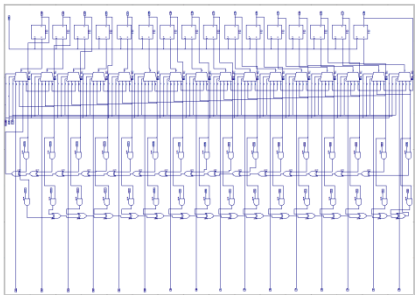
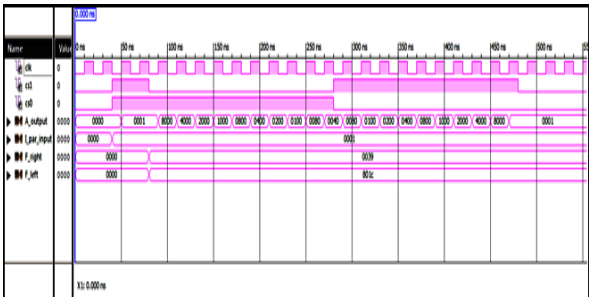
Τ.Ε.Ι. ΚΡΗΤΗΣ / ΠΑΡΑΡΤΗΜΑ ΧΑΝΙΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΗΣ

Μελέτη, σχεδίαση και υλοποίηση σε FPGA, προγραμματιζόμενων LFSR με δυνατότητα παραγωγής τόσο της ορθής όσο και της ανάστροφης ακολουθίας τους.





ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εμμανουήλ Σαλούστρου

Επιβλέπων : Δρ. Μηχ. Νικόλαος Στ. Πετράκης
Καθηγητής Εφαρμογών

Χανιά 2012

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα κ. Πετράκη Νικόλαο, Καθηγητή Εφαρμογών του τμήματος Ηλεκτρονικής του Τεχνολογικού Εκπαιδευτικού Ιδρύματος Κρήτης για τις πολύτιμες συμβουλές και κατευθύνσεις που μου παρείχε για την ολοκλήρωση αυτής της πτυχιακής εργασίας, αλλά και για την άψογη συνεργασία που είχαμε καθ' όλη την διάρκεια των σπουδών μου.

Επίσης, θα ήθελα να ευχαριστήσω θερμά τον φοιτητή Τζαβάρα Σάββα, για την βοήθειά του στην σχεδίαση της εφαρμογής και στη συγγραφή του κώδικα.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου για την ηθική αλλά και οικονομική στήριξη που μου παρείχαν.

Περίληψη

Η παρούσα πτυχιακή εργασία αναφέρεται στη μελέτη, στη σχεδίαση και στην υλοποίηση προγραμματιζόμενων LFSR με δυνατότητα αντιστροφής της ακολουθίας τους. Αρχικά, γίνεται αναλυτική περιγραφή των ειδών, των κυκλωματικών διαγραμμάτων, της λειτουργίας και των εφαρμογών των LFSR. Στη συνέχεια γίνεται κατασκευή προγραμματιζόμενων LFSR όπως και LFSR με δυνατότητα παραγωγής αντίστροφης ακολουθίας. Κατόπιν, σχεδιάζεται το κύκλωμα του αμφίδρομου προγραμματιζόμενου καταχωρητή ολίσθησης με ανάδραση και με παράλληλη φόρτωση, που περιλαμβάνει τις δύο παραπάνω δυνατότητες, και υλοποιείται, στα 16 bit, στη σχηματική σχεδίαση του ISE Xilinx Design Suite 13.1. Τέλος, σχεδιάζεται και υλοποιείται μία εφαρμογή, με χρήση ψηφιακής σχεδίασης και κώδικα της γλώσσας περιγραφής υλικού VHDL, στο αναπτυξιακό Spartan 3 Starter Kit Board της Digilent, με βάση το κύκλωμα που σχεδιάστηκε, όπου ελέγχεται η είσοδός του μέσω διακοπτών και μπουτόν (push buttons) ενώ γίνεται προβολή της εξόδου του σε τετραψήφια οθόνη επτά τμημάτων (7-segment display).

Abstract

The present work refers to the study, designing and implementation of programmable LFSRs with possibility of reversing their sequence. Initially, becomes analytic description of types, diagrams of circuits, operation and applications of LFSR. Afterwards, becomes manufacture of programmable LFSRs as well as LFSRs capable of generating reverse sequence. Then, is drawn the circuit of bidirectional programmable shift register with feedback and parallel load, that includes the above two possibilities, and is implemented, at 16 bits, in the schematic designing of ISE Xilinx Design Suite 13.1. Finally, is designed and implemented an application, with the use of digital design technics and the hardware description language VHDL, at the Spartan 3 Starter Kit Board of Digilent, based on the circuit that was designed before, where its input is controlled via switches and push buttons while its output is projected on a four digit 7-segment display.

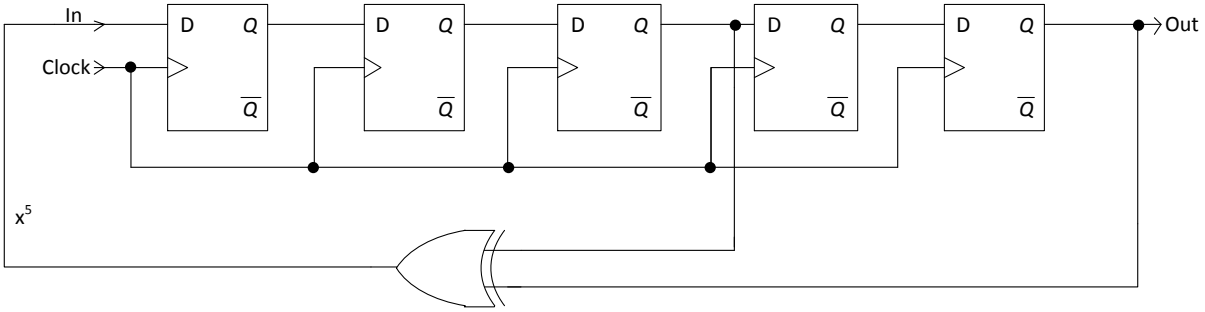
Περιεχόμενα

1. Εισαγωγή

Η παρούσα πτυχιακή εργασία αναφέρεται στη μελέτη, σχεδίαση και υλοποίηση σε FPGA, προγραμματιζόμενου LFSR που να έχει την δυνατότητα να αντιστρέφει την ακολουθία εξόδου του.

Η ιδέα για την εκπόνηση αυτής της πτυχιακής εργασίας προήλθε από συζήτηση που έγινε με τον επιβλέποντα της πτυχιακής μου εργασίας, κ. Πετράκη Νικόλαο σε συνδυασμό με την εύρεση πληροφοριών που αναφέρονται στο θέμα του προγραμματισμού των LFSR, κυρίως από μία παλαιότερη πτυχιακή εργασία ,του φοιτητή του τμήματος μας, Γεωργιάδη Μιχαήλ, με τίτλο « Μελέτη, Σχεδίαση και Κατασκευή Προγραμματιζόμενων LFSR που θα χρησιμοποιηθούν σαν Pattern Generators και Signature Analyzers».

Τα LFSR είναι ψηφιακά κυκλώματα τα οποία αποτελούνται από καταχωρητές ολίσθησης κατασκευασμένους από ακμοπυροδοτούμενα flip-flop τύπου D, κάποιες από τις εξόδους των οποίων ,μέσω λογικών πυλών XOR (ή XNOR), ανατροφοδοτούμε πίσω στην είσοδο, όπως φαίνεται και στην παρακάτω εικόνα.



Εικόνα 1.1 : Παράδειγμα καταχωρητή ολίσθησης με γραμμική ανάδραση (LFSR).

Η κύρια εφαρμογή των LFSR είναι η παραγωγή ψευδοτυχαίων ακολουθιών. Χρησιμοποιούνται στην κρυπτογραφία, στην κινητή τηλεφωνία στην επιλογή της συχνότητας, ως μετρητές ή διαιρέτες συχνότητας σε άλλα ψηφιακά κυκλώματα, στην περίπλεξη δεδομένων στις ψηφιακές επικοινωνίες, σε κυκλώματα προσομοίωσης λευκού θορύβου, στον έλεγχο δεδομένων σε σκληρούς δίσκους, σε αναλυτές υπογραφής και άλλες εφαρμογές.

Μια σύντομη περιγραφή των θεμάτων που καλύπτονται σε κάθε κεφάλαιο παρατίθεται παρακάτω:

Μετά το πρώτο κεφάλαιο της παρούσας εργασίας που αποτελεί μια σύντομη εισαγωγή, το δεύτερο κεφάλαιο αναφέρεται στο θεωρητικό μέρος των LFSR και περιέχει την περιγραφή αυτών, καθώς και τα ψηφιακά κυκλωματικά σχέδια και τις εφαρμογές αυτών. Αρχικά, το κεφάλαιο ξεκινάει με τη γενική περιγραφή του LFSR. Στη συνέχεια περιγράφονται τα είδη του LFSR, Fibonacci και Galois, με τη κυκλωματική δομή αυτών καθώς και ο διαχωρισμός τους σε πρωταρχικά και μη πρωταρχικά. Στη συνέχεια περιγράφεται αναλυτικά η λειτουργία τους και εξηγείται πως γίνεται η υλοποίηση των πολυωνύμων στα LFSR. Στο τέλος αυτού του κεφαλαίου υπάρχουν οι εφαρμογές των LFSR.

Το τρίτο κεφάλαιο της παρούσας εργασίας περιέχει τον τρόπο με τον οποίο μπορεί ένα LFSR να γίνει προγραμματιζόμενο, με χρήση πυλών AND, όπως επίσης και ο τρόπος να έχουμε αντιστροφή της ακολουθίας του LFSR. Στην ενότητα αυτή γίνεται εξαγωγή εξισώσεων μέσω χαρτών Karnaugh για την υλοποίηση του κυκλώματος που να εκτελεί αντίστροφη ακολουθία και κατασκευάζονται τα κυκλωματικά διαγράμματα. Παρουσιάζεται το κύκλωμα που μπορεί να προγραμματιστεί και να εκτελεί αντίστροφη ακολουθία και οι εξισώσεις με όρους πολυωνύμου που συνδέουν τα LFSR που

εκτελούν ορθή μέτρηση με τα LFSR που εκτελούν αντίστροφη μέτρηση. Στη συνέχεια του κεφαλαίου παρουσιάζεται ο προγραμματιζόμενος αμφίδρομος καταχωρητής ολίσθησης με παράλληλη φόρτωση και με δυνατότητα αντιστροφής της ακολουθίας του, κύκλωμα το οποίο ενώνει τα δύο κυκλώματα των LFSR ορθής - αντίστροφης μέτρησης. Στο τέλος του κεφαλαίου γίνεται σχηματική σχεδίαση του παραπάνω κυκλώματος σε μήκος 16 bit στο πρόγραμμα ISE Xilinx 13.1 και παρουσιάζεται η προσομοίωση του κυκλώματος όπου και επεξηγείται. Τονίζεται ότι η σχεδίαση του δεκαεξάμπιτου αμφίδρομου καταχωρητή ολίσθησης με παράλληλη φόρτωση και με δυνατότητα αντιστροφής της ακολουθίας του έγινε με κύριο γνώμονα τη λειτουργικότητα του και όχι τις επιδόσεις του.

Το τέταρτο κεφάλαιο της εργασίας, παρουσιάζει την εφαρμογή που υλοποίησα στο αναπτυξιακό της Digilent ,Spartan-3 Starter Kit Board. Η εφαρμογή που σχεδίασα και υλοποίησα είναι η προβολή της ακολουθίας εξόδου του 16 bit αμφίδρομου καταχωρητή ολίσθησης με παράλληλη φόρτωση και με δυνατότητα αντιστροφής της ακολουθίας του στο 7-segment display του αναπτυξιακού και ο έλεγχος αυτού μέσω διακοπτών και push buttons του αναπτυξιακού. Στην αρχή του κεφαλαίου περιγράφονται τα χαρακτηριστικά και οι δυνατότητες του αναπτυξιακού της Digilent και γίνεται μια σύντομη περιγραφή των FPGA. Στη συνέχεια παρουσιάζεται η λειτουργία του 7-segment display του αναπτυξιακού. Μετά γίνεται επεξήγηση της εφαρμογής και παρουσίαση του τρόπου λειτουργίας των διακοπτών και των push-button του αναπτυξιακού. Κατόπιν παρουσιάζεται το block διάγραμμα της σχεδίασης, το περιεχόμενο του, το οποίο περιέχει τρεις καταχωρητές 16 bit, το κύκλωμα του δεκαεξάμπιτου αμφίδρομου καταχωρητή ολίσθησης με παράλληλη φόρτωση και με δυνατότητα αντιστροφής της ακολουθίας που σχεδιάστηκε σχηματικά στο προηγούμενο κεφάλαιο, έναν πολυπλέκτη 16 σε 4, ένα μετατροπέα δεκαεξαδικού σε κώδικα για 7-segment, έναν αποκωδικοποιητή 2 σε 4, ένα δυαδικό μετρητή μήκους δύο bit, δύο διαιρέτες συχνότητας 250 Hz και 1Hz και μία μονάδα ελέγχου που συντονίζει όλο το σύστημα. Στη συνέχεια παρουσιάζεται το διάγραμμα καταστάσεων της μονάδας ελέγχου όπου και επεξηγείται και στο τέλος του κεφαλαίου παρουσιάζονται οι κώδικες που υλοποιούν τα προηγούμενα εξαρτήματα, στη γλώσσα περιγραφής υλικού VHDL, καθώς και ο κώδικας της συνολικής σχεδίασης που διασυνδέει τα παραπάνω εξαρτήματα.

Το παράρτημα Α, περιέχει την περιγραφή της διαδικασίας που ακολουθείται για τη σχηματική σχεδίαση ψηφιακών κυκλωμάτων, θέμα το οποίο δεν έχει διδαχθεί σε κάποιο μάθημα.

Το παράρτημα Β, περιέχει μία λίστα με μερικά πρωταρχικά πολυώνυμα $16^{ου}$ βαθμού τα οποία μπορούν να χρησιμοποιηθούν στο προγραμματισμό του LFSR.

Τελειώνοντας την εργασία, υπάρχει ένα τελευταίο κεφάλαιο, με τα συμπεράσματα και το πώς μπορεί η σχεδίαση να βελτιστοποιηθεί.

Ασφαλώς, στο τέλος, υπάρχει βιβλιογραφία με τα βιβλία, τα άρθρα και τους ιστότοπους που χρησιμοποιήθηκαν για την εκπόνηση της παρούσας πτυχιακής εργασίας.
Για την καταγραφή του κώδικα και της προσομοίωσης χρησιμοποιήθηκε το πρόγραμμα ISE Project Navigator 13.1 της Xilinx. Για τη συγγραφή του κειμένου της παρούσας πτυχιακής εργασίας και για τον σχεδιασμό των κυκλωμάτων και των block διαγραμμάτων, χρησιμοποιήθηκε το Microsoft Word 2010 και το Microsoft Visio 2010 αντίστοιχα.

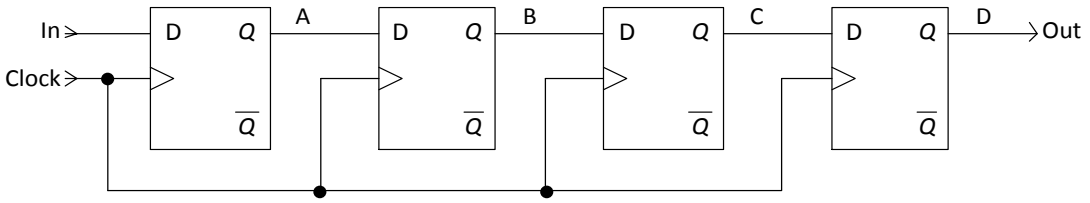
Οι κώδικες των εξαρτημάτων που σχεδιάστηκαν μαζί με τις κυματομορφές ορθής λειτουργίας τους και το κείμενο της πτυχιακής αυτής μπορούν να βρεθούν στο CD που συνοδεύει την εργασία.

2. Θεωρητικό Μέρος

Στο προηγούμενο κεφάλαιο έγινε μια εισαγωγή για το τι περιέχει αυτή η εργασία. Στο παρόν κεφάλαιο εξετάζεται το θεωρητικό μέρος των LFSR. Περιγράφονται τα είδη των LFSR, Fibonacci και Galois, με τη κυκλωματική δομή αυτών καθώς και ο διαχωρισμός τους σε πρωταρχικά και μη πρωταρχικά. Στη συνέχεια περιγράφεται αναλυτικά η λειτουργία τους και εξηγείται πως γίνεται η υλοποίηση των πολυωνύμων στα LFSR. Στο τέλος αυτού του κεφαλαίου υπάρχουν οι εφαρμογές τους.

2.1. Γενικά για το LFSR

Το LFSR είναι σύντμηση των λέξεων **Linear Feedback Shift Register** (γραμμικός καταχωρητής ολίσθησης με ανάδραση). Είναι ένας καταχωρητής ολίσθησης του οποίου τα δεδομένα εισόδου (bits εισόδου) αποτελούν γραμμική συνάρτηση των προηγούμενων καταστάσεων. Στο σχήμα 2.1 φαίνεται ένας απλός καταχωρητής ολίσθησης χωρητικότητας τεσσάρων δυαδικών ψηφίων (bits) που χρησιμοποιείται για να μετατοπίζει τα περιεχόμενά του κατά μία θέση προς τα δεξιά. Δομείται από flip-flop τύπου D η έξοδος των οποίων ακολουθεί την τιμή της εισόδου D κατά την εφαρμογή των κατάλληλων παλμών συγχρονισμού (στη μετάβαση από το ‘0’ στο ‘1’). Τα bits των δεδομένων τοποθετούνται (ή αλλιώς φορτώνονται) στον καταχωρητή με σειριακό τρόπο με τη βοήθεια της εισόδου *In*.



Εικόνα 2.1 : Καταχωρητής ολίσθησης 4 bits.

Το περιεχόμενο του κάθε flip-flop μεταφέρεται στο επόμενο σε κάθε θετικό μέτωπο του ωρολογιακού σήματος (clock). Μια απεικόνιση της διαδικασίας μεταφοράς φαίνεται στο σχήμα 2.2, όπου παρουσιάζεται τι συμβαίνει όταν οι τιμές σήματος που διαβιβάζονται στην είσοδο *In* είναι : 1, 0, 1, 1, 1, 0, 0 και 0, θεωρώντας ότι αρχικά όλα τα flip-flop είναι μηδενισμένα.

	In	*A*	*B*	*C*	*D = Out*
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

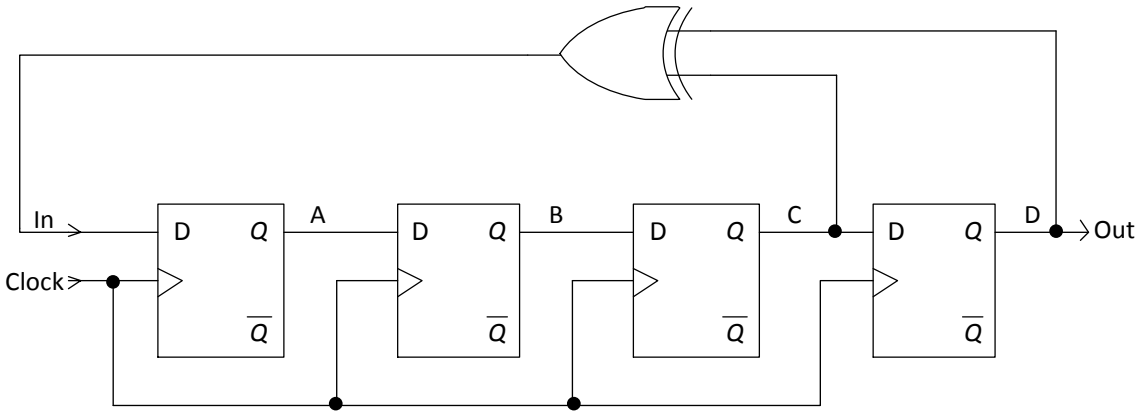
Εικόνα 2.2 : Παράδειγμα φόρτωσης και ολίσθησης.

Για να υλοποιηθεί ο καταχωρητής ολίσθησης, είναι αναγκαίο να χρησιμοποιηθούν flip-flop τύπου master-slave ή τύπου σκανδαλισμού μετώπου (ακμοπυροδοτούμενα flip-flop). Οι χρονιζόμενοι μανδαλωτές που εμφανίζουν ευαισθησία επιπέδου δεν είναι κατάλληλοι, επειδή μια αλλαγή στην τιμή της εισόδου *In* θα διαδοθεί σε περισσότερα από ένα flip-flop κατά το χρονικό διάστημα όπου ο ωρολογιακός παλμός είναι ίσος με το επίπεδο που είναι ευαίσθητα. Ο αριθμός των bits ενός καταχωρητή ολίσθησης εξαρτάται πάντα από τον αριθμό των flip-flop και άρα τον αριθμό των

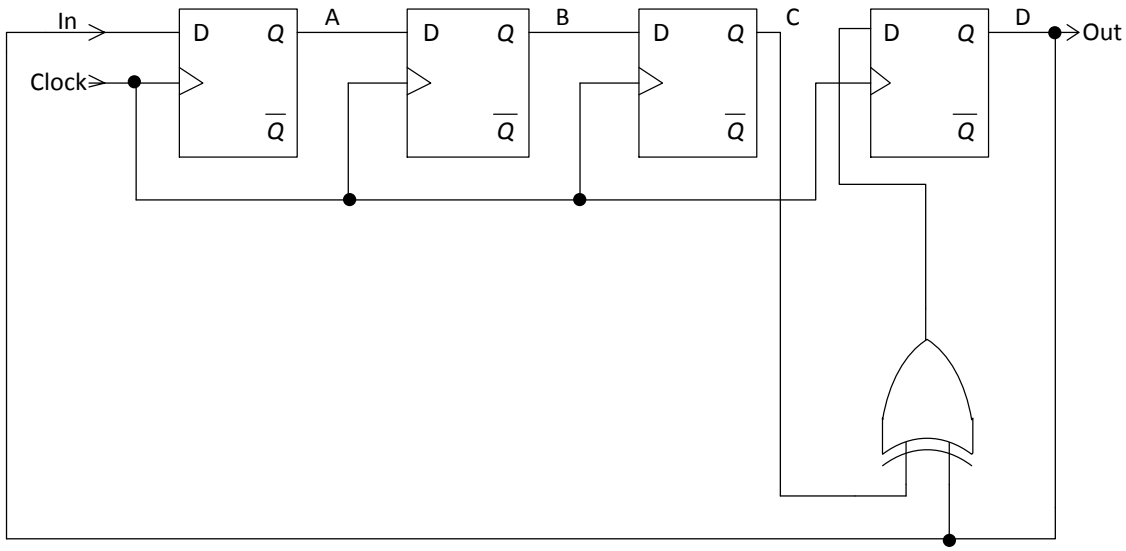
εξόδων. Όταν χρειαζόμαστε μεγαλύτερες ακολουθίες σε μήκος πρέπει να μεγαλώσουν και οι φυσικές διαστάσεις του κυκλώματος [1].
Για να ολοκληρωθεί το LFSR το μόνο που πρέπει να γίνει είναι η ανατροφοδότηση. Αυτό επιτυγχάνεται με μία πύλη XOR (ή XNOR) που στην είσοδό της συλλέγει κάποιες από τις εξόδους των flip-flop και στην έξοδό της παίρνουμε το σήμα ανατροφοδότησης το οποίο συνδέεται στην είσοδο του πρώτου flip-flop του καταχωρητή ολίσθησης.

2.2. Είδη LFSR

Υπάρχουν δύο είδη LFSR ανάλογα με το πώς συνδέεται η πύλη XOR (ή XNOR) στο κύκλωμα του καταχωρητή ολίσθησης. Η πρώτη υλοποίηση ονομάζεται **Fibonacci ή external (εξωτερικό) ή many-to-one LFSR** ενώ η δεύτερη **Galois ή internal (εσωτερικό) ή one-to-many LFSR**. Η πρώτη φαίνεται στην εικόνα 2.3 και η δεύτερη στην εικόνα 2.4 . Στην υλοποίηση Fibonacci, οι έξοδοι των flip-flop συνδέονται στις εισόδους της πύλης και η έξοδος της συνδέεται πίσω στην είσοδο. Στην υλοποίηση Galois η πύλη τοποθετείται ανάμεσα στα flip-flop.



Εικόνα 2.3 : Fibonacci LFSR.



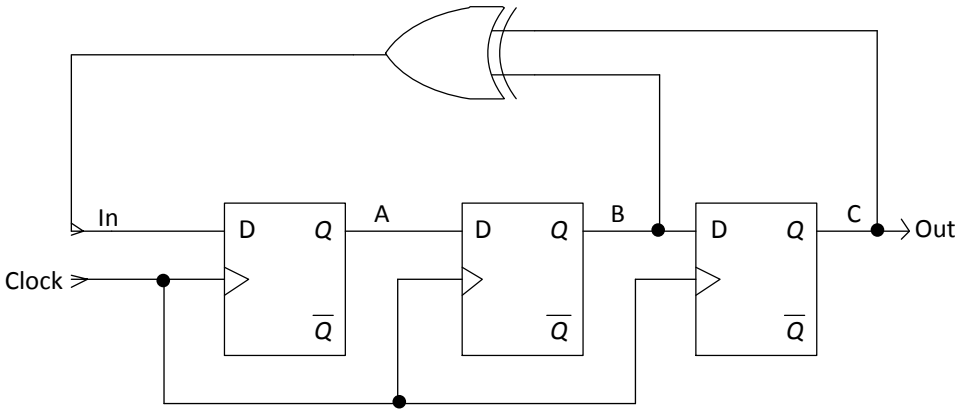
Εικόνα 2.4 : Galois LFSR.

Επίσης τα LFSR διακρίνονται σε **πρωταρχικά** (**primitive**) και σε **μη πρωταρχικά** (**non primitive**).
Πρωταρχικό (primitive) είναι ένα LFSR που στις εξόδους του παίρνουμε όλους τους δυνατούς συνδυασμούς ψηφιακών τιμών όταν του δώσουμε αρκετούς παλμούς και αυτό ξαναγυρνάει στην

πρώτη ψηφιακή λέξη που του ορίσαμε (seed value) για να ξαναπάρει πάλι όλους τους δυνατούς συνδυασμούς κ.ο.κ.. Αυτά τα LFSR ονομάζονται και **maximal** επειδή μπορούν να πάρουν όλες τις δυνατές τιμές και να παράξουν μέγιστες ακολουθίες.
Μη πρωταρχικό (non primitive) είναι ένα LFSR που στις εξόδους του δεν παίρνουμε όλους τους δυνατούς συνδυασμούς αλλά ένα μέρος αυτών ανάλογα με την αρχική τιμή (seed value) και την ανάδραση. Για παράδειγμα ένα μη πρωταρχικό 4bit LFSR που κάνει έναν πλήρη κύκλο σε 8 βήματα αντί για 15 που θα είχαμε αν ήταν πρωταρχικό. Αυτό δεν σημαίνει ότι μόνο τα primitive LFSR είναι χρήσιμα. Υπάρχουν εφαρμογές, για παράδειγμα, που αν υλοποιούσαμε ένα 64bit LFSR θα χρειαζόντουσαν ολόκληρες ώρες για να ολοκληρωθεί ένας πλήρης κύκλος λειτουργίας ενώ στη πράξη μας ενδιαφέρει ένα μέρος των ψευδοτυχαίων ακολουθιών που παράγονται. Οι ακολουθίες αυτές όμως δεν είναι ίδιες (σε σειρά) σε πρωταρχικό και σε μη πρωταρχικό LFSR[2].

2.3. Λειτουργία του LFSR

Στην παρακάτω εικόνα φαίνεται ένα LFSR 3bit Fibonacci Primitive.



Εικόνα 2.5 : 3bit Fibonacci Primitive LFSR.

Όταν ο καταχωρητής ολίσθησης είναι «φορτωμένος» με μία αρχική τιμή (οποιαδήποτε τιμή εκτός του να είναι όλα μηδέν) και ενεργοποιηθεί το ρολόι (clock) του κυκλώματος τότε η έξοδος του LFSR(C) θα είναι μια ψευδοτυχαία ακολουθία αποτελούμενη από ‘0’ και ‘1’. Στην εικόνα 2.6 παρουσιάζεται η ακολουθία που παράγεται από το κύκλωμα της εικόνας 2.5 για αρχική τιμή (seed) ‘111’.

A	B	C
1	1	1
0	1	1
0	0	1
1	0	0
0	1	0
1	0	1
1	1	0
1	1	1

Εικόνα 2.6 : Ακολουθία LFSR για το κύκλωμα της εικόνας 2.5 και αρχική τιμή ‘111’.

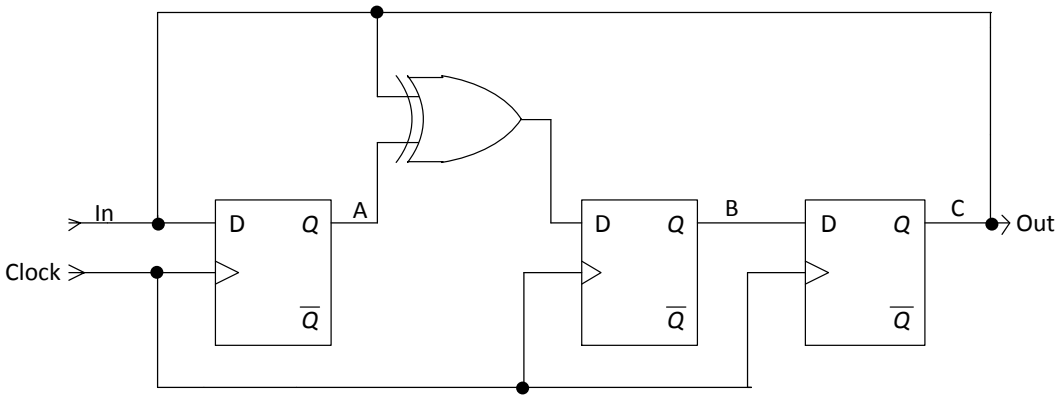
Η ψευδοτυχαία ακολουθία μπορεί να δημιουργηθεί χρησιμοποιώντας και πύλες XNOR. Σε αυτή τη περίπτωση δεν πρέπει ποτέ η αρχική τιμή να λάβει ‘111’.
Στον παραπάνω πίνακα ορίσαμε ως αρχική τιμή ‘111’. Τα δύο τελευταία ψηφία οδηγούνται στην είσοδο της πύλης XOR, οπότε σύμφωνα με τον πίνακα αληθείας της πύλης (εικόνα 2.7) θα έχουμε έξοδο ‘0’. Άρα στον επόμενο παλμό του ρολογιού (clock) η έξοδος του πρώτου Flip-Flop(A) θα γίνει ‘0’ κ.ο.κ. Αυτό θα συνεχιστεί έως ότου περάσουν 7 παλμοί ρολογιού ή $\mathbf{2^N-1}$ συνδυασμοί

(primitive LFSR), όπου N είναι ο αριθμός των Flip-Flop που υπάρχουν στο κύκλωμα. Το «-1» είναι εκεί γιατί αν βάλουμε αρχική τιμή (seed value) ‘000’ τότε το LFSR δεν θα ξεκινήσει ποτέ να μετράει γιατί η έξοδος της πύλης XOR θα είναι πάντα ‘0’ (ατέρμων βρόχος). Στην πράξη δεν βάζουμε ποτέ αρχική τιμή ‘000’ και επιθυμούμε να μην φτάσει ποτέ σ’ αυτήν την τιμή γιατί ουσιαστικά διακόπτεται η λειτουργία του (dead state). Το ίδιο ισχύει και αν κάνουμε χρήση πυλών XNOR μόνο που εδώ η αρχική τιμή δεν πρέπει ποτέ να είναι ‘111’(βλέπε πίνακα αληθείας της πύλης XNOR στην εικόνα 2.7). Αν συμβεί αυτό το LFSR δεν θα ξεκινήσει ποτέ να μετράει γιατί η έξοδος της πύλης XNOR θα είναι πάντα ‘1’[2].

Είσοδος A	Είσοδος B	Έξοδος πύλης XOR	Έξοδος πύλης XNOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Εικόνα 2.7 : Πίνακας αληθείας των λογικών πυλών XOR και XNOR.

Τα ίδια ισχύουν και για τα Galois LFSR. Στην παρακάτω εικόνα φαίνεται ένα LFSR 3bit Galois Primitive.



Εικόνα 2.8 : 3bit Galois Primitive LFSR.

Στην εικόνα 2.9 παρουσιάζεται η ακολουθία που παράγεται από το κύκλωμα της εικόνας 2.8 για αρχική τιμή (seed) ‘111’.

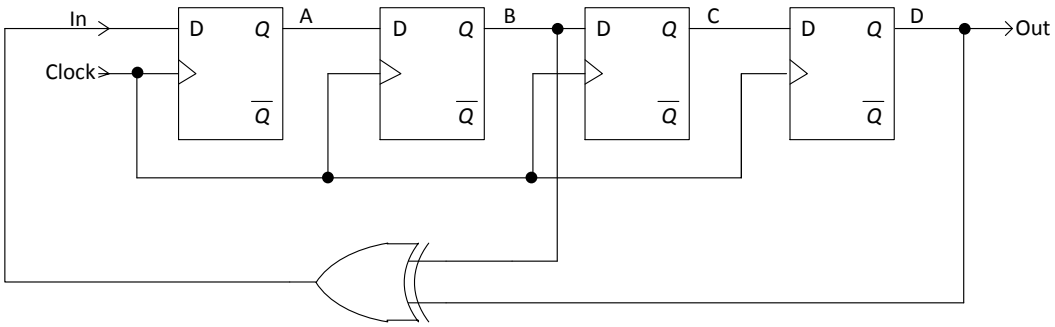
A	B	C
1	1	1
1	0	1
1	0	0
0	1	0
0	0	1
1	1	0
0	1	1
1	1	1

Εικόνα 2.9 : Ακολουθία LFSR για το κύκλωμα της εικόνας 2.8 και αρχική τιμή ‘111’.

