Implementazione VHDL dell’algoritmo Scanline

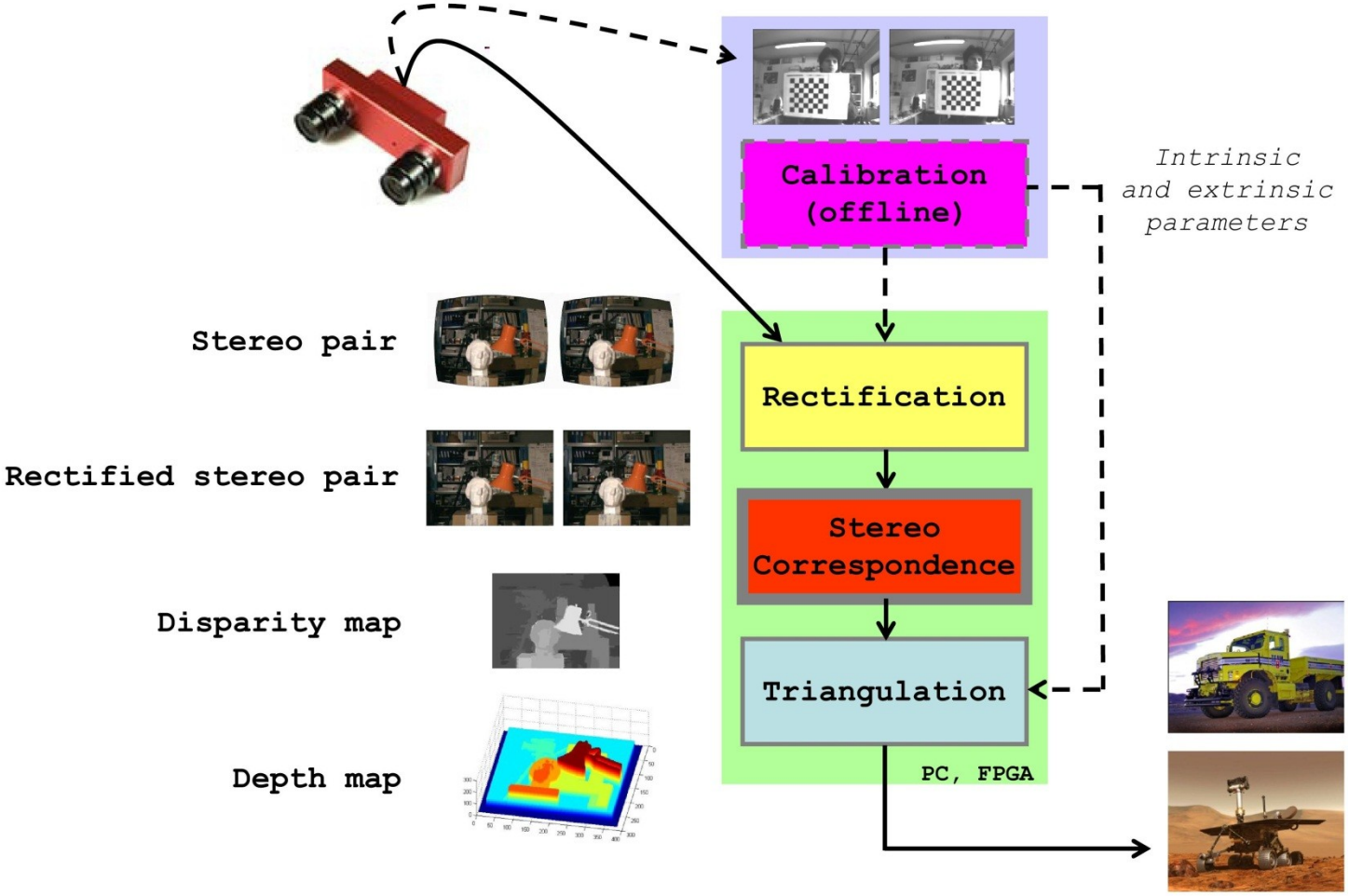
Stereo Vision

- L’analisi stereo è il processo di misurazione della distanza di un oggetto attraverso il confronto di due o più immagini provenienti da due o più telecamere che inquadrano una scena da differenti posizioni.
- La *triangolazione*, alla base della stereo vision, mette in relazione i punti (detti *omologhi*) ottenuti dalla proiezione, su due o più piani immagine, di un punto specifico della scena.
- L’individuazione dei punti omologhi (*problema delle corrispondenze*) determina la *disparità* da cui, noti la posizione reciproca delle telecamere ed altri parametri del sistema stereo (*calibrazione*), è possibile ricostruire la posizione tridimensionale del punto nella scena.

Sistema stereo vision - 1

- Ricostruzione struttura tridimensionale attraverso quattro fasi:
  - ***calibrazione:*** stima parametri intrinseci (es. orientazione telecamere) ed estrinseci (es. distorsione lenti).
  - ***rettificazione:*** elimina distorsioni dovute alle lenti delle telecamere *(forma standard).*
  - ***problema delle corrispondenze:*** calcola disparità tra ogni punto di un immagine reference e il corrispondente dell'immagine target.
  - ***triangolazione:*** ricostruisce la struttura tridimensionale della scena dai parametri di calibrazione e dalla mappa di disparità.

Sistema stereo vision - 2



Scanline Optimization

- metodo semi-globale in cui il problema delle corrispondenze si riduce ad un sottoinsieme di punti dell'immagine (es. righe di un'immagine, *scanline*).
- per ogni punto di questo sottoinsieme minimizza una funzione energia.
- buon compromesso tra accuratezza e velocità.
- tecnica pronta a soddisfare anche specifiche real-time.

Scanline Optimization – funzione energia

$$L(p,d) = C(p,d) + \min \begin{cases} L(p-1,d) \\ L(p-1,d-1) + P_1 \\ L(p-1,d+1) + P_1 \\ L(p-1,d') + P_2, \forall d' : |d - d'| > 1 \end{cases} - \min L(p-1,k)$$

