Driver Linux per carrier TRAM

E.Giani[1],C.Baffa[1],A.Checcucci[1]

[1]Osservatorio Astrofisico di Arcetri

Arcetri Technical Report N° 4/99
Firenze 1999

Sommario

Nel presente rapporto viene documentata la riscrittura del device driver Linux per le schede di interfaccia tra ISA e PCI e le reti di transputer.

1 Introduzione

Nelle prime fasi di lavoro, la camera infrarossa NICS sarà gestita da un sistema di laboratorio che condivide, con il sistema definitivo, l'elettronica di piano focale([1]).
Il sistema di laboratorio è costituito essenzialmente da un PC con sistema operativo Linux e da una scheda transputer TMB-17 della Transtech ([2]) che si connette al bus PCI del PC.
Mentre la ditta Transtech fornisce i device drivers per il funzionamento della scheda sotto i sistemi operativi DOS e Win95, non esiste al momento un device driver Linux specifico per la stessa scheda, ma solo un driver sviluppato da Christoph Niemann[1] per le schede transputer B004 compatibili (vedi §2).
Parte del lavoro da noi svolto ed il cui resoconto sarà descritto in questo rapporto tecnico, riguarda le modifiche che abbiamo effettuato al driver Linux di Niemann per adattarne il funzionamento alla scheda TMB-17 della Transtech.
La rimanente parte del lavoro ha compreso la parziale riscrittura della parte di codice del driver di Niemann, che riguarda il supporto delle schede transputer B008 compatibili[2](vedi §2).

2 Schede TRAM B004 e B008 compatibili

Lo standard più comune per una scheda transputer per macchine PC AT, viene comunemente denominata *interfaccia B004*: la scheda transputer B004 è stata una dei primi prodotti della ditta Inmos/SGS-Thomson ed è diventata di fatto uno standard.
Una scheda TRAM[3] viene detta B004 compatile, quando possiede la stessa mappa dei registri della scheda Inmos B004. Alla stessa famiglia di schede TRAM della Inmos, appartengono le schede madri IMS B008([3]). Queste rappresentano un'evoluzione delle antecedenti schede B004 in quanto alle funzionalità di quest'ultime aggiungono la possibilità di configurare la scheda per con il supporto per il DMA

[1] i file sorgenti di questo driver sono disponibili liberamente in rete

[2] l'autore stesso avverte gli utenti della possibilità che il supporto per le schede B008 compatibili non sia perfettamente implementato.
Abbiamo potuto provare tale codice su una scheda IMS B008 su bus ISA ed abbiamo sperimentato personalmente la non corretta funzionalità di tale supporto.

[3] TRAM è l'abbreviazione di Transputer Module: un processore ed una certa quantità di RAM vengono montati su una scheda di piccole dimensione con un'interfaccia standard (TRAM). Le schede madri TRAM possono allocare diversi moduli di questo tipo.

e con la generazione di interrupt in corrispondenza del verificarsi di alcuni eventi specifici.
Oltre ad avere gli stessi registri delle schede TRAM B004 (in modo da mantenerne la compatibilità con il software sviluppato), le schede B008 hanno tre nuovi registri che forniscono il supporto per le nuove funzionalità della scheda.
L'interfaccia delle schede IMS B004/B008 con il link 0 del *root* transputer (TRAM 0) è costituita da un chip IMS C012 che esegue le conversioni parallelo/seriale tra una porta bidirezionale ad 8 bit da una parte ed il link transputer seriale dall'altra.
La mappa dei registri per le schede B004/B008 compatibili viene riportata in Appendice A.

2.1 Scheda TMB-17 della Transtech

L'hardware transputer usato per il progetto di laboratorio è basato su una scheda transputer TMB-17 della Transtech[4] che si connette alla scheda madre del PC per mezzo di uno slot PCI. L'interfaccia tra il bus PCI ed il link transputer è costituita da un controller PCI, un buffer FIFO a 32 bit e da un chip IMS C012.
Secondo le specifiche tecniche della casa produttrice, il controller PCI fornisce oltre ad una configurabilità della scheda, una completa compatibilità con le schede madri TRAM B004/B008.
In realtà[5] la scheda TMB-17 risulta compatibile solamente con le schede TRAM B004: la TMB17 supporta la sola mappa dei registri della scheda B004 alla quale si aggiungono quattro registri specifici del buffer FIFO, mentre non ha le estensioni B008.
In appendice B riportiamo la mappa completa dei registri della TMB-17.

