio AIC33

L'interfaccia ASP (Audio Serial Port) del DM644x sul Modulo Digitale è multiplexata su due differenti audio codec, l'AIC33 (U1) e il PCM3008 (oppure AK4550), quest'ultimo presente sul Modulo Analogico PSTN. Gli ingressi audio sono rappresentati da una presa jack per collegamento microfono (J2) e una presa jack per un segnale stereo preamplificato (J1). Due connettori, dello stesso tipo, sono dedicati all'output su cuffia (J4) e verso un amplificatore esterno (J3).
Il driver è stato implementato sul kernel 2.6.25-davinci1 nei file:

```
/sound/soundcore.ko
/sound/oss/davinci-audio-dma-intfc.ko
/sound/oss/davinci-audio.ko
/sound/oss/davinci-audio-aic33.ko
```

e vanno caricati in questo stesso ordine. Il driver del AIC33 è rispondente allo standard OSS (Open Sound System) ver 1.11. L'accesso al driver da user space avviene attraverso i file /dev/dsp (IO) e /dev/mixer (controllo) e le chiamate di sistema open(), close(), read(), write(), ioctl(), select().

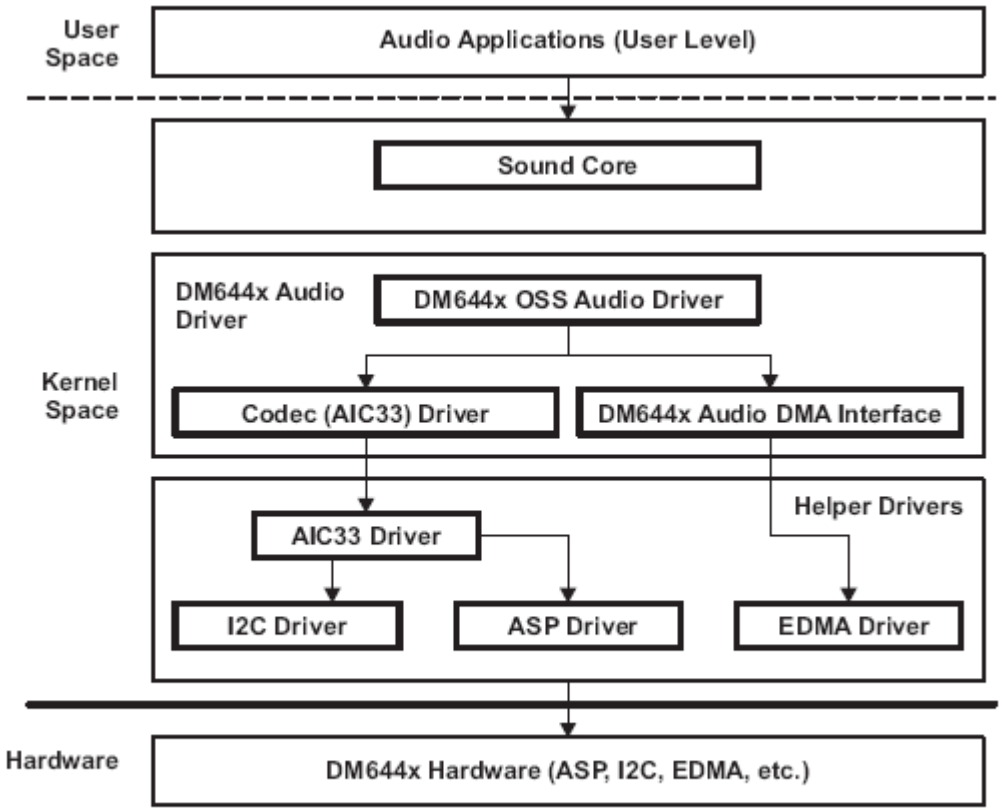


Figure 2-2. DaVinci Audio Driver Architecture

Vediamo in questo paragrafo come installare e testare il driver utilizzando il NFS. Per default, in /arch/arm/configs/davinci_ipvp_defconfig il driver è stato configurato come built-in. Quindi

113) assicurarsi che le seguenti opzioni siano abilitate in .config

```
CONFIG_SENSORS_TLV320AIC33=y
CONFIG_SOUND=y
CONFIG_SOUND_PRIME=y
CONFIG_SOUND_DAVINCI=y
CONFIG_SOUND_DAVINCI_AIC33=y
CONFIG_MONOSTEREO_SAMEJACK=y
```

114) Compilare immagine kernel e moduli, e trasferirli nel root filesystem

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
host $ cp arch/arm/boot/uImage /tftpboot
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- modules
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le-
INSTALL_MOD_PATH=/home/user/workdir/filesys modules_install
```

115) Sul NFS

```
host $ mkdir /home/user/workdir/filesys/opt/audio
host $ cd /home/user/workdir/filesys/opt/audio
```

116) Creare un file audioloop.c tipo

```
/*****************************************************************
* AUDIOLOOP.C
* Gabriele Filosofi 2008 - ADFL Consulting s.r.l.
* Simple program to make an audio loopback on the IPvPhone board
*****************************************************************/
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <linux/soundcard.h>
#include <time.h>

#define SOUND_DEVICE    "/dev/dsp"
#define MIXER_DEVICE    "/dev/mixer"
#define FAILURE -1

/* The number of channels of the audio codec */
#define NUM_CHANNELS           2

/* The sample rate of the audio codec */
#define SAMPLE_RATE            48000

/* The gain (0-100) of the left and right channels */
```

```
#define LEFT_GAIN              64     //1..100
#define RIGHT_GAIN             64     //1..100

time_t starttime;
time_t currenttime;

int oss_open()
{
    int     vol        = LEFT_GAIN | (RIGHT_GAIN << 8);
    int     sampleRate = SAMPLE_RATE;
    int     channels   = NUM_CHANNELS;
    int     format     = AFMT_S16_NE;
    int     soundFd;
    int     mixerFd;
    int     recMask;
    int     recorder;

    printf("Microphone recorder selected\n");
    recorder = SOUND_MASK_MIC;

    /* Select the right capture device and volume*/
    mixerFd = open(MIXER_DEVICE, O_RDONLY);

    if (mixerFd == -1) {
        printf("Failed to open %s\n", MIXER_DEVICE);
        return FAILURE;
    }

    if (ioctl(mixerFd, SOUND_MIXER_READ_RECMASK, &recMask) == -1) {
        printf("Failed to ask mixer for available recorders.\n");
        return FAILURE;
    }

    if ((recMask & recorder) == 0) {
        printf("Recorder not supported\n");
        return FAILURE;
    }

    if (ioctl(mixerFd, SOUND_MIXER_WRITE_RECSRC, &recorder) == -1) {
        printf("Failed to set recorder.\n");
        return FAILURE;
    }

    if (ioctl(mixerFd, SOUND_MIXER_WRITE_IGAIN, &vol) == -1) {
        printf("Failed to set the volume of microphone/line in.\n");
        return FAILURE;
    }

    vol = 0x2020;
    if (ioctl(mixerFd, SOUND_MIXER_READ_VOLUME, &vol) == -1) {
        printf("Failed to set the volume of headphone/line out.\n");
        return FAILURE;
    }
     printf("0x%08x\n",vol);

    close(mixerFd);
```

```
    /* Open the sound device for writing */
    soundFd = open(SOUND_DEVICE, O_RDWR);

    if (soundFd == -1) {
        printf("Failed to open the sound device (%s)\n", SOUND_DEVICE);
        return FAILURE;
    }

    /* Set the sound format (only AFMT_S16_LE supported) */
    if (ioctl(soundFd, SNDCTL_DSP_SETFMT, &format) == -1) {
        printf("Could not set format %d\n", format);
        return FAILURE;
    }

    /* Set the number of channels */
    if (ioctl(soundFd, SNDCTL_DSP_CHANNELS, &channels) == -1) {
        printf("Could not set mixer to %d channels\n", channels);
        return FAILURE;
    }

    /* Set the sample rate */
    if (ioctl(soundFd, SNDCTL_DSP_SPEED, &sampleRate) == -1) {
        printf("Could not set sample rate (%d)\n", sampleRate);
        return FAILURE;
    }
    printf("open sucessfully\n");

    return soundFd;
}

void oss_close(int fd)
{
    printf("before close\n");
    close(fd);
    printf("close sucessfully\n");
}

void audioloop(unsigned char timesecs)
{
    int fd;
    signed short buf[512];
    int numBytes;
    int i=0;
    int done = 0;
    fd = oss_open();

    time(&starttime);

    while (done==0)
    {
        numBytes = read(fd, buf, 512);
        //printf("%i ",numBytes);
        write(fd, buf, numBytes);
        if (timesecs > 0)
        {
            time(&currenttime);
            if ((int)(currenttime - starttime) > timesecs)
```

```
            {
                done=1;
            }
        }
    }

    oss_close(fd);
}

int main()
{
    audioloop (20);
    return 0;
}
```

117) compilare

```
host $ arm_v5t_le-gcc audioloop.c -o audioloop
```

118) Accendere il target avendo inserito le cuffie (J4) e il microfono (J2).
119) Dalla shell linux del target

```
target $ cd /opt/audio/
target $ ./audioloop
```

Nei successivi *20 sec*, parlando nel microfono si dovrebbe ascoltare la propria voce nelle cuffie

120) Lo stesso risultato si otterrebbe con una semplice ridirezione da shell

```
target $ cat /dev/dsp > /dev/dsp
```

121) Per riprodurre in cuffia un file audio .wav (non compresso), ad esempio bach.wav (precedentemente copiato in /opt/audio/)

```
target $ cat bach.wav > /dev/dsp
```

8.2 Driver per PLL & VXCO programmabile CDCE925

A generare i sincronismi principali per l'interfaccia ASP (Audio Serial Port) del DM644x e per l'interfaccia verso il Modulo Analogico PSTN provvedono i componenti TI CDCE925 (Programmable VCXO + 2 PLLs, U29) e ICS67401 (Programmable Clock Divider, U30).
Nel kernel 2.6, tutti i devices e gli adapters I2C appaiono sotto il filesystem sysfs. I devices I2C si trovano in /sys/bus/i2c/devices, ordinati secondo il loro numero di adapter e indirizzo del device.

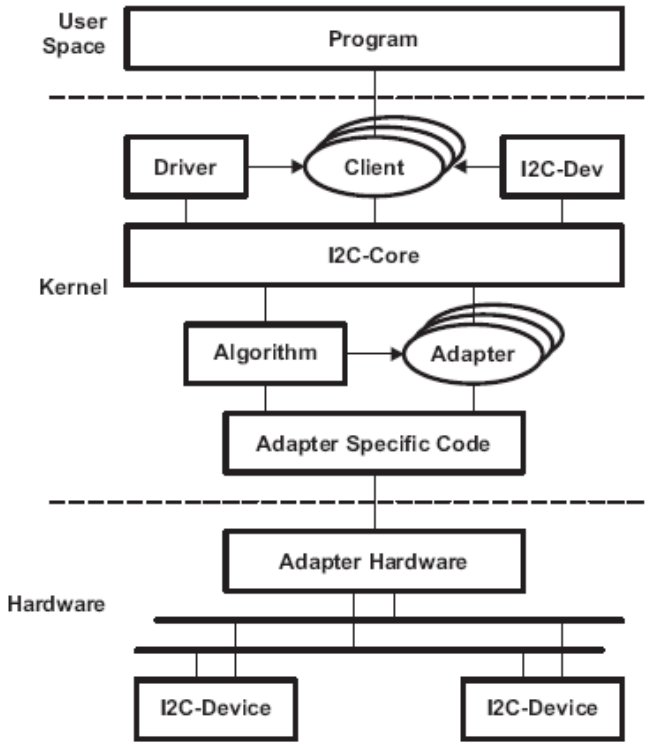


Figure 2-90. Linux Kernel I2C Driver Architecture

Il driver è stato implementato sul kernel 2.6.25-davinci1 nei file:

```
drivers/i2c/chips/cdce925.c
```

Vediamo in questo paragrafo come installare e testare il driver del CDCE925 utilizzando il NFS

122) assicurarsi che le seguenti opzioni siano abilitate in .config

```
CONFIG_SENSORS_CDCE925 =m
```

123) Compilare immagine kernel e moduli, e trasferirli nel root filesystem

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
host $ cp arch/arm/boot/uImage /tftpboot
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- modules
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le-
INSTALL_MOD_PATH=/home/user/workdir/filesys modules_install
```

