ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΙΣΤΩΝ

ΨΗΦΙΑΚΑ VLSI – 8ο ΕΞΑΜΗΝΟ

Μαρκέτος Νικόδημος – ΑΜ: 03117095

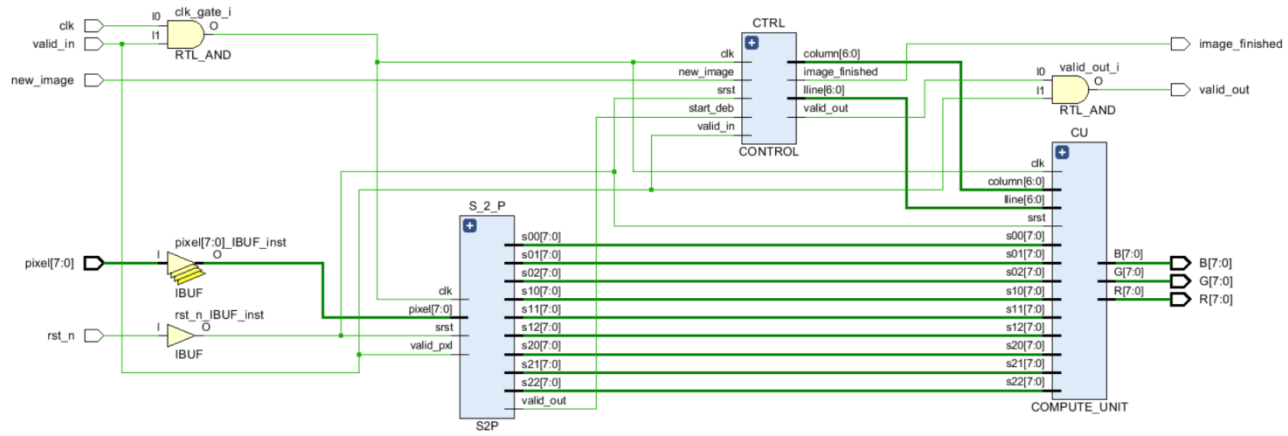
Τζομάκα Αφροδίτη – ΑΜ: 03117107

Ομάδα Β2

Υλοποίηση Debayering φίλτρου σε FPGA

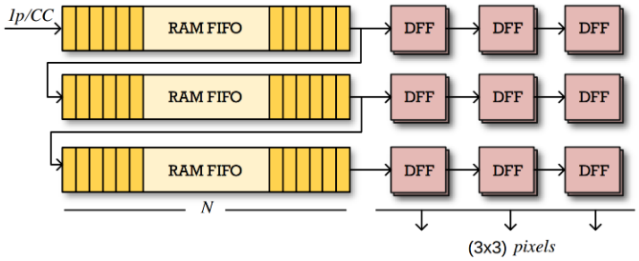
Λεπτομέρειες – Ανάλυση Υλοποίησης:

Στην παρούσα εργαστηριακή άσκηση κληθήκαμε να υλοποιήσουμε το Debayering φίλτρο για επεξεργασία εικόνας με στόχο την μετατροπή της σε RGB. Η υλοποίηση που ακολουθήσαμε είναι η προτεινόμενη που δίνεται από την εκφώνηση. Παρακάτω φαίνεται το σχηματικό του elaborated design από το Vivado για το συνολικό πρόγραμμα (DEBAYERING.vhd):



