UNIVERSITA' DI PISA

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Elettronica.

Implementazione gioco Arcade basato su *Space Invaders*

Materia
Sistemi Embedded

Studente
Iacopo Bonfiglio
Roberto Caocci
Antonio Rasulo

Docente
F. Baronti

Anno Accademico
2019-2020

Sommario

1 Introduzione ed obbiettivi

L'obiettivo del progetto è quello di progettare e realizzare un sistema embedded per la visualizzazione ed il controllo di un videogioco *Arcade*, sfruttando una scheda di sviluppo dell'Altera chiamata DE10-Lite.

Il progetto prevede quindi la creazione di un computer mediante il tool Qsys, nonché la configurazione del processore Nios II presente all'interno della FPGA MAX10 e dei relativi componenti utili al corretto funzionamento dello stesso.

Il videogioco realizzato è una versione del popolare videogioco *Arcade* "Space Invaders" il quale viene visualizzato su un monitor con interfaccia VGA.

Il controllo del videogioco avviene attraverso l'utilizzo dei *pushbutton* e dell'accelerometro on-board.

Gli obbiettivi che ci siamo prefissati e che abbiamo raggiunto sono i seguenti:

- Generazione di un output video VGA a risoluzione 640x480;
- Controllo della navicella tramite accelerometro;
- Presenza di "invasori" che percorrono un percorso fino ad arrivare a collidere con la navicella, causando il "Game Over";
- Possibilità di sparare con la navicella e distruggere gli "invasori";
- Schermate di "Game Over";
- Possibilità da parte degli "invasori" di sparare per danneggiare la navicella
- Visualizzazione del numero di vite della navicella sul monitor;
- Cambiamento del colore della navicella in base al numero di colpi subiti.

2 Hardware utilizzato

Per la realizzazione del progetto ci si è avvalsi dei seguenti componenti:

- Scheda programmabile *Terasic DE10-Lite*
- Personal Computer
- Monitor VGA

2.1 DE10-Lite

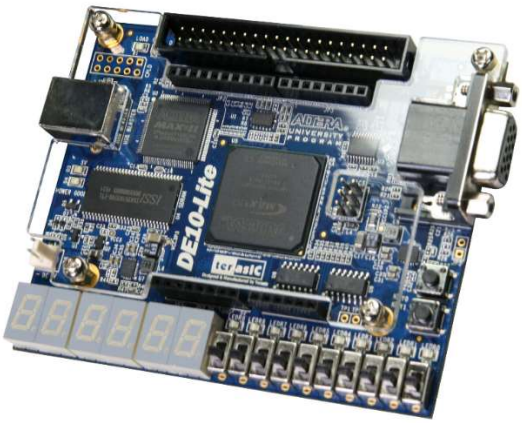


Fig. 2.1.1 – DE10-Lite Board.

Questa board presenta una piattaforma hardware robusta costruita attorno ad Altera MAX 10 FPGA. Include hardware on-board come un accelerometro a 3 assi, SDRAM ed un output VGA. Con FPGA MAX 10 è possibile ottenere applicazioni a basso costo e consumo con elevate performance.

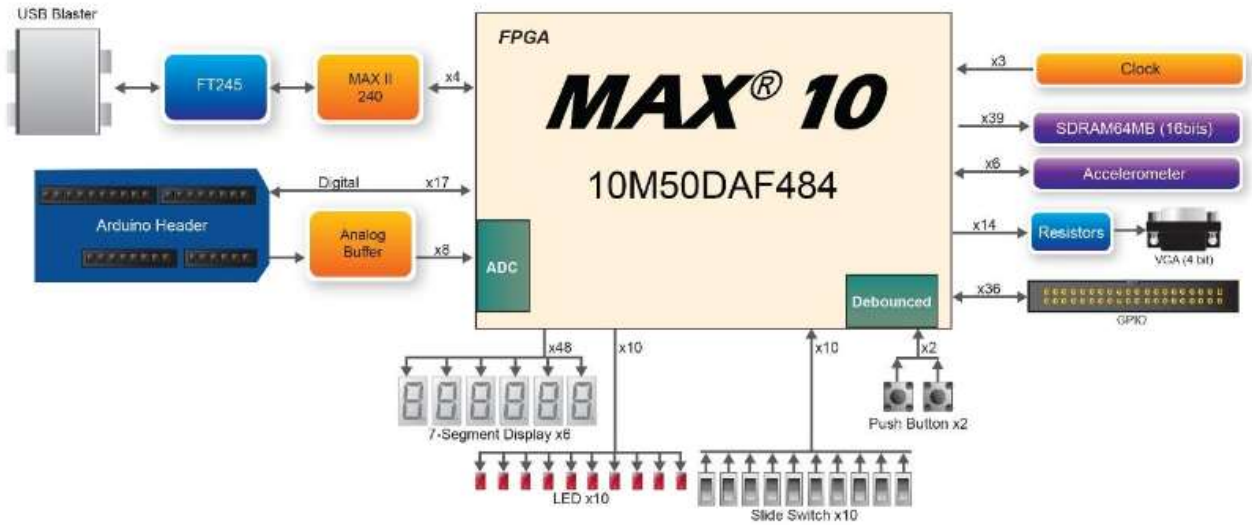


Fig. 2.1.2 - Schema a blocchi DE10-Lite.

3 Video

3.1 Background

Il video viene prodotto mostrando frame (detti anche immagini) in rapida successione. Tipicamente i frame vengono mostrati tra 30÷120 volte per secondo (cioè tra 30÷120 Hz). Un frame è un array di pixel a due dimensioni. Con risoluzione di un frame si definisce il numero di pixel sugli assi *x* ed *y*.

Un controller dedicato (*DMA Controller*) legge continuamente i dati dal pixel buffer, utilizzando la sua interfaccia *Avalon-MM Master*, ed invia questi frame video attraverso la sua interfaccia Avalon-ST.

Il controller del *Pixel Buffer* presenta un interfaccia programmabile sotto forma di set di registri.

Address	Register Name	R/W	Bit Description								
			31...24	23...16	15...12	11...8	7...6	5...3	2	1	0
0xFF203020	Buffer	R	Buffer's start address								
0xFF203024	BackBuffer	R/W	Back buffer's start address								
0xFF203028	Resolution	R	Y		X						
0xFF20302C	Status	R	m	n	*(1)*	BS	SB	*(1)*	EN	A	S
	Control	W	*(1)*						EN	*(1)*	

Fig. 3.1 – Registri *Pixel Buffer Controller*.

Nel concetto di double-buffering sono coinvolti due pixel buffer.

I registri *Buffer* e *BackBuffer* contengono l'indirizzo di partenza di questi due pixel buffer. Il registro *Buffer* contiene l'indirizzo del *Pixel Buffer* che viene mostrato a schermo, cioè in altre parole, il controller del *Pixel Buffer* va a leggere il buffer il cui indirizzo di partenza è indicato nel registro *Buffer*.

L'operazione di swap scambia il contenuto dei registri *Buffer* e *BackBuffer* e non avviene in maniera immediata, ma solo nel momento in cui il frame è stato completamente disegnato a schermo ovvero ogni 1/60s.

3.2 Video-out port

Il computer include una porta per l'output video connessa al controller VGA on-board, la quale può essere connessa ad un monitor VGA standard. Questa porta supporta una risoluzione dello schermo 640x480. L'immagine che viene mostrata a video deriva da due sorgenti: *Pixel Buffer* e *Character Buffer*.

3.3 Pixel Buffer

Il pixel buffer contiene i dati per ogni pixel che verrà mostrato a schermo. Il *Pixel Buffer* fornisce una risoluzione dell'immagine 160x120 pixel, con le coordinate (0,0) ad indicare l'angolo in alto a sinistra dell'immagine. Dal momento che la video-out port supporta una risoluzione dello schermo di 640x480, ogni valore del pixel del *Pixel Buffer* viene replicato 4 volte sia lungo *x* che lungo *y* quando l'immagine viene mostrata a schermo.

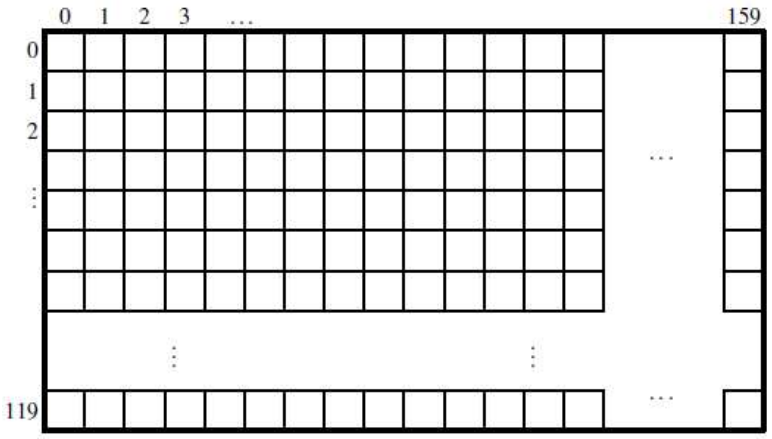


Fig. 3.3.1 - Coordinate del pixel buffer

Il colore di ogni pixel è rappresentato da una half-word a 16-bit, con cinque bit per le componenti blu e rossa e sei bit per la componente verde.

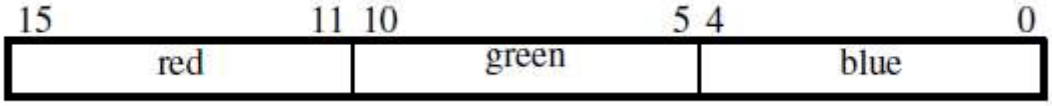


Fig. 3.3.2 - Colore dei pixel.

L'indirizzo dei pixel è una combinazione di un indirizzo base ed un offset *x-y*. L'indirizzo base del *Pixel Buffer* è 0x08000000, che corrisponde all'indirizzo di partenza della *On-chip Memory*.

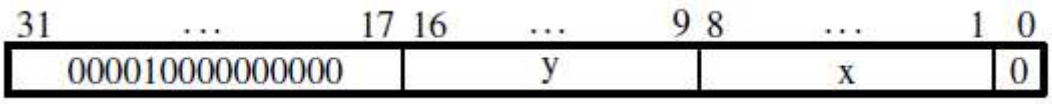


Fig. 3.3.3 – Indirizzamento dei Pixel.

Un *Pixel Buffer Controller* dedicato legge continuamente i pixel in maniera sequenziale nella memoria per poterli mostrare a schermo. Il dato dei pixel può essere cambiato anche quando l'immagine è mostrata, tuttavia è possibile evitare ciò utilizzando il concetto di double-buffering.

3.4 Character Buffer

Il *Character Buffer*, così come il pixel buffer, si trova su una *On-chip Memory* nel FPGA sulla DE10-Lite board.

Il buffer fornisce una risoluzione di 80x60 caratteri, dove ogni carattere occupa un blocco 8x8 di pixel sullo schermo. I caratteri sono immagazzinati in ognuna delle locazioni del buffer usando il loro codice ASCII; quando questi codici devono essere utilizzati per mostrare i caratteri a schermo, il character buffer genera automaticamente il pattern di pixel corrispondente per ogni carattere.

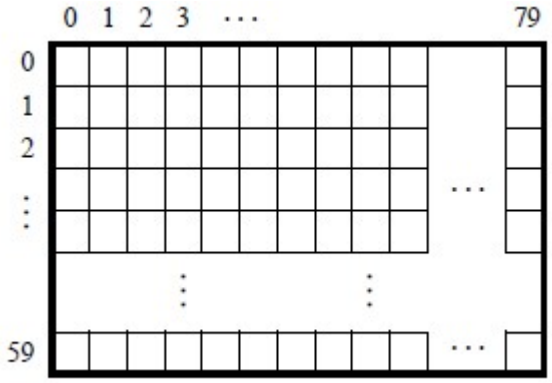


Fig. 3.4.1 – *Character buffer*

I caratteri sono indirizzati in memoria usando la combinazione di un indirizzo base, che ha valore 0x09000000 ed un offset *x*, *y*.

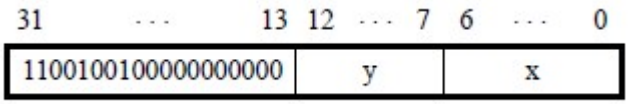


Fig. 3.4.2 – Indirizzamento *Character Buffer*.

3.5 Video Stream Packet format

I core trasferiscono i frame usando interfacce *Avalon Streaming*. Ogni pacchetto nello stream rappresenta un frame di dati video. I frame video vengono trasferiti un pixel per volta partendo dalle righe di ordine minore (cioè partendo dai pixel per y=0). Il primo pixel, cioè quello in alto a sinistra nel frame, viene segnalato dal bit *start-of-packet* nell'interfaccia *Avalon Streaming*. L'ultimo pixel, quello in basso a destra del frame, viene segnato dal bit *end-of-packet* nell'interfaccia *Avalon Streaming*.

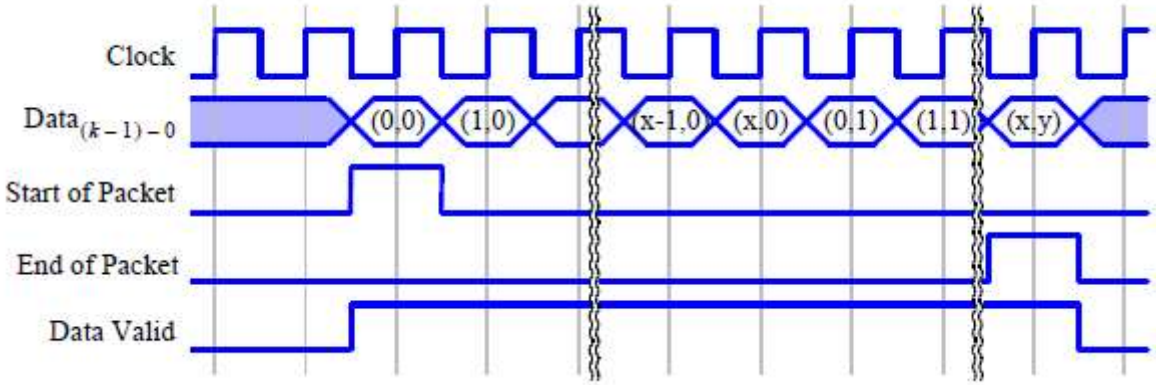


Fig. 3.5 – Formato del pacchetto streaming del frame video

Il formato di ogni pixel nel pacchetto dipende dallo spazio dei colori; come descritto in precedenza nel caso in esame lo spazio dei colori è RGB.

Alcuni video core non richiedono la conoscenza dello spazio dei colori ma richiedono di sapere il numero di bit per pixel. Per questi core, è importante sapere il numero di data bit per simbolo ed il numero di simboli per beat.

Con simbolo si indica la più piccola unità di dati. Uno o più simboli compongono una singola unità di dati trasferiti in un ciclo o un beat.

Con beat si indica un singolo ciclo di trasferimento di uno o più simboli tra due interfacce, una *sink* ed una *source*.

4 Computer

4.1 Introduzione al computer

Il computer è stato progettato mediante il tool *Qsys* presente in *Quartus Prime*. Questo tool permette di assemblare l'architettura del computer attraverso componenti hardware presenti in libreria e a componenti hardware descritti dall'utente in linguaggio HDL. Per la realizzazione di questo progetti, non è stato necessario creare dei custom component, ma ci si è avvalsi esclusivamente di componenti già presenti in libreria.

