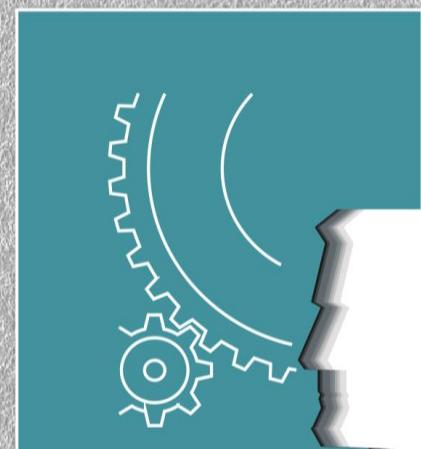


ΣΗΜΕΙΩΣΕΙΣ ΠΡΟΗΓΜΕΝΩΝ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ

Computer – Ανάλυση



ανάλιση
COMPUTER

ΕΡΓΑΣΤΗΡΙΟ ΕΛΕΥΘΕΡΩΝ ΣΠΟΥΔΩΝ
ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

...λυση
σπουδων

ΑΓ. ΑΝΔΡΕΟΥ 130-132 & ΓΟΥΝΑΡΗ, Τ.Κ. 26222, ΠΑΤΡΑ
ΤΗΛ: 310-097, 324-900 - ΤΗΛ/FAX: 313-547
E-MAIL: janelisj@otenet.gr, janelisk@otenet.gr

Περιεχόμενα

1	Κεφάλαιο - Η Φύση του Υλικού και του Λογισμικού	7
	Εισαγωγή στη Συσχεδίαση Υλικού/Λογισμικού	7
1.1.1	Υλικό	7
1.1.2	Λογισμικό	9
1.1.3	Υλικό και Λογισμικό	10
1.1.4	Ορίζοντας τη Συσχεδίαση Υλικού/Λογισμικού	11
	Η Αναζήτηση για Ενεργειακή Αποδοτικότητα	12
1.1.5	Απόδοση.....	12
1.1.6	Ενεργειακή Αποδοτικότητα	13
	Οι παράγοντες οδήγησης στη Συσχέτιση Υλικού / Λογισμικού	14
	Ο Χώρος Συσχεδίασης Υλικού – Λογισμικού	15
1.1.7	Ο Χώρος Σχεδιασμού Πλατφόρμας	16
1.1.8	Απεικόνιση εφαρμογών.....	17
	Ο Δυϊσμός του Σχεδιασμού Υλικού και του Σχεδιασμού Λογισμικού	18
	Μοντελοποίηση Επιπέδου Αφαίρεσης.....	20
2	Κεφάλαιο - Μοντελοποίηση και Μετασχηματισμός Ροής Δεδομένων	21
	Εισαγωγή γράφων ροής δεδομένων	21
2.1.1	Σύμβολα, Δρώντες και Ουρές	22
2.1.2	Ρυθμοί Πυροδότησης, Κανόνες Πυροδότησης και Χρονοδιαγράμματα	25
2.1.3	Σύγχρονοι Γράφοι Ροής Δεδομένων (synchronous data flow –SDF).....	25
2.1.4	Οι γράφοι SDF είναι καθορισμένοι (Determinate).....	26
	Ανάλυση Σύγχρονων Γράφων Ροής Δεδομένων.....	27
2.1.5	Εξαγωγή Περιοδικά Αποδεκτών Ακολουθιακών Χρονοδιαγραμμάτων.....	28
	Προσθέτοντας χρόνο και πόρους	31
2.1.6	Περιορισμοί πραγματικού χρόνου και Ρυθμός Δειγματοληψίας Εισόδου / Εξόδου	31
2.1.7	Μονέλο Ροής Δεδομένων Πόρων	31
	Μετασχηματισμοί	33
2.1.8	Επέκταση Πολλαπλών Ρυθμών.....	34
2.1.9	Επαναχρονισμός	35
2.1.10	Διοχέτευση (Pipelining).....	36
2.1.11	Άπλωμα (Unfolding).....	37
	Προβλήματα.....	38
3	Κεφάλαιο - Υλοποίηση Ροής Δεδομένων σε Λογισμικό και Υλικό	41

Υλοποίηση Ροής Δεδομένων σε Λογισμικό	41
3.1.1 Μετατροπή Ουρών και Δρώντων σε Λογισμικό	41
Υλοποίηση Ροής Δεδομένων σε Υλικό	44
3.1.2 Γράφοι SDF μονού-ρυθμού σε Υλικό	44
3.1.3 Διοχέτευση	46
Υλοποίηση Υλικού / Λογισμικού της Ροής Δεδομένων	48
Σύνοψη	49
Προβλήματα.....	50
4 Κεφάλαιο Ανάλυση Ροής Ελέγχου και Ροής Δεδομένων	51
Ακμές Δεδομένων και Ελέγχου ενός Προγράμματος C	51
Υλοποίηση Ακμών Δεδομένων και Ελέγχου	53
Κατασκευή του Γράφου Ροής Ελέγχου	54
Κατασκευή του Γράφου Ροής Δεδομένων.....	56
Εφαρμογή: Μεταφράζοντας τη C σε Υλικό.....	60
4.1.1 Σχεδιασμός της Διαδρομής Δεδομένων	60
Προβλήματα.....	64
5 Κεφάλαιο - Μηχανή Πεπερασμένων Καταστάσεων με Διαδρομή Δεδομένων	66
Υλικό Παράλληλων Bit Βασισμένο σε Κύκλους	66
5.1.1 Απεικόνιση Εκφράσεων σε Υλικό	66
Μηχανές Πεπερασμένων Καταστάσεων με Διαδρομή Δεδομένων.....	66
Σύνοψη	68
6 Κεφάλαιο - Μικροπρογραμματιζόμενες Αρχιτεκτονικές	69
Περιορισμοί Μηχανών Πεπερασμένων Καταστάσεων	69
6.1.1 Έκρηξη Καταστάσεων.....	69
6.1.2 Διαχείριση Εξαιρέσεων (Exception Handling)	70
6.1.3 Ευελιξία χρόνου εκτέλεσης.....	71
Μικροπρογραμματιζόμενος Έλεγχος.....	71
6.1.4 Πεδίο Άλματος	73
6.1.5 Πεδίο Εντολής	74
Η Μικροπρογραμματιζόμενη Διαδρομή Δεδομένων	76
6.1.6 Αρχιτεκτονική Διαδρομής Δεδομένων.....	76
6.1.7 Αρχιτεκτονική Διαδρομής Δεδομένων.....	78
Διοχέτευση Μικροπρογράμματος	80
6.1.8 Καταχωρητής Μικρο-εντολής.....	81

6.1.9	Καταχωρητής Κωδικού-συνθήκης Διαδρομής Δεδομένων	81
	Μικροπρογραμματισμός με Μικροελεγκτές	82
6.1.10	Αρχιτεκτονική Συστήματος	82
	Προβλήματα.....	83
7	Κεφάλαιο - Ενσωματωμένοι Πυρήνες Γενικού Σκοπού.....	86
	Επεξεργαστές	86
7.1.1	Από τη C στις εντολές Συμβολικής Γλώσσας	86
	Η Διοχέτευση RISC	88
7.1.2	Κίνδυνοι Ελέγχου	91
7.1.3	Κίνδυνοι Δεδομένων.....	92
7.1.4	Δομικοί Κίνδυνοι.....	93
8	Κεφάλαιο - Σύστημα στο Ολοκληρωμένο.....	95
	Η έννοια του Συστήματος στο Ολοκληρωμένο	95
8.1.1	Ο Θίασος των Παικτών.....	96
8.1.2	Διασυνδέσεις SoC για Εξατομικευμένο Υλικό	97
	Τέσσερις Αρχές Σχεδιασμού στην Αρχιτεκτονική SoC	99
8.1.3	Ετερογενής και Κατανεμημένη Επεξεργασία Δεδομένων.....	99
8.1.4	Ετερογενείς και Κατανεμημένες Επικοινωνίες.....	100
8.1.5	Ετερογενής και Κατανεμημένη αποθήκευση	100
8.1.6	Ιεραρχικός έλεγχος.....	102
	Προβλήματα.....	102
9	Κεφάλαιο - Αρχές Επικοινωνίας Υλικού / Λογισμικού	104
	Σύνδεση Υλικού και Λογισμικού	104
	Σχήματα Συγχρονισμού.....	105
9.1.1	Έννοιες Συγχρονισμού	105
9.1.2	Σηματοφόρος	107
9.1.3	Μονόδρομη και διμερής Χειραψία	109
9.1.4	Ανασταλτική και Μη-Ανασταλτική Μεταφορά Δεδομένων.....	111
	Περιορισμένη Επικοινωνία έναντι Περιορισμένων υπολογισμών	112
	Ισχυρή και Χαλαρή Σύζευξη	115
	Προβλήματα.....	116
10	Κεφάλαιο - Δίαυλοι εντός – Ολοκληρωμένου	118
	Συστήματα Διαύλου εντός – Ολοκληρωμένου	118
10.1.1	Λίγα υπαρκτά Πρότυπα Διαύλων εντός – Ολοκληρωμένου	118

10.1.2	Στοιχεία σε Κοινόχρηστο Δίαυλο	118
10.1.3	Στοιχεία σε Δίαυλο Σημείου – σε – Σημείο.....	120
10.1.4	Φυσική Υλοποίηση Διαύλων εντός – Ολοκληρωμένου.....	120
10.1.5	Σύμβαση Ονομασίας Διαύλου	121
10.1.6	Διάγραμμα Χρονισμού Διαύλου	123
10.1.7	Ορισμός του Γενικού Διαύλου	124
	Μεταφορές Διαύλων	124
10.1.8	Απλές Μεταφορές Ανάγνωσης και Εγγραφής.....	124
10.1.9	Ορισμοί Μεγέθους Μεταφορών και Endianess	126
10.1.10	Βελτιωμένες Μεταφορές Διαύλων	127
	Συστήματα Διαύλων Πολλαπλών-Αφεντών	131
10.1.11	Προτεραιότητα Διαύλου	132
10.1.12	Κλείδωμα Διαύλου	133
	Τοπολογίες Διαύλου	136
10.1.13	Μεταγωγείς Διαύλου	137
10.1.14	Δίκτυο σε Ολοκληρωμένο	138
	Προβλήματα.....	141
11	Άσκηση 2.1	143
12	Άσκηση 2.3	144
13	Άσκηση 2.5	145
14	Άσκηση 2.7	146
15	Άσκηση 2.9	147
16	Άσκηση 3.3	149
17	Άσκηση 3.7	150
18	Άσκηση 4.1	151
19	Άσκηση 4.2	153
20	Άσκηση 4.3	154
21	Άσκηση 4.4	156
22	Άσκηση 4.5	157
23	Άσκηση 4.7	159
24	Άσκηση 6.1	160
25	Άσκηση 6.3	162
26	Άσκηση 6.5	163
27	Άσκηση 8.1	164

28	Άσκηση 8.2	164
29	Άσκηση 8.3	167
30	Πρόβλημα 9.1.....	170
31	Πρόβλημα 9.2.....	171
32	Πρόβλημα 9.3.....	172
33	Άσκηση 10.2	174
34	Άσκηση 10.4	175
35	Επίλυση Θεμάτων Ιουνίου 2020	177
	Ερώτηση 1	177
	Ερώτηση 2	179
36	Επίλυση Θεμάτων Σεπτεμβρίου 2020.....	183

1 Κεφάλαιο - Η Φύση του Υλικού και του Λογισμικού

Εισαγωγή στη Συσχεδίαση Υλικού/Λογισμικού

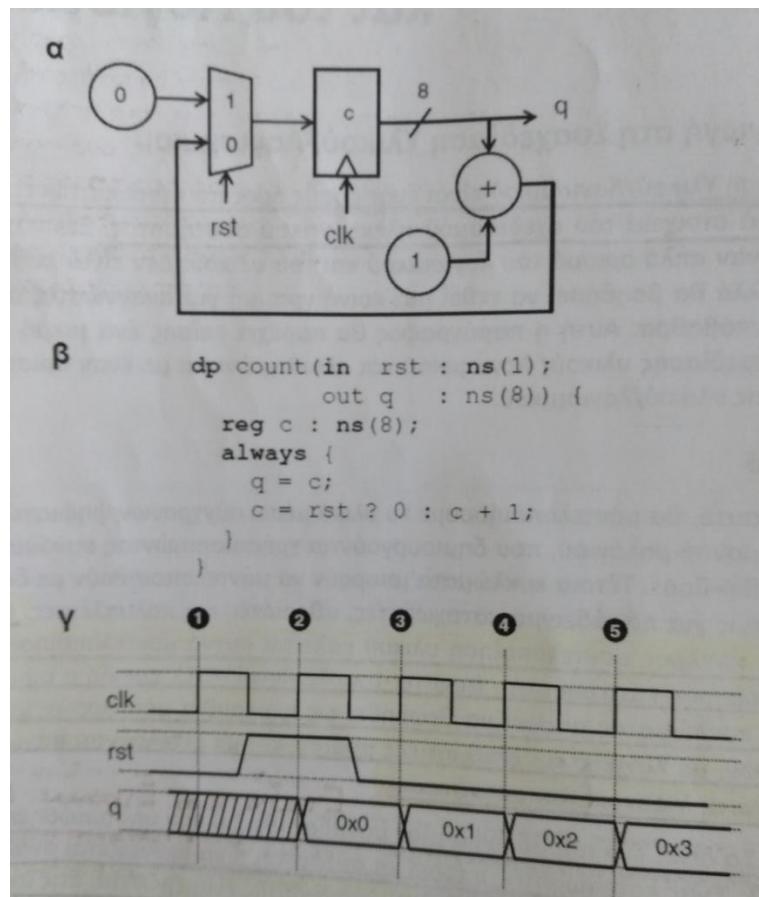
Η Συσχεδίαση Υλικού/Λογισμικού είναι ένας ευρύς όρος που ενσωματώνει πολλά διαφορετικά στοιχεία του σχεδιασμού ηλεκτρονικού συστήματος.

1.1.1 Υλικό

Σημείωση: RTL

Η βασισμένη σε κύκλους μοντελοποίηση υλικού καλείται συχνά μοντελοποίηση επιπέδου μεταφοράς καταχωρητή – (register – transfer – level – RTL), επειδή η συμπεριφορά ενός κυκλώματος μπορεί να θεωρηθεί ως ακολουθία μεταφορών μεταξύ καταχωρητών, με λογικές και αριθμητικές πράξεις και να εκτελούνται πάνω στα σήματα κατά τη διάρκεια των μεταφορών.

Ένας καταχωρητής μπορεί να αυξάνεται ή να καθαρίζεται ανάλογα με την τιμή του σήματος ελέγχου `rst`. Ο καταχωρητής ενημερώνεται στις ανοδικές ακμές ενός σήματος ρολογιού `clk`. Το μήκος λέξης του καταχωρητή είναι 8 bit. Παρόλο που οι συνδέσεις σε αυτή την εικόνα έχουν σχεδιαστεί ως μονές γραμμές, κάθε γραμμή αντιπροσωπεύει μια δέσμη οκτώ καλωδίων.



Εικόνα 1 - Εξαρτήματα Υλικού

Η Εικόνα 1.a χρησιμοποιεί τα γραφικά για να αποτυπώσει το κύκλωμα. Σε αυτό το βιβλίο θα χρησιμοποιήσουμε μια γλώσσα περιγραφής υλικού που ονομάζεται GEZEL. Η Εικόνα 1.b δείχνει την ισοδύναμη περιγραφή



αυτού του κυκλώματος στη γλώσσα GEZEL. Το Κεφάλαιο 5 θα περιγράψει λεπτομερώς τη μοντελοποίηση σε GEZEL.

Η Εικόνα 1.γ απεικονίζει τη συμπεριφορά του κυκλώματος χρησιμοποιώντας ένα διάγραμμα χρονισμού. Σε ένα τέτοιο διάγραμμα, ο χρόνος τρέχει από αριστερά προς τα δεξιά και οι γραμμές του διαγράμματος αντιπροσωπεύουν διαφορετικά σήματα στο κύκλωμα.

Σε αυτό το διάγραμμα, ο καταχωρητής καθαρίζεται στην ακμή ρολογιού 2, και αυξάνεται στην ακμή ρολογιού 3, 4 και 5. Πριν από την ακμή ρολογιού 2, η τιμή του καταχωρητή είναι άγνωστη και το διάγραμμα χρονισμού υποδεικνύει την τιμή του q ως σκιασμένη περιοχή. Θα χρησιμοποιήσουμε διαγράμματα χρονισμού για να περιγράψουμε τη χαμηλού επιπέδου συμπεριφορά των διασυνδέσεων υλικού/λογισμικού και για να περιγράψουμε γεγονότα στους επί-του-ολοκληρωμένου διαύλους (on-chip buses).

Σημείωση: Μοντέλο ενός ρολογιού

a) Το μοντέλο μονού – ρολογιού είναι μια πολύ βολική θεώρηση για έναν σχεδιαστή που απεικονίζει συμπεριφορές (π.χ. κάποιο αλγόριθμο) σε διακριτά βήματα ενός κύκλου ρολογιού. b) Αυτό επιτρέπει στο σχεδιαστή να οραματιστεί πως θα πρέπει να μοιάζει η υλοποίηση σε υλικό ενός συγκεκριμένου αλγορίθμου

Σημείωση: Μειονεκτήματα μοντέλου μονού ρολογιού

c) Το σύγχρονο μοντέλο μονού – ρολογιού δεν μπορεί να περιγράψει όλα τα δυνατά κυκλώματα υλικού. Για παράδειγμα, δεν μπορεί να μοντελοποιήσει γεγονότα σε χρονική ανάλυση μικρότερη από έναν κύκλο ρολογιού.

Ως αποτέλεσμα, ορισμένα στυλ σχεδιασμού υλικού δεν μπορούν να αποτυπωθούν με ένα σύγχρονο μοντέλο μονού – ρολογιού, συμπεριλαμβάνοντας το ασύγχρονο υλικό, δυναμική λογική, υλικό με ρολόι πολλών-φάσεων, και υλικό με μανδαλωτές. Ωστόσο, το σύγχρονο υλικό μονού-ρολογιού αρκεί για να εξηγήσει τις βασικές έννοιες της συσχεδίασης υλικού – λογισμικού σε αυτό το βιβλίο.

Αίστα 1.1 Παράδειγμα C – Εύρεση μέγιστου στοιχείου πίνακα

```
1      int max;  
2  
3      int findmax (int a [10] ) {  
4          unsigned i ;  
5          max = a [0] ;  
6          for (i = 1 ; i < 10 ; i ++ )  
7              if (a [ i ] > max) max = a [ i ] ;  
8      }
```



1.1.2 Λογισμικό

Σημείωση: Συσχεδίαση H/S

Η συσχεδίαση υλικού – λογισμικού ασχολείται με διασυνδέσεις υλικού/λογισμικού. Οι λεπτομέρειες κατασκευής χαμηλού επιπέδου του λογισμικού είναι σημαντικές, επειδή επηρεάζουν άμεσα την απόδοση και το κόστος υλοποίησης της διασύνδεσης υλικού/λογισμικού.

Σημείωση: Αντιστοίχιση εννοιών αποθήκευσης και τύπων αποθήκευσης

Οι μεταβλητές της C αποθηκεύονται σε ένα ενιαίο, κοινόχρηστο χώρο μνήμης, που αντιστοιχεί στη μνήμη που είναι συνδεδεμένη στον μικροεπεξεργαστή. Υπάρχει μια στενή αντιστοιχία μεταξύ των εννοιών αποθήκευσης ενός μικροεπεξεργαστή (καταχωρητές, στοίβα) και στους τύπους αποθήκευσης που υποστηρίζονται στην C (register, int, τοπικές μεταβλητές). Επιπλέον, οι συνήθεις τύποι δεδομένων στην C (char, int) απεικονίζονται απευθείας σε μονάδες αποθήκευσης μικροεπεξεργαστή (byte, λέξη). Κατά συνέπεια, μια λεπτομερής κατανόησης της εκτέλεσης σε C, συνδέεται στενά με μια λεπτομερή κατανόηση της δραστηριότητας του μικροεπεξεργαστή σε χαμηλότερο επίπεδο αφαίρεσης.

Λίστα 1.2 Παράδειγμα συμβολικού κώδικα ARM

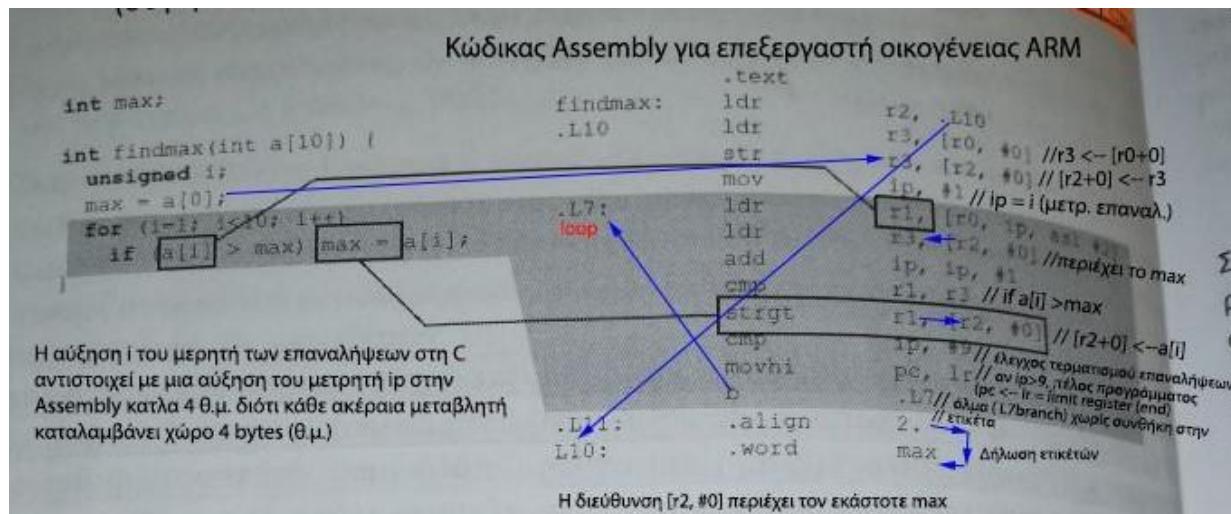
```
.text
findmax:
    ldr r2,    .L10
    ldr r3,    [r0, #0]
    str r3,    [r2, #0]
    mov ip,    #1

.L7:
    ldr r1,    [r0, ip, asl, #2]
    ldr r3,    [r2, #0]
    add ip,    ip, #1
    cmp r1,    r3
    strgt r1, [r2, #0]
    cmp ip,    #9
    movhi pc, lr
    b .L7

.L11:
    .align    2
.L10:
    .word    max
```

Σημείωση: Σύγκριση λίστας 1.1. και 1.2.

Η εντολή if στη C απαιτεί τον υπολογισμό μιας συνθήκης μεγαλύτερης από. Στη συμβολική γλώσσα, μπορεί να βρεθεί μια ισοδύναμη εντολή cmp (σύγκριση).



Eikόνα 2 - Απεικόνιση C σε συμβολική γλώσσα

Σημείωση: Για ποιο λόγο κάνουμε αριστερή ολίσθηση κατά 2 θέσεις στον καταχωρητή r0

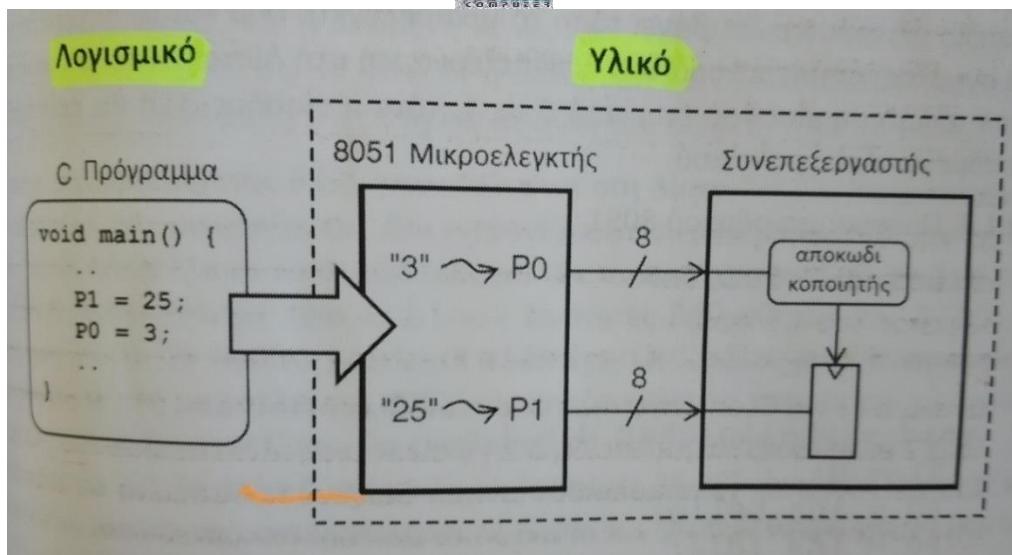
Αυτό δείχνει ότι οι τελεστές r1 και r3 της εντολής σύγκρισης πρέπει να περιέχουν το a[i] και το max του προγράμματος C. Και οι δύο μεταβλητές είναι αποθηκευμένες στη μνήμη. Το a[i] επειδή αποτελεί μια δεικτοδοτημένη μεταβλητή, και το max επειδή είναι καθολική μεταβλητή. Πράγματι, εξετάζοντας την προηγούμενη εντολή στο πρόγραμμα C, μπορείτε να δείτε ότι τόσο η r1 και η r3 ορίζονται από εντολές ldr (load – register – φόρτωσης καταχωρητή) που απαιτούν μια διεύθυνση.

Η διεύθυνση για την φόρτωση του r1 ισούται με [r0, ip, asl, #2], που ισοδυναμεί με την έκφραση r0 + (ip << 2). Αυτό μπορεί να μην είναι προφανές αν είναι η πρώτη φορά που βλέπετε συμβολική γλώσσα του ARM, αλλά είναι κάτι που γρήγορα θα θυμόσαστε μετά από μια σχετική χρήση της συμβολικής γλώσσας. Στην πραγματικότητα, η μορφή της έκφρασης είναι εύκολο να εξηγηθεί. Ο καταχωρητής ip περιέχει τον μετρητή βρόχου, δεδομένου ότι το ip αυξάνεται μία φορά εντός τους σώματος του βρόχου, και η τιμή του ip συγκρίνεται με την τιμή ορίου βρόχου 9. Ο καταχωρητής r0 είναι η διεύθυνση βάσης (base address) του πίνακα a [], η θέση στη μνήμη όπου αποθηκεύεται το a [0]. Η ολίσθηση κατά 2 είναι απαραίτητη επειδή ο a [] είναι ένας πίνακας ακεραίων. Οι μικροεπεξεργαστές χρησιμοποιούν μνήμη διευθυνσιοδοτημένη σε επίπεδο byte (byte – addressable), και οι ακέραιοι αριθμοί αποθηκεύονται σε τμήμα εύρους 4-byte.

Τέλος, η εκχώρηση υπό όρους της μεταβλητής max στη C δεν υλοποιείται με τη χρήση εντολών διακλάδωσης υπό συνθήκη στη συμβολική γλώσσα. Αντ' αυτού χρησιμοποιείται μια εντολή strgt (store-if-greater-αποθήκευσε εάν μεγαλύτερο). Αυτή είναι μια εκτελούμενη υπό συνθήκη (predicted) εντολή, που εκτελείται μόνο όταν μια δεδομένη σημαία συνθήκης είναι αληθής.

1.1.3 Υλικό και Λογισμικό

Ένα πρόγραμμα RTL είναι ένα μοντέλο ενός δικτυώματος λογικών πυλών.



Εικόνα 3 - Ένα μοντέλο συσχεδίασης

Στην Εικόνα 3 παρουσιάζεται ένας μικροελεγκτής 8051 και ένας συνδεδεμένος συνεπεξεργαστής. Ο συνεπεξεργαστής είναι συνδεδεμένος στον μικροελεγκτή 8051 μέσω δύο 8-bit θυρών P0 και P1. Ένα πρόγραμμα C εκτελείται στον μικροελεγκτή 8051 και αυτό το πρόγραμμα περιέχει εντολές για την εγγραφή δεδομένων σε αυτές τις δύο θύρες. Όταν εμφανίζεται μια δεδομένη, προκαθορισμένη τιμή στη θύρα P0, ο συνεπεξεργαστής θα κάνει ένα αντίγραφο της τιμής που βρίσκεται στη θύρα P1 σε έναν εσωτερικό καταχωρητή.

1.1.4 Ορίζοντας τη Συσχεδίαση Υλικού/Λογισμικού

Σημείωση: Ορισμός Συσχεδίασης

Το προηγούμενο παράδειγμα μας οδηγεί στον ακόλουθο παραδοσιακό ορισμό της συσχεδίασης υλικού / λογισμικού.

Σημείωση: Βασικός Ορισμός Συσχεδίασης H/S

Η Συσχεδίαση Υλικού/Λογισμικού είναι ο σχεδιασμός συνεργαζόμενων τμημάτων υλικού και τμημάτων λογισμικού σε μια ενιαία προσπάθεια σχεδιασμού.

Σημείωση: Εφαρμογή Συσχεδίασης

Για παράδειγμα, εάν θα σχεδιάζατε την αρχιτεκτονική ενός επεξεργαστή και την ίδια στιγμή αναπτύσσατε ένα πρόγραμμα που θα μπορούσε να τρέξει σε αυτόν τον επεξεργαστή, τότε θα χρησιμοποιούσατε συσχεδίαση υλικού/λογισμικού.

Σημείωση: Παραδείγματα πλατφορμών υλικού

- Μια Συστοιχία Πυλών Προγραμματιζόμενου Πεδίου (Field Programmable Gate Array – FPGA) είναι ένα κύκλωμα υλικού που μπορεί να επαναπρογραμματιστεί σε ένα καθοριζόμενο από το χρήστη κατάλογο κόμβων (netlist) ψηφιακών πυλών. Το πρόγραμμα για ένα FPGA είναι ένα 'bitstream', και χρησιμοποιείται για τη διαμόρφωση της τοπολογίας των κόμβων του.



- Ένας μαλακός πυρήνας (soft-core) είναι ένας επεξεργαστής που υλοποιείται στο bitstream ενός FPGA. Ωστόσο, ο ίδιος ο μαλακός-πυρήνας μπορεί επίσης να εκτελέσει ένα πρόγραμμα C.
- Ένας Επεξεργαστής Ψηφιακού Σήματος (Digital-Signal Processor – DSP) είναι ένας επεξεργαστής με εξειδικευμένο σύνολο εντολών, βελτιστοποιημένος για εφαρμογές επεξεργασίας σημάτων.
- Ο Επεξεργαστής Συνόλου Εντολών Ειδικής Εφαρμογής (Application – Specific Instruction – set Processor – ASIP) είναι ένας επεξεργαστής με ένα προσαρμόσιμο σύνολο εντολών. Το υλικό ενός τέτοιου επεξεργαστή μπορεί να επεκταθεί, και αυτές οι επεκτάσεις υλικού μπορούν να ενσωματωθούν ως νέες εντολές για τον επεξεργαστή.
- Ο Επεξεργαστής CELL (CELL processor), ο οποίος χρησιμοποιείται στο Playstation-3, περιέχει έναν επεξεργαστή ελέγχου και οκτώ επεξεργαστές – υπηρέτες, που διασυνδέονται μέσω ενός δικτύου υψηλής ταχύτητας επί του ολοκληρωμένου.

Σημείωση: Ορισμός Εφαρμογής

Ας ορίσουμε την εφαρμογή (application) ως τη συνολική λειτουργία ενός σχεδιασμού, καλύπτοντας την υλοποίησή της τόσο σε υλικό όσο και λογισμικό. Μπορούμε να ορίσουμε τη συσχεδίαση υλικού/λογισμικού ως εξής:

Σημείωση: Εναλλακτικός Ορισμός Συσχεδίασης S/H

Συσχεδίαση Υλικού/Λογισμικού είναι ο διαμελισμός και σχεδιασμός μιας εφαρμογής μέσω σταθερών και ευέλικτων εξαρτημάτων.

Χρησιμοποιήσαμε τον όρο 'σταθερό εξάρτημα' αντί του εξαρτήματος υλικού, και 'ευέλικτο εξάρτημα' αντί του λογισμικού. Ένα σταθερό εξάρτημα είναι συχνά υλικό, αλλά δεν περιορίζεται σε αυτό. Ομοίως, ένα ευέλικτο εξάρτημα είναι συχνά λογισμικό, αλλά δεν περιορίζεται σε αυτό.

Η Αναζήτηση για Ενεργειακή Αποδοτικότητα

Στην πραγματικότητα, η επιλογή μεταξύ μιας υλοποίησης υλικού και μιας υλοποίησης λογισμικού είναι πολύ πιο λεπτή και καθοδηγείται τόσο από τεχνολογικούς όσο και από οικονομικούς λόγους. Ξεκινάμε με δύο τεχνολογικά επιχειρήματα (την απόδοση και την ενεργειακή αποδοτικότητα), και στη συνέχεια παρέχουμε μια πιο ισορροπημένη οπτική για το συμβιβασμό μεταξύ υλικού και λογισμικού.

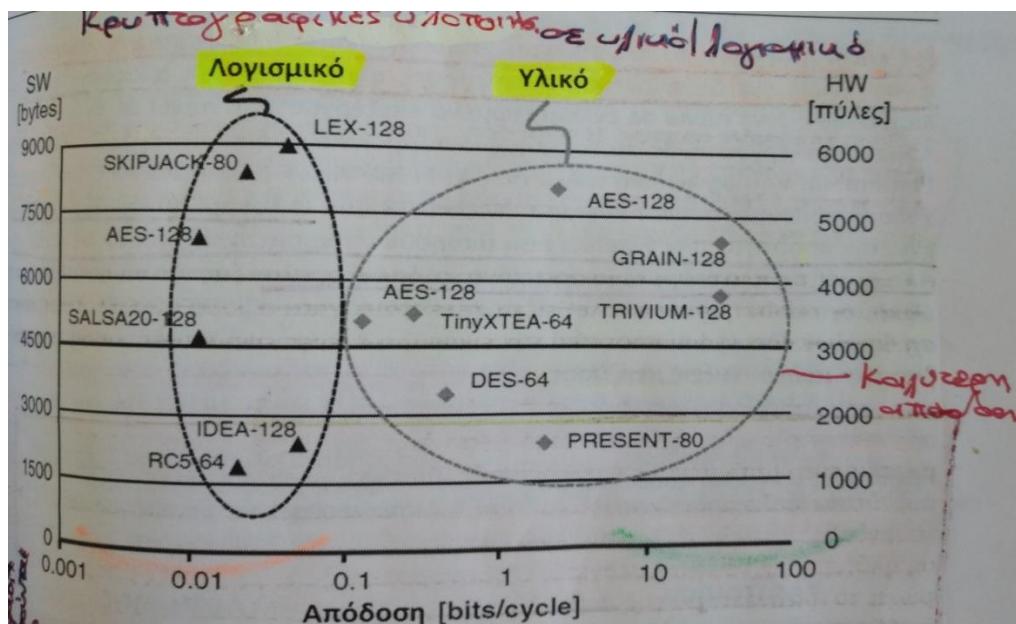
1.1.5 Απόδοση

Σημείωση: Αναζήτηση Ενεργειακής Αποδοτικότητας / Σύγκριση Υλικού / Λογισμικού

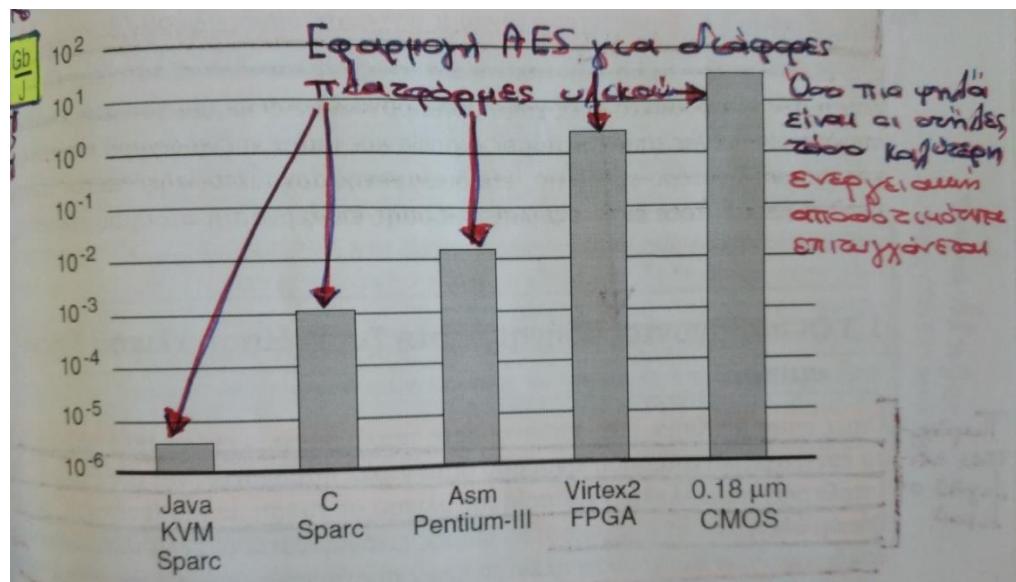
Ένας άλλος τρόπος για να συγκρίνετε υλικό και λογισμικό είναι να τα συγκρίνετε με βάση την απόδοσή τους. Η απόδοση μπορεί να εκφραστεί ως η ποσότητα της εργασίας που γίνεται ανά μονάδα χρόνου. Ας ορίσουμε ως μονάδα εργασίας την επεξεργασία 1 bit δεδομένων. Η μονάδα χρόνου μπορεί να εκφραστεί σε κύκλους ρολογιού ή σε δευτερόλεπτα.

1.1.6 Ενεργειακή Αποδοτικότητα

Ένας δεύτερος σημαντικός παράγοντας στην επιλογή μεταξύ υλικού και λογισμικού είναι η ενέργεια που απαιτείται για τους υπολογισμούς. Αυτό είναι ιδιαίτερα σημαντικό για φορητές εφαρμογές που λειτουργούν με μπαταρία. Η ενεργειακή αποδοτικότητα (energy-efficiency) είναι η ποσότητα της χρήσιμης εργασίας που γίνεται ανά μονάδα ενέργειας. Καλύτερη ενεργειακή αποδοτικότητα, σαφώς συνεπάγει μεγαλύτερη διάρκεια ζωής της μπαταρίας.



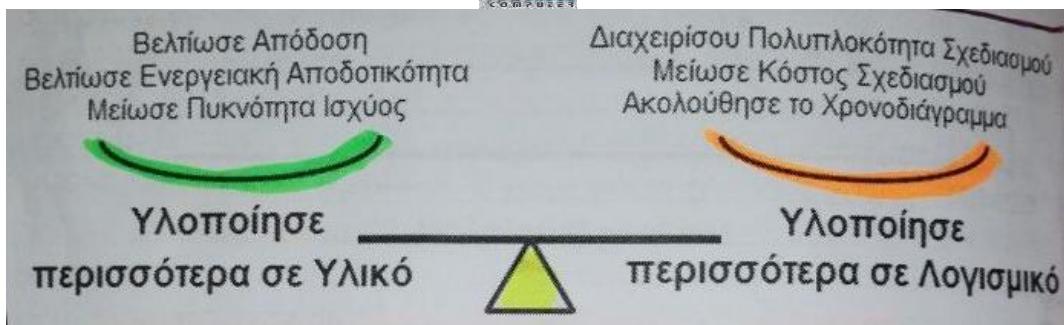
Εικόνα 4 - Κρυπτογραφία σε μικρές ενσωματωμένες πλατφόρμες



Εικόνα 5 - Ενεργειακή αποδοτικότητα

Η Εικόνα 5 δείχνει το παράδειγμα μιας συγκεκριμένης εφαρμογής κρυπτογράφησης (AES) για διαφορετικές πλατφόρμες υλοποίησης.

Ο λογαριθμικός άξονας Y δείχνει την ποσότητα των Gigabits που μπορούν να κρυπτογραφηθούν σε κάθε μια από αυτές τις πλατφόρμες με ένα μόνο Joule ενέργειας.



Εικόνα 6 - Παράγοντες οδήγησης στη συσχεδίαση υλικού/λογισμικού

Οι παράγοντες οδήγησης στη Συσχέτιση Υλικού / Λογισμικού

Η ενεργειακή αποδοτικότητα και η σχετική απόδοση είναι δύο σημαντικοί παράγοντες για την επιλογή μιας (σταθερής, παράλληλης) υλοποίησης υλικού σε σχέση με μια (ευέλικτη, ακολουθιακή) υλοποίηση λογισμικού. Στο σχεδιασμό των σύγχρονων ηλεκτρονικών συστημάτων, πρέπει να γίνουν πολλοί συμβιβασμοί μεταξύ αντικρουόμενων στόχων. Η Εικόνα 6 δείχνει ότι ορισμένοι παράγοντες υποστηρίζουν μεγαλύτερη χρήση λογισμικού, ενώ άλλοι παράγοντες συνηγορούν για περισσότερο υλικό. Τα παρακάτω είναι επιχειρήματα υπέρ της αύξησης της ποσότητας του επί-του-ολοκληρωμένου αφοσιωμένου υλικού.

Σημείωση: Επιχειρήματα που συνηγορούν στην αύξηση του υλικού

- Απόδοση:** Το κλασικό επιχείρημα υπέρ του σχεδιασμού ειδικού υλικού είναι η αυξημένη απόδοση: ολοκληρώνεται περισσότερη εργασία ανά κύκλο ρολογιού. Η αυξημένη απόδοση προκύπτει από την εξειδίκευση της αρχιτεκτονικής που χρησιμοποιείται για την υλοποίηση της εφαρμογής.
- Ενεργειακή αποδοτικότητα:** Σχεδόν κάθε ηλεκτρονικό προϊόν σήμερα φέρει μια μπαταρία (iPod, PDA, κινητό τηλέφωνο, συσκευή Bluetooth, ...). Οι μπαταρίες έχουν περιορισμένες δυνατότητες αποθήκευσης ενέργειας. Από την άλλη πλευρά, αυτές οι συσκευές χρησιμοποιούνται για παρόμοιες υπηρεσίες και εφαρμογές με τους παραδοσιακούς προσωπικούς υπολογιστές υψηλών επιδόσεων. Μετακινώντας (μέρος) του ευέλικτου λογισμικού ενός σχεδιασμού σε σταθερό υλικό, η ενεργειακή αποδοτικότητα της συνολικής εφαρμογής μπορεί να αυξηθεί (βλ. Εικόνα 5).
- Πυκνότητα ισχύος:** Η απορρόφηση ισχύος ενός κυκλώματος και το σχετικό θερμικό προφίλ, είναι άμεσα ανάλογα με τη συχνότητα ρολογιού τους. Στους σύγχρονους επεξεργαστές, η πυκνότητα ισχύος βρίσκεται στα όρια της οικονομικά αποδοτικής τεχνολογίας ψύξης. Περαιτέρω βελτίωση της απόδοσης δεν μπορεί, συνεπώς, να επιτευχθεί με την περαιτέρω αύξηση της συχνότητας ρολογιού.

Σημείωση: Παράλληλες Αρχιτεκτονικές

Στις σημερινές υποψηφιότητες για παράλληλους υπολογιστές περιλαμβάνονται οι συμμετρικοί πολυεπεξεργαστές που είναι συνδεδεμένοι στην ίδια μνήμη (SMP), οι Συστοιχίες Πυλών Προγραμματίζόμενου Πεδίου που χρησιμοποιούνται ως μηχανές επιτάχυνσης για κλασικούς επεξεργαστές (FPGA), και οι αρχιτεκτονικές πολλαπλών – πυρήνων όπως οι Μηχανές Επεξεργασίας Γραφικών με γενικού σκοπού υπολογιστικές δυνατότητες (GPU). Είναι πιθανό ότι όλες αυτές οι αρχιτεκτονικές θα συνυπάρχουν για ένα μεγάλο χρονικό διάστημα.



Σημειώστε ότι οι παράλληλες αρχιτεκτονικές δεν προγραμματίζονται σε C. Οι τρέχουσες παράλληλες αρχιτεκτονικές συνδέονται όλες με το δικό τους στυλ παράλληλου προγραμματισμού.

Σημείωση: Επιχειρήματα που συνηγορούν στην αύξηση του λογισμικού

Τα ακόλουθα επιχειρήματα, από την άλλη πλευρά, υποστηρίζουν την ευελιξία και, συνεπώς, την αύξηση της χρήση του λογισμικού επί του ολοκληρωμένου.

- **Σχεδιαστική Πολυπλοκότητα:** Τα σύγχρονα ηλεκτρονικά συστήματα είναι τόσο περίπλοκα, που δεν είναι καλή ιδέα να κάνετε hard-code όλες τις αποφάσεις σχεδιασμού σε σταθερό υλικό. Αντ' αυτού, μια συνήθης προσέγγιση είναι να διατηρείται η υλοποίηση όσο το δυνατόν πιο ευέλικτη, συνήθως χρησιμοποιώντας προγραμματιζόμενους επεξεργαστές που εκτελούν λογισμικό. Το λογισμικό και η ευελιξία που παρέχει, χρησιμοποιείται ως μηχανισμός για την ανάπτυξη της εφαρμογής σε ένα υψηλότερο επίπεδο αφαίρεσης (στο λογισμικό), ως μηχανισμός αντιμετώπισης των μελλοντικών αναγκών, και ως κάποιος μηχανισμός επίλυσης σφαλμάτων.
- **Κόστος Σχεδιασμού:** Τα νέα ολοκληρωμένα είναι πολύ ακριβά στο σχεδιασμό. Ως αποτέλεσμα, οι σχεδιαστές υλικού κάνουν τα ολοκληρωμένα προγραμματιζόμενα έτσι ώστε αυτά τα ολοκληρωμένα να μπορούν να επαναχρησιμοποιηθούν για πολλαπλά προϊόντα ή γενιές προϊόντων.
- **Συρρίκνωση Χρονοδιαγράμματος Σχεδιασμού:** Κάθε νέα γενιά τεχνολογίας τείνει να αντικαταστήσει την παλαιότερη πιο γρήγορα. Επιπλέον, κάθε μία από αυτές τις νέες τεχνολογίες είναι πιο περίπλοκη σε σύγκριση με την προηγούμενη γενιά. Για έναν μηχανικό σχεδιασμού, αυτό σημαίνει ότι κάθε νέα γενιά προϊόντων απαιτεί περισσότερη δουλειά και η δουλειά θα πρέπει να ολοκληρώνεται σε μικρότερο χρονικό διάστημα.

Η συρρίκνωση των χρονοδιαγραμμάτων σχεδιασμού απαιτεί οι ομάδες μηχανικών να δουλεύουν σε πολλαπλές εργασίες ταυτόχρονα: το υλικό και το λογισμικό αναπτύσσονται ταυτόχρονα. Η ομάδα ανάπτυξης λογισμικού θα ξεκινήσει την ανάπτυξη λογισμικού μόλις καθοριστούν τα χαρακτηριστικά της πλατφόρμας υλικού.

Η εύρεση της σωστής ισορροπίας μεταξύ όλων αυτών των παραγόντων είναι προφανώς ένα εξαιρετικά περίπλοκο πρόβλημα.

Η προσθήκη υλικού σε μια λύση λογισμικού μπορεί να αυξήσει την απόδοση της συνολικής εφαρμογής, αλλά θα απαιτήσει επίσης περισσότερους πόρους.

[Ο Χώρος Συσχεδίασης Υλικού – Λογισμικού](#)

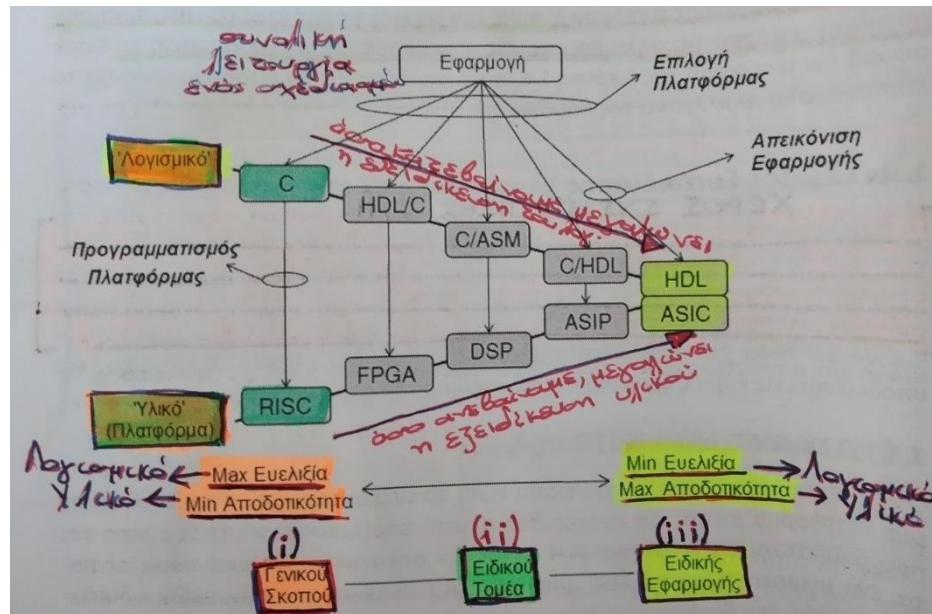
Σημείωση: Χώρος Συσχεδίασης S/H

Οι συμβιβασμοί που συζητήθηκαν στην προηγούμενη παράγραφο πρέπει να γίνουν στο πλαίσιο ενός χώρου σχεδιασμού (design space). Για μια συγκεκριμένη εφαρμογή, υπάρχουν πολλές διαφορετικές πιθανές λύσεις. Η συλλογή όλων αυτών των υλοποιήσεων αποκαλείται χώρος συσχεδίασης υλικού-λογισμικού.

1.1.7 Ο Χώρος Σχεδιασμού Πλατφόρμας

Η απεικόνιση μιας εφαρμογής σε μια πλατφόρμα απαιτεί το γράψιμο λογισμικού για αυτήν την πλατφόρμα, και, αν χρειαστεί, προσαρμογή του υλικού της πλατφόρμας. Το λογισμικό, όπως και το υλικό, έχουν πολύ διαφορετικό νόημα ανάλογα με την πλατφόρμα.

Στην περίπτωση ενός επεξεργαστή RISC, το λογισμικό γράφεται σε C, ενώ το υλικό είναι ένας επεξεργαστής γενικού σκοπού.



Εικόνα 7 - Ο χώρος συσχεδίασης υλικού/λογισμικού

Σημείωση: Περιγραφή Εικόνας 7

- Στην περίπτωση μιας Συστοιχίας Πυλών Προγραμματιζόμενου Πεδίου, λογισμικό γράφεται σε μια Γλώσσα Περιγραφής Υλικού (HDL). Όταν FPGA περιέχει έναν επεξεργαστή μαλακού - πυρήνα, όπως αναφέρθηκε παραπάνω, θα γράψουμε επίσης πρόσθετο λογισμικό σε C.
- Ένας Επεξεργαστής Ψηφιακού Σήματος (DSP) χρησιμοποιεί ένα συνδυασμό C και συμβολικού κώδικα για λογισμικό. Το υλικό είναι μια εξειδικευμένη αρχιτεκτονική επεξεργαστή, προσαρμοσμένη στις λειτουργίες επεξεργασίας σημάτων.
- Ένας Επεξεργαστής Συνόλου Εντολών Ειδικής Εφαρμογής είναι ένας επεξεργαστής που μπορεί να ειδικευτεί σε συγκεκριμένο πεδίο εφαρμογής για παράδειγμα με την προσθήκη νέων εντολών και επεκτείνοντας τη διαδρομή δεδομένων του επεξεργαστή. Το "λογισμικό" ενός ASIP μπορεί να περιέχει κώδικα C καθώς και μια περιγραφή υλικού για τις επεκτάσεις του επεξεργαστή.
- Τέλος, στην περίπτωση ενός ASIC, η εφαρμογή γράφεται σε HDL, η οποία στη συνέχεια μετατρέπεται σε ένα hardcoded κατάλογο κόμβων. Σε αντίθεση με άλλες πλατφόρμες, οι ASICs είναι συνήθως μη προγραμματιζόμενοι. Σε ένα ASIC, η εφαρμογή και η πλατφόρμα έχουν συγχωνευθεί σε μια ενιαία οντότητα.

Οι πλατφόρμες στην Εικόνα 7 είναι οργανωμένες, από αριστερά προς τα δεξιά, σύμφωνα με την ευελιξία τους: Οι πλατφόρμες γενικού σκοπού, όπως οι RISC και τα FPGA, είναι σε θέση να υποστηρίξουν ένα ευρύ

φάσμα εφαρμογών. Οι πλατφόρμες που σχετίζονται με την εφαρμογή, όπως το ASIC, έχουν βελτιστοποιηθεί ώστε να εκτελούν μια μοναδική εφαρμογή. Ανάμεσα στη γενικού σκοπού πλατφόρμα και στην ειδικής-εφαρμογής, βρίσκεται μια τρίτη κατηγορία αρχιτεκτονικών που ονομάζεται πλατφόρμα ειδικού-τομέα (domain-specific). Οι πλατφόρμες ειδικού-τομέα έχουν βελτιστοποιηθεί για να εκτελούν εφαρμογές από έναν συγκεκριμένο τομέα. Η επεξεργασία σήματος, η κρυπτογραφία, η δικτύωση, είναι όλα παραδείγματα τομέων. Ένας τομέας μπορεί να έχει υποτομείς. Για παράδειγμα, θα μπορούσαμε να διαχωρίσουμε περαιτέρω την επεξεργασία σήματος στην επεξεργασία φωνητικού σήματος και στην επεξεργασία σήματος βίντεο και να αναπτύξουμε βελτιστοποιημένες πλατφόρμες για κάθε μια από αυτές τις περιπτώσεις. Το DSP και το ASIP είναι δύο παραδείγματα πλατφορμών ειδικού-τομέα.

1.1.8 Απεικόνιση εφαρμογών

Κάθε μία από τις παραπάνω πλατφόρμες στην Εικόνα 7 παρουσιάζει ένα διαφορετικό συμβιβασμό μεταξύ ευελιξίας και απόδοσης. Το σφηνοειδές σχήμα της Εικόνας 7 εκφράζει αυτήν την ιδέα και αυτό μπορεί να ερμηνευτεί ως εξής.

Σημείωση: Ευελιξία πλατφόρμας

Ευελιξία (flexibility) σημαίνει πόσο καλά μπορεί να προσαρμοστεί η πλατφόρμα σε διαφορετικές εφαρμογές. Η ευελιξία στις πλατφόρμες είναι επιθυμητή επειδή επιτρέπει στους σχεδιαστές να κάνουν αλλαγές στην εφαρμογή μετά την κατασκευή της πλατφόρμας. Πολύ ευέλικτες πλατφόρμες, όπως η RISC και τα FPGA, προγραμματίζονται με γλώσσες γενικού σκοπού. Όταν μια πλατφόρμα γίνεται πιο εξειδικευμένη, ο προγραμματισμός τείνει, επίσης, να γίνει πιο εξειδικευμένος. Αυτό μπορούμε να το απεικονίσουμε, με το να σχεδιάσουμε την εφαρμογή πιο κοντά στην πλατφόρμα.

Διαφορετικές πλατφόρμες μπορεί επίσης να παρέχουν διαφορετικά επίπεδα απόδοσης. Η απόδοση (efficiency) μπορεί είτε να αφορά την απόλυτη απόδοση (δηλαδή την απόδοση χρόνου) είτε την απόδοση χρησιμοποιώντας ενέργεια για την υλοποίηση υπολογισμών. Η σωστή εφαρμογή, σε μια εξειδικευμένη πλατφόρμα θα είναι πιο αποδοτική από μια γενική πλατφόρμα, επειδή τα εξαρτήματα υλικού της βελτιστοποιούνται για αυτήν την εφαρμογή. Μπορούμε να το φανταστούμε αυτό μεταφέροντας την πλατφόρμα πλησιέστερα στην εφαρμογή στην περίπτωση εξειδικευμένων πλατφορμών.

Σημείωση: Επιλογή Πλατφόρμας για μια Προδιαγραφή Απεικόνιση Εφαρμογής σε Επιλεγμένη Πλατφόρμα

Ένα ενδιαφέρον, αλλά πολύ δύσκολο ερώτημα είναι πως μπορεί κανείς να επιλέξει μια πλατφόρμα για μια δεδομένη προδιαγραφή και πως μπορεί κάποιος να απεικονίσει μια εφαρμογή σε μια επιλεγμένη πλατφόρμα. Από αυτά τα δύο ερωτήματα, το πρώτο είναι το πιο δύσκολο. Οι σχεδιαστές συνήθως απαντούν με βάση την προηγούμενη εμπειρία τους με παρόμοιες εφαρμογές. Το δεύτερο ερώτημα είναι επίσης πολύ δύσκολο, αλλά είναι δυνατό να απαντηθεί με ένα πιο συστηματικό τρόπο, χρησιμοποιώντας μια μεθοδολογία σχεδιασμού.

Σημείωση: Μέθοδος σχεδιασμού

Μια μέθοδος σχεδιασμού (design method) είναι μια συστηματική ακολουθία βημάτων για την μετατροπή μιας προδιαγραφής σε μια υλοποίηση. Οι μέθοδοι σχεδιασμού καλύπτουν πολλές πτυχές της απεικόνισης εφαρμογής, όπως η βελτιστοποίηση της χρήσης μνήμης, η απόδοση του σχεδιασμού, η χρήση πόρων, η ακρίβεια και ανάλυση των τύπων δεδομένων κ.ο.κ. Μια μέθοδος σχεδιασμού είναι μια τυποποιημένη σειρά βημάτων σχεδιασμού.

Ο Δυϊσμός του Σχεδιασμού Υλικού και του Σχεδιασμού Λογισμικού

Στην πραγματικότητα, το υλικό και το λογισμικό είναι δυϊκά μεταξύ τους από πολλές απόψεις.

- Παράδειγμα σχεδίασης:** Σε ένα μοντέλο υλικού, τα στοιχεία κυκλώματος λειτουργούν παράλληλα. Έτσι, χρησιμοποιώντας περισσότερα στοιχεία κυκλώματος μπορεί να γίνει περισσότερη δουλειά μέσα σε ένα μόνο κύκλο ρολογιού. Το λογισμικό, από την άλλη πλευρά, λειτουργεί ακολουθιακά. Χρησιμοποιώντας περισσότερες λειτουργίες, ένα πρόγραμμα λογισμικού θα χρειαστεί περισσότερο χρόνο για να ολοκληρωθεί. Ο σχεδιασμός απαιτεί την αποσύνθεση μιας προδιαγραφής σε χαμηλότερου επιπέδου πρωταρχικά στοιχεία, όπως π.χ. πύλες (σε υλικό) και εντολές (σε λογισμικό). Ο σχεδιαστής υλικού επιλύει προβλήματα κάνοντας αποσύνθεση στο χώρο, ενώ ένας σχεδιαστής λογισμικού λύνει προβλήματα με αποσύνθεση στο χρόνο.
- Κόστος πόρων:** Το κόστος πόρων υπόκειται σε παρόμοια δυϊκότητα μεταξύ του υλικού και το λογισμικό. Η αποσύνθεση στο χώρο, όπως χρησιμοποιείται από έναν σχεδιαστή υλικού, σημαίνει ότι ένας πιο σύνθετος σχεδιασμός απαιτεί περισσότερες πύλες. Με την αποσύνθεση στο χρόνο, όπως χρησιμοποιείται από έναν σχεδιαστή λογισμικού, συνεπάγεται ότι ένας πιο πολύπλοκος σχεδιασμός απαιτεί περισσότερες εντολές για να ολοκληρωθεί. Επομένως, το κόστος πόρων για το υλικό είναι η επιφάνεια κυκλώματος, ενώ το κόστος πόρων για το λογισμικό είναι ο χρόνος εκτέλεσης.
- Ευελιξία:** Το λογισμικό υπερέχει έναντι του υλικού στην υποστήριξη της ευελιξίας. Ευελιξία είναι η ευκολία με την οποία η εφαρμογή μπορεί να τροποποιηθεί ή να προσαρμοστεί. Στο λογισμικό, η ευελιξία είναι ουσιαστικά δωρεάν. Στο υλικό, από την άλλη πλευρά, η ευελιξία δεν είναι τετριμμένη. Η ευελιξία στο υλικό απαιτεί τα στοιχεία του κυκλώματος να μπορούν να επαναχρησιμοποιηθούν εύκολα για διαφορετικές δραστηριότητες ή υπολειτουργίες σε ένα σχεδιασμό. Ένας σχεδιαστής υλικού πρέπει να σκεφτεί προσεκτικά για την επαναχρησιμοποίηση: η ευελιξία πρέπει να σχεδιαστεί στο κύκλωμα.
- Παραλληλία:** Στο υλικό, η παραλληλία έρχεται δωρεάν ως μέρος του υποδείγματος σχεδιασμού. Για το λογισμικό, από την άλλη πλευρά, η παραλληλία αποτελεί σημαντική πρόκληση. Εάν μόνο ένας επεξεργαστής είναι διαθέσιμος, το λογισμικό μπορεί να υλοποιήσει μόνο ταυτοχρονισμό (concurrency), ο οποίος απαιτεί τη χρήση ειδικών προγραμματιστικών κατασκευών όπως τα νήματα. Όταν υπάρχουν πολλοί επεξεργαστές, μπορεί να γίνει μια πραγματικά παράλληλη υλοποίηση λογισμικού, αλλά η επικοινωνία μεταξύ των επεξεργαστών και ο συγχρονισμός αποτελεί μια πρόκληση.