3 Le funzioni dei registri di interfaccia

Le diverse schede transputer presentano una mappa completa dei registri di I/O che dipendente in modo peculiare dalle caratteristiche aggiuntive che esse posseggono rispetto all' interfaccia standard B004.
Di seguito illustreremo le funzionalità dei registri di I/O delle schede TRAM B004 compatibili, B008 compatibili e TMB-17.

[4] i prodotti della ditta Transtech sono compatibili con i prodotti Inmos/SGS-Thomson

[5] abbiamo ricevuto conferma dal supporto tecnico della Transtech

3.1 Registri base delle schede TRAM B004 compatibili

I registri base dell'interfaccia standard B004 sono in totale sei (vedi Tab. A1 in App. A). Dei primi quattro, la prima coppia (*C012 input data register* e *C012 output data register*) consente di accedere ai dati in entrata ed in uscita sul link: tali registri vengono letti o scritti un byte alla volta. La seconda coppia (in ogni registro ci sono due soli bit significativi) consente invece di determinare lo stato del link: pronto per la lettura (*C012 input status register*) oppure pronto per la scrittura (*C012 output status register*).
Gli altri due registri (*Reset/Error register* e *Analyse register*: di ognuno un solo bit è significativo) forniscono un modo per accedere ai servizi sui transputer connessi al carrier TRAM.

3.2 Registri delle schede TRAM B008 compatibili

Le schede B008 compatibili hanno tre registri aggiuntivi oltre (vedi Tab. A1 in App. A).
La scrittura di questi registri permette di controllare rispettivamente la generazione di un interrupt (*Interrupt enable register*) in corispondenza di quattro eventi distinti, di configurare la direzione di trasferimento dei dati con il DMA (*DMA request register*) e di assegnare, all' interno di un gruppo limitato di possibili valori, l'IRQ e il canale DMA utilizzato dalla scheda TRAM (*DMA and interrupt channel select register*)[6].

3.3 Registri della scheda TRAM TMB-17

La scheda TMB-17 oltre ai sei registri base che ne garantiscono la compatibilità con la B004, possiede quattro registri relativi al buffer FIFO (vedi Tab. B1 in App. B).
Due di questi registri consentono di leggere (*Rx Data 8/16/32*) o scrivere (*Tx Data 8/16/32*) nel buffer FIFO due o quattro bytes alla volta, garantendo in questo modo rendimenti superiori del sistema rispetto all'uso in modalità B004 compatibile.
Per assicurare un'operazione valida di I/O, il numero di bytes effettivamente disponibili in lettura o liberi per la scrittura nel buffer FIFO, viene determinato dal valore degli altri due registri: *Rx FIFO Level* e *Tx FIFO Level*.

[6] quest'ultimo registro non è presente su tutte le schede B008 compatibili, perché in alcuni casi la configurablità dell'IRQ e del DMA viene effettuata attraverso il posizionamento sulla scheda madre dei dip-switch.

4 Device driver per Linux

Come altri sistemi operativi, Linux supporta hardware di vario tipo gestito mediante i *device driver*. Per ogni tipo di periferica installata sul sistema, il kernel di Linux deve includere al suo interno un *device driver*, cioè una parte di codice che esegue le operazioni di controllo e di I/O essenziali al funzionamento del dispositivo: il device driver costituisce perciò un'interfaccia tra il dispositivo hardware ed il software applicativo.
Per rendere il sistema operativo più snello e funzionale, il kernel di Linux è strutturato in maniera modulare. Invece di riservare il controllo del sistema ad un unico blocco monolitico, la gestione di molte periferiche (stampanti, schede di rete, adattatori etc.) viene comunemente controllata da moduli caricabili, ciascuno dei quali implementa, in genere, un unico device driver. Ciascun modulo è costituito da un codice oggetto che può essere installato e disinstallato dinamicamente dal kernel utilizzando i comandi *insmod* e *rmmod*. Tale sistema risulta più flessibile perché durante la sua esecuzione possono essere aggiunte al kernel nuove funzionalità che possono essere successivamente rimosse, se non più necessarie.

