UNIVERSITÀ DI PISA

Progettazione del videogioco Breakout attraverso il linguaggio di descrizione dell'hardware Verilog

Studenti:
Antonio Rasulo
Antonio Di Vito

Indice

Introduzione

Lo scopo del progetto è stato quello di realizzare una versione semplificata del popolare videogioco “Arkanoid” attraverso il linguaggio di descrizione dell’ hardware Verilog e con l’utilizzo della board Altera DE1.
Il gioco consiste nel controllo da parte del giocatore di una navicella, allo scopo di colpire grazie ad una palla tutti i mattoni colorati presenti nell’area di gioco. E’ necessario prevenire la caduta della palla al di sotto del limite inferiore dello schermo, pena la perdita di una vita disponibile.
Il giocatore perde la partita quando esaurisce il numero di vite.

Caratteristiche della versione progettata

La versione del gioco che abbiamo progettato ha delle differenze rispetto al gioco originale, differenze dovute sia a questioni estetiche che a questioni di tempo:

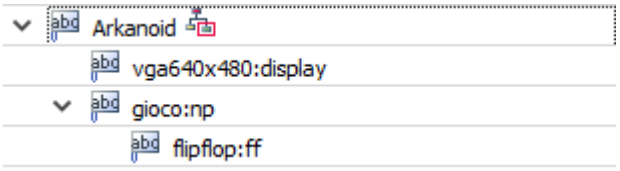
- Il numero di mattoni è pari a 10;
- Il numero di vite a disposizione del giocatore è pari a 3;
- I mattoni rossi hanno bisogno di 3 colpi per essere eliminati, i mattoni ciano hanno bisogno di 2 colpi e ai mattoni verdi basta un colpo;
- La palla si muove a 45° e non cambia velocità;
- Il numero di vite rimanenti viene visualizzato sul display a sette segmenti della board DE1;
- La pallina è rappresentata da un quadrato di dimensioni 2x2 pixel;
- I mattoni sono rappresentati da rettangoli 60x20 pixel;

Caratteristiche della scheda FPGA e del Software

Per la progettazione è stata usata una board prodotta dalla Terasic con FPGA della Altera, in particolare modello DE1.
Il software di sviluppo impiegato è Quartus II ideato dalla Intel Corporation, che ha permesso la programmazione Verilog dell’FPGA. Questo ambiente di sviluppo dispone di numerose risorse per l’analisi e la sintesi completa di un progetto di descrizione dell’hardware.

Gerarchia

La top level entity è il livello più alto della gerarchia e si interfaccia direttamente con la board DE1 attraverso i pin. La top level entity è il modulo “Arkanoid”. Il modulo Arkanoid va ad interfacciarsi sia col modulo “gioco” che col modulo “vga640x480”. Arkanoid sfrutta il modulo vga640x480 per poter visualizzare il videogioco sul monitor VGA. Per fare ciò ha bisogno di ricevere le informazioni grafiche del videogioco (posizione della palla, posizione della navicella, quali mattoni non sono stati eliminati e stato del videogioco); queste informazioni vengono fornite dal modulo “gioco”.
Il modulo gioco implementa una macchina di Mealy, ed utilizza un flipflop in modo da poter conservare le varie variabili (posizioni palla e navicella, numero di vite, ecc…) fra il passaggio da uno stato all’altro.



1.Gerarchia del progetto

Protocollo VGA

Si è adottato lo standard 640x480 @60Hz. I timing relativi a questa configurazione sono i seguenti:

a)

Pixel freq.	25.175 MHz

Scanline part	Pixels
Visible area	640
Front porch	16
Sync pulse	96
Back porch	48

b)

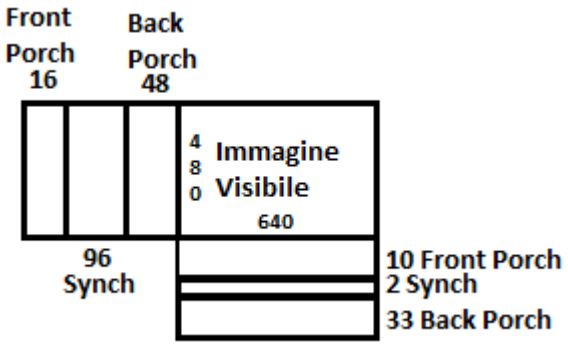
Frame part	Lines
Visible area	480
Front porch	10
Sync pulse	2
Back porch	33

c)

2.a)Clock dei pixel, b)Timing orizzontale, c)Timing verticale

Due dei segnali principali utilizzati dal protocollo VGA sono il sync orizzontale ed il sync verticale. Le sincronizzazioni garantiscono la corretta visualizzazione delle immagini a video. Definiscono l’inizio e la fine delle linee da visualizzare a video, evitando che a monitor l’immagine risulti tagliata e/o inclinata.

I segnali VGA hanno due fasi: una fase in cui vengono disegnati i pixel e l'intervallo di blanking. I segnali di sincronizzazione si verificano all'interno degli intervalli di blanking, separati dalla fase di disegno dei pixel dal front porch e dal back porch. L'intervallo di blanking permette di tornare all' inizio di una linea. Il clock richiesto dai pixel è all'incirca di 25 MHz; visto che la scheda può fornire un clock di 50 MHz, è bastato utilizzare un divisore di frequenza per ottenere il clock desiderato.
L'immagine sottostante chiarisce il significato di ogni valore:



3) Timing

La scheda controlla la porta VGA attraverso 3 bus di 4 bit per i canali colore (anche se per il progetto abbiamo utilizzato solo 2 bit per ogni colore) e due segnali per la sincronizzazione (orizzontale e verticale). Le analisi verranno effettuate considerando un piano cartesiano con l'origine nell'angolo in alto a sinistra del front porch, l'asse delle ascisse (d'ora in avanti ci si riferisce ad esso come asse x) è diretto da sinistra verso destra, mentre l'asse delle ordinate (d'ora in avanti ci si riferisce ad esso come asse y) è diretto dall'alto verso il basso.
Le coordinate dei pixel dell'immagine visibile verranno normalizzate in maniera tale che l'origine del piano cartesiano sia il lato in alto a sinistra del monitor.

Il modulo vga640x480

Porte

```
module vga640x480(
    input wire i_clk,           // clock
    input wire i_pix_stb,       // clock dei pixel
    input wire i_rst,           // reset
    output wire o_hs,           // sync orizzontale
    output wire o_vs,           // sync verticale
    output wire o_animate,      // alto quando vengono disegnati i pixel attivi (pixel dello schermo)
    output wire [9:0] o_x,      // posizione x del pixel corrente
    output wire [8:0] o_y       // posizione y del pixel corrente
    );
```

4) Porte del modulo vga640x480

Nell'immagine 4) vengono mostrate le porte di input e di output del modulo vga640x480:

- **i_clk**: input, è il clock da 50 MHz che viene fornito dalla board;
- **i_pix_stb**: input, è il clock da 25 MHz dei pixel, viene fornito dal modulo "Arkanoid";
- **i_rst**: input, è il pulsante di reset;
- **o_hs**: output, è la sincronizzazione orizzontale da fornire al monitor;
- **o_vs:** output**,** è la sincronizzazione verticale da fornire al monitor;
- **o_animate**: output, diventa alto quando vengono disegnati i pixel visibili, cioè i pixel del monitor;
- **o_x**: output, fornisce la coordinata x del pixel;
- **o_y**: output, fornisce la coordinata y del pixel;

Parametri e variabili

All'interno del modulo sono stati definiti dei parametri utili riguardanti il timing:

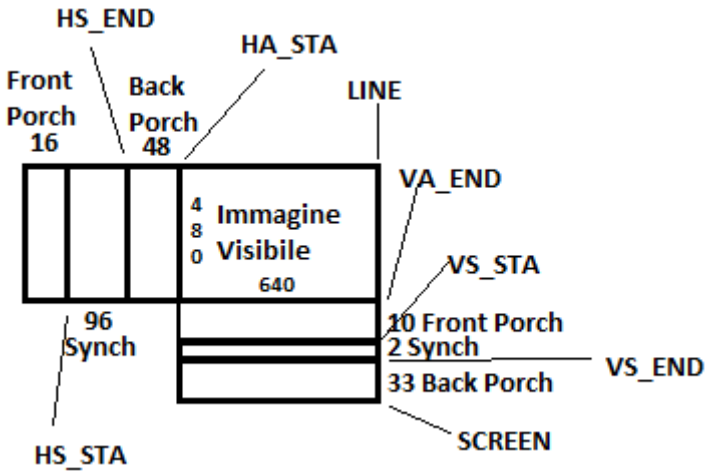
```
localparam HS_STA = 16;              // inizio sync orizzontale
localparam HS_END = 16 + 96;         // fine sync orizzontale
localparam HA_STA = 16 + 96 + 48;    // inizio dei pixel in orizzontale attivi (lato sx dello schermo)
localparam VS_STA = 480 + 10;        // inizio sync verticale
localparam VS_END = 480 + 10 + 2;    // fine sync verticale
localparam VA_END = 480;             // fine dei pixel in verticale attivi (lato in basso dello schermo)
localparam LINE   = 800;             // riga di pixel completa
localparam SCREEN = 525;             // colonna di pixel completa
```

5) Parametri definiti all' interno del modulo

Facendo riferimento alla figura 5):

- **HS_STA**: rappresenta l'inizio della porzione di sync orizzontale;
- **HS_END**: rappresenta la fine della porzione di sync orizzontale;
- **HA_STA**: rappresenta il lato sinistro dello schermo visibile;
- **VS_STA**: rappresenta l'inizio della porzione di sync verticale;
- **VS_END**: rappresenta la fine della porzione di sync verticale;
- **VA_STA**: rappresenta il lato in basso dello schermo visibile;
- **LINE**: rappresenta la fine della riga di pixel, notare che è la somma dei termini relativi al timing orizzontale;
- **SCREEN**: rappresenta l'ultima linea, notare che è la somma dei termini relativi al timing verticale.

  La seguente immagine aiuta a comprendere meglio il significato dei singoli parametri:



6) Spiegazione dei parametri presente nel modulo

All'interno del modulo vengono definite due variabili, che tengono conto della posizione riga e posizione colonna, utili per calcolare le sincronizzazioni, le coordinate dei pixel ed il parametro o_animate:

```
reg [9:0] h_count;  // posizione colonna
reg [9:0] v_count;  // posizione riga
```

7) Variabili definite all'interno del modulo

Blocco always

Di seguito viene mostrato come vengono valutate le variabili h_count e v_count:

```
always @ (posedge i_clk)
begin
    if (i_rst)  // reset
    begin
        h_count <= 0;
        v_count <= 0;
    end
    if (i_pix_stb)  // per ogni pixel
    begin
        if (h_count == LINE)  // nel caso di fine della linea, va all'inizio della linea successiva
        begin
            h_count <= 0;
            v_count <= v_count + 10'b 1;
        end
        else
            h_count <= h_count + 10'b 1;    //altrimenti va al pixel successivo sulla stessa linea

        if (v_count == SCREEN)  // fine dello schermo => si torna alla prima linea
            v_count <= 0;
    end
end
```

8) Blocco always presente all' interno del modulo

Le assegnazioni all'interno del blocco always sono non bloccanti, ciò significa che i valori vengono calcolati ma vangono aggiornati alla fine del blocco always. Si usa l'assegnazione non bloccate per descrivere una logica sequenziale. Nello specifico la

logica è sequenziale sincrona: la valutazione delle due variabili avviene sul fronte positivo del segnale i_clk (che come già visto in precedenza è il clock della scheda da 50 MHz). Nel caso di reset totale del sistema le due variabili assumono entrambe il valore “0”; facendo riferimento alla figura 3) ciò sta ad indicare che assumono il valore che rappresenta l’angolo in alto a sinistra del front porch orizzontale.
Si nota che ogni 25 MHz (come detto in precedenza e come vedremo nel dettaglio successivamente, i_pix_stb è un clock a 25MHz)
viene controllato se h_count ha raggiunto il valore LINE e se v_count ha raggiunto il valore SCREEN. Dalla figura 5 si nota che questi parametri valgono rispettivamente “800” e “525”; facendo riferimento alla figura 3), viene quindi verificato se h_count ha raggiunto la fine della linea, e se v_count è giunto all’ultima linea. Se h_count vale LINE quindi la variabile assume il valore “0” e v_count incrementa di uno, ciò rappresenta il passaggio all’inizio della linea successiva. Se invece h_count non vale LINE allora h_count viene incrementata, ciò rappresenta il passaggio alla colonna successiva; in questo caso la linea non cambia.
Per quanto riguarda v_count, se questa variabile assume il valore SCREEN, allora essa assumerà il valore “0”; ciò rappresenta il raggiungimento dell’ultima line e quindi la variabile viene portata alla prima riga.

Assegnamenti continui

Di seguito viene riportato come i vari parametri sono funzione delle variabili h_count e v_count in maniera continua.

