Εθνικό Μετσόβιο Πολυτεχνείο, ΣΗΜΜΥ
Ψηφιακά Συστήματα VLSI

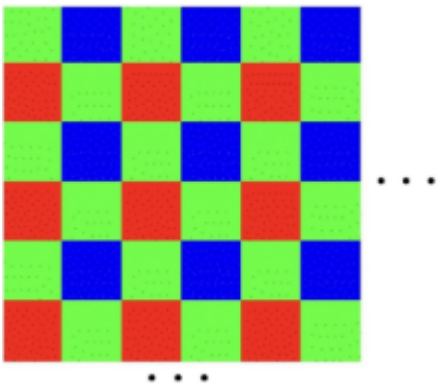
6η Εργαστηριακή Άσκηση “Υλοποίηση Debayering Φίλτρου με AXI διεπαφή σε Zynq SoC FPGA”

Ομάδα 19
Παναγιώτης Μπέλσης, ΑΜ: 03120874
Θεοδώρα Εξάρχου, ΑΜ : 03120865

Μέρος Α

Ζητούμενο Εργαστηριακής Άσκησης

Στα πλαίσια της παρούσας Εργαστηριακής Άσκησης, καλείστε να υλοποιήσετε 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 Σχήμα 2.



Χωρίζουμε το κώδικα μας στα εξής μέρη
-**Debayering Filter**
-Serial to Parallel
-Debayering box

Debayering Filter →Το Debayering φίλτρο λαμβάνει ένα pixel ανα κύκλο ρολογιου (1p/CC) με raster scan ordering και ότι η επεξεργασία (υπολογισμός μέσων όρων) πραγματοποιείται σε 3x3 γειτονιές

● clk**:** το σήμα ρολογιού, με βάση το οποίο συγχρονίζεται όλο το κύκλωμα.

● rst_n**:** το ασύγχρονο σήμα μηδενισμού, με βάση το οποίο μηδενίζονται οι καταχωρητές και οι έξοδοι του κυκλώματος. Ενεργοποιείται στο λογικό '0'. Κατά τη λειτουργία του κυκλώματος, είναι απενεργοποιημένο (βρίσκεται μόνιμα στο λογικό '1').

● pixel[7:0]**:** σήμα εισόδου των 8 bits, το οποίο τροφοδοτεί το κύκλωμα με τα 8-bit pixels της Bayer εικόνας.

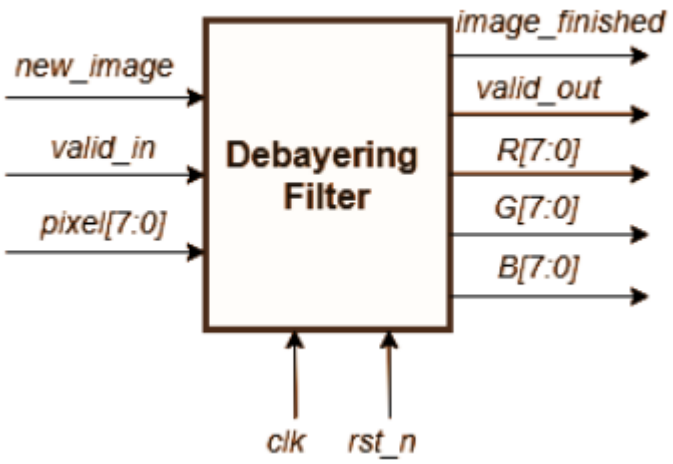
● valid_in: σήμα εισόδου, το οποίο ενεργοποιείται (στο λογικό '1') για να υποδείξει ότι η τιμή του σήματος pixel είναι έγκυρη. Ενεργοποιείται για αριθμό κύκλων ρολογιού ίσο με το μέγεθος της εικόνας.

● new_image**:** σήμα εισόδου, το οποίο ενεργοποιείται (στο λογικό '1') για 1 κύκλο ρολογιού ταυτόχρονα με το πρώτο valid_in σήμα, ώστε να υποδείξει ότι θα αρχίσει η επεξεργασίας μιας νέας εικόνας.

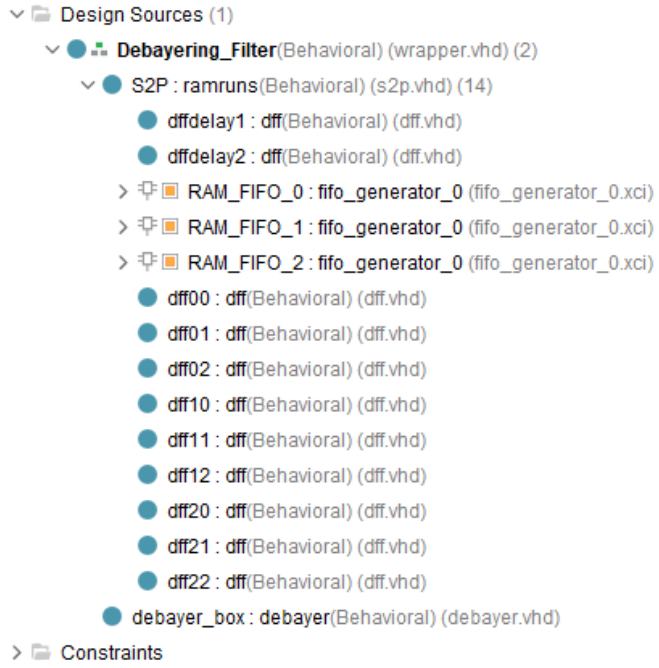
● R[7:0], G[7:0], B[7:0]: σήματα εξόδου των 8 bits, τα οποία υποδεικνύουν τις χρωματικές συνιστώσες. Για κάθε pixel, εμφανίζονται ταυτόχρονα στην έξοδο και οι τρεις χρωματικές συνιστώσες. Επίσης, με βάση την λειτουργία του Debayering φιλτρου, για κάθε pixel, κάποιο από τα R, G, και B θα έχει την τιμή του σήματος pixel, ενώ τα άλλα δύο θα υπολογίζονται με τους μέσους όρους των γειτονικών pixels.

