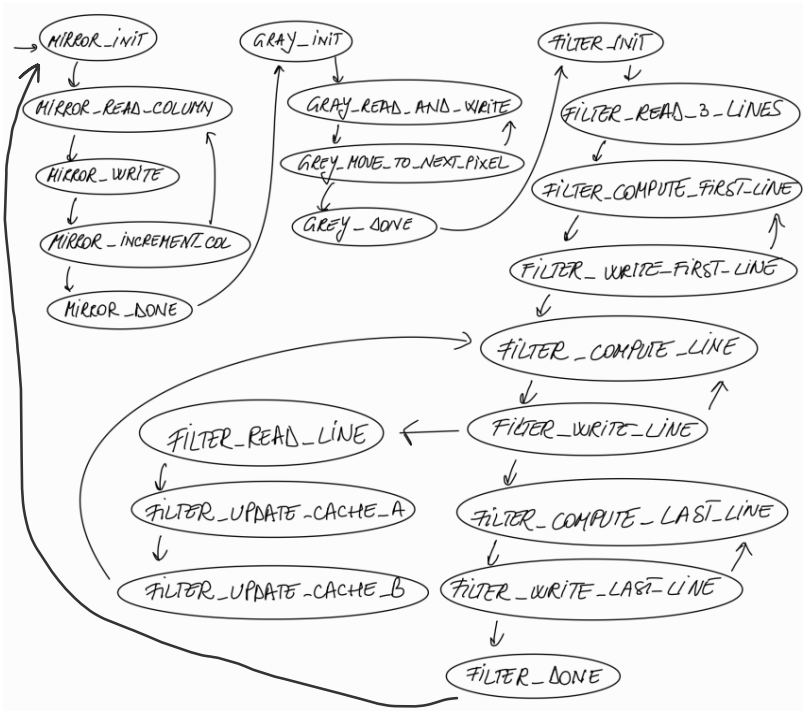
Colcer Alexia Ioana – 334AA

README Tema1



O imagine este o matrice 64×64, asupra careia facem 3 operatii: mirror (oglindire), grayscale, sharpness.

Avem 3 operatii majore (mirror, grayscale, sharpness) de aplicat, fiecare operatie avand starile ei. Finite State Machine (FSM) va avea urmatoarele stari (dupa cum se observa si din imaginea atasata):

- eventuale initializari pentru mirror <-- mirror

- citire coloana0, scriere coloana 0, ..., citire coloana63, scriere coloana 63

- setat MIRROR_DONE timp de un ciclu de ceas

- eventuale initializari pentru grayscale <-- grayscale

- citire pixel curent, calculeaza grayscale, scrie grayscale (pentru pixelul [0][0])

...

- citire pixel curent, calculeaza grayscale, scrie grayscale (pentru pixelul [63][63])

- setat GRAY_DONE timp de un ciclu de ceas

- eventuale initializari pentru sharpness <-- sharpness

- citire primele 3 linii, si cache-uirea lor

- calculat convolutia pentru prima linie

- calculat convolutia pentru a doua linie

- citire a patra linie, cache-uirea ei, calculat si scris produsele de convolutie cu centrul pentru elementele din a treia linie

...

- citire ultima linie, calculat si scris produsele de convolutie cu centrul in penultima linie

- calculat sharpness pentru elementele cu centrul in ultima linie

- setat SHARPNESS_DONE timp de un ciclu de ceas

Fiindca dupa fiecare operatie trebuie ca "semnalele *_done trebuie pastrate pe 1 cel putin un ciclu de ceas":

- intai aplicam prima operatie (mirror) asupra intregii poze, apoi a doua (grayscale), apoi a treia (sharpness)

- nu putem aplica o operatie pana ce operatia precedenta nu a fost aplicata in totalitate

Fiindca nu avem voie sa stocam mai mult de 6 randuri (poza avand 64 linii):

- nu putem citi o singura data un pixel (sau cateva linii) si sa calculam imediat toate cele 3 valori pentru el (mirror, grayscale, sharpness)