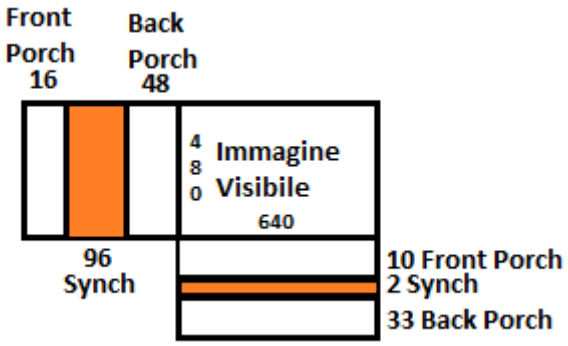
```
// generazione dei segnali di sync (sono attivi bassi per 640x480)
assign o_hs = ~((h_count >= HS_STA) & (h_count < HS_END));
assign o_vs = ~((v_count >= VS_STA) & (v_count < VS_END));

// si mantengono x ed y all'interno dei pixel attivi (pixel dello schermo)
assign o_x = (h_count < HA_STA) ? 0 : (h_count - HA_STA);
assign o_y = (v_count >= VA_END) ? (VA_END - 1) : (v_count);

// animate: alto quando vengono disegnati i pixel attivi (pixel dello schermo)
assign o_animate = ((v_count == VA_END - 1) & (h_count == LINE));
```

8) Assegnamenti continui presenti all’interno del modulo

Come già detto in precedenza, o_hs e o_vs sono rispettivamente le sincronizzazioni orizzontale e verticale da fornire al monitor vga. Per lo standard vga640x480 le sincronizzazioni sono attive basse. Si nota che o_hs vale “0” se h_count è compreso fra HS_STA e HS_END, cioè all’interno dell’intervallo di sync orizzontale. Analogalmente o_vs vale “0” se v_count è compreso fra VS_STA e VS_END, cioè all’interno dell’intervallo di sync verticale.



9) Intervalli in cui o_vs e/o o_hs sono attivi

Ricordiamo che o_x ed o_y forniscono le coordinate x ed y dei pixel dell'immagine visibile. Si nota che se h_count è minore ad HA_STA allora o_x assume il valore "0", questo perché se h_count è minore di HA_STA rappresenta una colonna che è fuori dall' immagine visibile (reminder: HA_STA rappresenta il lato sinistro del monitor); nel caso contrario invece o_x assumerà il valore h_count-HA_STA, che non è altro che la coordinata x se consideriamo che l'origine degli assi cartesiani sia nell'angolo in alto a sinistra del monitor. In maniera analoga, se v_count è maggiore di VA_END, allora o_y assume il valore VA_END-1, altrimenti assume il valore della variabile v_count; ricordando che VA_END rappresenta il lato in basso del monitor, nel primo caso si ha che o_y assume il valore della coordinata y dell'ultima linea di pixel visibili a schermo, nel secondo caso invece v_count assume il valore della coordinata y di una linea di pixel visibili a schermo, per questo motivo o_y assume il suo valore. O_animate è un segnale che assume il valore "1" quando vengono disegnati i pixel visibili a schermo, cioè quando v_count è uguale a VA_END-1 (cioè se rappresenta l'ultima linea di pixel visibili a schermo) e h_count è uguale a LINE (cioè se rappresenta l'ultima colonna). Questo segnale è utile per separare la fase di blanking con la fase di disegno dei pixel.

Il modulo Arkanoid

Il modulo Arkanoid è la top level del progetto. Questo modulo riceve le informazioni riguardanti il gioco dal modulo "gioco" e utilizza il modulo "vga640x480". È il modulo che si interfaccia con la board DE1. Inoltre, grazie a questo modulo, è possibile visualizzare il numero di vite a disposizione del giocatore sul display a sette segmenti della board.

Porte

```
module Arkanoid(
    input CLK,                  // clock della scheda
    input RST_BTN,              // bottone di reset
    input DX,                   // bottone dx, la navicella va a destra
    input SX,                   // bottone sx, la navicella va a sinistra
    input start,                // pulsante di start
    output VGA_HS_O,            // sync orizzontale
    output VGA_VS_O,            // sync verticale
    output reg VGA_RED,         // VGA_R[3] per il monitor VGA
    output reg VGA_GREEN,       // VGA_G[3] per il monitor VGA
    output reg VGA_BLUE,        // VGA_B[3] per il monitor VGA
    output reg RED,             // VGA_R[2] per il monitor VGA
    output reg GREEN,           // VGA_G[2] per il monitor VGA
    output reg BLUE,            // VGA_B[2] per il monitor VGA
    output reg [6:0] HEX0       // uscite per il display a sette segmenti
);
```

10) Porte di input e di output del modulo

Nell'immagine 10) vengono mostrate le porte di input e di output del modulo Arkanoid:

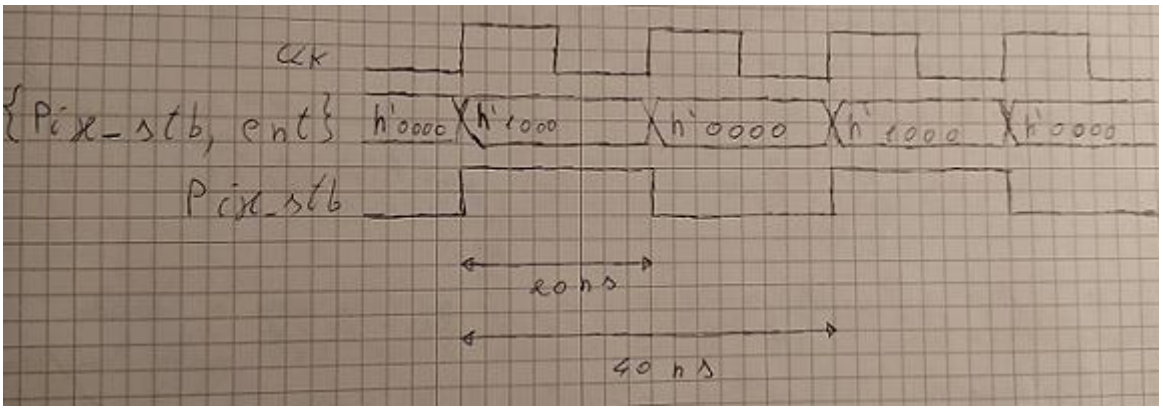
- **CLK**: input, è il clock da 50 MHz che viene fornito dalla board;
- **RST_BTN**: input, è il pulsante di reset;
- **DX**: input, è il pulsante utilizzato per muovere la navicella verso destra;
- **SX**: input, è il pulsante utilizzato per muovere la navicella verso sinistra;
- **start**: input, è il pulsante utilizzato per far iniziare una partita;
- **VGA_HS_O**: output, è la sincronizzazione orizzontale da fornire allo schermo;
- **VGA_VS_O**: output, è la sincronizzazione verticale da fornire allo schermo;
- **VGA_RED, VGA_GREEN, VGA_BLUE**: output, rappresentano rispettivamente i colori rosso, verde e blue da fornire al monitor VGA. Nello specifico questi tre pin vengono utilizzati per visualizzare i mattoni, la palla e la navicella;
- **RED, GREEN, BLUE**: output, vengono utilizzati in maniera analoga ai pin descritti precedentemente. Nello specifico vengono utilizzati per visualizzare le scritte di partita persa e di partita vinta;
- **HEX0**: output, rappresenta il vettore che pilota i led del display a sette segmenti.

Clock da 25MHz

Come già detto in precedenza, questo modulo fornisce al modulo vga640x480 il clock da 25MHz dei pixel. Ciò viene effettuato attraverso la definizione di un divisore di frequenza.

```
//generazione clock a 25 MHz per i pixel
reg [15:0] cnt = 0;
reg pix_stb = 0;
always @(posedge CLK)
   {pix_stb, cnt} <= cnt + 16'h8000;
```

11) Divisore di frequenza



12) Temporizzazione

Nella figura 12) è rappresentata la temporizzazione del divisore di frequenza, si nota che il periodo di pix_stb è doppio rispetto al periodo di CLK e quindi ha frequenza metà.

Wire e istanze

```
wire rst = ~RST_BTN;       // i KEY della scheda sono attivi bassi
wire [9:0] x;              // posizione x del pixel
wire [8:0] y;              // posizione y del pixel
wire animate;              // alto per un impulso quando vengono disegnati i pixel attivi (pixel dello schermo)
```

```
wire ok,ko;                          // permettono di capire se la partita è stata vinta o se è stata persa
wire navicella;                      // utile per disegnare la navicella
wire [9:0] nav_x1,nav_x2;            // lati laterali della navicella
wire [9:0] nav_y1,nav_y2;            // lati sopra e sotto navicella
wire palla;                          // utile per disegnare la palla
wire [9:0] pal_x1,pal_x2;            // lati laterali della palla
wire [9:0] pal_y1,pal_y2;            // lati sopra e sotto della palla
wire  bri1;                          // questi 10 wire dicono quali brick visualizzare e quali no
wire  bri2;
wire  bri3;
wire  bri4;
wire  bri5;
wire  bri6;
wire  bri7;
wire  bri8;
wire  bri9;
wire  bri10;
wire brick1;                         // questi 10 wire vengono usati per disegnare i vari mattoni
wire brick2;
wire brick3;
wire brick4;
wire brick5;
wire brick6;
wire brick7;
wire brick8;
wire brick9;
wire brick10;
reg B20,B21,B22,B23,B24,B25,B26,B27,B28,B29,B30,B31;          //vengono usati per la grafica di partita persa o vinta
wire [1:0] lfdisplay;                // fornisce il numero di vite, utile per visualizare sul display a sette segmenti
```

13) Wire definiti all'interno del modulo

Nella figura 13) sono mostrati i wire che vengono definiti all' interno del modulo. La maggior parte di questi wire vengono utilizzati per effettuare i collegamenti alle porte delle istanze dei moduli "vga640x480" e "gioco". Si nota che rst è il negato di RST_BTN, questo perché i pulsanti della scheda (detti anche KEY) sono attivi bassi e così facendo, quando ci si riferirà a rst si va ad utilizzare il bottone come se fosse attivo alto. x ed y rappresentano naturalmente la coordinata x e la coordinata y del pixel visibile a schermo.
animate rappresenta il segnale che è alto quanto vengono disegnati i pixel visibili a schermo.
ok e ko rappresentano due segnali che che vengono utilizzati dal modulo Arkanoid per capire se la partita è stata rispettivamente vinta o persa.
nav_x1 e nav_x2 rappresentano le coordinate x dei lati sinistro e destro della navicella e analogalmente nav_y1 e nav_y2 rappresentano le coordinate dei lati superiore ed inferiore della navicella; questi quattro wire vengono utilizzati per disegnare la navicella attraverso il wire navicella.
Discorso analogo può essere fatto per le coordinate dei lati della palla pal_x1, pal_x2, pal_y1 e pal_y2.
I wire briZ, con Z che va da 1 a 10, rappresentano dei segnali che fanno capire al modulo Arkanoid quali brick disegnare e quali no: se valgono 1 allora i brick non sono ancora stati eliminati e quindi vanno disegnati, altrimente se valgono 0 significa che il brick è stato eliminato e quindi non va disegnato.
I wire brickZ, con Z che va da 1 a 10, aiutano nel disegnare i vari mattoni.
Le variabili BZ, con Z che va da 20 a 31, vengono utilizzate per disegnare le scritte di partita persa e partita vinta.

Il wire lfdisplay rappresenta il numero di vite a disposizione del giocatore e viene utilizzato per visualizzarle sul display a sette segmenti della board.
Nell' immagine seguente vengono mostrate le istanze ai moduli vga640x480 e gioco:

```
vga640x480 display(
    .i_clk(CLK),
    .i_pix_stb(pix_stb),
    .i_rst(rst),
    .o_hs(VGA_HS_O),
    .o_vs(VGA_VS_O),
    .o_x(x),
    .o_y(y),
    .o_animate(animate)
);
```

```
gioco np(
    .i_clk(pix_stb),
    .i_ani_stb(pix_stb),
    .i_rst(rst),
    .i_animate(animate),
    .i_dx(DX),
    .i_sx(SX),
    .i_start(start),
    .o_x1_nav(nav_x1),
    .o_x2_nav(nav_x2),
    .o_y1_nav(nav_y1),
    .o_y2_nav(nav_y2),
    .o_x1_pal(pal_x1),
    .o_x2_pal(pal_x2),
    .o_y1_pal(pal_y1),
    .o_y2_pal(pal_y2),
    .bri1(bri1),
    .bri2(bri2),
    .bri3(bri3),
    .bri4(bri4),
    .bri5(bri5),
    .bri6(bri6),
    .bri7(bri7),
    .bri8(bri8),
    .bri9(bri9),
    .bri10(bri10),
    .ok(ok),
    .ko(ko),
    .ff_life(lfdisplay)
);
```

14) Istanze ai moduli vga640x480 e gioco

Disegno dei vari oggetti

Come già detto in precedenza, i vari oggetti vengono disegnati grazie all' aiuto di alcuni wire:

```
assign navicella =((x > nav_x1) & (y > nav_y1) &
    (x < nav_x2) & (y < nav_y2)) ? 1'b1 : 1'b0;
assign palla =((x > pal_x1) & (y > pal_y1) &
    (x < pal_x2) & (y < pal_y2)) ? 1'b1 : 1'b0;
assign brick1 =bri1 ?(((x > 138) & (y > 50) &
    (x < 202) & (y < 70)) ? 1'b1 : 1'b0):1'b0;
assign brick2 =bri2 ?(((x > 232) & (y > 50) &
    (x < 296) & (y < 70)) ? 1'b1 : 1'b0):1'b0;
assign brick3 =bri3 ?(((x > 346) & (y > 50) &
    (x < 410) & (y < 70)) ? 1'b1 : 1'b0):1'b0;
assign brick4 =bri4 ?(((x > 440) & (y > 50) &
    (x < 504) & (y < 70)) ? 1'b1 : 1'b0):1'b0;
assign brick5 =bri5 ?(((x > 184) & (y > 100) &
    (x < 248) & (y < 120)) ? 1'b1 : 1'b0):1'b0;
assign brick6 =bri6 ?(((x > 288) & (y > 100) &
    (x < 352) & (y < 120)) ? 1'b1 : 1'b0):1'b0;
assign brick7 =bri7 ? (((x > 392) & (y > 100) &
    (x < 456) & (y < 120)) ? 1'b1 : 1'b0):1'b0;
assign brick8 =bri8 ? (((x > 232) & (y > 150) &
    (x < 296) & (y < 170)) ? 1'b1 : 1'b0):1'b0;
assign brick9 =bri9 ? (((x > 346) & (y > 150) &
    (x < 410) & (y < 170)) ? 1'b1 : 1'b0):1'b0;
assign brick10 =bri10 ? (((x > 288) & (y > 200) &
    (x < 352) & (y < 220)) ? 1'b1 :1'b0):1'b0;
```