● valid_out**:** σήμα εξόδου, το οποίο ενεργοποιείται (στο λογικό '1') για να υποδείξει ότι οι τιμές των σημάτων R, G, B είναι έγκυρες. Ενεργοποιείται για αριθμό κύκλων ρολογιού ίσο με το μέγεθος της εικόνας.

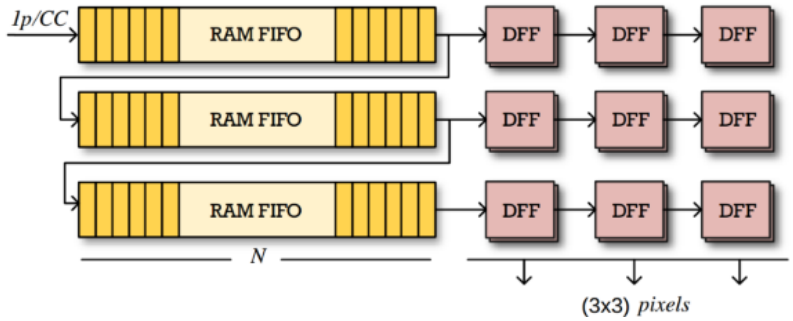
● image_finished**:** σήμα εξόδου, το οποίο ενεργοποιείται (στο λογικό '1') για 1 κύκλο ρολογιού ταυτόχρονα με το τελευταίο valid_out σήμα, ώστε να υποδείξει ότι η επεξεργασία της εικόνας τελείωσε.



Ιεράρχηση των αρχείων μας για το design:



Serial to Parallel →Μετατροπέας σειριακού-σε-παράλληλο (serial-to-parallel converter). Το κύκλωμα του μετατροπέα περιλαμβάνει προσωρινή μνήμη (buffers) και μία συστοιχία από καταχωρητές (registers), ώστε να αποθηκεύει τα pixels εισόδου και να τροφοδοτεί την είσοδο του κυκλώματος υπολογισμού των μέσων όρων με όλα τα pixels της 3x3 γειτονιάς παράλληλα.



RamRuns (Serial-to-parallel)

Η “ramruns” σχεδιάζει ένα block που φροντίζει για την παραλληλοποίηση των σειριακών pixels με την χρήση 3 ram fifo και 9 βασικών DFF με βάση την σχεδίαση που προτάθηκε στην εκφώνηση. Επιπλέον χρησιμοποιήθηκαν άλλα 2 DFF για να καθυστερήσουμε την είσοδο των δεδομένων στην πρώτη Ram, έτσι ώστε να πετύχουμε καλύτερο συγχρονισμό των Ram με το ολικό κύκλωμα.

Τρέχοντας στο χαρτί το πώς γεμίζουν οι Rams και βλέποντας το πότε βγάζουν τα πρώτα δεδομένα, είδαμε ότι το πρώτο αποτέλεσμα βγαίνει στον 2N+2 κύκλο, και έπειτα σε κάθε κύκλο ρολογιού είχαμε νέο αποτέλεσμα.

Αρχικά όλα τα σήματα write,read enable είναι αρχικοποιημένα στο 0.

Απο 1 → N cc

όταν valid_in=1
write_enable0=1
read_enable0=0

Από N +1 → 2N cc

write_enable0=1
read_enable0=1
write_enable1=1

Από 2N+1 → 3N cc

write_enable0=1
read_enable0=1
write_enable1=1
read_enable1=1
write_enable2=1

Στον 2N+2 cc
έχουμε το πρώτο valid_out=1 αποτέλεσμα.

Από 3N+1 → 2N+1 + N*N cc

write_enable0=1
read_enable0=1
write_enable1=1
read_enable1=1
write_enable2=1
read_enable2=1

Οι παραπάνω θεωρητικές τιμές δεν αντικατοπτρίζουν στο 100% την συμπεριφορά των ram στο simulation καθώς στό χαρτί θεωρούμε ότι τα σήματα write, read enable αποκτούν αμέσως την τιμή που τους δίνουμε, κάτι που στην πραγματικότητα έχει καθυστέρηση ενός κύκλου.
Αφότου τρέξαμε ένα πρόγραμμα μόνο με τις rams για να καταλάβουμε το πώς γεμίζουν και το πώς διαχειρίζονται τα σήματα που τους δίνουμε καταλήξαμε στην παρακάτω υλοποίηση του κώδικα που τρέχει λίγο διαφορετικά σε σχέση με την θεωρητική υλοποίηση.

```
-------------------------------------------------------------------------------------------------------------
else if( valid_in = '1' ) then

           wr_en0 <= '1';
           counter_begin <= counter_begin + 1 ;

            if (to_integer(unsigned(counter_begin)) >= N-2) then
                    rd_en0 <= '1';
                    wr_en1 <='1';
            end if;
            if(to_integer(unsigned(counter_begin)) >=  2*N-3) then
                    rd_en1 <='1';
            end if;
            if(to_integer(unsigned(counter_begin)) >=  2*N-2) then
                    wr_en2 <='1';
            end if;
            if(to_integer(unsigned(counter_begin)) >= 3*N-3) then
                       rd_en2 <='1';
            end if;
-------------------------------------------------------------------------
```

