



*Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni*

Implementazione sull'Architettura Parallela Simpil del Processo di Generazione dell'Immagine in Fotocamere Digitali

A. Gentile – S. Vitabile – F. Sorbello

RT-ICAR-PA-03-01

gennaio 2003



Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)

– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: www.icar.cnr.it

– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: www.na.icar.cnr.it

– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: www.pa.icar.cnr.it



*Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni*

Implementazione sull'Architettura Parallela Simpil del Processo di Generazione dell'Immagine in Fotocamere Digitali

A. Gentile² – S. Vitabile¹ – F. Sorbello²

**Rapporto Tecnico N.1:
RT-ICAR-PA-03-01**

**Data:
gennaio 2003**

¹ Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sezione di Palermo

² Università degli Studi di Palermo - Dipartimento di Ingegneria Informatica

I rapporti tecnici dell'ICAR-CNR sono pubblicati dall'Istituto di Calcolo e Reti ad Alte Prestazioni del Consiglio Nazionale delle Ricerche. Tali rapporti, approntati sotto l'esclusiva responsabilità scientifica degli autori, descrivono attività di ricerca del personale e dei collaboratori dell'ICAR, in alcuni casi in un formato preliminare prima della pubblicazione definitiva in altra sede.

***Implementazione sull'Architettura Parallela Semplice
del Processo di Generazione dell'Immagine
in Fotocamere Digitali***

A. Gentile – S. Vitabile – F. Sorbello

Rapporto tecnico I.C.A.R. 1/03

Indice

1. Introduzione	3
2. Descrizione di un tipico sistema DSC.....	5
2.1 - <i>Pipeline</i> di elaborazione	9
3. L'Architettura Simpil.....	14
3.1 - Architettura dei singoli PE e dell'ACU	16
3.2 – Efficienza dell'architettura Simpil	20
3.3 - Il Simulatore SIMPil.....	23
3.3.1 Formato delle istruzioni SIMPil.....	25
3.3.2 Set di istruzioni vettoriali (SIMPil Processor ISA).....	28
3.3.3 - Set di istruzioni scalari (SIMPil Controller ISA)	33
4. Implementazione del Processo di Elaborazione dell'Immagine Digitale.....	40
4.1 - Sequenza di Elaborazione.....	41
4.2 - Analisi Funzionale	45
4.2.1 - Analisi con PPE variabile	46
4.2.2 - Analisi con PPE fissato.....	49
4.3 - Analisi delle prestazioni del Sistema.....	53
4.3.1 - Analisi con PPE variabile	53
4.3.2 - Analisi con PPE fissato.....	56
5. Valutazioni e Conclusioni.....	60
Bibliografia	63

1. Introduzione

Entertainment and Communication sono diventate, oramai, le parole d'ordine dello sviluppo tecnologico. Infatti, sono questi i campi, più di ogni altro, che spingono nello sviluppo di tecnologie sempre più sofisticate e potenti.

Le applicazioni multimediali ad elevate prestazioni (ambienti di gioco realistici in 3D, effetti speciali cinematografici, ecc.) e l'introduzione della terza generazione delle comunicazioni wireless, sono solo alcuni degli aspetti innovativi che hanno già fatto il loro ingresso nel circuito della grande produzione e che si apprestano ad invadere il campo domestico. Attualmente le più grandi aziende del settore stanno lavorando alla terza generazione dei sistemi di telefonia cellulare (3G wireless o 3G UMTS standard). Con l'introduzione di questi nuovi standard della comunicazione, l'utente avrà a disposizione l'accesso a servizi basati sull'elaborazione ad alta velocità di grandi quantità di dati, come l'accesso wireless a Internet o la trasmissione wireless di immagini video fisse ed in movimento. Ciò impone l'adozione di soluzioni architetturali sempre più avanzate. E poiché i requisiti legati alla dissipazione di potenza precludono la possibilità di incrementare la frequenza di clock, per raggiungere le prestazioni richieste in termini di GOPS (Miliardi d'Operazioni per Secondo), una soluzione è quella di utilizzare architetture parallele, che sfruttino l'elevato parallelismo intrinseco nei dati delle applicazioni.

A tal proposito, una nuova classe d'architetture si sta diffondendo: quella dei cosiddetti PMMS (Portable MultiMedia Supercomputers). Questi permettono di combinare le caratteristiche di elevate performance computazionali, necessarie alle applicazioni multimediali, con una grande efficienza energetica, necessaria ai dispositivi portatili.

Tra i PMMS, l'architettura SIMPil (SIMD Pixel Processor, sviluppato dalla PICA Research Group della Georgia Tech.) soddisfa tali requisiti, specie nelle applicazioni video e di elaborazione di immagini digitali che risultano terreno fertile per sfruttare a pieno le sue potenzialità.

Questo lavoro ha lo scopo di mostrare come sia indicato l'utilizzo di un dispositivo multiprocessor per l'elaborazione di immagini nell'ambito delle *Digital Still Camera* fornendo una valida analisi degli algoritmi implementati che fanno parte di un processo di elaborazione ben più ampio che inizia dal momento in cui le immagini vengono catturate attraverso i sensori, al momento in cui vengono compresse e memorizzate su appositi supporti.

Per esprimere un giudizio sulla qualità degli algoritmi presentati, si confronteranno i risultati ottenuti, eseguendo un test sull'architettura SIMPil, con quelli derivati dal processo di elaborazione di una macchina fotografica digitale già disponibile sul mercato: il TMS320C549 della Texas Instruments.

Il presente lavoro è organizzato nel modo seguente. Nella sezione 2 si discuterà della tipica architettura di una Digital Still Camera e sulle operazioni che compongono l'intero processo di elaborazione. Nella sezione 3 sarà introdotta l'architettura SIMPil, confrontando le caratteristiche dell'architettura con gli ambienti paralleli già presenti nel settore. Verrà anche introdotto l'ambiente di simulazione per poter chiarire le potenzialità di sviluppo di programmi che sfruttano la piattaforma SIMPil. La sezione 4 è dedicata all'analisi dell'intero processo di elaborazione. Si darà spazio a considerazioni sia dal punto di vista funzionale che dal punto di vista delle risorse di sistema impegnate, rimandando il confronto con il dispositivo della Texas Instruments (TMS320C549) nell'ultima sezione.

2. Descrizione di un tipico sistema DSC

Negli ultimi anni si è avuto un enorme sviluppo d'applicazioni e dispositivi per l'elaborazione dei segnali video in movimento e delle immagini digitali fisse.

L'enorme sviluppo di questo settore è testimoniato dal notevole aumento delle vendite di macchine fotografiche digitali rispetto alle macchine fotografiche tradizionali. In Figura 2.1 è rappresentato il grafico delle vendite nel mondo riguardo agli ultimi anni, insieme con una stima per i prossimi anni, da cui si evince che entro il 2005 si avranno più macchine digitali che tradizionali [1].

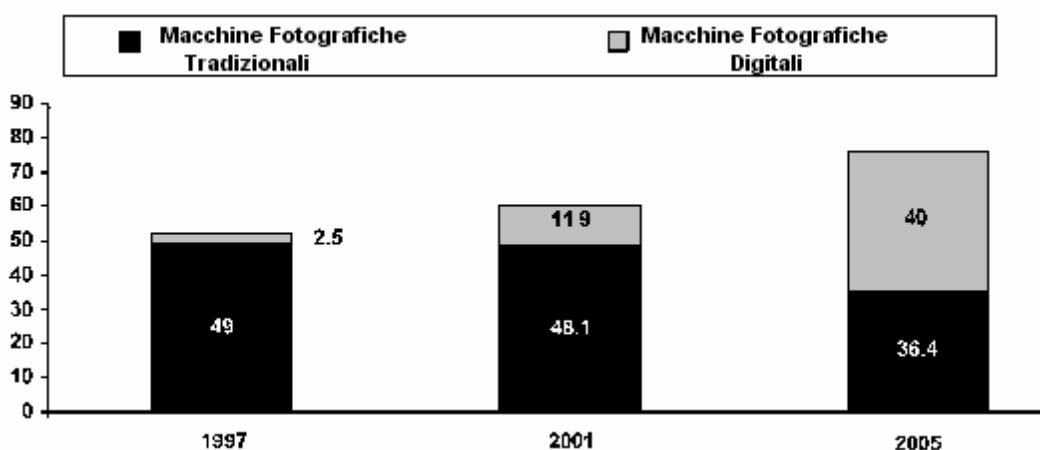


Figura 2.1 - Macchine fotografiche vendute nel mondo (prev. in milioni.d'unità) ¹

Questi dispositivi, detti DSC (Digital Still Camera), offrono diversi vantaggi rispetto alle tradizionali macchine fotografiche in termini d'immagazzinamento dati, manipolazione e trasmissione.

La rappresentazione digitale delle immagini catturate permette, agli utenti, di incorporarle e trasferirle in diversi dispositivi elettronici per il trattamento d'immagini quali TV o PC. Inoltre, si ha la possibilità di rivederle istantaneamente ed, eventualmente, riacquisirle o modificarle dopo l'acquisizione.

¹ Stime fornite da: Nikkei Market Access, IDC, Lyra, Data Quest, TI.

Le fotocamere digitali utilizzano un sensore elettronico per acquisire le variazioni spaziali dell'intensità di luce. L'*Image Processing* consiste nell'applicare appositi algoritmi per ricostruire la figura a colori dalle informazioni ricevute.

Esistono due tecnologie fondamentali per la fabbricazione dei sensori ottici: CCD (Charge Coupled Device) e CMOS (Complementary Metal Oxide Semiconductor). Sia i sensori CCD che CMOS si basano sullo stesso principio di funzionamento: convertire la luce in elettricità.

I sensori CCD [3] sono un insieme di minuscoli diodi fotosensibili che convertono fotoni (elementi della luce) in elettroni (elementi dell'elettricità). Questi diodi sono chiamati *photosite*: più è forte la luce più esso viene eccitato e più alta è la tensione accumulata. Le tensioni registrate da ogni *photosite*, sono trasportate in un punto preciso del chip (di solito in un angolo del dispositivo) dove vengono passati, uno dopo l'altro, ad un convertitore analogico/digitale che provvederà a trasformarle in un formato digitale.

Nei sensori CMOS [3] ci sono alcuni transistor, per ogni pixel, che amplificano e trasferiscono le cariche registrate usando i più tradizionali conduttori metallici. L'approccio tramite la tecnologia CMOS è più flessibile perché, in questo modo, ogni pixel può essere letto individualmente, mentre, la tecnologia CCD prevede uno speciale processo di fabbricazione dei sensori che rende possibile il trasferimento delle cariche, all'interno del chip, senza produrre distorsioni. I chip CMOS, d'altro canto, sono fabbricati con procedimenti standard, gli stessi utilizzati per fabbricare i più comuni chip. A causa di questa differenza, i dispositivi CCD hanno una più alta qualità e sono meno suscettibili ai disturbi dei segnali elettrici prodotti, mentre i dispositivi CMOS sono più vulnerabili ai disturbi elettrici; ciò perché vicino ad ogni pixel, su un chip CMOS, sono dislocati dei transistor che ne riducono la sensibilità all'intensità luminosa, infatti, molti fotoni, invece di colpire il *photosite*, colpiscono i transistor.

I dispositivi CMOS hanno il vantaggio di consumare poca potenza elettrica, mentre, un dispositivo CCD può arrivare a consumare 100 volte di più e questo influenza molto le decisioni progettuali per la costruzione di apparecchiature portatili quali le fotocamere digitali.

In definitiva i sensori CCD sono utilizzati nella fabbricazione di macchine fotografiche di fascia alta dove l'elevato costo è giustificato dalla maggiore qualità delle immagini ottenute. Di contro i sensori CMOS sono impiegati nella costruzione di macchine di fascia più bassa dove il minor costo compensa la qualità più bassa delle immagini. Il colore, così come viene percepito dall'occhio umano, può essere decomposto in tre colori principali: verde, rosso e blu. In questo caso si parla di rappresentazione RGB (Red, Green, Blue) del colore [4].

Gli attuali sensori non hanno alcuna conoscenza cromatica della realtà. Reagiscono ai fotoni liberando elettroni senza essere, di fatti, sensibili alle frequenze in gioco, ovvero al colore della luce percepita. Dunque un sensore CCD (o CMOS), almeno allo stato attuale, è un dispositivo rigidamente monocromatico (“vede” a livelli di grigio). Per quanto detto, nella sua accezione più semplice, l'acquisizione a colori si riduce ad effettuare tre singole esposizioni, antepoendo, all'obiettivo di ripresa, un filtro verde, un filtro rosso e uno blu. Inutile sottolineare che un dispositivo di ripresa organizzato in questo modo crea, di certo, non pochi problemi, primo tra tutti che fotocamera e soggetto ripreso, durante le tre esposizioni, devono rimanere assolutamente immobili per non provocare evidenti e antiestetiche sbavature di colore nei contorni dei dettagli. Per lo stesso motivo sarebbe, poi, impossibile realizzare telecamere a colori con sensore CCD singolo.

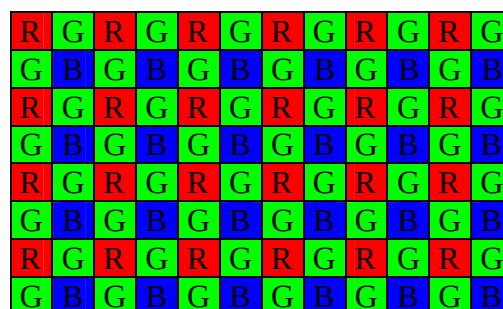


Figura 2.2 - *Color Filtered Array*

L'ostacolo delle tre riprese successive può essere aggirato utilizzando tre diversi sensori: uno per ogni colore base. Sfortunatamente questa soluzione è alquanto costosa e quindi, nella stragrande maggioranza di fotocamere digitali, si è pensato di adottare una soluzione diversa: si utilizza un

unico sensore CCD sul quale viene posta una fitta rete di microfiltri RGB, in modo che ogni pixel dia un'informazione cromatica diversa da quelli adiacenti.

Questa particolare disposizione è mostrata in Figura 2.2 e prende il nome di *Color Filtered Array* (CFA). L'immagine che se ne ricava, risulta essere espressa in un formato detto *Bayer Pattern*.

Si noti, in Figura 2.2, come siano presenti più elementi filtrati in verde rispetto a quelli filtrati in rosso e in blu. I primi sono esattamente il doppio degli altri. Ciò non è casuale perché la regione del verde è quella di maggiore sensibilità per il nostro apparato visivo ed è proprio in quell'intervallo dello spettro visibile che riusciamo a riconoscere un numero maggiore di dettagli e sfumature.

Descriviamo, brevemente, adesso, i blocchi funzionali più importanti di un tipico sistema DSC [2]. In Figura 2.3 ne viene riportato uno schema. Il *Driver* e il *Timing Generator* sono preposti alla generazione del segnale di clock del CCD. Il *Correlated Double Sampling* e l'*Automatic Gain Control* sono usati per ottenere una buona qualità del segnale di immagine proveniente dal sensore CCD, successivamente questo dato viene digitalizzato nel blocco *A/D Converter* e quindi trasferito al *DSC Engine*. Tale blocco è quello più importante ed è anche quello all'interno del quale si opererà per l'implementazione dell'algoritmo in SIMPil. In questo blocco, infatti, vengono eseguite tutte le operazioni di *Preprocessing*, *Image Processing* e di *Image Compression*.

Nella maggior parte dei sistemi DSC, gli utenti hanno la possibilità di vedere l'immagine che sarà acquisita in anteprima su un display LCD. L'immagine acquisita verrà quindi immagazzinata su una memoria flash e potrà essere rivista successivamente. Molti sistemi DSC, inoltre, forniscono un segnale video NTSC/PAL, che permette di vedere le immagini catturate su un monitor TV. Gli attuali sistemi DSC consentono anche di collegarsi a PC o a stampanti tramite porte RS232 o USB. Versioni ancora più versatili sono attese per il futuro; queste permetteranno di associare del testo o del parlato alle immagini, includeranno, inoltre, un modem e un'interfaccia TCP/IP, tale da permettere direttamente una connessione ad internet.