15) Assign presenti all'interno del modulo

Si nota che navicella vale "1" se le coordinate x ed y sono comprese fra le coordinate dei lati della navicella. Discorso analogo si può fare per il wire palla.
Per quanto riguarda i mattoni invece, affinchè essi vengano disegnati, non basta che le coordinate x ed y siano comprese fra le coordinate dei lati ma essi devono anche essere attivi, cioè i wire briZ (con Z che va da 1 a 10) devono valere "1".
Per disegnare i vari oggetti viene utilizzato il seguente blocco always:

```
always@(*)
    begin
        VGA_RED <= palla||brick1||brick2||brick3||brick4;
        VGA_GREEN <= palla||brick8||brick9||brick10||brick7||brick6||brick5;
        VGA_BLUE <= palla||brick5||brick6||brick7||navicella;
    end
```

16) Disegno dei vari oggetti

Per disegnare i vari oggetti vengono forniti i wire relativi alle porte di output VGA_RED, VGA_GREEN e VGA_BLUE, che successivamente verranno collegate ai pin della board relativi rispettivamente alle tonalità più accese dei colori rosso, verde e blu.

Disegni di partita persa o partita vinta

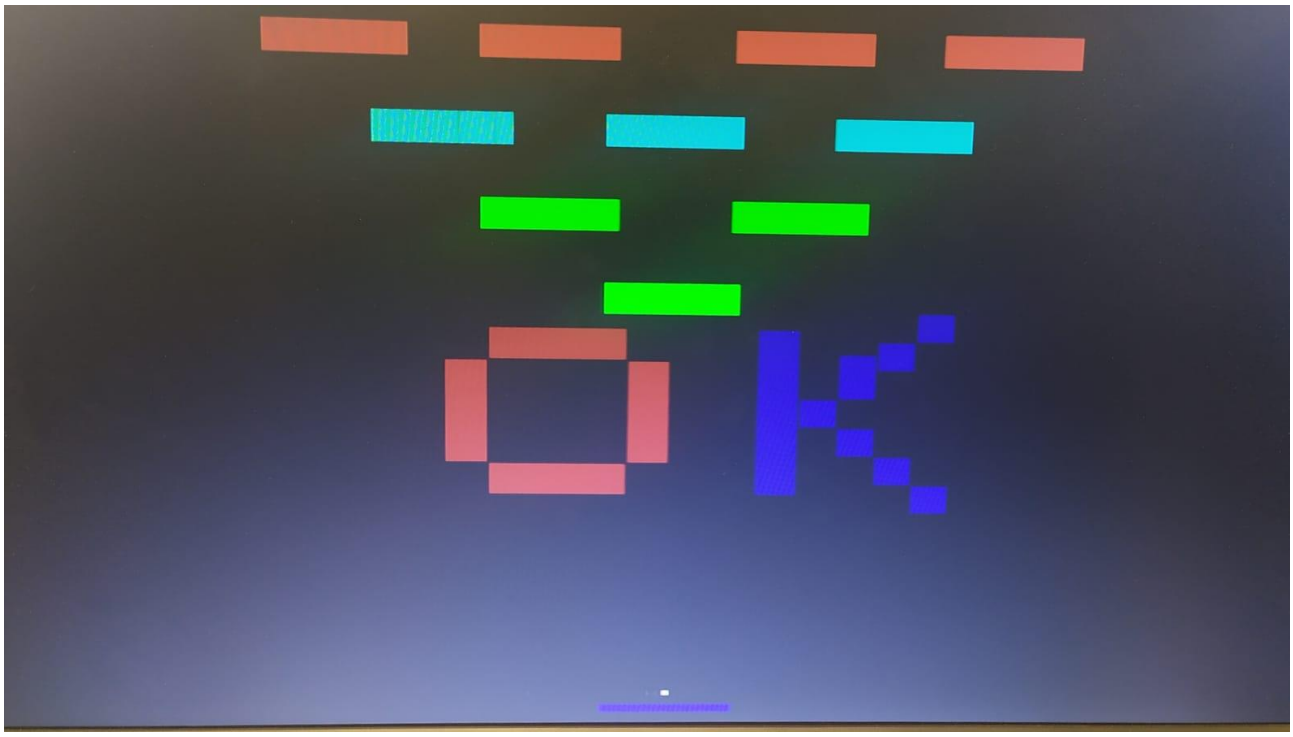
Il modulo, per capire se la partita è stata vinta o è stata persa utilizza i wire ok e ko.

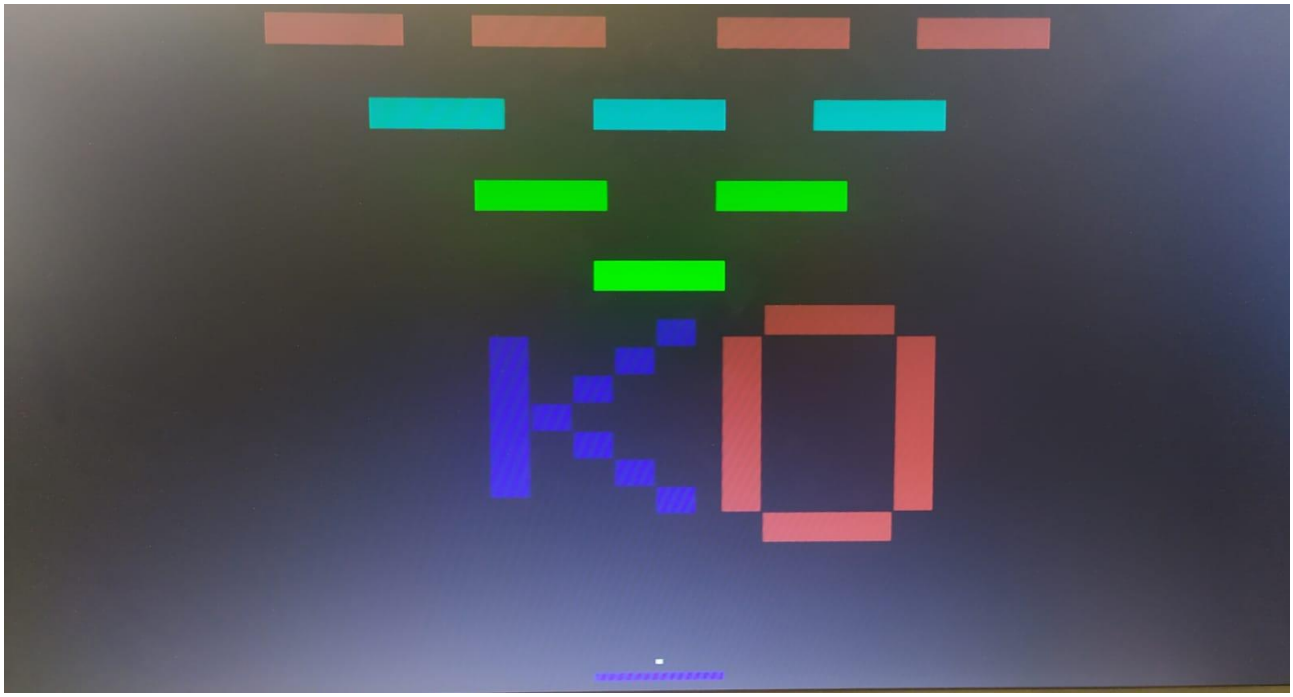
```
always@(*)
begin
    if(ok)  // nel caso di partita vinta ok è alto e viene disegnata la scritta "ok"
    begin
        B21<=((x>216)&(x<236)&(y>248)&(y<312))?1'b1:1'b0;
        B20<=((x>236)&(x<300)&(y>228)&(y<248))?1'b1:1'b0;
        B22<=((x>236)&(x<300)&(y>312)&(y<332))?1'b1:1'b0;
        B23<=((x>300)&(x<320)&(y>248)&(y<312))?1'b1:1'b0;
        B24<=((x>360)&(x<380)&(y>228)&(y<332))?1'b1:1'b0;
        B25<=((x>380)&(x<398)&(y>271)&(y<289))?1'b1:1'b0;
        B26<=((x>398)&(x<416)&(y>243)&(y<271))?1'b1:1'b0;
        B27<=((x>416)&(x<434)&(y>235)&(y<253))?1'b1:1'b0;
        B28<=((x>398)&(x<416)&(y>289)&(y<307))?1'b1:1'b0;
        B29<=((x>416)&(x<434)&(y>307)&(y<325))?1'b1:1'b0;
        B30<=((x>434)&(x<452)&(y>325)&(y<343))?1'b1:1'b0;
        B31<=((x>434)&(x<452)&(y>217)&(y<235))?1'b1:1'b0;
        RED<= B20||B21||B22||B23;
        BLUE<= B24||B25||B26||B27||B28||B29||B30||B31;
    end
    else if(ko) // nel caso di partita vinta ko è alto e viene disegnata la scritta "ko"
        begin
        B20<=((x>238)&(x<258)&(y>248)&(y<352))?1'b1:1'b0;
        B21<=((x>258)&(x<278)&(y>291)&(y<309))?1'b1:1'b0;
        B22<=((x>278)&(x<298)&(y>273)&(y<291))?1'b1:1'b0;
        B23<=((x>298)&(x<318)&(y>255)&(y<273))?1'b1:1'b0;
        B24<=((x>318)&(x<338)&(y>237)&(y<255))?1'b1:1'b0;
        B25<=((x>278)&(x<298)&(y>309)&(y<327))?1'b1:1'b0;
        B26<=((x>298)&(x<318)&(y>327)&(y<345))?1'b1:1'b0;
        B27<=((x>318)&(x<338)&(y>345)&(y<363))?1'b1:1'b0;
        B28<=((x>350)&(x<370)&(y>248)&(y<312))?1'b1:1'b0;
        B29<=((x>370)&(x<434)&(y>228)&(y<248))?1'b1:1'b0;
        B31<=((x>434)&(x<454)&(y>248)&(y<312))?1'b1:1'b0;
        B30<=((x>370)&(x<434)&(y>312)&(y<332))?1'b1:1'b0;
        BLUE<=B20||B21||B22||B23||B24||B25||B26||B27;
        RED<= B28||B29||B30||B31;
        end
    else
                begin
                B20<=1'b0;
                B21<=1'b0;
                B22<=1'b0;
                B23<=1'b0;
                B24<=1'b0;
                B25<=1'b0;
                B26<=1'b0;
                B27<=1'b0;
                B28<=1'b0;
                B29<=1'b0;
                B31<=1'b0;
                B30<=1'b0;
                BLUE<=1'b0;
                RED<=1'b0;
                end
```

17) Come avviene il disegno delle schermate di vittoria o sconfitta

BZ, con Z che va da 20 a 31, rappresentano i mattoni utilizzati per effettuare le scritte “ok” e “ko”. Per disegnare i vari oggetti vengono forniti i wire relativi alle porte di

output BLUE e RED, che successivamente verranno collegate ai pin della board relativi rispettivamente ai colori blu e rosso.
L’ ultimo blocco begin-end evita la presenza di inferred latch.

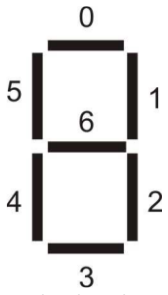




18) Schermate di partita vinta e di partita persa

Display a sette segmenti

Il numero di vite a disposizione del giocatore viene visualizzato sul display a sette segmenti fornito dalla board.
I pin dei sette segmenti sono attivi bassi, quindi applicando al pin un livello logico basso (“0”) il segmento si accende, viceversa applicando al pin un livello logico alto (“1”) il segmento si spegne.
Ogni segmento è identificato da un indice da 0 a 6.



19) Posizione ed indice di ogni segmento

Nell’ immagine seguente viene mostrato il blocco always presente in Arkanoid che permette di visualizzare il numero di vite sul display a sette segmenti.

```
always@(*)                          // visualizzazione delle vite sul display a sette segmenti
   case(lfdisplay)
      1: begin
            HEX0[0]=1'b1;HEX0[1]=1'b0;HEX0[2]=1'b0;HEX0[3]=1'b1;HEX0[4]=1'b1;HEX0[5]=1'b1;HEX0[6]=1'b1;
         end
      2: begin
            HEX0[0]=1'b0;HEX0[1]=1'b0;HEX0[2]=1'b1;HEX0[3]=1'b0;HEX0[4]=1'b0;HEX0[5]=1'b1;;HEX0[6]=1'b0;
         end
      3: begin
            HEX0[0]=1'b0;HEX0[1]=1'b0;HEX0[2]=1'b0;HEX0[3]=1'b0;HEX0[4]=1'b1;HEX0[5]=1'b1;;HEX0[6]=1'b0;
         end
      0: begin
            HEX0[0]=1'b0;HEX0[1]=1'b0;HEX0[2]=1'b0;HEX0[3]=1'b0;HEX0[4]=1'b0;HEX0[5]=1'b0;;HEX0[6]=1'b1;
         end
      default:
         begin
            HEX0[0]=1'b0;HEX0[1]=1'b0;HEX0[2]=1'b0;HEX0[3]=1'b0;HEX0[4]=1'b0;HEX0[5]=1'b0;;HEX0[6]=1'b0;
         end
   endcase
```

20) Visualizzazione del numero di vite sul display a sette segmenti

Viene fatto un case sul numero di vite a disposizione del giocatore (lfdisplay) ed in base al suo valore vengono accesi/spenti in maniera opportuna i sette segmenti.