$$\min(L(p-1,d'), \forall d' : |d - d'| > 1) = \min L(p-1,k)$$

$$k \in [d_{\min}, d_{\max}]$$

Scanline Optimization – algoritmo

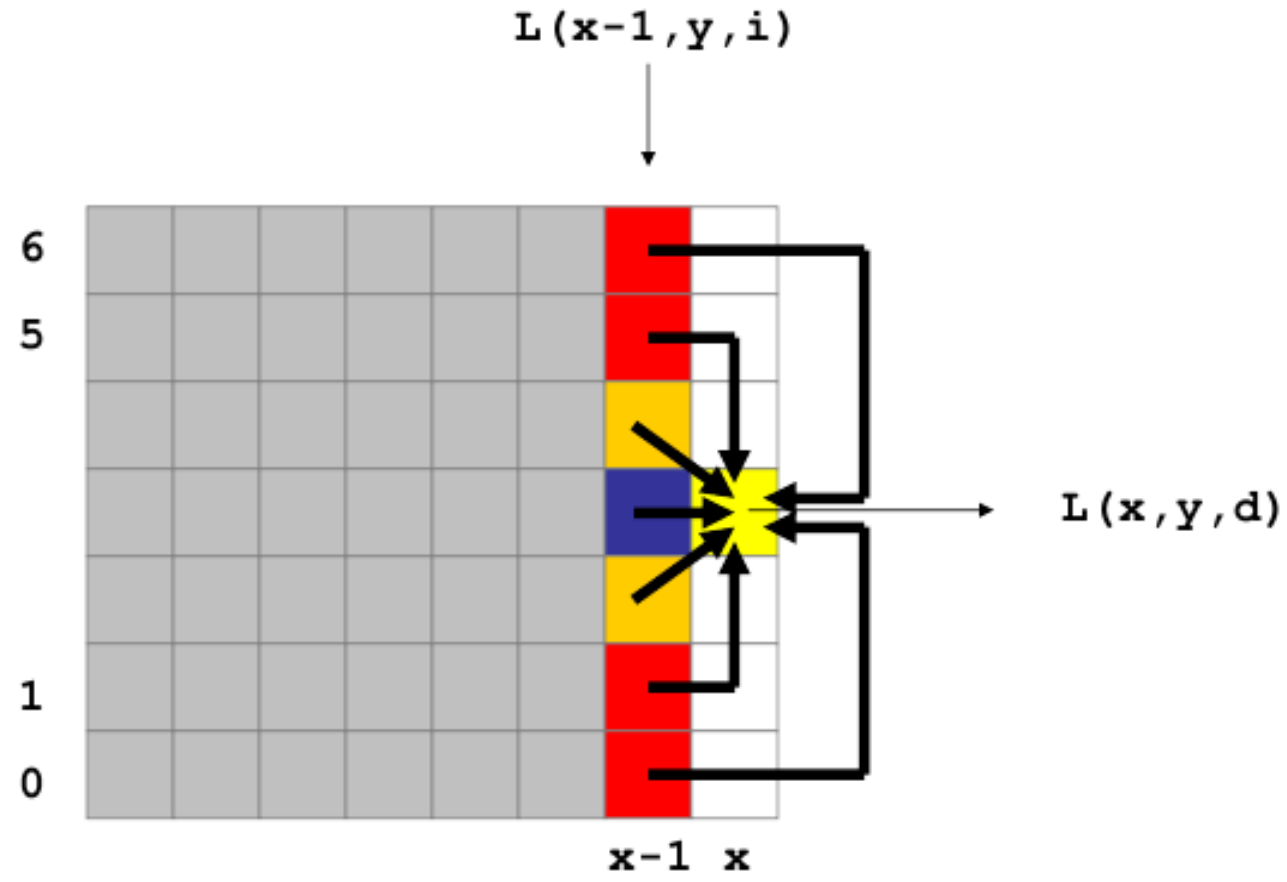
Per un punto *p* dell'immagine reference e per ogni $d' \in [d_{\min}, d_{\max}]$:

1. Calcolo del ***costo locale C*** come valore assoluto della differenza tra i valori d'intensità del pixel reference e del pixel target a distanza *d'*.
2. Calcolo del ***costo globale L*** come somma del *costo locale C* e del minimo *costo globale L* del punto precedente con penalità:
   - *P1* per $|d - d'| = 1$
   - *P2* per $|d - d'| > 1$, $(P2 > P1)$
   - 0 per $d = d'$
3. Si sottrae a *L* il minimo costo globale del punto precedente, senza penalità, evitando possibili casi di overflow.

Dal vettore dei costi globali *L* del punto *p,* si determina il valore di disparità da assegnare alla mappa di disparità alle medesime coordinate di *p*:

$$d_{best}(p) = \arg\min_{d}\{L(p,d)\}$$

Scanline Optimization – algoritmo



Progetto - Obiettivo

- Realizzazione modulo ***VHDL*** dell'algoritmo *Scanline Optimization* per FPGA *xilinx spartan 3*.

Fasi Di Realizzazione

- Dalla discrizione al implementazione del algoritmo in C.

  - Codice pensato per futura implementazione in FPGA.
  - Realizzazione del algoritmo per ogni direzzioni separatamente.
  - Analisi dei punti critici e risorse comuni.

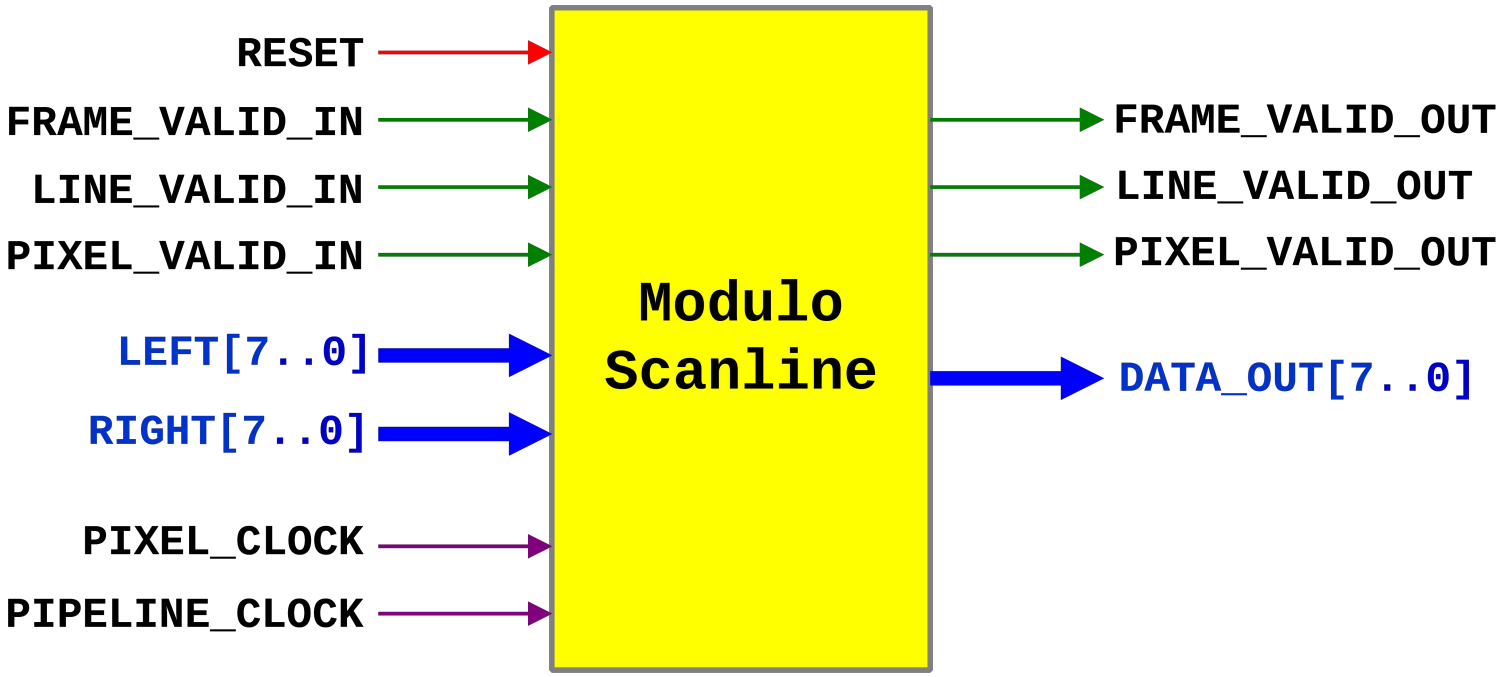
Fasi Di Realizzazione

- Prima versione del algoritmo in VHDL.

