УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
Трг Доситеја Обрадовића 6, 21000 Нови Сад, Југославија
Деканат: 021 350-413; 021 450-810; Централа: 021 350-122
Рачуноводство: 021 58-220; Студентска служба: 021 350-763
Телефакс: 021 58-133; e-mail: ftndean@uns.ns.ac.yu

PROJEKAT

Iz Projektovanja složenih digitalnih sistema

TEMA PROJEKTA:

Projektovanje sistema u vivado programskom jeziku za ekstrahovanje karakterističnih tačaka na fotografiji pomoću SURF algoritma

TEKST ZADATKA:

Napraviti IP jezgro za optimizaciju SURF algoritma, za generisanje bitnih tačaka na izlazu sistema.

Mentor:
Ivan Čejić

Studenti:
Ristić Dejana 56/2019
Vig Aleksandar 142/2019
Žarković Nemanja 69/2019

U Novom Sadu, 18.09.2024.

Sadržaj

1. Teorijska analiza

Na osnovu rezultata profajliranja koje je uradjeno na projektu projektovanje elektronskih uređaja na sistemskom nivou, došli smo do zaključka da najviše procesorskog vremena oduzima funkcija makeDescriptor. U ovoj funkciji poziva se createVector koja troši najviše od tog vremena pa je zato odluka bila da se realizuje for petlja u kojoj se poziva funkcija AddSample i funkcija PlaceInIndex koja se poziva u funkciji AddSample. Zatim je odlučeno da se ove tri funkcije spoje u jednu, zbog lakše realizacije njihove funckionalnosti i pravljenja ASMD dijagrama.

2. Implementacija algoritma u C++ jeziku

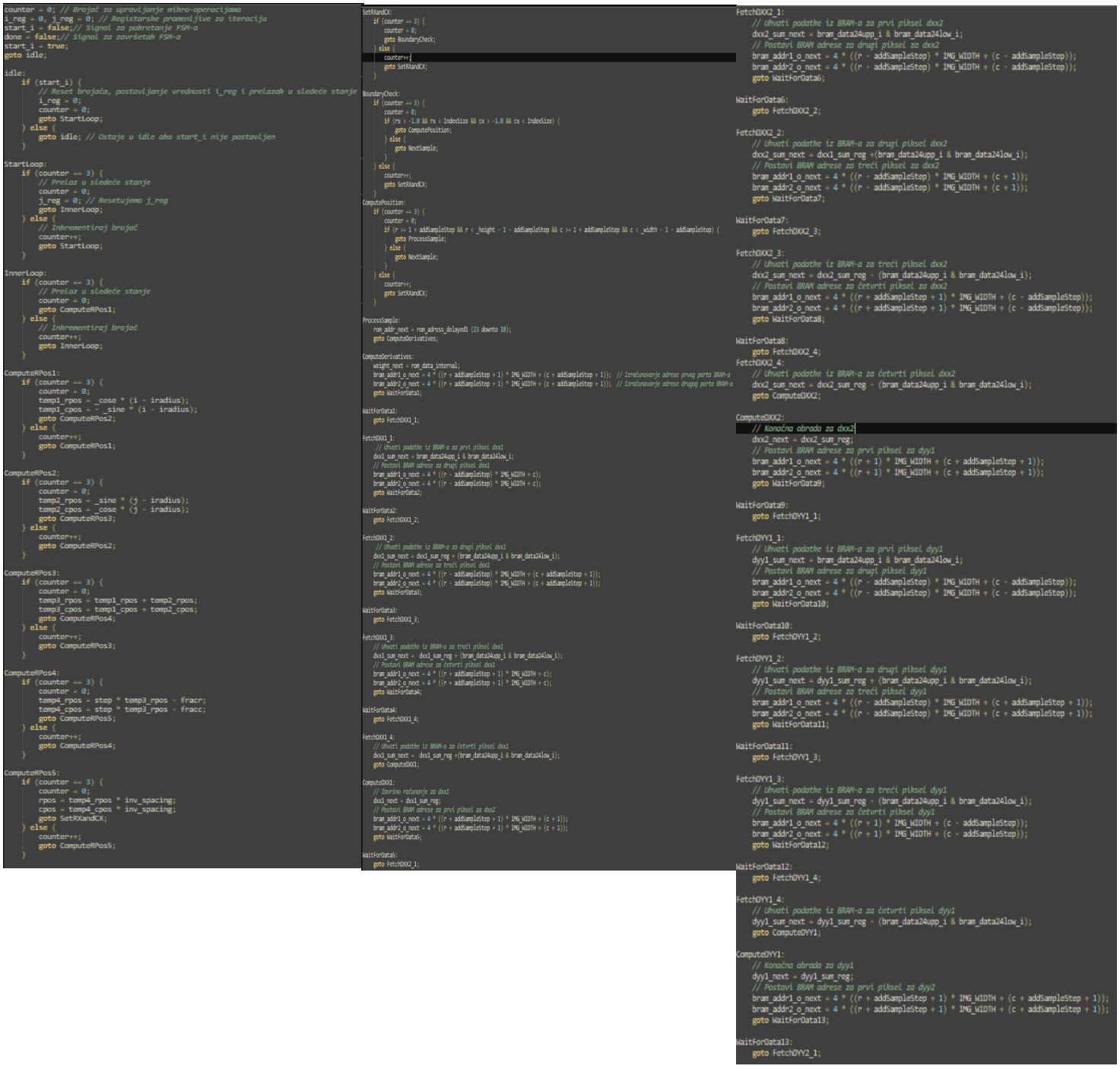
```
void createVector(double scale, double row, double col) {
    int iradius, iy, ix;
    double spacing, radius, rpos, cpos, rx, cx;
    int step = MAX((int)(scale/2 + 0.5),1);

    iy = (int) (row + 0.5);
    ix = (int) (col + 0.5);

    double fracy = row-iy;
    double fracx = col-ix;
    double fracr =  _cose * fracy + _sine * fracx;
    double fracc = - _sine * fracy + _cose * fracx;

    // The spacing of _index samples in terms of pixels at this scale
    spacing = scale * _MagFactor;

    // Radius of _index sample region must extend to diagonal corner of
    // _index patch plus half sample for interpolation.
    radius = 1.4 * spacing * (_IndexSize + 1) / 2.0;
    iradius = (int) (radius/step + 0.5);

    // Examine all points from the gradient image that could lie within the
    // _index square.
    // Examine all points from the gradient image that could lie within the _index square.
        for (int i = 0; i <= 2*iradius; i++) {
            for (int j = 0; j <= 2*iradius; j++) {
                // Rotate sample offset to make it relative to key orientation.
                // Uses (x,y) coords.  Also, make subpixel correction as later image
                // offset must be an integer.  Divide by spacing to put in _index units.
                rpos = (step * (i_cose * (i - iradius) + i_sine * (j - iradius)) - fracr) * inv_spacing;

                cpos = (step * (- i_sine * (i - iradius) + i_cose * (j - iradius)) - fracc) * inv_spacing
                // Compute location of sample in terms of real-valued _index array
                // coordinates.  Subtract 0.5 so that rx of 1.0 means to put full
                // weight on _index[1]
                rx = rpos + 2.0 - 0.5;
                cx = cpos + 2.0 - 0.5;

                // Test whether this sample falls within boundary of _index patch
                if (rx > -1.0 && rx < (double) _IndexSize  &&
                    cx > -1.0 && cx < (double) _IndexSize) {

                    num_i r = iy + (i - iradius) * step;
                    num_i c = ix + (j - iradius) * step;
                    num_i ori1, ori2;
                    num_i ri, ci;

                    num_i addSampleStep = int(scale);

                    num_f weight;
                    num_f dxx1, dxx2, dyy1, dyy2;
                    num_f dx, dy;
                    num_f dxx, dyy;

                    num_f rfrac, cfrac;
                    num_f rweight1, rweight2, cweight1, cweight2;

                    num_f dx1, dx2, dy1, dy2;
```

