Implementazione in VHDL di un

Modulo per la gestione della tastiera

Su scheda Xess XSA-50

Christian Gregorutti

1 Introduzione

Il presente progetto implementa un modulo per la comunicazione via porta PS2 tra una tastiera (PS2 o AT/XT) e la scheda Xess XSA-50. Questo lavoro si appoggia in parte a quello svolto dalla'universitá del Queensland (*http://www.uq.edu.au/*): per i dettagli si veda la sezione 3.

L'idea che sta alla base è quella di avere un modulo duttile che possa interfacciarsi ad un'eventuale logica presente sull'FPGA fornendole valori (numeri o stringhe) suddivisi per parametri.

Nella sezione 2 viene presentato in sintesi il protocollo PS2 (conoscenza imprescindibile per capire il funzionamento del progetto), mentre nella sezione 3 verranno presi in esame i dettagli dell'implementazione.

2 Il protocollo ps2

2.1 Introduzione

Il PS2 è un protocollo seriale, sincrono e bidirezionale sviluppato dall'IBM e ampiamente usato al giorno d'oggi per far comunicare PC con tastiere e mouse.

La scheda XSA-50 ha in dotazione una porta PS2 a 6-pin Mini-DIN. Per le porte PS2 esiste anche un altro connettore, il 5-pin DIN (AT/XT): i due connettori sono (dal punto di vista elettrico) perfettamente compatibili (il protocollo è sempre il PS2, l'unica differenza sta nella disposizione dei pin), per cui nel caso si possedesse una tastiera con connettore a 5-pin è sufficiente l'uso di un adattatore.

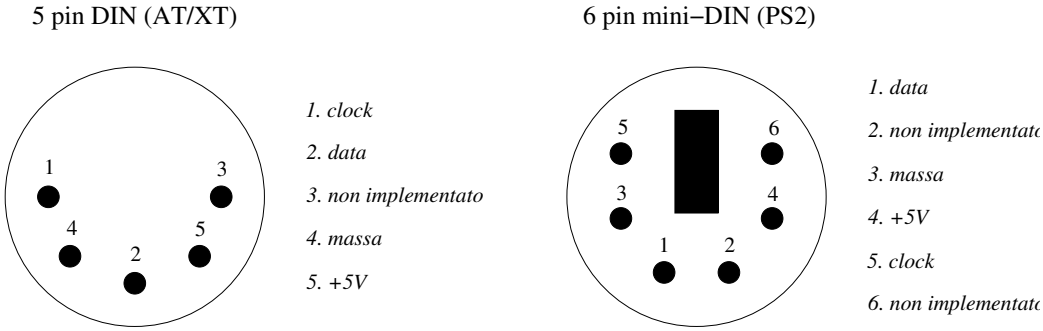


Figura 1: Connettori maschi per porte PS2

2.2 La comunicazione

Poiché tramite il protocollo PS2 è possibile gestire l'I/O da tastiera con molti dispositivi, per mantenere una certa generalitá in questo contesto ci si riferirá alla scheda XSA-50 con il generico termine di *host*.

Come si può capire dalla figura 1, tutta la comunicazione si basa su un bus formato da due linee bidirezionali: la linea di clock e la linea dati. Il bus è in uno stato di attesa quando le due linee di clock e di dati sono in uno stato logico alto: questo è l'unico stato nel quale è permesso alla tastiera trasmettere dati. L'*host* ha sempre il controllo finale sul bus e puó in ogni momento inibire la comunicazione mettendo a massa la linea di clock; alla tastiera invece (e solo a lei) spetta il compito di generare il clock.

I dati spediti dalla tastiera all'*host* sono letti sul fronte di discesa del clock; viceversa i dati trasmessi dall'*host* alla tastiera vengono letti sul fronte di salita. La frequenza del clock deve stare all'interno del range 10 - 16.7 kHz. Questo significa che il clock deve rimanere alto per 30 - 50 μs. Tali parametri sono assolutamente cruciali e agire su di essi è estremamente delicato: questo è il motivo per cui si è scelto di adattare il progetto della Queensland University agendo

su un divisore di clock a monte anziché ritarando le temporizzazioni (si veda a tal proposito la sezione 3).

2.2.1 XSA-50 → Tastiera

Se l'*host* vuole trasmettere dei dati deve innanzitutto inibire la comunicazione mettendo a massa il clock per almeno 100 μs, quindi mettere a massa anche la linea dati e rilasciare il clock: questa sequenza fa capire alla tastiera che sono in arrivo dei dati e quest'ultima inizia a generare il clock.