4.1 Device files

In tutte le versioni di Unix, compreso Linux, i dispositivi hardware sono visti dai programmi applicativi come file speciali, detti file di dispositivo (*device files*), localizzati nella directory */dev*.
Sia il device (o più precisamente il file ad esso associato) sia le operazioni eseguibili su di esso, sono identificati all'interno del kernel da due strutture distinte di dati. La prima[7] definisce le caratteristiche che descrivono in modo completo l'oggetto su cui il driver lavora, cioè il file di dispositivo. Questa struttura di dati viene generata dopo che una chiamata di sistema *open()* è stata eseguita sul device file e viene successivamente rilasciata al momento della chiusura del file di dispositivo in seguito alla chiamata di sistema *close()*.
La seconda struttura[8] descrive l'insieme completo di operazioni o metodi eseguibili dal driver sul device file. A ciascun campo di questa struttura viene associata, nel kernel, una funzione che gestisce l'operazione relativa: ad esempio in corrispondenza dei metodi `open()`, `read()`, `write()`, etc. il device driver scrive i puntatori alle funzioni di apertura, lettura, scrittura, etc. del device file. Questa struttura

[7] struct file
[8] struct file_operations

costituisce essenzialmente una lista delle funzioni che devono essere definite per creare il device driver.

5 Struttura di un device driver

Creare un device driver coinvolge la scrittura di alcune funzioni e la registrazione di queste all'interno del kernel. In questo modo quando si accede al device file associato al dispositivo gestito dal driver, il kernel è in grado di eseguire le funzioni definite.
Quando il driver viene implementato come modulo il codice deve contenere almeno due task principali: *init_module()* e *cleanup_module* (vedi §4) che vengono rispettivamente "attivati" dai comandi *insmod* e *rmmod*.
I passaggi necessari per la creazione di un device driver sono perciò sostanzialmente due. Il primo consiste nel definire le due funzioni `init_module()` e `cleanup_module()`.
Il compito svolto da `init_module()` comprende l'inizializzazione del dispositivo controllato dal driver e, qualora individuato correttamente, la registrazione delle risorse di sistema necessarie per il corretto funzionamento dell'hardware. Il ruolo di `cleanup_module()` è quello invece di chiudere il dispositivo e rilasciare le risorse di sistema precedentemente allocate.
Il secondo passo consiste nello scrivere il codice delle funzioni supportate dal driver: ognuna viene registrata in corrispondenza del relativo campo della struttura `struct file_operations`.
Linux fornisce inoltre molte routine di servizio che i device drivers usano comunemente.
Per maggiori dettagli rimandiamo alla documentazione relativa [4].

5.1 Inizializzazione dell'hardware

L'inizializzazione hardware di un dispositivo di interfaccia implica generalmente l'assegnazione di un insieme di risorse del sistema come:

- l'indirizzo base dello spazio di I/O del PC in cui viene mappato l'hardware fisico (registri)
- il canale IRQ
- il canale DMA.

In un sistema normale parte di queste risorse sono già allocate, per cui è necessario individuare quelle disponibili e tra queste quelle che sono supportate dall'hardware da installare.

