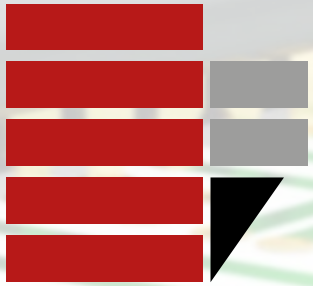
Università della Calabria

Dipartimento di Informatica, Modellistica, Elettronica e Sistematica



Corso di studio in

Ingegneria Elettronica

" FPGA implementation of spatial filtering circuit for 2D grayscale images"

Studenti

Giuseppe Pirilli 237909

Giuseppe Vizza 235491

Anno Accademico 2023/2024

Indice

- Introduzione
- Chip FPGA
- Filtraggio spaziale 2D
- Architettura del circuito di filtraggio
  - Memoria cache immagine
  - Memoria cache kernel isotropico
  - Modulo Convolutore (Pre-Adders + Multipliers + Albero di somma Carry-Save finale + stadi di pipeline)
  - Modulo di controllo (Macchina a stati finiti sincrona (FSM) + Contatore sincrono)
- Simulazione behavioral
- Implementazione su FPGA
- Simulazione post-implementation
- Analisi (Reports timing + performance + utilization + power)
- Risultati
- Conclusioni
- Miglioramenti futuri

Introduzione

Oggi giorno l'elaborazione di immagini digitali è applicata in diversi ambiti, a partire dal settore commerciale fino a quello della salute e della sicurezza.

L'image processing consiste nel sottoporre le immagini digitali, mediante l'impiego di specifici algoritmi e dispositivi elettronici, a una serie di operazioni matematiche al fine di apportare opportune modifiche.

Il filtraggio nel dominio spaziale rappresenta una delle tecniche di image processing più frequentemente utilizzate e si occupa di varie applicazioni pratiche come: nitidezza dell'immagine, sfocatura/smoothing, riduzione del rumore, accentuazione dei bordi, ecc.

Il filtraggio costituisce spesso la fase iniziale di algoritmi più complessi di image processing, che hanno come obiettivo la classificazione/rilevamento di oggetti, l'estrazione di caratteristiche specifiche, ecc.

Il filtraggio spaziale opera direttamente sul piano dell'immagine manipolando i valori d'intensità dei pixels della stessa mediante l'applicazione di maschere o finestre spaziali, anche note come filtri o kernel.

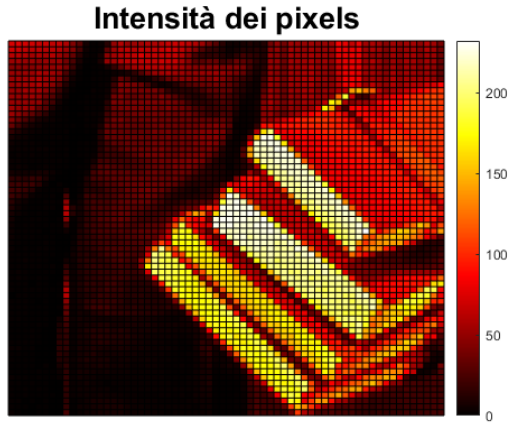
Introduzione

L'obiettivo è quello di gettare le basi per la conversione di algoritmi di filtraggio delle immagini ad alto livello, guidati da software, in implementazioni hardware in tecnologia FPGA. L'obiettivo a lungo termine è incentrato sulla riduzione dei vincoli temporali e sul miglioramento della velocità di elaborazione in modo da rendere tale circuito adatto per applicazioni in tempo reale.

A tale scopo, si procede con la progettazione, implementazione e analisi di un circuito in tecnologia FPGA capace di filtrare immagini di dimensioni 64×64 in scala di grigio a 8 bit unsigned impiegando kernel isotropici di tipo Laplaciano e Laplaciano-Gaussiano di dimensioni 5×5 con coefficienti a 8 bit signed in complemento a due.

Le prestazioni del circuito vengono analizzate in termini di latenza, frequenza di clock, utilizzo delle risorse e dissipazione di potenza ed i risultati ottenuti vengono confrontati, attraverso opportune metriche, con quelli generati dall'algoritmo di filtraggio ad alto livello.

Immagine originale



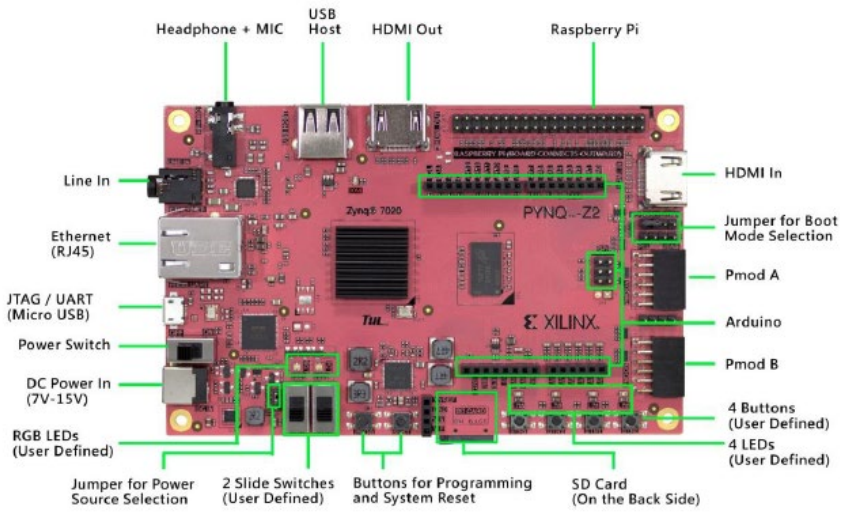
Kernel isotropico

KC1	KC5	KC2	KC5	KC1
KC5	KC3	KC4	KC3	KC5
KC2	KC4	KC6	KC4	KC2
KC5	KC3	KC4	KC3	KC5
KC1	KC5	KC2	KC5	KC1

Chip FPGA

FPGA scelto per l'implementazione e l'analisi del circuito di filtraggio

$Zynq\ XC7Z020 - 1CLG400C$ in dotazione alla scheda di sviluppo $Pynq - Z2$



ZYNQ XC7Z020-1CLG400C
- 650MHz dual-core Cortex-A9 processor
- DDR3 memory controller with 8 DMA channels and 4 High Performance AXI3 Slave ports
- High-bandwidth peripheral controllers: 1G Ethernet, USB 2.0, SDIO
- Low-bandwidth peripheral controller: SPI, UART, CAN, I2C
- Programmable from JTAG, Quad-SPI flash, and microSD card
- Programmable logic equivalent to Artix-7 FPGA
  - 13,300 logic slices, each with four 6-input LUTs and 8 flip-flops
  - 630 KB of fast block RAM
  - 4 clock management tiles, each with a phase-locked loop (PLL) and mixed-mode clock manager (MMCM)
  - 220 DSP slices
  - On-chip analog-to-digital converter (XADC)

Memory
- 512MB DDR3 with 16-bit bus @ 1050Mbps
- 16MB Quad-SPI Flash with factory programmed 48-bit globally unique EUI-48/64™ compatible identifier
- microSD slot

Power
- Powered from USB or 7V-15V external power source

USB and Ethernet
- Gigabit Ethernet PHY
- Micro USB-JTAG Programming circuitry
- Micro USB-UART bridge
- USB 2.0 OTG PHY (supports host only)

Audio and Video
- HDMI sink port (input)
- HDMI source port (output)
- I2S interface with 24bit DAC with 3.5mm TRRS jack
- Line-in with 3.5mm jack

Switches, Push-buttons and LEDs
- 4 push-buttons
- 2 slide switches
- 4 LEDs
- 2 RGB LEDs

Expansion Connectors
- Two standard Pmod ports
  - 16 Total FPGA I/O (8 shared pins with Raspberry Pi connector)
- Arduino Shield connector
  - 24 Total FPGA I/O
  - 6 Single-ended 0-3.3V Analog inputs to XADC
- Raspberry Pi connector
  - 28 Total FPGA I/O (8 shared pins with Pmod A port)

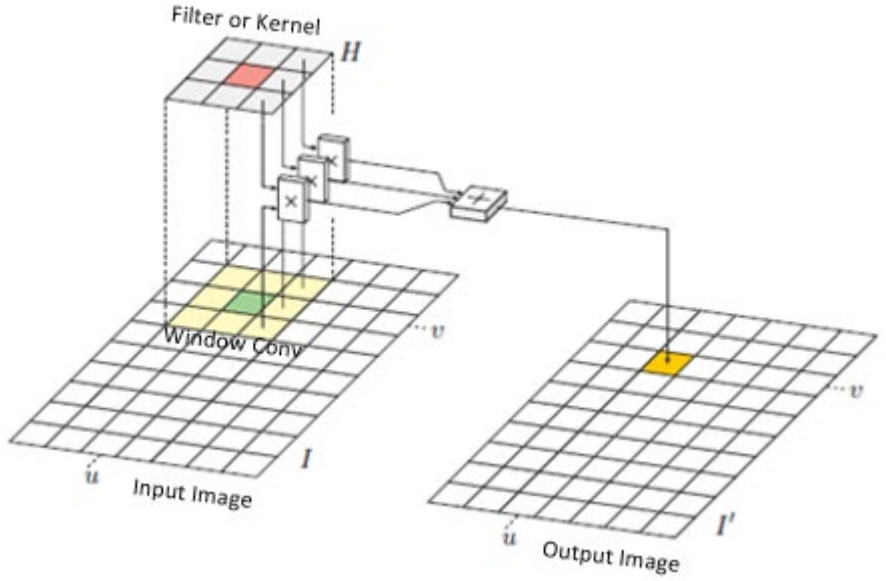
Filtraggio spaziale 2D

Il filtraggio spaziale è riconducibile ad un'operazione di convoluzione che esegue la somma dei prodotti tra l'immagine di input e una maschera/finestra più piccola cosiddetta filtro/kernel centrata, generalmente, sul generico pixel da filtrare.

La dimensione del kernel determina la dimensione dell'intorno del generico pixel da filtrare, che costituisce la cosiddetta finestra di convoluzione.

La scelta dei coefficienti del kernel determina l'obiettivo del filtraggio: nitidezza, sfocatura/smoothing, riduzione del rumore, rilevamento dei bordi, miglioramento del contrasto, correzione del colore, ecc.

Utilizzo di tecniche di padding per risolvere le problematiche introdotte dai pixels situati ai bordi dell'immagine.

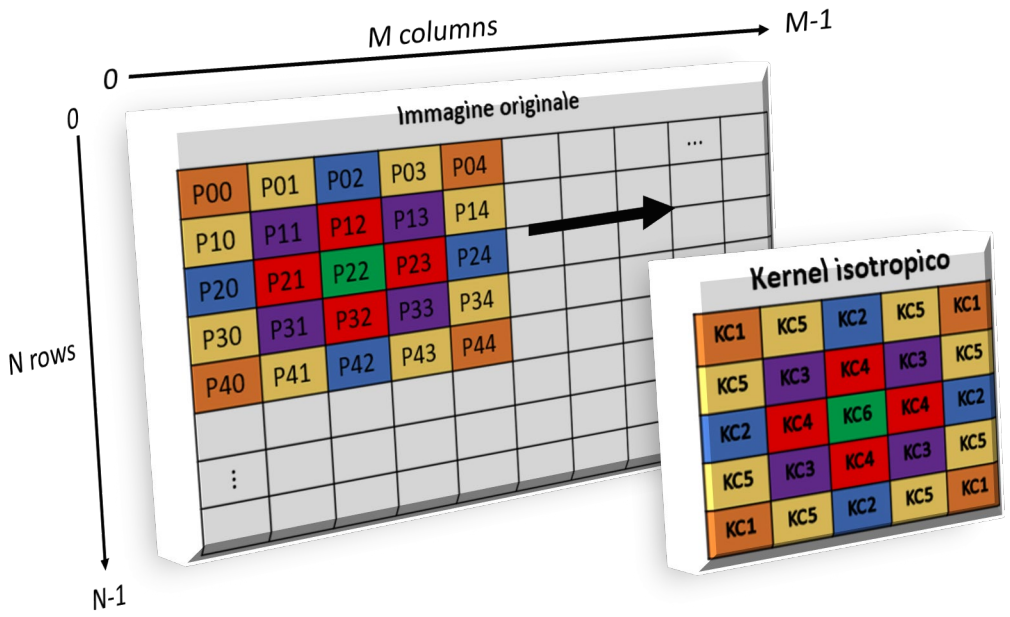


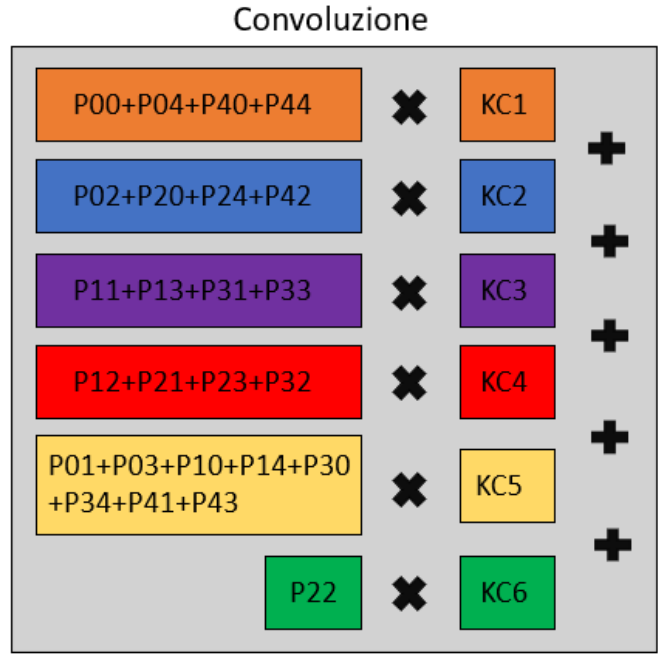
$$I'(u,v) = \sum_{i=0}^{kernel_{height}-1} \sum_{j=0}^{kernel_{width}-1} I(u+i, v+j) \cdot H(i,j)$$

