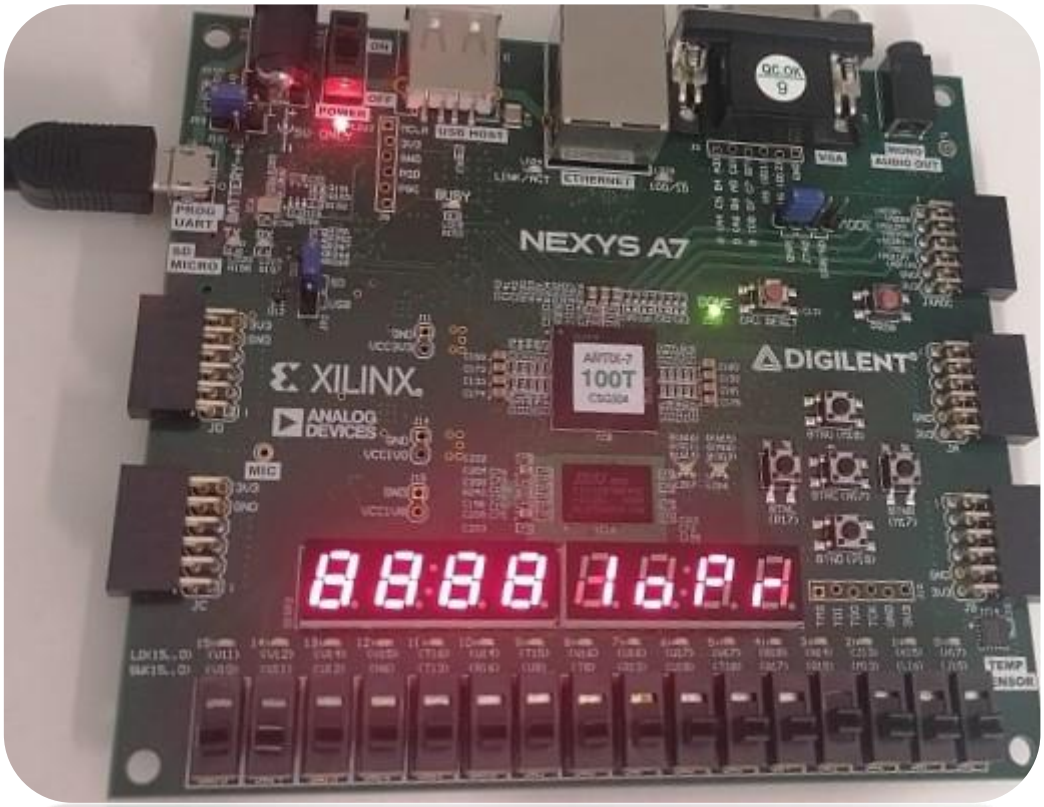
1η Εργασία Εξαμήνου – Οδηγός Ένδειξης 7 Τμημάτων



Γαρυφαλιά Γεωργιτζίκη
ΑΕΜ: 03218
Ημ/νία : 04/11/2022

ΠΕΡΙΛΗΨΗ

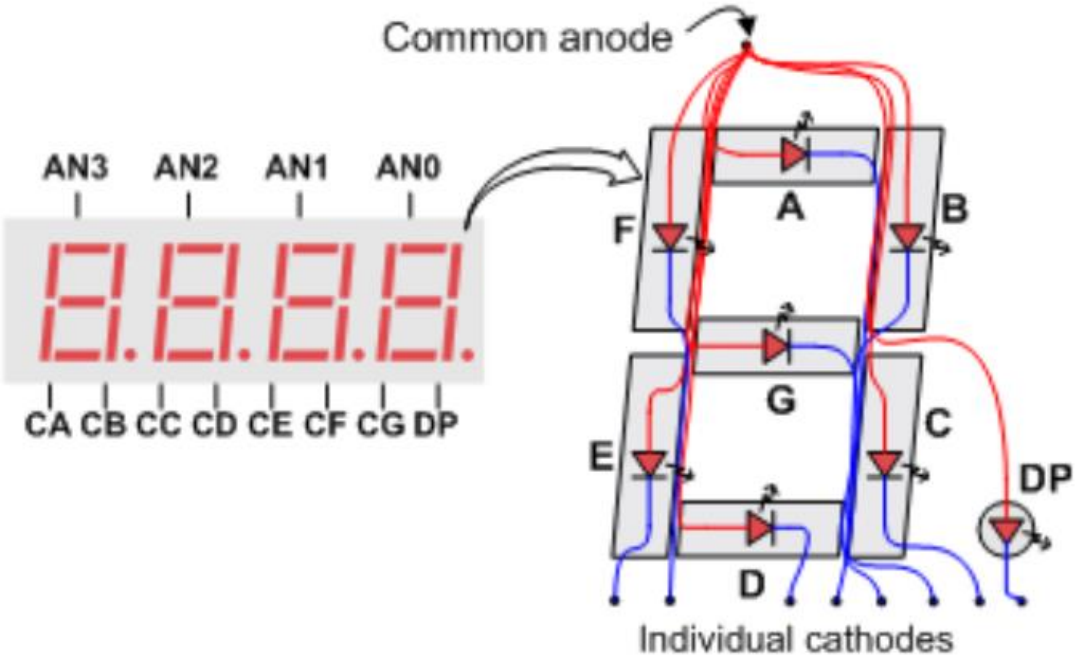
Στην εργαστηριακή εργασία «Οδηγός Ένδειξης 7 Τμημάτων» θα δούμε την διαδικασία υλοποίησης του μοντέλου μας, μέσα από τα Μέρη Α, Β, Γ και Δ που περιέχει η εργασία.

