Università della Calabria

Dipartimento di ingegneria Informatica, Modellistica, Elettronica e Sistemistica

Corso di Laurea
triennale in Ingegneria
Informatica

Tesi di Laurea

PROGETTAZIONE E CARATTERIZZAZIONE DI UN SISTEMA EMBEDDED PER IL CALCOLO APPROSSIMATO DELLA FUNZIONE ESPONENZIALE

Relatore
Prof.ssa Stefania Perri

Candidato
Antonella Fortuna
Matricola 200831

Antonella Fortuna

Anno Accademico 2021 / 2022

Sommario

Introduzione 3
Capitolo 1. Motivazioni 4
Capitolo 2. Approssimazione della funzione esponenziale 8
Soluzione Software 8
Soluzione Hardware 11
Capitolo 3. Piattaforma e tools utilizzati 13
Scheda FPGA PYNQ-Z2 13
XILINX SDK 14
VIVADO DESIGN SUITE 17
Capitolo 4. Risultati 18
Conclusione 25
Riferimenti 26

Introduzione

Euclide nella sua opera l'"Ottica" propose quattordici postulati sulla percezione degli oggetti. In particolare, in uno di essi scrisse

"*Gli oggetti che si vedono con più angoli si distinguono più chiaramente*" [1]

Con questo principio Euclide volle far capire che: c'è un distinzione importante tra le cose "come sono" e le cose "come appaiono" e solo guardando un oggetto da più prospettive si può vederlo nella sua interezza e ricreare una sua proiezione tridimensionale. Questo pensiero di Euclide si pone alla base della stereovisione; esso si riflesse dapprima nell'arte dopodiché nella scienza e nell'informatica, e ne conseguirono esperimenti e ricerche che hanno portano a numerose invenzioni, tra cui la ricostruzione 3D e la costruzione di una mappa dell'ambiente per i robot.

In questo elaborato si descrive e si elabora una soluzione al problema del calcolo della funzione esponenziale, mediante un apposito sistema embedded; tale funzione è utilizzata nella stereo visione per la computazione della distanza tra pixels appartenenti a due immagini differenti che ritraggono lo stesso soggetto permettendo così di ricostruire l'oggetto desiderato. La funzione esponenziale pur essendo necessaria è molto dispendiosa da calcolare così com'è, in particolare per l'hardware, l'obiettivo è creare una sua semplificazione efficiente. Da notare che nei sistemi embedded non è importante solo il codice ma anche la piattaforma utilizzata, una sezione dell'elaborato sarà perciò dedicata alla motivazione dietro la scelta di una scheda fpga per la costruzione del sistema. Infine, verranno illustrati i risultati ottenuti facendo anche uso delle funzionalità che il tool Vivado dà a disposizione.

Capitolo 1. Motivazioni

Il riconoscimento facciale, i sistemi di sorveglianza e i veicoli autonomi sono alcuni esempi di come la tecnologia ogni giorno si propone di semplificare e migliorare la vita di tutti. Quello che accomuna questi esempi è la stereo visione, una tecnica che ha come obiettivo quello di eguagliare il sistema visivo umano, ovvero, di ricomporre attraverso segnali digitali l'ambiente circostante e di poterlo fare in tempo reale. I principali strumenti di cui fa uso questa tecnica sono due: gli algoritmi che ponderano la profondità da foto scattate da differenti angolazioni e l'uso della geometria per ricostruire il più fedelmente possibile l'immagine tridimensionale [2].

Come nella visione binoculare umana anche nella computer stereo visione si hanno due telecamere posizionate parallelamente l'una all'altra che immortalano il soggetto desiderato fornendo due immagini.

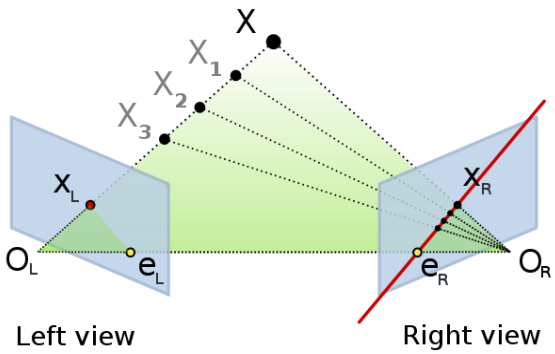

Figura 1 prospettiva di due camere nella stereovisione

Il primo passo da svolgere quando si vuole ricostituire un'immagine tramite la stereo visione è quello di rettificare i prodotti delle telecamere in modo da compensare l'imperfetto allineamento tra le due telecamere e le distorsioni geometriche.

Successivamente si calcola la disparità tra i matching pixels che compongono le immagini. Date due immagini, la disparità è la distanza che sussiste per ogni coppia di pixels che si riferiscono allo stesso punto 3D e l'immagine che scaturisce dalla disparità tra ogni coppia di matching pixels è detta mappa di disparità; essa è il fulcro dei sistemi stereo visivi.

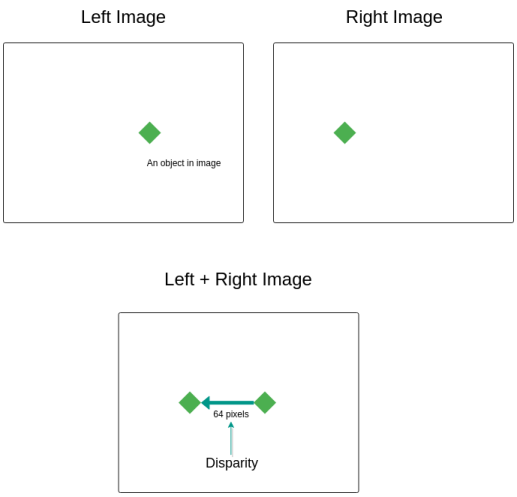

Figura 2 disparità tra due pixels

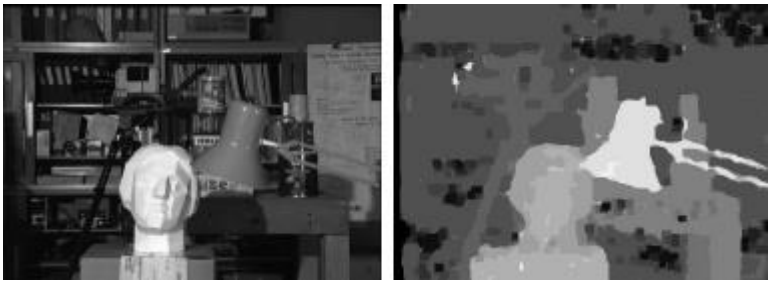

Figura 3 a sinistra l'immagine originale, a destra la disparity map

Numerosi sono gli algoritmi che possono essere utilizzati per calcolare la mappa di disparità e ognuno di essi ha pro e contro. L'algoritmo preso come riferimento in questo elaborato è frutto di due algoritmi, il Support Local Binary Pattern (SLBP) e l'Adaptive Census Transform (ACT). Esso prevede di suddividere le due immagini in matrici dette finestre e di calcolare per ognuna di esse dei vettori per il cui calcolo si prendono come riferimento i pixels della finestra. Sia W l'altezza e la larghezza della finestra, l'approccio SLBP prevedeva di avere W=3 e di calcolare W^2 vettori, il nuovo algoritmo invece elabora finestre di dimensioni più grandi e per non sovraccaricare il sistema calcola $4W - 3$ vettori anziché W^2, utilizzando come pixels di riferimento solo quelli collocati nelle bisettrici e nelle diagonali della finestra.

