Webserver su FPGA

Michele Bianchi
Ivan Simonini

22 agosto 2011

Indice

1 Obiettivi, Componenti e Passi previsti

La consegna di questo progetto prevede l'allestimento di un piccolo server web su una piattarma *XILINX Virtex-II Pro ML310*.

1.1 Componenti utilizzati

Per realizzare i nostri scopi, ci sono stati forninti alla consegna alcuni materiali indispensabili:

- la scheda FPGA ***XILINX Virtex-II Pro ML310***, completa della proprio alimentazione e di un piccolo monitor LCD;
- i due pacchetti software necessari per lo sviluppo sulla piattaforma, ***XILINX ISE*** `9.1i` e ***XILINX EDK*** `9.1i`;
- il ***Programmatore parallelo***, componente hardware proprietario responsabile della comunicazione con la scheda, da utilizzare come strumento di controllo a monte del sistema residente installato.

Nel corso della lavorazione, in seguito alle scoperte sulle reali funzionalità della scheda e al sorgere di nuove necessità, alcuni altri componenti sono stati aggiunti all'insieme:

- un ***convertitore Parallela-USB*** per connettere il programmatore parallelo ai nostri portatili;
- un ***convertitore Seriale-USB*** per connettere il canale I/O primario della scheda ai nostri portali;
- un ***cardreader USB*** compatibile con ***CompactFlash***;
- una scheda di memoria ***CompactFlash*** da `4.0GB` in aggiunta e sostituzione della scheda di memoria originale;
- un ***HDD Maxtor IDE*** da `40GB` per aumentare lo storage permanente senza inficiare la memoria Flash.

Per giungere ai risultati ottenuti, abbiamo fatto uso dei nostri portatili:

- un ACER ***Aspire 6930*** con Debian GNU/Linux a `32bit`;
- un SONY ***Vaio EB1A4E*** con Ubuntu GNU/Linux a `64bit`;
- un ASUS ***A3500G*** con Windows XP SP3 a `32bit`.

1.2 Passi previsti

Secondo le nostre conoscenze iniziali, i passi da compiere per raggiungere gli obiettivi del progetto sarebbero stati, nell'ordine:

- la realizzazione di una ***bitstream*** di un microprocessore MicroBlaze;
- il trasferimento di questa immagine sulla piattaforma, tramite il ***programmatore seriale***;
- la compilazione di un recente kernel Linux, accompagnato da una distribuzione GNU minimale, e la sua posa come sistema residente;
- la compilazione e l'installazione di un piccolo webserver sul sistema appena realizzato;
- l'installazione di alcune piccole applicazioni web a scopo dimostrativo.

Come stiamo per narrare, la strada da noi seguita si è distaccata non poco da questa lista, a causa del sorgere di numerosi imprevisti.

Di seguito andiamo a descrivere per prima cosa la procedura ideale, esaminando poi nel dettaglio i numerosi errori e imprevisti riscontrati, congiuntamente alle soluzioni adottate – quando possibile – per procedere.

2 Installazione dei software

Lo sviluppo per piattaforme FPGA è un'operazione vasta e complessa, per la quale il produttore ha realizzato una vasta gamma di applicazioni di supporto e sviluppo.

Abbiamo a disposizione due pacchetti, entrambi alla (ormai piuttosto antiquata) versione `9.1i`:

- la ***ISE Design Suite*** è una vasta collezione di librerie e applicazione di supporto; in particolare, ad essa appartengono ***Impact*** e ***webupdate***;
- l'***Embedded Development Kit EDK*** è una (meno) vasta collezione che include indispensabili applicazioni come ***XPS*** e ***XMD***.

Entrambi i pacchetti, per quanto sorpassati da lungo tempo, dispongono di alcuni aggiornamenti tuttora disponibili; entrambi, inoltre, sono dichiaratamente compatibili per l'installazione su sistemi `Windows`, `MacOs` e `Linux Redhat`.

2.1 Installazione in ambiente `Linux`

La versione per `Redhat`, per quanto funzionante, non è priva di problemi quando installata sotto `Debian` oppure `Ubuntu`. In particolare:

- vi sono stati gravi problemi di valutazione dello spazio libero disponibile su disco;
- il comportamento dell'installazione si è rivelato non coerente con quello d'esecuzione;
- gli aggiornamenti diretti non sono effettuabili dietro un proxy.

La procedura automatica d'installazione non fa uso della comune utility `df` e pare riconoscere come pieno qualunque disco di destinazione. È stato necessario quindi forzare manualmente l'installazione.

La stessa procedura, tra le operazioni preliminari, registra alcune variabili d'ambiente, quali la cartella d'installazione e il path per le applicazioni o le librerie condivise. Appare evidente come queste informazioni non vengano registrate, o per lo meno caricate, al momento del lancio: l'interezza delle applicazioni XILINX fallisce all'avvio. Si è rivelato dunque necessario creare manualmente l'apposito ambiente, attraverso il seguente script:

```
alias gmake='make'
export XILINX=/opt/xilinx/
export XILINX_EDK=/opt/edk/
export LD_LIBRARY_PATH=/opt/xilinx/bin/lin/:/opt/edk/bin/lin/

export CROSS_COMPILE=microblaze-unknown-linux-gnu-
export PATH=$PATH:/opt/xilinx/bin/lin/:/opt/edk/bin/lin/
```

Infine, i principali componenti da utilizzare prevedono l'aggiornamento di alcuni contenuti tramite il gestore degli aggiornamenti ***webupdate***, membro di ***ISE***. Per quanto presenti un pannello per la configurazione dei parametri di connessione, il programma ignora tali parametri e tenta sempre e comunque la connessione diretta alla rete. Trovandoci a lavorare all'interno della rete della facoltà, che si trova dietro un server proxy, questo comportamento ci ha impedito di aggiornare il software.

Fortunatamente, esistono dei ***service pack*** – sia per ***ISE*** (`SP2`) che per ***EDK*** (`SP3`) – disponibili gratuitamente per il download dal sito ufficiale. Essi vanno installati come le altre applicazioni e l'operazione è andata a buon fine, nonostante abbia presentato gli stessi problemi nella stima dello spazio libero su disco, ed abbia richiesto la forzatura manuale.

Tristemente, uno dei due ***pack*** – nel dettaglio, gli aggiornamenti per ***EDK/XPS*** – è stato causa di fastidiosi problemi di compatibilità. La ***suite*** completa (`9.x`) supporta non soltanto le piattaforme ***Virtex-II Pro*** `ML300/310`, ma anche la gamma di sorelle minori ***Spartan*** `2/2e` e `3/3a/3e`. L'installazione di questo aggiornamento ha compromesso il funzionamento di ***XPS*** per via di alcune collisioni nella configurazione.