Slika 1-1 Izgled CreateVector funkcije

```
                    num_f dx1, dx2, dy1, dy2;

                    if (r >= 1 + addSampleStep && r < _height - 1 - addSampleStep && c >= 1 + addSampleStep && c < _width - 1 - addSampleStep) {
                        weight = _lookup2[num_i(rpos * rpos + cpos * cpos)];

                        dxx1 = _Pixels[r + addSampleStep + 1][c + addSampleStep + 1] + _Pixels[r - addSampleStep][c] - _Pixels[r - addSampleStep][c + addSampleStep + 1] - _Pixels[r + addSampleStep + 1][c];
                        dxx2 = _Pixels[r + addSampleStep + 1][c + 1] + _Pixels[r - addSampleStep][c - addSampleStep] - _Pixels[r - addSampleStep][c + 1] - _Pixels[r + addSampleStep + 1][c - addSampleStep];
                        dyy1 = _Pixels[r + 1][c + addSampleStep + 1] + _Pixels[r - addSampleStep][c - addSampleStep] - _Pixels[r - addSampleStep][c + addSampleStep + 1] - _Pixels[r + 1][c - addSampleStep];
                        dyy2 = _Pixels[r + addSampleStep + 1][c + addSampleStep + 1] + _Pixels[r][c - addSampleStep] - _Pixels[r][c + addSampleStep + 1] - _Pixels[r + addSampleStep + 1][c - addSampleStep];

                        dxx = weight * (dxx1 - dxx2);
                        dyy = weight * (dyy1 - dyy2);

                        dx1 = _cose * dxx;
                        dx2 = _sine * dyy;
                        dx = dx1 + dx2;

                        dy1 = _sine * dxx;
                        dy2 = _cose * dyy;
                        dy = dy1 - dy2;

                        if (dx < 0) ori1 = 0;
                        else ori1 = 1;

                        if (dy < 0) ori2 = 2;
                        else ori2 = 3;

                        if (rx < 0)  ri = 0;
                        else if (rx >= _IndexSize) ri = _IndexSize - 1;
                        else ri = rx;

                        if (cx < 0) ci = 0;
                        else if (cx >= _IndexSize) ci = _IndexSize - 1;
                        else ci = cx;

                        rfrac = rx - ri;
                        cfrac = cx - ci;

                        if (rfrac < 0.0) rfrac = 0.0;
                        else if (rfrac > 1.0) rfrac = 1.0;

                        if (cfrac < 0.0) cfrac = 0.0;
                        else if (cfrac > 1.0) cfrac = 1.0;

                        rweight1 = dx * (1.0 - rfrac);
                        rweight2 = dy * (1.0 - rfrac);
                        cweight1 = rweight1 * (1.0 - cfrac);
                        cweight2 = rweight2 * (1.0 - cfrac);

                        if (ri >= 0 && ri < _IndexSize && ci >= 0 && ci < _IndexSize) {
                            _index[ri][ci][ori1] = cweight1;
                            _index[ri][ci][ori2] = cweight2;
                        }
                    }
                }
            }
        }
    }
```