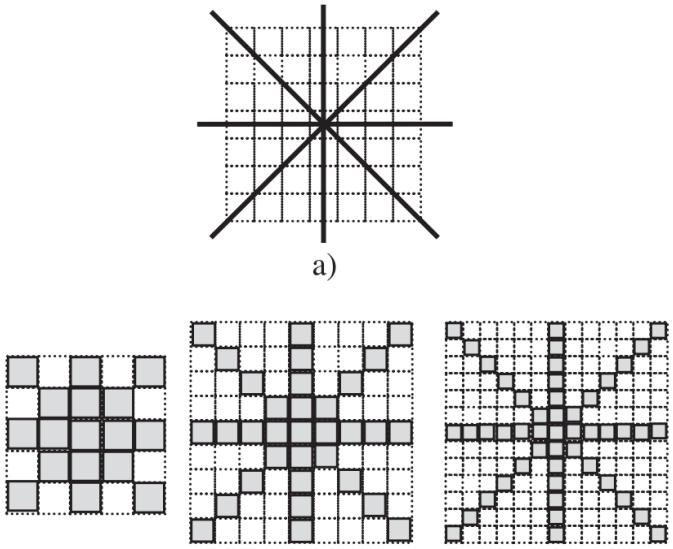

Figura 4 pixels da visitare

Gli elementi dei vettori vengono calcolati con l'approccio ACT, ovvero, si paragona il pixel centrale P della finestra con ogni suo vicino R. Il k-esimo elemento del vettore è uguale a $-w(R,P)$ se il pixel designato come vicino è minore del pixel centrale, altrimenti è uguale a $w(R,P)$.

$w(R,P)$ è il peso di supporto relativo al pixel R ed è calcolato come

$$w(R,P) = e^{-(\frac{\Delta c}{y_c}+\frac{\Delta g}{y_p})}$$

Equazione 1

dove Δc e Δg sono rispettivamente la differenza colorimetrica e spaziale tra P ed R, e y_c e y_p sono costanti.

Una volta calcolati i vettori per l'immagine sinistra e per l'immagine destra, viene calcolato il matching cost di ogni pixel come somma delle differenze assolute di ogni elemento del vettore.

Dalla combinazione di matching costs e support weights si ottiene la dissimilarità che servirà a calcolare la disparità di ogni coppia di pixel. Infatti, la disparità sarà data dalla minor dissimilarità calcolata tra le dissimilarità del pixel dell'immagine destra e quello dell'immagine sinistra.

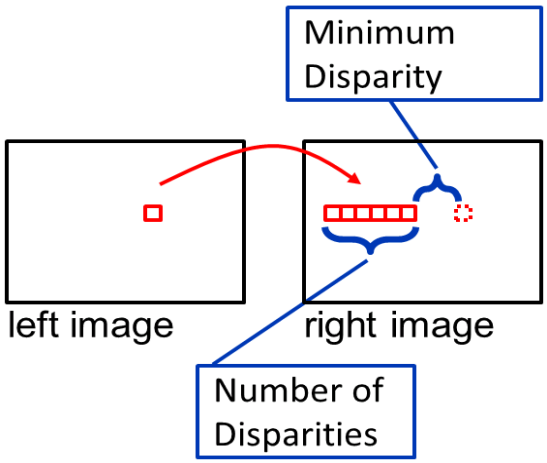

Figura 5 calcolo della disparità

Così com'è stato descritto quest'algoritmo non è hardware friendly, per far in modo di favorire la realizzazione hardware bisogna semplificare la funzione esponenziale presente in Equazione 1. Come verrà dimostrato in Capitolo 2, essa può essere facilitata senza trascurare l'accuratezza facendo uso di una funzione piecewise [3].

Capitolo 2. Approssimazione della funzione esponenziale

Il compito di questo elaborato, come precedentemente trattato, è di creare un sistema embedded da utilizzare all'interno di un calcolatore più grande che semplifichi e acceleri il calcolo della funzione esponenziale descritta in Equazione 1.

Prima di procedere con la semplificazione bisogna fare delle constatazioni: come specificato in [3] la differenza spaziale Δg può essere trascurata e considerando che in un'immagine a toni grigi, Δc è il valore assoluto delle intensità dei pixels e sia γ_c pari a 4, il valore massimo di $\frac{\Delta c}{\gamma_c}$ è 16 . Il peso di supporto diventa quindi pari a:

$$w(R,P) = e^{-(\frac{\Delta c}{4})}$$

Equazione 2

E verrà semplificato come segue

$$w(R,P) = 4 \times \begin{cases} 16 & | \quad \lfloor \Delta c/\gamma_c \rfloor = 0 \\ 12 & | \quad \lfloor \Delta c/\gamma_c \rfloor = 1 \\ 8 & | \quad 2 \leq \lfloor \Delta c/\gamma_c \rfloor < 4 \\ 4 & | \quad 4 \leq \lfloor \Delta c/\gamma_c \rfloor < 8 \\ 2 & | \quad 8 \leq \lfloor \Delta c/\gamma_c \rfloor < 12 \\ 1 & | \quad \lfloor \Delta c/\gamma_c \rfloor \geq 12 \end{cases}$$

Figura 6 esponenziale sottoforma di funzione piecewise

Soluzione Software

Supponiamo che l'intensità di ogni pixel possa essere descritta da un vettore a 8 bit, Δc varierà tra 0 e 255. Per ogni possibile valore di Δc calcoliamo $\frac{\Delta c}{4}$ e confrontiamo il risultato ottenuto in un blocco if-elseif-else come indicato in Figura 1 ottenendo un risultato parziale che verrà moltiplicato per 4 in modo da raggiungere il risultato finale. Da notare che l'approssimazione della funzione esponenziale è pari a 64 volte la funzione esponenziale precisa.

```
XTime_GetTime(&ti1);
expprec=64*exp((double)-deltac_4);
XTime_GetTime(&tf1);
```

Figura 7 funzione esponenziale precisa

```
XTime_GetTime(&ti2);
if((int)deltac_4==0){expapprox=16;}
else if((int)deltac_4==1){expapprox=12;}
else if(deltac_4>=2 && deltac_4<4){expapprox=8;}
else if(deltac_4>=4 && deltac_4<8){expapprox=4;}
else if(deltac_4>=8 && deltac_4<12){expapprox=2;}
else if(deltac_4>=12 && deltac_4<16){expapprox=1;}
else expapprox=0;

expapprox=expapprox*4;
XTime_GetTime(&tf2);
```

Figura 8 funzione esponenziale approssimata

Per ogni valore di Δc viene fatto partire un timer che si fermerà al raggiungimento del risultato di ogni esponenziale calcolato. Il timer XTime_GetTime è un contatore appartenente alla libreria xtime_l.h di C utilizzato con processori di tipo Cortex-A9 e viene incrementato ogni due cicli di processore, motivo per cui i risultati ottenuti dovranno essere moltiplicati per 2 in modo da avere il numero di cicli di clock effettivi. Sarà questo contatore ad aiutarci a stabilire la velocità alla quale le due funzioni vengono calcolate.

```
totclockp+=(tf1-ti1);

if(numclockp<tf1-ti1){
      numclockp=tf1-ti1;
      deltatempop=deltac;
}
```

Figura 9 velocità funzione esponenziale precisa

```
totclocka+=(tf2-ti2);

if(numclocka<tf2-ti2){
	numclocka=tf2-ti2;
	deltatempoa=deltac;
}
```

Figura 10 velocità funzione esponenziale approssimata

```
//tempi medii
//in alcuni punti moltiplico per due perché il contatore di XGetTime si
//incremente ogni due clock di processore
totclockp*=2;
totclocka*=2;
totclockp/=256;
totclocka/=256;
```

Figura 11 calcolo velocità media

Dopo aver calcolato la funzione esponenziale precisa e la funzione esponenziale approssimata vengono controllati i loro valori e comparati in modo da stabilire il livello di attendibilità dei risultati.

```
	differr=fabs(expprec-expapprox);

	errmedio+=differr;

	if(differr>maxerr){
		maxerr=differr;
		deltacmax=deltac;
	}

	if(differr<=minerr){
		minerr=differr;
		deltacmin=deltac;
	}

}
errmedio/=256;
```

Figura 12 calcolo degli errori

Soluzione Hardware

La progettazione di un circuito per il calcolo approssimato della funzione esponenziale può essere descritta dal diagramma a blocchi in Figura 13.