124) Accendere il target. Dopo il boot, dalla shell linux

```
target $ insmod /lib/modules/2.6.25-rc1-
davinci1/kernel/drivers/i2c/chips/cdce925.ko
```

Il caricamento del modulo cdce925.ko fa sì che sulla scheda venga rilevato e registrato il dispositivo CDCE925 connesso al bus I2C. L'indirizzo di questo device è 0x64, sicchè verrà creata la cartella /sys/bus/i2c/devices/1-0064/.
Leggendo il file /sys/bus/i2c/devices/1-0064/name avremo

```
target $ cat /sys/bus/i2c/devices/1-0064/name
cdce925
```

Le funzioni di I/O esportate da tutti i driver I2C in user space sono rappresentate da semplici file ascii presenti nella directory del device. Ciascun file contiene un solo valore in virgola fissa (attributi). Il cdce925 ha due attributi in scrittura (clkin, pstnfs) e uno in lettura (chipid). I comandi standard della shell permettono di accedere a questi file, sia in lettura con

```
target $ cat /sys/attribute_file
```

che in scrittura con

```
target $ echo xyz > /sys/attribute_file
```

Per leggere e modificare (p.es. a 32000) la frequenza del sincronismo PSTN_WCLOCK

```
target $ cat /sys/bus/i2c/devices/1-0064/pstnfs
48000
target $ echo 32000 > /sys/bus/i2c/devices/1-0064/pstnfs
target $ cat /sys/bus/i2c/devices/1-0064/pstnfs
32000
```

Le frequenze permesse sono 32000, 44100 e 48000.
Per leggere e modificare la selezione del clock in ingresso al PLL esterno cdce925, p.es. da modo XTAL (valore 0) a modo VXCO (valore 1),

```
target $ cat /sys/bus/i2c/devices/1-0064/clkin
0
target $ echo 1 > /sys/bus/i2c/devices/1-0064/clkin
target $ cat /sys/bus/i2c/devices/1-0064/clkin
1
```

8.3 Driver per CMOS Camera Sensor OV7670

L'acquisizione delle immagini per la funzione videotelefono è demandata alla video-camera CMOS OMNIVISION OV07670-VL2A, posizionata subito sopra il display. La programmazione del sensore avviene per tramite del bus I2C..
Nel kernel 2.6, tutti i devices e gli adapters I2C appaiono sotto il filesystem sysfs. I devices I2C si trovano in /sys/bus/i2c/devices, ordinati secondo il loro numero di adapter e indirizzo del device.
Il driver è stato implementato sul kernel 2.6.25-davinci1 nei file:

```
drivers/media/video/ov7670.c
```

Vediamo in questo paragrafo come installare e testare il driver utilizzando il NFS

125) assicurarsi che le seguenti opzioni siano abilitate in .config

```
CONFIG_VIDEO_OV7670 =m
```

126) Compilare immagine kernel e moduli, e trasferirli nel root filesystem

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
host $ cp arch/arm/boot/uImage /tftpboot
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- modules
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le-
INSTALL_MOD_PATH=/home/user/workdir/filesys modules_install
```

127) Accendere il target. Dopo il boot, dalla shell linux

```
target $ insmod /lib/modules/2.6.25-rc1-
davinci1/kernel/drivers/media/video/ov7670.ko
```

Il caricamento del modulo ov7670.ko fa sì che sulla scheda venga rilevato e registrato il dispositivo connesso al bus I2C. L'indirizzo di questo device è 0x21, sicchè verrà creata la cartella /sys/bus/i2c/devices/1-0021/.
Leggendo il file /sys/bus/i2c/devices/1-0021/name avremo

```
target $ cat /sys/bus/i2c/devices/1-0021/name
ov7670
```

Questo driver ha un attributo in scrittura (write) e uno in lettura (chipid).
L'attributo in lettura fornisce gli ID interni del dispositivo

```
target $ cat /sys/bus/i2c/devices/1-0021/chipid
PID=0x76 VER=0x73
```

L'attributo in scrittura è stato modificato in modo da effettuare delle operazioni base sul sensore, a seconda del valore che si inserisce.
Ad esempio, per eseguire il reset del dispositivo

```
target $ echo 0 > /sys/bus/i2c/devices/1-0021/write
```

Per inizializzare i registri interni del dispositivo

```
target $ echo 1 > /sys/bus/i2c/devices/1-0021/write
```

Per stampare a monitor il contenuto dei registri interni

```
target $ echo 2 > /sys/bus/i2c/devices/1-0021/write
```

8.4 Driver per SIM Card Reader TDA8023

Il Modulo Digitale è dotato di interfaccia SIM card (Subscriber Identification Module). La scheda monta un controllore SMART card NXP TDA8023 (U43), rispondente alle specifiche GSM11.11, GSM11.12 (Global System for Mobile communication), ISO 7816-3 e EMV 2000 (payment system). Anche la programmazione del TDA8023 avviene tramite bus I2C.
Il driver è stato implementato sul kernel 2.6.25-davinci1 nei file:

```
drivers/i2c/chips/tda8023.c
```

Vediamo in questo paragrafo come installare e testare il driver utilizzando il NFS

128) assicurarsi che le seguenti opzioni siano abilitate in .config

```
CONFIG_TDA8023 =m
```

129) Compilare immagine kernel e moduli, e trasferirli nel root filesystem

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
host $ cp arch/arm/boot/uImage /tftpboot
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- modules
```

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le-
INSTALL_MOD_PATH=/home/user/workdir/filesys modules_install
```

130) Accendere il target. Dopo il boot, dalla shell linux

```
target $ insmod /lib/modules/2.6.25-rc1-
davinci1/kernel/drivers/i2c/chips/tda8023.ko
```

Il driver rileva automaticamente l'inserzione e l'estrazione della SIM card nell'apposito slot.
Quando si inserisce la card, ovvero ogni volta che la card è attivata, questa trasmette un pacchetto chiamato ATR (Answer To Reset), contenente informazioni sulla card stessa, i protocolli che supporta, ecc.
L'indirizzo del TDA8023 è 0x22, sicchè verrà creata la cartella /sys/bus/i2c/devices/1-0022/.
Il driver ha due attributi in scrittura (write, test) e uno in lettura (atr).
Per trasmettere un singolo frame (1 byte) alla card

```
target $ echo 85 > /sys/bus/i2c/devices/1-0022/write
```

Per leggere la sequenza di ATR

```
target $ cat /sys/bus/i2c/devices/1-0022/atr
```

L'attributo test serve a eseguire una serie di funzioni di test. Ad esempio, per stampare a monitor il contenuto del registro interno di stato

```
target $ echo 6 > /sys/bus/i2c/devices/1-0022/test
```

Per stampare a monitor il contenuto del buffer di ricezione (tutti i frame ricevuti fino ad ora)

```
target $ echo 9 > /sys/bus/i2c/devices/1-0022/test
0x3b 0x98 0x94 0x00 0x93 0x91 0x11 0x03 0x04 0x03 0x04 0x01
```

Per eseguire il warm reset del TDA8023

```
target $ echo 0 > /sys/bus/i2c/devices/1-0022/test
```

Per attivare la card

```
target $ echo 1 > /sys/bus/i2c/devices/1-0022/test
```

Per programmare a 5V il livello VCC sulla card

```
target $ echo 2 > /sys/bus/i2c/devices/1-0022/test
```

Per programmare a 1.8V il livello VCC sulla card

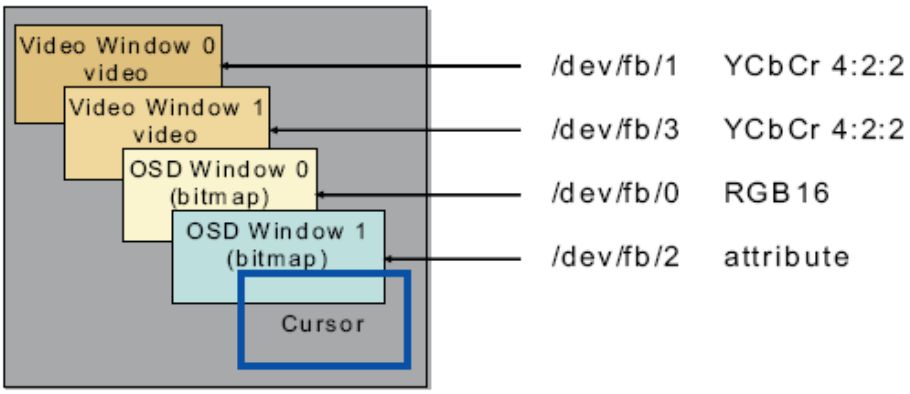
```
target $ echo 3 > /sys/bus/i2c/devices/1-0022/test
```

La lettura dei dati interni alla SIM card (pin, numeri di telefono, ecc.) richiede l'implementazione del protocollo ISO7816-3, non contemplato tra le funzionalità del driver.

8.5 Driver FBdev

Nel kernel space Linux prevede già delle API standard per l'I/O dei file video (FBdev/DirectFB, V4L2). Per la parte video-capture l'interfaccia standard è V4L2, che supporta già i modi BT.656 e

Y/C (ma non supporta le funzioni resizer e istogramma). Per la parte video-display FBdev è un driver di tipo char che mappa in user space il frame buffer del display. Per visualizzare delle immagini il SW applicativo deve solamente copiare l'immagine in questo buffer.
I'LCD controller del DaVinci ha fino a quattro window, o layer, di visualizzazione OSD0,WIN0,OSD1,WIN1 (più un background e un cursore), cui corrispondono i quattro nodi di accesso indipendenti /dev/fb/0,1,2,3. Con le usuali funzioni di open, close, read, write è possibile lavorare sul display, e conoscerne-modificarne le proprietà hardware con delle ioctl predefinite.



Ciascun layer può funzionare in modo Field (interallacciato) o Frame (progressivo).
Il layer OSD0 è usato per visualizzare la grafica (bitmap o RGB656). In RGB656 ogni pixel ha 16 bit (64k colori). I dati RGB presenti in ram sono convertiti in YCbCr prima di passare al modulo OSD.
Il layer OSD1 è di solito usato come una attribute-window, per implementare il blending tra i layer OSD0 e WIN0.
L'attivazione a basso livello del controller VPBE in funzione del modulo display del IPvPhone è implementata nel file

```
arch/arm/mach-davinci/lcd.c
```

Per testare la funzionalità del display con FBdev sul target IPvPhone utilizzando il NFS

131) assicurarsi che le seguenti opzioni siano abilitate in .config

```
CONFIG_FB =y
CONFIG_FB_DAVINCI =y
```

132) Compilare immagine kernel

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
```

133) Accendere il target. Dopo il boot, dalla shell linux

```
target $ cd /opt/video/demo
target $ bash demo &
```

A questo punto lo script demo carica sul buffer del OSD0 (/dev/fb/0) 4 immagini 320x240, a rotazione. Le immagini sono di tipo RGB656 e sono state ottenute da immagini RGB888 tramite l'eseguibile bmpToRgb16, presente sempre nella cartella /demo.
Le impostazioni di default per FBdev (quali layers attivare, con quale risoluzione e dimensione di buffer) possono essere date come parametri di boot. Ad esempio

```
IPvPhone# setenv bootargs $(bootargs)
video=davincifb:output=lcd:osd0=320x240x16,300K@0,0:osd1=off:vid0=off:vi
d1=off
```

Invece da shell esiste il commando fbdev.
Ad esempio, per impostare la window VID0 per accettare un formato NTSC con un buffering triplo:

```
target $ fbset -fb /dev/fb/1 -xres 720 -yres 480 -vxres 720 -vyres 1440
```

8.6 Driver per il Real Time Clock PCF8563

Il Modulo Digitale del videotelefono è dotato di Real Time Clock/Calendar. Il dispositivo è un NXP PCF8563TS (U15). Un Supercap da 1F provvede alla alimentazione dell'RTC anche in assenza di alimentazione esterna.
Anche la programmazione del PCF8563 avviene tramite bus I2C. L'indirizzo del PCF8563 è 0x51.
Il driver è stato implementato sul kernel 2.6.25-davinci1 nei file:

```
/drivers/rtc/rtc-core.ko
/drivers/rtc/rtc-pcf8563.ko
```

e vanno caricati in questo stesso ordine.
Vediamo in questo paragrafo come installare e testare il driver utilizzando il NFS. Per default, in /arch/arm/configs/davinci_ipvp_defconfig il driver è stato configurato come built-in. Quindi

134) assicurarsi che le seguenti opzioni siano abilitate in .config

```
CONFIG_RTC_DRV_PCF8563 =y
CONFIG_RTC_INTF_DEV =y
```

L'accesso al driver da user space avviene attraverso il device node /dev/rtc0 (253/0) e le chiamate di sistema open(), close(), read(), write(), ioctl(), select().

