ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA
DIPARTIMENTO DISI

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA
IN
CALCOLATORI ELETTRONICI L-M

Progetto di un sistema per il trasferimento in DMA di flussi video su piattaforma Zynq

CANDIDATO:
Andrea Romanelli

RELATORE:
Ing. Stefano Mattoccia

CORRELATORI:
Prof. Ing. Giovanni Neri
Dott. Marco Casadio

Anno Accademico 2012/2013

Sessione II

INDICE

1 - INTRODUZIONE

Sia in ambito industriale sia in quello scientifico, hanno larga diffusione i sistemi di visione in grado di ricostruire una scena 3D a partire da informazioni bidimensionali; l'importanza di questo tipo di sistemi è data dalla molteplicità di applicazioni nelle quali essi possono essere sfruttati. Nello specifico, di particolare interesse, è la visione stereoscopica, in cui la ricostruzione della scena tridimensionale avviene a partire da immagini provenienti da due (o più) sensori di immagine.

Il seguente lavoro di tesi si colloca all'interno di un progetto accademico molto amplio e complesso, finalizzato alla realizzazione di un sistema di visione stereo dotato di due sensori d'immagine, le cui immagini in output sono elaborate direttamente su scheda da dispositivi elettronici programmabili (FPGA) e, poi, inviate ad un *host* che ha il compito di acquisire e visualizzare i dati ricevuti. Il sistema è, attualmente, implementato su scheda Xilinx Spartan-6 e prevede la comunicazione con un calcolatore attraverso una connessione USB 2.0.

L'obiettivo della tesi è di eliminare la separazione tra l'elaborazione su FPGA e la visualizzazione su Host. Per fare questo è stata utilizzata la Zedboard (Zynq Evaluation & Development Board), una scheda Xilinx basata sul componente Zynq-7000. Tale dispositivo è costituito da una parte di logica programmabile (FPGA Xilinx di Serie 7) fortemente accoppiata con un processore ARM dual-core (Cortex-A9). Proprio grazie a questa caratteristica innovativa è stato possibile realizzare un sistema che permette la gestione di uno (o più) flussi video, siano questi immagini provenienti direttamente da

sensori o il risultato di una pipeline di elaborazione. In particolare, il sistema si occupa di memorizzare i frame di ogni sequenza in memoria, realizzando di fatto un frame buffer circolare. Tale frame buffer sarà quindi disponibile per il processore ARM, che estrarrà e visualizzerà ogni singolo frame.

Il progetto si articola in due fasi ben distinte: nella prima, è stata sviluppata la parte hardware mappata poi su FPGA e responsabile della gestione dei flussi video; successivamente è stato scritto il codice C che si occupa della programmazione di alcuni componenti del sistema, oltre all'estrazione dei frame dalla memoria. Questi due livelli di programmazione sono stati realizzati utilizzando l'ambiente di sviluppo ISE Design Suite 14.4 offerto da Xilinx che fornisce sia gli strumenti per la programmazione della FPGA (Xilinx Platform Studio) sia il tool (Software Development Kit) necessario all'implementazione del software C eseguito dal processore.

2 - ARCHITETTURA ZYNQ-7000

La scheda impiegata per lo sviluppo del progetto, denominata ZedBoard [1], è basata sul componente Zynq-7000 All Programmable SoC e ha, inoltre, un insieme di periferiche utilizzabili per le operazioni di input ed output. I circuiti integrati della famiglia Zynq-7000 offrono elevata flessibilità e scalabilità, caratteristiche tipiche di una FPGA, unite alla potenza di calcolo di un sistema basato su processore ARM. Tutti i dispositivi della famiglia sono quindi costituiti da due parti:

- **Processing System (PS)**: comprende quattro blocchi principali che sono l'Application Processor Unit (APU), l'interfaccia con la memoria, le periferiche di I/O (IOP) e le interconnessioni tra le varie parti

- **Programmable Logic (PL):** parte del circuito dedicata alla realizzazione della logica custom; il numero di celle di logica riconfigurabile disponibili dipende dalla fascia del dispositivo utilizzato

L'integrazione di queste due parti su singolo chip permette di avere delle soluzioni con prestazioni non ottenibili attraverso una realizzazione equivalente ma basata su due chip distinti; non è, infatti, possibile avere le stesse prestazioni per quanto riguarda i tempi di latenza, la larghezza di banda per operazioni di I/O e i consumi energetici. Queste due parti sono alimentate

separatamente, permettendo eventualmente la disattivazione di una sola delle due per effettuare della manutenzione o modifiche al sistema.

L'inclusione del APU rende anche possibile l'impiego di un sistema operativo, il più utilizzato è sicuramente Linux, ma non è l'unico disponibile.

Il dispositivo della famiglia presente nella scheda utilizzata è lo Zynq-7020 che rappresenta una soluzione di fascia intermedia, sia a livello di costi sia di prestazioni.

Rappresentazione ad alto livello dello Zynq-7020 e i dispositivi presenti sulla ZedBoard:

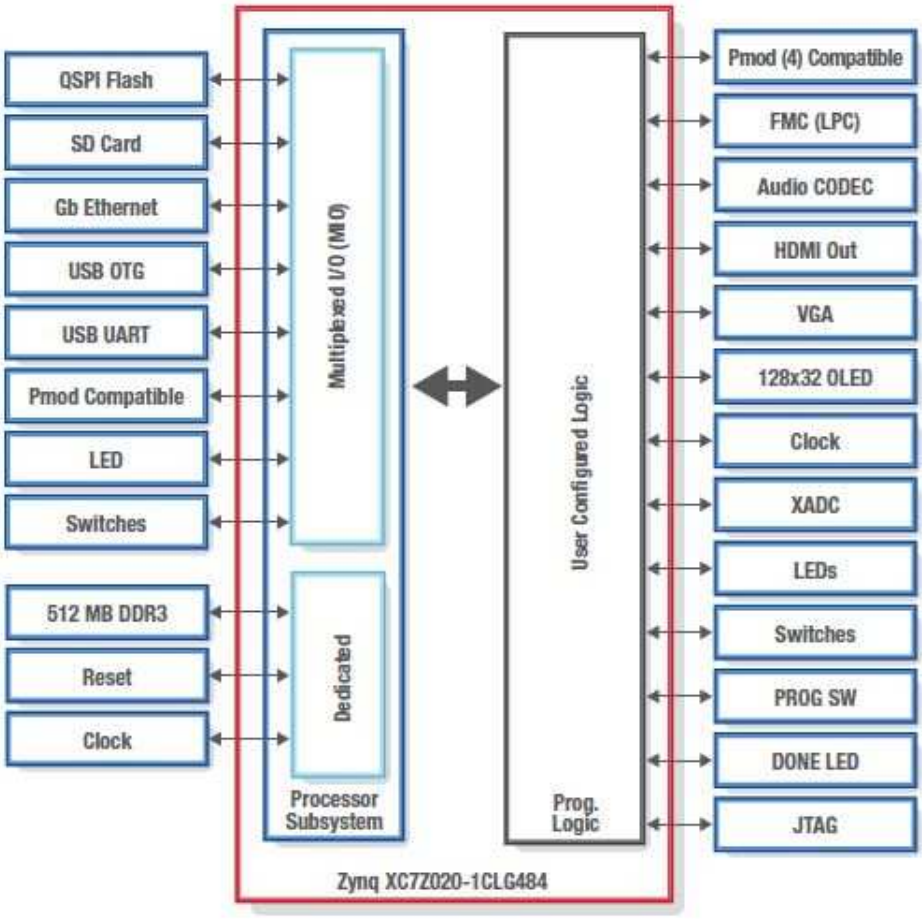


Fig.1: Schema a blocchi della ZedBoard

La presenza di tutti questi componenti rende la scheda molto flessibile e adatta ad ogni esigenza poiché permette di configurare a piacimento le risorse da utilizzare per l'applicazione specifica.

2.1 - Processing System

Come anticipato, il Processing System è costituito da quattro blocchi:

- Application Processor Unit
- Memory Interface
- I/O Peripherals
- Interconnect

In figura è presentato uno schema dell'architettura ZYNQ:

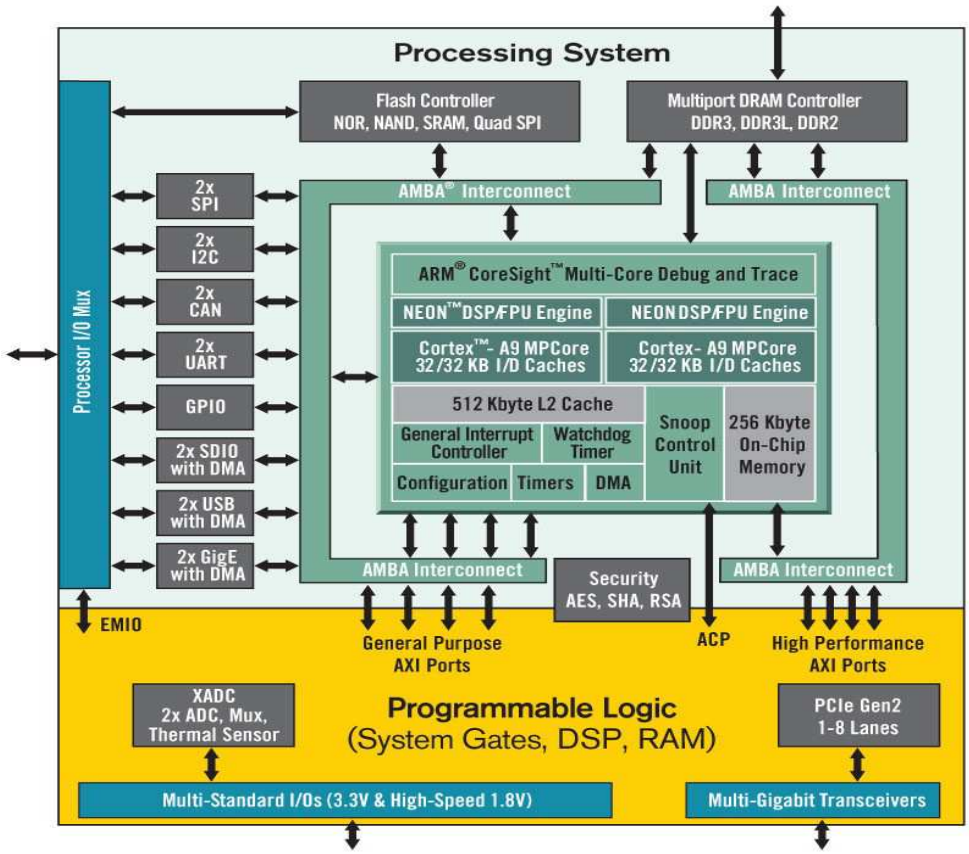


Fig.2: Architettura ZYNQ [2]

Per lo Zynq-7020 il processore è un ARM dual core Cortex-A9, in grado di operare su un singolo core oppure in modalità multiprocessore simmetrica o asimmetrica. L'ARM presente sulla scheda può lavorare con una frequenza massima di 667 Mhz ed è in grado di eseguire istruzioni con una velocità di 2,5 DMIPS/Mhz (Milioni di istruzione per Mhz con benchmark Dhrystone). Ogni core dispone di unità a virgola mobile e NEON media processing engine, il quale permette di supportare operazioni di tipo Single Instruction Multiple Data (SIMD). I due core dispongono ognuno di cache di primo livello separate per dati e istruzioni, di dimensione 32 KByte l'una e di tipo set associative a 4 vie; la cache di secondo livello è invece unificata, ha grandezza 512 KByte ed è di tipo set associative a 8 vie.

Nel chip è presente, inoltre, una RAM da 256 KB, accessibile sia dai processori sia dalla logica programmabile e progettata per le applicazioni in cui è richiesto un basso tempo di latenza da parte della CPU.

L'ARM contiene al suo interno anche un Interrupt Controller e un DMA; quest'ultimo è in grado di gestire fino ad 8 canali e diversi tipologie di trasferimenti: *memory-to-memory, memory-to-peripheral e peripheral-to-memory*.

Per quanto riguarda le interfacce con le memorie, lo Zynq dispone di un controller per memorie dinamiche integrato capace di supportare i formati DDR3, DDR2 o LPDDR2, oltre ad un insieme di controllori flash che gestiscono la memorizzazione in fase di avvio e di configurazione. In particolare, lo Zynq-7020 dispone di 512 MB di DDR3 (parallelismo 32 bit) e 256 MB di QSPI Flash.

Il dispositivo utilizzato include anche numerose periferiche [3] di I/O che sono:

- Due Gigabit Ethernet
- Due USB On-The-Go
- Due CAN 2.0
- Due Interfacce I2C
- Due SD/SDIO 2.0
- Due porte SPI full-duplex
- Due UART
- Fino a 118 bit per I/O di tipo General Purpose

Queste periferiche comunicano con i dispositivi esterni attraverso 54 pin multiuso (MIO), la cui configurazione può essere effettuata in modo dinamico e flessibile anche se ogni periferica ha a disposizione solo alcuni banchi di pin da poter utilizzare; chiaramente questi pin non risultano sufficienti nel caso in cui si vogliano sfruttare tutte le periferiche a disposizione. La maggior parte delle periferiche sono accessibili anche dalla logica programmabile (EMIO), una volta effettuata la corretta configurazione. In figura 3 è mostrato lo schema delle connessioni tra IOP, MIO e EMIO.

Le parti fino ad ora descritte (APU, IOP e memory interfaces) comunicano tra loro attraverso lo standard ARM AMBA AXI [4], capace di gestire connessioni multiple Master-Slave. Grazie a questo insieme d'interfacce, Xilinx ha mappato i registri delle periferiche in memoria, anche per quanto riguarda i dispositivi implementati nella logica programmabile, rendendo la comunicazione tra PS e PL possibile solo attraverso questo standard.

Lo Zynq supporta uno spazio di indirizzamento della dimensione di 4 GB (32 bit), organizzati nel modo seguente:

Start Address	Size (MB)	Descrizione
0x0000_0000	1,024	DDR e memoria on-chip
0x4000_0000	1,024	Porta 0 PL Slave AXI
0x8000_0000	1,024	Porta 1 PL Slave AXI
0xE000_0000	256	IOP
0xF000_0000	128	Riservati
0xF800_0000	32	Registri programmabili accessibili tramite bus AMBA APB
0xFA00_0000	32	Riservati
0xFC00_0000	64 MB - 256 KB	Quad-SPI Flash
0xFFFC_0000	256 KB	Memoria on-chip quando è mappata negli indirizzi alti

Tab.1: Spazio indirizzamento Zynq

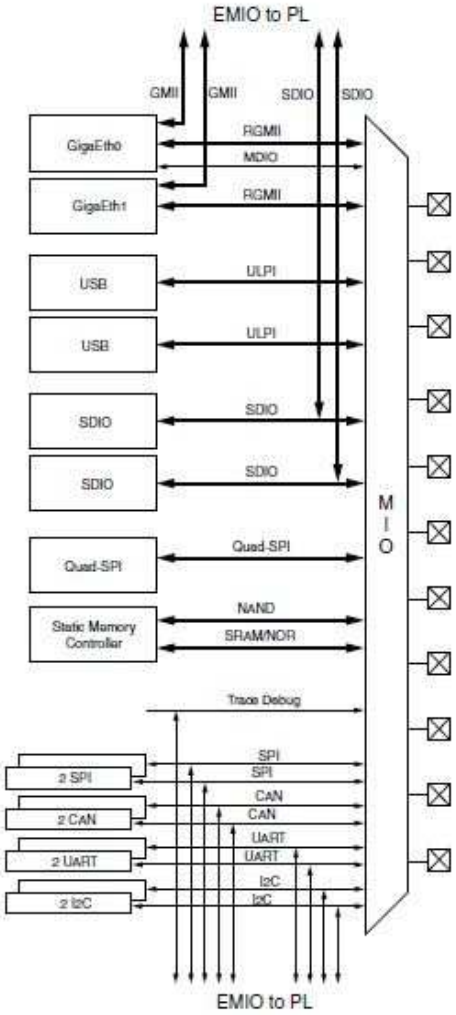


Fig.3: Schema delle connessione tra IOP, MIO e EMIO [3]

2.2 - PL e connessioni con PS

La parte di logica riconfigurabile appartiene alla serie-7 della Xilinx e le risorse disponibili variano tra i diversi dispositivi della famiglia; Di seguito sono riportate le principali caratteristiche [3] relative ad ogni dispositivo della famiglia Zynq:

Zynq-7000 All Programmable SoC							
	Device Name	**Z-7010**	**Z-7015**	**Z-7020**	**Z-7030**	**Z-7045**	**Z-7100**
	Part Number	**XC7Z010**	**XC7Z015**	**XC7Z020**	**XC7Z030**	**XC7Z045**	**XC7Z100**
Programmable Logic	Xilinx 7 Series Programmable Logic Equivalent	Artix®-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Kintex®-7 FPGA	Kintex-7 FPGA	Kintex-7 FPGA
	Programmable Logic Cells (Approximate ASIC Gates)(3)	28K Logic Cells (~430K)	74K Logic Cells (~1.1M)	85K Logic Cells (~1.3M)	125K Logic Cells (~1.9M)	350K Logic Cells (~5.2M)	444K Logic Cells (~6.6M)
	Look-Up Tables (LUTs)	17,600	46,200	53,200	78,600	218,600	277,400
	Flip-Flops	35,200	92,400	106,400	157,200	437,200	554,800
	Extensible Block RAM (# 36 Kb Blocks)	240 KB (60)	380 KB (95)	560 KB (140)	1,060 KB (265)	2,180 KB (545)	3,020 KB (755)
	Programmable DSP Slices (18x25 MACCs)	80	160	220	400	900	2,020
	Peak DSP Performance (Symmetric FIR)	100 GMACs	200 GMACs	276 GMACs	593 GMACs	1,334 GMACs	2,622 GMACs
	PCI Express® (Root Complex or Endpoint)	—	Gen2 x4	—	Gen2 x4	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs					
	Security(2)	AES and SHA 256b for Boot Code and Programmable Logic Configuration, Decryption, and Authentication					

Tab.2: Caratteristiche FPGA famiglia Zynq-7000

Come è possibile notare dalla tabella precedente le differenze in termini di risorse disponibili risultano notevoli, si passa da un minimo di 28.000 celle logiche a un massimo di circa 150.000, con un rapporto quindi di uno a cinque. Differenze significative tra le varie versioni possono essere osservate analizzando le Look-Up Table, i flip-flop e le block RAM.

Alla PL è collegato un oscillatore che fornisce in ingresso un clock a 100 Mhz, utilizzabile eventualmente per la generazione dei clock necessari

all'applicazione specifica; alternativamente è possibile utilizzare uno dei clock che il Processing System genera con le proprie PLL, a partire da un altro clock esterno a 33.3 MHz.

I pin fisici della PL vengono suddivisi in cinque banchi, tre sono alimentati a 3.3V mentre per gli altri due è possibile scegliere, attraverso un jumper, se alimentarli a 2.5V o 1.8V.

Questa suddivisione in banchi permette di alimentare correttamente le varie periferiche connesse alla PL:

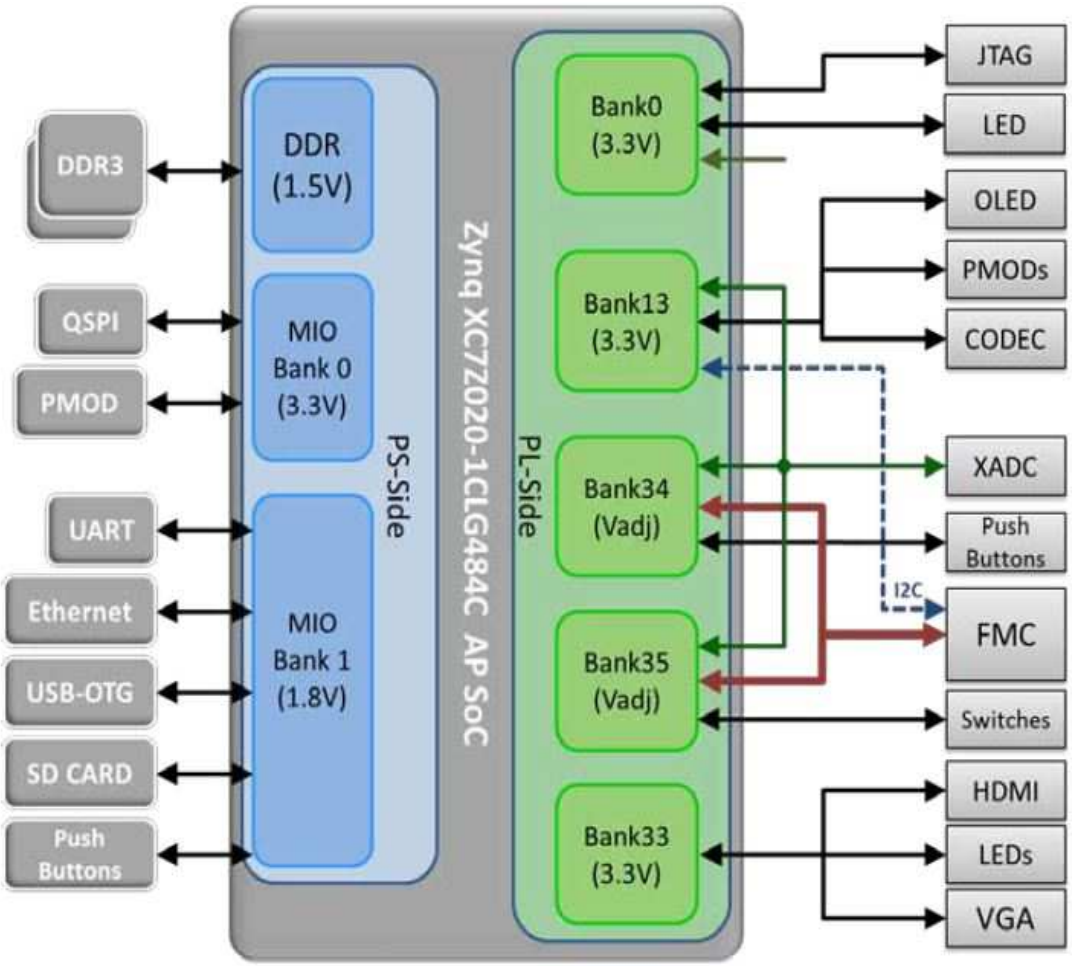


Fig.4: Periferiche e banchi fisici dello Zynq-7020 nella ZedBoard

Per il momento PS e PL sono stati considerati in maniera separata ma i vantaggi principali dello Zynq risiedono proprio nella capacità di

comunicazione tra queste parti. Innanzitutto il PS rende disponibili quattro segnali di clock e quattro di reset liberamente utilizzabili nella PL, gli riserva 4 canali DMA e la possibilità di sfruttare le proprie periferiche di I/O non utilizzate.

