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.

Il presente documento contiene le specifiche tecniche della parte software del solo Modulo Digitale. L'hardware (descritto nel documento IPvphone_Specifiche di Progetto - Part I) è basato sul TMS320DM6441 "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 DM6441 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, ecc.

3 Definizione degli ambienti di sviluppo

3.1 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.

3.2 Opzioni Target Kernel Linux

- MontaVista™ Pro Linux 4.0 per DaVinci (LSP con kernel 2.6.10 e drivers per DM6441)
- 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

3.3 Target Bootloader

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. 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.

- 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)

---

[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)

3.4 Tools e Ambienti di Sviluppo

3.4.1 Sviluppo applicazioni ARM

http://wiki.davincidsp.com/index.php?title=Linux_Toolchain
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 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 lo sviluppo di una applicazione ARM che usa codec che girano sul DSP (remote codec) occorre avere/consultare:

- Codec packages
- A Codec Server DSP executable
- An Engine config file (.cfg)
- Codec Engine API Reference (CE_INSTALL_DIR/docs/html/index.html)
- Example Build and Run Instructions (CE_INSTALL_DIR/examples/build_instructions.html)

Il lavoro consiste nello scrivere il codice applicativo, generare i file .c e .xdl (linker command file) a partire dal file .cfg (remote.cfg) e usando i tools XDC, compilare e linkare (makefile.xdc). Il risultato è l'applicazione eseguibile.

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)

3.4.2 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

Per lo sviluppo di un codec DSP occorre avere/consultare:

- Codec Engine Algorithm Creator User's Guide (SPRUED6)
- Codec Engine SPI Reference Guide (CE_INSTALL_DIR/docs/spi/html/index.html)
- xDAIS-DM (Digital Media) User Guide (SPRUEC8)
- xDM API Reference (XDAIS_INSTALL_DIR/docs/html/index.html)
- TMS320 DSP Algorithm Standard Rules and Guidelines (SPRU352)

- TMS320 DSP Algorithm Standard API Reference (SPRU360)
- TMS320 DSP Algorithm Standard Developer's Guide (SPRU424)
- Example codecs
- Codec Engine Server Integrator's Guide (SPRUED5)
- Configuration Reference Guide (CE_INSTALL_DIR/xdoc/index.html)
- Example Codec Servers
- Chapter 5 of SPRUE67D, Integrating an Engine
- Example Build and Run Instructions (CE_INSTALL_DIR/examples/build_instructions.html)
- Example configuration scripts (*.cfg)

3.5 DVSDK

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

Nota: Per scaricare i componenti SW aggiornati del DVEVM e del DVSDK andare sul sito TI protetto
https://www-a.ti.com/extranet/cm/product/dvevmsw/dspswext/general/homepage.shtml (e-mail: gabrielef@cosmed.it; password: Porchetta1)

3.5.1 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

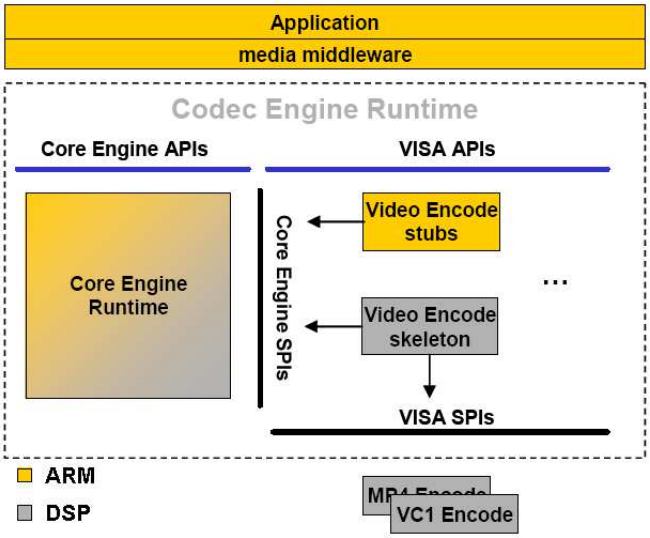
3.5.2 TI TMS320DM644x SoC Analyzer

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

3.5.3 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 DM6441 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.

3.5.4 Versioni Aggiornate a marzo 2008

- DSP Link 1.50 (1.30 in DVEVM 1.20)
- DSP/BIOS 5.32.01 (5.31.1 in DVSDK 1.20; 5.31.8.15 in DVSDK 1.30)
- C6x Codegen tools 6.0.16 (6.0.11.1 in DVSDK 1.20; 6.0.15.1 in DVSDK 1.30)
- Framework Components 2.10
- xDAIS 6.10 (5.10 in DVEVM 1.20; 6.0 in DVEVM 1.30)
- CMEM 2.10 (1.02 in DVEVM 1.20; 2.0 in DVEVM 1.30)
- XDC Tools 3.00.06 (2.94 in DVSDK 1.20; 3.0.2.14 in DVSDK 1.30)
- Codec Engine 2.10 (1.10 in DVEVM 1.20; 2.0 in DVEVM 1.30)
- DM64x SoC Analyzer (1.0.0.16 in DVSDK 1.20; 1.2.0.9.1 in DVSDK 1.30)

Il Codec Engine 2.10 con il DVSDK versione 1.30 o precedenti.

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

cioè

- IALG - xDAIS Algorithm Interface This is the xDAIS IALG interface.

- IDMA3 (C64P) The IDMA3 interface enables algorithms to request and receive handles representing private logical DMA resources.
- ACPY3 (C64P) The ACPY3 module provides a comprehensive list of DMA operations an algorithm can perform on logical DMA channels it acquires through the IDMA3 protocol. Example of ACPY3 operations include channel configuration, DMA transfer scheduling, and DMA transfer synchronization.
- DMAN3 (C64P) DMAN3 is an application interface to the 3rd generation DMA Manager. It provides routines for granting and reclaiming DMA resources used by algorithms or other non-algorithm users. This module is used by frameworks or applications which have XDAIS algorithms that implement the IDMA3 interface.
- DSKT2 The XDAIS Socket Library provides services to support the creation, initialization, control, and deletion of XDAIS algorithm instance objects.
- DBC
- xDAIS Types and Constants

5 Moduli Software

I seguenti parametri illustrano i moduli SW del sistema IPvPhone.

