Progettino di Calcolatori Elettronici

C.d.L. in Ingegneria Informatica, Ordinamento DM 270, emergenza COVID-19

8 luglio 2020

1. Colleghiamo al sistema una perifierica PCI di tipo `ce`, con vendorID `0xedce` e deviceID `0x1234`. Le periferiche `ce` sono simili a piccoli hard disk e sono in grado di operare in PCI Bus Mastering autonomamente.

   Ogni periferica contiene un certo numero di settori, ciascuno grande `DIM_SECT`, numerati a partire da 0, ed è in grado di trasferire un settore alla volta da o verso la memoria.

   Mentre è in corso una operazione la periferica non è in grado di avviarne un'altra. Al termine dell'operazione in corso la periferica invia una richiesta di interruzione. La lettura del registro di stato fa da risposta alla richiesta di interruzione. Le interruzioni sono sempre abilitate, ma la periferica non invia uteriori richieste finché non ha ricevuto risposta all'ultima inviata.

   Ogni periferica `ce` usa 16 byte nello spazio di I/O a partire dall'indirizzo base specificato nel registro di configurazione BAR0, sia b. I registri accessibili al programmatore sono i seguenti:

   1. **SNR** (indirizzo b, 4 byte): Sector NumberR; il numero del settore che si vuole leggere o scrivere;
   2. **CMD** (indirizzo $b + 4$, 4 byte): CoMmanD; scrivendo 1 in questo registro si avvia una operazione di scrittura (trasferimento dall'indirizzo ADR della memoria al settore SNR); scrivendo 2 si avvia una operazione di lettura (trasferimento dal settore SNR verso l'indirizzo ADR della memoria);
   3. **STS** (indirizzo $b + 8$, 4 byte): STatuS, di sola lettura; nei 16 bit meno significativi contiene il numero di settori che la periferica implementa;
   4. **ADR** (indirizzo $b + 12$, 4 byte): ADdRess; indirizzo di memoria a partire dal quale la periferica deve leggere (se CMD vale 1) o scrivere (se CMD vale 2);

   Vogliamo permettere all'utente di leggere o scrivere dalle periferiche `ce` come se conteneressero un'unica sequenza di byte, numerati a partire da 0, ignorando la suddivisione in settori. Per esempio, assumiamo che `DIM_SECT` sia 32 e che l'utente chieda di leggere 30 byte a partire dal byte numero 42: la primitiva di lettura restituirà gli ultimi 22 (cioè $32-(42 \,\%\, 32)$) byte del settore numero 1 e i primi 8 (cioè $(42+30) \,\%\, 8$) del settore numero 2. Si noti che, per fare questo, la primitiva dovrà necessariamente trasferire i settori in un suo buffer interno, da cui poi estrarrà solo i byte richiesti dall'utente.

   Tutte le periferiche di tipo `ce` presenti nel sistema sono identificate durante la fase di inizializzazione e vengono numerate a partire da 0.

   Aggiungiamo dunque le seguenti primitive (abortiscono il processo in caso di errore):

   - `bool ceread(natl id, natl off, char *buf, natl quanti)`: legge `quanti` byte dalla periferica `ce` numero `id` e li copia nel buffer puntato da `buf`; non è richiesto che tale buffer sia residente o condiviso; è un errore se la periferica `id` non esiste; la primitiva restituisce `false` se qualcuno dei i byte $[\texttt{off}, \texttt{off} + \texttt{quanti})$ cade al di fuori di quelli effettivamente contenuti nella periferica;
   - `bool cewrite(natl id, natl off, char *buf, natl quanti)`: scrive `quanti` byte nella periferica `ce` numero `id`, prendendoli dal buffer puntato da `buf`; non è richiesto che tale buffer sia residente o condiviso; *solo* i byte indicati devono essere modificati all'interno della periferica; è un errore se la periferica `id` non esiste; la primitiva restituisce `false` se qualcuno dei byte $[\texttt{off}, \texttt{off} + \texttt{quanti})$ cade al di fuori di quelli effettivamente contenuti nella periferica;

Modificare i file `io.s` e `io.cpp` in modo da realizzare la primitiva come descritto. Controllare eventuali problemi di Cavallo di Troia.