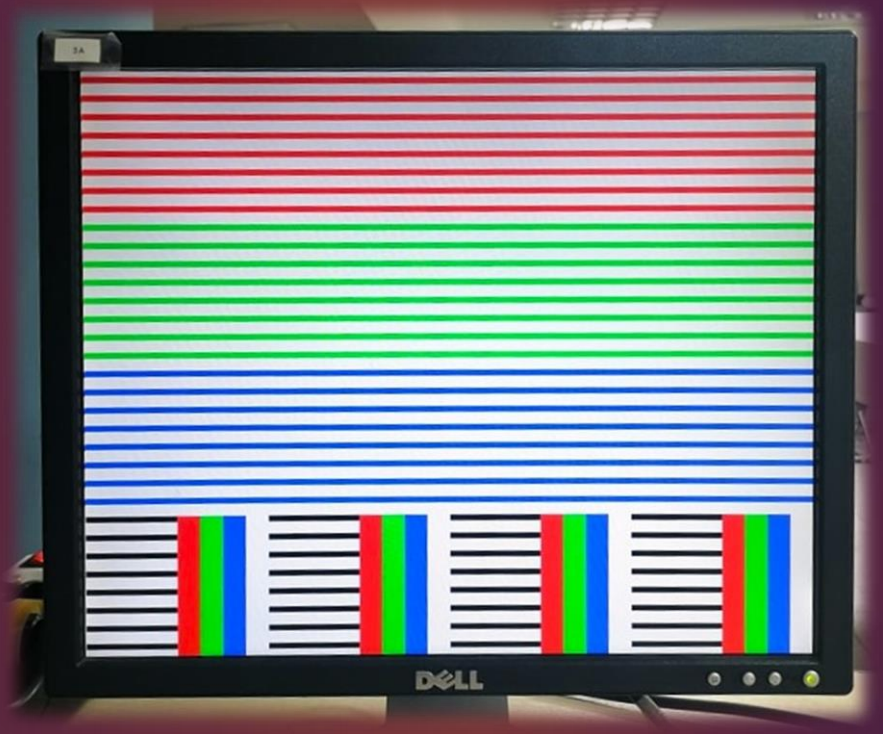
12/12/2022



3η Εργαστηριακή Εργασία
ΥΛΟΠΟΙΗΣΗ ΕΛΕΓΚΤΗ VGA

ΑΕΜ: 3218
ΟΝ/ΜΟ: ΓΑΡΥΦΑΛΙΑ ΓΕΩΡΓΙΤΖΙΚΗ

ΜΑΘΗΜΑ: ΕΡΓΑΣΤΗΡΙΟ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ

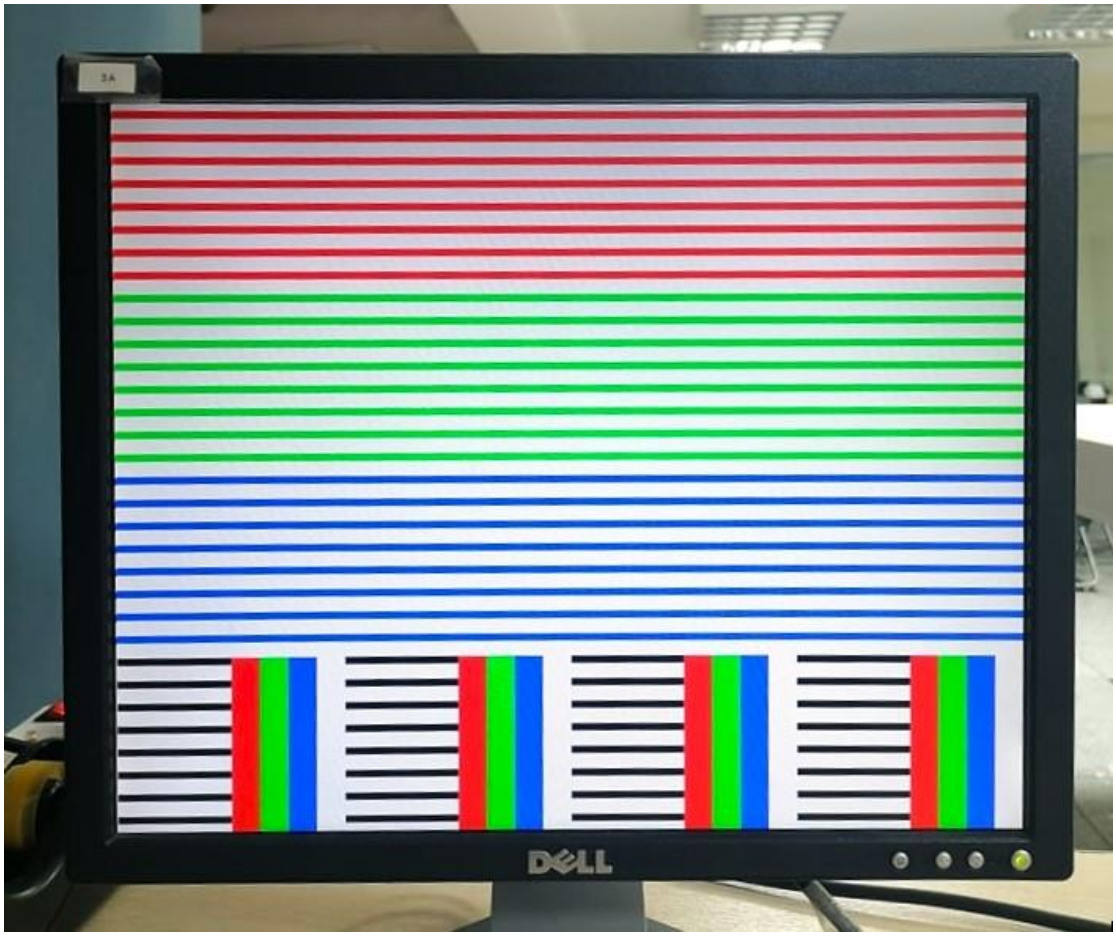
Η εργασία ‘ΥΛΟΠΟΙΗΣΗ ΕΛΕΓΚΤΗ VGA’ διαμορφώνεται σε τρία μέρη τα οποία είναι:

- ΜΕΡΟΣ Α: Υλοποίηση VRAM
- ΜΕΡΟΣ Β: Υλοποίηση HSYNC και Οριζόντιου Μετρητή Pixel
- ΜΕΡΟΣ Γ: Υλοποίηση VSYNC και Κατακόρυφου Μετρητή Pixel — Ολοκλήρωση του Ελεγκτή/Οδηγού VGA

Μέσα από τα παραπάνω, θα δούμε βήμα-βήμα την υλοποίηση, επαλήθευση και το τελικό αποτέλεσμα του Ελεγκτή VGA.

ΕΙΣΑΓΩΓΗ

Στόχος της 3ης εργαστηριακής εργασίας είναι η υλοποίηση ενός Ελεγκτή/Οδηγού θύρας οθόνης VGA (Video Graphics Array), έτσι ώστε να επιτύχουμε την οδήγηση μιας συμβατικής οθόνης και την εμφάνιση εικόνας σε αυτήν. Για τον σκοπό αυτό, χρησιμοποιήθηκε μέρος της εσωτερικής μνήμης RAM της FPGA και ορίστηκε σαν Μνήμη Εικόνας/Όρασης - Video RAM (VRAM) του οδηγού, έτσι ώστε οδηγώντας την VGA συνεχόμενα στην οθόνη, να εμφανίζεται η εικόνα που επιλέξαμε. Η ανάλυση της εικόνας είναι 640 × 480 pixel και ο ρυθμός ανανέωσης αυτής είναι 60 Hz. Οι στόχοι της εργασίας υλοποιήθηκαν επιτυχώς και αποτέλεσμα είναι η εικόνα που φαίνεται παρακάτω:



ΜΕΡΟΣ Α: Υλοποίηση VRAM

Στο Μέρος Α στόχος ήταν η υλοποίηση της VRAM με βάση τα κατάλληλα πρότυπα BRAM. Για την δημιουργία της εικόνας χρησιμοποιήθηκε ο παρακάτω πίνακας χρωμάτων όπου μας δείχνει την 3'bit κωδικοποίηση για καθένα από τα 8 χρώματα που μπορούν να αναπαρασταθούν με χρήση των τριών βασικών χρωμάτων Κόκκινο, Πράσινο και Μπλε. Συγκεκριμένα, η εικόνα που δημιουργήσαμε, αποτελείται από **Μαύρες**, **Μπλε**, **Πράσινες**, **Κόκκινες** και **Άσπρες** γραμμές και στήλες.

Κόκκινο (Red)	Πράσινο (Green)	Μπλε (Blue)	Συνισταμένη Χρώματος
0	0	0	Μαύρο
0	0	1	Μπλε
0	1	0	Πράσινο
0	1	1	Κυανό
1	0	0	Κόκκινο
1	0	1	Μοβ
1	1	0	Κίτρινο
1	1	1	Άσπρο

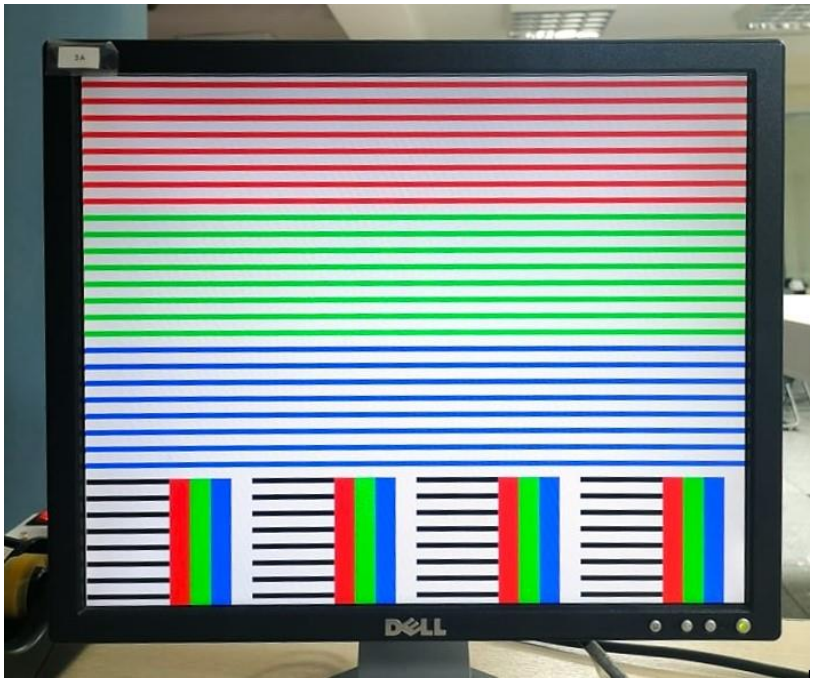
Για τις ανάγκες της δημιουργίας της εικόνας μας, υλοποιήσαμε **τρεις (3) BRAM** μνήμες, μία για κάθε βασικό χρώμα (Κόκκινο, Πράσινο, Μπλε) μεγέθους **18Kbytes × 1** έτσι ώστε να δημιουργήσουμε την απαιτούμενη **VRAM μεγέθους 128 × 96 pixel** καθώς λόγω των περιορισμών των συσκευών FPGA σε προσπελάσιμη μνήμη BRAM, το μέγεθος της VRAM, που θα υλοποιηθεί με BRAM θα πρέπει να είναι το 1/5 σε κάθε διάσταση, δηλ. 128 × 96. Ο Ελεγκτής με την σειρά του θα μεγεθύνει την VRAM ώστε να καλύπτει ανάλυσης 640 × 480.