5.1 Introduzione

Il DM6441 contiene al suo interno un processore general purpose ARM9 e un DSP C64x+. Nel IPvPhone L'ARM9 è l'"application processor" sul quale gira il sistema operativo Linux, l'interfaccia utente e le routine di controllo generale del sistema, mentre il DSP e il modulo acceleratore video sono visti come dei Server che eseguono i task intensivi di elaborazione, come i codec.
L'applicazione ARM9 è multi-threaded (utilizza POSIX Pthreads), scritta in C e basata su Linux.
L'applicazione DSP è single-threaded, scritta in C e basata su DSP/BIOS.
Le chiamate dall'ARM al DSP sono tutte eseguite da un unico linux thread, e questo thread è fermo finchè il DSP non ha eseguito l'ultima chiamata.

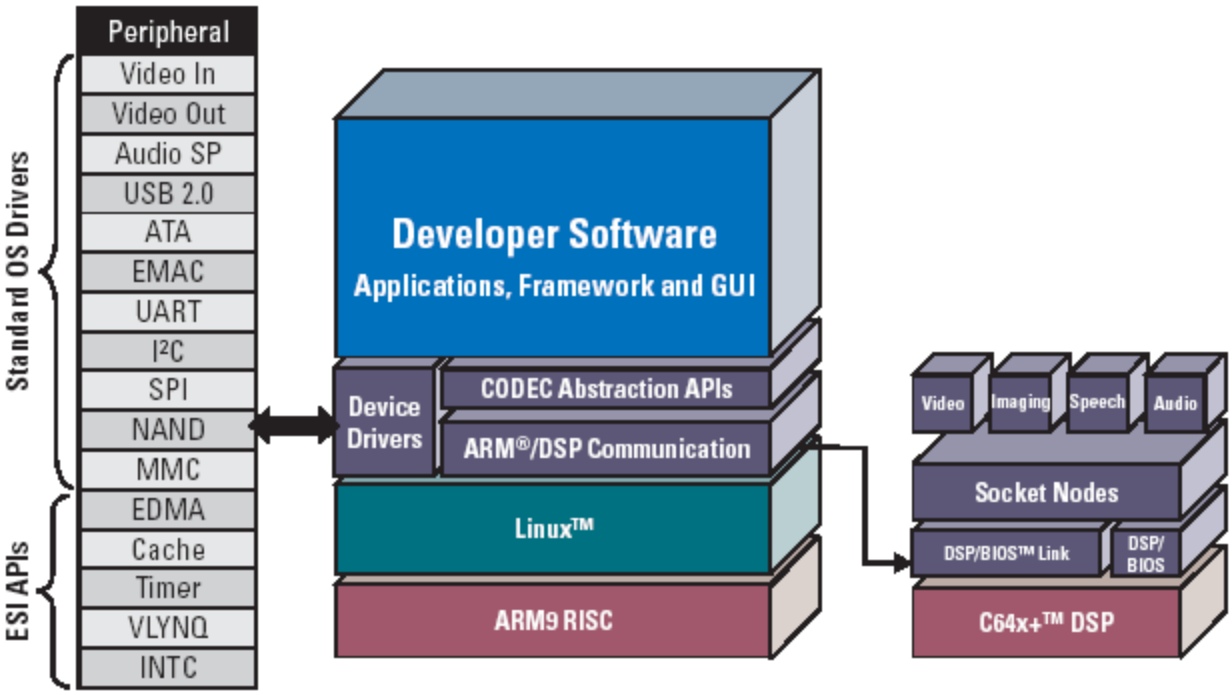


Figura 1. Piattaforma Software del Modulo Digitale IPvPhone

L'applicazione DSP consiste nei seguenti moduli:

- Codifica video CIF (352 x 288) @30 fps
  - H.263 BP (Baseline Profile)
  - H.264/MPEG-4 SP (Simple Profile)
- Decodifica video CIF (352 x 288) @30 fps
  - H.263 BP (Baseline Profile)
  - H.264/MPEG-4 SP (Simple Profile)
- Codifica audio
  - G.711 (uncompressed)
  - AAC (audio format from MPEG-4)
  - MP3 (MPEG-1 layer 3)
  - WMA (Windows Audio)
  - G.723.1
  - G.729
- Decodifica audio
  - G.711
  - AAC
  - MP3
  - WMA
  - G.723.1
  - G.729

- Line Echo Cancellation, ITU-T G.165
- Voice Activity Detection (VAD), and Comfort Noise Generation (CNG)
- Lettura CID (Caller ID)
- Videoconferenza a 5 vie H.263 o 6 vie H.264/MPEG-4

L'applicazione ARM9 consiste nei seguenti moduli

- kernel Linux v2.6, con device drivers per periferiche DM6441 e stack TCP/IP
- protocolli di rete RTP/RTCP, PPP, HTTP, TFTP, SMTP, Telnet, DNS, DHCP
- device drivers per periferiche esterne (controllore Smart Card, I2C Expander, ExtPLL, ecc.)
- shell Linux
- gestione del sistema nelle varie modalità operative
- interfaccia grafica utente (GUI) e OSD
- gestione dell'aggiornamento firmware da remoto
- gestione della segnalazione di chiamata A/V su IP (H.323 o SIP)
- inizializzazione, comunicazione e controllo DSP

In generale esiste una separazione netta tra l'applicazione ARM-Linux e l'applicazione DSP, in quanto l'ARM si limita ad allocare i buffers di I/O e a chiamare le funzioni DSP attraverso delle API VISA (*Application Program Interface for Video/Imaging/Speech/Audio*). Queste API sono raggruppate in VIDENC, VIDDEC, IMGENC, IMGDEC, SPHENC, SPHDEC, AUDENC e AUDDEC. Così, ad esempio, il thread di decodifica H.264 conterrà le 4 API: VIDDEC_CREATE, VIDDEC_DELETE, VIDDEC_PROCESS e VIDDEC_CONTROL. Il vantaggio è che, ad esempio, per cambiare un codec da MP3 a AAC non devo modificare il sorgente applicativo ma solo la configurazione.