Από κάτω παραθέτουμε τον ολοκληρωμένο κώδικα της “ramruns”

```
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 ramruns is
    generic ( N: integer := 32 );
    port (
        clk : IN STD_LOGIC;
        rst : IN STD_LOGIC;
        pixel : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        col,row : OUT std_logic_vector(integer(ceil(log2(real(N))))-1
downto 0)  :=  (others => '0') ;
         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_in,new_image : IN STD_LOGIC; --signals when it is valid to
start the multiplication process
         image_finished: OUT STD_LOGIC := '0' ;
        valid_out: OUT STD_LOGIC;

        out_row , out_col : OUT
std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0)  :=  (others
=> '0')

    );
end ramruns;

architecture Behavioral of ramruns 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
  );
END COMPONENT;

component dff is
    port(
      d : in  std_logic_vector(7 downto 0);
      q : out  std_logic_vector(7 downto 0);
      clk : in std_logic;
      rst : in std_logic
     );
end component;


signal ram_out_2, ram_out_1, ram_out_0 : 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  wr_en0: std_logic := '0';
signal  wr_en1: std_logic := '0';
signal  wr_en2: std_logic := '0';
signal t1,t2,t3,t4,t5,t6,t7,t8,t9 : STD_LOGIC_VECTOR(7 DOWNTO 0)  :=
(others => '0');

signal full0, full1, full2, empty0, empty1, empty2: std_logic;
signal pixel_delay1 ,pixel_delay2  : STD_LOGIC_VECTOR(7 DOWNTO 0)  :=
(others => '0');
signal col_temp,row_temp :
std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0) :=  (others =>
'0');
signal  col_signal : std_logic_vector(integer(ceil(log2(real(N))))-1
downto 0)  :=  (others => '0') ;


signal counter_begin :  std_logic_vector(integer(ceil(log2(real(N*N))))
downto 0) :=  (others => '0') ;

begin

dffdelay1 : dff port map(d=>pixel, clk=>clk, rst=>rst, q=>pixel_delay1
);
```

```
dffdelay2 : dff port map(d=>pixel_delay1, clk=>clk, rst=>rst,
q=>pixel_delay2 );


RAM_FIFO_0 : fifo_generator_0
    PORT MAP (
        clk => clk,
        srst => rst,
        din => pixel_delay2,
        wr_en => wr_en0,
        rd_en => rd_en0,
        dout => ram_out_0,
        full => full0,
        empty => empty0
    );

RAM_FIFO_1 : fifo_generator_0
PORT MAP (
    clk => clk,
    srst => rst,
    din => ram_out_0,
    wr_en => wr_en1,
    rd_en => rd_en1,
    dout => ram_out_1,
    full => full1,
    empty => empty1
);

RAM_FIFO_2 : fifo_generator_0
PORT MAP (
    clk => clk,
    srst => rst,
    din => ram_out_1,
    wr_en => wr_en2,
    rd_en => rd_en2,
    dout => ram_out_2,
    full => full2,
    empty => empty2
);

 dff00 : dff port map(d=>ram_out_0, clk=>clk, rst=>rst, q=>t1);
 s00 <=t1;
dff01 : dff port map(d=>t1, clk=>clk, rst=>rst, q=>t2);
 s01 <=t2;
dff02 : dff port map(d=>t2, clk=>clk, rst=>rst, q=>t3);
 s02 <=t3;
```

```
dff10 : dff port map(d=>ram_out_1, clk=>clk, rst=>rst, q=>t4);
 s10 <=t4;
dff11 : dff port map(d=>t4, clk=>clk, rst=>rst, q=>t5);
 s11 <=t5;
dff12 : dff port map(d=>t5, clk=>clk, rst=>rst, q=>t6);
 s12 <=t6;
dff20 : dff port map(d=>ram_out_2, clk=>clk, rst=>rst, q=>t7);
 s20 <=t7;
dff21 : dff port map(d=>t7,clk=>clk, rst=>rst, q=>t8);
 s21 <=t8;
dff22 : dff port map(d=>t8, clk=>clk, rst=>rst, q=>t9);
 s22 <=t9;



 process(clk,rst)
 --constant N : integer :=32;
 begin

   if( rst = '1') then

                  wr_en0 <= '0';
                  rd_en0 <= '0';
                  wr_en1 <= '0';
                  rd_en1 <= '0';
                  wr_en2 <= '0';
                  rd_en2 <= '0';
                  counter_begin <=(others => '0');
                  row<= (others => '0');
                  col <=  (others => '0');


     --RESET
    elsif rising_edge(clk) then
           valid_out <='0';
           image_finished <='0';


       if (valid_in = '1' AND new_image ='1' ) then

           counter_begin <= (others => '0');
           row_temp<= (others => '0') ;
           col_temp <= (others => '0') ;
           image_finished <='0';
```

```
      else if( valid_in = '1' ) then

          wr_en0 <= '1';
          counter_begin <= counter_begin + 1 ;

           if (to_integer(unsigned(counter_begin)) >= N-2) then
                   rd_en0 <= '1';
                   wr_en1 <='1';
           end if;
           if(to_integer(unsigned(counter_begin)) >=  2*N-3) then
                   rd_en1 <='1';
           end if;
           if(to_integer(unsigned(counter_begin)) >=  2*N-2) then
                   wr_en2 <='1';
           end if;
           if(to_integer(unsigned(counter_begin)) >= 3*N-3) then
                      rd_en2 <='1';
           end if;



    if( counter_begin > 2*N +1) then

              valid_out <='1';
              col_signal <=col_temp;

              if(row_temp = 0 and col_temp = 0) then
                 col <=  col_temp + 1 ;
              else
                 col <=  col_temp+1;
              end if;

              row <=  row_temp;
              out_row <= row_temp;
              out_col <= col_temp;
              col_temp <= col_temp+1;

              if (to_integer(unsigned(col_signal)) = N-2) then    --allazw
grammh
                    row_temp <= row_temp + 1 ;
                    col_temp <= (others => '0') ;
              end if;
              if (to_integer(unsigned(row_temp))= N-1 and
to_integer(unsigned(col_temp)) = N-1) then
                     image_finished <= '1';
              end if;
```

```
        end if;
        end if;
    end if;
  end if;

  end process;
end Behavioral;
```

