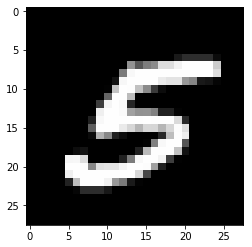
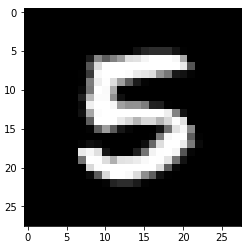
**Deskju(eng. Deskew) algoritam za**

**obradu slike i njegova hardverska implementacija**

**Opis algoritma**

Uloga algoritma je da preprocesira sliku pre nego što se ona klasifikuje pomoću SVM algoritma. Preprocesiranje se svodi na to da broj koji se nalazi na slici, u slučaju da je nakrivljen ispravi. Primer toga prikazan je na sledećoj slici:

*1.Pre preprocesiranja 2.Nakon preprocesiranja*

Razlog zašto se ovo radi je da bi se povećao broj slika koje klasifikator uspešno prepoznaje.

Način na koji algoritam funkcioniše svodi se na pronalaženje prostornih momenata prvog i nultog reda i centralnih momenata korišćenjem sledećih matematičkih formula:

Generalna formula za računanje prostornih momenata:

Formula vrši sumiranje prateći x i y koordinate slike, I(x,y) predstavlja vrednost piksela u datoj kordinati. U zavisnosti od vrednosti parametara „i“ i „j“ mogu da se računaju momenti prvog, drugor, trećeg itd reda. Na primer ako uzmemo da je i=0, i da je j=1, tada računamo moment prvog reda gde gledamo zakrivljenost slike u pravcu y ose.

Za potrebe ovog algoritma neophnodni su prostorni momenti nultog i prvog reda, te se prethodna formula uopštuje:

Moment nultog reda:

Ovaj momenat računa težinu slike, odnosno sumira sve piksele slike. I on je neophodan radi računanja centralnih momenata koji će biti objašnjeni kasnije.

Momenti prvog reda:

Kao što je prethodno napomenuto ovi momenti računaju zakrivljenost slike u pravcu x ose M10 odnosno u pravcu y ose M01.

Momenti prvog reda su korisni u slučaju da je slika centrirana, ali može da se desi da to nije slučaj, i iz tog razloga neophodno je pronaći centralne momente koji se računaju na sledeći način:

Generalna formula za računanje centralnih momenata:

i predstavljaju sumu svih piksela po x odnosno po y (m10, m01) osi podeljenu sa ukupnom težinom slike m00. Centralni momenti koji su dali najbolje rezultati prilikom deskjuovanja(eng. deskewing) slike su mu11 i mu02 te su oni računati u algoritmu.

Uopštena formule za računanje momenata mu11 i mu02 su:

Na osnovu mu11 i mu02 se računa iskošenost (eng. skew) na sledeći način:

**Realizacija algoritma**

Algoritam koji realizuje prethodno navede proračune je:

for (int x=0; x<28; x++)

for(int y = 0; y<28; y++)

{

m00 += image[x+y\*28];

if(m00.overflow\_flag())

cout<<"m00 overflow"<<endl;

m10 += image[x+y\*28]\*x;

if(m10.overflow\_flag())

cout<<"m10 overflow"<<endl;

m01 += image[x+y\*28]\*y;

if(m01.overflow\_flag())

cout<<"m10 overflow"<<endl;

}

x\_mc = m10/m00;

y\_mc = m01/m00;

for (int x=0; x<28; x++)

for(int y = 0; y<28; y++)

{

mu02 += (image[x+y\*28]\*pow((y-y\_mc),2));

if(mu02.overflow\_flag())

cout<<"mu02 overflow:"<<mu02<<endl;

mu11 += (image[x+y\*28]\*(x-x\_mc)\*(y-y\_mc));

if(mu11.overflow\_flag())

cout<<"mu11 overflow:"<<mu11<<endl;

}

skew = mu11/mu02;

Kao što se može videti iz prethodno priloženog koda prva dupla for petlja računa prostorne momente nultog i prvog reda tako što prolazi kroz sve piksele slike i proračunava prethodno navedene sume za dobijanje vrednosti momenata m00, m10,m01. Nakon završetka prve duple for petlje izračunavaju se i koji su u kodi napisani kao x\_mc i y\_mc. Nakon toga kreće druga dupla for petlja koja takođe prolazi kroz sve piksele slike idući po njihovim koordinatama i proračunava prethodo definisane sume za pronalaženje centralnih momenata mu11, mu02.Nakon što je to završeno, proračuvana se skew koji je u stvari samo odnos centralnih momenata mu11 i mu02.

Kada je skew izračunat, na osnovu njegove vrednosti se vrši ispravljanje (eng. deskewing) slike. I za to se koristi sledeći algoritam:

M[0][0] = 1;

M[0][1] = skew;

M[0][2] = -0.5\*28\*skew;

M[1][0] = 0;

M[1][1] = 1;

M[1][2] = 0;

for (int x=0; x<28; x++)

for(int y = 0; y<28; y++)

{

xp = (M[0][0]\*x + M[0][1]\*y + M[0][2]);

yp = (M[1][0]\*x + M[1][1]\*y + M[1][2]);

if(xp<27 && yp<27 && xp>= 0 && yp >=0)

{

x1=(int)xp;

x2=(x1+1);

y2=(y1+1);

R2 = image[x1+y2\*28] + (xp-x1)/(x2-x1)\*(image[x2+y2\*28]-image[x1+y2\*28]);

new\_image[x+y\*28]=R2;

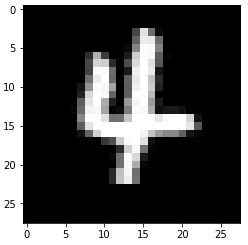
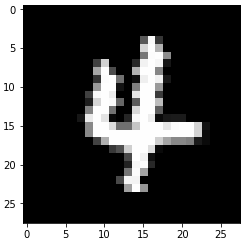
}

else

new\_image[x+y\*28]=0;

