UNIVERSITÀ
DI TRENTO

Progetto per il corso di Reti Logiche

Space Evaders

Anno Accademico 2020-2021

Thomas Nonis, Lisa Santarossa, Tommaso Canova

Introduzione

Il progetto nasce con lo scopo di realizzare, su scheda FPGA, una versione alternativa del famoso videogame arcade *Space Invaders*.
A differenza del gioco originale, nel quale si raggiunge la vittoria eliminando tutti gli alieni invasori, l'utente vince quando riesce ad evitare l'ultima ondata di alieni che, durante la partita, cercheranno di catturare il giocatore. Se questo infatti entra in contatto con un alieno perde la partita.
In *Space Invaders* il giocatore deve salvare la Terra dall'invasione di una flotta aliena, mentre in *Space Evaders* l'utente dovrà evitare gli impatti con gli alieni per fuggire da una fatale collisione che lo porterebbe a diventare uno di loro.

Code overview

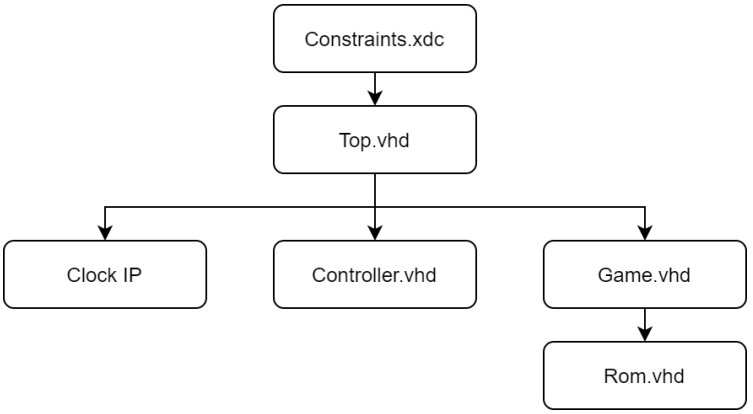


Figura 1 - Gerarchia file progetto

Top

L'architettura è composta da tre principali componenti: un generatore di clock, un controller VGA e un generatore d'immagine, il quale gestisce tutti gli elementi grafici e la logica del gioco.
Il modulo Top permette di effettuare i collegamenti tra questi blocchi, interfacciandosi con la scheda mediante le porte specificate nel file dei constraints.
In particolare collega il generatore di clock al clock di sistema e agli altri moduli, in modo da fornire il clock richiesto a questi ultimi. Al modulo di gestione della logica del gioco vengono poi collegati il controller VGA, le interfacce utente (pulsanti) e le uscite per la porta VGA.
Per il collegamento dei pulsanti, dopo aver effettuato test, si è optato per non utilizzare alcun tipo di debouncer, in quanto il suo effetto non sarebbe apprezzabile data la velocità dell'interfaccia.

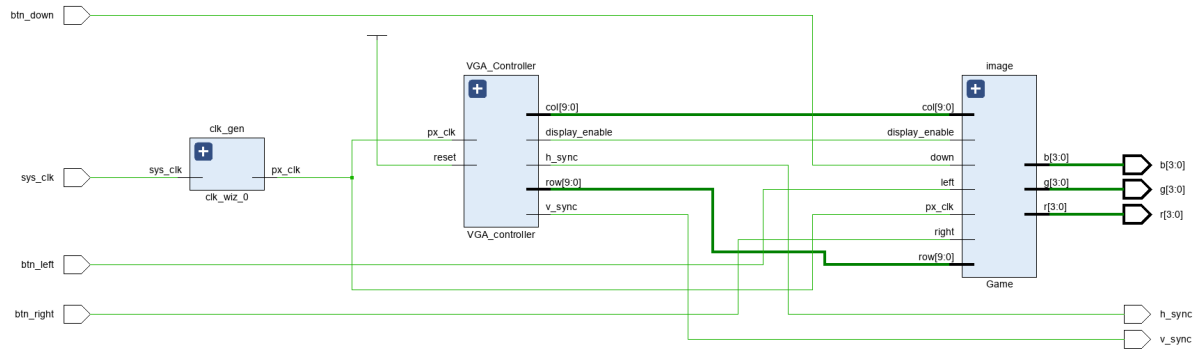


Figura 2 - Schematico componenti utilizzati ottenuto con Vivado

Clock Generator

In ingresso riceve il clock di sistema *sys_clk* e in uscita fornisce *px_clock*, segnale che governa il controller VGA e il generatore di immagini.
Le specifiche VGA (Fig. 3) per la risoluzione scelta richiedono un px_clock di 25.175 [MHz]. Per ottenerlo è stato utilizzato il clock wizard presente nell'IP Catalog di Vivado impostandolo alla frequenza richiesta. La frequenza effettiva ottenuta risulta pari a 25.17301 [MHz], che è sufficientemente simile a quella richiesta.