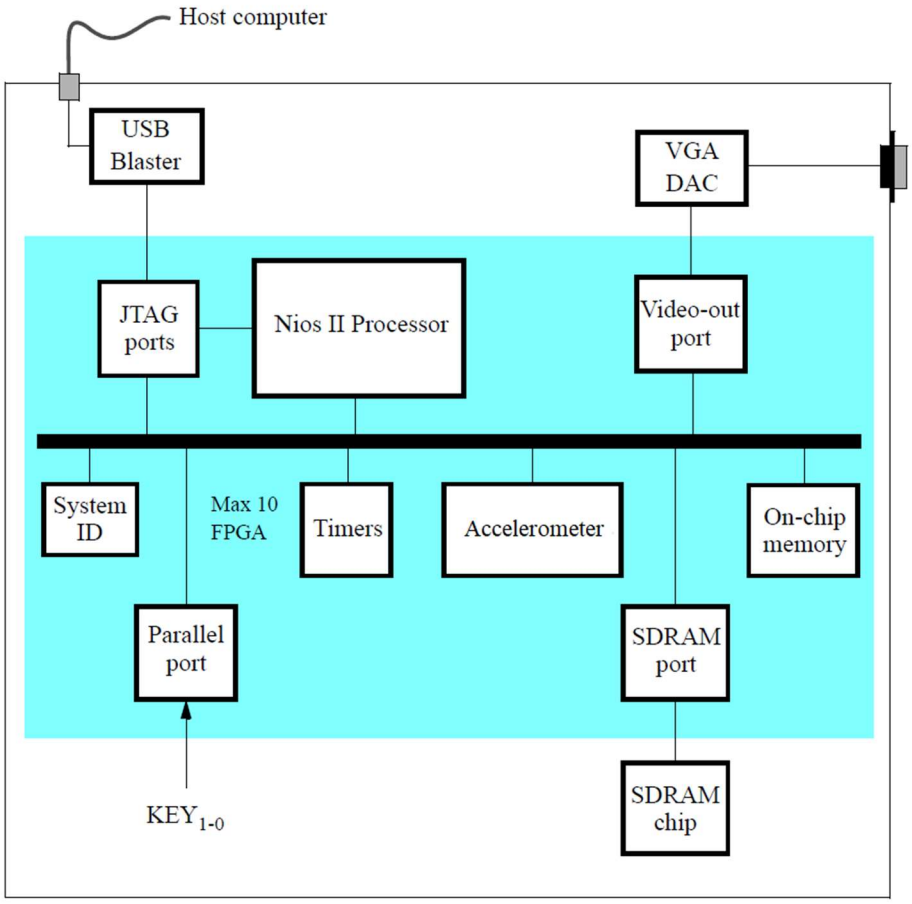


Fig. 4.1 – Schema a blocchi del Computer.

4.2 Implementazione computer su Qsys

Il computer in generale è costituito da un'istanza principale chiamata *Computer_system* e da altre due chiamate *Char_Buf_Subsystem* e *VGA_Subsystem*.

4.2.1 Computer_system

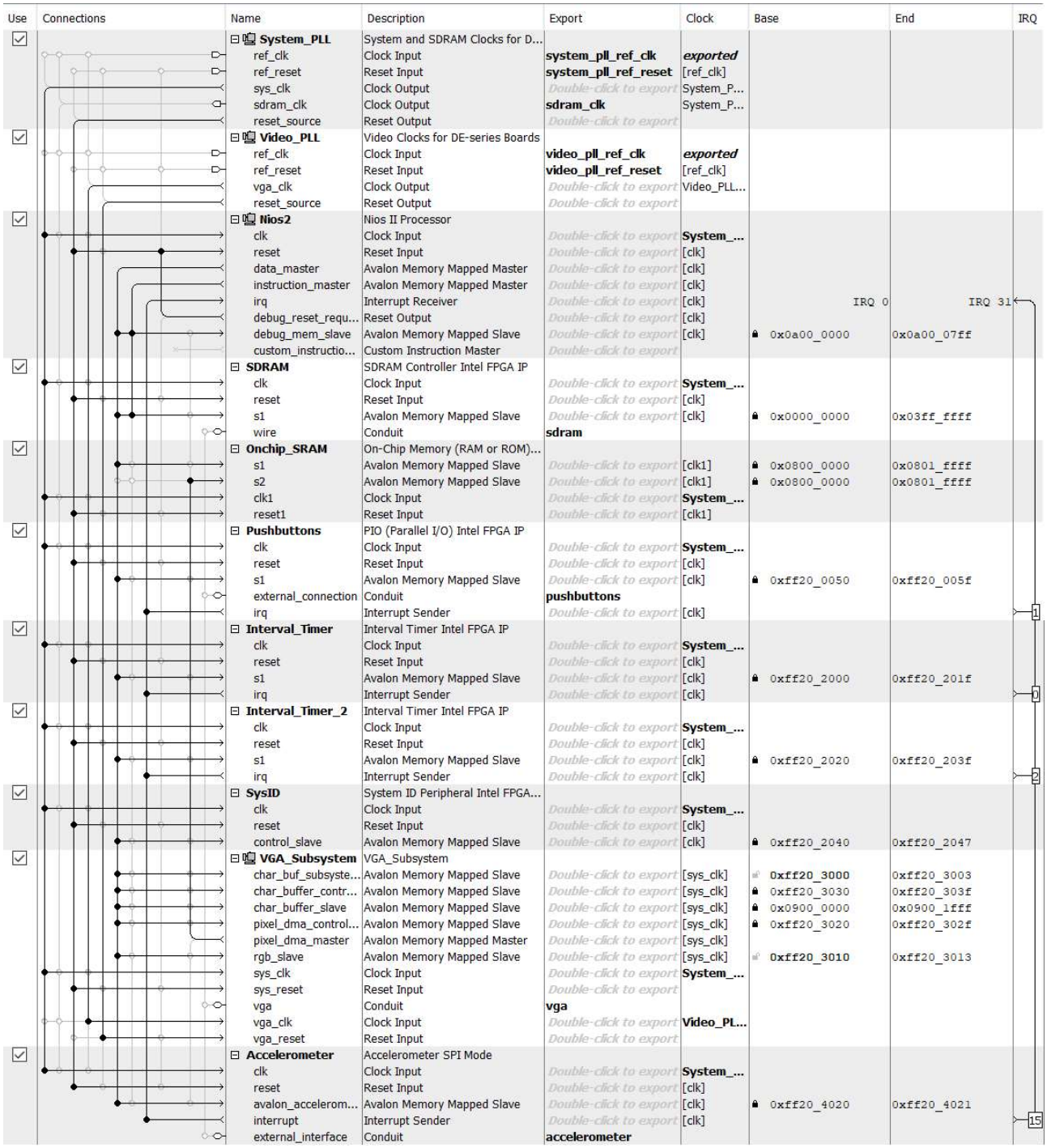


Fig. 4.2.1 – Istanziazione *Computer_system*.

Il sistema *Computer_system* è formato dai seguenti componenti:

- *System_PLL*
- *Video_PLL*
- *Nios2 Processor*
- *SDRAM*
- *On-chip RAM*
- *Pushbuttons*
- Due *Interval Timer*
- *SysID*
- *VGA_Subsystem*
- *Accelerometer*

4.2.1 System PLL

La DE10-Lite Board include un oscillatore on-board che genera un clock a frequenza 50MHz.

Questo clock può essere usato direttamente per i registri nel FPGA. Tuttavia la *SDRAM* ed il chip VGA DAC sulla board richiedono clock specifici. Questi clock possono essere generati partendo dal clock da 50 MHZ utilizzando dei PLL (*Phase-Locked Loop*) on-board.

Il tool *Qsys* fornisce da libreria due core di PLL differenti per la *SDRAM* ed il chip VGA DAC; questi core producono circuiti praticamente identici. Uno schema a blocchi generico di questi core è mostrato nella **Fig. 4.2.1**.

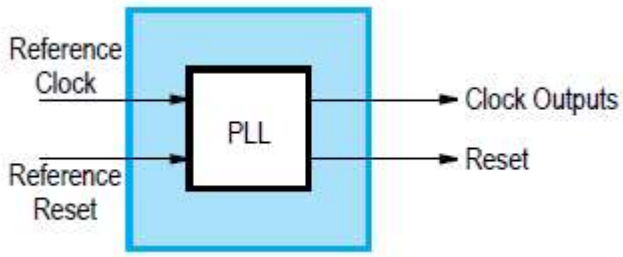


Fig. 4.2.1 – Schema *PLL*.

Vengono richiesti un reset ed un clock di riferimento, connessi al clock ed al reset del FPGA. Questi core producono uno o più clock in uscita ed un reset. Il reset in uscita viene connesso come reset dei componenti che utilizzano il/i clock in uscita dal PLL.

La SDRAM richiede un clock alla stessa frequenza del sistema embedded ma con uno shift di fase specifico.

Le varie SDRAM delle varie board della serie DE richiedono uno shift di fase differente. Fortunatamente, il core "*System and SDRAM PLL*" può generare un clock per il sistema ad una frequenza scelta dal progettista e crea un appropriato clock per la SDRAM con un appropriato shift di fase in base alla board selezionata dal progettista.

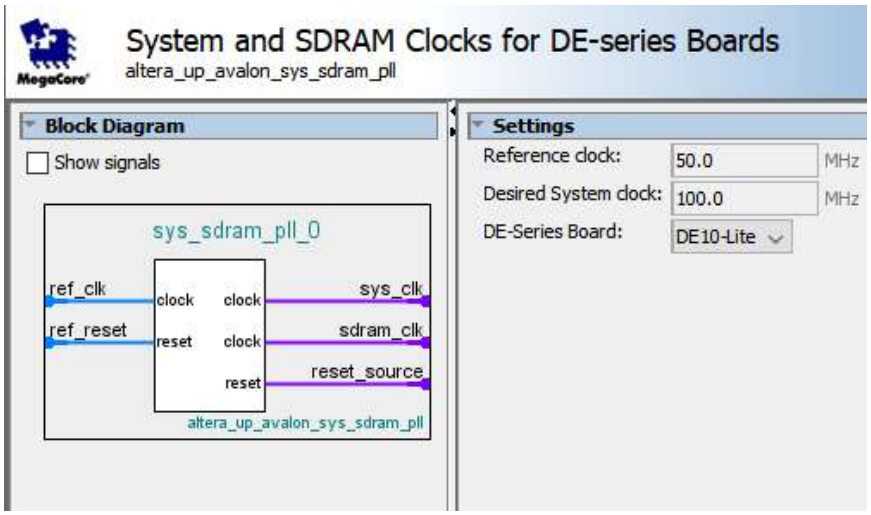


Fig. 4.2.2 – Blocco *System and SDRAM PLL*.

Quando viene istanziato il core è possibile scegliere:

- *Reference clock*: permette al progettista di specificare la frequenza del clock di riferimento. Nel nostro caso 50 MHz;
- *Desired System clock*: permette al progettista di specificare la frequenza desiderata per il clock del sistema. Impostata a 100 MHz;
- *DE-Series Board*: il progettista deve specificare la board utilizzata. Nel nostro caso DE10-Lite.

4.2.2 Video_PLL

La frequenza richiesta dal VGA DAC dipende dalla risoluzione del frame desiderata.

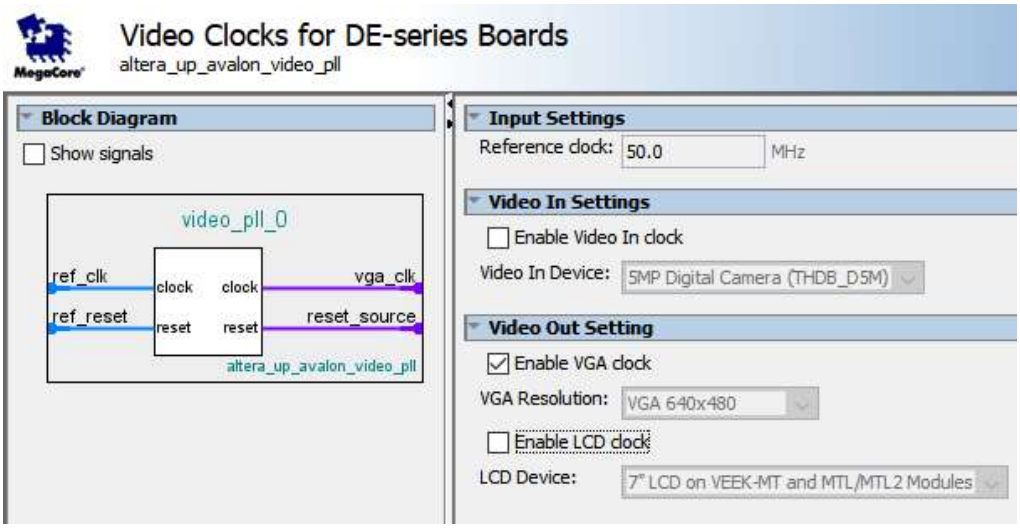


Fig. 4.2.3 - Blocco *Video Clocks*

Quando viene istanziato il core è possibile specificare:

- *Reference clock*: impostato a 50MHz;
- *Enable Video In clock*: non selezionato perché non utile ai fini del progetto;
- *Enable VGA clock*: selezionato, permette al progettista di specificare se generare il clock per il *VGA DAC on-board*;
- *VGA Resolution*: permette al progettista di specificare la risoluzione VGA desiderata. Nel caso in esame è impostata a "VGA 640x480";
- *Enable LCD clock*: non selezionato perché non utile ai fini del progetto.

4.2.3 Nios II Processor

Il processore presenta un'architettura RISC general purpose con istruzioni e registri implementati su 32 bit.

Per il computer in questione, si è utilizzata la versione *Economy* del NIOS II Processor.

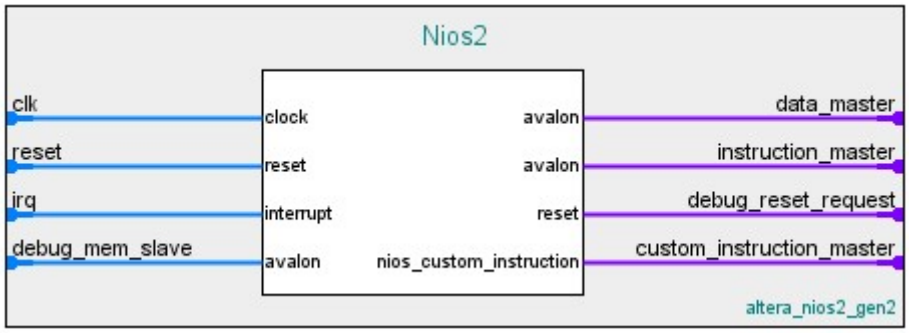


Fig. 4.2.3 – Schema *Nios II Processor*

Il blocco presenta le seguenti interfacce:

- *Avalon Clock:* serve per la ricezione del segnale di clock dal *System and SDRAM PLL*;
- *Avalon Reset:* serve per la ricezione del segnale di reset dal *System and SDRAM PLL*;
- *Avalon Interrupt Receiver:* serve per l'acquisizione del segnale di *Interrupt Request* (IRQ) proveniente dai componenti *Pushbuttons*, *Interval_Timer*, *Interval_Timer_2* ed *Accelerometer*;
- *Avalon MM Slave debug_mem_slave:* serve per la gestione dei break introdotti dall'utilizzo del Debug Module;
- *Avalon MM Master Data:* serve per la gestione della lettura/scrittura dati da e verso i componenti *SDRAM*, *Onchip_SRAM*, *Pushbuttons*, *Interval_Timer*, *Interval_Timer_2*, *SysID*, *VGA_Subsystem* (l'interfaccia sarà collegata alla *On-Chip RAM* contenente il *Character Buffer*, al *Char_Buf_DMA*, al *RGB Resampler* e al *VGA_Pixel_DMA*) e *Accelerometer*;
- *Avalon MM Master Istructions:* serve per la gestione di lettura istruzioni dalla *SDRAM*;
- *Avalon Reset Output:* serve per il pilotaggio di un segnale di reset interno al computer e ai componenti presenti su di esso.

4.2.4 SDRAM

Il controller SDRAM fornisce un'interfaccia alla SDRAM da 64 MB sulla DE10-Lite board, la quale è organizzata come 32M x 16 bits. È mappata sullo spazio di indirizzamento da 0x00000000 a 0x03FFFFFF.