Assegnamenti dei pin

Il modulo Arkanoid è la top level entity, quindi sono le sue porte che si interfacciano con i pin della board. Nell’immagine seguenti sono mostrati gli assegnamenti delle porte con i pin della board.

To	Assignment Name	Value
out RED	Location	PIN_A7
out VGA_GREEN	Location	PIN_A8
out BLUE	Location	PIN_A10
out VGA_HS_O	Location	PIN_A11
out VGA_RED	Location	PIN_B7
out GREEN	Location	PIN_B9
out VGA_BLUE	Location	PIN_B10
out VGA_VS_O	Location	PIN_B11
out HEX0[6]	Location	PIN_E2
out HEX0[5]	Location	PIN_F1
out HEX0[4]	Location	PIN_F2
out HEX0[3]	Location	PIN_H1
out HEX0[2]	Location	PIN_H2
out HEX0[1]	Location	PIN_J1
out HEX0[0]	Location	PIN_J2
in CLK	Location	PIN_L1
in SX	Location	PIN_R21
in DX	Location	PIN_R22
in RST_BTN	Location	PIN_T21
in start	Location	PIN_T22

21) Assegnamento dei pin

Il modulo gioco

Come già detto in precedenza, lo scopo di questo modulo è quello di calcolare le informazioni del gioco per poi comunicarle al modulo Arkanoid, il quale le gestirà in maniera opportuna per visualizzarle sul monitor.
Si è utilizzata una macchina a stati sincrona, nello specifico una macchina di Mealy; questo perché le uscite sono funzione degli stati e degli ingressi attuali.
Il modulo utilizza un flipflop per conservare i dati durante il passaggio da uno stato della macchina all' altro.

Porte

Le porte di input e di output del modulo sono le seguenti:

```
module gioco(
    i_ani_stb,                                  // clock dei pixel
    i_clk,                                      // clock
    i_rst,                                      // reset
    i_animate,                                  // alto quando vengono disegnati i pixel attivi (pixel dello schermo)
    i_dx,                                       // tasto destro
    i_sx,                                       // tasto sinistro
    i_start,                                    // tasto di start
    o_x1_pal, o_x2_pal,                         // lati della palla
    o_y1_pal, o_y2_pal,
    o_x1_nav, o_x2_nav,                         // lati della navicella
    o_y1_nav, o_y2_nav,
    bri1,bri2,bri3,bri4,bri5,bri6,bri7,bri8,bri9,bri10,// permettono di capire quali mattoni visualizzare e quali no
    ok, ko,                                     // permettono di capire se la partita è stata vinta o persa
    ff_life                                     // numero di vite
);
```

22) Porte del modulo gioco

Il significato delle porte è facilmente intuibile dai capitoli precedenti.

Flipflop

```
// il flipflop restituisce informazioni riguardanti quali e quanti mattoni sono attivi, le posizioni della palla e della navicella,
// le direzioni della palla, quante volte sono stati colpiti i mattoni
wire ff_bri1, ff_bri2, ff_bri3, ff_bri4,ff_bri5, ff_bri6, ff_bri7, ff_bri8,ff_bri9, ff_bri10;
wire [9:0] ff_x_nav,ff_y_nav,ff_x_pal,ff_y_pal;
wire ff_x_dirpal,ff_y_dirpal;
wire [3:0] ff_brick;
wire [1:0] ff_red1,ff_red2,ff_red3,ff_red4;
wire [1:0] ff_blue5,ff_blue6,ff_blue7;

reg [3:0] snext,sreg;
reg[9:0] x_nav;                          // coordinata x del centro della navicella
reg[9:0] y_nav;                          // coordinata y del centro della navicella
reg[9:0] x_pal;                          // coordinata x del centro della palla
reg[9:0] y_pal;                          // coordinata y del centro della palla
reg x_dir_pal;                           // se vale 1 palla a dx, vale 0 palla a sx
reg y_dir_pal;                           // se vale 1 la palla va in basso, se vale 0 la palla va in alto
reg [1:0] life;
reg [3:0] brick;
reg [1:0] red1,red2,red3,red4;
reg [1:0] blue5,blue6,blue7;
```

23) Reg e wire definiti all'interno del modulo

Il flipflop conserva le informazioni del gioco e le restituisce al modulo attraverso i wire definiti:

- **ff_briX**: se "1" il mattone è attivo, se "0" no. X va da 1 a 10;
- **ff_x_nav**, **ff_y_nav**: sono rispettivamente la coordinata x e la coordinata y della navicella;
- **ff_x_pal, ff_y_pal:** sono rispettivamente la coordinata x e la coordinata y della palla;
- **ff_x_dirpal, ff_y_dirpal:** indicano rispettivamente la direzione lungo x e la direzione lungo y della palla;
- **ff_brick:** indica il numero di brick che sono attivi;
- **ff_redX**: indica il numero di colpi che mancano al mattone X per venire eliminato. Viene usato per i mattoni rossi. X va da 1 a 4;
- **ff_blueX**: ha un significato analogo a ff_redX, solo che viene usato per i mattoni ciano. X va da 5 a 7.

Snext ed Sreg sono le variabili utilizzate per la transizione degli stati e per la transizione delle uscite della macchina a stati.
I reg definiti nell' immagine 23) vengono utilizzate come variabili del gioco e vengono campionati dal flip flop.

```
flipflop ff(                                    // chiamata al flipflop
    .clk(i_clk),
    .rst(i_rst),
    .i_xdirpal(x_dir_pal),
    .i_ydirpal(y_dir_pal),
    .i_xnav(x_nav),
    .i_ynav(y_nav),
    .i_xpal(x_pal),
    .i_ypal(y_pal),
    .i_bri1(bri1),
    .i_bri2(bri2),
    .i_bri3(bri3),
    .i_bri4(bri4),
    .i_bri5(bri5),
    .i_bri6(bri6),
    .i_bri7(bri7),
    .i_bri8(bri8),
    .i_bri9(bri9),
    .i_bri10(bri10),
    .i_life(life),
    .i_brick(brick),
    .i_red1(red1),
    .i_red2(red2),
    .i_red3(red3),
    .i_red4(red4),
    .i_blue5(blue5),
    .i_blue6(blue6),
    .i_blue7(blue7),
    .o_xdirpal(ff_x_dirpal),        // le uscite del flipflop si collegano ai wire creati appositamente
    .o_ydirpal(ff_y_dirpal),
    .o_xnav(ff_x_nav),
    .o_ynav(ff_y_nav),
    .o_xpal(ff_x_pal),
    .o_ypal(ff_y_pal),
    .o_bri1(ff_bri1),
    .o_bri2(ff_bri2),
    .o_bri3(ff_bri3),
   .o_bri4(ff_bri4),
   .o_bri5(ff_bri5),
   .o_bri6(ff_bri6),
   .o_bri7(ff_bri7),
   .o_bri8(ff_bri8),
   .o_bri9(ff_bri9),
   .o_bri10(ff_bri10),
   .o_life(ff_life),
   .o_brick(ff_brick),
   .o_red1(ff_red1),
   .o_red2(ff_red2),
   .o_red3(ff_red3),
   .o_red4(ff_red4),
   .o_blue5(ff_blue5),
   .o_blue6(ff_blue6),
   .o_blue7(ff_blue7)
);
```

24) Creazione istanza del flipflop

Viene riportato per completezza la descrizione del flipflop

```
module flipflop(clk,rst,i_xdirpal,i_ydirpal,i_xnav,i_ynav,i_xpal,i_ypal,
                i_bri1,i_bri2,i_bri3, i_bri4,i_bri5,i_bri6,i_bri7, i_bri8,
                i_bri9, i_bri10,o_bri1,o_bri2,o_bri3, o_bri4,o_bri5,o_bri6,
                o_bri7, o_bri8, o_bri9, o_bri10,i_life, o_life, i_brick, o_brick,
                i_red1, i_red2, i_red3, i_red4, i_blue5, i_blue6, i_blue7, o_red1,o_red2,o_red3,o_red4,
                o_blue5, o_blue6, o_blue7,o_xdirpal,o_ydirpal,o_xnav,o_ynav,o_xpal,o_ypal
);

input clk,rst;
input i_xdirpal, i_ydirpal;
input [9:0] i_xnav,i_ynav,i_xpal,i_ypal;
input i_bri1,i_bri2,i_bri3, i_bri4,i_bri5,i_bri6,i_bri7, i_bri8,i_bri9,i_bri10 ;
 //ingressi flipflop
output reg [9:0] o_xnav,o_ynav,o_xpal,o_ypal;
output reg  o_xdirpal,o_ydirpal;
output reg o_bri1,o_bri2,o_bri3, o_bri4,o_bri5,o_bri6,o_bri7, o_bri8, o_bri9, o_bri10;//uscite flipflop
output reg [1:0] o_life;
output reg [3:0] o_brick;
```

```
input [1:0]  i_life;
input [3:0]  i_brick;
input [1:0] i_red1, i_red2, i_red3, i_red4;
input[1:0] i_blue5, i_blue6, i_blue7;
output reg [1:0]  o_red1,o_red2,o_red3,o_red4;
output reg [1:0]   o_blue5, o_blue6, o_blue7;


parameter IX_NAV = 10'd 320;  //coordinata x iniziale del centro della navicella
parameter IY_NAV = 10'd 470;
parameter IX_PAL = 10'd 320;  //coordinata x iniziale del centro della palla
parameter IY_PAL = 10'd 450;
parameter VITE = 2'd 3;
parameter BLOCCHI = 4'd 10;
parameter RED = 2'd 3;
parameter BLUE = 2'd 2;
        if(rst) begin                       // se reset condizioni iniziali
                o_bri1<=1'b1;  o_bri2<=1'b1; o_bri3<=1'b1;
                o_bri4<=1'b1;  o_bri5<=1'b1; o_bri6<=1'b1;
                o_bri7<=1'b1;  o_bri8<=1'b1; o_bri9<=1'b1;
                o_bri10<=1'b1;
                o_xdirpal<=1'b0;
                o_xnav<=IX_NAV;
                o_xpal<=IX_PAL;
                o_ydirpal<=1'b0;
                o_ynav<=IY_NAV;
                o_ypal<=IY_PAL;
                o_life<=VITE;
                o_brick<=BLOCCHI;
                o_blue5<=BLUE;
                o_blue6<=BLUE;
                o_blue7<=BLUE;
                o_red1<=RED;
                o_red2<=RED;
                o_red3<=RED;
                o_red4<=RED;
            end
          else  begin                      // sennò campionamento
                   o_bri1<=i_bri1; o_bri2<=i_bri2; o_bri3<=i_bri3;
                   o_bri4<=i_bri4; o_bri5<=i_bri5; o_bri6<=i_bri6;
                   o_bri7<=i_bri7; o_bri8<=i_bri8; o_bri9<=i_bri9;
                   o_bri10<=i_bri10;
                   o_xdirpal<=i_xdirpal;
                   o_xnav<=i_xnav;
                   o_xpal<=i_xpal;
                   o_ydirpal<=i_ydirpal;
                   o_ynav<=i_ynav;
                   o_ypal<=i_ypal;
                   o_life<=i_life;
                   o_brick<=i_brick;
                   o_blue5<=i_blue5; o_blue6<=i_blue6; o_blue7<=i_blue7;
                   o_red1<=i_red1;
                   o_red2<=i_red2;
                   o_red3<=i_red3;
                   o_red4<=i_red4;
               end
      endmodule
```

Parametri

```
parameter H_SIZE_BRI = 10'd 32;
parameter L_SIZE_BRI = 10'd 10;
parameter H_SIZE_NAV = 10'd 32;
parameter L_SIZE_NAV = 10'd 3;
parameter IX_NAV = 10'd 320;
parameter IY_NAV = 10'd 470;
parameter D_WIDTH = 10'd 640;
parameter D_HEIGHT = 10'd 480;
parameter H_SIZE_PAL = 10'd 2;
parameter IX_PAL = 10'd 320;
parameter IY_PAL = 10'd 460;
parameter VITE = 2'd 3;
parameter BLOCCHI = 4'd 10;
parameter ROSSO = 2'd 3;
parameter BLU = 2'd 2;
```

25) Parametri presenti all'interno del modulo

All' interno del modulo vengono definiti dei parametri, alcuni rappresentano le condizioni iniziali del gioco ed alcuni vengono utilizzati per fare assegnazioni continue oppure per il calcolo di parametri di gioco.