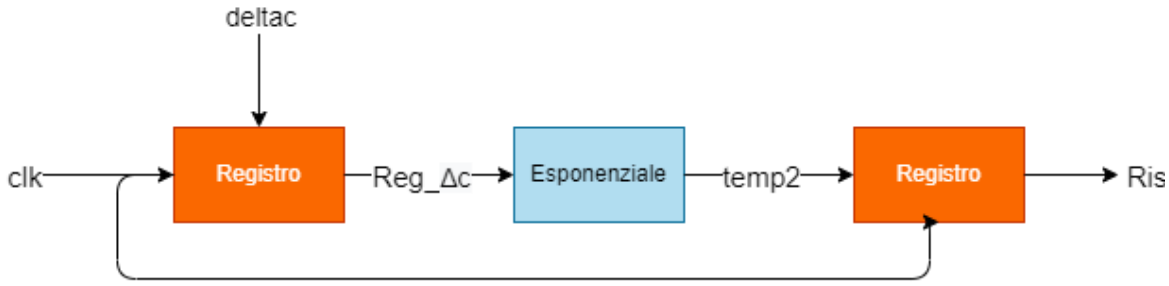

Figura 13 schema a blocchi del circuito

Il modulo prende in ingresso Δc, un vettore a 8 bit, che viene inserito in un registro, tale vettore sarà mantenuto nel registro fino al rising edge del segnale di clock, dopodiché sarà inviato al di fuori del registro dove verrà impiegato nei calcoli che saranno eseguiti per sviluppare l'approssimazione della funzione esponenziale. A questo proposito, si procede computando $\frac{\Delta c}{4}$ tramite l'operazione di shifting di due posizioni verso destra effettuata sul vettore Δc, e l'esito sarà comparato in un blocco when-else come mostrato in [4] in modo da ottenere il risultato parziale. Shiftando di due posizioni verso sinistra il prodotto del passo precedente si giungerà all'approssimazione della funzione esponenziale, essa sarà inoltrata in un registro e verrà trasmessa come output allo scoccare del rising edge del segnale di clock.

```
    Port (
           deltac : in STD_LOGIC_VECTOR (7 downto 0);
           clk : in STD_LOGIC;
           ris : out STD_LOGIC_VECTOR (7 downto 0));
end fakexp;

architecture Behavioral of fakexp is
    signal Rdeltac : STD_LOGIC_VECTOR (7 downto 0);
    signal deltac_4 : STD_LOGIC_VECTOR (7 downto 0);
    signal temp : STD_LOGIC_VECTOR(7 downto 0);
    signal temp2 : STD_LOGIC_VECTOR(7 downto 0);
begin

    deltac_4(7 downto 0)<="00"&Rdeltac(7 downto 2);
    temp<= conv_std_logic_vector(0,8) when (deltac_4(4)='1' or deltac_4(5)='1') else
           conv_std_logic_vector(1,8) when (deltac_4(3)='1' and deltac_4(2)='1') else
           conv_std_logic_vector(2,8) when (deltac_4(3)='1' and deltac_4(2)='0') else
           conv_std_logic_vector(4,8) when (deltac_4(2)='1') else
           conv_std_logic_vector(8,8) when (deltac_4(1)='1') else
           conv_std_logic_vector(12,8) when (deltac_4(0)='1') else
           conv_std_logic_vector(16,8) when (deltac_4(0)='0') else
           "XXXXXXXX";
     temp2<=temp(5 downto 0)&"00";

     process(clk)
        begin
        if rising_edge(clk) then
            Rdeltac<=deltac;
            ris<=temp2;
        end if;
     end process;

end Behavioral;
```

Figura 14 funzione esponenziale approssimata in VHDL

In Figura 15 è raffigurato lo schematic del circuito sopra descritto: i rettangoli all'inizio e alla fine di esso rappresentano i registri, o per meglio dire strumenti che memorizzano per un determinato periodo le informazioni inserite in essi; i trapezi invece delineano i multiplexer, dei selettori che insieme alle porte and e or costituiscono il blocco when-else e le sue condizioni entrambi fondamentali per la realizzazione della funzione.

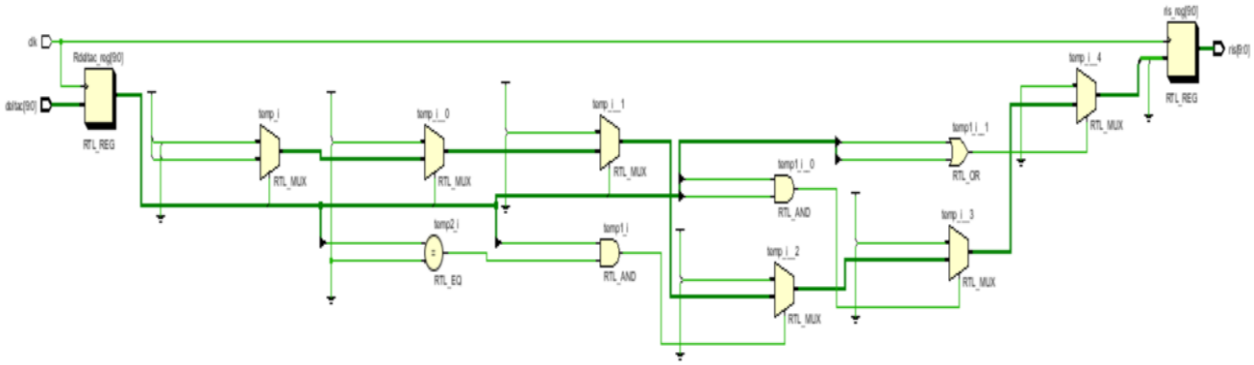

Figura 15 schematic del circuito

Capitolo 3. Piattaforma e tools utilizzati

Per realizzare un sistema embedded si utilizzano sia dispositivi fisici che tools digitali, per questo sistema embedded sono stati utilizzati:

- Una scheda FPGA PYNQ-Z2
- Xilinx SDK
- Vivado Design Suite

Scheda FPGA PYNQ-Z2

Per i loro requisiti, gli algoritmi per sistemi real-time sono spesso implementati utilizzando hardware appositi come: digital signal processors (DSPs), graphics processing units (GPUs), application specific integrated circuits (ASICs), and field programmable gate arrays (FPGAs). I processori di segnali digitali (DSP) hanno molta più potenza dei processori general purpose ma sono limitati dalla sequenzialità. Le GPUs permettono il parallelismo delle operazioni e questo le rende un'ottima scelta per l'implementazione di complessi algoritmi di ottimizzazione globale, ma utilizzano troppa potenza. Una soluzione efficiente è l'utilizzo di schede FPGA, esse permettono il parallelismo ed hanno una grande potenza computazionale [5].

La scheda PYNQ-Z2 (Figura 16) è una scheda di sviluppo FPGA realizzata da Xilinx che opera con il framework opensource PYNQ per la realizzazione di applicazioni embedded.