Πίνακας 1.1 Ο δυισμός του σχεδιασμού υλικού και λογισμικού

Κριτήρια δυϊσμού	Υλικό	Λογισμικό
Υπόδειγμα Σχεδιασμού	Αποσύνθεση στο χώρο	Αποσύνθεση στο χρόνο
Κόστος πόρων	Επιφάνεια (#πυλών)	Χρόνος (#εντολών)
Ευελιξία	Πρέπει να σχεδιαστεί-εντός	Αυτονόητη
Παραλληλία	Αυτονόητη	Πρέπει να σχεδιαστεί-εντός
Μοντελοποίηση	Μοντέλο ≠ υλοποίηση	Μοντέλο ~ υλοποίηση
Επαναχρησιμοποίηση	Μη σύνηθες	Σύνηθες

- Μοντελοποίηση:** Στο λογισμικό, η μοντελοποίηση και η υλοποίηση είναι πολύ κοντά. Πράγματι, όταν ένας σχεδιαστής γράφει ένα πρόγραμμα C η μεταγλώττιση του προγράμματος για τον κατάλληλο επεξεργαστή στόχο θα έχει επίσης ως αποτέλεσμα την υλοποίηση του προγράμματος. Στο υλικό, το μοντέλο και η υλοποίηση ενός σχεδίου είναι διακριτές έννοιες. Αρχικά, ένας σχεδιασμός υλικού μοντελοποιείται χρησιμοποιώντας μια Γλώσσα Περιγραφής Υλικού (Hardware Description Language). Μια τέτοια περιγραφή υλικού μπορεί να προσομοιωθεί, αλλά δεν αποτελεί υλοποίηση του πραγματικού κυκλώματος. Οι σχεδιαστές υλικού χρησιμοποιούν μια γλώσσα περιγραφής υλικού, και τα προγράμματά τους είναι τα μοντέλα τα οποία αργότερα μετασχηματίζονται σε υλοποίηση. Οι σχεδιαστές λογισμικού χρησιμοποιούν μια γλώσσα προγραμματισμού λογισμικού και τα προγράμματα τους αποτελούν από μόνα τους μια υλοποίηση.
- Επαναχρησιμοποίηση:** Τέλος, το υλικό και το λογισμικό είναι επίσης πολύ διαφορετικά όταν πρόκειται για Επαναχρησιμοποίηση Πνευματικής Ιδιοκτησία (Intellectual property – IP), ή επαναχρησιμοποίηση IP. Η ιδέα της επαναχρησιμοποίησης IP είναι ότι ένα εξάρτημα ενός μεγαλύτερου κυκλώματος ή ένα πρόγραμμα, μπορεί να κατοχυρωθεί και αργότερα να επαναχρησιμοποιηθεί στα πλαίσια ενός διαφορετικού σχεδιασμού. Στο λογισμικό, η επαναχρησιμοποίηση IP έχει γνωρίσει δραματικές αλλαγές τα τελευταία χρόνια λόγω του λογισμικού ανοιχτού κώδικα και της εξάπλωσης ανοιχτών πλατφορμών. Όταν σχεδιάζεται ένα σύνθετο πρόγραμμα αυτές τις μέρες, οι σχεδιαστές θα ξεκινήσουν από ένα σύνολο από τυπικές βιβλιοθήκες που είναι καλά τεκμηριωμένες και είναι διαθέσιμες σε μεγάλο εύρος πλατφορμών. Για τον σχεδιασμό υλικού, η επαναχρησιμοποίηση IP εξακολουθεί να είναι στα σπάργανα. Σε σύγκριση με το λογισμικό, η επαναχρησιμοποίηση-IP για υλικό, έχει να διανύσει πολύ δρόμο.

Αυτή η συνοπτική σύγκριση δείχνει ότι από πολλές απόψεις, ο σχεδιασμός υλικού και ο σχεδιασμός λογισμικού βασίζονται σε δυϊκές έννοιες. Κατά συνέπεια, το να είμαστε σε θέση να μεταβούμε αποτελεσματικά από το έναν κόσμο σχεδιασμού στον άλλο, είναι ένα σημαντικό πλεονέκτημα για ένα συσχεδιαστή



υλικού/λογισμικού. Στο βιβλίο αυτό, θα βασιστούμε σε αυτό το δυϊσμό, και θα προσπαθήσουμε να συνδυάσουμε τις καλύτερες ιδέες του σχεδιασμού υλικού με τις καλύτερες ιδέες του σχεδιασμού λογισμικού.

[Μοντελοποίηση Επιπέδου Αφαίρεσης](#)

Σημείωση: Σημασία / Ορισμός Επιπέδου Αφαίρεσης

Το επίπεδο αφαίρεσης ενός μοντέλου είναι η ποσότητα λεπτομέρειας που είναι διαθέσιμη σε ένα μοντέλο. Ένα χαμηλότερο επίπεδο αφαίρεσης έχει περισσότερες λεπτομέρειες, αλλά η κατασκευή ενός σχεδιασμού σε χαμηλότερο επίπεδο αφαίρεσης απαιτεί περισσότερη προσπάθεια.

Σημείωση: Κριτήριο Επιπέδου Αφαίρεσης

Θα διαφοροποιήσουμε τα επίπεδα αφαίρεσης με βάση τη **χρονική λεπτομέρεια (time – granularity)**. Μια μικρότερη χρονική λεπτομέρεια τυπικά σημαίνει ότι οι δραστηριότητες εκφράζονται σε ένα μεγαλύτερο αριθμό από (συνήθως μικρά) χρονικά βήματα. Υπάρχουν πέντε επίπεδα αφαίρεσης που χρησιμοποιούνται συνήθως από μηχανικούς υπολογιστών για το σχεδιασμό ηλεκτρονικών συστημάτων υλικού/λογισμικού. Ξεκινώντας από το χαμηλότερο επίπεδο αφαίρεσης, απαριθμούμε τα πέντε επίπεδα.

2 Κεφάλαιο - Μοντελοποίηση και Μετασχηματισμός Ροής Δεδομένων

Εισαγωγή γράφων ροής δεδομένων

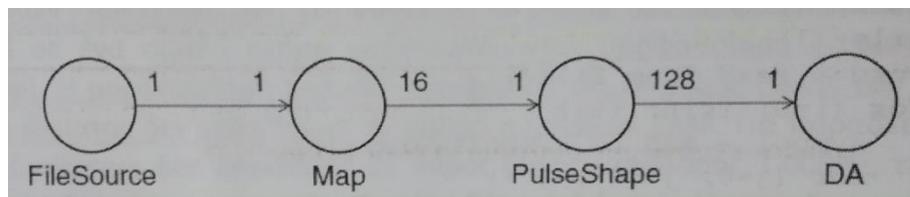
Από τη φύση του, το υλικό είναι παράλληλο και το λογισμικό είναι ακολουθιακό. Ως αποτέλεσμα, τα μοντέλα λογισμικού (προγράμματα C) δεν είναι και τόσο κατάλληλα για την απόδοση υλοποιήσεων υλικού, και αντίστροφα, τα μοντέλα υλικού (προγράμματα RTL) δεν αποτελούν καλή θεώρηση για την περιγραφή του λογισμικού.

Σημείωση: Σημασία Μπλοκ διαγραμμάτων

Οι μηχανικοί επεξεργασίας σημάτων περιγράφουν πολύπλοκα συστήματα, όπως τα ψηφιακά ραδιόφωνα και τις μονάδες επεξεργασίας ραντάρ, χρησιμοποιώντας μπλοκ διαγράμματα (block diagrams). Ένα μπλοκ διάγραμμα είναι μια υψηλού επιπέδου αναπαράσταση του συστήματος στόχου ως μια συλλογή μικρότερων λειτουργιών. Ένα μπλοκ διάγραμμα δεν καθορίζει εάν ένα εξάρτημα θα πρέπει να υλοποιηθεί ως υλικό ή λογισμικό, αλλά εκφράζει μόνο τις λειτουργίες που εκτελούνται σε σήματα δεδομένων. Μας ενδιαφέρουν ειδικά τα ψηφιακά συστήματα επεξεργασίας σημάτων.

Σημείωση: Προτεινόμενα στυλ μοντελοποίησης

Γενικά, η κατασκευή μιας ακολουθιακής υλοποίησης για ένα παράλληλο μοντέλο (όπως ένα μπλοκ διάγραμμα) είναι πολύ ευκολότερη από το αντίθετο – την κατασκευή μιας παράλληλης υλοποίησης από ένα ακόλουθο μοντέλο. Συνεπώς, προτιμούμε τα στυλ μοντελοποίησης που επιτρέπουν σε έναν σχεδιαστή να εκφράσει παράλληλες δραστηριότητες στο υψηλότερο επίπεδο αφαίρεσης.



Εικόνα 8 - Μοντέλο ροής δεδομένων για το Σύστημα διαμόρφωσης πλάτους παλμών

Σημείωση: Ροή δεδομένων και ουρές και δρώντες

Σε αυτό το κεφάλαιο θα συζητήσουμε μια τεχνική μοντελοποίησης, που ονομάζεται Ροή Δεδομένων (Data Flow), η οποία επιτυγχάνει το στόχο ενός παράλληλου μοντέλου. Τα μοντέλα Ροής Δεδομένων μοιάζουν πολύ με διαγράμματα μπλοκ. Το σύστημα PAM-4, ως μοντέλο Ροής Δεδομένων, παρουσιάζεται στην Εικόνα 8. Στην περίπτωση αυτή, οι διάφορες λειτουργίες του συστήματος απεικονίζονται ως μεμονωμένες οντότητες ή δρώντες (actors) όπως οι FileSource, Map, PulseShape και DA. Αυτοί οι δρώντες συνδέονται μέσω καναλιών επικοινωνίας ή ουρών (queues). Οι είσοδοι και οι έξοδοι κάθε δρώντα σημειώνονται με το σχετικό ρυθμό των επικοινωνιών. Για παράδειγμα, ο Map παράγει 16 δείγματα για κάθε δείγμα εισόδου. Κάθε δρών είναι μια ανεξάρτητη μονάδα, που ελέγχει συνεχώς τις εισόδους της για τη διαθεσιμότητα των δεδομένων. Μόλις εμφανιστούν δεδομένα, κάθε δρών θα υπολογίσει την αντίστοιχη έξοδο, και θα περάσει το αποτέλεσμα στον επόμενο δρώντα στη σειρά.

Σημείωση: Χαρακτηριστικά των μοντέλων ροής δεδομένων

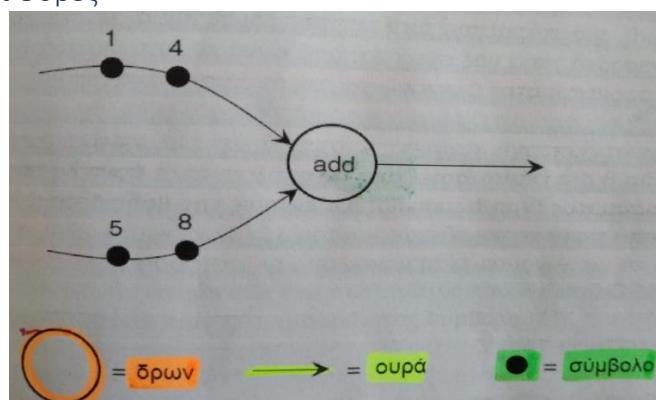
- Ένα δυνατό σημείο των μοντέλων Ροής Δεδομένων, και ο κύριος λόγος για τον οποίο οι μηχανικοί επεξεργασίας σημάτων αγαπούν να τα χρησιμοποιούν, είναι ότι **ένα μοντέλο Ροής Δεδομένων είναι ένα ταυτόχρονο μοντέλο**. Πράγματι, οι δρώντες στην Εικόνα 8 λειτουργούν και εκτελούνται ως ξεχωριστές ταυτόχρονες οντότητες. Ένα ταυτόχρονο μοντέλο μπορεί να απεικονιστεί σε μια παράλληλη ή μια ακολουθιακή υλοποίηση και έτσι μπορούν να στοχεύσουν τόσο σε υλικό όσο και σε λογισμικό.
- **Τα μοντέλα Ροής Δεδομένων είναι κατανεμημένα και δεν υπάρχει ανάγκη για κεντρικό ελεγκτή ή "οδηγό (conductor) στο σύστημα** για να κρατάει τα επιμέρους εξαρτήματα του συστήματος σε βηματισμό. Στην Εικόνα 8, δεν υπάρχει κεντρικός ελεγκτής που να λέει στους δρώντες πότε να λειτουργούν. Κάθε δρών μπορεί να αποφασίσει για τον εαυτό του πότε ήρθε η ώρα να εργαστεί.
- **Τα μοντέλα Ροής Δεδομένων είναι αρθρωτά.** Μπορούμε να αναπτύξουμε μια βιβλιοθήκη σχεδιασμού εξαρτημάτων ροής δεδομένων και στη συνέχει να χρησιμοποιήσουμε αυτή τη βιβλιοθήκη με τρόπο plug-and-play για να κατασκευάσουμε συστήματα ροής δεδομένων.
- **Τα μοντέλα Ροής Δεδομένων είναι δυνατό να αναλυθούν.** Ορισμένες ιδιότητες, όπως η ικανότητά τους να αποφευχθεί το αδιέξοδο, μπορεί να προσδιοριστεί άμεσα από το σχεδιασμό. Το αδιέξοδο (deadlock) είναι μια κατάσταση που αντιμετωπίζεται μερικές φορές από συστήματα υπολογιστών πραγματικού χρόνου, στην οποία το σύστημα δεν ανταποκρίνεται. Η δυνατότητα ανάλυσης της συμπεριφοράς των μοντέλων σε επίπεδο αφαίρεσης αποτελεί ένα σημαντικό πλεονέκτημα. Τα προγράμματα C, για παράδειγμα, δεν προσφέρουν μια τέτοια ευκολία, στην πραγματικότητα, ένας σχεδιαστής C συνήθως αποφασίζει την ορθότητα ενός προγράμματος C εκτελώντας το, αντί να το αναλύει!

Η Ροή Δεδομένων υπάρχει για ένα εκπληκτικό μεγάλο χρονικό διάστημα, όπως έχει σε μεγάλο βαθμό επισκιαστεί από το υπολογιστικό μοντέλο του αποθηκευμένου προγράμματος (Von Neumann).

Σημείωση: Σημασία ροής δεδομένων

Σήμερα, η ροή δεδομένων παραμένει πολύ δημοφιλής για την περιγραφή συστημάτων επεξεργασίας σημάτων. Για παράδειγμα, εμπορικά εργαλεία όπως Simulink®; βασίζονται στις ιδέες της ροής δεδομένων.

2.1.1 Σύμβολα, Δρώντες και Ουρές

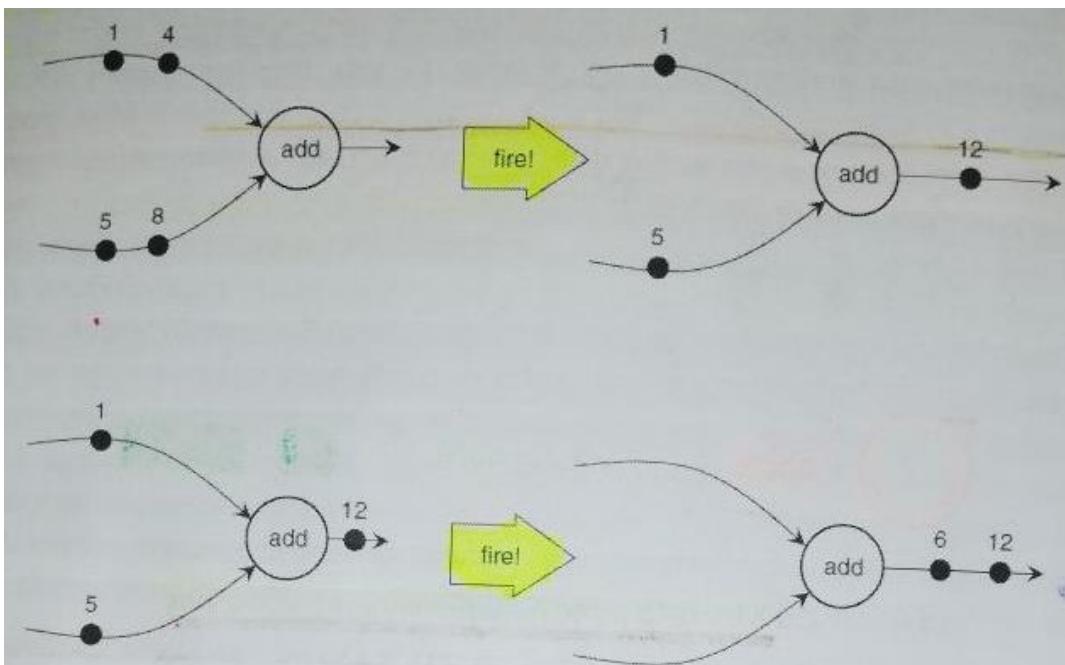


Εικόνα 9 - Μοντέλο ροής δεδομένων μιας άθροισης

Η Εικόνα 9 δείχνει το μοντέλο ροής δεδομένων μιας απλής άθροισης. Αυτό το μοντέλο περιέχει τα ακόλουθα στοιχεία.

Σημείωση: Στοιχεία μοντέλου ροής δεδομένων

- Οι δρώντες (actors) περιέχουν τις πραγματικές λειτουργίες. Οι δρώντες έχουν μια οριθετημένη συμπεριφορά (που σημαίνει ότι έχουν μια καθορισμένη αρχή και τέλος) και επαναλαμβάνουν αυτή τη συμπεριφορά από την αρχή μέχρι την ολοκλήρωση. Μία τέτοια επανάληψη ονομάζεται πυροδότηση δρώντα. Στο παρόντα παράδειγμα, κάθε πυροδότηση ενός δρώντα θα πραγματοποιούσε μία μεμονωμένη πρόσθεση.
- Τα σύμβολα (tokens) μεταφέρουν πληροφορίες από έναν δρώντα στον άλλο. Ένα σύμβολο έχει μια τιμή, όπως τα '1', '4', '5' και '8' στην Εικόνα 9.
- Οι ουρές (queues) είναι μονοκατευθυντήριοι σύνδεσμοι επικοινωνίας που μεταφέρουν σύμβολα από τον έναν δρώντα στον άλλο. Οι ουρές Ροής Δεδομένων έχουν άπειρη ποσότητα αποθηκευτικού χώρου, έτσι ώστε τα σύμβολα να μην χάνονται ποτέ σε μια ουρά. Οι ουρές Ροής Δεδομένων είναι first-in first – out. Στην Εικόνα 10, υπάρχουν δύο σύμβολα στην επάνω ουρά, ένα με τιμή '1' και ένα με τιμή '4'. Το σύμβολο '4' μπήκε πρώτο στην ουρά, και το σύμβολο '1' μπήκε μετά από αυτό. Όταν ο δρών 'add' θα διαβάσει ένα σύμβολο από αυτή την ουρά, ο δρών θα διαβάσει πρώτα το σύμβολο με την τιμή '4' και στη συνέχεια το σύμβολο με τιμή '1'.



Εικόνα 10 - Τα αποτελέσματα δύο πυροδοτήσεων του δρώντα άθροισης, με καθένα να έχει ως αποτέλεσμα διαφορετική σήμανση

Σημείωση: Πυροδότηση δρώντα

Όταν εκτελείται ένα μοντέλο ροής δεδομένων, οι δρώντες θα διαβάζουν σύμβολα από τις ουρές εισόδου τους. Θα διαβάσουν την τιμή αυτών των συμβόλων, θα υπολογίσουμε την αντίστοιχη τιμή εξόδου και θα δημιουργήσουν νέα σύμβολα στις ουρές εξόδου τους. Κάθε μεμονωμένη εκτέλεση ενός δρώντα ονομάζεται πυροδότηση

(firing) αυτού του δρώντα. Η εκτέλεση της ροής δεδομένων εκφράζεται στη συνέχεια ως μια ακολουθία (πιθανώς ταυτόχρονων) πυροδοτήσεων δρώντων.

Σημείωση:

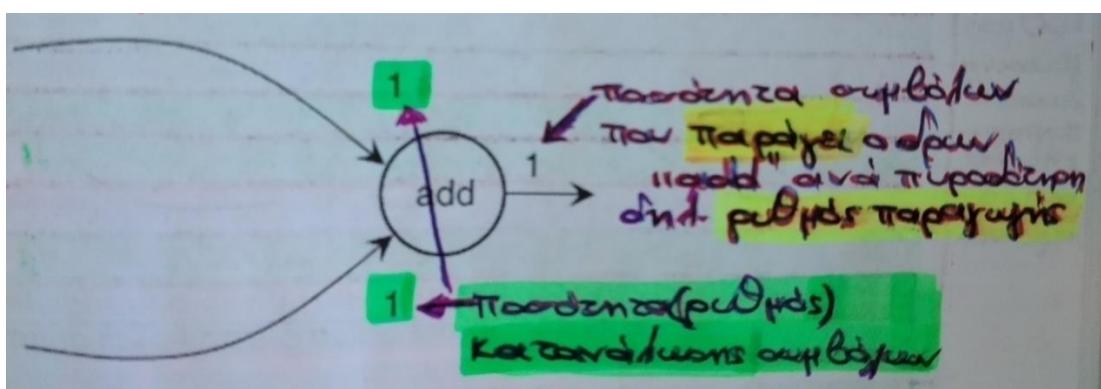
Απουσία χρονική σήμανσης (Untimed)

Απουσία χρονική σήμανσης (Untimed) δεν σημαίνει μηδενικό χρόνο, σημαίνει μόνο ότι ο χρόνος δεν έχει σημασία για τα μοντέλα ροής δεδομένων. Πράγματι, στη ροή δεδομένων, η εκτέλεση καθοδηγείται από την παρουσία δεδομένων, και όχι από έναν μετρητή προγράμματος ή από ένα σήμα ρολογιού. Ένας δρών δεν πρόκειται ποτέ να πυροδοτηθεί εάν δεν υπάρχουν δεδομένα εισόδου, αλλά αντίθετα θα περιμένει μέχρι να υπάρχουν διαθέσιμα επαρκή δεδομένα στις εισόδους του.

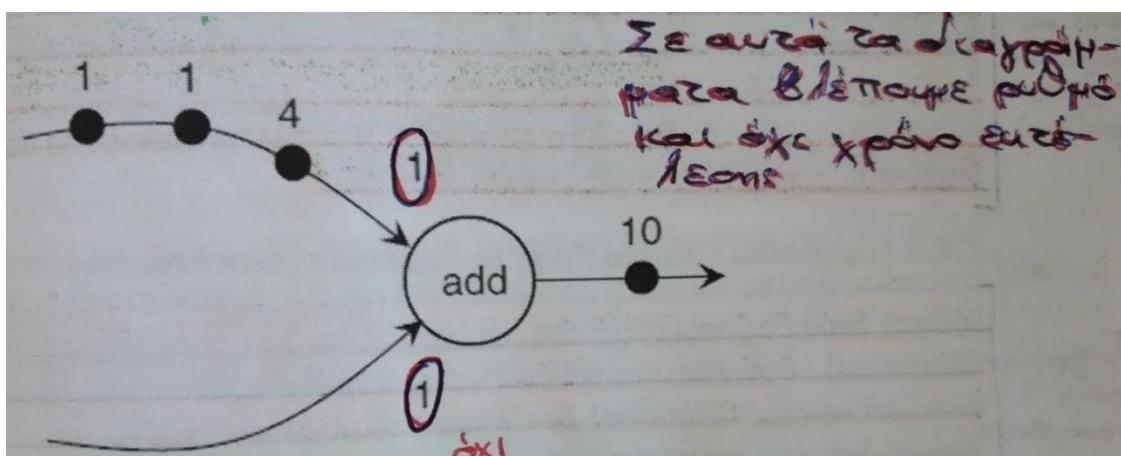
Σημείωση:

Σήμανση σε μοντέλο ροής δεδομένων

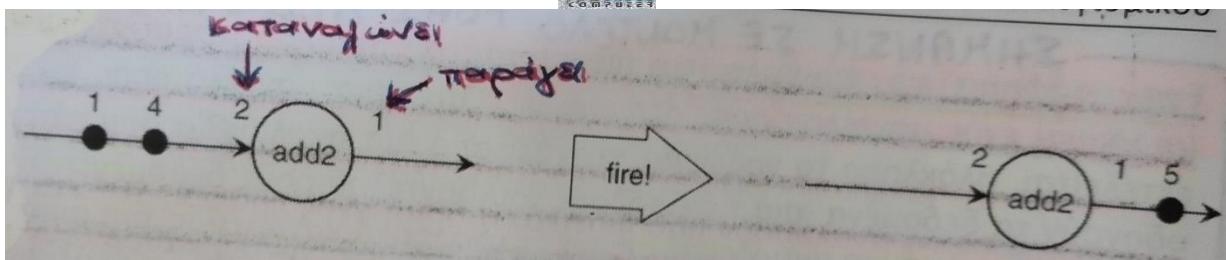
Ένα γράφημα με σύμβολα κατανεμημένα σε ουρές ονομάζεται σήμανση (marking) ενός μοντέλου ροής δεδομένων. Όταν ένα μοντέλο ροής δεδομένων εκτελείται, ολόκληρο το γράφημα περνάει από μια ακολουθία σημάνσεων που οδηγούν τα δεδομένα από τις εισόδους του μοντέλου ροής δεδομένων στις εξόδου. Κάθε σήμανση αντιστοιχεί σε μια διαφορετική κατάσταση του συστήματος και η εκτέλεση ενός μοντέλου ροής δεδομένων ορίζεται από μια ακολουθία σημάνσεων. Σε έναν εξωτερικό παρατηρητή, η σήμανση (δηλαδή η κατανομή των συμβόλων στις ουρές) είναι η μόνη παρατηρήσιμη κατάσταση στο σύστημα. Αυτή η παρατήρηση είναι κρίσιμη! Υποδηλώνει ότι ένας δρών δεν μπορεί να χρησιμοποιήσει εσωτερικές μεταβλητές κατάστασης που θα επηρέαζαν την εκτέλεση του συστήματος, και κατά συνέπεια και την ακολουθία σημάνσεων.



Εικόνα 11 - Δρών ροής δεδομένων με ρυθμούς παραγωγής/κατανάλωσης



Εικόνα 12 - Μπορεί αυτό το μοντέλο να κάνει χρήσιμους υπολογισμούς;



Εικόνα 13 - Παράδειγμα μοντέλου ροής δεδομένων πολλαπλού ρυθμού

2.1.2 Ρυθμοί Πυροδότησης, Κανόνες Πυροδότησης και Χρονοδιαγράμματα

Πότε πρέπει να πυροδοτήσει ένας δρών; Ο κανόνας πυροδότησης (firing – rule) περιγράφει τις αναγκαίες και ικανές συνθήκες για να πυροδοτήσει ένας δρών. Απλοί δρώντες, όπως ο δρών άθροισης, μπορούν να πυροδοτήσουν όταν υπάρχει ένα σύμβολο σε κάθε ουρά εισόδου του.

Σημείωση: Ρυθμός κατανάλωσης και παραγωγής σε απλό δρώντα

Ένας κανόνας πυροδότησης περιλαμβάνει συνεπώς τον έλεγχο του αριθμού των συμβόλων σε κάθε ουρά εισόδου του. Ο απαιτούμενος αριθμός συμβόλων μπορεί να υποσημειωθεί στην είσοδο του δρώντα. Ομοίως, η ποσότητα των συμβόλων που παράγει ο δρών ανά πυροδότηση μπορεί αν υποσημειωθεί στην έξοδο του δρώντα. Αυτοί οι αριθμοί ονομάζονται ρυθμός κατανάλωσης συμβόλων (στις εισόδους του δρώντα) και ρυθμός παραγωγής συμβόλων (στις εξόδους του δρώντα). Οι ρυθμοί παραγωγής/κατανάλωσης του δρώντα άθροισης, θα μπορούσαν να γραφούν όπως φαίνεται στις Εικόνες 11 ή 12.

Σημείωση: Μοντέλα ροής δεδομένων πολλαπλού ρυθμού

Οι δρώντες Ροής Δεδομένων ενδέχεται να καταναλώνουν ή να παράγουν περισσότερα από ένα σύμβολα ανά πυροδότηση δρώντα. Αυτά τα μοντέλα ονομάζονται μοντέλα ροής δεδομένων πολλαπλού ρυθμού. Για παράδειγμα, ο δρών στην Εικόνα 13 έχει ρυθμό κατανάλωσης 2 και ρυθμό παραγωγής 1. Θα καταναλώνει δύο σύμβολα ανά πυροδότηση από την είσοδο του, θα τα προσθέτει και θα παράγει ένα σύμβολο εξόδου ως αποτέλεσμα.

2.1.3 Σύγχρονοι Γράφοι Ροής Δεδομένων (synchronous data flow –SDF)

Σημείωση: Σύγχρονη ροή δεδομένων

Όταν ο αριθμός των συμβόλων που καταναλώνονται/παράγονται ανά πυροδότηση δρώντα είναι μια καθορισμένη και σταθερή τιμή, η κατηγορία συστημάτων που προκύπτει ονομάζεται σύγχρονη ροή δεδομένων (synchronous data flow) ή SDF. Ο όρος σύγχρονος σημαίνει ότι οι ρυθμοί παραγωγής και κατανάλωσης συμβόλων είναι γνωστοί, καθορισμένοι αριθμοί. Η σημασιολογία SDF δεν είναι καθολική.

Σημείωση: Μειονέκτημα γράφων SDF

Για παράδειγμα, δεν μπορεί κάθε πρόγραμμα C να μεταφραστεί σε ισοδύναμο γράφο SDF. Η εκτέλεση εξαρτώμενη από δεδομένα δεν μπορεί να εκφραστεί ως γράφος SDF: η εκτέλεση εξαρτώμενη από δεδομένα υποδηλώνει ότι η πυροδότηση δρώντα ορίζεται όχι μόνο από την παρουσία των συμβόλων, αλλά και από την τιμή τους.



Σημείωση: Μαθηματική ανάλυση των ιδιοτήτων των γράφων SDF

Παρόλα αυτά, οι γράφοι SDF έχουν ένα σημαντικό πλεονέκτημα: οι ιδιότητες τους μπορούν να αναλυθούν μαθηματικά. Η δομή ενός γράφου SDF και οι ρυθμοί παραγωγής / κατανάλωσης των συμβόλων στους δρώντες καθορίζουν εάν είναι πιθανό, είναι εφικτό χρονοδιάγραμμα πυροδότησης των δρώντων.

2.1.4 Οι γράφοι SDF είναι καθορισμένοι (Determinate)

Σημείωση: Η εκτέλεση SDF είναι καθορισμένη

Υποθέτοντας ότι κάθε δρών υλοποιεί μια καθορισμένη συνάρτηση, τότε ολόκληρη η εκτέλεση SDF είναι καθορισμένη. Αυτό σημαίνει ότι τα αποτελέσματα, που υπολογίζονται χρησιμοποιώντας έναν γράφο SDF, θα είναι πάντα τα ίδια. Αυτή η ιδιότητα ισχύει ανεξάρτητα από τη σειρά (ή το χρονοδιάγραμμα) πυροδότησης των δρώντων.

Σημείωση: Απόδειξη

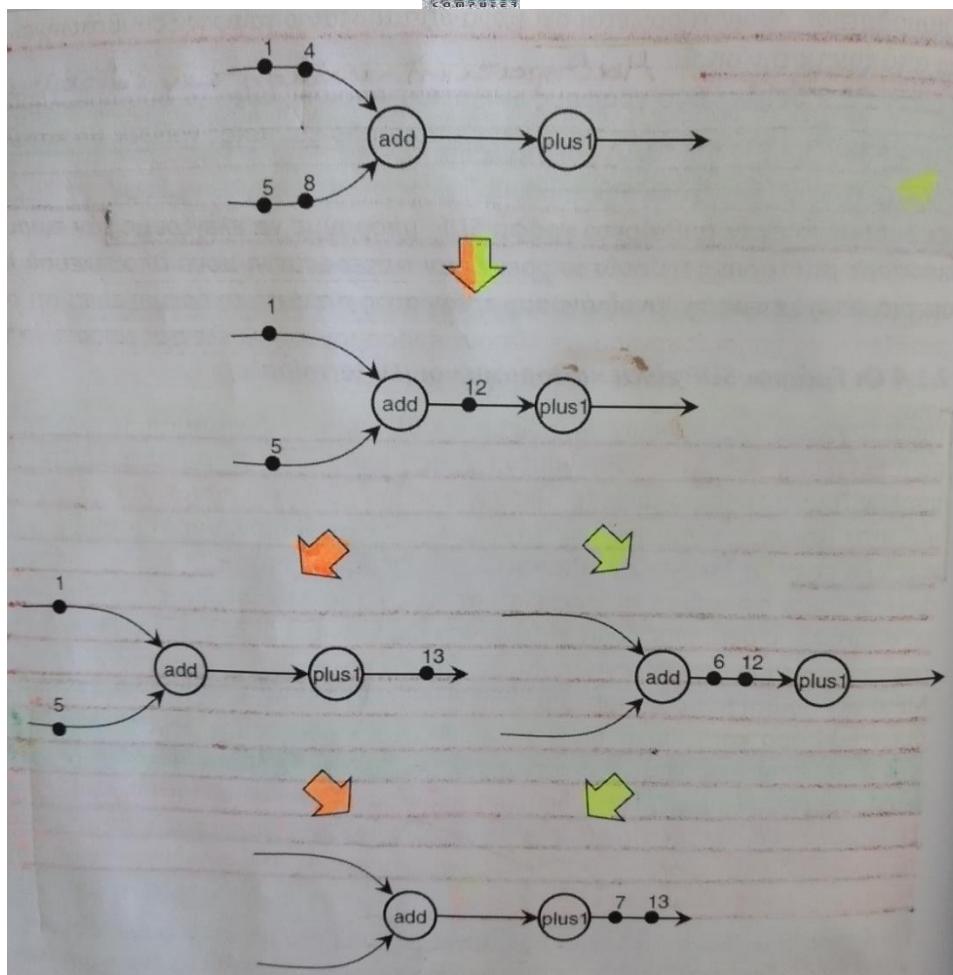
Ο γράφος περιέχει δρώντες με μοναδιαίους ρυθμούς παραγωγής / κατανάλωσης. Ένας δρών προσθέτει σύμβολα, ο δεύτερος δρών αυξάνει την τιμή των συμβόλων. Καθώς αρχίζουμε την πυροδότηση των δρώντων, τα σύμβολα μεταφέρονται μέσω του γράφου. Μετά την πρώτη πυροδότηση, μια ενδιαφέρουσα κατάσταση εμφανίζεται: τόσο ο δρών add όσο και ο δρών plus1 μπορούν να πυροδοτήσουν. Μια πρώτη περίπτωση, η οποία φαίνεται στα αριστερά της Εικόνας 14, υποθέτει ότι ο δρών plus1 πυροδοτεί πρώτα. Μια δεύτερη περίπτωση, που φαίνεται στα δεξιά της Εικόνας 14, υποθέτει ότι ο δρών add πυροδοτεί πρώτα. Ωστόσο, ανεξάρτητα από το ποια διαδρομή ακολουθείται, η σήμανση γράφου τελικά συγκλίνει στο αποτέλεσμα που εμφανίζεται στο κάτω μέρος.

Σημείωση: Καθορισμένη λειτουργία SDF

Δεδομένου ότι οι γράφοι SDF είναι καθορισμένοι, δεν έχει σημασία ποιος επεξεργαστής εκτελεί ποιον δρώντα: τα αποτελέσματα θα είναι πάντα τα ίδια. Με άλλα λόγια, ένα σύστημα SDF θα λειτουργεί όπως έχει προδιαγραφεί, ανεξάρτητα από την τεχνολογία που χρησιμοποιείται για την υλοποίησή του.

Σημείωση: Προϋπόθεση

Φυσικά, οι δρώντες πρέπει να είναι πλήρως και ορθά υλοποιημένοι, να πυροδοτούνται με βάση τον κανόνα και ως σύνολο.

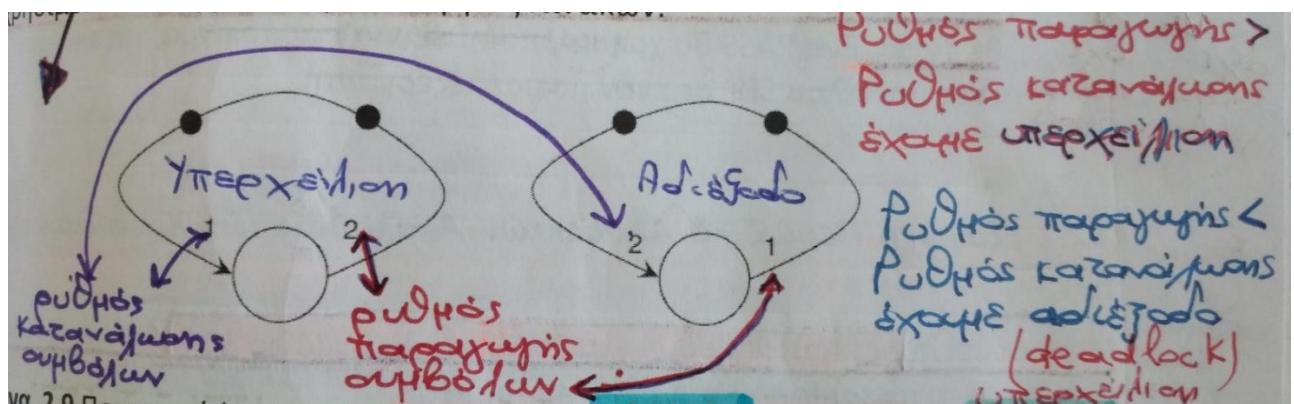


Εικόνα 14 - Οι γράφοι SDF είναι καθορισμένοι

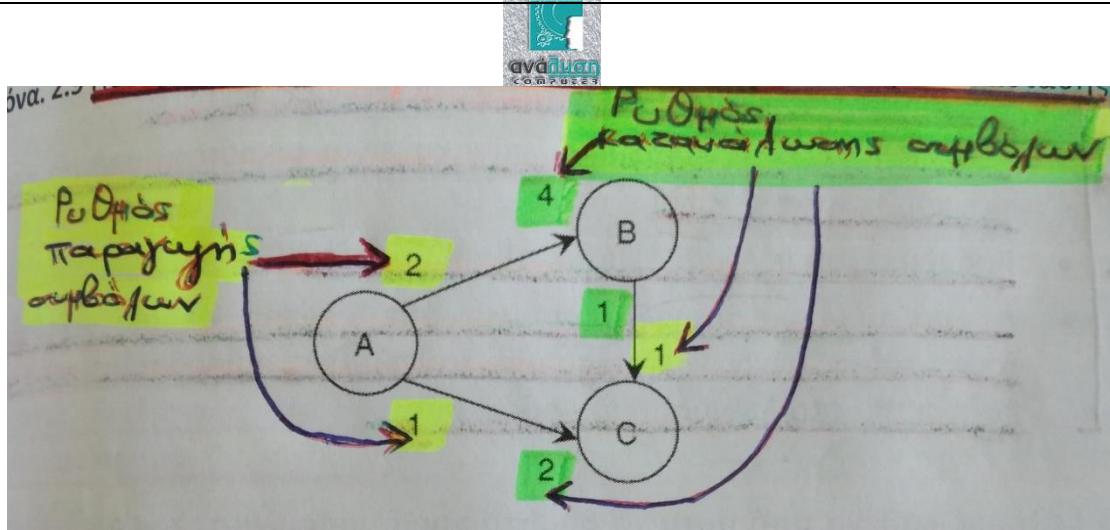
Ανάλυση Σύγχρονων Γράφων Ροής Δεδομένων

Ένα αποδεκτό χρονοδιάγραμμα (schedule) για ένα γράφο SDF είναι αυτό που θα μπορεί να τρέχει για πάντα χωρίς να προκαλέσει αδιέξοδο και χωρίς να υπερχειλίσει οποιαδήποτε από τις ουρές επικοινωνίας. Ο όρος απεριόριστη εκτέλεση (unbounded execution) χρησιμοποιείται για να υποδείξει ότι ένα μοντέλο λειτουργεί πάντα. Ο όρος πεπερασμένος απομονωτής (bounded buffer) χρησιμοποιείται για να υποδείξει ότι καμία ουρά επικοινωνίας δεν χρειάζεται άπειρο βάθος. Μια κατάσταση αδιεξόδου εμφανίζεται όταν ο γράφος SDF καταλήγει σε μια σήμανση στην οποία δεν μπορεί πλέον να πυροδοτηθεί κανένας δρών.

Η Εικόνα 15 δείχνει δύο γράφους SDF στους οποίους αυτά τα δύο προβλήματα είναι εμφανή. Ποιος γράφος θα έλθει σε αδιέξοδο, και ποιος γράφος θα καταλήξει με άπειρη ποσότητα συμβόλων;



Εικόνα 15 - Ποιος γράφος SDF θα οδηγήθει σε αδιέξοδο και ποιος είναι ασταθής;



Εικόνα 16 - Παράδειγμα γράφου SDF για κατασκευή PASS

Σημείωση: Περιοδικά αποδεκτοί γράφοι – PASS

Θα μελετήσουμε τη μέθοδο του Lee για τη δημιουργία των λεγόμενων Περιοδικά Αποδεκτών Χρονοδιαγραμμάτων (Periodic Admissible Schedules – PASS). Ένα PASS ορίζεται ως εξής:

Ένα χρονοδιάγραμμα είναι η σειρά με την οποία πρέπει να πυροδοτήσουν οι δρώντες.

Σημείωση: Χρονοδιάγραμμα

Ένα αποδεκτό χρονοδιάγραμμα είναι μια σειρά πυροδοτήσεων που δεν θα προκαλέσει αδιέξοδο και η οποία θα παράγει πεπερασμένη αποθήκευση.

Σημείωση: Αποδεκτό χρονοδιάγραμμα → Περιοδικά Αποδεκτό Χρονοδιάγραμμα

Τέλος, ένα περιοδικά αποδεκτό χρονοδιάγραμμα, είναι ένα χρονοδιάγραμμα που είναι κατάλληλο για απεριόριστη εκτέλεση, επειδή είναι περιοδικό (δηλαδή, μετά από κάποιο χρονικό διάστημα, θα επαναληφθεί η ίδια ακολουθία σήμανσης). Θα εξετάσουμε στη συνέχεια τα Περιοδικά Αποδεκτά Ακολουθιακά Χρονοδιαγράμματα ή en συντομία Pass, (Periodic Admissible Sequential Schedules). Ένα τέτοιο ακολουθιακό χρονοδιάγραμμα απαιτεί μόνο ένας μεμονωμένος δρών να πυροδοτείται κάθε φορά.

2.1.5 Εξαγωγή Περιοδικά Αποδεκτών Ακολουθιακών Χρονοδιαγραμμάτων

Μπορούμε να δημιουργήσουμε ένα PASS για ένα γράφο SDF (και να εξετάσουμε εάν υπάρχει) με τα παρακάτω τέσσερα βήματα.

Σημείωση: Δημιουργία PASS για γράφο SDF

- Δημιουργούμε τον πίνακα τοπολογίας G του γράφου SDF.
- Επαληθεύουμε ότι η τάξη του πίνακα είναι μικρότερη από τον αριθμό των κόμβων στο γράφο. $\text{rank}(G) < \text{αριθμός κόμβων}$
- Προσδιορίζουμε ένα διάνυσμα πυροδότησης



- Δοκιμάζουμε να πυροδοτήσουμε κάθε δρώντα με μια μεθοδολογία round robin, μέχρι να φτάσουμε στο πλήθος πυροδοτήσεων που έχει καθοριστεί στο διάνυσμα πυροδότησης.

Βήμα 1

Δημιουργήστε ένα πίνακα τοπολογίας για αυτόν το γράφο. Αυτός ο πίνακας τοπολογίας έχει τόσες γραμμές, όπως οι ακμές του γράφου (ουρές FIFO) και τόσες στήλες, όσες οι κόμβοι του γράφου. Το στοιχείο (i, j) αυτού του πίνακα θα είναι θετικό εάν ο κόμβος j παράγει σύμβολα στην ακμή i του γράφου. Το στοιχείο (i, j) θα είναι αρνητικό εάν ο κόμβος j καταναλώνει σύμβολα από την ακμή i του γράφου. Για τον παραπάνω γράφο, μπορούμε έτσι να δημιουργήσουμε τον παρακάτω πίνακα τοπολογίας. Σημειώστε ότι ο G δεν χρειάζεται να είναι τετραγωνικός – εξαρτάται από το πλήθος των ουρών και των δρώντων του συστήματος.

$$G = \begin{bmatrix} 2 & -4 & 0 \\ 1 & 0 & -2 \\ 0 & 1 & -1 \end{bmatrix} \leftarrow \begin{array}{l} \text{edge } (A, B) \\ \text{edge } (A, C) \\ \text{edge } (B, C) \end{array} \quad (2.1)$$

Βήμα 2

Σημείωση: Προϋπόθεση ύπαρξης ενός PASS

Η προϋπόθεση για να υπάρξει ένα PASS είναι ότι η τάξη του G πρέπει να είναι μικρότερη κατά ένα από τον αριθμό των κόμβων στο γράφο. Η τάξη ενός πίνακα είναι ο αριθμός των ανεξάρτητων εξισώσεων στον G . Δεδομένου ότι υπάρχουν τρεις κόμβοι στον γράφο και η τάξη του G είναι 2, ένα PASS είναι πιθανό.

Το βήμα 2 επαληθεύει ότι δεν μπορούν να συσσωρευτούν σύμβολα σε καμία από τις ακμές του γράφου. Μπορούμε να βρούμε τον αριθμό των συμβόλων που προκύπτει, επιλέγοντας ένα διάνυσμα πυροδότησης και κάνοντας έναν πολλαπλασιασμό πινάκων. Για παράδειγμα, υποθέστε ότι ο A πυροδοτείται δύο φορές, και οι B και C πυροδοτούνται μηδέν φορές. Αυτό δίνει το ακόλουθο διάνυσμα πυροδότησης:

$$q = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} \quad (2.2)$$

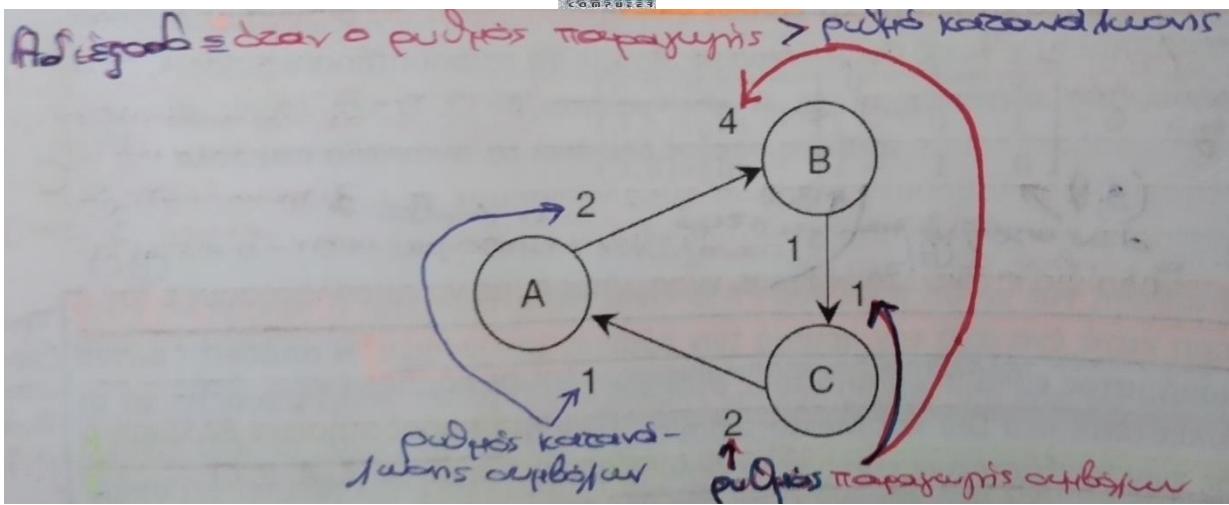
Τα υπόλοιπα σύμβολα που απομένουν στις ακμές μετά από αυτές τις πυροδοτήσεις είναι δύο σύμβολα στην ακμή (A, B) και ένα σύμβολο στην ακμή (A, C) :

$$b = Gq = \begin{bmatrix} 2 & -4 & 0 \\ 1 & 0 & -2 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} \quad (2.3)$$

Βήμα 3

Προσδιορίστε ένα περιοδικό διάνυσμα πυροδότησης. Το διάνυσμα πυροδότησης που αναφέρθηκε παραπάνω δεν είναι καλή επιλογή για να αποκτήσετε ένα PASS κάθε φορά που εκτελείται το συγκεκριμένο διάνυσμα πυροδότησης, προσθέτει τρία σύμβολα στο σύστημα. Αντίθετα, ενδιαφερόμαστε για διανύσματα πυροδοτήσεων που δεν αφήνουν πρόσθετα σύμβολα στις ουρές. Με άλλα λόγια, το αποτέλεσμα πρέπει να ισούται με το μηδενικό διάνυσμα.

$$Cq_{PASS} = 0 \quad (2.4)$$



Εικόνα 17 - Ένας γράφος με αδιέξοδο

Σημείωση: Πώς γίνεται η επιλογή του διανύσματος q_{PASS}

Δεδομένου ότι η τάξη του G είναι μικρότερη από τον αριθμό των κόμβων, το σύστημα αυτό έχει άπειρο αριθμό λύσεων. Διαισθητικά, αυτό πρέπει να περιμένουμε. Εάν υποθέσουμε ότι ένα διάνυσμα πυροδοτήσεων (a, b, c) θα είναι μια λύση που μπορεί να δώσει ένα PASS, τότε επίσης το (2a, 2b, 2c) θα είναι μια λύση, και ομοίως είναι το (3a, 3b, 3c) κ.ο.κ. Απλά πρέπει να βρείτε το πιο απλό. Μια πιθανή λύση που δίνει ένα PASS είναι να πυροδοτήσει ο Α δύο φορές και οι Β και Κ ο καθένας από μια φορά:

$$q_{PASS} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} \quad (2.5)$$

Η ύπαρξη ενός διανύσματος πυροδοτήσεων PASS δεν εγγυάται ότι θα υπάρχει επίσης ένα PASS. Για παράδειγμα, απλώς αλλάζοντας την κατεύθυνση της ακμής (A, C) θα βρείτε το ίδιο q_{PASS} , αλλά ο γράφος που προκύπτει θα είναι σε αδιέξοδο, καθώς όλοι οι κόμβοι περιμένουν ο ένας τον άλλον. Επομένως, υπάρχει ακόμα ένα τέταρτο βήμα: η κατασκευή ενός έγκυρου PASS.

Βήμα 4

Σημείωση: Πότε ένας κόμβος είναι σε αδιέξοδο

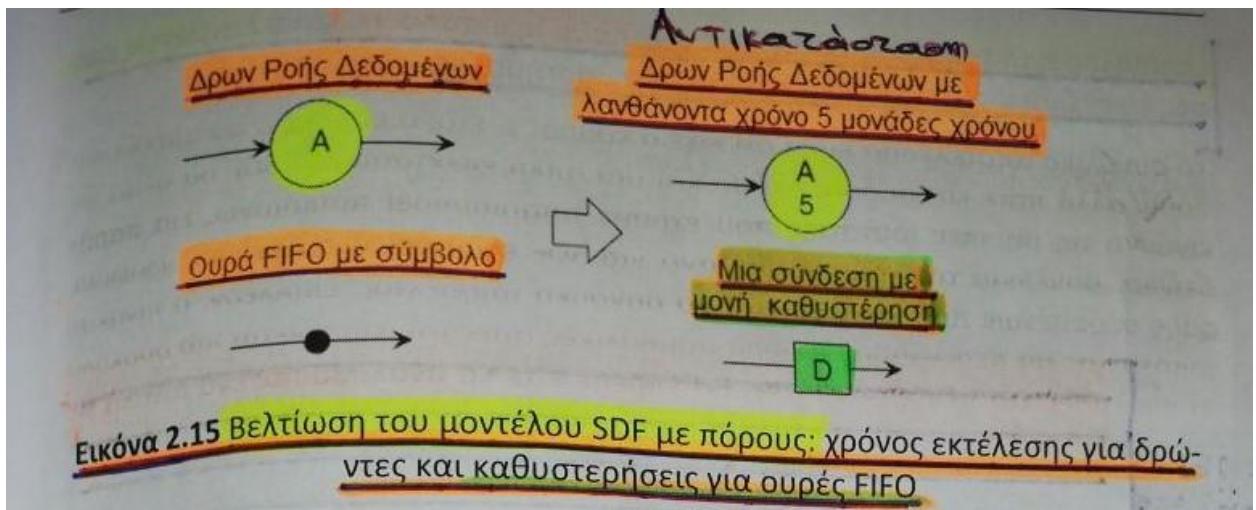
Κατασκευάστε ένα PASS. Προσπαθούμε τώρα να πυροδοτήσουμε κάθε κόμβο τόσες φορές όσες ο αριθμός των φορών που καθορίζεται στο q_{PASS} . Κάθε κόμβος που έχει τον επαρκή αριθμό από σύμβολα στις ουρές εισόδου του θα πυροδοτήσει όταν γίνει δοκιμή. Αν βρούμε ότι δεν μπορούμε να πυροδοτήσουμε επιπλέον κόμβους, και ότι ο αριθμός των πυροδοτήσεων κάθε κόμβου είναι μικρότερος από τον αριθμό που προσδιορίζεται στο q_{PASS} , ο γράφος που προκύπτει θα βρίσκεται σε αδιέξοδο.

Εφαρμόζουμε αυτό στον αρχικό γράφο χρησιμοποιώντας το διάνυσμα πυροδότησης ($A = 2$, $B = 1$, $C = 1$). Πρώτα προσπαθούμε να πυροδοτήσουμε το Α, το οποίο αφήνει δύο σύμβολα στο (A, B) και ένα στο (A, C). Στη συνέχεια, προσπαθούμε να πυροδοτήσουμε το B – 0 ο οποίος δεν έχει τα αναγκαία σύμβολα για να πυροδοτήσει.

Προσθέτοντας χρόνο και πόρους

Σημείωση: Για ποιο λόγο πετυχαίνεται βελτίωση του μοντέλου SDF

Μπορούμε να χρησιμοποιήσουμε το μοντέλο ροής δεδομένων για να κάνουμε ανάλυση απόδοσης. Με την εισαγωγή ενός μοντέλου ελαχίστων πόρων (χρόνος εκτέλεσης δρώντα και πεπερασμένες ουρές FIFO), μπορούμε να αναλύσουμε την απόδοση του συστήματος σε ένα γράφο ροής δεδομένων. Επιπλέον, μπορούμε να αναλύσουμε το αποτέλεσμα των μετασχηματισμών για βελτίωση της απόδοσης στον γράφο ροής δεδομένων.



Εικόνα 18 –Βελτίωση του μοντέλου SDF με πόρους: χρόνος εκτέλεσης για δρώντες και καθυστερήσεις για ουρές FIFO

Για βελτίωση του μοντέλου SDF μπορούμε να προσθέσουμε πόρους, που είναι i) λανθάνοντας χρόνος στους δρώντες και ii) καθυστερήσεις ($D = \text{delay}$) στις ουρές.

2.1.6 Περιορισμοί πραγματικού χρόνου και Ρυθμός Δειγματοληψίας Εισόδου / Εξόδου

Σημείωση: Ορισμός γράφου ροής δεδομένων

Ένας γράφος ροής δεδομένων είναι ένα μοντέλο για μια επαναλαμβανόμενη δραστηριότητα.

Σημείωση: Ρυθμός μετάδοσης εισόδου / εξόδου και λανθάνων χρόνος

Ορίζουμε το ρυθμό μετάδοσης (throughput) εισόδου ως την ποσότητα των δειγμάτων εισόδου ανά δευτερόλεπτο. Παρομοίως, ορίζουμε το ρυθμό μετάδοσης εξόδου ως ποσότητα δειγμάτων εξόδου ανά δευτερόλεπτο. Ο λανθάνων χρόνος (latency) είναι ο χρόνος που απαιτείται για την επεξεργασία ενός μεμονωμένου συμβόλου από την είσοδο στην έξοδο. Ο ρυθμός μετάδοσης και ο λανθάνων χρόνος είναι δύο σημαντικοί περιορισμοί του συστήματος.

2.1.7 Μοντέλο Ροής Δεδομένων Πόρων

Σημείωση: Πως λειτουργεί ένας δρών με τον λανθάνοντα χρόνο εκτέλεσης

- Κάθε δρών επισημαίνεται με ένα λανθάνοντα χρόνο εκτέλεσης (execution latency). Αυτός είναι ο χρόνος που χρειάζεται ο δρών για να ολοκληρώσει έναν υπολογισμό. Υποθέτουμε ότι το εσωτερικό πρόγραμμα του δρώντα απαιτεί όλες οι εισόδου να είναι διαθέσιμες στην αρχή της εκτέλεσης. Ο λανθάνων χρόνος

εκφράζεται σε μονάδες χρόνου και ανάλογα με τον στόχο υλοποίησης μπορεί να επιλέγει μια κατάλληλη μονάδα – κύκλοι ρολογιού, nsec κ.ο.κ.

Σημείωση: Καθυστέρηση D

- Κάθε ουρά FIFO αντικαθίσταται από ένα κανάλι επικοινωνίας με σταθερό αριθμό καθυστερήσεων. Μια καθυστέρηση είναι μια θέση αποθήκευσης που μπορεί να κρατήσει ένα σύμβολο. Μια μονή καθυστέρηση μπορεί να κρατήσει ένα σύμβολο για μια μεμονωμένη εκτέλεση ενός δρώντα. Η αντικατάσταση μιας ουράς FIFO με θέσεις αποθήκευσης καθυστέρησης σημαίνει επίσης ότι πρέπει να αλλάξει ο κανόνας πυροδότησης του δρώντα. Αντί να ελέγχουμε τον αριθμό των στοιχείων σε μια ουρά FIFO, ο δρών θα ελέγχει τώρα για την παρουσία ενός συμβόλου σε μια θέση καθυστέρησης αποθήκευσης.

Η Εικόνα 19 δείχνει τρεις μονού ρυθμού γράφους ροής δεδομένων, που έγιναν με δύο δρώντες A και B. Ο δρών A χρειάζεται πέντε μονάδες λανθάνοντα χρόνου, ενώ ο δρών B απαιτεί τρεις μονάδες λανθάνοντα χρόνου. Αυτός ο γράφος ροής δεδομένων φέρει επίσης μια σύνδεση εισόδου και εξόδου, μέσω των οποίων το σύστημα μπορεί να δεχτεί ένα ρεύμα δειγμάτων εισόδου και να παραδώσει ένα ρεύμα δειγμάτων εξόδου. Για την ανάλυσή μας, δεν καθορίζουμε ρυθμό δειγματοληψίας εισόδου ή εξόδου. Αντ' αυτού, μας ενδιαφέρει να μάθουμε πόσο γρήγορα μπορούν να τρέξουν αυτοί οι γράφοι ροής δεδομένων.

Τελικά, αυτή η ανάλυση αποδίδει τις χρονικές στιγμές που ο γράφος διαβάζει από την είσοδο του συστήματος ή γράφει στην έξοδο του συστήματος.

