ΕΡΓΑΣΤΗΡΙΟ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΤΕΧΝΙΚΗ ΑΝΑΦΟΡΑ 1ΗΣ ΕΡΓΑΣΙΑΣ
ΤΙΤΛΟΣ: ΟΔΗΓΟΣ ΕΝΔΕΙΞΗΣ 7-ΤΜΗΜΑΤΩΝ

ΠΑΝΑΓΙΩΤΗΣ ΑΝΣΤΑΣΙΑΔΗΣ 2134 **25-10-2018**

1. Περίληψη Τεχνικής Αναφοράς

Το περιεχόμενο της παρόν τεχνική αναφοράς περιγράφεται περιληπτικά με βάση τα παρακάτω τμήματα ως εξής:

- Εισαγωγή στην αναφορά με την ανάλυση των στόχων της 1ης εργασίας και στον τρόπο που αυτοί επιτεύχθηκαν βάσει της τελικής υλοποίησης του συστήματος.
- Μέρος Α Εργασίας: Σχεδιασμός του αποκωδικοποιητή LED των 7-τμημάτων. Ως είσοδο, εφαρμόστηκε καταλλήλως να δέχεται 37 ψηφία, συγκεκριμένα όλα τα γράμματα της λατινικής αλφαβήτου (abc) συν το κενό και τα δεκαδικά ψηφία 0-9.
- Μέρος Β Εργασίας: Οδήγηση τεσσάρων σταθερών ψηφίων πάνω στη πλακέτα. Το μήνυμα των ψηφίων που υλοποιήθηκε είναι το “FPgA”.
- Μέρος Γ Εργασίας: Βηματική περιστροφή ενός μηνύματος πάνω στη πλακέτα με το πάτημα ενός κουμπιού. Τα 4 ψηφία είναι μεταβλητά και παρουσιάζουν με κάθε χρήση του κουμπιού διαδοχικά το μήνυμα 15 ψηφίων που επιλέχθηκε. Το μήνυμα εδώ είναι το “FPgA SPArtAn 3 ”.
- Μέρος Δ Εργασίας: Βηματική περιστροφή του μηνύματος πάνω στη πλακέτα αυτόματα και με μια σταθερή καθυστέρηση. Η καθυστέρηση αυτή που μεσολαβεί πριν από κάθε επόμενη προβολή ενός τμήματος του μηνύματος (ένα τμήμα ισοδυναμεί με 4 ψηφία του μηνύματος, όσα δηλαδή μπορεί να δείξει κάθε στιγμή η οθόνη) είναι περίπου 1,3421 δευτερόλεπτα. Το μήνυμα εδώ είναι και πάλι το “FPgA SPArtAn 3 ”.
- Συμπέρασμα: Εξαγωγή συμπεράσματος ως προς τα σύνολο της εργασίας και συγκεκριμένα τη σχεδίαση, την επαλήθευση και τις δυσκολίες της υλοποίησης των κυκλωμάτων.

2. Στόχοι της 1ης εργασίας

Ζητούμενο της 1ης εργασίας αποτέλεσε η υλοποίηση ενός οδηγού των τεσσάρων ενδείξεων 7-τμημάτων LED της πλακέτας Spartan 3, με σκοπό να πραγματοποιηθεί η περιστροφική παρουσίαση ενός μηνύματος.

Στη 1η εργασία, όπως αναφέρθηκε, κύριος σκοπός ήταν η κατάλληλη οδήγηση με 3 διαφορετικούς τρόπους της οθόνης 4 ενδείξεων και 7 τμημάτων LED η καθεμία, της πλακέτας Spartan 3. Μετά την επιτυχή εξέταση της λειτουργίας του κυκλώματος και στα 4 μέρη της εργασίας από τους βοηθούς του εργαστηρίου επιβεβαιώνεται πως όλα τα τμήματα υλοποιήθηκαν επιτυχώς.

Στο πρώτο μέρος, στόχος ήταν η αποκωδικοποίηση – μετάφραση των ψηφίων που επιλέχθηκαν ως έγκυρα, σε κατάλληλα και ευανάγνωστα σχήματα χρησιμοποιώντας τα 7 τμήματα των ενδείξεων της οθόνης. Για να επιτευχθεί αυτό, στα ψηφία 0-9 ακολουθήθηκε η ευρέως διαδεδομένη και προ υπάρχουσα δομή τους ως ψηφία 7-τμημάτων.Παρόλα αυτά για να παρουσιαστεί κάθε γράμμα της λατινικής αλφαβήτου έπρεπε να γίνει μια μίξη κεφαλαίων και μικρών γραμμάτων, καθώς και κάποιες συμβάσεις ως προς το παρουσιαστικό τους στη LED οθόνη,

καθώς γράμματα όπως το “k,w,t ” ήταν δύσκολο να προβληθούν πανομοιότυπα με τα σύμβολα που τα αντιπροσωπεύουν στη πραγματικότητα.

Για το δεύτερο μέρος της εργασίας, σκοπός ήταν η συνεχής και σταθερή προβολή ενός μηνύματος για τα 4 ψηφία της οθόνης χωρίς το μήνυμα αυτό να αλλάζει κατά τη διάρκεια της λειτουργίας. Για να έρθει εις πέρας το μέρος β, αρχικά ενσωματώθηκε στο κύκλωμα ένας DCM (Digital Control Manager), ο οποίος διαιρεί το ρολόι 16 φορές για να πετύχει τη καθυστέρηση φόρτισης των 0,32 μsec που απαιτεί η οδήγηση κάθε ψηφίου. Στη συνέχεια, όπως θα τονιστεί αναλυτικά παρακάτω χρησιμοποιήθηκε ένας μετρητής 4-bit, με βάση των οποίο οδηγούνται κατάλληλα τόσο τα σήματα an3 , an2 , an1, an0 όσο και τα σήματα των 7 τμημάτων a,b,c,d,e,f,g κάθε ψηφίου.

Στο τρίτο μέρος, ένα μήνυμα έπρεπε να περιστρέφεται βηματικά στην οθόνη με τη χρήση ενός κουμπιού και όταν φτάνει στο τέλος του να ξεκινά από την αρχή κυκλικά. Η υλοποίηση του μέρους Γ έγινε με τη χρήση μνήμης στην οποία μένει αποθηκευμένο το μήνυμα κα ένας μετρητής δείχνει κάθε φορά στη διεύθυνση της μνήμης. Η οδήγηση των σημάτων anX και a,b,c,d,e,f,g παραμένει ίδια με το μέρος Β. Τέλος χρησιμοποιείται μια μονάδα Anti-Bounce για να ελέγχεται κατάλληλα το φαινόμενο αναπηδήσεων στο πάτημα του κουμπιού.

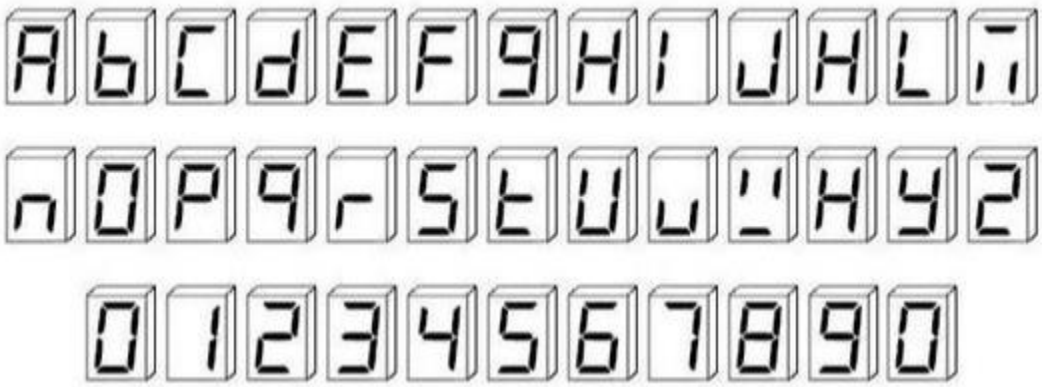
Στο τέταρτο και τελευταίο μέρος ζητήθηκε η βηματική περιστροφή του μηνύματος να πραγματοποιείται αυτόματα και βάσει σταθερής καθυστέρησης, αντί της χρήσης κουμπιού. Για το μέρος Δ διαφοροποιήθηκε η υλοποίηση του προηγούμενο μέρους της εργασίας, ώστε τη λειτουργία του κουμπιού να αντικαταστήσει ένας καινούριος μετρητής 22-bit, ο οποίος αυξάνει τον μετρητή που δείχνει στη διεύθυνση της μνήμης κάθε περίπου 1,3421 δευτερόλεπτα. Η δομή οδήγησης των anX και a,b,c,d,e,f,g παραμένει και εδώ η ίδια.

3. Μέρος Α – Υλοποίηση Αποκωδικοποιητή 7-τμημάτων

Ο LED Decoder δέχεται ως έγκυρες εισόδους:

- *Την λατινική αλφάβητο (ABC).*
- *Τα δεκαδικά ψηφία 0-9.*
- *Τον κενό χαρακτήρα « ».*

Η απεικόνιση των ψηφίων πάνω στην οθόνη βασίστηκε στην παρακάτω εικόνα, η οποία περιγράφει έναν από τους μερικούς τρόπους που μπορεί να σχεδιαστεί ψηφιακά η λατινική αλφάβητος. Στην εικόνα αυτή δε συμπεριλαμβάνεται ο κενός χαρακτήρας που επίσης υλοποιήθηκε, είναι όμως προφανής ο τρόπος απεικόνισης του στην οθόνη της πλακέτας (a,b,c,d,e,f,g τμήματα όλα οδηγημένα στο 1).



