Περιεχόμενα

Το CD που συνοδεύει τις έντυπες σημειώσεις περιέχει:

- **Τελική εφαρμογή του προγράμματος σε Visual Basic**
  Cd:\VB

- **Πηγαίο κώδικα του παραπάνω προγράμματος**
  Cd:\VB

- **Πηγαίο κώδικα της εφαρμογής σε VHDL**
  Cd:\VHDL

- **Διάφορα ηλεκτρονικά αρχεία (.pdf) που χρησιμοποιήσαμε**
  Cd:\E-FILES

- **Αυτές τις σημειώσεις σε ηλεκτρονική μορφή (notes.doc και notes.pdf)**
  Cd:\NOTES

- **Μικροπρογράμματα που αναπτύχθηκαν κατά την διάρκεια της πτυχιακής εργασίας**
  Cd:\VARIOUS

Γλωσσάριο

Aliasing : Aliazing ονομάζεται η περίπτωση που μαζεύετε η ίδια ψηφιακή υπογραφή στον Analyzer ενώ τα κυκλώματα προς εξέταση είναι τα ίδια.

Multiplexer : Ο πολυπλέκτης είναι ψηφιακό εξάρτημα με το οποίο επιτυγχάνουμε να ορίσουμε πια από τις εισόδους που διαθέτει θα περάσει στην έξοδο. Ο έλεγχος γίνεται από μια γραμμή Select.

Flip flop : Το Flip Flop είναι ακολουθιακό εξάρτημα. Έχει μόνο μια είσοδο δεδομένων και μια είσοδο για παλμούς χρονισμού. Όταν ένας παλμός χρονισμού φτάσει στην είσοδο χρονισμού του, τότε μεταφέρεται στην έξοδο του η λογική κατάσταση που υπάρχει στην είσοδο δεδομένων.

Shift Register : Αποτελείται από πολλά Flip Flops στην σειρά συνδεμένα. Η έξοδος του πρώτου γίνεται είσοδος για το δεύτερο και ούτω καθεξής.

CUT: (Circuit Under Test). Το υπό έλεγχο κύκλωμα που συνδέουμε μεταξύ του Pattern Generator και του Signature Analyzer για να ελέγξουμε αν δουλεύει σωστά.

LFSR : Linear Feedback Shift Register (Γραμμικός Ανατροφοδοτούμενος Καταχωρητής Ολίσθησης).

Pattern Generator : Γεννήτορας Συνδυασμών. Βασίζεται στην αρχή λειτουργίας των LFSR και παράγει ψευδοτυχαίες ακολουθίες.

Signature Analyzer : Αναλυτής υπογραφής. Βασίζεται στην αρχή λειτουργίας των LFSR και παράγει ψηφιακές υπογραφές από ψευδοτυχαίες ακολουθίες που λαμβάνει στην είσοδο του.

Galois : Τύπος LFSR όπου οι πύλες XOR είναι εσωτερικά, δηλαδή ανάμεσα, στα Flip Flop του καταχωρητή ολίσθησης.

Fibonacci : Τύπος LFSR όπου οι πύλες XOR είναι εξωτερικά των Flip Flop του καταχωρητή ολίσθησης.

Primitive : Πολυώνυμο λειτουργίας κατά το οποίο λαμβάνονται όλες οι δυνατές ψηφιακές τιμές όταν αυτό υλοποιείται σε ένα LFSR.

Non-primitive : Πολυώνυμο λειτουργίας κατά το οποίο λαμβάνονται μόνο μερικές ψηφιακές τιμές όταν αυτό υλοποιείται σε ένα LFSR.

USB : Universal Serial Bus (Σειριακός δίαυλος γενικής χρήσης). Αντικαθιστά την παλαιότερη σειριακή RS-232 στους υπολογιστές καθώς ενσωματώνει καινούργια χαρακτηριστικά και μεγαλύτερη ταχύτητα.

Seed value : Αρχική τιμή από την οποία ξεκινάει να τρέχει το LFSR. Όταν μετά από ώρα η έξοδος του LFSR πάρει την τιμή που είχαμε θέσει σαν αρχική τότε θεωρείται ότι έγινε ένας πλήρης κύκλος λειτουργίας.

FPGA : (Field Programmable Gate Array). Τύπος ολοκληρωμένου το οποίο προγραμματίζουμε με το πρόγραμμα που κάναμε στην γλώσσα περιγραφής υλικού (VHDL).

Visual Basic: Γλώσσα προγραμματισμού που δίνει δυνατότητα για δημιουργία αυτόνομων εφαρμογών που τρέχουν σε περιβάλλον Windows.

GUI : Graphical User Interface (Γραφικό Περιβάλλον Χρήστη). Οπτικός προγραμματισμός που χρησιμοποιείται στην Visual Basic.

Clock Pulses : Παλμοί χρονισμού που χρησιμοποιούνται στην εφαρμογή μας κατά κύριο λόγο στο γέμισμα των καταχωρητών ολίσθησης αλλά και στο τρέξιμο του LFSR.

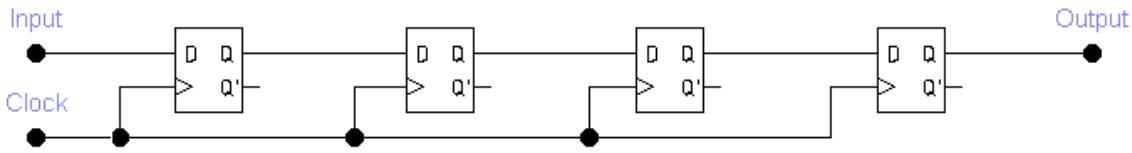
1.Γενικά για τα LFSR.

- *Τι είναι LFSR.*

LFSR είναι σύντμηση των αγγλικών λέξεων Linear Feedback Shift Register **(Γραμμικός Ανατροφοδοτούμενος Καταχωρητής Ολίσθησης).**

Στην υποενότητα αυτή θα εξετάσουμε από τι αποτελείται ένα LFSR και ποια είναι τα χαρακτηριστικά του.

Ένα LFSR αποτελείται από καταχωρητές ολίσθησης όπως μας φανερώνει και το όνομα του. Οι καταχωρητές ολίσθησης έχουν την μορφή D Flip Flops συνδεμένα σε σειρά και κάνουν ολίσθηση προς δεξιά ή αριστερά. Συνήθως χρησιμοποιούνται αυτά που κάνουν ολίσθηση προς δεξιά. Παρακάτω δίνουμε το σχηματικό διάγραμμα ενός 4bit καταχωρητή ολίσθησης (4bit Shift Register).



Σχήμα 1.1 4bit Shift Register

Από τον πίνακα αληθείας του D Flip Flop που παραθέτουμε παρακάτω γίνεται αντιληπτό ότι ένα bit εισόδου θα βγει στην έξοδο μετά από 4 παλμούς ρολογιού (clock pulses)[13].

CP (είσοδος)	D (είσοδος)	Q (έξοδος)	Q' (έξοδος)
↑	1	1	0
↑	0	0	1

1.1 Πίνακας αληθείας D Flip Flop

Το ↑ σημαίνει ότι έχουμε αλλαγή κατάστασης στο CP από λογικό «0» σε λογικό «1» και ο παραπάνω πίνακας αληθείας ισχύει για positive edge triggered D Flip Flop (θετικό ακμοπυροδοτούμενο Flip Flop).

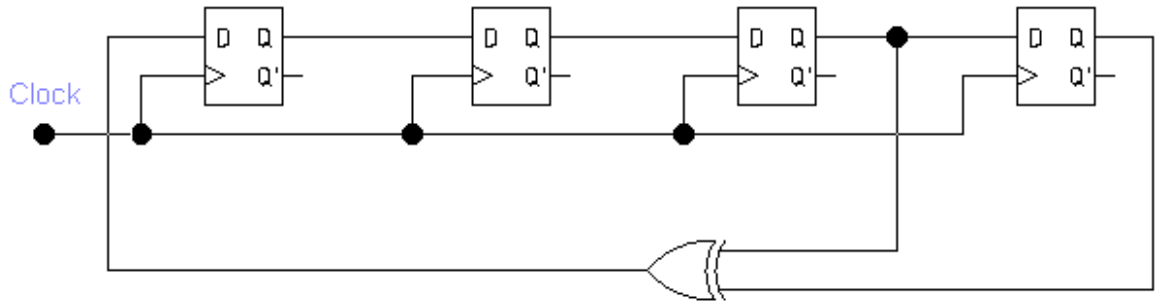
Ο αριθμός των bits ενός καταχωρητή ολίσθησης εξαρτάται πάντα από τον αριθμό των D Flip Flops και άρα τον αριθμό των εξόδων των D Flip Flops που υπάρχουν στο κύκλωμα μας. Όταν χρειαστούμε μεγαλύτερες ψευδοτυχαίες ακολουθίες σε μήκος πρέπει να μεγαλώσουν και οι φυσικές διαστάσεις του Hardware.

Για να ολοκληρωθεί ένα LFSR το μόνο που μένει να γίνει είναι η ανατροφοδότηση. Αυτό επιτυγχάνεται με την βοήθεια μιας πύλης XOR που στην είσοδό της συλλέγει κάποιες από τις εξόδους των D Flip Flops και στην έξοδό της παίρνουμε το σήμα ανατροφοδότησης που συνδέεται στην είσοδο του πρώτου D Flip Flop.

Υπάρχουν δύο βασικά είδη LFSR ανάλογα με το πώς συνδέεται η πύλη XOR στο κύκλωμα του καταχωρητή ολίσθησης. Η πρώτη υλοποίηση ονομάζεται Fibonacci ή external LFSR και η δεύτερη Galois ή internal LFSR.

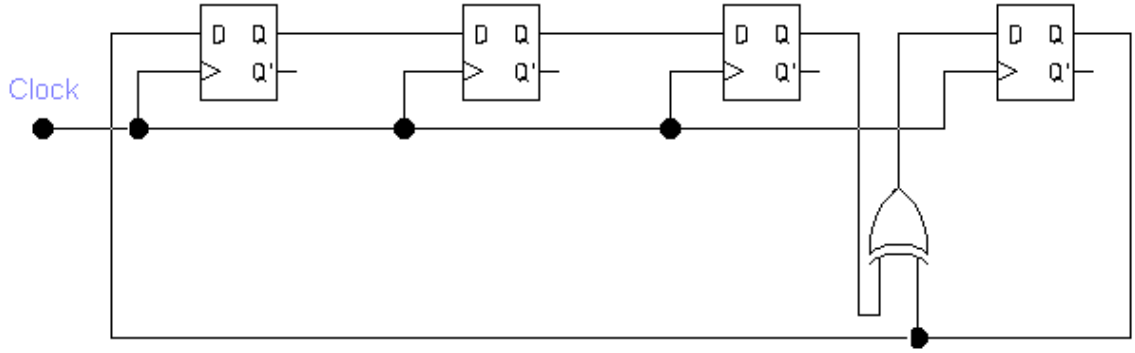
Στην παρουσίαση αυτής της πτυχιακής θα χρησιμοποιήσουμε τους όρους external και internal LFSR.

Το παρακάτω κύκλωμα δείχνει ένα 4bit external LFSR.



4bit external LFSR

Αντίθετα ένα 4bit internal LFSR θα είναι όπως φαίνεται στο παρακάτω σχήμα.



4bit internal LFSR

Θα πρέπει να διευκρινίσουμε ότι η σειρά των ψευδοτυχαίων ακολουθιών που λαμβάνουμε από ένα external και από ένα internal LFSR θα είναι εντελώς διαφορετική. Παραμένει ίδιος όμως ο αριθμός των εξαρτημάτων που χρειαζόμαστε για να υλοποιήσουμε ένα LFSR.

A (είσοδος)	B (είσοδος)	Y (έξοδος)
0	0	0
0	1	1
1	0	1
1	1	0

1.2 Πίνακας αληθείας της πύλης XOR

Επίσης ανάλογα με το ποιες είναι οι συνδέσεις των D Flip Flops με την πύλη XOR χωρίζουμε τα LFSR σε primitive ή non-primitive (πρωτόγονο ή μη πρωτόγονο).

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 (seed value) για να ξαναπάρει πάλι όλους τους δυνατούς συνδυασμούς κ.ο.κ.

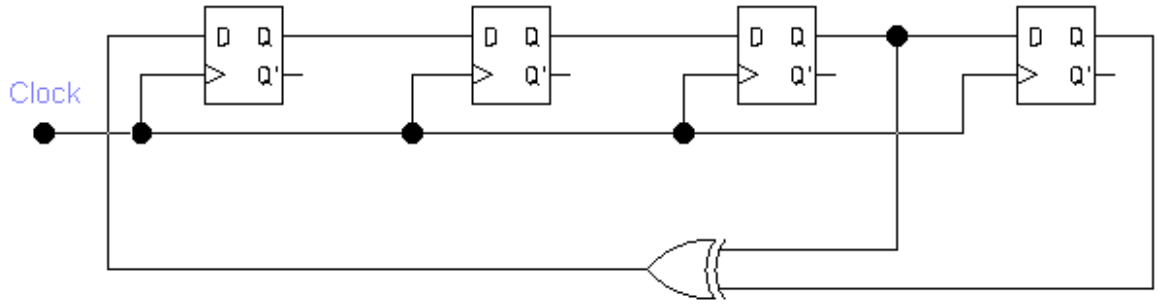
Non-primitive σημαίνει ότι δεν παίρνουμε όλους τους δυνατούς συνδυασμούς αλλά ένα μέρος αυτών ανάλογα με την αρχική τιμή (seed value). Αναφέρουμε για παράδειγμα ένα 4bit LFSR που κάνει ένα πλήρη κύκλο λειτουργίας σε 8 βήματα αντί για 15 που θα είχαμε αν ήταν primitive το πολυώνυμο.

Αυτό δεν σημαίνει ότι μόνο τα primitive LFSR είναι χρήσιμα. Υπάρχουν εφαρμογές για παράδειγμα που αν υλοποιούσαμε ένα 64bit LFSR θα κάναμε ώρες ολόκληρες για να ολοκληρώσουμε έναν πλήρη κύκλο λειτουργίας ενώ στην πράξη μας ενδιαφέρει ένα μέρος των ψευδοτυχαίων ακολουθιών που παράγονται.

Για να πάρουμε ένα μόνο μέρος των ψευδοτυχαίων ακολουθιών θα μπορούσαμε επίσης να περιορίσουμε τα βήματα (steps) που κάνει ο γεννήτορας συνδυασμών. Οι ψευδοτυχαίες ακολουθίες όμως δεν θα ήταν ίδιες με τον τρόπο του non-primitive LFSR. Επιπλέον για να αναπαραχθούν ξανά οι ίδιες ακολουθίες θα έπρεπε να ξαναβάλουμε στον γεννήτορα την ίδια αρχική τιμή με πριν κάτι που δεν χρειάζεται στον πρώτο τρόπο γιατί όταν ολοκληρωθεί ο κύκλος λειτουργίας θα υπάρχει η αρχική τιμή έτοιμη.

- *Πως δουλεύει το LFSR.*

Για να μπορέσουμε να κατανοήσουμε πως δουλεύει ένα LFSR θα δώσουμε ένα σχήμα όπου διακρίνεται ένα 4 bit external primitive LFSR.



4 bit external primitive LFSR

Εξετάζουμε το παραπάνω LFSR βήμα-βήμα και σημειώνουμε την έξοδό του για κάθε CP (Clock Pulse) στην είσοδο. Έτσι συμπληρώνουμε τον παρακάτω πίνακα. Βοηθητικά παρατηρούμε και την έξοδο της πύλης XOR (τελευταία στήλη στον πίνακα).

CP		FF4	FF3	FF2	FF1	XOR
Seed	**0**	**0**	**0**	**0**	**1**	**1**
	1	**1**	**0**	**0**	**0**	**0**
	2	**0**	**1**	**0**	**0**	**0**
	3	**0**	**0**	**1**	**0**	**1**
	4	**1**	**0**	**0**	**1**	**1**
	5	**1**	**1**	**0**	**0**	**0**
	6	**0**	**1**	**1**	**0**	**1**
	7	**1**	**0**	**1**	**1**	**0**
	8	**0**	**1**	**0**	**1**	**1**
	9	**1**	**0**	**1**	**0**	**1**
	10	**1**	**1**	**0**	**1**	**1**

11	**1**	**1**	**1**	**0**	**1**
12	**1**	**1**	**1**	**1**	**0**
13	**0**	**1**	**1**	**1**	**0**
14	**0**	**0**	**1**	**1**	**0**
15	**0**	**0**	**0**	**1**	**1**

Πίνακας 1.3

Στον παραπάνω πίνακα θέσαμε ως seed value (αρχική τιμή) την 0001 ή 1hex. Αφού τα 2 τελευταία ψηφία οδηγούνται στην είσοδο της πύλης XOR θα έχουμε σύμφωνα με τον πίνακα αληθείας της XOR, έξοδο «1». Άρα στο επόμενο CP η έξοδος του πρώτου D Flip Flop θα γίνει «1» κ.ο.κ.

Αυτό θα συνεχιστεί έως ότου περάσουν 15 παλμοί χρονισμού (Clock Pulses) ή 2^N-1 συνδυασμοί (primitive LFSR). Όπου N είναι ο αριθμός των Flip Flops που υπάρχουν στο κύκλωμα και το –1 γιατί αν βάλουμε seed value 0000 ή 0hex τότε το LFSR δεν θα ξεκινήσει ποτέ να μετράει γιατί η έξοδος της πύλης XOR θα είναι πάντα «0» **(ατέρμονος βρόγχος) Στην πράξη δεν βάζουμε ποτέ αρχική τιμή (seed value) 00 και επιθυμούμε να μην φτάσει ποτέ σ' αυτήν την τιμή γιατί ουσιαστικά διακόπτεται η λειτουργία του.**

- ***Υλοποίηση διάφορων πολυωνύμων***

Μπορούμε να υλοποιήσουμε πολλά διαφορετικά πολυώνυμα. Ο περιοριστικός παράγοντας εδώ είναι ότι θα πρέπει να χρησιμοποιήσουμε τόσα D Flip Flops όσο είναι και ο βαθμός του πολυωνύμου που θέλουμε να υλοποιήσουμε στο LFSR. Στον πίνακα 1.4 δείχνουμε μερικά primitive πολυώνυμα. Primitive πολυώνυμα δεν είναι όμως μόνο τα παρακάτω. Μερικοί συνδυασμοί πολυωνύμων (όπως το $1 \oplus x^3 \oplus x^4$) αποτελούν non-primitive πολυώνυμα[4].

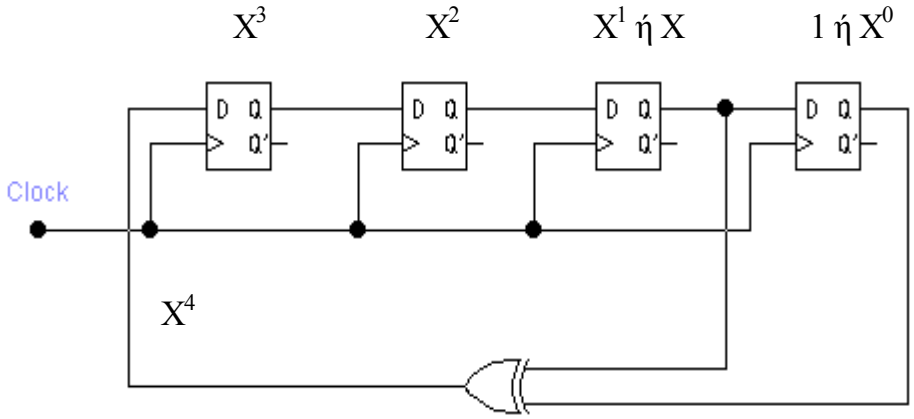
Βαθμός	Πολυώνυμο	**Βαθμός**	**Πολυώνυμο**
1	$1 \oplus X$	**17**	$1 \oplus X^3 \oplus X^{17}$
2	$1 \oplus X \oplus X^2$	**18**	$1 \oplus X^7 \oplus X^{18}$
3	$1 \oplus X \oplus X^3$	**19**	$1 \oplus X \oplus X^5 \oplus X^6 \oplus X^{19}$
4	$1 \oplus X \oplus X^4$	**20**	$1 \oplus X^3 \oplus X^{20}$
5	$1 \oplus X^2 \oplus X^5$	**21**	$1 \oplus X^2 \oplus X^{21}$
6	$1 \oplus X \oplus X^6$	**22**	$1 \oplus X \oplus X^{22}$
7	$1 \oplus X \oplus X^7$	**23**	$1 \oplus X^5 \oplus X^{23}$
8	$1 \oplus X^5 \oplus X^6 \oplus X^8$	**24**	$1 \oplus X^3 \oplus X^4 \oplus X^{24}$
9	$1 \oplus X^4 \oplus X^9$	**25**	$1 \oplus X^3 \oplus X^{25}$
10	$1 \oplus X^3 \oplus X^{10}$	**26**	$1 \oplus X \oplus X^7 \oplus X^8 \oplus X^{26}$
11	$1 \oplus X^2 \oplus X^{11}$	**27**	$1 \oplus X \oplus X^7 \oplus X^8 \oplus X^{27}$
12	$1 \oplus X^3 \oplus X^4 \oplus X^7 \oplus X^{12}$	**28**	$1 \oplus X^3 \oplus X^{28}$
13	$1 \oplus X \oplus X^3 \oplus X^4 \oplus X^{13}$	**29**	$1 \oplus X^2 \oplus X^{29}$
14	$1 \oplus X \oplus X^{11} \oplus X^{12} \oplus X^{14}$	**30**	$1 \oplus X \oplus X^{15} \oplus X^{16} \oplus X^{30}$
15	$1 \oplus X \oplus X^{15}$	**31**	$1 \oplus X^3 \oplus X^{31}$
16	$1 \oplus X^2 \oplus X^3 \oplus X^{15} \oplus X^{16}$	**32**	$1 \oplus X \oplus X^{27} \oplus X^{28} \oplus X^{32}$

Πίνακας 1.4 Μερικά primitive πολυώνυμα.

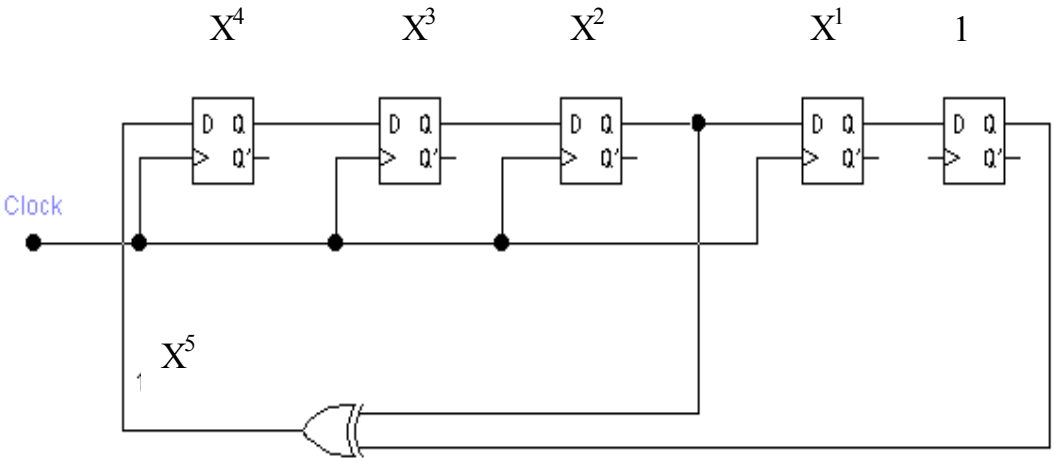
Υπάρχουν βέβαια primitive πολυώνυμα με βαθμό μεγαλύτερο από 32 bit αλλά δεν κρίθηκε αναγκαίο να αναφερθούν για τους σκοπούς αυτής της πτυχιακής.

Όπως έχουμε προαναφέρει υπάρχουν δυο τύποι LFSR. Ο πρώτος τρόπος με τις πύλες ανατροφοδότησης XOR εξωτερικά που ονομάζεται και Fibonacci και ο δεύτερος με τις πύλες ανατροφοδότησης XOR εσωτερικά που ονομάζεται και Galois.

Παρακάτω θα δούμε τον τρόπο που υλοποιούνται τα παραπάνω πολυώνυμα σε μορφή πυλών και Flip Flops. Πρώτα θα εξετάσουμε την υλοποίηση με εξωτερική πύλη XOR που είναι και η πιο συνηθισμένη. Δίνεται ένα παράδειγμα.

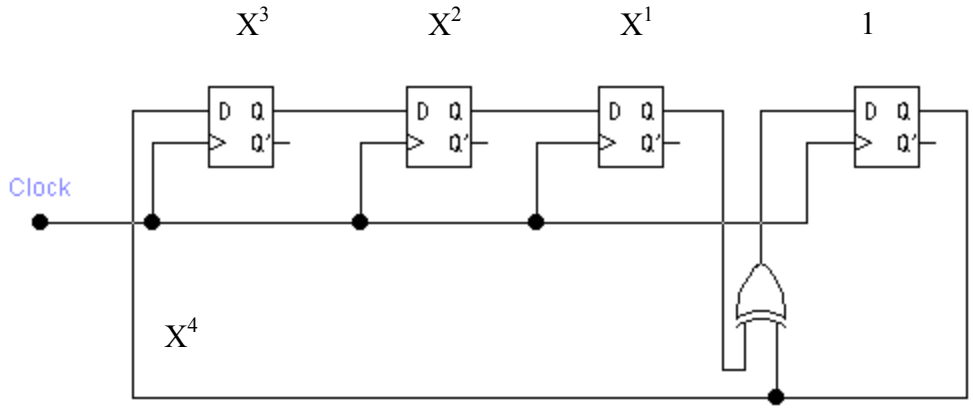


Το παραπάνω κύκλωμα είναι ένα εξωτερικό primitive LFSR με πολυώνυμο $1 \oplus X \oplus X^4$. Σκοπός είναι να συνδέουμε στην έξοδο του Flip Flop που δείχνει το πολυώνυμο μια πύλη XOR που θα ανατροφοδοτήσει (feedback) το κύκλωμα. Παρατηρούμε ότι η αρίθμηση των Flip Flop γίνεται στην σειρά. Από δεξιά προς αριστερά έχουμε το X^0 ή 1 που υπάρχει πάντα (αλλιώς δεν έχει ουσία να έχουμε συνδεμένο 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 Flops) μετράμε αθροίζοντας κατά ένα τους εκθέτες του X μέχρι το τέλος του. Για παράδειγμα δίνουμε το παρακάτω primitive LFSR $5^{ου}$ βαθμού $1 \oplus X^2 \oplus X^5$ που διαφέρει από το προηγούμενο $4^{ου}$ βαθμού.

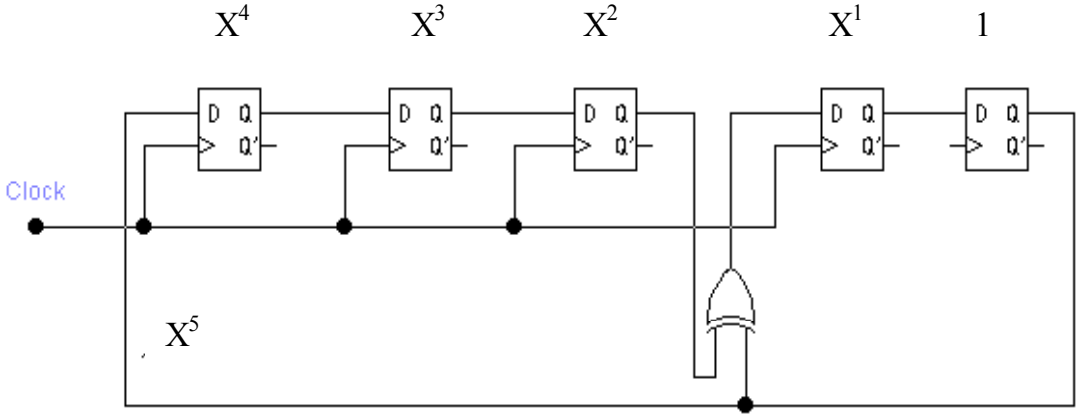


$5^{ου}$ βαθμού primitive external LFSR.

Τα ίδια ισχύουν και για εσωτερικά LFSR. Η μόνη διαφορά βέβαια είναι ότι οι πύλες XOR τοποθετούνται ανάμεσα στα Flip Flops του κυκλώματος. Παρακάτω δίνονται τα αντίστοιχα internal LFSR για τις παραπάνω περιπτώσεις.

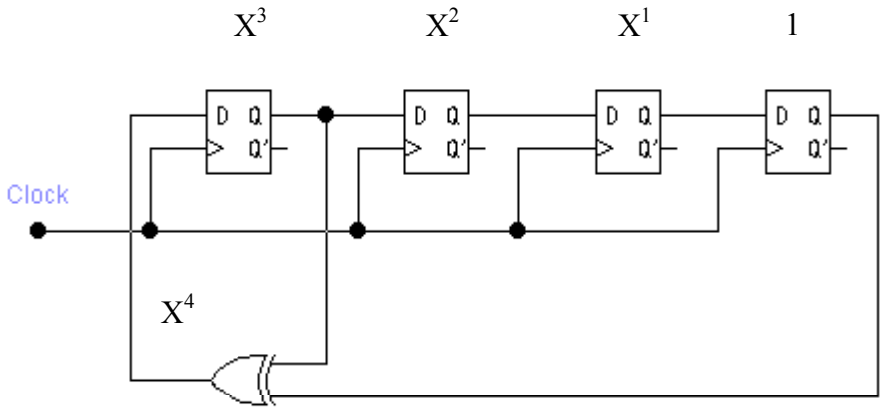


4ου βαθμού primitive internal LFSR.



5ου βαθμού primitive internal LFSR.

Όπως γίνεται κατανοητό δεν παίζει κανένα ρόλο αν το LFSR θα είναι primitive ή non-primitive. Το παρακάτω κύκλωμα είναι non-primitive external LFSR με πολυώνυμο $1 \oplus X^3 \oplus X^4$ (ισχύουν τα ίδια για internal LFSR). Όπως βλέπουμε ακολουθεί τον ίδιο τρόπο για να μετατραπεί από πολυώνυμο σε κύκλωμα με πύλες και Flip Flop.



4ου βαθμού non-primitive external LFSR.