}

Na početku se proračunavaju vrednosti karakteristične matrice M. Način na koji se proračunavaju vrednosti ove matrice je stadardni način koji daje dobre rezultate prilikom ispravljanja slike. Nakon tog proračuna započinje jedna dupla for petlja koja iterira po x i y koordinatama slike. Način na koji se prolazi kroz piksele slike je takav da se slika prelazi po vertikalnoj(y) osi, a to je u algoritmu urađeno tako što se uzimaju vrednosti iz niza image na pozicijama x + 28\*y. Svaka iteracija prelazi na naredni piksel slike. U svakoj iteraciji se proračunavaju xp i yp koji se računaju na osnovu matrice M. Vrednosti xp i yp predstavljaju koordinate gde bi piksel na poziciji (x,y) trebao da se nalazi. Nakon što su koordinate xp i yp pronađene vrši se dodatno popravljanje vrednosti na toj koordinati, gde se uzimaju u obzir i vrednosti piksela na okolnim koordinatama (u navedenom kodu to su x1, x2, y2 koordinate), I u zavnisnosti od vrednosti okolnih piksela proračunava se nova vrednost piksela na koordinati (xp, yp), i smešta se u promenljivu R2. Vrednost promenljive R2 se postavlja na koordinatu piksela određenu iteratorima (x, y). Ovaj poslednji korak proračunavanja nove vrednosti piksela na datoj poziciji je neophodan kako bi se izbeglo sledeće:



Slika 3. Slika 4.

Slika 3. Predstavlja situaciju kada se ne uzmu u obzir vrednosti okolnih piksela, dok slika 4. Predstavlja situaciju kada se uzmu u obzir vrednosti okolnih piksela. Kao što se može videte rezultat je bolji na slici 4.

**Hardverska implementacija algoritma(Deskew modul)**

Hardverska implementacija se sastoji iz sledećih koraka:

1. Opis rada hardverskog sistema
2. Eliminacija naredbi ponavljanja iz algoritma
3. Definisanja interfejsa digitalnog Sistema
4. Projektovanja ASMD dijagrama
5. Pisanja HDL modela
6. Testiranja ispravnosti rada HDL modela
7. Oklopljavanje HDL modela sa AXI interfejsom
8. **Opis rada hardverskog Sistema (Deskew modul)**

Uloga Deskew modula je kao što je već opisano da ispravi iskrivljenu sliku i na taj način poboljša klasifikaciju brojeva koju obavlja SVM modul. Slika koju Deskew treba da obradi nalazi se u dvopristupnoj BRAM memoriji kod koje je na jedan pristup povezan Deskew modul koji ima mogućnost čitanja I pisanja iz memorije, a preko drugog pristupa processor upisuje sliku koja treba da se obradi, i kada deskew obavi svoj posao ima mogućnost isčitavanja obrađene slike. Veličina BRAM memorije je 2048 lokacija, svaka od lokacija predstavlja vrednost jednog piksela, gde se na prvih 784 lokacije upisuje slika koja treba da se obradi, a na lokacije od 784 do 1577 deskew upisuje obrađenu sliku (slike koje se obrađuju su u format 28x28). Kada neka slika treba da se obradi, processor nju smešta u BRAM memoriju. Kada je slika upisana u BRAM memoriju, processor obaveštava deskew modul da počne sa radom, a kada je deskew završio posao I upisao obrađenu sliku u memoriju on obaveštava processor, koji sad može da pročita obrađenu sliku iz memorije.

1. **Eliminacija naredbi ponavljanja iz algoritma**

Svodi na eliminaiju for petlji iz algoritma i njihovom zamenom korišćenjem odgovarajućih “if” i “goto” naredbi. To je urađeno na sladeći način:

i1:y = 0;

i2:m00 += image[x+y\*28];

m10 += image[x+y\*28]\*x;

m01 += image[x+y\*28]\*y;

y++;

if(y == 28)

{

x++;

if(x == 28)

goto i3;

goto i1;

}

else

goto i2;

i3:x\_mc = m10/m00;

y\_mc = m01/m00;

x = 0;

i4:y = 0;

i5:mu02 += (image[x+y\*28]\*pow((y-y\_mc),2));

mu11 += (image[x+y\*28]\*(x-x\_mc)\*(y-y\_mc));

y++;

if(y == 28)

{

x++;

if(x == 28)

goto i6;

goto i4;

}

else

goto i5;

i6:skew = mu11/mu02;

M[0][0] = 1;

M[0][1] = skew;

M[0][2] = -0.5\*28\*skew;

M[1][0] = 0;

M[1][1] = 1;

M[1][2] = 0;

x = 0;

i7:y = 0;

i8:xp = (M[0][0]\*x + M[0][1]\*y + M[0][2]);

yp = (M[1][0]\*x + M[1][1]\*y + M[1][2]);

if(xp<27 && yp<27 && xp>= 0 && yp >=0)

{

x1=(int)xp;

x2=(x1+1);

y2=(y1+1);

R2 = image[x1+y2\*28] - (xp-x1)/(x2-x1)\*image[x1+y2\*28];

R2 += (xp-x1)/(x2-x1)\*image[x2+y2\*28];

new\_image[x+y\*28]=R2;

}

else

new\_image[x+y\*28]=0;

y++;

if(y == 28)

{

x++;

if(x == 28)

goto i9;

goto i7;

}

else

goto i8;

i9:end

1. **Definisanje interfejsa digitalnog Sistema**

Digitalni sistem ima sledeće interfejse:

Ulazni i izlazni interfejs podataka

* address – izlazna adresna magistrala tipa lodžik *(eng. logic)* širine 13 bita
* in\_data – ulazna magistrala podataka tipa lodžik *(eng. logic)* širine 16 bita preko koje se u sistem iz memorije isčitavaju pikseli slike koja se obrađuje
* out\_data – izlazna magistrala podataka tipa lodžik (eng. *logic)* širine 16 bita preko koje se obrađeni pikseli upisuju nazad u memoriju
* en – izlazna kontrolna magistrala tipa lodžik(eng. *logic*) širine 1 bit preko koje se omogućava upis ili čitanje iz memorije u kojoj se nalazi slika
* we – izlazna kontrolna magistala tipa lodžik(eng. logic) širine 4 bita koja se koristi za omogućavanje upisa podataka u memoriju.