Σημείωση: Εικόνα 19α, 19β,

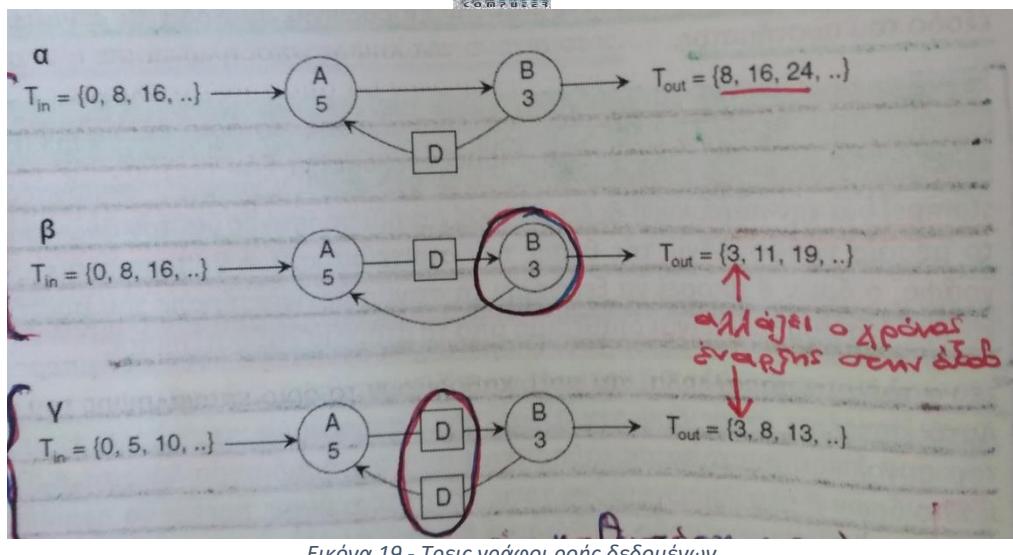
Στους γράφους των Εικόνων 19α, 19β υπάρχει ένα μοναδικό στοιχείο καθυστέρησης στο βρόχο. Η είσοδος / έξοδος δεδομένων ορίζεται από το συνδυασμένο χρόνο εκτέλεσης του δρώντα A και του δρώντα B. Οι χρονικές σημάνσεις (time stamps) για την παραγωγή δεδομένων είναι διαφορετικές για τον άνω γράφο και το μεσαίο γράφο, λόγω της θέσης του στοιχείου καθυστέρησης: για το μεσαίο γράφο, ο δρών B μπορεί να ξεκινήσει τη στιγμή αρχικοποίησης του συστήματος, καθώς ένα σύμβολο είναι διαθέσιμο από το στοιχείο καθυστέρησης.

Σημείωση: Εικόνα 19γ

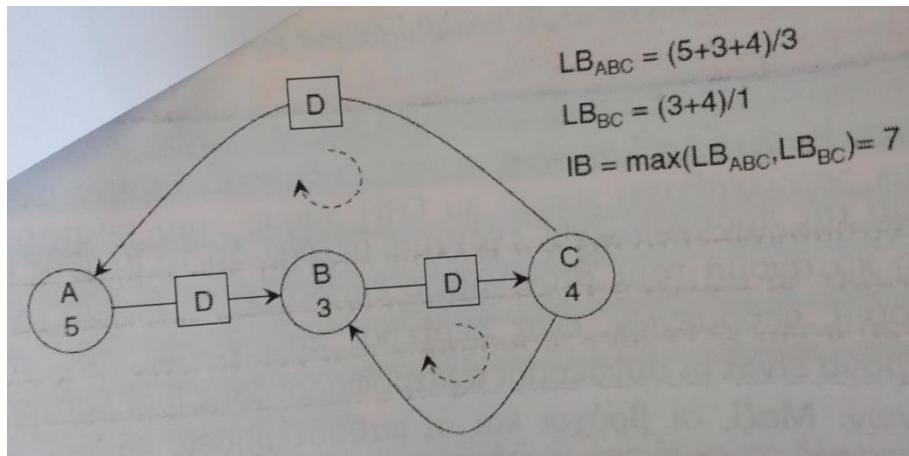
Στους γράφους της Εικόνας 19γ, υπάρχουν δύο στοιχεία καθυστέρησης στο βρόχο. Αυτό επιτρέπει στους δρώντες A και B να λειτουργούν παράλληλα. Η απόδοση του συνολικού συστήματος καθορίζεται από τον βραδύτερο δρώντα A. Παρόλο που ο δρών B ολοκληρώνεται σε τρεις μονάδες χρόνου, πρέπει να περιμένει την επόμενη διαθέσιμη είσοδο έως ότου ο δρών A ενημερώσει το στοιχείο καθυστέρησης στην έξοδο του.

Σημείωση: Ρυθμοί μετάδοσης Εικόνων 19

Συνεπώς, συμπεραίνουμε ότι οι δύο άνω γράφοι έχουν ρυθμό μετάδοσης 1 δείγμα ανά 8 μονάδες χρόνου και ότι ο κάτω γράφος έχει ρυθμό μετάδοσης 1 δείγμα ανά 5 μονάδες χρόνου.

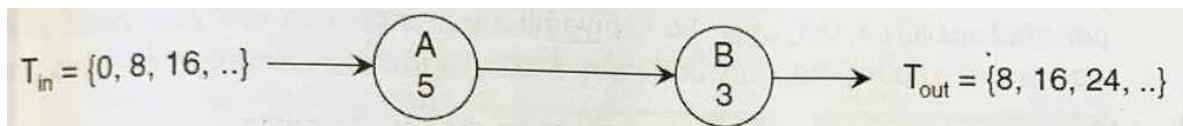


Εικόνα 19 - Τρεις γράφοι ροής δεδομένων



Εικόνα 20 - Υπολογισμός ορίου βρόχου και ορίου επανάληψης

Οι δρώντας Α και Κέι έχουν καθυστερήσεις στις εισόδους τους, έτσι ώστε να μπορούν πάντα να εκτελούνται παράλληλα. Ο δρών Β, ωστόσο πρέπει να περιμένει ένα αποτέλεσμα από τον Κέι προτού μπορέσει να υπολογίσει την έξοδο του.



Εικόνα 21 - Όριο επανάληψης για γραμμικό γράφο

Ένα νέο δείγμα δεν μπορεί να διαβαστεί στην είσοδο μέχρι να παραχθεί το δείγμα εξόδου, που να αντιστοιχεί στο προηγούμενο δείγμα εισόδου. Η Εικόνα 21 δείχνει ακριβώς αυτό, για ένα γραμμικό γράφο με δύο δρώντες Α και Β. Όταν ο Α διαβάσει ένα νέο δείγμα, ο Β θα υπολογίσει μια αντίστοιχη έξοδο τη χρονική στιγμή 8, καθώς η απουσία καθυστέρησης μεταξύ των Α και Β το αποτρέπει νωρίτερα. Η ανάλυση των γραμμικών τμημάτων στις εισόδους ή/και εξόδους ενός γράφου ροής δεδομένων μπορεί εύκολα να ληφθεί υπόψη υποθέτοντας μια εννοούμενη ανατροφοδότηση από κάθε έξοδο ενός γράφου ροής δεδομένων σε κάθε είσοδο.

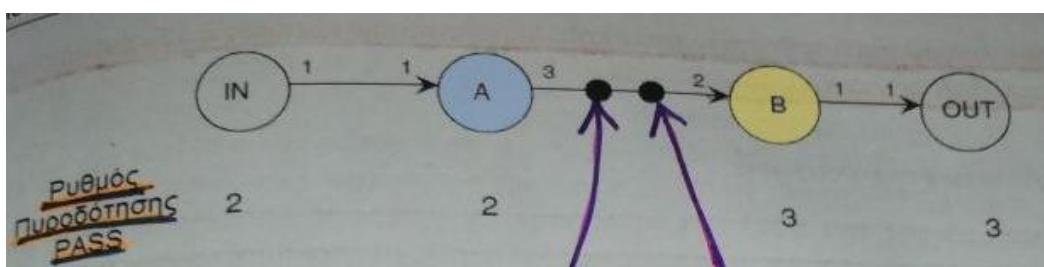
Μετασχηματισμοί

Σημείωση: Σημασία Μετασχηματισμών

Μπορούμε να αξιολογήσουμε τους κατάλληλους μετασχηματισμούς για να βελτιώσουμε την απόδοση των αργών γράφων ροής δεδομένων. Μας ενδιαφέρουν μετασχηματισμοί που διατηρούν τη λειτουργικότητα ενός γράφου ροής δεδομένων, αλλά αυξάνουν το ρυθμό μετάδοσης και/ή μειώνουν τον λανθάνοντα χρόνο.

Σημείωση: Είδη (κατηγορίες) μετασχηματισμών και χαρακτηριστικά

- **Η Επέκταση Πολλαπλών Ρυθμών (Multi-rate Expansion)** χρησιμοποιείται για τη μετατροπή ενός σύγχρονου γράφου ροής δεδομένων πολλαπλών ρυθμών σε ένα σύγχρονο γράφο ροής δεδομένων μονού ρυθμού. Αυτός ο μετασχηματισμός είναι χρήσιμος επειδή οι άλλοι μετασχηματισμοί υποθέτουν συστήματα SDF μονού – ρυθμού.
- **Ο Επαναχρονισμός (Retiming)** εξετάζει την ανακατομή στοιχείων καθυστέρησης σε ένα γράφο ροής δεδομένων, προκειμένου να βελτιστοποιήσει το ρυθμό μετάδοσης του γράφου. Ο επαναχρονισμός δεν μεταβάλλει τον λανθάνοντα χρόνο ή τη μεταβατική συμπεριφορά ενός γράφου ροής δεδομένων.
- **Η Διοχέτευση (Pipelining)** εισάγει πρόσθετα στοιχεία καθυστέρησης σε ένα γράφο ροής δεδομένων, με σκοπό τη βελτιστοποίηση του ορίου επανάληψης του γράφου. Η διοχέτευση αλλάζει το ρυθμό μετάδοσης και τη μεταβατική συμπεριφορά ενός γράφου ροής δεδομένων.
- **Το Άπλωμα (Unfolding)** αυξάνει την υπολογιστική παραλληλία σε ένα γράφο ροής δεδομένων διπλασιάζοντας τους δρώντες. Το άπλωμα δεν αλλάζει τη μεταβατική συμπεριφορά ενός γράφου ροής δεδομένων, αλλά μπορεί να τροποποιήσει το ρυθμό μετάδοσης.



Εικόνα 22 - Γράφημα πολλαπλού ρυθμού και ροής δεδομένων

2.1.8 Επέκταση Πολλαπλών Ρυθμών

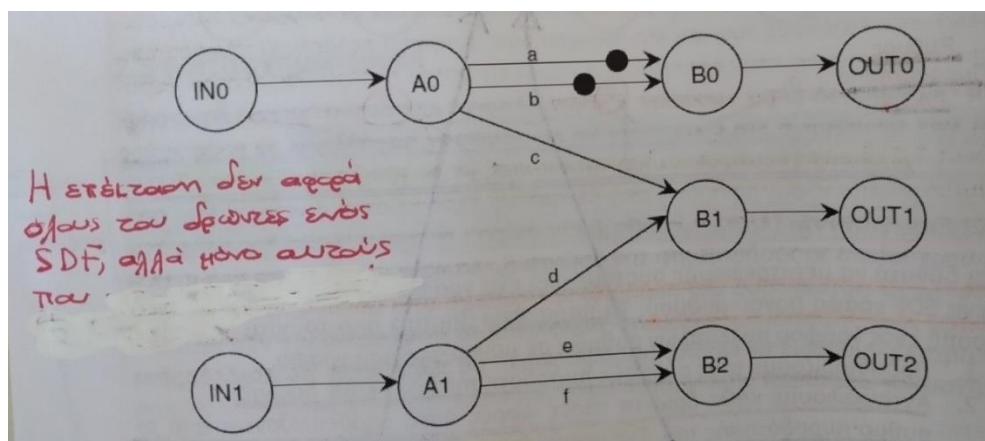
Είναι δυνατό να μετατρέψουμε συστηματικά ένα γράφο SDF πολλαπλών ρυθμών σε ένα SDF γράφου μονού – ρυθμού. Τα παρακάτω βήματα απαιτούνται για τη μετατροπή ενός γράφου πολλαπλών ρυθμών σε μονού – ρυθμού γράφο.

1. Αντιγράφουμε κάθε δρώντα τόσες φορές όσες υποδεικνύονται από το ρυθμό πυροδότησής του. Για παράδειγμα, δεδομένου ενός δρώντα A με ρυθμό πυροδότησης 2, δημιουργούμε τους A0 και A1. Αυτοί οι δρώντες είναι δύο πανομοιότυπα αντίγραφα του ίδιου γενικού δρώντα A.
2. Μετατρέπουμε κάθε είσοδο/έξοδο δρώντα πολλαπλών ρυθμών σε πολλαπλές εισόδους / εξόδους μονού ρυθμού. Για παράδειγμα, αν η είσοδος ενός δρώντα έχει ρυθμό κατανάλωσης 3, την αντικαθιστούμε με τρεις εισόδους μονού ρυθμού.

3. Εισάγουμε ξανά τα αρχικά σύμβολα στο σύστημα, κατανέμοντάς τα ακολουθιακά πάνω στις ουρές μονού - ρυθμού.

Εξετάστε το παράδειγμα ενός γράφου SDF πολλαπλών ρυθμών στην Εικόνα 22. Ο δρών A παράγει τρία σύμβολα ανά πυροδότηση, ο δρών B καταναλώνει δύο σύμβολα ανά πυροδότηση. Οι ρυθμοί πυροδότησης που προκύπτουν είναι 2 και 3 αντίστοιχα.

Οι δρώντες έχουν αντιγραφεί σύμφωνα με τους ρυθμούς πυροδότησής τους, και όλες οι πολλαπλές ρυθμών θύρες μετατράπηκαν σε θύρες μονού ρυθμού. Τα αρχικά σύμβολα ανακατανέμονται στις ουρές που συνδέουν τα στιγμιότυπα των A και B. Η κατανομή των συμβόλων ακολουθεί την αλληλουχία των ουρών μεταξύ των A, B (δηλαδή ακολουθεί τη σειρά a, b, κλπ)



Εικόνα 23 - Επέκταση γράφου SDF πολλαπλών ρυθμών σε μονό-ρυθμό

2.1.9 Επαναχρονισμός

Σημείωση: Σημασία επαναχρονισμού

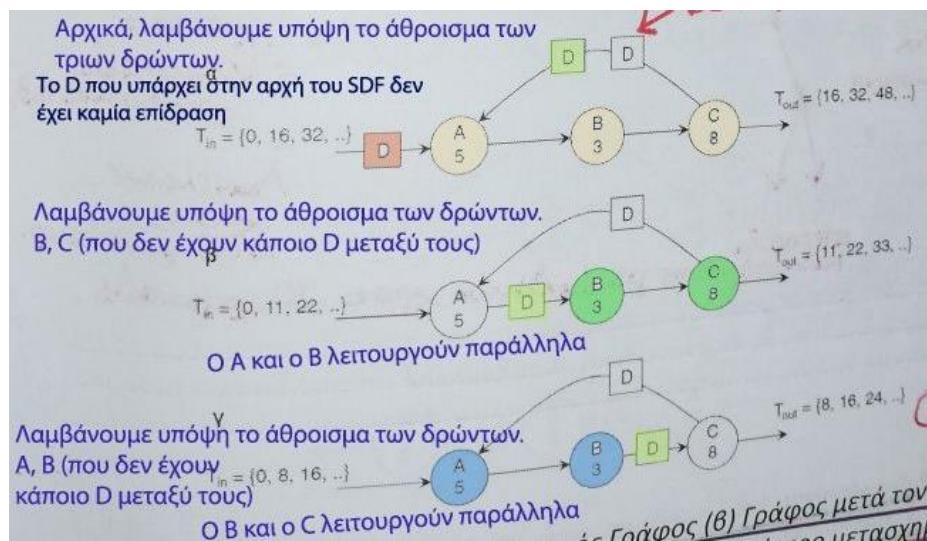
Ο επαναχρονισμός είναι ένας μετασχηματισμός σε γράφο ροής δεδομένων που δεν μεταβάλλει τον συνολικό αριθμό καθυστερήσεων μεταξύ εισόδου και εξόδου ενός γράφου ροής δεδομένων. Αντιθέτως, ο επαναχρονισμός είναι η ανακατομή των καθυστερήσεων στο γράφος ροής δεδομένων. Με τον τρόπο αυτό, η άμεση εξάρτηση μεταξύ των δρώντων μπορεί να σπάσει, επιτρέποντάς τους να λειτουργούν παράλληλα. Ένας επαναχρονισμένος γράφος μπορεί να έχει αυξημένο ρυθμό μετάδοσης συστήματος. Ο μετασχηματισμός επαναχρονισμού είναι εύκολο να κατανοηθεί.

Σημείωση: Παράδειγμα επαναχρονισμού

Η Εικόνα 24 απεικονίζει τον επαναχρονισμό χρησιμοποιώντας ένα παράδειγμα. Ο επάνω γράφος ροής δεδομένων, στην Εικόνα 24α, απεικονίζει το αρχικό σύστημα. Αυτός ο γράφος έχει ένα όριο επανάληψης 8. Ωστόσο, η περίοδος εξόδου δεδομένων της Εικόνας 24α είναι 16 μονάδες χρόνου, επειδή οι δρώντες A, B και C πρέπει να εκτελούνται ως ακολουθία. Εάν υποθέσουμε ότι ο δρών A θα πυροδοτήσει μία φορά, τότε θα καταναλώσει τα σύμβολα (καθυστερήσεις) στις εισόδους του και θα παράγει ένα σύμβολο εξόδου. Ο γράφος που προκύπτει φαίνεται στην Εικόνα 24β. Αυτή τη φορά, η περίοδος εξόδου δεδομένων έχει μειωθεί σε 11 μονάδες χρόνου.

Σημείωση: Εικόνα 24 και ταχύτερη υλοποίηση

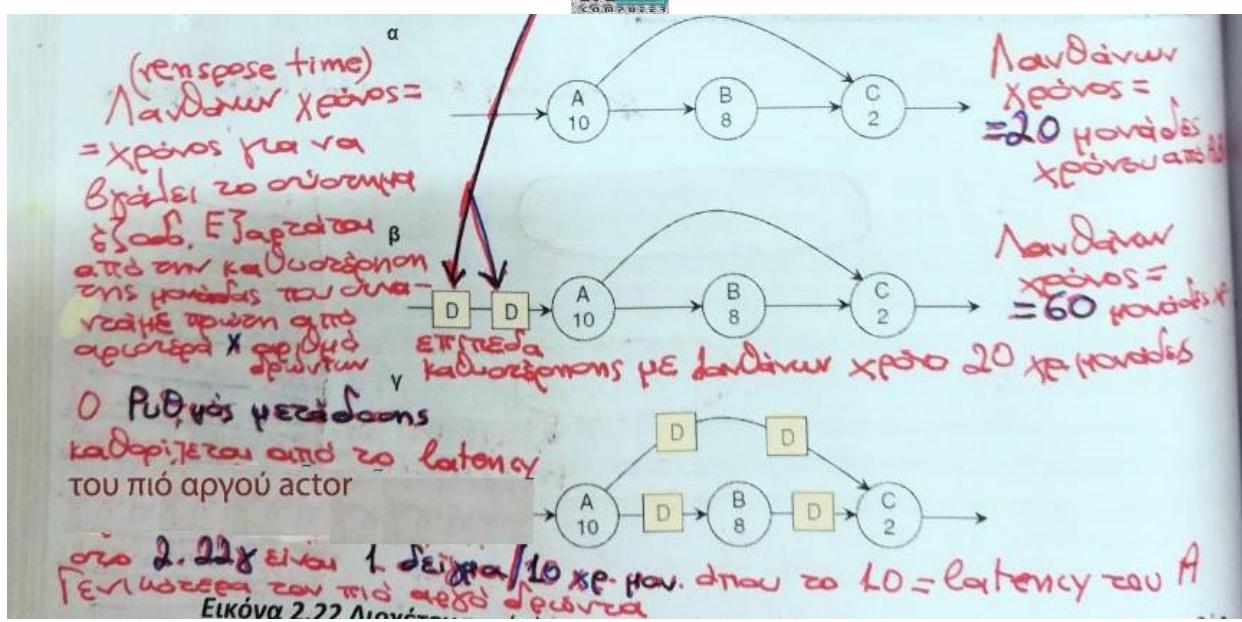
Τέλος, η Εικόνα 24γ δείχνει το αποτέλεσμα της μετατόπισης της καθυστέρησης διαμέσου του δρώντα B, για να αποκτηθεί μια ακόμα ισοδύναμη σήμανση. Αυτή η υλοποίηση είναι ταχύτερη από την προηγούμενη. Στην πραγματικότητα αυτή η υλοποίηση επιτυγχάνει το όριο επανάληψης των 8 μονάδων χρόνου ανά δείγμα. Η μετατόπιση της καθυστέρησης στην ακμή BC θα είχε ένα αποτέλεσμα μια καθυστέρηση στις εξόδους του δρώντα C: μία στην ουρά εξόδου και μία στον βρόχο ανάδρασης. Αυτός ο τελικός μετασχηματισμός αναδεικνύει μια σημαντική ιδιότητα του επαναχρονισμού: δεν είναι δυνατό να αυξηθεί ο αριθμός των καθυστερήσεων σε ένα βρόχο μέσω επαναχρονισμού.



Εικόνα 24 – Επαναχρονισμός (α) Αρχικός Γράφος (β) Γράφος μετά τον πρώτο μετασχηματισμό επαναχρονισμού (γ) Γράφος μετά το δεύτερο μετασχηματισμό επαναχρονισμού

2.1.10 Διοχέτευση (Pipelining)

Η διοχέτευση (pipelining) αυξάνει το ρυθμό μετάδοσης ενός γράφου ροής δεδομένων με κόστος τον αυξημένο λανθάνοντα χρόνο. Η διοχέτευση μπορεί εύκολα να γίνει κατανοητή ως ένας συνδυασμός επαναχρονισμού και προσθήκης καθυστερήσεων. Η Εικόνα 25 παρουσιάζει τη διοχέτευση με ένα παράδειγμα. Ο αρχικός γράφος στην Εικόνα 25α επεκτείνεται με δύο καθυστερήσεις διοχέτευσης στην Εικόνα 25β. Η προσθήκη επίπεδων (stages) καθυστερήσεις στην είσοδο αυξάνει το λανθάνοντα χρόνο του γράφου. Πριν από τα επίπεδα καθυστέρησης ο λανθάνων χρόνος του συστήματος ήταν 20 μονάδες χρόνου. Μετά την προσθήκη των επιπέδων καθυστέρησης, ο λανθάνων χρόνος του συστήματος αυξάνεται σε 60 μονάδες χρόνου (3 δείγματα με λαμβάνων χρόνο 20 μονάδων χρόνου το καθένα). Ο ρυθμός μετάδοσης του συστήματος είναι 1 δείγμα ανά 20 μονάδες χρόνου. Μπορούμε τώρα να αυξήσουμε το ρυθμό μετάδοσης του συστήματος με επαναχρονισμό του διοχετευμένου γράφου, έτσι ώστε να λάβουμε την Εικόνα 25γ. Ο ρυθμός μετάδοσης του γράφου είναι τώρα 1 δείγμα ανά 10 μονάδες χρόνου και ο λανθάνων χρόνος είναι 30 μονάδες χρόνου (3 φορές 10 μονάδες χρόνου). Αυτή η ανάλυση επισημαίνει μια σημαντική ιδιότητα της διοχέτευσης, το πιο αργό επίπεδο της διοχέτευσης καθορίζει το ρυθμό μετάδοσης του συνολικού συστήματος.



Εικόνα 25 - Διαχέτευση (α) Αρχικός Γράφος (β) Γράφος μετά την προσθήκη δύο επιπέδων διοχέτευσης (γ) Γράφος μετά τον επαναχρονισμό των επιπέδων της διοχέτευσης

2.1.11 Άπλωμα (Unfolding)

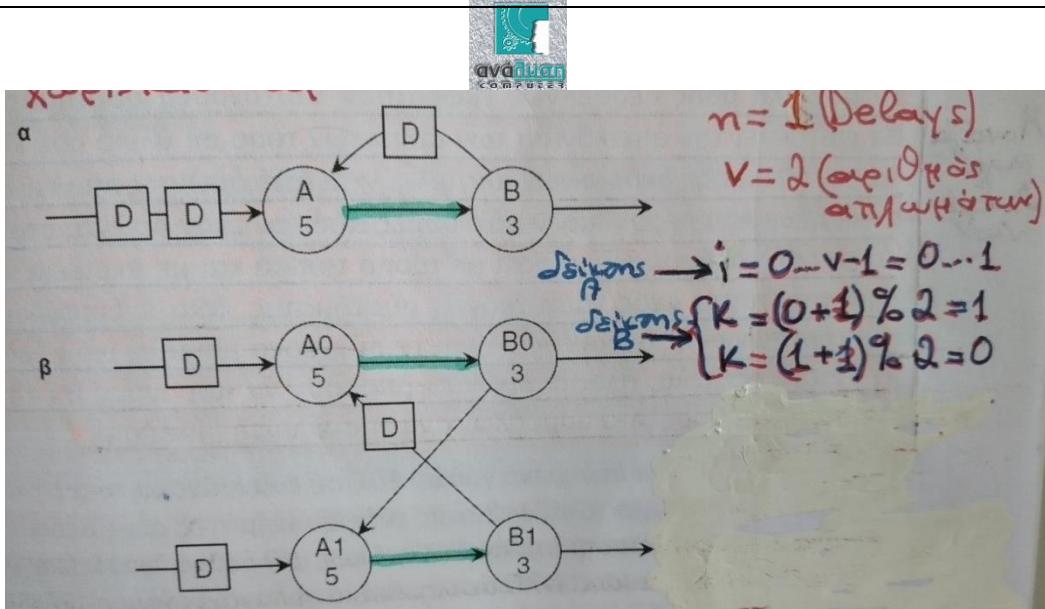
Σημείωση: Ιδέα απλώματος

Ο τελικός μετασχηματισμός που συζητάμε είναι το άπλωμα (ή ξεδίπλωμα). Η ιδέα του απλώματος είναι η παράλληλη υλοποίηση πολλαπλών στιγμιότυπων ενός δεδομένου γράφου ροής δεδομένων. Για παράδειγμα, υποθέστε ένα γράφο ροής δεδομένων G που επεξεργάζεται μια ροή δειγμάτων. Ο γράφος δύο απλωμάτων G_2 αποτελείται από δύο στιγμιότυπα του G . Ο γράφος G_2 επεξεργάζεται δύο δείγματα τη φορά.

Σημείωση: Διαδικασία απλώματος

Κάθε δρών A του απλωμένου συστήματος επαναλαμβάνεται τόσες φορές όσες απαιτούνται για το άπλωμα. Στη συνέχεια, γίνονται οι διασυνδέσεις σεβόμενοι την ακολουθία δειγμάτων του αρχικού συστήματος. Τέλος, οι καθυστερήσεις ανακατανέμονται πάνω στις διασυνδέσεις. Η διαδικασία απλώματος τυποποιείται ως εξής:

- Θεωρούμε ένα γράφο G με έναν δρώντα A και μια ακμή AB που μεταφέρει η καθυστερήσεις. Προσοχή: το n αναφέρεται μόνο στα D της ακμής AB
- Το v -άπλωμα του γράφου G θα αναπαράγει τον δρώντα A v φορές δηλαδή A_0, A_1, \dots, A_{v-1} . Η διασύνδεση AB αντιγράφεται επίσης v φορές δηλαδή $AB_0, AB_1, \dots, AB_{v-1}$.
- Η ακμή AB_i συνδέσει το A_i με το B_k , όπου $i:0..v-1$ και $k = (i + n) \% v$
- Η ακμή AB_i μεταφέρει $[(i + n) / v]$ καθυστερήσεις. Αν $n < v$, τότε θα υπάρχουν $v - n$ ακμές χωρίς καθυστέρηση.



Εικόνα 26 - Άπλωμα: (α) Αρχικός γράφος, (β) Γράφος μετά από δύο-απλώματα

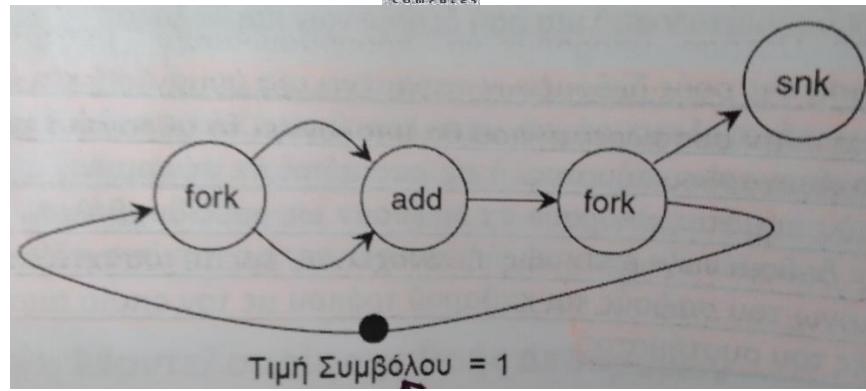
Σημείωση: Σημασία απλώματος

Η Εικόνα 26 απεικονίζει τη διαδικασία απλώματος με ένα παράδειγμα γράφου απλωμένου δύο φορές. Μπορείτε να παρατηρήσετε ότι ο απλωμένος γράφος έχει δύο εισόδους και δύο εξόδους και επομένως είναι σε θέση να δέχεται δύο φορές περισσότερα δεδομένα ανά επανάληψη από τον αρχικό γράφο ροής δεδομένων. Από την άλλη, το άπλωμα του γράφου φαίνεται να τον επιβραδύνει. Ο κρίσιμος βρόχος περιλαμβάνει τώρα τα A_0 , B_0 , A_1 και B_1 , αλλά εξακολουθεί να υπάρχει μόνο ένα στοιχείο καθυστέρησης στο συνολικό βρόχο. Επομένως, το όριο επανάληψης ενός v -απλωμένου γράφου έχει αυξηθεί v φορές.

Το άπλωμα των γράφων ροής δεδομένων χρησιμοποιείται για την επεξεργασία ροών δεδομένων με πολύ υψηλούς ρυθμούς δειγματοληψίας. Σε αυτή την περίπτωση, η ροή υψηλής ταχύτητας επεκτείνεται σε v παράλληλες ροές. Η ροή i μεταφέρει τα δείγματα $s_i, s_{i+v}, s_{i+2v}, \dots$ της αρχικής ροής. Για παράδειγμα στην Εικόνα 26, τα άρτια δείγματα θα υποστούν επεξεργασία από τον A_0 ενώ τα περιττά δείγματα θα υποστούν επεξεργασία από τον A_1 . Ωστόσο, επειδή το άπλωμα μειώνει το όριο επανάληψης, η συνολική ταχύτητα υπολογισμού του συστήματος μπορεί να επηρεαστεί.

Προβλήματα

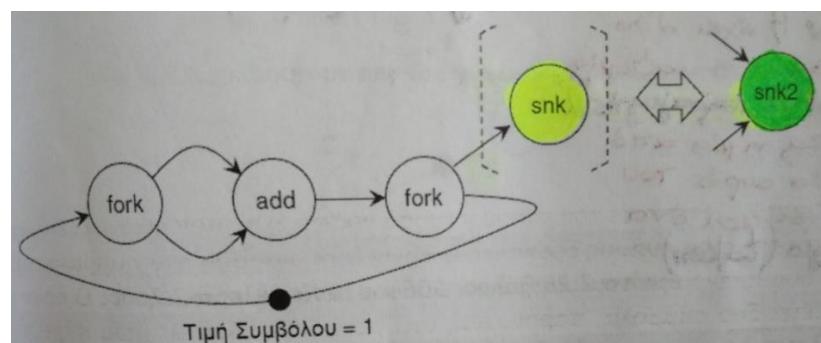
Πρόβλημα 2.1. Εξετάστε τον μονού ρυθμού γράφο SDF στην Εικόνα 27. Ο γράφος περιέχει τρεις τύπους δρώντων. Ο δρών fork διαβάζει ένα σύμβολο και παράγει δύο αντίγραφα του συμβόλου εισόδου, ένα σε κάθε έξοδο. Ο δρών add προσθέτει δύο σύμβολα, παράγοντας ένα μοναδικό σύμβολο που φέρει το άθροισμα των συμβόλων εισόδου. Ο δρών snk είναι ένας αποδέκτης – συμβόλων (signal-sink) που καταγράφει την ακολουθία των συμβόλων που εμφανίζονται στην είσοδο του. Ένα μοναδικό αρχικό σύμβολο, με τιμή 1, τοποθετείται σε αυτόν το γράφο. Βρείτε τη τιμή των συμβόλων που παράγονται στον δρώντα snk. Βρείτε έναν συμβολισμό συντομογραφίας για αυτήν την ακολουθία αριθμών.



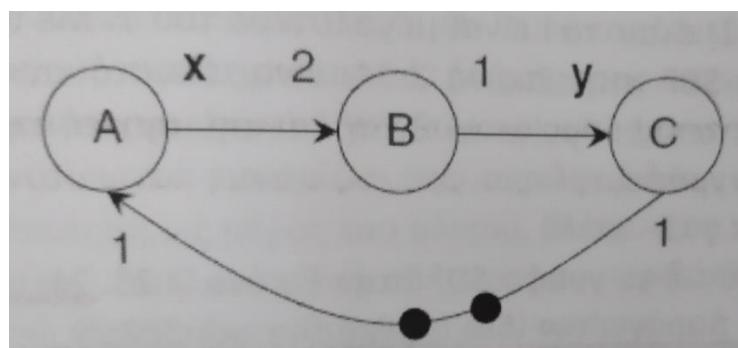
Εικόνα 27 - Γράφος SDF για το Πρόβλημα 2.1

Πρόβλημα 2.2. Εξετάστε το γράφο SDF στην Εικόνα 28. Μετατρέψτε αυτόν το γράφο έτσι ώστε να παράγει την ίδια ακολουθία συμβόλων ως πλειάδες αντί ως ακολουθία μονών τιμών (singletons). Για να το υλοποιήσετε, αντικαταστήστε τον δρώντα snk με τον snk2, έναν δρώντα που απαιτεί δύο σύμβολα σε δύο διαφορετικές εισόδους για να πυροδοτήσει.

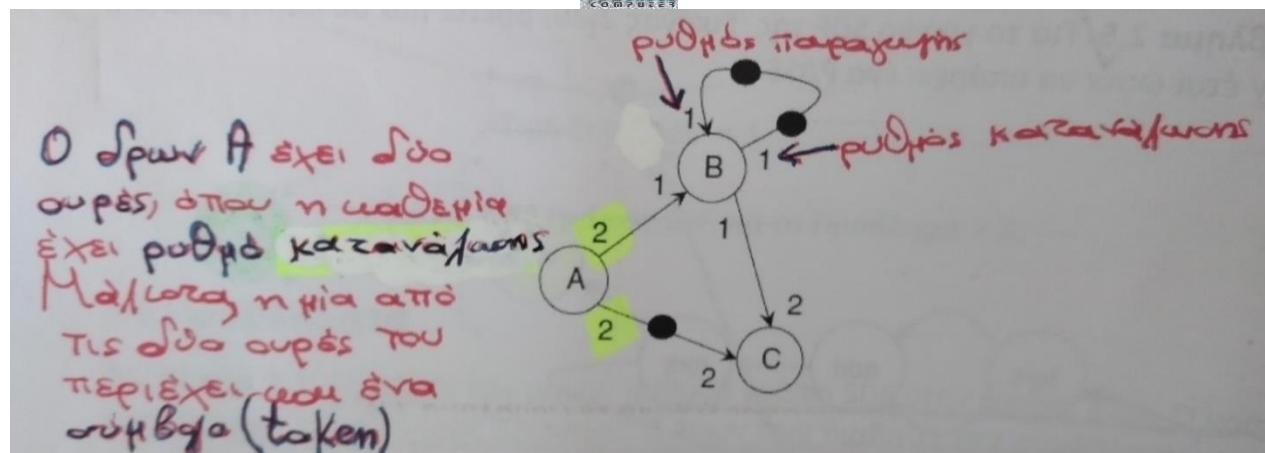
Πρόβλημα 2.3. Γι` α το γράφο SDF της Εικόνας 29, βρείτε μια συνθήκη μεταξύ x και y έτσι ώστε να υπάρχει ένα PASS.



Εικόνα 28 - Γράφος SDF για το Πρόβλημα 2.2

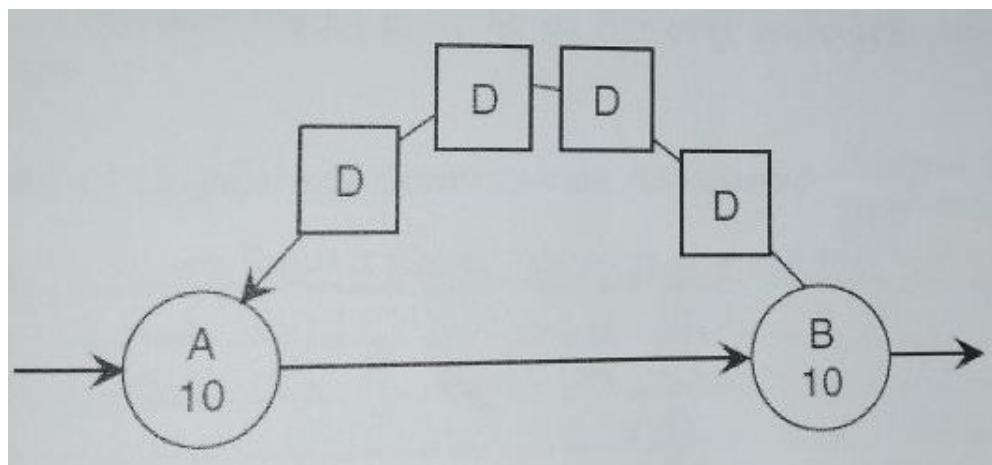


Εικόνα 29 - Γράφος SDF για το Πρόβλημα 2.3



Εικόνα 30 - Γράφος SDF για το Πρόβλημα 2.4

Πρόβλημα 2.4 Σχεδιάστε την επέκταση πολλαπλών ρυθμών για το SDF που δίνεται στην Εικόνα 30. Μην ξεχάσετε να αναδιανείμετε τα αρχικά σύμβολα στο αποτέλεσμα της πολλαπλού - ρυθμού επέκτασης.



Εικόνα 31 - Γράφος SDF για το Πρόβλημα 2.5

Πρόβλημα 2.5 Να απλώσετε το γράφο της Εικόνας 31, τρεις φορές. Προσδιορίστε το όριο επανάληψης, πριν και μετά τη διαδικασία απλώματος

3 Κεφάλαιο - Υλοποίηση Ροής Δεδομένων σε Λογισμικό και Υλικό

Υλοποίηση Ροής Δεδομένων σε Λογισμικό

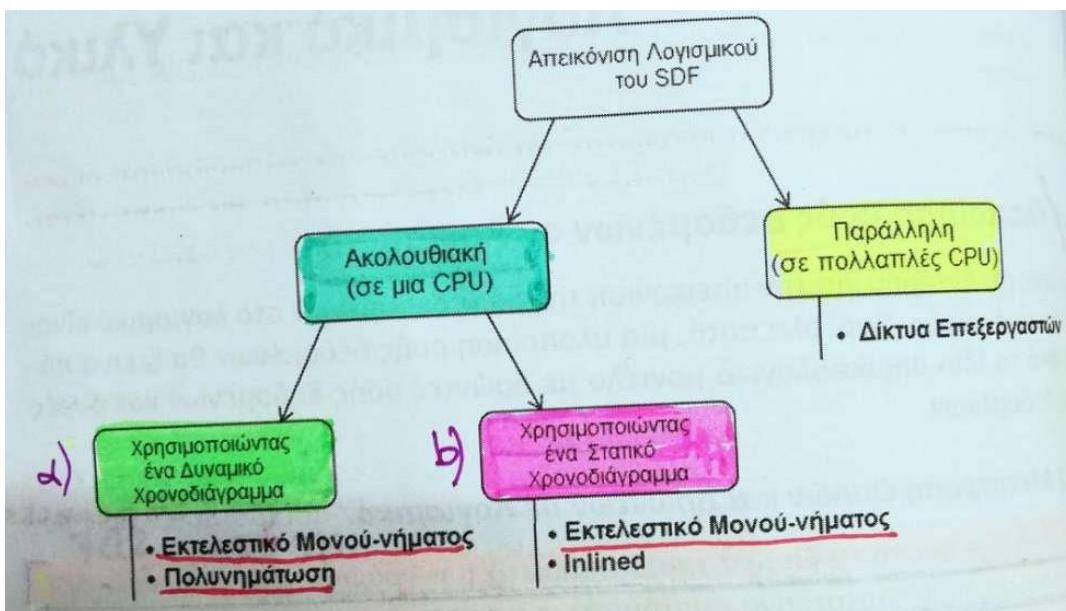
3.1.1 Μετατροπή Ουρών και Δρώντων σε Λογισμικό

Σημείωση: Χαρακτηριστικά γράφων SDF

Ας θυμηθούμε πρώτα τα βασικά χαρακτηριστικά των γράφων SDF. Οι γράφοι SDF αντιπροσωπεύουν ταυτόχρονα συστήματα, και χρησιμοποιούν δρώντες που επικοινωνούν μέσω ουρών FIFO. Η πυροδότηση δρώντος εξαρτάται μόνο από τη διαθεσιμότητα δεδομένων (συμβόλων) στις ουρές FIFO. Οι συνθήκες πυροδότησης διατυπώνονται στον κανόνα πυροδότησης για το συγκεκριμένο δρώντα. Η ποσότητα των συμβόλων που παράγεται/καταναλώνεται ανά πυροδότηση στην έξοδο / είσοδο ενός δρώντα, καθορίζεται από το ρυθμό παραγωγής / ρυθμό κατανάλωσης για αυτήν την έξοδο / είσοδο. Κατά την υλοποίηση ενός γράφου SDF στο λογισμικό, πρέπει να απεικονίσουμε όλα τα στοιχεία του γράφου SDF στο λογισμικό: δρώντες, ουρές και κανόνες πυροδότησης. Σε ορισμένες περιπτώσεις, η υλοποίηση λογισμικού μπορεί να βελτιστοποιηθεί. Για παράδειγμα, όταν είναι εφικτό να βρεθεί μια σταθερή σειρά εκτέλεσης των δρώντων (ένα PASS) μπορεί να είναι δυνατή η παράληψη του ελέγχου (testing) των κανόνων πυροδότησης. Ωστόσο, εξακολουθεί να ισχύει η αρχή ότι η υλοποίηση πρέπει να ακολουθήσει τους κανόνες σημασιολογίας της ροής δεδομένων.

Σημείωση: Τρόποι απεικόνισης ροής δεδομένων σε λογισμικό

Η Εικόνα 32 δείχνει διάφορες προσεγγίσεις για την απεικόνιση της ροής δεδομένων σε λογισμικό. Θα διακρίνουμε την απεικόνιση της ροής δεδομένων σε ένα σύστημα πολυεπεξεργασίας (multi-processor) από την απεικόνιση της ροής δεδομένων σε ένα σύστημα μονού επεξεργαστή.



Εικόνα 32 - Επισκόπηση πιθανών προσεγγίσεων για τη απεικόνιση ροής δεδομένων στο λογισμικό

Ωστόσο, το επίκεντρο αυτής της παραγράφου θα είναι η υλοποίηση μονού επεξεργαστή ενός γράφου ροής δεδομένων. Ο βασικό στόχος της υλοποίησης μονού – επεξεργαστή ενός συστήματος ροής δεδομένων, είναι η αποτελεσματική υλοποίηση ενός ακολουθιακού χρονοδιαγράμματος. Υπάρχουν δύο μέθοδοι για την υλοποίηση ενός τέτοιου ακολουθιακού χρονοδιαγράμματος.

Σημείωση: Μέθοδοι υλοποίησης ακολουθιακής SDF

α) Μπορούμε να χρησιμοποιούμε ένα δυναμικό χρονοδιάγραμμα (dynamic schedule), το οποίο θα υπολογίζει τη σειρά εκτέλεσης των δρώντων κατά την εκτέλεση του γράφου SDF. Έτσι, κατά τον χρόνο εκτέλεσης, το λογισμικό θα υπολογίζει τον κανόνα πυροδότησης των δρώντων και θα αποφασίσει εάν το σώμα του δρώντος πρέπει να εκτελεστεί ή όχι. Ένα δυναμικό χρονοδιάγραμμα μπορεί να υλοποιηθεί χρησιμοποιώντας εκτελεστικό μονού-νήματος ή αλλιώς χρησιμοποιώντας πολυνηματικό εκτελεστικό (multi-thread executive).

β) Μπορούμε επίσης να χρησιμοποιήσουμε ένα στατικό χρονοδιάγραμμα (static schedule), που σημαίνει ότι η σειρά εκτέλεσης των δρώντων καθορίζεται κατά το χρόνο – σχεδιασμού. Ένα στατικό χρονοδιάγραμμα μπορεί να υλοποιηθεί χρησιμοποιώντας ένα εκτελεστικό μονού – νήματος.

3.1.1.1 Ουρές FIFO

Το σύστημα SDF απαιτεί, κατ' αρχή, απείρως μεγάλες ουρές FIFO. Τέτοιες ουρές δεν μπορούν να υλοποιηθούν. Στην πράξη, μια υλοποίηση πρέπει να έχει πεπερασμένη αποθήκευση. Εάν γνωρίζουμε ένα PASS, μπορούμε να εξάγουμε στατικό χρονοδιάγραμμα και να καθορίσουμε το μέγιστο αριθμό συμβόλων σε κάθε ουρά και έπειτα να επιλέξουμε το κατάλληλο μέγεθος για κάθε ουρά.



Εικόνα 33 - Μια ουρά λογισμικού

Η Εικόνα 33 δείχνει μια διασύνδεση λογισμικού σε ένα αντικείμενο ουράς. Η διασύνδεση λογισμικού έχει δύο παραμέτρους και τρεις μεθόδους.

- Τον αριθμό των στοιχείων N που μπορούν να αποθηκευτούν από την ουρά (παράμετρος)
- Τον τύπο δεδομένων του στοιχείου για τα στοιχεία ουράς (παράμετροι)

Σημείωση: Μέθοδοι

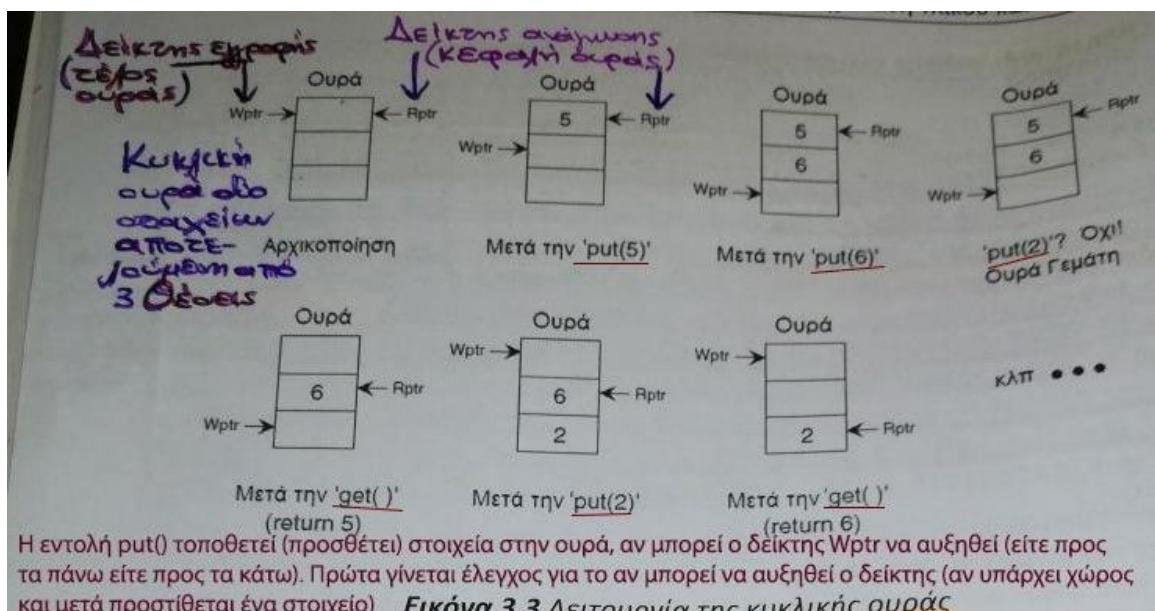
- Μια μέθοδο για να βάζουμε στοιχεία στην ουρά
- Μια μέθοδο για να παίρνουμε στοιχεία από την ουρά
- Μια μέθοδο για τον έλεγχο του αριθμού των στοιχείων στην ουρά

Σημείωση: Τρόπος λειτουργίας κυκλικής ουράς

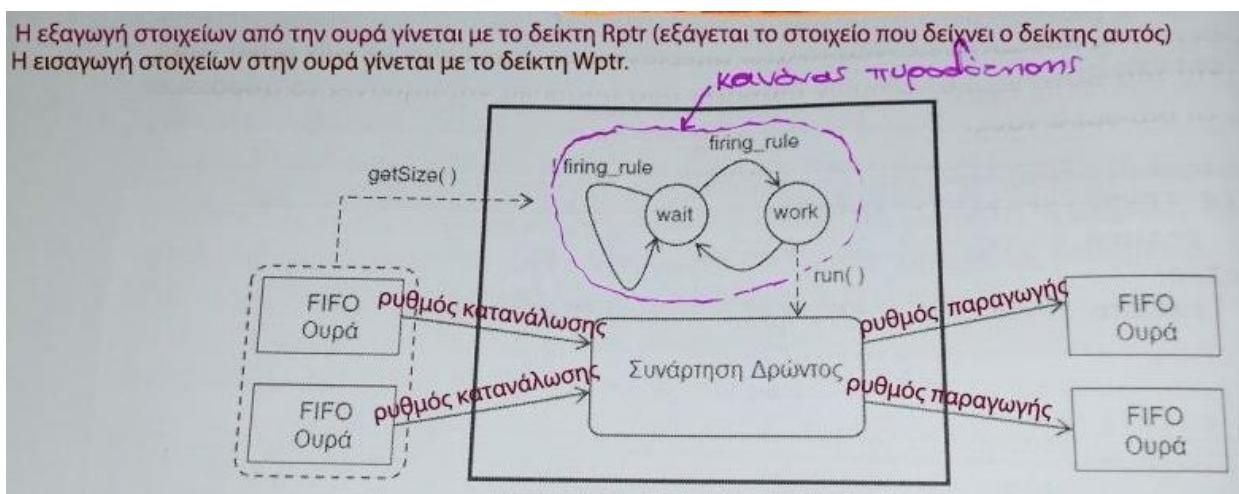
Η οργάνωση της αποθήκευσης μπορεί να γίνει με μια τυπική δομή δεδομένων όπως μια κυκλική ουρά. Μια κυκλική ουρά είναι μια δομή δεδομένων που αποτελείται από έναν πίνακα από θέσεις μνήμης, ένα δείκτη εγγραφής και ένα δείκτη ανάγνωσης. Η Εικόνα 34 απεικονίζει τη λειτουργία μιας κυκλικής ουράς δύο στοιχείων. Μια τέτοια ουρά χρησιμοποιεί ένα πίνακα τριών θέσεων. Ο δείκτης ανάγνωσης και εγγραφής, απεικονίζουν τις σχετικές διευθύνσεις ουράς σε διευθύνσεις πίνακα χρησιμοποιώντας διευθυνσιοδότηση modulo. Η κεφαλή της ουράς βρίσκεται στο Rptr. Το στοιχείο I της ουράς είναι στο $(Rptr + I) \bmod 3$. Το τέλος (tail) της ουράς είναι στο $(Wptr - 1) \bmod 3$.

3.1.1.2 Δρώντες

Ένας δρών ροής δεδομένων μπορεί να υλοποιηθεί ως συνάρτηση C, με κάποια επιπλέον υποστήριξη για διασύνδεση με τις ουρές FIFO. Οι σχεδιαστές συχνά θα κάνουν διάκριση μεταξύ των εσωτερικών δραστηριοτήτων ενός δρώντα και της συμπεριφοράς εισόδου – εξόδου.



Η εξαγωγή στοιχείων από την ουρά γίνεται με το δείκτη Rptr (εξάγεται το στοιχείο που δείνει ο δείκτης αυτός)
Η εισαγωγή στοιχείων στην ουρά γίνεται με το δείκτη Wptr.



Εικόνα 35 - Υλοποίηση σε λογισμικό του δρώντος ροής δεδομένων

Σημείωση: Τρόπος Λειτουργίας Εικόνας 35



Η Εικόνα 35 δείχνει ότι η λογική του κανόνα πυροδότησης υλοποιείται ως ένας μικρός, τοπικός ελεγκτής μέσα στον δρόντα. Ο τοπικός ελεγκτής περνάει από δύο καταστάσεις. Στην κατάσταση wait ο δρόν παραμένει αδρανής, αλλά δοκιμάζει τον κανόνα πυροδότησης σε κάθε κλήση (invocation) του δρόντα (μια κλήση ισοδυναμεί με την κλήση της συνάρτησης C που υλοποιεί τον δρόντα ροής δεδομένων). Όταν ο κανόνας πυροδότησης υπολογίζει αληθές, ο δρόν προχωρά στην κατάσταση work. Σε αυτήν την κατάσταση, η κλήση του δρόντος θα διαβάσει σύμβολα από την (τις) ουρά(ές) εισόδου, θα εξαγάγει τις τιμές τους και θα τις τροφοδοτήσει στο σώμα του δρόντος. Στη συνέχεια, οι παραγόμενες τιμές εξόδου γράφονται στις ουρές εξόδου και ο δρόν επιστρέφει στην κατάσταση wait. Όταν υλοποιούμε τον δρόντα, πρέπει να φροντίσουμε να υλοποιήσουμε την πυροδότηση όπως έχει προδιαγραφεί. Η πυροδότηση του δρόντος SDF υποδηλώνει ότι ο δρόν πρέπει να διαβάσει όλες τις ουρές εισόδου σύμφωνα με τους καθορισμένους ρυθμούς κατανάλωσης και πρέπει να γράφει όλες τις ουρές εξόδου σύμφωνα με τους καθορισμένους ρυθμούς παραγωγής.

Υλοποίηση Ροής Δεδομένων σε Υλικό

3.1.2 Γράφοι SDF μονού-ρυθμού σε Υλικό

Σημείωση: SDF γράφος μονού ρυθμού

Η απλούστερη περίπτωση είναι η απεικόνιση από έναν γράφο SDF μονού ρυθμού στο υλικό. Σε ένα μονού ρυθμού χρονοδιάγραμμα, ο σχετικός ρυθμός πυροδότησης όλων των δρώντων είναι ίσος με 1. Αυτό σημαίνει ότι όλοι οι δρώντες θα εκτελούνται με τον ίδιο ρυθμό. Εάν οι δρώντες υλοποιηθούν σε υλικό, όλοι θα δουλεύουν με την ίδια συχνότητα ρολογιού.

Σημείωση: Απεικόνιση μονού γράφου σε υλικό

Θα απεικονίσουμε έναν τέτοιο γράφο μονού ρυθμού στο υλικό χρησιμοποιώντας τους ακόλουθους τρεις κανόνες.

1. Όλοι οι δρώντες υλοποιούνται ως συνδυαστικά κυκλώματα
2. Όλες οι ουρές επικοινωνίας υλοποιούνται ως καλώδια (χωρίς αποθήκευση – storage)
3. Κάθε αρχικό σύμβολο σε μια ουρά επικοινωνίας αντικαθίσταται από έναν καταχωρητή.

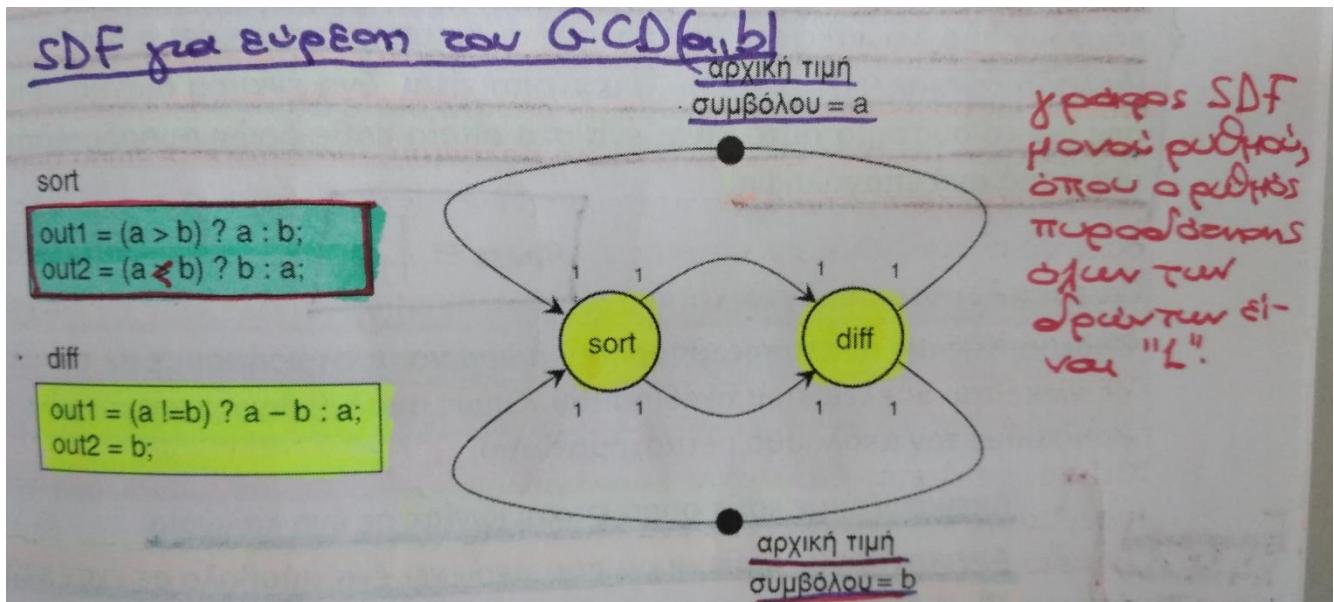
Σημείωση: Λανθάνοντας χρόνος μονοπατιού.

Μπορούμε να ορίσουμε το κρίσιμο μονοπάτι σε έναν γράφο ροής δεδομένων μέσω του μοντέλου πόρων που παρουσιάστηκε στην παράγραφο 2.4.2. Κατ' αρχάς, ορίζουμε το λανθάνοντα χρόνο του μονοπατιού ως το άθροισμα του λανθάνοντα χρόνου των δρώντων που περιλαμβάνονται στη διαδρομή.

Σημείωση: Συνδυαστικά μονοπάτια

Στη συνέχεια, καλούμε συνδυαστικό μονοπάτι, ένα μονοπάτι που δεν περιέχει αρχικά σύμβολα σε καμία από τις ουρές επικοινωνίας που περιλαμβάνονται στο μονοπάτι. Τέλος, ονομάζουμε κρίσιμο μονοπάτι ενός γράφου ροής δεδομένων το μεγαλύτερο συνδυαστικό μονοπάτι στον γράφο.

Παρακάτω παρουσιάζεται η υλοποίηση σε υλικό των γράφων SDF μονού ρυθμού με ένα παράδειγμα: ένα σύστημα SDF για τον Ευκλείδειο Αλγόριθμο του Μέγιστου Κοινού Διαιρέτη. Ο SDF στην Εικόνα 36 υπολογίζει το μέγιστο κοινό διαιρέτη δύο αριθμών a και b. Χρησιμοποιεί δύο δρώντες: τους sort και diff.



Εικόνα 36 - Ευκλείδειος Μέγιστος Κοινός Διαιρέτης ως γράφος SDF

Ο δρών sort διαβάζει δύο αριθμούς, τους ταξινομεί και τους αντιγράφει στην έξοδο. Ο δρών diff αφαιρεί τον μικρότερο αριθμό από τον μεγαλύτερο, εφόσον είναι διαφορετικοί. Εάν το σύστημα τρέξει για κάποιο χρονικό διάστημα, η τιμή των συμβόλων που περιφέρονται, συγκλίνει στον μέγιστο κοινό διαιρέτη των δύο αριθμών a και b.

