ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ (ΗΥ430)

ΤΕΧΝΙΚΗ ΑΝΑΦΟΡΑ 3ΗΣ ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

ΠΑΝΑΓΙΩΤΗΣ ΑΝΑΣΤΑΣΙΑΔΗΣ 2134

ΤΙΤΛΟΣ: «ΥΛΟΠΟΙΗΣΗ ΕΛΕΓΚΤΗ VGA»

Περίληψη τεχνικής αναφοράς

Η τεχνική αναφορά της 3ης εργαστηριακής άσκησης περιγράφει τα 3 διαφορετικά μέρη από τα οποία σταδιακά υλοποιήθηκε ο ελεγκτής VGA (Video Ram, HSYNC, VSYNC), μαζί με το τελευταίο κομμάτι που οι μονάδες συνενώνονται σε μια και υλοποιούν σαν μια ενιαία οντότητα τον ελεγκτή..

- ***Στόχοι***: Εισαγωγή στην 3η εργαστηριακή εργασία, αναλύοντας τα ζητούμενα και τους στόχους που απαιτεί να επιτευχθούν κατά τη δομή της τελικής υλοποίησης.
- ***Μέρος Α***: Σχεδιασμός της Video Ram. Η Video Ram απαρτίζεται από 3 block ram 16k x 1 της πλακέτας SPARTAN 3 (μια για κάθε χρώμα R,G,B) και στο εσωτερικό της είναι αποθηκευμένη η εικόνα επαλήθευσης του ελεγκτή VGA.
- ***Μέρος Β***: Υλοποίηση του HSYNC Driver και HPIXEL Controller. Η πρώτη μονάδα είναι υπεύθυνη για τον σωστό χρονισμό του σήματος HSYNC που απαιτεί μια οθόνη προδιαγραφών 640x480, 60 Hz. Η δεύτερη μονάδα επιδεικνύει την οριζόντια διεύθυνση του εκάστοτε ενεργού pixel, βάσει του οποίου πρέπει η VRAM να εξάγει το σωστό συνδυασμό χρωμάτων που του αναλογούν.
- ***Μέρος Γ:*** Αντιστοίχως με το μέρος β, στο μέρος Γ υλοποιείται ο VSYNC Driver και ο VPIXEL Controller. Ομοίως με το προηγούμενο μέρος, η πρώτη μονάδα οδηγεί το VSYNC σήμα με το κατάλληλο χρονισμό, ενώ η δεύτερη επιδεικνύει τη κάθετη διεύθυνση του εκάστοτε ενεργού pixel για τον λόγο που αναφέρθηκε προηγουμένως.
- ***Ένωση μερών****:* Το κομμάτι αυτό δεν ορίζεται επισήμως από την εκφώνηση της εργασίας, αλλά εμπεριέχεται στο μέρος Γ αυτής. Στη παρούσα τεχνική αναφορά θα αναφερθεί ως ξεχωριστό πλαίσιο κατά το οποίο τα μέρη Α,Β,Γ θα ενωθούν σε ένα ενιαίο συνθέσιμο σύστημα και θα αποτελέσουν τον VGA ελεγκτή.
- ***Συμπέρασμα****:* Εξαγωγή συμπεράσματος για το σύνολο των τμημάτων της εργασίας, τις δυσκολίες που προέκυψαν κατά τον σχεδιασμό αλλά και το τελικό αποτέλεσμα.

Στόχοι της 3ης εργασίας

Η τρίτη εργασία θέτει ως ζητούμενο την υλοποίηση ενός οδηγού θύρας οθόνης VGA (Video Graphics Array) για την οδήγηση μιας συμβατικής οθόνης. Στην οθόνη αυτή με την επιπρόσθετη κατασκευή μιας video ram θα εμφανίζεται μια στατική εικόνα, η οποία θα επαληθεύει τη σωστή υλοποίηση του ελεγκτή VGA.

Αντικείμενο της παρούσας εργαστηριακής άσκησης (τρίτη σε σειρά), είναι η δημιουργία του ελεγκτή VGA πάνω στη πλακέτα συν την κατασκευή μιας Video Ram που θα περιέχει την στατική εικόνα επαλήθευσης του ελεγκτή.

Για το πρώτο μέρος, σχεδιάζεται η Video Ram. Η Video Ram απαρτίζεται από τρεις Block Ram 16k x 1 της πλακέτας, κάθε μια από τις οποίες χρησιμοποιήθηκε για να αποθηκευτούν οι τιμές για καθένα από τα τρία R,G,B χρώματα των pixel της 128x96 ανάλυσης. Αυτή η ανάλυση αργότερα θα μεγεθύνεται από τον ελεγκτή στην επιθυμητή ανάλυση των 640x480 pixels.

Στο δεύτερο μέρος, κατασκευάστηκε η μονάδα του hsync driver και η μονάδα hpixel controller. Με τη πρώτη μονάδα, το σήμα hsync συγχρονίζεται καταλλήλως με τους χρόνους που επιδεικνύει η εκφώνηση αλλά και το τεχνικό δελτίο της πλακέτας για την επιλεγμένη ανάλυση και τον επιλεγμένο ρυθμό ανανέωσης. Η υλοποιημένη μονάδα του hpixel controller, όπως προαναφέρθηκε, έχει στόχο να επιδεικνύει στη Video Ram την οριζόντια διεύθυνση του ενεργού pixel. Επειδή οριζοντίως υπάρχουν 640 pixels και η Video Ram έχει μέγεθος 128x96 (1/5), η μονάδα επιδεικνύει κάθε pixel από 5 φορές για να πετύχει τη μεγέθυνση.

Στο τρίτο μέρος, αναλόγως υλοποιήθηκε η μονάδα του vsync driver και του vpixel controller. Συνεπώς, ο vsync driver οδηγεί το σήμα vsync παράλληλα με τους χρόνους που ορίζει η εκφώνηση και ο vpixel_controller ορίζει την κάθετη διεύθυνση του ενεργού pixel, κρατώντας ταυτόχρονα την κάθετη τιμή 5 φορές για να επιτευχθεί η μεγέθυνση από 96 κάθετα pixels σε 480.

Στο τελευταίο μέρος της εργαστηριακής άσκησης, γίνονται instantiate όλες οι προηγούμενες μονάδες στη συνθέσιμη μονάδα του vga_controller. Η μονάδα αυτή αποτελεί την τελική δομή του ελεγκτή και είναι αυτή που ελέγχεται στη πλακέτα για να επαληθευτεί η σωστή λειτουργία της. Ο ελεγκτής στην τελική υλοποίηση του, οδηγεί τα 2 σήματα των HSYNC και VSYNC και τα σήματα των τριών χρωμάτων VGA_RED, VGA_GREEN και VGA_BLUE.

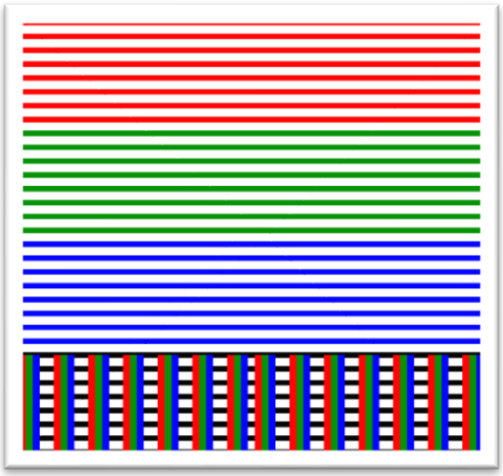
Μέρος Α – Υλοποίηση Video Ram

Στη Video Ram που σχεδιάστηκε, τρεις δομές Block Ram της πλακέτας απαρτίζουν το σύνολο της μνήμης στο οποίο είναι αποθηκευμένη μια στατική εικόνα ελέγχου. Η στατική εικόνα που επιλέχθηκε ως το πλαίσιο που επαληθεύει τη λειτουργία του οδηγού θύρας VGA, είναι επακριβώς η εικόνα που δίνεται από την εκφώνηση της άσκησης.

Η λογική της υλοποίησης της Video Ram έχει ως βάση της, την αποθήκευση των χρωμάτων της εικόνας-ελέγχου σε κάθε μια από τις τρεις block ram. Συγκεκριμένα, η πρώτη block ram περιέχει την τιμή του κόκκινου για κάθε κωδικοποίηση χρώματος RGB των pixel της εικόνας και οι επόμενες 2 περιγράφουν αντιστοίχως τις τιμές του πράσινου και του μπλε χρώματος των pixels.

Τα χαρακτηριστικά των Block Rams που χρησιμοποιήθηκαν, είναι ότι αποτελούν μνήμες 16k x 1 bit, είναι έτοιμα templates και πάρθηκαν από το εργαλείο ISE και ειδικότερα από την επιλογή Language Templates.

Η στατική εικόνα που αποθηκεύεται στην Video Ram έχει μέγεθος 128x96 pixels και πρόκειται να μεγεθυνθεί στην επιθυμητή ανάλυση των 640x480 pixels μέσα από τη δομή του ελεγκτή VGA.



Υλοποίηση:

Η μονάδα της Video Ram ως προς τον κώδικα Verilog που τη διέπει είναι αρκετά απλή και επαναλαμβανόμενη.

Ο τρόπος αποθήκευσης κάθε τιμής χρώματος των pixels γίνεται θεωρώντας κάθε block ram ως δομή «μονοδιάστατου πίνακα». Έτσι βλέποντας αναλόγως την ανάλυση 128x96 ως δομή «δισδιάστατου πίνακα», κάθε διεύθυνση pixel διαστήματος (0-95, 0-127), μεταφράζεται σε διεύθυνση διαστήματος (0-(128x96)) = (0-12287). Αυτό γίνεται εφικτό χρησιμοποιώντας καταλλήλως μια assign εντολή για τη θέση-index κάθε pixel μέσα στις Blocks Rams.