Komandni interfejs:

* start – ulazni jednobitni port tipa lodžik(*eng. logic*) koji pokreće proces ispravljanja (eng. deskewing) slike.

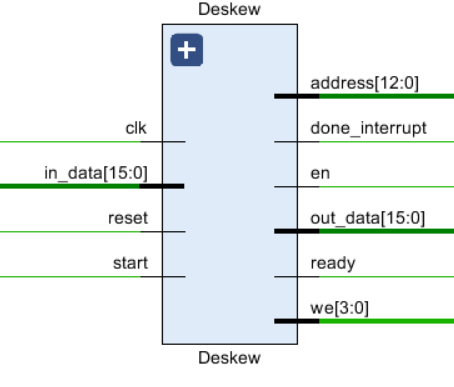
Statusni interfejs:

* ready – izlazni jednobitni port tipa lodžik (*eng. logic*) koji govori da li je deskew spreman da preprocesira narednu sliku
* interrupt\_done – izlazni jednobitni port tipa lodžik (*eng. logic*) koji govori da li je deskew završio sa obradom slike

Standardni interfejs:

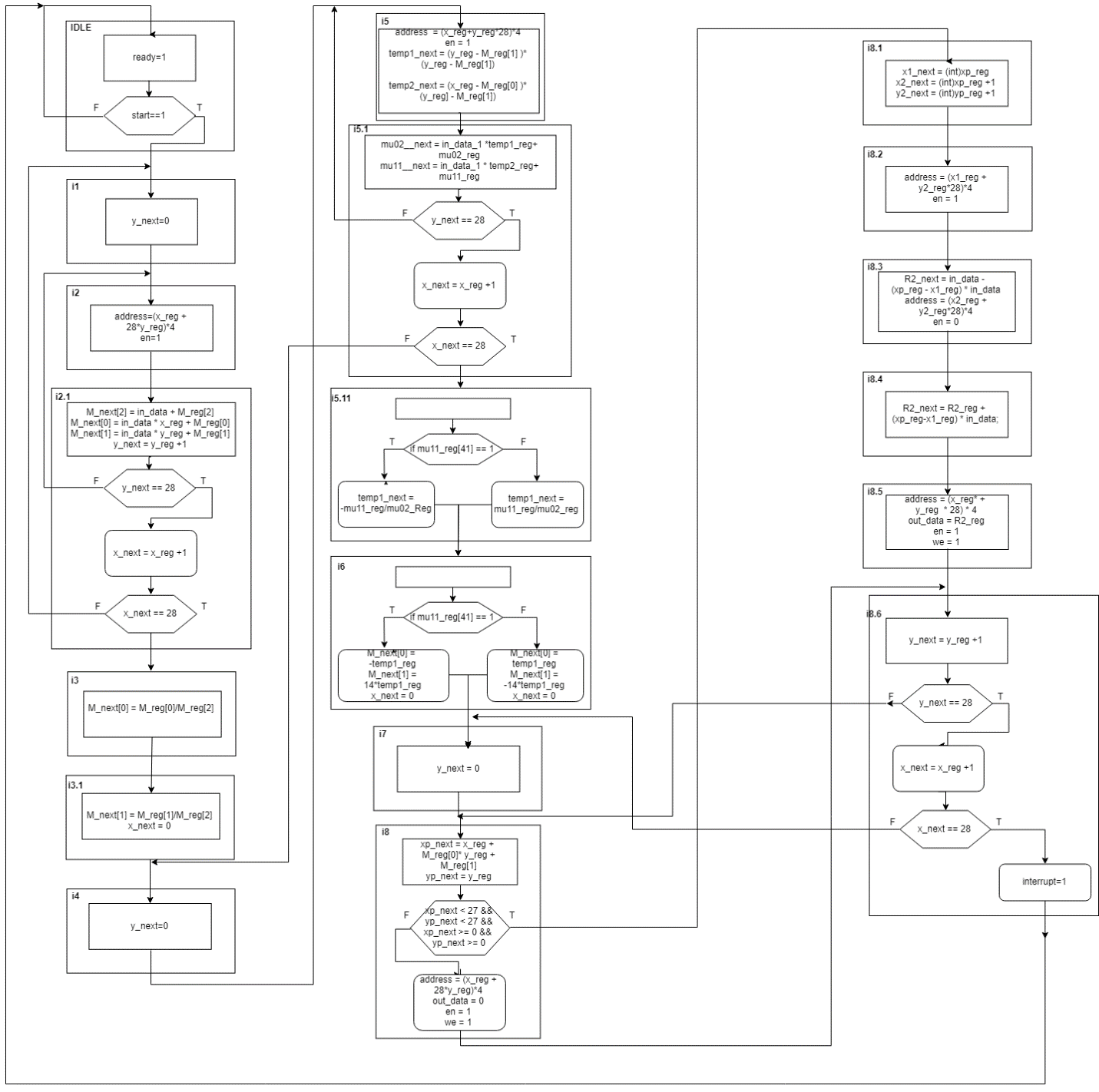
* clk – ulazni port preko koga se dovodi klok signal
* reset – ulazni port preko koga se dovodi reset signal

Na sledećoj slici može se videte interfejs deskew modula:



1. **Projektovanja ASMD dijagrama**

ASMD dijagram na osnovu koga se pravi RTL model digitalnog Sistema je prikazan na sledećoj slici:



ASMD dijagram sastoji se iz 20 stanja, I nazivi tih stanja su: idle ,i1 ,i2, i2\_1, i3,i3\_1, i4, i5, i5\_1, i5\_11, i5\_2, i6, i7, i8, i8\_1, i8\_2, i8\_3, i8\_4, i8\_5, i8\_6. Stanja dijagram prate labele koje se nalaze u kodu nakon eliminacije for petlji pomoću „if“ i „goto“ naredbi.