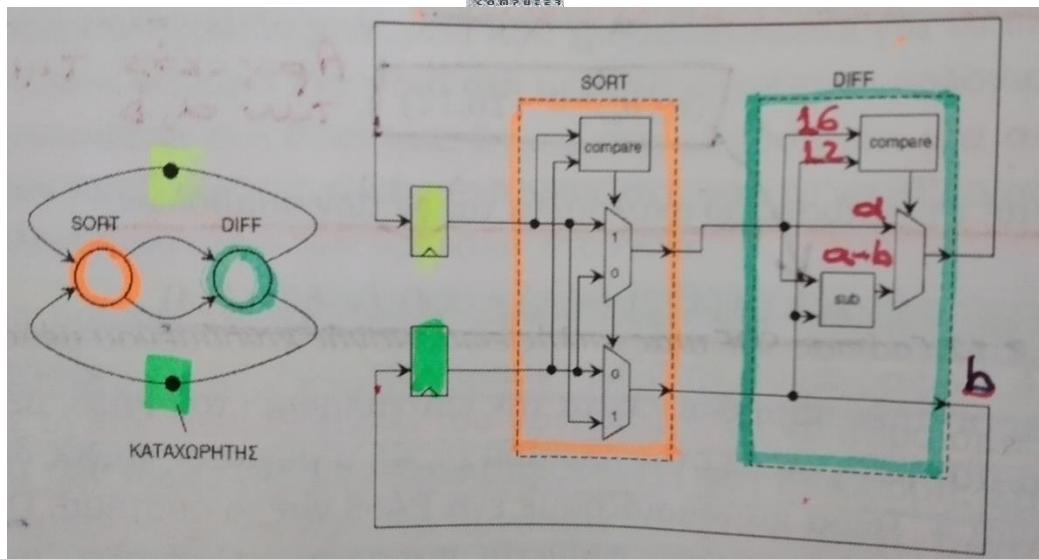
Ο πίνακας τροπολογίας G για αυτό το γράφο φαίνεται παρακάτω. Οι στήλες, από αριστερά προς τα δεξιά, αντιστοιχούν σε κάθε κόμβο του γράφου SDF, από αριστερά προς τα δεξιά.

Η τάξη αυτού του πίνακα είναι ένα, αφού οι στήλες αλληλοσυμπληρώνονται. Υπάρχουν δύο δρώντες στο γράφο, οπότε συμπεραίνουμε ότι η συνθήκη για PASS (δηλαδή $\text{rank}(G) = \text{κόμβοι}-1$) ικανοποιείται. Ένα έγκυρο διάνυσμα πυροδότησης για το σύστημα αυτό είναι ένα στο οποίο κάθε δρών πυροδοτείται ακριβώς μια φορά ανά επανάληψη.

$$q_{pass} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Σημείωση: Εφαρμογή κανόνων

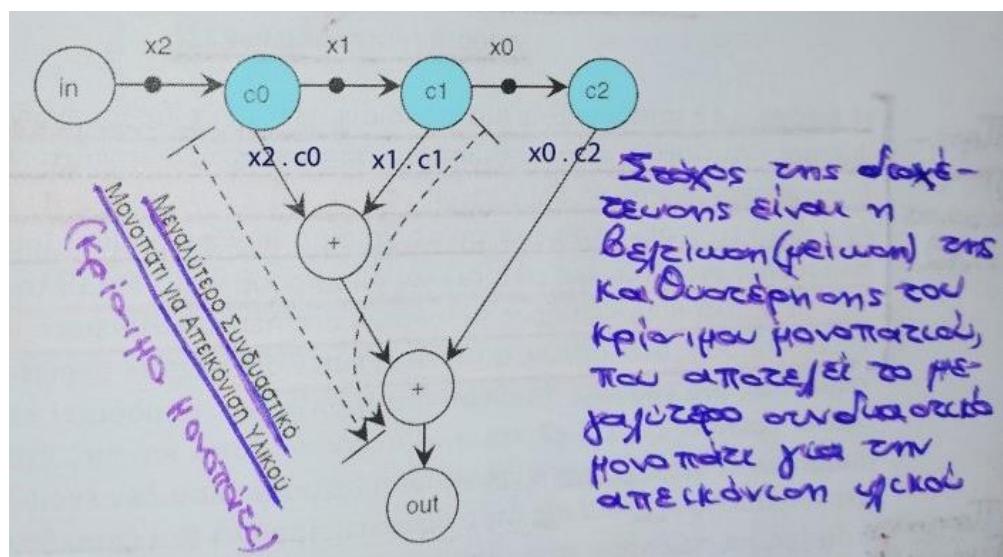
1. Απεικονίζουμε κάθε ουρά επικοινωνίας σε ένα καλώδιο.
2. Απεικονίζουμε κάθε ουρά που περιέχει ένα σύμβολο σε έναν καταχωρητή. Η αρχική τιμή του καταχωρητή πρέπει να ισούται με την αρχική τιμή του συμβόλου.
3. Απεικονίζουμε κάθε δρώντα σε ένα συνδυαστικό κύκλωμα, το οποίο ολοκληρώνει μια πυροδότηση μέσα σε έναν κύκλο ρολογιού. Τόσο ο δρών sort όσο και ο diff απαιτούν όχι παραπάνω από μια μονάδα σύγκρισης, μερικούς πολυπλέκτες και έναν αφαιρέτη.



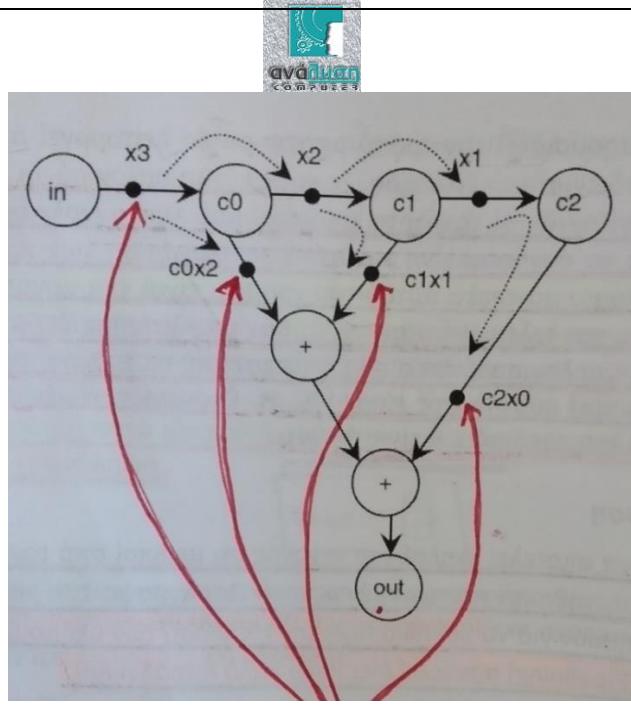
Εικόνα 37 - Υλοποίηση σε υλικό του αλγόριθμου του Ευκλείδη

3.1.3 Διοχέτευση

Δεν θα πρέπει να αποτελεί έκπληξη το γεγονός ότι μερικοί από τους μετασχηματισμού που συζητήθηκαν προηγουμένως στην Παράγραφο 2.5, μπορούν επίσης να χρησιμοποιηθούν για να βελτιώσουμε την απόδοση των υλοποιήσεων υλικού. Η Διοχέτευση (Pipelining) αποτελεί ένα πολύ καλό παράδειγμα.



Εικόνα 38 - Γράφος SDF μιας απλής εφαρμογής κινούμενου μέσου όρου



Εικόνα 39 - Διοχέτευση του φίλτρου κινούμενου μέσου όρου με την εισαγωγή πρόσθιτων συμβόλων (1)

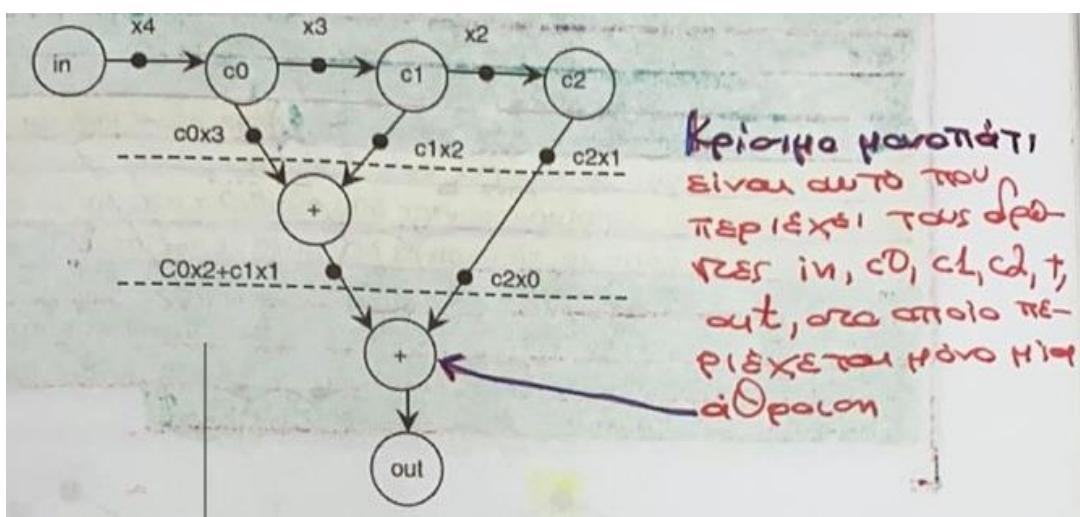
Σημείωση: Περιγραφή Εικόνας 38

Η Εικόνα 38 παρουσιάζει μια προδιαγραφή ροής δεδομένων ενός ψηφιακού φίλτρου. Υπολογίζει ένα σταθμισμένο άθροισμα των δειγμάτων μιας ροής εισόδου, με το άθροισμα να ορίζεται ως $out = x0.c2 + x1.c1 + x2.c0$.

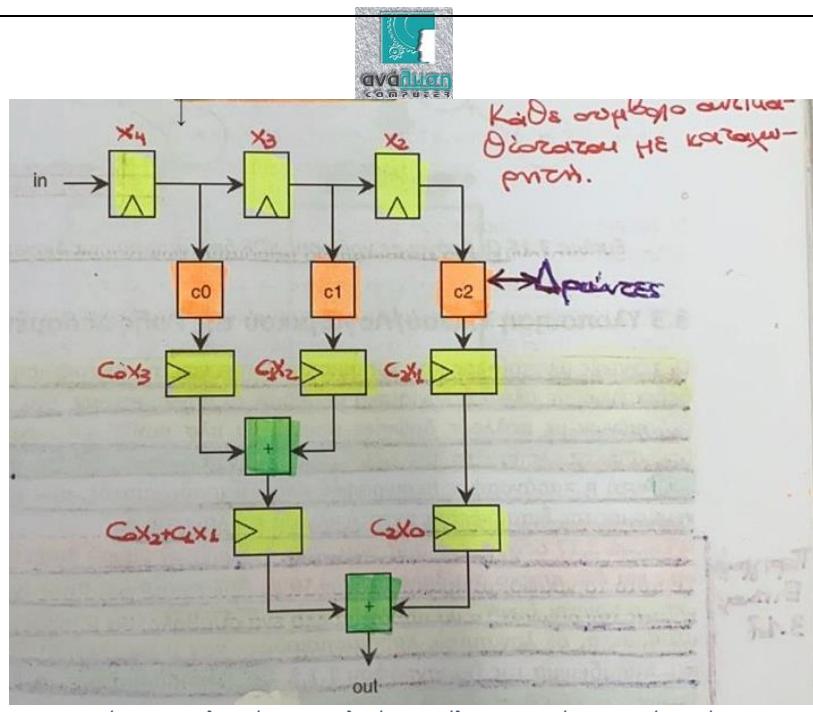
Από αυτό το γράφο φαίνεται ότι το κρίσιμο μονοπάτι είναι ίσιο με έναν πολλαπλασιασμό σταθεράς (με c_0 ή c_1) και δύο προσθέσεις. Θα θέλαμε να "σπρώξουμε" τα αρχικά σύμβολα προς τα "κάτω" στο δένδρο αθροιστή. Με τους κανόνες εκτέλεσης ροής δεδομένων, αυτό είναι εύκολο.

Σημείωση: Περιγραφή Εικόνας 39 – Μείωση κρίσιμου μονοπατιού

Σε αυτό το γράφο, το κρίσιμο μονοπάτι μειώνεται σε μόνο δύο αθροίσεις. Αφήνοντας τον δρώντα in να παράγει άλλο ένα σύμβολο θα μπορούμε να μειώσουμε το κρίσιμο μονοπάτι σε μία μόνο άθροιση, όπως φαίνεται στην Εικόνα 40. Ο διοχετευόμενος γράφος SDF που προκύπτει μπορεί να υλοποιηθεί όπως φαίνεται στην Εικόνα 41.



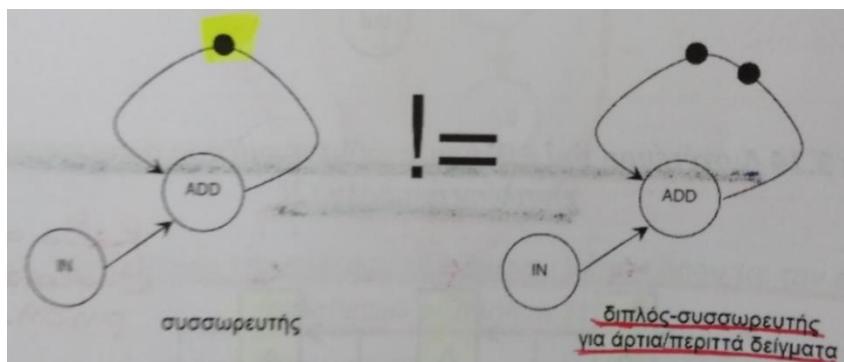
Εικόνα 40 -- Διοχέτευση του φίλτρου κινούμενου μέσου όρου εισάγοντας επιπλέον σύμβολα (2)



Εικόνα 41 – Υλοποίηση σε Υλικό του φίλτρου κινούμενου μέσου όρου

Σημείωση: Υπάρχουν δύο διαφορετικοί τρόποι (a) και (b) με τους οποίους μπορούμε να εκτελέσουμε διοχέτευση

Θυμηθείτε ότι η διοχέτευση απαιτεί να εισαγάγετε επιπλέον σύμβολα. Αυτό μπορεί να αλλάξει τη συμπεριφορά του γράφου ροής δεδομένων. Η αλλαγή συμπεριφοράς είναι προφανής στην περίπτωση βρόχων ανάδρασης, όπως φαίνεται στο κύκλωμα συσσωρευτή της Εικόνας 42. a) Χρησιμοποιώντας ένα μεμονωμένο σύμβολο στο χρόνο ανάδρασης ενός δρώντα add θα συγκεντρωθούν όλα τα δείγματα εισόδου. Χρησιμοποιώντας δύο σύμβολα στο βρόχο ανάδρασης θα συγκεντρωθούν τα άρτια και τα περιττά δείγματα b) Για να αποφύγετε την εισαγωγή τυχαίων συμβόλων σε έναν βρόχο, μπορείτε επίσης να εφαρμόσετε διοχέτευση ως εξής: εισαγάγετε αρχικά σύμβολα στην είσοδο ή την έξοδο του γράφου, έξω από οποιοδήποτε βρόχο.



Εικόνα 42 - Οι βρόχοι σε γράφους SDF δεν μπορούν να διοχετευθούν

Υλοποίηση Υλικού / Λογισμικού της Ροής Δεδομένων

Σημείωση: Υβριδικό SDF

Οι τεχνικές υλοποίησης που χρησιμοποιούνται για την απεικόνιση γράφων ροής δεδομένων σε υλικό ή λογισμικό μπορούν να συνδυαστούν. Ένα σύστημα ροής δεδομένων με πολλούς δρώντες μπορεί να υλοποιηθεί έτσι ώστε τμήμα των δρώντων να υλοποιείται στο υλικό, ενώ το άλλο μισό να υλοποιείται σε λογισμικό. Αυτή η

παράγραφος περιγράφει, μέσω παραδείγματος, πως μπορεί να γίνει χειρισμός της διασύνδεσης μεταξύ υλικού και λογισμικού.

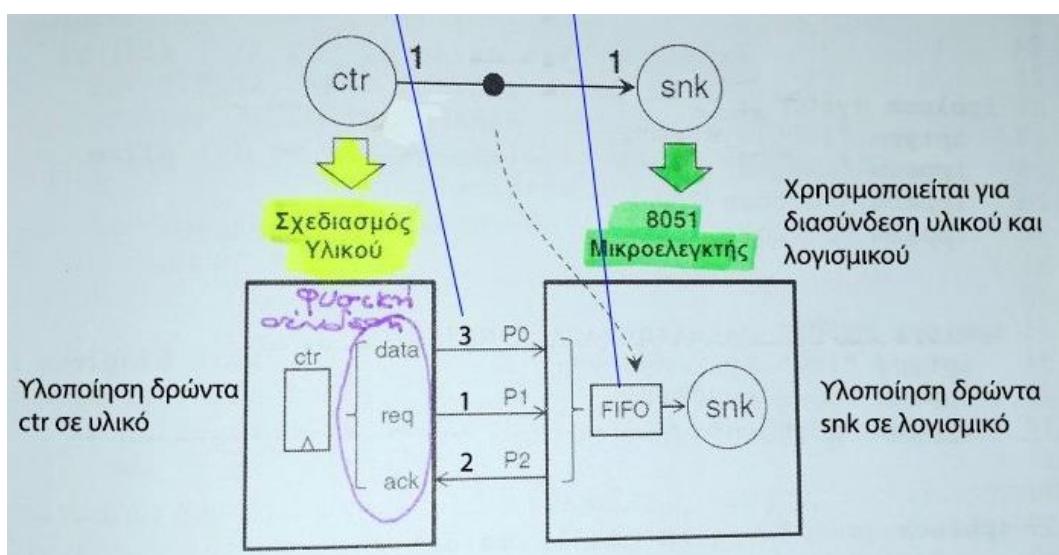
Σημείωση: Περιγραφή Εικόνας 43

Η Εικόνα 43 δείχνει ένα σύστημα ροής δεδομένων μονού ρυθμού με δύο δρώντες και ένα αρχικό σύμβολο μεταξύ τους. Απεικονίζουμε αυτό το σύστημα έτσι ώστε ο πρώτος δρών, ctr, να υλοποιείται στο υλικό, ενώ ο δεύτερος δρών, snk, υλοποιείται σε λογισμικό. Χρησιμοποιούμε έναν μικροελεγκτή 8051. Όπως και στο παραδειγμα της Παραγράφου 1.1.3 θα χρησιμοποιήσουμε θύρες μικροελεγκτή για τη σύνδεση υλικού και λογισμικού.

Η φυσική διασύνδεση μεταξύ υλικού και λογισμικού αποτελείται από τρεις διαφορετικές συνδέσεις: έναν δίαυλο δεδομένων, μια σύνδεση req (αίτημα) από υλικό σε λογισμικό και μια σύνδεση ack (επιβεβαίωση) από λογισμικό σε υλικό.

Σημείωση: Περιγραφή σημάτων data, req, ack της Εικόνας 43

Ο ρόλον των req και ack είναι να συγχρονίσουν το υλικό και το λογισμικό όταν μεταβιβάζουν ένα σύμβολο. Μια ουρά επικοινωνίας στη ροή δεδομένων χρειάζεται επίσης χώρο αποθήκευσης. Αυτή η αποθήκευση υλοποιείτε στον επεξεργαστή 8051 ως ουρά FIFO.



Εικόνα 43 - Υβριδική υλοποίηση υλικού/λογισμικού ενός γράφου ροής δεδομένων

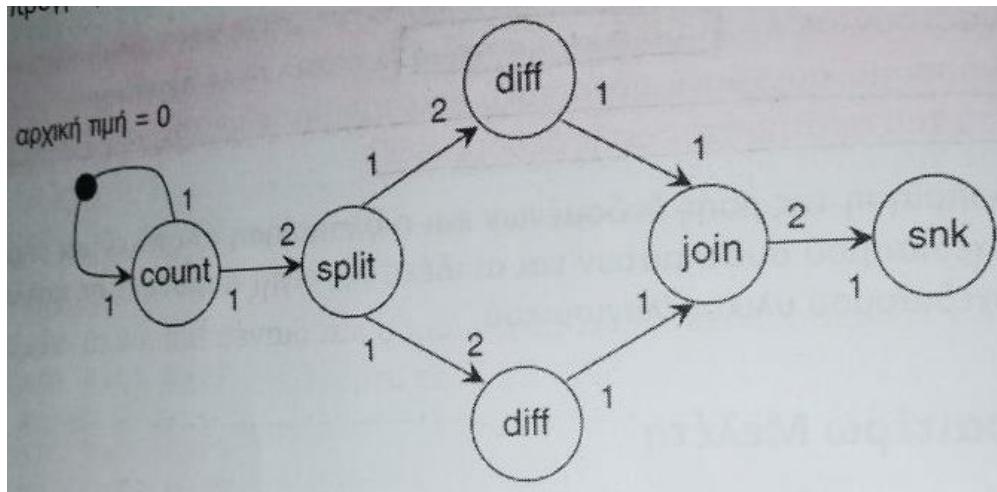
Σύνοψη

Συζητήσαμε τρις πιθανές υλοποιήσεις στόχους για συστήματα ροής δεδομένων λογισμικό, υλικό και συνδυασμένο υλικό/λογισμικό. Ο χώρος σχεδιασμού είναι ευρύς και προσφέρει πολλές εναλλακτικές λύσεις.

Για μια ακολουθιακή υλοποίηση λογισμικού, μπορούμε να χρησιμοποιήσουμε είτε νήματα είτε αλλιώς στατική χρονοδρομολόγηση συναρτήσεων C για να καταγράψουμε την ταυτόχρονη συμπεριφορά ενός συστήματος ροής δεδομένων. Τεχνικές βελτιστοποίησης, όπως η ενσωμάτωση της στατικής ροής δεδομένων αποδίδουν συμπαγείς και αποδοτικές υλοποιήσεις συστημάτων ροής δεδομένων σε λογισμικό.

Για την υλοποίηση σε υλικό μια απλή τεχνική μετατροπής "ενός-προς-ένα" μεταφράζει γράφους SDF μονού ρυθμού σε υλικό. Οι τεχνικές μετασχηματισμού ροής δεδομένων που συζητήσαμε νωρίτερα, συμπεριλαμβανομένης της διοχέτευσης και του επαναχρονισμού, μπορούν να εφαρμοστούν για τη βελτιστοποίηση της απόδοσης των γράφων με ένα μακρύ κρίσιμο μονοπάτι.

Τέλος, συζητήσαμε πως οι δύο αυτές τεχνικές σε υλικό και λογισμικό, μπορούν να συνδυαστούν σε ένα υβριδικό σύστημα. Σε αυτήν την περίπτωση, πρέπει να δημιουργήσουμε συγχρονισμό μεταξύ του τμήματος υλικού και του τμήματος λογισμικού του συστήματος ροής δεδομένων.



Εικόνα 44 - Γράφος SDF για το Πρόβλημα 3.3

Προβλήματα

Πρόβλημα 3.1 Προσθέστε την ακόλουθη βελτίωση στο σχεδιασμό της ουράς FIFO της Λίστας 3.1. Ξεκινώντας, κάνετε το αρχικό μέγεθος FIFO πολύ μικρό, για παράδειγμα δύο στοιχεία. Στη συνέχεια, υλοποιήστε δυναμική μεταβολή διαστάσεων (sizing) της FIFO: όταν η FIFO υπερχειλίσει με την `put_fifo`, διπλασιάστε τη δεσμευμένη μνήμη για τη FIFO. Χρησιμοποιήστε δυναμική δέσμευση μνήμης, όπως τη `malloc` στη C. Υλοποιήστε τις απαραίτητες αλλαγές στις `put_fifo` και `get_fifo`, για να καταγράφετε το μεταβλητό βάθος ουράς ροής δεδομένων.

Πρόβλημα 3.2 Χρησιμοποιώντας το API Quickthreads που ορίστηκε παραπάνω, δημιουργήστε μια προσομοίωση ροής δεδομένων για το γράφο SDF που φαίνεται στην Εικόνα 44.

Πρόβλημα 3.3 Μετατρέψτε το γράφο SDF της Εικόνας 44 σε μια μονού-ρολογιού υλοποίηση σε υλικό. Εκτελέστε πρώτα μια επέκταση πολλαπλών ρυθμών. Μπορείτε να υποθέσετε ότι ο δρών `snk` υλοποιείται ως θύρα εξόδου σε επίπεδο-συστήματος.

4 Κεφάλαιο Ανάλυση Ροής Ελέγχου και Ροής Δεδομένων

Ακμές Δεδομένων και Ελέγχου ενός Προγράμματος C

Σημείωση: Μοντέλο υπολογισμού ροής δεδομένων

Στο προηγούμενο κεφάλαιο, συζητήσαμε το μοντέλο υπολογισμού ροής δεδομένων. Θεμελιώδες σε αυτό το μοντέλο είναι η αποσύνθεση ενός συστήματος σε μεμονωμένους κόμβους (δρώντες), οι οποίοι επικοινωνούν μέσω μονοκατευθυνόμενων, σημείου σε σημείο, καναλιών (ουρές). Το παραγόμενο μοντέλο συστήματος αναπαρίσταται ως γράφος. Το μοντέλο υπολογισμού ροής δεδομένων περιγράφει ταυτόχρονους υπολογισμούς.

Για έναν σχεδιαστή λογισμικού, ένα πρόγραμμα C είναι λογισμικό, και είναι ακολουθιακό. Για έναν συσχεδιαστή υλικού – λογισμικού, ωστόσο, ένα πρόγραμμα C μπορεί να είναι υλικό ή λογισμικό, ανάλογα με τις απαιτήσεις και τις ανάγκες της εφαρμογής. Προφανώς, δεν μπορεί κάποιος να κάνει άμεση μετατροπή της C σε υλικό – ένα σημαντικό εμπόδιο είναι ότι το υλικό είναι φύσει παράλληλο, ενώ η C είναι ακολουθιακή.

Μπορούμε να σκεφτούμε ένα πρόγραμμα C ως μια υψηλού επιπέδου περιγραφή της συμπεριφοράς μιας υλοποίησης, χωρίς να καθορίσουμε συγκεκριμένα τις ακριβείς λεπτομέρειες της υλοποίησης.

Ορίζουμε δύο τύπους σχέσεων μεταξύ των λειτουργιών ενός προγράμματος C: ακμές δεδομένων και ακμές ελέγχου.

Μια ακμή δεδομένων είναι μια σχέση μεταξύ δύο λειτουργιών, τέτοια ώστε τα δεδομένα που παράγονται από την μία λειτουργία να καταναλώνονται από την άλλη.

Μια ακμή ελέγχου είναι μια σχέση μεταξύ δύο λειτουργιών, τέτοια ώστε η μία λειτουργία να πρέπει να εκτελεστεί μετά την άλλη.

Σημείωση: Πρώτο Παράδειγμα Ακμών Δεδομένων / Ελέγχου

Ο ορισμός φαίνεται παρόμοιος, όμως δεν είναι πανομοιότυπος. Ας προσπαθήσουμε να αναγνωρίσουμε τις ακμές δεδομένων και τις ακμές ελέγχου στην ακόλουθη συνάρτηση C, η οποία βρίσκει το μέγιστο δύο μεταβλητών.

```
int max (int a, b) {  
    int r;  
    if (a > b)  
        r = a;  
    else  
        r = b;  
    return r;  
}
```

Σημείωση: Εύρεση λειτουργιών συνάρτησης max

Η συνάρτηση περιέχει δύο εντολές εκχώρησης και ένα κλάδο if-then-else. Για τους σκοπούς της ανάλυσής μας, θα εξισώνουμε τις εντολές της C με "λειτουργίες". Επιπλέον, ορίζουμε τα σημεία εισόδου και τα σημεία εξόδου της συνάρτησης ως δύο επιπλέον λειτουργίες. Επομένως, η συνάρτηση max περιλαμβάνει πέντε λειτουργίες.

```
int max (int a, b) { // λειτουργία 1 – μπες στην συνάρτηση  
    int r;  
    if (a > b) // λειτουργία 2 – if-then-else  
        r = a; // λειτουργία 3
```

```

else
    r = b;                                // λειτουργία 4
    return r;                               // λειτουργία 5 – επέστρεψε max
}

```

Σημείωση: Ακμές Ελέγχου

Για να βρούμε τις ακμές ελέγχου, σε αυτή τη συνάρτηση, πρέπει να βρούμε την ακολουθία λειτουργιών αυτής της συνάρτησης. Μπορεί να υπάρχουν περισσότερες από μια πιθανές ακολουθίες, όταν το πρόγραμμα περιέχει if-then-else εντολές και βρόχους. Οι ακμές ελέγχου θα πρέπει να καταγράφουν όλα τα πιθανά μονοπάτια. Στο παράδειγμα, η λειτουργία 2 θα εκτελείται πάντα μετά τη λειτουργία 1. Επομένως, υπάρχει μια ακμή ελέγχου από τις λειτουργίες 1 στη 2. Μια εντολή if-then else εισάγει δύο ακμές ελέγχου, μία για κάθε πιθανό αποτέλεσμα του ελέγχου if-then-else. Εάν η συνθήκη $a > b$ είναι αληθής, τότε η λειτουργία 3 θα ακολουθήσει τη λειτουργία 2, διαφορετικά η λειτουργία 4 θα ακολουθήσει τη λειτουργία 2. Υπάρχει μια ακμή ελέγχου από τη λειτουργία 2 σε κάθε μία από τις λειτουργίες 3 και 4. Τέλος, η λειτουργία 5 θα ακολουθήσει είτε τη λειτουργία 3 είτε την 4. Υπάρχει μια ακμή ελέγχου από κάθε μία από τις λειτουργίες 3 και 4 στη λειτουργία 5. Συνοψίζοντας, η εύρεση των ακμών ελέγχου αντιστοιχεί στην εύρεση των δυνατών μονοπατιών εκτέλεσης στο πρόγραμμα C και στη σύνδεση των λειτουργιών σε αυτά τα μονοπάτια εκτέλεσης με ακμές.

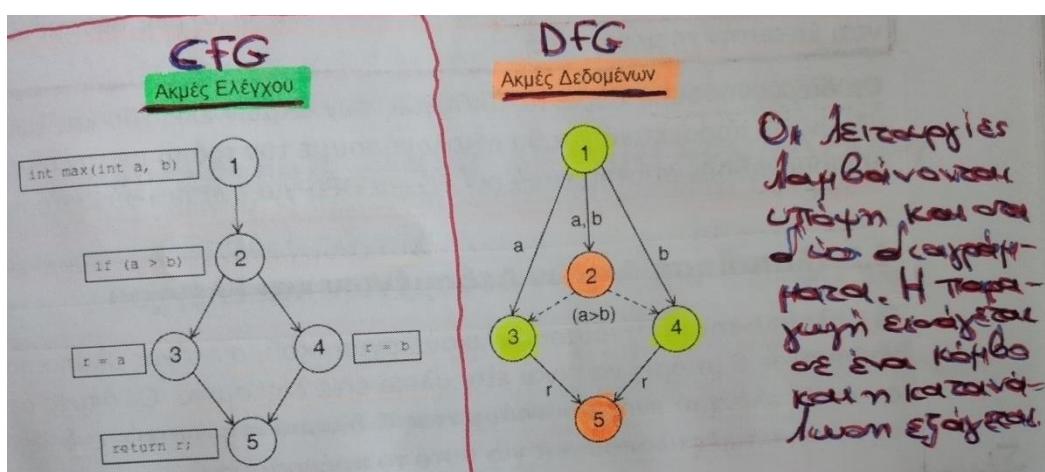
Σημείωση: Ακμές Δεδομένων

Για να βρούμε τις ακμές δεδομένων σε αυτή τη συνάρτηση, εξετάζουμε τα μοντέλα παραγωγής/κατανάλωσης δεδομένων σε κάθε λειτουργίας.

```

int max (int a, b) {                                // λειτουργία 1 – παρήγαγε a, b
    int r;
    if (a > b)                                     // λειτουργία 2 – κατανάλωσε a, b
        r = a;                                       // λειτουργία 3 – κατανάλωσε a και (a>b)
                                                       // παρήγαγε r
    else
        r = b;                                       // λειτουργία 4 - κατανάλωσε b και (a>b)
                                                       // παρήγαγε r
    return r;                                      // λειτουργία 5 – κατανάλωσε r
}

```



Εικόνα 45 - Ακμές ελέγχου και ακμές δεδομένων ενός απλού προγράμματος C

Οι ακμές δεδομένων ορίζονται μεταξύ των αντίστοιχων λειτουργιών παραγωγής/κατανάλωσης. Για παράδειγμα, η λειτουργία 1 ορίζει την τιμή των a και b . Αρκετές λειτουργίες θα κάνουν χρήση αυτών των τιμών. Η τιμή του a χρησιμοποιείται



από τις λειτουργίες 2 και 3. Συνεπώς, υπάρχει μια ακμή δεδομένων από τις λειτουργίες 1 στη 2, καθώς επίσης και μια ακμή δεδομένων από τις λειτουργίες 1 στη 3. Το ίδιο ισχύει για την τιμή του b, που παράγεται κατά τη λειτουργία και καταναλώνεται στις λειτουργίες 2 και 4. Υπάρχει μια ακμή δεδομένων για το b από τις λειτουργίες 1 στη 2, καθώς και από τις λειτουργίες 1 στη 4.

Οι εντολές ελέγχου στη C μπορεί επίσης να παράγουν ακμές δεδομένων. Σε αυτή την περίπτωση, η εντολή if-then-else υπολογίζει μια σημαία και η τιμή αυτής της σημαίας απαιτείται πριν από την εκτέλεση των επόμενων λειτουργιών. Για παράδειγμα, η λειτουργία 3 θα εκτελεστεί μόνο όταν η έκφραση συνθήκης ($a > b$) είναι αληθής. Μπορούμε να σκεφτούμε μια σημαία boolean που θα κρατά την τιμή του ($a > b$) από τις λειτουργίες 2 στην 3. Ομοίως, η λειτουργία 4 θα εκτελεστεί μόνο όταν η έκφραση συνθήκης ($a > b$) είναι ψευδής. Υπάρχει μια σημαία boolean που θα κρατά την τιμή του ($a > b$) από τις λειτουργίες 2 στη 4.

Σημείωση: Σύγκριση ακμών δεδομένων και ελέγχου (1)

Οι ακμές δεδομένων και οι ακμές ελέγχου των λειτουργιών από τη συνάρτηση μαχ μπορούν να απεικονιστούν σε ένα γράφο, όπου κάθε λειτουργία αντιπροσωπεύει έναν κόμβο. Το αποτέλεσμα φαίνεται στην Εικόνα 45 και αναπαριστά το γράφο ροής ελέγχου (CFG) και το γράφο ροής δεδομένων (DFG) για το πρόγραμμα. Οι ακμές ελέγχου εκφράζουν μια γενική σχέση μεταξύ δύο κόμβων (λειτουργίες), ενώ οι ακμές δεδομένων εκφράζουν μια σχέση μεταξύ δύο κόμβων για μια συγκεκριμένη μεταβλητή. Κατά συνέπεια, οι ακμές δεδομένων επισημαίνονται με αυτήν τη μεταβλητή.

Υλοποίηση Ακμών Δεδομένων και Ελέγχου

Σημείωση: Σύγκριση ακμών δεδομένων και ελέγχου (2)

- **Μια ακμή δεδομένων αντανακλά μια απαίτηση για τη ροή των πληροφοριών.** Εάν αλλάζετε τη ροή των πληροφοριών, αλλάζετε το νόημα του αλγορίθμου. Για το λόγο αυτό, μια ακμή δεδομένων είναι μια θεμελιώδης ιδιότητα της συμπεριφοράς που εκφράζεται σε αυτό το πρόγραμμα C.
- **Μια ακμή ελέγχου, από την άλλη, είναι συνέπεια της σημασιολογίας εκτέλεσης της γλώσσας του προγράμματος,** αλλά δεν είναι θεμελιώδους σημασίας για τη συμπεριφορά που εκφράζεται σε αυτό το πρόγραμμα C.

Μια ερώτηση ανακύπτει αναπόφευκτα: ποια είναι τα σημαντικά μέρη ενός προγράμματος C που θα είναι παρόντα σε οποιαδήποτε υλοποίηση του προγράμματος; Η απάντηση στην ερώτηση αυτή δίνεται από τις ακμές ελέγχου και τις ακμές δεδομένων του προγράμματος και συνοψίζεται ως εξής.

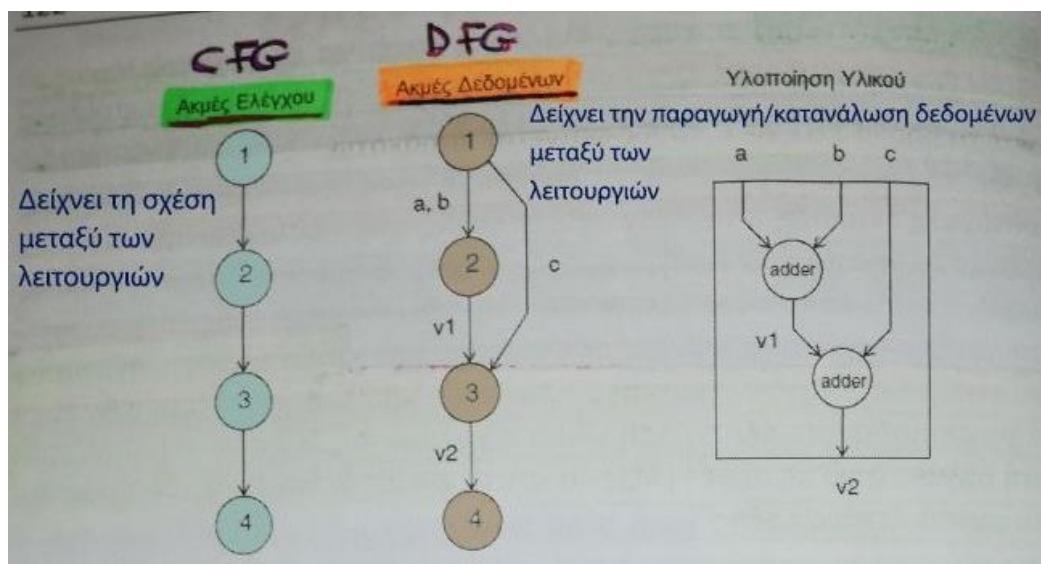
Μια ακμή δεδομένων πρέπει πάντα να υλοποιείται ανεξάρτητα από την υποκείμενη αρχιτεκτονική.

Μια ακμή ελέγχου μπορεί να απομακρυνθεί αν η υποκείμενη αρχιτεκτονική μπορεί να διαχειριστεί τον παραγόμενο ταυτοχρονισμό.

Με άλλα λόγια, οι ακμές ελέγχου μπορούν να απομακρυνθούν, μέσω της δημιουργίας επαρκούς παραλληλίας στην υποκείμενη αρχιτεκτονική.

Σημείωση: Δεύτερο Παράδειγμα Ακμών Δεδομένων / Ελέγχου

```
int sum (int a, b, c) { // λειτουργία 1
    int v1, v2;
    v1 = a + b; // λειτουργία 2
    v2 = v1 + c; // λειτουργία 3
    return v2 // λειτουργία 4
}
```



Εικόνα 46 - Υλοποίηση υλικού μιας αλυσιδωτής άθροισης

Είναι εύκολο να σχεδιάσετε μια πλήρως παράλληλη υλοποίηση υλικού αυτής της λειτουργίας. Αυτή η υλοποίηση παρουσιάζεται, μαζί με το γράφο ροής δεδομένων και το γράφο ροής ελέγχου της συνάρτησης, στην Εικόνα 46.

Κατασκευή του Γράφου Ροής Ελέγχου

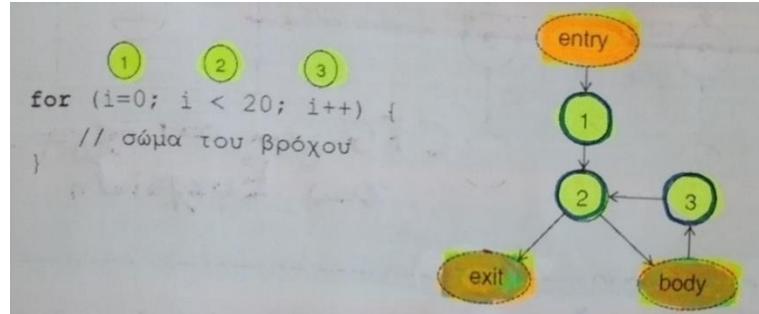
Σημείωση: Τι είναι ένας CFG

Ένα πρόγραμμα C μπορεί να μετατραπεί με τρόπο συστηματικό σε μια ενδιάμεση αναπαράσταση που ονομάζεται Γράφος Ροής Ελέγχου (Control Flow Graph – CFG). Ένας CFG είναι ένας γράφος που περιέχει όλες τις ακμές ελέγχου ενός προγράμματος. Κάθε κόμβος στο γράφο αντιπροσωπεύει μία μόνο λειτουργία (ή μια εντολή C). Κάθε ακμή του γράφου υποδεικνύει μια ακμή ελέγχου, δηλαδή μια σειρά εκτέλεσης για τις δύο λειτουργίες που συνδέονται με την εν λόγω ακμή.

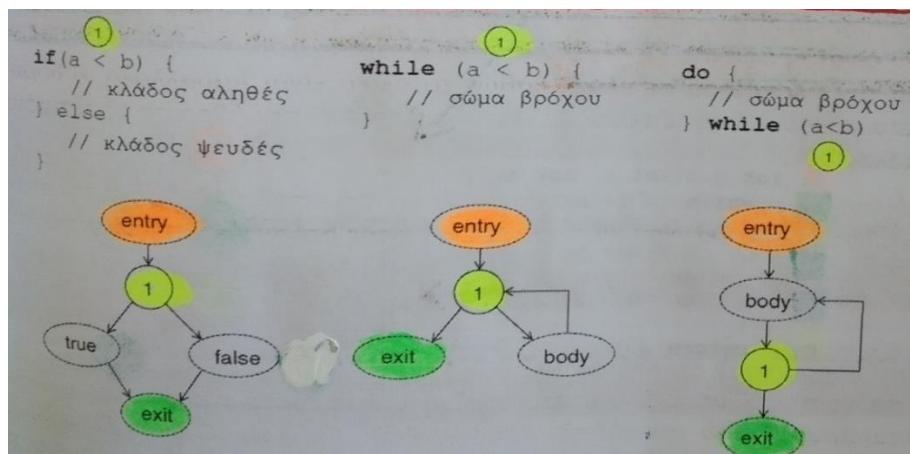
Καθώς η C εκτελείται ακολουθιακά, αυτή η μετατροπή είναι απλή. Ωστόσο, ορισμένες περιπτώσεις απαιτούν περαιτέρω προσοχή. Οι εντολές ελέγχου (όπως οι βρόχοι) ενδέχεται να απαιτούν πολλαπλές λειτουργίες. Επιπλέον, όταν γίνεται λήψη αποφάσεων, πολλαπλές ακμές ελέγχου μπορεί να προέρχονται από μία μόνο λειτουργία.

Θεωρήστε το βρόχο for σε C, όπως φαίνεται παρακάτω.

```
for (i = 0; i < 20; i++) {
    // σώμα βρόχου
}
```



Εικόνα 47 – CFG ενός βρόχου for



Εικόνα 48 - CFG των if-then-else, while-do, do-while

Αυτή η εντολή περιλαμβάνει τέσσερα ξεχωριστά μέρη: την αρχικοποίηση του βρόχου, τη συνθήκη του βρόχου, τη λειτουργία αύξησης του μετρητή βρόχου και το σώμα του βρόχου. Ο βρόχος for συνεισφέρει έτσι τρεις λειτουργίες στο CFG, όπως φαίνεται στην Εικόνα 47. Οι διακεκομμένοι κόμβοι σε αυτή την εικόνα (entry, exit, body), αντιπροσωπεύουν άλλα μέρη του προγράμματος C, καθένα από τα οποία είναι ένας πλήρης CFG, μονής εισόδου, μονής-εξόδου.

Ο βρόχος do-while και ο βρόχος while-do είναι παρόμοιες επαναληπτικές δομές. Η Εικόνα 48 παρουσιάζει ένα πρότυπο για κάθε μία από αυτές, καθώς και για την εντολή if-then-else.

Σημείωση: Τρίτο Παράδειγμα Ακμών Δεδομένων / Ελέγχου

Ως παράδειγμα, ας δημιουργήσουμε το CFG της ακόλουθης συνάρτησης C. Αυτή η συνάρτηση υπολογίζει τον Μέγιστο Κοινό Διαιρέτη (Greatest Common Divisor – GCD) χρησιμοποιώντας τον αλγόριθμο του Ευκλείδη.

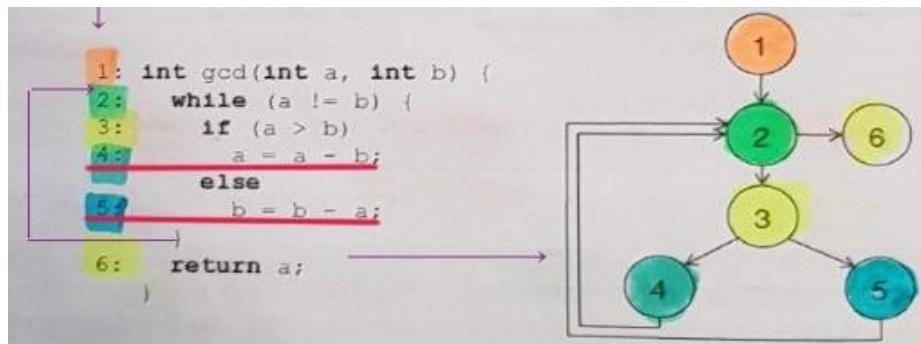
```
int gcd (int a, int b) {
    while (a != b) {
        if (a > b)
            a = a - b;
        else
```

```

b = b - a;
}
return a;
}

```

Για να κατασκευάσουμε το CFG αυτού του προγράμματος, μετατρέπουμε κάθε εντολή σε μία ή περισσότερες λειτουργίες στο CFG και στη συνέχεια, συνδέουμε τις λειτουργίες χρησιμοποιώντας ακμές ελέγχου. Το αποτέλεσμα αυτής της μετατροπής φαίνεται στην Εικόνα 49.



Εικόνα 49 - CFG από το πρόγραμμα GCD

Κατασκευή του Γράφου Ροής Δεδομένων

Σημείωση: Ορισμός DFG, Κόμβος και ακμή στο DFG

Ένα πρόγραμμα C μπορεί να μετατραπεί με τρόπο συστηματικό σε μια δομή δεδομένων που ονομάζεται Γράφος Ροής Δεδομένων (Data Flow Graph – DFG). Ένας DFG είναι ένας γράφος που αντανακλά όλες τις ακμές δεδομένων ενός προγράμματος. Κάθε κόμβος στο γράφο αντιπροσωπεύει μία μόνο λειτουργία (ή εντολή C). Κάθε ακμή του γράφου υποδηλώνει μια ακμή δεδομένων, δηλαδή μια σχέση παραγωγής / κατανάλωσης μεταξύ δύο λειτουργιών στο πρόγραμμα.

Σημείωση: Δημιουργία ενός DFG από ένα CFG

Προφανώς, οι CFG και DFG θα περιέχουν το ίδιο σύνολο κόμβων. Μόνο οι ακμές θα είναι διαφορετικές. Δεδομένου ότι μια μεταβλητή στη C μπορεί να γράφεται/διαβάζεται έναν αυθαίρετο αριθμό φορών, μπορεί να είναι δύσκολο να βρεθούν ταιριαστά ζεύγη ανάγνωσης-εγγραφής στο πρόγραμμα. Ο ευκολότερος τρόπος για να κατασκευάσετε ένα DFG, είναι να κατασκευάσετε πρώτο το CFG και στη συνέχεια να χρησιμοποιήσετε το CFG σε συνδυασμό με το πρόγραμμα C για να εξαγάγετε τον DFG. Το κόλπο είναι να εντοπιστούν οι διαδρομές ελέγχου και ταυτόχρονα να αναγνωριστούν οι αντίστοιχες λειτουργίες ανάγνωσης και εγγραφής μεταβλητών.

Σημείωση: Βήματα ανάκτησης ακμών δεδομένων

Η διαδικασία για την ανάκτηση των ακμών δεδομένων που σχετίζονται με τις εντολές εκχώρησης έχει ως εξής:

- Στον CFG, επιλέξτε έναν κόμβο όπου μια μεταβλητή χρησιμοποιείται ως ένας τελεστέος σε μια έκφραση. Σημειώστε αυτόν τον κόμβο ως κόμβο ανάγνωσης.



2. Βρείτε τους κόμβους CFG που εκχωρούν αυτή τη μεταβλητή. Σημειώστε αυτούς τους κόμβους ως κόμβους – εγγραφής.
3. Εάν υπάρχει μια άμεση διαδρομή ελέγχου από έναν κόμβο-εγγραφής σε έναν κόμβο-ανάγνωσης που δεν διέρχεται από άλλο κόμβο – εγγραφής τότε έχετε εντοπίσει μια ακμή δεδομένων. Η ακμή των δεδομένων ξεκινά από τον κόμβο -εγγραφής και τελειώνει στον κόμβο-ανάγνωσης.
4. Επαναλάβετε τα προηγούμενα βήματα για όλες τις λειτουργίες ανάγνωσης μεταβλητών σε κάθε κόμβο.

Αυτή η διαδικασία αναγνωρίζει όλες τις ακμές δεδομένων που σχετίζονται με τις εντολές εκχώρησης, αλλά όχι εκείνες που προέρχονται από τις εκφράσεις υπό συνθήκη στις εντολές ελέγχου ροής Ωστόσο, αυτές οι ακμές δεδομένων βρίσκονται εύκολα: προέρχονται από τον υπολογισμό της συνθήκης και επηρεάζουν όλες τις λειτουργίες των οποίων η εκτέλεση εξαρτάται από αυτή τη συνθήκη.

Σημείωση: DFG

Ας πάρουμε το γράφο ροής δεδομένων του προγράμματος GCD που δίνεται στην Εικόνα 49. Σύμφωνα με τη διαδικασία, επιλέγουμε έναν κόμβο όπου διαβάζεται μια μεταβλητή. Για παράδειγμα, ο κόμβος 5 στο CFG διαβάζει τις μεταβλητές a και b.

$$b = b - a;$$

Σημείωση: Όταν δηλώνουμε μια μεταβλητή θεωρούμε ότι γράφουμε σε αυτή

Πρώτον, επικεντρωθείτε στον τελεστέο b. Πρέπει να βρούμε όλους τους κόμβους που γράφουν στη μεταβλητή b. Στο CFG, μπορούμε να εντοπίσουμε τους προκατόχους (predecessor) κόμβους για αυτόν τον κόμβο μέχρι να βρεθεί κάποιος που γράφει στη μεταβλητή b. Στους προκάτοχους του κόμβου 5 περιλαμβάνονται: κόμβος 3, κόμβος 2, κόμβος 1, κόμβος 4 και κόμβος 5. Και οι δύο κόμβοι 1 και 5 γράφουν στο b. Επιπλέον, υπάρχει ένα μονοπάτι από τον κόμβο 1 στον 5 (π.χ. 1-2-3-5) και υπάρχει επίσης ένα μονοπάτι από τον κόμβο 5 στον 5 (π.χ. 5-2-3-5) Σε καθένα από αυτά τα μονοπάτια, κανένας άλλος κόμβος δεν γράφει στην b, εκτός από τον τελικό κόμβο 5. Υπάρχει, επομένως μια ακμή δεδομένων για τη μεταβλητή b από τον κόμβο 1 στον 5 και από τον κόμβο 5 στον 5. Ξεκινώντας από τον ίδιο κόμβο ανάγνωσης 5, μπορούμε επίσης να βρούμε όλους τους προκατόχους που καθορίζουν την τιμή του τελεστέου a. Σε αυτήν την περίπτωση, βρίσκουμε ότι οι κόμβοι 1 και 4 γράφουν στη μεταβλητή a, και ότι υπάρχει ένα απευθείας μονοπάτι από τον κόμβο 1 στον 5, καθώς και από τον κόμβο 4 στον 5. Συνεπώς, υπάρχει μια ακμή δεδομένων για τη μεταβλητή a από τον κόμβο 1 στον 5 και από τον κόμβο 4 στον 5.

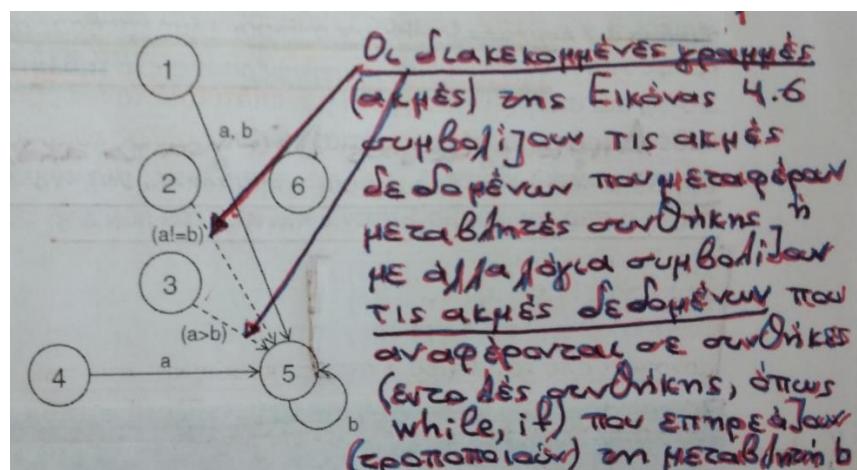
Σημείωση: Εύρεση εκφράσεων συνθηκών

Για να ολοκληρώσουμε το σύνολο των ακμών δεδομένων στον κόμβο 5, πρέπει επίσης να εντοπίσουμε όλες τις εκφράσεις συνθηκών που επηρεάζουν το αποτέλεσμα του κόμβου 5. Εξετάζοντας τις εντολές ελέγχου αυτής της συνάρτησης, βλέπουμε ότι ο κόμβος 5 εξαρτάται από τη συνθήκη που υπολογίζεται στον κόμβο 3 ($a > b$) καθώς και τη συνθήκη που υπολογίζεται στον κόμβο 2 ($a! = b$). Υπάρχει μια ακμή δεδομένων από καθένα από τους κόμβους 2 και 3, στον κόμβο 5, η οποία μεταφέρει το αποτέλεσμα αυτής της συνθήκης. Η συλλογή όλων

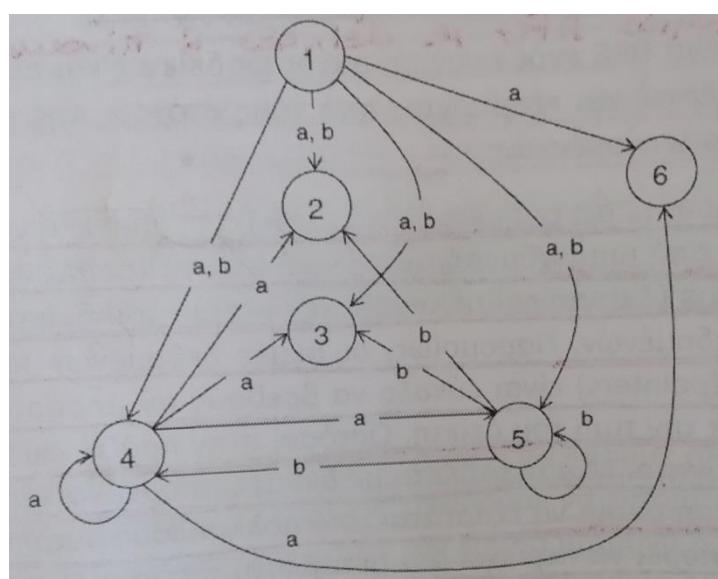
των ακμών δεδομένων στον κόμβο 5 μπορεί τώρα να σχολιασθεί στον DFG, με αποτέλεσμα τον μερικό (partial) DFG της Εικόνας 50. Μπορούμε να επαναλάβουμε αυτή τη διαδικασία για κάθε άλλο κόμβο του γράφου έτσι ώστε να κατασκευάσουμε τον πλήρη DFG. Το αποτέλεσμα αυτής της ανάλυσης φαίνεται στην Εικόνα 51. Αυτός ο γράφος δεν περιέχει τις ακμές δεδομένων που προέρχονται από τις εκφράσεις υπό συνθήκη.

Σημείωση: Σχεδιασμός DFG με δείκτες και πίνακας

Καταρχήν, παρατηρήστε ότι μια μεταβλητή με δείκτες (indexed variable) δεν διαφέρει πραγματικά από μια βαθμωτή μεταβλητή με την προϋπόθεση ότι μπορούμε να υπολογίσουμε (determine) με ακρίβεια την τιμή του δείκτη κατά την ανάλυση της ροής δεδομένων. Παρομοίως, οι ακμές δεδομένων που προκύπτουν από τους δείκτες (pointers) είναι εύκολο να βρεθούν αν μπορούμε να υπολογίσουμε με ακρίβεια την τιμή του δείκτη. Ωστόσο, στην πράξη, αυτό μπορεί να είναι δύσκολο ή αδύνατο. Μια μεταβλητή με δείκτες μπορεί να έχει μια πολύπλοκη έκφραση δείκτη η οποία να εξαρτάται από πολλαπλούς μετρητές βρόχων ή η έκφραση δείκτη μπορεί να περιέχει μια μεταβλητή η οποία είναι άγνωστη κατά το χρόνο μεταγλώττισης.



Εικόνα 50 - Εισερχόμενες ακμές δεδομένων για τον κόμβο 5 στο πρόγραμμα GCD



Εικόνα 51 - Ακμές δεδομένων για όλους τους κόμβους στο πρόγραμμα GSD, εκτός των ακμών που μεταφέρουν τις μεταβλητές συνθήκης

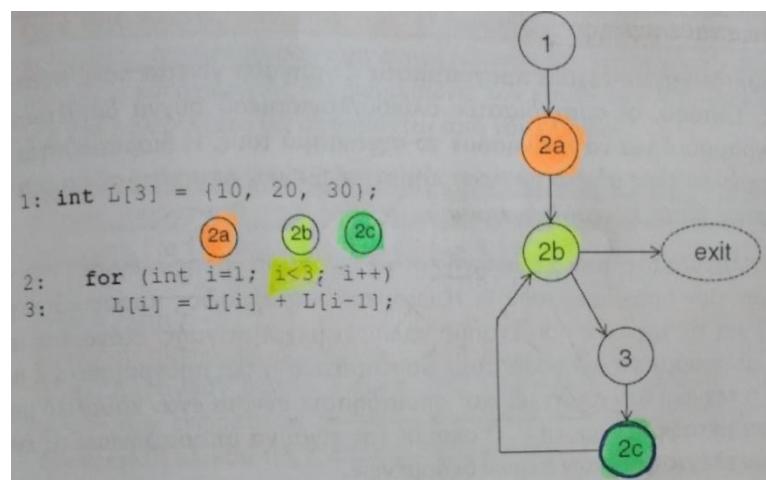
Σημείωση: Δημιουργία ενός DFG για το ακόλουθο πρόγραμμα

Εξηγούμε αυτή την προσέγγιση χρησιμοποιώντας το ακόλουθο παράδειγμα. Ο CFG του παρακάτω βρόχου φαίνεται στην Εικόνα 52.

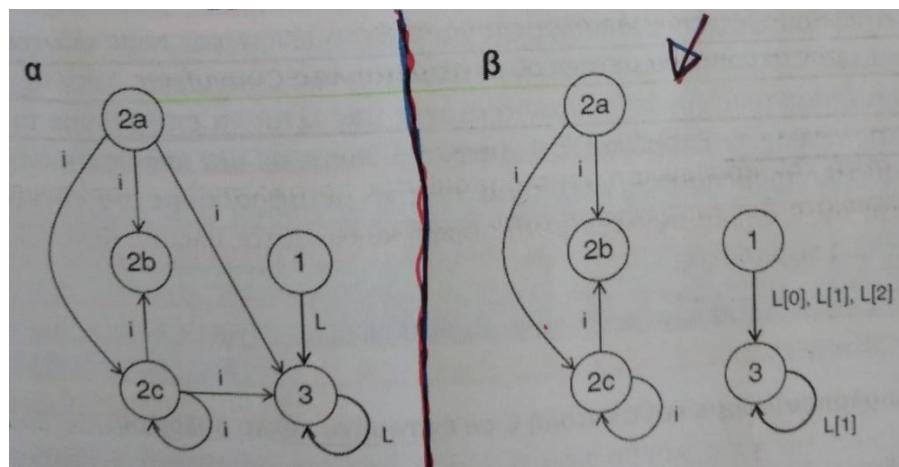
```
int L[3] = {10, 20, 30};  
for (int i = 1; i<3; i++)  
    L[i] = L[i] + L[i - 1];
```

Σημείωση: Χρήση δύο διαφορετικών τρόπων για τη δημιουργία του DFG του παραπάνω προγράμματος

Για να δημιουργήσουμε ένα DFG για αυτό το πρόγραμμα, προχωρούμε όπως προηγουμένως. Για κάθε κόμβο που διαβάζει από μια μεταβλητή, βρίσκουμε τους κόμβους που γράφουν σε αυτή τη μεταβλητή μέσω ενός απευθείας μονοπατιού στο CFG. Όπως συζητήσαμε παραπάνω, μπορούμε να χειριστούμε την ανάλυση της μεταβλητής με δείκτη L με δύο διαφορετικούς τρόπους. a) Στην πρώτη προσέγγιση, βλέπουμε το L ως μία απλή μονολιθική μεταβλητή, έτσι ώστε μια ανάγνωση από οποιαδήποτε θέση από το L να αντιμετωπίζεται ως τμήμα της ίδιας ακμής δεδομένων. b) Στη δεύτερη προσέγγιση διαχωρίζουμε τις επιμέρους θέσεις του L , έτσι ώστε κάθε θέση του L να συμβάλλει σε μια διαφορετική ακμή δεδομένων. Η πρώτη προσέγγιση απεικονίζεται στην Εικόνα 53α, ενώ η δεύτερη προσέγγιση απεικονίζεται στην Εικόνα 53β.