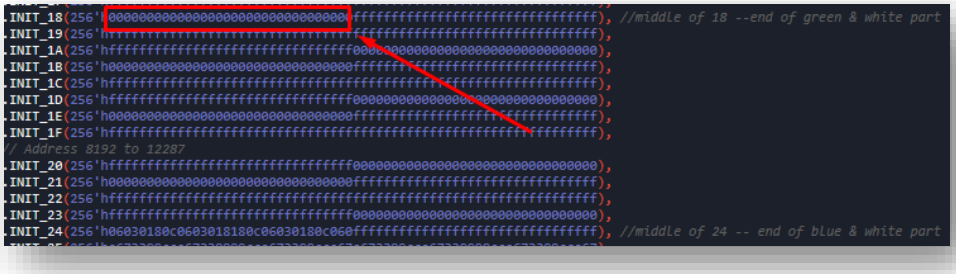
Η εντολή αυτή έχει ως εξής:

```
//width of screen 128x96
parameter videoRamWidth  = 14'd128;
//convert index of 2-D array to index of 1-D array
assign index = ( ver_px * videoRamWidth ) + hor_px;
```

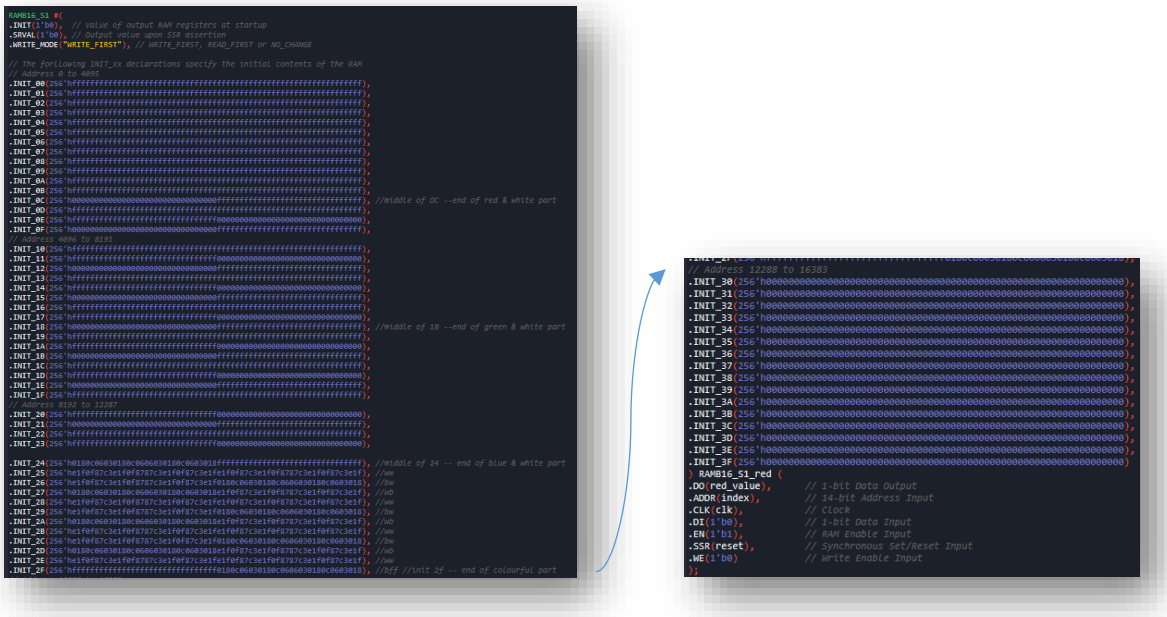
Το επόμενο στάδιο της υλοποίησης αφορά την ένθεση των τιμών των χρωμάτων στις Block Rams. Η αρχικοποίηση των Block Rams γίνεται μέσω των .INIT_xx τιμών, τα οποία προσφέρονται προς ανάθεση από το instantiation της μνήμης σε επίπεδο Verilog.

Κάθε .INIT_XX τιμή της Block Ram περιγράφει 256 bits και έχει διεύθυνση που είναι κωδικοποιημένη σε 64-bit στο δεκαεξαδικό σύστημα. Αυτό συνεπάγεται πως κάθε δεκαεξαδικό ψηφίο της 64-bit δεκαεξαδικής διεύθυνσης αντιστοιχεί σε 4 bits – pixels. Έτσι σε κάθε .INIT_XX παράμετρο αρχικοποίησης, αναθέτουμε τη κωδικοποίηση των χρωμάτων που αντιστοιχεί σε 2 γραμμές (128 bit η μια γραμμή, 64 x 4 = 256 => 2 γραμμές συνολικά)).

Παραδείγματος χάρη, για την τιμή της πρώτης μπλε γραμμής στη Block Ram του πράσινου χρώματος στο R,G,B (η πρώτη μπλε γραμμή από πάνω προς τα κάτω όπως φαίνεται στην δοσμένη εικόνα παραπάνω), αναθέτουμε 32 bit μηδενικά (μπλε χρώμα: R = 0, G = 0, B = 1) που αντιστοιχούν σε 128 pixels.



Η διαδικασία που ακολουθείται είναι η παραπάνω για κάθε γραμμή της ανάλυσης 128x96 και η τιμή της INIT_XX παραμέτρου αλλάζει αναλόγως το χρώμα ή τα χρώματα που αντιστοιχούν στις εκάστοτε γραμμές. Για λόγους πληρότητας δίδεται παρακάτω ολόκληρο το instantiation της Block Ram του κόκκινου χρώματος στη Verilog, ενώ οι άλλες δυο μνήμες αρχικοποιούνται και γίνονται instantiate αναλόγως.



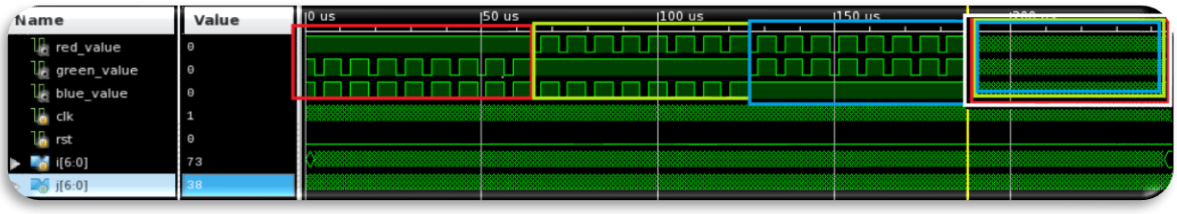
Επαλήθευση:

Για την επαλήθευση της μονάδας της Video Ram κατασκευάζουμε το κατάλληλο πλαίσιο δοκιμής που επαληθεύει πως για κάθε εισερχόμενη διεύθυνση της ανάλυσης 128x96, η μνήμη εξάγει τη σωστή κωδικοποίηση χρωμάτων R,G,B για το εκάστοτε δοσμένο pixel. Συνεπώς με τη προώθηση κάθε δυνατής διεύθυνσης pixel, από τη (0,0) μέχρι και την (95, 127) και εφόσον η Video Ram λειτουργεί ορθώς, τα pixel της στατικής εικόνας της εκφώνησης θα έχουν τα σωστά και ακριβή χρώματα.

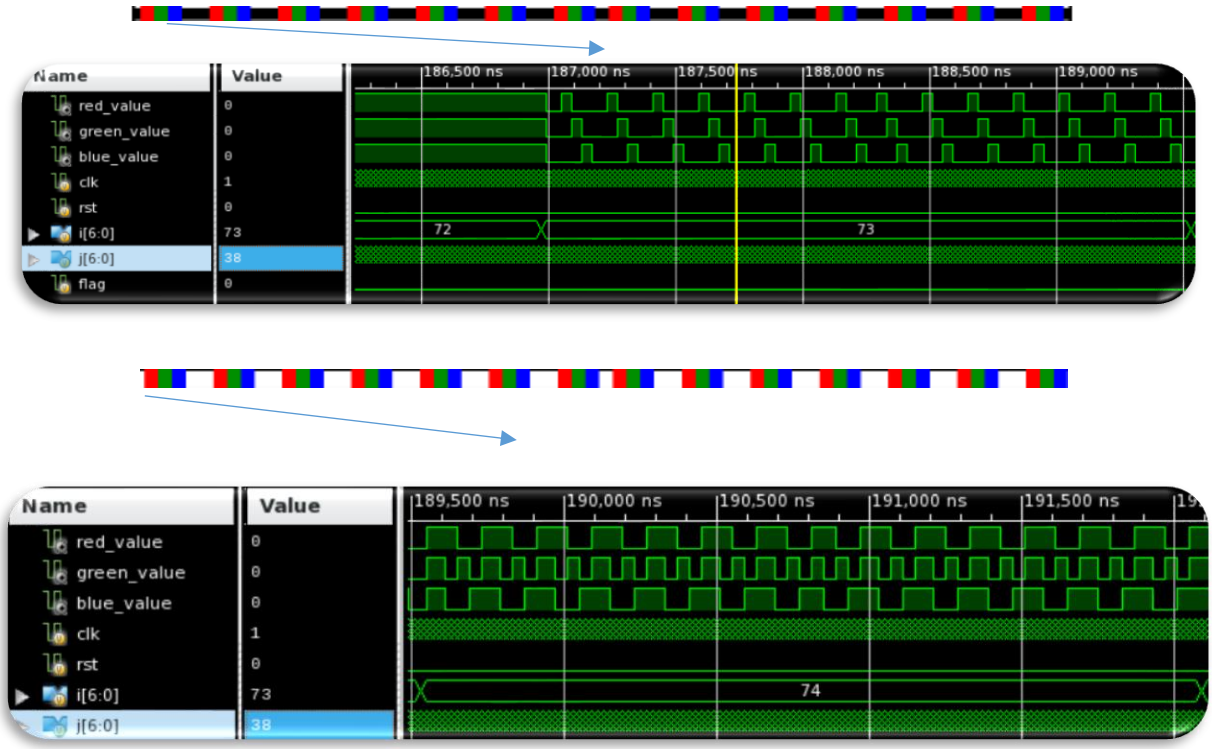
```
module video_ram_tb ();

reg clk = 0;
reg rst = 1;
reg [6:0] i;
reg [6:0] j;
reg flag;

//Initialize cirquit at a random moment (after 74ns)

initial begin
    #74 rst=0;
end

//Fpga spartan 3 pixel clock frequency ==> 50 Mhz

always #10 clk <= ~clk;

//Test every pixel of 128x96 screen.

always @(posedge clk or posedge rst) begin
    if (rst) begin
        // reset
        i = 0;
        j = 0;
        flag = 1'b1;
    end
    else if ( flag == 2'b1 ) begin
        flag = 1'b0;
    end
    else if (i == 7'd95 && j == 7'd127 ) begin
        i = 0;
        j = 0;
        $finish;
    end
    else if ( j == 7'd127 ) begin
        i = i + 1;
        j = 0;
    end
    else begin
        j = j + 1;
    end
end

//instantiate video ram to extract colours for every pixel

video_ram video_ram_0 (.reset(rst),.clk(clk), .ver_px(i), .hor_px(j),
                       .red_value(red_value), .green_value(green_value), .blue_value(blue_value) );

endmodule
```

Το πλαίσιο δοκιμής σε κώδικα Verilog δίνεται στην εικόνα παραπάνω. Στο always block που διακρίνεται, το i αντιστοιχεί στην κάθετη διεύθυνση (0-95) και το j στην οριζόντια (0-127). Έτσι σε κάθε κύκλο, στο instantiation της Video Ram εισέρχεται η διεύθυνση (i,j) μέχρι αυτή να φτάσει την τιμή (95,127) και άρα την εντολή $finish, γεγονός που δηλώνει ότι και οι 12288 (128x96=12288) πιθανές τιμές έχουν ελεγχθεί.

Κυματομορφές: Οι κυματομορφές που προκύπτουν μετά την προσομοίωση συμπεριφοράς, εξάγουν το συμπέρασμα πως η Video Ram εξάγει τη σωστή κωδικοποίηση χρωμάτων για κάθε δοσμένη διεύθυνση. Ένα στιγμιότυπο σε μέγιστη θέα δίνεται παρακάτω.