U idle stanju se sistem nalazi sve dok mu se ne pošalje signal da započne sa obradom slike, i sve dok je u tom stanju on drži ready signal na jedinici. U trenutku kada start signal skoči na logičku jedinicu ready se spušta na nulu, i sistem prelazi u naredno stanje.Naredna 3 stanja (i1, i2, i2.1) predstavljaju prolazak kroz prvu dostruku for petlju početnog algoritma, odnosno izračunavanje momenata m00, m10, m01. U stanju i1 se iterator „y“ resetuje na nulu, u stanju i2 postavlja se vrednost na address izlaz koja predstavlja lokaciju na kojoj se nalazi piksel koji je potrebno isčitati da bi se izvršio proračun. U stanjima i3 i i3.1 vrši se proračun koji se izvršava kod labele i3 koda navedenog pod 2). U stanjima i4, i5, i5.1 se izvršava druga dupla for petlja početnog algoritma, odnosno proračun centralnih momenata mu11 i mu02. U stanju i5.11 vrši se provera da li je centralni momenat mu11 negativan ili pozitivan, u u zavisnosti od toga radi se određena akcija. Ovo je urađeno kako bi narednom stanju (i6) imali deljenje samo pozitivnih brojeva, i na taj način omogućili da sistem radi na višoj frekvenciji, jer je deljenje negativnih brojeva složeniji proces, pa se vreme propagacije povećava. U stanju i6 proračunati su vrednosti karakteristične matrice M koda navedenog u stavci 2) labela i6, stim što su ovde proračunate samo dve vrednosti, jer su ostale vrednosti matrice M konstante. U stanjima od i7 pa do i8.6 vrši se proračun treće duple for petlje gde se piksel po piksel početne slike obrađuje (slika se ispravlja) i upisuju u BRAM memoriju. Kada je cela slika obrađena, sistem šalje interapt signal kako bi obavestio procesor da je završio sa obradom, i vraća se u idle stanje gde čeka sve dok mu se ne pošalje signal da ponoo započne sa obradom slike.

1. **Pisanje HDL modela**

Hdl model kojim je opisan ASMD naveden pod 4) je:

module Deskew#

(

parameter integer WIDTH = 16,

parameter integer ADDRESS = 4

)

(

//clock, reset

input logic clk,

input logic reset,

//command and status signals

input logic start,

output logic ready,

output logic done\_interrupt,

//

//Data transfer IO

output logic [12:0] address,

input logic [WIDTH-1 : 0] in\_data,

output logic [WIDTH-1 : 0] out\_data,

output logic en,

output logic [3:0] we

);

// REG and NEXT signals

logic [4 : 0] x\_reg, y\_reg, x\_next, y\_next;

//registers needed to calculate image moments (fixed point)

logic [WIDTH-1 + 12 : 0] M\_reg[3], M\_next[3];

logic [WIDTH-1 + 26 : 0] temp1\_reg, temp1\_next, temp2\_reg, temp2\_next;

logic [WIDTH-1 + 26 : 0] temp1,temp2;

//logic [WIDTH-1 + 36 : 0] mu02\_next, mu02\_reg;

logic [WIDTH-1 + 26 : 0] mu11\_next, mu11\_reg, mu02\_next, mu02\_reg; //61

//registers needed to deskew image (fixed point)

logic [WIDTH-1 + 26 : 0] xp\_reg, yp\_reg, xp\_next, yp\_next;

logic [WIDTH-1 + 26 : 0] R2\_reg, R2\_next;

logic [WIDTH-1 + 12 : 0] x1\_reg, x1\_next, x2\_reg, y2\_reg, x2\_next, y2\_next;

// ASMD states=

typedef enum logic[4:0]{idle ,i1 ,i2, i2\_1, i3,i3\_1, i4, i5, i5\_1, i5\_11, i5\_2, i6, i7, i8, i8\_1, i8\_2, i8\_3, i8\_4, i8\_5, i8\_6} states;

states state, state\_next;

const logic [25 : 0] one = {13'b0,1'b1,14'b0};

//Register tranfer procedural block

always\_ff @(posedge clk) begin

if(!reset)begin

x\_reg <= 0;

y\_reg <= 0;

mu11\_reg <= 0;

mu02\_reg <= 0;

M\_reg[0] <= 0;

M\_reg[1] <= 0;

M\_reg[2] <= 0;

xp\_reg <= 0;

yp\_reg <= 0;

R2\_reg <= 0;

x1\_reg <= 0;

x2\_reg <= 0;

y2\_reg <= 0;

temp1\_reg <= 0;

temp2\_reg <= 0;

state <= idle;

end

else begin

x\_reg <= x\_next;

y\_reg <= y\_next;

mu11\_reg <= mu11\_next;

mu02\_reg <= mu02\_next;

M\_reg[0] <= M\_next[0];

M\_reg[1] <= M\_next[1];

M\_reg[2] <= M\_next[2];

xp\_reg <= xp\_next;

yp\_reg <= yp\_next;

R2\_reg <= R2\_next;

x1\_reg <= x1\_next;

x2\_reg <= x2\_next;

y2\_reg <= y2\_next;

state <= state\_next;

temp1\_reg <= temp1\_next;

temp2\_reg <= temp2\_next;

end // if (reset == 1)

end // always @ (posedge clk)