Εικόνα 52 - CFG για έναν απλό βρόχο με μια μεταβλητή με δείκτη



Εικόνα 53 - DFG για έναν απλό βρόχο με μια μεταβλητή με δείκτη



Εφαρμογή: Μεταφράζοντας τη C σε Υλικό

Μια ωραία εφαρμογή της ανάλυσης της ροής-δεδομένων και ροής-ελέγχου σε ένα πρόγραμμα C είναι η συστηματική μετάφραση της C σε υλικό. Αυτό το πρόβλημα είναι εξαιρετικά σύνθετο, όταν ο στόχος μας είναι να επιλύσουμε τη μετάφραση της γενικής C.

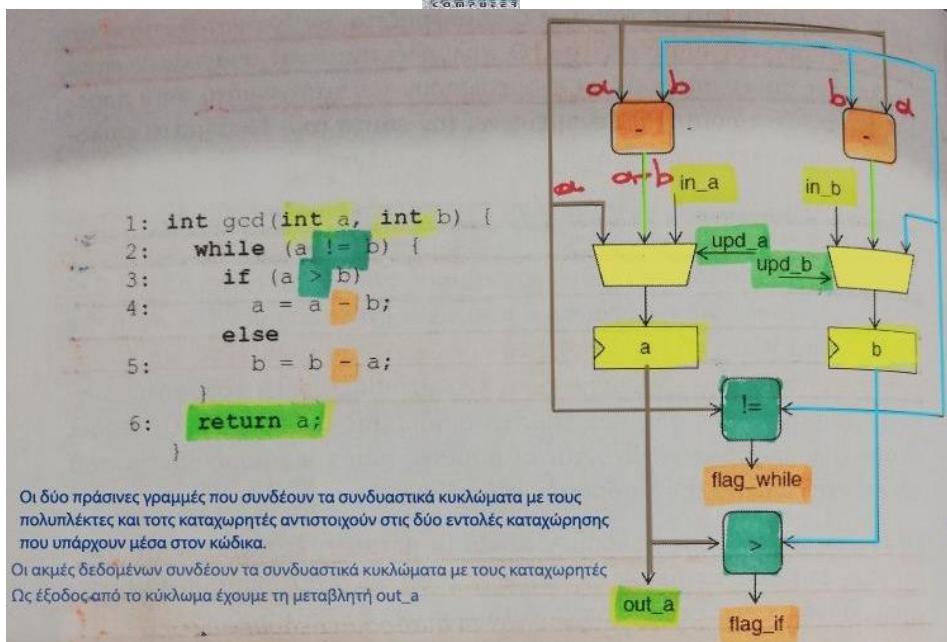
4.1.1 Σχεδιασμός της Διαδρομής Δεδομένων

Με τα CFG και DFG διαθέσιμα, οι ακόλουθοι κανόνες θα καθορίσουν την υλοποίηση της διαδρομής δεδομένων σε υλικό.

1. Κάθε μεταβλητή στο πρόγραμμα C μεταφράζεται σε ένα καταχωρητή με έναν πολυπλέκτη μπροστά του. Ο πολυπλέκτης είναι αναγκαίος, όταν πολλαπλές πηγές μπορούν να ενημερώσουν τον καταχωρητή.
2. Για κάθε έκφραση C ενσωματωμένη σε έναν κόμβο του CFG, δημιουργήστε ένα ισοδύναμο συνδυαστικό κύκλωμα για να υλοποιήσετε αυτήν την έκφραση. Για παράδειγμα, εάν ένας κόμβος στο CFG αντιστοιχεί στη C εντολή $a = b - a$, τότε η έκφραση C που είναι ενσωματωμένη σε αυτή την εντολή είναι $b - a$. Το συνδυαστικό κύκλωμα που απαιτείται για την υλοποίηση αυτής της έκφρασης είναι ένας αφαιρέτης.
3. Το κύκλωμα διαδρομής δεδομένων και οι μεταβλητές καταχωρητή συνδέονται με βάση τις ακμές δεδομένων του DFG. Κάθε λειτουργία εικχώρησης συνδέει ένα συνδυαστικό κύκλωμα με ένα καταχωρητή. Κάθε ακμή δεδομένων συνδέει ένα καταχωρητή με την είσοδο ενός συνδυαστικού κυκλώματος. Τέλος, συνδέουμε επίσης τις εισόδους συστήματος και τις εξόδους συστήματος σε εισόδους κυκλωμάτων της διαδρομής δεδομένων και τις εξόδους καταχωρητή αντίστοιχα.

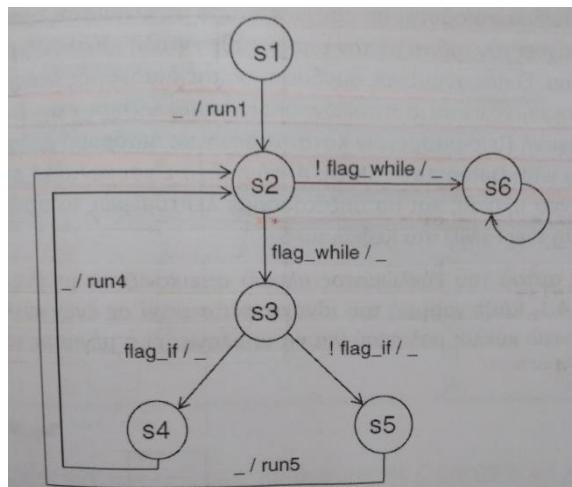
Σημείωση: Υλοποίηση GCD σε υλικό

Το πρόγραμμα GCD μπορεί τώρα να μετατραπεί σε υλοποίηση υλικού ως εξής. Χρειαζόμαστε δύο καταχωρητές, για κάθε μία από τις μεταβλητές a και b. Οι εκφράσεις συνθήκης για τις εντολές if και while χρειάζονται ένα συγκριτή μεγαλύτερου και ένα ισότητας. Οι αφαιρέσεις b-a και a-b υλοποιούνται χρησιμοποιώντας έναν αφαιρέτη. Η συνδεσιμότητα των εξαρτημάτων ορίζεται από τις ακμές δεδομένων του DFG.



Εικόνα 54 - Υλοποίηση υλικού της διαδρομής δεδομένων GCD

Η παραγόμενη διαδρομή δεδομένων έχει δύο εισόδους δεδομένων (in_a και in_b) και μια έξοδο δεδομένων (out_a). Το κύκλωμα απαιτεί να λειτουργούν δύο μεταβλητές ελέγχου (upd_a και upd_b) και παράγει δύο σημαίες (flag_while και flag_if). Οι μεταβλητές ελέγχου και οι σημαίες είναι οι έξοδοι και οι είσοδοι, αντίστοιχα, του ελεγκτή αυτής της διαδρομής δεδομένων βλέπε Εικόνα 54.



Εικόνα 55 - Προδιαγραφές ελέγχου για τη διαδρομή δεδομένων GSD

Μια προδιαγραφή μηχανής πεπερασμένων καταστάσεων (finite-state machine – FSM) για τον αλγόριθμο GSD φαίνεται στην Εικόνα 55. Η αντιστοιχία με το CFG είναι προφανής. Κάθε μία από τις μεταβάσεις σε αυτήν την FSM χρειάζεται ένα κύκλο ρολογιού για να ολοκληρωθεί. Οι δραστηριότητες των FSM εκφράζονται ως πλειάδες συνθήκη/εντολή. Για παράδειγμα, /run1 σημαίνει ότι κατά τη διάρκεια αυτού του κύκλου ρολογιού, οι σημαίες συνθήκης είναι αδιάφορες (don't care), ενώ η εντολή για τη διαδρομή δεδομένων είναι το σύμβολο run1.

Σημείωση: Πρόγραμμα μοναδικής ανάθεσης

Μπορούμε να επιτύχουμε καλύτερα αποτελέσματα με την ακόλουθη τεχνική. Ένα πρόγραμμα C μπορεί να μεταφραστεί σε ένα πρόγραμμα μοναδικής ανάθεσης (single-assignment program). Κάθε μεταβλητή σε αυτό το πρόγραμμα εκχωρείται μια μοναδική φορά μέσα σε ένα μοναδικό λεξικό-γραφικό στιγμιότυπο του



προγράμματος. Ας δείξουμε τη μετατροπή με ένα απλό παράδειγμα. Υποθέτουμε ότι έχουμε ένα απόσπασμα C που φαίνεται ως εξής.

a = a + 1 ;

a = a * 3 ;

a = a - 2 ;

Αυτό το τμήμα κώδικα περιέχει τρεις εκχωρήσεις/αναθέσεις στην a. Χρησιμοποιώντας την προηγούμενη στρατηγική μας, θα χρειαζόμαστε τρεις κύκλους ρολογιού για να εκτελέσουμε αυτό το κομμάτι. Αντ' αυτού, μπορούμε να ξαναγράψουμε αυτό το πρόγραμμα έτσι ώστε κάθε μεταβλητή να εκχωρείται μόνο μια φορά. Αυτό απαιτεί την εισαγωγή πρόσθετων μεταβλητών.

a2 = a1 + 1 ;

a3 = a2 * 3 ;

a4 = a3 - 2 ;

Η διαφορά με το προηγούμενο πρόγραμμα είναι ότι κάθε εκχώρηση αντιστοιχίζεται σε μια διαφορετική λειτουργία ανάγνωσης. Με άλλα λόγια, η μορφή μοναδικής – ανάθεσης του προγράμματος απεικονίζει τις ακμές δεδομένων του προγράμματος στον πηγαίο κώδικα: η εκχώρηση μίας δεδομένης μεταβλητής υποδηλώνει την αρχή μιας ακμής δεδομένων, ενώ η ανάγνωση της ίδιας μεταβλητής υποδεικνύει το τέλος της ακμής δεδομένων. Έτσι, στο προηγούμενο παράδειγμα, ο αριθμός κύκλων μπορεί να μειωθεί απεικονίζοντας τις a2 και a3 σε ένα καλώδιο κρατώντας όμως την a4 σε ένα καταχωρητή. Αυτό θα ομαδοποιούσε τις τρεις εντολές C σε ένα μόνο κύκλο ρολογιού.

Κατά τη μετατροπή των προγραμμάτων C σε μορφή μοναδικής – ανάθεσης, πρέπει να ληφθούν υπόψη όλες οι εκχωρήσεις σε μια μεταβλητή.

Σημείωση: Δημιουργία ασάφειας

Ας εξετάσουμε το ακόλουθο παράδειγμα: έναν βρόχο που υπολογίζει το άθροισμα των αριθμών 1-5.

a = 0;

for (i = 1; i < 6; i++)

 a = a + i;

Στη μορφή μοναδικής – ανάθεσης, οι εκχωρήσεις στην a μπορούν να γίνουν μοναδικές, αλλά παραμένει ασαφές ποια έκδοση του a πρέπει να διαβαστεί μέσα στο βρόχο.

a1 = 0;

for (i = 1; i < 6; i++)

 a2 = a + i; // ποια έκδοση του a να διαβάσω;

Η απάντηση είναι ότι τόσο η a1 όσο και η a2 αποτελούν έγκυρες λύσεις για αυτό το πρόγραμμα: εξαρτάται από την επανάληψη εντός του βρόχου. Όταν μπαίνουμε για πρώτη φορά τον βρόχο, θα γράψουμε:

a2 = a1 + 1;

Μετά την πρώτη επανάληψη βρόχου, αντιθέτως θα γράφαμε:

a2 = a2 + i; // όταν $i > 1$

Σημείωση: Μοναδική ανάθεση – Επίλυση προηγούμενης ασάφειας

Για την επίλυση αυτής της ασάφειας, τα προγράμματα μοναδικής-ανάθεσης χρησιμοποιούν μια συνάρτηση merge, μια λειτουργία που μπορεί να συγχωνεύσει πολλαπλές ακμές δεδομένων σε μία.

a1 = 0;

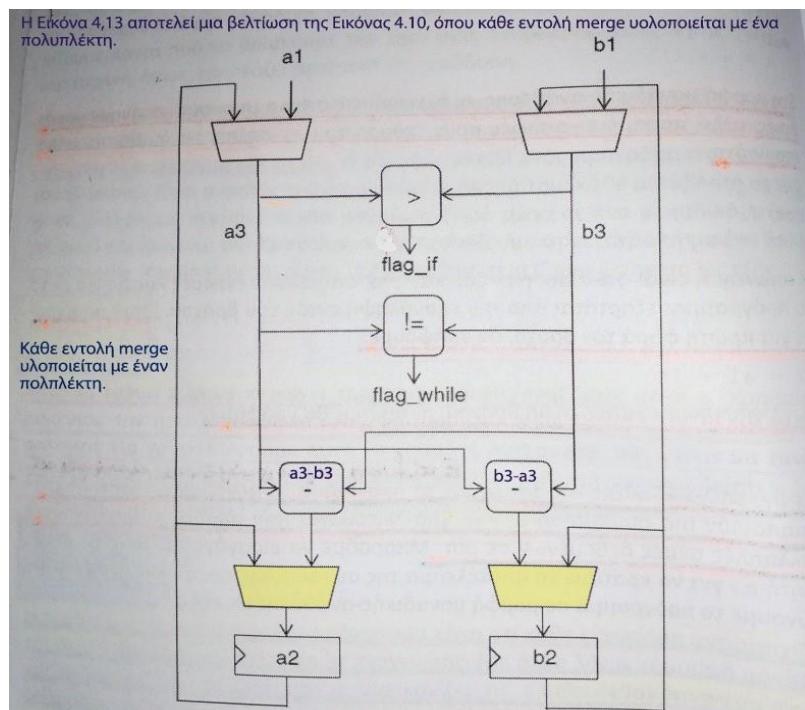
for (i = 1; i < 6; i++) {

 a3 = merge (a2, a1);

 a2 = a3+ i;

Το ισοδύναμο υλικού της συνάρτησης merge θα ήταν ένας πολυπλέκτης υπό τον έλεγχο ενός κατάλληλου σήματος επιλογής. Σε αυτή την περίπτωση, ($i == 0$) θα ήταν ένα κατάλληλο σήμα επιλογής. Οι παραπάνω κανόνες μετάφρασης μπορούν να χρησιμοποιηθούν και σε πιο περίπλοκα προγράμματα. Για παράδειγμα, το πρόγραμμα GCD μπορεί να μετατραπεί ως εξής.

```
int gcd (int a, int b) {
    while (a != b) {
        if (a > b)
            a = a - b;
        else
            b = b - a;
    }
    return a;
}
```



Εικόνα 56 - Διαδρομή δεδομένων GCD από κύδικα μοναδικής - ανάθεσης



Σημείωση: Περιγραφή εικόνας 56

Η ισοδύναμη μορφή μοναδικής ανάθεσης του GSD φαίνεται παρακάτω. Η έκφραση συνθήκης στην εντολή while χρησιμοποιεί μεταβλητές είτε από την είσοδο της συνάρτησης είτε από το σώμα του βρόχου. Για το λόγο αυτό, η έκφραση συνθήκης χρησιμοποιεί τη συνάρτηση merge ως τελεστέο.

```
int gcd (int a1, int b1) {  
    while (merge (a1, a2) != merge (b1, b2) ) {  
        a3 = merge (a1, a2);  
        b3 = merge (b1, b2);  
        if (a3 > b3)  
            a2 = a3 - b3;  
        else  
            b2 = b3 - a3;  
    }  
    return a2;  
}
```

Ένα πρόγραμμα μοναδικής ανάθεσης όπως αυτό είναι πολύτιμο, επειδή οπτικοποιεί τις ακμές δεδομένων στον πηγαίο κώδικα του προγράμματος, καθιστώντας την σύνδεση με το υλικό πιο προφανή. Επιπλέον, οι συναρτήσεις merge μπορούν να απεικονιστούν σε πολυπλέκτες στο υλικό. Μια πιθανή διαδρομή δεδομένων που αντιστοιχεί σε αυτή την έκδοση μοναδικής ανάθεσης ενός GCD απεικονίζεται στην Εικόνα 56. Αυτή η διαδρομή δεδομένων μοιάζει πολύ με το προηγούμενο σχεδιασμό (Εικόνα 54) αλλά αυτός ο σχεδιασμός προέρχεται από ένα πρόγραμμα C με τέσσερις εκχωρήσεις. Μόνο οι a2 και b2 απεικονίστηκαν σε ένα καταχωρητή, ενώ οι άλλες μεταβλητές είναι απλά καλώδια. Ο σχεδιασμός στην Εικόνα 56 εκτελεί πολλαπλές εντολές C ανά κύκλο ρολογιού.

Προβλήματα

Πρόβλημα 4.1 Κάντε τα παρακάτω για το πρόγραμμα C στη λίστα 4.1

Λίστα 4.1 Πρόγραμμα για το Πρόβλημα 4.1

```
int addall (int a, int b, int c, int d) {  
    a = a + b;  
    a = a + c;  
    a = a + d;  
    return a;  
}
```

(α) Εξάγετε και σχεδιάστε τους CFG και DFG

(β) Το μήκος ενός μονοπατιού σε ένα γράφο ορίζεται ως ο αριθμός των ακμών σε αυτό το μονοπάτι. Βρείτε το μακρύτερο μονοπάτι στο DFG.



(γ) Ξαναγράψτε το πρόγραμμα στη Λίστα 4.1 έτσι ώστε να μειώνεται το μέγιστο μήκος μονοπατιού στο DFG. Υποθέστε ότι μπορείτε να εκτελέσετε μόνο μία αριθμητική λειτουργία ανά εντολή C. Σχεδιάστε το DFG που προκύπτει.

Πρόβλημα 4.2 Σχεδιάστε τους CFG και DFG του προγράμματος στη Λίστα 4.2. Συμπεριλαμβάνετε όλες τις εξαρτήσεις ελέγχου στο CFG. Συμπεριλάβετε τις εξαρτήσεις δεδομένων για τις μεταβλητές a και b του DFG.

Λίστα 4.2 Πρόγραμμα για το Πρόβλημα 4.2

```
int count (int a, int b) {  
    while (a < b)  
        a = a * 2;  
    return a + b;  
}
```

Πρόβλημα 4.3 Σχεδιάστε μια διαδρομή δεδομένων στο υλικό για το πρόγραμμα που εμφανίζεται στη Λίστα 4.3. Εκχωρήστε καταχωρητές και τελεστές. Υποδείξτε τις εισόδους ελέγχου που απαιτούνται από τη διαδρομή δεδομένων και τις σημαίες συνθηκών που παράγονται από τη διαδρομή δεδομένων.

Λίστα 4.3 Πρόγραμμα για το Πρόβλημα 4.3

```
unsigned char mysqrt (unsigned int N) {  
    unsigned int x, j;  
    x = 0;  
    for (j = 1<<7; j != 0; j>>=1) {  
        x = x + j;  
        if ( x*x > N)  
            x = x - j;
```

5 Κεφάλαιο - Μηχανή Πεπερασμένων Καταστάσεων με Διαδρομή Δεδομένων Υλικό Παράλληλων Bit Βασισμένο σε Κύκλους

5.1.1 Απεικόνιση Εκφράσεων σε Υλικό

Για κάθε έκφραση που περιλαμβάνει σήματα και καταχωρητές καθορισμένου πρόσημου και ακρίβειας, υπάρχει ένα ισοδύναμο κύκλωμα υλικού.

Αριθμητικές Πράξεις. Η πρόσθεση (+), η αφαίρεση (-), ο πολλαπλασιασμός (*) χρησιμοποιούνται συνήθως στο σχεδιασμό υλικού διαδρομής δεδομένων.

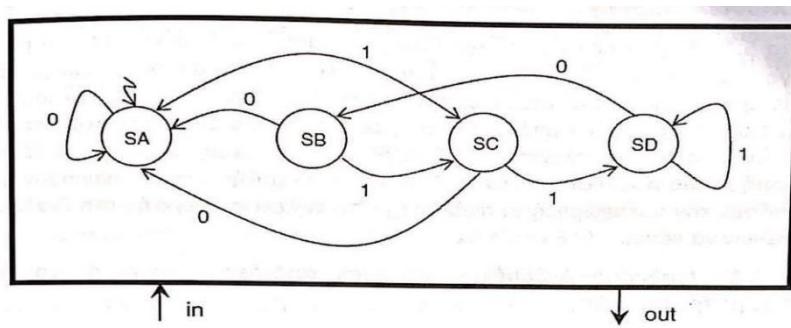
Πράξεις σε Bit. Όλες οι πράξεις διαχείρισης δυαδικών ψηφίων (bitwise operations), συμπεριλαμβανομένων των AND (&), OR (|), XOR (^) και NOT (~) έχουν άμεσο ισοδύναμο σε λογικές πύλες.

Πράξεις σύγκρισης. Όλες οι πράξεις σύγκρισης επιστρέφουν ένα μόνο μη προσημασμένο bit (ns (1)). Αυτές οι πράξεις χρησιμοποιούν έναν αφαιρέτη για να συγκρίνουν δύο αριθμούς και στη συνέχεια χρησιμοποιούν τις σημαίες προσήμου/υπερχείλισης του αποτελέσματος για να εκτιμήσουν το αποτέλεσμα της σύγκρισης.

Σε αυτή την παράγραφο, περιγράφουμε ένα σύνθετο μοντέλο ελέγχου για την περιγραφή του υλικού, που ονομάζεται Μηχανή Πεπερασμένων Καταστάσεων (Finite State Machine – FSM). Μια FSM μπορεί να χρησιμοποιηθεί για την περιγραφή της παραγωγής αλληλουχιών (sequencing) και της λήψης αποφάσεων.

Μια FSM είναι μια ακολουθιακή ψηφιακή μηχανή η οποία χαρακτηρίζεται από:

- Ένα σύνολο καταστάσεων
- Ένα σύνολο εισόδων και εξόδων
- Μια συνάρτηση μετάβασης καταστάσεων
- Μια συνάρτηση εξόδου



Εικόνα 57 - Mealy FSM για αναγνώριση του μοτίβου '110'

Μηχανές Πεπερασμένων Καταστάσεων με Διαδρομή Δεδομένων

Σημείωση: FSMD

Μια Μηχανή Πεπερασμένων Καταστάσεων με Διαδρομή Δεδομένων (Finite StateMachine with Datapath) συνδυάζει ένα μοντέλο ελέγχου υλικού (μια FSM) με μια διαδρομή δεδομένων. Η διαδρομή δεδομένων



περιγράφεται ως βασισμένο-σε-κύκλους υλικό παράλληλων-bit χρησιμοποιώντας εκφράσεις. Ωστόσο, σε αντίθεση με τις διαδρομές δεδομένων που χρησιμοποιούν μόνο μπλοκ always, οι διαδρομές δεδομένων FSMD ορίζουν επίσης μία ή περισσότερες εντολές: υπό συνθήκη εκτελούμενα μπλοκ 'always'.

Σημείωση: FSMD

- Οι καταστάσεις μετάβασης καταστάσεων μια FSM πρέπει να αποθηκευτούν σε καταχωρητές, γεγονός που εισάγει λανθάνοντα χρόνο ενός κύκλου ρολογιού για κάθε υπό συνθήκη μετάβαση κατάστασης. Όταν καταγράφετε μια FSM με εκφράσεις διαδρομής δεδομένων, οι συνθήκες μετάβασης καταστάσεων μπορούν να δημιουργηθούν και να υπολογιστούν στον ίδιο κύκλο ρολογιού.
- Κατά την αποτύπωση της FSMD σε μια διαδρομή δεδομένων, οι εκφράσεις στη διαδρομή δεδομένων περιλαμβάνουν χρονοδρομολόγηση καθώς και επεξεργασία δεδομένων. Από την άλλη, σε μια FSMD με ξεχωριστές περιγραφές της FSM και της διαδρομής δεδομένων, οι εκφράσεις διαδρομής δεδομένων, αντιπροσωπεύουν μόνο την επεξεργασία δεδομένων.

Σημείωση: Σε ποιους σχεδιασμούς χρησιμοποιείται η FSMD

- Μια FSMD που αποτυπώνεται ως μια ενιαία διαδρομή δεδομένων είναι καλή για σχεδιασμούς με απλό ή καθόλου έλεγχο χρονοδρομολόγησης, όπως οι σχεδιασμοί με απαιτήσεις υψηλής απόδοσης.

Σημείωση: Αποφάσεις που λαμβάνει ένας σχεδιαστής που αναπτύσσει μια FSMD

Ένας σχεδιαστής που αναπτύσσει μια FSMD θα λάβει τις ακόλουθες γενικές αποφάσεις

- Ο σχεδιαστής καθορίζει την ποσότητα της εργασίας που γίνεται σε ένα κύκλο ρολογιού. Αυτή είναι απλά το σύνολο όλων των εντολών διαδρομής δεδομένων που θα εκτελούνται ταυτόχρονα σε ένα κύκλο ρολογιού. Ως εκ τούτου, προκειμένου να επιτευχθεί το καλύτερο δυνατό μοίρασμα, ένας σχεδιαστής πρέπει να διανείμει παρόμοιες λειτουργίες σε πολλαπλούς κύκλους ρολογιού. Για παράδειγμα, αν υπάρχουν 16 πολλαπλασιασμοί που πρέπει να εκτελεστούν με έναν προϋπολογισμό κύκλων ρολογιού ίσο με τέσσερεις κύκλους, τότε μια υλοποίηση με 4 πολλαπλασιασμούς σε κάθε κύκλο ρολογιού θα είναι πιθανότητα μικρότερη από μία που εκτελεί 16 πολλαπλασιασμούς στον πρώτο κύκλο ρολογιού και τίποτα στους επόμενους τρεις κύκλους.

Σημείωση: Πως μπορεί μια FSMD να οδηγήσει σε καλύτερη απόδοση υλικού

- Ο σχεδιαστής μπορεί επίσης να επηρεάσει, έμμεσα, τη μέγιστη συχνότητα ρολογιού που μπορεί να επιτευχθεί από το σχεδιασμό υλικού. Αυτή η συχνότητα καθορίζεται από την πολυπλοκότητα των εκφράσεων που δίνονται στις εντολές διαδρομής δεδομένων. Προφανώς, μικρές και σύντομες εκφράσεις θα έχουν ως αποτέλεσμα μικρότερη και ταχύτερη λογική στη διαδρομή δεδομένων. Εάν οι εκφράσεις που χρησιμοποιούνται για την αποτύπωση της διαδρομής δεδομένων γίνουν υπερβολικά περίπλοκες, ο παραγόμενος σχεδιασμός μπορεί να είναι πολύ αργός για την επιθυμητή περίοδο ρολογιού του συστήματος.



Σύνοψη

Σε αυτήν την παράγραφο συζητήσαμε ένα σύγχρονο μηχανισμό μοντελοποίησης υλικού, που αποτελείται από μια διαδρομή δεδομένων σε συνδυασμό με έναν ελεγκτή FSM. Το παραγόμενο μοντέλο ονομάζεται FSMD (Μηχανή Πεπερασμένων Καταστάσεων με Διαδρομή Δεδομένων). Ένα μοντέλο FSMD μοντελοποιεί τις εντολές της διαδρομής δεδομένων με εκφράσεις και ελέγχει με ένα γράφο μετάβασης καταστάσεων. Οι εκφράσεις διαδρομής δεδομένων δημιουργούνται με χρήση μεταβλητών καταχωρητή και σημάτων (καλωδίων). Οι μεταβλητές καταχωρητών συνδέονται έμμεσα στο καθολικό σήμα ρολογιού. Οι εντολές διαδρομής δεδομένων (ομάδες εκφράσεων διαδρομής δεδομένων) αποτελούν τη σύνδεση μεταξύ του ελεγκτή και της διαδρομής δεδομένων.

Ένα δεδομένος σχεδιασμός FSMD δεν είναι μοναδικός.

Το μοντέλο FSMD είναι γενικό και μπορεί να αποτυπωθεί σε οποιαδήποτε κατάλληλη γλώσσα περιγραφής υλικού.

6 Κεφάλαιο - Μικροπρογραμματιζόμενες Αρχιτεκτονικές

Περιορισμοί Μηχανών Πεπερασμένων Καταστάσεων

Σημείωση: Πλεονεκτήματα και μειονεκτήματα των FSM

Οι μηχανές πεπερασμένων καταστάσεων είναι κατάλληλες για την αποτύπωση της ροής ελέγχου και των αλγορίθμων λήψης αποφάσεων. Τα διαγράμματα μετάβασης καταστάσεων των FSM μοιάζουν ακόμη και με γράφους εξαρτήσεων ελέγχου (control dependency graphs – CDG). Παρόλα αυτά, οι FSM δεν αποτελούν καθολική λύση για τη μοντελοποίηση ελέγχου. Χαρακτηρίζονται από αρκετές αδυναμίες μοντελοποίησης, ειδικά όταν ασχολούνται με σύνθετους ελεγκτές.

Σημείωση: Αδυναμίες μοντελοποίησης των FSM

Σημείωση: 1^ο Μειονέκτημα FSM

FSM είναι ένα επίπεδο μοντέλο, χωρίς κάποια ιεραρχία. Ένα επίπεδο (flat) μοντέλο ελέγχου είναι σαν ένα πρόγραμμα C που αποτελείται από μία μόνο συνάρτηση main. Τα πραγματικά συστήματα δεν χρησιμοποιούν ένα επίπεδο μοντέλο ελέγχου: χρειάζονται μια ιεραρχία ελέγχου. Πολλοί από τους περιορισμούς των FSMs οφείλονται στην έλλειψη ιεραρχίας.

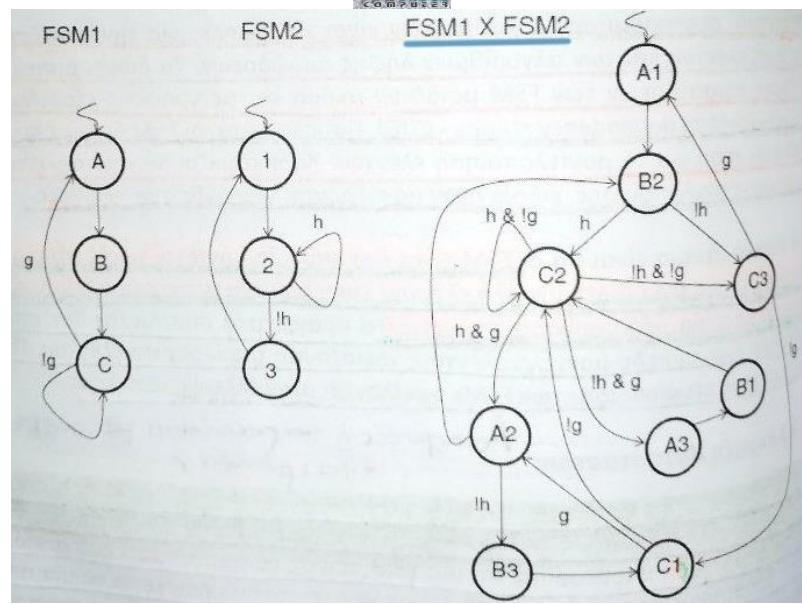
6.1.1 Έκρηξη Καταστάσεων

Σημείωση: Αναλυτική παρουσίαση μειονεκτημάτων εξαιρέσεων FSM

Σημείωση: 2^ο Μειονέκτημα FSM

Μια επίπεδη FSM πάσχει από έκρηξη καταστάσεων (state explosion), η οποία συμβαίνει όταν πολλαπλές ανεξάρτητες δραστηριότητες συνδυάζονται σε ένα ενιαίο μοντέλο. Ας υποθέσουμε ότι μια FSM πρέπει να αποτυπώσει δύο ανεξάρτητες δραστηριότητες, καθεμία από τις οποίες μπορεί να βρίσκεται σε μία από τις τρεις καταστάσεις. Η παραγόμενη FSM, που ονομάζεται γινόμενο μηχανών καταστάσεων (product state-machine), χρειάζεται εννέα καταστάσεις για να καταγράψει τη ροή ελέγχου του συνολικού μοντέλου. Το γινόμενο μηχανών – καταστάσεων πρέπει να παρακολουθεί την τρέχουσα κατάσταση από δύο ανεξάρτητες μηχανές καταστάσεων ταυτόχρονα. Η Εικόνα 58 απεικονίζει την επίπτωση της έκρηξης καταστάσεων σε ένα γινόμενο μηχανών – καταστάσεων.

Αυτό έχει ως αποτέλεσμα πολλαπλές ενδιάμεσες καταστάσεις όπως οι A1, A2 και A3.



Εικόνα 58 - Έκρηξη καταστάσεων σε FSM κατά τη δημιουργία ενός γινομένου μηχανών-καταστάσεων

6.1.2 Διαχείριση Εξαιρέσεων (Exception Handling)

Σημείωση: Ορισμός εξαίρεσης

Σημείωση: 3^o μειονέκτημα

Ένα δεύτερο ζήτημα με ένα επίπεδο FSM είναι η προβληματική διαχείριση εξαιρέσεων. Μια εξαίρεση είναι μια συνθήκη που μπορεί να προκαλέσει άμεση μετάβαση κατάστασης, ανεξάρτητα από την τρέχουσα κατάσταση της μηχανής πεπερασμένων καταστάσεων.

Σημείωση: Σκοπός εξαίρεσης

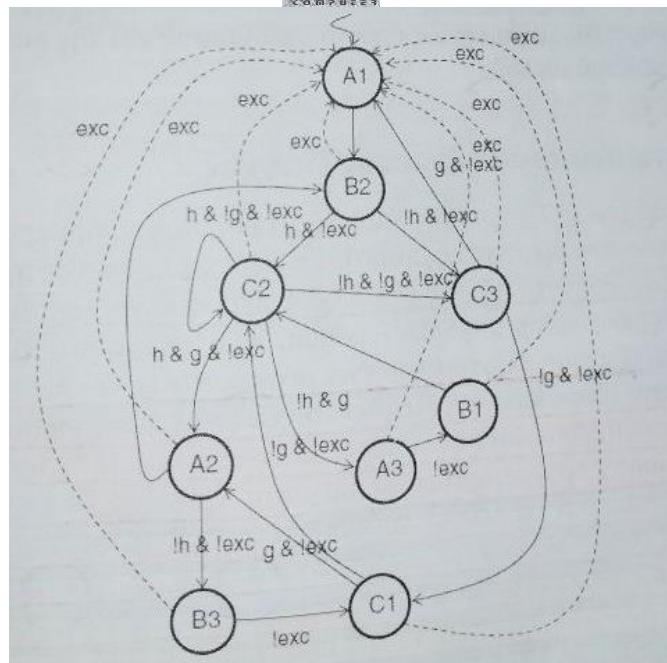
Ο σκοπός μιας εξαίρεσης είναι να σταματήσει την κανονική ροή ελέγχου και να μεταφέρει τον έλεγχο σε έναν αφοσιωμένο χειριστή εξαιρέσεων (dedicated exception-handler). Μια εξαίρεση μπορεί να έχει εσωτερικές αιτίες, όπως μια κατάσταση υπερχείλισης σε μια διαδρομή δεδομένων, ή εξωτερικές αιτίες, όπως μια διακοπή (interrupt). Ανεξάρτητα από την αιτία, το αποτέλεσμα μιας εξαίρεσης σε ένα μοντέλο μηχανής πεπερασμένων καταστάσεων είναι δραματικό: μια πρόσθετη μετάβαση κατάστασης πρέπει να προστεθεί σε κάθε κατάσταση της μηχανής πεπερασμένων καταστάσεων.

Σημείωση: Παράδειγμα εξαίρεσης

Για παράδειγμα, υποθέστε ότι το γινόμενο μηχανών – καταστάσεων στην Εικόνα 58 πρέπει να περιλαμβάνει μια είσοδο εξαίρεσης που ονομάζεται exc και ότι η δήλωση (assertion) αυτής της εισόδου απαιτεί άμεση μετάβαση στην κατάσταση A1.

Σημείωση: Συνέπεια εξαίρεσης

Η παραγόμενη FSM, που φαίνεται στην Εικόνα 59, δείχνει πως οι εξαιρέσεις υποβαθμίζουν τη σαφήνεια του γράφου μετάβασης καταστάσεων της FSM.



Εικόνα 59 - Η προσθήκη μιας απλής καθολικής εξαιρεσης επιδεινώνει σημαντικά την αναγνωσιμότητα της FSM

6.1.3 Ευελιξία χρόνου εκτέλεσης

Σημείωση: Έλλειψη ευελιξίας από FSM

Σημείωση: 4^o μειονέκτημα

Ένας σημαντικός προβληματισμός, από την άποψη της συσχεδίασης υλικού / λογισμικού, είναι ότι μια μηχανή πεπερασμένων καταστάσεων είναι ένα μη ευέλικτο μοντέλο. Μόλις καθοριστούν οι καταστάσεις και οι μεταβάσεις καταστάσεων, η ροή ελέγχου της FSM έχει σταθεροποιηθεί. Η υλοποίηση υλικού μιας FSM οδηγεί σε έναν καλωδιωμένο (hardwired) ελεγκτή που δεν μπορεί να τροποποιηθεί μετά την υλοποίηση.

Σημείωση: Δημιουργία ευελιξίας με μικροπρογραμματισμό

Ως αποτέλεσμα, οι σχεδιαστές πρότειναν βελτιωμένες τεχνικές για την προδιαγραφή και την υλοποίηση του ελέγχου, προκειμένου να αντιμετωπίσουν την ευελιξία, τις εξαιρέσεις και την ιεραρχική μοντελοποίηση. Ο μικροπρογραμματισμός (microprogramming) είναι μια τέτοια τεχνική.

Μικροπρογραμματιζόμενος Έλεγχος

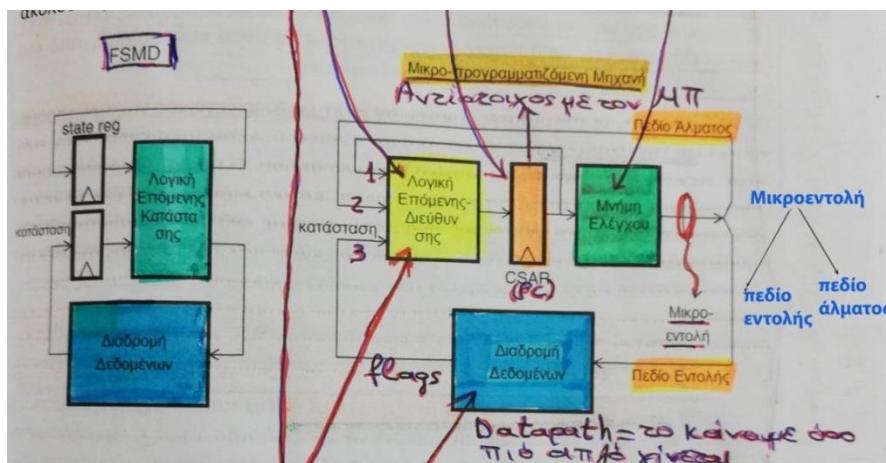
Η Εικόνα 60 δείχνει μια μικροπρογραμματιζόμενη μηχανή δίπλα σε μια FSMD. Η θεμελιώδης ιδέα στο μικροπρογραμματισμό είναι να αντικαταστήσουμε τη λογική της επόμενης – κατάστασης της μηχανής πεπερασμένων καταστάσεων με μια προγραμματιζόμενη (programmable) μνήμη, που ονομάζεται μνήμη ελέγχου (control store).

Η μνήμη ελέγχου περιέχει μικρο-εντολές και διευθυνσιοδοτείται χρησιμοποιώντας έναν καταχωρητή που ονομάζεται Καταχωρητής Διεύθυνσης Μνήμης Ελέγχου (CSAR – Control Store Address Register). Ο CSAR είναι το ισοδύναμο ενός μετρητή προγραμμάτων σε μικροεπεξεργαστές.

Η επόμενη τιμή του CSAR καθορίζεται από τη λογική επόμενης – διεύθυνσης, χρησιμοποιώντας την τρέχουσα τιμή του CSAR, την τρέχουσα μικρο-εντολή και την τιμή των σημαιών κατάστασης που υπολογίζονται από τη διαδρομή δεδομένων.

Η προεπιλεγμένη επόμενη-τιμή του CSAR αντιστοιχεί στην προηγούμενη τιμή του CSAR αυξημένη κατά ένα. Ωστόσο, η λογική επόμενης – διεύθυνσης μπορεί επίσης να υλοποιήσει άλματα (jumps) υπό συνθήκη ή άμεσα άλματα.

Επομένως, η λογική επόμενης – διεύθυνσης ο CSAR και η μνήμη ελέγχου υλοποιούν το ισοδύναμο ενός κύκλου ανάκλησης – εντολής σε έναν μικροεπεξεργαστή.



Εικόνα 60- Σε αντίθεση με το βασισμένο σε FSM έλεγχο, ο μικροπρογραμματισμός χρησιμοποιεί ένα ευέλικτο σχήμα ελέγχου

Σημείωση: Λειτουργίες CSAR

Σημείωση: Πεδία μικροεντολής

- Ο CSAR παρέχει μια διεύθυνση στη μνήμη ελέγχου, η οποία ανακτά μια μικρο-εντολή. Η μικρο-εντολή χωρίζεται σε δύο μέρη: ένα πεδίο-εντολής και ένα πεδίο-άλματος. Το πεδίο-εντολής χρησιμεύει ως εντολή για τη διαδρομή-δεδομένων. Το πεδίο-άλματος πηγαίνει στη λογική επόμενης-διεύθυνσης.
- Η διαδρομή δεδομένων εκτελεί την εντολή που είναι κωδικοποιημένη στη μικρο-εντολή και επιστρέφει τις πληροφορίες κατάστασης στη λογική επόμενης-διεύθυνσης.

Σημείωση: Συνδυασμός 3 πραγμάτων από τη λογική

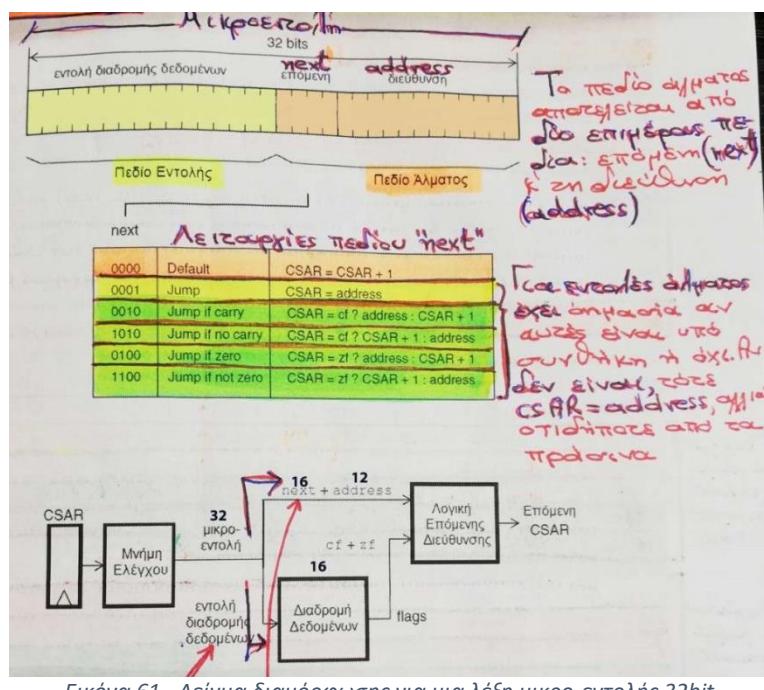
- Η λογική επόμενης – διεύθυνσης συνδυάζει το πεδίο άλματος της μικροεντολής, την προηγούμενη τιμή του CSAR και την κατάσταση της διαδρομής δεδομένων. Η λογική επόμενης - διεύθυνσης ενημερώνει τον CSAR. Συνεπώς, το κρίσιμο μονοπάτι της μικροπρογραμματιζόμενης μηχανής στην Εικόνα 60 καθορίζεται από τη συνδυασμένη καθυστέρηση της λογικής μέσα από την μνήμη ελέγχου, τη λογική επόμενης – διεύθυνσης και τη διαδρομή – δεδομένων.

Αν και ο μικροπρογραμματιζόμενος ελεγκτής είναι πιο περίπλοκος από μια μηχανή πεπερασμένων καταστάσεων, αντιμετωπίζει επίσης τα προβλήματα των FSM.

Σημείωση: Πλεονεκτήματα μικροπρογραμματιζόμενου ελεγκτή

- Ο μικροπρογραμματιζόμενος ελεγκτής κλιμακώνεται (scale) καλά με την πολυπλοκότητα.
- Μια μικροπρογραμματιζόμενη μηχανή διαχειρίζεται πολύ καλά την ιεραρχία ελέγχου.
- Μια μικροπρογραμματιζόμενη μηχανή μπορεί να αντιμετωπίσει αποτελεσματικά τη διαχείριση των εξαιρέσεων, καθώς η λογική επόμενης – διεύθυνσης διαχειρίζεται άμεσα τις καθολικές εξαιρέσεις (global exception) ανεξάρτητα από την μνήμη ελέγχου.
- Τέλος, τα μικροπρογράμματα είναι ευέλικτα και αλλάζουν πολύ εύκολα μετά την υλοποίηση της μικροπρογραμματιζόμενης μηχανής. Για να αλλάξετε το πρόγραμμα της μηχανής αρκεί απλά να αλλάξετε τα περιεχόμενα της μνήμης ελέγχου.

6.1.4 Πεδίο Άλματος



Εικόνα 61 - Δείγμα διαμόρφωσης για μια λέξη μικρο-εντολής 32bit

Σημείωση: Σημασία πεδίου "next" και πεδίου "address"

Το σχήμα 61 δείχνει μια λέξη μικρο-εντολής 32bit, με 16 bit προοριζόμενα για τη διαδρομή δεδομένων και 16 bits κρατημένα για τη λογική επόμενης – διεύθυνσης. Ας εξετάσουμε πρώτα το μέρος για τη λογική επόμενης – διεύθυνσης. Το πεδίο address κρατά μια απόλυτη διεύθυνση στόχου, δείχνοντας σε μια θέση στη μνήμη ελέγχου. Σε αυτήν την περίπτωση, η διεύθυνση είναι 12 bit, πράγμα που σημαίνει ότι αυτή η διαμόρφωση μικρο-εντολής θα ήταν κατάλληλη για μια μνήμη ελέγχου με 4096 θέσεις. Το πεδίο next κωδικοποιεί τη λειτουργία που θα οδηγήσει στην επόμενη τιμή του CSAR. Η προεπιλεγμένη λειτουργία είναι η αύξηση του CSAR. Για τέτοιες εντολές, το πεδίο διεύθυνσης παραμένει αχρησιμοποίητο. Διαφορετικά, το πεδίο next θα χρησιμοποιηθεί για την κωδικοποίηση διαφόρων εντολών άλματος. Ένα απόλυτο άλμα θα μεταφέρει την τιμή του πεδίου address στον CSAR. Ένα άλμα υπό συνθήκη θα χρησιμοποιήσει την τιμή μίας σημαίας για να ενημερώσει υπό συνθήκη τον CSAR ή διαφορετικά να τον αυξήσει.

Το πεδίο address για παράδειγμα, χρησιμοποιείται μόνο για εντολές άλματος.

6.1.5 Πεδίο Εντολής

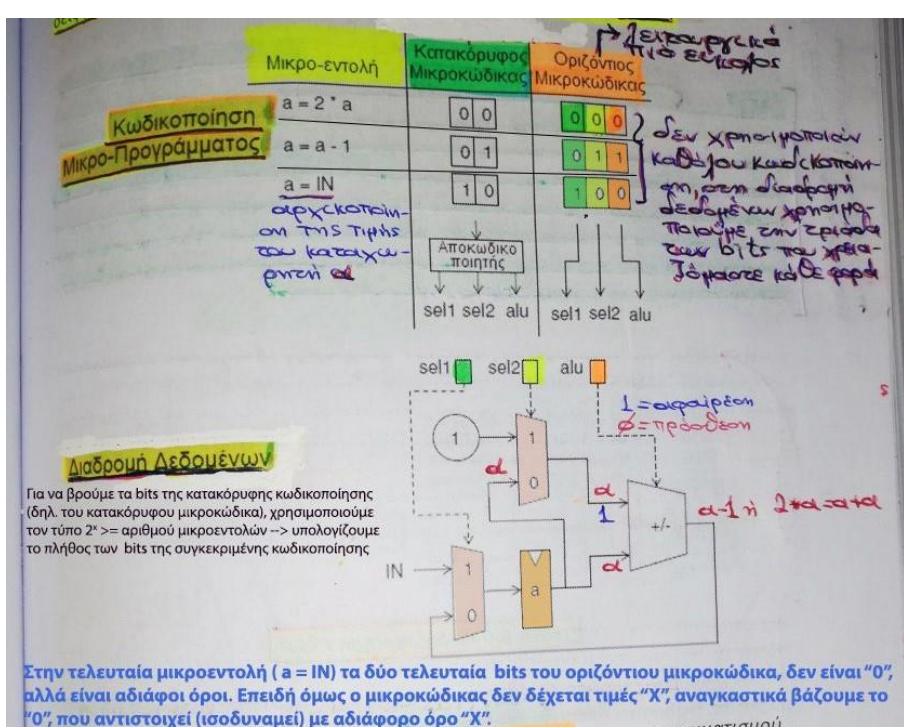
Σημείωση: Μεγάλου και μικρού μήκους μικροεντολές

Μπορούμε να επιλέξουμε μια πολύ μεγάλου μήκους (wide) λέξη μικρο-εντολής, είτε αλλιώς μπορούμε να προτιμήσουμε μια μικρού μήκους (narrow) λέξη μικρο-εντολής. Μια μεγάλου μήκους λέξη μικρο-εντολής επιτρέπει σε κάθε bit ελέγχου της διαδρομής δεδομένων να αποθηκεύεται χωριστά. Μία μικρού μήκους λέξη μικρο-εντολής, από την άλλη, θα απαιτήσει τη δημιουργία "συμβολικών εντολών", οι οποίες είναι κωδικοποιημένες ομάδες bit ελέγχου για τη διαδρομή δεδομένων.

Κάθε μία από τις παραπάνω προσεγγίσεις έχει ένα συγκεκριμένο όνομα. *Oι οριζόντιες μικρο-εντολές δε χρησιμοποιούν καθόλου κωδικοποίηση.* Αναπαριστούν κάθε bit ελέγχου στη διαδρομή δεδομένων με ένα ξεχωριστό bit στη διαμόρφωση μικρο-εντολής. *Oι κατακόρυφες μικρο-εντολές από την άλλη, κάνουν όσο το δυνατόν περισσότερη κωδικοποίηση των bit ελέγχου για τη διαδρομή δεδομένων.* Μερικά κομμάτια της μικρο-εντολής μπορούν να ορίσουν την τιμή πολλών περισσότερων bit ελέγχου στη διαδρομή δεδομένων.

Σημείωση: Δημιουργία μικροπρογραμματιζόμενης μηχανής 3 εντολών

Η Εικόνα 62 δείχνει ένα παράδειγμα κατακόρυφων και οριζόντιων μικροεντολών στη διαδρομή δεδομένων. Θέλουμε να δημιουργήσουμε μια μικροπρογραμματιζόμενη μηχανή με τρεις εντολές σε ένα απλό καταχωρητή a. Οι τρεις εντολές κάνουν ένα από τα παρακάτω: διπλασιάζουν την τιμή του a, μειώνουν την τιμή του a ή αρχικοποιούν την τιμή του a. Η διαδρομή δεδομένων που φαίνεται στο κάτω μέρος της Εικόνας 62 περιέχει δύο πολυπλέκτες και έναν προγραμματιζόμενο αθροιστή/αφαιρέτη. Για τον **κατακόρυφο μικροκώδικα, χρησιμοποιούμε τον τύπο: $2^\lambda \geq k$** (όπου $k =$ πλήθος μικροεντολών και $\lambda =$ πλήθος bits που απαιτούνται για την κωδικοποίηση των μικροεντολών). Για τον **οριζόντιο μικροκώδικα, χρησιμοποιούμε τα σήματα ελέγχου του κυκλώματος που μας δίνεται, προκειμένου να δημιουργήσουμε κάθε φορά τη μικροεντολή που μας δίνεται.** Ο ελεγκτής στην κορυφή παρουσιάζει δύο δυνατές κωδικοποιήσεις για τις τρεις εντολές: μια οριζόντια και μια κάθετη κωδικοποίηση.



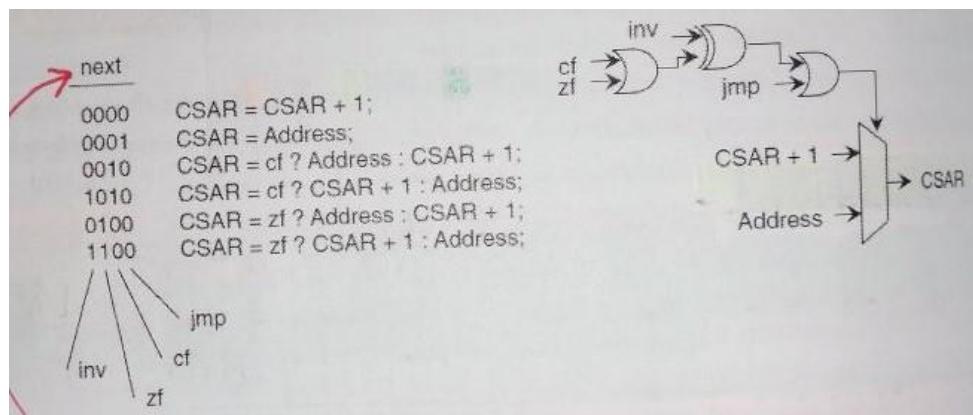
Στην περίπτωση εκτέλεσης της εντολής $a = IN$, θέλουμε από τον κάτω πολυπλέκτη να επιλεγεί η είσοδος στο «1», δηλαδή το IN, προκειμένου η συγκεκριμένη είσοδος να εισαχθεί στον καταχωρητή "a", όπου και θα αποθηκευτεί. Τα άλλα δύο σήματα ελέγχου δεν χρησιμοποιούνται, με αποτέλεσμα να είναι αδιάφοροι όροι (X). Όμως επειδή στον μικροπρογραμματισμό απαγορεύεται η χρήση αδιάφορων όρων, αντικαθιστούμε τα X με "0".

Σημείωση: Κωδικοποίηση μικροπρογράμματος και Κωδικοποίηση μικροεντολών

- Στην περίπτωση του κατακόρυφου μικροκώδικα, οι μικρο-εντολές θα κωδικοποιηθούν. Δεδομένου ότι υπάρχουν τρεις διαφορετικές εντολές, μπορούμε να υλοποιήσουμε αυτή τη μηχανή με μια λέξη μικρο-εντολής 2-bit. Για να δημιουργήσουμε τα bits ελέγχου για τη διαδρομή δεδομένων, θα πρέπει να αποκωδικοποιήσουμε κάθε μία από τις λέξεις μικρο-εντολής σε τοπικά σήματα ελέγχου στη διαδρομή δεδομένων.
- Στην περίπτωση του οριζόντιου μικροκώδικα, η μνήμη ελέγχου θα περιλαμβάνει κάθε ένα από τα bit ελέγχου στη διαδρομή δεδομένων ως ένα bit στη λέξη μικρο-εντολής. Επομένως σε αυτήν την περίπτωση, η κωδικοποίηση των εντολών αντανακλά ακριβώς την απαιτούμενη ρύθμιση των στοιχείων της διαδρομής δεδομένων για κάθε μικρο-εντολή.

Σημείωση: Πλεονεκτήματα – Μειονεκτήματα κατακόρυφων μικροπρογραμμάτων

Τα κατακόρυφα μικροπρογράμματα έχουν καλύτερη πυκνότητα κώδικα η οποία πλεονεκτεί ως προς το μέγεθος της μνήμης ελέγχου. Στην Εικόνα 62 η κατακόρυφα – κωδικοποιημένη έκδοση του μικροπρογράμματος θα έχει μόνο τα 2/3 του μεγέθους της οριζόντια – κωδικοποιημένης έκδοσης. Από την άλλη, τα κατακόρυφα μικροπρογράμματα χρησιμοποιούν ένα επιπλέον επίπεδο κωδικοποίησης, έτσι ώστε κάθε μικρο-εντολή πρέπει πρώτα να αποκωδικοποιηθεί, πριν να μπορέσει να οδηγήσει τα bits ελέγχου της διαδρομής δεδομένων. Έτσι, η μηχανή με το κατακόρυφα κωδικοποιημένο μικρό πρόγραμμα μπορεί να είναι πιο αργή.



Εικόνα 63 - Κωδικοποίηση CSAR

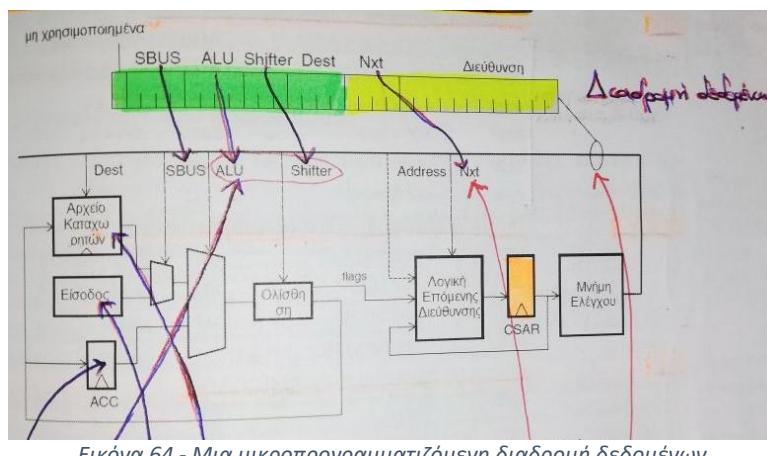
Σημείωση: Επιλογή οριζόντιας – κατακόρυφης κωδικοποίησης

Η επιλογή μεταξύ κατακόρυφης και οριζόντιας κωδικοποίησης πρέπει να γίνει προσεκτικά. Στην πράξη, οι σχεδιαστές χρησιμοποιούν έναν συνδυασμό κατακόρυφων και οριζόντιων εννοιών κωδικοποίησης, έτσι ώστε η παραγόμενη αρχιτεκτονική να είναι συμπαγής και αποτελεσματική. Εξετάστε, για παράδειγμα την τιμή του πεδίου next της λέξη μικρο-εντολής στην Εικόνα 61. Υπάρχουν έξι διαφορετικοί τύποι εντολών άλματος, πράγμα που υποδηλώνει ότι μια κατακόρυφη μικροεντολή δε θα χρειαζόταν περισσότερα από 3 bit για να κωδικοποιήσει αυτά τα έξι άλματα. Ωστόσο έχουν χρησιμοποιηθεί 4 bit, υποδεικνύοντας ότι έχει απομείνει κάποιος πλεονασμός (redundancy). Η κωδικοποίηση επιλέχθηκε για να απλοποιηθεί ο σχεδιασμό της λογικής επόμενης-διεύθυνσης, η οποία φαίνεται στην Εικόνα 63.

Η Μικροπρογραμματιζόμενη Διαδρομή Δεδομένων

Σημείωση: Τι κάνει και τι περιέχει μια διαδρομή δεδομένων

Η διαδρομή δεδομένων μιας μικροπρογραμματιζόμενης μηχανής αποτελείται από τρία στοιχεία: μονάδες υπολογισμού, μνήμη και διαύλους επικοινωνίας. Καθένα από αυτά μπορεί να συνεισφέρει λίγα bit ελέγχου στη λέξη μικρο-εντολής. Για παράδειγμα, οι μονάδες υπολογισμού, πολλαπλών λειτουργιών έχουν δυαδικά ψηφία επιλογής λειτουργίας, οι μονάδες αποθήκευσης έχουν bit διεύθυνσης και bit εντολών ανάγνωσης/εγγραφής και οι δίαυλοι επικοινωνίας έχουν bit ελέγχου πηγής/προορισμού. Η διαδρομή δεδομένων μπορεί επίσης να παράγει σημαίες κατάστασης για τον μικροπρογραμματιζόμενο ελεγκτή.