Στο δοσμένο στιγμιότυπο είναι εύκολα διακριτό πως αρχικά εμφανίζεται σωστά το πρώτο μέρος της εικόνας με τις εναλλάξ κόκκινες και άσπρες γραμμές (κόκκινο: RGB (1,0,0), άσπρο: RGB (1,1,1)). Στη συνέχεια δίνεται το δεύτερο μέρος με τις εναλλάξ πράσινες και άσπρες γραμμές (πράσινο: RGB (0,1,0)) και ακολουθεί το τρίτο μέρος με τις εναλλάξ μπλε και άσπρες γραμμές (μπλε: RGB (0,0,1)). Τέλος, στο τελευταίο κουτί δεξιά, επισημαίνεται το πολύχρωμο μέρος της στατικής εικόνας. Το τελευταίο μέρος που αποτελείται από εναλλάξ μαύρες και άσπρες γραμμές, στις οποίες βρίσκονται κάθετες τρίχρωμες στήλες, μπορεί να φανεί καλύτερα και σε μεγέθυνση παρακάτω.



Κάλυψη λειτουργίας: Τα διανύσματα ελέγχου που παρέχονται από το πλαίσιο ελέγχου στη μονάδα της Video Ram, αντιστοιχούν σε όλες τις δυνατές διευθύνσεις της ανάλυσης 128x96 pixels.Κατά συνέπεια, κάθε pixel της στατικής εικόνας ελέγχεται ως προς τη κωδικοποίηση των χρωμάτων του και η κάλυψη της λειτουργίας της μονάδας φτάνει στο μέγιστο.

Μέρος Β – Υλοποίηση HSYNC και οριζοντίου μετρητή Pixel

Το σήμα HSYNC πρέπει να οδηγείται στους χρόνους που ορίζει η ανάλυση 640 x 480 και ο ρυθμός ανανέωσης των 60 Hz. Παράλληλα ο μετρητής που παρέχει την οριζόντια διεύθυνση του εκάστοτε ενεργού pixel στη Video Ram είναι απαραίτητο να βρίσκεται σε συγχρονισμό με το HSYNC.

Στο μέρος Β της εργασίας, υλοποιούνται οι μονάδες **hsync_driver** και **hpixel_controller.**

- Βασικός ρόλος της μονάδας του hsync_driver είναι να οδηγεί το σήμα HSYNC καταλλήλως στους χρόνους που απαιτούνται. Πέρα από το HSYNC, η μονάδα οδηγεί στην έξοδο ένα σήμα (βλέπε παρακάτω σήμα new_line) που δηλώνει τον χρόνο απεικόνισης της γραμμής ή αλλιώς το χρόνο στον οποίον κάθε επόμενη οριζόντια γραμμή είναι ενεργή.

- Από την άλλη μεριά, η μονάδα hpixel_controller δεχόμενη το παραπάνω σήμα (new_line), συγχρονίζει την οριζόντια διεύθυνση του εκάστοτε ενεργού pixel, με τρόπο ώστε να στέλνεται κάθε φορά που το σήμα οδηγείται στο 1. Επειδή ο χρόνος σάρωσης αντιστοιχεί σε 1280 κύκλους ρολογιού (Χρόνος ενεργής γραμμής = 25.6 μsec, Ρολόι Πλακέτας 50 MHz), οι 640 οριζόντιες διευθύνσεις pixel πρέπει να σταλούν ομοιόμορφα, δηλαδή μια διεύθυνση **ανά 2 κύκλους.**

  Επιπρόσθετα, οι 640 οριζόντιες διευθύνσεις πρέπει να αντιστοιχιστούν επακριβώς με τις διευθύνσεις της αποθηκευμένης εικόνας στη Video Ram ανάλυσης 128x96. Για την αντιστοίχιση αυτή, δεδομένου ότι 128 = (1/5) *(640), ο hpixel_controller «μεγεθύνει» την εικόνα της Video Ram στέλνοντας την εκάστοτε διεύθυνση του διαστήματος (0-127) συνολικά **5 φορές.**

  Συνδυάζοντας τα παραπάνω πορίσματα, προκύπτει το γεγονός ότι η οριζόντια διεύθυνση που στέλνει στην έξοδο ο hpixel_controller αλλάζει την τιμή της κάθε **10 κύκλους** στο χρονικό διάστημα των 25.6 μsec της απεικόνισης γραμμής.

Υλοποίηση:

hsync_driver module: Σε πρώτο στάδιο εστιάζουμε στη μονάδα του hsync_driver. Η μονάδα αυτή χρησιμοποιεί κατάλληλους μετρητές για να ορίσει τους τους χρόνους που απαιτεί το HSYNC σήμα για τα χαρακτηριστικά οθόνης 640x480, 60 Hz.

Οι μετρητές της μονάδας και ο αντίστοιχος χρόνος που μετράνε όταν φτάσουν σε μια ορισμένη τιμή έχουν ως εξής:

	scanline_ counter	hsync_palm_counter	back_porch_counter	active_line_counter
Διάστημα που ορίζει	Σάρωση γραμμής	Πλάτος παλμού HSYNC	Πίσω όψη	Χρόνος απεικόνισης γραμμής
Χρόνος σε μsec	32 μsec	3.84 μsec	1.92 μsec	25.6 μsec
Χρόνος σε κύκλους ρολογιού 50 MHz – Μέγιστη τιμή μετρητή	1600	192	96	1280

1ο τμήμα always block (hsync_driver.v): Το πρώτο από τα 3 τμήμα always block της μονάδας διαχειρίζεται τους μετρητές scanline_counter και hsync_palm_counter. Ο πρώτος μετρητής κάθε φορά που φτάνει στη μέγιστη τιμή του, οδηγεί το σήμα HSYNC στο 0 και παράλληλα ο δεύτερος ορίζει αντίστοιχα το χρονικό διάστημα που θα διατηρεί ο παλμός τη τιμή πριν ξανά οδηγηθεί και πάλι στο 1. Οι αντίστοιχοι χρόνοι για κάθε μετρητή παρατίθενται στο πίνακα παραπάνω, ενώ το always block φαίνεται στο στιγμιότυπο παρακάτω.

```
//Manage hsync signal by controlling scanline time period = 32 usec after next edge.
//Also hold hsync palm active for 3.84 usec with hsync_palm_counter

always @(posedge clk or posedge reset) begin
    if (reset) begin
        // reset
        hsync = 1'b1;
        hsync_palm_counter = 8'd0;
        scanline_counter = 11'd0;
    end
    //scanLine counter will be reset in next cycle so we need 1599 cycles in order to achive exactly 32 usec time period
    else if ( scanline_counter == 11'd1599 ) begin
        hsync = 1'b0;
        hsync_palm_counter = hsync_palm_counter + 1;
        scanline_counter = 11'd0;
    end
    else if ( hsync_palm_counter == 8'd192 ) begin
        hsync = 1'b1;
        hsync_palm_counter = 8'd0;
        scanline_counter = scanline_counter + 1;
    end
    else if ( hsync == 1'b0 ) begin
        hsync_palm_counter = hsync_palm_counter + 1;
        scanline_counter = scanline_counter + 1;
    end
    else begin
        scanline_counter = scanline_counter + 1;
    end
end
```

2ο τμήμα always block (hsync_driver.v): Στο δεύτερο τμήμα always, ο μετρητής back_porch_counter μετρά τον χρόνο της πίσω όψης που αντιστοιχεί στο διάστημα μετάβασης από την οδήγηση του σήματος HSYNC στο 1 (αφού πριν ήταν στο 0) μέχρι τη στιγμή που ξεκινά ο χρόνος σάρωσης γραμμής. Ένα σήμα back_porch_enable οδηγημένο στο 1, υποδηλώνει την έναρξη του μετρητή και άρα του διαστήματος της πίσω όψης.

```
//In this always block we manage back portch duration before screen go active
//Back portch counter counts 1.92 usec back portch time

always @( posedge clk or posedge reset ) begin
    if (reset) begin
        // reset
        back_porch_counter = 7'd0;
        back_porch_enable = 1'b0;
    end
    else if ( back_porch_counter == 7'd97 ) begin
        back_porch_counter = 0;
        back_porch_enable= 1'b0;
    end
    else if ( ( hsync == 1'b1  && back_porch_enable == 1'b1 )  ) begin
        back_porch_counter = back_porch_counter + 1;
        back_porch_enable = 1'b1;
    end

    else if ( back_porch_enable == 1'b1 ) begin
        back_porch_counter = back_porch_counter + 1;
    end
    else if ( hsync_palm_counter == 8'd192 ) begin
        back_porch_enable = 1'b1;
    end
end
```

3ο τμήμα always block (hsync_driver.v): Στο τελευταίο τμήμα always, με το που διαπιστωθεί ότι το διάστημα της πίσω όψης ολοκληρωθεί, το σήμα new_line οδηγείται στο 1 και υποδηλώνει ότι γραμμή ξεκινά να σαρώνεται. Ο μετρητής active_line_counter μετρά το χρόνο που αντιστοιχεί στο διάστημα σάρωσης γραμμής (βλέπε πίνακα παραπάνω) και όταν φτάσει στην μέγιστη τιμή του άρα και το τέλος της σάρωσης, οδηγεί το σήμα new_line στο 0.

```
//In this always block we manipulate for how long the line will retain in its active state
//A signal new_line indicates if screen is off or on.
//Max value of active line counter indicates the start of the front_portch duration (time before next hsync_palm)

always @(posedge clk or posedge reset) begin
    if (reset) begin
        // reset
        active_line_counter = 11'd0;
        new_line = 1'b0;
    end
    else if ( active_line_counter == 11'd1279 ) begin
        active_line_counter = 11'd0;
        new_line = 1'b0;
    end
    else if ( back_porch_counter == 7'd97 ) begin
        new_line = 1'b1;
    end
    else if ( new_line == 1'b1 ) begin
        active_line_counter = active_line_counter + 1;
    end
end

endmodule
```

hpixel_controller module: Το μέρος Β συνεχίζει με τον hpixel_controller. Ο ελεγκτής δέχεται ως είσοδο το σήμα new_line από τον hsync_driver (περιεγράφηκε προηγουμένως) και ορίζει κατάλληλα την οριζόντια διεύθυνση στο χρονικό διάστημα που ορίζει η σάρωση της γραμμής.

Μηχανή πεπερασμένων καταστάσεων του ελεγκτή(hpixel_controller.v): Όπως ειπώθηκε στην εισαγωγική περιγραφή του μέρους Β, ο χρόνος απεικόνισης γραμμής αντιστοιχεί σε 25.6 usec που μεταφράζεται σε 1280 κύκλους ρολογιού συχνότητας 50 MHz. Για να σταλθούν κατάλληλα αλλά και ομοιόμορφα σε όλο το διάστημα της σάρωσης οι 640 διευθύνσεις, η μονάδα απαρτίζεται από 2 καταστάσεις STATE_set και STATE_wait:

- ***STATE_wait***: Η μονάδα, εφόσον βρίσκεται στο χρονικό διάστημα της σάρωσης γραμμής, περιμένει ένα κύκλο χωρίς να πραγματοποιεί κάποια ενέργεια και στη συνέχεια μεταβαίνει στη κατάσταση STATE_set. Εκτός διαστήματος σάρωσης, το κύκλωμα βρίσκεται πάντα στη κατάσταση STATE_wait.
- ***STATE_set***: Η μονάδα οδηγεί στο 1 ένα σήμα scale_done, το οποίο υποδηλώνει στο always block που παρουσιάζεται στην επομένη παράγραφο, είτε να στείλει την επομένη οριζόντια διεύθυνση είτε να αυξήσει τον μετρητή που αναλαμβάνει την μεγέθυνση των 128 pixels σε 640 (αναλυτικά στην επόμενη παράγραφο).