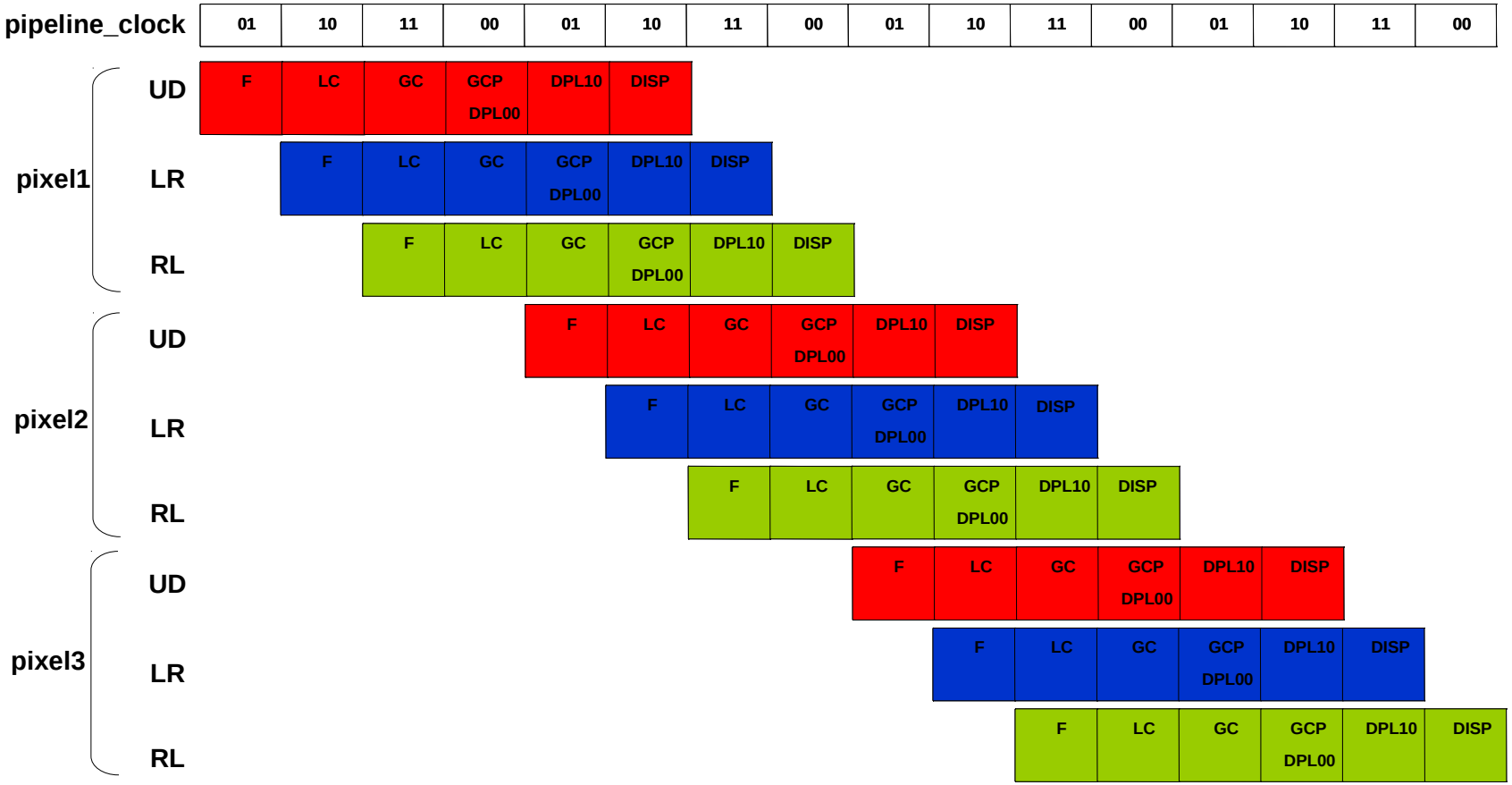
VHDL != C

- Individuazione degli stadi della pipeline(PEXEL_CLOCK).
  - codice VHDL per la direzione LR
  - Analizzi di punti critici.
  - Calcolo parallelizzato
  - Semplificazione.
- Stadi di pipeline (PEPELINE_CLOCK)
- Calcolo del minimo a pipeline parallelo.
- Codice per ogni direzione
- Unione del codice, individuazione del nuovi stadi