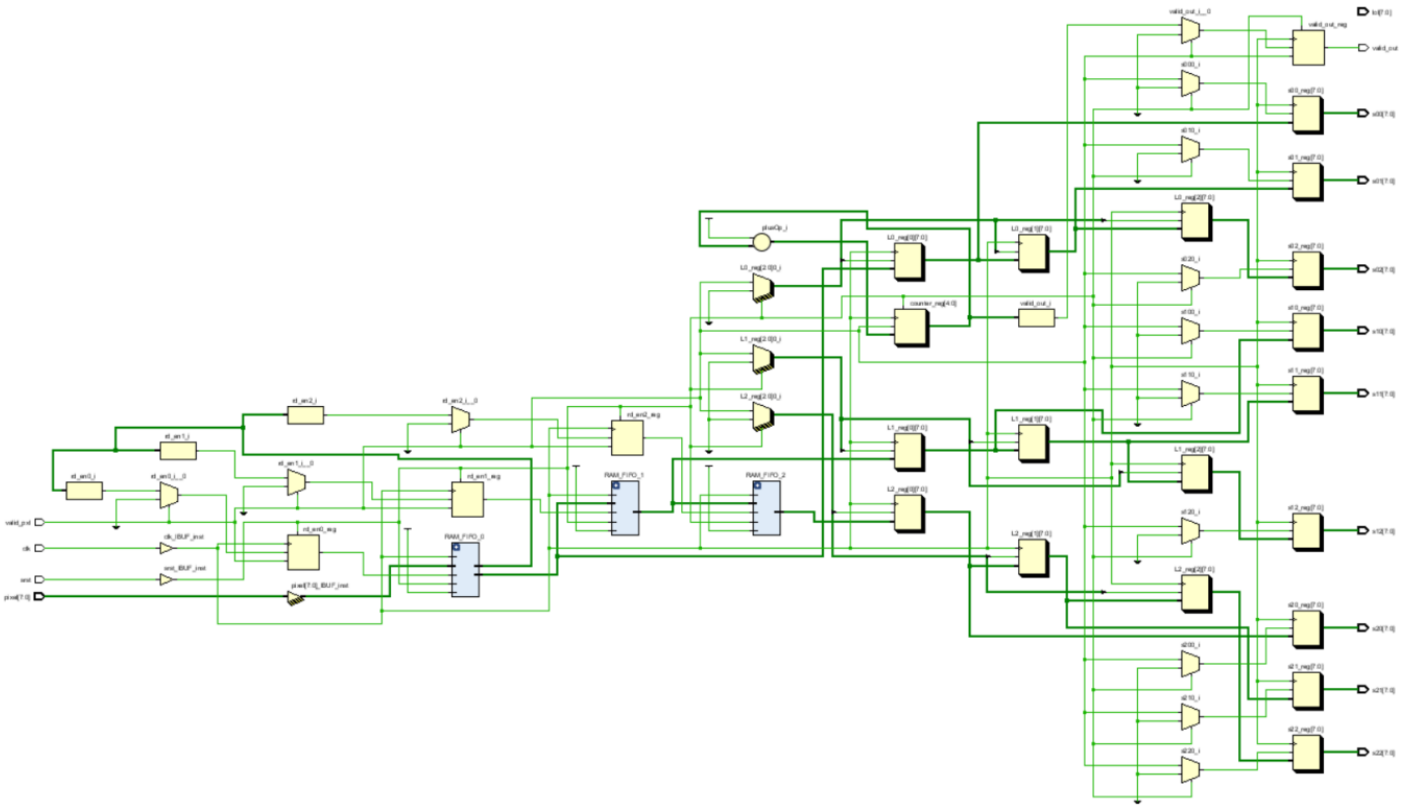
και ακολουθεί η ανάλυση κάθε ενός component ξεχωριστά.

- **S2P – Serial to Parallel Converter**: Πρόκειται ουσιαστικά για 3 RAM FIFO κατάλληλα συνδεδεμένες με μια συστοιχία 9 shift registers όπως προτείνεται και από την εκφώνηση.



Χρησιμοποιήσαμε το έτοιμο IP της Xilinx από το Vivado για τις RAM FIFO, επιλέγοντας ως μέγεθος τις 256 θέσεις (max μέγεθος εικόνας που θα μας δινόταν) και embedded output

registers (οι οποίοι προσθέτουν μία επιπλέον καθυστέρηση στο read operation – read latency = 2). Προκειμένου να καταφέρουμε οι FIFO να συμπεριφέρονται παραμετρικά ως προς το μέγεθος της εικόνας χρησιμοποιούμε την έξοδο datacount του IP η οποία μετρά το πλήθος των δεδομένων που περιέχει η FIFO. Για τον συντονισμό με την συστοιχία των registers θα πρέπει να ξεκινάμε να διαβάζουμε = αδειάζουμε τις FIFO μόλις η πρώτη περιέχει N – 3 δεδομένα. Ύστερα, χρησιμοποιείται ένας ακόμα counter για την εμφάνιση του vadid_out. Το valid_out σηματοδοτεί την πρώτη σωστή έξοδο του S2P, δηλαδή πότε μπορεί να υπολογιστεί το RGB του πρώτου pixel ή ισοδύναμα πότε αυτό φτάνει στο μεσαίο DFF της παραπάνω εικόνας. Αυτό συμβαίνει όταν ο counter ισούται με 2N + 2. Ωστόσο για λόγους συγχρονισμού με το compute unit το εμφανίζουμε στο 2N + 3. Παρατίθενται το σχηματικό και ο κώδικας του S2P.



