Πρόσθεση/Αφαίρεση

Γκέκας Γεώργιος: 2423

Μαραγκός Παναγιώτης: 2472

Εφαρμογές της πράξης, υλοποίηση και βελτιστοποιήσεις

Που χρησιμοποιείται

- Όχι μόνο στις αμιγείς αριθμητικές πράξεις της πρόσθεσης και αφαίρεσης π.χ. συνηθισμένους υπολογισμούς
- Είναι παρούσα παντού μέσα στην λειτουργία ενός ολοκληρωμένου
  - Στον καθορισμό της τιμής του program counter π.χ. στα διάφορα jumps. Δηλαδή προσθέτει ή αφαιρεί από τον PC, τόσο όσο χρειάζεται για να μεταβούμε στο σημείο εκτέλεσης που εμείς θέλουμε, διακόπτοντας την φυσιολογική ροή του προγράμματος.
  - Στη στοίβα έτσι ώστε να αυξάνουμε ή να μειώνουμε κάθε φορά τον δείκτη (index) της στοίβας στην οποία κρατάμε την κατάσταση του προγράμματος που θα πρέπει αργότερα να μεταβούμε
  - Στην έμμεση διευθυνσιοδότηση προκειμένου να υπολογιστεί η διεύθυνση στην οποία πρέπει να μεταβούμε για να πραγματοποιηθεί η εντολή π.χ. add counter, var[80]

- Μπορεί να αντικαταστήσει ακόμα και πολύπλοκες πράξεις όπως αυτές του πολλαπλασιασμού και της διαίρεσης.
  - Πολύ βολικό σε φτηνά και μικρά κυκλώματα που δεν έχουμε ξεχωριστούς πολλαπλασιαστές/διαιρέτες. Η υλοποίηση των πράξεων αυτών επιτυγχάνεται με το λογισμικό του chip ύστερα από επαναλαμβανόμενες προσθέσεις (στην περίπτωση του πολλαπλασιασμού) ή επαναλαμβανόμενες αφαιρέσεις (στην περίπτωση της διαίρεσης).

Σπουδαιότητα της πράξης

- **Παίζει, λοιπόν, κυρίαρχο ρόλο. Θα πρέπει να είναι (ΟΠΩΣΔΗΠΟΤΕ) bug-free και αρκετά γρήγορη.**
- **Bugs στο υποσύστημα της πρόσθεσης/αφαίρεσης δεν οδηγούν μόνο σε λάθος αριθμητικά αποτελέσματα αλλά και σε λάθος εκτέλεση κώδικα (π.χ. λάθος στην υπολογιζόμενη τιμή του pc μπορεί να είναι καταστροφική).**
- **Χρησιμοποιούνται πολύ στην floating point αριθμητική. Π.χ. Χρειαζόμαστε δύο προσθέτες/αφαιρέτες. Έναν για την mantissa και έναν για τον εκθέτη.**
- **Είναι χρονολογικά η παλαιότερη και γι' αυτό υπάρχουν πολυάριθμες υλοποιήσεις της με διαφορετικά πλεονεκτήματα και μειονεκτήματα η καθεμία.**
- **Με τις σημερινές διαστάσεις που έχει πάρει η τεχνολογία των MSI, LSI και VLSI ολοκληρωμένων, ο σχεδιαστής δεν χρειάζεται να «κουραστεί» ιδιαίτερα για τα υποσυστήματα πρόσθεσης/αφαίρεσης. Μια μελέτη όμως της συμπεριφοράς τους και βελτιστοποίησης (κυρίως στην διάδοση του κρατουμένου), οδηγεί στην καλύτερη κατανόηση και σχεδίαση ολόκληρης της αριθμητικής του ολοκληρωμένου.**

Τελεστές και αποτελέσματα

- Η βασικότερη λοιπόν αριθμητική πράξη 2 δυαδικών αριθμών είναι η πρόσθεση. Η πράξη αυτή έχει τρεις τελεστές ,και δύο αποτελέσματα. Οι τρεις τελεστές είναι οι C_{in}, x, y και τα δύο αποτελέσματα είναι τα C_{out} και S.

x	y	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Υλοποιήσεις

- Η πιο απλή υλοποίηση που μπορεί να σκεφτεί κανείς, είναι η δημιουργία ενός κυκλώματος βασισμένου στον πίνακα αλήθειας της δυαδικής πρόσθεσης.
- Πίνακας αλήθειας ημιαθροιστή
  - S = x'y + xy'
  - C = xy

x/y	0	1
0	0	1
1	1	0

Πίνακας Αληθείας για το S

x/y	0	1
0	0	0
1	0	1

Πίνακας αληθείας για το C

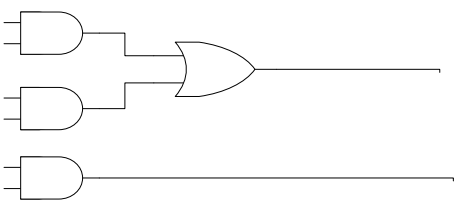
ΗΜΙΑΘΡΟΙΣΤΗΣ

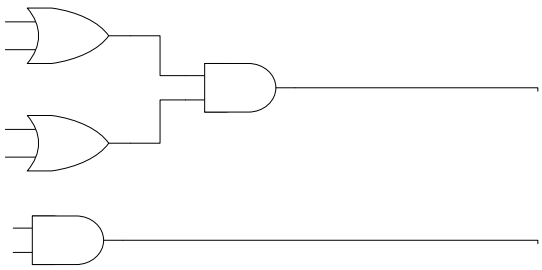
- Ο ημιαθροιστής χρειάζεται 2 δυαδικές εισόδους(προσθετέοι) και 2 μεταβλητές εξόδου(άθροισμα , κρατούμενο)
- Από τον πίνακα αλήθειας που δείξαμε συμπεραίνουμε ότι το κρατούμενο C είναι πάντα 0 εκτός αν και οι 2 είσοδοι είναι 1.Η έξοδος S είναι το λιγότερο σημαντικό bit.
- Από τον πίνακα αλήθειας μπορούμε να εξάγουμε και τις απλοποιημένες συναρτήσεις Boole των εξόδων και είναι :

