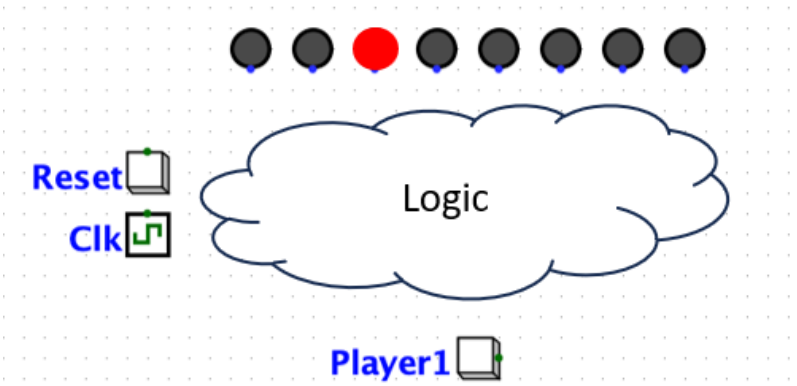
ΕΙΚΟΝΙΚΟ ΕΡΓΑΣΤΗΡΙΟ #2
Το παιχνίδι του Ping Pong

Τσαλκιτζής Νικόλαος 03121123
Ρουσουνέλου Αικατερίνη 03121846

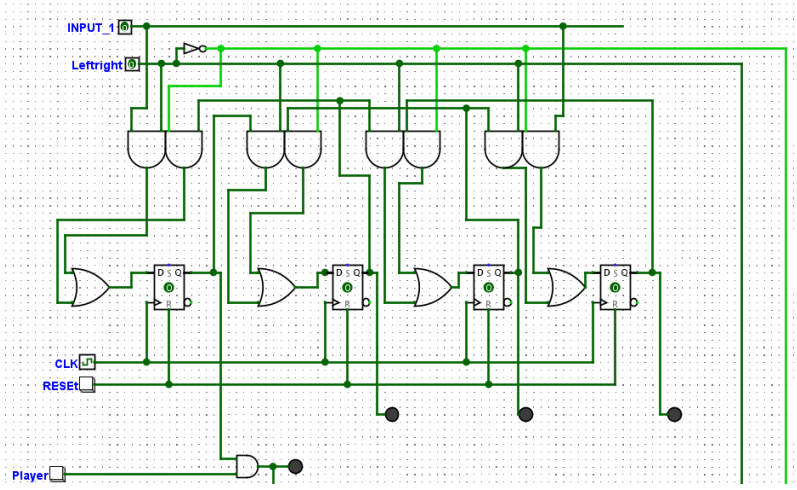
Εισαγωγή

Αρχικά, θα αναλύσουμε αφαιρετικά την λειτουργία του παιχνιδιού και θα υλοποιήσουμε μια μηχανή πεπερασμένων καταστάσεων. Τα components του, είναι το σήμα Reset μέσω του οποίου σε οποιοδήποτε στάδιο μπορούμε να επανέλθουμε στην αρχική κατάσταση και μέσω του πατήματος αυτού του button εκκινούμε.Έπειτα, χρησιμοποιούμε ένα ακόμη πλήκτρο: PLAYER_1 , μέσω του οποίου ο χρήστης μπορεί να αποκρούει την εικονική μπάλα όταν αυτή βρίσκεται σε κάποια από τις ακραίες θέσεις (με δείκτη 1 ή N), ενώ εάν πατηθεί σε κάποια από τις ενδιάμεσες καταστάσεις ο παίκτης χάνει και επιστρέφει στην αρχική κατάσταση. Τα σήματα RESET, PLAYER_1 θα αποτελούν τους τελεστές μετάβασης κατάστασης και η θέση της εικονικής μπάλας μέσω των LED κωδικοποιείται ως δείκτης κατάστασης (ως q0 θα θεωρήσουμε την κατάσταση όπου θεωρούμε ότι το RESET δεν εχει πατηθεί και κατ επέκταση δεν έχει αρχίσει το παιχνίδι).



Επομένως, οφείλουμε να χρησιμοποιήσουμε για την ανάλυση μας την είσοδο του RESET,PLAYER_1 και την μοντελοποίηση για την φορά κίνησης της μπάλας και θα πρέπει να έχουμε ένα κύκλωμα ελέγχου τόσο για τις ακραίες θέσεις (όπου εκεί θα πρέπει να αλλάζει η φορά της μπάλας) όσο και όταν είμαστε στις μεσαίες-ενδιάμεσες θέσεις (όπου εκει αν πατηθεί το κουμπί Player η κατεύθυνση δεν αλλάζει αλλά πρέπει να χάνει ο παίκτης και να επιστρέφουμε στην αρχική κατάσταση).