Επειδή είναι σχετικά δύσκολο και χρονοβόρο να υπολογίσουμε τις εξόδους ενός 8 bit primitive LFSR (255 συνδυασμοί) με το χέρι αναπτύξαμε ένα πρόγραμμα γραμμένο σε Visual Basic που μπορεί να παράγει, οπτικά αλλά και πρακτικά με την βοήθεια της παράλληλης πόρτας του υπολογιστή, τις εξόδους ενός τέτοιου LFSR. Εκεί έχουμε την δυνατότητα να βλέπουμε βήμα-βήμα τις εξόδους των D Flip Flops ή να βάλουμε το πρόγραμμα σε λειτουργία «auto generate**» όπου γρηγορότερα πλέον παράγονται οι έξοδοι ανάλογα με την χρονική σταθερά που έχουμε ορίσει (interval value). Επίσης έχουμε την δυνατότητα να συνδέσουμε στην παράλληλη θύρα του υπολογιστή κυκλώματα για προσομοίωση της λειτουργίας ενός LFSR.**

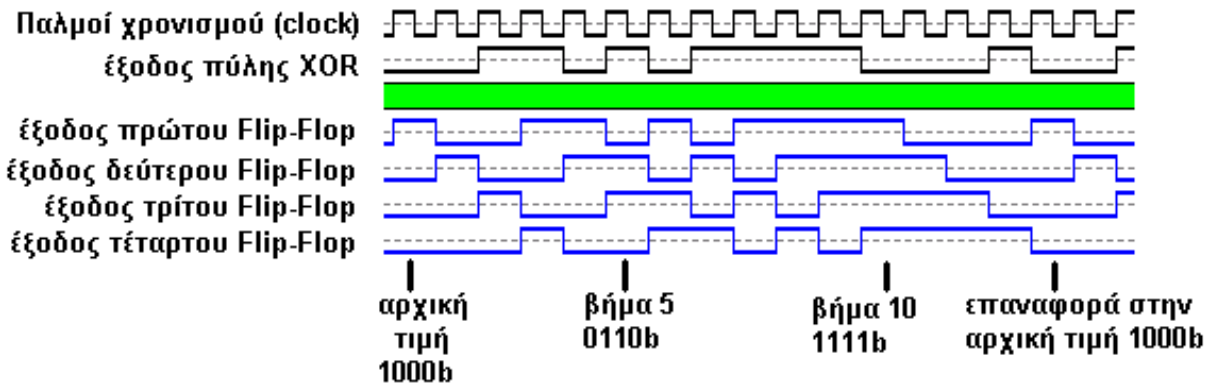
Το πρόγραμμα αυτό με την ονομασία «LFSR implementation**» βρίσκεται στο CD που συνοδεύει αυτήν την πτυχιακή.**

- *LFSR σαν Pattern Generator.*

Ένα LFSR λέμε ότι λειτουργεί σαν Pattern Generator (γεννήτορας συνδυασμών) **όταν έχει ως είσοδο παλμούς χρονισμού και αρχική τιμή (seed value) και σαν έξοδο την έξοδο ενός D Flip Flop. Στην περίπτωση που η έξοδος του είναι οι έξοδοι όλων των D Flip Flop του LFSR τότε έχουμε ένα Parallel Pattern Generator[4].**

Η έξοδος του Pattern Generator ονομάζεται ψευδοτυχαία ακολουθία και εξαρτάται άμεσα από το πολυώνυμο που έχουμε επιλέξει.

Σαν παράδειγμα Pattern Generator δίνουμε το παρακάτω LFSR και την έξοδο του γραφικά. Χαρακτηριστικά:
External Parallel Pattern Generator primitive 4bit με αρχική τιμή 8hex ή «1000b» για έναν πλήρη κύκλο, είναι το παρακάτω.



Αποτελέσματα των παράλληλων εξόδων του γραφικά

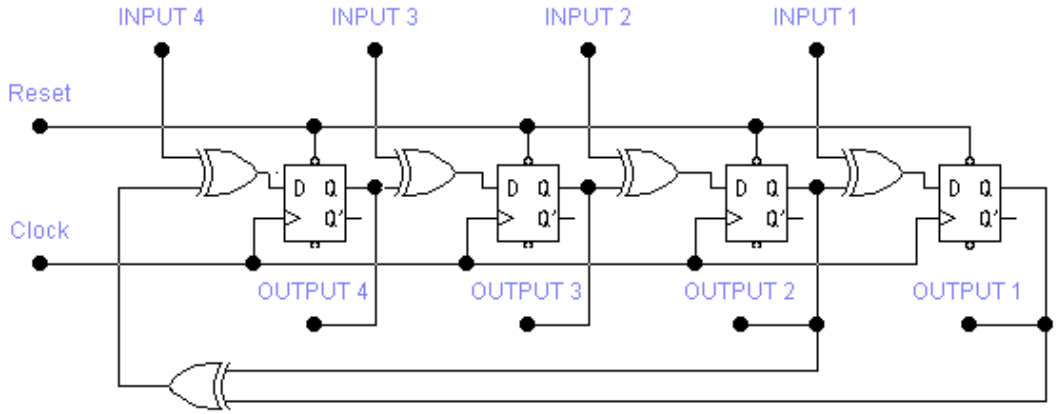
Βοηθητικά και πάλι δίνουμε την έξοδο της πύλης XOR του κυκλώματος. Το κυκλωματικό διάγραμμα που αντιστοιχεί στο παραπάνω γραφικό διάγραμμα είναι το σχήμα της υποενότητας «Πως δουλεύει το LFSR». Η έξοδος του πρώτου Flip Flop είναι η έξοδος του FF1 στο κυκλωματικό διάγραμμα κτλ.

Το παραπάνω γραφικό διάγραμμα έγινε με την χρήση του πακέτου «Xilinx Foundation 3.1». Περισσότερα για αυτό στην ενότητα 3.Σχεδίαση σε VHDL.

- *LFSR σαν Signature Analyser.*

Ένα LFSR λειτουργεί ως Signature Analyser (αναλυτής υπογραφής) όταν έχει σαν είσοδο παλμούς χρονισμού και είσοδο δεδομένων και σαν έξοδο τις

εξόδους των D Flip Flop ενός LFSR. Στην περίπτωση που έχουμε πολλές παράλληλες εισόδους τότε λέμε ότι έχουμε έναν Parallel Signature Analyser PSA (παράλληλος αναλυτής υπογραφής). Ένας 4 bit PSA φαίνεται παρακάτω.



4ου βαθμού primitive external Parallel Signature Analyser.

Παρατηρούμε ότι ανάμεσα στα Flip Flops βάζουμε πύλες XOR με την βοήθεια των οποίων έχουμε παράλληλη είσοδο στο κύκλωμα. Αυτές οι πύλες δεν αποτελούν μέρος της ανατροφοδότησης. Η κάτω πύλη του παραπάνω σχήματος αποτελεί την ανατροφοδότηση (feedback) του κυκλώματος. Έτσι μπορούμε να δούμε ότι ο PSA είναι 4ου βαθμού (4 bit) external και με πολυώνυμο λειτουργίας $1 \oplus X \oplus X^4$.

Ένας PSA πρέπει να έχει οπωσδήποτε μια γραμμή reset (επαναφοράς) και clock (χρονισμού). Μέσω της γραμμής χρονισμού γίνεται σειριακή ολίσθηση όπως συμβαίνει και στον Pattern Generator με την μόνη διαφορά ότι εδώ έχουμε ολίσθηση ανάλογα με τα δεδομένα εισόδου (γραμμές INPUT 1 έως INPUT 4).

Ο PSA έχει σκοπό να μετατρέψει τα δεδομένα που λαμβάνει στην είσοδο του, σε μια ψηφιακή υπογραφή στην έξοδο του. Η υπογραφή αυτή αντιπροσωπεύει τα δεδομένα που ελήφθησαν στην είσοδο για τους τελευταίους παλμούς χρονισμού[1].

Εδώ πρέπει να εισάγουμε και την έννοια του alias (ψευδώνυμο). Aliases έχουμε όταν τα δεδομένα στην είσοδο είναι διαφορετικά αλλά τυχαίνει να έχουν την ίδια ψηφιακή υπογραφή στην έξοδο. Αυτό συμβαίνει από διαδοχικές ακυρώσεις λαθών που πρέπει να είναι περισσότερο από μια για να έχουμε aliasing. Για παράδειγμα 2+3+4+6+2=17 όπως επίσης και 2+3+3+7+2=17. Παρατηρούμε ότι ενώ το τρίτο και τέταρτο ψηφίο έχουν αλλάξει το αποτέλεσμα παραμένει ίδιο.

Το φαινόμενο αυτό στην πράξη προσπαθούμε πάντα να το αποφύγουμε διαλέγοντας ένα άλλο πολυώνυμο λειτουργίας για τον PSA ή ελέγχουμε το αποτέλεσμα συγκρίνοντάς το με μια γνωστή τιμή μας ανά διαστήματα. Στην διεθνή βιβλιογραφία υπάρχει εκτεταμένη μελέτη για τις πιθανότητες ύπαρξης Alias σε ένα LFSR. Πολλές φορές μάλιστα υπάρχει διαμάχη για το ποιος τύπος LFSR δίνει τα λιγότερα Alias (primitive ή non-primitive)[5].

Εμείς δεν θα αναφερθούμε στις μελέτες αυτές γιατί ξεφεύγουμε από τους σκοπούς αυτής της πτυχιακής. Για κάποιον που πιθανότατα ενδιαφέρεται για αυτές τις μελέτες υπάρχει υλικό σε ψηφιακή μορφή στο CD που συνοδεύει την πτυχιακή.

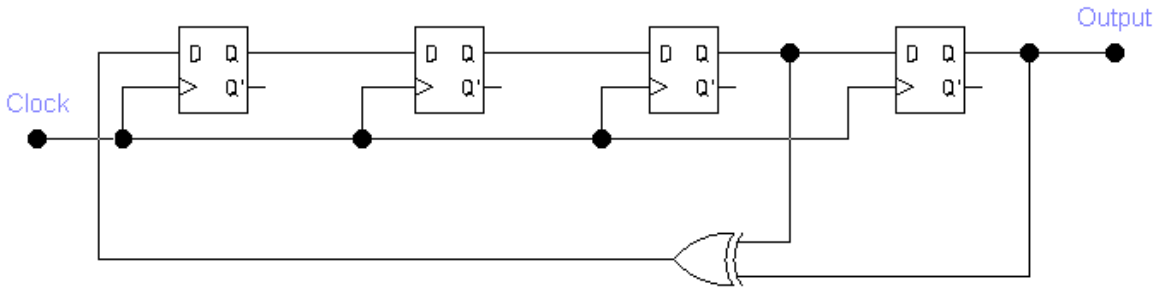
Οι PSA έχουν πολυώνυμο λειτουργίας αλλά δεν έχουν αρχική τιμή. Θα πρέπει όμως υποχρεωτικά να μηδενίζουμε τα δεδομένα που υπάρχουν αποθηκευμένα στους καταχωρητές ολίσθησης γιατί αλλιώς δεν θα έχουμε γραμμικότητα στα αποτελέσματα που λαμβάνουμε. Αυτό επιτυγχάνεται με γραμμή reset στα D Flip Flop.

- *Εφαρμογές των LFSR.*

Οι εφαρμογές των LFSR ποικίλουν. Οι χρήσεις τους είναι πολλές αλλά πολύ λίγες είναι ευρέως γνωστές. Τα LFSR συνήθως αποτελούν υποσύστημα από κάποια περιφερειακή μονάδα.

Η πιο ενδιαφέρουσα ίσως εφαρμογή είναι η χρήση των LFSR στους σύγχρονους σκληρούς δίσκους. Εδώ ένας PSA (Parallel Signature Analyzer) αντικαθιστά τον ελεγκτή ισοτιμίας (Parity Checker) που χρησιμοποιούν για παράδειγμα οι παλιότερες σειριακές θύρες των υπολογιστών. Με αυτόν τον τρόπο έχουμε έναν πολύ πιο αξιόπιστο σύστημα ελέγχου για το αν τα δεδομένα που φεύγουν από τον σκληρό δίσκο προς την κύρια μονάδα επεξεργασίας και το αντίθετο, έχουν φτάσει στον προορισμό τους χωρίς αλλοίωση. Αυτή η χρήση παίρνει ακόμα μεγαλύτερη σημασία στους Serial ATA σκληρούς δίσκους που θα βγουν στην παραγωγή σε μερικά χρόνια. Η μεταφορά των δεδομένων σ' αυτούς θα γίνεται σειριακά και όχι παράλληλα όπως οι σημερινοί σκληροί δίσκοι γλιτώνοντας καλώδια διασύνδεσης και φυσικά τις απώλειες που έχουμε με αυτά. Μάλιστα υπολογίζεται ότι η ταχύτητα μετάδοσης δεδομένων θα είναι 33% περισσότερη από τους σημερινούς σκληρούς δίσκους[19].

Μια άλλη εφαρμογή των LFSR είναι η χρήση τους στο Αμερικάνικο σύστημα κινητής τηλεφωνίας CDMA (Code Division Multiple Access). Το σύστημα αυτό αναμένεται να αποδεχθεί και η Ευρώπη γιατί έτσι έχουμε καλύτερη αξιοποίηση του διαθέσιμου εύρους ζώνης και μεγαλύτερη ταχύτητα μετάδοσης δεδομένων συγκριτικά με το πρωτόκολλο GSM900 και GSM1800. Αναλυτικότερα με την χρήση του παρακάτω κυκλώματος και θέτοντας μια αρχική τιμή (seed value) παράγουμε στην έξοδο του κυκλώματος μια ψευδοτυχαία σειριακή ακολουθία[3].



Κύκλωμα παραγωγής κωδίκων.

Όπως φαίνεται και από τον πίνακα 1.3 η έξοδος του κυκλώματος παραγωγής κωδίκων (ουσιαστικά η έξοδος του FF4) θα είναι «1000100110101111». Από τον κώδικα αυτόν εξαρτάται για παράδειγμα το ποια χρονοθυρίδα θα καταλάβει ή σε ποια συχνότητα θα λειτουργήσει το κινητό τηλέφωνο (κινητός σταθμός) για να επικοινωνήσει με κάποιον άλλον χρήστη. Μάλιστα τα κινητά τηλέφωνα τρίτης γενιάς που ήδη κυκλοφορούν πειραματικά στο εμπόριο κάνουν χρήση αυτής της τεχνολογίας.

Τέλος θα αναφερθούμε στην εφαρμογή που αναλάβαμε να περατώσουμε εμείς. Πρόκειται γενικά για μια συσκευή στην οποία συνδέουμε ένα ψηφιακό κύκλωμα, από ένα απλό ολοκληρωμένο μέχρι ένα συνδυασμό ολοκληρωμένων, και ελέγχουμε αν η υπό έλεγχο συσκευή (circuit under test) **έχει βλάβη ή όχι. Απαραίτητη προϋπόθεση είναι να έχουμε συνδέσει από πριν μια συσκευή που σίγουρα λειτουργεί σωστά. Η συσκευή αυτή επικοινωνεί μέσω του δίαυλου USB με τον υπολογιστή με την βοήθεια του οποίου γίνεται ο**

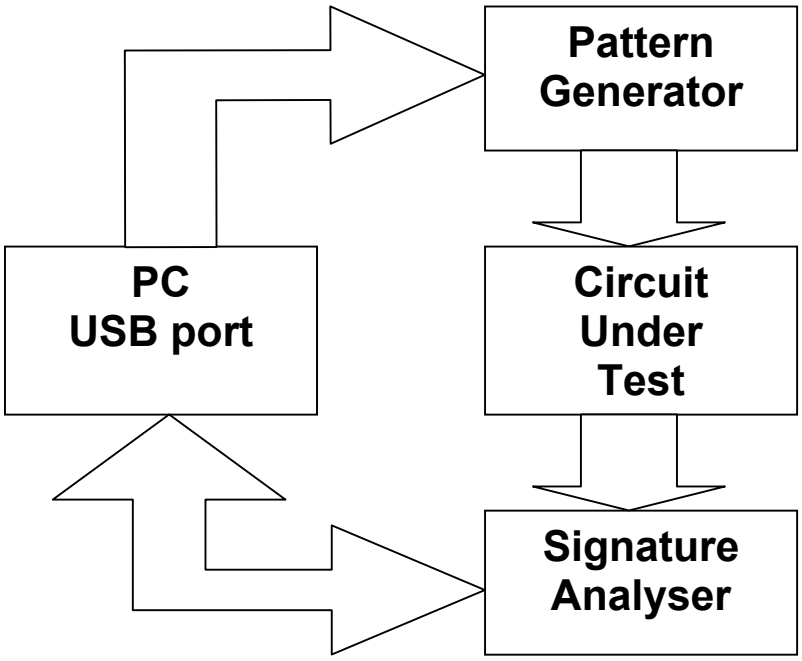
προγραμματισμός της και η εμφάνιση των αποτελεσμάτων. Περισσότερα όμως για την εφαρμογή μας στο επόμενο κεφάλαιο.

2. Η εφαρμογή του LFSR.

- ***Έλεγχος ψηφιακών κυκλωμάτων με LFSR.***

Όπως προαναφέραμε εν συντομία στην προηγούμενη ενότητα, η δική μας εφαρμογή αποτελεί ένα σταθμό ελέγχου καλής λειτουργίας ψηφιακών κυκλωμάτων. Συγκεκριμένα πρόκειται για μια συσκευή στην οποία συνδέουμε ένα ψηφιακό κύκλωμα, από ένα απλό ολοκληρωμένο μέχρι ένα συνδυασμό ολοκληρωμένων, και ελέγχουμε αν η υπό έλεγχο συσκευή (circuit under test) έχει βλάβη ή όχι.

Απαραίτητη προϋπόθεση είναι να έχουμε συνδέσει από πριν μια συσκευή που σίγουρα λειτουργεί σωστά. Η συσκευή αυτή επικοινωνεί μέσω του δίαυλου USB με τον υπολογιστή με την βοήθεια του οποίου γίνεται ο προγραμματισμός της και η εμφάνιση των αποτελεσμάτων.



Μπλοκ διάγραμμα της εφαρμογής μας.

Μέσω του διαύλου USB προγραμματίζουμε και θέτουμε σε λειτουργία την συσκευή. Επίσης παρέχεται η δυνατότητα να εισάγουμε στον υπολογιστή τα δεδομένα που παρήγαγε ο Signature Analyzer για αποθήκευση και περαιτέρω επεξεργασία.

Με κατάλληλο προγραμματισμό θα μπορούσαμε να δημιουργήσουμε μια βάση δεδομένων για όλα τα γνωστά ψηφιακά ολοκληρωμένα κυκλώματα όπως πύλες, καταχωρητές, πολυπλέκτες, κωδικοποιητές κα. Με τον τρόπο αυτό θα αρκούσε να διαλέξουμε από μια λίστα τον κωδικό του ολοκληρωμένου και να μην είναι απαραίτητο να συνδέσουμε πρώτα μια συσκευή που λειτουργεί σωστά. Ο τρόπος αυτός δεν θα ίσχυε στην περίπτωση που η συσκευή που εξετάζουμε είναι συνδυασμός ψηφιακών ολοκληρωμένων.

- ***Προϋποθέσεις που θέσαμε***

Σκοπός μας εξ αρχής ήταν να κατασκευάσουμε ένα προγραμματιζόμενο Pattern Generator. Αυτό επεκτάθηκε στην συνέχεια σε ελεγκτή ψηφιακών κυκλωμάτων και έτσι ήταν απαραίτητη η προσθήκη ενός Signature Analyzer.

Με τον όρο προγραμματιζόμενο εννοούμε ότι θα μπορεί να αλλάζει πολυώνυμο λειτουργίας και αρχικές τιμές χωρίς να είναι απαραίτητο να κάνουμε αλλαγές στο υλικό (Hardware) μέρος της συσκευής μας (εισαγωγή νέων εξαρτημάτων, αφαίρεση κάποιων που δεν είναι απαραίτητα ή έλεγχος μηχανικών διακοπτών κτλ). Άντ' αυτού θα πρέπει με κάποιον τρόπο να εισάγουμε αυτές τις τιμές στη συσκευή. Ο μόνος τρόπος που έμενε ήταν να χρησιμοποιήσουμε ηλεκτρονικό υπολογιστή. Έτσι ξεκινήσαμε να βρούμε ποιος είναι ο ποιο κατάλληλος τρόπος για να γίνει η επικοινωνία με την συσκευή μας.

Καταρχήν αποκλείστηκαν όλοι οι ασύρματοι τρόποι μετάδοσης δεδομένων (infrared, bluetooth κτλ) καθώς δεν υπάρχει κανένας λόγος για ασύρματη μετάδοση και είναι κάτι που αυξάνει το κόστος κατασκευής. Άλλωστε η συσκευή πρέπει να είναι πάντα κοντά στον υπολογιστή ώστε ο χειριστής να αλλάζει τα προς εξέταση κυκλώματα και να θέτει ξανά την συσκευή σε λειτουργία.

Από ενσύρματους τρόπους μετάδοσης υπάρχει μεγάλη γκάμα διαύλων. Για παράδειγμα (σειριακή θύρα, παράλληλη, PCI, ISA και άλλες). Από όλες αυτές εμείς διαλέξαμε την θύρα USB που στην ουσία αποτελεί αναβάθμιση της παλαιότερης RS-232. Αυτό έγινε αφενός γιατί είναι η πιο ενδεδειγμένη για συνδέσεις περιφερειακών μικρής αλλά και μεγάλης ταχύτητας και αφετέρου για να αποδείξουμε ότι δεν είναι τόσο περίπλοκη μια κατασκευή με χρήση αυτής της θύρας. Επίσης με αυτόν τον τρόπο η συσκευή μας αποκτά μια διαχρονικότητα επειδή η παλαιότερη σειριακή RS-232 αλλά και η παράλληλη οδεύουν σε κατάργηση. Ήδη οι καινούργιες μητρικές υπολογιστών πωλούνται με την σειριακή θύρα προαιρετική και οι καινούργιοι εκτυπωτές έχουν καταργήσει την παράλληλη θύρα.

Αν δεν ήταν οι παραπάνω λόγοι η πλέον κατάλληλη θύρα για μια τέτοια εφαρμογή θα ήταν η παράλληλη. Αυτό γιατί δεν είναι απαραίτητο να υπάρχει μικροελεγκτής πάνω στην συσκευή που αναπτύσσουμε κάτι που την κάνει πιο οικονομική λύση και πιο αξιόπιστη. Ούτως ή άλλως η απόσταση του υπολογιστή με το περιφερειακό δεν είναι ανάγκη να υπερβαίνει τα 5 μέτρα που είναι σχεδιασμένη να λειτουργεί η παράλληλη θύρα χωρίς προβλήματα.

Είναι γεγονός ότι η θύρα USB είναι πολύ γρηγορότερη από την σειριακή και την παράλληλη. Δεν ήταν όμως ο λόγος αυτός καθοριστικός στην επιλογή μας. Ξέραμε εκ των προτέρων τις συχνότητες λειτουργίας που επιθυμούσαμε από τον Pattern Generator. Για να παράγουμε ένα παλμό χρονισμού clock της τάξεως του 1KHz μπορούσαμε να χρησιμοποιήσουμε ακόμα και την RS - 232 που είναι η πιο αργή από όλες τις άλλες ενσύρματες θύρες στον υπολογιστή.

Τέλος, ένας λόγος που επιλέξαμε τον δίαυλο USB ήταν το γεγονός ότι αυτός είναι bi-directional (είσοδος και έξοδος από την ίδια ενσύρματη γραμμή). Έτσι θα μπορούσε να γίνει ευκολότερα εισαγωγή των δεδομένων που παράγει ο Signature Analyzer από ότι στην περίπτωση της παράλληλης θύρας.

Αφού επιλέξαμε την θύρα που θα χρησιμοποιήσουμε θέσαμε τα απαιτούμενα χαρακτηριστικά λειτουργίας. Έτσι αποφασίσαμε ο Pattern Generator αλλά και ο Signature Analyzer να είναι 8bit. Δηλαδή ο μέγιστος

δυνατός βαθμός πολυωνύμου για τον Pattern Generator είναι $8^{ου}$ βαθμού. Το ίδιο ισχύει και για το Signature Analyzer. Με αυτόν τον τρόπο μπορούμε να ελέγχουμε 8bits ψηφιακά κυκλώματα. Για παράδειγμα αν θέλουμε να ελέγξουμε 8 πύλες NOT δεν έχουμε παρά να τις συνδέσουμε στο κύκλωμα και να το βάλουμε σε λειτουργία. Αν όμως είχαμε 4bit συσκευή και θέλαμε να ελέγξουμε τις παραπάνω πύλες NOT θα έπρεπε να τεμαχίσουμε το συνολικό κύκλωμα στην μέση (δυο 4bits πύλες) και να θέσουμε την συσκευή μας δύο φορές σε λειτουργία.

Στην περίπτωση που το κύκλωμα που εξετάζουμε είναι τέσσερις πύλες AND για παράδειγμα, γίνεται αντιληπτό ότι θα χρησιμοποιηθούν 8 έξοδοι από τον Pattern Generator από τις 8 που διαθέτει και 4 είσοδοι από τον Signature Analyzer από τις 8 που διαθέτει. Σ' αυτήν την περίπτωση καλό είναι οι υπόλοιπες είσοδοι που δεν χρησιμοποιούνται να οδηγηθούν στην γείωση του κυκλώματος.

Μια από τις προϋποθέσεις που θέσαμε ήταν και ο τρόπος τροφοδοσίας της συσκευής μας και της συσκευής που επιθυμούμε να ελέγξουμε. Από το δεύτερο γίνεται φανερό ότι δεν είναι δυνατόν να λογαριάσουμε το συνολικό ρεύμα που απαιτείται. Ξέρουμε όμως το ρεύμα που χρειάζεται η δική μας συσκευή. Έχουμε 17 ολοκληρωμένα τεχνολογίας CMOS και μπορούμε να πούμε ότι χρειαζόμαστε περίπου:

$$17 * 10mA = 170mA$$

Αν προσθέσουμε και το ρεύμα των LED που τοποθετήσαμε στην συσκευή για λόγους καλύτερου ελέγχου της λειτουργίας της έχουμε ότι:

$$(16 * 20mA) + 170mA = 490mA$$

Συνολικά χρειαζόμαστε 490mA στην περίπτωση όμως που όλα τα ολοκληρωμένα δουλεύουν και όλα τα LED είναι σε κατάσταση ON (φωτοβολούν). Επειδή αυτό δεν συμβαίνει ποτέ ή συμβαίνει εξαιρετικά σπάνια η συνολική κατανάλωση θα είναι αρκετά μικρότερη.

Σε όλα αυτά βέβαια δεν συνυπολογίσαμε την κατανάλωση του κυκλώματος υπό εξέταση (Circuit Under Test). Σε κανονικές συνθήκες το απαιτούμενο ρεύμα είναι από 10mA έως 200mA αν υπολογίσουμε για ένα κύκλωμα με 20 ολοκληρωμένα. Υπάρχει πάντα περίπτωση το κύκλωμα υπό εξέταση να διαθέτει ρελέ ή ενδεικτικές λυχνίες ή LED και έτσι το ρεύμα που απαιτείται να είναι αρκετά μεγαλύτερο των 200mA. Συνολικά λοιπόν θα πρέπει να διαθέτουμε ένα τροφοδοτικό περίπου 1A.

Το ρεύμα που χρειαζόμαστε θα μπορούσαμε να το «πάρουμε» από τον δίαυλο USB. Στην δική μας περίπτωση όμως δεν είναι δυνατόν να μας δώσει τόσο μεγάλο ρεύμα. Όπως αναφέρεται και στην ενότητα 4 (Ο δίαυλος USB) το μέγιστο ρεύμα που μπορεί να προέλθει από τον δίαυλο είναι περίπου 100mA. Με ειδικές εντολές όμως μπορούμε να πάρουμε μέχρι και 500mA. Αφού και αυτός ο τρόπος δεν αρκεί η μόνη λύση που μένει είναι να τοποθετήσουμε στην συσκευή μας δικό της ανεξάρτητο τροφοδοτικό και να αφήσουμε ανεκμετάλλευτο το ρεύμα που μας δίνει ο δίαυλος.

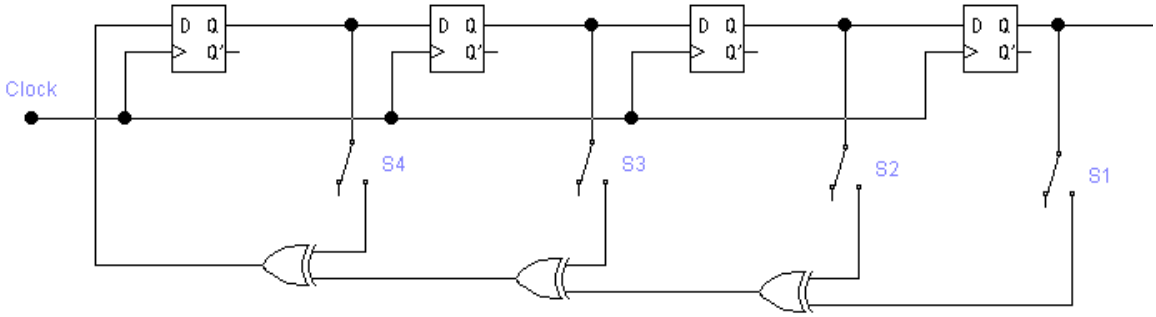
Τέλος καταλήξαμε στο να χρησιμοποιήσουμε εξωτερικό LFSR, και για τον Pattern Generator και για τον Signature Analyzer, γιατί με αυτόν τον τρόπο γίνεται πολύ πιο εύκολη η διαδικασία της επιλογής του πολυωνύμου όπως θα δούμε στην επόμενη υποενότητα.

- ***Πως γίνεται η επιλογή του πολυωνύμου***

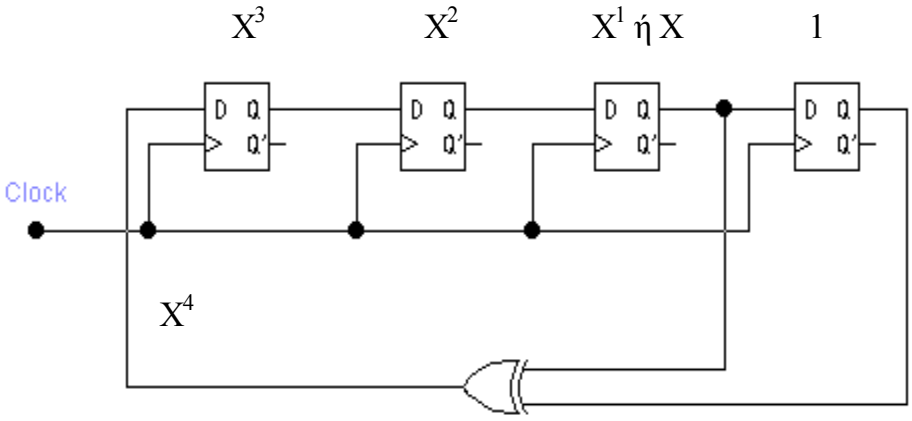
Για να μπορέσουμε να κάνουμε το LFSR προγραμματιζόμενο θα πρέπει να βρούμε έναν μηχανισμό με τον οποίο κάποια δεδομένα από τον δίαυλο USB θα αποθηκεύονται προσωρινά στην συσκευή και θα επενεργούν σε κάποια κυκλώματα.

Ο μόνος γνωστός και εύκολος τρόπος αποθήκευσης ψηφιακών δεδομένων που γνωρίζουμε είναι οι καταχωρητές ολίσθησης. Θα μπορούσαμε βέβαια να χρησιμοποιήσουμε μνήμες αλλά στην συγκεκριμένη περίπτωση δεν μας ενδιαφέρει το μέγεθος του αποθηκευτικού χώρου. Μας ενδιαφέρει περισσότερο η ευκολία πρόσβασης στην μνήμη (read / write).

Ουσιαστικά αυτό που προσπαθούμε να πετύχουμε είναι αυτό που φαίνεται στο παρακάτω σχήμα μόνο που αντί για μηχανικούς διακόπτες να χρησιμοποιήσουμε ηλεκτρονικούς διακόπτες.