Nota: menuconfig mette a disposizione anche altre interfacce alle funzioni RTC, come attributi sysfs in sola lettura (in /sys/class/rtc) e una interfaccia procfs (in /proc/driver/rtc)

135) Compilare immagine kernel e moduli, e trasferirli nel root filesystem

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
host $ cp arch/arm/boot/uImage /tftpboot
```

136) Sul NFS

```
host $ mkdir -p /home/user/workdir/filesys/opt/rtc
host $ cd /home/user/workdir/filesys/opt/rtc
```

137) Creare un file rtctest.c tipo

```
/*****************************************************************
 * RTCTEST.C
 * Gabriele Filosofi 2008 - ADFL Consulting s.r.l
 * RTC Driver Test Program on the IPvPhone board
 *****************************************************************/
#include <stdio.h>
```

```
#include <linux/rtc.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>

#define CMD_SET_DATETIME   "set"
#define CMD_RD_DATETIME    "read"

static const char default_rtc[] = "/dev/rtc0";
static const char default_cmd[] = CMD_RD_DATETIME;

int main(int argc, char **argv)
{
      int i, fd, retval, irqcount = 0;
      const char *cmd = default_cmd;
      unsigned long tmp, data;
      struct rtc_time rtc_tm;
      const char *rtc = default_rtc;
      switch (argc) {
      case 3:
      {
            rtc = argv[1];
            cmd = argv[2];
      }
      break;
      case 2:
            rtc = argv[1];
      break;
      case 1:
      break;
      default:
            fprintf(stderr, "usage: rtctest [rtcdev] [set/read]\n");
            return 1;
      }

      fd = open(rtc, O_RDWR);

      if (fd ==  -1) {
            perror(rtc);
            exit(errno);
      }

      fprintf(stderr, "IPvPhone: RTC Driver Test.\n");

test_WRITE:
      if(!strcmp(cmd,CMD_SET_DATETIME))
      {
            int year, month, day, hour, min, sec;
            printf("\nPlease, set current Date and Time.\n");
            printf(" set day (1..31) ");
            scanf("%d", &day);
            printf(" set month (1..12) ");
            scanf("%d", &month);
```

```
            printf(" set year (2000..) ");
            scanf("%d", &year);
            printf(" set hours (0..23) ");
            scanf("%d", &hour);
            printf(" set minutes (0..59) ");
            scanf("%d", &min);
            printf(" set seconds (0..59) ");
            scanf("%d", &sec);

            rtc_tm.tm_year = year - 1900;
            rtc_tm.tm_mon = month - 1;
            rtc_tm.tm_mday = day;
            rtc_tm.tm_hour = hour;
            rtc_tm.tm_min = min;
            rtc_tm.tm_sec = sec;
            retval = ioctl(fd, RTC_SET_TIME, &rtc_tm);
            if (retval == -1) {
                 perror("RTC_SET_TIME ioctl");
                 exit(errno);
            }
      }

test_READ:
      /* Read the RTC time/date */
      retval = ioctl(fd, RTC_RD_TIME, &rtc_tm);
      if (retval == -1) {
            perror("RTC_RD_TIME ioctl");
            exit(errno);
      }

      fprintf(stderr, "Current RTC date/time is %d-%d-%d,
%02d:%02d:%02d.\n",
            rtc_tm.tm_mday, rtc_tm.tm_mon + 1, rtc_tm.tm_year + 1900,
            rtc_tm.tm_hour, rtc_tm.tm_min, rtc_tm.tm_sec);
done:
      close(fd);

      return 0;
}
```

138) compilare

```
host $ arm_v5t_le-gcc rtctest.c -o rtctest
```

139) Accendere il target
140) Dalla shell linux del target è possibile leggere la data/ora

```
target $ cd /opt/rtc/
target $ ./rtctest /dev/rtc0 read

IPvPhone: RTC Driver Test.

Current RTC date/time is 30-10-2008, 00:36:35.
```

141) Oppure inserire una nuova data/ora

```
target $ ./rtctest /dev/rtc0 set
```

```
IPvPhone: RTC Driver Test.

Please, set current Date and Time.
 set day (1..31) 31
 set month (1..12) 10
 set year (2000..) 2008
 set hours (0..23) 12
 set minutes (0..59) 21
 set seconds (0..59) 0
Current RTC date/time is 31-10-2008, 12:21:00.
```

Sul dispositivo PCF8563 è possibile programmare una serie di allarmi in base alle informazioni di data/ora. Gli eventi allarme sono comunicati attraverso una linea di share interrupt della IPvPhone. Al momento il driver non implementa queste features sicchè i corrispondenti comandi di ioctl non vengono gestiti.

8.7 Seriale ausiliaria UART-1

Nel Modulo Digitale la seriale TTL UART1 è utilizzata come canale di controllo verso il Modulo Analogico PSTN. Tramite comandi AT, su questa seriale l'applicazione è in grado di controllare il funzionamento del Modulo Analogico. La UART1 è mappata sotto in /dev/tts/1.
Il canale di controllo è configurato a 38400 bps.
Per verificarne il funzionamento utilizzando il NFS

142) Collegare la seriale di un PC ai pin 1 (+3.3V), 5 (GND) , 20 (DR), 22 (DX) di J12, utilizzando un convertitore TTL/RS-232
143) Aprire sul PC una sessione hyperterminal sulla porta collegata
144) Accendere il target. Dopo il boot, dalla shell linux lanciare utilità di configurazione minicom

```
target $ minicom -s
```

145) Settare Serial Port Setup

```
Serial device: /dev/tts/1
Bps/Par/Bits: 115200 8N1
Flow Control: No
```

e salvare configurazioni

```
Save setup as dfl
```

A questo punto la comunicazione bidirezionale tra i due terminali è funzionante

Più in generale il kernel è configurato per disporre di 3 porte seriali asincrone.

```
target $ cat /proc/tty/driver/serial
serinfo:1.0 driver revision:
0: uart:16550A mmio:0x01C20000 irq:40 tx:2561 rx:108 RTS|CTS|DTR|DSR
1: uart:16550A mmio:0x01C20400 irq:41 tx:0 rx:0 CTS|DSR
2: uart:16550A mmio:0x01C20800 irq:42 tx:0 rx:0 CTS|DSR
```

La UART2 non è usata sul Modulo Digitale, mentre UART0 è la seriale console.

9 Software

To do

9.1 DVSDK

9.1.1 Re-build del SW DVSDK 1.30

146) andare nella directory di installazione del DVSDK

```
host $  cd /home/user/dvsdk_1_30_00_40
```

147) assicurarsi che in Rules.make la piattaform e tutte le path siano corretti

```
PLATFORM=dm6446
DVSDK_INSTALL_DIR=$(HOME)/dvsdk_1_30_00_40
CE_INSTALL_DIR=$(DVSDK_INSTALL_DIR)/codec_engine_2_00_01
XDAIS_INSTALL_DIR=$(DVSDK_INSTALL_DIR)/xdais_6_00_01
LINK_INSTALL_DIR=$(DVSDK_INSTALL_DIR)/dsplink_140-05p1
CMEM_INSTALL_DIR=$(DVSDK_INSTALL_DIR)/cmem_2_00_01
CODEC_INSTALL_DIR=$(DVSDK_INSTALL_DIR)/dm6446_dvsdk_combos_1_34
XDC_INSTALL_DIR=$(DVSDK_INSTALL_DIR)/xdc_3_00_02
FC_INSTALL_DIR=$(DVSDK_INSTALL_DIR)/framework_components_2_00_01
BIOS_INSTALL_DIR=$(DVSDK_INSTALL_DIR)/bios_5_31_08
LINUXKERNEL_INSTALL_DIR=$(HOME)/workdir/lsp/ti-davinci
MVTOOL_DIR=/opt/mv_pro_4.0.1/montavista/pro/devkit/arm/v5t_le
MVTOOL_PREFIX=$(MVTOOL_DIR)/bin/arm_v5t_le-
EXEC_DIR=$(HOME)/workdir/filesys
```

148) compilare

```
host $ make clean
host $ make
host $ make install
```

I file generati da questa compilazione si trovano in una cartella del NFS, e precisamente in

```
/home/user/workdir/filesys/opt/dvsdk/dm644x/
```

In particolare ci sono I moduli **cmemk.ko**, **dsplinkk.ko** e **loadmodules.sh**. Questi tre files devono essere sempre presenti. Gli altri files rappresentano delle applicazioni specifiche (demo), e potrebbero cambiare, oltre che trovarsi in una cartella differente del filesystem.

Nota: se è necessario debuggare le applicazioni demo allora eseguire prima

```
host $ make install debug
```

9.2 Utilizzare il ddd Debugger

Il ddd è un'applicazione con interfaccia grafica che gira sull'host e permette di debuggare un'applicazione sul target. Vediamo come si applica all'applicazione demo encodedecode

149) andare nella directory di installazione del DVSDK

```
host $ cd /home/user/dvsdk_1_30_00_40
```

150) compilare il dvsdk in modalità debug

```
host $ make clean
host $ make
host $ make install debug
```

151) copiare nel filesystem l'applicazione debug **encodedecoded**

```
host $ cp demos/dm6446/encodedecode/debug/encodedecoded
~/workdir/filesys/dvsdk/dm6446/
```

152) accendere il target in modalità NFS ed eseguire

```
target $ cd /opt/dvsdk/dm6446/
target $ ./loadmodules.sh
target $ gdbserver 192.160.0.243:1000 encodedecoded
Process decoded created; pid = 602
Listening on port 1000
```

Nota: l'indirizzo IP è quello dell'host
Nota: è possibile specificare eventuali argomenti (p.es. -h)

153) sull'host eseguire

```
host $ ddd -debugger
/opt/dm644x/mv_pro_4.0.1/montavista/pro/devkit/arm/v5t_le/bin/arm_v5t_le
-gdb encodedecoded
```

154) dal prompt ddd connettersi al target

```
(gdb) target remote 192.168.0.244:1000
```

Nota: l'indirizzo IP è quello del target

155) attivare un breakpoint e lanciare l'applicazione con CONT (non usare Run)
156) continuare l'esecuzione con NEXT

9.3 Utilizzare il DVTB

Il DVTB è un utile strumento per acquisire, codificare, decodificare e mandare in playback uno o più stream di tipo audio, speech o video

157) compilare il DVTB

```
host $ cd dvtb_1_12_000/
host $ make clean CONFIGPKG=dm6446
host $ make CONFIGPKG=dm6446
```

Nota: attenzione a non postporre “clean” altrimenti viene cancellato il file dm6446

158) copiare i files **dvtb-d** e **dvtb-r** nella cartella del filesystem dove si trovano gli eseguibili DSP (con estensione .x64P), ad esempio in

```
/home/user/workdir/filesys/opt/dvsdk/dm644x/
```

Per entrare nel prompt dei comandi DVTB sul Target eseguire

```
target $ ./loadmodules.sh
target $ ./dvtb-r
<DVTB>$
```

Esempio: per decodificare e mandare in playback sulla cuffia il file pinocchio.mp3

```
<DVTB> $ setp engine name decode
<DVTB> $ setp auddec codec mp3dec
<DVTB> $ setp auddec maxSampleRate 48000
<DVTB> $ func auddec -s pinocchio.mp3
```

Esempio: per acquisire un video PAL in formato YUV422

```
<DVTB> $ setp vpfe standard 2
<DVTB> $ setp vpfe width 720
<DVTB> $ setp vpfe height 576
<DVTB> $ func videnc –t capture.yuv --nodsp
```

Per bloccare digitare `exit`

Esempio: per acquisire e codificare un video PAL in formato MPEG4

```
<DVTB> $ setp vpfe width 720
<DVTB> $ setp vpfe height 480
<DVTB> $ setp engine name encode
<DVTB> $ setp videnc codec mpeg4enc
<DVTB> $ setp videnc numFrames 300
<DVTB> $ func videnc –t capture.mpeg4
```

Per bloccare digitare `exit`

Nota: lo stesso vale per un file .264, basta usare h264enc invece di mpeg4enc
Nota: il codificatore non codifica frame con altezza maggiore di 480 linee
Nota: se poi si deve visualizzare su un 320x240 allora conviene acquisire in questo formato, tuttavia cambiando semplicemente i parametri vpfe width e height non si ha un vero resizing ma solo un restringimento del campo.

Esempio: per decodificare e mandare in playback sull’uscita CVBS un file video MPEG4 , ad esempio lo stesso file acquisito nell’esempio precedente

```
<DVTB> $ setp vpbe device /dev/fb/3
<DVTB> $ setp engine name decode
<DVTB> $ setp viddec numFrames 300
```

```
<DVTB> $ setp viddec codec mpeg4dec
<DVTB> $ setp viddec maxWidth 720
<DVTB> $ setp viddec maxWidth 480
<DVTB> $ func viddec -s ./capture.mpeg4
```

Nota: la prima linea specifica che il video venga mandato sulla window video vid1. Se si volesse usare vid0 basta indirizzare /dev/fb/1
Nota: lo stesso vale per un file .264, basta usare h264enc invece di mpeg4enc
Nota: ovviamente per vedere un video PAL occorre aver impostato le giuste opzioni al boot, come p.es.

