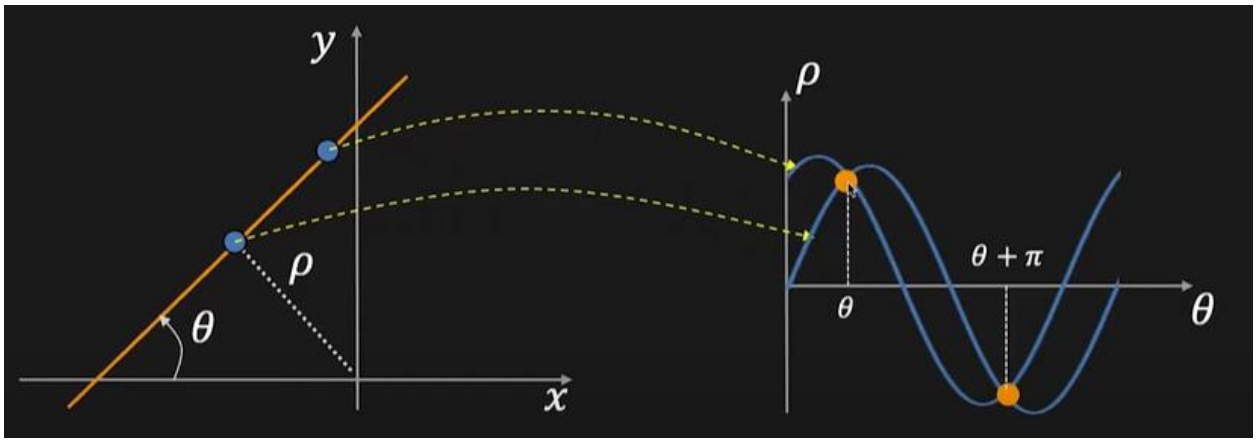
Пројекат: Детектовање правих линија на фотографији и њихово исцртавање

Документација за предмет: Пројектовање сложених дигиталних система

1. Опис алгоритма

Тема нашег пројекта јесте детектовање правих линија (праваца) на фотографији која се прослеђује програму, те њихово исцртавање. Централни део алгоритма јесте „Hough“ трансформација, која представља једну линеарну трансформацију. Принцип трансформације јесте следећи: Свака тачка (пиксел) из једног домена (наша фотографија) се пресликава у праву у другом домену. Информације о тим правама се складиште унутар акумулатора, који је представља матрицу, у којој је једна димензија коефицијент правца, а друг пресек са y-осом трансформисане праве. Међутим, ту се наилази на проблем, јер коефицијент правца може да се налази у интервалу од – бесконачно до + бесконачно, те се зато примењује другачији облик алгоритма. У овом алгоритму се користи тригонометријски облик праве, где је она одређена са растојањем од координатног почетка (ρ) и углом под којим сече x-оцу (Θ). Зато се у овом случају добија акумулатор, у којем је једна димензија додељена за тета, а друга за ро. У овом случају почетног проблема нема, јер је угао ограничен (0 до 359). Заправо, (за сада) довољно је да се посматрају само углови од 0 до 179, јер се праве са угловима Θ и Θ + 180 поклапају. Иницијална вредност свих поља унутар акумулатора је нула, а током рада алгоритма он се испуњава информацијама о трансформисаним линијама. На крају трансформације унутар акумулатора ће се наћи велике вредности у пољима са информацијама о правим уочљивим на оригиналној фотографији.

За низ углова Θ добијају се одговарајуће вредности ρ. Ипоставља се да низ ових вредности даје синусоиду (као на слици 1). За две тачке ћемо добити две синусоиде, а пресек тих синусоида даје ρ и Θ праве која пролази кроз те две тачке.



Слика 1. Hough трансформација

Изводи се де формула за рачунање ρ : ρ(Θ) = x*sin(Θ) - y*cos(Θ). Међутим, по овој формули када права сече x осу у тачки где је x < 0. Пошто је ρ растојање (за које се подразумева да је позитивно), могла би се наћи апсолутна вредност од ρ те увећати одговарајуће поље акумулатора. Међутим, треба обратити пажњу да постоји права која x-осу сече под истим углом за случај x > 0, те бисмо изгубили информацију о којој се заправо правој ради.

Зато се за ρ < 0 узима његова апсолутна вредност, али се и вредност Θ повећава за 180 (што има смисла, јер је то и даље иста права) те се проблем превазилази. Ситуацију ρ < 0 је могуће добити само за Θ < 90 (права сече x осу у тачки где је x < 0 ,права са тупим улом у том случају не би пролазила кроз први квадрант, који заправо и репрезентује фотографију, тако да та права није од интереса). Зато се у акумулатору резервише простор за Θ од 0 до 269 (уместо првобитних 179).

2. Имплементација алгоритма у C језику

```
for (p = img, y = 0; y < height; y++)
{
    for (x = 0; x < width; x++, p++)
    {
        if (*p > transform_threshold)
        {
            for (theta = 0; theta < THETA; theta++)
            {
                sin_pom = sin(theta*M_PI/180);
                cos_pom = cos(theta*M_PI/180);
                rho = x*sin_pom - y*cos_pom;
                if (rho >= 0)
                {
                    acc[theta][rho]++;
                }
                else
                {
                    acc[THETA + theta][-rho]++;
                }

            }
        }
    }
}
```