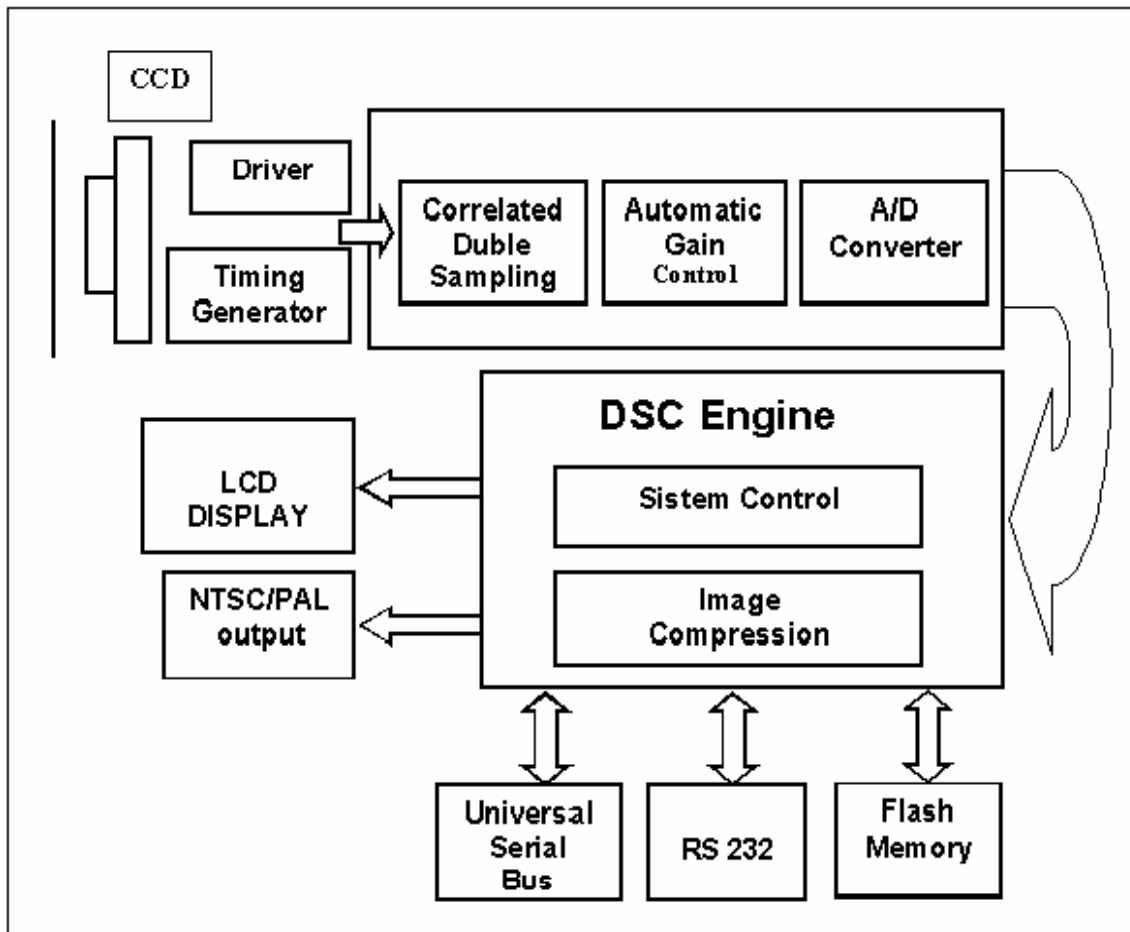


Figura 2.3 - Tipico sistema DSC

2.1 - Pipeline di elaborazione

Un sistema DSC deve eseguire diversi passi d'elaborazione prima che un'immagine di elevata qualità possa essere immagazzinata. Il primo di questi passi è l'acquisizione dell'immagine attraverso un sensore CCD o in alcuni casi un sensore CMOS. L'immagine catturata è mascherata, con dei filtri di colore, al fine di fornire un'immagine a colori. L'immagine così ottenuta è identificata come *Color Filtered Array* (CFA). Il modello di mascheramento e di filtraggio dei colori primari può variare secondo le case costruttrici. Nei sistemi DSC, il modello di CFA tipicamente utilizzato è il cosiddetto *CFA Bayer pattern*.

L'immagine in formato CFA *Bayer pattern*, come detto, deve subire tutta una serie di elaborazioni prima che questa sia portata ad un formato compresso ed immagazzinata. L'insieme di questi passi di elaborazione, prende il nome di *Image Pipeline* [5].

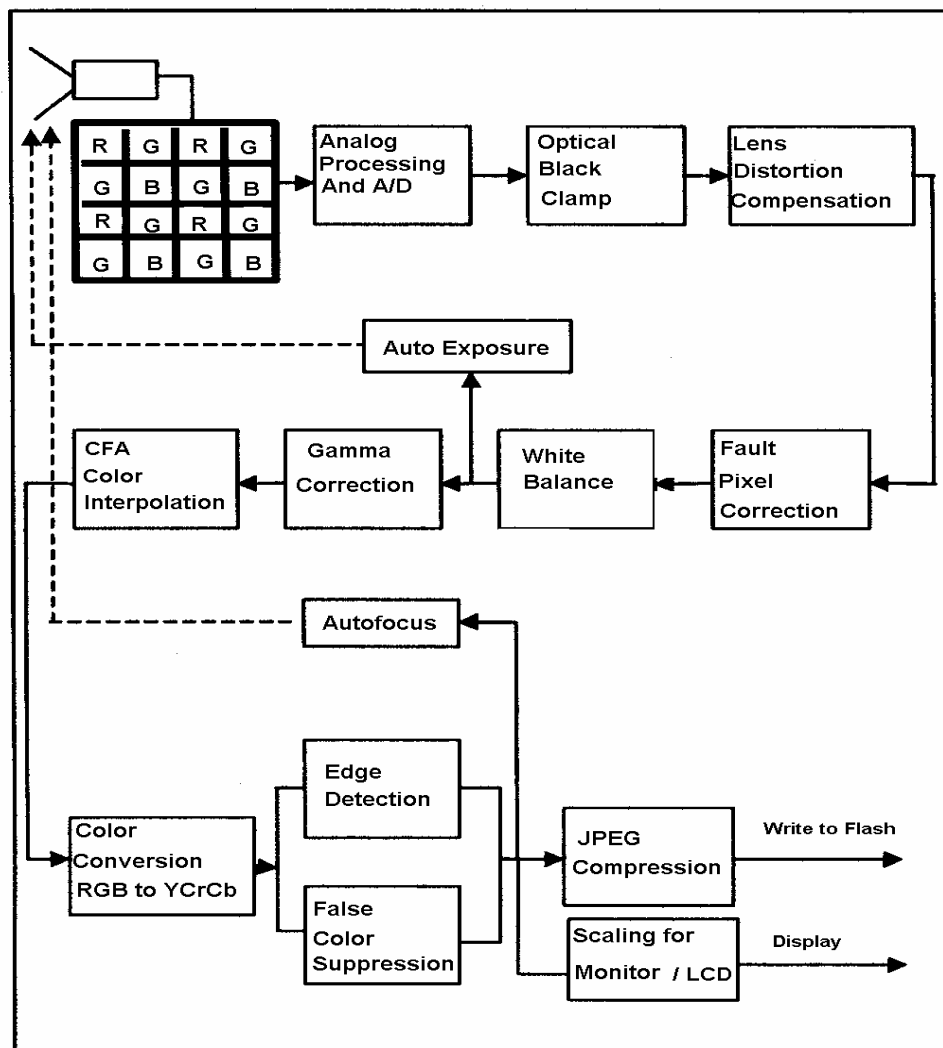


Figura 2.4 - Tipica “pipeline” d'elaborazione d'immagine digitale.

In Figura 2.4 è illustrata una tipica *Image Pipeline* per un sistema DSC, dove ogni blocco rappresenta uno dei passi necessari all'elaborazione di un'immagine digitale.

Tutte queste elaborazioni sono state eseguite su un sistema d'elaborazione di segnali digitali, *Digital Signal Processing* (DSP), della famiglia della Texas Instruments TMS320C54x.

Le prestazioni di tale dispositivo saranno prese come riferimento per analizzare l'algoritmo di elaborazione implementato su SIMPil. Uno dei vantaggi di questo dispositivo è il fatto di usare un Processore di Segnali Digitali (DSP) specifico per l'elaborazione e la compressione dell'immagine proveniente dal sensore. I passi che tale sistema deve svolgere sono i seguenti [2]:

- **Black Clamp:** tale processo serve ad ottimizzare il range dinamico dei valori dei pixel rilevati dal sensore CCD. Questo viene fatto correggendo i valori dei pixel neri, portando il loro valore a zero. Ciò viene fatto sottraendo un determinato valore di offset dal valore di ogni pixel, evitando però di ottenere valori negativi, attraverso un'operazione di Clamping/Clapping.
- **Lens Distorsion Compensation:** questa funzione serve a compensare la non linearità di luminosità, dovuta ad alcune imperfezioni delle lenti. Queste non linearità riducono la luminosità tra il centro dell'immagine ed i suoi bordi. La compensazione di tali effetti avviene aggiustando la luminosità d'ogni pixel in funzione della loro locazione spaziale.
- **Fault Pixel Interpolation:** a causa di difetti di fabbricazione, spesso alcuni pixel del sensore CCD, non rilevano alcun valore. Per tale motivo, il valore di questi pixel mancanti, viene ripristinato attraverso delle tecniche di interpolazione dei valori dei pixel circostanti.
- **White Balance:** serve a ripristinare il valore del bianco, ossia il valore (255,255,255), nello spazio dei colori RGB, per quelle parti d'immagini che originariamente erano bianche e che dopo l'acquisizione risultano alterate. Inoltre la variazione di sensibilità dei tre canali RGB provoca una rappresentazione non corretta dei toni di grigio o dei colori neutri. Per compensare questo sbilanciamento dei colori, viene effettuata una equalizzazione del guadagno dei canali rosso, verde e blu.
- **Gamma Correction:** dispositivi di visualizzazione delle immagini, quali monitor o stampanti, introducono una non-linearità nel mapping tra l'immagine generata dal sensore CCD e l'immagine visualizzata sul monitor o sulla stampante. La funzione Gamma Correction serve, appunto, a compensare queste differenze.

I passi fin qui descritti, sono considerati come fase di *Preprocessing* dell'immagine. Di seguito sono descritti i passi riguardanti la fase di *Color Processing* [2]:

- **CFA Interpolation:** Nell'acquisizione dell'immagine è utilizzato un CFA Bayer pattern. Pertanto, per ogni pixel dell'immagine rilevata, si ha a disposizione il valore di uno soltanto dei tre possibili colori RGB. In fase d'elaborazione si ha la necessità di conoscere il valore di tutti e tre i colori d'ogni pixel, pertanto, i due valori mancanti per ogni pixel, sono ricostruiti, per mezzo di tecniche d'interpolazione dei colori dei pixel vicini a quello da ricostruire.
- **Color Space Conversion:** Molti algoritmi di compressione delle immagini, quale lo JPEG, operano sullo spazio dei colori YCbCr. Una conversione dello spazio di colore è necessaria per passare dallo spazio RGB allo spazio YCbCr. Questa conversione rappresenta una trasformazione lineare dei valori R G B, nei valori Y Cr Cb, tramite una matrice di conversione opportuna.
- **Edge Detection:** l'operazione d'interpolazione dei colori, può provocare un'attenuazione (*Smoothing*) dei contorni delle immagini. L'Edge Detection permette di aumentare il contrasto locale dell'immagine in modo da arricchire i dettagli ed evidenziare i contorni degli oggetti (*Sharpening*). Per eseguire l'operazione di *Sharpening* si utilizza un *Edge Detector* per calcolare l'ampiezza del bordo nel canale Y di ogni pixel. Tale ampiezza è quindi scalata e sommata alla luminanza originale (Y) dell'immagine.
- **False Color Correction:** l'operazione di *Edge Detection*, come descritto, viene eseguita nel solo canale Y. Questo produce un disallineamento dei canali del colore, che si manifesta con la comparsa di colori inesistenti nell'immagine originale. Sopprimendo le due componenti cromatiche Cr e Cb nei contorni, si riducono questi effetti.
- **Autofocus:** in un sistema DSC è possibile eseguire la messa a fuoco delle lenti tramite un processo d'elaborazione d'immagine, che opera attraverso un ciclo retroazionato, con cui si

controlla la qualità di messa a fuoco dell'immagine ed iterativamente vengono spostate le lenti, fin quando non si raggiunge una posizione ottimale.

- **Auto Exposure:** a causa delle variazioni di luminosità nelle immagini, per ottenere una buona qualità dell'immagine complessiva, è necessario controllare il tempo d'esposizione del CCD. Questa operazione viene fatta rilevando la luminosità media dell'immagine, regolando di conseguenza il tempo di esposizione del CCD e/o del guadagno. Similmente all'Autofocus, il sistema esegue quest'operazione tramite un ciclo retroazionato.
- **Image Compression:** a causa del limitato spazio di memoria presente in una macchina fotografica, prima di essere immagazzinate, le immagini acquisite ed elaborate subiscono un successivo passo, quello della compressione. La maggior parte dei sistemi DSC utilizza come formato di compressione lo JPEG anche se si pensa che future DSC migreranno al più recente standard di compressione: lo JPEG 2000.

3. L'Architettura Simpil

SIMPil è un sistema SIMD (Single Instruction Multiple Data), sviluppato dal PICA Research Group del Georgia Institute of Technology di Atlanta, USA..

L'architettura generale di un sistema SIMD [14] è illustrata in Figura 3.1.

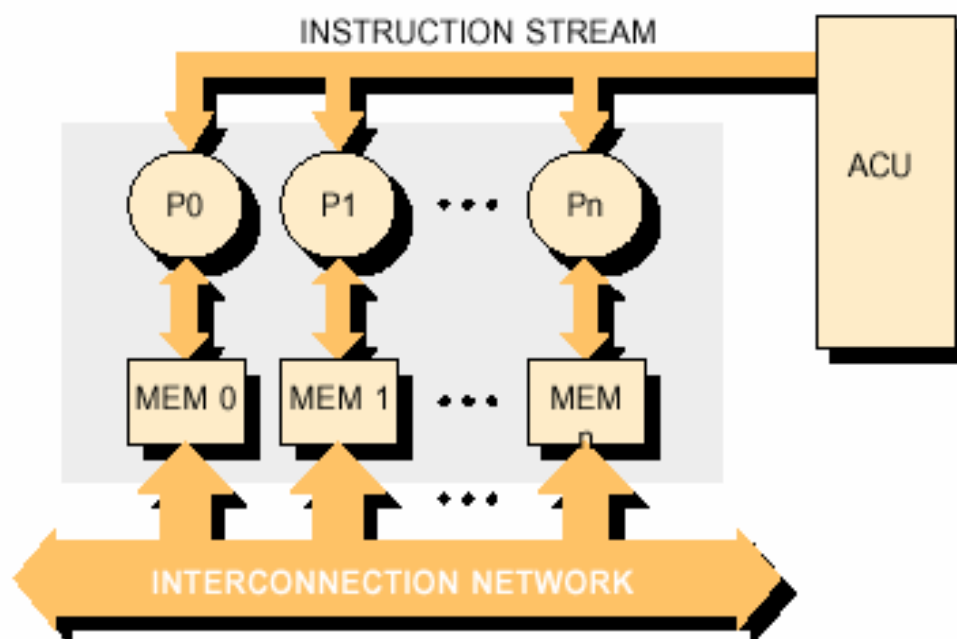


Figura 3.1 - Architettura SIMD

Il programma è memorizzato in un Array Control Unit (ACU), ed ogni istruzione è inviata a ciascun processore P_n del sistema (Single Instruction). Ogni nodo esegue le istruzioni ricevute sul dato presente nella memoria locale (Multiple Data). Esso può scambiare il dato con gli altri nodi attraverso la rete di interconnessione, la quale ha una struttura a mesh di tipo NEWS; questa può assumere anche una configurazione toroidale in modo da rendere la comunicazione tra i PE più potente e flessibile.

La caratteristica principale di un sistema SIMD è quindi l'elaborazione parallela dei dati in modo da permettere una minimizzazione sia dei tempi computazionali che dei movimenti dei dati e di conseguenza del consumo energetico. Inoltre un sistema SIMD consente di avere una singola

unità di controllo e permette di sfruttare le tecnologie di integrazione in quanto costituito da un ripetersi di nodi di identica struttura.

Il SIMPil è in grado di soddisfare i requisiti necessari al real-time nelle applicazioni multimediali grazie ad un maggiore equilibrio tra processing, immagazzinamento e comunicazione rispetto alle architetture DSP [15].

Come si può notare dalla Figura 3.2, SIMPil è un'architettura eterogenea che include: un *array SIMD* di processori, chiamati *Processor Element* (PE); un processore *ILP* che consente loro di comunicare e di scambiare dati; un'interfaccia *DEI* che trasferisce dati tra una parola indirizzata nella memoria di ogni PE e un array nella mappa di memoria *DSP*. Il nucleo *DSP* riguarda la parte sequenziale delle applicazioni.

L'*array SIMD* permette un elevato parallelismo, collegamenti corti ed una microarchitettura che consente un notevole aumento dell'efficienza energetica.

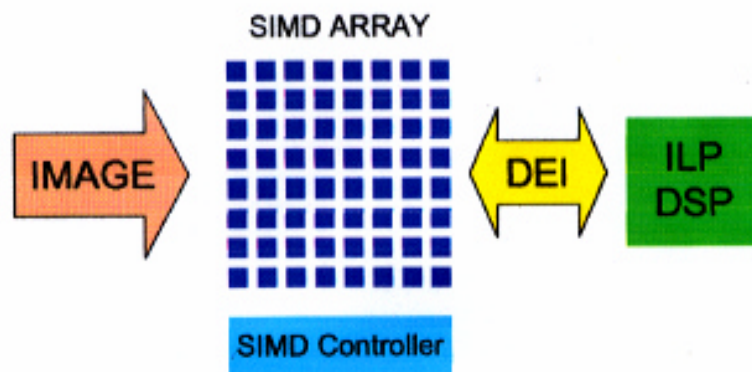


Figura 3.2 - Architettura SIMPil

Una caratteristica chiave dell'architettura SIMPil è il processing-in-place, ovvero i dati in ingresso nel SIMD array vengono processati localmente, riducendo, in questo modo, il movimento dei dati, minimizzando l'uso della memoria ed aumentando il throughput.

Per realizzare applicazioni multimediali è necessaria un'elevata larghezza di banda. Sono state studiate molte soluzioni per realizzarla, ma si è sempre giunti ad utilizzare come fonte di dati di ingresso la memoria con conseguente rallentamento dell'esecuzione del processo.

L'attuale architettura SIMPil utilizza un array di dispositivi di I/O per accedere direttamente ai processori, in questo modo è possibile realizzare il parallelismo direttamente sul flusso dei dati piuttosto che sui dati contenuti in memoria.