```
setenv bootargs $(bootargs)
video=davincifb:vid0=720x576x16,2500K:vid1=720x576x16,2500K:osd0=320x240
x16,450K@100,100 davinci_enc_mngr.ch0_output=COMPOSITE
davinci_enc_mngr.ch0_mode=$(videostd)
```

Nota: se il video non si vede una possibile spiegazione è che è coperto da un layer osd definito al boot con la stessa risoluzione e posizione del layer video. Il background del layer osd1 è blu, del layer osd0 è nero, mentre quello dei layer vid è verde.

Esempio: per codificare un file video YUV422 (p.es. `capture.yuv`) in formato mpeg4

```
<DVTB> $ setp engine name encode
<DVTB> $ setp videnc inputWidth 720
<DVTB> $ setp videnc inputHeight 576
<DVTB> $ setp videnc codec mpeg4enc
<DVTB> $ func videnc -s capture.yuv -t capture.mpeg4
```

Per bloccare digitare `exit`

Nota: lo stesso vale per una codifica in .264, basta usare h264enc invece di mpeg4enc

E' anche possibile eseguire degli script, con estensione .dvs

```
target $ ./dvtb-r -s myscript.dvs
```

9.4 Codec Server e Codec Engine

9.4.1 Ricompilare i Codec Engine di esempio presenti nel DVSDK

Per testare la corretta installazione di tutti i componenti legati al DSP è bene ricompilare il CE Server di esempio.

159) spostarsi nella cartella /home/user/dvsdk_1_30_00_40/codec_engine_2_00_01/examples/
160) editare il file **user.bld** andando a

- specificare le path dei compilatori MV per ARM e TI per DSP
- commentare le righe relative ai target che non interessano

161) editare il file **xdcpaths.mak** andando a

- specificare le path della cartelle di installazione di tutti i componenti SW già installati

162) ricompilare tutti i *codecs*

```
host $ cd ti/sdo/ce/examples/codecs
host $ gmake clean
host $ gmake
```

163) ricompilare tutti gli *example extensions*

```
host $ cd ../extensions
host $ gmake clean
host $ gmake
```

164) ricompilare tutti i *DSP servers*

```
host $ cd ../server
host $ gmake clean
host $ gmake
```

165) ricompilare tutte le applicazioni *GPP* (cioè del core ARM)

```
host $ cd ../apps
host $ gmake clean
host $ gmake
```

Nota: potrebbe essere necessario specificare la path del compilatore MV in qualche makefile specifico.
Nota: in alcuni casi gli eseguibili per l'ARM hanno estensione *.x470MV*, in altri *.out*; gli eseguibili per il DSP hanno estensione *.x64P*. I files con estensione *.dat* sono samples audio o video.

A questo punto le applicazioni eseguibili sono pronte per essere trasferite sul target ed eseguite.

9.4.2 Ricompilare il DSP server video_copy esistente e provarlo sul target

Cerchiamo di ricompilare il progetto **video_copy** ed eseguirlo sul target.
Questa applicazione prende un file di dati video non compressi (*in.dat*), lo codifica e successivamente lo decodifica. Il risultato è un file che viene salvato nella directory corrente (*out.dat*) e che dovrà essere uguale al file originale.

166) spostarsi nella cartella dell'applicazione ARM

```
codec_engine_2_00_01/examples/ti/sdo/ce/examples/apps/video_copy/dualcpu
/evmDM6446/
```

167) ricompilare

```
host $ gmake clean
host $ gmake
```

168) copiare l'eseguibile ARM e il file di dati in una directory del file system

```
host $ cp app.out /home/user/workdir/filesys/opt/video_copy/
host $ cp in.dat /home/user/workdir/filesys/opt/video_copy/
```

169) spostarsi nella cartella dell'applicazione DSP server

```
codec_engine_2_00_01/examples/ti/sdo/ce/examples/servers
/video_copy/evmDM6446/
```

170) ricompilare

```
host $ gmake clean
host $ gmake
```

171) copiare l'eseguibile DSP nel file system

```
host $ cp video_copy.x64P /home/user/workdir/filesys/opt/video_copy/
```

172) copiare

```
codec_engine_2_00_01/examples/apps/system_files/davinci/cmemk.ko
codec_engine_2_00_01/examples/apps/system_files/davinci/dsplinkk.ko
codec_engine_2_00_01/examples/apps/system_files/davinci/loadmodules.sh
```

nel file system. I due moduli devono essere ricompilati solo se è stato ricompilato il kernel, altrimenti si possono prendere quelli già presenti in questa directory

173) dal target lanciare l'applicazione

```
target $ cd /opt/video_copy
target $ ./loadmodules.sh
target $ ./app.out
```

L'applicazione dovrebbe creare un file out.dat identico a in.dat.

9.4.3 Ricompilare il DSP server video_copy con una mappa di memoria ristretta a 128MB e provarlo sul target

Cerchiamo adesso di ricompilare il progetto video_copy con la mappa di memoria modificata. La seguente figura mostra un esempio del mappa ottenibile su un banco da 128 MB

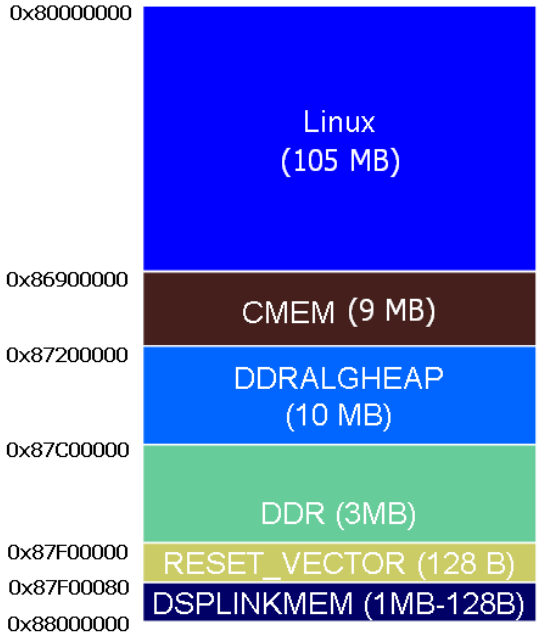


Figura 1. Memory Map 128 MB DDR2 (esempio)

174) Per conservare i file originali, spostarsi nella cartella /home/user/dvsdk_1_30_00_40/ e creare una copia della cartella codec_engine_2_00_01/examples

```
host $ cp -rf codec_engine_2_00_01/examples/ adflce/
```

175) In realtà il file di configurazione .tcf del DSP server video_copy include quello del DSP server all_codecs, in quanto è identico. Per modificarlo conviene copiarlo tale e quale

```
host $ cd adflce/examples/ti/sdo/ce/examples/servers/
host $ cp all_codecs/all.tcf video_copy/evmDM6446/video_copy.tcf
```

176) Editiamo il file

```
host $ cd video_copy/evmDM6446/
host $ vim video_copy.tcf
```

Il file deve essere modificato nel seguente modo

```
var mem_ext = [
{
    comment:    "DDRALGHEAP: off-chip memory for dynamic algmem allocation",
    name:       "DDRALGHEAP",
    base:       0x87200000,   //114 MB
    len:        0x00A00000,   // 10 MB
    space:      "code/data"
},
{
    comment:    "DDR2: off-chip memory for application code and data",
    name:       "DDR2",
    base:       0x87C00000,   //124 MB
    len:        0x00300000,   //  3 MB
```

```
    space:      "code/data"
},
{
    comment:    "RESET_VECTOR: off-chip memory for the reset vector table",
    name:       "RESET_VECTOR",
    base:       0x87F00000,   //127 MB
    len:        0x00000080,   //128 B
    space:      "code/data"
},
{
    comment:    "DSPLINK: off-chip memory reserved for DSPLINK code and data",
    name:       "DSPLINKMEM",
    base:       0x87F00080,   //127 MB + 128 B
    len:        0x000FFF80,   //  1 MB - 128 B
    space:      "code/data"
},
```

];

Commentare inoltre le seguenti linee

```
/*  ============================================================
 *  MEM : Global
 *  ============================================================*/
//prog.module("MEM").BIOSOBJSEG = bios.DDR2;    //comment line out if present
//prog.module("MEM").MALLOCSEG  = bios.DDR2;  //comment line out if present

/*  ============================================================
 *  TSK : Global
 *  ============================================================*/
```

177) Rebuild del DSP server

```
host $ gmake clean
host $ gmake
```

Copiare il file di output nel target file system.

178) Il passo successivo è eseguire il build dell'applicazione ARM, dopo aver editato la mappa di memoria in *ceapp.cfg*.

```
host $ cd ../../../apps/video_copy/dualcpu/evmDM6446/
host $ vim ceapp.cfg
```

La mappa dovrebbe essere del tipo

```
osalGlobal.armDspLinkConfig = {
    memTable: [
        ["DDRALGHEAP",    {addr: 0x87200000, size: 0x00A00000, type: "other"}],
        ["RESET_VECTOR", {addr: 0x87F00000, size: 0x00000080, type: "reset"}],
```

```
        ["DDR2",          {addr: 0x87C00000, size: 0x00300000, type: "main" }],
        ["DSPLINKMEM",    {addr: 0x87F00080, size: 0x000FFF80, type: "link" }],
    ],
};
```

Copiare nel target file system app.out

179) Il passo successivo è modificare lo script loadmodules.sh

```
host $ cd ../../../ ../../../ ../../apps/system_files/davinci/
host $ vim loadmodules.sh
```

Modificare nel seguente modo

```
insmod cmemk.ko phys_start=0x86900000 phys_end=0x87200000
pools=20x4096,10x131072, 2x1048576
```

180) Resettare il target e aggiornare l'opzione di boot che definisce l'area di memoria riservata a Linux. Nell'esempio,

```
> setenv bootargs console=ttyS0,115200n8 noinitrd rw ip=192.168.0.244
root=/dev/nfs nfsroot=192.168.0.243:/home/gabriele/workdir/filesys,nolock
mem=105M
```

Dopo il reboot della scheda lanciare l'applicativo

```
host $ cd /opt/video_copy/
host $ sh loadmodules.sh
host $ ./app.out
```

In qualsiasi momento è possibile conoscere nell'applicazione la quantità di memoria allocata da CMEM, basta aggiungere la chiamata

```
system( "/bin/cat /proc/cmem" );
```

nel sorgente ARM.
Per conoscere la memoria allocata da un processo si può usare la chiamata

```
usedmem = Engine_getUsedMem(engineHandle);
```

Per avere il log dettagliato di tutte le operazioni interne è possibile lanciare l'applicazione con la seguente opzione