Στο παράδειγμα με τους μηχανικούς διακόπτες βλέπουμε ότι είναι κλειστοί (ON) οι διακόπτες S1 και S2 τότε έχουμε το primitive πολυώνυμο λειτουργίας $1 \oplus X \oplus X^4$ όπως είδαμε από την προηγούμενη ενότητα. Δηλαδή το ισοδύναμο κύκλωμα θα είναι:



Οι άλλοι διακόπτες που είναι σε κατάσταση (OFF), ανοικτοί δεν επηρεάζουν το συνολικό κύκλωμα γιατί η είσοδος στην πύλη XOR είναι λογικό «0». Όπως έχουμε δει στην ενότητα 1 από τον πίνακα αληθείας της πύλης

XOR όταν μια είσοδος της είναι «0» τότε είναι σαν να μην υπάρχει η πύλη γιατί ότι έρθει στην άλλη είσοδο περνάει και στην έξοδο.

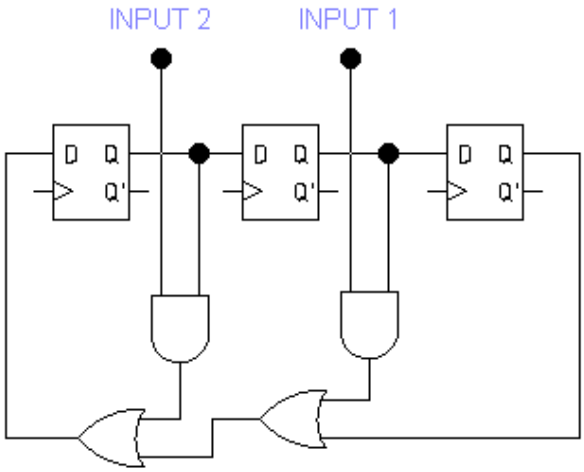
Αντί για μηχανικούς διακόπτες αυτό που κάναμε ήταν να χρησιμοποιήσουμε πύλες AND. Για βοήθεια δίνουμε παρακάτω τον πίνακα αληθείας της πύλης AND:

A (είσοδος)	B (είσοδος)	Y (έξοδος)
0	0	0
0	1	0
1	0	0
1	1	1

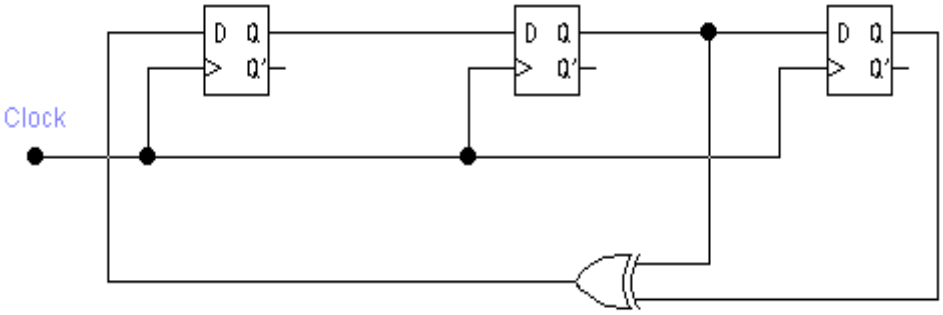
Πίνακας αληθείας της πύλης AND

Άλλος τρόπος για αλλαγή του μηχανικού διακόπτη με ηλεκτρονικό θα ήταν αν χρησιμοποιούσαμε ρελέ ή τρανζίστορ σε συνδεσμολογία διακόπτη. Η πρώτη λύση βέβαια απορρίφθηκε αμέσως λόγο της υψηλής απαίτησης σε ρεύμα αλλά και της ασυμβατότητας των ψηφιακών κυκλωμάτων με ηλεκτρομηχανικά μέρη (διαφορετικές τάσεις λειτουργίας, χρόνους ON-OFF κτλ). Η λύση των τρανζίστορ δεν ήταν τόσο ανέφικτη. Ωστόσο προτιμήθηκε η λύση της πύλης AND λόγο της πλήρης συμβατότητας με τα υπόλοιπα ολοκληρωμένα πάνω στην συσκευή.

Για να κατανοήσουμε πλήρως πως γίνεται ο ηλεκτρονικός έλεγχος των διακοπτών δίνουμε το σχήμα :

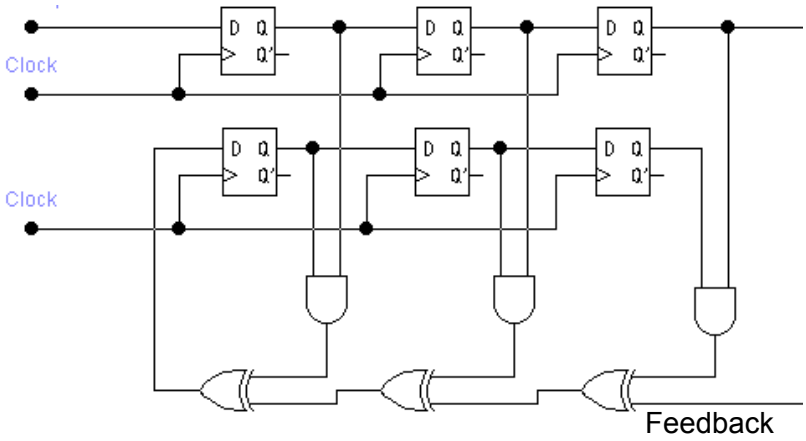


Για παράδειγμα αν στον ακροδέκτη INPUT 1 δώσουμε λογικό «1» και στον ακροδέκτη INPUT 2 λογικό «0» θα έχουμε σαν αποτέλεσμα το ισοδύναμο κύκλωμα του παρακάτω σχήματος που είναι ένας Pattern Generator $3^{ου}$ βαθμού με πολυώνυμο λειτουργίας $1 \oplus X \oplus X^3$.



Αυτό συμβαίνει γιατί αν δούμε τον πίνακα αληθείας της πύλης AND όταν δώσουμε λογικό «1» στην μια είσοδο τότε στην έξοδο θα έχουμε ότι υπάρχει στην δεύτερη είσοδο. Ουσιαστικά είναι σαν μην υπάρχει όπως βλέπουμε στο ισοδύναμο κύκλωμα.

Μέχρι τώρα λοιπόν το πρόβλημα περιορίζεται στο πως μπορούμε να μεταφέρουμε από τον υπολογιστή μέσω του διαύλου USB λογικές στάθμες στις εισόδους INPUT 1 και INPUT 2. Αν το συνδυάσουμε αυτό με τις μνήμες που προαναφέραμε σε μορφή shift register (καταχωρητή ολίσθησης) που στην ουσία αποτελούν έναν σειριακό σε παράλληλο μετατροπέα θα έχουμε:



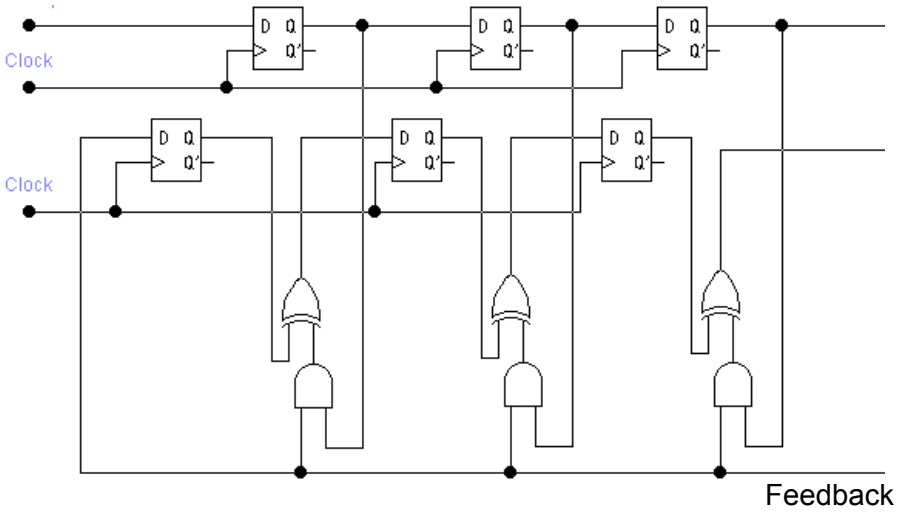
Βλέπουμε έτσι ότι οι είσοδοι INPUT 1 και INPUT 2 αποτελούν έξοδο του πάνω καταχωρητή ολίσθησης. Έτσι αν γεμίσουμε τον πάνω καταχωρητή ολίσθησης σειριακά με την βοήθεια παλμών χρονισμού από τον δίαυλο USB θα έχουμε στην ουσία προγραμματίσει τον Pattern Generator για ένα συγκεκριμένο πολυώνυμο λειτουργίας. Ο κάτω καταχωρητής ολίσθησης αποτελεί μέρος του LFSR.

Εδώ θα δώσουμε δύο παραδείγματα:

1. Ας υποθέσουμε ότι εισάγουμε την τιμή 5 σειριακά στον πάνω καταχωρητή ολίσθησης. Έτσι θα έχουμε ότι στο INPUT 0 = λογικό «1» , INPUT 1 = λογικό «0» και INPUT 2 = λογικό «1». Έτσι το πολυώνυμο λειτουργίας του θα είναι $1 \oplus X^2 \oplus X^3$ που δεν είναι primitive.
2. Στο δεύτερο παράδειγμα θα υποθέσουμε ότι εισάγουμε σειριακά την τιμή 6. Αυτό θα σημαίνει ότι INPUT 0 = λογικό «0» , INPUT 1 = λογικό «1» και INPUT 2 = λογικό «1». Συμπεραίνουμε ότι δεν υπάρχει ανατροφοδότηση από το INPUT 0. Άρα έχουμε ένα $2^{ου}$ βαθμού LFSR με πολυώνυμο λειτουργίας $1 \oplus X \oplus X^2$.

Δηλαδή ενώ το υλικό μέρος του LFSR είναι 3bit εμείς μπορούμε να χρησιμοποιήσουμε ένα μόνο μέρος αυτού. Αν λοιπόν το υλικό μέρος του LFSR είναι 8bit εμείς με κατάλληλες τιμές στις εισόδους INPUT μπορούμε να πάρουμε LFSR από $2^{ου}$ βαθμού έως $8^{ου}$ βαθμού με όλα τα δυνατά πολυώνυμα.

Όμοια ισχύουν και για εσωτερικά LFSR:



Μεταξύ εσωτερικού και εξωτερικού LFSR δεν υπάρχει καμιά διαφορά στο αριθμό των εξαρτημάτων που πρέπει να χρησιμοποιηθούν. Υπάρχει διαφορά μόνο στον τρόπο που αυτά συνδέονται. Επειδή στο εξωτερικό LFSR οι συνδέσεις είναι πιο εύκολα ορατές, γίνεται πολύ πιο εύκολα η εισαγωγή αρχικών τιμών αλλά και επειδή τα εξωτερικά LFSR χρησιμοποιούνται πιο συχνά αποφασίσαμε να υλοποιήσουμε τον Pattern Generator και τον Signature Analyzer με εξωτερικό LFSR.

Για να ολοκληρωθεί το προγραμματιζόμενο LFSR θα πρέπει όπως έχουμε προαναφέρει να εισάγουμε με κάποιον τρόπο την αρχική τιμή. Το θέμα αυτό εξετάζουμε στην επόμενη υποενότητα.

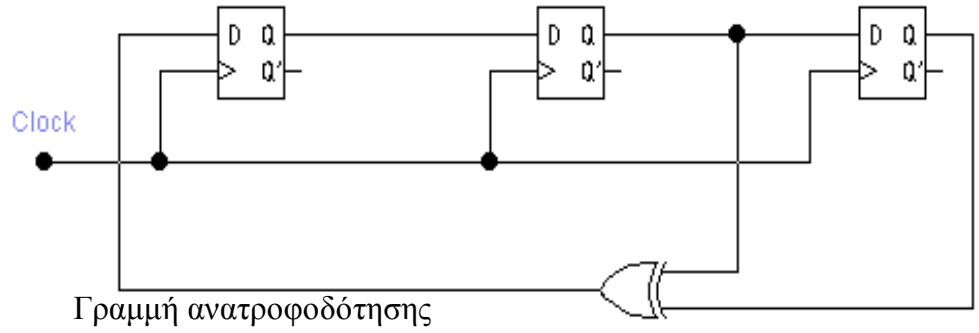
- ***Πως δουλεύουν οι Multiplexers και πως χρησιμοποιούνται στην εφαρμογή μας***

Όσο εύκολο ήταν να σχεδιάσουμε τον τρόπο που γίνεται η επιλογή του πολυωνύμου στον Pattern Generator και στον Signature Analyzer τόσο δυσκολευτήκαμε να εισάγουμε αρχική τιμή στον Pattern Generator. Όπως έχουμε προαναφέρει ο Signature Analyzer δεν χρειάζεται αρχικές τιμές αλλά Reset μετά από κάθε έλεγχο που κάνουμε.

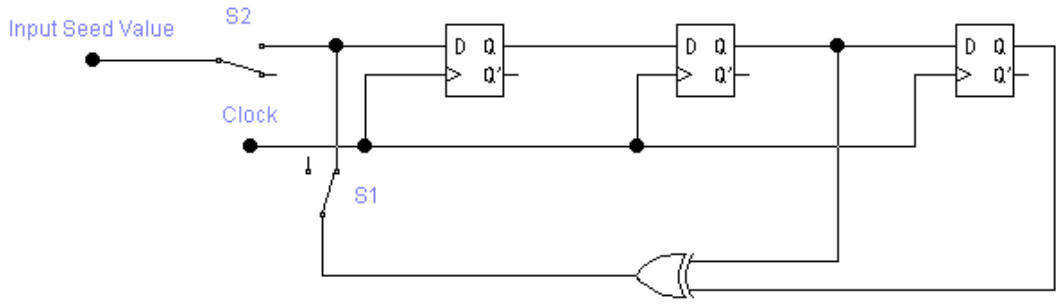
Η δυσκολία είναι στο γεγονός ότι η αρχική τιμή πρέπει να περάσει μέσα στον Pattern Generator. Δεν είναι δηλαδή ένα επιπλέον περιφερειακό του κύκλωμα όπως η επιλογή του πολυωνύμου που είδαμε πριν. Με άλλα λόγια πρέπει να διακοπεί η κανονική λειτουργία του για να γίνει ο προγραμματισμός.

Εκμεταλλευόμαστε όμως την ιδιότητα των LFSR που αν αφαιρέσουμε την ή τις πύλες XOR από ένα LFSR έχουμε έναν απλό καταχωρητή ολίσθησης. Ο καταχωρητής αυτός που αποτελείται από Flip Flops μπορεί να προγραμματιστεί σειριακά όπως και ο καταχωρητής ολίσθησης που ρυθμίζει

το πολυώνυμο λειτουργίας. Σ' αυτήν την περίπτωση οι παλμοί χρονισμού των καταχωρητών θα είναι κοινοί.



Για να διακοπεί όμως η γραμμή ανατροφοδότησης και να μείνει κενή η είσοδος του πρώτου Flip Flop (είσοδος του καταχωρητή ολίσθησης) χρειαζόμαστε έναν μηχανισμό μεταλλαγής. Έτσι στην μια κατάσταση θα συνδέει την γραμμή ανατροφοδότησης στην είσοδο του πρώτου Flip Flop και στην άλλη κατάσταση θα συνδέει την είσοδο του πρώτου Flip Flop με μια από τις εξόδους του μικροελεγκτή της θύρας USB.



Το ισοδύναμο κύκλωμα με μηχανικούς διακόπτες είναι αυτό που φαίνεται παραπάνω. Εύκολο γίνεται αντιληπτό ότι θα πρέπει να έχουμε ή τον S1 σε κατάσταση ON ή τον S2. Όταν έχουμε S1 ON και S2 OFF τότε το παραπάνω κύκλωμα δουλεύει σαν ένας Pattern Generator εφόσον φτάνουν παλμοί χρονισμού (clock) στα Flip Flops. Αντίθετα όταν έχουμε S1 OFF και S2 ON τότε γίνεται σειριακή φόρτωση του καταχωρητή ολίσθησης με την αρχική τιμή. Αν και οι δυο διακόπτες είναι σε κατάσταση ON δημιουργούνται προβλήματα συμπεριφοράς στο κύκλωμα.

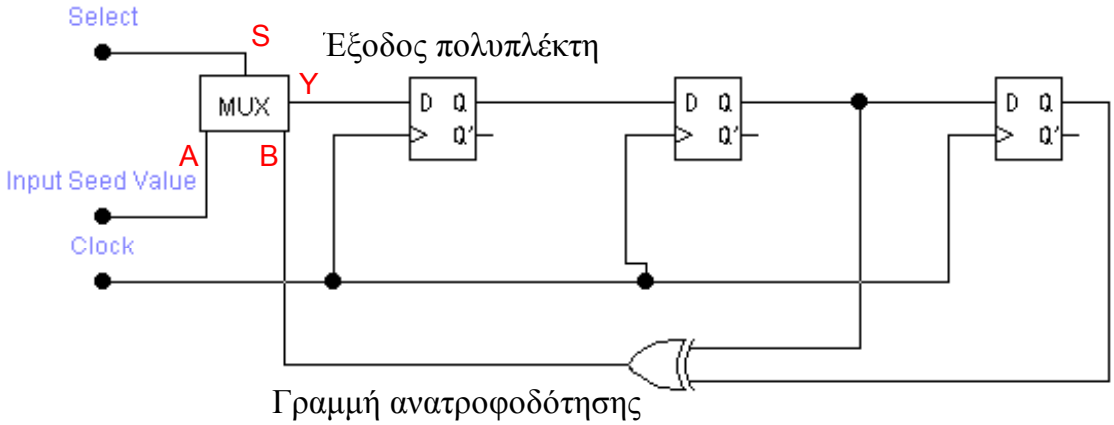
Όπως και στην περίπτωση του προγραμματισμού του πολυωνύμου θέλουμε έναν ηλεκτρονικό τρόπο μεταλλαγής αυτών των δυο διακοπτών. Οι λύσεις που ξέρουμε από τα ηλεκτρονικά δεν είναι πολλές. Μπορούμε να σχεδιάσουμε ένα κύκλωμα με τρανζίστορ αλλά αυτό θα ήταν κάτι πολύ δύσκολο και ιδιαίτερα χρονοβόρο.

Από τα ψηφιακά ηλεκτρονικά το μόνο «εξάρτημα» που γνωρίζουμε να μπορεί να βοηθήσει εδώ είναι ο multiplexer (πολυπλέκτης). Για βοήθεια δίνουμε τον πίνακα αληθείας του :

Inputs			Output Y
Select	Input A	Input B	
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1

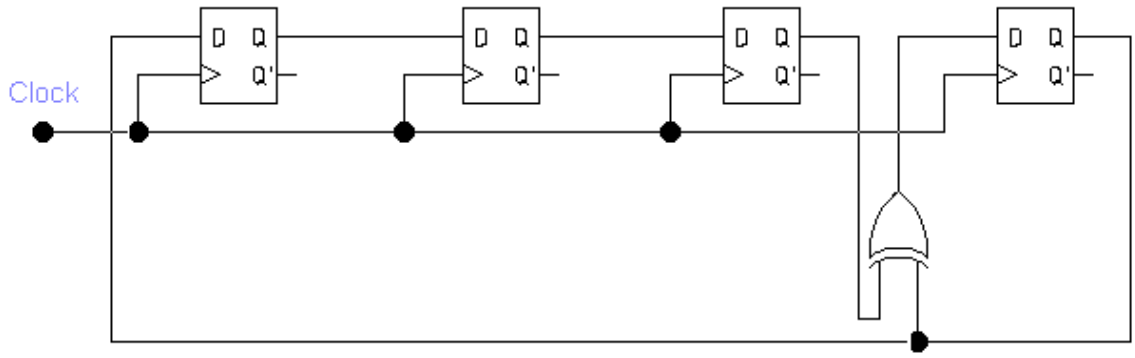
Όπου 1 = λογικό «1», 0 = λογικό «0», X = αδιάφορη κατάσταση

Ο τρόπος που συνδέσαμε τον πολυπλέκτη στο κύκλωμα φαίνεται παρακάτω.



Αν υποθέσουμε ότι στην γραμμή επιλογής (select) έχουμε λογικό «0» τότε η έξοδος του πολυπλέκτη ισούται με την λογική κατάσταση στην γραμμή ανατροφοδότησης. Αντίθετα αν η γραμμή επιλογής του πολυπλέκτη έχει λογικό «1» τότε περνούν τα δεδομένα (αρχική τιμή) από την δεύτερη είσοδο στο πρώτο Flip Flop του καταχωρητή ολίσθησης. Ουσιαστικά θέτοντας την γραμμή επιλογής σε λογικό «0» ή «1», θέτουμε και το συνολικό κύκλωμα σε κατάσταση προγραμματισμού ή κανονικής λειτουργίας, δηλαδή σαν Pattern Generator.

Θα παρατηρήσουμε ότι αν είχαμε εσωτερικό LFSR τα πράγματα θα ήταν πολύ πιο δύσκολα. Αυτό γιατί το κύκλωμα δεν μετατρέπεται σε καταχωρητή ολίσθησης όταν δεν δίνουμε ανατροφοδότηση.



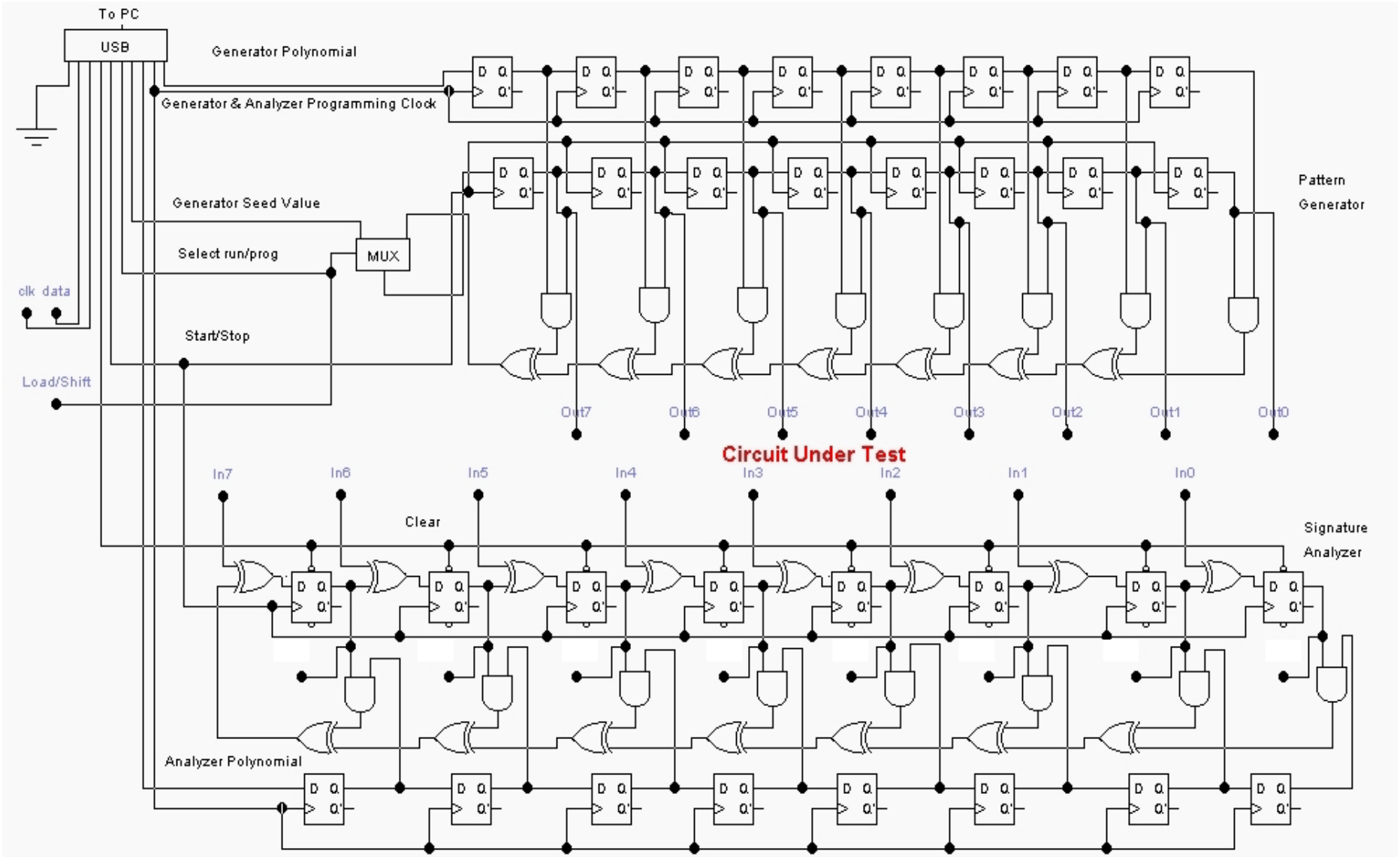
Αφού η πύλη XOR είναι ανάμεσα στα Flip Flops δεν έχουμε μετατροπή σε καταχωρητή ολίσθησης. Για να έχουμε καταχωρητή ολίσθησης θα πρέπει η μια είσοδος της πύλης XOR να έχει λογικό «0» ώστε στην έξοδο της να έχουμε ότι στην άλλη είσοδο. Τότε μόνο θα μπορούσαμε να εφαρμόσουμε όσα αναφέραμε για την εισαγωγή αρχικής τιμής στα εξωτερικά LFSR.

Δεδομένου ότι η κατάσταση στην μια είσοδο της πύλης XOR δεν μπορεί να ελεγχθεί εύκολα (ακόμα περισσότερα ψηφιακά στοιχεία) η εισαγωγή αρχικής τιμής στο LFSR γίνεται ασύμφορη. Αυτός ήταν και ένας από τους βασικότερους λόγους που επιλέξαμε να εφαρμόσουμε εξωτερικό LFSR στον Pattern Generator.

- ***Παρουσίαση και επεξήγηση του τελικού κυκλώματος***

Συνθέτοντας όλα όσα έχουμε πει μέχρι τώρα για τα LFSR μπορούμε να υλοποιήσουμε το τελικό κύκλωμα που εξ αρχής ζητήθηκε.

Στην επόμενη σελίδα φαίνονται αναλυτικά οι συνδέσεις των Flip Flops μεταξύ τους, οι συνδέσεις των πυλών XOR και AND και η συνδεσμολογία του μοναδικού πολυπλέκτη που χρησιμοποιείται. Ο δίαυλος USB φαίνεται στο σχήμα ως ένα μαύρο κουτί (black box). Ουσιαστικά πρόκειται για τις εξόδους / εισόδους του μικροελεγκτή CY7C63001 που είναι υπεύθυνος για την επικοινωνία του υπολογιστή με την συσκευή μας.


To PC
USB
Generator Polynomial
Generator & Analyzer Programming Clock
Generator Seed Value
Select run/prog
MUX
clk data
Start/Stop
Load/Shift
Pattern
Generator
Out7
Out6
Out5
Out4
Out3
Out2
Out1
Out0
Circuit Under Test
In7
In6
In5
In4
In3
In2
In1
In0
Clear
Signature
Analyzer
Analyzer Polynomial

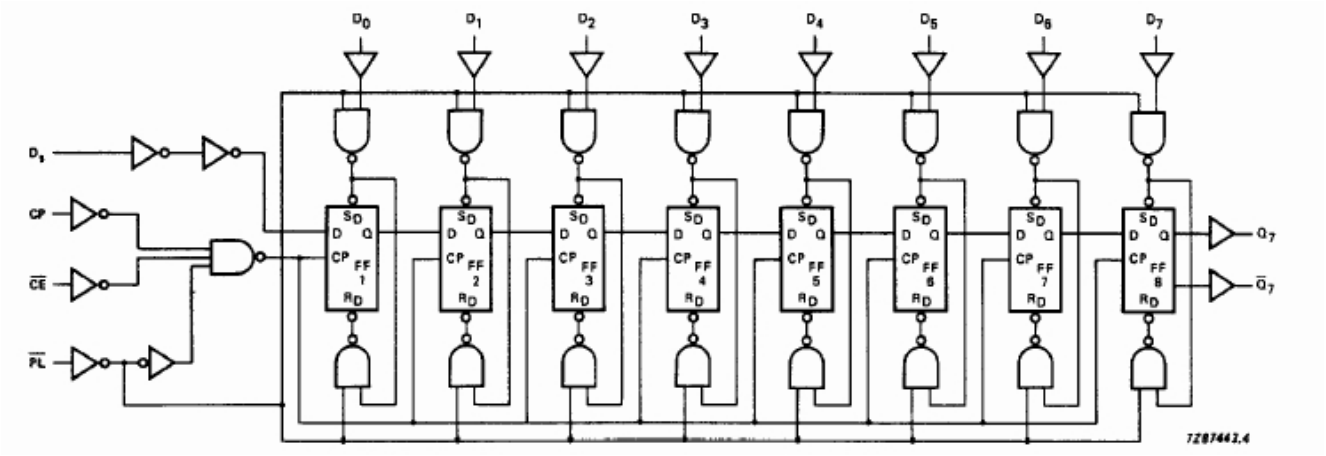
Στο παραπάνω ηλεκτρονικό σχέδιο ξεχωρίζουν ο 8bit Pattern Generator Out0-Out7 και ο 8bit Signature Analyzer In0-In7. Οι έξοδοι του Signature Analyzer είναι αριθμημένες από το 0-7. Το υπό έλεγχο κύκλωμα τοποθετείται ανάμεσα σ' αυτές τις βαθμίδες.

Στην πλακέτα που έχουμε αναπτύξει υπάρχουν συνδεδεμένα LED μέσω αντιστάσεων περιορισμού του ρεύματος 470Ω στις εξόδους του Pattern Generator και στην έξοδο του Signature Analyzer. Θα μπορούσαμε να τοποθετήσουμε LED και στην είσοδο του Signature Analyzer, για να βλέπουμε κάθε στιγμή την αλλοίωση που προκαλεί το υπό έλεγχο κύκλωμα στα δεδομένα που έρχονται από τον Pattern Generator, αλλά αυτό κρίθηκε περιττό.

Στο ηλεκτρονικό σχέδιο που παρουσιάσαμε πριν, ίσως φανεί παράξενο το γεγονός ότι υπάρχουν 3 γραμμές που δεν συνδέονται πουθενά. Αυτές είναι στην άκρη αριστερά πάνω στο σχέδιο :

Γραμμή clk
Γραμμή data
Γραμμή load / shift

Όλες αυτές οι γραμμές μαζί με τις εξόδους του Signature Analyzer οδηγούνται σε ένα ολοκληρωμένο που για λόγους χωρητικότητας δεν παρουσιάσαμε μαζί με τα άλλα. Αυτό είναι ένας μετατροπέας παράλληλης εισόδου σε σειριακή έξοδο (Parallel In - Serial Out ή σε συντομογραφία PISO). Η χρήση του είναι πολύ σημαντική γιατί μέσω αυτού τα αποτελέσματα από τον Signature Analyzer γυρνούν στον υπολογιστή από τον δίαυλο USB. Το ηλεκτρονικό διάγραμμα του παράλληλου σε σειριακού μετατροπέα δίνεται παρακάτω.