Ανάλυση Με Καταχωρητή Διπλής Ολίσθησης



Αρχικά, θα δούμε μια υλοποίηση του καταχωρητή διπλής ολίσθησης μέσω του Logisim, που παρουσιάζεται στην διπλανή εικόνα.Η κατασκευή αυτή, εκτελεί δεξιά ολίσθηση όταν το σήμα Leftright είναι 0 και όταν είναι 1 εκτελείται αριστερή ολίσθηση.
Εδώ πρέπει να επισημάνουμε ότι το κύκλωμα έχει σχεδιαστεί για 4 LED και προφανώς μπορεί να επεκταθεί και για μεγαλύτερο αριθμό LED. Απομένει λοιπόν, να σχεδιάσουμε μία δομή ελέγχου η οποία τσεκάρει τις προϋποθέσεις της εισαγωγής. Για τις ενδιάμεσες καταστάσεις μπορούμε να χρησιμοποιήσουμε μία πύλη OR , η οποία ως inputs την κατάσταση των LED 1,..,N-2 έτσι ώστε η

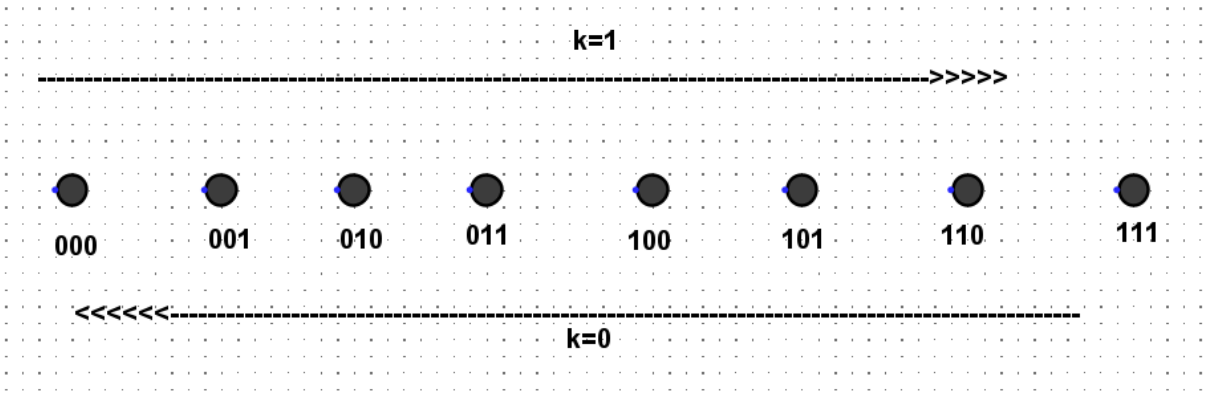
έξοδος της OR να κωδικοποιεί **εάν** ένα από τα ενδιάμεσα λαμπάκια είναι ON. Παράλληλα, λαμβάνοντας το αποτέλεσμα της πύλης OR και εκτελώντας το λογικό AND με το button PLAYER, το output θα κωδικοποιεί την πρώτη συνθήκη και αν το αποτέλεσμα του λογικού ΚΑΙ είναι 1 εκτελούμε reset στα FFs και επιστρέφουμε στην αρχική κατάσταση.

Για την δεύτερη προϋπόθεση πρέπει η μπάλα να αρχίσει να κινείται από τα δεξιά προς τα αριστερά. Αυτό επιτυγχάνεται με με τον συνδυασμό input/Leftright=1,1 και μόλις έρθει παλμός ρολογιού το πρώτο από δεξιά ανάβει.Στο σημείο αυτό, εμείς αλλάζουμε την κατάσταση σε input/Leftright=0,1. Γίνεται propagate της εικονικής μπάλας και μόλις φτάσει στην τελευταία θέση τότε θετουμε input/Leftright=1,0 στην αρνητική ακμή θέτουμε την κατάσταση και η άφιξη ενός νέου παλμού σηματοδοτεί την ισότητα:input/Leftright=0,0. Για την ρύθμιση της κατεύθυνσης, χρησιμοποιείται το σήμα Leftright. Εμεις πρακτικά θέλουμε όταν το player πατηθεί και το LED είναι σε ακραία θέση (αυτή η έκφραση κωδικοποιείται ως μία λογική πύλη AND μεταξύ του output της κάθε ακραίας θέση με το button του παίκτη) αν το αποτέλεσμα είναι 1 τότε αντιστρέφουμε το σήμα Leftright. Αυτό θα γίνει με ένα T FF(ένα για την θέση 0 και ένα για την N-1) και ως enable θα χρησιμοποιήσουμε το αποτέλεσμα της AND.

Μένει να επιλέξουμε το σήμα Leftright ή το ανεστραμμένο του, προκειμένου να τροφοδοτηθούν τα FFs. Για αυτόν τον σκοπό, χρησιμοποιούμε πολυπλέκτες, τόσους όσα και τα FFs. Αν έχει πατήσει το κουμπί ο χρήστης, τότε επιλέγουμε το αντίστοιχο σήμα κατεύθυνσης. Προφανώς αφού θα αλλάζει το Leftright πρέπει να φροντίσουμε και την αλλαγή του αντιστρόφου του για να λειτουργεί ομαλά το κύκλωμα. Τελικά το κύκλωμα,δεδομένης της υλοποίησης του παραπάνω register αμφίδρομης φοράς, περιλαμβάνει επιπρόσθετα δυο TFF για την κίνηση της μπάλας μία πύλη AND και 4 MUX. Εν γένει τα TFFs είναι φθηνότερα των DFFs και τα τελευταία πιο φθηνα των JK.