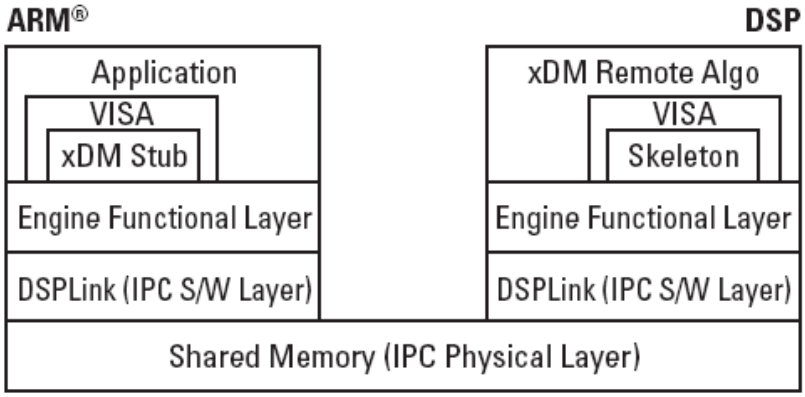


Figura 1. Schema semplificato del framework CE

Nel suo Kernel space, Linux prevede già delle API standard per l'I/O dei file A/V (FBdev/DirectFB, V4L2, OSS/ALSA, ecc.) ma le API EPSI (*Easy Peripheral Software Interface*) ne espandono le potenzialità. Ad esempio, l'interfaccia standard di video capture di Linux è V4L2, e supporta già i modi BT.656 e Y/C, ma non supporta le funzioni resizer e istogramma; FBdev/DirectFB è l'interfaccia per il VPBE (video display), ma anche qui molte funzioni non sono implementate.

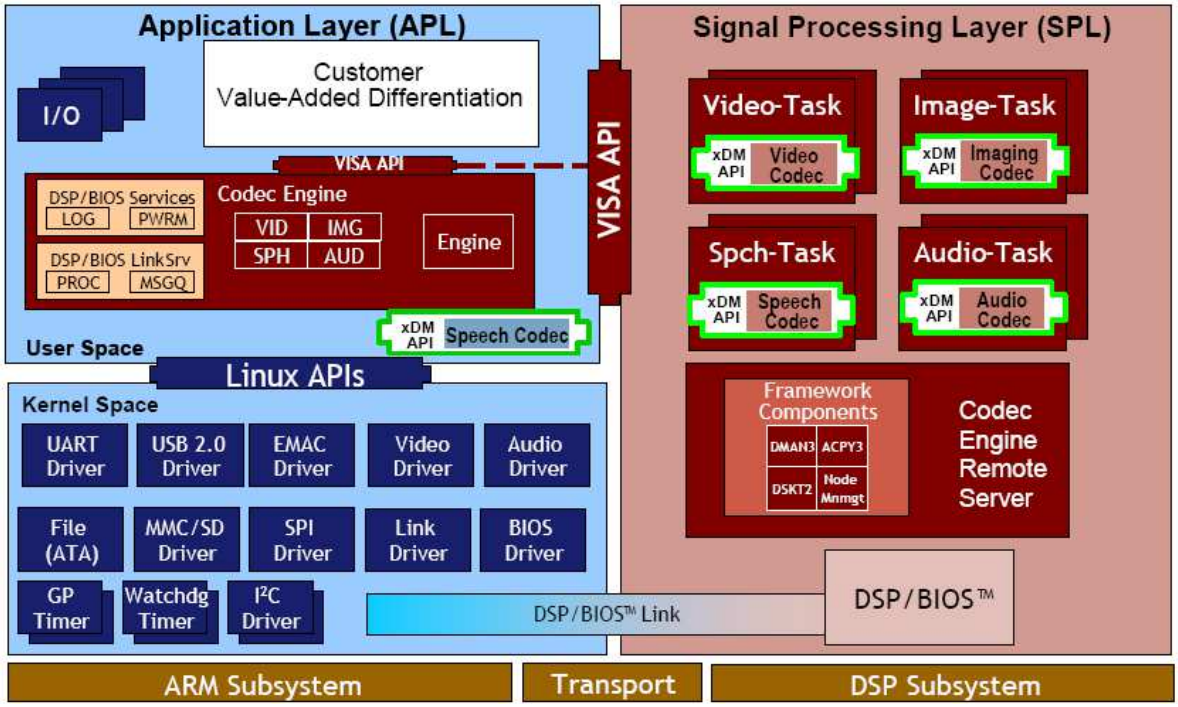


Figura 1. Piattaforma Software del Modulo Digitale IPvPhone

5.2 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 480 x 2) = 691200 bytes = 675 KB
frame video codificato -> 675 KB
frame audio raw da codificare -> 4 KB
frame audio codificato -> 4 KB
frame video da decodificare -> 675 KB
frame video decodificato -> 675 KB
frame audio da decodificare -> 4 KB
frame audio decodificato -> 4 KB

Quindi, prima che l'applicazione parta, allocheremo 3 MB di CMEM nel seguente modo

```
insmod cmemk.ko phys_start=0x88000000 phys_end=0x88300000 pools=4x691200,4x4096
```

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().

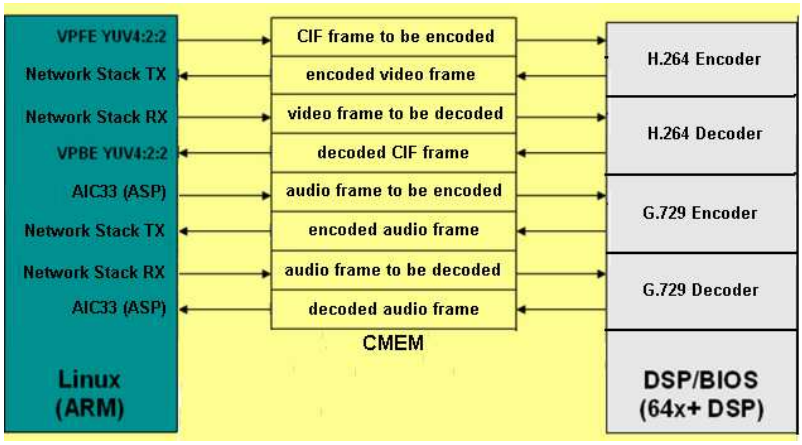


Figura 1. CMEM per videochiamata (esempio)

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.
La seguente figura mostra un esempio del risultato ottenibile su un banco da 128 MB

