POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Progetto di Reti Logiche

Anno Accademico 2020-2021

Equalizzazione dell'istogramma di una immagine in scala di grigi

---

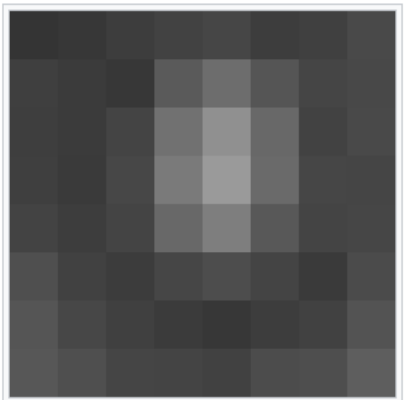
Autori:

Simone Giampà matricola 909739

Francesco Leone matricola 910196

Introduzione e Specifica del Progetto

Il metodo dell'equalizzazione dell'istogramma è utilizzato nell'ambito dell'elaborazione delle immagini per l'ottenimento delle stesse con un contrasto migliorato. Il metodo incrementa il contrasto globale di molte immagini e risulta utile specialmente quando i dati delle immagini sono rappresentati da valori numerici molto ravvicinati, quindi immagini a basso contrasto globale. Attraverso questa modifica l'intensità può risultare meglio distribuita sull'istogramma complessivo. In questo modo le aree a basso contrasto locale guadagnano un contrasto più elevato. Il metodo dell'equalizzazione dell'istogramma riesce ad ottenere questi risultati in modo abbastanza efficace estendendo i valori di intensità più frequenti.



52	55	61	59	79	61	76	61
62	59	55	104	94	85	59	71
63	65	66	113	144	104	63	72
64	70	70	126	154	109	71	69
67	73	68	106	122	88	68	68
68	79	60	70	77	66	58	75
69	85	64	58	55	61	65	83
70	87	69	68	65	73	78	90

Immagine in scala di grigi a 8 bit, e la sua corrispondente rappresentazione numerica decimale

Ogni immagine è rappresentata in una scala di grigi a 8 bit, quindi ogni pixel assume un valore numerico compreso tra 0 e 255, dove allo 0 corrisponde un pixel nero e al 255 un pixel bianco. Ogni pixel viene rielaborato e viene prodotto un nuovo valore numerico a partire dal precedente, sempre nella stessa scala di grigi.

