Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

6η Σειρά Ασκήσεων

Μάθημα: Ψηφιακά Συστήματα VLSI

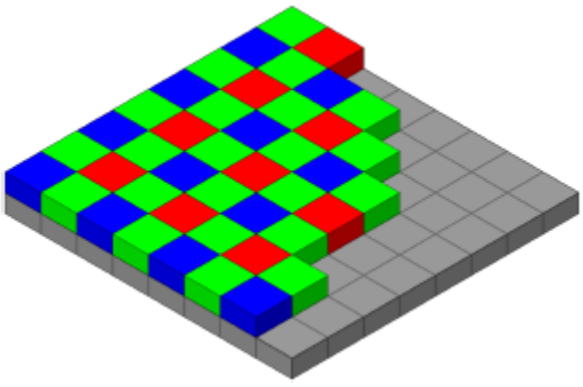
Εξάμηνο: 8ο

Ονοματεπώνυμα: Αλεξοπούλου Γεωργία, Γκενάκου Ζωή

Υλοποίηση Debayering Φίλτρου σε FPGA

Εισαγωγή

Μία συστοιχία χρωματικών φίλτρων (Color Filter Array) αποτελεί ένα μωσαϊκό από μικροσκοπικά χρωματικά φίλτρα, το οποίο τοποθετείται πάνω από τα pixels των αισθητήρων κάμερας ώστε να συλλάβει τις πληροφορίες χρώματος. Ένα από τα πιο ευρέως χρησιμοποιούμενα μωσαϊκά είναι το Bayer, το οποίο παρουσιάζεται στο παρακάτω σχήμα. Η κατανομή των χρωματικών φίλτρων στο μωσαϊκό Bayer είναι ως εξης: 50% πράσινα, 25% κόκκινα, και 25% μπλε.



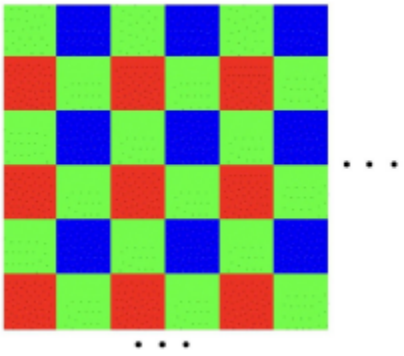
Για κάθε pixel αποθηκεύεται μόνο μία από τις τρεις χρωματικές συνιστώσες, η οποία προσδιορίζεται από το αντίστοιχο χρωματικό φίλτρο. Υπάρχουν τέσσερα διαφορετικά πρότυπα Bayer, τα οποία προσδιορίζονται από το χρώμα των τεσσάρων pixels (2x2 γειτονιά): GBRG, GRBG, BGGR, RGGB.

Green	Blue	Green	Red	Blue	Green	Red	Green
Red	Green	Blue	Green	Green	Red	Green	Blue

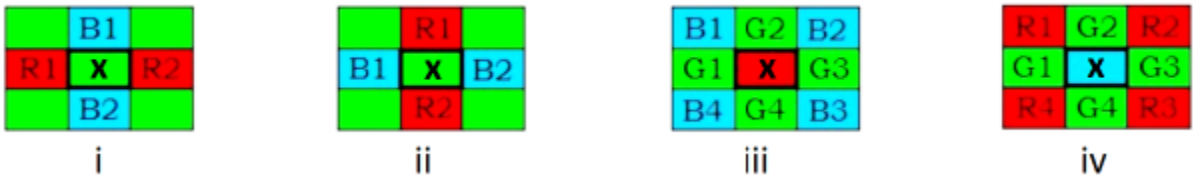
Για να μετατραπεί η εικόνα που προκύπτει από το μωσαϊκό Bayer σε RGB, θα πρέπει να υπολογιστούν οι δύο χρωματικές συνιστώσες που λείπουν για κάθε pixel. Η διαδικασία αυτή ονομάζεται Debayering ή Demosaicing.

Ζητούμενο Εργαστηριακής Άσκησης

Στα πλαίσια της παρούσας Εργαστηριακής Άσκησης, καλούμαστε να υλοποιήσουμε το φίλτρο Debayering, το οποίο θα μετατρέπει την Bayer εικόνα σε RGB υπολογίζοντας τους μέσους όρους των γειτονικών pixels σε 3x3 γειτονιές. Θεωρούμε ότι το μωσαικό Bayer ακολουθεί το πρότυπο GBRG, δηλαδή η Bayer εικόνα ακολουθεί το μοτίβο που παρουσιάζεται στο παρακάτω σχήμα.



Με βάση την κατανομή των χρωματικών συνιστωσών, υπάρχουν συνολικά τέσσερις διαφορετικές περιπτώσεις όπου γνωρίζουμε την μία χρωματική συνιστώσα και θέλουμε να βρούμε τις άλλες δύο. Οι περιπτώσεις αυτές παρουσιάζονται παρακάτω:



Με βάση το παραπάνω σχήμα, πραγματοποιούνται οι ακόλουθοι υπολογισμοί για το κεντρικό pixel X:

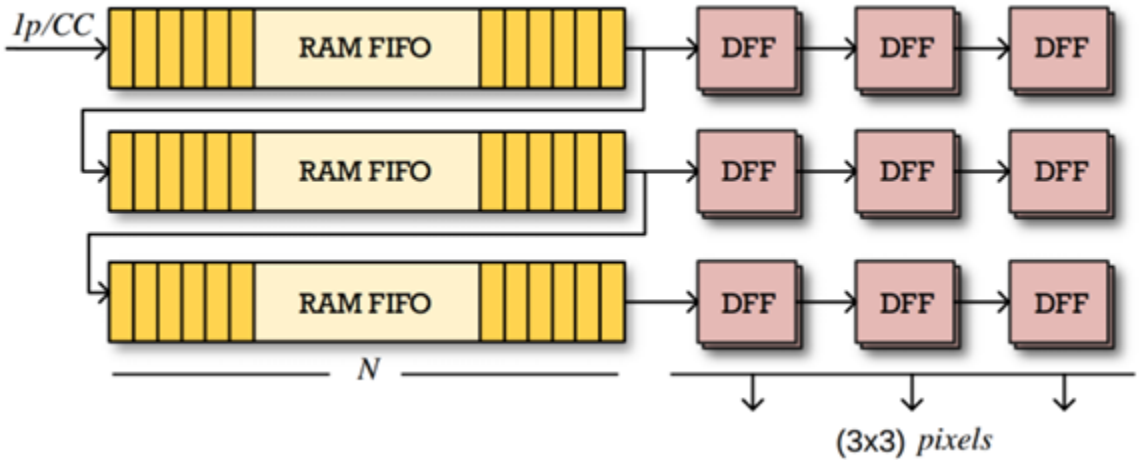
- Αν αυτό είναι πράσινο (περ. i, ii), η κόκκινη (μπλε) χρωματική του συνιστώσα υπολογίζεται ως ο μέσος όρος των δύο κόκκινων (μπλε) γειτονικών pixels.
- Αν αυτό είναι κόκκινο (περ. iii), η πράσινη (μπλε) χρωματική του συνιστώσα υπολογίζεται ως ο μέσος όρος των τεσσάρων πράσινων (μπλε) γειτονικών pixels.
- Αν αυτό είναι μπλε (περ. iv), η πράσινη (κόκκινη) χρωματική του συνιστώσα υπολογίζεται ως ο μέσος όρος των τεσσάρων πράσινων (κόκκινων) γειτονικών pixels.

Λεπτομέρειες Σχεδίασης

Για τη σχεδίαση του Debayering φίλτρου αξιοποιούνται 3 components.

1. Serial to Parallel

Το ζητούμενο φίλτρο λαμβάνει ένα pixel ανά κύκλο ρολογιού με raster scan ordering. Η επεξεργασία της εικόνας γίνεται σε γειτονιές 3x3. Για τον σκοπό αυτό, αξιοποιείται η δομή FIFO, η οποία περιλαμβάνει μνήμη μήκους *N* και 3 DFF's. Για να υλοποιήσουμε μια 3x3 γειτονιά, απαιτούνται 3 τέτοιες δομές, οι οποίες συνδέονται μεταξύ τους με βάση το παρακάτω σχήμα.



Η εικόνα σκανάρεται από τα αριστερά προς τα δεξιά και από πάνω προς τα κάτω. Αυτό σημαίνει πως για το πρώτο κεντρικό pixel, το οποίο είναι το πάνω αριστερά (όπως φαίνεται στις παραπάνω εικόνες), θα χρειαστεί zero padding για τα pixels που "λείπουν" (πάνω-αριστερά, κέντρο-αριστερά, κάτω-αριστερά, πάνω, πάνω-δεξιά). Η παραπάνω διαδικασία βοηθάει τόσο στο να τοποθετούνται τα pixels στην τοπολογία που αναδεικνύεται στην εικόνα (παρόλο που σκανάρονται γραμμικά), όσο και στο να διατηρούνται ατόφια τα pixels που πρέπει να χρησιμοποιηθούν πολλαπλές φορές, για την επεξεργασία όλων των γειτονιών στις οποίες ανήκουν.

Ο κώδικας για το component s2p φαίνεται παρακάτω:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity s2p is
    generic (N: integer := 32);
    Port (
            clock : in std_logic;
            reset : in std_logic;
            pixel : in std_logic_vector(7 downto 0);
            valid_in : in std_logic;
            p00 : out std_logic_vector(7 downto 0);
            p01 : out std_logic_vector(7 downto 0);
            p02 : out std_logic_vector(7 downto 0);
            p10 : out std_logic_vector(7 downto 0);
            p11 : out std_logic_vector(7 downto 0);
            p12 : out std_logic_vector(7 downto 0);
            p20 : out std_logic_vector(7 downto 0);
            p21 : out std_logic_vector(7 downto 0);
            p22 : out std_logic_vector(7 downto 0);
            valid_out : out std_logic
     );
end s2p;

architecture Behavioral of s2p is

    component fifo_generator_0
    Port(
        clk : in std_logic;
        srst : in std_logic;
        din : in std_logic_vector(7 downto 0);
        wr_en : in std_logic;
        rd_en : in std_logic;
        dout : out std_logic_vector(7 downto 0);
        full : out std_logic;
```

```
        empty : out std_logic;
        valid : out std_logic;
        data_count : out std_logic_vector(9 downto 0)
        );
    end component;

    signal fout2, fout1, fout0 : std_logic_vector(7 downto 0) :=
(others => '0') ;
    signal full0, full1, full2, empty0, empty1, empty2, valid0,
valid1, valid2 : std_logic;
    signal datacount0, datacount1, datacount2 : std_logic_vector(9
downto 0);
    signal counter : std_logic_vector(integer(ceil(log2(real(N*N))))
downto 0) :=  (others => '0') ;
    signal srst : std_logic := '0';

    signal  rd_en0: std_logic := '0';
    signal  rd_en1: std_logic := '0';
    signal  rd_en2: std_logic := '0';
    signal  wr_en0: std_logic := '1';
    signal  wr_en1: std_logic := '1';
    signal  wr_en2: std_logic := '1';

    type arr is array (2 downto 0) of std_logic_vector (7 downto 0);
    signal line0 : arr := (others => (others => '0'));
    signal line1 : arr := (others => (others => '0'));
    signal line2 : arr := (others => (others => '0'));

begin

    RAM_FIFO_0 : fifo_generator_0
    PORT MAP (
        clk => clock,
        srst => srst,
        din => pixel,
        wr_en => wr_en0,
        rd_en => rd_en0,
        dout => fout0,
        full => full0,
```

```
    empty => empty0,
    valid => valid0,
    data_count => datacount0
);

RAM_FIFO_1 : fifo_generator_0
PORT MAP (
    clk => clock,
    srst => srst,
    din => fout0,
    wr_en => wr_en1,
    rd_en => rd_en1,
    dout => fout1,
    full => full1,
    empty => empty1,
    valid => valid1,
    data_count => datacount1
);