Non avendo comunque ricavato alcun vantaggio evidente da questi ***service pack***, si raccomanda di NON INSTALLARE gli aggiornamenti e di mantenersi alla versione del CD-ROM.

2.2 Installazione in ambiente `Windows`

L'installazione di entrambi i pacchetti sotto `Windows XP`, al contrario, non ha dato il benché minimo problema.

L'intera procedura è stata completata automaticamente, senza perdere un colpo; il ***webupdate*** ha seguito la configurazione di sistema per il collegamento alla rete ed ha correttamente aggiornato tutte le parti interessate, ed il processo non ha creato alcun conflitto.

Nel complesso, l'installazione – completa di tutti gli aggiornamenti – arriva ad occupare poco meno di ***5,7GB*** di spazio disco su entrambi i sistemi.

3 Messa in posa di un sistema residente

Il processore FPGA è un sistema volatile il cui contenuto viene perduto in mancanza di corrente, pertanto è necessario imprimere su di esso l'immagine di un microprocessore compatibile ad ogni avvio, così come dopo ogni pressione del tasto ***SysACE RESET***.

Illustriamo ora nei dettagli la procedura di creazione e posa di ***bitstream*** ed eseguibile, accompagnato da una breve guida sulle applicazioni da utilizzare nei vari passaggi. Il tutto – ovviamente – sotto l'ipotesi che si disponga di tutti gli strumenti necessari; per la reale procedura seguita, si veda la sezione successiva.

3.1 Realizzazione di un *bitstream*

Il primo, importante passaggio per il funzionamento della piattaforma è la generazione di un ***bitstream*** compatibile, un blob binario rappresentante

la mappa delle connessioni da realizzare all'interno del processore FPGA, al fine di emulare un microprocessore.

Questa operazione è significativamente complessa e si compone di molte fasi. Fortunatamente, la quasi totalità delle operazioni può essere eseguita in modo automatico ad opera di una lunga catena di applicazioni, una ***toolchain***. Per gestire questo processo è stato creato ***XILINX Platform Studio*** o ***XPS***, principale componente di ***EDK***.

XPS si presenta come un tipico IDE, offrendo una procedura quasi completamente automatica per creare un nuovo progetto (alternativamente, è possibile apportare modifiche ad un progetto già presente). Questa procedura prevede innanzitutto la scelta dell'architettura base del processore; per la nostra ***Virtex-II PRO ML310*** le alternative sono due, il microprocessore proprietario XILINX ***MicroBlaze***, appositamente ideato per schede come questa, oppure un ***PowerPC405***.

In seguito a questa scelta preliminare passiamo attraverso una lunga serie di opzioni che ci permettono, tra le altre cose, di collegare o meno gli elementi al bus, di inserire o rimuovere componenti quali il controller `PCI` o il clock ad alta precisione, regolare dimensioni, ruoli e numero di livelli della cache.

In seguito all'accettazione delle opzioni, è possibile apportare numerosissime modifiche al progetto. Senza entrare nei dettagli, che porterebbero via molto spazio e molto tempo, ci limitiamo a dire che quasi ogni singola connessione di ogni singolo componente è configurabile.

Una volta raggiunto il risultato desiderato, è caldamente consigliato salvare i propri progressi. In seguito è possibile invocare la ***toolchain*** e dare inizio ad una lunga serie di operazioni che corrispondono alla raccolta di numerosi file `vhdl` – ciascuno corrispondente ad un componente hardware detto ***core*** – alla realizzazione di una mappa preliminare di connessioni, a varie ottimizzazioni e raffinazioni, fino alla finale conversione in ***bitstream***.

Nella sua interezza, questa procedura si compone di una ventina di fasi e può prendere – almeno per quanto abbiamo potuto vedere nel corso di numerosi tentativi – un arco di tempo che va dai venti minuti ad oltre mezz'ora. Si raccomanda pertanto pazienza oppure l'utilizzo di una macchina abbastanza potente.

Va detto che la potenza di ***XPS*** non si limita a questo: il programma è anche in grado di generare, per ciascuna particolare architettura realizzata, strumenti ausiliari come il ***DTS*** e librerie, necessari alla compilazione di un kernel compatibile.

3.2 Trasferimento del *bitstream*

Occorre ora trasferire il ***bitstream*** appena generato sul processore FPGA. La via più immediata consiste nel comunicare direttamente con il microcontrollore dedicato, il ***SysACE Controller***. Questo componente è in grado di apportare modifiche all'FPGA basandosi su due fonti, il ***programmatore parallelo*** oppure la memoria ***CompactFlash*** (della quale parleremo in una fase successiva).

Il ***programmatore parallelo*** è un congegno che si collega alla piattaforma su un canale riservato tramite una connessione a `14pin` e interagisce via una porta parallela a `21pin`. Esso deve essere alimentato separatamente ed è in grado di scambiare dati con la piattaforma, agendo a monte del processore emulato, rendendo possibile – qualora necessario – il completo controllo dell'esecuzione e la verifica di ogni stato della macchina.

Per interfacciarsi con il programmatore occorre utilizzare un programma dedicato, che in questo caso è ***Impact***, un componente di ***ISE***.

Questo programma è in grado di manipolare i ***bitstream*** così come i file `ACE` (dei quali parleremo tra poco), e di trasferirli alla piattaforma.

È possibile scegliere fino ad un massimo di otto diverse configurazioni, le quali possono risiedere contemporaneamente sulla piattaforma (sebbene una soltanto possa essere attiva).

3.3 Trasferimento del kernel

Anche con un processore emulato, la piattaforma non è pronta per funzionare: occorre infatti che il processore abbia qualcosa da eseguire. È necessario a questo punto aggiungere anche un file eseguibile formato `ELF`.

Posto di avere un kernel compatibile a disposizione (di questo e della relativa compilazione parleremo più avanti), è possibile utilizzare un altro programma – ***XMD***, componente di ***EDK*** – per trasferire un eseguibile.

In effetti, ***XMD*** è un programma che permette di controllare il flusso delle operazioni eseguite dal processore emulato, ed è quindi un potente strumento di debug; esso è anche in grado di modificare lo stato della memoria, più nello specifico può essere usato per trasferire un intero kernel e poi lanciarne l'esecuzione.