Слика 2.Имплементација алгоритма у C језику

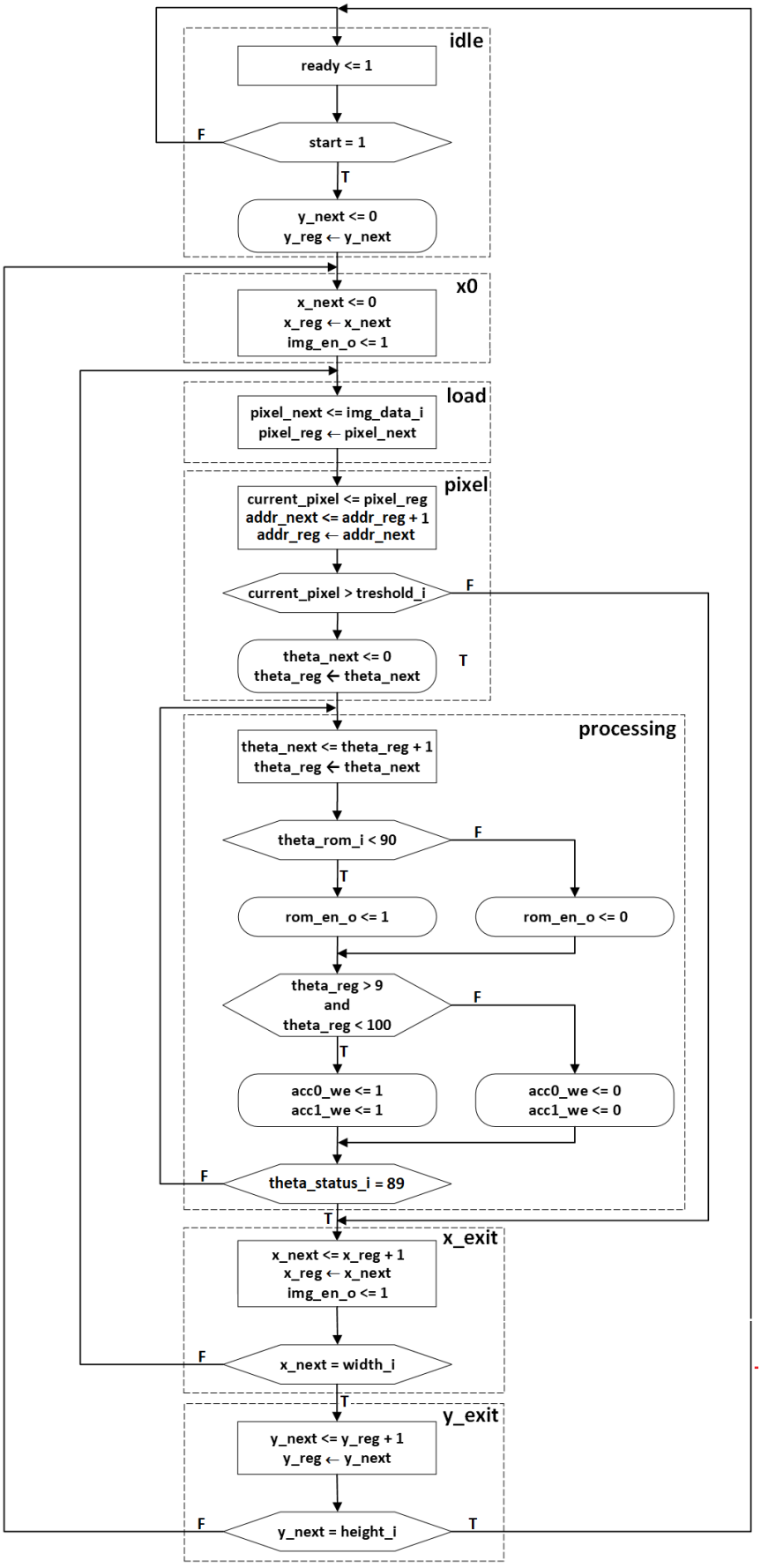
3. Уклањање петљи из алгоритма

```
idle:
        p = img;
        y = 0;
x0:
            x = 0;

pixel:
                current_pixel = *p;
                if (current_pixel > transform_treshold_i) then
                    theta = 0;
                    goto processing;
                else
                    goto x_exit;
processing:
                    sin_pom = sin(theta*M_PI/180);
                    cos_pom = cos(theta*M_PI/180);
                    rho = x*sin_pom - y*cos_pom;
                    if (rho >= 0) then
                        acc[theta][rho] = acc[theta][rho] + 1;
                    else
                        acc[THETA + theta] = acc[THETA + theta][-rho] + 1;
                    theta = theta + 1;
                    if (theta = 180) then
                        goto x_exit;
                    else
                        goto processing;
x_exit:
                x = x + 1;
                if (x = width_i) then
                    goto y_exit;
                else
                    goto pixel;
y_exit:
            y = y + 1;
            if (y = height_i) then
                goto stop;
            else
                goto x0
stop:
        nop;
```

Слика 3.Имплементација алгоритма без петљи

4. ASM дијаграм



Слика 4. ASM дијаграм

Коментар: Када се пореде ASM дијаграм и размотана петља види се да је на дијаграму једно стање више (***load***). Оно је уведено ради побољшања перформанси као стање где се пиксел учитава у регистар, одакле ће се потом проверавати да ли је испуњен услов за трансформацију.

Систем је оптимизован на два начина: ρ(Θ)

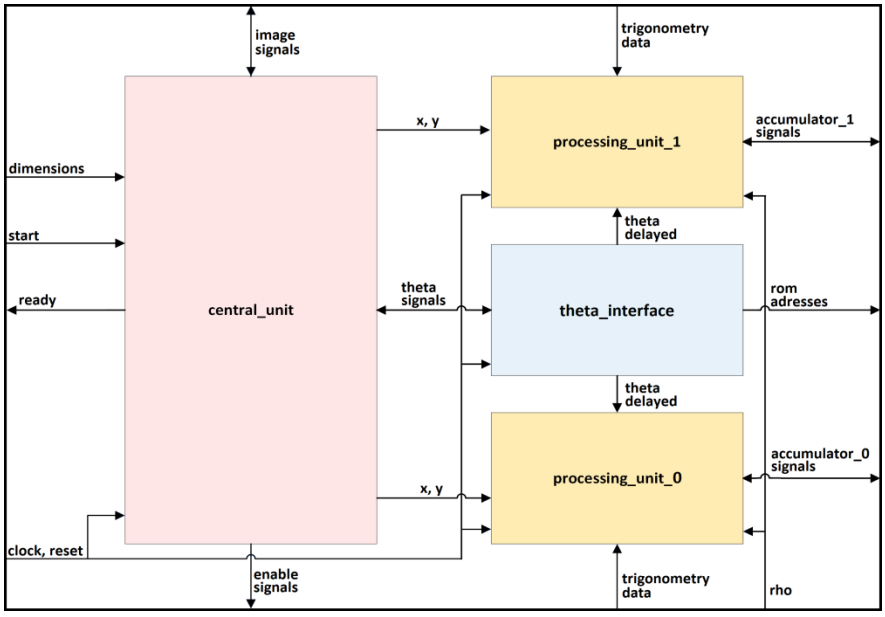
1. Искоришћена је метода делимичног одмотавања петље са фактором два (Partial loop unrolling). Уместо за једну, у систему ће се паралелно рачунати ρ за две вредности Θ (једна парна, друга непарна, почев од пара 0 и 1 па надаље). У складу с тим и акумулатор је подељен на два дела, за парне и непарне вредности . Да не би изашло ван опсега и Θ потребно за генерисање адресе акумулатора се полови, те се мења од 0 до 89, а уколико добијемо негативно ρ, тренутно Θ ће се увећати за 90, односно највећа могућа вредност за ново Θ јесте 134. За адресу ROM-а (у ком су смештене тригонометријске вредности), користе се реалне вредности Θ, и то на тај начин што се оно које се мења помножи са 2 (за добијање парних), те се на ту помножену вредност додаје 1 (за добијање непарних).