RAM_FIFO_2 : fifo_generator_0
PORT MAP (
    clk => clock,
    srst => srst,
    din => fout1,
    wr_en => wr_en2,
    rd_en => rd_en2,
    dout => fout2,
    full => full2,
    empty => empty2,
    valid => valid2,
    data_count => datacount2
);

process(clock, reset, datacount0)
begin
    --valid_out <= '0';

    if reset = '0' then
        rd_en0 <= '0';
```

```
        rd_en1 <= '0';
        rd_en2 <= '0';
        valid_out <= '0';
        counter <= (others => '0');
        srst <= '1';
        p00 <= (others => '0');
        p01 <= (others => '0');
        p02 <= (others => '0');
        p10 <= (others => '0');
        p11 <= (others => '0');
        p12 <= (others => '0');
        p20 <= (others => '0');
        p21 <= (others => '0');
        p22 <= (others => '0');

    elsif (clock'event and clock = '1') then
        srst <= '0';
        if valid_in = '1' then
            counter <= counter + 1;
            if to_integer(unsigned(counter)) = 2*N+3 then
                valid_out <= '1';
            end if;
            if to_integer(unsigned(datacount0)) = N-2 then
                rd_en0 <= '1';
                rd_en1 <= '1';
                rd_en2 <= '1';
            end if;
            line0 <= line0(1 downto 0) & fout0;
            line1 <= line1(1 downto 0) & fout1;
            line2 <= line2(1 downto 0) & fout2;
            p00 <= line0(0);
            p01 <= line0(1);
            p02 <= line0(2);
            p10 <= line1(0);
            p11 <= line1(1);
            p12 <= line1(2);
            p20 <= line2(0);
            p21 <= line2(1);
            p22 <= line2(2);
```

```
                if counter > N*N + 2*N + 3 then
                    valid_out <= '0';
                end if;
            end if;
        end if;
    end process;


end Behavioral;
```

Για την υλοποίηση του κώδικα, αξιοποιείται το IP component *FIFO generator*, με τα παρακάτω specifications (τα οποία συνάδουν με τα ζητούμενα της άσκησης):

Block RAM resource(s) (18K BRAMs): 1
Block RAM resource(s) (36K BRAMs): 0

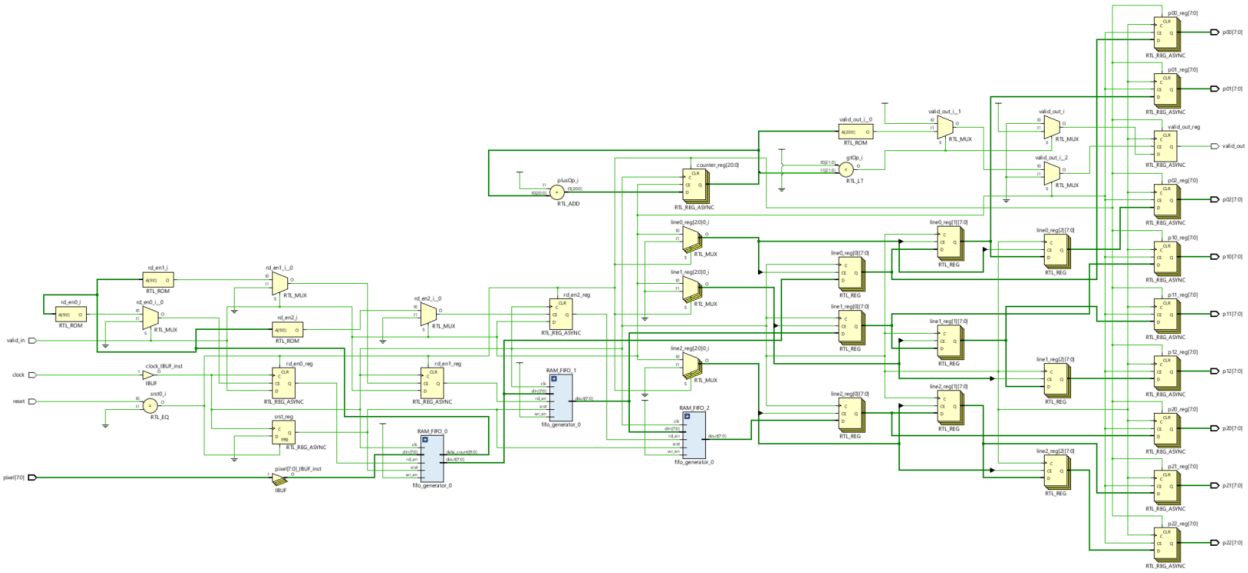
Clocking Scheme	Common Clock
Memory Type	Block RAM
Model Generated	Behavioral Model
Write Width	8
Write Depth	1024
Read Width	8
Read Depth	1024
Almost Full/Empty Flags	Not Selected/Not Selected
Programmable Full/Empty Flags	Not Selected/Not Selected
Data Count Outputs	Selected
Handshaking	Selected
Read Mode / Reset	Standard FIFO / Synchronous
Read Latency (From Rising Edge of Read Clock)	1

Ο κώδικας που παρέχεται για το s2p εκτελεί την εξής λειτουργία:

- Τα δεδομένα εισόδου εισέρχονται στο FIFO με την ενεργοποίηση του σήματος *wr_en0*. Τα δεδομένα εξόδου των FIFO αντιστοιχούν στις μεταβλητές *fout0, fout1, fout2*.
- Κατά την ενημέρωση των εξόδων, τα δεδομένα εξάγονται από τα FIFO και τοποθετούνται αντίστοιχα στους πίνακες *line0, line1, line2*, ανάλογα με το πόσα δεδομένα είναι διαθέσιμα στο FIFO και την κατάσταση των σημάτων ελέγχου. Καθένας από τους παραπάνω πίνακες line*i* περιέχει τα στοιχεία *pi1-2*.
- Ο μετρητής counter αυξάνεται σε κάθε κύκλο. Όταν ο μετρητής φτάσει την τιμή *2*N+1*. Τότε το σήμα εξόδου valid_out γίνεται triggered, υποδεικνύοντας πως η μετατροπή serial to parallel ολοκληρώθηκε. Επίσης, ελέγχεται ο αριθμός των δειγμάτων που έχουν διαβαστεί από το FIFO. Όταν αυτός γίνεται ίσος με *N-2*, ενεργοποιούνται τα σήματα εγγραφής *rd_en0, rd_en1, rd_en2*, για την ανάγνωση περαιτέρω δεδομένων από το FIFO.

- Όταν ο counter υπερβεί την τιμή *N*N + 2*N + 3*, το σήμα valid_out γίνεται ίσο με 0, υποδεικνύοντας το τέλος της διαδικασίας μετατροπής.

Παρακάτω φαίνεται το RTL-design του s2p:



2. Calculation

Θεωρώντας το πρότυπο Bayer GBRG, γνωρίζουμε εκ των προτέρων ότι το 1ο έγκυρο pixel που θα λάβουμε ως είσοδο είναι G, το 2ο είναι B, το 3ο είναι G, κ.ο.κ. Αντίστοιχα, για την επόμενη γραμμή της εικόνας, το 1ο είναι R, το 2ο είναι G, το 3ο είναι R, κ.ο.κ. Το κύκλωμα χρειάζεται να γνωρίζει τι χρώμα είναι το κάθε pixel, ώστε να πραγματοποιήσει τους αντίστοιχους υπολογισμούς για τις χρωματικές συνιστώσες που λείπουν. Αυτό επιτυγχάνεται με την χρήση μετρητών (counters), τόσο για τις γραμμές όσο και για τα pixels της κάθε γραμμής. Ο μετρητής γραμμών υποδεικνύει το μοτίβο (GBG ή RGR) που θα έχουν τα επόμενα N pixels, ενώ ο μετρητής pixels υποδεικνύει το χρώμα του pixel.

Για την εκτέλεση της παραπάνω λειτουργίας, υλοποιούμε ένα component ονομαζόμενο *Calculation*.

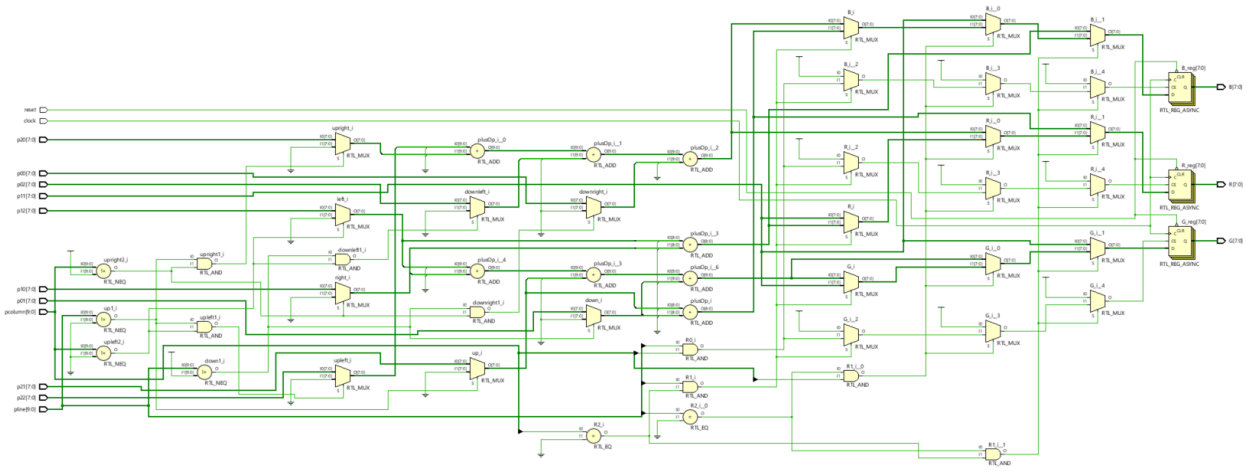
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use ieee.math_real.all;


entity calculation is    generic (N: integer := 1024);
    Port (
```

```
          clock : in std_logic;
          reset : in std_logic;
          p00 : in std_logic_vector(7 downto 0);
          p01 : in std_logic_vector(7 downto 0);
          p02 : in std_logic_vector(7 downto 0);
          p10 : in std_logic_vector(7 downto 0);
          p11 : in std_logic_vector(7 downto 0);
          p12 : in std_logic_vector(7 downto 0);
          p20 : in std_logic_vector(7 downto 0);
          p21 : in std_logic_vector(7 downto 0);
          p22 : in std_logic_vector(7 downto 0);
          pline : in std_logic_vector(integer(ceil(log2(real(N))))-1
downto 0);
          pcolumn : in
std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0);
          R : out std_logic_vector(7 downto 0);
          G : out std_logic_vector(7 downto 0);
          B : out std_logic_vector(7 downto 0)
          );
end calculation;

architecture Behavioral of calculation is

    signal x, up, down, left, right, upleft, upright, downleft,
downright : std_logic_vector(7 downto 0);
    signal sum_row, sum_column, sum_cross, sum_outer :
std_logic_vector(7 downto 0);
    signal temp0, temp1 : std_logic_vector(9 downto 0);
    signal temp2, temp3 : std_logic_vector(8 downto 0);

    begin

    upleft <= p22 when to_integer(unsigned(pline)) /= 0 and
to_integer(unsigned(pcolumn)) /= 0 else (others => '0');
    up <= p21 when to_integer(unsigned(pline)) /= 0 else (others =>
'0');
    upright <= p20 when to_integer(unsigned(pline)) /= 0 and
to_integer(unsigned(pcolumn)) /= N-1 else (others => '0');
    left <= p12 when to_integer(unsigned(pcolumn)) /= 0 else (others
```

```
=> '0');
    x <= p11;
    right <= p10 when to_integer(unsigned(pcolumn)) /= N-1 else
(others => '0');
    downleft <= p02 when to_integer(unsigned(pline)) /= N-1 and
to_integer(unsigned(pcolumn)) /= 0 else (others => '0');
    down <= p01 when to_integer(unsigned(pline)) /= N-1 else (others
=> '0');
    downright <= p00 when to_integer(unsigned(pline)) /= N-1 and
to_integer(unsigned(pcolumn)) /= N-1 else (others => '0');

    temp0 <= ("00" & left) + ("00" & right) + ("00" & up) + ("00" &
down);
    sum_cross <= temp0(9 downto 2);

    temp1 <= ("00" & upleft) + ("00" & upright) + ("00" & downleft) +
("00" & downright);
    sum_outer <= temp1(9 downto 2);

    temp2 <= ('0' & up) + ('0' & down);
    sum_column <= temp2(8 downto 1);

    temp3 <= ('0' & left) + ('0' & right);
    sum_row <= temp3(8 downto 1);

    process(clock, reset)
    begin
        if reset = '0' then
            R <= (others => '0');
            G <= (others => '0');
            B <= (others => '0');
        elsif (clock'event and clock = '1') then
            if pline(0) = '0' and pcolumn(0) ='0' then      --
periptwsh (ii)
                R <= sum_column;
                G <= x;
                B <= sum_row;
            elsif pline(0) = '0' and pcolumn(0) = '1' then
--periptwsh (iv)
```

```
                R <= sum_outer;
                G <= sum_cross;
                B <= x;
            elsif pline(0) = '1' and pcolumn(0) = '0' then
--periptwsh (iii)
                R <= x;
                G <= sum_cross;
                B <= sum_outer;
            elsif pline(0) = '1' and pcolumn(0) = '1' then
--periptwsh (i)
                R <= sum_row;
                G <= x;
                B <= sum_column;
            end if;
        end if;
    end process;

end Behavioral;
```

Το RTL design που προκύπτει για τον παραπάνω κώδικα είναι το εξής:



Ο κώδικας του calculation εκτελεί τις παρακάτω λειτουργίες:

- Υπολογίζει τα γειτονικά pixel του κεντρικού pixel και τα αποθηκεύει στα κατάλληλα σήματα (*up, down, left, right, uploeft, upright, downleft, downright*).
- Υπολογίζει τις συνολικές τιμές για τα γειτονικά pixel. Ανάλογα με την κατηγορία στην οποία ανήκει το κεντρικό pixel, προκύπτει η ανάγκη υπολογισμού σημάτων όπως *sum_row, sum_column, sum_cross, sum_outer*.