3.1 - Architettura dei singoli PE e dell'ACU

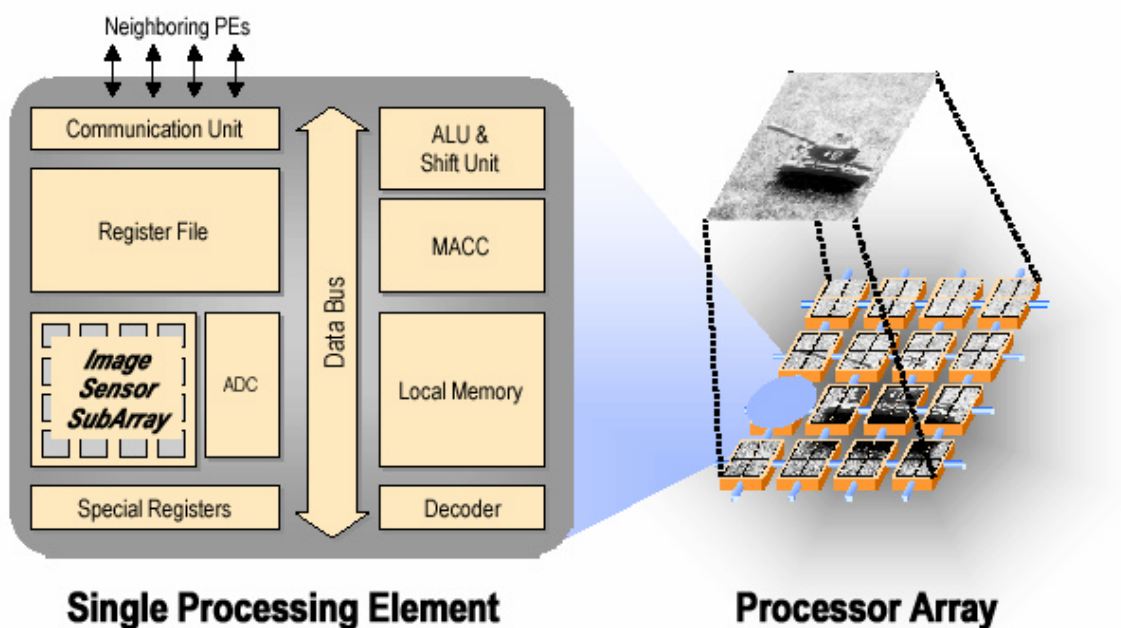


Figura 3.3 - Matrice di PE SIMPil e diagramma a blocchi di un PE

In Figura 3.3 è illustrata la matrice di processori SIMPil con il diagramma a blocchi di un singolo PE [14] [16]. Questo ultimo ha un *data path* di tipo RISC che possiede unità specializzate per le operazioni di tipo SIMD e meccanismi specifici per le stringhe di dati dell'area di I/O.

Di seguito sono elencate le unità funzionali presenti in ogni PE.

- **ALU e Shift Unit:** unità logico aritmetica e di shift.
- **Multiply-Accumulator Unit:** moltiplicatore e accumulatore a doppia precisione (32 bit).
- **Local Memory:** memoria con parole a 16 bit (fino ad un massimo di 32K parole).
- **Communication Unit:** unità che consente al PE di comunicare con quelli confinanti.
- **Serial I/O unit:** è l'unità seriale di I/O.
- **Sleep unit:** unità preposta al controllo dell'attività del PE.
- **Decoder:** decodificatore per le istruzioni da eseguire.
- **Special registers:** sono dei registri speciali.
- **ADC:** convertitore analogico-digitale ad 8 bit, per digitalizzare i segnali, ottenuti dalla matrice di rivelatori ottici, relativi all'intensità luminosa dei pixel.
- **Pixel registers:** registri nei quali vengono caricati i valori digitalizzati delle intensità luminose dei pixel.
- **Register file:** registri a 16 bit in numero pari a 16.
- **Data bus:** bus dei dati.

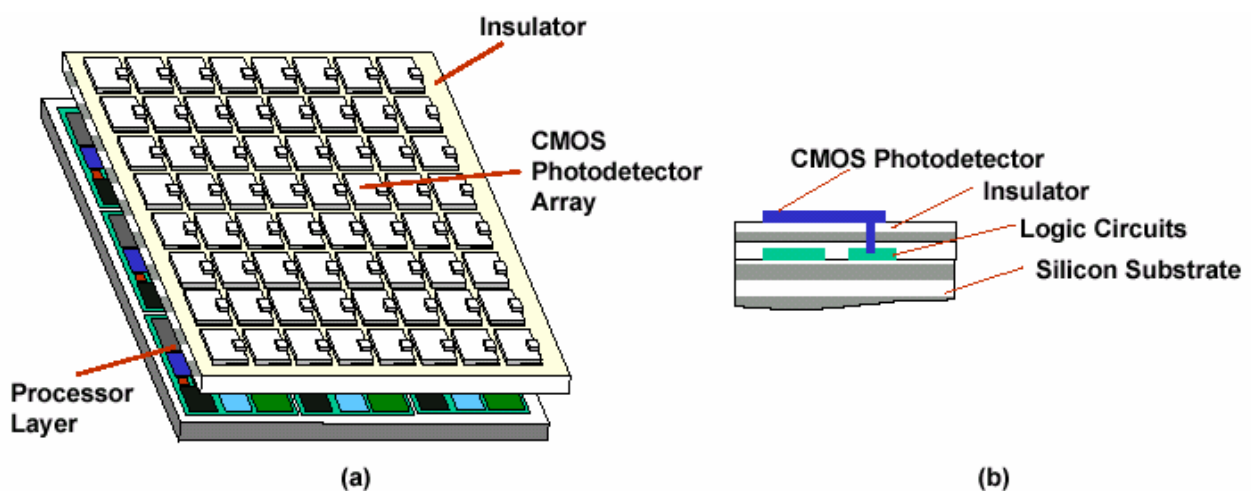


Figura 3.4 - (a) Una porzione del detector array dedicata a un singolo microprocessore; (b) Sensore di immagine multi-livello VLSI CMOS.

Ogni singolo PE può indirizzare una submatrice di rivelatori ottici (fino ad un massimo di 32K rivelatori). Il numero di rivelatori ottici indirizzati da un singolo PE indica il PPE ratio (Pixel per Processing Element ratio).

Una versione della matrice I/O dell'architettura SIMPil è realizzata impegnando un array di sensori dell'immagine CMOS come si percepisce dalla Figura 3.4.

Ciascuno nodo di SIMPil si connette ad un piccolo array di rivelatori di immagine che sono un sottoinsieme del vettore 2-D di pixel del piano focale [15].

Ciascun microprocessore incorpora un convertitore analogico-digitale per trasformare le intensità analogiche, incidenti sui rivelatori, in valori digitali.

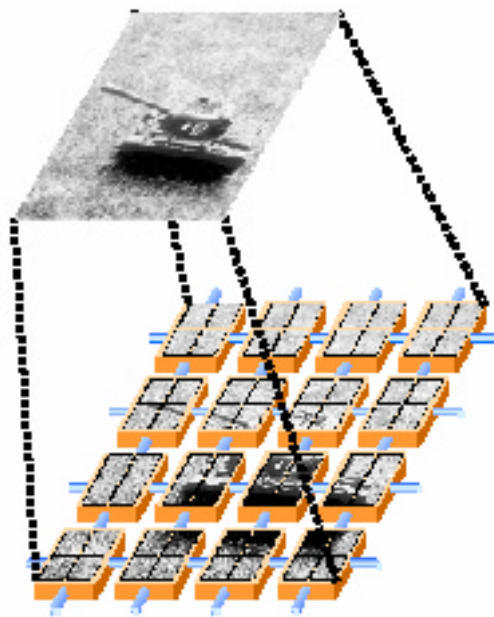


Figura 3.5 - Area di I/O per il mapping dell'immagine nei PE

L'istruzione SAMPLE, campiona simultaneamente i valori di tutti i rivelatori e li rende disponibili per successive elaborazioni. Il modello d'esecuzione SIMD permette all'intera immagine proiettata sui diversi PE di essere campionata in un singolo ciclo, risparmiando il tempo per la distribuzione dei dati ad ogni singolo PE.

Tale campionamento assegna la porzione di immagine ad ogni singolo PE ottenendo una partizione della stessa come mostrato in Figura 3.5.

L'architettura SIMPil è stata implementata in due prototipi funzionali: SIMPil8 e SIMPil16. Il primo prototipo è una implementazione a 8 bit realizzata su tecnologia MOSIS HP da 0.8 μm . Il secondo prototipo, SIMPil16, è stato progettato per una elaborazione completamente digitale per le applicazioni dei portable multimedia supercomputer (PMMS). SIMPil16 è una implementazione a 16 bit, realizzata su tecnologia MOSIS HP da 0.8 μm a CMOS. In Figura 3.6 è mostrato il prototipo di un singolo PE.

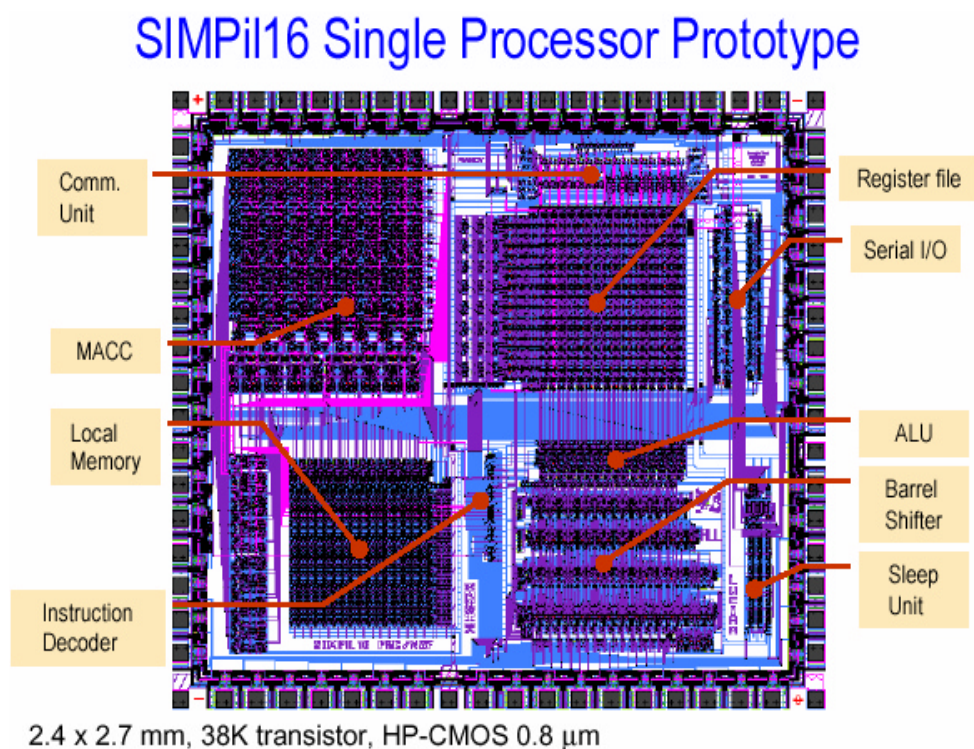


Figura 3.6 - Prototipo di un singolo PE di SIMPil16.

Il PE prototipo misura 2.4 x 2.7 mm., e contiene un totale di 38590 transistor inclusi clock driver, circuiti di testing e I/O pad. Un singolo PE si stima consumi circa 44.1 mW (nello stato attivo) a 5 V, girando a 25 MHz. Le unità funzionali sono illustrate nella Tabella 2.1 in termini di area di silicio e numero di transistor.

Nell'architettura SIMPil, come in ogni architettura SIMD, è presente un Array Control Unit (ACU). Anch'esso ha la stessa struttura semplice dei PE, che consente una ulteriore facilità di integrazione. Nel controllore, però, non sono presenti le stesse unità funzionali che si trovano nei PE. Le principali sono:

- **ALU e Shift unit:** unità logico aritmetica e di shift.
- **Scalar memory:** memoria di 64 parole a 16 bit.
- **Special registers:** registri speciali.
- **Scalar registers:** registri a 16 bit in numero pari a 8.

Unità Funzionali	Area (mm ²)	Numero di transistor
MACC	0.955	13844
MEMORY	0.477	8098
REGFILE	0.697	5974
COMM UNIT	0.214	672
SERIAL I/O	0.207	1006
ALU	0.099	1620
BARREL SHIFTER	0.625	3118
SLEEP UNIT	0.111	394
DECODER	0.031	212
BUS DRIVER	0.033	98

Tabella 3.1 - Unità funzionali di un singolo PE in termini di area di silicio e numero di transistor.

Nel controllore è memorizzato il programma da eseguire, ed ogni istruzione è da esso inviata ai singoli PE. Oltre alle varie istruzioni particolari che il controllore elabora, vi sono anche quelle di salto condizionato ed incondizionato, infatti, tra gli special registers del controllore è presente l'IP (Instruction Pointer).

3.2 – Efficienza dell'architettura Simpil

L'architettura SIMPil è stata progettata per realizzare applicazione di tipo PMMS, come anticipato nell'introduzione del presente capitolo, essendo dei dispositivi portatili devono eseguire operazioni a basso consumo di potenza per poter mantenere il più a lungo possibile la carica delle

batterie. Dal momento che devono eseguire normalmente un gran numero di elaborazioni, è necessario fare in modo che queste vengano svolte in maniera efficiente. Vi sono diversi parametri che consentono di diagnosticare l'efficienza del sistema: l'efficienza di potenza, l'efficienza di area ed il Throughput. La prima è data dal rapporto tra il Throughput e l'area di silicio occupata, la seconda si ottiene dividendo il Throughput per la potenza consumata. Valori elevati di questi due parametri indicano una buona efficienza dell'intero sistema. Il Throughput è dato dal rapporto tra il numero di operazioni ed il tempo necessario per eseguirle, pertanto indica il numero di operazioni eseguite nell'unità di tempo. Per valori elevati di Throughput si hanno maggiori prestazioni del sistema. Gli attuali dispositivi DSP low power hanno un'efficienza di potenza che varia da 0.5 Gops/Joule a 20 Gops/Joule.

Le Figure 3.7 e 3.8 illustrano due grafici in cui si riportano i valori dell'efficienza di potenza e di area dell'architettura SIMPil in relazione con quelli di altre architetture [15].

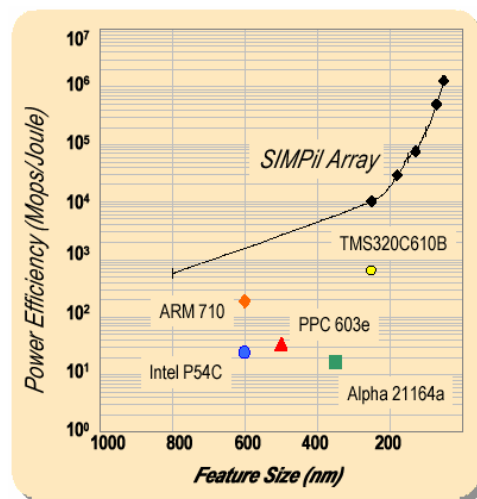


Figura 3.7 - Efficienza di potenza di SIMPil confrontata con quella di altri sistemi

L'architettura SIMPil mostra un'efficienza maggiore di due o tre ordini di grandezza perché ogni processore SIMPil ha una piccolissima local memory che limita il consumo di area di silicio favorendo l'integrazione su un unico chip di un gran numero di PE.

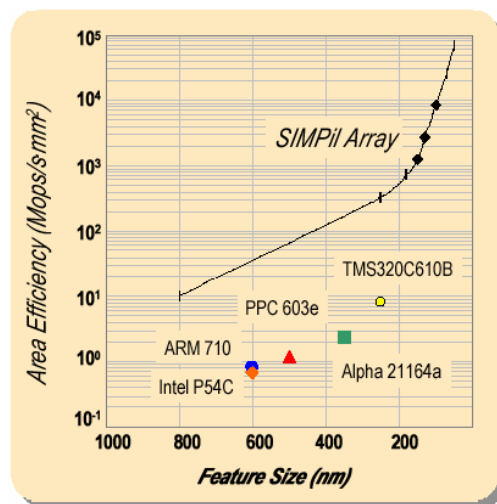


Figura 3.8 - Efficienza di area di SIMPil confrontata con quella di altri sistemi

L'area di I/O manda i necessari dati dell'immagine agli appropriati processori mantenendo alto il throughput dei dati per tenere occupata la matrice di processori. Inoltre, l'architettura SIMPil distribuisce le unità di elaborazione e la memoria tra la matrice di processori ed evita le unità funzionali centralizzate, in questo modo si hanno comunicazioni localizzate e si evitano lunghe interconnessioni globali.

L'architettura implementata per il SIMPil fa parte di una classe di sistemi paralleli che sono stati progettati e realizzati già da qualche anno. Per valutare al meglio le prestazioni dell'ambiente SIMPil, esso è stato sottoposto a banchi di prova dedicati utilizzando una suite di applicazioni appropriate per saggiarne le qualità.

I risultati ottenuti sono stati messi a confronto con quelli riscontrati in altri ambienti multiprocessor [15]. È importante sottolineare che molti di questi sistemi, riportati in Tabella 2.2,

furono implementati alcuni anni fa utilizzando le tecnologie disponibili allora. Il confronto proposto, quindi, non ha la pretesa di trarre conclusioni in termini competitivi tra le piattaforme, ma, piuttosto, vuole fare chiarezza sulle potenzialità che possono essere sfruttate e sull'ordine di grandezza delle caratteristiche in gioco.

Piattaforme	Numero PE	Ck [Mhz]	Dimensione Immagine	Tempo d'Esecuzione [μs]	Prestazione (pxl/cls)	Prestazione (Mpxl/sec)
SIMPil	4.096	500	256x256	3	47	23.456
Apx	256	12,5	512x512	5.000	4	52
CLIP4	9.216	40	96x96	16.050	0,01	1
DAP	16.384	12,5	64x64	336	1	12
MMP	16.385	10	128x128	80	41	410
GAPP	12.288	10	96x108	300	4	35
MGAP	16.385	25	128x128	57	12	287
Slim	5.000	25	100x50	7	29	714
ICPA	64	100	640x480	1.730	2	177
RaPiD	16	100	256x256	655	1	100

Tabella 3.2 - Confronto tra SIMPil ed altri sistemi multiprocessor

3.3 - Il Simulatore SIMPil

In questa sezione si fa riferimento allo strumento software "SIMPil 16 Simulator v3.20". Esso è un simulatore a livello d'istruzione che permette l'editing, l'assembling, l'esecuting e il debugging di applicazioni parallele in un singolo IDE (Integrated Development Environment) per l'architettura SIMPil. Una schermata del simulatore è mostrata in Figura 3.9.

Il linguaggio utilizzato nel simulatore, è un linguaggio assembler che, come si è visto nel set d'istruzioni, è composto da istruzioni scalari e vettoriali. Le istruzioni scalari sono eseguite nel controllore, mentre quelle vettoriali sono inviate a ciascun PE. Le costanti sono utilizzate per definire parametri architetturali quali la dimensione della memoria locale di ciascun PE, la

dimensione della matrice di processori, il PPE ratio e la topologia della rete (standard NEWS o Torus-NEWS). È disponibile un help on-line per avere informazioni sui formati delle istruzioni, sull'architettura del sistema e sulle operazioni e funzionalità del simulatore.