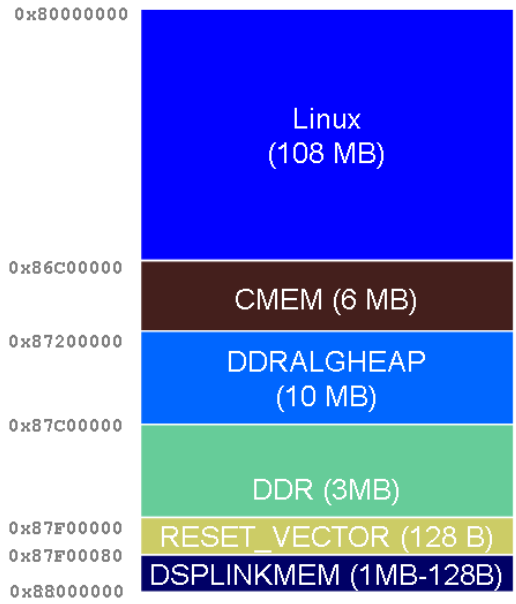


Figura 1. Memory Map 128 MB DDR2 (esempio)

Quello che resta da fare dipende dalla versione del CE. Fino a Codec Engine 1.20 occorre il rebuild del DSP Link. Per Codec Engine 1.20 o successivo, che usa DSP Link version 1.40, basta il rebuild dell'applicazione ARM.

Poi bisogna eseguire il build del DSP Server. Il DSP Server ha un file *.tcf* che specifica il layout della memoria (da editare), e un file *.cfg* di configurazione del Codec Engine che elenca i codec necessari (gli altri vanno tolti). Per il build editare

```
make clean
make
```

Copiare il file di output nel target file system. Il passo successivo è eseguire il build dell'applicazione ARM (solo per CE 1.20 o successivo), dopo aver editato la mappa di memoria in *ceapp.cfg*. Copiare nel target file system app.out e tutti gli altri file e moduli necessari (sample input file, cmemk.ko, dsplinkk.ko, loadmodules.sh, ecc).
Successivamente bisogna aggiornare la variabile di configurazione del bootloader. Nell'esempio,

```
> setenv bootargs 'console=ttyS0,115200n8 root=/dev/nfs mem=108M
nfsroot=192.168.1.101:/opt/montavista/pro/devkit/arm/v5t_le/target,nolock'
```

Dopo il reboot della scheda lanciare l'applicativo

```
sh loadmodules.sh
./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.

5.3 I/O Buffers

Uno dei compiti del codice applicativo è quello di allocare e deallocare i buffer di l'I/O per ciascun codec. Questi buffer devono essere contigui (non frammentati) e cache-aligned. Se la linea di cache L2 è da 128 byte, allora il buffer deve iniziare a un indirizzo multiplo di 128, e anche la lunghezza deve essere un multiplo di 128.

5.4 Flash

In Flash sono presenti il seguenti file:

- Bootloader (u-boot)
- Parametri di u-boot
- Linux kernel con drivers built-in per il DM644x (uImage)
- JFFS2 File system contenente la shell, l'applicazione IPvPhone, utility di supporto run-time

La seguente tabella mostra la mappa fisica di memoria Flash del Modulo Digitale.

<table>
<tr><th>Sector</th><th>Start Address</th><th>File</th><th>Descrizione</th><th>Name</th></tr>
<tr><td>0</td><td>0x02000000</td><td>u-boot.bin (max 0x20000)</td><td>bootloader</td><td>mtd0 / mtdblock0</td></tr>
<tr><td>1</td><td>0x02020000</td><td>Params (max 0x20000)</td><td>Parametri di u-boot</td><td>mtd1 / mtdblock1</td></tr>
<tr><td>2</td><td>0x02040000</td><td>Spare sector (max 0x20000)</td><td>Not used</td><td>mtd2 / mtdblock2</td></tr>
<tr><td>3</td><td>0x02060000</td><td rowspan="3">uImage (max 0x200000)</td><td rowspan="3">Kernel</td><td rowspan="3">mtd3 / mtdblock3</td></tr>
<tr><td>…</td><td>….</td></tr>
<tr><td>18</td><td>0x02240000</td></tr>
<tr><td>19</td><td>0x02260000</td><td rowspan="3">rootfs.jffs2 (max 0x1DA0000)</td><td rowspan="3">File system</td><td rowspan="3">mtd4 / mtdblock4</td></tr>
<tr><td>…</td><td>…</td></tr>
<tr><td>255</td><td></td></tr>
</table>

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

5.5 File System

Prima del boot del kernel, u-boot può caricare un initial RAM disk dalla NOR flash (dove ne esiste una copia) alla DDR2. Il file system presente nel RAM disk viene chiamato initial RAM disk file system, o initrd. Questo file system può essere montato come root file system e l'applicazione può essere eseguita da qui. Il file system rappresenta la memoria locale del kernel. Dato che initrd è installato su una memoria volatile, il contenuto viene perso allo spegnimento della macchina. Se si desidera salvare dei parametri generati runtime, abbiamo bisogno di un file system nella flash. Esistono diversi file systems di questo tipo, come il Journalling Flash File System (JFFS2) e il Yet Another Flash File System (YAFFS2).
Il root filesystem di Linux contiene files ed eseguibili che servono al kernel e al system administration. Il kernel monta una partizione di memoria fisica sulla directory /. Sotto / esistono le seguenti directory:

- /bin, /sbin – contengono eseguibili di sistema come init, ifconfig, mount, cd, mkdir, ping
- /lib – contiene librerie condivise (libc, ecc.) e il loader dinamico di Linux
- /etc – contiene scripts e files di configurazione di sistema
- /dev – contiene files per dispositivi specifici
- /usr – programmi aggiuntivi e librerie
- /var – contiene files temporanei, come log files, print, mail-queues, ecc.
- /home – contiene dati personali dell'utente
- /opt – contiene software di terze parti

6 Installazione SW - base

Leggere il documento relativo

7 Approfondimenti sull'installazione SW base

7.1.1 Modificare DVFlasher/UBL in funzione del target

1) Modificare il file `/home/user/workdir/DVFlasher/ubl/dm644x.c` in base al target, in particolare la parte DDR2 (cfr. IPvphone_Specifiche di Progetto – Part I). Il sorgente originale gestisce DDR tipo Micron MT47H64M16BT-37E, mentre il Modulo Digitale può montare:

   MT47H32M16BN-5E (default)
   MT47H16M16BG-5E
   MT47H64M16HR-5E

2) assicurarsi di avere il compilatore ARM nella PATH