Lo *SDRAM controller core* fornisce un'interfaccia *Avalon Memory-Mapped* verso la *SDRAM*.

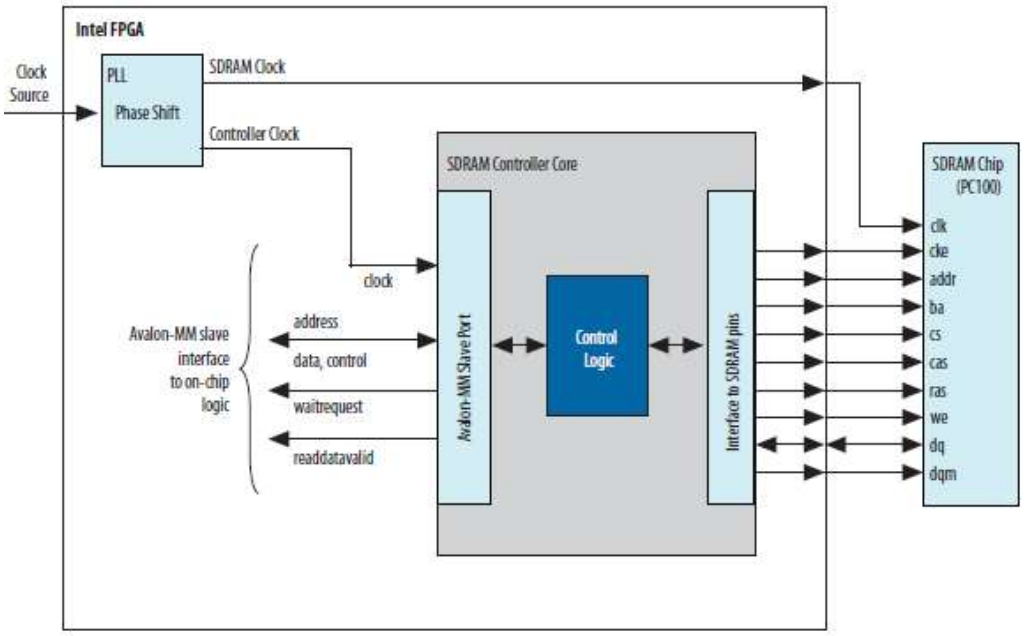


Fig. 4.2.4.1 - Diagramma a blocchi del core *SDRAM Controller* connesso al chip *SDRAM*

Il clock del chip *SDRAM* deve avere la stessa frequenza del clock per l'interfaccia Avalon-MM sul controller *SDRAM*: ciò è garantito dal *System_PLL*.

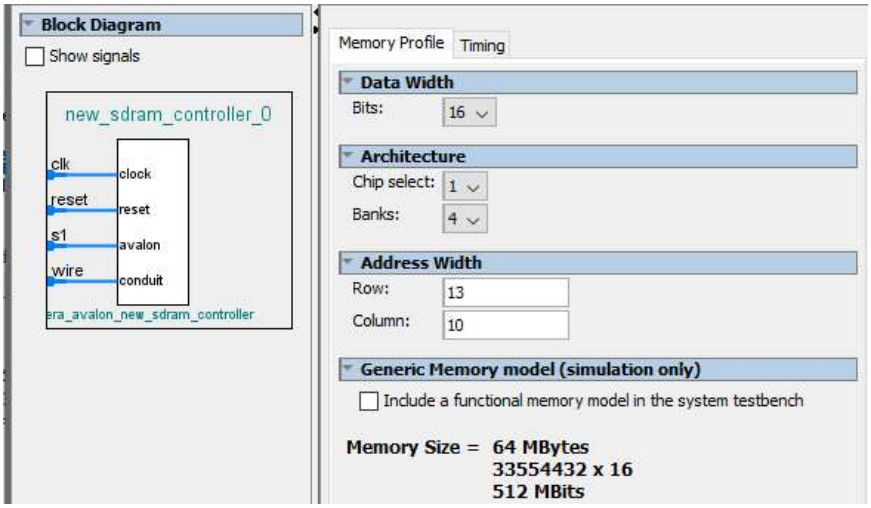


Fig. 4.2.4.2 - Configurazione utilizzata per il core *SDRAM controller – Memory Profile*

La configurazione delle voci nel *Memory profile* e nel *Timing* sono state effettuate consultando il datasheet della memoria SDRAM utilizzata.

Nel nostro caso è stata utilizzata la SDRAM "IS42S16320F – 7TL".

- *Data width*: settata a 16. Questo valore determina il numero di bit per il bus *dq* (bus dati) ed il bus *dqm* (byte-enable).
- *Chip select*: settata ad 1. Indica il numero di chip di SDRAM indipendenti.
- *Banks*: settata a 4. Numero di banchi della SDRAM. Questo valore determina il numero di bit del bus ba (bank address).
- *Address settings*: i valori *Row* e *Column* dipendono dalla geometria della SDRAM scelta. Nel nostro caso *Row* è stata impostata a 13 e *Column* è stata impostata a 10.

In base alle scelte di progetto viene mostrata la capacità della memoria in unità di megabytes, numero di parole indirizzabili e megabits.

La pagina *Timing* permette al progettista di impostare le specifiche di timing del chip SDRAM usato. I valori corretti sono disponibili nel datasheet menzionato in precedenza.

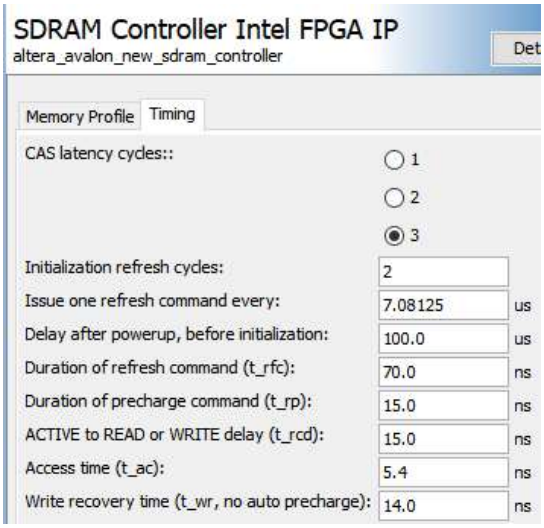


Fig. 4.2.4.3 - Configurazione utilizzata per il core SDRAM controller – Timing

4.2.5 Onchip_RAM

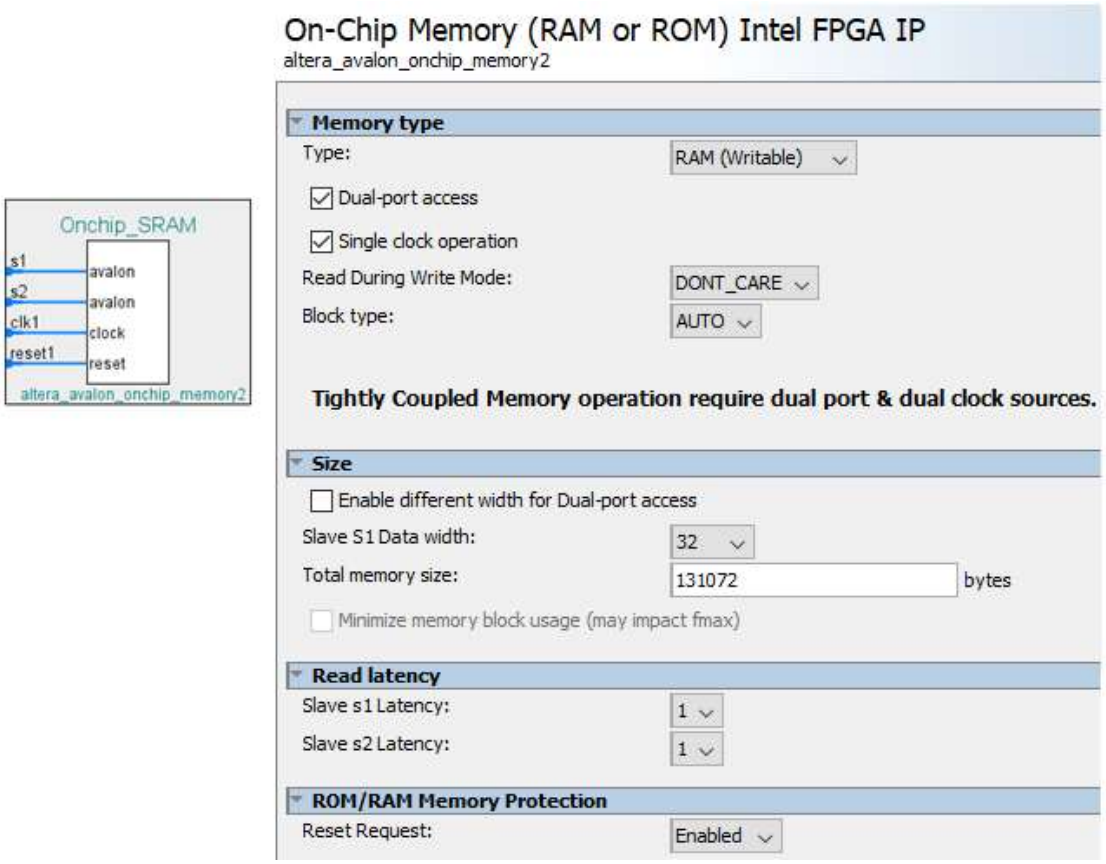


Fig. 4.2.5.1 - Componente *On-Chip Memory* (RAM) (prima parte)

Questa memoria è organizzata come 2K x 32 bits, ed è mappata agli indirizzi nel range da 0x08000000 a 0x0800FFFF. Questa memoria contiene il *Frontbuffer* ed il *Backbuffer*.
Blocco *On-Chip Memory* è una memoria del tipo:

- *RAM* (Writable), le periferiche master possono sia leggere che scrivere nella memoria;
- *Dual-port access,* così che due master possano accedere alla stessa memoria contemporaneamente;
- *Single clock operation*, cioè entrambi gli slave hanno la stessa sorgente di clock;
- Non importa qual è il dato di output della memoria quando vengono fatte simultaneamente una lettura ed una scrittura alla stessa locazione di memoria;
- Quando il software esegue il fitting sulla scheda sceglie in maniera automatica il blocco di memoria.
- Andando a collegare l'interfaccia data master del processore alla *On-chip Memory* abbiamo impostato la larghezza dei dati in memoria a 32 bit, cioè la stessa larghezza dei dati del data master.
- La dimensione totale del blocco di memoria è stata impostata a 131072 bytes.
- L'opzione "*Read latency*" permette di specificare il numero di cicli di latenza in lettura richiesti per accedere i dati; è stato impostato un ciclo di latenza per entrambi gli slave (dual port-access).
- RAM/ROM Memory Protection: abilitato, viene creata una porta di richiesta di reset aggiuntiva per la protezione della memoria durante il reset.

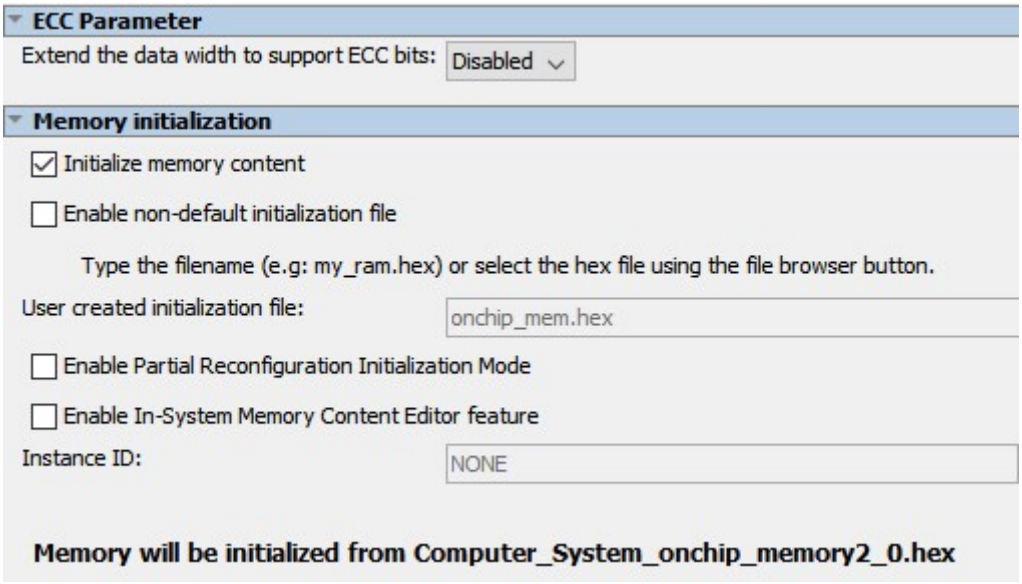


Fig. 4.2.5.2 - Componente *On-Chip Memory* (RAM) (seconda parte)

La feature di estensione della larghezza dei dati per supportare i bit ECC è disabilitata.

Per quanto riguarda l'inizializzazione della memoria:

- Abbiamo abilitato l'inizializzazione del contenuto della memoria
- Non abbiamo specificato il file di inizializzazione, verrà quindi utilizzato il file di default creato dal *Platform Designer*.
- Le altre due voci sono disabilitate.

Il blocco On-Chip Memory presenta le seguenti interfacce:

- *Interfaccia Avalon Clock* per la ricezione del segnale di clock dal System and SDRAM PLL;
- *Interfaccia Avalon Reset input* per la ricezione del segnale di reset dal System and SDRAM PLL;
- *Interfaccia Avalon MM Slave s1* per la gestione della lettura/scrittura dati da parte del NIOS II Processor;
- *Interfaccia Avalon MM Slave s2* collegata all'interfaccia Avalon MM Master del DMA Controller.

4.2.6 Pushbuttons

Il core *Parallel Input/Output* (PIO) con interfaccia Avalon fornisce un'interfaccia Memory-Mapped tra una porta slave Avalon MM e le porte di I/O general purpose. Nel progetto il core *PIO* è stato utilizzato per implementare i *Pushbutton*.

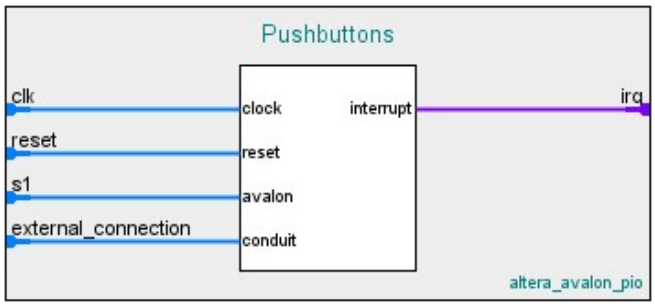


Fig. 4.2.6.1 – Schema *Pushbuttons*

Il blocco presenta le seguenti interfacce:

- *Avalon Clock* per la ricezione del segnale di clock dal System and SDRAM PLL;
- *Avalon Reset* per la ricezione del segnale di reset dal System and SDRAM PLL;
- *Avalon MM Slave* per la gestione della lettura/scrittura dati da parte del NIOS II Processor;
- *Avalon Conduit* per la connessione alle porte di I/O general purpose (*Pushbuttons*);
- *Interrupt Sender* collegata all'interfaccia Interrupt Receiver del processore NIOS II.