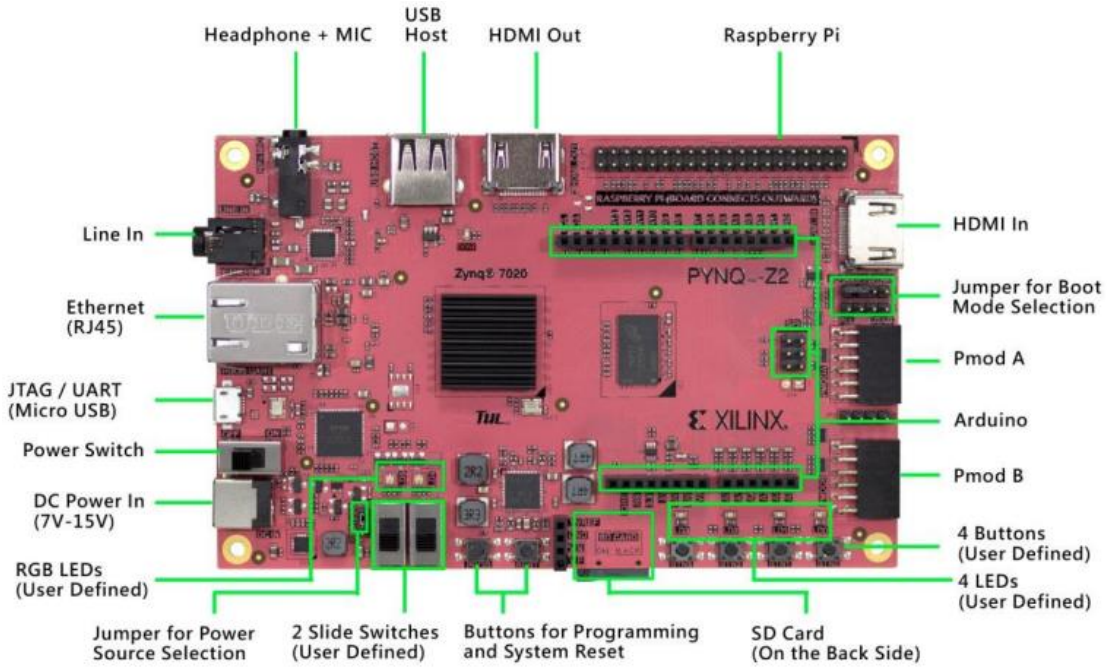

Figura 16 Scheda FPGA PYNQ-Z2

Le specifiche della scheda sono illustrate nella seguente figura.

	PYNQ-Z2
Device	Zynq Z7020
Memory	512MB DDR3
Storage	MicroSD
Video	HDMI In & Out ports
Audio	ADAU1761 codec with HP + Mic, Line in
Network	10/100/1000 Ethernet
Expansion	USB host (PS)
GPIO	1x Arduino Header
	2x Pmod*
	1x RaspberryPi header*
Other I/O	6x user LEDs
	4x Pushbuttons
	2x Dip switches
Dimensions	3.44” x 5.51” (87mm x 140mm)
Webpage	**TUL PYNQ-Z2 webpage**

Figura 17 caratteristiche scheda FPGA PYNQ-Z2

XILINX SDK

Xilinx Software Development Kit (SDK): è l'ambiente di progettazione integrato per la creazione di applicazioni embedded su microprocessori Xilinx. L'SDK si basa sullo standard open source Eclipse. Le funzionalità dell'SDK includono:

- Editor di codice C/C++ ricco di funzionalità e ambiente di compilazione;
- Gestione di progetto;
- Configurazione della build dell'applicazione e generazione automatica di Makefile;
- Errore di navigazione;
- Ambiente ben integrato per il debug e la profilazione senza interruzioni degli obiettivi incorporati;
- Controllo della versione del codice sorgente;
- Analisi delle prestazioni a livello di sistema;
- Strumenti speciali mirati per configurare FPGA;
- Creazione di immagini di avvio;
- Programmazione Flash;
- Strumento da riga di comando con script.

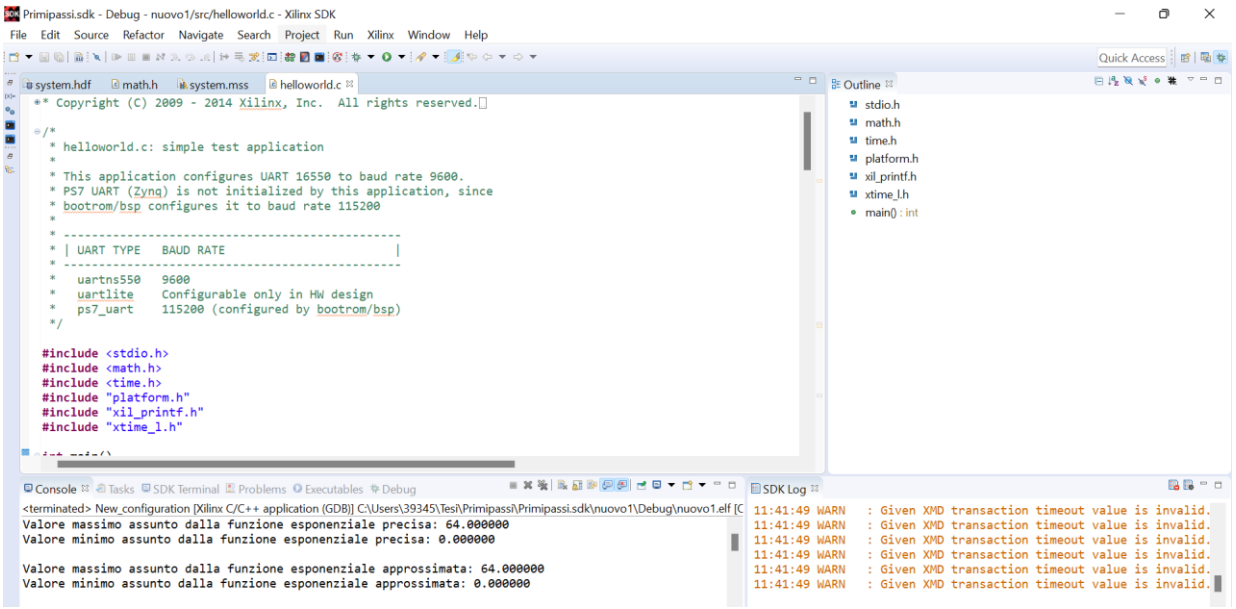

Figura 18 interfaccia Xilinx SDK

In Figura 18 viene mostrata l'interfaccia della piattaforma Xilinx SDK. Essa è l'ambiente di sviluppo attraverso la quale l'utente programma la scheda FPGA. Per far ciò, mediante il tool Vivado viene generato un file di Bitstream che contiene tutte le informazioni sulla scheda FPGA utilizzata. Una volta collegata la scheda al PC tramite collegamento USB, in essa viene inserito il file di Bitstream come mostrato in Figura 19.