Dal punto di vista della scrittura del driver la peculiarità delle schede PCI rispetto a quelle ISA, rilevante per la fase iniziale di riconoscimento e configurazione del dispositivo, è la presenza del supporto per l'autorilevamento della scheda. Le schede PCI vengono automaticamente riconosciute durante l'operazione di boot: le informazioni relative alla configurazione sono presenti nel bios della scheda ed il driver vi può accedere attraverso l'uso di funzioni interne del kernel. Il riconoscimento automatico delle schede PCI evita problemi di conflitti hardware tra i diversi dispositivi presenti. Tale problema può invece manifestarsi per le schede ISA, la cui configurazione, spesso, attraverso jumper è a carico dell'utente. Per individuare la presenza di una scheda ISA, il device driver deve individuare le risorse di sistema allocabili dal dispositivo hardware e se disponibili, assegnarle.
La fase di *probing* può risultare critica se prima non viene eseguito un controllo sulle porte di I/O i canali IRQ e DMA già in uso da parte di altri dispositivi hardware.

5.2 Lettura e scrittura di un file di dispositivo

Il ruolo fondamentale di un device driver è quello di fornire un metodo per lo scambio di dati tra il dispositivo e lo spazio di memoria o i registri del processore.
Il modo con cui la richiesta di trasferimento viene eseguita consente di classificare i dispositivi fisici in due categorie principali: i dispositivi a caratteri ed i dispositivi a blocchi. Noi prenderemo in considerazione solo i primi.
I dispositivi a caratteri possiedono funzioni di lettura e scrittura che vengono chiamate direttamente tutte le volte che un'operazione di I/O è richiesta attraverso le chiamate di sistema `read()` o `write()`. Le funzioni di lettura e scrittura assumono quindi, per i dispostivi a caratteri, un ruolo fondamentale all'interno del codice del device driver. La funzionalità e l'efficienza del device dipende dal metodo con il quale vengono implementate le funzioni di I/O nel driver. In generale esistono due metodologie distinte per trasferire i dati in applicazioni di acquisizione: l'I/O a controllo di interrupt e l'I/O a controllo di programma (o `polling`) [9].
Nel primo caso il dispositivo tramite l'interfaccia che lo gestisce, deve essere in grado di generare un *interrupt* ogni volta che è disponibile ad un'operazione di I/O. Le interruzioni generate dall'hardware impongono una sospensione momentanea del processo che la CPU sta eseguendo in quel momento e costringono quest'ultima a porre in ese-

[9] la scelta tra i due metodi dipende ovviamente anche dalle specifiche dell'hardware utilizzato

cuzione la routine di servizio dedicata alla gestione dell'evento verificatosi. Una volta terminata la routine di servizio dell'interrupt, il processo rientra nel flusso di esecuzione originale.
Con il *polling*, invece, la CPU controlla ad intervalli regolari lo stato del dispositivo di I/O per accertarsi della disponibilità dei dati. Lo svantaggio di questo metodo è ovvio: il processore è pesantemente occupato con il processo di polling rallentando gli altri processi eseguiti in background.
A questi due metodi se ne affianca un terzo che ricorre all'Accesso Diretto alla Memoria (*Direct Memory Access* o DMA). Questo è un meccanismo capace di trasferire un blocco di dati sfruttando l'intera larghezza di banda del bus tra un *device* di I/O e la memoria del PC senza alcun intervento da parte della CPU.
La tecnica che ricorre all'uso del DMA è intrinsecamente più veloce rispetto al controllo di programma e di interruzione poichè il processore non viene coinvolto direttamente durante l'operazione di trasferimento.
Attualmente, dispositivi su bus PCI raggiungono velocità di trasferimento decisamente elevate (133 Mbyte/s), per cui talvolta risulta superfluo ricorrere all' uso del DMA.

6 Il driver Linux per le schede INMOS-B004/B008 di C.Niemann

Il device driver di C. Niemann per il supporto sotto Linux delle schede INMOS B004/B008 su bus ISA, è stato scritto per il kernel Linux 1.3.72 e successive versioni inferiori alla 2.0.
Il codice del driver è costituito da un file sorgente (*link.c*) e da tre include file (*link.h*, *link-common.h* e *link-ioctl.h*) in cui sono definite le macro e le variabili globali.
Per quanto detto nel §.5 la struttura del device driver include i due task `init_module()` e `cleanup_module()` ed il codice delle funzioni corrisponenti alle operazioni supportate dal device.
Le routine `init_module()` e `cleanup_module()` si appoggiano su una serie di funzioni sia interne al kernel, sia definite nel codice sorgente del driver, per eseguire il riconoscimento, la configurazione e la successiva chiusura del link transputer.
Oltre a queste vengono definiti solo alcuni dei metodi complessivamente registrabili della struttura `struct file_operations`. Infatti non tutte le funzionalità devono necessariamente essere implementate; a seconda del tipo di hardware e del tipo di operazioni che il software applicativo deve compiere su di esso.