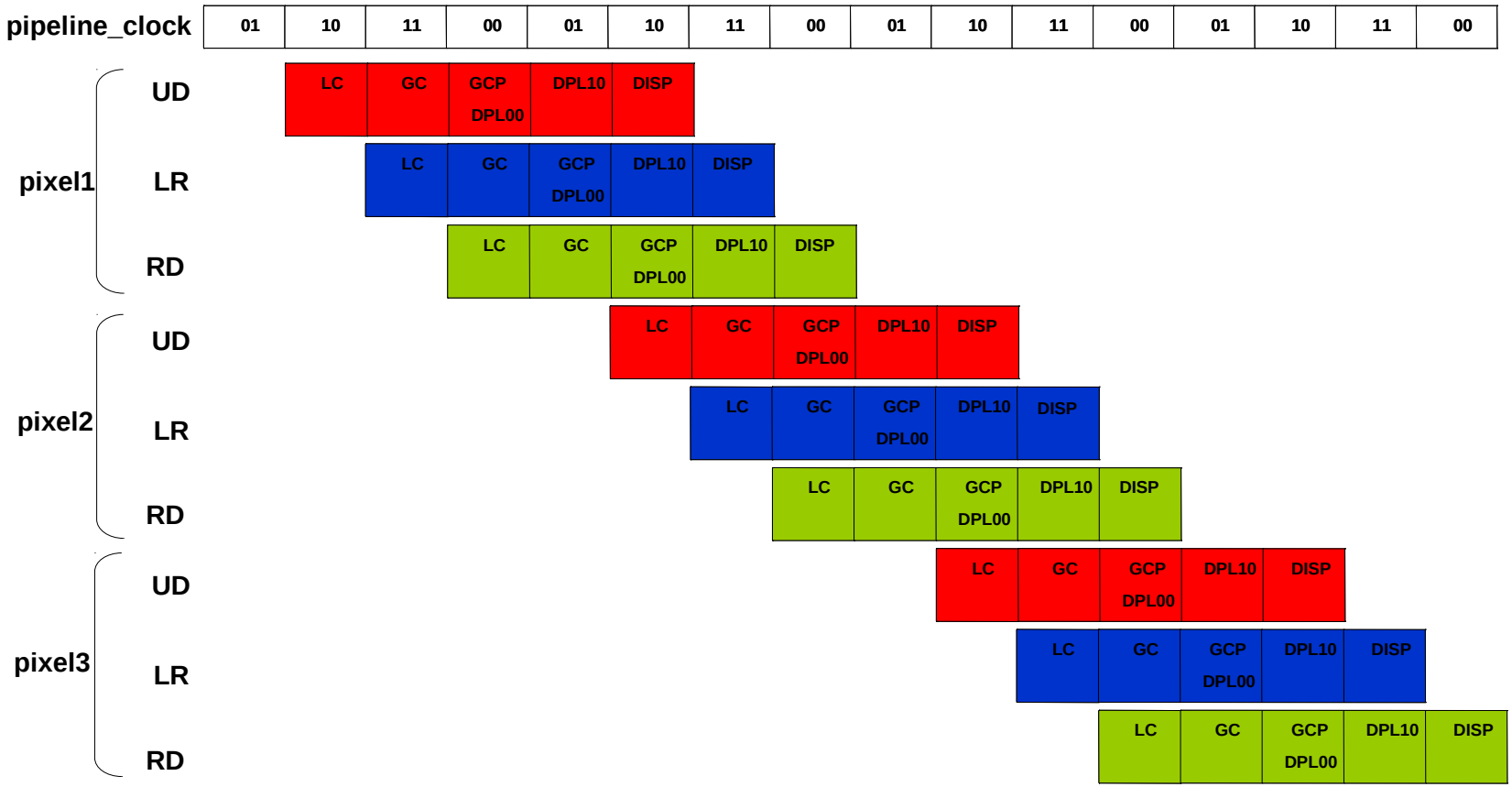
Progetto – Interfaccia modulo Scanline



Progetto – Implementazione modulo Scanline UD LR RL

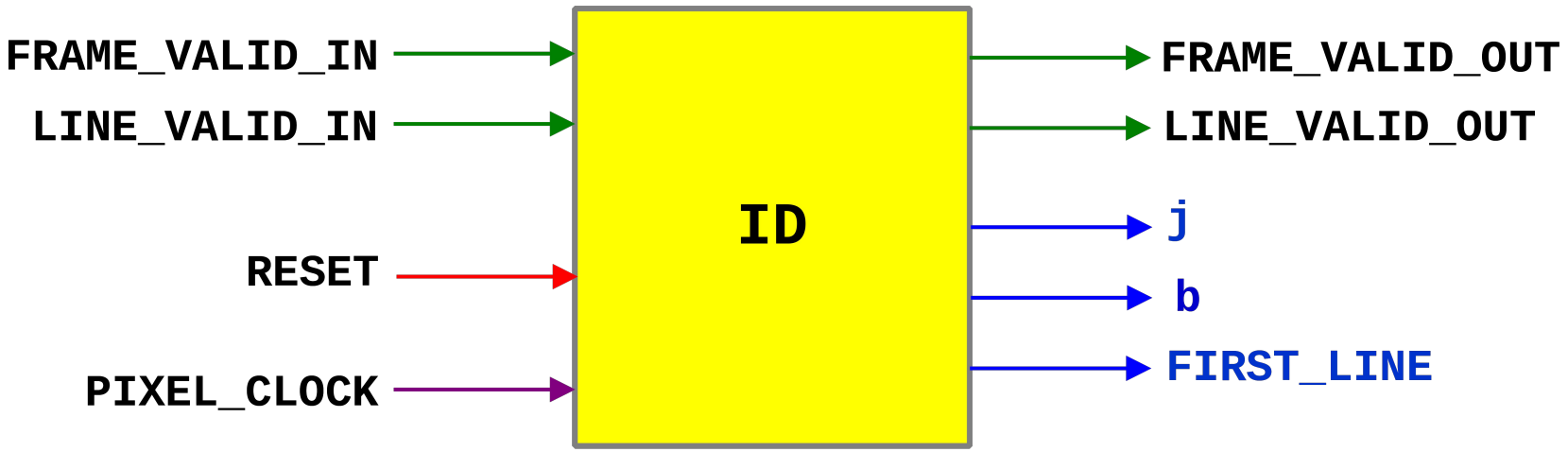


Progetto – Implementazione modulo Scanline UD RD LR



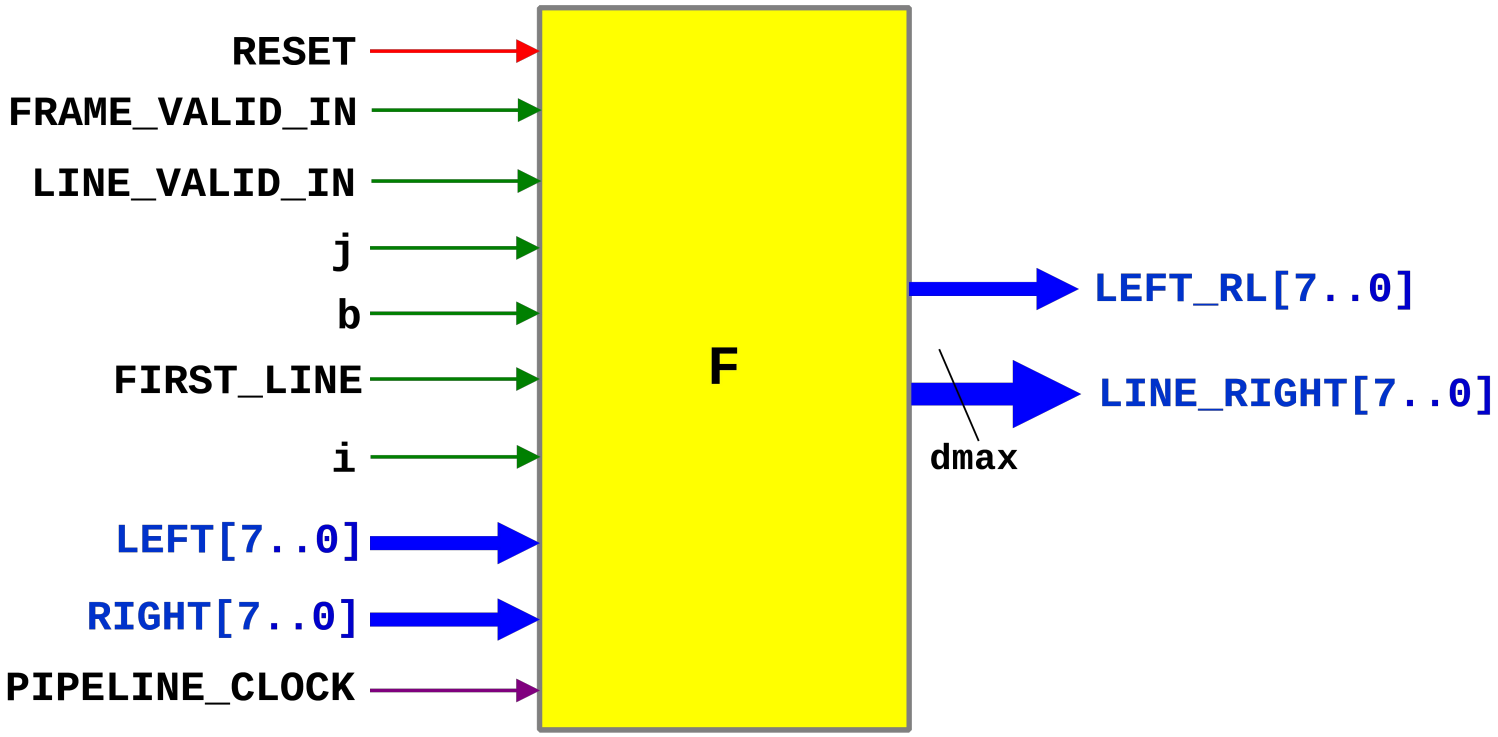
Progetto – process ID

- Aggiorna i segnali di controllo per il corretto impiego di alcuni componenti della pipeline.