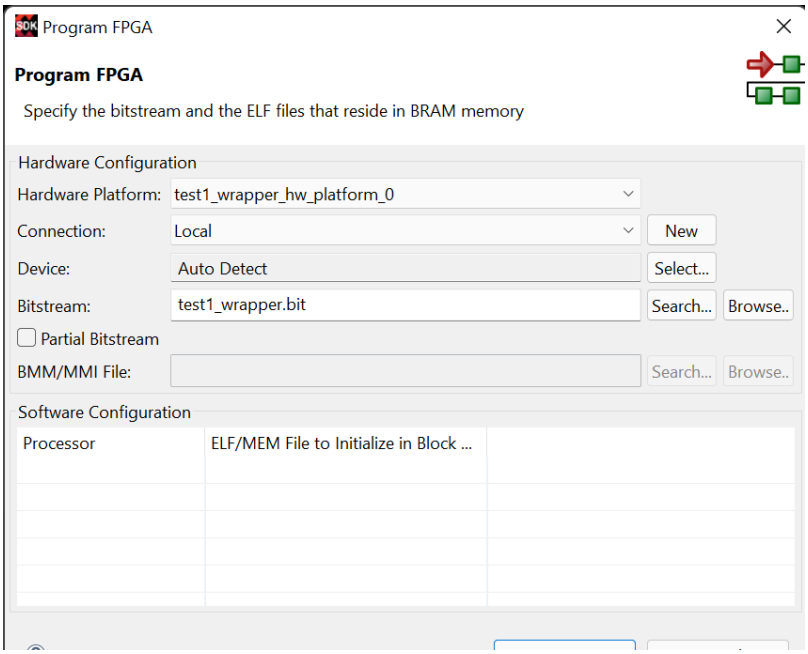

Figura 19 programmazione della scheda FPGA

Dopodiché viene scritto il codice in linguaggio C o C++, si configurano le impostazioni di esecuzione e infine si esegue il programma. In Figura 20 troviamo un esempio di come possono essere configurate le impostazioni di esecuzione. [6]

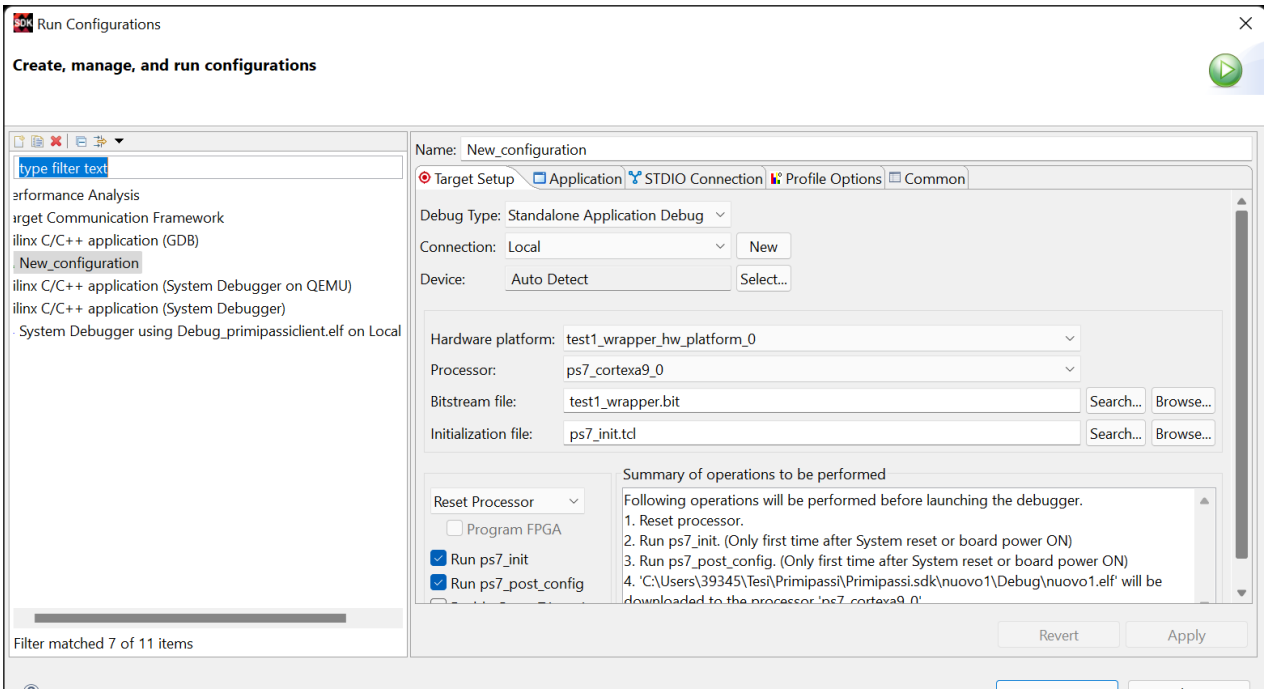

Figura 20 configurazioni di esecuzione

VIVADO DESIGN SUITE

È un prodotto software prodotto dall'azienda Xilinx per la sintesi e l'analisi dei progetti di linguaggio di descrizione hardware. Alcune delle funzionalità di questo strumento sono:

- Progettazione di livello RTL in VHDL, Verilog e SystemVerilog;
- Simulazione comportamentale, funzionale e temporale con il simulatore Vivado;
- Sintesi Vivado;
- Implementazione Vivado;
- Analisi di potenza Vivado;
- Analisi statica dei tempi;
- Generazione di bitstream. [7]

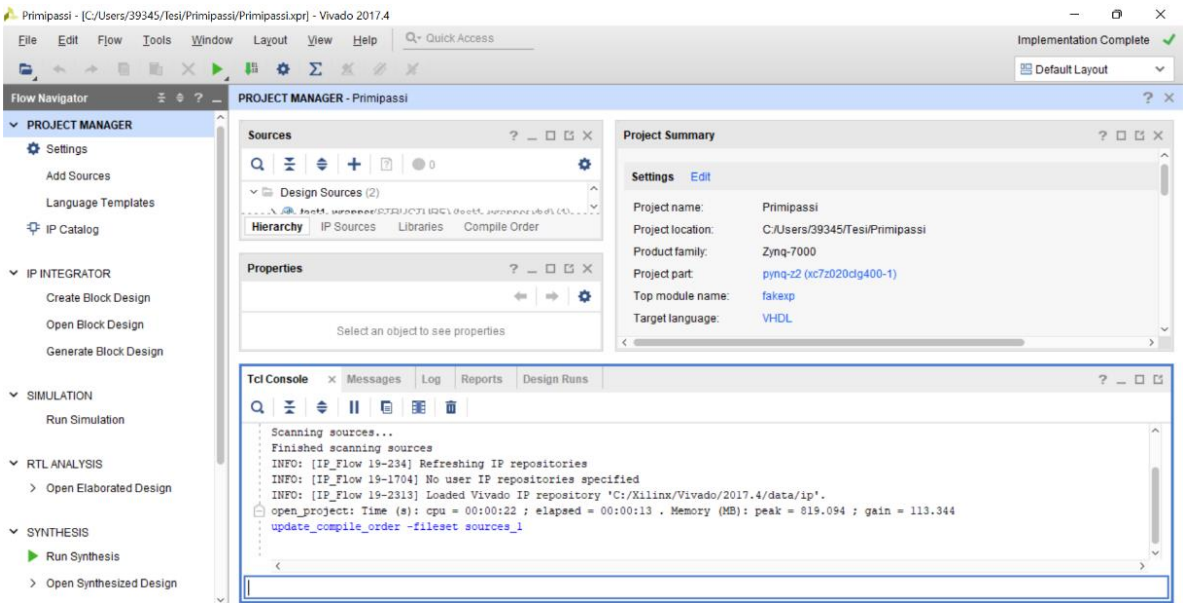

Figura 21 Interfaccia Vivado Design Suite

In Figura 21 è rappresentata l'interfaccia di Vivado Design Suite: nella parte in basso le sezioni Tcl Console, Messages e log danno informazioni sul processo, ovvero sul programma in esecuzione, segnalando eventuali errori o warnings. Nella sezione Sources verranno inseriti i moduli del programma, essi sono divisi in tre macro aree: Design Sources, Constraints e Simulation Sources. Infine, nel menù a sinistra si trovano tutte le funzionalità che vivado offre per l'analisi dei circuiti.