Στον παραπάνω πίνακα ορίσαμε ως αρχική τιμή ‘111’. Το τελευταίο και το πρώτο ψηφίο οδηγείται στην είσοδο της πύλης XOR αφού η πύλη τοποθετείται ανάμεσα στα δύο flip-flop. Από τους δύο

πίνακες των εικόνων 2.6 και 2.9 μπορούμε να δούμε ότι η ακολουθία εξόδου (C) του Galois LFSR είναι ίδια με του Fibonacci LFSR αν μετατοπίσουμε το ρολόι κατά 1 παλμό [8].

Ένα μη πρωταρχικό LFSR φαίνεται στην εικόνα 2.10



Εικόνα 2.10 : 4bit Fibonacci Non Primitive LFSR.

Στην εικόνα 2.11 παρουσιάζεται η ακολουθία που παράγεται από το κύκλωμα της εικόνας 2.10 για αρχική τιμή (seed) '1111'.

A	B	C	D
1	1	1	1
0	1	1	1
0	0	1	1
1	0	0	1
1	1	0	0
1	1	1	0
1	1	1	1

Εικόνα 2.11 : Ακολουθία LFSR για το κύκλωμα της εικόνας 2.10 και αρχική τιμή '1111'.

Στον παραπάνω πίνακα ορίσαμε ως αρχική τιμή την '1111'. Οι δύο έξοδοι, B και D, οδηγούνται στην είσοδο της πύλης XOR, οπότε σύμφωνα με τον πίνακα αληθείας της πύλης (εικόνα 2.7) θα έχουμε έξοδο '0'. Άρα στον επόμενο παλμό του ρολογιού (clock) η έξοδος του πρώτου flip-flop (A) θα γίνει '0' κ.ο.κ. Μετά όμως από 6 παλμούς ρολογιού παρατηρούμε ότι έχει επανέλθει η αρχική κατάσταση '1111' ενώ αν είχαμε πρωταρχικό LFSR θα χρειαζόντουσαν $\mathbf{2^N-1}$ παλμοί, όπου N είναι ο αριθμός των Flip-Flop που υπάρχουν στο κύκλωμα, άρα 15 παλμούς. Επίσης θα είχαμε και όλες τις δυνατές ψηφιακές τιμές, δηλαδή 0001 έως και 1111 ενώ τώρα έχουμε μόνο τις τιμές 0111, 0011, 1001, 1100, 1110 και 1111.

Το μήκος της ψευδοτυχαίας ακολουθίας εξαρτάται από το μήκος του καταχωρητή ολίσθησης (αριθμός flip-flop) ενώ οι δυαδικές τιμές που περιλαμβάνει καθώς και η σειρά τους εξαρτώνται από τον αριθμό και την θέση των αναδράσεων. Ο αριθμός και η θέση των αναδράσεων αντιπροσωπεύονται από ένα πολυώνυμο. Στα LFSR των εικόνων 2.5 και 2.8 το πολυώνυμο είναι το $x^3 + x + 1$ ενώ στην εικόνα 2.10 το πολυώνυμο είναι το $x^4 + x^2 + 1$. Περισσότερα σχετικά με τα πολυώνυμα παρουσιάζονται στην ενότητα 2.3.
Από εδώ και κάτω θα αναφερόμαστε μόνο σε πύλες XOR αφού τα LFSR με πύλες XNOR λειτουργούν με παρόμοιο τρόπο.

2.3. Υλοποίηση Πολυωνύμων

Με τα LFSR μπορούμε να δημιουργήσουμε πολλά διαφορετικά πολυώνυμα. Ο περιοριστικός παράγων είναι ότι θα πρέπει να χρησιμοποιήσουμε τόσα flip flop όσο είναι και ο βαθμός του πολυωνύμου που θέλουμε να υλοποιήσουμε. Όπως είπαμε και προηγουμένως υπάρχουν δύο είδη

πολυωνύμων. Τα πρωταρχικά (primitive) και τα μη πρωταρχικά (non primitive). Με τα πρώτα μπορούμε να παράξουμε μέγιστες ακολουθίες (όσο ορίζει ο εκάστοτε βαθμός του πολυωνύμου) ενώ με τα δεύτερα μπορούμε να πάρουμε ένα μέρος της ακολουθίας ανάλογα το πολυώνυμο που χρησιμοποιούμε και όχι σε σωστή σειρά. **Μαθηματικά, τα πρωταρχικά πολυώνυμα βαθμού n είναι αυτά τα οποία διαιρούνται μόνο με τον εαυτό τους και διαιρούν το πολυώνυμο $x^k + 1$ με $k = 2^n - 1$ αλλά όχι με $k < 2^n - 1$** [3]. Στον πίνακα 2.12 παρουσιάζονται κάποια από τα πρωταρχικά πολυώνυμα έως και 73ου βαθμού[4].

Βαθμός (n)	Πολυώνυμο	Βαθμός (n)	Πολυώνυμο
2, 3, 4, 6, 7, 15, 22, 60, 63	$x^n + x + 1$	12	$x^n + x^7 + x^4 + x^3 + 1$
5, 11, 21, 29, 35	$x^n + x^2 + 1$	33	$x^n + x^{13} + 1$
8, 19, 38, 43	$x^n + x^6 + x^5 + x + 1$	34	$x^n + x^{15} + x^{14} + x + 1$
9, 39	$x^n + x^4 + 1$	36	$x^n + x^{11} + 1$
10, 17, 20, 25, 28, 31, 41, 52	$x^n + x^3 + 1$	37	$x^n + x^{12} + x^{10} + x^2 + 1$
13, 24, 45, 64	$x^n + x^4 + x^3 + x + 1$	40	$x^n + x^{21} + x^{19} + x^2 + 1$
14, 16	$x^n + x^5 + x^4 + x^3 + 1$	42	$x^n + x^{23} + x^{22} + x + 1$
18, 57	$x^n + x^7 + 1$	46	$x^n + x^{21} + x^{20} + x + 1$
23, 47	$x^n + x^5 + 1$	54	$x^n + x^{37} + x^{36} + x + 1$
26, 27	$x^n + x^{12} + x^{11} + x + 1$	55	$x^n + x^{24} + 1$
30, 51, 53, 61, 70	$x^n + x^{16} + x^{15} + x + 1$	58	$x^n + x^{19} + 1$
32, 48	$x^n + x^{28} + x^{27} + x + 1$	65	$x^n + x^{18} + 1$
44, 50	$x^n + x^{27} + x^{26} + x + 1$	69	$x^n + x^{29} + x^{27} + x^2 + 1$
49, 68	$x^n + x^9 + 1$	71	$x^n + x^6 + 1$
56, 59	$x^n + x^{22} + x^{21} + x + 1$	72	$x^n + x^{53} + x^{47} + x^6 + 1$
66, 67, 74	$x^n + x^{10} + x^9 + x + 1$	73	$x^n + x^{25} + 1$

Εικόνα 2.12 : Πρωταρχικά Πολυώνυμα έως και 73ου βαθμού.

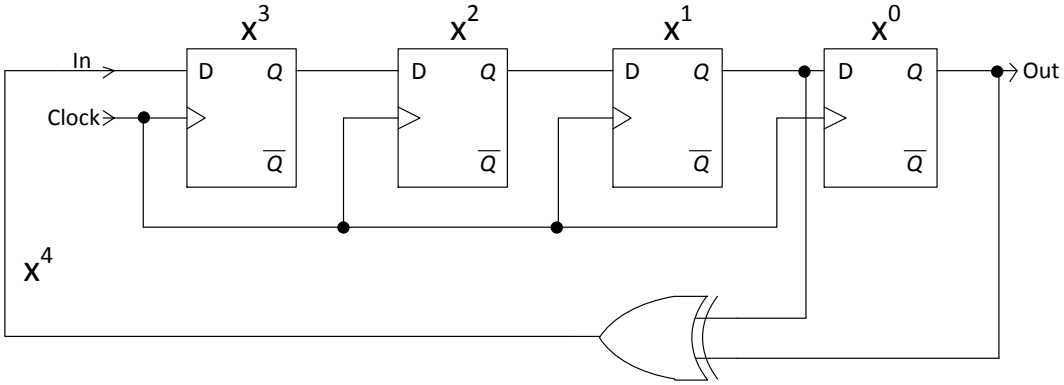
Υπάρχουν βέβαια πολυώνυμα μεγαλύτερα του 73ου βαθμού αλλά δεν κρίθηκε αναγκαίο να αναφερθούν για τους σκοπούς αυτής της πτυχιακής. Στον πίνακα 2.13 παρουσιάζονται και τα μήκη των ακολουθιών για πρωταρχικά πολυώνυμα 2ου έως και 32ου βαθμού[10].

Βαθμός πολυωνύμου	Μήκος της ακολουθίας	Βαθμός πολυωνύμου	Μήκος της ακολουθίας
2	3	18	262.143
3	7	19	524.287
4	15	20	1.048.575
5	31	21	2.097.151
6	63	22	4.194.303
7	127	23	8.388.607
8	255	24	16.777.215
9	511	25	33.554.431
10	1.023	26	67.108.863
11	2.047	27	134.217.727
12	4.095	28	268.435.455
13	8.191	29	536.870.911
14	16.383	30	1.073.741.823
15	32.767	31	2.147.483.647
16	65.535	32	4.294.967.295
17	131.071		

Εικόνα 2.13 : Μήκη ακολουθιών ανάλογα με το βαθμό του πολυωνύμου.

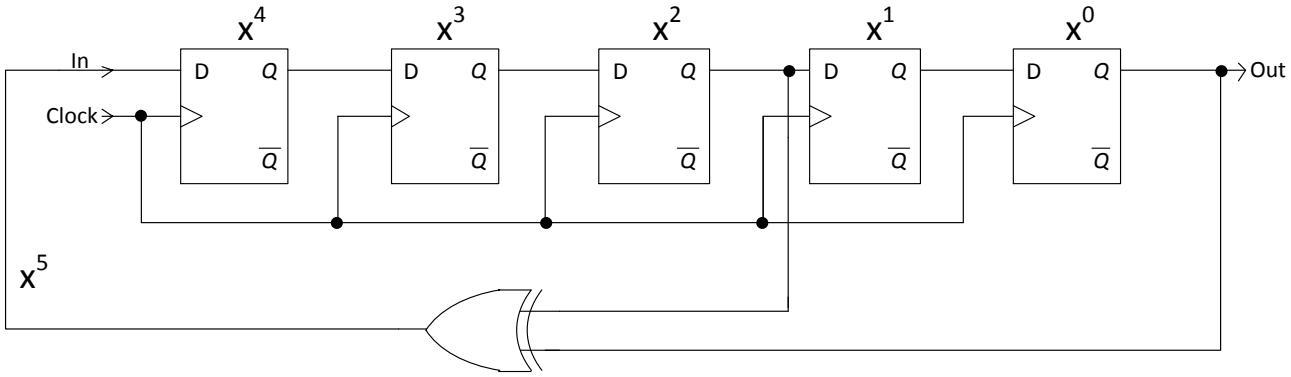
Όπως έχει προαναφερθεί υπάρχουν δύο τύποι LFSR. Ο πρώτος τύπος με τις πύλες ανατροφοδότησης XOR εξωτερικά που ονομάζεται και Fibonacci και ο δεύτερος με τις πύλες ανατροφοδότησης XOR ανάμεσα στα flip flop που ονομάζεται και Galois.

Παρακάτω θα δούμε τον τρόπο που υλοποιούνται τα παραπάνω πολυώνυμα. Πρώτα θα εξεταστεί το Fibonacci(εξωτερικό) LFSR που είναι ποιο συνηθισμένο. Δίνεται το ακόλουθο κύκλωμα:

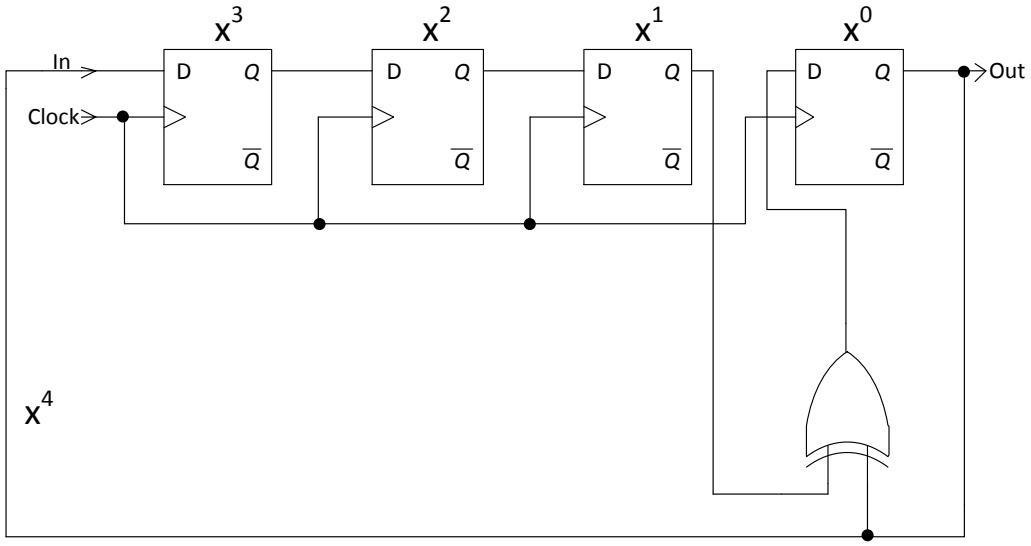


Εικόνα 2. 14 : Fibonacci Primitive LFSR 4ου βαθμού

Το παραπάνω κύκλωμα είναι ένα εξωτερικό πρωταρχικό LFSR με πολυώνυμο $x^4 + x + 1$. Σκοπός είναι να συνδέσουμε στην έξοδο των flip-flop που δείχνει το πολυώνυμο μια πύλη XOR που θα ανατροφοδοτήσει το κύκλωμα. Παρατηρούμε ότι η αρίθμηση των flip flop γίνεται στην σειρά. Από δεξιά προς αριστερά έχουμε το x^0 ή 1 που υπάρχει πάντα, διαφορετικά είναι περιττό να έχουμε συνδεδεμένο το τελευταίο flip flop. Ακολουθεί το x^1 ή x, το x^2 και το x^3. Το x^4 υποδηλώνει την ανατροφοδότηση. Ανάλογα με το μήκος του LFSR (αριθμός των flip flop) μετράμε αθροίζοντας κατά ένα τους εκθέτες του x μέχρι το τέλος του. Για παράδειγμα δίνεται το παρακάτω primitive LFSR 5ου βαθμού με πολυώνυμο $x^5 + x^2 + 1$ που διαφέρει από το προηγούμενο 4ου βαθμού.

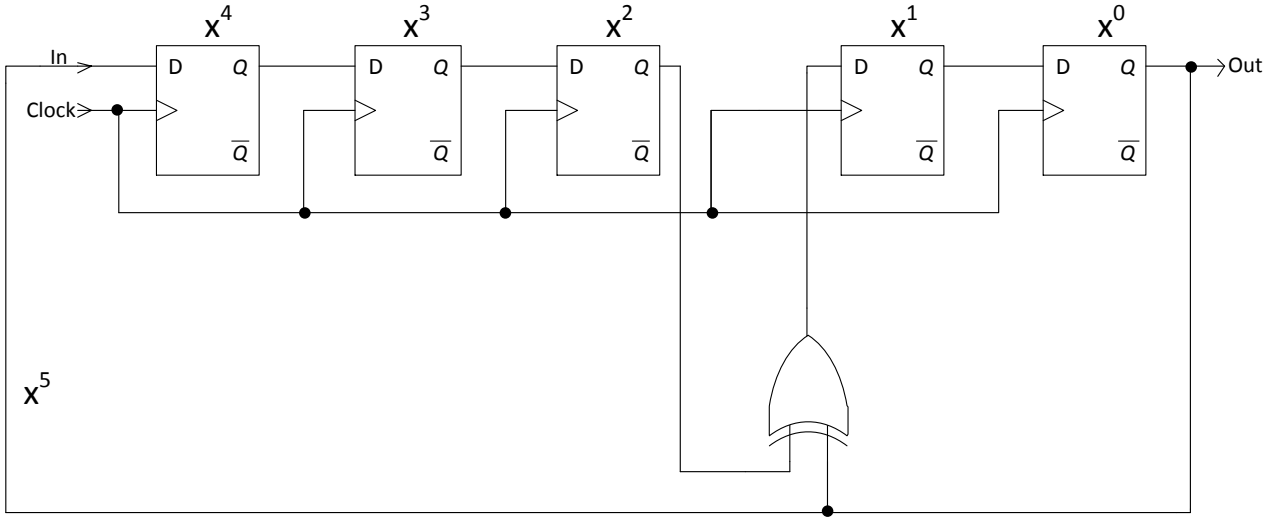


Εικόνα 2.15 : Fibonacci Primitive LFSR 5ου βαθμού.



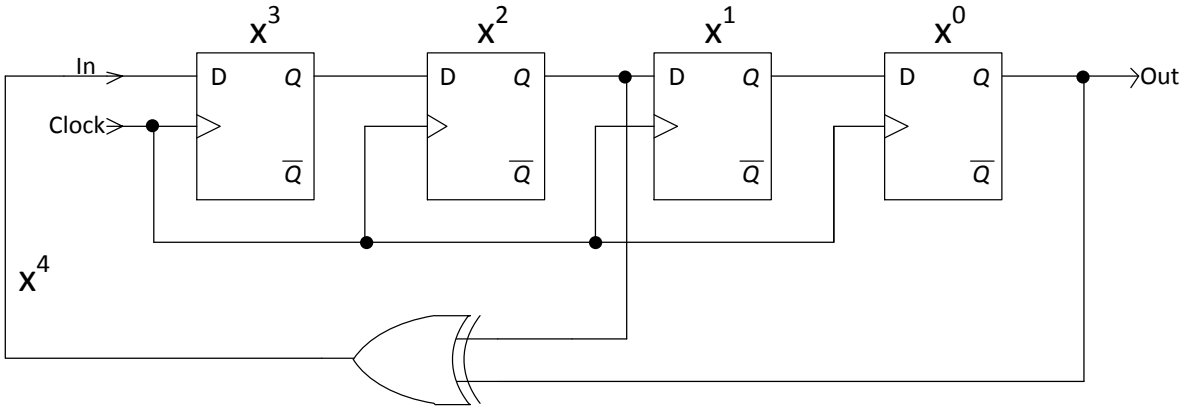
Εικόνα 2.16 : Galois Primitive LFSR 4ου βαθμού.

Τα ίδια ισχύουν και για Galois (εσωτερικά) LFSR. Η μόνη διαφορά είναι ότι οι πύλες XOR τοποθετούνται ανάμεσα στα flip flop του κυκλώματος. Στις εικόνες 2.16 και 2.17 δίδονται τα αντίστοιχα εσωτερικά LFSR για τις παραπάνω περιπτώσεις.



Εικόνα 2.17 : Galois Primitive LFSR 5ου βαθμού.

Όπως γίνεται κατανοητό δεν παίζει κανένα ρόλο αν το LFSR θα είναι πρωταρχικό ή μη πρωταρχικό. Το παρακάτω κύκλωμα είναι μη πρωταρχικό Fibonacci LFSR 4ου βαθμού με πολυώνυμο $x^4 + x^2 + 1$ (ισχύουν παρόμοια για Galois LFSR). Όπως βλέπουμε ακολουθεί τον ίδιο τρόπο για να μετατραπεί από πολυώνυμο σε κύκλωμα με λογικές πύλες και flip flop[9].



Εικόνα 2.18 : Fibonacci No Primitive LFSR 4ου βαθμού.

2.4. Εφαρμογές των LFSR

Οι εφαρμογές των LFSR ποικίλουν. Οι χρήσεις τους είναι πολλές αλλά πολύ λίγες είναι ευρέως γνωστές. Τα LFSR συνήθως αποτελούν υποσύστημα σε κάποια περιφερειακή μονάδα. Οι εφαρμογές τους είναι οι εξής [11]:

- ***Ως διαιρέτης ρολογιού ή ως μετρητής***. Η επαναλαμβανόμενη ακολουθία που παράγει ένα LFSR μπορεί να χρησιμοποιηθεί ως διαιρέτης ρολογιού για να παραχθούν μικρότερες συχνότητες ή ως μετρητής όταν μία μη δυαδική ακολουθία είναι αποδεκτή από το σύστημα. Ωστόσο είναι αναγκαίο να εξασφαλιστεί ότι ποτέ το LFSR δεν θα αποκτήσει την τιμή ‘0000…0’ στην περίπτωση της πύλης XOR και την τιμή ‘1111…1’ στην περίπτωση της πύλης XNOR.

- ***Στην κρυπτογραφία***. Χρησιμοποιούνται εδώ και καιρό ως γεννήτριες ψευδοτυχαίων αριθμών για χρήση σε αλγόριθμους κρυπτογράφησης (ειδικά σε στρατιωτική κρυπτογραφία). Αυτό οφείλεται στην ευκολία κατασκευής τους από ηλεκτρομηχανικά ή ηλεκτρονικά κυκλώματα, στις μεγάλες περιόδους που παρέχουν και στις ομοιόμορφα κατανεμημένες ακολουθίες εξόδου. Ωστόσο, επειδή το LFSR είναι γραμμικό σύστημα, οδηγεί σε εύκολο τρόπο κρυπτανάλυσης. Για παράδειγμα, εάν δοθούν κάποια από τα χαρακτηριστικά του, μπορεί ένας εισβολέας να παρακολουθήσει και να ανακτήσει τμήμα της ακολουθίας βασιζόμενος στα χαρακτηριστικά που έχει. Από αυτό το τμήμα μπορεί να ανακτήσει όλο το υπόλοιπο. Υπάρχουν τρεις γενικοί μέθοδοι που μειώνουν αυτό το πρόβλημα:

    1. Μη γραμμικός συνδυασμός από διάφορα bits του LFSR.
    2. Μη γραμμικός συνδυασμός από διάφορα bits δύο ή περισσότερων LFSR.
    3. Αντικανονικό χρονισμό του LFSR, όπως στη γεννήτρια εναλλασσομένου βήματος.

- ***Στην περίπλεξη δεδομένων***. Χρησιμοποιούνται στις ψηφιακές επικοινωνίες στην διαδικασία της περίπλεξης (scrambling). Για να αποφευχθεί η επανάληψη σύντομων ακολουθιών (σειρές από ‘0’ και ‘1’) από την διαμόρφωση των φασματικών γραμμών που μπορεί να περιπλέξουν την παρακολούθηση των συμβόλων στο δέκτη ή να παρεμποδίσουν άλλες μεταδόσεις, τα LFSR συχνά χρησιμοποιούνται για να δημιουργήσουν «τυχαίες» σειρές του μεταδιδόμενου bitstream. Η τυχαιοποίηση αυτή αφαιρείται στον δέκτη κατά τη διαδικασία της αποδιαμόρφωσης.
- ***Στην προσέγγιση του λευκού θορύβου***. Χρησιμοποιούνται για την προσέγγιση του λευκού θορύβου σε προγραμματιζόμενες γεννήτριες ήχου ή σε συστήματα επικοινωνιών για να αυξήσουν το επίπεδο του θορύβου.
- ***Στην κινητή τηλεφωνία***. Χρησιμοποιούνται στο σύστημα κινητής τηλεφωνίας CDMA (Code Division Multiple Access). Με τα LFSR παράγονται ακολουθίες μέσω των οποίων επιλέγεται ποια χρονοθυρίδα θα καταλάβει ή σε ποια συχνότητα θα λειτουργήσει το κινητό τηλέφωνο για να επικοινωνήσει με κάποιον άλλο χρήστη.

Άλλα συστήματα που κάνουν χρήση των LFSR είναι τα ακόλουθα:

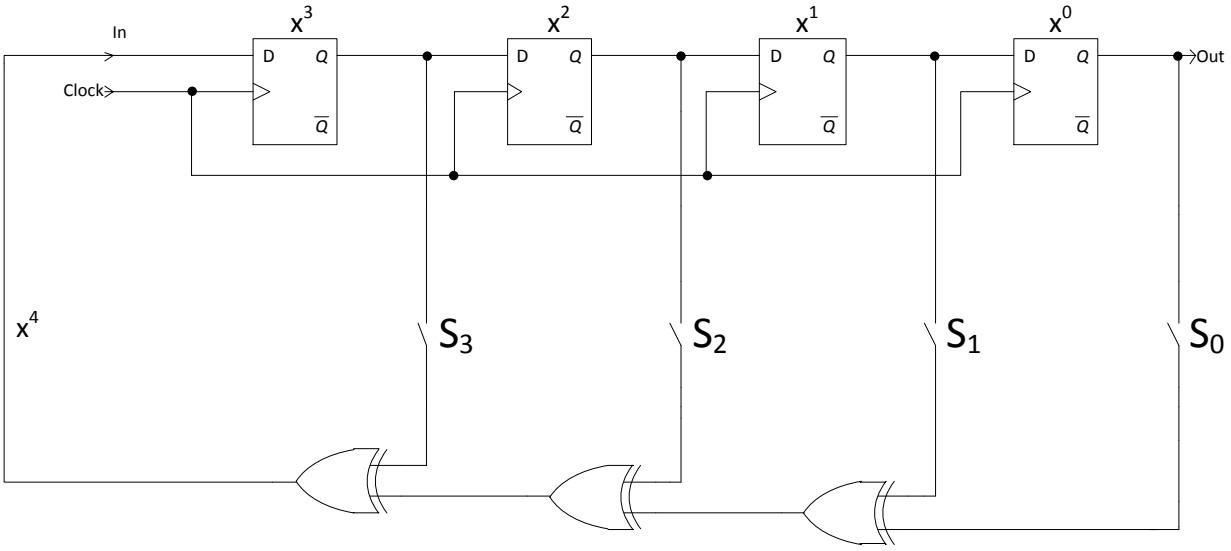
- ATSC Standards (digital TV transmission system – North America)
- DAB (Digital Audio Broadcasting system – for radio)
- DVB-T (digital TV transmission system – Europe, Australia, parts of Asia)
- NICAM (digital audio system for television)
- IBS (INTELSAT business service)
- IDR (Intermediate Data Rate service)
- SDI (Serial Digital Interface transmission)
- Data transfer over PSTN (according to the ITU-T V-series recommendations)
- 100BASE-T2 "fast" Ethernet
- 1000BASE-T Ethernet
- PCI Express 3.0
- SATA
- USB 3.0
- IEEE 802.11a (WiFi)

3. Σχεδίαση Προτεινόμενου LFSR

Στο προηγούμενο κεφάλαιο αναλύθηκε το θεωρητικό μέρος των LFSR. Περιγράφηκε από ποια εξαρτήματα απαρτίζονται, ποιες είναι οι μορφές τους (Fibonacci και Galois), πως γίνεται η επιλογή και η υλοποίηση των πολυωνύμων καθώς και εφαρμογές στις οποίες μετέχουν. Στο παρόν κεφάλαιο περιγράφεται ο τρόπος με τον οποίο μπορεί ένα LFSR να γίνει προγραμματιζόμενο όπως επίσης και ο τρόπος να έχουμε αντιστροφή της ακολουθίας της εξόδου του. Στο τέλος του κεφαλαίου σχεδιάζεται ένα κύκλωμα μέσω του οποίου υλοποιούνται τα παραπάνω.

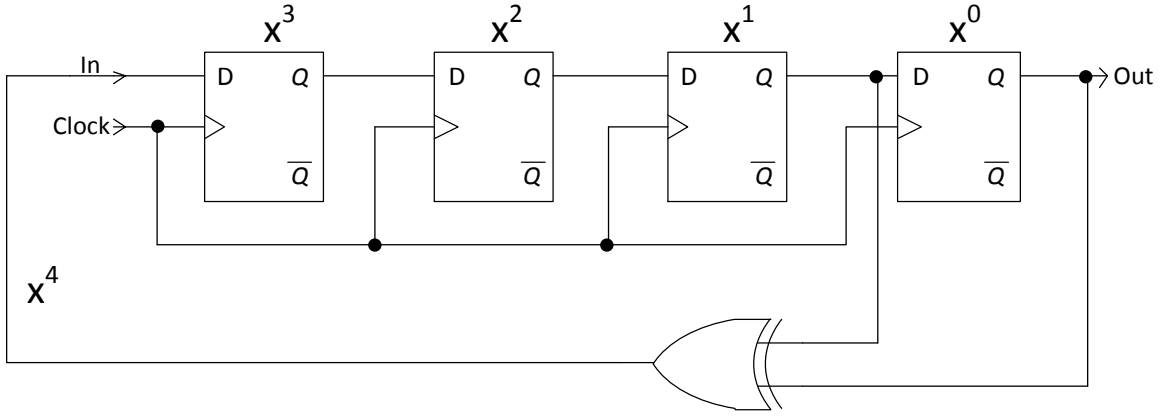
3.1. Προγραμματισμός του LFSR

Ο προγραμματισμός του LFSR αναφέρεται στην επιλογή του πολυωνύμου. Δηλαδή, με κάποιον τρόπο, να μπορούμε να επιλέξουμε ποιες από τις εξόδους των flip-flop θα οδηγούνται στην είσοδο της πύλης XOR. Ουσιαστικά αυτό που προσπαθούμε να πετύχουμε είναι αυτό που φαίνεται στην εικόνα 3.1 μόνο που αντί για μηχανικούς διακόπτες να χρησιμοποιήσουμε ηλεκτρονικούς διακόπτες.



Εικόνα 3.1 : Επιλογή πολυωνύμου μέσω των διακοπτών S_0-S_3.

Αν για παράδειγμα κλείσουν (κατάσταση ON) οι διακόπτες S_0 και S_1 τότε θα έχουμε το πρωταρχικό πολυώνυμο λειτουργίας $x^4 + x + 1$ όπως είδαμε στο προηγούμενο κεφάλαιο. Δηλαδή το ισοδύναμο κύκλωμα θα είναι:



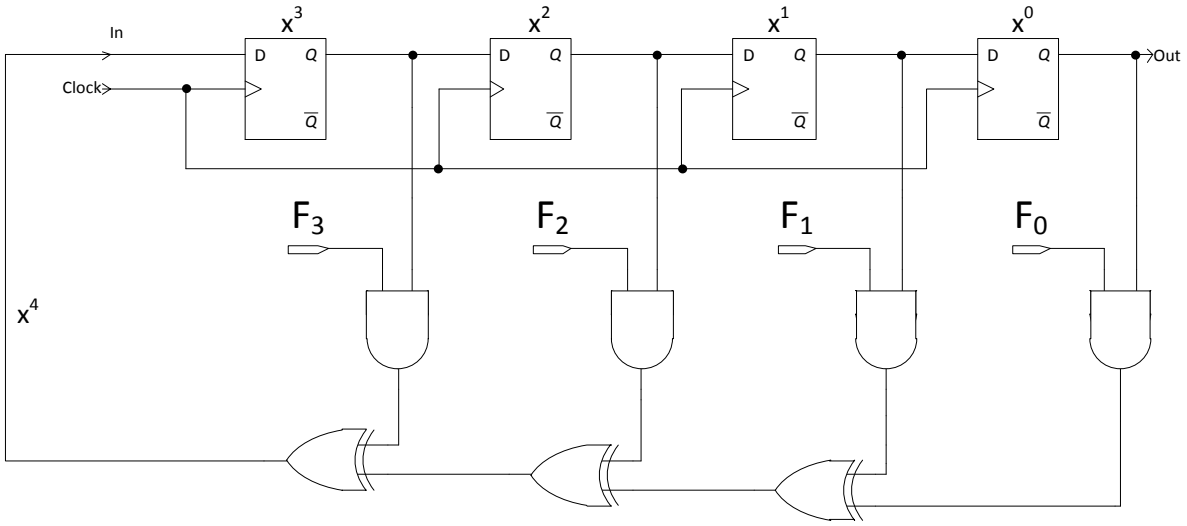
Εικόνα 3.2 : Ισοδύναμο κύκλωμα της εικόνας 3.1 με κλειστούς τους διακόπτες S_0 και S_1.

Οι άλλοι διακόπτες που είναι ανοιχτοί (σε κατάσταση OFF), δεν επηρεάζουν το συνολικό κύκλωμα γιατί η είσοδος της πύλης XOR είναι λογικό ‘0’. Όπως έχουμε δει στο κεφάλαιο 2 από τον πίνακα αληθείας της πύλης XOR (εικόνα 2.7) όταν μια είσοδος της είναι ‘0’ τότε είναι σαν να μην υπάρχει η πύλη γιατί ότι έρθει στην άλλη είσοδο περνάει και στην έξοδο.
Αντί για μηχανικούς διακόπτες χρησιμοποιούμε λογικές πύλες AND. Για βοήθεια δίδεται ο πίνακας αληθείας της πύλης AND στην εικόνα 3.3.