Nel caso particolare, il device driver registra sei operazioni effettuabili sul file di dispositivo da parte dell' applicazione utente, in corrispondenza dei seguenti metodi: `lseek()`, `read()`, `write()`,`open()`, `release()`, `ioctl()`. Di queste ci limiteremo a prendere in considerazione le funzioni preposte all'operazione di lettura e scrittura del link transputer perché sono quelle che coinvolgono più direttamente le differenti caratteristiche delle schede transputer prese in considerazione.

6.1 link_read() e link_write()

Il driver scritto da Niemann ricorre alla tecnica del controllo di programma per effettuare la lettura e scrittura del link transputer. Questo metodo è l'unico utilizzabile dalle schede B004 compatibili, in quanto il loro hardware non supporta la gestione a controllo di interrupt, nè il trasferimento DMA.
Le interfacce INMOS B004 (vedi §.3.1) presentano quattro registri relativi alle operazioni di I/O sul link: questi registri vengono utilizzati dal device driver per comunicare con il link transputer.
Una chiamata di sistema `read()` o `write()` eseguita sul file associato al link transputer[10] da parte dell' applicazione, attiva le corrispondenti routine del device driver[11]. Queste due funzioni presentano la stessa struttura logica (concorde con la tecnica di gestione a controllo di programma): esaminano ad intervalli regolari i registri di stato analizzandone il valore. Se questo segnala la disponibilità di dati sul link in entrata o in uscita, allora il corrispondente registro di dati viene letto o scritto, altrimenti si attende un intervallo di tempo dopo il quale l'operazione viene ripetuta. Per un corretto funzionamento del sistema complessivo le funzioni di I/O implementano un timeout e durante i cicli di attesa il tempo di CPU viene rilasciato a beneficio degli altri processi in esecuzione.
Le operazioni di I/O sui registri di dati vengono eseguite un byte alla volta: operazioni di lettura e scrittura che convolgono un numero elevato di byte, impongono un uso intensivo della CPU.

[10] il device file è /dev/link0 con numero maggiore 24 e numero minore 0.

[11] in questo caso le funzioni relative alle due chiamate di sistema sono: link_poll_read() e link_poll_write().

7 Modifiche del driver per il supporto delle funzionalità della scheda TMB-17

La scheda TRAM TMB-17 su bus PCI, implementa rispetto all'interfaccia standard B004 un buffer FIFO aggiuntivo a cui si può accedere attraverso i relativi registri fino a 4 bytes per volta. Questa caratteristica permette di accrescere notevolmente la velocità di trasferimento dei dati da e verso il link transputer.
Per poter sfruttare queste nuove specifiche, abbiamo eseguito alcune modifiche al codice del driver relativo alla funzione di lettura del link transputer. Inoltre abbiamo ritenuto opportuno riscrivere le routine di inizializzazione dell'hardware in modo da includere nel driver la possibilità di individuare la presenza sia di schede TRAM ISA (come le INMOS B004 e B008) sia PCI: in questo modo il driver sviluppato comprende il supporto per entrambe le interfacce.
Di seguito descriviamo dettagliatamente il funzionamento delle routine modificate.