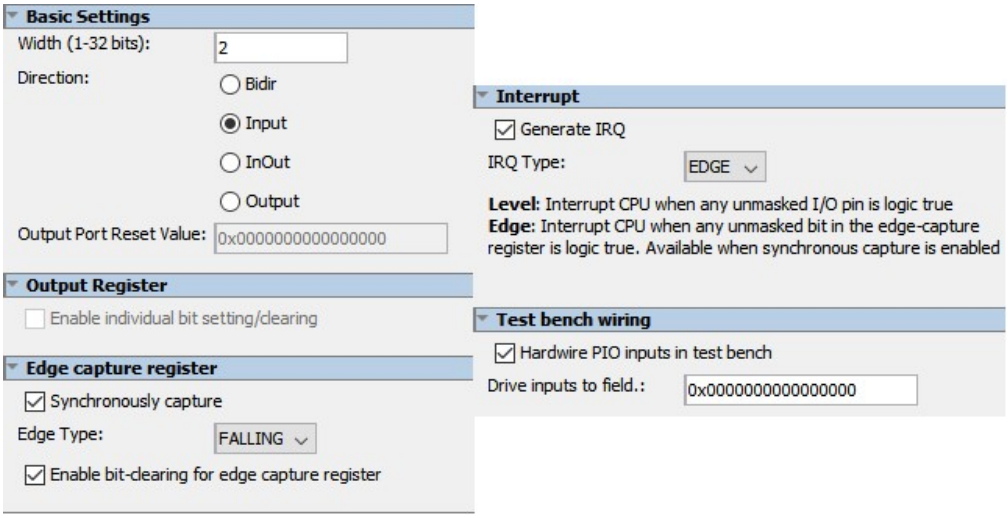


Fig. 4.2.6.2 – Configurazione dei *Pushbuttons*

Configurazione dei *Pushbuttons*:

- Vi sono due *Pushbutton* sulla board, catturano l'input;
- È presente il registro di *edgecapture*;
- Viene generato un segnale IRQ in uscita quando un bit dell'*edgecapture* va al livello logico alto.

4.2.7 Interval Timer

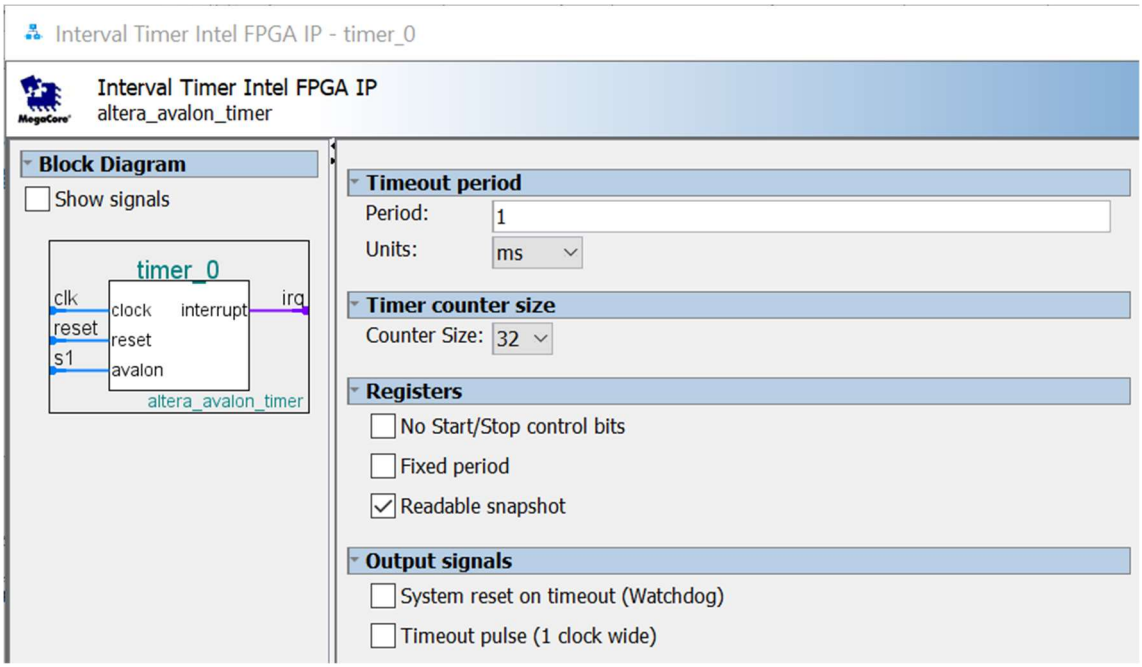


Fig. 4.2.7 - Componente *Interval Timer*

Sono presenti due *Interval Timer*, configurati allo stesso modo. Uno gestisce il movimento degli invasori ed uno gestisce lo spostamento dei proiettili.

Gli *Interval Timer* sono configurati con un contatore a 32 bit; non sono stati configurati con periodo fisso in modo da poterlo cambiare facilmente attraverso il codice sorgente durante la fase di progetto.

Oltre al periodo, nel codice viene impostata la modalità di conteggio come *continuous count-down*: quando il contatore raggiunge lo zero riprende a contare dal valore inziale.

Quando il contatore raggiunge il valore zero viene generato un segnale di interrupt IRQ e viene eseguita una certa routine di gestione dell'interrupt.

Blocco *Interval Timer* presenta le seguenti interfacce:

- *Avalon Clock Input:* serve per la ricezione del segnale di clock dal System and SDRAM PLL;
- *Avalon Reset input:* serve per la ricezione del segnale di reset dal System and SDRAM PLL;
- *Avalon MM Slave:* serve per la gestione della lettura/scrittura dati da parte del NIOS II Processor;
- *Avalon Interrupt Sender:* serve per la generazione del segnale di Interrupt Request (IRQ) verso il NIOS II Processor.

4.2.8 SysID

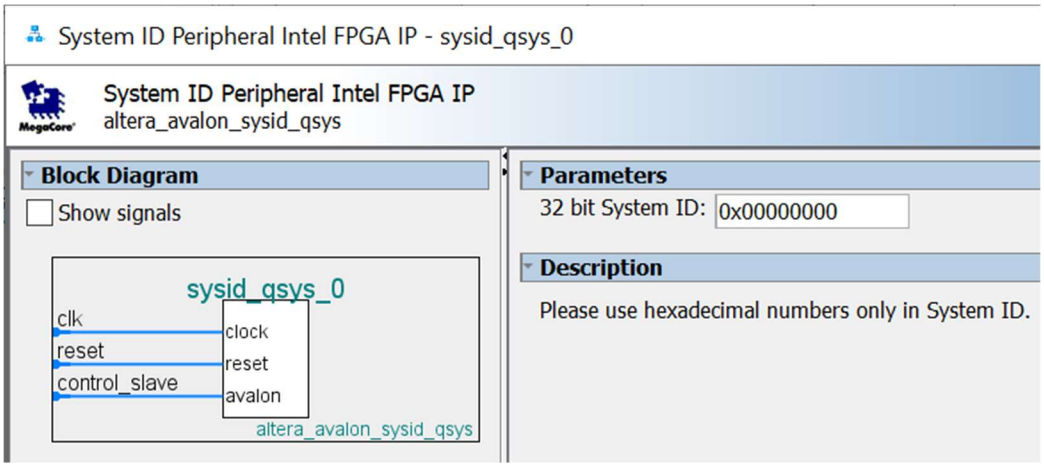


Fig. 4.2.8 - Componente *System ID Peripheral*.

Il core *System ID* è un semplice dispositivo di sola lettura che fornisce al processore NIOS II un identificatore univoco.

Blocco *System ID Peripheral* presenta le seguenti interfacce:

- *Avalon Clock Input:* serve per la ricezione del segnale di clock dal System and SDRAM PLL;
- *Avalon Reset input:* serve per la ricezione del segnale di reset dal System and SDRAM PLL;
- *Avalon MM Slave:* serve per la gestione di lettura dati da parte del NIOS II Processor.

4.2.9 VGA_Subsystem

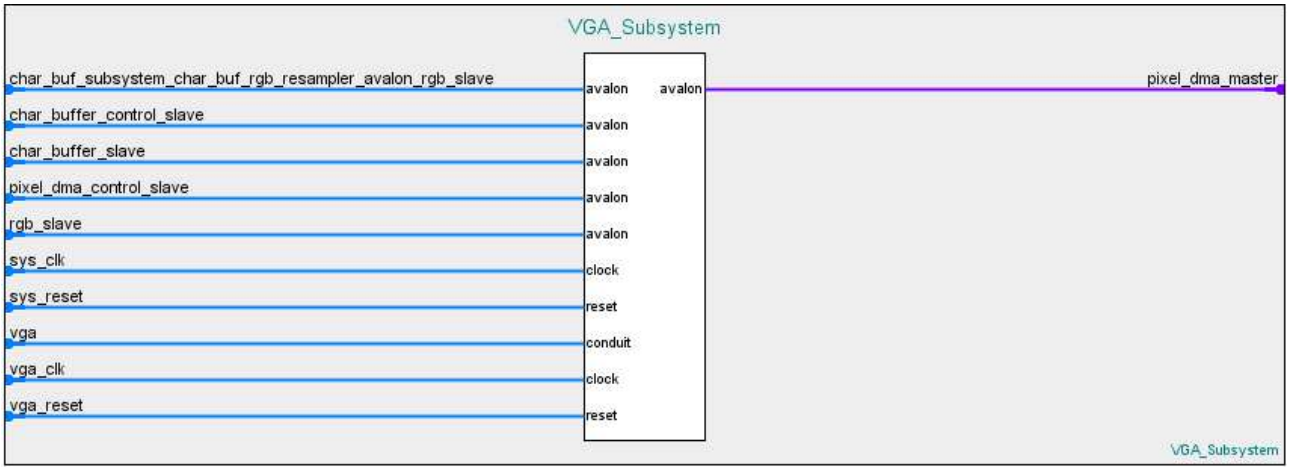


Fig. 4.2.9 –Interfacce del sottosistema VGA

Questo sottosistema legge ed elabora i dati del *Pixel Buffer* e del *Character Buffer* (questo sottosistema comprende il *Char_Buf_Subsystem*) per poterli visualizzare in modo appropriato sul monitor VGA.

Presenta le seguenti interfacce:

- *Avalon MM char_buf_..._rgb_slave* per la gestione della lettura/scrittura dati da parte del NIOS II Processor;
- *Avalon MMchar_buffer _control_slave* per la gestione della lettura/scrittura dati da parte del NIOS II Processor;
- *Avalon MMchar_buffer _slave* per la gestione della lettura/scrittura dati da parte del NIOS II Processor;
- *Avalon MM pixel_dma_control_slave* per la gestione della lettura/scrittura dati da parte del NIOS II Processor;
- *Avalon MM rgb_slave* per la gestione della lettura/scrittura dati da parte del NIOS II Processor;
- *Avalonsys_clk* per la ricezione del segnale di clock dal System and SDRAM PLL;
- *Avalonsys_reset* per la ricezione del segnale di reset dal System and SDRAM PLL;
- *Avalon Conduit* per le connessioni con la porta VGA;
- *Avalonsys_clk* per la ricezione del segnale di clock dal Video PLL;
- *Avalonsys_reset* per la ricezione del segnale di reset dal Video PLL;
- *Avalon MM pixel_dma_master* per la lettura/scrittura dalla On-chip RAM contenente il pixel buffer.

4.2.10 Accelerometer

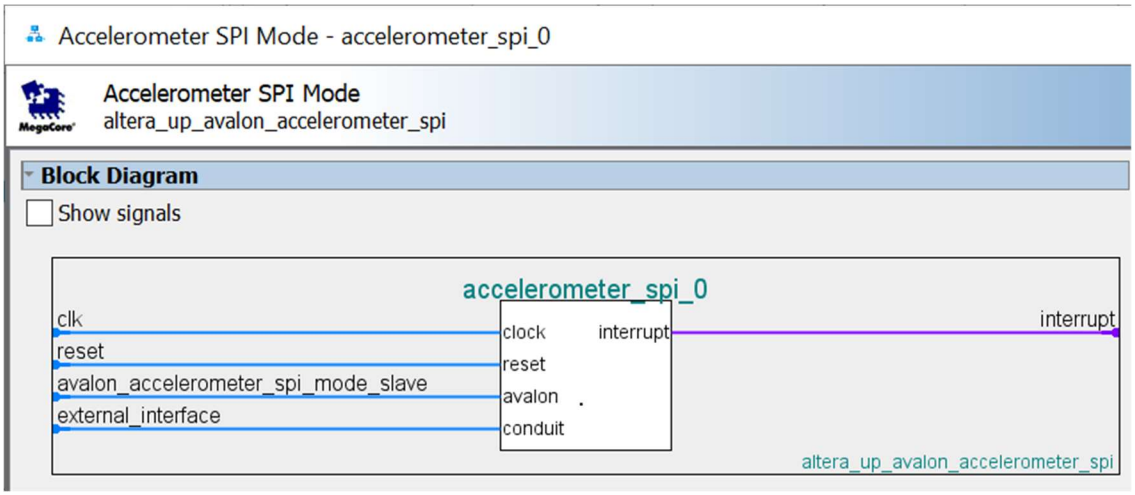


Fig. 4.2.10 - Componente *Accelerometer*.

L'accelerometro on-board consiste nel chip *ADXL345*. Questo chip misura l'accelerazione in tre direzione, riferite come asse-x, asse-y ed asse-z. La misura ha una risoluzione di 13 bit, nel range di ±16g.

Il chip include un set di registri ad 8-bit per immagazzinare i risultati delle misure e per impostare le varie modalità di utilizzo dell'accelerometro. Per via della risoluzione di 13-bit, le misure lungo x, y e z sono immagazzinate in coppie di registri.

Una descrizione completa dell'accelerometro può essere trovata nel datasheet a cura di Analog Device: Digital Accelerometer ADXL345.

Il componente Accelerometro presenta le seguenti interfacce:

- *Avalon Clock* per la ricezione del segnale di clock dal System and SDRAM PLL;
- *Avalon Reset* per la ricezione del segnale di reset dal System and SDRAM PLL;
- *Avalon MM* Slave spi_mode per la gestione della lettura dei dati da parte del NIOS II Processor;
- *Avalon conduit* per la connessione con l'accelerometro on board;
- *Avalon Interrupt Sender* per la generazione del segnale di Interrupt Request (IRQ) verso il NIOS II Processor.

4.3 VGA_Subsystem

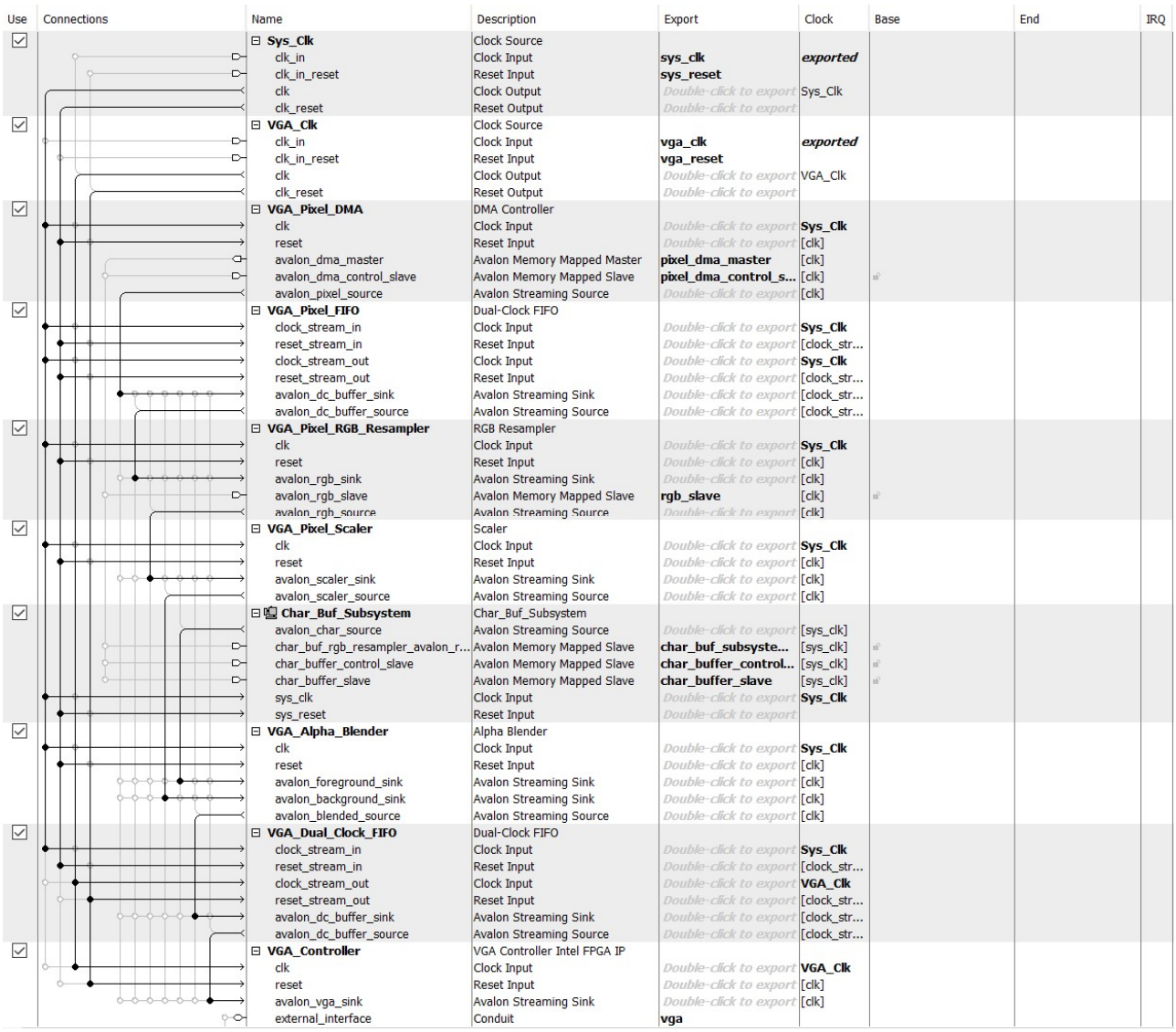


Fig. 4.3 – Istanziazione *VGA_Subsystem*.