//Combination circuit realising ASMD

always@(\*) begin

x\_next = x\_reg;

y\_next = y\_reg;

mu11\_next = mu11\_reg;

mu02\_next = mu02\_reg;

M\_next[0] = M\_reg[0];

M\_next[1] = M\_reg[1];

M\_next[2] = M\_reg[2];

xp\_next = xp\_reg;

yp\_next = yp\_reg;

R2\_next = R2\_reg;

x1\_next = x1\_reg;

x2\_next = x2\_reg;

y2\_next = y2\_reg;

state\_next = state;

temp1\_next = temp1\_reg;

temp2\_next = temp2\_reg;

address = 11'b0;

out\_data = 16'b0;

en = 0;

we = 0;

ready = 0;

done\_interrupt = 0;

case (state)

idle:begin

ready = 1;

if(start == 1)begin

x\_next = 0;

y\_next = 0;

mu11\_next = 0;

mu02\_next = 0;

M\_next[0] = 0;

M\_next[1] = 0;

M\_next[2] = 0;

xp\_next = 0;

yp\_next = 0;

R2\_next = 0;

x1\_next = 0;

x2\_next = 0;

y2\_next = 0;

state\_next = i1;

end

else begin

state\_next = idle;

end

end // case: idle

i1:begin

y\_next = 0;

state\_next = i2;

end

i2:begin

address = (x\_reg + y\_reg\*28) \* ADDRESS;

en = 1;

state\_next = i2\_1;

end

i2\_1:begin

M\_next[2] = in\_data + M\_reg[2];//m00

M\_next[0] = in\_data \* x\_reg + M\_reg[0];//m10

M\_next[1] = in\_data \* y\_reg + M\_reg[1];//m01

y\_next = y\_reg + 1;

if(y\_next == 28)begin

x\_next = x\_reg + 1;

if(x\_next == 28)

state\_next = i3;

else

state\_next = i1;

end

else begin

state\_next = i2;

end

end // case: i2\_1

i3:begin

M\_next[0] = {M\_reg[0][27:10], 4'b0} / M\_reg[2][27:10] ;// x\_mc\_next = m10/m00

x\_next = 0;

state\_next = i3\_1;

end

i3\_1:begin

M\_next[1] = {M\_reg[1][27:10], 4'b0} / M\_reg[2][27:10];// y\_mc\_next = m01/m00

state\_next = i4;

end

i4:begin

y\_next = 0;

state\_next = i5;

end

i5:begin

address = (x\_reg + y\_reg\*28) \* ADDRESS;

en = 1;

temp1\_next = ({y\_reg,14'b0} - {M\_reg[1][17:0], 10'b0}) \* ({y\_reg,14'b0} - {M\_reg[1][17:0], 10'b0});

temp2\_next = ({x\_reg,14'b0} - {M\_reg[0][17:0], 10'b0}) \* ({y\_reg,14'b0} - {M\_reg[1][17:0], 10'b0});

state\_next = i5\_1;

end

i5\_1:begin

mu02\_next = in\_data \* temp1\_reg[41:14] + (mu02\_reg);

mu11\_next = $signed(in\_data)\*$signed(temp2\_reg[41:14]) + $signed(mu11\_reg);

y\_next = y\_reg + 1;

if(y\_next == 28)begin

x\_next = x\_reg + 1;

if(x\_next == 28)

state\_next = i5\_11;

else

state\_next = i4;

end

else begin

state\_next = i5;

end

end // case: i5\_1

i5\_11:begin

if(mu11\_reg[41] == 1)begin//this here was done so that further on in the code we woould have only positive division. That was done to increase the frequency of the circuit

temp1\_next = - mu11\_reg;

end

else

temp1\_next = mu11\_reg;

state\_next = i5\_2;

end

i5\_2:begin

temp1\_next = ({temp1\_reg[40:23],5'b0})/(mu02\_reg[40:23]);//this here is division of positive numbers.

state\_next = i6;

end

i6:begin

temp1 = {temp1\_reg[18:0],9'b0}\* $signed({10'b0,4'b1110,14'b0});

if(mu11\_reg[41] == 1)begin

M\_next[0] = - {temp1\_reg[18:0],9'b0};//-mu11/mu02

M\_next[1] = temp1[41:14];//14\*mu11/mu02

end

else begin

M\_next[0] = {temp1\_reg[18:0],9'b0};//mu11/mu02

M\_next[1] = - temp1[41:14];//14\*mu11/mu02

end

x\_next = 0;

state\_next = i7;

end

i7:begin

y\_next = 0;

state\_next = i8;

end

i8:begin

xp\_next = ({x\_reg,28'b0}) + $signed(M\_reg[0]) \* $signed({9'b0,y\_reg,14'b0}) + ({M\_reg[1], 14'b0});

yp\_next = {23'b0, y\_reg,28'b0};

if(xp\_next < {5'b11011,28'b0} && yp\_next <{5'b11011,28'b0} && xp\_next >= 0 && yp\_next >=0)

state\_next = i8\_1;

else begin

address = (x\_reg + y\_reg\*28 + 784) \* ADDRESS;

out\_data = 0;

//out\_data = 4'b1010;

en = 1;

we = 4'b0011;

state\_next = i8\_6;

end

end // case: i8

i8\_1:begin

x1\_next = {9'b0 ,xp\_reg[32:28], 14'b0};

x2\_next = {9'b0 ,xp\_reg[32:28], 14'b0} + one;

y2\_next = {9'b0 ,yp\_reg[32:28], 14'b0} + one;

state\_next = i8\_2;

end

i8\_2:begin

address = (x1\_reg[18:14] + y2\_reg[18:14] \* 28) \* ADDRESS ;

en = 1;

state\_next = i8\_3;

end

i8\_3:begin

R2\_next = ({26'b0, in\_data,14'b0}) - (($signed(xp\_reg[41:14]) - $signed(x1\_reg)) \* in\_data);

address = (x2\_reg[18:14] + y2\_reg[18:14] \* 28) \* ADDRESS ;

en = 1;

state\_next = i8\_4;

end

i8\_4:begin

R2\_next = R2\_reg + ($signed(xp\_reg[41:14]) - $signed(x1\_reg)) \* in\_data;

state\_next = i8\_5;

end

i8\_5:begin

address = (784 + x\_reg + y\_reg\*28) \* ADDRESS;

out\_data = R2\_reg[29:14];