- trebuie sa citim valoarea fiecarui pixel de 3 ori (odata din poza initiala, apoi din poza oglindita, apoi din poza grayscale)

Colcer Alexia Ioana – 334AA

Avem nevoie 6 linii de cache, 3 pentru a stoca date si 3 pentru a le manipula pe celelalte. Pentru stocat date:

- o coloana (64 pixeli cache-uiti) o vom folosi pentru oglindire <-- folosim una din liniile cache de la sharpness

- 3 linii (3 * 64 pixeli cache-uiti) le vom folosi pentru a stoca datele necesare operatiei sharpness/produs de convolutie

Pentru a manipula liniile de cache, la convolutie: vrem ca line_1 sa fie cu 2 linii deasupra, line_2 sa fie linia de deasupra, line_3 linia curenta.

- cand terminam de citit o noua linie curenta, vrem sa updatam (line_1, line_2, line_3) <- (line_1, line_2, line_3)

- nu putem copia linii in o singura stare/tranztitie, ne trebuie 2 stari, deci 3 linii de cache auxiliare

- in total 6 linii: 3 pt stocat informatie, si inca 3 pentru a putea sa le updatam pe primele 3

Algoritm mirror

Pentru coloana curenta, la pozitia (a, b), putem scrie rezultatul final pentru pozitia (63 - a, b).

- vom citi o coloana intreaga (P[a][b] pentru toti b = 0, .., 63 pentru fiecare a = 0, .., 63), cate o celula la fiecare tick ascendent de ceas

- pentru aceasta coloana, pentru pixelul curent P[a][b], vom scrie P[63 - a][b], cate o celula la fiecare tick ascendent de ceas

- din starea "citeste coloana x" trecem succesiv in "citeste coloana x" pana terminam coloana, apoi trecem in "scrie coloana x"

- din starea "scrie coloana x" trecem succesiv in "scrie coloana x" pana terminam coloana

- apoi, cand am terminat de scris coloana x, trecem in "citeste coloana 1+x" sau, la final (x == 63), in "seteaza mirror_done pentru 1 ciclu de ceas"

Stari necesare:

- eventuale initializari

- citeste coloana (pentru coloana 0, pe care o citim pixel cu pixel)

- scrie coloana (pentru coloana 0, pe care o scriem pixel cu pixel)

....

- citeste coloana (pentru coloana 63)

- scrie coloana (pentru coloana 63)

- seteaza mirror_done pentru 1 ciclu de ceas

Algoritm grayscale

Citim imaginea pixel cu pixel, mergand la urmatorul pixel cu fiecare schimbare de stare:

- imaginea (matricea) va fi astfel parcurs top -> down (liniile) si pentru fiecare linie left -> right (coloanele)

- pixelul curent este compus din 3 bytes Pixel=abc; valoarea sa finala va fi PixelGrayscale=0medie0, unde medie = [max(a,b,c) + min(a,b,c)] / 2

- cand calculam valoarea grayscale a pixelului curent, o si scriem in rezultat, la urmatoarea schimbare de stare

Dorim sa calculam valoarea grayscale, facand o singura atribuire registrelor (max, min, rezultat).

Stari necesare:

- eventuale initializari

- citeste pixel curent, calculeazai grayscale si scriei valoarea = in aceeasi stare, deoarece nu avem nevoie de informatie in plux

- mergi la pixelul urmator

- seteaza grayscale_done pentru 1 ciclu de ceas

Algoritm filter/sharpness

Definitie: coltul stanga sus al unei matrice = matrice[0][0]; coltul dreapta jos = matrice[N-1][N-1], cand matricea are N linii.

a) citim matricea linie cu linie, stanga sus -> dreapta jos, si o cache-uim. Tinem mereu minte ultimele 3 linii citite.

- cum avem 3 linii (cele mai de sus 3), putem calcula rezultatul pentru cele mai de sus 2 linii. Il calculam si il scriem.

b) repetat, citim linia urmatoare si o cache-uim.