Tra PS e PL sicuramente le interfacce di comunicazione più significative sono quelle basate sullo standard AXI. In particolare, vi sono:

- due interfacce AXI Master General Purpose a 32 bit
- due interfacce AXI Slave General Purpose a 32 bit
- quattro porte AXI slave High-Performance a 32 o 64 bit configurabili che hanno accesso diretto alla DDR e alla memoria on-chip
- una porta AXI a 64 bit (ACP) per aggiungere eventualmente degli acceleratori coerenti con la cache

Di seguito è riportata una panoramica dello Zynq, nella quale sono state evidenziate le interfacce AXI presenti tra PS e PL:

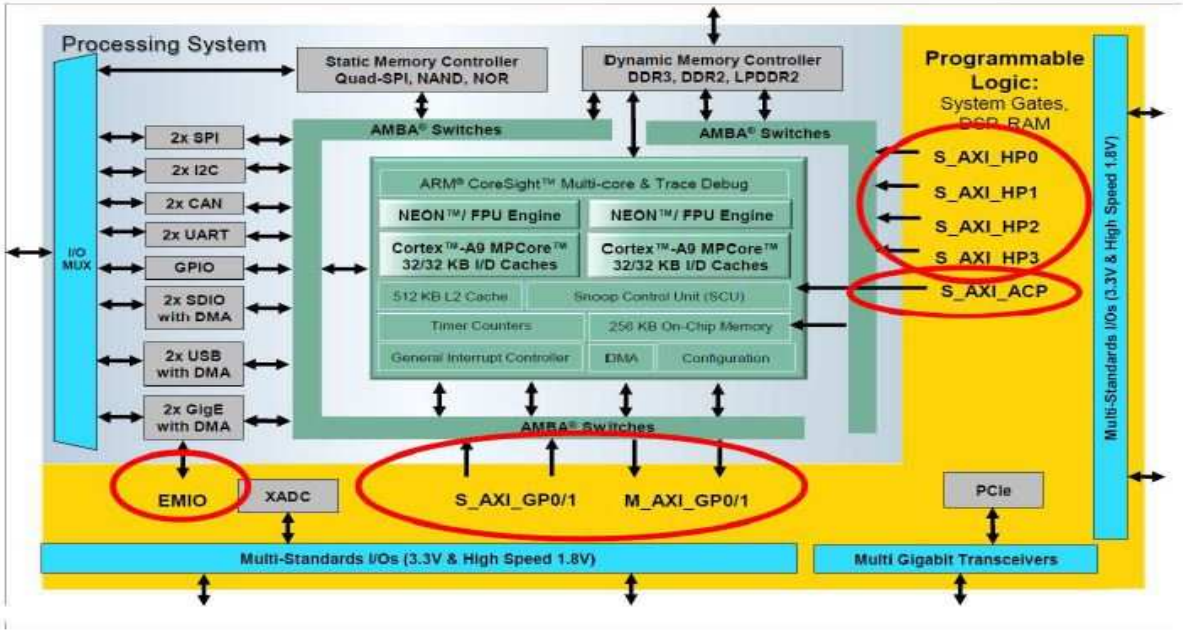


Fig.5: porte AXI tra PS e PL

3 - STRUMENTI DI SVILUPPO

Il progetto è stato completamente sviluppato utilizzando i potenti strumenti del framework Xilinx Embedded Development Kit 14.4 (EDK) che, come suggerisce il nome, raccoglie tutti i software utili alla progettazione di sistemi embedded. EDK fa parte di Xilinx ISE Design Suite e accorpa in se tutti i tools necessari allo sviluppo di sistemi basati su Zynq.

Considerando la struttura dello Zynq, è necessario effettuare sia la trattazione del sistema hardware, da mappare nella PL, sia lo sviluppo del software che l'ARM deve eseguire. Questo ambiente di sviluppo è stato realizzato proprio per facilitare lo progettazione delle varie parti di un sistema su Zynq, tramite strumenti che offrono supporto all'integrazione tra esse. In particolare, i tool impiegati per lo svolgimento del seguente lavoro di tesi sono stati:

- **Xilinx Platform Studio (XPS):** permette la configurazione completa dell'hardware dello Zynq, maggiori dettagli vengono forniti nella sezione 3.2
- **Software Development Kit (SDK):** offre gli strumenti necessari allo sviluppo del software, sia per applicazioni bare-metal sia per applicazioni Linux-based
- **Ise Project Navigator:** impiegato per la stesura del codice VHDL dei componenti necessari alla realizzazione del progetto
- **ISim:** strumento utile alla verifica, attraverso la simulazione, del comportamento di singoli componenti e parti del sistema

In questo capitolo sarà data una panoramica di quelle che sono le caratteristiche e le potenzialità dei software sopracitati, fornendo inoltre, alcuni brevi esempi di utilizzo; prima di procedere è necessario, però, introdurre il concetto di IP-Core, dato il ruolo centrale che assume nel seguente elaborato.

3.1 - IP-Core

Il termine “IP-Core” (Intellectual Property-Core) indica [5] un’entità logica riutilizzabile, impiegata sia nella creazione di logiche “statiche” (ASIC) sia nei circuiti riconfigurabili. In altre parole, un IP-Core è un’unità che, all’interno di un sistema hardware più ampio, svolge un determinato compito. Come esprime il termine stesso, un IP-Core è una proprietà intellettuale: vale a dire, che esso è una risorsa che può essere posseduta da un singolo o da una società, oppure può essere messa a disposizione di chiunque la voglia utilizzare (in questo caso si parla di open core). Una tipica classificazione degli IP-Core è la seguente:

- **Hard cores:** implementazioni fisiche su dispositivi a semiconduttore
- **Firm (semi-hard) cores:** progetti che contengono informazioni relative al proprio posizionamento ma possono anche essere esportate su varie architetture, non necessarimanete basate su logiche riconfigurabili
- **Soft cores:** definibili come una rete di porte logiche e registri, implementata spesso in un linguaggio di descrizione dell’hardware come il VHDL e mappabili su dispostivi riconfigurabili

Un IP-Core può essere sia un componente master, per esempio la stessa CPU, oppure un componente slave; esso può inoltre rappresentare entità come un bus o porta seriale, costituendo quindi una infrastruttura di comunicazione tra altri IP-Core. Nel seguito con il termine IP-Core si indicherà esclusivamente ai Soft cores, cioè quelli che possono essere istanziati su una FPGA.

Per essere inserito efficacemente all'interno di un sistema più grande, un core deve essere dotato di un'interfaccia verso l'esterno. Tale interfaccia permette di accedere al canale di comunicazione (per esempio un bus) e dialogare con gli altri componenti dell'architettura. Per questo motivo, con il termine IP-Core d'ora in poi si farà riferimento ad un blocco logico dotato di un *frontend ad hoc*, adatto ad essere inserito in un sistema che adotti l'infrastruttura di comunicazione per il quale esso è stato progettato. In pratica, un IP-Core è costituito da un "guscio" esterno che racchiude il core ed un'opportuna interfaccia di comunicazione, come mostrato in figura 6.

Un core, che segue il tipo di struttura sopra menzionata, prende, spesso, il nome di IP-Core user logic, proprio per sottolineare il fatto che è effettivamente utilizzabile dall'utente.

Xilinx offre un ampio catalogo di IP-Core, alcuni gratuiti ed altri a pagamento, ottimizzati per essere mappati sui propri dispositivi. Per la realizzazione di questo progetto si è cercato di sfruttare il più possibile tali componenti, sia per aumentare la velocità di sviluppo, sia per limitare al minimo l'utilizzo di risorse della PL.

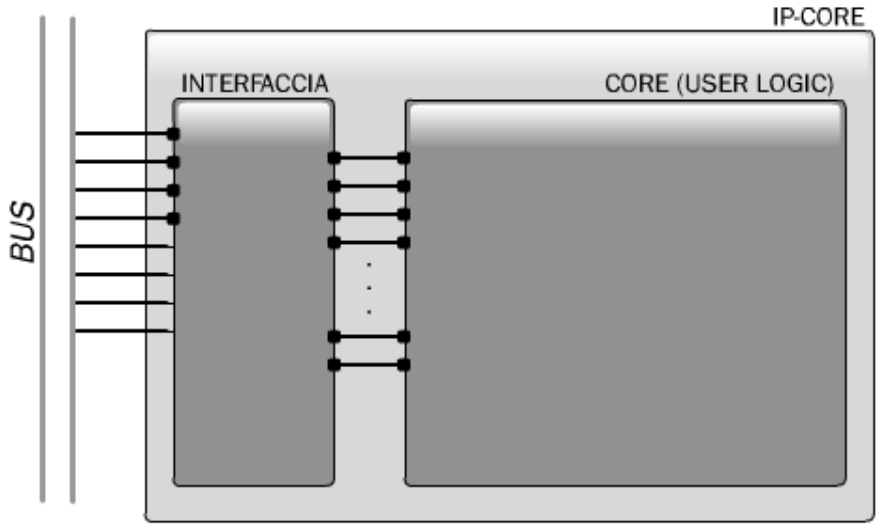


Fig.6: Struttura IP-Core

3.2 - XPS

La realizzazione di un sistema su Zynq richiede, per quanto riguarda la parte hardware, la creazione di una piattaforma embedded, costituita da uno (o più) processori e moduli collegati ad un (o più) bus; risulta perciò conveniente adottare un approccio alla progettazione di tipo modulare. Nell'ambiente di sviluppo questa piattaforma hardware è descritta in un file nel formato MHS (Microprocessor Hardware Specification).

Occorre, inoltre, realizzare una piattaforma software, cioè l'insieme dei driver per i moduli ed, eventualmente, di un sistema operativo per l'architettura creata; tale piattaforma viene descritta in un file MSS (Microprocessor Software Specification).

XPS è il tool Xilinx che offre la possibilità di generare o modificare i file MSS e MHS, di creare una visualizzazione schematica dell'architettura realizzata e accedere agli ulteriori tool che compongono la suite EDK; i principali sono:

- **Base System Builder (BSB)**: wizard che permette la facile e veloce creazione di un nuovo sistema
- **Platform Generator (Platgen):** permette di tradurre la descrizione della piattaforma in una netlist HDL che può essere implementata su una FPGA. In particolare invoca XST (illustrato in seguito) per la sintesi degli IP-Core utilizzati
- **Create and Import Peripheral Wizard:** supporta l'utente nella creazione dei propri IP-Core; ulteriori informazioni saranno fornite nella sezione 3.5

- **Library Generator (Libgen):** configura la piattaforma software, comprese le applicazioni che si vogliono utilizzare, gestendo il file MSS

Una caratteristica importante di EDK è che, nel costruire un'architettura collegando insieme vari IP-Core, esso non analizza il contenuto di ciascun componente, ma si interessa solamente dei segnali di Input/Output di ciascuno di essi permettendo, quindi, una gestione del sistema ad alto livello. Si analizza ora, in dettaglio, com'è possibile sfruttare XPS per creare, modificare e configurare un nuovo sistema d'elaborazione.

Questo strumento permette, innanzitutto, la configurazione del Processing System dello Zynq tramite un'interfaccia grafica molto semplice ed immediata, nella quale ne è presentata l'architettura:

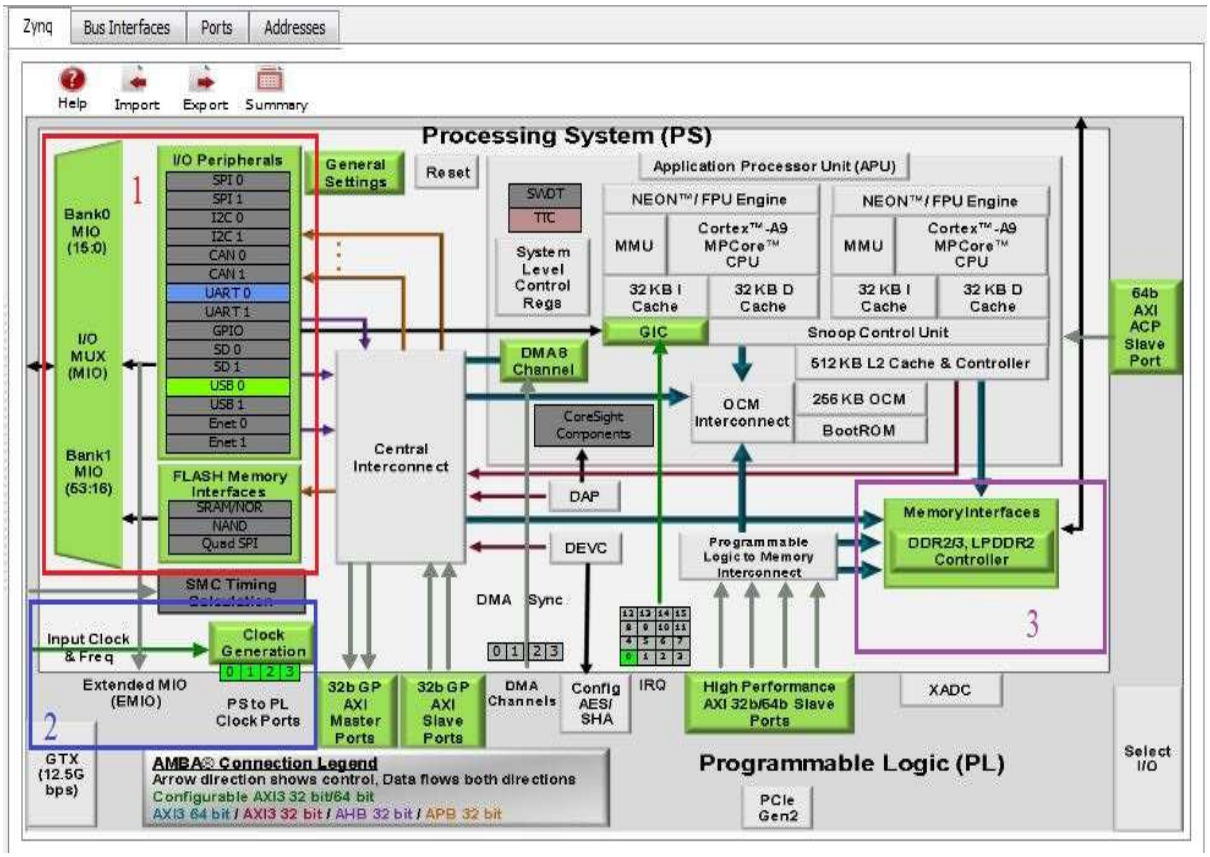


Fig.7: XPS Processing System Configuration

In figura 7 sono evidenziate (colore verde) le parti del PS configurabili dall'utente (illustrate nel capitolo precedente); tramite un doppio click su di esse si aprirà, infatti, la relativa finestra di configurazione.

Sono ora mostrate tali parti e le relative viste di configurazione. Si inizia presentando una panoramica della gestione delle periferiche di I/O, situate nella parte sinistra della vista (riquadro 1), e dei relativi collegamenti con i pin di I/O multiplexati; tale finestra è presentata in figura 8. Si può notare che, nella parte sinistra, sono presenti tutte le IOP, attivabili o meno attraverso le relative *checkbox*; attraverso un *menù a tendina*, situato subito dopo il nome della periferica, sono impostati i banchi di pin a cui essa è collegata. Nella parte destra è presente, invece, un riassunto di tutti i 54 pin fisici e le periferiche ad essi collegate.

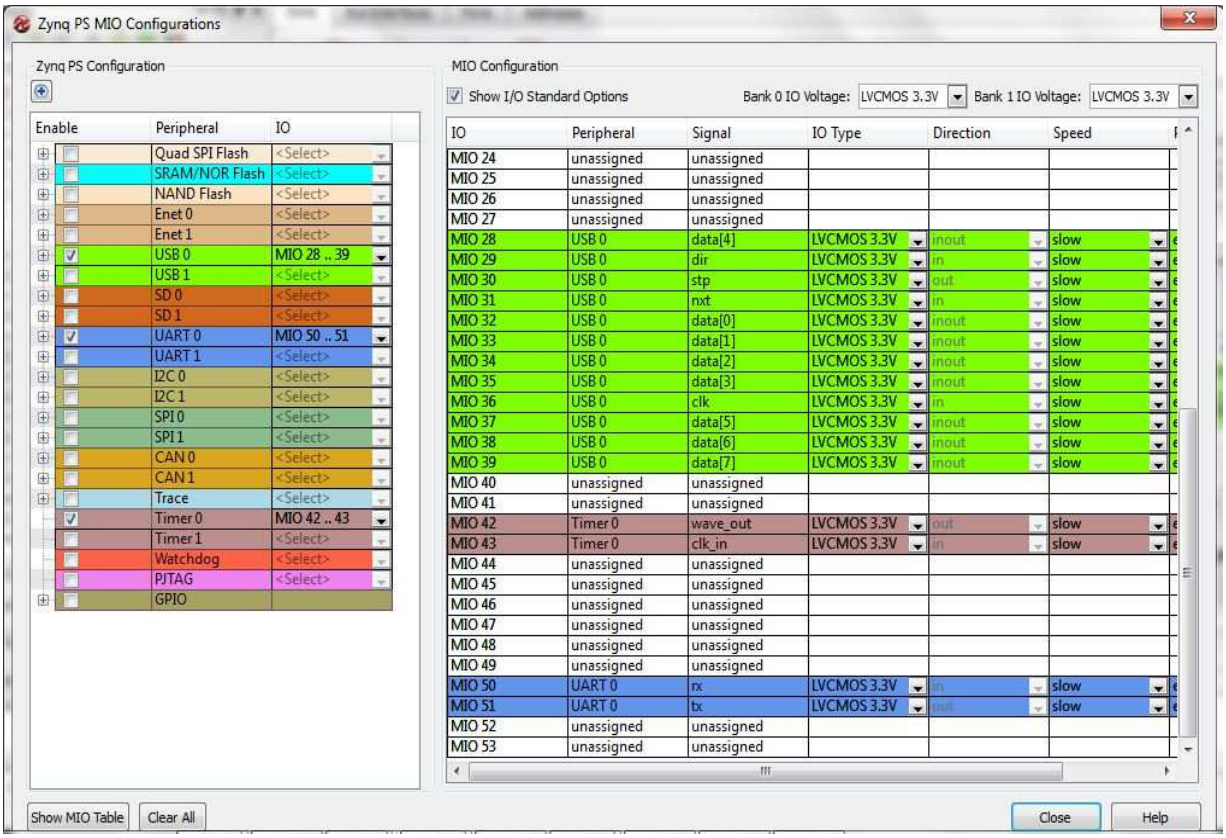


Fig.8: XPS Processing System IOP e MIO Configurations

Subito sotto le IOP sono presenti i generatori di clock (figura 7, riquadro 2), essi saranno utilizzati sia all'interno del PS, sia resi disponibili alla PL. La gestione di tali clock è realizzata, in maniera molto semplice ed immediata, tramite il *Clock Wizard* (figura 9). Si può notare che, tra i tanti clock impostabili, vi è anche il clock del processore e delle DDR.

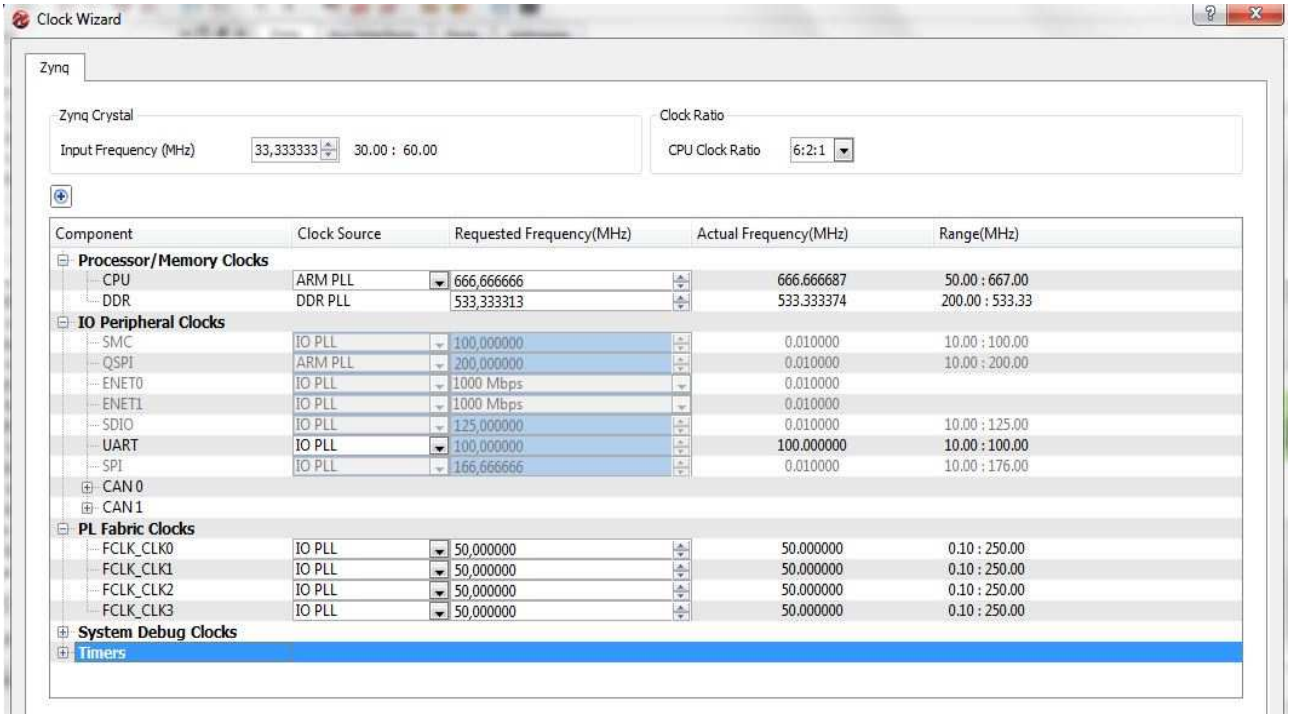


Fig.9: XPS Clock wizard

Le altre impostazioni relative alle DDR sono effettuate tramite la finestra (fig.11), aperta con un click sul relativo controller (fig.7, riquadro 3).