Slika 1-2 Ostatak koda CreateVector funkcije

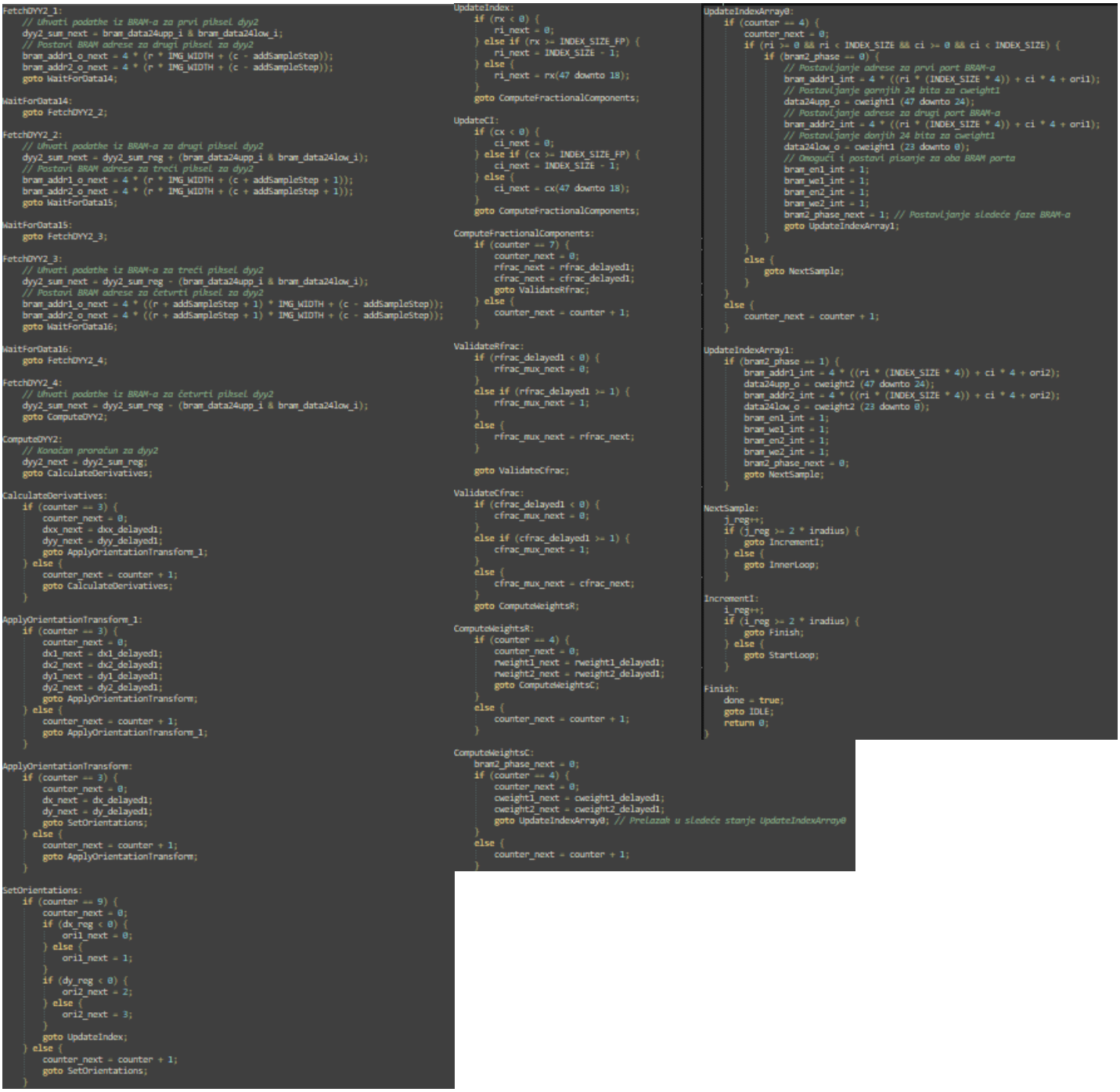
3. Kod sa uklonjenim petljama iz algoritma I izgled IP jezgra



```
counter = 0; // Brojač za upravljanje mikro-operacijama
i_reg = 0, j_reg = 0; // Registarske promenljive za iteraciju
start_i = false;// Signal za pokretanje FSM-a
done = false;// Signal za završetak FSM-a
start_i = true;
goto idle;

idle:
    if (start_i) {
        // Reset brojača, postavljanje vrednosti i_reg i prelazak u sledeće stanje
        i_reg = 0;
        counter = 0;
        goto StartLoop;
    } else {
        goto idle; // Ostaje u idle ako start_i nije postavljen
    }

StartLoop:
    if (counter == 3) {
        // Prelaz u sledeće stanje
        counter = 0;
        j_reg = 0; // Resetujemo j_reg
        goto InnerLoop;
    } else {
        // Inkrementiraj brojač
        counter++;
        goto StartLoop;
    }

InnerLoop:
    if (counter == 3) {
        // Prelaz u sledeće stanje
        counter = 0;
        goto ComputeRPos1;
    } else {
        // Inkrementiraj brojač
        counter++;
        goto InnerLoop;
    }

ComputeRPos1:
    if (counter == 3) {
        counter = 0;
        temp1_rpos = _cose * (i - iradius);
        temp1_cpos = - _sine * (i - iradius);
        goto ComputeRPos2;
    } else {
        counter++;
        goto ComputeRPos1;
    }

ComputeRPos2:
    if (counter == 3) {
        counter = 0;
        temp2_rpos = _sine * (j - iradius);
        temp2_cpos = _cose * (j - iradius);
        goto ComputeRPos3;
    } else {
        counter++;
        goto ComputeRPos2;
    }

ComputeRPos3:
    if (counter == 3) {
        counter = 0;
        temp3_rpos = temp1_rpos + temp2_rpos;
        temp3_cpos = temp1_cpos + temp2_cpos;
        goto ComputeRPos4;
    } else {
        counter++;
        goto ComputeRPos3;
    }

ComputeRPos4:
    if (counter == 3) {
        counter = 0;
        temp4_rpos = step * temp3_rpos - fracr;
        temp4_cpos = step * temp3_rpos - fracc;
        goto ComputeRPos5;
    } else {
        counter++;
        goto ComputeRPos4;
    }

ComputeRPos5:
    if (counter == 3) {
        counter = 0;
        rpos = temp4_rpos * inv_spacing;
        cpos = temp4_cpos * inv_spacing;
        goto SetRXandCX;
    } else {
        counter++;
        goto ComputeRPos5;
    }
```

```
SetRXandCX:
    if (counter == 3) {
        counter = 0;
        goto BoundaryCheck;
    } else {
        counter++;
        goto SetRXandCX;
    }

BoundaryCheck:
    if (counter == 3) {
        counter = 0;
        if (rx > -1.0 && rx < IndexSize && cx > -1.0 && cx < IndexSize) {
            goto ComputePosition;
        } else {
            goto NextSample;
        }
    } else {
        counter++;
        goto SetRXandCX;
    }

ComputePosition:
    if (counter == 3) {
        counter = 0;
        if (r >= 1 + addSampleStep && r < _height - 1 - addSampleStep && c >= 1 + addSampleStep && c < _width - 1 - addSampleStep) {
            goto ProcessSample;
        } else {
            goto NextSample;
        }
    } else {
        counter++;
        goto SetRXandCX;
    }

ProcessSample:
    rom_addr_next = rom_adress_delayed1 (23 downto 18);
    goto ComputeDerivatives;

ComputeDerivatives:
    weight_next = rom_data_internal;
    bram_addr1_o_next = 4 * ((r + addSampleStep + 1) * IMG_WIDTH + (c + addSampleStep + 1));  // Izračunavanje adrese prvog porta BRAM-a
    bram_addr2_o_next = 4 * ((r + addSampleStep + 1) * IMG_WIDTH + (c + addSampleStep + 1));  // Izračunavanje adrese drugog porta BRAM-a
    goto WaitForData1;

WaitForData1:
    goto FetchDXX1_1;

FetchDXX1_1:
    // Uhvati podatke iz BRAM-a za prvi piksel dxx1
    dxx1_sum_next = bram_data24upp_i & bram_data24low_i;
    // Postavi BRAM adrese za drugi piksel dxx1
    bram_addr1_o_next = 4 * ((r - addSampleStep) * IMG_WIDTH + c);
    bram_addr2_o_next = 4 * ((r - addSampleStep) * IMG_WIDTH + c);
    goto WaitForData2;

WaitForData2:
    goto FetchDXX1_2;

FetchDXX1_2:
    // Uhvati podatke iz BRAM-a za drugi piksel dxx1
    dxx1_sum_next = dxx1_sum_reg + (bram_data24upp_i & bram_data24low_i);
    // Postavi BRAM adrese za treći piksel dxx1
    bram_addr1_o_next = 4 * ((r - addSampleStep) * IMG_WIDTH + (c + addSampleStep + 1));
    bram_addr2_o_next = 4 * ((r - addSampleStep) * IMG_WIDTH + (c + addSampleStep + 1));
    goto WaitForData3;

WaitForData3:
    goto FetchDXX1_3;

FetchDXX1_3:
    // Uhvati podatke iz BRAM-a za treći piksel dxx1
    dxx1_sum_next =  dxx1_sum_reg + (bram_data24upp_i & bram_data24low_i);
    // Postavi BRAM adrese za četvrti piksel dxx1
    bram_addr1_o_next = 4 * ((r + addSampleStep + 1) * IMG_WIDTH + c);
    bram_addr2_o_next = 4 * ((r + addSampleStep + 1) * IMG_WIDTH + c);
    goto WaitForData4;

WaitForData4:
    goto FetchDXX1_4;

FetchDXX1_4:
    // Uhvati podatke iz BRAM-a za četvrti piksel dxx1
    dxx1_sum_next =  dxx1_sum_reg +(bram_data24upp_i & bram_data24low_i);
    goto ComputeDXX1;

ComputeDXX1:
    // Završno računanje za dxx1
    dxx1_next = dxx1_sum_reg;
    // Postavi BRAM adrese za prvi piksel za dxx2
    bram_addr1_o_next = 4 * ((r + addSampleStep + 1) * IMG_WIDTH + (c + 1));
    bram_addr2_o_next = 4 * ((r + addSampleStep + 1) * IMG_WIDTH + (c + 1));
    goto WaitForData5;

WaitForData5:
    goto FetchDXX2_1;
```

```
FetchDXX2_1:
    // Uhvati podatke iz BRAM-a za prvi piksel dxx2
    dxx2_sum_next = bram_data24upp_i & bram_data24low_i;
    // Postavi BRAM adrese za drugi piksel za dxx2
    bram_addr1_o_next = 4 * ((r - addSampleStep) * IMG_WIDTH + (c - addSampleStep));
    bram_addr2_o_next = 4 * ((r - addSampleStep) * IMG_WIDTH + (c - addSampleStep));
    goto WaitForData6;

WaitForData6:
    goto FetchDXX2_2;

FetchDXX2_2:
    // Uhvati podatke iz BRAM-a za drugi piksel dxx2
    dxx2_sum_next = dxx1_sum_reg +(bram_data24upp_i & bram_data24low_i);
    // Postavi BRAM adrese za treći piksel za dxx2
    bram_addr1_o_next = 4 * ((r - addSampleStep) * IMG_WIDTH + (c + 1));
    bram_addr2_o_next = 4 * ((r - addSampleStep) * IMG_WIDTH + (c + 1));
    goto WaitForData7;

WaitForData7:
    goto FetchDXX2_3;

FetchDXX2_3:
    // Uhvati podatke iz BRAM-a za treći piksel dxx2
    dxx2_sum_next = dxx2_sum_reg - (bram_data24upp_i & bram_data24low_i);
    // Postavi BRAM adrese za četvrti piksel za dxx2
    bram_addr1_o_next = 4 * ((r + addSampleStep + 1) * IMG_WIDTH + (c - addSampleStep));
    bram_addr2_o_next = 4 * ((r + addSampleStep + 1) * IMG_WIDTH + (c - addSampleStep));
    goto WaitForData8;

WaitForData8:
    goto FetchDXX2_4;
FetchDXX2_4:
    // Uhvati podatke iz BRAM-a za četvrti piksel dxx2
    dxx2_sum_next = dxx2_sum_reg - (bram_data24upp_i & bram_data24low_i);
    goto ComputeDXX2;

ComputeDXX2:
    // Konačna obrada za dxx2
    dxx2_next = dxx2_sum_reg;
    // Postavi BRAM adrese za prvi piksel za dyy1
    bram_addr1_o_next = 4 * ((r + 1) * IMG_WIDTH + (c + addSampleStep + 1));
    bram_addr2_o_next = 4 * ((r + 1) * IMG_WIDTH + (c + addSampleStep + 1));
    goto WaitForData9;

WaitForData9:
    goto FetchDYY1_1;

FetchDYY1_1:
    // Uhvati podatke iz BRAM-a za prvi piksel dyy1
    dyy1_sum_next = bram_data24upp_i & bram_data24low_i;
    // Postavi BRAM adrese za drugi piksel dyy1
    bram_addr1_o_next = 4 * ((r - addSampleStep) * IMG_WIDTH + (c - addSampleStep));
    bram_addr2_o_next = 4 * ((r - addSampleStep) * IMG_WIDTH + (c - addSampleStep));
    goto WaitForData10;

WaitForData10:
    goto FetchDYY1_2;

FetchDYY1_2:
    // Uhvati podatke iz BRAM-a za drugi piksel dyy1
    dyy1_sum_next = dyy1_sum_reg + (bram_data24upp_i & bram_data24low_i);
    // Postavi BRAM adrese za treći piksel dyy1
    bram_addr1_o_next = 4 * ((r - addSampleStep) * IMG_WIDTH + (c + addSampleStep + 1));
    bram_addr2_o_next = 4 * ((r - addSampleStep) * IMG_WIDTH + (c + addSampleStep + 1));
    goto WaitForData11;

WaitForData11:
    goto FetchDYY1_3;

FetchDYY1_3:
    // Uhvati podatke iz BRAM-a za treći piksel dyy1
    dyy1_sum_next = dyy1_sum_reg - (bram_data24upp_i & bram_data24low_i);
    // Postavi BRAM adrese za četvrti piksel dyy1
    bram_addr1_o_next = 4 * ((r + 1) * IMG_WIDTH + (c - addSampleStep));
    bram_addr2_o_next = 4 * ((r + 1) * IMG_WIDTH + (c - addSampleStep));
    goto WaitForData12;

WaitForData12:
    goto FetchDYY1_4;

FetchDYY1_4:
    // Uhvati podatke iz BRAM-a za četvrti piksel dyy1
    dyy1_sum_next = dyy1_sum_reg - (bram_data24upp_i & bram_data24low_i);
    goto ComputeDYY1;

ComputeDYY1:
    // Konačna obrada za dyy1
    dyy1_next = dyy1_sum_reg;
    // Postavi BRAM adrese za prvi piksel za dyy2
    bram_addr1_o_next = 4 * ((r + addSampleStep + 1) * IMG_WIDTH + (c + addSampleStep + 1));
    bram_addr2_o_next = 4 * ((r + addSampleStep + 1) * IMG_WIDTH + (c + addSampleStep + 1));
    goto WaitForData13;

WaitForData13:
    goto FetchDYY2_1;
```



```
FetchDYY2_1:
    // Uhvati podatke iz BRAM-a za prvi piksel dyy2
    dyy2_sum_next = bram_data24upp_i & bram_data24low_i;
    // Postavi BRAM adrese za drugi piksel za dyy2
    bram_addr1_o_next = 4 * (r * IMG_WIDTH + (c - addSampleStep));
    bram_addr2_o_next = 4 * (r * IMG_WIDTH + (c - addSampleStep));
    goto WaitForData14;

WaitForData14:
    goto FetchDYY2_2;

FetchDYY2_2:
    // Uhvati podatke iz BRAM-a za drugi piksel dyy2
    dyy2_sum_next = dyy2_sum_reg + (bram_data24upp_i & bram_data24low_i);
    // Postavi BRAM adrese za treći piksel za dyy2
    bram_addr1_o_next = 4 * (r * IMG_WIDTH + (c + addSampleStep + 1));
    bram_addr2_o_next = 4 * (r * IMG_WIDTH + (c + addSampleStep + 1));
    goto WaitForData15;

WaitForData15:
    goto FetchDYY2_3;

FetchDYY2_3:
    // Uhvati podatke iz BRAM-a za treći piksel dyy2
    dyy2_sum_next = dyy2_sum_reg - (bram_data24upp_i & bram_data24low_i);
    // Postavi BRAM adrese za četvrti piksel za dyy2
    bram_addr1_o_next = 4 * ((r + addSampleStep + 1) * IMG_WIDTH + (c - addSampleStep));
    bram_addr2_o_next = 4 * ((r + addSampleStep + 1) * IMG_WIDTH + (c - addSampleStep));
    goto WaitForData16;

WaitForData16:
    goto FetchDYY2_4;

FetchDYY2_4:
    // Uhvati podatke iz BRAM-a za četvrti piksel dyy2
    dyy2_sum_next = dyy2_sum_reg - (bram_data24upp_i & bram_data24low_i);
    goto ComputeDYY2;

ComputeDYY2:
    // Konačan proračun za dyy2
    dyy2_next = dyy2_sum_reg;
    goto CalculateDerivatives;

CalculateDerivatives:
    if (counter == 3) {
        counter_next = 0;
        dxx_next = dxx_delayed1;
        dyy_next = dyy_delayed1;
        goto ApplyOrientationTransform_1;
    } else {
        counter_next = counter + 1;
        goto CalculateDerivatives;
    }

ApplyOrientationTransform_1:
    if (counter == 3) {
        counter_next = 0;
        dx1_next = dx1_delayed1;
        dx2_next = dx2_delayed1;
        dy1_next = dy1_delayed1;
        dy2_next = dy2_delayed1;
        goto ApplyOrientationTransform;
    } else {
        counter_next = counter + 1;
        goto ApplyOrientationTransform_1;
    }

ApplyOrientationTransform:
    if (counter == 3) {
        counter_next = 0;
        dx_next = dx_delayed1;
        dy_next = dy_delayed1;
        goto SetOrientations;
    } else {
        counter_next = counter + 1;
        goto ApplyOrientationTransform;
    }

SetOrientations:
    if (counter == 9) {
        counter_next = 0;
        if (dx_reg < 0) {
            ori1_next = 0;
        } else {
            ori1_next = 1;
        }
        if (dy_reg < 0) {
            ori2_next = 2;
        } else {
            ori2_next = 3;
        }
        goto UpdateIndex;
    } else {
        counter_next = counter + 1;
        goto SetOrientations;
    }
```

```
UpdateIndex:
    if (rx < 0) {
        ri_next = 0;
    } else if (rx >= INDEX_SIZE_FP) {
        ri_next = INDEX_SIZE - 1;
    } else {
        ri_next = rx(47 downto 18);
    }
    goto ComputeFractionalComponents;

UpdateCI:
    if (cx < 0) {
        ci_next = 0;
    } else if (cx >= INDEX_SIZE_FP) {
        ci_next = INDEX_SIZE - 1;
    } else {
        ci_next = cx(47 downto 18);
    }
    goto ComputeFractionalComponents;

ComputeFractionalComponents:
    if (counter == 7) {
        counter_next = 0;
        rfrac_next = rfrac_delayed1;
        cfrac_next = cfrac_delayed1;
        goto ValidateRfrac;
    } else {
        counter_next = counter + 1;
    }

ValidateRfrac:
    if (rfrac_delayed1 < 0) {
        rfrac_mux_next = 0;
    }
    else if (rfrac_delayed1 >= 1) {
        rfrac_mux_next = 1;
    }
    else {
        rfrac_mux_next = rfrac_next;
    }

    goto ValidateCfrac;

ValidateCfrac:
    if (cfrac_delayed1 < 0) {
        cfrac_mux_next = 0;
    }
    else if (cfrac_delayed1 >= 1) {
        cfrac_mux_next = 1;
    }
    else {
        cfrac_mux_next = cfrac_next;
    }
    goto ComputeWeightsR;

ComputeWeightsR:
    if (counter == 4) {
        counter_next = 0;
        rweight1_next = rweight1_delayed1;
        rweight2_next = rweight2_delayed1;
        goto ComputeWeightsC;
    }
    else {
        counter_next = counter + 1;
    }

ComputeWeightsC:
    bram2_phase_next = 0;
    if (counter == 4) {
        counter_next = 0;
        cweight1_next = cweight1_delayed1;
        cweight2_next = cweight2_delayed1;
        goto UpdateIndexArray0; // Prelazak u sledeće stanje UpdateIndexArray0
    }
    else {
        counter_next = counter + 1;
    }
```

