

Implementazione del gioco “Pong” su FPGA Virtex

di Gabriele Del Prete

Abstract

Viene presentato il progetto e l'implementazione del videogioco “Pong” (anche noto come “Tennis”) sulla scheda di sviluppo XSV800 di XESS Corp., realizzata attorno ad una FPGA Virtex XCV800-4-HQ240-C. Il gioco viene comandato attraverso una tastiera PS2 e l'output viene mostrato su uno schermo VGA compatibile. L'implementazione del sistema è stata fatta facendo uso del linguaggio VHDL. Vengono inoltre descritti i passi necessari per la compilazione dei file VHDL, della programmazione del generatore di clock sulla scheda e della memorizzazione del bitstream contenente il gioco sulla FlashRam della scheda affinché il gioco parta appena viene fornita l'alimentazione.

Introduzione al gioco Pong e considerazioni sugli aspetti implementativi

Il gioco Pong viene giocato da due giocatori, che controllano ognuno una racchetta (di seguito chiamate “pad”) con la quale colpire una pallina che si muove sul campo delimitato in alto e in basso da un confine sul quale essa rimbalza. Perde la partita il giocatore che non riesce a respingere con la propria racchetta la pallina.

Il progetto sviluppato consente di giocare utilizzando una tastiera PS2 standard collegata all'apposito ingresso sulla scheda XSV800 per muovere le pad dei due giocatori e un monitor VGA collegato all'omonima uscita della scheda per visualizzare il campo di gioco, le pad e la pallina che si muove. I due display a sette segmenti presenti sulla scheda mantengono il conto dei punti dei due giocatori. Infine, è possibile porre il gioco in pausa in ogni momento mediante la pressione di un tasto sulla tastiera. Il tasto della pausa serve anche a iniziare una partita.

I tasti della tastiera che vengono utilizzati sono:

- A: per spostare la pad alla sinistra dello schermo in su
- Z: per spostare la pad alla sinistra dello schermo in giù
- K: per spostare la pad alla destra dello schermo in su
- M: per spostare la pad alla destra dello schermo in giù
- P: per porre il gioco in pausa, ripartire e iniziare una partita.

Il gioco può essere giocato con due livelli di difficoltà, che corrispondono alla velocità con cui la pallina si muove lungo l'asse orizzontale del campo. La difficoltà può essere variata ad ogni istante utilizzando il dipswitch 1 sulla scheda. Se esso è sulla posizione ON, la pallina si muove a bassa velocità, altrimenti la velocità è alta. La velocità della pallina influenza anche la velocità di movimento delle pad (che raddoppia quando la pallina si muove a velocità alta).

L'angolo con cui la pallina rimbalza sulle pad è determinato casualmente ad ogni collisione, e può essere di 0, 22.5, 45 e 67.5 gradi (angolo di incidenza rispetto alla perpendicolare alla pad). Invece quando la pallina rimbalza su uno dei bordi superiore o inferiore viene solo invertito il verso del moto verticale (l'angolo di incidenza e l'angolo di riflessione sono sempre uguali).

Sono casuali anche l'angolo con cui la pallina parte dal centro schermo all'inizio di ogni partita e la direzione iniziale del moto (verso l'alto o verso il basso; verso destra o verso sinistra).

Alla fine di ogni partita (quando uno dei due giocatori segna un punto) la pallina viene riportata al centro e il gioco viene posto in pausa; per iniziare la nuova partita è sufficiente togliere la pausa premendo il tasto P sulla tastiera.

È anche possibile resettare il gioco in un qualunque momento agendo sul pulsante SW1 della scheda XSV800; in questo caso anche i punteggi vengono resettati. Dopo il reset, per iniziare la partita è necessario premere il tasto P.

Breve descrizione della scheda XSV800 di XESS Corp.

La scheda XSV800 di XESS Corp. è una scheda di sviluppo basata sull'FPGA Virtex XCV800-4-HQ240-C (per il datasheet, vedi [2]) di Xilinx. La scheda dispone (vedi [1]), oltre all'FPGA, di due banchi di memoria da 1Mbyte ciascuno, una porta PS2 per il collegamento di una tastiera compatibile, una porta USB, un ingresso ed un'uscita audio, una porta RJ45 associata ad un PHY Ethernet, una porta parallela IEEE1284, una porta seriale RS232, un'entrata video PAL/NTSC (con connettore sia composito che S-Video), e una uscita VGA i cui segnali possono essere pilotati direttamente dalla FPGA o attraverso l'ausilio di un RAMDAC (per la descrizione del RAMDAC, vedi [3]). Dispone inoltre di due display a sette segmenti, una barra di 10 led, una serie di 8 dipswitch e 4 pulsanti (vedi [1], pag. 30-33). Oltre alla FPGA, sulla scheda è presente anche un altro dispositivo programmabile: una CPLD XC95108 (vedi [12] per il datasheet) sempre di Xilinx. La CPLD è utilizzata principalmente per consentire la programmazione della FPGA attraverso la porta parallela, quella seriale, il connettore Xchecker di Xilinx, o la Flash Ram, in modo che la scheda possa essere utilizzata senza un computer host che la programmi ad ogni accensione. È presente altresì un generatore programmabile di clock (datasheet in [4]) da 100MHz a 48.7KHz che pilota la FPGA. Il gioco Pong utilizza un clock da 50MHz.

La Gestione della Tastiera

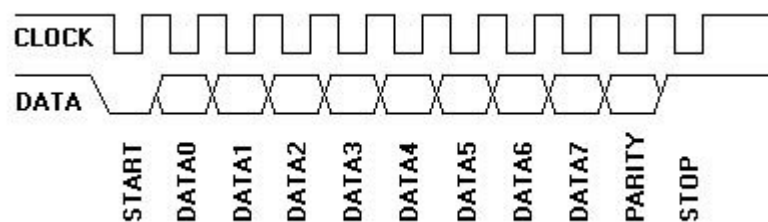
Descrizione del protocollo PS2.

L'input del gioco avviene principalmente attraverso una tastiera PS2 collegata all'ingresso omonimo presente sulla scheda XSV800. Il bus PS2, composto da due segnali, è collegato direttamente a due pin della FPGA (vedi pag. 33 in [1]).

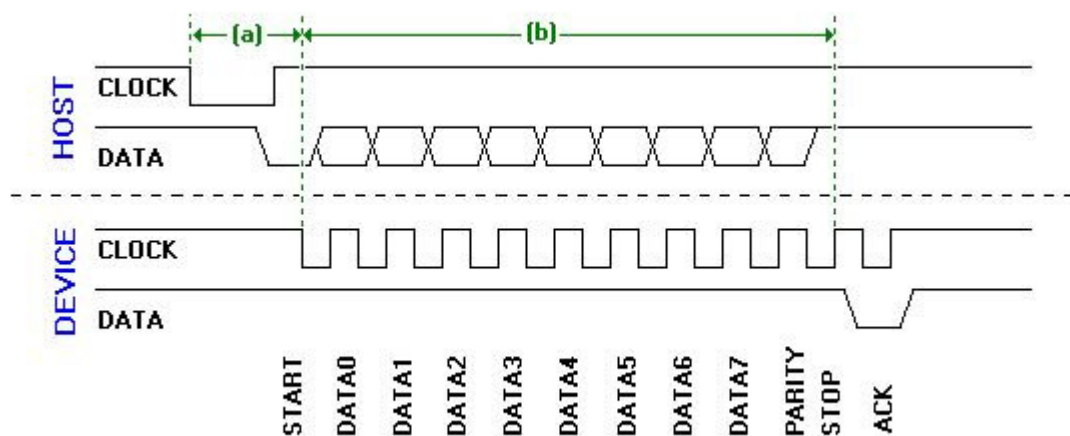
Per una migliore comprensione del funzionamento delle entity VHDL relative alla comunicazione con la tastiera, è utile soffermarsi sul bus PS2 e dare qualche cenno sul protocollo che FPGA e tastiera seguono per comunicare.

Il bus PS2 è un bus seriale sincrono bidirezionale (vedi [7]) che utilizza due segnali, un segnale di clock e un segnale su cui vengono trasmessi i bit di dati. Sia l'host (il computer generalmente o l'FPGA nel nostro caso) sia il dispositivo (la tastiera) possono iniziare una comunicazione, e l'arbitraggio viene effettuato utilizzando il segnale di clock.

Nello specifico, come descritto in [7], quando è la tastiera a voler comunicare, inizia la trasmissione solo se il segnale di clock è a livello alto (stato di bus libero; vedi figura sottostante). In questo caso inizia la generazione del clock e i bit vengono posti sul segnale data; il protocollo specifica che i bit devono essere letti sul fronte di discesa del clock.



Quando invece è l'host a voler comunicare con la tastiera (vedi sempre [7]), questo deve segnalare alla tastiera la sua intenzione a trasmettere, ponendo la linea clock bassa per almeno 100 microsecondi (ciò inibisce la trasmissione di bit da parte della tastiera) e quindi abbassando la linea data prima di rilasciare la linea clock (vedi figura sottostante, nella parte "HOST", nell'intervallo di tempo detto "a"). A questo punto (diagramma "DEVICE", intervallo di tempo "b") la tastiera capisce che l'host vuole inviare dei dati e inizia a generare il clock (è sempre la tastiera a generare il clock). La tastiera campiona la linea data sul fronte di salita del clock (da notare che nel caso precedente il dato veniva catturato sul fronte di discesa).



La comunicazione seriale è composta da flussi di bit lunghi 11 bit così composti:

- 1 bit di start, sempre a 0
- 8 bit di dati, con il bit meno significativo inviato per primo

- 1 bit di parità
- 1 bit di stop, sempre a 1

Il bit di parità è calcolato in modo che il numero di bit a 1 nella concatenazione del campo dati con il bit di parità stesso sia dispari (metodo della parità dispari).

In più, nel caso dell'invio di bit da host a tastiera, dopo l'invio del bit di stop da parte dell'host, la tastiera invia in corrispondenza del successivo fronte di salita del clock un bit a zero (detto bit di acknowledge) per confermare l'avvenuta ricezione dei bit.

I byte che vengono inviati dalla tastiera all'host sono i codici dei tasti premuti. I byte inviati dall'host alla tastiera sono invece comandi che la tastiera può eseguire (accendi/spegni led sui tasti, abilita la ripetizione dei tasti, ecc.).

Codici inviati dalla tastiera e comandi accettati dalle tastiere PS2

I codici inviati dalla tastiera vengono detti “scan codes” e indicano quale tasto l'utente ha premuto o ha rilasciato (per una migliore comprensione di questa sezione, vedere [8]). Esistono 3 possibili insiemi di scan codes, detti “scancode set”, che si differenziano principalmente per i codici inviati dalla tastiera all'host e dai comandi accettati dalla tastiera. Il primo scancode set, detto “scancode set 1” è l'originale inventato da IBM per la tastiera dell'IBM XT e non viene utilizzato più, il secondo (“scancode set 2”) è quello creato per l'IBM AT, viene utilizzato attualmente ed è un'estensione del primo dovuto all'aggiunta, sulla tastiera dell'AT, di alcuni tasti che non esistevano su quella dell'XT, e il terzo (“scancode set 3”) è un codice creato ex-novo da IBM per i computer della serie PS2 che non è stato però mai utilizzato da nessuno. Tutte le tastiere supportano lo scancode set 2, quasi tutte lo scancode set 3 ma quasi nessuna lo scancode set 3.

Quando si preme un tasto, viene generato un codice che è detto “make code”, mentre quando un tasto viene rilasciato viene generato un “break code”. Sia nel caso dello scancode set 2 che nel caso dello scancode set 3 il make code è composto dal byte con il codice del tasto, mentre il break code è lo stesso byte preceduto però da un altro byte di valore 0xF0. Inoltre, quando la tastiera è in scancode set 3 i tasti di default non generano il break code (generano solo il make code e i codici di ripetizione).

