Οδηγός Ένδειξης 7-τμημάτων

Εργαστήριο Ψηφιακών Συστημάτων (2023-24)

Ιωάννης Αθανασιάδης 03491

15/11/2023

Περίληψη

Αναφορά για την εργασία του μαθήματος του Εργαστηρίου Ψηφιακών Κυκλωμάτων (ECE333), μέσω της οποία γίνεται ανάλυση των μεθόδων *ανάπτυξης* και *debugging* ενός *RTL design* στα πλαίσια του προγράμματος ***Xilinx Vivado*** και της ***Digilent Nexys A7-100T*** *FPGA*. Για να το κάνουμε αυτό αναλύουμε με διάφορους μεθόδους (όπως σχήματα ροής δεδομένων) τις κυκλωματική υλοποίηση της ***Verilog*** που αποτελούνε την εργασίας.

Εισαγωγή

Ο στόχος της εργασίας ήταν η οδήγηση μιας τετραψήφιας οθόνης 7-τμημάτων που είναι ενσωματωμένη στην *Nexys A7-100T*. Πιο αναλυτικά η περιστροφική παρουσίαση ενός μηνύματος ακριβώς 16 χαρακτήρων. Η περιστροφή θα λειτουργεί είτε με το πάτημα ενός κουμπιού είτε μετά από ένα χρονικό διάστημα, κάνοντας ολίσθηση προς τα δεξιά σε κάθε περίπτωση. Η εργασία θεωρήθηκε επιτυχημένη αφού όλα τα μέρη που την αποτελούν όπως και οι στόχοι που αναφέραμε παραπάνω ολοκληρώθηκαν με επιτυχία.

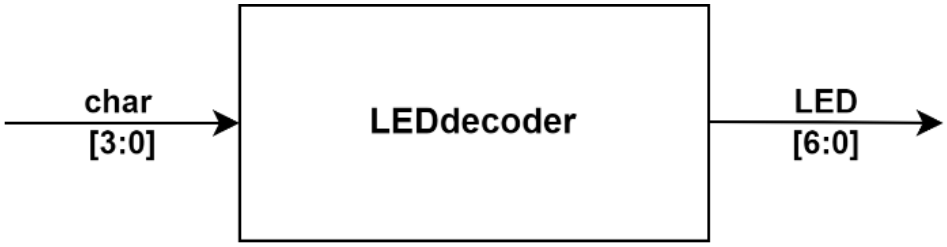
Μέρος Α – Υλοποίηση Αποκωδικοποιητή 7-τμημάτων

Το πρώτο μέρος της εργασίας είναι αρκετά απλό μιας και είναι η υλοποίηση ενός απλού αποκωδικοποιητή μέσα στο *module* ***LEDdecoder***.

Για την αντιστοιχία των τιμών εισόδων/εξόδων χρησιμοποιήθηκε ο παρακάτω πίνακας στον οποίο αντιστοιχίζεται κάθε τιμή ενός μονοψήφιου δεκαεξαδικού αριθμού με την αντίστοιχη εμφάνιση του στην οθόνη, για παράδειγμα όταν:

$$char = 0x9 \ \rightarrow \textit{το εν λόγω ψηφίο της οθόνης δείχνει } 9$$

char	LED
0x0	0000001
0x1	1001111
0x2	0010010
0x3	0000110
0x4	1001100
0x5	0100100
0x6	0100000
0x7	0001111
0x8	0000000
0x9	0000100
0xA	0001000
0xB	1100000
0xC	0110001
0xD	1000010
0xE	0110000
0xF	0111000



Σημείωση: παρόλο που η εκφώνηση μιλάει για οδήγηση του κάθε τμήματος της οθόνης στο 1 ισχύει το αντίθετο, έτσι χρειάστηκε να βρω τους αντίστροφους του τιμών του LED σε κάθε περίπτωση.

Επαλήθευση

Το πλαίσιο δοκιμών του πρώτου μέρους είναι αρκετά απλό γιατί το κύκλωμα μας είναι συνδυαστικό οπότε δεν χρειάζονται σήματα όπως *ρολόι* και *reset* .

```
initial begin
        char = 4'h0;
    #10 char = 4'h1;
    #10 char = 4'h2;
    #10 char = 4'h3;
    #10 char = 4'h4;
    #10 char = 4'h5;
    #10 char = 4'h6;
    #10 char = 4'h7;
    #10 char = 4'h8;
    #10 char = 4'h9;
    #10 char = 4'ha;
    #10 char = 4'hb;
    #10 char = 4'hc;
    #10 char = 4'hd;
    #10 char = 4'he;
    #10 char = 4'hf;
    #10 $finish;
end
```

Τα διανύσματα που χρησιμοποιήθηκαν είναι πρακτικά όλοι οι συνδυασμοί ενός *4-bit* αριθμού, δηλαδή *από 0 έως 16 (δεκαδικό).* Το *testbench* θα μπορούσαμε να χρησιμοποιήσουμε μία ***for-loop*** αλλά τα διανύσματα προς δοκιμή θεωρήθηκαν λίγα ώστε να είναι εύκολη η γραφή με τον τρόπο που φαίνεται στο πλαίσιο στα δεξιά (*χειροκίνητα*).