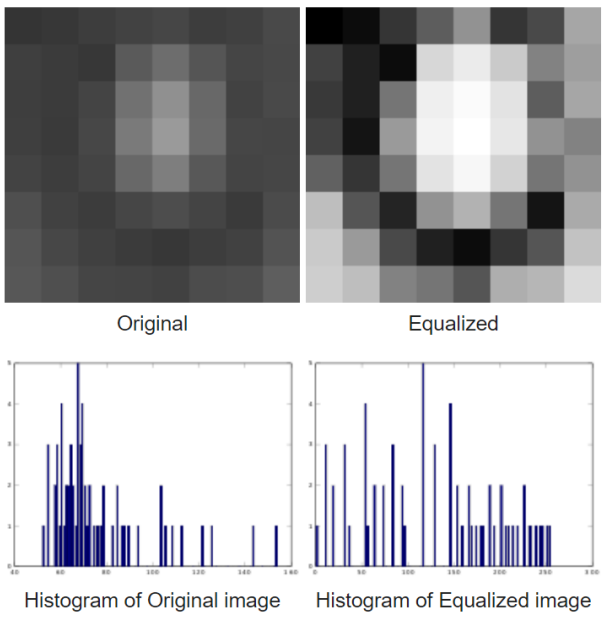


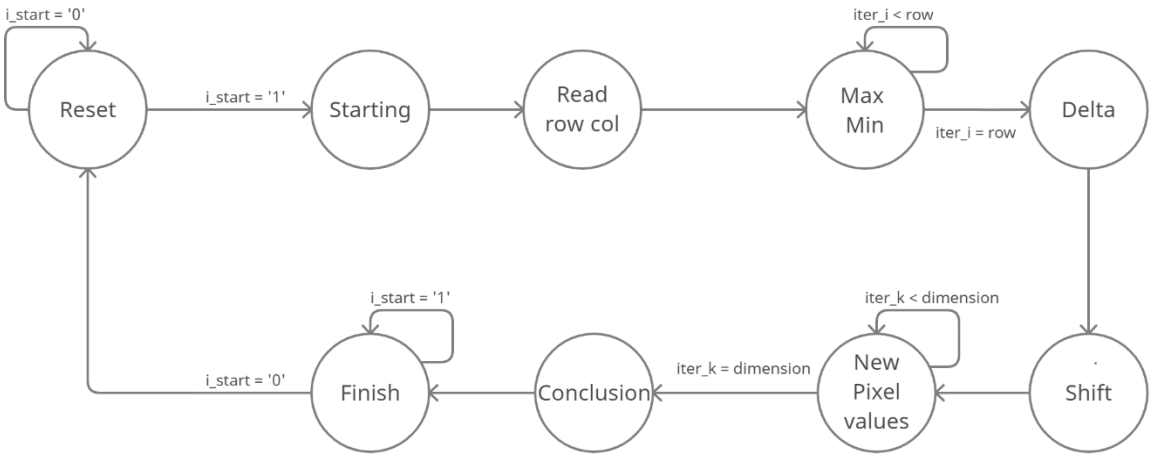
Immagine originale e relativo istogramma di rappresentazione a sinistra; accanto l'immagine equalizzata e il nuovo istogramma

Fonte: [Histogram Equalization - Wikipedia]

Descrizione Progettuale

L'approccio usato nel progetto è stato quello della macchina a stati. Nella figura che segue c'è una rappresentazione ad alto livello della macchina in cui alcuni stati sono implementati con più di uno stato. Il progetto è stato realizzato creando una sola *architecture* e una sola *entity*. La macchina a stati è sincronizzata sul fronte di salita del clock. Lo stato di reset può essere attivato alla fine dell'elaborazione di un'immagine oppure in un qualunque momento dell'esecuzione; in tal caso l'esecuzione si interrompe e la macchina torna nello stato di reset.

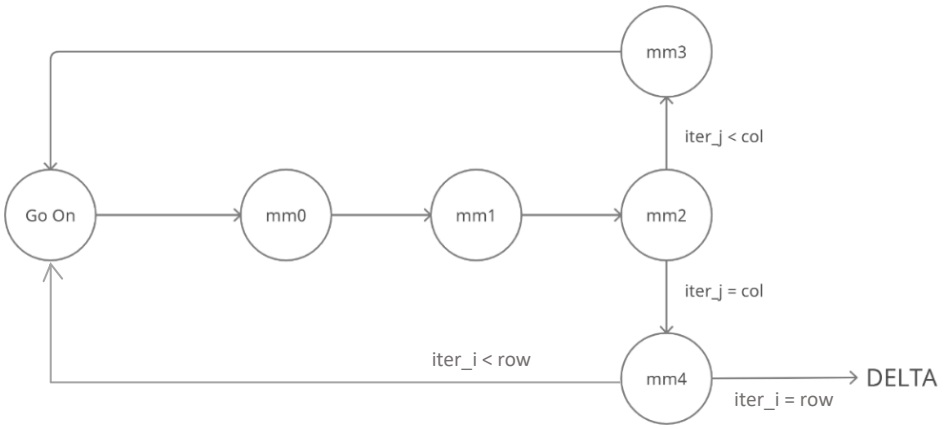
Qui di seguito viene riportata la macchina a stati finiti del progetto e una sua rappresentazione grafica ad alto livello. Per due stati in particolare (*Max Min* e *New Pixel Values*), viene fornita anche una descrizione a più basso livello e la struttura interna usata per la rappresentazione all'interno del codice VHDL.