$S = x'y + xy'$, $C = xy$

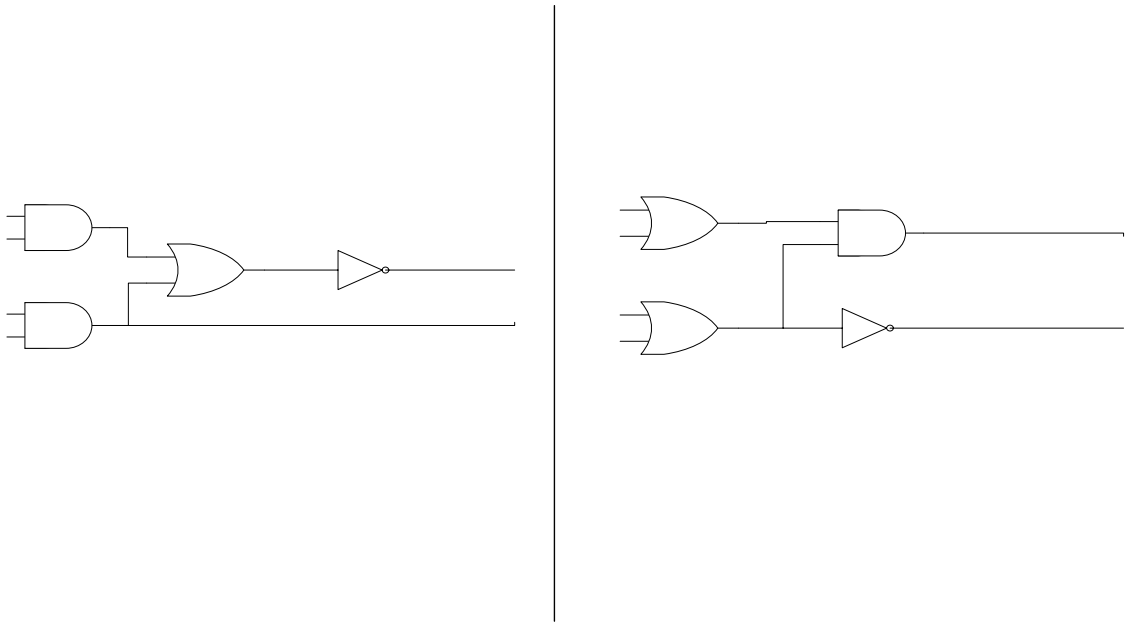
Έτσι έχουμε το ακόλουθο λογικό διάγραμμα ,αλλά και άλλες 4 υλοποιήσεις που επιτυγχάνουν το ίδιο αποτέλεσμα.

ΥΛΟΠΟΙΗΣΕΙΣ ΗΜΙΑΘΡΟΙΣΤΩΝ



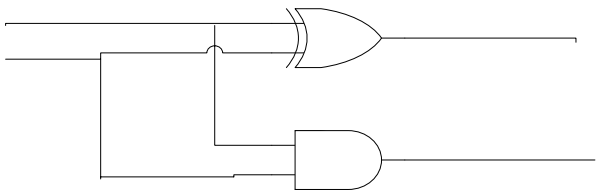


ΥΛΟΠΟΙΗΣΕΙΣ ΗΜΙΑΘΡΟΙΣΤΩΝ



ΥΛΟΠΟΙΗΣΕΙΣ ΗΜΙΑΘΡΟΙΣΤΩΝ

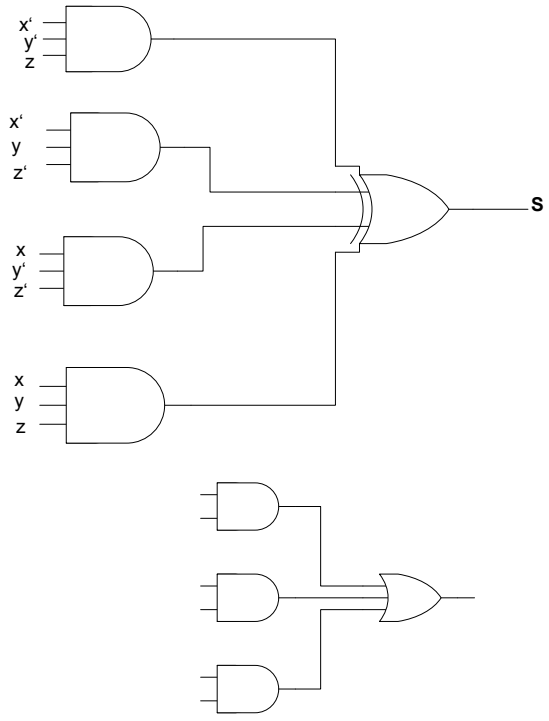
- Το πιο απλό όμως είναι ο ημιαθροιστής να υλοποιηθεί με μια XOR και μια AND όπως φαίνεται στο παρακάτω σχήμα. Αυτή χρησιμοποιείται για να δείξει ότι 2 ημιαθροιστικά κυκλώματα είναι απαραίτητα για την κατασκευή ενός πλήρη αθροιστή.



ΑΘΡΟΙΣΤΕΣ

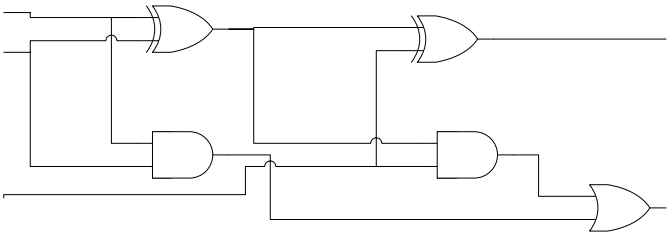
- Ο ημιαθροιστής όμως δεν ικανοποιεί την ανάγκη μας για πρόσθεση ή αφαίρεση περισσότερων ψηφίων.
- Όταν οι προσθετέοι περιέχουν και άλλα σημαντικά ψηφία ,το κρατούμενο που βγαίνει από την πρόσθεση 2 bits προστίθεται στο επόμενο μεγαλύτερης σημαντικότητας ζευγάρι bits.
- Το κύκλωμα που εκτελεί πρόσθεση 2 bits λέγεται ημιαθροιστής (half-adder).
- Το κύκλωμα που εκτελεί την πρόσθεση 3 bits(2 σημαντικών & ενός κρατούμενου) λέγεται πλήρης αθροιστής(full-adder). Είναι αυτός που ουσιαστικά χρησιμοποιείται στην πρόσθεση πολλών ψηφίων. Το όνομα του προέρχεται από το γεγονός ότι 2 ημιαθροιστές μπορούν να χρησιμοποιηθούν για να υλοποιήσουν ένα πλήρη αθροιστή.

