ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ (ΗΥ430)

ΤΕΧΝΙΚΗ ΑΝΑΦΟΡΑ 4ΗΣ ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

ΠΑΝΑΓΙΩΤΗΣ ΑΝΑΣΤΑΣΙΑΔΗΣ 2134

ΤΙΤΛΟΣ: «ΥΛΟΠΟΙΗΣΗ ΟΔΗΓΟΥ ΕΝΔΕΙΞΗΣ LCD»

Περίληψη Τεχνικής Αναφοράς

Η 4η εργαστηριακή άσκηση υλοποιήθηκε με τη χρήση τριών μηχανών πεπερασμένων καταστάσεων.

Συγκεκριμένα οι 3 ΜΠΚ που χρησιμοποιήθηκαν αποτελούνται από τη ΜΠΚ αρχικοποίησης της ένδειξης LCD, την ΜΠΚ διαχείρισης των χρονικών περιορισμών και την κεντρική ΜΠΚ που πραγματοποιεί τη διαμόρφωση της ένδειξης καθώς και την εγγραφή των δεδομένων σε αυτή..

- ***Στόχοι***: Εισαγωγή στην 4η εργαστηριακή εργασία, αναλύοντας τα ζητούμενα και τους στόχους που απαιτεί αυτή να επιτευχθούν ολοκληρώνοντας την υλοποίηση.

- ***Υλοποίηση ΜΠΚ διαχείρισης χρονικών περιορισμών****:* Η μηχανή αυτή αναλαμβάνει, εφόσον ενεργοποιηθεί από την ΜΠΚ κεντρικού ελέγχου να διαχειρίζεται το σήμα LCD_E και να στέλνει κατάλληλα σήματα που επιδεικνύουν στην κεντρική μηχανή πότε να αλλάξει τα δεδομένα (εντολών ή χαρακτήρων) ώστε να αποφευχθούν φαινόμενα setup και hold. Δεν χρησιμοποιείται κατά την αρχικοποίηση της ένδειξης.

- ***Υλοποίηση ΜΠΚ αρχικοποίησης της ένδειξης:*** Η μηχανή καταστάσεων της αρχικοποίησης εκτελεί βήμα προς βήμα τις διαδικασίες που ορίζει η εκφώνηση ότι πρέπει να πραγματοποιηθούν μετά την αρχικοποίηση του κυκλώματος.

- ***Υλοποίηση κεντρικής ΜΠΚ διαμόρφωσης ένδειξης & αποστολής δεδομένων****:* Η κεντρική μηχανή ξεκινά την λειτουργία της μετά το στάδιο της αρχικοποίησης. Ρόλος της είναι να διαμορφώνει την οθόνη με κάθε ανανέωση της (ενεργοποιεί ένδειξη, ορίζει κατάλληλες παραμέτρους, καθαρίζει την οθόνη) και να στέλνει τις κατάλληλες εντολές ή τα δεδομένα ASCII, τα οποία βρίσκονται αποθηκευμένα σε μια διαμόρφωση Block Ram. Για κάθε εντολή ή δεδομένο που μεταφέρει στην ένδειξη, χρησιμοποιεί την ΜΠΚ διαχείρισης χρονικών περιορισμών, ώστε αυτά να στέλνονται εντός του επιθυμητού χρόνου.

- ***Κατασκευή του LCD Driver:*** Στο στάδιο αυτό οι κατασκευασμένες ΜΠΚ που αναφέρθηκαν παραπάνω, ενώνονται κατάλληλα ως υπό-κυκλώματα στο εσωτερικό του top-level συστήματος, το οποίο και αποτελεί τον οδηγό ένδειξης της οθόνης LCD.

- ***Τελικό Πείραμα:*** Δοκιμή του κυκλώματος στην πλακέτα και σχολιασμός των αποτελεσμάτων.

- ***Συμπέρασμα****:* Εξαγωγή συμπεράσματος για την συνολική δομή της εργασίας, το τελικό πείραμα και τις δυσκολίες που προέκυψαν κατά τον σχεδιασμό ή την δοκιμή στην πλακέτα.

Στόχοι Της 4ης Εργασίας

Η τέταρτη εργασία θέτει ως ζητούμενο την υλοποίηση ενός οδηγού ένδειξης LCD για την πλακέτα Spartan 3E της Xilinx. Με την ολοκλήρωση της υλοποίησης του οδηγού και τον προγραμματισμό του πάνω στη πλακέτα, η ένδειξη LCD θα πρέπει, ως αποτέλεσμα αυτού, να εμφανίζει ένα μήνυμα 31 χαρακτήρων μαζί με έναν δρομέα που θα δείχνει την ανανέωση της οθόνης.

Αντικείμενο της τέταρτης εργαστηριακής άσκησης, είναι η κατασκευή του οδηγού ένδειξης LCD, συγκεκριμένα της οθόνης LCD 32 χαρακτήρων (5 x 8 pixel) της πλακέτας Spartan 3E (Xilinx).

Βασικό σκοπό της εργασίας αποτελεί η παρουσίαση ενός μηνύματος στην οθόνη **(2x16 χαρακτήρων)** μέσα από τον ελεγκτή, το οποίο απαρτίζεται από τα εξής δυο τμήματα:

- **το τμήμα 16 χαρακτήρων «ABCDEFGHIJKLMNOP» στο πάνω μέρος.**
- **το τμήμα 15 χαρακτήρων «abcdefghijklmno» συν τον δρομέα που αντιστοιχεί στην τελευταία θέση και επιδεικνύει την ανανέωση της οθόνης.**

Για το μήνυμα που αναφέρθηκε, στόχος είναι η χρησιμοποίηση μιας διαμόρφωσης Block Ram στην οποία θα αποθηκευτεί κατάλληλα. Η BRAM θα είναι των 8-bit δεδομένων και θα αντιστοιχιστεί στην πάνω αριστερά BRAM της συσκευής.

Στη συνέχεια, αναφορικά με τις διαδικασίες αρχικοποίησης που απαιτεί η χρήση της ένδειξης LCD, θα υλοποιηθεί μια μηχανή πεπερασμένων καταστάσεων που θα περιλαμβάνεται σε ένα ανεξάρτητο module της Verilog. Η μονάδα, αφού πραγματοποιήσει την αρχικοποίηση της ένδειξης, θα αποστείλει ένα σήμα που δηλώνει την ολοκλήρωση αυτή στην κεντρική μονάδα του συστήματος.

Μετέπειτα, ως προς τους χρονικούς περιορισμούς στην οδήγηση των σημάτων του ελεγκτή που ορίζονται αναλυτικά από το τεχνικό δελτίο της πλακέτας Spartan 3E, μια ξεχωριστή μονάδα θα οδηγεί ως ΜΠΚ το σήμα ενεργοποίησης LCD_E κατάλληλα για κάθε εντολή ή δεδομένο που πρόκειται να σταλεί. Επιπρόσθετα, στόχος της μονάδας είναι να δηλώνει στην κεντρική μονάδα πότε είναι ασφαλές να αλλάξει τα δεδομένα προς την ένδειξη, ώστε να αποφευχθούν φαινόμενα SETUP και HOLD.

Τελευταίο τμήμα της υλοποίησης πριν την ένωση των επιμέρους μονάδων, είναι η υλοποίηση της κεντρικής μηχανής πεπερασμένων καταστάσεων. Ως άλλη μια ξεχωριστή μονάδα, η κεντρική μηχανή χρησιμοποιώντας την ΜΠΚ των χρονικών περιορισμών, θα εκτελεί την διαδικασία διαμόρφωσης της ένδειξης πριν από κάθε προβολή του μηνύματος και λαμβάνοντας το μήνυμα από την Block Ram θα το στέλνει στο κατάλληλο χρονικό περιθώριο. Η ανανέωση του δρομέα και κατά επέκταση της οθόνης θα πραγματοποιείται κάθε ένα δευτερόλεπτο.

Η ένωση των μονάδων που αναφέρθηκαν αποτελεί και τον τελικό σκοπό, η οποία θα γίνει μέσα από ένα top-level module στη Verilog που κάνοντας Instantiate όλες τις μονάδες, θα οδηγεί καταλλήλως τα σήματα του ελεγκτή προς την ένδειξη LCD.

Μέρος Α - Υλοποίηση ΜΠΚ Χρονικών Περιορισμών

Η υλοποίηση της μηχανής καταστάσεων των χρονικών περιορισμών διαχειρίζεται το σήμα ενεργοποίησης της ένδειξης LCD_E, με τρόπο ώστε να μεσολαβούν οι κατάλληλες χρονικές διάρκειες που αποτρέπουν την εμφάνιση SETUP και HOLD παραβιάσεων.

Βάσει του τεχνικού δελτίου της πλακέτας SPARTAN-3E στην οποία και προγραμματίζεται η ένδειξη LCD, απαιτούνται τα εξής στοιχεία ως προς την διαχείριση του χρόνου και αναφέρονται εδώ επιγραμματικά.

- **Τα σήματα SF_D<11:8> και LCD_RS πρέπει να είναι έτοιμα τουλάχιστον 40 ns πριν τη άνοδο του LCD_E.**
- **Τα σήματα SF_D<11:8> και LCD_RS πρέπει να διατηρήσουν την τιμή τους για τουλάχιστον 10 ns μετά την πτώση του LCD_E.**
- **Απαιτείται τουλάχιστον 240ns χρονική διάρκεια για το σήμα LCD_E να παραμείνει στο 1.**
- **Απαιτείται μεσολάβηση τουλάχιστον 1 μs χρονικού διαστήματος από την αποστολή των Upper 4 bits μιας 8-bit εντολής ή 8-bit δεδομένων μέχρι την αποστολή των Lower Bits.**
- **Απαιτείται μεσολάβηση τουλάχιστον 40μs χρονικού διαστήματος από την αποστολή των Lower Bits μέχρι την επόμενη αποστολή δεδομένων ή εντολής.**

Είσοδοι/Εξοδοι Μονάδας:

Η δομή της μονάδας παίρνει για είσοδο εκτός από το reset και το ρολόι, ένα σήμα ενεργοποίησης/απενεργοποίησης **fsm_enable** που εκκινεί την μηχανή ή την κρατά αδρανή.

Ένα σήμα **lcd_e_tc_fsm** βγαίνει στην έξοδο της μονάδας και αφορά το παραγόμενο LCD_E σήμα, το οποίο στη συνέχεια θα διαχειριστεί μια μονάδα πολυπλεκτών (βλέπε παρακάτω, Μέρος Γ) για να το επιλέξει ή όχι ως το βασικό σήμα LCD_E εξόδου του οδηγού.

Τέλος δύο σήματα **send_upper** και **send_lower** (1-bit και διάρκειας 1 κύκλου) οδηγούνται στην έξοδο για να δηλώσουν στην κεντρική μηχανή διαμόρφωσης τον κύκλο στον οποίο πρέπει να στείλει στην έξοδο τα ανάλογα 4-bit δεδομένα.

Υλοποίηση:

time_control_fsm module: Η δομή της μηχανής 5 καταστάσεων που υπάρχει στο εσωτερικό της μονάδας φαίνεται αναλυτικά παρακάτω:

```
always @(posedge clk or posedge reset) begin

    if (reset) begin
        // reset
        state = STATE_IDLE;
    end
    else begin
        state = next_state;
    end

end

//States management

always @(*) begin

    next_state = state;
    setup_counter_enable = 1'b0;
    palm_enable = 1'b0;
    wait_enable = 1'b0;
    send_upper = 1'b0;
    send_lower = 1'b0;

    case (state)

        STATE_IDLE: begin

            if ( fsm_enable == 1'b1 ) begin //module is idle until fsm is activated.
                next_state = STATE_SETUP;
            end

        end
        STATE_SETUP: begin

            setup_counter_enable = 1'b1; //enable setup counter

            if ( setup_counter == 2'd1 ) begin //if setup counter reaches max value, proceed to next state
                next_state = STATE_LCD_E_PALM;
            end

        end
        STATE_LCD_E_PALM: begin

            palm_enable = 1'b1; //enable lcd_e palm counter

            if ( palm_counter == 4'd11 ) begin //if palm counter reaches max value, proceed
                next_state = STATE_HOLD;
            end

        end
        STATE_HOLD:  begin

            if ( switch_to_1us == 1'b0 ) begin  //if upper bits (new command or date) are to be sent,
                send_upper = 1'b1;              //..send to output 1-cycle-active signal for upper bits
            end
            else begin //if Lower bits are to be sent, send to output 1-cycle-active signal for lower bits
                send_lower = 1'b1;
            end

            next_state = STATE_WAIT;  //hold state lasts one cycle until next state

        end
        STATE_WAIT: begin

            wait_enable = 1'b1; //start wait counter for 40us or 1us

            if ( count_to_1us == 6'd49 || count_to_40us == 11'd1999 ) begin //if one of the counters is active and reaches max valu
                next_state = STATE_SETUP;                                   //..proceed to state setup for new data to be sent.
            end
            else if ( fsm_enable == 0 ) begin //if input signal disabled fsm, go to state IDLE
                next_state = STATE_IDLE;
            end

        end
```

- ***STATE_IDLE***: Η μονάδα παραμένει στην κατάσταση αυτή, εφόσον το σήμα εισόδου fsm_enable είναι 0, αλλιώς μεταβαίνει στην κατάσταση STATE_SETUP.