Filtraggio spaziale 2D

Kernel non isotropico → Richiede una notevole quantità di calcoli → Per un kernel 5×5 sono richieste 25 moltiplicazioni e 24 addizioni

Kernel isotropico → Consente di ridurre il numero di moltiplicazioni attraverso il raggruppamento dei pixels accomunati dallo stesso coefficiente → Per un kernel 5×5 sono richieste 6 moltiplicazioni e 24 addizioni





Architettura del circuito di filtraggio

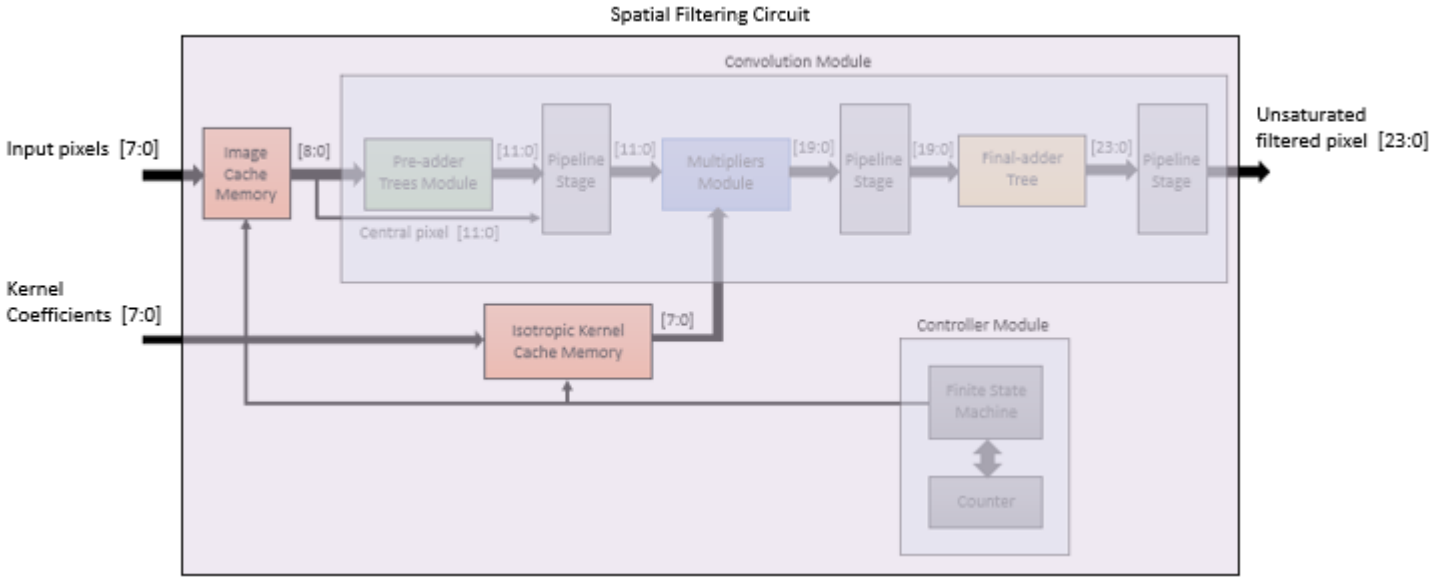
Il circuito è composto dai seguenti moduli:

1. Cache associata alla memorizzazione dei pixels di input

2. Cache associata alla memorizzazione dei coefficienti del kernel

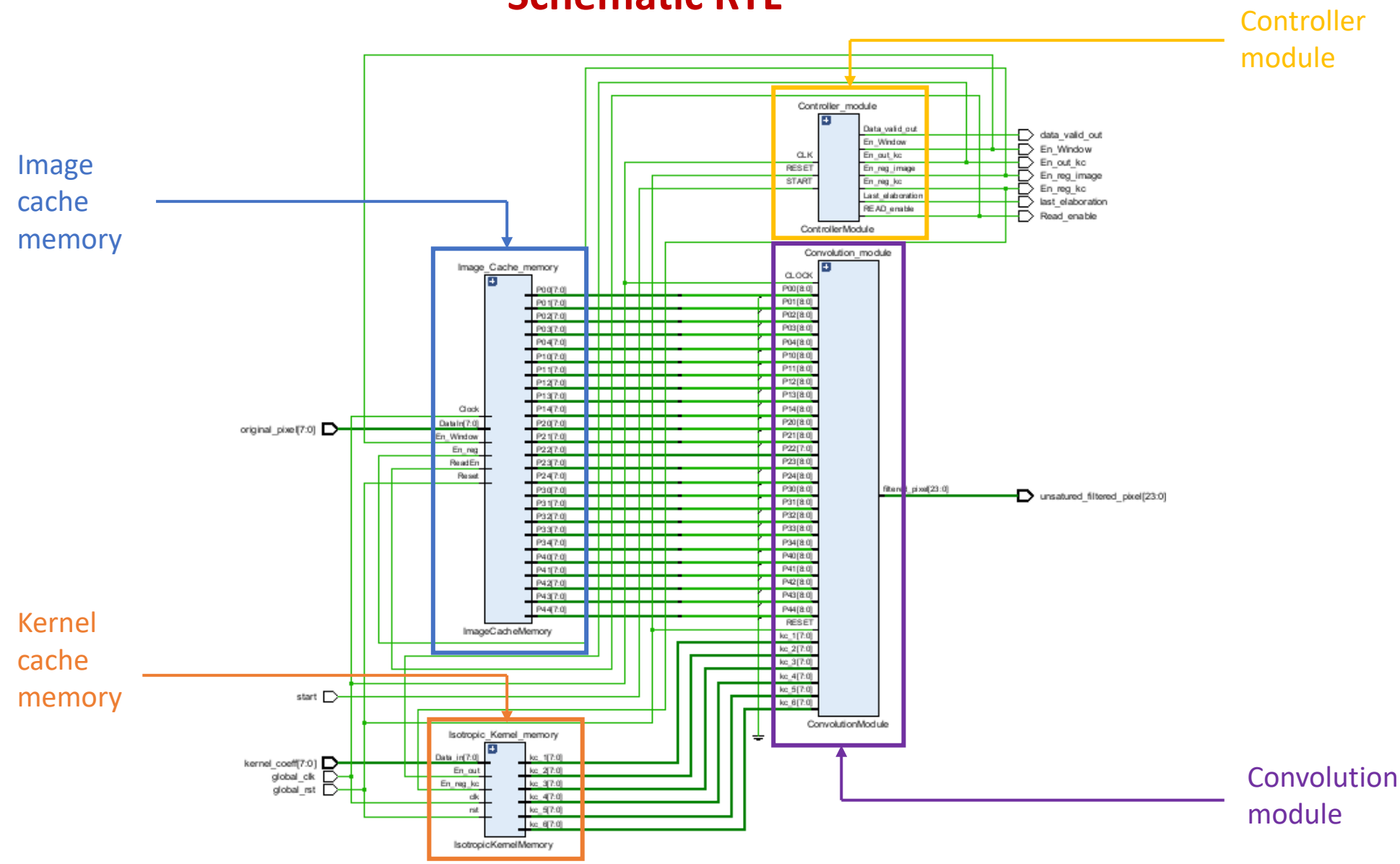
3. Modulo convolutore
   - Modulo Pre-adders
   - Modulo Multipliers
   - Albero di somma Carry-Save finale
   - Stadi di pipeline