Tutti i dati vengono trasmessi in modo seriale un byte alla volta e ogni byte viene inserito in un frame di undici bit:

1 bit di start. Questo bit è sempre posto a 0.

8 bit di dati, ordinati dal meno significativo.

1 bit di paritá.

1 bit di stop. Questo bit è sempre posto a 1.

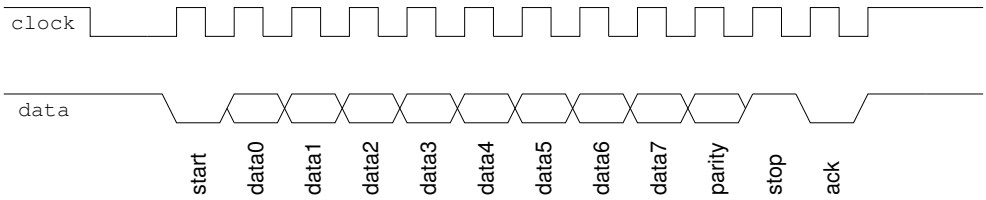


Figura 2: Frame e temporizzazioni

Alla fine di questo frame l'*host* si aspetta un acknowledge bit da parte della tastiera (nel caso il dato sia stato correttamente interpretato), quindi rilascia la linea dati. Se l'*host* non rilascia la linea dati dopo l'undicesimo ciclo di clock la tastiera capisce che c'è stato un errore nella comunicazione e continua a generare il clock finché la linea dati non viene rilasciata.

2.2.2 Tastiera → XSA-50

Quando la tastiera vuole spedire un dato, come prima cosa controlla la linea di clock per assicurarsi che sia in uno stato logico alto. Se non lo è significa che l'*host* sta inibendo la comunicazione e la tastiera deve *bufferizzare* i dati da spedire finché la linea di clock non viene rilasciata per almeno 50 microsecondi, quindi può spedire i suoi dati.

Per trasmettere i dati la tastiera usa lo stesso protocollo a undici bit viste nella sezione precedente.

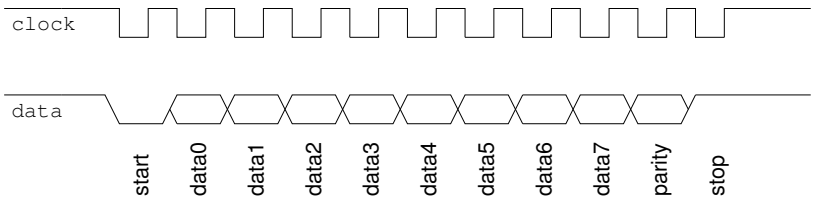


Figura 3: Frame e temporizzazioni

I dati che vengono trasmessi dalla tastiera corrispondono al codice dei tasti che vengono premuti, viceversa i dati spediti dalla scheda corrispondono ad alcuni comandi di controllo per la tastiera (si veda la sezione 2.3.3).

2.3 L'interfaccia

La tastiera consiste in una matrice di tasti ciascuno dei quali viene monitorizzato da un processore interno alla tastiera. Ogni tastiera possiede un processore specifico (diverso dagli altri in base alle caratteristiche della tastiera), ma tutti quanti i processori fanno sostanzialmente lo stesso lavoro: controllano quale tasto viene premuto/rilasciato e spediscono all'*host* il dato appropriato; all'occorrenza si occupano di *bufferizzare* i dati se la linea di comunicazione risulta occupata.

2.3.1 Scan codes

Come abbiamo giá visto, il processore interno alla tastiera spende la gran parte del suo tempo monitorizzando la matrice di tasti; quando vede che un tasto è stato premuto, rilasciato o tenuto premuto, si occupa di trasmettere all'*host* un pacchetto di informazioni chiamato "scan code".

Esistono due tipi di scan codes:

- **make codes**: vengono spediti quando un tasto è premuto o viene *tenuto premuto*;
- **break codes**: utilizzati quando un tasto è rilasciato.

Ogni tasto è associato ad un unica coppia < make code / break code >, in tal modo l'*host* puó determinare esattamente cos'è successo e a che tasto semplicemente guardando il singolo scan code. Il set di make e break code per ogni tasto forma quello che viene chiamato "**scan code set**". Esistono tre scan code set standard chiamati uno, due e tre: tra di loro si differenziano sostanzialmente nella codifica usata per identificare i tasti della tastiera e i comandi inviati dall'*host*; quale di questi standard usare dipende da particolari d'implementazione (ad esempio, come vedremo, lo scan code set 3 aggiunge particolari estensioni che utilizzeremo). É importante avere sempre presente che i codici inviati identificano univocamente un *tasto sulla tastiera* e non un carattere: ovvero non è definita alcuna relazione tra scan code e codifica ASCII (tradurre scan codes in codici ASCII è compito dell'*host*).