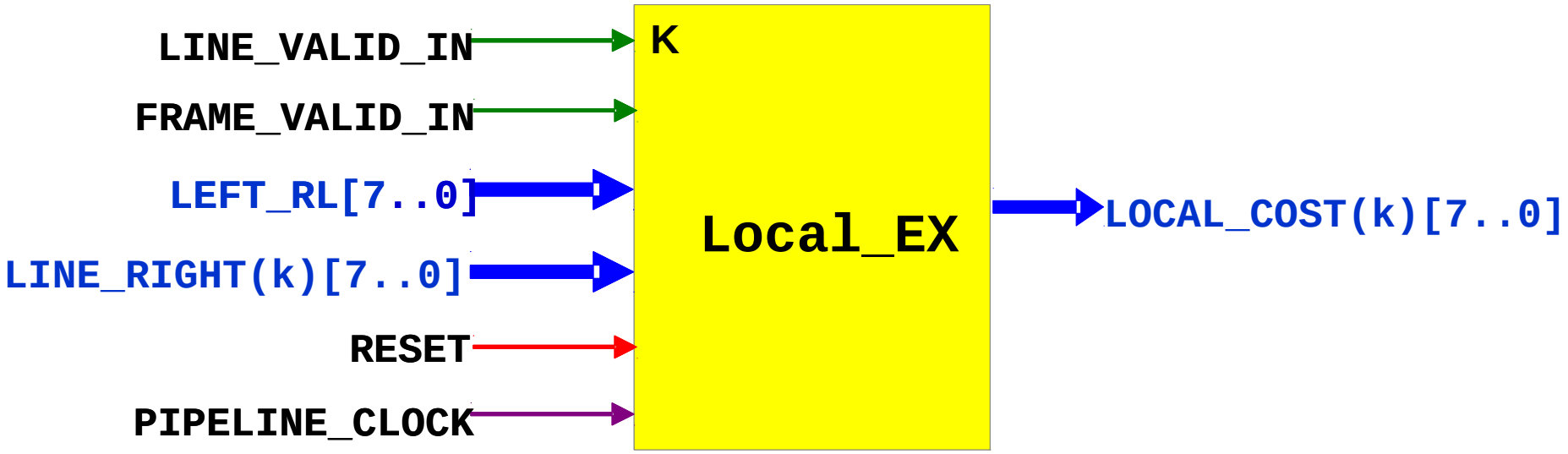
Progetto – process F

- Lettura pixel d'ingresso e loro memorizzazione in opportune strutture (registri e block ram). Serve solo in presenza di direzzione RL.
  - 2 blockram + shiftregister



Progetto – process Local_EX

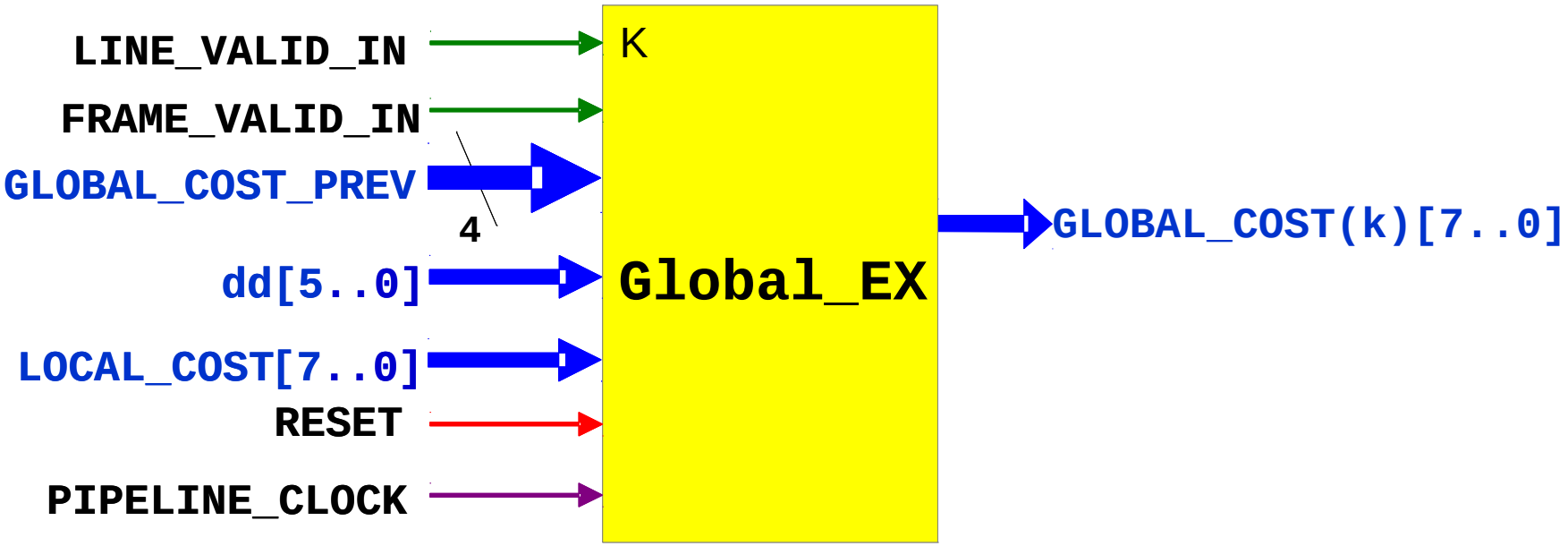
- dmax processi Local_EX. Ogni processo calcola il costo locale a partire dai valori forniti dallo stadio *F*.



Progetto – process Global_EX

- dmax processi Global_EX. Dal costo locale in uscita da *LOCAL_EX* e dal costo globale e disparità minima del pixel precedente, determina il costo globale del pixel corrente.
- Confronto di 4 numeri a 8 bit