Υλοποίηση

Η υλοποίηση του LED αποκωδικοποιητή είναι αρκετά απλή, αποτελεί μια και μόνο μονάδα και σχεδιάζεται όπως παρακάτω:

```
parameter

zero            =  6'b000000,
one             =  6'b000001,
two             =  6'b000010,
three           =  6'b000011,
four            =  6'b000100,
five            =  6'b000101,
six             =  6'b000110,
seven           =  6'b000111,
eight           =  6'b001000,
nine            =  6'b001001,
letter_A        =  6'b001010,
letter_b        =  6'b001011,
letter_C        =  6'b001100,
letter_d        =  6'b001101,
letter_E        =  6'b001110,
letter_F        =  6'b001111,
letter_g        =  6'b010000,
letter_H        =  6'b010001,
letter_I        =  6'b010010,
letter_J        =  6'b010011,
letter_k        =  6'b010100,
letter_L        =  6'b010101,
letter_M        =  6'b010110,
letter_n        =  6'b010111,
letter_O        =  6'b011000,
letter_P        =  6'b011001,
letter_q        =  6'b011010,
letter_r        =  6'b011011,
letter_S        =  6'b011100,
letter_T        =  6'b011101,
letter_U        =  6'b011110,
letter_v        =  6'b011111,
letter_W        =  6'b100000,
letter_X        =  6'b100001,
letter_y        =  6'b100010,
letter_Z        =  6'b100011,
space           =  6'b100100;
```

```
always @( char ) begin
  case ( char )
     zero:          LED = 7'b0000001;
     one:           LED = 7'b1001111;
     two:           LED = 7'b0010010;
     three:         LED = 7'b0000110;
     four:          LED = 7'b1001100;
     five:          LED = 7'b0100100;
     six:           LED = 7'b0100000;
     seven:         LED = 7'b0001111;
     eight:         LED = 7'b0000000;
     nine:          LED = 7'b0000100;
     letter_A:      LED = 7'b0001000;
     letter_b:      LED = 7'b1100000;
     letter_C:      LED = 7'b0110001;
     letter_d:      LED = 7'b1000010;
     letter_E:      LED = 7'b0110000;
     letter_F:      LED = 7'b0111000;
     letter_g:      LED = 7'b0000100;
     letter_H:      LED = 7'b1001000;
     letter_I:      LED = 7'b1111001;
     letter_J:      LED = 7'b1000011;
     letter_k:      LED = 7'b1110000;
     letter_L:      LED = 7'b1110001;
     letter_M:      LED = 7'b0101011;
     letter_n:      LED = 7'b1101010;
     letter_O:      LED = 7'b0000001;
     letter_P:      LED = 7'b0011000;
     letter_r:      LED = 7'b1111010;
     letter_S:      LED = 7'b0100100;
     letter_T:      LED = 7'b1001110;
     letter_U:      LED = 7'b1000001;
     letter_v:      LED = 7'b1100011;
     letter_W:      LED = 7'b1010101;
     letter_X:      LED = 7'b1001000;
     letter_y:      LED = 7'b1000100;
     letter_Z:      LED = 7'b0010010;
     space:         LED = 7'b1111111;

// If input does not contain valid character, display error character in display ("-").

     default:       LED = 7'b1111110;
  endcase
end
```

- Στις παραμέτρους δεξιά αριθμούμε με έναν 6-bit αριθμό με τη σειρά τα 0-9 , την αλφάβητο ABC και το κενό χαρακτήρα.
- Παρακάτω στο κώδικα και συγκεκριμένα στο always block (αριστερά), κάθε φορά που η είσοδος char δέχεται μια νέα τιμή, αν είναι έγκυρη βγαίνει στην έξοδο η αντίστοιχη αποκωδικοποίηση της ως προς τα 7-τμήματα της οθόνης a,b,c,d,e,f,g.
- Αν δεν είναι έγκυρη η είσοδος, αποκωδικοποιείται ο κατά σύμβαση “error” χαρακτήρας η «-». Αυτός αντιστοιχεί στην έξοδο a,b,c,d,e,f = 1 και g = 0.

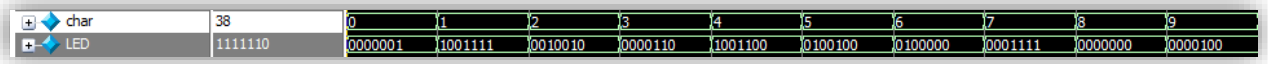
Επαλήθευση

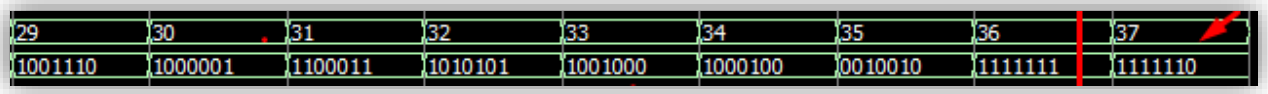
Για την επαλήθευση του κυκλώματος υλοποιήθηκε ένα απλό testbench. Όπως φαίνεται παρακάτω τροφοδοτείται η είσοδος του LED Decoder σειριακά και με καθυστέρηση 5 ns, από την τιμή 0 μέχρι την τιμή 37, έτσι ώστε στην έξοδο να εμφανιστούν όλες οι 37 έγκυρες είσοδοι και για το τελευταίο μη έγκυρο αριθμό 37 να εμφανιστεί και η παύλα «-» που αντιστοιχεί σε κάθε μη έγκυρη είσοδο.

```
module LEDdecoder_tb ();


reg [5:0] char = 6'b000000;
wire [6:0] LED;

initial begin
 repeat (38)
    #5 char = char + 1;
    $finish;
end

LEDdecoder LEDdecoder_0 (.char(char), .LED (LED));

endmodule
```

Τα αποτελέσματα της προσομοίωσης συμπεριφοράς ακολουθούν παρακάτω. Στο πρώτο στιγμιότυπο φαίνεται η σωστή αντιστοίχιση των 0-9 αριθμών στην 7-τμηματική απεικόνιση τους.

Στο δεύτερο στιγμιότυπο φαίνεται το τέλος της προσομοίωσης όπου ο τελευταίος χαρακτήρας είναι η παύλα.





Κάλυψη

Για την επαλήθευση του κυκλώματος με το συγκεκριμένο testbench υπάρχει 96.1% κάλυψη χρησιμοποιώντας τη λειτουργία coverage του Questa Sim. Η κοντά στο 100% κάλυψη ήταν αναμενόμενη, διότι ελέγχονται όλες οι πιθανοί είσοδοι που μπορεί να δεχτεί το κύκλωμα.

```
# ================================Toggle Details================================
#
# Toggle Coverage for ALL instances --
#                                              Node      1H->0L      0L->1H  "Coverage"
#                                              ----------------------------------------
#               /LEDdecoder_tb/LEDdecoder_0/char[0]           1           1      100.00
#               /LEDdecoder_tb/LEDdecoder_0/char[1]           1           1      100.00
#               /LEDdecoder_tb/LEDdecoder_0/char[2]           1           1      100.00
#               /LEDdecoder_tb/LEDdecoder_0/char[3]           1           1      100.00
#               /LEDdecoder_tb/LEDdecoder_0/char[4]           1           1      100.00
#               /LEDdecoder_tb/LEDdecoder_0/char[5]           0           1       50.00
#                /LEDdecoder_tb/LEDdecoder_0/LED[6]           1           1      100.00
#                /LEDdecoder_tb/LEDdecoder_0/LED[5]           1           1      100.00
#                /LEDdecoder_tb/LEDdecoder_0/LED[4]           1           1      100.00
#                /LEDdecoder_tb/LEDdecoder_0/LED[3]           1           1      100.00
#                /LEDdecoder_tb/LEDdecoder_0/LED[2]           1           1      100.00
#                /LEDdecoder_tb/LEDdecoder_0/LED[1]           1           1      100.00
#                /LEDdecoder_tb/LEDdecoder_0/LED[0]           1           1      100.00
#
# Total Node Count     =         13
# Toggled Node Count   =         12
# Untoggled Node Count =          1
#
# Toggle Coverage      =       96.1% (25 of 26 bins)
```

4. Μέρος Β – Οδήγηση Τεσσάρων Ψηφίων

Στο μέρος Β οδηγούνται τα 4 ψηφία σταθερά με το μήνυμα "FPgA". Για την εναλλάξ οδήγηση των ψηφίων εφαρμόζεται κάθε φορά μια σταθερή καθυστέρηση 0,32 μsec, ώστε να επιτυγχάνεται η φόρτιση τους. Αυτό γίνεται διαιρώντας τη συχνότητα 50 MHz (20ns) στο ρολόι της πλακέτας στη τιμή 3.125 MHz.