Le restanti parti attive aprono tutte la stessa vista, all'interno della quale sono presenti:

- Impostazioni generali dell' Application Processor Unit
- impostazioni del DMA
- porte AXI Master e Slave general purpose presenti tra PS e PL

- porte AXI Slave High-Performance
- porta ACP

Qui di seguito è riportata tale vista:

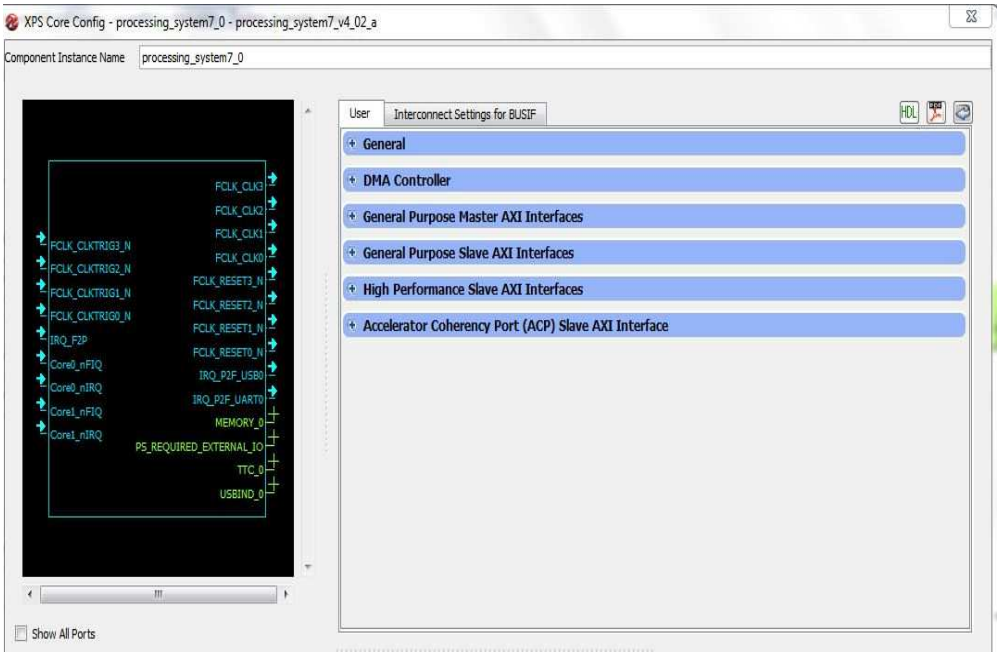


Fig.10: XPS PS Configurations

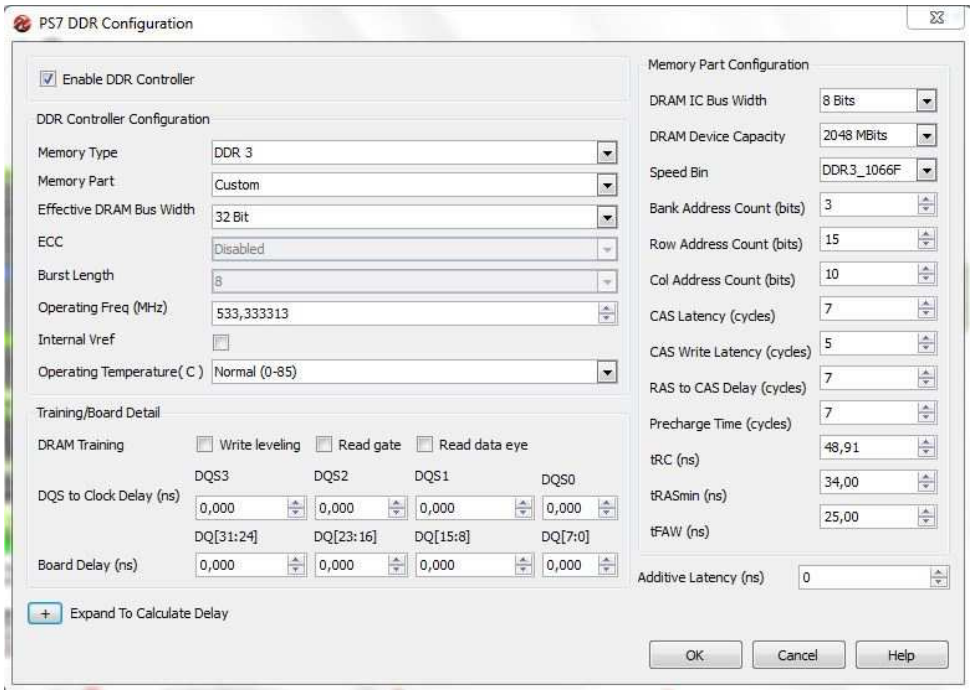


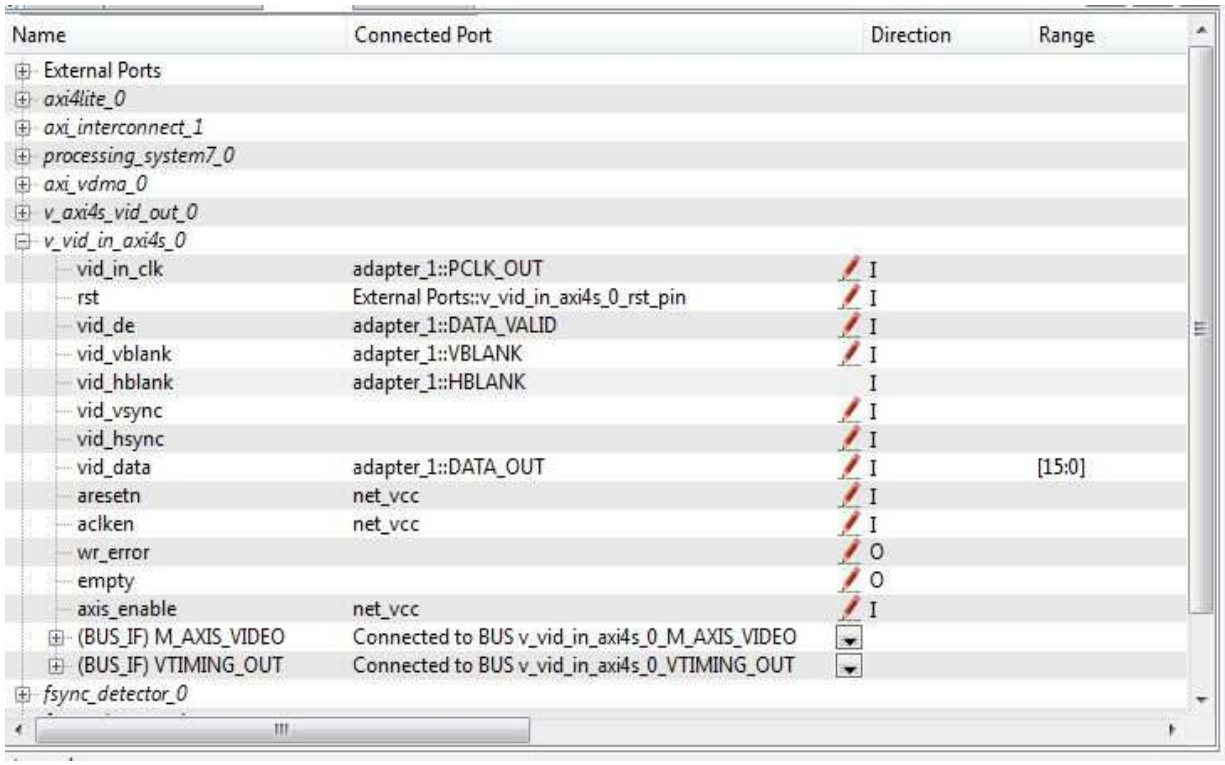
Fig.11: XPS DDR Configurations View

Terminata la configurazione del Processing System, è possibile passare alla realizzazione del sistema nella logica riconfigurabile; la composizione del sistema si effettua tramite una gestione ad alto livello, resa possibile dall'utilizzo di IP-Core.

Ogni componente aggiunto nel sistema (PS compreso) è quindi un IP-Core; di questi, alcuni sono messi a disposizione gratuitamente da Xilinx, altri implementabili in VHDL e resi IP-Core tramite lo strumento *Create and Import Peripheral Wizard.*

XPS prevede due fasi per la realizzazione del sistema, una nella quale aggiungere gli IP-Core e interfacciarli con i bus di sistema (AXI per lo Zynq), un'altra in cui collegare i restanti segnali dei componenti. Sempre in tale seconda fase sono impostati i segnali portati verso l'esterno, dichiarandoli di tipo *EXTERNAL*. L'aggiunta di un IP-Core avviene selezionandolo nella lista di quelli disponibili, trascinandolo poi nell'elenco dei componenti appartenenti al sistema. Se il tool riconosce, nel nuovo IP-Core, l'interfaccia ad un bus standard compatibile con uno di quelli già presenti nel sistema, richiede all'utente se lo vuole collegare ad esso. Ciò avviene anche, nel caso dello Zynq, con tutte le periferiche da connettere al processore, le quali presenteranno l'interfaccia AXI-Lite, illustrata meglio in seguito.

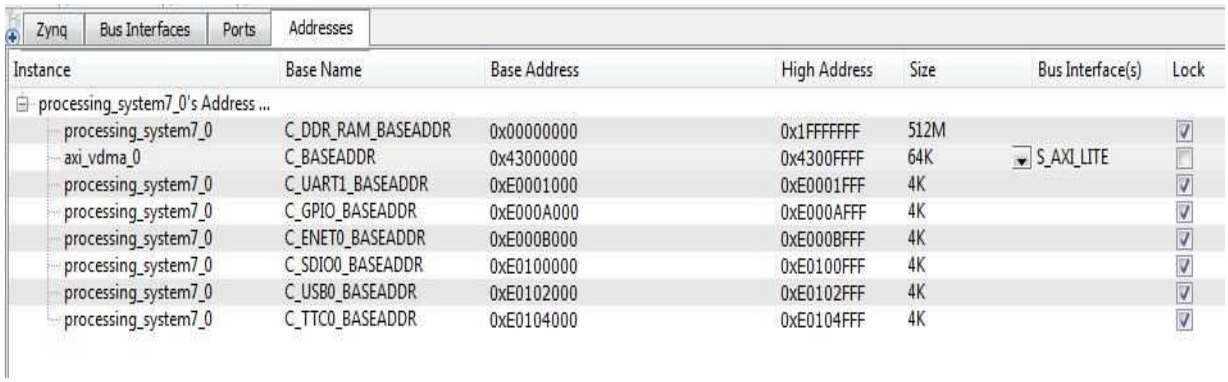
Successivamente, è possibile andare a connettere le restanti porte di ogni componente, creando così tutti i collegamenti necessari al corretto funzionamento della piattaforma d'elaborazione. Questa operazione avviene attraverso la seguente finestra, nella quale sono presenti tutti gli IP-Core e le relative interfacce verso l'esterno, come mostrato in figura 12.



Name	Connected Port	Direction	Range
External Ports			
axi4lite_0			
axi_interconnect_1			
processing_system7_0			
axi_vdma_0			
v_axi4s_vid_out_0			
v_vid_in_axi4s_0			
vid_in_clk	adapter_1::PCLK_OUT	I	
rst	External Ports::v_vid_in_axi4s_0_rst_pin	I	
vid_de	adapter_1::DATA_VALID	I	
vid_vblank	adapter_1::VBLANK	I	
vid_hblank	adapter_1::HBLANK	I	
vid_vsync		I	
vid_hsync		I	
vid_data	adapter_1::DATA_OUT	I	[15:0]
aresetn	net_vcc	I	
aclken	net_vcc	I	
wr_error		O	
empty		O	
axis_enable	net_vcc	I	
(BUS_IF) M_AXIS_VIDEO	Connected to BUS v_vid_in_axi4s_0_M_AXIS_VIDEO		
(BUS_IF) VTIMING_OUT	Connected to BUS v_vid_in_axi4s_0_VTIMING_OUT		
fsync_detector_0			

Fig.12: XPS IP-Core ports configuration

Un altro aspetto molto importante, gestibile grazie ad XPS, riguarda la scelta degli indirizzi, mappati in memoria, dei componenti del sistema. Ad ogni IP-Core aggiunto al progetto è assegnato un indirizzo di default, eventualmente modificabile a piacimento dal progettista. Il tool si occupa anche di verificare la compatibilità degli indirizzi dei vari dispositivi, non consentendo la sovrapposizione tra spazi d'indirizzamento. La situazione riassuntiva è presentata nel seguente modo:



Zynq | Bus Interfaces | Ports | Addresses

Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Lock
processing_system7_0's Address ...						
processing_system7_0	C_DDR_RAM_BASEADDR	0x00000000	0x1FFFFFFF	512M		☑
axi_vdma_0	C_BASEADDR	0x43000000	0x4300FFFF	64K	S_AXI_LITE	☐
processing_system7_0	C_UART1_BASEADDR	0xE0001000	0xE0001FFF	4K		☑
processing_system7_0	C_GPIO_BASEADDR	0xE000A000	0xE000AFFF	4K		☑
processing_system7_0	C_ENET0_BASEADDR	0xE000B000	0xE000BFFF	4K		☑
processing_system7_0	C_SDIO0_BASEADDR	0xE0100000	0xE0100FFF	4K		☑
processing_system7_0	C_USB0_BASEADDR	0xE0102000	0xE0102FFF	4K		☑
processing_system7_0	C_TTC0_BASEADDR	0xE0104000	0xE0104FFF	4K		☑

Fig.13: Processing System Address Table

Per poter generare il file *Bitstream*, utilizzato per la programmazione della PL, si deve, infine, realizzare il file *UCF (User Constraint File)*, il quale assegna i segnali degli IP-Core ai pin fisici dello Zynq (o più in generale di un FPGA). Per la realizzazione di tale compito XPS sfrutta lo strumento XST [7]; quest'ultimo crea una netlist descritta in un file NGC, effettuando una mappatura del sistema su una rete logica composta da blocchi combinatori, sequenziali e interconnessioni tra loro.

XPS offre, infine, la possibilità di esportare la piattaforma appena realizzata (file *.mss* e *.mhs*), rendendola così disponibile al tool di sviluppo del software (SDK), che si va ora ad illustrare.

3.3 - SDK

Il Software Development Kit è stato realizzato basandosi su Ecplise [8] ed offre il supporto, al progettista, per lo sviluppo e il debug di codice C o C++ per la piattaforma hardware specifica; offre la possibilità di realizzare applicazioni bare-metal (senza sistema operativo) e applicazioni Linux-based, sia per sistemi monoprocessore sia multiprocessore.

L'obiettivo di questo tool è quello di compilare il codice scritto dal progettista, includendo i driver delle periferiche presenti nel sistema ed eventuali altre librerie, al fine di creare un file in formato ELF (Executable Linked Format), direttamente eseguibile da parte del processore della piattaforma.

Il flusso di generazione del file ELF è mostrato nella figura seguente:

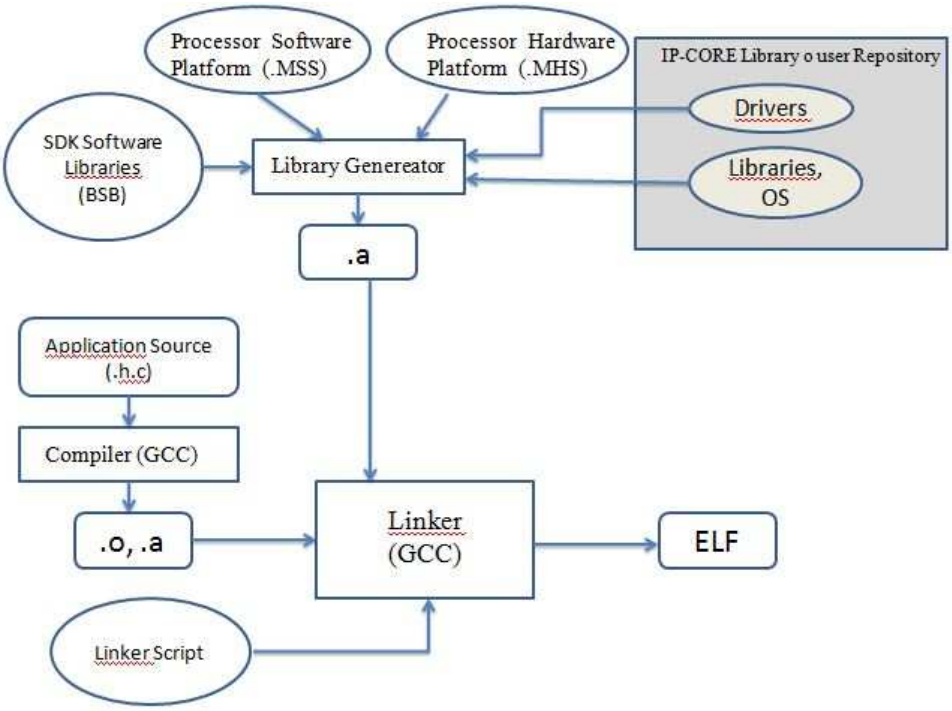


Fig.14: SDK ELF generation flow

Il processo di generazione del file ELF da parte di SDK sfrutta il tool Xilinx LibGen, oltre al compilatore e al linker GCC. LibGen impiega i file MHS e MMS (creati tramite XPS), i driver degli IP-Core, eventuali librerie e sistema operativo per la creazione del Board Support Package (BSP), utilizzabile per lo sviluppo del software. SDK, è in grado di creare il file ELF; ciò è possibile solo dopo aver generato, tramite wizard, l'opportuno Linker Script.

3.4 - ISE

Integrated Software Environment (ISE), anch'esso prodotto da Xilinx, è un insieme di tool che permette di effettuare il design e la sintesi di un'architettura descritta tramite linguaggi come il VHDL o il Verilog. Nel seguito si parlerà sempre di VHDL, poiché è il linguaggio utilizzato per l'implementazione dei componenti.

L'interfaccia grafica di questo pacchetto di software si chiama Project Navigator, tramite il quale è possibile accedere a tutte le funzionalità implementative offerte da ISE; tale interfaccia è mostrata nella figura seguente:

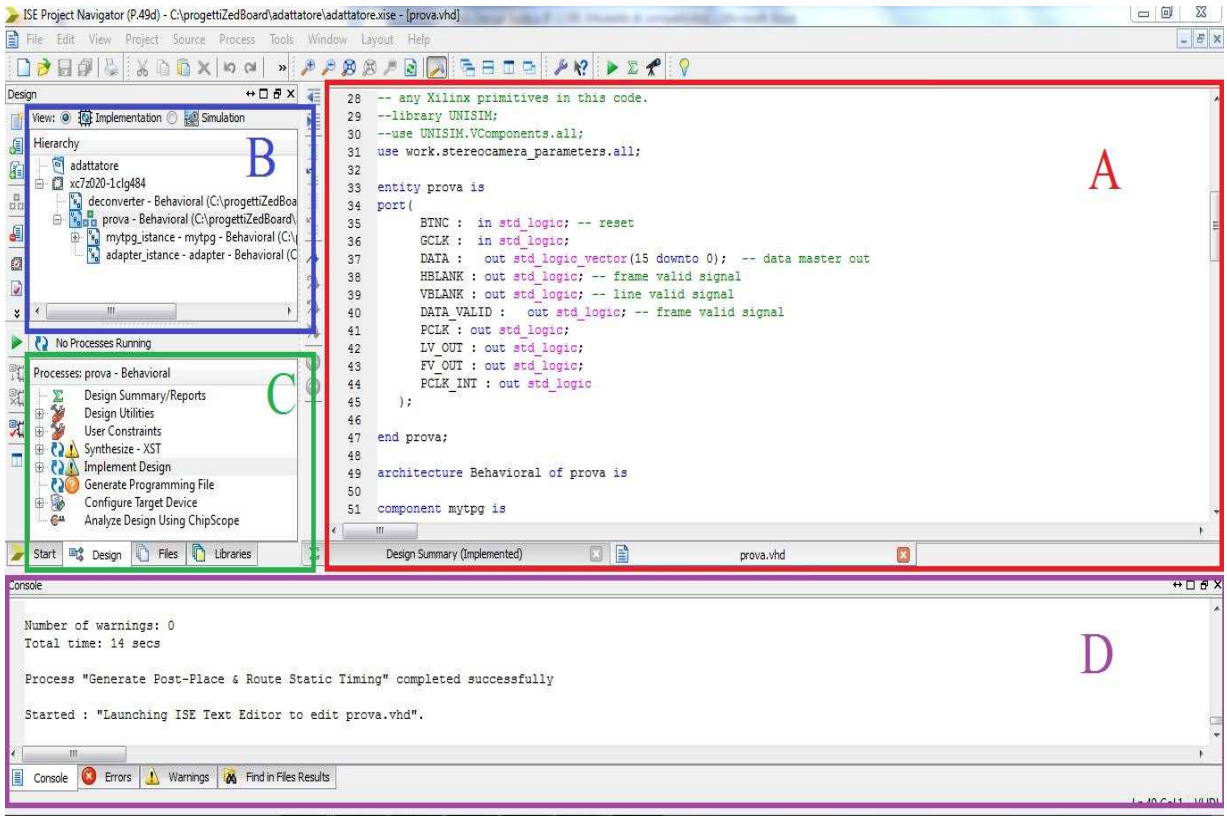


Fig.15: Interfaccia grafica Xilinx ISE

L’area di lavoro è divisa in quattro settori:

- un editor di testo (riquadro A), in cui è possibile editare i file sorgenti del progetto da sintetizzare. Sempre nella stessa finestra possono essere visualizzati i risultati di sintesi (Design Summary)
- una finestra (riquadro B) in cui è rappresentata la struttura del progetto, in termini di moduli e librerie utilizzate
- una finestra (riquadro C) da cui si possono lanciare le funzioni
- una shell (riquadro D) dove viene visualizzato il *log* delle operazioni effettuate dal programma

Le funzioni incluse in ISE sono divise in gruppi. Il primo gruppo (Design Utilities) comprende alcune utilità; si trovano poi le funzionalità *User Constraints*, che permettono di impostare dei vincoli prestazionali, di spazio e di tempo, al sistema creato. Il gruppo successivo è Synthesize - XST, necessario per compilare e sintetizzare il codice VHDL sfruttando il tool XST. Il processo di sintesi è estremamente complesso, e ciò è dovuto in buona parte alla ottimizzazioni spaziali (area occupata) e temporali (*maximum delay time*) effettuate. Sempre nell’ambito di sintesi XST si trova View RTL Schematic, che fornisce una visualizzazione schematica dei blocchi logici che costituiscono il sistema. Un altro gruppo di funzionalità è Implement Design, che mappa la rete risultante dal processo di sintesi su di una FPGA, il cui modello e versione sono decisi a priori, connettendo i segnali ai pin fisici secondo le specifiche presenti nel file con estensione UCF.

3.5 - Create and Import Peripheral Wizard

Uno dei tool che compone la suite EDK è *Create/Import Peripheral Wizard*; l'obiettivo di tale strumento è quello di aiutare lo sviluppatore nel realizzare IP-Core che siano compatibili con EDK. Si compone, come dice il nome, di due parti: la creazione di un nuovo componente (*create*) e l'importazione di un componente già esistente (*import*).

Per quanto riguarda la creazione di un nuovo componente, il tool offre un generatore automatico di template nel quale porre il codice che descrive la funzionalità dell'IP-Core che si vuole creare. Per template si intende la descrizione dell'IP-Core priva della parte di logica che esprime il comportamento del modulo. Sostanzialmente, il tool crea automaticamente il file VHDL che contiene l'istanza dell'interfaccia e della "user logic". Per quanto concerne quest'ultima, è generato un secondo file VHDL contenente la dichiarazione di entity, mentre la parte implementativa (architecture) è vuota ed è appunto la porzione che va realizzata da parte dello sviluppatore.

La funzione di importazione degli IP-Core, invece, prende in input il codice VHDL già completo che rappresenta un IP-Core e lo correda di alcuni file che lo rendono un componente realmente inseribile in un sistema EDK. Questa funzionalità è quella principalmente utilizzata nel presente lavoro di tesi.

Sostanzialmente sono creati due file:

- Peripheral Analyze Order (PAO), che definisce l'ordine dei file HDL necessari alla sintesi del componente

- Microprocessor Peripheral Definition (MPD), nel quale si trovano tutte le porte e i parametri dell'IP-Core importato, in modo che esso sia trattato in modo corretto da EDK

Il Create/Import Peripheral Wizard è dunque un utile strumento nell'ambito di lavoro di EDK, in quanto permette di facilitare e velocizzare il flusso di inserimento di un nuovo modulo all'interno di un sistema. La sua utilità è però molto ridotta al di fuori di tale ambiente, proprio perché si tratta di uno strumento sviluppato ad-hoc.