- ***STATE_SETUP & STATE_LCD_E_PALM**:* Οι καταστάσεις αυτές ενεργοποιούν από την αδράνεια το ακόλουθο always block. Το σήμα setup_counter_enable όταν είναι 1, ενεργοποιεί τον μετρητή setup_counter που μετρά συνολικά 40 ns. Όταν τελειώσει την μέτρηση, το σήμα lcd_e_tc_fsm γίνεται 1 και η μονάδα μεταβαίνει στην κατάσταση STATE_LCD_E_PALM, όπου το σήμα palm_enable ενεργοποιεί τον palm_counter. Ο τελευταίος αφού μετρήσει 240 ns ρίχνει το lcd_e_tc_fsm στο 0 και η μηχανή μεταβαίνει στην κατάσταση STATE_HOLD.

```
//----------------------ALWAYS BLOCK --> FOR STATES SETUP & LCD_E_PALM-----------------------------------//

//1.Setup counter counts 40ns, in order to raise LCD_E(lcd_e_tc_fsm) value to 1. (STATE SETUP)
//Setup counter only increments in state setup, when setup_enable == 1.

//2.Palm counter counts 240ns(12 cycles), for LCD_E(lcd_e_tc_fsm) to hold its value to 1. (STATE LCD_E_PALM)
//Palm counter only increments in state LCD_E_PALM, when palm_enable == 1.

always @(posedge clk or posedge reset) begin

    if ( reset ) begin // reset
        setup_counter = 2'd0;
        palm_counter = 4'd0;
        lcd_e_tc_fsm = 1'b0;
    end
    else if ( palm_counter == 4'd11 && palm_enable == 1'b1 ) begin
        palm_counter = 4'd0;
        lcd_e_tc_fsm = 1'b0;
    end
    else if ( palm_enable == 1'b1 ) begin
        palm_counter = palm_counter + 1;
    end
    else if ( setup_counter == 2'd1 && setup_counter_enable == 1'b1 ) begin
        setup_counter = 2'd0;
        lcd_e_tc_fsm = 1'b1;
    end
    else if ( setup_counter_enable == 1'b1 ) begin
        setup_counter = setup_counter + 1;
    end

end

//-------------------------------------------------------------------------------------------------------//
```

- ***STATE_HOLD:*** Η μηχανή στην κατάσταση STATE_HOLD ανάλογα με την τιμή του switch_to_1us στέλνει στην έξοδο είτε το σήμα send_upper = 1 είτε το σήμα send_lower = 1. Κατά την αρχικοποίηση της μονάδας το switch_to_1us έχει την τιμή 1. Οι καταχωρητές 1-bit switch_to_1us και switch_to_40us, όπως θα παρουσιαστεί παρακάτω, δηλώνουν κατά την κατάσταση STATE_WAIT αν πρόκειται το κύκλωμα να περιμένει 1 us για να στείλει μετέπειτα τα lower bits ή πρέπει να περιμένει 40 us ώστε να σταλθεί στη συνέχεια νέα εντολή. Η κεντρική μηχανή καταστάσεων (περιγράφεται παρακάτω) συμβουλεύεται τα σήματα send_lower και send_upper για να αλλάξει τα δεδομένα στον επόμενο κύκλο, επομένως μεσολαβεί χρονικό διάστημα 20 ns και έτσι αποφεύγεται η παραβίαση HOLD.

- ***STATE_WAIT:*** Στην κατάσταση αυτή το σήμα wait_enable γίνεται 1. Όπως φαίνεται στο always block παρακάτω, αναλόγως το ποιος register εκ των switch_to_1us και switch_to_40us είναι 1, ο αντίστοιχος μετρητής (count_to_1us ή count_to_40us) μετρά 1 us ή 40 us. Σε περίπτωση που η μηχανή απενεργοποιηθεί (fsm_enable == 0) οι μετρητές του block μηδενίζονται και η μηχανή επιστρέφει στην αδράνεια και την κατάσταση STATE_IDLE. Σε άλλη περίπτωση όταν ένας εκ των μετρητών count_to_1us και count_to_40us μετρήσει τον αντίστοιχο χρόνο, η μηχανή επιστρέφει στην κατάσταση STATE_SETUP, ώστε να σταλθούν είτε τα lower bits της τρέχουσας εντολής είτε τα upper bits της επόμενης.

```
//------------------------ALWAYS BLOCK --> FOR STATE WAIT-----------------------------------------------------//

//This always block is active when time_control fsm is activated (fsm_enable == 1)
//..and cirquit is going through state STATE_WAIT

//1.if fsm is deactivated (fsm_enable == 0), leave always block "idle" with same values as reset.

//2.when in state WAIT, and upper bits are already sent --> wait 1us (switch_to_1us == 1, switch_to_40us == 0),
//Increment count_to_1us counter.

//3.when in state WAIT, and next command or data are to be sent --> wait 40us (switch_to_1us == 0, switch_to_40us == 1),
//Increment count_to_40us counter.

always @(posedge clk or posedge reset) begin

  if (reset) begin
    count_to_1us = 6'd0;
    count_to_40us = 11'd0;
    switch_to_1us = 1'b1;
    switch_to_40us = 1'b0;
  end
  else if ( fsm_enable == 0 ) begin
    count_to_1us = 6'd0;
    count_to_40us = 11'd0;
    switch_to_1us = 1'b1;
    switch_to_40us = 1'b0;
  end
  else if ( wait_enable == 1'b1 && count_to_1us == 6'd49 && switch_to_1us == 1'b1 ) begin
    count_to_1us = 6'd0;
    switch_to_1us = 1'b0;
    switch_to_40us = 1'b1;
  end
  else if ( wait_enable == 1'b1 && switch_to_1us == 1'b1 ) begin
    count_to_1us = count_to_1us + 1;
  end
  else if ( wait_enable == 1'b1 && count_to_40us == 11'd1999 && switch_to_40us == 1'b1  ) begin
    count_to_40us = 11'd0;
    switch_to_1us = 1'b1;
    switch_to_40us = 1'b0;
  end
  else if ( wait_enable == 1'b1 && switch_to_40us == 1'b1 ) begin
    count_to_40us = count_to_40us + 1;
  end

end

//------------------------------------------------------------------------------------------------------------//
```

bram module: Για την επιλογή της Block Ram που απαιτεί η αποθήκευση του μηνύματος, επιλέχθηκε μια διαμόρφωση 8-bit από τα έτοιμα Language Templates του λογισμικού ISE της Xilinx.Το μήνυμα όπως φαίνεται παρακάτω αποθηκεύεται στις πρώτες θέσεις των INIT_xx παραμέτρων.

Σημείωση: Η σειρά «abcdefghijklmno(κέρσορας)» που αντιστοιχεί στην δεύτερη γραμμή της ένδειξης LCD, αποθηκεύεται 24 θέσεις μετά την πρώτη σειρά «ABCDEFGHIJKLMNOP», ενώ οι 24 θέσεις αυτές περιέχουν μηδενικά. Αυτό συμβαίνει γιατί οι 24 αντίστοιχες διευθύνσεις της ένδειξης LCD δεν προφέρονται για προβολή στην οθόνη.

Τέλος, αναφορικά με τον κέρσορα, η θέση του ορθογωνίου παραλληλογράμμου αποθηκεύεται στην προτελευταία θέση των δεδομένων, δηλαδή την θέση 55 της μνήμης, ενώ ο κενός χαρακτήρας στην τελευταία θέση των δεδομένων, την θέση 56.

Παρακάτω δίνεται ένα τμήμα του κώδικα της Block Ram στη Verilog και συγκεκριμένα το τμήμα των INIT_xx παραμέτρων που περιέχουν τα δεδομένα. Η δεκαεξαδική κωδικοποίηση κάθε χαρακτήρα βασίζεται στον πίνακα με τις κωδικοποιήσεις των χαρακτήρων που υπάρχει στο τεχνικό δελτίο της πλακέτας.

```
RAMB16_S9 #(
   .INIT(9'h000),  // Value of output RAM registers at startup
   .SRVAL(9'h000), // Output value upon SSR assertion
   .WRITE_MODE("WRITE_FIRST"), // WRITE_FIRST, READ_FIRST or NO_CHANGE

   // The following INIT_xx declarations specify the initial contents of the RAM
   // Address 0 to 511
   .INIT_00(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_50_4F_4E_4D_4C_4B_4A_49_48_47_46_45_44_43_42_41),
   .INIT_01(256'h00_00_00_00_00_00_00_20_FF_6F_6E_6D_6C_6B_6A_69_68_67_66_65_64_63_62_61_00_00_00_00_00_00_00),
   .INIT_02(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   .INIT_03(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   .INIT_04(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   .INIT_05(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   .INIT_06(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   .INIT_07(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   .INIT_08(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   .INIT_09(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   .INIT_0A(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   .INIT_0B(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   .INIT_0C(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   .INIT_0D(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   .INIT_0E(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   .INIT_0F(256'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00),
   // Address 512 to 1023
```

Επαλήθευση:

Για την επαλήθευση της μονάδας δημιουργείται το κατάλληλο κύκλωμα προσομοίωσης, στο οποίο γίνονται instantiate τόσο η μονάδα της μηχανής των χρονικών περιορισμών time_control_fsm όσο και η μονάδα της Block Ram, με σκοπό να ελεγχθούν οι χρονικοί περιορισμοί κατά την αποστολή δεδομένων.

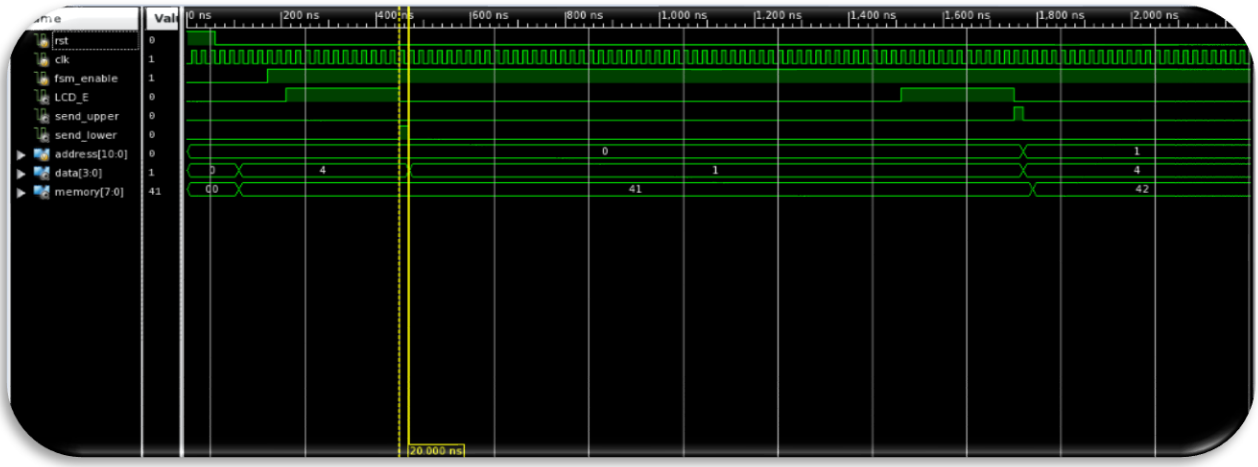
Για τον λόγο αυτό, ο κώδικας Verilog του Testbench που φαίνεται παρακάτω, σχεδιάστηκε έτσι ώστε να χρησιμοποιεί την μηχανή καταστάσεων χρονικών περιορισμών για να απεικονίσει στην έξοδο όλα τα δεδομένα της Block Ram που προορίζονται για την ένδειξη LCD.