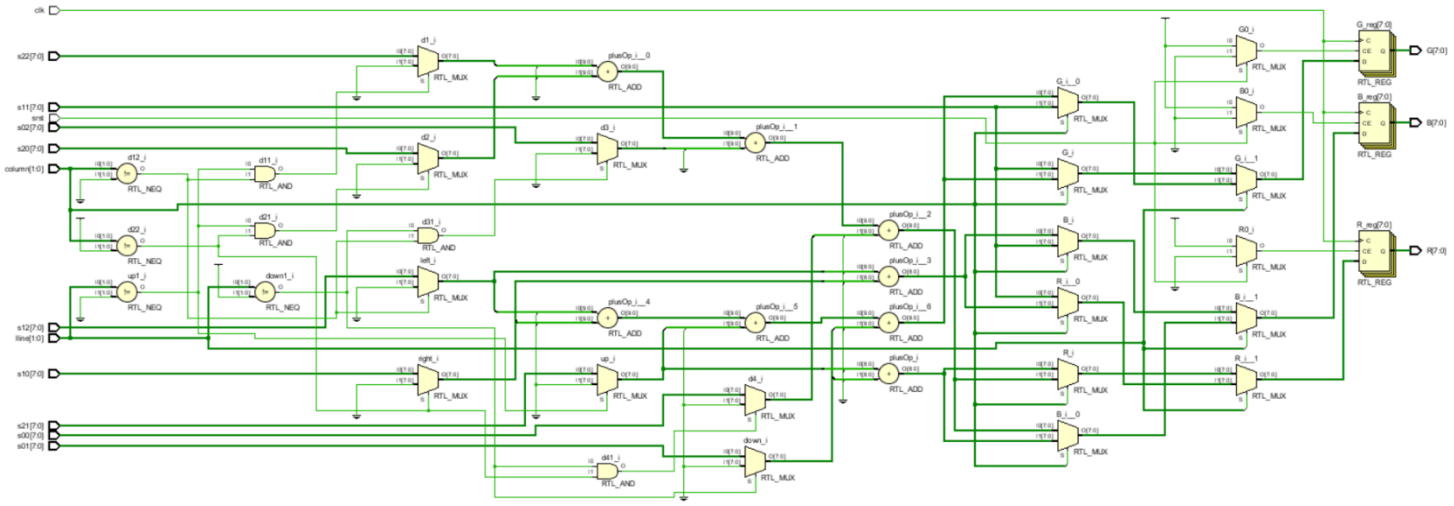
S2P

```
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  := 4 );
    port (
        clk : IN STD_LOGIC;
        srst : IN STD_LOGIC;
        pixel : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        valid_pxl : IN STD_LOGIC;
        s00 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        s01 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        s02 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        s10 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        s11 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        s12 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        s20 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        s21 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        s22 : 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(7 DOWNTO 0)
            );
    END COMPONENT;

    type arr is array (2 downto 0) of std_logic_vector (7 downto 0);
    signal L0 : arr := (others => (others => '0'));
    signal L1 : arr := (others => (others => '0'));
    signal L2 : arr := (others => (others => '0'));
    signal counter : std_logic_vector(integer(ceil(log2(real(N*N)))) downto 0) :=  (others => '0') ;
    signal fout2, fout1, fout0 : std_logic_vector(7 downto 0) :=  (others => '0') ;
    signal  rd_en0: std_logic := '0';
    signal  rd_en1: std_logic := '0';
    signal  rd_en2: std_logic := '0';
    signal full0, full1, full2, empty0, empty1, empty2, valid0, valid1, valid2: std_logic;
    signal datacount0, datacount1, datacount2: std_logic_vector(7 downto 0);

begin
    RAM_FIFO_0 : fifo_generator_0
    PORT MAP (
        clk => clk,
        srst => srst,
        din => pixel,
        wr_en => '1',
        rd_en => rd_en0,
        dout => fout0,
        full => full0,
        empty => empty0,
        valid => valid0,
        data_count => datacount0
    );
    RAM_FIFO_1 : fifo_generator_0
    PORT MAP (
        clk => clk,
        srst => srst,
        din => fout0,
        wr_en => '1',
        rd_en => rd_en1,
        dout => fout1,
        full => full1,
        empty => empty1,
        valid => valid1,
        data_count => datacount1
    );
    RAM_FIFO_2 : fifo_generator_0
    PORT MAP (
        clk => clk,
        srst => srst,
        din => fout1,
        wr_en => '1',
        rd_en => rd_en2,
        dout => fout2,
        full => full2,
        empty => empty2,
        valid => valid2,
        data_count => datacount2
    );

    process(clk, srst)
    begin
        if srst = '1' then
            rd_en0 <= '0';  rd_en1 <= '0';  rd_en2 <= '0';
            valid_out <= '0';  counter <= (others => '0');
        elsif rising_edge(clk) then
```

```
            if valid_pxl = '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-3 then
                    rd_en0 <= '1';
                    rd_en1 <= '1';
                    rd_en2 <= '1';
                end if;
                L0 <= L0(1 downto 0) & fout0;
                L1 <= L1(1 downto 0) & fout1;
                L2 <= L2(1 downto 0) & fout2;
                s00 <= L0(0); s01 <= L0(1); s02 <= L0(2);
                s10 <= L1(0); s11 <= L1(1); s12 <= L1(2);
                s20 <= L2(0); s21 <= L2(1); s22 <= L2(2);
            end if;
        end if ;
    end process;

end Behavioral ; -- Behavioral
```