Να σημειωθεί ότι για ανάλυση 640 × 480, απαιτούνται 640 × 480 = 307200 bits μνήμης, για κάθε μονόχρωμο pixel. Επομένως, εφόσον για την υλοποίηση μας χρειαζόμαστε τα τρία χρώματα που επιτρέπει η FPGA ανά pixel, απαιτούνται 3 bits ανά pixel, άρα η συνολική μνήμη που χρειάζεται για έγχρωμα pixel είναι 307200 × 3 = 115.2 KBytes μνήμης.

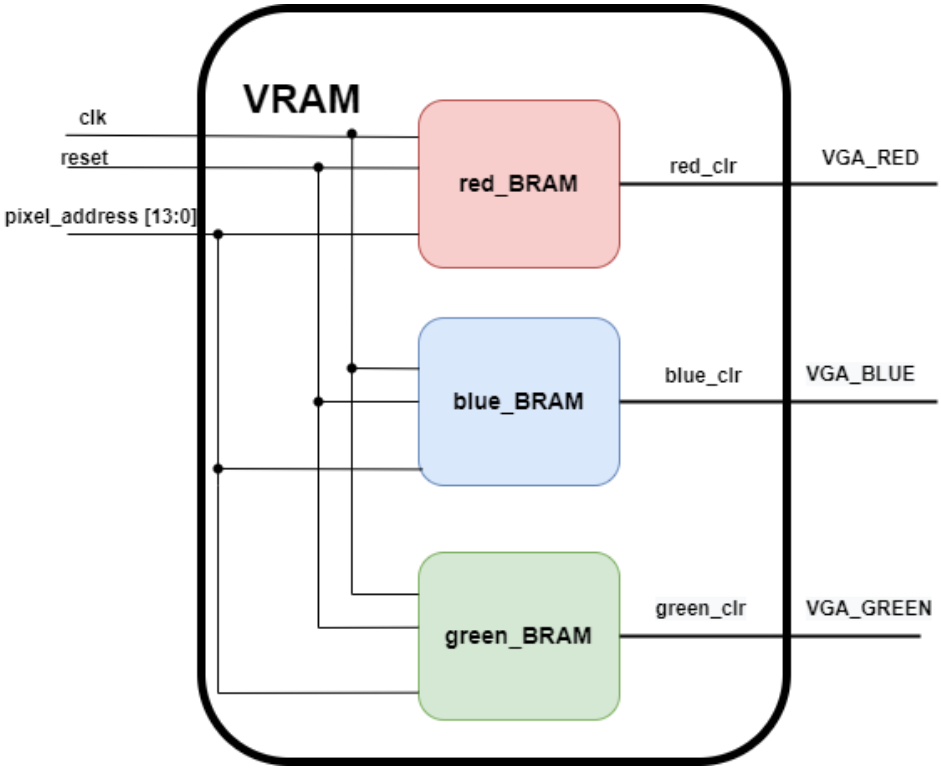
Κατά αυτόν τον τρόπο, η εικόνα που δημιουργήθηκε αποτελείται από (κοιτώντας την εικόνα από πάνω προς τα κάτω):

- Μία γραμμή άσπρη
- Οκτώ κόκκινες γραμμές εναλλάξ με 8 άσπρες διπλές γραμμές
- Οκτώ πράσινες γραμμές εναλλάξ με 8 άσπρες διπλές γραμμές

- Οκτώ μπλε γραμμές εναλλάξ με 8 άσπρες διπλές γραμμές
- Οκτώ μαύρες γραμμές εναλλάξ με 7 άσπρες διπλές γραμμές και μία άσπρη γραμμή στο τέλος
- ‘Μπροστά’ από την τελευταία κουκίδα γραμμών εμφανίζονται τέσσερις στήλες που η καθεμία αποτελείται από 4 μικρότερες στήλες με χρώματα Κόκκινο, Πράσινο, Μπλε και Άσπρο μοιρασμένες ανά ίσα διαστήματα στην οθόνη.



Το dataflow του Μέρους Α φαίνεται παρακάτω:



Οι τρεις μνήμες για κάθε βασικό χρώμα που αναφέρθηκαν παραπάνω αποτελούν τα **modules red_BRAM**, **blue_BRAM** και **green_BRAM** και όλες μαζί αποτελούν το **module VRAM.**

Το πρότυπο της BRAM επιλέχθηκε από το Vivado και χρησιμοποιήθηκε η «BRAM_SINGLE_MACRO». Οι παράμετροι και το Instantiation ορίστηκαν ως εξής:

```
BRAM_SINGLE_MACRO #(
      .BRAM_SIZE("18Kb")
      .DEVICE("7SERIES")
      .DO_REG(0)
      .INIT(36'h000000000)
      .INIT_FILE ("NONE"),
      .WRITE_WIDTH(1)
      .READ_WIDTH(1) //διαβάζουμε 1-bit χρώματος Pixel
      .SRVAL(36'h000000000)
      .WRITE_MODE("WRITE_FIRST")

      //lines katw <-- panw
      .INIT_00(256'hFFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF_F
      FFF_FFFF_FFFF_FFFF_FFFF_FFFF),//red(1h)-white
      .
      .
      .
      .INIT_2F(256'hF00F_FFFF_F00F_FFFF_F00F_FFFF_F00F_FFFF_F00F_0000_F
      00F_0000_F00F_0000_F00F_0000)//white-black(8h)
)
BRAM_SINGLE_MACRO_inst (
      .DO(red_clr)
      .ADDR(pxl_addr)
      .CLK(clk)
      //.DI(DI)
      .EN(1)
      //.REGCE(REGCE)
      .RST(reset)
      .WE(1'b0)
   );
```

(Παράδειγμα από την BRAM του κόκκινου χρώματος)

Οι παράμετροι **.WRITE_WIDTH** (δεν χρειάζεται απαραίτητα) και **.READ_WIDTH** αρχικοποιούνται σε 1 με βάση το manual της BRAM καθώς 1-bit διαβάζουμε (ή γράφουμε) κάθε φορά. Η αρχικοποίηση των **"INIT_xx"** έγινε με βάση τα χρώματα που θέλουμε να εμφανίζονται στην οθόνη. Κάθε δεκαεξαδικός αριθμός ενός **"INIT_xx"** αντιστοιχεί σε μία τετράδα δυαδικών αριθμών (πχ F-> 1111). Έτσι για κάθε μνήμη, ανάλογα με το εάν χρειάζεται να αναπαρίσταται το χρώμα της, αρχικοποιούμε κατάλληλα το κάθε pixel, δηλαδή το τελικό χρώμα κάθε pixel εξαρτάται

από τις αρχικοποιήσεις και των τριών μνημών. Επιπρόσθετα, κάθε **"INIT_xx"** περιέχει δύο γραμμές (0-127 και 128-256 bit) της VRAM άρα συνολικά 96 γραμμές, και 128 στήλες (128*96 μέγεθος). Στην έξοδο **.DO(x)** βγαίνει 1-bit, 0 ή 1, αναλόγως εάν το αντίστοιχο χρώμα θα χρησιμοποιηθεί για το χρώμα του Pixel, και η 14-bit έξοδος **.ADDR(y)** βγάζει την διεύθυνση του pixel. Παρόμοια υλοποιήθηκαν και οι υπόλοιπες BRAM.

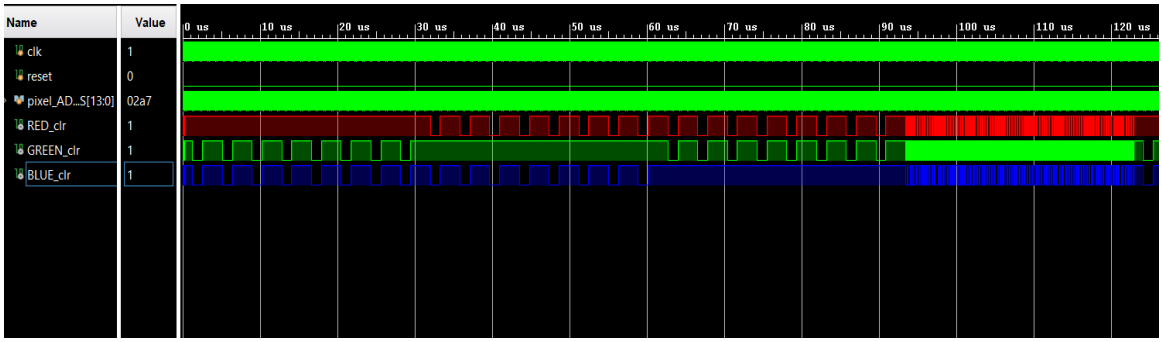
Παρακάτω φαίνεται το **module VRAM** στο οποίο κάνουμε instantiation τις τρείς μνήμες για κάθε χρώμα.

```
module VRAM(clk, reset, pixel_address, VGA_RED, VGA_GREEN, VGA_BLUE);
input clk, reset;
input [13:0] pixel_address;
output VGA_RED, VGA_GREEN, VGA_BLUE;

//instantiation of the three BRAMs one for its color
red_BRAM red_BRAM_inst(clk, reset, pixel_address, VGA_RED);
green_BRAM green_BRAM_inst(clk, reset, pixel_address, VGA_GREEN);
blue_BRAM blue_BRAM_inst(clk, reset, pixel_address, VGA_BLUE);

endmodule
```

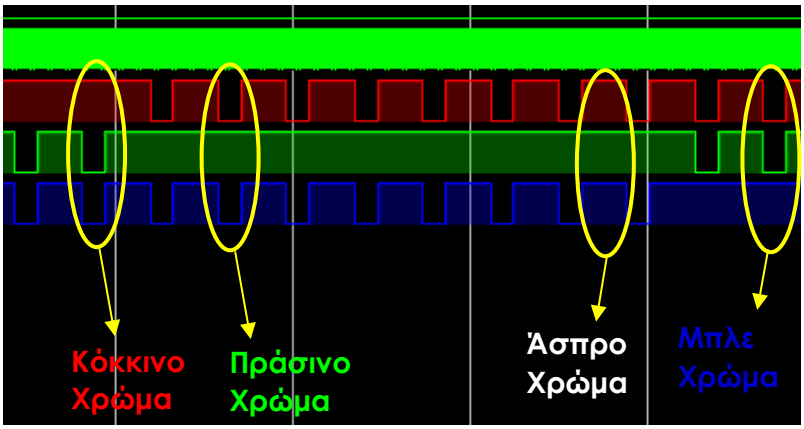
Όσον αφορά το testbench της VRAM (**module VRAM_tb**), υλοποιείται από ένα **initial block** με το οποίο αρχικοποιούμε το **clk**, το **reset** και το **pixel_ADDRESS** σε 0 και ένα μετρητή ο οποίος σε κάθε κύκλο ρολογιού αυξάνει την διεύθυνση κατά ένα ώστε να προσπελάσουμε όλη την μνήμη.

```
always@(posedge clk or posedge reset)
begin
    if(reset == 1'b1)
    begin
        pixel_ADDRESS = 14'b0000_0000_0000_0000;
    end
    else
    begin
        pixel_ADDRESS = pixel_ADDRESS + 14'b0000_0000_0000_0001;
    end
end
```