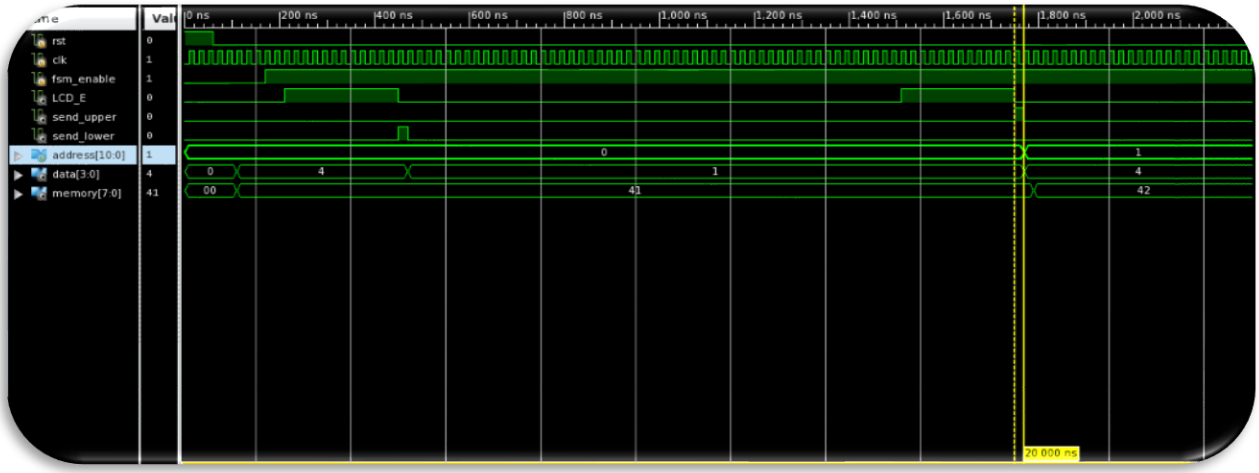
```
//Initialize cirquit at a random moment (after 74ns)

initial begin
    #60 rst=0;
    #110 fsm_enable = 1;
        LCD_RS = 1;
end

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

always #10 clk <= ~clk;

//Increment address counter to display the full message of Block Ram

always @(posedge clk or posedge rst) begin
    if ( rst ) begin
        address = 11'd0;
        lower_bits_time = 1'b0;
    end
    if ( address == 11'd55 && send_upper == 1'b1 ) begin
        address = 11'd0;
        fsm_enable = 0;
        lower_bits_time = 1'b0;
        LCD_RS = 0;
    end
    else if ( send_upper == 1'b1 ) begin
        address = address + 1;
        lower_bits_time = 1'b0;
    end
    else if ( send_lower == 1'b1 ) begin
        lower_bits_time = 1'b1;
    end
end

//Block Ram 8-bit Instantiation. Block ram contains the message for LCD Display

bram bram_0 ( .clk( clk ), .addr( address ), .char( memory ) );

//Assign to data output upper or lower 4-bits of 8-bit Block Ram element

assign data = ( lower_bits_time == 1'b1  ) ? memory[3:0] : memory [7:4];

//Instantiate and test time control FSM module

time_control_fsm time_control_fsm_0 (.reset( rst ), .clk( clk ), .fsm_enable( fsm_enable ),
                                     .lcd_e_tc_fsm( LCD_E ), .send_upper( send_upper ),
                                     .send_lower( send_lower ) );
```

Η λειτουργία του πλαισίου ελέγχου είναι συνοπτικά η εξής:

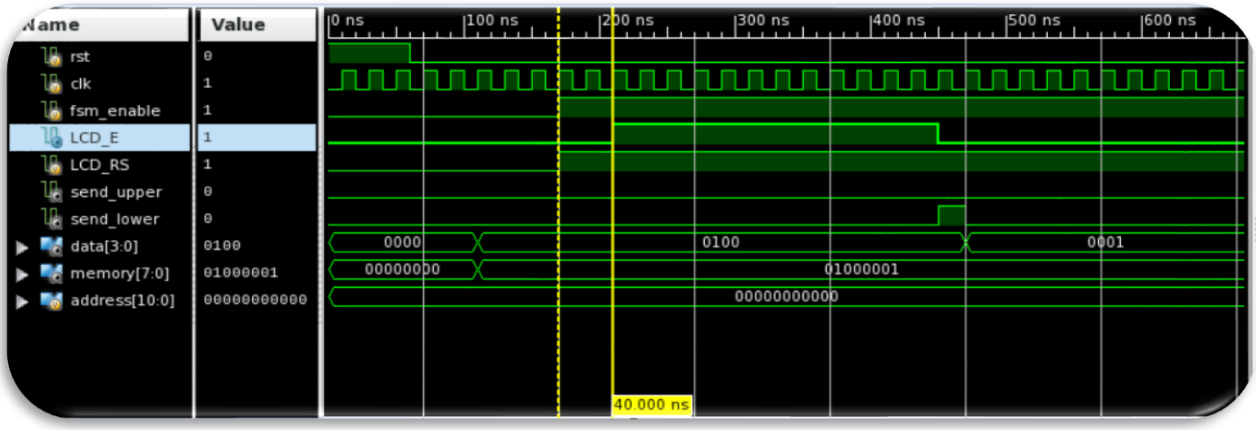
- Στο initial block στέλνεται σε τυχαία επιλεγμένη στιγμή, σήμα ενεργοποίησης της μηχανής καταστάσεων (fsm_enable = 1) και ταυτόχρονα ορίζεται το LCD_RS στην τιμή 1 για να δηλώσει την αποστολή δεδομένων που θα ακολουθήσει.
- Στο always block που ακολουθεί και σε κάθε σήμα send_upper που στέλνεται από τη μονάδα της μηχανής, η τιμή της διεύθυνσης της Block Ram (address) αυξάνεται κατά 1 για να προσπελαστεί το νέο 8-bit δεδομένο. Όταν αντίθετα στέλνεται σήμα send_lower, η τιμή του register lower_bits_time γίνεται 1. Τελικά, όταν όλες οι διευθύνσεις της μνήμης που αντιστοιχούν σε δεδομένα προσπελαστούν (address == 11'd55), η μονάδα της μηχανής απενεργοποιείται (fsm_enable == 0).
- Ολοκληρώνοντας με την εντολή assign που ακολουθεί, για lower_bits_time == 1, τα 4 λιγότερα σημαντικά bits της μνήμης οδηγούνται στην έξοδο data, ενώ στην αντίθετη περίπτωση οδηγούνται τα περισσότερα σημαντικά bits.

Κυματομορφές: Οι κυματομορφές που προκύπτουν από την προσομοίωση, επαληθεύουν ότι οι χρονικοί περιορισμοί τηρούνται αυστηρά κατά την αποστολή δεδομένων.





Για τον πρώτο αποθηκευμένο χαρακτήρα στη μνήμη, το γράμμα με κωδικοποίηση 0x41 (γράμμα «**A**»), τόσο για τα upper bits (πρώτη εικόνα) όσο και για τα lower bits (δεύτερη εικόνα), τα δεδομένα data κρατούν την τιμή τους σταθερή για 20 ns μετά την πτώση του LCD_E.



Με την ενεργοποίηση της FSM, επαληθεύεται από το στιγμιότυπο ότι το LCD_RS παίρνει την τιμή του 40 ns πριν την άνοδο του LCD_E. Επίσης, πριν από κάθε LCD_E == 1, τα δεδομένα data παίρνουν και διατηρούν την τιμή τους πολλά nanoseconds παραπάνω από το ελάχιστο των 40 ns που απαιτείται. Αυτό συμβαίνει γιατί τα δεδομένα αλλάζουν την τιμή τους ακριβώς μόλις περάσουν τα 20 ns που χρειάζονται για την αποφυγή HOLD παραβιάσεων μετά την πτώση του LCD_E.

Κάλυψη λειτουργίας: Η κάλυψη των διανυσμάτων ελέγχου του πλαισίου δοκιμής είναι η μέγιστη δυνατή, διότι ελέγχονται όλες οι δυνατές διευθύνσεις της Block Ram που περιέχουν δεδομένα. Συνεπώς, η μηχανή ενεργοποιείται και ελέγχεται για την αποστολή όλων των δεδομένων που προορίζονται για την ένδειξη LCD.

Μέρος Β – Υλοποίηση ΜΠΚ Αρχικοποίησης Της Ένδειξης

> *Η υλοποίηση της ΜΠΚ της αρχικοποίησης εκτελεί τις απαραίτητες ενέργειες που αφορούν την αρχικοποίηση του κυκλώματος. Ένα σήμα initialization_done οδηγείται στο 1 για να δηλώσει το τέλος της διαδικασίας, όταν αυτή ολοκληρωθεί.*

Για το μέρος Β, υλοποιείται η μηχανή καταστάσεων που αναλαμβάνει να πραγματοποιήσει την διαδικασία αρχικοποίησης της ένδειξης κατά την εκκίνηση λειτουργίας του συστήματος. Η μηχανή, σε κώδικα Verilog υλοποιείται ως ξεχωριστή μονάδα που εκτελεί κατά σειρά τις ενέργειες αρχικοποίησης, έτσι όπως περιγράφονται από την εκφώνηση. Η εικόνα που ακολουθεί, απεικονίζει αυτούσιο το κομμάτι της εκφώνησης της εργασίας που αφορά τα βήματα αρχικοποίηση της ένδειξης.

Αρχικοποίηση:

i) Αναμένουμε 15ms μετά την αρχικοποίηση του κυκλώματος.
ii) Γράφουμε SF_D<11:8> = 0x03, και δίνουμε θετικό παλμό στο LCD_E πλάτους 12 κύκλων.
iii) Αναμένουμε 4,1ms ή περισσότερο (205.000 κύκλοι των 50MHz).
iv) Γράφουμε SF_D<11:8> = 0x03, και δίνουμε θετικό παλμό στο LCD_E πλάτους 12 κύκλων.
v) Αναμένουμε 100μs ή περισσότερο (5.000 κύκλοι των 50MHz).
vi) Γράφουμε SF_D<11:8> = 0x03, και δίνουμε θετικό παλμό στο LCD_E πλάτους 12 κύκλων.
vii) Αναμένουμε 40μs ή περισσότερο (2.000 κύκλοι των 50MHz).
viii) Γράφουμε SF_D<11:8> = 0x02, και δίνουμε θετικό παλμό στο LCD_E πλάτους 12 κύκλων.
ix) Αναμένουμε 40μs ή περισσότερο (2.000 κύκλοι των 50MHz).

Υλοποίηση:

Initialization_fsm module: Κάθε αριθμημένη πρόταση της εκφώνησης παραπάνω υλοποιείται ως ξεχωριστή κατάσταση σε μια μηχανή 10 συνολικά καταστάσεων. Η τελευταία κατάσταση DONE σημαίνει το τέλος της αρχικοποίησης.