Υλοποίηση

Το μέρος Β της εργασίας, στο τελικό στάδιο της υλοποίησης του αποτελείται από τις εξής μονάδες:

Topmodule: Το top-level επίπεδο της υλοποίησης το οποίο περιέχει τα inputs / outputs του συνολικού κυκλώματος και κάνει instantiate τις επιμέρους μονάδες, δηλαδή τα υπό-κυκλώματα.

```
module Topmodule (reset, clk, an3, an2, an1, an0,
a, b, c, d, e, f, g, dp);

input clk, reset;
output an3, an2, an1, an0;
output a, b, c, d, e, f, g, dp;

wire an3, an2, an1, an0;
wire a, b, c, d, e, f, g,dp;
wire [6:0] LED;
wire [1:0] enable;
wire [5:0]charToDecode;
wire [3:0] counter;
wire reset_sync;

//Synchronize reset using two flip flops to avoid setup and hold violations.
reset_synchronizer reset_synchronizer_0 (reset,clk, reset_sync);

 DCM #(
      .SIM_MODE("SAFE"),  // Simulation: "SAFE" vs. "FAST", see "Synthesis and Simulation Design Guide" for details
      .CLKDV_DIVIDE(16.0), // Divide by: 1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
                           //   7.0,7.5,8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0 or 16.0
      .CLKFX_DIVIDE(1),   // Can be any integer from 1 to 32
      .CLKFX_MULTIPLY(4), // Can be any integer from 2 to 32
      .CLKIN_DIVIDE_BY_2("FALSE"), // TRUE/FALSE to enable CLKIN divide by two feature
      .CLKIN_PERIOD(0.0),  // Specify period of input clock
      .CLKOUT_PHASE_SHIFT("NONE"), // Specify phase shift of NONE, FIXED or VARIABLE
      .CLK_FEEDBACK("1X"),  // Specify clock feedback of NONE, 1X or 2X
      .DESKEW_ADJUST("SYSTEM_SYNCHRONOUS"), // SOURCE_SYNCHRONOUS, SYSTEM_SYNCHRONOUS or
                                            //   an integer from 0 to 15
      .DFS_FREQUENCY_MODE("LOW"),  // HIGH or LOW frequency mode for frequency synthesis
      .DLL_FREQUENCY_MODE("LOW"),  // HIGH or LOW frequency mode for DLL
      .DUTY_CYCLE_CORRECTION("TRUE"), // Duty cycle correction, TRUE or FALSE
      .FACTORY_JF(16'hC080),   // FACTORY JF values
      .PHASE_SHIFT(0),     // Amount of fixed phase shift from -255 to 255
      .STARTUP_WAIT("FALSE")   // Delay configuration DONE until DCM LOCK, TRUE/FALSE
   ) DCM_inst (
      .CLK0(CLK0),     // 0 degree DCM CLK output
     // .CLK180(CLK180), // 180 degree DCM CLK output
     // .CLK270(CLK270), // 270 degree DCM CLK output
     // .CLK2X(CLK2X),   // 2X DCM CLK output
     // .CLK2X180(CLK2X180), // 2X, 180 degree DCM CLK out
     // .CLK90(CLK90),   // 90 degree DCM CLK output
      .CLKDV(CLKDV),   // Divided DCM CLK out (CLKDV_DIVIDE)
     // .CLKFX(CLKFX),   // DCM CLK synthesis out (M/D)
     // .CLKFX180(CLKFX180), // 180 degree CLK synthesis out
     // .LOCKED(LOCKED), // DCM LOCK status output
     // .PSDONE(PSDONE), // Dynamic phase adjust done output
     // .STATUS(STATUS), // 8-bit DCM status bits output
      .CLKFB(CLK0),   // DCM clock feedback
      .CLKIN(clk),    // Clock input (from IBUFG, BUFG or DCM)
     // .PSCLK(PSCLK),   // Dynamic phase adjust clock input
     // .PSEN(PSEN),     // Dynamic phase adjust enable input
     // .PSINCDEC(PSINCDEC), // Dynamic phase adjust increment/decrement
      .RST(reset_sync)        // DCM asynchronous reset input
   );

////////////////////////////LAB 1 - Part B
//Always-on Message for the 4-dgits. The message is "FPGA"
```

```
////////////////////////////LAB 1 - Part B
//Always-on Message for the 4-dgits. The message is "FPGA"

ConstantMessage ConstantMessage_0 (reset_sync , CLKDV, an3, an2, an1, an0, charToDecode);
////////////////////////////////////////////////////////////////////////////////////////


//Instantiations of the LEDdecoder module and the FourBitCounter module.

LEDdecoder LEDdecoder_0 (charToDecode, LED);
FourBitCounter FourBitCounter_0 (CLKDV, reset_sync , counter, enable);

//Instantiate driver for signals AnX and a.b.c.d.e.f.g,dp of the 7-Segment Display.

FourDigitLEDdriver FourDigitLEDdriver_0 (reset_sync , CLKDV, LED, enable, counter, an3, an2, an1, an0, a, b, c, d, e, f, g, dp);
```

Στη παράγραφο αυτή του μέρους Β σκόπιμα δείχνεται ολόκληρη η δομή του Topmodule με σκόπο να αποδωθεί η εικόνα του συνολικού κυκλώματος. Τονίζεται πως στα παρακάτω μέρη της εργασίας (Γ, Δ) θα επιδεικνύονται μόνο οι αλλαγές στη συγκεκριμένη μονάδα καθώς η βασική της δομή θα μένει ίδια.

Αναλύοντας τη μονάδα του topmodule ταυτόχρονα αναλύεται η ροή με την οποία λειτουργεί του κύκλωμα στο σύνολο του. Η αναλύση αυτή συνοπτικά είναι η εξής:

- Με το πάτημα του reset αρχικοποιείται το κύκλωμα αφού πρώτα το αρχικό σήμα περάσει από τη μονάδα του συγχρονιστή (reset_synchronizer).
- Το ρολόι της πλακέτας (50 MHz), όπως προαναφέρθηκε, διαιρείται κατά 16 φορές μέσα από το Digital Control Manager και η περίοδος του νέου ρολογιού είναι 320 ns. Το νέο αυτό ρολόι οδηγεί τις υπόλοιπες μονάδες του συνολικού κυκλώματος.
- Η μονάδα ConstantMessage είναι αυτή που παράγει σταθέρα το μήνυμα «FPgA» και περνάει καταλλήλως τις τιμές της στον αποκωδικοποιητή LED. Η δομή της αναλύεται παρακάτω.
- Ο αποκωδικοποιητής LED αποκωδικοποιεί κάθε χαρακτήρα που δέχεται ως είσοδο όπως εξηγήθηκε παραπάνω στο μέρος Α.
- Ο 4-bit μετρητής είναι η μονάδα που συντονίζει την οδήγηση των σημάτων an3, an2, an1, an0 για τα 4 ψηφία και τα a,b,c,d,e,f,g για τα 7-τμήματα κάθε ψηφίου. Αυτό συμβαίνει διότι βάσει συγκεκριμένων κατάστασεων του μετρητή η μονάδα FourDigitLEDdriver οδηγεί κατάλληλα τα προαναφερθέντα σήματα.
- Ο FourDigitLEDdriver είναι το κύκλωμα το οποίο διαμορφώνει κατάλληλα τις εξόδους του συνολικού κυκλώματος, αφού ανάλογα τη κατάσταση που βρίσκεται ο 4-bit μετρητής, οδηγεί εναλλάξ τα σήματα an3, an2, an1, an0. Πριν από κάθε σήμα anX -και πάλι με βάση τον 4-bit Counter- έχει περάσει τα δεδομένα εξόδου του LED Decoder στα σήματα a,b,c,d,e,f,g.

reset_synchronizer Module: Η μονάδα συγχρονιστή του reset είναι απλή ως προς τη πολυπλοκότητα της όπως φαίνεται παρακάτω. Η ύπαρξη της οφείλεται στην ανάγκη συγχρονισμού του reset με το ρολόι ώστε να αποφευχθούν παραβιάσεις φαινομένου setup ή hold κατά την ενεργοποίηση του σήματος. Τέλος, η δομή της αποτελείται από 2 Flip-Flops που συνδέονται σειριακά και συμβάλουν στην αποφυγή αυτή της μεταστάθειας.

```
module reset_synchronizer (reset, clk, reset_sync);


input reset, clk;
output reg reset_sync;
reg reset_1st_flipflop;

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

always @(posedge clk)
begin
    reset_1st_flipflop = reset;
end

always @(posedge clk)
begin
    reset_sync = reset_1st_flipflop;
end

endmodule
```

ConstantMessage Module: Στο ConstantMessage Module παράγεται το σταθερό μήνυμα «FPgA», δίνοντας κάθε φορά στον LEDdecoder εναλλάξ τους χαρακτήρες F,P,g,A. Η δομή της είναι η εξής:

- Σε κάθε reset αρχικοποίηση περνά το γράμμα «F» ως το γράμμα που πρέπει να αποκωδικοποιήσει ο LED Decoder, καθώς αποτελεί το πρώτο γράμμα από αριστερά στην οθόνη της πλακέτας και συνεπώς αντιστοιχεί στη κατάσταση οδήγησης του an3 που συμβαίνει πρώτη.

- Έπειτα και κατά τη λειτουργία του κυκλώματος, κάθε φορά που ένα σήμα anX οδηγείται στο 0, η μονάδα ConstantMessage περνά στην είσοδο του LED Decoder το γράμμα εκείνο που θα πρέπει να εμφανίσει η οθόνη της πλακέτας κατά την οδήγηση του επομένου σήματος anX.