La tastiera supporta inoltre una funzionalità di ripetizione tasti (funzionalità “typematic”) con un delay pre-ripetizioni e una velocità di ripetizione entrambi programmabili. I codici di ripetizione (typematic codes) sono uguali al make code.

Gli scan codes corrispondenti ai tasti utilizzati dal gioco Pong (A,Z,K,M,P) sono: A: 0x1C Z: 0x1A K: 0x42 M: 0x32 P: 0x4D.

Oltre a trasmettere gli scan codes, la tastiera può ricevere dei comandi dall'host (vedi sempre [8]).

Quelli utilizzati per il Pong sono:

- 0xFF – comanda alla tastiera di resettarsi e di effettuare il self-test (che è detto BAT)
- 0xF0 – per impostare lo scancode set
- 0xF3 – per impostare il delay pre-ripetizioni e la velocità di ripetizione dei tasti

Ad ogni byte inviatole, la tastiera risponde con un byte 0xFA (byte di acknowledge) o 0xFE (byte di errore, quando il comando inviato non è valido).

Il byte 0xFF serve a resettare la tastiera: svuota i buffer ed effettua il test diagnostico (è la fase in cui lampeggiano i led della tastiera). La tastiera invia, oltre al byte di acknowledge, il byte 0xAA per confermare l'esito positivo del test, altrimenti 0xFC (errore nel reset).

Il comando 0xF0 imposta invece lo scancode set che la tastiera deve utilizzare; il parametro che indica quale scancode set selezionare (0x01, 0x02, 0x03) viene inviato come byte successivo.

Il comando 0xF3 imposta invece contemporaneamente la velocità con cui la tastiera genera i byte di ripetizione e l'intervallo di tempo da aspettare prima di iniziare la generazione. Il parametro viene inviato come byte successivo al comando (per l'elenco dei valori possibili vedere la tabella in [8]; da notare però che con la tastiera utilizzata i valori della tabella non risultano veritieri).

Occorre notare che non tutte le tastiere supportano tutte le funzionalità, ad esempio alcune tastiere non supportano lo scancode set 3, altri supportano la ripetizione automatica dei tasti solo quando utilizzate con lo scancode set 3 (come quella utilizzata), ecc...

Requisiti per la tastiera per il gioco Pong

Il gioco Pong sfrutta la tastiera principalmente per comandare il movimento delle pad. Per far compiere un movimento fluido alle pad, è possibile scegliere fra due implementazioni:

1. Rilevare il make code di uno dei tasti di interesse, quindi con un clock generare un treno di impulsi fino a quando non si rileva l'arrivo di un break code. La frequenza del treno di impulsi determina la velocità di movimento delle pad. A tale scopo, la tastiera deve essere programmata per non generare lei stessa i codici di ripetizione, ma unicamente i codici di make e i codici di break. L'FPGA deve allora aspettare l'arrivo di uno dei codici di make (es. 0x1C per 'A') e generare il treno di impulsi fino all'arrivo del break code corrispondente (nell'esempio, 0xF0 0x1C) oppure l'arrivo del break code di un altro tasto (perché quando si tiene premuto un tasto e se ne preme un altro, il precedente viene "annullato" senza che venga inviato il suo break code; questo comportamento della tastiera non è modificabile e ha come risultato che un giocatore può inibire l'altro che si muove semplicemente premendo un altro tasto sulla tastiera). Si tratterebbe insomma di implementare una macchina a stati.
2. Attivare la ripetizione sulla tastiera con gli appositi comandi, impostare la frequenza di ripetizione dei tasti, e disabilitare i break code. In questo modo ad ogni make code di interesse rilevato si fa corrispondere l'impulso che muove la pad. Per implementare questa seconda soluzione basta impostare la tastiera allo scancode set 3, che disabilita il break code e attiva make code e ripetizione.

È stata scelta la seconda strada perché di più facile utilizzo e più economica: poiché si era deciso comunque di resettare la tastiera ad ogni reset del gioco, il costo dell'aggiunta dell'invio dei quattro byte 0xF0 0x03 (per impostare lo scancode set 3) e 0xF3 0x00 (per impostare il minimo delay pre-ripetizioni e una adeguata frequenza di ripetizione dei tasti) era basso.

Descrizione dei file VHD di gestione della tastiera

La gestione del bus PS2 viene fatta utilizzando delle entity VHDL realizzate dall'Università del Queensland (di seguito abbreviata in UDQ; vedi [6]), opportunamente modificate per togliere la visualizzazione dei byte ricevuti sui due display a sette segmenti. Sono stati quindi aggiunte delle entity per inviare i comandi di configurazione alla tastiera dopo il reset e per riconoscere i tasti di interesse e generare dei segnali a impulsi quando ne viene ricevuto uno. Inoltre, poiché le entity sviluppate dalla UDQ sono pensate per funzionare a 1Mhz, esiste anche un divisore di clock che porta il clock di sistema di 50Mhz a 1Mhz.

Diamo qui di seguito una breve descrizione delle entity coinvolte nella gestione della tastiera. Per una descrizione dettagliata delle stesse, riferirsi ai commenti presenti nei file stessi. Tutti i circuiti sono sincroni, se non diversamente segnalato, con clk come segnale di clock e rstn come segnale di reset (attivo basso).

ps2dcdr – Entity di gestione del bus PS2, si occupa di inviare e ricevere i byte sul bus, controllando la parità, nel rispetto delle regole di arbitraggio del bus (come descritto in [7]). I segnali con cui questa entity comunica con l'esterno sono:

ps2data: linea dati bus PS2 (input/output)

ps2clk: linea clock bus PS2 (input/output)

ps2sync: da asserire quando si ha un fronte di discesa sulla linea di clock del bus PS2 (input)

setTimer: asserito per far partire il timer di 64 microsecondi (output)

timeout: da asserire quando il timer di 64 microsecondi ha concluso il conteggio (input)

count: incrementa il contatore da 0 a 7 (output)
zero: da asserire quando il contatore raggiunge lo zero (input)
parBit: bit attuale di parità da spedire o con cui controllare quello ricevuto (input)
parout: bit di dato della sequenza di bit di cui calcolare la parità (output)
parSet: imposta il generatore/verificatore di parità (output)
shift: asserito quando su shiftData c'è un nuovo bit di un byte ricevuto dalla tastiera (output)
shiftData: ultimo bit ricevuto (output)
latch: indica che su shiftData c'è l'ultimo bit di una sequenza di 8 (byte completato) (output)
error: asserito quando il byte ricevuto non verifica il controllo di parità (output)
latchIn: asserito per comandare il caricamento di un byte nel registro del PiSo¹ di input (output)
serialIn: bit attuale del byte da spedire (input)
shiftIn: asserito per estrarre un bit dal PiSo di input (output)
dataReady: da asserire per comandare l'invio di un byte (input)
busy: asserito quando l'entity è occupata a ricevere o inviare un byte (output)

devsync – Entity che rileva i fronti di discesa del clock del bus seriale PS2. ps2clk è la linea di clock del bus PS2, sync è asserito quando viene osservato un fronte di discesa della linea. Utilizzato dall'entity ps2dcdcr per la gestione del protocollo PS2 (vedi anche [7]).

parity – Entity asincrona che calcola il bit di parità (secondo la regola di parità dispari) di una sequenza di bit. Utilizzato sia durante l'invio per calcolare il bit di parità da inviare sia durante la ricezione per verificare il bit di parità spedito dalla tastiera. DataIn è il bit in ingresso, set è il segnale che inizializza l'entity, parout è il valore che deve avere il bit di parità (ricevuto o da spedire) affinché risulti dispari il numero di bit a 1 nella concatenazione fra il byte e il bit di parità. L'inizializzazione pone lo stato interno a 1, e ogni bit in ingresso a 1 inverte lo stato; l'uscita equivale al valore corrente dello stato.

clockpause – Entity che crea un delay di 64 cicli di clock (a 1Mhz, 64 microsecondi). Utilizzato da ps2dcdcr.vhd per gestire le temporizzazioni di arbitraggio del bus (come spiegato sopra, lo standard prevede 100 microsecondi (vedi [7]), ma evidentemente le tastiere richiedono molto meno). start è il segnale da asserire per far partire il timer, done è il segnale asserito quando il timer ha completato il conteggio.

siposhiftreg – usato per convertire da seriale a parallelo una sequenza di bit. È utilizzato per convertire il byte ricevuto e porlo su un bus parallelo per consentirne un più facile utilizzo. I bit vengono accodati a partire dal meno significativo. SI è il segnale con il bit di ingresso, PO è il bus in uscita a 8 segnali su cui c'è il byte, shift è il segnale da asserire per caricare il bit su SI all'interno del registro parallelo.

counter8 – contatore da 0 a 7. Utilizzato ad esempio dall'entity ps2dcdcr per contare i bit ricevuti e quelli da inviare. count è il segnale da asserire per incrementare il contatore, zero è il segnale asserito quando il contatore è a zero.

pisoshifreg – usato per convertire un byte posto in ingresso sul bus parallelo e memorizzato in un registro interno, in una sequenza di bit in uscita. È utilizzato per convertire il byte da spedire alla tastiera in una sequenza di bit da iniettare sul segnale di entrata serialIn di ps2dcdcr, partendo dal bit meno significativo del registro interno. dataIn è il bus parallelo in entrata, serialOut la linea seriale in uscita, latch comanda di caricare il valore sul bus dataIn nel registro interno, shift comanda di porre sul segnale serialOut il bit successivo.

¹ PiSo: Parallel In/Serial Out register, usato per passare da una sequenza di bit di un byte ad un byte con i bit in parallelo

Tutte queste entità vengono utilizzate e “inglobate” all’interno dell’entità ps2keyboard_pong.

ps2keyboard_pong – gestisce tutta la comunicazione con la tastiera, con input e output su bus a 8 bit. Inoltre, questa entity si resetta automaticamente quando il decoder PS2 segnala un errore in ricezione; questo comportamento è necessario perché altrimenti i successivi byte ricevuti dalla tastiera vengono mal decodificati. I segnali che questa entity accetta in ingresso e in uscita sono:

clk: clock a 1 Mhz (input)

rstn: segnale di reset attivo basso (output)

ps2data: linea dati del bus PS2 (input/output)

ps2clk: linea di clock del bus PS2 (input/output)

send: quando asserito (posto a zero) comanda l’invio del byte posto su dataIn alla tastiera (input)