Στο always block που ακολουθεί, ένας κοινός για όλες τις καταστάσεις μετρητής **clock_counter** ενεργοποιείται με το σήμα **counter_enable** και αναλόγως την τρέχουσα κατάσταση έχει και διαφορετική μέγιστη τιμή. Ένα σήμα **done = 1** στέλνεται κάθε φορά που ο μετρητής αγγίζει τη μέγιστη τιμή που του έχει ανατεθεί. Το σήμα done δηλώνει σε κάθε κατάσταση, εξαιρώντας την κατάσταση DONE, ότι ο απαιτούμενος χρόνος μετρήθηκε και η μηχανή πρέπει να μεταβεί στην επομένη κατάσταση.

```
//-----------------------ALWAYS BLOCK --> INITIALIZATION PROCESS-----------------------------------//

//Depending on current state, increment clock_counter to specific max value (-WHEN INITIALIZATION IS STILL ON PROGRESS-)

always @(posedge clk or posedge reset) begin

  if (reset) begin //reset
    clock_counter = 20'd0;
    done = 0;
  end
  else if ( clock_counter == 20'd750000 && next_state == INIT_1 ) begin //count 15ms for init state 1
    clock_counter = 20'd0;
    done = 1;
  end
  else if ( clock_counter == 20'd11 && ( next_state == INIT_2 || next_state == INIT_4 || next_state == INIT_6 || next_state == INIT_8 )) begin  //count 12 cycles for LCD_E == 1 -> init states 2,4,6,8
    clock_counter = 20'd0;
    done = 1;
  end
  else if ( clock_counter == 20'd204999 && next_state == INIT_3 ) begin //count 4.1ms for init state 3
    clock_counter = 20'd0;
    done = 1;
  end
  else if ( clock_counter == 20'd4999 && next_state == INIT_5 ) begin //count 100us for init state 5
    clock_counter = 20'd0;
    done = 1;
  end
  else if ( clock_counter == 20'd1999 && ( next_state == INIT_7 || next_state == INIT_9 )) begin  //count 40us for init states 7,9
    clock_counter = 20'd0;
    done = 1;
  end
  else if ( counter_enable == 1'b1 ) begin
    clock_counter = clock_counter + 1;
    done = 0;
  end

end
```

Η μηχανή των 10 συνολικά καταστάσεων φαίνεται παρακάτω. Η λειτουργία κάθε κατάστασης είναι η ακόλουθη:

- **INIT_1:** Αναμένουμε 15 ms.
- **INIT_2:** lcd_e_init (LCD_E) = 1 για 240 ns, init_data (SF_DATA<11:8>) = 0x03.
- **INIT_3:** Αναμένουμε 4.1ms.
- **INIT_4:** lcd_e_init (LCD_E) = 1 για 240 ns, init_data (SF_DATA<11:8>) = 0x03.
- **INIT_5:** Αναμένουμε 100us.
- **INIT_6:** lcd_e_init (LCD_E) = 1 για 240 ns, init_data (SF_DATA<11:8>) = 0x03.
- **INIT_7:** Αναμένουμε 40us.
- **INIT_8:** lcd_e_init (LCD_E) = 1 για 240 ns, init_data (SF_DATA<11:8>) = 0x02.
- **INIT_9:** Αναμένουμε 40us.
- **DONE:** Ολοκλήρωση αρχικοποίησης, initialization_done = 1.

```
always @(posedge clk or posedge reset) begin

    if (reset) begin
        // reset
        state = INIT_1;
    end
    else begin
        state = next_state;
    end

end

//States management

always @(*) begin

    next_state = state;

    counter_enable = 1'b0;
    lcd_e_init = 0;
    init_data = 4'h3;
    initialization_done = 1'b0;

    case (state)

        INIT_1: begin //wait 15ms

            counter_enable = 1'b1;

            if ( done == 1'b1 ) begin
                next_state = INIT_2;
                lcd_e_init = 1'b1;
                init_data = 4'h3;
            end

        end
        INIT_2: begin  //LCD_E = 1, 12 cycles, SF_D = 0x03

            counter_enable = 1'b1;
            lcd_e_init = 1'b1;
            init_data = 4'h3;

            if ( done == 1'b1 ) begin
                lcd_e_init = 1'b0;
                next_state = INIT_3;
            end

        end
        INIT_3: begin //wait 4.1ms

            counter_enable = 1'b1;

            if ( done == 1'b1 ) begin
                next_state = INIT_4;
                lcd_e_init = 1'b1;
                init_data = 4'h3;
            end

        end
        INIT_4: begin //LCD_E = 1, 12 cycles, SF_D = 0x03

            counter_enable = 1'b1;
            lcd_e_init = 1'b1;
            init_data = 4'h3;

            if ( done == 1'b1 ) begin
                lcd_e_init = 1'b0;
                next_state = INIT_5;
            end

        end
```

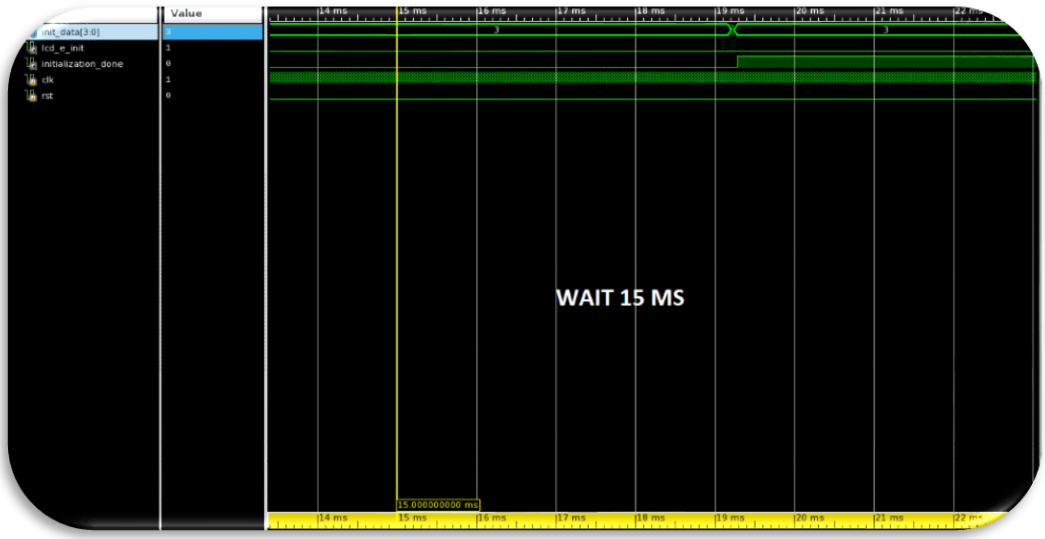
```
        INIT_5: begin //wait 100us

            counter_enable = 1'b1;

            if ( done == 1'b1 ) begin
                next_state = INIT_6;
                lcd_e_init = 1'b1;
                init_data = 4'h3;
            end

        end
        INIT_6: begin //LCD_E = 1, 12 cycles, SF_D = 0x03

            counter_enable = 1'b1;
            lcd_e_init = 1'b1;
            init_data = 4'h3;

            if ( done == 1'b1 ) begin
                lcd_e_init = 1'b0;
                next_state = INIT_7;
            end

        end
        INIT_7: begin //wait 40us

            counter_enable = 1'b1;
            init_data = 4'h2;

            if ( done == 1'b1 ) begin
                next_state = INIT_8;
                lcd_e_init = 1'b1;
            end

        end
        INIT_8: begin //LCD_E = 1, 12 cycles, SF_D = 0x02

            counter_enable = 1'b1;
            lcd_e_init = 1'b1;
            init_data = 4'h2;

            if ( done == 1'b1 ) begin
                lcd_e_init = 1'b0;
                next_state = INIT_9;
            end

        end
        INIT_9: begin //wait 40us

            counter_enable = 1'b1;

            if ( done == 1'b1 ) begin
                next_state = DONE;
                counter_enable = 1'b0;
            initialization_done = 1'b1;
            end

        end
        DONE: begin //Finish LCD Initialization

            counter_enable = 1'b0;
            initialization_done = 1'b1;

        end

    endcase
```

Επαλήθευση:

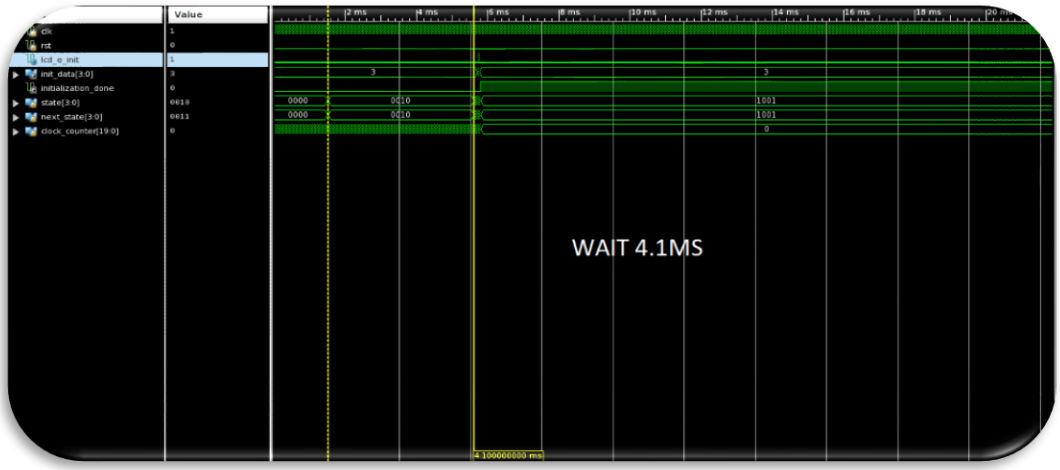
Για την επαλήθευση της ΜΠΚ της αρχικοποίησης, κατασκευάζεται ένα απλό στη δομή του πλαίσιο δοκιμής το οποίο κάνει instantiate τη μονάδα initialization_fsm και ελέγχει την λειτουργία της.

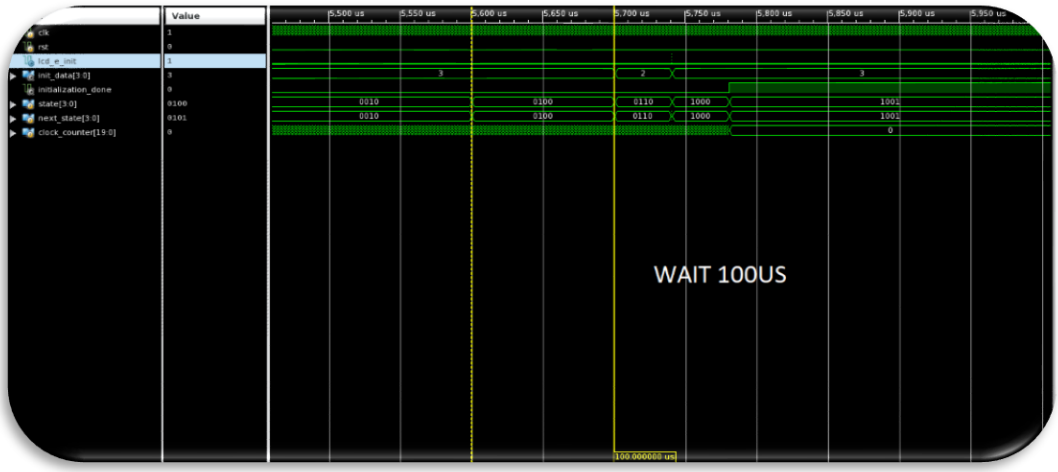
```
`timescale 1ns/1000ps

module initialization_tb ();

reg clk = 0;
reg rst = 1;

wire [3:0] init_data;

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

initial begin
    #50 rst=0;
end

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

always #10 clk <= ~clk;

//Test LCD initialization process of LCD Display

initialization_fsm initialization_fsm_0 ( .reset( rst ), .clk( clk ), .lcd_e_init( lcd_e_init ),
                                          .init_data( init_data ), .initialization_done( initialization_done ) );

endmodule
```

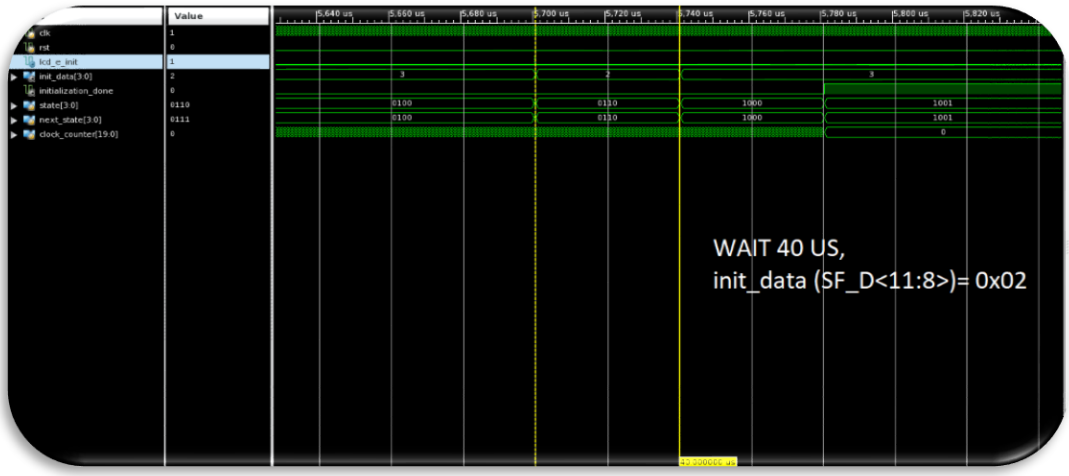
Κυματομορφές: Οι κυματομορφές της προσομοίωσης επαληθεύουν τις αναθέσεις και τους χρόνους που απαιτούν οι ενέργειες της αρχικοποίησης. Παρακάτω δίνονται μερικά στιγμιότυπα των κυματομορφών που δείχνουν τους χρόνους αναμονής, με τη σειρά που δίνονται στην εκφώνηση της εργασίας.