DFF

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

--DFF for 8 bit pixel

entity dff is
    Port ( d : in STD_LOGIC_VECTOR (7 downto 0);
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           q : out STD_LOGIC_VECTOR (7 downto 0));
end dff;

architecture Behavioral of dff is

begin
    process(clk, rst) begin
        if rst='1' then
            Q<="00000000";
        elsif rising_edge(clk) then
            Q<=D;
        end if;
    end process;
end Behavioral;
```

Debayering box →Δέχεται το 3x3 παράθυρο, πρώτα ελέγχεται αν κάποιο pixel πρέπει να θεωρηθεί 0 βάσει της θέσης που έχει το pixel που επεξεργαζόμαστε τη δεδομένη στιγμή, αντιστρέφουμε το πίνακα που δημιουργήσαμε στο ramruns(Serial to Parallel) έτσι ώστε να έχει τη σωστή μορφή.

Ύστερα υπολογίζει τους 4 δυνατούς μέσους (γραμμή, στήλη, σταυρός, γωνιακά) και τέλος βάσει του state τα RGB παίρνουν τιμές είτε από τους παραπάνω υπολογισμούς είτε απευθείας από το μεσαίο pixel του παραθύρου.

```
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 debayer is
  generic( N: integer := 32);
  Port (
  clk : IN STD_LOGIC;
  rst: IN STD_LOGIC;
  s2p_00 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_01 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_02 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_10 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_11 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_12 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_20 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_21 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_22 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  line_img : IN std_logic_vector(integer(ceil(log2(real(N))))-1 downto
0);
  column_img : 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 debayer;

architecture Behavioral of debayer is
```

```
signal box_00, box_01, box_02, box_10, box_11, box_12, box_20, box_21,
box_22 : std_logic_vector(7 downto 0);
signal add_outer, add_cross,add_row,add_col : std_logic_vector(7 downto
0);
signal add_row_bef, add_col_bef: std_logic_vector(8 downto 0);
signal add_outer_bef, add_cross_bef: std_logic_vector(9 downto 0);
signal unsigned_x1,unsigned_x2,unsigned_x3,unsigned_x4 : unsigned(7
downto 0);
signal sum_unsigned : unsigned(8 downto 0);
signal divided_unsigned : unsigned(8 downto 0);
signal sum_unsigned_2 : unsigned(9 downto 0);
--constant N : integer := 32;
begin

box_00 <= s2p_22 when to_integer(unsigned(line_img)) /= 0 and
to_integer(unsigned(column_img)) /= 0 else (others => '0');
box_01 <= s2p_21 when to_integer(unsigned(line_img)) /= 0  else (others
=> '0');
box_02 <= s2p_20 when to_integer(unsigned(line_img)) /= 0 and
to_integer(unsigned(column_img)) /=  N-1 else (others => '0'); -
box_10 <= s2p_12 when to_integer(unsigned(column_img)) /= 0 else (others
=> '0');
box_11 <= s2p_11;
box_12 <= s2p_10 when to_integer(unsigned(column_img)) /= N-1 else
(others => '0');
box_20 <= s2p_02 when to_integer(unsigned(line_img)) /= N-1 and
to_integer(unsigned(column_img)) /= 0 else (others => '0');
box_21 <= s2p_01 when to_integer(unsigned(line_img)) /= N-1  else
(others => '0');
box_22 <= s2p_00 when to_integer(unsigned(line_img)) /= N-1 and
to_integer(unsigned(column_img)) /= N-1 else (others => '0');



add_row_bef <= ('0' & box_10) + ('0' & box_12); --kanw to add_row_bef 9
bits
add_row <= add_row_bef(8 downto 1); -- me leftshift kanw /2

add_col_bef <= ('0' & box_01) + ('0' & box_21);
add_col <= add_col_bef(8 downto 1);

add_cross_bef <= ("00" & box_01) + ("00" & box_10) + ("00" & box_12) +
("00" & box_21);
add_cross <= add_cross_bef(9 downto 2); -- 2 leftshift ara kanw /4

add_outer_bef <= ("00" & box_00) + ("00" & box_02) + ("00" & box_20) +
```

```
("00" & box_22);
add_outer <= add_outer_bef(9 downto 2);


 process(clk,rst)
    begin
        if rising_edge(clk)  then
            if line_img(0) = '0' and column_img(0) = '0' then  -- (ii)
                    R <= add_col;
                    G <= box_11;
                    B <= add_row;
            elsif line_img(0) = '0' and column_img(0) = '1' then --(iv)
                    R <= add_outer;
                    G <= add_cross;
                    B <= box_11;
           elsif line_img(0) = '1' and column_img(0) = '0' then  --(iii)
                    R <= box_11;
                    G <= add_cross;
                    B <= add_outer;
           elsif line_img(0) = '1' and column_img(0) = '1' then   --(i)
                   R <= add_row;
                   G <= box_11;
                   B <= add_col;
           end if;
           end if;
      end process;
end Behavioral;
```

Debayering Filter

Ο κώδικας του “Debayering_Filter” αρχείου είναι το συνολικό wrapper που κάνει το mapping των κοινών σημάτων του “ramruns” με του “debayer” .