2. Искоришћена је и проточна обрада којом је на фазе подељен поступак рачунања ρ, генерисање адресе за акумулатор, читање из акумулатора, увећавање учитане вредности и њен испис назад.

4. Блок схеме

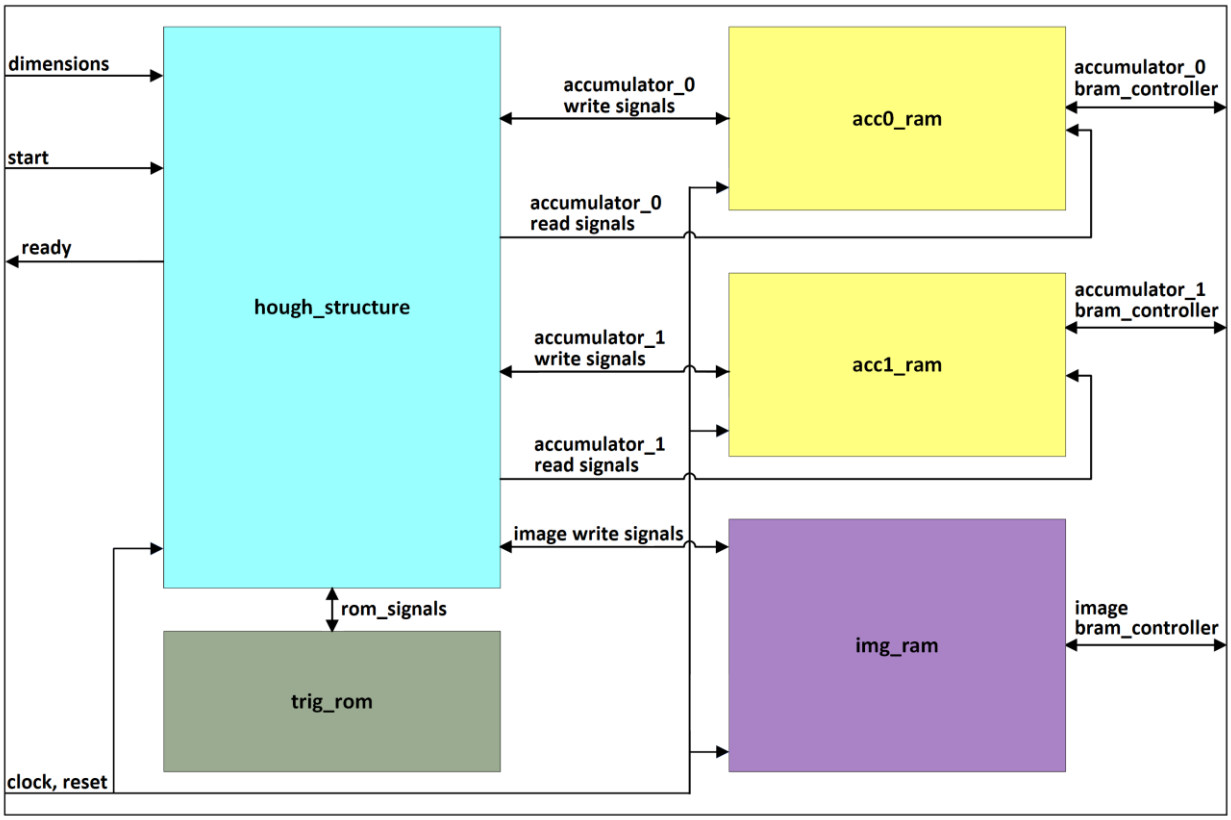
Структура која врши тражену функционалност се назива *hough_structure* и она се састоји из неколико делова:

- *central_unit* – у оквиру које је моделован део data-path са регистрима у које су смештене променљив x, y, theta, img_addr (алгоритам пролази редом пиксел по пиксел, тако да за генерисање адресе није потребно ништа више од периодичног инкрементовања) и pixel
- *processing_unit* (0 и 1) – јединица у оквиру које је имплементирана проточна обрада, те у систему постоје две идентичне целине, за непарне и парне вредности theta.
- *theta_interface* – јединица уоквиру које се добијају реалне вредности углова које служе за адресе ROM-а. Осим тога, у овој јединици се закашњава theta у складу са фазама проточне обраде из претходно описане јединице и из ње се шаљу сигнали потребни за генерисање статусних сигнала који се прослеђују *control-path*-у. Блок схема је дата на слици 5:



