

## Računanje magnitude gradijenta slike

Potrebno je implementirati metodu za izračunavanje magnitude gradijenta slike na FPGA korišćenjem VHDL jezika i demonstrirati rad na FPGA razvojnom sistemu. U ovom dokumentu je opisana prva faza projekta koji se sastoji iz dve celine koje se nadovezuju jedna na drugu.

### 1 Mreža za računanje kvadratnog korena

Za potrebe računanja magnitude gradijenta slike, potrebno je izračunati kvadratni koren zbira kvadrata gradijenata po horizontalnoj i vertikalnoj osi. Modul za računanje kvadratnog korena treba da bude realizovan korišćenjem metode „cifru-po-cifru“ (engl. *digit-by-digit*).

#### 1.1 Algoritam za računanje korena „cifru-po-cifru“ za dekadni brojevni sistem

Najpre, objasnimo izvođenje algoritma. Pretpostavimo da već znamo deo korena broja  $x$  i nazovimo taj deo  $P$  (**prethodni koren**). Želimo da pronađemo narednu cifru  $b$ . Kada dodajemo novu cifru  $b$  na kraj decimalnog broja  $P$ , novi broj zapravo ima vrednost  $10P + b$ . Naš cilj je da kvadrat tog novog broja bude što bliži broju  $x$ , ali ne veći od njega:

$$(10P + b)^2 \leq x$$

Iterativnim postupkom, tražimo cifru  $b$  tako da prethodna nejednakost bude zadovoljena. Nejednakost se razvijanjem kvadrata binoma može dalje napisati na sledeći način:

$$(10P)^2 + 2 \cdot (10P) \cdot b + b^2 \leq x$$

$$100P^2 + 20Pb + b^2 \leq x$$

$$20Pb + b^2 \leq x - 100P^2 = R,$$

gde je  $R$  **trenutni ostatak**. Faktorisanjem leve strane dobijamo ključnu formulu algoritma:

$$b \cdot (20P + b) \leq R.$$

Sada je moguće postaviti algoritam za izračunavanje korena cifru po cifru. Najpre je potrebno uraditi inicijalizaciju a zatim iterativno odrediti jednu po jednu cifru. Koraci su dati u nastavku, uz primer. Primer rezultata svake iteracije obeležen je posebnom bojom radi lakšeg praćenja (**inicijalizacija**, **iteracija 1**, **iteracija 2**, **iteracija 3**).

Neka je  $x$  broj čiji koren tražimo (**primer 645.16**).

1. **Inicijalizacija:** Podeliti  $x$  u blokove od po dve cifre  $B_1, B_2, \dots, B_n$  (**06 45 16**). Ako  $x$  ima neparan broj cifara pre vodeće cifre treba dopisati 0.
2. **Prvi korak ( $i = 1$ ):** Naći najveće  $b_1$  tako da je  $b_1^2 \leq B_1$  ( **$2^2 \leq 06$** ). Postaviti  $P_1 = b_1$  (**2**) i ostatak  $R_1 = B_1 - b_1^2$  ( **$6 - 4 = 2$** ).
3. **Iteracija ( $i + 1$ ):**

- Uzeti sledeći par cifara  $B_{i+1}$  i dopisati ga ostatku  $R_i$ . Time se formira novi ostatak  $R_{new} = R_i B_{i+1} = 100R_i + B_{i+1}$  (245) (2016)
- Tražimo najveću cifru  $b_{i+1}$  takvu da je:

$$b_{i+1} \cdot (20P_i + b_{i+1}) \leq R_{new} (b_2 = 5)(b_3 = 4)$$

- Ažurirati trenutni koren:  $P_{i+1} = 10P_i + b_{i+1}$  (što je praktično dopisivanje nove cifre na kraj) ( $P_2 = 25$ ) ( $P_3 = 254$ )
- Izračunati novi ostatak:  $R_{i+1} = R_{new} - b_{i+1} \cdot (20P_i + b_{i+1})$ .  $R_2 = 245 - 5 \cdot (20 \cdot 2 + 5) = 20$   
 $R_3 = 2016 - 4 \cdot (20 \cdot 25 + 4) = 0$