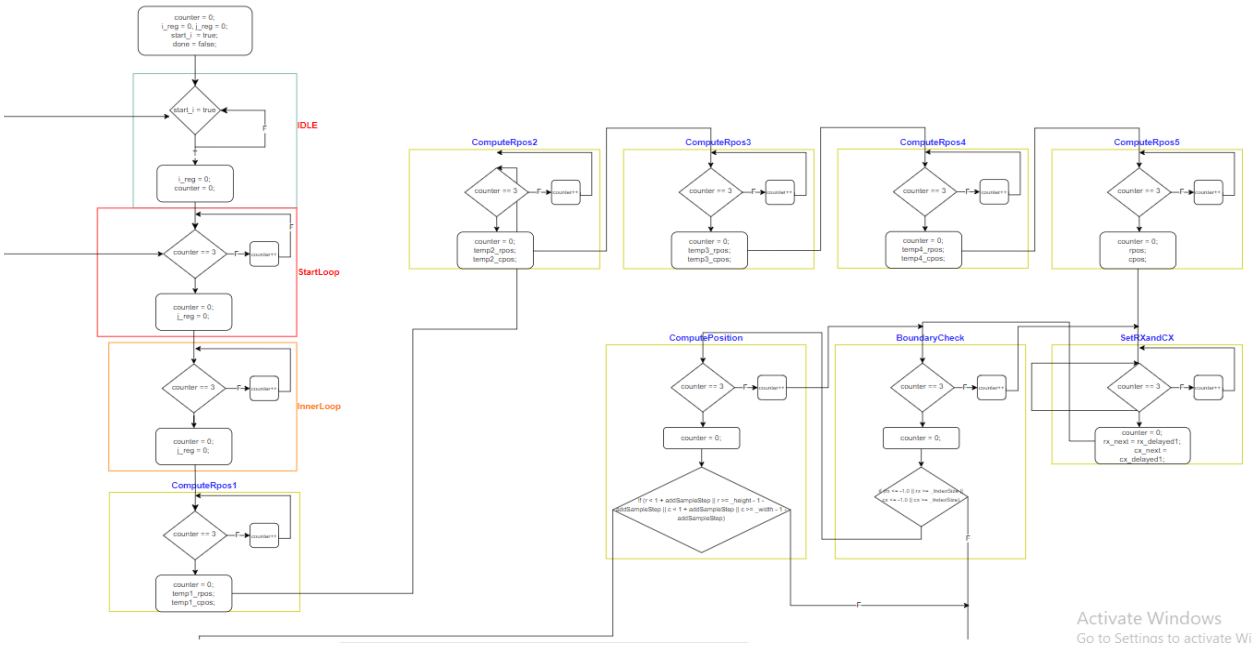
```
UpdateIndexArray0:
    if (counter == 4) {
        counter_next = 0;
        if (ri >= 0 && ri < INDEX_SIZE && ci >= 0 && ci < INDEX_SIZE) {
            if (bram2_phase == 0) {
                // Postavljanje adrese za prvi port BRAM-a
                bram_addr1_int = 4 * ((ri * (INDEX_SIZE * 4)) + ci * 4 + ori1);
                // Postavljanje gornjih 24 bita za cweight1
                data24upp_o = cweight1 (47 downto 24);
                // Postavljanje adrese za drugi port BRAM-a
                bram_addr2_int = 4 * ((ri * (INDEX_SIZE * 4)) + ci * 4 + ori1);
                // Postavljanje donjih 24 bita za cweight1
                data24low_o = cweight1 (23 downto 0);
                // Omogući i postavi pisanje za oba BRAM porta
                bram_en1_int = 1;
                bram_we1_int = 1;
                bram_en2_int = 1;
                bram_we2_int = 1;
                bram2_phase_next = 1; // Postavljanje sledeće faze BRAM-a
                goto UpdateIndexArray1;
            }
        }
        else {
            goto NextSample;
        }
    }
    else {
        counter_next = counter + 1;
    }

UpdateIndexArray1:
    if (bram2_phase == 1) {
        bram_addr1_int = 4 * ((ri * (INDEX_SIZE * 4)) + ci * 4 + ori2);
        data24upp_o = cweight2 (47 downto 24);
        bram_addr2_int = 4 * ((ri * (INDEX_SIZE * 4)) + ci * 4 + ori2);
        data24low_o = cweight2 (23 downto 0);
        bram_en1_int = 1;
        bram_we1_int = 1;
        bram_en2_int = 1;
        bram_we2_int = 1;
        bram2_phase_next = 0;
        goto NextSample;
    }

NextSample:
    j_reg++;
    if (j_reg >= 2 * iradius) {
        goto Increment1;
    } else {
        goto InnerLoop;
    }

Increment1:
    i_reg++;
    if (i_reg >= 2 * iradius) {
        goto Finish;
    } else {
        goto StartLoop;
    }

Finish:
    done = true;
    goto IDLE;
    return 0;
}
```