4 - PROGETTO HARDWARE

Dopo aver fornito una panoramica sul dispositivo Zynq e sugli strumenti forniti da Xilinx per lo sviluppo di sistemi che lo utilizzano, si passa ad analizzare meglio l'obiettivo del presente lavoro di tesi e la soluzione implementata.

Il progetto è quello di un meccanismo in grado di ricevere le immagini provenienti da uno (o più) sensori e, tramite opportune elaborazioni, memorizzarle nella memoria DDR3 della Zedboard, realizzando di fatto un *frame buffer*. La criticità di un sistema come questo è la notevole quantità di dati da trasferire.

Il flusso video è trasferito in memoria sfruttando il Bus AMBA e il memory controller, presenti nel Processing System dello Zynq. I frame sono poi prelevati dall'ARM, per eventuali ulteriori elaborazioni, o per la semplice visualizzazione. Il percorso dei dati in scrittura e in lettura, è evidenziato nella figura 16.

In questo esempio è visualizzata solo la gestione di un singolo flusso proveniente dall'esterno, in realtà si dovranno anche gestire degli ulteriori flussi di immagini, risultanti da elaborazioni compiute nella logica programmabile. Il percorso dei dati di questi flussi sarà assolutamente identico, e parallelo, a quello proveniente dall'esterno.

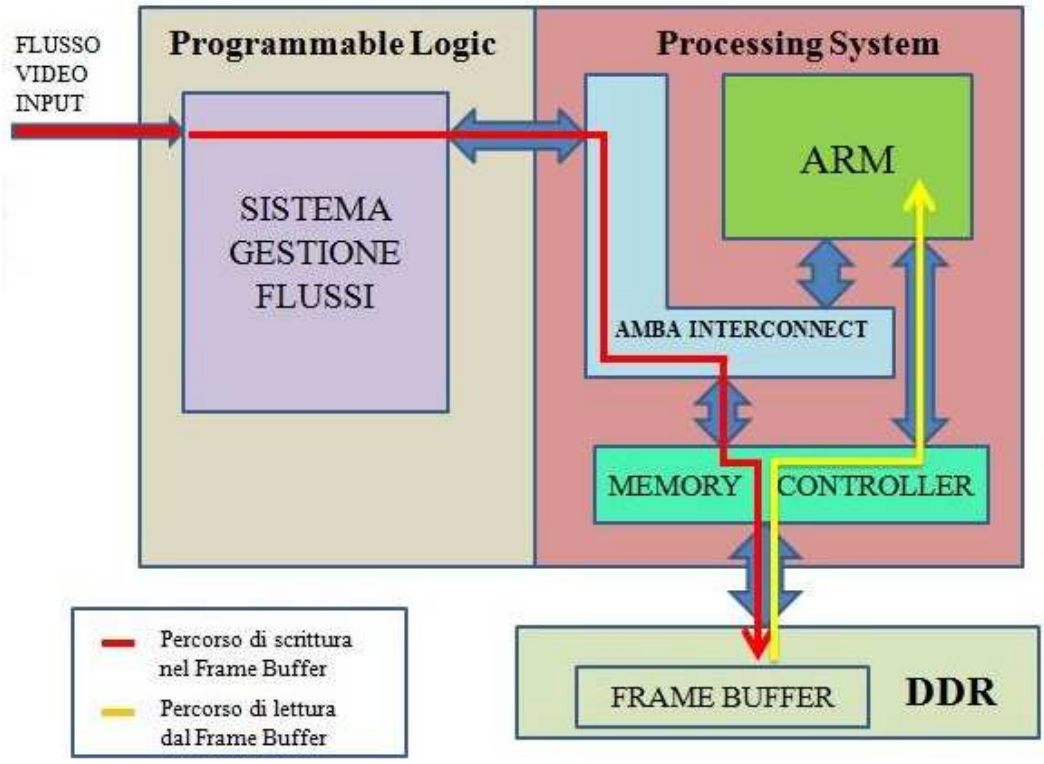


Fig.16: Percorsi dal e verso il Frame Buffer

Analizziamo ora quali sono le operazioni da realizzare, nella logica programmabile, per poter scrivere i frame nella DDR; occorre, innanzitutto, effettuare le opportune elaborazioni sul flusso di dati, in modo che siano in un formato compatibile con componenti a valle. In particolare, dovranno essere gestibili da un DMA (Direct Memory Access), il quale permetterà di depositarli in memoria, senza richiedere l'intervento del processore. Il DMA si occuperà, quindi, di andare a scrivere ogni frame in memoria, generando anche gli opportuni indirizzi. L'accesso al Memory Controller è reso possibile dalle porte presenti tra PS e PL, come illustrato nella sezione 2.2.

Mostriamo ora, in figura 17, i principali blocchi del sistema, utilizzati per la gestione di un singolo flusso video proveniente dall'esterno:

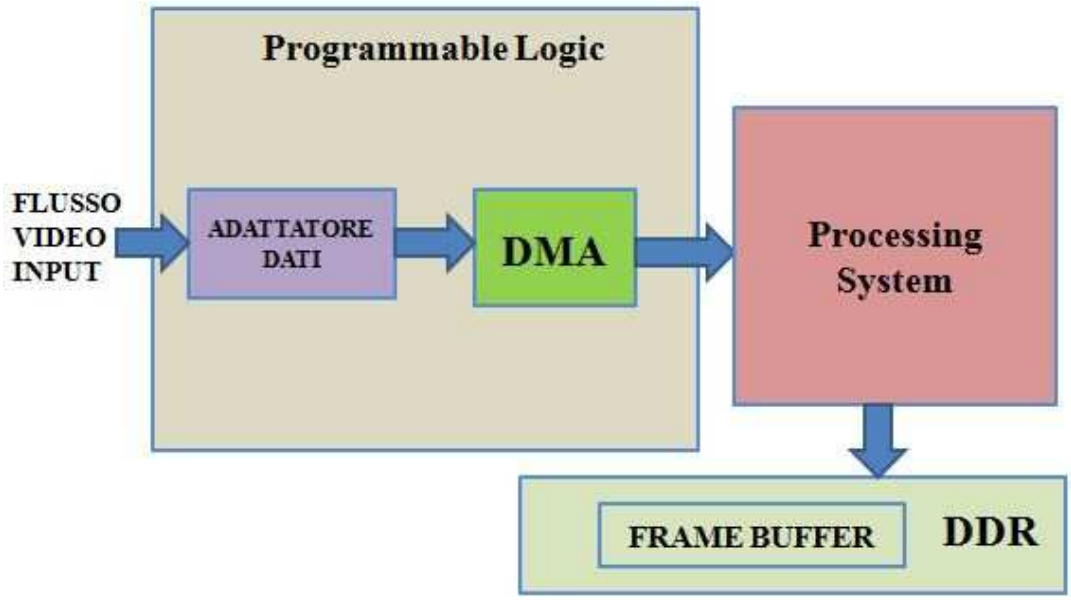


Fig.17: Sistema di gestione flussi video

Nel caso, invece, di flussi multipli, occorre duplicare il sistema, come è mostrato nella seguente figura:

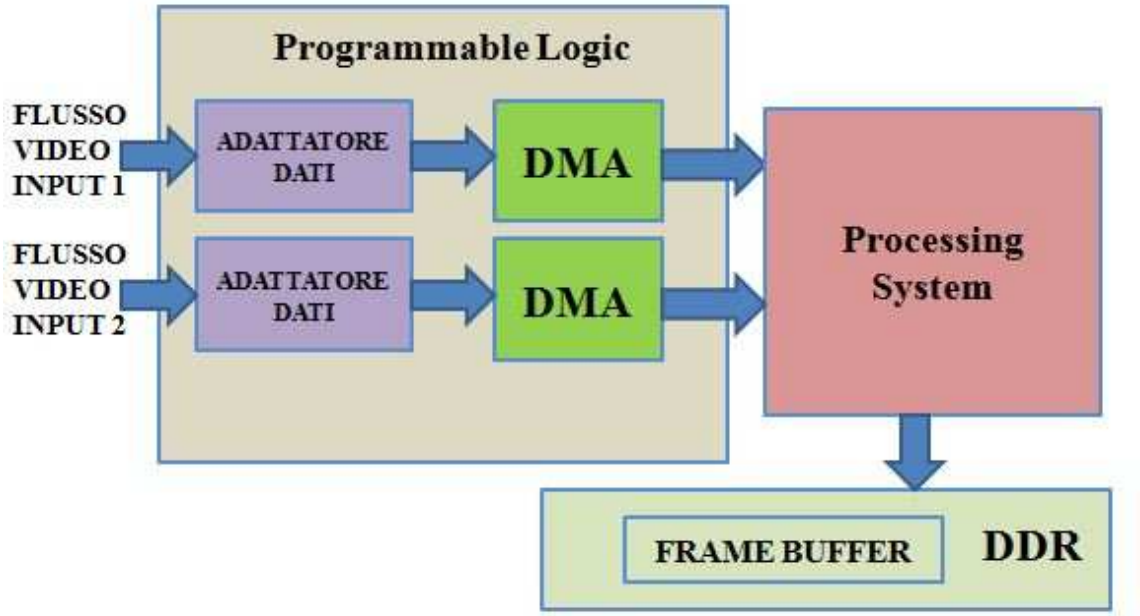


Fig.18: Sistema di gestione due flussi video

In realtà, nel caso di sensori stereo, sfruttando la perfetta sincronizzazione tra le immagini, è possibile gestire i due flussi come se fossero uno unico. Tale soluzione, illustrata meglio in seguito, ha permesso la riduzione delle risorse della PL richieste per la realizzazione del sistema. In figura 19 è mostrato il caso appena descritto, con un ulteriore flusso che è stato generato da un'elaborazione, effettuata sulle immagini provenienti dai due sensori. E' trattato anche il caso limite di quattro flussi, due provenienti dai sensori e due ottenuti dalla *pipeline* di elaborazione.

Prima di procedere in un'analisi maggiormente dettagliata del sistema realizzato, si mostra come sono trasmesse, dai sensori, le immagini in arrivo.

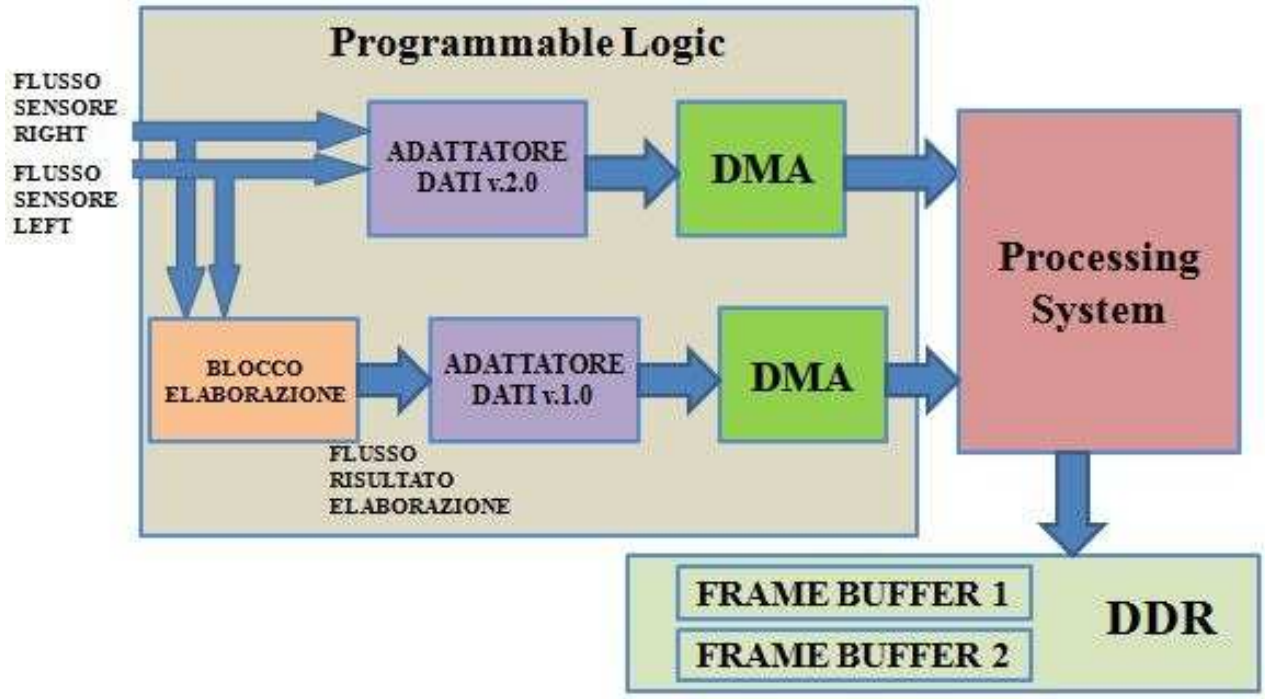


Fig.19: Sistema per la gestione di flussi multipli

4.1 - FORMATO FLUSSO VIDEO

L’immagine è fornita, in uscita dal sensore, tramite scansioni di linee, effettuate in modo progressivo. I sensori utilizzati nel progetto di ricerca, forniscono immagini con risoluzione 640x480. Come si può vedere dalla figura 20, la regione dell’immagine, considerata valida, è circondata da zone di blanking orizzontale e verticale:

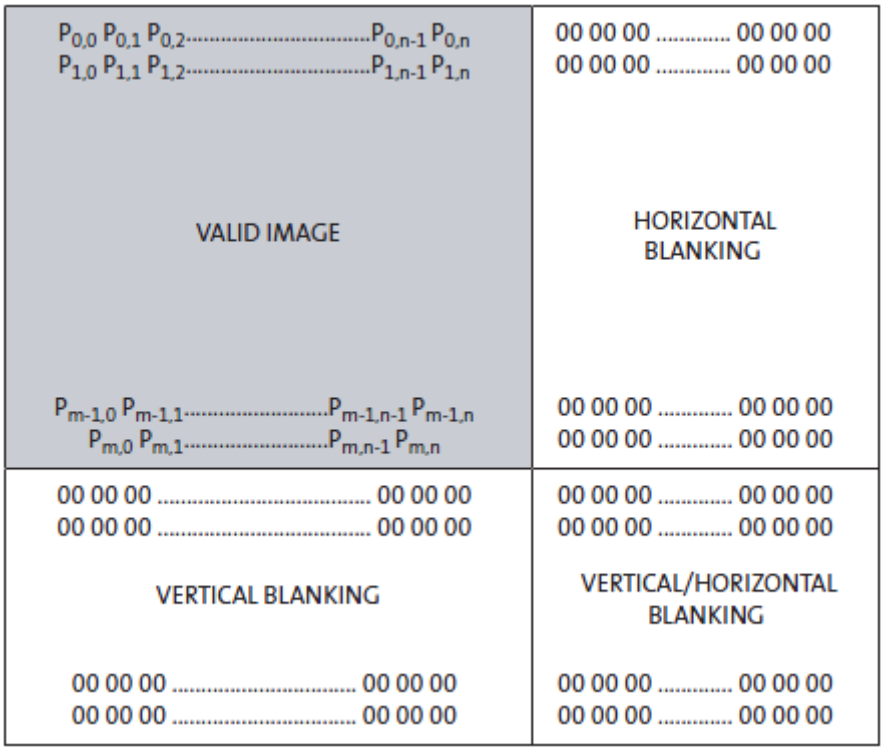


Fig.20: Formato immagine sensore (caso APTINA [9])

Nell'esempio appena mostrato, le zone di blank sono posizionate alla fine di una linea e alla fine di tutte le linee ma, in realtà, i sensori possono presentare caratteri di blanking anche prima; comunque ciò non influisce, in nessun modo, nel comportamento del sistema, poiché in tali zone non sono attivati i

segnali che indicano che un pixel appartiene all'immagine effettiva e di conseguenza, sono scartati. I segnali utilizzati dal sensore per l'invio di un'immagine, sono:

- FRAME VALID: indica che la linea che si sta trasmettendo appartiene all'immagine effettiva
- LINE VALID: indica che il pixel che si sta trasmettendo appartiene ad una linea dell'immagine
- PIXEL CLOCK: utilizzato per sincronizzare i dati in uscita
- DATA [0...N]: segnali utilizzati per la trasmissione del valore del singolo pixel

Tutti i segnali in uscita dal sensore sono sincronizzati con il PIXEL CLOCK; i segnali DATA sono utilizzati per l'invio, in parallelo, del valore dello specifico pixel. Quando il LINE VALID è HIGH, ad ogni periodo del PIXEL CLK, un pixel è emesso in uscita. Il segnale di FRAME VALID è a valore HIGH fino a quando non è terminata la trasmissione dell'intero frame catturato dal sensore. Una volta che il frame è stato trasmesso, il segnale si abbassa fino a quando non è trasmesso un nuovo frame.

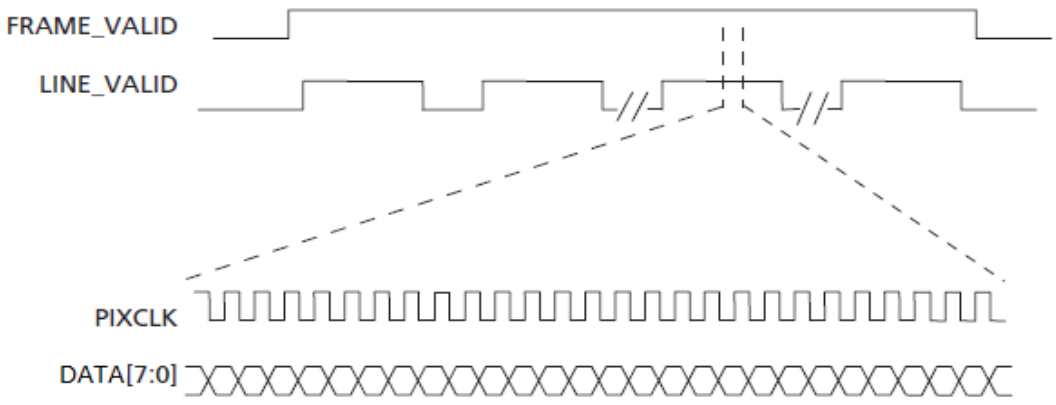


Fig.21: Trasmissione frame da parte del sensore

Nel caso di telecamere stereo, le immagini provenienti dai due sensori sono perfettamente sincronizzate, in modo da permettere elaborazioni hardware su due flussi contemporaneamente. Il sistema però è capace di gestire anche flussi non sincroni, permettendo quindi una notevole flessibilità di utilizzo. Il test con telecamere reali avrebbe reso più complicata la fase di test e per questo sono stati utilizzati dei componenti che ne simulino il comportamento.

4.2 - GENERATORE DI PATTERN

Per favorire la fase di debug e di test sono stati utilizzati dei generatori d'immagini sintetiche, realizzati ad hoc per valutare e risolvere alcune problematiche riscontrate.

L'utilizzo di questi componenti, il cui comportamento è facilmente modificabile, ha permesso anche la valutazione del funzionamento del sistema per diverse tipologie di sensori, soprattutto per quanto riguarda la frequenza di invio delle immagini.

Questi generatori d'immagini, forniscono frame con risoluzione 640x480, sfruttando segnali e temporizzazioni identici a quelli dei sensori reali. Come segnali di input, il generatore di pattern prevede invece, un RESET ed un clock, utilizzato da un generatore di clock interno per la creazione del PIXEL CLOCK richiesto. Per la realizzazione su scheda è stato adottato come clock quello esterno e disponibile per la PL che è a 100 MHz. In figura 22 è riportata una rappresentazione del componente, con tutti i suoi segnali d'ingresso e di uscita:

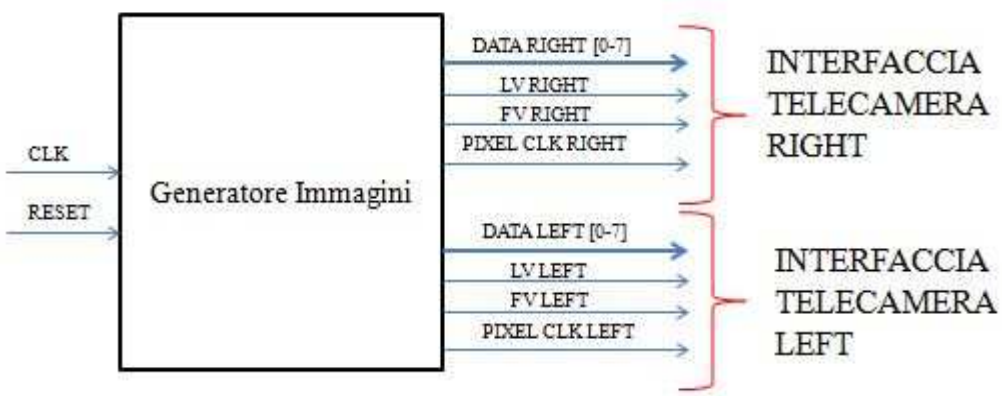


Fig.22: Generatore immagini sintetiche

La prima versione, già esistente, genera costantemente una scacchiera, alternando quadrati di colore bianco e nero, ognuno di lato pari a 16 pixel. Questo generatore fornisce immagini significative, a livello di singolo frame, poiché si conosce a priori il numero di pixel consecutivi uguali e si propone come un importante strumento per la verifica della scrittura in memoria. Però non permette valutazioni utili tra frame consecutivi, poichè tutti uguali.

Tale componente è stato poi modificato perché fosse possibile accertarsi che la scrittura di un frame iniziasse all'indirizzo corretto; in particolare, che il primo pixel fosse inserito in corrispondenza del primo indirizzo. Questo generatore fornisce, infatti, immagini con il primo pixel al valore 0x05, tutti gli altri al valore 0xFF.

La terza e ultima versione genera immagini con frame che contengono solo pixel identici, che però sono diversi tra frame successivi; il primo frame ha tutti i pixel al valore 0x05, il secondo a 0x06, il terzo a 0x07 e poi a 0x08, dopo di che riparte, in maniera ciclica, dal valore 0x05. Genera un numero di valori che è diverso dal numero di frame memorizzati; ciò per garantire che, in memoria, due frame memorizzati consecutivamente, nello stesso slot, abbiano valori differenti.

Tali componenti sono stati immessi nel sistema rendendoli IP-Core, sfruttando il wizard *Create and Import Peripheal Wizard*, già descritto nella sezione 3.5.

4.3 - PROTOCOLLO AXI

I DMA coinvolti nel trasferimento devono sfruttare una canale di comunicazione tra PL e PS. Basandosi sulle considerazioni fatte nel capitolo 2, risulta evidente che il metodo migliore per il raggiungimento di tale obiettivo è lo sfruttamento delle porte AXI Slave High-Performance. Per farlo, occorre utilizzare un IP-Core, in grado di svolgere il ruolo di Master per il protocollo AXI; tra quelli messi a disposizione da Xilinx, la scelta è ricaduta sull'IP-Core AXI Video DMA (VDMA). Prima di analizzare le funzionalità offerte da questo modulo, è opportuno comprendere al meglio l'interfaccia AXI e le diverse tipologie esistenti.