  GC(k); GK(k-1) + P1; GK(k+1) + P1; GC(Kmin) + P2



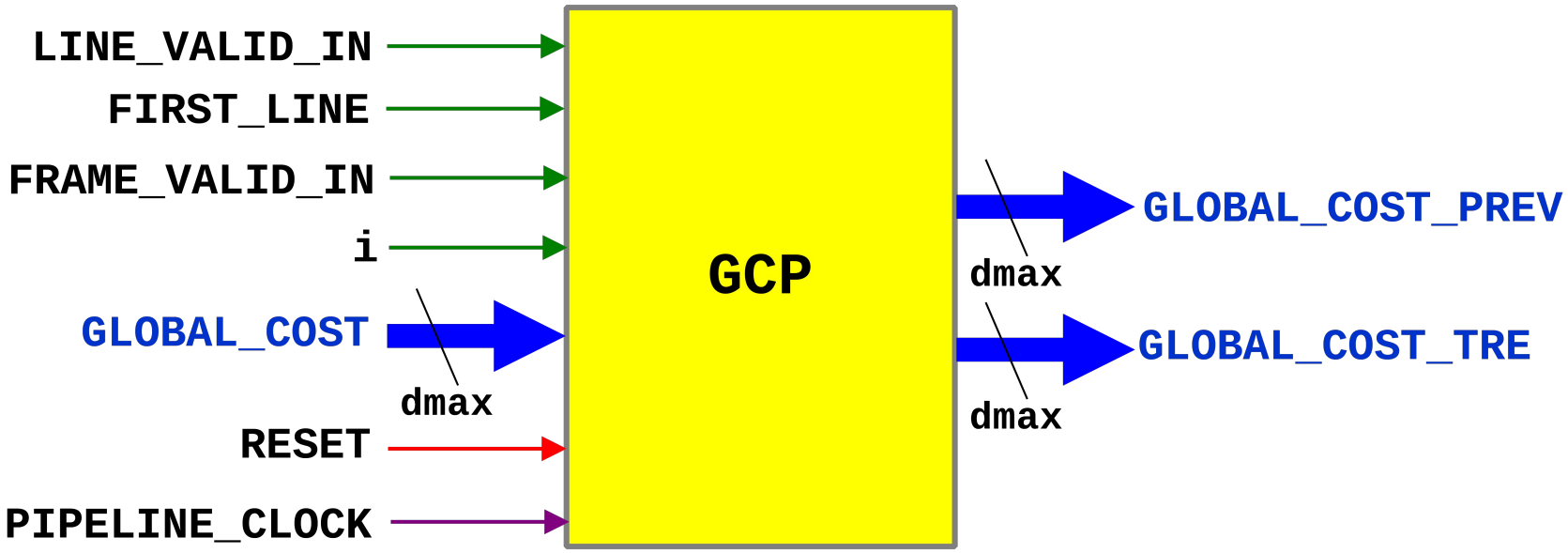
Confronto A;B;C;D

```
R1:=A
If(A > B) then
        R1 := B;
end if;
If(R1 > C) then
        R2 := C;
end if;
If(R2 > D) then
        R3:=D;
end if;
<SOMMA>;
```

```
If(A>B) then
        R1:=B;
end if;
If(R1 > C) then
        If(C > D) then
                <SOMMA>;
        else
                <SOMMA>;
        end if;
else
        If(R1 > D) then
                <SOMMA>;
        else
                <SOMMA>;
        end if;
end if;
```

Progetto – process GCP

- Somma tra loro i costi globali delle singole direzioni e invia il costo globale del pixel precedente al blocco *GLOBAL_EX* o direttamente al DPL00.



Somma di 3 vettori di 64 numeri a 8 bit;

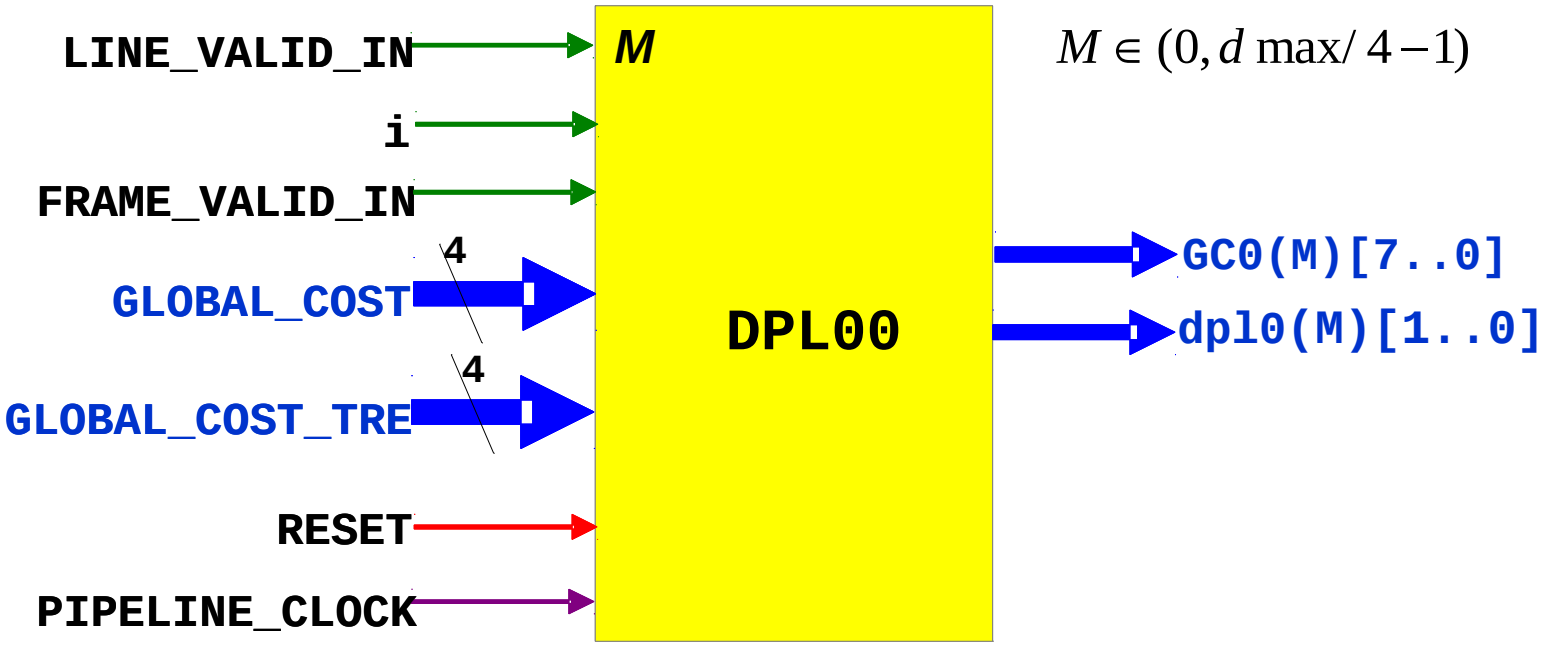
```
for k in 0 to dmax - 1 loop
    GC_TRE(k) := GC_TRE(k) + GlobalCostUD(k)(7 downto 2);
end loop;

for k in 0 to dmax - 1 loop
    GC_TRE(k) := GC_TRE(k) + GlobalCostLR(k)(7 downto 2);
end loop;

for k in 0 to dmax - 1 loop
    GC_TRE(k) := GC_TRE(k) + GlobalCostRL(k)(7 downto 2);
end loop;
```