Είσοδος A	Είσοδος B	Έξοδος πύλης AND
0	0	0
0	1	0
1	0	0
1	1	1

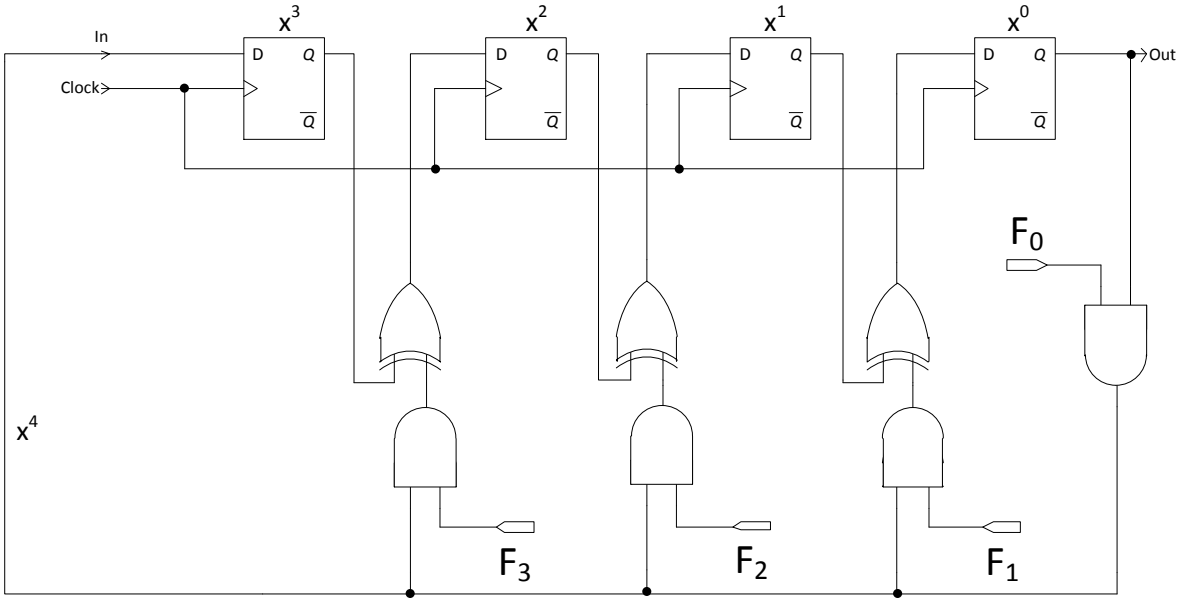
Εικόνα 3.3 : Πίνακας αληθείας της λογικής πύλης AND.

Οπότε το κύκλωμα της εικόνας 3.1 με τους διακόπτες μετατρέπεται στο παρακάτω κύκλωμα της εικόνας 3.4 με τις πύλες AND στη θέση των διακοπτών.



Εικόνα 3.4 : Επιλογή πολυωνύμου μέσω των λογικών πυλών AND σε Fibonacci LFSR.

Για παράδειγμα, αν στους ακροδέκτες F_0 και F_1 δώσουμε λογικό ‘1’ ενώ στους υπόλοιπους (F_2 και F_3) δώσουμε λογικό ‘0’ θα έχουμε ως αποτέλεσμα το ισοδύναμο κύκλωμα της εικόνας 3.2 με πολυώνυμο λειτουργίας $x^4 + x + 1$. Αυτό συμβαίνει γιατί αν κοιτάξουμε τον πίνακα αληθείας της πύλης AND, που βρίσκεται στην εικόνα 3.3, όταν δώσουμε λογικό ‘1’ στην μία της είσοδο τότε στην έξοδο της θα έχουμε ότι υπάρχει στην δεύτερη είσοδο. Ουσιαστικά είναι σαν να μην υπάρχει όπως βλέπουμε στο ισοδύναμο κύκλωμα. Ενώ όταν δώσουμε λογικό ‘0’ σε οποιαδήποτε είσοδο της, είναι σαν να την απενεργοποιούμε και στην έξοδο της έχουμε λογικό ‘0’. Η τελευταία πύλη AND με την μία είσοδο στον ακροδέκτη F_0 μπορεί να παραληφθεί καθώς είναι απαραίτητη η ανατροφοδότηση από την έξοδο του τελευταίου flip-flop (x^0).
Παρόμοια ισχύουν και για Galois (εσωτερικά) LFSR όπως φαίνεται και στην εικόνα 3.5 καθώς δεν υπάρχει καμία διαφορά με τα Fibonacci (εξωτερικά) LFSR παρά μόνο στο τρόπο που αυτά συνδέονται. Τα εξωτερικά LFSR χρησιμοποιούνται πιο συχνά επειδή οι συνδέσεις είναι πιο εύκολα ορατές απ’ ότι στα εσωτερικά.

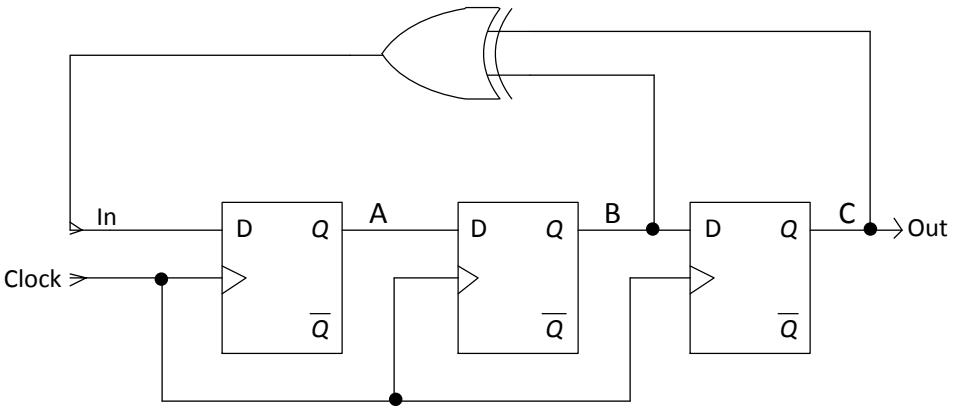


Εικόνα 3.5 : Επιλογή πολυωνύμου μέσω των λογικών πυλών AND σε Galois LFSR.

Παρόμοια, η τελευταία πύλη AND με την μία είσοδο στον ακροδέκτη F_0 μπορεί να παραληφθεί καθώς είναι απαραίτητη η ανατροφοδότηση από την έξοδο του τελευταίου flip-flop (x^0).
Συνοψίζοντας, μπορούμε να υλοποιήσουμε οποιοδήποτε πολυώνυμο (ανάλογα το βαθμό και τον αριθμό των flip-flop του κυκλώματος) εφαρμόζοντας λογικό '1' στις αντίστοιχες πύλες AND, ενεργοποιώντας δηλαδή την αντίστοιχη ανατροφοδότηση που οδηγείται στην είσοδο της πύλης XOR.

3.2. Αντιστροφή ακολουθίας του LFSR

Στην ενότητα αυτή παρουσιάζουμε το πως μπορούμε να κάνουμε το LFSR να μετράει και αντίστροφα. Έστω το πρωταρχικό 3^{ου} βαθμού Fibonacci LFSR το οποίο βρίσκεται στην εικόνα 3.6. Το LFSR αυτό έχει πολυώνυμο λειτουργίας $x^3 + x + 1$. Η ακολουθία του LFSR φαίνεται στην εικόνα 3.7 όπου στην παρούσα κατάσταση έχουμε τις δυαδικές τιμές σε σειρά ('001'-'111'), εκτός τη τιμή '000' γιατί οδηγεί σε διακοπή λειτουργίας του κυκλώματος (dead state) και στην επόμενη κατάσταση έχουμε τις ανάλογες δυαδικές τιμές, που προέρχονται από τις εξόδους των flip-flop, για κάθε τιμή της παρούσας κατάστασης.



Εικόνα 3.6 : Fibonacci πρωταρχικό LFSR 3^{ου} βαθμού.

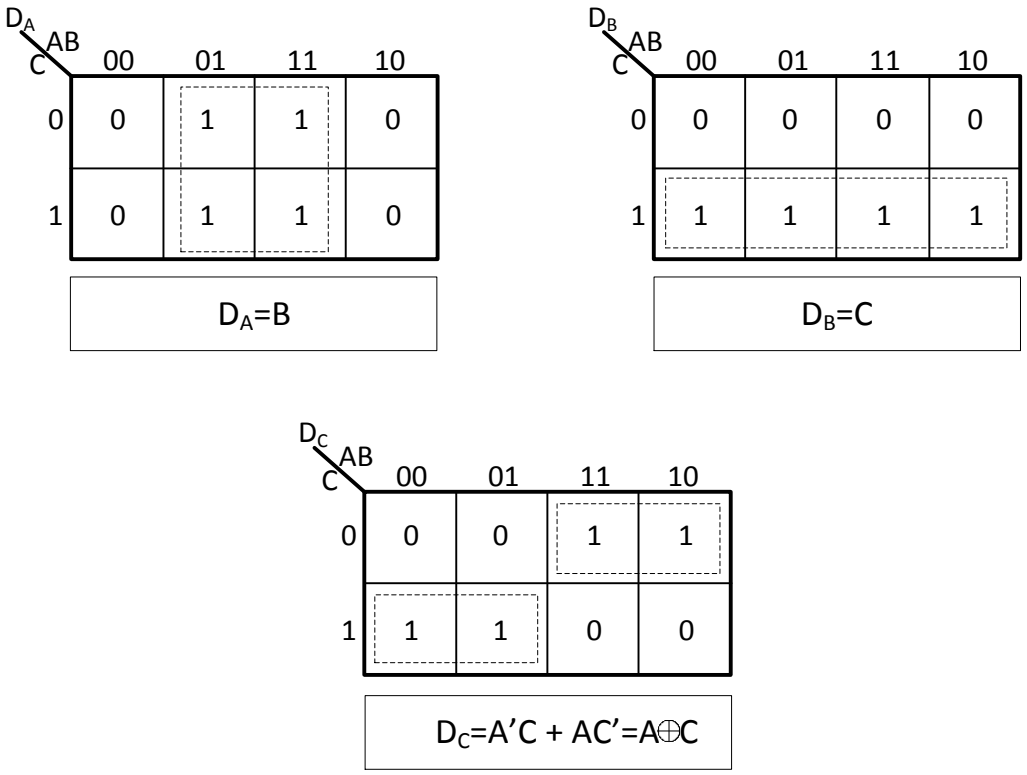
Παρούσα Κατάσταση			Επόμενη Κατάσταση		
A	*B*	*C*	*A*	*B*	*C*
0	0	1	1	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	0	1	0
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	1	1

Εικόνα 3.7 : Πίνακας ακολουθίας του κυκλώματος της εικόνας 3.6.

Τα A, B και C είναι οι έξοδοι του πρώτου, δευτέρου και αντίστοιχα τρίτου flip-flop. Εμείς επιθυμούμε να επιστρέψουμε από την επόμενη κατάσταση πίσω στην παρούσα κατάσταση. Δηλαδή από την τιμή ‘100’ να επιστρέψουμε στην ‘001’, από την τιμή ‘101’ να επιστρέψουμε στην ‘010’ κ.ο.κ.. Οπότε δημιουργούμε έναν νέο πίνακα, όπως φαίνεται στην εικόνα 3.8, όπου τώρα η επόμενη κατάσταση αποτελεί παρούσα κατάσταση και η παρούσα κατάσταση αποτελεί επόμενη κατάσταση.

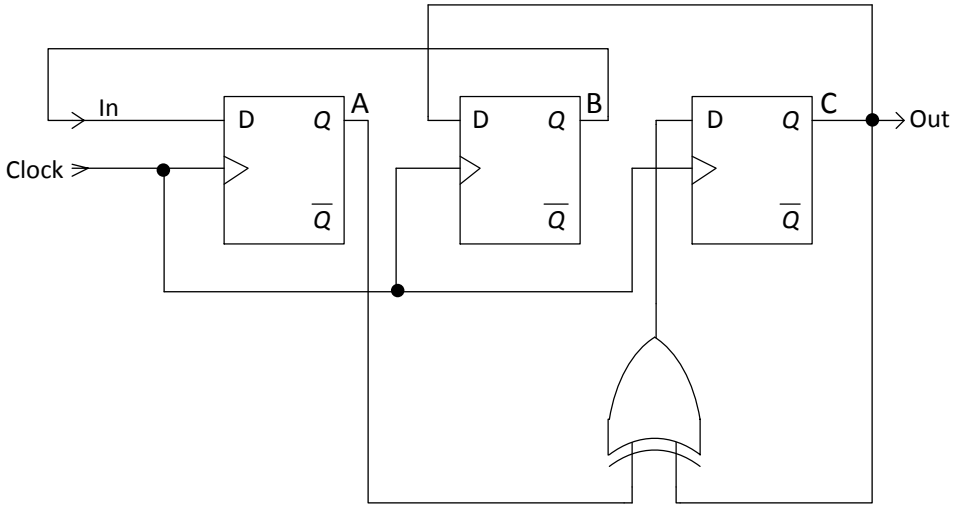
Παρούσα Κατάσταση			Επόμενη Κατάσταση		
A	*B*	*C*	*A*	*B*	*C*
1	0	0	0	0	1
1	0	1	0	1	0
0	0	1	0	1	1
0	1	0	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0
0	1	1	1	1	1

Εικόνα 3.8 : Πίνακας δυαδικών τιμών για ανάστροφη ακολουθία.



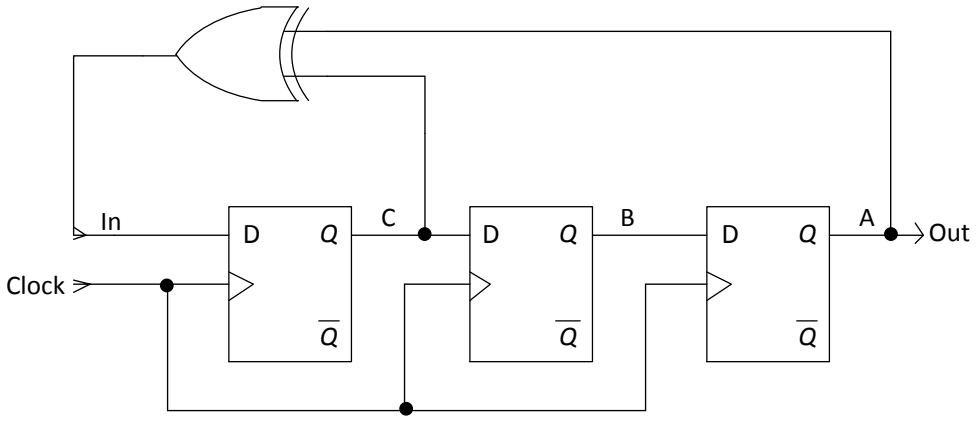
Εικόνα 3.9 : Χάρτες Karnaugh για ελαχιστοποίηση των συναρτήσεων της εικόνας 3.8.

Τώρα με χάρτες Karnaugh μπορούμε να εξάγουμε τις αντίστοιχες συναρτήσεις και να κατασκευάσουμε ένα αντίστοιχο κύκλωμα που να εκτελεί αντίστροφη ακολουθία.
Στην εικόνα 3.9 φαίνονται οι χάρτες Karnaugh και οι συναρτήσεις των εισόδων των flip-flop (όπου D_A,D_B και D_C είναι οι είσοδοι του πρώτου, δεύτερου και τρίτου flip-flop αντίστοιχα) και στην εικόνα 3.10 φαίνεται το αντίστοιχο κύκλωμα που κατασκευάστηκε από τις συναρτήσεις που εξήχθησαν από τους χάρτες.



Εικόνα 3.10 : Κύκλωμα σχεδιασμένο από τις συναρτήσεις της εικόνας 3.9.

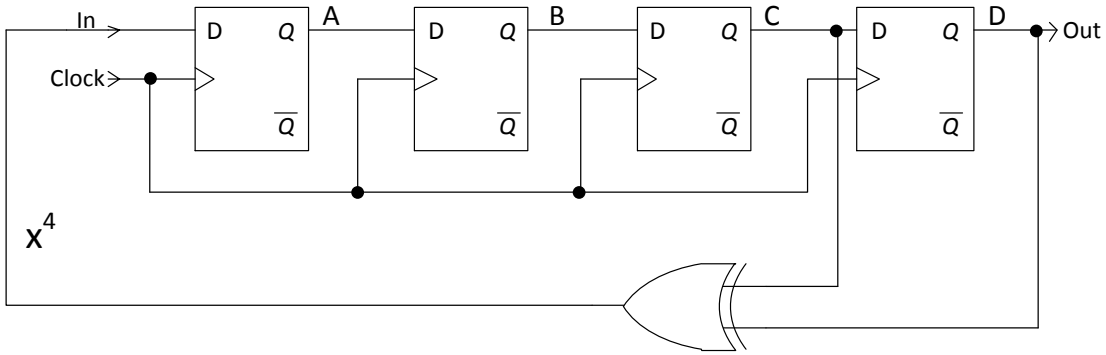
Το παραπάνω κύκλωμα μπορεί να απεικονιστεί σε μία πιο ευδιάκριτη μορφή αντιστρέφοντας τα flip-flop όπως φαίνεται στην εικόνα 3.11.



Εικόνα 3.11 : Το κύκλωμα της εικόνας 3.10 αλλά με ανεστραμμένα flip-flop.

Παρατηρώντας τα κυκλώματα που σχεδιάστηκαν παραπάνω βλέπουμε ότι το πρώτο flip-flop με είσοδο D_A τροφοδοτείται από την έξοδο του δευτέρου flip-flop B. Αντίστοιχα το δεύτερο flip-flop με είσοδο D_B τροφοδοτείται από την έξοδο του τρίτου flip-flop D. Επίσης το τρίτο flip-flop παίρνει είσοδο από την έξοδο της πύλης XOR. Η πύλη XOR συλλέγει στις εισόδους τις, τις εξόδους του πρώτου και τρίτου flip-flop (A και C αντίστοιχα).
Αν κοιτάξουμε τώρα στο αρχικό LFSR της εικόνας 3.6 το οποίο εκτελεί ορθή μέτρηση, βλέπουμε ότι το δεύτερο flip-flop με είσοδο D_B τροφοδοτείται από την έξοδο του πρώτου flip-flop A. Το τρίτο flip-flop με είσοδο D_C τροφοδοτείται από την έξοδο του δευτέρου flip-flop B. Επίσης το πρώτο flip-flop (με είσοδο D_A) παίρνει είσοδο από την έξοδο της πύλης XOR. Η πύλη XOR συλλέγει στις εισόδους τις, τις εξόδους του δευτέρου και τρίτου flip-flop (B και C αντίστοιχα).
Στο αρχικό κύκλωμα που είναι για ορθή μέτρηση το πρώτο flip-flop τροφοδοτεί το δεύτερο, το δεύτερο flip-flop τροφοδοτεί το τρίτο ενώ στο κύκλωμα που σχεδιάστηκε προηγουμένως για

ανάστροφη μέτρηση το δεύτερο flip-flop τροφοδοτεί το πρώτο, το τρίτο flip-flop τροφοδοτεί το δεύτερο.
Ας δούμε και ένα πρωταρχικό 4ου βαθμού Fibonacci LFSR το οποίο βρίσκεται στην εικόνα 3.12. Το LFSR αυτό έχει πολυώνυμο λειτουργίας $x^4 + x + 1$.



Εικόνα 3.12 : Πρωταρχικό 4ου βαθμού Fibonacci LFSR.

Η ακολουθία του LFSR φαίνεται στην εικόνα 3.13 όπου στην παρούσα κατάσταση έχουμε τις δυαδικές τιμές σε σειρά (‘0001’-‘1111’) εκτός τη τιμή ‘0000’ γιατί οδηγεί σε διακοπή λειτουργίας του κυκλώματος (dead state) και στην επόμενη κατάσταση έχουμε τις ανάλογες δυαδικές τιμές, που προέρχονται από τις εξόδους των flip-flop, για κάθε τιμή της παρούσας κατάστασης.

Παρούσα Κατάσταση				**Επόμενη Κατάσταση**			
A	*B*	*C*	*D*	*A*	*B*	*C*	*D*
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	0	0	0	1
0	1	0	0	0	0	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	0	1	1
0	1	1	1	0	0	1	1
1	0	0	0	0	1	0	0
1	0	0	1	1	1	0	0
1	0	1	0	1	1	0	1
1	0	1	1	0	1	0	1
1	1	0	0	0	1	1	0
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1

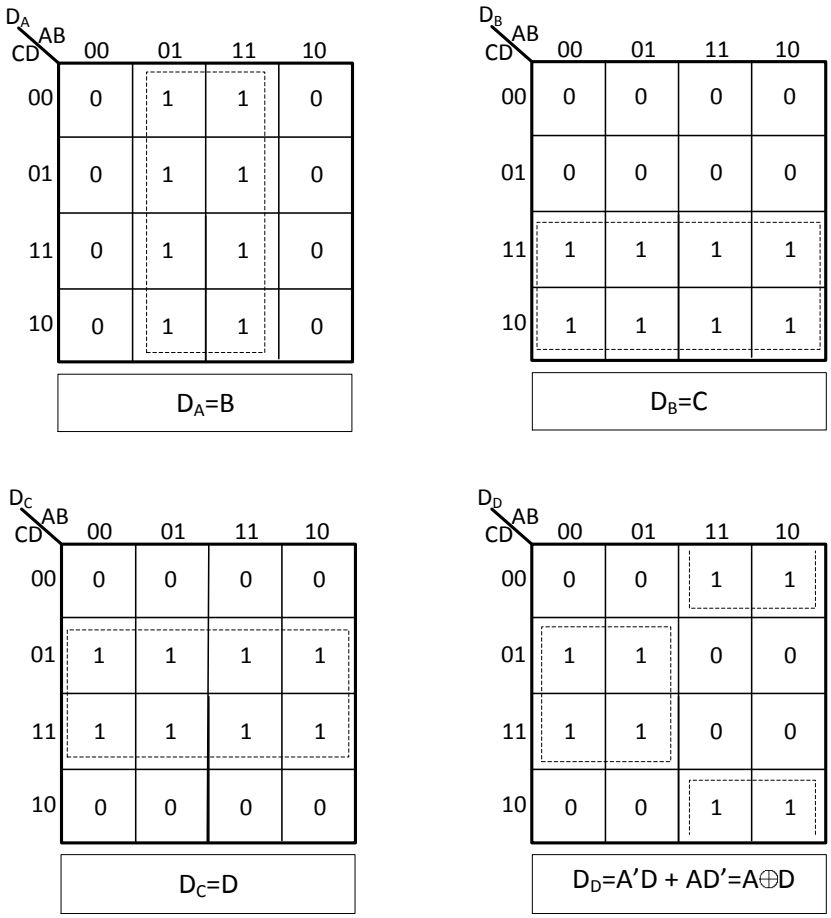
Εικόνα 3.13 : Πίνακας ακολουθίας του κυκλώματος της εικόνας 3.12.

Τα A, B, C και D είναι οι έξοδοι του πρώτου, δευτέρου, τρίτου και αντίστοιχα τέταρτου flip-flop. Εμείς επιθυμούμε να επιστρέψουμε από την επόμενη κατάσταση πίσω στην παρούσα κατάσταση. Δηλαδή από την τιμή ‘1000’ να επιστρέψουμε στην ‘0001’, από την τιμή ‘1001’ να επιστρέψουμε στην ‘0010’ κ.ο.κ.. Οπότε δημιουργούμε έναν νέο πίνακα, όπως φαίνεται στην εικόνα 3.14, όπου τώρα η επόμενη κατάσταση αποτελεί παρούσα κατάσταση και η παρούσα κατάσταση αποτελεί επόμενη κατάσταση και με χάρτες Karnaugh μπορούμε να εξάγουμε τις αντίστοιχες συναρτήσεις και να κατασκευάσουμε ένα αντίστοιχο κύκλωμα που να εκτελεί αντίστροφη ακολουθία.

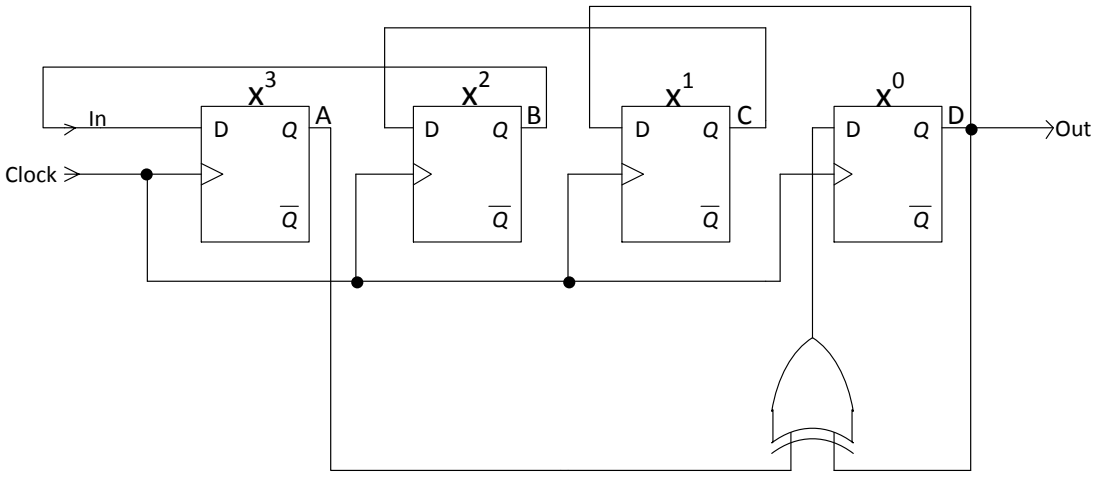
Στην εικόνα 3.15 φαίνονται οι χάρτες Karnaugh και οι συναρτήσεις των εισόδων των flip-flop (όπου D_A, D_B, D_C και D_D είναι οι είσοδοι του πρώτου, δεύτερου, τρίτου και τέταρτου flip-flop αντίστοιχα) και στην εικόνα 3.16 φαίνεται το αντίστοιχο κύκλωμα που κατασκευάστηκε από τις συναρτήσεις που εξήχθησαν από τους χάρτες.

Παρούσα Κατάσταση				**Επόμενη Κατάσταση**			
A	*B*	*C*	*D*	*A*	*B*	*C*	*D*
1	0	0	0	0	0	0	1
1	0	0	1	0	0	1	0
0	0	0	1	0	0	1	1
0	0	1	0	0	1	0	0
1	0	1	0	0	1	0	1
1	0	1	1	0	1	1	0
0	0	1	1	0	1	1	1
0	1	0	0	1	0	0	0
1	1	0	0	1	0	0	1
1	1	0	1	1	0	1	0
0	1	0	1	1	0	1	1
0	1	1	0	1	1	0	0
1	1	1	0	1	1	0	1
1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1

Εικόνα 3.14 : Πίνακας δυαδικών τιμών για ανάστροφη ακολουθία.

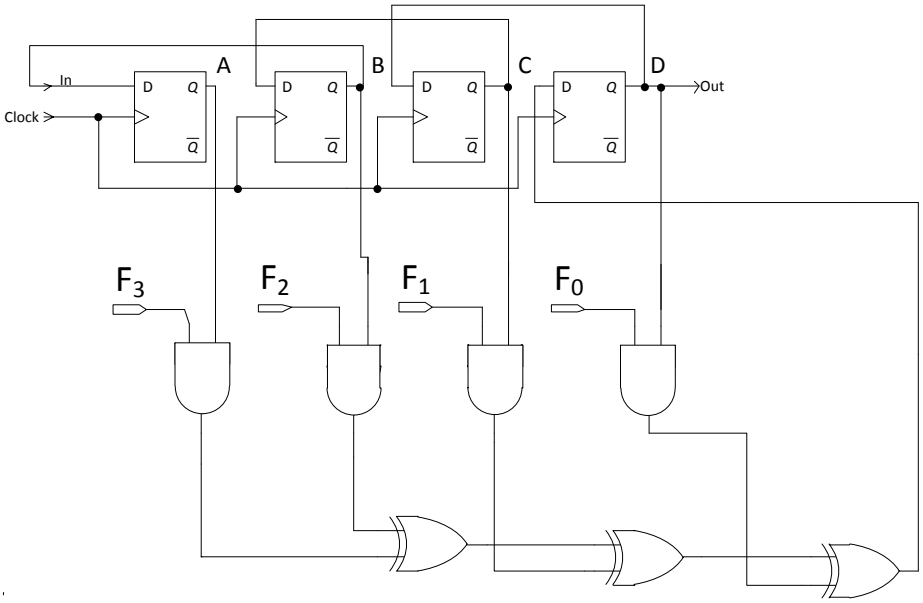


Εικόνα 3.15 : Χάρτες Karnaugh για ελαχιστοποίηση των συναρτήσεων της εικόνας 3.14.



Εικόνα 3.16 : Κύκλωμα σχεδιασμένο από τις συναρτήσεις της εικόνας 3.15.

Κοιτάζοντας το κύκλωμα της εικόνας 3.10 που είναι 3 bit βλέπουμε ότι είναι σχεδόν το ίδιο με το παραπάνω (εικόνα 3.16) απλώς προστίθεται ακόμα ένα flip-flop και η πύλη XOR μετατίθεται μία θέση δεξιά.
Το κύκλωμα μπορεί και να γίνει προγραμματιζόμενο όπως το αντίστοιχο, για ορθή μέτρηση, της εικόνας 3.4. Έτσι έχουμε:



Εικόνα 3.17 : Προγραμματιζόμενο Fibonacci LFSR 4 bit για ανάστροφη μέτρηση.

Οι συναρτήσεις των εισόδων των flip-flop για το αρχικό 4bit LFSR και για το αντίστοιχο κύκλωμα που εκτελεί αντίστροφη ακολουθία είναι:

Συναρτήσεις αρχικού LFSR	Συναρτήσεις ανάστροφου LFSR
$D_A = C \oplus D$	$D_A = B$
$D_B = A$	$D_B = C$
$D_C = B$	$D_C = D$
$D_D = C$	$D_D = A \oplus D$

Εικόνα 3.18 : Συναρτήσεις εισόδων 4 bit Fibonacci LFSR που εκτελούν ορθή-ανάστροφη μέτρηση.

Με όρους πολυωνύμου ο παραπάνω πίνακας είναι:

Συναρτήσεις αρχικού LFSR	Συναρτήσεις ανάστροφου LFSR
$DX^3 = X^1 \oplus X^0$	$DX^3 = X^2$
$DX^2 = X^3$	$DX^2 = X^1$
$DX^1 = X^2$	$DX^1 = X^0$
$DX^0 = X^1$	$DX^0 = X^3 \oplus X^0$

Εικόνα 3.19 : Συναρτήσεις εισόδων 4 bit Fibonacci LFSR που εκτελούν ορθή-ανάστροφη μέτρηση με όρους πολυωνύμου.

Αυτά ισχύουν για όλα τα πρωταρχικά (primitive) LFSR επειδή παράγουν όλες τις δυαδικές τιμές (μέγιστες) σε κάθε βαθμό. Δεν ισχύουν όμως για τα μη πρωταρχικά (non primitive) LFSR επειδή δεν παράγουν μέγιστες ακολουθίες αλλά παράγουν ακολουθίες σύμφωνα με τις αναδράσεις που θα πάρουν και την αρχική τιμή. Μπορούμε βέβαια να πάρουμε την αντίστροφη ακολουθία ενός μη πρωταρχικού LFSR αλλά ο τρόπος δόμησης του κυκλώματος διαφέρει με ένα άλλο μη πρωταρχικό LFSR ενώ στα πρωταρχικά ο τρόπος δόμησης είναι παρόμοιος για όλα.
Στην παρούσα πτυχιακή έχω σχεδιάσει ένα 16 bit Fibonacci LFSR. Αν του δώσουμε, για παράδειγμα, (το προγραμματίσουμε) με το πρωταρχικό πολυώνυμο $x^{16} + x^5 + x^4 + x^3 + 1$ τότε οι συναρτήσεις εισόδου για το αρχικό και για το αντίστροφο Fibonacci LFSR θα είναι:

Συναρτήσεις αρχικού LFSR	Συναρτήσεις ανάστροφου LFSR
$DX^{15} = X^5 \oplus X^4 \oplus X^3 \oplus X^0$	$DX^{15} = X^{14}$
$DX^{14} = X^{15}$	$DX^{14} = X^{13}$
$DX^{13} = X^{14}$	$DX^{13} = X^{12}$
$DX^{12} = X^{13}$	$DX^{12} = X^{11}$
$DX^{11} = X^{12}$	$DX^{11} = X^{10}$
$DX^{10} = X^{11}$	$DX^{10} = X^9$
$DX^9 = X^{10}$	$DX^9 = X^8$
$DX^8 = X^9$	$DX^8 = X^7$
$DX^7 = X^8$	$DX^7 = X^6$
$DX^6 = X^7$	$DX^6 = X^5$
$DX^5 = X^6$	$DX^5 = X^4$
$DX^4 = X^5$	$DX^4 = X^3$
$DX^3 = X^4$	$DX^3 = X^2$
$DX^2 = X^3$	$DX^2 = X^1$
$DX^1 = X^2$	$DX^1 = X^0$
$DX^0 = X^1$	$DX^0 = X^{15} \oplus X^4 \oplus X^3 \oplus X^2$

Εικόνα 3.20 : Συναρτήσεις εισόδων 16 bit Fibonacci LFSR που εκτελούν ορθή-ανάστροφη μέτρηση με όρους πολυωνύμου.

Έτσι, μπορούμε να πούμε ότι το πολυώνυμο για ορθή μέτρηση είναι το $x^{16} + x^5 + x^4 + x^3 + 1$ ενώ για την ανάστροφη μέτρηση είναι το $x^{15} + x^4 + x^3 + x^2$. Στο παράρτημα Β υπάρχει λίστα με μερικά πρωταρχικά πολυώνυμα $16^{ου}$ βαθμού

Παρόμοια πράγματα συμβαίνουν και στα Galois (εξωτερικά) LFSR μόνο που εδώ αλλάζουν οι συναρτήσεις των εισόδων των flip-flop. Οι συναρτήσεις των εισόδων των flip-flop για το αρχικό 4bit Galois LFSR και για το αντίστοιχο κύκλωμα που εκτελεί ανάστροφη ακολουθία είναι:

Συναρτήσεις αρχικού LFSR	Συναρτήσεις ανάστροφου LFSR
$D_A = D$	$D_A = B$
$D_B = A$	$D_B = C$
$D_C = B$	$D_C = A \oplus D$
$D_D = C \oplus D$	$D_D = A$

Εικόνα 3.21 : Συναρτήσεις εισόδων 4 bit Galois LFSR που εκτελούν ορθή-ανάστροφη μέτρηση.

Με όρους πολυωνύμου ο παραπάνω πίνακας είναι:

Συναρτήσεις αρχικού LFSR	Συναρτήσεις ανάστροφου LFSR
$DX^3 = X^0$	$DX^3 = X^2$
$DX^2 = X^3$	$DX^2 = X^1$
$DX^1 = X^2$	$DX^1 = X^3 \oplus X^0$
$DX^0 = X^1 \oplus X^0$	$DX^0 = X^3$

Εικόνα 3.22 : Συναρτήσεις εισόδων 4 bit Galois LFSR που εκτελούν ορθή-ανάστροφη μέτρηση με όρους πολυωνύμου.