```
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_Filter is
  generic( N: integer :=32 );

  Port (
  clk : IN STD_LOGIC;
  rst : IN STD_LOGIC;
  pixel : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  valid_in,new_image : IN STD_LOGIC; --signals when it is valid to start
the multiplication process
  R : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) ;
  G : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) ;
  B : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) ;
  --outputs :out STD_LOGIC_VECTOR(31 DOWNTO 0):=  (others => '0');
  valid_out: OUT STD_LOGIC;
  image_finished: OUT STD_LOGIC;
  out_row_all , out_col_all : OUT
std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0)  :=  (others
=> '0')

  ); -----------fdsfs
end Debayering_Filter;

architecture Behavioral of Debayering_Filter is

component ramruns is
    generic ( N: integer := 32 );
    port (
        clk : IN STD_LOGIC;
        rst : IN STD_LOGIC;
        pixel : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        valid_in,new_image : IN STD_LOGIC; --signals when it is valid to
start the multiplication process
        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);
        col,row :  OUT std_logic_vector(integer(ceil(log2(real(N))))-1
downto 0)  :=  (others => '0') ;
        valid_out: OUT STD_LOGIC;
        image_finished: OUT STD_LOGIC;
         out_row , out_col : OUT
std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0)  :=  (others
=> '0')

    );
end component;

component debayer is
  generic ( N: integer := 32 );
  Port (
  clk : IN STD_LOGIC;
  rst: IN STD_LOGIC;
  s2p_00 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_01 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_02 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_10 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_11 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_12 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_20 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_21 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  s2p_22 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  line_img : IN std_logic_vector(integer(ceil(log2(real(N))))-1 downto
0);
  column_img : 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;

--constant N : integer := 32;
signal clk_signal, rst_signal: STD_LOGIC;
signal s00_signal, s01_signal, s02_signal, s10_signal, s11_signal,
s12_signal: std_logic_vector(7 downto 0) :=  (others => '0');
signal s20_signal, s21_signal, s22_signal: std_logic_vector(7 downto 0)
:=  (others => '0');
```

```
signal col_signal, row_signal:
std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0);
 signal out_col_signal, out_row_signal:
std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0);
--signal Rsig :  STD_LOGIC_VECTOR(7 DOWNTO 0) :=  (others => '0');
--signal Gsig :  STD_LOGIC_VECTOR(7 DOWNTO 0) :=  (others => '0');
--signal Bsig :  STD_LOGIC_VECTOR(7 DOWNTO 0) :=  (others => '0');

begin

 S2P : ramruns
    generic map ( N => N )
    port map(
    clk => clk,
    rst => rst,
    pixel => pixel,
    valid_in => valid_in,
    new_image => new_image,
    s00 => s00_signal,
    s01 => s01_signal,
    s02 => s02_signal,
    s10 => s10_signal,
    s11 => s11_signal,
    s12 => s12_signal,
    s20 => s20_signal,
    s21 => s21_signal,
    s22 => s22_signal,
    valid_out => valid_out,
   image_finished =>  image_finished,
    row => row_signal,
    col => col_signal,
    out_row  => out_row_all,
    out_col  => out_col_all

    );


 debayer_box : debayer
    generic map ( N => N )
    port map(
        clk => clk,
        rst => rst,
        s2p_00 => s00_signal,
        s2p_01 => s01_signal,
        s2p_02 => s02_signal,
        s2p_10 => s10_signal,
```

```
        s2p_11 => s11_signal,
        s2p_12 => s12_signal,
        s2p_20 => s20_signal,
        s2p_21 => s21_signal,
        s2p_22 => s22_signal,
        line_img => row_signal,
        column_img => col_signal,
        R => R,
        G => G,
        B => B);