Πλήρης αθροιστής



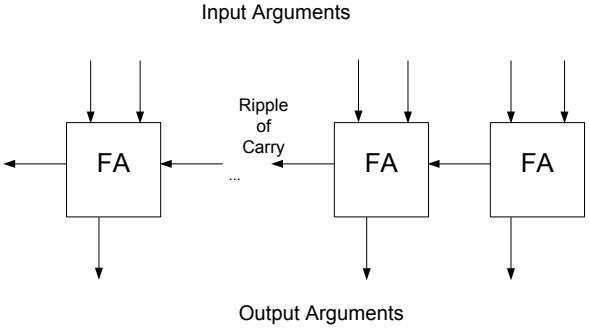
x	y	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Πλήρης αθροιστής



Παράλληλος Αθροιστής Ν ψηφίων

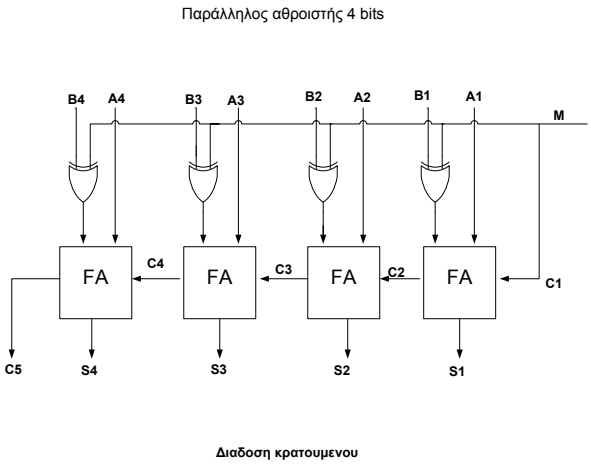
- Χρησιμοποιώντας τώρα το προηγούμενο κύκλωμα επαναληπτικά μπορούμε να δημιουργήσουμε έναν αθροιστή/αφαιρέτη Ν bits με το κρατούμενο να διαδίδεται στην επόμενη βαθμίδα. Αυτός λέγεται παράλληλος αθροιστής διάδοσης κρατουμένου. Υπάρχουν διάφορες υλοποιήσεις αυτού του είδους αθροιστή ανάλογα πάντα με την αριθμητική που χρησιμοποιούμε.
- **Σημείωση: Πριν συνεχίσουμε την παρουσίαση αναφέρουμε ότι ουσιαστικά η αφαίρεση και η πρόσθεση γίνονται στο ίδιο κύκλωμα. Διότι και η αφαίρεση δύο αριθμών μπορεί να αναλυθεί ως εξής: A – B = A + (-B). Έτσι το μόνο που έχουμε να κάνουμε είναι να μετατρέψουμε τον αφαιρέτη στον αντίθετο του. Αυτό επιτυγχάνεται πολύ γρήγορα με την χρήση της XOR η οποία έχει την ιδιότητα όταν ο ένας της τελεστής είναι 1 να αντιστρέφει τον άλλο και όταν ο ένας είναι 0 να αφήνει τον άλλο ανεπηρέαστο. Αυτό βρίσκει εφαρμογή στην μετατροπή ενός αριθμού στον αντίθετο του επειδή οι περισσότερες αριθμητικές βασίζονται στην αντιστροφή των bit για την αναπαράσταση θετικών και αρνητικών.**



Υλο

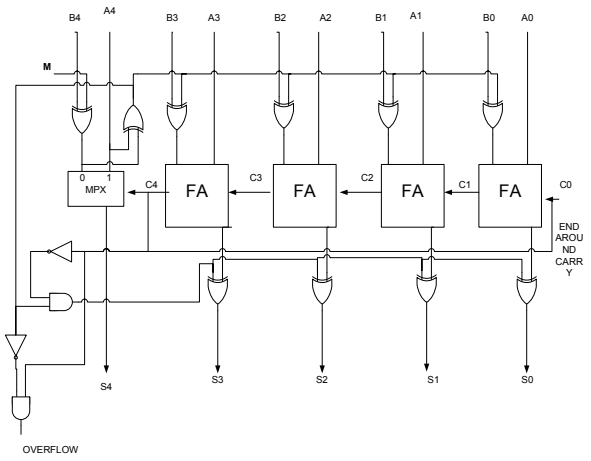
Παράλληλος αθροιστής για αριθμητική συμπλήρωμα ως προς 2

- Ο εν λόγω αθροιστής χρησιμοποιεί το πιο διαδεδομένο είδος αριθμητικής. Το συμπλήρωμα ως προς δύο για αριθμούς N bits έχει εύρος $[-2^{N-1}, 2^{N-1}-1]$ και χρησιμοποιεί το πιο σημαντικό ψηφίο (N-1) για ένδειξη πρόσημου. Το λογικό του διάγραμμα φαίνεται στην διπλανή εικόνα. Το bit M χρησιμοποιείται για την δήλωση της πράξης (M=0 πρόσθεση και M=1 αφαίρεση).



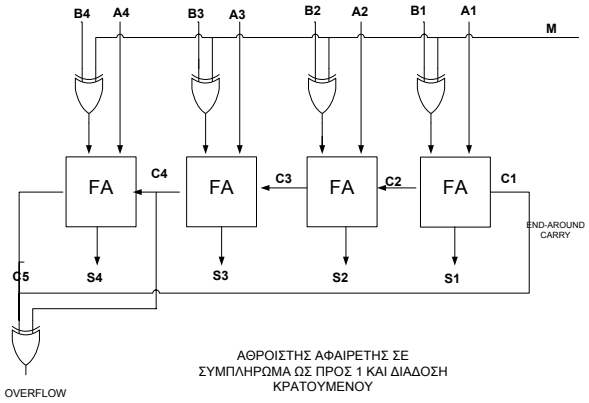
Παράλληλος αθροιστής για αριθμητική πρόσημου-μέτρου

- **Το συγκεκριμένο είδος αριθμητικής είναι το πιο απλό στην σύλληψη του αλλά χάνουμε σε εύρος αριθμών, λόγω του επιπλέον bit που μπαίνει για την ένδειξη του πρόσημου. Το εν λόγω bit δεν παίζει ρόλο στις πράξεις ή στην εσωτερική αναπαράσταση του αριθμού αλλά μόνο στο πρόσημο του. Έτσι το εύρος των αριθμών εδώ είναι $[-(2^{N-1}-1), 2^{N-1}-1]$. Παρατηρούμε ότι χάνουμε έναν αριθμό στους αρνητικούς και αυτό συμβαίνει γιατί έχουμε αναγκαστικά δύο αναπαραστάσεις του 0 (0...00 και 1...00).**



Παράλληλος αθροιστής για αριθμητική συμπλήρωμα ως προς 1

- Η εν λόγω αριθμητική δεν λύνει το πρόβλημα της προηγούμενης αριθμητικής αλλά απλοποιεί κατά πολύ την πολυπλοκότητα του κυκλώματος της πρόσθεσης/αφαίρεσης του πρόσημου μέτρου. Λόγω της αφαίρεσης αυτής της πολυπλοκότητας επιτυγχάνεται ταυτόχρονα και αύξηση της επίδοσης. Εδώ το πρόσημο ενός αριθμού φαίνεται από το πιο σημαντικό bit του αλλά εάν είναι αρνητικός τότε πρέπει να αντιστρέψουμε τα bit του για να βρούμε τον αριθμό που αναπαριστά. Το κυκλωματικό διάγραμμα του εν λόγω προσθέτη/αφαιρέτη φαίνεται δίπλα. Η μόνη διαφορά με αυτόν του συμπληρώματος ως προς 2 είναι ότι προσθέτουμε το carry ξανά, επομένως παίρνουμε το αποτέλεσμα σε χρόνο διπλάσιο από τον χρόνο του συμπληρώματος ως προς δύο. Αυτό το κρατούμενο ονομάζεται end-around carry.