La tastiera inoltre supporta un'altra funzionalitá: il **typematic**; quando un tasto viene tenuto premuto, la tastiera invia un make code seguito da un treno di make code caratterizzati da un delay iniziale e da una velocitá di ripetizione entrambi programmabili.

2.3.2 Reset

Al momento dell'accensione la tastiera esegue un auto-test diagnostico denominato BAT (Basic Assurance Test) e carica i seguenti valori di default:

- Typematic - delay iniziale: 500 ms;
- Typematic - velocitá ripetizioni: 10.9 cps;
- Scan code set 2;
- Abilitati i make code, i break code e il typematic.

Al momento di iniziare il test la tastiera accende i suoi LED e li spegne una volta terminatolo. A questo punto la tastiera spedisce all'*host* un codice di successo (0xAA) o di errore (0xFC). Questo codice dev'essere spedito 500-750 millisecondi dopo l'accensione.

2.3.3 Command set

Si chiama "command set" l'insieme dei comandi di controllo che *host* e tastiera possono scambiarsi. In questa sede verranno presentati solo i comandi utilizzati nel presente lavoro (per una trattazione completa dell'argomanto si rimanda a [1]).

Sotto vengono riportati alcuni comandi che l'*host* puó mandare alla tastiera e la reazione della tastiera a tali comandi:

0xFF (Reset) → La tastiera risponde con un "ack" (0xFA), quindi entra in modalitá "reset".

0xFE (Resend) → La tastiera risponde spedendo nuovamente l'ultimo byte.

0xF0 (Set Scan Code Set) → La tastiera risponde con un "ack", quindi legge il dato successivo che l'host le invia, che puó essere:

0x01 → Scan Code Set 1;

0x02 → Scan Code Set 2;

0x03 → Scan Code Set 3.

La tastiera quindi risponde con un secondo "ack" e carica il relativo Scan Code Set.

0xF3 (Set Typematic Rate/Delay) → Serve per settare i due parametri del Typematic; la tastiera aspetta dall'*host* i dati successivi che definiscono delay e velocitá delle ripetizioni;

0xF6 (Set Default) → La tastiera carica i valori di default.

I prossimi comandi possono venir spediti alla tastiera in ogni momento ma influenzano il suo reale comportamento solo se la tastiera è in Scan Code Set 3:

0xF9 (Set All Keys Make) → La tastiera risponde con un "ack" e disabilita per tutti i tasti i break code e il typematic;

0xF8 (Set All Keys Make/Break) → Simile al precedente, con la differenza che disabilita solo il typematic;

0xF7 (Set All Keys Typematic) → Come il precedente, solo che disabilita il break code.

In particolare in questo progetto viene usato lo Scan Code Set 3, con l'opzione 0xF9 (vengono disabilitati il break code e il typematic). La codifica relativa allo Scan Code Set la si puó trovare in [2].

3 L'implementazione

Il presente progetto utilizza il controller PS2 scritto dall'universitá del Queensland per la scheda XSV800, semplificando e adattandone la logica alle (ristrette) risorse della XSA-50; la figura 4 mostra schematicamente la provenienza dei sorgenti.

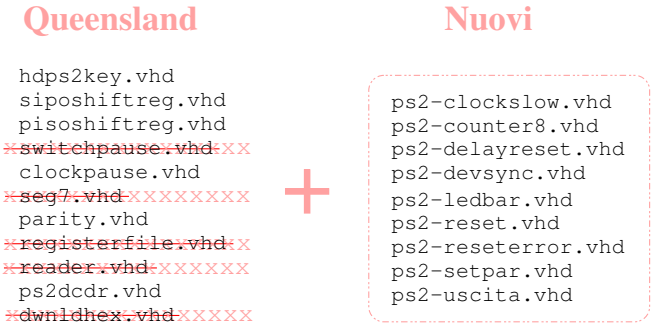


Figura 4: Provenienza dei sorgenti

Il progetto originale si trova all'indirizzo *www.xess.com* (sezione Examples).