Η μηχανή των καταστάσεων δίνεται στο παρακάτω στιγμιότυπο:

```
//FSM of 2 states for hpixel_controller

always @(posedge clk or posedge reset) begin
    if (reset) begin
        // reset
        state = STATE_wait;
    end
    else begin
        state = next_state;
    end
end

//States management
//Scan-Line time consists of 1280 cycles. Therefore, 640 pixels should be sent after every two clock cycles.

always @(*) begin
    next_state = state;
    scale_done = 1'b0;
    case (state)
        STATE_set: begin // Assign address of horizontal pixel
            if ( new_line == 1 ) begin
                next_state = STATE_wait;
                scale_done = 1'b1;
            end
        end
        STATE_wait: begin //Wait one cycle before next asssign.
            if ( new_line == 1 ) begin
                next_state = STATE_set;
            end
        end
    endcase
end
```

Τμήματα always για την διαχείριση της οριζόντιας διεύθυνσης (hpixel_controller.v): Όταν το κύκλωμα βρίσκεται στη κατάσταση STATE_set, ένας μετρητής scaleTo640_counter αυξάνεται κάθε φορά κατά 1 μέχρι την μέγιστη τιμή 5. Όταν ο μετρητής φτάσει στην τιμή 5, μέσα από το δεύτερο always block στέλνεται η επόμενη οριζόντια διεύθυνση pixel. Η διαδικασία αυτή έχει ως σκοπό να στέλνεται το ίδιο pixel 5 φορές, με συνέπεια τα 128 pixels να αντιστοιχούν ή αλλιώς να «μεγεθύνονται» σε 640. Τέλος ένα σήμα end_of_line οδηγείται στο 1 κάθε φορά που και τα 640 pixels έχουν σταλθεί και συνεπώς η διαδικασία πρέπει να μηδενίσει τους μετρητές.

```
always @(posedge clk or posedge reset) begin
    if (reset) begin
        // reset
        scaleTo640_counter = 3'b0;
        end_of_line = 1'b0;
    end
    else if ( new_line == 1'b0 ) begin
        end_of_line = 1'b0;
    end
    else if ( hpixel == 7'd127 && scale_done == 1'b1 && scaleTo640_counter == 3'd4 ) begin
        scaleTo640_counter = 3'd0;
        end_of_line = 1'b1; //end of the line indicates that the final horizontal pixel was sent to output.
    end
    else if ( scale_done == 1'b1 && scaleTo640_counter == 3'd4 ) begin
        scaleTo640_counter = 3'd0;
    end
    else if ( scale_done == 1'b1 ) begin
        scaleTo640_counter = scaleTo640_counter + 1;
    end
end

//Increment horizontal pixel when new_line is active and according to scaleTo640_counter value

always @(posedge clk or posedge reset) begin
    if (reset) begin
        // reset
        hpixel = 7'd0;
    end
    else if ( hpixel == 7'd127 && end_of_line == 1'b1 ) begin
        hpixel = 7'd0;
    end
    else if ( scaleTo640_counter == 3'd0 && scale_done == 1'b1 ) begin
        //after scaleTo640_counter indicates that same pixel was sent for five times, next pixel sould be sent.
        hpixel = hpixel + 1;
    end
end
```

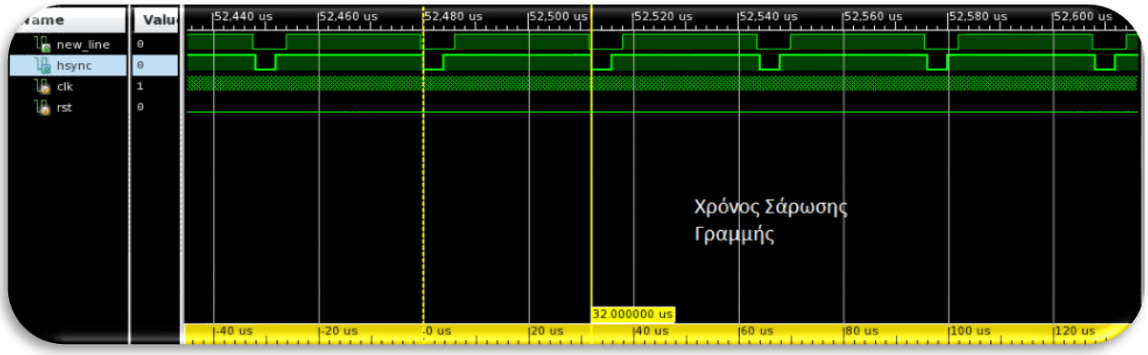
Επαλήθευση:

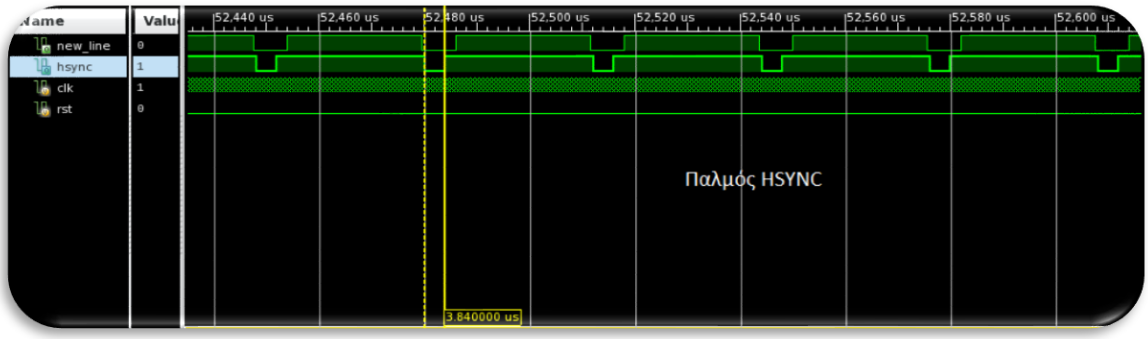
Για την επαλήθευση του μέρους Β, ένα απλό στη δομή του πλαίσιο ελέγχου κάνει instantiate τις παραπάνω μονάδες και συνδέει τον hpixel_controller με τον hsync_driver μέσω του κοινού σήματος new_line. Μέσω της δοκιμής αυτής είναι εφικτό να επαληθευτούν τόσο οι χρόνοι γύρω από το σήμα HSYNC, όσο και οι οριζόντιες διευθύνσεις.

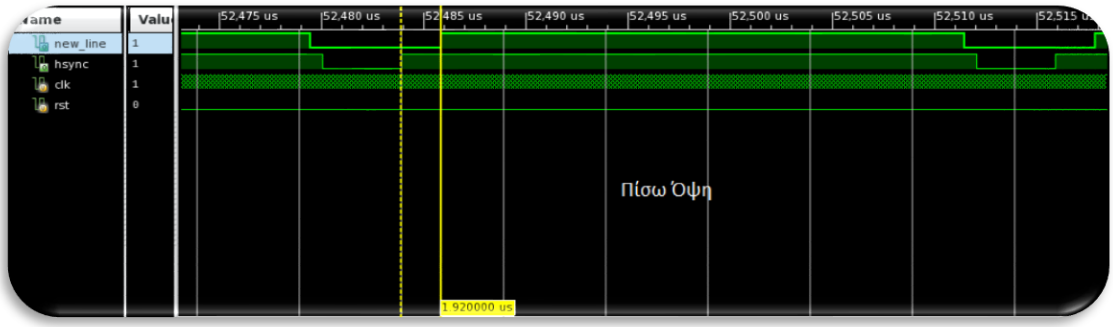
Το πλαίσιο ελέγχου δίνεται παρακάτω σε κώδικα Verilog:

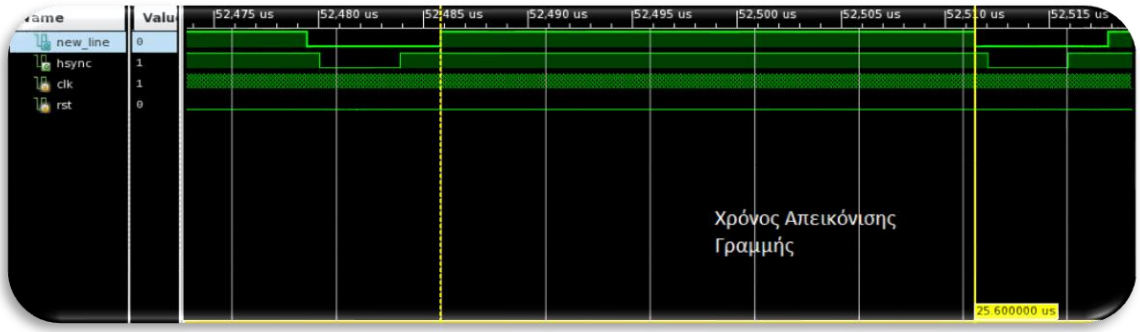
```
module hsync_driver_tb ();

reg clk = 0;
reg rst = 1;

//Initialize cirquit at a random moment (after 74ns)

initial begin
    #74 rst=0;
end

//Fpga spartan 3 clock frequency ==> 50 Mhz

always #10 clk <= ~clk;

hsync_driver hsync_driver_0 (.reset(rst), .clk(clk), .hsync(hsync), .new_line(new_line));

wire [6:0] hpixel;

hpixel_controller hpixel_controller_0 (.reset(rst), .clk(clk), .new_line(new_line), .hpixel(hpixel) );
endmodule
```

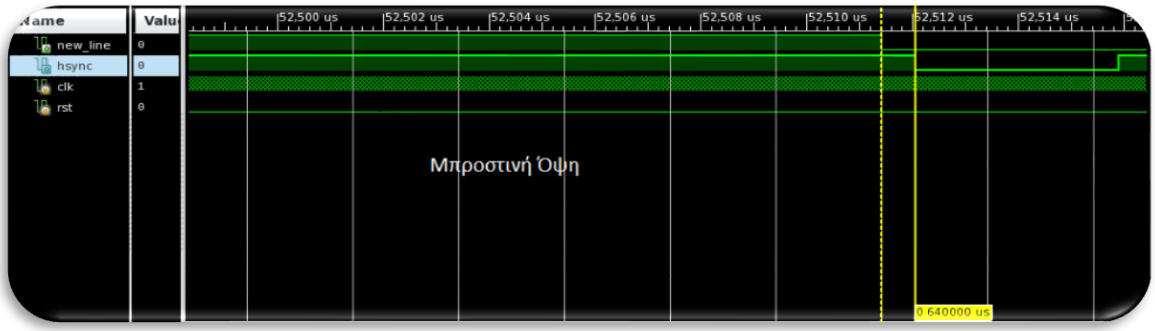
Κυματομορφές: Οι κυματομορφές της προσομοίωσης επαληθεύουν ότι οι χρόνοι γύρω από το HSYNC είναι σωστοί και ακριβείς.