Χρονική ανάλυση του παράλληλου αθροιστή

- Αν θεωρήσουμε ότι κάθε carry μεταδίδεται στον επόμενο full adder σε χρόνο t_{rc} και ο πλήρης αθροιστής χρειάζεται t_d χρόνο για να υπολογίσει το S, τότε ο δεύτερος εν σειρά αθροιστής θα χρειαστεί $t_d + t_{rc}$ για να υπολογίσει το S_1 και $2t_{rc}$ για να υπολογίσει το C_1. Επομένως αν έχουμε N ψηφία, τότε τα αποτελέσματα θα ολοκληρωθούν σε χρόνο $t_d + (N-1)t_{rc}$ ενώ το τελευταίο carry θα εξαχθεί σε χρόνο Nt_{rc}. Αυτό είναι αρκετά «αργό» σε περιπτώσεις που θέλουμε γρήγορη εξαγωγή αποτελεσμάτων. Πολλές φορές δηλαδή αυτή η διάδοση του κρατουμένου μας τρώει αρκετό από τον διαθέσιμο κύκλο ρολογιού καθώς εξαρτάται γραμμικά από το μήκος του adder.

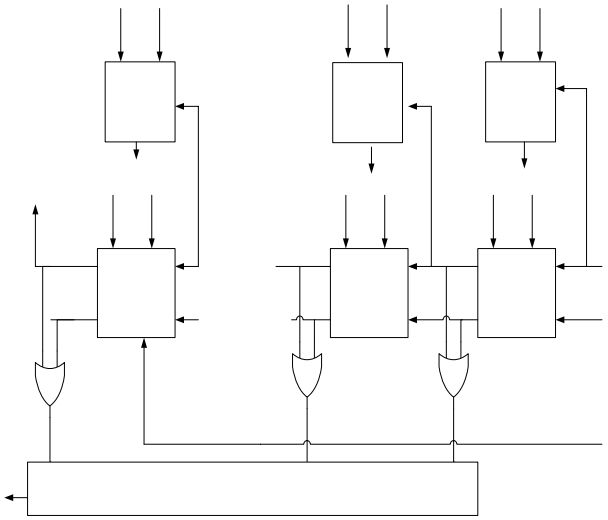
Βελτιστοποίηση παράλληλου αθροιστή

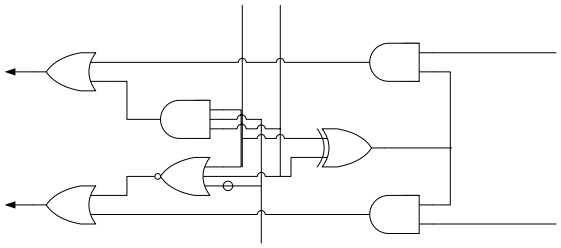
- Το μεγάλο εμπόδιο στην απόδοση του παράλληλου αθροιστή είναι η διάδοση του κρατουμένου στις επόμενες βαθμίδες. Υπάρχουν πολλές τεχνικές που μπορούμε να χρησιμοποιήσουμε προκειμένου να εξαλείψουμε αυτές τις καθυστερήσεις.
- Τεχνικές επιτάχυνσης παράλληλου αθροιστή
  - Χρησιμοποίηση Carry-Completion Sensing Adder
  - Χρησιμοποίηση Conditional-Sum Adder
  - Χρησιμοποίηση Carry-Select Adder
  - Χρησιμοποίηση Carry-Lookahead Adder

Carry-Completion Sensing Adder

- Ο εν λόγω αθροιστής βασίζεται σε μια πάρα πολύ σημαντική παρατήρηση όσον αφορά την πρόσθεση.
- Κρατούμενο διαδίδεται στην επόμενη βαθμίδα μόνο και μόνο εάν και τα δύο ψηφία της παρούσας βαθμίδας είναι 1. Κρατούμενο δεν διαδίδεται σε καμία περίπτωση μόνο και μόνο εάν και τα δύο ψηφία της παρούσας βαθμίδας είναι 0. Στην κλασσική περίπτωση του full-adder το κρατούμενο C_i υπολογίζεται ως $C_i = (A_i \text{ XOR } B_i)C_{i-1} + A_i B_i$. Παρατηρούμε δηλαδή ότι εξαρτάται από το προηγούμενο κρατούμενο. Όμως με βάση τα παραπάνω υπάρχει περίπτωση να ολοκληρωθεί η διαδικασία δημιουργίας του κρατουμένου πριν έρθει το προηγούμενο. Χρειαζόμαστε λοιπόν μία βαθμίδα ανίχνευσης ολοκλήρωσης των κρατουμένων. Αυτή θα βγάζει 1 όταν θα έχει ολοκληρωθεί η δημιουργία όλων των σωστών κρατουμένων.
- Με αυτή τη τεχνική ο χρόνος υπολογισμού της πράξης εξαρτάται άμεσα από τα δεδομένα αφού, η μονάδα ανίχνευσης ολοκληρώνει σε διαφορετικό χρόνο κάθε φορά ανάλογα με τα δεδομένα. Έτσι ο αθροιστής αυτός είναι αυτοσυγχρονιζόμενος.
- Οι εξισώσεις που έχουμε για την «οπωσδήποτε» διάδοση κρατουμένου ή «οπωσδήποτε» μη διάδοση κρατουμένου είναι οι $C_i = A_iB_iH_i + C_{i-1}(A_i \text{ XOR } B_i)$ και $D_i = A'_iB'_iH_i + D_{i-1}(A_i \text{ XOR } B_i)$, όπου D_i είναι ένδειξη ότι δεν μεταδίδεται κρατούμενο. Έτσι η μονάδα ανίχνευσης θα αποτελείται από μια AND όλων των D_i OR C_i.