- **H_SIZE_BRI**: rappresenta metà larghezza dei mattoni;
- **L_SIZE_BRI**: rappresenta metà altezza dei mattoni;
- **H_SIZE_NAV**: rappresenta metà larghezza della navicella;
- **L_SIZE_NAV**: rappresenta metà altezza della navicella;
- **IX_NAV**: rappresenta la coordinata x iniziale della navicella;
- **IY_NAV**: rappresenta la coordinata y iniziale della navicella;
- **D_WIDTH**: rappresenta la larghezza dello schermo;
- **D_HEIGHT**: rappresenta l'altezza dello schermo;
- **H_SIZE_PAL**: rappresenta metà lato della palla;
- **IX_PAL**: rappresenta la coordinata x iniziale della palla;
- **IY_PAL**: rappresenta la coordinata y iniziale della palla;
- **VITE**: rappresenta il numero delle vite a disposizione del giocatore all'inizio del gioco;
- **BLOCCHI**: rappresenta il numero di mattoni presenti all'inizio del gioco;
- **ROSSO**: rappresenta il numero di volte che i mattoni rossi devono essere colpiti per venire eliminati;
- **BLU**: rappresenta il numero di volte che i mattoni blu devono essere colpiti per venire eliminati.

Assegnamenti continui

Le coordinate x ed y dei lati di un oggetto (della navicella e della palla) sono dati dalla somma o sottrazione della coordinata x o y del centro dell'oggetto e di metà altezza o larghezza dell' oggetto.

```
assign o_x1_nav = x_nav - H_SIZE_NAV;  //coordinata x del lato sx della navicella
assign o_x2_nav = x_nav + H_SIZE_NAV;  //coordinata x del lato dx della navicella
assign o_y1_nav = y_nav -L_SIZE_NAV;   //coordinata y del lato alto della navicella
assign o_y2_nav = y_nav + L_SIZE_NAV;  //coordinata y del lato basso della navicella
assign o_x1_pal = x_pal - H_SIZE_PAL;  //coordinata x del lato sx della palla
assign o_x2_pal = x_pal + H_SIZE_PAL;  //coordinata x del lato dx della palla
assign o_y1_pal = y_pal -H_SIZE_PAL;   //coordinata y del lato alto della palla
assign o_y2_pal = y_pal + H_SIZE_PAL;  //coordinata y del lato basso della palla
```

26) Assegnamenti continui presenti nel modulo

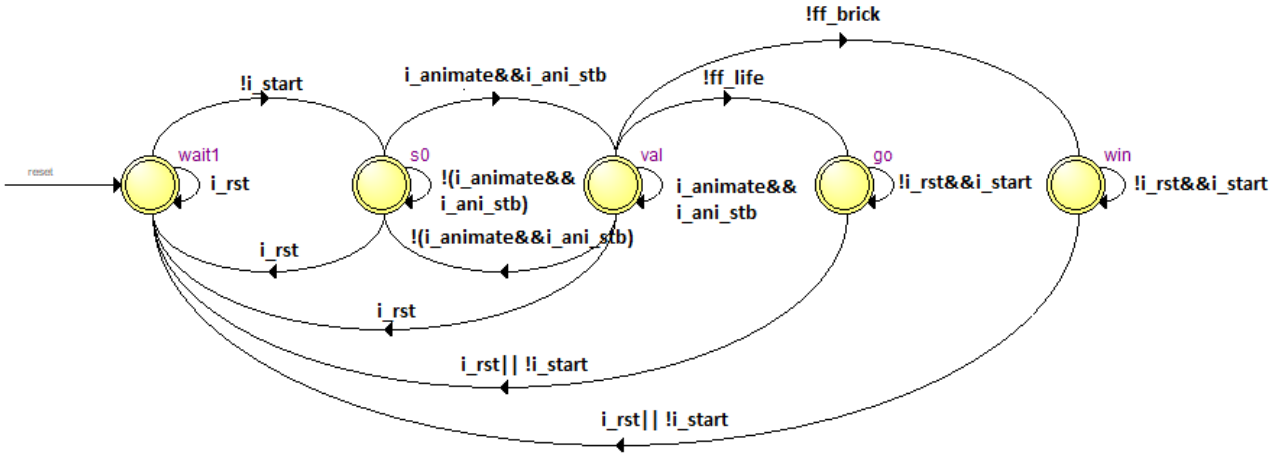
Macchina a stati

Come già detto in precedenza, in questo modulo viene definita una macchina a stati sincrona, modello di Mealy; questo perché i parametri di uscita dipendono sia dagli ingressi che dallo stato attuale.
Sono stati definiti cinque stati; la codifica degli stati nelle variabili di stato è la seguente:

```
parameter wait1=4'b 0000;
parameter s0=4'b 0001;
parameter val=4'b 0010;
parameter win=4'b 0100;
parameter go=4'b 1000;
```

27) Codifica degli stati

Si nota che è stata utilizzata una codifica "quasi one-hot", quindi per n stati (nel nostro caso n=5) vengono utilizzate n-1 variabili di stato. Nel nostro caso vi sono 11 stati che non sono stati definiti.
Nelle immagini seguenti vengono mostrati il diagramma di stato della macchina ed i blocchi always che gestiscono la transizione degli stati.




```
always@(posedge i_clk)
      if(i_rst)
         sreg<=wait1;
      else sreg<=snext;
```

```
always@(*)
      case(sreg)
         wait1:if(!i_start) snext=s0;
               else snext=wait1;

         s0:   if(i_animate&&i_ani_stb) snext=val;
               else snext=s0;
         val:
             if(!ff_life)
               snext=go;
             else if(!ff_brick)
               snext=win;
             else if(!(i_animate&&i_ani_stb))
                  snext=s0;
             else snext = val;
         win: if(!i_start) snext=wait1;
              else snext=win;
         go:  if(!i_start) snext=wait1;
              else snext=go;
      endcase
```

28) Diagramma di stato e blocchi always per la transizione degli stati

Per comprendere il diagramma di stato e la transizione degli stati è necessario conoscere lo scopo di ciascuno stato:

- **wait1**: è lo stato di partenza della macchina. In questo stato tutte le variabili del gioco sono al loro valore iniziale ed il gioco è fermo;
- **val**: è lo stato in cui vengono valutate tutte le variabili del gioco;
- **s0**: è lo stato di conservazione delle variabili del gioco durante il disegno dei pixel;
- **win**: è lo stato di partita vinta;
- **go**: è lo stato di partita persa.

Si nota che la macchina presenta un reset sincrono: da qualsiasi stato, premendo il tasto di reset, si torna nello stato iniziale wait1.
Si passa dallo stato wait1 allo stato s0 premendo il tasto i_start (ricordare che i pulsanti della scheda sono attivi bassi), in caso contrario la macchina rimane in wait1.
Una volta che la macchina è in s0, c'è il passaggio a val nel caso siano alti sia i_animate che i_ani_stb, cioè se sono stati disegnati tutti i pixel dello schermo e se il clock dei pixel è alto; in caso contrario la macchina rimane in s0.
Quando la macchina è in val:

- nel caso in cui ff_life sia uguale a "0", cioè se il numero di vite a disposizione del giocatore siano esaurite, allora lo stato successivo sarà go;
- invece nel caso in cui ff_brick sia pari a "0", cioè se sono stati eliminati tutti i mattoni, allora lo stato successivo sarà win;
- nel caso in cui i_animate oppure i_pix_stb siano bassi allora lo stato successivo sarà s0;
- se non si verifica nessuno dei casi precedenti la macchina rimane in val.

Quando lo stato della macchina è lo stato go, se viene premuto il pulsante di start la macchina torna nello stato wait1, altrimenti rimane in go.
Analogalmente allo stato g0, quando lo stato della macchina è lo stato win, se viene premuto il pulsante di start la macchina torna nello stato wait1, altrimenti rimane in win.

Transizione delle uscite

Per evitare inferred latch, è fondamentale nel blocco always per la transizione delle uscite, inizializzare le variabili.

```
always@(*)
    begin
        x_dir_pal=0;
        y_dir_pal=0;
        x_nav=IX_NAV;
        y_nav=IY_NAV;
        x_pal=IX_PAL;
        y_pal=IY_PAL;
        bri1=1; bri2=1; bri3=1; bri4=1; bri5=1;
        bri6=1; bri7=1; bri8=1; bri9=1; bri10=1;
        life=VITE;
        brick=BLOCCHI;
        ok=0;
        ko=0;
        red1=ROSSO;
        red2=ROSSO;
        red3=ROSSO;
        red4=ROSSO;
        blue5=BLU;
        blue6=BLU;
        blue7=BLU;
```

29) Inizializzazione delle variabili

Successivamente viene valutato in quale stato si trova la macchina andando ad effettuare un case su sreg.
Come già detto in precedenza, wait1 è lo stato iniziale della macchina, in cui tutti gli oggetti sono fermi e nella loro posizione iniziale.

```
case(sreg)
      wait1:
      begin
         // condizioni iniziali
         x_dir_pal=0;
         y_dir_pal=0;
         x_nav=IX_NAV;
         y_nav=IY_NAV;
         x_pal=IX_PAL;
         y_pal=IY_PAL;
         bri1=1; bri2=1; bri3=1; bri4=1; bri5=1;
         bri6=1; bri7=1; bri8=1; bri9=1; bri10=1;
         life=VITE;
         brick=BLOCCHI;
         red1=ROSSO;
         red2=ROSSO;
         red3=ROSSO;
         red4=ROSSO;
         blue5=BLU;
         blue6=BLU;
         blue7=BLU;
```

30) Stato wait1

Lo stato val è lo stato in cui vengono fatti calcoli riguardanti il gioco, come per esempio il movimento della palla, il movimento della navicella e le collisioni della palla.

Vengono inanzitutto sovrascritte le variabili con i valori provenienti dal flipflop.

```
val:
   begin
      // le varabili vengono sovrascritte
      x_dir_pal=ff_x_dirpal;
      y_dir_pal=ff_y_dirpal;
      x_nav=ff_x_nav;
      y_nav=ff_y_nav;
      x_pal=ff_x_pal;
      y_pal=ff_y_pal;
      bri1=ff_bri1; bri2=ff_bri2;bri3=ff_bri3;
      bri4=ff_bri4; bri5=ff_bri5;bri6=ff_bri6;
      bri7=ff_bri7; bri8=ff_bri8; bri9=ff_bri9;
      bri10=ff_bri10;
      life=ff_life;
      brick=ff_brick;
      red1=ff_red1;
      red2=ff_red2;
      red3=ff_red3;
      red4=ff_red4;
      blue5=ff_blue5;
      blue6=ff_blue6;
      blue7=ff_blue7;
```

31) Sovrascrittura variabili in val

Successivamente viene calcolata la posizione della palla in base alle direzioni lungo gli assi x ed y.

```
// movimento della palla
x_pal = (x_dir_pal) ? x_pal + 10'd 2 : x_pal - 10'd 2;
y_pal = (y_dir_pal) ? y_pal + 10'd 2 : y_pal - 10'd 2;
```

32) Movimento della palla

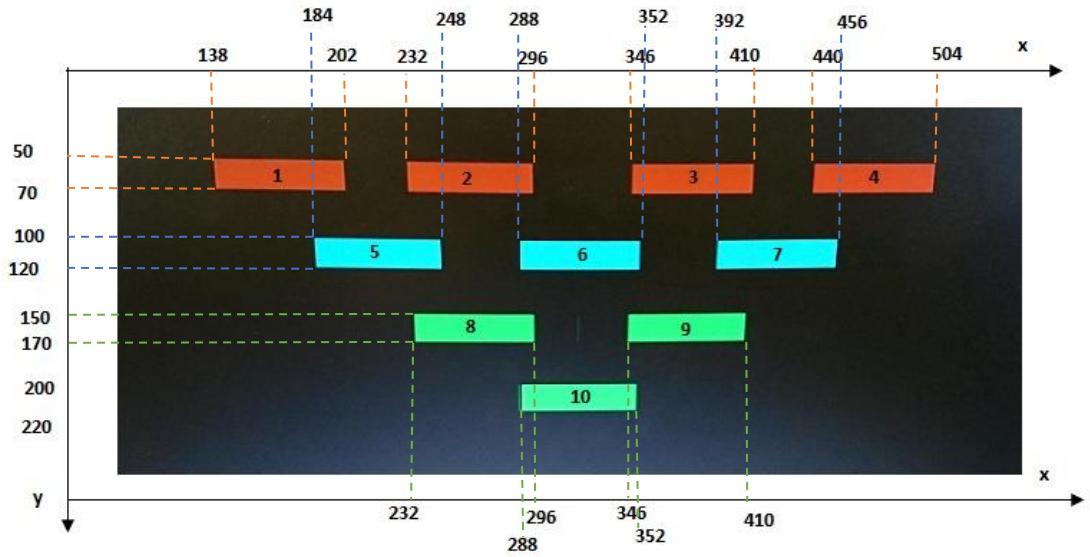
Si nota che se x_dir_pal vale “1”, allora la palla si muove verso destra, altrimenti se vale “0” la palla si muove verso sinistra.
Invece se y_dir_pal vale “1”, allora la palla si muove verso il basso, altrimenti se vale “0” la palla si muove verso l’alto.
Successivamente viene calcolata la posizione della navicella in base agli input i_dx e i_sx:

```
// movimento della navicella
if (i_sx==1 && i_dx==0)                          // se viene premuto il tasto destro
   begin                                         // NB: i tasti sono attivi bassi
   x_nav =x_nav+10'd 3;                          // la navicella si muove verso destra
      if (x_nav >= (D_WIDTH-H_SIZE_NAV-12'd 1))  // se la navicella è al margine dx dello schermo
         x_nav = (D_WIDTH-H_SIZE_NAV-10'd 1);    // allora rimane dove è
   end

else if(i_dx==1 && i_sx ==0)                     // se viene premuto il tasto sinistro
   begin
   x_nav = x_nav - 10'd 3;                       // la navicella si muove verso sinistra
      if (x_nav <= H_SIZE_NAV+12'd 1)            // se la navicella è al margine sx dello schermo
         x_nav = H_SIZE_NAV + 10'd 1;            // allora rimane dove è
   end
      else
         x_nav=x_nav;                            // nel caso in cui non viene premuto nessun tasto
```

33) Movimento navicella

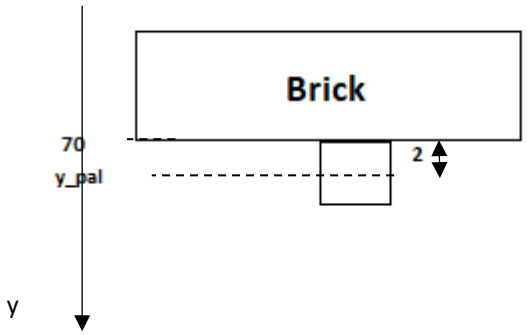
Ricordare che i pulsanti sono attivi bassi. Se viene premuto il pulsante destro o il tasto sinistro la navicella si muove rispettivamente verso destra o verso sinistra; altrimenti la navicella rimane ferma. Notare che quando viene effettuato lo spostamento verso destra o verso sinistra viene verificato se la navicella si trova ai margini dello schermo. La navicella viene vincolata a non muoversi oltre i margini dello schermo.
Successivamente vi è il calcolo delle varie variabili nel caso di collisione con i mattoni. Per semplicità verrà analizzata la collisione con il lato inferiore del mattone numero 1; analisi analoghe possono essere fatte con le collisioni con i lati degli altri mattoni (nel codice vi sono 40 descrizioni di questo tipo, 4 per ogni mattone).
La seguente immagine aiuta a comprendere le coordinate dei lati dei mattoni e mostra la numerazione dei mattoni.