Εικόνα από τις κυματομορφές που παράγονται:

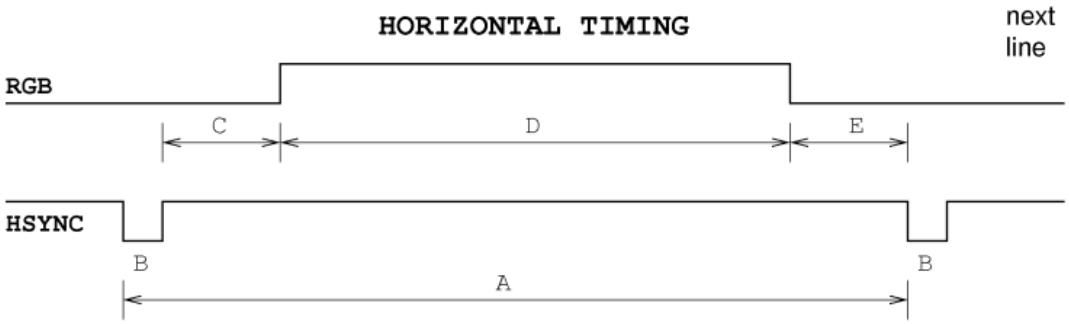


Με βάση τις κυματομορφές βλέπουμε ότι η εικόνα που θα παραχθεί είναι η επιθυμητή και επιβεβαιώνουμε ότι έχει γίνει σωστή αρχικοποίηση της μνήμης.



ΜΕΡΟΣ Β: Υλοποίηση HSYNC και Οριζόντιου Μετρητή Pixel

Σε αυτό το μέρος της εργασίας θα δούμε την υλοποίηση του σήματος HSYNC, σήμα που ελέγχει τον οριζόντιο χρονισμό της οθόνης, και του οριζόντιου μετρητή HPIXEL, που δείχνει κάθε φορά στο pixel στο οποίο βρίσκεται. Η υλοποίηση θα γίνει για ανάλυση 640 × 480 και ρυθμό ανανέωσης 60Hz. Όπως βλέπουμε στην παρακάτω εικόνα, το σήμα HSYNC παραμένει ένα για τα διαστήματα C, D, E και μηδέν στο διάστημα B.



Οι χρόνοι που υπολογίστηκαν για το κάθε διάστημα με βάση και τον παρακάτω πίνακα είναι:

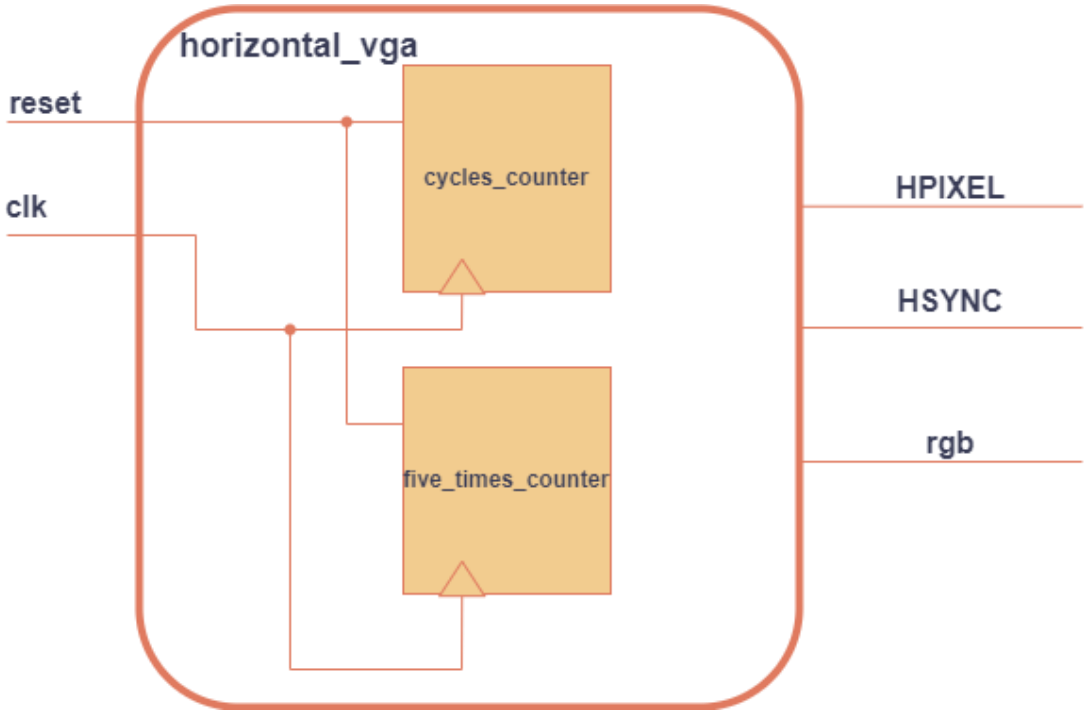
A	Χρόνος Σάρωσης Γραμμής	Scanline Time	32 μsec
B	Πλάτος Παλμού HSYNC	HSYNC Pulse Width	3.84 μsec
C	Πίσω Όψη	Back Porch	1.92 μsec
D	Χρόνος Απεικόνισης	Display Time	25.6 μsec
E	Μπροστινή Όψη	Front Porch	0.640 μsec

- **HSYNC Pulse Width:** 10ns αντιστοιχούν σε 1 κύκλο ρολογιού άρα 3,84us = 3840ns αντιστοιχούν σε **B=384** κύκλους ρολογιού. Ομοίως:
- **Back Porch: C=192** κύκλοι ρολογιού.
- **Front Porch: E=64** κύκλοι ρολογιού.
- **Display Time: D=2560** κύκλοι ρολογιού.
- **Scanline Time: A=3200** κύκλοι ρολογιού όπου A = B+C+E+D.

Η εικόνα παραμένει ενεργή για το χρονικό διάστημα D (Display Time). Μετά, για χρόνο E (Front Porch) η εμφάνιση της εικόνας απενεργοποιείται. Στην συνέχεια ακολουθεί το χρονικό διάστημα B (HSYNC Pulse Width), όπου το πλάτος και η συχνότητα του παλμού αυτού καταδεικνύουν στην οθόνη την οριζόντια ανάλυση. Τέλος, για

χρόνο C (Back Porch) η εικόνα μένει απενεργοποιημένη και συνεχίζει στην επόμενη γραμμή για χρόνο D.

Παρακάτω βλέπουμε το dataflow του μέρους B:



Για τις ανάγκες αυτού του μέρους της εργασίας, υλοποιήθηκε το **module horizontal_vga,** το οποίο στηρίζει την λειτουργία του σε μία **Mealy FSM 4 καταστάσεων**, τον **μετρητή HPIXEL**, τον **μετρητή cycles_counter** και τον **μετρητή five_times_counter.**

Η υλοποίηση του **μετρητή cycles_counter** φαίνεται παρακάτω:

```
always@(posedge clk or posedge reset)
begin
    if(reset == 1'b1)
    begin
        cycles_counter <= 12'b0000_0000_0000;
    end
    else
    begin
        if(rst_counter == 1'b1)
        begin
            cycles_counter <= 12'b0000_0000_0000;
        end
        else
        begin
            cycles_counter <= cycles_counter + 12'b0000_0000_0001;
        end
    end
end
```

Ο μετρητής αυτός μετράει κάθε φορά τους κύκλους ρολογιού ανάλογα με το state που βρίσκεται και όταν φτάνει στο μέγιστο η FSM πάει στο next state.

Η υλοποίηση του **μετρητή five_times_counter** φαίνεται παρακάτω:

```
always@(posedge clk or posedge reset)
begin
    if(reset == 1'b1)
    begin
        five_times_counter <= 8'b0000_0000;
    end
    else
    begin
        if(rst_5times_counter == 1'b1)
        begin
            five_times_counter <= 8'b0000_0000;
        end
        else
        begin
            five_times_counter <= five_times_counter + 8'b0000_0001;
        end
    end
end
```

Η λειτουργία αυτού του μετρητή είναι να στέλνει το ίδιο pixel, και επειδή D/pixel ανά γραμμή = 2560/640 = 4 , η μέγιστη τιμή του μετρητή είναι 4*5=20, άρα ουσιαστικά στέλνει 20 φορές το κάθε pixel.

Η υλοποίηση του **μετρητή HPIXEL** φαίνεται παρακάτω:

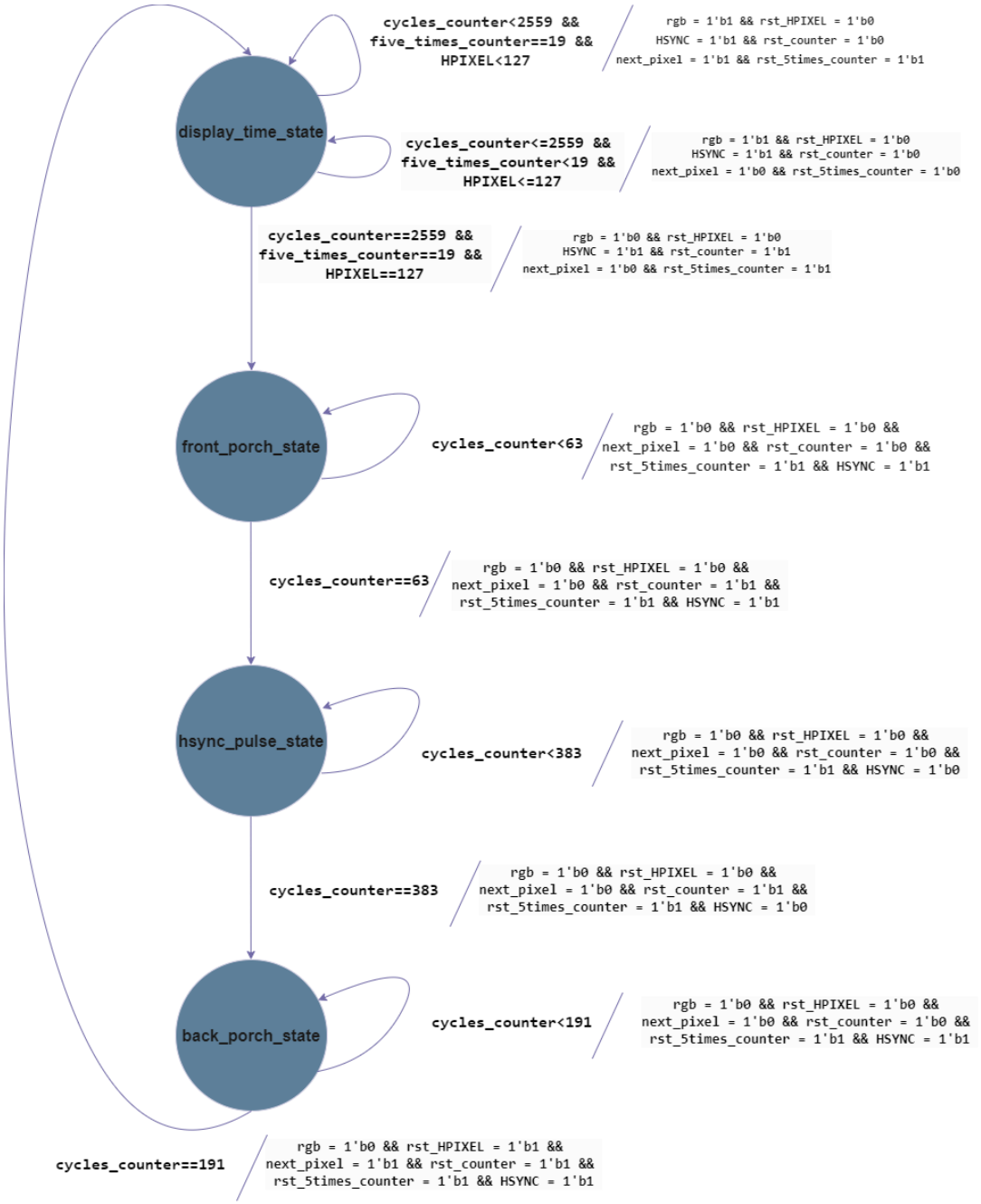
```
always@(posedge clk or posedge reset)
begin
    if(reset == 1'b1)
    begin
        HPIXEL = 7'b000_0000;
    end
    else
    begin
        if(rst_HPIXEL == 1'b1)
        begin
            HPIXEL = 7'b000_0000;
        end
        else if(next_pixel == 1'b1)
        begin
            HPIXEL = HPIXEL + 7'b000_0001;
```

```
        end
    end
end
```