Slika 2-1,2,3,4,5,6 Izgled pseudo algoritma, u kome su eliminisane petlje

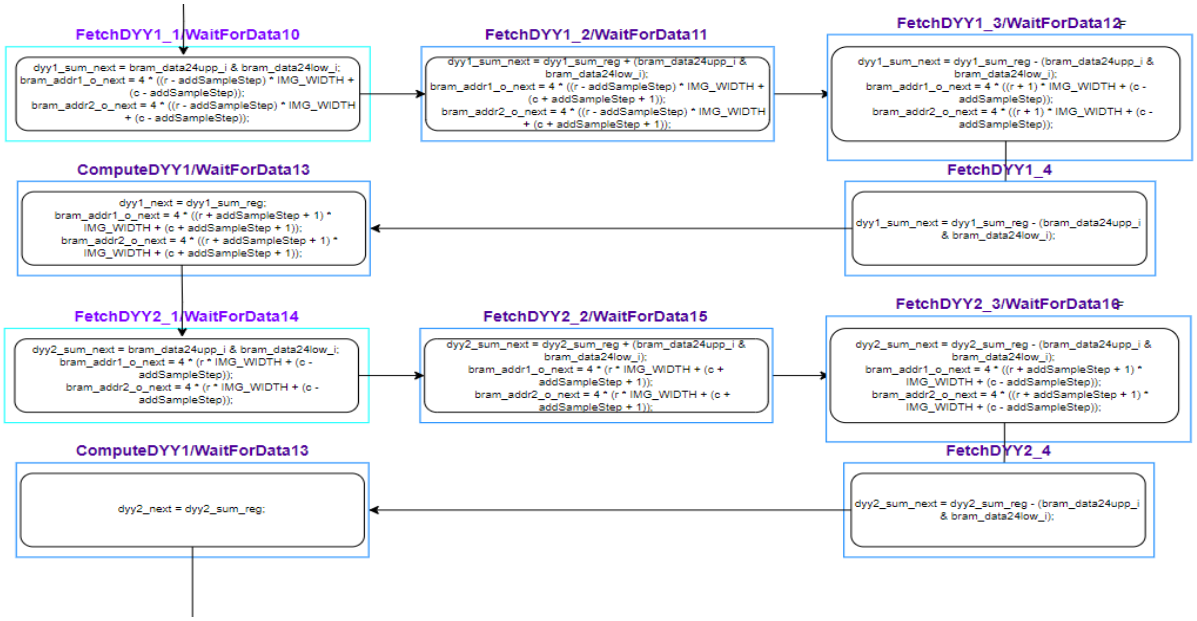
Slika 3 Ulazni i izlazni signali sistema

Ovo je blok svih ulaza i izlaza sistema i na osnovu njega i pseduo koda u kome smo eliminisali obe for petlje i zamenili ih sa goto naredbama za prelazak u sledeće stanje, napravljen je ASMD dijagram. U modifikovanom algoritmu napravljena su sva prelazna stanja i dodati su komentari za objašnjenja svakog od njih.