Il codice originale prevede l'utilizzo di un clock di 1 MHz; questo progetto invece fa uso di un clock a 100MHz: si rende pertanto necessario un adattamento dei due progetti. Nella documentazione fornita dalla Queensland si legge che è possibile far lavorare il loro progetto con qualsiasi clock a patto di adattare alcune temporizzazioni (necessarie per il corretto dialogo tra

tastiera e scheda attraverso il portocollo ps2); si è tuttavia riscontrato che questo rappresenta un punto molto critico per l'intero sistema: pur facendo le opportune modifiche il progetto funziona solo per un breve periodo di tempo, dopo di che si blocca inesorabilmente. La soluzione piú veloce è quella di continuare ad usare un clock di 100 MHz per il nostro lavoro, con l'accortezza di aggiungere un divisore di clock "100 to 1" all'ingresso del sistema che gestisce la tastiera.

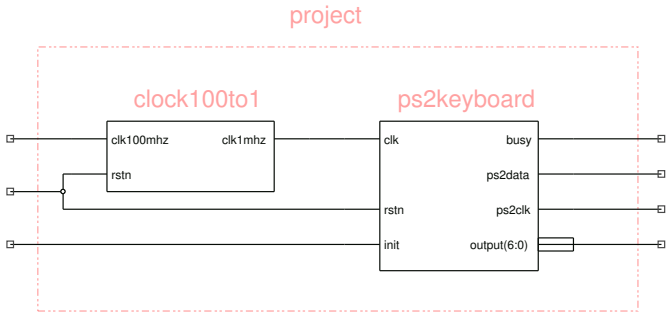


Figura 5: Il divisore di clock a monte

In appendice A è presente lo schema del progetto della parte a valle del divisore di clock (il blocco che in figura 5 è chiamato `ps2keyboard`); i segnali con cui questo modulo per la gestione della tastiera si interfaccia con l'esterno sono:

clk *in*: master clock del progetto (a 100MHz);

rstn *in*: il reset globale (attivo basso);

init *in*: pulsante per la inizializzazione della tastiera;

busy *out*: indica quando il modulo è occupato;

ps2data *out*: la linea dati (bidirezionale);

ps2clk *out*: la linea di clock (bidirezionale);

output *out*: l'uscita che pilota i led a sette segmenti.

A questo punto analizziamo ogni singolo modulo per studiarne la funzione e l'interfacciamento con l'esterno.

3.1 `ps2dcdr`

È il cuore dell'intero sistema; questo modulo implementa una macchina a stati che si occupa di gestire la comunicazione attraverso il bus: riceve e spedisce i dati, controlla la paritá, bufferizza i dati in uscita nel caso il bus sia occupato, il tutto secondo le regole del protocollo PS2 visto nella sezione 2.
I segnali che vengono gestiti da questo modulo sono:

clk *in*: master clock del modulo;

rstn *in*: reset asincrono (attivo basso) `OR` error[1];

ps2data *inout*: linea dati bidirezionale;

ps2clk *inout*: linea clock, anch'essa bidirezionale;

[1] A tal proposito si veda la sezione 3.9

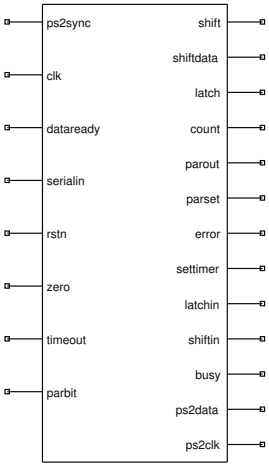


Figura 6 Modulo ps2dcdr.

parBit *in*: indica il bit di paritá attuale;

count *out*: incrementa il contatore `counter8`;

error *out*: asserito in caso di errore;

ps2sync *in*: indica la presenza di un fronte di discesa sulla linea precedente;

dataReady *in*: viene asserita quando un byte e' pronto per essere spedito;

serialIn *in*: bit da trasmettere provveniente dal registro Parallelo/Seriale;

zero *in*: asserito quando il contatore `counter8` registra il valore 0;

timeout *in*: asserito quando il contatore `clockpause` ha misurato un tempo di 64μs;

shift *out*: se asserito provoca uno shift dei dati all'interno del registro Seriale/Parallelo;

shiftData *out*: dati seriali in uscita da passare al registro Seriale/Parallelo;

latch *out*: indica quando il registro Seriale/Parallelo ha dei dati validi (possono essere presi);

parOut *out*: bit di dato inviato al modulo `parity` per il calcolo della paritá;

parSet *out*: asserito quando si vuole inizializzare (a 1) il modulo `parity`;

setTimer *out*: asserito quando si vuole far partire il contatore `clockpause` (64μs);

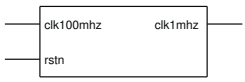
latchIn *out*: asserito provoca il caricamento di un byte len modulo Parallelo/Seriale;

shiftIn *out*: se asserito provoca l'uscita di un bit dal registro Parallelo/Seriale;

busy *out*: asserito quando il bus è occupato (sta inviando o ricevendo dati).