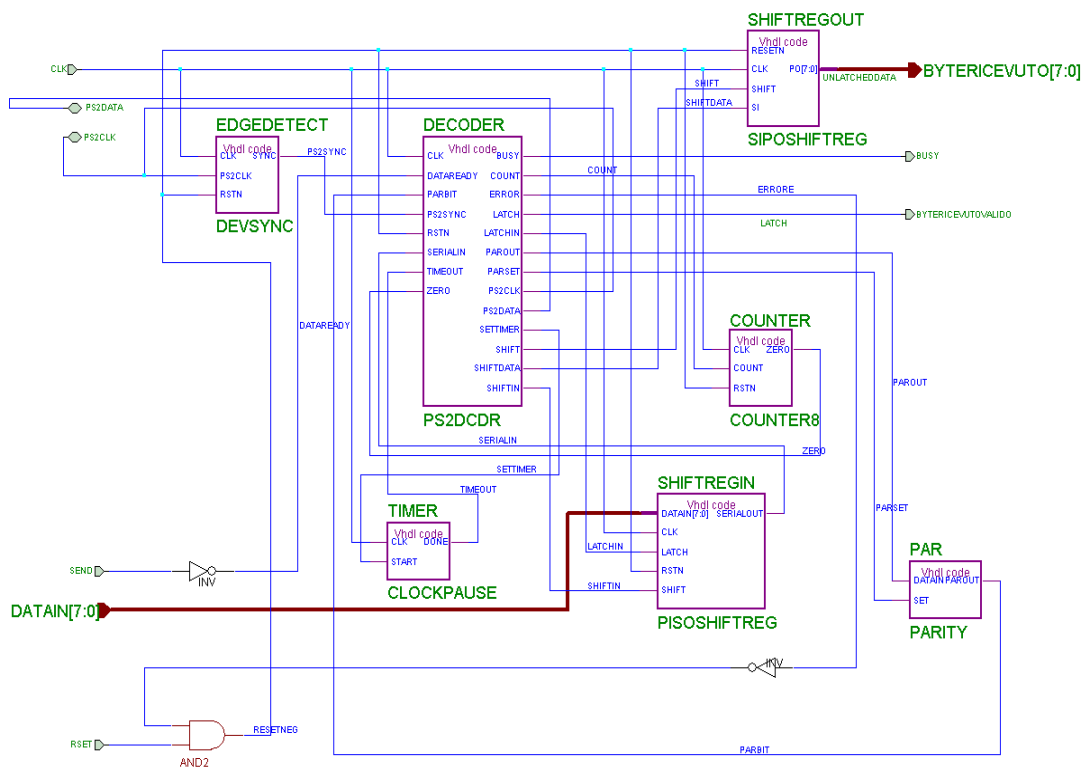
dataIn: byte da spedire alla tastiera (input)

busy: indica che il decoder PS2 è impegnato in una ricezione o una trasmissione (output)

bytericevuto: byte ricevuto dalla tastiera (output)

bytericevutovalido: asserito per un ciclo di clock quando è valido il valore su bytericevuto (output)

Per la comprensione del funzionamento di questa entity, che principalmente non fa altro che da collante per le entity sopradescritte, conviene riferirsi allo schema seguente. La rete con l’AND e l’invertitore è il circuito di reset che non viene comandato solo dall’esterno, ma che si attiva anche quando viene rilevato un errore in ricezione (linea errore asserita).

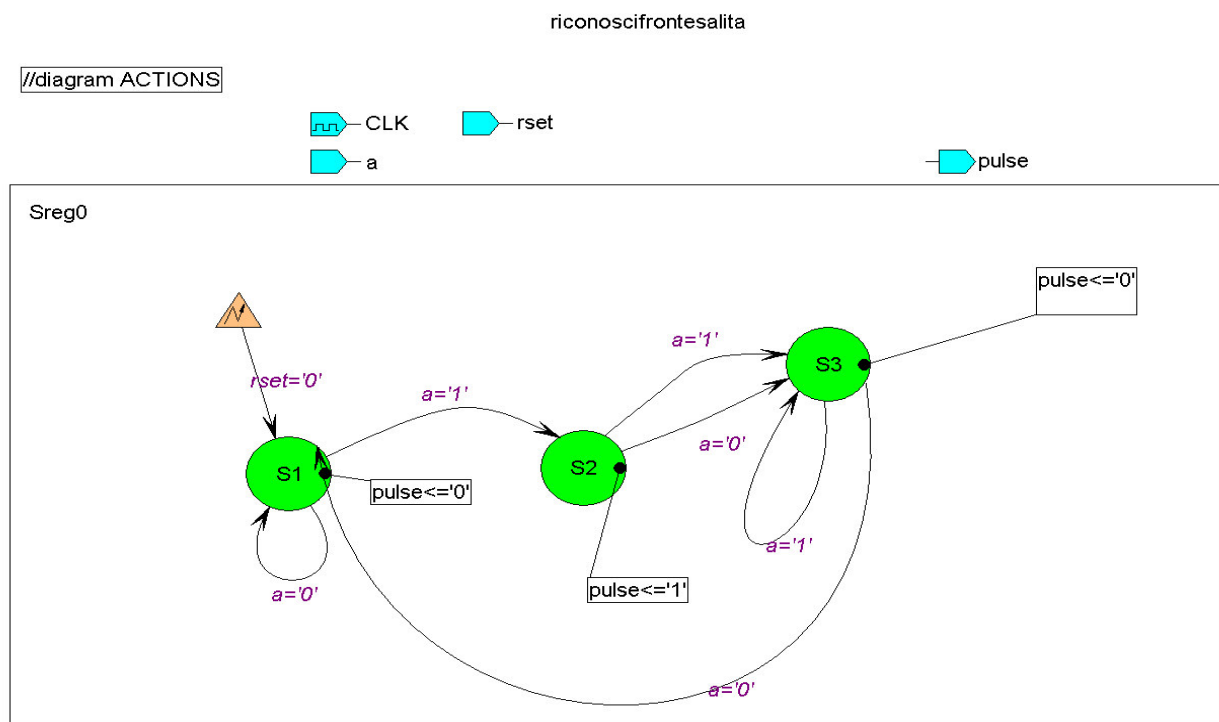


L’entity ps2keyboard_pong viene utilizzata assieme alle seguenti entity per formare l’entity GestioneTastieraPong:

divisore50 – divisore per 50. Utilizzato per portare il clock di 50Mhz generato dall'oscillatore sulla scheda XSV (vedi [4]) a 1Mhz che è il clock richiesto dalle entity di gestione della tastiera. clkin è il clock di ingresso a 50Mhz, clokout è il clock di uscita (1Mhz).

riconoscifrontesalita – Entity implementata come macchina a stati finiti. È un rilevatore di fronte di salita: l'output viene posto a 1 per un colpo di clock quando in input il segnale passa da 0 a 1; l'output rimane a zero fino a quando non si ha un altro fronte di salita. Il segnale a è l'input, pulse è il segnale di output a impulso che indica che sull'ingresso è stato notato un fronte di salita; clk è il clock, e rstn è il reset. Questa entity viene utilizzata per "accorciare" il segnale bytericevutovalido (che indica che il byte sul bus bytericevuto è un byte valido); il segnale bytericevutovalido in effetti sarebbe anch'esso un impulso (vedere file ps2dcd.vhd) ma poiché la gestione della tastiera implementata nell'entity ps2keyboard_pong avviene con un clock di 1mhz, rispetto al resto del progetto il segnale latch e' un "impulso" lungo 50 colpi di clock, quindi si utilizza questa entity per accorciarlo.

Schema della macchina a stati finiti:



riconoscitasti – Entity che genera impulsi su 5 segnali distinti ogni volta che il byte ricevuto dalla tastiera coincide con lo scan code di uno dei tasti di interesse; in pratica, un comparatore. Questo circuito è asincrono. CharIn è il bus parallelo a 8 segnali su cui porre il byte da confrontare con gli scan codes di interesse; pulse è un segnale ad impulso che indica quando su charIn il dato è valido; pl1su, pl1giu, pl2su, pl2giu e pausa sono i segnali a impulso in uscita; sono attivi solo per un colpo di clock e solo quando su charIn c'è lo scan codes dei tasti (rispettivamente) A, Z, K, M, P.

configtastiera – Macchina a stati finiti che invia una sequenza di comandi di configurazione alla tastiera.

Il byte da inviare viene posto sulla linea Byte, collegato al bus dataIn dell'entity ps2keyboard_pong; l'invio si ha quando la linea startsend, collegata al piedino send dell'entity ps2keyboard_pong, viene posto a 0 (il segnale è attivo quando basso). Da notare che il byte da inviare deve essere stabile sul bus byte quando la linea startsend viene asserita, quindi il dato viene posto sempre 2 colpi di clock prima di asserire startsend (altrimenti, da prove effettuate, il funzionamento risulta instabile). Dopo l'invio di un comando bisogna aspettare un certo tempo per dare alla tastiera la possibilità di eseguire il comando. Questo viene fatto facendo partire un timer esterno (vedere entity delay131ms) asserendo la linea startdelay; si continua inviando il comando successivo quando il delay asserisce la linea delaydone. Fa eccezione l'invio del primo comando, che resetta la tastiera, in cui si usa un delay più lungo (mezzo secondo circa) sfruttando l'entity delayresettast attraverso le linee startresetdelay e doneresetdelay, poiché la tastiera impiega più tempo per resettarsi e completare il self test.

Alla tastiera vengono spediti questi 5 byte, nell'ordine:

0xFF: reset tastiera

0xF0: imposta scancode set

0x03: scancode set da impostare

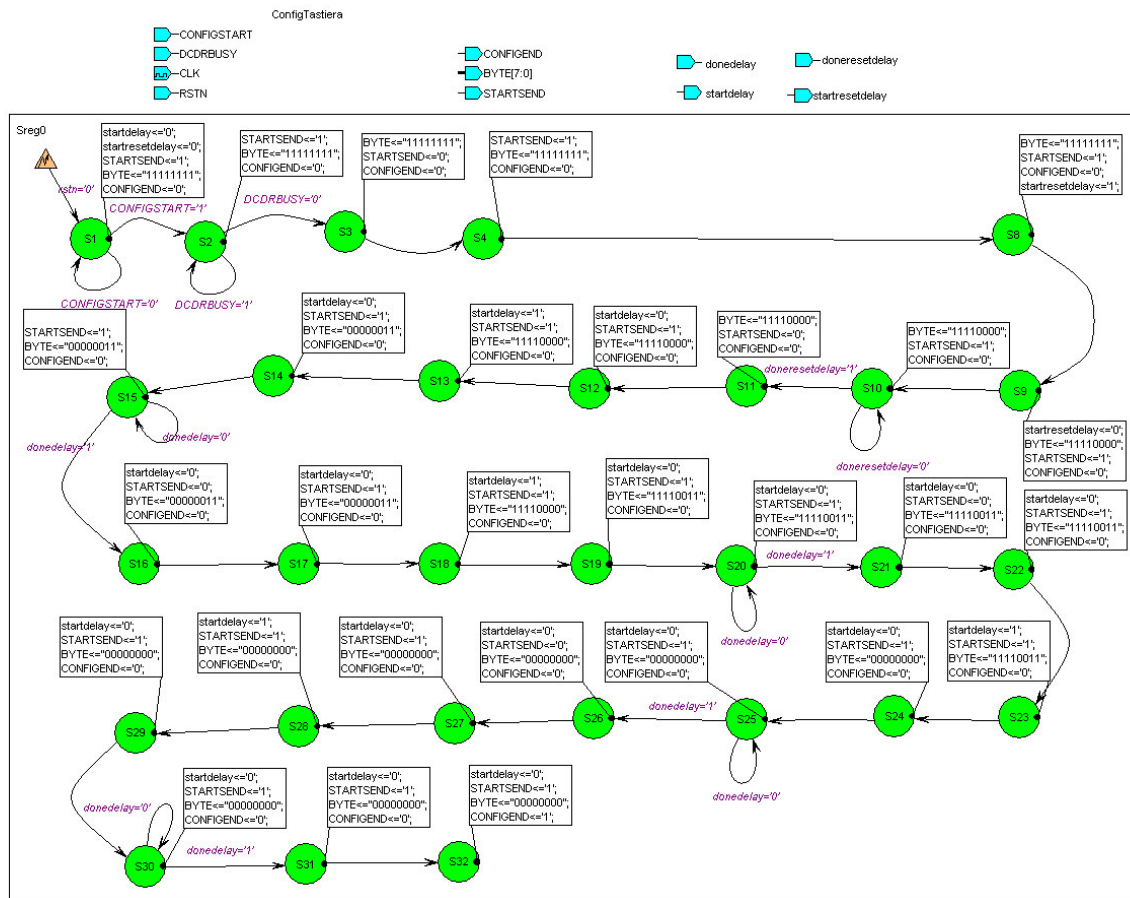
0xF3: imposta velocità ripetizione

0x00: velocità di ripetizione massima, delay pre-ripetizioni minimo.

La configurazione della tastiera ha inizio quando, dopo il reset generale della scheda, viene asserita la linea configstart. Simmetricamente, quando sono stati inviati tutti i comandi, l'entity configtastiera asserisce la linea configend.

Inoltre, prima di spedire il primo comando, come misura cautelativa, si aspetta che il decoder ps2 sia libero (aspetto finché il segnale dcdrrbusy è a zero).