Il *VGA_Subsystem* è formato dai seguenti componenti:

- *Due Clock Source* (*Sys_Clk* e *VGA_Clk*)
- *DMA Controller*
- *Due Dual-Clock FIFO* (*VGA_Pixel_FIFO* e *VGA_Dual_Clock_FIFO*)
- *RGB Resampler*
- *Scaler*
- *Char_Buf_Subsystem*
- *Alpha Blender*
- *VGA Controller*

4.3.1 Clock Source

Permette di distribuire il segnale di clock ed il segnale di reset agli altri moduli. La voce *Clock frequency is known* non viene settata, così facendo si indica che la frequenza verrà impostata al di fuori del sottosistema.

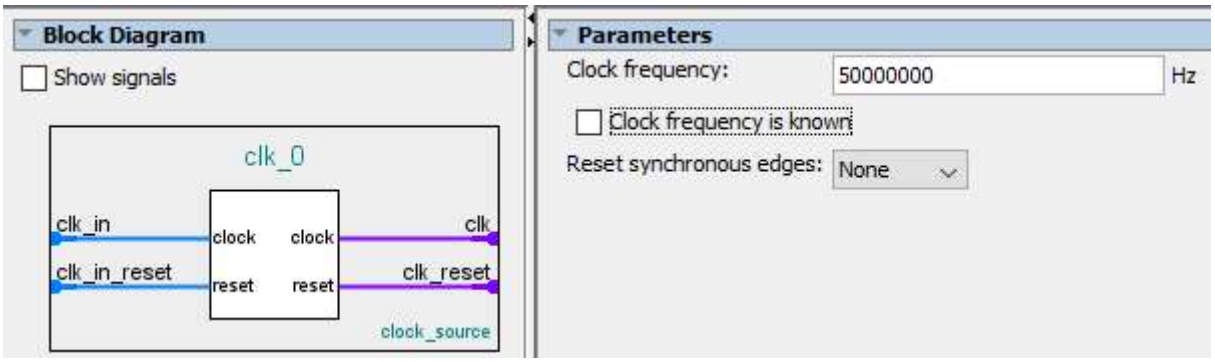


Fig. 4.3.1 - Componente *Clock Source.*

Le interfacce del componente sono le seguenti:

- *Clock Input*: utilizzata per prendere in ingresso il segnale di clock;
- *Reset Input*: utilizzata per prendere in ingresso il segnale di reset;
- *Clock Output*: utilizzata per fornire in uscita il segnale di clock;
- *Reset Output*: utilizzata per fornire in uscita il segnale di reset;

Come scritto in precedenza, nel *VGA_Subsystem* ci sono due *Clock Source*:

- *Sys_Clk*: prende in ingresso il segnale di clock ed il segnale di reset provenienti da System and SDRAM PLL e li distribuisce al DMA Controller, al DC FIFO fra il DMA Controller ed il RGB Resampler (sia come clock/reset di ingresso sia come clock/reset di uscita), al RGB Resampler, allo Scaler, al Char_Buf_Subsystem, all' Alpha Blender e come clock/reset di ingresso del secondo DC FIFO;
- *VGA_Clk:* prende in ingresso il segnale di clock ed il segnale di reset provenienti da Video_PLL e li distribuisce come clock/reset di uscita del secondo DC FIFO e al VGA Controller.

4.3.2 DMA Controller

Il core *DMA Controller* immagazzina e recupera frame video da e verso la memoria. Il *DMA controller* è settato nella modalità operativa *From Memory to Stream*. In questa modalità il DMA Controller usa la propria interfaccia Avalon MM Master per leggere frame video da una memoria esterna. Successivamente invia questi frame video in uscita attraverso la propria interfaccia Avalon streaming. L'interfaccia Avalon MM slave del DMA controller viene usata per comunicare con i registri interni del controller.

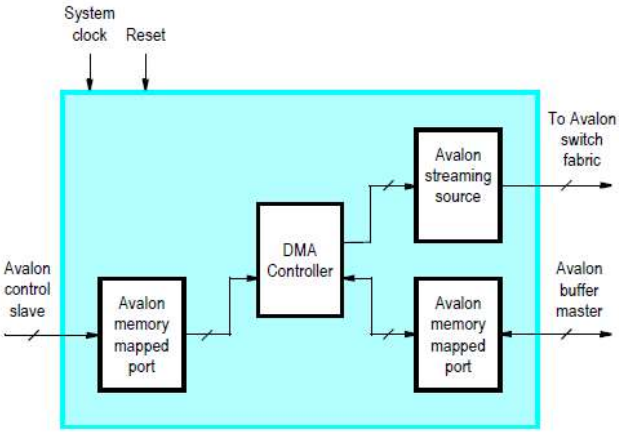


Fig. 4.3.2.1–Diagramma a blocchi DMA Controller

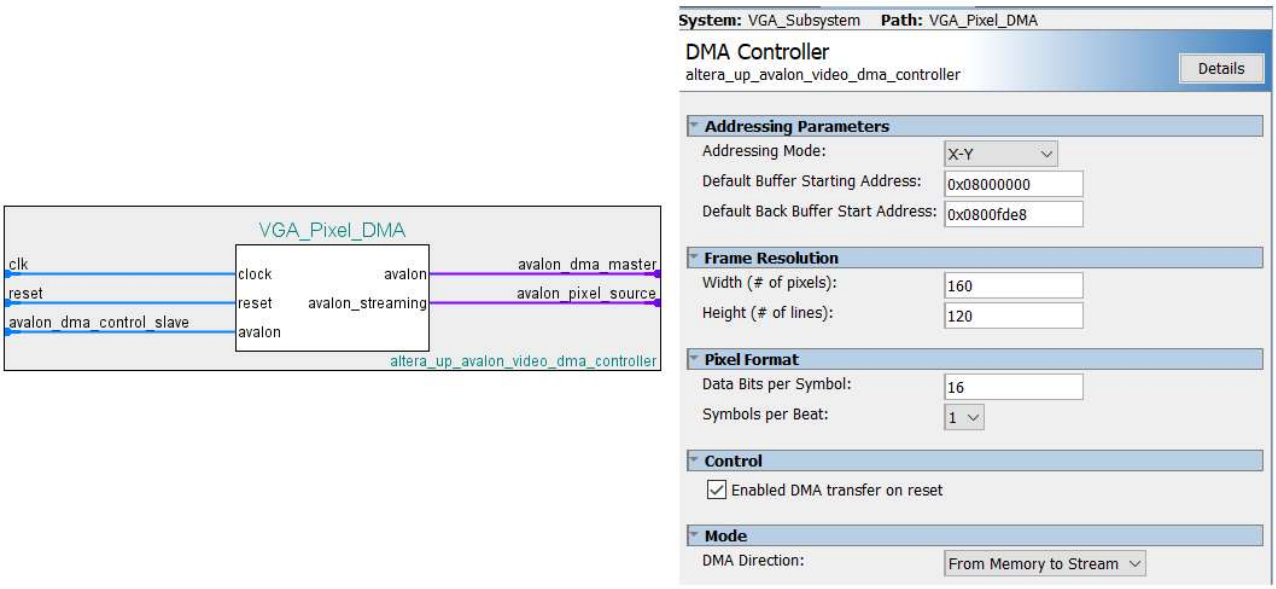


Fig. 4.3.2.2 – Parametri del *DMA Controller.*

Per scrivere e leggere i frame verso e dalla memoria il *DMA Controller* utilizza la modalità di indirizzamento X-Y. L'indirizzo di partenza del *Buffer* è 0x08000000 mentre quello del *Back Buffer* è 0x0800fde8.

La risoluzione del frame letto è 160x120. Il numero di data bits per simbolo è 16 mentre il numero di simboli per beat è 1.

4.3.3 Dual Clock FIFO (DC FIFO)

Questi buffer di dati video aiutano nel trasferimento di stream video tra due domini di clock.

I dati video entrano in ingresso alla frequenza del clock di input. Il dato viene messo in una memoria FIFO, e successivamente viene trasmesso in uscita alla frequenza del clock di output.

Nel progetto sono presenti due DC FIFO: uno collegato tra il DMA Controller ed il RGB Resampler e l'altro collegato tra l'Alpha Blender ed il VGA Controller.

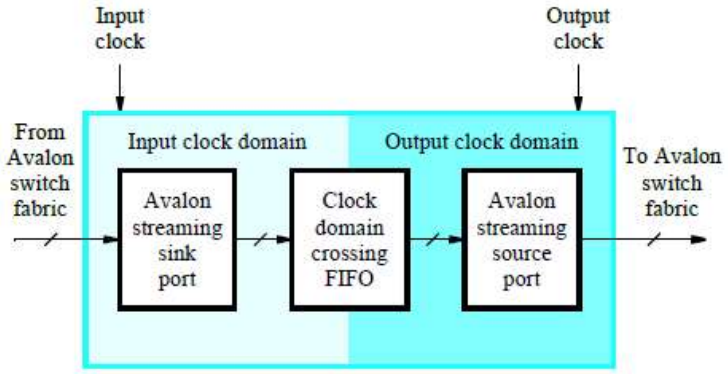


Fig. 4.3.3.1–Diagramma a blocchi DC FIFO

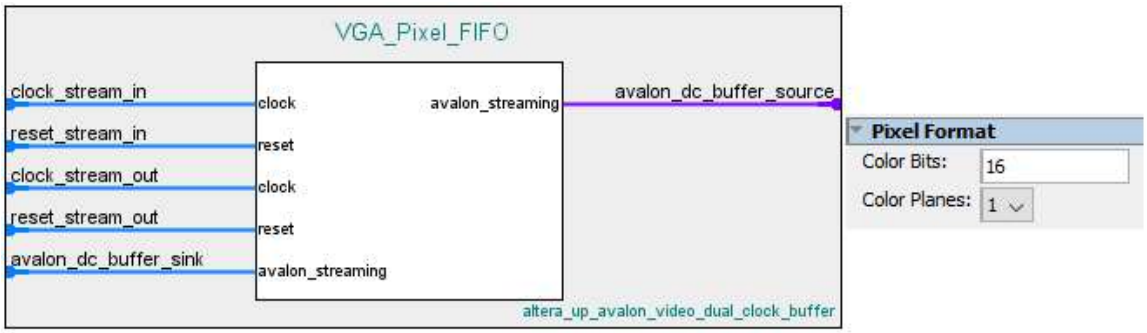


Fig. 4.3.3.2–Interfacce e parametri del DC FIFO

Per quanto riguarda il primo DC FIFO della catena il numero di color bit è 16 mentre il numero *color planes* è 1.

Per il secondo DC della catena il numero di color bit è 10 mentre il numero di *color planes* è 3.

4.3.4 RGB Resampler

Il core *RGB Resampler* converte flussi video tra i formati dello spazio di colore RGB.

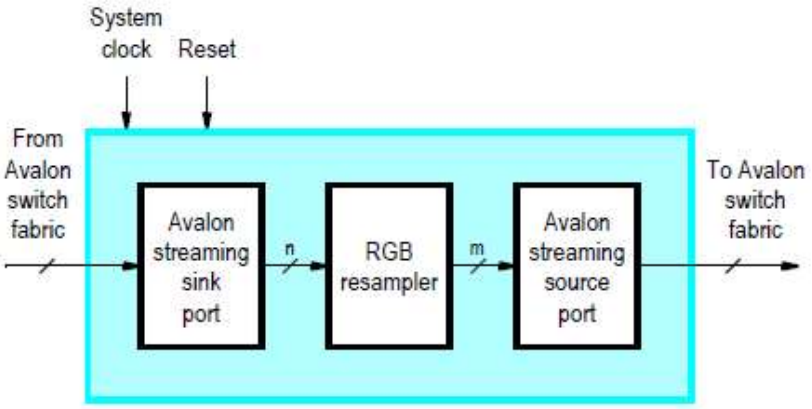


Fig. 4.3.4.1–Diagramma a blocchi *RGB Resampler*

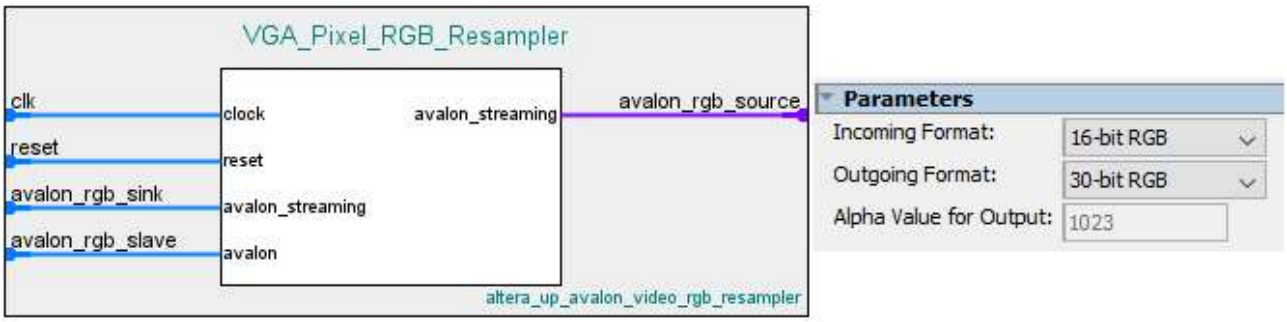


Fig. 4.3.4.2–Schema e configurazione RGB Resampler

Il formato del flusso che arriva in ingresso è 16-bit RGB, mentre quello in uscita è 30-bit RGB.

4.3.5 Scaler

Il core *Scaler* modifica la risoluzione del flusso video. Questo avviene convertendo la risoluzione del flusso video in ingresso aggiungendo o rimuovendo intere righe e/o colonne di pixel.