```
host $ CE_TRACE="*=01234567" TRACEUTIL_DSP0TRACEMASK="*=01234567"
TRACEUTIL_DSP0TRACEFILE="cedsp0log.txt" CE_TRACEFILEFLAGS="w"
CE_TRACEFILE="cearmlog.txt" TRACEUTIL_REFRESHPERIOD=200 ./app.out
```

I file di log si trovano nella directory corrente (cearmlog.txt e cedsp0log.txt)

9.4.4 Creare un nuovo DSP server a partire da video_copy

To do..

DSP codec server:
- fatto una copia zippata di backup di dvsdk_1_30_00_40/codec_engine_2_00_01/examples/
  host $ tar -cvzf examples_org.tar.gz examples/
- host $ cd examples/ti/sdo/ce/examples/
- copiato video_copy/ in adflsrv/
- cd adflserv/
- modificato package.xdc
- cd evmDM6446
- modificato package.xdc
- video_copy.cfg rinominato adflsrv.cfg
- modificato adflsrv.cfg aggiungendo due codec per lo speech (SPHDEC_COPY e SPHENC_COPY)
  in questo file modificare anche le altre cose, come desiderato
- copiato servers/all_codecs/all.tcf in servers/adflsrv/evmDM6446/adflsrv.tcf
- modificato adflsrv.tcf per mappa di memoria a 128MB
- modificato makefile
- modificato main.c e se serve link.cmd
- non toccare le cartella package/ e video_copy/ in quanto verranno rigenerate dal build!!!!
- make clean + make

ARM app:
- host $ cd examples/ti/sdo/ce/examples/apps
- copiato video_copy/ in adflsrv/
- aggiunto in makefile
  $(MAKE) -C adflsrv/dualcpu $@
- cd adflserv/
- rimuovere le cartelle singlecpu/, dualcpu_separateconfig/ e dualcpu_separateconfig_dll/
- cd dualcpu/
- modificato package.xdc
- cd evmDM6446
- modificato package.xdc
- modificato ceapp.cfg per mappa di memoria a 128MB
- make clean + make
- in apps/system_files/davinci modificato loadmodules.sh per mappa di memoria a 128MB

9.4.5 Creare un Server Package con RTSC Server Package Wizard

La strada del copia e incolla per creare dei codec è sempre rischiosa. *RTSC Server Package Wizard* è un tool automatizza la generazione dei files XDC necessari alla creazione di un package di un codec (tramite XDC tools). E' installabile sia su Linux che su Windows.
I files XDC generati sono del tipo:

```
package.xdc
package.bld
package.xs
H264DEC.xdc
link.xdt
```

L'utente può anche inserire altri files (header, applicazioni, librerie, documentazione), ma una libreria almeno è necessaria.

181) Installare in /home/user/ cg_xml 2.12.00 o superiore

```
host $ ./cg_xml-v2_12_00-Linux-x86-Install
```

182) Installare in /home/user/ ceutils 1.06 o superiore

```
host $ ./ceutils-v1.06-Linux-x86-Install
```

183) Assicurarsi di avere installato anche

- Codec Engine 1.20 o superiore
- XDAIS 5.21 o superiore
- XDC Tools 2.95 o superiore

184) Aggiungere in .bashrc le path dei vari componenti SW installati. Ad esempio

```
PATH="/home/gabriele/dvsdk_1_30_00_40/xdc_3_00_02/:$PATH"
export PATH

#to get XDC tools in this location
export
XDCPATH="/home/gabriele/dvsdk_1_30_00_40/codec_engine_2_00_01/packages;/
home/gabriele/dvsdk_1_30_00_40/xdais_6_00_01/packages;/home/gabriele/ceu
tils_1_06/packages"
```

185) Lanciare il RTSC Wizard in modalità GUI

```
host $ xs ti.sdo.codecutils.genpackage -g
```

186) Proseguire seguendo l'help

9.4.6 Installare i Codec Combo ver 1.35

Il DVSDK 1.30 esce con una libreria di codec, server e applicazioni di test presente in

dvsdk_1_30_00_40/dm6446_dvsdk_combos_1_34/

Vediamo come installare la versione 1.35

187) Dal sito

https://www-a.ti.com/extranet/cm/product/dvevmsw/dspswext/general/dm_sw_eval_codecs.shtml/

scaricare il file indicato con

For DVSDK 1.30.00.40 - Updated DM644x Codec Servers - EVALUATION

(versione 1.35 per dm644x)

Il nome del file è `dm644x_dvsdk_codec_servers_1_35_eval-e.zip`

188) Estrarre sotto windows e lanciare il setup

```
dvsdk_servers_dm644x_1_35_e-1.0-Setup.exe
```

189) Lanciare il comando di installazione dalla dalla cartella

```
C:\Programmi\dvsdk_servers_dm644x_1_35_e\dm6446_dvsdk_combos_1_35_evalua
tion\
```

190) Copiare sul Linux host, directory `/home/gabriele/dvsdk_1_30_00_40`, ed estrarre, il file `dm6446_dvsdk_combos_1_35_evaluation.tar`

```
host $ tar -zxf dm6446_dvsdk_combos_1_35_evaluation.tar
host $ cd dm6446_dvsdk_combos_1_35/
```

191) Editare i file config.bld e Makefile.eval inserendo le giuste path per le toolchains e per la directory di installazione del DVSDK

192) Ricompilare

```
host $ gmake -f Makefile.eval clean
host $ gmake -f Makefile.eval
```

193) Copiare gli eseguibili ARM e DSP nel filesystem

```
host $ cp packages-evaluation/ti/sdo/app/decode/decode-sample.x470MV
~/workdir/filesys/opt/dvsdk/combosrv/
host $ cp packages-evaluation/ti/sdo/app/encode/encode-sample.x470MV
~/workdir/filesys/opt/dvsdk/combosrv/
host $ cp packages-evaluation/ti/sdo/servers/decode/decodeCombo.x64P
~/workdir/filesys/opt/dvsdk/combosrv/
host $ cp packages-evaluation/ti/sdo/servers/encode/encodeCombo.x64P
~/workdir/filesys/opt/dvsdk/combosrv/
```

9.4.7 Utilizzare le aplicazioni demo presenti nel dvsdk

Il DVSDK 1.30 esce con tre applicazioni demo, **decode**, **encode** e **encodedecode**, presenti nella directory

```
dvsdk_1_30_00_40/demos/dm6446/
```

La compilazione del dvsdk esegue anche la compilazione delle demo.
Per poter fare in modo che questi i codec `dm6446_dvsdk_combos_1_35` possano essere utilizzati anche dalle applicazioni demo, senza dover modificare la XDC_PATH nei makefile delle demo, rinominiamo la cartella

```
dm6446_dvsdk_combos_1_35/packages-evaluation/
```

```
in
```

```
dm6446_dvsdk_combos_1_35/packages/
```

e modifichiamo in tal senso anche i files Makefile.prod e Makefile.eval, sempre in dm6446_dvsdk_combos_1_35/. Inoltre modifichiamo Rules.make in modo che la path predefinita dei codecs sia `dm6446_dvsdk_combos_1_35/`.

194) Il passo successivo è ricompilare il dvsdk e installare tutti i files necessari nel filesystem

```
host $ make clean
host $ make
host $ make install
```

I file generati da questa compilazione vengono copiati dal comando install in una cartella del NFS, e precisamente in

```
/home/user/workdir/filesys/opt/dvsdk/dm644x/
```

In particolare ci sono I moduli **cmemk.ko**, **dsplinkk.ko** e **loadmodules.sh**.
Se necessario il file loadmodules.sh deve essere modificato in funzione della mappa di memoria

195) Resettare il target e impostare la seguente opzione di boot

```
IPvPhone# setenv bootargs console=ttyS0,115200n8 noinitrd rw
ip=192.168.0.244 root=/dev/nfs
nfsroot=192.168.0.243:/home/gabriele/workdir/filesys,nolock mem=105M
IPvPhone# setenv bootargs $(bootargs)
video=davincifb:vid0=720x576x16,2500K:vid1=720x576x16,2500K:osd0=720x576x16
,1620K:osd1=720x576x4,810K davinci_enc_mngr.ch0_output=COMPOSITE
davinci_enc_mngr.ch0_mode=$(videostd)
```

Dopo il reboot della scheda lanciare l'applicativo (nell'esempio decode)

```
target $ cd /opt/dvsdk/dm6446/
target $ sh loadmodules.sh
target $ ./decode -v videofile.mpeg4 -s speechfile.g711
```

9.4.8 Lanciare le applicazioni demo da remoto usando script cgi

E' possibile lanciare le demo anche da remoto (dall'host) mediante uno script CGI attivabile da una pagina http che è resa disponibile da un web server **thttpd** che gira sul target (default porta 80).
Per lanciare il server

```
target $ cd /opt/dvsdk/dm6446/web
target $ ./thttpd_wrapper
```

Nota: verificare che nel file `thttpd_wrapper` sia presente anche l'opzione "–u root"
Nota: per verificare che il server ("www") è in ascolto usare

```
target $ netstat -l
```

196) dall'host collegarsi con un browser alla pagina http

```
http://192.168.0.244/index.html
```

Nota: l'indirizzo IP è quello del target
Nota: se si usa Firefox deselezionare l'opzione *Work Offline* nel menu *File*
Nota: i permessi di index.html devono essere 666 (-rw-rw-rw-)

197) attivare gli script CGI con un click

Nota: i permessi dei file in /cgi-bin devono essere 755 (-rwxr-xr-x)

Il comando `make install` eseguito sul DVSDK copia nel file system (sotto /`opt/dvsdk/dm6446/`) la cartella

`dvsdk_1_30_00_40/examples/dm6446/web/`

contenente tutti i files necessari, compreso il programma web server thttpd.
Se è necessario ricompilarlo lo si può fare, sull'host, nel seguente modo

```
host $ cd /home/user/dvsdk_1_30_00_40/examples/dm6446/thttpd-2.25b/
host $ CC=arm_v5t_le-gcc ./configure
host $ make
host $ cp thttpd ../web
host $ cd /home/user/dvsdk_1_30_00_40/
host $ make install
```

10 Appendice 1 – Boot della IPvPhone (Modulo Digitale)

Il bootloader si trova in Flash. Ha il compito di inizializzare l'HW (memoria, interfaccia di rete, ecc.), caricare il kernel in RAM e andare a eseguirlo (/sbin/init), eventualmente passando anche dei parametri. La sequenza di eventi prevede i seguenti passaggi

1. ARM boots and starts executing U-boot code from NOR in-place
2. U-boot copies Kernel to RAM (a 0x80008000)
3. U-boot copies filesystem to RAM (copy can be avoided using flash filesystem)
4. U-Boot sets parameters and starts Kernel
5. Kernel uncompresses itself
6. Kernel initialization
7. Driver Initialization
8. Init
9. Init scripts
10. Shell

Il kernel può essere originariamente in Flash oppure su un host collegato tramite rete, nel qual caso viene trasferito al target tramite TFTP. Il root file system può essere fisicamente in Flash oppure sull'host e reso disponibile tramite NFS. Tipicamente si lavora sull'host durante lo sviluppo. Descriviamo qui le prime fasi di boot sulla scheda target IPvPhone.

10.1.1 Modalità di Boot ARM

Al momento in cui il pin /RESET torna alto lo stato logico presente su alcuni pin del DM644x definisce una delle due opzioni: boot da UART (ROM); boot da Flash. La scelta è selezionabile in HW tramite il Dip-Switch SW1 (interruttore #4).

10.1.2 Boot da UART (SW1-4 ON)

Il boot da ROM è la condizione operativa necessaria quando la Flash è vuota ed occorre programmarla.
Nel boot da ROM l'ARM salta all'indirizzo 0x00004000 dove un Bootloader di primo livello (RBL) permette di trasferire un Bootloader di secondo livello (User Bootloader, UBL) che, tipicamente,

1) copia se stesso nella IRAM e salta all'entry point

2) configura il controllore DDR2, il clock di sistema, lo stack, ecc. (dm644x.c)
3) permette di caricare in Flash un Bootloader di terzo livello (U-Boot)

10.1.3 Boot da Flash (SW1-4 OFF)

Il boot da Flash è la condizione operativa normale del Modulo Digitale.
Nel boot da Flash l'ARM inizia il fetch delle istruzioni all'indirizzo 0x02000000 cioè all'inizio della Flash. Chiaramente questa opzione va usata quando la Flash contiene già un programma (come ad esempio UBL oppure direttamente U-Boot, necessario per caricare in RAM il kernel Linux uImage).
La seguente figura da un'idea della successione degli eventi nei due tipi di boot previsti nel IPvPhone.

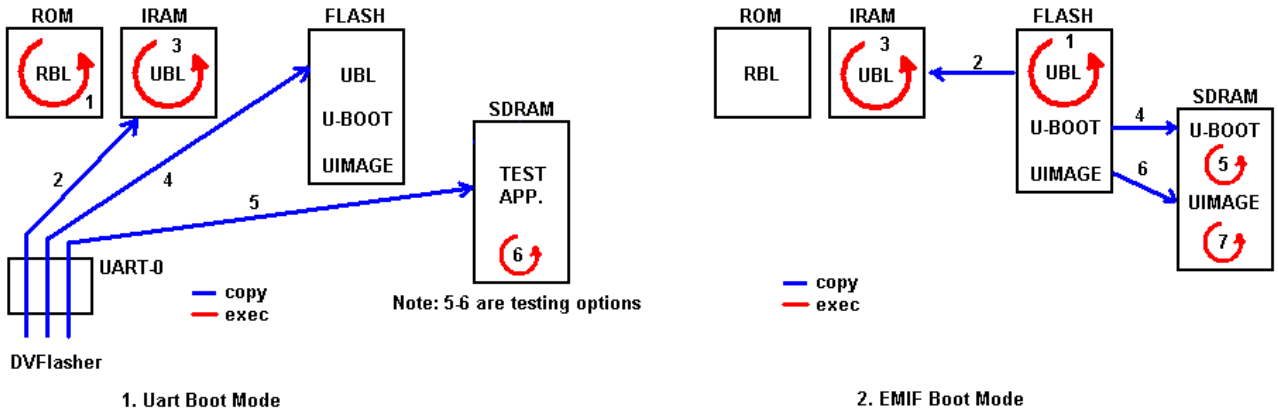


Figura 6. Boot dell'ARM nel Modulo Digitale del IPvPhone

Il processo di boot si articola nei seguenti passi:

2. Dopo il reset, viene eseguito il codice di startup UBL nella IRAM
3. UBL inizializza la CPU e la SDRAM, copia U-Boot in RAM e salta alla prima istruzione del u-boot
4. Il bootloader u-boot decomprime il kernel in RAM dalla flash (o dal TFTP server nel caso di boot da rete) e salta alla prima istruzione del kernel
5. Il kernel configura i registri dell'ARM e invoca `start_kernel`, che è il punto di inizio della parte del kernel indipendente dall'architettura
6. Il kernel inizializza la cache e altre periferiche hardware
7. Il kernel monta il root file system
8. Il kernel esegue il processo `init`
9. `init` carica le librerie condivise (shared runtime libraries)
10. `init` legge il suo file di configurazione, `/etc/inittab`, ed esegue gli scripts
11. `init` entra nel runlevel

11 Appendice – Installazione MPEG Audio Player mpg123

mpg123 è un audio player MPEG 1.0/2.0/2.5 per i layer 1,2 e 3. Sono qui riportati i passi da seguire per compilare e installare mpg123.

1) Sull'host, come root, creare nel NFS la cartella e copiarci l'archivio dei sorgenti (scaricare da http://www.mpg123.de/)