Questo è il grafico della macchina a stati:



Per ogni byte spedito si usano 8 stati. Da notare che l'invio del byte n-esimo (cioè, nell'istante in cui startsend viene posto a 0) viene fatto subito dopo che è trascorso il delay per il comando (n-1)-esimo, e analogamente si prepara l'invio del byte (n+1)-esimo (caricando sul bus byte il codice dell' (n+1)-esimo tasto) poco prima di avviare il delay di attesa a seguito dell'invio del byte n-esimo. Ad esempio, queste sono le operazioni svolte dalla macchina a stati finiti, stato per stato, per l'invio del byte 0xF0:

- S8 -> avvia il timer startresetdelay (assegnazione "startresetdelay <= '1' ") che aspetta la conclusione dell'elaborazione del comando 0xFF (comando di reset)
- S9 -> poni sul bus byte il secondo comando da trasmettere (0xF0)
- S10 -> mantieni sul bus byte il valore 0xF0 e aspetta finché il delay si è concluso
- S11 -> invia il byte 0xF0 (assegnazione "startsend <= '0' ")
- S12 -> rimetti a 1 il segnale startsend
- S13 -> fa partire il timer che attende il completamento dell'esecuzione del comando ("startdelay <= '1' ")
- S14 -> rimetti a zero startdelay e carica il nuovo comando da spedire sul bus Byte (0x03)
- S15 -> aspetta finché il delay di attesa di completamento del comando 0xF0, fatto partire in S13, non si conclude.
- S16 -> invia il byte 0x03 alla tastiera ("startsend <= '0' ")

delayresettast – timer utilizzato dalla macchina a stati finiti configtastiera per attendere il completamento dell'esecuzione del comando di reset, che impiega un tempo maggiore degli altri

comandi per essere eseguito. start è il segnale da asserire per resettare il contatore e farlo partire, done è il segnale asserito quando il timer ha concluso di contare.

delay131ms – timer utilizzato dalla macchina a stati finiti configtastiera per attendere il completamento dell'esecuzione dei comandi che non sono quello di reset; inoltre viene utilizzato dall'entity gestionetastierapong per comandare il segnale configstart che fa partire l'invio dei comandi di configurazione alla tastiera. Come per delayresettast, start è il segnale da asserire per resettare il contatore e farlo partire, done è il segnale asserito quando il timer ha concluso di contare.

gestionetastierapong – Questa entity racchiude in sé tutta la parte di gestione della tastiera del gioco Pong, sfruttando le entity descritte qui sopra e l'entity ps2keyboard_pong che si occupa della comunicazione con la tastiera sul bus PS2. In particolare, essa genera il clock a 1mhz che tutte le entity per la gestione della tastiera sviluppate dall'Università del Queensland richiedono, riceve gli scan codes dalla tastiera, riconosce quali tasti sono stati premuti e nel caso siano i tasti per il controllo delle pad e della pausa, attiva i segnali a impulso in uscita corrispondenti; inoltre al reset invia i comandi di configurazione alla tastiera.

I segnali di entrata e uscita di questa entity sono, oltre al clock a 50Mhz e al reset:

ps2clk: linea di clock del bus PS2

ps2data: linea dati del bus PS2

pl1su: segnale a impulso che indica la pressione del tasto A

pl1giu: segnale a impulso che indica la pressione del tasto Z

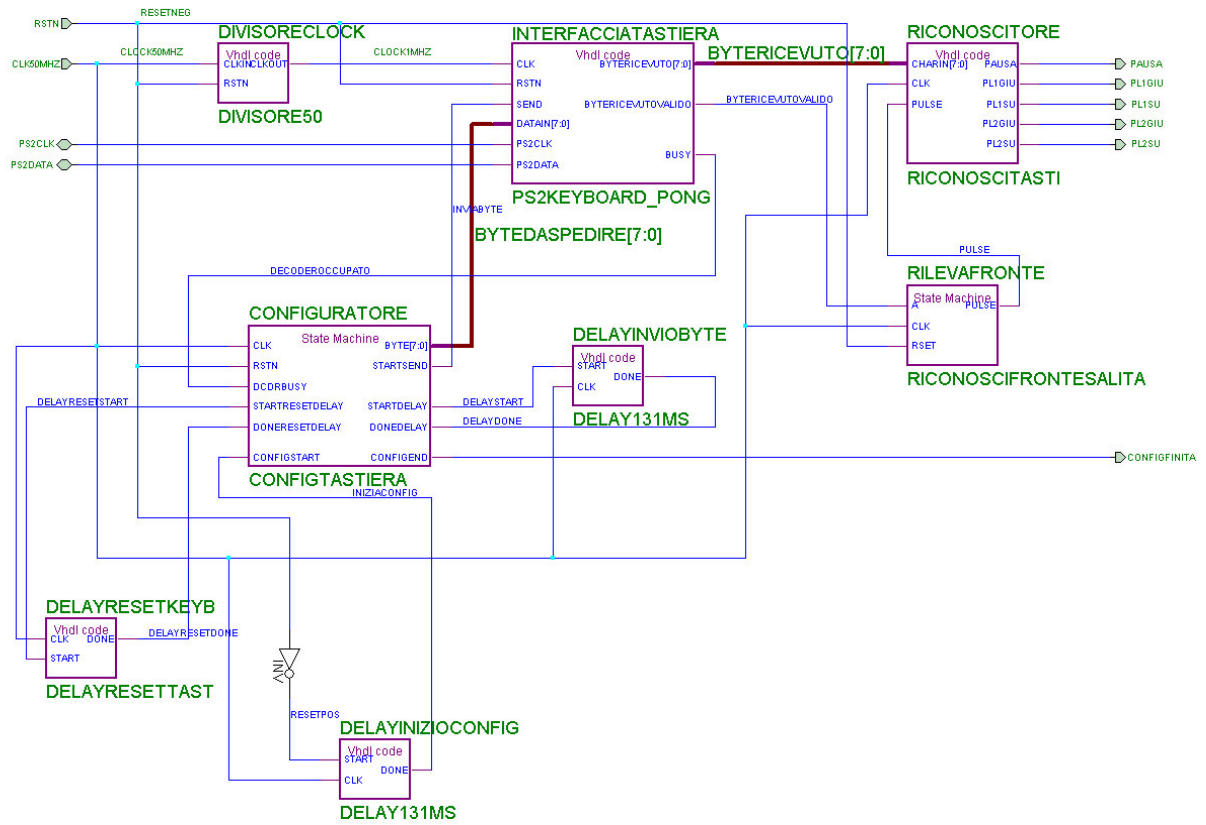
pl2su: segnale a impulso che indica la pressione del tasto K

pl2giu: segnale a impulso che indica la pressione del tasto M

pausa: segnale a impulso che indica la pressione del tasto P

configfinita: asserito quando la configurazione della tastiera è completa (i 5 byte sono stati inviati)

Questo è uno schema dell'entity gestionetastierapong:



La Gestione dell'Uscita VGA

Per la gestione dell'interfaccia VGA è stato modificato il progetto "VGA Interface for the XSV Board" creato dalla UDQ (vedi [5]).

L'uscita VGA presente sulla scheda può essere pilotata o direttamente dall'FPGA, che genera i segnali di sincronismo e i segnali R, G, B del colore, oppure utilizzando il RAMDAC Bt481A (per il datasheet, vedi [3]) e facendo generare alla FPGA solo i segnali di sincronismo (vedi pag. 23 di [1]).

Il RAMDAC può essere utilizzato in due modalità differenti (vedi [3]): Nella prima modalità (detta "palette"), i segnali del colore sono generati a partire da un indice che seleziona il colore con cui rappresentare il pixel attuale da una tabella interna in cui sono contenuti i valori delle componenti rosso, verde e blu di 256 colori impostabili. Nella seconda modalità (detta "true color"), le componenti del colore vengono passate direttamente dall'esterno, e questo rende possibile la generazione di uno qualunque dei colori possibili per ogni pixel. I colori possibili possono essere 16.777.216, 65536 o 32768 a seconda del numero di bit per canale che si sceglie (rispettivamente 8:8:8, 5:6:5 o 5:5:5).

L'indice o la componente viene comunicata sul bus P (bus a 8 bit) (vedi [3], pag. 7 e 8). Nel caso della modalità palette, l'indice è memorizzato in concomitanza con il fronte di salita del clock del RAMDAC, detto pixel clock. Per il caso della modalità true color si possono scegliere due protocolli di comunicazione: protocollo Dual-Edge Clock e protocollo Single-Edge Clock. Con il primo protocollo, il RAMDAC campiona il bus P sia sul fronte di salita che sul fronte di discesa del clock; con il secondo, si utilizza sempre solo il fronte di salita. Quindi utilizzando la modalità true color, possono servire 2 o 3 cicli di clock a seconda della profondità colore voluta e il protocollo scelto.

Per il progetto del gioco Pong, si è utilizzata la modalità palette in quanto bastavano pochi colori.

Per programmare i registri della palette e i registri di configurazione, si utilizza il bus composto da RS, D, RD, WD (vedi [3], pag. 9-11). RD e WR devono essere asseriti per leggere o scrivere sul registro specificato dalle linee RS; il dato da trasferire (sia in lettura che scrittura) si trova sulle linee D. Per caricare la palette, si impostano le linee RS al valore 001 ("comanda la palette") e poi sul bus D si pongono in sequenza l'indice del colore della palette da modificare, la componente rossa, quella verde e quella blu. Alla fine dei quattro cicli di scrittura, il colore all'indice specificato viene modificato e inoltre l'indice stesso viene incrementato internamente al valore successivo, in modo da poter subito caricare le componenti del colore successivo.

Gli altri registri accessibili mediante il bus composto da RS, WR, RD e D sono il Command Register A e il Command Register B. Il primo controlla la profondità colore e il protocollo utilizzato sul bus P, il secondo funzionalità accessorie come la modalità Sleep (vedi [3], pag. 19-20).

Per ulteriori informazioni sul RAMDAC, consultare [3].

Alcune linee del RAMDAC sono condivise dal controller Ethernet e dal banco sinistro di memoria SRAM, che quindi devono essere disabilitati (vedi [1], pag. 24).

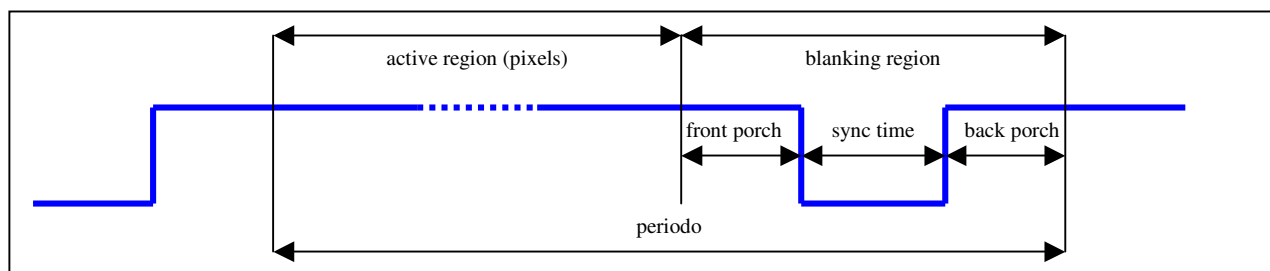
I segnali di sincronismo sono generati dalla FPGA, con il segnale di sincronismo verticale generato sugli overflow del segnale di sincronismo orizzontale, secondo lo standard VGA (vedere la descrizione dell'entity *vgacore* per maggiori informazioni).

Descrizione dei file VHD di gestione dell'interfaccia VGA

Le entity VHDL che gestiscono l'uscita VGA sono:

vgacore – genera i segnali di sincronismo verticale e orizzontale, e indica anche le coordinate del pixel che viene disegnato istante per istante (sui due bus in uscita *hloc* e *vloc*). Nel file *vgacore.vhd*

è presente anche una descrizione, che riportiamo, di come devono essere gestiti i segnali di sincronismo per aderire allo standard VGA.