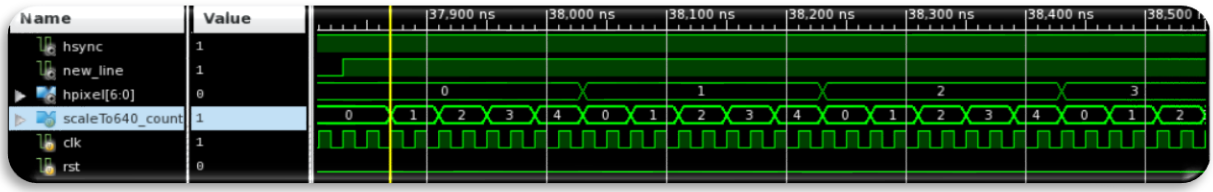








Όσον αφορά τον hpixel_controller μπορεί να φανεί διαισθητικά ότι οι οριζόντιες διευθύνσεις στέλνονται σωστά. Κάθε διεύθυνση αλλάζει ορθώς μετά από 5 τιμές του scaleTo640_counter και συνολικά κάθε 10 κύκλους.



Κάλυψη λειτουργίας: Ως προς τη κάλυψη λειτουργίας του κυκλώματος, το πλαίσιο ελέγχου καλύπτει στο μέγιστο τις 2 μονάδες, διότι τα διανύσματα εισόδου που είναι απαραίτητα για τη λειτουργία είναι μόνο το σήμα reset και το ρολόι. Για τον hpixel_controller σήμα εισόδου είναι και το new_line, το οποίο όμως λαμβάνει από τον hsync_driver.

Μέρος Γ – Υλοποίηση VSYNC και Κάθετου Μετρητή Pixel

> *Όπως και με το HSYNC έτσι και το VSYNC πρέπει να οδηγείται κατάλληλα στους χρόνους που ορίζει η ανάλυση 640x480 pixels και ο ρυθμός ανανέωσης των 60 Hz. Ταυτόχρονα ο μετρητής της κάθετης διεύθυνσης pixel πρέπει να είναι συγχρονισμένος με το VSYNC.*

Σε αντιστοιχία με το μέρος Β, στο μέρος Γ υλοποιούνται οι μονάδες **vsync_driver** και **vpixel_controller.**

- Όπως λειτουργεί ο hsync_driver για το σήμα HSYNC, έτσι και ο vsync_driver οδηγεί κατάλληλα το αντίστοιχο σήμα VSYNC, συγκεκριμένα συγχρονισμένο κατάλληλα στους χρόνους που απαιτεί η επιλεγμένη ανάλυση. Ο vsync_driver εκτός από το σήμα VSYNC οδηγεί στην έξοδο και το σήμα new_frame, το οποίο υποδηλώνει, όσο είναι στο 1, τον ενεργό χρόνο απεικόνισης ενός frame.

- Η μονάδα του vpixel_controller διαχειρίζεται την κάθετη διεύθυνση του εκάστοτε ενεργού pixel. Οι διευθύνσεις πρέπει να είναι σε αριθμό 480, παρόλα αυτά λειτουργώντας αναλόγως με τον hpixel_controller, η μονάδα στέλνει την ίδια διεύθυνση 5 φορές. Με αυτόν το τρόπο οι 96 διευθύνσεις της Video Ram επί 5 φορές η καθεμία, μεγεθύνονται σε 480 (5x96 = 480). Τέλος, ο vpixel_controller για να εκτελέσει σωστά τη λειτουργία του, έχει ως είσοδο τόσο το σήμα new_line του hsync_driver όσο και το σήμα new_frame του vsync_driver.

Υλοποίηση:

vsync_driver module: Αρχικά εστιάζουμε στη μονάδα του vsync_driver. Όπως και με τον hsync driver έτσι και εδώ, η μονάδα χρησιμοποιεί κατάλληλους μετρητές για να πετύχει τα χρονικά διαστήματα που απαιτεί η επιλεγμένη ανάλυση για το σήμα VSYNC.

Οι μετρητές της μονάδας περιγράφονται στον πίνακα που ακολουθεί.

	frame_ counter	**vsync_palm_counter**	**back_porch_counter**	**active_frame_counter**	first_time_counter
Διάστημα που ορίζει	Συνολικός χρόνος εικόνας	Πλάτος παλμού VSYNC	Πίσω όψη	Χρόνος απεικόνισης frame	Οδήγηση του VSYNC τη πρώτη φορά για συγχρονισμό με το HSYNC
Χρόνος σε μsec ή msec	16.672 msec	64 μsec	928 μsec	15.36 msec	5.74 μsec
Χρόνος σε κύκλους ρολογιού 50 MHz – Μέγιστη τιμή μετρητή	833500	3200	46400	768000	287

1ο τμήμα always block (vsync_driver.v): Το πρώτο από τα 3 τμήμα always block της μονάδας έχει παρόμοια υλοποίηση με το αντίστοιχο του hsync_driver με μια εξαίρεση. Για να συγχρονιστεί το vsync κατάλληλα με το hsync και συγκεκριμένα ο χρόνος της πρώτης ενεργής γραμμής με τον χρόνο που ξεκινά το ενεργό frame, προστίθεται στην υλοποίηση ο first_time_counter και το σήμα first_time_flag που υποδεικνύουν την πρώτη και μόνο αυτή οδήγηση του VSYNC στο 0.

```
always @(posedge clk or posedge reset) begin
    if (reset) begin
        // reset
        vsync = 1'b1;
        vsync_palm_counter = 12'd0;
        frame_counter = 20'd0;
        first_time_flag = 1'b1;
        first_time_counter = 9'd0;
    end
    //after exception of first time period. drive vsync to zero and start main process.
    else if ( first_time_counter == 9'd287 ) begin
        first_time_flag = 1'b0;
        vsync = 1'b0;
        vsync_palm_counter = vsync_palm_counter + 1;
        frame_counter = 20'd0;
        first_time_counter = 9'd0;
    end
    //frame counter will be reset in next cycle so we need 833599 cycles in order to achive exactly 32 usec time period
    else if ( frame_counter == 20'd833599 ) begin
        vsync = 1'b0;
        vsync_palm_counter = vsync_palm_counter + 1;
        frame_counter = 20'd0;
    end
    else if ( vsync_palm_counter == 12'd3200 ) begin
        vsync = 1'b1;
        vsync_palm_counter = 12'd0;
        frame_counter = frame_counter + 1;
    end
    else if ( vsync == 1'b0 ) begin
        vsync_palm_counter = vsync_palm_counter + 1;
        frame_counter = frame_counter + 1;
    end
    //First-time-counter counts a specific time period after reset in order vsync frame time to be synchronized with scanline time.
    else if ( first_time_flag == 1'b1 ) begin
        frame_counter = frame_counter + 1;
        first_time_counter = first_time_counter + 1;
    end
    else begin
        frame_counter = frame_counter + 1;
    end
end
```

2ο τμήμα always block (vsync_driver.v): Στο δεύτερο τμήμα always, η υλοποίηση είναι ίδια με την αντίστοιχη της μονάδας του hsync_driver και διαχειρίζεται το χρονικό διάστημα της πίσω όψης για VSYNC.

```
//In this always block we manage back portch duration before screen go active
//Back portch counter counts 928 usec back portch time

always @( posedge clk or posedge reset ) begin
    if (reset) begin
        // reset
        back_porch_counter = 16'd0;
        back_porch_enable = 1'b0;
    end
    else if ( back_porch_counter == 16'd46401 ) begin
        back_porch_counter = 0;
        back_porch_enable= 1'b0;
    end
    else if ( ( vsync == 1'b1 && back_porch_enable == 1'b1 )  ) begin
        back_porch_counter = back_porch_counter + 1;
        back_porch_enable = 1'b1;
    end

    else if ( back_porch_enable == 1'b1 ) begin
        back_porch_counter = back_porch_counter + 1;
    end
    else if ( vsync_palm_counter == 12'd3200 ) begin
        back_porch_enable = 1'b1;
    end
end
```

3ο τμήμα always block (vsync_driver.v): Στο τελευταίο τμήμα always και πάλι σε αντιστοιχία με τον hsync_driver και τον μετρητή της ενεργής γραμμής, έτσι και εδώ το block διαχειρίζεται τον μετρητή του χρονικό διαστήματος του ενεργού frame, οδηγώντας το αντίστοιχο σήμα new_frame.

```
always @(posedge clk or posedge reset) begin
    if (reset) begin
        // reset
        active_screen_counter = 20'd0;
        new_frame = 1'b0;
    end
    else if ( active_screen_counter == 20'd767999 ) begin
        active_screen_counter = 20'd0;
        new_frame = 1'b0;
    end
    else if ( back_porch_counter == 16'd46401 ) begin
        new_frame = 1'b1;
    end
    else if ( new_frame == 1'b1 ) begin
        active_screen_counter = active_screen_counter + 1;
    end
end

endmodule
```

vpixel_controller module: Η υλοποίηση του μέρους Γ ολοκληρώνεται με τον vpixel_controller. Ο ελεγκτής δέχεται ως είσοδο το σήμα new_line από τον hsync_driver και το σήμα new_frame από τον vsync_driver και ορίζει κατάλληλα την κάθετη διεύθυνση στο χρονικό διάστημα που ορίζει η ενεργή γραμμής και το ενεργό frame.

Μηχανή πεπερασμένων καταστάσεων του ελεγκτή(vpixel_controller.v): Η μηχανή πεπερασμένων καταστάσεων του vpixel_controller έχει ρόλο που διαφοροποιείται από την αντίστοιχη μηχανή του hpixel_controller. Η λογική της συγκεκριμένης μηχανής 3 καταστάσεων είναι να ορίζει πότε αλλάζει η διεύθυνση του κάθετου pixel ή απλά αυξάνεται ο μετρητής που κρατά 5 φορές την ίδια διεύθυνση για να μεγεθύνει τις 96 συνολικά διευθύνσεις σε 480.