Descrizione ad alto livello della FSM

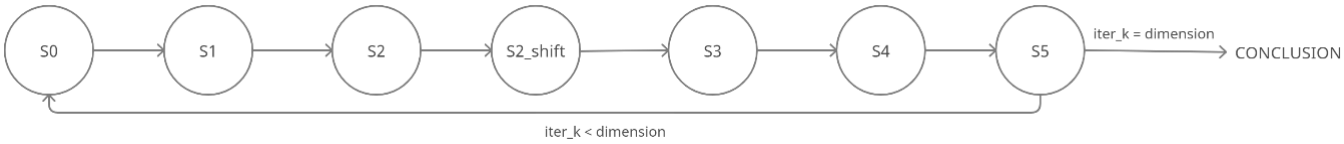
- **Starting***:* viene caricato il primo indirizzo della memoria.
- **Read Row Col:** vengono letti i primi due indirizzi di memoria salvando i valori di numero di righe e numero di colonne.
- **Max Min:** viene scansionata tutta la matrice alla ricerca dei valori massimo e minimo che vengono salvati in due registri.
- **Delta:** viene calcolata la differenza tra massimo e minimo e la dimensione in pixel della matrice.
- **Shift:** viene calcolato lo shift value dell'immagine con dei controlli a soglia.
- **New Pixel Values:** viene calcolato il nuovo valore di ogni pixel e viene scritto in memoria. Le iterazioni terminano quando *iter_i* diventa uguale alla dimensione dell'immagine.
- **Conclusion:** viene impostato ad alto il segnale *o_done*.
- **Finish:** si attende che il segnale *i_start* torni a 0 per processare un'altra immagine.

Ricerca di massimo e minimo: stato MAX MIN



- **Go On:** se i valori di riga e di colonna della matrice in ingresso non sono validi, ovvero se è presente una matrice "vuota" (dove il numero di righe o colonne vale 0), allora lascia intatta la memoria e conclude la computazione, spostandosi nello stato CONCLUSION. Altrimenti procede nello stato MM0.
- **MM0:** viene letto un nuovo pixel dalla memoria.
- **MM1:** viene confrontato il pixel letto con massimo e minimo e se necessario si aggiornano i valori di massimo e minimo.
- **MM2**: va in MM3 per leggere i pixel della stessa riga e in MM4 quando deve iniziare a leggere una riga nuova.
- **MM3:** viene aggiornato il segnale di uscita *o_address* con il nuovo indirizzo di lettura.
- MM4: se è stata letta tutta l'immagine si procede nello stato DELTA altrimenti si inizia a leggere una nuova riga.

Scrittura in memoria dell'immagine equalizzata: stato NEW PIXEL VALUES



- **S0:** il segnale *o_we* passa a basso (abilita la lettura in memoria) e viene letto un nuovo pixel.
- **S1:** viene incrementato l'indirizzo di lettura.
- **S2:** viene calcolata la differenza tra il pixel letto e il valore minimo.
- **S2_shift:** al pixel viene applicato lo shift e viene salvato il nuovo valore.
- **S3:** il segnale *o_we* passa ad alto (abilita la scrittura in memoria) e impostato l'indirizzo in cui scrivere il nuovo valore del pixel. I valori calcolati sono limitati a 255.
- **S4:** viene incrementato l'indirizzo di scrittura in memoria per il pixel successivo.
- **S5:** viene scritto su *o_data* il nuovo valore del pixel. Se l'immagine equalizzata è stata completamente scritta si passa allo stato CONCLUSION altrimenti si torna in S0 e ricomincia il ciclo.

Segnali utilizzati nel codice VHDL

Qui di seguito vengono riportati i segnali interni alla *architecture* implementata. Per ogni segnale definito viene descritto il funzionamento, lo scopo, e il tipo di dato usato per descriverlo. Inoltre per ogni segnale è anche spiegata la scelta progettuale per il numero di bit utilizzati. La maggior parte dei segnali utilizzati è di tipo *unsigned* in quanto descrivono valori numerici senza segno. La scelta del tipo è stata fatta sulla base di una comodità di elaborazione numerica dei valori, di interoperabilità tra segnali di diverso tipo e di facilità di rappresentazione per il tool di sintesi.