Ανάλυση Με Ακολουθιακό Κύκλωμα

Θα ασχοληθούμε με το πρόβλημα των 8 LEDs το οποίο εύκολα μπορεί να γενικευθεί όσο αφορά το μεγαλύτερο πλήθος είτε με σειριακή σύνδεση δύο τέτοιων κυκλωμάτων είτε με νοηματική επέκταση του πρώτου. Όπως και πριν, πρέπει να βρίσκουμε συνέχεια την κατεύθυνση της νοητής μπάλας σε κάθε πιθανή κατάσταση και για αυτόν τον λόγο χρησιμοποιούμε ένα Flip Flop. Μετά όσον αφορά τις καταστάσεις, χρειαζόμαστε 3 FF για την περιγραφή για 8 LEDs, ή γενικότερα χρειαζόμαστε log2n με n το πλήθος των led. Ταυτόχρονα με την παρουσία του δυαδικού αριθμού με τρια bits ο οποίος δείχνει την θέση της μπάλας έχουμε δύο εξωτερικά σήματα εισόδου μη συγχρονισμένα με το ρολόι. Το ένα είναι το κουμπί του παίκτη (button player1), που χρησιμεύει για την απόκρουση της μπάλας στις ακριανές θέσεις και το άλλο είναι το Reset το οποίο αρχικοποιεί την εκκίνηση του παιχνιδιού.Σε περίπτωση λάθους του παίκτη, επιστρέφει στην πρωτεύουσα κατάσταση. Για τις ανάγκες του πίνακα αλήθειας, ονομάζουμε το πρώτο p και το δεύτερο r. Τέλος, ορίζουμε ένα bit-control το οποίο είναι υπεύθυνο για την έκφραση της κατεύθυνσης της μπάλας σε πλήρη συγχρονισμό με το ρολόι. Αυτό δεν το ελέγχει ο χρήστης, και η προς τα δεξιά κίνηση καθορίζεται με 1 ενώ η προς τα αριστερά με 0 και αυτό καλείται ως κ. Ενδεικτικά ένα κύκλωμα για μια απλοποιημένη αναπαράσταση:



Δεδομένου ότι έχουμε 6 εισόδους έχουμε 2^6=64 καταστάσεις τις οποίες αναλύουμε στον παρακάτω πίνακα:

		Παρούσα Κατάσταση F-F				**Επόμενη Κατάσταση F-F**				**Είσοδοι D F-F**			
r	p	K_n	A_n	B_n	C_n	K_{n+1}	A_{n+1}	B_{n+1}	C_{n+1}	K	A	B	C
0	0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	1	1	0	0	1	1
0	0	0	0	1	1	0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	1	0	1	0	1	0	1
0	0	0	1	0	1	0	1	1	0	0	1	1	0
0	0	0	1	1	0	0	1	1	1	0	1	1	1
0	0	0	1	1	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	0	0	0	1	0	0	0
0	0	1	0	1	0	1	0	0	1	1	0	0	1
0	0	1	0	1	1	1	0	1	0	1	0	1	0
0	0	1	1	0	0	1	0	1	1	1	0	1	1
0	0	1	1	0	1	1	1	0	0	1	1	0	0
0	0	1	1	1	0	1	1	0	1	1	1	0	1
0	0	1	1	1	1	1	1	1	0	1	1	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	0	0	0	0	0	0	0
0	1	0	1	1	1	1	1	1	1	1	1	1	1
0	1	1	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	0	0	0	0	0	0	0	0

0	1	1	0	1	0	0	0	0	0	0	0	0	0
0	1	1	0	1	1	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	1	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0	0

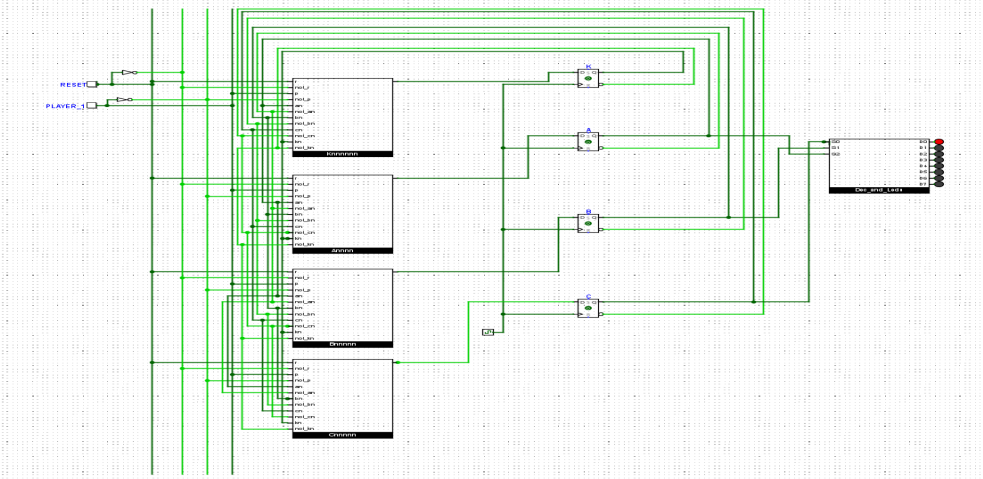
1	1	0	1	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	1	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0

Αξιοποιώντας τον παραπάνω πίνακα, έχουμε ότι όλες οι καταστάσεις των flip flops είναι εξαρτήσεις των 6 λογικών μεταβλητών, των δύο εισόδων και της προηγούμενης κατάστασης των τεσσάρων FFs. Σε αυτό το σημείο να τονίσουμε πώς τα ABC αποτελούν τη κωδικοποίηση της θέσης της εικονικής μπάλας.