```
/opt/mv_pro_4.0/montavista/pro/devkit/arm/v5t_le/bin
```

3) Ricompilare sotto Linux, da /home/user/workdir/DVFlasher. I makefile del progetto possono essere 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

4) Se il target monta DM6441 allora

```
host $ make DM6441
```

7.1.2 Re-build di u-boot

5) Log in come user
6) copiare la cartella dei sorgenti u-boot-1.1.4 in /home/user/workdir

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

7) creare una nuova board:

- nella sottocartella /board creare una copia della cartella davinci e rinominarla ipvp
- rinominare /board/ipvp/davinci.c in /board/ipvp/ipvp.c
- modificare in /board/ipvp/ i file .c e .S in funzione dell'hardware della scheda IPvPhone

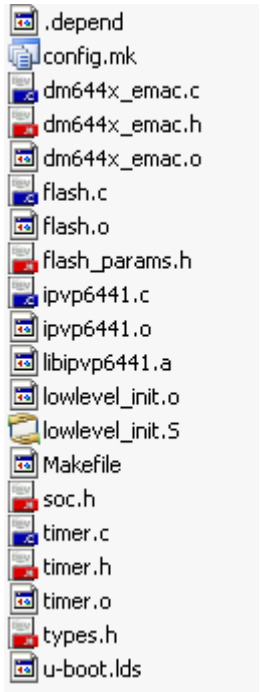


Figura 1. u-boot - directory della board IPvPhone (ipvp)

- modificare il top Makefile aggiungendo sotto alla riga relativa alla scheda davinci la riga

```
ipvp_config :    unconfig
   @./mkconfig $(@:_config=) arm arm926ejs ipvp
```

- creare /include/configs/ipvp.h partendo da una copia di /include/configs/davinci.h
- apportare a ipvp.h tutte le modifiche necessarie, in particolare definire la costante `CFG_IPVP`
- in /cpu/arm926ejs/start.S modificare la riga

  ```
  #ifdef CFG_DAVINCI
  ```

  in

  ```
  #if defined (CFG_DAVINCI) || defined (CFG_IPVP)
  ```

8) ricompilare

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

9) 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. 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.

7.1.3 Re-build del kernel (linux-2.6.10_mvl401)

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

```
host $ cd /home/user/
host $ mkdir -p workdir/lsp
host $ cd workdir/lsp
host $ cp -R /opt/mv_pro_4.0.1/montavista/pro/devkit/lsp/ti-davinci_evm-
arm_arm_v5t_le/linux-2.6.10_mvl401 .
host $ mv linux-2.6.10_mvl401 ti-davinci
host $ cd ti-davinci
```

Nota: in SPRUE66D (pag.4-14) viene copiato invece il contenuto di
`/opt/mv_pro_4.0.1/montavista/pro/devkit/lsp/ti-davinci`

12) 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
```

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
```

Nota: Si può partire da una copia di davinci_dm644x_defconfig

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
```

13) Per modificare il nome dell'immagine creata è sufficiente modificare il file

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

Per la scheda IPvPhone il file conterrà la stringa "*-davinci_ipvp*", sicchè il nome dell'immagine sarà

Linux-2.6.10_mvl401-davinci_ipvp

14) per abilitare il Linux Trace per il SoC Analyzer.

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

- set General Setup->Linux Trace Toolkit Support as "built-in"
- set Device Drivers->Filesystems->Pseudo Filesystems->Relayfs file system support to "built-in"
- save your changes and exit the configuration screen

15) per ricompilare, sempre come user,

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

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`.

16) se si é configurato il kernel con almeno un modulo (p.es. avendo usato davinci_dm644x_defconfig)

```
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
```

17) copiare l'immagine nella cartella dove U-Boot possa utilizzare il TFTP per il download

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

7.1.4 build di un Initial RAM Disk File System (initrd)

18) assicurarsi di avere i seguenti componenti SW in

```
/opt/mv_pro_4.0/montavista/pro/devkit/arm/v5t_le/packages/pro
```

(oppure ../pro/optional)

- busybox 1.00r3-5.0.0 Combines small versions of many common Linux utilities
- initscript 2.85-3.0.0 Contains basic system script used to boot the system
- netbase 4.17-1.0.1 Provides necessary infrastructure for TCP/IP networking
- thttpd 2.25b-1.0.0 Contains a small, fast, and secure web server, including CGI support, URL traffic based throttling and basic authentication.

19) Per risparmiare tempo, utilizzare il RAM disk pronto

```
/opt/mv_pro_4.0/montavista/pro/devkit/arm/v5t_le/images/ramdisk.gz
```

Esso, una volta decompresso, occuperà circa 6.3MB in DDR.

Continuare a pag 7 di SPRAAH2

1.1 Creare il supporto alla board IPVP nel Montavista LSP

Per creare nel LSP una configurazione del kernel specifica per la scheda IPvPhone occorre definire/usare le costanti MACH_DAVINCI_IPVP e CONFIG_MACH_DAVINCI_IPVP in tutti i punti in cui sono già definite/usate le costanti MACH_DAVINCI_EVM e CONFIG_MACH_DAVINCI_EVM. Le direttive di compilazione #ifdef che fanno riferimento alle costanti EVM devono essere opportunamente modificate con && o || in modo da comprendere anche il test sulle costanti IPVP.
I file da modificare sono

```
/home/user/workdir/lsp/ti-davinci/include/asm-arm/mach-types.h
/home/user/workdir/lsp/ti-davinci/arch/arm/tools/mach-types
/home/user/workdir/lsp/ti-davinci/arch/arm/mach-davinci/Kconfig
/home/user/workdir/lsp/ti-davinci/arch/arm/Kconfig
/home/user/workdir/lsp/ti-davinci/driver/net/Kconfig
/home/user/workdir/lsp/ti-davinci/driver/char/Kconfig
/home/user/workdir/lsp/ti-davinci/driver/usb/musb/davinci.c
/home/user/workdir/lsp/ti-davinci/arch/arm/mach-davinci/Makefile
```

Inoltre devono essere creati i file

```
/home/user/workdir/lsp/ti-davinci/arch/arm/mach-davinci/board-ipvp.c
/home/user/workdir/lsp/ti-davinci/arch/arm/mach-davinci/leds-ipvp.c
```

Sulla falsariga di quelli già presenti specifici per la EVM.
Inoltre devono essere creati ex novo i file

```
/home/user/workdir/lsp/ti-davinci/arch/arm/mach-davinci/gpio-ipvp.c
/home/user/workdir/lsp/ti-davinci/arch/arm/mach-davinci/gpio-ipvp.h
/home/user/workdir/lsp/ti-davinci/arch/arm/mach-davinci/lateinit-ipvp.c
```

La creazione di questi nuovi file comporta anche ovviamente la modifica del Makefile

1.2 Creare un driver

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”.