Segnale	Descrizione e Funzionamento	Tipo di dato
row	Numero di righe	*unsigned a 8 bit*
col	Numero di colonne	
max	Valore massimo tra i pixel dell'immagine	
min	Valore minimo tra i pixel dell'immagine	
iter_i	Contatore usato per scorrere le righe	
iter_j	Contatore usato per scorrere le colonne	
iter_k	Contatore usato per scorrere tutti i pixel. Viene confrontato con *dimension* nel ciclo	*unsigned a 16 bit*
dimension	Dimensione in pixel dell'immagine. Viene calcolato dopo la prima lettura dell'immagine usando l'indirizzo dell'ultimo pixel in memoria. Dimensione in bit adatta per contenere indirizzi di immagine di dimensione massima (128x128)	
tmp_num_8bit	Registro in cui vengono salvati i valori numerici prima di essere shiftati o comparati con *max* e *min*	*unsigned a 8 bit*
tmp_num_16bit	Registro in cui viene salvato il nuovo valore numerico shiftato	*unsigned a 16 bit*
delta_val	Viene salvata la differenza tra *max* e *min.* Nel caso peggiore vale 255, quindi servono 8 bit per rappresentarlo	*unsigned a 8 bit*
shift_level	Viene salvato il numero di bit di shift per l'immagine. Dato che è un valore compreso tra 0 e 8, bastano 4 bit per rappresentarlo	*unsigned a 4 bit*
read_address	Indirizzo del pixel da leggere in memoria. Viene incrementato dopo ogni lettura e scrittura	*std_logic_vector 16 bit*
write_address	Indirizzo di memoria su cui scrivere il nuovo valore del pixel. Viene incrementato dopo ogni lettura e scrittura	

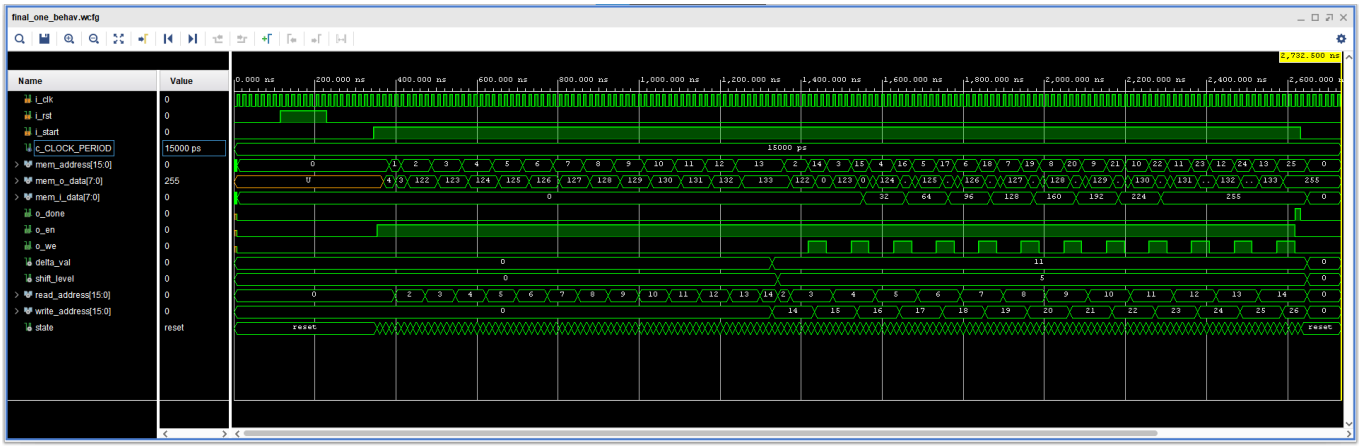
Risultati Sperimentali

Simulazioni del circuito sintetizzato

Sono stati effettuati vari test per stressare il circuito sintetizzato in vario modo. La FPGA utilizzata per la simulazione e la sintesi del circuito è xc7a200tfbg484-1, della famiglia Artix-7. Di seguito sono illustrati vari testbench e relativi risultati di simulazione, con particolare focus ai casi limite.