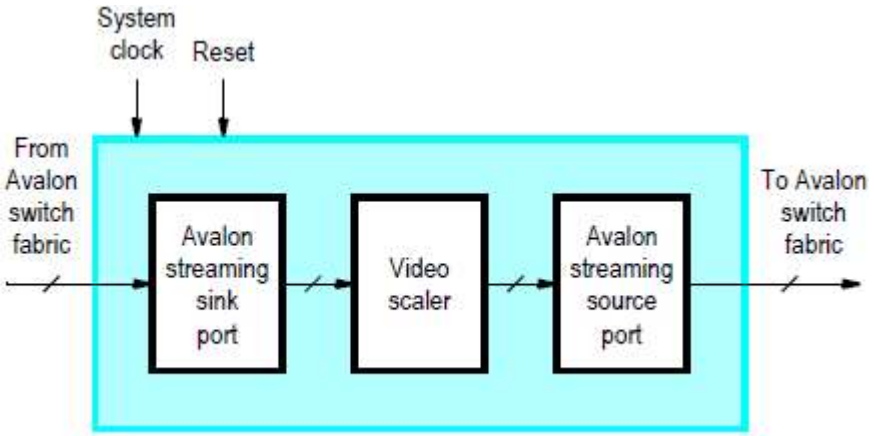


Fig. 4.3.5.1–Schema a blocchi dello *Scaler*

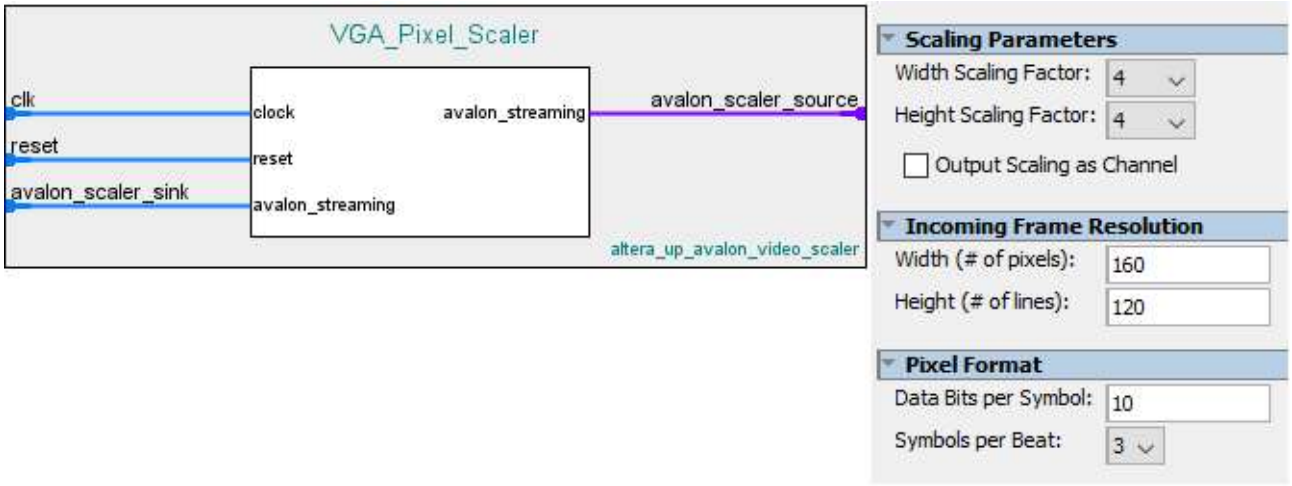


Fig. 4.3.5.2–Configurazione *Scaler*

I fattori di scaling di riga e colonna sono impostati entrambi a 4. La risoluzione del frame in ingresso è 160x120. Il numero di data bit per simbolo è 10 mentre il numero di simboli per beat è 3.

4.3.6 Char_Buf_Subsystem

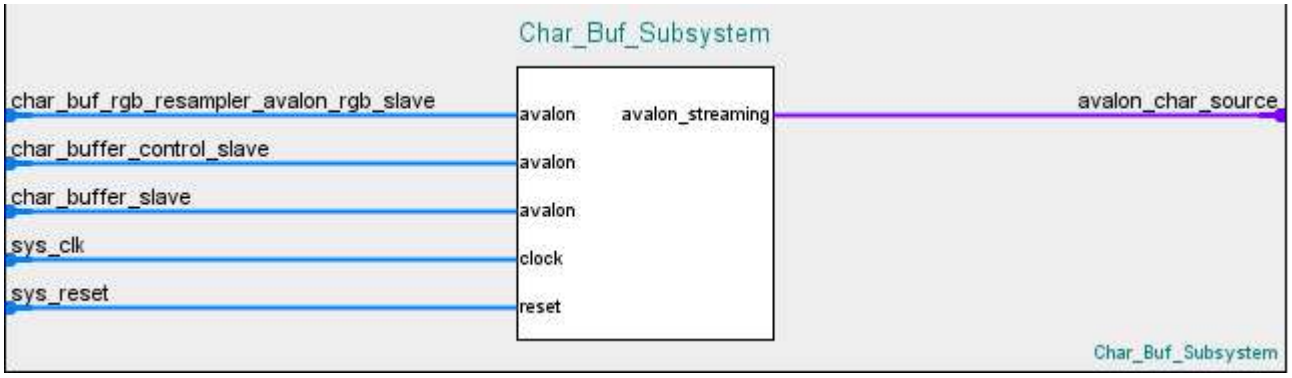


Fig. 4.3.6 – Interfacce *Char_Buf_Subsystem*

Questo sottosistema legge ed elabora i dati del *Character Buffer* per poter visualizzarli in modo appropriato sul monitor VGA.

Il componente presenta le seguenti interfacce:

- *Avalon MM char_buf_rgb_resampler_avalon_rgb_slave*: gestisce la lettura/scrittura dati da parte del NIOS II Processor;
- *Avalon MMchar_buffer _control_slave*: gestisce la lettura/scrittura dati da parte del NIOS II Processor;
- *Avalon MMchar_buffer _slave*: gestisce la lettura/scrittura dati da parte del NIOS II Processor;
- *Avalon Clock*: serve per la ricezione del segnale di clock dal *Sys_Clk*;
- *Avalon Reset*: serve per la ricezione del segnale di reset dal *Sys_Clk*;
- *Avalon Streaming*: serve per il flusso video *foreground*.

4.3.7 Alpha Blender

Il core *Alpha Blender* combina due flussi video in un unico flusso video. I due flussi video in ingresso sono detti "*foreground*" e "*background*". Il flusso video "*foreground*" deve essere nel formato 40-bit RGBA mentre il flusso video "*background*" deve essere nel formato 30-bit RGB. Il flusso video in uscita sarà nel formato 30-bit RGB.

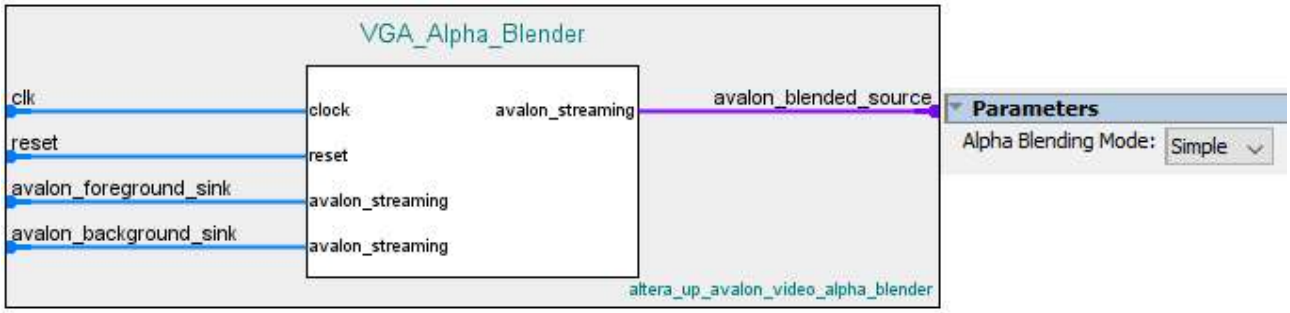


Fig. 4.3.7.1 – Interfacce e configurazione dell'*Alpha Blender*.

L'Alpha Blender è configurato in modalità "*Simple*": il valore alpha viene arrotondato a 0 o a 1, così viene semplificata la circuiteria.

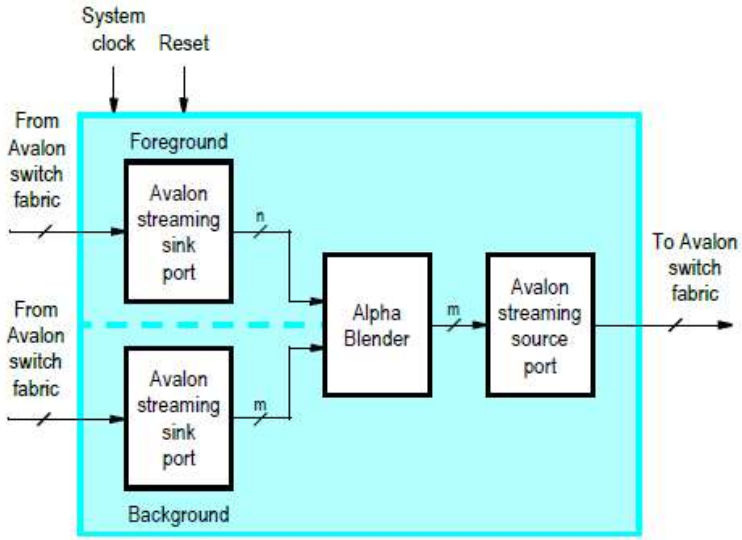


Fig. 4.3.7.2 – Schema a blocchi dell'Alpha Blender

4.3.8 VGA Controller

Il core *VGA Controller* genera i segnali di timing richiesti dalla VGA DAC on-board. I dati sono forniti al *VGA Controller* attraverso la sua interfaccia *Avalon Streaming*. Il controller prende i dati in ingresso, aggiunge i segnali di timing appropriati ed invia queste informazioni al VGA DAC on-board.

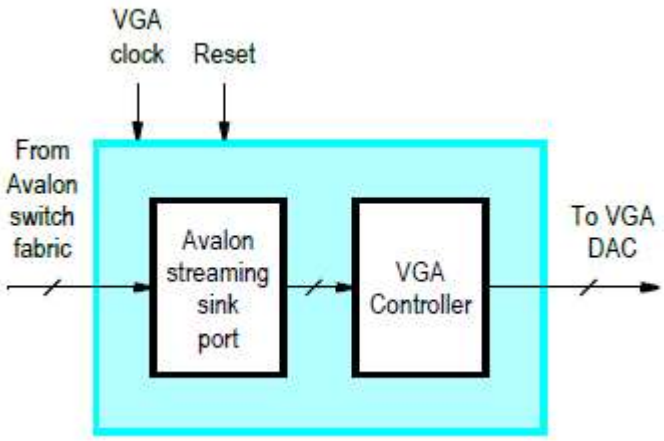


Fig. 4.3.8.1 – Schema a blocchi del *VGA Controller*

Fra i segnali di timing generati dal *VGA Controller* vi sono i segnali di sincronizzazione orizzontale e verticale. Per generare le informazioni di timing in maniera corretta deve essere fornito al *VGA Controller* un segnale di clock da 25MHz.

Fig. 4.3.8.2 – Interfacce e configurazione del *VGA Controller*.

4.4 Char_Buf_Subsystem

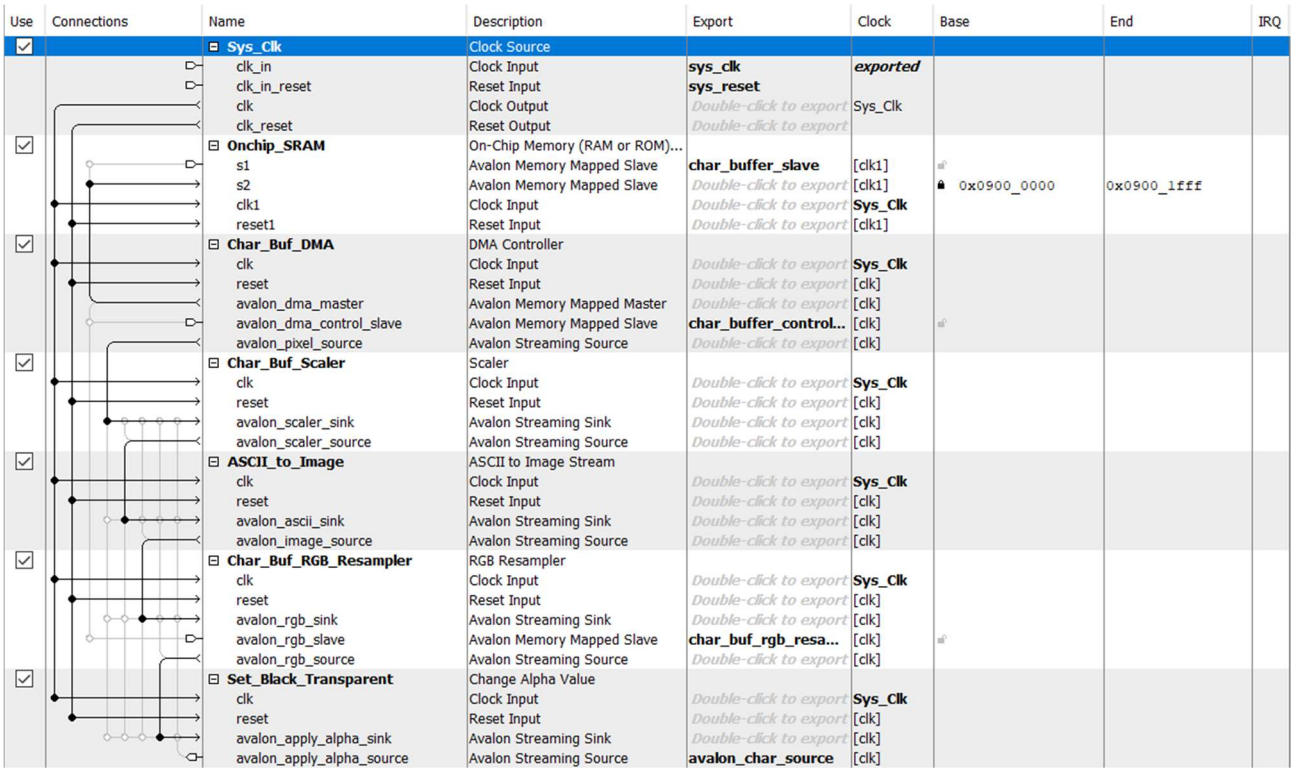


Fig. 4.4 – Istanziazione *Char_Buf_Subsystem*.

In questo sottosistema vengono utilizzati i seguenti componenti:

- *Clock source* (chiamato"*Sys_Clk*")
- *On-Chip Memory* (RAM)
- *DMA Controller*
- *Scaler*
- *ASCII to Image Stream*
- *RGB Resampler*
- *Change Alpha Value*

4.4.1 Clock source

Viene utilizzato per distribuire il segnale di clock e di reset a tutti i componenti

4.4.2 On-Chip Memory (RAM)

Questa On-Chip Memory contiene il *Character Buffer*.