ΕΙΣΑΓΩΓΗ

Στόχος της εργασίας είναι η υλοποίηση ενός οδηγού ένδειξης 7 τμημάτων LED της πλακέτας Nexys A7-100T, που θα επιτυγχάνει την περιστροφική παρουσίαση του μηνύματος «1oProject2022-23». Το μήνυμα θα παρουσιάζεται μετακινόντας τους χαρακτήρες προς τα αριστερά είτε με το πάτημα ενός κουμπιού, είτε μετά από περίπου 1.6777214 δευτερόλεπτα. Όλοι οι στόχοι επιτεύχθηκαν επιτυχώς, ύστερα βέβαια από αρκετές αποτυχίες και λάθη που παρατηρήθηκαν κατά τη ν διάρκεια της υλοποίησης.

ΜΕΡΟΣ Α – "ΥΛΟΠΟΙΗΣΗ ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΗ 7–ΤΜΗΜΑΤΩΝ"

Σε αυτό το μέρος της εργασίας υλοποιήθηκε ένας αποκωδικοποιητής 7 τμημάτων, με οδήγηση 4 ψηφίων LED. Τα 4 LED οδηγούνται από τις ανόδους AN3, AN2, AN1 και AN0 ενώ τα 7 τμήματα του κάθε LED είναι τα CA, CB, CC, CD, CE, CF, CG και DP. Τα ψηφία ανάβουν όταν η άνοδος (ANx) που οδηγεί το συγκεκριμένο LED οδηγείται στο ένα (1) και τα αντίστοιχα τμήματά του στο μηδέν (0).



Καθώς το ρολόι της FPGA είναι 100MHz, οι άνοδοι εναλλάσσονται κάθε 0.2 μs, χρόνος που δεν επιτρέπει στο ανθρώπινο μάτι να αντιληφθεί το «αναβόσβημα» των ψηφίων.

Η υλοποίηση του κυκλώματος αυτού, δημιουργήσαμε έναν συνδυαστικό αποκωδικοποιητή, με είσοδο "char", δηλαδή τον χαρακτήρα που θέλουμε να αναπαραστήσουμε και έξοδο "LED", που οδηγεί σε 1 (για να σβήσει) και 0 (για να ανάψει) τα τμήματα του ψηφίου που αντιπροσωπεύουν τον χαρακτήρα μας. Το τμήμα DP των LED, το κρατάμε σταθερά ένα (σβηστό) καθώς δεν τον χρησιμοποιήσαμε καθόλου.

```
module LEDdecoder(char, LED);
.  .  .  .  .  .  .  .  .  .
endmodule
```

Οι χαρακτήρες του μηνύματος «1oProject2022-23» έχουν παραμετροποιηθεί όπως φαίνεται ενδεικτικά παρακάτω:

```
parameter one = 4'b0000;
parameter letter_o = 4'b0001;
parameter letter_P = 4'b0010;
parameter letter_r = 4'b0011;
.  .  .  .  .  .  .  .  .  .  .
```

Στην συνέχεια με την χρήση του always block, σε κάθε αλλαγή του char, αναπαραστήσαμε τον κάθε χαρακτήρα, οδηγώντας σε 0 ή 1 τα αντίστοιχα τμήματα του LED.

```
always@(char)
begin
    case(char)//display of the characters "1oProject2022-23"
    //the series of the parts of its LED is ca,cb,cc,cd,ce,cf,cg
    zero: LED = 7'b0000001;//0
    one: LED = 7'b1001111;//1
    .  .  .  .  .  .  .  .  .  .  .  .
    default: LED = 7'b1111111;//all the parts of the LED are off
    endcase
end
```

Για την επαλήθευση της υλοποίησής μας, χρησιμοποιήσαμε ένα testbench, το οποίο με καθυστέρηση μιας μονάδας (#1), εμφάνιζε όλους του χαρακτήρες του μηνύματος με την κωδικοποιημένη binary μορφή τους “char” και την αντίστοιχη αναπαράσταση με “LED”.

```
initial
begin
 $monitor( "CHAR=%b, LED=%b ", char, LED);
  #1 char = one;
  #1 char = o;
  #1 char = P;
  #1 char = r;
  .  .  .  .  .  .  .  .
  #1 char = three;
end
```

ΜΕΡΟΣ Β – «ΟΔΗΓΗΣΗ ΤΕΣΣΑΡΩΝ ΨΗΦΙΩΝ»

Η οδήγηση των τεσσάρων ψηφίων της FPGA υλοποιείται στο Μέρος Β της εργασίας. Πιο συγκεκριμένα για να πετύχουμε την εναλλαγή των ψηφίων με την προτεινόμενη ελάχιστη καθυστέρηση των 0.2 μs, χρησιμοποιήσαμε, όπως μας υποδείχθηκε, την δομική μονάδα MMCM (Mixed Mode Clock Manager). Η υλοποίηση της πραγματοποιήθηκε στο «module FourDigitLEDdriver_partB» που είναι το top module μας μαζί με τα instantiation των υπολοίπων modules που θα δούμε παρακάτω. Από το πρότυπο της "MMCM2_BASE" κρατήσαμε μόνο τις παραμέτρους που επηρέαζαν το κύκλωμα μας, δηλαδή τις:

```
.CLKFBOUT_MULT_F(6.0)
.CLKIN1_PERIOD(10.0)
.CLKOUT1_DIVIDE(120)
.DIVCLK_DIVIDE(1)
```

Οι τιμές που δώσαμε στις παραμέτρους αυτές προέκυψαν από τον τύπο

$$F_{OUT} = F_{CLKIN} \times \frac{M}{D \times O}$$ όπου

- **F_{CLKIN}** = 100 MHz , συχνότητα ρολογιού της FPGA
- M = 6 (2 < M < 64) , η ελάχιστη αποδεκτή τιμή για να μην επηρεάζεται η τιμή της **Fvco** με τύπο $$F_{VCO} = F_{CLKIN} \times \frac{M}{D}$$
- D = 1 (όπως μας υποδείχθηκε στο εργαστήριο) για να μην επηρεάζεται η τιμή της **Fvco**
- και O = 120 (1 < O < 128)

ώστε να έχουμε **F_{OUT}** = 1/**T_{OUT}** = 1/0.2μs = 5MHz , η οποία είναι η επιθυμητή συχνότητα .. για να ..