20) dopo aver modificato un modulo occorre compilarlo

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

21) 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 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`
   Verrà aggiunta la riga
   `CONFIG_IPVP=m`

c) Compilare

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

d) Copiare nel filesystem

   ```
   host $ cp /home/user/workdir/lsp/ti-
   davinci/drivers/ipvp/ipvp_core.ko
   /home/user/workdir/filesys/lib/modules/2.6.10_mvl401-
   davinci_ipvp/kernel/drivers/ipvp/ipvp_core.ko
   ```

8 Installazione SW – componenti aggiuntivi

Leggere il documento relativo

1.3 Installazione componenti aggiuntivi DVSDK

Il pacchetto contiene

- Framework Components 1.10.04 - xDAIS utility library for instantiating xDAIS algorithms, DMA Manager library, High-Performance Functional DMA Interface, and framework utilities
- DVTB 1.01 - (Digital Video Test Bench) A package containing a set of tools to facilitate verification and validation of the product

Attenzione: Il DVSDK deve essere installato dopo aver installato I pacchetti DVEVM aventi la stessa versione!

1.4 Installazione componenti GStreamer

La libreria GStreamer permette di astrarre la manipolazione dei media con il concetto di pipeline, semplificando la programmazione al livello di applicazione. Consiste di libreire di core e librerie Plug-Ins.

Il pacchetto contiene componenti open source (gstreamer_opensource.tar.gz)

- glib-2.12.4 (http://ftp.gnome.org/pub/GNOME/sources/glib/2.12/)
- check-0.9.3 (http://sourceforge.net/project/showfiles.php?group_id=28255&package_id=20116)
- liboil-0.3.9 (http://liboil.freedesktop.org/download/)
- libxml2-2.6.16 (http://ftp.gnome.org/pub/GNOME/sources/libxml2/2.6/)
- gstreamer-0.10.1 (http://gstreamer.freedesktop.org/src/)
- gst-plugins-base-0.10.11
- mpegts_demux
- OSS (open sound system) audio sink driver

e component TI (gstreamer_tibuild.tar.gz)

- fbvideosink (Framebuffer video sink)
- gdecoder (Video Decoder)
- adecoder (Audio Decoder)
- tiavidemux (AVI Demuxer)
- tiasfdemux (ASF Demuxer)

8.1.1 Build e installazione delle librerie

22) unzip gstreamer_opensource.tar.gz e gstreamer_tibuild.tar.gz in /home/user/workdir/gstreamer.
Ciò crea le directory `ti_build/` e `opensource_build/`

23) aggiungere i tool di cross-compilazione MV in PATH:

```
host $ export PATH=$PATH:/opt/MontaVista/pro/devkit/arm/v5t_le/bin/
```

24) creare sul target file system le directory

```
target $ mkdir /home/user/workdir/filesys/opt/gstreamer
target $ mkdir /home/user/workdir/filesys/opt/system_files_gstreamer
```

25) creare i link simbolici

```
host $ ln -s /opt/dvevm/mv_pro_4.0/montavista/opt/montavista
host $ ln -s /bin/sed/opt/montavista/common/bin/sed
```

26) build

```
host $ cd /home/user/workdir/gstreamer/opensource_build
host $ ./make_opensource.sh
host $ cd /home/user/workdir/gstreamer/ti _build
host $ ./make_tiplugins.sh
```

27) per lanciare la demo dulla DVEVM seguire istruzioni a pag.5 di SPRAAQ9.

9 Installazione git kernel

9.1.1 Re-build del kernel git linux-2.6.25

Per installare la verisione git del kernel (2.6.25-davinci1)

28) Log in come user

29) da http://source.mvista.com/git/?p=linux-davinci-2.6.git;a=shortlog;h=refs/tags/v2.6.25-davinci1 selezionare **snapshot** all'estrema destra della pagina, in corrispondenza dell'immagine

30) scaricare, copiare in /home/user/workdir/lsp ed estrarre

```
host $ tar xvfz linux-davinci-2.6.git-
7c2d65c9a06f7faa11e5b1659c6800bf0a030cce.tar.gz
```

31) rinominare

```
host $ mv linux-davinci-2.6 linux-davinci-2.6.25
host $ cd linux-davinci-2.6.25
```

32) configurazione e ricompilazione di prova

```
host $ make clean
host $ make davinci_evm_dm644x_defconfig
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
```

Nota: compilando la versione git 2.6.25 per davinci può capitare il seguente errore

```
>   CC      arch/arm/kernel/asm-offsets.s
> arch/arm/kernel/asm-offsets.c:1: error: invalid ABI option:
> -mabi=aapcs-linux
```

In tal caso disabilitare l'opzione CONFIG_AEABI tramite menuconfig

9.1.2 Installare e utilizzare quilt

Per poter tenere traccia delle modifiche da apportare al kernel è consigliabile utilizzare il sistema di gestione delle patch. *quilt* è un programma che permette questo in maniera più semplice e automatizzata rispetto al tradizionale metodo con *diff* e *patch*.