Ο μετρητής αυξάνει τιμή κάθε φορά που έρχεται το σήμα **next_pixel == 1** από την FSM, ώστε να πάει στο επόμενο Pixel. Η μέγιστη τιμή του μετρητή είναι 127.

Η **FSM** είναι **Mealy** καθώς η έξοδος κάθε κατάστασης εξαρτάται από την είσοδο και την τρέχουσα κατάσταση.

Το διάγραμμα της FSM φαίνεται παρακάτω:



Οι καταστάσεις της είναι:

- **display_time_state:** κατάσταση κατά την οποία ο μετρητής HPIXEL αυξάνεται όταν κάθε pixel στέλνεται 5 φορές (διότι η VRAM είναι το 1/5 σε κάθε διάσταση) και HSYNC = 1
- **front_porch_state:** κατάσταση κατά την οποία η οθόνη παραμένει απενεργοποιημένη και HSYNC = 1
- **hsync_pulse_state:** κατάσταση κατά την οποία η οθόνη παραμένει απενεργοποιημένη και HSYNC = 0
- **back_porch_state:** κατάσταση κατά την οποία η οθόνη παραμένει απενεργοποιημένη και HSYNC = 1

```
parameter display_time_state = 2'b00;
parameter front_porch_state = 2'b01;
parameter hsync_pulse_state = 2'b10;
parameter back_porch_state = 2'b11;
```

Το sequential μέρος της φαίνεται παρακάτω:

```
//sequential fsm - allagi katastasis sthn FSM - ylopoiisi fsm gia ana-
parastasi twn pixel orizontiws
always@(posedge clk or posedge reset)
begin
    if(reset == 1'b1)
    begin
        CurrentState <= display_time_state;

    end
    else
    begin
        CurrentState <= NextState;
    end
end
```

Όταν έρχεται σήμα reset == 1, η FSM μεταβαίνει στο display_time_state αλλιώς στο επόμενο state.

Το combinational μέρος της είναι (παραλείπονται μέρη του κώδικα):

```
always@(CurrentState or cycles_counter or five_times_counter or HPIXEL)
begin
    case(CurrentState)
      display_time_state:
       begin
            .
            .
            if(cycles_counter < 12'b1001_1111_1111 && five_times_coun-
ter == 8'b0001_0011 && HPIXEL<7'b111_1111)
            begin
```

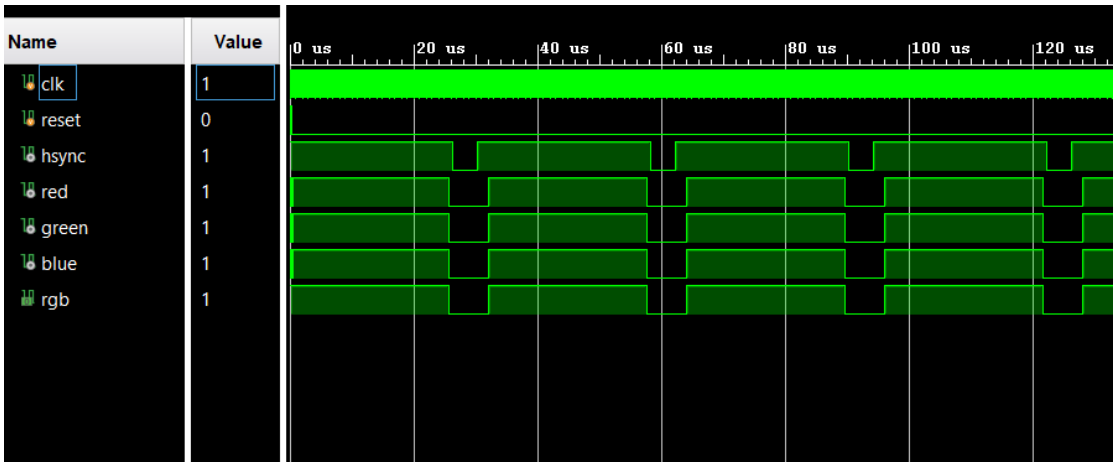
```
                .
                .
                NextState = display_time_state;
            end

            else if(cycles_counter == 12'b1001_1111_1111 &&
five_times_counter == 8'b0001_0011 && HPIXEL == 7'b111_1111)//HPIXEL ==
127
            begin
                .
                .
                NextState = front_porch_state;
            end
            else
            begin
                .
                .
                NextState = display_time_state;
            end
        end

        front_porch_state:
        begin
            .
            .
            if(cycles_counter == 12'b0000_0011_1111)//64-1=63
            begin
                .
                NextState = hsync_pulse_state;
            end
            else
            begin
                NextState = front_porch_state;
            end
        end

        hsync_pulse_state:
        begin
            .
            .
            if(cycles_counter == 12'b0001_0111_1111)//384-1=383
            begin
                .
                NextState = back_porch_state;
            end
            else
            begin
                NextState = hsync_pulse_state;
```

```
            end
        end

        back_porch_state:
        begin
            .
            .
            if(cycles_counter == 12'b0000_1011_1111)//192-1=191
            begin
                .
                NextState = display_time_state;
            end
            else
            begin
                NextState = back_porch_state;
            end
        end
    endcase
end
```

Για την επαλήθευση του Μέρους Β δημιουργήθηκε ένα απλό testbench που κάνει instantiation το Module horizontal_vga και VRAM. Στην VRAM για την διεύθυνση του κάθε pixel (pixel_address)κάνει concat μία τιμή για το VPIXEL που την δίνουμε εμείς από το testbench και την τιμή του hpixel, ώστε να δημιουργήσει μία 14-bit διεύθυνση.

```
VRAM VRAM_inst(.clk(clk), .reset(reset), .pixel_address({0,hpixel}),
.VGA_RED(VGA_RED), .VGA_GREEN(VGA_GREEN), .VGA_BLUE(VGA_BLUE));
```

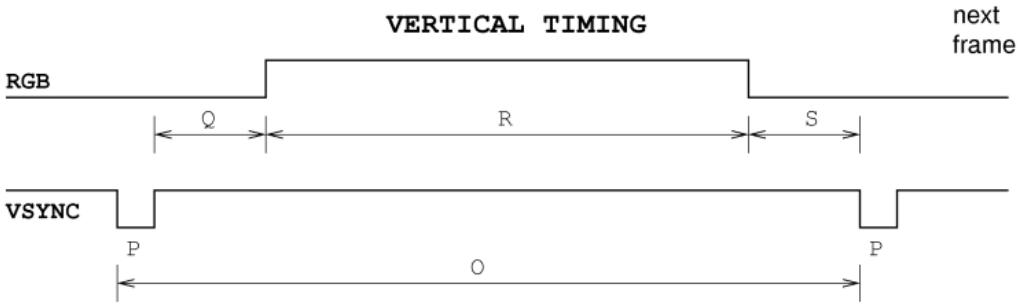
Έτσι για το παραπάνω instantiation της VRAM παίρνουμε τις παρακάτω κυματομορφές, οι οποίες είναι σωστές καθώς στην πρώτη γραμμή της οθόνης το χρώμα που εμφανίζεται είναι το άσπρο, άρα και τα τρία βασικά χρώματα πρέπει να είναι 1.



Παρατηρούμε ότι όταν το σήμα **rgb** γίνεται 1, τα χρώματα οδηγούνται όλα στο 0, δηλαδή η οθόνη απενεργοποιείται.

ΜΕΡΟΣ Γ: Υλοποίηση VSYNC και Κατακόρυφου Μετρητή Pixel — Ολοκλήρωση του Ελεγκτή/Οδηγού VGA

Στο μέρος Γ της εργασίας πραγματοποιείται η υλοποίηση του σήματος VSYNC, σήμα που ελέγχει τον κατακόρυφο χρονισμό της οθόνης, του κατακόρυφου μετρητή VPIXEL, που δείχνει κάθε φορά στην γραμμή της VRAM όπου βρίσκεται και τέλος ο συνδυασμός του μέρους Β με το Γ βασιζόμενοι στον κατάλληλο συγχρονισμό της κατακόρυφης με την οριζόντια σάρωση. Η υλοποίηση θα γίνει για ανάλυση 640 × 480 και ρυθμό ανανέωσης 60Hz. Το σήμα VSYNC παραμένει ένα για τα διαστήματα Q, R, S και μηδέν στο διάστημα P.