Слика 5. Блок схема „hough _structure“

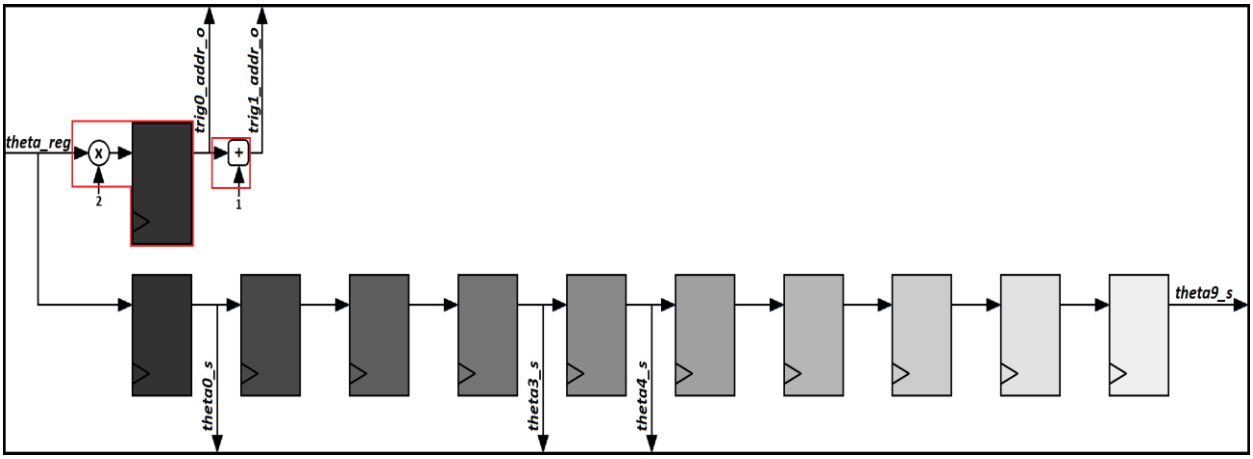
Ова структура је даље повезана са околним меморијама ROM (у којој су тригонометријске вредности), два акумулатора и меморија за фотографију. Пиксели се читају само са једног порта, док је код акумулатора ситуација таква да се са једног порта чита, док се на други паралалено уписују вредности. Један порт меморије за фотографију је намењен BRAM-Controller-у, док код акумулатора један порт потпуно припада hough_structure-и, док други дели са BRAM-Controller-ом. Мимо меморија, језгру се прослеђују и информације о димензијама (*dimensions*) фотографије на основу које систем и ради, те сигнали за почетак и провере да ли је систем спреман за рад (*star*t и *ready*)Блок схема овог система се налази на слици 6.