- ***STATE_zero***: Η κάθετη διεύθυνση, με την έναρξη λειτουργίας της μονάδας, αρχικοποιείται στο 0. Επίσης τη πρώτη φορά που η πρώτη ενεργή γραμμή απεικονίζεται, η διεύθυνση του κάθετου pixel είναι όντως η μηδενική. Για το λόγο αυτό στο STATE_zero το κύκλωμα μένει χωρίς να αυξήσει κάποιο μετρητή, για κάθε ενεργό frame, τη φορά της πρώτης ενεργής γραμμής. Μόλις την ανιχνεύσει μεταβαίνει στη κατάσταση STATE_one.
- ***STATE_one***: Στη κατάσταση αυτή και όσο το κύκλωμα βρίσκεται ακόμα στο χρονικό διάστημα του ενεργού frame, η μονάδα παραμένει στάσιμη μέχρι να ανιχνεύσει τη στιγμή που ο χρόνος της ενεργής γραμμής θα γίνει 0. Έτσι με το που περάσει ο χρόνος της ενεργής γραμμής, στέλνει το σήμα scale_done = 1'b1 και μεταβαίνει στην επόμενη κατάσταση STATE_IDLE. Ουσιαστικά, κάθε νέα κάθετη διεύθυνση παίρνει τη τιμή της όσο η γραμμή είναι ανενεργή, έτσι τη στιγμή που το new_line γίνει και πάλι 1, το pixel έχει τη σωστή κάθετη διεύθυνση για όλες τις αντίστοιχες οριζόντιες της εκάστοτε γραμμής.
- ***STATE_IDLE:*** Όσο το new_line είναι 0 και το κύκλωμα βρίσκεται στη διάρκεια του ενεργού frame, η κάθετη διεύθυνση έχει πάρει τη σωστή τιμή της από τη κατάσταση STATE_one ένα κύκλο πριν μετέβη στη STATE_IDLE. Κατά συνέπεια, για το υπόλοιπο διάστημα που η γραμμή είναι ανενεργή, το κύκλωμα δεν πρέπει να πραγματοποιήσει περαιτέρω αύξηση του μετρητή. Οπότε, η κατάσταση περιμένει να ξαναγίνει το new_line ίσο 1 για να μεταβεί και πάλι στο STATE_one.

Η μηχανή των καταστάσεων δίνεται παρακάτω:

```
//FSM of 3 states for vpixel_controller

always @(posedge clk or posedge reset) begin
    if (reset) begin
        // reset
        state = STATE_zero;
    end
    else begin
        state = next_state;
    end
end

//States management

always @(*) begin
    next_state = state;
    scale_done = 1'b0;
    case (state)
        STATE_zero: begin //initialization state..
            if ( new_line == 1 && new_frame == 1) begin
                next_state = STATE_one;
            end
        end
        STATE_one: begin //when new_line is set to zero, send scale_done signal to indicate that scale_counter should increase.
            if ( new_line == 0 && new_frame == 1) begin
                scale_done = 1'b1;
                next_state = STATE_IDLE;
            end
        end
        STATE_IDLE: begin //controller should stay idle the whole time when new_line is 1 except the first_time (State_zero)
            if ( new_line == 1'b1 && new_frame == 1) begin
                next_state = STATE_one;
            end
        end
    endcase
end
```

Τμήμα always για την διαχείριση της κάθετης διεύθυνσης (vpixel_controller.v): Όταν το κύκλωμα βρίσκεται στη κατάσταση STATE_one, ένας σήμα scale_done γίνεται 1 για έναν κύκλο. Το σήμα αυτό υποδεικνύει στον scaleTo480_counter να αυξηθεί κατά 1, ενώ μπορεί να φτάσει μέχρι την μέγιστη τιμή 5. Όταν πάρει την τιμή αυτή, ένα if else στο always block κάνει την κάθετη διεύθυνση να αυξηθεί. Με τον τρόπο αυτό, γίνεται η μεγέθυνση των 96 κάθετων διευθύνσεων σε 480 στο ενεργό διάστημα του συνολικού χρόνου απεικόνισης του εκάστοτε frame. Τέλος μόλις σταλθούν όλες οι διευθύνσεις, οι μετρητές μηδενίζονται και πάλι.

```
//scaleTo480 counter forces vpixel to keep its value five times of new_line signal being active.

always @(posedge clk or posedge reset) begin
    if (reset) begin
        // reset
        scaleTo480_counter = 3'b0;
        vpixel = 7'd0;
    end
    else if ( vpixel == 7'd95 && scaleTo480_counter == 3'd4 && scale_done == 1'b1 ) begin
        vpixel = 7'd0;
        scaleTo480_counter = 3'd0;
    end
    else if ( scaleTo480_counter == 3'd4 && scale_done == 1'b1 ) begin
        scaleTo480_counter = 3'd0;
        vpixel = vpixel + 1;
    end
    else if ( scale_done == 1'b1 ) begin
        scaleTo480_counter = scaleTo480_counter + 1;
    end
end
```

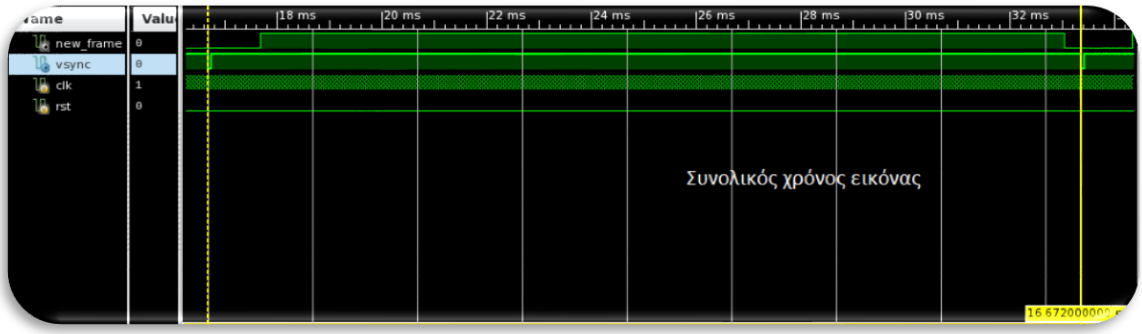
Επαλήθευση:

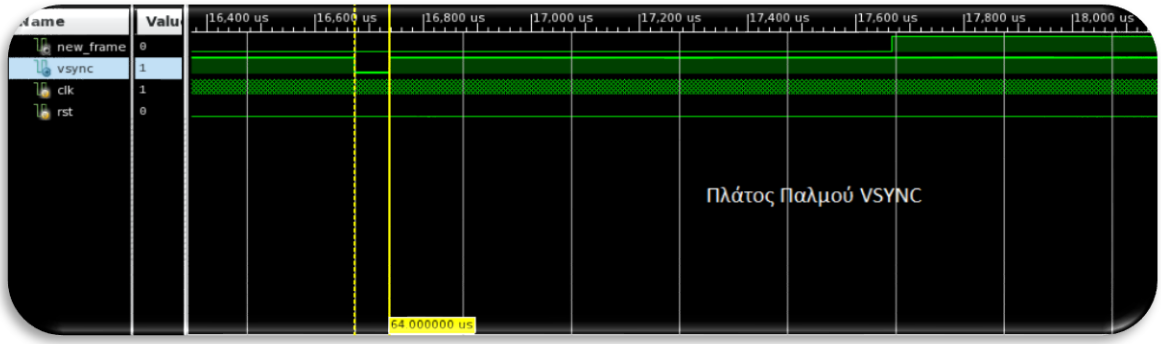
Για την επαλήθευση του μέρους Γ, και πάλι ένα απλό στη δομή του πλαίσιο ελέγχου κάνει instantiate τις μονάδες hsync_driver και vsync_driver και τις συνδέει με τον vpixel_controller μέσω των κοινών σημάτων new_line και new_frame. Μέσω της δοκιμής αυτής είναι εφικτό να επαληθευτούν τόσο οι χρόνοι γύρω από το σήμα VSYNC, όσο και οι κάθετες διευθύνσεις.

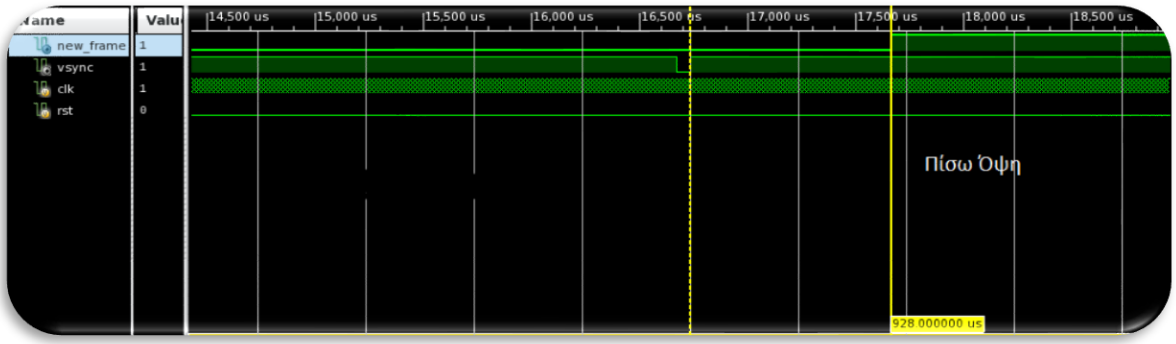
Το πλαίσιο ελέγχου δίνεται παρακάτω σε κώδικα Verilog:

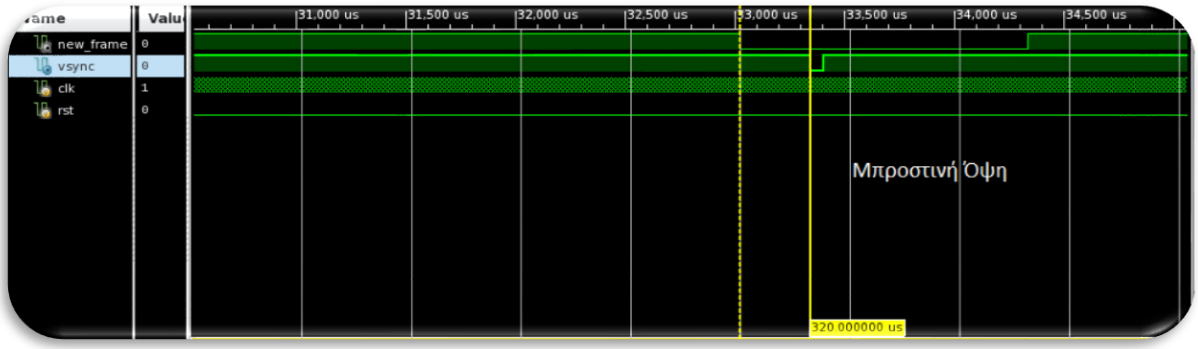
```
module vsync_driver_tb ();

reg clk = 0;
reg rst = 1;
wire new_line;
wire new_frame;
wire [6:0] vpixel;

//Initialize cirquit at a random moment (after 74ns)

initial begin
    #74 rst=0;
end

//Fpga spartan 3 clock frequency ==> 50 Mhz

always #10 clk <= ~clk;


//Instantiate hsync, vsync driver to extract new_line and new_frame signal.

hsync_driver hsync_driver_0 (.reset(rst), .clk(clk), .hsync(hsync), .new_line(new_line));
vsync_driver vsync_driver_0 (.reset(rst), .clk(clk), .vsync(vsync), .new_frame(new_frame));

//Instantiate vpixel to test vertical pixels synchronization with vsync and hsync.

vpixel_controller vpixel_controller_0 (.reset(rst), .clk(clk),.new_frame(new_frame), .new_line(new_line), .vpixel(vpixel) );


endmodule
```

Κυματομορφές: Οι κυματομορφές της προσομοίωσης επαληθεύουν ότι οι χρόνοι γύρω από το VSYNC έχουν τις σωστές τιμές.