end ;
```

Testbench

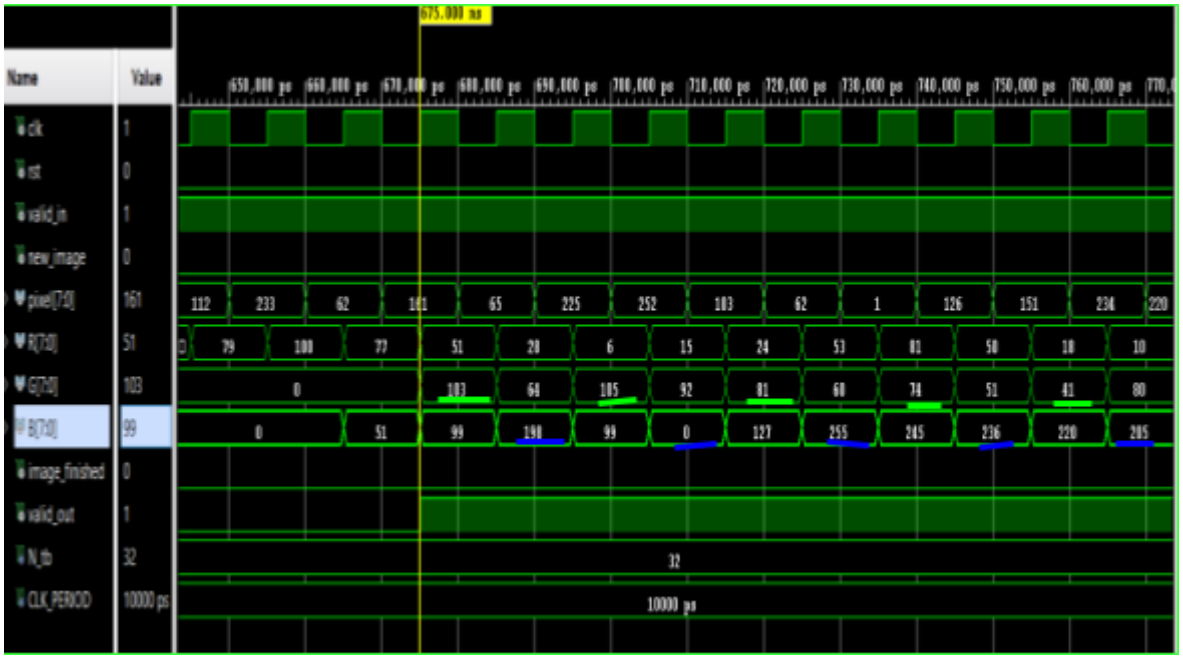
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.math_real.all;
use STD.TEXTIO.ALL;

entity Debayering_Filter_tb is
end Debayering_Filter_tb;

architecture test_bench of Debayering_Filter_tb is
 constant N_tb : integer := 32;

 component Debayering_Filter
 Port (
  clk : IN STD_LOGIC;
  rst : IN STD_LOGIC;
  pixel : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
   valid_in,new_image : IN STD_LOGIC; --signals when it is valid to start
the multiplication process
  R : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) ;
  G : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) ;
  B : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) ;
  --outputs :out STD_LOGIC_VECTOR(31 DOWNTO 0):=  (others => '0');
  valid_out: OUT STD_LOGIC;
  image_finished: OUT STD_LOGIC;
               out_row_all        ,        out_col_all        :        OUT
std_logic_vector(integer(ceil(log2(real(   N_tb))))-1   downto   0)     :=
(others => '0')
  );
 end component;

 -- Input Signals
 signal clk : std_logic := '0';
 signal rst: std_logic := '0';
 signal valid_in : std_logic := '1';
 signal new_image: std_logic := '0';
 signal pixel : STD_LOGIC_VECTOR(7 DOWNTO 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 image_finished: STD_LOGIC;
 signal valid_out : std_logic := '0';
```

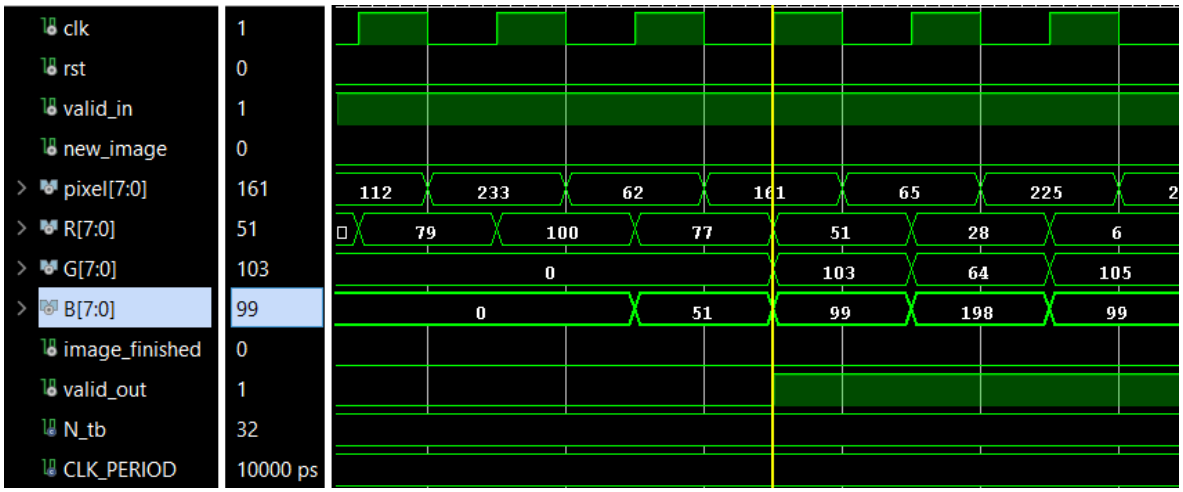
```
            signal            out_col_signal,            out_row_signal:
std_logic_vector(integer(ceil(log2(real(N_tb))))-1 downto 0);
 -- Clock
 constant CLK_PERIOD : time := 10ns;

 -- File I/O
     file      input_file      :      text      open      read_mode      is
"C:\Users\basok\Desktop\input.txt";
     file      output_file      :      text      open      write_mode      is
"C:\Users\basok\Desktop\output.txt";

begin
 wrapper_instance: Debayering_Filter
 port map(
  clk => clk,
  rst => rst,
  pixel => pixel,
  valid_in => valid_in,
  new_image => new_image,
  R => R,
  B => B,
  G => G,
  image_finished => image_finished,
  valid_out => valid_out,
   out_row_all  =>out_row_signal ,
   out_col_all =>out_col_signal

 );

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

 CHANGE_RST_validin: process
 begin
  rst <= '0';
  valid_in <= '1';
  wait for N_tb * N_tb * CLK_PERIOD;
 end process;

 CHANGE_new_image: process
 begin
```

```
 new_image <= '1';
 wait for CLK_PERIOD;
 new_image <= '0';
 wait for ((2*N_tb +3+ (N_tb * N_tb ))* CLK_PERIOD) - CLK_PERIOD;
end process;

-- Read pixel values from input file
READ_PIXELS: process
 variable input_line : line;
 variable input_pixel : integer;
 variable end_of_file : boolean := false;
begin
 while not end_of_file loop
  if endfile(input_file) then
   end_of_file := true;
  else
   readline(input_file, input_line);
   read(input_line, input_pixel);
   pixel <= std_logic_vector(to_unsigned(input_pixel, 8));
   wait for CLK_PERIOD;
  end if;
 end loop;
 wait;
end process;

-- Write RGB values to output file
WRITE_OUTPUTS: process
 variable output_line : line;
begin
 while true loop
  wait until clk'event and clk = '1';
  if valid_out = '1' then
   write(output_line, integer'image(to_integer(unsigned(R))));
   write(output_line, string'(" "));
   write(output_line, integer'image(to_integer(unsigned(G))));
   write(output_line, string'(" "));
   write(output_line, integer'image(to_integer(unsigned(B))));
   writeline(output_file, output_line);
  end if;
 end loop;
end process;

end test_bench;
```