Con il termine AXI (Advanced eXtensible Interface) si intende un protocollo standard, che definisce le specifiche per l'interfacciamento tra un Master ed uno Slave, rappresentati da IP Core che debbono scambiarsi informazioni. AXI è basato sulle specifiche AMBA (Advanced Microcontroller Bus Architecture) e la sua prima versione è stata inclusa in AMBA 3.0; la seconda versione (AXI4) è stata rilasciata nel 2010 ed inclusa in AMBA 4.0 [10]. Esistono tre tipi di interfaccia AXI4:

- **AXI4**: Fornisce elevate prestazioni per accessi di tipo memory mapped. Permette fino a 256 trasferimenti con singolo indirizzamento (*burst*)
- **AXI4-Lite:** Previsto per semplici operazioni (trasferimenti singoli), di tipo memory mapped e che richiedono basso livello di throughput (utilizzato principalmente per la scrittura e la lettura di registri di stato e di controllo)

- **AXI4-Stream:** Per lo streaming, ad alte prestazioni, di dati. Non prevede indirizzamento e permette data burst di dimensione illimitata

I vantaggi principali offerti dall'utilizzo di questo protocollo riguardano:

- **PRODUCTIVITY:** Attraverso la standardizzazione dell'interfaccia AXI, gli sviluppatori necessitano di apprendere un singolo protocollo per IP Core
- **FLEXIBILITY:** Essendoci tre modalità differenti, offre la possibilità di scegliere il protocollo più adatto ad ogni esigenza
- **AVAILABILITY:** Si ha accesso non solo al catalogo di IP Xilinx, ma anche a quelli di tutti i partner ARM

4.4 - AXI4 e AXI4-Lite

Quest'interfaccia rappresenta, per lo Zynq, l'unico modo possibile per effettuare lo scambio di dati tra Processing System e Programmable Logic; ciò, è possibile, sfruttando le porte General Purpose o High Performance, presenti tra le due parti.

AXI prevede indirizzi a 32 bit e parallelismo configurabile (a 32, 64, 128, 256, 512 e 1024 bit), il quale andrà, quindi, a influenzarne prestazioni e occupazione di risorse. Sia l'interfaccia AXI4 sia l' AXI4-Lite sono costituite da cinque canali differenti:

- Read Address Channel
- Write Address Channel
- Read Data Channel
- Write Data Channel
- Write Response Channel

I dati scambiati possono viaggiare simultaneamente in entrambe le direzioni, con trasferimenti a dimensione variabile; la bi-direzionalità è possibile poiché sono utilizzate connessioni separate (indirizzi e dati), in caso di lettura o scrittura. Di seguito sono mostrati i due casi, con i relativi canali coinvolti nelle operazioni.

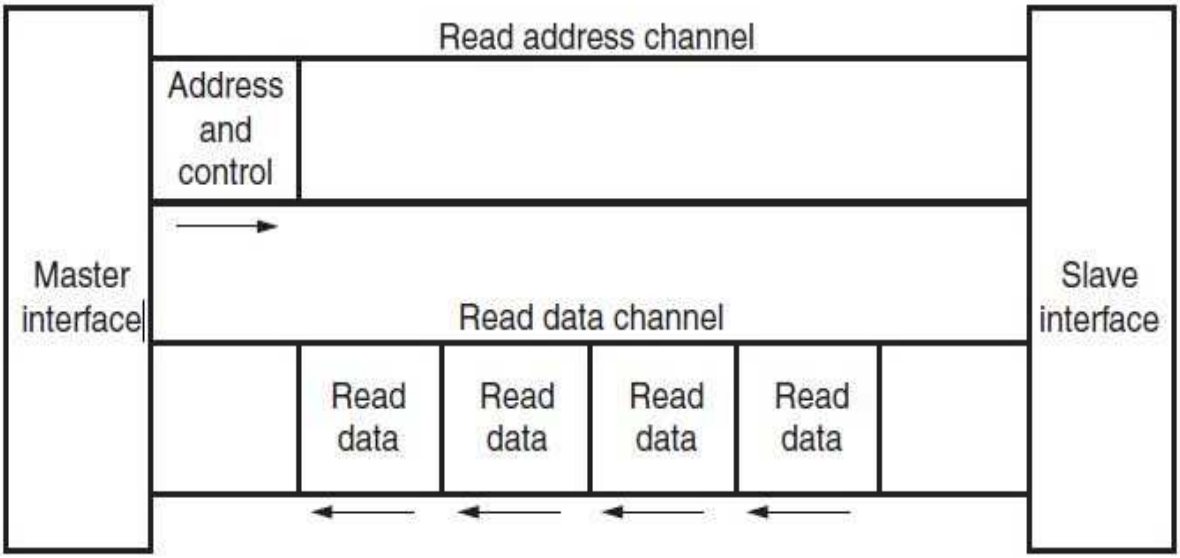


Figure 23: Channel Architecture of Reads

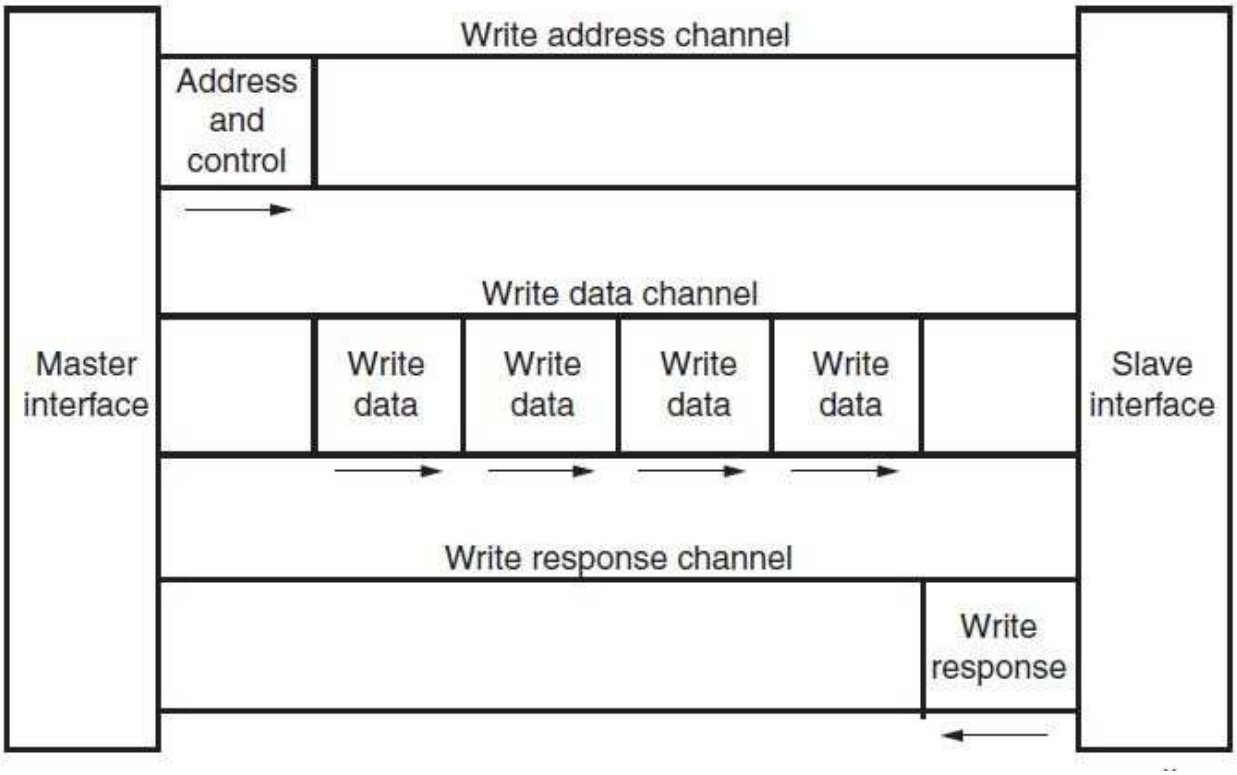


Figure 24: Channel Architecture of Writes

Il protocollo offre, oltre al bursting, altre funzionalità per migliorare le prestazioni del sistema:

- Data downsizing e upsizing
- multiple outstanding addresses
- out-of-order transaction processing

Il master e lo slave AXI devono essere connessi tramite un blocco, denominato *Interconnect*, il quale permette la connessione tra uno o più master e uno o più slave, altrimenti non possibile. Questo IP-Core offre anche numerose funzionalità aggiuntive, alcune delle quale saranno presentate nella sezione ad esso relativa. Esistono due tipi di *Interconnect*, uno specifico per AXI4-Stream (AXIS Interconnect) e un altro da utilizzare per l'AXI4 e l'AXI4-Lite (AXI Interconnect).

4.5 - AXI4-Stream

Questo protocollo risulta particolarmente adatto per applicazioni nelle quali è richiesto lo scambio di una notevole quantità di dati e in cui non è presente, o non è richiesto, il concetto di indirizzamento. Ogni connessione AXI-Stream (AXI4S) è *point-to-point*, monodirezionale e di tipo *Handshake.* Dato l'alto numero di ambiti in cui esso è utilizzabile, i segnali del protocollo assumono significato diverso in base al contesto nel quale sono utilizzati; riportiamo, di seguito, i principali segnali del protocollo e il significato assunto nell'ambito di applicazioni video:

Function	**Width**	**AXIS Signal Name**	**Video Specific Name**
Video Data	8, 16, 24, 32, 40, 48,56,64	tdata	DATA
Valid	1	tvalid	VALID
Ready	1	tready	READY
Start Of Frame	1	tuser	SOF
End Of Line	1	tlast	EOL
Clock	1	aclk	ACLK

Tab.3: AXI4-Stream Video Protocol Signals

Prima di dare una descrizione dettagliata di tali segnali, e di come essi siano utilizzati, occorre specificare che AXI-Stream prevede il trasferimento dei soli pixel effettivamente appartenenti all'immagine.

L'unico segnale controllato dal dispositivo Slave è il *Ready* ed è utilizzato per indicare la disponibilità a ricevere dati; tutti gli altri sono sotto la responsabilità del Master.

Il Master utilizza il segnale *Valid* per indicare il trasferimento di un pixel appartenente all'immagine, il cui valore è trasmesso attraverso i segnali *Video Data*.

Il segnale Start Of Frame è attivato finché non ne è confermata la ricezione, durante la trasmissione del primo pixel di un frame; il segnale End Of Line indica, invece, la fine di una linea, ed è attivato durante la trasmissione dell'ultimo pixel della stessa. Ecco un esempio di utilizzo di questo protocollo per la trasmissione di immagini aventi N pixel per riga

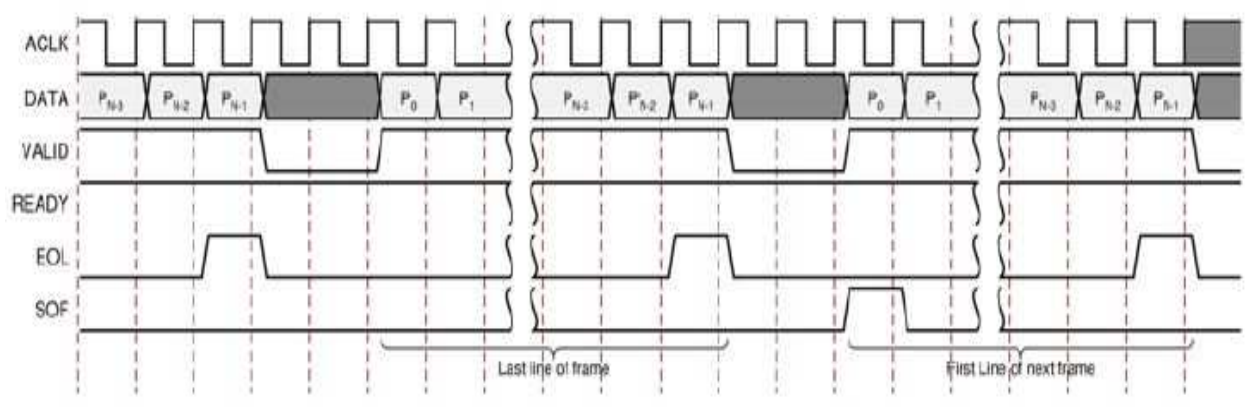


Fig.25: Esempio trasferimento flusso video con Axi-Stream [10]

4.6 - AXI INTERCONNECT CORE

Questo IP-Core offre la possibilità di collegare fino a 16 Master AXI4 (o AXI4-Lite), con un massimo di 16 Slave [11]; i dispositivi, connessi tramite questa infrastruttura di comunicazione, possono avere interfacce con parallelismo diverso e clock differenziati, rendendo l'AXI molto flessibile e potente. Se lo si utilizza per connettere semplicemente un Master ed uno Slave, aventi lo stesso parallelismo e lo stesso clock, questo IP-Core si comporterà come una serie di fili tra essi, senza andare ad occupare nessuna risorsa della PL (Fig.26); aumentando le funzionalità richieste, la fase di sintesi richiederà maggiori risorse e saranno introdotti dei ritardi nel canale di comunicazione, causati dalle operazioni effettuate per rendere compatibili tutti i dispositivi interconnessi (Fig.27).

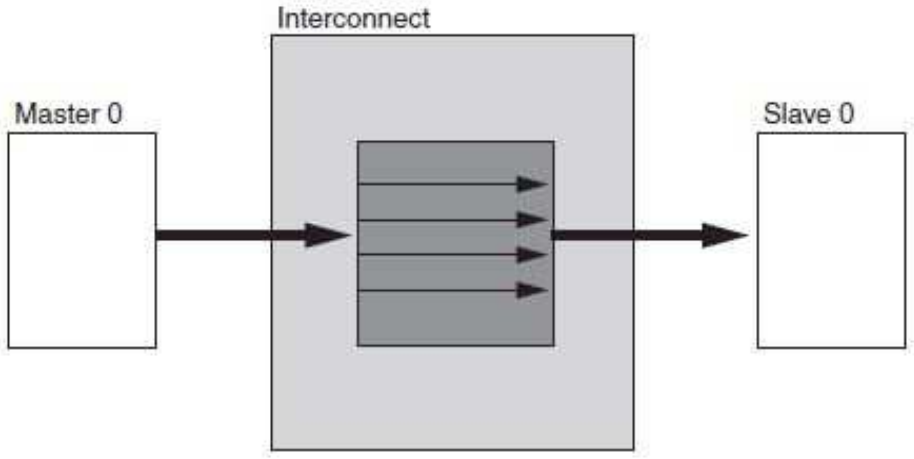


Fig.26: 1-1 Pass-through AXI Interconnect

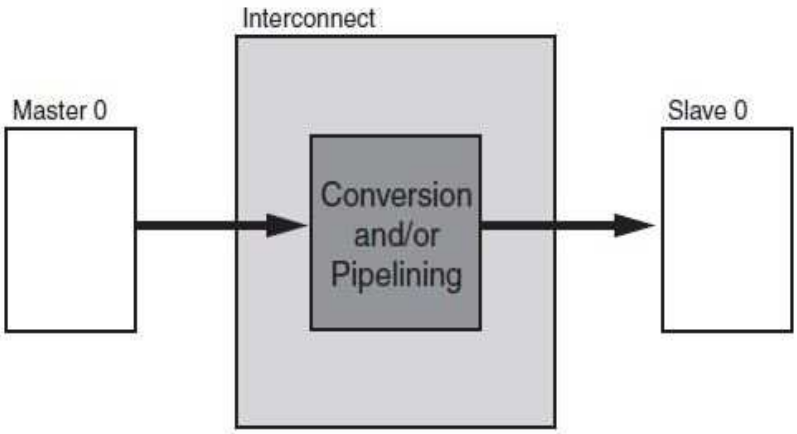


Fig.27: 1-1 AXI Interconnect con conversioni

Un'altra caratteristica importante è che, tramite lo stesso blocco *Interconnect*, è possibile far comunicare dispositivi sia con interfaccia AXI4 sia con AXI4-Lite. Si mostra ora una visione ad alto livello della struttura di questo IP-Core:

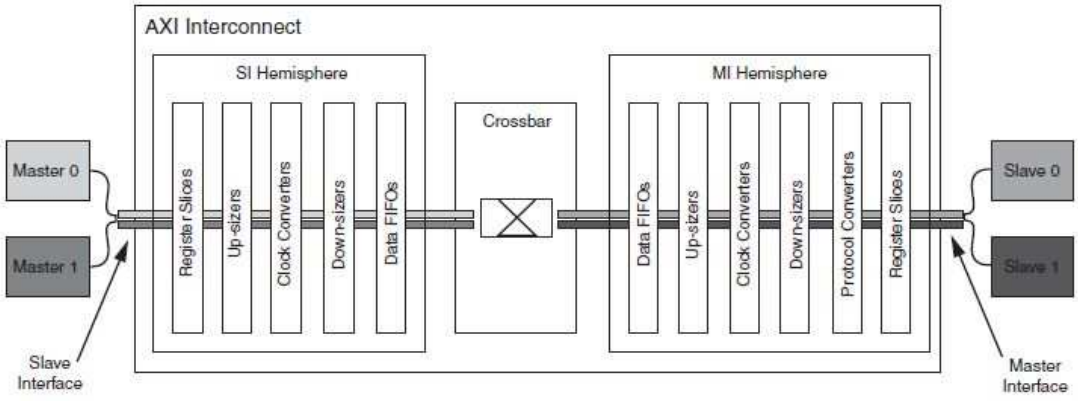


Fig.28: IP-Core AXI Interconnect

L'AXI Interconnect è costituito da una *Slave Interface* (SI), una *Master Interface* (MI) e dall'unità funzionale che crea un percorso tra esse (crossbar). Le funzionalità aggiuntive, descritte in precedenza, sono realizzate dai

componenti presenti tra la singola interfaccia e la crossbar stessa. Nel caso in cui sono presenti più master, la priorità tra essi è decisa da un arbitro, configurabile con priorità fisse o per lavorare in modo Round Robin. Un esempio nel quale è necessario l'intervento dell'arbitro, è il seguente:

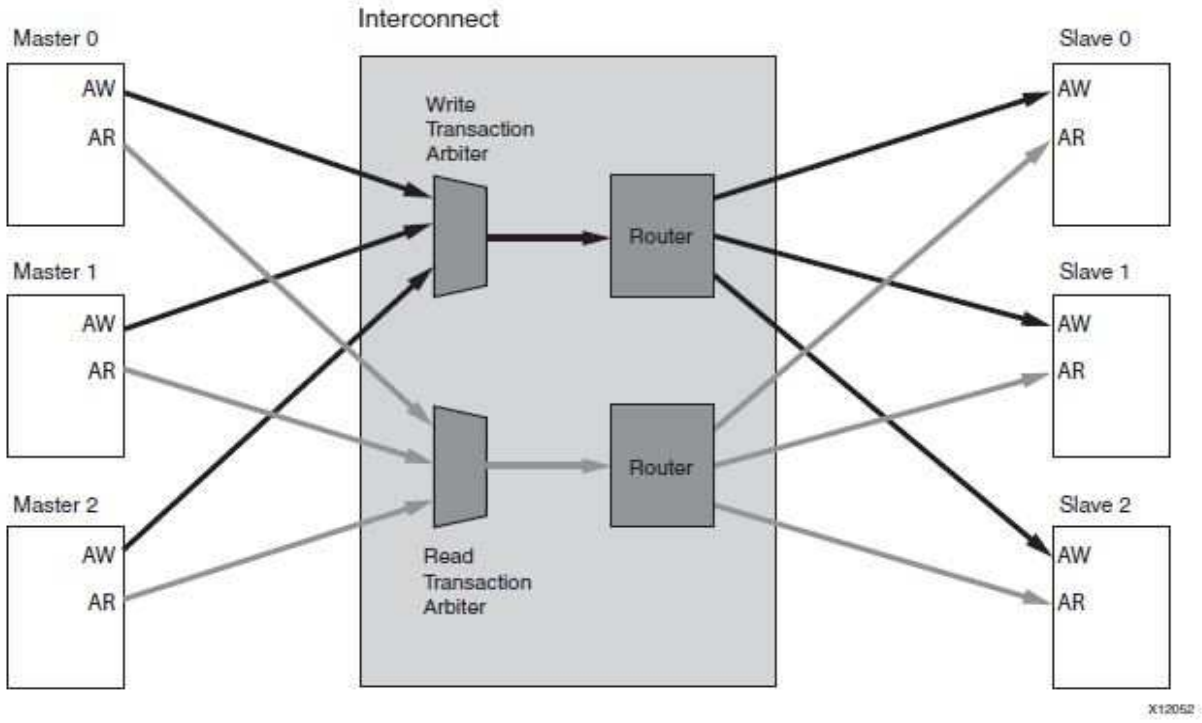


Fig.29: Shared Write and Read Address Arbitration

4.7 - AXI VDMA

Questo IP-Core permette il trasferimento di dati, fornendo una grande larghezza di banda, tra la memoria e un dispositivo che presenta l'interfaccia AXI-Stream [12]. Questo componente è costituito da due canali:

- **CANALE DI SCRITTURA (S2MM):** sfruttando questo canale è in grado di ricevere immagini attraverso l'AXIS e, assumendo il ruolo di Master AXI, bufferizza i frame in memoria.
- **CANALE DI LETTURA (MM2S):** si tratta dell'operazione duale, intendendo con questo termine l'estrazione dalla memoria di un frame, il quale è inviato successivamente ad uno slave con l'interfaccia AXIS.

Questo IP-Core assume, quindi, il ruolo di Master e quello di Slave, sia per l'AXI sia per l'AXI-Stream; presenta, inoltre, un'interfaccia Slave AXI-Lite, da utilizzare per la sua programmazione. Nel caso dello Zynq entrambi i canali sfruttano, per accedere alla memoria, una delle porte High-Performance presenti tra PS e PL, poiché, come detto in precedenza, sono quelle che garantiscono il maggior *throughput*. Ogni canale è attivabile (e configurabile) in maniera totalmente indipendente dall'altro. In figura 30 è riportato un diagramma a blocchi del VDMA.