- Εξετάζει σε ποια από τις 4 κατηγορίες ανήκει το κεντρικό pixel x. Ανάλογα με το αποτέλεσμα, προκύπτει ένα μοτίβο GBG ή RGR, σύμφωνα με το οποίο ανανεώνονται οι τιμές των *R, G, B*.

3. Controller

Το component αυτό ελέγχει την ορθή λειτουργία του κυκλώματος και μεταβιβάζει τα κατάλληλα σήματα στο *calculation* component. Αφού λάβει για πρώτη φορά το σήμα valid_out = '1' από το component *s2p*, επιβεβαιώνεται το πρώτο σωστό αποτέλεσμα και ξεκινά η ανανέωση των στηλών και των γραμμών, ανάλογα με τη θέση του pixel στην οποία βρισκόμαστε. Τέλος, επιτρέπει την κατάλληλη ενεργοποίηση του σήματος image_finished όταν ολοκληρωθεί η επεξεργασία της εικόνας.

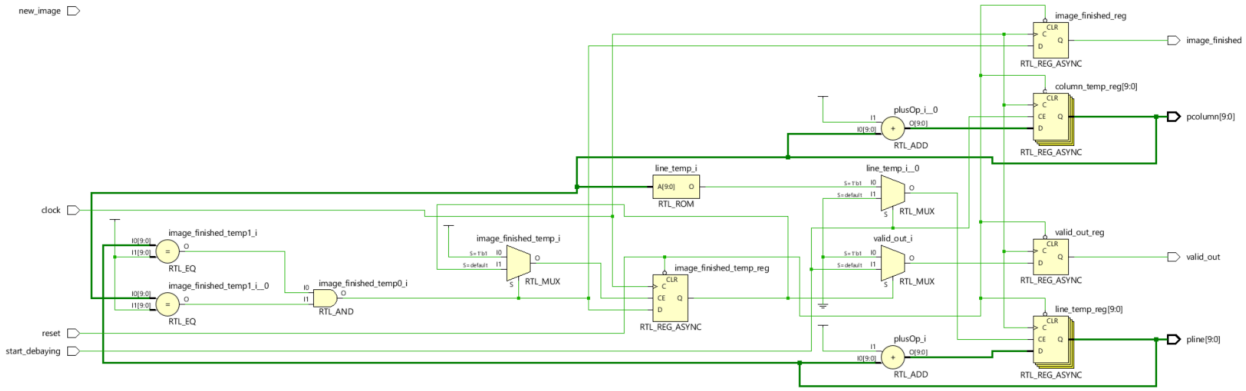
Παρατίθεται ο κώδικας για τον *Controller :*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity controller is
    generic (N: integer := 1024);
    Port (
          clock : in std_logic;
          reset : in std_logic;
          new_image : in std_logic;
          start_debaying : in std_logic;
          pline : out std_logic_vector(integer(ceil(log2(real(N))))-1
downto 0);
          pcolumn : out
std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0);
          valid_out : out std_logic;
          image_finished : out std_logic
          );
end controller;

architecture Behavioral of controller is

    signal column_temp, line_temp :
```

```
std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0) := (others
=> '0');
    signal image_finished_temp : std_logic := '0';

begin

    pcolumn <= column_temp;
    pline <= line_temp;

    process(clock, reset)
    begin
        if reset = '0' then
            line_temp <= (others => '0');
            column_temp <= (others => '0');
            valid_out <= '0';
            image_finished_temp <= '0';
            image_finished <= '0';

        elsif (clock'event and clock = '1') then
            valid_out <= '0';
            if start_debaying = '1' then
                valid_out <= '1';
                column_temp <= column_temp + 1;
                if to_integer(unsigned(column_temp)) = N-1 then
                    line_temp <= line_temp + 1;
                end if;
            end if;
            if (image_finished_temp = '1') then
                valid_out <= '0';
                image_finished_temp <= '0';

            end if;
            --if (to_integer(unsigned(line_temp)) = 4 and
to_integer(unsigned(column_temp)) = 28) then
            if (to_integer(unsigned(line_temp)) = N-1 and
to_integer(unsigned(column_temp)) = N-1) then
                image_finished <= '1';
                image_finished_temp <= '1';
            else
```

```
                image_finished <= '0';
            end if;
        end if;
    end process;


end Behavioral;
```

To RTL-Schema:



4. <u>Debayering filter:</u>

Ο κώδικας του τελικού *Debayer* component συνδυάζει τις λειτουργίες όλων των παραπάνω components.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use ieee.math_real.all;


entity debayering is
    generic (N: integer := 1024);
    Port (
          clock : in std_logic;
          reset : in std_logic;
          valid_in : in std_logic;
```

```
          pixel : in std_logic_vector(7 downto 0);
          new_image : in std_logic;
          counter : out std_logic_vector(20 downto 0) := (others =>
'0');
          R : out std_logic_vector(7 downto 0);
          G : out std_logic_vector(7 downto 0);
          B : out std_logic_vector(7 downto 0);
          valid_out : out std_logic;
          image_finished : out std_logic;
          n_line, n_column: out
std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0)
     );
end debayering;

architecture Behavioral of debayering is

component s2p is
    generic (N: integer);
    Port (
            clock : in std_logic;
            reset : in std_logic;
            pixel : in std_logic_vector(7 downto 0);
            valid_in : in std_logic;
            p00 : out std_logic_vector(7 downto 0);
            p01 : out std_logic_vector(7 downto 0);
            p02 : out std_logic_vector(7 downto 0);
            p10 : out std_logic_vector(7 downto 0);
            p11 : out std_logic_vector(7 downto 0);
            p12 : out std_logic_vector(7 downto 0);
            p20 : out std_logic_vector(7 downto 0);
            p21 : out std_logic_vector(7 downto 0);
            p22 : out std_logic_vector(7 downto 0);
            valid_out : out std_logic
     );
end component;

component calculation is
    generic (N: integer);
    Port (
```

```
            clock : in std_logic;
            reset : in std_logic;
            p00 : in std_logic_vector(7 downto 0);
            p01 : in std_logic_vector(7 downto 0);
            p02 : in std_logic_vector(7 downto 0);
            p10 : in std_logic_vector(7 downto 0);
            p11 : in std_logic_vector(7 downto 0);
            p12 : in std_logic_vector(7 downto 0);
            p20 : in std_logic_vector(7 downto 0);
            p21 : in std_logic_vector(7 downto 0);
            p22 : in std_logic_vector(7 downto 0);
            pline : in std_logic_vector(integer(ceil(log2(real(N))))-1
downto 0);
            pcolumn : in
std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0);
            R : out std_logic_vector(7 downto 0);
            G : out std_logic_vector(7 downto 0);
            B : out std_logic_vector(7 downto 0)
            );
end component;

component controller is
    generic (N: integer);
    Port (
            clock : in std_logic;
            reset : in std_logic;
            new_image : in std_logic;
            start_debaying : in std_logic;
            pline : out std_logic_vector(integer(ceil(log2(real(N))))-1
downto 0);
            pcolumn : out
std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0);
            valid_out : out std_logic;
            image_finished : out std_logic
            );
end component;

signal clock_gate : std_logic;
signal valid : std_logic;
```

```
signal counter_temp : std_logic_vector(20 downto 0) := (others =>
'0');
signal pix00, pix01, pix02, pix10, pix11, pix12, pix20, pix21, pix22
: std_logic_vector(7 downto 0) := (others => '0');
signal line_temp, column_temp :
std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0)  :=
(others => '0');
signal start : std_logic;
signal valid_out_temp : std_logic;
signal stop : std_logic := '0';


begin

ctrl : controller
    generic map (N => N)
    port map (
        clock => clock_gate,
        reset => reset,
        new_image => new_image,
        start_debaying => start,
        pline => line_temp,
        pcolumn => column_temp,
        valid_out => valid_out_temp,
        image_finished => image_finished
    );

calc : calculation
    generic map (N => N)
    port map (
        clock => clock_gate,
        reset => reset,
        p00 => pix00,
        p01 => pix01,
        p02 => pix02,
        p10 => pix10,
        p11 => pix11,
        p12 => pix12,
        p20 => pix20,
```

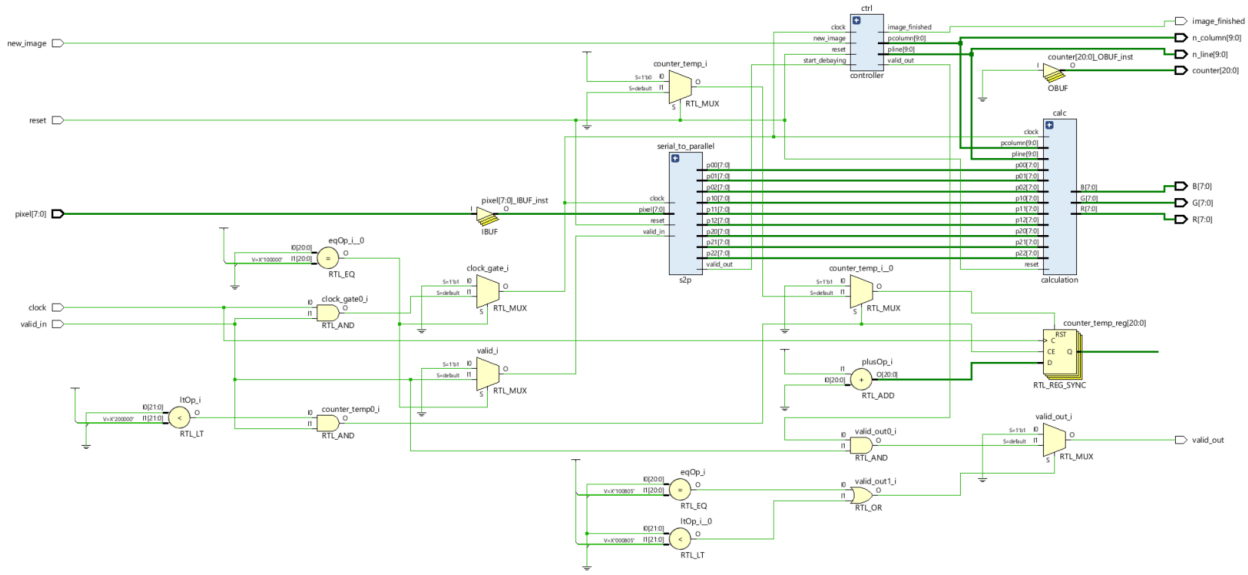
```
        p21 => pix21,
        p22 => pix22,
        pline => line_temp,
        pcolumn => column_temp,
        R => R,
        G => G,
        B => B
    );

serial_to_parallel : s2p
    generic map (N => N)
    port map (
        clock => clock_gate,
        reset => reset,
        pixel => pixel,
        valid_in => valid,
        p00 => pix00,
        p01 => pix01,
        p02 => pix02,
        p10 => pix10,
        p11 => pix11,
        p12 => pix12,
        p20 => pix20,
        p21 => pix21,
        p22 => pix22,
        valid_out => start
     );

process(clock)
begin
    if reset = '0' then
        counter_temp <= (others => '0');
        stop <= '0';
    end if;

    if (clock'event and clock = '1') then
        if counter_temp < 2*N*N and valid_in = '1' then
            counter_temp <= counter_temp + 1;
        end if;
```

```
    end if;
end process;

valid <= stop when counter_temp = N*N else valid_in;
valid_out <= (valid_out_temp and stop) when (counter_temp = (N*N +
2*N + 5) or counter_temp < (2*N+5)) else (valid_out_temp and
valid_in);
clock_gate <= (clock and stop) when counter_temp = N*N else (clock
and valid_in);
n_line <= line_temp;
n_column <= column_temp;
counter <= counter_temp;
--valid_out <= valid_out_temp and valid_in;
--clock_gate <= clock and valid_in;

end Behavioral;
```

Η ζητούμενη διαδικασία του φίλτρου εκτελείται μόνο σε rising clock edge και μόνο όταν ισχύει `valid_in = '1'`. Αυτό διασφαλίζει ότι η διαδικασία θα εκτελείται μόνο όταν υπάρχουν νέα image input data προς επεξεργασία. Ο μετρητής *counter_temp* αυξάνεται κάθε φορά που εκτελείται η διαδικασία και υπάρχουν έγκυρα δεδομένα εισόδου. Η μεταβλητή αυτή χρησιμεύει για τον έλεγχο της προόδου της επεξεργασίας της εικόνας. Επιπλέον, όταν ο μετρητής φτάσει τον αριθμό των pixel της εικόνας (`counter_temp = N*N`). τότε η μεταβλητή *stop* γίνεται triggered. Διακόπτοντας την εκτέλεση της διαδικασίας.