Controller VGA

Lo scheda si interfaccia con lo schermo tramite un *controller*, quest'ultimo riceve in ingresso:

- *px_clock* : clock che scandisce la generazione di ogni pixel;
- *reset* : segnale di reset asincrono

I segnali generati sono:

- *h_sync* e *v_sync* : segnali di sincronizzazione per lo schermo, che differiscono in base alle caratteristiche scelte;
- *col* e row*:* segnali che forniscono le coordinate del singolo pixel al generatore di immagini;
- *display_enable* : segnale di abilitazione per il generatore di immagini, che indica quando il pixel considerato è all'interno della porzione attiva dello schermo.

Nel progetto è stato implementata lo standard VGA in basic mode, che prevede un display di 640x480 pixel e 60 frame per secondo.
La tabella riporta le specifiche di diversi standard VGA, tra cui quello preso in considerazione:

			Horizontal (pixel clocks)				Vertical (rows)			
Resolution	Refresh Rate (Hz)	Pixel Clock (MHz)	Display	Front Porch	Sync Pulse	Back Porch	Display	Front Porch	Sync Pulse	Back Porch
640x350	70	25.175	640	16	96	48	350	37	2	60
640x350	85	31.5	640	32	64	96	350	32	3	60
640x400	70	25.175	640	16	96	48	400	12	2	35
640x400	85	31.5	640	32	64	96	400	1	3	41
640x480	60	25.175	640	16	96	48	480	10	2	33
640x480	73	31.5	640	24	40	128	480	9	2	29
640x480	75	31.5	640	16	64	120	480	1	3	16

Figura 3 - Tabella degli standard VGA

Nella figura a seguire viene indicato il comportamento dei due segnali *h_sync* e *v_sync*:

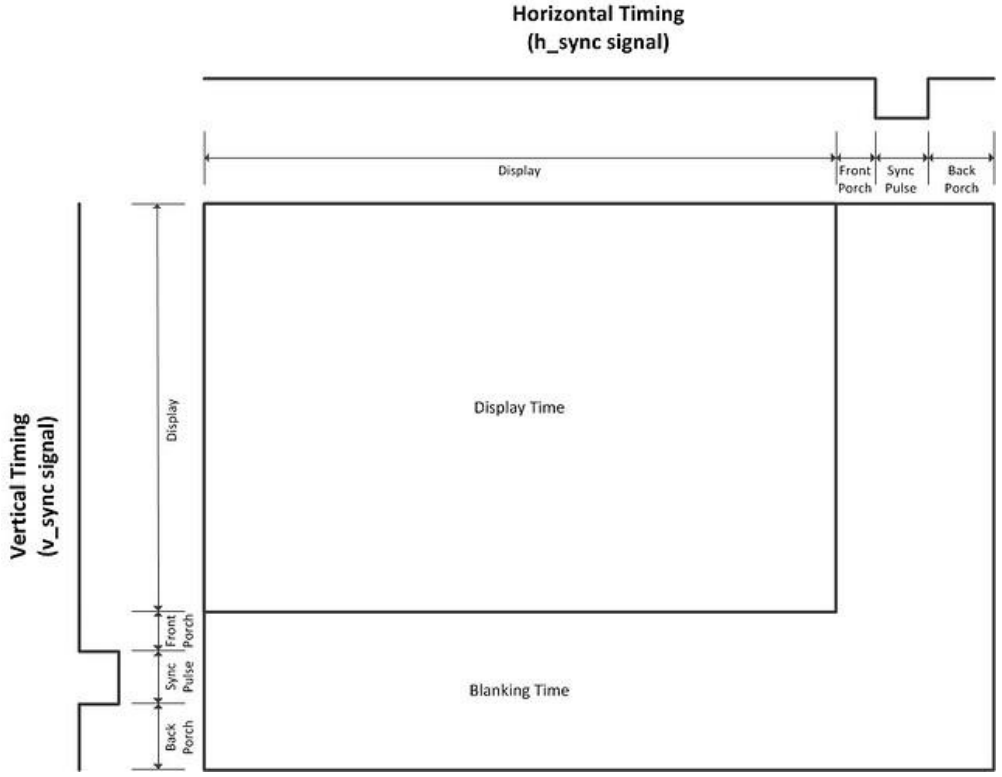


Figura 4 - Porzioni di schermo secondo lo standard VGA con annessi segnali di sincronizzazione

L'architettura del controller implementa due contatori, che incrementando ad ogni fronte del px_clock scandendo i segnali di timing verticale ed orizzontale.