Παράλληλα, στο πρότυπο "MMCM2_BASE_inst" διατηρήσαμε τις εξόδους

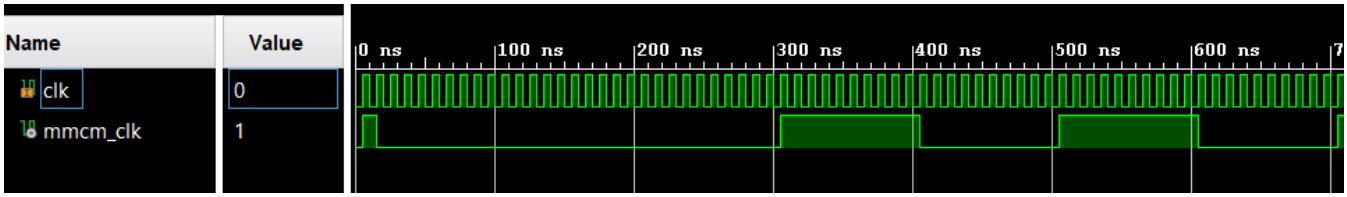
```
.CLKOUT1(mmcm_clk)
.CLKFBOUT(CLKFBOUT)
```

καθώς χρησιμοποιήσαμε μόνο την έξοδο «CLKOUT1» και ορίσαμε ως έξοδο της MMCM στο κύκλωμά μας την «mmcm_clk», και τις εισόδους

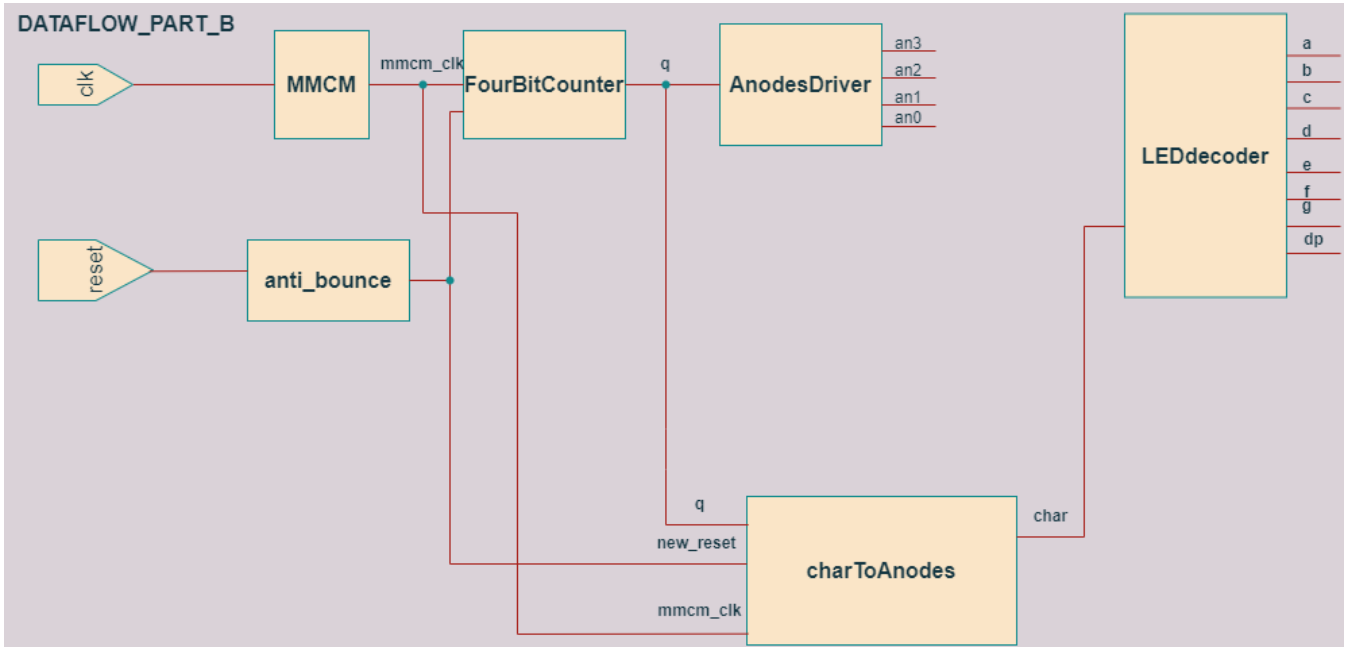
```
.CLKIN1(clk)
.CLKFBIN(CLKFBOUT)
```

όπου χρησιμοποιήσαμε ως είσοδο της MMCM την «CLKIN1» που στο κύκλωμα μας ήταν «clk», ενώ ταυτόχρονα ανατροφοδοτούμε την έξοδο

«CLKFBOUT» στην είσοδο «CLKFBIN». Αποτέλεσμα όλων αυτών είναι οι περίοδος του ρολογιού να γίνεται 20 φορές μεγαλύτερη, δηλαδή από clk = 10ns περίοδο, να έχουμε mmcm_clk = 200ns = 0.2μs περίοδο.