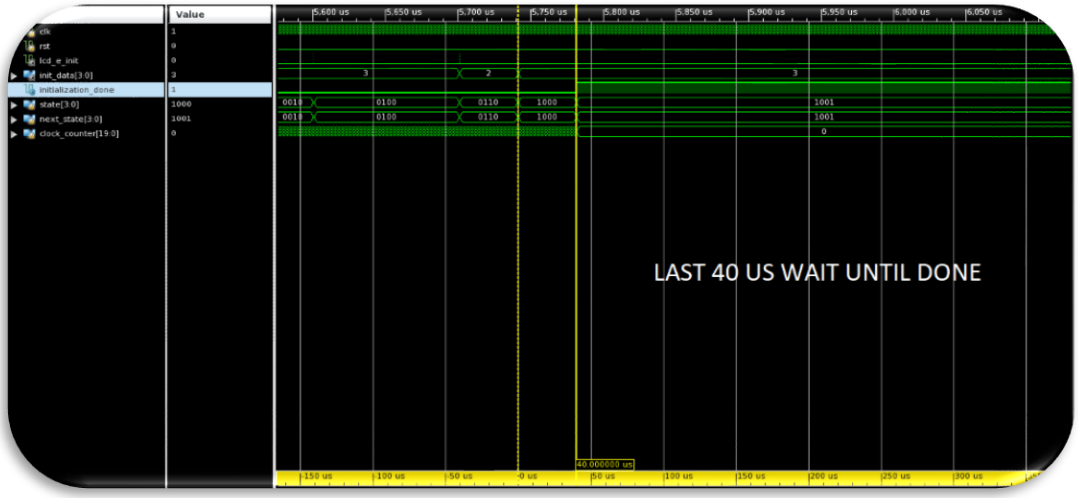



lcd_e_init (LCD_E) => 240 ns


WAIT 4.1MS


WAIT 100US





Κάλυψη λειτουργίας Η κάλυψη λειτουργίας φτάνει στο μέγιστο, καθώς τα διανύσματα εισόδου για την μονάδα της μηχανής καταστάσεων της αρχικοποίησης είναι μόνο το reset και το ρολόι, τα οποία οδηγούνται κατάλληλα.

Μέρος Γ – Υλοποίηση ΜΠΚ Διαμόρφωσης Ένδειξης και Αποστολής Δεδομένων

Η μονάδα που αναλαμβάνει την διαμόρφωση της ένδειξης αλλά και την αποστολή δεδομένων προς αυτή, έχει τον ρόλο της κεντρικής μηχανής καταστάσεων του συστήματος, καθώς αλληλοεπιδρά με τις 2 προηγούμενες ΜΠΚ, προβάλει το μήνυμα στην ένδειξη και προκαλεί την ανανέωση της οθόνης κάθε ένα δευτερόλεπτο.

Οι βασικές λειτουργίες της μονάδας configuration_fsm πραγματοποιούνται μέσα από ένα σύνολο καταστάσεων που απαρτίζει την ΜΠΚ στο εσωτερικό της. Οι κύριες ενέργειες της μονάδας είναι οι ακόλουθες:

- Πραγματοποιεί την διαδικασία διαμόρφωσης της ένδειξης την πρώτη φορά μετά την αρχικοποίηση της, αλλά και κάθε φορά που η οθόνη ανανεώνεται. Τα βήματα της διαμόρφωσης φαίνονται παρακάτω στην εικόνα που πάρθηκε αυτούσια από την εκφώνηση της εργασίας. Κάθε βήμα – αριθμημένη πρόταση μεταφράστηκε σε μια κατάσταση της μηχανής στον κώδικα Verilog της μονάδας.

Διαμόρφωση:

α′) Στέλνουμε εντολή Καθιέρωσης Λειτουργίας, 0x28, για να διαμορφώσουμε την Ένδειξη.

β′) Στέλνουμε εντολή Καθιέρωσης Κατάστασης, 0x06, προσδιορίζοντας ότι ο ελεγχτής θα αυξάνει αυτόματα την διεύθυνση.

γ′) Στέλνουμε εντολή Ενεργοποίησης Ένδειξης, 0x0C, ενεργοποιόντας την Ένδειξη, και σβήνοντας τον αυτόματο δρομέα.

δ′) Στέλνουμε εντολή Εκκαθάρισης Έκδειξης, για να καθαρίσει η οθόνη.

ε′) Αναμένουμε τουλάχιστον 1,64ms (82.000 κύκλοι των 50MHz).

Μετά τις δυο παραπάνω διαδικασίες μπορούμε να στείλουμε εντολή Ανάθεσης Διεύθυνσης και κατόπιν Εγγραφής δεδομένων στην DDRAM.

- Κάθε ένα δευτερόλεπτο η οθόνη πρέπει να ανανεώνεται και αυτό οπτικά πρέπει να φανεί με την αλλαγή του συμβόλου στην θέση του κέρσορα που βρίσκεται τέρμα δεξιά στην κάτω γραμμή της ένδειξης. Η ανανέωση αυτή πραγματοποιείται μέσα από τη μονάδα, αφού κάθε δευτερόλεπτο η μηχανή ξανατρέχει τις καταστάσεις που αφορούν την διαδικασία της διαμόρφωσης και ταυτόχρονα αλλάζει το σύμβολο του κέρσορα προσπελαύνοντας διαφορετική κάθε φορά θέση στη Block Ram.

- Τέλος η μηχανή αλληλοεπιδρά με τις προηγούμενες ΜΠΚ που περιεγράφηκαν, διότι ξεκινά την λειτουργία της μόνο όταν λάβει το σήμα initialization_done = 1 από την μηχανή αρχικοποίησης, αλλιώς μένει αδρανής. Επίσης, κάθε εντολή ή δεδομένο που στέλνεται προς την ένδειξη, πρέπει να τηρεί τους προαναφερθέντες χρονικούς περιορισμούς και συνεπώς για κάθε κατάσταση που αφορά εντολή διαμόρφωσης ή αποστολή χαρακτήρα, η μηχανή χρησιμοποιεί την μονάδα διαχείρισης των χρονικών περιορισμών (time_control_fsm).

Υλοποίηση:

configuration_fsm module: Η μονάδα περιέχει στο εσωτερικό της μηχανή 9 καταστάσεων, η οποία φαίνεται αναλυτικά στην παρακάτω εικόνα με τον κώδικα Verilog. Κατηγοριοποιώντας σε πρώτη φάση τις καταστάσεις έχουμε:

Κατάσταση αδράνειας της μονάδας: Κατάσταση IDLE. Στο στάδιο αυτό η μηχανή στέκεται αδρανής μέχρι να δοθεί το σήμα initialization_done = 1 που σημαίνει το τέλος της διαδικασίας της αρχικοποίησης.

Καταστάσεις που αφορούν εντολές της διαδικασίας διαμόρφωσης: Οι καταστάσεις FUNCTION_SET, ENTRY_MODE_SET, DISPLAY_ON_OFF, CLEAR_DISPLAY και SET_DDRAM_ADDRESS είναι όλες βήματα της διαδικασίας διαμόρφωσης έτσι όπως αυτή παρουσιάστηκε από την εκφώνηση.

Κατάσταση αποστολής δεδομένων: Κατάσταση DATA_TRANSFER. Στο στάδιο αυτό η μονάδα προσπελαύνει την Block Ram και στέλνει ένα προς ένα τα δεδομένα της μνήμης προς την ένδειξη.

Καταστάσεις αναμονής συγκεκριμένου χρονικού διαστήματος: Καταστάσεις WAIT_1_64_MS και WAIT_1_SEC. Στη πρώτη κατάσταση, όπως αναφέρεται στην διαδικασία της διαμόρφωσης, η μονάδα περιμένει αδρανής για 1,64 ms μέχρι να στείλει την εντολή ανάθεση διεύθυνσης. Στη δεύτερη κατάσταση, η μονάδα έχει πραγματοποιήσει την διαμόρφωση και έχει στείλει όλα τα δεδομένα στην ένδειξη. Συνεπώς, περιμένει αδρανής 1 δευτερόλεπτο για να ανανεώσει και πάλι την οθόνη.

```
IDLE: begin

   timing_control_enable = 1'b0; //Stay idle until initialization process is done.

   if ( initialization_done == 1'b1 ) begin //Proceed to LCD Configuration process (First command - Function Set)
      next_state = FUNCTION_SET;
      timing_control_enable = 1'b1; // Enable time_control_fsm module.
   end

end
FUNCTION_SET: begin

   if ( change_state == 1'b1 ) begin
      next_state = ENTRY_MODE_SET;
   end

end
ENTRY_MODE_SET: begin

   if ( change_state == 1'b1 ) begin
      next_state = DISPLAY_ON_OFF;
   end

end
DISPLAY_ON_OFF:  begin

   if ( change_state == 1'b1 ) begin
      next_state = CLEAR_DISPLAY;
   end

end
CLEAR_DISPLAY: begin

   if ( change_state == 1'b1 ) begin
      next_state = WAIT_1_64_MS;
   end

 end
WAIT_1_64_MS: begin //After the end of configuration wait 1,64 ms before data transfering.

   if ( send_upper == 1'b1 || clock_counter != 0 ) begin //keep clock_counter enable until max value of 8200 cycles - 1.64ms.
      timing_control_enable = 1'b0; //disable timing_control for the waiting process.
      counter_enable = 1'b1;
   end
   else if ( done == 1'b1 ) begin //when waiting process is finished proceed to next state.
      counter_enable = 1'b0;
      timing_control_enable = 1'b1; //enable again timing_control_fsm.
      next_state = SET__DDRAM_ADDRESS;
   end

end
SET__DDRAM_ADDRESS: begin //set DDRAM address to 0x00 address (Top Left Corner of Display).

   if ( change_state == 1'b1 ) begin
      next_state = DATA_TRANSFER;
   end

end
DATA_TRANSFER: begin

   if ( all_addresses_sent == 1'b1 ) begin //if all data were sent, proceed to second waiting state (1 second)
      next_state = WAIT_1_SEC;
      timing_control_enable = 1'b0;
      counter_enable = 1'b1;
   end

end
WAIT_1_SEC: begin

   if ( clock_counter != 0 ) begin
      timing_control_enable = 1'b0;
      counter_enable = 1'b1;
   end
   else if ( done == 1'b1 ) begin //when one second is passed, go to state IDLE and refresh display, by doing configuration p
      counter_enable = 1'b0;
      timing_control_enable = 1'b0;
      next_state = IDLE;
   end
```

Περιγραφή των καταστάσεων

Κατάσταση IDLE:

Με την αρχικοποίηση του κυκλώματος η μηχανή ξεκινά από την κατάσταση IDLE στην οποία η μονάδα μένει αδρανής. Το σήμα εισόδου initialization_done προέρχεται από την μηχανή initialization_fsm και δηλώνει την ολοκλήρωση της αρχικοποίησης της ένδειξης. Συνεπώς, μόλις το σήμα λάβει την τιμή 1, η μηχανή μεταβαίνει στην κατάσταση FUNCTION_SET, δηλαδή τη πρώτη σε σειρά ενέργεια στη διαδικασία διαμόρφωσης της ένδειξης.

