Σταθουδάκης Σταύρος, AEM:03512

# 17/11/2023 Lab1
# 7 Segment Display Driver

## Περίληψη

## Εισαγωγή

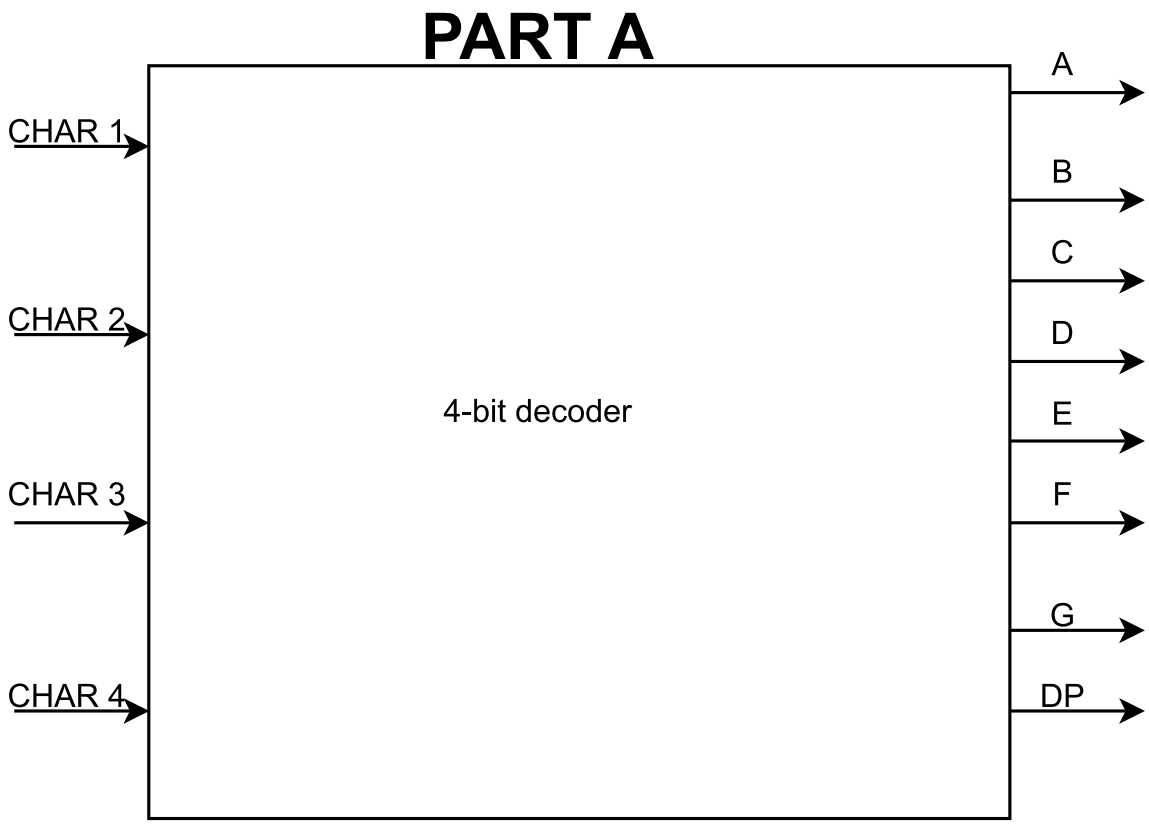
Οι στόχοι της εργασίας ήταν να δημιουργήσουμε ένα περιστροφικό μύνημα σε ένα LED display 4 χαρακτήρων. Η εργασία στην ουσία μας βοήθησε να δημιουργήσουμε ένα ένα τα modules τα οποία θα χρησιμοποιούσαμε για να καταφέρουμε να φτάσουμε στον τελικό μας στόχο. Ενδιάμεσα μας έβαλε να δημιουργήσουμε κάποιες πιο εύκολες μορφές αυτού του τελικού στόχου έτσι ώστε να προλάβουμε να εξοικειωθούμε. Πχ για να καταφέρουμε να έχουμε περιστροφικό μύνημα, έπρεπε πρώτα να καταφέρουμε να δείχνουμε το μύνημα στην οθόνη. Αυτός ήταν ο σκοπός του PartB (το πιο δύσκολο για εμένα). Έπειτα έπρεπε να περιστρέφουμε το μύνημα με το πάτημα ενός κουμπιού (PartC) και τέλος να περιστρέφεται μόνιμα από μόνο του μετα από ένα συγκεκριμένο χρονικό διάστημα. Επιτεύχθηκαν όλοι αυτοί οι στόχοι.

## Μέρος Α - Υλοποίηση Αποκωδικοποίησης 7-τμημάτων

### Υλοποίηση

- Το κύκλωμα π δημιουργείτε σε αυτό το μέρος είναι ένας 4-bit αποκωδικοποιητής ο οποίος έχει 8 outputs, τα οποίο στην ουσία επιλέγουν ποία τμήματα του LED display θα είναι