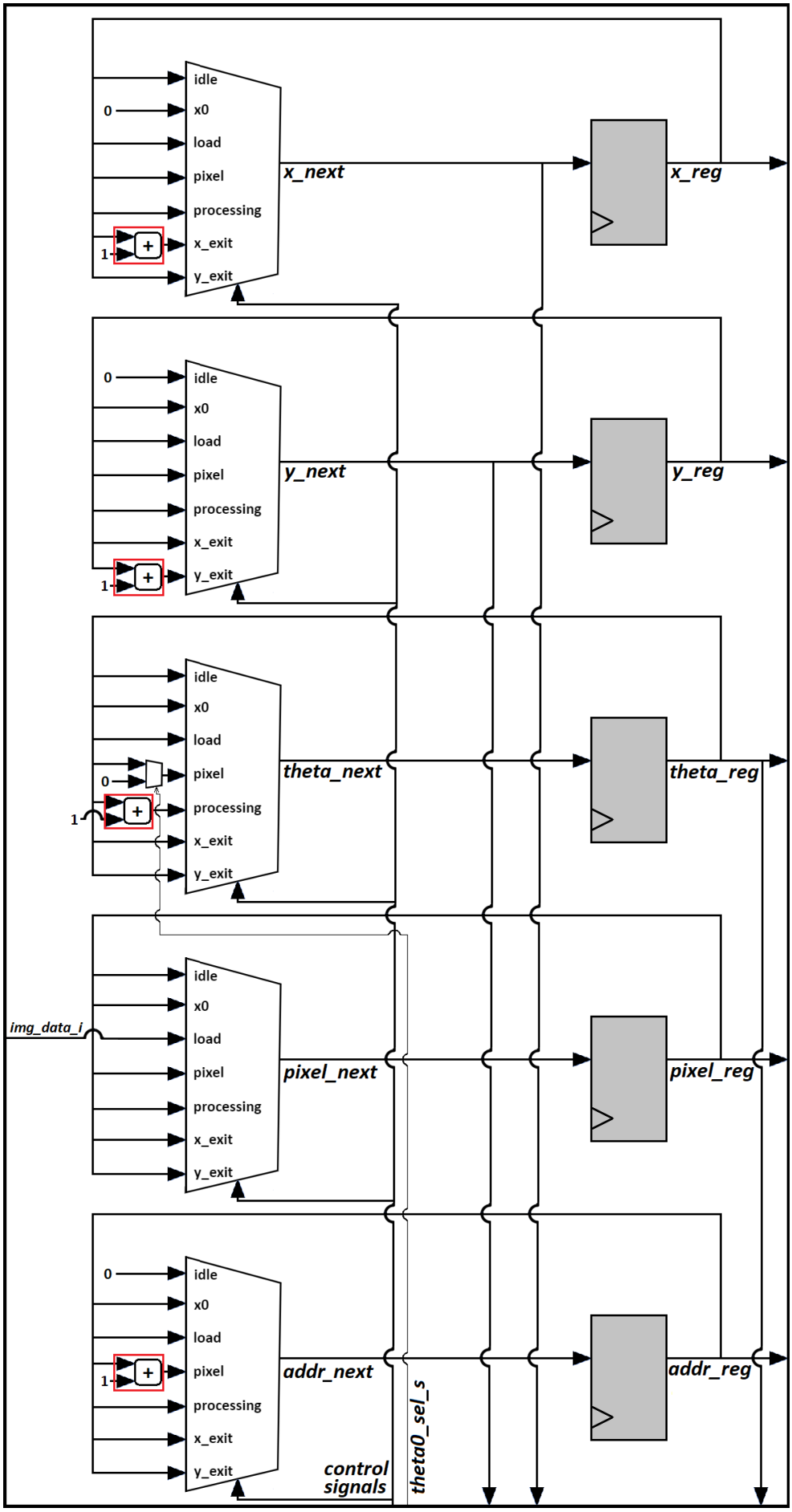
Слика 5. Блок схема „hough _core“

6. Детаљи дизајна

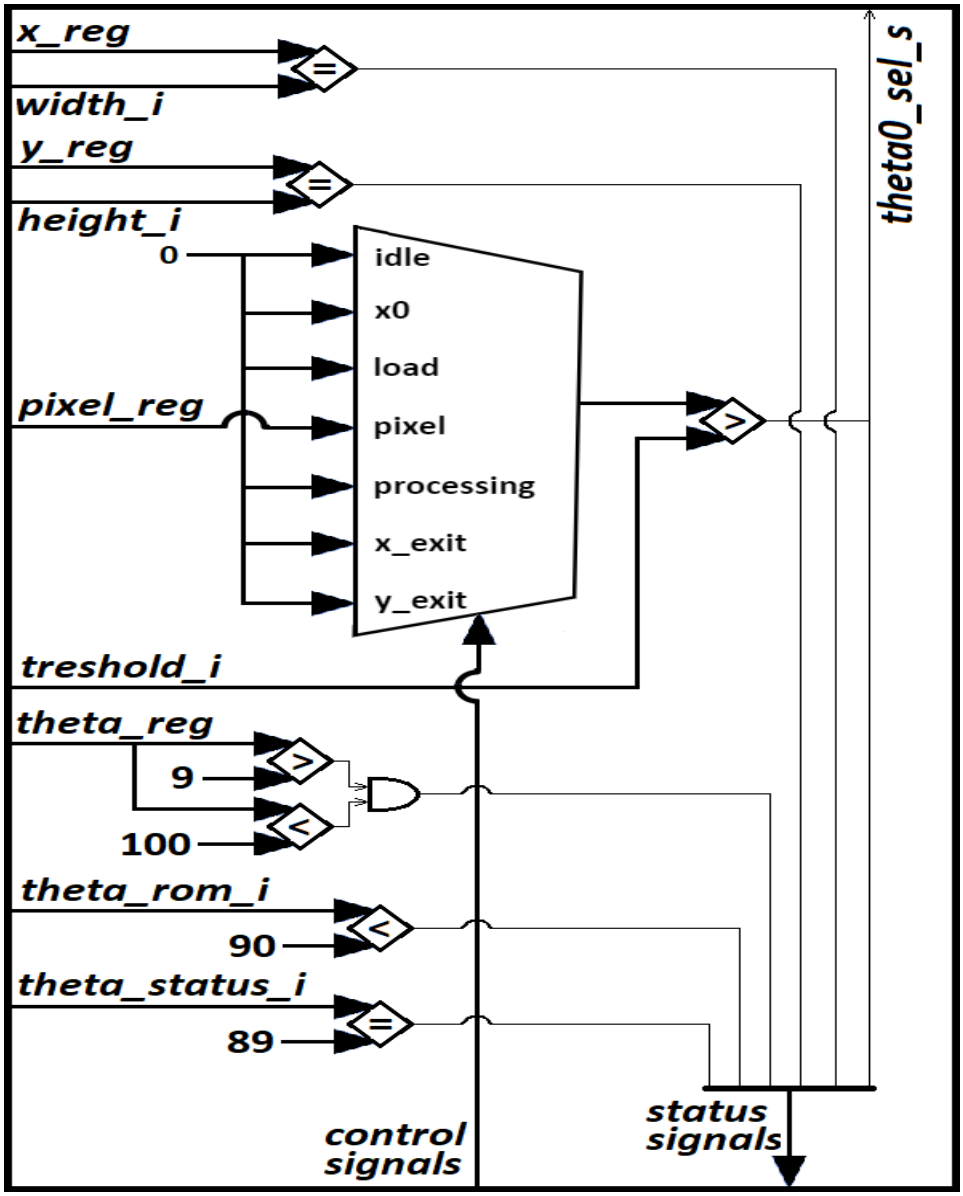
На сликама у наставку су, ради бољег увида у дизајнм, представљене детаљније шеме набројаних јединица унутар *hough_structure.*