Τα παραπάνω μπορούν να προκύψουν ως συμπέρασμα, βλέποντας τη δομή του κυκλώματος της μονάδας και συγκεκριμένα του always block.

```
//////////////////////////////PART B//////////////////////////////////
// Homework part b: message to be displayed on the 7-segment display.
// The message is "FPgA".

module ConstantMessage (reset, clk, an3, an2, an1, an0, charToDecode);

input clk, reset, an3, an2, an1, an0;
output reg [5:0] charToDecode;


parameter

letter_F          =  6'b001111,
letter_P          =  6'b011001,
letter_g          =  6'b010000,
letter_A          =  6'b001010;

wire an3, an2, an1, an0;

//Every time an AnX digit is on, LEDdecoder module will decode the data for the next AnX digit.

always @(posedge clk or posedge reset) begin
    if (reset) begin
            // reset
        charToDecode = letter_F;
    end
    else if (an3 == 0 || an2 == 0 || an1 == 0 || an0 == 0)  begin
        case (charToDecode)
            letter_F: charToDecode = letter_P;
            letter_P: charToDecode = letter_g;
            letter_g: charToDecode = letter_A;
            letter_A: charToDecode = letter_F;
        endcase
    end
 end

endmodule
```

FourDigitLEDdriver Module & 4bitCounter Module: Η μονάδα του Driver, όπως προαναφέρθηκε, οδηγεί κατάλληλα τις εξόδους an3, an2, an1, an0 και a,b,c,d,e,f,g. Ο τρόπος με τον οποίον θέτει σωστά τις τιμές για τα σήματα αυτά, είναι και αυτός που προτείνεται από την εκφώνηση της 1ης εργασίας. Έτσι λοιπόν βάσει των καταστάσεων που δημιουργεί ο 4-bit μετρητής, ο driver κάνει τα εξής:

- Σε κάθε reset ενεργοποίηση τα σήματα anX και τα σήματα των 7-τμημάτων οδηγούνται στο 1, δηλαδή οι ενδείξεις μένουν κλειστές. Στη μονάδα του 4bit – Counter ο μετρητής αρχικοποιείται στη τιμή 1111.
- Στις καταστάσεις που ο 4-bit μετρητής είναι μια τιμή εκ των 1110, 1010, 0110, 0010, οδηγούνται αντίστοιχα με τις δυαδικές τιμές τα σήματα an3, an2, an1, an0 στο 0. Έτσι από τη στιγμή που ο μετρητής αυξάνεται με κάθε νέα ακμή του ρολογιού, πετυχαίνουμε να μεσολαβούν ακριβώς 3 κύκλοι ρολογιού μετά από κάθε οδήγηση ενός σήματος anX.
- Με παρόμοιο τρόπο λειτουργεί και η ανάθεση τιμών – δεδομένων στις εξόδους a,b,c,d,e,f,g. Τα σήματα αυτά παίρνουν τιμές όταν ο μετρητής βρίσκεται σε μια από τις τιμές – καταστάσεις 1100, 1000, 0100, 0000. Και εδώ η στρατηγική αυτή συμβάλει στο να μεσολαβεί ένας κύκλος ρολογιού μετά από την οδήγηση κάποιας ανόδου.
- Με τη χρήση assign οδηγεί το σήμα dp συνέχεια στη τιμή 1, δηλαδή το κρατά κλειστό κάθε όλη τη διάρκεια λειτουργίας του κυκλώματος.
- **ΠΕΡΙΠΤΩΣΗ-ΕΞΑΙΡΕΣΗ:** Μετά την ενεργοποίηση του reset ο μετρητής όπως θα δούμε παρακάτω αρχικοποιείται στο 1111. Αυτό πρακτικά σημαίνει ότι από την άνοδο an3 και την ανάθεση δεδομένων στα 7-τμήματα, δε προλαβαίνει να μεσολαβήσει 1 κύκλος, διότι η an3 οδηγείται στη κατάσταση 1110 και τα δεδομένα κατά το reset αρχικοποιούνται στη τιμή 1, δηλαδή μένουν κλειστά. Για το λόγο αυτό επιστρατεύεται ο register enable μεγέθους 2-bit, ο οποίος στο reset αρχικοποιείται στη τιμή 2 και λειτουργεί ως ακολούθως:
    - **Όταν Enable = 2:** Παγώνει τον 4bit Counter στην αρχική τιμή του 1111 και περνά τα δεδομένα των 7-τμημάτων στα a,b,c,d,e,f,g που αντιστοιχούν στην άνοδο an3. Μειώνεται κατά 1.

- **Όταν Enable = 1:** Ο 4bit Counter παραμένει ακόμα παγωμένος στο 1111 και τα δεδομένα παραμένουν μέσα στα 7-τμήματα. Πρακτικά η κατάσταση αυτή αποτελεί τον κύκλο ρολογιού που πρέπει να μεσολαβήσει ανάμεσα στη ανάθεση των δεδομένων στις ενδείξεις a,b,c,d,e,f,g και στην οδήγηση της ανόδου της an3. Μειώνεται το enable κατά 1.
- **Όταν Enable = 0:** Ξεκίνα τον μετρητή 4bit και τη κανονική λειτουργία του κυκλώματος εφεξής.

```
module FourBitCounter (clk, reset, counter, enable);
    input clk;
    input reset;
    input [1: 0] enable;
    output reg [3:0] counter;


//4-bit counter needed for the correct drive of signa
// Signal enable is needed to delay the counter for t

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            counter = 4'b1111;
        end
        else if (enable != 2'b00) begin

        end
        else if (counter == 4'b0000) begin
            counter = 4'b1111;
        end
        else begin
            counter = counter - 1;
        end
    end

endmodule
```

```
always @(posedge clk or posedge reset ) begin
    if (reset) begin
        // reset
        a = 1;
        b = 1;
        c = 1;
        d = 1;
        e = 1;
        f = 1;
        g = 1;
        enable = 2'b10;
    end
    else if (enable == 2'b10) begin
        a = LED[6];
        b = LED[5];
        c = LED[4];
        d = LED[3];
        e = LED[2];
        f = LED[1];
        g = LED[0];
        enable = enable - 1;
    end
    else if (enable == 2'b01) begin
        enable = enable - 1;
    end
    else begin
        if ( counter == 4'b1100 || counter == 4'b1000 || counter == 4'b0100 || counter == 4'b0000 ) begin
                a = LED[6];
                b = LED[5];
                c = LED[4];
                d = LED[3];
                e = LED[2];
                f = LED[1];
                g = LED[0];
        end

    end
end
```

Κατάλληλη οδήγηση των ενδείξεων a,b,c,d,e,f,g -> FourDigitLEDdriver Module

```
always @(posedge clk or posedge reset ) begin
    if (reset ) begin
        an3 = 1;
        an2 = 1;
        an1 = 1;
        an0 = 1;
    end
    else begin
        case (counter)
            4'b1110: an3 = 0;
            4'b1010: an2 = 0;
            4'b0110: an1 = 0;
            4'b0010: an0 = 0;
            default: begin
                        an3 = 1;
                        an2 = 1;
                        an1 = 1;
                        an0 = 1;
                        end
        endcase
    end
end
```

Οδήγηση των ανόδων an3, an2, an1, an0 -> FourDigitLEDdriver Module

Επαλήθευση

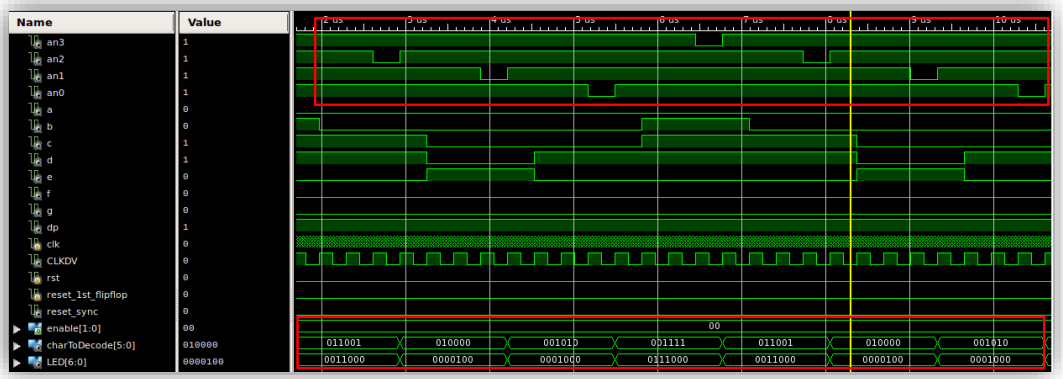
Για την επαλήθευση του κυκλώματος του μέρους Β κατασκευάστηκε ένα κύκλωμα που οδηγεί κατάλληλα το ρολόι (περίοδος 50 MHz όσο το ρολόι της πλακέτας) και το σήμα reset. Αφού τεθεί σε λειτουργία το κύκλωμα, μετά από 100663 ns καθυστέρηση, το testbench ξανά ενεργοποιεί το reset στη τιμή 1 για 259 ns. Η ενέργεια αυτή δεν είναι απαραίτητη, γίνεται περισσότερο για να ελέγξουμε τη λειτουργία του reset_synchronizer και τη συμπεριφορά του ίδιου του σήματος reset όταν το κύκλωμα έχει ήδη περάσει ένα χρονικό διάστημα λειτουργίας.