Το σήμα εξόδου ορίζεται στο επίπεδο του *valid_out_temp* όταν η μεταβλητή *stop* είναι ενεργοποιημένη και ο μετρητής *counter_temp* βρίσκεται είτε πριν το τέλος της επεξεργασίας (`counter_temp = N*N + 2*N + 5`), είτε στους πρώτους *2*N+5* κύκλους της επεξεργασίας (`counter_temp < 2*N + 5`). Αυτός ο έλεγχος εξασφαλίζει ότι το σήμα εξόδου είναι ενεργό μόνο στα στάδια της επεξεργασίας που απαιτείται.

Παρακάτω φαίνεται το RTL design του debayering:



Με τον παρακάτω κώδικα testbench, τρέχουμε το simulation για N = {32, 64, 128}:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.TEXTIO.all;

entity debayering_tb is
end debayering_tb;

architecture Behavioral of debayering_tb is

    constant N : integer := 32;

    component debayering is
    generic (N: integer );
    Port (
          clock : in std_logic;
          reset : in std_logic;
          valid_in : in std_logic;
          pixel : in std_logic_vector(7 downto 0);
          new_image : in std_logic;
          counter : out std_logic_vector(20 downto 0) := (others =>
'0');
          R : out std_logic_vector(7 downto 0);
```

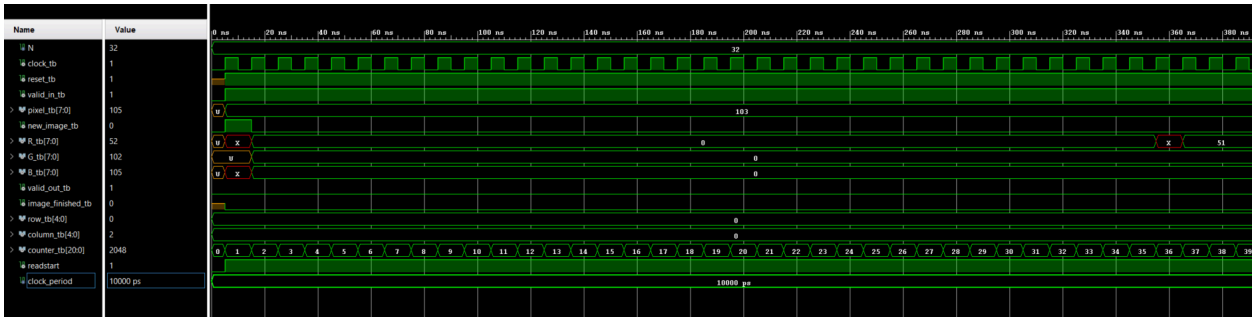
```
            G : out std_logic_vector(7 downto 0);
            B : out std_logic_vector(7 downto 0);
            valid_out : out std_logic;
            image_finished : out std_logic;
            n_line, n_column: out std_logic_vector(9 downto 0)
     );
    end component;

    signal clock_tb : std_logic := '0';
    signal reset_tb : std_logic;
    signal valid_in_tb : std_logic := '0';
    signal pixel_tb : std_logic_vector(7 downto 0);
    signal new_image_tb : std_logic := '0';

    signal R_tb : std_logic_vector(7 downto 0);
    signal G_tb : std_logic_vector(7 downto 0);
    signal B_tb : std_logic_vector(7 downto 0);
    signal valid_out_tb : std_logic;
    signal image_finished_tb : std_logic;
    signal row_tb, column_tb : std_logic_vector(9 downto 0) ;
    signal counter_tb : std_logic_vector(20 downto 0) := (others =>
'0');

    signal readstart : std_logic := '0';

    constant clock_period : time := 10ns;

begin
    dut: debayering
    generic map(N => N)
    port map(
        clock => clock_tb,
        reset => reset_tb,
        valid_in => valid_in_tb,
        pixel => pixel_tb,
        new_image => new_image_tb,
        counter => counter_tb,
        R => R_tb,
        G => G_tb,
```

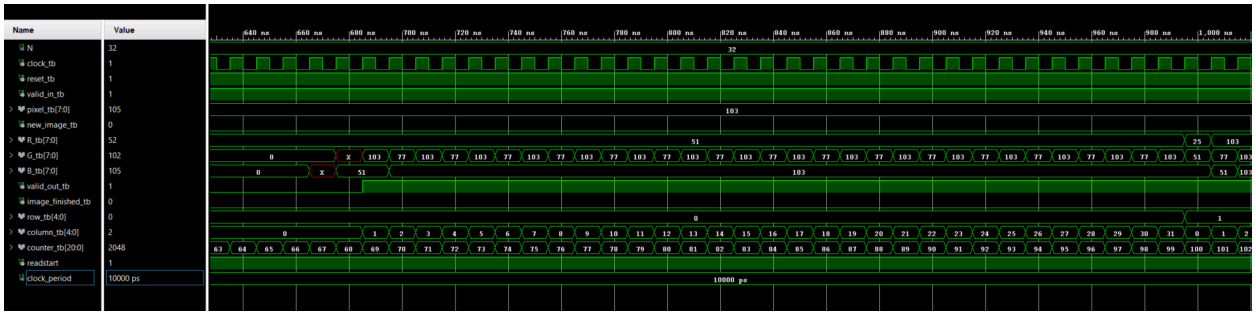
```
        B => B_tb,
        valid_out => valid_out_tb,
        image_finished => image_finished_tb,
        n_line => row_tb,
        n_column => column_tb
        );

    clock_process : process
    begin
        clock_tb <= '0';
        wait for clock_period/2;
        clock_tb <= '1';
        wait for clock_period/2;
    end process;

    init : process
    begin
        wait for clock_period/2;
        reset_tb <= '1';
        readstart <='1';
--        valid_in_tb <= '1';
--        wait for 50*clock_period;
--        reset_tb <= '0';
--        readstart <='0';
--        valid_in_tb <= '1';
--        wait for 10*clock_period;
--        reset_tb <= '1';
--        readstart <='1';
        valid_in_tb <= '1';
       wait for 1048577*clock_period;
--        reset_tb <= '0';
--        readstart <='0';
--        wait for 10*clock_period;
        wait;
    end process ; -- init

    readFile : process(clock_tb, reset_tb)

    file inputFile : text open read_mode is
```

```
"C:\Users\Gina\OneDrive\Desktop\DVLSI\Lab\bayer1024x1024.txt";
    variable rowRead : line;
    variable pixelRead : integer;
    variable pixelRowCounter : integer := 0;

    file outputFile : text open write_mode is
"C:\Users\Gina\OneDrive\Desktop\DVLSI\Lab\outputDebayer1024x1024.txt"
;
    variable rowWrite : line;

    begin
        if (reset_tb = '0') then
            pixelRowCounter := 0;
            pixelRead := -1;
            pixel_tb <= (others => '0');
            write(rowWrite, string'("RESET!"));
            writeline(outputFile, rowWrite);
            file_close(inputFile);
            file_open(inputFile,
"C:\Users\Gina\OneDrive\Desktop\DVLSI\Lab\bayer1024x1024.txt",
read_mode);
        elsif (clock_tb'event and clock_tb = '1') then
            if (readstart = '1') then
                if (not endfile(inputFile)) then
                    if (pixelRowCounter = 0) then
                        new_image_tb <= '1';
                    else
                        if (pixelRowCounter = N*N - 1) then
                            pixelRowCounter := 0;
                        end if;
                        new_image_tb <= '0';
                    end if;
                    pixelRowCounter := pixelRowCounter +1;
                    readline(inputFile, rowRead);
                end if;
                read(rowRead, pixelRead);
--                valid_in_tb <= '1';
                pixel_tb <=
std_logic_vector(to_unsigned(pixelRead,8));
```

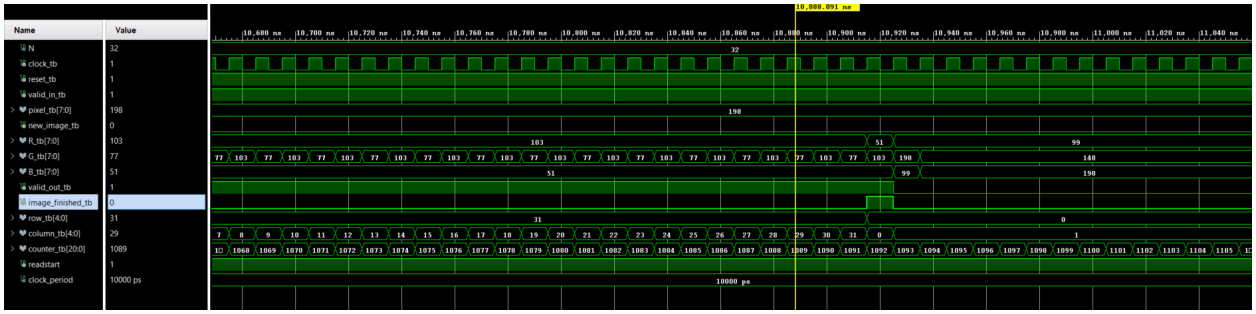
```
            end if;
            if(valid_out_tb = '1') then
                write(rowWrite, to_integer(unsigned(R_tb)), left, 4);
                write(rowWrite, to_integer(unsigned(G_tb)), left, 4);
                write(rowWrite, to_integer(unsigned(B_tb)), left, 4);
                writeline(outputFile, rowWrite);
            end if;
            if (image_finished_tb = '1') then
                write(rowWrite, string'("IMAGE DONE!"));
                writeline(outputFile, rowWrite);
                file_close(outputFile);
            end if;
        end if;

    end process;