Με βάση τον παρακάτω πίνακα οι χρόνοι που υπολογίστηκαν για το κάθε διάστημα είναι:

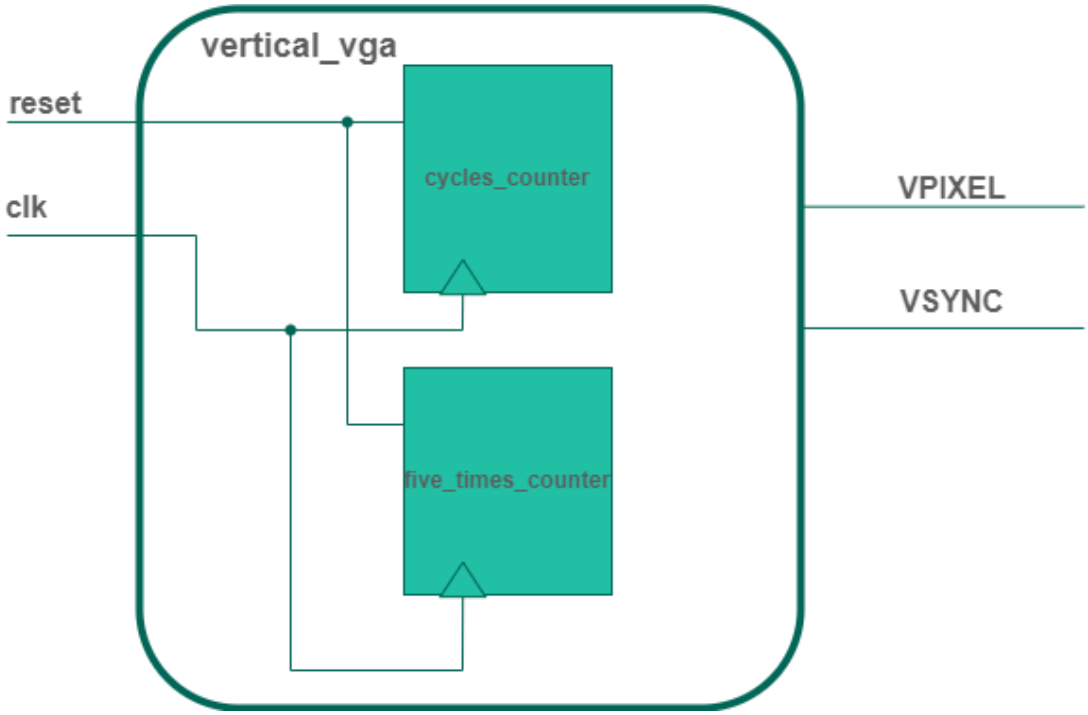
O	Συνολικός Χρόνος Εικόνας	Total Frame Time	16.67 msec
P	Πλάτος Παλμού VSYNC	VSYNC Pulse Width	64 μsec
Q	Πίσω Όψη	Back Porch	928 μsec
R	Χρόνος Ενεργής Απεικόνισης	Active Video Time	15.36 msec
S	Μπροστινή Όψη	Front Porch	320 μsec

- **VSYNC Pulse Width:** 10ns αντιστοιχούν σε 1 κύκλο ρολογιού άρα 64us = 64000ns αντιστοιχούν σε **P=6400** κύκλους ρολογιού. Ομοίως:
- **Back Porch: Q=92800** κύκλοι ρολογιού.
- **Front Porch: S=32000** κύκλοι ρολογιού.
- **Active Video Time: R=1.536.000** κύκλοι ρολογιού.

- **Total Frame Time: O= 1.667.200** κύκλοι ρολογιού όπου O = P+Q+S+R.

Για το χρονικό διάστημα R (Active Video Time) η εικόνα παραμένει ενεργή. Μετά, για χρόνο S (Front Porch) η εμφάνιση της εικόνας απενεργοποιείται. Στην συνέχεια ακολουθεί το χρονικό διάστημα P (VSYNC Pulse Width), όπου το πλάτος και η συχνότητα του παλμού αυτού καταδεικνύουν στην οθόνη την κατακόρυφη ανάλυση. Τέλος, για χρόνο Q (Back Porch) η εικόνα μένει απενεργοποιημένη και συνεχίζει στο επόμενο frame για χρόνο R.

Παρακάτω φαίνεται το dataflow του μέρους Γ:



Και σε αυτό το μέρος της εργασίας, η υλοποίηση του **module vertical_vga,** λειτουργεί με βάση μία **Mealy FSM 4 καταστάσεων,** τον **μετρητή VPIXEL,** τον **μετρητή cycles_counter** και τον **μετρητή five_times_counter,** όπου οι δύο τελευταίοι είναι ακριβώς ίδιοι με το προηγούμενο μέρος(το μόνο που αλλάζει είναι οι μέγιστες τιμές που λαμβάνουν).

Ο μετρητής **cycles_counter** είναι 24'bit αυτή την φορά καθώς καλείται να μετρήσει πολύ περισσότερους κύκλους ρολογιού:

```
always@(posedge clk or posedge reset)
begin
    if(reset == 1'b1)
    begin
        cycles_counter <= 24'b0000_0000_0000_0000_0000_0000;
    end
```

```
    else
    begin
        if(rst_counter == 1'b1)
        begin
            cycles_counter <= 24'b0000_0000_0000_0000_0000_0000;
        end
        else
        begin
            cycles_counter <= cycles_counter +
24'b0000_0000_0000_0000_0000_0001;
        end
    end
end
```

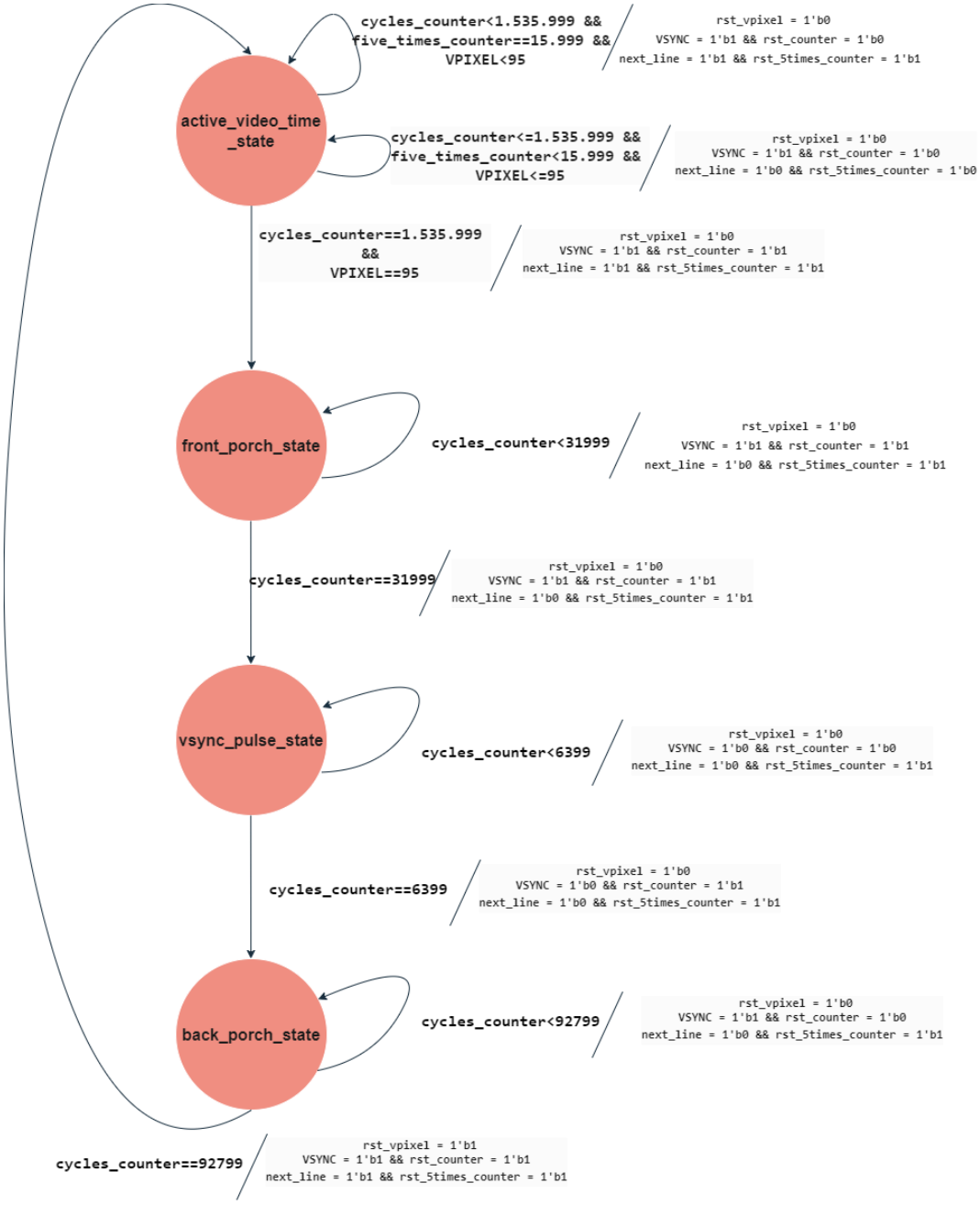
Ο μετρητής **five_times_counter** δεν διαφέρει με αυτόν του μέρους Β γι αυτό και δεν παρατίθεται ο κώδικας.

Όσον αφορά τον μετρητή **VPIXEL**, δεν διαφέρει ως προς την υλοποίηση αλλά διαφέρει ως προς την λειτουργία, δηλαδή ο VPIXEL μετράει τις γραμμές τις VRAM και κάθε φορά που φτάνει στο μέγιστο πάει στο επόμενο frame.

```
always@(posedge clk or posedge reset)
begin
    if(reset == 1'b1)
    begin
        VPIXEL = 7'b000_0000;
    end
    else
    begin
        if(rst_vpixel == 1'b1)
        begin
            VPIXEL = 7'b000_0000;
        end
        else if(next_line == 1'b1)
        begin
            VPIXEL = VPIXEL + 7'b000_0001;
        end
    end
end
```

Η **FSM** είναι και αυτή **Mealy** καθώς η έξοδος κάθε κατάστασης εξαρτάται από την είσοδο και την τρέχουσα κατάσταση.

Το διάγραμμα της FSM φαίνεται παρακάτω:



Οι καταστάσεις της είναι:

- **Active_video_time_state:** κατάσταση κατά την οποία ο μετρητής VPIXEL αυξάνεται όταν κάθε γραμμή στέλνεται 5 φορές (διότι η VRAM είναι το 1/5 σε κάθε διάσταση) και VSYNC = 1