3.2 `ps2-clock100to1`

Questo semplice modulo implementa il divisore di clock di cui si è parlato all'inizio: in uscita propone un clock cento volte piú lento del clock ricevuto in ingresso; serve per adattare i moduli presi dal progetto dell'universitá del Queensland a questo lavoro.



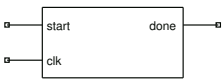
clk100MHz *in*: il master clock della scheda a 100MHz;

rstn *in*: il master reset (attivo basso);

clk1MHz *buffer*: il clock a 1MHz che andrá a pilotare `ps2keyboard`.

3.3 `ps2-clockpause`

Questo modulo conta un ritardo di 64 cicli di clock (clock che va a una frequenza di 1MHz, quindi 1μs a ciclo, in tutti 64μs).



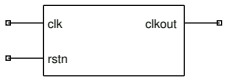
start *in*: asserito fa partire il calcolo del delay;

clk *in*: il clock a 1MHz;

done *out*: asserito quando il modulo ha contato 64 cicli di clock.

3.4 `ps2-clockslow`

Questo modulo implementa un ulteriore divisore di clock. In uscita genera un clock molto lento (circa un secondo per ciclo) che serve a pilotare il lampeggiare del led al momento dell'inserimento di un valore per un parametro (si veda la sezione 3.13).



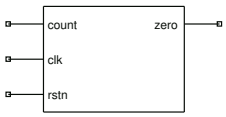
clk *in*: il clock a 1MHz;

rstn *in*: il master reset (attivo basso);

clkout *buffer*: clock che pilota i led del modulo `setparameters`.

3.5 `ps2-counter8`

Contatore che - come suggerisce il nome stesso - serve a contare otto unitá. Viene usato per raggruppare bit in byte; l'uscita è presa in ingresso dal modulo `ps2dcdr`.



count *in*: se asserito conta i cicli di clock;

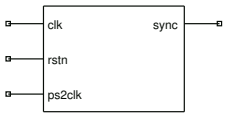
clk *in*: il clock a 1MHz;

rstn *in*: il master reset (attivo basso) `OR` error;

zero *out*: asserito quando il contatore assume il valore zero.

3.6 `ps2-devsync`

Questo modulo serve a rilevare i fronti di discesa del clock proveniente dalla tastiera.



clk *in*: il clock a 1MHz;

rstn *in*: il master reset (attivo basso) `OR` error;

ps2clk *inout*: linea clock, anch'essa bidirezionale;

sync *out*: in presenza di un fronte di discesa del clock questa linea viene asserita per un intero ciclo di clock.

3.7 ps2-reset

Questo è il modulo che si occupa di inizializzare la tastiera. In figura 7 possiamo vedere una semplificazione della macchina a stati che implementa l'inizializzazione.