I segnali devono essere allo stato logico alto nella nell'area in cui vengono disegnati i pixel (active region), e poi andare allo stato logico basso (e rimanervi per un tempo detto “sync time”) nella regione in cui i pixel non devono essere disegnati sullo schermo (blanking region) dopo un tempo detto “front porch”. I segnali di sincronismo tornano allo stato logico alto entro un tempo chiamato “back porch” prima di uscire dalla blanking region. La durata dei tempi “sync time”, “front porch” e “back porch” è calcolata a partire dalla risoluzione dello schermo desiderata.

L'andamento dei segnali di sincronismo orizzontale e verticale è lo stesso, cambia solo il concetto di blanking region (e quindi di active region): per il sincronismo orizzontale la blanking region è il tempo in cui il pennello elettronico transita dalla fine della linea corrente all'inizio della linea successiva (horizontal retrace), per il sincronismo verticale il tempo in cui esso ritorna dalla fine dell'ultima linea all'inizio della prima (vertical retrace) , secondo il caratteristico movimento a zig-zag.

Questa entity è composta da 5 statement process: il primo genera il contatore orizzontale (il periodo è pari al tempo di linea, cioè il tempo necessario per disegnare i pixel della linea più il tempo necessario per front porch e back porch più il tempo di sync in cui il segnale di sincronismo orizzontale è basso), il secondo il contatore verticale (incrementato ad ogni overflow del contatore orizzontale; il periodo è calcolato analogamente al caso orizzontale), il terzo comanda il valore del segnale di sincronismo orizzontale come spiegato sopra, il quarto il valore del segnale di sincronismo verticale e l'ultimo process asserisce il segnale di blanking (per inibire gli output colore del RAMDAC nelle regioni non attive) a seconda della posizione del pixel data dai contatori orizzontale e verticale. I due bus hloc e vloc in uscita (coordinate del pixel attuale) sono determinati dai valori correnti dei contatori orizzontale e verticale.

prgramdac – macchina a stati che si occupa di programmare il RAMDAC Bt841a al reset, scrivendo negli appositi registri di configurazione e inizializzando i colori nella palette. La macchina a stati funziona pilotando le linee RS, D e WR in maniera opportuna per programmare i valori di configurazione nei registri A e B, quindi passa a riempire la palette (vedi [3]). La palette viene riempita con 8 colori differenti, ognuno ripetuto 32 volte.

La palette è programmata con questa sequenza di colori:

Indice palette	Colore
0-31	Nero
32-63	Blu
64-95	Verde
96-127	Celeste
128-159	Rosso
160-191	Viola
192-223	Giallo
224-255	Bianco

La programmazione del RAMDAC inizia con l'asserzione della linea start dell'entity; la configurazione è completata quando la linea done viene asserita. WRn, RDn, data, RS sono le linee che pilotano il RAMDAC.

vga – Entity top della gestione della interfaccia VGA. Genera i segnali di sincronismo usando l'entity *vgacore*, programma il RAMDAC al reset utilizzando l'entity *prgramdac* e in seguito disegna sullo schermo le pad, la pallina, i bordi superiore e inferiore del campo di gioco, e colora lo sfondo di nero durante il gioco e di viola durante la pausa. Inoltre disabilita l'interfaccia Ethernet e il banco destro di memoria SRAM.

I segnali di input e output di questa entity sono:

clk: segnale di clock (input)

rstn: reset asincrono attivo basso (input)

reg_pos_sx: posizione del centro della pad sinistra (input)

reg_pos_dx: posizione del centro della pad destra (input)

pos_x_pallina: posizione x del centro della pallina (input)

pos_y_pallina: posizione y del centro della pallina (input)

pixel: colore del pixel attuale (bus da collegare al bus P del RAMDAC) (output)

blankn: segnale di blank (disabilita le uscite del RAMDAC) (output)

RDn: comanda la lettura dei registri del RAMDAC (output)

WRn: comanda la scrittura dei registri del RAMDAC (output)

RAMDACD: linea dati per il RAMDAC (input/output)

RS: linea Register Select per il RAMDAC (input/output)

hsync: segnale di sincronismo orizzontale generato dall'entity *vgacore* (output)

vsync: segnale di sincronismo verticale generato dall'entity *vgacore* (output)

triste: per disabilitare l'interfaccia Ethernet (output)

rramce: per disabilitare il banco destro di memoria SRAM (chip enable) (output)

pixelclk: pixel clock, cioè il clock con cui viene pilotato il RAMDAC (output)

giocoattivo: seleziona il colore dello sfondo (input)

L'entity è composta da tre process (i primi due collegati logicamente). Il primo e il secondo process implementano una macchina a stati che comanda la partenza della configurazione del RAMDAC a seguito del reset. Il terzo blocco process si occupa di selezionare l'indice della palette da porre sul bus "pixel" a seconda delle coordinate del pixel che viene disegnato in quel momento e la posizione delle pad e della pallina. Per la scelta dell'indice sono utilizzate due funzioni, *intervallo_verticale_pad* e *regione_pallina*; entrambe ritornano un valore booleano che indica se il pixel corrente appartiene, rispettivamente, alla pallina (la condizione testata è se il pixel corrente è compreso fra i lati della pallina, calcolati in funzione della posizione del centro della pallina) o ad una pad (si testa la posizione verticale del pixel da disegnare, perché la posizione orizzontale viene testata fuori dalla funzione essendoci due pad: se la coordinata verticale del pixel è compresa fra quella del bordo alto e quella del bordo basso, si ritorna il valore booleano vero, altrimenti falso). L'entity definisce alcune costanti per i colori e altre per le dimensioni di pad, limiti del campo e pallina e le distanze della pad e dei limiti del campo dai bordi dello schermo. Queste costanti devono essere uguali a quelle dichiarate nell'entity *algoritmpong* che gestisce tra le altre cose le collisioni, se non si vuole avere una visualizzazione coerente con i comandi inviati.

Descrizione dei file VHD che implementano il gioco Pong

Passiamo infine a descrivere le entity VHDL che gestiscono il nucleo del gioco e la top level entity:

algoritmipong – questa è l'entity che si occupa di:

- muovere la pallina, con la velocità orizzontale impostata dall'utente mediante il dipswitch 1 sulla scheda XSV800;
- muovere le pad sinistra e destra a seconda dei tasti premuti dall'utente;
- capire quando la pallina colpisce una pad e quando la manca;
- aggiungere un punto al giocatore opposto a quello che non prende la pallina;
- decidere in maniera casuale la direzione del moto della pallina all'inizio della partita;
- decidere in maniera casuale l'angolo di riflessione della pallina dopo la collisione con una delle pad;
- inibire o consentire gli spostamenti della pallina e delle pad a seconda che il gioco sia o non sia in pausa;
- modificare gli incrementi di spostamento lungo l'asse x e y della pallina e lungo l'asse y delle pad a seconda della velocità della pallina.

Questa entity è una entity sincrona composta da 5 blocchi sequenziali (blocchi process) e una funzione generatrice di numeri pseudocasuali.

Gli ingressi e le uscite dell'entity sono:

clk: clock a 50mhz (input)

rstn: segnale di reset per l'entity attivo basso (input)

nuovapartita: quando posto a 0 riporta le pad e la pallina in posizione centrale (input)

pl1su: segnale che indica che il tasto A è stato premuto (input)

pl1giu: segnale che indica che il tasto Z è stato premuto (input)

pl2su: segnale che indica che il tasto K è stato premuto (input)

pl2giu: segnale che indica che il tasto M è stato premuto (input)

pos_pad_sx: indica all'entity VGA la coordinata y del centro della pad sinistra (output)

pos_pad_dx: indica all'entity VGA la coordinata y del centro della pad destra (output)

pos_x_pallina: indica all'entity VGA la coordinata x del centro della pallina (output)

pos_y_pallina: indica all'entity VGA la coordinata y del centro della pallina (output)

vincesx: asserito quando la pallina ha toccato il limite destro dello schermo (output)

vincedx: asserito quando la pallina ha toccato il limite sinistro dello schermo (output)

enable: da asserire per abilitare il movimento della pallina e delle pad (input)

switchvelocita: imposta la velocità orizzontale della pallina (input)

Il primo blocco process si occupa di incrementare (spostare verso il basso) o decrementare (spostare verso l'alto) la coordinata y del centro delle pad a seguito della pressione dei tasti A, Z, K, M, limitando però la massima e minima coordinata in modo che la pad resti confinata nel campo di gioco.

Il secondo blocco divide il clock a 50Mhz per 2^{20} (circa un milione) per ottenere un clock da 47,6Hz che è il clock con cui si alimenta il terzo blocco, che gestisce l'incremento e il decremento delle coordinate (secondo le quantità *incrementoxpallina* e *incrementoyballina*) del centro della pallina (detto in altri termini, realizza lo spostamento della pallina sullo schermo) secondo la direzione (verso l'alto/verso il basso; verso sinistra/verso destra) attuale di movimento.

Il valore *incrementoxpallina* vale 2 pixel se sono nella modalità facile di gioco, 4 pixel se sono in quella difficile (la modalità è determinata dal segnale *switchvelocita*); i valori possibili di *incrementoyballina* sono 0,1,2,3 per il gioco facile, e 0,2,4,6 per il gioco difficile, corrispondenti a movimento orizzontale (0 gradi), movimento a 22,5 gradi, movimento a 45 gradi e movimento a 67,5 gradi (i gradi sono dell'angolo di riflessione, rispetto alla normale alle pad). Il valore di *incrementoyballina* è determinato da un altro segnale, *fattoreincrementoy*, che viene generato

casualmente ad ogni collisione. Il legame fra i valori dei due segnali è stabilito nel quinto blocco process.

Il quarto blocco rileva le collisioni fra la pallina e le pad, pallina e bordo campo e pallina e bordo sinistro e destro dello schermo, confrontando le coordinate. Quando viene rilevata una collisione con una delle due pad, si utilizza la funzione pseudorandom() per generare un numero intero da assegnare a `fattoreincrementoy`, che determina il valore di `incrementoy` della pallina come spiegato sopra. Inoltre viene invertita la direzione orizzontale della pallina.

Quando si rileva la collisione con uno dei bordi del campo invece si inverte unicamente la direzione verticale della palla.

Quando si rileva la collisione con il bordo destro o sinistro del campo, viene asserita la linea `vincesx` o `vincedx` (rispettivamente) per indicare che il giocatore sinistro o quello destro ha segnato un punto e viene inoltre calcolata la direzione iniziale della pallina per la successiva partita.

Il movimento della pallina e delle pad è consentito solo se il gioco non è in pausa (vedere i costrutti IF con la condizione di `enable` pari a 1 nei process che comandano il movimento della pallina e le pad).

Per la generazione dei numeri casuali si utilizza un contatore a 32 bit (sesto blocco process) azzerato al reset della scheda XSV800, e una rete logica di xor sui bit del contatore, implementata nella funzione pseudorandom(). Assumendo che l'utente non inizi la prima partita sempre allo stesso istante di clock, i bit del contatore si troveranno sempre in uno stato differente ogni volta che viene invocata la funzione pseudorandom(), garantendo risultati pseudocasuali.

fsmgioco – macchina a stati finiti che controlla lo svolgersi del gioco. Gli ingressi e le uscite sono:

`clk`: clock

`rstn`: reset asincrono attivo basso

`resetgioco`: comanda all'entity `algoritmipong` di spostare pad e pallina in posizione centrale (output)

`giocoinpausa`: indica alle entity `algoritmipong` e `vga` che il gioco deve considerarsi in pausa (output)

`incpuntisx`: per incrementare il punteggio del giocatore sinistro (output)

`incpuntidx`: per incrementare il punteggio del giocatore destro (output)

`vincesx`: asserito da `algoritmipong` quando la pallina raggiunge il bordo destro dell'area (input)

`vincedx`: asserito da `algoritmipong` quando la pallina raggiunge il bordo sinistro dell'area (input)

`configtastierafinita`: da asserire quando la configurazione della tastiera è completata (input)

`premutopausa`: da asserire quando l'utente ha premuto il tasto della pausa (input)

Diamo qui di seguito la descrizione del funzionamento di questa macchina a stati finiti. Per la comprensione conviene riferirsi allo schema alla pagina successiva.