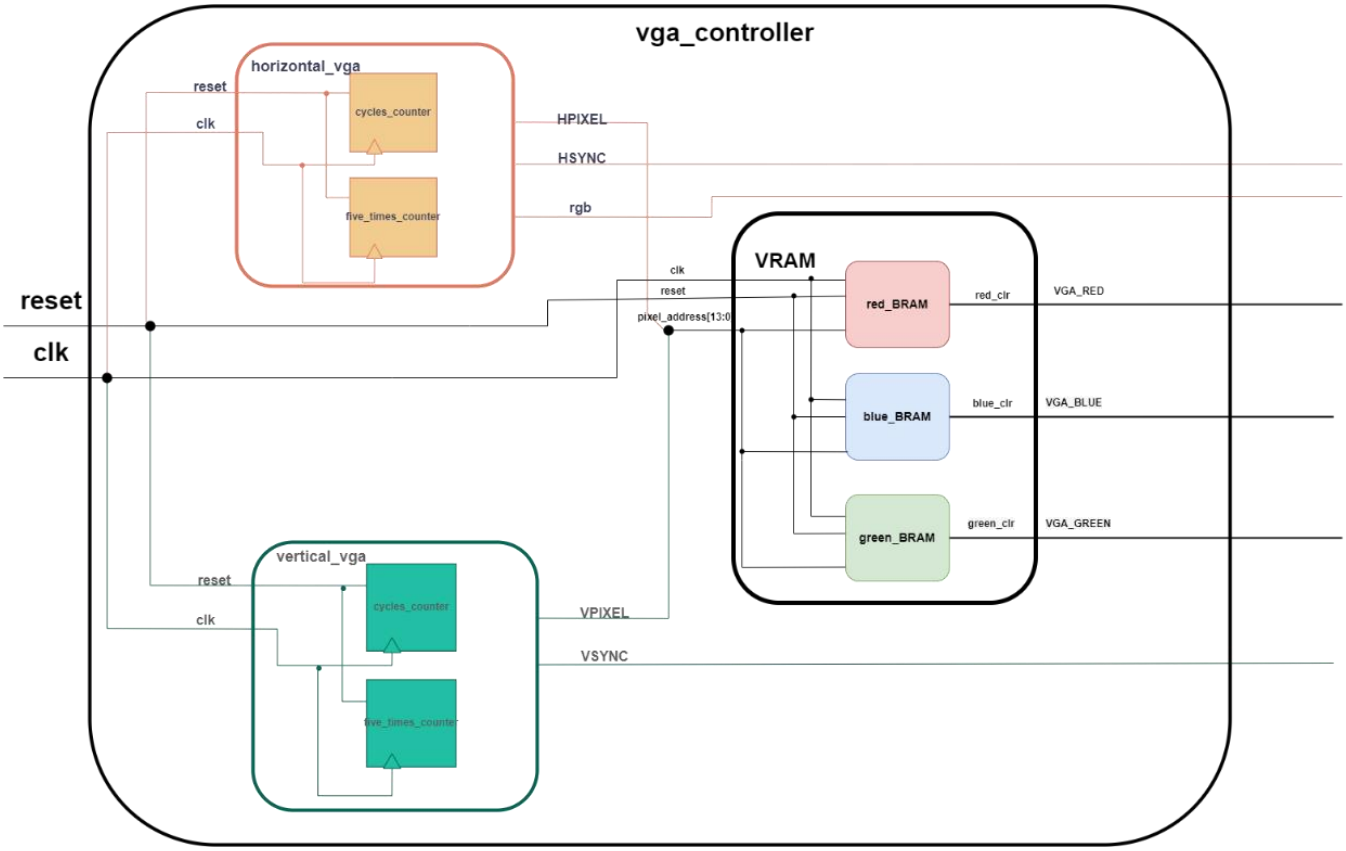
- **front_porch_state:** κατάσταση κατά την οποία η οθόνη παραμένει απενεργοποιημένη και VSYNC = 1
- **vsync_pulse_state:** κατάσταση κατά την οποία η οθόνη παραμένει απενεργοποιημένη και VSYNC = 0
- **back_porch_state:** κατάσταση κατά την οποία η οθόνη παραμένει απενεργοποιημένη και VSYNC = 1

Η υλοποίηση σε γλώσσα Verilog είναι σχεδόν ίδια, με μόνη διαφορά τα states και τις μέγιστες τιμές των μετρητών, γι αυτό και δεν παρατίθεται παρακάτω ο σχετικός κώδικας.

Έχει δημιουργηθεί και κατάλληλο testbench (vertical_vga_tb) για το μέρος Γ, το οποίο για συγκεκριμένη τιμή του Hpixel που δίνεται από τον χειριστή του testbench δείχνει τα χρώματα της οθόνης αναλόγως το Vpixel.

Το κυριότερο σημείο αυτού του μέρους είναι η ένωση των **module horizontal_vga** και **vertical_vga** με την **VRAM**, στο **module vga_controller** για την δημιουργία της τελικής εικόνας στην οθόνη.

Παρακάτω φαίνεται το dataflow του **module vga_controller**:



Όπως φαίνεται παραπάνω, οι τιμές των μετρητών HPIXEL και VPIXEL δημιουργούν την διεύθυνση της VRAM pixel_address (pixel_address = {VPIXEL,

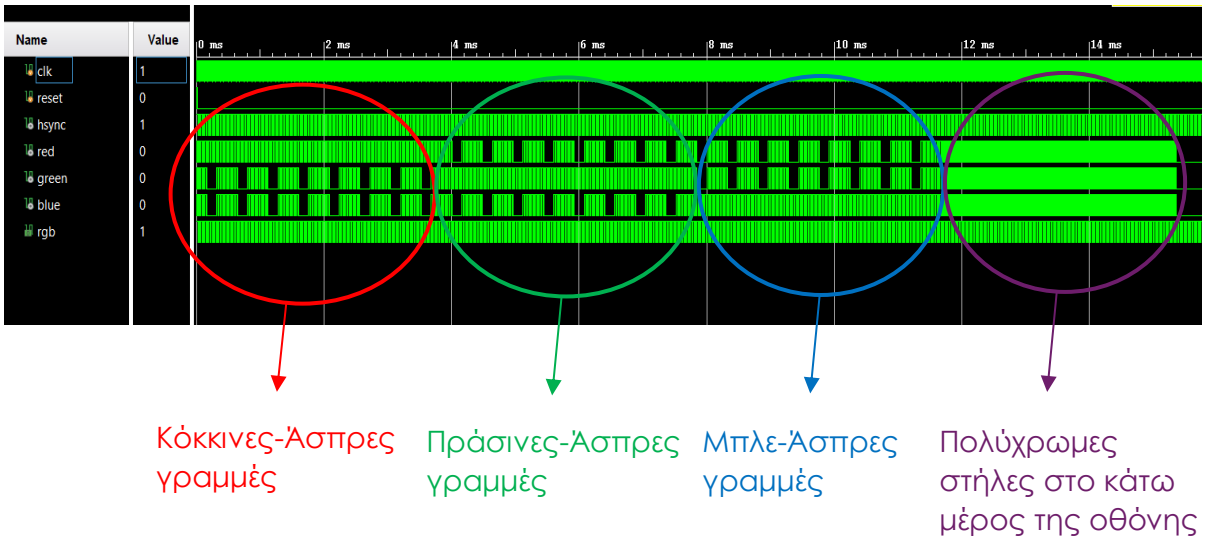
HPIXEL}). Η VRAM ανάλογα με την διεύθυνση του αντίστοιχου pixel βγάζει στην έξοδο 1bit από την κάθε BRAM για τα τρία βασικά διαφορετικά χρώματα και ο συνδυασμός τους συνιστά το τελικό χρώμα του Pixel στην οθόνη.

Το στο **module vga_controller** αποτελείται από το instantiation των τριών προαναφερθεισών modules και ένα always block το οποίο είναι υπεύθυνο για την απενεργοποίηση της οθόνης όπως αυτή ορίζεται από το σήμα rgb.

```
horizontal_vga horizontal_vga_inst(.clk(clk), .reset(reset),
.HPIXEL(hpixel), .HSYNC(VGA_HSYNC), .rgb(rgb));
vertical_vga vertical_vga_inst(.clk(clk), .reset(reset),
.VPIXEL(vpixel), .VSYNC(VGA_VSYNC));
VRAM VRAM_inst(.clk(clk), .reset(reset), .pixel_address({vpixel,
hpixel}), .VGA_RED(red), .VGA_GREEN(green), .VGA_BLUE(blue));

always@(posedge clk or posedge reset)
begin
    if(reset == 1'b1)
    begin
        VGA_RED <= 1'b0;
        VGA_BLUE <= 1'b0;
        VGA_GREEN <= 1'b0;
    end
    else
    begin
        if(rgb == 1'b1)
        begin
            VGA_RED <= red;
            VGA_BLUE <= blue;
            VGA_GREEN <= green;
        end
        else
        begin
            VGA_RED <= 1'b0;
            VGA_BLUE <= 1'b0;
            VGA_GREEN <= 1'b0;
        end
    end
end
```

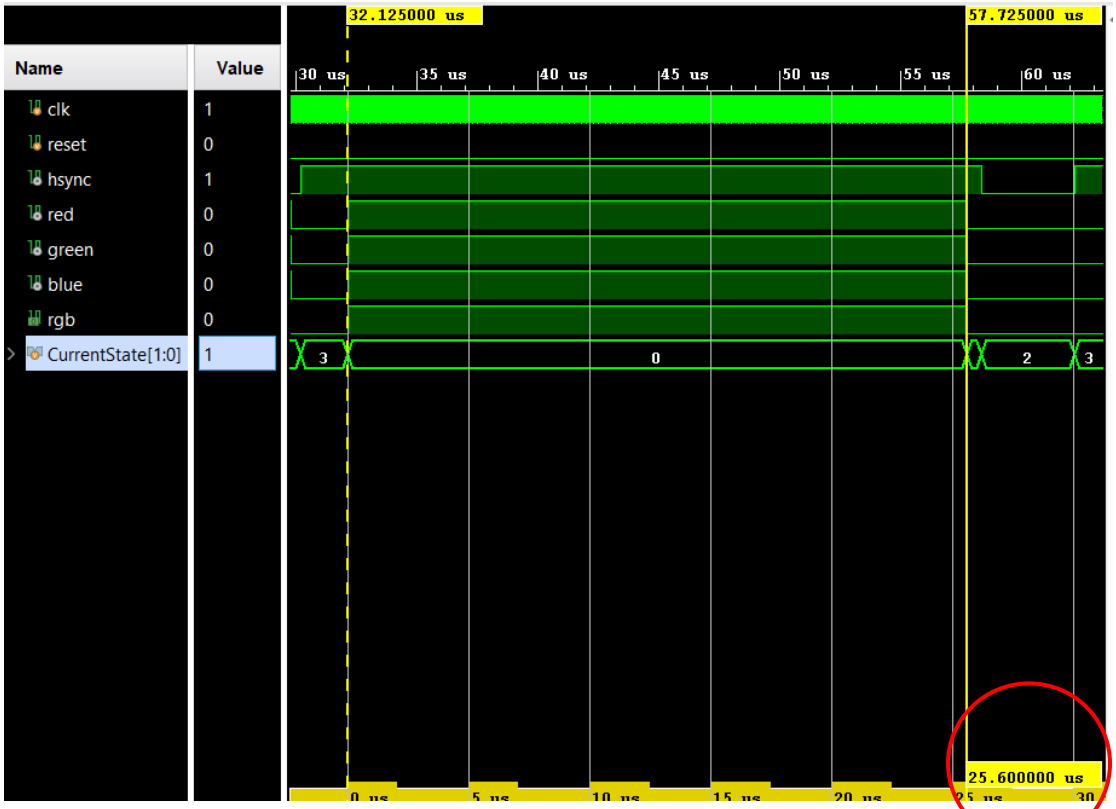
Για το module αυτό, υπάρχει και το σχετικό testbench (vga_controller_tb) το οποίο κάνει instantiation το vga_controller module και παράγονται οι παρακάτω κυματομορφές:

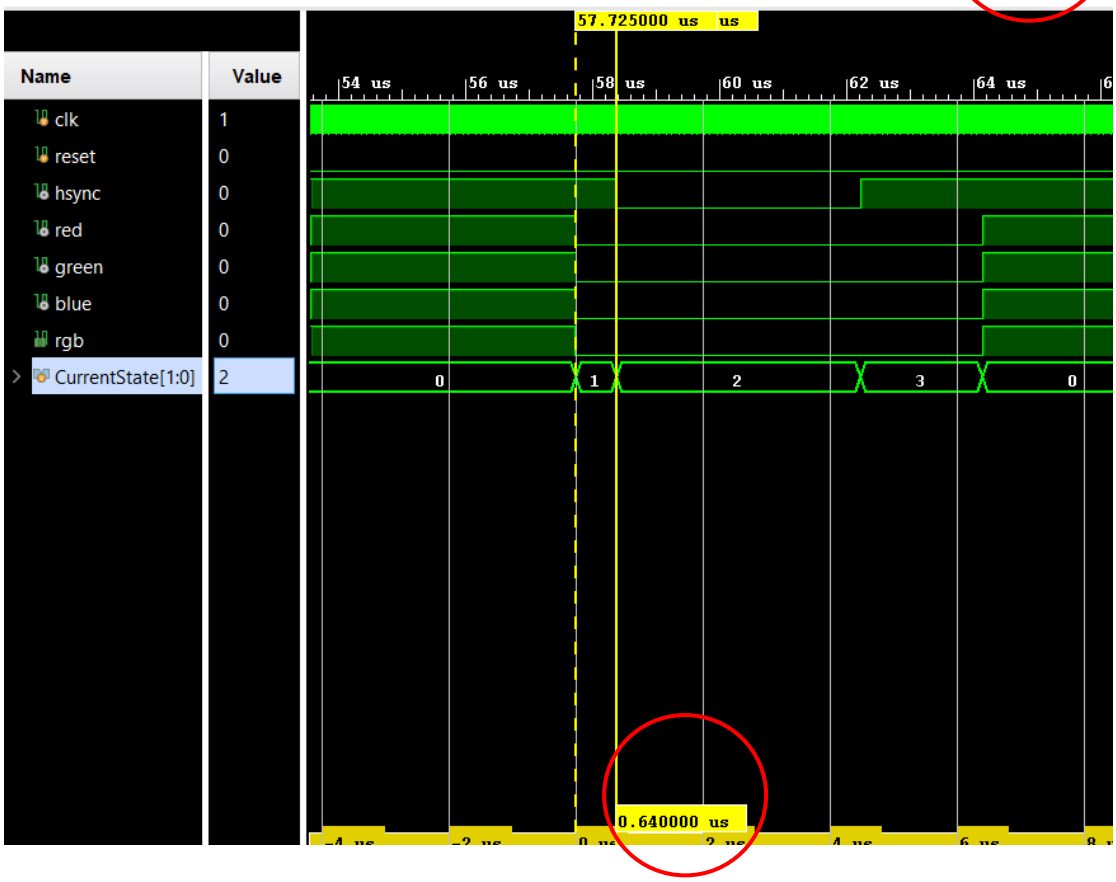
Εδώ βλέπουμε τις κυματομορφές για ένα frame και πως αναπαρίσταται κάθε κομμάτι της οθόνης.

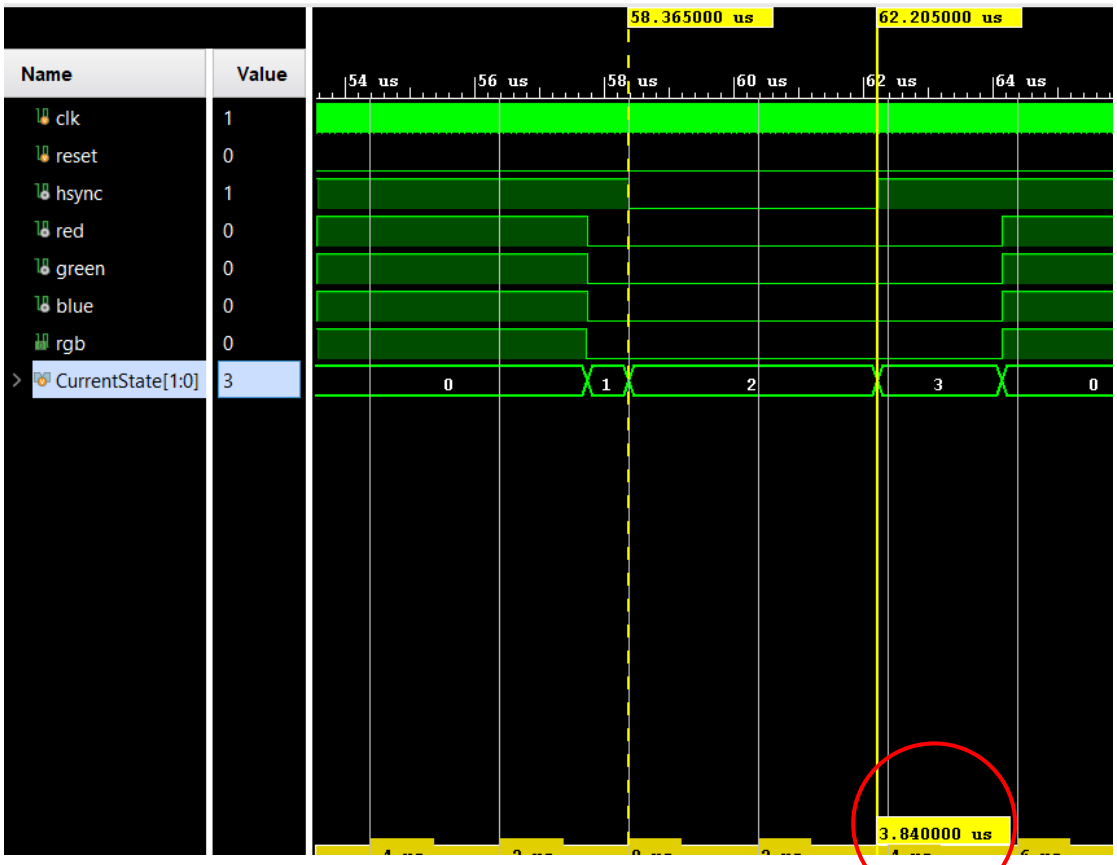


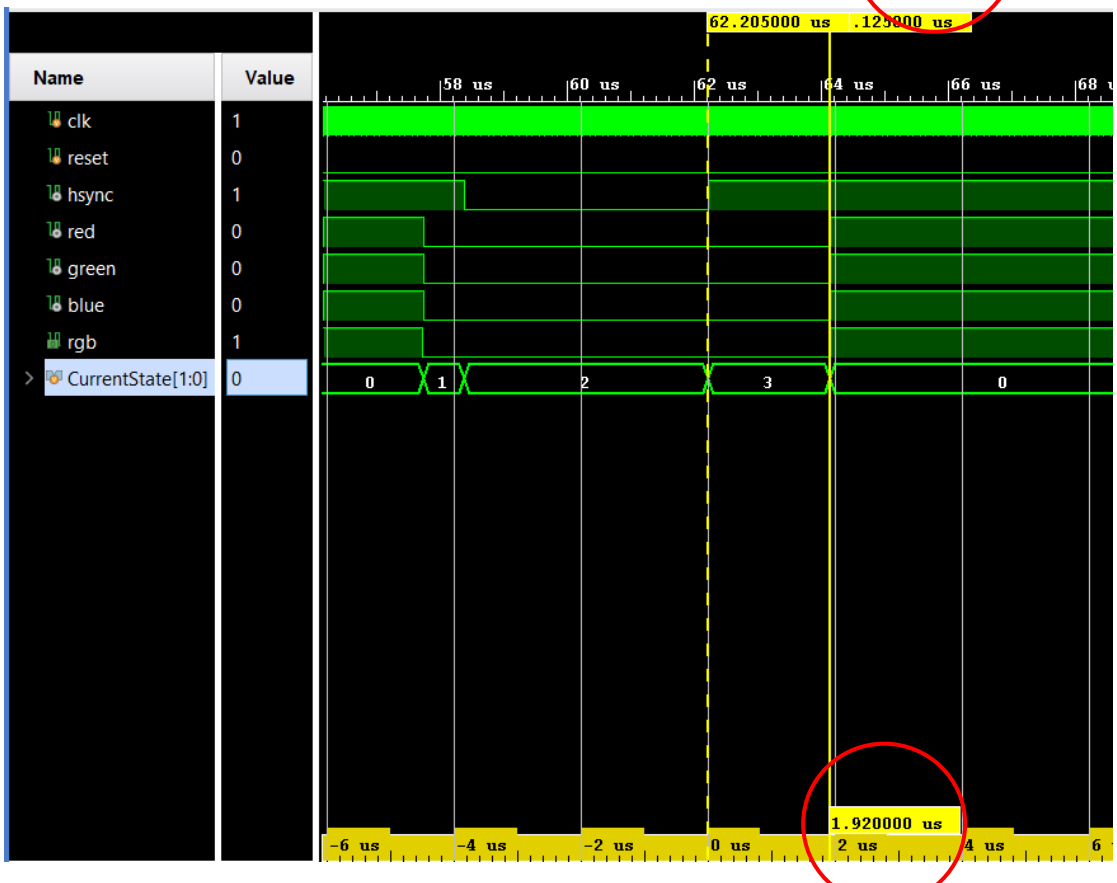
Για να βεβαιωθούμε ότι οι εναλλαγές γίνονται στο σωστό χρόνο μετράμε για κάθε state από τα modules horizontal_vga και vertical_vga τους χρόνους των καταστάσεων.

Για την οριζόντια σάρωση έχουμε:


Name
Value
clk
1
reset
0
hsync
1
red
0
green
0
blue
0
rgb
0
CurrentState[1:0]
1
32.125000 us
57.725000 us
30 us
35 us
40 us
45 us
50 us
55 us
60 us
3
0
2
3
25.600000 us


Name
Value
clk
1
reset
0
hsync
0
red
0
green
0
blue
0
rgb
0
CurrentState[1:0]
2
57.725000 us
54 us
56 us
58 us
60 us
62 us
64 us
0
1
2
3
0
0.640000 us