//out\_data = 4'b1010;

en = 1;

we = 4'b0011;

state\_next = i8\_6;

end

i8\_6:begin

y\_next = y\_reg + 1;

if(y\_next == 28)begin

x\_next = x\_reg + 1;

if(x\_next == 28)begin

done\_interrupt = 1;

state\_next = idle;

end

else

state\_next = i7;

end

else begin

state\_next = i8;

end

end // case: i8\_7

default:begin

//state\_next = state;

end

endcase // case (state)

end // always\_comb

endmodule

Kao što se iz priloženo koda može videti, RTL metodologija je realizovana preko dva „always“ proceduralna bloka, gde prvi „always“ blok realizuje promenu stanja svih registara sistema na svaku rastuću ivicu klok signala, dok drugi „always“ blok realizuje kombinacionu mrežu koja izvršava određenu operaciju u zavisnosti od stanja u kome se sistem nalazi, i rezultat tog izvršavanja upisuje u određeni registar. „Always“ blok koji realizuje kombinacionu mrežu usko prati ASMD koji je opisan u stavci 4).

1. **Testiranja ispravnosti rada HDL modela**

Kod kojim je izvršeno testiranje HDL modela je:

module Deskew#

(

parameter integer WIDTH = 16,

parameter integer ADDRESS = 4

)

(

//clock, reset

input logic clk,

input logic reset,

//control signals IO

input logic start,

output logic ready,

output logic done\_interrupt,

//

//Data transfer IO

output logic [12:0] address,

input logic [WIDTH-1 : 0] in\_data,

output logic [WIDTH-1 : 0] out\_data,

output logic en,

output logic [3:0] we

);

// REG and NEXT signals

logic [4 : 0] x\_reg, y\_reg, x\_next, y\_next;

//registers needed to calculate image moments (fixed point)

logic [WIDTH-1 + 12 : 0] M\_reg[3], M\_next[3];

logic [WIDTH-1 + 26 : 0] temp1\_reg, temp1\_next, temp2\_reg, temp2\_next;

logic [WIDTH-1 + 26 : 0] temp1,temp2;

//logic [WIDTH-1 + 36 : 0] mu02\_next, mu02\_reg;

logic [WIDTH-1 + 26 : 0] mu11\_next, mu11\_reg, mu02\_next, mu02\_reg; //61

//registers needed to calculate DESKEW (fixed point)

logic [WIDTH-1 + 26 : 0] xp\_reg, yp\_reg, xp\_next, yp\_next;

logic [WIDTH-1 + 26 : 0] R2\_reg, R2\_next;

logic [WIDTH-1 + 12 : 0] x1\_reg, x1\_next, x2\_reg, y2\_reg, x2\_next, y2\_next;

// ASMD states=

typedef enum logic[4:0]{idle ,i1 ,i2, i2\_1, i3,i3\_1, i4, i5, i5\_1, i5\_11, i5\_2, i6, i7, i8, i8\_1, i8\_2, i8\_3, i8\_33, i8\_4, i8\_5, i8\_6} states;

states state, state\_next;

const logic [25 : 0] one = {13'b0,1'b1,14'b0};

//FSM STATES

//VARIABLE REGISTER TRANSFER

always\_ff @(posedge clk) begin

if(!reset)begin

x\_reg <= 0;

y\_reg <= 0;

mu11\_reg <= 0;

mu02\_reg <= 0;

M\_reg[0] <= 0;

M\_reg[1] <= 0;

M\_reg[2] <= 0;

xp\_reg <= 0;

yp\_reg <= 0;

R2\_reg <= 0;

x1\_reg <= 0;

x2\_reg <= 0;

y2\_reg <= 0;

temp1\_reg <= 0;

temp2\_reg <= 0;

state <= idle;

end

else begin

x\_reg <= x\_next;

y\_reg <= y\_next;

mu11\_reg <= mu11\_next;

mu02\_reg <= mu02\_next;

M\_reg[0] <= M\_next[0];

M\_reg[1] <= M\_next[1];

M\_reg[2] <= M\_next[2];

xp\_reg <= xp\_next;

yp\_reg <= yp\_next;

R2\_reg <= R2\_next;

x1\_reg <= x1\_next;

x2\_reg <= x2\_next;

y2\_reg <= y2\_next;

state <= state\_next;

temp1\_reg <= temp1\_next;

temp2\_reg <= temp2\_next;

end // if (reset == 1)

end // always @ (posedge clk)

//Combination circuit realising ASMD

always@(\*) begin

x\_next = x\_reg;

y\_next = y\_reg;

mu11\_next = mu11\_reg;

mu02\_next = mu02\_reg;

M\_next[0] = M\_reg[0];

M\_next[1] = M\_reg[1];

M\_next[2] = M\_reg[2];

xp\_next = xp\_reg;

yp\_next = yp\_reg;

R2\_next = R2\_reg;

x1\_next = x1\_reg;

x2\_next = x2\_reg;

y2\_next = y2\_reg;

state\_next = state;

temp1\_next = temp1\_reg;

temp2\_next = temp2\_reg;

address = 11'b0;

out\_data = 16'b0;

en = 0;

we = 0;

ready = 0;

done\_interrupt = 0;

case (state)

idle:begin

ready = 1;

if(start == 1)begin

x\_next = 0;

y\_next = 0;

mu11\_next = 0;

mu02\_next = 0;