Εικόνα 64 - Μια μικροπρογραμματιζόμενη διαδρομή δεδομένων

6.1.6 Αρχιτεκτονική Διαδρομής Δεδομένων

Η Εικόνα 64 απεικονίζει έναν μικροπρογραμματιζόμενο ελεγκτή με μια συνδεδεμένη διαδρομή δεδομένων. Η διαδρομή δεδομένων περιλαμβάνει μια μονάδα ALU με ολισθητή, ένα αρχείο καταχωρητών με οκτώ στοιχεία, έναν καταχωρητή-συσσωρευτή και μια θύρα εισόδου. Η λέξη μικρο-εντολής εμφανίζεται στην κορυφή της Εικόνας 64 και περιέχει έξι πεδία. Δύο πεδία, τα **Nxt** και **Address**, χρησιμοποιούνται από τον μικροπρογραμματιζόμενο ελεγκτή. Τα άλλα χρησιμοποιούνται από τη διαδρομή δεδομένων. Ο τύπος της κωδικοποίησης είναι μικρή οριζόντια/κατακόρυφη. Η συνολική μηχανή χρησιμοποιεί μια οριζόντια κωδικοποίηση: κάθε μονάδα της μηχανής ελέγχεται ανεξάρτητα.

Πίνακας 6.1: Κωδικοποίηση μικρο-εντολών της μηχανής παραδείγματος

Πεδίο	Μή- κος	Κωδικοποίηση
-------	------------	--------------

SBUS	4	Επιλέγει τον τελεστέο που θα οδηγήσει το S-Bus
	0000	R0
	0001	R1
	0010	R2
	0011	R3
	0100	R4
		0101 R5
		0110 R6
		0111 R7
		1000 Input
		1001 Address/Constant

ALU	4	Επιλέγει τη λειτουργία που εκτελείται από την ALU
------------	----------	---

0000	ACC	0110	ACC – S-Bus
0001	S-Bus	0111	not S-Bus
0010	ACC + SBus	1000	ACC + 1
0011	ACC - SBus	1001	SBus – 1
0100	SBus - ACC	1010	0
0101	ACC & SBus	1011	1

Shifter 3 Επιλέγει τη λειτουργία του προγραμματιζόμενου ολισθητή

000	logical SHL(ALU)	100	arith SHL(ALU)
001	logical SHR(ALU)	101	arith SHR(ALU)
010	rotate left ALU	111	ALU
011	rotate right ALU		

Dest 4 Επιλέγει τον στόχο που θα αποθηκεύσει S-Bus

0000	R0	0101	R5
0001	R1	0110	R6
0010	R2	0111	R7
0011	R3	1000	ACC
0100	R4	1111	unconnected

Next 4 Επιλέγει την επόμενη τιμή για τον CSAR

0000	CSAR + 1	1010	cf ? CSAR + 1: Address
0001	Address	0100	zf ? Address: CSAR + 1
0010	cf ? Address : CSAR + 1	1100	zf ? CSAR + 1: Address

Οι υπομονάδες μέσα στη μηχανή, από την άλλη, χρησιμοποιούν μια κατακόρυφη κωδικοποίηση. Για παράδειγμα, το πεδίο ALU περιέχει 4 bit. Σε αυτήν την περίπτωση, η μονάδα ALU στη διαδρομή δεδομένων, θα εκτελέσει έως και 16 διαφορετικές εντολές.

Η μηχανή ολοκληρώνει μία εντολή ανά κύκλο ρολογιού. Η μονάδα ALU συνδυάζει έναν τελεστέο από τον καταχωρητή συσσωρευτή με έναν τελεστέο από το αρχείο καταχωρητών ή τη θύρα εισόδου. Το αποτέλεσμα της λειτουργίας επιστρέφεται στο αρχείο καταχωρητών ή στο συσσωρευτή. Η επικοινωνία που χρησιμοποιείται για τις λειτουργίες διαδρομής δεδομένων ελέγχεται από δύο πεδία στη λέξη μικρο-εντολής. Το πεδίο SBUS και το πεδίο Dest υποδεικνύουν την πηγή και τον προορισμό αντίστοιχα.

Σημείωση: Σημαίες (flags) που παράγει ο Shifter

Η μονάδα Shifter παράγει επίσης σημαίες, οι οποίες χρησιμοποιούνται από τον μικροπρογραμματιζόμενο ελεγκτή για την υλοποίηση αλμάτων υπό συνθήκη. Δημιουργούνται δύο σημαίες: μια μηδενική σημαία η οποία

είναι υψηλή (1) όταν η έξοδος του ολισθητή είναι μηδενική, και μια σημαία-κρατουμένου, η οποία περιέχει το bit ολισθημένο-έξω από την πιο σημαντική θέση.

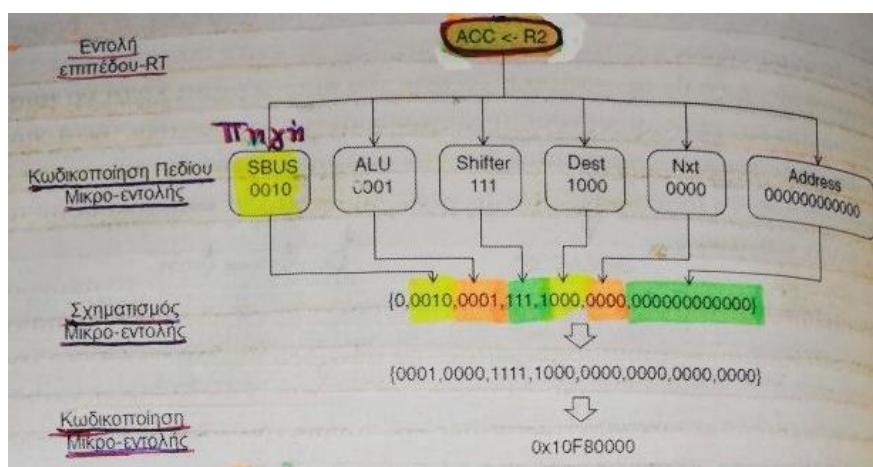
6.1.7 Αρχιτεκτονική Διαδρομής Δεδομένων

Σημείωση: Πως σχηματίζεται μια μικρο-εντολή

Μια μικρο-εντολή μπορεί να σχηματιστεί επιλέγοντας μια συνάρτηση μονάδας για κάθε μονάδα της μικρο-προγραμματιζόμενης μηχανής, συμπεριλαμβανομένης μιας επόμενης-διεύθυνσης για το πεδίο Address. Όταν ένα πεδίο παραμένει αχρησιμοποίητο κατά τη διάρκεια μιας συγκεκριμένης εντολής, μπορεί να επιλεγεί μια τιμή αδιάφορου όρου (don't care). Η τιμή αδιάφορου όρου πρέπει να επιλεγεί προσεκτικά, ώστε να αποφεύγονται ανεπιθύμητες αλλαγές κατάστασης στη διαδρομή δεδομένων.

Σημείωση: Παράδειγμα μικρο-εντολής

Για παράδειγμα, μία εντολή για την αντιγραφή του καταχωρητή R2 στον καταχωρητή συσσωρευτή ACC, θα σχηματιστεί όπως απεικονίζεται στην Εικόνα 65. Η εντολή θα πρέπει να διαβάζει τον καταχωρητή R2 από το αρχείο καταχωρητών, να διαβιβάζει τα περιεχόμενα του καταχωρητή μέσω του SBUS, μέσα από την μονάδα ALU και του ολισθητή και να γράφει το αποτέλεσμα στον καταχωρητή ACC. Αυτή η παρατήρηση επιτρέπει να προσδιοριστεί η τιμή κάθε πεδίου στην μικρο-εντολή.



Εικόνα 65 - Δημιουργία μικρο-εντολών από τις εντολές μεταφοράς καταχωρητή

Σημείωση: Δημιουργία μικροεντολών

- Το SBUS χρειάζεται να κρατά την τιμή του R2. Χρησιμοποιώντας τον Πίνακα 6.1 βρίσκουμε ότι το SBUS ισούται με 0010.
- Η μονάδα ALU πρέπει να περάσει την είσοδο SBUS στην έξοδο. Βάσει του Πίνακα 6.1 η ALU πρέπει να είναι ίση με 0001.
- Ο ολισθητής μεταβιβάζει την έξοδο ALU χωρίς τροποποίηση, επομένως ο Shifter πρέπει να ισούται με 111.
- Η έξοδος του ολισθητή χρησιμοποιείται για να ενημερώσει το συσσωρευτή, έτσι ώστε το πεδίο Dest είναι ίσο με 1000.

- Υποθέτοντας ότι δεν εκτελείται άλμα ή μεταφορά ελέγχου από αυτήν την εντολή, η επόμενη μικρο-εντολή θα είναι απλά μία μετά από την τρέχουσα θέση στον CSAR. Αυτό σημαίνει ότι ο Nxt θα πρέπει να ισούται με 0000 και ο Address θα είναι κάποιος αδιάφορος όρος, για παράδειγμα όλα μηδενικά.
- Τέλος, μπορούμε να βρούμε το συνολικό κώδικα μικρο-εντολής τοποθετώντας όλα τα πεδία εντολών μαζί. Η Εικόνα 65 απεικονίζει αυτή τη διαδικασία. Συμπεραίνουμε ότι μια μικρο-εντολή για την αντιγραφή του R2 στον ACC μπορεί να κωδικοποιηθεί ως 0x10F80000 στη μνήμη ελέγχου.

Η εγγραφή επομένως, ενός μικροπρογράμματος αποτελείται από τη διατύπωση της επιθυμητής συμπεριφοράς ως μια ακολουθία μεταφορών καταχωρητών (register transfers) και έπειτα την κωδικοποίηση αυτών των μεταφορών καταχωρητών ως πεδίων μικρο-εντολής.

Σημείωση: Υπολογισμός GCD με Ευκλείδη

Ένα μικρό πρόγραμμα που διαβάζει δύο αριθμούς από τη θύρα εισόδου και υπολογίζει τον μέγιστο κοινό διαιρέτη (GCD) τους χρησιμοποιώντας τον αλγόριθμο του Ευκλείδη. Το πρώτο βήμα είναι να αναπτύξουμε ένα μικροπρόγραμμα σε όρους μεταφοράς καταχωρητών. Μια πιθανή προσέγγιση παρουσιάζεται στη Λίστα 6.1. Οι γραμμές 2 και 3 αυτού του προγράμματος διαβάζουν δύο τιμές από τη θύρα εισόδου και αποθηκεύουν αυτές τις τιμές στους καταχωρητές R0 και ACC. Στο τέλος του προγράμματος, ο υπολογιζόμενος GCD θα είναι διαθέσιμος είτε στον ACC είτε στον R0 και το πρόγραμμα θα συνεχιστεί μέχρις ότου και οι δύο τιμές είναι ίσες. Ο έλεγχος τερματισμού υλοποιείται στη γραμμή 4, χρησιμοποιώντας την αφαίρεση των δύο καταχωρητών και ένα άλμα υπό συνθήκη με βάση τη μηδενική σημαία. Υποθέτοντας ότι οι δύο καταχωρητές περιέχουν διαφορετικές τιμές, το πρόγραμμα θα συνεχίσει να αφαιρεί το μεγαλύτερο καταχωρητή από το μικρότερο. Αυτό απαιτεί να βρεθεί ποιος από τους R0 και ACC είναι μεγαλύτερος και υλοποιείται με ένα άλμα υπό συνθήκη στη γραμμή 5. Ο έλεγχος μεγαλύτερου – από υλοποιείται χρησιμοποιώντας μια αφαίρεση, μια αριστερή-ολίσθηση και ένα έλεγχο στην παραγόμενη σημαία-κρατουμένου. Εάν η σημαία-κρατουμένου έχει τεθεί, τότε το πιο σημαντικό bit της αφαίρεσης θα είναι ένα, υποδεικνύοντας ένα αρνητικό αποτέλεσμα στη λογική του συμπληρώματος ως προς δύο. Αυτό το υπό συνθήκη άλμα εάν υπάρχει κρατούμενο θα προκύψει αν ο R0 είναι μικρότερος από ACC. Ο συνδυασμός των γραμμών 5-7 δείχνει πώς μια εντολή if-then-else μπορεί να δημιουργηθεί χρησιμοποιώντας πολλαπλές εντολές άλματος με ή χωρίς συνθήκη.

Λίστα 6.1 Μικρο-πρόγραμμα για τον υπολογισμό ενός GCD

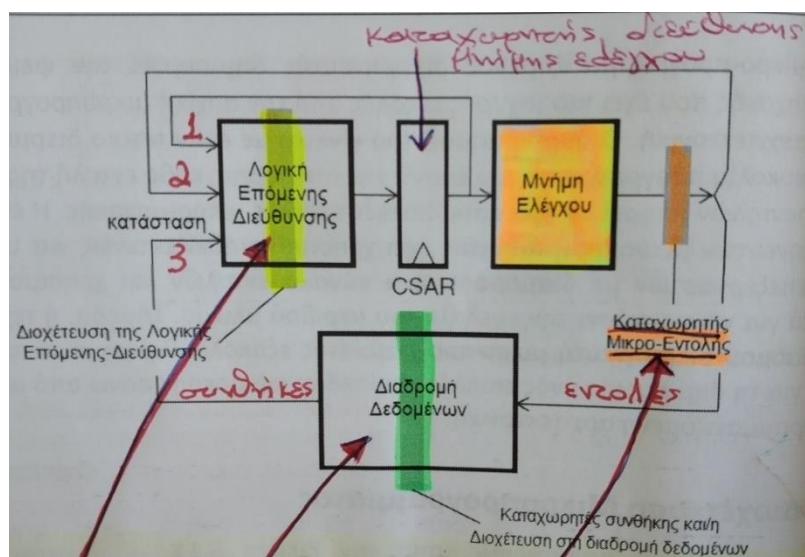
1 ; Πεδίο Εντολής	Πεδίο Άλματος
2 IN -> R0	
3 IN -> ACC	
4 Lcheck: (R0 – ACC)	JUMP_IF_Z Ldone
5 (R0 – ACC) << 1	JUMP_IF_C LSmall
6 R0 – ACC -> R0	JUMP Lcheck
7 Lsmall: ACC – R0 -> ACC	JUMP Lcheck
8 Ldone	

Διοχέτευση Μικροπρογράμματος

Σημείωση: Εισαγωγή καταχωρητών διοχέτευσης

Όπως μπορεί να παρατηρηθεί από την Εικόνα 66 ο ελεγκτής μικροπρογράμματων μπορεί να είναι μέρος μιας μακράς αλυσίδας συνδυαστικής λογικής. Καταχωρητές διοχέτευσης μπορούν να χρησιμοποιηθούν για να σπάσουν αυτές τις μεγάλες αλυσίδες. Ωστόσο, η εισαγωγή καταχωρητών διοχέτευσης έχει σημαντικό αντίκτυπο στο σχεδιασμό των μικρο-προγραμμάτων.

Πρώτα, παρατηρήστε στην Εικόνα 66 ότι ο καταχωρητής CSAR είναι μέρος πιθανώς τριών βρόχων λογικής. Ο πρώτος βρόχος περνάει από τη λογική επόμενης-διεύθυνσης. Ο δεύτερος βρόχος διέρχεται από την μνήμη ελέγχου και τη λογική επόμενης – διεύθυνσης. Ο τρίτος βρόχος διέρχεται από τη μνήμη ελέγχου, τη διαδρομή δεδομένων και τη λογική επόμενης-διεύθυνσης. Αυτά τα συνδυαστικά μονοπάτια μπορεί να περιορίσουν τη μεγιστηριακή συχνότητα ρολογιού της μικροπρογραμματιζόμενης μηχανής.



Εικόνα 66 –Τυπική χωροθέτηση των καταχωρητών διοχέτευσης σε ένα διερμηνευτή μικρο-προγραμμάτων

Σημείωση: Τρεις πιθανές τοποθετήσεις καταχωρητή διοχέτευσης

1. Μια συνήθης θέση για την εισαγωγή ενός καταχωρητή διοχέτευσης βρίσκεται στην έξοδο της μνήμης ελέγχου. Ένας καταχωρητής σε αυτή τη θέση ονομάζεται καταχωρητής μικρο-εντολής και επιτρέπει την επικάλυψη του υπολογισμού της διαδρομής δεδομένων, του υπολογισμού επόμενης διεύθυνσης και της ανάκλησης μικροεντολής.
2. Μια άλλη θέση για τους καταχωρητές διοχέτευσης είναι η διαδρομή δεδομένων. Εκτός από τον καταχωρητή διοχέτευσης μέσα στη διαδρομή δεδομένων, μπορούν να τοποθετηθούν επιπλέον καταχωρητές κωδικού συνθήκης στις εξόδους της διαδρομής δεδομένων.
3. Τέλος, η λογική επόμενης-διεύθυνσης μπορεί επίσης να διοχετευθεί, σε περίπτωση που απαιτείται λειτουργία υψηλής ταχύτητας και η διεύθυνση στόχου του CSAR δεν μπορεί να υπολογιστεί μέσα σε ένα μόνο κύκλο ρολογιού.

Πίνακας 6.2 Επίδραση του καταχωρητή μικρο-εντολής στις εντολές άλματος

Κύκλος	CSAR	Καταχωρητής μικρο-εντολής
N	4	
N+1	5	CSTORE(4) = JUMP 10
N+2	10	CSTORE(5) πρέπει να ακυρωθεί
N+3	11	CSTORE(10) εκτέλεση

6.1.8 Καταχωρητής Μικρο-εντολής

Σημείωση: Επίδραση προσθήκης καταχωρητή μικροεντολής

Ας εξετάσουμε πρώτα το αποτέλεσμα της προσθήκης του καταχωρητή μικροεντολής. Εξαιτίας αυτού του καταχωρητή, η ανάκληση μικρο-εντολών (δηλαδή η αναφορά στον CSTORE και η ανάκτηση της επόμενης μικρο-εντολής) μετατοπίζεται κατά έναν κύκλο από το υπολογισμό αυτής της μικρο-εντολής. Για παράδειγμα, όταν ο CSAR θα μεταφέρει την εντολή i μιας ακολουθίας εντολών, η διαδρομή δεδομένων και η λογική επόμενης – διεύθυνσης θα εκτελεί την εντολή i-1.

Ο Πίνακας 6.2 απεικονίζει την επίδραση αυτής της μετατόπισης στη ροή εντολών, όταν αυτή η ροή περιέχει μια εντολή άλματος. Ο μικρο-προγραμματιστής εισήγαγε μια εντολή JUMP 10 στην τοποθεσία CSTORE 4 και αυτή η εντολή θα ανακληθεί στον κύκλο ρολογιού N. Στον κύκλο ρολογιού N + 1, η μικροεντολή θα εμφανιστεί στην έξοδο του καταχωρητή μικρο-εντολής. Η εκτέλεση αυτής της εντολή θα ολοκληρωθεί στον κύκλο N + 2. Ως αποτέλεσμα, μια εντολή JUMP δε μπορεί να τροποποιήσει την τιμή του CSAR μέσα σε ένα μόνο κύκλο ρολογιού. Αν το CSTORE (4) περιέχει μια JUMB, τότε η εντολή που βρίσκεται στο CSTORE (5) θα ανακληθεί επίσης. Ο μικρο-προγραμματιστής πρέπει να το γνωρίζει αυτό. Οι πιθανές στρατηγικές είναι: (α) να λαμβάνετε υπόψη ότι μια JUMP θα εκτελείται με έναν κύκλο καθυστέρηση (αποκαλούμενος "καθυστερούμενος κλάδος"), ή (β) να περιλαμβάνετε υποστήριξη στη μικρο-προγραμματιζόμενη μηχανή για την ακύρωση της εκτέλεσης μιας εντολής σε περίπτωση άλματος.

Πίνακας 6.3 Επίδραση του καταχωρητή μικρο-εντολής και του καταχωρητη κωδικού συνθήκης στις εντολές άλματος υπό-συνθήκη

Κύκλος	CSAR	Καταχωρητής μικρο-εντολής
N	4	
N+1	5	CSTORE(4) = JUMP 10
N+2	10	CSTORE(5) πρέπει να ακυρωθεί
N+3	11	CSTORE(10) εκτέλεση

6.1.9 Καταχωρητής Κωδικού-συνθήκης Διαδρομής Δεδομένων

Ως δεύτερη περίπτωση, υποθέστε ότι έχουμε έναν καταχωρητή κώδικα υπό-συνθήκη στη διαδρομή δεδομένων, πέραν του καταχωρητή μικρο-εντολής. Το καθαρό αποτέλεσμα ενός καταχωρητή κωδικού συνθήκης είναι ότι μια τιμή κατάστασης θα είναι διαθέσιμη μόνο ένα κύκλο ρολογιού μετά από την αντίστοιχη λειτουργία διαδρομής δεδομένων. Ως αποτέλεσμα, μια εντολή άλματος-υπό συνθήκη μπορεί να λειτουργήσει μόνο στις

συνθήκες διαδρομής δεδομένων από τον προηγούμενο κύκλο ρολογιού. Ο πίνακας 6.3 προβάλλει αυτό το αποτέλεσμα. Η εντολή διακλάδωσης στο CSTORE (4) είναι ένα άλμα υπό συνθήκη. Όταν η συνθήκη είναι αληθής, το άλμα θα εκτελεστεί με καθυστέρηση ενός κύκλου ρολογιού. Ωστόσο, ο J2 υπολογίζεται στον κύκλο $N + 2$ με βάση έναν κώδικα συνθήκης που παράγεται στον κύκλο $N + 1$. Έτσι, ο μικρο-προγραμματιστής πρέπει να γνωρίζει ότι οι συνθήκες πρέπει να είναι διαθέσιμες έναν κύκλο ρολογιού πριν χρησιμοποιηθούν σε άλματα υπό συνθήκη.

Μικροπρογραμματισμός με Μικροελεγκτές

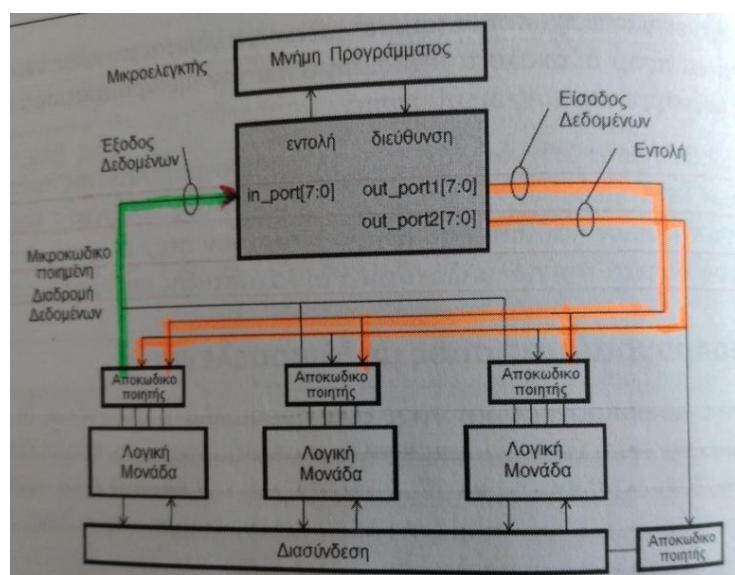
6.1.10 Αρχιτεκτονική Συστήματος

Ένας μικροελεγκτής έχει ελάχιστες υπολογιστικές δυνατότητες, όπως μια ALU με βασικές λογικές και αριθμητικές λειτουργίες. Ωστόσο, χρησιμοποιούνται ευρέως για όλα τα είδη εφαρμογών ελέγχου. Σε αυτήν την παράγραφο, θα συζητήσουμε τη χρήση τους ως ελεγκτές μικρο-προγραμμάτων.

Η Εικόνα 67 δείχνει ένα παράδειγμα μιας μικροπρογραμματιζόμενης αρχιτεκτονικής που χρησιμοποιεί έναν μικροελεγκτή. Υποθέτουμε μια συσκευή η οποία φέρει ψηφιακές θύρες 8bit I/O. Ο μικροελεγκτής έχει αφοσιωμένες εντολές για να διαβάσει από, και, να γράψει σε, τέτοιες θύρες. Στο σχεδιασμό της Εικόνας 67, οι θύρες I/O χρησιμοποιούνται για τον έλεγχο μιας μικροκωδικοποιημένης διαδρομής-δεδομένων. Υπάρχουν τρεις εμπλεκόμενες θύρες.

Σημείωση: Είδη θυρών μικροελεγκτή

- Μια ψηφιακή θύρα εισόδου χρησιμοποιείται για τη μεταφορά δεδομένων ή κατάστασης από τη μικροκωδικοποιημένη διαδρομή δεδομένων στον μικροελεγκτή.
- Μια πρώτη ψηφιακή θύρα εξόδου χρησιμοποιείται για τη μεταφορά δεδομένων από τον μικροελεγκτή στη μικροκωδικοποιημένη διαδρομή δεδομένων.
- Μια δεύτερη ψηφιακή θύρα εξόδου χρησιμοποιείται για τη μεταφορά ελέγχου ή μικροεντολών από τον μικροελεγκτή στη μικροκωδικοποιημένη διαδρομή δεδομένων.



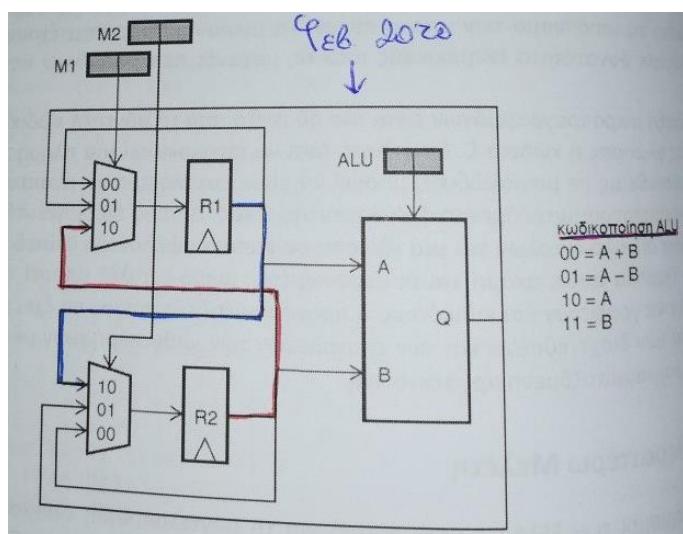
Εικόνα 67 - Χρήση μικροελεγκτή ως ελεγκτή μικροπρογράμματος

Η μηχανή λειτουργεί ως εξής. Για κάθε μικρο-εντολή, ο μικροελεγκτής θα συνδυάσει μια μικρο-εντολή με ένα προαιρετικό όρισμα και θα το στείλει στη μικρο-κωδικοποιημένη διαδρομή δεδομένων. Η μικροκωδικοποιημένη διαδρομή δεδομένων θα επιστρέψει στη συνέχεια οποιοδήποτε αποτέλεσμα στον μικροελεγκτή.

Πρέπει να ληφθεί μέριμνα ώστε να διατηρηθούν συγχρονισμένοι η μικροκωδικοποιημένη διαδρομή δεδομένων και ο μικροελεγκτής. Εάν η εκτέλεση μιας μικροκωδικοποιημένης εντολής διαρκεί πολλαπλούς κύκλους, ο μικροελεγκτής θα χρειαστεί να καθυστερήσει την ανάγνωση των δεδομένων εξόδου και/ή την κατάσταση, για έναν κατάλληλο αριθμό κύκλων.

Προβλήματα

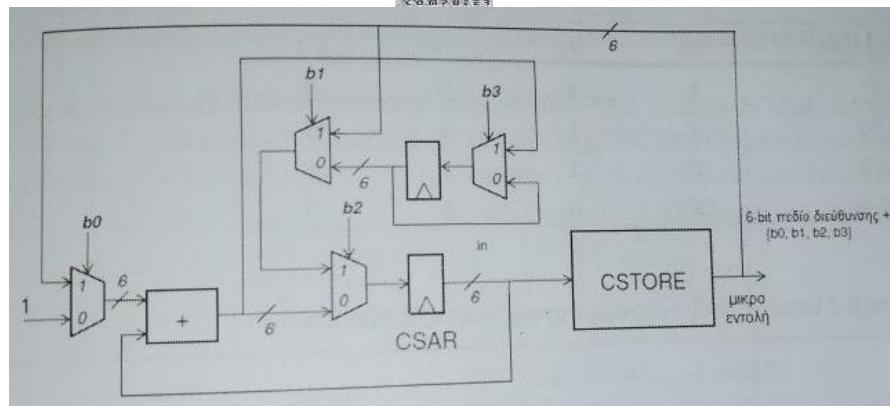
Πρόβλημα 6.1 Η Εικόνα 68 δείχνει μια μικροπρογραμματιζόμενη διαδρομή δεδομένων. Υπάρχουν έξι bit ελέγχου για τη διαδρομή δεδομένων: 2 bits για κάθε έναν από τους πολυπλέκτες M1 και M2 και 2 bits για την ALU. Η κωδικοποίηση των bit ελέγχου υποδεικνύεται στην εικόνα.



Εικόνα 68 - Διαδρομή δεδομένων για το Πρόβλημα 6.1

Πίνακας 6.4 Μικρο-εντολές για το Πρόβλημα 6.1

SWAP	Ανταλλάξτε το περιεχόμενο των R1 και R2
ADD Rx	Προσθέστε τα περιεχόμενα των R1 και R2 και αποθηκεύστε τα περιεχόμενα στον Rx, που είναι ίσος με R1 ή R2. Υπάρχουν δύο παραλλαγές αυτής της εντολής εξαρτώμενες από τον Rx.
COPY Rx	Αντιγράψτε τα περιεχόμενα του Ry στον Rx. Το (Rx, Ry) είναι είτε (R1, R2) είτε (R2, R1). Υπάρχουν δύο παραλλαγές αυτής της εντολής ανάλογα με τον Rx
NOP	Μην κάνεις τίποτα



Εικόνα 69 - Διαδρομή Δεδομένων για το Πρόβλημα 6.3

(α) Αναπτύξετε μια οριζόντια κωδικοποίηση μικρο-εντολής για τη λίστα των μικροεντολών που δίνονται στον Πίνακα 4.

(β) Αναπτύξετε μιας κατακόρυφη κωδικοποίηση μικρο-εντολών για την ίδια λίστα εντολών. Χρησιμοποιήστε μια ικανοποιητική κωδικοποίηση που καταλήγει σε ένα συμπαγή και αποδοτικό αποκωδικοποιητή για τη διαδρομή δεδομένων.

Πρόβλημα 6.2 Η Εικόνα 69 δείχνει την υλοποίηση ενός αποκωδικοποιητή επόμενης-διεύθυνσης. Συνολικά 10 bit από την μικρο-εντολή χρησιμοποιούνται για τον έλεγχο της λογικής επόμενης - διεύθυνσης: ένα πεδίο διεύθυνσης 6-bit και τέσσερα bit ελέγχου, b0, b1, b2 και b3.

Για καθέναν από τους συνδυασμούς bit ελέγχου που φαίνονται στον Πίνακα 5, βρείτε μια καλή περιγραφή της εντολής που αντιστοιχεί στις τιμές των δυαδικών ψηφίων ελέγχου που φαίνονται. Μην γράφετε γενικές περιγραφές αλλά δώστε μια υψηλού επιπέδου περιγραφή της εντολής που υλοποιείται. Χρησιμοποιήστε όρους που μπορεί να καταλάβει ένας προγραμματιστής λογισμικού.

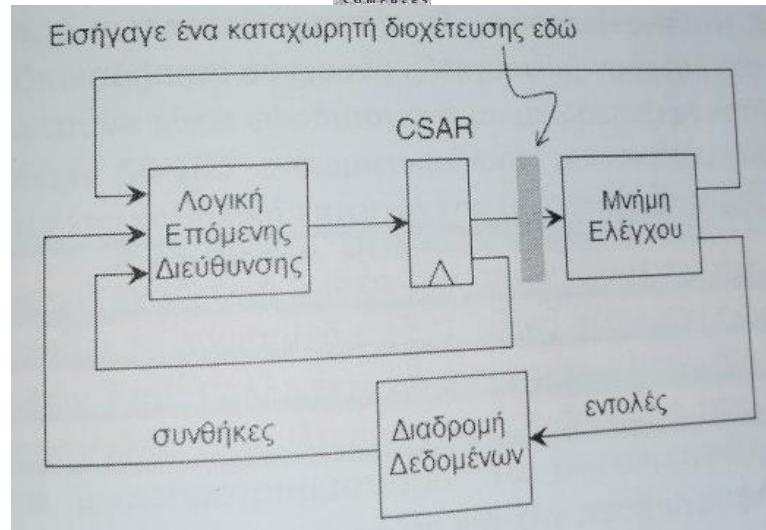
Πίνακας 6.5 Εντολές επόμενης-διεύθυνσης για το Πρόβλημα 6.2

Συνδυασμός	b0	b1	b2	b3
Εντολή 1	1	X	0	0
Εντολή 2	X	1	1	0
Εντολή 3	0	1	1	1
Εντολή 4	X	0	1	0

Πρόβλημα 6.3 Η βελτίωση είναι να εισαγάγετε έναν καταχωρητή διοχέτευσης ακριβώς μπροστά από τη μνήμη ελέγχου.

(α) Μειώνει αυτός ο πρόσθετος καταχωρητής το κρίσιμο μονοπάτι της συνολικής αρχιτεκτονικής.

(β) Ο συνάδελφός σας το αποκαλεί αρχιτεκτονική διπλού-νήματος και ισχυρίζεται ότι αυτή η βελτίωση επιτρέπει στη μηχανή μικρο-ελέγχου να εκτελεί δύο εντελώς ανεξάρτητα προγράμματα με εναλλασσόμενο τρόπο. Συμφωνείτε με αυτό ή όχι;



Εικόνα 70 - Διαδρομή Δεδομένων για το Πρόβλημα 6.5

7 Κεφάλαιο - Ενσωματωμένοι Πυρήνες Γενικού Σκοπού

Επεξεργαστές

Υπάρχουν διάφοροι λόγοι για την καθολική επιτυχία του μικροεπεξεργαστή

Σημείωση: Σημασία μικροεπεξεργαστών – Χαρακτηριστικά

- Οι μικροεπεξεργαστές ή η ιδέα του αποθηκευμένου προγράμματος γενικά, διαχωρίζει το λογισμικό από το υλικό μέσω του ορισμού ενός συνόλου-εντολών. Καμία άλλη τεχνική ανάπτυξης υλικού δεν ήταν ποτέ σε θέση να αποσυνδέσει το υλικό και το λογισμικό με παρόμοιο τρόπο.
- Οι μικροεπεξεργαστές συνοδεύονται από εργαλεία (μεταγλωττιστές συμβολομεταφραστές) που βοηθούν ένα σχεδιαστή να δημιουργεί εφαρμογές.
- Καμία άλλη συσκευή δεν μπόρεσε να αντιμετωπίσει τόσο αποτελεσματικά την επαναχρησιμοποίηση όσο και οι μικροεπεξεργαστές. Η επαναχρησιμοποίηση (reuse), γενικά, είναι η δυνατότητα εξοικονόμησης σχεδιαστικής προσπάθειας σε πολλαπλές εφαρμογές. Ένας ενσωματωμένος πυρήνας γενικού σκοπού είναι ένα εξαιρετικό παράδειγμα επαναχρησιμοποίησης από μόνο του.
- Τέταρτον, κανένα άλλο προγραμματιζόμενο εξάρτημα δεν έχει την ίδια δυνατότητα κλιμάκωσης με ένα μικροεπεξεργαστή. Οι μικροεπεξεργαστές έχουν επίσης αποκτήσει σημαντική έλξη ως κεντρικές μονάδες σε σύνθετα ολοκληρωμένα κυκλώματα. Σε αυτήν την προσέγγιση, η οποία ονομάζεται Σύστημα στο Ολοκληρωμένο (System – on – Chip, SoC), ένας μικροεπεξεργαστής ελέγχει τη συνεργασία ενός ή περισσοτέρων πολύπλοκων περιφερειακών.

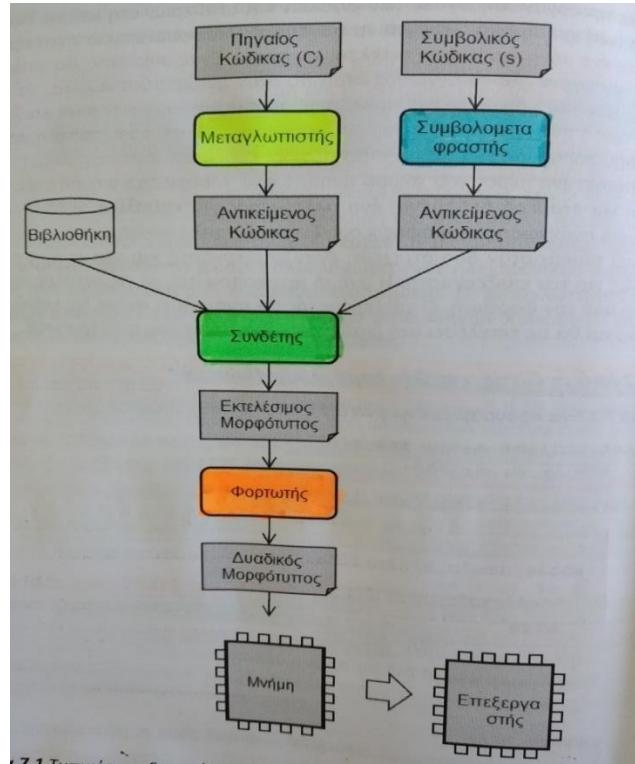
7.1.1 Από τη C στις εντολές Συμβολικής Γλώσσας

Λίστα 7.1 Ένα πρόγραμμα C για να βρείτε το μέγιστο κοινό διαιρέτη

```
1   int gcd (int a [ 5 ], int b [ 5 ]) {  
2       int i, m, n, max;  
3       max = 0 ;  
4       for (i = 0 ; i < 5 ; i++) {  
5           m = a [ i ] ;  
6           n = b [ i ] ;  
7           while (m != n ) {  
8               if (m > n)  
9                   m = m - n ;  
10              else  
11                  n = n - m ;  
12          }  
13          if (max > m)  
14              max = m ;  
15      }  
16      return max ;  
17  }  
18  
19  int a [ ] = {26, 3, 33, 56, 11} ;  
20  int b [ ] = {87, 12, 23, 47, 17} ;
```

```

21
22 int main () {
23     return gcd (a, b) ;
24 }
```



Εικόνα 71 - Τυπικός σχεδιασμός ροής του πηγαίου κώδικα λογισμικού στις εντολές επεξεργαστή

Λίστα 7.2 Εκτύπωση Συμβολικού Κώδικα της Λίστας 7.1

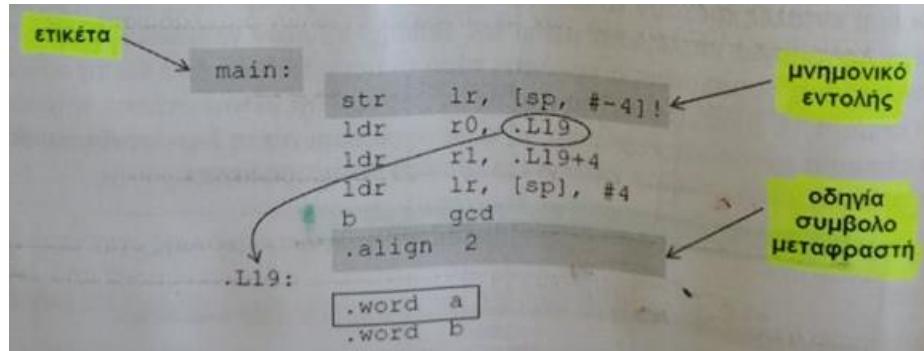
```

1      gcd:
2          str lr, [sp, #-4]!
3          mov lr, #0
4          mov ip, lr
5      .L13:
6          ldr r3, [r0, ip, asl #2]
7          ldr r2, [r1, ip, asl #2]
8          cmp r3, r2
9          beq .L17
10     .L11:
11         cmp r3, r2
12         rsbgt r3, r2, r3
13         rsble r2, r3, r2
14         cmp r3, r2
15         bne .L11
16     .L17:
17         add ip, ip, #1
18         cmp lr, r3
19         mov lr, r3
20         cmp ip, #4
21         movgt r0, lr
22         ldrgt pc, [sp], #4
23         b .L13
24     a:
25         .word 26, 3, 33, 56, 11
```

```

26   b:
27       .word 87, 12, 23, 45, 17
28   main:
29       str lr, [sp, #-4]!
30       ldr r0, .L19
31       ldr r1, .L19+4
32       ldr lr, [sp], #4
33       b gcd
34       .align 2
35   .L19:
36       .word a
37       .word b

```



Εικόνα 72 - Στοιχεία συμβολικού προγράμματος που παράγεται από τον gcc

Η Λίστα 7.2 είναι το συμβολικό πρόγραμμα που παράγεται από το πρόγραμμα C στη Λίστα 7.1. Η Εικόνα 72 απεικονίζει ορισμένα αξιοσημείωτα χαρακτηριστικά ενός συμβολικού προγράμματος. Το πρόγραμμα περιλαμβάνει τρία στοιχεία: ετικέτες, εντολές και οδηγίες συμβολομεταφραστή. Οι ετικέτες είναι συμβολικές διευθύνσεις. Χρησιμοποιούνται ως θέσεις στόχου για εντολές διακλάδωσης και ως συμβολικές θέσεις για μεταβλητές. Στην Εικόνα 72, για παράδειγμα, οι μεταβλητές `a` και `b` καλούνται με τις ετικέτες `.L19` και `.L19 + 4` αντίστοιχα. Οι οδηγίες (directives) συμβολομεταφραστή ξεκινούν με μια τελεία, δεν αποτελούν τμήμα του συμβολικού προγράμματος, αλλά χρησιμοποιούνται από τον συμβολομεταφραστή.

Η Διοχέτευση RISC

Αυτή η ενότητα περιγράφει την εσωτερική αρχιτεκτονική ενός πολύ συνηθισμένου τύπου μικροεπεξεργαστή, του Υπολογιστή Περιορισμένου Συνόλου Εντολών (Reduced Instruction Set Computer – RISC). Θα δούμε τις βασικές ιδέες στον σχεδιασμό αρχιτεκτονικής RISC. Το υλικό σε αυτήν την παράγραφο καλύπτεται συνήθως, σε πολύ μεγαλύτερο βάθος, σε ένα μάθημα αρχιτεκτονικής υπολογιστών.

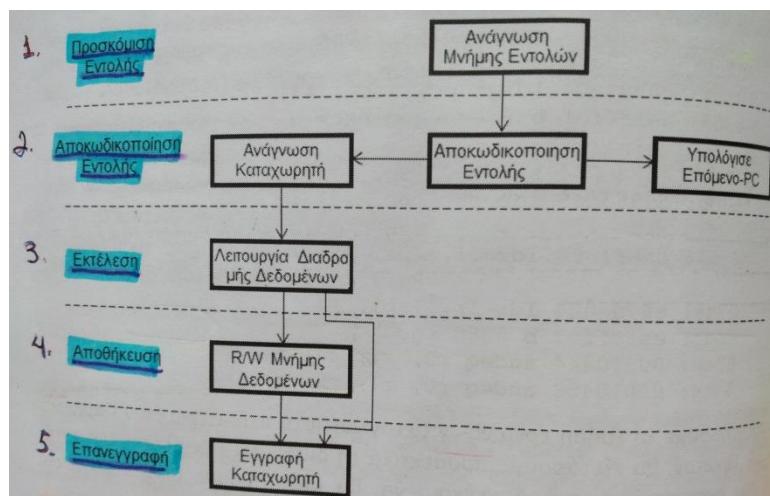
Σε έναν επεξεργαστή RISC, η εκτέλεση μιας απλής εντολής χωρίζεται σε διαφορετικά στάδια, τα οποία είναι συνδεδεμένα διαδοχικά ως μια διοχέτευση. Κάθε εντολή λειτουργεί σε ένα σύνολο καταχωρητών που βρίσκονται μέσα στον επεξεργαστή. Οι καταχωρητές επεξεργαστή χρησιμοποιούνται ως τελεστέοι ή ως στόχοι για τις εντολές του επεξεργαστή καθώς και για έλεγχο. Για παράδειγμα, ο επεξεργαστής ARM περιέχει 17 καταχωρητές: καταχωρητές δεδομένων `r0` έως `r14`, έναν καταχωρητή μετρητή προγράμματος `pc` και έναν καταχωρητή κατάστασης επεξεργαστή `cpsr`. Ο επεξεργαστής Microblaze διαθέτει 32 καταχωρητές γενικού σκοπού (`r0` έως `r31`)

και μέχρι 18 καταχωρητές ειδικού σκοπού (όπως ο μετρητής προγράμματος, ο καταχωρητής κατάστασης και άλλοι)

Σημείωση: μεταξύ ARM και Microblaze επιλέγουμε Microblaze, γιατί έχει πιο πολλά χαρακτηριστικά. Πιο σύνθετος είναι ο επεξεργαστής Microblaze, διότι έχει πιο πολλές εντολές.

Σημείωση: Μέρη Εντολής

Κάθε στάδιο μιας διοχέτευσης RISC χρειάζεται ένα κύκλο ρολογιού για να ολοκληρωθεί. Μια τυπική διοχέτευση RISC έχει τρία ή πέντε στάδια, και η Εικόνα 73 απεικονίζει μια διοχέτευση πέντε σταδίων. Τα πέντε στάδια της διοχέτευσης αποκαλούνται Προσκόμιση Εντολής (Instruction Fetch), Αποκωδικοποίηση Εντολής (Decode Instruction), Εκτέλεση (Execute), Αποθήκευση (Bubble) και Επανεγγραφή (Write-back). Καθώς εκτελείται μια εντολή, κάθε ένα από τα στάδια εκτελεί τις παρακάτω δραστηριότητες.



Εικόνα 73 - Μια διοχέτευση RISC πέντε σταδίων

- Προσκόμιση εντολής:** Ο επεξεργαστής φέρνει την εντολή που προσδιορίζεται από τον μετρητή προγράμματος, από τη μνήμη εντολών.
- Αποκωδικοποίηση Εντολής:** Ο επεξεργαστής εξετάζει τον κωδικό λειτουργίας της εντολής. Στην περίπτωση μιας εντολής διακλάδωσης, θα ενημερωθεί ο μετρητής προγράμματος.
- Εκτέλεση:** Η εντολή χρειάζεται να προσπελάσει τη μνήμη δεδομένων.
- Αποθήκευση:** Σε αυτό το στάδιο, ο επεξεργαστής μπορεί να προσπελάσει τη μνήμη δεδομένων, για ανάγνωση ή για εγγραφή.
- Επανεγγραφή:** Στο τελικό στάδιο της διοχέτευσης, ενημερώνονται οι καταχωρητές του επεξεργαστή.

Σημείωση: Χαρακτηριστικά Λειτουργίας Διοχετευμένου επεξεργαστή

Σημείωση: Πλεονέκτημα

Υπό ιδανικές συνθήκες, η πέντε – σταδίων διοχέτευση RISC είναι σε θέση να δεχτεί μια νέα εντολή σε κάθε κύκλο ρολογιού. Έτσι, ο ρυθμός διεκπεραίωσης εντολής (instruction throughput) σε έναν επεξεργαστή RISC μπορεί να είναι τόσο υψηλός όσο μία εντολή σε κάθε κύκλο ρολογιού. Λόγω της διοχέτευσης, κάθε εντολή μπορεί να χρειαστεί έως και πέντε κύκλους ρολογιού για να υπολογιστεί. Επομένως ο λανθάνων χρόνος εντολής (instruction latency) μπορεί να φτάσει έως και πέντε κύκλους ρολογιού. Μια διοχέτευση RISC βελτιώνει το ρυθμό διεκπεραίωσης εντολή σε βάρος του λανθάνοντα χρόνου εντολής. Ωστόσο, ο αυξημένος λανθάνων χρόνος εντολής ενός επεξεργαστή RISC συνήθως δεν είναι πρόβλημα, επειδή η συχνότητα ρολογιού ενός διοχετευμένου επεξεργαστή είναι υψηλότερη από αυτή ενός μη-διοχετευμένου επεξεργαστή.

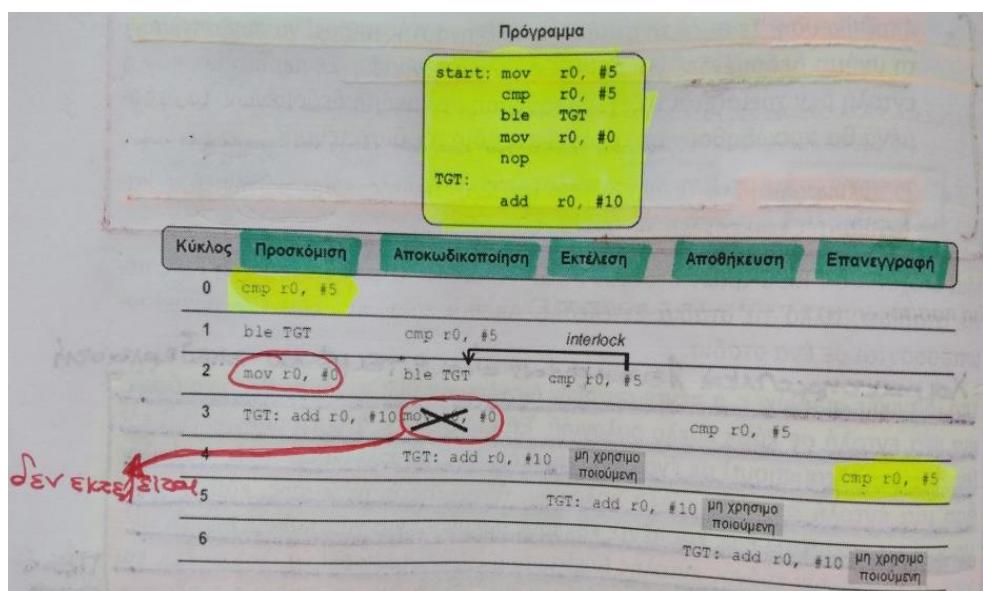
Σημείωση: Hazards

Σε ορισμένες περιπτώσεις δεν είναι δυνατό να ολοκληρωθεί μια εντολή μέσα σε πέντε κύκλους ρολογιού. Μια αναμονή (stall) διοχέτευσης εμφανίζεται όταν η πρόοδος των εντολών μέσω της διοχέτευσης διακόπτεται προσωρινά. Η αιτία ενός τέτοιου προβλήματος είναι ένας κίνδυνος διοχέτευσης (pipeline hazard). Σε προηγμένους επεξεργαστές RISC, το υλικό για εξαρτήσεις και παύσεις διοχέτευσης (pipeline interlock) μπορεί να ανιχνεύσει και να επιλύσει αυτόματα τους κινδύνους διοχέτευσης. Ακόμα και όταν υπάρχει υλικό interlock, οι κίνδυνοι διοχέτευσης ενδέχεται να εξακολουθούν να εμφανίζονται.

Σημείωση: Κατηγορίες Hazards

Οι τρεις κατηγορίες είναι οι ακόλουθες:

- Οι κίνδυνοι ελέγχου (control hazards) είναι κίνδυνοι διοχέτευσης που προκαλούνται από διακλαδώσεις.
- Οι κίνδυνοι δεδομένων (data hazards) είναι κίνδυνοι διοχέτευσης που προκαλούνται από ανεκπλήρωτες εξαρτήσεις δεδομένων.
- Οι δομικοί κίνδυνοι (structural hazards) προκαλούνται από συγκρούσεις πόρων και αστοχίες (misses) κρυφής μνήμης.



Εικόνα 74 - Παράδειγμα ενός κινδύνου ελέγχου



7.1.2 Κίνδυνοι Ελέγχου

Οι εντολές διακλάδωσης αποτελούν την πιο συνήθη μορφή αναμονών διοχέτευσης. Όπως υποδεικνύεται στην Εικόνα 73, μια διακλάδωση εκτελείται μόνο (δηλαδή τροποποιεί τον καταχωρητή του μετρητή προγράμματος) στο στάδιο δύο της διοχέτευσης. Την ίδια στιγμή, μια άλλη εντολή έχει ήδη εισέλθει στη διοχέτευση. Καθώς αυτή η εντολή βρίσκεται μετά από την εντολή διακλάδωσης, αυτή η εντολή θα πρέπει να πεταχτεί για να διατηρήσουμε τη σημασιολογία ακολουθιακής εκτέλεσης.

Η Εικόνα 74 απεικονίζει έναν κίνδυνο ελέγχου. Η διοχέτευση δίνεται σχεδιασμένη από το πλάι, και διατρέχεται από αριστερά προς τα δεξιά. Ο χρόνος τρέχει κατά μήκος των γραμμών. Υπάρχει κίνδυνος ελέγχου εξαιτίας της εντολής διακλάδωσης ble TGT. Στον κύκλο 2, η νέα τιμή του μετρητή προγράμματος υπολογίζει τη διεύθυνση προορισμού της διακλάδωσης, TGT. Σημειώστε ότι παρόλο που η ble είναι μια διακλάδωση υπό συνθήκη που χρησιμοποιεί το αποτέλεσμα της εντολής ακριβώς πριν από αυτή (cmp r0, #5), η συνθήκη διακλάδωσης είναι διαθέσιμη στον κύκλο 2, λόγω του υλικού interlock στη διοχέτευση. Ξεκινώντας στον κύκλο 3, οι εντολές από τη διεύθυνση προορισμού TGT μπαίνουν στη διοχέτευση. Την ίδια στιγμή, η εντολή αμέσως μετά τη διακλάδωση ακυρώνεται στο στάδιο αποκωδικοποίησης. Αυτό οδηγεί σε μια αχρησιμοποίητη θέση εντολής (instruction slot) αμέσως μετά την εντολή διακλάδωσης.

Ορισμένοι επεξεργαστές RISC περιλαμβάνουν μια εντολή διακλάδωσης με καθυστέρηση (delayed-branch). Ο σκοπός αυτής της εντολής είναι να επιτρέψει στην εντολή αμέσως μετά την εντολή διακλάδωσης να ολοκληρωθεί ακόμη και όταν έχει ακολουθηθεί η διακλάδωση. Αυτό θα αποτρέψει "αχρησιμοποίητες" θέσεις διοχέτευσης όπως φαίνεται στην Εικόνα 74.

Για παράδειγμα, η ακόλουθη συνάρτηση C:

```
1 int accumulate () {  
2     int i, j;  
3     for (i = 0 ; i < 100 ; i++) {  
4         j += i ;  
5     return j ;  
6 }
```

οδηγεί στον ακόλουθο συμβολικό κώδικα για έναν επεξεργαστή Microblaze:

addk r4, r0, r0 ; μηδένισε r4 (κράτα i)
addk r3, r3, r4 ; j = j + i

\$L9:

addik r4, r4, 1 ; i = i + 1
addik r18, r0, 99 ; r18 <- 99
bgeid r18, \$L9 ; διακλάδωση με καθυστέρηση εάν ίσο
addk r3, r3, r4 ; j = j + i -> θέση διακλάδωσης με καθυστέρηση

Σημείωση: Εντολή bgeid

Η εντολή διακλάδωσης με καθυστέρηση είναι η bgeid, η οποία είναι μια "διακλάδωση εάν μεγαλύτερο ή ίσο με καθυστέρηση" ("branch if-greater – or – equal delayed"). Η εντολή αμέσως μετά την διακλάδωση αντιστοιχεί στο σώμα βρόχου $j = j + i$. Επειδή πρόκειται για μια εντολή διακλάδωσης με καθυστέρηση, θα εκτελεστεί ανεξάρτητα από το εάν ο διακλάδωση υπό συνθήκη ακολουθηθεί ή όχι.

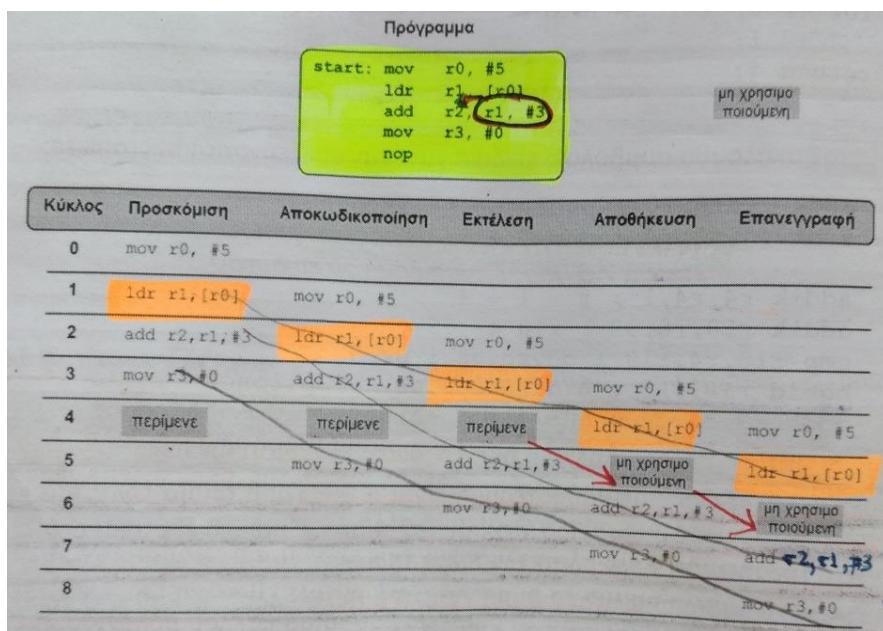
7.1.3 Κίνδυνοι Δεδομένων

Μια δεύτερη αιτία των αναμονών διοχέτευσης είναι οι κίνδυνοι δεδομένων: καθυστερήσεις διοχέτευσης που προκαλούνται από τη μη διαθεσιμότητα δεδομένων. Οι καταχωρητές επεξεργαστών ενημερώνονται στο τέλος κάθε εντολής, κατά τη διάρκεια του σταδίου επανεγγραφής. Όμως τι γίνεται αν τα δεδομένα απαιτούνται πριν να έχουν ενημερώσει ένα καταχωρητή του επεξεργαστή; Παρόλα αυτά, όπως υποδεικνύεται στο διάγραμμα διοχέτευσης στην Εικόνα 73, το στάδιο επανεγγραφής έρχεται δύο κύκλους πίσω από το στάδιο εκτέλεσης. Μια εντολή που φτάνει στο στάδιο επανεγγραφής είναι δύο εντολές μετά από την εντολή που εκτελείται αυτήν τη στιγμή. Στο ακόλουθο απόσπασμα, η εντολή add θα βρίσκεται στο στάδιο αποθήκευσης κατά τη στιγμή που η εντολή πον θα φτάσει στο στάδιο επανεγγραφής και η άθροιση θα έχει ήδη ολοκληρωθεί.

mov r0, #5

add r1, r0, r1

Σε μια διοχέτευση RISC, αυτό αντιμετωπίζεται μέσω υλικού εξαρτήσεων και παύσεων διοχέτευσης (interlock). Το υλικό interlock διοχέτευσης, παρακολουθεί τα μοτίβα ανάγνωσης / εγγραφής όλων των εντολών, που ρέουν αυτήν τη στιγμή στη διοχέτευση RISC και εξασφαλίζει ότι αντλούν δεδομένα από την σωστή πηγή.



Εικόνα 75 - Παράδειγμα κινδύνου δεδομένων

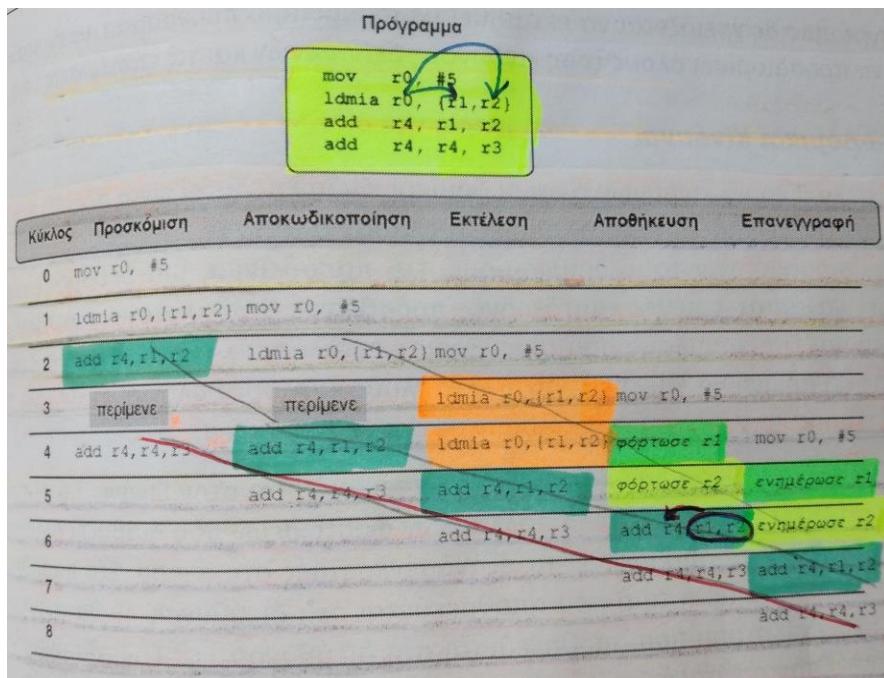
Σημείωση: Επίλυση εξαρτήσεων με προώθηση

Ας εξετάσουμε ξανά το προηγούμενο παράδειγμα. Όταν η εντολή add βρίσκεται στο στάδιο εκτέλεσης, θα χρησιμοποιήσει το αποτέλεσμα της εντολής πον όσο αυτό θα βρίσκεται στο στάδιο αποθήκευσης (buffer) της διοχέτευσης. Αυτή η δραστηριότητα ονομάζεται προώθηση (forwarding) και γίνεται αυτόματα από τον επεξεργαστή.