Χρησιμοποιώντας ανάλυση Karnaugh οι τελικές συναρτήσεις για την επόμενη κατάσταση είναι οι ακόλουθες:

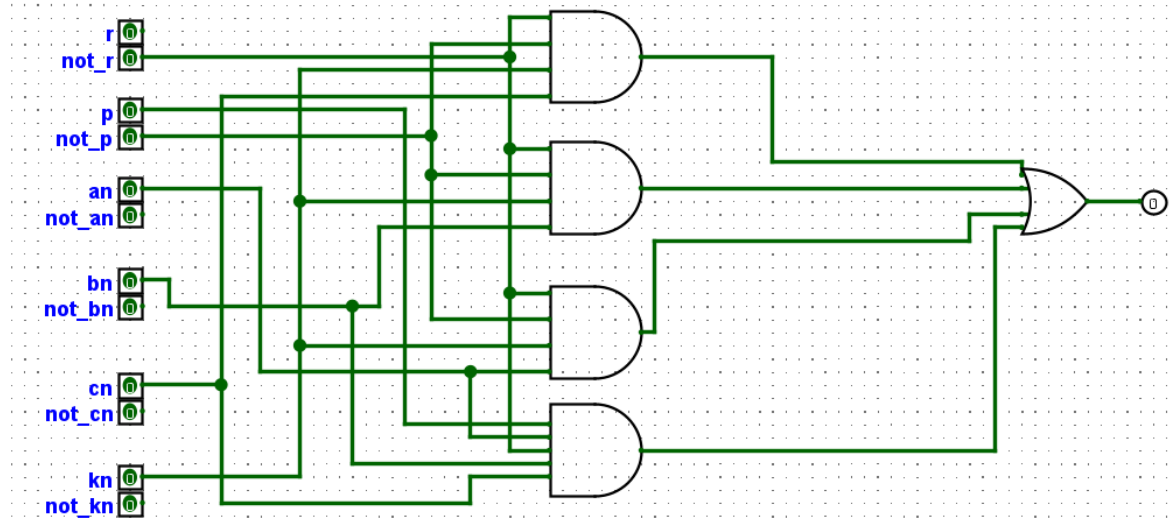
- **Kn+1(r, p, Kn, An, Bn, Cn) = r'p'KnCn + r'p'KnBn + r'p'KnAn + r'pAnBnCn**
- **An+1(r, p, Kn, An, Bn, Cn) = r'p'Kn'An'BnCn + r'pAnBnCn + r'p'Kn'AnBn' + r'p'AnBnCn' + r'p'KnAnCn**
- **Bn+1(r, p, Kn, An, Bn, Cn) = r'p'Kn'Bn'Cn + r'p'Kn'BnCn' + r'p'KnBnCn + r'p'KnAnBn'Cn' + r'pAnBnCn**
- **Cn+1(r, p, Kn, An, Bn, Cn) = r'p'Kn'Cn' + r'p'BnCn' + r'p'AnCn' + r'pAnBnCn**

Δεδομένων των παραπάνω δημιουργήσαμε την παρακάτω υλοποίηση στο Logisim-Evolution η οποία επισυνάπτεται μαζί με την αναφορά μας.