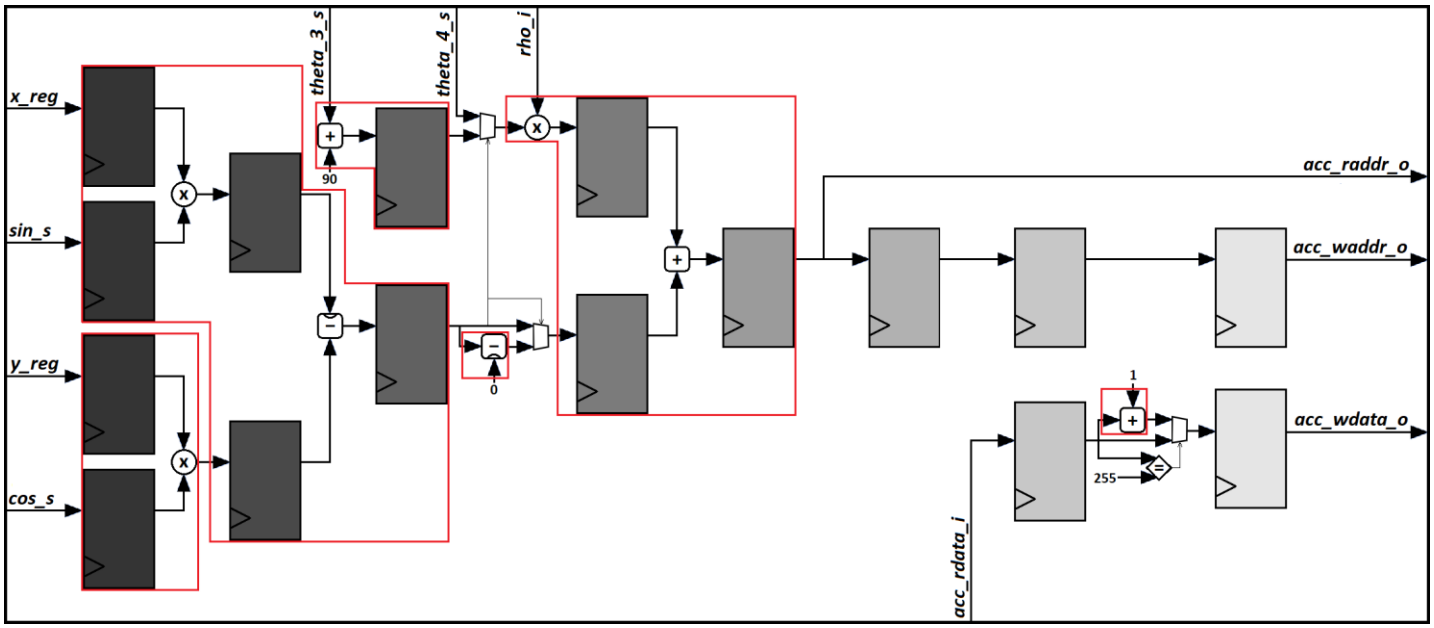
Слика 6. „theta_interface“



Слика 7. Регистарске променљиве (из „central_unit“)



Слика 8. Генерисање статусних сигнала



Слика 9. „processing_unit“

На претходним сликама се уочавају и извесне целине уоквирене црвеном бојом. Оне представљају искоришћене „DSP“ јединице. У табели 1. Дат је детаљан преглед који се поклапа са резултатима синтезе). Треба имати на уму да су у систему две јединице за обраду, те се зато њихов број обрачунава два пута.

функционална јединица	central_unit	theta_interface	processing_unit (x2)	УКУПНО
број *DSP* -ева	4	2	6 (x2)	18

Табела 1. Преглед искоришћених DSP јединица

Меморија за фотографију, акумулаторе и тригонометријске вредности се мапира на „BRAM“ јединице (величине 4KB) и то на начин приказан у табели 2 (величине су одређене у складу са потребама алгоритма). Ипак, с обзиром на ограниченост ресурсима, а великом потребом за меморијом (посебно због акумулатора), наш систем је у стању да обрађује фотографије чија дијагонала није већа од 424 пиксела.

функционална јединица	trig_rom	img_ram	acc_ram (x2)	УКУПНО
број *BRAM*-ова	1	24	16 (x2)	57

Табела 2. Преглед искоришћених BRAM јединица

На слици 10 је резултат искоришћених ресурса након имплементације.

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	195	17600	1.11
LUTRAM	4	6000	0.07
FF	280	35200	0.80
BRAM	57	60	95.00
DSP	18	80	22.50

Слика 10. Искоришћени ресурси

7. Фреквенција рада и критична путања *hough_core*-а

Clock Summary

Name	Waveform	Period (ns)	Frequency (MHz)
clk	{0.000 3.500}	7.000	142.857

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0,028 ns	Worst Hold Slack (WHS):	0,156 ns	Worst Pulse Width Slack (WPWS):	2,520 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:	3119	Total Number of Endpoints:	2329	Total Number of Endpoints:	416

All user specified timing constraints are met.

Слика 11. Информације о тајмингу

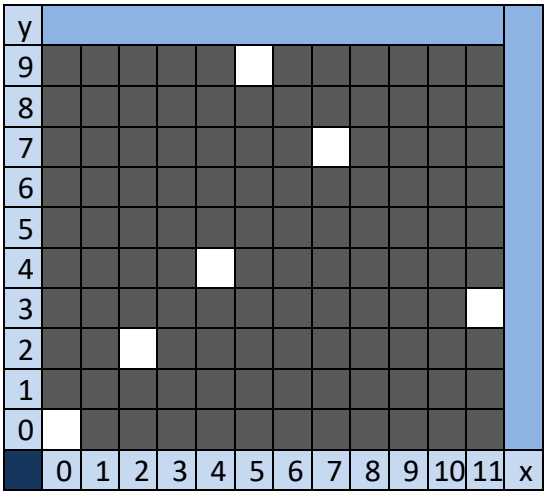
На тренутном нивоу дизајна систем може да ради на фреквенцији од 142.857 (као што је и приказано на слици 11). Уз додатне јединице, могућ је пад фреквенције, те ће се крајњи исход знати послије. Пошто је синтеа рађена у моду out_of_context, подаци о критичној путањи су непоуздани јер се односе на путању од *img_bram-a* до излаза *img_data_cont_o*.

Name	Slack ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	0.028	2	1	img_ram/RAM_reg_0_1/CLKARDCLK	img_data_cont_o[1]	4.964	3.421	1.543	7.0	clk	clk
Path 2	0.028	2	1	img_ram/RAM_reg_0_6/CLKARDCLK	img_data_cont_o[6]	4.964	3.421	1.543	7.0	clk	clk

Слика 12. Критична путања

8. Тестбенч

За систем је направљено и једноставно верификационо окружење ради провере функционалности. Систему се прослеђује фотогарфија формата 12x10 (за те димензије одговарајуће улазно РО износи 15), претежно тамна са само пар светлих пиксела, те акумулатор(и) потребне величине.



Слика 13. Изглед улазне фотографије са помоћним координатама

x	y	theta_real	theta_alg	rho	rho_red	theta_red	addr
7	7	135 = 2*67 + *1*	67	9.899 → +9	9	67 = 67+0 (+)	1014 = 67*15 + 9 (*1*)
7	7	60 = 2*30 + *0*	30	2.56 → +2	2	30 = 30+0 (+)	452 = 30*15 + 2 (0)
7	7	30 = 2*15 + *0*	15	-2.56 → -3	3	105 = 15+90 (-)	1578 = 105*15 + 3 (0)
7	7	45 = 2*22 + *1*	22	0 → +0	0	22 = 22+0 (+)	330 = 22*15 + 0 (*1*)
11	3	135 = 2*67 + *1*	67	9.899 → +9	9	67 = 67+0 (+)	1014 = 67*15 + 9 (*1*)