- **CU – Compute Unit**: Στο σημείο αυτό, πραγματοποιούνται όλες οι αριθμητικές πράξεις και σύμφωνα με τα σήματα από το control εμφανίζονται τα αντίστοιχα RGB. Πιο συγκεκριμένα, τρεις είναι οι δουλειές που πρέπει να γίνουν εδώ. Αρχικά, η σωστή τοποθέτηση των τιμών από το S2P στις πάνω, κάτω, δεξιά, αριστερά και διαγώνιες θέσεις της κάθε 3χ3 «γειτονιάς» ενώ ταυτόχρονα και ο έλεγχος για τις ακραίες περιπτώσεις όπου κάποιοι «γείτονες» θα πρέπει να μηδενιστούν. Στη συνεχεία, με βάση τις παραπάνω τιμές υπολογίζονται 4 μέσες τιμές: πάνω – κάτω, δεξιά – αριστερά, διαγώνια και πάνω – κάτω – δεξιά – αριστερά. Τέλος, ελέγχοντας σε ποιον συνδυασμό στήλης – γραμμής βρισκόμαστε (ζυγή/μονή) βγάζουμε στα RGB τις αντίστοιχες μέσες τιμές. Παρατίθενται το σχηματικό και ο κώδικας του COMPUTE_UNIT.