Το διάγραμμα DATAFLOW του μέρους B της εργασίας φαίνεται παρακάτω:



Στη συνέχεια, χρησιμοποιήθηκε μία μονάδα «anti_bounce», σκοπός της οποίας είναι να αποφευχθούν οι απρόβλεπτες αναπηδήσεις στο σήμα κατά το πάτημα του κουμπιού BTNC (reset στην ακίδα N17 της FPGA), που επαναφέρει το κύκλωμα την αρχική του κατάσταση. Η γενική ιδέα για το anti_bounce, ήταν το σήμα του κουμπιού ή του reset να φιλτράρεται, διερχόμενο μέσα από δύο Flip Flops (οδηγούμενα από ένα αργό ρολόι) σε σειρά, και μία πύλη XOR, η οποία θα δεχόταν ως εισόδους , τις εξόδους των δύο FFs, και όσο αυτές είχαν διαφορετικές τιμές, ο counter, ο οποίος θα ελεγχόταν από την πύλη XOR, θα έκανε reset.

Η αρχική κατάσταση που αναφέρθηκε είναι η αναπαράσταση των τεσσάρων πρώτων χαρακτήρων του μηνύματος «1oPr»:

Για να επιτευχθεί η εν λόγω αναπαράσταση υλοποιήθηκε ένας μετρητής 4-bit, ο «FourBitCounter», ο οποίος μετράει αντίστροφα από το 15 στο 0, ενεργοποιώντας τις ανόδους εναλλάξ όπως φαίνεται στον πίνακα:

Τιμή μετρητή	AN3	AN2	AN1	AN0
1111	1	1	1	1
1110	**0**	1	1	1
1101	1	1	1	1
1100	1	1	1	1
1011	1	1	1	1
1010	1	**0**	1	1
1001	1	1	1	1
1000	1	1	1	1
0111	1	1	1	1
0110	1	1	**0**	1
0101	1	1	1	1
0100	1	1	1	1
0011	1	1	1	1
0010	1	1	1	**0**
0001	1	1	1	1
0000	1	1	1	1

Η καθυστέρηση 0.2μs που αναφέραμε προηγουμένως, είναι πρακτικά ο χρόνος μετάβασης από την μια κατάσταση του μετρητή στην άλλη όπως δείχνει το κόκκινο βέλος και συμβάλλει στην αποφυγή σύγχυσης μεταξύ των ενδείξεων των ανόδων.

Η “always” του module “FourBitCounter”, είτε επαναφέρει το q (μετρητής) στην αρχική του κατάσταση (αριθμός 15) σε κάθε θετική ακμή του reset (κάθε φορά που πατάμε το κουμπί), είτε αφαιρεί 1 από το q σε κάθε θετική ακμή του ρολογιού μέχρι να φτάσει στο 0.

```
always@(posedge clk or posedge reset)
begin
    if(reset)
        begin
        q <= 4'b1111;
        end
    else
        q <= (q == 0) ? (4'b1111):(q - 4'b0001);
end
```

Στη συνέχεια, το q γίνεται είσοδος στο module “AnodesDriver”, όπου οδηγεί τις ανόδους AN3, AN2, AN1 και AN0, όπως φαίνεται στον παραπάνω πίνακα κάθε φορά που αλλάζει η τιμή του q.

```
always@(q)
begin
    case(q)
        //seira anodwn an3, an2, an1, an0
        4'b1110: anodes = 4'b0111;
        4'b1010: anodes = 4'b1011;
        4'b0110: anodes = 4'b1101;
        4'b0010: anodes = 4'b1110;
        default: anodes = 4'b1111;
    endcase
end
```

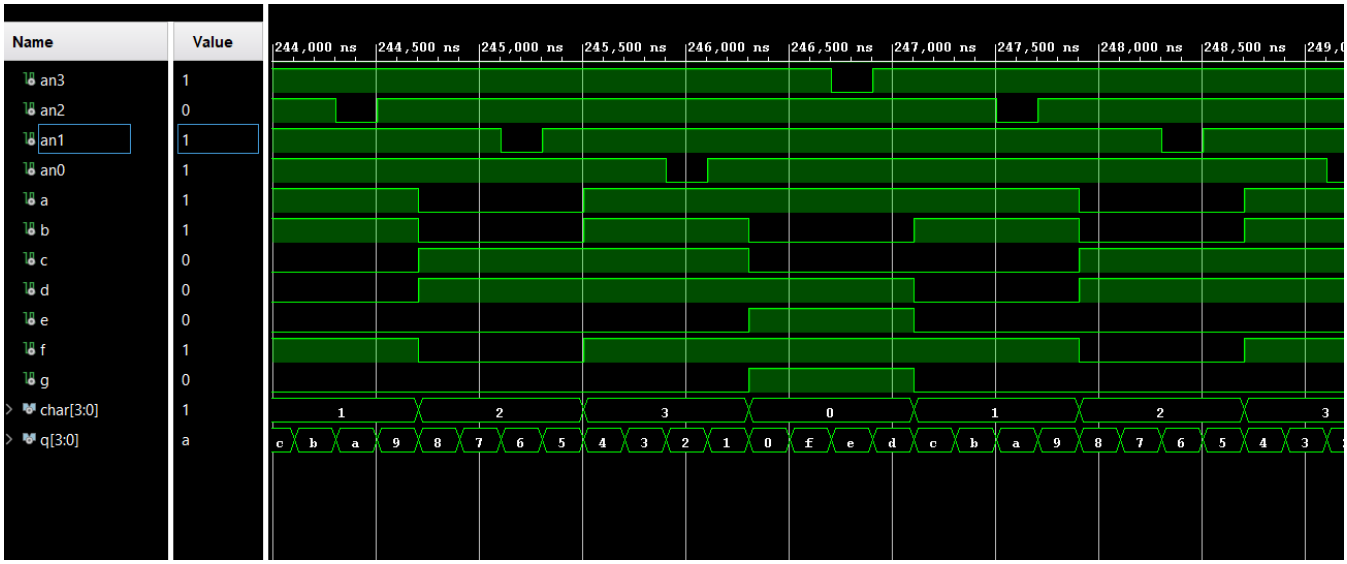
Ύστερα, η έξοδος q του μετρητή, γίνεται είσοδος στο module “charToAnodes” το οποίο πλέον μέσω της παρακάτω “always” ελέγχει ποιος χαρακτήρας θα βγει στην έξοδο, ενώ φαίνεται και ο χρόνος προετοιμασίας μεταξύ των ανόδων.

```
always@(q)
begin
    case(q)
        4'b1111:char = one;
        4'b1110:char = one;
        4'b1101:char = one;
        4'b1100:char = letter_o;
       .  .  .  .  .  .  .  .  .
       4'b0000:char = one;
    endcase
end
```