Καταστάσεις FUNCTION_SET, ENTRY_MODE_SET, DISPLAY_ON_OFF, CLEAR_DISPLAY και SET_DDRAM_ADDRESS:

Οι καταστάσεις που περιγράφουν τις εντολές διαμόρφωσης ενεργοποιούν από την αδράνεια το παρακάτω always block. Ο 4-bit καταχωρητής config_data λαμβάνει την κατάλληλη τιμή των upper ή lower bits για την εκάστοτε εντολή (βλέπε τεχνικό δελτίο SPARTAN 3E για την δεκαεξαδική κωδικοποίηση κάθε εντολής). Τα σήματα εισόδου send_upper και send_lower προέρχονται από την μηχανή time_control_fsm και καθορίζουν ποια τετράδα bits πρέπει να ανατεθεί στον καταχωρητή. Ένα σήμα change_state = 1’b1 καθορίζει στο always block των καταστάσεων τον κύκλο που πρέπει η μηχανή να μεταβεί στην επόμενη κατάσταση.

```
//----------------------ALWAYS BLOCK --> FOR STATES OF CONFIGURATION LCD COMMANDS-------------------------//

//Always block affects ONLY states FUNCTION_SET, ENTRY_MODE_SET, DISPLAY_ON_OFF, CLEAR_DISPLAY, SET_DDRAM_ADDRESS.

//According to state - lcd_command, assign the right value to config_data register.

//According to input signal send_upper or send_lower, assign to config_data the right upper or lower bits.

//...(See Spartan-3E User - Guide for details about configuration commands data)...//

//signal change_state informs fsm to procceed to next state.

always @(posedge clk or posedge reset) begin

    if ( reset ) begin
        // reset
        config_data = 4'b0000;
        change_state = 1'b1;
    end
    else if ( send_lower == 1'b1 && next_state == FUNCTION_SET ) begin
        config_data = 4'b1000;
        change_state = 1'b1;
    end
    else if ( send_upper == 1'b1 && next_state == ENTRY_MODE_SET ) begin
        config_data = 4'b0000;
    end
    else if ( send_lower == 1'b1 && next_state == ENTRY_MODE_SET ) begin
        config_data = 4'b0110;
        change_state = 1'b1;
    end
    else if ( send_upper == 1'b1 && next_state == DISPLAY_ON_OFF ) begin
        config_data = 4'b0000;
    end
    else if ( send_lower == 1'b1 && next_state == DISPLAY_ON_OFF ) begin
        config_data = 4'b1100;
        change_state = 1'b1;
    end
    else if ( send_upper == 1'b1 && next_state == CLEAR_DISPLAY ) begin
        config_data = 4'b0000;
    end
    else if ( send_lower == 1'b1 && next_state == CLEAR_DISPLAY ) begin
        config_data = 4'b0001;
        change_state = 1'b1;
    end
    else if ( send_lower == 1'b1 && next_state == SET__DDRAM_ADDRESS ) begin
        config_data = 4'b0000;
        change_state = 1'b1;
    end
    else if ( next_state == SET__DDRAM_ADDRESS ) begin
        config_data = 4'b1000;
    end
    else if ( next_state == FUNCTION_SET ) begin
        config_data = 4'b0010;
    end
    else begin
        change_state = 1'b0;
    end

end

//-------------------------------------------------------------------------------------------------------//
```

Κατάσταση DATA_TRANSFER:

Η κατάσταση DATA_TRANSFER ενεργοποιεί από την αδράνεια το παρακάτω always block, υπεύθυνο για την αποστολή των δεδομένων του μηνύματος που βρίσκονται στη Block Ram. Ένας μετρητής address αντιστοιχεί στις θέσεις της μνήμης και αυξάνεται κάθε φορά που το σήμα send_upper είναι 1, γεγονός που δηλώνει τον κύκλο που πρέπει να σταλεί ο επόμενος χαρακτήρας στην ένδειξη. Κατά το πρώτο send_upper = 1, όταν το κύκλωμα βρίσκεται στην κατάσταση DATA_TRANSFER, το σήμα lcd_rs γίνεται 1 για να δηλώσει την ερχομένη αποστολή δεδομένων και όχι εντολών προς την ένδειξη.

Στην άλλη περίπτωση, όταν το σήμα send_lower γίνει 1, ο μετρητής address δεν αυξάνεται αλλά το σήμα lower_bits_time αλλάζει την τιμή του σε 1. Τότε, η ανάθεση assign, όπως φαίνεται στην εικόνα, υποδηλώνει ότι πρέπει να σταλούν στην ένδειξη τα lower bits του χαρακτήρα για την τρέχουσα διεύθυνση της μνήμης.

Όταν ο μετρητής address φτάσει στην θέση 54 της μνήμης, δηλαδή μια θέση πριν τα δεδομένα του κέρσορα, συμβουλεύεται το σήμα last_char_change. Όπως προαναφέρθηκε στο μέρος Α, η θέση 55 περιέχει τον χαρακτήρα του χρωματιστού ορθογωνίου παραλληλογράμμου (κωδικοποίηση 0xff), ενώ η θέση 56 αυτόν του κενού χαρακτήρα (κωδικοποίηση 0x20). Έτσι αναλόγως την τιμή του last_char_change η μνήμη προσπελαύνει μια από τις 2 θέσεις και αλλάζει την τιμή του last_char_change, ώστε στην επομένη ανανέωση να δείξει τον άλλο χαρακτήρα από αυτόν που έδειξε στην τρέχουσα.

Τέλος, όταν όλα οι χαρακτήρες του μηνύματος σταλούν προς την ένδειξη, ένα σήμα all_address_sent γίνεται 1 για έναν κύκλο και δηλώνει στην μηχανή ότι πρέπει να προχωρήσει στην κατάσταση WAIT_1_SEC.

```
//----------------------ALWAYS BLOCK --> FOR DATA_TRANFER STATE ONLY || BRAM INSTANTIATION---------------//

//1.Always-block affects ONLY the DATA_TRANSFER state.

//2.When in DATA_TRANSFER, after first send_upper signal, raise lcd_rs to 1.
//..Then, after every other send_upper or send_lower signal, increment address counter.

//3.address counter corresponds to Bram Memory address.

//4.Lower_bits_time register is equal to 1 when lower bits of 8-bit data are to be sent.
//..This signal indicates to character_data assignment to consider the lower 4 bits of bram memory for the given address.

//5.address = 11'd55 corresponds to "rectangle" character and address = 11'd56 corresponds to "space" character.
//..After every renewal of the LCD Display, last_char_change indicates what "else if" should be considered for the last lcd character,
//....depending on the previous visited "else if".

//6.Finaly, when all data are finally written in the LCD DISPLAY make all_addresses_sent singal equal to 1,
//..in order to inform the state machine to proceed to WAIT_1_SEC State.

always @(posedge clk or posedge reset) begin

    if (reset) begin
        // reset
        address = 11'd0;
        lcd_rs = 1'b0;
        all_addresses_sent = 1'b0;
        last_char_change = 1'b1;
        lower_bits_time = 1'b0;
    end
    else if ( all_addresses_sent == 1'b1 ) begin
        all_addresses_sent = 1'b0;
    end
    else if ( ( address == 11'd56 || address == 11'd55 ) && send_upper == 1'b1 && next_state == DATA_TRANSFER ) begin
        lcd_rs = 1'b0;
        address = 11'b0;
        all_addresses_sent = 1'b1;
        lower_bits_time = 1'b0;
    end
    else if ( last_char_change == 1'b1 && address == 11'd54 && lcd_rs == 1'b1 && send_upper == 1'b1 && next_state == DATA_TRANSFER ) begin
        address = address + 1;
        last_char_change = 1'b0;
        lower_bits_time = 1'b0;
    end
    else if ( last_char_change == 1'b0 && address == 11'd54 && lcd_rs == 1'b1 && send_upper == 1'b1 && next_state == DATA_TRANSFER ) begin
        address = address + 2;
        last_char_change = 1'b1;
        lower_bits_time = 1'b0;
    end
    else if ( lcd_rs == 1'b1 && send_upper == 1'b1 && next_state == DATA_TRANSFER  ) begin
        address = address + 1;
        lower_bits_time = 1'b0;
    end
    else if ( lcd_rs == 1'b1 && send_lower == 1'b1 && next_state == DATA_TRANSFER  ) begin
        lower_bits_time = 1'b1;
    end
    else if ( send_upper == 1'b1 && next_state == DATA_TRANSFER ) begin
        lcd_rs = 1'b1;
        all_addresses_sent = 1'b0;
        lower_bits_time = 1'b0;
    end

end


bram bram_0 (.clk(clk), .addr(address), .char(memory));

assign character_data = ( lower_bits_time == 1'b1  ) ? memory[3:0] : memory [7:4];
```

Καταστάσεις WAIT_1_64_MS και WAIT_1_SEC:

Οι καταστάσεις WAIT_1_64_MS και WAIT_1_SEC είναι καταστάσεις αναμονής της μονάδας και ενεργοποιούν το παρακάτω always block. Όπως φαίνεται στον κώδικα της Verilog, ένας μετρητής clock_counter ανάλογα την κατάσταση που ενεργοποίησε το block (μέσω του σήματος counter_enable = 1), μετρά είτε μέχρι το 1 δευτερόλεπτο είτε μέχρι τα 1.64 ms. Μόλις η τιμή του clock_counter φτάσει στο μέγιστο, ένα σήμα done = 1 δηλώνει στην εκάστοτε κατάσταση να προχωρήσει στην επόμενη.

Η κατάσταση WAIT_1_64_MS, με την ολοκλήρωση της αναμονής προχωρά στην κατάσταση ανάθεσης διεύθυνσης SET_DDRAM_ADDRESS. Αντίστοιχα, η μηχανή από την κατάσταση WAIT_1_SEC μεταβαίνει ξανά στην κατάσταση IDLE για να δρομολογηθεί η ανανέωση της οθόνης.

```
//-----------------------ALWAYS BLOCK --> FOR WAIT_1_64_MS & WAIT_1_SEC STATES ONLY-----------------------//

//Clock counter counts 1 sec when on state WAIT_1_SEC. done == 1 after finish.

//Clock counter counts 40 us when on state WAIT_1_64_MS. done == 1 after finish.

always @(posedge clk or posedge reset) begin

    if (reset) begin
        // reset
        clock_counter = 26'd0;
        done = 0;
    end
    else if ( done == 1 ) begin
        done = 0;
    end
    else if ( clock_counter == 26'd81999 && next_state == WAIT_1_64_MS ) begin
        clock_counter = 26'd0;
        done = 1;
    end
    else if ( clock_counter == 26'd49_999_999 && next_state == WAIT_1_SEC ) begin
        clock_counter = 26'd0;
        done = 1;
    end
    else if ( counter_enable == 1'b1 ) begin
        clock_counter = clock_counter + 1;
    end

end

//-------------------------------------------------------------------------------------------------------//
```

Επαλήθευση:

Η επαλήθευση της μονάδας θα πραγματοποιηθεί από το πλαίσιο δοκιμής ολόκληρου του LCD οδηγού στο μέρος Δ της αναφοράς. Αυτό συμβαίνει διότι η κεντρική μονάδα configuration_fsm πρακτικά εξαρτάται από όλες τις μονάδες που περιεγράφηκαν (bram, initialization_fsm, time_control_fsm), μονάδες οι οποίες μαζί με την πρώτη αποτελούν σχεδόν ολόκληρο το εσωτερικό του συστήματος του οδηγού. Συγκεκριμένα, όπως θα τονιστεί, τα μόνο επιπλέον modules που προστίθενται στη συνέχεια είναι το απλό κύκλωμα του reset_synchronizer για το συγχρονισμό του reset και ο signal_controller που αποτελεί ένα κύκλωμα πολυπλεκτών που διαχειρίζεται τα σήματα των μονάδων.

Μέρος Δ – Κατασκευή LCD Driver

> *Στο στάδιο της ένωσης μερών, ολοκληρώνεται ο οδηγός της ένδειξης LCD με την ένωση των μερών Α,Β,Γ σε ένα ενιαίο συνθέσιμο σύστημα. Με την ολοκλήρωση του μέρους Δ, ο οδηγός είναι σε θέση να προγραμματιστεί πάνω στη πλακέτα για να ελεγχθεί η λειτουργία του.*

Το μέρος Δ της εργασίας αποτελείται από 3 τμήματα:

- Ο **lcd_driver** 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 **signal_controller**, της οποίας η υλοποίηση αναθέτει στα σήματα LCD_E, LCD_RS, LCD_RW και SF_D<11:8> τις κατάλληλες τιμές. Πρακτικά, διαχειρίζεται τα σήματα εξόδου των μονάδων – μηχανών και τα συνδέει με τα σήματα εξόδου του ελεγκτή τη σωστή στιγμή.
- Αρχείο **UCF** με τις κατάλληλες συνδέσεις των σημάτων του οδηγού στα pins της πλακέτας.