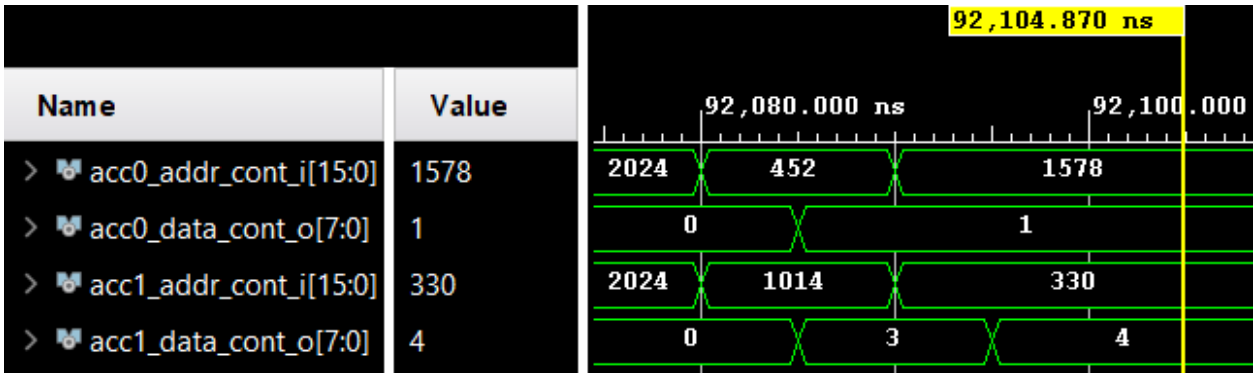
Табела 3. Прорачун адреса за акумулаторе

У табели 3 је представљен прорачун ро-а и адреса акумулатора у зависности од вредности угла за пар примера. За пикселе са координатама (7, 7) и (11, 3) се за угао од 135 степени поклапају праве, те се у адреси и види да се за та два примера приступа истим адресама акумулатора (у колони **addr** табееле је унутар заграда означено којем од акумулатора се за дати случају приступа). Осим тога, исти резултат се очекује и за пиксел (5, 9), те би на крају трансформације на адреси 1014 акумулатора 1 требало да остане вредност 3.

Слично, за угао 45 степени се очекује једнако ро (=0) за пикселе (0, 0), (2, 2), (4, 4) и (7,7), те и вредност 4 на адреси 330 акумулатора 1 на крају трансформације.

Такође, према резултату из табеле 3 на адресама 452 и 1578 акумулатора 0 очекују се вредности 1, јер се права под истим углом ни од једног другог пиксела не треба поклапати са посматраним.

Резултати симулације потврђују очекивани резултат, као што се и види на слици 14. Наравно, у акумулатору има још мноштво вредности (пре свега 0) које остају да се провере, што би био дуг и сложен поступак, а наведени опис је само илустрација да систем испуњава основну спецификацију.



Слика 14. Резултати симулације у Виваду

9. Паковање *IP*-језгра

Да би се омогућила комуникација са процесором, потребно је успоставити неки одговарајући протокол који је обезбеђен на *Zybo* платформи. За овај пројекат је искоришћен AXI-Lite протокол за упис и читање одговарајућих регистара, који се налазе у меморијском подсистему који служи као интерфејс између AXI-Lite улазно-излазног контролера и јединице hough_core. Регистри за упис и читање су: start, reset, width, height, rho и threshold, док се из ready регистар може само читати. Сегмент кода на слици 15 показује начин на који су повезани поменути регистри са одговарајућим периферијама hough_core

```
-- Add user logic here
start_o <= slv_reg0(0);
reset_o <= slv_reg1(0);
width_o <= slv_reg2(DIMENSIONS_WIDTH - 1 downto 0);
height_o <= slv_reg3(DIMENSIONS_WIDTH - 1 downto 0);
rho_o <= slv_reg4(XY_WIDTH - 1 downto 0);
treshold_o <= slv_reg5(TRESHOLD_WIDTH - 1 downto 0);
slv_reg6 <= std_logic_vector(to_unsigned(0, C_S_AXI_DATA_WIDTH - 1)) & ready_i;
-- User logic ends
```

Слика 15. Повезивање меморијског подсистема и сигнала hough_core-а

С друге стране, што се тиче портова намењених за унутрашње брамове, они су и даље остали слободни, јер је, као што је и раније наведено у документу, предвиђено да се они попуњавају путем *AXI-BRAM* контролера.

На слици 16 дат је преглед порта на највишем топ-модула запакованог језгра. Он се састоји из два дела: портова за AXI-Lite интерфејс, те портова за брамове који се налазе унутар језгра.

```
    port (
        -- Users to add ports here
        acc0_addr_cont_i: in std_logic_vector(ACC_ADDR_WIDTH - 1 downto 0);
        acc0_data_cont_o: out std_logic_vector(BUS_DATA_WIDTH - 1 downto 0);
        acc0_data_cont_i: in std_logic_vector(BUS_DATA_WIDTH - 1 downto 0);
        acc0_we_cont: in std_logic_vector(3 downto 0);
        acc1_addr_cont_i: in std_logic_vector(ACC_ADDR_WIDTH - 1 downto 0);
        acc1_data_cont_o: out std_logic_vector(BUS_DATA_WIDTH - 1 downto 0);
        acc1_data_cont_i: in std_logic_vector(BUS_DATA_WIDTH - 1 downto 0);
        acc1_we_cont: in std_logic_vector(3 downto 0);
        img_addr_cont_i: in std_logic_vector(IMG_ADDR_WIDTH - 1 downto 0);
        img_data_cont_o: out std_logic_vector(BUS_DATA_WIDTH - 1 downto 0);
        img_data_cont_i: in std_logic_vector(BUS_DATA_WIDTH - 1 downto 0);
        img_we_cont: in std_logic_vector(3 downto 0);
        -- User ports ends
        -- Do not modify the ports beyond this line


        -- 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
    );
end Hough_IP_v1_0;
```