Σχετικά με την επαλήθευση του κυκλώματος, μέσα από το testbench που δημιουργήσαμε για το μέρος Β και την παρακάτω “always”, ορίζουμε το ρολόι ώστε στο μισό της περιόδου να αντιστρέφεται (από 0->1 και 1->0). Έτσι παρατηρήσαμε ότι όντως η περίοδος γίνεται επί 20.

```
always
begin
   #(`period/2)clk = ~clk;
end
```

Όσον αφορά την εμφάνιση του μηνύματος «1oPr», στις παρακάτω κυματομορφές φαίνεται πως κατά την εναλλαγή του q, αλλάζουν οι άνοδοι και τελικά αναπαρίσταται το μήνυμα. Όταν το “char” παίρνει την τιμή 0, στην οθόνη LED βλέπουμε το «1», για “char” = 1, βλέπουμε «o», για “char” = 2, βλέπουμε «P» και για “char” = 3, βλέπουμε «r» («1oPr»).

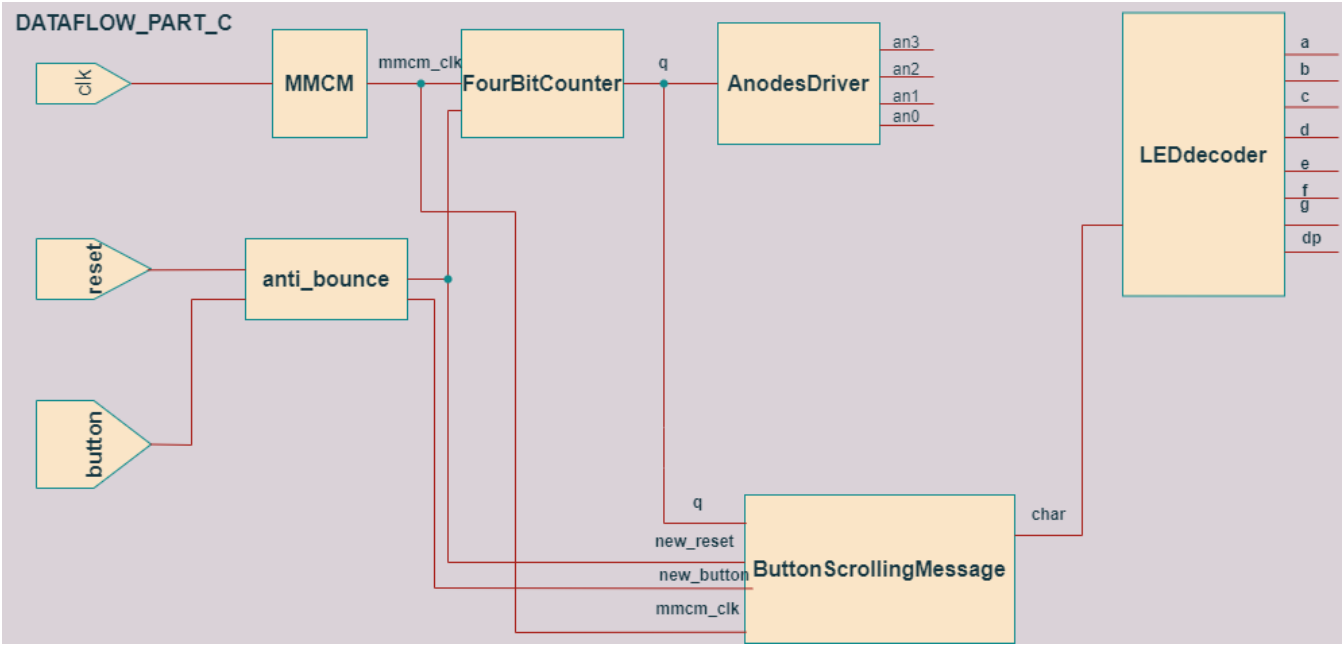


Δυσκολία αντιμετώπισα στην υλοποίηση του anti_bounce module, και ίσως στην κατανόηση σχετικά με το πώς δουλεύει, ωστόσο ύστερα από αρκετές δοκιμές θεωρώ πως κατανοήθηκε ο ρόλος του στο κύκλωμα. Δυστυχώς όπως μας υποδείχθηκε στο εργαστήριο, παρόλο που έτρεχε κανονικά το κύκλωμα στην πλακέτα, η υλοποίησή του anti_bounce ήταν λάθος και κατά την προσπάθεια αλλαγής του κώδικα για το anti_bounce, προέκυψε κάποιο σφάλμα και όχι μόνο δεν υλοποιήθηκε σωστά το νέο anti_bounce, αλλά δεν δούλευε ούτε το παλιό module, που φαινομενικά πριν λειτουργούσε γι αυτό και δεν υπήρξε αρκετό υλικό στην αναφορά σχετικά με αυτό (εξήγηση always, κυματομορφές κλπ).

ΜΕΡΟΣ Γ – «ΒΗΜΑΤΙΚΗ ΠΕΡΙΣΤΡΟΦΗ ΜΗΝΥΜΑΤΟΣ ΜΕ ΧΡΗΣΗ ΚΟΥΜΠΙΟΥ»

Στόχος του Μέρους Γ της εργασίας είναι η βηματική περιστροφή του μηνύματος μας με την χρήση του κουμπιού BTNR που αντιστοιχεί στην ακίδα M17 της πλακέτας μας.