Al reset la macchina parte dallo stato `Inizio`, che resetta l'entity `algoritmipong` (uscita `resetgioco` posta a zero) e da cui esce solo quando viene asserito il segnale `configtastierafinita` che indica che la configurazione della tastiera è stata completata (il segnale è generato dall'entity `GestioneTastieraPong`).

Si passa quindi allo stato `Waste`, dove viene tolta la condizione di reset, e quindi allo stato `StandBy`, che pone l'uscita `giocoinpausa` a 1, quindi ponendo il gioco in pausa. Dallo stato `StandBy` si esce e si passa allo stato `Gioco` quando l'utente preme il tasto della pausa (condizione `premutopausa='1'`). Lo stato `Gioco` è quello in cui l'entity `algoritmipong` accetta gli input dell'utente e muove le pad e la pallina e rileva le collisioni della pallina con i bordi campo, i bordi schermo e le pad. Dallo stato `Gioco` si può andare o nello stato `Pausa`, quando l'utente preme il tasto della pausa (stato nel quale la linea `giocoinpausa` viene nuovamente asserita, inibendo quindi i movimenti di pad e pallina; si torna in `Gioco` quando viene di nuovo premuto il tasto pausa), oppure si procede verso i due stati `PuntiDX` o `PuntiSX` quando l'entity riconosce l'arrivo della pallina ad uno dei bordi destro o sinistro dello schermo (segnale `VinceDX` o `VinceSX` asserito). Nei due stati `PuntiDX` e `PuntiSX` il gioco viene fermato (`giocoinpausa <= 1`) e i segnali `incpuntidx` e `incpuntisx` (a seconda del caso) vengono posti a 1 per un ciclo di clock (il segnale viene asserito nello stato `PuntoDX` o `PuntoSX`, a seconda del caso, e viene tolta l'asserzione negli stati `WasteDX` e `WasteSX`). Indipendentemente da chi

Gli ingressi e le uscite di questa entity sono:

clk: clock (input)

rstn: reset asincrono attivo basso (input)

incrementa: quando a 1 incrementa il contatore ad ogni ciclo di clock (input)

punti: bus a 4 bit che mantiene il valore corrente del contatore (output)

seg7 – decoder a sette segmenti. Pilota le linee di un display a sette segmenti per visualizzare in esadecimale un valore fra 0 e 15 su un bus a 4 bit. Ne vengono istanziati due per visualizzare i punteggi dei due giocatori.

Gli ingressi e le uscite di questa entity sono:

binary: bus a 4 bit in ingresso con il valore da visualizzare (input)

hex: bus di 7 linee che pilotano un display a sette segmenti (output)

pong – entity top level del progetto: utilizzando l'entity *GestioneTastieraPong*, l'entity *vga* e le entity descritte qui sopra (*algoritmipong*, *fsmgioco*, *rallentaripetizioni*, *contatorepunteggio*, *seg7*) realizza il gioco Pong.

Gli ingressi e le uscite di questa entity sono:

clk50mhz: clock a 50Mhz (input)

rstn: reset asincrono attivo basso (input)

ps2data: linea dati bus PS2 (input/output)

ps2clk: clock bus PS2 (input/output)

hexouthi: bus del display a sette segmenti sinistro (output)

hexoutlo: bus del display a sette segmenti destro (output)

switchvelocita: dipswitch 1, utilizzato per scegliere la velocità di gioco

pixel: colore del pixel attuale (bus collegato al RAMDAC) (output)

blankn: segnale di blank (disabilita le uscite del RAMDAC) (output)

RDn: comanda la lettura dei registri del RAMDAC (output)

WRn: comanda la scrittura dei registri del RAMDAC (output)

RAMDACD: linea dati per il RAMDAC (input/output)

RS: linea Register Select per il RAMDAC (input/output)

hsync: segnale di sincronismo orizzontale generato dall'entity *vgacore* (output)

vsync: segnale di sincronismo verticale generato dall'entity *vgacore* (output)

triste: per disabilitare l'interfaccia Ethernet (output)

ramce: per disabilitare il banco destro di memoria SRAM (output)

pixelclk: pixel clock (output)

pin230: segnale sempre fisso a massa, da collegare al pin 230 della FPGA (output)

Si tratta di una entity che principalmente collega fra di loro le altre entity sopra elencate, senza aggiungere nessuna logica.

L'entity *GestioneTastieraPong* si occupa di programmare la tastiera al reset in maniera opportuna e in seguito di segnalare quando vengono premuti i tasti A, Z, K, M, P all'entity *algoritmipong* che cambia opportunamente le posizioni delle pad e che gestisce autonomamente il moto della pallina, la rilevazione delle collisioni con i bordi superiore e inferiore dell'area di gioco e con i bordi sinistro e destro dello schermo.

L'entity *algoritmipong* e' collegata all'entity *VGA* che si occupa di visualizzare le pad e la pallina sullo schermo nelle posizioni stabilite da *algoritmipong*, gestendo tutta la parte di generazione dell'immagine sul monitor collegato all'uscita VGA sulla scheda XSV800, dalla generazione dei sincronismi all'impostazione del RAMDAC.

Inoltre *algoritmipong* è collegata anche con la macchina a stati finiti *fsmgioco* che si occupa di gestire le partite e la pausa gioco e di aggiornare i punti dei due giocatori quando *algoritmipong* segnala che la pallina ha toccato il bordo sinistro o destro dello schermo.

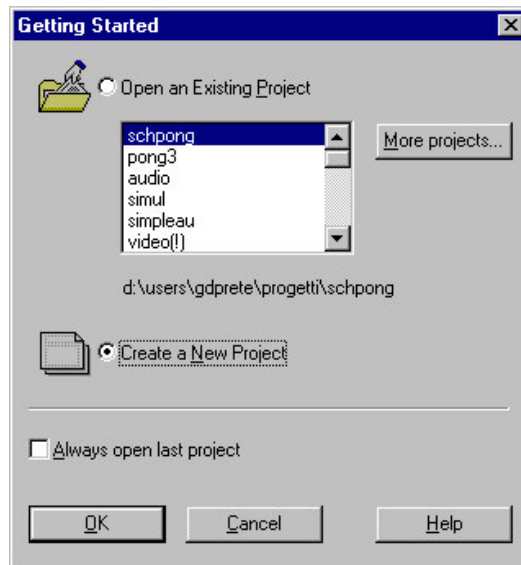
I punti vengono mantenuti da due contatori sincroni e visualizzati sui due display a sette segmenti.

Istruzioni per la compilazione del progetto

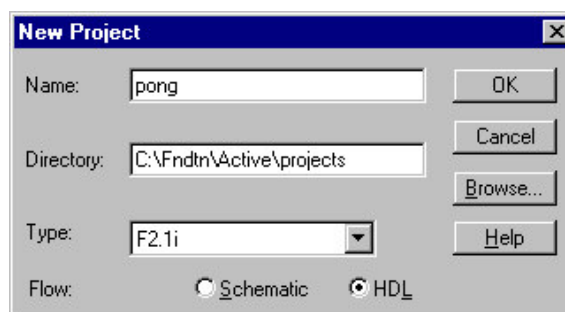
Vengono forniti i file sorgente VHDL e i file delle macchine a stati finiti (file ASF) delle entity descritte sopra.

Per creare il file .bit per programmare la scheda XSV800, attenersi a questa procedura:

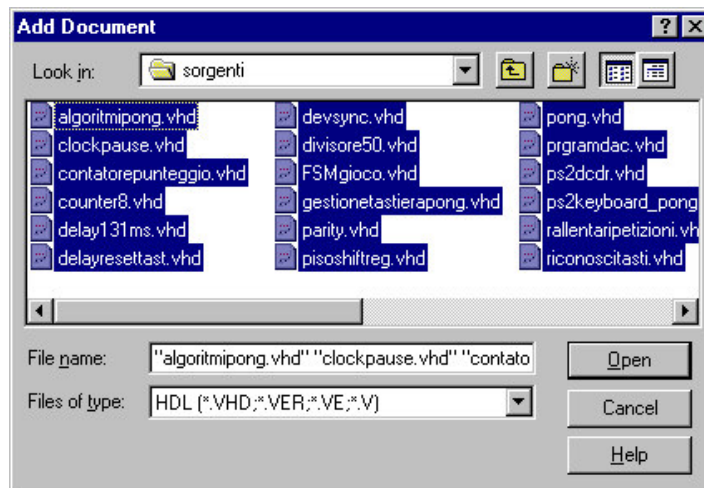
- 1) Aprire il tool Project Manager di Xilinx Foundation Series 2.1i facendo doppio click sull'icona relativa, presente solitamente sul desktop o nel menù Start di Windows.
- 2) Al lancio del Project Manager appare automaticamente la finestra "Getting Started". Selezionare "Create a New Project" e cliccare su "Ok", come da figura.



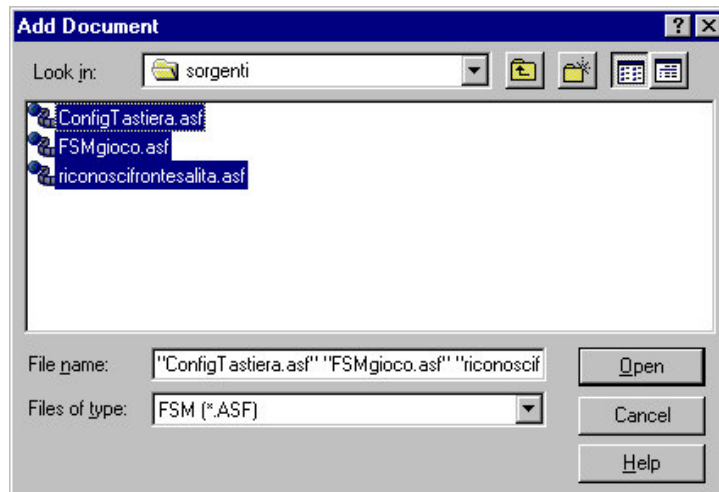
- 3) Compilare la finestra "New Project". Inserire il nome del progetto (si consiglia "Pong" per non dover rinominare in seguito il file di constraints), selezionare la directory in cui creare il progetto utilizzando il bottone "Browse...", quindi impostare il tipo di Flow Chart a "HDL", come mostrato in figura. Premere "Ok".