7.1 Descrizione delle routine d'inizializzazione dell'hardware

Il compito di individuare l'hardware transputer viene svolto dalla funzione `link_find_hw()` chiamata da `init_module()`. La funzione `link_find_hw()` esegue due compiti distinti:
- registra in corrispondenza del driver il numero maggiore che consente di individuare il driver associato con il dispositivo
- chiama la funzione di rilevamento dell'hardware `link_detect()`.
Questa si appoggia a due funzioni distinte: la prima, `link_detect_pci()`, effettua la ricerca di eventuali schede transputer su bus PCI. Qualora non ne venga individuata alcuna, viene chiamata la seconda funzione, `link_detect_isa()`, che cerca di individuare la presenza di una eventuale scheda transputer sul bus ISA. Se anche questo tentativo fallisce, viene segnalato un messaggio di errore.
Come accennato nel §5.1, i problemi maggiori durante il rivelamento dell'hardware sorgono per le schede ISA, in quanto è neccessario individuare quale, tra gli indirizzi base a cui può essere mappato l'hardware,sia quello giusto: la scrittura o lettura di porte di I/O appartenti ad hardware diverso, può determinare infatti un blocco del sistema.
Per evitare questa situazione, abbiamo seguito le indicazioni suggerite in [4].
Il codice della routine `link_detect_isa()` esegue prima un esame degli indirizzi di base a cui può essere configurato l'hardware transputer

per determinare se le relative porte sono già occupate: esiste un numero limitato di indirizzi base di I/O a cui una specifica intefaccia ISA viene indirizzata, per cui la fase preliminare si riduce al controllo di un numero limitato di combinazioni.
Questo tipo di operazione non è invece necessaria per le schede ad interfaccia PCI in quanto le informazioni necessarie, come appunto l'indirizzo base della scheda e il canale IRQ assegnato, sono contenute nello spazio di configurazione a cui la funzione `link_detect_pci()` accede utilizzando le funzioni interne del kernel Linux.
L'inizializzazione hardware delle schede transputer ISA e PCI si differenzia perciò solo nella fase preliminare di rilevamento: una volta individuata la presenza dell'hardware transputer l'operazione successiva svolta dalla funzione `link_detect()` consiste nel registrare le porte di I/O, riservando l'indirizzo base a cui è stata individuata la scheda e l'intervallo di I/O.
A questo punto il modulo risulta installato correttamente e pronto per eseguire le funzioni richieste dall' applicazione.

7.2 Descrizione della routine di lettura

Come accennato nel §2.1, la scheda TMB-17 non include il supporto hardware per la generazione degli interrupt, per cui anche in questo caso, come per le schede INMOS B004, dobbiamo ricorrere al metodo del polling per le operazioni di I/O con la significativa differenza che la presenza del buffer FIFO permette di effettuare letture e scritture di 4 byte alla volta invece di uno solo (occupando l'interezza del bus PCI).
Dal punto di vista del codice, questo ha significato modificare la routine di lettura (`link_poll_read()`) in modo che, invece di leggere il registro di stato, venga esaminato prima il registro contenente l'informazione sul numero di byte disponibili in lettura. Se il numero contenuto nel registro è non inferiore a 4, è possibile effettuare una lettura o scrittura multibyte ed i primi quattro byte presenti nel buffer FIFO vengono letti (o scritti).
In caso contrario la funzione ha un comportamento del tutto analogo a quello descritto al §6.1.

8 Implementazione funzionante del trasferimento DMA per le schede B008 compatibili

Il driver fornito da C. Niemann non risulta funzionante quando la modalità B008 compatibile (cioè il supporto per il DMA) viene attivata. Abbiamo perciò deciso di individuarne i problemi e correggerli.
La procedura necessaria per la gestione a controllo DMA generalmente consiste in due task distinti: la programmazione del dispositivo di I/O per l'uso del DMA e la programmazione del controller DMA (DMAC).