Per prima cosa è necessario connettersi al ***SysACE Controller***, operazione che richiede – come parametro – l'architettura alla quale ci si connette. Di seguito, si può trasferire un qualsiasi file `ELF` ed eseguirlo.

Come ultima operazione, se tutto è andato come ci si aspetta, dovrebbe essere possibile loggarsi sul sistema residente attraverso il canale di I/O principale.

Se le varie operazioni sono state eseguito come abbiamo elencato, questo canale sarà la porta seriale: è quindi sufficiente collegare il proprio PC alla porta seriale – tramite un adattatore USB, nel caso – settando la connessione su `9600baud` senza bit di parità (`8N1`). Il kernel comincia a comunicare su questa porta non appena avviato, ciò permette di visionare anche il log del boot up, in seguito al quale dovrebbe apparire il login.

Si noti anche che questo è un uso marginale dell'applicazione: nel caso in cui gli scopi fossero diversi da quelli da noi prefissi, é possibile, in questa fase, controllare passo passo l'esecuzione di un qualunque programma scaricato, senza la necessità di comunicare attraverso una porta, e senza avere un kernel residente attivo.

3.4 Boot da *CompactFlash*

Ma attenzione, poiché questo non è l'unico modo per trasportare la coppia ***bitstream***/eseguibile `ELF` sulla piattaforma: quello descritto è semplicement il sistema più comodo per chi volesse sviluppare e testare un proprio microprocessore ed il sistema che deve girare su di esso.

Per chi come noi deve invece utilizzare la piattaforma come fosse un server, esiste un'alternativa che prevede una maggiore quantità di lavoro a monte ma permette di risparmiare molto tempo al boot.

Per quanto infatti il trasferimento del ***bitstream*** sia relativamente rapido (qualche decina di secondi per trasferire poco meno di `1MB`), il trasferimento del kernel (il cui peso, nel nostro caso, variava da alcune decine ad oltre un centinaio di `MB`) può prendere diversi minuti. Il processo deve ovviamente essere ripetuto per intero ad ogni modifica.

Esiste quindi la possibilità di trasferire entrambi i file su un supporto duraturo come la memoria ***CompactFlash*** ed effettuare il boot da essa. Questa è una possibilità offerta dal ***SysACE Controller***.

Come accennato precedentemente, questo controllore supporta un massimo di otto configurazioni. La scelta di quale lanciare è delegata ad un piccolo componente hardware, il ***SystemACE configuration DIP switch***.

Esiste, per nostra fortuna, una configurazione di base: come fornita dal produttore, la ***CompactFlash*** dispone degli otto slot divisi come segue:

- lo slot 0 é riservato al ***SysACE loader***, un minuscolo applicativo il cui unico scopo è permettere la selezione di un altro slot a runtime;

- gli slot 1~5 contengono varie configurazioni di test, dal semplice ***RAM checker*** ad una completa distribuzione ***MontaVista Linux***;

- gli slot 6 e 7 sono lasciati vuoti per le configurazioni utente.

Come brevemente accennato precedentemente, la generazione del file `ACE` – che non è altro che un blob contenente entrambi il ***bitstream*** e l'eseguibile `ELF` – è uno dei compiti che ***Impact*** è in grado di portare a termine.

Il boot da ***CompactFlash*** ha il vantaggio di segnalare le proprie fasi attraverso i LED presenti sulla scheda, che lampeggiano e si stabilizzano sul verde nel momento in cui l'architettura viene creata. Immediatamente dopo, al boot dell'immagine selezionata, questa produce output attraverso la porta seriale.

Per la verifica è sufficiente collegarsi a questa porta attraverso un opportuno emulatore di terminale seriale, come ***minicom*** nel nostro caso, oppure ***PuTTY*** in ambiente Windows.

4 Problemi riscontrati

Fin qui abbiamo descritto i passi necessari ad avere un sistema operativo residente sulla piattaforma.

Va detto però che durante nostri tentativi siamo incappati in numerosi imprevisti che hanno interrotto e portato al catastrofico fallimento la totalitá delle operazioni descritte.

Nell'ordine:

- alcuni componenti chiave per il disegno del microprocessore, di seguito ***core***, sono soggetti a licenze delle quali non disponiamo;

- passaggi quali la generazione di `DTS`, librerie e patch per il kernel necessitano del progetto completo, mentre il solo ***bitstream*** non è sufficiente;

- la comunicazione attraverso la porta parallela si è rivelata tutt'altro che stabile;

- formattare correttamente la ***CompactFlash*** è stato tutt'altro che banale;

- il kernel ***MontaVista 2.4***, appositamente patchato per il supporto alla configurazione hardware attraverso specifici file header e device driver, non viene più distribuito;

- la distribuzione ***μCLinux***, specificamente patchata per ***MicroBlaze***ed consigliata nella maggior parte dei tutorial, non é piú disponibile gratuitamente;

- il download delle immagini del kernel attraverso ***XMD*** non è mai giunta al termine, nè ha mai dato segno di star facendo progressi.

Segue una descrizione dettagliata dei problemi incontrati e delle soluzioni adottate per aggirarli.

4.1 Licenze scadute

Ciascuno degli elementi fondamentali per la progettazione del microprocessore all'interno di ***XPS*** – i vari ***core*** – è proprietà intellettuale (***Intellectual Property*** o ***IP***) appartenente a XILINX.

Alcuni di essi, nel nostro caso particolare i controller `I²C`, `PCI` e `Ethernet`, sono soggetti a licenze separate da quelle di ***ISE/EDK***.

Queste specifiche licenze, acquistate assieme all'intera suite software, sono scadute nel 2008, ed essendo le versioni `9.x` non piú supportate, non sono nemmeno rinnovabili.

In assenza di queste licenze, ***XPS*** interdice qualunque passaggio ulteriore nello sviluppo del progetto. Questo significa che è comunque possibile apportare modifiche, integrare oppure manipolare il progetto e varie altre operazioni poco significative, ma non è possibile generare né ***bitstream***, nè `DTS`, nè librerie.

Sebbene il controller `I²C` fosse completamente opzionale e potesse essere rimosso, il controller `Ethernet` era assolutamente indispensabile agli scopi del progetto, così come il controller `PCI`, al quale il precedente si appoggia.

Ci siamo quindi trovati nell'impossibilità di creare, sulla piattaforma, un processore personalizzato. La soluzione adottata è stata il recupero di immagini ***bitstream*** già pronte, fortunatamente reperibili in rete con una certa facilità, dovuta alla seppur scarsa presenza di altri progetti simili al nostro nel recente passato.