Σημείωση: Επεξήγηση παραδείγματος 75

Σε ορισμένες περιπτώσεις, η προώθηση δεν είναι δυνατή επειδή τα δεδομένα απλά δεν είναι ακόμα διαθέσιμα. Αυτό συμβαίνει όταν μια εντολή ανάγνωσης από τη μνήμη ακολουθείται από μια εντολή που χρησιμοποιεί τα δεδομένα που προέρχονται από τη μνήμη. Ένα παράδειγμα αυτής της περίπτωσης φαίνεται στην Εικόνα 75. Η

δεύτερη εντολή προσκομίζει δεδομένα από τη μνήμη και τα αποθηκεύει στον καταχωρητή r1. Η ακόλουθη εντολή add χρησιμοποιεί τα δεδομένα από τον καταχωρητή ως τελεστέο. Στον κύκλο 4, η εντολή add φτάνει στο στάδιο εκτέλεσης. Ωστόσο, κατά τη διάρκεια του ίδιου κύκλου ρολογιού, η εντολή ldmia εξακολουθεί να προσπελαύνει τη μνήμη δεδομένων. Η νέα τιμή του r1 είναι διαθέσιμη μόνο στην αρχή του κύκλου 5. Επομένως, το υλικό interlock θα αναβάλει όλα τα στάδια που προηγούνται του σταδίου αποθήκευσης στον κύκλο 4. Ξεκινώντας στον κύκλο 5, ολόκληρη η διοχέτευση κινείται ξανά προς τα εμπρός, αλλά λόγω της αναμονής στον κύκλο 4, μια αχρησιμοποίητη θέση διοχέτευσης εξουδετερώνεται στους κύκλους 5 και 6.



Εικόνα 76 - Παράδειγμα δομικού κινδύνου

Σημείωση: Μειονέκτημα κινδύνων δεδομένων

Οι κίνδυνοι δεδομένων ενδέχεται να επιμηκύνουν τον χρόνο εκτέλεσης μιας εντολής που κανονικά θα τελείωνε σε μόλις πέντε κύκλους ρολογιού.

7.1.4 Δομικοί Κίνδυνοι

Σημείωση: Παράδειγμα Δομικού κινδύνου

Η τρίτη κατηγορία κινδύνων είναι οι δομικοί κίνδυνοι. Αυτοί είναι οι κίνδυνοι που προκαλούνται από εντολές οι οποίες απαιτούν από έναν επεξεργαστή περισσότερους πόρους από τους διαθέσιμους. Για παράδειγμα, μια δεδομένη εντολή μπορεί να απαιτεί πέντε ταυτόχρονες προσθέσεις ενώ υπάρχει μόνο μία ALU διαθέσιμη. Για την υλοποίηση μιας τέτοιας εντολής, το στάδιο εκτέλεσης της εντολής θα πρέπει να επεκταθεί σε πολλαπλούς κύκλους ρολογιού, ενώ τα στάδια διοχέτευσης πριν από αυτό θα τεθούν σε αναμονή (stalled).

Ένα άλλο παράδειγμα δομικού κινδύνου απεικονίζεται στην Εικόνα 76. Η εντολή ldmia (ARM) είναι μια εντολή πολλαπλών φορτώσεων που διαβάζει διαδοχικές θέσεις μνήμης και αποθηκεύει τις εξαγόμενες τιμές στη μνήμη. Στο παράδειγμα που παρουσιάζεται, η τιμή που αποθηκεύεται στη διεύθυνση r0 θα αντιγραφεί στον r1,



ενώ η τιμή που αποθηκεύεται στη διεύθυνση $r0 + 4$. Θα αντιγραφεί στον $r2$. Όταν η εντολή `ldmia` φτάσει στο στάδιο εκτέλεσης, το στάδιο εκτέλεσης θα είναι απασχολημένο για δύο κύκλους ρολογιού για να υπολογίσει τις διευθύνσεις μνήμης $r0$ και $r0+4$. Επομένως, όλα τα στάδια της διοχέτευσης πριν από το στάδιο εκτέλεσης διακόπτονται για ένα μόνο κύκλο ρολογιού. Μετά από αυτό, η διοχέτευση προχωρά κανονικά.

Σημείωση: Αιτίες πρόκλησης δομικών κινδύνων

Ένας δομικός κίνδυνος προκαλείται από την αρχιτεκτονική του επεξεργαστή, αλλά μπορεί να έχει ένα μεγάλο πλήθος από αιτίες: το πλάτος των θυρών μνήμης, τον αριθμό των μονάδων εκτέλεσης στη διαδρομή δεδομένων ή τους περιορισμούς στους διαύλους επικοινωνίας.

8 Κεφάλαιο - Σύστημα στο Ολοκληρωμένο

Η έννοια του Συστήματος στο Ολοκληρωμένο

Σημείωση: Εξαρτήματα ενός SoC

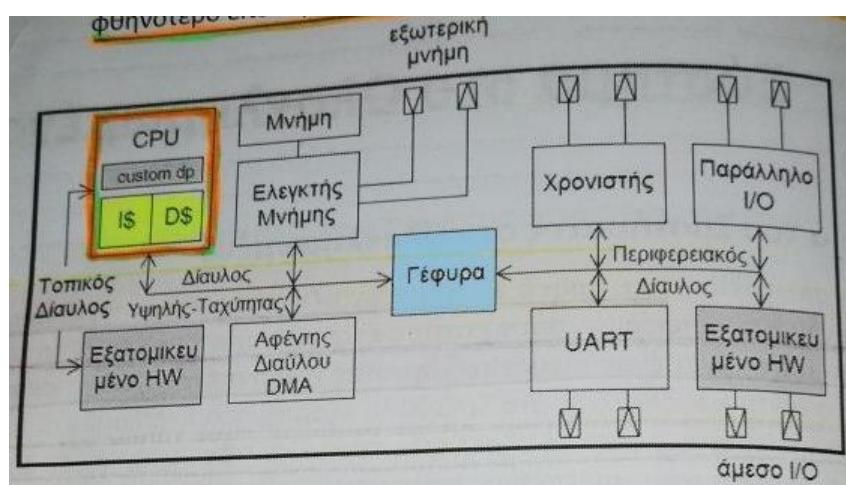
Η Εικόνα 77 απεικονίζει ένα τυπικό Σύστημα στο Ολοκληρωμένο (System-on-chip, SoC). Συνδυάζει διάφορα εξαρτήματα σε ένα σύστημα διαύλου. Ένα από αυτά τα εξαρτήματα είναι ένας μικροεπεξεργαστής (τυπικά ένας RISC) ο οποίος λειτουργεί ως ο κεντρικός ελεγκτής στο SoC. Άλλα εξαρτήματα περιλαμβάνουν τη μνήμη σε ολοκληρωμένο, τις εκτός-ολοκληρωμένου διεπαφές μνήμης, τα αφοσιωμένα περιφερειακά, τους συνεπεξεργαστές υλικού και τους συνδέσμους επικοινωνίας εξαρτήματος-προς-εξάρτημα.

Σημείωση: Διαφορά PC και SoC

Μια συγκεκριμένη διαμόρφωση αυτών των στοιχείων ονομάζεται πλατφόρμα (platform). Ακριβώς όπως ένας προσωπικός υπολογιστής είναι μια πλατφόρμα για τους υπολογισμούς γενικού σκοπού, ένα σύστημα σε ολοκληρωμένο είναι μια πλατφόρμα για υπολογισμούς εξειδικευμένους-στην-περιοχή, δηλαδή για ένα σύνολο εφαρμογών που είναι τυπικές μιας δεδομένης περιοχής εφαρμογής.

Σημείωση: Χαρακτηριστικά SoC: α) Εξειδίκευση, β) Ευελιξία

- **Η εξειδίκευση (specialization) της πλατφόρμας** εξασφαλίζει ότι η αποδοτικότητα της επεξεργασίας είναι υψηλότερη σε σύγκριση με εκείνη των λύσεων γενικού σκοπού. Η αυξημένη αποδοτικότητα επεξεργασίας σημαίνει χαμηλότερη κατανάλωση ενέργειας (μεγαλύτερη διάρκεια ζωής μπαταρίας) ή μεγαλύτερη απόλυτη επίδοση.
- **Η ευελιξία (flexibility) της πλατφόρμας** εξασφαλίζει ότι είναι μια επαναχρησιμοποιήσιμη λύση που λειτουργεί σε πολλαπλές εφαρμογές. Ως αποτέλεσμα, το κόστος σχεδιασμού-ανά-εφαρμογή μειώνεται, οι εφαρμογές μπορούν να αναπτυχθούν γρηγορότερα και το ίδιο το SoC γίνεται φθηνότερο επειδή μπορεί να κατασκευαστεί για μεγαλύτερη αγορά.



Εικόνα 77 - Γενικό πρότυπο για Σύστημα στο Ολοκληρωμένο

8.1.1 Ο Θίασος των Παικτών

Σημείωση: Διαστάσεις Εικόνας 77

Μια αρχιτεκτονική όπως αυτή στην Εικόνα 77 μπορεί να αναλυθεί σε τέσσερις ορθογώνιες διαστάσεις: έλεγχος, επικοινωνία, υπολογισμός και αποθήκευση. Ο ρόλος του κεντρικού ελεγκτή δίνεται στον μικροεπεξεργαστή, ο οποίος είναι υπεύθυνος να εκδίδει σήματα ελέγχου προς και να συλλέγει σήματα κατάστασης από τα διάφορα εξαρτήματα του συστήματος. Ο μικροεπεξεργαστής μπορεί να έχει ή να μην έχει τοπική μνήμη εντολών.

Σημείωση: Σημασία CPU στο SoC

Τα σύμβολα I\$ και D\$ στην Εικόνα 77 αντιπροσωπεύουν τις κρυφές μνήμες εντολών και δεδομένων στον μικροεπεξεργαστή. Στο πλαίσιο της αρχιτεκτονικής SoC, αυτές οι κρυφές μνήμες θα είναι χρήσιμες μόνο για το λογισμικό που εκτελείται στον μικροεπεξεργαστή. Αυτό το προφανές γεγονός έχει μια συχνά αγνοημένη συνέπεια: κάθε φορά που ο μικροεπεξεργαστής χρειάζεται να αλληλεπιδράσει με ένα περιφερειακό, τα δεδομένα μετακινούνται έξω από την κρυφή μνήμη. Η ιεραρχία μνήμης, η οποία καθιστά την CPU τόσο γρήγορη και ισχυρή, δεν λειτουργεί για μετακίνηση δεδομένων. Ένας μικροεπεξεργαστής σε ένα SoC είναι επομένως χρήσιμος ως κεντρικός ελεγκτής, όμως δεν είναι πολύ αποτελεσματικός όταν μεταφέρει τα δεδομένα στο σύστημα.

Σημείωση: Δίαυλοι και γρ. ελέγχου των διαύλων SoC – Χάρτης διευθύνσεων μνήμης Soc

Το SoC υλοποιεί την επικοινωνία χρησιμοποιώντας διάυλους με εμβέλεια συστήματος (system-wide). Κάθε δίαυλος είναι μια δέσμη σημάτων που περιλαμβάνει σήματα διευθύνσεων, δεδομένων, ελέγχου και συγχρονισμού. Οι μεταφορές δεδομένων σε ένα δίαυλο εκφράζονται ως λειτουργίες ανάγνωσης και εγγραφής με μια συγκεκριμένη διεύθυνση μνήμης. Οι γραμμές ελέγχου του δίαυλου υποδεικνύουν τη φύση της μεταφοράς (ανάγνωση/εγγραφή, μέγεθος, αφετηρία, προορισμός), ενώ τα σήματα συγχρονισμού εξασφαλίζουν ότι ο αποστολέας και ο παραλήπτης στο δίαυλο ευθυγραμμίζονται στο χρόνο κατά τη διάρκεια της μεταφοράς δεδομένων. Κάθε εξάρτημα συνδεδεμένο σε ένα δίαυλο θα ανταποκριθεί σε ένα συγκεκριμένο εύρος διευθύνσεων μνήμης. Το σύνολο των εξαρτημάτων μπορεί έτσι να εκπροσωπεύται σε έναν χάρτη διευθύνσεων (address map), έναν καταλόγο όλων των σχετικών διευθύνσεων διαύλου-συστήματος.

Σημείωση: Διαίρεση διαύλων Soc σε τμήματα

Είναι συνήθης η διαίρεση των διαύλων Soc σε τμήματα. Κάθε τμήμα συνδέει έναν περιορισμένο αριθμό εξαρτημάτων, ομαδοποιημένα ανάλογα με τις ανάγκες επικοινωνίας τους. Στο παράδειγμα, ένας δίαυλος επικοινωνίας υψηλής ταχύτητας χρησιμοποιείται για να διασυνδέσει τον μικροεπεξεργαστή, μια διασύνδεση μνήμης υψηλής ταχύτητας και ένα ελεγκτή Άμεσης Προσπέλασης Μνήμης (Direct Memory Access – DMA). Ένας DMA είναι μια συσκευή εξειδικευμένη στην εκτέλεση μεταφορών μπλοκ στο δίαυλο, για παράδειγμα για να αντιγράψει μια περιοχή μνήμης σε μία άλλη. Δίπλα σε ένα δίαυλο επικοινωνίας υψηλής ταχύτητας, μπορείτε επίσης να βρείτε έναν περιφερειακό δίαυλο, που προορίζεται για εξαρτήματα χαμηλότερης ταχύτητας, όπως ένα χρονιστή (timer) και περιφερειακά εισόδου-εξόδου. Τμηματοποιημένοι δίαυλοι διασυνδέονται με μια γέφυρα διαύλου (bus bridge), ένα εξάρτημα που μεταφέρει τις μεταφορές διαύλου από ένα τμήμα σε άλλο τμήμα.

Σημείωση: Διαιτητής διαύλου σε SoC

Οι γραμμές ελέγχου διαύλου του κάθε τμήματος διαύλου είναι υπό τις εντολές του αφέντη του διαύλου (bus master), το εξάρτημα που αποφασίζει τη φύση μιας δεδομένης μεταφοράς διαύλου. Οι σκλάβοι διαύλου (bus slaves) θα ακολουθήσουν τις οδηγίες του αφέντη διαύλου. Κάθε τμήμα διαύλου μπορεί να περιέχει έναν ή περισσότερους αφέντες διαύλου. Σε περίπτωση που υπάρχουν πολλαπλοί αφέντες, η ταυτότητα του αφέντη διαύλου μπορεί να πολυπλεχθεί μεταξύ των εξαρτημάτων αφέντη διαύλου κατά το χρόνο εκτέλεσης. Στην περίπτωση αυτή θα χρειαστεί ένας διαιτητής διαύλου (bus arbiter) για να αποφασίσει ποιο εξάρτημα μπορεί να γίνει αφέντης διαύλου για μια συγκεκριμένη μεταφορά διαύλου.

Σημείωση: Λειτουργίες γέφυρας

Μια γέφυρα διαύλου μπορεί να είναι είτε αφέντης είτε σκλάβος, ανάλογα με την κατεύθυνση των μεταφορών. Για παράδειγμα, όταν πηγαίνουμε από τον δίαυλο υψηλής ταχύτητας προς τον περιφερειακό δίαυλο, η γέφυρα διαύλου θα λειτουργήσει ως σκλάβος διαύλου στον δίαυλο υψηλής ταχύτητας και ως αφέντης διαύλου στο περιφερειακό δίαυλο. Κάθε μεταφορά στον δίαυλο υψηλής ταχύτητας και στον περιφερειακό δίαυλο θα γίνεται ανεξάρτητα. Επομένως, η τμηματοποίηση των διαύλων που χρησιμοποιούν γέφυρες διαύλου οδηγεί σε ένα διπλό πλεονέκτημα.

Σημείωση: Πλεονεκτήματα γέφυρας μεταξύ διαύλων και τμηματοποίησης διαύλων

a) Κατ' αρχάς, τα τμήματα διαύλου μπορούν να ομαδοποιούν τα εξαρτήματα με αντίστοιχες ταχύτητες ανάγνωσης και εγγραφής, παρέχοντας έτσι τη βέλτιστη χρήση του διαθέσιμου εύρους ζώνης διαύλου β) Δεύτερον, τα τμήματα διαύλου επιτρέπουν παραλληλία στην επικοινωνία.

8.1.2 Διασυνδέσεις SoC για Εξατομικευμένο Υλικό

Σημείωση: Τρόποι διασύνδεσης SoC με Εξατομικευμένο Υλικό

Στο πλαίσιο αυτού του κεφαλαίου ως "εξατομικευμένη μονάδα" νοείται μια αφοσιωμένη ψηφιακή μηχανή με τη μορφή μηχανής FSMD ή μικροπρογραμματιζόμενης μηχανής. Τελικά, όλο το εξατομικευμένο υλικό θα βρίσκεται υπό τον έλεγχο του κεντρικού επεξεργαστή στο. Η αρχιτεκτονική SoC προσφέρει διάφορες πιθανές διασυνδέσεις υλικού/λογισμικού, για να προσαρτήσει εξατομικευμένες μονάδες υλικού. Μπορείτε να διακρίνετε τρεις προσεγγίσεις στην Εικόνα 77 ως σκιασμένα μπλοκ.

a) Η πιο γενική προσέγγιση είναι να ενσωματωθεί μια εξατομικευμένη μονάδα υλικού ως ένα τυπικό περιφερειακό σε ένα δίαυλο συστήματος. Ο μικροεπεξεργαστής επικοινωνεί με την εξατομικευμένη μονάδα υλικού μέσω της προσπέλασης μνήμης για ανάγνωση/εγγραφή. Φυσικά, οι διευθύνσεις μνήμης που καταλαμβάνονται από την εξατομικευμένη μονάδα υλικού δε μπορούν να χρησιμοποιηθούν για άλλους σκοπούς. Τα ολοκληρωμένα μικροελεγκτών με πολλά διαφορετικά περιφερειακά χρησιμοποιούν συνήθως αυτή τη στρατηγική χαρτογράφησης-μνήμης για την προσάρτηση περιφερειακών.

Σημείωση: Πλεονεκτήματα



Το ισχυρό σημείο αυτής της προσέγγισης είναι ότι ένας καθολικός μηχανισμός επικοινωνίας (λειτουργίες ανάγνωσης/εγγραφής μνήμης) μπορεί να χρησιμοποιηθεί για ένα ευρύ φάσμα εξατομικευμένων μονάδων υλικού.

Σημείωση: Μειονέκτημα

Το αντίστοιχο μειονέκτημα, φυσικά, είναι ότι μια τέτοια προσέγγιση βασισμένη σε δίαυλο για την ενσωμάτωση του υλικού δεν είναι πολύ επεκτάσιμη από την άποψη της απόδοσης: ο δίαυλος συστήματος γίνεται γρήγορα σημείο συμφόρησης όταν απαιτείται έντονη επικοινωνία μεταξύ ενός μικροεπεξεργαστή και των συνδεδεμένων μονάδων υλικού.

β) Ένας δεύτερος μηχανισμός είναι να προσαρτήσουμε εξατομικευμένο υλικό μέσω ενός τοπικού συστήματος διαύλου ή διασύνδεσης συνεπεξεργαστή που παρέχεται από τον μικροεπεξεργαστή. Σε αυτήν την περίπτωση, η επικοινωνία μεταξύ της μονάδας υλικού και του μικροεπεξεργαστή θα ακολουθήσει ένα συγκεκριμένο πρωτόκολλο, που καθορίζεται από το τοπικό σύστημα διαύλου ή τη διασύνδεση συνεπεξεργαστή.

Σημείωση: Πλεονεκτήματα

Σε σύγκριση με τις διασυνδέσεις διαύλου – συστήματος, οι διασυνδέσεις συνεπεξεργαστών έχουν υψηλό εύρος ζώνης και χαμηλό λανθάνοντα χρόνο.

Ο μικροεπεξεργαστής μπορεί επίσης να παρέχει ένα συγκεκριμένο σύνολο εντολών για την επικοινωνία μέσω αυτής της διασύνδεσης.

Σημείωση: Μειονέκτημα

Οι τυπικές διασυνδέσεις συνεπεξεργαστή δεν περιλαμβάνουν κάποια διεύθυνση μνήμης. Αυτός ο τύπος συνεπεξεργαστή απαιτεί προφανώς έναν μικροεπεξεργαστή με διασύνδεση συνεπεξεργαστή ή τοπικού διαύλου.

γ) Οι μικροεπεξεργαστές μπορούν επίσης να παρέχουν ένα μέσο για την ενσωμάτωση μιας διαδρομής δεδομένων εξατομικευμένου υλικού στο εσωτερικό του μικροεπεξεργαστή. Στη συνέχεια, το σύνολο εντολών του μικροεπεξεργαστή επεκτείνεται με πρόσθετες, νέες εντολές για την οδήγηση αυτού του εξατομικευμένου υλικού.

Σημείωση: Πλεονεκτήματα

Το κανάλι επικοινωνίας ανάμεσα στην εξατομικευμένη διαδρομή δεδομένων και τον επεξεργαστή είναι συνήθως διαμέσου του αρχείου καταχωρητών επεξεργαστή, με αποτέλεσμα ένα πολύ υψηλό εύρος ζώνης επικοινωνίας.

Σημείωση: Μειονέκτημα

Ωστόσο, η πολύ στενή ολοκλήρωση του εξατομικευμένου υλικού με έναν μικροεπεξεργαστή σημαίνει επίσης ότι τα παραδοσιακά σημεία συμφόρησης του μικροεπεξεργαστή αποτελούν σημεία συμφόρησης και για τις μονάδες εξατομικευμένου υλικού. Εάν ο μικροεπεξεργαστής βρεθεί σε αναμονή (stalled) εξαιτίας εξωτερικών συμβάντων (όπως είναι ο ρυθμός μεταφοράς προσπελάσεων – μνήμης), η εξατομικευμένη διαδρομή δεδομένων επίσης τίθεται σε αναμονή.



Σημείωση: Τρόποι και Κριτήρια Ενσωμάτωσης Υλικού / Λογισμικού

Δεν υπάρχει μόνο ένας καλός τρόπος για την ολοκλήρωση υλικού και λογισμικού. Υπάρχουν πολλές πιθανές λύσεις, καθεμία με τα πλεονεκτήματα και τα μειονεκτήματά της. Η επιλογή της σωστής προσέγγισης περιλαμβάνει την αντιστάθμιση πολλών παραγόντων, συμπεριλαμβανομένου του απαιτούμενου εύρους ζώνης επικοινωνίας, της πολυπλοκότητας σχεδιασμού της εξατομικευμένης διασύνδεσης υλικού, του λογισμικού του διαθέσιμου χρόνου σχεδιασμού και του συνολικού προϋπολογισμού κόστους.

Τέσσερις Αρχές Σχεδιασμού στην Αρχιτεκτονική SoC

Ένα SoC είναι εξειδικευμένο για ένα συγκεκριμένο πεδίο εφαρμογής. Ο στόχος είναι να αποσαφηνιστούν τέσσερις αρχές σχεδιασμού που διέπουν την πλειονότητα των μοντέρνων αρχιτεκτονικής SoC. Αυτές οι τέσσερις αρχές περιλαμβάνουν a) ετερογενείς και κατανεμημένες επικοινωνίες b) ετερογενή και κατανεμημένη επεξεργασία δεδομένων, c) ετερογενή και κατανεμημένη αποθήκευση και d) ιεραρχικό έλεγχο.

8.1.3 Ετερογενής και Κατανεμημένη Επεξεργασία Δεδομένων

Ένα πρώτο προεξέχον χαρακτηριστικό μιας αρχιτεκτονικής SoC είναι η ετερογενής και κατανεμημένη επεξεργασία δεδομένων. Ένα SoC μπορεί να περιέχει πολλαπλές ανεξάρτητες (κατανεμημένες) υπολογιστικές μονάδες. Επιπλέον, αυτές οι μονάδες μπορεί να είναι ετερογενείς.

Σημείωση: Μορφές παραλληλίας στην επεξεργασία δεδομένων

Μπορούμε να διακρίνουμε τρεις μορφές παραλληλίας επεξεργασίας δεδομένων. Η πρώτη είναι η παραλληλία σε επίπεδο λέξης η οποία επιτρέπει την παράλληλη επεξεργασία πολλαπλών bits σε μια λέξη. Η δεύτερη είναι η παραλληλία σε επίπεδο-λειτουργίας η οποία επιτρέπει την εκτέλεση πολλαπλών εντολών ταυτόχρονα. Η τρίτη είναι η παραλληλία επιπέδου – διεργασίας (task-level) η οποία επιτρέπει την ανεξάρτητη εκτέλεση πολλαπλών ανεξάρτητων νημάτων ελέγχου. η παραλληλία σε επίπεδο – λέξης και η παραλληλία σε επίπεδο – λειτουργίας είναι διαθέσιμες σε όλες τις αρχιτεκτονικές μηχανών που συζητήσαμε μέχρι τώρα: FSMD, Μικροπρογραμματιζόμενες μηχανές, RISC και επίσης SoC. Ωστόσο, μόνο ένα SoC υποστηρίζει πραγματική παραλληλία επιπέδου διεργασίας. Σημειώστε ότι η πολυνημάτωση σε ένα RISC δεν αποτελεί παραλληλία επιπέδου – διεργασίας, είναι ταυτοχρονισμός επιπέδου – διεργασίας πάνω από μια ακολουθιακή μηχανή.

Σημείωση: Σημασία Μορφών Παραλληλίας

Κάθε μία από τις υπολογιστικές μονάδες ενός SoC μπορεί να εξειδικευθεί σε μια συγκεκριμένη λειτουργία. Το συνολικό SoC περιλαμβάνει επομένως μια συλλογή ετερογενών υπολογιστικών μονάδων. Για παράδειγμα, ένα ολοκληρωμένο ψηφιακής επεξεργασίας σήματος σε μια φωτογραφική μηχανή μπορεί να περιέχει εξειδικευμένες μονάδες για την εκτέλεση επεξεργασίας εικόνας. Η υπολογιστική εξειδίκευση είναι βασικό στοιχείο για την υψηλή απόδοση. Επιπλέον, η παρουσία όλων των μορφών παραλληλίας (επίπεδο-λέξης, επίπεδο – λειτουργίας, επίπεδο – διεργασίας) διασφαλίζει ότι ένα SoC μπορεί να εκμεταλλευτεί πλήρως την τεχνολογία.



8.1.4 Ετερογενείς και Κατανεμημένες Επικοινωνίες

Ο κεντρικός δίαυλος σε ένα σύστημα σε ολοκληρωμένο είναι ένας κρίσιμος πόρος. Διαμοιράζεται από πολλά εξαρτήματα σε ένα SoC. Μια προσέγγιση για να αποφευχθεί η μετατροπή αυτού του πόρου σε σημείο συμφόρησης, είναι να χωρίσουμε το δίαυλο σε πολλαπλά τμήματα διαύλου χρησιμοποιώντας γέφυρες διαύλων. Η γέφυρα διαύλου είναι ένας μηχανισμός για τη δημιουργία κατανεμημένων επικοινωνιών επί του ολοκληρωμένου. Οι απαιτήσεις επικοινωνίας επί του ολοκληρωμένου, εμφανίζουν συνήθως μεγάλες διακυμάνσεις σε ένα SoC. Συνεπώς, οι μηχανισμοί διασύνδεσης SoC θα πρέπει να είναι ετερογενείς. Μπορεί να υπάρχουν κοινόχρηστοι δίαυλοι, συνδέσεις σημείου-σε-σημείο, σειριακές συνδέσεις και παράλληλες συνδέσεις.

Οι ετερογενείς και κατανεμημένες επικοινωνίες SoC δίνουν τη δυνατότητα σε έναν σχεδιαστή να εκμεταλλευτεί το εύρος ζώνης επικοινωνίας στο ολοκληρωμένο. Στη σύγχρονη τεχνολογία, αυτό το εύρος ζώνης είναι εξαιρετικά υψηλό.

8.1.5 Ετερογενής και Κατανεμημένη αποθήκευση

Ένα **τρίτο χαρακτηριστικό της αρχιτεκτονικής Συστήματος-σε-Ολοκληρωμένο είναι μια κατανεμημένη και ετερογενής αρχιτεκτονική αποθήκευσης**. Αντί για μια μονή, κεντρική μνήμη, ένα SoC θα χρησιμοποιήσει μια συλλογή από αφοσιωμένες μνήμες. Οι επεξεργαστές και οι μικρο-κωδικοποιημένες μηχανές μπορεί να περιέχουν τοπικές μνήμες εντολών. Οι επεξεργαστές μπορεί επίσης να χρησιμοποιούν κρυφές μνήμες για να διατηρούν τα τοπικά αντίγραφα των δεδομένων και των εντολών. Οι συνεπεξεργαστές και τα άλλα ενεργά εξαρτήματα θα χρησιμοποιούν τοπικά αρχεία καταχωρητών.

Σημείωση: Κατηγορίες (Είδη) Μνημών

Αυτή η αποθήκευση υλοποιείται με μια συλλογή από μνήμες διαφορετικών τεχνολογιών. Υπάρχουν σήμερα πέντε μεγάλες κατηγορίες αποθηκευτικών χώρων με βάση το πυρίτιο.

- **Οι καταχωρητές είναι ο ταχύτερος τύπος διαθέσιμης μνήμης.** Οι καταχωρητές ονομάζονται επίσης μνήμη πρώτου επιπέδου (foreground memory).
- **Η Δυναμική Μνήμη Τυχαίας Προσπέλασης (Dynamic Random Access Memory – DRAM)** παρέχει φθηνή αποθήκευση σε πολύ υψηλές πυκνότητες. Οι DRAM και όλες οι ακόλουθες κατηγορίες ονομάζονται μνήμη παρασκηνίου (background memory). Οι μνήμες DRAM συνήθως έρχονται σε ξεχωριστή συσκευασία (package).
- **Η Στατική Μνήμη Τυχαίας Προσπέλασης (Static Random Access Memory – SRAM)**, χρησιμοποιείται όπου απαιτείται γρήγορη αποθήκευση ανάγνωσης και εγγραφής. Η SRAM έχει χαμηλότερη πυκνότητα και υψηλότερη κατανάλωση ισχύος από την DRAM. Δεν χρησιμοποιείται για τον κύριο όγκο αποθήκευσης του υπολογιστή, αλλά για εξειδικευμένες εργασίες όπως οι κρυφές μνήμες, προσωρινές μνήμες βίντεο (video buffers).

- **Η Μη Πτητική Μνήμη Ανάγνωσης Μόνο (Non-volatile Read-Only Memory – NVROM)** χρησιμοποιείται για εφαρμογές που απαιτούν μόνο προσπέλαση ανάγνωσης σε μια μνήμη, όπως για παράδειγμα για την αποθήκευση των εντολών ενός προγράμματος. Οι μη πτητικές μνήμες έχουν μεγαλύτερη πυκνότητα από την SRAM.
- **Η Μη Πτητική Μνήμη Τυχαίας Προσπέλασης (Non-volatile Random Access Memory – NVRAM)** χρησιμοποιείται για τις εφαρμογές που χρειάζονται μνήμες ανάγνωσης και εγγραφής οι οποίες δεν χάνουν τα δεδομένα όταν αφαιρεθεί η τροφοδοσία.

Σημείωση: Βασικά Χαρακτηριστικά Διαφορετικών Τύπων Μνήμης

Ο Πίνακας 8.1 συνοψίζει τα βασικά χαρακτηριστικά αυτών των διαφορετικών τύπων μνήμης. Οι εγγραφές στον πίνακα έχουν την ακόλουθη σημασία.

- Το μέγεθος κυττάρου (cell size) είναι η επιφάνεια πυριτίου που απαιτείται για την αποθήκευση ενός μόνο bit.
- Ο χρόνος συγκράτησης (retention time) εκφράζει πόσο μπορεί να κρατηθεί ένα bit σε μια μη-τροφοδοτούμενη μνήμη.
- Ο μηχανισμός διευθυνσιοδότησης (addressing) δείχνει τον τρόπο ανάκτησης των bit από τη μνήμη.
- Ο χρόνος προσπέλασης (access time) είναι ο χρόνος που απαιτείται για την προσκόμιση ενός στοιχείου δεδομένων από τη μνήμη.
- Η κατανάλωση ισχύος (power consumption) είναι μια ποιοτική εκτίμηση για την κατανάλωση ενέργειας μιας μνήμης.

Πίνακας 8.1 Τύποι μνημών

Τύπος	Καταχωρητής Αρχείο καταχωρητών	DRAM	SRAM	NVROM (ROM, PROM, EPROM)	NVRAM, (Flash, EEPROM)
Μέγεθος κυττάρου (bit)	10 τρανζίστορ	1 τρανζίστορ	4 τρανζίστορ	1 τρανζίστορ	1 τρανζίστορ
Χρόνος συγκράτησης	0	Δεκάδες ms	0	∞	10 χρόνια
Διευθυνσιοδότηση	Υπονοούμενη	Πολυπλεγμένη	Μη –	Μη –	Μη –
Χρόνος προσπέλασης	Λιγότερο από 1 ns	Λιγότερο από 20 ns	Λιγότερο από 10 ns	20 ns	20 ns (ανάγνωση) 100μs (εγγραφή)
Κατανάλωση ισχύος	Υψηλή	Χαμηλή	Υψηλή	Πολύ χαμηλή	Πολύ χαμηλή
Ανθεκτικότητα Εγγραφής	∞	∞	∞	1-φορά	Ένα εκατομμύριο φορές



Σημείωση: Προβλήματα διαμοιρασμού δεδομένων μεταξύ εφαρμογών

Η παρουσία κατανεμημένων αποθηκευτικών χώρων περιπλέκει σημαντικά την έννοια ενός κεντρικού χώρου διευθύνσεων μνήμης, ο οποίος είναι τόσο χρήσιμος στο SoC. Όσο τα δεδομένα μέσα σε αυτές τις κατανεμημένες μνήμες είναι τοπικά διαθέσιμα σε ένα μόνο εξάρτημα, αυτό δεν προκαλεί κανένα πρόβλημα. Ωστόσο γίνεται προβληματικό όταν τα δεδομένα πρέπει να διαμοιράζονται μεταξύ των στοιχείων α) Πρώτον, όταν πολλαπλά αντίγραφα ενός μόνο στοιχείου δεδομένων υπάρχουν σε διαφορετικές μνήμες, όλα αυτά τα αντίγραφα πρέπει να συνάδουν. β) Δεύτερον, η ενημέρωση ενός διαμοιραζόμενου στοιχείου δεδομένων πρέπει να υλοποιηθεί με τρόπο που δεν θα παραβιάζει τις εξαρτήσεις δεδομένων μεταξύ των εξαρτημάτων που μοιράζονται το στοιχείο δεδομένων.

8.1.6 Ιεραρχικός έλεγχος

Η τελική ιδέα στην αρχιτεκτονική ενός SoC είναι η ιεραρχία του ελέγχου μεταξύ των εξαρτημάτων. Μια ιεραρχία ελέγχου σημαίνει ότι ολόκληρο το SoC λειτουργεί ως ενιαία λογική οντότητα. Αυτό σημαίνει ότι τα εξαρτήματα ενός SoC λαμβάνουν εντολές από ένα κεντρικό σημείο ελέγχου. Μπορούν να λειτουργούν χαλαρά, σχεδόν ανεξάρτητα, μεταξύ τους, αλλά σε κάποιο σημείο πρέπει να συγχρονίζονται και να δίνουν αναφορές στο κεντρικό σημείο ελέγχου.

Από την πλευρά του συνεπεξεργαστή, το εξατομικευμένο υλικό θα περιμένει αρχικά τους τελεστέους από τον περιφερειακό δίαυλο, στην συνέχεια θα τους επεξεργαστεί και τελικά θα σημάνει την ολοκλήρωση της λειτουργίας (για παράδειγμα, θέτοντας μια σημαία κατάστασης). Το λογισμικό στον επεξεργαστή RISC και οι δραστηριότητες στον συνεπεξεργαστή δεν είναι συνεπώς ανεξάρτητες. Ο τοπικός ελεγκτής στον συνεπεξεργαστή μπορεί να αναπτυχθεί χρησιμοποιώντας μια FSM ή μια τεχνική μικροπρογραμματισμού. Ο επεξεργαστής RISC θα διατηρήσει το συνολικό έλεγχο στο σύστημα και θα διανείμει εντολές στο εξατομικευμένο υλικό.

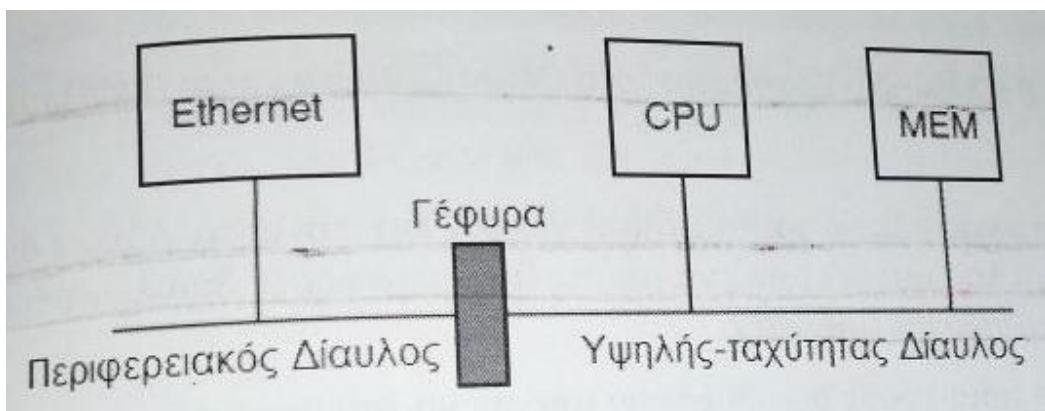
Ο σχεδιασμός μιας καλής ιεραρχίας ελέγχου είναι ένα δύσκολο πρόβλημα. Από τη μία πλευρά, θα πρέπει να εκμεταλλευτεί την κατανεμημένη φύση του SoC όσο το δυνατόν καλύτερα – αυτό σημαίνει να κάνουμε πολλά πράγματα παράλληλα. Από την άλλη πλευρά, θα πρέπει να ελαχιστοποιήσει τον αριθμό των συγκρούσεων που προκύπτουν ως αποτέλεσμα της εκτέλεσης πραγμάτων παράλληλα. Τέτοιες συγκρούσεις μπορεί να είναι αποτέλεσμα της υπερφόρτωσης του διαθέσιμου συστήματος – διαύλου ή του ρυθμού μεταφοράς της μνήμης ή της υπερβολικής χρονοδρομολόγησης ενός συνεπεξεργαστή.

Προβλήματα

Πρόβλημα 8.1

Εξετάστε το απλό μοντέλο SoC στην Εικόνα 78. Ας υποθέσουμε ότι ο δίαυλος υψηλής ταχύτητας μπορεί να μεταφέρει 200MWord/s και ο περιφερειακός δίαυλος μπορεί να μεταφέρει 30 MWord/s. Η CPU δεν έχει κρυφή μνήμη και απαιτεί από το σύστημα τις ακόλουθες ροές δεδομένων: 80 MWord/s εύρος ζώνης ανάγνωσης – μόνο

για εντολές, 40 MWord/s εύρος ζώνης ανάγνωσης / εγγραφής για δεδομένα και 2 MWord/s για είσοδο / έξοδο πακέτων Ethernet.

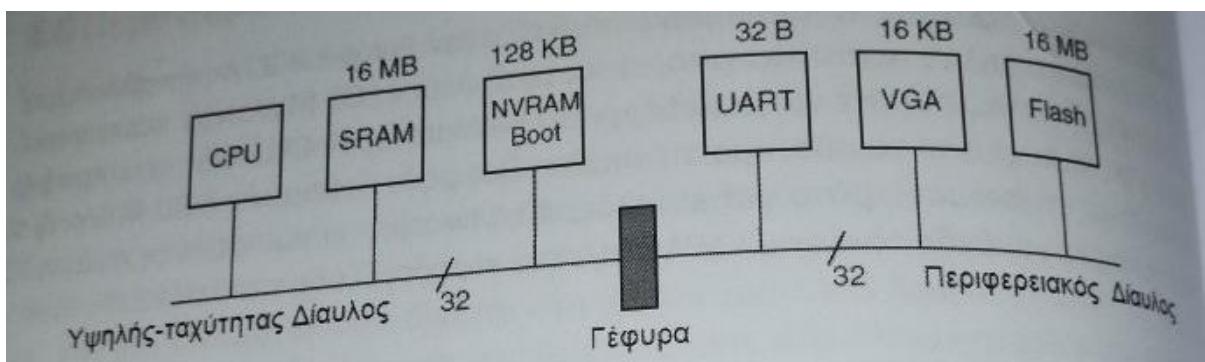


Εικόνα 78 - Μοντέλο Συστήματος στο Ολοκληρωμένο για το πρόβλημα 8.1

- (α) Ποιο είναι το εύρος ζώνης δεδομένων μέσω της γέφυρας διαύλου

Πρόβλημα 8.2

Το σύστημα περιέχει ένα δίαυλο υψηλής ταχύτητας και ένα περιφερειακό δίαυλο και οι δύο με χώρο διεύθυνσης 32 bit και οι δύο μεταφέρουν λέξεις (32 bit). Τα εξαρτήματα του συστήματος περιλαμβάνουν ένα RISC, 16MB μνήμη RAM, 128KB μη πτητική μνήμη προγράμματος, 16 MB μνήμη Flash.



Εικόνα 79 - Μοντέλο Συστήματος στο Ολοκληρωμένο για το Πρόβλημα 8.2

- (α) Σχεδιάστε έναν πιθανό χάρτη μνήμης για τον επεξεργαστή. Λάβετε υπόψη ότι η Γέφυρα Διαύλου μπορεί μόνο να μετατρέπει μεταφορές διαύλων μέσα σε ένα ενιαίο, συνεχή χώρο διεύθυνσης.
 (β) Καθορίστε ποιο εύρος διευθύνσεων μπορεί να αποθηκευτεί σε κρυφή μνήμη από τον επεξεργαστή. Μια "περιοχή διευθύνσεων κρυφής μνήμης" σημαίνει ότι μια ανάγνωση μνήμης σε μια διεύθυνση σε αυτό το εύρος θα οδηγήσει σε ένα αντίγραφο ασφαλείας αποθηκευμένο στην κρυφή μνήμη.

9 Κεφάλαιο - Αρχές Επικοινωνίας Υλικού / Λογισμικού

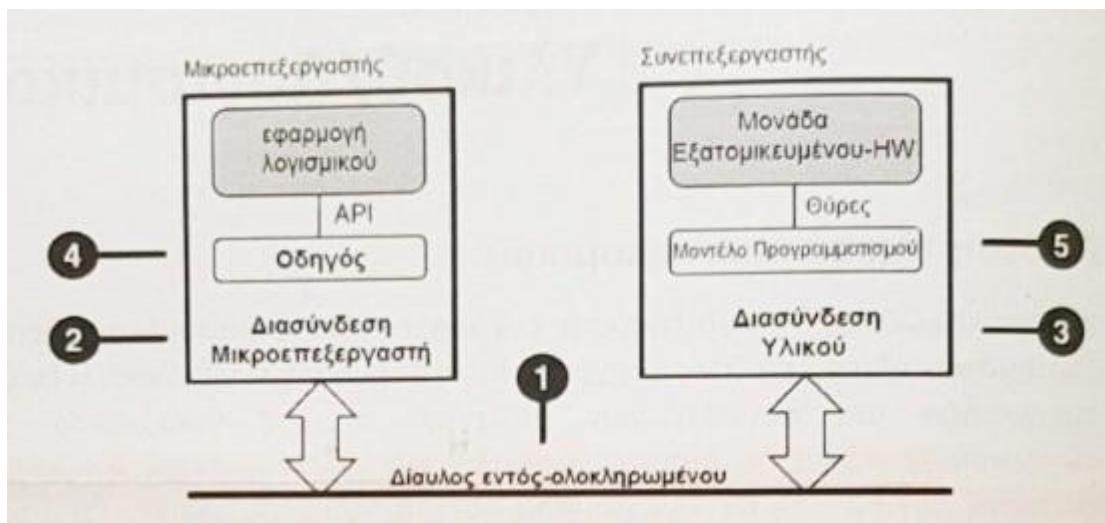
Σύνδεση Υλικού και Λογισμικού

Σημείωση: Στόχος διασύνδεσης Η/S

Ο στόχος της διασύνδεσης υλικού/λογισμικού είναι να συνδέσει την εφαρμογή λογισμικού είναι να συνδέσει την εφαρμογή λογισμικού με την μονάδα εξατομικευμένου – υλικού. Υπάρχουν πέντε στοιχεία που εμπλέκονται.

Σημείωση: Εμπλεκόμενα στοιχεία στη διασύνδεση υλικού – λογισμικού

1. Ο μικροεπεξεργαστής και ο συνεπεξεργαστής συνδέονται και οι δύο σε ένα μηχανισμό επικοινωνίας στο ολοκληρωμένο, όπως ένας δίαυλος εντός ολοκληρωμένου. Ο δίαυλος ενός ολοκληρωμένου (on-chip bus) μεταφέρει δεδομένα από τη μονάδα μικροεπεξεργαστή στη μονάδα εξατομικευμένου-υλικού.
2. Τόσο ο μικροεπεξεργαστής όσο και ο συνεπεξεργαστής χρειάζονται μια διασύνδεση με τον εντός ολοκληρωμένου δίαυλο επικοινωνίας. Η διασύνδεση μικροεπεξεργαστή περιλαμβάνει το υλικό και το χαμηλού επιπέδου υλικολογισμικό (firmware) για να επιτρέψει σε ένα πρόγραμμα λογισμικού να βγει 'έξω' από τον μικροεπεξεργαστή.
3. Η διασύνδεση υλικού περιλαμβάνει το υλικό που απαιτείται για τη σύνδεση του συνεπεξεργαστή στο υποσύστημα επικοινωνίας ενός ολοκληρωμένου. Για παράδειγμα, στην περίπτωση ενός διαύλου εντός ολοκληρωμένου, η διασύνδεση υλικού χειρίζεται δεδομένα που προέρχονται από και πηγαίνουν στο δίαυλο εντός ολοκληρωμένου.



Εικόνα 80 - Η διασύνδεση υλικού / λογισμικού

Σημείωση: Οδηγός Λογισμικού (software driver)

4. Η εφαρμογή λογισμικού συνδέεται στη διασύνδεση μικροεπεξεργαστή μέσω οδηγού λογισμικού (software driver), μιας μικρής μονάδας που περικλείει τις συναλλαγές μεταξύ υλικού και λογισμικού σε κλήσεις συναρτήσεων λογισμικού. Αυτός ο οδηγός μετατρέπει τα λογισμικοκεντρικά παραδείγματα (δείκτες, πίνακες, δομές δομένων μεταβλητού μήκους) σε δομές που είναι κατάλληλες για επικοινωνία με υλικό.

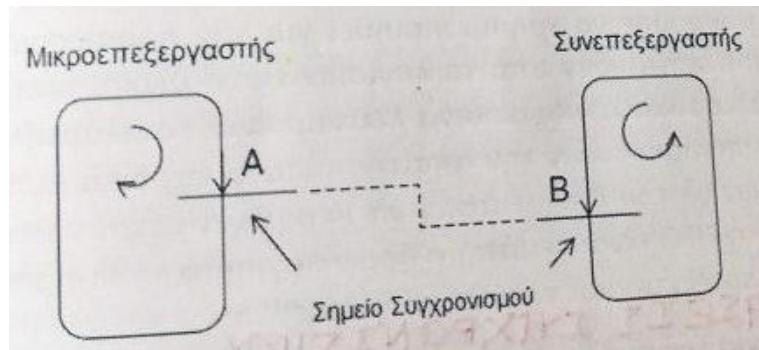
Σημείωση: Μονάδα εξατομικευμένου υλικού

5. Η μονάδα εξατομικευμένου υλικού συνδέεται με τη διασύνδεση εξατομικευμένου υλικού μέσω ενός μοντέλου προγραμματισμού, μιας δομής που παρουσιάζει μια αφαίρεση του υλικού στην εφαρμογή λογισμικού. Η διασύνδεση υλικού ενσωματώνει την μονάδα εξατομικευμένου υλικού, η οποία μπορεί να έχει έναν αυθαίρετο αριθμό θυρών, παραμέτρων.

Σχήματα Συγχρονισμού

Σημείωση: Ορισμός Συγχρονισμού

Πως μπορούμε να εγγυηθούμε ότι η εφαρμογή λογισμικού και η μονάδα εξατομικευμένου υλικού θα παραμείνουν συγχρονισμένες, δεδομένου ότι είναι ανεξάρτητες οντότητες εκτέλεσης; Πως γνωρίζει μια μονάδα υλικού ότι ένα πρόγραμμα λογισμικού επιθυμεί να επικοινωνήσει μαζί της; Η απάντηση σε αυτές τις ερωτήσεις απαιτεί να επιλέξουμε ένα σχήμα συγχρονισμού.



Εικόνα 81 - Σημείο συγχρονισμού

9.1.1 Έννοιες Συγχρονισμού

Σημείωση: Ορισμός συγχρονισμού και σημασία

Ορίζουμε τον συγχρονισμό (synchronization) ως τη δομημένη αλληλεπίδραση δύο, ανεξάρτητων και παράλληλων, εκτός και αν αναφέρεται διαφορετικά, οντότητων. Η Εικόνα 80 απεικονίζει την κεντρική ιδέα του συγχρονισμού. Δύο οντότητες, στην προκειμένη περίπτωση ένας μικροεπεξεργαστής και ένας συνεπεξεργαστής, έχουν η καθεμία ένα ανεξάρτητο νήμα εκτέλεσης. Μέσω του συγχρονισμού, ένα σημείο στο νήμα εκτέλεσης του μικροεπεξεργαστή συνδέεται με ένα σημείο της ροής ελέγχου του συνεπεξεργαστή. Αυτό είναι το σημείο συγχρονισμού. Ο συγχρονισμός πρέπει να εγγυάται ότι, όταν ο μικροεπεξεργαστής βρίσκεται στο σημείο Α, ο συνεπεξεργαστής θα βρίσκεται στο σημείο Β.

Απαιτείται συγχρονισμός για να υποστηριχθεί η επικοινωνία μεταξύ των παραλλήλων υποσυστημάτων: κάθε ομιλητής (talker) πρέπει να έχει έναν ακροατή (listener) να τον ακούει.

Σημείωση: Απαίτηση συγχρονισμού σε ένα SDF

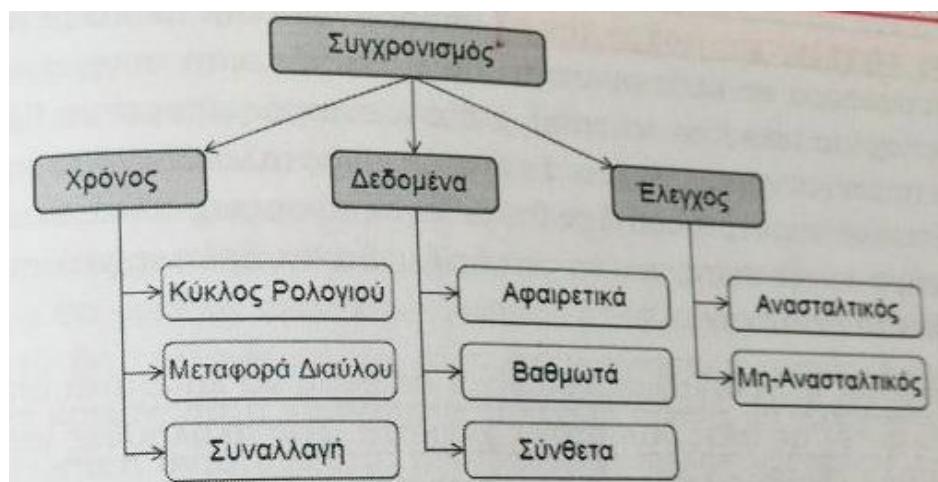
Στη ροή δεδομένων, διαφορετικοί δρώντες επικοινωνούν μεταξύ τους μέσω της ανταλλαγής των συμβόλων. Υποθέστε ότι ένας δρών υλοποιείται στο λογισμικό και ένας άλλος υλοποιείται ως μονάδα εξατομικευμένου υλικού. Επίσης, υποθέστε ότι ο δρών λογισμικού στέλνει σύμβολα στο δρώντα του υλικού. Σύμφωνα με τους

κανόνες της ροής δεδομένων, κάθε σύμβολο που παράγεται πρέπει τελικά να καταναλωθεί και αυτό σημαίνει ότι ο δρών υλικού πρέπει να γνωρίζει πότε ο δρών λογισμικού στέλνει αυτό το σύμβολο. Με άλλα λόγια: οι δρώντες υλικού και λογισμικού θα χρειαστεί να συγχρονιστούν όταν μεταδίδουν ένα σύμβολο. Φυσικά, υπάρχουν πολλοί διαφορετικοί τρόποι για να πραγματοποιήσουμε μια επικοινωνία ροής δεδομένων, ανάλογα με το πώς υλοποιούμε την ακμή ροής δεδομένων. Όμως, ανεξάρτητα από την υλοποίηση, η απαίτηση συγχρονισμού δεν εξαφανίζεται.

Σημείωση: Διαστάσεις Συγχρονισμού

Ο συγχρονισμός είναι ένα ενδιαφέρον πρόβλημα επειδή έχει αρκετές διαστάσεις, καθεμία με διάφορα επίπεδα αφαίρεσης. Η Εικόνα 82 δείχνει τις τρεις διαστάσεις που ενδιαφέρουν: χρόνος, δεδομένα και έλεγχος.

α) Η διάσταση του χρόνου εκφράζει το βαθμό πιστότητας στον οποίο συγχρονίζονται δύο παράλληλες οντότητες. Η ακρίβεια κύκλου-ρολογιού είναι απαραίτητη όταν διασυνδέουμε μεταξύ τους δύο εξαρτήματα υλικού. Η ακρίβειας μεταφοράς διαύλου είναι απαραίτητη όταν ο βαθμός πιστότητας του συγχρονισμού εκφράζεται με όρους ενός συγκεκριμένου πρωτοκόλλου διαύλου, όπως μια μεταφορά δεδομένων από έναν αφέντη σε ένα σκλάβο. Τέλος, η ακρίβεια συναλλαγής απαιτείται όταν ο βαθμός πιστότητας του συγχρονισμού είναι μια λογική συναλλαγή από τη μια οντότητα στην άλλη. Σημειώστε ότι η σημασία του χρόνου ποικίλει ανάλογα με το επίπεδο αφαίρεσης και δεν είναι απαραίτητο πάντα να μετριέται με κλασσικές μονάδες χρόνου. Αντιθέτως, ο συγχρονισμός χρόνου μπορεί επίσης να αναφέρεται σε κύκλους ρολογιού, μεταφορές διαύλου και λογικές μεταφορές. Ο συγχρονισμός χρόνου μπορεί ακόμη και να περιορίζεται σε μια μερική ταξινόμηση, όπως, για παράδειγμα, ότι το A δεν συμβαίνει πριν από το B.



Εικόνα 82 - Διαστάσεις του προβλήματος συγχρονισμού

β) Η διάσταση δεδομένων του συγχρονισμού καθορίζει το μέγεθος του περιέκτη (container) που συμμετέχει στο συγχρονισμό. Όταν δεν υπάρχουν καθόλου δεδομένα, ο συγχρονισμός μεταξύ δύο οντοτήτων είναι αφηρημένος. Ο αφηρημένος συγχρονισμός είναι χρήσιμος για τη διαχείριση της πρόσβασης σε έναν κοινόχρηστο πόρο.

Σημείωση: Διαφορά ανασταλτικού και μη-ανασταλτικού σχήματος αναφορικά με το συγχρονισμό.

γ) Η διάσταση έλέγχου του συγχρονισμού υποδεικνύει τον τρόπο με τον οποίο η τοπική συμπεριφορά σε κάθε οντότητα θα υλοποιήσει το συγχρονισμό. Σε ένα ανασταλτικό σχήμα (blocking scheme), ο συγχρονισμός μπορεί



να θέσει σε αναμονή (stall) το τοπικό νήμα ελέγχου. Σε ένα μη-ανασταλτικό σχήμα (non-blocking scheme), η τοπική συμπεριφορά δεν θα τεθεί σε αναμονή, αλλά αντίθετα θα εκδοθεί ένα σήμα κατάστασης για να υποδείξει ότι το πρωταρχικό στοιχείο συγχρονισμού δεν τα κατάφερε.

Σημείωση: Παραδείγματα Συγχρονισμού – Πρωταρχικά Στοιχεία

9.1.2 Σηματοφόρος

Σημείωση: Ορισμός Σηματοφόρου

Ένας σηματοφόρος (semaphore) είναι ένα πρωταρχικό στοιχείο (primitive) συγχρονισμού το οποίο δεν περιλαμβάνει τη μεταφορά δεδομένων, αλλά ελέγχει την πρόσβαση σε ένα αφηρημένο, κοινόχρηστο πόρο. Ένας σηματοφόρος S είναι ένας κοινόχρηστος πόρος που υποστηρίζει δύο λειτουργίες: αρπαγή (grab) του σηματοφόρου ($P(S)$) και απελευθέρωση (release) του σηματοφόρου ($V(S)$).

Σημείωση: Λειτουργίες του Σηματοφόρου

Αυτές οι λειτουργίες μπορούν να εκτελεστούν από αρκετές ταυτόχρονες οντότητες. Στην περίπτωση αυτή, θα υποθέσουμε ότι υπάρχουν δύο οντότητες που ανταγωνίζονται για τον σηματοφόρο. Τα P και V είναι τα πρώτα γράμματα των ολλανδικών ρημάτων "proberen" και "verhogen", που επιλέχθηκαν από τον επιστήμονα Edsger Dijkstra που πρότεινε τη χρήση σηματοφόρων στο λογισμικό συστήματος.

Σημείωση: Σημασία λειτουργιών $P(S)$ και $V(S)$

Η σημασία των $P(S)$ και $V(S)$ έχει ως εξής. Οι $P(S)$ και $V(S)$ είναι αδιαίρετες λειτουργίες που χειρίζονται την τιμή ενός σηματοφόρου. Αρχικά, η τιμή του σηματοφόρου είναι 1. Η λειτουργία $P(S)$ θα μειώσει το σηματοφόρο κατά ένα. Αν μια οντότητα προσπαθήσει να $P(S)$ το σηματοφόρο ενώ είναι μηδέν, τότε η $P(S)$ θα θέσει σε αναμονή την περαιτέρω εκτέλεση αυτής της οντότητας έως ότου ο σηματοφόρος γίνει μη-μηδενικός. Εν τω μεταξύ, μια άλλη οντότητα μπορεί να αυξήσει το σηματοφόρο καλώντας την $V(S)$. Όταν η τιμή του σηματοφόρου είναι μη-μηδενική, οποιαδήποτε οντότητα η οποία είχε τεθεί σε αναμονή για μια λειτουργία $P(S)$, θα μειώσει το σηματοφόρο και θα προχωρήσει. Σε περίπτωση αποκλεισμού πολλαπλών οντοτήτων σε ένα σηματοφόρο, μια από αυτές, επιλεγμένη τυχαία, θα μπορέσει να προχωρήσει. Η μέγιστη τιμή του βασικού δυαδικού σηματοφόρου είναι 1. Η κλήση $V(S)$ αρκετές φορές δε θα αυξήσει το σηματοφόρο πάνω από 1, αλλά ούτε και θα προκαλέσει αναμονή. Υπάρχουν, επίσης, πιο περίπλοκες υλοποιήσεις σηματοφόρων, όπως οι σηματοφόροι μέτρησης. Για το σκοπό μας όμως, οι δυαδικοί σηματοφόροι είναι επαρκείς.