Το διάγραμμα DATAFLOW του μέρους Γ είναι:



Η λειτουργία της MMCM είναι ίδια με προηγουμένως, όπως και του module “anti_bounce” μόνο που τώρα φιλτράρεται και το σήμα του κουμπιού BTNR για την αποφυγή αναπηδήσεων στο σήμα και στο module “ButtonScrollingMessage” εισέρχεται το “new_button”.

Αποθηκεύσαμε το μήνυμά μας σε μία μνήμη 16-bit “**reg [3:0] message [0:15];**” στο module “ButtonScrollingMessage” όπως φαίνεται παρακάτω.

```
always@(posedge clk)
begin
    message[0] <= one;
    message[1] <= letter_o;
    message[2] <= letter_P;
    message[3] <= letter_r;
    .  .  .  .  .  .  .  .  .
    message[15] <= three;
end
```

Στην συνέχεια ακολουθούν τρία always blocks:

1. Εδώ η always δημιουργεί το σήμα “button_raise”, το οποίο οδηγείται από το σήμα του κουμπιού της πλακέτας σε κάθε θετική ακμή ρολογιού

```
always@(posedge clk)
begin
    if (button_old != button && button == 1'b1)
    begin
        button_raise <= 1'b1;
    end
    else
    begin
         button_raise <= 1'b0;
    end
    button_old <= button;
end
```

2.Στην δεύτερη always, σε κάθε θετική ακμή ρολογιού ή του reset, εάν reset == 1 τότε αρχικοποιείται ο “counter_button” σε 0 αλλιώς εάν `button_raise == 1,` ο “counter_button” αρχικοποιείται σε 0 αν είναι ίσος με 15 αλλιώς αυξάνεται η τιμή του κατά ένα.

```
always@(posedge clk or posedge reset)
begin
    if (reset)
        begin
            counter_button <= 4'b0000;
        end
        else if(button_raise == 1'b1)
        begin
            if(counter_button == 4'b1111)
            begin
                counter_button <= 4'b0000;
            end
            else
            begin
                counter_button <= counter_button + 1;
            end
        end
end
```

3.Στην τελευταία always σε κάθε αλλαγή της τιμής του q (μετρητή), ο counter_button που υπολογίστηκε πριν, δηλώνει τον χαρακτήρα του μηνύματος που θα αναπαρασταθεί από την μνήμη και όπως αναφέρθηκε προηγουμένως πρέπει να έχει ένα περιθώριο αλλαγής ενός βήματος.

```
always@(q)
begin
    case(q)
        4'b1111:char_button <= message[counter_button];
        4'b1110:char_button <= message[counter_button];
        4'b1101:char_button <= message[counter_button];
        4'b1100:char_button <= message[counter_button + 1];
        .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
        4'b0000:char_button <= message[counter_button];
    endcase
end
```

Τέλος, στο αρχείο για τα constraints .xdc, προσθέσαμε την γραμμή **"set_property -dict { PACKAGE_PIN M17 IOSTANDARD LVCMOS33 } [get_ports { button }];"** η οποία αντιστοιχίζει το σήμα "button" με την ακίδα M17 της πλακέτας μας, που είναι το κουμπί για την περιστροφική κίνηση του μηνύματος που χρησιμοποιήσαμε παραπάνω.

Σχετικά με το testbench, για τα μέρη Γ και Δ, χρησιμοποιήθηκε το ίδιο testbench και για να τεστάρουμε το κουμπί απλά αλλάζαμε το σήμα του κουμπιού ή του reset με κάποια καθυστέρηση ώστε να ελέγξουμε τις κυματομορφές.

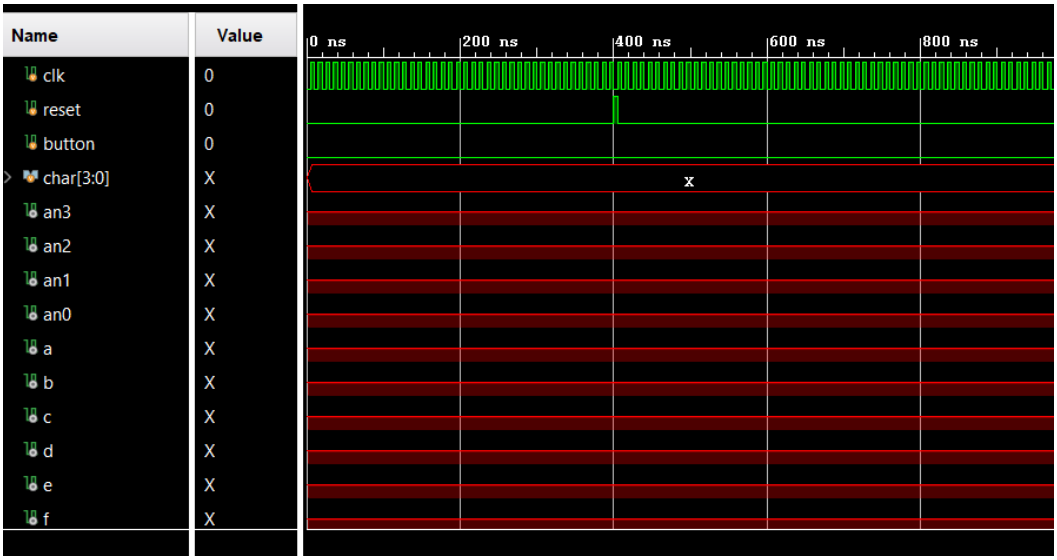
```
#400 reset = 1'b1;
#5 reset = 1'b0;

#2000 button = 1'b1;
#10000 button = 1'b0;
#2000 button = 1'b1;

#(500*`period)reset = 1'b1;
#(5*`period)reset = 1'b0;
```