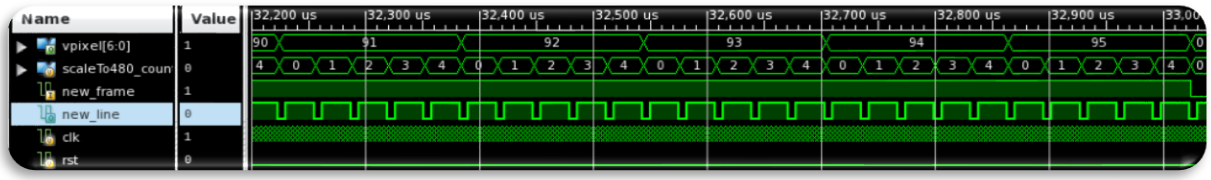








Όσον αφορά τον vpixel_controller μπορεί να φανεί διαισθητικά ότι οι κάθετες διευθύνσεις στέλνονται σωστά. Κάθε διεύθυνση αλλάζει ορθώς μετά από 5 σύνολο ενεργές γραμμές new_line.



Κάλυψη λειτουργίας: Για τη κάλυψη λειτουργίας του κυκλώματος, η τιμή της φτάνει το μέγιστο, καθώς και εδώ τα διανύσματα εισόδου που είναι απαραίτητα για το κύκλωμα είναι και εδώ μόνο το σήμα reset και το ρολόι. Για τον vpixel_controller, σήμα εισόδου είναι και το new_line και new_frame, τα οποία όμως λαμβάνει από τον hsync_driver και vsync_driver αντίστοιχα.

Ένωση Μερών Α,Β,Γ - Ολοκλήρωση Οδηγού VGA – Πείραμα στη Πλακέτα

Στο στάδιο της ένωσης μερών, ολοκληρώνεται ο οδηγός VGA με την ένωση των μερών Α,Β,Γ σε ένα ενιαίο συνθέσιμο σύστημα. Ο ελεγκτής VGA εκτός από το πλαίσιο δοκιμής, ελέγχεται στη πλακέτα για να διαπιστωθεί η σωστή λειτουργία του.

Το τελευταίο κομμάτι της εργασίας αποτελείται από 4 τμήματα:

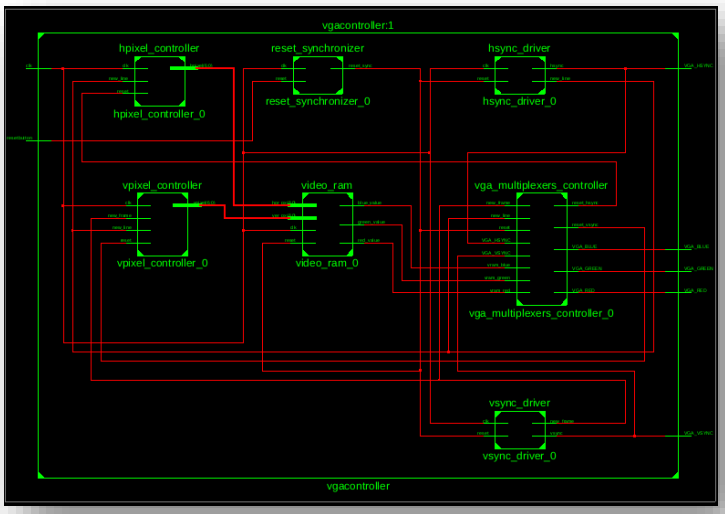
- Ο **vga_controller** που αποτελεί τη top-level μονάδα του τελικού κυκλώματος και κάνει instantiate όλες τις επιμέρους μονάδες.
- Ο **reset_synchronizer**, μονάδα έτοιμη από τις προηγούμενες εργασίες που συγχρονίζει τις το ασύγχρονο σήμα reset με το ρολόι.
- Η μονάδα **vga_multiplexers_controller**, της οποίας η υλοποίηση ήταν αναγκαία μετά από πειράματα του συστήματος πάνω στη πλακέτα. Η μονάδα αυτή αποτελείται μόνο από πολυπλέκτες που διαχειρίζονται τις τιμές του R,G,B του συστήματος, καθώς και τα σήματα reset των μονάδων hpixel_controller και vpixel_controller.
- Αρχείο vga_connections.ucf με τις απαραίτητες αντιστοιχίσεις των σημάτων του ελεγκτή VGA σε αυτά τις πλακέτας. Οι κατάλληλες αντιστοιχίσεις πάρθηκαν από το τεχνικό δελτίο της πλακέτας SPARTAN 3 με σκοπό την εφαρμογή του κυκλώματος πάνω σε αυτή.

Υλοποίηση:

vga_controller module: Η μονάδα του vga_controller αποτελεί το top module του συστήματος. Μέσα σε αυτό γίνονται instantiate όλες οι μονάδες του συστήματος που είναι οι εξής:

- Video Ram
- hsync_driver και hpixel_controller
- vsync_driver και vpixel_controller
- reset_synchronizer
- vga_multiplexers_controller

Μέσα από το εργαλείο ISE, εξάγουμε το σχηματικό της RTL του ολοκληρωμένου κυκλώματος με το αποτέλεσμα να είναι το παρακάτω:



Ο ελεγκτής δεν έχει περαιτέρω λογική πέρα των instantiations, ενώ ο κώδικας Verilog της μονάδας δίνεται παρακάτω:

```
//Synchronize reset using two flip flops to avoid setup and hold violations.

reset_synchronizer reset_synchronizer_0 (.reset(resetbutton), .clk(clk), .reset_sync(reset));

//1.Multiplexers Controller set zero on RGB Values when new_frame or new_line are is displayed.
//2.Also resets modules of hpixel or vpixel controller respectively when VGA_HSYNC or VGA_VSYNC is driven to zero.

vga_multiplexers_controller vga_multiplexers_controller_0 (.reset(reset), .VGA_HSYNC(VGA_HSYNC), .VGA_VSYNC(VGA_VSYNC),
                                                           .new_line(new_line), .new_frame(new_frame), .vram_red(vram_red),
                                                           .vram_green(vram_green), .vram_blue(vram_blue), .reset_hsync(reset_hsync),
                                                           .reset_vsync(reset_vsync), .VGA_RED(VGA_RED), .VGA_GREEN(VGA_GREEN), .VGA_BLUE(VGA_BLUE));

//Drive hsync signal correctly (correct timing for 640x480 60hz monitor) through hsync_driver instantiation

hsync_driver hsync_driver_0 (.reset( reset ), .clk( clk ), .hsync( VGA_HSYNC ), .new_line ( new_line ));

//Drive vsync signal correctly (correct timing for 640x480 60hz monitor) through vsync_driver instantiation

vsync_driver vsync_driver_0 (.reset( reset ), .clk( clk ), .vsync( VGA_VSYNC ),.new_frame ( new_frame ));

//Instantiate controller for horizontal pixel. Hpixel is synchronized with hsync signal.

hpixel_controller hpixel_controller_0 (.reset( reset_hsync ), .clk( clk ), .new_line ( new_line ), .hpixel( hpixel ));

//Instantiate controller for verical pixel. Vpixel is synchronized with vsync signal.

vpixel_controller vpixel_controller_0 (.reset( reset_vsync ), .clk( clk ), .new_frame ( new_frame ), .new_line ( new_line ), .vpixel( vpixel ) );

//Instantiate Video Ram. Video Ram consists of 3 block rams and contains correct colors for each pixel in screen.

video_ram video_ram_0(.reset(reset), .clk(clk), .ver_px ( vpixel ), .hor_px ( hpixel ) , .red_value( vram_red ), .green_value ( vram_green ), .blue_value( vram_blue ));
```

vga_multiplexer_controller module: Όπως ειπώθηκε κατά την εισαγωγή, η μονάδα αποτελείται μόνο από πολυπλέκτες και χειρίζεται τα σήματα R,G,B του ελεγκτή VGA και τα σήματα reset των μονάδων hpixel και vpixel.

Συγκεκριμένα οι τιμές του R,G,B γίνονται ίσες με τις τιμές εξόδου της Video Ram εντός χρονικού διαστήματος ενεργής γραμμής και εικόνας(frame) και ίσες με 0 εκτός των διαστημάτων. Από την άλλη τα σήματα reset των μονάδων hpixel και vpixel ελεγκτών διακρίνονται στο σήμα reset_hsync για τη πρώτη και το σήμα reset_vsync για τη δεύτερη. Συγκεκριμένα το σήμα reset_hsync ενεργοποιείται είτε από το top_level reset του κυκλώματος είτε από την οδήγηση του HSYNC στο 0. Ίδια διεργασία πραγματοποιείται και με το reset_vsync του vpixel_controller αλλά με τον έλεγχο του VSYNC αντί του HSYNC σήματος.

Ακολουθεί ο κώδικας Verilog της μονάδας:

```
module vga_multiplexers_controller (reset, VGA_HSYNC, VGA_VSYNC, new_line, new_frame,
                                    vram_red, vram_green, vram_blue, reset_hsync, reset_vsync,
                                    VGA_RED, VGA_GREEN, VGA_BLUE);

input reset;
input VGA_HSYNC, VGA_VSYNC;
input new_line, new_frame;
input vram_red, vram_green, vram_blue;

output VGA_RED, VGA_GREEN, VGA_BLUE, reset_hsync, reset_vsync;

//reset_hsync and reset_vsync are reset signals for hpixel_controller and vpixel_controller
//Except toplevel reset, after every VGA_HSYNC == 0 and VGA_VSYNC == 0 all counters of the above modules should reset.

assign reset_hsync = ( reset == 1'b1 || VGA_HSYNC == 1'b0 ) ? 1'b1 : 1'b0;
assign reset_vsync = ( reset == 1'b1 || VGA_VSYNC == 1'b0 ) ? 1'b1 : 1'b0;

//assign zero to each color when time exceeds timing part of frame or line.

assign VGA_RED   = ( new_line == 1'b1 && new_frame == 1'b1 ) ? vram_red   : 1'b0;
assign VGA_GREEN = ( new_line == 1'b1 && new_frame == 1'b1 ) ? vram_green : 1'b0;
assign VGA_BLUE  = ( new_line == 1'b1 && new_frame == 1'b1 ) ? vram_blue  : 1'b0;

endmodule
```

Αρχείο UCF συνδέσεων (vga_connections.ucf) Παρακάτω δίνεται το αρχείο UCF με τις κατάλληλες αντιστοιχίσεις στη πλακέτα.

```
NET clk LOC = T9;

NET VGA_RED LOC = R12;
NET VGA_GREEN LOC = T12;
NET VGA_BLUE LOC = R11;

NET VGA_HSYNC LOC = R9;
NET VGA_VSYNC LOC = T10;

NET resetbutton LOC = L14;

NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 20 ns HIGH 50 %;
```

Επαλήθευση:

Για την επαλήθευση όλου του κυκλώματος μέσα από κυματομορφές πριν το τελικό πείραμα, κατασκευάζεται ένα πλαίσιο ελέγχου που είναι σε θέση να επαληθεύσει το σύστημα σε επίπεδο συμπεριφοράς αλλά και μετά το στάδιο **Place & Route.**