Μέρος Β – Οδήγηση Τεσσάρων Ψηφίων

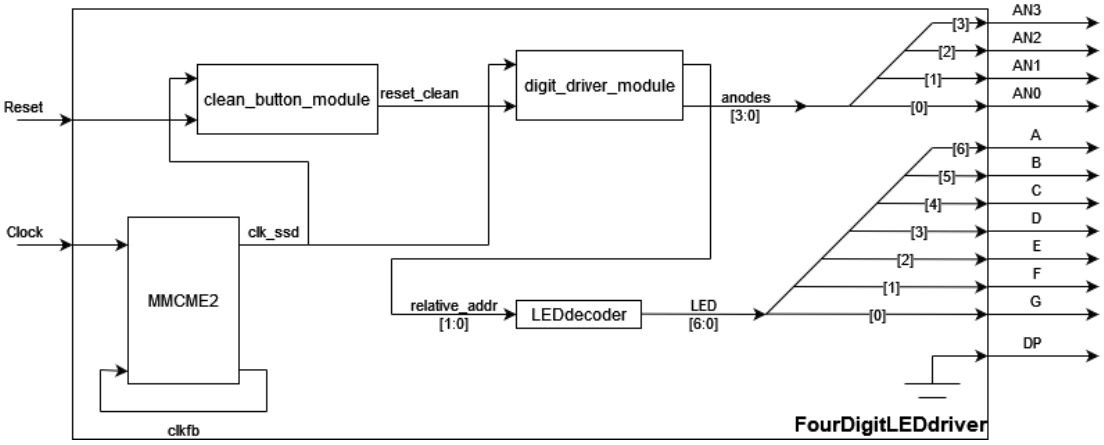
Το ζητούμενο στο δεύτερο μέρος της εργασίας είναι η οδήγηση και των τεσσάρων ψηφίων που αποτελούν την οθόνη μας.

Για την οδήγηση του κάθε ψηφίου χρησιμοποιούνται 4 είσοδοι (***AN3, AN2, AN1, AN0***) μέσω των οποίων επιλέγεται ένα ψηφίο την φορά (*που οδηγείται στο **0***), *σαν κωδικοποίηση one-hot*. Για παράδειγμα όταν:

$$\{AN3, AN2, AN1, AN0\} = 4'b10111$$
$$\rightarrow \textit{οδηγείται } 2^{o} \textit{ πιο σημαντικό ψηφίο}$$

Μονάδα *FourDigitLEDdriver*

Για *top-level* μονάδα έχουμε το ***FourDigitLEDdriver***:



Σημειώσεις:

- Το ***DP*** είναι μονίμως ***pull-down*** γιατί θέλουμε η υποδιαστολή της οθόνης να είναι πάντα *σβηστή*.
- Οι δίαυλοι ***anodes*** και ***LED*** σπάνε σε *μονάδες* για να οδηγήσουν τις εξόδους.
- Η έξοδος ***relative_addr*** του *digit_driver_module* συνδέεται απευθείας στην είσοδο ***char*** του *LEDdecoder*, για περισσότερες πληροφορίες πάτε στην παράγραφο “ Μονάδα digit_driver_module”.

Μονάδα *MMCME2*

Για να πετύχουμε ένα πιο αργό clock όπως ζητάει η εκφώνηση με την χρήση του ***MMCME2*** *block* χρειάζεται να επιλέξουμε κάποιες παραμέτρους που θα μας μετασχηματίζουν το ρολόι της *FPGA (@100MHz)* σε ένα άλλο με περίοδο .20 μs *(ή **5MHz**).*

Σαν πρώτο βήμα παίρνουμε το έτοιμο **template** του *MMCME2_base* που μας παρέχει το *Vivado* και κάνουμε της παρακάτω αλλαγές:

- Περνάμε στην παράμετρο *CLKIN1_PERIOD* την περίοδο του ρολογιού εισόδου *(δηλαδή των 100MHz)* που είναι στα 10ns, δηλαδή τιμή ***10.0*** .
- Περνάμε στην παράμετρο *CLKFBOUT_MULT_F* την τιμή **6.0** καθώς το VCO πρέπει να έχει εύρος τιμών *600MHz~1.600MHz.*
- Βάζουμε ως διαιρετή *CLKOUT1_DIVIDE* την τιμή **120**

Έτσι στην πράξη:

$$fclkout = \frac{fclkin * 50}{120} = 5\ MHz, \text{όπου } fvco = fclkin * 50$$

<u>Σημείωση:</u> για την σωστή λειτουργία του MMCME2 χρειάζεται ένα wire να που κάνει feedback την μονάδα διαχείρισης του ρολογιού, για χάρη του οποίου ορίστηκε το clkfb.

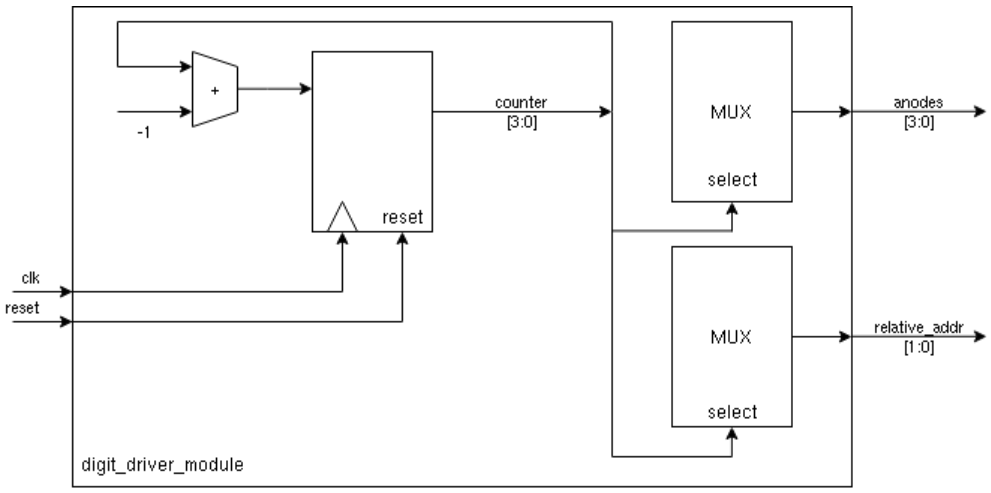
Μονάδα *digit_driver_module*

Αυτή είναι η μονάδα που οδηγεί τα 4 διαφορετικά ψηφία της οθόνης της FPGA. Έχει εισόδους το ρολόι clk και το ασύγχρονο reset, ενώ για εξόδους έχει διαύλους το anodes, για να επιλέγουμε το ψηφίο που θα γράψουμε, και το relative_addr το οποίο επιλέγει τα δεδομένα που θα γραφτούν σε κάθε ψηφίο.

Ο μηχανισμός στον οποίον βασίζεται το relative_addr είναι σχετικός με τα επόμενα μέρη της εργασίας όπου έχουμε το μήνυμα αποθηκευμένο σε ένα κομμάτι μνήμης. Παρόλα αυτά για να διατηρήσουμε ένα modularity χρησιμοποιούμε το ίδιο module και εδώ. Πρακτικά αυτή η έξοδος είναι ένας μετρητής που για το <u>πρώτο ψηφίο</u> δείχνει στην *<u>θέση 0</u>* της μνήμης, για το <u>δεύτερο</u> στην *<u>θέση 1</u>* της μνήμης και ου το καθεξής. Όμως για το *μέρος Β* θα πάρει το relative_add θα πάρει την θέση του *char*.

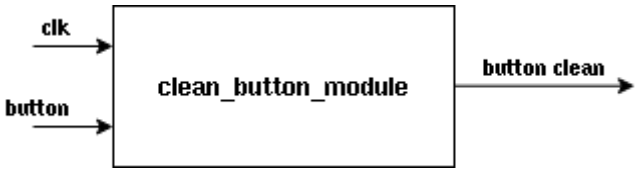
Πιο αναλυτικά:

- Μια ***always*** ακολουθιακής λογικής με *λίστα ευαισθησίας* στην *θετική ακμή του clk και reset* για τον μετρητή *counter*
- Μια ***always*** συνδυαστικής λογικής με *λίστα ευαισθησίας* τον *counter* για να μεταβάλλονται οι τιμές των εξόδων.