Capitolo 4. Risultati

Come sarà mostrato di seguito il sistema embedded creato soddisfa tutti i requisiti prefissati. La funzione esponenziale approssimata si è difatti rivelata più veloce e meno dispendiosa della funzione esponenziale precisa nonostante qualche margine di errore.

In tabella 1 e 2 sono riportati tutti i risultati ottenuti dal codice descritto nel Paragrafo 2.1; in particolare nella Tabella 1 sono stati inseriti i risultati riguardanti la velocità delle due funzioni, mentre nella Tabella 2 ci sono i risultati che riguardano l'accuratezza di calcolo.

Dalla tabella 1 possiamo notare che la funzione approssimata, ricavata come funzione a tratti, è mediamente 3 volte circa più veloce della funzione esponenziale. Il valore Δc per cui si registra il maggior numero di clocks scanditi è 0, ed in questo caso la funzione esponenziale approssimata è addirittura 5 volte circa più veloce della funzione esponenziale precisa.

	Funzione esponenziale precisa	Funzione esponenziale approssimata
Valore massimo assunto dalla funzione	64	64
Valore minimo assunto dalla funzione	0	0
Numero di clock medio necessario per calcolare la funzione	221,51	75,82
Numero di clock massimo necessario per calcolare la funzione	908	198
Δc per cui si registra il numero massimo di clock	0	0

Tabella 1 Tabella dei risultati

Per quanto riguarda l'accuratezza, già un primo risultato può essere ricavato dalla Tabella1 : le due funzioni hanno ugual valore massimo e valore minimo; in questi casi si registra la minore discrepanza tra le due funzioni, che è nulla. Nella Tabella 2 si nota che in media le due funzioni differiscono per un valore pari a 3,36 e la massima differenza tra le due è 3,87, la disuguaglianza è abbastanza irrisoria. Si può concludere, quindi, che la funzione approssimata è decisamente più veloce della funzione esponenziale precisa e si ha un errore tale da poterlo considerare trascurabile.

Errore medio	3,36
Errore massimo	3,87
Δc per cui si registra l'errore massimo	7
Errore minimo	0
Δc per cui si registra l'errore minimo	0

Tabella 2 Tabella degli errori

Grazie all'ausilio di Microsoft Excel sono state riprodotte le due funzioni in modo da rappresentarne l'andamento e poterle confrontare (Figura 22).

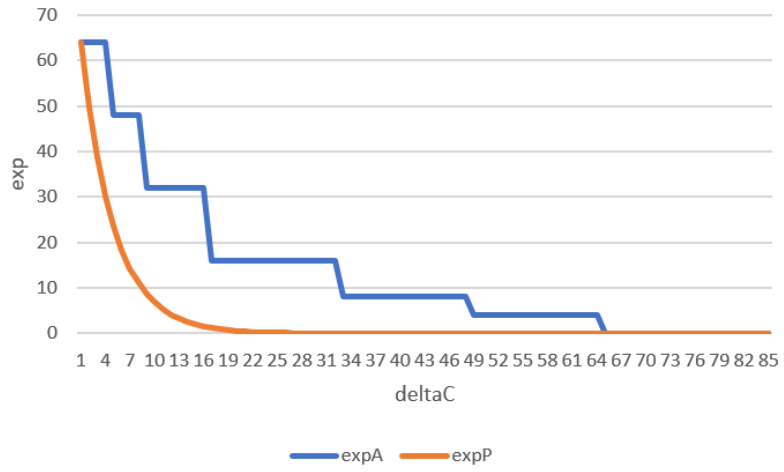

Figura 22 esponenziali a confronto

Per poter visualizzare i risultati hardware bisogna simulare il componente in cui si trova la funzione esponenziale.

Come descritto in Figura 23, il componente viene simulato per 256 valori di Δc, da 0 a 255. Il clock inizialmente è impostato a 0, o in altre parole il clock è in "tempo di OFF". Esso viene convertito da ON ad OFF e viceversa ogni 5ns; un intero ciclo di clock dura 10ns.

```
architecture Behavioral of simfakexp is
    signal IA : std_logic_vector(7 downto 0);
    signal Iclk : std_logic:='0';
    constant Tclk: time:=10 ns;
    signal O : std_logic_vector(7 downto 0);


    component fakexp
        Port (
              deltac : in STD_LOGIC_VECTOR (7 downto 0);
              clk : in STD_LOGIC;
              ris : out STD_LOGIC_VECTOR (7 downto 0));
    end component;

begin
    circuito: fakexp port map (IA,Iclk,O);

    process
        begin
            for i in 0 to 255 loop
                IA<=conv_std_logic_vector(i,8);
                wait for Tclk;
            end loop;
    end process;

    process
        begin
            wait for Tclk/2;
            Iclk<=not Iclk;
    end process;

end Behavioral;
```

Figura 23 codice della simulazione del circuito

Come descritto nel capitolo precedente in Vivado si possono effettuare tre tipi di simulazione:

- Behavioural Simulation;
- Post-Synthesis Timing Simulation;
- Post-Implementation Timing Simulation.

Per ogni simulazione vengono rappresentati tanti segnali quanti se ne usa nel codice. Nel caso trattato ci saranno i segnali: Ia che rappresenta i valori assunti da Δc, Tclk che è il periodo di clock, Iclk che delinea il segnale di clock ed infine O che mostra i risultati generati dalla funzione.

La simulazione comportamentale (Figura 24) è una simulazione ideale, in cui i ritardi dovuti alla preparazione di tutti i componenti del circuito sono considerati nulli.

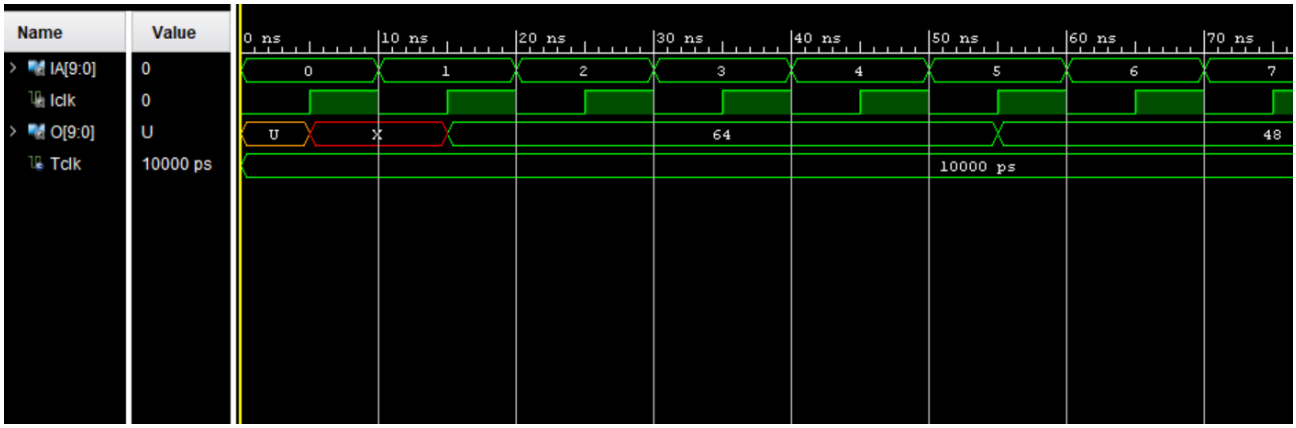

Figura 24 Behavioural Simulation

Durante il primo tempo di OFF il sistema si trova in uno stato indeterminato, seguito da uno stato di errore. Questo accade poiché il circuito a 5 ns, ovvero al primo rising edge del clock, legge il segnale Ia che era stato inserito in un registro e, una volta calcolata la funzione esponenziale, inserisce il risultato in un registro e deve aspettare il rising edge successivo, che sarà a 15ns, per poterlo inviare in output. Come si può notare infatti a 15ns il segnale O avrà valore 64.