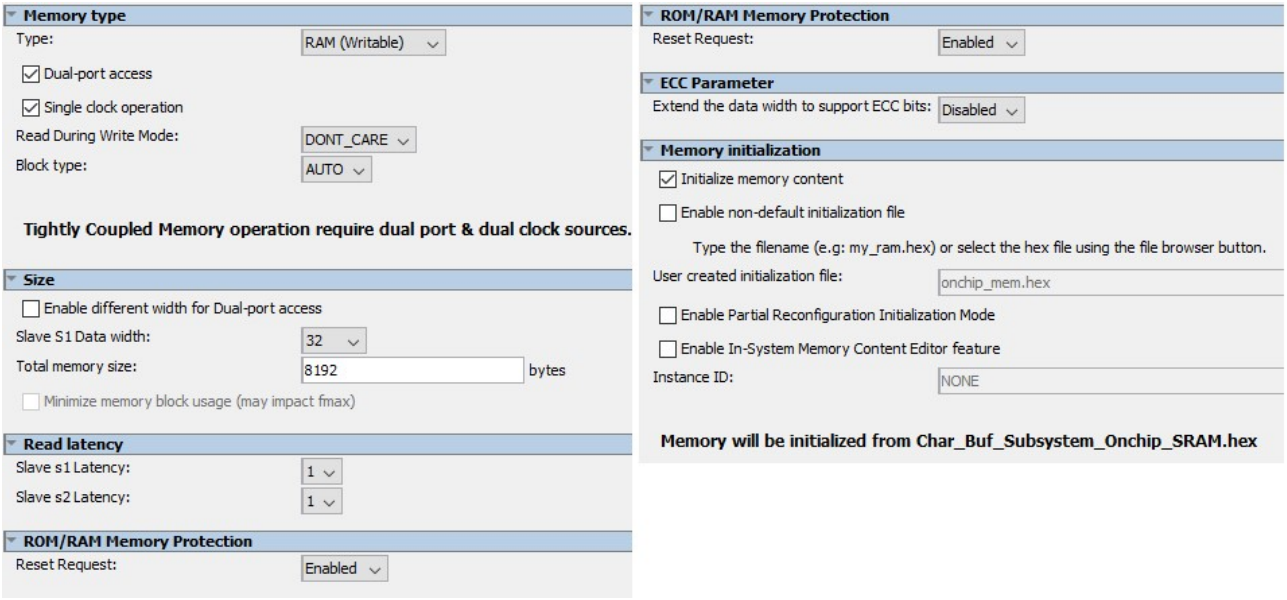


Fig. 4.4.2 – Configurazione della *On-Chip RAM* contenente il *Character Buffer*

4.4.3 DMA Controller

Questo *DMA Controller* immagazzina e legge frame da e verso il *Character Buffer*.

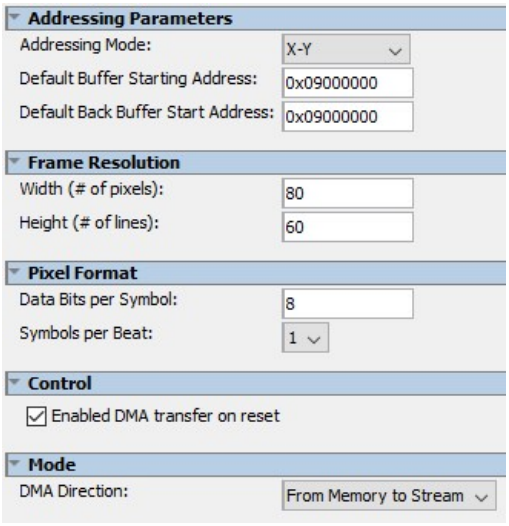


Fig. 4.4.3 – Configurazione del *DMA Controller*.

4.4.4 Scaler

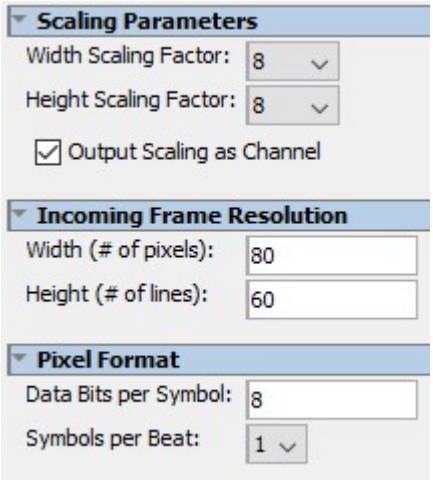


Fig. 4.4.4 – Configurazione dello *Scaler*.

4.4.5 ASCII to image stream

Questo componente renderizza un flusso di caratteri ASCII in ingresso in una rappresentazione grafica da mostrare a schermo. Il formato dei caratteri utilizzato è 1-bit Black/White: i caratteri verranno disegnati in bianco con uno sfondo trasparente. La risoluzione in uscita da questo blocco deve essere quella finale del display.

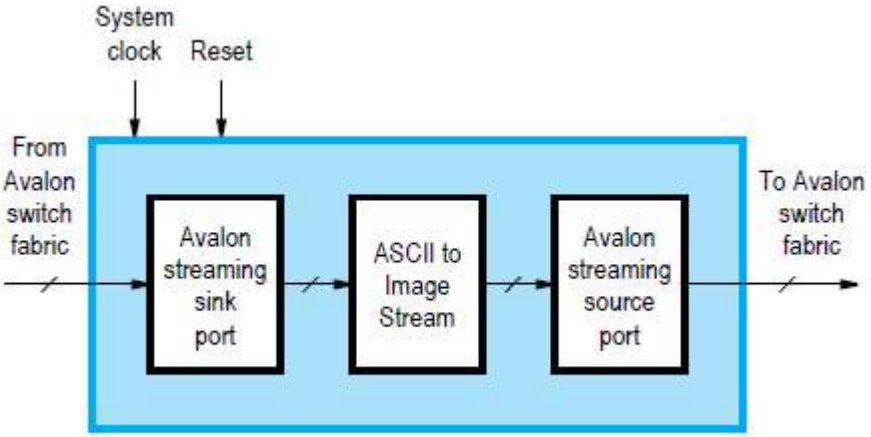


Fig. 4.4.5 –Schema a blocchi del ASCII to image stream

4.4.6 RGB Resampler

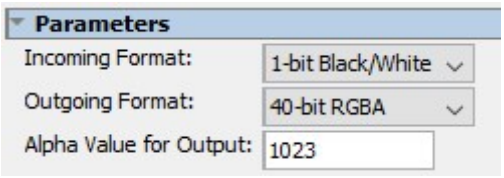


Fig. 4.4.6 –Configurazione del *RGB Resampler*

4.4.7 Change Alpha Value

Questo componente rimpiazza il valore alpha per tutti i pixel di un certo colore. Ciò è utile per utilizzare un colore per rappresentare la trasparenza.

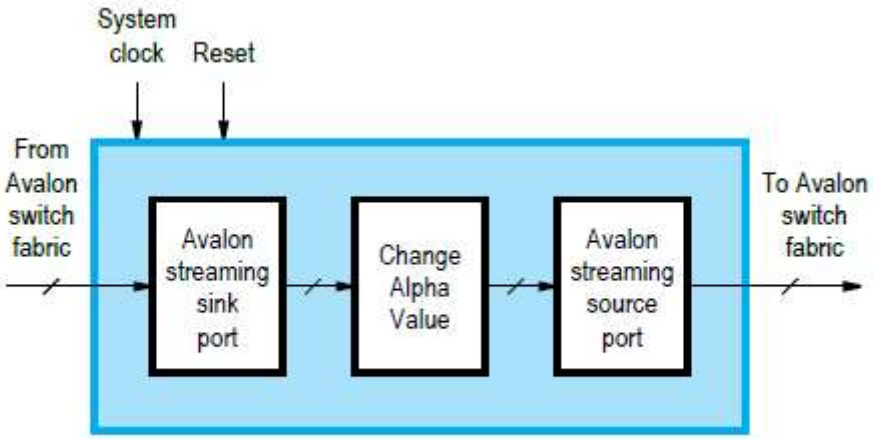


Fig. 4.4.7.1 –Schema a blocchi del *Change Alpha Value*.

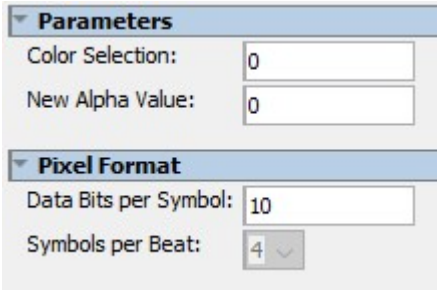


Fig. 4.4.7.2 – Configurazione del *Change Alpha Value*.

5 Implementazione del gioco

5.1 Macchina a stati

Il codice del gioco consiste principalmente in una macchina a stati con cinque stati:

- *RESET*: è lo stato di partenza. Premendo KEY_0 da qualsiasi altro stato si torna in questo stato;
- *GAME*: è lo stato in cui è possibile giocare;
- *PAUSE*: premendo KEY_1 dallo stato di gioco è possibile mettere il gioco in pausa;
- *WIN*: è lo stato in cui il gioco va nel caso di vittoria della partita;
- *LOSE*: se il giocatore perde il gioco va in questo stato.

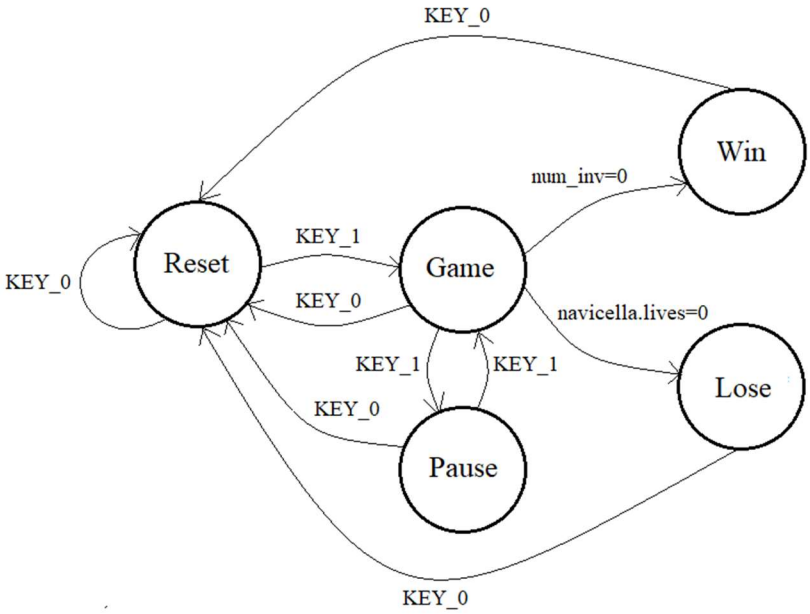


Fig. 5.1 – Schema della macchina a stati

5.2 Istruzioni HAL API

Nel codice sorgente sono presenti un gran numero di istruzioni *HAL API*. Questa scelta porta *Pro* e *Contro*:

Pro:

- Sono “Bug free”;
- E’ più veloce sviluppare il codice;
- Tolleranza ai cambiamenti hardware durante lo sviluppo del software;
- Facilita lo sviluppo parallelo del software

Contro:

- Codice poco ottimizzato: performance peggiori e spazio in memoria occupato maggiore

Le istruzioni *HAL API* utilizzate sono le seguenti:

- *alt_irq_register()*: viene utilizzata per registrare un ISR;
- *alt_up_video_dma_draw_box():* permette di disegnare un rettangolo;
- *alt_up_video_dma_open_dev():* apre il dispositivo *DMA Controller* specificato*;*
- *alt_up_accelerometer_spi_open_dev():* apre il dispositivo accelerometro specificato;
- *alt_up_video_dma_screen_clear():* pulisce l'intero schermo;
- *alt_up_video_dma_draw_string():* permette di scrivere una stringa di testo a schermo;
- *alt_up_video_dma_ctrl_check_swap_status()*: controlla se è stata completata l'operazione di swap fra il frontbuffer ed il backbuffer;
- *alt_up_accelerometer_spi_read_x_axis():* legge i valori dei registri dell' accelerometro relativi all'asse x e li converte in un intero con segno;
- *alt_up_video_dma_ctrl_swap_buffers():* esegue l'operazione di swap;
- *ALT_UP_VIDEO_RESAMPLE_RGB_24BIT_TO_16BIT():* converte il formato di un colore da 24 bit a 16 bit.

L'*Hardware Abstraction Layer* definisce anche strutture per i device:

- *alt_up_video_dma_dev*;
- *alt_up_accelerometer_spi_dev*.

5.3 Stato GAME

Di seguito viene mostrato in pseudocodice la struttura dello stato GAME:

```
while(1)
{
…
        Se è stata completata l'operazione di swap dei buffer
        {
                1. Pulizia del Backbuffer
                2. Elaborazione del movimento degli invasori
                3. Elaborazione del movimento dei proiettili
                4. Elaborazione del movimento della navicella
                5. Disegno degli elementi sul Backbuffer
                6. Viene eseguito il comando di swap dei buffer
        }
…

}
```

6 Analysis & Synthesis

In **Fig. 6** vengono riportati i risultati dell'Analisi e Sintesi del sistema embedded implementato.

Analysis & Synthesis Status	Successful - Fri May 29 14:19:22 2020
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	New_Computer_Space_Invaders
Top-level Entity Name	New_Computer_Space_Invaders
Family	MAX 10
Total logic elements	4,632
Total registers	2646
Total pins	63
Total virtual pins	0
Total memory bits	1,142,912
Embedded Multiplier 9-bit elements	0
Total PLLs	2
UFM blocks	0
ADC blocks	0

Fig. 6 - Risultati *Analysis & Synthesis*.

6.1 Fitter

Nella **Fig. 6.1** sono presenti i dati relativi al fitting sulla FPGA MAX10.

Fitter Status	Successful - Fri May 29 14:21:04 2020
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	New_Computer_Space_Invaders
Top-level Entity Name	New_Computer_Space_Invaders
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	4,377 / 49,760 (9 %)
Total registers	2680
Total pins	63 / 360 (18 %)
Total virtual pins	0
Total memory bits	1,142,912 / 1,677,312 (68 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	2 / 4 (50 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

Fig 6.1 - Risultati *Fitter*

6.2 Analisi di Timing

Si riportano nel seguito i risultati ottenuti dalle simulazioni timing effettuate sull'architettura implementata. La simulazione fa riferimento al caso in cui:

- Vcc = 1.2V;
- Temp = 85°C.

	Clock Name	Type	Period	Frequency	Rise	Fall	Duty Cycle	Divide by	Multiply by	Phase
1	CLOCK2_50	Base	20.000	50.0 MHz	0.000	10.000				
2	CLOCK_50	Base	20.000	50.0 MHz	0.000	10.000				
3	The_System|system_pll|sys_pll|PLL_for_DE_Series_Boards|auto_generated|pll1|clk[0]	Generated	10.000	100.0 MHz	0.000	5.000	50.00	1	2	
4	The_System|system_pll|sys_pll|PLL_for_DE_Series_Boards|auto_generated|pll1|clk[1]	Generated	10.000	100.0 MHz	7.000	12.000	50.00	1	2	-108.0
5	The_System|video_pll|video_pll|PLL_for_DE_Series_Boards|auto_generated|pll1|clk[1]	Generated	40.000	25.0 MHz	0.000	20.000	50.00	2	1	

Fig. 6.2.1 – *Clock*.

	Fmax	Restricted Fmax	Clock Name
1	108.24 MHz	108.24 MHz	The_System|system_pll|sys_pll|PLL_for_DE_Series_Boards|auto_generated|pll1|clk[0]
2	152.49 MHz	152.49 MHz	The_System|video_pll|video_pll|PLL_for_DE_Series_Boards|auto_generated|pll1|clk[1]

Fig. 6.2.2 – *Fmax*

Si vede che la frequenza massima utilizzabile è maggiore di quella da noi impostata, è un buon risultato in quanto non si riscontrano problematiche con i tempi di campionamento.

	Clock	Slack
1	The_System|system_pll|sys_pll|PLL_for_DE_Series_Boards|auto_generated|pll1|clk[0]	0.761
2	The_System|video_pll|video_pll|PLL_for_DE_Series_Boards|auto_generated|pll1|clk[1]	33.442

Fig. 6.2.3 – *Slack* [ns] sui tempi di setup

	Clock	Slack
1	The_System|system_pll|sys_pll|PLL_for_DE_Series_Boards|auto_generated|pll1|clk[0]	0.293
2	The_System|video_pll|video_pll|PLL_for_DE_Series_Boards|auto_generated|pll1|clk[1]	0.342

Fig. 6.2.4 – *Slack* [ns] sui tempi di hold.

7 Ottimizzazione

Come ultimo step, si è deciso di ottimizzare il codice scritto mediante l'uso del *BSP Editor*, una GUI che permette di settare diversi parametri ed in particolare sono state effettuate diverse prove utilizzando i possibili flag nella voce *bsp_cflags_optimization*.