Λογικό Διάγραμμα CCSA





Κελί CP (Carry Propagation)

An-1 Bn-1 A1 B

Χρονική Ανάλυση του CCSA

FA

- Ο εν λόγω αθροιστής ολοκληρώνει σε χρόνο <=Δ, όπου Δ ο χρόνος ολοκλήρωσης της πράξης του απλού παράλληλου αθροιστή. Στην ουσία γλιτώνουμε χρόνο μερικές φορές ανάλογα με τα δεδομένα και κυρίως όταν έχουν πολλά ίδια ψηφία γιατί τότε μπορούμε να ανιχνεύσουμε αμέσως την ύπαρξη ή μη του κρατουμένου.
- Πρωτότυπο αυτού του αθροιστή πρωτοδημιουργήθηκε από τον Gilchrist και τα πειραματικά αποτελέσματα του, έδειξαν μια βελτίωση 8 προς 1 για 40-bit δεδομένα, σε σχέση με τον απλό παράλληλο αθροιστή διάδοσης κρατουμένου.

A1 B

CP

Conditional-Sum Addition

- Η εν λόγω τεχνική χρησιμοποιεί δύο εξισώσεις, μία για την περίπτωση που υπάρχει κρατούμενο και μία για την περίπτωση που δεν υπάρχει (από την προηγούμενη βαθμίδα). Έτσι έχουμε τις παρακάτω εξισώσεις

  $S_i^0 = A_i$ XOR B_i

  $C_i^0 = A_i B_i$, για $C_{i-1} = 0$

  $S_i^1 = A_i$ XNOR B_i

  $C_i^1 = A_i + B_i$, για $C_{i-1} = 1$

  Οι παραπάνω εξισώσεις παράγονται αν αντικαταστήσουμε στις εξισώσεις του full adder το αντίστοιχο C_{i-1}. Εν συνεχεία μπορούμε να επιλέξουμε με πολυπλέκτες, ανάλογα με το κρατούμενο της προηγούμενης βαθμίδας ποιο αποτέλεσμα θα κρατήσουμε στην έξοδο. Ο αριθμός των επιπέδων που χρειαζόμαστε ασφαλώς εξαρτάται άμεσα από τον αριθμό των ψηφίων n των δεδομένων. Έτσι χρειαζόμαστε t = ceil($\log_2$n) επίπεδα κάθε ένα από τα οποία περιέχει πολυπλέκτες ολοένα αυξανόμενων εξόδων.

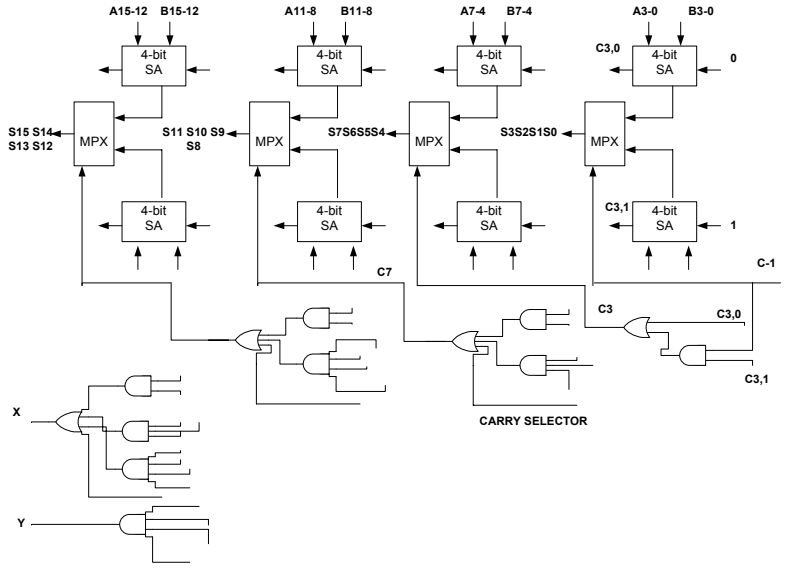
Τι κερδίζουμε και τι χάνουμε με τον Conditional-Sum Adder

- **Πλεονεκτήματα**
  - Παραγωγή όλων των δυνατών περιπτώσεων
  - Παραγωγή σε κάθε επίπεδο όλων των carry
  - Γρήγορη απόκριση εξόδου
- **Μειονεκτήματα**
  - Πολύπλοκη λογική
  - Καταλαμβάνει μεγάλο όγκο στο ολοκληρωμένο
  - Ο αριθμός των επιπέδων αυξάνεται λογαριθμικά σε σχέση με το μέγεθος των δεδομένων

Carry-Select Adder

- Ο προηγούμενος αθροιστής, είχε αρκετά μειονεκτήματα με κυριότερο από όλα τον μεγάλο του όγκο.
- Μια μικρή παραλλαγή του οδηγεί στον CSA adder, ο οποίος είναι καταλαμβάνει πολύ μικρότερο χώρο. Και εδώ επικρατεί η λογική ότι υπολογίζουμε a priori τα αποτελέσματα σε κάθε βαθμίδα, είτε έχουμε είτε δεν έχουμε κρατούμενο. Στην συνέχεια ένας μόνο πολυπλέκτης επιλέγει την επιθυμητή έξοδο ανάλογα με το προηγούμενο κρατούμενο. Το προηγούμενο κρατούμενο δημιουργείται κατευθείαν από την προηγούμενη βαθμίδα με μία λογική που μοιάζει αρκετά στην πρόβλεψη κρατουμένου.
- Επίσης ο αθροιστής χωρίζεται σε N κομμάτια όπου N = ceil(n/4). Δηλαδή αποτελείται από δυάδες αθροιστών 4 ψηφίων. Σε κάθε δυάδα ο ένας αθροιστής κάνει υπολογισμούς σαν να υπάρχει το carry, ενώ στην άλλη σαν να μην υπάρχει. Στο σχήμα φαίνεται πιο καλά η τεχνική αυτή.

Λογικό διάγραμμα CSA



Carry-Lookahead Adder

- Ο παράλληλος αθροιστής με την διάδοση κρατουμένου μπορεί να βελτιωθεί πολύ με μια τεχνική που λέγεται πρόβλεψη κρατουμένου (carry lookahead adder).
- Η συγκεκριμένη τεχνική βασίζεται στο γεγονός ότι όλα τα κρατούμενα που εισάγονται σε όλες τις θέσεις του παράλληλου αθροιστή, υπολογίζονται ταυτόχρονα με επιπρόσθετη λογική. Αυτό συντελεί σε ένα χρόνο απόκρισης σταθερό και ανεξάρτητο του μήκους του αθροιστή (δηλαδή του word).
- Εξαγωγή συναρτήσεων
  - Χρησιμοποιούμε βοηθητικές συναρτήσεις σε κάθε στάδιο και παίρνουμε το «δημιουργούμενο κρατούμενο» G_i και το «διαδιδόμενο κρατούμενο» P_i. Δηλαδή $G_i = A_i B_i$ και $P_i = A_i$ XOR B_i.