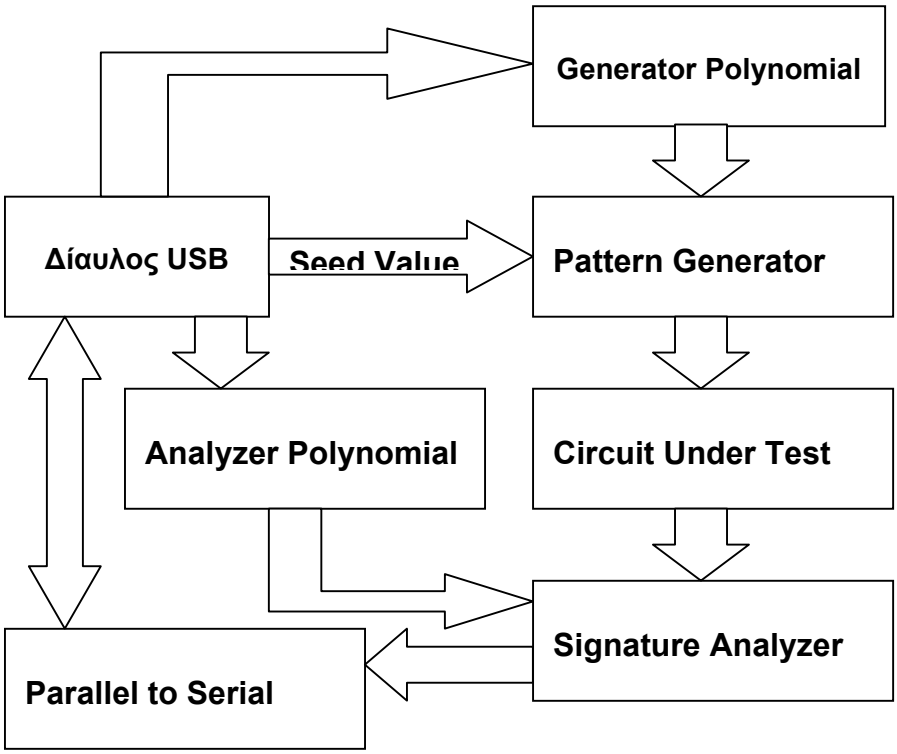
Σαν είσοδο του μετατροπέα από παράλληλο σε σειριακό (D0 – D7) έχουμε τις εξόδους του Signature Analyzer (0 – 7). Η γραμμή data της εφαρμογής μας συνδέεται με την έξοδο Q7 (άκρη δεξιά) του ολοκληρωμένου και η γραμμή clk με την είσοδο CP (άκρη αριστερά). Μέσω των δυο αυτών γραμμών γίνεται η επικοινωνία με τον δίαυλο. Συγκεκριμένα ο υπολογιστής «στέλνει» παλμούς χρονισμού και η συσκευή μας απαντά με σειριακά δεδομένα.

Αν θα γίνεται παράλληλη είσοδος ή ολίσθηση των δεδομένων μέσα στο ολοκληρωμένο εξαρτάται από την λογική κατάσταση στην είσοδο PL (άκρη αριστερά). Συγκεκριμένα αν στην γραμμή Load/Shift πάνω στην συσκευή μας έχουμε λογική κατάσταση «1» τότε είμαστε σε φάση ολίσθησης ενώ αν έχουμε

λογικό «0» θα γινόταν παράλληλη φόρτωση των δεδομένων από τις εξόδους 0 – 7 του Signature Analyzer.

Η είσοδος CE (clock enable) συνδέεται μόνιμα στην γείωση (λογικό «0») του κυκλώματος ώστε να μην επηρεάζει καθόλου, δηλαδή οι παλμοί χρονισμού να περνούν πάντα από αυτόν τον έλεγχο που επιτυγχάνεται με την χρήση πύλης NOR εσωτερικά του ολοκληρωμένου.

Το μπλοκ διάγραμμα όλων όσων προαναφέρθηκαν είναι μια καλή λύση στο να εξακριβώσουμε με μια απλή ματιά τις διασυνδέσεις μεταξύ των βαθμίδων.



Μπλοκ διάγραμμα της συσκευής μας.

Σε τεχνικό επίπεδο υπήρξαν πολλές δυσκολίες στην υλοποίηση της συσκευής. Το μεγαλύτερο πρόβλημα ήταν οι καλωδιώσεις μεταξύ των ολοκληρωμένων. Υπολογίζεται ότι χρησιμοποιήθηκαν περίπου 120 καλώδια για τις συνδέσεις.

Αυτό βέβαια είναι ένα μεγάλο μειονέκτημα γιατί οι παρεμβολές μεταξύ των καλωδίων μειώνουν την αξιοπιστία του τελικού κυκλώματος αλλά και την μέγιστη επιτρεπόμενη συχνότητα λειτουργίας.

Το δεύτερο μεγαλύτερο πρόβλημα ήταν η τροφοδοσία του κυκλώματος των 17 ολοκληρωμένων. Υπήρξαν προβλήματα σταθεροποίησης της τροφοδοσίας όταν χρησιμοποιούσαμε μόνο τους σταθεροποιητές LM7805. Χρειάστηκε η προσθήκη πυκνωτών εξομάλυνσης της τροφοδοσίας για την επίλυση κάποιων δυσλειτουργιών.

Περισσότερα για τα προβλήματα τροφοδοσία και την επίλυσή τους εξετάζουμε στην ενότητα 6 (συμπεράσματα και επέκταση τελικού κυκλώματος).

Στον πίνακα που ακολουθεί βλέπουμε τα 17 ολοκληρωμένα που χρησιμοποιήθηκαν καθώς και την επεξήγηση για το καθένα από αυτά. Παρατηρούμε ότι στην πλειοψηφία τους είναι πύλες και Flip Flops.

Ποσότητα	Κωδικός	Επεξήγηση
1	CY7C63001	Ελεγκτής του διαύλου USB
3	74HC574	Οκταπλό D – Flip Flop 3-state (θετικό ακμοπυροδοτούμενο)
1	74HC273	Οκταπλό D – Flip Flop με Reset (θετικό ακμοπυροδοτούμενο)
1	74HC165	Παράλληλος σε σειριακός μετατροπέας (PISO)
1	74HC157	Τετραπλός πολυπλέκτης 2 σε 1 (MUX 2 –1)
4	74HC08	Τέσσερις δύο εισόδων πύλες ΚΑΙ (AND gates)
6	CD4030	Τέσσερις δύο εισόδων πύλες αποκλειστικού Η (XOR gates)

Ολοκληρωμένα που χρησιμοποιήθηκαν.

Τα 17 ολοκληρωμένα τοποθετήθηκαν σε μια ορθογώνια πλακέτα διαστάσεων 16Χ10 cm πάνω σε βάσεις ώστε να είναι εύκολη η αλλαγή τους αλλά και για να τα προστατέψουμε από τις μεγάλες θερμοκρασίες που αναπτύσσονται κατά την διάρκεια της κόλλησης.

Πρέπει να σημειώσουμε ότι τεχνικές πληροφορίες για καθένα από τα ολοκληρωμένα που χρησιμοποιήθηκαν μπορείτε να αντλήσετε από το cd που συνοδεύει την παρούσα πτυχιακή σε μορφή (.pdf) στο φάκελο **Cd:\E-FILES**

Cd:\E-FILES\cy7c63001.pdf (datasheet του μικροελεγκτή του διαύλου USB)

Cd:\E-FILES\74hc574.pdf (datasheet του οκταπλού D – Flip Flop)

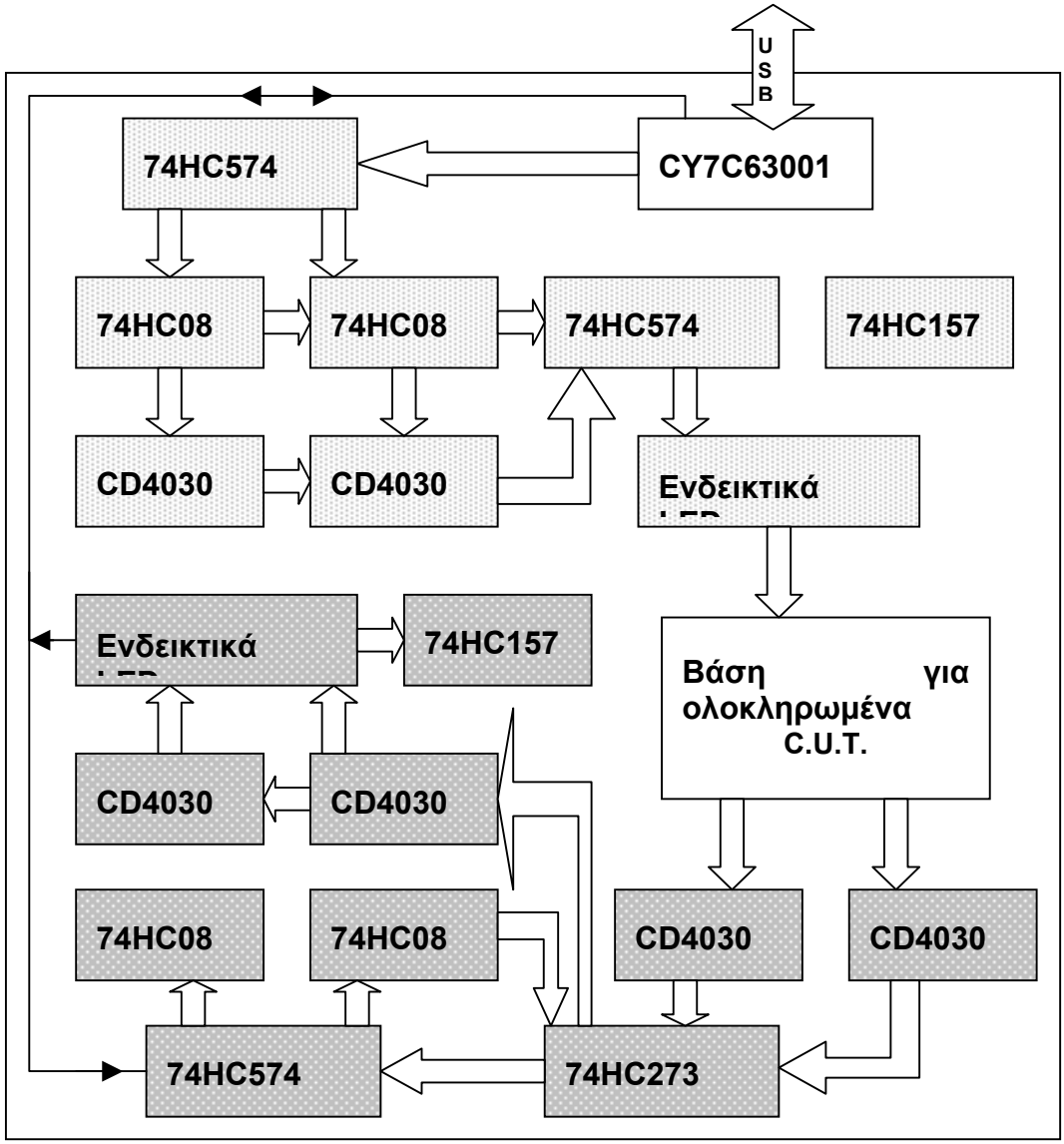
Cd:\E-FILES\74hc273.pdf (datasheet του οκταπλού D – Flip Flop με Reset)

Cd:\E-FILES\74hc165.pdf (datasheet του παράλληλου σε σειριακού μετατροπέα)

Cd:\E-FILES\74hc157.pdf (datasheet του τετραπλού πολυπλέκτη 2 σε 1)

Cd:\E-FILES\74hc08.pdf (datasheet των τεσσάρων δύο εισόδων πυλών ΚΑΙ)

Cd:\E-FILES\cd4030.pdf (datasheet των τεσσάρων δύο εισόδων πυλών αποκλειστικό Η)



Σχεδιάγραμμα της πλακέτας.

Στο σχεδιάγραμμα της πλακέτας ξεχωρίζουμε δυο διαφορετικές σκιάσεις. Η πιο ανοιχτή σκίαση μας δείχνει το κύκλωμα του Pattern Generator ενώ η πιο κλειστή σκίαση είναι ο Signature Analyzer. Το ολοκληρωμένο CY7C63001 αλλά και το και το μπλοκ της βάσης για ολοκληρωμένα δεν έχουν σκίαση γιατί δεν ανήκουν σε καμία από τις δυο παραπάνω κατηγορίες.

Επίσης πάνω στην πλακέτα τοποθετήθηκε ένας μηχανικός διακόπτης ο οποίος δεν φαίνεται στο παραπάνω σχεδιάγραμμα. Ο διακόπτης ενεργοποιεί ή απενεργοποιεί όλα τα κόκκινα LED που είναι πάνω στην πλακέτα (χρησιμεύουν για οπτική ένδειξη της καταστάσεως των εξόδων του Pattern Generator και Signature Analyzer) για λόγους που έχουν με εξοικονόμηση ενέργειας και καλύτερης σταθεροποίησης της παρεχόμενης τροφοδοσίας στα ολοκληρωμένα. Το πράσινο LED ενεργοποιείται όταν υπάρχει τροφοδοσία στη συσκευή και δεν είναι δυνατόν να απενεργοποιηθεί με μηχανικό ή ηλεκτρονικό τρόπο. Ο διακόπτης προτείνεται να είναι σε κατάσταση OFF (ανοικτός) ώστε να μην ανάβουν τα κόκκινα LED κατά την διάρκεια που η συσκευή βρίσκεται σε κατάσταση πλήρους λειτουργίας.

Κλείνοντας την ενότητα 2 να αναφέρουμε ότι η συσκευή παρουσιάζει προβλήματα στον έλεγχο κυκλωμάτων με μικροεπεξεργαστές ή με μνήμες μεγάλης χωρητικότητας και γενικά με κυκλώματα που δεν έχουν γραμμή Reset (επανατοποθέτηση). Αυτό γίνεται γιατί κάθε φορά που θα γίνεται

έλεγχος με την μέθοδο που περιγράψαμε θα παίρνουμε διαφορετικά δεδομένα στον Signature Analyzer ακόμα και αν ελέγχουμε επαναλαμβανόμενα την ίδια συσκευή που εκ των προτέρων γνωρίζουμε ότι δεν έχει βλάβη.

Περισσότερα για τα συμπεράσματα και τις προτάσεις μας για την παρούσα εφαρμογή μπορείτε να βρείτε στην ενότητα 6 (συμπεράσματα και επέκταση του τελικού κυκλώματος).

3. Σχεδίαση σε VHDL

- ***Σχεδίαση σε VHDL***

Η **VHDL (Hardware Description Language ή Γλώσσα περιγραφής υλικού)** είναι μια γλώσσα περιγραφής υλικού για την ανάπτυξη ψηφιακών ηλεκτρονικών συστημάτων. Η λέξη **VHDL** θα λέγαμε, είναι συντόμευση της συντόμευσης **VHSIC (Very High-Speed Integrated Circuit ή Ολοκληρωμένα Κυκλώματα Πολύ Υψηλής Ταχύτητας)[15].**

Η VHDL ως γλώσσα προγραμματισμού μπορεί να χρησιμοποιηθεί για περιγραφή της συμπεριφοράς ψηφιακών ηλεκτρονικών συστημάτων. Για αυτό και χαρακτηρίζεται σαν ένα εργαλείο **CAD (Computer Aided Design).** Αυτό μεταφράζεται στα Ελληνικά σε Σχεδίαση Βοηθούμενη από Ηλεκτρονικό Υπολογιστή.

Σήμερα με την χρήση τέτοιων εργαλείων έχει αναπτυχθεί η τεχνολογία των ημιαγωγών στην κατασκευή ψηφιακών ολοκληρωμένων κυκλωμάτων. Πλέον η προσοχή των μηχανικών έχει δοθεί στην διαχείριση της αυξανόμενης πολυπλοκότητας. Είναι γεγονός ότι σήμερα **ο μηχανικός περιορίζεται περισσότερο από την δυνατότητά του να αντεπεξέλθει στην σχεδίαση του λόγω πολυπλοκότητας παρά από την ικανότητα της τεχνολογίας να τον υποστηρίξει.**

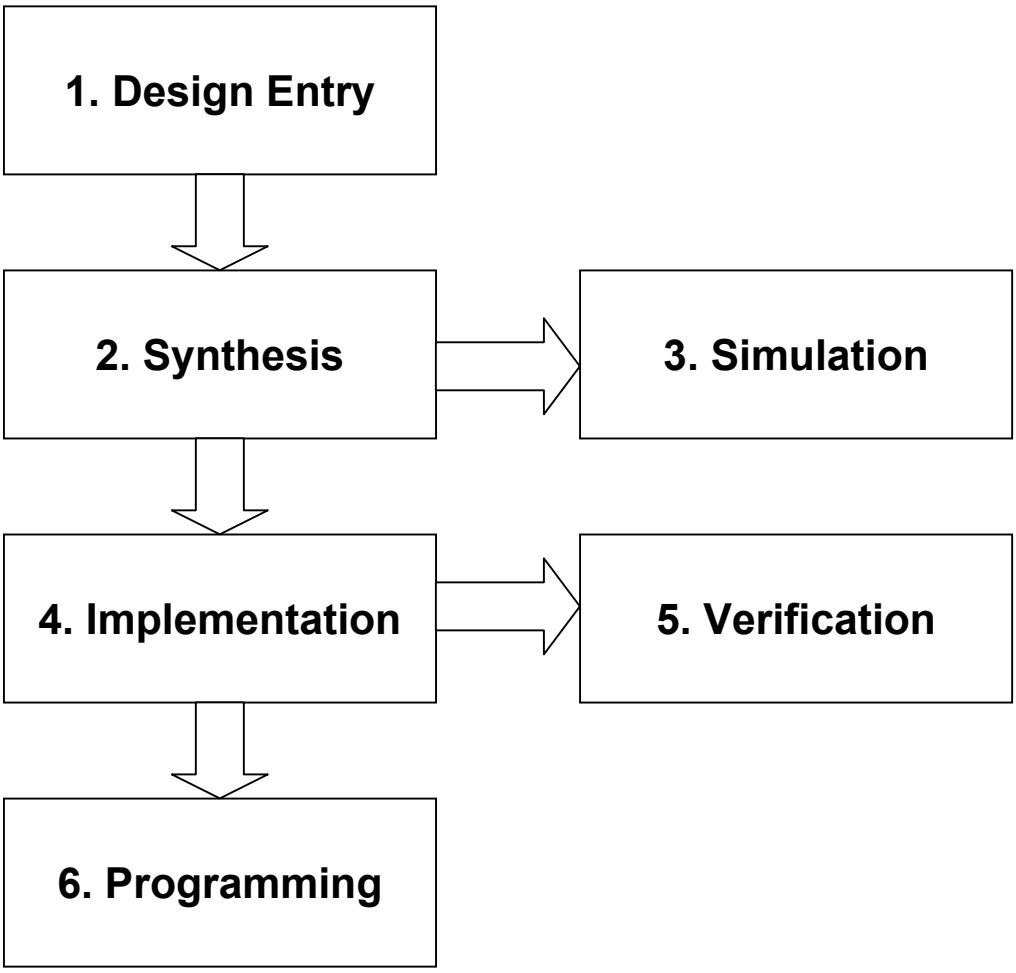
Αυτό το χάσμα γεφυρώνει η γλώσσα προγραμματισμού VHDL επιτρέποντας στον σχεδιαστή να χρησιμοποιήσει υψηλού επιπέδου περιγραφή της σχεδίασής του.

Η VHDL ως γλώσσα περιγραφής υλικού μπορεί να χρησιμοποιηθεί για την περιγραφή ενός ψηφιακού κυκλώματος, από μια απλή πύλη ως ένα ολοκληρωμένο ηλεκτρονικό σύστημα. Ένα ψηφιακό κύκλωμα που σχεδιάζουμε στην VHDL ονομάζεται **entity** (οντότητα). Όταν ένα entity εμπεριέχεται μέσα σε ένα άλλο entity τότε το πρώτο ονομάζεται **component** (εξάρτημα). Μέσα σε ένα entity μπορούν να υπάρχουν όσα components επιθυμούμε.

Αυτή είναι και η μεγάλη δύναμη της γλώσσας VHDL. Ο σχεδιαστής μπορεί να δημιουργήσει μια φορά ένα component όσο και περίπλοκο να είναι και να χρησιμοποιήσει μετά στο κυρίως πρόγραμμα του (entity) όσες φορές επιθυμεί.

Πρέπει επίσης να αναφέρουμε την μεγάλη βοήθεια που προσφέρουν οι έτοιμες βιβλιοθήκες του πακέτου **Xilinx Foundation 3.1c** που χρησιμοποιήσαμε. Μπορούμε για παράδειγμα να δημιουργήσουμε ένα component, όπως για παράδειγμα έναν multiplexer 2 σε 1, με μερικά μόνο click του ποντικιού. Αν και στην εφαρμογή που αναπτύξαμε εμείς χρησιμοποιήσαμε ελάχιστα τις έτοιμες βιβλιοθήκες πρέπει να τονίσουμε ότι είναι εξαιρετικά χρήσιμες.

Για την ανάπτυξη μιας εφαρμογής σε VHDL η σειρά που ακολουθείται είναι η εξής:



Αναλυτικά για το παραπάνω διάγραμμα:

1. **Design Entry.** Σ' αυτήν την φάση γράφουμε το πρόγραμμα που περιγράφει την συσκευή μας με χρήση της γλώσσας VHDL. Μπορούμε να χρησιμοποιήσουμε έναν από τους τρεις τρόπους σχεδίασης ή και συνδυασμό αυτών. Οι τρόποι αυτοί περιγράφονται στην επόμενη υποενότητα.
2. **Synthesis.** Το πρόγραμμα που έχουμε γράψει από την προηγούμενη φάση το «συνθέτουμε», δηλαδή το περνάμε νοητά σε ένα ολοκληρωμένο FPGA για να παρατηρήσουμε αργότερα την συμπεριφορά του. Σ' αυτήν την φάση πρέπει να διαλέξουμε και το ολοκληρωμένο προορισμό. Για παράδειγμα το 4003EPC84 της οικογένειας XC4000E.
3. **Simulation.** Αυτό το βήμα είναι προαιρετικό αν και συστήνεται πάντοτε να το εκτελούμε. Όπως φαίνεται και από την ονομασία του αποτελεί μια πρώτη εξομοίωση για το αν αυτό που «συνθέσαμε» στο προηγούμενο βήμα λειτουργεί όπως περιμέναμε. Αν έχουμε σωστά αποτελέσματα προχωράμε στο επόμενο βήμα.
4. **Implementation.** Implementation σημαίνει υλοποίηση. Ουσιαστικά μετατρέπουμε τον κώδικά μας σε γλώσσα μηχανής, που χρειάζεται για

να προγραμματίσουμε ένα FPGA. Είναι το τελευταίο στάδιο πριν τον προγραμματισμό του FPGA.

5. **Verification.** Είναι το δεύτερο και τελευταίο στάδιο εξομοίωσης της καλής λειτουργίας της εφαρμογής μας. Μπορούμε να το παραλείψουμε αλλά προτείνεται να το εκτελούμε. Η διαφορά του verification με το simulation είναι ότι εδώ στα αποτελέσματα προστίθεται και ο χρόνος που καθυστερεί το ίδιο το FPGA. Στο simulation κάνουμε εξομοίωση για ιδανικές συνθήκες κάτι που βεβαίως δεν ισχύει στην πράξη.

Όπως βλέπουμε από αυτά που έχουν προηγηθεί, καθοριστικός παράγοντας της σχεδίασης μας είναι το ολοκληρωμένο FPGA που θα «φιλοξενήσει» το πρόγραμμα. Σύμφωνα με αυτό θα γίνουν τα βήματα 2 και 4. Τα αποτελέσματα του verification είναι διαφορετικά για ολοκληρωμένα FPGA άλλης σειράς.

- ***Τρόποι σχεδίασης σε VHDL***

Υπάρχουν τρεις διαφορετικοί τρόποι σχεδίασης στην γλώσσα VHDL όπως έχει προαναφερθεί. Οι τρόποι αυτοί διαχωρίζονται ανάλογα με το πώς συνδέουμε εξαρτήματα μεταξύ τους. Έτσι έχουμε:

1. **Δομικός τρόπος σχεδίασης (Structural style of modeling)**

2. **Σχεδίαση με ροή δεδομένων (Dataflow style of modeling**

3. **Συμπεριφερικός τρόπος σχεδίασης (Behavioral style of modeling)**

Ο δομικός τρόπος σχεδίασης περιγράφει το κύκλωμα κυρίως με βάση τα εξαρτήματα (components) που το απαρτίζουν καθώς και τον τρόπο που συνδέονται μεταξύ τους. Για παράδειγμα στον δομικό τρόπο σχεδίασης ανήκει η εντολή :

NAND port map (A, B, Z);

Όπου NAND είναι ένα component που έχουμε ορίσει από πριν. Τα A, B είναι αντίστοιχα οι είσοδοι και Z η έξοδος της πύλης.

Με την χρήση ροής δεδομένων για την υλοποίηση ενός κυκλώματος περιγράφουμε απλά την ροή των δεδομένων μεταξύ στοιχείων συνδυαστικής λογικής όπως απλές πύλες, αθροιστές, κωδικοποιητές κα.

output <= not (A and B);

Το σήμα “output” έχει δηλωθεί ως έξοδος του κυκλώματος. Παίρνει την τιμή του ανεστραμμένου A ΚΑΙ B. Επειδή το A και B είναι στην παρένθεση θα εκτελεστούν πρώτα. Μέχρι εδώ έχουμε μια πύλη AND. Το ανάστροφο της AND είναι μια πύλη NAND.

Στον συμπεριφερικό τρόπο σχεδίασης έχουμε παράλληλες δηλώσεις με κομμάτια σειριακών δηλώσεων οι οποίες περιγράφουν τις εξόδους του κυκλώματος σε συγκεκριμένες χρονικές στιγμές και με συγκεκριμένες εισόδους. Ο συμπεριφερικός τρόπος χρησιμοποιεί άμεσα τις έννοιες του

χρόνου και του ελέγχου. Μπορούμε να πούμε ότι μοιάζει πολύ με γλώσσα προγραμματισμού όπως η BASIC ή C++. Για παράδειγμα:

```
If enable = '1' then
output <= not (A and B);
end if;
```

Εδώ ένα σήμα (signal) που εκ των προτέρων έχουμε ορίσει παίρνει την ανεστραμμένη έξοδο των A ΚΑΙ B (σειριακή δήλωση). Ουσιαστικά έχουμε πάλι μια πύλη NAND όπως στο προηγούμενο παράδειγμα. Αυτό συμβαίνει μόνο αν το σήμα "enable" είναι λογικό «1» (παράλληλη δήλωση).

Αξίζει να αναφερθεί ότι ο συμπεριφερικός τρόπος σχεδίασης μπορεί να περιγράψει και συνδυαστικά αλλά και ακολουθιακά κυκλώματα. Είναι το ιδανικό εργαλείο για την υλοποίηση **μηχανών πεπερασμένων καταστάσεων (Finite State Machines).**

Σαν έναν **τέταρτο τρόπο σχεδίασης** μπορεί να αναφερθεί ο συνδυασμός των τριών παραπάνω τρόπων σχεδίασης. Στην VHDL είναι δυνατόν να έχουμε και τους τρεις τρόπους σχεδίασης στο ίδιο entity ή στο ίδιο component.

Η εφαρμογή που αναλάβαμε να υλοποιήσουμε εμείς με χρήση της γλώσσας VHDL είναι ότι ακριβώς αναφέρεται στην ενότητα **2. Η εφαρμογή του LFSR.** Όπως είδαμε στην ενότητα 2, έχουμε μια συσκευή που αποτελεί ένα σταθμό ελέγχου καλής λειτουργίας ψηφιακών κυκλωμάτων. Αυτή επικοινωνεί με τον υπολογιστή με την βοήθεια του διαύλου USB. Αποτελείται από 17 ολοκληρωμένα και χωρίζεται σε δυο κύριες βαθμίδες. Τον Pattern Generator και τον Signature Analyzer.

Φυσικά δεν είναι δυνατόν να ενσωματώσουμε μέσα στο FPGA που πρόκειται να προγραμματίσουμε τον μικροελεγκτή που είναι υπεύθυνος για την επικοινωνία της συσκευής μας με τον υπολογιστή. Ο μικροελεγκτής αυτός είναι ο CY7C63001 της εταιρίας Cypress και είναι ουσιαστικά το μόνο εξωτερικό εξάρτημα εκτός της τροφοδοσίας.

Παρατηρούμε λοιπόν την μεγάλη ευκολία που μας προσφέρει η γλώσσα VHDL. Στην δική μας εφαρμογή χρησιμοποιούμε μόνο 2 ολοκληρωμένα αντί για 17 και γλιτώνουμε αρκετές καλωδιώσεις μεταξύ τους. Επίσης η επιτρεπόμενη συχνότητα λειτουργίας των Pattern Generator και Signature Analyzer είναι πολύ μεγαλύτερη με την χρήση FPGA αφού δεν υπάρχουν παρεμβολές μεταξύ καλωδίων που έχουμε όταν χρησιμοποιούμε συμβατικά ολοκληρωμένα σε διάτρητη πλακέτα.

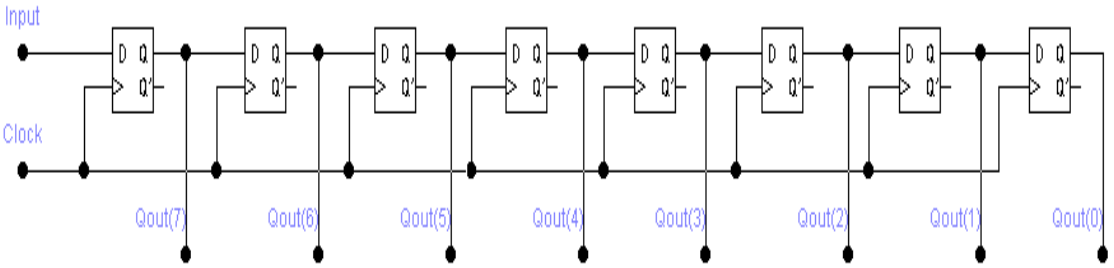
Στα αρνητικά συγκαταλέγουμε μόνο το κόστος απόκτησης του πακέτου και την ανάγκη ύπαρξης προγραμματιστή. Το κόστος των FPGA δεν είναι μικρό, αλλά μπροστά στο κόστος απόκτησης πολλών ολοκληρωμένων που θα κάνουν την ίδια δουλειά με το FPGA, είναι τελικά πιο συμφέρον οικονομικά.

- ***Σχεδίαση του Pattern Generator***

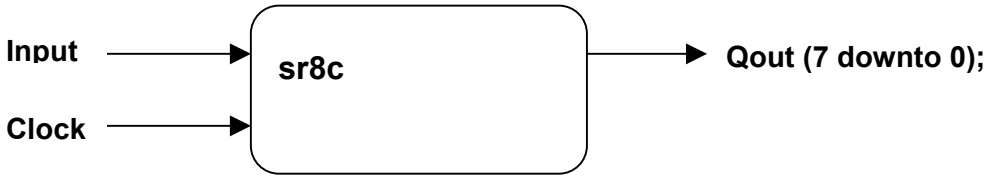
Ο πιο εύκολος τρόπος να υλοποιήσουμε έναν Pattern Generator (και γενικά οποιοδήποτε κύκλωμα) είναι να το τεμαχίσουμε και στην συνέχεια να ενώσουμε τα κομμάτια μεταξύ τους ώστε να σχηματιστεί το τελικό ζητούμενο κύκλωμα.

