Rusu Diana-Elena

Arhitectura Calculatoarelor

Tema a cuprins 4 task-uri, iar ca să le pot implementa am avut nevoie de realizarea unui automat finit cu 18 stări pe 5 biți fiecare pentru a putea numerota fiecare stare. Am folosit diferite stări pentru citirea elementelor(cu setarea lui row și col), rezolvarea task-urilor și scrierea lor în out_pix, care reprezintă pixelul de ieșire.

Pentru realizarea temei, pe lângă cele 18 stări am avut nevoie si de variabile interne care să mă ajute în reținerea anumitor valori pentru fiecare task în parte.

Prima problemă întâmpinată a fost la primul task, aceea că intram în timeout, acest lucru se întâmpla din cauză că aveam un loop infinit, prin atribuirea blocantă în blocul combinațional a lui row și col. Prin urmare, pe lângă atribuirea continuă a lui ”state” în partea secvențială, am adăugat o construcție de control ”case”, cum aveam și în partea combinațională pentru a putea incrementa și atribui noi valori lui row și col, am adăugat doar stările în care schimbăm valorile celor două variabile(row și col). Iar pentru a funcționa programul am făcut toate schimbările de care aveam nevoie cu un ciclu de ceas înainte, pentru că aceasta funcționează ca un ”next_state”, deci însemna că toate valorile s-ar fi schimbat abia după ce s-a terminat starea respectivă și a început următoarea (mai exact, starea în care aveam nevoie de valorile schimbate).

La primul task am stările (S0, S1, S2, S3 și S4). Starea de start începe tot procesul, așa că am avut nevoie aici să inițializez toate variabilele de care aveam nevoie cu 0. Ce am făcut mai exact a fost să interschimb primul element de pe o coloana cu ultimul element de pe aceeși coloana, iar prin folosirea blocului ”always” am putut face acest proces repetat (crescând numărul de randuri, iar în momentul în care ajunge la jumătatea lor, se incrementează numărul de coloane) pana la o conditie de oprire, aceea fiind când ajungem la ultima coloană și ajungem la jumătate din rânduri, deoarece noi lucrăm doar până la jumătatea matricei, deoarece noi interschimbăm valorile de la poziția (i,j) cu (63-i,j) și dacă am fi mers până la final cu numărul de rânduri am fi inversat-o la loc.

M-am folosit de S1 pentru a citi elementul de pe poziție (row, col), după aceea de S2 pentru a citi elementul de pe poziția (63-row,col), am interschimbat pixelii și am scris out_pix pe poziția (63-row,col), pentru a putea scrie pe poziția (row, col), am utilizat starea S3, iar de acolo am putut incrementa pentru a pune și condiția de finalizare a matricei și trecerea la starea S4, care era realizată pentru ”mirror_done”, deoarece aveam nevoie să mențin semnalul HIGH timp de un ciclu de ceas și trebuia să am o stare în care nu fac citiri sau scrieri.

Pentru al doilea task m-am folosit doar de două stări (a doua fiind cea de gray_done), deoarece nu aveam nevoie de schimbarea coloanelor sau rândurilor de mai multe ori, ci doar o singură dată când utilizam incrementarea, ceea ce se întâmpla la finalul task-ului. Așa că am reluat procesul de l-am explicat mai sus, am citit pixelul de pe poziția (row,col), dar de data aceasta am luat pixelul pe cele 3 canale de câte 8 biți pentru a vedea exact care este maximul si minimul dintre cele 3 și pentru a putea face media dintre cele 2 valori. Deoarece ne trebuia doar valoarea lui G în pixelul de ieșire, am utilizat concatenarea și am pus in locul lui R și B, valoarea 0 pe 8 biți.

La cel de al treilea task, am folosit restul stărilor, am vrut ca pentru fiecare element din matricea de 3x3 pe care trebuia să o calculăm să am câte o nouă stare în care să memorez cate un nou pixel din matricea pe care trebuie aplicat filtrul. Problema a apărut la reținerea valorilor vechi, în

modulul process la aplicarea sharpness-ului eu nu mă folosesc de valorile inițiale, ci valorile care sunt deja calculate, deoarece apelez in_pix, iar valorile se schimbă automat în momentul în care sunt scrise în out_pix, deci valorile de pe rândul de sus al pixelului citit din matricea de 3x3 și pixelul din stânga vor fi schimbați, după cum se poate observa și din tester. Ideea pe care voiam să o implementez în plus pe lângă ce am făcut, a fost să fac o matrice de 3 rânduri și 64 de coloane ca să rețin toate valorile, acest lucru l-aș fi făcut într-un for cu pas constant, iar eu ce ar fi trebuit să fac ar fi fost să salvez primele 2 rânduri de pe primele două poziții, să prelucrez primul rând din matricea cache-uita, pe care ar fi trebuit să-l memorez într-un vector (deoarece nu aș fi putut suprascrie informația), după aceea să scriu rândul 1 din matricea mea de 3x64 pe linia 0, sa shiftez rândurile în sus cu o poziție și abia la final să citesc al 3-lea rând pe a doua poziție. Neimplementând acest lucru, valorile pe care le-am folosit au fost cele deja calculate înainte, dar algoritmul s-a păstrat. Am lucrat cu valorile canalelor R, G, B pentru a avea mai multa precizie in momentul in care fac suma, sumă pe care am pus-o pe 13 biți pentru ca m-am gândit ca este mai mare decât 255*9 care reprezintă 12 biti, deoarece daca R, G sau B ar fi trecut de 255, bitul care era in plus ar fi trecut la următorul canal, ceea ce nu ne-am fi dorit, deoarece s-ar fi stricat rezultatul, așa că fac adunările si înmulțirile pe fiecare 8 biți din cei 24, iar la final verific dacă sumele se află în intervalul [0,255], iar dacă nu, vor primi ori 0, ori 255 in funcție dacă sunt mai mici sau mai mari decât intervalul precizat. În plus, am incrementat sau decrementat valorile lui row și col în partea secvențială pentru a putea seta pixelul.

La al patrulea task am folosit automatul de stări finite pentru a îngloba toate cele 3 task-uri și a le folosi în ordine toate stările de care am avut nevoie pentru îndeplinirea cerințelor.

Concluzii si observatii:

Fiecare task a necesitat o abordare meticuloasă și adaptată cerințelor specifice, iar implementările au fost realizate cu atenție la performanță și precizie. În cadrul proiectului am experimentat cu diverse soluții pentru a rezolva diferite provocări și cerințe întâmpinate, încercând să obțin cele mai bune rezultate.