Μονάδα *clean_button_module*

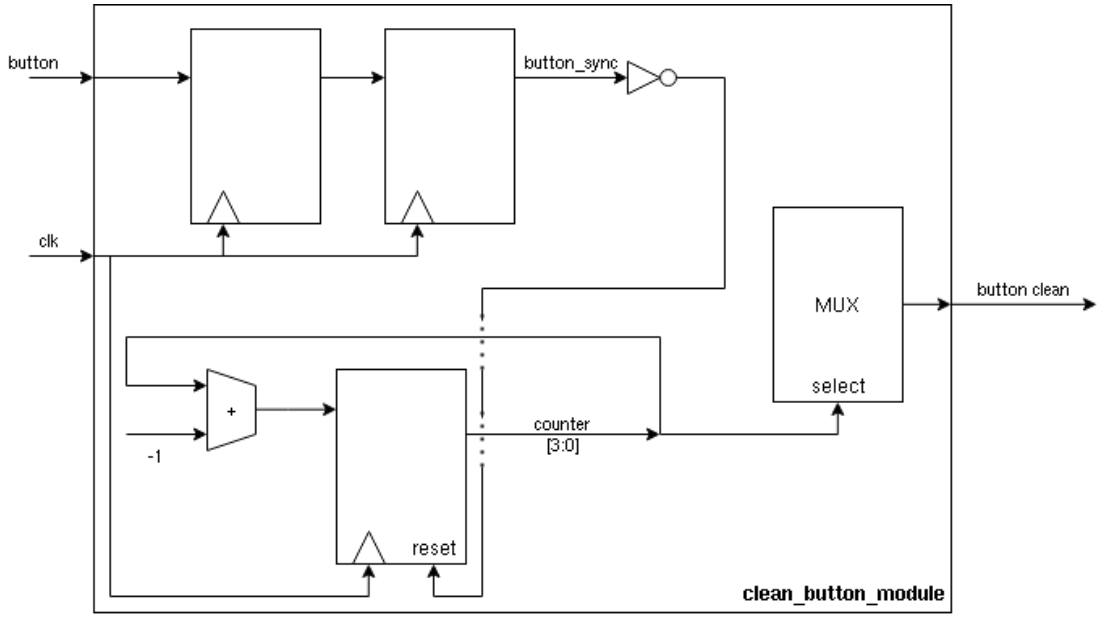
Η μονάδα αυτή είναι για την αποφυγή της εισόδου εξωτερικού σήματος κοντά σε κρίσιμο χρόνο (*όπως στο* <u>set-up time</u>) και για εφαρμογή συστήματος ***anti-bounce***, όπως το *reset* και το *button* για να ολισθαίνει το μήνυμα *(Μέρος Γ)*.



Πιο αναλυτικά:

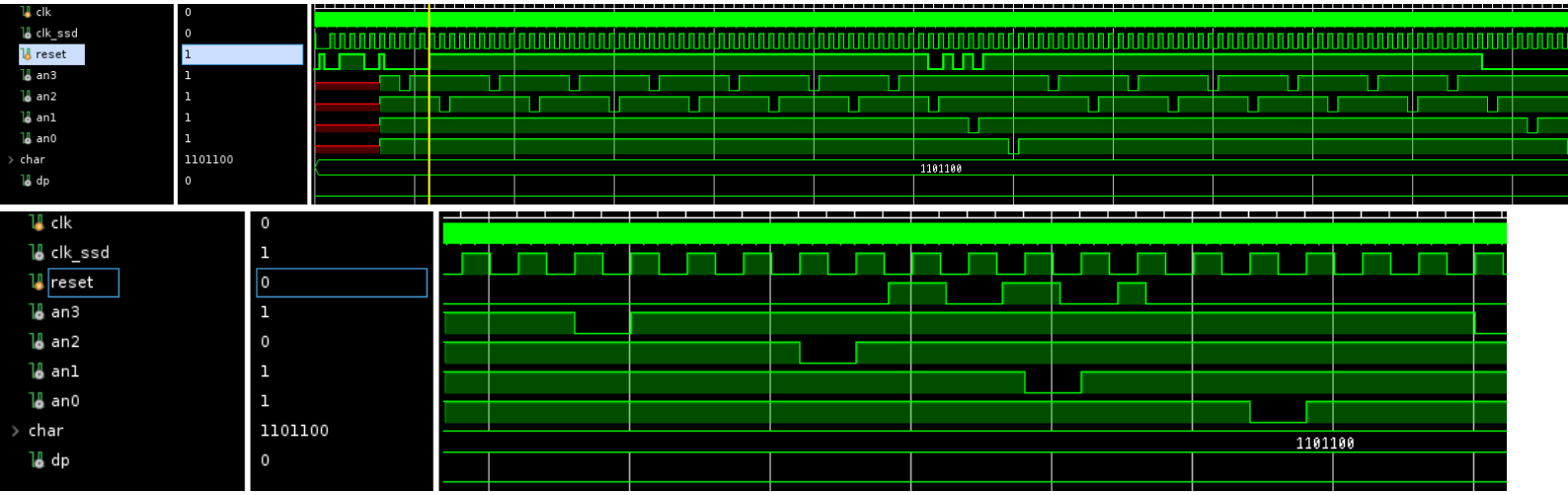
- Δύο σε σειρά ***always*** *blocks* ακολουθιακής λογικής για τον κρίσιμο χρόνο που παίρνουν το εξωτερικό σήμα και το *"περνάνε"* μέσα στο κύκλωμα μετά το <u>*hold*</u> και πριν το <u>set-up time</u>.
- Ένα ***always*** *block* ακολουθιακής λογικής που *"περνάει"* το σήμα στο υπόλοιπο κύκλωμα μόνο όταν αυτό διατηρήσει την τιμή του *θετική* για ένα προκαθορισμένο χρονικό διάστημα.