M\_next[0] = 0;

M\_next[1] = 0;

M\_next[2] = 0;

xp\_next = 0;

yp\_next = 0;

R2\_next = 0;

x1\_next = 0;

x2\_next = 0;

y2\_next = 0;

state\_next = i1;

end

else begin

state\_next = idle;

end

end // case: idle

i1:begin

y\_next = 0;

state\_next = i2;

end

i2:begin

address = (x\_reg + y\_reg\*28) \* ADDRESS;

en = 1;

state\_next = i2\_1;

end

i2\_1:begin

address = (x\_reg + y\_reg\*28) \* ADDRESS;

en = 1;

M\_next[2] = in\_data + M\_reg[2];//m00

M\_next[0] = in\_data \* x\_reg + M\_reg[0];//m10

M\_next[1] = in\_data \* y\_reg + M\_reg[1];//m01

y\_next = y\_reg + 1;

if(y\_next == 28)begin

x\_next = x\_reg + 1;

if(x\_next == 28)

state\_next = i3;

else

state\_next = i1;

end

else begin

state\_next = i2;

end

end // case: i2\_1

i3:begin

M\_next[0] = {M\_reg[0][27:10], 4'b0} / M\_reg[2][27:10] ;// x\_mc\_next = m10/m00

x\_next = 0;

state\_next = i3\_1;

end

i3\_1:begin

M\_next[1] = {M\_reg[1][27:10], 4'b0} / M\_reg[2][27:10];// y\_mc\_next = m01/m00

state\_next = i4;

end

i4:begin

y\_next = 0;

state\_next = i5;

end

i5:begin

address = (x\_reg + y\_reg\*28) \* ADDRESS;

en = 1;

temp1\_next = ({y\_reg,14'b0} - {M\_reg[1][17:0], 10'b0}) \* ({y\_reg,14'b0} - {M\_reg[1][17:0], 10'b0});

temp2\_next = ({x\_reg,14'b0} - {M\_reg[0][17:0], 10'b0}) \* ({y\_reg,14'b0} - {M\_reg[1][17:0], 10'b0});

state\_next = i5\_1;

end

i5\_1:begin

address = (x\_reg + y\_reg\*28) \* ADDRESS;

en = 1;

mu02\_next = in\_data \* temp1\_reg[41:14] + (mu02\_reg);

mu11\_next = $signed(in\_data)\*$signed(temp2\_reg[41:14]) + $signed(mu11\_reg);

y\_next = y\_reg + 1;

if(y\_next == 28)begin

x\_next = x\_reg + 1;

if(x\_next == 28)

state\_next = i5\_11;

else

state\_next = i4;

end

else begin

state\_next = i5;

end

end // case: i5\_1

i5\_11:begin

if(mu11\_reg[41] == 1)begin//this here was done so that further on in the code we woould have only positive division. That was done to increase the frequency of the circuit

temp1\_next = - mu11\_reg;

end

else

temp1\_next = mu11\_reg;

state\_next = i5\_2;

end

i5\_2:begin

temp1\_next = ({temp1\_reg[40:23],5'b0})/(mu02\_reg[40:23]);//this here is division of positive numbers.

state\_next = i6;

end

i6:begin

temp1 = {temp1\_reg[18:0],9'b0}\* $signed({10'b0,4'b1110,14'b0});

if(mu11\_reg[41] == 1)begin

M\_next[0] = - {temp1\_reg[18:0],9'b0};//-mu11/mu02

M\_next[1] = temp1[41:14];//14\*mu11/mu02

end

else begin

M\_next[0] = {temp1\_reg[18:0],9'b0};//mu11/mu02

M\_next[1] = - temp1[41:14];//14\*mu11/mu02

end

x\_next = 0;

state\_next = i7;

end

i7:begin

y\_next = 0;

state\_next = i8;

end

i8:begin

xp\_next = ({x\_reg,28'b0}) + $signed(M\_reg[0]) \* $signed({9'b0,y\_reg,14'b0}) + ({M\_reg[1], 14'b0});

yp\_next = {23'b0, y\_reg,28'b0};

if(xp\_next < {5'b11011,28'b0} && yp\_next <{5'b11011,28'b0} && xp\_next >= 0 && yp\_next >=0)

state\_next = i8\_1;

else begin

address = (x\_reg + y\_reg\*28 + 784) \* ADDRESS;

out\_data = 0;

//out\_data = 4'b1010;

en = 1;

we = 4'b0011;

state\_next = i8\_6;

end

end // case: i8

i8\_1:begin

x1\_next = {9'b0 ,xp\_reg[32:28], 14'b0};

x2\_next = {9'b0 ,xp\_reg[32:28], 14'b0} + one;

y2\_next = {9'b0 ,yp\_reg[32:28], 14'b0} + one;

state\_next = i8\_2;

end

i8\_2:begin

address = (x1\_reg[18:14] + y2\_reg[18:14] \* 28) \* ADDRESS ;

en = 1;

state\_next = i8\_3;

end

i8\_3:begin

address = (x1\_reg[18:14] + y2\_reg[18:14] \* 28) \* ADDRESS ;

en = 1;

R2\_next = ({26'b0, in\_data,14'b0}) - (($signed(xp\_reg[41:14]) - $signed(x1\_reg)) \* in\_data);

state\_next = i8\_33;

end

i8\_33:begin

address = (x2\_reg[18:14] + y2\_reg[18:14] \* 28) \* ADDRESS ;

en = 1;

state\_next = i8\_4;

end

i8\_4:begin

address = (x2\_reg[18:14] + y2\_reg[18:14] \* 28) \* ADDRESS ;

en = 1;