4. Modulo di controllo
   - Contatore
   - Macchina a stati finiti (FSM)



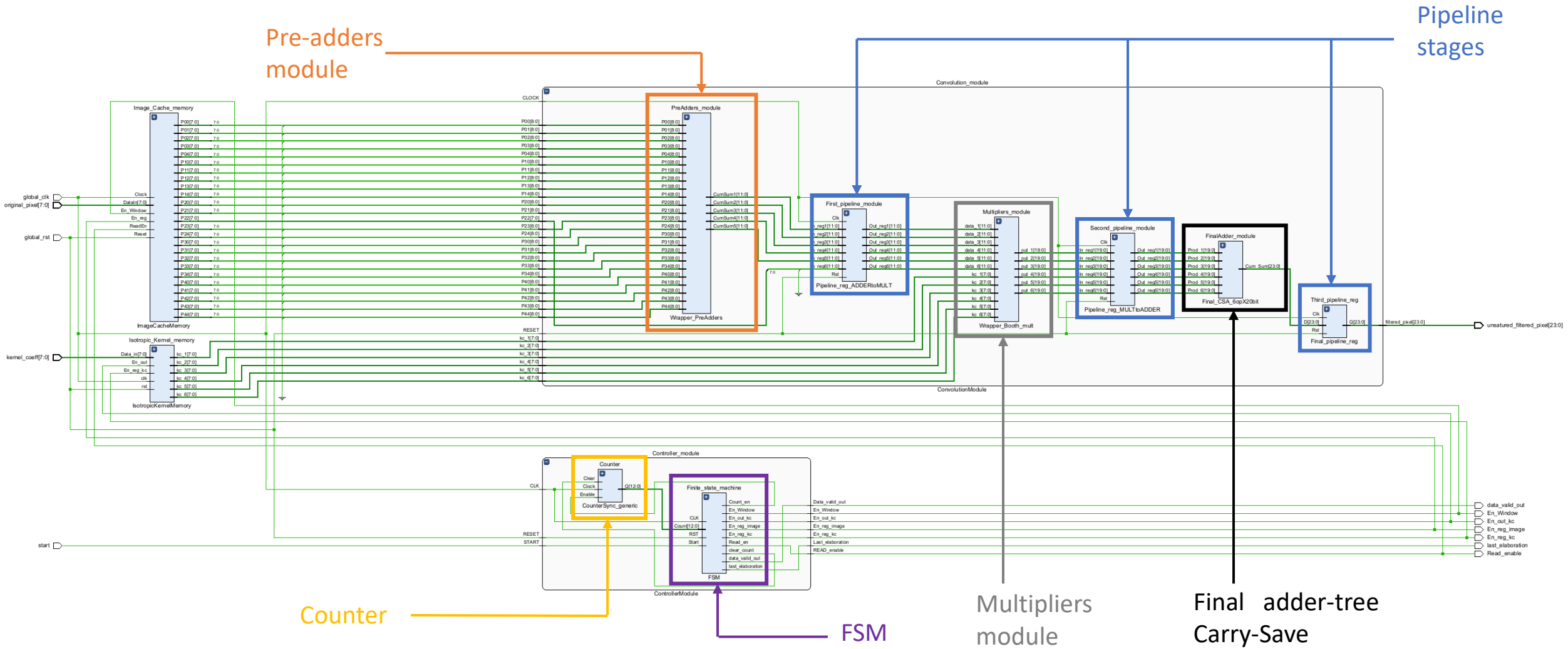
Architettura del circuito di filtraggio

Schematic RTL



Architettura del circuito di filtraggio

Schematic RTL esploso



Memoria cache immagine

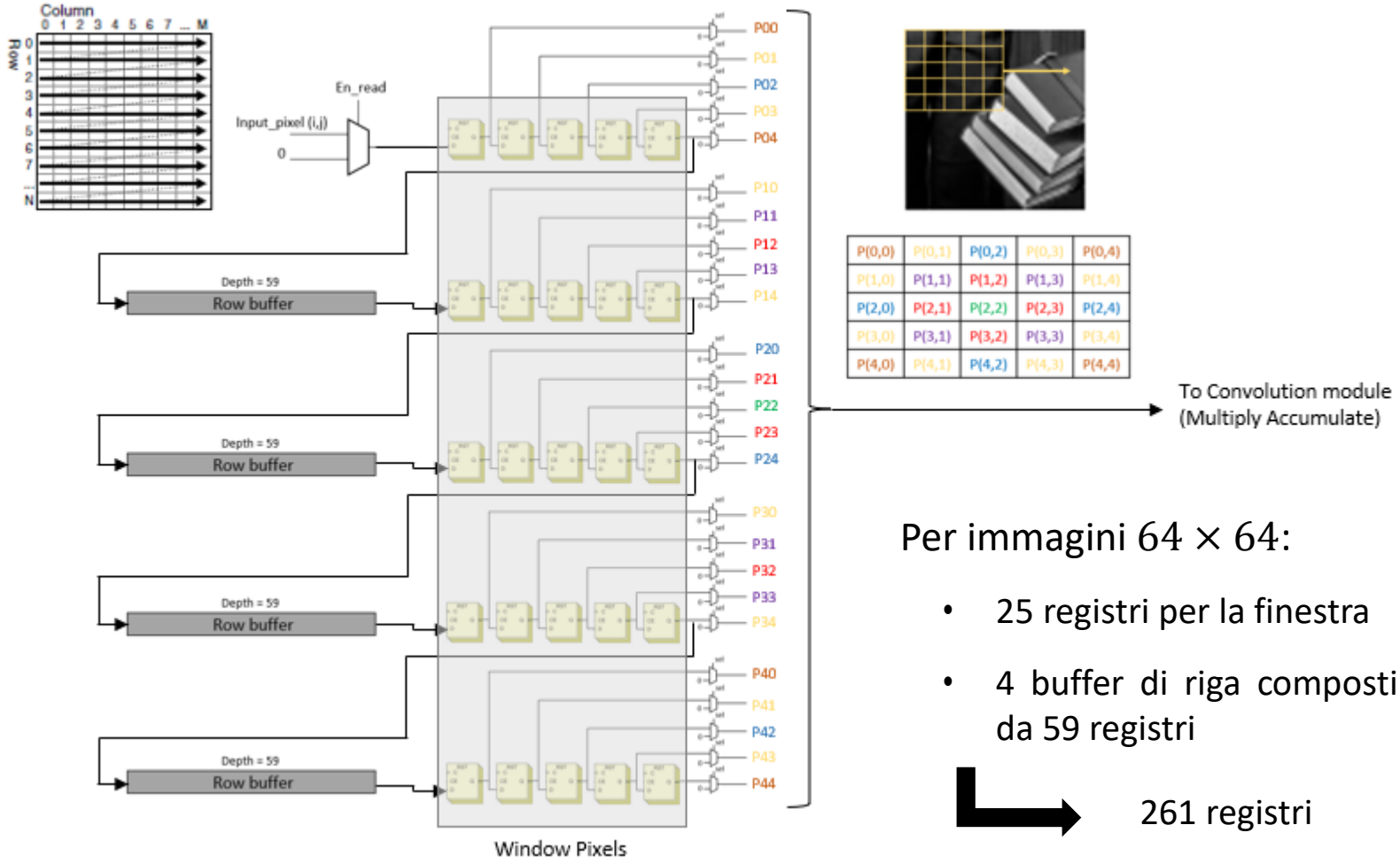
Si tratta sostanzialmente di un buffer composto da una cascata di registri sincroni.

Vantaggi:

- Evita la bufferizzazione dell'intera immagine nelle BRAMs presenti sul chip FPGA.
- Garantisce la corretta costruzione della finestra di convoluzione del generico pixel da filtrare, essendo questa costituita da pixels tra loro non adiacenti.
- Risolve la problematica legata ai pixels situati ai bordi dell'immagine attuando un'estensione ibrida (zero padding + toroidale).

Svantaggi:

- Latenza del circuito pari a 131 cicli di clock prima di ottenere la finestra utile all'elaborazione.



Per immagini 64×64:

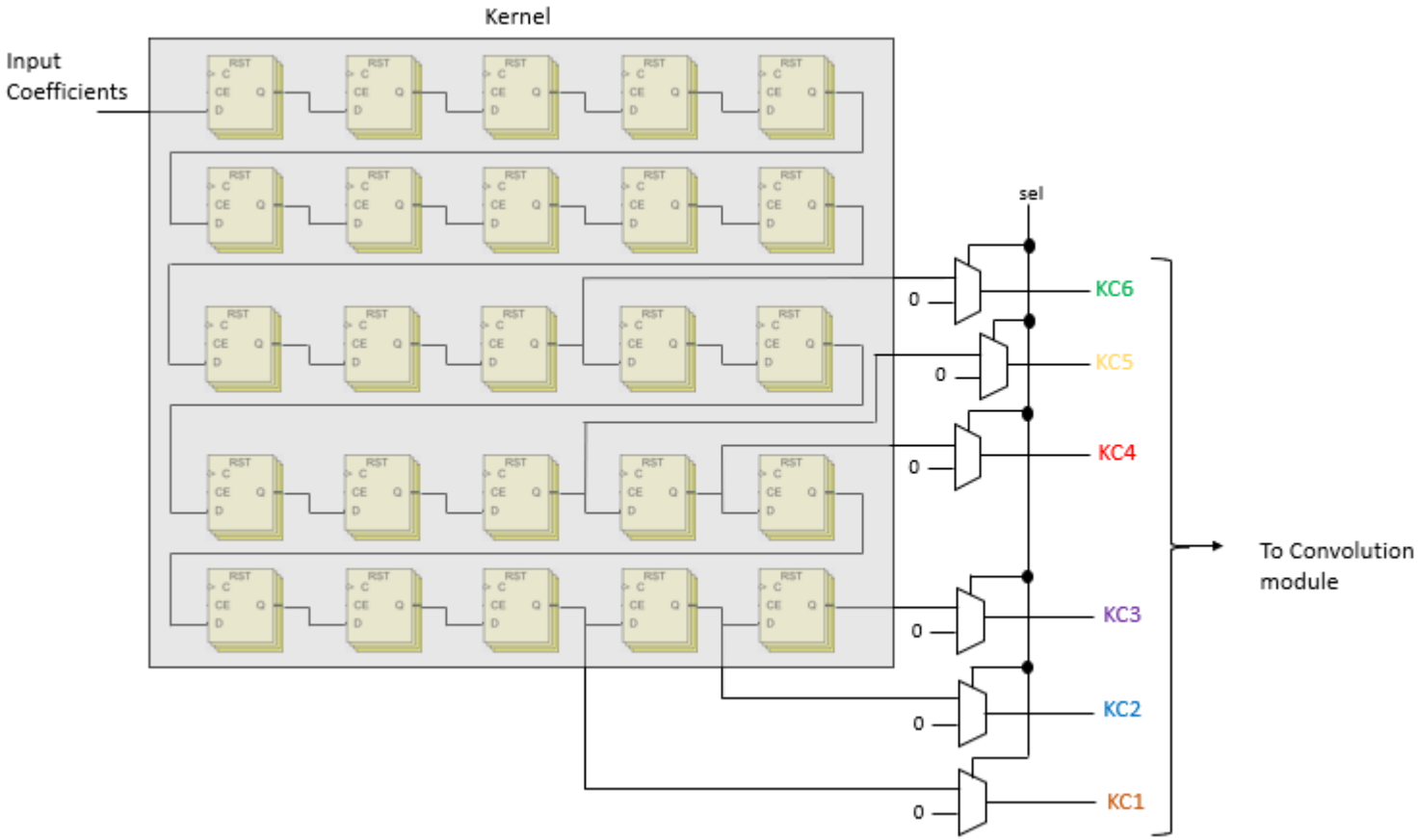
- 25 registri per la finestra
- 4 buffer di riga composti da 59 registri

→ 261 registri

Memoria cache kernel isotropico

Buffer composto da una cascata di registri sincroni.

Utilizzato per la memorizzazione dei coefficienti del kernel.



Per un kernel isotropico 5×5:

- 25 registri.

Modulo Pre-adders

Somma i pixels accomunati dallo stesso coefficiente.

Composto da:

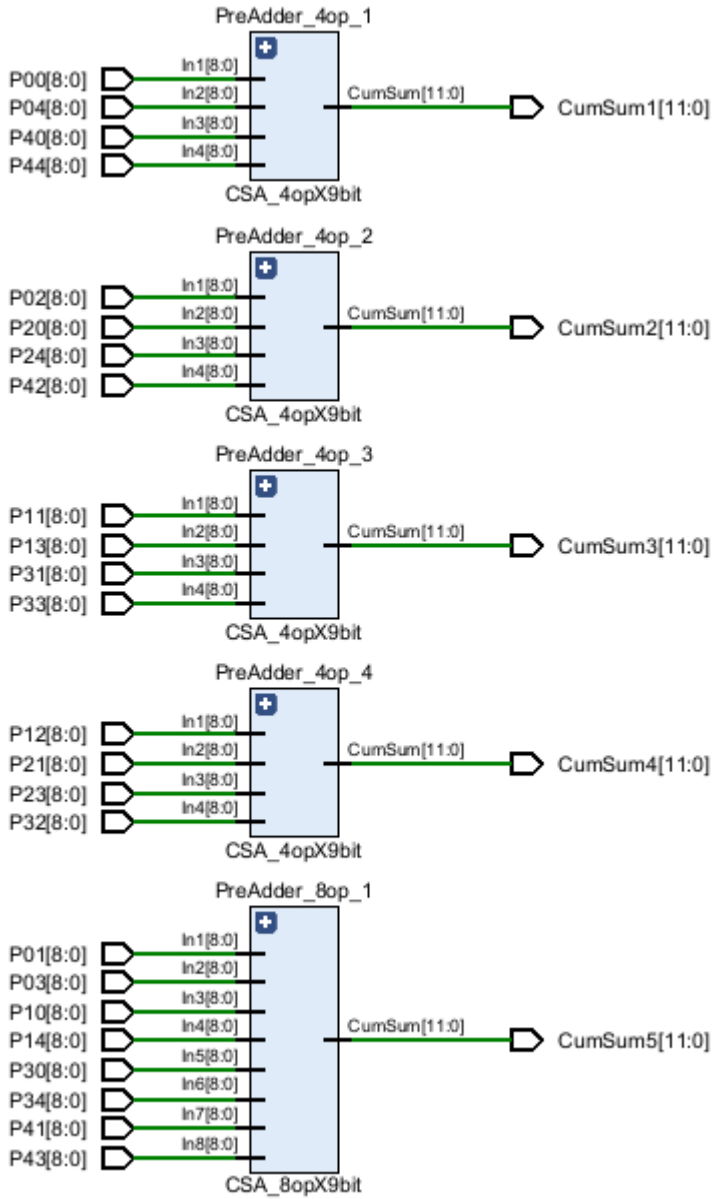
- 4 alberi di somma Carry-Save a 4 operandi
- 1 albero di somma Carry-Save a 8 operandi

Vantaggi approccio Carry-Save rispetto all'impiego di un sommatore tradizione Ripple-Carry:

- Riduzione del tempo di elaborazione

Sommatore a 4 operandi	
Albero Ripple-Carry-Adder	**Albero Carry-Save-Adder**
$\tau = \tau_{RCA(n+1)} + \tau_{RCA(n+2)}$	$\tau = 2\tau_{FA} + \tau_{RCA(n+2)}$

Sommatore a 8 operandi	
Albero Ripple-Carry-Adder	**Albero Carry-Save-Adder**
$\tau = \tau_{RCA(n+1)} + \tau_{RCA(n+2)} + \tau_{RCA(n+3)}$	$\tau = 4\tau_{FA} + \tau_{RCA(n+4)}$

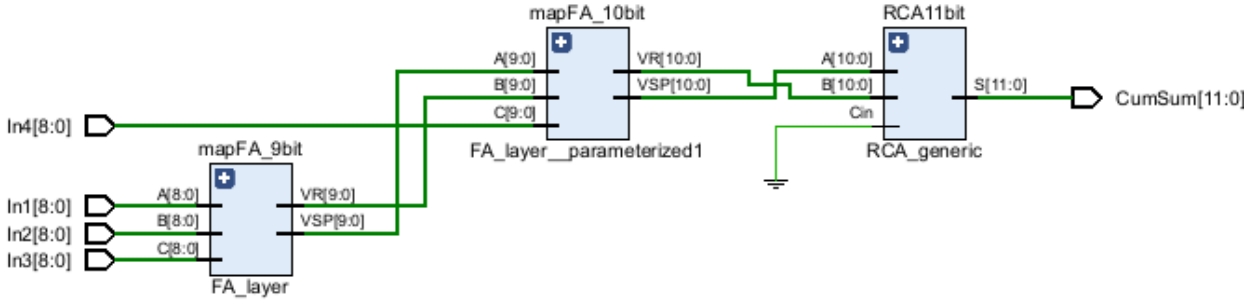


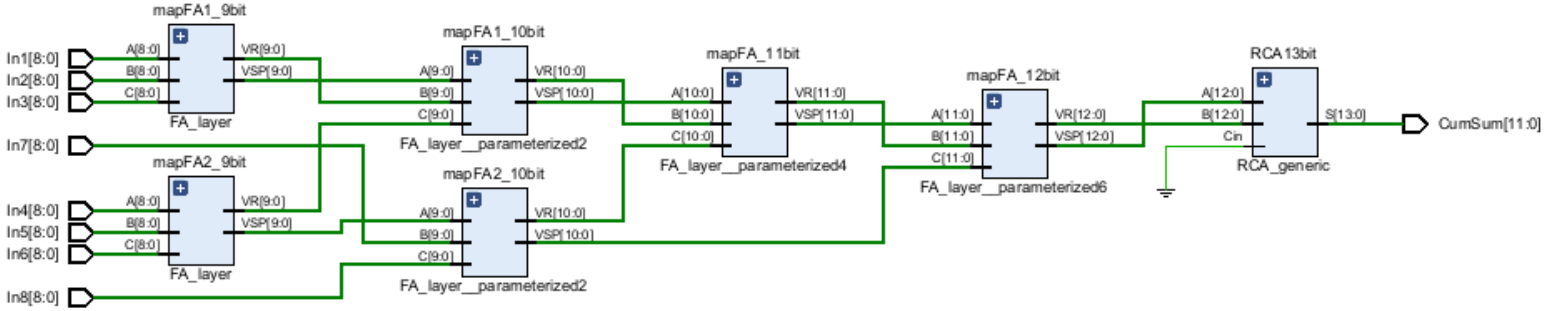
Modulo Pre-adders

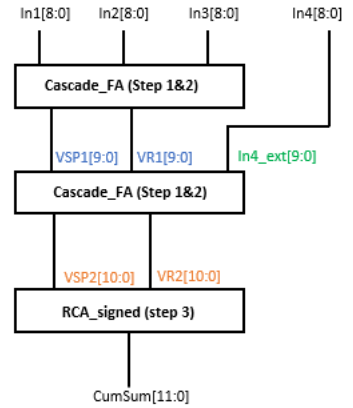
Step 1: sommiamo le terne di operandi singolarmente e raggruppiamo i bit di somma e di carry in VSP e VR