end Behavioral;
```

- Για N = 32 :





Παρατηρούμε πως στον 69ο παλμό ενεργοποιείται το σήμα *valid_out_tb*, ενώ είναι η πρώτη φορά στην οποία και τα τρία σήματα *R, G, B* λαμβάνουν μη μηδενική τιμή (σύμφωνα με τους υπολογισμούς του calculation).



Παρατηρούμε πως στο 1093ο παλμό, το σήμα *image_finished_tb* ενεργοποιείται για έναν παλμό. Κάτι τέτοιο επιβεβαιώνει τον κώδικά μας, καθώς η επεξεργασία της εικόνας ολοκληρώνεται 32*32 = 1024 κύκλους μετά την έναρξη της επεξεργασίας (1024 + 69 = 1093). Στον επόμενο παλμό, τα σήματα *valid_out_tb, image_finished_tb, row_tb, column_tb* μηδενίζονται, καθώς η επεξεργασία της εικόνας έχει έρθει σε πέρας. Ελέγχουμε το output file και επιβεβαιώνουμε πως οι τιμές των δεδομένων έχουν πράγματι εγγραφεί:

outputDebayer1024x1024

File Edit View

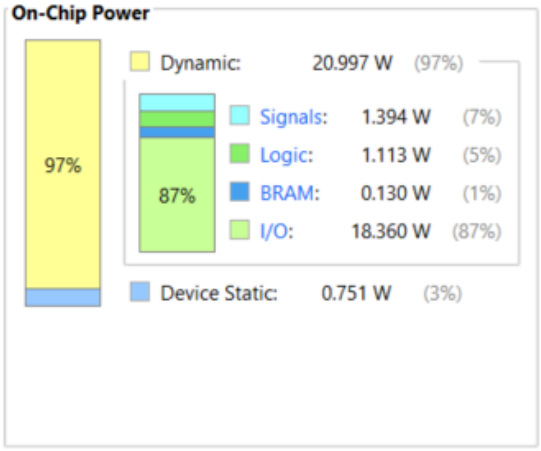
```
51  103 51
51  77  103
51  103 103
51  77  103
51  103 103
51  77  103
51  103 103
51  77  103
51  103 103
51  77  103
51  103 103
51  77  103
51  103 103
51  77  103
51  103 103
51  77  103
51  103 103
51  77  103
51  103 103
51  77  103
51  103 103
51  77  103
51  103 103
51  77  103
51  103 103
51  77  103
51  103 103
51  77  103
51  103 103
```

Η ανάλυση πόρων:

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

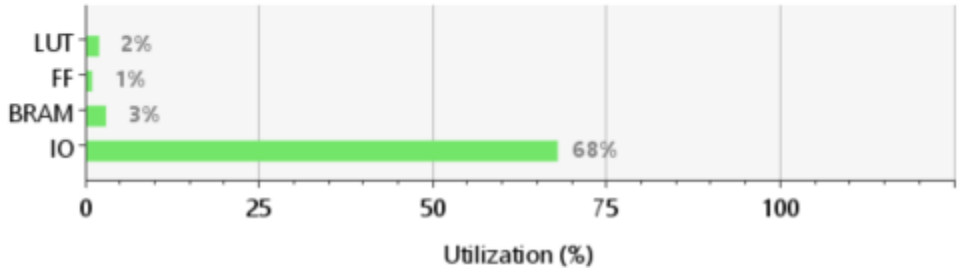
Total On-Chip Power:	**21.748 W (Junction temp exceeded!)**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**125,0°C**
Thermal Margin:	-190,8°C (-16,0 W)
Effective θJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



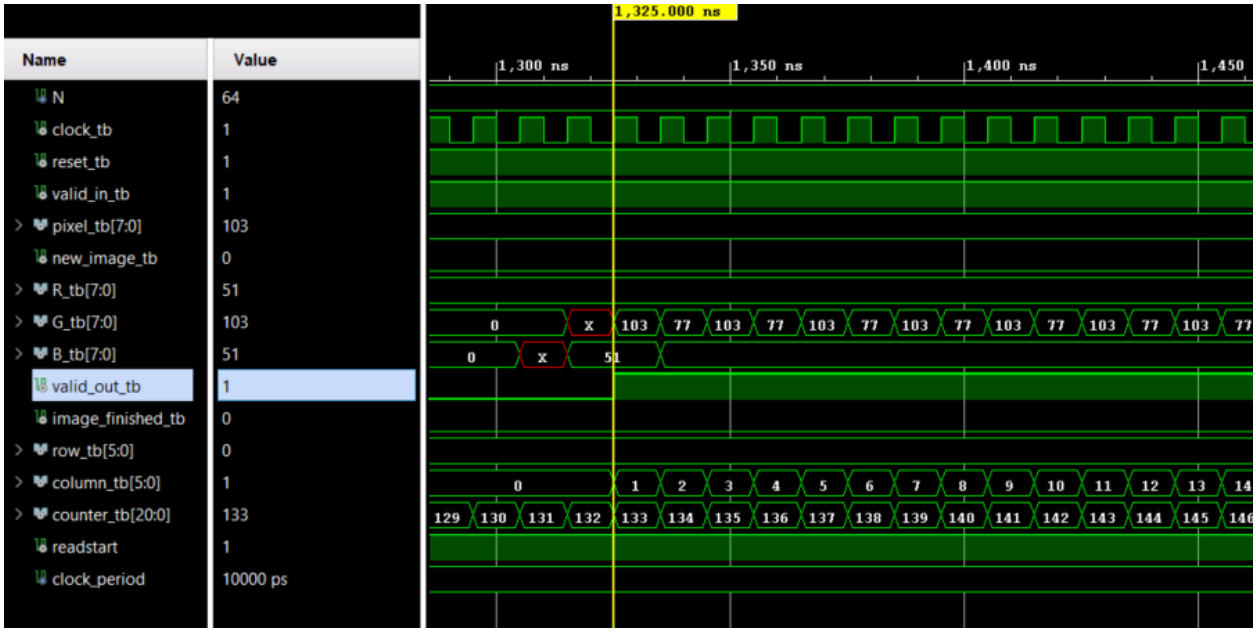
Utilization	Name	Signals (W)	Data (W)	Clock Enable (W)	Logic (W)	BRAM (W)	I/O (W)
20.997 W (97% of total)	debayering						
18.459 W (85% of total)	Leaf Cells (132)						
1.328 W (6% of total)	serial_to_parallel (s2p)	0.643	0.595	0.048	0.556	0.13	<0.001
0.81 W (4% of total)	ctrl (controller)	0.347	0.347	<0.001	0.462	<0.001	<0.001
0.401 W (2% of total)	calc (calculation)	0.331	0.331	<0.001	0.069	<0.001	<0.001

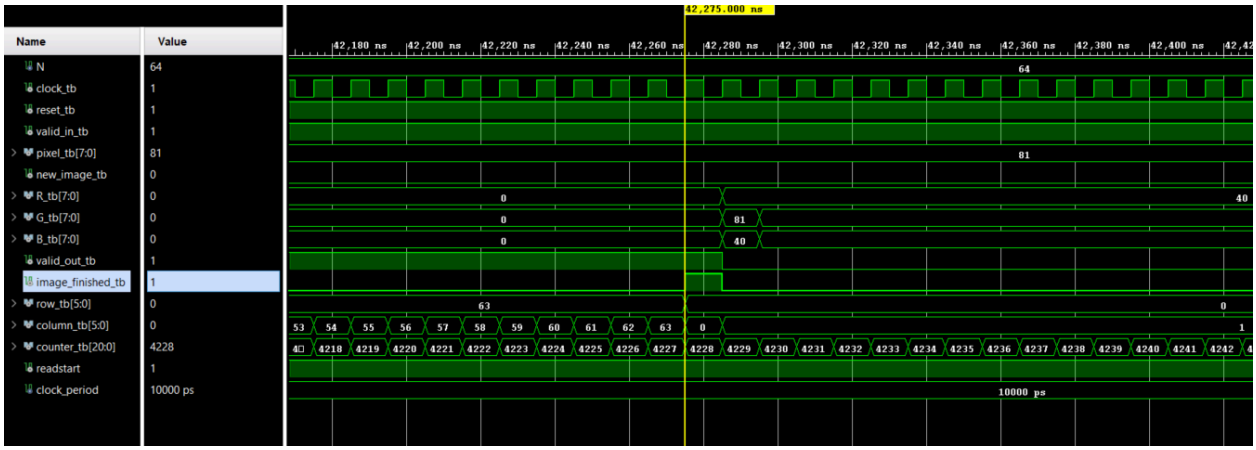
Resource	Utilization	Available	Utilization %
LUT	341	17600	1.94
FF	381	35200	1.08
BRAM	1.50	60	2.50
IO	68	100	68.00



- <u>Για N = 64 ;</u>

Οι εικόνες που προκύπτουν για την προσομοίωση επιβεβαιώνουν και πάλι πως η επεξεργασία της εικόνας εκκινεί και τερματίζει στον σωστό αριθμό παλμών.



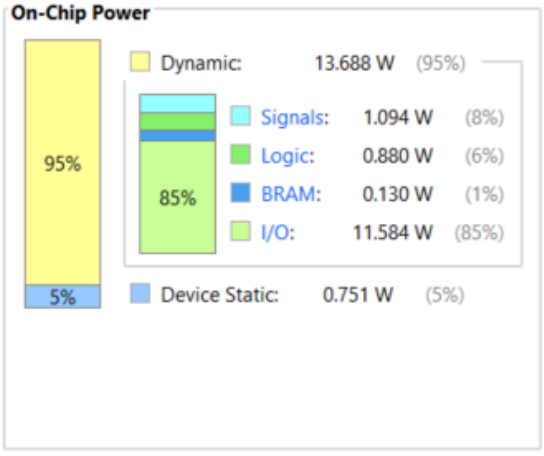


Utilization Report:

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

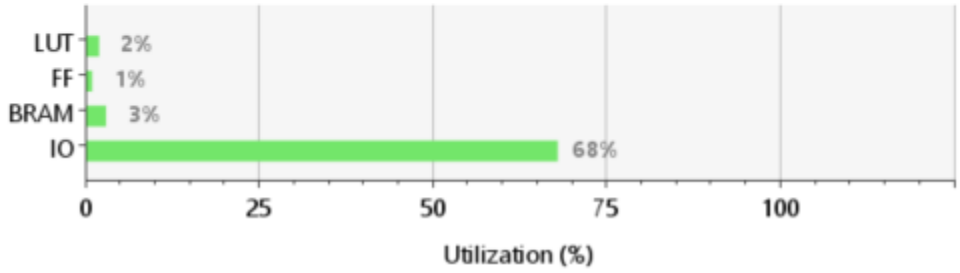
Total On-Chip Power:	**14.438 W (Junction temp exceeded!)**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**125,0°C**
Thermal Margin:	-106,5°C (-8,7 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

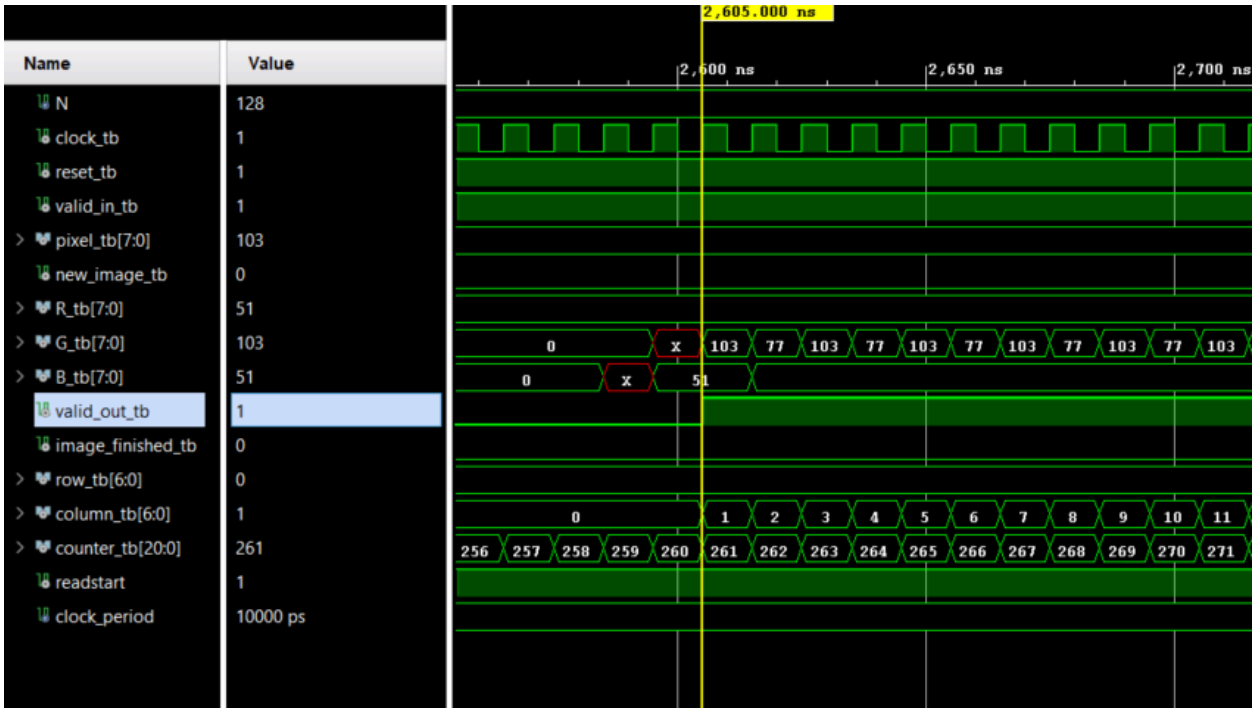


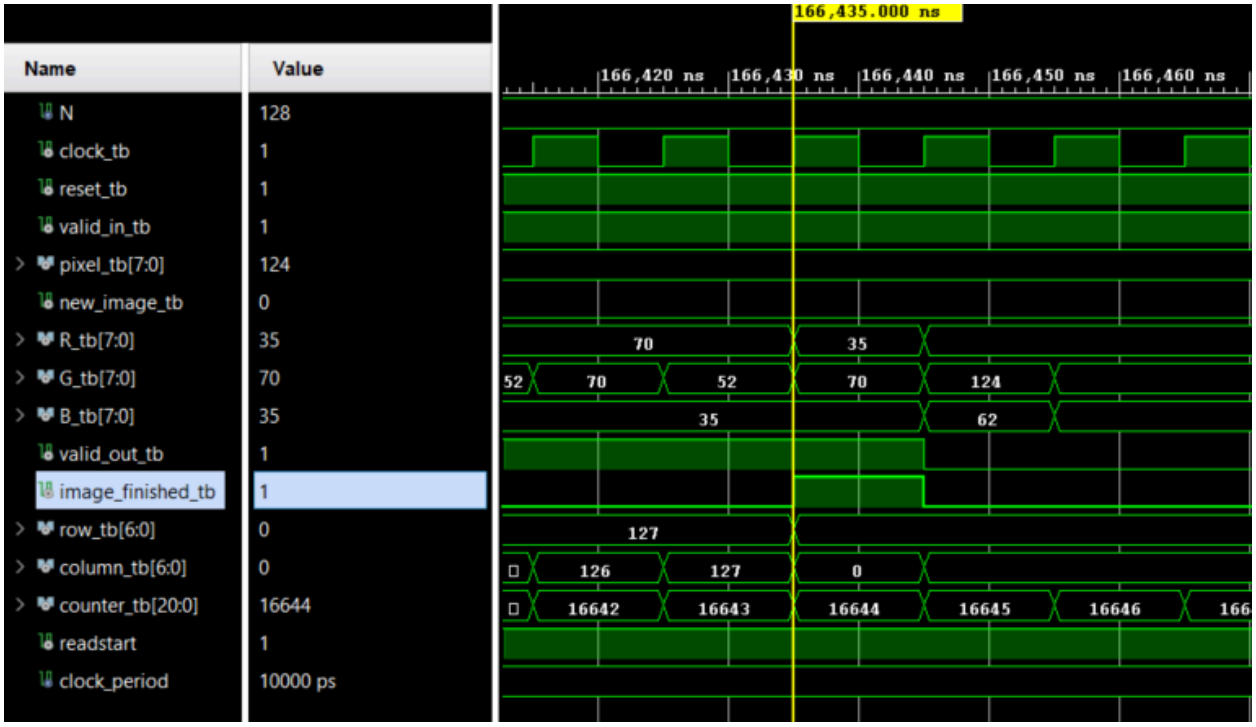
Utilization	Name	Signals (W)	Data (W)	Clock Enable (W)	Logic (W)	BRAM (W)	I/O (W)
13.688 W (95% of total)	debayering						
11.681 W (81% of total)	Leaf Cells (132)						
1.382 W (10% of total)	serial_to_parallel (s2p)	0.699	0.652	0.047	0.554	0.13	<0.001
0.317 W (2% of total)	ctrl (controller)	0.076	0.076	<0.001	0.241	<0.001	<0.001
0.307 W (2% of total)	calc (calculation)	0.25	0.25	<0.001	0.058	<0.001	<0.001

Resource	Utilization	Available	Utilization %
LUT	341	17600	1.94
FF	381	35200	1.08
BRAM	1.50	60	2.50
IO	68	100	68.00



- <u>Για N = 128;</u>



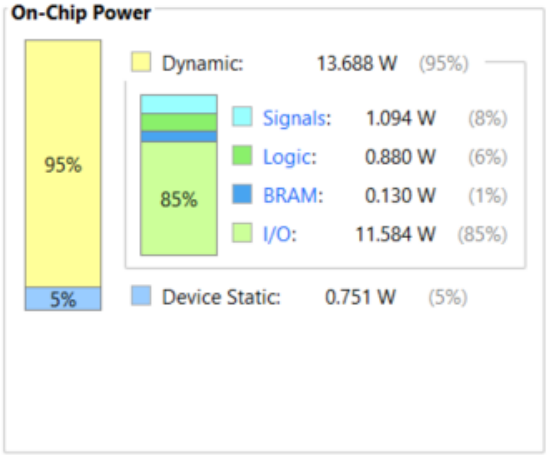


Utilization report:

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

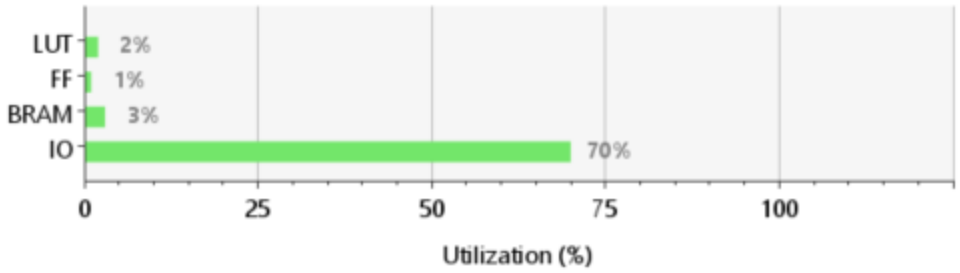
Total On-Chip Power:	**14.438 W (Junction temp exceeded!)**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**125,0°C**
Thermal Margin:	-106,5°C (-8,7 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



Utilization	Name	Signals (W)	Data (W)	Clock Enable (W)	Logic (W)	BRAM (W)	I/O (W)
13.688 W (95% of total)	N debayering						
11.681 W (81% of total)	Leaf Cells (132)						
1.382 W (10% of total)	serial_to_parallel (s2p)	0.699	0.652	0.047	0.554	0.13	<0.001
0.317 W (2% of total)	ctrl (controller)	0.076	0.076	<0.001	0.241	<0.001	<0.001
0.307 W (2% of total)	calc (calculation)	0.25	0.25	<0.001	0.058	<0.001	<0.001

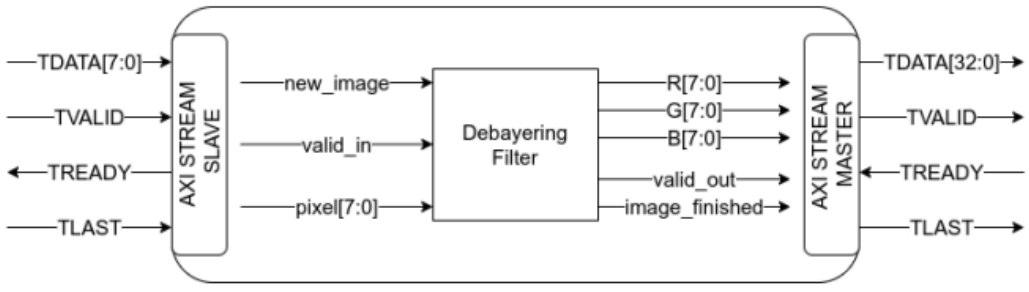
Resource	Utilization	Available	Utilization %
LUT	302	17600	1.72
FF	362	35200	1.03
BRAM	1.50	60	2.50
IO	70	100	70.00



Προγραμματισμός SoC FPGA

Στο Β Μέρος της εργαστηριακής άσκησης κληθήκαμε να προγραμματίσουμε την πλακέτα ZYBO, ώστε να υλοποιεί ένα Debayering φίλτρο, στο οποίο τα δεδομένα εισόδου θα αποστέλλονται από τον ενσωματωμένο επεξεργαστή (ARM) προς το FPGA για επεξεργασία, και αντίστροφα για τα αντίστοιχα αποτελέσματα. Καλούμαστε να συνδέσουμε το Debayering φίλτρο που έχουμε ήδη υλοποιήσει στο Μέρος Α της άσκησης με AXI4-Stream διεπαφές (AXI-Stream Slave/Master Interface). Η διασύνδεση των σημάτων εισόδου και εξόδου του φίλτρου με την AXI διεπαφή να υλοποιηθεί όπως φαίνεται παρακάτω:

- Slave_TDATA[7:0] = pixel[7:0]
- Slave_TVALID = valid_in
- Master_TDATA[32:0] = “00000000” & R[7:0] & G[7:0] & B[7:0]
- Master_TVALID = valid_out
- Master_TLAST = image_finished



Επιπλέον αναπτύσσουμε την ανάλογη εφαρμογή λογισμικού για την αποστολή των σημάτων εισόδου και την λήψη των σημάτων εξόδου του φίλτρου από τον ενσωματωμένο επεξεργαστή. Η επικοινωνία μεταξύ PS-PL θα υλοποιείται μέσω Direct Memory Access (DMA) λογικής, δηλαδή τα hardware components μπορούν να μεταφέρουν δεδομένα απευθείας από και προς τη μνήμη χωρίς να απαιτείται συνεχής παρέμβαση από την CPU. Η εφαρμογή λογισμικού θα είναι υπεύθυνη για την προετοιμασία του DMA ώστε να στέλνει τα pixel προς το φίλτρο και να δέχεται τα αποτελέσματα. Επίσης, η εφαρμογή λογισμικού υλοποιεί και reference software το οποίο θα υπολογίζει και αυτό τα αποτελέσματα του Debayering φίλτρου και θα τα συγκρίνει με τα αποτελέσματα που θα λαμβάνονται από το FPGA.

Για την διευκόλυνση μας θα βασιστούμε στο project dvlsi2021_lab5 που παρέχεται από το εργαστήριο. Αφού κατεβάσουμε το αρχείο και κάνουμε export τα αρχεία του φακέλου, ανοίγουμε το Vivado. Στο αρχικό παράθυρο επιλέγουμε: Tools → Run Tcl Script. Στο παράθυρο που θα ανοίξει πηγαίνουμε στο φάκελο που έχουμε κάνει export και μέσα στο φάκελο scripts (πχ. /Desktop/dvlsi2021_lab5/scripts) επιλέγουμε το αρχείο dvlsi2021_lab5_prj.tcl. Πατάμε OK. Το Vivado αυτόματα θα δημιουργήσει ένα καινούργιο project για να υλοποιήσετε την άσκηση. Από αυτό το σημείο και μετά προσθέτουμε τα δικά μας αρχεία.

Αρχικά, προσθέτουμε το αρχείο με την υλοποίηση σου Debayering φίλτρου (Add Sources). Στο project υπάρχουν τα αρχεία dvlsi2021_lab5_top και design_1_wrapper. Εμείς θα τροποποιήσουμε τον κώδικα του dvlsi2021_lab5_top αρχείου.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dvlsi2021_lab5_top is
  port (
        DDR_cas_n         : inout STD_LOGIC;
        DDR_cke           : inout STD_LOGIC;
        DDR_ck_n          : inout STD_LOGIC;
        DDR_ck_p          : inout STD_LOGIC;
        DDR_cs_n          : inout STD_LOGIC;
        DDR_reset_n       : inout STD_LOGIC;
        DDR_odt           : inout STD_LOGIC;
        DDR_ras_n         : inout STD_LOGIC;
        DDR_we_n          : inout STD_LOGIC;
        DDR_ba            : inout STD_LOGIC_VECTOR( 2 downto 0);
        DDR_addr          : inout STD_LOGIC_VECTOR(14 downto 0);
        DDR_dm            : inout STD_LOGIC_VECTOR( 3 downto 0);
```

```
        DDR_dq            : inout STD_LOGIC_VECTOR(31 downto 0);
        DDR_dqs_n         : inout STD_LOGIC_VECTOR( 3 downto 0);
        DDR_dqs_p         : inout STD_LOGIC_VECTOR( 3 downto 0);
        FIXED_IO_mio      : inout STD_LOGIC_VECTOR(53 downto 0);