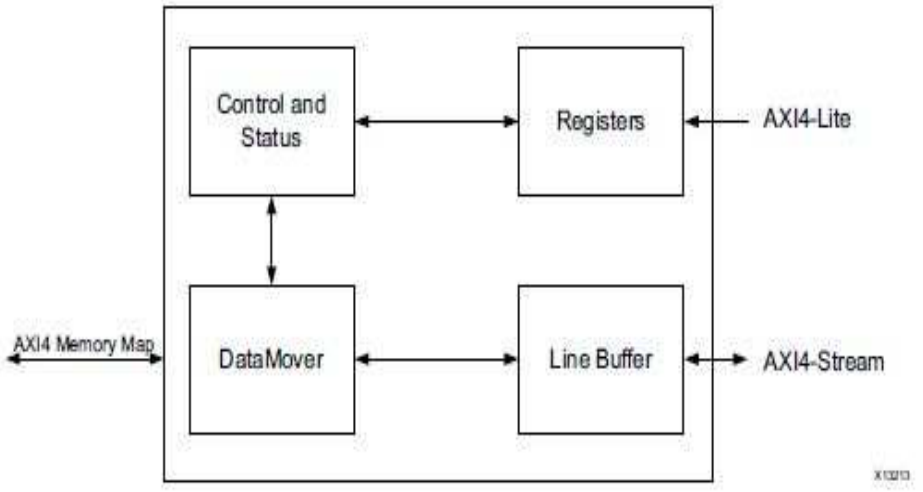


Fig.30: Diagramma a blocchi del VDMA

Dopo aver programmato i registri, attraverso l'AXI-Lite, il blocco denominato *Control and Status* genera i comandi appropriati per il *DataMover*, il quale inizia la lettura o la scrittura tramite l'interfaccia AXI4. Come si può notare, è presente, inoltre, un *Line Buffer* nel quale sono depositati i valori dei pixel, prima che essi siano trasferiti.

Il VDMA offre la possibilità di realizzare un frame buffer circolare, memorizzando un numero configurabile di frame (al massimo 32).

Per il seguente lavoro di tesi si è sfruttato solamente il canale di scrittura verso la memoria, l'estrazione dei frame è stata poi effettuata tramite il processore. Mostriamo ora (figura 31) un esempio, con relative temporizzazioni, del funzionamento di questo canale.

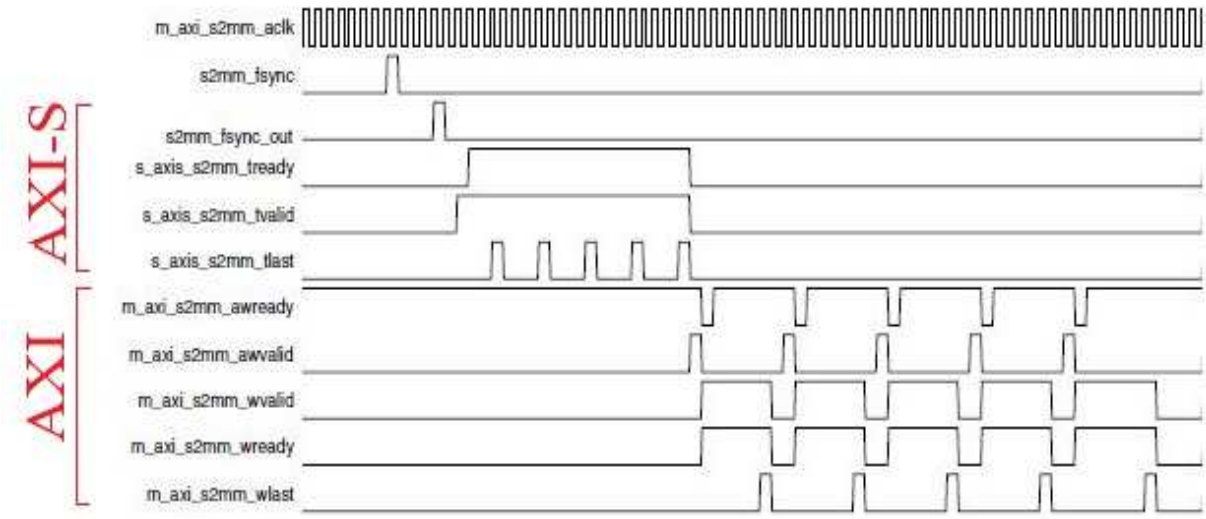


Fig.31: Esempio VDMA S2MM Channel

In tal caso, un'immagine è costituita da 5 linee, ognuna di 16 byte. Il VDMA, dopo aver ricevuto il segnale fsync, si rende disponibile a ricevere dati sull'interfaccia AXIS, asserendo il segnale tready; lo streaming di dati è memorizzato nel line buffer e, successivamente, trasferito in memoria sfruttando l'interfaccia AXI.

Il VDMA offre anche la possibilità di sincronizzare il canale di scrittura con quello di lettura, al fine di evitare l'accesso contemporaneo allo stesso frame. Parte di questo meccanismo è stato sfruttato per evitare che il processore prelevi il frame che il VDMA sta scrivendo, operazione assolutamente da non compiere.

Xilinx offre, per la programmazione di questo IP-Core, i relativi driver, sia per Linux sia per applicazioni di tipo Bare-Metal. L'esistenza di questi driver evita, al programmatore, di dover configurare il componente tramite la lettura e scrittura diretta dei registri, operazione che risulterebbe molto delicata e complessa.

Per il raggiungimento dell'obiettivo di questo lavoro di tesi si vogliono gestire dei flussi d'immagini provenienti da sensori e servirà quindi aggiungere, a monte del VDMA, un componente in grado di presentare tali immagini in modo compatibile con l'interfaccia AXI-Stream; tale componente esiste tra quelli offerti da Xilinx, ed è il *Video in to Axi-Stream*, illustrato nella prossima sezione.

In figura 32 viene mostrato il VDMA, e le relative interfacce:

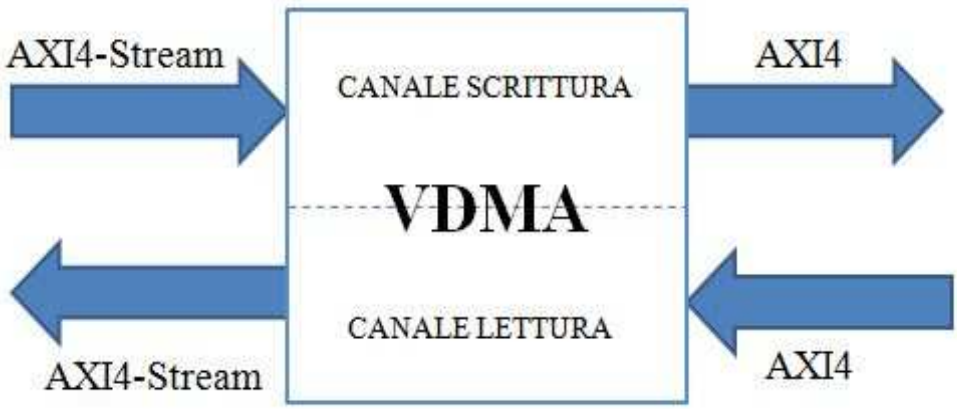


Fig.32: Interfacce VDMA

Per risparmiare risorse della PL è stato disattivato completamente il canale di lettura dei VDMA, poiché non utilizzato in questo progetto.

4.8 - VIDEO IN TO AXI4-STREAM

Questo IP-Core è progettato [13] per interfacciare un flusso video con un componente che presenta l'interfaccia AXI-Stream. I principali utilizzi del core sono per le seguenti fonti video:

- DVI
- HDMI
- Sensori d'immagine
- Generatori d'immagini sintetiche

Il flusso in entrata deve essere di tipo parallelo, con un pixel clock e deve presentare uno di questi gruppi di segnali:

- Vsync, Hsync e Data Valid
- Vblank, Hblank e Data Valid
- Vsync, Hsync, Vblank, Hblank e Data Valid

Qualsiasi insieme di questi segnali è sufficiente al corretto funzionamento del componente; in uscita presenta, invece, un'interfaccia Master Axi-Stream, in grado di trasmettere, nel caso specifico al VDMA, le immagini da trasferire in memoria. Essendoci due interfacce, una di input e una di output, esse possono presentare due clock differenti: sarà quindi compito dell'IP-Core gestire

correttamente tale situazione; ciò è possibile grazie all'utilizzo di una FIFO, dimensionata opportunamente. Nel caso in cui il clock dell'interfaccia AXI-Stream abbia frequenza uguale o maggiore del clock in ingresso è sufficiente la FIFO di dimensione minima (32 slot). In figura 33 ne è mostrato uno schema semplificato.

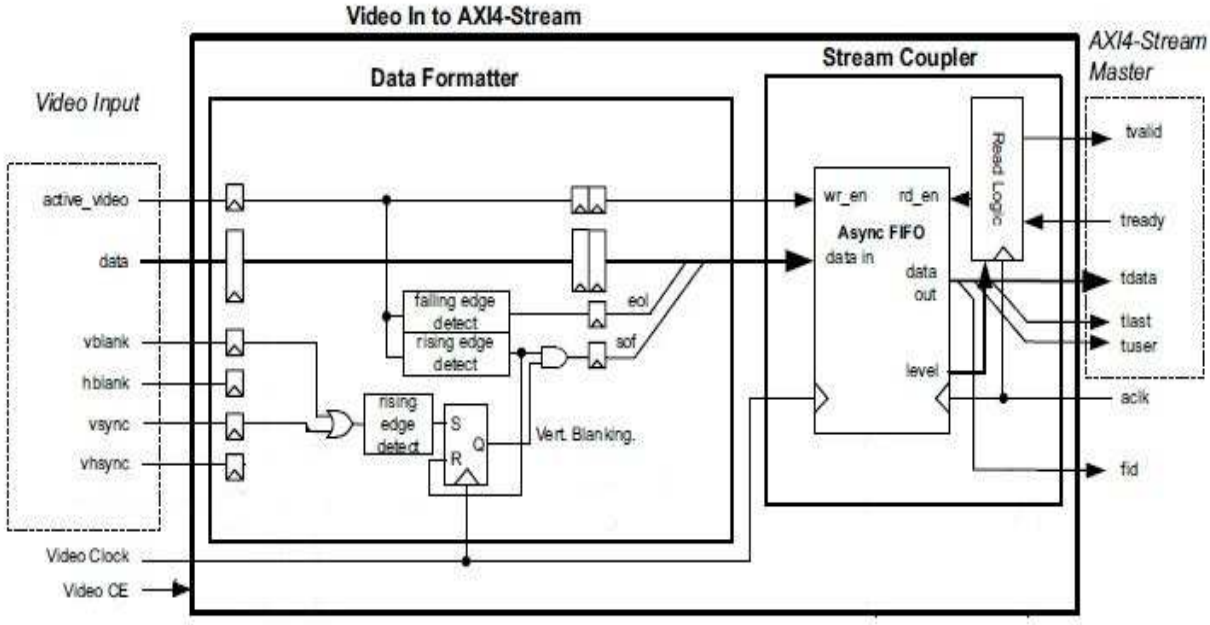


Fig.33: Video In To Axi-Stream

Il componente supporta, per quanto concerne l'interfaccia di input, molti formati per le immagini: RGB, YUV e Monochrome; il numero di bit per componente è variabile e va configurato prima della sintesi.

Anche per questo IP-Core, come per gli altri, la quantità di risorse della logica programmabile occupate, dipende fortemente dalle funzionalità specifiche richieste.

4.9 - ADATTATORE SEGNALI

Il Video In to AXIS offre la possibilità di scegliere tra vari gruppi di segnali da utilizzare per fornire il flusso video in ingresso, ma tra questi non c'è ne nessuno che sfrutti direttamente FV e LV. Tali segnali, però, non sono altro che i complementari di Horizontal Blank e Vertical Blank e, quindi, non si deve far altro che utilizzare un NOT per ottenerli. La generazione alternativa di FSync e VSync sarebbe sicuramente risultata più difficoltosa e, potendo scegliere, si è optato per la soluzione più veloce. Esempio di un'immagine trasmessa con segnali di blank e sync:

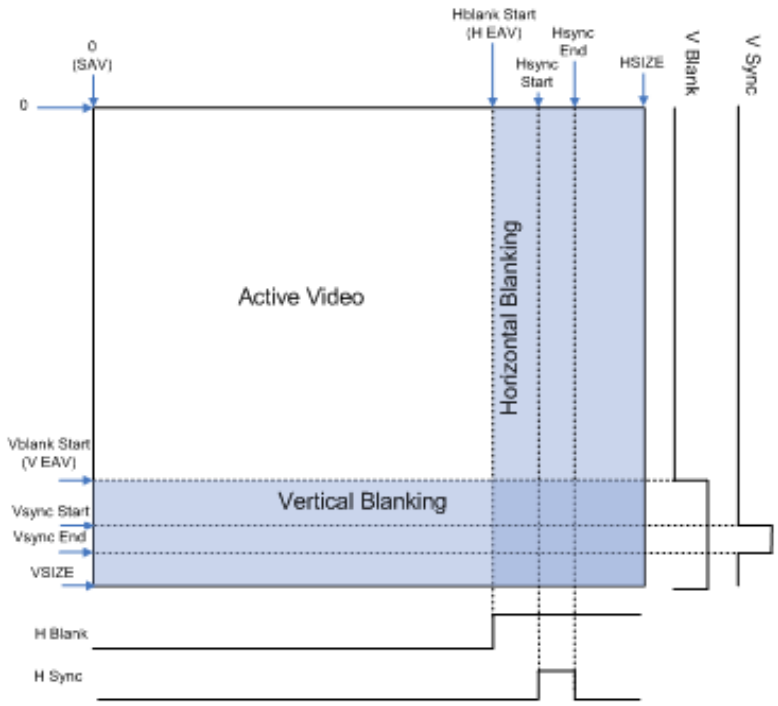


Fig.34: Immagine trasmessa con segnali di blank e sync

Occorre, inoltre, generare il segnale di ACTIVE VIDEO, il quale indica che stiamo trasferendo un pixel appartenente all'immagine effettiva; ciò, è possibile, sfruttando sempre LV e FV. Infatti, ACTIVE VIDEO non è altro che un AND tra essi. Il componente genera quindi questi segnali, appena descritti, per un singolo flusso:

Fig.35 Segnali di input ed output Adattatore

Per l'invio dei due flussi, dato il perfetto sincronismo, si è pensato di unificarli a monte del sistema, considerandolo come se fosse uno unico. Tale aggregazione è stata effettuata in una seconda versione del componente, il quale fornisce in uscita 16 bit per pixel, con un Byte che rappresenta il pixel proveniente dal sensore destro e l'altro quello del sensore sinistro. I vantaggi apportati da questa scelta progettuale sono notevoli e saranno discussi in modo dettagliato più avanti, nel capitolo 6.

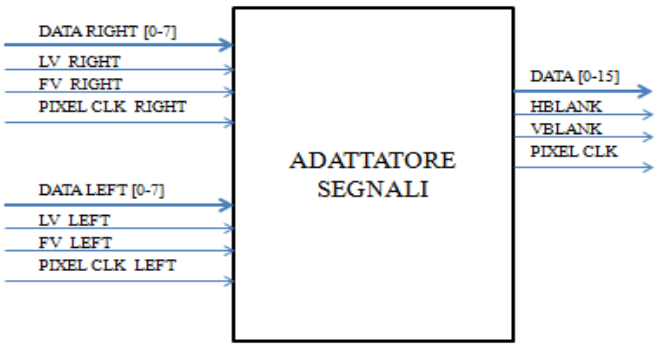


Fig.36 Segnali di input ed output dell'adattatore per immagini stereo

4.10 - SISTEMA GESTIONE FLUSSI

Dopo aver fornito una panoramica sui componenti, andiamo a vedere come è stato possibile sviluppare l'architettura che li sfrutta. Inizieremo analizzando la composizione del sistema implementato per la gestione di un singolo flusso e che è rappresentato nella seguente figura:

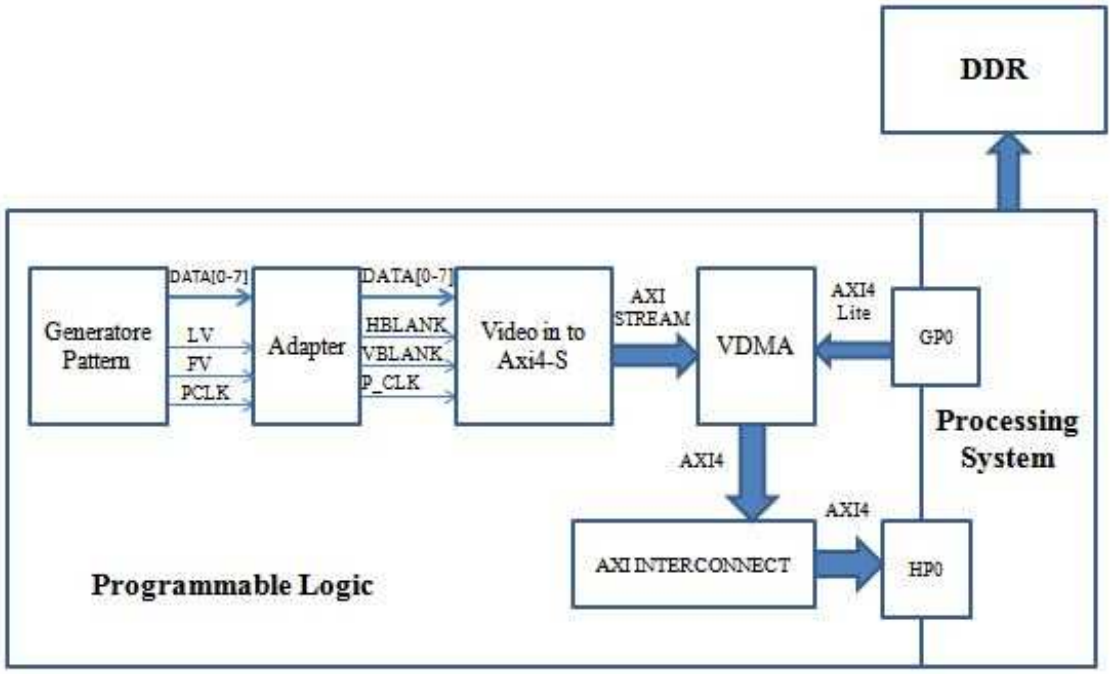


Fig.37: Sistema hardware per la gestione di un singolo flusso

Le immagini provenienti dal generatore di pattern (o dal sensore), in formato parallelo con LV e FV, sono inviate all'adattatore, il quale le converte e genera i segnali compatibili con il Video in to AXI4-S. Tale IP-Core, sfruttando un bus AXI-Stream, li invia al VDMA che, dopo essere stato opportunamente programmato, inizia i trasferimenti verso la porta High-Performance del Processing System. Per la programmazione dei registri del VDMA si è

sfruttata una interconnessione AXI-Lite, utilizzata dal PS tramite una porta AXI di tipo General Purpose. L'AXI4 Stream utilizza un bus dati con parallelismo 8 bit, ed è stato fatto lavorare con una frequenza pari al pixel clock; velocità maggiori non avrebbero portato vantaggi, ma solo maggiori consumi, poiché il Video in to AXIS avrebbe rallentato, infatti, il suo normale funzionamento, attendendo, tra un invio e l'altro, l'arrivo del pixel successivo. Per quanto riguarda il bus AXI4 e l'AXI4-Lite, sono stati utilizzati dei clock più veloci, soprattutto nel caso di gestione di flussi multipli; maggiori dettagli saranno dati nel capitolo 6, relativo alla valutazione dei risultati ottenuti. Una volta giunti all'interno del PS, i dati sono instradati, tramite il bus AMBA, fino al Memory Controller che ne permetterà la scrittura nella DDR3 della Zedboard.

Si passa ora ad analizzare il caso che riguarda la gestione dei flussi provenienti dai sensori stereo. Come anticipato, essi sono stati trattati come se fossero uno unico. E’ l'Adattatore, infatti, che si occupa di unificarli, fornendoli al Video in to AXIS con un parallelismo di 16 bit; di conseguenza, si è dovuto cambiare anche il parallelismo dell'AXI-Stream, per permettergli di continuare a trasmettere alla velocità del pixel clock. Non sono servite, invece, modifiche al bus AXI: esso lavorava già con un parallelismo di 32 bit e una frequenza che ha permesso anche la gestione del nuovo flusso, nonostante la larghezza di banda richiesta sia doppia. In figura 38 è mostrato tale sistema.

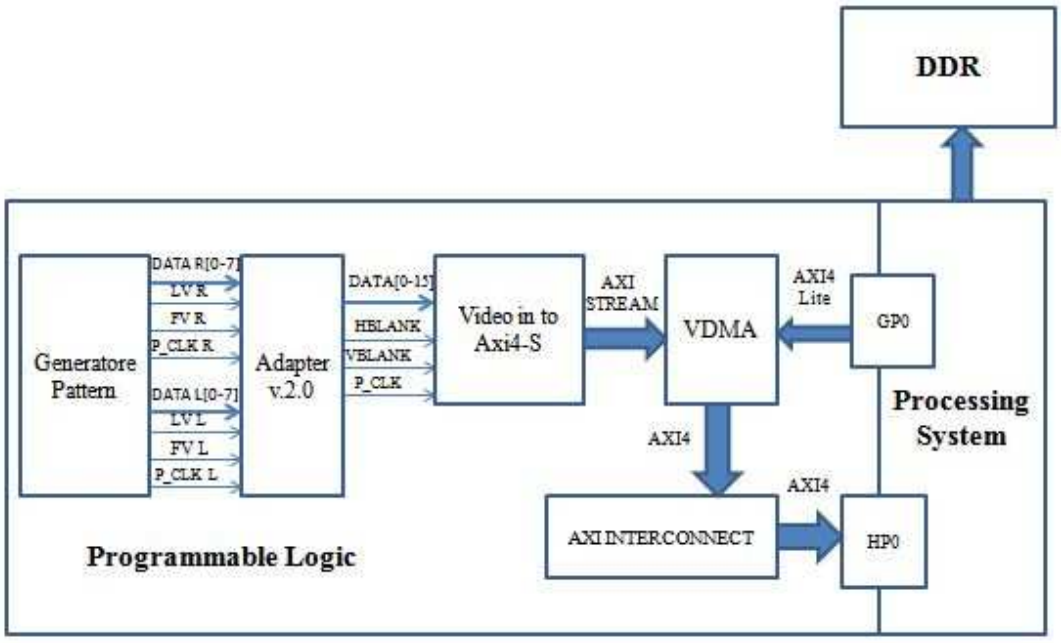


Fig.38: Sistema che gestisce flusso di sensori stereo

Come caso limite è stata considerata la gestione di 4 flussi, due provenienti dai sensori stereo e altri due ottenuti da elaborazioni, effettuate in hardware, su di essi. I flussi provenienti da elaborazioni non sono perfettamente sincronizzati con quelli provenienti dalle telecamere, ovvero il loro inizio e fine non coincide pur mantenendo lo stesso clock,perciò vanno considerati in maniera indipendente da essi.