Υλοποίηση:

lcd_driver module: Η μονάδα lcd_driver είναι το top-level module του συστήματος και κάνει instantiate σε επίπεδο Verilog όλες τις επιμέρους μονάδες του οδηγού. Οι μονάδες που γίνονται instantiate στο εσωτερικό του είναι οι εξής:

- reset_synchronizer
- initialization_fsm
- configuration_fsm
- time_control_fsm
- signal_controller

Ο κώδικας Verilog της μονάδας φαίνεται παρακάτω:

```
module lcd_driver( clk, rst, LCD_RS, LCD_RW, LCD_E, SF_D11, SF_D10, SF_D9, SF_D8 );

input clk, rst;
output LCD_RS, LCD_RW, LCD_E, SF_D11, SF_D10, SF_D9, SF_D8;

wire fsm_enable;
wire [3:0] init_data;
wire [3:0] config_data;
wire [3:0] character_data;

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

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

//Instantiation of initialization_fsm. Module that handles LCD initialization process.

initialization_fsm initialization_fsm_0 ( .reset( reset ), .clk( clk ), .lcd_e_init( lcd_e_init ),
                                          .init_data( init_data ), .initialization_done( initialization_done ) );

//Instantiation of configuration_fsm. Module that handles LCD configuration Process and data tranfer.

configuration_fsm configuration_fsm_0 ( .reset( reset ), .clk( clk ), .initialization_done( initialization_done ),
                                        .timing_control_enable( fsm_enable ), .send_upper( send_upper ), .send_lower( send_lower ),
                                        .lcd_rs( LCD_RS ), .config_data( config_data ), .character_data ( character_data ) );

//Instantiation of time_control_fsm. Module that handles LCD timing restrictions between the commands or data that are to be sent serially.

time_control_fsm time_control_fsm_0 (.reset( reset ), .clk( clk ), .fsm_enable( fsm_enable ), .lcd_e_tc_fsm( lcd_e_tc ), .send_upper( send_upper ), .send_lower( send_lower ) );

//Signal controller manages output signals of the other modules and choose the correct values for LCD Driver main outputs.

signal_controller signal_controller_0 ( .initialization_done ( initialization_done ), .LCD_RS( LCD_RS ),
                                        .lcd_e_init( lcd_e_init ), .lcd_e_tc( lcd_e_tc ),
                                        .init_data( init_data ), .config_data( config_data ), .character_data( character_data ),
                                        .SF_D11( SF_D11 ), .SF_D10( SF_D10 ), .SF_D9( SF_D9 ), .SF_D8( SF_D8 ),
                                        .LCD_RW( LCD_RW ), .LCD_E( LCD_E ) );

endmodule
```

signal_controller module: Ο κώδικας Verilog της μονάδας φαίνεται στην παρακάτω εικόνα.

Στο always block ο signal_controller αναθέτει στην τιμή του LCD_E την τιμή του lcd_e_init της μονάδας αρχικοποίησης initialization_fsm. Όταν η αρχικοποίηση τελειώσει (initialization_done = = 1'b1), το LCD_E παίρνει την τιμή lcd_e_tc που παράγει η μονάδα διαχείρισης των χρονικών περιορισμών (time_control_fsm).

Το σήμα LCD_RW αναθέτεται μέσω assign πάντα στο 0. Αυτό συμβαίνει γιατί η λειτουργία του οδηγού που υλοποιείται **δεν** απαιτεί σε κάποιο σημείο τη χρήση εντολών ανάγνωσης από την οθόνη (LCD_RW = 1).

Στα 4-bit command_data, αναθέτονται τα δεδομένα που αφορούν μόνο εντολές. Έτσι όταν πρόκειται για το στάδιο ακόμα της αρχικοποίησης, παίρνουν την τιμή των δεδομένων init_data από την μονάδα initialization_fsm. Όταν η αρχικοποίηση ολοκληρωθεί (initialization_done == 1'b1), τα δεδομένα command_data αρχίζουν και λαμβάνουν τα δεδομένα εντολών config_data της διαμόρφωσης από την μονάδα configuration_fsm.

Στη συνέχεια, τα 4-bit δεδομένων LCD_D αντιστοιχούν στα δεδομένα που εξέρχονται από τον οδηγό LCD και αυτά μπορεί να είναι είτε δεδομένα εντολών είτε δεδομένα-χαρακτήρες. Τον τύπο των δεδομένων καθορίζει το σήμα LCD_RS, το οποίο όταν βρίσκεται σε άνοδο, το LCD_D πρέπει να αντιστοιχεί σε χαρακτήρες (character_data), ενώ όταν βρίσκεται σε κάθοδο πρέπει να αντιστοιχεί σε εντολές (command_data).

Τέλος, τα 4-bit δεδομένα LCD_D χωρίζονται σε 1-bit σήματα, έτσι δηλαδή όπως πρέπει να σταλούν από τον οδηγό στην ένδειξη. Συγκεκριμένα, μέσω assign, τα LCD_D μοιράζονται στα SF_D11, SF_D10, SF_D9, SF_D8.

```
//If system is running under the initialization process, consider lcd_e_init from initialization_fsm as LCD_E signal.

always @(*) begin

    if ( initialization_done == 1'b1 ) begin
        LCD_E = lcd_e_tc;
    end
    else  begin
        LCD_E = lcd_e_init;
    end

end

//Because of the lack of need for reading operations, assign to LCD_RW value of 0.

assign LCD_RW = 1'b0;

//If system is running under the initialization process, consider init_data from initialization_fsm as the output data.

assign command_data = ( initialization_done == 1'b1 ) ? config_data : init_data ;

//LCD_RS indicates whether commands or data are to be sent. When LCD_RS == 1, LCD Driver output data should be the data from

assign  LCD_D = ( LCD_RS == 1'b1 ) ? character_data : command_data ;

//Seperate 4-bit LCD Data to 1-bit values.

assign SF_D11   = LCD_D[3];
assign SF_D10   = LCD_D[2];
assign SF_D9    = LCD_D[1];
assign SF_D8    = LCD_D[0];

endmodule
```

Αρχείο UCF συνδέσεων (lcd_driver.ucf): Παρακάτω δίνεται το αρχείο UCF με τις κατάλληλες αντιστοιχίσεις στη πλακέτα SPARTAN 3E.

```
NET clk LOC = C9;

NET "LCD_E"  LOC = "M18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "LCD_RS" LOC = "L18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "LCD_RW" LOC = "L17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;

# The LCD four-bit data interface is shared with the StrataFlash.

NET "SF_D8"  LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "SF_D9"  LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "SF_D10" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "SF_D11" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;

NET rst LOC = D18;

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

INST "configuration_fsm_0/bram_0/RAMB16_S9_inst" LOC = "RAMB16_X0Y9";
```

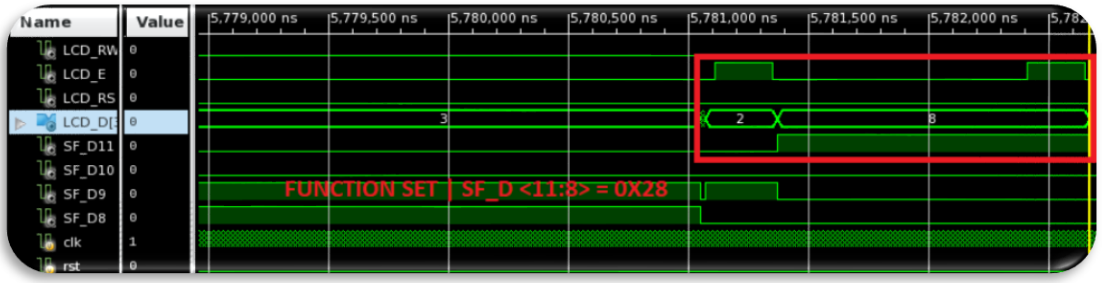
Επαλήθευση:

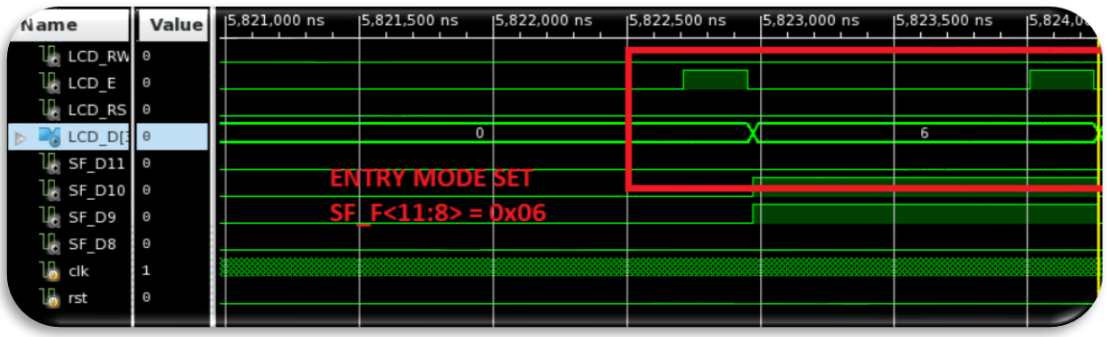
Για την επαλήθευση όλου του κυκλώματος μέσα από κυματομορφές πριν το τελικό πείραμα, κατασκευάζεται ένα πλαίσιο ελέγχου που είναι σε θέση να επαληθεύσει το σύστημα του οδηγού της ένδειξης LCD μετά το **Place & Route.**

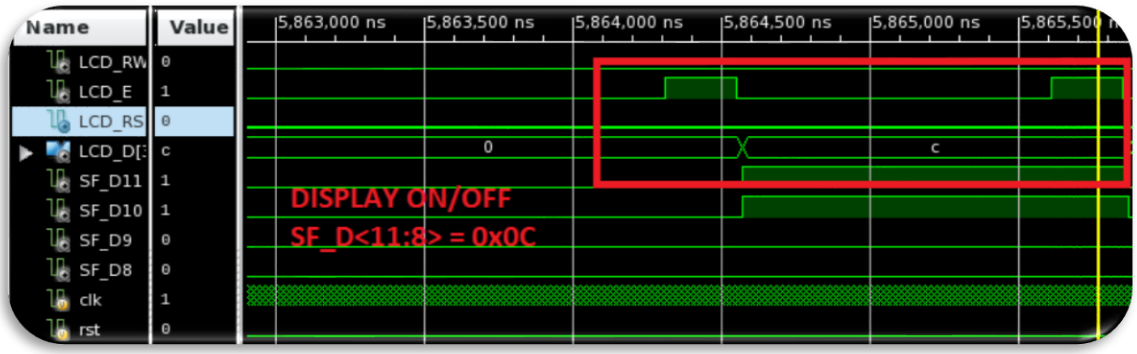
Το πλαίσιο δοκιμής είναι απλό και κάνει instantiate τον LCD Driver για να επαληθεύσει τις εξόδους του. Δίνεται παρακάτω:

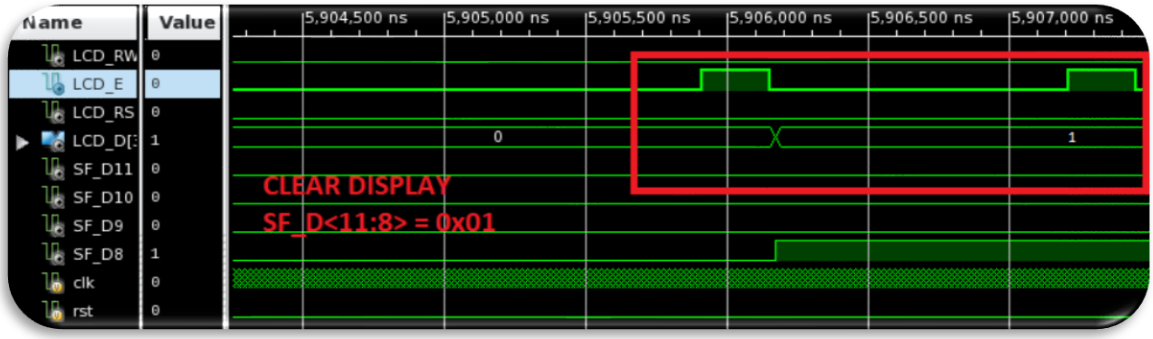
```
`timescale 1ns/1000ps

module lcd_driver_tb ();

reg clk = 0;
reg rst = 1;

initial begin
    #60 rst=0;
end

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

always #10 clk <= ~clk;

//Test LCD Driver

lcd_driver lcd_driver_0 (.clk( clk ), .rst( rst ),
                         .LCD_RS( LCD_RS ),.LCD_RW( LCD_RW ),.LCD_E( LCD_E ),
                         .SF_D11( SF_D11 ),.SF_D10( SF_D10 ),.SF_D9( SF_D9 ),.SF_D8( SF_D8 ) );

endmodule
```

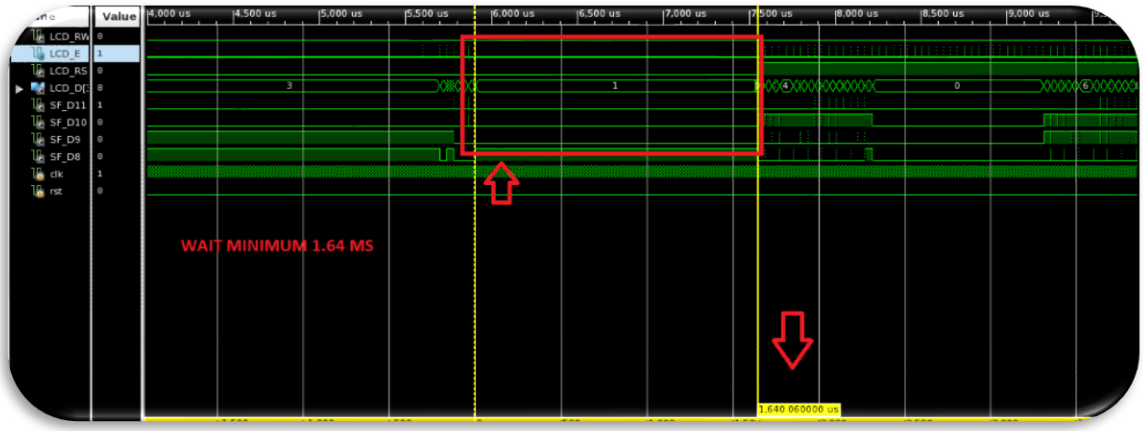
Κυματομορφές: Σε πρώτο στάδιο και μελετώντας τις παραγόμενες κυματομορφές, μπορεί να διαπιστωθεί η σωστή λειτουργία της διαμόρφωσης της ένδειξης που πραγματοποιείται από την μονάδα configuration_fsm. Παρακάτω δίνονται οι κυματομορφές με τις τιμές των δεδομένων LCD_D να αντιστοιχούν στα δεδομένα των εντολών της διαμόρφωσης.