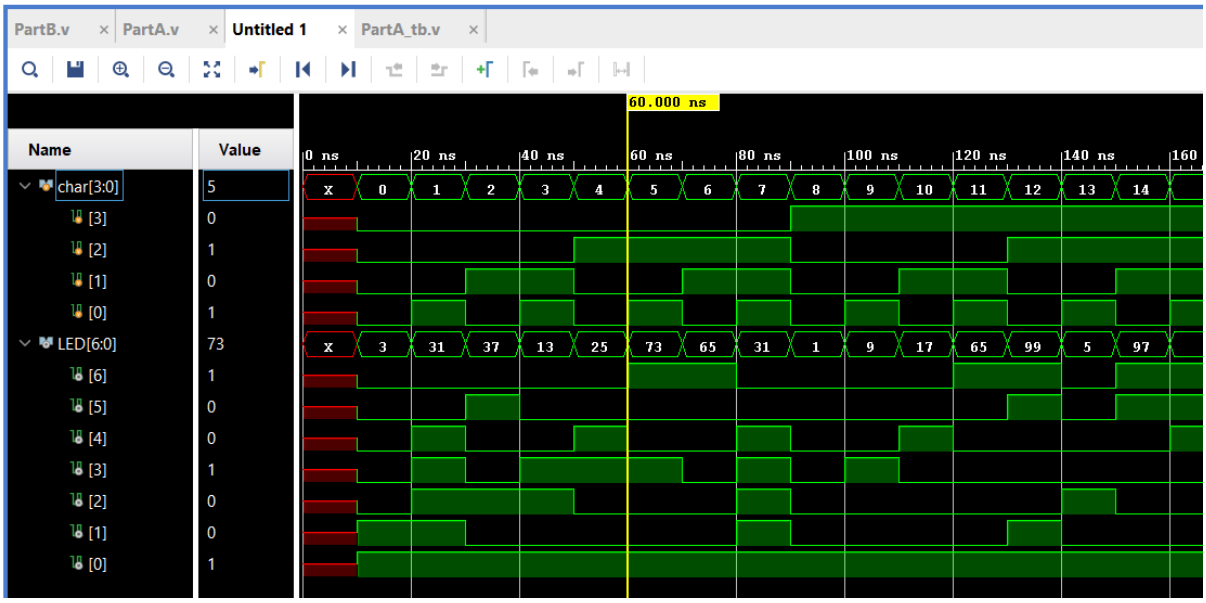
αναμένα, για να δημιουργηθεί το κατάλληλο σχήμα.




- Ο κώδικας περιέχει μία ασύχρονη always ή οποία έχει ως έισοδο 4 bit και έξοδο 8 bit.
- Μέσα στην always έχει χρησιμοποιθεί μία case εντολή για να κάνει τον κώδικα πιο ευανάγνωστο. Τα assignments μέσα στην Verilog είναι blocking. Στην case εμπεριέχονται όλες οι πιθανές τιμές του char για την αποφυγή δημιουργίας Latches.

## Επαλήθευση

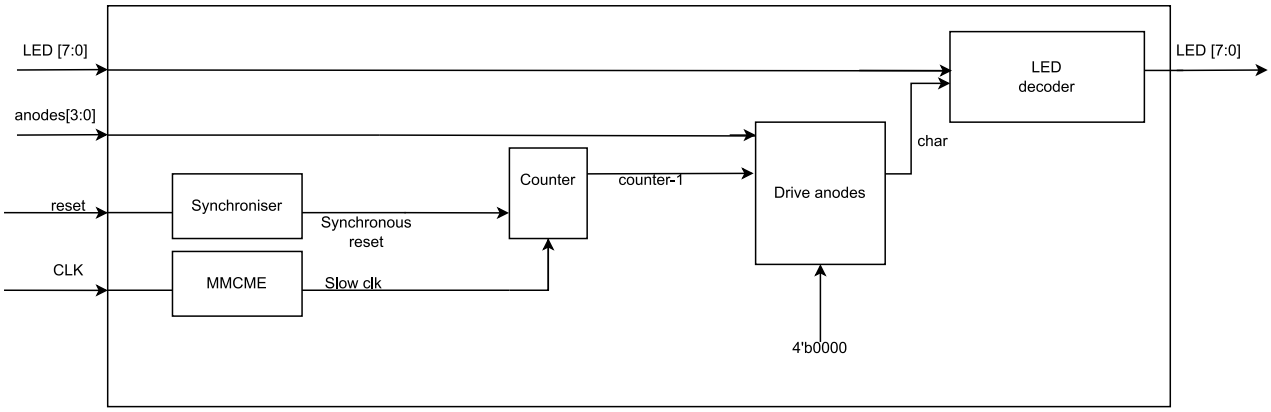
- Στο testbench δοκιμάζονται όλες οι πιθανές τιμές που υπάρχουν ως input (16 τιμες αφού ειναι 4 bit) και έπειτα επαληθεύονται εξετάζοντας τις κυματομορφές των σημάτων εξόδου. Αυτό γίνεται βλέπωντας στο κύκλωμα τι τιμή θα έπρεπε να έχουν οι έξοδοι ανάλογα με το char.
- Το ποσοστό κάλυψης του κώδικα είναι 100% καθώς όπως ανέφερα δοκιμάζονται όλες οι πιθανές τιμές που υπάρχουν ως input.
- Δεν υπήρχαν προβλήματα
- Παρακάτω έχω μία φωτογραφία από τις κυματομορφές της προσομοίωσης, η οποία επαληθεύει την ορθή λειτουργία του πρώτου μέρους. Πράγματι όπως βλέπουμε για κάθε τιμή του char, το LED bus αλλάζει, που είναι και αυτό που επιδιώκουμε.