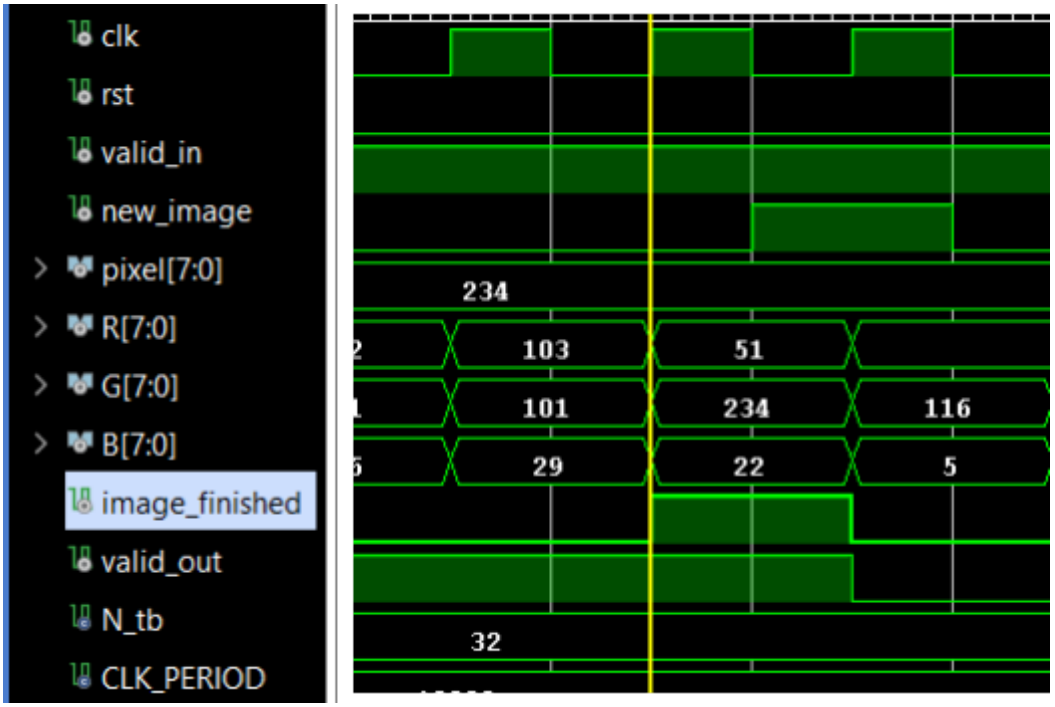
Τρέξαμε το φίλτρο με N=32, αλλά έχουμε διασφαλίσει ότι είναι μεταβλητό. Σε περίπτωση που θέλουμε να το αλλάξουμε, αρκεί να εισάγουμε τον επιθυμητό αριθμό.
generic(N: integer := 32);

Στο testbench βαζουμε τα δεδομενα απο input.txt και τα αποτελεσματα output.txt
Ελέγξαμε εαν δουλευει ο κωδικας μας με τα δεδομενα που δόθηκαν στο Helios
TEST INPUT DATA