<u>*Σημείωση:*</u> Όλα τα ***always*** της μονάδας έχουν *λίστα ευαισθησίας* την θετική ακμή του ρολογιού.



Επαλήθευση

Για την επαλήθευση της μονάδας ***FourDigitLEDdriver*** δοκιμάζουμε το *anti-bounce* και αν οι άνοδοι οδηγούν σωστά το ψηφίο συνοδευμένο από τον σωστό χαρακτήρα σε κάθε περίπτωση.



Σημείωση: Για την προσημείωση χρησιμοποιείται *counter* στον ***anti-bounce*** με μικρότερο μέγεθος (*των 2-bit αντί για 22-bit*).

Μέρος Γ — Βηματική Περιστροφή Μηνύματος με Χρήση Κουμπιού

Το μέρος Γ αποτελεί επέκταση του μέρους Β ως προς τους χαρακτήρες προς προβολή. Για να μπορέσουμε σε ένα κύκλωμα να προβάλουμε μέσω της οθόνης όλους του χαρακτήρες που μπορεί να αποκωδικοποιήσει η μονάδα *LEDdecoder*. Έτσι με την χρήση *εξωτερικού* σήματος μέσω κουμπιού θέλουμε να μπορούμε να ολισθήσουμε το κείμενο έναν χαρακτήρα την φορά.