# Μέρος Β - Οδήγηση Τεσσάρων Ψηφίων

## Υλοποίηση

- Στον κώδικα εμπεριέχονται:
  - MMCME module: αφορά την μορφοποίηση του ρολογιού στα επιθυμητά επίπεδα
  - Synchronizer module: συγχρονιζει το reset
  - Counter Module: Διαχειρίζετε τον counter ο οποίος είναι υπεύθυνος για την οδήγηση των ανόδων
  - Anodes module: Οδηγεί τις ανόδους και αλλάζει το output char.
  - LED decoder: PartA



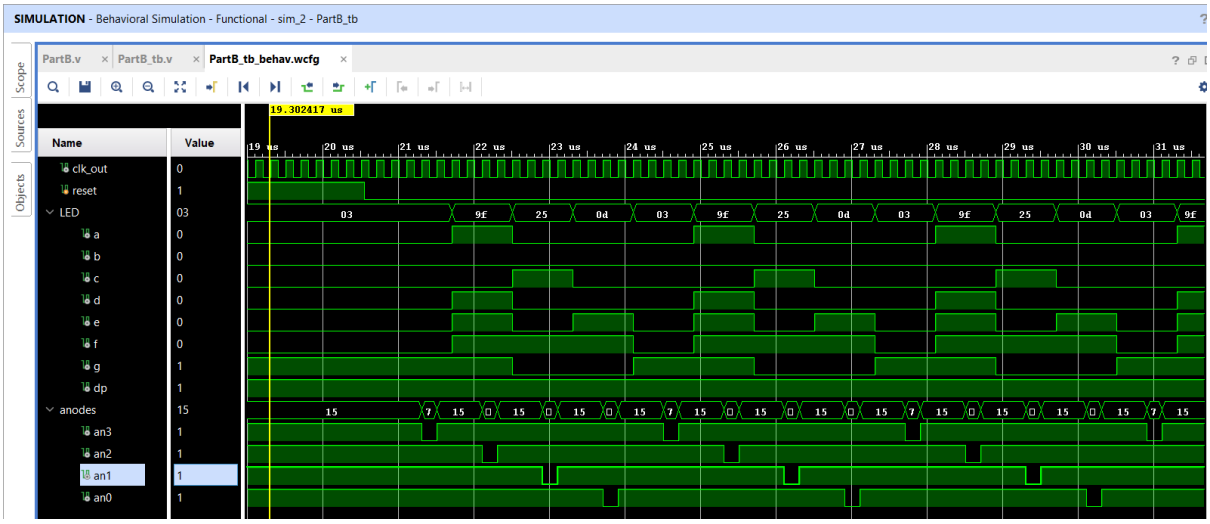

- Μονάδες
  - MMCME module: Μειώνω την περίοδο του ρολογιού από 10 ns (10 MHz) πολλαπλασιάζοντα την αρχική συχνότητα του ρολογιού με 6 κάνοντας την συχνότητα 600 MHz (όπου είναι η minimum τιμή λειτουργίας για το ρολόι) και έπειτα το διαιρώ με 120 για να φέρω την περίοδο στην επιθυμητή τιμή, η οποία είναι 200 ns (5 MHz).

    - Synchronizer module: Περνάω το reset από 3 συγχρονα sequential always (3 flip-flop) και μία XOR (μέσω μίας always) για να αποκλείσω όποια περίπτωση μεταστάθειας.
    - Counter Module: Μειώνω τον counter κατα 1 σε κάθε θετική ακμή του ρολογιού εκτός και αν έχω reset=1 όπου τότε επαναφέρω τον counter στην αρχική του τιμή, η οποία είναι 4'd15. Αυτό το κάνω με την χρήση μίας always και ενός if/else statement.
    - Anodes module: Οπώς προτείνει η εκφώνηση, χρησιμοποιώ έναν 4-bit counter για να αφήσω περιθώριο στους πυκνωτές του LED display να αποφωρτίσουν. Αυτό το καταφέρνω αφήνοντας κενό ένα κύκλο ανάμεσα απο την αλλαγή της ένδειξης και την οδήγηση της κατάλληλης ανόδου. Σαν είσοδο το module αυτό έχει και το 4'b0000, αυτό υπάρχει διότι εμείς σε αυτό το μέρος της εργασίας θέλουμε απλά το μύνημα μας να μένει σταθερό, ενώ για τα επόμενα μέρη αυτό αλλάζει. Οπότε έχω προσαρμόσει το module αυτό ώστε να είναι συμβατο με όλα τα μέρη. Αυτό το καταφέρνω με την χρήση μίας σύγχρονης ακολουθιακής always και μία switch case όπου να περιέχει όλες τις πιθανές τιμές του counter για την αποφυγή δημιουργίας Latches.
- Γενικά ο κώδικας αποτελείται από το main module (FourDigitLEDdriver) το οποίο περιέχει μόνο τα MMCME modules και instantiations όλων των υπόλοιπων modules που ανέφερα παραπάνω. Αυτό γίνεται για να υπάρχει μία τάξη στον κώδικα και να είναι πιο εύκολη η αντιμετώπιση πιθανών προβλημάτων. Επίσης όλα αυτα τα modules τα χρησιμοποιώ και στα επόμενα μέρη της εργασίας, οπότε με αυτό τον τρόπο δεν χρειάστηκε να τα ξαναγράψω. Ο κώδικας περιέχει σχόλια στα αγγλικά σε όλα τα σημεία του, έτσι ώστε να είναι ευανάγνωστος και κατανοητός.