Στον Pattern Generator που εξετάσαμε στην ενότητα 2 προσπαθούμε να ξεχωρίσουμε κομμάτια που μπορούν να ομαδοποιηθούν.

Έτσι αυτό που καταλήξαμε να κάνουμε ήταν να δημιουργήσουμε ένα entity με το όνομα sr8c (από τις λέξεις Shift Register 8bit with clock). Αυτό φαίνεται στο παρακάτω σχήμα.



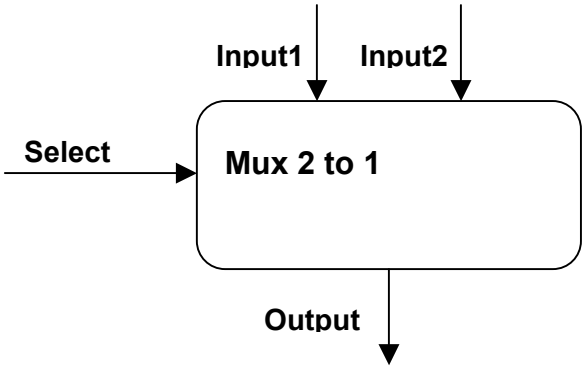
Το entity αυτό το μετατρέπουμε σε component και το βλέπουμε από το κυρίως entity ως ένα μαύρο κουτί (black box) με τις εξής ιδιότητες.



Η είσοδος (Input) είναι ουσιαστικά η είσοδος του πρώτου Flip Flop από αριστερά προς δεξιά. Το clock βέβαια είναι κοινό για όλα τα Flip Flop της σχεδίασης. Μεγάλη διαφορά αποτελεί η έξοδος που την έχουμε ορίσει ως std_logic_vector δηλαδή πίνακας μεταβλητών.

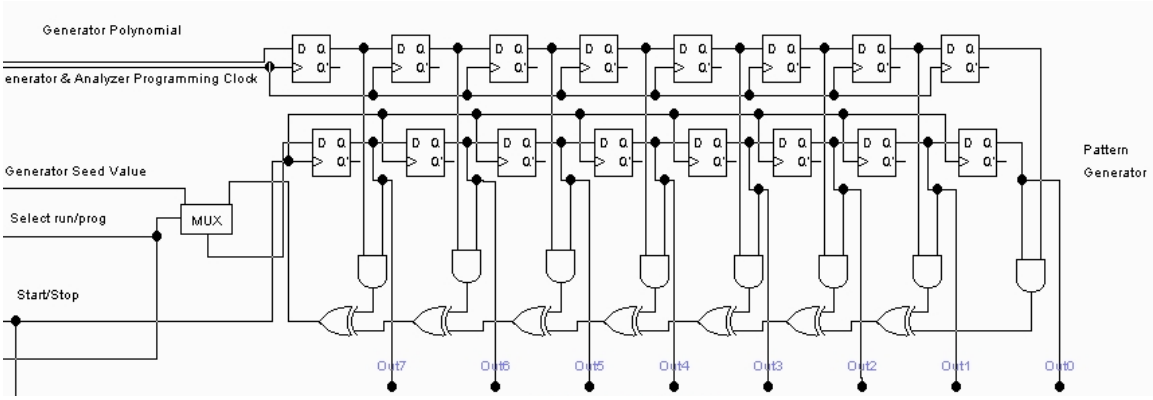
Για παράδειγμα αν θέλουμε να πάρουμε την έξοδο του πρώτου Flip Flop από δεξιά προς αριστερά θα πρέπει να γράψουμε Qout(0). Για την έξοδο του δεύτερου Flip Flop γράφουμε Qout(1) και ούτω καθεξής.

Ένα άλλο entity που έπρεπε να δημιουργήσουμε και στην συνέχεια μετατρέψουμε σε component είναι ο πολυπλέκτης 2 σε 1 (multiplexer 2 to 1). Επειδή υπάρχει στις βιβλιοθήκες ένα component έτοιμο πολυπλέκτης 2 σε 1 δεν χρειάστηκε να το κάνουμε εμείς. Ο πολυπλέκτης ήταν το μόνο component που πήραμε έτοιμο. Οι αντίστοιχες ιδιότητες του είναι:



Σ’ αυτήν την περίπτωση δεν χρειάστηκε να έχουμε vector. Όλες οι είσοδοι και έξοδοι είναι τύπου std_logic.

Τελικά συνδυάζοντας τα δύο παραπάνω components υλοποιούμε τον Pattern Generator. Συγκεκριμένα πρέπει να βάλουμε δύο φορές τον καταχωρητή ολίσθησης που παρουσιάσαμε παραπάνω και μόνο μια φορά τον πολυπλέκτη 2 σε 1.



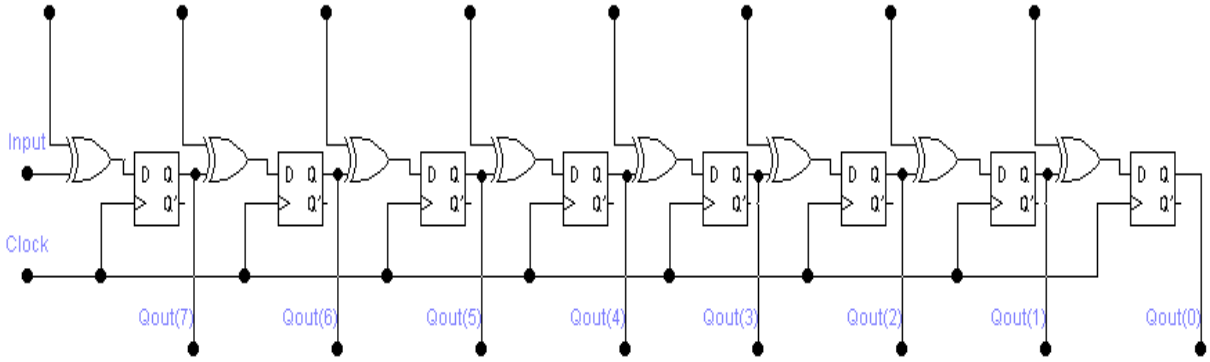
το κύκλωμα του Pattern Generator που υλοποιήσαμε.

- ***Σχεδίαση του Signature Analyzer***

Ομοίως με τον Pattern Generator υλοποιούμε και τον Signature Analyzer. Η διαφορά βέβαια των δύο είναι μεγάλη καθώς στον Signature Analyzer υπάρχουν πύλες XOR ανάμεσα στα Flip Flops. Επίσης το component sr8c δεν έχει γραμμή Reset που είναι απαραίτητη για τον Signature Analyzer. Έτσι δεν μπορούμε να χρησιμοποιήσουμε το component sr8c που ήδη έχουμε κατασκευάσει.

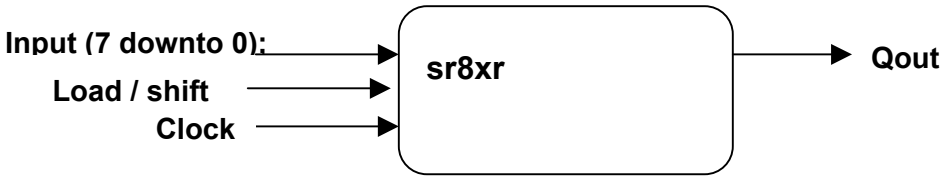
Η διαφορά είναι εμφανής στο παρακάτω σχήμα.

είσοδος δεδομένων από CUT



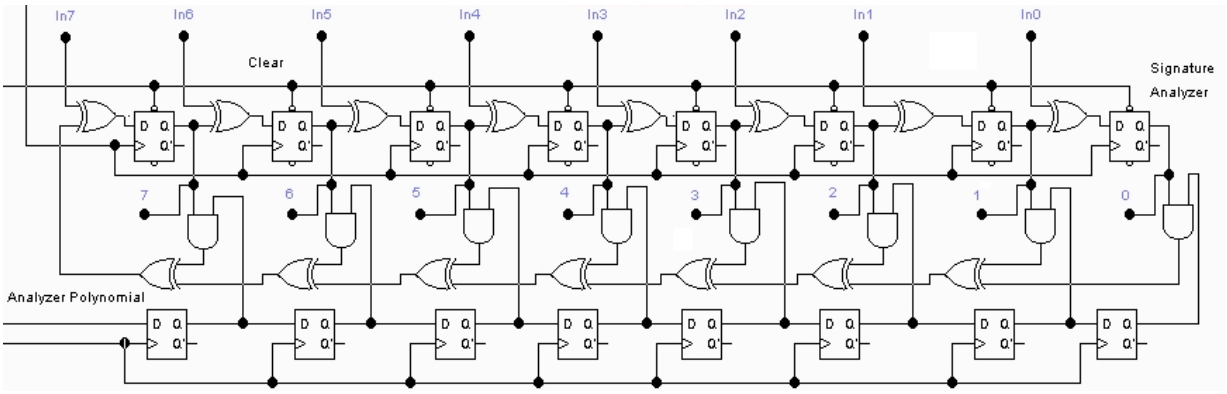
Ο μόνος τρόπος επίλυσης του προβλήματος ήταν να δημιουργήσουμε ένα άλλο component το οποίο θα περιέχει τις πύλες XOR ανάμεσα στα Flip Flops και γραμμή Reset. Έτσι λοιπόν τον νέο αυτό component το ονομάσαμε sr8xr (από τις λέξεις Shift Register 8bit με πύλες XOR και γραμμή Reset).

Για να ολοκληρωθεί ο Signature Analyzer πρέπει να προσθέσουμε με κάποιον τρόπο ένα κύκλωμα που θα υλοποιεί το ολοκληρωμένο 74HC165 (Parallel In – Serial Out). Τέτοιο κύκλωμα δεν ήταν δυνατόν να βρούμε στις έτοιμες βιβλιοθήκες του Xilinx Foundation και έτσι για άλλη μια φορά αναγκαστήκαμε να το κατασκευάσουμε εμείς. Τελικά οι ιδιότητες του είναι αυτές που φαίνεται παρακάτω:



Σε αντίθεση με το sr8c το sr8xr component έχει τύπου vector την είσοδο και τύπου std_logic όλες τις άλλες εισόδους και εξόδους (παράλληλη είσοδος σε σειριακή έξοδο).

Συνδυάζοντας τα δύο παραπάνω components αλλά και ένα sr8c που χρησιμοποιήσαμε στον Pattern Generator υλοποιούμε τον Signature Analyzer. Το sr8c το χρησιμοποιούμε για να αποθηκεύσουμε το πολυώνυμο λειτουργίας για τον Signature Analyzer.

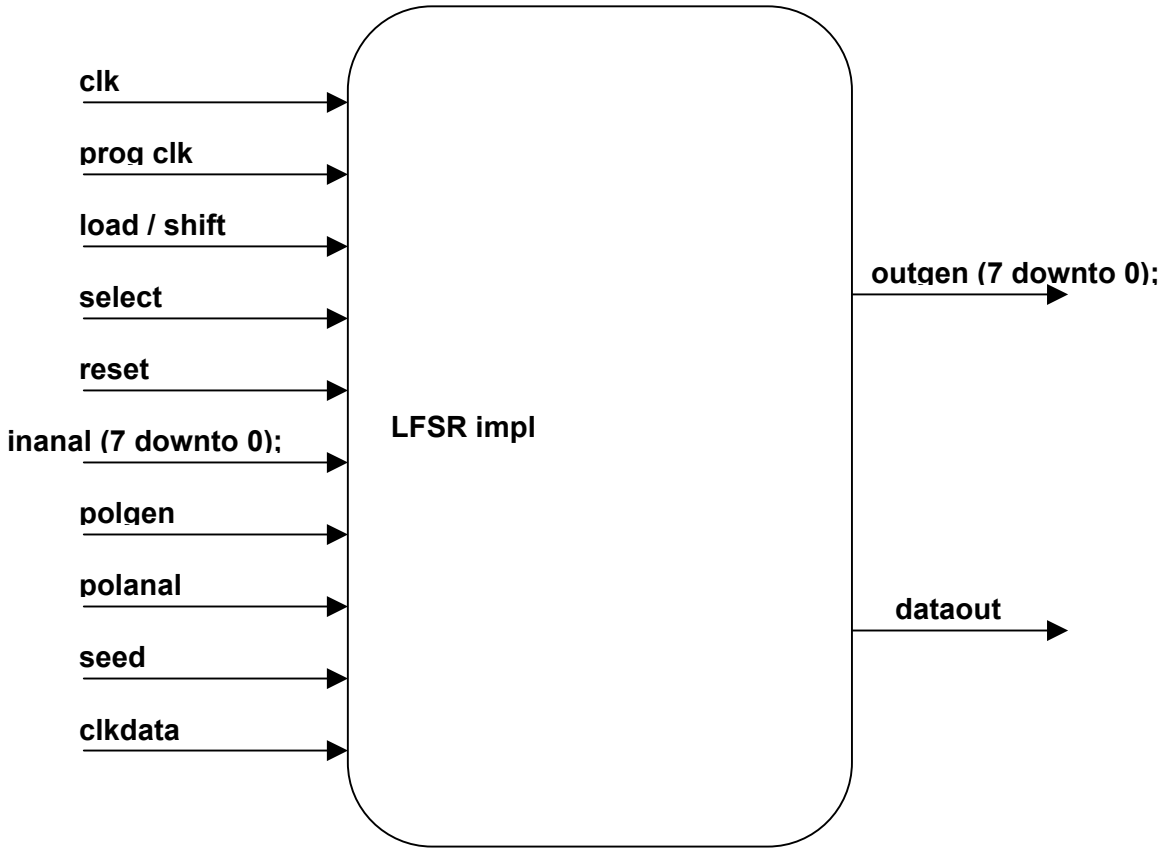


το κύκλωμα του Signature Analyzer που υλοποιήσαμε χωρίς το Parallel In / Serial out κύκλωμα.

- ***Παρουσίαση και επεξήγηση της τελικής σχεδίασης***

Στο κυρίως entity που ονομάσαμε LFSRimpl (από το LFSR implementation) συνθέτουμε ότι έχουμε πει μέχρι τώρα για τον Pattern Generator και τον Signature Analyzer.

Το entity αυτό μπορούμε να το χρησιμοποιήσουμε σαν ένα απλό component σε μια άλλη σχεδίαση με τις εξής ιδιότητες:



Όλες οι γραμμές εισόδου εξόδου που φαίνονται σ' αυτό το σχήμα αποτελούν το κυρίως entity. Μέσα εδώ υπάρχουν όλα τα components που αναφέραμε πριν.

Η έξοδος του Pattern Generator είναι η γραμμή genout (7 downto 0); και είναι τύπου vector. Πάνω εδώ συνδέουμε την συσκευή υπό έλεγχο (circuit under test). Η έξοδος της συσκευής υπό έλεγχο πρέπει να συνδεθεί στην είσοδο inanal (7 downto 0); (είσοδος του Signature Analyzer). Οι άλλες είσοδοι στο LFSRimpl βοηθούν στον προγραμματισμό της συσκευής όπως οι γραμμές polgen (πολυώνυμο του Pattern Generator), polanal (πολυώνυμο του Signature Analyzer), seed value, reset και οι απαραίτητοι παλμοί χρονισμού (clock).

Τέλος πρέπει να αναφέρουμε ότι το LFSRimpl σε πραγματικές συνθήκες θα ήταν ένα προγραμματιζόμενο FPGA. Άρα για να ολοκληρωθεί πρακτικά η συσκευή θα έπρεπε να συνδεθεί στον μικροελεγκτή του USB (cy7c63001) και στην απαραίτητη τροφοδοσία.

Τονίζουμε ότι τα προγράμματα που γράφτηκαν σε Visual Basic θα δουλεύουν κανονικά και σ' αυτήν την περίπτωση. Ουσιαστικά δεν υπάρχει καμία απολύτως διαφορά αν το κύκλωμα γίνει με συμβατικά ολοκληρωμένα ή με χρήση προγραμματιζόμενου FPGA. Στο cd που συνοδεύει την πτυχιακή υπάρχει ο πηγαίος κώδικας του LFSRimpl στον φάκελο :

Cd:\VHDL\LFSRimpl.zip

Επίσης δίνουμε ξεχωριστά τα αρχεία που αποτελούν το LFSRimpl:

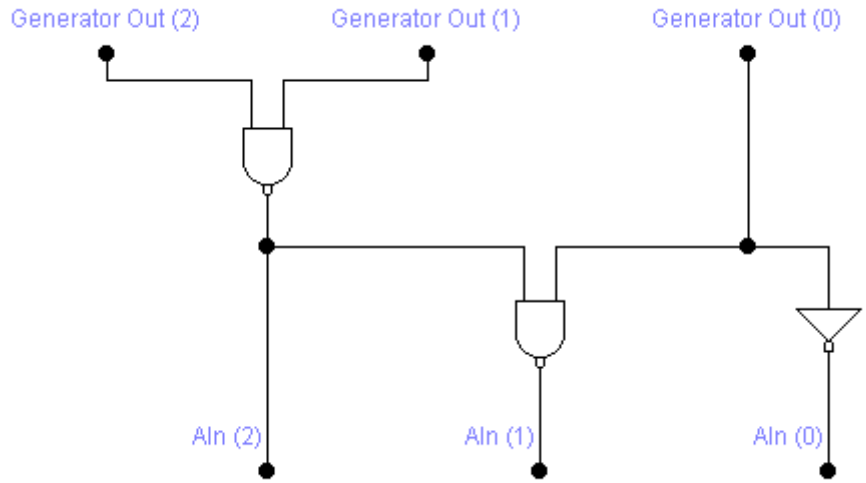
Cd:\VHDL\lfsr.vhd

Cd:\VHDL\sr8c.vhd
Cd:\VHDL\sr8xr.vhd
Cd:\VHDL\partoser.vhd
*τα αρχεία αυτά δεν αρκούν από μόνα τους για να τρέξει η εφαρμογή καθώς χρησιμοποιήσαμε και μια έτοιμη βιβλιοθήκη (m2_1).

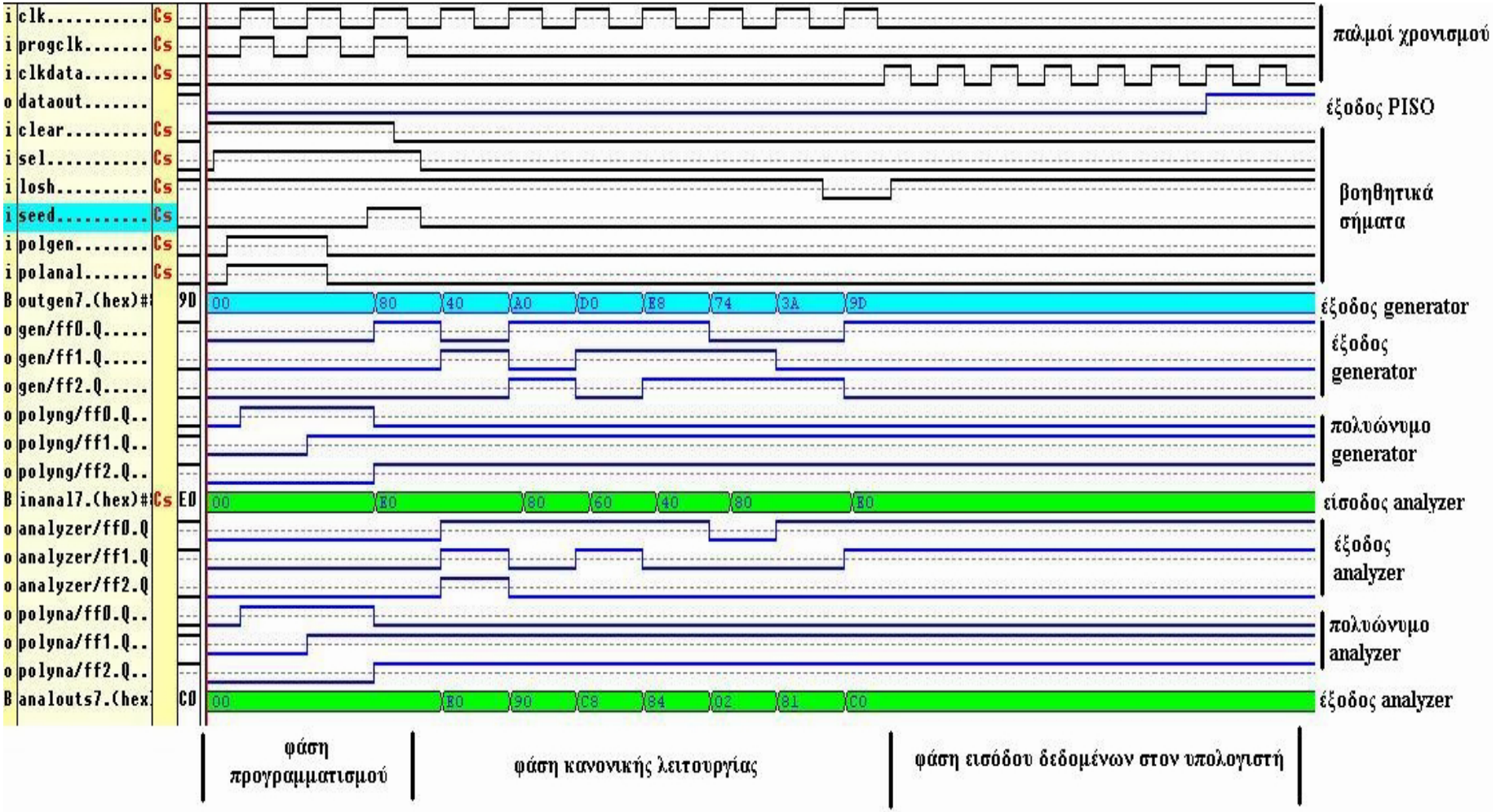
- ***Simulation της τελικής σχεδίασης***

Στην επόμενη σελίδα δίνουμε το τελικό Simulation της εφαρμογής μας. Το simulation παρήχθη από το πακέτο Xilinx Foundation.

Σαν παραδοχή παίρνουμε το γεγονός ότι θέλουμε να υλοποιήσουμε ένα Pattern Generator 3 bit και έναν Signature Analyzer επίσης 3bit. Το κύκλωμα που θα ελέγξουμε είναι το εξής:



Για να βοηθήσουμε τον αναγνώστη να κατανοήσει το χρονοδιάγραμμα της επόμενη σελίδας έχουμε σημειώσει πάνω του τις λειτουργίες που γίνονται για συγκεκριμένες περιόδους.


i clk.....Cs
i progclk.....Cs
i clkdata.....Cs
o dataout.....
i clear.....Cs
i sel.....Cs
i losh.....Cs
i seed.....Cs
i polgen.....Cs
i polanal.....Cs
B outgen7.(hex)#
9D
00
80
40
A0
D0
E8
74
3A
9D
o gen/ff0.Q.....
o gen/ff1.Q.....
o gen/ff2.Q.....
o polyng/ff0.Q..
o polyng/ff1.Q..
o polyng/ff2.Q..
B inanal7.(hex)#
Cs
E0
00
E0
80
60
40
80
E0
o analyzer/ff0.Q
o analyzer/ff1.Q
o analyzer/ff2.Q
o polyna/ff0.Q..
o polyna/ff1.Q..
o polyna/ff2.Q..
B analouts7.(hex
C0
00
E0
90
C8
84
02
81
C0
παλμοί χρονισμού
έξοδος PISO
βοηθητικά σήματα
έξοδος generator
έξοδος generator
πολυώνυμο generator
είσοδος analyzer
έξοδος analyzer
πολυώνυμο analyzer
έξοδος analyzer
φάση προγραμματισμού
φάση κανονικής λειτουργίας
φάση εισόδου δεδομένων στον υπολογιστή

Οι τρεις φάσεις λειτουργίας είναι:

1. **Φάση προγραμματισμού.** Κατά την χρονική διάρκεια των 3 παλμών χρονισμού γεμίζουμε τα 3 πρώτα Flip Flops του καταχωρητή ολίσθησης. Αρκεί να γεμίσουμε 3 Flip Flops για να έχουμε το πλήρες πολυώνυμο λειτουργίας για τον Pattern Generator και τον Signature Analyzer δεδομένου έχουμε 3bit LFSR. Στην πραγματικότητα όμως πρέπει να γεμίσουμε και τα υπόλοιπα Flip Flops με λογικό «0» για να είμαστε σίγουροι ότι δεν θα υπάρχει από προηγούμενη διαδικασία κάποιο λογικό «1» που θα υλοποιεί άλλο πολυώνυμο. Ταυτόχρονα εισάγουμε και την αρχική τιμή στον Pattern Generator.
2. **Φάση κανονικής λειτουργίας**. Εδώ χρειαζόμαστε 8 παλμούς χρονισμού για να ολοκληρώσει το LFSR έναν πλήρη κύκλο λειτουργίας. Θα μπορούσαμε βέβαια να δώσουμε περισσότερα ή λιγότερα βήματα στον Generator. Δεν το κάναμε όμως για να φανεί ότι μετά από οκτώ παλμούς χρονισμού η έξοδος του Pattern Generator θα είναι ίδια με την αρχική τιμή που βάλαμε στην προηγούμενη φάση (001).
3. **Φάση εισόδου στον υπολογιστή.** Στην τελευταία αυτή φάση χρειαζόμαστε οκτώ παλμούς χρονισμού ώστε τα δεδομένα που μάζεψε ο Signature Analyzer, και είναι ήδη φορτωμένα στον Parallel In – Serial Out μετατροπέα, να ολισθήσουν και να τα διαβάσουμε από τον υπολογιστή όπου και θα αποθηκευθούν.

Στον πίνακα που ακολουθεί επαληθεύουμε το χρονοδιάγραμμα της προηγούμενης σελίδας.

Έξοδος Generator						**Έξοδος CUT**				**Έξοδος Analyzer**			
Seed:	**0**	**0**	**1**	**80h**		**1**	**1**	**1**		**0**	**0**	**0**	**00h**
1:	**0**	**1**	**0**	**40h**		**1**	**1**	**1**		**1**	**1**	**1**	**E0h**
2:	**1**	**0**	**1**	**A0h**		**1**	**0**	**0**		**0**	**0**	**1**	**90h**
3:	**0**	**1**	**1**	**D0h**		**0**	**1**	**1**		**0**	**1**	**1**	**C8h**
4:	**1**	**1**	**1**	**E8h**		**0**	**1**	**0**		**0**	**0**	**1**	**84h**
5:	**1**	**1**	**0**	**74h**		**1**	**0**	**0**		**0**	**0**	**0**	**02h**
6:	**1**	**0**	**0**	**3Ah**		**1**	**0**	**0**		**0**	**0**	**1**	**81h**
7:	**0**	**0**	**1**	**9Dh**		**1**	**1**	**1**		**0**	**1**	**1**	**C0h**

Πρέπει να σημειώσουμε ότι οι τιμές σε δεκαεξαδικό στην έξοδο του Pattern Generator δεν αντιστοιχούν στις τιμές σε δυαδικό που βλέπουμε από δίπλα γιατί ο καταχωρητής ολίσθησης δεν κόβεται μετά το 3 Flip Flop και έτσι τα δεδομένα έχουν σειριακή ολίσθηση μέχρι το τελευταίο Flip Flop.

Εκτός από την αλλοίωση που έχουν στην τιμή που παίρνουμε στην τελική έξοδο του Pattern Generator δεν μας επηρεάζουν αλλού γιατί δεν περνούν με κάποιο τρόπο στο κύκλωμα υπό έλεγχο που έχουμε συνδέσει.

Το ίδιο συμβαίνει και στον Signature Analyzer μόνο που εκεί η ολίσθηση εξαρτάται από τις πύλες XOR που έχουμε βάλει ανάμεσα στα Flip Flops. Στον Signature Analyzer μάλιστα μας βολεύει αυτό το γεγονός γιατί **μειώνει την πιθανότητα να έχουμε alias** (ψευδώνυμο). Αυτό γίνεται γιατί στο τέλος διαβάζουμε από τον υπολογιστή τα περιεχόμενα όλου του καταχωρητή ολίσθησης και όχι μόνο των καταχωρητών που συμμετείχαν στον Signature Analyzer.

4. Ο δίαυλος USB

- ***Ιστορικά και συγκριτικά στοιχεία***

Τα αρχικά USB σημαίνουν Universal Serial Bus. Αντιμετωπίζοντάς το εντελώς απλοϊκά και επιφανειακά, το USB είναι μία θύρα που βρίσκουμε στο πίσω μέρος του υπολογιστή μας, δίπλα από τις θύρες για το πληκτρολόγιο και το ποντίκι.

Το USB όμως δεν είναι απλώς και μόνο μία καινούργια θύρα. Είναι πολλά παραπάνω. Είναι το σημείο από το οποίο ουσιαστικά ξεκινάει ένας καινούργιος δίαυλος μία λεωφόρος εξυπηρέτησης ενός μεγάλου αριθμού συσκευών, που μάλλον ποτέ δεν θα χρησιμοποιήσουμε τόσες. Η θύρα USB, στο πίσω μέρος του υπολογιστή μας είναι απλά η πόρτα από την οποία περνούν όλες οι πληροφορίες και τα δεδομένα που στέλνουν και λαμβάνουν οι διάφορες συσκευές USB.

Σε ακόμα πιο τεχνικό επίπεδο, το USB είναι ένας δίαυλος, στον οποίο η μεταφορά δεδομένων γίνεται σειριακά, δηλαδή περίπου με τον ίδιο τρόπο που γίνεται και μέσω της σειριακής θύρας. Τα δεδομένα ταξιδεύουν στη σειρά, μόνο που η ταχύτητα με την οποία ταξιδεύουν είναι πολύ μεγαλύτερη και μπορούν να κατευθυνθούν όχι μόνο προς μία, αλλά έως και προς 127 συσκευές.

Η σκέψη για το USB ξεκίνησε το 1995 από επτά εταιρείες (Compaq Digital Equipment Corp., IBM, Intel, Microsoft, NEC και Northern Telecom) οι οποίες δημιούργησαν το USB Implementers Forum (USBIF) με σκοπό την επιτάχυνση της παραγωγής συσκευών USB. Αυτοί οι κατασκευαστές συσκευών είχαν εντοπίσει τους περιορισμούς της σειριακής και των άλλων συνδέσεων με τον υπολογιστή και έψαχναν να βρουν έναν τρόπο να ξεπεράσουν όλα τα προβλήματα και τις δυσλειτουργίες που υπήρχαν, ώστε να μπορούμε να εκμεταλλευτούμε όλες τις δυνατότητες του δίαυλου USB.

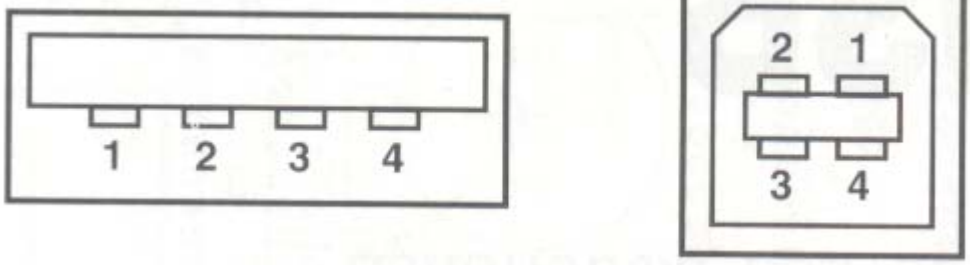
Ο δίαυλος USB με τη βοήθεια του λογισμικού μπορεί να χρησιμοποιηθεί για εφαρμογές σχετικές με μετρήσεις και έλεγχο. Το πρόβλημα ήταν το πώς θα αναπτύξουμε συμβατό περιφερειακό με ικανοποιητικές επιδόσεις, και που θα βρούμε τις σχετικές πληροφορίες.