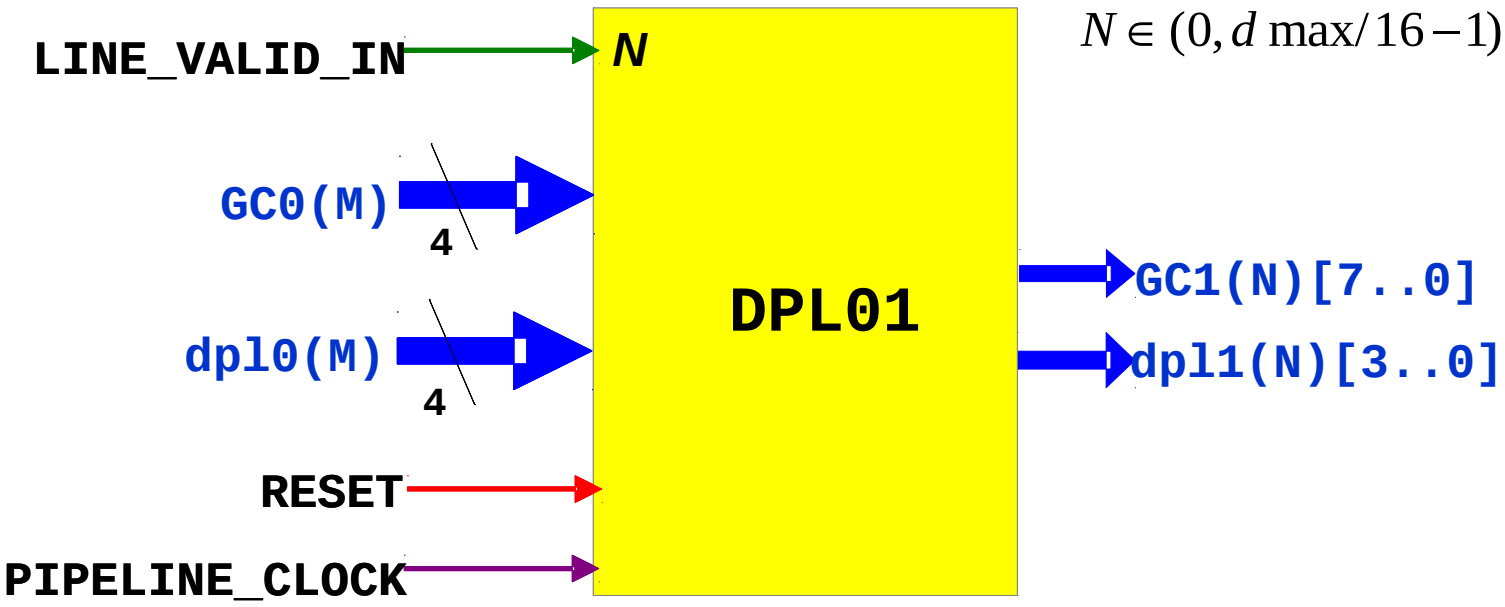
Progetto – process DPL00

- dmax / 4 processi. Restituisce il primo livello di costi globali minimi. Vettore di GC viene diviso in gruppi di 4, per ogni gruppo viene calcolato il minimo. GC0 – minimi, dp0 – indice di minimo in ogni gruppo.



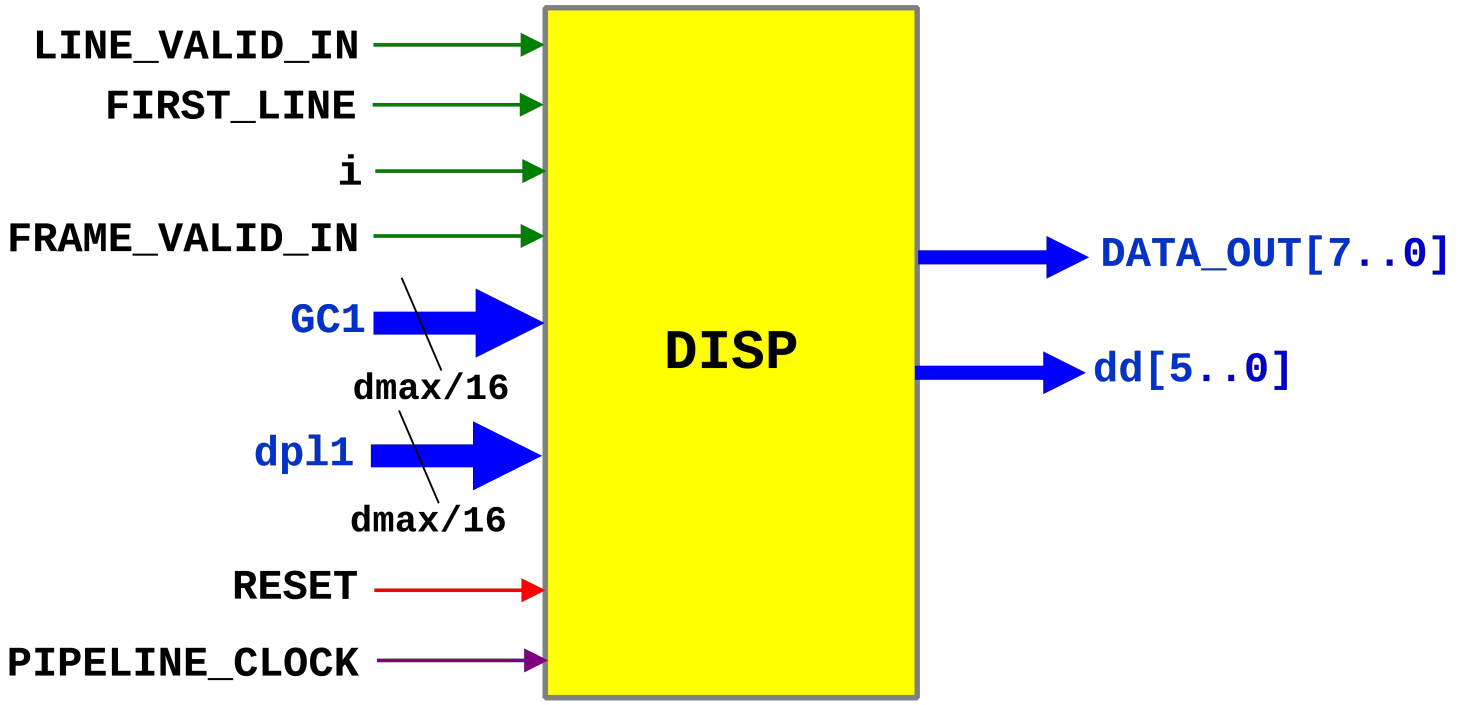
Progetto – process DPL10

- dmax/16 processi. Dai risultati ottenuti da *DPL00*, fornisce il secondo livello di costi globali minimi (e relative disparità). Alla fine di questo stadio abbiomo i minimi di gruppi a 16 (GC1) e costi indici di minimi.