```
host $ cd /home/user/workdir/filesys_2.6.25/opt/
host $ mkdir -p mpg123
host $ cd mpg123/
host $ cp /cartella_sorgente/ mpg123-1.5.1.tar.bz2 .
```

2) Sul target, come root, unzippare i sorgenti

```
target $ cd /opt/mpg123/
target $ tar zvjf mpg123-1.5.1.tar.bz2
target $ rm mpg123-1.5.1.tar.bz2
target $ cd mpg123-1.5.1/
```

3) creare il makefile scegliendo il compilatore arm e il tipo di driver audio

```
target $ ./configure CC=arm_v5t_le-gcc LDFLAGS="-L/usr/lib" CFLAGS="-O3
-mlittle-endian -march=armv5t -mtune=arm9tdmi -nostdinc -
B/usr/lib/gcc/armv5tl-montavista-linuxeabi/3.4.3/ -isystem /usr/include
-isystem /usr/lib/gcc/armv5tl-montavista-linuxeabi/3.4.3/include -
I/usr/include -I/usr/lib/gcc/armv5tl-montavista-linuxeabi/3.4.3/include"
--enable-optimizations --with-audio=oss
```

4) build e installazione

```
target $ make
target $ make install
```

L'eseguibile mpg123 viene copiato in /usr/local/bin e la libmpg123 in /usr/local/lib.

5) mandare in playback un file mp3 (precedentemente copiato nella cartella) e ascoltarlo in cuffia

```
target $ mpg123 pinocchio.mp3

High Performance MPEG 1.0/2.0/2.5 Audio Player for Layers 1, 2 and 3
version 1.5.1; written and copyright by Michael Hipp and others
free software (LGPL/GPL) without any warranty but with best wishes
Playing MPEG stream 1 of 1: pinocchio.mp3 ...
Title:   Traccia 1   Artist: Artista traccia
Comment:
Album:   Fiorenzo Carpi - Le Avventure di Pinocchio
Year:              Genre:  Other
MPEG 1.0 layer III, 320 kbit/s, 44100 Hz joint-stereo
```

6) se la velocità della CPU è insufficiente è necessario interporre un buffer. Ad esempio

```
target $ mpg123 -b 2048 pinocchio.mp3
```

In questo caso l'inizio dell'esecuzione è ritardata in proporzione alla dimensione del buffer. Se la dimensione del buffer è insufficiente periodicamente si sentiranno delle interruzioni dovute al buffer overflow.

12 Appendice – Installazione driver convertitore USB-RS232

Sono qui riportati i passi da seguire per installare il driver del convertitore USB-RS232 della FTDI

7) Come root, copiare in /tmp/dnld e unzippare il driver

```
host $ tar zxf libftd2xx0.4.13.tar.tar
```

8) copiare il driver in /usr/local/lib

```
host $ cp libftd2xx.so.0.4.13 /usr/local/lib
```

9) in /usr/local/lib creare un link simbolico

```
host $ cd /usr/local/lib
host $ ln -s libftd2xx.so.0.4.13 libftd2xx.so
```

10) in /usr/lib creare un link simbolico

```
host $ cd /usr/lib
host $ ln -s /usr/local/lib/libftd2xx.so.0.4.13 libftd2xx.so
```

11) aggiungere in /etc/fstab la seguente linea:

```
none /proc/bus/usb usbdevfs defaults,devmode=0666 0 0
```

(se ci sono problemi provare: none /proc/bus/usb usbfs defaults,mode=0666 0 0)

12) Remount tutto in fstab

```
host $ mount -a
```

Nota: Il punto di mount è /dev/ttyUSB0.
Per verificare che l'inserimento del device viene riconosciuto usare

```
host $ lsusb
```

e inoltre in /dev deve esserci ttyUSB0

13) cambiare permessi

```
host $ chmod 777 /dev/ttyUSB0
```

Per poter usare la seriale come user è necessario eseguire questo comando ogni volta che si reinserisce il connettore USB. Per evitare ciò si può aggiungere al file /home/user/.bashrc la riga

```
sudo chmod 777 /dev/ttyUSB0
```

13 Appendice – Creare un file di backup

Sono qui riportati i passi da seguire creare un file di backup, ad esempio della cartella LSP

```
host $ cd /home/user/workdir/
host $ tar -cvzf lsp_280708.tar.gz lsp/
```

Il file viene creato nella stessa directory e può essere copiato su un qualsiasi supporto di storage. P.es.

```
host $ mv lsp_backup_280708.tar.gz /media/SWISSMEMORY/
```

Per ripristinare il file da un backup

```
host $ tar -xzf lsp_280708.tar.gz
```

14 Appendice – Parametri di boot ausiliari

Per aggiungere parametri di boot ausiliari (relativi al video) ai parametri già esistenti è sufficiente inviare il seguente comando sotto u-boot

```
CON USCITA SU VIDEO AN.
IPvPhone# setenv bootargs $(bootargs)
video=davincifb:vid0=720x576x16,2500K:vid1=720x576x16,2500K:osd0=720x576
x16,2025K davinci_enc_mngr.ch0_output=COMPOSITE
davinci_enc_mngr.ch0_mode=$(videostd)

CON USCITA SU LCD
IPvPhone# setenv bootargs $(bootargs)
video=davincifb:vid0=320x240x16,450K:vid1=off:osd0=320x240x16,300K
davinci_enc_mngr.ch0_mode=320x240 davinci_enc_mngr.ch0_output=LCD
```

Questi parametri servono a inizializzare variabili che servono al driver encoder presente della versione ufficiale TI del kernel (2.6.10, 15 Apr 2008). Il git kernel non ha questo driver, quindi i parametri di cui sopra non verrebbero riconosciuti.
Invece, per usare il driver FBDev presente sotto il kernel git 2.6.25

```
IPvPhone# setenv bootargs $(bootargs)
video=davincifb:output=lcd:osd0=800x480x16,2000K@0x0,0:osd1=800x480,2000
K@0x0,0:vid0=800x576x16,3000K@0x0,0:vid1=800x576x16,3000K@0x0,0
```

Per lanciare uno script o eseguibile al boot del target, ad esempio /bin/myscript,

```
IPvPhone# setenv bootargs $(bootargs)init=/bin/myscript
```

15 Appendice – Creare un nuovo RAM Disk

Per creare un file system ext2 in Flash occorre sempre passare per un RAM disk. Nel paragrafo relativo alla creazione di una immagine JFFS2 siamo partiti da un file predefinito (ramdisk.gz). Nel caso lo spazio disponibile su questo file system fosse insufficiente per contenere le nostre applicazioni occorre crearne uno più grande. La procedura per creare, ad esempio, un RAM disk con 16384 blocchi da 1KB è la seguente (assumiamo esista già il device node /dev/ram)

```
host $ mkdir -p rootfs
```

```
host $ mke2fs -vm0 /dev/ram 16384
host $ mount -t ext2 /dev/ram rootfs
host $ copiare in rootfs le cartelle /bin, /sbin, /etc, /dev ...
host $ umount rootfs/
host $ dd if=/dev/ram bs=1k count=16384 of=ramdisk
host $ gzip -9 ramdisk
```

L'ultimo comando avrà creato il file desiderato ramdisk.gz.
Si noti che i comandi suddetti possono necessitare dei permessi di amministratore, nel qual caso è sufficiente il prefisso `sudo`.

Per andare olre il 16MB, ad esempio 32MB,

```
host $ dd if=/dev/ram bs=1k count=32768 of=ramdisk
host $ mke2fs -vm0 ramdisk 32768
Proceed anyway? (y,n) y
host $ mount -o loop -t ext2 ramdisk rootfs
host $ copiare in rootfs le cartelle /bin, /sbin, /etc, /dev ...
host $ umount rootfs/
host $ gzip -9 ramdisk
```

Per verificare le dimensioni di una directory usare