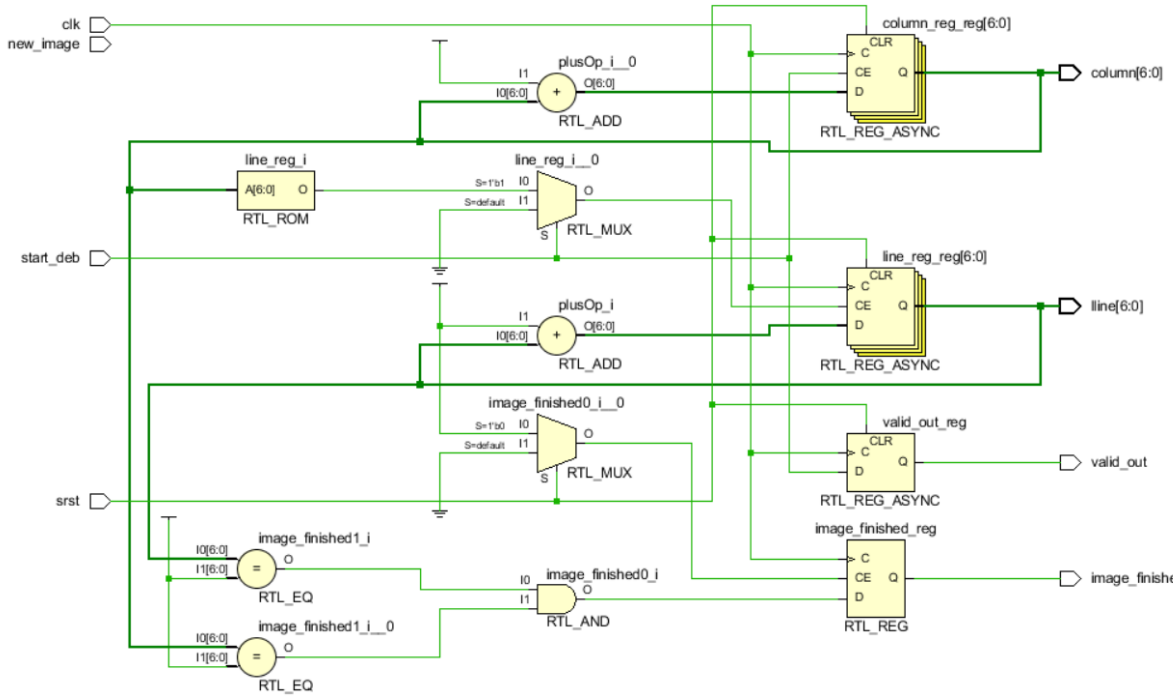
COMPUTE_UNIT

```
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 COMPUTE_UNIT is
    generic( N: integer := 4);
    port(
```

```
        clk : IN STD_LOGIC;
        srst : IN STD_LOGIC;
        s00 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        s01 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        s02 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        s10 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        s11 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        s12 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        s20 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        s21 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        s22 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        lline : IN std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0);
        column : 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 COMPUTE_UNIT;

architecture Behavioral of COMPUTE_UNIT is

    signal d4, d3, d2, d1, right, left, cc, up, down : std_logic_vector(7 downto 0);
    signal validout : std_logic;
    signal temp0 : std_logic_vector(9 downto 0);
    signal temp1 : std_logic_vector(9 downto 0);
    signal temp2 : std_logic_vector(8 downto 0);
    signal temp3 : std_logic_vector(8 downto 0);
    signal mv_ud : std_logic_vector(7 downto 0);
    signal mv_lr : std_logic_vector(7 downto 0);
    signal mv_d : std_logic_vector(7 downto 0);
    signal mv_t : std_logic_vector(7 downto 0);
    signal temp : std_logic_vector(7 downto 0) := "11111111";
    signal startup : std_logic_vector(1 downto 0) := (others => '0');

    begin

    --+--------------------+
    --|  d1  |  up  |  d2  |
    --| left |  cc  | right|
    --|  d3  | down |  d4  |
    --+--------------------+

    temp0 <= ("00" & left) + ("00" & right)  + ("00" & up) + ("00" & down);
    temp1 <= ("00" & d1) + ("00" & d2)  + ("00" & d3) + ("00" & d4);
    temp2 <= ('0' & up) + ('0' & down);
    temp3 <= ('0' & left) + ('0' & right);
    mv_t  <= temp0(9 downto 2) ;
    mv_d  <= temp1(9 downto 2) ;
    mv_ud <= temp2(8 downto 1) ;
    mv_lr <= temp3(8 downto 1) ;

    d1 <= s22 when to_integer(unsigned(lline)) /= 0 and to_integer(unsigned(column)) /= 0 else (others => '0');
    up <= s21 when to_integer(unsigned(lline)) /= 0 else (others => '0');
    d2 <= s20 when to_integer(unsigned(lline)) /= 0 and to_integer(unsigned(column)) /= N-1 else (others => '0');
    left <= s12 when  to_integer(unsigned(column)) /= 0 else (others => '0');
    cc <= s11;
    right <= s10 when to_integer(unsigned(column)) /= N-1 else (others => '0');
    d3 <= s02 when to_integer(unsigned(lline)) /= N-1 and to_integer(unsigned(column)) /= 0 else (others => '0');
    down <= s01 when to_integer(unsigned(lline)) /= N-1  else (others => '0');
    d4 <= s00 when to_integer(unsigned(lline)) /= N-1 and to_integer(unsigned(column)) /= N-1 else (others => '0');

    process(clk, srst)
    begin
        if srst = '1' then
            --image_finished <= '0';
        elsif rising_edge(clk)  then
            if lline(0) = '0' then
                if column(0) ='0' then --green
                    R <= mv_ud;
                    G <= cc;
                    B <= mv_lr;
                else                    --blue
                    R <= mv_d;
                    G <= mv_t;
                    B <= cc;
                end if;
            else
                if column(0) ='0' then  --red
                    R <= cc;
```

```
                    G <= mv_t;
                    B <= mv_d;
                else                        --green
                    R <= mv_lr;
                    G <= cc;
                    B <= mv_ud;
                end if;
            end if;
        end if;
    end process;
end Behavioral ;
```

- **CTRL – Control:** Το component αυτό ελέγχει την ορθή λειτουργία του κυκλώματος και στέλνει τα κατάλληλα σήματα στο compute unit για την εμφάνιση σωστών αποτελεσμάτων. Πιο συγκεκριμένα, μόλις λάβει το σήμα της πρώτης έγκυρης εξόδου του S2P σηματοδοτεί το πρώτο σωστό αποτέλεσμα και ξεκινά την αύξηση των στηλών και των γραμμών ανάλογα με την θέση του pixel που βρισκόμαστε. Τέλος είναι υπεύθυνο και για την σωστή εμφάνιση του image_finished όταν βρεθούμε στο τελευταίο pixel. Παρατίθενται το σχηματικό και ο κώδικας του CONTROL.