    Γνωρίζουμε ότι $S_i = (A_i \text{ XOR } B_i) \text{ XOR } C_{i-1} = P_i \text{ XOR } C_{i-1}$

    $C_i = A_i B_i + (A_i \text{ XOR } B_i) C_{i-1} = G_i + P_i C_{i-1}$
  - Οι παραπάνω εξισώσεις δείχνουν ότι μπορούν να εφαρμοστούν αναδρομικά και να λύσουμε το σύνολο των αποτελεσμάτων κατευθείαν από τους τελεστές A και B μαζί, φυσικά, και με το αρχικό κρατούμενο Cin.

Εξισώσεις CLA

- Οι εξισώσεις για 4 ψηφία είναι οι εξής:

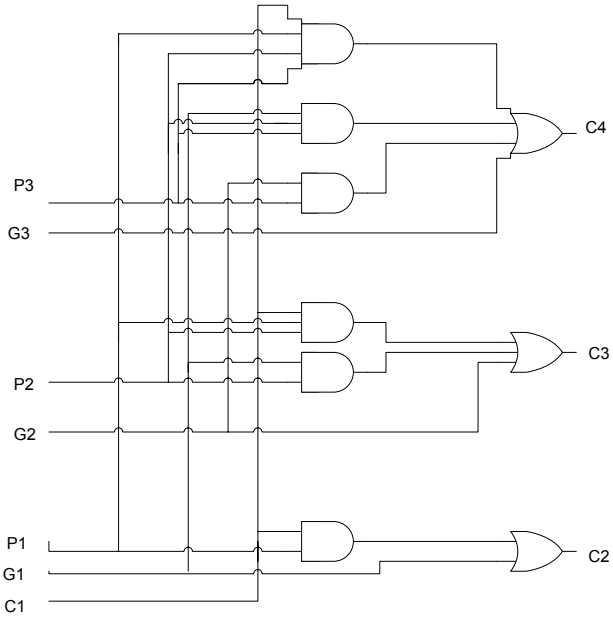
  $C_0 = G_0 + C_{in} P_0$

  $C_1 = G_1 + G_0 P_1 + C_{in} P_0 P_1$

  $C_2 = G_2 + G_1 P_2 + G_0 P_1 P_2 + C_{in} P_0 P_1 P_2$

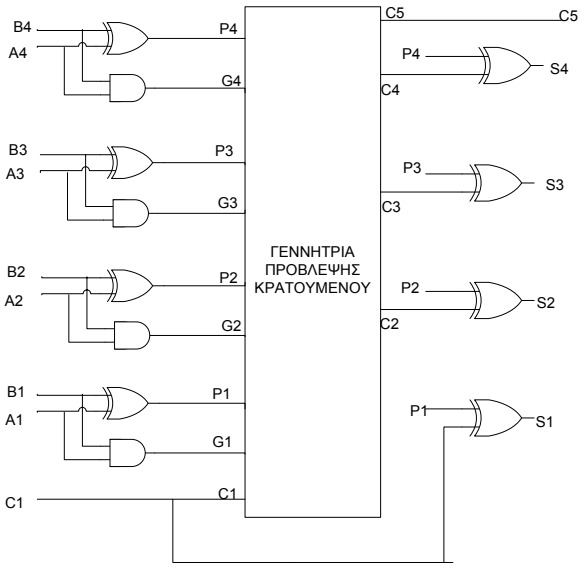
  $C_3 = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + C_{in} P_0 P_1 P_2 P_3$
  - Παρατηρούμε ότι για περισσότερα ψηφία αυξάνεται πολύ το πλήθος των πυλών που πρέπει να χρησιμοποιηθούν για να υπολογιστούν τα επιμέρους κρατούμενα. Με την αύξηση κάθε bit, αυξάνονται κατά μία οι πύλες or και κάθε προηγούμενη πύλη and αυξάνει κατά μία τις εισόδους της. Η αύξηση στην πολυπλοκότητα είναι απαγορευτική για την χρησιμοποίηση σε κυκλώματα πολλών bit. Ωστόσο όταν θέλουμε να εκμεταλλευτούμε τα πλεονεκτήματα που προσφέρει ο CLA, μπορούμε να χρησιμοποιήσουμε μια υβριδική τεχνική στην οποία χρησιμοποιούμε blocks 4-bit CLA's και αυτά μεταξύ τους μεταδίδουν το κρατούμενο.

Λογικό διάγραμμα πρόβλεψης κρατουμένου 4-bit.



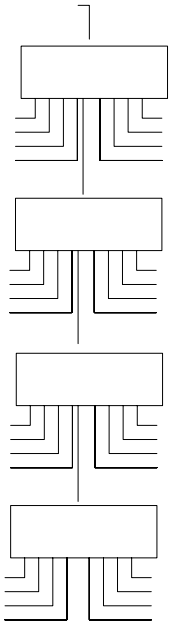
ΛΟΓΙΚΟ ΔΙΑΓΡΑΜΜΑ ΜΙΑΣ ΓΕΝΝΗΤΡΙΑΣ ΠΡΟΒΛΕΨΗΣ ΚΡΑΤΟΥΜΕΝΟΥ(CARRY LOOK-AHEAD)