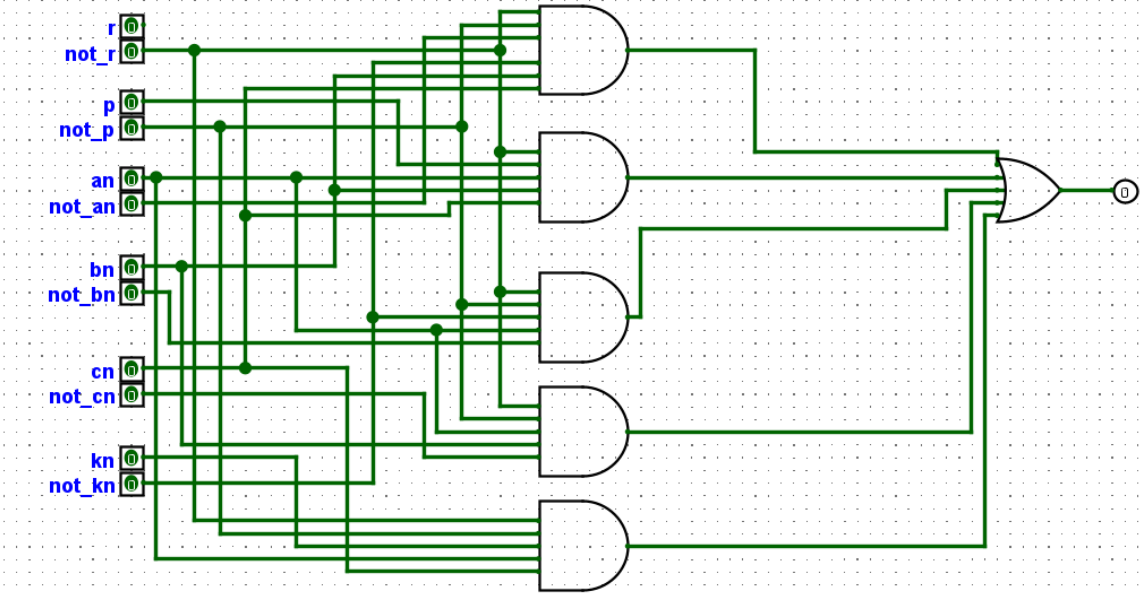
Implementation of Kn FF

r'p'KnCn + r'p'KnBn + r'p'KnAn + r'pAnBnCn

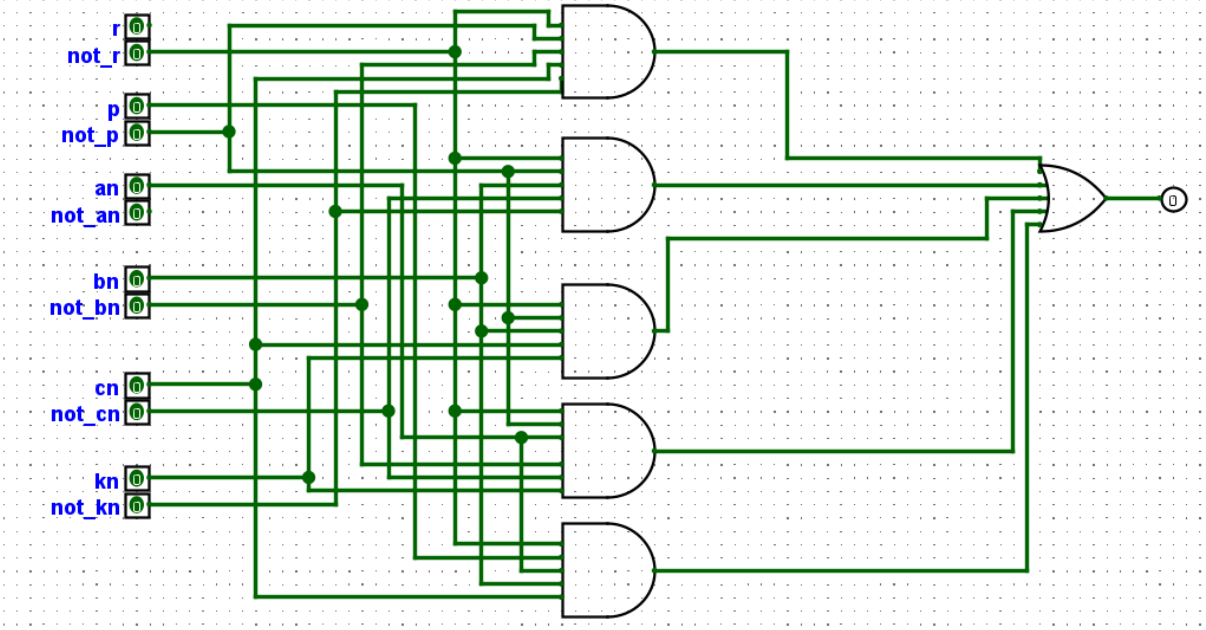


Implementation of An FF

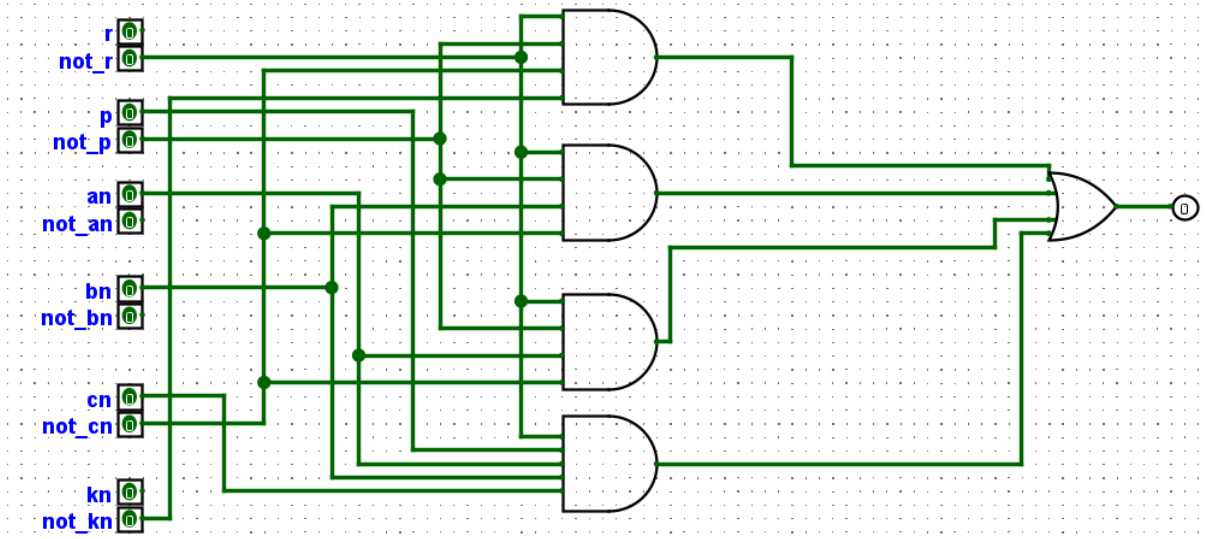
r'p'Kn'An'BnCn + r'pAnBnCn + r'p'Kn'AnBn' + r'p'AnBnCn' + r'p'KnAnCn