## Επαλήθευση

- Το testbench περιλαμβάνει τον ορισμό ενός timescale και την παραγωγή ενός ρολογιού 10 ns το οποίο προσομοιώνει το ρολόι της fpga. Αρχικοποιεί τις τιμές του clock και του reset, για να μην είναι undefined (X) και έπειτα εναλάζει τις τιμές του. Για να Ελεγξουμε την ορθή λειτουργία του κυκλώματος, αρκεί να δούμε ότι οι άνοδοι ενεργοποιούνται με διαφορά 3 κύκλων ρολογιού και ότι οι ενδείξεις παραμένουν οι ίδιες για 4 κύκλους. Όλα αυτά εξαρτούντε από τον counter.
- Επομένως για να σιγουρευτούμε ότι λειτουργεί σωστά το κύκλωμα μας αρκεί να δούμε τα outputs του κυκλώματος ότι λειτουργούν σωστά. Για Διάνυσμα ελέγχου χρησιμοποίησα το char το οποίο συνδέει στην ουσία το modules των ανόδων και του PartA. Επίσης στην αρχή είχα και το αργο clock που δημιουργούμε απο το MMCME, για να σιγουρευτώ ότι έκανα τις σωστές πράξεις. Το ποσοστό κάλυψης λειτουργίας του κυκλώματος είναι 100% καθώς ελέγχουμε όλες τις πιθανές συμπεριφορές. Δηλαδή και σε κατάσταση ηρεμίας αλλά και στην περίπτωση όπου πατηθεί το reset.
- Το μοναδικό πρόβλημα που αντιμετώπισα είναι με το σετάρισμα του clock, μέχρι να καταλάβω ακριβώς την λειτουργία του. Με βοήθησαν οι κυματομορφές γιατί έτσι έβλεπα αν η περίοδος ηταν στην επιθυμητή τιμή (200 ns), επομένως και αν χρειάζεται κάποια αλλαγη.
- Παρακάτω παραδίσω μία φωτογραφία από τις κυματομορφές της προσμοίωσης. Εκεί μπορεί να φανεί το clk_out όπου είναι το αργό ρολόι των 200 ns που καλούμαστε να δημιουργήσουμε. Επίσης μπορούμε να παρατηρήσουμε ότι κάθε 3 κύκλους ενεργοποιείται μία από τις 4 ανόδους και ότι το LED display αλλάζει και αυτό κάθε 3 κύκλους, αφήνωντας χρόνο τους πυκνωτές της πλακέτας να φορτίσουν και να αποφορτίσουν. Επίσης φαίνεται ότι

όταν το reset=1 δεν οδηγείται καμία άνοδος. Ακόμα φαίνεται και η λειτουργία του synchronizer καθώς αφού το reset από την πλακέτα γίνει 1, υπάρχει περιθώριο 3 κύκλων μέχρι να περάσει αυτό το μήνυμα στην πλακέτα.




## Πείραμα/Τελική Υλοποίηση

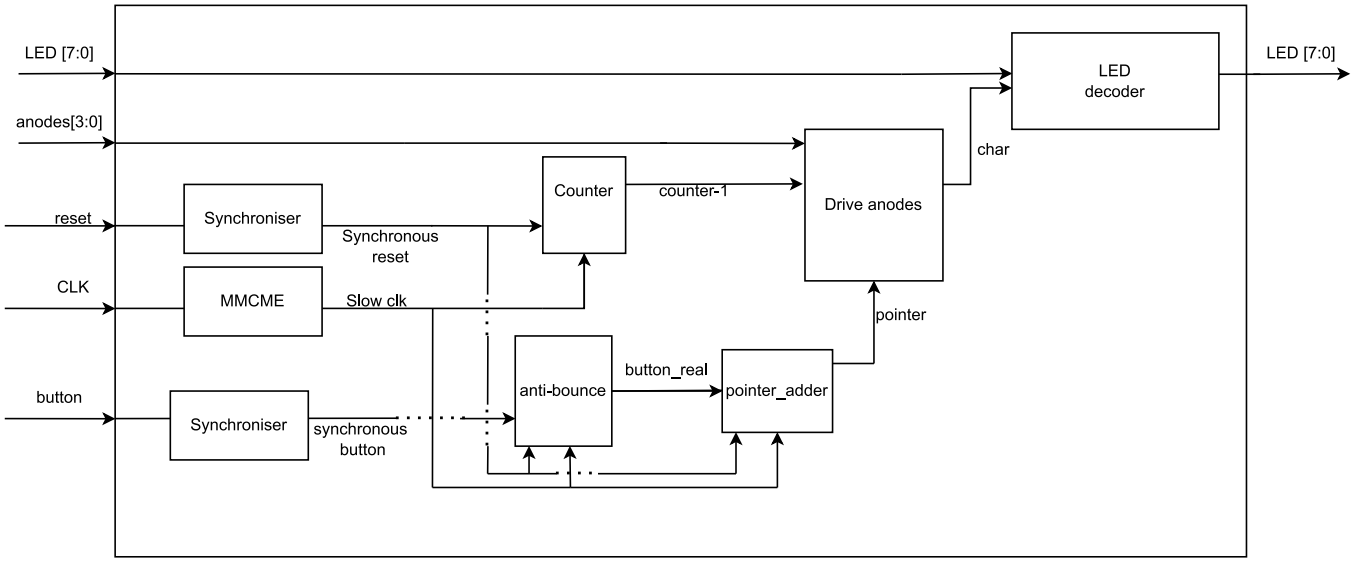
- Αρχικά ακόμα και αν δεν είχα πρόβλημα στο testbench, μου εμφανιστηκαν κάποια errors και κάποια critical warnings στην σύνθεση. Πιο πολύ ήταν κάποια λάθη διαδικαστικά καθώς ακόμα δεν ήξερα το workflow του vivado ακόμα. Δεν χρειάστηκαν γενικά πολλές υλοποιήσεις καθώς το κύκλωμα ήταν σχετικά απλό.
- Εφόσον είχα επιλύσει τα προβλήματα με την σύνθεση και την υλοποίηση του κώδικα, το μόνο πρόβλημα που αντιμετώπισα ήταν ότι οι ενδείξεις ήταν ανάποδα επειδή στην εκφώνηση λέει ότι το σχετικο σήμα καθόδου οδηγείται στο 1 ενώ πάνω στην πλακέτα οδηγείται στο 0.
- Δεν είχα ηλεκτρικά προβήματα
- Άλλαξα μόνο τις ενδείξεις στο PartA που ήταν ανάποδα λόγω της εκφώνησης, δλδ έκανα 1->0 και τα 0->1.