## 1.2 Modifikacija algoritma „cifru-po-cifru“ za binarni brojevni sistem

U binarnom sistemu, dopisivanje cifre na kraj broja  $P$  odgovara operaciji  $2P + b$ . Kvadrat tog novog broja je:

$$(2P + b)^2 = 4P^2 + 4Pb + b^2 \leq x.$$

Kao i ranije, iterativnim postupkom, tražimo cifru  $b$  tako da prethodna nejednakost bude zadovoljena. Nejednakost se pojednostavljuje na izraz:

$$4Pb + b^2 = b \cdot (4P + b) \leq R$$

S obzirom na to da je  $b$  uvek 1 ili 0, provera cifre ( $b_{i+1} \cdot (4P_i + b_{i+1}) \leq R_{new}$ ) postaje veoma jednostavna

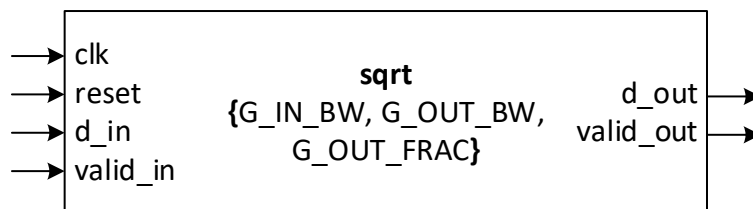
- Ako je ostatak veći ili jednak  $4P + 1$ , nova cifra je 1,
- Ako je ostatak manji od  $4P + 1$ , nova cifra je 0.

Program u Pajtonu koji ilustruje ovaj algoritam dat je u pratećim fajlovima ovog zadatka. Ovaj program se može iskoristiti kao polazna osnova za realizaciju u VHDLu.

## 1.3 Realizacija modula za izračunavanje korena iterativnim sekvencijalnim metodom

Realizovati hardverski modul kojim se implementira algoritam za računanje korena opisan u prethodnim odeljcima. Entity modula treba nazvati **sqrt** i realizovati ga tako da zauzima minimalnu količinu hardverskih resursa. Arhitekturu sistema nazvati **Behavioral\_sqrt\_seq**.

Interfejs ove komponente i funkcionalnost opisani su na slici 1 i u tabelama 1 i 2.




Slika 1 – Blok dijagram modula za računanje korena

Tabela 1 – Opis generika modula za računanje korena

Generik	Tip	Značenje
G_IN_BW	natural	Ukupna bitska širina ulaznog podatka.
G_OUT_BW	natural	Ukupna bitska širina izlaznog podatka.
G_OUT_FRAC	natural	Broj bita za razlomljeni deo izlaza.

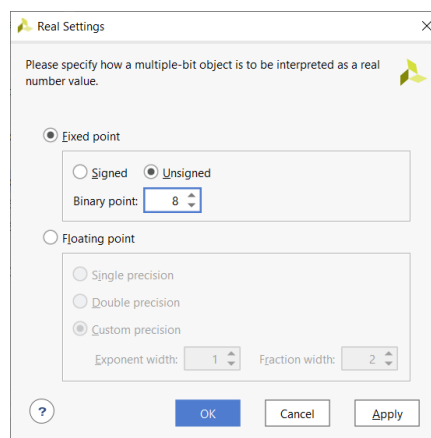
Tabela 2 – Opis ulaznih i izlaznih signala modula za računanje korena

Signal	Bitska širina	Smer	Značenje
clk	1	ulaz	Signal takta.
reset	1	ulaz	Signal reseta aktivan u logičkoj jedinici. Resetuje izlazne registre i trenutno stanje modula. Ukoliko je započeto izračunavanje i signal reseta postao aktivan, uneti podatak se gubi. Nakon otpuštanja reseta, neophodno je da se pojavi novi podatak kako bi se započelo novo izračunavanje.
d_in	G_IN_BW	ulaz	Ulazni podatak: vektor bita širine G_IN_BW koji predstavlja nenegativan ceo broj.
valid_in	1	ulaz	Ulazni signal kojim se signalizira da je podatak na ulazu d_in validan. Trajanje ovog signala je proizvoljno. Modul za računanje korena treba da uzima podatak sa ulaza kada god je spreman da ga prihvati i kada god je signal valid_in aktivan. Ostatak sistema vodi računa o trenucima postavljanja signala.
d_out	G_OUT_BW	izlaz	Izlazni podatak: vektor bita širine G_OUT_BW koji predstavlja nenegativan rezultat operacije kvadratnog korena nad ulaznim podatkom d_in. Broj bita za razlomljeni deo u predstavi izlaznog broja je G_OUT_FRAC.
valid_out	1	izlaz	Izlazni signal kojim se signalizira da je podatak na izlazu d_out validan. Trajanje ovog signala mora biti tačno jedan ciklus signala takta za svaki podatak.