4.2 Utilizzo della porta parallela

Anche posto d'avera a disposizione un ***bitstream***, questo ed anche il relativo file eseguibili devono essere trasferiti sulla piattaforma. Prima di scoprire la via alternativa tramite diretto accesso alla ***CompactFlash***, l'unica strada passava per il ***programmatore parallelo***.

Questo congegno – e le due applicazioni che ne fanno uso, ***Impact*** e ***XMD*** – è abbastanza capriccioso. Esso esige due particolari condizioni che non è stato semplice soddisfare.

Primo, per la comunicazione si deve avere accesso diretto a basso livello sulla porta. Molte delle primitive di comunicazione di questo tipo non possono essere emulate dal controller `USB`, impedendo l'utilizzo della porta attraverso un adattatore.

Questo è il motivo per cui abbiamo scomodato il portatile ASUS, il quale – un po' per l'età avanzata, un po' per caso – è dotato di una porta parallela nativa. Il passaggio obbligato dai nostri due portatili a questo ha ulteriormente rallentato il lavoro, anche se l'alternativa (eseguire anche le pesanti operazioni di generazione e compilazione sul portatile dotato di porta parallela) è stata accantonata quasi immediatamente, di fronte ai prolungati tempi di esecuzione.

Secondo, entrambi i software hanno tentato di utilizzare il cavo di collegamento (un `Parallel Cable IV`) al massimo delle sue possibilità, con una comunicazione bidirezione `half-duplex` detta ***EPC - Enhanced Parallel Cable*** che permette una frequenza di `2MHz`, attraverso alcuni appositi driver.

Questi driver – che è stato necessario installare a parte del resto della ***suite***, sia su `Linux` e che su `Windows` – si sono rivelati completamente inutilizzabili.

La guida ufficiale di ***JUNGO*** – azienda produttrice e distributrice di questi driver, nonché partner ufficiale di XILINX – elenca una (ristretta) lista di schede madri le cui porte parallele sono state testate con i driver in questione. Questa lista non riporta la scheda del nostro ASUS, e tutto ciò che ***JUNGO*** consiglia è di procurarsene una inclusa nella lista.

La più ragionevole guida XILINX consiglia invece di passare alla cosiddetta ***"modalità compatibile"***, che consiste nel trattare il cavo come fosse della generazione precedente (un `Parallel Cable III`), il che limita la comunicazione a `200KHz`.

Per quanto tecnicamente funzionanti, queste soluzioni temporanee hanno comunque limitato il trasferimento del ***bitstream*** a velocità bassissime.

Per quanto riguarda il trasferimento del kernel attraverso l'uso di ***XMD***, nessun tentativo ha mai dato esito positivo. Tutti i numerosi tentativi di download (con numerosi kernel diversi) si sono bloccati immediatamente dopo il lancio, senza dare alcun altro segno, né dal software, né dai LED presenti sulla scheda.

Alla fine, l'intero passaggio è stato evitato, e lo sviluppo è passato a basarsi sul boot da ***CompactFlash***.

4.3 Clonazione della *CompactFlash*

Nella previsione di un considerevole numero di operazioni, non conoscendo l'età della ***CompactFlash*** in dotazione, nonché preoccupati sulle dimensioni (la scheda, da `512MB` complessivamente, pareva avere soltanto `5MB` di spazio libero, decisamente insufficiente ad ospitare i programmi che avremmo dovuto installare), abbiamo deciso di clonarla su una scheda nuova, da `4GB`.

Abbiamo quindi formattato la nostra ***CompactFlash*** come l'originale ed abbiamo copiato entrambe le partizioni presenti.

Realizzare la corretta formattazione si è rivelata un'operazione insospettabilmente ostica.

L'originale CompactFlash si presenta con due partizioni primarie

- la prima in `vfat`, contentente un semplicissimo file indice per il ***SysACE controller*** ed una serie di otto cartelle, una per configurazione, contente ciascuna almeno il file `ACE` (alcune riportano, per ridondanza, il ***bitstream*** e l'eseguibile `ELF`)
- la seconda in `ext2`, contenente la root di un file system `Linux` e identificata come `/dev/xsysace/disk0/part2`

In seguito a vari tentativi falliti, ai quali il ***SysACE Controller*** risponde semplicemente accendendo un LED rosso, che segnala una grave incompatibilità degli `ACE`, abbiamo eseguito ispezioni del contenuto della partizione (che è risultato corretto) e poi man mano più approfondite osservazioni sulla formattazione della partizione `vfat`.

È emerso come il ***SysACE Controller*** sia estremamente puntiglioso su questo particolare:

- la partizione riservata deve essere la prima, e deve essere primaria e non logica;
- la partizione può essere in `FAT12` oppure `FAT16`, ma nient'altro, né successivo né precedente;
- la partizione DEVE avere una e una sola tabella FAT, che DEVE cominciare esattamente al secondo cilindro, e DEVE pertanto essere formattata con un singolo settore riservato;

- la partizione deve essere completamente indicizzabile, quindi, in base alle sue dimensione, è necessario aggiustare il numero di `settori-per-cluster` così come la dimensione dei settori (in `cilindri` e non in `byte`).

Attraverso `mkdosfs` NON è possibile (per quanto la documentazione ufficiale asserisca il contrario) specificare un numero di settori inferiore a 8, fatto che compromette completamente il funzionamento della partizione.

L'unica soluzione efficace è stata raggiunta in ambiente Windows, attraverso l'utilizzo di un'utilità di formattazione freeware, ***EaseUS Partition Master*** `9.0`.

La seconda partizione è invece molto più flessibile: può essere formattata in `ext2` oppure `ext3`, ma a patto che la dimensione degli `inode` non superi i `128byte`. La mancanza nel seguire questo consiglio impedisce ai kernel utilizzati di montare il `filesystem`.

4.4 Compilazione del kernel

Era nostra intenzione compilare il kernel piú recente disponibile, allora il `2.6.39`. I kernel `2.6.x` supportano configurazioni hardware non standard tramite ***DTS – Device Tree Source*** anziché attraverso file header come le precendenti versioni `2.4.x`.

Il file `dts`, che rende molto piú semplice descrivere un'architettura in modo portabile, è generabile da ***XPS*** a partire dal progetto di un processore, a patto che il progetto sia manipolabile: nel nostro caso, il problema con le licenze preclude questa funzione.

Come abbiamo potuto verificare, senza questo file il kernel può essere compilato e installato. Purtroppo però, pur essendo compatibile con il processore, esso non riconosce la configurazione hardware sulla quale viene montato ed è quindi impossibilitato a comunicare con qualunque periferica hardware.