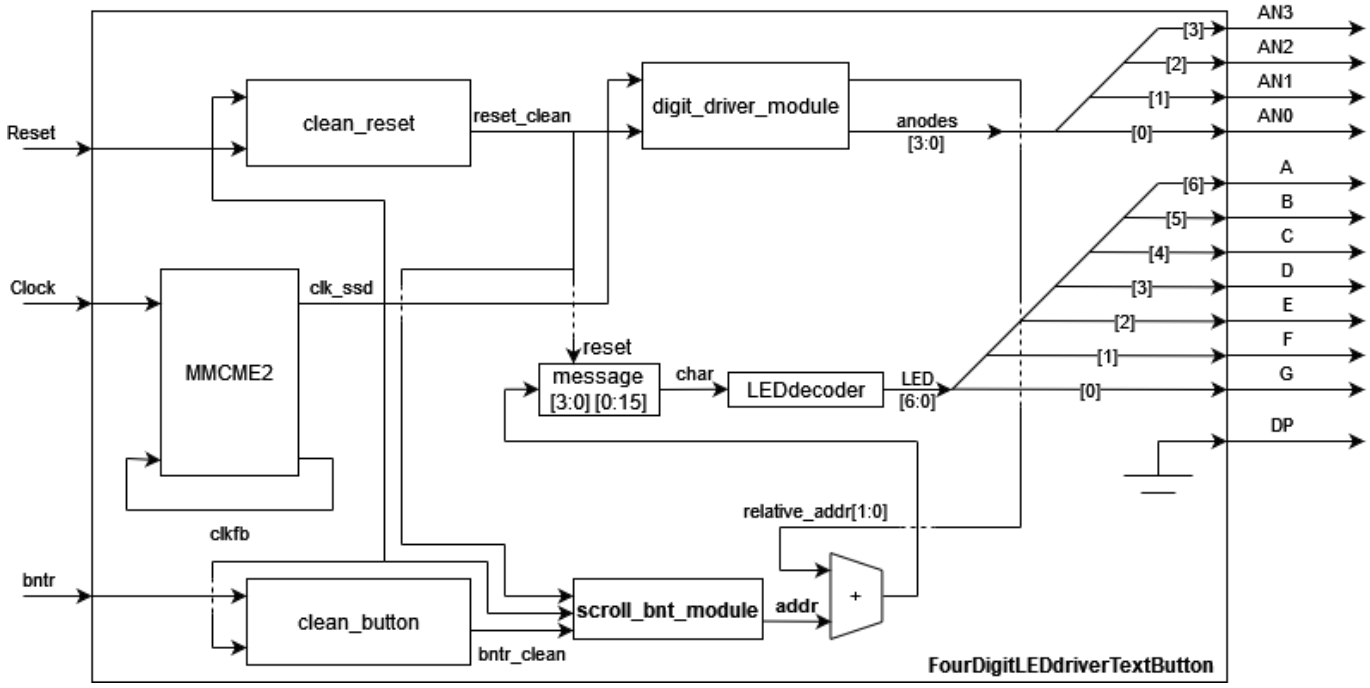
Αρχείο Περιορισμών

Για να χρησιμοποιήσουμε το κουμπί πρέπει να ορίσουμε τις παραμέτρους του στο αρχείο περιορισμών προσθέτοντας το τέλος του αρχείου:

```
set_property -dict { PACKAGE_PIN M17 IOSTANDARD LVCMOS33 } [get_ports { btnr}];
```

Μονάδα FourDigitLEDdriverTextButton

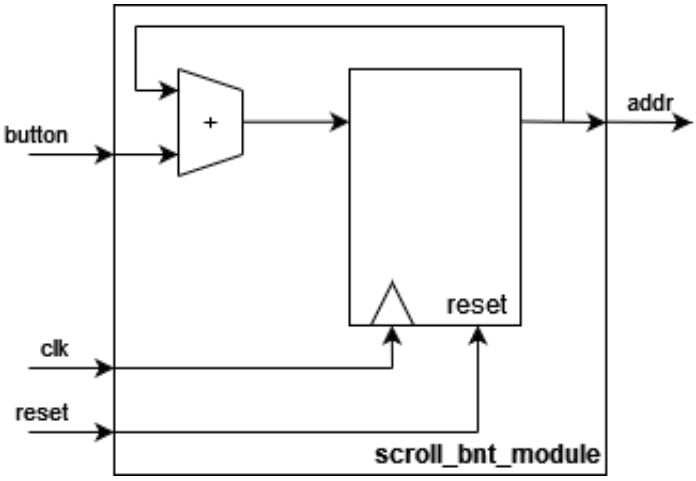
Η μονάδα αυτή αποτελεί τροποποίηση της ***FourDigitLEDdriver*** καθώς έχουμε μία παραπάνω είσοδο (το κουμπί ***btnr*** που ορίσαμε προηγουμένως) και μια έξτρα μονάδα την ***scroll_bnt_module***, για την αλλαγή του μηνύματος, και ένα έξτρα *instance* για ***anti-bounce*** του *bntr.*



Μονάδα *scroll_bnt_module*

Αυτό το module περιέχει τον μηχανισμό για την ολίσθηση του κειμένου με χρήση κουμπιού.

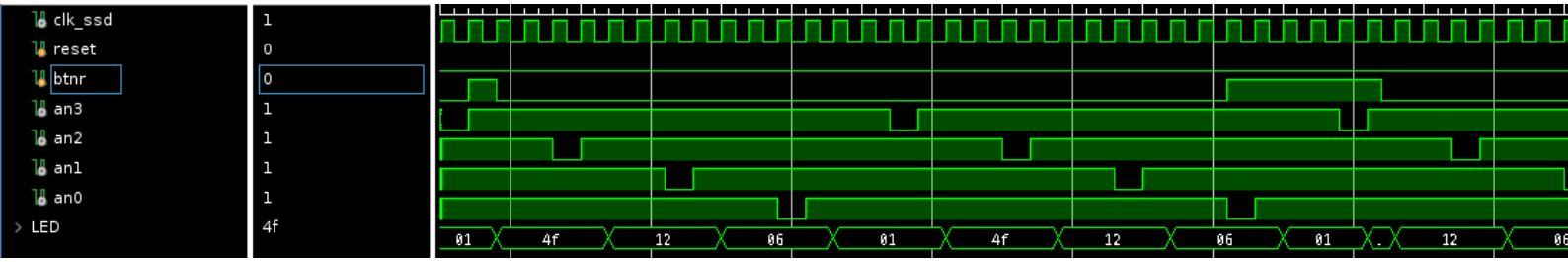
Πιο αναλυτικά η μονάδα έχει ως έξοδο έναν ***δείκτη*** σε μνήμη *addr* ο οποίος για κάθε *θετική* ακμή του ρολογιού αυξάνεται κατά ***ένα***, ενώ σε ασύγχρονο *reset* επιστρέφει στην *αρχική τιμή* του 0.