33) Copiare ed estrarre i sorgenti

```
host $ cp /home/gabriele/dvsdk_#_##_##_##/quilt/quilt401.tar.gz .
host $ tar zxf quilt401.tar.gz
```

34) Verificare che l'eseguibile quilt sia quello appena installato

```
host $ which quilt
```

in caso contrario aggiungere in /home/user/.bashrc

```
PATH="/home/user/workdir/lsp/linux-davinci-
2.6.25/montavista/pro/bin/:$PATH"
Export PATH
```

35) Questo sorgente fa riferimento all'eseguibile diff presente in /opt/dm644x/mvl_pro_4.0.1/montavista/… Sicchè alcuni warnings compariranno usando alcuni comandi quilt. E' quindi necessario modificare la path presente in alcuni file sotto montavista/ in modo tale che tutto funzioni correttamente

36) Creare una patch

```
host $ quilt new patch1
```

37) Aggiungere il primo file (il top Makefile del LSP)

```
host $ quilt add Makefile
```

38) Cambiare la costante EXTRAVERSION in Makefile

```
host $ vim Makefile
…
```

39) Registrare la modifica nella patch

```
host $ quilt refresh
```

40) verificare

```
host $ cat patches/patch1
```

41) Aggiungere alla patch tutti gli altri file che dovranno essere modificati o creati ex novo

```
host $ quilt add arch/arm/tools/mach-types
host $ quilt add arch/arm/mach-davinci/Kconfig
host $ quilt add arch/arm/mach-davinci/board-ipvp.c
host $ quilt add arch/arm/mach-davinci/Makefile
host $ quilt add arch/arm/mach-davinci/psc.c
host $ quilt add include/asm-arm/mach-types.h
host $ quilt add arch/arm/configs/davinci_ipvp_defconfig
…
```

42) Applicare tutte le modifiche ai suddetti files (ovviamente creare quelli che vanno creati ex novo)

43) Registrare le modifiche nella patch

```
host $ quilt refresh
```

44) Per tornare al punto di partenza basta togliere tutte le patch (per ora abbiamo solo patch1)

```
host $ quilt pop -a
```

45) Per applicare nuovamente

```
host $ quilt push -f
```

Altre patch possono essere create (new), applicate (push) e tolte (pop), come su uno stack.

46) Per applicare una patch ottenuta da fonti esterne (sia `nuova_patch`), rinominandola ad esempio `patch2`

```
host $ quilt import nuova_patch -n patch2
host $ quilt push patch2
```

9.1.3 Re-build del kernel git per IPvPhone (linux-2.6.25-rc1)

Dopo aver fatto le modifiche ai sorgenti originali del kernel e averle registrate nelle patch

47) riconfigurare il kernel

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

48) ricompilare

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

49) Compilare i moduli

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

50) Installare i moduli nel file system

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

Se si parte dalla versione open source occorre copiare prima le cartelle /montavista (per quilt) e /patches, poi eseguire

```
host $ quilt push -f
host $ quilt refresh
```

e infine configurare e ricompilare.

1.1 Creare uno script bash che viene lanciato allo startup del sistema

51) Sia myrootfs il root filesystem

```
host $ cd /home/Gabriele/workdir/myrootfs
host $ sudo touch etc/init.d/hello
host $ sudo vim etc/init.d/hello.sh
```

52) Aggiungere le seguenti due righe a hello.sh

```
   #!/bin/bash
   echo "Hello World!!!"

host $ sudo chmod +x etc/init.d/hello.sh
host $ cd etc/rc.d/rcS.d
host $ sudo ln -s ../init.d/hello.sh S50hello
```

1.1 Creare un programma che accede al I2C I/O Expander tramite Sysfs

53) Log in con un user account sul NFS

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

54) Creare un file toggle_io.c tipo

```
/*****************************************************************
* TOGGLE_IO.C
* Gabriele Filosofi 2008 - ADFL Consulting s.r.l.
* Simple program to toggle on and off a port of the I2C I/O expander
* on the IPvPhone board (U13)
*****************************************************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <signal.h>

#define I2C_DEVICE      "/dev/i2c/1"
#define I2C_DEV_ADDR    0x20
#define I2C_SLAVE       0x0706

int main()
{
      int i2c_fd;
      int state = 0;
      unsigned char val = 0x00;
      int ret;
      int i = 0;
      unsigned int b = 0;

      struct timespec req = {
            .tv_sec = 0,
            .tv_nsec = 100000000, //1 s
      };

      if((i2c_fd = open(I2C_DEVICE,O_RDWR)) < 0) {
            printf("Unable to open the i2c port!\n");
```

```
        exit(1);
    }

    if(ioctl(i2c_fd, I2C_SLAVE, I2C_DEV_ADDR)== -1) {
        close( i2c_fd );
        printf("Unable to setup the i2c port!\n");
        exit(1);
    }

    read( i2c_fd, &val, 1);
    printf("I2C I/O expander val = 0x%02x\n", val);
    printf("toggling BCLK_SEL ...\n");
    do {
        if(b==0)
        {
            val = 0xFF;
            b = 1;
        }
        else
        {
            val = 0xF7;
            b = 0;
        }
        write( i2c_fd, &val, 1);
    } while (!nanosleep (&req, NULL));
    close ( i2c_fd );
    printf ("End.\n");
    return 0;
```

55) compilare

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

56) dalla target window di linux

```
target $ cd /opt/toggle_io
target $ ./toggle_io
```

57) Verificando con un oscilloscopio, lo stato della linea BCLK_SEL (pin P3 di U13) dovrebbe cambiare periodicamente

1.2 Accedere a una USB Memory Stick

58) Assicurarsi di aver attivato come built-in le seguenti opzioni del kernel tramite

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

- Device Drivers->Block devices->Low Performance USB Block driver CONFIG_BLK_DEV_UB=y
- File Systems->DOS/FAT/NT Filesystems->VFAT Support CONFIG_VFAT_FS=y

59) Dopo il boot, inserire sul target la memory stick sulla porta USB. Sulla shell dovrebbe essere notificato il riconoscimento del device

```
target $ <6>usb 1-1: new high speed USB device using musb_hdrc and
address 3
usb 1-1: new high speed USB device using musb_hdrc and address 3
<6>usb 1-1: configuration #1 chosen from 1 choice
usb 1-1: configuration #1 chosen from 1 choice
<6> ubb: ubb: uba1 uba1
```

60) Verificare che in /dev vi sia la partizione (tipo block) denominata uba1
61) Creare la cartella di mountpoint in /media

```
target $ mkdir -p /mnt/usbflash
```

62) Montare un filesytem di tipo VFAT su uba1

```
target $ mount -t vfat /dev/uba1 /mnt/usbflash
```

63) Verificare il contenuto della memory stick in /mnt/usbflash

```
target $ ls /mnt/usbflash
```

64) Per smontare

```
target $ umount /mnt/usbflash
```

65) Per vedere i dispositivi USB connessi usare il comando lsusb. Si ottiene una lista di righe del tipo

```
Bus 001 Device ZZZ: ID XXXX:YYYY
```

Dove ZZZ è il numero assegnato al device da linux, XXXX è il Vendor ID e YYYY è il Product ID.