Il sistema è stato composto mettendo in parallelo i componenti necessari alla gestione di un flusso stereo, oltre a quelli di due flussi indipendenti; tali componenti sono:

- **3 VDMA**: tutti e tre presentano la medesima interfaccia AXI4 mentre, il parallelismo dell'interfaccia AXI-Stream, è di 16 bit in un caso e di 8 bit negli altri due
- **3 Video In To Axis:** due con interfacce dati a 8 bit e uno a 16. Quello a 16 è utilizzato per la gestione del flusso stereo, gli altri due per quelli singoli

- **3 Adattatori:** due nella versione 1.0 e uno nella versione 2.0. Come per il Video In To Axis, la versione 2.0 è utilizzate per il flusso stereo
- **1 AXI-Lite Interconnect:** tutti i dispositivi da programmare (3 VDMA) sono collegati ad esso
- **1 AXI Interconnect:** tutte le interfacce Master AXI dei VDMA sfruttano lo stesso, permettendo un notevole risparmio di risorse
- **1 Porta AXI HP del PS:** Utilizzando un unico blocco Interconnect è possibile utilizzare una sola porta HP, lasciando le altre disponibili per altri utilizzi

Mostriamo, in figura 39, come è stato realizzato il sistema che utilizza questi componenti:

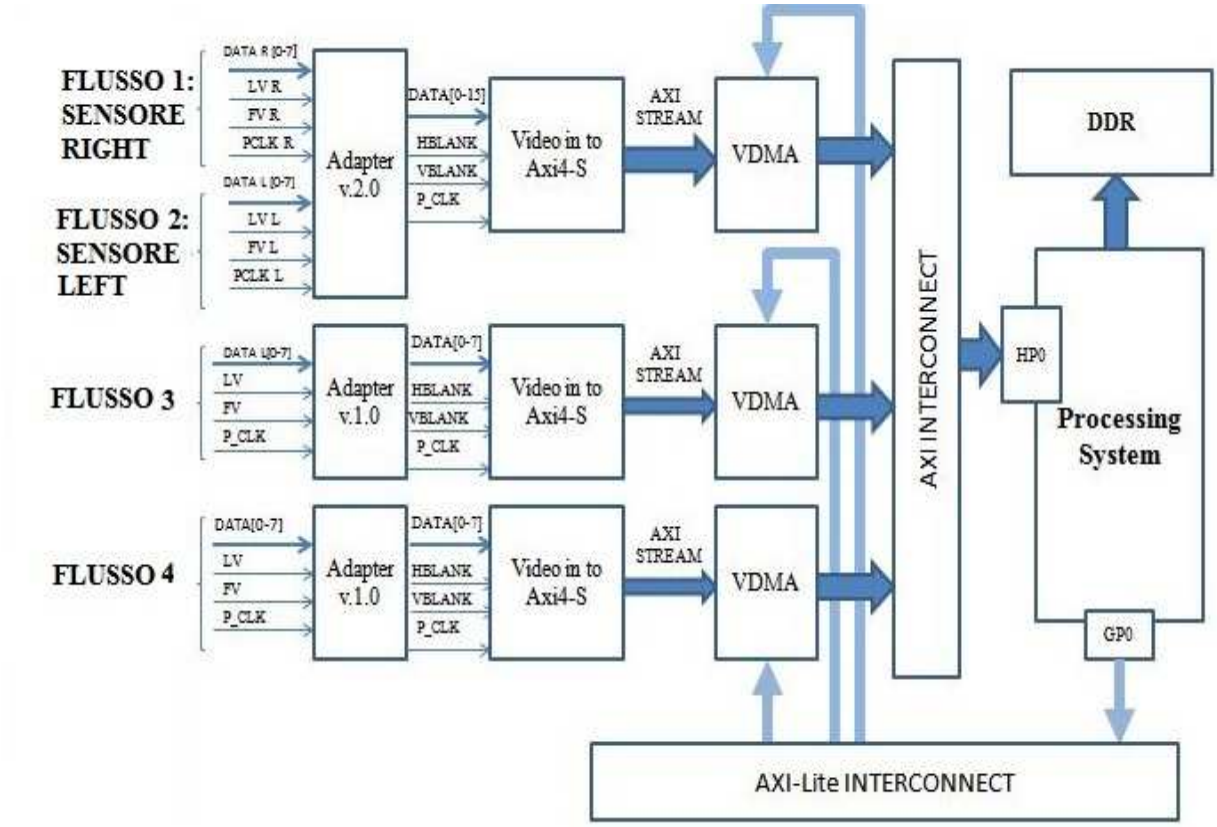


Fig.39: Sistema gestione 4 flussi

4.11 - ACCORGIMENTI PROGETTUALI

La realizzazione di tale sistema ha comportato non poche difficoltà, soprattutto legate all'impossibilità di effettuare simulazione del funzionamento del sistema completo. Tuttavia il simulatore ISim è stato sfruttato sia per verificare la correttezza dei singoli componenti sia per validare versioni ridotte del sistema.

Il problema principale è stato infatti causato dal VDMA, poiché senza essere stato preventivamente programmato, esso non trasferisce dati. Come supporto alla programmazione del VDMA è stato realizzato, nella PL, un rilevatore di fronti di salita, collegato poi al segnale di sincronizzazione *FSync_out*, generato dal VDMA. L'uscita del rilevatore di fronti è stata connessa ad uno dei led della Zedboard, per notificare che il VDMA stesse trasferendo dei frame. La necessità di avere un rilevatore di fronti è nata dal fatto che, il segnale FSync, è attivato per pochi clock, risultando quindi non visualizzabile dal led.

Non è stato poi possibile inviare direttamente verso l'esterno i segnali dell'AXI4 (a differenza di quelli dell'AXI-Stream), poiché questa possibilità non è supportata dall'ambiente di sviluppo, in quanto non ne permette la sintesi.

Come illustrato nel capitolo 3, il progetto mappato nella PL è stato sviluppato sfruttando XPS. Questo tool permette una gestione ad alto livello degli IP-Core, fornendo controlli aggiuntivi alla composizione del sistema; uno di questi riguarda la tipologia dei segnali delle interfacce. Essi sono, infatti, suddivisi in classi, non compatibili tra loro.

Queste classi sono:

- SEGNALI DI RESET
- SEGNALI DI CLOCK
- SEGNALI DI INTERRUPT
- TUTTI GLI ALTRI TIPI DI SEGNALI

Questa distinzione ha causato difficoltà quando è stato utilizzato il wizard *Create and Import Peripheral;* tramite questo tool non è possibile, infatti, cambiare la classe dei segnali del componente che si sta importando, rendendoli di fatto inutilizzabili. Tale limitazione è stata superata modificando autonomamente, mediante un semplice editor di testi, il file *.mpd,* generato automaticamente dai tool e che presenta la lista di tutti i segnali del componente. Ad esempio, per il segnale Pixel Clock in uscita dall'adapter:

```
PORT PCLK_OUT = "", DIR = O
```

Tale riga, deve essere modificata nel seguente modo:

```
PORT PCLK_OUT = "", DIR = O, SIGIS = CLK
```

Una volta effettuata la modifica occorre riavviare XPS, altrimenti tale cambiamento non è riconosciuto dal tool. A questo punto è possibile collegare il pixel clock solo ad un segnale, di un'interfaccia, appartenente alla stessa categoria.

Per la generazione dei clock, necessari al funzionamento del sistema, sono state utilizzate le PLL del Processing System, semplicemente configurabili attraverso il clock wizard, come mostrato nella sezione 3.2.

Durante la fase di test è stato riscontrato un problema relativo al funzionamento del VDMA: esso non scriveva il primo pixel di un frame nel primo indirizzo disponibile. Questo comportamento è stato causato dal fatto che, nel momento in cui era terminata la programmazione del VDMA, esso iniziava a trasferire il flusso di dati, senza attendere l'inizio del frame successivo. Questo problema è stato risolto bloccando il flusso dati, a monte del VDMA, finché non ne è terminata la relativa programmazione. Questa interruzione è stata effettuata sfruttando il segnale di *RESET* del Video In to AXIS. Tale componente, infatti, inizia a trasmettere un frame, tramite l'AXI-Stream, solo dall'inizio, rendendo quindi possibile azionarlo in un qualsiasi momento per avere la garanzia di corretto funzionamento del sistema. Il reset è stato portato all'esterno e collegato ad uno degli switch (SW0) della Zedboard.

Una possibile alternativa, che però non si è ritenuto opportuno implementare, è quella di gestire questo segnale attraverso un comando software, ad esempio assegnandogli un bit in un registro mappato in memoria.

Tra le alternative progettuali valutate, vi è quella di utilizzare più AXI Interconnect e, di conseguenza, anche porte AXI High Performance. Questa soluzione, però, non porta vantaggi per il tipo di flussi video che interessano la tesi e, quindi, si è preferito tenere limitate le risorse della PL occupate. Per raggiungere lo stesso obiettivo è stato utilizzato, per tutte le interfacce AXI, lo stesso clock; richiedendo meno funzionalità al core Interconnect, esso è sintetizzato senza i moduli specifici, necessari alle conversioni di frequenza. Sempre per lo stesso motivo, è stato utilizzato un altro blocco AXI Interconnect, per effettuare l'interfacciamento con i dispositivi del sistema; in particolare, è stata utilizzata l'AXI-Lite, poiché più che sufficiente per leggere e scrivere registri di stato e di controllo.

Terminata ora la descrizione del progetto, da mappare nella logica programmabile, passiamo a descrivere il software integrante necessario al funzionamento del sistema.

5 - PROGETTO SOFTWARE

La parte software del progetto consiste nello sviluppo di un'applicazione bare-metal ed è stata realizzata utilizzando, come descritto nel capitolo 3, il tool Software Development Kit; questo strumento offre potenti mezzi, sia per la stesura del codice, sia per la fase di test e di debug. In particolare, è stato impiegato il Board Support Package (BSP), il quale raccoglie tutti i driver e le ulteriori librerie Xilinx disponibili per il dispositivo.

Il software sviluppato per l'ARM è parte integrante del progetto hardware realizzato nella PL poiché, senza l'opportuna programmazione, i VDMA non sono in grado di gestire i flussi video.

Sono state realizzate, inoltre, opportune funzioni per effettuare la lettura dei frame buffer, utilizzate principalmente per due scopi: verificare il corretto funzionamento del VDMA e valutare le prestazioni dell'ARM. Maggiori dettagli a riguardo verranno dati in seguito.

Una volta terminato lo sviluppo del codice, tramite SDK, viene creato il file *ELF*, eseguibile direttamente sulla scheda, previa programmazione della PL. Questa modalità è stata scelta poiché favorisce la fase di test e di debug, rispetto all'utilizzo di una scheda SD in cui andrebbero, ad ogni nuova modifica, copiati i file *BIT (bitstream)* ed *ELF*. Un ulteriore vantaggio è apportato dalla possibilità di collegare direttamente il terminale di SDK ad una porta seriale, senza adottare un'applicazione specifica (es: TeraTerm), altrimenti necessaria. Questa porta seriale è stata collegata, tramite mini-USB, all'UART (Universal Asynchronous Receiver-Transmitter) dello Zynq,

permettendo così l'invio, da parte del processore, di messaggi e risultati ottenuti; sebbene la sua velocità sia molto ridotta risulta più che sufficiente per il testing e il debug del software.

Si prosegue ora ad illustrare come sia stato possibile implementare il codice in grado di configurare i VDMA, in particolare, per la realizzazione di un buffer circolare, presentando, inoltre, alcuni accorgimenti utilizzati per effettuare una lettura consistente di tali buffer.

Prima di procedere con la descrizione del software sviluppato è fornita una panoramica dello spazio d'indirizzamento del processore, il quale è suddiviso tra tutti i dispositivi del sistema.

5.1 - SPAZIO D'INDIRIZZAMENTO

Gli indirizzi sono resi disponibili al programmatore tramite il file *system.xml*, facilmente accessibile da SDK; gli indirizzi base sono raccolti anche, come costanti ed assieme ad altri parametri, nell'header file *xparameters.h* e, quindi, direttamente utilizzabili nel codice.

Durante la fase di assegnamento degli indirizzi si è dovuto tener conto dei vincoli previsti dallo Zynq: non tutti gli indirizzi, infatti, sono liberamente utilizzabili, come descritto in sezione 2.1. Nella tabella seguente è mostrato lo spazio d'indirizzamento di una delle soluzioni realizzate:

DISPOSITIVO	NOME COSTANTE	INDIRIZZO INIZIALE	INDIRIZZO FINALE	SPAZIO INDIRIZZAMENTO
DDR	DDR_RAM_BASEADDR	0x00000000	0x1FFFFFFF	512 MB
VDMA 0	AXI_VDMA_0_BASEADDR	0x43000000	0x4300FFFF	64 KB
VDMA 1	AXI_VDMA_1_BASEADDR	0x43010000	0x4301FFFF	64 KB
VDMA 2	AXI_VDMA_2_BASEADDR	0x43020000	0x4302FFFF	64 KB
TIMER	TIMER_0_BASEADDR	0x42000000	0x42000FFF	4 KB
UART	UART_0_BASEADDR	0xE0001000	0xE0001FFF	4 KB
GPIO	GPIO_BASEADDR	0xE000A000	0xE000AFFF	4 KB
SDIO 0	SDIO_0_BASEADDR	0xE0100000	0xE0100FFF	4 KB

Tab.4: Spazio d'indirizzamento del sistema realizzato

Il numero di frame da bufferizzare rappresenta uno dei parametri del sistema e, per lo sviluppo del seguente progetto, si è preferito tenerlo contenuto. La bufferizzazione di due soli frame rappresenta un caso limite che, seppur possa essere più che sufficiente per la maggior parte delle applicazioni, si è preferito evitare, optando per la memorizzazione di tre frame.

Dovendo realizzare un sistema flessibile, in grado di supportare dinamicamente il cambiamento del tipo di sensori da collegare, gli indirizzi per i vari frame sono stati presi molto distanti tra loro. Nel caso analizzato, con sensori VGA in grey-scale con un Byte per pixel, per memorizzare un frame di un singolo flusso occorrono 307200 Byte mentre per i flussi stereo unificati 614400 Byte. Questi indirizzi sono mostrati, nel caso dei quattro flussi illustrato nel capitolo precedente, nella tabella seguente:

VDMA	FRAME	INDIRIZZO INIZIALE	INDIRIZZO FINALE
0	0	0x08000000	0x08095FFF
	1	0x09000000	0x09095FFF
	2	0x0A000000	0x0A095FFF
1	0	0x0B000000	0x0B04AFFF
	1	0x0C000000	0x0C04AFFF
	2	0x0D000000	0x0D04AFFF
2	0	0x1D000000	0x1D04AFFF
	1	0x1E000000	0x1E04AFFF
	2	0x1F000000	0x1F04AFFF

Tab.5: Indirizzi frame buffer

5.2 - DRIVER VDMA

Questi driver permettono la realizzazione di funzioni di più alto livello per la gestione del sistema, con particolare riferimento alla programmazione di più VDMA. Il comportamento del VDMA è completamente configurabile, ed è, quindi, possibile modificarne le specifiche a run-time. Oltre alla possibilità di scegliere la modalità di funzionamento desiderata, è possibile scegliere il numero di frame da bufferizzare ed impostare anche gli indirizzi della DDR utilizzati a tale scopo. Anche se questa opportunità non è stata sfruttata in questo lavoro di tesi, occorre notare che il canale di scrittura e quello di lettura di un VDMA non debbono per forza impiegare lo stesso buffer, quindi sono completamente indipendenti. Come questo sia possibile sarà mostrato in seguito, offrendo qualche dettaglio a riguardo.

I Driver di questo dispositivo sono contenuti nel BSP e suddivisi, come tipicamente avviene, in due file:

- **xvdma.h:** contiene la definizione delle strutture dati e delle funzioni
- **xvdma.c:** contiene l'implementazione delle funzioni

Le strutture dati principalmente utilizzate sono:

- **XAxiVdma:** rappresenta l'istanza del VDMA specifico, le informazioni principali in essa contenute riguardano: i canali attivi (lettura e scrittura), le relative strutture che ne contengono la configurazione e l'indirizzo iniziale del componente

- **XAxiVdma_Config**: raccoglie le informazioni riguardanti la configurazione hardware del componente; ogni VDMA deve avere la propria
- **XAxiVdma_DmaSetup:** contiene le informazioni necessarie alla configurazione software del componente per un singolo canale. Alcune di esse riguardano la tipologia del frame buffer, la dimensione dei frame del flusso video e gli indirizzi da utilizzare per la memorizzazione

La prima operazione da compiere consiste nell'inizializzazione del Driver. Successivamente si fanno partire i trasferimenti in DMA e, se fosse necessario, si impostano le routine di risposta agli interrupt. Quest'ultima possibilità non è stata sfruttata nello sviluppo di questo progetto e perciò non è illustrata.

Qui di seguito si mostra ora qual è la sequenza di operazioni da compiere per l'inizializzazione del driver. Occorre, innanzitutto, creare la struttura necessaria alla configurazione del VDMA; può essere fatto manualmente o attraverso la funzione:

XAxiVdma_LookupConfig (Device ID);

Il passo successivo consiste nella vera e propria inizializzazione del driver e del dispositivo, ciò è possibile attraverso la funzione:

XAxiVdma_CfgInitialize (XAxiVdma, XAxiVdma_Config, BaseAddr)

Per alcune possibili configurazioni del VDMA in questa fase occorrono ulteriori operazioni da compiere ma, non interessando direttamente il progetto, non sono illustrate e si può passare direttamente allo step successivo.

Per poter iniziare i trasferimenti in DMA rimangono da compiere alcune importanti sequenze d'istruzioni:

- **XAxiVdma_DmaConfig (XAxiVdma, XAxiVdma_DmaSetup)**: serve alla configurazione del singolo canale del VDMA; le relative informazioni di configurazione sono raccolte nella struttura dati *XAxiVdma_DmaSetup*. Ne esiste una per il canale di scrittura e una per il canale di lettura e, proprio grazie a queste, essi sono completamente indipendenti
- **XAxiVdma_DmaSetBufferAddr (XAxiVdma, ChannelDir, Buffer_Addresses)**: permette di impostare a piacimento gli indirizzi fisici utilizzati per la bufferizzazione dei frame; la struttura dati *Buffer_Addresses* conterrà, quindi, un numero di elementi pari al numero di frame gestiti
- **XAxiVdma_DmaStart (XAxiVdma, ChannelDir)**: con questa, terminata la configurazione, si possono iniziare i trasferimenti

Il driver fornisce anche il supporto necessario alla fase di debug; innanzitutto, tutte le funzioni precedentemente descritte, restituiscono un esito che permette di sapere se l'operazione effettuata è andata a buon fine. Un'altro utile strumento è rappresentato dalla funzione sotto riportata:

XAxiVdma_DumpRegister (XAxiVdma, ChannelDir)

Essa stampa i registri di stato e di controllo del VDMA.

Una dei limiti principali del driver è rappresentato dall'assenza di meccanismi per la gestione della coerenza delle cache dati che deve essere, quindi, sotto la responsabilità dell'applicazione.

5.3 - TIMER

Prima di procedere con la descrizione di come sia possibile estrarre i frame, è illustrato un componente realizzato ad hoc, ed utilizzato per la valutazione delle prestazioni dell'ARM. Questo IP-Core è stato implementato sfruttando la funzionalità di creazione del tool *Create and Import Peripheral Wizard* (esso prepara il template nel quale andare a scrivere la logica custom, occupandosi di realizzare le interfacce del componente). Si è preferito adottare un proprio timer, rispetto all'uso di funzioni standard meno precise, poiché le potenzialità dello Zynq ne hanno permesso la realizzazione con uno sforzo esiguo.

Questo dispositivo non è altro che un counter, azionato da un clock a 100 MHz e gestito attraverso un registro di controllo. Il numero di cicli di clock conteggiati è disponibile in un secondo registro e l'accesso ad entrambi è effettuato tramite l'interfaccia AXI-Lite Slave presentata dal componente. Si mostra, in figura 40, un semplice schema del Timer:

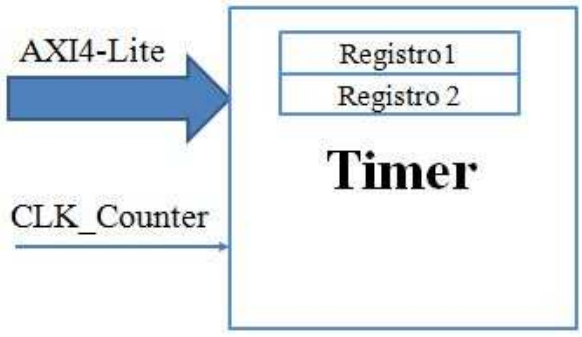


Fig.40: Schema del timer con relativi segnali

Il primo registro è quello di controllo e ne sono utilizzati solo due bit: il primo (bit 0) assume funzione di reset del conteggio e l'ultimo (bit 31) serve per

indicare se il contatore è attivo o meno. Il secondo registro è quello in cui è memorizzato il valore conteggiato; trattandosi di registri a 32 bit (ed essendo azionato da un clock con periodo di 10 ns), questo timer è in grado di conteggiare fino a circa 40 s, valore molto grande rispetto alle esigenze per il quale è utilizzato. Sebbene sia introdotto un leggero ritardo (250-300 ns), causato dalla latenza dell'interconnessione AXI-Lite, tale dispositivo fornisce ottime prestazioni, almeno per il tipo di misurazioni per le quali è stato sfruttato in questo progetto.

Per semplificarne l'adozione nel codice sviluppato, sono state create alcune funzioni di più alto livello:

- resetTimer (TimerAddress)
- startTimer (TimerAddress)
- stopTimer (TimerAddress)
- getElapsedTime (Timer Address)

Una volta inserito nel sistema è possibile utilizzarlo per valutare il tempo impiegato dal processore per eseguire un blocco d'istruzioni, come mostrato nel seguente esempio in pseudo-codice:

resetTimer (TimerAddress);

startTimer (TimerAddress);

PORZIONE DI CODICE DA VALUTARE

stopTimer (TimerAddress);

getElapsedTime (Timer Address);

Tale uso ne è stato fatto per valutare il tempo impiegato dall'ARM, nei vari casi studiati, per l'estrazione di un frame completo.

5.4 - ESTRAZIONE FRAME

L'estrazione dei frame dalla memoria è stata realizzata tramite il processore ARM, cercando di sfruttare il più possibile il parallelismo del bus diretto al memory controller, che è di 32 bit.