        FIXED_IO_ddr_vrn  : inout STD_LOGIC;
        FIXED_IO_ddr_vrp  : inout STD_LOGIC;
        FIXED_IO_ps_srstb : inout STD_LOGIC;
        FIXED_IO_ps_clk   : inout STD_LOGIC;
        FIXED_IO_ps_porb  : inout STD_LOGIC
      );
end entity; -- dvlsi2021_lab5_top

architecture arch of dvlsi2021_lab5_top is

  constant N : integer := 1024;

  component design_1_wrapper is
    port (
          DDR_cas_n         : inout STD_LOGIC;
          DDR_cke           : inout STD_LOGIC;
          DDR_ck_n          : inout STD_LOGIC;
          DDR_ck_p          : inout STD_LOGIC;
          DDR_cs_n          : inout STD_LOGIC;
          DDR_reset_n       : inout STD_LOGIC;
          DDR_odt           : inout STD_LOGIC;
          DDR_ras_n         : inout STD_LOGIC;
          DDR_we_n          : inout STD_LOGIC;
          DDR_ba            : inout STD_LOGIC_VECTOR( 2 downto 0);
          DDR_addr          : inout STD_LOGIC_VECTOR(14 downto 0);
          DDR_dm            : inout STD_LOGIC_VECTOR( 3 downto 0);
          DDR_dq            : inout STD_LOGIC_VECTOR(31 downto 0);
          DDR_dqs_n         : inout STD_LOGIC_VECTOR( 3 downto 0);
          DDR_dqs_p         : inout STD_LOGIC_VECTOR( 3 downto 0);
          FIXED_IO_mio      : inout STD_LOGIC_VECTOR(53 downto 0);
          FIXED_IO_ddr_vrn  : inout STD_LOGIC;
          FIXED_IO_ddr_vrp  : inout STD_LOGIC;
          FIXED_IO_ps_srstb : inout STD_LOGIC;
          FIXED_IO_ps_clk   : inout STD_LOGIC;
          FIXED_IO_ps_porb  : inout STD_LOGIC;
```

```
-------------------------------------------------------------------------
-----
          ------------------------------------------------ PL (FPGA)
COMMON INTERFACE
          ACLK                                    : out STD_LOGIC;
          ARESETN                                 : out STD_LOGIC;

-------------------------------------------------------------------------
---------------
          -- PS2PL-DMA AXI4-STREAM MASTER INTERFACE TO ACCELERATOR
AXI4-STREAM SLAVE INTERFACE
          M_AXIS_TO_ACCELERATOR_tdata         : out
STD_LOGIC_VECTOR(7 downto 0);
          --M_AXIS_TO_ACCELERATOR_tkeep         : out
STD_LOGIC_VECTOR( 0    to 0);
          M_AXIS_TO_ACCELERATOR_tlast         : out STD_LOGIC;
          M_AXIS_TO_ACCELERATOR_tready        : in  STD_LOGIC;
          M_AXIS_TO_ACCELERATOR_tvalid        : out STD_LOGIC;

-------------------------------------------------------------------------
---------------
          -- ACCELERATOR AXI4-STREAM MASTER INTERFACE TO PL2P2-DMA
AXI4-STREAM SLAVE INTERFACE
          S_AXIS_S2MM_FROM_ACCELERATOR_tdata  : in
STD_LOGIC_VECTOR(31 downto 0);
          --S_AXIS_S2MM_FROM_ACCELERATOR_tkeep  : in
STD_LOGIC_VECTOR( 3 downto 0);
          S_AXIS_S2MM_FROM_ACCELERATOR_tlast  : in  STD_LOGIC;
          S_AXIS_S2MM_FROM_ACCELERATOR_tready : out STD_LOGIC;
          S_AXIS_S2MM_FROM_ACCELERATOR_tvalid : in  STD_LOGIC
         );
  end component design_1_wrapper;

  component debayering is
    generic (N: integer := 1024);
    Port (
          clock : in std_logic;
          reset : in std_logic;
```

```
            valid_in : in std_logic;
            pixel : in std_logic_vector(7 downto 0);
            new_image : in std_logic;
            R : out std_logic_vector(7 downto 0);
            G : out std_logic_vector(7 downto 0);
            B : out std_logic_vector(7 downto 0);
            valid_out : out std_logic;
            image_finished : out std_logic
        );
end component;

-----------------------------------------
-- INTERNAL SIGNAL & COMPONENTS DECLARATION

  signal aclk    : std_logic;
  signal aresetn : std_logic;

  signal tmp_tdata  : std_logic_vector(7 downto 0);
  signal R : std_logic_vector(7 downto 0);
  signal G : std_logic_vector(7 downto 0);
  signal B : std_logic_vector(7 downto 0);
  signal tmp_tdata_32 : std_logic_vector(31 downto 0);
  signal tmp_tlast_ps_to_pl : std_logic;
  signal tmp_tlast_pl_to_ps : std_logic;
  signal tmp_tready_s : std_logic;
  signal tmp_tready_m : std_logic;
  signal tmp_tvalid_in : std_logic;
  signal tmp_tvalid_out : std_logic;
  signal tmp_tkeep  : std_logic_vector(0 downto 0);
  signal tmp_tlast  : std_logic;
  signal tmp_tready : std_logic;
  signal tmp_tvalid : std_logic;
  signal count : std_logic_vector(3 downto 0) := (others => '0');
  signal new_image : std_logic;

begin

  tmp_tready_s <= tmp_tready_m and tmp_tvalid_in when tmp_tvalid_out
= '1' else tmp_tvalid_in;
```

```
  tmp_tdata_32 <= "00000000" & R & G & B;

  PROCESSING_SYSTEM_INSTANCE : design_1_wrapper
    port map (
              DDR_cas_n         => DDR_cas_n,
              DDR_cke           => DDR_cke,
              DDR_ck_n          => DDR_ck_n,
              DDR_ck_p          => DDR_ck_p,
              DDR_cs_n          => DDR_cs_n,
              DDR_reset_n       => DDR_reset_n,
              DDR_odt           => DDR_odt,
              DDR_ras_n         => DDR_ras_n,
              DDR_we_n          => DDR_we_n,
              DDR_ba            => DDR_ba,
              DDR_addr          => DDR_addr,
              DDR_dm            => DDR_dm,
              DDR_dq            => DDR_dq,
              DDR_dqs_n         => DDR_dqs_n,
              DDR_dqs_p         => DDR_dqs_p,
              FIXED_IO_mio      => FIXED_IO_mio,
              FIXED_IO_ddr_vrn  => FIXED_IO_ddr_vrn,
              FIXED_IO_ddr_vrp  => FIXED_IO_ddr_vrp,
              FIXED_IO_ps_srstb => FIXED_IO_ps_srstb,
              FIXED_IO_ps_clk   => FIXED_IO_ps_clk,
              FIXED_IO_ps_porb  => FIXED_IO_ps_porb,

-------------------------------------------------------------------
-----
              ------------------------------------------------ PL
(FPGA) COMMON INTERFACE
              ACLK                                    => aclk,    --
clock to accelerator
              ARESETN                                 => aresetn, --
reset to accelerator, active low

-------------------------------------------------------------------
---------------
              -- PS2PL-DMA AXI4-STREAM MASTER INTERFACE TO
ACCELERATOR AXI4-STREAM SLAVE INTERFACE
```

```
              M_AXIS_TO_ACCELERATOR_tdata         => tmp_tdata,
              --M_AXIS_TO_ACCELERATOR_tkeep         => tmp_tkeep,
              M_AXIS_TO_ACCELERATOR_tlast         =>
tmp_tlast_ps_to_pl,
              M_AXIS_TO_ACCELERATOR_tready        => tmp_tready_s,
              M_AXIS_TO_ACCELERATOR_tvalid        => tmp_tvalid_in,

------------------------------------------------------------------------
---------------
              -- ACCELERATOR AXI4-STREAM MASTER INTERFACE TO
PL2P2-DMA AXI4-STREAM SLAVE INTERFACE
              S_AXIS_S2MM_FROM_ACCELERATOR_tdata  => tmp_tdata_32,
              --S_AXIS_S2MM_FROM_ACCELERATOR_tkeep  => tmp_tkeep &
tmp_tkeep & tmp_tkeep & tmp_tkeep,
              S_AXIS_S2MM_FROM_ACCELERATOR_tlast  =>
tmp_tlast_pl_to_ps,
              S_AXIS_S2MM_FROM_ACCELERATOR_tready => tmp_tready_m,
              S_AXIS_S2MM_FROM_ACCELERATOR_tvalid => tmp_tvalid_out
             );

ACCELERATOR : debayering
    port map(
        clock => aclk,
        reset => aresetn,
        new_image => new_image,
        valid_in => tmp_tvalid_in,
        pixel => tmp_tdata,
        R => R,
        G => G,
        B => B,
        valid_out => tmp_tvalid_out,
        image_finished => tmp_tlast_pl_to_ps
    );
----------------------------
-- COMPONENTS INSTANTIATIONS

process(aclk)
begin
    if (aclk'event and aclk = '1') then
```

```
            new_image <= tmp_tlast_ps_to_pl;
        end if;
    end process;
    end architecture; -- arch
```