Η USB είναι ένα σύστημα σειριακής επικοινωνίας (δίαυλος) σχεδιασμένο για σύνδεση πολλών περιφερειακών συσκευών. Η USB είναι πιο σύνθετη από την RS232, αλλά και πολύ πιο γρήγορη φτάνοντας **ταχύτητες των 1,5Mbit/s (για αργές συσκευές) ή 12Mbit/s (για μέγιστη ταχύτητα).** Επιπροσθέτως οι σχεδιαστές της USB έχουν δώσει πολύ προσοχή στην απλότητα της κατασκευής και στην ευκολία χρήσης. Δεν θα ήταν άδικο αν αποδίδαμε στη USB την εφαρμογή για πρώτη φορά της λειτουργίας Plug and Play, η οποία επιτρέπει σε περιφερειακές συσκευές να αναγνωρίζονται αυτόματα από τον υπολογιστή κατά την σύνδεσή τους. Με την USB δεν μπλεκόμαστε με διακοπές στη λειτουργία (κολλήματα), λάθος διευθύνσεις και χαμένους οδηγούς. Οτιδήποτε είναι απλούστερο όμως, και ευκολότερο για τον χρήστη σημαίνει περισσότερη και δυσκολότερη δουλειά για τον σχεδιαστή[17].

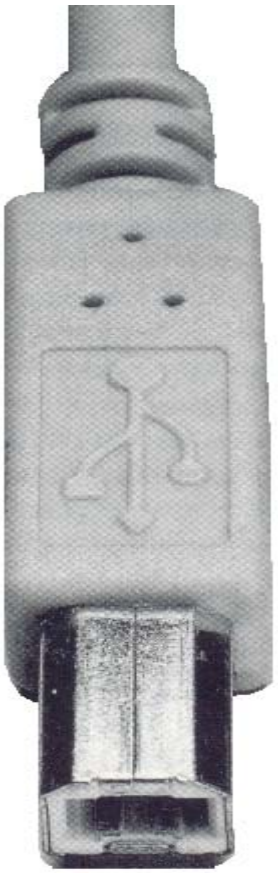
Παρακάτω θα δούμε τι ακριβώς πρέπει να έχει κανείς υπόψη του για να μπορέσει να αναπτύξει με επιτυχία μια εφαρμογή που θα κάνει χρήση του διαύλου USB.

- ***Πως δουλεύει ο δίαυλος USB***

Υπάρχουν δύο διαφορετικοί τύποι βυσμάτων: τύπος Α και τύπος Β. Η σχεδίαση του συστήματος δεν επιτρέπει λάθος στις συνδέσεις τους. Αντίθετα με την RS232, δεν χρησιμοποιούνται εδώ διασταυρούμενα καλώδια. "Όλες οι καλωδιώσεις γίνονται με απλή αντιστοιχία από βύσμα σε βύσμα και οι ακροδέκτες έχουν σταθερή λειτουργία:



1. +5V 2. Data 3. Data+ 4. Γείωση



Βύσμα τύπου Α Βύσμα τύπου Β

Πίσω από κάθε σύγχρονο υπολογιστή θα βρείτε φυσιολογικά δύο εισόδους του Α τύπου. Αυτές χρησιμοποιούνται για άμεση σύνδεση σε δύο συσκευές. Τα σχετικά μικρά και αργά περιφερειακά, όπως το

ποντίκι, χρησιμοποιούν συνήθως ένα λεπτό προκατασκευασμένο καλώδιο με βύσμα Α τύπου. Σε όλες τις άλλες περιπτώσεις, τα περιφερειακά έχουν τη δική τους υποδοχή Β τύπου. Η σύνδεσή τους με τον υπολογιστή γίνεται με ένα καλώδιο Α-Β τύπου.

Στον δίαυλο USB υπάρχουν +5V τάση τροφοδοσίας, από την οποία μπορούμε να τραβήξουμε περίπου μέχρι 100mA. Πολλά ψηφιακά κυκλώματα και μικροεπεξεργαστές λειτουργούν στα 5V. Όταν συνδέσουμε μια τέτοια συσκευή στην είσοδο USB, παίρνουμε και την τάση τροφοδοσίας.

Οι δύο γραμμές δεδομένων μπορούν να χρησιμοποιηθούν μόνο σε συνδυασμό με ειδικά USB εξαρτήματα όπως κάποιοι μικροεπεξεργαστές οι οποίοι παρεμπιπτόντως τροφοδοτούνται από την USB. Εάν υπάρξει μια αίτηση αναγνώρισης, η περιφερειακή συσκευή μπορεί να καταναλώσει έως 500mA.

Η διαθέσιμη τάση τροφοδοσίας από την USB μπορεί να κυμανθεί από 4,2V έως 5.25V. Ολόκληρο το σύστημα συσκευών και καλωδίων εξασφαλίζει ότι η μέγιστη πτώση τάσης δεν θα αφήσει την τροφοδοσία να πέσει κάτω από τα 4,2V. Οι συσκευές που χρειάζονται περισσότερα από 100mA πρέπει να το γνωστοποιήσουν στο σύστημα και τους επιτρέπεται η πρόσβαση μόνον εάν υπάρχει διαθέσιμο το περισσότερο ρεύμα.

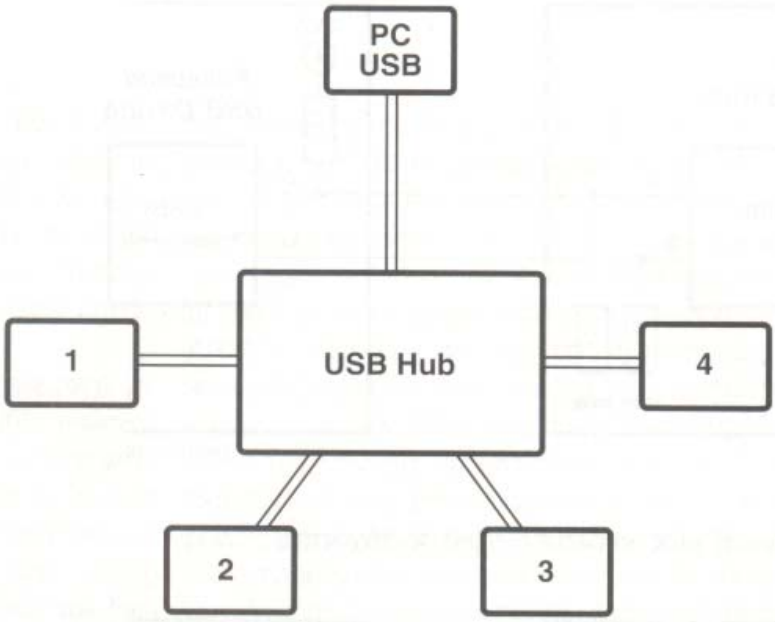
Μεταξύ των περιφερειακών που είναι αυτοτροφοδοτούμενα και αυτών που χρησιμοποιούν την τάση USB, γίνεται μια ξεκάθαρη διάκριση. Αν ζητηθεί περισσότερο από το επιτρεπόμενο ρεύμα από μια συσκευή, η παροχή διακόπτεται. Η δική μας συσκευή είναι αυτοτροφοδοτούμενη. Αυτό το κάναμε γιατί λογαριάσαμε ότι η συνολική τροφοδοσία που χρειαζόμαστε για την σωστή λειτουργία της συσκευής μας μπορεί σε ορισμένες περιπτώσεις να ξεπερνά τα 500mA που είναι η μέγιστη επιτρεπόμενη τάση από τον δίαυλο.

Μόλις συνδέσουμε κάποια συσκευή στον δίαυλο διαπιστώνουμε ότι τα Windows βρίσκουν αυτόματα και εγκαθιστούν τον απαιτούμενο οδηγό. Ακριβέστερα, τα Windows θα εγκαταστήσουν τον οδηγό HID (Human Interface Device) ο οποίος υπάρχει μέσα στον υπολογιστή έως ότου μια συσκευή της αντίστοιχης τάξης που χρησιμοποιεί σύνδεση USB, συνδεθεί σε αυτόν.

Τα HID συμπεριλαμβάνουν ποντίκια, πληκτρολόγια, συσκευές ένδειξης, και joysticks και άλλα. Συμπληρωματικά με τα HID υπάρχουν τάξεις USB για scanners, εκτυπωτές, και άλλες περιφερειακές συσκευές. Δηλαδή, όλες οι κοινές συσκευές χωρίζονται σε τάξεις για τις οποίες είναι διαθέσιμοι οι κατάλληλοι οδηγοί. Αυτός ο διαχωρισμός σε τάξεις δημιουργεί συγκεκριμένα επίπεδα προδιαγραφών. Έτσι και αλλιώς, οι κατασκευαστές των συσκευών πρέπει να εξασφαλίζουν ότι τα προϊόντα τους συμφωνούν με τις προδιαγραφές της τάξης που τους αναλογεί.

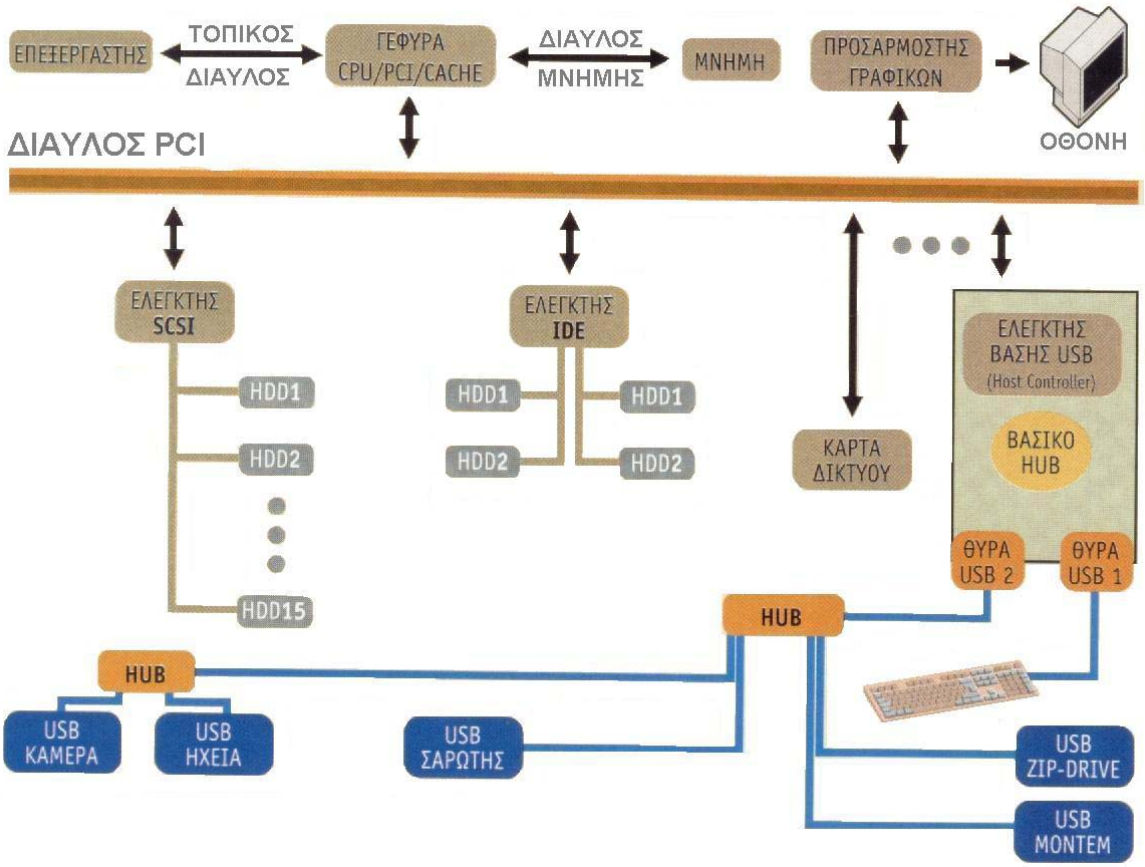
Οι συμβατές με USB συσκευές δεν είναι πλέον ακριβές. Το ίδιο ισχύει και για τις συσκευές USB που φτιάχνουμε εμείς. Μπορούμε να χρησιμοποιήσουμε οποιοδήποτε USB περιφερειακό συνδεδεμένο στον υπολογιστή, γράφοντας τα δικά μας προγράμματα σε γλώσσα DELPHI ή Visual C++ (η Visual Basic δυστυχώς δεν υποστηρίζει προγραμματισμό του διαύλου).

Ο USB δίαυλος είναι ένας κόμβος σε σχήμα αστεριού με μια κεντρική κύρια μονάδα. Για να συνδεθούν πολλές συσκευές USB χρειάζεται ένας κόμβος, ο οποίος να είναι απλά ένας διανομέας του δίαυλου με πολλές εισόδους.



Ο διανομέας αυτός αποτελεί και τον κόμβο του επιμέρους δίαυλου. Ο κόμβος USB μπορεί να έχει μια έξοδο (upstream] και τέσσερις εισόδους (downstream). Ο υπολογιστής. εν τω μεταξύ, έχει ενσωματωμένο στην βασική πλακέτα (motherboard) έναν κόμβο (root hub) για την ένωση των δύο USB εισόδων. Στην είσοδο ενός κόμβου μπορεί να συνδεθεί η έξοδος ενός άλλου κόμβου. Συνολικά επτά κόμβοι μπορούν να συνδεθούν διαδοχικά με αυτόν τον τρόπο δίνοντας δυνατότητα να συνδεθούν μέχρι 127 συσκευές. Αυτό βέβαια είναι ένα θεωρητικό νούμερο γιατί το διαθέσιμο εύρος ζώνης πρέπει να μοιραστεί ανάμεσα σε όλες αυτές τις συσκευές.

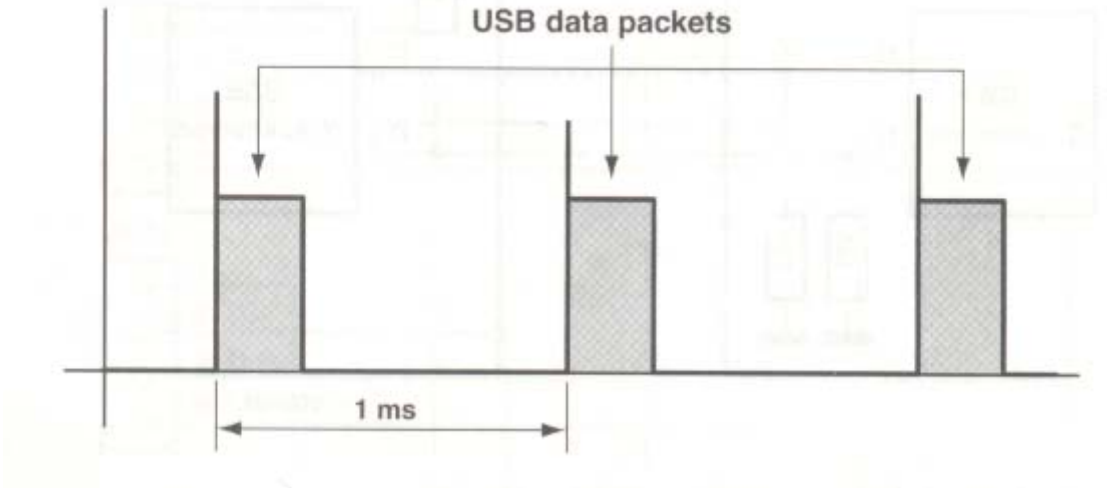
Το πού ακριβώς βρίσκεται συνδεμένο το root hub μέσα στον υπολογιστή είναι κάτι που θα εξετάσουμε στην συνέχεια[7].



Ο ελεγκτής βάσης USB είναι υπεύθυνος για την διεκπεραίωση όλων των εργασιών, που οι συνδεδεμένες στο δίαυλο USB συσκευές, ζητούν να εκτελεστούν. Βρίσκεται σε άμεση επαφή με τον δίαυλο PCI και καταλαμβάνει το μοναδικό IRQ που χρειάζεται για όλο το δίαυλο και τις συσκευές που θα συνδεθούν πάνω του. Οι δύο Θύρες USB που βρίσκονται στο πίσω μέρος του υπολογιστή μας είναι ουσιαστικά το αρχικό hub (root hub). Σε αυτές μπορούν να συνδεθούν είτε συσκευές USB, είτε άλλα hub που με την σειρά τους μπορούν να δεχτούν νέες συσκευές.

Τα σήματα των γραμμών D+ και D- είναι διαφορικά σήματα με στάθμες μεταξύ 0 και 3,3V. Ο μικροεπεξεργαστής του περιφερειακού USB θα λειτουργεί τυπικά στα 3,3V.

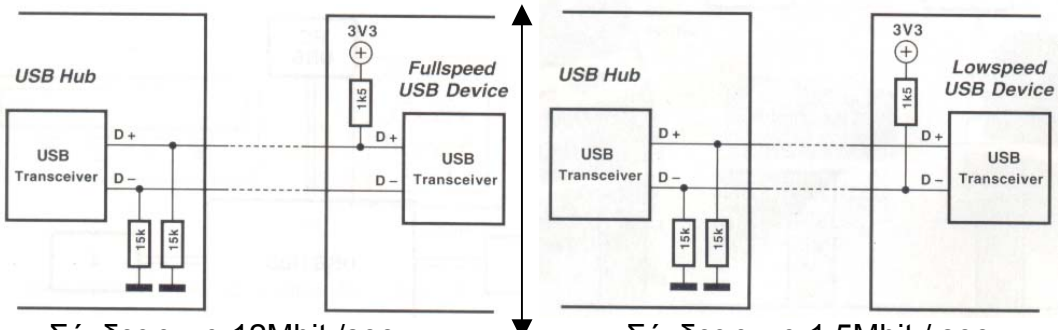
Η σύνδεση USB αποτελείται από έναν δίαυλο με μια κύρια μονάδα. Όλες οι ενέργειες γίνονται από τον υπολογιστή. Τα δεδομένα μεταφέρονται σε πακέτα των οκτώ έως 256 Byte. Ο υπολογιστής έχει τη δυνατότητα να ζητάει δεδομένα από το περιφερειακό. Αντίθετα το περιφερειακό μπορεί μόνο να στείλει δεδομένα. Όλες οι μεταφορές δεδομένων γίνονται σε πλαίσια διάρκειας περίπου 1 msec. Μέσα σε αυτό το πλαίσιο μπορεί να υπάρχουν επιμέρους άλλα πακέτα για διάφορες συσκευές.



Οι συσκευές χαμηλής ταχύτητας λειτουργούν με ροή δεδομένων έως 1,5Mbit/s, δηλαδή το μήκος ενός bit είναι 666,7ns. Στις υψηλής ταχύτητας συνδέσεις η μετάδοση φτάνει τα 12Mbit/s που σημαίνει ότι το μήκος ενός bit είναι 83,33ns. Η ταχύτητα καθορίζεται μόνον από την κύρια μονάδα. Όλα τα περιφερειακά πρέπει να συγχρονίζονται στην ροή αυτή. Επειδή δεν υπάρχουν επιμέρους σήματα συγχρονισμού, το σήμα αυτό πρέπει να δημιουργείται από τη ροή δεδομένων. Για αυτή την εφαρμογή χρησιμοποιείται η αρχή λειτουργίας NRZI (non return to zero). Κατά αυτήν, μόνο τα λογικά «0» αλλάζουν την στάθμη αυτής της τάσης, ενώ στα λογικά «1» παραμένει σταθερή.

Γενικά μια συσκευή USB έχει αρκετές μνήμες FIFO (First In – First Out Memories) για να καταχωρεί δεδομένα. Στον προγραμματισμό της συσκευής προστίθεται ένα σημείο τερματισμού που δείχνει που πρέπει να φτάσουν τα δεδομένα ή από που ξεκίνησαν. Ένας εκτυπωτής USB, για παράδειγμα, έχει πάντα ένα σημείο τερματισμού 1 και ένα σημείο τερματισμού 0. Το δεύτερο χρησιμεύει για την εγκατάσταση. Ο εσωτερικός μικροεπεξεργαστής γράφει τα πραγματικά δεδομένα σε μια FIFO στο σημείο τερματισμού 1 σε κανονικά διαστήματα, απ' όπου αποστέλλονται στον υπολογιστή.

Οι δύο γραμμές δεδομένων είναι σε χαμηλή στάθμη με αντιστάσεις 15ΚΩ ως προς τη γείωση στην αρχή κάθε νέας αρίθμησης. Κάθε περιφερειακό USB έχει μια εσωτερική αντίσταση 1,5ΚΩ από τα +3,3V στην D+ γραμμή αν είναι συσκευή υψηλής ταχύτητας, και στην D- γραμμή αν είναι χαμηλής ταχύτητας.



Σύνδεση για 12Mbit /sec

Σύνδεση για 1,5Mbit / sec

Η πληροφορία αυτή βοηθάει τον κόμβο να αντιληφθεί τον τύπο του περιφερειακού, και να προετοιμάσει την απαιτούμενη ροή δεδομένων. Με την αρίθμηση μιας συσκευής, φορτώνονται αυτόματα οι κατάλληλοι οδηγοί. Επίσης, το σύστημα θα αντιληφθεί πότε μια συσκευή αποσυνδέθηκε από το

δίαυλο. Στην περίπτωση αυτή ο οδηγός διαγράφεται από τη μνήμη. Έτσι γίνεται απλή η σύνδεση μιας συσκευής USB σε διαφορετικούς υπολογιστές. Το μόνο που χρειάζεται είναι να την αποσυνδέσετε από τον έναν υπολογιστή, και να την συνδέσετε στον άλλο. Στον υπολογιστή αυτόν ξεκινάει μια καινούργια διαδικασία αρίθμησης.

Από τη στιγμή που συνδεθεί στον δίαυλο μια οποιαδήποτε συσκευή, η μία από τις δύο γραμμές δεδομένων (D- και D+), που καταλήγουν στον διανομέα, οδηγείται σε υψηλό δυναμικό. Στη συνέχεια ακολουθεί μια “συνομιλία” που τα κύρια σημεία της συνοψίζονται παρακάτω:

- Ο διανομέας γνωστοποιεί την παρουσία της συσκευής στον υπολογιστή.
- Ο υπολογιστής ρωτά τον διανομέα ποια είναι η συγκεκριμένη θύρα στην οποία είναι συνδεδεμένη η συσκευή. Μόλις λάβει την απάντηση, στέλνει στον διανομέα μια εντολή αρχικοποίησης.
- Ο διανομέας παράγει το αντίστοιχο σήμα οδηγώντας ταυτόχρονα και τις δύο γραμμές δεδομένων σε χαμηλή στάθμη για 10 msec. Μετά από αυτή την κίνηση, η συσκευή είναι πλέον έτοιμη να ανταλλάξει δεδομένα με τον υπολογιστή χρησιμοποιώντας σαν διεύθυνση επικοινωνίας την 0. Τη διεύθυνση αυτή την διατηρεί έως ότου ο υπολογιστής ορίσει μια διαφορετική.
- Ο υπολογιστής “διαβάζει” τα πρώτα byte που περιγράφουν τη συσκευή με σκοπό να προσδιορίσει το μήκος του πακέτου πληροφορίας που μεταδίδεται. Σύμφωνα με αυτά δεσμεύει τον κατάλληλο χώρο στη μνήμη του.
- Ο υπολογιστής ορίζει μια καινούργια διεύθυνση για τη συσκευή. Στη συνέχεια “διαβάζει” όλες τις υπόλοιπες πληροφορίες που αφορούν τη συσκευή χρησιμοποιώντας την καινούργια διεύθυνση. Κατόπιν επιλέγει έναν από τους πιθανούς τρόπους επικοινωνίας για την ανταλλαγή δεδομένων με τη συσκευή. Στη φάση αυτή ιδιαίτερη σημασία έχει ο περιορισμός της κατανάλωσης της συσκευής στα επίπεδα που ορίζονται από τα byte που μόλις διαβάστηκαν.

Ο μικροελεγκτής της συσκευής διαθέτει μια εσωτερική μνήμη εισόδου τύπου FIFO στην οποία εγγράφεται κάθε μήνυμα που φθάνει από τον υπολογιστή. Μόλις συμβεί αυτό παράγεται αυτόματα ένα σήμα διακοπής το οποίο ειδοποιεί την CPU του ολοκληρωμένου. Η τελευταία επεξεργάζεται το μήνυμα με σκοπό να διαπιστώσει τον τύπο του. Συνήθως τα πρώτα byte ζητούν από τη συσκευή να στείλει στον υπολογιστή τα στοιχεία της. Η αντίδραση του μικροελεγκτή στη συγκεκριμένη αίτηση είναι η αναζήτησή τους στην εσωτερική ROM και η μετέπειτα εγγραφή τους σε μια δεύτερη μνήμη FIFO. Από εκεί τα κυκλώματα εκπομπής της USB αναλαμβάνουν να τα μεταδώσουν στον κεντρικό υπολογιστή. Τα πρώτα 18 byte που αποστέλλονται περιγράφουν τη συσκευή. Για τον λόγο αυτό αναφέρονται πολλές φορές σαν bytes περιγραφής (Descriptor bytes). Ο υπολογιστής “διαβάζει” αρχικά τα 8 πρώτα. Τα υπόλοιπα μεταφέρονται σε αυτόν αμέσως μετά τον καθορισμό της διεύθυνσης της συσκευής μέσα στον δίαυλο. Το σύνολο των 18 byte μεταβιβάζεται στον υπολογιστή σε τρεις διαδοχικές φάσεις, αφού το FIFO της συσκευής έχει χωρητικότητα 8 byte.

Κάθε μια εταιρία που διαθέτει στην αγορά συσκευές USB έχει και μια διαφορετική ταυτότητα. Η ταυτότητα αυτή εκχωρείται από τον Οργανισμό USB στην ενδιαφερόμενη εταιρία, κατόπιν αίτησής της. Με παρόμοιο τρόπο

ορίζεται και η ταυτότητα της συσκευής. Με τη βοήθεια των δύο αυτών ταυτοτήτων ο κεντρικός υπολογιστής καταφέρνει να αναγνωρίσει τη συσκευή και να φορτώσει τα κατάλληλα προγράμματα - οδηγούς.

Στον δίαυλο USB υπάρχουν τέσσερις διαφορετικοί τρόποι για τη μετάδοση δεδομένων από και προς μια συσκευή συνδεδεμένη σ' έναν PC.

- ***Μετάδοση σημάτων Ελέγχου (Control Transfer)***

Χρησιμοποιείται όταν πρέπει να γίνει μεταφορά πληροφοριών που αφορούν σήματα ελέγχου. Τα σήματα αυτά έχουν υψηλή προτεραιότητα, μεταδίδονται με υψηλή ταχύτητα και συμπεριλαμβάνουν byte που διευκολύνουν την αυτόματη διόρθωση σφαλμάτων. Σε κάθε αίτηση αποστέλλονται το πολύ 64 byte.

- ***Μετάδοση μέσω σημάτων Διακοπής (Interrupt Transfer)***

Με τον τρόπο αυτό επικοινωνούν με τον υπολογιστή ‘αργές’ συσκευές, όπως τα πληκτρολόγια και τα ποντίκια, που γενικά ανταλλάσσουν πακέτα δεδομένων μικρού μήκους. Το όνομα του συγκεκριμένου τρόπου μετάδοσης υπονοεί τη χρήση σημάτων διακοπής, αλλά αυτό στην πραγματικότητα δεν ισχύει. Δεν θα μπορούσε άλλωστε να συμβαίνει σε συστήματα που υποστηρίζουν την παρουσία ενός μόνο κεντρικού υπολογιστή (single master). Το υπολογιστικό σύστημα αναζητεί δεδομένα κάθε 10 msec περίπου. Ο ρυθμός μετάδοσης ανά πακέτο δεδομένων φθάνει τα 8 byte το πολύ.

- ***Μετάδοση μεγάλου Όγκου δεδομένων (Bulk Transfer)***

Χρησιμοποιείται όταν πρέπει να μεταφερθεί μεγάλος όγκος δεδομένων έχοντας προστασία σφαλμάτων, αλλά χωρίς κάποια χρονική κρισιμότητα. Τυπικά παραδείγματα του συγκεκριμένου τρόπου μεταφοράς αποτελούν η λήψη δεδομένων από έναν σαρωτή ή η αποστολή κάποιων άλλων σε έναν εκτυπωτή. Ο ρυθμός μετάδοσης καθορίζεται από το εκάστοτε εύρος ζώνης του διαύλου και της ενεργοποιημένης συσκευής. Η προτεραιότητα των παραπάνω πακέτων είναι χαμηλή.

- ***Ισόχρονη Μετάδοση (Isochronous Transfer)***

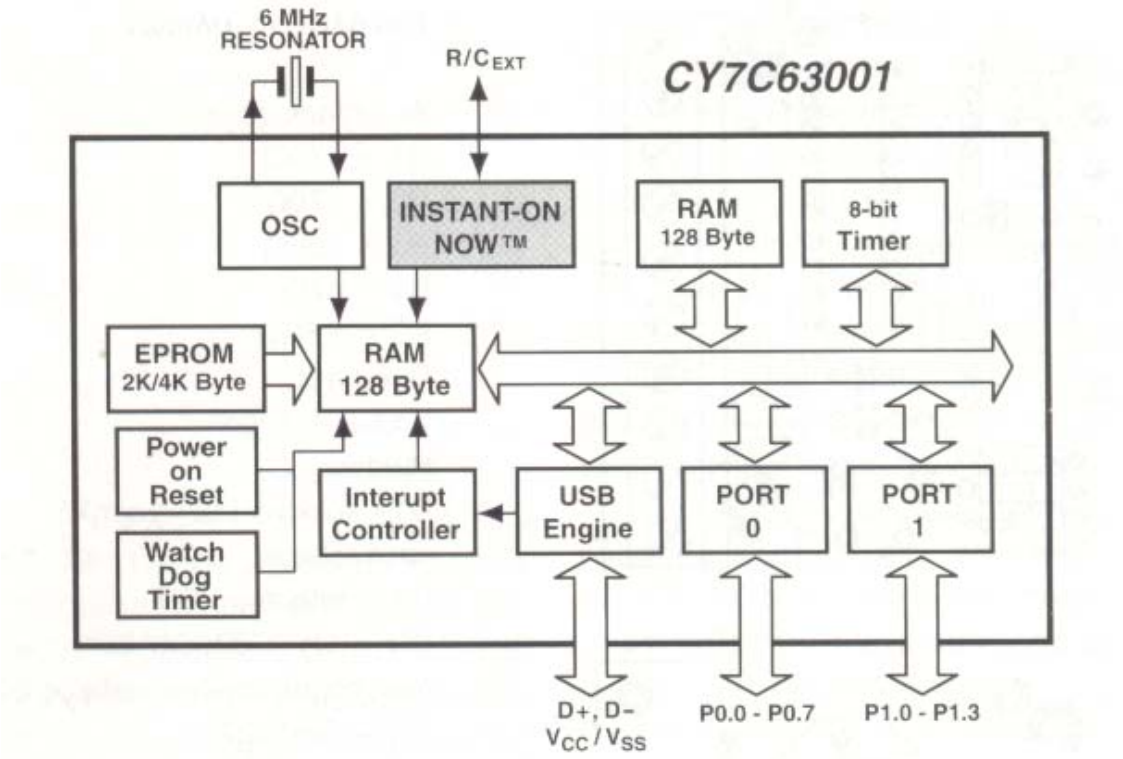
Στη μέθοδο αυτή ισχύουν τα αντίθετα από την προηγούμενη. Υπάρχει και πάλι μεγάλος όγκος δεδομένων, αλλά ο ρυθμός μετάδοσής τους πρέπει να είναι σταθερός. Επίσης, δεν υποστηρίζεται καμία μέθοδος προστασίας σφαλμάτων. Τυπικό παράδειγμα: η μετάδοση δεδομένων που αφορούν ψηφιοποιημένο ήχο σε μια εξωτερική κάρτα ήχου. Η αλλοίωση μερικών byte αδυνατεί να προξενήσει αντιληπτά σφάλματα στον ακροατή.