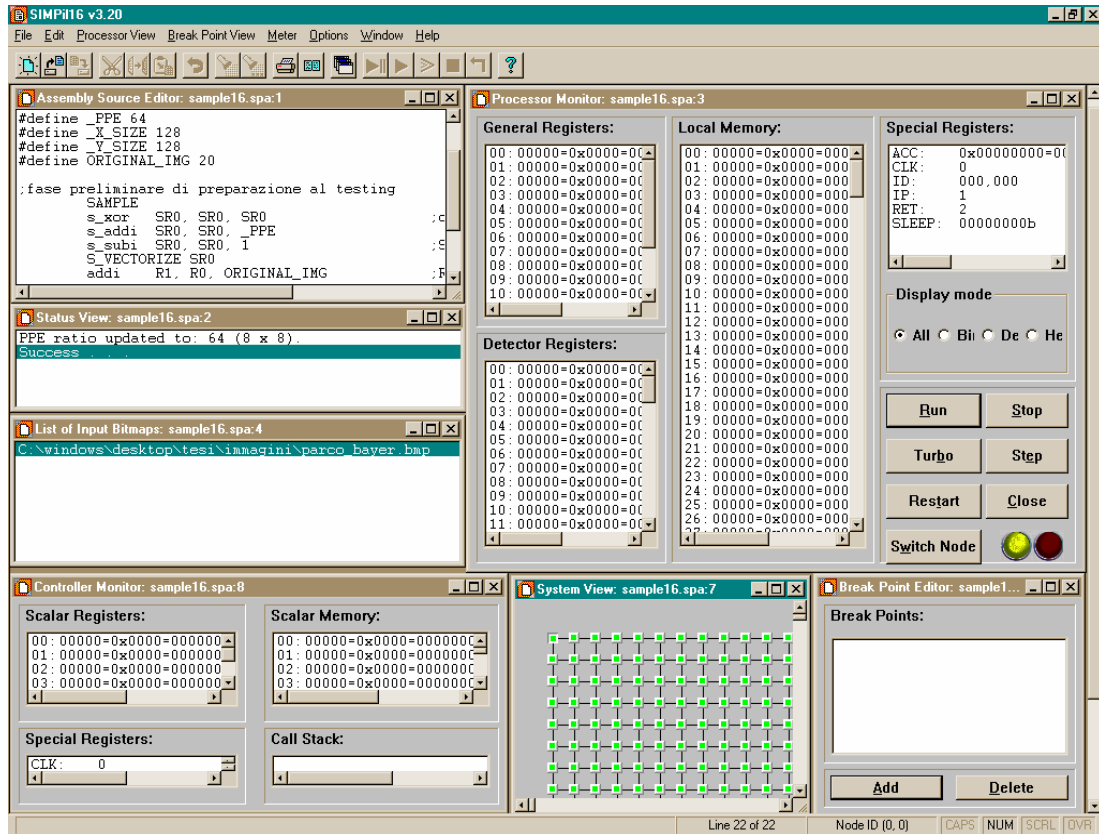


Figura 3.9 - Schermata del simulatore SIMPIL

Per ogni PE che è possibile selezionare si può controllare il suo stato (valori dei registri, della memoria e dei rivelatori). Tramite la finestra “Controller Monitor View” si può controllare lo stato del controllore. Sono disponibili tre differenti modi d'esecuzione del programma per permettere un facile debugging d'elaborazione e possono essere impostati dei breakpoint per fermare l'esecuzione. Complessivamente, l'attività del sistema può essere esaminata per mezzo del Sistem View che permette una veloce selezione dei PE. Il simulatore è anche progettato per misurare il livello di concorrenza e la distribuzione delle istruzioni durante l'esecuzione dell'applicazione. Questa funzionalità, assieme al numero di cicli di clock effettuati per ogni applicazione, servirà nel calcolo delle misure di prestazioni e nel confronto con altri sistemi.

3.3.1 Formato delle istruzioni SIMPil

L'Instruction Set Architecture (ISA) del SIMPil, è stato progettato tenendo in considerazione i requisiti prestazionali e le limitazioni nel consumo energetico che un sistema come il SIMPil, indirizzato alle applicazioni multimediali su dispositivi portatili, deve soddisfare.

	opcode	dest	src1	src2	
Instruction Format A:	21-16	15-12	11-8	7-4	3-0
	opcode	dest	src1	immediate	
Instruction Format B:	21-16	15-12	11-8	7-0	
	opcode		src1	immediate	
Instruction Format C:	21-16	15-12	11-8	7-0	
	opcode			immediate	
Instruction Format D:	21-16	15-8		7-0	
	opcode	dest	src1		
Instruction Format E:	21-16	15-12	11-8	7-0	
	opcode	dest		immediate	
Instruction Format F:	21-16	15-12	11-8	7-0	
	opcode				
Instruction Format G:	21-16	15-0			
	opcode	dest			
Instruction Format H:	21-16	15-12	11-0		
	opcode	immediate			
Instruction Format I:	21-16	15-0			

Figura 3.10 - *Elenco dei formati delle istruzioni.*

Sono stati compiuti studi sul progetto dell'ISA per determinare quali caratteristiche, più di altre, conducono ad operazioni ad elevata efficienza energetica. Le due filosofie contrastanti sono il modello CISC e quello RISC. Il CISC ISA raggiunge maggiori densità di codice per mezzo dei

formati a lunghezza variabile che riducono l'energia per fetch d'istruzione e la lunghezza del programma. Al contrario, il RISC ISA permette operazioni con maggiore efficienza d'energia grazie ai più semplici dati e path di controllo, al costo di una maggiore lunghezza di programma. Inoltre, il formato delle istruzioni è di dimensione fissa a 24 bit per ridurre ulteriormente la logica di decodifica delle istruzioni.

Classi di istruzioni	Istruzioni
<i>Nop</i>	NOP
<i>Arithmetic</i>	ADD, ADDI, SUB, SUBI, MUL, MULI
<i>Logical</i>	AND, ANDI, OR, ORI, XOR, XORI
<i>Shift</i>	LSH, LSHI, ASH, ASHI, ROT, ROTI
<i>Sleep</i>	SGT, SGTI, SGE, SGEI, SLT, SLTI, SLE, SLEI, SEQ, SEQI, SNE, SNEI, USGT, USGTI, USGE, USGEI, USLT, USLTI, USLE, USLEI, USEQ, USEQI, USNE, USNEI
<i>Wakeup</i>	WAKEUP, WAKEUPI
<i>Communication</i>	XFER
<i>Memory</i>	LOAD, STORE, LOADI
<i>Multiply-Accumulate</i>	MACC, ZACC, LACCL, LACCH
<i>Detector</i>	SAMPLE, PLOAD, PLOADS

Tabella 3.3 - Elenco delle istruzioni del SIMPIL Processor ISA.

Tutte le configurazioni di formato disponibili sul SIMPIL sono illustrate nella Figura 3.10. I campi del formato delle istruzioni sono di seguito descritti:

- nel campo Opcode di 5-bit, va inserito il codice dell'istruzione da eseguire;
- i campi Dest, Src1, Src2, tutti di 4 bit, contengono rispettivamente l'indirizzo dei registri di destinazione, di sorgente 1 e di sorgente 2 che sono utilizzati nell'istruzione;
- nel campo Immediate di 8 o 16 bit va inserita una costante numerica intera.

Per distinguere le istruzioni da inviare ai nodi o al controllore, si implementano due insiemi di istruzioni: il set di istruzioni vettoriali proprio dei PE (SIMPil Processor ISA), e il set di istruzioni scalari proprio del Controllore (SIMPil Controller ISA). Inoltre, vi sono delle istruzioni specifiche del simulatore dell'architettura SIMPil.

Classi di istruzioni	Istruzioni
<i>Raise-Hand</i>	S_RAISEHAND
<i>Vectorize</i>	S_VECTORIZE
<i>Arithmetic</i>	S_ADD, S_ADDI, S_SUB, S_SUBI, S_MUL, S_MULI
<i>Logical</i>	S_AND, S_ANDI, S_OR, S_ORI, S_XOR, S_XORI
<i>Shift</i>	S_LSH, S_LSHI, S_ASH, S_ASHI, S_ROT, S_ROTI
<i>Branch</i>	S_BGT, S_BGE, S_BLT, S_BLE, S_BEQ, S_BNE, S_BRA
<i>Subroutine</i>	S_CALL, S_RETURN
<i>Memory</i>	S_LOAD, S_STORE, S_LOADI
<i>End</i>	S_END

Tabella 3.4 - Elenco delle istruzioni del SIMPil Controller ISA.

Su elaboratori con elevato numero d'operazioni per unità di tempo, il più semplice RISC ISA è preferibile perché il maggior numero di processori, che lavorano concorrentemente, può facilmente attenuare l'incremento della lunghezza del programma, pertanto visto il campo d'applicazione e l'elevato throughput, l'architettura SIMPil utilizza un RISC ISA a virgola fissa.

Nella Tabella 2.3 sono state elencate le istruzioni del *SIMPil Processor ISA*, vale a dire le istruzioni riguardanti i singoli PE; queste sono delle istruzioni vettoriali, perché un'unica istruzione viene eseguita contemporaneamente su più processori e su dati diversi. Nella Tabella 2.4, invece, sono state elencate le istruzioni del *SIMPil Controller ISA*, vale a dire le istruzioni riguardanti il controllore. Queste ultime, sono delle istruzioni scalari perché eseguite solamente dal controllore e non inviate ai singoli PE.

3.3.2 Set di istruzioni vettoriali (SIMPil Processor ISA)

Sono l'insieme delle istruzioni che vengono svolte in ogni processore elementare. Durante l'esecuzione, ogni nodo eseguirà la stessa istruzione, tuttavia, ognuno avrà il relativo insieme di dati. Di seguito sono elencate le istruzioni, spiegate per classe di appartenenza, con un esempio di sintassi e una breve descrizione. I parametri x, y, z rappresentano dei numeri interi tra 0 e 15.

- Istruzioni Aritmetiche

Eseguono le operazioni di somma, sottrazione e prodotto. I formati utilizzati sono A e B.

ADD	Rz, Rx, Ry	Addizione con segno tra i registri Rx e Ry con inserimento della somma nel registro Rz. ($Rz \leq Rx + Ry$)
ADDI	Rz, Rx, IMMED	Addizione con segno tra il registro Rx e la costante IMMED con inserimento della somma in Rz. ($Rz \leq Rx + IMMED$)
SUB	Rz, Rx, Ry	Sottrazione tra i registri Rx e Ry con inserimento della differenza in Rz. ($Rz \leq Rx - Ry$)
SUBI	Rz, Rx, IMMED	Sottrazione tra il registro Rx e la costante IMMED con inserimento della differenza in Rz. ($Rz \leq Rx - IMMED$)
MUL	Rz, Rx, Ry	Moltiplicazione tra i registri Rx e Ry con inserimento del risultato in Rz. ($Rz \leq Rx + IMMED$)
MULI	Rz, Rx, IMMED	Moltiplicazione tra il registro Rx e la costante IMMED con inserimento del risultato in Rz. ($Rz \leq Rx + IMMED$)

- Istruzioni di Moltiplicazione e Accumulazione

Sono istruzioni speciali che utilizzano un registro accumulatore (ACC) di 32 bit. I formati utilizzati sono A, G e I.

MACC	Rz, Rx, Ry	Moltiplica con segno i registri Rx e Ry, somma al prodotto il valore dell'accumulatore ed inserisce il risultato nei registri Rz e ACC. (Rz, ACC $\leftarrow$ Rx * Ry + ACC)
ZACC		Azzera l'accumulatore (ACC $\leftarrow$ 0)
LACCL	Rz	Carica i 16 bit meno significativi dell'accumulatore nello specificato registro destinazione Rz $\leftarrow$ (ACC & 0xFFFF)
LACCH	Rz	Carica i 16 bit più significativi dell'accumulatore nello specificato registro destinazione Rz $\leftarrow$ ((ACC >> 16) & 0xFFFF)

- Istruzioni Logiche

Eseguono le operazioni logiche bit a bit tra numeri binari. I formati utilizzati sono A e B

AND	Rz, Rx, Ry	AND tra i bit dei registri Rx e Ry con inserimento del risultato nel registro Rz. (Rz $\leftarrow$ Rx AND Ry)
ANDI	Rz, Rx, IMMED	AND tra i bit di Rx e la costante IMMED con inserimento del risultato nel registro Rz. (Rz $\leftarrow$ Rx AND IMMED)
OR	Rz, Rx, Ry	OR inclusivo tra i bit dei registri Rx e Ry con inserimento del risultato nel registro Rz. (Rz $\leftarrow$ Rx OR Ry)
ORI	Rz, Rx, IMMED	OR inclusivo tra i bit di Rx e la costante IMMED con inserimento del risultato nel registro Rz. (Rz $\leftarrow$ Rx OR IMMED)
XOR	Rz, Rx, Ry	XOR esclusivo tra i bit dei registri Rx e Ry con inserimento del risultato nel registro Rz. (Rz $\leftarrow$ Rx XOR Ry)
XORI	Rz, Rx, IMMED	XOR esclusivo tra i bit di Rx e la cost. IMMED con inserimento del risultato nel registro Rz. (Rz $\leftarrow$ Rx XOR IMMED)

- Istruzioni di Shift

Eseguono le operazioni di shift, logico o aritmetico, e di rotazione avvalendosi di un registro speciale (barrel shift) di 16 bit. I formati che utilizzano sono A e B.

<i>LSH</i>	Rz, Rx, Ry	Shift Logico di Rx di bit pari a Ry con inserimento del risultato in Rz. (Rz <= Rx shift logico di Ry)
<i>LSHI</i>	Rz, Rx, IMMED	Shift Logico di Rx di bit pari a IMMED con inserimento del risultato in Rz. (Rz <= Rx shift log. Di IMMED)
<i>ASH</i>	Rz, Rx, Ry	Shift Aritmetico di Rx di bit pari a Ry con inserimento del risultato in Rz. (Rz <= Rx shift aritmetico di Ry)
<i>ASHI</i>	Rz, Rx, IMMED	Shift Aritmetico di Rx di bit pari a IMMED con inserimento del risultato in Rz. (Rz <= Rx shift aritmetico di IMMED)
<i>ROT</i>	Rz, Rx, Ry	Rotazione di Rx di bit pari a Ry con inserimento del risultato in Rz. (Rz <= Rx rotate di Ry)
<i>ROTI</i>	Rz, Rx, IMMED	Rotazione di Rx di bit pari a IMMED con inserimento del risultato in Rz. (Rz <= Rx rotate di IMMED)

La direzione dello shift o rotazione è determinata dal segno (+/-) del registro Ry o del valore IMMED. Un valore positivo di Ry o di IMMED rappresenta uno shift od una rotazione verso sinistra. Un valore negativo di Ry o di IMMED rappresenta uno shift od una rotazione verso destra.

Ricordando che il SIMPil lavora con numeri binari rappresentati in complemento a uno, nello shift logico e nella rotazione si può perdere il significato del bit di segno. Invece nello shift aritmetico si mantiene sempre il valore del bit di segno (viene aggiunto dopo lo shift).

- Istruzione di Campionamento

Effettua il campionamento dell'immagine dai sensori ottici tramite un convertitore A/D. Il formato utilizzato è G.

<i>SAMPLE</i>	Campiona i dati dei rivelatori e li memorizza nei Pixel Register
----------------------	--

- Istruzioni di Sleep

Eseguono le operazioni di tipo condizionale disattivando i nodi in cui la condizione è verificata. Il formato utilizzato è C.

SGT	Rz, Rx	Disattiva se Rz maggiore di 0, S_VECT <= S_VECT OR Rx
SGTI	Rz, IMMED	Disattiva se Rz maggiore di 0, S_VECT <= S_VECT OR IMMED
SGE	Rz, Rx	Disattiva se Rz magg. o uguale a 0, S_VECT <= S_VECT OR Rx
SGEI	Rz, IMMED	Disattiva se Rz magg o ug. a 0, S_VECT <= S_VECT OR IMMED
SLT	Rz, Rx	Disattiva se Rz minore di 0, S_VECT <= S_VECT OR Rx
SLTI	Rz, IMMED	Disattiva se Rz minore di 0, S_VECT <= S_VECT OR IMMED
SLE	Rz, Rx	Disattiva se Rz minore o uguale a 0, S_VECT <= S_VECT OR Rx
SLEI	Rz, IMMED	Disattiva se Rz min. o uguale a 0, S_VECT<=S_VECT OR IMMED
SEQ	Rz, Rx	Disattiva se Rz uguale a 0, S_VECT <= S_VECT OR Rx
SEQI	Rz, IMMED	Disattiva se Rz uguale a 0, S_VECT <= S_VECT OR IMMED
SNE	Rz, Rx	Disattiva se Rz diverso da 0, S_VECT <= S_VECT OR Rx
SNEI	Rz, IMMED	Disattiva se Rz diverso da 0, S_VECT <= S_VECT OR IMMED
USGT	Rz, Rx	Disattiva se Rz maggiore di 0, S_VECT <= S_VECT OR Rx
USGTI	Rz, IMMED	Disattiva se Rz maggiore di 0, S_VECT <= S_VECT OR IMMED
USGE	Rz, Rx	Disattiva se Rz maggiore o uguale a 0, S_VECT<= S_VECT OR Rx
USGEI	Rz, IMMED	Disattiva se Rz magg. o ug. a 0, S_VECT<= S_VECT OR IMMED
USLT	Rz, Rx	Disattiva se Rz minore di 0, S_VECT <= S_VECT OR Rx
USLTI	Rz, IMMED	Disattiva se Rz minore di 0, S_VECT <= S_VECT OR IMMED
USLE	Rz, Rx	Disattiva se Rz minore o uguale a 0, S_VECT <= S_VECT OR Rx
USLEI	Rz, IMMED	Disattiva se Rz min. o uguale a 0, S_VECT<=S_VECT OR IMMED
USEQ	Rz, Rx	Disattiva se Rz uguale a 0, S_VECT <= S_VECT OR Rx
USEQI	Rz, IMMED	Disattiva se Rz uguale a 0, S_VECT <= S_VECT OR IMMED
USNE	Rz, Rx	Disattiva se Rz diverso da 0, S_VECT <= S_VECT OR Rx
USNEI	Rz, IMMED	Disattiva se Rz diverso da 0, S_VECT <= S_VECT OR IMMED

Nel registro speciale S_VECT di 8 bit, presente in ogni PE, viene inserito, se la condizione è verificata, l'OR bit a bit tra S_VECT e Rx o IMMED. Quando il PE è disattivo le istruzioni vettoriali che vengono eseguite sono quelle di comunicazione, di riattivazione, di campionamento e

quelle appartenenti alla famiglia USXXX. Queste servono maggiormente per aggiornare lo sleep vector anche quando il PE è disattivato.