Στον κώδικα αφού αρχικοποιηθεί το `entity dvlsi2021_lab5_top,` ορίζεται η αρχιτεκτονική που ορίζει όλα τα εσωτερικά σήματα και τα components. Συγκεκριμένα ορίζεται το component του `design_1_wrapper` το οποίο είναι το interface για το FPGA design και περιέχει σήματα για την μεταφορά δεδομένων και το component του `debayering.`

Επίσης αρχικοποιούνται σήματα που θα μας βοηθήσουν για την υλοποίηση μας. Τα σήματα `tmp_tdata, R, G, B` αντιπροσωπεύουν τα data buses για την μεταφορά των pixel και των χρωματικών συνιστωσών. Το σήμα `tmp_tdata_32` είναι το 32-bit data bus που συνδυάζουμε τα R, G και B. Τα σήματα `tmp_tlast_ps_to_pl` και `tmp_tlast_pl_to_ps` είναι σήματα ελέγχου που σηματοδοτούν το τέλος μίας μεταφοράς δεδομένων.

Στην συνέχεια τα block `PROCESSING_SYSTEM_INSTANCE` και `ACCELERATOR` κάνουν map τα σήματα του `dvlsi2021_lab5_top` με τα σήματα των components.

Αυτό που υλοποιούμε στον κώδικα είναι από την μεριά του PS2PL να βεβαιωθούμε ότι όταν δεχόμαστε `TVALID` θα πρέπει να παράξουμε αντίστοιχο `TREADY` προκειμένου να λάβουμε τα απαραίτητα δεδομένα. Επομένως, κάνουμε το `TREADY` map στο `valid_in` όσο δεν έχουμε `valid_out`. Επιπλέον, πρέπει να ληφθεί υπόψιν και το σήμα `TREADY` του master έτσι ώστε αν αυτό γίνει μηδέν, να γίνει μηδέν και το `TREADY` του slave, προκειμένου να μην δεχθούμε άλλα δεδομένα και το Debayering να “παγώσει” μέχρι το PL2PS να είναι έτοιμο να ξαναδεχθεί δεδομένα.

Από την μεριά του PL2PS παράγουμε το 32-bit σήμα `TDATA` με τα δεδομένα από το Debayering ως R (8 bit) & G (8 bit) & B (8 bit) και 8 μηδενικά στην αρχή για να συμπληρωθούν τα 32bit, το σήμα `TLAST` δηλαδή το image_finished με το οποίο σηματοδοτείται η λήξη της λήψης των δεδομένων και το σήμα `TVALID` δηλαδή το αντίστοιχο `valid_out` του Debayering.

Άρα μπορούμε να περιγράψουμε το Data Flow ως εξής:

1. Master to Slave (PS to Accelerator):

    - Το PS στέλνει δεδομένα (`TDATA`) στον accelerator μαζί με τα σήματα ελέγχου `TLAST` και `TVALID.`
    - Ο accelerator περιμένει το PS να στείλει το `TREADY` προτού δεχτεί δεδομένα.
    - Μόλις ο accelerator δεχτεί δεδομένα, τα επεξεργάζεται ανάλογα.

2. Slave to Master (Accelerator to PS):

    - Μετά την επεξεργασία, ο accelerator στέλνει τα επεξεργασμένα δεδομένα πίσω στο PS.
    - Επιβεβαιώνει το `TVALID` για να υποδείξει έγκυρα δεδομένα και περιμένει το PS να δηλώσει `TREADY` πριν στείλει το επόμενο σύνολο δεδομένων.
    - Το PS διαβάζει τα δεδομένα από τα `TDATA` και τα χρησιμοποιεί για περαιτέρω επεξεργασία ή αποθήκευση.

Αφού αποθηκεύσουμε τον κώδικα μας, μπορούμε κατευθείαν να κάνουμε Generate Bitstream καθώς το Block Design είναι ήδη υλοποιημένο.

Αφού ολοκληρωθέι αυτό το κομμάτι εκτελούμε τα βήματα File → Export → Export Hardware και τέλος File → Launch SDK και ανοίγει αυτόματα το SDK.