Слика 16. Порт запакованог језгра

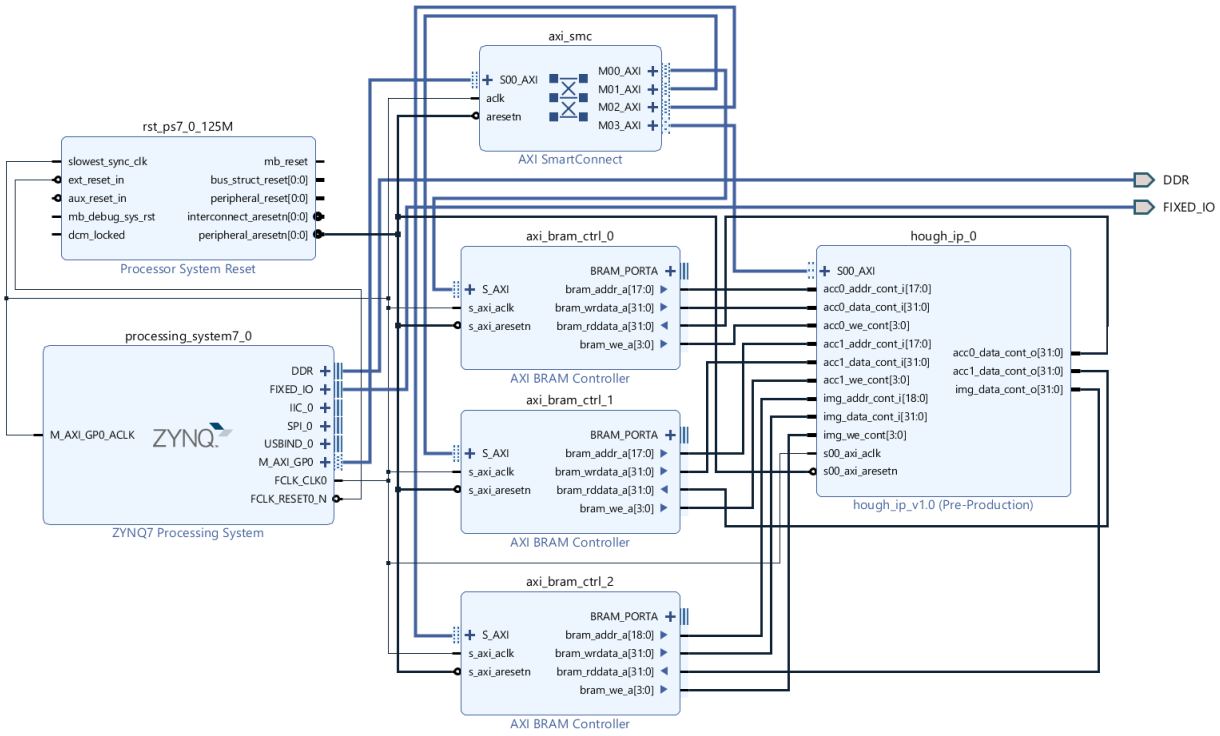
На крају су додате неке основне информације о језгру, међу којима су најважнији његов назив *hough_ip* и назив који се приказује (*display_name*) *hough_ip_v1.0*.

10. Блок-дизајн

Последња фаза јесте креирање блок-дизајна. У дизајн је прво убачена компонента Zynq processing_system7_0, која представља процесор. Затим су извршена извесна подешавања ове компоненте. Омогућен је рад разних периферија, као што су SPI0, I2C0, UART1, GPIO, SD0, USB0, ENET0 и QUAD SPI. Осим тога, подешена је и фреквенција језгра на 125MHz, а након процеса имплементације је и утврђено да она одговара систему.

Поред њега, додата је и јединица hough_ip_0, као и три AXI-BRAM контролера, од којих је сваки предодређен за један брам унутар система. Свака BRAM јединица је подешена на кевивалентан начин: садржи један порт, подешен је AXI4 протокол за комуникацију са процесорком јединицом, те је уз то и омогућен AXI-Burst режим рада. Такође, подешени су и адресни опсези за сваки од њих.

Након тога, извршено је аутоматско повезивање јединица унутар система (све осим веза hough_core и BRAM констролера, које су извршене ручно). На крају је систем добио изглед као на слици 17:



Слика 17. Блок дизајн

11. Фреквенција рада и критична путања целог система

Након извршене синтезе и имплементације показало се да систем може да ради на фреквенцији која износи 125MHz. Што се тиче критичне путање, она се налази између регистра стања у оквиру FSM-а и img-брама, као што је приказано на слици 18.

Source	hough_i/hough_ip_0/U0/my_core/hough_unit/central_unit/FSM_sequential_state_reg_reg[0]/C
Destination	hough_i/hough_ip_0/U0/my_core/img_ram/RAM_reg_0_6/ENBWREN

Слика 18. Критична путања

12. Тестирање рада у *Vitis*-у

Након синтезе и имплементације, генерисан је и *Bitstream* , те је систем спуштен на уређај без оперативног система и написана је *bare-metal* апликација.

Унутар ње, упис у регистре се извршава путем функције (**Xil_Out8** и **Xil_Out16**) у зависности од дужине података који се прослеђује, док се ишчитавање врши функцијом **Xil_Out32**. На почетку кода, генерисани су макрои за адресе регистара , као на слици 19.

```
#define START_REG_OFFSET 0
#define RESET_REG_OFFSET 4
#define WIDTH_REG_OFFSET 8
#define HEIGHT_REG_OFFSET 12
#define RHO_REG_OFFSET 16
#define TRESHOLD_REG_OFFSET 20
#define READY_REG_OFFSET 24
```

Слика 19. Регистарски макрои

Што се тиче попуњавања и ишчитавања *BRAM*-ова, с обзиром на то да је омогућен *Burst*-режим, на тај начин је и реализован у овом окружењу. Он је омогућен потем команде **Xil_SetTlbAttributes(XPAR_AXI_BRAM_CTRL_n_S_AXI_BASEADDR, 0x15de6);** (уместо n, уписју се 0, 1 или 2 за сваки од *BRAM*-ова), док се за трансфер података користи команд *memcpy*.

13. Скрипта

Да би се цео систем реализовања система аутоматизовао, креирана је и *TCL* скрипта. Она се састоји од учитавања свих потребних фајлова, паковања језгра, креирања блок дизајна, подешавања и повезивања компоненти, те на крају синтезе, имплементације, генерисања Bitstream-а и његовог отпремања.