Lo stato di errore o inconsistenza nelle simulazioni Post-Synthesis e Post-Implementation (Figura 25 e Figura 26) si protrae più a lungo perché oltre al periodo di errore dovuto alla sincronizzazione con il clock bisogna tenere conto dei ritardi causati dalla preparazione di tutti i componenti appartenenti al circuito.

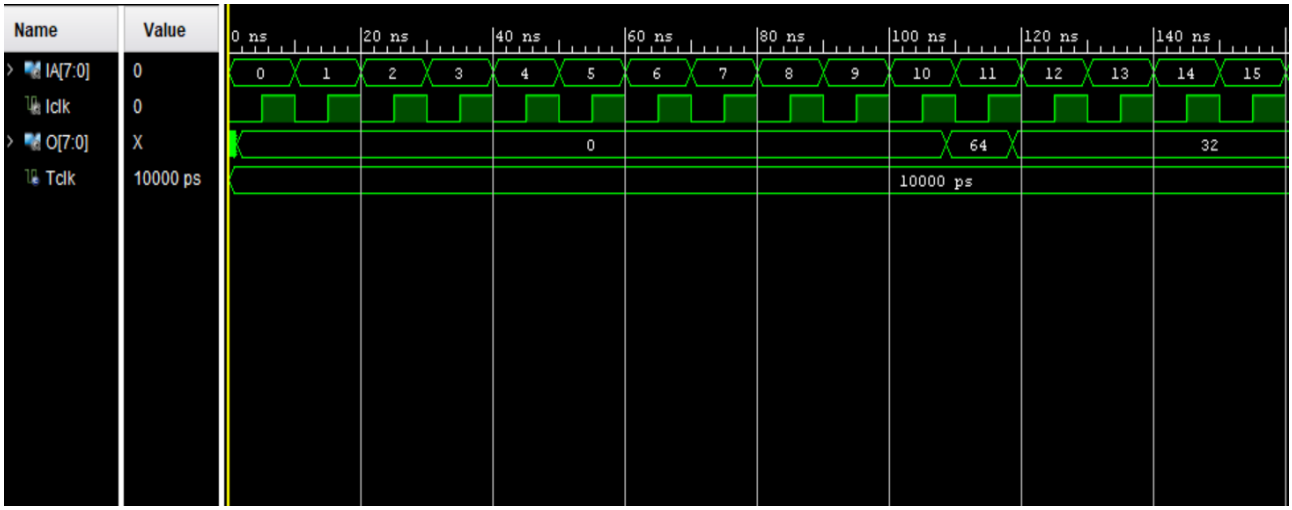

Figura 25 Post-Synthesis Timing Simulation

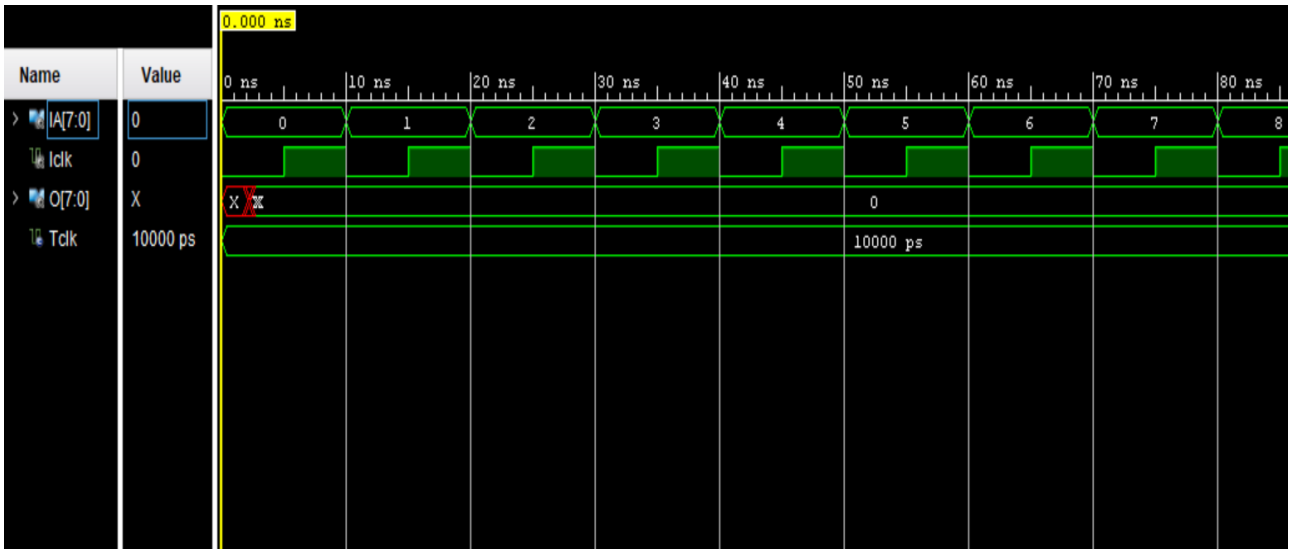

Figura 26 Post-Implementation Timing Simulation

La parte di circuito utilizzata, come si può notare dallo schema del device in Figura 27a, è la sezione azzurra nella regione di clock X0Y0. In Figura 27b la possiamo vedere in dettaglio.

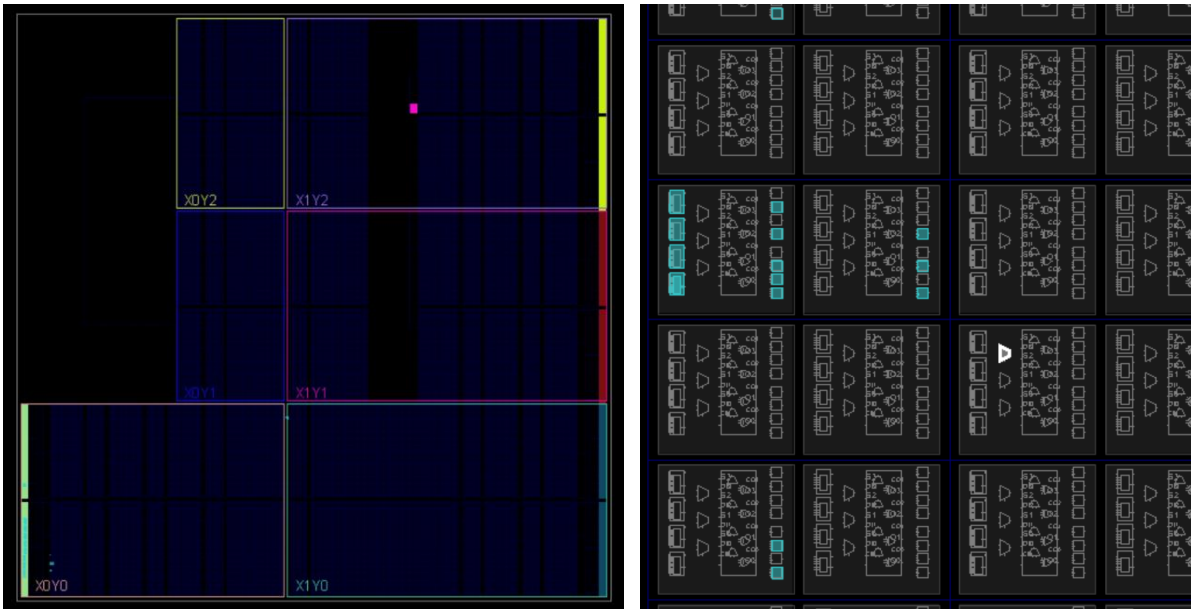

Figura 27 (a) Device View, (b)Dettaglio Device View

Per descrivere al meglio l'immagine 27b ci si può avvalere dello Slice Logic in Figura 28 estrapolato dal report utilization generato da Vivado in seguito alla sintesi e all'implementazione del circuito. Per eseguire l'approssimazione della funzione esponenziale si sono utilizzate quattro LUTs, ovvero strutture che assegnano ad ogni combinazione d'ingresso un valore in uscita, ed undici registri di tipo FLIP-FLOPs.