 U nekom alatu za crtanje (drawio, Visio, i sl.) nacrtati blok šemu realizovanog sistema i dijagram mašine stanja. Slike eksportovati u .jpg, .png ili .pdf fajl i priložiti uz kod prilikom slanja rešenja projekta, najbolje združeno u jednom dokumentu u kome treba diskutovati ostale rezultate iz odeljka 1.5.

Napisati testbench kojim se verifikuje razvijena komponenta. U prilogu ovog fajla dat je primer testa koji učitava tekstualni fajl, konvertuje binarne podatke u brojeve koji treba da predstavljaju ulaz modula za koren. Tekstualni fajl treba importovati u Vivado kao simulacioni fajl. U fajlu se simulira ponašanje modula za koren u dummy procesu i proverava se da li je koren broja sa ulaza isti kao brojevi dobijeni iz drugog tekstualnog fajla. Test za komponentu **sqrt** treba napisati na sličan način, ali je umesto dummy procesa potrebno instancirati napisanu hardversku komponentu, a tekstualne fajlove generisati iz Pajton skripte korišćenjem dostupnog koda za računanje korena. Testiranje je uspešno ako su sve vrednosti korena tačno izračunate.


**Napomena:** U waveform viewer-u se mogu odabrati različiti formati prikaza binarnih vektora (Radix), npr. unsigned integer je koristan za ulazne podatke. Ipak, za izlazne podatke treba koristiti neku vrstu fixed point predstave realnog broja. To se postiže odabirom opcije Real Settings i podešavanjem pozicije decimalne tačke kao na slici desno. Primer sa slike prikazuje predstavu neoznačenog broja gde je 8 bita iskorišćeno za razlomljeni deo.



## 1.4 Realizacija modula za izračunavanje korena pajplajnovanom mrežom

Realizovati hardverski modul kojim se implementira algoritam za računanje korena ali tako da je svaka iteracija algoritma poseban stepen pajplajna. Broj stepeni pajplajna zavisi od broja iteracija, a broj iteracija od bitskih širina podataka (pogledati ponovo Pajton kod). Time se postiže veće zauzeće resursa, ali je protok sistema značajno veći nego kod iterativnog metoda. Ovaj pristup se naziva raspetljavanje petlje (engl. *loop unrolling*) i čest je u hardverskim realizacijama iterativnih algoritama.

Entity modula treba da ostane isti, a samim tim i interfejs, a novu realizaciju treba napisati u odvojenoj arhitekturi koju treba nazvati **Behavioral\_sqrt\_pipelined**.

 U nekom alatu za crtanje (drawio, Visio, i sl.) nacrtati blok šemu realizovanog sistema i dijagram mašine stanja. Slike eksportovati u .jpg, .png ili .pdf fajl i priložiti uz kod prilikom slanja rešenja projekta, najbolje združeno u jednom dokumentu u kome treba diskutovati ostale rezultate iz odeljka 1.5.

Po uzoru na testbench iz prethodne tačke, napisati test koji testira pajplajnovanu arhitekturu. Test treba da generiše validne podatke na svaki takt i da čeka rezultat koji bi, ako je komponenta ispravno napisana, trebalo da bude validan takođe na svaki takt (nakon inicijalnog kašnjenja). Proveriti rezultate za sve kombinacije 16-bitnog ulaza, 16-bitnog izlaza i 8-bitnog razlomljenog dela izlaza. Testirati komponentu za još bar dve kombinacije bitskih širina i priložiti test fajlove.

## 1.5 Poređenje rezultata

U posebnom dokumentu (može se koristiti šablon priložen uz ovaj dokument) nacrtati tabelu koja poredi ove dve realizacije za isti set generika.

# 2 Implementacija algoritma za određivanje magnitude gradijenta slike

## 2.1 Memorija za čuvanje slike

Grayscale slika veličine  $256 \times 256$  8-bitnih piksela je smeštena u inicijalizovanoj BRAM memoriji koja je data kao prateći fajl ovog dokumenta (*im\_ram.vhd*). Za inicijalizaciju se koristi fajl *cameraman.dat* koji je potrebno smestiti na istu lokaciju kao i *im\_ram.vhd* i importovati ga u Vivado kao source fajl. Memorija je realizovana kao "Simple Dual Port" memorija, s obzirom na to da će se u kasnijim fazama zadatka koristiti jedan port za čitanje, a drugi za upis u memoriju. Dubina memorije je  $256 \times 256 = 65536$ , a na svakoj memorijskoj lokaciji se nalazi 8-bitna vrednost koja predstavlja intenzitet piksela. Pikseli su poređani po redovima, tj. u prvih 256 lokacija se nalazi 256 piksela prvog reda slike, u drugih 256 lokacija se nalaze pikseli drugog reda slike itd.