Οι παραπάνω συναρτήσεις αποδεικνύονται αν κάνουμε χρήση χαρτών Karnaugh όπως και στο Fibonacci LFSR .

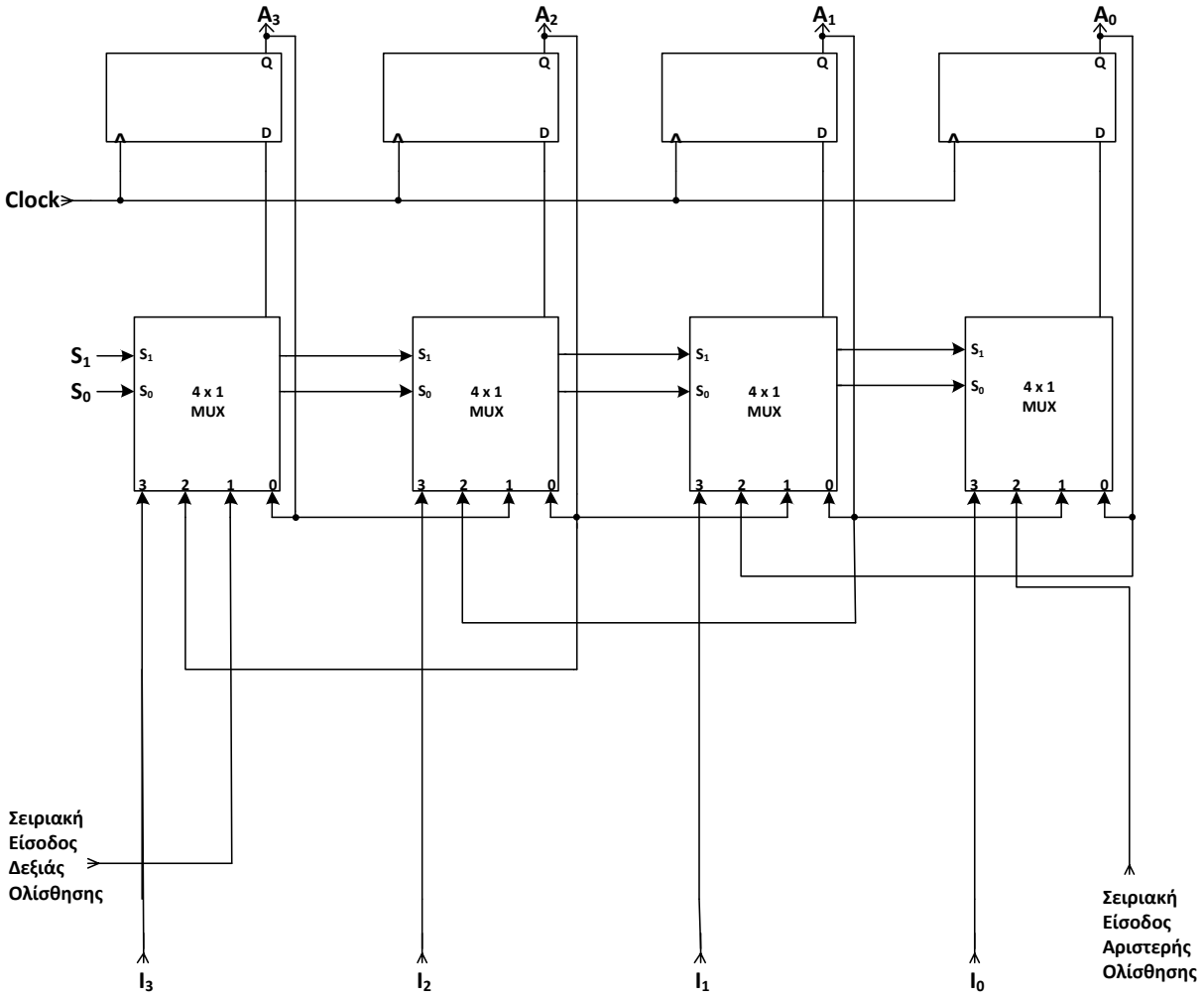
3.3. Αμφίδρομος Προγραμματιζόμενος Καταχωρητής Ολίσθησης

Μπορούμε να συμπεριλάβουμε τις παραπάνω λειτουργίες (ορθή και αντίστροφη ακολουθία) σε ένα κύκλωμα. Αυτό το κύκλωμα έχει τις εξής εισόδους και λειτουργίες:

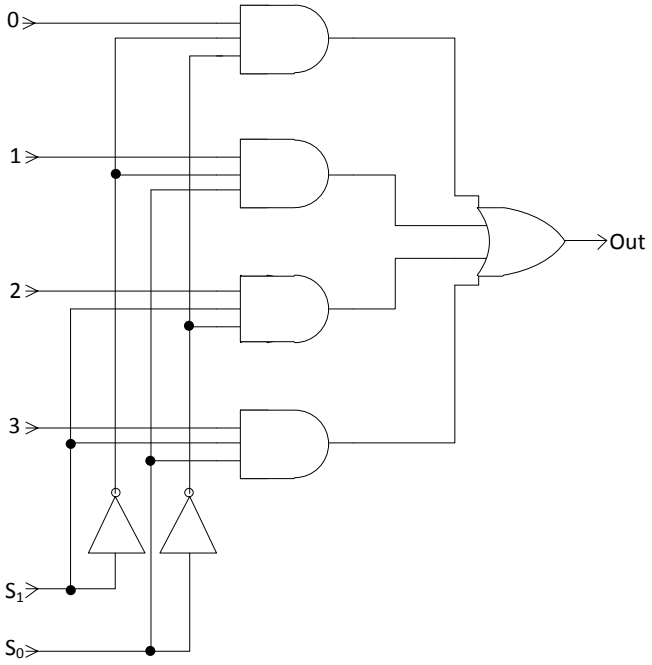
- Μία είσοδο *Clock* για τους παλμούς του ρολογιού για να συγχρονίζουν όλες τις λειτουργίες.
- Μία είσοδο ελέγχου «δεξιάς ολίσθησης» («shift right»), που να επιτρέπει τη λειτουργία ολίσθησης προς τα δεξιά.
- Μία είσοδο ελέγχου «αριστερής ολίσθησης» («shift left»), που να επιτρέπει τη λειτουργία ολίσθησης προς τα αριστερά.
- Μία είσοδο ελέγχου «παράλληλης φόρτωσης» («parallel load»), που να επιτρέπει τη μεταφορά των δεδομένων που βρίσκονται στις γραμμές παράλληλης εισόδου μέσα στον καταχωρητή.
- Μια κατάσταση ελέγχου, που αφήνει τις πληροφορίες μέσα στον καταχωρητή αναλλοίωτες, άσχετα με τους παλμούς του ρολογιού που συνεχίζουν να έρχονται.

Ένας καταχωρητής ικανός για ολισθήσεις τόσο δεξιά όσο και αριστερά λέγεται «αμφίδρομος καταχωρητής ολίσθησης» («bidirectional shift register»). Εάν μπορεί να κάνει ολισθήσεις μόνο προς τη μία κατεύθυνση, ονομάζεται «μονόδρομος καταχωρητής ολίσθησης» («unidirectional shift register»). Εάν ο καταχωρητής μπορεί να κάνει και ολισθήσεις και επίσης να φορτωθεί παράλληλα, τότε λέγεται «καταχωρητής ολίσθησης με παράλληλη φόρτωση» («shift register with parallel load»).
Η εικόνα 3.23 δείχνει έναν καταχωρητή ολίσθησης που έχει όλες τις παραπάνω ικανότητες. Αποτελείται από 4 flip-flop τύπου D (θα μπορούσαν να χρησιμοποιηθούν και τύπου RS, αν βάζαμε έναν αντιστροφέα ανάμεσα στα R και S). Οι πολυπλέκτες (MUX) αποτελούν μέρος του καταχωρητή και εμφανίζονται εδώ σε συμβολική μόνο μορφή (βλέπε εικόνα 3.24 για το λογικό διάγραμμα ενός πολυπλέκτη). Αυτοί οι 4 πολυπλέκτες έχουν δύο κοινές μεταβλητές επιλογής, S_1 και S_0. Όταν $S_1S_0 = 00$, τότε ο κάθε πολυπλέκτης επιλέγει την είσοδο 0. Όταν $S_1S_0 = 01$, τότε ο κάθε πολυπλέκτης επιλέγει την είσοδο 1. Όταν $S_1S_0 = 10$, τότε ο κάθε πολυπλέκτης επιλέγει την είσοδο 2. Όταν $S_1S_0 = 11$, τότε ο κάθε πολυπλέκτης επιλέγει την είσοδο 3.

Οι είσοδοι S_1 και S_0 ελέγχουν τον τρόπο λειτουργίας του καταχωρητή, όπως περιγράφει ο πίνακας στην εικόνα 3.25. Όταν $S_1S_0 = 00$, η παρούσα τιμή του καταχωρητή εφαρμόζεται στις εισόδους *D* των flip-flop. Έχουμε δηλαδή ένα δίαυλο από την έξοδο κάθε flip-flop πίσω ξανά στην είσοδο του. Άρα, ο επόμενος παλμός του ρολογιού «φορτώνει» ξανά το flip-flop με την ίδια τιμή που ήδη είχε, κι έτσι δεν υπάρχει καμιά αλλαγή κατάστασης. Όταν $S_1S_0 = 01$, οι είσοδοι 1 των πολυπλεκτών οδηγούνται στις εισόδους D των flip-flop. Αυτό προκαλεί μια ολίσθηση προς τα δεξιά, το δε flip-flop A_3 φορτώνεται από την αντίστοιχη σειριακή είσοδο. Όταν $S_1S_0 = 10$, έχουμε ολίσθηση προς τα αριστερά και συγχρόνως το flip-flop A_0 φορτώνεται από την άλλη σειριακή είσοδο. Τελικά, όταν $S_1S_0 = 11$, οι πληροφορίες που βρίσκονται στις γραμμές παράλληλης εισόδου μεταφέρονται μέσα στον καταχωρητή και στις εξόδους των flip-flop, όλες μαζί με τον επόμενο παλμό του ρολογιού (clock)[5].



Εικόνα 3.23 : Ένας αμφίδρομος καταχωρητής ολίσθησης 4 bit με παράλληλη φόρτωση.

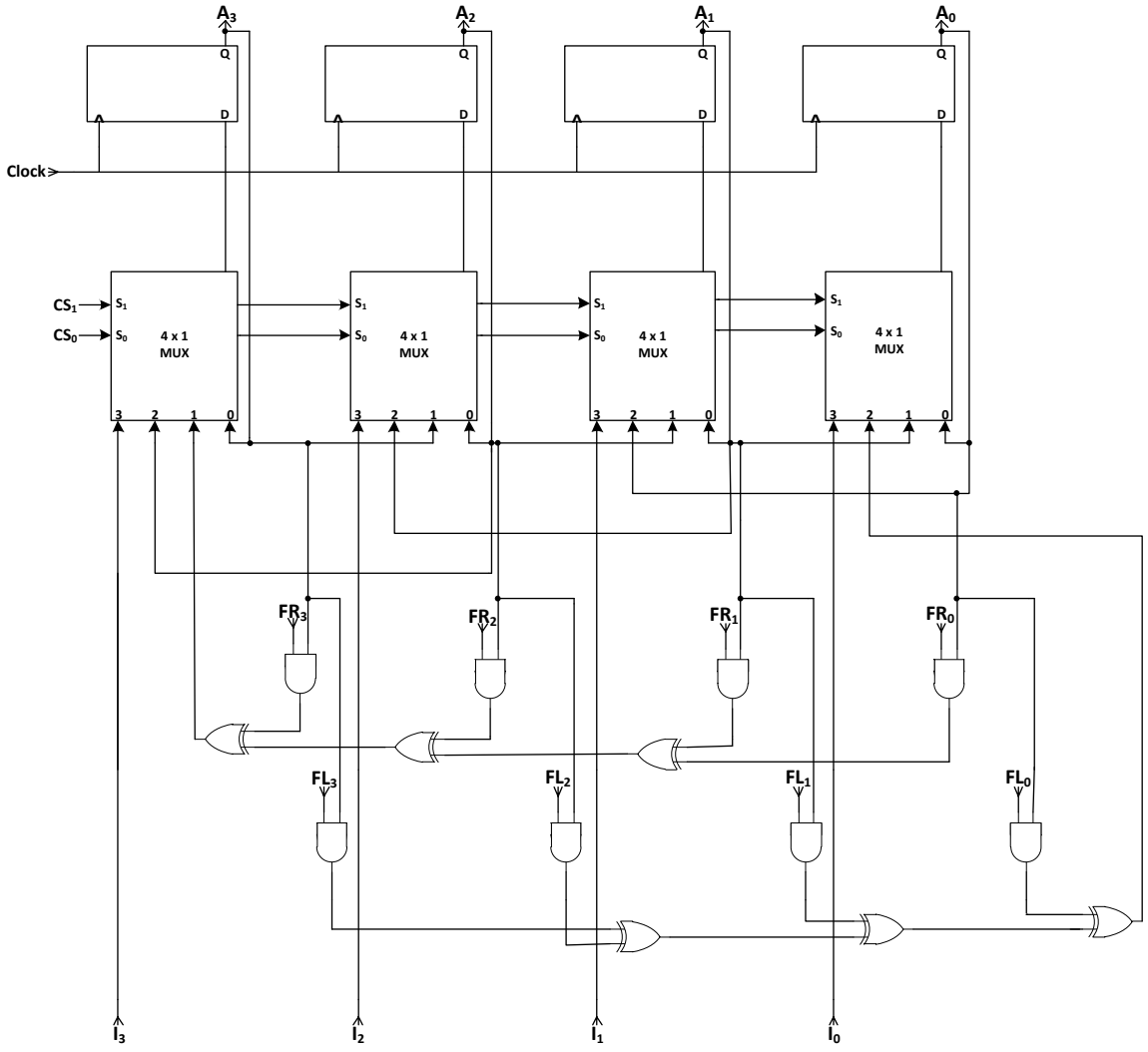


Εικόνα 3.24 : Λογικό διάγραμμα πολυπλέκτη 4 σε 1.

S1	S0	Λειτουργία του καταχωρητή
0	0	Αναλλοίωτος
0	1	Δεξιά ολίσθηση
1	0	Αριστερή ολίσθηση
1	1	Παράλληλη φόρτωση (seed)

Εικόνα 3.25 : Πίνακας λειτουργιών του καταχωρητή της εικόνας 3.12.

Για να ολοκληρωθεί το κύκλωμα πρέπει να προσθέσουμε τα κυκλώματα της δεξιάς και της αριστερής ολίσθησης όπως φαίνεται στην εικόνα 3.26. Με τις εισόδους FR και FL προγραμματίζουμε το αντίστοιχο LFSR (FR για το LFSR που εκτελεί ορθή-δεξιά ολίσθηση και FL για το LFSR που εκτελεί αντίστροφη-αριστερή ολίσθηση).

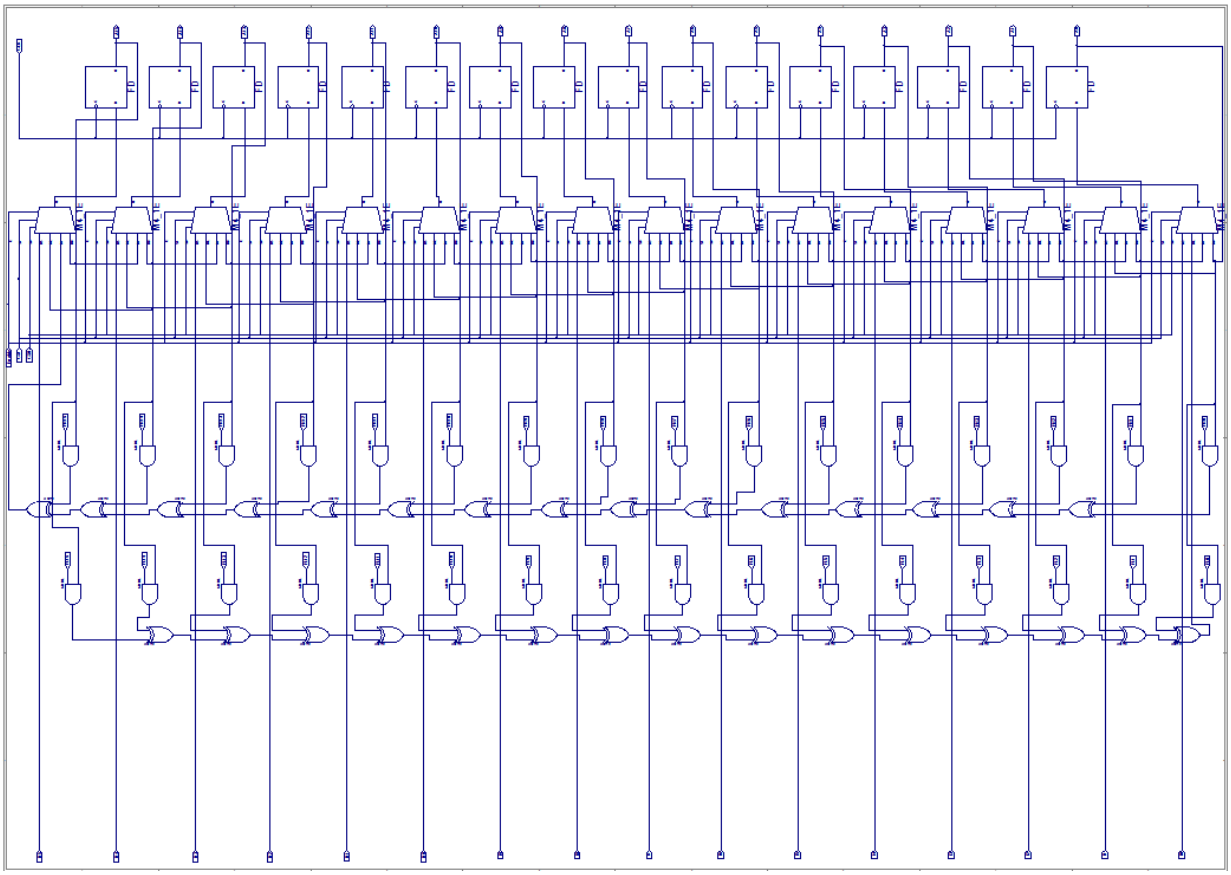


Εικόνα 3.26 : Ένας αμφίδρομος προγραμματιζόμενος καταχωρητής ολίσθησης 4 bit με ανάδραση και με παράλληλη φόρτωση.

Όπως βλέπουμε, έχουν ενσωματωθεί τα κυκλώματα της εικόνας 3.4 (Για ορθή-δεξιά ολίσθηση) και 3.17 (Για ανάστροφη-δεξιά ολίσθηση). Το κύκλωμα αυτό μπορεί να αυξομειωθεί για την κατασκευή LFSR πιο μεγάλου ή πιο μικρού βαθμού αρκεί να μεγαλώσουν/μικρύνουν οι φυσικές διαστάσεις του κυκλώματος. Μπορεί να γίνει παραμετροποίηση του κυκλώματος ώστε να χρησιμοποιηθούν Galois LFSR σύμφωνα με τις εξισώσεις των εισόδων τους που βρίσκονται στις εικόνες 3.21 και 3.22. Στην παρούσα πτυχιακή δεν γίνεται αυτό αλλά σχεδιάζεται το Fibonacci LFSR.

3.4. Σχηματική Σχεδίαση στην εφαρμογή ISE Xilinx 13.1

Στην παρούσα πτυχιακή επέλεξα να σχεδιάσω ένα 16 bit προγραμματιζόμενο LFSR με δυνατότητα αντιστροφής της ακολουθίας της εξόδου του που πληροί όλα τα παραπάνω. Η σχεδίαση έγινε με σχηματικό τρόπο στην εφαρμογή ISE Xilinx 13.1 επειδή ήθελα να δοκιμάσω αυτό το μέρος του προγράμματος εκτός της σχεδίασης σε κώδικα VHDL. Στο παράρτημα Α υπάρχει ένας οδηγός που περιγράφει την διαδικασία σχηματικής σχεδίασης. Στην εικόνα 3.27 φαίνεται το σχηματικό που σχεδιάστηκε.



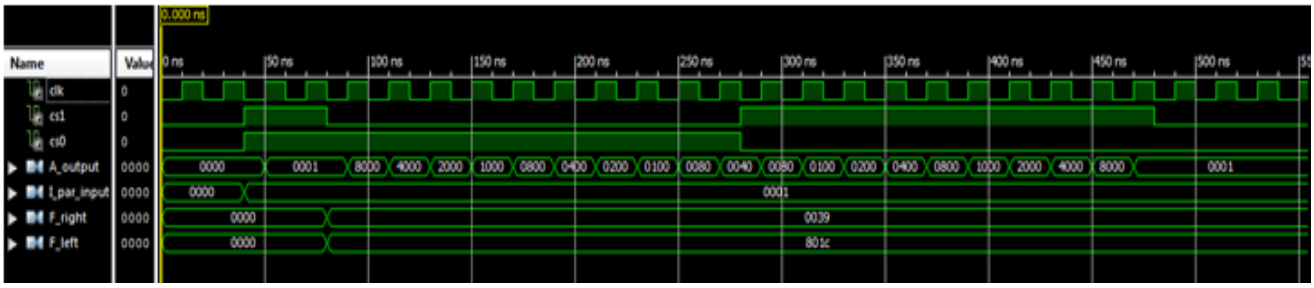
Εικόνα 3.27 : Σχηματική σχεδίαση στην εφαρμογή ISE Xilinx 13.1.

Μπορείτε να ανατρέξετε στο αρχείο της σχεδίασης (βρίσκεται μέσα στο CD που συνοδεύει την παρούσα εργασία) για μια πιο λεπτομερή απεικόνιση του σχηματικού.

Στην εικόνα 3.28 φαίνεται η προσομοίωση του σχεδιασθέντος κυκλώματος στο ISE Xilinx 13.1 για πολυώνυμο δεξιάς ολίσθησης (ορθή) το $x^{16} + x^5 + x^4 + x^3 + 1$, πολυώνυμο αριστερής ολίσθησης (αντίστροφη) το $x^{15} + x^4 + x^3 + x^2$ και αρχική τιμή (seed) το 0001 hex.

Στο παράρτημα Β υπάρχει λίστα με μερικά πρωταρχικά πολυώνυμα $16^{ου}$ βαθμού. Τα πολυώνυμα αυτά δίνονται σε σειρά αριθμών και το 0 (x^0) υπονοείται σε κάθε γραμμή πολυωνύμου. Περισσότερα πολυώνυμα $16^{ου}$ βαθμού αλλά και οποιουδήποτε άλλου βαθμού υπάρχουν στην ιστοσελίδα http://homepage.mac.com/afj/taplist.html σε αρχεία txt.

3.5. Προσομοίωση του Σχεδιασθέντος Κυκλώματος



Εικόνα 3.28 : Προσομοίωση του 16 bit προγραμματιζόμενου LFSR με δυνατότητα αντιστροφής της ακολουθίας.

Όπως βλέπουμε στην εικόνα προσομοίωσης, από πάνω προς τα κάτω, έχουμε:

- Το *clk* που είναι το ρολόι του κυκλώματος με περίοδο 20 nS (ρυθμίζεται από το αρχείο test bench).
- Τις δύο εισόδους *cs1 και cs0* που είναι οι είσοδοι επιλογής των πολυπλεκτών του κυκλώματος.
- Την έξοδο *A_output* που περιέχει τις εξόδους των flip-flop (*A0,A1,A2,…,A14 και A15*). Έχει ρυθμιστεί από το πρόγραμμα να εμφανίζει την τιμή της στο δεκαεξαδικό σύστημα για ευκολότερη ανάγνωση.
- Την είσοδο *I_par_input* που περιέχει τις εισόδους της παράλληλης φόρτωσης (*I0,I1,I2,…,I14 και I15*). Παρόμοια, έχει ρυθμιστεί να εμφανίζει την τιμή της στο δεκαεξαδικό σύστημα.
- Τις εισόδους *F_right* και *F_left* που περιέχουν τις εισόδους του προγραμματισμού του LFSR για την δεξιά ολίσθηση (*FR0,FR1,FR2,…,FR14 και FR15*) και αντίστοιχα για την αριστερή ολίσθηση (*FL0,FL1,FL2,…,FL14 και FL15*) πάλι ρυθμισμένες στο δεκαεξαδικό σύστημα.

Ξεκινώντας η προσομοίωση βλέπουμε ότι όταν οι είσοδοι *cs1 και cs0* είναι ‘00’ η έξοδος είναι ‘0000’. Αυτό συμβαίνει γιατί στο ‘00’ παραμένει αναλλοίωτη η έξοδος του κυκλώματος και επειδή τώρα ξεκινάει η προσομοίωση υποθέτει ότι είναι ‘0000’ η προηγούμενη κατάσταση της εξόδου. Μετά το *cs1 και cs0* παίρνει τη τιμή ‘11’και γίνεται παράλληλη φόρτωση (Η τιμή της εισόδου *I_par_input* μεταφέρετε στην έξοδο) όταν έλθει θετικό μέτωπο του ρολογιού (50 nS). Η είσοδος είναι ‘0001’ άρα και η έξοδος παίρνει την τιμή ‘0001’. Μετά το *cs1 και cs0* παίρνει τη τιμή ‘01’ όπου και γίνεται δεξιά ολίσθηση σύμφωνα με το πολυώνυμο που δώσαμε (αφού το πολυώνυμο είναι το $x^{16} + x^5 + x^4 + x^3 + 1$ τότε η είσοδος *F_right* παίρνει την τιμή 0000 0000 0011 1001 ή 0039 hex όπως φαίνεται) μόλις έρθει θετικό μέτωπο του ρολογιού (90 nS) . Μετά το *cs1 και cs0* παίρνει τη τιμή ‘10’ όπου και γίνεται αριστερή ολίσθηση σύμφωνα με το πολυώνυμο που δώσαμε (Αφού το πολυώνυμο είναι το $x^{15} + x^4 + x^3 + x^2$ τότε η είσοδος *F_ left* παίρνει την τιμή 1000 0000 0001 1100 ή 801c hex όπως φαίνεται) μόλις έρθει θετικό μέτωπο του ρολογιού (280 nS). Τέλος το *cs1 και cs0* παίρνει τη τιμή ‘00’ όπου η έξοδος του κυκλώματος παραμένει αναλλοίωτη. Αυτό συμβαίνει μόλις έρθει θετικό μέτωπο του ρολογιού (490 nS). Βλέπουμε ότι το κύκλωμα δουλεύει σωστά καθώς στην δεξιά ολίσθηση η έξοδος λαμβάνει τιμές ‘8000’, ‘4000’,… ,‘0040’ ενώ στην αριστερή ολίσθηση παίρνουμε τις τιμές με αντίστροφη φορά ‘0080’, ‘0100’, …, ‘0001’.

4. Υλοποίηση σε FPGA

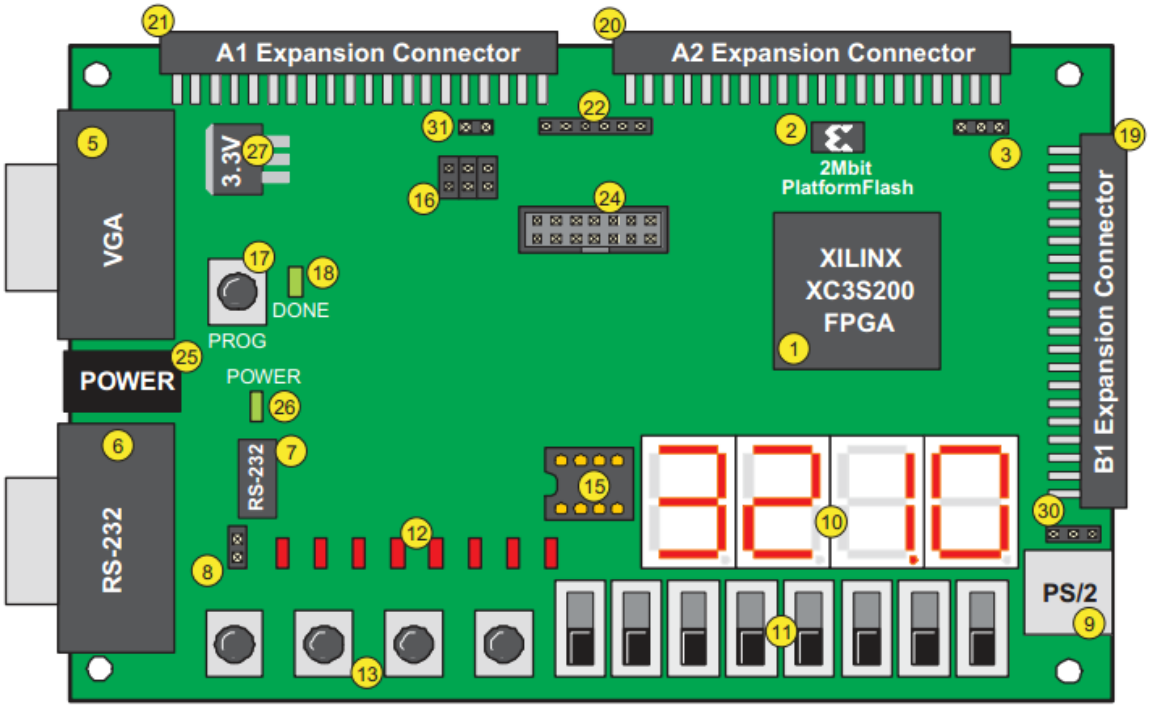
Στο προηγούμενο κεφάλαιο επεξηγήθηκε ο τρόπος με τον οποίο μπορεί ένα LFSR να γίνει προγραμματιζόμενο όπως επίσης και ο τρόπος να έχουμε αντιστροφή της ακολουθίας της εξόδου του. Στο παρόν κεφάλαιο περιγράφετε η εφαρμογή που υλοποίησα στο αναπτυξιακό Spartan-3 Starter Kit Board και δίδονται πληροφορίες για το αναπτυξιακό και τα εξαρτήματα που το συντελούν. Στο τέλος του κεφαλαίου καταγράφονται οι κώδικες στη γλώσσα VHDL που υλοποιούν την εφαρμογή.

4.1. Μελέτη Αναπτυξιακού

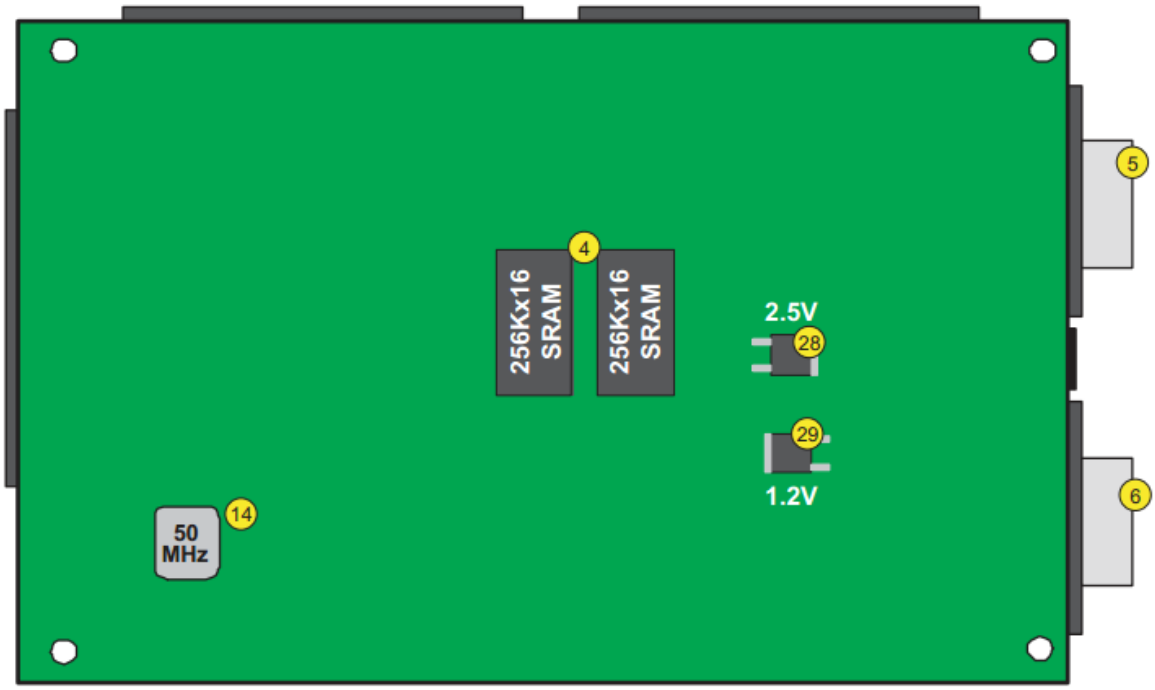
Στα πλαίσια της πτυχιακής μου εργασίας σχεδίασα και υλοποίησα μία εφαρμογή της ψηφιακής σχεδίασης με βάση το «**Spartan-3 Starter Kit Board**» όπου είναι ένα αναπτυξιακό της **Digilent** και έχει σαν πυρήνα το FPGA **Spartan-3** της Xilinx. Το FPGA ή Field Programmable Gate Array (συστοιχία επιτόπια προγραμματιζόμενων πυλών) είναι τύπος προγραμματιζόμενου ολοκληρωμένου κυκλώματος γενικής χρήσης το οποίο διαθέτει πολύ μεγάλο αριθμό τυποποιημένων πυλών και άλλων ψηφιακών λειτουργιών όπως απαριθμητές, καταχωρητές μνήμης, γεννήτριες PLL κα. Σε ορισμένα από αυτά ενσωματώνονται και αναλογικές λειτουργίες. Κατά τον προγραμματισμό του FPGA, ο οποίος γίνεται πάντοτε ενώ αυτό είναι τοποθετημένο στο τυπωμένο κύκλωμα, ενεργοποιούνται οι επιθυμητές λειτουργίες και διασυνδέονται μεταξύ τους έτσι ώστε το FPGA να συμπεριφέρεται ως ολοκληρωμένο κύκλωμα με συγκεκριμένη λειτουργία[13].