Site Type	Used	Fixed	Available	Util%
Slice LUTs	4	0	53200	<0.01
LUT as Logic	4	0	53200	<0.01
LUT as Memory	0	0	17400	0.00
Slice Registers	11	0	106400	0.01
Register as Flip Flop	11	0	106400	0.01
Register as Latch	0	0	106400	0.00
F7 Muxes	0	0	26600	0.00
F8 Muxes	0	0	13300	0.00

Figura 28 Slice Logic

In Figura 29 è descritto che tipo di FLIP-FLOPs e LUTs sono stati utilizzati e viene calcolato quanto spazio in essi è ancora disponibile.

Site Type	Used	Fixed	Available	Util%
Slice	4	0	13300	0.03
SLICEL	1	0		
SLICEM	3	0		
LUT as Logic	4	0	53200	<0.01
using O5 output only	0			
using O6 output only	3			
using O5 and O6	1			
LUT as Memory	0	0	17400	0.00
LUT as Distributed RAM	0	0		
LUT as Shift Register	0	0		
LUT Flip Flop Pairs	4	0	53200	<0.01
fully used LUT-FF pairs	1			
LUT-FF pairs with one unused LUT output	3			
LUT-FF pairs with one unused Flip Flop	3			
Unique Control Sets	1			

* Note: Review the Control Sets Report for more information regarding control sets.

Figura 29 Slice Logic Distribution

Un altro dato che si può estrarre dal report utilization, come si vede in figura 30 è che: non si è fatto utilizzo di alcuna memoria.

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	0	0	140	0.00
RAMB36/FIFO*	0	0	140	0.00
RAMB18	0	0	280	0.00

Figura 30 Memory

Dalle Figure 31 e 32 possono essere estratte alcune informazioni riguardanti la velocità del circuito: il clock constraint utilizzato ha un periodo di 10ns e la massima frequenza a cui lavora il circuito è 100MHz. Con un periodo pari a quello utilizzato, tutte le specifiche sono soddisfatte e il WNS è pari

ad 8.164ns, questo vuol dire che il circuito può lavorare con un periodo di clock maggiore o uguale di

$$periodo\ attuale\ -\ WNS\ =\ 10ns\ -\ 8{,}164ns = 1{,}836ns$$

Clock	Waveform(ns)	Period(ns)	Frequency(MHz)
myClock	{0.000 5.000}	10.000	100.000

Figura 31 clock summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8,164 ns	Worst Hold Slack (WHS):	0,137 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	5	Total Number of Endpoints:	5	Total Number of Endpoints:	12

All user specified timing constraints are met.

Figura 32 Design Timing Summary

In Figura 27b la potenza dinamica è rappresentata dai dispositivi in azzurro, mentre la rimanente parte del circuito rappresenta la potenza statica. La potenza dinamica fa riferimento alla frequenza di funzionamento settata nel constraint. Quando la frequenza è pari a $100MHz$, la potenza dinamica dissipata è pari a:

$$Pdinamica = 0.001\text{W}$$

Dunque la potenza utilizzata dal circuito è uguale all'1%; si deduce che ciò che dispende più potenza è il clock che utilizza il 50% della potenza dinamica.

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.106 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,2°C**
Thermal Margin:	58,8°C (4,9 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

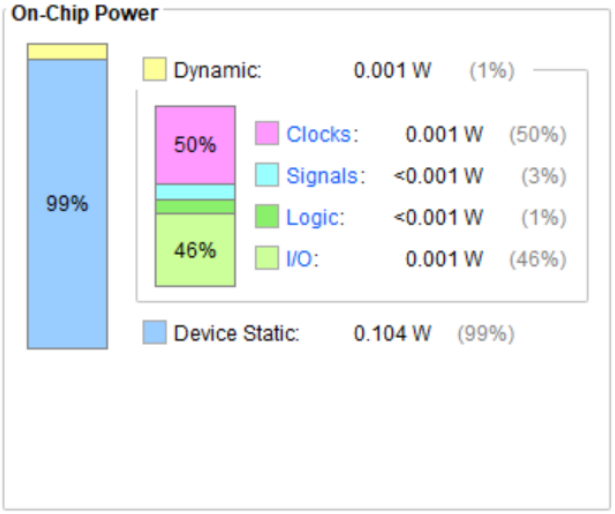

Figura 33 Analisi della dissipazione di potenza

Conclusione

Il problema della semplificazione del calcolo della funzione esponenziale, centrale nella computazione dei matching pixels per la ricostruzione computerizzata di immagini, è stato risolto ricorrendo ad una funzione piecewise. Tale funzione fa leva principalmente su due cose: la moltiplicazione e la divisione per un multiplo di 2 facilmente riproducibili in linguaggio di descrizione hardware tramite lo shifting di due bit verso destra o verso sinistra ed un if statement che controlla al più sei bit del vettore in input. La funzione approssimata non si è dimostrata efficiente solamente dal punto di vista dello sviluppo hardware, ma anche da quello software: è stato comprovato infatti, che la funzione esponenziale approssimata è molto più veloce della funzione esponenziale precisa seppur con un piccolo margine di errore.

In conclusione, la funzione approssimata soggetto di questo elaborato, con dei piccoli miglioramenti per quanto riguarda l'accuratezza, risulta essere un punto di partenza per ulteriori sviluppi futuri al fine di rendere il calcolo dei matching pixels quanto più efficiente possibile.

Riferimenti

[1] Euclide, «Prospettiva,» in *Ottica*, 300 a.C, pp. 80-84.

[2] Y. a. J. K. A. Liu, «Local and global stereo methods,» in *Handbook of Image and Video Processing*, 2005, pp. 297-308.

[3] P. C. F. F. S. P. Giuseppe Cocorullo, «An efficient hardware-oriented stereo matching algorithm,» *Microprocessors and Microsystems*, vol. 46, pp. 21-33, 2016.

[4] F. F. F. S. P. C. Stefania Perri, «Stereo vision architecture for heterogeneous systems-on-chip,» *Journal of Real-Time Image Processing*, vol. 17, p. 393–415, 2018.

[5] J. L. W. Z. H. Y. Y. W. a. X. G. Jingting Ding, «Real-time stereo vision system using adaptive,» *EURASIP Journal on Image and Video Processing*, n. 20, 2011.

[6] Xilinx Inc., «Getting Started with Xilinx SDK,» 2018. [Online]. Available: https://www.xilinx.com/html_docs/xilinx2018_1/SDK_Doc/sdk_getting_started/sdk_getting_started.html#sdk_getting_started.

[7] Xilinx Inc., «Vivado Design Suite User Guide,» 30 Ottobre 2019. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug906-vivado-design-analysis.pdf.