- cum am terminat de citit linia curenta, putem calcula si scrie rezultatul pentru linia anterioara

c) dupa ce am ajuns la ultima linie, si am calculat si scris raspunsul pentru penultima linie:

- calculam raspunsul pentru matricea 3x3 centrata in fiecare element al ultimei linii

- acesta este un caz special, ce are starea sa in AFS (FILTER_COMPUTE_LAST_LINE)

Stari necesare:

- eventuale initializari

- citeste primele 3 linii

- calculeaza si scrie raspunsurile la primele 2 linii

- citeste o noua linie

- calculeaza si scrie raspunsul pentru predecesoarea ei (pentru matricile 3x3 cu coltul dreapta jos in acea linie)

- calculeaza, pentru ultima linie, raspunsul pentru matricile 3x3 cu centrele in acea linie

- seteaza sharpness_done pentru 1 ciclu de ceas

Algoritm pastrare *_done timp de minim 1 ciclu ceas

Daca setam mirror_done, gray_done sau filter_done, tot ce ramane de facut este sa nu mai facem alte procesari in acelasi ciclu de ceas.

Colcer Alexia Ioana – 334AA

Algoritm calculare produs de convolutie

Suntem nevoiti sa cunoastem cei 8 vecini (respectiv 3 sau 5 daca actualul pixel este la margine):

- vom citi cate 3 randuri, si le vom tine minte (avem voie sa cache-uim pana la 6 randuri)

- vom calcula produsul de convolutie abia cand cunoastem vecinii

Este eficient sa nu avem 8 verificari (pentru fiecare vecin), daca vecinul respectiv este in afara pozei:

- cand un pixel este in mijlocul matricei, citesc cei 6 vecini si produsul de convolutie este trivial.

- cand acel pixel este la marginea matricei (pe o muchie sau un colt), ii inlocuiesc vecinii inexistenti cu 0.

(a, b, c) (-1, -1, -1)

(d, e, f) X (-1, 9, -1) = 9e - a - b - c - d - f - g - h - i

(g, h, i) (-1, -1, -1)

Extragem din pixelii matricei doar partea Grey (in rest oricum au bitii 0).

- cand facem produsul de convolutie, ne trebuie registre de dimensiune mare, datorita overflowului

- cand am calculat rezultatul final, daca e mai mare decat 255, vom seta bitii Grey ai rezultatului la 255.

- daca nu este > 255, setam cei mai putini semnificativi, facem astfel o trunchiere.

Algoritm updatare a celor 3 linii cache-uite, necesare filtrului de sharpness

- cand citim un pixel: scriem fiecare pixel citit in "cache_3", la pozitia corespunzatoare ("col" citita -> "cache_3[col]")

- cand am ajuns la sfarsit de rand (row == 63), am mai citit o linie de cache:

 - vrem sa copiem "cache_2" in "cache_1"

 - vrem sa copiem "cache_3" in "cache_2"

 - vrem sa copiem "cache_4" in "cache_4"

- dar nu este posibil sa facem aceasta in o singura stare/tranzitie; daca este sa repetam acceeasi tranzitie inaintea unui tick the ceas, vom suprascrie cach-ul

- solutie: avem 6 linii cache si 2 stari de manipulare a cache-ului

 - starea A: copiem liniile de cache (2, 3, 4) in liniile (6, 5, 4) <- linia 4 ramane nechimbata; putem face asta de multe ori in aceeasi state

 - starea B: copiem liniile (6, 5, 4) in liniile (1, 2, 3) <- este in siguranta daca facem asta de mai multe ori in aceeasi stare

Acest program este sintetizabil

- updateaza starea curenta in un bloc secvential, precum si linia si coloana curenta, pe front crescator al semnalului de ceas

- calculeaza starea viitoare, si parametrii de iesire, intr-un bloc combinational, la orice modificare de semnal

- nu foloseste constructii Verilog nesintetizabile (while, repeat, for cu numar variabil de iteratii, etc.)