Non potendo accedere ad alcun dispositivo di Input/Output, né tantomeno alla memoria, un sistema così realizzato è intrinsecamente inutile e inutilizzabile.

Siamo quindi andati in cerca di altri kernel compatibili.

Ci siamo rivolti direttamente al repository GIT ufficiale XILINX, dal quale abbiamo recuperato i sorgenti patchati del kernel `2.6.37` per avere il supporto a ***core*** proprietari.

Gli aggiornamenti di queste patch ufficiali riguardano esclusivamente i prodotti supportati. Attualmente, schede `ML410` o successive; la nostra `ML310` è obsoleta e non viene più supportata.

Inoltre, anche questo kernel richiede il `dts`.

Un porting specifico del kernel Linux per ***MicroBlaze*** è stato realizzato come parte della distribuzione μCLinux. Questa particolare distribuzione è divenuta commerciale con il nome di ***PetaLinux*** e non è piú disponibile gratuitamente.

Una versione patchata del kernel `2.4.20` viene segnalata dal sito ufficiale XILINX come parte della distribuzione ***MontaVista***. Il link indicato non é valido e tutte le versioni disponibili di questa distribuzione sono a pagamento.

Quando ci è apparso chiaro che nessuna delle alternative viste fino ad allora era percorribile e nessun kernel compilabile, siamo ricorsi all'unico sistema Linux funzionante disponibile, ossia il ***MontaVista Linux Professional 3.1*** installato sul slot 1 della ***CompactFlash***.

5 Crosscompilazione

La configurazione dello slot 1, sulla quale abbiamo basato tutto il lavoro successivo, si compone di un ***bitstream*** che ricrea un ***PowerPC405*** e una distribuzione ***MontaVista Linux Professional 3.1***, discendente di ***Hard Hat Linux***.

Questa distribuzione, per quanto stabile e ben fornita, è piuttosto antiquata e manca dei fondamentali strumenti quali un compilatore e molte librerie di sviluppo. Il passo successivo ha visto quindi la necessità di installare questi strumenti.

Inutile dire che la presenza di binari compatibili con questo kernel (versione `2.4.20`), con questa glibc (versione `2.3.2`) e con quest'architettura (`PPC405`) è sostanzialmente nulla.

Il fatto non stupisce ed è ampiamente giustificato dall'estrema specificità della nostra situazione, così come dall'età della piattaforma: il suo rilascio risale al 15/08/2004, mentra la sua messa fuori produzione si attesta attorno al 2007.

Ci siamo quindi trovati nell'insolita necessità di dover costruire un crosscompilatore per un'architettura specifica e contemporaneamente per una specifica versione del sistema.

Per risolvere questo ostico problema ci siamo appoggiati ad un potente e specifico strumento, ***Crosstool***. Scopo di questa ***toolchain*** è creare un completo ambiente di crosscompilazione, da un'architettura di costruzione ad un'architettura target.

Per 'completo' si intende, in questo caso, l'intero insieme delle applicazioni utilizzate dal `GNU build system`: occorrono un preprocessore, un compilatore, un assemblatore e un linker per produrre i binari dai sorgenti, occorre un archiver per produrre le librerie statiche, occorre la glibc perché tutto possa essere linkato correttamente, una volta a destinazione, occorrono tutte le eventuali librerie, e infine occorre che tutto questo insieme sia isolato dal sistema ospite.

Si tratta, in sostanza, di costruire una vecchia versione di `gcc` (la versione più vicina al kernel target è la `3.4.4`) e di tutti i suoi programmi ausiliari, che possa compilare per il sistema residente sulla piattaforma, in accordo con il kernel presente.

Purtroppo, lo stesso ***crosstool*** è uno strumento ormai deprecato, visto che il progetto è stato abbandonato nel 2006 e sostituito con i più recenti ***Crosstool-NG*** e ***Builtroot/Crosstoolchain***, che tuttavia non supportano versioni tanto vecchie, si tratti di kernel, di glibc o di compilatore.

5.1 Utilizzo di *crosstool*

Seguendo pedissequamente il tutorial, i passi da seguire non sono poi tanti; si tratta di costruire un breve file di configurazione che deve elencare:

- l'architettura di destinazione;
- la versione del kernel di destinazione;
- la versione del compilatore da utilizzare;
- la versione della glibc da utilizzare con il compilatore;
- la versione di binutils e gdb.

Tale file deve essere agganciato dallo script utilizzato per lanciare la compilazione: nel nostro caso, il file di configurazione è `gcc-3.4.4-glibc-2.3.2.dat`

```
BINUTILS_DIR=binutils-2.15
GCC_DIR=gcc-3.4.4
GLIBC_DIR=glibc-2.3.2
LINUX_DIR=linux-2.4.20
GLIBCTHREADS_FILENAME=glibc-linuxthreads-2.3.2
GDB_DIR=gdb-6.5
```

Questo file va segnalato per l'utilizzo nello script `demo-powerpc-405.sh`, il cui contenuto modificato dovrebbe corrispondere a

```
set -ex
TARBALLS_DIR=$HOME/downloads
RESULTTOP=/opt/crosstool
export TARBALLS_DIR RESULT_TOP
GCC_LANGUAGES="c,c++"
export GCC_LANGUAGES

mkdir -p $RESULT_TOP
eval `cat powerpc-405.dat gcc-3.4.4-glibc-2.3.2.dat` sh all.sh --notest

echo 'Done.'
```

Al primo lancio, questo script scarica tramite `wget` tutti i pacchetti necessari alla compilazione. Ecco alcune considerazioni da tenere a mente:

- non è necessario scaricare nuovamente tutti i pacchetti, dopo il primo errore; per indicare allo script che i sorgenti sono già disponibile lo è sufficiente aggiungere l'opzione `--nounpack` alla penultima riga;
- il download di questi pacchetti viene effettuato attraverso dei repository il cui indirizzo è ben integrato nei meandri dello script; alcune versioni possono non essere disponibili, così come il download può fallire in seguito al sovraccarico del server; in entrambi i casi è consigliato rimuovere i pacchetti parzialmente scaricati. La mancata esecuzione di questo passo comporta il fallimento dei test e l'interruzione dello script;
- lo script di configurazione – `all.sh`, che effettua numerossissimi test di compatibilità sul sistema ospitante – è piuttosto antiquato ed estremamente capriccioso; può succedere che alcuni test falliscano senza motivo appartente; è stato necessario, ad esempio, modificare manualmente lo script perchè accettasse il nostro compilatore (un gcc `4.6.1`) come funzionante (veniva altrimenti scambiato per una versione obsoleta).