# Μέρος Γ

## Υλοποίηση

Στον κώδικα εμπεριέχονται:

- MMCME module: αφορά την μορφοποίηση του ρολογιού στα επιθυμητά επίπεδα
- Synchronizer module: συγχρονιζει το reset και το button
- Counter Module: Διαχειρίζετε τον counter ο οποίος είναι υπεύθυνος για την οδήγηση των ανόδων
- Anodes module: Οδηγεί τις ανόδους και αλλάζει το output char.
- LED decoder: PartA
- Anti-bounce module: Ομαλοποιεί το ασταθές σήμα του κουμπιού
- Pointer-Adder module: Αλλάζει την ένδειξη στο LED display



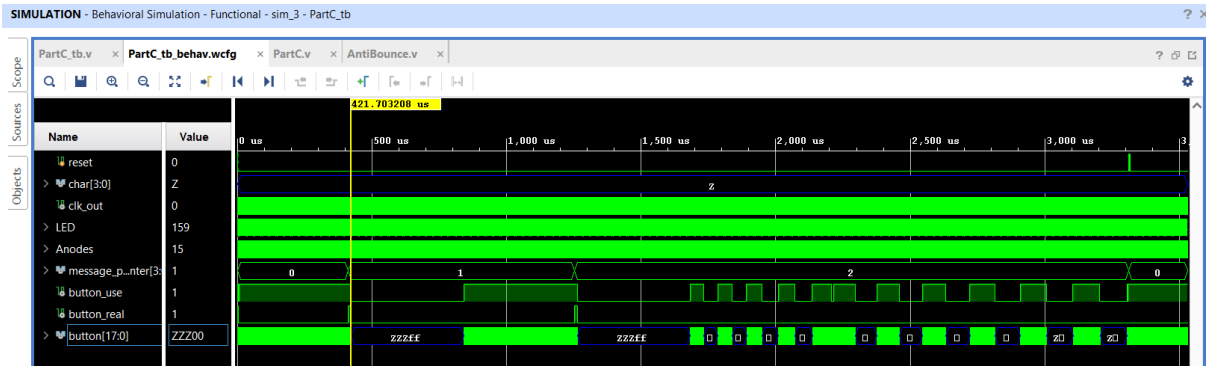

- Modules:
  - MMCME module: Μειώνω την περίοδο του ρολογιού από 10 ns (10 MHz) πολλαπλασιάζοντα την αρχική συχνότητα του ρολογιού με 6 κάνοντας την συχνότητα 600 MHz (όπου είναι η minimum τιμή λειτουργίας για το ρολόι) και έπειτα το διαιρώ με 120 για να φέρω την περίοδο στην επιθυμητή τιμή, η οποία είναι 200 ns (5 MHz).
  - Synchronizer module: Περνάω το reset και το κουμπί από 3 συγχρονα sequential always (3 flip-flop) και μία XOR (μέσω μίας always) για να αποκλείσω όποια περίπτωση μεταστάθειας.
  - Counter Module: Μειώνω τον counter κατα 1 σε κάθε θετική ακμή του ρολογιού εκτός και αν έχω reset=1 όπου τότε επαναφέρω τον counter στην αρχική του τιμή, η οποία είναι 4'd15. Αυτό το κάνω με την χρήση μίας always και ενός if/else statement.
  - Anti-bounce module: Κάθε φορά όπου έρχεται σήμα 1 από το κουμπί (M17) τότε αρχίζει να μετράει ένας counter (52 ms) μέχρι να μηδενιστεί ή το σήμα να προλάβει να γίνει 0 παλι. Αυτό γίνεται επειδή δεν θέλουμε κάθε φορά που το σήμα του κουμπιού είναι ένα να το περάσουμε στο κύκλωμα ως 1, διότι πολλές φορές αφού πατάμε το κουμπί στην αρχή υπάροχυν κάποια αναπηδήματα που είναι ανεπιθύμητα. Όταν λοιπόν προλάβει να μηδενιστεί αυτός ο counter (δηλαδή όταν σταθεροποιηθεί το σήμα), τότε περνάω μία μετανλητή (button real) στο κύκλωμα ως 1, η οποία θα παίζει τον ρόλο του κουμπιού. Αυτό υλοποιείται με μία ακολουθιακή ασύγχρονη always και έναν 18-bit counter.
  - Pointer_Adder module: Μετά απο αυτό περνάμε σε μία ακόμα ακολουθιακή ασύγχρονη always, η οποία αυξάνει τον pointer (δείχνει σε ποιό αρχικό στοιχείο του μυνήματος βρισκόμαστε) μόνο αν το button real, που παίρνω από την προηγούμενη always είναι 1. Για να αποφύγω την συνεχή κίνηση του pointer, χρησιμοποιώ ένα flag (στο πρόγραμμα ονομάζεται second_counter) το οποίο στην ουσία μου λέει αν απο την στιγμή που το button_real έγινε 1 έχει αυξηθεί ο pointer ή όχι. Με αυτό τον τρόπο αποτρέπω την συνεχή κύλιση των ψηφίων όσο το button_real είναι 1.
  - Anodes module: Οπώς προτείνει η εκφώνηση, χρησιμοποιώ έναν 4-bit counter για να αφήσω περιθώριο στους πυκνωτές του LED display να αποφωρτίσουν. Αυτό το καταφέρνω αφήνοντας κενό ένα κύκλο ανάμεσα απο την αλλαγή της ένδειξης και

την οδήγηση της κατάλληλης ανόδου. Σαν είσοδο το module αυτό έχει και το message[pointer] το οποίο είναι το starting στοιχείο το οποίο θα βρίσκετε στο πιο αριστερό display. Με αυτό τον τρόπο καταφέρνω ότι κάθε φορά που αυξάνουμε το pointer, αλλάζει και το display. Αυτό το καταφέρνω με την χρήση μίας σύγχρονης ακολουθιακής always και μία switch case όπου να περιέχει όλες τις πιθανές τιμές του counter για την αποφυγή δημιουργίας Latches.