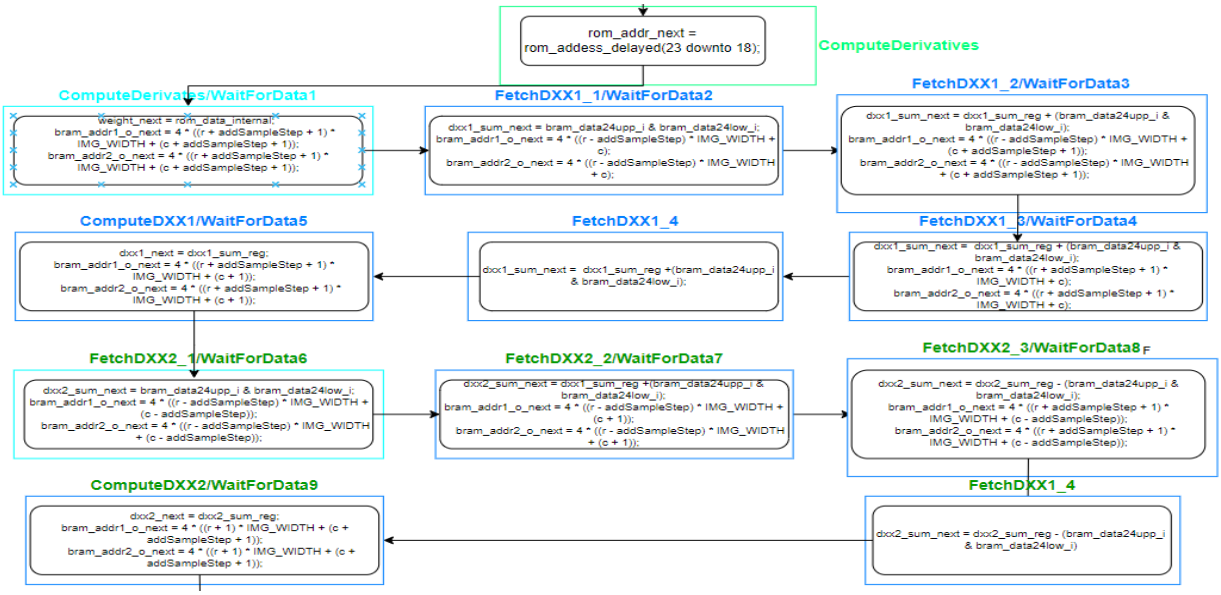
4. ASMD dijagram



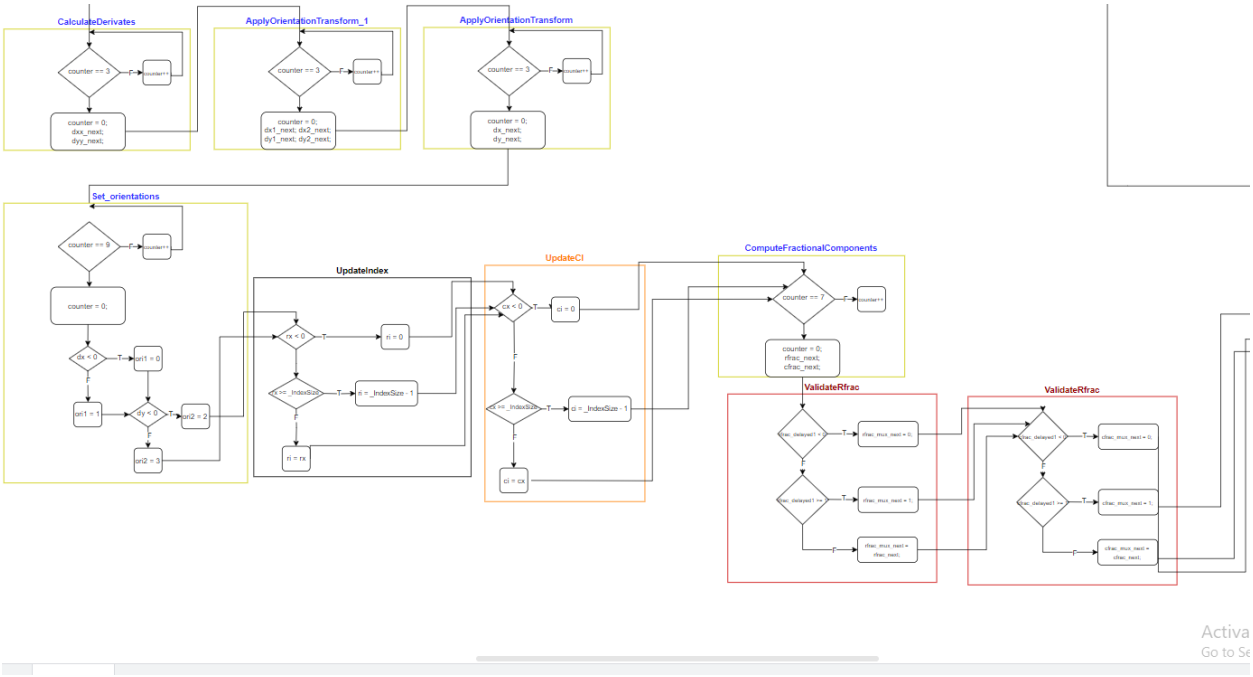
Slika 4-1Ulazak u petlju i računanje vrednosti iz DSP jedinica



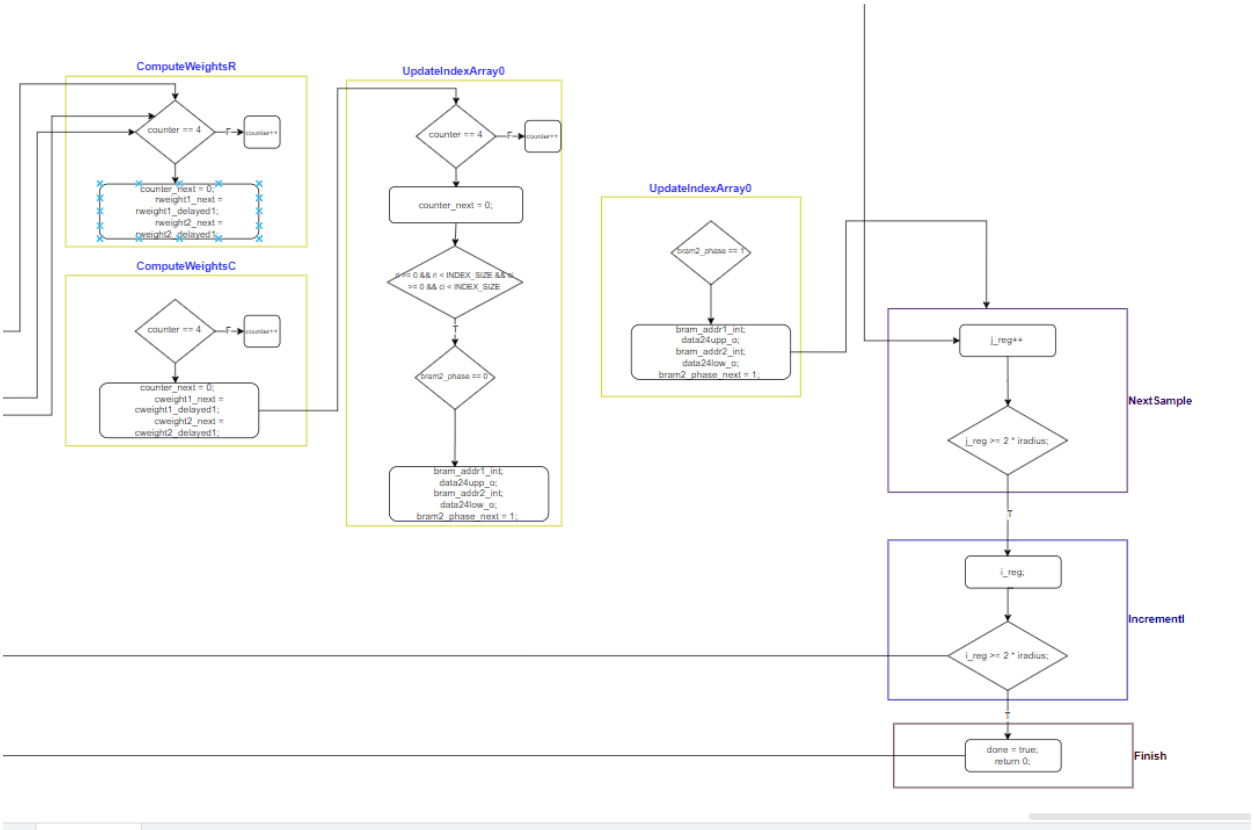
Slika 4-2 Postavljanje adresa na bramove i računanje vrednosti dxx1 i dxx2



Slika 4-3 Računanje vrednosti dyy1 i dyy2 preko bramova



Slika 4-4Izračunavanje svih potrebnih vrednosti, i njihova priprema za cweight



Slika 4-5Računanje cweight vrednosti , upisivanje u indeks niz i završetak petlje

5. Analiza utrošenih resursa

Resource	Estimation	Available	Utilization...
LUT	6362	17600	36.15
LUTRAM	652	6000	10.87
FF	2535	35200	7.20
DSP	26	80	32.50

Slika 5Iskorišćenost resursa za IP pre generisanja axi lite

Na osnovu slike vidimo da sistem koristi zadovoljavajuće vrednosti, i utrošenost je u granicama. Pošto je korišćen *mode out_of_context* u sistemu nisu prikazani ulazni i izlazni pinovi, pa zbog toga može doći do nepouzdanih podataka u kritičnoj putanji, što ćemo videti kasnije.

Dakle sineza i implementacija su prošle, sada je samo potrebno objasniti gde se koriste DSP jedinice.

DSP1:

Koristi se za izračunavanje međurezultata vezanih za radijalne i kružne pozicije, specifično u *temp1_rpos_inc_dsp, temp1_cpos_inc_dsp, temp2_rpos_inc_dsp* i *temp2_cpos_inc_dsp* za izračunavanje *temp1_rpos_delayed, temp1_cpos_delayed, temp1_rpos_delayed1*. I *temp1_cpos_delayed1* Ove jedinice koriste DSP blok za obavljanje množenja koje uključuje kosinus ili sinus uglova (*i_cose, neg_i_sine*).

DSP2:

Koristi se u *temp3_rpos_inc_dsp , temp3_cpos_inc_dsp dx_inc_dsp, dy_inc_dsp I rom_adress_inc_dsp* za dodavanje međurezultata i generisanje adresa, koristeći sabiranje ili oduzimanje na osnovu definisanja potrebe te jedinice.(ADD_SUB signal kontroliše dešavanja)

DSP3:

Angažovan u izračunavanju *temp4_rpos_inc_dsp I temp4_cpos_inc_dsp* za dalju obradu pozicionih podataka, uključujući sabiranje i množenje u nizu.

DSP4:

Upotrebljava se u *rpos_inc_dsp, cpos_inc_dsp, rpos_squared_inc_dsp , cpos_squared_inc_dsp, dx1_inc_dsp, dx2_inc_dsp, dy1_inc_dsp i dy2_inc_dsp* se obrađuju pozicioni podaci, koristeći operacije množenja za skaliranje.

DSP5:

Koristi se u *rx_inc_dsp I cx_inc_dsp* za obradu podataka visoke preciznosti u fiksnom zarezu, koji su uključeni u indeksiranje pozicija i transformacije.

DSP6:

Koristi se u *rfrac_inc_dsp I cfrac_inc_dsp* za izračunavanje frakcijskih delova izračunatih pozicija, primenjujući operacije kao što su oduzimanja ili sabiranja u zavisnosti šta je potrebno u datom trebutku

DSP7:

Pojavljuje se u proračunima gde su višestruke operacije kao što su sabiranje i množenje povezane, na primer u izračunavanju novih pozicija *r_inc_dsp I c_inc_dsp*.