Όσον αφορά δυσκολίες που προέκυψαν, έκανα μία αλλαγή που έγινε σχετικά με το top module, καθώς είχα υλοποιήσει ένα για όλα τα μέρη της εργασίας και όπως μου υποδείχθηκε κατά την εξέταση του εργαστηρίου, και δημιούργησα από ένα top module για κάθε μέρος Β, Γ και Δ της εργασίας. Υποθέτω ότι είτε έγινε κάποιο λάθος σε μία από τις δύο διορθώσεις του κώδικα (είτε εκεί , είτε στο anti_bounce διότι και εκεί έγιναν αλλαγές όπως προανέφερα) και μετά από πολύωρη προσπάθεια να διορθωθεί ο κώδικας και να δουλεύει σωστά ξανά, δεν υπήρξε κάποιο ουσιώδες αποτέλεσμα, γι'

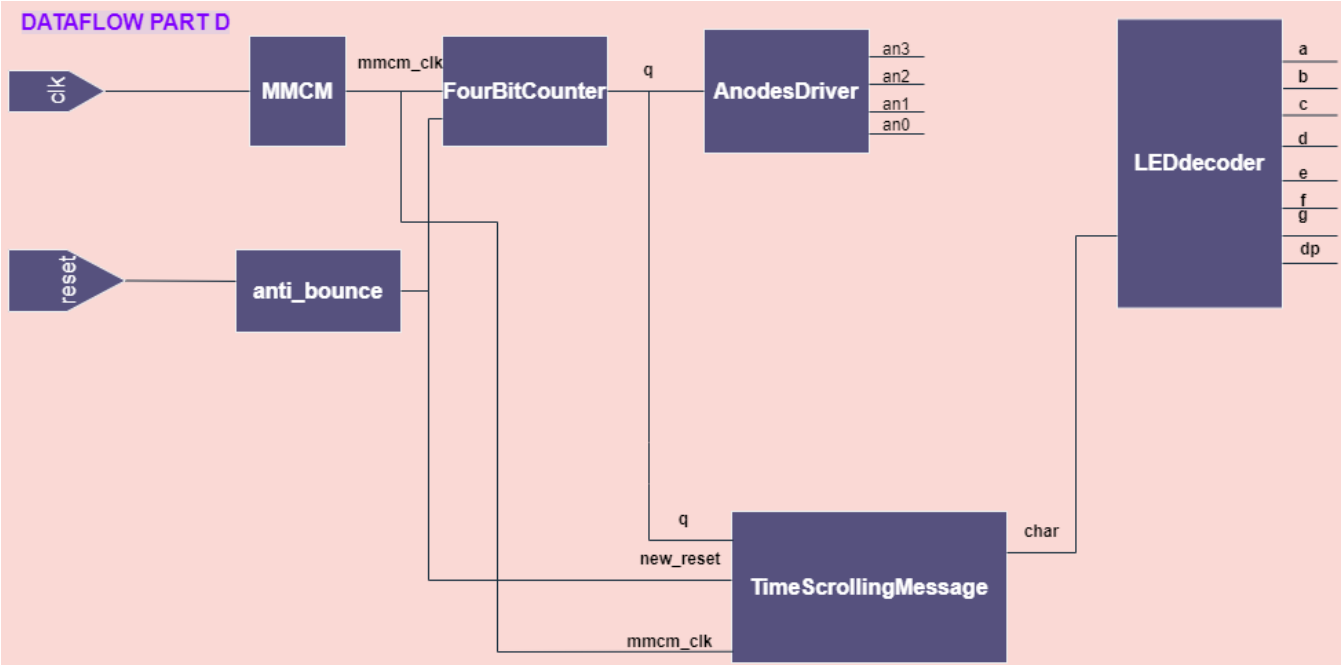
αυτό και στο μέρος Γ οι κυματομορφές δείχνουν Χ στις περισσότερες εξόδους.



ΜΕΡΟΣ Δ – «ΒΗΜΑΤΙΚΗ ΠΕΡΙΣΤΡΟΦΗ ΜΗΝΥΜΑΤΟΣ ΜΕ ΣΤΑΘΕΡΗ ΚΑΘΥΣΤΕΡΗΣΗ»

Στο Μέρος Δ της εργασίας, το ζητούμενο ήταν η υλοποίηση της βηματικής περιστροφής του μηνύματός μας με χρονική καθυστέρηση, ώστε το μήνυμα να κινείται προς τα αριστερά κάθε 1,6777214 δευτερόλεπτα. Για την επίτευξη αυτού, χρησιμοποιήθηκε ένας 23-bit μετρητής για να μετράει αυτό τον χρόνο.

Το DATAFLOW διάγραμμα φαίνεται παρακάτω:



Το module που αλλάζει σε αυτή την υλοποίηση είναι το “TimeScrollingMessage”. Συγκεκριμένα, και σε αυτό το module ξεκινάμε με την αρχικοποίηση της μνήμης με το μήνυμά μας.

```
always@(posedge clk)
begin
    message[0] <= one;
    message[1] <= letter_o;
    message[2] <= letter_P;
    message[3] <= letter_r;
    .  .  .  .  .  .  .  .  .
    message[15] <= three;
end
```

Στην συνέχεια, έχουμε τον 23-bit μετρητή, υλοποιημένο μέσα σε μία always που τρέχει σε κάθε θετική ακμή του ρολογιού ή του reset.

```
//23 bit counter for counting 1,6777214 seconds
always@(posedge clk or posedge reset)
begin
    if(reset == 1 || counter == 23'b000_00000_00000_00000_00000)
    begin
        counter <= 23'b111_11111_11111_11111_11111;
    end
    else if(counter != 23'b000_00000_00000_00000_00000)
    begin
        counter <= counter - 23'b000_00000_00000_00001;
    end
end
```

Ακολουθεί always για τον έλεγχο του σήματος “counter_raise”, το οποίο γίνεται 1 κάθε φορά που ο μετρητής γίνεται 0, δηλαδή έχει μετρήσει 1,6777214 δευτερόλεπτα, αλλιώς παραμένει στο 0.