Λογικό διάγραμμα ολοκληρωμένου CLA 4-bit



Σύνδεση 4 CLA's για την δημιουργία ενός 16-bit CLA

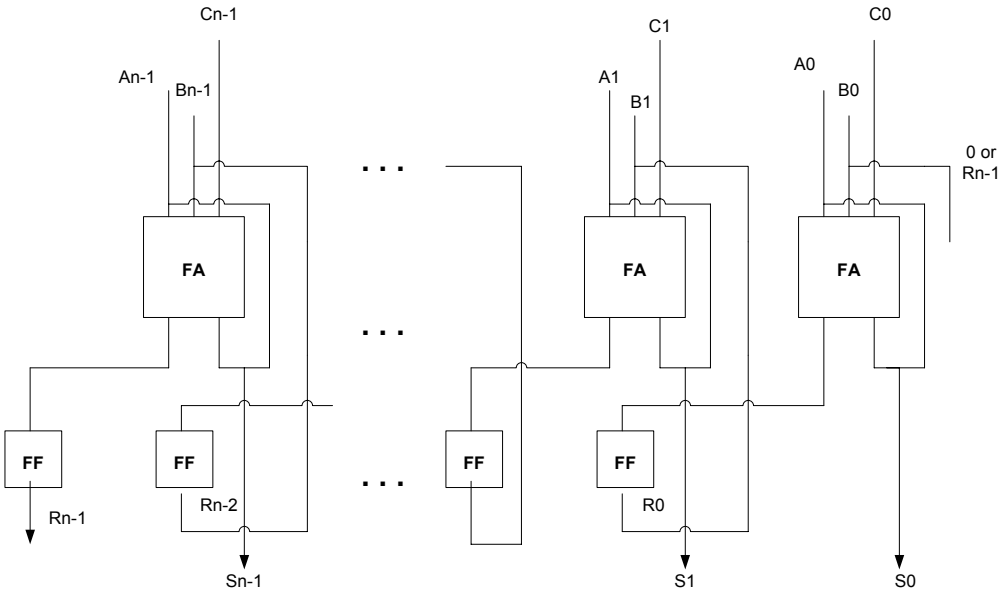
- Δίπλα δείχνουμε πως μπορούμε να συνδυάσουμε την τεχνική του carry-ripple μαζί με το carry-lookahead προκειμένου να μειώσουμε τον χρόνο απόκρισης σε συστήματα με πολλά bits. Στο εν λόγω σύστημα αν το carry ενός CLA κάνει δ χρόνο για να δημιουργηθεί, τότε σε έναν CLA N bits (όπου N = 4κ, κ ακέραιος), τότε το τελικό κρατούμενο κάνει χρόνο κδ για να δημιουργηθεί. Το δ ισούται με t_{5or} + t_{5and}, όπου t_{5or} η καθυστέρηση διάδοσης μιας πύλης or 5 εισόδων και t_{5and} η καθυστέρηση διάδοσης μιας πύλης and 5 εισόδων, σε αντίθεση με τον απλό παράλληλο αθροιστή όπου η καθυστέρηση διάδοσης είναι ανάλογη του αριθμού των bit N.



Αθροιστές πολλών τελεστών

- Μέχρι τώρα εξετάσαμε αθροιστές δύο τελεστών. Υπάρχει μια ειδική κατηγορία αθροιστών η οποία καταφέρνει να κάνει την πράξη της πρόσθεσης ή αφαίρεσης σε περισσότερους από δύο τελεστές.
- Είναι πολύ σημαντικοί διότι χρησιμοποιούνται κατά κόρον σε πράξεις όπως ο πολλαπλασιασμός ή η διαίρεση. Σε αυτές τις πράξεις πρέπει να είναι δυνατό το άθροισμα πολλών αριθμών, για την ακρίβεια τόσων όσο είναι και το πλήθος των bit των πολλαπλασιαστέων.
- Θα παρουσιάσουμε έναν μόνο τέτοιον αθροιστή τον Carry-Save Adder
  - Η υλοποίηση του είναι σχετικά απλή και βασίζεται στην προσωρινή αποθήκευση του κρατουμένου σε ένα D flip-flop, ώστε αργότερα να το επανατροφοδοτήσουμε στην επόμενη βαθμίδα για την επόμενη πρόσθεση.
  - **Βήματα**
    1. **Φορτώνουμε τους τρεις πρώτους αριθμούς στις εισόδους A, B, C.**
    2. **Μετά από ένα κύκλο πρόσθεσης αποκτούμε την έξοδο R_i, που στην ουσία είναι το κρατούμενο της προηγούμενης βαθμίδας καθυστερημένο κατά μία χρονική μονάδα. Φορτώνουμε, εν συνεχεία, τα S και R στις εισόδους A και B αντίστοιχα.**
    3. **Επαναλαμβάνουμε το βήμα 2 για τους υπόλοιπους k-4 αριθμούς, έτσι ώστε όλοι οι k αριθμοί να μπουν στον αθροιστή. Το παραπάνω βήμα θέλει k-2 κύκλους πρόσθεσης για να ολοκληρωθεί**
    4. **Επαναλαμβάνουμε το βήμα 3 μέχρι μηδενικά να γεμίσουν όλα τα R_i bits. Τότε μπορεί να χρειαστούμε μέχρι και n-1 κύκλους για να ολοκληρωθεί ολόκληρη η διαδικασία της διάδοσης κρατουμένου και να πάρουμε εν τέλει το αποτέλεσμα της πρόσθεσης.**

Λογικό διάγραμμα CSA



Ειδικοί αθροιστές

- Σε αυτή την ενότητα θα παρουσιάσουμε δύο ειδικούς αθροιστές οι οποίοι χρησιμοποιούνται σε μεμονωμένες περιπτώσεις όπου το απαιτεί η εκάστοτε εφαρμογή.
  1. Αθροιστής BCD. Χρησιμοποιείται κυρίως σε διατάξεις 7-segment LED, όπου θέλουμε αναπαράσταση σε BCD.
  2. Σειριακός αθροιστής. Είναι ένας σχετικά αργός αθροιστής, αφού παίρνει η κύκλους ρολογιού για να πάρουμε το αποτέλεσμα αλλά χρησιμοποιείται όπου έχουμε αυστηρές απαιτήσεις σε χώρο και χαλαρές σε ταχύτητα.

Αθροιστής BCD

- Οι υπολογιστές που εκτελούν αριθμητικές λειτουργίες κατευθείαν στο δεκαδικό αριθμητικό σύστημα συμβολίζουν τους αριθμούς με κάποιο δυαδικό κώδικα .Ένας αθροιστής για έναν τέτοιο υπολογιστή πρέπει να χρησιμοποιεί αριθμητικά κυκλώματα που δέχονται κωδικοποιημένους δεκαδικούς αριθμούς και παρουσιάζουν τα αποτελέσματα στον κατάλληλο κώδικα.
- Αν θεωρήσουμε την αριθμητική πρόσθεση 2 δεκαδικών ψηφίων σε BCD,μαζί με ένα πιθανό κρατούμενο από προηγούμενη βαθμίδα ,τότε επειδή κάθε ψηφίο εισόδου δεν ξεπερνάει το 9,το άθροισμα εξόδου δεν μπορεί να είναι μεγαλύτερο από 9+9+1=19. συνεπώς αν τροφοδοτήσουμε 2 ψηφία BCD σε ένα δυαδικό αθροιστή 4 bits,ο αθροιστής θα σχηματίσει το άθροισμα δυαδικά και θα παράγει ένα αποτέλεσμα που μπορεί να κυμαίνεται από 0 μέχρι 19.