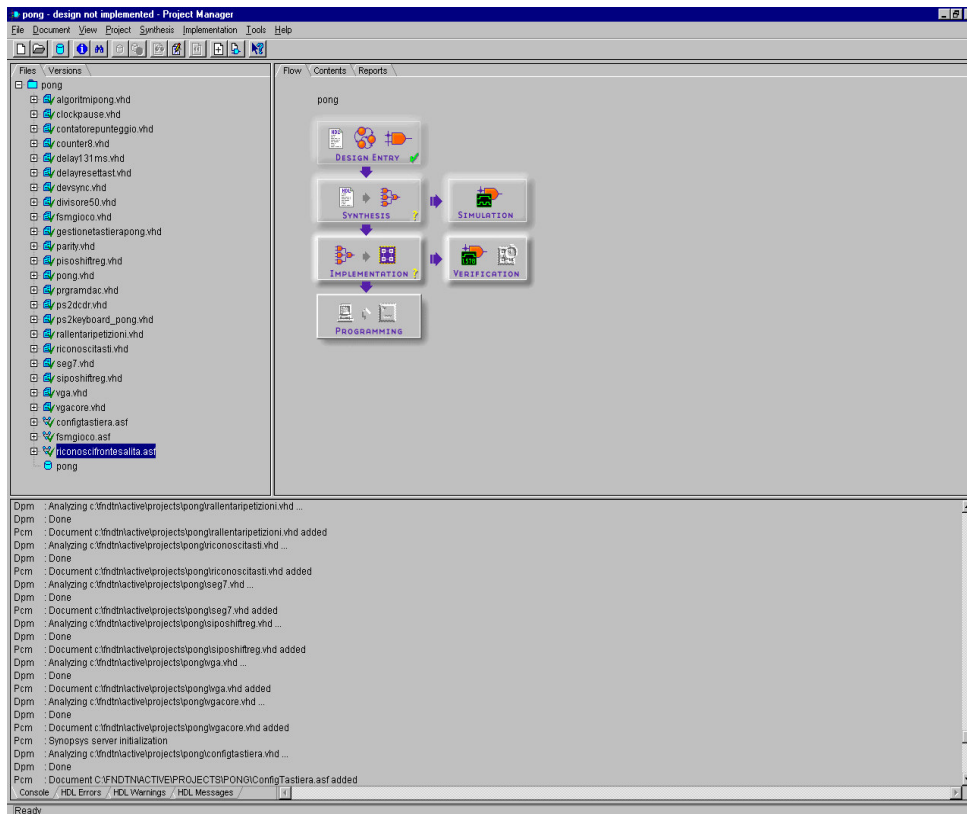
- 4) La finestra principale di Project Manager diventa attiva. Da menù "Project" selezionare la voce "Add Source File(s)...". Selezionare nella finestra "Add Document" che appare la directory contenente i file sorgente VHDL. Quindi selezionare con il mouse tutti i file presenti nella directory (conviene usare la scorciatoia da tastiera Ctrl+A), e cliccare su "Open".



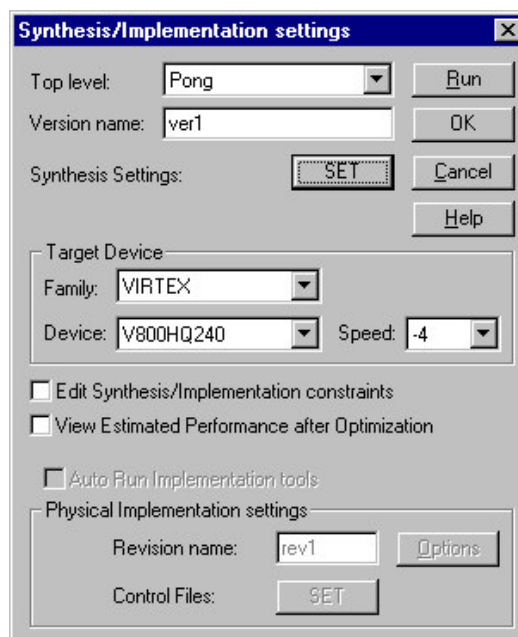
- 5) Selezionare di nuovo la voce “Add Source File(s)...” del menù “Project”, ma questa volta cambiare nella finestra “Add Document” il campo “Files of type” da “HDL” a “FSM”. Portarsi di nuovo nella directory contenente i sorgenti VHDL e selezionare questa volta tutte le macchine a stati finiti. Cliccare su “Open”.



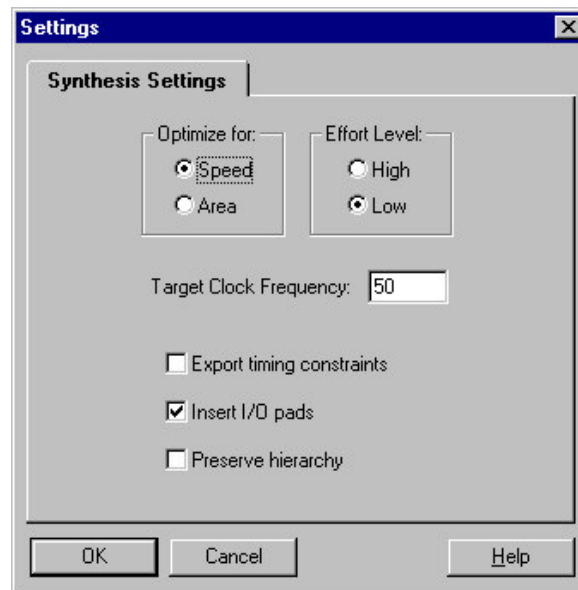
- 6) La finestra principale di Project Manager a questo punto mostra sulla sinistra tutti i file VHDL e le macchine a stati aggiunte al progetto.



- 7) Adesso si può far partire la parte di sintesi cliccando sul bottone “Synthesis” della Flow Chart sulla destra della finestra. Così facendo, appare la finestra “Synthesis/Implementation settings”. Bisogna a questo punto selezionare come “Top level” l’entity “Pong”, e impostare i campi della parte “Target Device” come in figura, cioè ponendo “Family” a “VIRTEX”, “Device” a “V800HQ240” e “Speed” a “-4”.

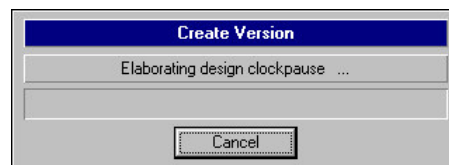


Il bottone “SET” accanto alla dicitura “Synthesis Settings” apre un’altra finestra che consente di impostare altri parametri. Assicurarsi che i campi siano impostati come nella figura seguente:



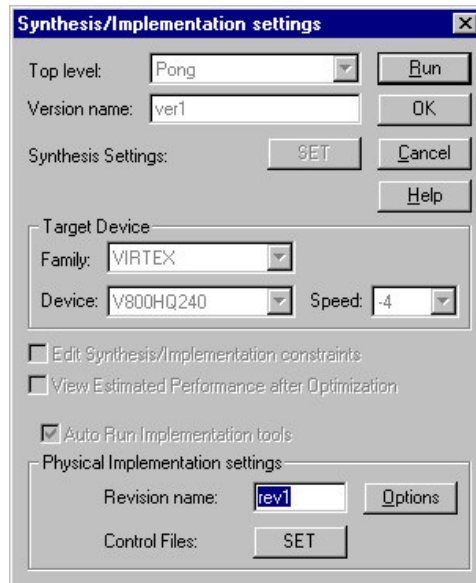
Dopo aver chiuso la finestra Settings premendo “Ok” si ritorna alla finestra “Synthesis/Implementation settings”. In questa finestra, premere il bottone “Run”. Questo fa partire la sintesi del progetto.

- 8) Le fasi della sintesi vengono descritte in una finestrella che compare in alto sullo schermo, simile a questa:

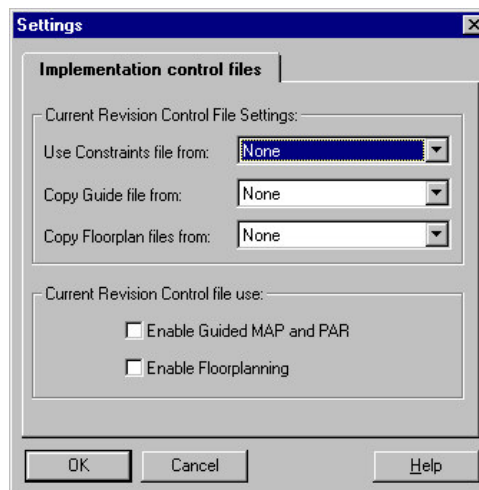


Dopo un po’, la fase di sintesi si conclude, e, se non ci sono errori, il riquadro “Synthesis” della Flow Chart ottiene un simbolo di “visto” color verde. Questo indica appunto che la sintesi si è svolta con successo.

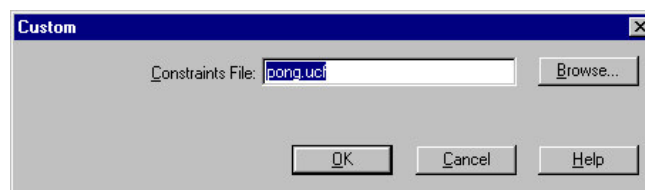
- 9) Si passa quindi alla fase di implementazione. Cliccare sul riquadro “Implementation”. Si apre nuovamente la finestra “Synthesis/Implementation settings”, ma questa volta sono attivi solo i due campi inferiori.



Cliccare sul bottone “SET” relativo alla dicitura “Control Files”. Appare un’altra finestra intitolata “Settings”.

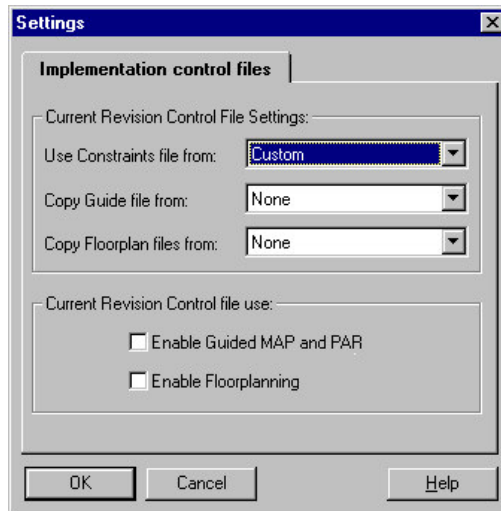


Nella casellina relativa a “Use Constraints file from:” selezionare la voce “Custom”; questo farà sì che si apra un’altra finestra, come in figura:



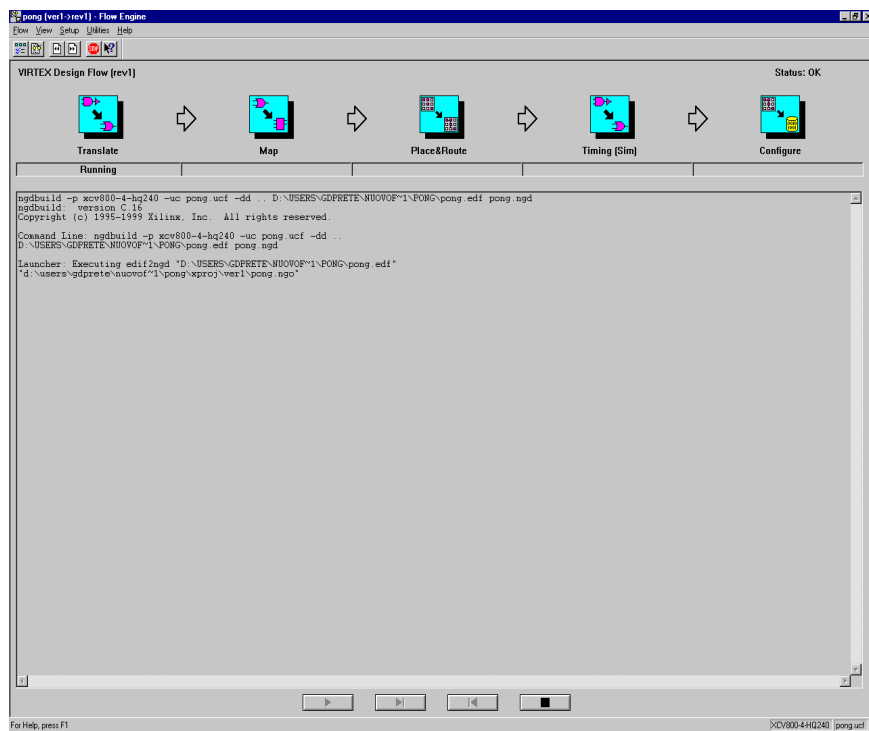
Utilizzando il bottone “Browse...” e la finestra di dialogo di selezione file che il bottone fa comparire, portarsi nella directory dei file sorgente e selezionare il file chiamato pong.ucf. Attenzione: la finestra “Custom” propone già un file chiamato “pong.ucf”, ma si tratta del file di constraints creato automaticamente alla creazione del progetto, e che è vuoto.