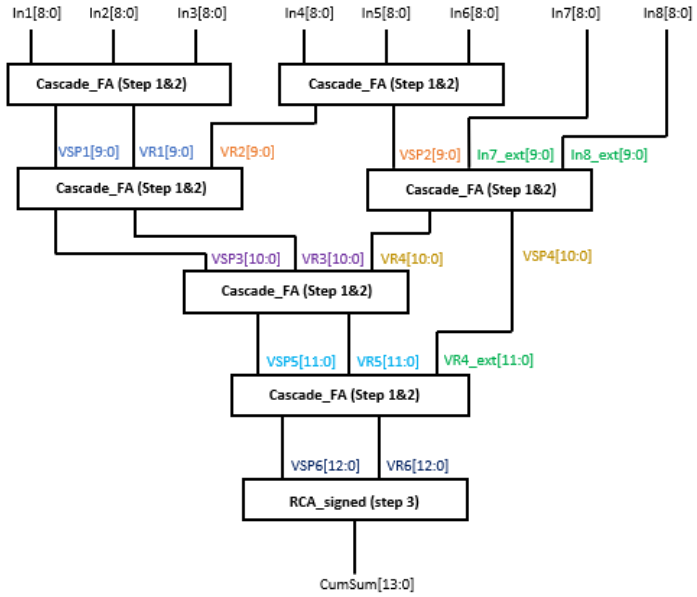
Step 2: estensione di VSP e shift a sinistra di VR

Step 3: somma finale con sommatore tradizionale









Il risultato del sommatore a 8 operandi può essere rappresentato su 12 bit

Sommatori progettati con notazione signed in complemento a 2

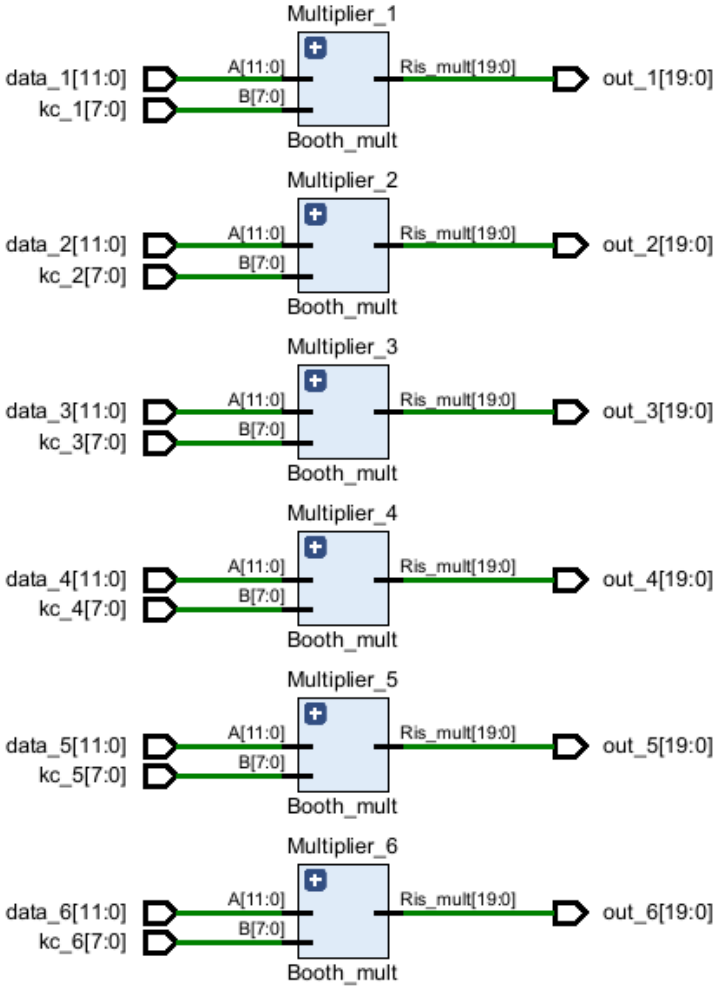
Pixels in ingresso estesi con zeri a 9 bit per una corretta rappresentazione del risultato finale

Modulo Multipliers

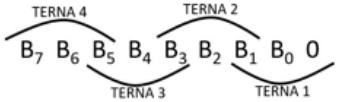
Si occupa della moltiplicazione tra le uscite del modulo Pre-adders + pixel centrale ed il rispettivo coefficiente del kernel.

Composto da:

- 6 moltiplicatori di Booth in parallelo



Suddivisione in terne e codifica del moltiplicatore per dimezzare il numero di prodotti parziali.



Terna b_{i+1} b_i b_{i-1}	Digit codificato	Prodotto parziale
0 0 0	0	0
0 0 1	1	A
0 1 0	1	A
0 1 1	2	2A
1 0 0	-2	-2A
1 0 1	-1	-A
1 1 0	-1	-A
1 1 1	0	0

MOLTIPLICATORE = COEFFICIENTE KERNEL

MOLTIPLICANDO = USCITA PRE ADDERS & PIXEL CENTRALE

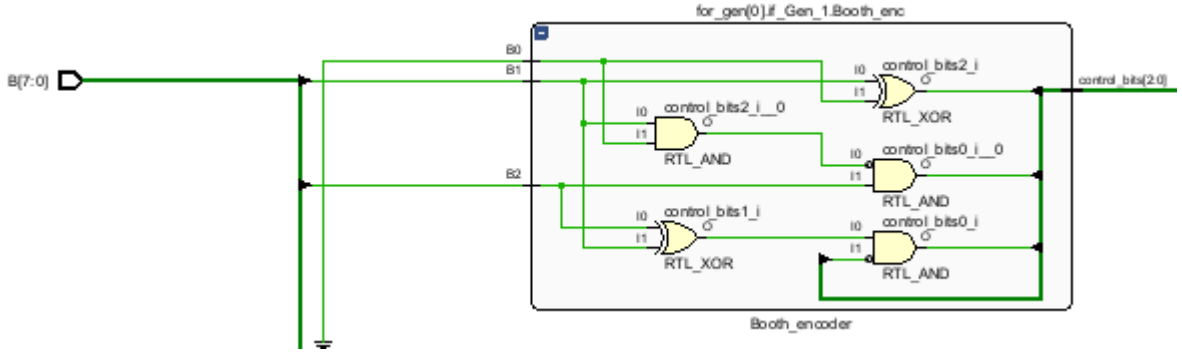
Modulo Multipliers

Un moltiplicatore di Booth è composto da:

- Booth encoders, per la codifica delle terne di bit del moltiplicatore che verrà utilizzata come selettore dei multiplexers a valle.

Codifica effettuata utilizzando la rappresentazione modulo e segno.

B(i+1) B(i) B(i-1)	Cj(2:0)	Cj(2) Cj(1) Cj(0)
0 0 0	0	0 0 0
0 0 1	1	0 0 1
0 1 0	1	0 0 1
0 1 1	2	0 1 0
1 0 0	-2	1 1 0
1 0 1	-1	1 0 1
1 1 0	-1	1 0 1
1 1 1	0	0 0 0

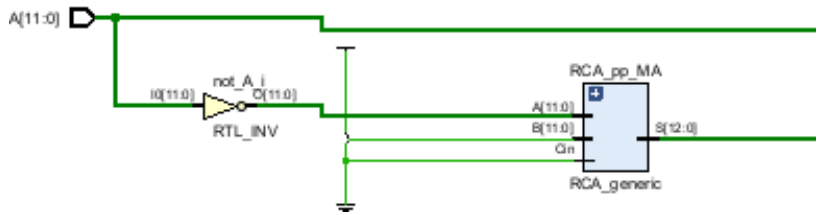



```
control_bits(2)<= (B1 nand B0) and B2;
control_bits(1)<= (B2 xor B1) and (not(B1 xor B0));
control_bits(0)<= (B1 xor B0);
```

Riduzione del fan-in dei multiplexers e, conseguentemente, della dissipazione di potenza.

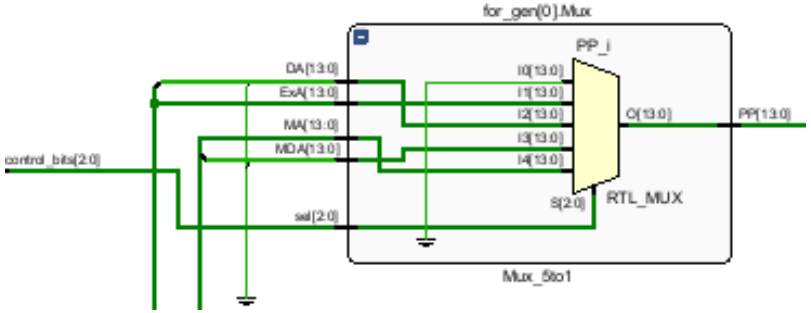
Modulo Multipliers

- Logica di pre-calcolo, restituisce i prodotti parziali che possono essere associati alla generica terna.




```
not_A <= not(A);
ExA <= (A(11)&A(11)&A);          -- (1A), estensione con segno di A
DA <= (ExA(12 downto 0)&'0');    -- (2A), estensione e shift a sinistra di A
RCA_pp_MA: RCA_generic generic map(WidthRCA=>12) port map(A=>not_A, B=>one12, Cin=>'0', S=>pp_MA);  -- (-A), per complemento a 2 di A
MA <= (pp_MA(12)&pp_MA);         -- (-1A), estensione con segno di -A
MDA <= (MA(12 downto 0)&'0');    -- (-2A), estensione e shift a sinistra di -A
```

- Multiplexers, sulla base della codifica dei Booth encoders, selezionano e portano in uscita il prodotto parziale associato a quella particolare codifica.

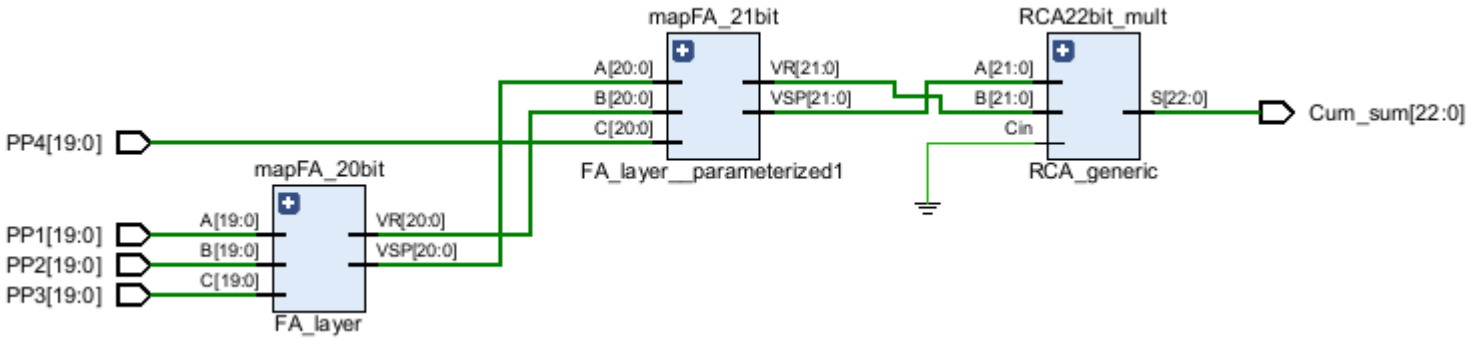



```
process(sel,ExA, DA, MA, MDA)
    begin
        case sel is
            when "000"  =>
                PP <= (others =>'0');    --0 (digit codificato pari a 1)
            when "001"  =>
                PP <= ExA;               --1A (digit codificato pari a 1)
            when "010"  =>
                PP <= DA;                --2A (digit codificato pari a 2)
            when "110"  =>
                PP <= MDA;               -- -2A (digit codificato pari a -2)
            when "101"  =>
                PP <= MA;                -- -1A (digit codificato pari a -1)
            when others =>
                PP <= (others => 'X');
        end case;
end process;
```