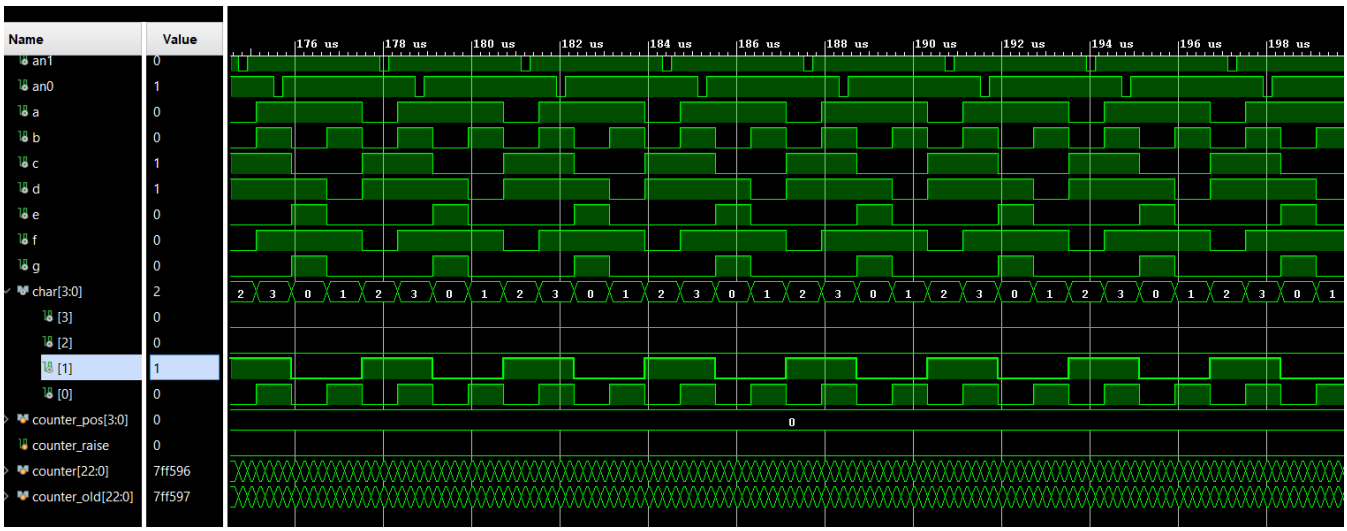
```
always@(posedge clk)
begin
    // detect rising edge
    if (counter_old != counter && counter == 23'b000_00000_00000_00000_00000)
    begin
        counter_raise <= 1'b1;
    end
    else
    begin
         counter_raise <= 1'b0;
    end
    counter_old <= counter;
end
```

Τέλος, παρακάτω βλέπουμε τον τρόπο με τον οποίο το σήμα “counter_raise” ελέγχει τον μετρητή “counter_pos”, δηλαδή δίνει το σήμα ώστε το μήνυμα να κινηθεί κατά ένα χαρακτήρα, και αυξάνει τον “counter_pos ” για να πάει στην κατάλληλη θέση στην μνήμη.

```
//counter_pos for counting the position of its display character in the message
always@(posedge clk or posedge reset)
begin
    if (reset)
    begin
        counter_pos <= 4'b0000;
    end
    else if(counter_raise == 1'b1)
```

```
    begin
        if(counter_pos == 4'b1111)
        begin
            counter_pos <= 4'b0000;
        end
        else
        begin
            counter_pos <= counter_pos + 1;
        end
    end
end
always@(q)
begin
    case(q)
        4'b1111:char_button <= message[counter_pos];
        4'b1110:char_button <= message[counter_pos];
        .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
        4'b0000:char_button <= message[counter_pos];
    endcase
end
```

Στην παρακάτω εικόνα φαίνονται οι κυματομορφές από την υλοποίηση του μέρους Δ. Παρατηρούμε ότι στο σήμα “char” η ακολουθία είναι 0123, και δεν αλλάζει σε 1234 -> 2345 -> 3456 κλπ. Αυτό συμβαίνει διότι ο χρόνος αλλαγής χαρακτήρα οδηγείται από έναν 23-bit μετρητής, με αποτέλεσμα να είναι τεράστιος για τα δεδομένα της προσομοίωσης και πρέπει να αφήσουμε την προσομοίωση να τρέξει για αρκετό χρόνο για να δούμε την εναλλαγή των χαρακτήρων στις κυματομορφές.



Λόγω λάθους τρόπου «κωδικοποίησης» των χαρακτήρων (παραμέτρων) αντιμετώπισα δυσκολία στον έλεγχο των κυματομορφών, καθώς έπρεπε κάθε φορά να ελέγχω ποιος χαρακτήρας αντιστοιχεί στο

ανάλογο 4-bit «κωδικό». Έτσι άλλαξα ξανά τις παραμέτρους ώστε το ίδιο μήνυμα (“1oProject2022-23”) να βγαίνει κωδικοποιημένο σε μορφή 0,1,2,3,4,5,….15.

Συμπερασματικά, η υλοποίηση της 1ης εργασίας, ολοκληρώθηκε σε αρκετά καλό βαθμό, παρά τις δυσκολίες που προέκυψαν. Η σχεδίαση και η επαλήθευση του οδηγού ένδειξης 7 τμημάτων LED απαιτούσε καλή κατανόηση της πλακέτας Nexys A7-100T. Όσον αφορά την δοκιμή στην πλακέτα, υπήρξε αρκετό άγχος σχετικά με το εάν όλο αυτό που υλοποιήσαμε στο Vivado, θα αναπαρασταθεί σωστά στην πλακέτα μας ή όχι. Πάντως δεν παύει να είναι μία εξαιρετικά ενδιαφέρουσα εργασία.