8.1 Programmazione dei registri di I/O della scheda IMS B008 per il trasferimento DMA

La programmazione del device per i trasferimenti con il DMA consiste nella configurazione di alcuni bit dei registri di I/O del dispositivo in modo che venga generata una richiesta quando il dispositivo è disponibile ad un trasferimento di dati.
La scheda IMS B008 possiede tre registri la cui programmazione consente di configurare rispettivamnete il canale DMA, l'IRQ, la direzione del trasferimento (*da* o *verso* la memoria del PC) e la generazione di un interrupt in corrispondenza della fine del trasferimento DMA.
In Appendice C riportiamo le tabelle che illustrano le funzionalità dei tre registri e le specifiche per la loro configurazione.
La registrazione del canale DMA ed IRQ viene eseguita durante la fase di inizializzazione hardware dalla routine `link_detect()`. Il kernel fornisce alcune routine di I/O per l'accesso agli interrupt ed ai canali DMA: abbiamo implementato queste funzioni all' interno della routine `link_detect()` con l'accortezza che se una delle due risorse non può essere assegnata, la scheda transputer viene fatta lavorare come una scheda B004 compatibile, cioè a controllo di programma.
Per quanto riguarda le routine di lettura e scrittura, queste sono state scritte da Niemann in modo da supportare sia il polling per il funzionamento in modalità B004 compatibile, sia il DMA per il funzionamento B008 compatibile: in particolar la funzione `link_read()` chiama la routine `link_polling_read()` nel primo caso e `link_dma_read()` nel secondo. Discorso analogo per la funzione `link_write()`.
Le routine di lettura e scrittura con il DMA hanno una struttura logica analoga: la differenza sostanziale consiste nella direzione in cui i dati vengono trasferiti e nei registri di I/O da configurare. Le modifiche che abbiamo apportato ad entrambe le funzioni, sono perciò molto simili.

Entrambe le funzioni eseguono come prima operazione una chiamata alla funzione `link_setup_dma()` che (vedi §8.2) configura opportunamente i registri del DMAC. Successivamente vengono programmati i due registri di interfaccia (*Interrupt enable register* e *C012 status register*). La configurazione del primo abilita la generazione da parte dell'hardware di un segnale di interrupt al termine del trasferimento dati con il DMA. La programmazione del secondo dà origine, in presenza di un dato valido nel corrispondente registro, ad un segnale che viene riconosciuto dalla logica del DMA come una richiesta di trasferimento.
Il trasferimento dei dati da o verso la memoria viene effettuato successivamente, dopo l'abilitazione del canale DMA e dopo la scrittura del registro di richiesta DMA (*DMA request register*) la cui configurazione determina la direzione del trasferimento.
L'operazione di trasferimento viene regolata da una funzione di temporizzazione che implementa un timeout. Nell'intervallo di tempo durante il quale viene effettuato il trasferimento dei dati tra il dispositivo e la memoria, il processo rilascia il controllo della CPU: in particolare con la chiamata alla funzione interna `interruptible_sleep_on()` il processo viene posto a "dormire" in una coda di attesa dalla quale viene rimosso quando "risvegliato" dal verificarsi di uno dei seguenti eventi distinti:

- l'interrupt generato alla fine del trasferimento dei dati
- la scadenza del timeout

Ad entrambi gli eventi sono associate due funzioni distinte[12] i cui compiti consistono nel risvegliare il processo di I/O per mezzo della chiamata della funzione interna `wake_up_interruptible()`, disabilitare il canale DMA e segnalare il tipo di evento verificatosi.
Una volta "risvegliato", il processo di lettura o scrittura inizializza di nuovo i registri di I/O precedentemente programmati e trasferisce i dati all'applicazione utente.

8.2 Programmazione del controller DMA

La programmazione del controller DMA integrato nel chipset della scheda madre viene eseguita dalla routine *link_setup_dma()*: il tipo di trasferimento è individuato dai parametri della funzione di I/O chiamante.
La programmazione del DMAC è notevolmente semplificata poichè il kernel Linux fornisce una classe di funzioni interne che eseguono la

[12] link_interrupt() e link_times_out()

configurazione dei registri del controller DMA relativi al canale selezionato.
I passi per la configurazione del controller DMA sono fissi: in [4] viene fornita la lista di funzioni e l'ordine con cui queste devono essere chiamate.
In particolare prima della scrittura dei registri del DMAC, la funzione di configurazione disabilita gli interrupt ed il canale DMA. Successivamente viene programmata la direzione di trasferimento che stabilisce se il canale viene letto o scritto dal dispositivo, viene registrato l'indirizzo del buffer di memoria utilizzato per contenere i dati ed infine viene assegnato il numero di byte da trasferire.
Una volta eseguite queste operazioni, gli interrupt vengono nuovamente abilitati. L'attivazione del canale DMA, invece, non è eseguita dalla funzione di configurazione, ma da quella chiamante (vedi §.8.2) nel momento in cui il dispositivo che genera la richiesta DMA è pronto a gestirla. Se il canale DMA viene abilitato mentre lo stato della linea di richiesta è sconosciuto, il controller può iniziare a trasferire i dati prematuramente.