Questo flag ci permette di ottimizzare il codice verso due principali direzioni:

- *Speed*: permette appunto di incrementare la velocità di esecuzione impostando i flag 1, 2 e 3 (grado di ottimizzazione crescente)
- *Size*: permette di sfruttare al meglio le risorse a disposizione in modo da occupare meno spazio; questo avviene impostando il flag *s*.

Inoltre sono state utilizzate la *Small newlib C Library* e versioni dei *Device Driver* più leggere.

Sono state fatte diverse prove al fine di individuare i miglioramenti passando da un flag all'altro.

Di default il flag è impostato su *–O0*, quindi nessuna ottimizzazione; in tal caso il gioco risulta essere poco fluido; impostando invece il flag su *–O1*, -O2 , –O3 e -OS invece si nota un sensibile aumento della fluidità del gioco.

Utilizzando la libreria *Small newlib C*, versioni dei *Device Driver* più leggere ed impostando il flag su -OS si ha che la dimensione del programma (codice e dati inizializzati) è di 19KB e si ha uno spazio libero di 65515 KB per stack ed heap.

8 Conclusioni

A conclusione del lavoro svolto diamo qualche spunto per possibili sviluppi futuri del gioco:

- Sostituire le navicelle con delle immagini più complesse
- Inserimento di suoni
- Introduzione di velocità diverse in modo tale da velocizzare il movimento degli invasori, complicando così il gioco
- Introduzione del punteggio all'interno del gioco

Questo ci ha permesso di poterci dividere i compiti e conseguentemente di migliorare la descrizione e la leggibilità dei vari blocchi in modo da facilitare la collaborazione. Le problematiche insorte durante lo sviluppo del gioco ci hanno permetto non solo di acquisire conoscenza nella progettazione di un computer embedded, ma anche di migliorare la nostra capacità di analisi e risoluzione di problemi più o meno complessi.

La versione del gioco è una versione *Lite* ma è un ottimo esempio di cosa sia possibile fare con schede programmabili quali la DE10-Lite.

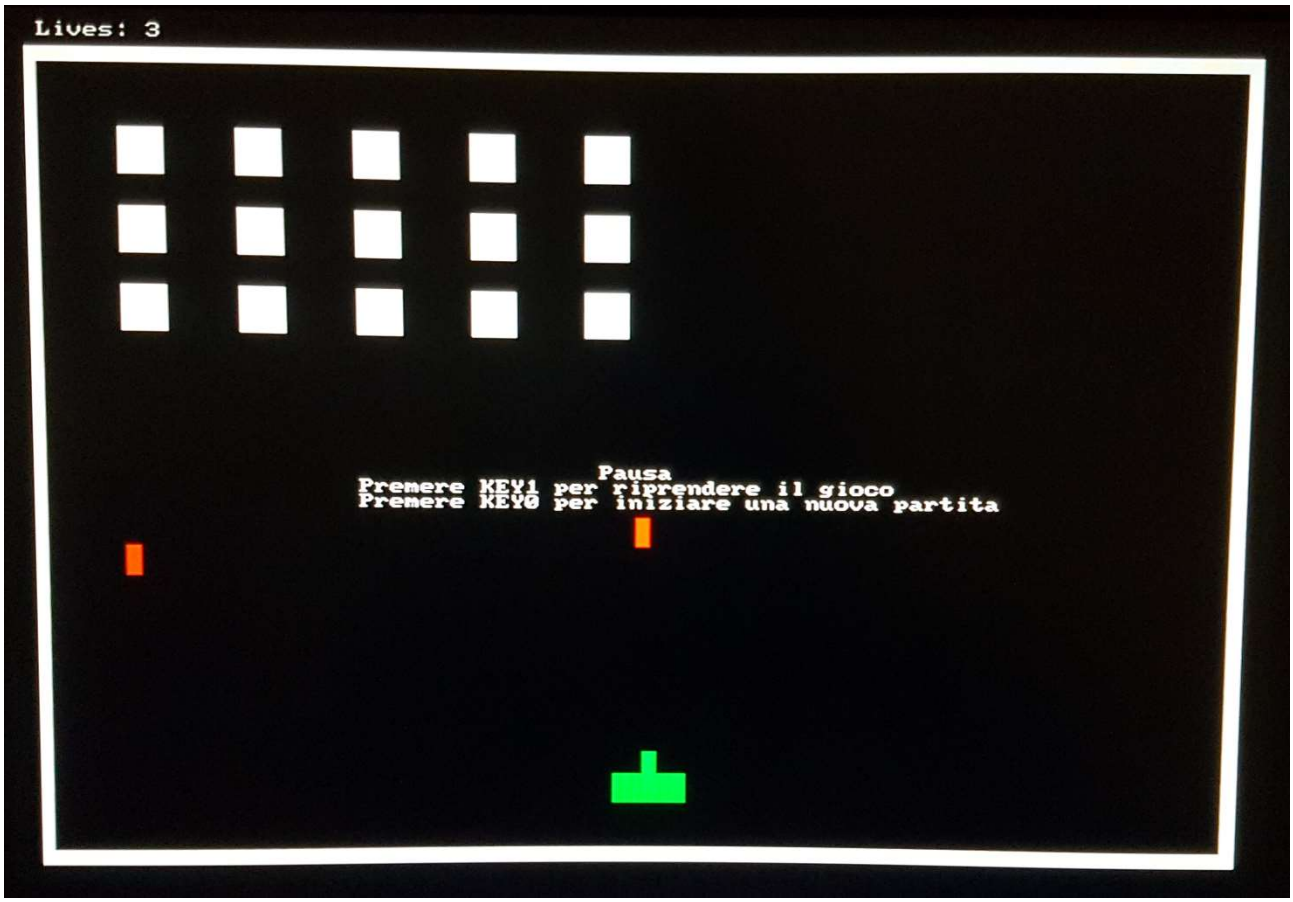


Fig. 8 – Schermata stato *Pausa* del gioco.

9 References

DE10-Lite_User_Manual

Accelerometer SPI Mode Core for DE-Series Boards

Digital Accelerometer ADXL345

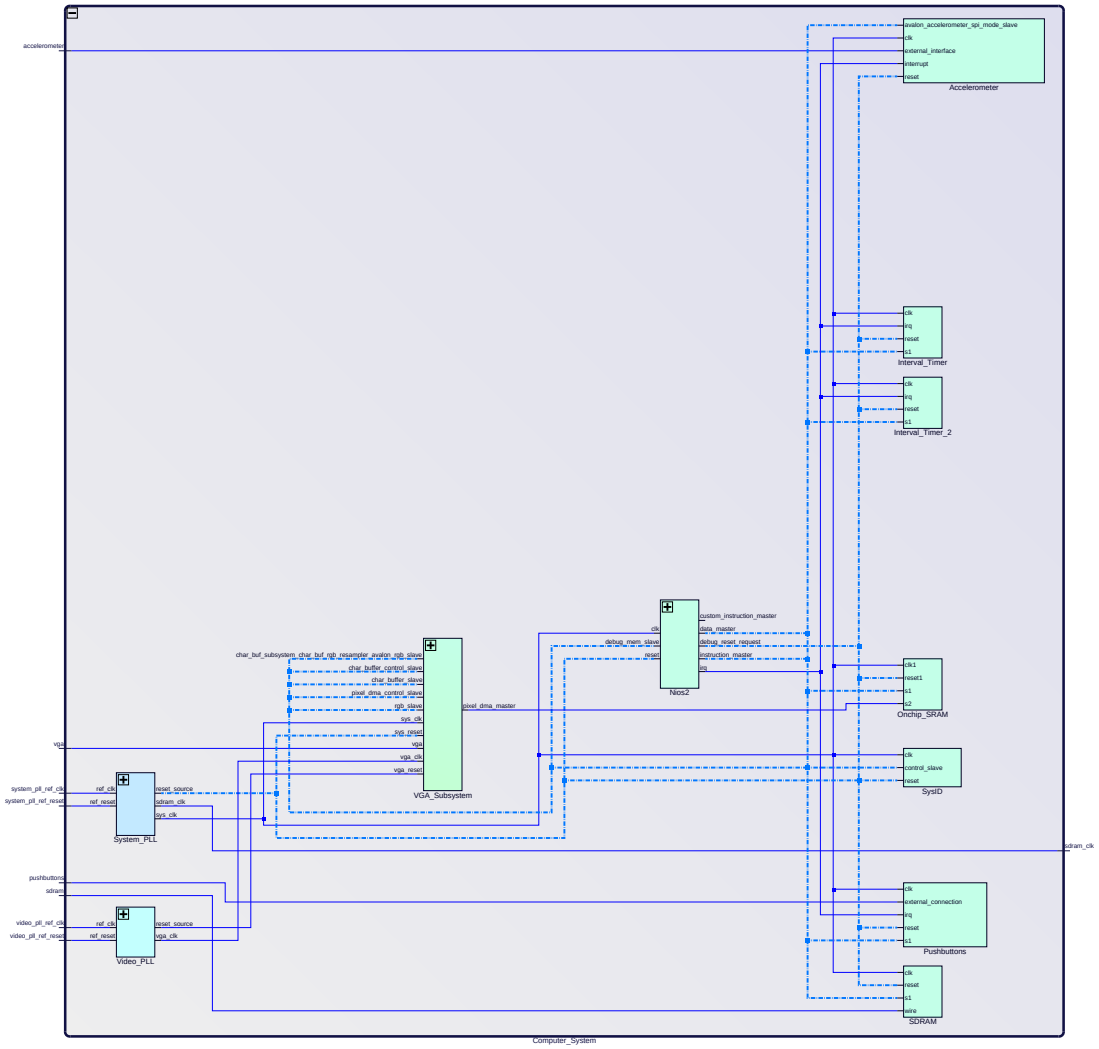
DE10-Lite Computer System

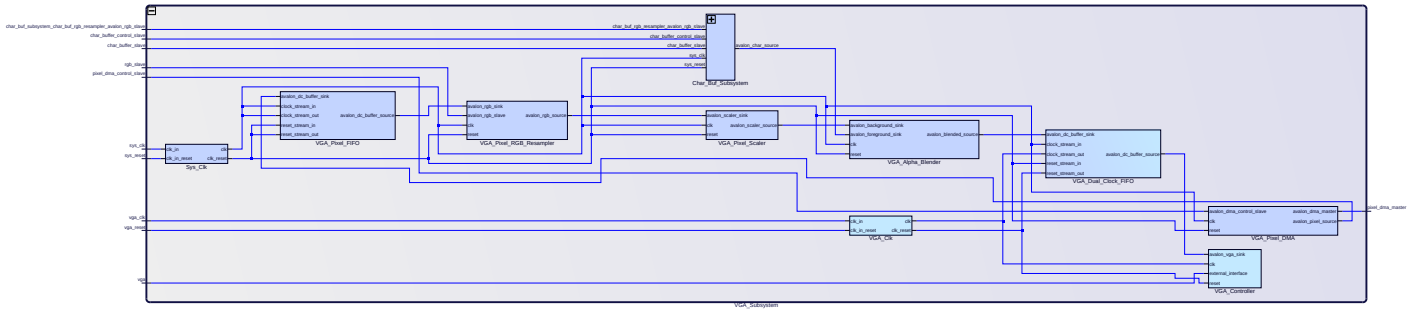
Video IP Cores for Intel DE-Series Boards

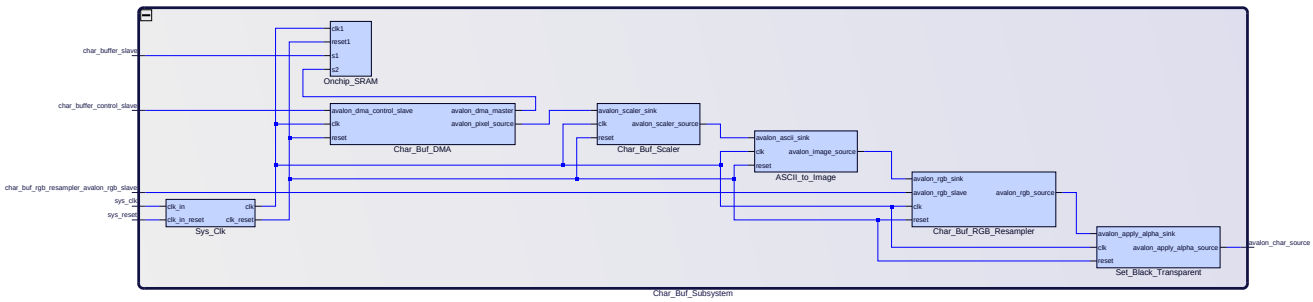
Clocks for Altera DE-Series Boards

Embedded peripherals IP User Guide

IS42S16320F-7TLI Scheda dati


accelerometer
avalon_accelerometer_spi_mode_slave
clk
external_interface
interrupt
reset
Accelerometer
clk
irq
reset
s1
Interval_Timer
Interval_Timer_2
custom_instruction_master
data_master
debug_reset_request
instruction_master
irq
Nios2
clk1
reset1
s1
s2
Onchip_SRAM
control_slave
SysID
char_buf_subsystem_char_buf_rgb_resampler_avalon_rgb_sink
char_buffer_control_slave
char_buffer_slave
pixel_dma_control_slave
rgb_slave
sys_clk
sys_reset
vga
vga_clk
vga_reset
pixel_dma_master
VGA_Subsystem
vga
system_pll_ref_clk
system_pll_ref_reset
ref_clk
ref_reset
reset_source
sdram_clk
sys_clk
System_PLL
sdram_clk
pushbuttons
sdram
external_connection
Pushbuttons
video_pll_ref_clk
video_pll_ref_reset
vga_clk
Video_PLL
reset
wire
SDRAM
Computer_System


char_buf_subsystem_char_buf_rgb_resampler_avalon_rgb_slave
char_buffer_control_slave
char_buffer_slave
rgb_slave
pixel_dma_control_slave
sys_clk
sys_reset
vga_clk
vga_reset
vga_out
char_buf_rgb_resampler_avalon_rgb_slave
char_buffer_control_slave
char_buffer_slave
avalon_char_source
bits_clk
sys_reset
Char_Buf_Subsystem
avalon_dc_buffer_sink
clock_stream_in
clock_stream_out
avalon_dc_buffer_source
reset_stream_in
reset_stream_out
VGA_Pixel_FIFO
avalon_rgb_sink
avalon_rgb_slave
clk
reset
avalon_rgb_source
VGA_Pixel_RGB_Resampler
avalon_scaler_sink
clk
reset
avalon_scaler_source
VGA_Pixel_Scaler
avalon_background_sink
avalon_foreground_sink
clk
reset
avalon_blended_source
VGA_Alpha_Blender
avalon_dc_buffer_sink
clock_stream_in
clock_stream_out
avalon_dc_buffer_source
reset_stream_in
reset_stream_out
VGA_Dual_Clock_FIFO
clk_in
clk_in_reset
clk
clk_reset
Sys_Clk
clk_in
clk_in_reset
clk
clk_reset
VGA_Clk
avalon_dma_control_slave
clk
reset
avalon_dma_master
avalon_pixel_source
VGA_Pixel_DMA
avalon_vga_sink
clk
external_interface
reset
VGA_Controller
pixel_dma_master
VGA_Subsystem


char_buffer_slave
char_buffer_control_slave
char_buf_rgb_resampler_avalon_rgb_slave
sys_clk
sys_reset
clk1
reset1
s1
s2
Onchip_SRAM
avalon_dma_control_slave
clk
reset
avalon_dma_master
avalon_pixel_source
Char_Buf_DMA
avalon_scaler_sink
clk
reset
avalon_scaler_source
Char_Buf_Scaler
avalon_ascii_sink
clk
reset
avalon_image_source
ASCII_to_Image
avalon_rgb_sink
avalon_rgb_slave
clk
reset
avalon_rgb_source
Char_Buf_RGB_Resampler
avalon_apply_alpha_sink
clk
reset
avalon_apply_alpha_source
Set_Black_Transparent
avalon_char_source
clk_in
clk_in_reset
clk
clk_reset
Sys_Clk
Char_Buf_Subsystem