Modulo Multipliers

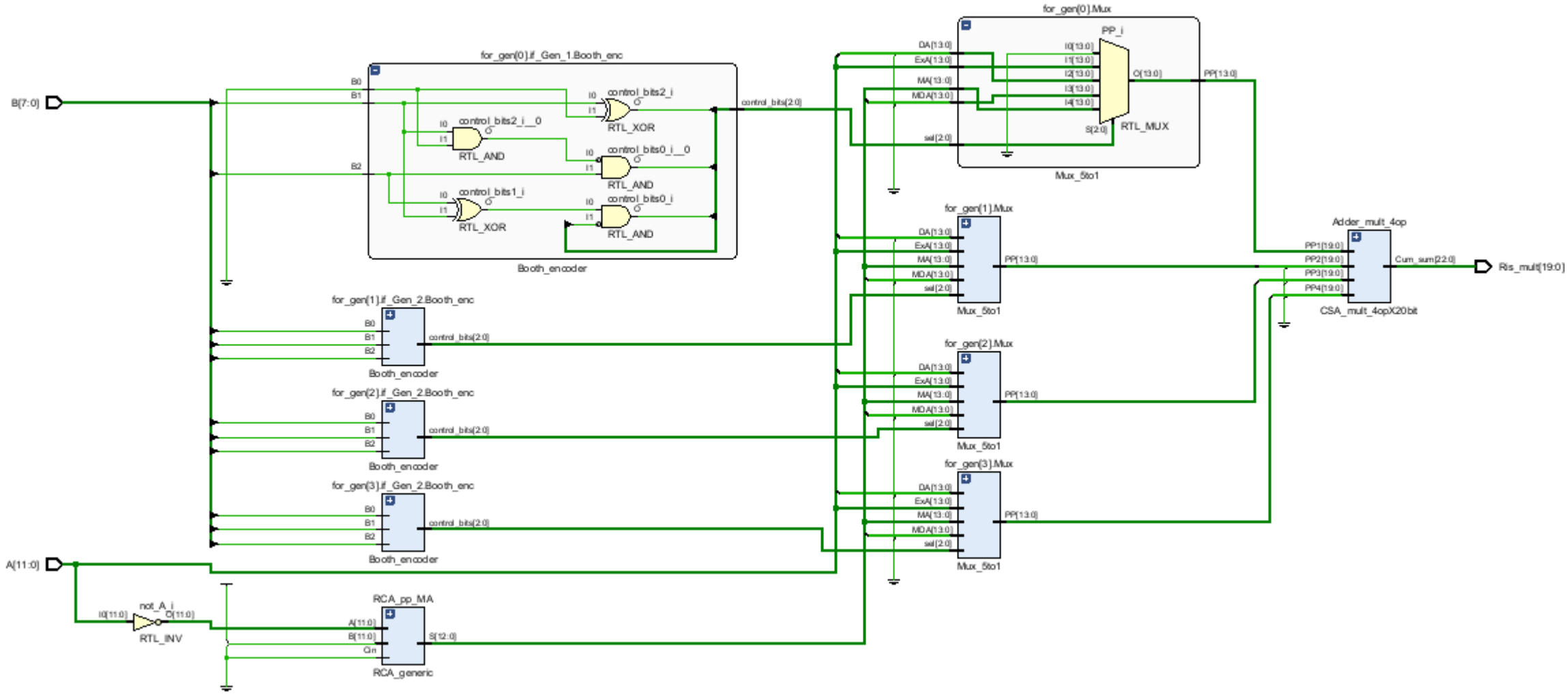
- Albero di somma Carry-Save, si occupa della somma dei prodotti parziali in uscita dai multiplexers precedentemente allineati (shift + estensione con segno).

```
--allineamento dei prodotti parziali in uscita dai MUX (shift ed estensione consegno)
PP1_int <= (19 downto 14 => partial_prod(0)(13)) & partial_prod(0);
PP2_int <= (19 downto 16 => partial_prod(1)(13))& partial_prod(1) & "00";
PP3_int <= (19 downto 18 => partial_prod(2)(13))&partial_prod(2)&"0000";
PP4_int <= partial_prod(3)&"000000";
```



Modulo Multipliers

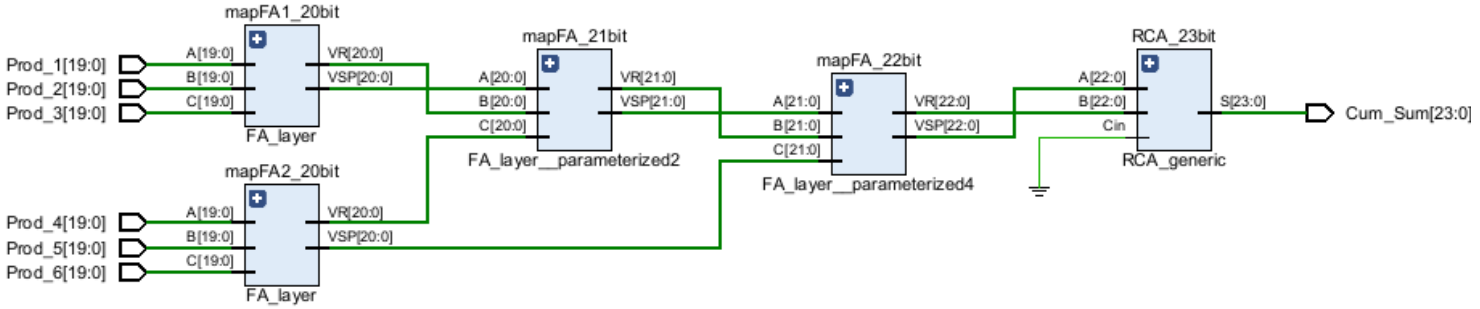
Schematic RTL

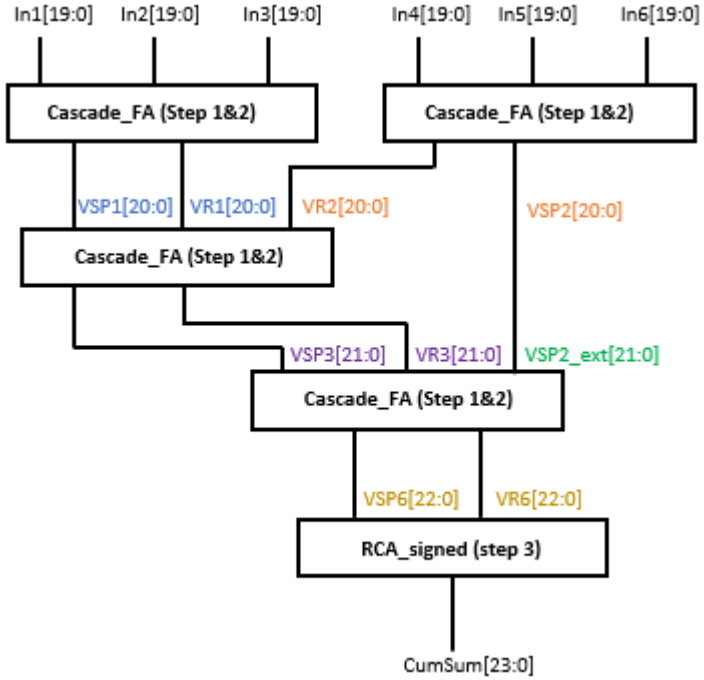


Il risultato dell'operazione di moltiplicazione può essere rappresentato su 20 bit

Albero di somma Carry-Save finale

Si occupa di sommare i dati in uscita dal modulo multipliers generando il risultato complessivo dell'elaborazione (pixel filtrato).

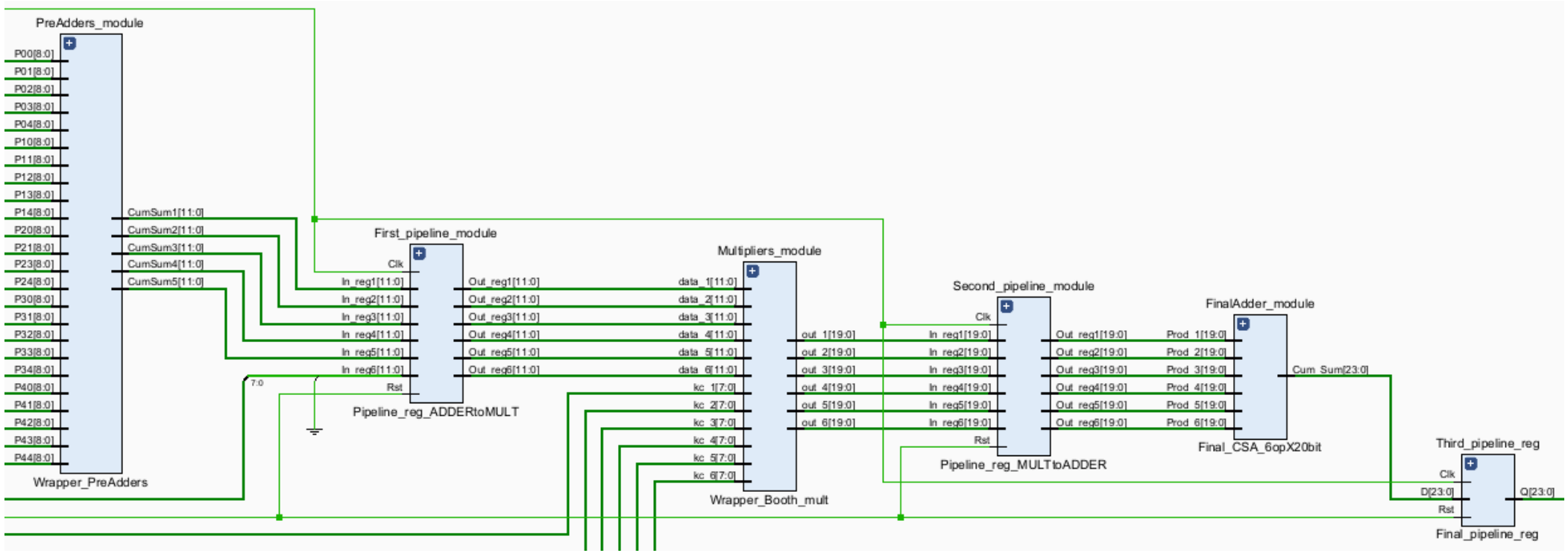




Stadi di pipeline

Consentono di spezzare la logica combinatoria distribuendola su più cicli di clock, riducendo il ritardo combinatorio.

- Migliora il throughput complessivo del circuito (si raggiunge un throughput unitario).
- Riduzione dei glitches sui segnali in uscita da ciascun blocco combinatorio con conseguente riduzione della dissipazione di potenza.