Η εφαρμογή που σχεδίασα και υλοποίησα είναι η προβολή της ακολουθίας εξόδου του αμφίδρομου καταχωρητή ολίσθησης με παράλληλη φόρτωση και με δυνατότητα αντιστροφής της ακολουθίας του, μήκους 16 bit, που σχεδιάστηκε παραπάνω, στο 7-segment display του αναπτυξιακού και ο έλεγχος αυτού μέσω διακοπτών και push buttons του αναπτυξιακού. Στην αντίστοιχη ενότητα δίνονται περαιτέρω πληροφορίες.

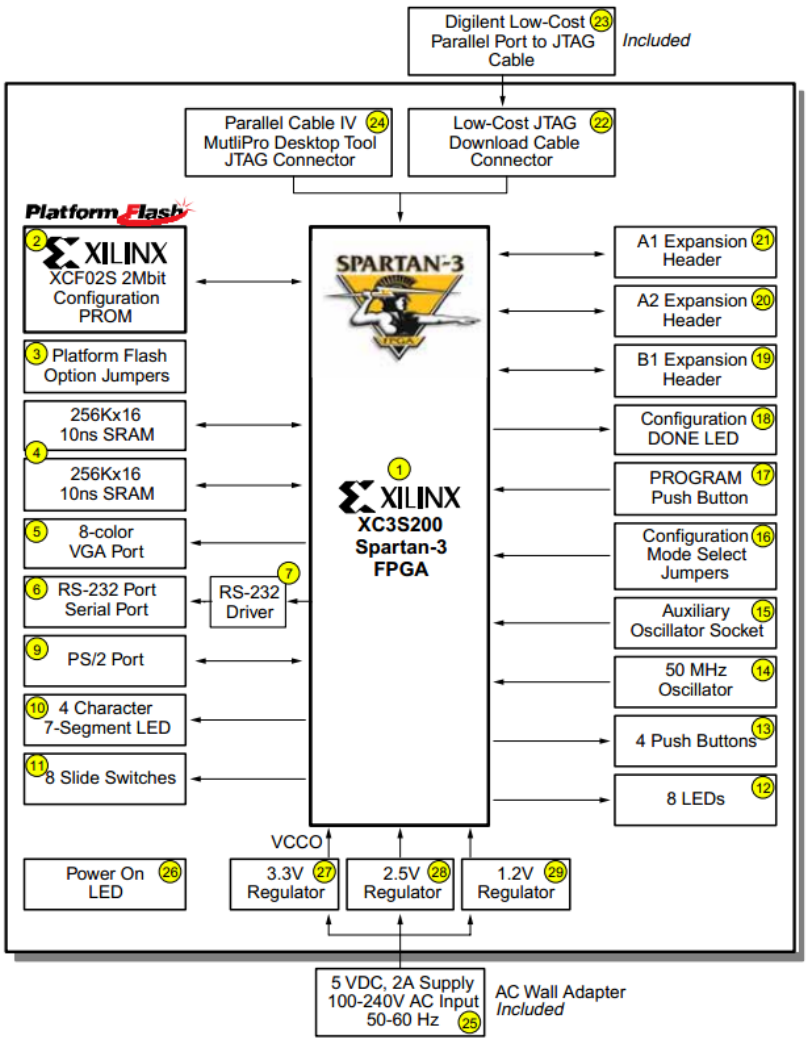
Στις παρακάτω εικόνες φαίνονται και αναλύονται τα χαρακτηριστικά του αναπτυξιακού:



Εικόνα 4.1 : Η πλακέτα του αναπτυξιακού (Πρόσθια όψη).



Εικόνα 4.2 : Η πλακέτα του αναπτυξιακού (Οπίσθια όψη).

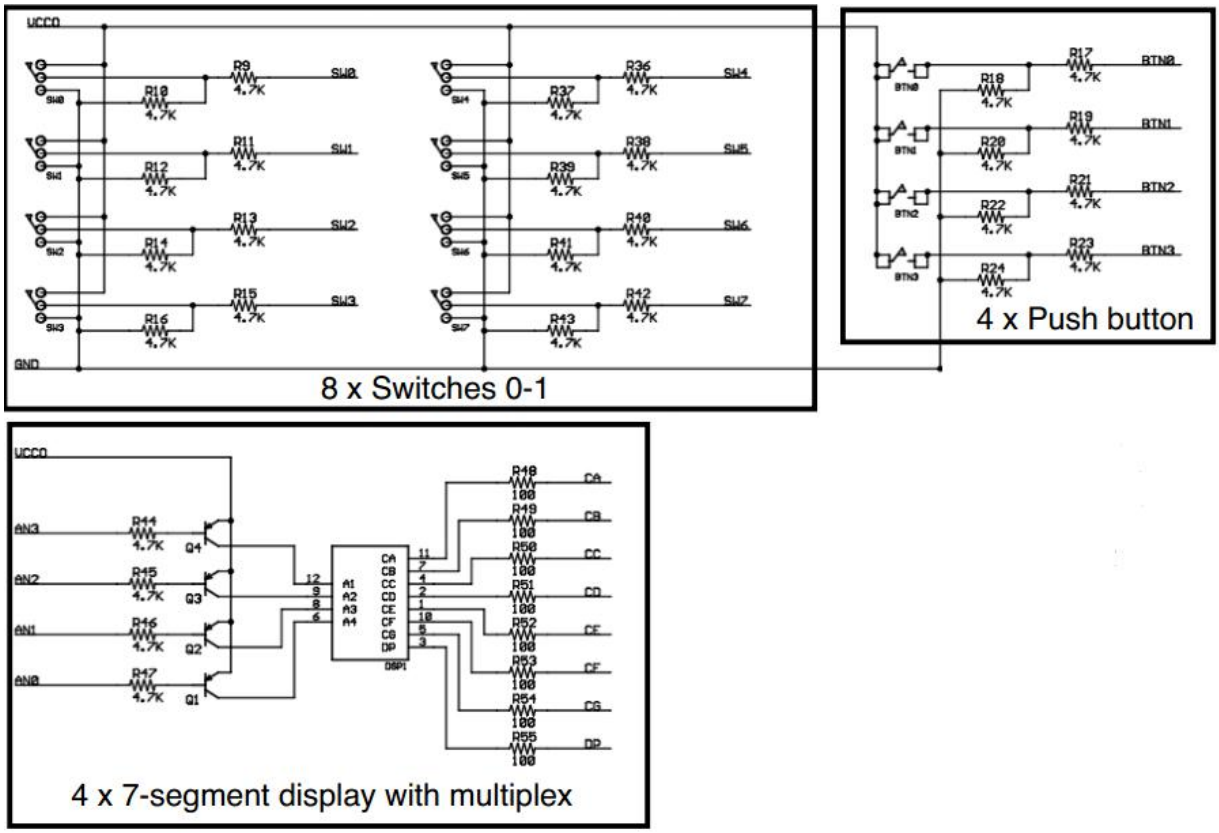


Εικόνα 4.3 : Διάγραμμα βαθμίδας του αναπτυξιακού.

Τα νούμερα στις παραπάνω εικόνες υποδηλώνουν τα παρακάτω[7][14]:

1. Ο πυρήνας είναι το FPGA Spartan-3 (στην έκδοση που έχουμε «φοράει» το **XC3S1000** σε BGA package με 256 Pins)
2. Μια 2 Mbit flash στην οποία φορτώνεται ο χάρτης προγραμματισμού του FPGA
3. DIP διακόπτες που ρυθμίζουν τον προγραμματισμό και την εκκίνηση του FPGA
4. Δύο ασύγχρονες μνήμες RAM διαταγμένες στα 16 bit με 256K θέσεις και με access time 10ns
5. Μια πόρτα για την διασύνδεση με οθόνη VGA
6. Πρώτη πόρτα σειριακής επικοινωνία RS-232 με κάποια άλλη συσκευή
7. Μετατροπέας στάθμης για τις δύο πόρτες σειριακή επικοινωνίας (MAX-232)
8. Δεύτερη πόρτα σειριακής επικοινωνίας RS-232
9. Πόρτα διασύνδεσης με συσκευές PS/2
10. Τέσσερα 7-segment display με αρνητική οδήγηση και πολύπλεξη
11. 8 διακόπτες 0-1 τύπου κύλισης
12. 8 led
13. 4 πιεστικοί διακόπτες θετικής λογικής
14. Ένας 50 MHz κρυσταλλικός ταλαντωτής οδηγούμενος σε μία από τις 8 εισόδους ρολογιού του FPGA
15. Πόρτα οδήγησης δευτερεύοντος ρολογιού
16. DIP διακόπτες για την ρύθμιση της λειτουργίας του FPGA
17. Πιεστικός διακόπτης αρχικοποίησης του FPGA
18. Led αρχικοποίησης του FPGA
19. 40 pin πόρτα B1
20. 40 pin πόρτα A2
21. 40 pin πόρτα A1
22. JTAG για τον προγραμματισμό του FPGA
23. Καλώδια USB σε JTAG
24. Parallel JTAG για το debug
25. Είσοδος για Εξωτερικό τροφοδοτικό 5V
26. Led ένδειξης λειτουργίας
27. Γραμμικός σταθεροποιητής 3.3V
28. Γραμμικός σταθεροποιητής 2.5V
29. Γραμμικός σταθεροποιητής 1.2V
30. Επιλογή τάσης τροφοδοσίας PS/2 (3.3V-5V)

Στην εφαρμογή που σχεδίασα χρησιμοποίησα τους 2 από τους 4 πιεστικούς διακόπτες, τους 8 διακόπτες 0-1 και τα 4 7-segment display. Παραθέτω παρακάτω τα κυκλωματικά τους διαγράμματα και τις ονομασίες τους :



Εικόνα 4.4 : Κυκλωματικά διαγράμματα.

Switch	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
FPGA Pin	K13	K14	J13	J14	H13	H14	G12	F12

Εικόνα 4.5 : Ονόματα Διακοπτών.

Push Button	BTN3 (User Reset)	BTN2	BTN1	BTN0
FPGA Pin	L14	L13	M14	M13

Εικόνα 4.6 : Ονόματα push buttons.

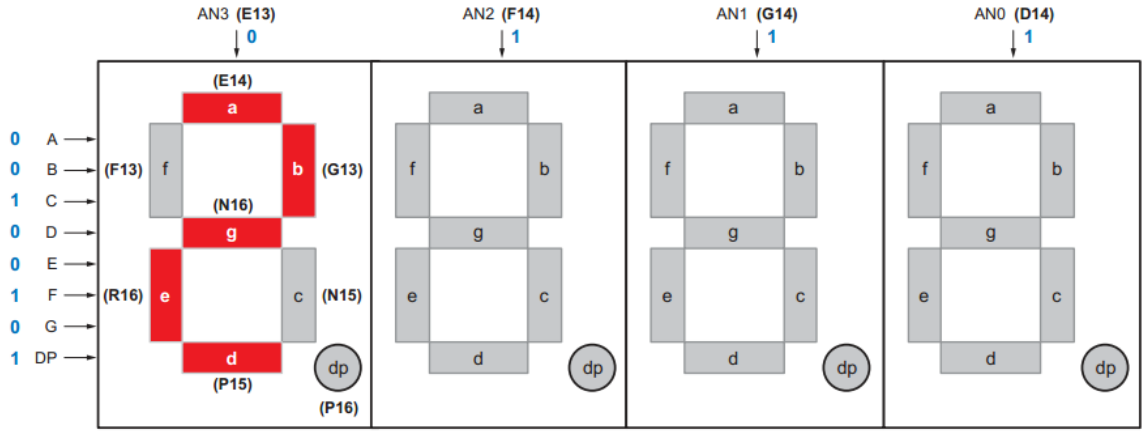
Τα ονόματα των 7-segment displays δίδονται παρακάτω στην ενότητα 4.2.

Οι 8 διακόπτες δίνουν ‘0’ ή ‘1’ ανάλογα με τη θέση που έχουν. Όταν ο διακόπτης είναι κάτω δίνει ‘0’ ενώ όταν είναι πάνω δίνει ‘1’. Στην κατάσταση μετάβασης των διακοπτών η τιμή που δίνουν είναι ‘0’ λόγω της pull down αντίστασης που έχουν στη μεσαία λήψη.
Οι 4 πιεστικοί διακόπτες είναι τύπου NO (Normal Open) και όταν πατηθούν δίνουν ‘1’ αλλιώς με τις pull down αντιστάσεις που έχουν δίνουν ‘0’.
Τα 7-segment είναι αρνητικής λογικής και τα καλώδια πολύπλεξης επίσης. Αυτά οδηγούν το ανάλογο display που θέλουμε να τροφοδοτήσουμε[7][15].

4.2. Τα 7-segment LED Display

Το αναπτυξιακό **Spartan-3 Starter Kit Board** περιλαμβάνει ένα τεσσάρων χαρακτήρων 7-segment LED Display που ελέγχεται από το FPGA με τα I/O pins, όπως φαίνεται στην εικόνα 4.7. Κάθε ψηφίο διαμοιράζεται τα οκτώ κοινά σήματα ελέγχου (A, B, C,…, G ,DP) που τροφοδοτούν

τα LED κάθε χαρακτήρα. Κάθε χαρακτήρας έχει μία ξεχωριστή είσοδο ελέγχου, κοινής ανόδου(AN3, AN2, AN1, AN0). Ο αριθμός του κάθε pin που συνδέεται με το FPGA με κάθε LED φαίνεται μέσα σε παρένθεση. Για να ενεργοποιήσουμε κάθε LED (να ανάψει δηλαδή) πρέπει να του εφαρμόσουμε δυναμικό λογικής στάθμης '0'. Επίσης για να ενεργοποιήσουμε τον κάθε χαρακτήρα (από τους τέσσερις) πρέπει και αυτός στην είσοδο ελέγχου που διαθέτει να τεθεί σε λογικό '0'. Στην εικόνα 4.7 ,για παράδειγμα, ο αριστερός-σημαντικότερος χαρακτήρας απεικονίζει την τιμή '2'. Οι ψηφιακές τιμές που οδηγούν το display σε αυτό το παράδειγμα φαίνονται δίπλα στα οκτώ κοινά σήματα ελέγχου. Η AN3 είσοδος ελέγχου κοινής ανόδου είναι σε λογικό '0' για να ενεργοποιήσει τον αριστερό-σημαντικότερο χαρακτήρα. Οι υπόλοιποι που είναι σε λογικό '1' δεν ενεργοποιούνται. Οι είσοδοι ελέγχου του segment , A έως G και DP σε χαμηλό δυναμικό ενεργοποιούν (ανάβουν) τα αντίστοιχα LED ενώ σε υψηλό δυναμικό απενεργοποιούν (σβήνουν) τα LED[14].



Εικόνα 4.7 : Έλεγχος 7-segment display

Στην εικόνα 4.8 υπάρχει μία λίστα των συνδέσεων του FPGA που οδηγούν τα LED σε κάθε 7-segment χαρακτήρα. Στην εικόνα 4.9 υπάρχουν οι συνδέσεις του FPGA που ενεργοποιούν κάθε χαρακτήρα. Στην εικόνα 4.10 υπάρχουν οι δυαδικές τιμές που πρέπει να τροφοδοτήσουν τα LED για να απεικονίσουν κάθε δεκαεξαδικό χαρακτήρα και στην εικόνα 4.11 υπάρχουν οι χαρακτήρες που απεικονίζονται στο seven-segment LED για κάθε δεκαεξαδικό χαρακτήρα.

Segment	FPGA Pin
A	E14
B	G13
C	N15
D	P15
E	R16
F	F13
G	N16
DP	P16

Εικόνα 4.8 : Συνδέσεις FPGA στο Seven-segment Display (ενεργές στο λογικό '0').

Anode Control	AN3	AN2	AN1	AN0
FPGA Pin	E13	F14	G14	D14

Εικόνα 4.9 : Συνδέσεις FPGA για ενεργοποίηση των ανάλογων χαρακτήρων (ενεργές στο λογικό '0').

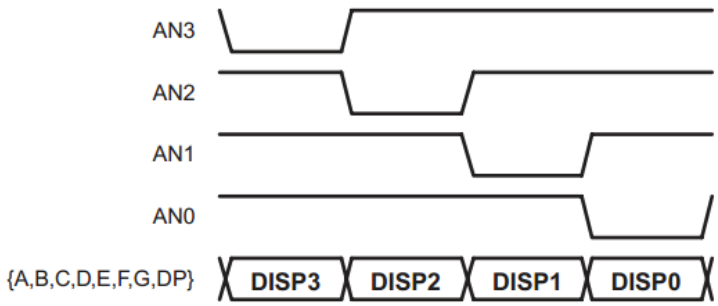
Character	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
b	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
d	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

Εικόνα 4.10 : Εικονιζόμενοι χαρακτήρες και οι λογικές τιμές που τροφοδοτούν τα LED του 7-segment για την εμφάνιση τους.

ΔΕΚΑΕΞΑΔΙΚΟ ΨΗΦΙΟ	ΧΑΡΑΚΤΗΡΑΣ ΣΤΟ 7-SEGMENT
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	A
B	b
C	C
D	d
E	E
F	F

Εικόνα 4.11 : Εικονιζόμενοι χαρακτήρες στο 7-segment display για κάθε δεκαεξαδικό ψηφίο.

Τα σήματα ελέγχου των LED πολυπλέκονται χρονικά (time-multiplexed) για να απεικονίσουν τα δεδομένα και στους τέσσερις χαρακτήρες, όπως φαίνεται στην εικόνα 4.12.
Η πολύπλεξη των τεσσάρων 7-segment Display γίνεται με την χρήση του φαινομένου του μεταισθήματος όπου το ανθρώπινο μάτι αντιλαμβάνεται τις πολύ γρήγορες διαδοχικές κινήσεις σαν μία συνεχή κίνηση (π.χ. ένας φακός που τον βλέπουμε από απόσταση, μετακινείται γρήγορα και σχηματίζει μια συνεχή γραμμή, εικόνα που στην πραγματικότητα δεν υπάρχει απλά έτσι την επεξεργάζεται ο εγκέφαλος μας λόγω του φαινομένου αυτού).

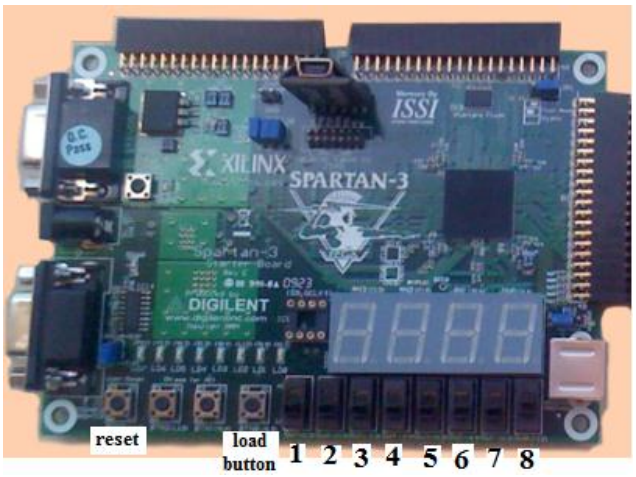


Εικόνα 4.12 : Οδήγηση εισόδων κοινής ανόδου με χαμηλό δυναμικό για την ενεργοποίηση κάθε χαρακτήρα από τους τέσσερις.

Η τεχνική της πολύπλεξης γίνεται για να γλιτώσουμε καλώδια οδήγησης από τα 32 καλώδια (8 data x 4 digit) που θα χρειάζονταν για οδήγηση του κάθε στοιχείου ξεχωριστά με 12 καλώδια (8 data + 4 multiplex) έχουμε το ίδιο αποτέλεσμα[7][14].

4.3. Η εφαρμογή

Η εφαρμογή που σχεδίασα και υλοποίησα είναι η **προβολή της ακολουθίας εξόδου του κυκλώματος της εικόνας 3.27 (**16 bit αμφίδρομος καταχωρητής ολίσθησης με παράλληλη φόρτωση και με δυνατότητα αντιστροφής της ακολουθίας του) **που σχεδιάστηκε παραπάνω στο 7-segment display και ο έλεγχος αυτού μέσω διακοπτών και push buttons του αναπτυξιακού**.
Πιο συγκεκριμένα το αναπτυξιακό όπως ειπώθηκε παραπάνω έχει 8 διακόπτες κύλισης 0-1 και 4 push buttons. Στην εφαρμογή αυτή κάνω χρήση και των 8 διακοπτών κύλισης και 2 εκ των 4 push button. Στην εικόνα 4.13 φαίνεται το αναπτυξιακό με αριθμημένους τους διακόπτες κύλισης και τα 2 push buttons που χρησιμοποίησα.



Εικόνα 4.13 : Φωτογραφία Αναπτυξιακού.

Οι διακόπτες κύλισης και τα push-buttons διαδραματίζουν τον εξής ρόλο:

Με τους **διακόπτες 3 και 4 να είναι '00'** (προς τα κάτω δηλαδή) εισάγουμε την ***τιμή της παράλληλης φόρτωσης (seed)*** από τους **διακόπτες 5 έως και 8 (το περισσότερο σημαντικό ψηφίο στο διακόπτη 5 και το λιγότερο σημαντικό ψηφίο στο διακόπτη 8)** . Οι τιμές όμως είναι των 16 bit. Άρα θα πρέπει να τις χωρίσουμε σε 4 nibbles (τετράδες από bit). Αυτό γίνεται χρησιμοποιώντας και τους **διακόπτες 1 και 2**. Όταν **1 και 2 είναι '00'** εισάγουμε την **πρώτη τετράδα** (την δημιουργούμε στους διακόπτες 5 έως και 8). Μόλις την εισάγουμε πιέζουμε και αφήνουμε το **push button "load button"** αριστερά του διακόπτη 1 όπως φαίνεται στην εικόνα 4.13 για να εισαχθεί η τιμή στο FPGA. Συνεχίζουμε με την **δεύτερη τετράδα** θέτοντας τους διακόπτες **1 και 2 να είναι '01'** και πιέζουμε πάλι το "load button" . Κατόπιν με την **τρίτη τετράδα** θέτοντας τους διακόπτες **1 και 2 να είναι '10'** και πιέζουμε πάλι το "load button" και τέλος με την **τέταρτη**

τετράδα θέτοντας τους διακόπτες **1 και 2 να είναι '11'** και πιέζουμε ξανά το "load button". Έτσι, τέλειωσε η εισαγωγή της τιμής της παράλληλης φόρτωσης (seed).
Με τους **διακόπτες 3 και 4 να είναι '01'** εισάγουμε την ***τιμή του πολυωνύμου δεξιάς ολίσθησης*** από τους **διακόπτες 5 έως και 8**. Οι τιμές όμως είναι και εδώ των 16 bit. Άρα θα πρέπει να τις χωρίσουμε και εδώ σε 4 nibbles (τετράδες από bit). Αυτό γίνεται χρησιμοποιώντας και τους **διακόπτες 1 και 2**. Όταν **1 και 2 είναι '00'** εισάγουμε την **πρώτη τετράδα** (την δημιουργούμε στους διακόπτες 5 έως και 8). Μόλις την εισάγουμε πιέζουμε και αφήνουμε το **push button "load button"** αριστερά του διακόπτη 1 όπως φαίνεται στην εικόνα 4.13 για να εισαχθεί η τιμή στο FPGA. Συνεχίζουμε με την **δεύτερη τετράδα** θέτοντας τους διακόπτες **1 και 2 να είναι '01'** και πιέζουμε πάλι το "load button" . Κατόπιν με την **τρίτη τετράδα** θέτοντας τους διακόπτες **1 και 2 να είναι '10'** και πιέζουμε πάλι το "load button" και τέλος με την **τέταρτη τετράδα** θέτοντας τους διακόπτες **1 και 2 να είναι '11'** και πιέζουμε ξανά το "load button". Έτσι, τέλειωσε η εισαγωγή της τιμής του πολυωνύμου δεξιάς ολίσθησης.
Με τους **διακόπτες 3 και 4 να είναι '10'** εισάγουμε την ***τιμή του πολυωνύμου αριστερής ολίσθησης*** από τους **διακόπτες 5 έως και 8**. Οι τιμές όμως είναι και εδώ των 16 bit. Άρα θα πρέπει να τις χωρίσουμε και εδώ σε 4 nibbles (τετράδες από bit). Αυτό γίνεται χρησιμοποιώντας και τους **διακόπτες 1 και 2**. Όταν **1 και 2 είναι '00'** εισάγουμε την **πρώτη τετράδα** (την δημιουργούμε στους διακόπτες 5 έως και 8). Μόλις την εισάγουμε πιέζουμε και αφήνουμε το **push button "load button"** αριστερά του διακόπτη 1 όπως φαίνεται στην εικόνα 4.13 για να εισαχθεί η τιμή στο FPGA. Συνεχίζουμε με την **δεύτερη τετράδα** θέτοντας τους διακόπτες **1 και 2 να είναι '01'** και πιέζουμε πάλι το "load button" . Κατόπιν με την **τρίτη τετράδα** θέτοντας τους διακόπτες **1 και 2 να είναι '10'** και πιέζουμε πάλι το "load button" και τέλος με την **τέταρτη τετράδα** θέτοντας τους διακόπτες **1 και 2 να είναι '11'** και πιέζουμε ξανά το "load button". Έτσι, τέλειωσε η εισαγωγή της τιμής του πολυωνύμου αριστερής ολίσθησης.
Οι τετράδες εισάγονται από την λιγότερο σημαντική (πρώτη τετράδα) μέχρι την περισσότερο σημαντική (τέταρτη τετράδα).

CS1	***CS0***	***Mode***		***Input***			
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	0	0	0	Παράλληλη Φόρτωση (seed)-Πρώτη Τετράδα			
0	1	0	0	Παράλληλη Φόρτωση (seed)-Δεύτερη Τετράδα			
1	0	0	0	Παράλληλη Φόρτωση (seed)-Τρίτη Τετράδα			
1	1	0	0	Παράλληλη Φόρτωση (seed)-Τέταρτη Τετράδα			
0	0	0	1	Πολυώνυμο Δεξιάς Ολίσθησης- Πρώτη Τετράδα			
0	1	0	1	Πολυώνυμο Δεξιάς Ολίσθησης- Δεύτερη Τετράδα			
1	0	0	1	Πολυώνυμο Δεξιάς Ολίσθησης- Τρίτη Τετράδα			
1	1	0	1	Πολυώνυμο Δεξιάς Ολίσθησης- Τέταρτη Τετράδα			
0	0	1	0	Πολυώνυμο Αριστερής Ολίσθησης- Πρώτη Τετράδα			
0	1	1	0	Πολυώνυμο Αριστερής Ολίσθησης- Δεύτερη Τετράδα			
1	0	1	0	Πολυώνυμο Αριστερής Ολίσθησης- Τρίτη Τετράδα			
1	1	1	0	Πολυώνυμο Αριστερής Ολίσθησης- Τέταρτη Τετράδα			
0	0	1	1	Κανονική Λειτουργία- Αμετάβλητη Έξοδος			
0	1	1	1	Κανονική Λειτουργία- Ολίσθηση Δεξιά			
1	0	1	1	Κανονική Λειτουργία- Ολίσθηση Αριστερά			
1	1	1	1	Κανονική Λειτουργία- Παράλληλη Φόρτωση			

Εικόνα 4.14 : Πίνακας Λειτουργιών Διακοπτών Αναπτυξιακού.

Τέλος, με τους **διακόπτες 3 και 4 να είναι '11'** είμαστε στη ***κανονική λειτουργία,*** δηλαδή εκεί που δουλεύει το LFSR. Οι διακόπτες 5 έως και 8 δεν παίζουν κανένα ρόλο εδώ καθώς δε χρειάζεται να εισάγουμε κάποια άλλη τιμή. Οι διακόπτες όμως 1 και 2 είναι σε αυτή τη περίπτωση οι είσοδοι επιλογής των πολυπλεκτών του κυκλώματος της εικόνας 3.25. Όταν **1 και 2 είναι '00'** τότε **η έξοδος του κυκλώματος παραμένει αμετάβλητη.** Όταν **1 και 2 είναι '01'** τότε το κύκλωμα κάνει

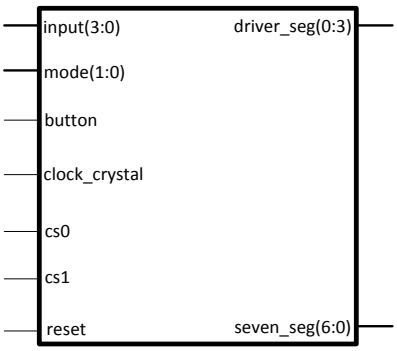
δεξιά ολίσθηση. Όταν **1 και 2 είναι '10'** τότε το κύκλωμα κάνει **αριστερή ολίσθηση** και όταν **1 και 2 είναι '11'** τότε το κύκλωμα κάνει **παράλληλη φόρτωση**. Όλες αυτές οι λειτουργίες φαίνονται στην εικόνα 4.14 .
Με το push button **reset** επανερχόμαστε σε μία αρχική κατάσταση που φαίνεται παρακάτω στο διάγραμμα καταστάσεων της σχεδίασης.

4.4. Σχεδίαση και Υλοποίηση σε κώδικα VHDL

Στην ενότητα αυτή γίνεται η σχεδίαση των κυκλωμάτων που απαιτούνται και η υλοποίηση τους στη γλώσσα περιγραφής υλικού VHDL.

4.4.1. Σχεδίαση της εφαρμογής

Το block διάγραμμα της τελικής (συνολικής) σχεδίασης φαίνεται στην εικόνα 4.15.

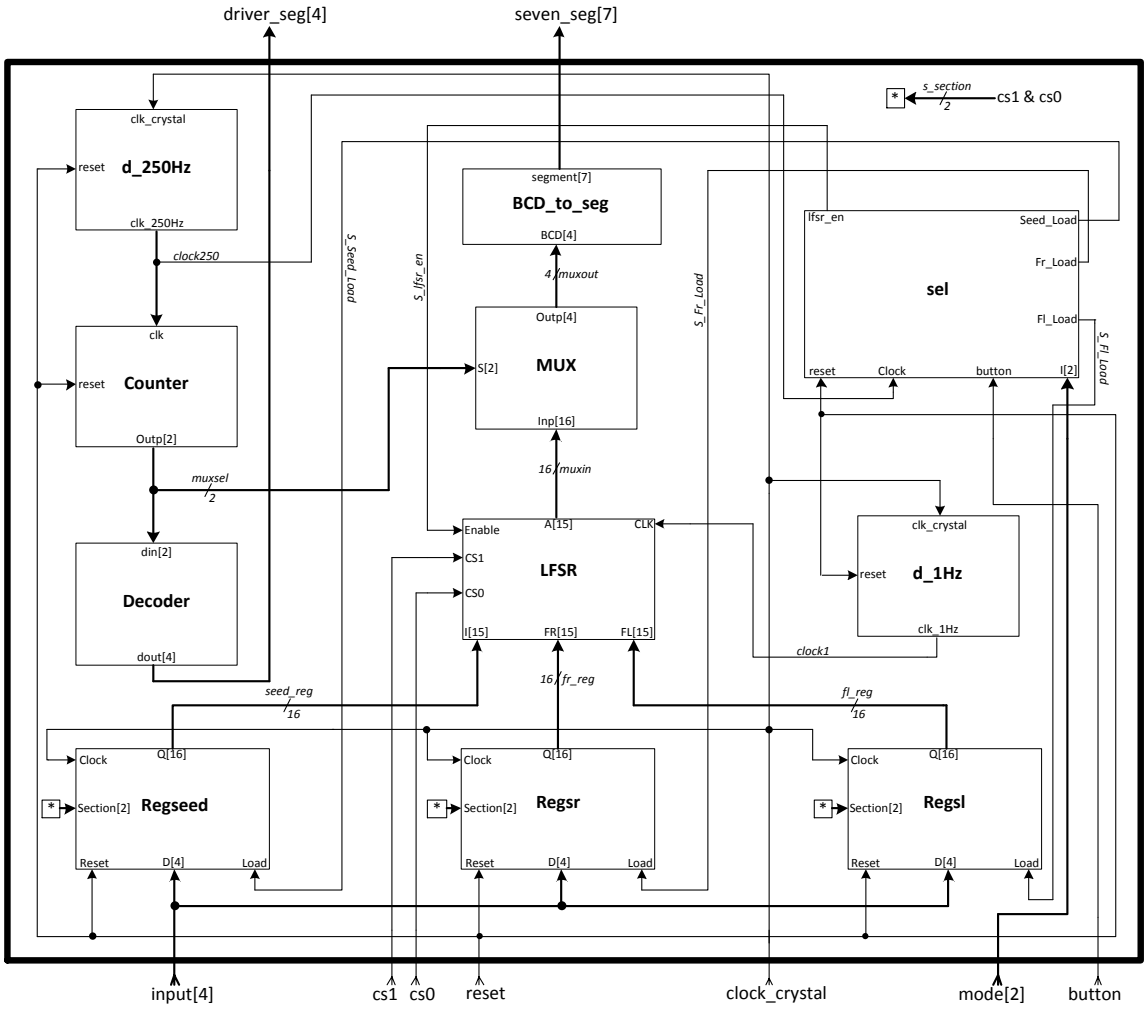


Εικόνα 4.15 : Block διάγραμμα συνολικής σχεδίασης.

Οι είσοδοι, οι έξοδοι και οι λειτουργίες αυτών φαίνονται στην εικόνα 4.16 και το εσωτερικό του παραπάνω block διαγράμματος στην εικόνα 4.17 .