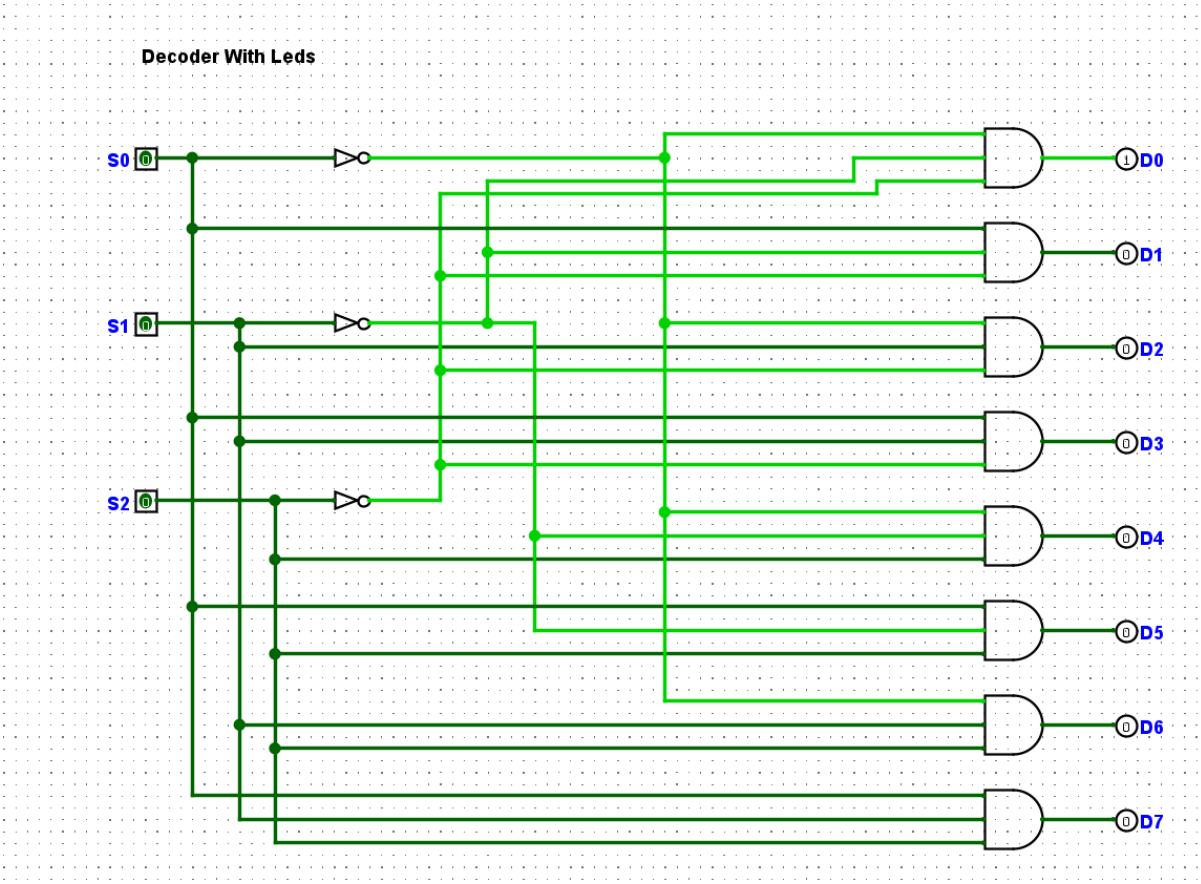


Implementation of Bn FF r'p'Kn'Bn'Cn + r'p'Kn'BnCn' + r'p'KnBnCn + r'p'KnAnBn'Cn' + r'pAnBnCn