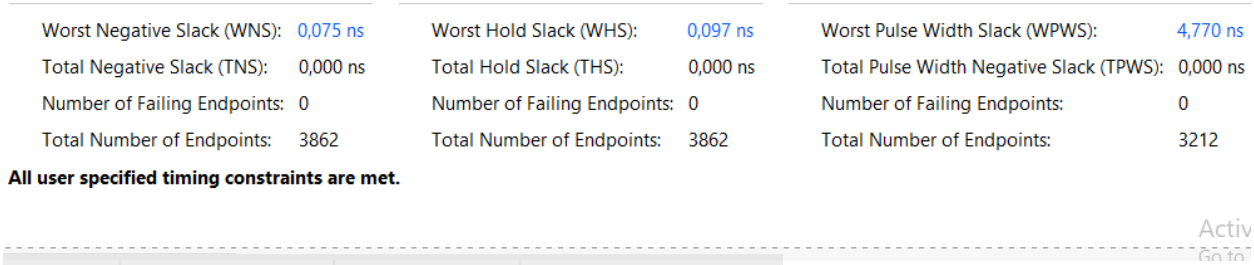
DSP8:

Koristi se u operacijama gde se računaju ukupni zbirovi, prethodnih dsp jedinica *dxx_inc_dsp, dyy_inc_dsp, rweight1_inc_dsp , rweight2_inc_dsp, cweight1_inc_dsp i cweight2_inc_dsp* što je ključno za aplikacije filtriranja ili transformacije.

6. Frekvencija rada sistema i kritična putanja

Na osnovu slike kada je došlo do stabilizovanja putanje zaključujemo da je kritična putanja od u3_1_reg instancioniranog u c_inc_dsp do res_reg instancioniranog u temp4_cpos_inc_dsp.

Pošto je korišćen mod out_of_context što će sprečiti vremensku procenu za clk deltay/skew.



Name	Waveform	Period (ns)	Frequency (MHz)
clk	{0.000 5.750}	11.500	86.957

Slika 3WNS i najveća frekvencija sistema

Path 2	0.105	17	16	187	c_inc_dsp/u3_1_reg_reg[5]/C	temp4_cpos_inc_dsp/res_reg_reg[29]/D	11.373	5.391	5.982
Path 3	0.170	17	15	175	c_inc_dsp/u3_1_reg_reg[7]/C	temp4_rpos_inc_dsp/res_reg_reg[30]/D	11.308	5.965	5.343
Path 4	0.186	17	15	175	c_inc_dsp/u3_1_reg_reg[7]/C	temp4_rpos_inc_dsp/res_reg_reg[28]/D	11.292	5.949	5.343

Slika 4Kritična putanja sistema

Takođe da bi smo dobili frekvenciju rada sistema potrebno je da se od periode rada clock signala (11,5 ns)oduzme Worst Negative Slack(WNS) tj 0,075ns i time se dobija frekvencija od 86,987 MHZ.

7. Pakovanje u IP jezgro

Za pakovanje sistema u ip jezgro koristi se AXI-lite protokol. AXI (Advanced eXtensible Interface) je deo ARM AMBA specifikacije za povezivanje i upravljanje funkcionalnim blokovima u sistemskom dizajnu System on Chip).U našem kodu, AXI interfejs se koristi za omogućavanje komunikacije između procesora i IP bloka), tj. SURF_V1_0 komponenta.

U memorijskom podsistemu SURF_V1_0_S00_AXI imamo 17 registara koji se koriste za smeštanje vrednosti koje dolaze preko AXI interfejsa i koje su dostupne na izlaznim portovima modula. Svaki od *slv_reg* se ponaša kao memorijska lokacija koja čuva podatke koji se mogu pisati ili čitati preko AXI magistrale. Prvih 10 se koristi za čuvanje gornjih i donjih vrednosti 24obithih registara koji dolaze iz AXI magistrale i oni su redom *fracr_axi_o, fracc_axi_o, spacing_axi_o , i_cose_axi_o , i_sine_axi_o.*

Zatim narednih 5 je napravljeno da vrednosti koje su tima num_i tj one su 11obitne I zato imamo po jedan registar za njih I to su *iy_axi_o, ix_axi_o, step_axi_o, scale_axi_o.*

Slv_reg_15 čuva vrednosti signala *start_i_axi*, u njemu se skladišti 0 kao deo kontrolnog signala.

Slv_reg_16 je takodje kontrolni signal i čuva vrednosti *ready_axi_i.*

```
fracr_axi_o <= slv_reg0(UPPER_SIZE -1 downto 0) & slv_reg1(LOWER_SIZE - 1 downto 0);


fracc_axi_o <= slv_reg2(UPPER_SIZE -1 downto 0) & slv_reg3(LOWER_SIZE - 1 downto 0);


spacing_axi_o <= slv_reg4(UPPER_SIZE -1 downto 0) & slv_reg5(LOWER_SIZE - 1 downto 0);


i_cose_axi_o <= slv_reg6(UPPER_SIZE -1 downto 0) & slv_reg7(LOWER_SIZE - 1 downto 0);


i_sine_axi_o <= slv_reg8(UPPER_SIZE -1 downto 0) & slv_reg9(LOWER_SIZE - 1 downto 0);

-- Povezivanje ostalih signala direktno sa registrima
iradius_axi_o <= slv_reg10(WIDTH - 1 downto 0);
iy_axi_o <= slv_reg11(WIDTH - 1 downto 0);
ix_axi_o <= slv_reg12(WIDTH - 1 downto 0);
step_axi_o <= slv_reg13(WIDTH - 1 downto 0);
scale_axi_o <= slv_reg14(WIDTH - 1 downto 0);
start_i_axi <= slv_reg15(0);

 slv_reg16 <= std_logic_vector(to_unsigned(0, 31)) & ready_axi_i;
```

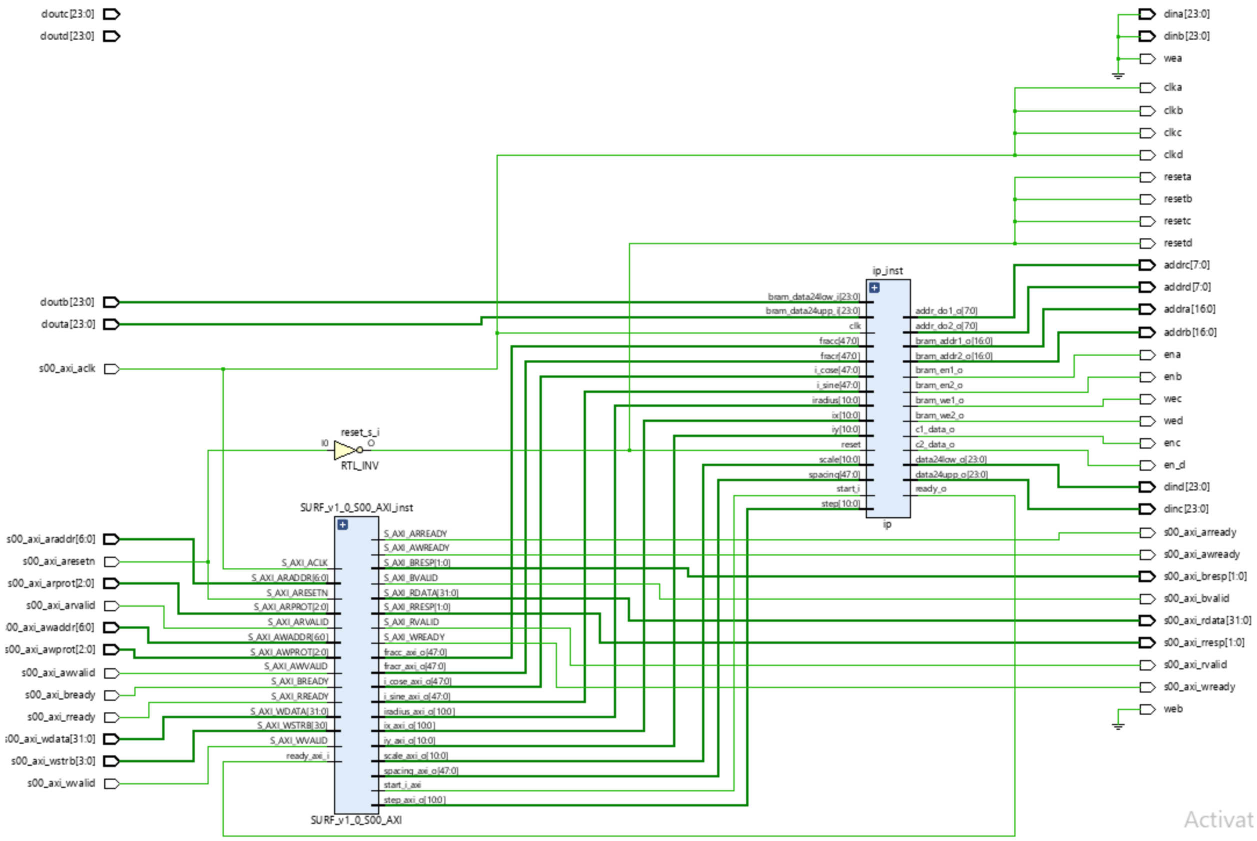
Slika 3Povezivanje memorijskog podsistema i ip-a