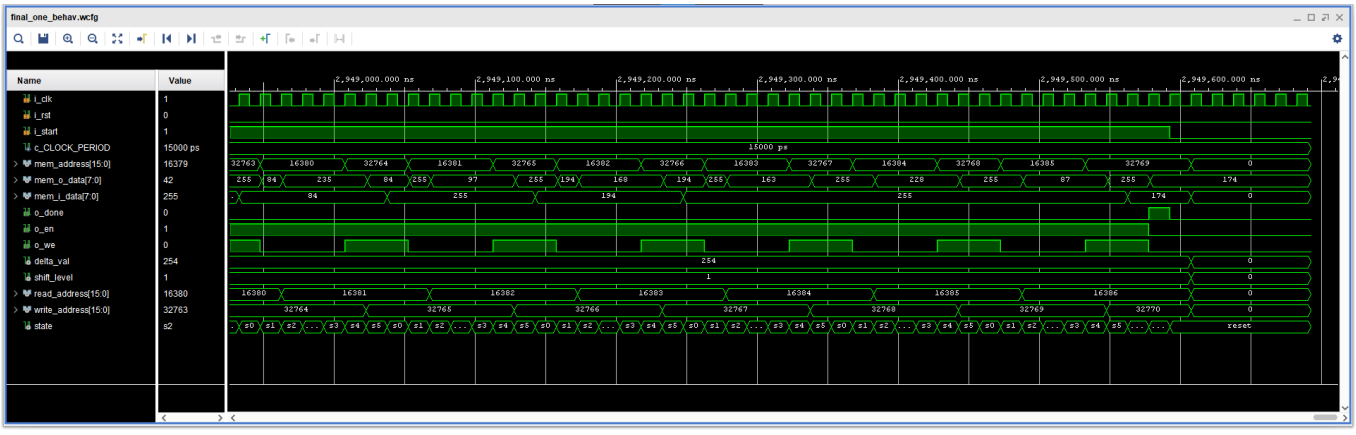
Testing Principale

Il test 1 è un test bench generico, che dà in ingresso alla memoria una immagine di 4x3 pixels. Con un clock da 15 ns, il tempo totale di esecuzione risulta di circa 2.7 µs.



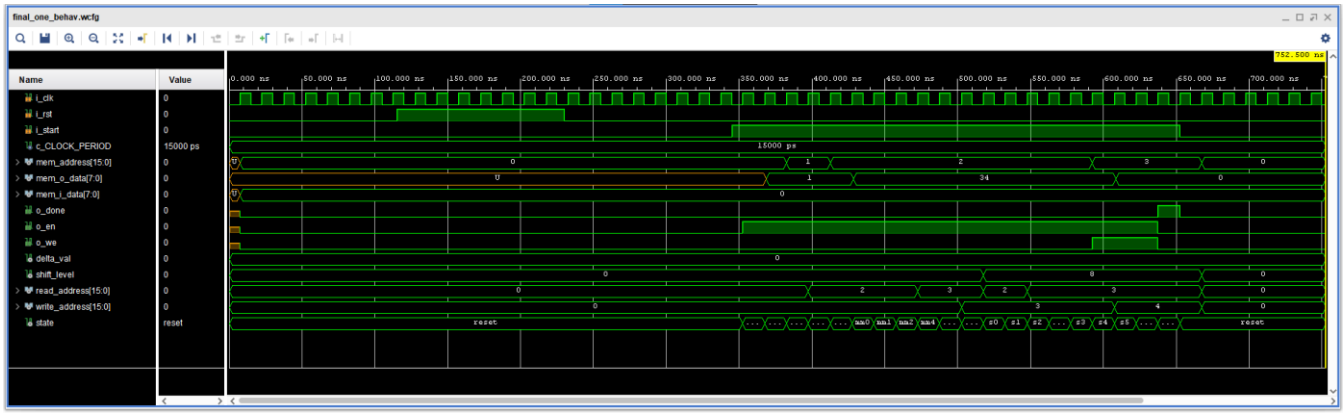
Segnali per la simulazione di un test bench generic, immagine 4x3

Il test 2 è un test volto a simulare l'elaborazione di una immagine di risoluzione massima, ovvero 128 x 128 pixels. Il tempo di esecuzione risulta significativamente maggiore rispetto ad altre immagini di risoluzione minore. Si riporta solo la parte finale della simulazione. Il tempo di esecuzione totale risulta di quasi 3 ms.



Segnali per la simulazione di un test bench con memoria inizializzata con una matrice di 128 x 128 pixels

Il test 3 è opposto al test 2, in quanto il suo obiettivo è di testare il comportamento del circuito nel caso la matrice in ingresso alla memoria sia costituita da un singolo pixel (matrice 1x1). Il risultato atteso è che la memoria debba rimanere inalterata. In questo caso, ovviamente, il tempo di esecuzione risulta molto più basso.



Simulazione di un test bench con memoria inizializzata con una matrice 1 x 1 contenente un singolo pixel

Altri testbench usati per testare le condizioni limite, che abbiamo simulato, danno in ingresso alla memoria matrici costituite da valori numerici molto ravvicinati tra di loro (così da rappresentare immagini a basso contrasto globale) e matrici con valori numerici estremi, ovvero 0 oppure 255 (così da rappresentare immagini a massimo contrasto globale possibile e già equalizzate).