Sintetisati memoriju *im\_ram* i uočiti da ona zauzima 16 blokova BRAM memorije, po dva za svaki bit. Dva bloka su kaskadirana na način opisan u odeljku "Cascadable Block RAM", na 23. strani dokumenta dostupnog na linku [https://docs.xilinx.com/v/u/en-US/ug473\\_7Series\\_Memory\\_Resources](https://docs.xilinx.com/v/u/en-US/ug473_7Series_Memory_Resources).

## 2.2 Implementacija Sobelovog operatora

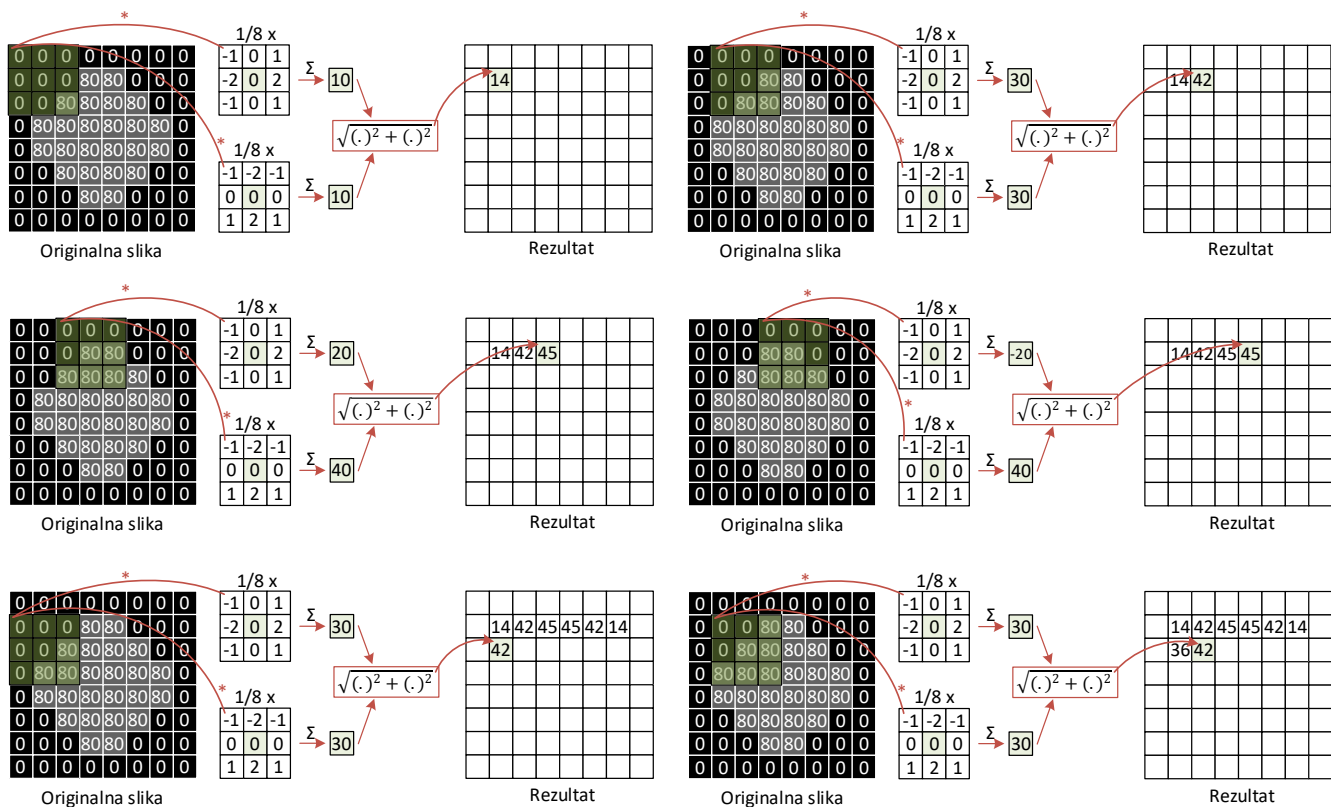
Kako bi se detektovale ivice potrebno je najpre uraditi visokofrekventno filtriranje slike po horizontalnoj i vertikalnoj osi. U te svrhe koriste se dve maske **coeffs\_H** i **coeffs\_V** koje odgovaraju horizontalnim i vertikalnim gradijentima. Za potrebe ovog zadatka koristi se Sobel operator koji se implementira kao standardan 2D filtar (u nastavku detaljnije). Koeficijenti filtra su definisani na sledeći način:

$$\text{Sobel operator: } coeffs\_H = \frac{1}{8} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ i } coeffs\_V = \frac{1}{8} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Ukoliko su rezultati filtriranja prostornim maskama **coeffs\_H** i **coeffs\_V** redom **gradient\_H** i **gradient\_V** onda je magnituda gradijenta definisana na sledeći način:

$$\text{Magnituda gradijenta se određuje kao: } gradient\_M = \sqrt{(gradient\_H)^2 + (gradient\_V)^2}.$$

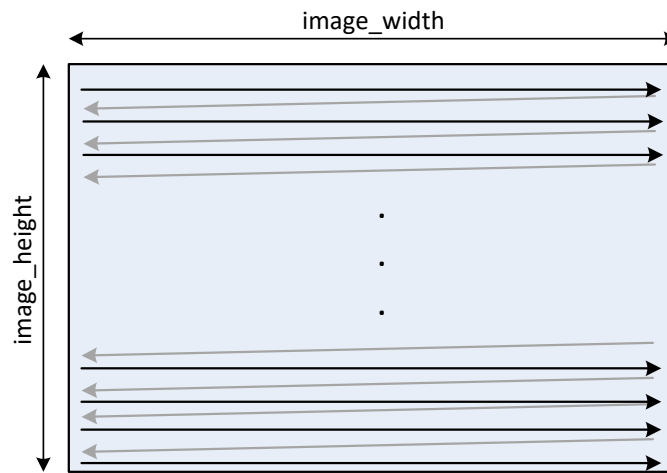
Potrebno je primeniti Sobelov operator nad slikom u memoriji. Filtrom se izvršava 2D korelacija piksela slike i maske (kernela) definisane gorenavedenim koeficijentima. Preporučujemo za čitanje članak sa linka: [https://en.wikipedia.org/wiki/Kernel\\_\(image\\_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing)). Primer obrade nekoliko piksela prikazan je na slici 2, a redosled procesiranja je ilustrovan na slici 3. Primer prikazuje sivi krug na crnoj pozadini. Da bi se dobio piksel na poziciji [1,1], pikseli na pozicijama [0,0], [0,1], [0,2], [1,0], [1,1], [1,2], [2,0], [2,1] i [2,2] množe se sa svakim elementom kernela a zatim se rezultati sumiraju. Primetiti da u hardveru nije potrebno raditi množenja, dovoljno je raditi samo sabiranja i oduzimanja vrednosti piksela koje su ili jednake originalnim ili pomebrane za jednu poziciju ulevo (množenje sa 2). Nakon sabiranja, deljenje sa 8 se postiže pomeranjem udesno za tri pozicije. Radi zadržavanja tačnosti, ovaj pomeraj se može raditi kasnije, nakon finalno izračunatog korena ili čak pre njega ali za 6 mesta.



Slika 2 – Primer primene Sobelovog operatora za računanje gradijenta slike

Kako su svi pikseli 8-bitni neoznačeni celi brojevi, za rezultate vertikalnog i horizontalnog gradijenta dovoljno je koristiti 10 bita (maksimalna vrednost je  $(1+2+1) \times 255 = 1020$ ). Očigledno, kvadrat 10-bitne vrednosti je maksimalno 20-bitna vrednost, a zbir dve 20-bitne vrednosti je maksimalno 21-bitna vrednost, pa je ulaz u koren 21-bitna vrednost proširena za jednu vodeću nulu, dakle vrednost predstavljena na 22 bita. Ipak, nakon operacije korena, veliki broj bita će biti odbačen, pa možemo i pre računanja korena uraditi deljenje sa 64, što je ekvivalentno deljenju sa 8 iz Sobelovog operatora. Deljenje sa 64 predstavlja pomeraj za 6 mesta udesno, pa

je ulaz u modul za računanje korena sada 16-bitni. Rezultat se može predstaviti na bilo kom broju bita većem od 7, ali se zbog tačnosti zaokruživanja koristi takođe 16 bita od čega je 8 za razlomljeni deo (a i da bismo koristili već testiran modul za koren). Rezultat je potrebno u hardveru zaokružiti na 8-bitni ceo broj.