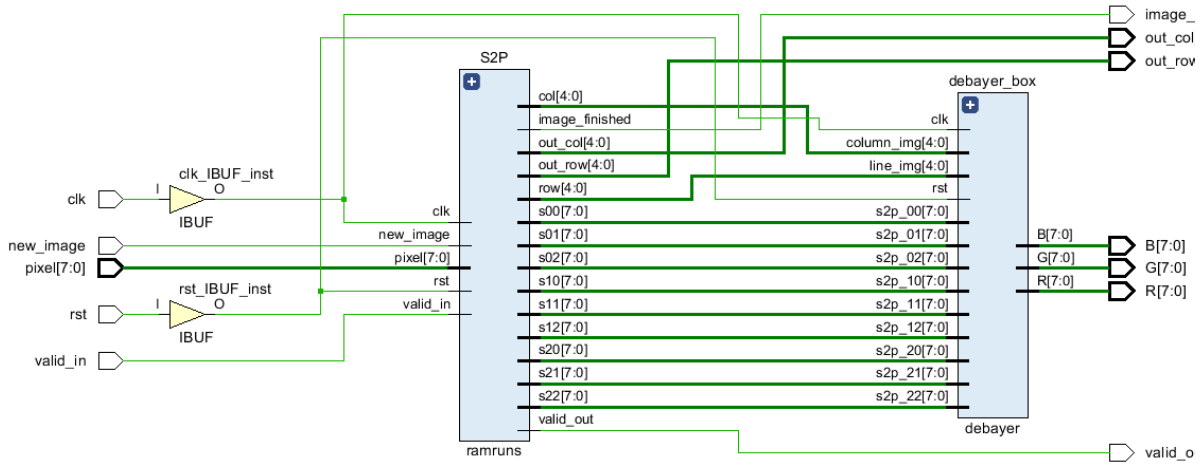


Στην εικόνα παραθέτουμε το simulation για το αρχικο pixel (103) που επεξεργαζόμαστε και βλέπουμε ότι το valid_out ενεργοποιείται (στο λογικό ‘1’) ώστε να υποδείξει ότι η επεξεργασία της εικόνας ξεκινησε.



Στην εικόνα παραθέτουμε το simulation για το τελευταίο pixel (234)που επεξεργαζόμαστε και βλέπουμε ότι το image_finished ενεργοποιείται (στο λογικό ‘1’) για 1 κύκλο ρολογιού ταυτόχρονα με το τελευταίο valid_out σήμα, ώστε να υποδείξει ότι η επεξεργασία της εικόνας τελείωσε.

Block diagram



Υπολογισμός latency & throughput

Το latency Το latency είναι ο αριθμός των κύκλων που απαιτούνται από τη στιγμή που τα δεδομένα εισάγονται στο σύστημα μέχρι να εμφανιστεί μια έγκυρη έξοδος. Εχουμε

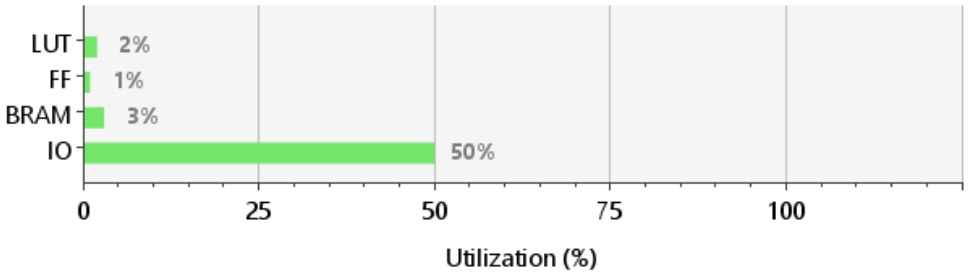
$Tlatency = Tenable + Tcenter = (N - 2) + (N + 4) + 1 \Rightarrow \mathbf{Tlatency = 2N + 3\ clocks}$

throughput = 1 μιας και λαμβάνουμε ένα νέο και έγκυρο συνδυασμό R,G,B σε κάθε κύκλο ρολογιού έπειτα από το αρχικό latency. Ακόμη, το σύστημα είναι συνεχούς διοχέτευσης, δηλαδή μπορεί να έρθει νέα εικόνα κατά τη διάρκεια της επεξεργασίας της προηγούμενης και θα γίνεται κανονικά η είσοδος των δεδομένων της στις fifo.

Καταγραφή και ανάλυση πόρων:

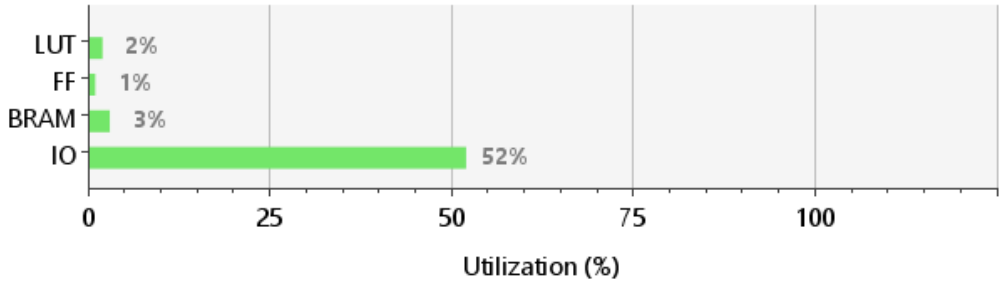
• Για N=64:

Resource	Utilization	Available	Utilization %
LUT	286	17600	1.63
FF	300	35200	0.85
BRAM	1.50	60	2.50
IO	50	100	50.00



• Για N=128:

Resource	Utilization	Available	Utilization %
LUT	301	17600	1.71
FF	308	35200	0.88
BRAM	1.50	60	2.50
IO	52	100	52.00



Δεν παρατηρείται κάποια διαφοροποίηση στους πόρους που χρησιμοποιούνται όταν αλλάζουμε την τιμή του N, καθώς τα FIFO που χρησιμοποιήθηκαν για τη μετατροπή του σειριακού σε παράλληλο έχουν μέγεθος 1024, το οποίο καλύπτει όλες τις περιπτώσεις χωρίς να προκαλείται υπερχείλιση. Επιπλέον, οι μετρητές (counters) που χρησιμοποιούνται για τον συγχρονισμό των σημάτων wr_enable και rd_enable των δύο τελευταίων FIFO παραμένουν ίδιοι, απλώς αλλάζει η μέγιστη τιμή που μπορούν να πάρουν.