CONTROL

```
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 CONTROL is
    generic(
        N: integer  := 4
    );
    port (
        clk : IN STD_LOGIC;
        srst : IN STD_LOGIC;
        valid_in : in std_logic;
        new_image : in std_logic;
        start_deb: in STD_LOGIC;
        lline : out std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0);
        column : out std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0);
```

```
        valid_out: OUT STD_LOGIC;
        image_finished: OUT STD_LOGIC
        );
end CONTROL;

architecture Behavioral of CONTROL is

    signal column_reg : std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0)  :=  (others => '0');
    signal line_reg : std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0)  :=  (others => '0');
begin

    column <= column_reg;
    lline <= line_reg;

    process(clk, srst)
    begin
        if srst = '1' then
            line_reg <= (others => '0') ;
            column_reg <= (others => '0') ;
            valid_out <= '0';
        elsif rising_edge(clk) then
            valid_out <= '0';
            if (start_deb = '1') then
                valid_out <= '1';
                column_reg <= column_reg + 1;
                if to_integer(unsigned(column_reg)) = N-1 then
                    line_reg <= line_reg + 1;
                end if;
            end if;

            if (to_integer(unsigned(line_reg)) = N-1 and to_integer(unsigned(column_reg)) = N-1) then
                image_finished <= '1';
            else
                image_finished <= '0';
            end if;
        end if;
    end process;
end Behavioral ; -- Behavioral
```

- **DEBAYERING :** Όσον αφορά τον τελικό κώδικα, βλέπουμε την διασύνδεση των επιμέρους κυκλωμάτων, το τελικό set του valid_out (valid out από το CONTROL και valid είσοδος) ενώ γίνεται και η χρήση της τεχνικής clock gating ώστε σε περίπτωση invalid εισόδου τα κυκλώματα να «παγώνουν». Παρατίθενται και ο κώδικας και το test bench του DEBAYERING. Στο test bench υλοποιείται επίσης το διάβασμα αλλά και το γράψιμο σε αρχείο του αναμενόμενου αποτελέσματος.

DEBAYERING

```
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 := 16);
    port(
        clk :       in std_logic;
        rst_n :     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 DEBAYERING;

architecture Behavioral of DEBAYERING is

    component s2p
```

```
        generic ( N: integer );
        port (
            clk : IN STD_LOGIC;
            srst : IN STD_LOGIC;
            pixel : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            valid_pxl : IN STD_LOGIC;
            s00 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            s01 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            s02 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            s10 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            s11 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            s12 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            s20 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            s21 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            s22 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            valid_out: OUT STD_LOGIC
        );
    end component;

    component COMPUTE_UNIT
        generic ( N: integer );
        port(
            clk : IN STD_LOGIC;
            srst : IN STD_LOGIC;
            s00 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            s01 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            s02 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            s10 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            s11 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            s12 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            s20 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            s21 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            s22 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            lline : IN std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0);
            column : 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 CONTROL
        generic ( N: integer );
        port (
            clk : IN STD_LOGIC;
            srst : IN STD_LOGIC;
            valid_in : in std_logic;
            new_image : in std_logic;
            start_deb : in std_logic;
            lline : out std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0);
            column : 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 counter : std_logic_vector(2*log2(N) downto 0);
    signal clk_gate : std_logic ;
    signal ss00, ss01, ss02, ss10, ss11, ss12, ss20, ss21, ss22  : std_logic_vector(7 downto 0);
    signal liner, columnr : std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0)  :=  (others => '0');
    signal startdeb : std_logic;
    signal valid_out1 : std_logic;

    begin

    CTRL : CONTROL
        generic map ( N => N )
        port map(
                clk => clk_gate,
                srst => rst_n,
                valid_in => valid_in,
                new_image => new_image,
                start_deb => startdeb,
                lline => liner,
                column => columnr,
                valid_out => valid_out1,
                image_finished => image_finished
        );
```

```
    CU : COMPUTE_UNIT
        generic map ( N => N )
        port map(
            clk => clk_gate,
            srst => rst_n,
            s00 => ss00,
            s01 => ss01,
            s02 => ss02,
            s10 => ss10,
            s11 => ss11,
            s12 => ss12,
            s20 => ss20,
            s21 => ss21,
            s22 => ss22,
            lline => liner,
            column => columnr,
            R => R,
            G => G,
            B => B
        );

    S_2_P : S2P
        generic map ( N => N )
        port map(
            clk => clk_gate,
            srst => rst_n,
            pixel => pixel,
            valid_pxl => '1',
            s00 => ss00,
            s01 => ss01,
            s02 => ss02,
            s10 => ss10,
            s11 => ss11,
            s12 => ss12,
            s20 => ss20,
            s21 => ss21,
            s22 => ss22,
            valid_out => startdeb
        );


    valid_out <= valid_out1 and valid_in;
    clk_gate <= (clk and valid_in);

end Behavioral ;
```