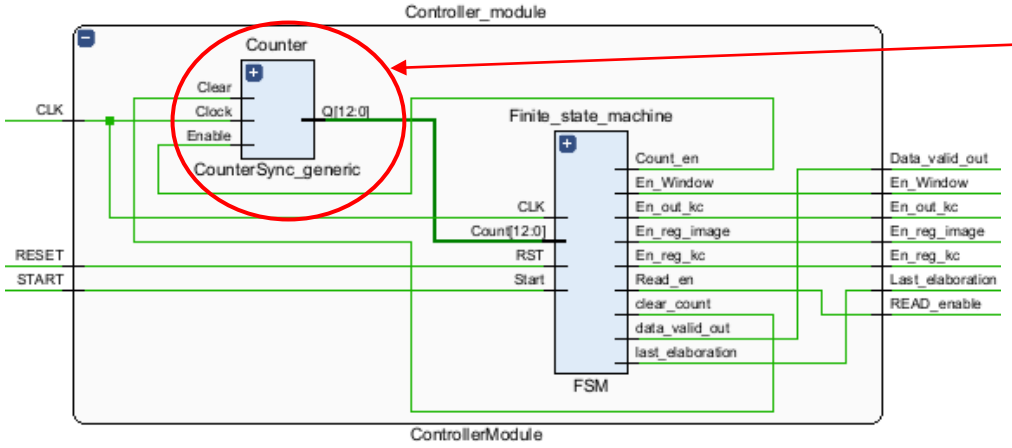


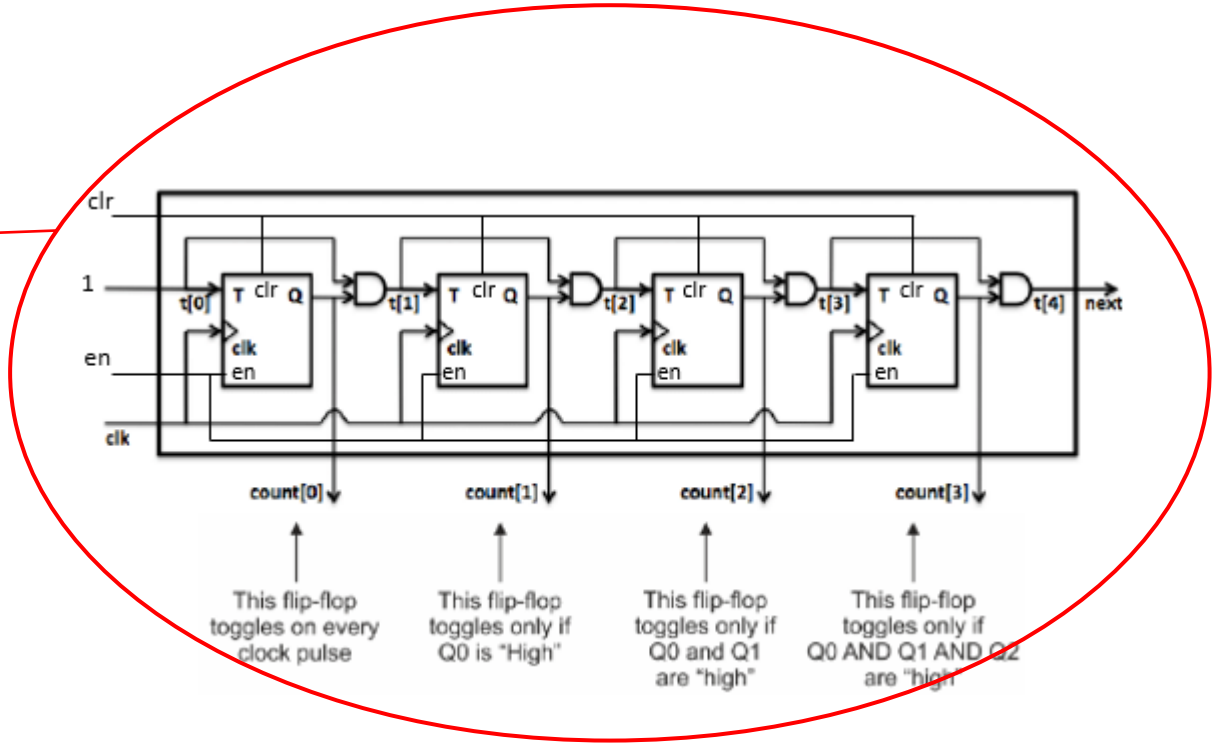
Modulo di controllo

Si occupa della gestione del circuito di filtraggio durante il processo di elaborazione dei dati.

Composto da:

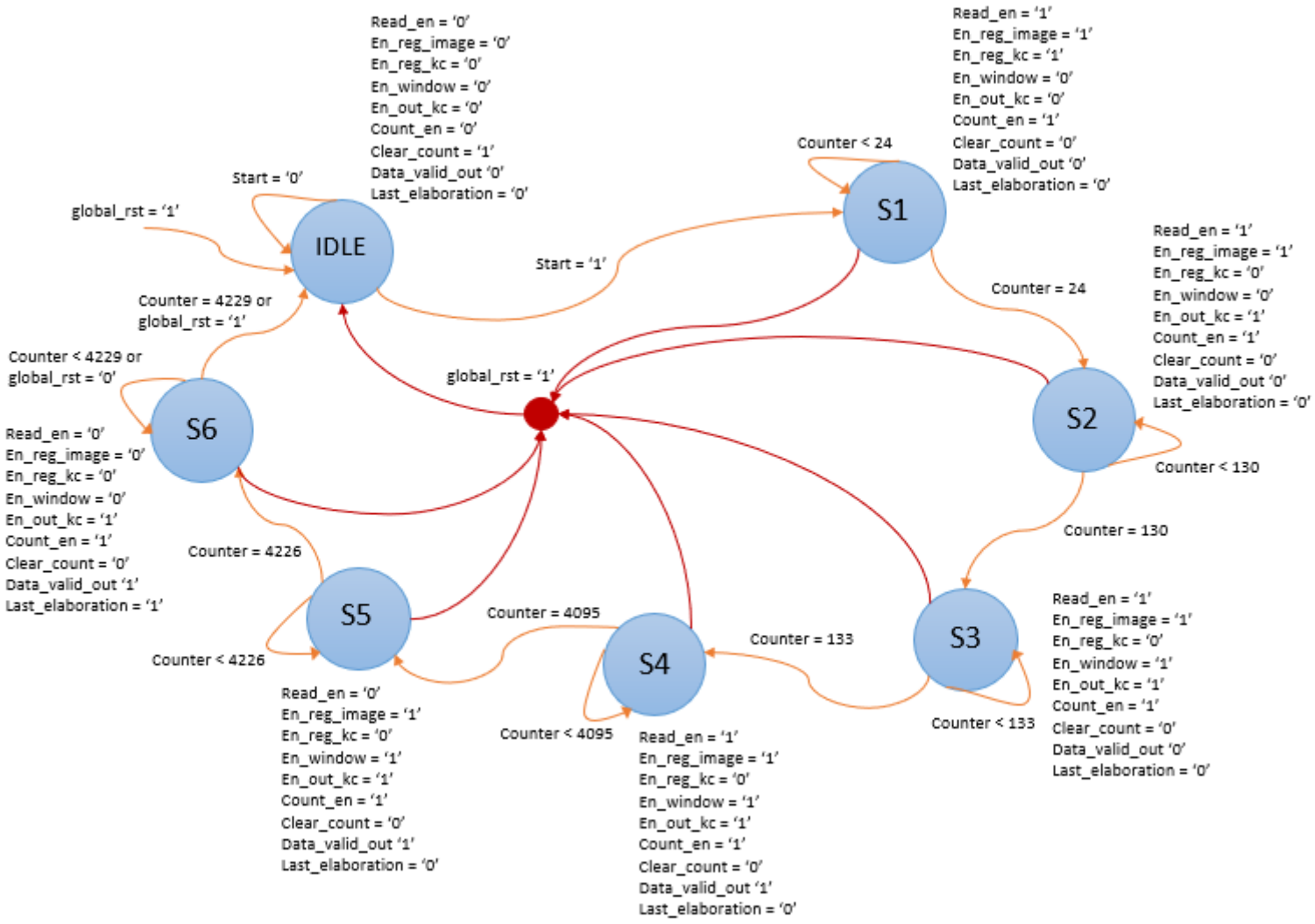
- Macchina a stati finiti (FSM) sincrona, gestisce gli stati del sistema agendo sui segnali di controllo dei vari moduli di quest'ultimo.
- Contatore sincrono, gestisce le transizioni dallo stato corrente a quello successivo.