Αθροιστής BCD, πίνακας

K	Z8	Z4	Z2	Z1	C	S8	S4	S2	S1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Αθροιστής BCD

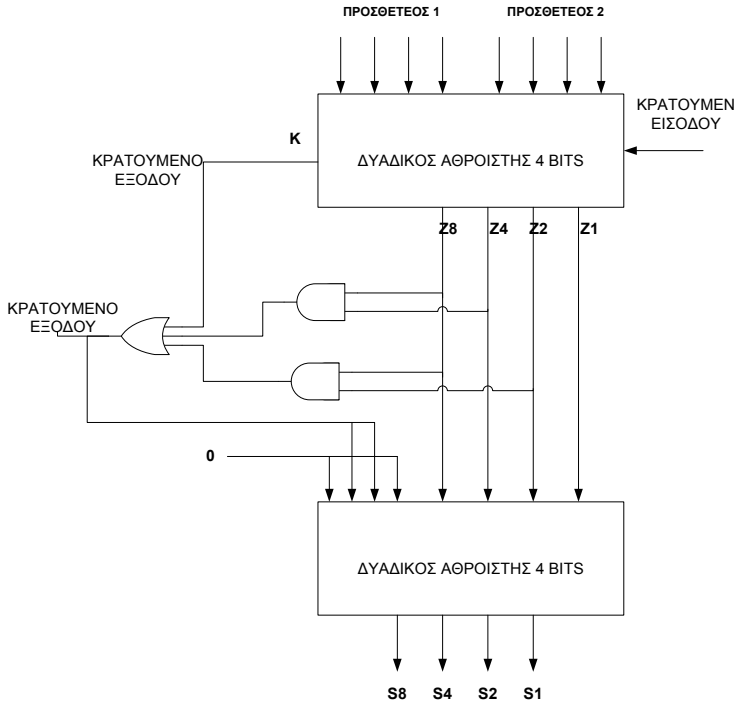
- Στον πίνακα βλέπουμε ότι το K είναι το κρατούμενο και οι δείκτες του γράμματος Z συμβολίζουν τα βάρη 8,4,2,1 που έχουν τα 4 bits του κώδικα BCD. Η 1η στήλη στον πίνακα δείχνει τα δυαδικά αθροίσματα ,όπως εμφανίζονται στις εξόδους ενός δυαδικού αθροιστή 4 bits. Το άθροισμα εξόδου 2 δεκαδικών ψηφίων ,όμως ,πρέπει να παρασταθεί σε κώδικα BCD και να εμφανιστεί στη μορφή που φαίνεται στην 2η στήλη . Το κύριο πρόβλημα μας είναι η εύρεση ενός κανόνα με τον οποίο ο δυαδικός αριθμός στην 1η στήλη να μπορεί να μετατραπεί στην σωστή παράσταση ψηφίων BCD της 2ης στήλης.
- Μελετώντας τον πίνακα συμπεραίνουμε ότι όταν το δυαδικό άθροισμα είναι μικρότερο ή ίσο του 1001 ο αντίστοιχος αριθμός BCD είναι ο ίδιος . Όταν όμως το δυαδικό άθροισμα είναι μεγαλύτερο από 1001,τοτε η έξοδος αυτή δεν είναι έγκυρος κώδικας BCD. Για να μετατραπεί στην σωστή έξοδο αρκεί να προσθέσουμε το 6(0110) στο δυαδικό άθροισμα ,και επίσης παράγεται και ένα κρατούμενο εξόδου όπως ακριβώς απαιτείται.

Αθροιστής BCD,κανόνας διόρθωσης

- Το λογικό κύκλωμα που θα ανιχνεύσει κατά πόσο χρειάζεται διόρθωση ή όχι ,μπορεί να προκύψει από τον πίνακα .φαίνεται ότι χρειάζεται διόρθωση τουλάχιστον όταν το δυαδικό άθροισμα έχει κρατούμενο εξόδου K=1.οι άλλοι 6 συνδυασμοί ,από 1010 μέχρι 1111 , που χρειάζονται διόρθωση έχουν Z8=1.για να τους ξεχωρίσουμε από τους 1000 και 1001 που έχουν επίσης Z8=1,καθοριζουμε επιπλέον ότι είτε το Z4 είτε το Z2 πρέπει να είναι 1. Συνεπώς από τα παραπάνω παίρνουμε την ακόλουθη συνθήκη:

  C =K + Z8 Z4 + Z8 Z2
- Όταν C = 1 ,είναι απαραίτητο να προσθέσουμε 0110 στο δυαδικό άθροισμα και να δώσουμε κρατούμενο εξόδου για την επόμενη βαθμίδα.
- Ένας αθροιστής BCD πρέπει να περιλαμβάνει τα κυκλώματα για την παραπάνω διόρθωση στην εσωτερική του κατασκευή. Για να προσθέσουμε το 0110 στο δυαδικό άθροισμα ,χρησιμοποιούμε ένα 2ο δυαδικό αθροιστή 4 bits. Τα 2 δεκαδικά ψηφία προστίθενται ,μαζί με το κρατούμενο εισόδου, πρώτα με τον επάνω δυαδικό αθροιστή και έτσι παίρνουμε το δυαδικό άθροισμα. Όταν το κρατούμενο εξόδου ισούται με 0 ,δεν προστίθεται τίποτα στο δυαδικό άθροισμα. Όταν είναι 1 ,προστίθεται ο δυαδικός 0110 στο δυαδικό άθροισμα μέσω του κάτω αθροιστή.

Αθροιστής BCD, σχήμα



Σειριακός αθροιστής ενός bit

- Εκτός από τους παράλληλους αθροιστές υπάρχουν και οι σειριακοί .ένας σειριακός αθροιστής χρησιμοποιεί έναν απλό αθροιστή και συναρμολογεί το άθροισμα SUM ακολουθιακά. Στο χρόνο t,υπολογίζεται το sum και αποθηκεύεται το carry σε έναν καταχωρητή .στο χρόνο t+1 , το άθροισμα χρησιμοποιεί το carry(t) για να υπολογίσει το καινούργιο sum.
- Οι 2 είσοδοι του αθροιστή αποθηκεύονται σε καταχωρητές των n-bits , η έξοδος sum αποθηκεύεται σε έναν καταχωρητή αποτελέσματος n-bits. Η πρόσθεση αρχίζει με τον καθαρισμό του καταχωρητή κρατουμένου ,μετά οι τελεστές εφαρμόζονται σειριακά στις εισόδους του αθροιστή ,με πρώτο το λιγότερο σημαντικό ψηφίο. Το παράδειγμα του σχήματος δείχνει την πρόσθεση των αριθμών 1 και 5 με αποτέλεσμα το 6.
- Ακόμα χρειάζονται n κύκλοι για να ολοκληρωθεί πρόσθεση n ψηφίων.
- Σε έναν σειριακό αθροιστή, συμφέρει να έχουμε ίδιες καθυστερήσεις για τα sum & carry,διότι οι καθυστερήσεις αυτές ορίζουν την μεγαλύτερη συχνότητα ρολογιού στην οποία μπορεί να λειτουργήσει ο αθροιστής.

Σειριακός αθροιστής ενός bit,σχήμα