Va aggiunto che, anche in mancanza di errori di configurazione, la compilazione può fallire in caso di utilizzo di compilatori e/o glibc recenti. Numerossissimi problemi sono stati incontrati anche durante i tentativi di compilazione in un ambiente a `64bit`, per problemi apparentemente minori quali – un esempio tra i tanti– la dimensione del tipo `long long int`.

In seguito ad uno spropositato numero di tentativi, congiuntamente ad altrettanto numerosi interventi manuali a basso livello per correggere – laddove possibile, altrimenti per escludere – gli errori (e le fonti di errori), la compilazione ha dato esito positivo. Questo ci ha permesso di creare un ambiente

di crosscompilazione, completo di tutti gli strumenti, da `i686-pc-linux-gnu` a `powerpc-405-linux-gnu`.

Grazie a questo ambiente, siamo stati nelle condizioni di compilare applicazioni e librerie da montare sul sistema residente sulla scheda.

5.2 Software installato sulla piattaforma

Questo è un elenco delle librerie e applicazioni che sono state compilate per la piattaforma e in seguito installate

- ***zlib*** `1.2.5`, libreria di compressione di largo utilizzo;
- ***libxml*** `2.7.7`, libreria di gestione e manipolazione dei file `XML`;
- ***libpcre*** `8.12`, libreria per l'interpretazione delle `PERL Regexp`;
- ***libiconv*** `1.14`, libreria per il supporto alla codifica internazione dei caratteri;
- ***sqlite*** `3070701`, motore per database estremamente leggero;
- ***e2fs*** `1.41.14`, collezione di strumenti di basso livello per la formattazione e l'accesso a file system `EXT2` e successivi;
- ***gd-libgd*** `1.6.3`, libreria per la manipolazione di immagini;
- ***openssl*** `1.0.0d`, strumenti di controllo e utilizzo di socket sicure;
- ***openssh*** `5-8p2`, emulatore di terminale attraverso socket sicure;
- ***dropbear*** `0.53.1`, leggero client/server `ssh`;
- ***php*** `5.3.6`, motore di renderizzazione per il linguaggio `PHP`;
- ***lighttpd*** `1.4.29`, piccolo webserver;

5.3 Crosscompilazione dei software necessari

L'elenco precedente è abbastanza corposo e tutti i suoi componenti sono stati compilati con il cosidetto metodo ***"canadian cross"***. Questo metodo si basa sui diffusissimi `GNU Autotools`, i quali permettono, congiuntamente con un crosscompilatore, di ottenere binari per una piattaforma anche completamente diversa.

La procedura non è affatto indolore: essa prevede la complessa interazione di `Autoconf` e di `Automake` con l'ambiente di crosscompilazione. Occorre

quindi fare in modo che, per ciascuna applicazione, questi tre enti collaborino. Il che, ovviamente, dipende moltissimo dalla versione dell'uno o dell'altro strumento utilizzata dallo sviluppatore del software.

Diamo ora un elenco puramente dimostrativo di alcune delle opzioni generalmente accettate dallo script per la configurazione dei `Makefile`, l'onnipresente `configure[.sh]`:

- innanzitutto, le tre opzioni fondamentali per il ***canadian cross***, ossia le tre architetture coinvolte: l'architettura che ospiterà il programma `--host=HOST`, l'architettura sulla quale si effettua la compilazione `--build=BUILD`, e infine l'architettura bersaglio per la quale si andrà a compilare `--target=TARGET` (quest'ultima, fortunatamente, da utilizzarsi soltanto per compilare compilatori)
  Si noti che, per quanto la documentazione ufficiale asserisca che in caso di mancata definizione, `target` coincida con `host` e quest'ultimo con `build`, varie combinazioni dei tre possono produrre diversi risultati, a volte desiderabili, a volte no;

- l'utilizzo di alcune librerie esterne – quali ad esempio le librerie di compressione – è spesso configurabile per essere aggiunto (`--with-XXX`) e in altri casi per essere rimosso `--without-XXX`)
  nel caso dell'inclusione, è a volte possibile, a volte necessario, specificare dove reperire le librerie (`--with-XXX-lib`) oppure gli header (`--with-XXX-dir`);

- molte opzioni di configurazione interna al programma (senza il coinvolgimento di librerie esterne) possono essere abilitate (`--enable-XXX`) o disabilitate (`--disable-XXX`) per ragioni di spazio, di prestazioni o di funzionalità;

- la destinazione dell'installazione, così come le cartelle per gli eseguibili, per la documentazione, le librerie e varie altre, sono configurabili per essere diverse tra il momento della compilazione e l'installazione, con opzioni come `--[e[xec-]]prefix`, `--bindir`, `--mandir`.

Oltre a questo, la compilazione risponde ad alcune variabili d'ambiente, che spesso vengono silenziosamente ignorate o altrettanto silenziosamente sovrascritte dal `configure`, ma che a volte sono necessarie per sopperire ad alcuni buchi nella configurazione

- il compilatore `CC`;

- il linker `LD`;

- il gestore di archivi `AR`;
- le opzioni del preprocessore `CPPFLAGS`;
- le opzioni del compilatore `CFLAGS`;
- le opzioni del linker `LDFLAGS`;
- le librerie condivise da linkare `LIBS`.

5.4 Problemi e soluzioni per la crosscompilazione

Data la straordinaria varietà di piattaforme software/hardware disponibili per lo sviluppo, per l'eterogeneità dei gruppi di lavoro e degli strumenti per gestire o creare un ***build system***, ci si trova spesso nella spiacevole situazione in cui gli stessi parametri e le stesse opzioni possono funzionare correttamente, essere ignorate o persino produrre errori.