IP modul SURF_V1_0 koristi AXI interfejs za prijem konfiguracije i kontrolnih signala od procesora. Ovo omogućava korisnik da dinamički kontroliše parametre obrade, kao i da inicira i nadzire status obrade preko standardizovanog komunikacijskog interfejsa, poboljšavajući modularnost i skalabilnost sistema.

```
-- Users to add ports here
--------------MEM INTERFEJS ZA SLIKU--------------------
clka       : out std_logic;
reseta     : out std_logic;
ena        : out std_logic;
addra      : out std_logic_vector (PIXEL_SIZE - 1 downto 0);
dina    : out std_logic_vector (BRAM_24_DATA - 1 downto 0);
douta : in std_logic_vector (BRAM_24_DATA - 1 downto 0);
wea        : out std_logic;
--------------MEM INTERFEJS ZA SLIKU--------------------
clkb       : out std_logic;
resetb     : out std_logic;
enb        : out std_logic;
addrb      : out std_logic_vector (PIXEL_SIZE - 1 downto 0);
dinb    : out std_logic_vector (BRAM_24_DATA - 1 downto 0);
doutb : in std_logic_vector (BRAM_24_DATA - 1 downto 0);
web        : out std_logic;
--------------MEM INTERFEJS ZA IZLAZ--------------------
clkc       : out std_logic;
resetc     : out std_logic;
enc        : out std_logic;
addrc      : out std_logic_vector (INDEX_ADDRESS_SIZE-1 downto 0)
dinc    : out std_logic_vector (BRAM_24_DATA - 1 downto 0);
doutc : in std_logic_vector (BRAM_24_DATA - 1 downto 0);
wec        : out std_logic;
--------------MEM INTERFEJS ZA IZLAZ--------------------
clkd       : out std_logic;
resetd     : out std_logic;
en_d        : out std_logic;
addrd      : out std_logic_vector (INDEX_ADDRESS_SIZE-1 downto 0)
dind    : out std_logic_vector (BRAM_24_DATA - 1 downto 0);
doutd : in std_logic_vector (BRAM_24_DATA - 1 downto 0);
wed        : out std_logic;
-- User ports ends
```

```
-- Ports of Axi Slave Bus Interface S00_AXI
s00_axi_aclk    : in std_logic;
s00_axi_aresetn : in std_logic;
s00_axi_awaddr  : in std_logic_vector(C_S00_AXI_ADDR_WIDTH-1 downto 0);
s00_axi_awprot  : in std_logic_vector(2 downto 0);
s00_axi_awvalid : in std_logic;
s00_axi_awready : out std_logic;
s00_axi_wdata   : in std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
s00_axi_wstrb   : in std_logic_vector((C_S00_AXI_DATA_WIDTH/8)-1 downto 0);
s00_axi_wvalid  : in std_logic;
s00_axi_wready  : out std_logic;
s00_axi_bresp   : out std_logic_vector(1 downto 0);
s00_axi_bvalid  : out std_logic;
s00_axi_bready  : in std_logic;
s00_axi_araddr  : in std_logic_vector(C_S00_AXI_ADDR_WIDTH-1 downto 0);
s00_axi_arprot  : in std_logic_vector(2 downto 0);
s00_axi_arvalid : in std_logic;
s00_axi_arready : out std_logic;
s00_axi_rdata   : out std_logic_vector(C_S00_AXI_DATA_WIDTH-1 downto 0);
s00_axi_rresp   : out std_logic_vector(1 downto 0);
s00_axi_rvalid  : out std_logic;
s00_axi_rready  : in std_logic
```

Slika 4Izgled port mape za AXI lite protokol



Slika 5Izgled AXI lite protokola za nas sistem

Takođe treba predstaviti utrošene resurse za ovaj sistem, gde se očekuje njihovo smanjenje, što će naredna slika i pokazati.

Resource	Estimation	Available	Utilization...
LUT	6433	17600	36.55
LUTRAM	598	6000	9.97
FF	2985	35200	8.48
DSP	14	80	17.50

Slika 6Utroseni resursi za AXI lite protokol

8.Frekvencija rada sistema i kritična putanja za AXI lite

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0,477 ns	Worst Hold Slack (WHS):	0,097 ns	Worst Pulse Width Slack (WPWS):	5,020 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4218	Total Number of Endpoints:	4218	Total Number of Endpoints:	3595

All user specified timing constraints are met.

Name	Waveform	Period (ns)	Frequency (MHz)
s00_axi_aclk	{0.000 6.000}	12.000	83.333

Slika 8-1 WNS i najveća frekvencija AXI lite sistema

Source	ip_inst/c_inc_dsp/u3_1_reg_reg[7]/C (rising edge-triggered cell FDRE clocked by s00_axi_aclk {rise@0.000ns fall@6.000ns period=12.000ns})
Destination	ip_inst/temp4_cpos_inc_dsp/res_reg_reg[29]/D (rising edge-triggered cell FDRE clocked by s00_axi_aclk {rise@0.000ns fall@6.000ns period=12.000ns})

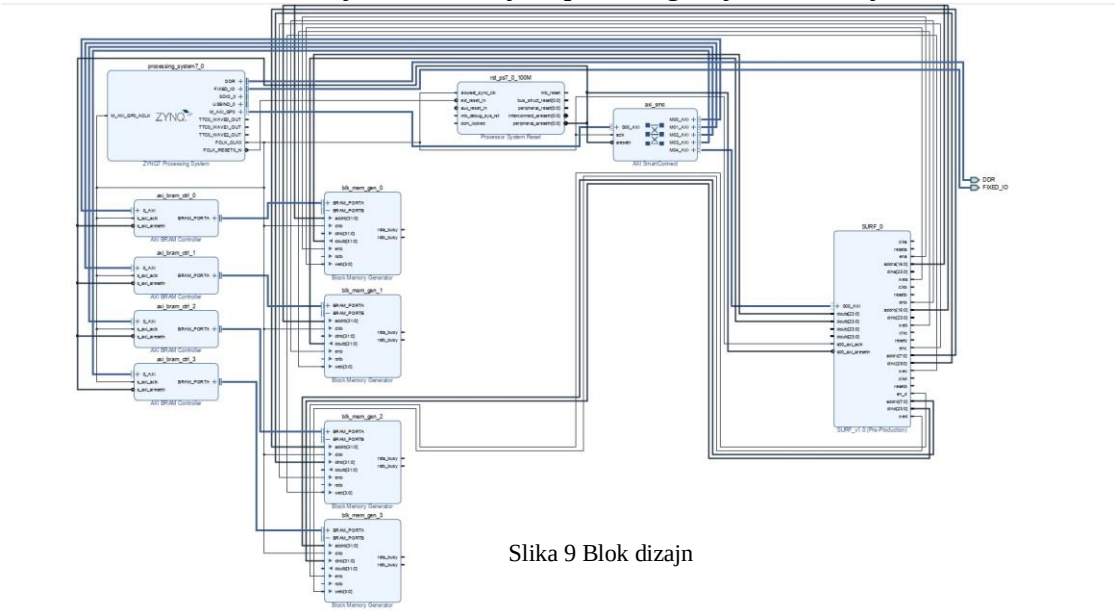
Slika 8-2Kritična putanja sistema

Na osnovu osnovu slika 8-1 i 8-2 možemo zaključiti da je frekvencija smanjena na **83,333 MHZ,** a kritična putanja je se nalazi od registra u3_1_reg koji je instanciran u c_inc_dsp do res_reg koji je instanciran u temp4_cpos_inc_dsp

9.Blok dizajn

Poslednja faza je prikaz interfejsa u blok dizajnu. U dizajn je prvo ubacen processing_system 7_0 koja predstavlja procseor sistema.

Zatim je ubačena SURFV1_0 jedinica, kao i 4 AXI Bram kontrolera koji imaju po jedan port, koji je podešen za AXI4 protokol i kojim se komunicira sa procesorskom jedinicom. Takođe napravljeni su i 4 blok memorijska generatora koji će dobijati vrednosti koje računamo i na izlazu u indeks nizu brojeva u dva fajla upisivati gornjih 24 i donjih 24 bita.



Slika 9 Blok dizajn