ROM

Ogni elemento grafico 2D del progetto è rappresentato mediante un'architettura specifica nel file *Rom.vhd*.
Il componente Rom è dotato di due ingressi per selezionare la riga e la colonna, entrambi unsigned a 5 bit, con i quali si può selezionare il pixel di riferimento e ricavarne il contenuto rgb a 3 bit, questo valore viene poi connesso alla porta d'uscita *rgb.*
Per definire una matrice di 32x32 pixel sono stati definiti due tipi di dato: *row_type* e *rom_type*.
Il primo tipo di dato citato rappresenta un array di 32 *std_logic_vector* a 3 bit, necessari per rappresentare una riga dell'immagine. Il secondo tipo di dato invece è un array di 32 *row_type*, permettendo così di ottenere una matrice.
Ogni architettura ha un contenuto matriciale diverso, che si riesce ad ottenere mediante uno script Python.

Script Python per la conversione da *gif* a *rom_type*

Nel file *gif2rom.py*, scritto in Python 3, vengono accettati file immagini di tipo *.gif*, utilizzando la codifica a 3 bit e sfruttando la libreria grafica utilizzata (Pillow).
Data un'immagine in ingresso, viene prodotto in output un file di testo con lo stesso nome del file immagine. Lo script analizza l'immagine e sostituisce i valori dei colori secondo una codifica alternativa, riportata con una tabella di seguito.

Valore GIF	0	1	2	3	4	5	6	7
Valore RGB	000	001	100	101	010	110	111	011

Utilizzando il seguente comando da shell:
`python3 gif2rom.py ship.gif`
verrà prodotto il seguente file *ship_rom.txt*. Il contenuto verrà inserito nel file *Rom.vhd*, in cui viene definito il *content* dell'oggetto associato.

```
architecture Ship of Rom is

    type row_type is array (0 to 31) of std_logic_vector(2 downto 0);
    type rom_type is array (0 to 31) of row_type;

    constant content : rom_type :=
    (
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "111", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "111", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "111", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "111", "000", "111", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000", "000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000", "000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "111", "000", "000", "000", "111", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000", "000", "000", "000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "111", "000", "000", "000", "000", "000", "111", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "111", "000", "000", "000", "000", "000", "000", "000", "111", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "111", "111", "000", "000", "000", "000", "000", "111", "000", "000", "000", "000", "000", "111", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "111", "000", "000", "000", "111", "111", "000", "000", "000", "000", "000", "111", "000", "111", "000", "000", "000", "000", "000", "111", "111", "000", "000", "000", "111", "000", "000", "000", "000"),
        ("000", "000", "111", "111", "111", "000", "111", "111", "000", "000", "000", "000", "000", "111", "000", "000", "000", "111", "000", "000", "000", "000", "000", "111", "111", "000", "111", "111", "111", "000", "000", "000"),
        ("000", "111", "111", "000", "111", "111", "111", "000", "000", "000", "000", "000", "111", "000", "000", "000", "000", "000", "111", "000", "000", "000", "000", "000", "111", "111", "111", "000", "111", "111", "000", "000"),
        ("000", "111", "000", "000", "000", "111", "000", "000", "000", "000", "000", "000", "111", "000", "111", "000", "111", "000", "111", "000", "000", "000", "000", "000", "000", "111", "000", "000", "000", "111", "000", "000"),
        ("000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000"),
        ("000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000", "000", "000", "000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000"),
        ("000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000"),
        ("000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000", "000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000"),
        ("000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000"),
        ("000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000", "000", "111", "000", "111", "000", "000", "000", "111", "000", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000"),
        ("000", "111", "000", "000", "000", "000", "000", "000", "000", "111", "111", "111", "000", "000", "000", "000", "000", "000", "000", "111", "111", "111", "000", "000", "000", "000", "000", "000", "000", "111", "000", "000"),
        ("000", "111", "111", "000", "000", "000", "000", "000", "111", "111", "000", "111", "111", "000", "000", "000", "000", "000", "111", "111", "000", "111", "111", "000", "000", "000", "000", "000", "111", "111", "000", "000"),
        ("000", "000", "111", "111", "000", "000", "000", "111", "111", "000", "000", "000", "111", "111", "000", "000", "000", "111", "111", "000", "000", "000", "111", "111", "000", "000", "000", "111", "111", "000", "000", "000"),
        ("000", "000", "000", "111", "111", "000", "111", "111", "000", "000", "000", "000", "000", "111", "111", "111", "111", "111", "000", "000", "000", "000", "000", "111", "111", "000", "111", "111", "000", "000", "000", "000"),
        ("000", "000", "000", "111", "000", "111", "000", "111", "000", "000", "000", "000", "000", "000", "111", "111", "111", "000", "000", "000", "000", "000", "000", "111", "000", "111", "000", "111", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000"),
        ("000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000", "000")
    );

begin

    rgb <= content(to_integer(row))(to_integer(col));

end Ship;
```