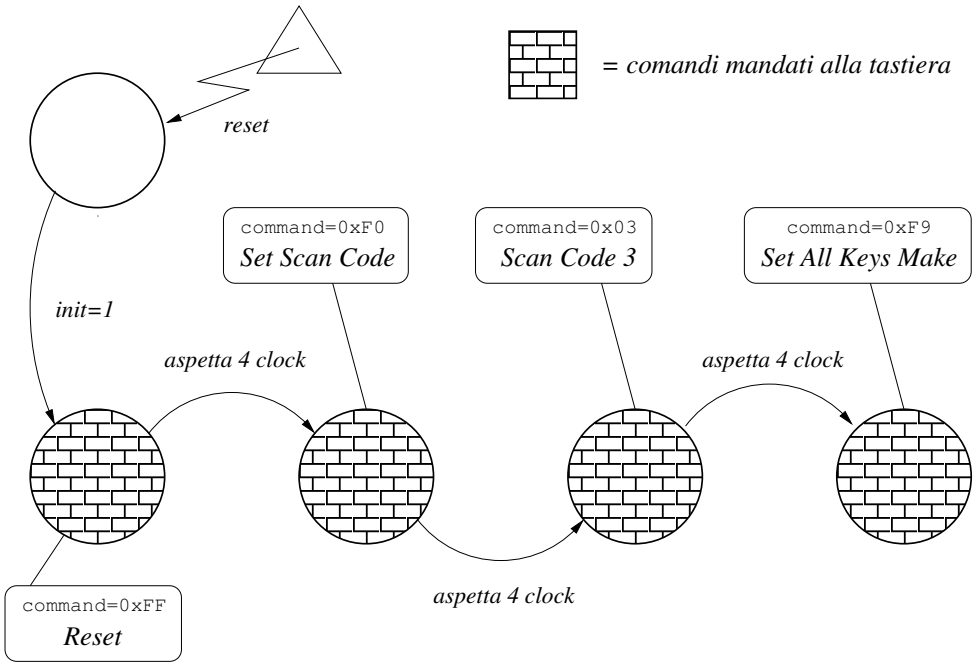


Figura 7: Reset - macchina a stati

Quando (tramite un pulsante esterno) viene dato il comando di settare la tastiera (*init=1*) l'FPGA (ovvero l'*host*) spedisce alla tastiera la sequenza di comandi:

0xFF → resetta tutto (imposta i valori di default);

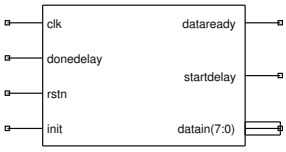
0xF0 → cambia Scan Code Set...

0x03 → e carica lo Scan Code Set 3;

0xF9 → disabilita i break code e il typematic.

Per essere sicuri che la tastiera abbia ricevuto i comandi e che sia pronta a riceverne altri, tra un comando e l'altro si aspetta un tempo minimo di 4 colpi di clock.

Vediamo ora in dettaglio ingressi e uscite del modulo:



clk *in*: il master clock della scheda a 100MHz;

donedelay *in*: alto quando sono passati quattro cicli di clock;

rstn *in*: il master reset (attivo basso);

init *in*: segnale che avvia l'inizializzazione;

dataIn *out*: bus in cui vengono messi i comandi da inviare alla tastiera;

dataready *out*: avvisa il controller che puó prelevare dal convertitore Parallelo/Seriale il comando da inviare alla tastiera;

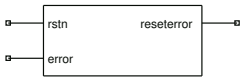
startdelay *out*: fa partire il conto dei quattro cicli di clock.

3.8 `ps2-delayreset`

Questa è l'entity che fornisce al modulo ps2reset il delay di 4 cicli di clock. La figura dell'entity è autoesplicativa per quanto riguarda ingressi e uscite.

3.9 `ps2-reseterror`

Abbiamo giá visto come il rispetto di alcune temporizzazioni sia un punto nevralgico dell'intero sistema; a volte puó capitare (anche nel progetto della Queensland) che dopo qualche minuto di funzionamento *host* e tastiera si trovino non sincronizzati e pertanto non capaci di continuare la comunicazione. Questa situazione di stallo viene evitata resettando automaticamente controllore e registri SIPO-PISO (e logica associata) al primo "ack" che il controllore `ps2dcdr` non riceve dalla tastiera (si veda la sezione 2.2.1). Si ponga particolare attenzione al fatto che si tratta di un reset che viene dato alla logica del FPGA e *non* alla tastiera: l'unico effetto di questo reset è una risincronizzazione tra i due dispositivi (la tastiera conserva le sue impostazioni e l'uscita principale del modulo non si accorge di nulla).



rstn *in*: il master reset (attivo basso);

error *in*: il segale di errore che l'entity `ps2dcdr` genera al primo "ack" non ricevuto da tastiera;

reseterror *out*: reset asincrono (attivo basso) `OR` error.

3.10 `pisoshiftreg` e `shiftregsipo`

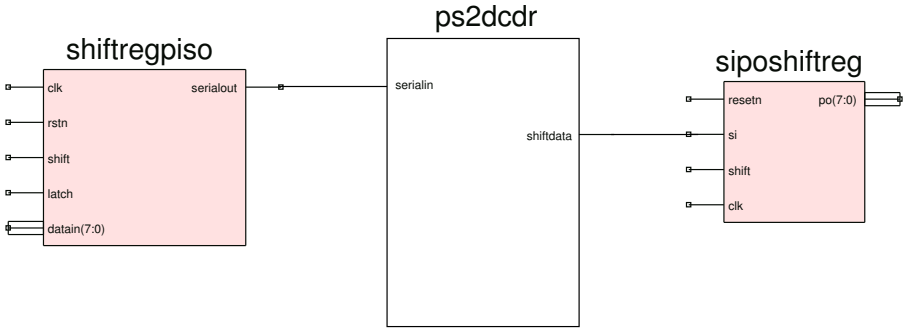
Questi due moduli sono dedicati a trasformare i comandi dell'*host* da parallelo (un comando = un byte) a seriale (pronti per essere spediti sulla linea `ps2data`) e da seriale (cosí come li spedisce la tastiera) a parallelo (pronti per essere elaborati dal modulo `ps2-setpar`). Vediamo in dettaglio l'I/O del modulo:

clk *in*: il clock a 1MHz;

rstn *in*: reset asincrono (attivo basso) `OR` error;

si *in*: linea dalla quale entrano nel registro i bit in modo seriale, sincronizzati con `ps2clok`;

shift *in*: quando questa linea è alta i registri lavorano, quando è bassa i registri stanno fermi;



po *out*: bus in cui esce il byte contenente il make code del tasto premuto;

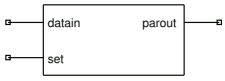
datain *in*: bus in cui entra il byte contenente il comando che l'*host* deve spedire alla tastiera;

latch *in*: segnale impulsivo che dice quando bufferizzare il dato all'interno dello shift-register parallelo/seriale;

serialout *out*: linea dalla quale escono dal registro i bit in modo seriale, sincronizzati con `ps2clok`;

3.11 `ps2-parity`

Questo è il blocco (asincrono) che si occupa del calcolo della paritá, sia in uscita (calcola la paritá dei bit inviati alla tastiera) che in ingresso (controlla che il bit di paritá inviatogli dalla tastiera corrisponda alla paritá dei bit ricevuti). Il metodo di paritá adottato dal protocollo ps2 è quello dispari, ovvero il bit di paritá è settato se nel campo dati sono presenti un numero pari di "1", mentre è messo a zero se ne sono presenti in numero dispari.



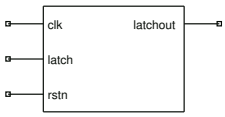
datain *in*: ingresso in cui vengono inseriti i bit per il calcolo della paritá;

set *in*: se asserito inizializza il conto della paritá (inizializzato a 1);

parout *out*: il bit di paritá.

3.12 `ps2-latchdelay`

Compito di questa entity è quello di avvisare il modulo `ps2-setpar` quando un dato spedito dalla tastiera è stato parallelizzato ed è pronto per essere portato in uscita. In realtá questo blocco altro non è che un delay sulla linea `latch` proveniente dal controller, ovvero introduce un ritardo per permettere al convertitore seriale/parallelo di preparare un'uscita stabile. In dettaglio **latch** (*in*) è il segnale proveniente dal controller, **latchout** (*out*) è il segnale ritardato di qualche ciclo di clock.



3.13 ps2-setpar

Questo modulo pilota il display a sette segmenti presente sulla XSA-50 e porta in uscita una coppia <parametro,numero> utilizzabile da future applicazioni.

Il cuore dell'entity è una macchina a stati (in figura 8 si puó vedere una sua semplificazione): al momento del reset il modulo è pronto a ricevere segnali e inizia a verificare se il controller (`ps2dcdr`) ha ricevuto dei dati dalla tastiera e se questi sono pronti (`latchdelayed`[2]`=1`); in questo stato la macchina accetta solo il codice dei tasti relativi ai parametri da settare (in questo caso "n" e "t"). Una volta che è stato selezionato il tasto relativo al parametro il sistema inizia a "rimbalzare" tra due stati con la frequenza data da `clockslow`: uno stato pilota l'accensione dei led corrispondenti alla lattera "n" o "t", l'altro forza i led a rimanere spenti; questo serve a far lampeggiare la lettera selezionata indicando che il modulo è pronto a ricevere il valore da associare al parametro.