- Γενικά ο κώδικας αποτελείται απο το main module (ButtonRotate) το οποίο περιέχει μόνο τα MMCME modules και instantiations όλων των υπόλοιπων modules που ανέφερα παραπάνω. Αυτό γίνεται για να υπάρχει μία τάξη στον κώδικα και να είναι πιο εύκολη η αντιμετώπιση πιθανών προβλημάτων. Ο κώδικας περιέχει σχόλια στα αγγλικά σε όλα τα σημεία του, έτσι ώστε να είναι ευανάγνωστος και κατανοητός.

## Επαλήθευση

- Το testbench περιλαμβάνει τον ορισμό ενός timescale και την παραγωγή ενός ρολογιού 10 ns το οποίο προσομοιώνει το ρολόι της fpga. Αρχικοποιεί τις τιμές του clock και του reset, για να μην είναι undefined (X) και έπειτα εναλάζει τις τιμές του. Για να Ελεγξουμε την ορθή λειτουργία του κυκλώματος, αρκεί να δούμε ότι με κάθε πάτημα του κουμπιού το οποίο είναι πάνω απο 52 ms η ένδειξη αυξάνεται μονάχα κατα ένα. Για αυτό τον λόγο στην ουσία αυτό που προσομοιώνω είναι το κουμί. Δηλαδή το ανοιγοκλείνω για διαστήματα > 52 ms και < 52 ms. Ταυτόχρονα πειράζω το reset ανάμεσα απο ανοιγοκλείσματα του κουμπιού για να βεβαιωθώ ότι λειτουργεί σωστά.
- Τα διανύσματα ελέγχου είναι αρκετά γιατί η επαλήθευση έπερεπε να γίνει τσεκάροντας πολλά σήματα. Αρχίζοντας με τον μεγάλο counter (18 bit), για να βεβαιωθώ ότι τριγκάρει όντως, το char από τα προηγούμενα part, τον pointer και ακόμα και το message[pointer]. Το ποσοστό κάλυψης λειτουργίας του κυκλώματος θεωρώ ότι είναι κοντά στο 100% αν όχι τόσο. Αυτό το λέω επειδή οι τιμές που επηρεάζουν το κύκλωμα τεστάρονται εκτενώς.
- Με βοήθησε να καταλάβω την λάθος λογική που είχα εφαρμόσει το module anti-bounce (δεν τρίγκαρε εξαρχής ο counter). Ο λόγος που το έκανα αυτό είναι επειδή μπέρδεψα κάποια if/else statements.
- Παρακάτω είναι μία φωτογραφία των κυματομορφών από την προσομοίωση του κυκλωματος. Δεν θα σταθώ στην σωστή λειτουργία των modules όπου χρησιμοποιήθηκαν και στα προηγούμενα parts, αλλά θα σταθώ σε αυτά όπου είναι μόνο για αυτό το part. Πριν προχωρήσω στην εξήγηση των σημάτων, πρέπει να αναφερθώ στα μυνήματα και να εξηγήσω τι είναι το καθένα. Καταρχάς το char είναι z διότι δεν το πέρασα στο testbench στην συγκεκριμένη προσομοίωση. Η μεταβλητή button είναι ο counter που έχει το antibounce (για λόγους ευκολίας στην προσομοίωση, τον έχω μειώσει στα 10 bit. Γιαυτό και είναι τα πρώτα bits). Επίσης το message_pointer είναι το μήνυμα το οποίο δείχνει το πιο αριστερό ψηφίο του LED display. Το button_use είναι το κουμπί που έρχεται από την πλακέτα και υποτίθεται πρέπει να μετακινεί τον pointer. Το button_real είναι ένα flag όπου ενεργοποιείται μόνο όταν έχει περάσει ο timer από το anti-bounce. Εφόσον λοιπόν έχουμε καταλάβει τι είναι το κάθε σήμα μπορούμε να καταλάβουμε ότι το κύκλωμα λειτουργεί σωστά καθώς ο pointer αυξάνεται μονάχα άμα το button_use μείνει παττημένο για περισσότερο χρονικό διάστημα από αυτό που αναφέρεται στον counter. Επίσης αυτό είναι εμφανές ακόμα και μετά όταν ο pointer είναι 2 και το button_use ανοιγοκλείνει πολύ γρήγορα και ο pointer δεν αυξάνεται, όπως και θα έπρεπε. Όταν γίνεται αυτό ξαναρχικοποιείται και ο counter και περιμένει πάλι σήμα από το κουμπί. Τέλος φαίνεται ότι όταν πατάω το reset η τιμή του counter επαναφέρεται στο 0 και το display είναι και πάλι τα 4 πρώτα ψηφία του μυνήματος.