Παρακάτω δίνεται η δομή του testbench για το μέρος Β:

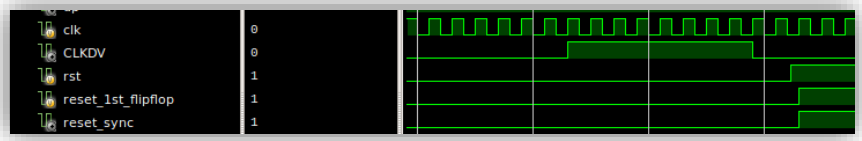
```
reg clk = 0;
reg rst = 1;


initial begin

    #60 rst=0;

    //Hit reset once again to test reset_synchronizer module

    #100663 rst = 1;
    #100922 rst = 0;
end

//Set clk period equal to spartan 3 clock frequency (50 MHZ)

always #10 clk <= ~clk;

Topmodule Topmodule_0 (rst, clk, an3, an2, an1, an0,
a, b, c, d, e, f, g, dp);
```

Οι κυματομορφές που προκύπτουν από τη προσομοίωση συμπεριφοράς του κυκλώματος εξάγουν τα ακόλουθα συμπεράσματα.

- Τα anX σήματα οδηγούνται σωστά εναλλάξ μεταξύ τους, διότι μεσολαβούν 3 κύκλοι κάθε φορά όπως αναφέρθηκε. .
- Πριν από κάθε άνοδο anX, το charToDecode (είσοδος LED Decoder) έχει τη σωστή είσοδο που πρέπει να απεικονιστεί κατά την άνοδο.
- Συγκρίνοντας τα a,b,c,d,e,f,g με την έξοδο του LED Decoder, διακρίνει κανείς ότι διοχετεύεται ορθώς η έξοδος στα σήματα και στο σωστό χρόνο.



- Τέλος επαληθεύεται και η σωστή λειτουργία του reset synchronizer καθώς το τυχαία ενεργοποιημένο σήμα rst (reset) συγχρονίζεται με το ρολόι.



Κάλυψη

Επειδή το κύκλωμα έχει μόνο 2 εισόδους, το ρολόι και το σήμα reset, καλύπτουμε πλήρως τη προσομοίωση συμπεριφοράς του. Το testbench αφού ορίσει καταλλήλως τη συχνότητα του ρολογιού και το σήμα reset, δεν έχει κάποιο άλλο διάνυσμα ελέγχου να θέσει στην είσοδο του κυκλώματος.

Πειραματικός έλεγχος και εφαρμογή στη πλακέτα

Το κύκλωμα κατά τη διάρκεια του εργαστηρίου λειτούργησε σωστά από την πρώτη δοκιμή και εμφάνιζε σταθερά και συνεχόμενα το μήνυμα “FPgA” που ήταν και το επιθυμητό.

Αυτό συνέβη, διότι η λογική του κυκλώματος με τον τρόπο που υλοποιήθηκε, κρατά τις ασφαλείς χρονικές αποστάσεις μεταξύ των ανόδων anX, αλλά και μεταξύ αυτών και των δεδομένων των ενδείξεων a,b,c,de,f,g.

Τέλος, ο reset synchronizer αποτρέπει κάθε ενδεχόμενο μεταστάθειας με την ενεργοποίηση του reset.

5. Μέρος Γ – Βηματική περιστροφή του μηνύματος με τη χρήση κουμπιού

Για το μέρος Γ προβάλουμε το μήνυμα “FPgA SPArtAn 3 ” στη πλακέτα. Το μήνυμα αυτό αποθηκεύεται σε δομή μνήμης και η διαδικασία αυτή γίνεται με τη χρήση κουμπιού που σε κάθε πάτημα του, το μήνυμα προχωρά κατά μια θέση-βήμα. Το μήνυμα περιστρέφεται κυκλικά και στο κουμπί συνδέεται κύκλωμα αποφυγής αναπηδήσεων για την ομαλή διοχέτευση του σήματος στο κύκλωμα.

Υλοποίηση

Στο μέρος Γ της εργασίας η δομή του συνολικού κυκλώματος βασίζεται στην υλοποίηση του μέρους Β, χρησιμοποιώντας ίδια λογική και υπό-μονάδες για τις βασικές λειτουργίες του. Συνεπώς:

- Οι FourDigitLEDdriver και 4bitCounter μονάδες ελέγχουν τη λειτουργία των ανόδων anX και των ενδείξεων όπως το Β.
- Ο reset_synchronizer ελέγχει και συγχρονίζει το σήμα reset όπως το Β.
- Ο LED decoder αποκωδικοποιεί όπως και πριν τις εισόδους σε ψηφία 7-τμημάτων.
- Το Topmodule παραμένει το ίδιο **με εξαίρεση την προσθήκη των instantiations των νέων μονάδων Debouncer Module για το κουμπί, clickCounter Module και RotationalMessageClick Module που εξηγούνται παρακάτω.**

clickCounter Module: Η υλοποίηση του clickCounter είναι απλή. Λειτουργία του αποτελεί να αυξάνει έναν 4-bit μετρητή κάθε φορά που το κουμπί πατιέται και να αρχικοποιείται στην αρχική του τιμή 0000 όταν ενεργοποιείται το reset. Επειδή όπως θα δούμε παρακάτω στη μονάδα RotationalMessageClick Module που παράγει το μήνυμα, ο clickCounter δείχνει στην ενεργή διεύθυνση της μνήμης που αυτό είναι αποθηκευμένο, συνεπώς όταν φτάσει στη τιμή b1110 = d14 ξανά αρχίζει από την αρχή γιατί στη θέση 14 είναι ο τελευταίος χαρακτήρας του μηνύματος.

```
module ClickCounter (reset, click, clickCounter);
    input click;
    input reset;
    output reg [3:0] clickCounter;

    always @(posedge reset or posedge click) begin
        if (reset) begin
            clickCounter = 4'b0000;
        end
        else if (clickCounter == 4'b1110) begin
            clickCounter = 4'b0000;
        end
        else begin
            clickCounter = clickCounter + 1;
        end
    end

endmodule
```

RotationalMessageClick Module: Η μονάδα που αποθηκεύει σε δομή μνήμης το μήνυμα και βάσει του clickCounter στέλνει κάθε φορά τον ανάλογα χαρακτήρα προς αποκωδικοποίηση στο LED Decoder. Συνοπτικά πραγματοποιεί τα εξής:

- Με την ενεργοποίηση του reset αποθηκεύει σε δομή μνήμης το μήνυμα.
- Σε κάθε οδήγηση anX διοχετεύει τον χαρακτήρα προς αποκωδικοποίηση στον LED Decoder που θα απεικονιστεί κατά την επόμενη ανόδο anX.
- Ανάλογα με τη διεύθυνση του μηνύματος που δείχνει ο clickCounter κάθε στιγμή, οι χαρακτήρες του μηνύματος που πρέπει να απεικονιστούν, είναι για την άνοδο:
    - An3 -> message [clickCounter] .
    - An2 -> message [clickCounter + 1]
    - An1 -> message [clickCounter + 2]
    - An0 -> message [clickCounter + 3] .
- **ΠΕΡΙΠΤΩΣΗ - ΕΞΑΙΡΕΣΗ:** Όταν ο clickCounter φτάσει να δείχνει στις τελευταίες θέσεις του μηνύματος {message[12], message[13], message[14] }, θέλουμε το μήνυμα να ξεκινά από την αρχή με τρόπο κυκλικό όπως στην εικόνα.

```
//Example 11->n _ 3 _     memory [11 12 13 14]
//         12->_ 3 _ f     memory [12 13 14  0]
//         13->3 _ f p     memory [13 14  0  1]
//         14->_ f p g     memory [14 0   1  2]
```

Για το λόγο αυτό χρησιμοποιούμε 3 νέους καταχωρητές flag2 , flag1 , flag0 σαν δείκτες που αφαιρούνται διαρκώς από τον δείκτη clickCounter και παίρνουν την τιμή 15, όταν ο clickCounter γίνει αντιστοίχως 14, 13, 12, αλλιώς παραμένουν στο 0. Έτσι πετυχαίνουμε τη κυκλική περιστροφή του μηνύματος για αυτές τις θέσεις όπως στην εικόνα-παράδειγμα παραπάνω. Η δομή της μονάδας σε Verilog παρακάτω δείχνει πρακτικά τη διαχείριση της περίπτωσης-εξαίρεσης, αλλά και τη δομή του υπό-κυκλώματος συνολικά.

```
always @(posedge reset or posedge click) begin
    if (reset) begin
        message[0]  = letter_F;
        message[1]  = letter_P;
        message[2]  = letter_G;
        message[3]  = letter_A;
        message[4]  = space;
        message[5]  = letter_S;
        message[6]  = letter_P;
        message[7]  = letter_A;
        message[8]  = letter_r;
        message[9]  = letter_T;
        message[10] = letter_A;
        message[11] = letter_n;
        message[12] = space;
        message[13] = three;
        message[14] = space;
        flag0 = 4'b0000;
        flag1 = 4'b0000;
        flag2 = 4'b0000;
    end
    else if (clickCounter == 4'b1011) begin
        flag0 = 4'b1111;
    end
    else if  (clickCounter == 4'b1100) begin
        flag1 = 4'b1111;
    end
    else if (clickCounter == 4'b1101) begin
        flag2 = 4'b1111;
    end
    else if (clickCounter == 4'b1110) begin
        flag0 = 4'b0000;
        flag1 = 4'b0000;
        flag2 = 4'b0000;
    end
end
```

```
//Every time an AnX digit is on, LEDdecoder module will decode the data for the next AnX digit.

always @(posedge clk or posedge reset ) begin
    if (reset) begin
        charToDecode = message[0];
    end
    else begin
        if (!an3) begin
            charToDecode = message [clickCounter + 1 - flag2];
        end
        else if (!an2) begin
            charToDecode = message [clickCounter + 2 - flag1];
        end
        else if (!an1) begin
            charToDecode = message [clickCounter + 3 - flag0];
        end
        else if (!an0) begin
            charToDecode = message [clickCounter];
        end
    end
end
```