34) Coordinate dei lati dei mattoni

```
if((y_pal <= 73)&&(y_pal >= 71))
begin
   if((x_pal >= 140)&&(x_pal <=202)&&bril) //se la palla colpisce brick1 da sotto
      begin
         y_dir_pal =1; //la palla va verso il basso
         red1=red1-2'd 1;
         if(!red1)
            begin
               brick = brick -4'd 1;
               bril= 1'd 0;
            end
      end
```

35) Collisione con il mattone 1



Premettendo che il mattone non sia stato eliminato, se la coordinata y della palla è compresa tra 71 e 73 e la coordinata x della palla è compresa fra le coordinate x dei lati sinistro e destro del mattone (nel caso in esame le coordinate x dei lati sono 140 e 202) allora vi è la collisione della palla con il lato inferiore del mattone. Le coordinate 71 e 73 sono state scelte in maniera opportuna, visto che la palla muovendosi si sposta di due pixel e tenendo conto che la distanza fra centro della palla ed un suo lato è pari a due. Quando avviene la collisione, la variabile riguardante la direzione della palla cambia in maniera opportuna; nel caso in esame visto che la palla colpisce il lato inferiore del mattone la sua direzione dopo la collisione dovrà essere verso il basso, e quindi y_dir_pal viene posto uguale a “1”. La variabile red1, che tiene conto del numero di volte che il mattone è stato colpito decrementa di uno; successivamente viene verificato se il mattone deve essere rimosso dallo schermo e se così fosse brick, il quale tiene conto del numero di mattoni presenti durante il gioco, decrementa di uno e bri1 viene posto uguale a “0” in maniera tale che il modulo Arkanoid non lo disegni.

```
if(x_pal <= H_SIZE_PAL+1)              //Se la palla è al margine sx dello schermo
   x_dir_pal=1;                      //Si sposta a dx
if(x_pal >= (D_WIDTH-H_SIZE_PAL-1))       //se la palla è al margine dx dello schermo
   x_dir_pal = 0;                    //si sposta a sx
if (y_pal <= H_SIZE_PAL + 1)     //se y è al margine in alto dello schermo
   y_dir_pal = 1;          //vai verso il basso
if (y_pal>=463)
   if ((x_pal >= x_nav-H_SIZE_NAV)&&(x_pal <= x_nav + H_SIZE_NAV))   //se la navicella colpisce la palla
      y_dir_pal = 0;                 //essa rimbalza
   else
      begin
         life=life-2'b01;
         x_pal = IX_PAL;
         y_pal = IY_PAL;
         x_nav = IX_NAV;
      end
```

36) Collisioni con i margini del monitor

Nel caso di collisione con i margini del monitor, gli argomenti degli if sono stati ricavati tenendo conto delle dimensioni della palla. Notare che se la coordinata y della palla supera una certa soglia possono esserci due eventi, in base alla coordinata x della palla "x_pal":

- se x_pal è compresa fra le coordinate x dei lati sinistro e destro della navicella, allora significa che la palla colpisce la navicella e quindi rimbalza; la direzione della palla quindi cambia e viene posto y_dir_pal uguale a "0";
- nel caso in cui invece x_pal non fosse compresa fra le coordinate dei lati destro e sinistro della navicella, allora significa che la navicella non è riuscita a colpire la palla, quindi il giocatore perde una vita e sia la navicella che la palla tornano nella loro posizione iniziale.

```
s0:
   begin
      x_dir_pal=ff_x_dirpal;
      y_dir_pal=ff_y_dirpal;
      x_nav=ff_x_nav;
      y_nav=ff_y_nav;
      x_pal=ff_x_pal;
      y_pal=ff_y_pal;
      bri1=ff_bri1;  bri2=ff_bri2;  bri3=ff_bri3;
      bri4=ff_bri4;  bri5=ff_bri5;  bri6=ff_bri6;
      bri7=ff_bri7;  bri8=ff_bri8;  bri9=ff_bri9;
      bri10=ff_bri10;
      life=ff_life;
      brick=ff_brick;
      red1=ff_red1;
      red2=ff_red2;
      red3=ff_red3;
      red4=ff_red4;
      blue5=ff_blue5;
      blue6=ff_blue6;
      blue7=ff_blue7;
   end
```

37) Descrizione dello Stato s0

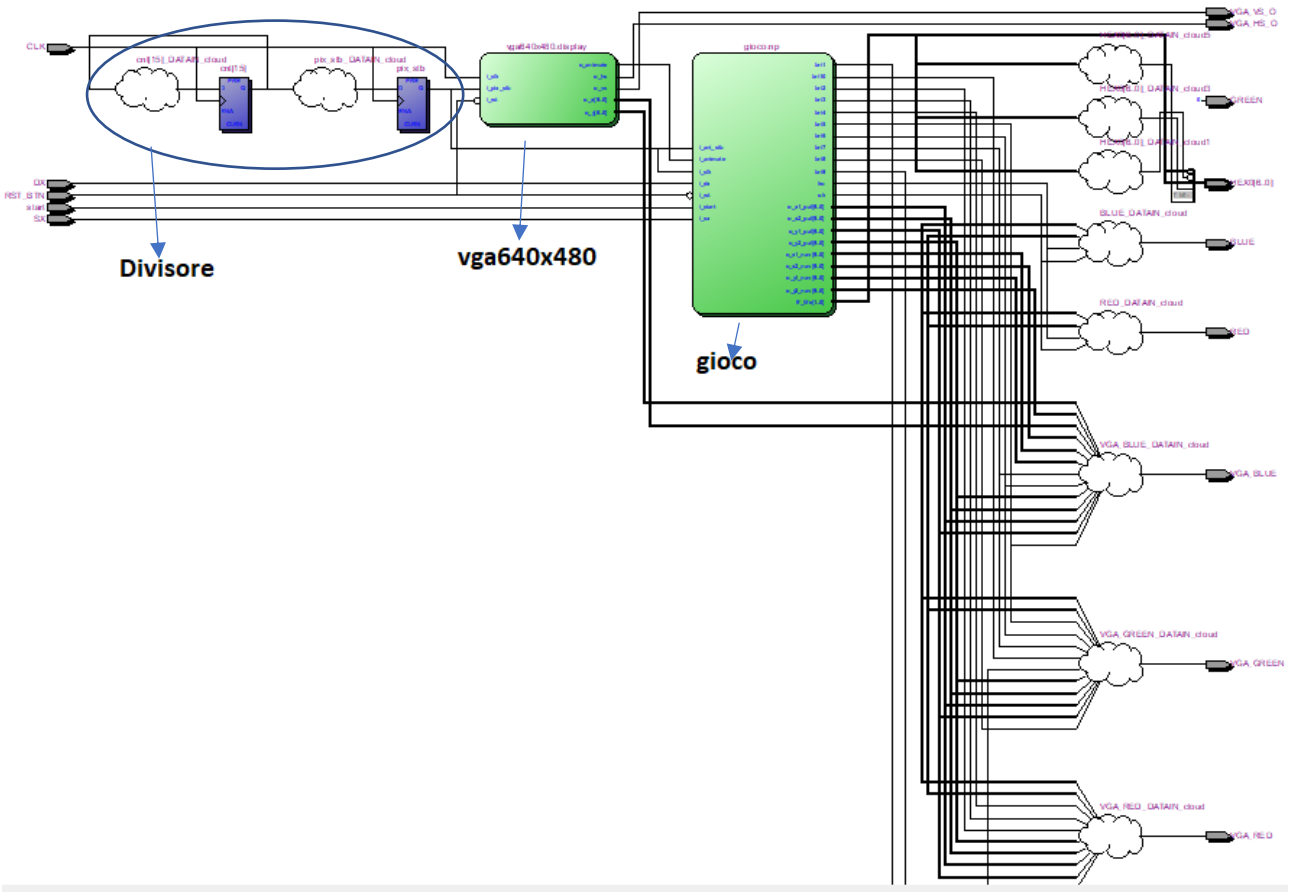
Nello stato s0 non viene fatto altro che sovrascrivere le variabili del gioco con i valori delle variabili provenienti dal flipflop.

```
win:begin
          ok=1;
     end
go:begin
          ko=1;
    end
default:
    begin
       x_dir_pal=0;
       y_dir_pal=0;
       x_nav=IX_NAV;
       y_nav=IY_NAV;
       x_pal=IX_PAL;
       y_pal=IY_PAL;
       bri1=1; bri2=1; bri3=1; bri4=1; bri5=1;
       bri6=1; bri7=1; bri8=1; bri9=1; bri10=1;
       life=VITE;
       brick=BLOCCHI;
       ok=0;
       ko=0;
       red1=ROSSO; red2=ROSSO; red3=ROSSO;
       red4=ROSSO;
       blue5=BLU; blue6=BLU; blue7=BLU;
```

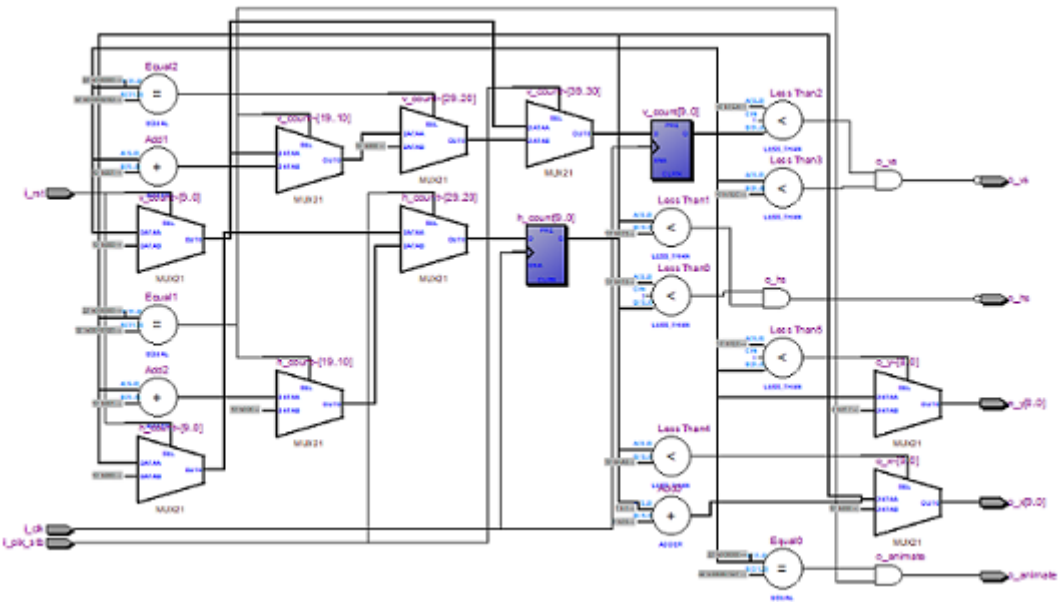
38) Descrizione degli stati win e go

Negli stati win e go vengono posti al valore “1” rispettivamente le variabili ok e ko, in maniera tale da poter disegnare attraverso il modulo Arkanoid le schermate di partita vinta o partita persa.

RTL viewer



Nell'immagine è mostrata la vista RTL del modulo Arkanoid. Si può notare la presenza del divisore di frequenza, del modulo vga640x480, del modulo gioco e delle varie porte di input e di output; le nuvolette rappresentano logiche combinatorie.



39) Vista RTL del modulo vga640x480

Per motivi di spazio non riportiamo la vista rtl del modulo gioco.