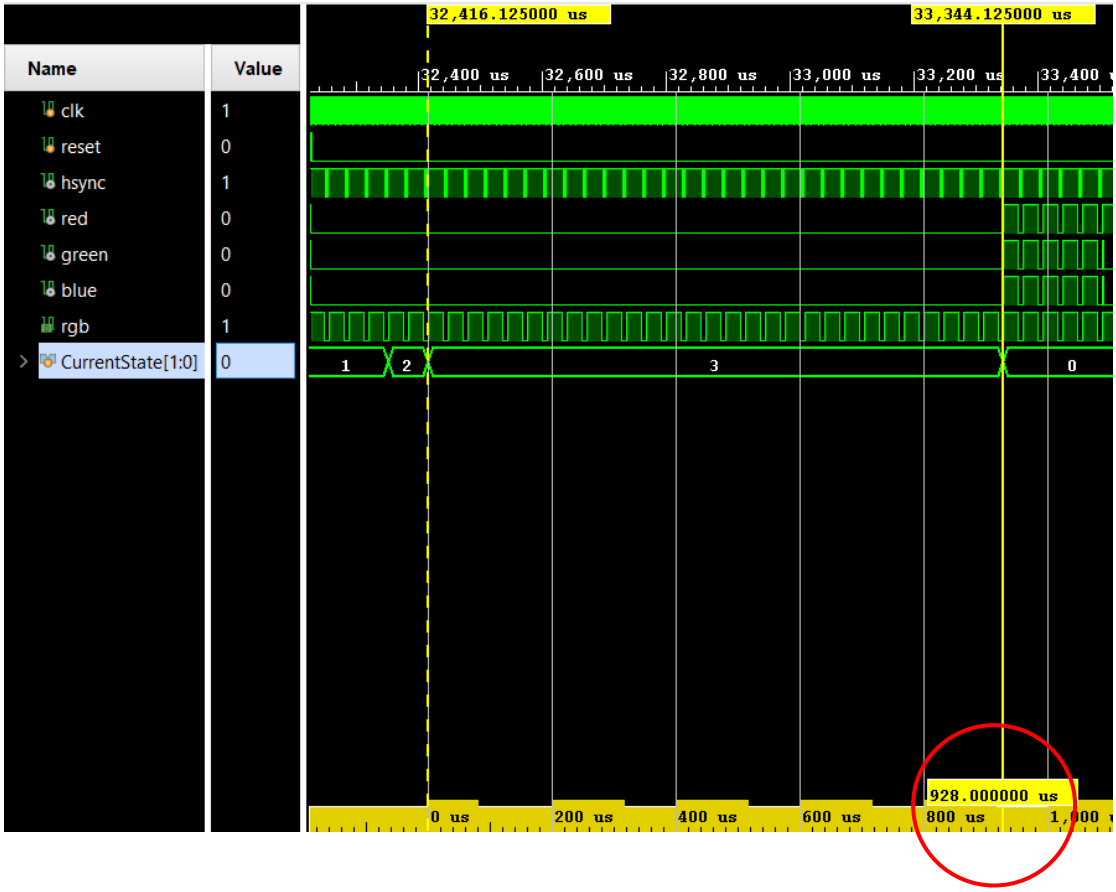




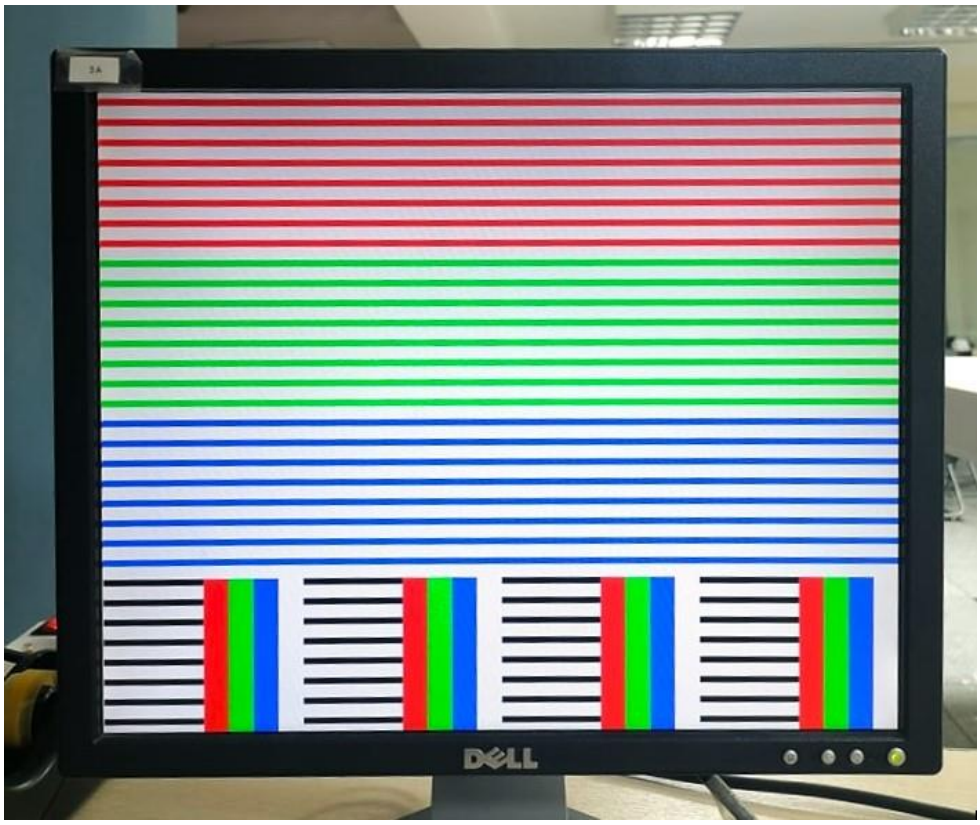
Για την κατακόρυφη σάρωση έχουμε:


Name
Value
clk
1
reset
0
hsync
1
red
0
green
0
blue
0
rgb
1
CurrentState[1:0]
1
16.672125000 ms
32.032125000 ms
10 ms
20 ms
30 ms
0
3
0
3
0
15.360000000 ms
-10 ms
0 ms
10 ms


Name
Value
clk
1
reset
0
hsync
1
red
0
green
0
blue
0
rgb
1
CurrentState[1:0]
2
332,032.125000 us
30,000 us
30,500 us
31,000 us
31,500 us
32,000 us
32,500 us
0
1
3
320.000000 us
-2,000 us
-1,500 us
-1,000 us
-500 us
0 us
500 us
Name
Value
clk
1
reset
0
hsync
1
red
0
green
0
blue
0
rgb
1
CurrentState[1:0]
3
32,352.125000 us
32,200 us
32,400 us
32,600 us
32,800 us
1
2
3
64.000000 us
-200 us
200 us
400 us
600 us

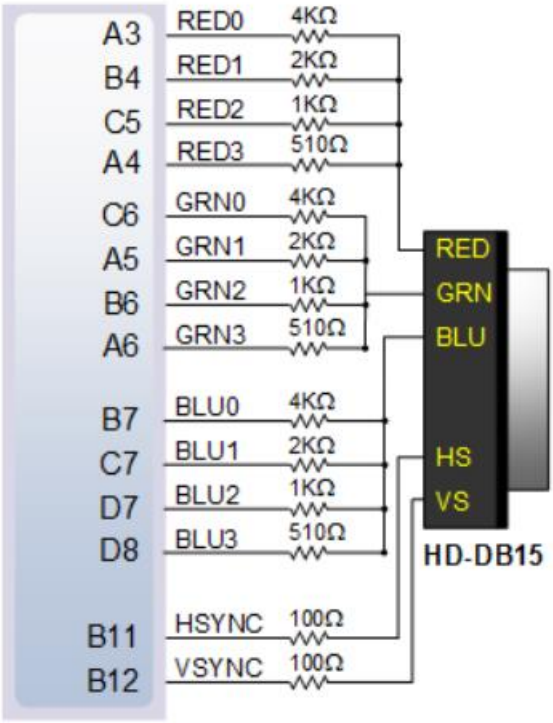


Έτσι με βάση την αρχικοποίηση της μνήμης που έχει προηγηθεί, δημιουργείται η παρακάτω εικόνα:



Η εμφάνιση της σωστής εικόνας στην οθόνη, χωρίς αλλοιώσεις στην στοίχιση των Pixels σημαίνει ότι επιτεύχθηκε με επιτυχία ο κατάλληλος συντονισμός των σημάτων HSYNC και VSYNC.

Σχετικά με το αρχείο **.xdc** η αντιστοίχιση των εισόδων-εξόδων με τα ports της πλακέτας Nexys A7-100T έγινε με βάση το manual της πλακέτας.



Έτσι, κάθε χρώμα συνδέθηκε στο αντίστοιχο port, μόνο που επειδή στην υλοποίηση μας, τα VGA_RES, VGA_GREEN και VGA_BLUE είναι 1-bit, συνδέσαμε 4 φορές το καθένα στα ports του αντίστοιχου χρώματος. Ακολουθεί το αρχείο:

```
## This file is a general .xdc for the Nexys A7-100T
### Clock Signal
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports {
clk }];
create_clock -add -name sys_clk -period 10.00 -waveform {0 5}
[get_ports { clk }];
###Colors Red-Green-Blue and signals HSYNC-VSYNC
set_property -dict { PACKAGE_PIN A3 IOSTANDARD LVCMOS33 } [get_ports {
VGA_RED }];
set_property -dict { PACKAGE_PIN B4 IOSTANDARD LVCMOS33 } [get_ports {
VGA_RED }];
set_property -dict { PACKAGE_PIN C5 IOSTANDARD LVCMOS33 } [get_ports {
VGA_RED }];
set_property -dict { PACKAGE_PIN A4 IOSTANDARD LVCMOS33 } [get_ports {
VGA_RED }];
set_property -dict { PACKAGE_PIN C6 IOSTANDARD LVCMOS33 } [get_ports {
VGA_GREEN }];
```

```
set_property -dict { PACKAGE_PIN A5 IOSTANDARD LVCMOS33 } [get_ports {
VGA_GREEN }];
set_property -dict { PACKAGE_PIN B6 IOSTANDARD LVCMOS33 } [get_ports {
VGA_GREEN }];
set_property -dict { PACKAGE_PIN A6 IOSTANDARD LVCMOS33 } [get_ports {
VGA_GREEN }];
set_property -dict { PACKAGE_PIN B7 IOSTANDARD LVCMOS33 } [get_ports {
VGA_BLUE }];
set_property -dict { PACKAGE_PIN C7 IOSTANDARD LVCMOS33 } [get_ports {
VGA_BLUE }];
set_property -dict { PACKAGE_PIN D7 IOSTANDARD LVCMOS33 } [get_ports {
VGA_BLUE }];
set_property -dict { PACKAGE_PIN D8 IOSTANDARD LVCMOS33 } [get_ports {
VGA_BLUE }];
set_property -dict { PACKAGE_PIN B11 IOSTANDARD LVCMOS33 } [get_ports {
VGA_HSYNC }];
set_property -dict { PACKAGE_PIN B12 IOSTANDARD LVCMOS33 } [get_ports {
VGA_VSYNC }];

set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS33 } [get_ports {
reset }];
```

ΣΥΜΠΕΡΑΣΜΑΤΑ

Τα γενικά συμπεράσματα που προκύπτουν μετά το τέλος της εργασίας αυτής είναι:

- Το μέρος Α της εργασίας ήταν δεν περιείχε ιδιαίτερη δυσκολία, αλλά απαιτούσε την πλήρη κατανόηση της λειτουργίας της BRAM για την σωστή αρχικοποίηση της.
- Τα μέρη Β και Γ, όσον αφορά την υλοποίηση των δύο FSM, είχαν την δυσκολία του συγχρονισμού της οριζόντιας με την κάθετη σάρωση.
- Οι δυσκολίες που αντιμετώπισα ήταν η αρχική κάπως λάθος κατανόηση της BRAM που οδήγησε σε λάθη κατά την οριζόντια και κάθετη σάρωση. Ωστόσο αυτά διορθώθηκαν και έτσι η εργασία υλοποιήθηκε επιτυχώς , έχοντας στην οθόνη την τελική εικόνα.