Riportiamo quindi un singolo esempio di ambiente caricato per la compilazione di `openssh`, la quale – per motivi che non siamo stati in grado di identificare – si rifiutava di includere i file `header` di `openssl` tramite le chiamate di `configure` ma ha positivamente reagito alle stesse opzioni, specificate tramite variabile d'ambiente.

```
#!/bin/bash
__PREFIX=/opt/crosstool/ppc405

export PATH=$__PREFIX/bin:$PATH
CFLAGS=-I$__PREFIX/usr/local/include
CFLAGS="$CFLAGS -I$__PREFIX/usr/local/ssl/include"
export CFLAGS
LDFLAGS=-L$__PREFIX/usr/local/lib
LDFLAGS="LDFLAGS -L$__PREFIX/usr/local/ssl/lib"
export LDFLAGS

LD_LIBRARY_PATH=$__PREFIX/usr/local/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

Quello che segue è invece lo script che raccoglie le numerose opzioni passate al `configure` nella prima fase della compilazione di `php`:

```
#!/bin/bash
time ./configure \
  --libdir=/opt/crosstool/ppc405/usr/local/lib \
```

```
--host=powerpc-405-linux-gnu \
--build=`./config.guess` \
--with-iconv=/opt/crosstool/ppc405/usr/local \
--without-xmlrpc --without-pear --disable-phar
```

Tra tutti i software compilati ed installati, `php` è il più pesante e complesso; inoltre, la sua compilazione ha presentato l'unione tutti i problemi di compilazione incontrati durante l'intero progetto.

Andiamo a riportare quindi cosa può andare storto e cosa si può tentare di fare.

La prima opzione (`--libdir`) indica allo script quale sia la cartella in cui andare a cercare le librerie condivise da utilizzare per la fase di linking al termine della compilazione.

Benché la stessa cartella fosse stata aggiunta tramite una variabile d'ambiente, infatti, questo particolare script ha richiesta la stessa definizione due volte.

La seconda e la terza riga specificano le architettura di destinazione e di costruzione: come si nota, a volte specificare manualmente l'architettura di construzione (normalmente `i686-pc-linux-gnu`, almeno nel nostro caso) non basta. Lo script `config.guess`, che raccoglie varie informazioni invocando il comando `uname`, viene usato per produrre il nome dell'architettura presente nel modo conforme alla presente versione di `configure`.

C'è stato poi il bizzarro caso di `libiconv`, un componente necessario a `php` per il supporto ai caratteri del quale – tecnicamente – si può fare a meno, ma a costo di bizzarri comportamenti a runtime, nel caso della non rara necessità di conversione dei caratteri `unicode`.

Questa libreria, pur compilata e presente nel `filesystem` in posizione perfettamente standard, non veniva rilevata dallo script. È stato necessario quindi indicare la sua posizione esatta.

Infine, nell'ultima riga si specificano tre moduli di cui `php` deve fare a meno. Questa è stata una scelta obbligata per un insormontabile problema.

Intanto, `xmlrpc` è un modulo che abilita il supporto al calcolo distribuito tramite richieste `xml` ed è nient'affatto necessario; `pear` è il modulo di gestione dei moduli, per il caricamento e la registrazione a runtime, utile in alcuni casi ma non strettamente necessario; `phar`, infine, è un sistema di archiviazione che permette alle applicazioni `PHP` di essere pacchettizzate per l'installazione e disinstallazione rapida, e dunque non è necessario.

Ora, per parlare del vero problema, diciamo che non tutti i software prestano la stessa attenzione alla possibilità che il software in questione possa o debba essere compilato su una piattaforma diversa da quella di destinazione e fanno quindi uso di una pratica che non è necessariamente malvagia, ma può – in casi come il nostro – produrre delle noie non trascurabili.

Come parte della compilazione, lo stesso `php` e i tre moduli richiedono l'esecuzione di codice. Nel primo caso, si tratta di compilare e lanciare dei minuscoli applicativi il cui unico scopo è generare alcuni altri file (alcuni sorgenti `.c` ed alcuni header `.h`) necessari ai passi successivi della compilazione, negli ultimi tre invece si tratta di eseguire corposi blocchi di codice `php`. Questa operazione è ovviamente impossibile, dato che l'architettura presente non è in grado di eseguire il codice in questione.

Per soddisfare queste dipendenze è stato quindi necessario, ad ogni singola iterazione, attendere l'arresto della compilazione, esaminarne i risultati per scoprire quale eseguibile dovessere essere lanciato e quale dovessere essere il file da produrre, quindi estrarre l'eseguibile in questione dai meandri dell'albero di costruzione, trasportarlo sulla piattaforma di destinazione, lanciarlo e raccoglierne i risultati, quindi impiantare il file appena prodotto nell'esatta locazione e lanciare una nuova compilazione.

Si raccomanda, in questi casi, ESTREMA ATTENZIONE alla PRESERVAZIONE di COPIE di questi file generati. Essi sono infatti intrinsecamente temporanei, ed hanno pertanto la tendenza a non sopravvivere alle ultime fasi della compilazione successiva. Può essere necessario dover nuovamente copiare questi file ad ogni successiva invocazione di `make`.

Qualche parola, infine, sui nostri tentativi di compilazione di un compilatore nativo per la piattaforma. ***gcc*** `3.4.4` è stata la nostra prima scelta, essendo questa la stessa versione del crossocompilatore utilizzato per tutte le altre operazioni.

Di fronte al fallimento, sono state prese in considerazione anche versioni precedenti (come la `3.0.1` e la `3.2`) o successive (`4.6.1`, attualmente la più recente disponibile).

Con ciascuna delle versioni succitate, abbiamo incontrato estrema resistenza da parte dello script `configure`, che pare avere un'avversione per qualunque versione della `glibc` o del compilatore utilizzati, siano essi precedenti o successivi.

Anche una volta scesi a compromessi con la configurazione, abbiamo incontrato crescenti ostacoli con il supporto per l'architettura `ppc40x`, con la generazione delle dipendenze e infine con la necessità di generare pezzi di codice tramite l'invocazione di `flex` e `bison`.

Al termine di una lunga serie di tentativi falliti, ammettiamo di aver rinuciato.

5.5 Installazione dei software necessari

Nel corso delle varie compilazione, le possibilità della piattaforma sono progressivamente aumentate, dandoci il potere di migliorare notevolmente i tempi d'installazione e manipolazione.

Così come disponibile all'inizio, la distribuzione ***MontaVista*** aveva la possibilità di rispondere via porta seriale, alla limitante velocità di `9600baud` e con la possibilità di trasferire un singolo file per volta.

Per installare un qualunque software era quindi necessario attendere l'arresto del sistema e lo spegnimento della piattaforma, quindi si doveva rimuore fisicamente la memoria ***CompactFlash***, montarla su un PC tramite l'adattatore, installare, riportare la memoria in sede e attendere il boot del sistema.

Ci viene in soccorso, in momenti come questo, un'importante variabile di `make`, `DESTDIR`. Supponendo che la ***CompactFlash*** sia stata montata su `/media/CompactFlash/`, con il comando