- Istruzione di Comunicazione

Permette lo scambio d'informazioni tra PE adiacenti. Il formato utilizzato è di tipo E. Questo tipo di comunicazione viene detta NEWS.

<i>XFER</i>	Rz, Rx, NORTH	Si trasferisce il valore contenuto in Rx al registro Rz del nodo confinante secondo la direzione specificata nel campo immediate (NORTH, EAST, WEST, SOUTH).
<i>XFER</i>	Rz, Rx, EAST	
<i>XFER</i>	Rz, Rx, WEST	
<i>XFER</i>	Rz, Rx, SOUTH	

- Istruzioni di Wakeup

Riattivano i nodi in cui l'AND bit a bit tra S_VECT e Rx o IMMED produce un valore maggiore di 0. Il formato utilizzato è D. Inoltre, se il PE viene attivato, il registro S_VECT viene posto a zero.

<i>WAKEUP</i>	Rx	Se S_VECT AND Rx maggiore di 0 si riattiva il nodo.
<i>WAKEUPI</i>	IMMED	Se S_VECT AND IMMED maggiore di 0 si riattiva il nodo.

- Istruzioni di Interfaccia con la Memoria Locale

Permettono il trasferimento di informazioni dalla memoria ai registri e viceversa. I formati che utilizzano sono E, F e I.

<i>LOAD</i>	Rz, Rx	Carica il valore contenuto nella locazione di memoria di indirizzo Rx nel registro Rz. $Rz \leq *(Rx)$
<i>STORE</i>	Rz, Rx	Carica il valore contenuto nel registro Rx nella locazione di memoria di indirizzo Rz. $*(Rz) \leq Rx$
<i>LOADI</i>	IMMED	Inserisce il valore IMMED nel registro R0

Converte ciascuna intensità dei pixel in un valore 8-bit senza segno e di conseguenza il valore di un pixel è un numero intero positivo compreso da 0 a 255. Questa istruzione è eseguita anche quando il nodo è disattivato. Dopo l'esecuzione dell'istruzione SAMPLE, per il trasferimento di informazioni dai Pixel Register ai registri vi sono delle istruzioni apposite di sotto elencate. Il formato utilizzato è F.

PLOAD	Rz, IMMED	Carica il valore del pixel con indice IMMED in Rz, (Rz <= P _{IMMED})
PLOADS	Rz, SRx	Carica il valore del pixel con indice SRx in Rz, (Rz <= SRx)

Il numero dei rivelatori ottici corrisponde al numero di pixel per processore elementare (PPE). Se la porzione di immagine, destinata ad ogni nodo, è un array di 4x4 pixel allora vi saranno 16 rivelatori (uno per pixel). I registri del Pixel Register sono numerati in maniera tale che: P0, P1, P2, e P3 sono quelli della prima riga; P4, P5, P6, e P7 sono quelli della seconda riga; P8, P9, P10, e P11 sono quelli della terza riga; P12, P13, P14, e P15 sono quelli della quarta riga.

- Istruzione NOP

L'istruzione NOP non compie nessuna operazione. Viene utilizzata principalmente per rendere più leggibile ed elegante il codice. Si può trovare da sola o accostata ad una Label. Il formato dell'istruzione è di tipo G.

3.3.3 - Set di istruzioni scalari (SIMPil Controller ISA)

Il set delle istruzioni scalari vengono eseguite solo dal controllore. Esse, per distinguersi dalle istruzioni vettoriali, hanno prefisso "S_" nel nome del comando.

Di seguito sono elencate le istruzioni scalari, spiegate per classe di appartenenza, con un esempio di sintassi e una breve descrizione. I parametri x, y, z rappresentano dei numeri interi tra 0 e 7.

- Istruzione Vectorize

Questa istruzione associa il valore di un registro scalare ai registri vettoriali R0 di ciascun PE.

Salta se SRx è minore di 0

S_VECTORIZE	SRz	Invia il contenuto di SRz al registro vettoriale R0 di ciascun PE. $R0 \leq SRz$
--------------------	-----	--

- Istruzione Raise-Hand

Nel corso dell'implementazione di un programma (in genere dopo l'esecuzione di una istruzione di SLEEP), per controllare se vi è almeno un PE attivo, si utilizza l'istruzione S_RAISEHAND. Se qualche PE (uno o più) è attivo, allora un 1 viene scritto nel registro scalare, altrimenti viene memorizzato uno 0. Il formato utilizzato è H.

S_RAISEHAND	SRz	Se vi è almeno un PE attivo si mette 1 in SRz, viceversa 0
--------------------	-----	--

- Istruzioni Aritmetiche

Eseguono le operazioni di somma, sottrazione e prodotto dei registri scalari. I formati che utilizzano sono A e B.

S_ADD	SRz, SRx, SRy	Somma con segno. $SRz \leq SRx + Sry$
S_ADDI	SRz, SRx, IMMED	Somma con segno. $SRz \leq SRx + IMMED$
S_SUB	SRz, SRx, SRy	Sottrazione con segno. $SRz \leq SRx - Sry$
S_SUBI	SRz, SRx, IMMED	Sottrazione con segno. $SRz \leq SRx - IMMED$
S_MUL	SRz, SRx, SRy	Moltiplicazione con segno. $SRz \leq SRx * Sry$
S_MULI	SRz, SRx, IMMED	Moltiplicazione con segno. $SRz \leq SRx * IMMED$

- Istruzione S_End

Questa istruzione indica la fine del programma principale. Il formato che utilizza è G.

S_END		Fine Programma.
--------------	--	-----------------

- Istruzioni Logiche

Effettuano operazioni logiche AND, OR e XOR bit a bit tra registri scalari. I formati che utilizzano sono A e B.

S_AND	SRz, SRx, SRy	AND bit a bit. $SRz \leftarrow SRx \text{ AND } SRy$
S_ANDI	SRz, SRx, IMMED	AND bit a bit, con una costante. $SRz \leftarrow SRx \text{ AND } IMMED$
S_OR	SRz, SRx, SRy	OR inclusivo bit a bit. $SRz \leftarrow SRx \text{ OR } SRy$
S_ORI	SRz, SRx, IMMED	OR inclusivo bit a bit. $SRz \leftarrow SRx \text{ OR } IMMED$
S_XOR	SRz, SRx, SRy	XOR esclusivo bit a bit. $SRz \leftarrow SRx \text{ XOR } SRy$
S_XORI	SRz, SRx, IMMED	XOR esclusivo bit a bit. $SRz \leftarrow SRx \text{ XOR } IMMED$

- Istruzioni di Shift.

Effettuano operazioni di Shift per i registri scalari. I formati che sono A e B.

S_LSH	SRz, SRx, SRy	Shift Logico di SRx di bit pari a SRy con inserimento del risultato in SRz. $SRz \leftarrow SRx \text{ shift logico di } SRy$
S_LSHI	SRz, SRx, IMMED	Shift Logico di SRx di bit pari a IMMED con inserimento del risultato in SRz. $SRz \leftarrow SRx \text{ shift logico di } IMMED$
S_ASH	SRz, SRx, SRy	Shift Aritmetico di SRx di bit pari a SRy con inserimento del risultato in SRz. $SRz \leftarrow SRx \text{ shift aritmetico di } SRy$
S_ASHI	SRz, SRx, IMMED	Shift Aritmetico di SRx di bit pari a IMMED con inserimento del risultato in SRz. $SRz \leftarrow SRx \text{ shift aritmetico di } IMMED$
S_ROT	SRz, SRx, SRy	Rotazione di SRx di bit pari a SRy con inserimento del risultato in SRz. $SRz \leftarrow SRx \text{ rotazione di } SRy$
S_ROTI	SRz, SRx, IMMED	Rotazione di SRx di bit pari a IMMED con inserimento del risultato in SRz. $SRz \leftarrow SRx \text{ rotazione di } IMMED$

- Istruzioni di Branch

Sono istruzioni di salto condizionato. Se la condizione è verificata, permettono di saltare all'istruzione con Label specificata ($IP = IP_{label}$), altrimenti si passa all'istruzione successiva ($IP = IP + 1$). Il formato utilizzato è C.

S_BGT	SRz, LBL	Branch If Greater Than - Salta se SRz è maggiore di zero
S_BGE	SRz, LBL	Branch If Greater Than or Equal -Salta se SRz è magg. o uguale a zero
S_BLT	SRz, LBL	Branch If Less Than - Salta se SRz è minore di zero
S_BLE	SRz, LBL	Branch If Less Than or Equal - Salta se SRz è minore o uguale a zero
S_BEQ	SRz, LBL	Branch If Equal - Salta se SRz è uguale a zero
S_BNE	SRz, LBL	Branch If Not Equal - Salta se SRz è diverso da zero

Queste istruzioni vengono utilizzate maggiormente per effettuare cicli, come il WHILE o il FOR per il linguaggio C. Vi è poi un'istruzione che permette il salto incondizionato, cioè cambia il puntatore all'istruzione ($IP = IP_{label}$) senza che sia verificata alcuna condizione. Il formato utilizzato è D.

S_BRA	LBL	Branch Always - Salta sempre ($IP = IP_{lbl}$)
--------------	-----	--

- Istruzioni per le Subroutine

Per evitare di scrivere più volte la stessa sequenza di istruzioni, in SIMPil è possibile scrivere Subroutine (sotto-blocchi di codice) ed effettuarne la chiamata quando sono necessari. L'istruzione che permette di far saltare il puntatore $IP+1$ (Instruction Pointer) al punto dove si trova la Subroutine è S_CALL seguita dalla LABEL di riferimento del sotto-blocco di codice. Il formato di questa istruzione è D. Alla fine del blocco deve essere presente l'istruzione S_RETURN che

permette di ritornare all'istruzione subito successiva alla chiamata *S_CALL*. Il formato di questa istruzione è G.

<i>S_CALL</i>	Nome_Subroutine	Chiama la Subroutine specificata. $RET \leq IP + 1$
<i>S_RETURN</i>		Ritorna dalla Subroutine. $IP \leq RET$

Nella chiamata di una subroutine, se si modificano dei registri, questi restano modificati anche nel programma principale. Quindi, si deve stare attenti, ogni volta, se registri che devono mantenere il proprio valore vengono per errore aggiornati, anche dopo il ritorno al blocco di codice iniziale. Inoltre in SIMPil vi sono solo 10 livelli di stack, quindi non si possono effettuare più di dieci chiamate iterativamente alla volta (non si possono fare ricorsioni).

- Istruzioni di interfaccia con la Memoria locale del Controllore

Permettono il trasferimento di informazioni dalla memoria scalare ai registri scalari e viceversa. I formati che utilizzano sono E, F e I.

<i>S_LOAD</i>	SRz, SRx	Carica il valore contenuto nella locazione di memoria di indirizzo SRx nel registro scalare SRz. $SRz \leq *(SRx)$
<i>S_STORE</i>	SRz, SRx	Carica il valore contenuto nel registro SRx nella locazione di memoria scalare di indirizzo SRz. $*(SRz) \leq SRx$
<i>S_LOADI</i>	IMMED	Inserisce il valore IMMED nel registro.

- Istruzioni specifiche del simulatore

Si è sviluppato un software che simuli il funzionamento dell'architettura SIMPil. Questo permette di verificare la correttezza e l'efficienza di un programma senza dover ricorrere all'architettura completa. Vi sono alcune istruzioni assembler che il simulatore SIMPil riconosce come estranee al SIMPil ISA. In più, come nel linguaggio assembler, il simulatore è sensibile al ritorno a capo della linea. Vale a dire, una singola istruzione (con l'etichetta relativa, se compresa)

deve essere contenuta in una singola linea. Inoltre, per scrivere un commento è necessario inserire il carattere di punto e virgola (;) che è valido fino al ritorno a capo della linea.

Tali direttive sono elencate di seguito, assieme ad una breve descrizione del significato:

- **OS_LOAD_ID :** *OS_LOAD_ID* R_x -> $R_x = \text{node ID}$

Direttiva con cui si carica, nel registro specificato R_x , il *node ID* che identifica la posizione di ogni PE nell'ambito dell'intero array di processori.

- **OS_BITMAP_OUT :** *OS_BITMAP_OUT Immediate_Value*

Serve a produrre un'immagine bitmap nell'Output Bitmap View, in base ai valori memorizzati in ciascun PE, a partire dalla locazione di memoria *Immediate_Value*.

- **OS_DUMP_MEM :** *OS_DUMP_MEM Immediate_Value*

Effettua il “dump”, cioè la trascrizione in un file, dei contenuti degli indirizzi di memoria di ciascun PE, a partire dalla locazione di memoria *Immediate_Value*.

- **OS_STREAM_IN :** *OS_STREAM_IN* SR_x

Legge un numero a 16 bit da un file di testo e lo mette nel registro scalare SR_x del controllore.

- **#DEFINE :** *#DEFINE constant_name constant_value*

Essa è una macro che serve per la definizione delle costanti.

- **#INCLUDE:** *#INCLUDE file_name*

Con tale direttiva, è possibile includere codice assembler SIMPil da altri file. In particolare è possibile richiamare codice appartenente a delle subroutine implementate nello stesso file oppure in altri file inclusi. La chiamata alle subroutine è possibile tramite il comando *S_CALL* come segue:

S_CALL subroutine_name: se la subroutine si trova nel file del corrente main.

S_CALL file_name.subroutine_name: se la subroutine si trova in un file diverso.

Ovviamente il file che contiene la subroutine da richiamare deve essere incluso con l'apposita direttiva. Inoltre è ammesso solamente un livello di profondità. Ciò vuol dire che non sono ammessi chiamate a subroutine annidate.

4. Implementazione del Processo di Elaborazione dell'Immagine Digitale

In questa sezione si effettuerà l'analisi dell'intero processo di elaborazione dell'immagine che inizia con la sua acquisizione per mezzo dei sensori ottici, per finire con la stessa immagine scomposta nello spazio YCrCb a cui vengono apportate le correzioni cromatiche implementate.

L'intero processo può essere diviso in due parti principali: il *Preprocessing* e il *Color Processing*.

Il *Preprocessing* rappresenta la prima parte di elaborazione eseguita e più specificatamente le operazioni di:

- **Black Clamping:** viene corretto il range dinamico dei pixel, evitando di attribuirgli valori negativi, utilizzando un offset fornito dal costruttore del sensore ottico.
- **Lens Distortion Compensation:** si modificano i valori dei pixel per eliminare le imperfezioni dettate dalla messa a fuoco delle lenti.
- **Fault Pixel Interpolation:** si identificano e correggono i valori dei pixel che corrispondono a sensori non funzionanti per difetto di fabbrica.
- **White Balancing:** viene ripristinato il valore del bianco (255, 255, 255) nelle parti di immagine sottoesposte cosicché il colore venga bilanciato proporzionalmente.
- **Gamma Correction:** viene corretta la non-linearità introdotta nel *Mapping* dell'immagine dai sensori ottici ai dispositivi di visualizzazione.

Nella parte di *Color Processing* rientrano le operazioni di:

- **CFA Interpolation:** a partire dall'immagine in formato *Bayer Pattern*, vengono ricavate distintamente le tre componenti RGB.
- **Color Space Conversion:** le tre componenti RGB vengono convertite nello spazio YCrCb ottenendo le relative componenti

- **Edge Detection e False Color Correction:** rappresentano l'ultimo passo da compiere per ottenere la completa correzione cromatica per poi passare alla visualizzazione o alla compressione dell'immagine; queste operazioni sono state ampiamente descritte e studiate in questa tesi.

4.1 - Sequenza di Elaborazione

I risultati ottenuti dall'elaborazione di un'immagine verranno mostrati in questo paragrafo. Si considera un'immagine di partenza CFA in formato *Bayer* (così come ottenuta dai sensori ottici). Inoltre, per rispettare condizioni non favorevoli di luminosità ambientale, l'immagine è sottoesposta, risultando, cioè, più scura del normale. Essa è mostrata dalla Figura 4.1.



Figura 4.1 - Immagine origine ottenuta dai sensori ottici

Notiamo come alcuni pixel, rappresentati dai punti bianchi in Figura 4.1, sono privi di valore significativo. Ciò è dovuto dal fatto che alcuni sensori possono essere danneggiati per motivi di fabbricazione e, quindi, non assolvono correttamente alla loro funzione. Il numero di pixel difettosi simulati risulta essere elevato più di quanto si possa ottenere anche in dispositivi di scarsa qualità perché si vuole mettere in palese evidenza il risultato delle elaborazioni.

La fase di *Preprocessing* consta di cinque passi, alla fine dei quali i risultati ottenuti sono rappresentati dalle figure seguenti.

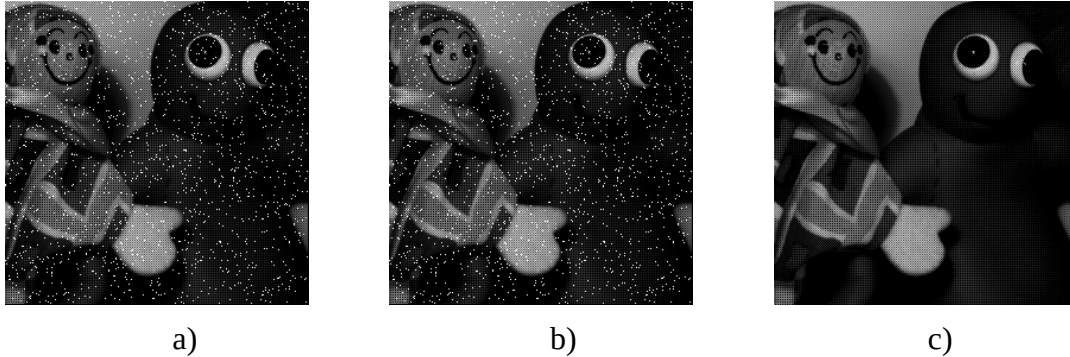


Figura 4.2 - a) *Black Camping*; b) *Lens Distortion*; c) *Falt Pixel Correction*

Le tre immagini di Figura 4.2 sono state ottenute applicando le maschere fornite dal costruttore del dispositivo. Per questa prova le maschere sono state simulate. I miglioramenti sono più evidenti nella terza figura in cui sono stati corretti i pixel con valori errati.