Debouncer Module: Για την αποφυγή αναπηδήσεων στο πάτημα του κουμπιού στη πλακέτα, υλοποιήθηκε μια κυκλωματική μονάδα anti – bounce. Εσωτερικά της μονάδας, ένας 23-bit counter αυξάνεται όσο το σήμα του κουμπιού είναι ενεργοποιημένο στη τιμή 1. Μόνο όταν φτάσει ο μετρητής στη τιμή d6000000, δηλαδή τη διάρκεια 6,000,000 κύκλων ρολογιού, η μονάδα θεωρεί έγκυρο το πάτημα του κουμπιού. Υπολογίζεται δηλαδή ότι αν το πάτημα διαρκέσει (με βάση τη περίοδο της πλακέτας που είναι 50 MHz) 120 milliseconds και πάνω, είναι έγκυρο για να περάσει στο κύκλωμα. Το χρονικό διάστημα των 120 msec επιλέχθηκε μετά από τη πειραματική εφαρμογή της υλοποίησης πάνω στη πλακέτα, όπως θα τονιστεί και παρακάτω.

Η δομή της μονάδας του Debouncer:

```
// If button_in retain value 1 for 6000000 clock cycles, then is valid and button_out becomes 1 for 1 cycle.

    always @( posedge clk or posedge reset) begin
        if (reset) begin
            button_out = 0;
            buttonClickedCounter = 0;
        end
        else if (button_in == 1'b1 && buttonClickedCounter != 23'd6000000 ) begin
            buttonClickedCounter = buttonClickedCounter + 1;
            button_out = 0;
        end
        else if (button_in == 1'b1 && buttonClickedCounter == 23'd6000000 ) begin
            button_out = 1;
            buttonClickedCounter = 0;

        end
        else if ( button_in == 1'b0 ) begin
            button_out = 0;
            buttonClickedCounter = 0;
        end
        else if (button_out == 1'b1) begin
            button_out = 0;
        end
    end
```

Επαλήθευση

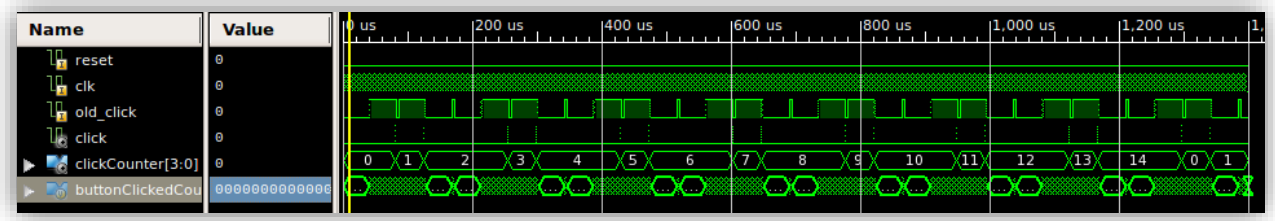
Για την επαλήθευση του κυκλώματος για το μέρος Γ της εργασίας χρειάζεται ένας testbench που εκτός από την κατάλληλη οδήγηση του ρολογιού και του reset, απαιτείται να προσομοιώνεται και η λειτουργία του κουμπιού. Το testbench που υλοποιήθηκε, επαληθεύει το κύκλωμα κάνοντας τα παραπάνω και συγκεκριμένα πέρα του ρολογιού και του reset, προσομοιώνει το κουμπί και τις αναπηδήσεις του. Η δομή του σε Verilog έχει ως ακολούθως:

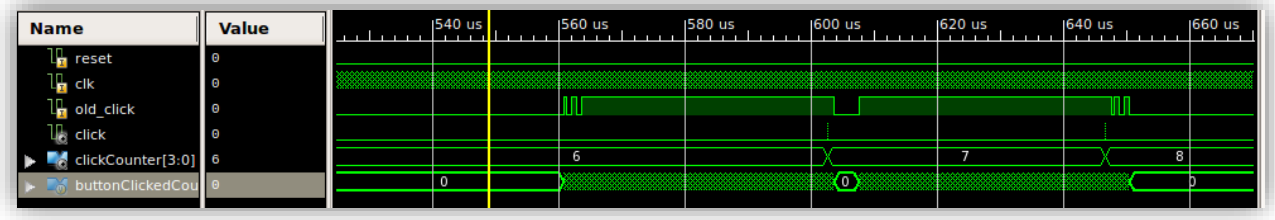
```
initial begin
    #60 rst=0;
end

always  begin
                #40000 click = 1'b1;

                #400 click = 1'b0;

                #800 click = 1'b1;

                #800 click = 1'b0;

                #800 click = 1'b1;

                #40000 click = 1'b0;

                #4000 click = 1'b1;

                #40000 click = 1'b0;

                #400 click = 1'b1;

                #800 click = 1'b0;

                #800 click = 1'b1;

                #800 click = 1'b0;

                #40000 click = 1'b1;

                #4000 click = 1'b0;
end

always #10 clk <= ~clk;

Topmodule Topmodule_0 (rst, clk,click, an3, an2, an1, an0,
a, b, c, d, e, f, g, dp);
```

Απαραίτητη διευκρίνηση!

Μόνο για την επαλήθευση του κυκλώματος σε επίπεδο συμπεριφοράς και όχι στην τελική υλοποίηση του, η μονάδα του debouncer δεν απαιτεί 6,000,000 κύκλους, αλλά μόνο 1950 για να θεωρήσει έγκυρο το κουμπί. Αυτό γίνεται γιατί ο χρόνος προσομοίωσης στη κανονική περίπτωση είναι αρκετά μεγάλος, ενώ οι 1950 κύκλοι αντιστοιχούν μόνο σε 39,000 ns.

Η λειτουργία του κουμπιού (σε full view στο Isim της Xilinx στην πρώτη εικόνα, σε zoom στη δεύτερη) φαίνεται παρακάτω. Από όλες τις αναπηδήσεις του κουμπιού, αυτές που είναι μεγαλύτερες των 1950 κύκλων τις θεωρεί έγκυρες.

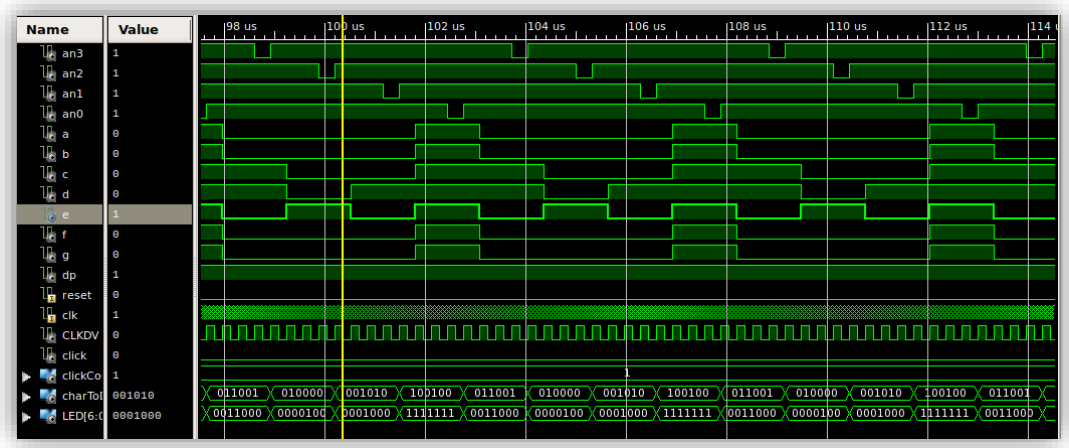


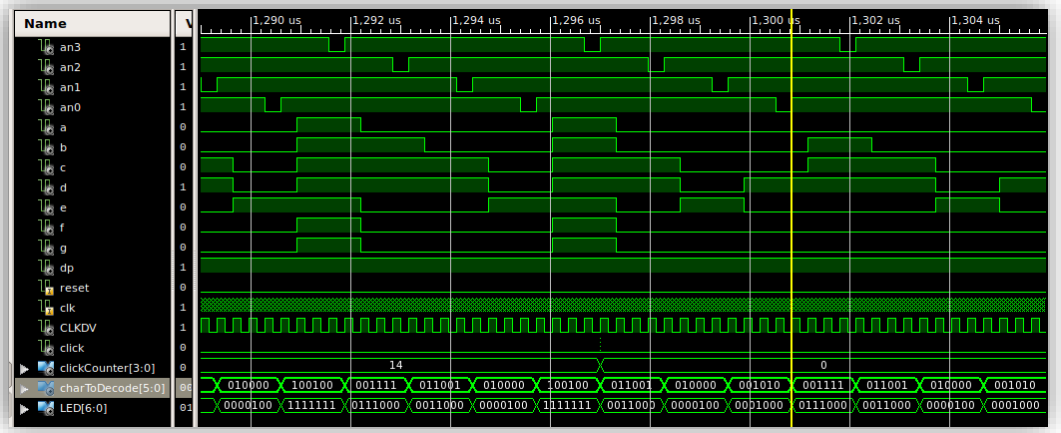


Επαληθεύεται ακόμα η ορθή λειτουργία του κυκλώματος ως προς τη περιστροφή του μηνύματος παρακολουθώντας στις παρακάτω κυματομορφές τα σήματα anX και a,c,d,e,f,g.