Οι συσκευές χαμηλής ταχύτητας αξιοποιούν κατά κανόνα τους δύο πρώτους τρόπους σε αντίθεση με τις “γρήγορες” που εργάζονται εξ ίσου καλά και με τους τέσσερις. Σε εφαρμογές που έχουν σχέση με έλεγχο, μετρήσεις ή ρυθμίσεις, ο καλύτερος τρόπος είναι ο πρώτος, αφού υποστηρίζει υψηλή ταχύτητα με ταυτόχρονο έλεγχο σφαλμάτων.

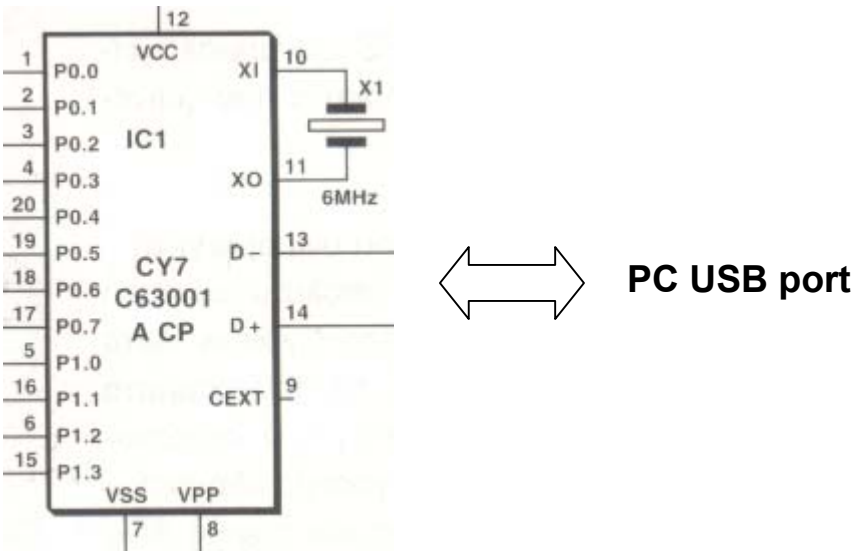
• *Πως χρησιμοποιείται ο δίαυλος στην εφαρμογή μας*

Το ολοκληρωμένο - μικροεπεξεργαστής που κάνει την μετατροπή των σημάτων από / προς τον δίαυλο USB είναι το CY7C63001. Το ολοκληρωμένο αυτό σχεδιάσθηκε για την υποστήριξη ποντικιών και χειριστηρίων παιγνιδιών με διασύνδεση USB, όπως υποστηρίζει η εταιρία που τον παράγει (Cypress Semiconductors). Περιλαμβάνει όλα τα κυκλώματα μιας θύρας USB και επικοινωνεί με τον PC στη χαμηλή ταχύτητα του 1,5 Mbit/sec.

Η βασικότερη εσωτερική βαθμίδα είναι αυτή που αναφέρεται σαν Μηχανή USB (USB Engine). Σ' αυτήν καταλήγουν οι ακίδες D+ και D- του διαύλου μέσω των οποίων μεταφέρονται οι δυαδικοί συρμοί των δεδομένων. Εκτός από αυτήν, στο ίδιο ολοκληρωμένο περιλαμβάνεται ένας επεξεργαστής RISC, μια μνήμη ROM μιας εγγραφής (OTP ROM), μια RAM και ένας χρονιστής. Υπάρχουν ακόμα και κυκλώματα οδήγησης των 12 γραμμών εισόδου / εξόδου που διαθέτει ο μικροελεγκτής[10].



Η σύνδεσή του μικροελεγκτή με την συσκευή μας εξασφαλίζεται μέσω των γραμμών, P0.3 - P0.7 και P1.0 – P1.3 . Οι υπόλοιπες τρεις γραμμές P0.0 - P0.3 δεν κρίθηκε αναγκαίο να χρησιμοποιηθούν. Παρακάτω φαίνεται το CY7C63001 και το διάγραμμά του (chip diagram).



Πειραματικά αυτές οι τρεις γραμμές P0.0 - P0.3 παρουσίασαν πρόβλημα στον χρονισμό, κάτι που πιθανότατα οφείλεται στο κώδικα με τον οποίο έχει προγραμματιστεί ο μικροεπεξεργαστής CY7C63001. Να σημειώσουμε ότι αυτή η πτυχιακή δεν ασχολήθηκε καθόλου με τον κώδικα του CY7C63001 ούτε και με την σύνταξη του αρχείου .INF γιατί αυτό θα μπορούσε να είναι από μόνο του μια πτυχιακή. Ουσιαστικά αυτός είναι και ο μόνος λόγος που θα απέτρεπε κάποιον να ασχοληθεί με τον δίαυλο USB.

Γι' αυτό η εταιρία Cypress, αλλά και αρκετές άλλες εταιρίες του χώρου, κυκλοφόρησαν έτοιμους προγραμματισμένους μικροεπεξεργαστές συνοδευόμενοι από μια δισκέτα που φέρει το αρχείο εγκατάστασης του ολοκληρωμένου (Cypress.INF για την δική μας περίπτωση). Έτσι έγινε σχετικά εύκολη η χρησιμοποίηση του διαύλου από τους απλούς χρήστες.

Το άλλο σημείο που υπήρχε δυσκολία από μεριάς απλών χρηστών ήταν το πως θα γίνει η επικοινωνία μεταξύ υπολογιστή και μικροεπεξεργαστή. Και για αυτό προνόησαν όμως και εκδώσαν φυλλάδια οδηγιών στο Internet για τις δημοφιλέστερες γλώσσες προγραμματισμού όπως Visual Basic, Visual C++, Delphi και άλλες.

Περισσότερες πληροφορίες για το πώς επιτεύχθηκε η επικοινωνία μεταξύ υπολογιστή και μικροεπεξεργαστή θα αναλύσουμε παρακάτω.

Για την πρόσβαση στο πρόγραμμα-οδηγό καλούμε τις έτοιμες από τον κατασκευαστή του μικροεπεξεργαστή συναρτήσεις, CreateFile, CloseHandle και DeviceIoControl, που όλες βρίσκονται μέσα στο αρχείο USB 1.BAS.

Το αρχείο φόρμας USBLFSR.FRM που δημιουργήσαμε με την Visual Basic επιτρέπει την πρόσβαση στις θύρες του μικροεπεξεργαστή. Οι λειτουργίες που υποστηρίζει απαιτούν την εισαγωγή ορισμένων τιμών, τη σημασία των οποίων θα δώσουμε παρακάτω. Η χρήση του δεν αναιρεί καμία από τις δυνατότητες του κώδικα που είναι ήδη αποθηκευμένος στην PROM του μικροελεγκτή.

Όταν καλούμε το πρόγραμμα-οδηγό μέσω της DeviceIoControl πρέπει να προσδιορίσουμε έναν αριθμό συνάρτησης, όπως επίσης και μια περιοχή αποθήκευσης δεδομένων εισόδου (1 In) και μια εξόδου (1 Out). Ο αριθμός συνάρτησης του οδηγού είναι πάντοτε ο 4, ενώ αυτή καθ' αυτή η λειτουργία που θέλουμε να εκτελεσθεί δηλώνεται μέσω του byte που εισάγεται στην περιοχή 1 In. Μετά την εκτέλεσή του το πρόγραμμα-οδηγός επιστρέφει δύο έως τέσσερα bytes ανάλογα με τη συνάρτηση. Το λιγότερο σημαντικό (least significant) απ' αυτά περιλαμβάνει πάντοτε πληροφορίες για το αν η προσπέλαση στον μικροελεγκτή ολοκληρώθηκε με επιτυχία και επιτρέπουν την πρόσβαση στις θύρες και στην εσωτερική μνήμη του μικροελεγκτή.

Έτσι με την συνάρτηση Write Port ή Write RAM γράφουμε έναν αριθμό, ο κώδικας του μικροελεγκτή τον διαβάζει και τον μεταφέρει στην επιθυμητή θύρα. Αντίστοιχα με την συνάρτηση Read Port ή Read RAM διαβάζουμε την κατάσταση της επιλεγμένης θύρας (Port 0 ή Port 1).

Για παράδειγμα η εντολή *WrRam 46,255* θέτει όλα τα pins της πόρτας 0 σε λογικό «1», ενώ η εντολή *WrRam 47,0* θέτει όλα τα pins της πόρτας 1 σε λογικό «0». Υπάρχουν βέβαια και άλλες τιμές εκτός των 0 και 255 με τις οποίες έχουμε διαφορετικούς συνδυασμούς λογικών σημάτων στα pins της εξόδου.

Περισσότερα για το πώς γίνεται ο διαχωρισμός των pins μιας πόρτας ώστε να μπορούν να ελεγχθούν χωριστά θα βρείτε στην ενότητα "5. Προγραμματισμός σε Visual Basic", αυτών των σημειώσεων.

Το απαραίτητο σήμα χρονισμού του μικροελεγκτή παράγεται με τη βοήθεια ενός κεραμικού ταλαντωτή 6 MHz. Οι απαραίτητοι πυκνωτές έχουν ενσωματωθεί στο εσωτερικό του μικροεπεξεργαστή.

Θα πρέπει να σημειώσουμε τέλος ότι ο CY7C63001 δεν είναι ο μόνος διαθέσιμος στο εμπόριο. Υπάρχει πληθώρα μικροεπεξεργαστών για τον δίαυλο USB στο εμπόριο. Στην ενότητα “7. Βιβλιογραφία“ παραθέτουμε όσους εντοπίσαμε. Το κριτήριο επιλογής του CY7C63001 για αυτήν την εργασία ήταν το χαμηλό του κόστος και η πληθώρα βοήθειας στο Internet από απλούς χρήστες αλλά και από την ίδια την εταιρία.

5. Προγραμματισμός σε περιβάλλον Visual Basic

- *Γενικά για την Visual Basic*

Η Visual Basic αποτελεί τον πιο γρήγορο και εύκολο τρόπο να δημιουργήσει κανείς πολυμεσικές εφαρμογές (multimedia applications) για την πλατφόρμα των Microsoft Windows. Χρησιμοποιείται από επαγγελματίες του χώρου αλλά και απλούς χρήστες. Η Visual Basic διαθέτει ένα πλήρες set εντολών που διευκολύνουν την δημιουργία της εφαρμογής μας[8].

Πιο αναλυτικά το κομμάτι "Visual" (οπτικό) αναφέρεται στον τρόπο που χρησιμοποιείται για να δημιουργηθεί το γραφικό περιβάλλον GUI (Graphical user interface). **Αντί λοιπόν να γράφουμε άπειρες γραμμές κώδικα για να περιγράψουμε το πώς θα φαίνεται η εφαρμογή μας, απλά προσθέτουμε ήδη φτιαγμένα αντικείμενα στην οθόνη και καθορίζουμε πως αυτά θα αλληλεπιδρούν μεταξύ τους. Είναι σαν να χρησιμοποιούμε ένα από τα γνωστά προγράμματα ζωγραφικής όπως το paint.**

Το κομμάτι "Basic" αναφέρεται στην γλώσσα προγραμματισμού "Basic" (Beginners All-Purpose Symbolic Instruction Code). **Είναι η γλώσσα που χρησιμοποιείται περισσότερο από κάθε άλλη γλώσσα προγραμματισμού στην ιστορία των ηλεκτρονικών υπολογιστών σύμφωνα με την ίδια την Microsoft. Η Visual Basic γεννήθηκε από την απλή Basic και τώρα συμπεριλαμβάνει εκατοντάδες χιλιάδες νέες εντολές μερικές εκ των οποίων σχετίζονται άμεσα με το γραφικό περιβάλλον GUI των Windows. Η Visual Basic θεωρείται από τις καλύτερες και πιο εύκολες στην εκμάθηση γλώσσες προγραμματισμού που υπάρχουν. Ο νέος αρχίζει μαθαίνοντας μερικές απλές εντολές και συνεχίζει με πιο εξειδικευμένες κατά την διάρκεια της προόδου του. Γενικά με την Visual Basic μπορεί να δημιουργηθεί οποιοδήποτε λογισμικό είναι δυνατόν να δημιουργηθεί.**

Η Visual Basic δεν είναι η μοναδική που διαθέτει οπτικό προγραμματισμό (visual programming). Υπάρχουν γλώσσες προγραμματισμού όπως η Visual C++, η Delphi 5.0 και άλλες λιγότερο γνωστές και πιο εξειδικευμένες σε κάποιες εφαρμογές όπως το ευρέως γνωστό πακέτο "Matlab" η το πακέτο "Xilinx Foundation" στο κομμάτι που αφορά τα "FSM" (Finite State Machine). Επίσης υπάρχουν εφαρμογές που υποστηρίζουν οπτικό προγραμματισμό όπως το "Microsoft Excel" και την "Microsoft Access". Η Visual Basic Scripting Edition είναι από τις πιο ευρέως διαδεδομένες γλώσσες κειμένου. Ουσιαστικά αποτελεί μια μικρότερη έκδοση της Visual Basic με μια υποκατηγορία εντολών.

Όπως και να έχει η Visual Basic είναι μια γλώσσα δημιουργίας ερασιτεχνικών ή και επαγγελματικών εφαρμογών που χρησιμοποιούνται για προσωπική χρήση ή και επαγγελματική. Είναι η καταλληλότερη για την διάθεση των εφαρμογών μέσω του διαδικτύου (Internet) καθώς παρέχει πολλά εργαλεία για αυτόν τον σκοπό.

Μερικές από τις χρήσεις της Visual Basic είναι οι ακόλουθοι:

- Data access. **Είναι λειτουργία που μας επιτρέπει να δημιουργήσουμε βάσεις δεδομένων (Databases), να γράψουμε λογισμικό για διακομιστές (servers) όπως ο SQL ή και άλλοι που κυκλοφορούν στην διεθνή αγορά.**

- ActiveX technologies.**Τεχνολογία που επιτρέπει να χρησιμοποιήσουμε την λειτουργικότητα που παρέχουν άλλα προγράμματα όπως το "Word". Μπορούμε να αυτοματοποιήσουμε διαδικασίες άλλων εφαρμογών μέσα από την δική μας εφαρμογή. Για παράδειγμα θα μπορούσαμε από την δική μας εφαρμογή να αλλάξουμε το φόντο (background) των Windows ή αν για παράδειγμα το "Word" θα**

εφαρμόζει ορθογραφικό έλεγχο στα κείμενα που πληκτρολογούμε. Σε πιο εξειδικευμένες περιπτώσεις μπορούμε να καθορίσουμε τον τρόπο που θα φέρονται τα Windows στην εφαρμογή μας.

- Internet capabilities. **Δυνατότητες που μας επιτρέπει να δημιουργούμε εύκολα εφαρμογές που έχουν άδεια να αλληλεπιδρούν με αρχεία και δεδομένα ή ακόμα και με άλλες εφαρμογές από το Internet ή το Intranet.**

- Autonomous applications. **Εφαρμογές που λειτουργούν αυτόνομα. Έχουν κατάληξη .exe. Είναι αρχεία που μπορούν να τρέξουν από τα Windows σαν ανεξάρτητα προγράμματα.**

Υπάρχουν διάφορες εκδόσεις της Visual Basic. Ανάλογα με το τι προσφέρουν και για ποιόν προορίζονται. Έτσι υπάρχει η «Learning edition» για τους νέους στον χώρο που δεν έχουν ιδιαίτερες απαιτήσεις η «Professional edition» για αυτούς που θέλουν κάτι παραπάνω όπως την δημιουργία ανεξάρτητων (stand-alone) επαγγελματικών εφαρμογών και η «Enterprise edition» που επιτρέπει την δημιουργία εφαρμογών ακόμα και για διακομιστές του διαδικτύου (Internet Servers).

Βεβαίως το κόστος απόκτησης ενός τέτοιου πακέτου είναι ανάλογο με τις δυνατότητες που σου δίνει. Αξίζει να σημειωθεί ότι το κόστος απόκτησης της «Enterprise Edition» που απευθύνεται κυρίως σε εταιρίες παραγωγής λογισμικού τιμάται όσο περίπου ένας καλός ηλεκτρονικός υπολογιστής των ημερών μας.

- *Πλεονεκτήματα και μειονεκτήματα*

Όπως προαναφέραμε η Visual Basic έχει σαν μεγάλο της πλεονέκτημα την εύκολη και γρήγορη σχεδίαση. Επίσης η πολύ καλή εμφάνιση της εφαρμογής και η πλήρης συμβατότητα με τα Windows είναι ένα άλλο μεγάλο πλεονέκτημα.

Το κυριότερο όμως μειονέκτημα είναι ότι δεν «τρέχει» γρήγορα και δεν μπορεί να κατέβει πολύ χαμηλά σε επίπεδο κώδικα. Για παράδειγμα θα αναφέρουμε ότι από την Visual Basic δεν μπορούμε να δημιουργήσουμε μια χρονοκαθυστέρηση της τάξεως του 1ms, ενώ από την Visual C++ η ελάχιστη χρονοκαθυστέρηση θα ήταν περίπου της τάξεως του 1μs. Αυτό έχει σαν αποτέλεσμα να μην μπορεί να «τρέξει» μια συσκευή σαν την δικιά μας με χρόνους κύκλου ρολογιού μεγαλύτερους από 1KHz.

Η χρονοκαθυστέρηση χρειάζεται στην δημιουργία του clock με το οποίο γίνονται τα πάντα πάνω στην συσκευή. Για παράδειγμα ο προγραμματισμός της συσκευής, η περίοδος που αυτή λειτουργεί σαν LFSR, το διάβασμα του Parallel In – Serial Out καταχωρητή κτλ.

Στην προκειμένη περίπτωση όμως η Visual Basic δεν μας περιόρισε γιατί η ίδια η συσκευή δεν "αντέχει" σε συχνότητες μεγαλύτερες από 500Hz όπως μετρήθηκε πειραματικά. Αυτό οφείλεται στο πρωτότυπο της συσκευής, και στο ότι δεν έγινε πάνω σε κανονικό PCB. Για σωστά αποτελέσματα όλη η συσκευή θα έπρεπε να ήταν ένα και μόνο chip, (βλέπε κεφάλαιο 3 - Σχεδίαση σε VHDL). Με αυτόν τον τρόπο θα ήταν δυνατόν να χρησιμοποιήσουμε συχνότητες της τάξεως των MHz όπως και στους σύγχρονους ηλεκτρονικούς υπολογιστές (Pentium III, Athlon κτλ). Σε μια τέτοια περίπτωση η Visual Basic θα μας περιόριζε σημαντικά και θα αναγκαζόμασταν να χρησιμοποιήσουμε άλλες γλώσσες προγραμματισμού όπως Visual C++ ή Delphi.

Παρακάτω θα εξετάσουμε τους χρόνους που χρειάζεται μια οποιαδήποτε συσκευή LFSR για να ολοκληρώσει έναν πλήρη κύκλο. Θα πάρουμε τέσσερις διαφορετικούς βαθμούς LFSR για τρεις διαφορετικές συχνότητες λειτουργίας (τα πολυώνυμα των LFSR είναι primitive, άρα έχουμε 2^n –1 συνδυασμούς).

- 8bit LFSR στο 1KHz. $\frac{255 συνδυασμοί}{1000 συνδιασμοί / \sec} = 0,255 \ ή \ 255ms$

Με τον ίδιο τρόπο βγάζουμε και τις τιμές για τους άλλους βαθμούς των LFSR.

	1 KHz	100KHz	10 MHz
8 bit LFSR	255 msec	2,55 msec	25,5 msec
16 bit LFSR	65,535 sec	655 msec	6,55 msec
32 bit LFSR	1193 hours	715 min	429 sec
64 bit LFSR	584942417 years	5849424 years	58494 years

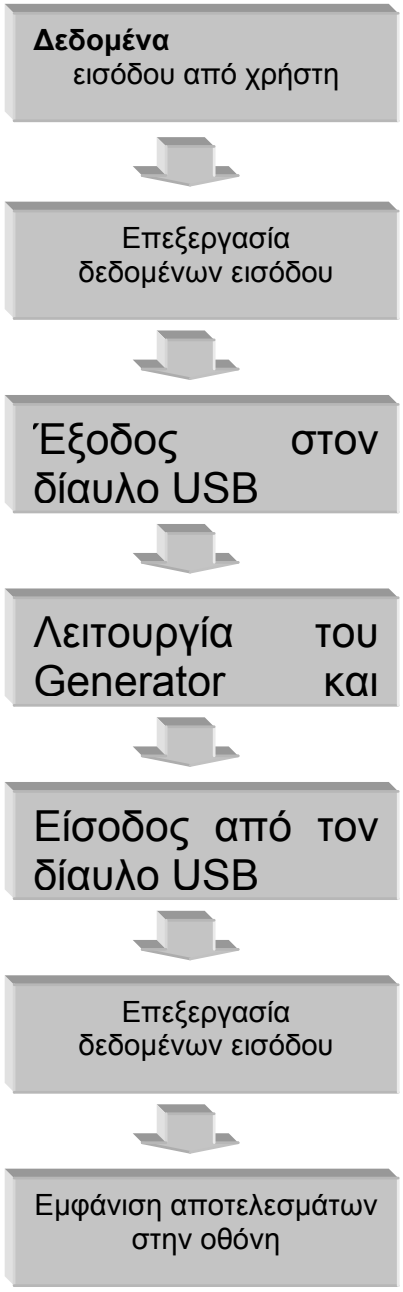
Όπως γίνεται αντιληπτό ο ρυθμός των συνδυασμών ανεβαίνει εκθετικά όσο ανεβαίνει και ο βαθμός του LFSR. Επίσης βλέπουμε πόσο βασικό είναι η συχνότητα λειτουργίας. Στο 1KHz θα χρειαζόμασταν 1193 ώρες για να ολοκληρώσουμε ένα πλήρη κύκλο στα 32bit LFSR! Αυτό θα ήταν αδύνατο να το ανεχθούμε σε εφαρμογές ελέγχου κυκλωμάτων. Να αναφέρουμε επίσης για λόγους εντυπωσιασμού ότι με 1KHz clock και 64bit LFSR θα χρειαζόμασταν για ένα πλήρη κύκλο περίπου 584942417 χρόνια! Με 10MHz clock ο αντίστοιχος χρόνος θα ήταν 701930 χρόνια.

- *Παρουσίαση συγκεκριμένης εφαρμογής*

Το πρόγραμμα οδηγός της κατασκευής μας έχει ως σκοπό να πάρει κάποια δεδομένα σαν είσοδο με την βοήθεια των οποίων θα καθοριστεί η λειτουργία του LFSR. Τα τρία στοιχεία που πρέπει να καθοριστούν είναι τα εξής:

1. αρχική τιμή του γεννήτορα (generator seed value)
2. το πολυώνυμο του γεννήτορα (generator polynomial)
3. το πολυώνυμο του αναλυτή υπογραφής(signature analyser polynomial)

Η συνολική λειτουργία του προγράμματος σε μπλοκ διάγραμμα φαίνεται παρακάτω.



Αναλυτικότερα αν μελετήσουμε το κάθε μπλοκ χωριστά έχουμε:

1. Δεδομένα εισόδου από χρήστη.

Για να εισαχθούν τα στοιχεία στο πρόγραμμα οδηγό έχουν προβλεφθεί διάφοροι τρόποι. Στόχος πάντοτε είναι η ευχρηστία του προγράμματος.

Α) κατά την εκκίνηση του προγράμματος οδηγού υπάρχει μια προεπιλεγμένη τιμή (default) 00000001b ή 01h. Αυτό σημαίνει ότι το ελάχιστα σημαντικό ψηφίο θα είναι «1» και όλα τα άλλα «0».

Β) υπάρχει η επιλογή να εισάγουμε σε δεκαεξαδική (hexadecimal) μορφή μια τιμή σε ένα κελί του προγράμματος οδηγού.

Γ) ο τελευταίος τρόπος εισαγωγής δεδομένων προβλέπει να υπάρχει μια λίστα από τις συνηθέστερες τιμές, και έτσι ο χρήστης να επιλέγει χωρίς να περιπλέκεται με δεκαεξαδικούς αριθμούς. Το πρόγραμμα αναλαμβάνει να κάνει την μετατροπή των επιθυμιών του χρήστη σε δεκαεξαδική μορφή.

2. Επεξεργασία δεδομένων εισόδου.

Σ' αυτήν την φάση γίνεται η μετατροπή από δεκαεξαδική μορφή που έχει δοθεί παραπάνω (μπλοκ 1), σε δυαδική που είναι απαραίτητη για να γίνει ο προγραμματισμός της συσκευής. Αυτό το βήμα θα μπορούσε να εξαλειφθεί αν αναγκάζαμε τον χρήστη να εισάγει τα δεδομένα εισόδου απευθείας σε δυαδική μορφή αλλά θεωρήσαμε ότι κάτι τέτοιο θα μείωνε την ευχρηστία του προγράμματος οδηγού.

3.Έξοδος των δεδομένων στον δίαυλο USB.

Αφού έχουμε έτοιμα δεδομένα σε δυαδική μορφή το μόνο που μένει είναι να το διαβάζουμε bit προς bit και να το εξάγουμε με την βοήθεια παλμών χρονισμού στον δίαυλο USB. Τα δεδομένα έχουν ως προορισμό καταχωρητές ολίσθησης (shift registers) όπου σειριακά με την βοήθεια παλμών ρολογιού, που επίσης παράγονται από το πρόγραμμα οδηγό, αποθηκεύονται προσωρινά. Συνολικά στην κατασκευή μας υπάρχουν τρεις καταχωρητές ολίσθησης που προγραμματίζονται.

4. Λειτουργία του Generator και Analyzer.

Εδώ το πρόγραμμα οδηγός είναι σε κατάσταση κανονικής λειτουργίας. Το μόνο που κάνει είναι να στέλνει παλμούς χρονισμού clock μέσω των οποίων η κατασκευή μας λειτουργεί σαν LFSR. Συνήθως τα βήματα που θα γίνουν είναι ανάλογα του πολυωνύμου του γεννήτορα αν ο χρήστης θέλει να ολοκληρώσει η συσκευή έναν πλήρη κύκλο. Αν για παράδειγμα το πολυώνυμο του γεννήτορα είναι $8^{ου}$ βαθμού τότε τα βήματα που πρέπει να γίνουν είναι 255 (2^8) για έναν πλήρη κύκλο. Εξαίρεση είναι όταν θέλουμε η συσκευή να δουλέψει σαν γεννήτορας ψευδοτυχαίων ακολουθιών (Pattern Generator) ή να την ρυθμίσουμε να μην κάνει όλα τα δυνατά βήματα (non primitive polynomial).

5. Είσοδος των δεδομένων από τον δίαυλο USB.

Το αποτέλεσμα της όλης διαδικασίας τώρα είναι σε ένα ολοκληρωμένο παράλληλης εισόδου – σειριακής εξόδου (parallel In – Serial Out). Με τους αναγκαίους παλμούς χρονισμού μέσω του διαύλου USB διαβάζουμε bit προς bit τα δεδομένα που υπάρχουν μέσα στο ολοκληρωμένο.

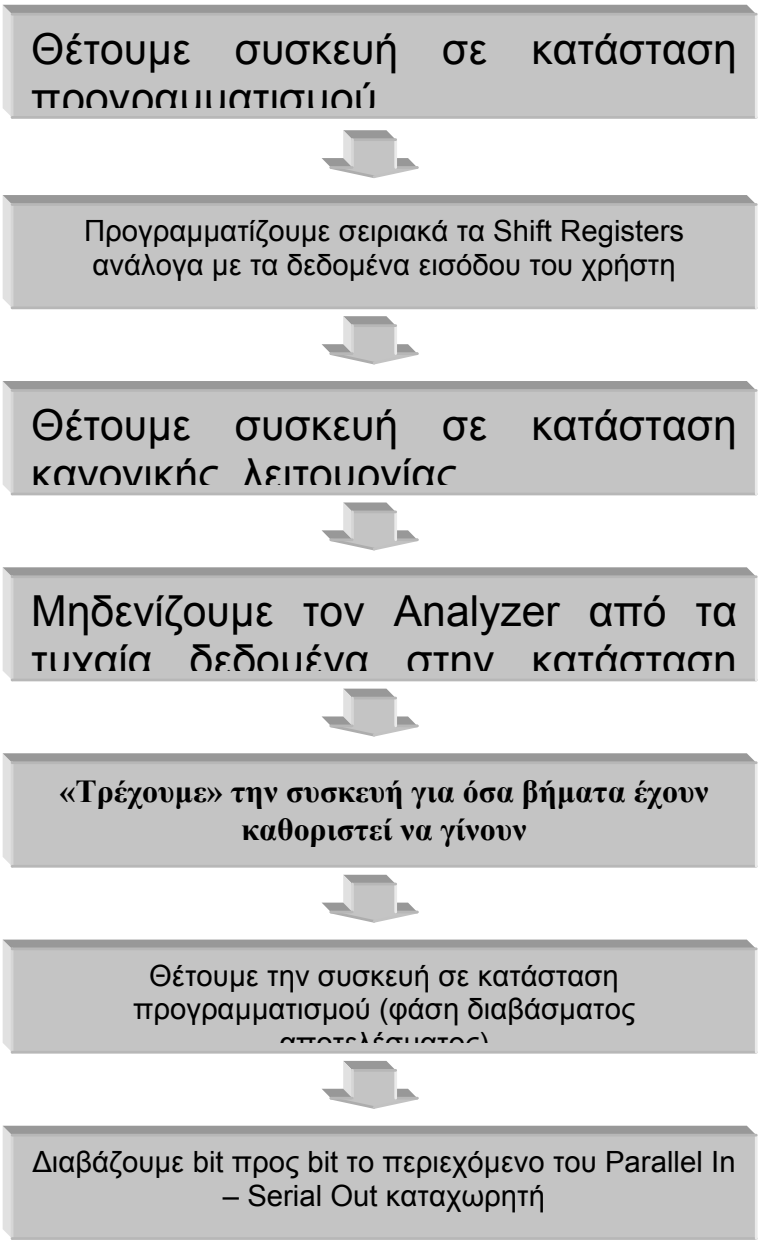
6. Επεξεργασία των δεδομένων εισόδου.

Τα αποθηκευμένα πλέον δυαδικά ψηφία που προήλθαν από το προηγούμενο βήμα πρέπει να μετατραπούν σε δεκαεξαδικά για να είναι ευανάγνωστα στον χρήστη. Ουσιαστικά πρόκειται για την αντίστροφη διαδικασία του μπλοκ 2.

7. Εμφάνιση αποτελεσμάτων στην οθόνη.