```
host $ df -h dirname
```

16 Appendice – Modificare la mappa di memoria

16.1 DDR2 Memory Map

All'indirizzo 0x80000000 inizia la memoria SDRAM DDR2. A quest'area sono dedicati al più 256MB. La seguente tabella mostra la mappa di memoria di default del DM644x (256 MB).

Address (hex)	Address (decimal)	Size	Segment	Comments
0x80000000 .. 0x87800000	0-120MB	120MB	Linux	booted with MEM=120M
0x87800000 .. 0x88000000	120-128MB	8MB	CMEM	shared buffers between GPP and DSP
0x88000000 .. 0x8FA00000	128-250MB	122MB	DDRALGHEAP *	DSP segment used exclusively for algorithm heaps
0x8FA00000 .. 0x8FE00000	250-254MB	4MB	DDR*	DSP segment for code, stack, and static data
0x8FE00000 .. 0x8FF00000	254-255MB	1MB	DSPLINKMEM *	memory for DSPLINK
0x8FF00000 .. 0x8FF00080	255MB-255MB	128B	CTRLRESET *	memory for reset vectors
0x8FF00080 .. 0x8FFFFFFF	255MB-256MB	1MB	-- unnamed --	

Tabella 1. Memory Map del DM644x (default)

Il DSP vede la memoria con indirizzi assoluti, mentre l'ARM vede indirizzi virtuali, in quanto ha l'MMU. Il minimo blocco di memoria che è possibile allocare sotto Linux è 4 KB.
Nella tabella, GPP (*General Purpose Processor*) indica il core ARM. CMEM è lo spazio riservato al video I/O buffering tra ARM e DSP. Il motivo per cui esiste CMEM è che il DSP necessita che i buffer siano contigui in memoria, cosa che l'allocazione sotto Linux non garantirebbe.
DDRALGHEAP è l'heap dinamico usato dai codec DSP. Tipicamente occupa da 2 a 200 MB in funzione di quali e quante istanze dei codec vengono contemporaneamente usate run-time dall'ARM. Il segmento DDR (o "DDR2" in CE versione 1.20 e successive), contiene codice, stack e dati statici del DSP. Tipicamente varia da 1-3 MB in funzione di quali codec si decide di utilizzare. La porzione rimanente (tipicamente 1 MB) serve al DSP ed è indipendente dai codec usati.
Il Modulo Digitale può montare tagli da 64 MB, 128 MB o 256 MB, conseguentemente lo spazio fisico massimo dedicato a ciascun segmento può variare rispetto a quello di default. Se lo spazio fisico è inferiore a 256 MB occorre modificare la mappa di memoria, sempre facendo in modo che lo spazio riservato a Linux sia il massimo possibile. Per misurare DDRALGHEAP ci sono vari metodi (http://wiki.davincidsp.com/index.php?title=Changing_the_DVEVM_memory_map#Introduction).
Per misurare DDR è sufficiente guardare il map file dell'applicazione DSP. Per CMEM occorre misurare/calcolare il totale e determinare il partizionamento del pool, cioè il size dei buffer che possono essere allocati. Nel fare questo si suggerisce di riservare la stessa quantità di memoria per il frame non compresso e per il frame compresso.

L'applicazione standard IPvPhone è una videochiamata. Considerando simultaneamente la codifica-decodifica di un canale video D1 e la codifica-decodifica di un canale audio, avremo

frame video raw yuv4:2:2 (2 byte/pixel) da codificare -> (720 x 576 x 2) = 829440 bytes = 810 KB
frame video codificato -> 810 KB
frame audio raw da codificare -> 4 KB
frame audio codificato -> 4 KB
frame video da decodificare -> 810 KB
frame video decodificato -> 810 KB
frame audio da decodificare -> 4 KB
frame audio decodificato -> 4 KB

Quindi, prima che l'applicazione parta, allocheremo 4 MB di CMEM nel seguente modo

```
insmod cmemk.ko phys_start=0x88000000 phys_end=0x88400000 pools=4x829440,4x4096
```

Successivamente l'ARM passerà il buffer del frame raw alla chiamata VIDENC_process() e il buffer del frame raw audio alla chiamata SPHENC_process(); il buffer del frame da decodificare alla chiamata VIDDEC_process() e il buffer del frame audio da decodificare alla chiamata SPHDEC_process().

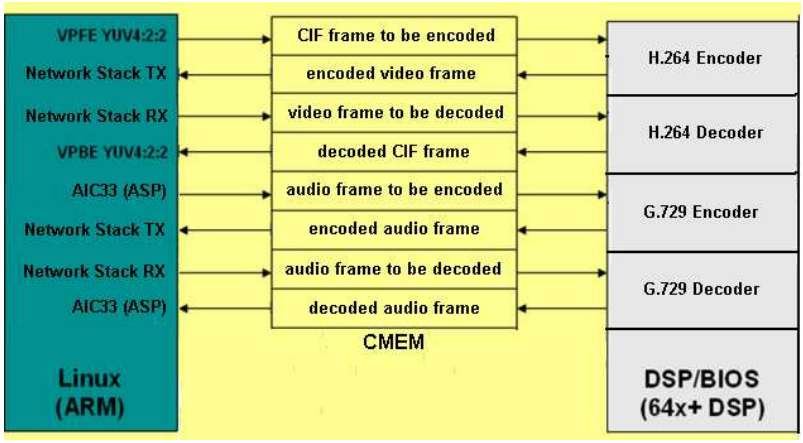
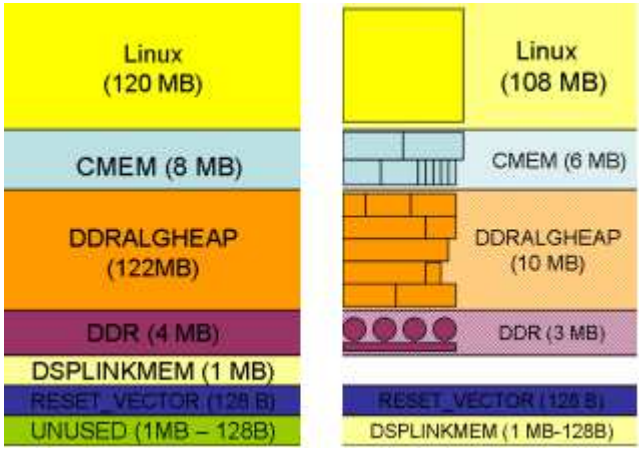


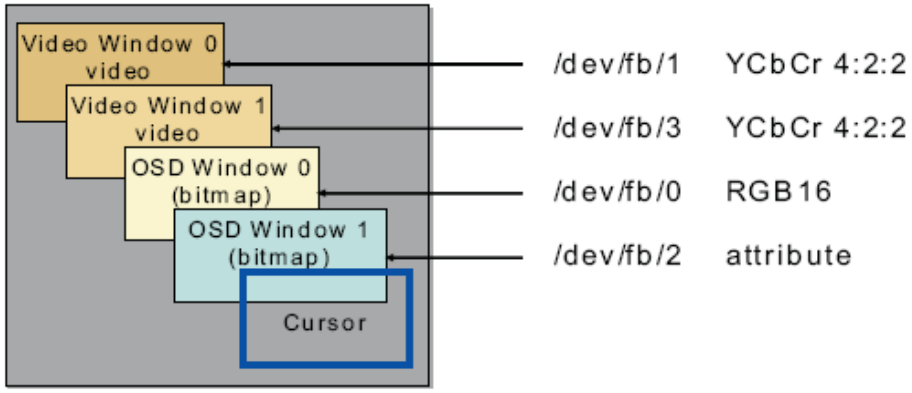
Figura 1. CMEM per videochiamata (esempio)

Una volta note le dimensioni di tutti i segmenti e del size totale della RAM, è necessario calcolare gli indirizzi fisici dove posizionarli, seguendo l'ordine indicato il figura, partendo dal CTRLRESET, avendo l'accortezza di allineare tutto a 4KB e di lasciare qualche margine. Lo spazio riservato a Linux è tutto ciò che rimane dopo aver posizionato il resto.



17 Appendice – Layer OSD1 come Attribute Window

l'LCD controller del DaVinci ha fino a quattro window, o layer, di visualizzazione OSD0,WIN0,OSD1,WIN1 (più un background e un cursore), cui corrispondono i quattro nodi di accesso indipendenti /dev/fb/0,1,2,3. Con le usuali funzioni di open, close, read, write è possibile lavorare sul display, e conoscerne-modificarne le proprietà hardware con delle ioctl predefinite.



Il layer OSD0 permette di aggiungere una immagine grafica 16bpp sul piano del video YCbCr (VID0 o VID1) con un livello di trasparenza programmabile uguale per tutti i pixel (blending).
Il layer OSD1 può funzionare come piano di grafica a 4bpp in aggiunta a OSD0, oppure come Attribute Window. Usato come Attribute Window, OSD1 permette di differenziare il livello di trasparenza di OSD0 e di attivare o meno il blink pixel per pixel (vedi figura).
Ciascun pixel dell'immagine OSD1 è codificato in 4 bit, di cui il più significativo è la flag di blink e i tre bit meno significativi sono il livello di trasparenza (da 0 a 7, **dove 0 è la massima trasparenza**). La parte di OSD1 non sovrapposta a OSD0 non ha influenza (passa il video).

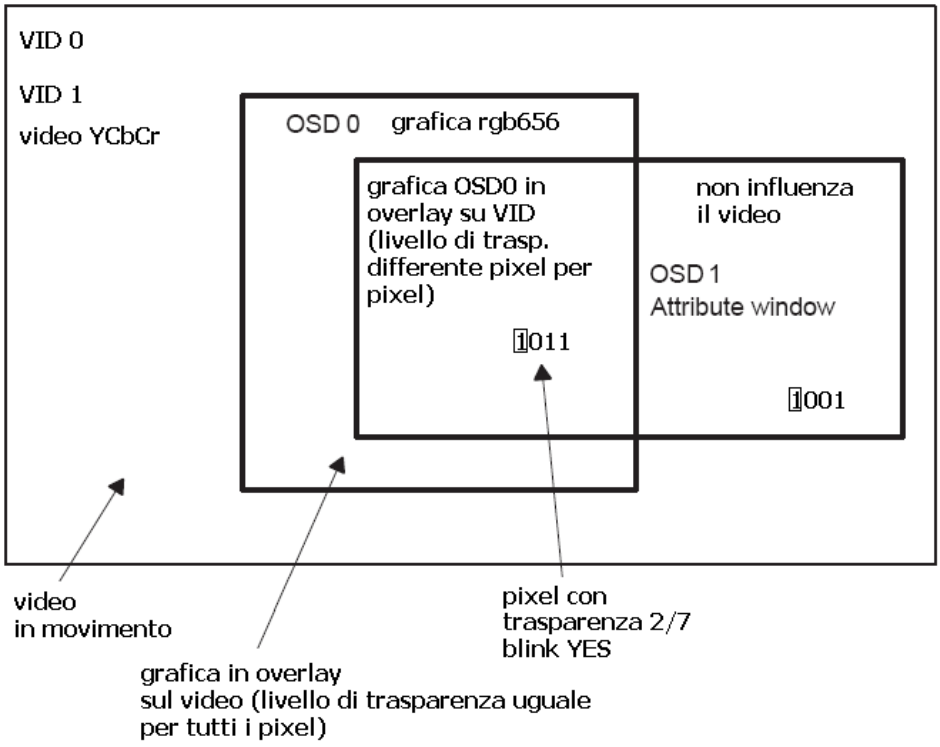


Figura 1. OSD1 come Attribute Window

Per creare un file grafico RGB565 si può procedere come segue

1. Si crea un'immagine bitmap truecolor (24bpp), avente la risoluzione desiderata (sia ad esempio `demo_osd0_640x480.r24`)
2. Si converte il file in RGB656 (16bpp) utilizzando l'utility di conversione di TI **bmpToRgb16** che gira sull'host

```
host $ bmpToRgb16 demo_osd0_640x480.r24 640 480
```

La stessa cosa si può fare sul target, basta ricompilare l'utility per l'ARM

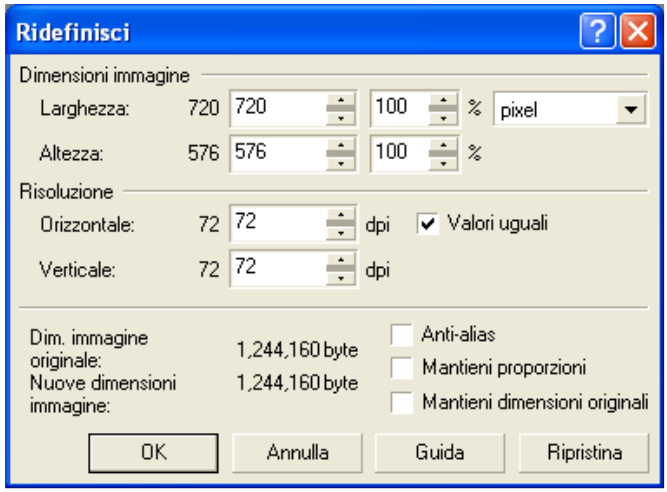
```
host $ arm_v5t_le-gcc bmpToRgb16.c -o bmpToRgb16
target $ bmpToRgb16 demo_osd0_640x480.r24 640 480
```

Il risultato è il file osd.r16, che può essere rinominato