Ακολουθούμε τα εξής βήματα, πάλι: File → New → Application Project, ονομάζουμε το πρόγραμμα μας Next → Hello World για να δημιουργηθεί αυτόματα το .c file μέσα στον φάκελο src που θα τροποποιήσουμε. Ο κώδικας του SDK φαίνεται παρακάτω:

```
#include <stdio.h>
#include <unistd.h>
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include "xparameters_ps.h"
#include "xaxidma.h"
#include "xtime_l.h"

#define TX_DMA_ID XPAR_PS2PL_DMA_DEVICE_ID
      //PS to PL
#define TX_DMA_MM2S_LENGTH_ADDR (XPAR_PS2PL_DMA_BASEADDR + 0x28)     // Reports
actual number of bytes transfERRORed from PS->PL (use Xil_In32 for report)

#define RX_DMA_ID XPAR_PL2PS_DMA_DEVICE_ID
      //PL to PS
#define RX_DMA_S2MM_LENGTH_ADDR (XPAR_PL2PS_DMA_BASEADDR + 0x58)     // Reports
actual number of bytes transfERRORed from PL->PS (use Xil_In32 for report)

#define TX_BUFFER (XPAR_DDR_MEM_BASEADDR + 0x10000000)                    // 0
+ 256MByte
#define RX_BUFFER (XPAR_DDR_MEM_BASEADDR + 0x18000000)                    // 0
+ 384MByte

/* User application global variables & defines */
```

```
#define N 0x20
#define IMG_LEN 0x400
      //(N*N)

/* Device instance definitions*/
XAxiDma TXAxiDma; //PS2PL DMA Engine
XAxiDma RXAxiDma; //PL2PS DMA Engine

unsigned char pixels[IMG_LEN];

void generateRandomPixels() {
    for (int i = 0; i < IMG_LEN; i++) {
        pixels[i] = rand() % 256; // Generate random number between 0 and 255
    }
}


int main(){

    Xil_DCacheDisable();

    XTime preExecCyclesFPGA = 0;
    XTime postExecCyclesFPGA = 0;
    XTime preExecCyclesSW = 0;
    XTime postExecCyclesSW = 0;

    print("HELLO : Start of the Program Complete\r\n");

    // User application local variables
    init_platform();

    /*************************DEVICE
INITIALIZATION*****************************/
    u8 *TxPtr;
    u32 *RxPtr;
    int State;

    TxPtr = (u8 *)TX_BUFFER;                          //8bit data to
accelerator
    RxPtr = (u32 *)RX_BUFFER;                         //32bit data from
accelerator

    // Step 1: Initialize TX-DMA Device (PS->PL)
```

```
    /* Configure & Initialize PS2PL */

    XAxiDma_Config *TXConfigPtr;

    int TXState;
    TXConfigPtr = XAxiDma_LookupConfig(TX_DMA_ID);
    if (!TXConfigPtr)
    {
        xil_printf("No Configuration found for %d\r\n", TX_DMA_ID);
        return XST_FAILURE;
    }

    TXState = XAxiDma_CfgInitialize(&TXAxiDma, TXConfigPtr);

    if (TXState != XST_SUCCESS)
    {
        xil_printf("Initialization Failed %d\r\n", TXState);
        return XST_FAILURE;
    }

    XAxiDma_IntrDisable(&TXAxiDma, XAXIDMA_IRQ_ALL_MASK,
XAXIDMA_DEVICE_TO_DMA);        // Disable intERRORupts, we use polling mode
    XAxiDma_IntrDisable(&TXAxiDma, XAXIDMA_IRQ_ALL_MASK,
XAXIDMA_DMA_TO_DEVICE);

    // Step 2: Initialize RX-DMA Device (PL->PS)
    /* Configure & Initialize PS2PL */

    XAxiDma_Config *RXConfigPtr;
    int RXState;

    RXConfigPtr = XAxiDma_LookupConfig(RX_DMA_ID);
    if (!RXConfigPtr)
    {
        xil_printf("No Configuration found for %d\r\n", RX_DMA_ID);
        return XST_FAILURE;
    }

    RXState = XAxiDma_CfgInitialize(&RXAxiDma, RXConfigPtr);

    if (RXState != XST_SUCCESS){
        xil_printf("Initialization failed %d\r\n", RXState);
        return XST_FAILURE;
    }
```

```
    XAxiDma_IntrDisable(&RXAxiDma, XAXIDMA_IRQ_ALL_MASK,
XAXIDMA_DEVICE_TO_DMA);        // Disable intERRORupts, we use polling mode
    XAxiDma_IntrDisable(&RXAxiDma, XAXIDMA_IRQ_ALL_MASK,
XAXIDMA_DMA_TO_DEVICE);
    print("Initialization DMA Complete\r\n");


      generateRandomPixels();

      for(int i = 0; i < IMG_LEN; i ++){ //Data to accelerator

        TxPtr[i] = pixels[i];
//        printf("%d\n", i);
    }

    for(int i = 0; i < 10; i ++){

        RxPtr[i] = i;
    }


/*****************************************************************************/
    print("Data Acquisition Complete\r\n");

    XTime_GetTime(&preExecCyclesFPGA);

    int sent, received = 0;
//    for(int i = 0; i < 10; i++)
//        printf("%lu \n",(unsigned long)RxPtr[i]);

    // Step 3 : Perform FPGA processing
    // 3a: Setup RX-DMA transaction

    State = XAxiDma_SimpleTransfer(&RXAxiDma,(UINTPTR) RxPtr,
                                        IMG_LEN*4, XAXIDMA_DEVICE_TO_DMA);
    if (State != XST_SUCCESS)
    {
        return XST_FAILURE;
    }
    print("Setup RX-DMA transaction Complete\r\n");

    // 3b: Setup TX-DMA transaction
    State = XAxiDma_SimpleTransfer(&TXAxiDma,(UINTPTR) TxPtr,
                                        IMG_LEN, XAXIDMA_DMA_TO_DEVICE);
    if (State != XST_SUCCESS)
```

```
    {
        return XST_FAILURE;
    }
    print("Setup TX-DMA transaction Complete\r\n");

    // 3c: Wait for TX-DMA & RX-DMA to finish

    sleep(5);
    sent = Xil_In32(TX_DMA_MM2S_LENGTH_ADDR);
    received = Xil_In32(RX_DMA_S2MM_LENGTH_ADDR);

    // Expecting sent == N*N && received == N*N*4
    printf("Sent = %d, Received = %d \n", sent, received);

    if (XAxiDma_Busy(&RXAxiDma,XAXIDMA_DEVICE_TO_DMA)) print("RX DEVICE -> DMA
WAITING...\r\n");
    if (XAxiDma_Busy(&RXAxiDma,XAXIDMA_DMA_TO_DEVICE)) print("RX DMA -> DEVICE
WAITING...\r\n");
    if (XAxiDma_Busy(&TXAxiDma,XAXIDMA_DEVICE_TO_DMA)) print("TX DEVICE -> DMA
WAITING...\r\n");
    if (XAxiDma_Busy(&TXAxiDma,XAXIDMA_DMA_TO_DEVICE)) print("TX DMA -> DEVICE
WAITING...\r\n");

//    for(int i = 0; i < 10; i++)
//        printf("%lu \n",(unsigned long)RxPtr[i]);

    XTime_GetTime(&postExecCyclesFPGA);


/*****************************************************************************/
    print("RX-DMA and TX-DMA Complete\r\n");

    XTime_GetTime(&preExecCyclesSW);

    // Step 5: Perform SW processing

    int *RED_SW = (int *)malloc(N*N*sizeof(int));
    int *GREEN_SW = (int *)malloc(N*N*sizeof(int));
    int *BLUE_SW = (int *)malloc(N*N*sizeof(int));
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            int r = (j < N - 1) ? (j + 1) : 0;
            int l = (j > 0) ? (j - 1) : 0;
```

```
            int u = (i > 0) ? (i - 1) : 0;
            int d = (i < N - 1) ? (i + 1) : 0;
            int right = (j < N - 1) ? pixels[r + N * i] : 0;
            int left = (j > 0) ? pixels[i * N + l] : 0;
            int up = (i > 0) ? pixels[u * N + j] : 0;
            int down = (i < N - 1) ? pixels[d * N + j] : 0;
            int upleft = ((i > 0) && (j > 0)) ? pixels[u * N + l] : 0;
            int upright = ((i > 0) && j < (N - 1)) ? pixels[u * N + r] : 0;
            int downleft = ((i < N - 1) && (j > 0)) ? pixels[d * N + l] : 0;
            int downright = ((i < N - 1) && (j < N - 1)) ? pixels[d * N + r] :
0;
            int x = pixels[i*N+j];
            if (i % 2)
            {
                if (j % 2)
                { //green
                    RED_SW[i*N+j] = (right + left) / 2;
                    GREEN_SW[i*N+j] = x;
                    BLUE_SW[i*N+j] = (up + down) / 2;
                }
                else
                { //red
                    RED_SW[i*N+j] = x;
                    GREEN_SW[i*N+j] = (up + down + right + left) / 4;
                    BLUE_SW[i*N+j] = (upleft + upright + downleft + downright)
/ 4 ;
                }
            }
            else
            {
                if (j % 2)
                { //blue
                    RED_SW[i*N+j] = (upleft + upright + downleft + downright) /
4;
                    GREEN_SW[i*N+j] = (up + down + right + left) / 4;
                    BLUE_SW[i*N+j] = x;
                }
                else
                { //green
                    RED_SW[i*N+j] = (up + down) / 2;
                    GREEN_SW[i*N+j] = x;
                    BLUE_SW[i*N+j] = (right + left) / 2 ;
                }
            }
        }
```

```
    }
    print("Software Pixel Calculation Complete\r\n");
    XTime_GetTime(&postExecCyclesSW);


/*****************************************************************************/
    // Step 6: Compare FPGA and SW results

    int *RED_HW = (int *)malloc(N*N*sizeof(int));
    int *GREEN_HW = (int *)malloc(N*N*sizeof(int));
    int *BLUE_HW = (int *)malloc(N*N*sizeof(int));
    int ERROR = 0;
    float percentage_error, FPGA_cycles, SW_cycles, speedup;



    for (int i = 0; i < IMG_LEN; i++)
    {
        RED_HW[i] = RxPtr[i] & 0x0000ff;
        GREEN_HW[i] = (RxPtr[i] & 0x00ff00) >> 2;
        BLUE_HW[i] = (RxPtr[i] & 0xff0000) >> 4;
        if (RED_HW[i] == RED_SW[i] && GREEN_HW[i] == GREEN_SW[i] && BLUE_HW[i]
== BLUE_SW[i]) continue;
        else ERROR++;
    }

    // 6a: Report total percentage Error
    percentage_error = ((float)ERROR / IMG_LEN) * 100.0f; // Cast ERROR to
float
    printf("Total Percentage Error: %.5f %% \n", percentage_error); // %.5f for
5 decimal places

    // 6b: Report FPGA execution time in cycles (use preExecCyclesFPGA and
postExecCyclesFPGA)
    FPGA_cycles = (float)(postExecCyclesFPGA - preExecCyclesFPGA); // Cast to
float
    printf("FPGA execution time in cycles: %f \n", FPGA_cycles);

    // 6c: Report SW execution time in cycles (use preExecCyclesSW and
postExecCyclesSW)
    SW_cycles = (float)(postExecCyclesSW - preExecCyclesSW); // Cast to float
    printf("SW execution time in cycles: %f \n", SW_cycles);

    // 6d: Report speedup (SW_execution_time / FPGA_exection_time)
    speedup = FPGA_cycles / SW_cycles ;
```

```
    printf("Speedup: %.5f \n", speedup);

    printf("END\r\n");
    printf("\n");
    cleanup_platform();

}
```

Ο κώδικας αυτός είναι υπεύθυνος για την μεταφορά και την επεξεργασία δεδομένων μεταξύ του PL και του PS. Αρχικά βάζουμε τα header files και τα definitions που θα χρειαστούμε. Τα definitions αφορούν τα ID του PS-to-PL και PL-to-PS DMA controllers, τα base addresses του buffer για την λήψη και μετάδοση των δεδομένων, το μέγεθος N όπου είναι οι διαστάση της εικόνας μας και το IMG_LEN, ο συνολικός αριθμός των pixel δηλαδή $N \times N$.

Στο main κομμάτι του κώδικα, ρυθμίζονται τα DMA transactions τόσο για το TX όσο και για το RX. Αυτό περιλαμβάνει τη διαμόρφωση των DMA controllers και την έναρξη μεταφοράς δεδομένων. Τα δεδομένα εισόδου (pixel) γίνονται generated μέσα στον κώδικα και μεταφέρονται από το PS στο PL για επεξεργασία. Τα επεξεργασμένα δεδομένα στη συνέχεια μεταφέρονται πίσω από το PL στο PS.

Το FPGA εκτελεί λειτουργίες επεξεργασίας εικόνας στα λαμβανόμενα δεδομένα, ενώ το λογισμικό εκτελεί τις ίδιες λειτουργίες σε ξεχωριστό αντίγραφο των δεδομένων και στην συνέχεια ο κώδικας συγκρίνει τα αποτελέσματα που λαμβάνονται από το FPGA και τις εφαρμογές λογισμικού. Υπολογίζει το ποσοστό σφάλματος μεταξύ των δύο αποτελεσμάτων και μετρά τους χρόνους εκτέλεσης και των δύο υλοποιήσεων για να αξιολογήσει το speedup που επιτυγχάνεται από το FPGA, δηλαδή τον λόγο `speedup = FPGA_cycles / SW_cycles`.

Τέλος, ο κώδικας απελευθερώνει πόρους μνήμης και καθαρίζει την πλατφόρμα. Τρέχοντας τώρα ρο project στο Terminal μας έχουμε την αναμενόμενη έξοδο.