1η εικόνα: Το διάστημα που ο counter βρίσκεται στη τιμή d1 και οι ενδείξεις πρέπει να δείχνουν το μήνυμα «PgA_».

2η εικόνα: Το διάστημα που ο counter βρίσκεται στις d14 και στη συνέχεια d0 και οι ενδείξεις πρέπει να δείχνουν το μήνυμα «_FPg» και «FPgA» αντιστοίχως.





Κάλυψη

Στο μέρος Γ, στα διανύσματα ελέγχου του testbench προστίθεται η είσοδος του κουμπιού. Η λειτουργία του κουμπιού σε επίπεδο συμπεριφοράς και κυματομορφών δεν μπορεί να ταυτιστεί επακριβώς με την ενέργεια πίεσης του στη πραγματικότητα πάνω στη πλακέτα, ούτε και με τις αναπηδήσεις που θα κάνει το σήμα του. Έτσι το ποσοστό κάλυψης ίσως δεν θεωρείται ξεκάθαρο εδώ, παρόλα αυτά με το testbench να προσομοιώνει σε κάποιο βαθμό τις αναπηδήσεις που θα μπορούσε να κάνει το κουμπί στη πραγματικότητα, το ποσοστό κάλυψης πλησιάζει και εδώ το μέγιστο.

Πειραματικός έλεγχος και εφαρμογή στη πλακέτα

- *Πόσες φορές απαιτήθηκε:*

Η πρακτική δοκιμή του κυκλώματος πάνω στη πλακέτα έδειξε ότι η λογική του κυκλώματος γύρω από τη βηματική περιστροφή του μηνύματος ήταν σωστή. Παρόλα αυτά χρειάστηκε να επαναληφθεί ο πειραματικός έλεγχος, εωσότου πετύχει η σωστή λειτουργία του κουμπιού.

- Αλλαγές στον κώδικα:

Οι διαφοροποιήσεις του κώδικα που πραγματοποιήθηκαν, αφορούσαν μόνο τη μονάδα του Debouncer. Η αρχική υλοποίηση με έναν 22-bit μετρητή που αντιστοιχούσε σε διάρκεια περίπου 83 millisec (4194304 κύκλοι) για να θεωρηθεί έγκυρο το σήμα του κουμπιού, πάνω στη πλακέτα έδειξαν πως ήταν μικρό χρονικό διάστημα. Το συμπέρασμα αυτό εξάγεται από το γεγονός ότι η χρήση του κουμπιού από τον βοηθό του εργαστηρίου προχωρούσε το μήνυμα άλλες φορές 1 βήμα που είναι το ορθό, άλλες φορές 2 απανωτά βήματα.

Η αλλαγή του μετρητή σε 23-bit και της διάρκειας που θεωρείται έγκυρο το σήμα σε περίπου 160 millisec () έκανε το χρονικό διάστημα να είναι λίγο μεγαλύτερο από το επιθυμητό, με συνέπεια το κουμπί να πρέπει να κρατηθεί περισσότερο πατημένο από το κανονικό για να προχωρήσει ένα βήμα το μήνυμα.

Η λύση βρέθηκε κάπου στη μέση και τελικά τα 120 milisec που προκύπτουν από την τελική υλοποίηση αρκούν για τη σωστή λειτουργία του κουμπιού.

6. Μέρος Δ – Βηματική περιστροφή του μηνύματος με σταθερή καθυστέρηση

Για τέταρτο και τελευταίο μέρος της εργασίας, ζητήθηκε το κύκλωμα να λειτουργεί ομοίως με το μέρος Γ, με τη διαφορά να υφίσταται στον τρόπο περιστροφής, καθώς τώρα η χρήση κουμπιού αντικαθίσταται με την αυτόματη βηματική περιστροφή του μηνύματος μετά από σταθερή καθυστέρηση. Η καθυστέρηση που μεσολαβεί πριν από κάθε περιστροφή αντιστοιχεί σε 1,3421 δευτερόλεπτα μετά και την τελική υλοποίηση του κυκλώματος.

Υλοποίηση

Η δομή του κυκλώματος αλλάζει ελάχιστα σε σχέση με το μέρος Γ. Οι νέες μονάδες ConstantDelayCounter και RotationalMessageAutomated αντικαθιστούν τον clickCounter και το RotationalMessageClick module του προηγούμενου μέρους αντιστοίχως. Μικρές δομικές αλλαγές γίνονται και στο topmodule του κυκλώματος, στο εσωτερικό του οποίου αφαιρείται η είσοδος του σήματος του κουμπιού και το instantiation του Debouncer Module για το κουμπί.

ConstantDelayCounter Module: Όπως και στους προηγούμενους μετρητές που υλοποιήθηκαν, η δομή του είναι παρόμοια και απλή. Υπάρχουν 2 μετρητές στη συγκεκριμένη μονάδα, ένας 22-bit που αφορά τη σταθερή καθυστέρηση που απαιτείται να μεσολαβεί πριν από κάθε περιστροφή και ένας μετρητής-δείκτης στη διεύθυνση του μηνύματος. Όταν ο 22-bit μετρητής φτάσει στη τελευταία του κατάσταση, γεγονός που σημαίνει πως έχουν μεσολαβήσει 4194304 κύκλοι, (1,3421 δευτερόλεπτα βάσει του διαιρεμένου ρολογιού του κυκλώματος με περίοδο 320 ns), ο δείκτης-μετρητής αυξάνεται κατά 1.

Παρακάτω στην εικόνα παρατίθεται η δομή του υπό-κυκλώματος:

```
//When counterEnableClick reaches final state which is equal to 4194304 clock cycles, increment clickCounter by 1.

    always @(posedge reset or posedge clk) begin
        if (reset) begin
            clickCounter = 4'b0000;
            counterEnableClick = 0;
        end
        else if (counterEnableClick == 22'b1111111111111111111111 && clickCounter == 4'b1110) begin
            clickCounter = 4'b0000;
            counterEnableClick = 0;
        end
        else if (counterEnableClick == 22'b1111111111111111111111) begin
            clickCounter = clickCounter + 1;
            counterEnableClick = 0;
        end
        else begin
            counterEnableClick = counterEnableClick + 1;
        end
    end
```

RotationalMessageAutomated Module: Η μονάδα που αποθηκεύει σε δομή μνήμης το μήνυμα και καταχωρεί κατάλληλα τα δεδομένα προς αποκωδικοποίηση, ώστε να εμφανιστούν σωστά κατά την άνοδο anX που αντιστοιχούν. Η υλοποίηση είναι ίδια με αυτή του μέρους Γ με τη διαφορά να βρίσκεται στο τρόπο που υλοποιείται η περίπτωση-εξαίρεση.

Συγκεκριμένα, επειδή το always block προηγουμένως λειτουργούσε έτσι ώστε το σήμα του κουμπιού να βρίσκεται στη sensitive list και να ενεργοποιεί την ανάθεση των flags δεικτών, στο μέρος Δ η ίδια διαδικασία γίνεται στις ακμές του ρολογιού. Όσο ο counter που ελέγχει την ενεργή διεύθυνση του μηνύματος παραμένει στις τιμές που ενεργοποιούν τα flags, το always block θα ανέθετε ξανά και ξανά τις ίδιες τιμές στους registers. Για το λόγο αυτό χρησιμοποιούνται στο μέρος Δ άλλοι 3 καταχωρητές ack2,ack1,ack0 αντίστοιχοι των flags, οι οποίοι επιτρέπουν την ενεργοποίηση των flags μόνο σε μια ακμή κάθε φορά και όχι συνεχώς.

Η δομή του RotationalMessageAutomated Module:

```
always @(posedge reset or posedge clk) begin
    if (reset) begin
        message[0]  = letter_F;
        message[1]  = letter_P;
        message[2]  = letter_g;
        message[3]  = letter_A;
        message[4]  = space;
        message[5]  = letter_S;
        message[6]  = letter_P;
        message[7]  = letter_A;
        message[8]  = letter_r;
        message[9]  = letter_T;
        message[10] = letter_A;
        message[11] = letter_n;
        message[12] = space;
        message[13] = three;
        message[14] = space;
        flag0 = 4'b0000;
        flag1 = 4'b0000;
        flag2 = 4'b0000;
        ack0 = 1'b0;
        ack1 = 1'b0;
        ack2 = 1'b0;
    end
    else if (clickCounter == 4'b1100 && ack0 == 1'b0) begin
        flag0 = 4'b1111;
        ack0 = 1'b1;
    end
    else if  (clickCounter == 4'b1101 && ack1 == 1'b0) begin
        flag1 = 4'b1111;
        ack1 = 1'b1;
    end
    else if (clickCounter == 4'b1110 && ack2 == 1'b0) begin
        flag2 = 4'b1111;
        ack2 = 1'b1;
    end
    else if (clickCounter == 4'b0000) begin
        flag0 = 4'b0000;
        flag1 = 4'b0000;
        flag2 = 4'b0000;
        ack0 = 1'b0;
        ack1 = 1'b0;
        ack2 = 1'b0;

    end
end
```

```
//Every time an AnX digit is on, LEDdecoder module will decode the data for the next AnX digit.

always @(posedge clk or posedge reset ) begin
    if (reset) begin
        charToDecode = message[0];
    end
    else begin
        if (!an3) begin
            charToDecode = message [clickCounter + 1 - flag2];
        end
        else if (!an2) begin
            charToDecode = message [clickCounter + 2 - flag1];
        end
        else if (!an1) begin
            charToDecode = message [clickCounter + 3 - flag0];
        end
        else if (!an0) begin
            charToDecode = message [clickCounter];
        end
    end
end
```

Επαλήθευση

Το τελευταίο μέρος της εργασίας απαιτεί την αυτόματη βηματική περιστροφή του μηνύματος. Συνεπώς το testbench που υλοποιήθηκε, χρειάζεται μονάχα να οδηγεί κατάλληλα το ρολόι και το reset, χωρίς τη χρήση άλλης εισόδου όπως του κουμπιού προηγουμένως. Η δομή του testbench είναι απλή και οδηγεί το ρολόι και πάλι στα 50 MHz.