Risultati della compilazione

Riepilogo

Flow Summary	
Flow Status	Successful - Fri Dec 27 12:32:45 2019
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	Arkanoid
Top-level Entity Name	Arkanoid
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	1,107 / 18,752 (6 %)
Total combinational functions	1,106 / 18,752 (6 %)
Dedicated logic registers	87 / 18,752 (< 1 %)
Total registers	87
Total pins	20 / 315 (6 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)

40) Flow summary della compilazione

Da notare che il programma sfrutta il 6% dei logic element presenti sulla board e meno dell' 1% dei registri logici dedicati. Il numero di pin utilizzati è pari a 20.

Analisi e sintesi

Riepilogo delle risorse stimate

	Resource	Usage
1	Estimated Total logic elements	1,107
2		
3	Total combinational functions	1106
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	633
2	-- 3 input functions	263
3	-- <=2 input functions	210
5		
6	Logic elements by mode	
1	-- normal mode	943
2	-- arithmetic mode	163
7		
8	Total registers	87
1	-- Dedicated logic registers	87
2	-- I/O registers	0
9		
10	I/O pins	20
11	Embedded Multiplier 9-bit elements	0
12	Maximum fan-out node	gioco:np|sreg.s0
13	Maximum fan-out	71
14	Total fan-out	3977
15	Average fan-out	3.28

41) Riepilogo delle risorse utilizzate per l'analisi e sintesi

Facendo riferimento all' immagine 41), si nota che dei logic element (LE) stimati, 633 sfruttano le Look up table (LUT) per realizzare funzioni a quattro ingressi, 263 sfruttano le LUT per realizzare funzioni a 3 ingressi e 210 usano le LUT per realizzare funzioni a 2 o meno ingressi. Inoltre 943 LE sono stati utilizzati in normal mode e 163 in arithmetic mode. Nell' immagine inoltre è mostrato il nodo con il massimo fan-out ed il suo fan-out (pari a 71), il fan-out totale ed il fan-out medio.

Utilizzo delle risorse per entità

Compilation Hierarchy Node	LC Combinationals	LC Registers
|Arkanoid	1106 (259)	87 (2)
|gioco:np|	781 (731)	65 (5)
|flipflop:ff|	50 (50)	60 (60)
|vga640x480:display|	66 (66)	20 (20)

42) Numero di risorse utilizzate da ciascuna entità

I numeri fuori dalla parentesi indicano il numero totale delle risorse usate dall'entità e da tutte le entità sottostanti nella gerarchia. I numeri all'interno della parentesi indicano il numero di risorse utilizzate soltanto dalla specifica entità.

Macchina a stati

	Name	sreg.wait1	sreg.go	sreg.win	sreg.val	sreg.s0
1	sreg.wait1	0	0	0	0	0
2	sreg.s0	1	0	0	0	1
3	sreg.val	1	0	0	1	0
4	sreg.win	1	0	1	0	0
5	sreg.go	1	1	0	0	0

43) Codifica degli stati nelle variabili di stato

La codifica degli stati nelle variabili di stato è stata sintetizzata per essere di tipo “one hot”.

Ottimizzazioni

	Register name	Reason for Removal
1	gioco:np|flipflop:ff|o_ynav[3,5,9]	Merged with gioco:np|flipflop:ff|o_ynav[0]
2	gioco:np|flipflop:ff|o_ynav[2,4,6..8]	Merged with gioco:np|flipflop:ff|o_ynav[1]
3	gioco:np|flipflop:ff|o_ypal[0]	Merged with gioco:np|flipflop:ff|o_xpal[0]
4	gioco:np|flipflop:ff|o_ynav[0]	Stuck at GND due to stuck port data_in
5	gioco:np|flipflop:ff|o_ynav[1]	Stuck at VCC due to stuck port data_in
6	gioco:np|flipflop:ff|o_xpal[0]	Stuck at GND due to stuck port data_in
7	Total Number of Removed Registers = 12	

44) Variabili rimosse durante la sintesi

	Statistic	Value
1	Total registers	87
2	Number of registers using Synchronous Clear	31
3	Number of registers using Synchronous Load	3
4	Number of registers using Asynchronous Clear	0
5	Number of registers using Asynchronous Load	0
6	Number of registers using Clock Enable	22
7	Number of registers using Preset	0

45) Statistiche generali dei registri

Messaggi di warning

```
⚠ 20028 Parallel compilation is not licensed and has been disabled
⚠ 10034 Output port "GREEN" at Arkanoid.v(13) has no driver
⚠ 10230 Verilog HDL assignment warning at vga640x480.v(29): truncated value with size 32 to match size of t
⚠ 10230 Verilog HDL assignment warning at vga640x480.v(30): truncated value with size 32 to match size of t
⚠ 13024 Output pins are stuck at VCC or GND
  ⚠ 13410 Pin "GREEN" is stuck at GND
  ⚠ 13410 Pin "HEX0[1]" is stuck at GND
```

Il secondo messaggio di warning riguarda il fatto di aver definito la porta di output “GREEN” del modulo Arkanoid e di non averla utilizzata.
Sono presenti due warning riguardanti il fatto che due parametri che sono esprimibili a 32 bit sono stati troncati perché vengono utilizzati nella stessa funzione di parametri che sono espressi con un numero di bit minore. L’ultimo warning

riguarda il fatto che due pin vengono collegati a GND: GREEN viene collegato a GND perché, come detto in precedenza, non viene utilizzato, mentre HEX0[1] viene collegato a GND perché assume sempre il valore di “1” logico (ricordare che i segmenti del display a sette segmenti sono attivi bassi).

Fitter

Riepilogo delle risorse utilizzate

	Resource	Usage
1	Total logic elements	1,107 / 18,752 (6 %)
1	-- Combinational with no register	1020
2	-- Register only	1
3	-- Combinational with a register	86
2		
3	Logic element usage by number of LUT inputs	
1	-- 4 input functions	633
2	-- 3 input functions	263
3	-- <=2 input functions	210
4	-- Register only	1
4		
5	Logic elements by mode	
1	-- normal mode	943
2	-- arithmetic mode	163

7	Total registers*	87 / 19,649 (< 1 %)
1	-- Dedicated logic registers	87 / 18,752 (< 1 %)
2	-- I/O registers	0 / 897 (0 %)
8		
9	Total LABs: partially or completely used	77 / 1,172 (7 %)
10	Virtual pins	0
11	I/O pins	20 / 315 (6 %)
1	-- Clock pins	1 / 8 (13 %)
12		
13	Global signals	2
14	M4Ks	0 / 52 (0 %)
15	Total block memory bits	0 / 239,616 (0 %)
16	Total block memory implementation bits	0 / 239,616 (0 %)
17	Embedded Multiplier 9-bit elements	0 / 52 (0 %)
18	PLLs	0 / 4 (0 %)
19	Global clocks	2 / 16 (13 %)
20	JTAGs	0 / 1 (0 %)
21	ASMI blocks	0 / 1 (0 %)
22	CRC blocks	0 / 1 (0 %)
23	Average interconnect usage (total/H/V)	1% / 1% / 1%
24	Peak interconnect usage (total/H/V)	7% / 6% / 10%
25	Maximum fan-out	71
26	Highest non-global fan-out	71
27	Total fan-out	3980
28	Average fan-out	3.26

Si nota che le risorse stimate nell’ analisi e sintesi non si discostano dalle risorse utilizzate. Notare che il numero di Logic Array Block (LAB) utilizzati è pari a 77. I clock globali sono due, come ci aspettavamo. Il massimo fan-out non si discosta da quello stimato nell’analisi e sintesi ma il fan-out totale è leggermente aumentato ed il fan-out medio è leggermente diminuito.

Internal Connections	
-- Total Connections	3980
-- Registered Connections	698
Partition Interface	
-- Input Ports	5
-- Output Ports	15
-- Bidir Ports	0

46) Numero di connessioni interne ed interfaccia delle porte

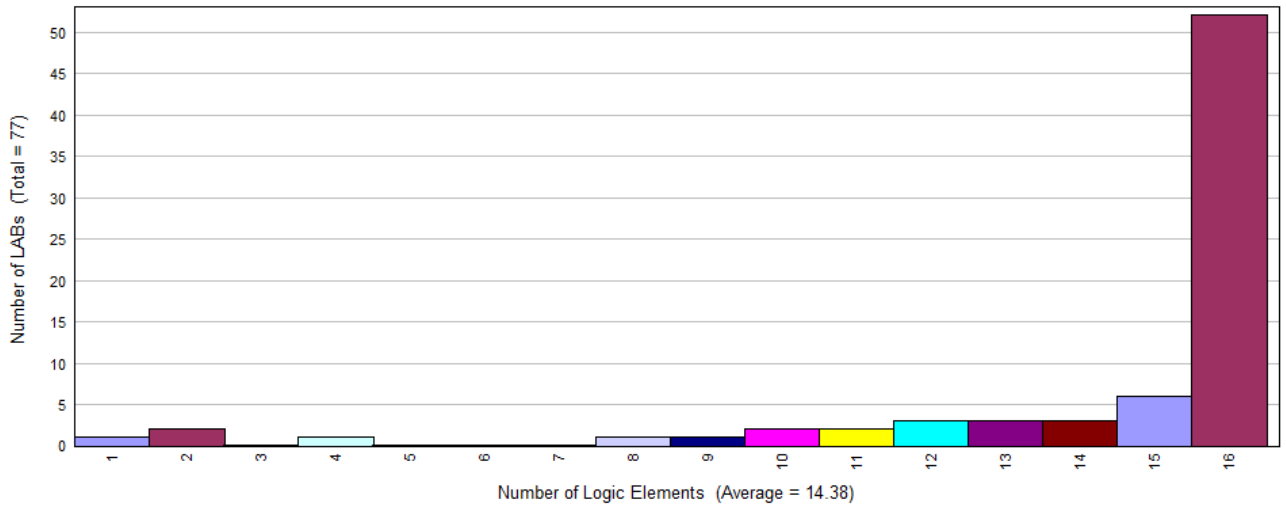
Utilizzo delle risorse per entità

Fitter Resource Utilization by Entity

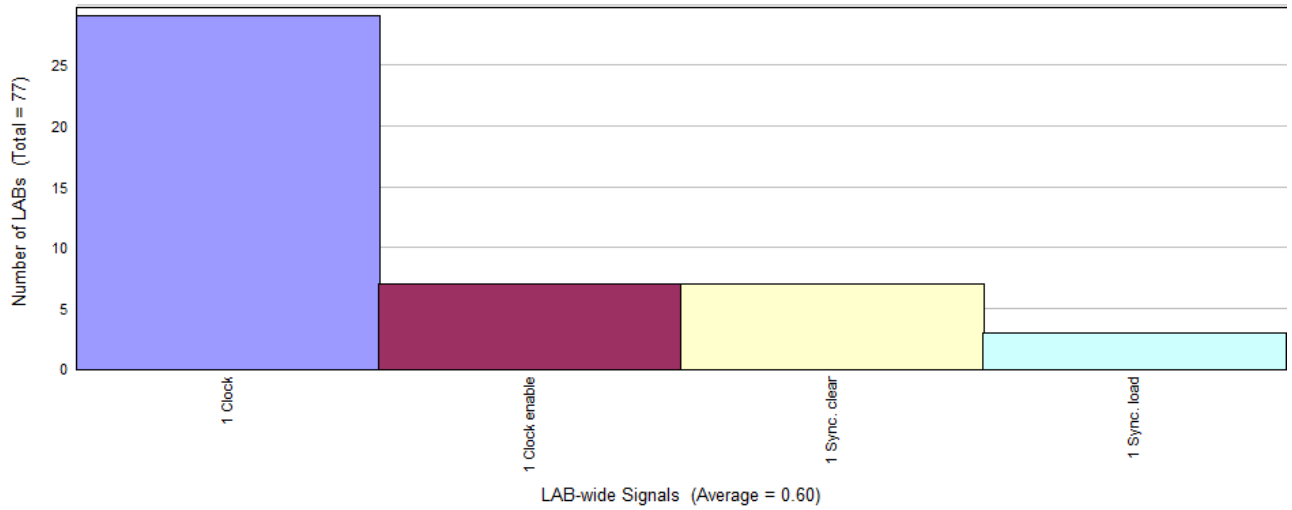
Compilation Hierarchy Node	Logic Cells	Dedicated Logic Registers
|Arkanoid	1107 (260)	87 (2)
|gioco:np|	781 (707)	65 (5)
|flipflop:ff|	75 (75)	60 (60)
|vga640x480:display|	66 (66)	20 (20)

Utilizzo delle risorse per entità si discosta dall' utilizzo stimato nell'analisi e sintesi.

Numero di LAB in funzione del numero di LE



Numero di LAB in funzione di funzionalità utilizzate



Timing Analyzer

L'analisi timing viene effettuata andando a considerare il modello slow ed il modello fast. Questi due modelli vanno a considerare se i segnali violano le condizioni di timing del circuito (e quindi possono non esserci dei corretti campionamenti dei segnali all'interno della struttura) nelle condizioni di corner di modellazione del circuito differenti, i quali possono far si che le reti combinatorie all' interno della struttura abbiano ritardi di propagazione massimo (corner slow) o minimo (corner fast).

Clock

	Clock Name	Type	Period	Frequency	Rise	Fall	Duty Cycle	Divide by	Multiply by
1	CLK	Base	20.000	50.0 MHz	0.000	10.000			
2	PIX	Generated	40.000	25.0 MHz	0.000	20.000		2	1

Per effettuare l'analisi timing abbiamo definito i due clock della struttura, il clock principale CLK da 50MHz ed il clock dei pixel da 25MHZ che viene generato da CLK.