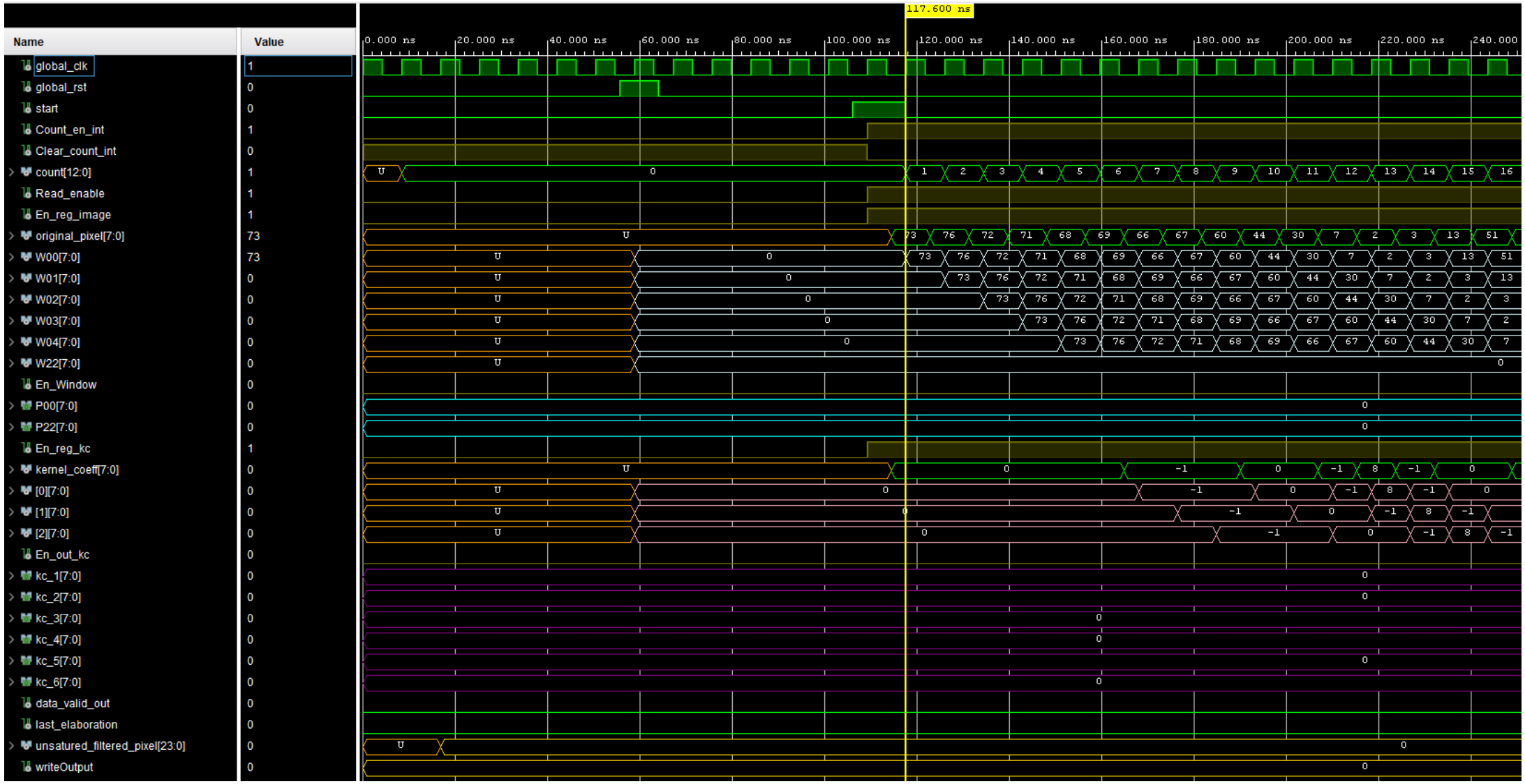
Modulo di controllo

Diagramma degli stati



Simulazione Behavioral

S1 - Inizio lettura pixels e coefficienti kernel

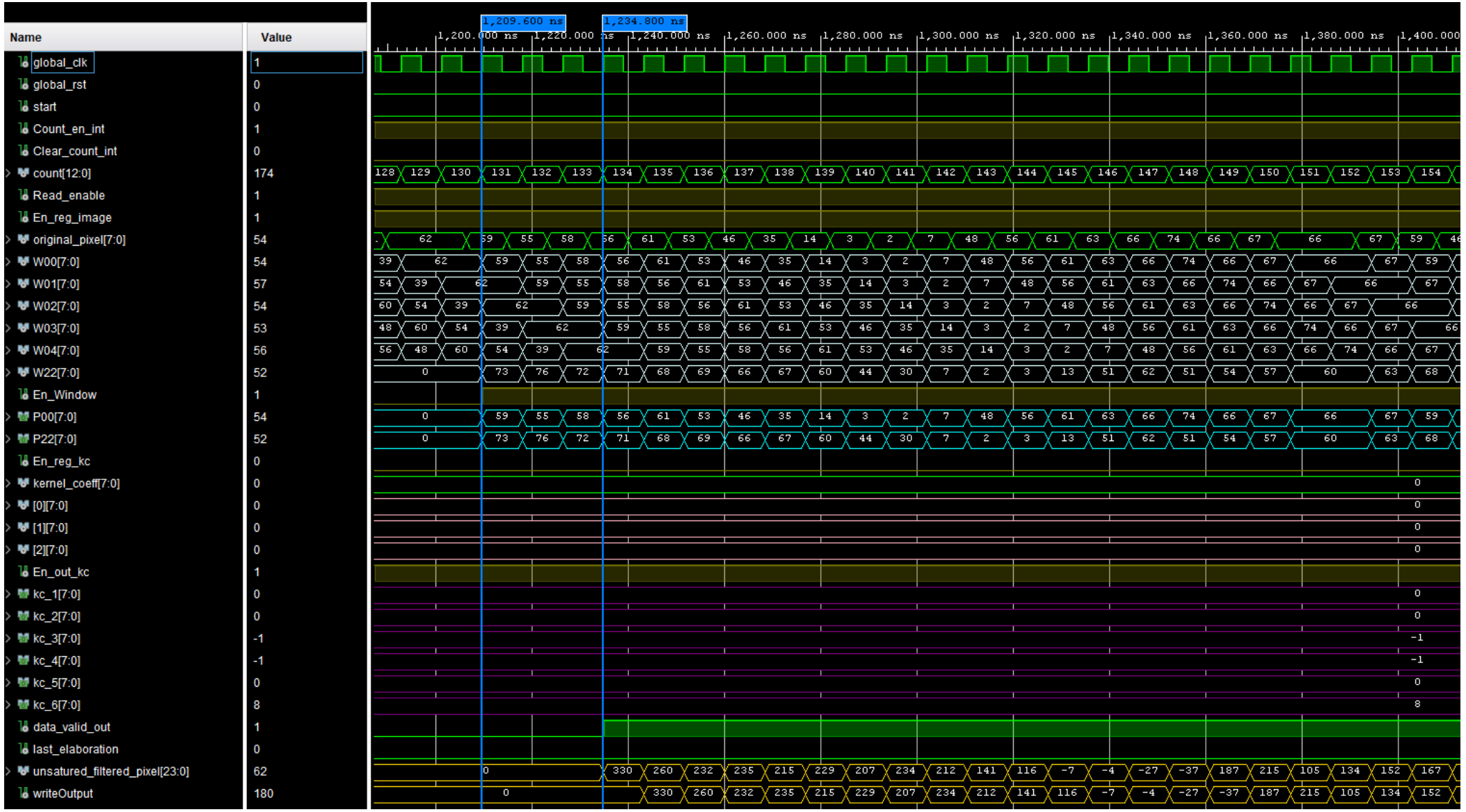


Simulazione Behavioral

S2 - Stop lettura coefficienti kernel

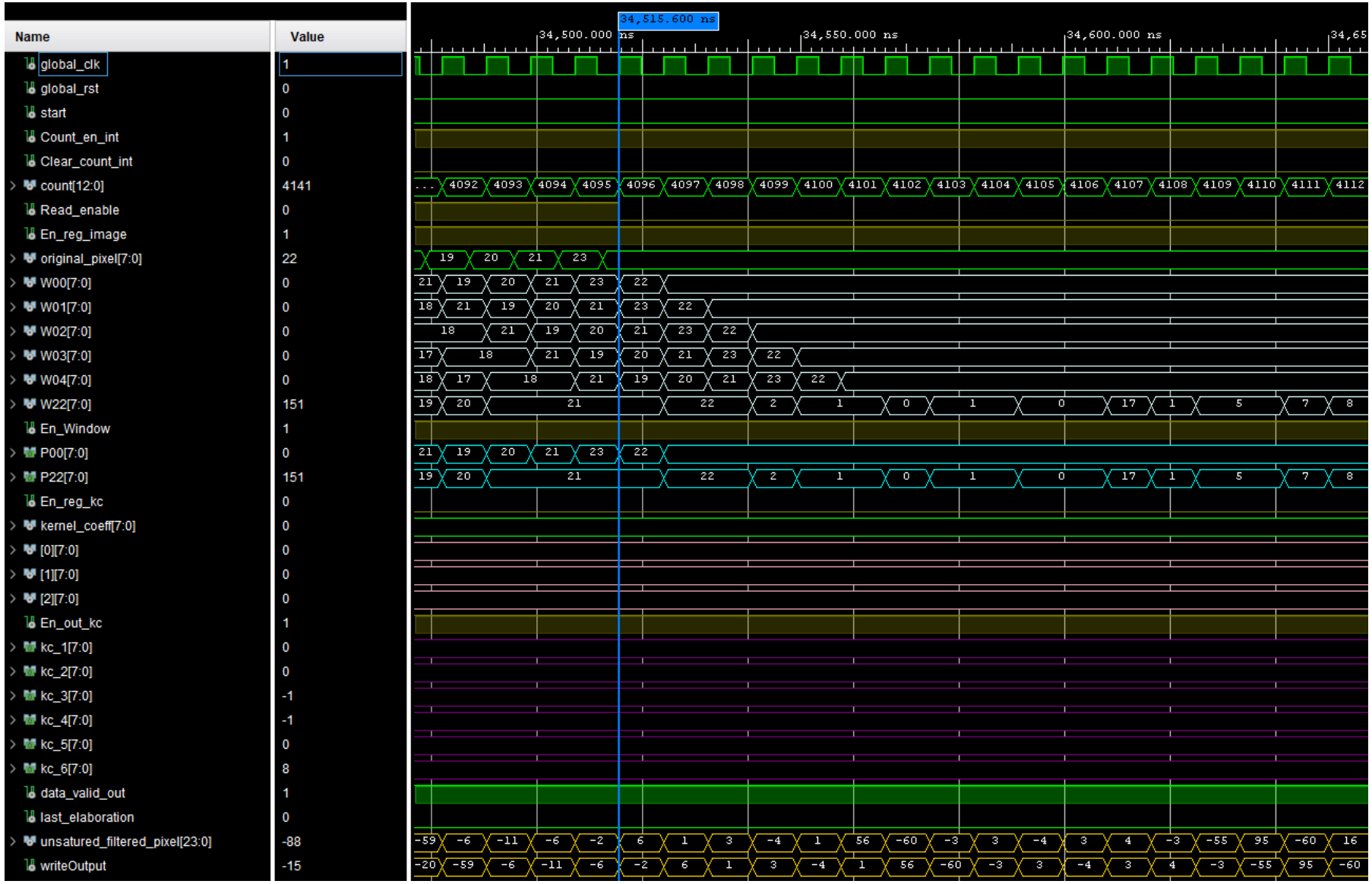
Simulazione Behavioral

S3/S4 – Inizio elaborazione pixels e generazione di risultati validi



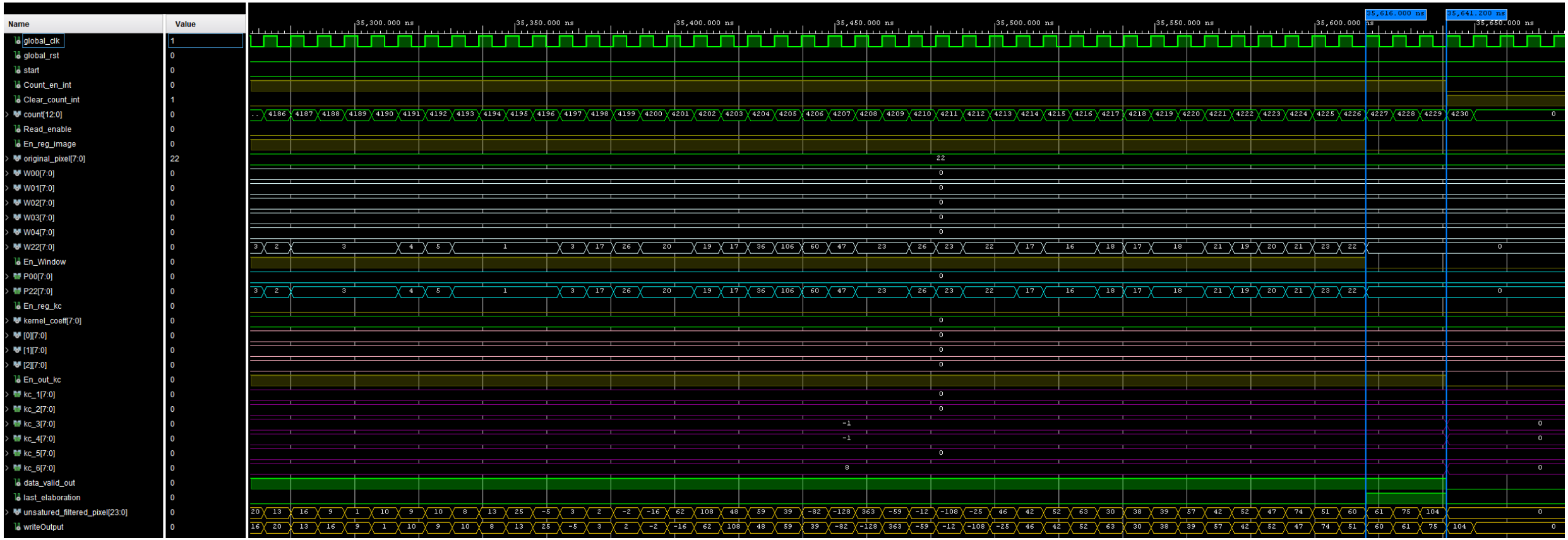
Simulazione Behavioral

S5 – Stop lettura pixels

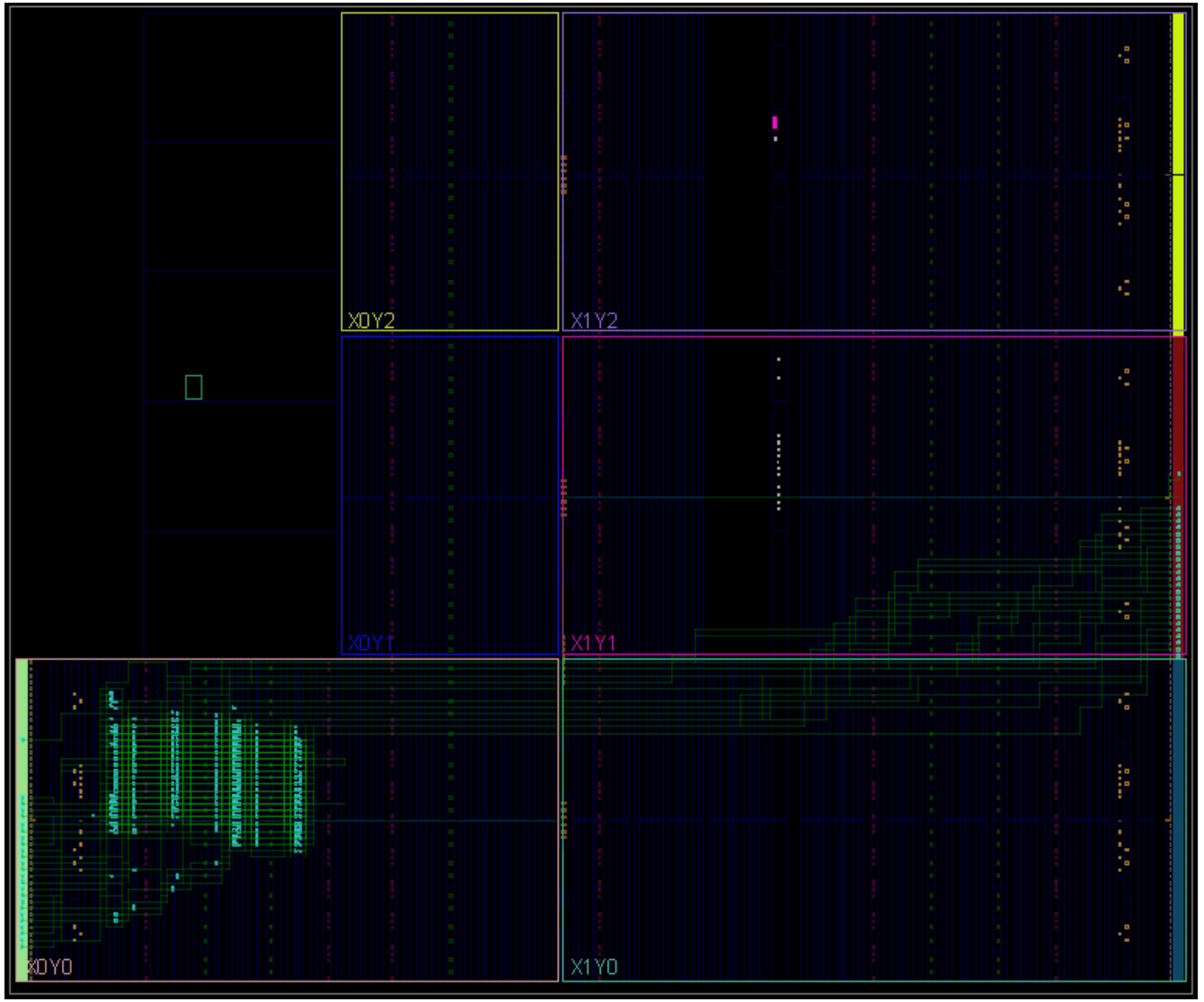


Simulazione Behavioral

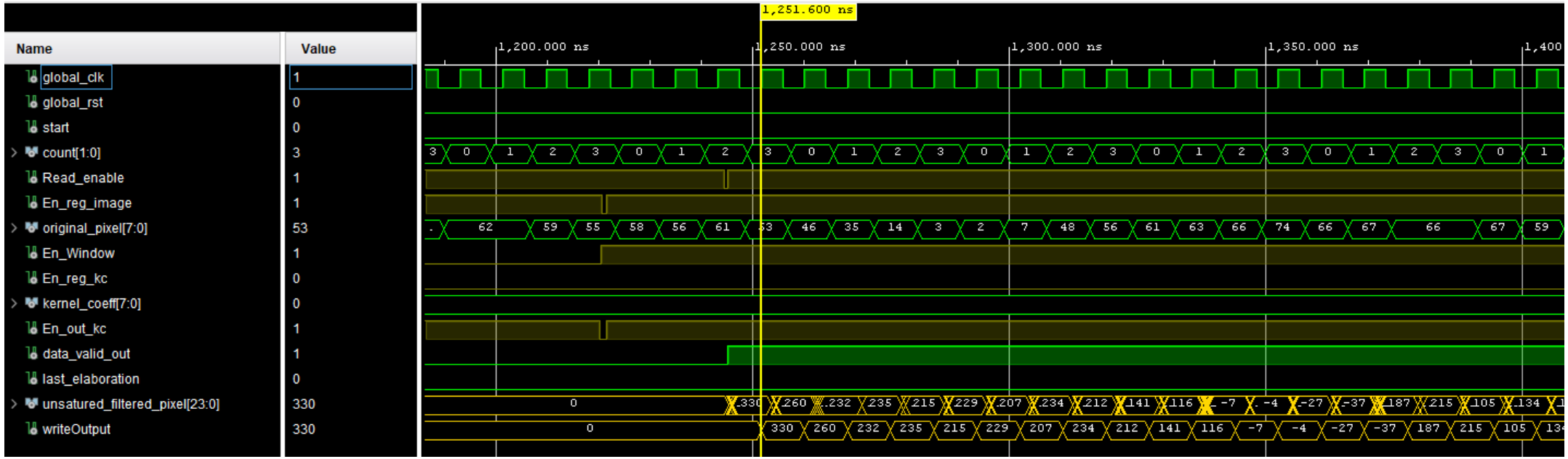
S6 – Elaborazione ultimo pixel

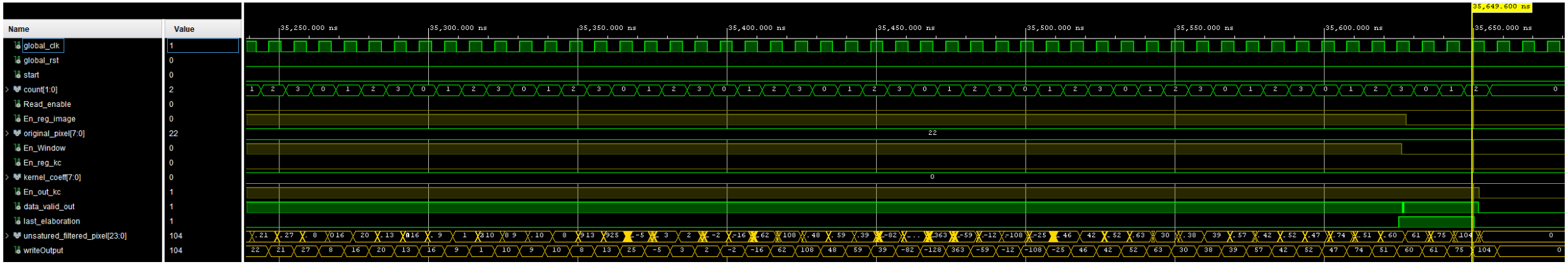


Implementazione su FPGA



Simulazione post-implementation





Analisi - Report timing

Il constraint di clock minimo per il quale è garantito il corretto funzionamento del circuito è pari a $8{,}4\ ns$.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0,169 ns	Worst Hold Slack (WHS):	0,087 ns	Worst Pulse Width Slack (WPWS):	3,220 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:	1153	Total Number of Endpoints:	1153	Total Number of Endpoints:	681

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             0.169ns  (required time - arrival time)
  Source:                 Controller_module/Finite_state_machine/FSM_onehot_present_state_reg[3]/C
                            (rising edge-triggered cell FDRE clocked by global_clk  {rise@0.000ns fall@4.200ns period=8.400ns})
  Destination:            Convolution_module/Second_pipeline_module/Out_regl_reg[18]/D
                            (rising edge-triggered cell FDRE clocked by global_clk  {rise@0.000ns fall@4.200ns period=8.400ns})
  Path Group:             global_clk
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            8.400ns  (global_clk rise@8.400ns - global_clk rise@0.000ns)
  Data Path Delay:        8.222ns  (logic 1.820ns (22.135%)  route 6.402ns (77.865%))
  Logic Levels:           11  (LUT3=3 LUT6=8)
  Clock Path Skew:        -0.050ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    4.615ns = ( 13.015 - 8.400 )
    Source Clock Delay      (SCD):    5.124ns
    Clock Pessimism Removal (CPR):    0.458ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

Analisi - Report performance

- Frequenza massima di funzionamento:

$$f_{max} = \frac{1}{\tau_{constr}} = \frac{1}{8.4\ ns} = 119.048\ MHz$$

- Latenza, ovvero, tempo necessario per generare un risultato valido dal momento in cui ha inizio la lettura dei dati:

$$Latency = 134\ cicli\ di\ clock$$

131 cicli necessari a generare la finestra utile da processare

3 cicli dovuti agli stadi di pipeline

- Throughput, ovvero, risultati validi generati ad ogni ciclo di clock una volta superata la latenza iniziale:

$$Throughput = 1$$

Analisi - Report utilization

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	1327	0	0	53200	2.49
LUT as Logic	1239	0	0	53200	2.33
LUT as Memory	88	0	0	17400	0.51
LUT as Distributed RAM	0	0			
LUT as Shift Register	88	0			
Slice Registers	592	0	0	106400	0.56
Register as Flip Flop	592	0	0	106400	0.56
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

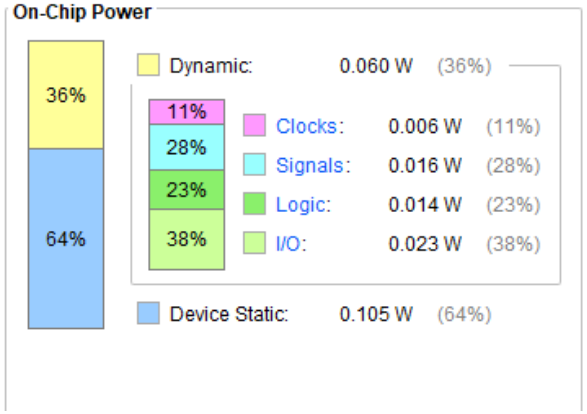
Name	Slice LUTs (53200)	Slice Registers (106400)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)
Spatial_Filtering_Circuit	1327	592	387	1239	88
Controller_module (ControllerModule)	71	20	47	71	0
Counter (CounterSync_generic)	21	13	7	21	0
Finite_state_machine (FSM)	50	7	41	50	0
Convolution_module (ConvolutionModule)	569	197	257	569	0
First_pipeline_module (Pipeline_reg_ADDERtoMU	438	60	205	438	0
Second_pipeline_module (Pipeline_reg_MULTtoA	131	113	110	131	0
Third_pipeline_reg (Final_pipeline_reg)	0	24	14	0	0
Image_Cache_memory (ImageCacheMemory)	299	291	128	235	64