Όνομα	*Κατεύθυνση*	*Ερμηνεία Λειτουργίας*
input (vector of 4 signals)	Input	Είσοδος δεδομένων από τους διακόπτες 5 έως και 8
mode (vector of 2 signals)	Input	Έλεγχος Λειτουργίας από τους διακόπτες 3 και 4
driver_seg (vector of 4 signals)	Output	Έξοδοι Πολύπλεξης των 7-segment display
seven_seg (vector of 7 signals)	Output	Έξοδοι Οδήγησης των 7-segment display
button	Input	Φόρτωση δεδομένων (Load)
clock_crystal	Input	Καλώδιο τροφοδότησης ρολογιού
cs0	Input	Είσοδος επιλογής πολυπλέκτη και εισαγωγής δεδομένων από τον διακόπτη 2
cs1	Input	Είσοδος επιλογής πολυπλέκτη και εισαγωγής δεδομένων από τον διακόπτη 1
reset	Input	Καλώδιο κεντρικής επανεκκίνησης

Εικόνα 4.16 : Πίνακας εισόδων, εξόδων και των λειτουργιών αυτών της τελικής σχεδίασης.



Εικόνα 4.17 : Εσωτερικό του block διαγράμματος της συνολικής σχεδίασης.

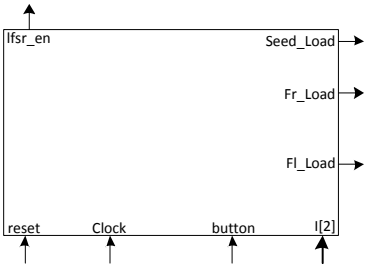
Όπως φαίνεται και στην εικόνα, στο εσωτερικό του block της συνολικής σχεδίασης, υπάρχουν τα παρακάτω components:

- **Regseed**, **Regsr** και **Regsl**. Οι μονάδες αυτές είναι καταχωρητές και αποθηκεύουν τις τιμές της εισόδου input. Στην είσοδο section που είναι 2 bit συνδέονται οι είσοδοι cs1 και cs0 (Το * τοποθετείται εκεί για να συμβολίσει αυτό το σήμα) .
- **LFSR**. Η μονάδα αυτή είναι το κύκλωμα της εικόνας 3.25.
- **MUX**. Η μονάδα αυτή είναι ένας πολυπλέκτης 16 σε 4. Διαχωρίζει τα 4 nibble της λέξης των 16 bit που λαμβάνει από την έξοδο του LFSR.
- **BCD_to_seg**. Η μονάδα αυτή μετατρέπει τη λέξη εισόδου μήκους 4 bit που συμβολίζει ένα δεκαεξαδικό αριθμό για να οδηγήσει τα 7-segment display.
- **Decoder**. Η μονάδα αυτή είναι ένας κωδικοποιητής 2 σε 4. Παίρνει είσοδο από το μετρητή Counter μήκους 2 και βγάζει διαδοχικά τις τιμές ‘0111’-‘1011’-‘1101’-‘1110’. Η έξοδος του οδηγεί τις εισόδους πολύπλεξης των 7-segment display για την ενεργοποίηση τους.
- **Counter**. Η μονάδα αυτή είναι ένας δυαδικός μετρητής που στην έξοδο του βγάζει τις τιμές ‘00’-‘01’-‘10’-‘11’
- **d_250Hz, d_1Hz**. Οι μονάδες αυτές είναι διαιρέτες συχνότητας. Παίρνουν είσοδο από το κρύσταλλο του αναπτυξιακού, συχνότητας 50 MHz, και στις εξόδους τους βγάζουν 250Hz ο πρώτος και 1 Hz ο δεύτερος.

- **sel**. Η μονάδα αυτή είναι η μονάδα ελέγχου του συστήματος (FSM) που περιγράφετε στις επόμενες σελίδες με επεξήγηση βαθμίδας και παρουσίαση των λειτουργιών σε διάγραμμα καταστάσεων. Αυτή η μονάδα ελέγχει τους καταχωρητές του κυκλώματος και ενεργοποιεί το LFSR σύμφωνα με τους διακόπτες 3 και 4.

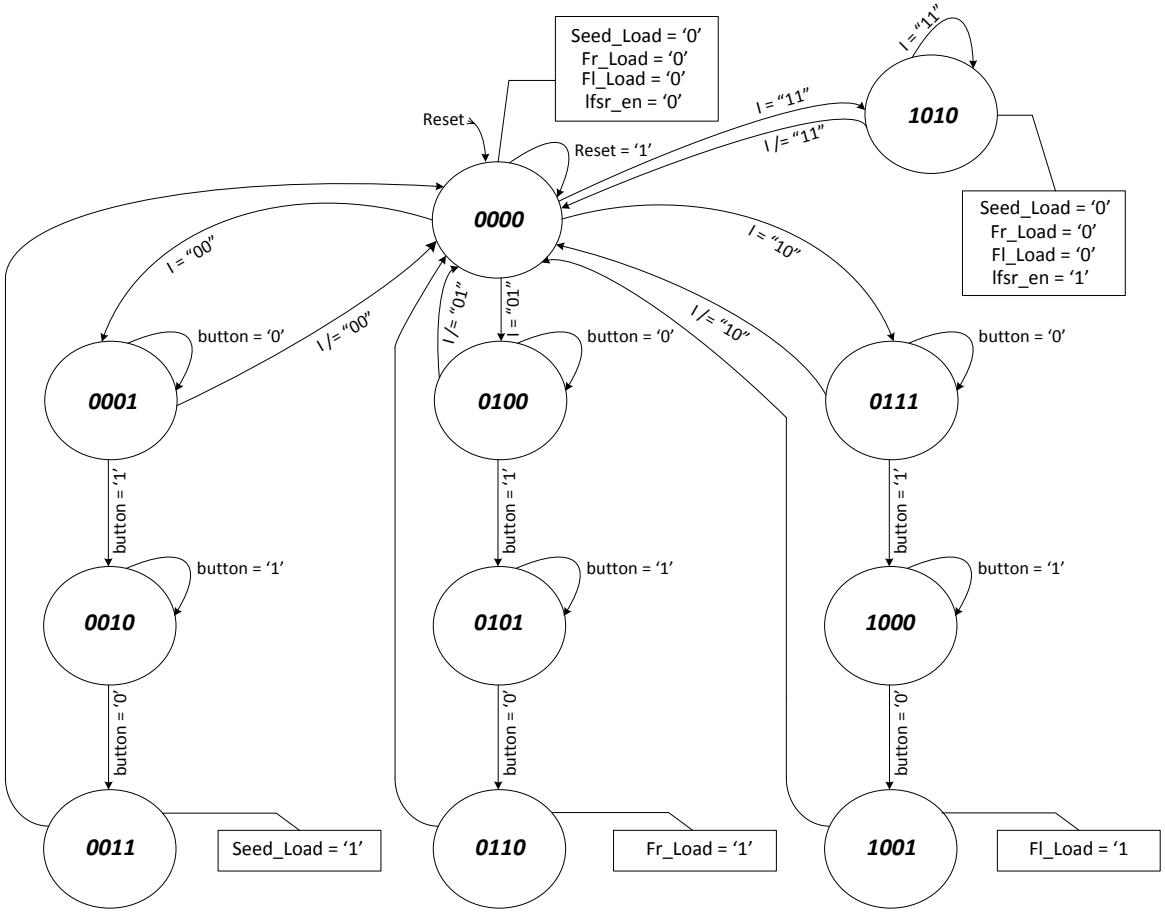
4.4.2. Η Μονάδα Ελέγχου

Για να λειτουργήσει το όλο κύκλωμα, χρειάζεται μια μονάδα, στην οποία θα γίνεται η διαχείριση όλων των σημάτων του. Αυτή η μονάδα ονομάζεται μονάδα ελέγχου και το μόνο που κάνει είναι να ελέγχει κάποιες από τις εισόδους των υπόλοιπων εξαρτημάτων ή εξωτερικών εισόδων του κυκλώματος και να βγάζει τα ανάλογα σήματα ελέγχου. Η μονάδα ελέγχου που υλοποιήθηκε παίρνει ένα σήμα με την κωδικοποίηση της εντολής που πρέπει να εκτελέσει. Συγκεκριμένα η κωδικοποίηση γίνεται με 4 ψηφία ξεκινώντας με το ‘0000’ και καταλήγοντας στο ‘1010’. Το block διάγραμμα της μονάδας φαίνεται στην εικόνα 4.18.



Εικόνα 4.18 : Block διάγραμμα μονάδας ελέγχου.

Το διάγραμμα καταστάσεων της μονάδας ελέγχου φαίνεται στην εικόνα 4.19 .



Εικόνα 4.19 : Διάγραμμα καταστάσεων της μονάδας ελέγχου.

Η μονάδα ελέγχου που υλοποιήθηκε έχει τις εξής εισόδους και εξόδους:

Όνομα	*Κατεύθυνση*	*Ερμηνεία Λειτουργίας*
I (vector of 2 signals)	Input	Είσοδος δεδομένων. Διακόπτες 3 και 4.
button	Input	Είσοδος ελέγχου φόρτωσης δεδομένων (Load)
Clock	Input	Είσοδος ρολογιού (250 Hz)
reset	Input	Είσοδος επανεκκίνησης
Seed_Load	Output	Έξοδος ενεργοποίησης του καταχωρητή που αποθηκεύει την τιμή της παράλληλης φόρτωσης
Fr_Load	Output	Έξοδος ενεργοποίησης του καταχωρητή που αποθηκεύει την τιμή του πολυωνύμου δεξιάς ολίσθησης
Fl_Load	Output	Έξοδος ενεργοποίησης του καταχωρητή που αποθηκεύει την τιμή του πολυωνύμου αριστερής ολίσθησης
lfsr_en	Output	Έξοδος ενεργοποίησης της μονάδας του LFSR

Εικόνα 4.20 : Πίνακας εισόδων, εξόδων και των λειτουργιών αυτών της μονάδας ελέγχου.

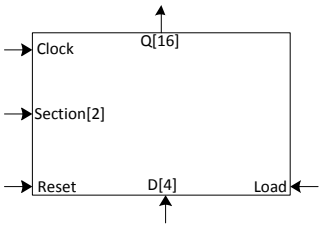
Ξεκινώντας με την ενεργοποίηση του αναπτυξιακού, η μονάδα ελέγχου έρχεται στην κατάσταση **‘0000’**. Στην κατάσταση αυτή μηδενίζονται όλα τα σήματα (signals) που διασύνδεουν τη μονάδα ελέγχου με τους καταχωρητές Regseed, Regsr και Regsl. Δηλαδή δεν μπορούν να πάρουν καμία τιμή οι καταχωρητές αφού η είσοδος τους “Load” είναι ‘0’. Επίσης απενεργοποιείται η μονάδα του LFSR. Όταν το **Reset** είναι **‘1’** τότε **παραμένουμε στην ίδια κατάσταση**. Όταν το **Reset** είναι **‘0’ ελέγχουμε την είσοδο I** για να δούμε σε ποια κατάσταση θα οδηγηθούμε. Αν το **I** είναι **‘00’** οδηγούμαστε στην κατάσταση **‘0001’**, αν είναι **‘01’** οδηγούμαστε στην κατάσταση **‘0100’**, αν είναι **‘10’** οδηγούμαστε στην κατάσταση **‘0111’** και αν είναι **‘11’** οδηγούμαστε στην κατάσταση **‘1010’.** Η είσοδος I εκπροσωπεί τους διακόπτες 3 και 4 του αναπτυξιακού όπως ειπώθηκε και παραπάνω. Στην κατάσταση **‘0001’** γίνεται έλεγχος του button. Αν η είσοδος **button** είναι **‘0’** (δεν πιέστηκε δηλαδή) τότε **παραμένουμε στην ίδια κατάσταση**. Αν είναι **‘1’** (πιέστηκε δηλαδή) **οδηγούμαστε στην κατάσταση ‘0010’**.Αν το **I** είναι **διαφορετικό** του **‘00’ επιστρέφουμε στην αρχική κατάσταση ‘0000’** για να μην κολλήσουμε στην παρούσα κατάσταση. Στην κατάσταση ‘0010’ γίνεται ξανά έλεγχος του button μόνο που όταν είναι ‘1’ παραμένουμε στην ίδια κατάσταση και όταν γίνει ‘0’ μεταφερόμαστε στην κατάσταση ‘0011’ όπου η έξοδος Seed_Load γίνεται ‘1’ ,ενεργοποιείται ο καταχωρητής **Regseed** για να λάβει την τιμή παράλληλης φόρτωσης και οδηγούμαστε πίσω στην αρχική κατάσταση ‘0000’. Αυτή **η διαδικασία ,με το button, έγινε επειδή θέλουμε να προχωράει στην επόμενη κατάσταση μόνο όταν το πιέσουμε (button = ‘1’) και το αφήσουμε (button = ‘0’) και όχι όταν το πιέζουμε παρατεταμένα.** Στις καταστάσεις **‘0100’-‘0101’-‘0110’** γίνονται ακριβώς τα ίδια μόνο που στην κατάσταση **‘0110’** η έξοδος **Fr_Load** γίνεται **‘1’** ,ενεργοποιείται ο καταχωρητής **Regsr** για να λάβει την τιμή του πολυωνύμου δεξιάς ολίσθησης και οδηγούμαστε πίσω στην αρχική κατάσταση ‘0000’. Στην κατάσταση **‘0100’** αν το **I** είναι **διαφορετικό** του **‘01’ επιστρέφουμε στην αρχική κατάσταση ‘0000’** για να μην κολλήσουμε στην παρούσα κατάσταση. Επίσης, στις καταστάσεις **‘0111’-‘1000’-‘1001’** γίνονται ακριβώς τα ίδια μόνο που στην κατάσταση **‘1001’** η έξοδος **Fl_Load** γίνεται **‘1’** ,ενεργοποιείται ο καταχωρητής **Regsl** για να λάβει την τιμή του πολυωνύμου αριστερής ολίσθησης και οδηγούμαστε πίσω στην αρχική κατάσταση ‘0000’. Στην κατάσταση **‘0111’** αν το **I** είναι **διαφορετικό** του **‘10’ επιστρέφουμε στην αρχική κατάσταση ‘0000’** για να μην κολλήσουμε στην παρούσα κατάσταση. Τέλος στην κατάσταση **‘1010’**, η έξοδος **lfsr_en** γίνεται **‘1’** και **ενεργοποιείται η μονάδα του LFSR**. Οι υπόλοιπες έξοδοι, **Seed_Load, Fr_Load και Fl_Load** γίνονται **‘0’** ώστε να μην επηρεάζουν τη λειτουργία των καταχωρητών την ώρα που λειτουργεί η μονάδα του LFSR. Στην κατάσταση **‘1010’** αν το **I** είναι **διαφορετικό** του **‘11’ επιστρέφουμε στην αρχική κατάσταση**

‘0000’ για να μην κολλήσουμε στην παρούσα κατάσταση και εάν είναι **‘11’** παραμένουμε σε αυτήν για να δουλεύει η μονάδα του LFSR.

4.4.3. Υλοποίηση σε κώδικα VHDL

4.4.3.1. Καταχωρητής 16 bit

Το μπλοκ διάγραμμα του καταχωρητή 16 bit είναι:



Εικόνα 4.21 : Block διάγραμμα καταχωρητή 16 bit.

Το εξάρτημα αυτό χρησιμοποιείται τρεις φορές στη σχεδίαση στους καταχωρητές **Regseed, Regsr** και. **Regsl.** Όταν το reset είναι ‘1’ μηδενίζεται η δεκαεξάμπιτη έξοδος Q του καταχωρητή. Όταν είναι το Reset ‘0’ και το Load ‘1’ (το Load ενεργοποιείται από τη μονάδα ελέγχου) και έλθει κατάλληλο ρολόι, ελέγχει το σήμα Section για τις τιμές μου λαμβάνει (‘00’-‘11’) και φορτώνει τον καταχωρητή με τετράμπιτες τιμές.

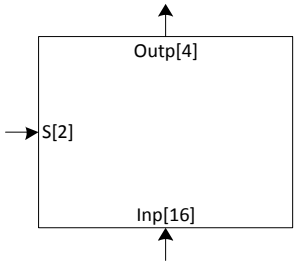
Ο κώδικας του είναι:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Register_16_bit is
Port ( D : in  STD_LOGIC_VECTOR (3 downto 0);
Reset : in  STD_LOGIC;
Load : in  STD_LOGIC;
Clock : in  STD_LOGIC;
Section: in STD_LOGIC_VECTOR (1 downto 0);
Q : out  STD_LOGIC_VECTOR (15 downto 0));
end Register_16_bit;

architecture Behavioral of Register_16_bit is

Begin
Process (Reset,Clock,Load)
Begin
  If Reset= '1' then
    Q <= "0000000000000000";
  Elsif Clock' EVENT and Clock = '0' and Load = '1'  then
    case Section is
      when "00" => Q(3 downto 0) <= D;
      when "01" => Q(7 downto 4) <= D;
      when "10" => Q(11 downto 8) <= D;
      when others => Q (15 downto 12) <= D;
    end case;
  End If;
End Process;
End Behavioral;
```

Εικόνα 4.22 : Κώδικας του καταχωρητή 16 bit.

4.4.3.2. Πολυπλέκτης 16 σε 4

Το μπλοκ διάγραμμα του πολυπλέκτη 16 σε 1 είναι:



Εικόνα 4.23 : Block διάγραμμα του πολυπλέκτη 16 σε 1.

Το εξάρτημα αυτό χρησιμοποιείται μία φορά στη σχεδίαση με το όνομα **MUX**.Ο πολυπλέκτης αυτός δέχεται στην είσοδο του τη δεκαεξάμπιτη έξοδο από τη μονάδα του LFSR και ανάλογα την είσοδο επιλογής του, μήκους 2 bits, βγάζει στην τετράμπιτη έξοδο του τμήμα της δεκαεξάμπιτης λέξης εισόδου.

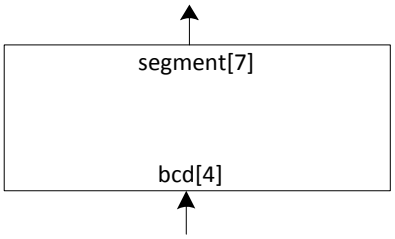
Ο κώδικας του είναι:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity multiplexer_16to4 is
    Port ( Inp : in  STD_LOGIC_VECTOR (15 downto 0);
           S : in  STD_LOGIC_VECTOR (1 downto 0);
           Outp : out  STD_LOGIC_VECTOR (3 downto 0));
end multiplexer_16to4;

architecture Behavioral of multiplexer_16to4 is

begin
process(Inp,S)
begin
   If S="00" then
      Outp<=Inp(3 downto 0);
   Elsif S="01" then
      Outp<=Inp(7 downto 4);
   Elsif S="10" then
      Outp<=Inp(11 downto 8);
   Else
      Outp<=Inp(15 downto 12);
   End If;
End process;
End Behavioral;
```

Εικόνα 4.24 : Κώδικας του πολυπλέκτη16 σε 4.

4.4.3.3. Μετατροπέας Hex σε 7-segment

Το μπλοκ διάγραμμα του μετατροπέα είναι:



Εικόνα 4.25 : Block διάγραμμα του μετατροπέα Hex σε 7-segment.

Το εξάρτημα αυτό χρησιμοποιείται μία φορά στη σχεδίαση με το όνομα **BCD_to_seg.** Ο μετατροπέας αυτός μετατρέπει τον κάθε συνδυασμό της εισόδου bcd που συμβολίζει αριθμούς του δεκαεξαδικού στην ανάλογη λέξη των 7 bits και την παρέχει στην έξοδο segment για την οδήγηση των 7-seg display.

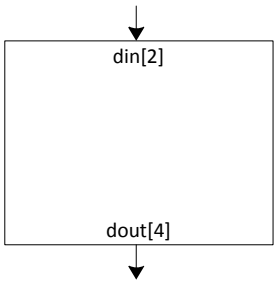
Ο κώδικας του είναι:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity BCD_to_7_segment_display_converter is
    Port ( bcd : in  STD_LOGIC_VECTOR (3 downto 0);
           segment : out  STD_LOGIC_VECTOR (0 to 6));
end BCD_to_7_segment_display_converter;

architecture Behavioral of BCD_to_7_segment_display_converter is

begin
process (bcd)
begin
            case  bcd is
            when "0000"=> segment(0 to 6) <="0000001";  -- '0'
            when "0001"=> segment(0 to 6) <="1001111";  -- '1'
            when "0010"=> segment(0 to 6) <="0010010";  -- '2'
            when "0011"=> segment(0 to 6) <="0000110";  -- '3'
            when "0100"=> segment(0 to 6) <="1001100";  -- '4'
            when "0101"=> segment(0 to 6) <="0100100";  -- '5'
            when "0110"=> segment(0 to 6) <="0100000";  -- '6'
            when "0111"=> segment(0 to 6) <="0001111";  -- '7'
            when "1000"=> segment(0 to 6) <="0000000";  -- '8'
            when "1001"=> segment(0 to 6) <="0000100";  -- '9'
            when "1010"=> segment(0 to 6) <="0001000";  -- 'A'
            when "1011"=> segment(0 to 6) <="1100000";  -- 'b'
            when "1100"=> segment(0 to 6) <="0110001";  -- 'C'
            when "1101"=> segment(0 to 6) <="1000010";  -- 'd'
            when "1110"=> segment(0 to 6) <="0110000";  -- 'E'
            when "1111"=> segment(0 to 6) <="0111000";  -- 'F'
            when others=> segment(0 to 6) <="1111111";  -- others
            End case;
End process;
End Behavioral;
```

Εικόνα 4.26 : Κώδικας του μετατροπέα Hex σε 7-segment.

4.4.3.4. Κωδικοποιητής 2 σε 4

Το μπλοκ διάγραμμα του κωδικοποιητή 2 σε 4 είναι:



Εικόνα 4.27 : Block διάγραμμα του κωδικοποιητή 2 σε 4.

Το εξάρτημα αυτό χρησιμοποιείται μία φορά στη σχεδίαση με το όνομα **Decoder**.Ο κωδικοποιητής αυτός δέχεται στην είσοδο του, din, την 2 bit έξοδο από τη μονάδα του μετρητή (counter) και βγάζει στην τετράμπιτη έξοδο του διαδοχικά τις τιμές ‘0111’-‘1011’-1101’-‘1110’ για να οδηγήσει τις εισόδους πολύπλεξης των 7-segment display.

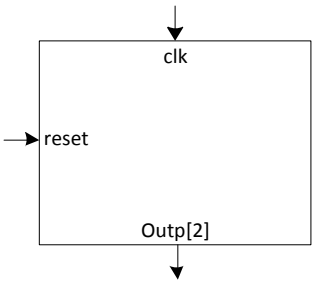
Ο κώδικας του είναι:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Decoder_2to4 is
    Port ( din : in  STD_LOGIC_VECTOR (1 downto 0);
           dout : out  STD_LOGIC_VECTOR (0 to 3));
end Decoder_2to4;

architecture Behavioral of Decoder_2to4 is

begin
process(din)
begin
     case din is
     when "00"=> dout <="0111";
     when "01"=> dout <="1011";
     when "10"=> dout <="1101";
     when "11"=> dout <="1110";
     when others=> dout <="1111";
     End case;
End process;
End Behavioral;
```

Εικόνα 4.28 : Κώδικας του κωδικοποιητή 2 σε 4.

4.4.3.5. Δυαδικός μετρητής 2 bit

Το μπλοκ διάγραμμα του δυαδικού μετρητή 2 bit είναι:



Εικόνα 4.29 : Block διάγραμμα του δυαδικού μετρητή 2 bit.

Το εξάρτημα αυτό χρησιμοποιείται μία φορά στη σχεδίαση με το όνομα **Counter**.Ο μετρητής αυτός δέχεται στην είσοδο του, clk, το ρολόι από τη μονάδα του διαιρέτη συχνότητας των 250 Hz (d_250Hz) και βγάζει στην έξοδο του διαδοχικά τις τιμές ‘00’-‘01’-10’-‘11’ για να οδηγήσει την είσοδος του Decoder και τις εισόδους επιλογής του MUX.

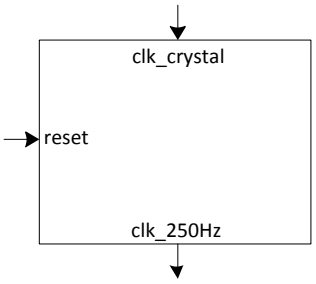
Ο κώδικας του είναι:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity counter_2_bit is
Port ( reset : in  STD_LOGIC;
       clk : in  STD_LOGIC;
       Outp : out  STD_LOGIC_VECTOR (1 downto 0));
end counter_2_bit;

architecture Behavioral of counter_2_bit is

signal temp : std_logic_vector(1 downto 0);
begin
process(clk,reset)
begin
If reset='1' then
    temp<= "00";
Else
     If(clk' event and clk='1') then
          temp<=temp + "1";
     End if;
End if;
End process;
Outp<=temp;
End Behavioral;
```

Εικόνα 4.30 : Κώδικας του δυαδικού μετρητή 2 bit.

4.4.3.6. Διαιρέτης Συχνότητας 250 Hz

Το μπλοκ διάγραμμα του διαιρέτη συχνότητας 250 Hz είναι:



Εικόνα 4.31 : Block διάγραμμα του διαιρέτη συχνότητας 250 Hz.

Το εξάρτημα αυτό χρησιμοποιείται μία φορά στη σχεδίαση με το όνομα **d_250Hz**. Η βαθμίδα αυτή δέχεται στην είσοδο clk_crystal τα 50 MHz από το κρύσταλλο που υπάρχει πάνω στο αναπτυξιακό και παράγει ένα σήμα συχνότητας 250 Hz στην έξοδο clk_250Hz το οποίο τροφοδοτεί το δυαδικό μετρητή 2 bit (Counter) και την μονάδα ελέγχου (sel).

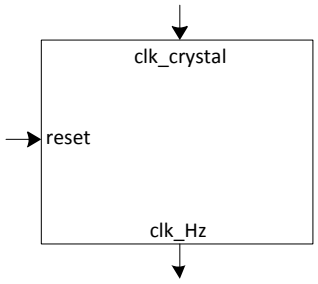
Ο κώδικας του είναι:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
use IEEE.numeric_std.ALL;

entity Clock_250Hz is
Port ( clk_crystal : in  STD_LOGIC;
       reset : in  STD_LOGIC;
       clk_250Hz : out  STD_LOGIC);
end Clock_250Hz;

architecture Behavioral of Clock_250Hz is

signal temp: std_logic_vector(17 downto 0);
signal temp_clk : std_logic;
begin
process(reset,clk_crystal,temp_clk)
begin
   If reset='1' then
       temp<="000000000000000000";
       temp_clk<='0';
   Else
      If(clk_crystal' event and clk_crystal='1') then
          temp <= temp + 1;
          If temp = 100000 then --50000000Hz/250Hz=200000/2=100000
             temp_clk <= not temp_clk;
          End if;
          If temp = 200000 then
             temp<="000000000000000000";
          End if;
      End if;
   End if;
clk_250hz <= temp_clk;
End process;
End Behavioral;
```

Εικόνα 4.32 : Κώδικας του διαιρέτη συχνότητας 250 Hz.

4.4.3.7. Διαιρέτης Συχνότητας 1 Hz

Το μπλοκ διάγραμμα του διαιρέτη συχνότητας 1 Hz είναι:



Εικόνα 4.33 : Block διάγραμμα του διαιρέτη συχνότητας 1 Hz.

Το εξάρτημα αυτό χρησιμοποιείται μία φορά στη σχεδίαση με το όνομα **d_1Hz**. Η βαθμίδα αυτή δέχεται στην είσοδο clk_crystal τα 50 MHz από το κρύσταλλο που υπάρχει πάνω στο αναπτυξιακό και παράγει ένα σήμα συχνότητας 1 Hz, στην έξοδο clk_1Hz, το οποίο τροφοδοτεί τη μονάδα του LFSR.

Ο κώδικας του είναι:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
use IEEE.numeric_std.ALL;

entity Clock_1Hz is
Port ( clk_crystal : in  STD_LOGIC;
       reset : in  STD_LOGIC;
       clk_1Hz : out  STD_LOGIC);
end Clock_1Hz;

architecture Behavioral of Clock_1Hz is

signal temp_clk : std_logic;
signal temp: std_logic_vector(25 downto 0);
begin
process(reset,clk_crystal,temp_clk)
begin
   If reset='1'  then
       temp <= "00000000000000000000000000";
       temp_clk<='0';
   Else
      If(clk_crystal' event and clk_crystal='1') then
           temp <= temp + 1;
           if temp = 25000000 then --50000000Hz/1Hz=50000000/2=25000000
              temp_clk <= not temp_clk;
           End if;
           If temp = 50000000 then
              temp <="00000000000000000000000000";
           End if;
       End if;
   End if;
clk_1hz <= temp_clk;
end process;
end Behavioral;
```

Εικόνα 4.34 : Κώδικας του διαιρέτη συχνότητας 1 Hz.

4.4.3.8. Η Μονάδα Ελέγχου

Η μονάδα ελέγχου επεξηγήθηκε στην ενότητα 4.4.2.Ο κώδικας που την υλοποιεί είναι:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Selection is
Port ( I : in  STD_LOGIC_VECTOR (1 downto 0);
       reset : in  STD_LOGIC;
       clock : in  STD_LOGIC;
       button : in  STD_LOGIC;
       Seed_Load : out  STD_LOGIC;
       lfsr_en : out  STD_LOGIC;
       Fr_Load : out  STD_LOGIC;
       Fl_Load : out  STD_LOGIC);
end Selection;

architecture Behavioral of Selection is

signal state: std_logic_vector(3 downto 0);
begin
process(reset,clock)
begin
If reset='1'  then state <= "0000";
              Seed_Load <= '0';
               Fr_Load <= '0';
               Fl_Load <= '0';
               lfsr_en <= '0';
elsif clock' event and clock = '1' then

               case state is

               when "0000" =>

               Seed_Load <= '0';
                Fr_Load <= '0';
                Fl_Load <= '0';
                 lfsr_en <= '0';

         If    I= "00" then state <= "0001";
         Elsif I= "01" then state <= "0100";
         Elsif I= "10" then state <= "0111";
         Else  state<= "1010";
         End If;

                when "0001" =>

         If button = '0' then state <= "0001";
         Else state <= "0010";
         End if;
         If I /= "00" then state <= "0000";
         End If;

                when "0010" =>

         If button = '1' then state <= "0010";
         Else state <= "0011";
         End if;

                when "0011" =>

               Seed_Load <= '1';
               state <= "0000";

                when "0100" =>

        If button = '0' then state <= "0100";
        Else state <= "0101";
         End If;
         If I /= "01" then state <= "0000";
         End if;

           when "0101" =>

        If button = '1' then state <= "0101";
        Else state <= "0110";
        End If;

           when "0110" =>

         Fr_Load <= '1';
         state <= "0000";

           when "0111" =>

        If button = '0' then state <= "0111";
        Else state <= "1000";
        End If;
        If I /= "10" then state <= "0000";
        End If;

           when "1000" =>

        If button = '1' then state <= "1000";
        Else state <= "1001";
        End If;

           when "1001" =>

           Fl_Load <= '1';
           state <= "0000";

           when "1010" =>

            lfsr_en <= '1';
            Fr_Load <= '0';
           Fl_Load <= '0';
           Seed_Load <= '0';
           If I= "11" then state <= "1010";
           Else state <= "0000";
           End If;

           when others => state <= "0000";

           End case;

End If;
End process;
End Behavioral;
```

Εικόνα 4.35 : Κώδικας της μονάδας ελέγχου.