In tutti i testbenches simulati, i risultati sono corretti.

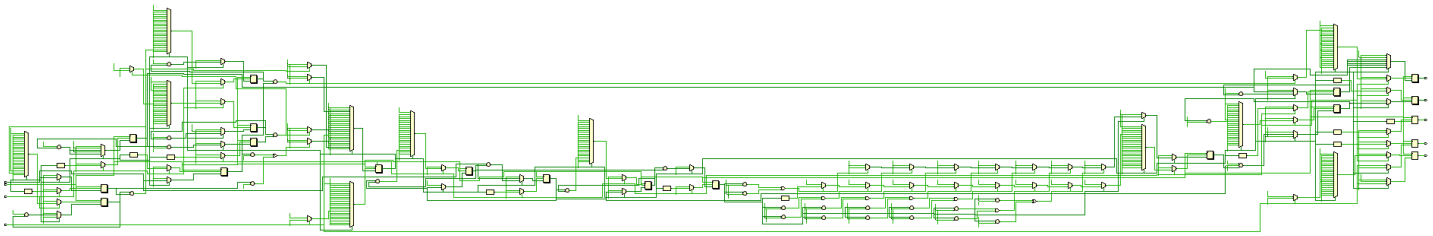
Stress Testing

Per mettere alla prova il circuito sintetizzato abbiamo sfruttato un generatore di file di test, messo a disposizione degli studenti del corso, per creare test benches molto più pesanti rispetto agli altri precedentemente descritti. Ogni testbench usato è stato simulato in *simulation – behavioral* e in *post synthesis – functional*. Ogni test effettuato ha dato esito positivo, senza alcun *warning* da parte del tool di sintesi. Di seguito illustriamo i casi più significativi.

- **Simulazione di 50 immagini consecutive**: abbiamo generato un file che istanzia la memoria con 50 diverse immagini. Ogni matrice ha una dimensione e dei valori generati in modo casuale. Ogni matrice è rappresentata da valori che variano da 0 a 255 e ogni valore è generato casualmente dal generatore. Il generatore stesso ha creato le istruzioni per il controllo dei valori scritti in memoria dal circuito sintetizzato. Il risultato finale è stato "test passato", a indicare la totale assenza di errori, anche in *post-synthesis*.
- **Simulazione di 3 immagini consecutive con segnale di reset intermedio**: un testbench che istanzia 3 diverse matrici in memoria, i cui valori numerici sono generati in modo casuale. Inoltre per complicare la simulazione, il segnale di reset viene portato alto durante il processamento della prima matrice. La risposta del circuito è stata l'azzeramento e il ripristino dei valori dei segnali ai valori di default iniziali. Dopodichè l'esecuzione normale è ripartita con le immagini successive in modo corretto.
- **Simulazione di reset multipli su una singola immagine**: questo testbench è stato usato per testare la correttezza del meccanismo di reset; portando il segnale di reset ad alto più volte durante la computazione di una singola immagine, il circuito si è comunque comportato in modo corretto.

Report di Sintesi del Circuito

I risultati della sintesi del circuito sono stati calcolati a partire dal processo di *report utilization*. Il circuito è correttamente sintetizzabile e implementabile da Vivado. A seguito del processo di sintesi risultano utilizzati in totale 195 LUT (LookUp Tables) e 197 FF (Flip-Flops). I numeri risultato molto più bassi rispetto alle reali capacità della FPGA utilizzata; infatti, il numero di FF è circa lo 0.07% del totale, mentre il numero di LUT è circa lo 0.14% del totale presente sulla scheda.



Schematica del circuito sintetizzato

Conclusioni del Progetto

La progettazione del circuito attraverso una singola macchina a stati finiti (FSM) ci ha permesso di imparare ad usare il software Vivado e il linguaggio di programmazione hardware VHDL. E' stato interessante progettare il circuito e verificare l'evoluzione dei segnali digitali nel tempo, come forma di debugging dei vari stati e il controllo della correttezza del circuito. L'algoritmo da sviluppare si è rivelato tanto impegnativo quanto stimolante per l'apprendimento del nuovo ambiente di sviluppo.