Figura 4.3 - a) *White Balancing*; b) *Gamma Correction*

La Figura 4.3 mostra i risultati degli ultimi passi del *Preprocessing*. I benefici ottenuti sono evidenti se le confrontiamo con l'immagine originale.

La fase eseguita successivamente è il *Color Processing*. Anch'essa è divisibile in passi dopo i quali otteniamo delle immagini, proposte di seguito, utili per valutare la bontà degli algoritmi implementati.

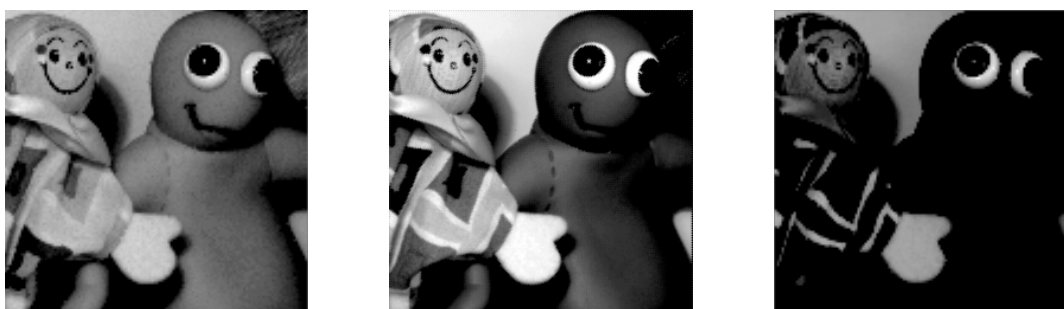


Figura 4.4 - CFA Interpolation: a) Canale Rosso; b) Canale Verde; c) Canale Blu

In Figura 4.4 sono mostrati le componenti RGB ottenute dopo l'*Interpolazione CFA*. Mentre in Figura 4.5 sono riportate le componenti YCrCb ottenute dopo la conversione dello spazio del colore.

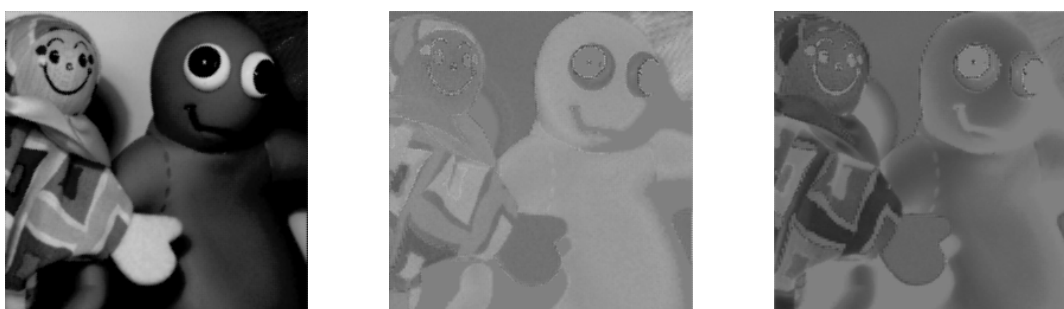


Figura 4.5 - Color Space Conversion: a) Canale Y; b) Canale Cr; c) Canale Cb

Il canale Y rappresenta la *Luminanza* dell'immagine e deriva direttamente dal canale G dello spazio RGB mentre i canali Cr e Cb, detti di *Crominanza*, derivano, rispettivamente, dalle componenti R e B dello spazio RGB.

Gli algoritmi implementati in questa tesi si occupano di correggere le imperfezioni apportate dai passi compiuti fino a questo punto. Tali imperfezioni non dipendono dagli algoritmi implementati dai colleghi ma bensì dalla manipolazione separata dei diversi canali coinvolti. I risultati ottenuti sono mostrati di seguito.

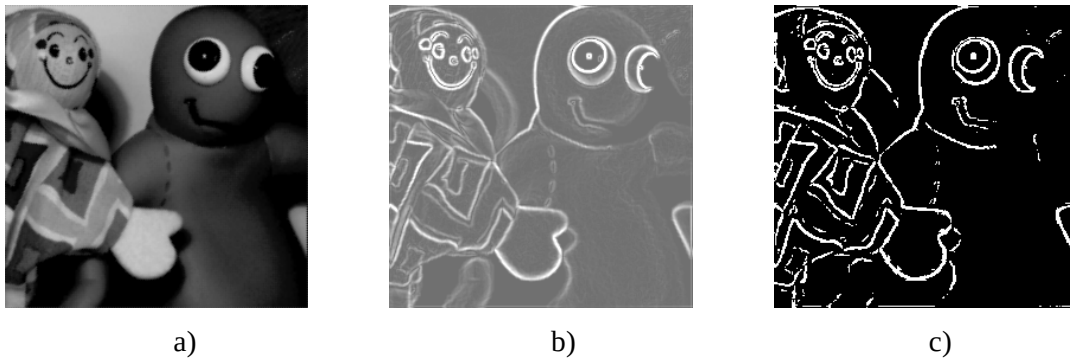


Figura 4.6 - a) Canale Y; b) *Edge Detection*; c) *Thresholding*

La fonte di partenza, per questa fase, è il canale di Luminanza ottenuto nel passaggio precedente. Applicando l'algoritmo di *Edge Detection* si ottiene l'immagine di Figura 4.6b. Va precisato che l'immagine è stata notevolmente schiarita per renderla visibile, ma, in realtà, i valori di convoluzione ottenuti sono molto bassi e quindi risultano un'immagine molto scura. Quest'ultima è applicata per la correzione diretta del canale di Luminanza. La Figura 4.7 è utilizzata per paragonare il risultato all'originale.

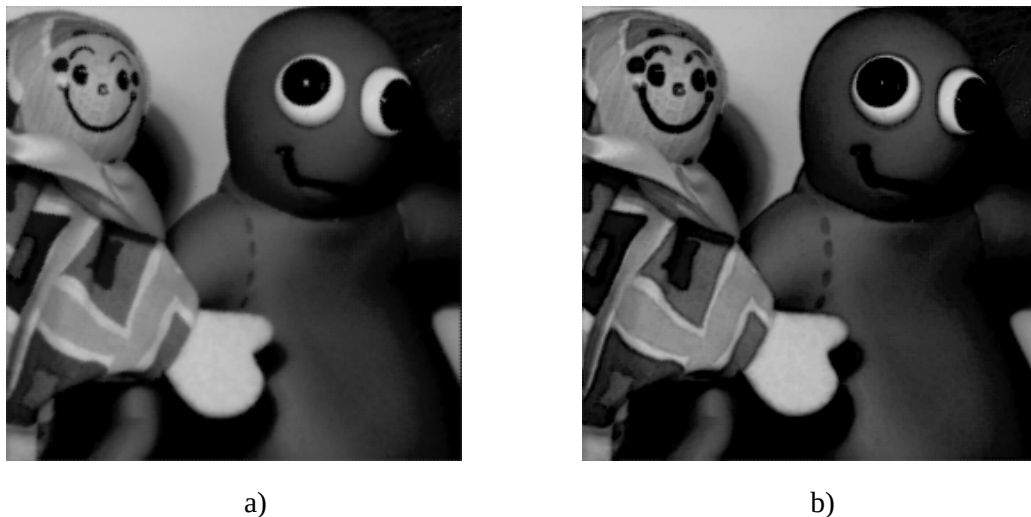


Figura 4.7 - a) Canale Y originale; b) Canale Y corretto

La differenza che salta all'occhio sono i bordi più marcati nella Figura 4.7b perché lo scopo della correzione è, appunto, diminuire l'attenuazione dei bordi dettata dall'*Interpolazione CFA*.

Il passo successivo prevede la correzione dei canali Cr e Cb utilizzando la maschera ottenuta dal *Thresholding* e mostrata in Figura 4.6c. I risultati sono mostrati di seguito.

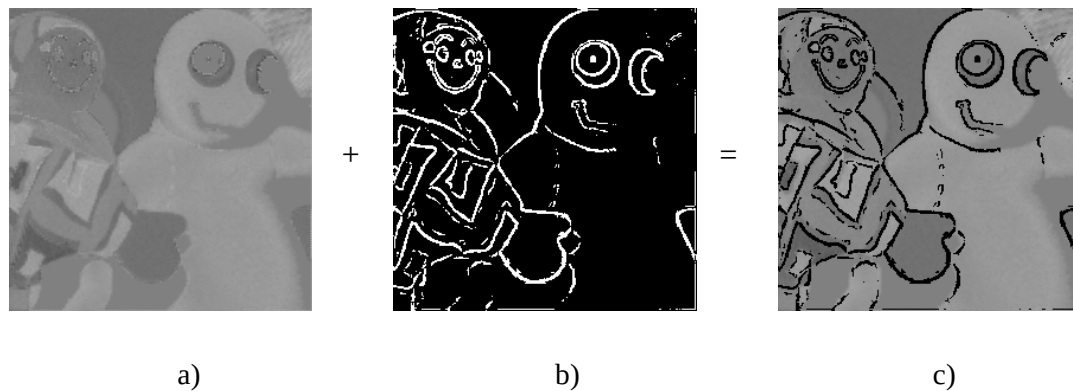


Figura 4.8 - Correzione cromatica: a) Canale Cr; b) Masc. Thresh.; c) Canale Cr corretto

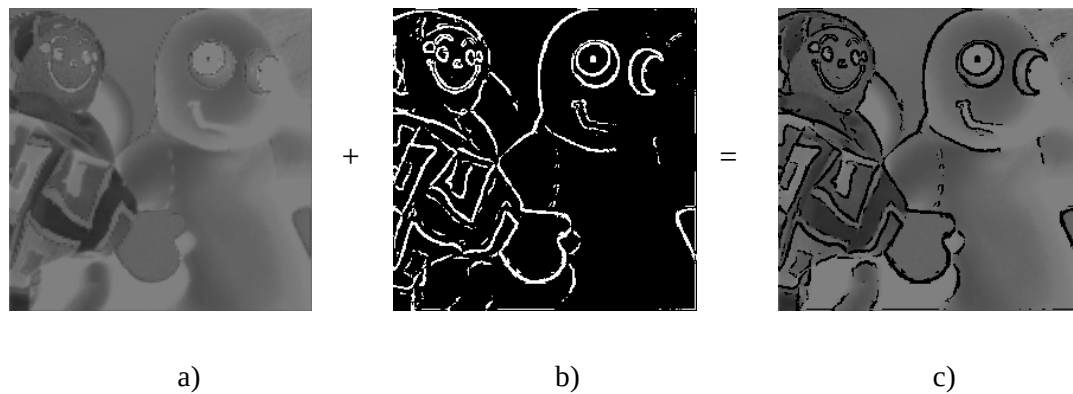


Figura 4.9 - Correzione cromatica: a) Canale Cb; b) Masc. Thresh.; c) Canale Cb corretto

Le correzioni sono effettuate sopprimendo semplicemente il valore cromatico dal canale relativo. In questo modo si eliminano i difetti di colore che compaiono a causa della correzione della *Luminanza*.

4.2 - Analisi Funzionale

L'approccio utilizzato per quest'analisi rispecchia, per coerenza, quella seguita nel capitolo precedente. Quindi, nell'ambito dell'analisi funzionale e dell'analisi di sistema, si tratteranno i dati ottenuti con PPE variabile separatamente da quelli ottenuti con PPE fissato

4.2.1 - Analisi con PPE variabile

In questo caso è interessante proporre i dati relativi ai gruppi funzionali confrontando assieme quelli appartenenti alle parti in gioco differenziandoli per valori di PPE.

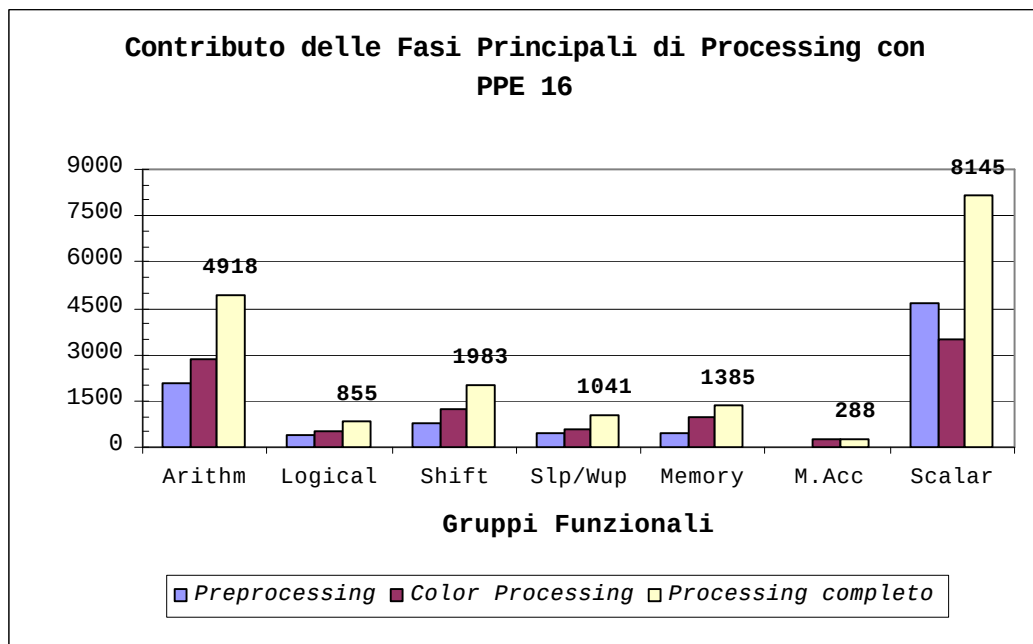


Grafico 4.1 - Gruppi funzionali relativi al Processing completo ed alle sue fasi principali

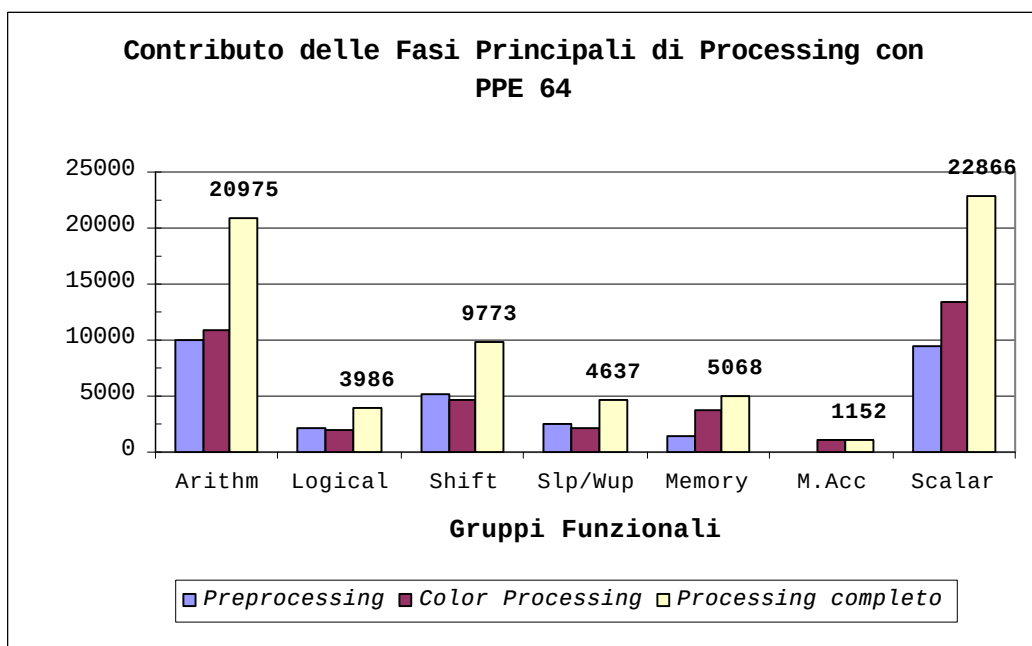


Grafico 4.2 - Gruppi funzionali relativi al Processing completo ed alle sue fasi principali

Nei grafici sono considerati solo gruppi funzionali non nulli e che godono di una certa rilevanza. Dal Grafico 4.1 possiamo notare come sia presente una certa proporzionalità tra le fasi principali del processo di elaborazione. In particolare possiamo vedere come le operazioni che coinvolgono l'accumulatore siano presenti solo nella fase di correzione cromatica e completamente assenti nella fase iniziale.