```
host $ mv osd.r16 demo_osd0_640x480.r16
```

Ora che abbiamo un file RGB565 da visualizzare su OSD0 è necessario generare il file OSD1 di maschera che abbia le stesse dimensioni. Per creare questo file si può procedere come segue

1. Si crea con **Paint** un'immagine bitmap 16 colori (cioè 4bpp) avente la stessa risoluzione del file RGB656, tutta nera (valore 00h). Per verificare la risoluzione verticale e orizzontale si può usare **Corel PHOTO PAINT** (Immagine->Ridefinisci)



2. In funzione del livello di trasparenza e della funzione blink desiderata si campiona un colore dal file Color Palette, 16 colori, 320x240 (**osd1_320x240_colorpalette.bmp**) e lo si usa per riempire l'area dell'item interessato nella posizione in cui si trova sul file RGB656
3. Ribaltare verticalmente l'immagine ottenuta. Questa operazione è necessaria in quanto lavorando con un file bmp i bytes in coda al file rappresentano i pixel in testa all'immagine
4. Con un HEX editor (p.es. **Hex Editor Workshop**) cancellare l'header bmp in testa al file (sono i primi 76h bytes)
5. Modificare l'estensione del file, da .bmp a .r4)

La seguente figura mostra la Color Palette per creare le maschere OSD1 lavorando in bitmap 16 colori.

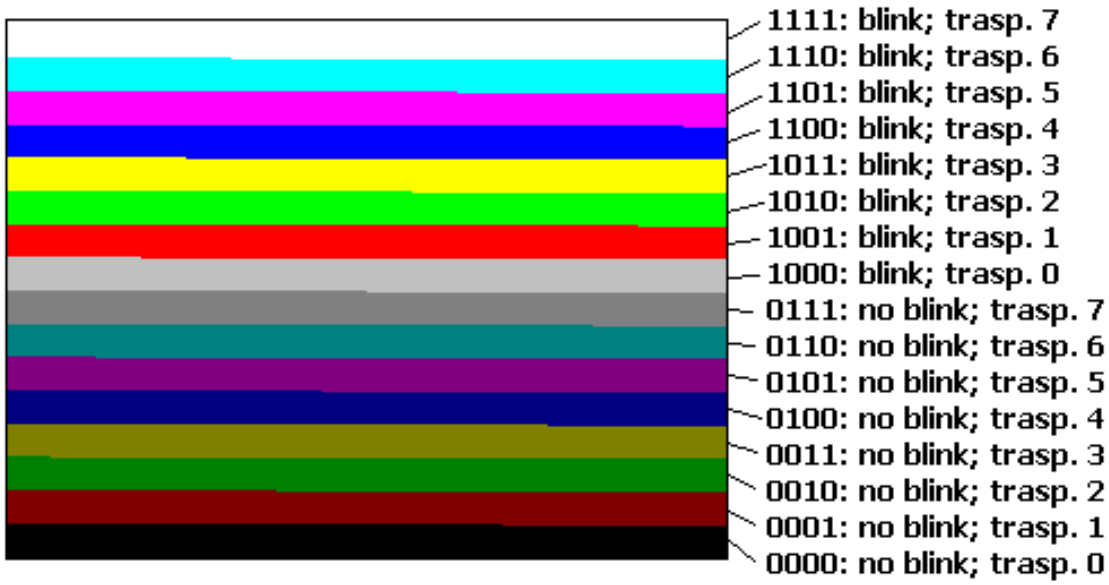


Figura 1. Color Palette per creare maschere OSD1 come Attribute Window

Un esempio di quanto detto è illustrato nella figura seguente, dove è mostrato sia il file grafico

`demo_osd0_640x480.r16`

che il corrispondente file di maschera per gli attributi OSD1

`demo_osd1_640x480.r4`

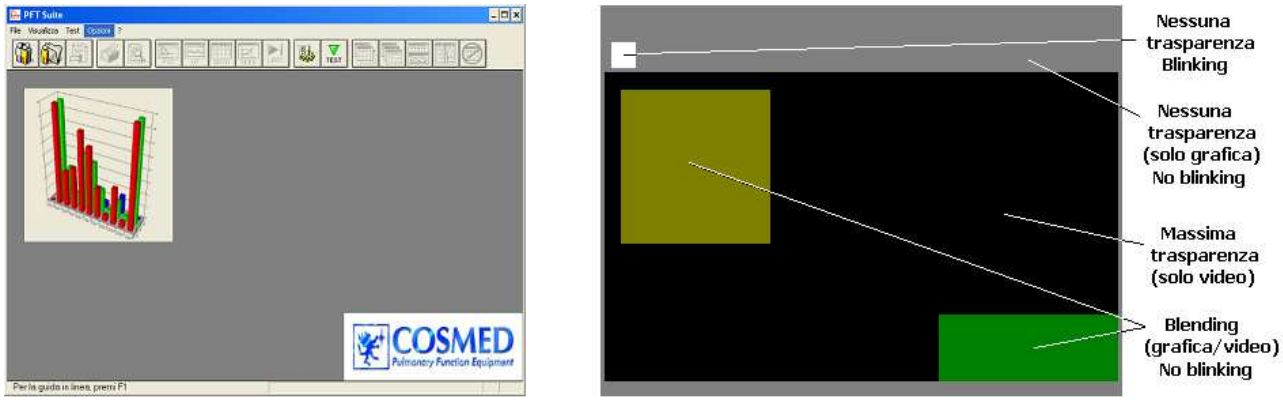


Figura 1. Esempio

Per visualizzare questa combinazione di grafica e video scegliamo le giuste opzioni di boot

```
target # setenv bootargs console=ttyS0,115200n8 noinitrd rw
ip=192.168.0.244 root=/dev/nfs
nfsroot=192.168.0.243:/home/gabriele/workdir/filesys,nolock mem=105M

target # setenv bootargs $(bootargs)
video=davincifb:vid0=720x576x16,2500K@0,0:vid1=720x576x16,2500K@0,0:osd0
=640x480x16,1200K@32,16:osd1=640x480x4,600K@32,16
davinci_enc_mngr.ch0_output=COMPOSITE
davinci_enc_mngr.ch0_mode=$(videostd)

target # boot
```

Dopo il boot spostiamoci in una cartella dove assumiamo presenti tutti i file necessari ed eseguiamo

```
target $ cp demo_osd0_640x480.r16 /dev/fb/0
target $ cp demo_osd1_640x480.r4 /dev/fb/2
```

Quindi mandiamo in esecuzione anche il video, p.es. con uno script DVTB

```
target $ ./loadmodules_sh.sh
target $ ./dvtb-r –s dvtb_dec_play_mpeg4.dvs
```

18 Appendice – Frame Buffer Console

Abilitando l'opzione di boot

CONFIG_FRAMEBUFFER_CONSOLE=y

è possibile ridirezionare lo standard output sul display LCD del target (layer OSD0), come visualizzare testo ASCII. Ad esempio

```
target $ cp /etc/fb.modes /dev/vc/0
```

oppure

```
target $ echo ciao pippo > /dev/vc/0
```

In questa modalità è attivo il savescreen, che oscura OSD0 dopo 5 min dal boot (in quanto non c'è tastiera o mouse a registrare attività). Per disabilitarlo:

```
target $ setterm -blank 0 > /dev/vc/0
```

Molti altri comandi possono essere usati:

```
setterm
  [ -term terminal_name ]
  [ -reset ]
  [ -initialize ]
  [ -cursor [on|off] ]
  [ -repeat [on|off] ]
  [ -appcursorkeys [on|off] ]
  [ -linewrap [on|off] ]
  [ -default ]
  [ -foreground black|blue|green|cyan|red|magenta|yellow|white|default ]
  [ -background black|blue|green|cyan|red|magenta|yellow|white|default ]
  [ -ulcolor black|grey|blue|green|cyan|red|magenta|yellow|white ]
  [ -ulcolor bright blue|green|cyan|red|magenta|yellow|white ]
  [ -hbcolor black|grey|blue|green|cyan|red|magenta|yellow|white ]
  [ -hbcolor bright blue|green|cyan|red|magenta|yellow|white ]
  [ -inversescreen [on|off] ]
  [ -bold [on|off] ]
  [ -half-bright [on|off] ]
  [ -blink [on|off] ]
  [ -reverse [on|off] ]
```

```
[ -underline [on|off] ]
[ -store ]
[ -clear [all|rest] ]
[ -tabs [ tab1 tab2 tab3 ... ] ]       (tabn = 1-160)
[ -clrtabs [ tab1 tab2 tab3 ... ] ]    (tabn = 1-160)
[ -regtabs [1-160] ]
[ -blank [0-60] ]
[ -dump   [1-NR_CONSOLES] ]
[ -append [1-NR_CONSOLES] ]
[ -file dumpfilename ]
[ -msg [on|off] ]
[ -msglevel [0-8] ]
[ -powersave [on|vsync|hsync|powerdown|off] ]
[ -powerdown [0-60] ]
[ -blength [0-2000] ]
[ -bfreq freqnumber ]
```

19 Riferimenti

- *LSP 1.20 DaVinci Linux EVM Installation User's Guide (SPRUFG0)*
- *LSP 1.20 DaVinci Linux NOR Flash Driver User's Guide.pdf (SPRUF10)*
- *TMS320DM644x DMSoC Universal Asynchronous Receiver/Transmitter (UART) User's Guide (SPRUE33)*
- *DVEVM Getting Started Guide (SPRUE66C)*
- *DVEVM Getting Started Guide (SPRUE66D)*
- *DVSDK Getting Started Guide (SPRUEG8)*
- *DaVinci DM6446 DVSDK 1.30 Release Notes*
- *Building GStreamer (SPRAAQ9)*
- *The TMS320DM642 Video Port Mini-Driver (SPRA918)*
- *Codec Engine Application Developer User's Guide (SPRUE67D)*
- *DSP/BIOS Driver Developer's Guide (SPRU616)*
- *TMS320C6000 DSP/BIOS Application Programming Interface (SPRU403)*
- *TMS320C6000 Chip Support Library API Reference Guide (SPRU401)*
- *TMS320C6000 Peripherals Reference Guide (SPRU190)*
- *TMS320C64x DSP Video Port/ VCXO Interpolated Control (VIC) Port Reference Guide (SPRU629).*
- *TMS320DM644x DMSoC ARM Subsystem Reference Guide (SPRUE14)*
- *TMS320DM644x DMSoC DDR2 Memory Controller User's Guide (SPRUE22c)*
- *Host USB Support on DVEVM (PRAAP8)*
- *Using Static IP Between Linux Host and DVEVM (SPRAAQ0)*
- *TMS320DM644x DMSoC Universal Asynchronous Receiver/Transmitter (UART) User's Guide (SPRUE33)*
- *TMS320DM644x DMSoC 64-Bit Timer User's Guide (SPRUE26)*
- *An Independent Analysis of the TEXAS INSTRUMENTS DIGITAL VIDEO EVALUATION MODULE (DVEVM) from BTDI*
- *Jeremiah Golston, White Paper "Reaping the Benefits of SoC Processors for Video Applications", from Texas Instruments*
- GNU Linker Manual
  http://www.gnu.org/software/binutils/manual/ld-2.9.1/ld.html
- GNU Binutils Manual
  http://www.gnu.org/software/binutils/manual/html_chapter/binutils_toc.html
- GNU gcc Manual

http://gcc.gnu.org/onlinedocs/gcc-3.4.6/gcc/
- Texas Instruments DaVinci™ Technology Developers Wiki: http://wiki.davincidsp.com/index.php?title=Main_Page&DCMP=OTC-DSP_DaVinciCRMOct07&HQS=Other+NL+davcrmoct07wiki
- *DaVinci EVM Home at Spectrum Digital:* http://c6000.spectrumdigital.com/davincievm/
- *TI Linux Community for DaVinci Processors (mailing list):* http://linux.davincidsp.com
- *Mailing list archives:* http://www.mail-archive.com/davinci-linux-open-source@linux.davincidsp.com/
- *Mailing list archives (text):* http://linux.omap.com/pipermail/davinci-linux-open-source/
- *info su sw from ti:* www.ti.com/digitalmediasoftware.
- *ARM Architecture:* http://infocenter.arm.com/help/index.jsp/
- *TI Software Updates:* http://focus.ti.com/dsp/docs/dspsplash.tsp?contentId=33169&DCMP=dm6446dvevm&HQS=EVM+OT+dvevmupdates
- *Montavista git kernel open source:* http://source.mvista.com/git/?p=linux-davinci-2.6.git;a=summary
- *IETF RFCs:* http://www.ietf.org/rfc.html
- *http://www.linux-mtd.infradead.org/*
- *Open Sound System (OSS) website: http://www.opensound.com*
- *Video for Linux 2 (v4l2) website: http://linux.bytesex.org/v4l2/*
- *FBdev website: http://linux-fbdev.sourceforge.net*
- *Programming with POSIX Threads, David R. Butenhof [ISBN 0201633922]*
- *Linux Device Drivers 3rd Edition, J. Corbet & A. Rubini [ISBN 0-596-00590-3]*