Figura 5 - Estratto di codice del associato alla navicella controllata dall'utente

Logica di gioco

La logica del gioco è gestita mediante il file *Game.vhd*.
L'interfaccia di tale modulo consiste in due parametri generici che definiscono le risoluzioni orizzontale e verticale scelte e, delle seguenti porte che permettono l'interfacciamento con gli altri componenti:

- *px_clk*: clock per il singolo pixel;
- *display_enable*: flag che permette di capire se *scrivere* su schermo oppure no;
- *row* e *col*: indicano il pixel considerato per ogni singolo istante;
- *down*, *left*, *right*: pulsanti fisici sulla scheda;
- r, *g*, *b*: segnali di uscita per la terna di colori da applicare ai pixel.

Gestione progressione gioco

La progressione del gioco avviene all'interno del processo *game_proc*. Per ogni frame viene abilitata la lettura dell'input da utente, che permette di muovere la navicella e di resettare il gioco. Ogni 16 frame viene incrementata di un valore costante *ENEMY_STEP* la posizione dell'orda di alieni, che si avvicina sempre più alla base dello schermo.

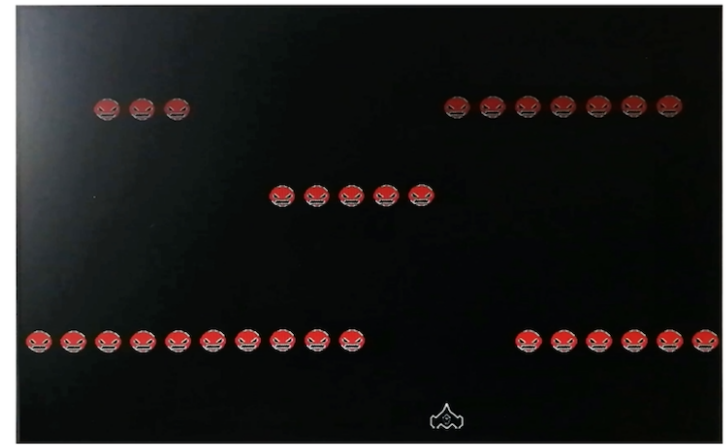
La vittoria avviene quando vengono attraversate tutte le orde senza toccare alcun alieno. In caso di tocco il gioco entra in gameover. Per ricominciare è sufficiente resettare il gioco premendo il tasto *down.*

Render engine

Gli elementi vengono mostrati su schermo mediante flag.
Per ogni pixel viene impostato un flag in base a quale elemento deve essere disegnato sullo stesso. In base al flag viene poi impostato un colore tramite costante oppure tramite lettura da ROM.
Il valore RGB che viene riportato in uscita è quello del pixel corrente se il controller si trova all'interno della zona del display, in caso contrario viene riportato il colore nero (RGB="000").

Immagini

Le tre situazioni di gioco **running**, **game over** e **winning** rispettivamente:



Conclusioni

Il progetto ha permesso al gruppo di affrontare nuove tematiche quali la gestione dei timing di un'interfaccia video come la VGA, la realizzazione di una versione elementare di una GPU e del problem-solving all'interno dell'ecosistema VHDL/Vivado.

Durante lo svolgimento del progetto sono state riscontrate numerose problematiche, molte delle quali risolte mediante testing e reiterazioni del codice.
In particolare è stato appurato che l'utilizzo di oggetti di tipo numerico (integer e natural) favorisce comportamenti inaspettati nella logica di funzionamento, dovuti probabilmente alla sintesi del circuito di tali tipi di dato, motivo per cui non sono stati scelti per la scrittura del codice.

Per concludere si propongono alcuni miglioramenti possibili in implementazioni future:

- Generazione pseudo-casuale del posizionamento e del numero di orde degli alieni.
- Miglioramento del codice per la gestione delle orde, evitando l'*hard-coding.*