9 Conclusioni

Il lavoro descritto in questa rapporto tecnico ha avuto come obiettivo lo sviluppo di un driver per il supporto in ambiente Linux di una scheda transputer TMB-17 della ditta Transtech.
Abbiamo preso come punto di partenza il driver Linux sviluppato da C.Niemann per la gestione delle schede TRAM INMOS B004/B008 ed abbiamo eseguito su di esso una serie di modifiche in modo da implementare le nuove specifiche della scheda che abbiamo in dotazione.
Siamo inoltre riusciti a modificare con successo la funzionalità del driver originale relativa al supporto DMA per le schede B008 compatibili.
Il codice del driver che abbiamo sviluppato, include il supporto completo per le schede B004/B008 compatibili ed inoltre corregge alcuni problemi di portabilità del driver per versioni del kernel comprese nel range 1.2-2.0.

10 References

[1] C.Baffa,G.Comoretto "Il software TransNix per la camera Infrarossa Nics", Arcetri Technical Report n.3/1996
[2] "Transputer Motherboard User Manual",Transtech Parallel Systems
[3] "IMS B008 User Guide and reference manual", INMOS Limited
[4] A.Rubini "Linux Device Drivers",O'Reilly
[5] H.Messmer,"The Indispensable PC Hardware Handbook",Addison-Wesley Pub.Co

11 Appendice A

Base Address + (hex)	**Register**
00h	C012 input data register
01h	C012 output data register
02h	C012 input status register
03h	C012 output status register
10h	reset error register
11h	analyse register
12h (B008 only)	DMA request register
13h (B008 only)	Interrupt enable register
14h (B008 only)	DMA and interrupt channel select register

Tab. A1 Mappa dei registri delle schede IMS B004/B008

12 Appendice B

Base + (hex)	**Description**	**Access Type**
00h	Rx Data 8 bit	Byte read only
01h	Tx Data 8 bit	Byte Write only
02h	Rx Status	Byte Read only
03h	Tx Status	Byte Read only
08h	Rx Data 8/16/32 bit	Byte/Word/DWord Read
0Ch	Tx Data 8/16/32 bit	Byte/Word/DWord Write
10h	Reset write/Error read	Byte Read/Write
11h	Analyse register	Bute read
18h	Rx FIFO Level	Word Read
1Ch	Tx FIFO Level	Word Write

Tab. B1 Mappa dei registri della scheda transputer TMB-17

13 Appendice C

Bit No.	Corresponding event
0	Dma end interrupt enable
1	Error interrupt enable
2	Link output interrupt enable (OutputInt asserted)
3	Link input interrupt enable (InputInt asserted)

Tab. C1 Funzioni corrispondenti ai bit dell'Interrupt enable register

Bit 0	Transfer direction
0	Transfer from PC memory to the link adaptor output data register
1	Transfer from the link adaptor input data register to PC memory

Tab. C2 Configurazione della direzione del trasferimento DMA

Bit No.	Corresponding event
0	IRQ channel select bit least significant bit
1	IRQ channel select bit most significanr bit
2	DMA channel select bit least significant bit
3	DMA channel selct bit most significant bit

Tab. C3 Funzioni corrispondenti ai bit del registro DMA e dell'interrupt channel register

Bit 1	Bit 0	IRQ channel
0	0	3
0	1	5
1	0	11
1	1	15

Tab. C4 Selezione del canale IRQ

Bit 1	Bit 0	DMA channel
0	0	0
0	1	1
1	0	DMA disabled
1	1	3

Tab. C5 Selezione del canale DMA