4.4.3.9. Η Συνολική Σχεδίαση

Το block διάγραμμα βρίσκεται στην εικόνα 4.15. Στην εικόνα 4.17 φαίνεται το εσωτερικό του διαγράμματος το οποίο είναι και το top module. Ο κώδικας που το υλοποιεί είναι:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Sinoliki_Sxediasi is
Port ( clock_crystal : IN  STD_LOGIC;
       cs0 : IN STD_LOGIC;
       cs1 : IN STD_LOGIC;
     input : IN STD_LOGIC_vector (3 downto 0);
     button : IN STD_LOGIC;
     mode : IN STD_LOGIC_vector (1 downto 0);
     reset : IN STD_LOGIC;
seven_seg : OUT STD_LOGIC_VECTOR (6 downto 0);
driver_seg : OUT STD_LOGIC_VECTOR (0 to 3));
end Sinoliki_Sxediasi;

architecture Behavioral of Sinoliki_Sxediasi
is

component LFSR_16bit
 port (   A0     :     OUT     STD_LOGIC;
          A1     :     OUT     STD_LOGIC;
          A2     :     OUT     STD_LOGIC;
          A3     :     OUT     STD_LOGIC;
          A4     :     OUT     STD_LOGIC;
          A5     :     OUT     STD_LOGIC;
          A6     :     OUT     STD_LOGIC;
          A7     :     OUT     STD_LOGIC;
          A8     :     OUT     STD_LOGIC;
          A9     :     OUT     STD_LOGIC;
          A10    :     OUT     STD_LOGIC;
          A11    :     OUT     STD_LOGIC;
          A12    :     OUT     STD_LOGIC;
          A13    :     OUT     STD_LOGIC;
          A14    :     OUT     STD_LOGIC;
          A15    :     OUT     STD_LOGIC;
          CLK    :     IN      STD_LOGIC;
          Enable :     IN      STD_LOGIC;
          CS1    :     IN      STD_LOGIC;
          CS0    :     IN      STD_LOGIC;
          I15    :     IN      STD_LOGIC;
          I14    :     IN      STD_LOGIC;
          I13    :     IN      STD_LOGIC;
          I12    :     IN      STD_LOGIC;
          I11    :     IN      STD_LOGIC;
          I10    :     IN      STD_LOGIC;
          I9     :     IN      STD_LOGIC;
          I8     :     IN      STD_LOGIC;
          I7     :     IN      STD_LOGIC;
          I6     :     IN      STD_LOGIC;
          I5     :     IN      STD_LOGIC;
          I4     :     IN      STD_LOGIC;
          I3     :     IN      STD_LOGIC;
          I2     :     IN      STD_LOGIC;
          I1     :     IN      STD_LOGIC;
          I0     :     IN      STD_LOGIC;
          FR0    :     IN      STD_LOGIC;
          FR1    :     IN      STD_LOGIC;
          FR2    :     IN      STD_LOGIC;
          FR3    :     IN      STD_LOGIC;
          FR4    :     IN      STD_LOGIC;
          FR5    :     IN      STD_LOGIC;
          FR6    :     IN      STD_LOGIC;
          FR7    :     IN      STD_LOGIC;
          FR8    :     IN      STD_LOGIC;
          FR9    :     IN      STD_LOGIC;
         FR10    :     IN      STD_LOGIC;
          FR11   :     IN      STD_LOGIC;
          FR12   :     IN      STD_LOGIC;
          FR13   :     IN      STD_LOGIC;
          FR14   :     IN      STD_LOGIC;
          FR15   :     IN      STD_LOGIC;
          FL0    :     IN      STD_LOGIC;
          FL1    :     IN      STD_LOGIC;
          FL2    :     IN      STD_LOGIC;
          FL3    :     IN      STD_LOGIC;
          FL4    :     IN      STD_LOGIC;
          FL5    :     IN      STD_LOGIC;
          FL6    :     IN      STD_LOGIC;
          FL7    :     IN      STD_LOGIC;
          FL8    :     IN      STD_LOGIC;
          FL9    :     IN      STD_LOGIC;
          FL10   :     IN      STD_LOGIC;
          FL11   :     IN      STD_LOGIC;
          FL12   :     IN      STD_LOGIC;
          FL13   :     IN      STD_LOGIC;
          FL14   :     IN      STD_LOGIC;
          FL15   :     IN      STD_LOGIC);
end component;

component Selection
port ( I : IN  STD_LOGIC_VECTOR (1 downto 0);
       reset : IN  STD_LOGIC;
       clock : IN  STD_LOGIC;
       button : IN  STD_LOGIC;
       Seed_Load : OUT  STD_LOGIC;
       lfsr_en : OUT  STD_LOGIC;
       Fr_Load : OUT  STD_LOGIC;
       Fl_Load : OUT  STD_LOGIC);
end component;

component multiplexer_16to4
port ( Inp: IN STD_LOGIC_VECTOR (15 downto 0);
       S : IN STD_LOGIC_VECTOR (1 downto 0);
   Outp : OUT STD_LOGIC_VECTOR (3 downto 0));
end component;

component counter_2_bit
 port ( reset : IN STD_LOGIC;
        clk : IN STD_LOGIC;
    Outp : OUT STD_LOGIC_VECTOR (1 downto 0));
end component;

component BCD_to_7_segment_display_converter
port ( bcd : IN STD_LOGIC_VECTOR (3 downto 0);
     segment : OUT STD_LOGIC_VECTOR (0 to 6));
end component;

component Register_16_bit
port ( D : IN STD_LOGIC_VECTOR (3 downto 0);
       Reset : IN STD_LOGIC;
       Load : IN STD_LOGIC;
       Clock : IN STD_LOGIC;
    Section: IN STD_LOGIC_VECTOR (1 downto 0);
      Q : OUT STD_LOGIC_VECTOR (15 downto 0));
end component;

component Clock_250Hz
port ( clk_crystal : IN  STD_LOGIC;
       reset : IN  STD_LOGIC;
       clk_250Hz : OUT  STD_LOGIC);
end component;
```

```
component Clock_1Hz
port ( clk_crystal : IN STD_LOGIC;
       reset       : IN STD_LOGIC;
       clk_1Hz     : OUT STD_LOGIC);
end component;

component Decoder_2to4
port ( din : IN STD_LOGIC_VECTOR (1 downto 0);
       dout : OUT STD_LOGIC_VECTOR (0 to 3));
end component;

signal muxin, seed_reg, fr_reg, fl_reg :
std_logic_vector(15 downto 0);
signal muxout : std_logic_vector(3 downto 0);
signal muxsel, s_section : std_logic_vector(1
downto 0);
signal clock250, clock1, S_Seed_Load,
S_lfsr_en, S_Fr_Load, S_Fl_Load : std_logic;

begin

LFSR : LFSR_16bit
port map(
         A0 => muxin(0),
         A1 => muxin(1),
         A2 => muxin(2),
         A3 => muxin(3),
         A4 => muxin(4),
         A5 => muxin(5),
         A6 => muxin(6),
         A7 => muxin(7),
         A8 => muxin(8),
         A9 => muxin(9),
         A10 => muxin(10),
         A11 => muxin(11),
         A12 => muxin(12),
         A13 => muxin(13),
         A14 => muxin(14),
         A15 => muxin(15)
         CLK => clock1,
         Enable => S_lfsr_en,
         CS1 => cs1,
         CS0 => cs0,
         I15 => seed_reg(15),
         I14 => seed_reg(14),
         I13 => seed_reg(13),
         I12 => seed_reg(12),
         I11 => seed_reg(11),
         I10 => seed_reg(10),
         I9 => seed_reg(9),
         I8 => seed_reg(8),
         I7 => seed_reg(7),
         I6 => seed_reg(6),
         I5 => seed_reg(5),
         I4 => seed_reg(4),
         I3 => seed_reg(3),
         I2 => seed_reg(2),
         I1 => seed_reg(1),
         I0 => seed_reg(0),
         FR0 => fr_reg(0),
         FR1 => fr_reg(1),
         FR2 => fr_reg(2),
         FR3 => fr_reg(3),
         FR4 => fr_reg(4),
         FR5 => fr_reg(5),
         FR6 => fr_reg(6),
         FR7 => fr_reg(7),
         FR8 => fr_reg(8),
         FR9 => fr_reg(9),
         FR10 => fr_reg(10),
         FR11 => fr_reg(11),
         FR12 => fr_reg(12),
         FR13 => fr_reg(13),
         FR14 => fr_reg(14),
         FR15 => fr_reg(15),
         FL0  => fl_reg(0),
         FL1  => fl_reg(1),
         FL2  => fl_reg(2),
         FL3  => fl_reg(3),
         FL4  => fl_reg(4),
         FL5  => fl_reg(5),
         FL6  => fl_reg(6),
         FL7  => fl_reg(7),
         FL8  => fl_reg(8),
         FL9  => fl_reg(9),
         FL10 => fl_reg(10),
         FL11 => fl_reg(11),
         FL12 => fl_reg(12),
         FL13 => fl_reg(13),
         FL14 => fl_reg(14),
         FL15 => fl_reg(15));

MUX : multiplexer_16to4
port map(
         Inp => muxin,
         S => muxsel,
         Outp => muxout);

Counter : counter_2_bit
port map (
          reset => reset,
          clk => clock250,
          Outp => muxsel);

sel:Selection
Port map (
          I => mode,
          reset => reset,
          clock => clock250,
          button => button,
          Seed_Load => S_Seed_Load,
          lfsr_en => S_lfsr_en,
          Fr_Load => S_Fr_Load,
          Fl_Load => S_Fl_Load);

Regseed : Register_16_bit
port map (
           D => input,
           Reset => reset,
           Load => S_Seed_Load,
          Clock => clock_crystal,
          Section=> s_section,
          Q => seed_reg );

Regsr : Register_16_bit
port map (
           D => input,
           Reset => reset,
           Load => S_Fr_Load,
           Clock => clock_crystal,
           Section => s_section,
           Q => fr_reg);

Regsl : Register_16_bit
port map (
           D => input,
           Reset => reset,
           Load => S_Fl_Load,
           Clock => clock_crystal,
           Section => s_section,
           Q => fl_reg);

Decoder : Decoder_2to4
port map (
           din => muxsel,
           dout => driver_seg);
```

```
BCD_to_seg : BCD_to_7_segment_display_converter
port map (
          bcd => muxout,
          segment => seven_seg);

d_250Hz : Clock_250Hz
port map (
          clk_crystal => clock_crystal,
          reset => reset,
clk_250Hz => clock250);

d_1Hz : Clock_1Hz
port map (
          clk_crystal => clock_crystal,
          reset => reset,
          clk_1Hz => clock1);

s_section <= cs1 & cs0;