R2\_next = R2\_reg + ($signed(xp\_reg[41:14]) - $signed(x1\_reg)) \* in\_data;

state\_next = i8\_5;

end

i8\_5:begin

address = (784 + x\_reg + y\_reg\*28) \* ADDRESS;

out\_data = R2\_reg[29:14];

//out\_data = 4'b1010;

en = 1;

we = 4'b0011;

state\_next = i8\_6;

end

i8\_6:begin

y\_next = y\_reg + 1;

if(y\_next == 28)begin

x\_next = x\_reg + 1;

if(x\_next == 28)begin

done\_interrupt = 1;

state\_next = idle;

end

else

state\_next = i7;

end

else begin

state\_next = i8;

end

end // case: i8\_7

default:begin

//state\_next = state;

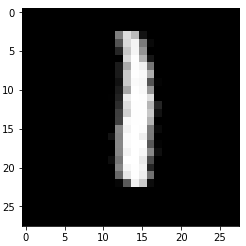
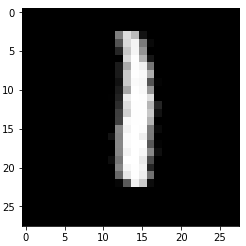
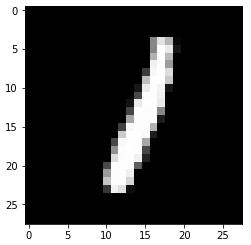
end

endcase // case (state)

end // always\_comb

endmodule

U ovom kodu su instancionirani deskew modul i modul koji predstavlja BRAM memoriju u kojoj će se nalaziti slika koja se obrađuje. U prvom „initial“ bloku vrši se učitavanje slika koje se obrađuju. Slike se učitavaju iz fajla u kome se one predstavljene u binarnom formatu i smešaju u red(*eng. queue*), a taj fajl je generisan korišćenjem matlab i pajton skripti. Takođe, u istom initial bloku se vrši učitavanje „zlatnih vektora“, odnosno slika koje su već ispravljene(*eng. deskewed*), koje će služiti kao način da se utvrdi da li deskew obavlja dobar posao, i one se smeštaju red pod nazivom „golden\_vectors“. One se takođe isčitavaju iz fajla u koji su upisane u binarnom formatu, i taj fajl je takođe generisan korišćenjem pajton i matlab skripti. Drugi initial blok sastoji se iz dva dela koji su međusobno isključivi, i u zavisnosti koji se od njih izvršava, obrađivaće se jedna ili više slika. Ako je makro „multiple\_image“ definisan, taj drugi initial blok obradiće sve raspoložive slike, dok ako je marko „one\_image“ definisan, obradiće se samo prva slika od svih raspoloživih slika. I u jednom i u drugom slučaju obavlja se isti posao, slika koja se obrađuje upisuje se u BRAM memoriju, start signal se postavlja na logičku jedinicu, čeka se da „deskew“ modul završi sa obradom slike, i nakon toga kreće poređenje slike koja je obrađena pomoću deskew modula, i zlatnog vektora generisanog od strane pajton skripte.U slučaju kada se obrađuje više slike, gleda se da li greška koju pravi „deskew“ modul prelazi određenu granicu, a to se vrši korišćenjem tvrđenja (*eng. assert*) ,dok kada se obrađuje samo jedna slika gleda se da li greška koju pravi „deskew“ modul prelazi određenu granicu, i još se obrađena slika se upisuje u fajl u binarnom formatu, kako bi korišćenjem matlab i pajton skripte mogli vizuelno da proverimo da li je „deskew“ modul obavio svoj posao. Rezultat toga je :

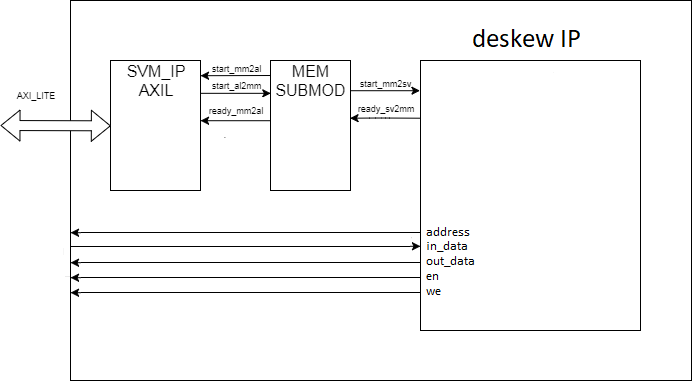


Slika 6 slika 7 slika 8

Slika 6 predstavlja sliku pre ispravljanja a slika 7 sliku posle ispravljanja pomoću „deskew modula“, slika 8 predstavlja sliku ispravljenu pomoću pajton skripte. Kao što se može videti rezultat je isti.

1. **Oklopljavanje HDL modela sa AXI interfejsom**

Nakon što je testiranana funkcionalnost “deskew” modula, on je oklopljen sa standardnim AXI lajt (eng. Axi light) interfejsom preko koga se omogućava standardizovan način komunikacije sa “deskew” modulom. Sledeća slika ilustruje na koji način je to učinjeno :



Slika 9.

Unutar oklopljenog modela sledeće komponente su neophodne:

* SVM\_IP\_AXIL modul preko koga se pristupa registrima unutar oklopljenog modela
* MEM\_SUBMOD modul je posrednik između AXI lajt( *eng. axi light*) interfejsa i „deskew“ modula. U njemu se nalaze dva registra *start* i *ready .* Upisivanjem logičke jedinice u *start* registar započinje se sa obradom slike, dok čitanjme ready registra saznajemo trenutno stanje modula.
* Deskew IP je modul koji vrši obradu slike, i opisivan je na prethodnim stranama.

Pored AXI lajt interfejsa, postoji i dodatni interfejs za komunikaciju sa BRAM memorijom unutar koje se nalazi slika koja se obrađuje, i u koju se smešta obrađena slika. Interfejs sa bram memorijuom je opisan u stavci 3).