Una delle problematiche principali riguarda la sincronizzazione tra ARM e VDMA. Elaborando in maniera asincrona e a velocità completamente diverse, occorre un meccanismo che garantisca la mutua esclusione dell'accesso ai buffer. In particolare, permettendo al processore il prelievo di un frame solo se il VDMA non lo sta scrivendo. A tal proposito, il VDMA rende disponibile l'indice del buffer in cui sta trasferendo, sia attraverso un registro, sia attraverso alcuni segnali che ne rappresentano il valore in binario (al massimo 32 frame memorizzabile e quindi 5 bit); nel caso di lettura effettuata dal processore, risulta sicuramente più comodo sfruttare la prima opzione. Senza dover accedere esplicitamente al registro è possibile utilizzare una funzione di più alto livello, resa disponibile dal Driver del dispositivo:

XAxiVdma_CurrFrameStore (XAxiVdma, ChannelDir)

Tramite tale funzione è possibile realizzare, con una gestione a *polling*, un ciclo iniziale di attesa passiva, evitando così che il processore legga proprio il frame che è scritto in quel preciso momento. Essendo l'ARM molto veloce, rispetto al tempo di arrivo dei frame, prima di passare alla lettura del buffer successivo deve attendere o può, alternativamente, sfruttare questo tempo per

elaborazioni real-time sui frame precedentemente estratti. La situazione è illustrata in figura 41, in cui è considerato il caso peggiore, cioè quando il processore inizia a leggere proprio dal frame che sta scrivendo il VDMA; le temporizzazioni mostrate sono solo indicative, maggiori dettagli a riguardo sono forniti nel capitolo relativo ai risultati sperimentali:

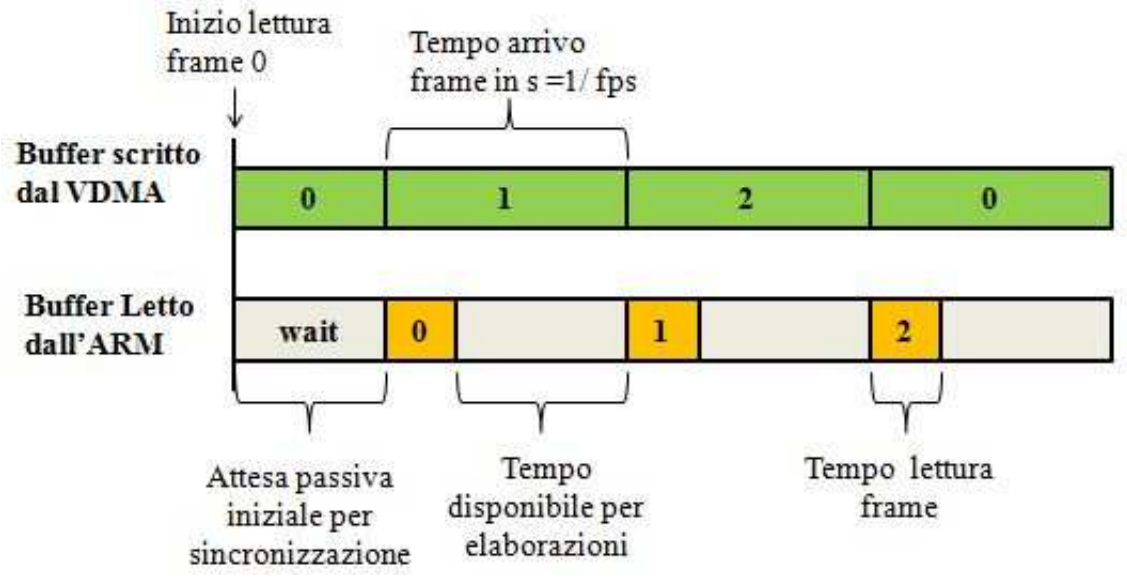


Fig.41: esempio sincronizzazione ARM e VDMA

Come illustrato precedentemente, nella sezione 5.2, i driver del VDMA non garantiscono la coerenza delle cache dati e, la prima soluzione realizzata, non ne prevedeva l'utilizzo. Senza sfruttare le cache, però, le prestazioni del processore non sono soddisfacenti al fine di elaborare successivamente tali frame e, perciò, questa ipotesi è stata abbandonata.

Una volta abilitate le cache, sono stati riscontrati dei problemi: nella maggior parte dei casi la lettura dai buffer non risultava corretta. Presumibilmente, nel momento in cui il processore richiede per la seconda volta un dato, esso è prelevato direttamente dalla cache mentre, nel frattempo, il VDMA lo ha

modificato. Tale problematica è stata risolta aggiungendo un'istruzione di *Flush* delle cache (funzione *Xil_DCacheFlush()*), prima di andare a prelevare, ogni volta, il contenuto di un buffer. In realtà, se si vogliono estrarre consecutivamente tutti i frame, questa operazione è da effettuare solo inizialmente, per avere comunque la garanzia che il dato fornito è sicuramente quello più aggiornato.

I test sull'effettivo funzionamento del VDMA e delle letture dei buffer sono stati effettuati tramite la realizzazione di funzioni di alto livello, in grado di verificare la consistenza dei frame memorizzati. Le immagini generate sinteticamente presentavano, infatti, caratteristiche particolari, proprio per controllare la correttezza del sistema, come spiegato nella sezione relativa. In particolare, utilizzando la versione che genera immagini che hanno ogni frame diverso dall'altro ma ognuno con tutti i pixel uguali, è stato possibile verificare che non fosse perso nessun frame.

E' facile verificare che si presenti una situazione come quella mostrata in figura, nella quale ogni elemento del buffer rappresenta, in realtà, l'insieme di tutti i pixel di un frame:

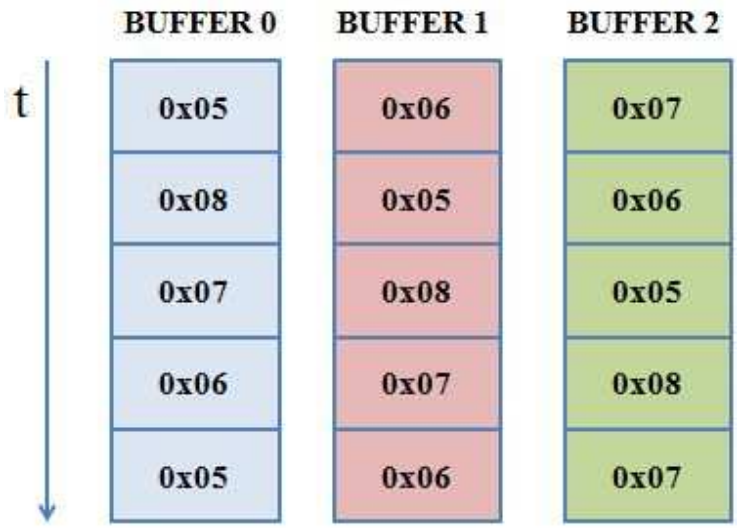


Fig.42: Evoluzione temporale del contenuto del buffer

6 - RISULTATI SPERIMENTALI

Uno degli aspetti principali che si è cercato di valutare, riguarda il tempo in cui il processore è disponibile per eseguire elaborazioni sui frame letti dal buffer. Per questo la maggior parte dell'attenzione è stata rivolta ai test effettuati per conoscere il tempo necessario all'ARM per l'estrazione dei frame dal buffer. Un secondo obiettivo perseguito, è quello di mantenere contenuta l'occupazione di risorse FPGA del sistema realizzato, cercando di analizzare quali sono le scelte migliori, in base all'esigenze specifiche.

Occorre, innanzitutto, specificare che il design implementato, nel caso della gestione di 4 flussi, occupa circa il 10% delle risorse della FPGA dello Zynq; rimangono, quindi, molte celle disponibili per integrare il sistema con ulteriori moduli o con pipeline d'elaborazione dei flussi. Maggiori dettagli sono forniti, per le diverse versioni del progetto, nella seguente tabella:

VERSIONE	DESCRIZIONE SISTEMA	Flip-Flop	LUT	Block Ram
1	FLUSSO SINGOLO	2873	3097	3
2	2 FLUSSI STEREO	2972	3140	3
3	2 FLUSSI	5077	5310	6
4	4 FLUSSI	8308	8176	8

Tab.6: Occupazione risorse FPGA dei diversi sistemi realizzati

I dati forniti riguardano il caso migliore possibile, con tutte le ottimizzazioni effettuate, ma comunque più che sufficiente a gestire la maggior parte dei flussi analizzati nella tesi.

Inoltre, già da questi risultati, è evidente che unificando i flussi dei sensori stereo si ha un notevole risparmio di risorse, rispetto ad una soluzione con gestione separata.

In tabella 7 sono raccolti i dati relativi alla sintesi di alcuni componenti nel sistema, nella versione 4 del sistema:

IP-Core	Flip-Flop	LUT	Block Ram
Axi-Lite	159	304	-
Axi Interconnect	739	780	3
Processing System	-	89	-
VDMA 0	2023	1913	2
Vid In To AXIS 0	132	99	-
Adapter v.2.0	-	3	-
Timer	111	118	-

Tab.7: Risorse PL per IP-Core

Come si può facilmente notare, il VDMA è il componente nettamente più oneroso, in termine di risorse utilizzate, degli altri, nonostante le funzionalità richieste siano minime rispetto alle possibilità che offre. L'altro IP-Core abbastanza oneroso è l'AXI Interconnect del quale, però, ne è presente una singola istanza a 32 bit. In altri casi potrebbe esserne aumentato il parallelismo

(da 32 a 64 bit) o addirittura duplicato. Questo aspetto sarà analizzato meglio alla fine di questo capitolo, dove si discuterà di alcuni accorgimenti adottati, utili all'ottimizzazione del sistema.

Il numero di connessioni AXI e il parallelismo necessari dipende, soprattutto, dalla banda totale richiesta dal sistema, facilmente calcolabile sommando la banda occupata da ogni singolo flusso gestito dal sistema. Comunque, per i sensori utilizzati nel progetto di ricerca, la soluzione con un singolo bus AXI risulta più che sufficiente a soddisfare le esigenze. La banda richiesta da alcuni sensori è riportata in tabella 8:

Formato	Byte pixel	fps=15	fps=25	fps=40	fps=60	fps=75
640x480	1	4.6	7.7	12.3	18.5	23
640x480	3	13.8	23.1	36.9	55.5	69
800x600	1	7.2	12	19.2	28.8	36
800x600	3	21.6	36	57.6	86.4	108
1280x720	1	13.8	23	36.9	55.3	69.1
1280x720	3	41.4	69	110.7	155.9	207.3
1920x1080	1	31.1	51.8	83	124.4	155.5
1920x1080	3	93.3	155.4	249	373.2	466.5

Tab.8: Banda (in MB/s) di alcuni flussi video

Si può facilmente notare come vi sia un netto cambio di esigenze, passando dal caso di sensori a 0,3 Mpixel a quello di sensori da 2 Mpixel: tale

variazione richiede, infatti, variazioni non banali all'architettura per le quali sono d'obbligo ulteriori approfondimenti.

Quindi, nei casi in cui i flussi da gestire superano la banda dell'AXI-Interconnect, si deve modificare il sistema per fare in modo che sia in grado di gestire anche la nuova situazione, considerando però che la banda della memoria è di circa 4GB/s e non ha perciò senso superare tale valore; si mostra ora, in figura 9, la banda fornita dall'AXI, nel caso di parallelismo a 32 bit e diversi valori di frequenza.

Frequenza (MHz)	Banda teorica (MB/s)
50	**200**
100	**400**
150	**300**
200	**400**

Tab.9: Banda AXI

Analizzando la tabella precedente, assieme a quella relativa alle tipologie di flussi, è evidente che per sensori di tipo VGA anche una frequenza inferiore a 50 Mhz sarebbe più che sufficiente; il caso limite, per immagini con un Byte per pixel, corrisponde ad un clock pari al pixel clock diviso per 4.

6.1 - PRESTAZIONI ARM

Si analizzano ora le prestazioni dell'ARM per la lettura di un frame nel buffer. Questo parametro risulta assolutamente fondamentale al fine di valutare il tempo residuo a disposizione di elaborazioni o per la visualizzare delle immagini. Questo tempo, come illustrato nelle sezione 5.4, è quello che intercorre tra la fine della lettura di un frame, da parte dell'ARM, e l'inizio della scrittura, da parte del VDMA, di quello successivo; tale considerazione vale solo nel caso in cui si vogliano compiere le stesse elaborazioni su tutti i frame trasmessi, se ciò non fosse necessario si potrebbe ritardare la lettura del frame successivo, dedicando più tempo all'elaborazione.

Questo tempo dipende, innanzitutto, dalla frequenza di arrivo dei frame, espressa tipicamente in fps (frame per secondo); nella tabella seguente sono forniti alcuni valori tipici per sensori reali:

fps	Periodo Frame (ms)
15	66,6
25	40
40	25
60	16,6
75	13,3

Tab.10: Tempo arrivo di un nuovo frame

L'utilizzo di sensori che forniscono immagini con maggiore frequenza non influisce, chiaramente, sulle prestazioni del processore in fase di lettura.

Occorre, però, avere la garanzia che il sistema sia in grado di gestire questo nuovo flusso, poiché richiede una maggiore larghezza di banda, come discusso nella sezione precedente.

L'altro aspetto da considerare riguarda il tempo necessario al processore per la lettura di un frame dal buffer; esso dipenderà dalla dimensione del frame e dal parallelismo sfruttato per la lettura nella DDR. In tutti i test effettuati sono stati letti blocchi di 4 Byte, non trovando nessun vantaggio nella lettura indipendente dei singoli Byte. Tramite il timer realizzato, ed illustrato nel capitolo precedente, sono state rilevate alcune tempistiche, relative alla lettura di alcuni frame di dimensioni tipiche:

Frame Width	**Frame Height**	**Pixel**	**Tempo impiegato lettura ARM (ms)**
640	480	307.200	**1,1**
800	600	480.000	**1,8**
1280	720	921.700	**3,5**
1920	1080	2.073.600	**7,8**
1600	1200	1.920.000	**7,2**

Tab.11: Tempo necessario all'ARM per l'estrazione di un frame

Tutti questi tempi si riferiscono a prove effettuate con le cache dati attive; senza utilizzarle, infatti, le tempistiche sono più grandi di un ordine di grandezza, non risultando idonee ad una successiva elaborazione dei frame.

Non sono state valutate tutte le possibilità poiché, infatti, risulta abbastanza semplice riuscire a calcolarsi questo tempo in altre situazioni. In tutti i casi riportati, ad esempio, sono state considerate immagini con il valore di un pixel

rappresentato da un singolo Byte. Se si utilizzano formati diversi, è sufficiente moltiplicare il tempo fornito in tabella con il numero di Byte per pixel nella nuova configurazione. Anche il caso dei flussi provenienti dai sensori stereo non è presentato, ma è semplicemente ottenibile moltiplicando per un fattore 2 il tempo necessario alla lettura di un frame singolo della stessa dimensione.

I tempi misurati sono stati utilizzati per il calcolo dell'intervallo di tempo l'acquisizione di due frame consecutivi in cui l'ARM è disponibile per le elaborazioni real-time e sono forniti in tabella 12:

Formato	Byte per pixel	fps=15	fps=25	fps=40	fps=60	fps=75
640x480	1	65.5	38.9	23.9	15.5	12.1
640x480	3	63.3	36.7	21.7	13.3	10.9
800x600	1	64.8	38.2	23.2	14.8	11.4
800x600	3	61.4	34.6	19.6	11.2	7.8
1280x720	1	63.1	36.5	21.5	13.1	9.7
1280x720	3	56.1	29.5	14.5	6.1	2.7
1920x1080	1	58.5	32.2	17.2	8.8	5.4
1920x1080	3	43.2	16.6	1.6	-	-

Tab 12: Tempo (ms) disponibile al processore per elaborazioni tra due frame consecutivi

Nel caso di maggior interesse, che riguarda direttamente il progetto di ricerca, i sensori sono di tipo VGA, con un Byte per pixel e a 60 fps; si può notare che il tempo disponibile per elaborare real-time ogni singolo frame è di circa 15 ms che, viste le potenzialità dell'ARM, permette di applicare molte tipologie di algoritmi.

Nei casi intermedi (formati medio-grandi soprattutto con fps elevati) tale tempo non consente di eseguire elaborazioni, se non la semplice visualizzazione.

Se si considerano i casi limite il tempo necessario all'ARM può anche non essere sufficiente alla lettura o comunque non permettere la successiva elaborazione. Tuttavia, questo non rappresenta una limitazione in senso assoluto perchè è possibile anche elaborare un frame ogni N. Sebbene il caso ideale di N=1 sia desiderabile, esistono molti contesti applicativi nei quali tale valore può essere maggiore.

6.2 - OTTIMIZZAZIONI AXI

Si passa ora ad effettuare qualche considerazione riguardo al bus AXI4, presente tra i VDMA e il PS, poiché è l'unico da configurare con attenzione per avere buone prestazioni del sistema. L'AXI-Lite, infatti, non influisce, visto che il suo utilizzo è sporadico e per singoli trasferimenti mentre l'AXI-Stream, lavorando alla stessa frequenza di arrivo dei dati, garantisce sempre la trasmissione.

Come illustrato nel capitolo 4, la connessione tramite l'AXI4 avviene sfruttando l'IP-Core AXI Interconnect; questo IP-Core è in grado di gestire più master e più slave, anche nel caso in cui presentino interfacce con frequenza diversa e con differente parallelismo; tutto ciò a discapito delle risorse richieste per la sua sintesi. Sono stati effettuati dei test sull'IP-Core utilizzato per realizzare l'AXI-Lite, valutando proprio l'incidenza che hanno alcuni moduli del componente sull'occupazione di risorse della logica riconfigurabile:

DESCRIZIONE	FF	LUT
Stesso Clock per Interconnect e tutti gli slave	159	305
Uno slave con clock diverso	412	372
Due slave con clock diverso	586	435
Clock Interconnect diverso da quello di tutti gli slave	1278	659

Tab.13: Influenza del clock delle interfacce

Questi risultati evidenziano come la necessità di convertire i clock influisca negativamente sulla quantità di risorse richieste ed è, quindi, opportuno evitare tale situazione, a meno che non sia strettamente necessario.

Come illustrato in 6.1, bisogna valutare attentamente quali sono i casi in cui si deve aumentarne il parallelismo dell'AXI4 o utilizzare più connessioni e, di conseguenza, sfruttare più porte High-Performance del Processing System. Nella tabella seguente è mostrato quanto incidono i cambiamenti relativi all'aumento del numero di connessioni:

NUMERO FLUSSI	NUMERO AXI INTERCONNECT	Flip-Flop	LUT	Block Ram
2	1	5077	5310	6
2	2	5449	5461	6
4	1	8308	8176	8
4	4	9045	8587	8

Tab.13: Influenza del numero di AXI-Interconnect sulle risorse PL

Appare evidente che l'aumento del numero di AXI Interconnect utilizzati comporta un aumento del 10% delle risorse di FPGA richieste; lo stesso aumento si ha nel caso di passaggio ad un parallelismo di 64 dove, però, sono altri i componenti che fanno aumentare le risorse occupate (principalmente VDMA e Processing System).

7 - CONCLUSIONI

Al termine della tesi è stato realizzato un sistema in grado di gestire flussi video sfruttando le potenzialità di una nuova classe di circuiti riconfigurabili, la famiglia Zynq-7000 All Programmable SoC.

Grazie all'utilizzo di un DMA, è stato possibile trasferire i frame di un flusso nelle DDR3, realizzando così un frame buffer circolare. Dal caso singolo si è esteso il progetto alla gestione di un numero maggiore di flussi, completamente indipendenti tra loro.

Tale sistema è stato implementato e testato su una scheda (ZedBoard) che dispone di un componente di fascia intermedia della famiglia Zynq; questo chip si è dimostrato idoneo alla gestione di molte tipologie diverse di flussi video, tipicamente utilizzati in ambito industriale e accademico. La medesima soluzione realizzata con due chip distinti avrebbe difficilmente permesso di ottenere una piattaforma con la stessa rapidità nei trasferimenti tra FPGA e CPU con simili consumi di potenza.

Nel caso dei flussi provenienti dai sensori impiegati nel progetto di ricerca, le prestazioni del sistema risultano molto buone; il processore dispone, infatti, di un tempo sufficiente, tra un frame e il successivo, per eseguire tipiche elaborazioni real-time sui flussi video; anche l'occupazione delle risorse della logica programmabile è molto contenuta, permettendo quindi l'integrazione di altri sistemi o pipeline che effettuino elaborazioni direttamenta sulla logica della FPGA.

Nonostante l'attenzione sia stata focalizzata su sensori VGA con immagini in grey-scale, uno degli aspetti più interessanti del sistema realizzato riguarda proprio la capacità di poterlo adattare, abbastanza velocemente, a nuove tipologie di sensori, a patto che presentino la stessa interfaccia di output. Tale vincolo sarebbe in ogni caso facilmente superabile con una opportuna rete logica di adattamento implementabilie sulla FPGA. Per questo il sistema progettato è facilmente adattabile sia al cambiamento del formato delle immagini, sia a una diversa rappresentazione dei pixel. Per la maggior parte dei flussi analizzati, è sufficiente riconfigurare il VDMA mentre, in alcuni casi, si dovrà anche modificare l'architettura del sistema, al fine di poter garantire una banda sufficiente alla gestione dei nuovi flussi. Si sarebbe potuto realizzare un sistema più generale ma, per mantenere limitate le risorse necessarie al mapping su FPGA, è stato realizzato un sistema ottimizzato per il caso di maggior interesse.

7.1 - SVILUPPI FUTURI

Sebbene il progetto realizzato soddisfi a pieno i requisiti prefissati, consentendo di sfruttare l'ARM per elaborare in real-time i flussi video forniti in input dalla logica programmabile, potrebbe in futuro essere esteso con alcune interessanti funzionalità.

La prima riguarda la possibilità di visualizzare i flussi acquisiti attraverso la logica programmabile, una prima possibile soluzione potrebbe essere quella di sfruttare l'interfaccia HDMI della Zedboard per la visualizzazione delle immagini. In questa eventualità, si dovrebbe utilizzare il canale di lettura di uno dei VDMA, oltre a dover aggiungere ulteriori componenti, in grado di gestire tale flusso e capaci di generare direttamente i segnali necessari all'interfaccia HDMI. Grazie alla flessibilità del VDMA sarebbe relativamente semplice riconfigurarlo dinamicamente, facendo in modo di riuscire a passare alla visualizzazione di un altro flusso video.

Inoltre, allo stato attuale, il sistema realizzato utilizza solo uno dei due *core* dell'ARM e si potrebbe utilizzare anche l'altro per effettuare elaborazioni sulle immagini, aumentando notevolmente la potenza di calcolo della piattaforma considerata.

Un'altra possibile modifica funzionale riguarda l’inserimento di un sistema operativo sulla/e CPU; questa soluzione permetterebbe di avere a disposizione una vera e propria piattaforma di lavoro con, integrato, il sistema hardware che gestisce i flussi.

Le modifiche appena descritte estenderebbero ulteriormente la spettro di utilizzo del sistema progettato nell’ambito di questa tesi.

Bibliografia

[1] *www.zedboard.org*

[2] *http://www.xilinx.com/content/xilinx/en/products/silicon-devices/soc/zynq-7000.html*

[3] *Xilinx, ds190-Zynq-7000-Overview.pdf*

[4] *ARM, Bus Axi specs.pdf*

[5] *http://whatis.techtarget.com/definition/IP-core-intellectual-property-core*

[6] *http://www.xilinx.com/content/xilinx/en/products/silicon-devices/soc/zynq-7000.html*

[7] *Xilinx, xst.pdf*

[8] *Xilinx, est_rm.pdf*

[9] *Aptina, mt9d112_misoc-2020_full.pdf*

[10] *Xilinx, axi_reference_guide.pdf*

[11] *Xilinx, ds768_axi_interconnect.pdf*

[12] *Xilinx, pg020_axi_vdma.pdf*

[13] *Xilinx, pg043_v_vid_in_axi4s.pdf*

[14] *http://www.cs.indiana.edu/hmg/le/project-home/xilinx/ise_13.2/ISE_DS/ISE/coregen/ip/xilinx/dsp/com/xilinx/ip/v_vdma_v1_1/delivery/doc/xvdma_8c.html*

[15] *Xilinx, xapp742-axi-vdma-reference-design.pdf*

[16] *Xilinx, ug934_axi_videoIP.pdf*

[17] *Xilinx, axi_lite_ipif_ds765.pdf*

[18] *Xilinx, xapp792.pdf*

INDICE DELLE IMMAGINI

INDICE DELLE TABELLE