```
make DESTDIR=/media/CompactFlash install
```

è possibile installare un software che utilizzi la destinazione indicata come fosse la radice del `filesystem` di destinazione.

Si noti che il nome di questa variabile non è poi standard come si potrebbe desiderare: è opportuno controllare manualmente, esaminando il `Makefile`. Alcuni software utilizzano nomi diversi, come `INSTALLDIR` oppure `CHOST`.

Con questo metodo, sono stati installati ***dropbear*** e tutte le librerie necessarie al suo funzionamento. Di questo pacchetto fanno parte `zlib`, `openssl` e `openssh`, `scp` e `sftp`.

Da quel punto in poi, finalmente, il sistema residente è stato in grado – previa generazione di chiavi e successiva autenticazione – di rispondere via `ssh` tramite la connessione `Ethernet`, cosa che non soltanto ha reso tutte le comunicazioni più rapide, ma ha permesso – tramite l'uso di `sshfs` – di montare la ***CompactFlash*** in remoto.

Tramite un semplice comando come

```
sshfs root@192.168.1.1:/ /mount/remote
```

è infatti possibile montare un `filesystem` remoto, le cui operazioni di lettura e scrittura sono opportunamente mappate via `ftp`.

Il resto delle installazioni è stato quindi effettuato online, con notevole risparmio di tempo. Tra queste, il webserver ***LightTPD***, il motore per database ***SQLite*** e il motore per ***PHP*** e tutte le necessarie librerie precedentemente elencate.

5.6 Stato attuale del sistema

Due script, `/etc/init.d/lighttpd` e `/etc/init.d/dropbear`, sono stati opportunamente linkati nella famiglia di cartelle `/etc/rc.d/rc{1..6}.d/` perchè entrambi i server LightTPD e Dropbear partano automaticamente all'avvio del sistema e vengano arrestati allo spegnimento dello stesso.

Tramite alcune piccole modifiche allo script `/etc/rc.d/rc.local`, inoltre, il sistema setta l'hostname a `Zeke` e la propria scheda di rete sull'indirizzo `192.168.1.1/24`, con Dropbear attivo sulla porta `22` e LightTPD sulla porta `80`.

Nel suo stato attuale, il sistema dispone di due utenti:

- `root`, la cui password è `310ml`, amministratore responsabile del sistema
- `feng`, la cui password è `feng`, utente creato appositamente per l'esecuzione di ***LightTPD***

Infine, ipotizzando probabile la necessità di uno spazio disco superiore ai `4GB`, e trovandoci nella possibilità di farlo, abbiamo deciso di aggiungere un disco fisso IDE. In seguito all'installazione del pacchetto `e2fsprogs` abbiamo potuto montare e formattare il disco.

Attualmente, la configurazione di LightTPD è settata per utilizzare questo disco – montato come `/mnt/harddrive/` – come `document-root`. Su di esso sono state montati una pagina di stato e due applicazioni dimostrative, disponibili rispettivamente agli indirizzi

- `192.168.1.1/status.php`
- `192.168.1.1/trunaluten/`
- `192.168.1.1/faiv/`

6 Conclusioni

Come si è potuto evincere dalla corposità delle due precedenti sezioni, questo progetto non è stato soltanto ostico, ma ha presentato difficoltà non previste

che ci hanno costretto a porgere la nostra attenzione su problemi diversi da quelli attesi.

Le cause di tutti questi problemi sono da imputare fondamentalmente all'ammontare di ***legacy*** con il quale abbiamo avuto a che fare.

Innanzitutto, bisogna riconoscere l'età del sistema: non tanto per la parte hardware – la ***Virtex-II*** funziona ancora perfettamente – quanto alla suite ***software*** allegata. Essendo questa un prodotto commerciale, ha la tendenza ad invecchiare molto in fretta.

Una circostanza molto sfortunata ha visto infatti nascere la versione `9.x` come l'ultima di una serie – iniziata con la versione `6.x` – che basava l'implementazione dei ***bitstream*** su una gamma di ***core*** denominata `obp`. Tutte le versioni succesive, dalla diretta discendente `10.x` in poi, hanno fatto uso dei diversi ***core*** `xps`.

La nostra scheda `ML310` è quindi nata a cavallo tra due ere. Questo ha portato ad una spaccatura: durante le nostre ricerche per reperire materiale sull'argomento, è emerso come tutti i lavori simili al nostro siano stati completati basandosi su versioni del software molto precendenti oppure immediatamente successive alla nostra.

Altre cose nelle quali siamo incappati con una certa frequenza sono alcuni progetti simili, basati sulla ***XUP Virtex-II Pro***, una piattaforma realizzata da XILINX appositamente per l'ambito universitario.

Queste due piattaforme, la nostra `ML310` e la `XUPV2P` sono estremamente simili nell'aspetto, hanno un funzionamento analogo e sono supportate da versioni simili del software di sviluppo. Come abbiamo avuto modo di scoprire, però, sono completamente incompatibili a livello binario.

Il resto delle difficoltà è da imputare alla mancanza di libertà offerta dal kernel utilizzato. Essendo tutte le alternative valide delle distribuzioni commerciali (e quelle gratuite oltre la nostra portata per la compilazione, per via del `dts` mancante), l'unica ancora di salvezza è stata l'installazione ***MontaVista*** inclusa nel pacchetto.

In quanto commerciale, questa distribuzione è granitica per quanto riguarda gli aggiornamenti gratuiti. Questo ci ha costretti ad avere a che fare con diversi livelli di software obsoleto.

6.1 Bibliografia e siti consultati

Includiamo, infine, una lista delle fonti consultate nel corso del progetto.

Per il materiale ufficiale

- http://xilinx.com, sito ufficiale XILINX
- http://www.xilinx.com/support/documentation/ml310.htm per la documentazione sulla piattaforma in uso
- http://www.xilinx.com/products/boards/ml310/current/index.html per i molti design, README e l'immagine di ***MontaVista***
- http://www.xilinx.com/univ/xupv2p.html per la ***XUPV2P***

Durante i tentativi di compilazione del kernel:

- http://kernel.org/
- http://buildroot.uclibc.org/

Per ***crosstool*** e il suo utilizzo:

- http://kegel.com/crosstool
- http://buffalo.nas-central.org/index.php/Build_(and_use)_a_cross-toolchain_using_Dan_Kegel's_crosstool_0.38

Per le librerie e i software installati, tra i tanti

- http://lighttpd.net
- http://matt.ucc.asn.au/dropbear/dropbear.html
- http://php.net
- http://gcc.gnu.org