DEBAYERING_tb

```
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 test_bench of DEBAYERING_tb is
    constant N : integer := 16;

    component DEBAYERING
    generic( N: integer );
    port (
        clk :       IN STD_LOGIC;
        rst_n :     IN STD_LOGIC;
        new_image : IN STD_LOGIC;
        valid_in :  IN STD_LOGIC;
        pixel :     IN STD_LOGIC_VECTOR(7 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);
        valid_out:      OUT STD_LOGIC;
        image_finished: OUT STD_LOGIC
    );
    end component;

--Input Signals
```

```
    signal clk :        STD_LOGIC := '0';
    signal rst_n:       STD_LOGIC := '0';
    signal new_image :  STD_LOGIC := '0';
    signal valid_in :   STD_LOGIC := '0';
    signal pixel :      STD_LOGIC_VECTOR(7 DOWNTO 0);
    signal ena:         STD_LOGIC := '0';
 --Output Signals
    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 valid_out: STD_LOGIC;
    signal image_finished: STD_LOGIC;
--Clock
    constant CLK_PERIOD : time := 1ns;

begin
    UUT:DEBAYERING
    generic map( N => N )
    port map(
        clk => clk,
        rst_n => rst_n,
        new_image => new_image,
        valid_in => valid_in,
        pixel => pixel,
        R => R,
        G => G,
        B => B,
        valid_out => valid_out,
        image_finished => image_finished
        );

    clk_proc: process
    begin
        clk <= '0';
        wait for CLK_PERIOD / 2;
        clk <= '1';
        wait for CLK_PERIOD / 2;
    end process;

    init : process
    begin
        wait for clk_period/2;
        rst_n <= '1';
        wait for clk_period;
        rst_n <= '0';
        ena <='1';
        wait;
    end process ; -- init

    p_read : process(clk, rst_n)

    file input_file      : text open read_mode is "C:\Users\Afroditi\Documents\8thSemester\dvlsi\Lab4\image_tb.txt";
    variable row_read         : line;
    variable pixel_read  : integer;
    variable pixel_row_counter : integer := 0;

    file output_file    : text open write_mode is "C:\Users\Afroditi\Documents\8thSemester\dvlsi\Lab4\image_deb.txt";
    variable row_write  : line;

    begin
        if(rst_n='1') then
            pixel_row_counter     := 0;
            pixel_read            := -1;
        elsif (rising_edge(clk)) then
            if (ena = '1') then
                -- read from input file in "row" variable
                if(not endfile(input_file)) then
                    if (pixel_row_counter = 0) then
                        new_image <= '1';
                    else
                        if (pixel_row_counter = N*N -1) then
                            pixel_row_counter := -1;
                        end if;
                        new_image <= '0';
                    end if;
                    pixel_row_counter := pixel_row_counter + 1;
                    readline(input_file,row_read);
                end if;
                -- read integer number from "row" variable in integer array
```

```
                read(row_read,pixel_read);
                valid_in <= '1';
                pixel <= std_logic_vector(to_unsigned(pixel_read,8));
            end if;
            if (valid_out = '1') then
                write(row_write, to_integer(unsigned(R)), left, 4);
                write(row_write, to_integer(unsigned(G)), left, 4);
                write(row_write, to_integer(unsigned(B)), left, 4);
                writeline(output_file,row_write);
            end if;
            if  (image_finished = '1') then
                write(row_write,string'("IMAGE FINISHED!"));
                writeline(output_file,row_write);
            end if;
        end if;
    end process ; -- p_read

end test_bench ; -- test_bench
```

Τέλος, επισημαίνουμε ότι στο zip με τους κώδικες των υλοποιήσεων υπάρχουν όλοι οι παραπάνω κώδικες καθώς και επιπλέον test bench για κάθε ξεχωριστό component.