Τέλος το μόνο που μένει είναι να εμφανίσουμε τα δεδομένα σε δεκαεξαδική μορφή σε συγκεκριμένα κελιά στο πρόγραμμα οδηγό. Τα αποτελέσματα αυτά συγκρίνονται με τις αναμενόμενες τιμές και βγαίνουν συμπεράσματα για την σωστή ή μη λειτουργία του κυκλώματος υπό επιτήρηση (Circuit Under Test CUT).

- Λειτουργία σε επίπεδο υλικού (Hardware).



Αναλυτικότερα μελετούμε το κάθε μπλοκ χωριστά:

1. **Υπάρχουν δύο καταστάσεις λειτουργίας. Η μια είναι η κατάσταση προγραμματισμού και η άλλη η κατάσταση κανονικής λειτουργίας. Στην ουσία εδώ οδηγούμε έναν πολυπλέκτη (Multiplexer) που επιλέγει αν θα έχουμε την ανατροφοδότηση στο πρώτο Flip Flop του LFSR, ή αν θα έχουμε μια έξοδο από το CY7C63001 με την οποία επιτυγχάνεται η είσοδος της αρχικής τιμής στο LFSR. Εδώ επιλέγουμε να μπούμε σε κατάσταση προγραμματισμού της αρχικής τιμής, του πολυωνύμου του Generator και του πολυωνύμου του Analyzer.**
2. **Αφού έχουμε επιλέξει να γίνει προγραμματισμός απλά δίνουμε στις κατάλληλες εξόδους του CY7C63001 τα κατάλληλα bits για να γίνει σειριακή φόρτωση των Flip Flop. Όλα αυτά βέβαια με την βοήθεια παλμών χρονισμού (clock).**

3. **Θέτουμε την συσκευή σε κατάσταση κανονικής λειτουργίας (normal operation). Αυτό σημαίνει ότι τώρα υπάρχει σύνδεση της γραμμής ανατροφοδότησης (έξοδος της τελευταίας πύλης XOR) με το πρώτο Flip Flop του LFSR.**
4. **Στην συνέχεια μηδενίζουμε τα περιεχόμενα του αναλυτή υπογραφής (signature analyzer). Αυτό γίνεται για να έχουμε κοινό μέτρο σύγκρισης με την επόμενη φορά που θα βάλουμε την συσκευή να "τρέξει". Αν δεν μηδενίσουμε τον αναλυτή είναι σαν να άρχισε από κάποια τυχαία αρχική τιμή.**
5. **Ανάλογα με το πόσες φορές έχει οριστεί να γίνουν πλήρεις κύκλοι του LFSR, παράγουμε από το λογισμικό ισάριθμους παλμούς ρολογιού (clock pulses).**
6. **Θέτουμε την συσκευή σε κατάσταση προγραμματισμού για δεύτερη φορά. Μ' αυτόν τον τρόπο θα γίνει ανάγνωση των περιεχομένων του καταχωρητή που μάζεψε την υπογραφή.**
7. **Με την βοήθεια ενός ξεχωριστού pin από τον CY7C63001 εισάγουμε bit προς bit τα δεδομένα από τον Parallel In – Serial Out καταχωρητή. Παράλληλα έχουμε την δυνατότητα να αρχικοποιήσουμε ξανά την συσκευή έτσι ώστε να μην χάνουμε χρόνο αρχικοποιώντας την αργότερα. Αν για παράδειγμα θέλουμε να δοκιμάσουμε άλλη συσκευή θα ξεκινήσουμε από το βήμα 3.**

Μπορεί το πρόγραμμα να φαίνεται τόσο εύκολο αλλά υπήρξαν πολλά πρακτικά προβλήματα κατά την υλοποίηση του. Καταρχήν η Visual Basic δεν είναι και η πιο εξειδικευμένη γλώσσα προγραμματισμού για συστήματα διαχείρισης δεδομένων εισόδου / εξόδου (data acquisition systems).

Για παράδειγμα θα αναφέρουμε ότι δεν έχει εντολές κατάλληλες για μετατροπή ενός δεκαεξαδικού αριθμού σε δυαδικό και το αντίστροφο. Έτσι για την συγκεκριμένη μετατροπή έπρεπε να μετατρέψουμε έναν δυαδικό σε δεκαδικό (ονομάστηκε hex2dec function) και αργότερα με εντολή της Visual Basic από δεκαδικό σε δεκαεξαδικό. Κάτι ανάλογο έγινε για την μετατροπή από δυαδικό σε δεκαεξαδικό.

Άλλες συναρτήσεις (functions αναφέρονται από την Visual Basic) που δημιουργήθηκαν είναι:

1. ο διαχωρισμός των δύο θυρών του CY7C63001.
2. ο διαχωρισμός των ακροδεκτών (pins) από κάθε μια πόρτα έτσι ώστε να μπορεί να ελέγχεται η κατάσταση τους χωριστά.
3. παραγωγή των παλμών χρονισμού (clock pulses) για την διάρκεια της κανονικής λειτουργίας.
4. μετατροπές από δεκαεξαδικό σε δυαδικό σύστημα αρίθμησης και το αντίστροφο.

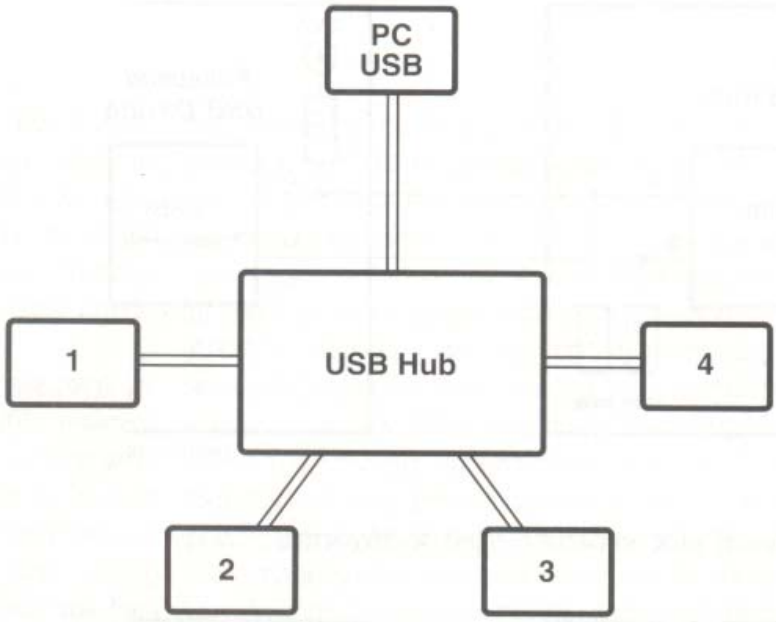
Υπήρξαν βέβαια και άλλες συναρτήσεις (functions) οι οποίες δεν αναφέρονται γιατί δεν παρουσιάζουν κάτι το μοναδικό και δύσκολο να δημιουργηθεί.

6. Συμπεράσματα και επέκταση τελικού κυκλώματος

- ***Επέκταση του τελικού κυκλώματος***

Σ' αυτήν την υποενότητα θα εξετάσουμε την δυνατότητα της εφαρμογής μας να επεκταθεί. Ο όρος επέκταση μπορεί να σημαίνει την αύξηση του βαθμού του πολυωνύμου του LFSR. Για παράδειγμα την υλοποίηση ενός LFSR 16 ή 32bits αντί για 8bits που έχουμε ήδη κατασκευάσει.

Μεγάλο πλεονέκτημα σ' αυτήν την προσπάθεια μας αποτελεί η επιλογή μας να χρησιμοποιήσουμε τον δίαυλο USB. Όπως αναφέρεται και στην ενότητα 4 (Ο δίαυλος USB) είναι ένας κόμβος σε σχήμα αστεριού με μια κεντρική κύρια μονάδα. Για να συνδεθούν πολλές συσκευές USB χρειάζεται ένας κόμβος, ο οποίος να είναι απλά ένας διανομέας του δίαυλου με πολλές εισόδους.



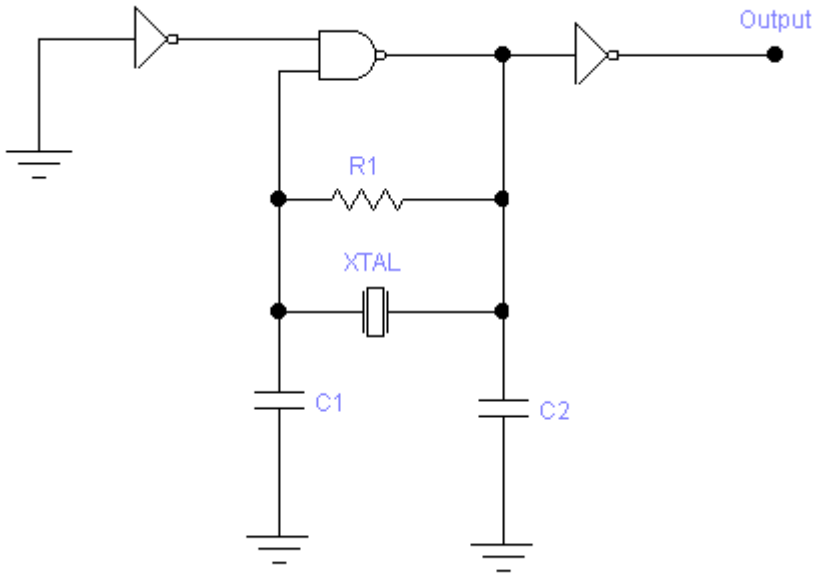
Ο διανομέας αυτός βρίσκεται στο πίσω μέρος του υπολογιστή μας. Ο κόμβος USB μπορεί να έχει μια έξοδο (upstream] και τέσσερις εισόδους (downstream) όπως φαίνεται παραπάνω. Σε κάθε μια είσοδο (downstream) μπορούμε να συνδέσουμε μια πανομοιότυπη συσκευή σαν αυτήν που κατασκευάσαμε στην ενότητα 2. Αν ενώσουμε αυτές τις συσκευές μεταξύ τους θα έχουμε ουσιαστικά ένα 32bit LFSR (4 φορές αυτό που εξετάσαμε στην ενότητα 2).

Επίσης μπορούμε να συνδέσουμε και άλλους διανομείς σε σειρά. Συνολικά έχουμε την δυνατότητα να συνδέσουμε μέχρι 127 συσκευές. Θεωρητικά λοιπόν αν είχαμε 127 ίδια LFSR των 8bit συνολικά μπορούμε να δημιουργήσουμε ένα των 1016bit.

Αυτός ο τρόπος δημιουργίας μεγάλου μήκους LFSR είναι πολύ βολικός γιατί αν υποθέσουμε ότι η συσκευή πωλείται στο εμπόριο ο κάθε χρήστης μπορεί να επιλέξει το μήκος του LFSR αγοράζοντας όσες πλακέτες επιθυμεί. Αν αργότερα χρειαστεί μεγαλύτερο μήκος ψευδοτυχαίων ακολουθιών δεν έχει παρά να αγοράσει ακόμα μερικές και να τις συνδέσει στον δίαυλο USB (αναβάθμιση συνολικού συστήματος).

Σαν επέκταση ακόμα μπορεί να θεωρηθεί η αύξηση της συχνότητας λειτουργίας. Για να γίνει αυτό καταρχήν θα πρέπει η συσκευή να υλοποιηθεί σε ολοκληρωμένο (όπως ένα FPGA που είδαμε στο κεφάλαιο 3) γιατί τα πολλά καλώδια πάνω σε μια διάτρητη πλακέτα βάζουν πολλούς περιορισμούς.

Στην επόμενη σελίδα δίνουμε την πρόταση μας για ένα γρήγορο Pattern Generator. Σύμφωνα μ' αυτήν το clock λειτουργίας του Pattern Generator δεν δίνεται από τον υπολογιστή αλλά από ένα εξωτερικό ταλαντωτή σαν αυτόν που φαίνεται παρακάτω[12].



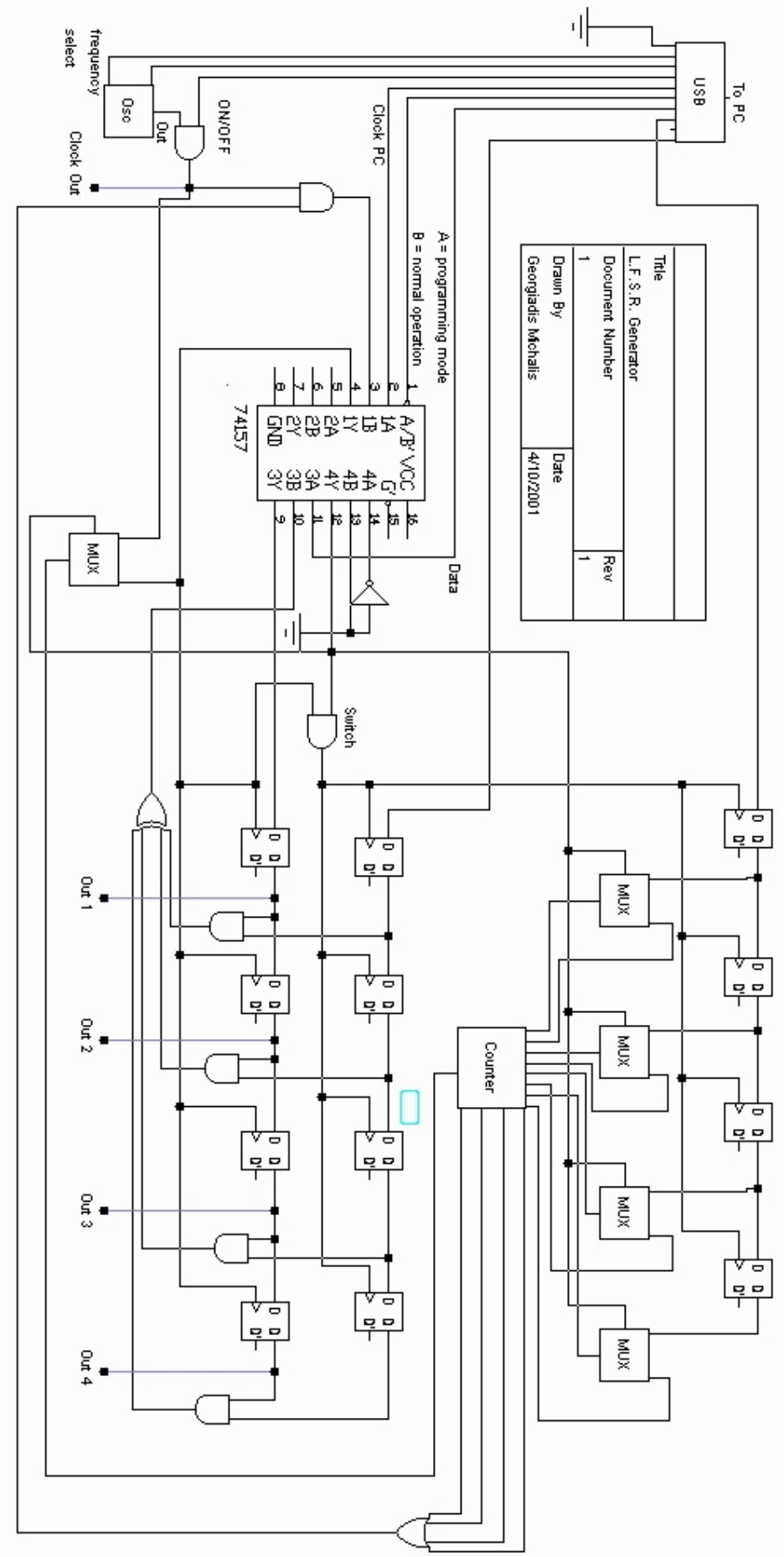
ταλαντωτής ελεγχόμενος από κρύσταλλο.

Επειδή το ξεκίνημα της διαδικασίας πρέπει να ελέγχεται από τον υπολογιστή συνδέουμε στην έξοδο (output) του ταλαντωτή μια πύλη AND σε συνδεσμολογία διακόπτη. Αν η μια της είσοδο που ελέγχεται από τον υπολογιστή έχει λογικό «1» τότε επιτρέπεται στους παλμούς clock να φτάσουν στα Flip Flops.

Η μεγάλη διαφορά όμως με την σχεδίαση της ενότητας 2 είναι ο μετρητής που προτείνουμε. Η χρήση του είναι να μετράει πόσα clock έχουν δοθεί στο LFSR. Αρχικά προγραμματίζεται πάλι με την χρήση ενός καταχωρητή ολίσθησης όπως γίνεται και η επιλογή του πολυωνύμου που έχουμε περιγράψει.

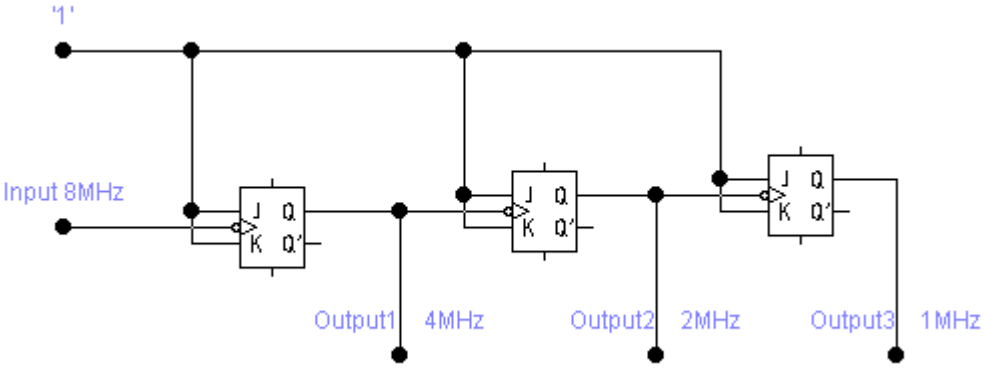
Όταν ο μετρητής αρχίζει να μετράει ουσιαστικά αφαιρεί από το περιεχόμενο των καταχωρητών ολίσθησης ένα για κάθε εισερχόμενο παλμό από αυτό που έχουμε φορτώσει. Αν χρησιμοποιήσουμε 8bit καταχωρητή ολίσθησης θα είναι δυνατόν να έχουμε έναν μετρητή προς τα κάτω (down counter) μέχρι 256 παλμούς χρονισμού. Όταν το περιεχόμενο του μετρητή μηδενίσει τότε με μια δεύτερη πύλη AND που επενεργεί στον ταλαντωτή διακόπτουμε την τροφοδοσία των παλμών στον Pattern Generator.

Το υπόλοιπο κύκλωμα δεν έχει διαφορά με αυτό που ήδη έχουμε δει (επιλογή πολυωνύμου και αρχικής τιμής), αν εξαιρέσουμε ότι το παρόν είναι 4 bit.



Επειδή ο Signature Analyzer δεν παρουσιάζει καμία διαφορά με αυτόν που ήδη έχουμε σχεδιάσει παραλείπουμε να τον σχεδιάσουμε αναλυτικά.

Μια άλλη πρόταση για επέκταση της συσκευής θα μπορούσε να είναι ο προγραμματιζόμενος ταλαντωτής[13]. Μέσω του υπολογιστή και του διαύλου USB είναι δυνατόν να διαιρέσουμε ένα ψηφιακό παλμοσειρμό δια του 2, δια του 4 κτλ. Ένα τέτοιο κύκλωμα κάνει χρήση των T Flip Flop που στην ουσία είναι J-K Flip Flop με τα J, K μόνιμα συνδεμένα σε λογικό «1». Ένας τέτοιος διαιρέτης δίνεται παρακάτω:



Μπορούμε πάλι με χρήση του διαύλου USB και χρήση πυλών AND σε συνδεσμολογία διακόπτη να διαλέξουμε να περάσει μόνο μια από τις τρεις αυτές εξόδους στην είσοδο clock του LFSR. Έτσι θα έχουμε ένα LFSR ρυθμιζόμενης συχνότητας μεταξύ τριών προεπιλεγμένων τιμών.

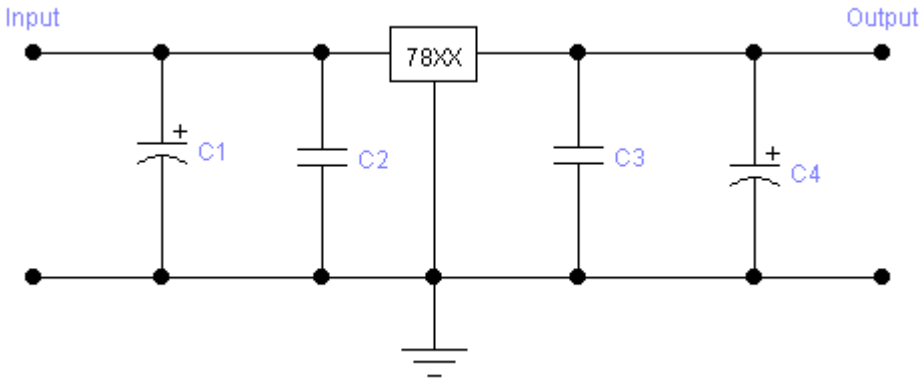
- ***Συμπεράσματα***

Από την κατασκευή της συσκευής βγάλαμε πολλά χρήσιμα συμπεράσματα όπως ότι η τροφοδοσία είναι ένας πολύ κρίσιμος παράγοντας για την σωστή λειτουργία μιας οποιασδήποτε συσκευής ιδιαίτερα σε ακολουθιακά κυκλώματα (κυκλώματα με clock). Πλέον μπορούμε να βγάλουμε το συμπέρασμα ότι όσο και καλή να είναι η σχεδίαση της πλακέτας, όσο έξυπνο και να είναι το ηλεκτρονικό σχέδιο, αν η τροφοδοσία δεν είναι επαρκώς σταθεροποιημένη τίποτα δεν θα δουλέψει σωστά.

Έτσι για την τροφοδοσία της συσκευής χρησιμοποιήσαμε σταθεροποιητές τριών ακροδεκτών (ολοκληρωμένα της σειράς 78XX). Η απαίτηση που έχουν αυτά τα ολοκληρωμένα είναι ότι η τάση εισόδου τους θα πρέπει να είναι τουλάχιστον 3V μεγαλύτερη από την τάση εξόδου που αναγράφεται πάνω στην συσκευασία του ολοκληρωμένου. Το πλεονέκτημα τους είναι ότι χρειάζονται λίγο χώρο πάνω στην πλακέτα και ελάχιστα εξωτερικά εξαρτήματα.

Επειδή παρουσιάστηκαν πολλά προβλήματα στην εφαρμογή μας λόγω κακής σταθεροποίησης της τάσης εξετάσαμε το θέμα σε βάθος.

Ένα τυπικό τροφοδοτικό που κάνει χρήση της σειράς ολοκληρωμένων 78XX είναι κάπως έτσι:

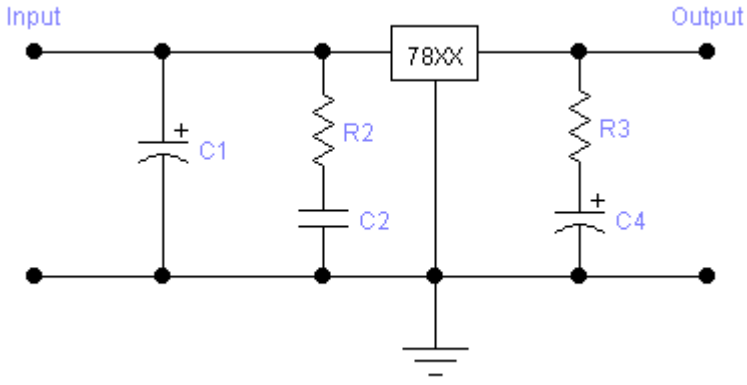


Ο πυκνωτής C1 φιλτράρει την τάση εισόδου στο ολοκληρωμένο από τυχόν κυματώσεις που μπορεί να οφείλονται σε κακή ποιότητα κατασκευής του ανορθωτή. Οι πυκνωτές C2 και C3 βελτιώνουν την σταθερότητα του ολοκληρωμένου ενώ ο ηλεκτρολυτικός C4 ενεργεί σαν μέσο αποθήκευσης ενέργειας για την περίπτωση που ζητηθεί στιγμιαία από το φορτίο μεγάλη ποσότητα[18].

Στην πράξη οι τιμές των πυκνωτών εξαρτώνται από το φορτίο αλλά και από την τάση λειτουργίας. Με εμπειρικούς κανόνες βρήκαμε ότι για την δική μας εφαρμογή έχουμε:

C1 = 1000μF
C2 = 100nF
C3 = 100nF
C4 = 47μF

Ψάξαμε αρκετά για να βρούμε γιατί προκαλούνται προβλήματα δυσλειτουργίας ακόμα και με την χρήση του τυπικού σταθεροποιητή με τις κατάλληλες τιμές πυκνωτών που είδαμε παραπάνω. Η απάντηση που πήραμε ήταν ότι έχουμε πολύ γρήγορες μεταβολές του φορτίου και το ολοκληρωμένο δεν μπορεί να ανταποκριθεί. Τελικά καταφύγαμε σε μια εντελώς εμπειρική λύση που μετατρέπει το τυπικό κύκλωμα σ' αυτό που φαίνεται παρακάτω.



Η μόνη διαφορά που παρατηρούμε στο κύκλωμα σε σχέση με πριν είναι ότι προστέθηκαν δυο αντιστάσεις και αφαιρέθηκε ο πυκνωτής C3. Πρέπει να αναφέρουμε ότι δεν υπάρχει σαφή επιστημονική εξήγηση γιατί αυτές οι δυο αντιστάσεις βελτιώνουν την απόδοση. Με εμπειρικούς κανόνες προσδιορίσαμε και πάλι τις τιμές των στοιχείων[18].

C1 = 1000μF
C2 = 100nF
R2 = 10Ω
C4 = 470μF
R4 = 0,33Ω

Στην πράξη μπορέσαμε πράγματι να διακρίνουμε καλύτερη συμπεριφορά του συνολικού κυκλώματος. Ενώ παλαιότερα πολλές φορές το γέμισμα των καταχωρητών ολίσθησης από τον υπολογιστή δεν γινόταν σωστά, παρατηρήσαμε ότι με αυτήν την μέθοδο σχεδόν πάντα δεν υπάρχει δυσλειτουργία. Ο λόγος που μερικές φορές δεν έχουμε καλή μεταφορά δεδομένων μπορεί να οφείλεται στον υπολογιστή, στον μικροελεγκτή που χρησιμοποιούμε ή στις παρεμβολές που προκαλούν τα πολλά καλώδια πάνω στην πλακέτα.

7. Βιβλιογραφία

[1] Prawat Nagvajara et al, "*Pseudorandom testing for boundary scan. Design with built-in self test*", IEEE Test & Design of Computers, 1991

[2] Peter Hortensius et al., "*Cellular automat based in signature analysis for built-in self test*", IEEE, Transactions of Computers, vol 39, October 1990

[3] Miller – Mil, "LFSRs as functional blocks in wireless applications", Xilinx, 2001

[4] Tom Balph, "LFSR counters implement binary polynomial generators", Motorola

[5] Ahmad A et al., "*Are primitive polynomials always best in signature analysis?*", IEEE Test & Design of Computers, 1990

[6] Peter Norton, "*Εισαγωγή στους υπολογιστές* ", εκδόσεις Τζιόλα, 1995

[7] Andrew S. Tanenbaum, "*Αρχιτεκτονική των υπολογιστών*", Amsterdam, 1995

[8] Πετρούτσος Ε, "*Πλήρες εγχειρίδιο Visual Basic 6*", εκδόσεις Γκιούρδας

[9] Greg Perry και Sanjaya Hettihewa, "*Μάθετε την Visual Basic 6 σε 24 ώρες*", εκδόσεις Γκιούρδας

[10] Gilmore, "*Μικροεπεξεργαστές θεωρία και εφαρμογές 2η έκδοση*", εκδόσεις Τζιόλα, 1996

[11] Kaufman – Seidman, "*Εγχειρίδιο ηλεκτρονικής*", εκδόσεις Τζιόλα, 1992

[12] Albert Paul Malvino, "*Ηλεκτρονική Malvino 5η έκδοση*", εκδόσεις Τζιόλα, 1995

[13] Morris Mano, "*Ψηφιακή σχεδίαση 2η έκδοση*", Prentice Hall Publishing – Παπασωτηρίου, Los Angeles, 1992

[14] Leach – Malvino, "*Ψηφιακά Ηλεκτρονικά, θεωρία και εφαρμογές 5η έκδοση*", εκδόσεις Τζιόλα, Santa Clara, 1996

[15] Allen Dewey, "*Analysis and design of digital systems with VHDL*", PWS Publishing, Boston, 1997

[16] John F. Wakerly, "*Digital design, principles and practices 3rd edition*", Prentice Hall Publishing, Stanford University, 2000

[17] Περιοδικό Ελέκτορ, "*Ενδιάμεσο USB*", τεύχη 233 και 234, Αθήνα, 2002

[18] Περιοδικό Ελέκτορ, "*Σταθεροποιητές τάσης στην πράξη*", τεύχος 231, Αθήνα, 2002

[19] Serial ATA workgroup, "*High speed serialized AT attachment*", August 2001 **Cd:\E-FILES\serial ATA 1.0.pdf**

- **Χρήσιμες δικτυακές διευθύνσεις**

http://www.usb.org Εδώ είναι οι κατασκευαστές της USB, δηλαδή αντιπρόσωποι κάποιων μεγάλων εταιριών οι οποίοι δημιούργησαν μια ομάδα και καθόρισαν το πρωτόκολλο USB.

http://www.cypress.com Πληροφορίες για τον μικροεπεξεργαστή USB, τον CY7C63001, που χρησιμοποιήσαμε στην εφαρμογή μας.

http://www.anchorchips.com Μικροεπεξεργαστής USB που βασίζεται σε έναν επεξεργαστή συμβατό με 8051, με εσωτερική RAM

http://www.semiconductor.philips.com Philips Semiconductors.
Η Philips διαθέτει δύο πομποδέκτες κατάλληλους για υποστήριξη διαύλων USB υψηλής ταχύτητας.
Το *PDIUSBD11* περιέχεται σε μια Θήκη 16 ακίδων, διαθέτει δύο FIFO των 8 ψηφίων και προγραμματίζεται από οποιονδήποτε μικροελεγκτή μέσω διαύλου I2C.
Το *PDIUSBD12* είναι ένα ακόμα ολοκληρωμένο USB υψηλής ταχύτητας συμβατό με 8051, που επικοινωνεί με οποιονδήποτε μικροελεγκτή μέσω μιας παράλληλης θύρας.

http://www.infineon.de Η εταιρία αυτή κατασκευάζει και διαθέτει στην αγορά τον C541U, έναν μικροελεγκτή υψηλής ταχύτητας συμβατό με τον 8051. Η κύρια χρήση του συγκεκριμένου μικροελεγκτή έχει να κάνει με τις τηλεπικοινωνίες και συγκεκριμένα με τα μόντεμ ISDN.

http://sps.motorola.com Η Motorola, μία από τις μεγαλύτερες εταιρίες κατασκευής ημιαγωγών, διαθέτει τον 68HC705JB3, έναν μικροελεγκτή USB χαμηλής ταχύτητας.

http://www.microchip.com Οι μικροελεγκτές που κατασκευάζει η Microchip για την υποστήριξη του διαύλου USB, εκτός από τη μηχανή USB περιλαμβάνουν επίσης έναν μετατροπέα A/D και ένα UART PIC16C745.