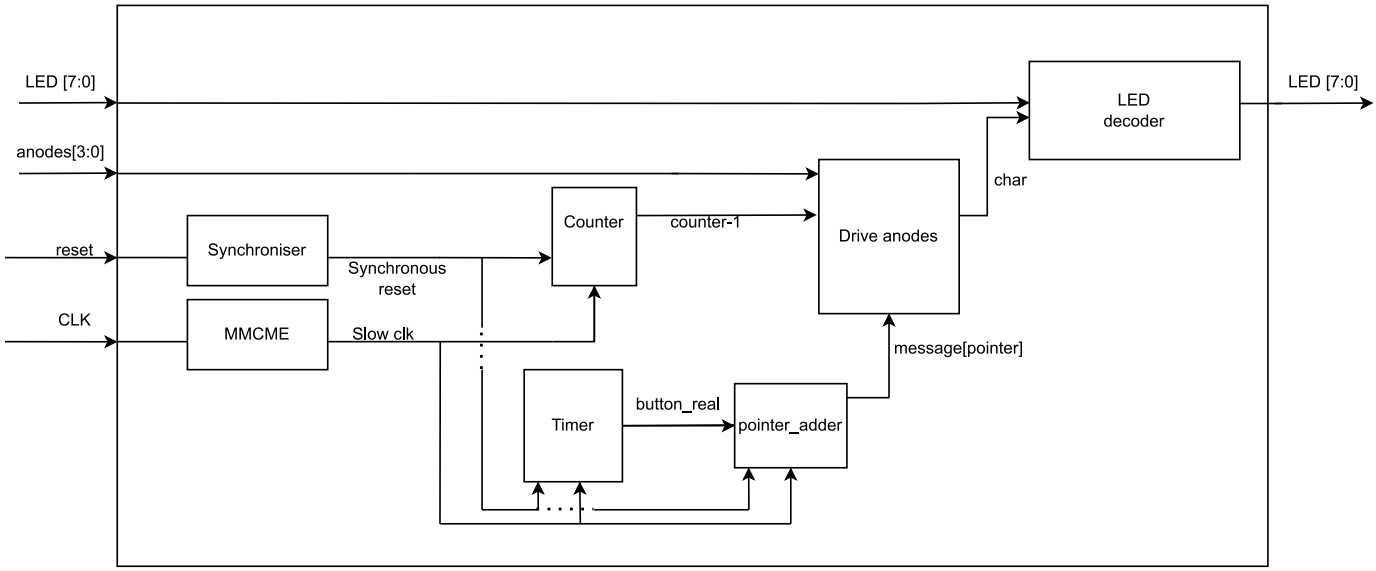

## Πείραμα/Τελική Υλοποίηση

- Αρχικά δεν είχα υλοποιήσει το anti-bounce και απλά δοκίμασας στην πλακέτα ότι λειτουγούν τα υπόλοιπα χωρίς ιδιαίτερα προβλήματα. Έπειτα έκανα και το anti-bounce module και έπειτα απο debugging με τις κυματομορφές, έμενε να θέσω σωστό χρόνο στον counter του anti-bounce.
- Γενικά το θέμα που είχα με τον σωστό προγραμματισμό της πλακέτας ήταν το σετάρισμα του timer του antibounce καθώς αυτό βγήκε πειραματικά. Αρχικά είχα βάλει πολύ μικρό χρονικό διάστημα και παρατηρούσα το φαινόμενο anti-bounce στις ενδείξεις της πλακέτας. Έπειτα έθεσα τον timer σε πολύ υψηλή τιμή και κάποια γρήγορα πατήματα που θα έπρεπε να μετράνε σαν πατήματα δεν περνούσαν στην πλακέτα. Έτσι κατέληξα στον χρόνο που έχει τώρα το πρόγραμμα πάνω.
- Δεν είχα ηλεκτρικά προβλήματα
- Άλλαξα κάποια if conditions που αφορύσαν edge cases και δεν είχα σκεφτεί απο μόνος μου καθώς έγραφα τον κώδικα.

# Μέρος Δ

## Υλοποίηση

- Στον κώδικα εμπεριέχονται:
    - MMCME module: αφορά την μορφοποίηση του ρολογιού στα επιθυμητά επίπεδα
    - Synchronizer module: συγχρονιζει το reset και το button
    - Counter Module: Διαχειρίζετε τον counter ο οποίος είναι υπεύθυνος για την οδήγηση των ανόδων
    - Anodes module: Οδηγεί τις ανόδους και αλλάζει το output char.
    - LED decoder: PartA
    - Pointer-Adder module: Αλλάζει την ένδειξη στο LED display
    - Timer module: Υλοποιεί τον timer που αναφέρει η εκφώνηση