Implementation of Cn FF r'p'Kn'Cn' + r'p'BnCn' + r'p'AnCn' + r'pAnBnCn





Ανάλυση Με Μετρητή

Σε αυτή τη περίπτωση, θα αναλύσουμε την υλοποίηση με βάση τον μετρητή που μετράει την θέση της μπάλας. Παράλληλα με αυτόν θα μπορούσε να χρησιμοποιήσουμε ακόμη έναν αφαιρέτη. Αυτό συμβαίνει, γιατί όταν θέλουμε να αλλάξουμε κατεύθυνση χρησιμοποιούμε εναλλαγή αθροιστή αφαιρέτη.Θεωρώντας φορά από αριστερά προς τα δεξιά (από την θέση 0 στην N-1), ο αθροιστής προσθέτει διαδοχικά. Μόλις φτάσει στην θέση N-1 υπάρχουν δύο περιπτώσεις. Εάν ο παίκτης δεν πατήσει το κουμπί χάνει και πηγαίνει στην αρχική κατάσταση, ενώ εάν πατηθεί τότε ενεργοποιείται ο αφαιρέτης για την αλλαγή κατεύθυνσης. Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας μια πύλη AND στην οποία οι είσοδοι αντικατοπτρίζουν το αν είναι set το τελευταίο LED και το button του παίκτη. Τότε εαν το output είναι 1 τότε πρέπει να αλλάξουμε σε λειτουργία αφαιρέτη παρέχοντας την έξοδο της AND ως enable στον αφαιρέτη. Αν είναι 0 απενεργοποιείται ο αφαιρέτης , και γίνεται reset ο αθροιστής για να αρχίσει να μετράει εκ νέου από το 0, αφού πρακτικά είμαστε στην περίπτωση που ο παίκτης χάνει. Ακριβώς ίδιο σκεπτικό εφαρμόζουμε στην αντίθετη περίπτωση. Δηλαδή, όταν ο αφαιρέτης φτάσει στο 0 πρέπει να ελέγχουμε εάν το κουμπί του παίκτη είναι ON. Εαν είναι ON τότε ενεργοποιείται ο adder και συνεχίζει στην άλλη κατεύθυνση. Σε αντίθετη περίπτωση, αν χάσει επανερχόμαστε στην αρχική κατάσταση. Προφανώς χρησιμοποιώντας μια πύλη OR με εισόδους όλα τα ενδιάμεσα LED ελέγχουμε αν είναι ένα από αυτά ON και εκτελώντας λογικό και με την κατάσταση του πληκτρου, έχουμε έλεγχο του αν ο παίκτης χάνει. Εναλλακτικά μπορούμε να χρησιμοποιήσουμε έναν μόνο ADDER και ανάλογα με την διεύθυνση του κρατούμενου μπορούμε να εναλλάξουμε σε λειτουργία αφαιρέτη. Αυτή είναι τελικά μια οικονομικότερη υλοποίηση καθώς μπορούμε με έναν αθροιστή να μοντελοποιήσουμε την λειτουργία αφαιρέτη.

Χονδροειδής Σύγκριση Υλοποιήσεων

Σε ένα τόσο αρχικό στάδιο δεν μπορούμε να παράξουμε αναλυτικά αποτελέσματα αναφορικά με το ποιά υλοποίηση είναι καλύτερη καθώς έχουμε μόνο ένα θεωρητικό πλαίσιο αυτών. Πιο συγκεκριμένα θα προσπαθήσουμε να αναλύσουμε διάφορους παράγοντες.

Ας αναλύσουμε τις τρεις υλοποιήσεις σε σχέση με το κόστος, τον χρόνο και την επεκτασιμότητά τους:

Υλοποίηση με Μετρητή:

- **Κόστος**: Η υλοποίηση αυτή απαιτεί τη χρήση μετρητή. Ο μετρητής μπορεί να απαιτήσει επιπλέον υλικό, όπως flip-flops και λογική πύλες, ανάλογα με τον αριθμό των LEDs και τις απαιτήσεις του συστήματος.
- **Χρόνος**: Η υλοποίηση αυτή μπορεί να απαιτήσει περισσότερο χρόνο για την ανάπτυξη και τον έλεγχο του κώδικα, καθώς απαιτεί περισσότερη συνολική λογική.
- **Επεκτασιμότητα**: Η υλοποίηση με μετρητή είναι επεκτάσιμη, καθώς μπορεί να προσαρμοστεί εύκολα για μεγαλύτερο αριθμό LEDs με αντίστοιχη προσθήκη flip-flops και λογικής.

Υλοποίηση με Διπλό Καταχωρητή:

- **Κόστος**: Αυξάνει το κόστος λόγω της ανάγκης για διπλό καταχωρητή.
- **Χρόνος**: Η ανάπτυξη μπορεί να απαιτήσει περισσότερο χρόνο λόγω της περισσότερης πολυπλοκότητας που ενέχει η χρήση διπλού καταχωρητή.
- **Επεκτασιμότητα**: Μπορεί να είναι περιορισμένη στην επέκταση, καθώς η προσθήκη περισσότερων LEDs θα απαιτούσε περισσότερους διπλούς καταχωρητές και πολυπλοκότερη λογική.