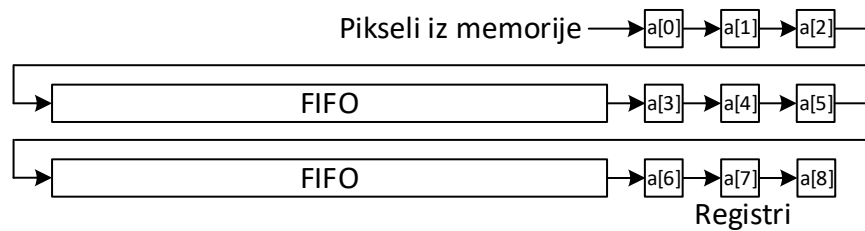


Slika 3 - Redosled procesiranja piksela ulazne slike

Treba obratiti pažnju da je uz ivice slike susedstvo piksela manje, tj. da neki pikseli nemaju sve susedne piksele sa svake strane. Zbog toga je potrebno računati gradijent samo za piksele za koje postoji kompletno susedstvo, što daje sliku dimenzija 254x254 piksela i što odgovara operaciji *correlate2d* iz *Scipy* paketa sa opcijom *mode='valid'*. U Pajton skripti je dato učitavanje slike iz fajla, računanje gradijenta na opisan način i prikaz rezultata. Takođe, dat je i kod koji iz slike generiše inicijalizacioni fajl za memoriju.

Modul koji implementira računanje gradijenta treba da čita piksele iz RAM memorije. Rezultat se mora upisati u istu memoriju, čime se originalni sadržaj memorije uništava. U drugoj fazi projekta, rezultat izračunavanja magnitude gradijenta će biti prikazan na računaru korišćenjem serijske komunikacije. Na UART se mogu slati svi pikseli, uključujući i ivične piksele, dakle svih 256x256 piksela. Nema potrebe raditi isecanje značajnih piksela slike u hardveru. Ipak, u ovoj, prvoj, fazi projekta, rezultat računanja gradijenta, pored upisa u memoriju, treba sačuvati u fajl čiji sadržaj treba učitati u Pajtonu i uporediti sa softverski izračunatim gradijentima (primer upisa u tekstualni fajl iz VHDL testbench-a je ranije dat). Potrebno je obezbediti upis u fajl direktno iz testbench-a u kom slučaju je neophodno prilagoditi test upisu u tekstualni fajl. Podatke je potrebno upisivati u tekstualni fajl u istom trenutku kada se dešava i upis u memoriju.

Sama korelacija sa maskama Sobelovog operatora se može realizovati na više načina, ali se preporučuje da se najpre dve linije slike baferuju u dva FIFO bafera i da se kreira struktura kao na slici 4. Pikseli iz memorije se čitaju na svaki takt po jedan. FIFO baferi su dubine  $256 - 3 = 253$ . Korišćenjem ovakve strukture postiže se da se lokalno susedstvo pomera na svaki takt i omogućava se obrada jednog piksela po taktu. Brzina obrade mora biti približna jednom pikselu po taktu. Preporučuje se studentima da najpre analiziraju tok podataka pre nastavka projekta. Primititi da se isto susedstvo koristi i za vertikalni i za horizontalni gradijent, pa nije potrebno kreirati dve ovakve strukture za oba računanja.



*Slika 4 – Primer strukture bafera za linije slike*

Implementirati računanje magnitude gradijenta slike opisano u ovom odeljku, a rezultate uporediti sa rezultatima iz Pajton skripte. Modul treba da počne sa obradom slike ako je ulazni signal *start* postavljen na logičku jedinicu.

Voditi računa da se logičke putanje što više skrate kako bi se dobila što veća maksimalna učestanost rada. Za računanje korena koristiti pajplajnovanu verziju arhitekture. Obezbediti da je broj ciklusa takta za obradu jedne slike što manji. Odrediti maksimalnu učestanost rada sistema kao i vreme potrebno za obradu jedne slike. Priložiti sve simulacione fajlove.

U pratećem dokumentu nacrtati blok šemu sistema i sve eventualne dijagrame mašina stanja. Napisati koja je ostvarena maksimalna učestanost rada i brzina obrade.