end Behavioral;
```

Εικόνα 4.36 : Κώδικας της συνολικής σχεδίασης.

Ο τελεστής **&** στη γραμμή του κώδικα **s_section <= cs1 & cs0;** ονομάζεται τελεστής συνένωσης ο οποίος είναι ενσωματωμένος στη VHDL και εκτελεί την αλληλουχία των bit vectors.Η παραπάνω φράση του κώδικα συνδέει το **cs1** με το αριστερό μισό του **s_section** και το **cs0** με το δεξί μισό του **s_section**. Το **&** επισυνάπτει το **cs0** στο τέλος του **cs1** για να σχηματιστεί ένα αποτέλεσμα που περιέχει 2 bits.

5. Συμπεράσματα

Σκοπός της παρούσας εργασίας ήταν η κατανόηση του θεωρητικού μέρους των LFSR, ο τρόπος υλοποίησης των διαφόρων πολυωνύμων, ο προγραμματισμός τους και το πώς μπορούμε να πάρουμε την αντίστροφη ακολουθία που παράγει στις εξόδους του ένα LFSR. Επίσης, η εκμάθηση της σχηματικής σχεδίασης του προγράμματος Project Navigator 13.1 της Xilinx. Σχεδιάστηκαν κυκλώματα με τα οποία μπορεί να προγραμματιστεί ένα LFSR και να παράξει την ανάστροφη ακολουθία. Έγινε σχεδίαση του δεκαεξάμπιτου αμφίδρομου καταχωρητή ολίσθησης με παράλληλη φόρτωση και με δυνατότητα αντιστροφής της ακολουθίας του που συμπεριλαμβάνει τα κυκλώματα δεξιάς-αριστερής ολίσθησης και προγραμματισμού, όπως και επεξήγηση του. Υλοποιήθηκε με σχηματικό τρόπο στο πρόγραμμα Project Navigator 13.1 της Xilinx και προσομοιώθηκε. Στη συνέχεια παρουσιάστηκε και επεξηγήθηκε η εφαρμογή που υλοποίησα στο αναπτυξιακό της Digilent, Spartan-3 Starter Kit Board, και δόθηκαν πληροφορίες για το αναπτυξιακό, για το 7-segment display και για την εφαρμογή. Τέλος έγινε καταγραφή του κώδικα VHDL των εξαρτημάτων που περιέχονται στην εφαρμογή και του κώδικα της συνολικής σχεδίασης που διασυνδέει τα υπόλοιπα εξαρτήματα.

Τα συμπεράσματα που εξήλθαν από την εργασία αυτή σε σχέση με τα LFSR είναι:

- Ο βαθμός του πολυωνύμου ενός LFSR εξαρτάται πάντα από τον αριθμό των flip-flop και άρα από τον αριθμό των εξόδων. Όταν χρειαζόμαστε μεγαλύτερες ακολουθίες σε μήκος πρέπει να μεγαλώσουν και οι φυσικές διαστάσεις του κυκλώματος.

- Ένα LFSR μήκους N μπορεί να περάσει από 2^N-1 διαφορετικές καταστάσεις, άρα μπορεί να «γεννήσει» ακολουθίες με μέγιστη περίοδο 2^N-1.

- Η ακολουθία εξόδου ενός LFSR εξαρτάται τόσο από την ανάδραση του όσο και από την αρχική του κατάσταση.

- Στην πράξη προτιμούμε LFSR που περνάνε από όλες τις καταστάσεις, έτσι ώστε η παραγόμενη ακολουθία να έχει τη μέγιστη δυνατή περίοδο.

- Οι ακολουθίες που παράγονται από LFSR μπορούν εύκολα να προβλεφθούν αν γνωρίζουμε ένα μικρό τμήμα της ακολουθίας τους.

Τα συμπεράσματα που εξήλθαν από την εργασία αυτή σε σχέση με τη σχηματική σχεδίαση και τη σχεδίαση στη γλώσσα περιγραφής υλικού VHDL, είναι:

- Με τη χρήση της γλώσσας VHDL, μπορούμε να σχεδιάσουμε εύκολα και γρήγορα ένα ψηφιακό κύκλωμα γράφοντας μερικές γραμμές κώδικα σε αντίθεση με τη σχηματική σχεδίαση που απαιτεί πολύ παραπάνω χρόνο και είναι πολύπλοκη.

- Με όση περισσότερη λεπτομέρεια σχεδιάζει κανείς ένα ψηφιακό κύκλωμα, τόσο περισσότερο κατανοεί τη λειτουργία του. Επίσης, όσο περισσότερο χρόνο αναλώνει κανείς για μία ψηφιακή σχεδίαση, τόσες περισσότερες λεπτομέρειες κατανοεί πάνω στη σχεδίαση αυτή. Για παράδειγμα, στη συγκεκριμένη εφαρμογή που υλοποιήθηκε, αρχικά σκέφτηκα ότι πρέπει να φτιάξω ένα ψηφιακό κύκλωμα που να προβάλει την ακολουθία εξόδου του κυκλώματος του αμφίδρομου προγραμματιζόμενου καταχωρητή ολίσθησης 16 bit με παράλληλη φόρτωση και με δυνατότητα αντιστροφής της ακολουθίας του, που σχεδιάστηκε στα παραπάνω κεφάλαια, στο 7-segment display και ο έλεγχος αυτού μέσω διακοπτών και μπουτόν (push buttons = κομβία πιέσεως) του αναπτυξιακού. Στη συνέχεια όρισα τους

διακόπτες και τα μπουτόν που χρειάζονται. Σκέφτηκα ποια θα είναι η λειτουργία τους και στη συνέχεια σκέφτηκα τι εξαρτήματα που χρειάζονται και τις συνδέσεις που πρέπει να κάνω για να υλοποιήσω την εφαρμογή. Έτσι κατανοείται σε όλο και μεγαλύτερο βάθος το κάθε ψηφιακό κύκλωμα και κάθε εφαρμογή.

Όσον αφορά τη βελτιστοποίηση, θα μπορούσε, στη σχεδίαση της εφαρμογής, ο διαιρέτης συχνότητας 1 Hz (d_1Hz) να είναι μεταβλητός και να μπορούμε να αυξομειώνουμε με χρήση κάποιων, για παράδειγμα, push buttons τη συχνότητα λειτουργίας του έτσι ώστε να αλλάζει η ταχύτητα εμφάνισης των δεκαεκαδικών τιμών πάνω στο 7-segment display.

Η εφαρμογή αυτή, που κάνει χρήση του αναπτυξιακού Spartan-3 Starter Kit Board της Digilent, θα μπορούσε να χρησιμοποιηθεί κάλλιστα είτε στο εργαστήριο Ψηφιακών Κυκλωμάτων ΙΙ είτε ή στο εργαστήριο Αρχιτεκτονικής Η/Υ ως εργαστηριακή άσκηση ώστε να μπορούν οι φοιτητές να πειραματιστούν. Επίσης θα μπορούσε να διδαχθεί και η σχηματική σχεδίαση.

Μελλοντικά, από άλλο φοιτητή, θα μπορούσε να γίνει ένα LFSR μεγαλύτερου μήκους ή ακόμα και συνδυασμός δύο ή περισσότερων LFSR για παραγωγή μη γραμμικών ακολουθιών που θα μπορούσε να χρησιμοποιηθεί στην κρυπτογραφία.

Παράρτημα Α

Σχηματική Σχεδίαση στο ISE Xilinx 13.1

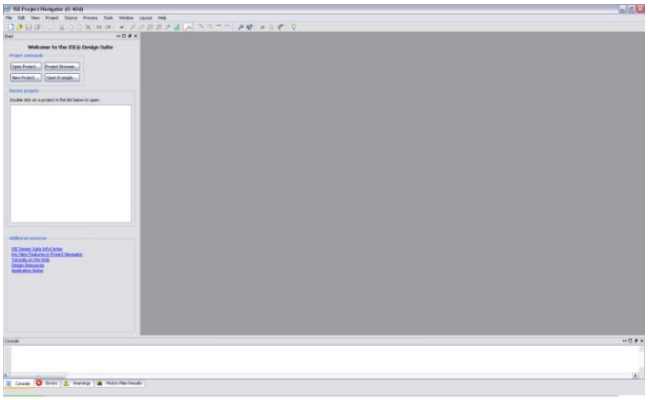
Έναρξη εφαρμογής

Από την επιφάνεια εργασίας μέσω του εικονιδίου:

Εικόνα Π.1 : Συντόμευση στην επιφάνεια εργασίας του προγράμματος ISE Xilinx Design Suite 13.1.

Διαφορετικά από την διαδρομή: **Έναρξη → Όλα τα προγράμματα → Xilinx ISE Design Suite 13.1 → ISE Design Tools → Project Navigator**

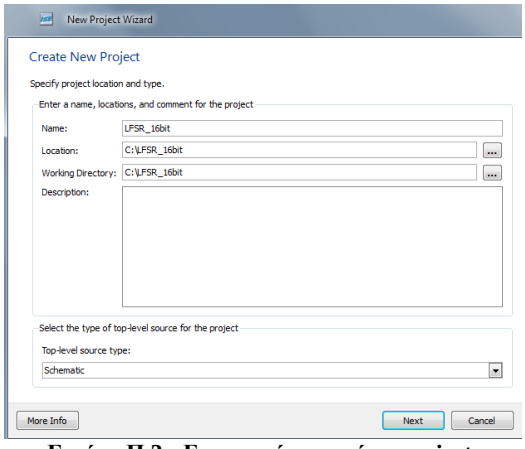
Η εφαρμογή μας εισάγει στο κεντρικό περιβάλλον:



Εικόνα Π.2 : Περιβάλλον Xilinx 13.1 για ανάπτυξη projects.

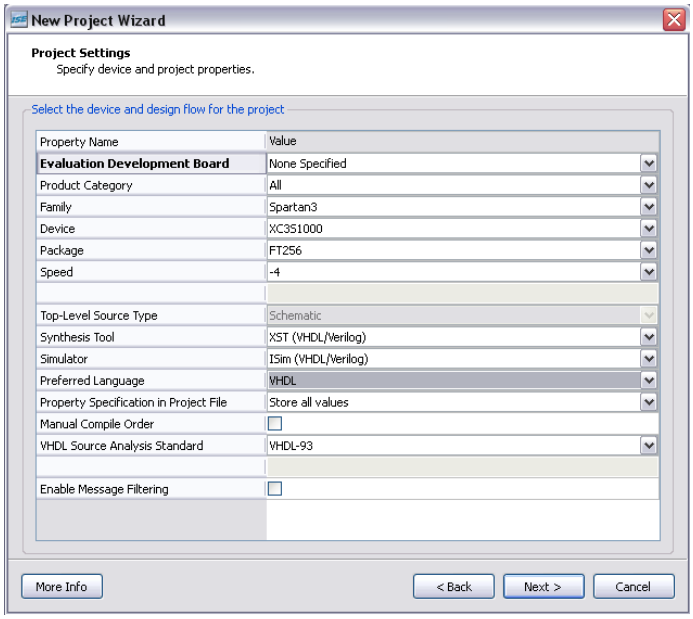
Δημιουργία νέου Project

Από το κεντρικό περιβάλλον επιλέγουμε τη διαδρομή **File → New Project** όπου και ανοίγει το παρακάτω παράθυρο:



Εικόνα Π.3 : Εισαγωγή στοιχείων project.

Στην περιοχή **Name** πληκτρολογούμε το όνομα που θέλουμε να έχει το project. Στο συγκεκριμένο δίνουμε το όνομα LFSR_16bit.*
Στην περιοχή **Location** μπορούμε να επιλέξουμε την τοποθεσία στην οποία θα αποθηκευτεί το Project (Η εφαρμογή από μόνη της δημιουργεί την διαδρομή και εμείς εάν θέλουμε την αλλάζουμε).
Στην περιοχή **Working Directory** δεν πληκτρολογούμε τίποτα. Το πρόγραμμα το ρυθμίζει από μόνο του.
Στην περιοχή **Description** μπορούμε, εάν θέλουμε, να γράψουμε λίγα λόγια για το τι είναι το project και τι περιέχει.
Τέλος, στην περιοχή **Top-level source type** επιλέγουμε **Schematic** αφού αυτή θα είναι η «γλώσσα» με την οποία θα εργαστούμε.
Κατόπιν, με αριστερό κλικ του ποντικιού επιλέγουμε το **Next** και ανοίγει το παρακάτω παράθυρο:



Εικόνα Π.4 : Εισαγωγή στοιχείων ολοκληρωμένου και εργαλείων.

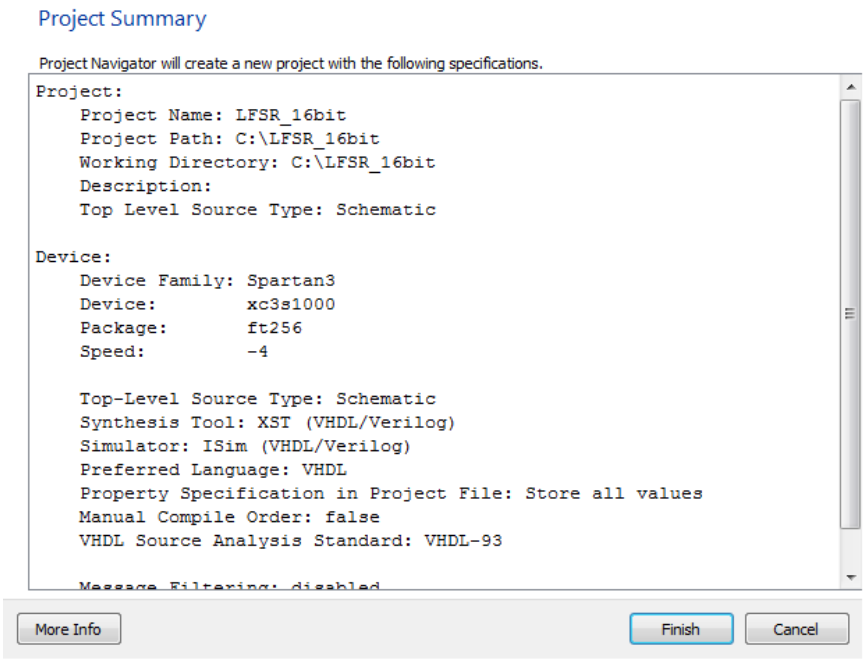
Στο παράθυρο αυτό εισάγουμε τα στοιχεία της συσκευής που θα χρησιμοποιήσουμε. Για το **Spartan-3 Starter Kit Board** που έχουμε εμείς επιλέγουμε τα παρακάτω:

Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3
Device	XC3S1000
Package	FT256
Speed	-4
Preferred Language	VHDL

Εικόνα Π.5 : Στοιχεία ολοκληρωμένου κυκλώματος.

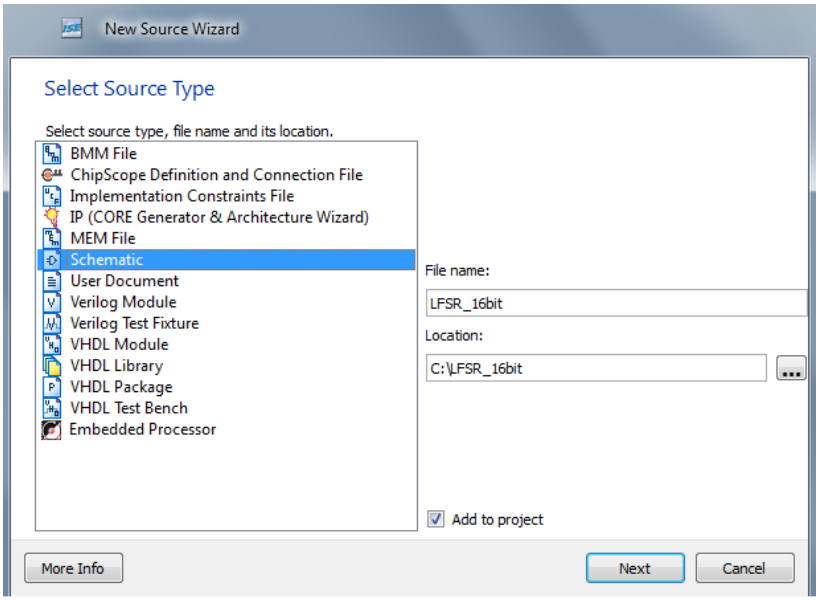
Όλες οι άλλες περιοχές παραμένουν ως έχουν.
Κατόπιν επιλέγουμε το **Next** και η εφαρμογή μας εμφανίζει ένα παράθυρο το οποίο μας εμφανίζει τις επιλογές ,που έχουμε κάνει μέχρι τώρα, για επιβεβαίωση.

* Στην περιοχή **Name** δεν μπορούμε να εισάγουμε κενό χαρακτήρα και σύμβολα παρά μόνο κάτω παύλα (_).



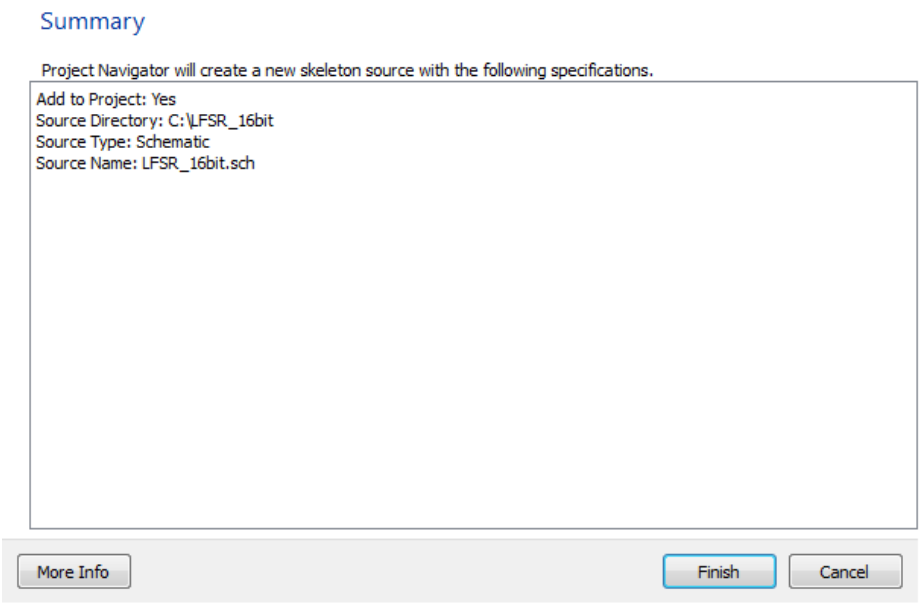
Εικόνα Π.6 : Επισκόπηση επιλογών ολοκληρωμένου κυκλώματος.

Τέλος, επιλέγουμε **Finish** και μας εμφανίζεται το κεντρικό παράθυρο της εφαρμογής. Τώρα επιλέγουμε το εικονίδιο  (new source) το οποίο βρίσκεται πάνω αριστερά στο παράθυρο αυτό ή κάνουμε δεξιό κλικ στην περιοχή Hierarchy και new source, όπου μας εμφανίζεται:



Εικόνα Π.7 : Εισαγωγή τύπου και ονόματος του νέου αρχείου.

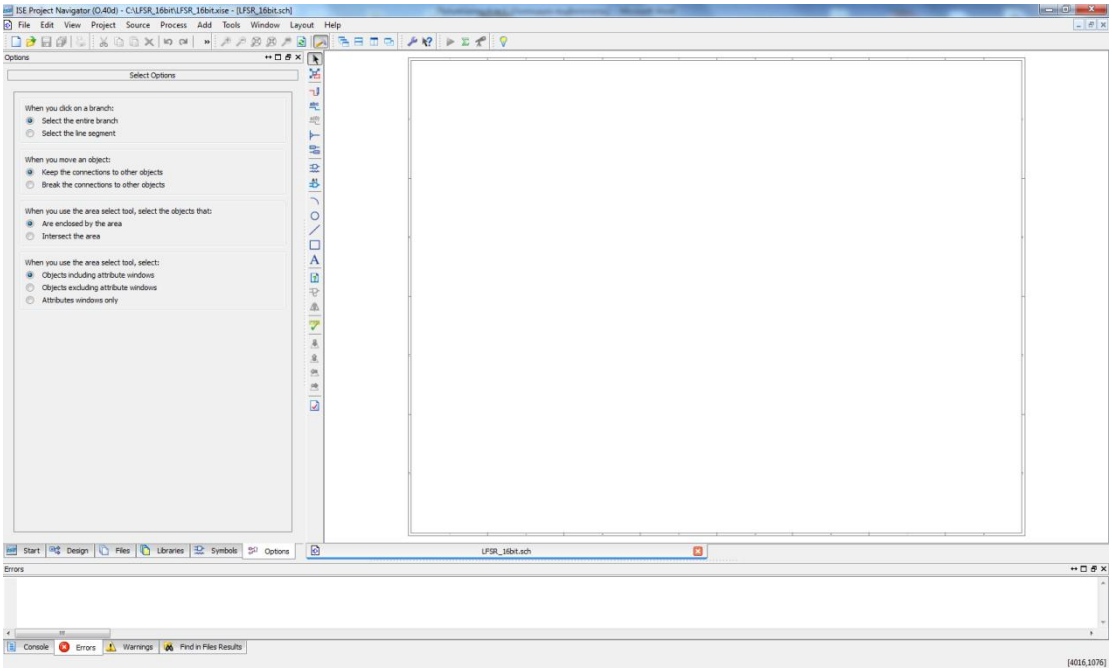
Διαλέγουμε από αριστερά το **Schematic** και από δεξιά στο **File name** πληκτρολογούμε το όνομα που θέλουμε να έχει το αρχείο (Μπορούμε να βάλουμε και το ίδιο όνομα με το project). Επίσης προσέχουμε το **Add to project** να είναι επιλεγμένο. Συνεχίζουμε επιλέγοντας **Next**, όπου μας εμφανίζεται ένα παράθυρο με τις πληροφορίες που εισάγαμε:



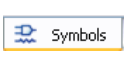
Εικόνα Π.8 : Επισκόπηση στοιχείων του νέου αρχείου.

Επιλέγουμε **Finish** και μπαίνουμε στο κεντρικό περιβάλλον σχηματικής σχεδίασης:

Νέο Σχηματικό



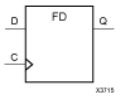
Εικόνα Π.9 : Εισαγωγή στο περιβάλλον της εργασίας στη σχηματική γλώσσα.

Ξεκινώντας τη σχεδίαση του σχηματικού, επιλέγουμε το εικονίδιο 

που βρίσκεται ακριβώς πάνω από την επικεφαλίδα **Welcome to the ISE® Design Suite.** Εναλλακτικά από την διαδρομή **Add→ Symbol** ή με **Ctrl + M** και μας εμφανίζονται από αριστερά οι επιλογές εισαγωγής συμβόλων. Στην λίστα **Categories** υπάρχουν ομαδοποιημένα όλα σχεδόν τα ψηφιακά κυκλώματα. Στη λίστα **Symbols** υπάρχουν τα ψηφιακά κυκλώματα κάθε κατηγορίας που επιλέξαμε από πριν. Στην περιοχή **Symbol Name Filter** αναζητούμε το σύμβολο που θέλουμε, πληκτρολογώντας το

όνομα του. Στην λίστα **Orientation** διαλέγουμε πως θα τοποθετήσουμε το κάθε σύμβολο στην περιοχή σχεδίασης.
Για τη σχεδίαση του 16 bit προγραμματιζόμενου αμφίδρομου καταχωρητή ολίσθησης με παράλληλη φόρτωση και με δυνατότητα αντιστροφής της ακολουθίας του, χρειαζόμαστε 16 flip-flop τύπου D, 16 πολυπλέκτες 4 σε 1 με είσοδο ενεργοποίησης, 30 πύλες XOR δύο εισόδων και 32 πύλες AND 2 εισόδων. Πληκτρολογούμε στην περιοχή **Symbol Name Filter** το όνομα του συμβόλου που θέλουμε (FD για το D flip-flop, M4_1E για το πολυπλέκτη 4 σε 1, XOR2 για την πύλη XOR και AND2 για την πύλη AND) ή αναζητούμε το σύμβολο από την λίστα **Symbols**. Για να σιγουρευτούμε ότι είναι αυτό το σύμβολο που θέλουμε επιλέγουμε το **Symbol Info** που είναι κάτω από την λίστα **Orientation** όπου μας εμφανίζει ένα παράθυρο με πληροφορίες για αυτό το σύμβολο (για παράδειγμα του D flip-flop):

Primitive: D Flip-Flop



Introduction

This design element is a D-type flip-flop with data input (D) and data output (Q). The data on the D inputs is loaded into the flip-flop during the Low-to-High clock (C) transition.
This flip-flop is asynchronously cleared, outputs Low, when power is applied. For FPGA devices, power-on conditions are simulated when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the appropriate STARTUP_*architecture* symbol.

Logic Table

Inputs		Outputs
D	C	Q
0	↑	0
1	↑	1

Design Entry Method

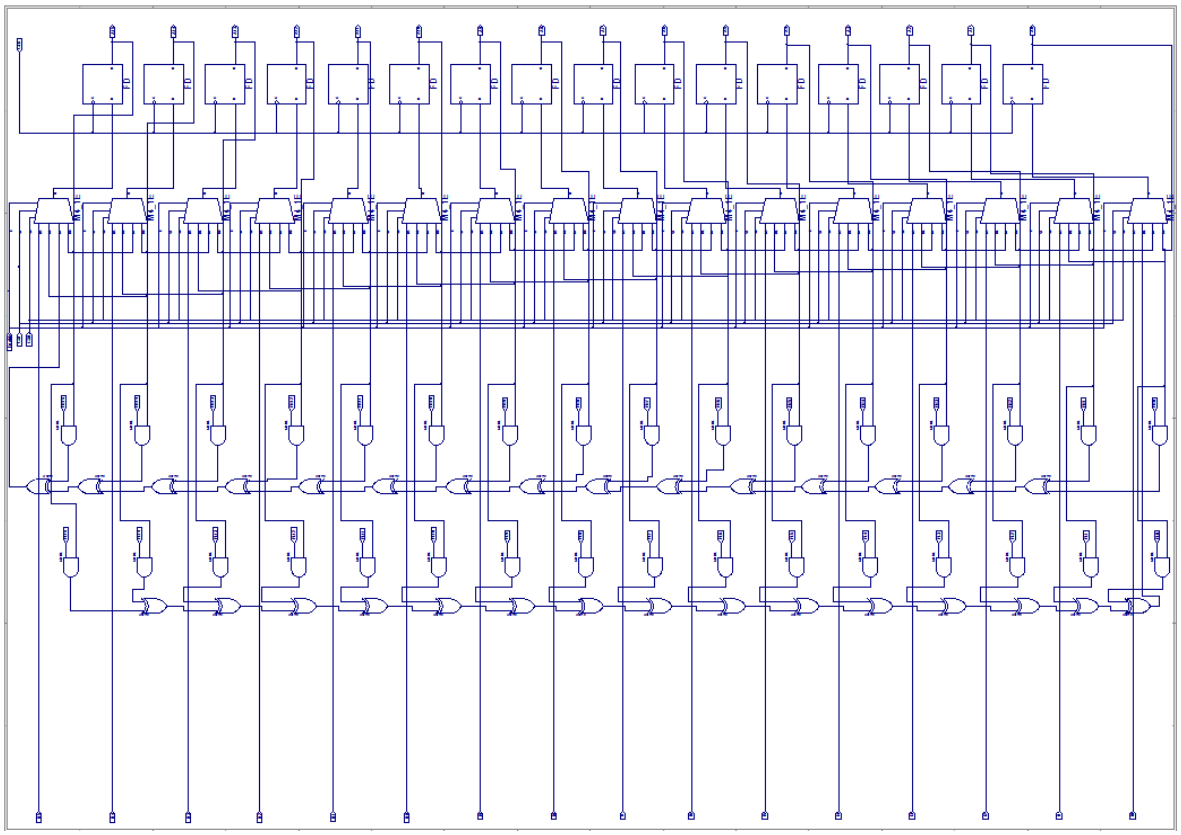
This design element is only for use in schematics.

Available Attributes

Attribute	Type	Allowed Values	Default	Description
INIT	Binary	0, 1	0	Sets the initial value of Q output after configuration

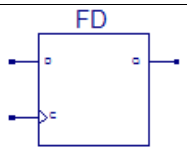
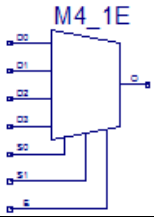
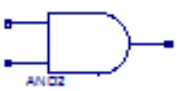
Εικόνα Π.10 : Πληροφορίες συμβόλου (D Flip-Flop).

Κατόπιν, κάνουμε αριστερό κλικ στο σύμβολο που βρήκαμε και το τοποθετούμε όσες φορές χρειάζεται στο χώρο σχεδίασης πάλι με αριστερό κλικ. Το ίδιο κάνουμε για όλα τα σύμβολα.
Αφού τοποθετήσουμε όλα τα σύμβολα στο χώρο εργασίας πρέπει να τα συνδέσουμε. Αυτό γίνεται επιλέγοντας το  ή από την διαδρομή **Add → Wire** ή με **Ctrl + W**.
Όταν τελειώσουμε με την σύνδεση των συμβόλων πρέπει να ονοματίσουμε τις εισόδους και τις εξόδους του κυκλώματος. Αυτό γίνεται επιλέγοντας το  ή από την διαδρομή **Add→ I/O Marker** ή με **Ctrl + G** όπου πάμε σε κάθε είσοδο και πατάμε αριστερό κλικ στην αρχή της. Για να τις μετονομάσουμε, πατάμε δεξιό κλικ σε κάθε είσοδο/έξοδο και **Rename Port**,πληκτρολογούμε το όνομα που θέλουμε και πατάμε OK. Εάν θέλουμε μπορούμε να ονοματίσουμε και τις συνδέσεις μεταξύ των συμβόλων. Αυτό γίνεται επιλέγοντας το  ή από την διαδρομή **Add→ Net Name** ή με **Ctrl + D**. Αριστερά στην περιοχή **Name** πληκτρολογούμε το όνομα που θέλουμε να έχει η σύνδεση και πατάμε αριστερό κλικ στην σύνδεση που θέλουμε. Αυτό γίνεται για όλες τις συνδέσεις.
Έτσι, σύμφωνα με τα παραπάνω, προκύπτει:



Εικόνα Π.11 : Η επιφάνεια εργασίας μετά την σχεδίαση του σχηματικού.

Τα σύμβολα που χρησιμοποιήθηκαν μαζί με τις ονομασίες τους στο πρόγραμμα, το σχήμα τους και τον κατάλογο στον οποίο βρίσκονται είναι:

Όνομα	**Όνομα στο πρόγραμμα**	**Σχήμα**	**Κατάλογος που βρίσκονται**
D Flip-Flop	FD		Flip-Flop
Πολυπλέκτης 4 σε 1 με είσοδο ενεργοποίησης	M4_1E		Mux
Πύλη XOR δύο εισόδων	XOR2		Logic
Πύλη AND δύο εισόδων	AND2		Logic

Εικόνα Π.12 : Χρησιμοποιούμενα σύμβολα.

Όλα τα σύμβολα που διαθέτει το πρόγραμμα βρίσκονται σε μορφή pdf στην ιστοσελίδα: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/7series_scm.pdf
Αφού ολοκληρώσουμε τη σχεδίαση πρέπει να την αποθηκεύσουμε. Αυτό γίνεται από την διαδρομή **File→ Save** ή **Save All.**

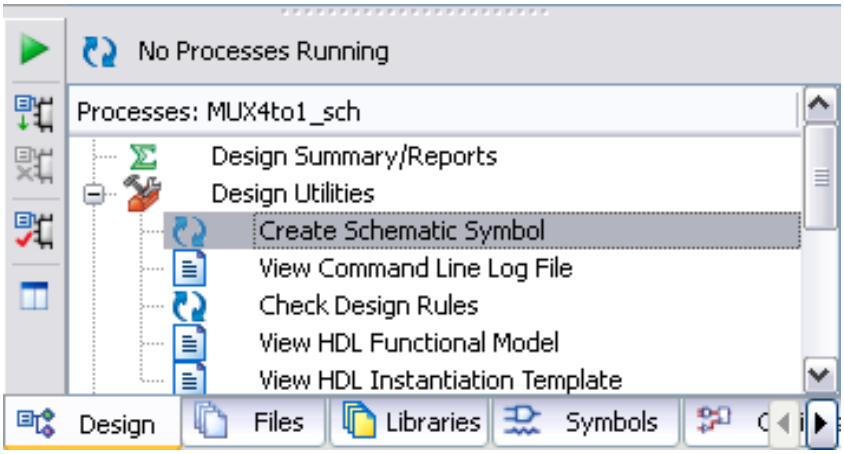
Ακόλουθα, πρέπει να «τσεκάρουμε» το σχηματικό, αν είναι σωστά σχεδιασμένο. Αυτό γίνεται επιλέγοντας το  ή από την διαδρομή **Tools→ Check Schematic.** Αν όλα πήγαν καλά, θα μας εμφανίσει στην κονσόλα η οποία είναι στο κάτω μέρος του παράθυρου το μήνυμα:

Start DRC ...
No errors or warnings were detected

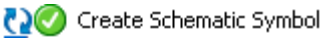
Δημιουργία Σχηματικού Συμβόλου

Αν θέλουμε να χρησιμοποιήσουμε ξανά το ψηφιακό κύκλωμα που σχεδιάσαμε προηγουμένως ολόκληρο, χωρίς να το ξαναδημιουργούμε εξ' αρχής σε κάθε σχεδίαση, μπορούμε να το αποθηκεύσουμε ως ένα σχηματικό σύμβολο και να το χρησιμοποιούμε αυτούσιο. Αυτό γίνεται ως εξής:

Στο παράθυρο σχεδίασης που ήμασταν πριν, πάμε στην καρτέλα **Design** που βρίσκεται αριστερά από τη καρτέλα **Symbols**, όπως φαίνεται και από την εικόνα:



Εικόνα Π.13 : Παράθυρο δημιουργίας σχηματικού συμβόλου.

Επιλέγουμε το «+» για να αναπτυχθεί η λίστα του **Design Utilities** και πατάμε διπλό αριστερό κλικ στο **Create Schematic Symbol**. Περιμένουμε λίγο μέχρι να μας εμφανιστεί το εικονίδιο:  .Επίσης η κονσόλα μας πληροφορεί με μήνυμα ότι το σύμβολο δημιουργήθηκε:

Process "Create Schematic Symbol" completed successfully

Το σύμβολο έχει αποθηκευτεί στο φάκελο που ορίσαμε στην αρχή όταν δηλώναμε το project. Για να το χρησιμοποιήσουμε σε άλλη σχεδίαση δεν έχουμε παρά να πάμε ξανά στην καρτέλα **Symbols** και στην λίστα **Categories** όπου μας δείχνει το φάκελο αποθήκευσης του project. Αν του κάνουμε αριστερό κλικ, στη λίστα **Symbols** μας εμφανίζει το σχηματικό σύμβολο που δημιουργήσαμε και που μπορούμε από εδώ και πέρα να το χρησιμοποιούμε κανονικά. Το σύμβολο, όπως δημιουργήθηκε σύμφωνα με τα παραπάνω, είναι[6][12]:

Εικόνα Π.14 : Σχηματικό σύμβολο.

Παράρτημα Β

Πρωταρχικά Πολυώνυμα 16ου Βαθμού

4 Ανατροφοδοτήσεις

16 15 13 4
16 15 12 10
16 15 12 1
16 15 10 4
16 15 9 6
16 15 9 4
16 15 7 2
16 15 4 2
16 14 13 11
16 14 13 5
16 14 12 7
16 14 11 7
16 14 9 7
16 14 9 4
16 14 8 3
16 13 12 11
16 13 12 7
16 13 11 6
16 13 9 6
16 13 6 4
16 12 9 7
16 12 9 6
16 11 10 5
16 11 9 8
16 11 9 7
16 10 9 6

6 Ανατροφοδοτήσεις

16 15 14 13 12 11
16 15 14 13 11 8
16 15 14 13 11 7
16 15 14 13 10 8
16 15 14 13 10 4
16 15 14 13 9 3
16 15 14 13 6 5
16 15 14 13 6 3
16 15 14 13 6 2
16 15 14 13 4 2
16 15 14 13 4 1
16 15 14 12 10 9
16 15 14 12 10 6
16 15 14 12 9 8
16 15 14 12 9 4
16 15 14 12 8 1
16 15 14 10 9 3
16 15 14 10 8 6
16 15 14 10 6 2
16 15 14 10 4 1
16 15 14 9 8 7
16 15 14 9 8 6
16 15 14 9 8 3
16 15 14 9 6 4
16 15 14 8 7 1
16 15 14 8 6 4
16 15 14 8 4 3
16 15 14 8 4 2
16 15 14 8 3 1
16 15 14 7 6 3
16 15 14 7 5 4
16 15 14 6 3 2
16 15 14 5 4 2
16 15 13 12 9 3
16 15 13 12 7 1
16 15 13 12 6 4
16 15 13 12 6 1
16 15 13 12 4 1
16 15 13 11 8 1
16 15 13 10 8 7
16 15 13 10 8 2
16 15 13 10 7 6
16 15 13 10 7 2
16 15 13 10 6 3
16 15 13 9 8 5
16 15 13 9 7 6
16 15 13 9 7 4
16 15 13 9 7 3
16 15 13 9 7 1
16 15 13 8 7 2
16 15 13 8 6 2
16 15 13 8 5 3
16 15 13 8 5 2
16 15 13 8 3 2
16 15 13 7 6 1
16 15 13 7 4 2
16 15 13 5 4 1
16 15 13 5 3 2
16 15 12 11 8 6
16 15 12 11 6 1
16 15 12 11 5 1
16 15 12 9 7 1
16 15 12 9 6 4
16 15 12 9 5 4
16 15 12 8 7 5
16 15 12 8 6 1
16 15 12 7 5 2
16 15 12 7 3 2
16 15 11 10 9 5
16 15 11 10 7 4
16 15 11 10 5 1
16 15 11 10 3 2
16 15 11 9 8 7
16 15 11 9 6 5
16 15 11 9 6 2
16 15 11 8 5 4
16 15 11 7 6 3
16 15 10 9 8 6
16 15 10 9 8 5
16 15 10 9 7 6
16 15 10 9 5 2
16 15 10 9 4 3
16 15 10 8 7 2
16 15 10 8 6 5
16 15 10 8 5 3
16 15 10 8 3 2
16 15 10 7 6 4
16 15 10 6 3 2
16 15 9 8 7 6
16 15 9 8 7 2
16 15 9 8 6 5
16 15 9 8 5 3
16 15 9 7 6 3
16 15 9 6 4 2
16 15 6 5 3 2
16 14 13 12 11 9
16 14 13 12 11 8
16 14 13 12 11 3
16 14 13 12 10 8
16 14 13 12 10 7
16 14 13 12 10 4
16 14 13 12 9 2
16 14 13 12 8 7
16 14 13 12 8 4
16 14 13 12 6 5
16 14 13 12 5 3

8 Ανατροφοδοτήσεις

16 15 14 13 12 11 10 4
16 15 14 13 12 11 9 8
16 15 14 13 12 11 9 6
16 15 14 13 12 11 9 5
16 15 14 13 12 11 8 2
16 15 14 13 12 11 5 2
16 15 14 13 12 10 9 5
16 15 14 13 12 10 5 1
16 15 14 13 12 9 8 7
16 15 14 13 12 9 7 1
16 15 14 13 12 9 6 3
16 15 14 13 12 9 6 2
16 15 14 13 12 9 3 2
16 15 14 13 12 8 4 1
16 15 14 13 12 7 6 4
16 15 14 13 12 6 5 3
16 15 14 13 12 6 5 2
16 15 14 13 12 6 3 2
16 15 14 13 12 5 4 3
16 15 14 13 12 5 3 2
16 15 14 13 11 10 8 4
16 15 14 13 11 10 6 1
16 15 14 13 11 10 5 4
16 15 14 13 11 10 3 1
16 15 14 13 11 9 8 5
16 15 14 13 11 9 8 2
16 15 14 13 11 9 7 4
16 15 14 13 11 9 7 3
16 15 14 13 11 9 6 5
16 15 14 13 11 9 6 4
16 15 14 13 11 9 6 1
16 15 14 13 11 9 4 3
16 15 14 13 11 8 5 2
16 15 14 13 11 8 5 1
16 15 14 13 11 7 5 2
16 15 14 13 11 7 4 3
16 15 14 13 11 5 4 1
16 15 14 13 10 9 7 2
16 15 14 13 10 9 6 1
16 15 14 13 10 9 3 1
16 15 14 13 10 9 2 1
16 15 14 13 10 6 5 3
16 15 14 13 10 6 3 1
16 15 14 13 9 8 6 5
16 15 14 13 9 8 6 4
16 15 14 13 9 8 6 2
16 15 14 13 9 7 3 2
16 15 14 13 8 7 6 1
16 15 14 13 8 6 5 2
16 15 14 13 8 6 3 2
16 15 14 13 8 5 4 1
16 15 14 13 8 4 2 1
16 15 14 13 7 6 5 3
16 15 14 13 7 5 4 1
16 15 14 13 6 4 3 1
16 15 14 13 5 4 2 1
16 15 14 12 11 10 9 1
16 15 14 12 11 10 7 4
16 15 14 12 11 10 6 2
16 15 14 12 11 10 4 3
16 15 14 12 11 10 3 2
16 15 14 12 11 9 8 1
16 15 14 12 11 9 4 2
16 15 14 12 11 8 6 2
16 15 14 12 11 8 6 1
16 15 14 12 11 8 3 1
16 15 14 12 11 8 2 1
16 15 14 12 11 7 6 4
16 15 14 12 11 7 6 1
16 15 14 12 11 7 4 3
16 15 14 12 11 5 4 3
16 15 14 12 11 5 2 1
16 15 14 12 10 9 4 2
16 15 14 12 10 8 7 3
16 15 14 12 10 8 7 1
16 15 14 12 10 8 6 3
16 15 14 12 10 7 5 4
16 15 14 12 10 7 4 3
16 15 14 12 10 6 4 2
16 15 14 12 10 4 3 2
16 15 14 12 9 7 4 2
16 15 14 12 9 7 3 2
16 15 14 12 9 6 5 1
16 15 14 12 9 6 3 2
16 15 14 12 8 7 4 1
16 15 14 12 8 6 5 4
16 15 14 12 8 6 3 1
16 15 14 12 8 5 3 2
16 15 14 11 10 9 8 5
16 15 14 11 10 9 8 2
16 15 14 11 10 9 6 3
16 15 14 11 10 9 4 1
16 15 14 11 10 8 3 2
16 15 14 11 10 7 5 1
16 15 14 11 10 7 2 1
16 15 14 11 10 6 5 2
16 15 14 11 10 6 3 1
16 15 14 11 10 5 2 1
16 15 14 11 9 8 5 1
16 15 14 11 9 7 4 1
16 15 14 11 9 6 5 2
16 15 14 11 9 6 4 1
16 15 14 11 8 7 5 3
16 15 14 11 8 7 4 2
16 15 14 11 8 7 4 1
16 15 14 11 8 6 5 1
16 15 14 11 8 6 4 2
16 15 14 11 8 6 3 1
16 15 14 11 6 5 3 1
16 15 14 10 9 8 6 1
16 15 14 10 9 8 5 3
16 15 14 10 9 6 5 2
16 15 14 10 9 5 4 3
16 15 14 10 9 4 3 1
16 15 14 10 8 7 6 4
16 15 14 10 8 7 5 4
16 15 14 10 8 6 4 1
16 15 14 10 8 5 4 1
16 15 14 10 8 5 3 2
16 15 14 10 7 6 5 3
16 15 14 10 7 6 5 1
16 15 14 10 7 6 4 2
16 15 14 10 6 5 4 3
16 15 14 9 8 7 6 5
16 15 14 9 8 7 6 3
16 15 14 9 8 6 5 4
16 15 14 9 7 6 5 2
16 15 14 9 7 4 3 1
16 15 14 9 6 4 3 2
16 15 14 8 7 6 4 3
16 15 14 8 7 6 3 2
16 15 14 8 6 5 4 3
16 15 14 8 6 5 4 2
16 15 14 7 6 5 4 1
16 15 14 6 5 4 3 1
16 15 13 12 11 10 9 5
16 15 13 12 11 10 6 5
16 15 13 12 11 10 5 3
16 15 13 12 11 10 3 1
16 15 13 12 11 9 8 2
16 15 13 12 11 9 7 5
16 15 13 12 11 9 7 1
16 15 13 12 11 9 6 2
16 15 13 12 11 9 5 4
16 15 13 12 11 9 4 2
16 15 13 12 11 8 7 4
16 15 13 12 11 7 5 3
16 15 13 12 11 7 3 2
16 15 13 12 11 7 3 1
16 15 13 12 11 6 5 3
16 15 13 12 11 6 4 1
16 15 13 12 11 6 3 2

10 Ανατροφοδοτήσεις

16 15 14 13 12 11 10 9 8 2
16 15 14 13 12 11 10 9 7 1
16 15 14 13 12 11 10 9 6 3
16 15 14 13 12 11 10 9 5 4
16 15 14 13 12 11 10 9 5 2
16 15 14 13 12 11 10 8 7 2
16 15 14 13 12 11 10 8 6 1
16 15 14 13 12 11 10 8 5 1
16 15 14 13 12 11 10 8 3 2
16 15 14 13 12 11 10 7 6 3
16 15 14 13 12 11 10 7 5 3
16 15 14 13 12 11 10 6 4 2
16 15 14 13 12 11 10 4 3 1
16 15 14 13 12 11 10 3 2 1
16 15 14 13 12 11 9 8 7 2
16 15 14 13 12 11 9 8 7 1
16 15 14 13 12 11 9 7 3 1
16 15 14 13 12 11 9 6 3 1
16 15 14 13 12 11 8 7 6 4
16 15 14 13 12 11 8 6 5 4
16 15 14 13 12 11 8 6 3 2
16 15 14 13 12 11 8 5 4 3
16 15 14 13 12 11 7 6 5 4
16 15 14 13 12 11 5 4 3 2
16 15 14 13 12 10 9 8 7 1
16 15 14 13 12 10 9 8 5 2
16 15 14 13 12 10 9 8 4 2
16 15 14 13 12 10 9 7 6 2
16 15 14 13 12 10 9 7 5 2
16 15 14 13 12 10 9 7 2 1
16 15 14 13 12 10 9 6 4 2
16 15 14 13 12 10 9 5 4 3
16 15 14 13 12 10 9 5 4 2
16 15 14 13 12 10 9 5 3 1
16 15 14 13 12 10 9 5 2 1
16 15 14 13 12 10 9 3 2 1
16 15 14 13 12 10 8 6 5 4
16 15 14 13 12 10 8 6 5 2
16 15 14 13 12 10 8 5 4 3
16 15 14 13 12 10 8 5 2 1
16 15 14 13 12 10 8 4 3 2
16 15 14 13 12 10 7 6 5 4
16 15 14 13 12 10 7 6 4 1
16 15 14 13 12 10 7 4 3 2
16 15 14 13 12 10 7 3 2 1
16 15 14 13 12 10 6 5 3 1
16 15 14 13 12 10 4 3 2 1
16 15 14 13 12 9 8 7 4 3
16 15 14 13 12 9 8 7 4 1
16 15 14 13 12 9 8 7 2 1

16 15 12 11 10 9 8 6 5 2
16 15 12 11 10 9 8 6 3 2
16 15 12 11 10 9 7 6 5 4
16 15 12 11 10 9 7 5 4 3
16 15 12 11 9 8 6 5 4 3
16 15 12 11 9 8 6 5 3 2
16 15 12 10 9 8 7 6 5 3
16 15 12 10 9 8 7 6 5 1
16 15 12 10 9 8 7 5 4 3
16 15 12 10 9 7 6 5 4 3
16 15 12 10 9 6 5 4 3 2
16 15 12 9 7 6 5 4 3 2
16 15 11 10 9 8 7 6 5 2
16 15 11 10 9 8 7 6 4 3
16 15 11 10 8 7 5 4 3 2
16 14 13 12 11 10 9 8 6 2
16 14 13 12 11 10 9 8 5 4
16 14 13 12 11 10 9 7 6 4
16 14 13 12 11 10 9 7 6 2
16 14 13 12 11 10 9 6 5 2
16 14 13 12 11 10 9 6 4 3
16 14 13 12 11 10 9 4 3 2
16 14 13 12 11 10 8 7 6 2
16 14 13 12 11 10 8 7 5 3
16 14 13 12 11 10 8 7 5 2
16 14 13 12 11 10 8 6 4 3
16 14 13 12 11 10 7 5 4 2
16 14 13 12 11 10 6 4 3 2
16 14 13 12 11 9 8 7 5 4
16 14 13 12 11 9 8 5 4 3
16 14 13 12 11 9 7 5 4 3
16 14 13 12 11 8 7 6 5 3
16 14 13 12 11 8 6 5 4 3
16 14 13 12 11 7 6 4 3 2
16 14 13 12 10 9 8 7 5 2
16 14 13 12 10 9 8 4 3 2
16 14 13 12 10 9 7 4 3 2
16 14 13 12 10 7 6 5 3 2
16 14 13 12 9 8 7 5 4 3
16 14 13 12 9 8 7 5 3 2
16 14 13 12 9 8 6 5 3 2
16 14 13 12 8 7 6 5 4 3
16 14 13 11 10 9 8 7 6 3
16 14 13 11 10 9 8 7 3 2
16 14 13 11 10 9 7 6 4 3
16 14 13 11 10 8 7 6 5 2
16 14 13 10 9 8 7 6 4 3
16 14 13 10 9 8 6 5 4 3
16 14 10 9 8 7 6 5 4 3
16 13 12 11 10 9 7 6 5 3
16 13 12 11 9 8 7 6 5 4

16 15 14 13 12 9 8 5 4 2
16 15 14 13 12 9 8 4 2 1
16 15 14 13 12 9 7 6 5 4
16 15 14 13 12 9 7 6 4 2
16 15 14 13 12 9 7 5 3 1
16 15 14 13 12 8 7 6 4 1
16 15 14 13 12 8 7 6 3 1
16 15 14 13 12 8 7 5 3 1
16 15 14 13 12 8 7 5 2 1
16 15 14 13 12 8 5 4 2 1
16 15 14 13 12 7 5 4 3 2
16 15 14 13 12 7 5 3 2 1
16 15 14 13 11 10 9 8 7 2
16 15 14 13 11 10 9 8 5 1
16 15 14 13 11 10 9 7 4 3
16 15 14 13 11 10 9 6 5 4
16 15 14 13 11 10 9 6 5 1
16 15 14 13 11 10 9 4 3 1
16 15 14 13 11 10 8 5 2 1
16 15 14 13 11 10 8 4 3 2
16 15 14 13 11 10 7 6 5 2
16 15 14 13 11 10 7 6 4 3
16 15 14 13 11 10 7 5 4 1
16 15 14 13 11 10 6 5 4 3
16 15 14 13 11 10 6 5 3 2
16 15 14 13 11 10 6 5 2 1
16 15 14 13 11 10 6 4 3 1
16 15 14 13 11 10 5 3 2 1
16 15 14 13 11 9 8 7 6 1
16 15 14 13 11 9 8 7 5 4
16 15 14 13 11 9 8 6 4 2
16 15 14 13 11 9 8 5 3 2
16 15 14 13 11 9 8 5 3 1
16 15 14 13 11 9 7 6 5 2
16 15 14 13 11 9 7 4 3 2
16 15 14 13 11 8 6 4 2 1
16 15 14 13 11 8 5 4 3 1
16 15 14 13 11 7 6 3 2 1
16 15 14 13 11 6 5 4 2 1
16 15 14 13 10 9 8 7 6 5
16 15 14 13 10 9 8 7 5 2
16 15 14 13 10 9 8 7 3 1
16 15 14 12 11 10 7 5 4 3
16 15 14 12 11 10 7 5 3 2
16 15 14 12 11 10 7 5 3 1
16 15 14 12 11 9 8 7 5 2
16 15 14 12 11 9 8 7 5 1
16 15 14 12 11 9 8 7 2 1
16 15 14 12 9 8 5 4 3 2
16 15 14 12 9 7 6 5 3 2
16 15 14 11 10 9 8 6 5 1

12 Ανατροφοδοτήσεις

16 15 14 13 12 11 10 9 8 7 5 2
16 15 14 13 12 11 10 9 8 6 5 4
16 15 14 13 12 11 10 9 8 5 4 3
16 15 14 13 12 11 10 9 8 5 4 2
16 15 14 13 12 11 10 9 8 5 4 1
16 15 14 13 12 11 10 9 7 6 5 3
16 15 14 13 12 11 10 8 7 5 4 1
16 15 14 13 12 11 10 8 7 4 3 1
16 15 14 13 12 11 10 8 6 5 4 2
16 15 14 13 12 11 10 8 5 4 3 2
16 15 14 13 12 11 10 7 6 5 3 2
16 15 14 13 12 11 10 7 4 3 2 1
16 15 14 13 12 11 9 8 7 6 5 4
16 15 14 13 12 11 9 8 7 6 3 2
16 15 14 13 12 11 9 8 7 6 3 1
16 15 14 13 12 11 9 8 6 5 3 2
16 15 14 13 12 11 9 8 6 4 3 1
16 15 14 13 12 11 9 8 6 4 2 1
16 15 14 13 12 11 9 7 6 5 2 1
16 15 14 13 12 11 9 6 5 4 3 2
16 15 14 13 12 11 8 7 5 4 3 1
16 15 14 13 12 10 9 8 7 6 3 2
16 15 14 13 12 10 9 8 7 5 4 2
16 15 14 13 12 10 9 8 7 5 4 1
16 15 14 13 12 10 9 8 6 3 2 1
16 15 14 13 12 10 9 7 6 5 4 3
16 15 14 13 12 10 9 7 6 5 4 1
16 15 14 13 12 10 9 7 6 5 3 1
16 15 14 13 12 10 9 7 6 4 3 2
16 15 14 13 12 10 9 7 4 3 2 1
16 15 14 13 12 10 8 7 6 4 2 1
16 15 14 13 12 10 7 6 5 4 3 1
16 15 14 13 12 9 8 7 6 3 2 1
16 15 14 13 12 9 8 7 5 3 2 1
16 15 14 13 11 10 9 8 7 6 5 1
16 15 14 13 11 10 9 8 7 5 4 3
16 15 14 13 11 10 9 8 7 5 4 1
16 15 14 13 11 10 9 8 7 4 2 1
16 15 14 13 11 10 9 8 6 4 2 1
16 15 14 13 11 10 9 8 5 4 3 2
16 15 14 13 11 10 9 7 6 4 3 1
16 15 14 13 11 10 9 7 5 3 2 1
16 15 14 13 11 10 8 7 6 4 3 1
16 15 14 13 11 10 8 7 5 4 3 1
16 15 14 13 11 9 8 7 6 5 3 1
16 15 14 13 11 9 8 7 6 3 2 1
16 15 14 13 11 9 8 6 5 4 3 1
16 15 14 13 11 9 7 6 5 4 3 2
16 15 14 13 11 8 7 6 5 4 3 2
16 15 14 13 11 8 7 6 5 4 3 1

16 15 14 13 10 9 8 7 5 4 3 1
16 15 14 12 11 10 9 8 7 6 3 2
16 15 14 12 11 10 9 8 6 5 4 1
16 15 14 12 11 10 9 8 6 4 2 1
16 15 14 12 11 10 9 8 5 4 3 1
16 15 14 12 11 10 9 8 5 4 2 1
16 15 14 12 11 10 9 7 5 4 3 2
16 15 14 12 11 10 9 6 5 4 3 1
16 15 14 12 11 9 8 7 6 5 4 1
16 15 14 12 11 9 8 7 6 4 3 2
16 15 14 12 11 9 8 7 6 4 2 1
16 15 14 12 11 8 7 6 5 4 3 1
16 15 14 12 10 9 8 7 6 4 3 1
16 15 14 12 10 9 8 6 5 4 3 2
16 15 14 11 9 8 7 6 5 4 3 1
16 15 14 10 9 8 7 6 5 4 3 2
16 15 13 12 11 10 9 8 7 5 3 2
16 15 13 12 11 10 9 8 6 5 3 1
16 15 13 12 11 10 9 7 6 5 4 3
16 15 13 12 11 10 8 7 6 5 4 1
16 15 13 12 11 9 8 7 5 4 3 2
16 15 13 12 11 9 7 6 5 4 3 2
16 15 13 12 10 9 8 7 6 5 3 2
16 15 12 11 9 8 7 6 5 4 3 2
16 15 12 10 9 8 7 6 5 4 3 2
16 14 13 12 11 10 9 8 6 5 4 2
16 14 13 12 11 10 9 7 6 5 3 2
16 14 13 12 11 9 8 7 6 5 4 2

14 Ανατροφοδοτήσεις

16 15 14 13 12 11 10 9 8 7 6 5 3 2
16 15 14 13 12 11 10 9 8 7 6 5 3 1
16 15 14 13 12 11 10 8 7 5 4 3 2 1
16 15 14 13 12 10 9 8 7 6 5 4 3 1
16 15 14 13 12 10 9 8 7 6 5 3 2 1
16 15 14 13 11 10 9 8 7 6 5 4 2 1

Βιβλιογραφία

Βιβλία

[1] «Σχεδίαση Ψηφιακών Συστημάτων με τη Γλώσσα VHDL», Stephen Brown & Zvonko Vranesic, Εκδόσεις Τζιόλα 2001

[2] «Μελέτη, Σχεδίαση και Κατασκευή Προγραμματιζόμενων LFSR που θα χρησιμοποιηθούν σαν Pattern Generators και Signature Analyzers», Πτυχιακή Εργασία, Γεωργιάδης Μιχαήλ, ΤΕΙ Κρήτης, Τμήμα Ηλεκτρονικής 2002

[3] «Τεχνικές Σχεδίασης Κυκλωμάτων CMOS», Πανεπιστήμιο Ιωαννίνων, Γ.Τσιατούχας

[4] «A Designer's Guide to Build-In Self-Test», Charles E. Stroud, Kluwer Academic Publishers 2002

[5] «Ψηφιακή Σχεδίαση», M.Morris Mano, 2η έκδοση, Εκδόσεις Παπασωτηρίου 1992

[6] «Δημιουργία τράπεζας Ψηφιακών Κυκλωμάτων με χρήση τεχνολογίας P.L.D», Πτυχιακή Εργασία, Κοκίδης Κωνσταντίνος, ΤΕΙ Θεσσαλονίκης, Τμήμα Ηλεκτρονικής 2009

[7] «Εφαρμογές της ψηφιακής σχεδίασης με χρήση της γλώσσας περιγραφής VHDL και υλοποίηση τους σε FPGA», Εργασία Πρακτικής Άσκησης, Τζαβάρας Σάββας-Χαράλαμπος, ΤΕΙ Κρήτης, Τμήμα Ηλεκτρονικής 2012

Άρθρα

[8] «Linear Feedback Shift Register v3.0», Xilinx Logic Core ,March 28 2003

[9] «LFSR counters implement binary polynomial generators», Tom Balph, Motorola Semiconductor

Ιστότοποι

[10] «Linear Feedback Shift Register» http://www.ece.ualberta.ca/~elliott/ee552/studentAppNotes/1999f/Drivers_Ed/lfsr.html

[11] «Linear Feedback Shift Register» http://en.wikipedia.org/wiki/Linear_feedback_shift_register

[12] «ISE In-Depth Tutorial, March 1, 2011» http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/ise_tutorial_ug695.pdf

[13] «FPGA» http://el.wikipedia.org/wiki/FPGA

[14] «Spartan 3 Starter Kit Board User Guide» http://www.digilentinc.com/Data/Products/S3BOARD/S3BOARD_RM.pdf

[15] «Spartan-3 Starter Kit Board Schematics» http://www.digilentinc.com/Data/Products/S3BOARD/S3BOARD-sch.pdf