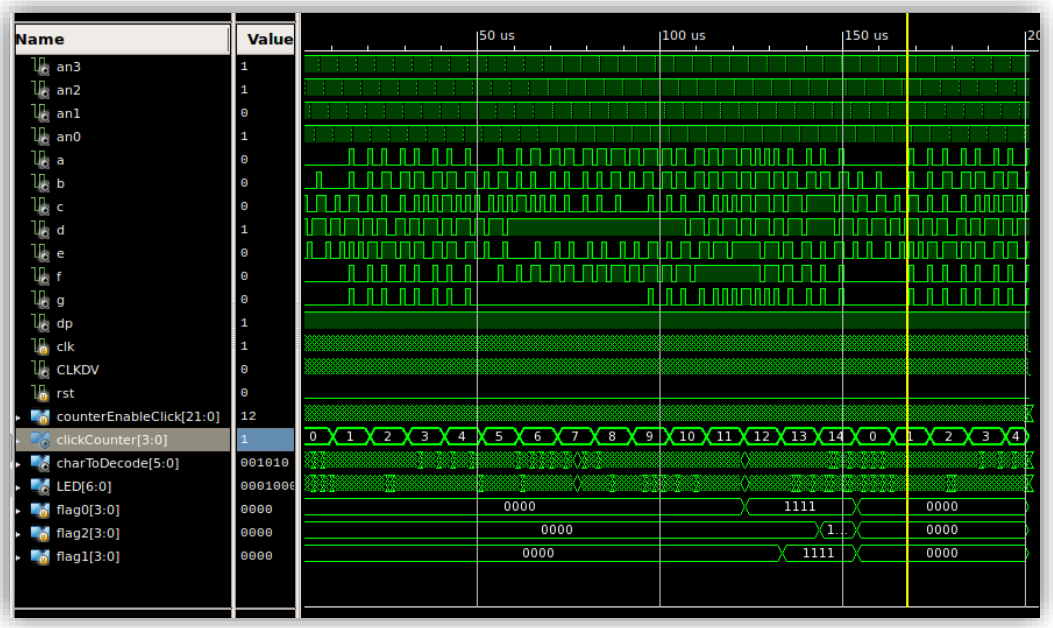
```
reg clk = 0;
reg rst = 1;


initial begin
    #60 rst=0;
end

always #10 clk <= ~clk;

Topmodule Topmodule_0 (rst, clk, an3, an2, an1, an0,
a, b, c, d, e, f, g, dp);
```

Απαραίτητη Σημείωση!

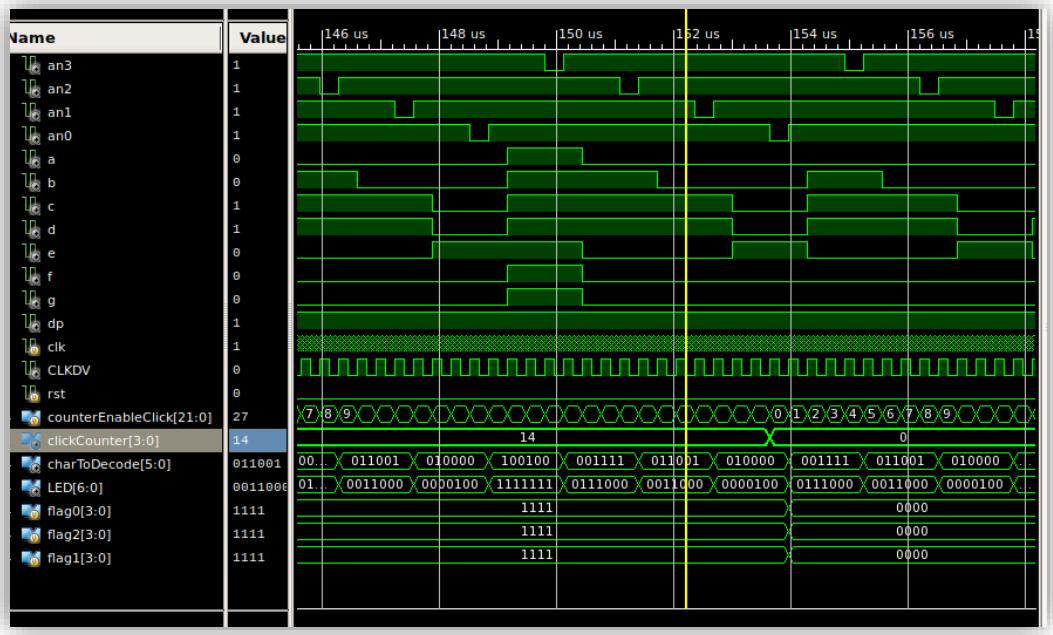
Ο μετρητής των 22-bit αντιστοιχεί, όπως επισημάνθηκε παραπάνω, σε χρόνο 1,3421 δευτερόλεπτα ανά περιστροφή. Ο χρόνος αυτός αποτελεί μεγάλο χρονικό διάστημα προσομοίωσης, οπότε για τις ανάγκες της επαλήθευσης σε επίπεδο συμπεριφοράς και κυματομορφών, ο μετρητής έγινε 5-bit. Αυτή η αλλαγή δεν αφορά τη τελική υλοποίηση.

Η ορθή λειτουργία του μέρους Δ διακρίνεται στις παρακάτω κυματομορφές:

Σε full view:



Στις καταστάσεις που η διεύθυνση μηνύματος βρίσκεται στο 14 και μετέπειτα στο 0:



Κάλυψη

Τα διανύσματα ελέγχου του κυκλώματος είναι όπως και στο μέρος Β το σήμα reset και το ρολόι. Τα 2 διανύσματα τίθενται και τα 2 κατάλληλα και η κάλυψη του κώδικα είναι πλήρης αφού δεν υπάρχει άλλη παράμετρος να οριστεί.

Πειραματικός έλεγχος και εφαρμογή στη πλακέτα

Για την επαλήθευση του κυκλώματος πάνω στη πλακέτα εφαρμόστηκε η δομή της τελικής υλοποίησης με τον 22-bit μετρητή αντί του 5-bit.

Το πείραμα έδειξε πως το κύκλωμα λειτουργεί σωστά, καθώς μεσολαβεί ένα χρονικό διάστημα περίπου ενός δευτερολέπτου μετά από κάθε περιστροφή.

7. Συμπέρασμα

Με το πέρας της προθεσμίας και την ολοκλήρωση των τεσσάρων μερών της εργασίας στη διάρκεια του εργαστηρίου, η παρούσα τεχνική αναφορά συλλέγει σε μορφή κειμένου όλες τις διαδικασίες που ακολουθήθηκαν για να ολοκληρωθεί η 1η Εργαστηριακή άσκηση του μαθήματος.

Συνοψίζοντας, η πορεία της 1ης εργασίας ολοκληρώθηκε επιτυχώς και τα μέρη Α,Β,Γ,Δ βρίσκονται στη τελική υλοποίηση τους.

Η σχεδίαση των κυκλωμάτων σε επίπεδο Verilog δε παρουσίασε απροσπέραστα προβλήματα και τα περισσότερα λύθηκαν κατευθείαν μετά την επαλήθευση σε επίπεδο συμπεριφοράς και την μελέτη των αντίστοιχων κυματομορφών.

Στο κομμάτι του εργαστηρίου οι βοηθοί συνέβαλλαν στη κατανόηση κυρίως των τομέων εκείνων που αφορούσαν την πλακέτα, λόγου χάρη του φαινομένου των αναπηδήσεων στο κουμπί και των μέτρων που θα έπρεπε να παρθούν για να αντιμετωπιστεί κατάλληλα.

Ο πειραματικός έλεγχος και η πρακτική εφαρμογή του κυκλώματος πάνω στη πλακέτα ήταν οι ενέργειες που έφεραν την εργασία και τα τμήματα της στη τελική της μορφή, καθώς είτε επιβεβαίωναν την ορθή λειτουργία τους όπως στο μέρος Α,Β,Δ, είτε παρουσίαζαν τις κινήσεις που έπρεπε να γίνουν για να φτάσουν στην ομαλή λειτουργικότητα (βλέπε φαινόμενο debounce κουμπιού - Μέρος Γ).

Τέλος