- Modules:
    - MMCME module: Μειώνω την περίοδο του ρολογιού από 10 ns (10 MHz) πολλαπλασιάζοντα την αρχική συχνότητα του ρολογιού με 6 κάνοντας την συχνότητα 600 MHz (όπου είναι η minimum τιμή λειτουργίας για το ρολόι) και έπειτα το διαιρώ με 120 για να φέρω την περίοδο στην επιθυμητή τιμή, η οποία είναι 200 ns (5 MHz).
    - Synchronizer module: Περνάω το reset και το κουμπί από 3 συγχρονα sequential always (3 flip-flop) και μία XOR (μέσω μίας always) για να αποκλείσω όποια περίπτωση μεταστάθειας.
    - Counter Module: Μειώνω τον counter κατα 1 σε κάθε θετική ακμή του ρολογιού εκτός και αν έχω reset=1 όπου τότε επαναφέρω τον counter στην αρχική του τιμή, η οποία είναι 4’d15. Αυτό το κάνω με την χρήση μίας always και ενός if/else statement.
    - Anodes module: Οπώς προτείνει η εκφώνηση, χρησιμοποιώ έναν 4-bit counter για να αφήσω περιθώριο στους πυκνωτές του LED display να αποφωρτίσουν. Αυτό το καταφέρνω αφήνοντας κενό ένα κύκλο ανάμεσα απο την αλλαγή της ένδειξης και την οδήγηση της κατάλληλης ανόδου. Σαν είσοδο το module αυτό έχει και το message[pointer] το οποίο είναι το starting στοιχείο το οποίο θα βρίσκετε στο πιο αριστερό display. Με αυτό τον τρόπο καταφέρνω ότι κάθε φορά που περνάει το συγκεκριμένο χρονικό διάστημα, αλλάζει και το display. Αυτό το καταφέρνω με την χρήση μίας σύγχρονης ακολουθιακής always και μία switch case όπου να περιέχει όλες τις πιθανές τιμές του counter για την αποφυγή δημιουργίας Latches.
    - Timer module: Χρησιμοποιώ μία ασύγχρονη ακολουθιακή always ή οποία μειώνει έναν 23-bit counter συνέχεια εκτός και αν πατήσω reset. Όταν ο counter μηδενιστεί μετά από 1.6s περ’ιπου, ο pointer μετακινείται κατα 1 αριστερά και ο counter επανέρχεται στην αρχική του τιμή και μετά ξαναρχίζει από την αρχή.
- Γενικά ο κώδικας αποτελείται απο το main module (AutoRotate) το οποίο περιέχει μόνο τα MMCME modules και instantiations όλων των υπόλοιπων modules που ανέφερα παραπάνω. Αυτό γίνεται για να υπάρχει μία τάξη στον κώδικα και να είναι πιο εύκολη η αντιμετώπιση πιθανών προβλημάτων. Ο κώδικας περιέχει σχόλια στα αγγλικά σε όλα τα σημεία του, έτσι ώστε να είναι ευανάγνωστος και κατανοητός.

## Επαλήθευση

- Το testbench περιλαμβάνει τον ορισμό ενός timescale και την παραγωγή ενός ρολογιού 10 ns το οποίο προσομοιώνει το ρολόι της fpga. Αρχικοποιεί τις τιμές του clock και του reset, για να μην είναι undefined (X) και έπειτα εναλάζει τις τιμές του. Για να Ελεγξουμε την ορθή λειτουργία του κυκλώματος, αρκεί να δούμε ότι κάθε 1.6s αυξάνετε ο pointer κατά 1 και κατα συνέπεια και όλες οι τιμές που εξαρτώνται απο αυτόν τις οποίες έχω αναφέρει παραπάνω. Για αυτό τον λόγο Απλά πειράζω λίγο το reset καθώς είναι το μόνο input του κυκλώματος το οποίο μπορώ να επηρεάσω άμεσα.
- Διανύσματα ελέγχου είναι προφανώς ο pointer για να δώ άμα μετακινείται όντως κατα 1, ο timer, για να δω αν τριγκάρετε και ο char για να δω αν ο pointer περνάει το μύνημα του στα επόμενα modules μέσω της μνήμης. Εφόσον το κύκλωμα στην ουσία μας δείχνει μόνο αν λειτουργεί ή όχι, το ποσοστο ελέγχου λειτουργίας είναι και εδώ 100%.
- Πάλι όταν είχα μπερδέψει καποια if statements και δεν τρίγκαρε ο timer, με βοήθησαν οι κυματομορφές να καταλάβω και να εντοπίσψω το λάθος μου.
- Παρακάτω είναι μία φωτογραφία από τις κυματομορφές της προσομοίωσης του κυκλώματος. Όπως και πριν δεν θα αναφερθώ στα σήματα που αφορούν τα modules που χρησιμοποιήθηκαν στα προηγούμενα parts. Το σήμα big_counter είναι ο 23-bit counter που μας δίνει την επιθυμητή καθυστέρηση των 1.6 seconds (για λόγους ευκολίας έχω αναθέσει μόνο τα 8 πρώτα bit του counter μόνο για την προσομοίωση). Όπως είναι εμφανές ο pointer αυξάνεται σταθερά κάθε φορά που μηδενίζει ο big_counter.


## Πείραμα/Τελική Υλοποίηση

- Το πρόγραμμα δούλεψε με την πρώτη καθώς ήταν παρεμφερές με το μέρος Γ και δεν χρειαζόταν πολύ debugging. Άρα χρειάστηκε μία μονάχα επανάληψη
- Δεν υπήρξαν δυσκολίες στην δοκιμή του κυκλώματος
- Δεν αντιμετώπισα ηλεκτρικά προβλήματα
- Δεν άλλαξα κάτι στον κώδικα αφού τον δοκίμασα πρακτικά

# Συμπέρασμα

Γενικά το πιο δύσκολο για μένα κομμάτι της εργασίας ήταν το σετάρισμα του MMCME clock καθώς ήθελε πολύ ψάξιμο στα datasheets, αλλά αυτό αντιμετωπίστηκε με το ψάξιμο. Κατά τάλλα με βοήθησε ότι στα δύο τελευταία εργαστήρια είχα έτοιμο τον κώδικα και απλά πήγαινα στο εργαστήριο και τον τέσταρα και αντιμετώπιζα τα προβλήματα που προέκυπταν εκείνη την στιγμή. Τα testbench ήταν αρκετά απλά και δεν είχαν τρομερές απαιτήσεις, που ήταν καλό επειδή διευκόλυνε στο debugging. Ο προγραμματισμός της πλακέτα πήγε αρκετά καλά καθώς δεν χρειάστηκε να δημιουργήσω πολλές φορές bitstream και εφόσον ήταν σωτές οι κυματομορφές, δούλευε και το πρόγραμμα πάνω στην πλακέτα. Το μόνο που ήταν trial and error ήταν η επιλογή του χρόνου για το antibounce. Μονάχα εκεί χρειάστηκε να κάνω bitstream αρκετές φορές.