10) Premere quindi “Ok” nella finestra “Custom” per confermare il file pong.ucf. La finestra Settings ora mostra “Custom” alla voce “Use Constraints file from”:

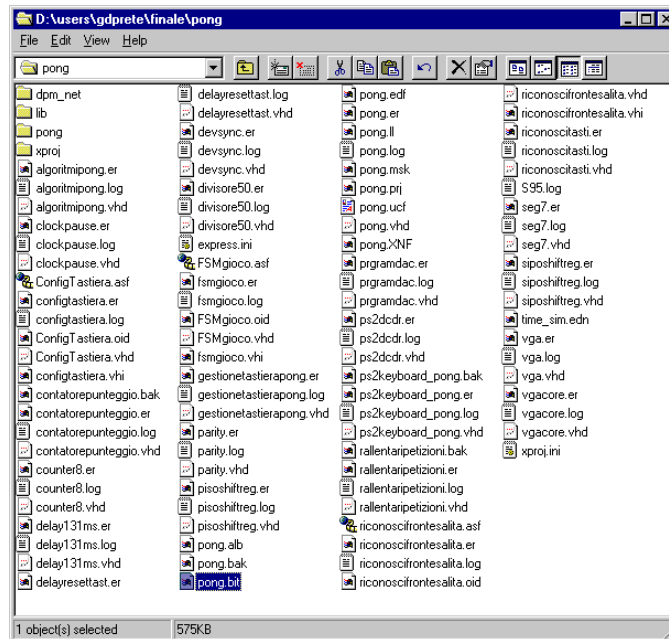


Premere “Ok” per chiudere la finestra “Settings”.

- 11) Ritornati alla finestra “Synthesis/Implementation settings”, premere “Run” per far partire la fase di implementazione del progetto. Appare la finestra Flow Engine che descrive l’andamento della fase di implementazione:



- 12) Quando la finestra Flow Engine scompare avendo completato tutte le fasi, è stato creato nella directory del progetto il file pong.bit da caricare nella FPGA:

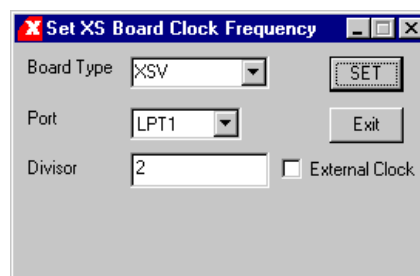


Istruzioni per la programmazione del clock

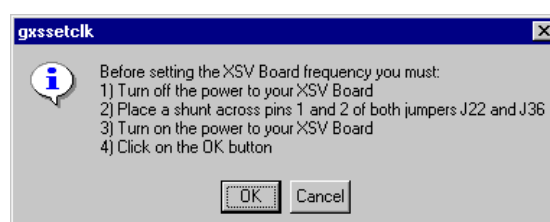
Il progetto Pong è stato pensato per essere eseguito dalla FPGA utilizzando un clock da 50Mhz. Nel caso la scheda XSV800 avesse il divisore di clock impostato per ottenere un'altra frequenza, bisogna riprogrammare la scheda per impostare il divisore di clock a 2.

Questi sono i passi da effettuare:

- 1) Lanciare il programma GXSETCLK cliccando sull'icona. Appare la finestra del programma:



- 2) Impostare i campi come in figura, cioè porre "Board Type" a "XSV", "Port" a "LPT1" (la porta parallela cui la scheda è collegata), "Divisor" a "2" e "External Clock" non abilitato. Premere quindi il bottone "SET".
- 3) Compare una finestra con l'elenco delle azioni da fare.



Basta seguire le istruzioni fornite, ricordandosi però che quando si deve togliere e ricollegare l'alimentazione, nello stesso passo si deve anche staccare e riattaccare il cavo della porta parallela (perché, come spiegato sul sito di XESS Corp., la porta parallela fornisce comunque energia alla scheda e questo impedisce al divisore di essere riprogrammato).

I passi da fare sono quindi:

- a) Togliere l'alimentazione e scollegare il cavo parallelo.
 - b) Impostare i jumper J22 e J36 entrambi nella posizione 1-2.
 - c) Collegare il cavo parallelo e ridare corrente alla scheda.
 - d) Premere il tasto "OK".
- 4) Viene riprogrammata la CPLD sulla scheda in modo da impostare il generatore di clock e quindi viene riprogrammato il divisore di clock.
- 5) Appare quindi un'altra finestra che indica le azioni da compiere per completare la configurazione:



- a) Staccare la corrente e il cavo parallelo.
- b) Posizionare il jumper J22 sulla posizione 2-3 e il jumper J36 sulla posizione 1-2.
- c) Ricollegare il cavo parallelo e ridare corrente alla scheda.
- d) Premere "OK" per chiudere la finestra.

In questo modo il divisore di clock è riprogrammato.

Bisogna però riprogrammare la CPLD per riattivare la programmazione della FPGA attraverso la porta parallela; per fare ciò, seguire le istruzioni della sezione successiva.

Come riabilitare la programmazione della FPGA attraverso la porta parallela

Per riprogrammare la CPLD affinché funga da "ponte" fra la porta parallela (il computer host) e la FPGA, seguire questi passi:

- 1) Eseguire il programma GXSLD. Appare la finestra del programma:



- 2) Usando il drag & drop, spostare il file `dnwldpar.svf` presente nella directory `C:\XSTOOLS` nell'area bianca in mezzo alla finestra.
- 3) La CPLD viene riprogrammata.

Istruzioni per il download del bitstream sulla scheda

Per caricare il file `pong.bit` compilato sulla scheda si utilizza il programma GXSLOAD.

- 1) Eseguire il programma GXSLOAD. Appare la finestra del programma:



- 2) Usando il drag & drop, spostare il file `pong.bit` frutto della compilazione nell'area bianca in mezzo alla finestra.
- 3) La FPGA viene programmata.
- 4) Alla fine della fase di programmazione la scheda si resetta e il programma viene eseguito. Il programma viene mantenuto nella FPGA fino a quando la scheda non viene spenta.

Istruzioni per il download del bitstream nella FlashRam

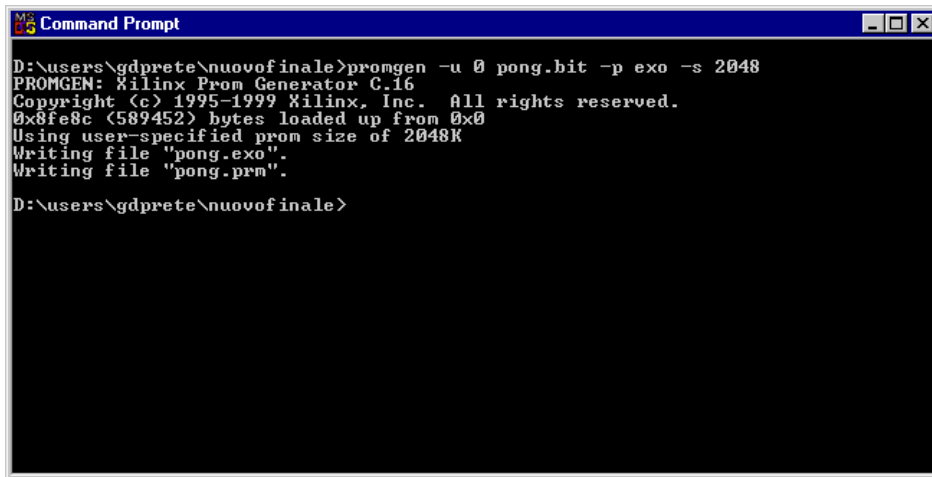
È possibile memorizzare un progetto compilato all'interno della memoria Flash presente sulla scheda, in modo che esso non venga cancellato quando si toglie l'alimentazione e venga eseguito appena viene data alimentazione alla scheda.

- 1) Per essere memorizzato nella Flash RAM, il file `pong.bit` deve essere convertito prima in un file in formato EXO. Per fare ciò, occorre aprire una finestra di MS-DOS ed eseguire il seguente comando:

```
promgen -u 0 pong.bit -p exo -s 2048
```

(l'opzione `-u` specifica l'indirizzo di inizio per la fase di programmazione, `-s` indica la grandezza della FlashRam, `-p` il formato del file da generare; per un elenco delle opzioni di `promgen`, basta eseguire il comando senza parametri)

Come da figura, vengono generati 2 file: `pong.exo` e `pong.prm`.



```
MS-DOS Command Prompt
D:\users\gdprete\nuovofinale>promgen -u 0 pong.bit -p exo -s 2048
PROMGEN: Xilinx Prom Generator C.16
Copyright (c) 1995-1999 Xilinx, Inc. All rights reserved.
0x8fe8c (589452) bytes loaded up from 0x0
Using user-specified prom size of 2048K
Writing file "pong.exo".
Writing file "pong.prm".

D:\users\gdprete\nuovofinale>
```

- 2) Eseguire il programma GXLOAD.
- 3) Assicurarsi che il banco di dipswitch abbia assolutamente tutti gli interruttori posti su OFF (come consigliato a pag. 14, in [1]).
- 4) Usando il drag & drop, spostare il file pong.exo appena creato nell'area bianca in mezzo alla finestra.
- 5) A questo punto, il programma GXLOAD compie le seguenti operazioni:
 - a) la memoria flash viene cancellata;
 - b) la CPLD sulla scheda viene programmata per fare da ponte fra il computer host collegato alla porta parallela e la Flash Ram;
 - c) la memoria flash viene programmata con il file EXO;
 - d) la CPLD viene riprogrammata in modo che all'accensione della scheda la FPGA si configuri utilizzando il bitstream memorizzato nella Flash.
- 6) Alla fine della fase di programmazione la scheda si resetta e il programma viene eseguito.

Quando si vuole caricare un nuovo programma sulla FPGA, è necessario riprogrammare la CPLD in modo da accettare i programmi dal computer host attraverso la porta parallela. Per fare ciò, attenersi alle istruzioni della sezione "Come riabilitare la programmazione della FPGA attraverso la porta parallela".

Bibliografia

- [1] “XSV Board V1.1 Manual”, XESS Corp.
(http://www.xess.com/manuals/xsv-manual-v1_1.pdf)
- [2] Virtex FPGA Datasheet
(<http://www.xilinx.com/bvdocs/publications/ds003.pdf>)
- [3] RAMDAC Bt481A Datasheet, Booktree Corporation
(http://www.erc.msstate.edu/~reese/EE4993/data_sheets/bt481a_c.pdf)
- [4] Dallas Semiconductor, DS1075 EconOscillator/Divider Datasheet
(<http://pdfserv.maxim-ic.com/arpdf/DS1075.pdf>)
- [5] Progetto “VGA Interface for the XSV Board” dell’Università del Queensland
[6] Progetto “PS2 Interface for the XSV Board” dell’Università del Queensland
(entrambi su <http://www.xess.com/ho03000.html#Examples>)
- [7] “PS/2 Mouse/Keyboard Protocol”, articolo sul funzionamento del bus PS2 di Adam Chapweske
(<http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/PS2/ps2.htm>)
- [8] “The AT/PS2 Keyboard Interface”
(<http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/keyboard/atkeyboard.html>)
- [9] “Introduzione al VHDL”
(<http://ticino.com/usr/pagna/Pagine/Documentazioni/Introduzione%20al%20VHDL.pdf>)
- [10] “VHDL Reference Manual”, Synario
(http://cslab.snu.ac.kr/course/cad99/vhdl_ref.pdf)
- [11] “VHDL”, Douglas L. Perry, McGraw-Hill, 1998.
Collocazione biblioteca tecnico-scientifica 21a/206
- [12] “Xilinx XC95108 In-System Programmable CPLD datasheet”
(<http://direct.xilinx.com/bvdocs/publications/95108.pdf>)