Καταγραφή και Ανάλυση των πόρων του FPGA:

Στο σημείο αυτό παρουσιάζονται οι πόροι του FPGA που χρησιμοποιήθηκαν για μέγεθος εικόνας N = 64, 128. Όπως θα δούμε και παρακάτω δεν υπάρχει μεγάλη διαφορά στα resources ενώ το μεγαλύτερο utilization έχουν τα I/O ports.

- N = 64:

Resource	Utilization	Available	Utilization %
LUT	255	17600	1.45
FF	268	35200	0.76
BRAM	1.50	60	2.50
IO	37	100	37.00

Name	Used
N DEBAYERING	3
S_2_P (S2P)	3
RAM_FIFO_0 (fifo_generator_0)	1
RAM_FIFO_1 (fifo_generator_0_HD3)	1
RAM_FIFO_2 (fifo_generator_0_HD24)	1

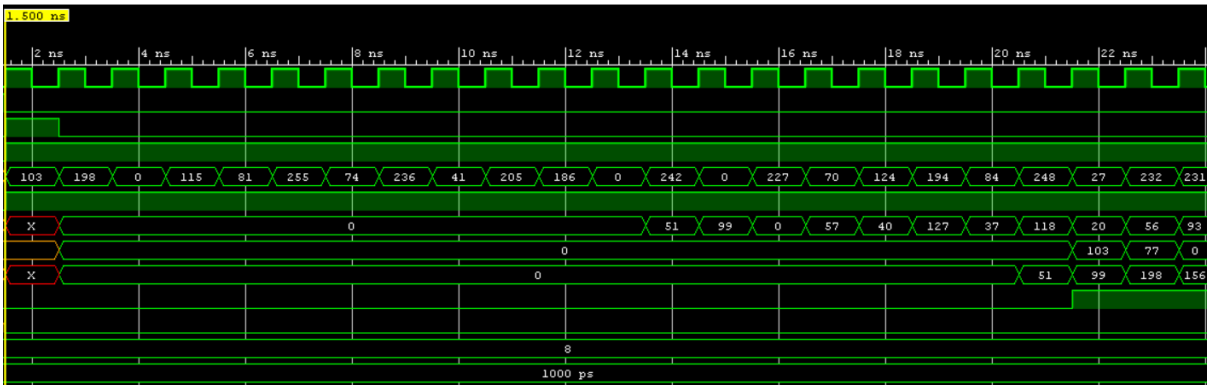
- N = 128:

Resource	Utilization	Available	Utilization %
LUT	261	17600	1.48
FF	272	35200	0.77
BRAM	1.50	60	2.50
IO	37	100	37.00

Name	Used
N DEBAYERING	3
S_2_P (S2P)	3
RAM_FIFO_0 (fifo_generator_0)	1
RAM_FIFO_1 (fifo_generator_0_HD3)	1
RAM_FIFO_2 (fifo_generator_0_HD24)	1

Υπολογισμός latency και throughput:

Η πρώτη έγκυρη έξοδος του κυκλώματος εμφανίζεται στον κύκλο 2N + 5. Αυτό προκύπτει και από την παραπάνω ανάλυση των επιμέρους components αφού θέλουμε 2N + 3 κύκλους από το S2P και στην συνέχεια άλλους 2 κύκλους μέχρι να περάσουν τα σήματα στο compute unit και να εμφανιστεί το αποτέλεσμα. Στην εικόνα που ακολουθεί φαίνονται τα παραπάνω για N = 8.



Το throughput των έγκυρων εξόδων είναι 1 εφόσον σε κάθε κύκλο έχουμε 1 σωστό αποτέλεσμα (τριπλέτα RGB).

Software Support:

Προκειμένου να ελέγξουμε την ορθότητα της υλοποίησης μας, δημιουργήσαμε μια software υποδομή σε python. Πιο συγκεκριμένα, έχουμε δύο προγράμματα:

- Create_Image.py: Δημιουργεί ένα αρχείο txt μιας εικόνας N x N με τυχαίες τιμές για το κάθε pixel στο διάστημα 0 – 255 σε μορφή ενός pixel ανά γραμμή.
- Debayering.py: Παίρνει ως είσοδο ένα αρχείο txt μίας εικόνας N x N σε μορφή ενός pixel ανά γραμμή και παράγει ένα αρχείο txt με τις τριπλέτες των RGB που προκύπτουν από την διαδικασία του debayering.