Υλοποίηση Ακολουθιακής Λογικής:

- **Κόστος**: Η χρήση ακολουθιακής λογικής ενδέχεται να απαιτήσει λιγότερους πόρους σε σύγκριση με τις άλλες υλοποιήσεις.
- **Χρόνος**: Μπορεί να απαιτήσει λιγότερο χρόνο για την ανάπτυξη, καθώς η ακολουθιακή λογική είναι συνήθως πιο εύκολη στην κατανόηση και την ανάπτυξη.
- **Επεκτασιμότητα**: Η επεκτασιμότητα μπορεί να είναι περιορισμένη, ανάλογα με την πολυπλοκότητα του συστήματος. Ωστόσο, ανάλογα με τον τρόπο υλοποίησης, μπορεί να είναι δυνατή η προσθήκη περισσότερων LEDs με σχετικά χαμηλό κόστος.

Συνολικά, η επιλογή της κατάλληλης υλοποίησης εξαρτάται από πολλούς παράγοντες, όπως οι απαιτήσεις του συστήματος, οι περιορισμοί στους πόρους και οι αναμενόμενοι χρόνοι ανάπτυξης

Επεκτάσεις

Για την προσθήκη δεύτερου παίκτη, πρέπει σίγουρα να προσθέσουμε ένα πλήκτρο Player_2. Οφείλουμε ακόμη να έχουμε κύκλωμα ελέγχου, το οποίο ρυθμίζει ότι οι παίκτες παίζουν διαδοχικά.

Πριν από αυτό, πρέπει να αναθέσουμε μία πλευρά σε κάθε παίκτη. Άρα, η πολυπλοκότητα αυξάνεται καθώς προστίθεται ένας ακόμα παίκτης τόσο λόγω του βοηθητικού κυκλώματος όσο και λόγω της προσθήκης πολυπλεκτών για την επιλογή του παίκτη που παίζει κάθε φορά σήμα επιλογής το ποιός έπαιξε την προηγούμενη φορά.

Εν γένει πρέπει να προσθέσουμε πολυπλέκτη για την επιλογή του παίκτη που παίζει την i-οστη φορά καθώς και ένα κύκλωμα που παράγει το σήμα επίτρεψης των MUX. Τελικά μπορούμε να εντάξουμε δεύτερο παίκτη με όχι τόσο μεγάλη πολυπλοκότητα. Το buzzer μπορεί να ενεργοποιείται όταν ένας παίκτης χάνει ή όταν αποκρούεται η μπάλα. Και σε αυτό το κύκλωμα η προστιθέμενη πολυπλοκότητα δεν είναι σημαντικά μεγάλη και για την πρώτη συνθήκη μπορούμε μέσω μίας πύλης OR η οποία έχει ως είσοδο όλα τα ενδιάμεσα LED και η έξοδος θα δείχνει εαν είναι ON ένα από αυτά. Μετά με AND με την έξοδο των πολυπλεκτών και την έξοδο της OR ενεργοποιούμε το BZZRT. Για την δευτερη υλοποιούμε την ίδια λογική απλά χρησιμοποιώντας τα ακραία LED.

Για την προσθήκη του score table, μπορούμε είτε να χρησιμοποιήσουμε 4 LED για κάθε παίκτη και με δυαδικό τρόπο να κρατάμε το score για κάθε παίκτη είτε να χρησιμοποιήσουμε την οθόνη led εκ νέου μια για κάθε παίκτη μέσω της οποίας θα κρατάμε το σκορ του καθένα. Με απόκρουση κερδίζεται ένας πόντος,ενώ χάνεται εάν γίνει κρούση του button όταν ενα απο τα ενδιάμεσα LEDs είναι ON.

Μια ωραία ιδέα είναι να δημιουργήσουμε ένα κύκλωμα σύνδεσης αυτών των οθονών LED και κάθε φορά που ένα παίκτης κάνει λάθος, να χάνει έναν πόντο να τον κερδίζει ο άλλος. Αυτό θα μπορούσε να είναι ένα πιο δύσκολο version του αρχικού και οι χρήστες θα έχουν την δυνατότητα επιλογής εύκολου ή δύσκολου επιπέδου μέσω ενός πολυπλέκτη. Επιπλέον, θα μπορούσαμε να κάνουμε ακόμα πιο challenging το παιχνίδι έχοντας διαφορετική (πιο γρήγορη) ταχύτητα μπάλας για αυτόν που κερδίζει. Κάτι τέτοιο είναι προφανώς πιο ακριβό, καθώς θα πρέπει να έχουμε δυο ρολόγια με διαφορετικές συχνότητες και ο MUX αυτών θα πρέπει να ελέγχεται από την διαφορά των πόντων των παικτών και αν είναι μεγαλύτερη απο ενα threshold τότε μπορούμε να ενεργοποιήσουμε το πιο γρήγορο ρολόι. Τέλος, μπορούμε ακόμη να θέσουμε ότι το παιχνίδι τελειώνει στα 3 και όταν κάποιος φτάσει σε αυτό το πλήθος πόντων μπορούμε να έχουμε μια πύλη OR των δύο οθονών LED και εαν είναι κάποια ίση με τρια αυτό να είναι ένα σήμα επίτρεψης στο πληκτρολόγιο που διαθετει το λογισμικό και να τυπώνουμε ένα μήνυμα νίκης στον νικήτη και ένα αντίστοιχο στον ηττημένο.