Slow Model

Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	29.58 MHz	29.58 MHz	PIX	
2	203.83 MHz	203.83 MHz	CLK	

47) Frequenze massime dei clock

Nell' immagine vengono mostrate le frequenze massime che possono assumere i segnali di clock.

Percorsi critici

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	6.196	gioco:np|flipflop:ff|o_xpal[2]	gioco:np|flipflop:ff|o_brick[3]	PIX	PIX	40.000	-0.002	33.840

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	15.094	vga640x480:display|h_count[8]	vga640x480:display|v_count[3]	CLK	CLK	20.000	-0.005	4.939

Nell' immagine vengono mostrati i percorsi critici per quanto riguarda il tempo di setup.

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-0.384	vga640x480:display|v_count[4]	gioco:np|sreg.val	CLK	PIX	0.000	2.712	2.614
2	-0.353	vga640x480:display|v_count[4]	gioco:np|sreg.s0	CLK	PIX	0.000	2.712	2.645
3	-0.104	vga640x480:display|h_count[2]	gioco:np|sreg.val	CLK	PIX	0.000	2.706	2.888
4	-0.104	vga640x480:display|h_count[2]	gioco:np|sreg.s0	CLK	PIX	0.000	2.706	2.888
5	-0.102	pix_stb	gioco:np|sreg.val	PIX	PIX	0.000	2.416	2.877
6	-0.102	pix_stb	gioco:np|sreg.s0	PIX	PIX	0.000	2.416	2.877
7	-0.023	vga640x480:display|h_count[0]	gioco:np|sreg.val	CLK	PIX	0.000	2.706	2.969
8	-0.023	vga640x480:display|h_count[0]	gioco:np|sreg.s0	CLK	PIX	0.000	2.706	2.969

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	0.445	cnt[15]	cnt[15]	CLK	CLK	0.000	0.000	0.731

Nell'immagine vengono mostrati i percorsi critici per quanto riguarda il tempo di hold. Si nota che ci sono 8 percorsi in cui vengono violate le condizioni di timing per avere un corretto campionamento; ciò è dovuto al tempo di propagazione elevato del percorso critico. Una soluzione può essere quella di effettuare pipeling per diminuire il tempo di propagazione: viene divisa la rete combinatoria in più reti combinatorie e fra le varie reti combinatorie vengono messi dei flipflop.

Tempi di setup

Setup Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	RST_BTN	CLK	6.502	6.502	Rise	CLK
2	DX	CLK	12.899	12.899	Rise	PIX
3	RST_BTN	CLK	5.109	5.109	Rise	PIX
4	SX	CLK	12.436	12.436	Rise	PIX
5	start	CLK	3.254	3.254	Rise	PIX

Tempi di hold

Hold Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	RST_BTN	CLK	-5.566	-5.566	Rise	CLK
2	DX	CLK	-3.318	-3.318	Rise	PIX
3	RST_BTN	CLK	-1.427	-1.427	Rise	PIX
4	SX	CLK	-3.249	-3.249	Rise	PIX
5	start	CLK	-2.428	-2.428	Rise	PIX

Tempi clock to output

Clock to Output Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	BLUE	CLK	20.425	20.425	Rise	CLK
2	RED	CLK	19.299	19.299	Rise	CLK
3	VGA_BLUE	CLK	17.808	17.808	Rise	CLK
4	VGA_GREEN	CLK	18.376	18.376	Rise	CLK
5	VGA_HS_O	CLK	9.151	9.151	Rise	CLK
6	VGA_RED	CLK	18.024	18.024	Rise	CLK
7	VGA_VS_O	CLK	9.494	9.494	Rise	CLK
8	BLUE	CLK	16.531	16.531	Rise	PIX
9	⌄ HEX0[*]	CLK	12.660	12.660	Rise	PIX
1	HEX0[0]	CLK	12.660	12.660	Rise	PIX
2	HEX0[2]	CLK	12.374	12.374	Rise	PIX
3	HEX0[3]	CLK	12.470	12.470	Rise	PIX
4	HEX0[4]	CLK	11.035	11.035	Rise	PIX
5	HEX0[5]	CLK	12.148	12.148	Rise	PIX
6	HEX0[6]	CLK	11.035	11.035	Rise	PIX
10	RED	CLK	16.224	16.224	Rise	PIX
11	VGA_BLUE	CLK	32.973	32.973	Rise	PIX
12	VGA_GREEN	CLK	33.396	33.396	Rise	PIX
13	VGA_RED	CLK	33.189	33.189	Rise	PIX

Tempi clock to output minimi

Minimum Clock to Output Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	BLUE	CLK	10.479	10.479	Rise	CLK
2	RED	CLK	10.424	10.424	Rise	CLK
3	VGA_BLUE	CLK	9.601	9.601	Rise	CLK
4	VGA_GREEN	CLK	10.264	10.264	Rise	CLK
5	VGA_HS_O	CLK	8.572	8.572	Rise	CLK
6	VGA_RED	CLK	10.836	10.836	Rise	CLK
7	VGA_VS_O	CLK	8.370	8.370	Rise	CLK
8	BLUE	CLK	11.694	11.694	Rise	PIX
9	⌄ HEX0[*]	CLK	11.035	11.035	Rise	PIX
1	HEX0[0]	CLK	12.419	12.419	Rise	PIX
2	HEX0[2]	CLK	12.219	12.219	Rise	PIX
3	HEX0[3]	CLK	12.229	12.229	Rise	PIX
4	HEX0[4]	CLK	11.035	11.035	Rise	PIX
5	HEX0[5]	CLK	11.750	11.750	Rise	PIX
6	HEX0[6]	CLK	11.035	11.035	Rise	PIX
10	RED	CLK	11.820	11.820	Rise	PIX
11	VGA_BLUE	CLK	12.144	12.144	Rise	PIX
12	VGA_GREEN	CLK	12.807	12.807	Rise	PIX
13	VGA_RED	CLK	13.119	13.119	Rise	PIX

Ritardi di propagazione

Propagation Delay

	Input Port	Output Port	RR	RF	FR	FF
1	DX	VGA_BLUE	31.287	31.287	31.287	31.287
2	DX	VGA_GREEN	31.710	31.710	31.710	31.710
3	DX	VGA_RED	31.503	31.503	31.503	31.503
4	SX	VGA_BLUE	30.824	30.824	30.824	30.824
5	SX	VGA_GREEN	31.247	31.247	31.247	31.247
6	SX	VGA_RED	31.040	31.040	31.040	31.040

Ritardi di propagazione minimi

Minimum Propagation Delay

	Input Port	Output Port	RR	RF	FR	FF
1	DX	VGA_BLUE	17.990	17.990	17.990	17.990
2	DX	VGA_GREEN	19.703	19.703	19.703	19.703
3	DX	VGA_RED	19.496	19.496	19.496	19.496
4	SX	VGA_BLUE	17.921	17.921	17.921	17.921
5	SX	VGA_GREEN	19.634	19.634	19.634	19.634
6	SX	VGA_RED	19.427	19.427	19.427	19.427

Fast model

Percorsi critici

Fast Model Setup: 'CLK'

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	18.131	vga640x480:display|h_count[8]	vga640x480:display|v_count[3]	CLK	CLK	20.000	-0.005	1.896

Fast Model Setup: 'PIX'

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	20.092	vga640x480:display|v_count[6]	gioco:np|sreg.val	CLK	PIX	20.000	1.647	1.587

Nell' immagine vengono mostrati i percorsi critici per quanto riguarda il tempo di setup.

Fast Model Hold: 'PIX'

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-0.776	vga640x480:display|v_count[4]	gioco:np|sreg.s0	CLK	PIX	0.000	1.647	1.023

Fast Model Hold: 'CLK'

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	0.215	cnt[15]	cnt[15]	CLK	CLK	0.000	0.000	0.367

Nell' immagine vengono mostrati i percorsi critici per quanto riguarda il tempo di hold. Si nota che vengono violate le condizioni di timing per avere un corretto campionamento; nell' immagine non vengono mostrati ma ci sono in totale 42

percorsi in cui non vengono rispettate le condizioni di timing. Una delle possibili soluzioni è stata descritta precedentemente.

Tempi di setup

Setup Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	RST_BTN	CLK	2.990	2.990	Rise	CLK
2	DX	CLK	4.596	4.596	Rise	PIX
3	RST_BTN	CLK	1.836	1.836	Rise	PIX
4	SX	CLK	4.387	4.387	Rise	PIX
5	start	CLK	1.059	1.059	Rise	PIX

Tempi di hold

Hold Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	RST_BTN	CLK	-2.515	-2.515	Rise	CLK
2	DX	CLK	-1.032	-1.032	Rise	PIX
3	RST_BTN	CLK	-0.277	-0.277	Rise	PIX
4	SX	CLK	-0.966	-0.966	Rise	PIX
5	start	CLK	-0.715	-0.715	Rise	PIX

Tempi di clock to output

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	BLUE	CLK	8.783	8.783	Rise	CLK
2	RED	CLK	8.408	8.408	Rise	CLK
3	VGA_BLUE	CLK	7.830	7.830	Rise	CLK
4	VGA_GREEN	CLK	8.053	8.053	Rise	CLK
5	VGA_HS_O	CLK	4.599	4.599	Rise	CLK
6	VGA_RED	CLK	7.939	7.939	Rise	CLK
7	VGA_VS_O	CLK	4.682	4.682	Rise	CLK
8	BLUE	CLK	7.999	7.999	Rise	PIX
9	HEX0[*]	CLK	6.488	6.488	Rise	PIX
1	HEX0[0]	CLK	6.488	6.488	Rise	PIX
2	HEX0[2]	CLK	6.431	6.431	Rise	PIX
3	HEX0[3]	CLK	6.444	6.444	Rise	PIX
4	HEX0[4]	CLK	5.932	5.932	Rise	PIX
5	HEX0[5]	CLK	6.345	6.345	Rise	PIX
6	HEX0[6]	CLK	5.933	5.933	Rise	PIX
10	RED	CLK	7.926	7.926	Rise	PIX
11	VGA_BLUE	CLK	13.914	13.914	Rise	PIX
12	VGA_GREEN	CLK	14.137	14.137	Rise	PIX
13	VGA_RED	CLK	14.023	14.023	Rise	PIX

Tempi di clock to output minimi

Minimum Clock to Output Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Referenc
1	BLUE	CLK	5.072	5.072	Rise	CLK
2	RED	CLK	5.093	5.093	Rise	CLK
3	VGA_BLUE	CLK	4.774	4.774	Rise	CLK
4	VGA_GREEN	CLK	5.066	5.066	Rise	CLK
5	VGA_HS_O	CLK	4.388	4.388	Rise	CLK
6	VGA_RED	CLK	5.243	5.243	Rise	CLK
7	VGA_VS_O	CLK	4.292	4.292	Rise	CLK
8	BLUE	CLK	6.200	6.200	Rise	PIX
9	⌄ HEX0[*]	CLK	5.932	5.932	Rise	PIX
1	HEX0[0]	CLK	6.399	6.399	Rise	PIX
2	HEX0[2]	CLK	6.377	6.377	Rise	PIX
3	HEX0[3]	CLK	6.355	6.355	Rise	PIX
4	HEX0[4]	CLK	5.932	5.932	Rise	PIX
5	HEX0[5]	CLK	6.184	6.184	Rise	PIX
6	HEX0[6]	CLK	5.933	5.933	Rise	PIX
10	RED	CLK	6.263	6.263	Rise	PIX
11	VGA_BLUE	CLK	6.370	6.370	Rise	PIX
12	VGA_GREEN	CLK	6.662	6.662	Rise	PIX
13	VGA_RED	CLK	6.744	6.744	Rise	PIX

Ritardi di propagazione

Propagation Delay

	Input Port	Output Port	RR	RF	FR	FF
1	DX	VGA_BLUE	13.180	13.180	13.180	13.180
2	DX	VGA_GREEN	13.403	13.403	13.403	13.403
3	DX	VGA_RED	13.289	13.289	13.289	13.289
4	SX	VGA_BLUE	12.971	12.971	12.971	12.971
5	SX	VGA_GREEN	13.194	13.194	13.194	13.194
6	SX	VGA_RED	13.080	13.080	13.080	13.080

Ritardi di propagazione minimi

Minimum Propagation Delay

	Input Port	Output Port	RR	RF	FR	FF
1	DX	VGA_BLUE	8.296	8.296	8.296	8.296
2	DX	VGA_GREEN	9.075	9.075	9.075	9.075
3	DX	VGA_RED	8.961	8.961	8.961	8.961
4	SX	VGA_BLUE	8.230	8.230	8.230	8.230
5	SX	VGA_GREEN	9.009	9.009	9.009	9.009
6	SX	VGA_RED	8.895	8.895	8.895	8.895

Possibili sviluppi futuri

- Effettuare pipelining in maniera tale da ridurre i tempi di propagazione delle reti combinatorie più grandi così da non violare le regole di timing e non avere problemi di campionamento;
- Aggiungere potenziamenti;
- Aggiungere uno o più livelli a cui è possibile accedere una volta che sono stati eliminati tutti i mattoni. È possibile fare ciò grazie alla macchina a stati;
- Modificare la fisica delle collisioni della palla con i vari oggetti;
- Aggiungere un pulsante di “pausa”. È possibile fare ciò grazie alla macchina a stati;
- Aggiungere suoni grazie al codec audio;
- Migliorare le schermate di partita vinta o partita persa utilizzando immagini inserite nella memoria Sram fornita dalla board.