1.1 Porting del DVSDK 1.20 su linux git 2.6.25

Hi,

Highlevel flow:
- Need to modify the dsplink release to compile against your kernel
- Need to modify cmemk to compile against your kernel

When you have dsplink and cmemk, you can basically run the codec servers and combos.

We have done this on the past and the issues I can remember are:
- Cmemk may need some patching to adjust the sysfs data structures for kernels beyond 2.6.22
- Current git kernel v4l2 drivers for davinci doesn't even compile afaik. They need some changes to match newer v4l2 stacks.
- Pretty much all of the encode/decode demos will not work without some patching, since they take advantage of specific header files for MV kernel.

That's what I can remember right now.

Regards,

Diego

On Aug 28, 2008, at 3:00 AM, Gabriele Filosofi wrote:

Hi,

The DVSDK v1.30 available from TI is based on the MontaVista Linux 2.6.10 Linux Support Package.
In these weeks I made some effort to port the git kernel 2.6.25 to my DM6446-based board.
Now, does anybody know how to make the DVSDK v1.30 working along with the git kernel 2.6.25?
Also a high level flow would suffice.

Thanks

Gabriele

Vedere
http://linux.omap.com/pipermail/davinci-linux-open-source/2008-February/005220.html

Seguire mailing di
Diego Dompe (diego.dompe@ridgerun.com)

Caglar Akyuz (caglarakyuz@gmail.com)

2 Altro

2.1 Troubleshooting

- *JFFS2 error: (1) jffs2_build_inode_pass1: child dir "hotplug" (ino #690) of dir ino #688 appears to be a hard link*
  Eliminato cancellando l'intera partizione dedicata al file system (/dev/mtd4) prima di scriverci dentro l'immagine del JFFS2
-

2.2 Protocolli di Streaming

- RTP: rfc1889
- RTSP: rfc2326
- SDP: rfc2327 (this is useful for rtsp or multicast streams)
- RTCP: rfc3550

2.3 Sysfs

Sysfs è un file system virtuale che esporta in userspace oggetti presenti in kernel mode, come drivers e devices. Per ciascun oggetto viene creata una directory in sysfs. La struttura delle directory in /sys/devices/ riflette la relazione parent/child. La sottodirectory /sys/bus/ contiene link simbolici, in funzione del bus di appartenenza del device. /sys/class/ mostra I devices raggruppati in classi, come network, mentre /sys/block/ contiene i block devices. Per i device drivers e per i devices si possono creare degli attributi che sono dei semplici file di testo presenti nella sottodirectory del device driver specifico. Questi file contengono un solo valore.

1.1 Creazione di video demos MPEG4

- ffmpeg
- XviD

1.2 Streaming Server SW

http://www.live555.com/mediaServer/

Open source C++ libraries to support RTSP streaming. There are test programs that are available with this library that can be easily cross-compiled on Montavista Linux platform. You can use the test program to stream MPEG4 elementary stream file contents generated by DM355 encoder and can be played back using any PC client like VLC.

1.3 PC Client Media Player SW

- Mplayer
- VLC media player
- QuickTime Player
- Amino set-top boxes (for playing MPEG Transport Streams only)
- The openRTSP command-line RTSP client (which receives/stores stream data, without playing it)
- FFmpeg – is a complete solution to record, convert and stream audio and video. It includes libavcodec, the leading audio/video codec library. FFmpeg is developed under Linux, but it can compiled under most operating systems, including Windows

1.4 Formato file A/V

- MPEG Transport Stream file (“.ts”)
- MPEG-1 or 2 Program Stream file (".mpg")
- MPEG-4 Video Elementary Stream file (".m4e")
- MPEG-1 or 2 (including layer III - i.e., 'MP3') audio file (".mp3")
- WAV (PCM) audio file (".wav")
- AMR audio file (".amr")
- AAC (ADTS format) audio file (".aac")

1.5 WiFi USB dongle

- ZyDas 1211b based WiFi dongle
  Driver: ZD1211LnxDrv_2_15_0_0 (http://linux.omap.com/pipermail/davinci-linux-open-source/2007-February/002478.html); zd1211rw
  Nota: occorre applicare delle patches al kernel di base MVL401 per avere questa funzionalità
- RT73 based WiFi dongle

1.6 NAND content for a linux system

UBL, U-Boot, Linux Kernel, JFFS2 root file system.

1.7 eXpressDSP Algorithm Interface Standard (xDAIS)

- Algorithms from multiple vendors can be integrated into a single system
- Algorithms are framework agnostic
- Algorithms can be deployed in statically scheduled and dynamically scheduled run-time environments
- Algorithms can be distributed in binary form
- Algorithm integration does not require compilation but may require reconfiguration and relinking

1.8 Linux Kernel

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

1.9 Packages e Utilities

- thttpd - HTTP web server (http://www.acme.com/software/thttpd)
- vedere i siti web SourceForge e FreshMeat per i vari protocolli e servizi di rete
- xinetd and tftp server
- kermit (usually called ckermit as a package)
- wireshark, the network analyzer

1.10 Creare e riprodurre video per DM6441 su Win PC

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

1.11 Cambiare la Memory Map

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

1.12 Risoluzioni video

Resolution:
QCIF (NTSC:176x120, PAL:176x144)
CIF (NTSC:352x240, PAL:352x288)

Frame rate:
25fps for PAL, 30fps for NTSC

1 Riferimenti

- *LSP 1.20 DaVinci Linux EVM Installation User's Guide (SPRUFG0)*
- *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)*
- *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/*

Riferimenti People at DaVinci mailing list

- *Lorenzo Lutti at emb-devices.com*
- *Emanuele.Ghidoli at speedautomazione.it*
- *Andrea Gasparini (gasparini@imavis.com) at www.imavis.com*