Fifth_RowWindow (Shift_reg)	27	40	31	27	0
First_RowWindow (Shift_reg_0)	84	40	49	84	0
Fourth_RowWindow (Shift_reg_1)	11	40	30	11	0
Row_buffer_1 (Row_buffer)	20	67	28	4	16
Row_buffer_2 (Row_buffer_2)	20	8	9	4	16
Row_buffer_3 (Row_buffer_3)	20	8	11	4	16
Row_buffer_4 (Row_buffer_4)	20	8	12	4	16
Second_RowWindow (Shift_reg_5)	31	40	32	31	0
Third_RowWindow (Shift_reg_6)	66	40	43	66	0
Isotropic_Kernel_memory (IsotropicKernelMemory)	391	84	176	367	24

Analisi - Report power

Generazione del file .$saif$ per tener conto dell'attività di switching effettiva dei segnali e ottenere un alto livello di confidenza.

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

Total On-Chip Power:	**0.165 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,9°C**
Thermal Margin:	58,1°C (4,9 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	High



	Dynamic power dissipation [W]	Total dynamic power [mW]
Clock	0.00628282	59.561
Signals	0.01645911	
logic	0.01357104	
I/O	0.02324796	

- Energia media per singola operazione:

$$E_{dyn} = P_{dyn} \cdot \tau_{constr} = 59.561\ [mW] \cdot 8.4[ns] = 500.312\ [pJ]$$

Risultati

Confronto dei risultati hardware (VHDL) e software (MATLAB).

Risultati hardware (no saturazione)

Original image

Laplaciano-Gaussiano1 VHDL

Laplaciano-Gaussiano2 VHDL

Laplaciano1 VHDL

Laplaciano2 VHDL

Laplaciano3 VHDL

Risultati software (no saturazione)

Original image

Laplaciano-Gaussiano1 MATLAB

Laplaciano-Gaussiano2 MATLAB

Laplaciano1 MATLAB

Laplaciano2 MATLAB

Laplaciano3 MATLAB

Risultati hardware (saturazione)

Original image

Laplaciano-Gaussiano1 VHDL

Laplaciano-Gaussiano2 VHDL

Laplaciano1 VHDL

Laplaciano2 VHDL

Laplaciano3 VHDL

Risultati software (saturazione)

Original image

Laplaciano-Gaussiano1 MATLAB

Laplaciano-Gaussiano2 MATLAB

Laplaciano1 MATLAB

Laplaciano2 MATLAB

Laplaciano3 MATLAB

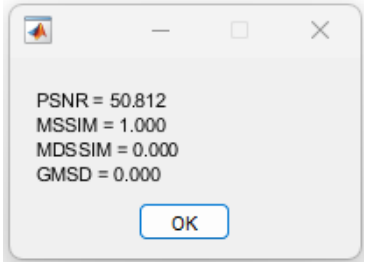
Risultati

Confronto effettuato sulla base dei parametri $PSNR, MSSIM, GMSD, MDSSIM$.

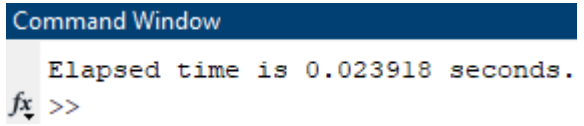
Forniscono una previsione abbastanza accurata sulla qualità di un'immagine rispetto a quella di riferimento attraverso un'analisi dell'errore pixel per pixel.

Laplaciano-Gaussiano1 MATLAB

Laplaciano-Gaussiano1 VHDL



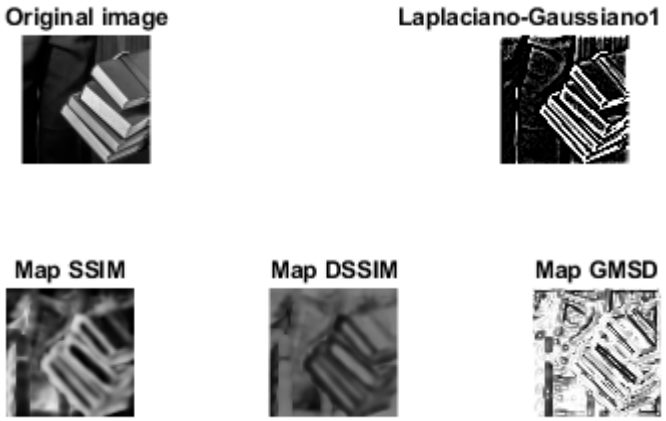
$Time\_elaboration\_hardware \cong 0{,}0357ms$

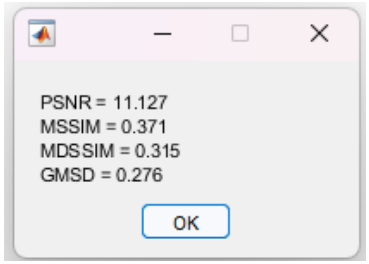


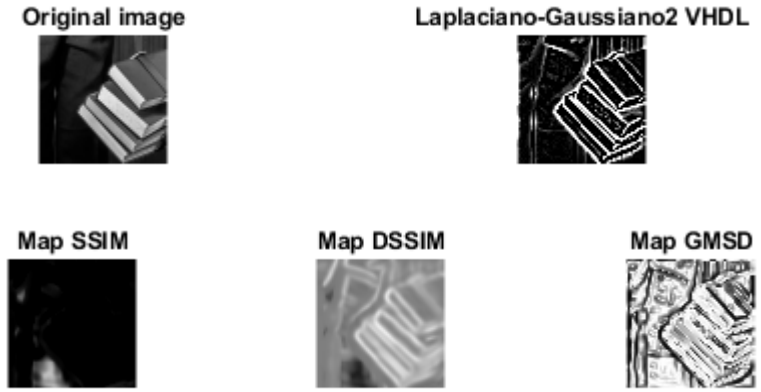
$Time\_elaboration\_software \cong 23{,}918ms$

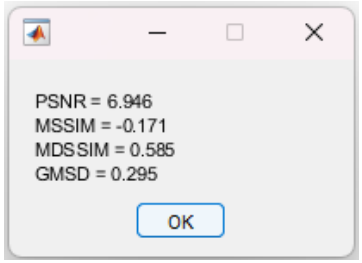
Risultati

Confronto tra immagine originale e immagine filtrata in termini di qualità.









Un'enfatizzazione pronunciata dei bordi comporta una degradazione della qualità dell'immagine e quindi un aumento della distorsione.

Conclusioni

L’implementazione di un circuito di filtraggio spaziale su FPGA può rappresentare un’ottima alternativa alle soluzioni tradizionali basate esclusivamente su software. In generale, in un’applicazione reale, l’approccio congiunto tra hardware e software, sfruttando la tecnologia FPGA, consente di massimizzare le prestazioni, garantendo al contempo la possibilità di personalizzare il circuito per soddisfare le esigenze specifiche dell’applicazione. Questo approccio rappresenta, dunque, una soluzione efficace per applicazioni di image processing che richiedono alta velocità e precisione.

Miglioramenti futuri

Esistono diverse aree in cui il progetto potrebbe essere ulteriormente migliorato:

1. Parallelizzazione delle operazioni di convoluzione;
2. Impiegare kernel di dimensioni maggiori;
3. Modificare l'architettura per supportare immagini a colori o con risoluzioni superiori;
4. Sostituire sommatori RCA con sommatori Carry-Select o Parallel-Prefix;
5. Scalabilità della frequenza;