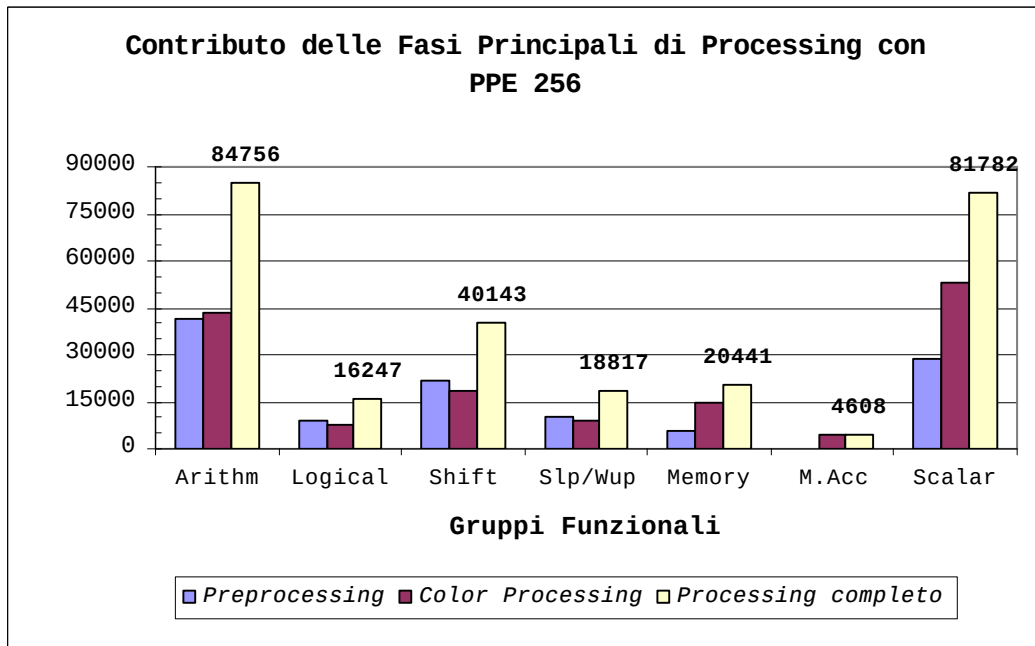


Grafico 4.3 - Gruppi funzionali relativi al Processing completo ed alle sue fasi principali

Ad un primo sguardo, i grafici proposti, nel loro complessivo, confermano il fatto che le fasi mantengano una certa proporzionalità, ma ad uno sguardo più attento non sfugge il fatto che si presenta un capovolgimento di fronte, seppur minimo. Infatti, con l'aumentare del PPE le operazioni di *Color Processing* aumentano più rapidamente rispetto a quelle di *Preprocessing*. Ciò perché le operazioni di interpolazione e di convoluzione effettuano molti calcoli matematici e gestiscono più cicli.

Unità funzionali	PPE 16 Img 512 x 512		PPE 64 Img 1024 x 1024		PPE 256 Img 2048 x 1536	
	Num istruz	(%)	Num istruz	(%)	Num istruz	(%)
<i>Nop</i>	558	2,56	2052	2,67	7968	2,70
<i>Arithmetic</i>	4918	22,53	20975	27,28	84756	28,68
<i>Logical</i>	855	3,92	3986	5,18	16247	5,50
<i>Shift</i>	1983	9,08	9773	12,71	40143	13,58
<i>Sleep</i>	1041	4,77	4637	6,03	18817	6,37
<i>Wakeup</i>	1041	4,77	4637	6,03	18817	6,37
<i>Communication</i>	1602	7,34	1666	2,17	1698	0,57
<i>Memory</i>	1385	6,34	5068	6,59	20441	6,92
<i>M.Accumulator</i>	288	1,32	1152	1,50	4608	1,56
<i>Detector</i>	17	0,08	65	0,08	257	0,09
<i>Scalar</i>	8145	37,31	22866	29,74	81782	27,67
Totale	21833	100,00	76877	100,00	295534	100,00

Tabella 4.1 - Gruppi funzionali del Processing completo al variare del PPE

Sempre a proposito delle operazioni aritmetiche e scalari, si osserva, dalla Tabella 4.1, che entrambe incidono in modo diverso a seconda della risoluzione dell'immagine: con un basso valore di PPE (immagine 512x512) si ha una leggera supremazia delle operazioni scalari (~ 37 % contro il 23 % circa delle aritmetiche). Questa differenza tende ad invertirsi con l'aumentare delle dimensioni dell'immagine, e quindi del valore di PPE fino a raggiungere la differenza, a favore delle operazioni aritmetiche, del 1 %.

Nella Tabella 9.2 sono riportati i valori relativi all'operazione di *Inizializzazione*.

Unità funzionali	PPE 16 Img 512 x 512		PPE 64 Img 1024 x 1024		PPE 256 Img 2048 x 1536	
	Num istruz	(%)	Num istruz	(%)	Num istruz	(%)
<i>Nop</i>	16389	5,61	16389	5,60	12293	5,52
<i>Arithmetic</i>	57597	19,72	57741	19,72	43933	19,72
<i>Logical</i>	2	0,00	2	0,00	2	0,00
<i>Shift</i>	1	0,00	1	0,00	1	0,00
<i>Sleep</i>	16384	5,61	16384	5,59	12288	5,52
<i>Wakeup</i>	16384	5,61	16384	5,59	12288	5,52
<i>Communication</i>	0	0,00	0	0,00	0	0,00
<i>Memory</i>	32823	11,24	32967	11,26	25351	11,38
<i>M.Accumulator</i>	0	0,00	0	0,00	0	0,00
<i>Detector</i>	51	0,02	195	0,07	771	0,35
<i>Scalar</i>	152507	52,20	152795	52,17	115851	52,00
Totale	292138	100,00	292858	100,00	222778	100,00

Tabella 4.2 - Gruppi funzionali relativi all'Inizializzazione con PPE variabile

Possiamo notare che alcuni valori coincidono al variare del PPE: *Nop* con PPE 16 e 64, *Sleep* e *Wakeup* con PPE 16 e 64, ecc. Mentre differiscono per altri gruppi. Questo è dovuto al fatto che sono stati eseguiti assieme l'operazione di *Raggruppamento*, che è sensibile solo al variare delle dimensioni dell'array di PE, e la memorizzazione locale delle maschere per il *Preprocessing* che, invece, dipende dal PPE.

4.2.2 - Analisi con PPE fissato

Ponendo il valore di PPE a 256 e mantenendolo costante, l'analisi si riduce a considerare array di processori di dimensioni variabili.

Durante la fase di prova è stato riscontrato che le operazioni di correzione cromatica non sono sensibili alla variazione delle dimensioni della matrice di processori, quindi mantenendo il valore di PPE pari a 256 otteniamo dei risultati costanti. Pertanto è sufficiente il Grafico 4.4 per riassumerne le entità e le percentuali di incidenza. Non si può dire lo stesso per la fase di *Preprocessing*.

La Tabella 4.3 rivela come queste operazioni risentano della variazione sia di PPE che delle dimensioni dell'array di PE. La spiegazione sta nel fatto che la funzione di bilanciamento del bianco deve determinare un massimo assoluto dell'intera immagine. Ciò vuol dire che il massimo valore registrato in ogni processore deve essere confrontato con gli altri e quindi si compiono delle operazioni di comunicazione. Il numero di operazioni compiute dipende dal numero di PE in gioco.

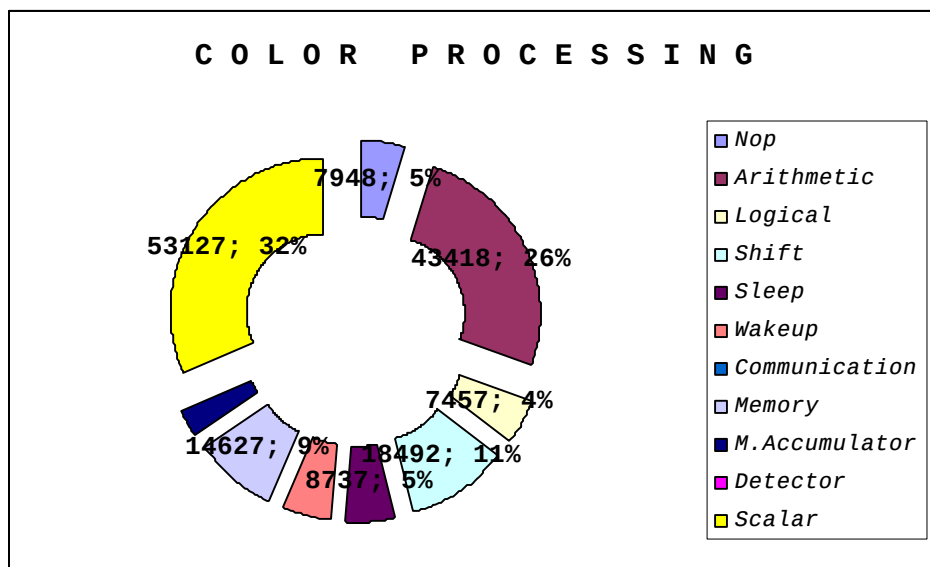


Grafico 4.4 - Gruppi funzionali relativi al Color Processing con PPE pari a 256

Unità funzionali	PE 32 x 32 Img 512 x 512		PE 64 x 64 Img 1024 x 1024		PE 128 x 96 Img 2048 x 1536	
	Num istruz	(%)	Num istruz	(%)	Num istruz	(%)
<i>Nop</i>	20	0,02	20	0,02	20	0,02
<i>Arithmetic</i>	39825	33,25	39837	32,90	41338	32,25
<i>Logical</i>	8508	7,10	8508	7,03	8790	6,86
<i>Shift</i>	20817	17,38	20817	17,19	21651	16,89
<i>Sleep</i>	9776	8,16	9782	8,08	10080	7,86
<i>Wakeup</i>	9776	8,16	9782	8,08	10080	7,86
<i>Communication</i>	444	0,37	828	0,68	1500	1,17
<i>Memory</i>	5345	4,46	5345	4,41	5806	4,53
<i>M.Accumulator</i>	0	0,00	0	0,00	0	0,00
<i>Detector</i>	257	0,21	257	0,21	257	0,20
<i>Scalar</i>	25024	20,89	25893	21,39	28663	22,36
Totale	119792	100,00	121069	100,00	128185	100,00

Tabella 4.3 - Gruppi funzionali del Preprocessing con PPE pari a 256

Quanto detto si ripercuote sull'intero processo dove, anche in questo caso, si registrano valori diversi al variare delle dimensioni dell'immagine elaborata. Nella Tabella 4.4 si mettono in luce i gruppi che incidono maggiormente nello sfruttamento delle risorse di calcolo. Sono le operazioni scalari ed aritmetiche (mediamente il 28 %) anche se le operazioni di trascinamento sono discretamente utilizzate (mediamente il 14 %).

La Tabella 4.4 viene proposta per dare un quadro generale dei valori ottenuti fornendo anche le percentuali di incidenza di ciascun gruppo funzionale.

Unità funzionali	PE 32 x 32 512 x 512 Img		PE 64 x 64 1024 x 1024 Img		PE 128 x 96 2048 x 1536 Img	
	Num istruz	(%)	Num istruz	(%)	Num istruz	(%)
<i>Nop</i>	7968	0,02	7968	0,02	7968	0,02
<i>Arithmetic</i>	83243	33,25	83255	32,90	84756	32,25
<i>Logical</i>	15965	7,10	15965	7,03	16247	6,86
<i>Shift</i>	39309	17,38	39309	17,19	40143	16,89
<i>Sleep</i>	18513	8,16	18519	8,08	18817	7,86
<i>Wakeup</i>	18513	8,16	18519	8,08	18817	7,86
<i>Communication</i>	642	0,37	1026	0,68	1698	1,17
<i>Memory</i>	19972	4,46	19972	4,41	20433	4,53
<i>M.Accumulator</i>	4608	0,00	4608	0,00	4608	0,00
<i>Detector</i>	257	0,21	257	0,21	257	0,20
<i>Scalar</i>	78151	20,89	79020	21,39	81790	22,36
Totale	287141	100,00	288418	100,00	295534	100,00

Tabella 4.4 - Gruppi funzionali del Processing completo con PPE pari a 256

Uno sguardo attento potrà accorgersi che le operazioni di *Detector*, preposte al caricamento delle immagini in memoria, sono utilizzate solo nella fase di *Preprocesso*. Ciò è normale dato che, una volta campionata, l'immagine resterà sempre a disposizione degli algoritmi nella memoria locale dei PE quindi, la fase di *Color Processing* non necessita di nessun altro campionamento.

La Tabella 4.5 riepiloga, invece, i valori inerenti alla fase di *Inizializzazione*. In essa si può notare lo stesso comportamento riscontrato nell'analisi a PPE variabile: alcuni gruppi funzionali riportano valori costanti altri no. La spiegazione è la stessa di quella data precedentemente: la fase di *Inizializzazione* consta di una parte esclusivamente dipendente dal valore di PPE e da un'altra parte dipendente, questa volta, dalle dimensioni dell'array di processori. L'unica differenza è che, i valori che risultano cambiare in Tabella 4.2 in questo caso, invece, risultano costanti e vice versa.

Unità funzionali	PE 32 x 32 Img 512 x 512		PE 64 x 64 Img 1024 x 1024		PE 128 x 96 Img 2048 x 1536	
	Num istruz	(%)	Num istruz	(%)	Num istruz	(%)
<i>Nop</i>	1029	4,61	4101	5,32	12293	5,52
<i>Arithmetic</i>	4413	19,76	15213	19,73	43933	19,72
<i>Logical</i>	2	0,01	2	0,00	2	0,00
<i>Shift</i>	1	0,00	1	0,00	1	0,00
<i>Sleep</i>	1024	4,59	4096	5,31	12288	5,52
<i>Wakeup</i>	1024	4,59	4096	5,31	12288	5,52
<i>Communication</i>	0	0,00	0	0,00	0	0,00
<i>Memory</i>	2823	12,64	8967	11,63	25351	11,38
<i>M.Accumulator</i>	0	0,00	0	0,00	0	0,00
<i>Detector</i>	771	3,45	771	1,00	771	0,35
<i>Scalar</i>	11243	50,35	39867	51,70	115851	52,00
Totale	22330	100,00	77114	100,00	222778	100,00

Tabella 4.5 - Gruppi funzionali relativi all'Inizializzazione con PPE pari a 256

Anche in questo caso, dal punto di vista funzionale, non è possibile confrontare i dati del testing con quelli del dispositivo campione in quanto, quest'ultimo, ha la caratteristica di essere programmabile e ciò rende vano qualsiasi confronto.

4.3 - Analisi delle prestazioni del Sistema

Per questa fase di analisi i risultati ottenuti sono suddivisi in tabelle ciascuna delle quali rappresenta una fase specifica dell'*Image Processing*. Coerentemente a quanto fatto finora, verranno proposti i risultati ottenuti sia a PPE variabile che a PPE fisso.

4.3.1 - Analisi con PPE variabile

Le tabelle di seguito riassumono i dati significativi relativi all'impiego delle risorse di sistema.

Dimens. Immagine	X_SIZE x Y_SIZE (Np)	Memoria		Cicli (N _i)	Utilizzo sistema	Tempo di esecuzione [μs]	Throughput
		PE [Byte]	System [KB]				
512 x 512	128 x 128 PPE 16	102	816	292138	0,310	584,276	2,54E+12
1024 x 1024	128 x 128 PPE 64	390	3120	292858	0,310	585,716	2,54E+12
2048 x 1536	128 x 96 PPE 256	1542	9252	222778	0,315	445,556	1,93E+12

Tabella 4.6 - Inizializzazione a PPE variabile

Per quanto riguarda la Tabella 4.6, il dato che balza all'occhio è il numero di cicli (N_i). Essendo dominato dalle operazioni di *Raggruppamento* è normale che tra la prima voce registrata (N_i = 292138) differisca di poco dalla seconda (N_i = 292858) e che ci si aspetti un valore maggiore per la terza. Ma così non accade. Ciò è dovuto dal fatto che, in realtà per l'immagine di dimensioni 2048 x 1536, si utilizzano meno processori (128 x 96) rispetto alle altre più piccole (128 x 128 PE). La conseguenza è la diminuzione di operazioni di raggruppamento e, quindi di cicli N_i

Dimens. Immagine	X_SIZE x Y_SIZE (Np)	Memoria		Cicli (N _i)	Utilizzo sistema	Tempo di esecuzione [μs]	Throughput
		PE [Byte]	System [KB]				
512 x 512	128 x 128 PPE 16	200	1600	10808	0,517	21,616	4,23E+12
1024 x 1024	128 x 128 PPE 64	712	5696	34496	0,642	68,992	5,26E+12
2048 x 1536	128 x 96 PPE 256	2696	16176	128185	0,663	256,370	4,07E+12

Tabella 4.7 - Preprocessing a PPE variabile

Dimens. Immagine	X_SIZE x Y_SIZE (Np)	Memoria		Cicli (Ni)	Utilizzo sistema	Tempo di esecuzione [μs]	Throughput
		PE [Byte]	System [KB]				
512 x 512	128 x 128 PPE 16	322	2576	283231	0,342	566,462	2,80E+12
1024 x 1024	128 x 128 PPE 64	1122	8976	260043	0,427	520,086	3,50E+12
2048 x 1536	128 x 96 PPE 256	4258	25548	167347	0,573	334,694	3,52E+12

Tabella 4.8 - Color Processing a PPE variabile

Dimens. Immagine	X_SIZE x Y_SIZE (Np)	Memoria		Cicli (Ni)	Utilizzo sistema	Tempo di esecuzione [μs]	Throughput
		PE [Byte]	System [KB]				
512 x 512	128 x 128 PPE 16	418	3344	294039	0,348	588,078	2,85E+12
1024 x 1024	128 x 128 PPE 64	1506	12048	294539	0,452	589,078	3,70E+12
2048 x 1536	128 x 96 PPE 256	5794	34764	295532	0,612	591,064	3,76E+12

Tabella 4.9 - Processing completo a PPE variabile

I valori riportati sono stati calcolati utilizzando le espressioni degli indici descritti nel capitolo precedente.

Possiamo notare come ci sia, in Tabella 4.9, una netta differenza tra i tempi di esecuzione. Questa divergenza è stata ereditata dagli algoritmi implementati, infatti la variazione dei valori è riscontrata e giustificata anche nel capitolo precedente. Ciò nonostante i tempi di esecuzione dell'intero processo si attestano su valori praticamente costanti: ~ 590 μs.