Progetto – process DISP

- Determina il valore di disparità associato ai costi globali minimi di ogni direzione e al minimo della loro somma (costo globale totale). Di quest'ultimo, il valore minimo di disparità viene fornito in uscita dal modulo *Scanline*



Progetto LR RL UD – Block RAM

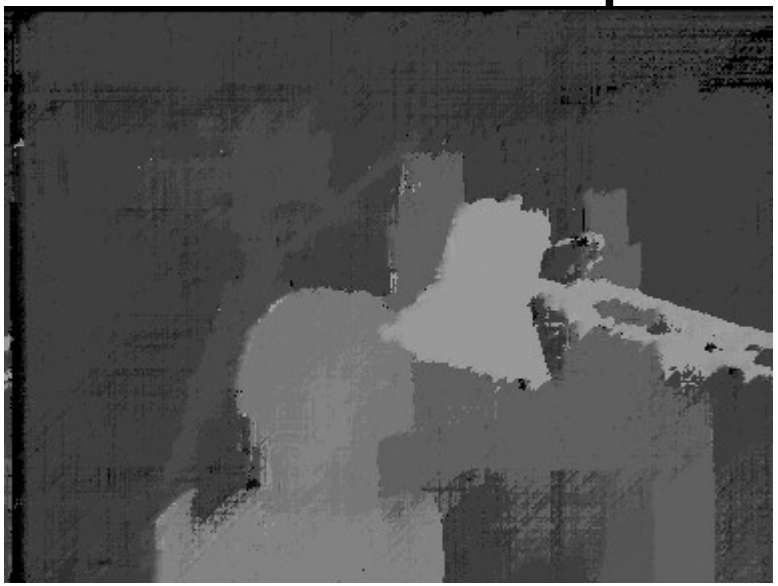
- Per la memorizzazione di alcuni dati sono state utilizzate 132 ***Block RAM*** da 9 Kb:
- *Block RAM Write First*:
  - *64 BRAM* in cui memorizzare il costo globale di UD su un'intera riga.
  - *1 BRAM* in cui memorizzare le disparità minime di UD di un'intera riga.
  - *1 BRAM* per mantenere i pixel di un'intera riga dell'immagine target.
- *Block RAM Read First*:
  - *64 BRAM* in cui memorizzare la somma dei costi globali di LR e UD su un'intera riga.
  - *1 BRAM* che memorizza, su un'intera riga di un'immagine, le disparità minime relative alla somma dei costi globali delle tre direzione UD, LR e RL.
  - *1 BRAM* per mantenere i pixel di un'intera riga dell'immagine reference.

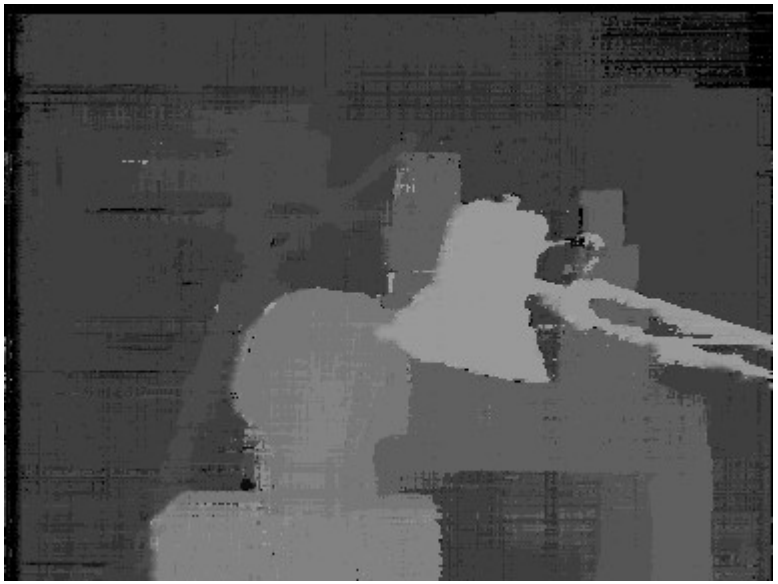
Progetto LR UD RD – Block RAM

- Per la memorizzazione di alcuni dati sono state utilizzate 130 ***Block RAM*** da 9 Kb:
- *Block RAM Write First*:
  - *64 BRAM* in cui memorizzare il costo globale di UD su un'intera riga.
  - *1 BRAM* in cui memorizzare le disparità minime di UD di un'intera riga.
  - *64 BRAM* in cui memorizzare il costo globale di RD su un'intera riga.
  - *1 BRAM* in cui memorizzare le disparità minime di RD di un'intera riga.

Differenze UD LR RL e UD RD LR

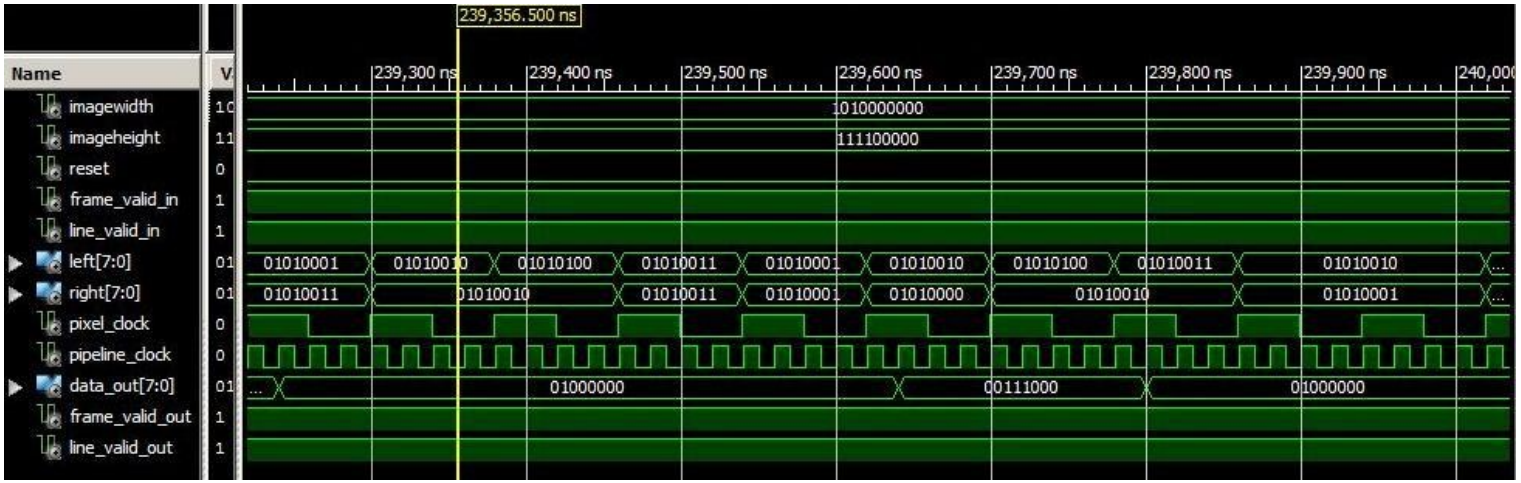
- Non serve stadio F(complessità minore)
- Meno memoria utilizata
- Ritardo di output minore.





Progetto – Testbench

- I test di simulazione e di sintesi sono stati eseguiti per immagini a 640x480 pixel, con disparità massima di *64* e penalità *P1 = 10* e *P2 = 40*.
- *PIXEL_CLOCK* a 40 nsec.
- *PIPELINE_CLOCK* a 10 nsec (ritaro da *PIXEL_CLOCK* di 1 nsec).



Conclusioni

- Sostituendo ad *RL* la direzione *RD* (*Right -> Down*) si possono ottenere risultati non molto dissimili con qualche miglioramento in termini di risorse utilizzate.
- In futuro si potrebbe pensare di ottimizare e aggiungere al modulo *Scanline* un'ulteriore direzione, a quelle attualmente presenti.
- Sono in corso tentativi d'integrazione del modulo *Scanline* nel progetto *Stereo Vision* del Prof. Mattoccia.