Για παράδειγμα ενώ αρχικά ο δείκτης έχει την τιμή 0 και η οθόνη προβάλει τον 1ο, 2ο, 3οκαι 4ο χαρακτήρα της μνήμης μετά το πάτημα του κουμπιού η οθόνη θα προβάλει τον 2ο, 3ο, 4οκαι 5ο.

Επαλήθευση

Εφόσον είμαστε σίγουροι ότι το ***reset*** λειτουργεί (*μαζί του και ο μηχανισμός* ***anti-bounce***) από το προηγούμενο μέρος της εργασίας δεν θα ασχοληθούμε σε βάθος με αυτό. Στο πάγκο δοκιμών θέλουμε να επαληθεύσουμε ότι λειτουργεί σωστά η **περιστροφή του κειμένου**.



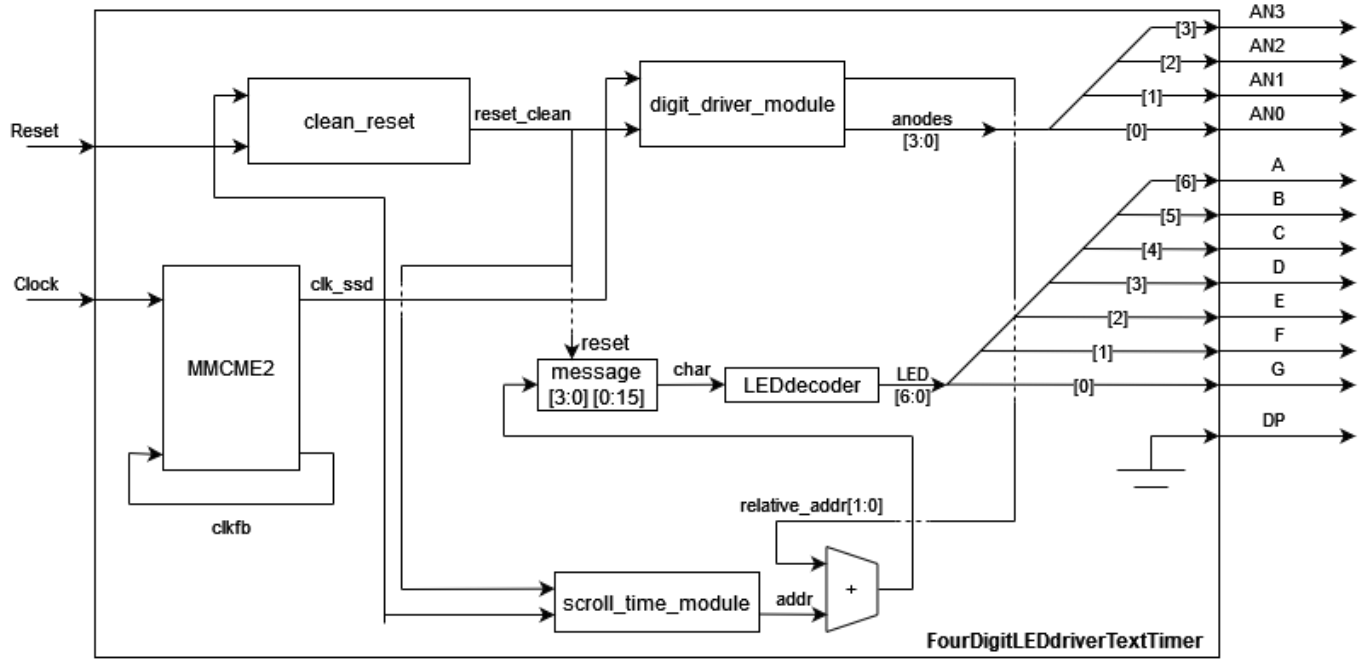
Σημείωση: Για την προσημείωση χρησιμοποιείται *counter* στον ***anti-bounce*** με μικρότερο μέγεθος (*των 2-bit αντί για 22-bit*).

Μέρος Δ — Βηματική Περιστροφή Μηνύματος με Σταθερή Καθυστέρηση

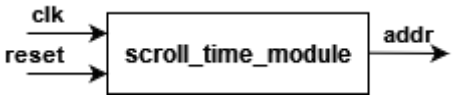
Στην τελική φάση της εργασίας μας ζητείται να υλοποιήσουμε την βηματική περιστροφή αυτή την φορά μέσω κάποιας σταθερής καθυστέρησης.