Un dato importante è il rapporto Cicli/pixel. Esso è sintetizzato nel Grafico 4.5 in cui possiamo individuare la fase di *Inizializzazione*, le fasi principali e quella completa dell'*Image Processing*

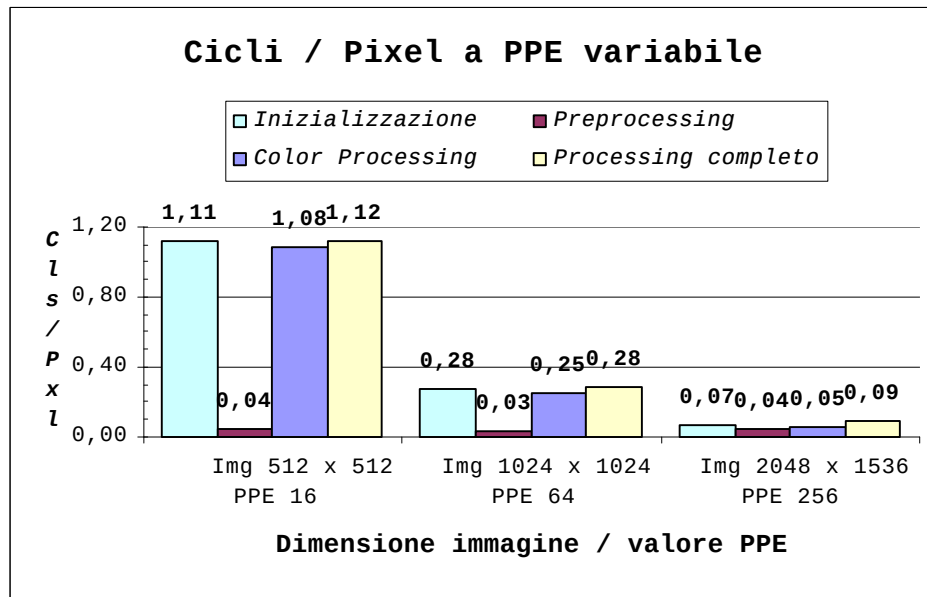


Grafico 4.5 - Cicli/Pixel delle fasi di elaborazione in funzione del PPE

Dal grafico appena mostrato possiamo notare come ci sia una differenza tra i valori registrati in fase di *Preprocessing* e quelli delle altre fasi.

4.3.2 - Analisi con PPE fissato

Fissando il valore di PPE a 256, i valori relativi alle risorse di sistema richiesti sono riassunti nelle tabelle seguenti.

Dimens. Immagine	X_SIZE x Y_SIZE (Np)	Memoria		Cicli (Ni)	Utilizzo sistema	Tempo di esecuzione [μs]	Throughput
		PE [Byte]	System [KB]				
512 x 512	32 x 32	1542	771	22330	0,359	44,660	1,84E+11
1024 x 1024	64 x 64	1542	3084	77114	0,324	154,228	6,63E+11
2048 x 1536	128 x 96	1542	9252	222778	0,315	445,556	1,93E+12

Tabella 4.10 - Inizializzazione con PPE pari a 256

Dimens. Immagine	X_SIZE x Y_SIZE (Np)	Memoria		Cicli (Ni)	Utilizzo sistema	Tempo di esecuzione [μs]	Throughput
		PE [Byte]	System [KB]				
512 x 512	32 x 32	2696	1348	119792	0,687	239,584	3,52E+11
1024 x 1024	64 x 64	2696	5392	120998	0,684	241,996	1,40E+12
2048 x 1536	128 x 96	2696	16176	128185	0,663	256,370	4,07E+12

Tabella 4.11 - Preprocessing con PPE pari a 256

Dimens. Immagine	X_SIZE x Y_SIZE (Np)	Memoria		Cicli (Ni)	Utilizzo sistema	Tempo di esecuzione [μs]	Throughput
		PE [Byte]	System [KB]				
512 x 512	32 x 32	4238	2119	167347	0,621	334,694	3,18E+11
1024 x 1024	64 x 64	4238	8476	167347	0,622	334,694	1,27E+12
2048 x 1536	128 x 96	4238	25428	167347	0,622	334,694	3,82E+12

Tabella 4.12 - Color Processing con PPE pari a 256

Il diverso comportamento delle due fasi riscontrato durante l'analisi funzionale a PPE fissato può essere riscontrato anche nell'analisi di sistema. Infatti, le Tabelle 4.11-12 mettono in evidenza come resti immutato il numero di cicli N_i per la fase di correzione cromatica ma varia in fase di *Preprocessing*. Ciò dipende dall'operazione di bilanciamento del colore.

Dimens. Immagine	X_SIZE x Y_SIZE (Np)	Memoria		Cicli (Ni)	Utilizzo sistema	Tempo di esecuzione [μs]	Throughput
		PE [Byte]	System [KB]				
512 x 512	32 x 32	2696	1348	287139	0,649	574,278	3,32E+11
1024 x 1024	64 x 64	2696	5392	288345	0,648	576,690	1,33E+12
2048 x 1536	128 x 96	2696	16176	295532	0,640	591,064	3,93E+12

Tabella 4.13 - Processing completo con PPE pari a 256

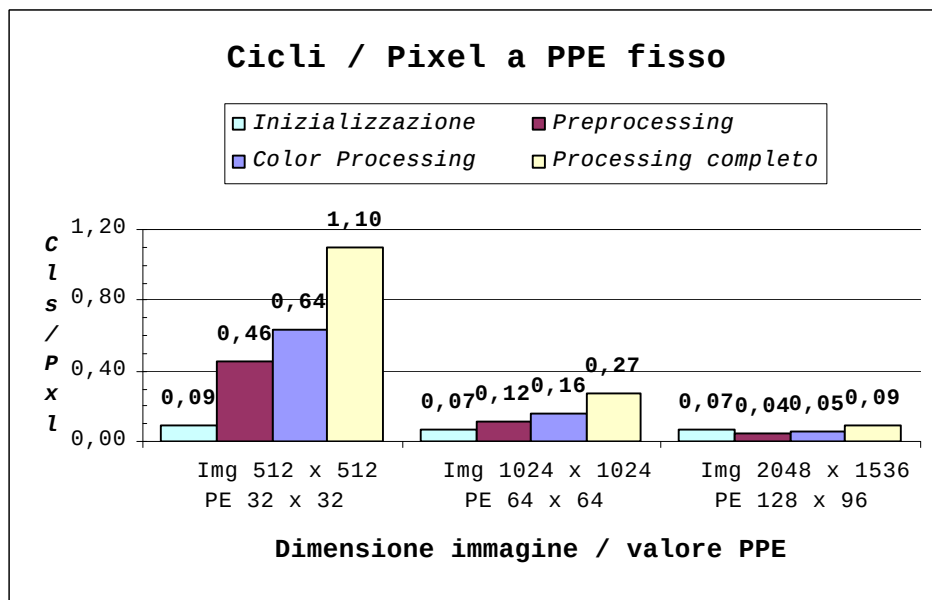


Grafico 4.6 - Cicli/Pixel delle fasi di elaborazione con PPE pari a 256

Quanto detto in precedenza si ripercuote sui risultati ottenuti riguardo all'intero processo di elaborazione. Tuttavia non si registrano sostanziali differenze: l'utilizzo del sistema rimane pressoché invariato mentre il tempo di esecuzione risulta essere sempre al di sopra dei 570 μ s.

Il Grafico 4.6 fornisce un dato molto importante: il numero di cicli che vengono eseguiti per ogni pixel. Si possono notare valore elevati per immagini relativamente piccole, ma per immagini molto più grandi, i valori diminuiscono drasticamente. Questo è un fatto molto importante perché vuol dire che più le immagini sono grandi, più veloce viene eseguita l'elaborazione. Ciò non rappresenta un controsenso perché siamo in presenza di un ambiente multiprocessore dove il lavoro viene sostanzialmente suddiviso tra tutti i processori coinvolti con notevoli vantaggi di tempo. Il fatto che i valori diminuiscano all'aumentare del numero di PE ne è la prova.

5. Valutazioni e Conclusioni

Lo studio condotto fino ad ora ci ha portato a compiere delle scelte che hanno prodotto dei risultati. Questi sono stati espressi anche in forma numerica per poter valutare obiettivamente le prestazioni degli algoritmi implementati in questa tesi e quelli implementati precedentemente dai miei colleghi. È utile, se non indispensabile, confrontare queste informazioni con quelle di dispositivi già collaudati e presenti sul mercato. A tale scopo è stata considerata una macchina fotografica digitale della Texas Instruments: TMS320C54x.

<i>TASK</i>	<i>Cicli/Pixel</i>
Preprocessing (Black Clamping, White Balance, Gamma Correction)	22
Interpolation	41
Color Space Conversion	10
Edge Detection, False Color Correction	27
JPEG Compression	62
TOTALE	152

Tabella 5.1 - Prestazioni del DSP C549

La Tabella 5.1 riassume i valori del rapporto Cicli/Pixel suddivisi per le fasi di elaborazione. Fatta eccezione per la compressione JPEG le parti in causa sono già state descritte nei capitoli precedenti.

Dalla Tabella precedente si può notare che per eseguire l'*Edge Detection* e *False Color Correction*, il dispositivo americano deve compiere 27 Cicli/Pixel per il dispositivo della Texas Instruments (TMS320C54x). Si noti, però, che i pixel elaborati sono 1 milione. Ciò equivale ad elaborare un'immagine di dimensione 1000x1000. Siccome viene utilizzato un solo processore e la

frequenza di clock è di 100 MHz, il tempo di esecuzione, per un'immagine si fatta è di ~ 270 ms. L'architettura SIMPil è basata su un tempo di clock di 500Mhz, quindi, esegue le stesse operazioni su un'immagine di dimensione 1024x1024 suddivisa in un array di PE di dimensioni 128 x 128 e PPE pari a 64, con 0,03 Cicli/Pixel (Grafico 3.4), e cioè in ~ 63 μ s: una differenza di ben quattro grandezze. Se volessimo considerare, invece, un'array di 64 x 64 PE e PPE pari a 256 ci dovremmo accontentare di 0,13 Cicli/pixel (Grafico 3.5) corrispondenti a ~ 273 μ s. Un ottimo risultato considerando il fatto che l'ambiente SIMPil elabora un immagine con circa 50.000 pixel in più. I dati appena calcolati sono riassunti nella Tabella 5.2.

Dimensione immagine	Correzione del colore	Preprocessing	Color Processing	Processo completo
TMS320C54x (TI) 1000 x 1000	27 C/P 270 ms	22 C/P 220 ms	78 C/P 780 ms	100 C/P 1 sec
SIMPil 1024x1024 128x128 PPE 64	0,03 C/P 63 μ s	0,03 C/P 63 μ s	0,25 C/P 524 μ s	0,28 C/P 587 μ s
SIMPil 1024x1024 64x64 PPE 256	0,13 C/P 273 μ s	0,12 C/P 252 μ s	0,16 C/P 335 μ s	0,28 C/P 587 μ s

Tabella 5.2 - Riepilogo delle valutazioni delle prestazioni

Valutiamo, adesso, le prestazioni dal punto di vista dell'intero processo di elaborazione distinguendo solamente le due parti principali. Sempre dalla Tabella 5.1 si può notare come, per il dispositivo della Texas Instruments (TMS320C54x), la parte del *Preprocessing*, consti di 22 Cicli/Pixel e quella del *Color Processing* di 78 Cicli/Pixel (100 Cicli/Pixel per il processo completo) ottenendo, come tempi di elaborazione, rispettivamente 220 ms e 780 ms. Dal Grafico 4.5 ricaviamo che per l'ambiente SIMPil occorrono 0.03 Cicli/Pixel per il *Preprocessing* e 0,25 Cicli/Pixel per il *Color Processing* (0,28 Cicli/Pixel per il processo completo) considerando sempre

un'immagine 1024 x 1024 suddivisa in un array di 128 x 128 PE con PPE pari a 64. I tempi di esecuzione sono riportati in Tabella 5.2 insieme a quelli relativi all'array di 64 x 64 processori con PPE pari a 256.

Da subito si nota come i tempi di elaborazione richiesti dal SIMPil risultino sempre nell'ordine dei micro secondi. Questa soglia non è mai raggiunta dal dispositivo della Texas Instruments che non riesce a completare un'intera elaborazione al di sotto del secondo. Situazione ben diversa per l'architettura SIMPil che raggiunge il suo scopo dopo solo 587 μ s.

Paradossalmente per immagini via via più piccole il dato aumenta. Ciò è dovuto al fatto che il numero di pixel diminuisce più rapidamente rispetto ai cicli necessari per portare a termine l'elaborazione. Ma se consideriamo un'immagine di 2048 x 1536 bastano solo 0,09 Cicli/Pixel arriviamo ad un tempo di esecuzione pari a 189 μ s.

Osservando più attentamente i Grafici 4.5-6 notiamo che il processo completo è portato a termine sempre con gli stessi cicli per pixel, e, cioè, con gli stessi tempi di elaborazione. Di conseguenza è influente organizzare l'ambiente SIMPil con un alto valore di PPE per risparmiare sul numero di processori, o vice versa. Il risultato sarà raggiunto allo stesso tempo.

In conclusione, già in fase di testing, e dalla documentazione fornita, risulta evidente che l'architettura SIMPil è, di gran lunga, più performante rispetto al TMS320C54x della Texas Instruments.

Bibliografia

- [1] Application report, **TMS320DSC21 - A high performance, programmable, single chip digital signal processing to digital still cameras**, *Texas Instruments Inc., Dallas, Texas*.
- [2] W. Ragadi, R. Talluri, K. Illgner, J. Liang, Y. Yoo, **Programmable DSP platform for digital still cameras**, *Texas Instruments Inc., Application Report, SPRA651 - April 2000*.
- [3] M. Mancuso, **An Introduction to the Digital Still Camera Technology**, *DSC & MM STMicroelectronics* (massimo.mancuso@st.com)
- [4] Andrea de Prisco, **I sensori CCD e l'acquisizione digitale delle immagini a colori**, Estratto da: <http://www.mclink.it/personal/MC0258/CCD.html>
- [5] Rajeev Raman, **Image Processing Data Flow in Digital Cameras**, Published by - *SPIE - The International Society for Optical Engineering, Volume 3302, 28 - 28 January 1998..*
- [6] Edge Detectors: <http://www.dai.ed.ac.uk/HIPR2/edgdetct.htm>
- [7] Compass Edge Detector: <http://www.dai.ed.ac.uk/HIPR2/prewitt.htm>
- [8] Sobel Edge Detector: <http://www.dai.ed.ac.uk/HIPR2/sobel.htm>
- [9] Roberts Cross Edge Detector: <http://www.dai.ed.ac.uk/HIPR2/roberts.htm>
- [10] Intensity histogram: <http://www.dai.ed.ac.uk/HIPR2/histogram.htm>
- [11] Thresholding: <http://www.dai.ed.ac.uk/HIPR2/threshld.htm>
- [12] Adaptative Thresholding: <http://www.dai.ed.ac.uk/HIPR2/adpthrsh.htm>
- [13] **Architetture parallele e Classificazione di Flynn:**
<http://www.multimedia.polito.it/sist-elab-2/aa2001/modulo1/contenuti/materiale-vario/parall-intro.pdf>
- [14] A. Gentile, D.S. Wills, J.L. Cruz-Rivera, F. Sorbello, **Real-Time, low level image processing on SIMPil - an embedded SIMD architecture**, in *AIIA Notizie, Processing*

of the Italian Association for Artificial Intelligence (AI*IA), 15(4), pages 40-43, February 2002, also appeared in *Proceedings of the Seventh Conference of the Italian Association for Artificial Intelligence (AI*IA)*, Bari, Italy, September 2001.

- [15] A. Gentile, A Lopez, Sek M. Chai, H. H. Cat, D. S. Wills, **Portable Multimedia Supercomputing**, Technical Report PICA Research Group, Georgia Institute of Technology, Atlanta, Georgia
- [16] A. Gentile, D. S. Wills, F. Sorbello, **Real-Time, Low Level Image Processing on SIMPil – an Embedded SIMD Architecture**, Technical Report PICA Research Group, Georgia Institute of Technology, Atlanta, Georgia.
- [17] SIMPil Home Page: <http://www.ece.gatech.edu/research/pica/simpil>
- [18] R. Kremens, **Image processing system application: from barbie cams to the space telescope**, Ph.D Rochester Institute for Technology Center for Imaging Science Digital Imaging and Remote Sensing Group and Pixelphysics, Inc., May 2001
- [19] Klaus Illgner, Hans-Georg Gruber, Pedro Gelabert, Jie Liang, Youngjun Yoo, Wissam Rabadi, Raj Talluri, **Programmable DSP Platform for Digital Still Cameras**, Texas Instrument Application Report SPRA 651- April 2000
- [20] Costantine N. Anagnostopoulos, Michel P. Lesser, Cameras and Systems for Electronic Photography and Scientific Imaging, *Proceedings SPIE*, February 1995, San Jose, California
- [21] Charles Poynton, **Basic Principles**, A Technical Introduction to Digital Video, John Wiley & Sons, 1996.
- [22] Prof. A. El Gamal, Introduction to Color Imaging, *Lecture Notes 10, EE392B*
- [23] Texas Instruments official TMS320C54x www site:

<http://www.ti.com/sc/docs/products/c5000/c54x/index.htm>
- [24] H. H. Cat, A. Gentile, J. C. Eble, M. Lee, O. Vendier, Y. J. Joo, D. S. Wills, M. Brooke, N. M. Jokerst, A. S. Brown, and R. Leavitt, **“SIMPil: An OE Integrated**

SIMD Architecture for Focal Plane Processing Applications”, *Massively Parallel Processing using Optical Interconnection (MPPOI-96)*, 1996.

- [25] A. Gentile, H. H. Cat, D. S. Wills: “**The SIMD Pixel Processor (SIMPil): SIMPILSimInstruction-Level Simulator for Windows.**” *Technical Report PICA Research Group, Georgia Institute of Technology, Atlanta, Georgia* 1997.
- [26] M. Uchiyama, F. Itoh, M. Ohki: “A Digital Still Camera”, Department of Electronic Engineering and Computer Science, Yamanashi University, Kofu, Japan, *IEEE Transactions On Consumer Electronics*, Vol 38, No. 3, August 1992
- [27] S. Okada, Ken’ichi Klyozaki: “**System on a chip for Digital Still Camera**” *SANYO Electric Co., ltd. Microelectronics Research Center 180 Ohmori, Anpachi-Cho, Anpachn , Gun, Gifu, 503-0195, Japan*
- [28] Wen-Hsin Chan, Ching-Twu Youe: “**Video CCD Based Portable Digital Still camera**”, *Industrial Technology research Institute ITRI/oes, 195-8, Chung-Hsing Rd. sec. 4 Chutung 310, Taiwan, ROC*