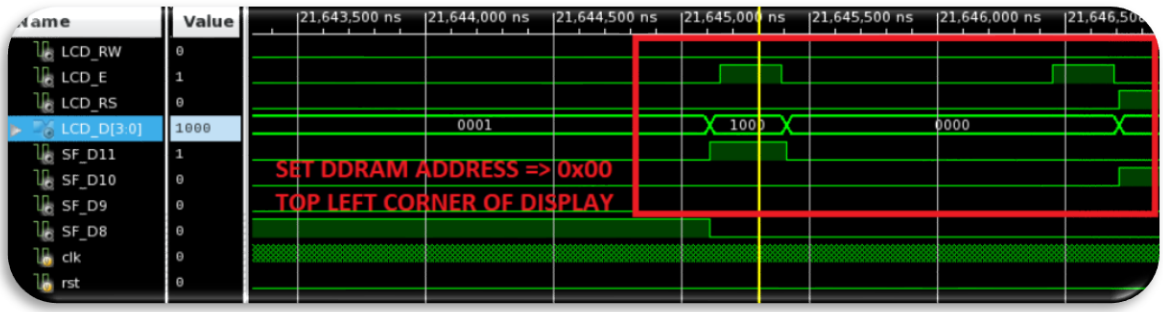




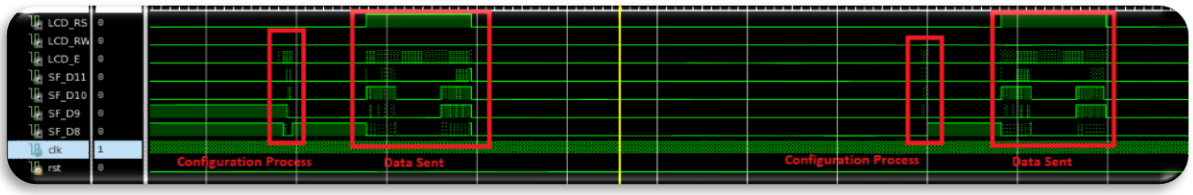

Name
Value
LCD_RW
LCD_E
LCD_RS
LCD_D[
SF_D11
SF_D10
SF_D9
SF_D8
clk
rst
5,863,000 ns
5,863,500 ns
5,864,000 ns
5,864,500 ns
5,865,000 ns
DISPLAY ON/OFF
SF_D<11:8> = 0x0C


Name
Value
LCD_RW
LCD_E
LCD_RS
LCD_D[
SF_D11
SF_D10
SF_D9
SF_D8
clk
rst
5,904,500 ns
5,905,000 ns
5,905,500 ns
5,906,000 ns
5,906,500 ns
5,907,000 ns
CLEAR DISPLAY
SF_D<11:8> = 0x01

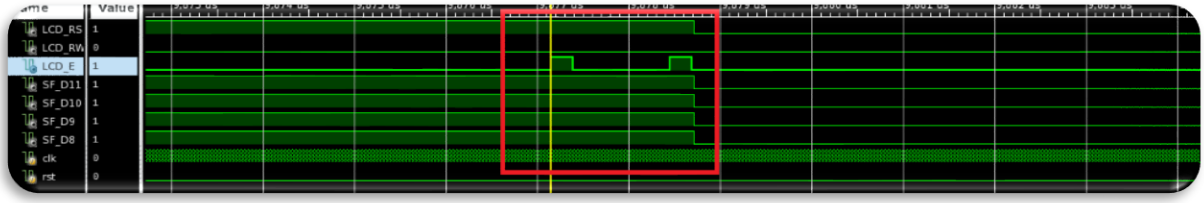

Value
WAIT MINIMUM 1.64 MS


Name
Value
LCD_RW
LCD_E
LCD_RS
LCD_D[3:0]
SF_D11
SF_D10
SF_D9
SF_D8
clk
rst
21,643,500 ns
21,644,000 ns
21,644,500 ns
21,645,000 ns
21,645,500 ns
21,646,000 ns
0001
1000
0000
SET DDRAM ADDRESS => 0x00
TOP LEFT CORNER OF DISPLAY

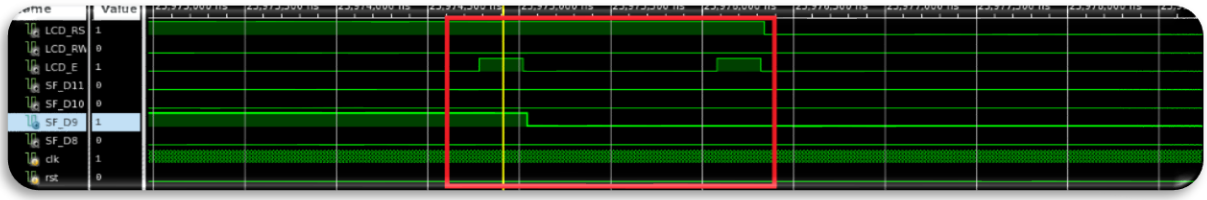
Οι κυματομορφές πέρα από την διαδικασία της διαμόρφωσης επαληθεύουν και την ορθή αποστολή των δεδομένων – χαρακτήρων προς την ένδειξη. Στη πρώτη από τις κυματομορφές που ακολουθούν, φαίνεται η λειτουργία του κυκλώματος για 2 ανανεώσεις της οθόνης.



Στην επόμενη κυματομορφή, μέσα από αρκετή μεγέθυνση μπορεί να δειχθεί ότι στην πρώτη αποστολή δεδομένων ο τελευταίος χαρακτήρας που στέλνεται, δηλαδή ο κέρσορας, έχει την τιμή 0xff (Upper Bits = 1111, Lower Bits = 1111) που είναι το σύμβολο του χρωματιστού ορθογωνίου παραλληλογράμμου.



Από την άλλη, στην επόμενη ανανέωση της οθόνης που έπεται της αποστολής δεδομένων της προηγούμενης κυματομορφής, μπορεί να φανεί ότι ο κέρσορας έχει την τιμή 0x20 (Upper Bits = 0010, Lower Bits = 0000) που αντιστοιχεί στον κενό χαρακτήρα.



Κάλυψη λειτουργίας: Δεδομένου ότι τα διανύσματα ελέγχου του lcd_driver είναι μόνο το reset και το ρολόι, η κάλυψη λειτουργίας είναι η μέγιστη. Το κύκλωμα αφού οδηγηθούν τα 2 σήματα, ελέγχεται πλήρως και δεν υπάρχει κάποιο κομμάτι του κώδικα που απαιτεί διαφορετική μεταχείριση των εισόδων.

Τελικό Πείραμα Στην Πλακέτα

Το κύκλωμα με την ολοκλήρωση των μερών Α, Β, Γ, Δ βρίσκεται στην συνθέσιμη μορφή του και μπορεί να προγραμματιστεί στη πλακέτα SPARTAN 3E με το εξαγόμενο bit file από το λογισμικό ISE και το αρχείο UCF.

Αποτελέσματα πειράματος: Τα αποτελέσματα του πειράματος αποδεικνύουν ότι ο οδηγός της ένδειξης LCD λειτουργεί ορθά και παράγει το σωστό μήνυμα. Η οθόνη ανανεώνεται κάθε ένα δευτερόλεπτο και ο κέρσορας δείχνει εναλλάξ τον κενό χαρακτήρα και το ορθογώνιο παραλληλόγραμμο. Τα αποτελέσματα του πειράματος φαίνονται στις παρακάτω εικόνες.

Αλλαγές στον κώδικα που προέκυψαν από την πρακτική δοκιμή: Η πρακτική δοκιμή δεν επέφερε ουσιαστικά την ανάγκη να πραγματοποιηθούν δομικές αλλαγές στον κώδικα του συστήματος. Το κύκλωμα δούλεψε στο επιθυμητό σημείο στη δεύτερη προσπάθεια που δοκιμάστηκε. Αυτό συνέβη διότι κατά τη πρώτη προσπάθεια ο μετρητής που έχει τον ρόλο να μετρά το 1 δευτερόλεπτο της ανανέωσης μετρούσε στην πραγματικότητα 10 ms, οπότε ο κέρσορας άλλαζε τιμή με υπερβολική ταχύτητα, Τα 10 ms που είχαν επιβληθεί, οφείλονταν στην ανάγκη να διευκολυνθεί η εξέταση της ανανέωσης μέσα από τις κυματομορφές, καθώς το 1 δευτερόλεπτο είναι αρκετά μεγάλο και αργό διάστημα κατά τη προσομοίωση.

Συμπέρασμα

Με το τέλος της περιγραφής των τμημάτων της εργασίας, ολοκληρώνεται η τεχνική αναφορά που σχετίζεται με την υλοποίηση του οδηγού της ένδειξης LCD. Παρακάτω δίδεται το συμπέρασμα που εξάγεται μετά το τέλος της υλοποίησης του οδηγού και την υλοποίηση της εργασίας.

Η τέταρτη εργαστηριακή άσκηση στο σύνολο της και αφού επήλθε η ολοκλήρωση της αποτέλεσε διαδικασία αυξημένης δυσκολίας.

Η περιπλοκότητα γύρω από την κατανόηση της λειτουργίας της ένδειξης LCD ήταν αρκετά μεγάλη, καθώς έπρεπε να γίνει κατανοητός ο τρόπος με τον οποίο η ένδειξη διαχειρίζεται τις εντολές και τα δεδομένα, αλλά και τις κωδικοποιήσεις αυτών.

Επιπρόσθετα, οι χρονικοί περιορισμοί που ήταν αναγκαίο να επιβληθούν στο κύκλωμα, κατέστησαν αναγκαία την υλοποίηση της μονάδας της διαχείρισης αυτών (time_control_fsm), η οποία θα έπρεπε να πετύχει αυστηρά τους χρόνους που εμποδίζουν τα φαινόμενα παραβίασης SETUP και HOLD.

Παράλληλα, η μονάδα της αρχικοποίησης της ένδειξης (initialization_fsm) δεν παρουσίασε ιδιαίτερη δυσκολία στο να υλοποιηθεί, ενώ αντίθετα αρκετό συλλογισμό προκάλεσε η κατασκευή της κεντρικής μηχανής (configuration_fsm), η οποία θα έπρεπε όχι μόνο να αλληλοεπιδρά κατάλληλα με τις άλλες μηχανές καταστάσεων, αλλά και να στέλνει σωστά δεδομένα προς την ένδειξη με κάθε ανανέωση της οθόνης.

Τέλος, με την ολοκλήρωση της γραφής του κώδικα Verilog και την επαλήθευση της σωστής λειτουργίας του κυκλώματος μέσα από τις κυματομορφές, το πείραμα στην πλακέτα δεν επέφερε κάποιο νέο εμπόδιο στην υλοποίηση και συνεπώς η πρακτική δοκιμή εκτελέστηκε επιτυχώς.

ΤΕΛΟΣ ΑΝΑΦΟΡΑΣ