Μονάδα *FourDigitLEDdriverTextTimer*

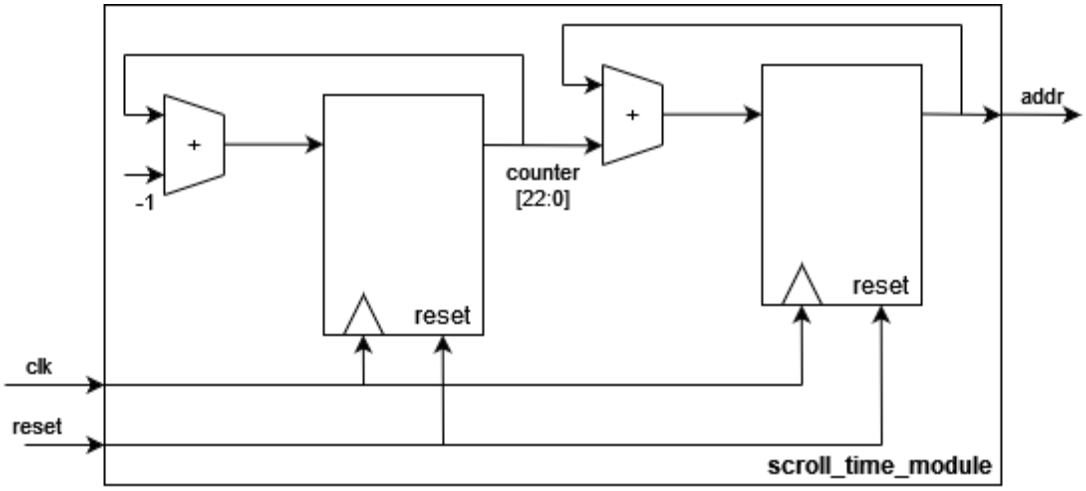
Είναι πανομοιότυπο με το ***FourDigitLEDdriverTextButton*** με μόνες διαφορές ότι αντί για *scroll_bnt_module* εδώ έχουμε ***scroll_time_module***, και ότι δεν υπάρχει *είσοδος* ***bnt*** (*μαζί με το τον μηχανισμό για το anti-bounce του*).



Μονάδα *scroll_time_module*

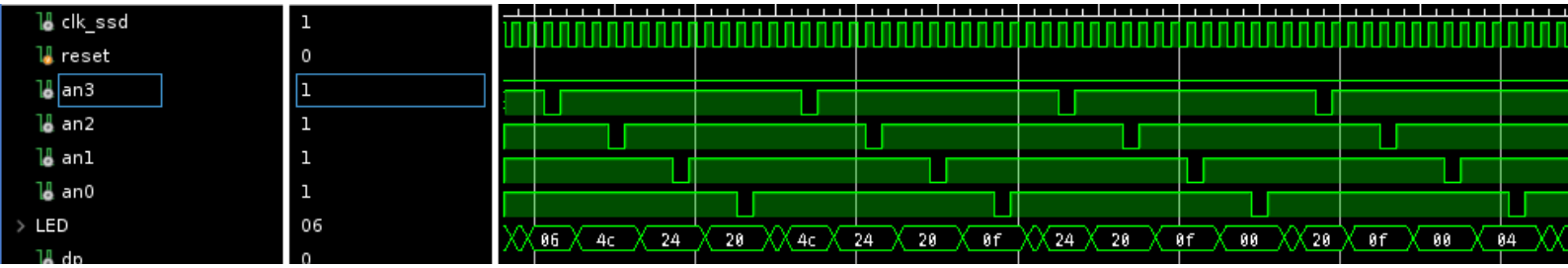


Για να το καταφέρουμε αυτό θα χρησιμοποιήσουμε έναν ***counter 23-bit***, *όπως αναφέρεται στην εκφώνηση*, με την σταθερή καθυστέρηση των *1,6777214* δευτερολέπτων. Έτσι υλοποιούμε μία νέα μονάδα μέσα στην οποία κάθε φορά που μηδενίζει ο counter αυξάνεται η διεύθυνση κατά ένα. Δηλαδή:



Επαλήθευση

Για να ελέγξουμε για την σωστή λειτουργία του κυκλώματος απλά τρέχουμε την προσομοίωση και περιμένουμε να δούμε το κείμενο να αλλάζει με ολίσθηση έναν χαρακτήρα την φορά.



<u>Σημείωση:</u> Για την προσημείωση χρησιμοποιείται *counter* με <u>μικρότερο</u> μέγεθος (*των 4-bit αντί για 23-bit*).