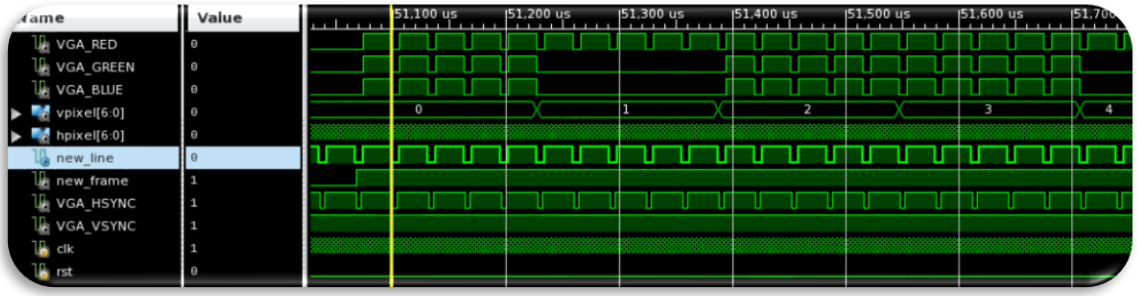
Το πλαίσιο δοκιμής είναι απλό και κάνει instantiate τον ελεγκτή VGA για να επαληθεύσει τις εξόδους του. Δίνεται παρακάτω:

```
`timescale 1ns/1000ps

module vga_controller_tb ();

reg clk = 0;
reg rst = 1;

//Initialize cirquit at a random moment (after 74ns)

initial begin
    #60000 rst=0;
end

//Fpga spartan 3 clock frequency ==> 50 Mhz

always #10 clk <= ~clk;

//test vga controller function

vgacontroller vgacontroller_0 (.resetbutton (rst), .clk(clk), .VGA_RED (VGA_RED), .VGA_GREEN (VGA_GREEN), .VGA_BLUE(VGA_BLUE),
                               .VGA_HSYNC(VGA_HSYNC), .VGA_VSYNC(VGA_VSYNC));

endmodule
```

Κυματομορφές: Οι κυματομορφές της προσομοίωσης το συστήματος μετά το στάδιο **Place & Route** επαληθεύουν το κύκλωμα. Διαισθητικά μπορεί να φανεί ότι το κύκλωμα λειτουργεί σωστά, βλέποντας τα παρακάτω στιγμιότυπα της προσομοίωσης.

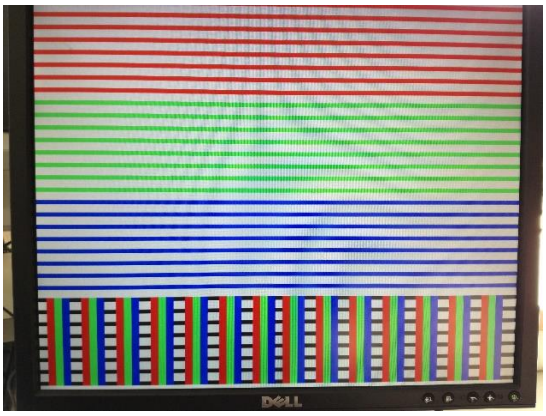
Σε πλήρη μεγέθυνση: Διακρίνονται τα 4 μέρη της στατικής εικόνας μέσα από τα σήματα VGA_RED, VGA_GREEN και VGA_BLUE

Σε εστίαση στο τμήμα των κόκκινων και άσπρων οριζοντίων γραμμών: Διακρίνεται πώς οι R,G,B τιμές σωστά λαμβάνουν τις τιμές της Video Ram εντός χρόνου ενεργής γραμμής (new_line) και ορθώς γίνονται 0 εκτός του διαστήματος.



Τελικό πείραμα:

Τελικό αποτέλεσμα πειράματος: Η φωτογραφία τραβήχτηκε από το εργαστήριο και δείχνει το τελικό αποτέλεσμα του πειράματος όταν η υλοποίηση ολοκληρώθηκε και το κύκλωμα λειτούργησε σωστά.



Δυσκολίες στην υλοποίηση: Βασική δυσκολία στην υλοποίηση αποτέλεσε η ανάγκη για βαθιά κατανόηση και μετέπειτα υλοποίηση του συγχρονισμού των hsync και vsync σημάτων. Όσο αυτά δεν συγχρονίζονταν η οθόνη στη διάρκεια το πειράματος έδειχνε μαύρο που επέδειχνε πως το πρόβλημα βρισκόταν στο συγχρονισμό. Μικρότερης δυσκολίας εμπόδιο αποδείχθηκε ο κατάλληλο ορισμός των διευθύνσεων, κάθετων και οριζοντίων. Μέχρι να επιλυθεί αυτό και το κύκλωμα να φτάσει στην τελική μορφή που περιεγράφηκε, η οθόνη έβγαζε μια μορφή «βίντεο» στο κάτω μέρος της και συγκεκριμένα στο πολύχρωμα τμήμα της αποθηκευμένης εικόνας με τις εναλλάξ οριζόντιες και κάθετες γραμμές.

Επαναλήψεις του πειράματος που χρειάστηκαν – Αλλαγές στο κώδικα: Οι επαναλήψεις στο πείραμα που χρειάστηκαν μπορούν να διαιρεθούν σε 3 τμήματα:

1. **Μαύρο στην οθόνη – Ένδειξη για μη αναγνώρισης της συχνότητας**: Το αποτέλεσμα των πρώτων πειραμάτων επέδειξε πρόβλημα στο συγχρονισμό, όπως ειπώθηκε παραπάνω. Αυτό συνέβη διότι στην αρχική υλοποίηση του κυκλώματος, λόγω λανθασμένης απορρόφησης των πληροφοριών του τεχνικού δελτίου της πλακέτας, το σύστημα και οι χρόνοι είχαν σχεδιαστεί για να λειτουργούν κάτω από ρολόι 25 MHz, όπως αναφέρει χαρακτηριστικά το τεχνικό δελτίο (*"pixel clock 25 MHz" – Spartan-3 Starter Kit Board User Guide – σελίδα 24*). Πέρα από το συγκεκριμένο πρόβλημα, το HSYNC δεν συγχρονιζόταν σωστά με το VSYNC και χρονικό διάστημα της πρώτης ενεργής γραμμής (διεύθυνση (0,0-639)) ξεκινούσε μετά από αυτό του ενεργού frame. Συνεπώς, οι χρόνοι άλλαξαν για να λειτουργούν κάτω από 50 MHz ρολόι ως προς το πρώτο πρόβλημα και προστέθηκε ο first_time_counter για το δεύτερο (βλέπε μέρος Γ της αναφοράς) και το πρόβλημα λύθηκε.

2. **Μορφή Video στο κάτω μέρος της οθόνης:** Όπως αναφέρθηκε στη προηγούμενη παράγραφο των δυσκολιών, στο σημείο που αναμενόταν το πολύχρωμο κομμάτι της αποθηκευμένης εικόνας, παραγόταν μια μορφή κινούμενης εικόνας. Το πρόβλημα αυτό οφειλόταν σε λάθος ορισμό των οριζοντίων και κάθετων διευθύνσεων και συγκεκριμένα στο ότι κάποιοι μετρητές δεν αρχικοποιούνταν στο 0 μετά το τέλος του χρονικού διαστήματος της απεικόνισης εικόνας και την αρχή του επόμενου. Για το λόγο αυτό κατασκευάστηκε η μονάδα των πολυπλεκτών vga_multiplexers_controller, η οποία αρχικοποιεί σωστά τις μονάδες hpixel_controller και vpixel_controller με ένα σύνθετο σήμα reset όταν οδηγούνται τα σήματα HSYNC και VSYNC στο 0.

3. **Τελικό αποτέλεσμα-σωστή αναπαραγωγή εικόνας:** Με την επίλυση των προβλημάτων που παρουσιάστηκαν στα τμήματα 1 και 2, το πείραμα έβγαζε στη πλακέτα το σωστό αποτέλεσμα στην οθόνη μετά από κάθε επαναληπτική δοκιμή. Το σωστό αποτέλεσμα στην οθόνη είναι αυτό ίδιο με την φωτογραφία που δόθηκε παραπάνω κατά τη διάρκεια του εργαστηρίου.

Συμπέρασμα

Με το τέλος της περιγραφής των τμημάτων της εργασίας, ολοκληρώνεται η τεχνική αναφορά που σχετίζεται με την υλοποίηση του οδηγού θύρας VGA. Παρακάτω δίδεται το συμπέρασμα που συνοψίζει τη πορεία κατασκευής από την αρχή της σχεδίασης μέχρι το τελικό πείραμα.

Η τρίτη εργαστηριακή άσκηση στο σύνολο της και αφού επήλθε η ολοκλήρωση της. μπορεί σε υποκειμενικό επίπεδο να χαρακτηριστεί άσκηση μέτριας προς μεγάλης δυσκολίας.

Η υλοποίηση της Video Ram που αφορά το μέρος Α της παρούσας εργασίας απαίτησε σημαντικό χρόνο από τον στάδιο μετάβασης της κατανόησης της εκφώνησης στο σχεδιασμό της σε κώδικα Verilog. Αφού οι οδηγίες αυτές έγιναν κατανοητές, το επόμενο μεγάλο διάστημα χρόνου συγκεντρώθηκε στην αποθήκευση της στατικής εικόνας στις τρεις Block Rams (μέσω της κατάλληλης αρχικοποίησης των .INIT_xx τιμών) και ιδιαίτερα στο τομέα τού κάτω μέρους της οθόνης με τα οριζόντια και κάθετα χρώματα.

Στη συνέχεια και όσον αφορά τις μονάδες των drivers των σημάτων HSYNC, VSYNC στα μέρη Β και Γ αντίστοιχα, δεν εμφανίστηκαν σημαντικά εμπόδια στον σχεδιασμό. Με την κατάλληλη υλοποίηση των counters που αναφέρθηκαν παραπάνω, ήταν εφικτό να συγχρονιστούν ακριβώς τα 2 σήματα με τους χρόνους που ορίζει η ανάλυση 640x480, αλλά και ο ρυθμός ανανέωσης των 60 Hz.

Εμμένοντας όμως στα μέρη Β και Γ, το επόμενο μερίδιο δυσκολίας αναφορικά με την υλοποίηση, αντιστοιχεί στον σχεδιασμό των controllers της οριζόντιας και κάθετης διεύθυνσης των pixels. Δεδομένης της ανάγκης να μεγεθυνθεί η αποθηκευμένη εικόνα ανάλυσης 128x96 στην επιθυμητή ανάλυση 640x480, έπρεπε ο σχεδιασμός των controllers να χρησιμοποιεί τους κατάλληλους counters στο εσωτερικό των μονάδων (scaleTo640_counter και scaleTo480_counter αντίστοιχα), οι οποίοι παράλληλα θα είναι συγχρονισμένοι με τα σήματα VSYNC και HSYNC. Αυτό ήταν απαραίτητο, ώστε η διεύθυνση κάθε pixel να μένει η ίδια για 5 φορές σε αριθμό, ώστε τα 128x96 = 12288 pixels να αντιστοιχούν επακριβώς σε 5 x122888 = 640 x 480 = 307200 pixels σύνολο.

ΤΕΛΟΣ ΑΝΑΦΟΡΑΣ