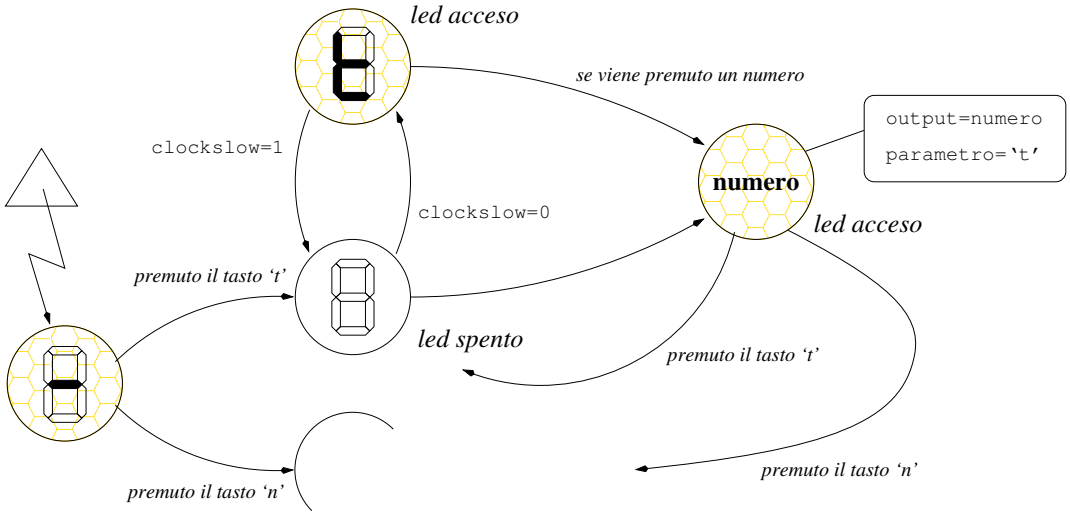


Figura 8: Setpar - metá macchina a stati

Se a questo punto viene inserita una cifra (dal tastierino numerico[3]) sul led compare il numero indicato e l'uscita si assesta sui valori parametro=*parametro_selezionato*, numero=*cifra*, e il modulo riprende ad "ascoltare" la linea `latchdelay`.

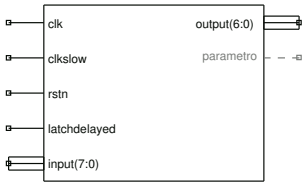


Figura 9 Modulo setpar.

Scendendo in dettaglio:

output *out*: questa uscita pilota il led a sette segmenti; tuttavia anziché associare a questo bus i sette bit relativi all'accensione degli opportuni led, è possibile assegnare (usandone otto) la codifica ASCII corrispondente al numero selezionato (o alla lettera selezionata...);

parout *out*: questa uscita in realtá non è stata implementata (in quanto in questo progetto dovrebbe rimanere una linea - anzi... un bus - scollegato, non essendo previsto un suo utilizzo). Tuttavia abilitarlo richiederebbe non piú di cinque righe, e il procedimento è lo stesso dell'uscita `output`.

[2]Sul perché sia un latch ritardato si guardi la sezione 3.12

[3]Ricordiamo che all'*host* arrivano make code che identificano una posizione sulla matrice di tasti, non un codice ASCII: di conseguenza premendo "5" dal tastierino numerico o dalla prima fila di tasti sulla tastiera si otterranno due codici diversi!

4 Funzionamento

Il funzionamento di questo progetto è immediato: dopo aver fatto il download del binario all'interno dell'FPGA è sufficiente resettare la logica (premendo il tasto S5[4]) e inviare alla tastiera il segnale di inizializzazione (prmendo il tasto S4 - si notará il lampeggiare dei led presenti sulla tastiera); a questo punto la logica interna all'FPGA aspetta la selezione dei parametri e l'assegnazione del loro valore come indicato in sezione 3.13.

Bibliografia

[**1**] The AT-PS/2 Keyboard Interface:
http://panda.cs.ndsu.nodak.edu/%7Eachapwes/PICmicro/keyboard/atkeyboard.html

[**2**] Tabella dello Scan Code Set 3:
http://panda.cs.ndsu.nodak.edu/%7Eachapwes/PICmicro/keyboard/scancodes3.html

[**3**] PS/2 Mouse/Keyboard Protocol:
http://panda.cs.ndsu.nodak.edu/%7Eachapwes/PICmicro/keyboard/atkeyboard.html

[**5**] Progetto "PS2 Interface for the XSV Board" dell Università del Queensland
http://www.xess.com/ho03000.html#Examples

[**4**] VHDL Reference Manual, Synario:
http://cslab.snu.ac.kr/course/cad99/vhdl_ref.pdf

[**6**] XSA-50 Board V1.2 Manual
http://www.xess.com/manuals/xsa-manual-v1_2.pdf

[4] S5 e S4 sono due pulsanti presenti nella scheda d'espansione XSA XStend v2: questo progetto infatti è stato pensato per fornire un'interfaccia per tastiera a logiche presenti sulle due schede collegate; questo tuttavia non presenta problemi nel caso si possegga solo una XSA-50, in quanto - con sforzo minimo - è possibile far partire l'inizializzazione subito dopo il reset asincrono, modifica che richiederebbe la presenza di un unico pulsante (come nel caso della XSA-50).

Appendice A

Schema generale del progetto
(moduli a valle del divisore di clock)
I blocchi colorati sono quelli presi
dal progetto dell'università di Queensland