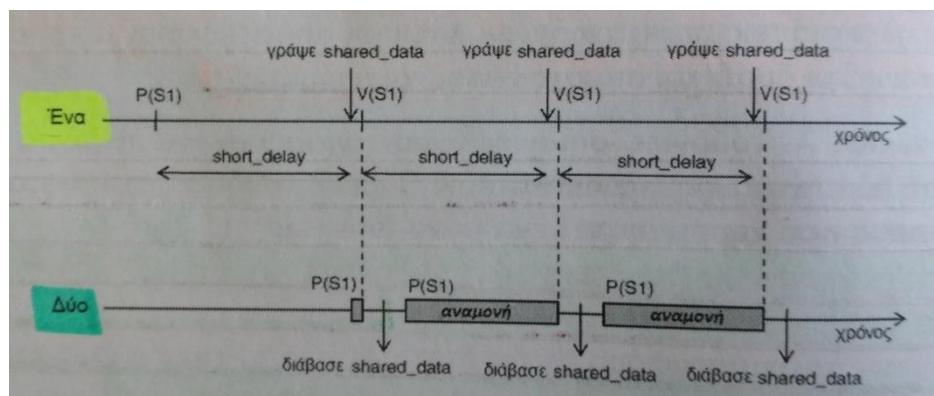
Σημείωση: Περιγραφή συγχρονισμού δύο ταυτόχρονων οντοτήτων

Χρησιμοποιώντας λειτουργίες σηματοφόρου, μπορούμε να περιγράψουμε το συγχρονισμό δύο ταυτόχρονων οντοτήτων. Ο ψευδοκώδικας στη Λίστα 9.1 είναι ένα παράδειγμα που χρησιμοποιεί ένα μόνο σηματοφόρο. Η πρώτη από τις δύο ταυτόχρονες οντότητες πρέπει αν στείλει δεδομένα στη δεύτερη οντότητα μέσω μιας κοινόχρηστης μεταβλητής shared data. Όταν ξεκινήσει η πρώτη οντότητα, μειώνει αμέσως το σηματοφόρο. Η οντότητα δύο, από την άλλη, περιμένει για λίγο, και στη συνέχεια θα τεθεί σε αναμονή για το σηματοφόρο. Εν τω

μεταξύ, η οντότητα ένα θα γράψει στην κοινόχρηστη μεταβλητή και θα αυξήσει το σηματοφόρο. Αυτό θα ξεκλειδώσει τη δεύτερη οντότητα, η οποία μπορεί πλέον να διαβάσει την κοινόχρηστη μεταβλητή. Η στιγμή εκείνη, όπου η οντότητα ένα καλεί την $V(S1)$ και η οντότητα δύο είναι σε αναμονή στην $P(S1)$, είναι ιδιαίτερου ενδιαφέροντος: αποτελεί το σημείο συγχρονισμού μεταξύ των οντοτήτων ένα και δύο.

Σημείωση: Σχήμα ενός σηματοφόρου

Η Εικόνα 83 απεικονίζει την αλληλεπίδραση μεταξύ των οντοτήτων ένα και δύο. Οι διακεκομμένες γραμμές υποδεικνύουν τα σημεία συγχρονισμού. Επειδή η οντότητα δύο συνεχίζει να μειώνει το σηματοφόρο ταχύτερα από ότι μπορεί να τον αυξήσει η οντότητα ένα, η οντότητα δύο πάντα θα τίθεται σε αναμονή.



Εικόνα 83 - Συγχρονισμός με ένα σηματοφόρο

Λίστα 9.1 Συγχρονισμός μονής κατεύθυνσης με έναν σηματοφόρο

```

int shared_data;
semaphore S1;
entity one {
    P(S1);
    while (1) {
        short_delay ();
        shared_data = ... ;
        V (S1); // σημείο συγχρονισμού
    }
}
entity two {
    short_delay ();
    while (1) {
        P(S1); // σημείο συγχρονισμού
        received_data = shared_data;
    }
}

```

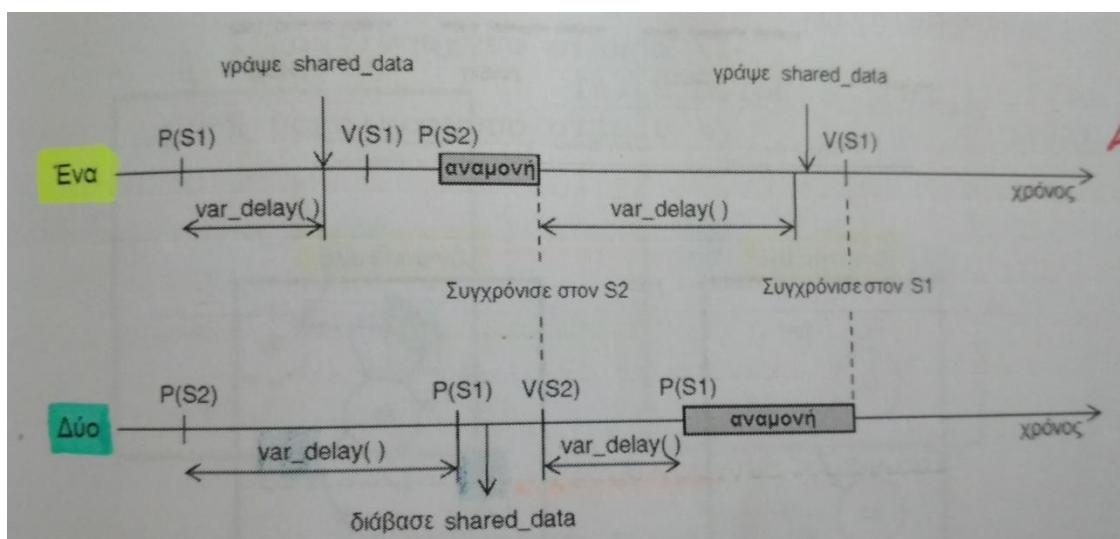
Ωστόσο, αυτό το σχήμα συγχρονισμού δεν είναι τέλειο, επειδή υποθέτει ότι η οντότητα δύο θα φτάνει πάντα πρώτη στο σημείο συγχρονισμού. Υποθέστε τώρα ότι η πιο αργή οντότητα θα ήταν η οντότητα δύο αντί της οντότητας ένα. Σύμφωνα με αυτή την υπόθεση, είναι πιθανό η οντότητα ένα να γράψει στην shared data αρκετές φορές πριν η οντότητα δύο μπορέσει να διαβάσει ένα μεμονωμένο στοιχείο. Πράγματι, η $V(S1)$ δεν θα τεθεί σε αναμονή ακόμη και αν κληθεί αρκετές φορές στη σειρά. Ένα τέτοιο σενάριο δεν είναι δύσκολο να γίνει

αντιληπτό: απλά μετακινήστε την κλήση συνάρτησης short delay() από το βρόχο while της οντότητας ένα, στον βρόχο – while στην οντότητα δύο.

Σημείωση: Σχήμα δύο σηματοφόρων

Αυτή η παρατήρηση οδηγεί στο συμπέρασμα ότι ο γενικός συγχρονισμός δύο ταυτόχρονων οντοτήτων πρέπει να λειτουργεί σε δύο κατευθύνσεις: μία οντότητα πρέπει να είναι σε θέση να περιμένει για την άλλη και αντίστροφα. Στο σενάριο παραγωγού/καταναλωτή που εξηγείται παραπάνω, ο παραγωγός θα πρέπει να περιμένει τον καταναλωτή, εάν ο καταναλωτής είναι αργός. Αντιστρόφως, ο καταναλωτής θα πρέπει να περιμένει τον παραγωγό, εάν ο παραγωγός είναι αργός. Μπορούμε να αντιμετωπίσουμε την κατάσταση των άγνωστων καθυτερήσεων με ένα σχήμα δύο σηματοφόρων, όπως φαίνεται στην Λίστα 9.2

Η Εικόνα 84 απεικονίζει την περίπτωση όπου χρησιμοποιούνται δύο σηματοφόροι. Στον πρώτο συγχρονισμό, η οντότητα ένα είναι ταχύτερη από την οντότητα δύο και ο συγχρονισμός γίνεται χρησιμοποιώντας το σηματοφόρο S2. Στο δεύτερο συγχρονισμό, η οντότητα δύο είναι ταχύτερη και στην περίπτωση αυτή, ο συγχρονισμός γίνεται χρησιμοποιώντας το σηματοφόρο S1.



Εικόνα 84 -Συγχρονισμός με δύο σηματοφόρους

9.1.3 Μονόδρομη και διμερής Χειραψία

Σημείωση: Πότε δεν χρησιμοποιείται σηματοφόρος

Σε παράλληλα συστήματα, οι ταυτόχρονες οντότητες μπορεί να είναι φυσικά διακριτές και η υλοποίηση ενός κεντρικού σηματοφόρου μπορεί να μην είναι εφικτή. Για να χειριστούμε αυτήν την κατάσταση, θα χρησιμοποιήσουμε μια χειραψία (handshake): ένα πρωτόκολλο σηματοδοσίας βασισμένο στα επίπεδα σήματος. Οι έννοιες του συγχρονισμού με βάση τους σηματοφόρους εξακολουθούν να ισχύουν: υλοποιούμε ένα σημείο συγχρονισμού κάνοντας μία οντότητα να περιμένει μια άλλη.

Λίστα 9.2 Συγχρονισμός δύο – κατευθύνσεων με δύο σηματοφόρουν

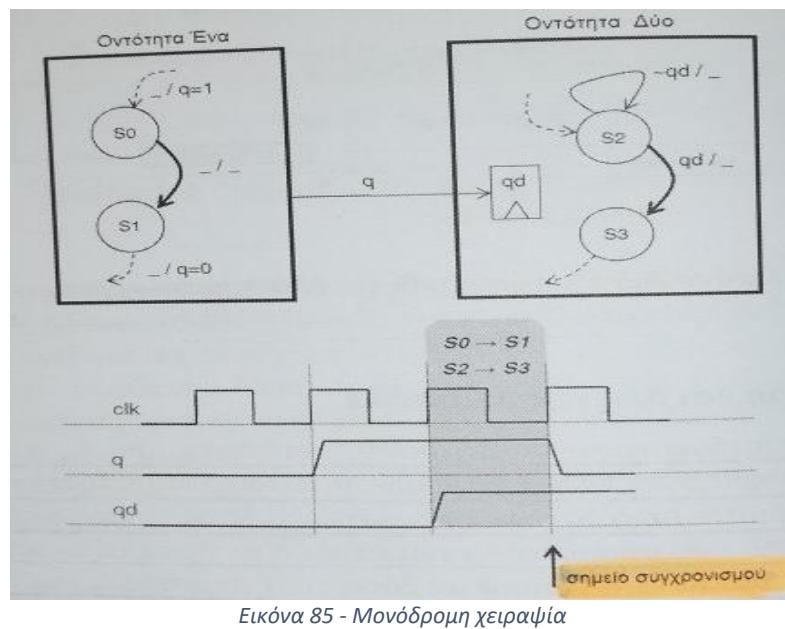
```
int shared_data;
semaphore S1, S2 ;
entity one {
```

```

P(S1) ;
while (1) {
    variable delay ( ) ;
    shared_data = ...;
    V(S1); // σημείο συγχρονισμού 1
    P(S2); // σημείο συγχρονισμού 2
}
}

entity two {
P(S2) ;
while (1) {
    variable_delay ( ) ;
    P(S1); //σημείο συγχρονισμού 1
    received_data = shared_data;
    V(S2); //σημείο συγχρονισμού 2
}
}
}

```

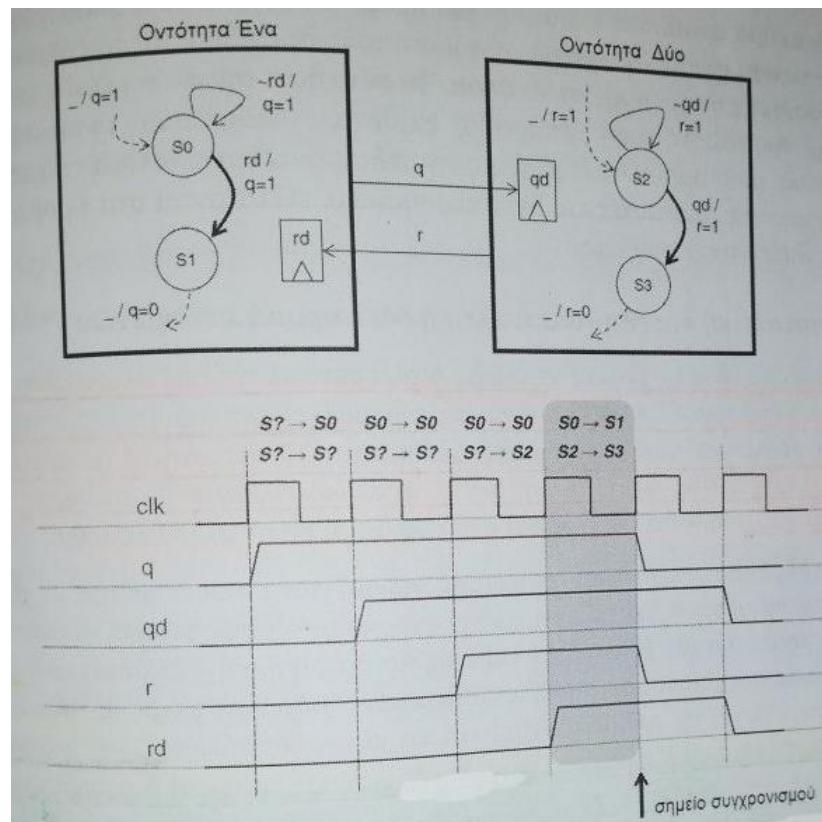


Σημείωση: Μονόδρομη χειραψία

Η πιο απλή υλοποίηση μιας χειραψίας είναι μια μονόδρομη χειραψία, η οποία χρειάζεται ένα μόνο καλώδιο. Η Εικόνα 85 διευκρινίζει την υλοποίηση αυτής της χειραψίας για την περίπτωση δύο μονάδων υλικού.

Σε αυτήν την εικόνα, η οντότητα ένα μεταδίδει ένα σήμα ερωτήματος στην οντότητα δύο. Η οντότητα δύο καταγράφει αυτό το σήμα σε έναν καταχωρητή και χρησιμοποιεί την τιμή του ως μια συνθήκη μετάβασης κατάστασης. Το σημείο συγχρονισμού είναι η μετάβαση της s0 στην s1 στην οντότητα ένα, με τη μετάβαση της s2 στην s3 στην οντότητα δύο. Η οντότητα δύο θα περιμένει την οντότητα ένα έως ότου και οι δύο να μπορούν να κάνουν αυτές τις μεταβάσεις στον ίδιο κύκλο ρολογιού. Η οντότητα ένα πρέπει να θέσει το σήμα επιβεβαίωσης

σε υψηλή τιμή, έναν κύκλο πριν από το πραγματικό σημείο συγχρονισμού, επειδή η είσοδος αιτήματος στην οντότητα δύο καταγράφεται σε έναν καταχωρητή.



Εικόνα 86 - Διμερής χειραψία

Σημείωση: Μειονέκτημα μονόδρομης χειραψίας

Ο περιορισμός μιας μονόδρομης χειραψίας είναι παρόμοιος με τον περιορισμό ενός σχήματος συγχρονισμού ενός-σηματοφόρου: επιτρέπει σε μια μόνο οντότητα να τεθεί σε αναμονή. Για να υποστηρίζουμε εκτελέσεις οποιασδήποτε διάταξης, χρειαζόμαστε μια διμερή χειραψία (two-way handshake) όπως φαίνεται στην Εικόνα 86. Στην περίπτωση αυτή, υλοποιούνται δύο συμμετρικές δραστηριότητες χειραψίας. Κάθε φορά, το σήμα ερωτήματος υψώνεται κατά τη διάρκεια της μετάβασης που προηγείται του σημείου συγχρονισμού. Στη συνέχεια, οι οντότητες περιμένουν μέχρι να λάβουν μια αντίστοιχη απάντηση. Στο διάγραμμα χρονισμού της Εικόνας 86, η οντότητα ένα φτάνει πρώτη στην κατάσταση s_0 και περιμένει. Δύο κύκλους ρολογιού αργότερα, η οντότητα δύο φτάνει στην κατάσταση s_2 . Ο επόμενος κύκλος ρολογιού είναι το σημείο συγχρονισμού: καθώς η οντότητα ένα προχωρά από την s_0 στην s_1 , η οντότητα δύο πραγματοποιεί μια αντίστοιχη μετάβαση από την s_2 στην s_3 . Επειδή η διαδικασία χειραψίας είναι αμφίδρομη, το σημείο συγχρονισμού εκτελείται σωστά ανεξάρτητα από το ποια οντότητα φθάνει πρώτη σε αυτό το σημείο.

9.1.4 Ανασταλτική και Μη-Ανασταλτική Μεταφορά Δεδομένων

Σημείωση: Σημασία σημείου συγχρονισμού



Οι σηματοφόροι και οι χειραψίες είναι διαφορετικοί τρόποι υλοποίησης ενός σημείου συγχρονισμού. Μια διασύνδεση υλικού/λογισμικού χρησιμοποιεί ένα σημείο συγχρονισμού για τη μεταφορά δεδομένων. Η πραγματική μεταφορά δεδομένων υλοποιείται χρησιμοποιώντας μια κατάλληλη διασύνδεση υλικού/λογισμικού.

Όσον αφορά τις λειτουργίες αποστολής/λήψης σε υλικό και λογισμικό, αυτές οι δύο περιπτώσεις διακρίνονται ως ανασταλτικές (blocking) μεταφορές δεδομένων και μη – ανασταλτικές (non - blocking) μεταφορές δεδομένων.

Σημείωση: Λειτουργίες Αποστολής / Λήψης σε Υλικό / Λογισμικό

Μια ανασταλτική μεταφορά δεδομένων θα θέσει σε αναμονή τη ροή εκτέλεσης του λογισμικού ή του υλικού μέχρι να ολοκληρωθεί η μεταφορά δεδομένων. Για παράδειγμα, εάν το λογισμικό έχει υλοποιήσει τη μεταφορά δεδομένων χρησιμοποιώντας κλήσεις συναρτήσεων, τότε μια ανασταλτική μεταφορά θα σημαίνει ότι αυτές οι συναρτήσεις δεν επιστρέφουν μέχρι να ολοκληρωθεί η μεταφορά δεδομένων. Από την οπτική του προγραμματιστή, τα ανασταλτικά πρωταρχικά στοιχεία είναι και τα ευκολότερα για να εργαστεί κανείς. Εντούτοις, μπορούν να θέσουν σε αναμονή το υπόλοιπο πρόγραμμα.

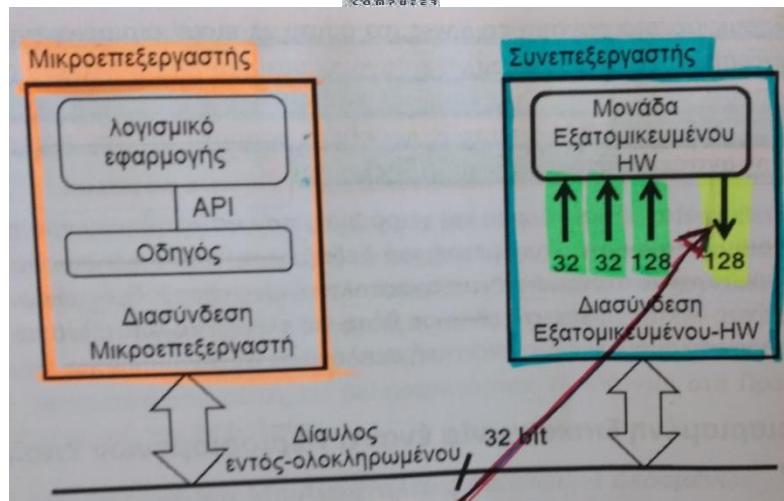
Σημείωση: Χαρακτηριστικά μη-ανασταλτικής μεταφοράς δεδομένων

Μια μη-ανασταλτική μεταφορά δεδομένων δε θα σταματήσει τη ροή εκτέλεσης του λογισμικού ή του υλικού, αλλά η μεταφορά δεδομένων μπορεί να είναι ανεπιτυχής. Επομένως, μια συνάρτηση λογισμικού που υλοποιεί μια μη-ανασταλτική μεταφορά δεδομένων θα χρειαστεί να εισαγάγει μια πρόσθετη σημαία κατάστασης που μπορεί να ελεγχθεί. Οι μη-ανασταλτικές μεταφορές δεδομένων δεν θα προκαλέσουν αναμονή, αλλά απαιτούν πρόσθετη προσοχή απ' τον προγραμματιστή για την αντιμετώπιση περιπτώσεων εξαίρεσης.

[**Περιορισμένη Επικοινωνία έναντι Περιορισμένων υπολογισμών**](#)

Γράφοντας μια υλοποίηση σε υλικό της XYZ, ο χρόνος εκτέλεσης μειώνεται σε 1ms. Επομένως, το σύστημα μπορεί να επιταχυνθεί κατά ένα συντελεστή 100.

Η συνολική εφαρμογή εξακολουθεί να εκτελείται στον αργό επεξεργαστή λογισμικού. Η εκτέλεση του XYZ σε γρήγορο υλικό δε βοηθά, αν η εφαρμογή λογισμικού δε μπορεί να χρησιμοποιήσει αποτελεσματικά τη μονάδα υλικού. Για παράδειγμα, ας πούμε ότι, λόγω μιας αναποτελεσματικής διασύνδεσης υλικού/λογισμικού, η κλήση της XYZ σε υλικό διαρκεί 20ms. Τότε, η επιτάχυνση του συστήματος είναι μόνο 5, όχι 100!



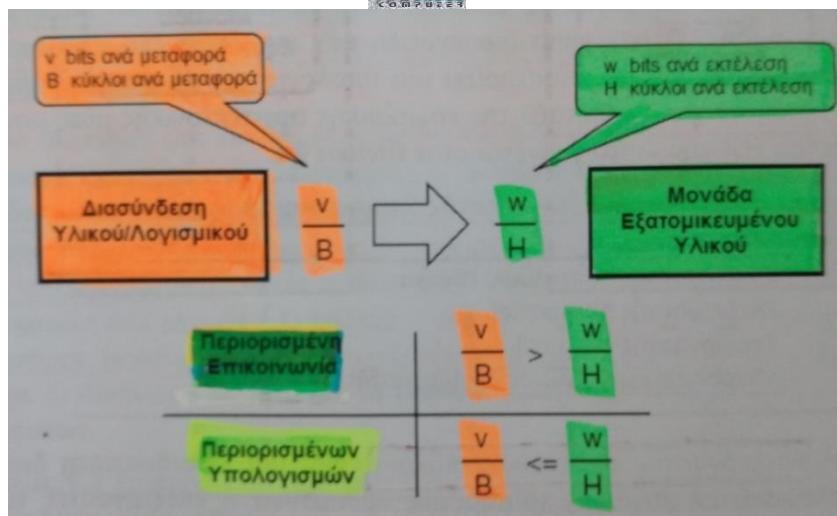
Εικόνα 87 - Περιορισμοί επικοινωνίας ενός συνεπεξεργαστή

Σημείωση: Σύστημα περιορισμένη επικοινωνίας

Σε πρακτικές περιπτώσεις, μπορούμε να αναλύσουμε τέτοια όρια απόδοσης. Ένα παράδειγμα δίνεται στην Εικόνα 87. Αυτή η μονάδα υλικού έχει τρεις θύρες εισόδου και μία θύρα εξόδου. Έτσι, κάθε φορά που καλούμε τον συνεπεξεργαστή υλικού, πρέπει να μεταφέρουμε $128 + 128 + 32 + 32 = 320$ ψηφία. Ας υποθέσουμε ότι αυτή η μονάδα εξατομικευμένου υλικού χρειάζεται πέντε κύκλους για να υπολογίσει ένα αποτέλεσμα. Συνεπώς, όταν αυτή η μονάδα είναι συνδεδεμένη στο λογισμικό και επιθυμούμε να τρέξουμε την μονάδα σε πλήρη απόδοση, θα χρειαστεί να υποστηρίξουμε ένα εύρος ζώνης δεδομένων $320/5 = 64$ ψηφία ανά κύκλο. Αυτό το εύρος ζώνης δεδομένων πρέπει να παρέχεται μέσω μιας διασύνδεσης υλικού/λογισμικού. Όπως απεικονίζεται στην Εικόνα 87, ένας δίαυλος 32 ψηφίων χρησιμοποιείται για την παροχή δεδομένων στον συνεπεξεργαστή. Δεδομένου ότι κάθε μεταφορά διαύλου απαιτεί τουλάχιστον έναν κύκλο ρολογιού, ο δίαυλος δε μπορεί να παρέχει περισσότερα από 32 ψηφία ανά κύκλο. Σαφώς, για πλήρη αξιοποίηση, ο συνεπεξεργαστής χρειάζεται ένα μεγαλύτερο εύρος ζώνης δεδομένων από αυτό που μπορεί να παρέχεται μέσω της διασύνδεσης υλικού/λογισμικού. Σε αυτήν την περίπτωση, το σύστημα είναι περιορισμένης επικοινωνίας (communication – constrained).

Σημείωση: Σύστημα περιορισμένων υπολογισμών

Τώρα, ας υποθέσουμε ότι ο συνεπεξεργαστής υλικού χρειάζεται 50 κύκλους (αντί για 5) για να ολοκληρώσει τη λειτουργία. Στην περίπτωση αυτή, η πλήρης αξιοποίηση του υλικού απαιτεί $320/50 = 6,4$ ψηφία ανά κύκλο. Ένας δίαυλος 32-bit μπορεί να είναι σε θέση να παρέχει τα δεδομένα που θα διατηρήσουν τη μονάδα υλικού πλήρως αξιοποιημένη. Επομένως, το σύστημα είναι περιορισμένων υπολογισμών (computation constrained).



Εικόνα 88 - Σύστημα περιορισμένης επικοινωνίας έναντι συστήματος περιορισμένων υπολογισμών

Σημείωση: Σύγκριση συστήματος περιορισμένης επικοινωνίας/υπολογισμών

Η Εικόνα 88 συνοψίζει αυτές τις παρατηρήσεις. Η διάκριση μεταξύ ενός συστήματος περιορισμένης – επικοινωνίας και ενός συστήματος περιορισμένων υπολογισμών είναι σημαντική, καθώς λέει στον σχεδιαστή που να τοποθετήσει την σχεδιαστική προσπάθεια. Σε ένα σύστημα περιορισμένης – επικοινωνίας, δεν έχει νόημα να υλοποιηθεί ένας ισχυρότερος συνεπεξεργαστής, δεδομένου ότι θα παραμείνει ανεπαρκώς αξιοποιημένος. Αντίστροφα, σε ένα σύστημα περιορισμένων-υπολογισμών, δεν χρειάζεται να ψάχνουμε για ταχύτερη διασύνδεση υλικού/λογισμικού. Ακόμα και αν τα ακριβή όρια απόδοσης σε ένα πρόβλημα συσχεδίασης υλικού/λογισμικού μπορεί να είναι πολύ δύσκολο να προσδιοριστούν, είναι συχνά εφικτό να γίνει ένας υπολογισμός back-of-the-envelope και να διαπιστωθεί αν ένα σύστημα είναι περιορισμένης επικοινωνίας ή περιορισμένων υπολογισμών.

Μια πρόσθετη γνώση μπορεί να αποκτηθεί από τον αριθμό των κύκλων ρολογιού που χρειάζονται ανά εκτέλεση της εξατομικευμένης μονάδας υλικού. Αυτός ο αριθμός ονομάζεται συντελεστής διαμοιρασμού υλικού (hardware sharing factor) ή HSF. Ο HSF ορίζεται ως ο αριθμός των κύκλων ρολογιού που είναι διαθέσιμοι ανάμεσα σε κάθε γεγονός εισόδου/εξόδου. Για παράδειγμα, η τιμή 10 για ένα HSF θα σήμαινε ότι μια δεδομένη αρχιτεκτονική υλικού έχει έναν προϋπολογισμό κύκλων ίσο με 10 κύκλους ρολογιού μεταξύ διαδοχικών γεγονότων εισόδου/εξόδου.

Ο HSF είναι μια ένδειξη εάν μια δεδομένη αρχιτεκτονική είναι αρκετά ισχυρή για να υποστηρίξει μια υπολογιστική απαίτηση. Πράγματι, υπάρχει ισχυρή συσχέτιση μεταξύ της εσωτερικής αρχιτεκτονικής μιας μονάδας υλικού και του HSF της. Αυτό φαίνεται στον Πίνακα 9.1

Πίνακας 9.1 Συντελεστής διαμοιρασμού υλικού

Αρχιτεκτονική	HSF
Επεξεργαστής Συστολικού Πίνακα	1
Επεξεργαστής bit-parallel	1-10
Επεξεργαστής bit-serial	10-100

Ισχυρή και χαλαρή σύζευξη

Σημείωση: Ορισμός σύζευξης

Η τρίτη γενική ιδέα στις διασυνδέσεις υλικού / λογισμικού είναι αυτή της σύζευξης (coupling). Η σύζευξη δείχνει το επίπεδο αλληλεπίδρασης μεταξύ της ροής εκτέλεσης του λογισμικού και της ροής εκτέλεσης σε εξατομικευμένο υλικό. Σε ένα σχήμα ισχυρής σύζευξης, το εξατομικευμένο υλικό και το λογισμικό συγχρονίζονται συχνά και συχνά ανταλλάσσουν δεδομένα, για παράδειγμα με ρυθμό λίγων εντολών στο λογισμικό.

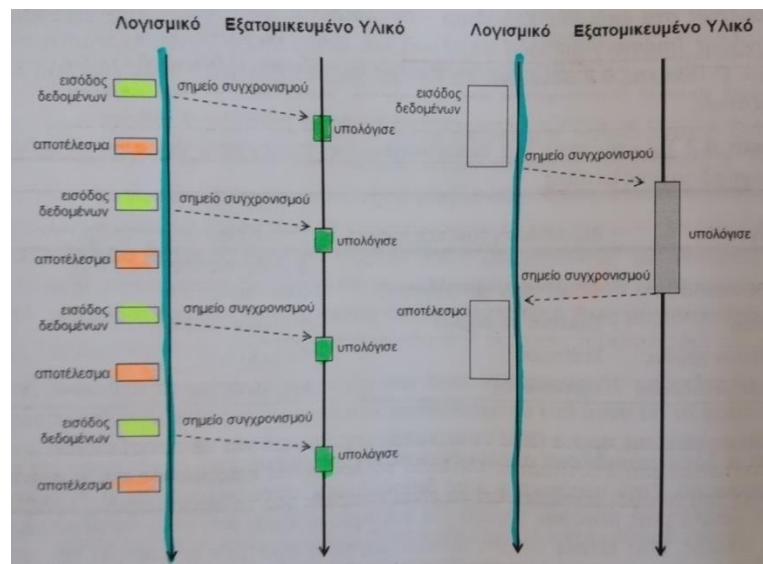
Σημείωση: Κατηγορίες σύζευξης

Σε ένα σχήμα χαλαρής σύζευξης, το υλικό και το λογισμικό συγχρονίζονται περιστασιακά, για παράδειγμα με συχνότητα μια συνάρτηση ή μια διεργασία στο λογισμικό. Έτσι, η σύζευξη συσχετίζει τις ιδέες του συγχρονισμού με την απόδοση.

Σημείωση: Που χρησιμοποιείται ισχυρή διασύνδεση (σύζευξη)

Μια δεδομένη εφαρμογή μπορεί να χρησιμοποιεί είτε ισχυρή σύζευξη είτε χαλαρή σύζευξη. Η Εικόνα 89 δείχνει πως η επιλογή για χαλαρή – σύζευξη ή για ισχυρή σύζευξη μπορεί να επηρεάσει τους λανθάνοντες χρόνους της εφαρμογής. Η αριστερή πλευρά της εικόνας απεικονίζει ένα σχήμα ισχυρής σύζευξης. Το λογισμικό θα στείλει τέσσερα ξεχωριστά στοιχεία δεδομένων στο εξατομικευμένο υλικό, συλλέγοντας κάθε φορά το αποτέλεσμα.

Αυτό είναι το σχήμα που θα μπορούσε να χρησιμοποιηθεί από μια διασύνδεση συνεπεξεργαστή. Το σημείο συγχρονισμού αντιστοιχεί στην εκτέλεση μιας εντολής συνεπεξεργαστή στο λογισμικό.



Εικόνα 89 - Ισχυρή σύζευξη έναντι χαλαρής σύζευξης

Σημείωση: Που χρησιμοποιείται η χαλαρή διασύνδεση (σύζευξη)

Η δεξιά πλευρά της εικόνας απεικονίζει ένα χαλαρά συζευγμένο σχήμα. Σε αυτήν την περίπτωση, το λογισμικό παρέχει ένα μεγάλο μπλοκ δεδομένων στο εξατομικευμένο υλικό, συγχρονίζεται με το υλικό και στη



συνέχεια περιμένει το εξατομικευμένο υλικό να ολοκληρώσει την επεξεργασία και να επιστρέψει το αποτέλεσμα. Αυτό το σχήμα θα χρησιμοποιηθεί από μια διασύνδεση χαρτογραφημένης μνήμης, για παράδειγμα χρησιμοποιώντας μια κοινόχρηστη μνήμη.

Σημείωση: Σύγκριση ισχυρής και χαλαρής σύζευξης

Τα σχήματα χαλαρής – σύζευξης τείνουν να παραδίδουν ελαφρώς πιο πολύπλοκους σχεδιασμούς υλικού επειδή το υλικό πρέπει να ασχοληθεί εκτενέστερα με την μετακίνηση δεδομένων μεταξύ υλικού και λογισμικού. Από την άλλη, τα σχήματα ισχυρής – σύζευξης στηρίζονται περισσότερο στο λογισμικό για τη διαχείριση της συνολικής εκτέλεσης του συστήματος. Η επίτευξη ενός υψηλού βαθμού παραλληλίας στο συνολικό σχεδιασμό μπορεί να επιτευχθεί ευκολότερα με ένα χαλαρά συζευγμένο σχήμα απ' ότι με ένα ισχυρά συζευγμένο σχήμα.

Προβλήματα

Πρόβλημα 9.1 Βρείτε τη μέγιστη ταχύτητα επικοινωνίας από την CPU1 στην CPU2 στην αρχιτεκτονική του συστήματος που φαίνεται στην Εικόνα 90. Υποθέστε ότι οι CPU έχουν στην διάθεσή τους ένα αφοσιωμένο κανάλι συγχρονισμού έτσι ώστε να μπορούν να επιλέξουν τη βέλτιστη στιγμή για να εκτελέσουν μια συναλλαγή ανάγνωσης ή εγγραφής. Χρησιμοποιήστε τις ακόλουθες σταθερές σχεδιασμού.

Κάθε συναλλαγή διαύλου στον δίαυλο υψηλής ταχύτητας διαρκεί 50 ns.

Κάθε συναλλαγή διαύλου στον δίαυλο χαμηλής ταχύτητας διαρκεί 200 ns.

Κάθε προσπέλαση μνήμης (ανάγνωση ή εγγραφή) διαρκεί 80 ns.

Κάθε μεταφορά γέφυρας διαρκεί 100 ns.

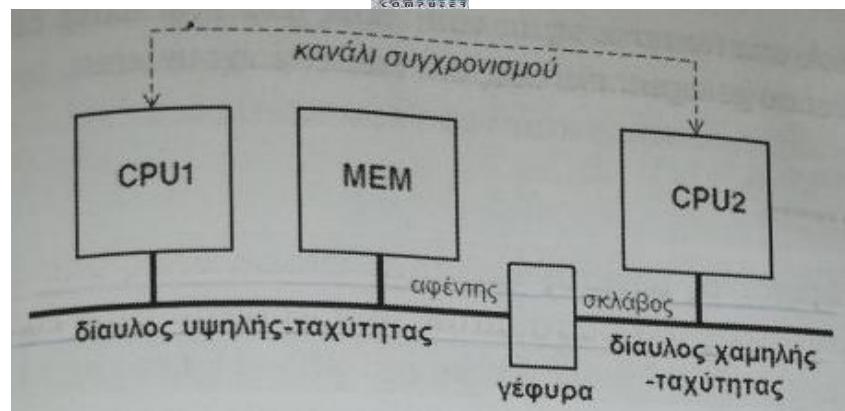
Οι CPU είναι πολύ ταχύτερες από το δίαυλο συστήματος και μπορούν να διαβάζουν/γράφουν δεδομένα στο δίαυλο σε οποιοδήποτε επιλεγμένο ρυθμό δεδομένων.

Πρόβλημα 9.2 Εξετάστε την διμερή χειραψία στην Εικόνα 91. Ένας πομπός συγχρονίζεται με ένα δέκτη και μεταδίδει μια ακολουθία συμβόλων δεδομένων μέσω ενός καταχωρητή δεδομένων.

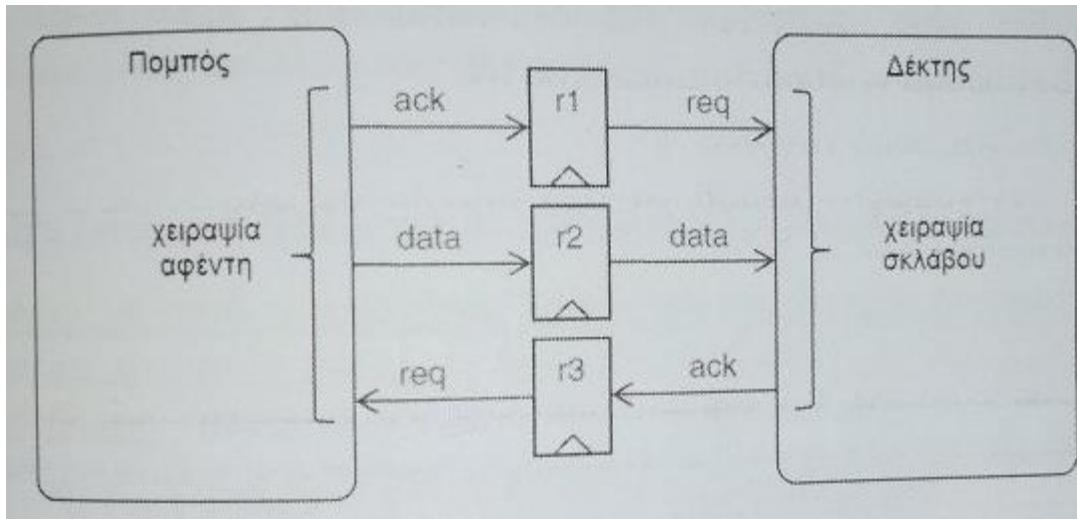
(α) Περιγράψτε κάτω από ποιες συνθήκες ο καταχωρητής r1 μπορεί να αφαιρεθεί χωρίς να ζημιωθεί η ακεραιότητα της επικοινωνίας. Υποθέστε ότι, αφού απομακρυνθεί ο r1 η είσοδος req του δέκτη συνδέεται με λογικό-1.

(β) Περιγράψτε κάτω από ποιες συνθήκες ο καταχωρητής r3 μπορεί να αφαιρεθεί χωρίς να ζημιωθεί η ακεραιότητα της επικοινωνίας. Υποθέστε ότι, αφού απομακρυνθεί ο r3 η είσοδος req του πομπού συνδέεται με λογικό-1.

(γ) Υποθέστε ότι πρόκειται να αντικαταστήσετε τον καταχωρητή r1 με δύο καταχωρητές στη σειρά, έτσι ώστε ολόκληρη η μετάβαση από το πομπό-ack στο δέκτη-req να διαρκεί πλέον δύο κύκλους ρολογιού αντί για έναν. Περιγράψτε την επίπτωση αυτής της αλλαγής στο ρυθμό μετάδοσης της επικοινωνίας και περιγράψτε την επίπτωση αυτής της αλλαγής στο λανθάνοντα χρόνο της επικοινωνίας.



Εικόνα 90 - Τοπολογία συστήματος για το Πρόβλημα 9.1



Εικόνα 91 - Διπλή χειραψία για Προβλήματα 9.1

Πρόβλημα 9.3 Μια συνάρτηση C έχει δέκα εισόδους και δέκα εξόδους, όλες ακέραιοι. Η συνάρτηση απαιτεί 1.000 κύκλους για εκτέλεση στο λογισμικό. Πρέπει να αξιολογήσετε αν είναι λογικό να κατασκευάσετε έναν συνεπεξεργαστή για αυτήν τη συνάρτηση. Υποθέστε ότι η συνάρτηση απαιτεί K κύκλους για εκτέλεση σε υλικό και ότι χρειάζεστε Q κύκλους για να μεταφέρετε μια λέξη μεταξύ του λογισμικού και του συνεπεξεργαστή μέσω ενός διαύλου συστήματος. Σχεδιάστε ένα γράφημα που απεικονίζει το Q συναρτήσει του K, και υποδείξτε ποιες περιοχές σε αυτό το γράφημα δικαιολογούν έναν συνεπεξεργαστή.



10 Κεφάλαιο - Δίαυλοι εντός – Ολοκληρωμένου

Συστήματα Διαύλου εντός – Ολοκληρωμένου

10.1.1 Λίγα υπαρκτά Πρότυπα Διαύλων εντός – Ολοκληρωμένου

Οι συζητήσεις σε αυτό το κεφάλαιο βασίζονται σε τέσσερα διαφορετικά **συστήματα διαύλων**: το δίαυλο AMBA, το δίαυλο CoreConnect, το δίαυλο Avalon και το δίαυλο Wishbone.

Σημείωση: Πρότυπα (Συστήματα) Διαύλων

- **AMBA** (Advanced Microcontroller Bus Architecture) είναι το σύστημα διαύλου που χρησιμοποιείται από τους επεξεργαστές ARM.
- **CoreConnect** είναι ένα σύστημα διαύλου που προτάθηκε από την IBM για την σειρά επεξεργαστών PowerPC. Παρόμοια με το δίαυλο AMBA, ο δίαυλος CoreConnect έρχεται σε διάφορες παραλλαγές.
- **Avalon** είναι ένα σύστημα διαύλου που αναπτύχθηκε από την Altera για χρήση σε εφαρμογές SoC για τον Nios επεξεργαστή της. Το Avalon ορίζεται με βάση τους διαφορετικούς τύπους διασυνδέσεων που παρέχει σε εξαρτήματα SoC.
- **Wishbone** είναι ένα σύστημα διαύλου ανοικτού κώδικα προτεινόμενο από την SiliCore Corporation. Είναι το απλούστερο από τα προηγούμενα πρότυπα.

Σημείωση: Διαμορφώσεις (είδη) διαύλου

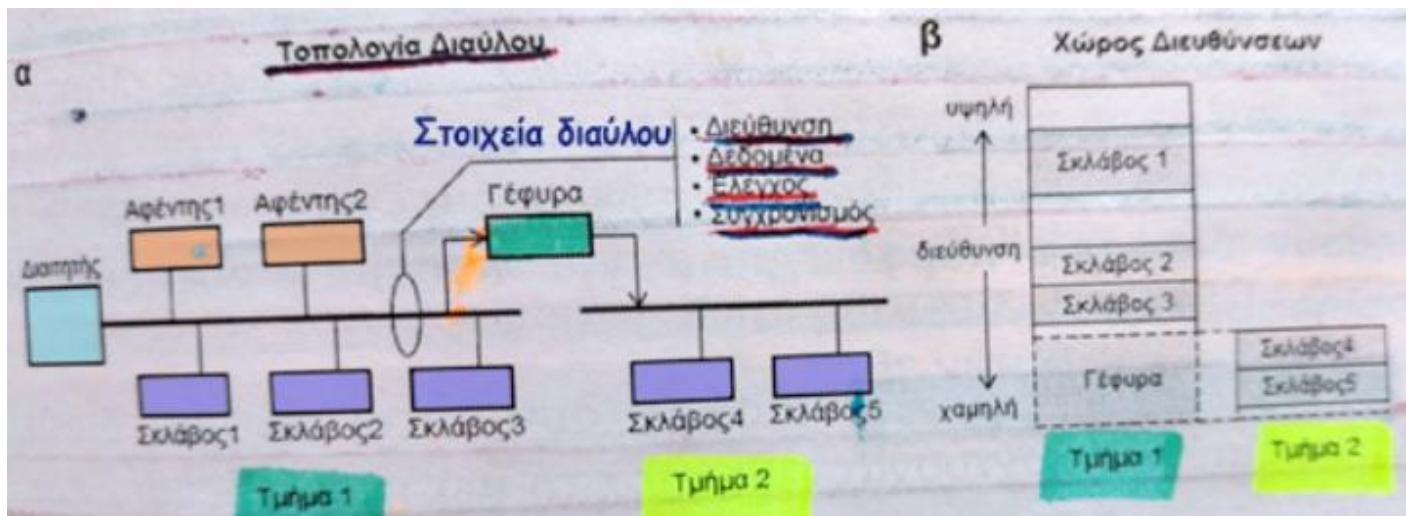
Διακρίνουμε δύο διαμορφώσεις διαύλου: τον κοινόχρηστο δίαυλο και τον δίαυλο σημείου – σε – σημείο (point – to – point). Η διαφορά μεταξύ αυτών των δύο είναι απλά ο αριθμός των εξαρτημάτων (συσκευών) που μοιράζονται το δίαυλο. Οι κοινόχρηστοι δίαυλοι είναι οι πιο συνηθισμένοι και αρκετά πρότυπα (AMBA, CoreConnect) ορίζουν πολλαπλές εκδόσεις τους, διαφορετικές ως προς την πολυπλοκότητα και την απόδοση.

10.1.2 Στοιχεία σε Κοινόχρηστο Δίαυλο

Σημείωση: Γέφυρες διαύλου

Ένα σύστημα διαύλου εντός ολοκληρωμένου υλοποιεί ένα **πρωτόκολλο διαύλου**: μια ακολουθία βημάτων για τη μεταφορά των δεδομένων κατά τρόπο ομαλό. Ένα τυπικό σύστημα διαύλου εντός – ολοκληρωμένου θα αποτελείται από ένα ή περισσότερα τμήματα διαύλου, όπως φαίνεται στην Εικόνα 92. Κάθε τμήμα διαύλου ομαδοποιεί έναν ή περισσότερους αφέντες διαύλου με σκλάβους διαύλου. Οι γέφυρες διαύλου είναι κατευθυνόμενα εξαρτήματα που συνδέουν τμήματα διαύλου. **Μια γέφυρα διαύλου λειτουργεί ως σκλάβος στην είσοδο και ως αφέντης στην έξοδο.** Σε οποιαδήποτε συγκεκριμένη στιγμή, ένα τμήμα διαύλου ελέγχεται είτε από έναν αφέντη διαύλου, είτε από έναν διαιτητή διαύλου. Ο ρόλος του διαιτητή διαύλου (bus arbiter) είναι να αποφασίσει ποιος αφέντης διαύλου επιτρέπεται να ελέγχει το δίαυλο. Η διαιτησία γίνεται για κάθε συναλλαγή διαύλου. Θα πρέπει όμως να γίνεται με δίκαιο τρόπο, ώστε κανένας αφέντης διαύλου να μην είναι μόνιμα αποκλεισμένος από

την πρόσβαση στο δίαυλο. Οι σκλάβοι διαύλου δεν μπορούν ποτέ να αποκτήσουν έλεγχο σε ένα τμήμα διαύλου, αλλά αντίθετα πρέπει να ακολουθήσουν τις οδηγίες του αφέντη που είναι κάτοχος του διαύλου.



Σημείωση: Χώρος διευθύνσεων συστήματος διαύλου

Ένα σύστημα διαύλου χρησιμοποιεί ένα χώρο διευθύνσεων (address space) για να οργανώσει την επικοινωνία μεταξύ των εξαρτημάτων. Ένας ενδεικτικός χώρος διευθύνσεων εμφανίζεται στα δεξιά της Εικόνας 92. Συνήθως, η μικρότερη διευθυνσιοδοτούμενη οντότητα σε ένα χώρο διευθύνσεων είναι 1 byte. **Κάθε μεταφορά δεδομένων πάνω από το δίαυλο συσχετίζεται με μια δεδομένη διεύθυνση προορισμού.** Η διεύθυνση προορισμού καθορίζει ποιο εξάρτημα θα πρέπει να πάρει τα δεδομένα. Οι γέφυρες διαύλων είναι διαφανείς ως προς την διεύθυνση: θα συγχωνεύουν τους χώρους διευθύνσεων των σκλάβων από την έξοδο τους και θα τους μετατρέψουν σε ένα ενιαίο χώρο διευθύνσεων σκλάβου στην είσοδο τους.

Σημείωση: Κατηγορίες καλωδίων διαύλου εντός ολοκληρωμένου

Ένας δίαυλος εντός – ολοκληρωμένου συντίθεται φυσικά από μια δέσμη καλωδίων, η οποία περιλαμβάνει τις ακόλουθες τέσσερις κατηγορίες: καλώδια διεύθυνσης, καλώδια δεδομένων, καλώδια εντολών και καλώδια συγχρονισμού.

- Τα **καλώδια δεδομένων** μεταφέρουν στοιχεία δεδομένων μεταξύ των εξαρτημάτων. Η καλωδίωση εντός – ολοκληρωμένου είναι πολύ πυκνή και τα καλώδια δεδομένων δεν χρειάζεται να πολυπλέκονται. Οι αφέντες, οι σκλάβοι και οι γέφυρες θα έχουν ξεχωριστές εισόδους δεδομένων και εξόδους δεδομένων.
- Τα **καλώδια διευθύνσεων** μεταφέρουν τη διεύθυνση που αντιστοιχεί σε ένα συγκεκριμένο στοιχείο δεδομένων. Η διαδικασία αναγνώρισης της διεύθυνσης προορισμού ονομάζεται αποκωδικοποίηση διεύθυνσης (address decoding). Μια προσέγγιση για την υλοποίηση της αποκωδικοποίησης διεύθυνσης είναι η υλοποίησή της μέσα στο σκλάβο διαύλου.
- Τα **καλώδια εντολών** περιγράφουν τη φύση της μεταφοράς που πρέπει να πραγματοποιηθεί. Οι απλές εντολές περιλαμβάνουν την εντολή διάβασε και την εντολή γράψε, αλλά τα μεγαλύτερα συστήματα διαύλων εντός-

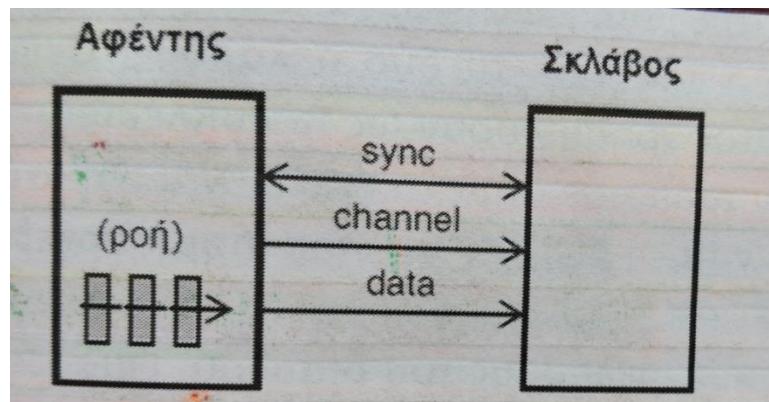
ολοκληρωμένου μπορεί να περιέχουν μια μεγάλη ποικιλία εντολών, που ικανοποιούν τις απαιτήσεις μιας δεδομένης εντολής ανάγνωσης ή εγγραφής.

- Τα καλώδια συγχρονισμού εξασφαλίζουν ότι οι αφέντες διαύλων και οι σκλάβοι διαύλων συγχρονίζονται κατά τη μεταφορά δεδομένων. Τα συνήθη συστήματα διαύλων εντός - ολοκληρωμένου σήμερα είναι **σύγχρονα**. Χρησιμοποιούν ένα μόνο σήμα ρολογιού ανά τμήμα διαύλου: όλα τα καλώδια δεδομένων, διεύθυνσης και εντολών αναφέρονται στις ακμές του ρολογιού του διαύλου. Εκτός από το σήμα ρολογιού, για το συγχρονισμό ενός αφέντη διαύλου και ενός σκλάβου, μπορούν να χρησιμοποιηθούν και επιπρόσθετα σήματα ελέγχου, για να υποδείξουν για παράδειγμα όρια χρόνου και να υποστηρίξουν τη σηματοδοσία αιτήματος – επιβεβαίωσης.

10.1.3 Στοιχεία σε Δίαυλο Σημείου – σε – Σημείο

Σημείωση: Δίαυλος σημείου – προς - σημείο

Για αφοσιωμένες συνδέσεις, τα συστήματα διαύλου μπορούν επίσης να υποστηρίζουν κάποια λειτουργία επικοινωνίας σημείου – σε – σημείο. Η Εικόνα 93 απεικονίζει μια τέτοια σύνδεση σημείου – σε – σημείο. Όπως ένας κοινόχρηστος δίαυλος, ένας δίαυλος σημείου – σε – σημείο αναγνωρίζει επίσης έναν αφέντη και ένα σκλάβο. **Δεν υφίσταται η έννοια του χώρου διευθύνσεων.** Αντ' αυτού, τα δεδομένα θεωρούνται ως μια άπειρη ροή στοιχείων. *Τα στοιχεία δεδομένων ενδέχεται να αντιστοιχίζονται σε λογικά κανάλια. Αυτό επιτρέπει πολλαπλές ροές να πολυπλέκονται πάνω από το ίδιο φυσικό κανάλι.* Κάθε μεταφορά δεδομένων θα αποτελείται τότε από μια πλειάδα (τιμή – δεδομένου, κανάλι). Ο δίαυλος σημείου – σε – σημείο έχει **καλώδια συγχρονισμού** για να διαχειριστεί την αλληλεπίδραση μεταξύ αφέντη και σκλάβου.



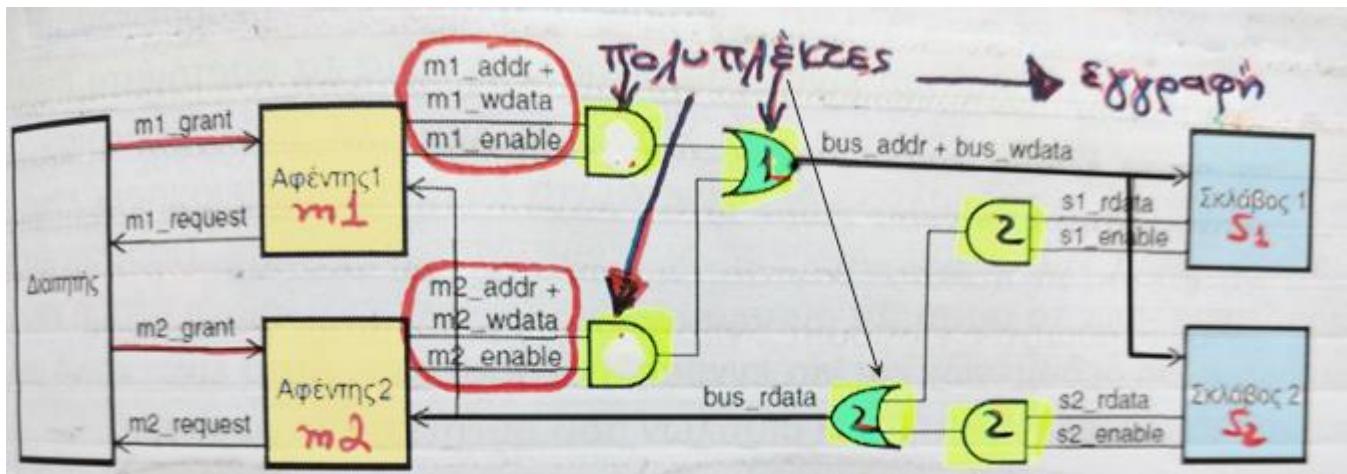
Εικόνα 93 - Δίαυλος σημείου-σε-σημείο

10.1.4 Φυσική Υλοποίηση Διαύλων εντός – Ολοκληρωμένου

Σημείωση: Συστήματα διαύλου εντός ολοκληρωμένου

Η Εικόνα 94 δείχνει τη φυσική διάταξη ενός τυπικού τμήματος διαύλου εντός – ολοκληρωμένου με δύο αφέντες και δύο σκλάβους. Οι πύλες AND και OR στο κέντρο του διαγράμματος χρησιμεύουν ως πολυπλέκτες. Αρκετά σήματα συγχωνεύονται με αυτόν τον τρόπο σε σήματα διευθύνσεων και δεδομένων, με εύρος-διαύλου. Για παράδειγμα, η διεύθυνση που δημιουργείται από τους αφέντες διαύλου συγχωνεύεται σε μία μόνο διεύθυνση διαύλου και αυτή η διεύθυνση διαύλου διανέμεται στους σκλάβους του δίαυλου. Ομοίως, η μεταφορά δεδομένων

από τους αφέντες στους σκλάβους συγχωνεύεται σε ένα σήμα εγγραφής δεδομένων με εύρος – διαύλου και η μεταφορά δεδομένων από τους σκλάβους στους αφέντες συγχωνεύεται σε ένα σήμα ανάγνωσης δεδομένων με εύρος – διαύλου. Αυτή η υλοποίηση είναι χαρακτηριστική για τα συστήματα διαύλου εντός ολοκληρωμένου. Εξαιτίας του χαμηλού κόστους ενός καλωδίου εντός – ολοκληρωμένου, η πολυπλεξία δε γίνεται για λόγους εξοικονόμησης επιφάνειας πυριτίου, αλλά για λόγους λογικής συγχώνευσης σημάτων.



Εικόνα 94 - Φυσική Διασύνδεση διαύλου. Τα σήματα *_addr, *_wdata, *_rdata είναι διανύσματα σημάτων. Τα σήματα *_enable, *_grant, *_request είναι σήματα ενός-bit

Σημείωση: Σύμβαση για τα δεδομένα

Δεδομένου ότι ένα τμήμα διαύλου μπορεί να ομαδοποιήσει ένα δυνητικά μεγάλο αριθμό σημάτων, απαιτείται η χρήση μιας σύμβασης ονομασίας. Μια τέτοια σύμβαση είναι χρήσιμη για την ανάγνωση ενός διαγράμματος χρονισμού. Η σύμβαση που συσχετίζει την κατεύθυνση των δεδομένων με την ανάγνωση και την εγγραφή των δεδομένων έχει ως εξής. Η **εγγραφή δεδομένων** σημαίνει: την αποστολή τους από έναν αφέντη σε ένα σκλάβο. Η **ανάγνωση δεδομένων** σημαίνει: την αποστολή τους, από ένα σκλάβο σε έναν αφέντη. Αυτή η σύμβαση επηρεάζει την κατεύθυνση εισόδου/εξόδου των σημάτων διαύλου σε εξαρτήματα σκλάβων και εξαρτήματα αφέντη.

Σημείωση: Σημασία διαιτησίας διαύλου

Στην Εικόνα 94 κάθε αφέντης παράγει το δικό του σήμα ενεργοποίησης διαύλου προκειμένου να οδηγήσει ένα στοιχείο δεδομένων ή μια διεύθυνση στο δίαυλο. Για παράδειγμα, όταν ο Master1 θα γράψει δεδομένα, το m1_enable θα είναι σε υψηλή στάθμη, ενώ το m2_enable θα είναι σε χαμηλή. Εάν και τα δύο σήματα επίτρεψης θα είναι υψηλά, η παραγόμενη διεύθυνση διαύλου και το σήμα εγγραφής δεδομένων θα είναι απροσδιόριστα. Έτσι, το πρωτόκολλο διαύλου θα λειτουργεί μόνο όταν τα εξαρτήματα συνεργάζονται και ακολουθούν τους κανόνες του πρωτοκόλλου. Ο διαιτητής διαύλου πρέπει αν διασφαλίσει ότι **μόνο** ένας αφέντης διαύλου κάθε φορά παίρνει τον έλεγχο του διαύλου. Η διαιτησία διαύλου μπορεί αν γίνει σε καθολικό επίπεδο, για ένα ολόκληρο τμήμα διαύλου ή μπορεί να γίνει ανά εξάρτημα σκλάβου.

10.1.5 Σύμβαση Ονομασίας Διαύλου

Σημείωση: Σύμβαση ονομασίας διαύλων

Δεδομένου ότι ένα τμήμα διαύλου μπορεί να ομαδοποιήσει ένα δυνητικά μεγάλο αριθμό σημάτων, τα συστήματα διαύλου θα ακολουθήσουν μια σύμβαση ονομασίας (naming convention).

Σημείωση: Σημασία σύμβασης ονομασίας διαύλων

Για παράδειγμα, μια σύμβαση ονομασίας είναι πολύ χρήσιμη για να διαβάσουμε ένα διάγραμμα χρονισμού. Μια σύμβαση ονομασίας μπορεί επίσης να βοηθήσει τους μηχανικούς να απεικονίσουν τη συνδεσιμότητα σε ένα (κειμενικό) κατάλογο-κόμβων (netlist) ενός κυκλώματος.

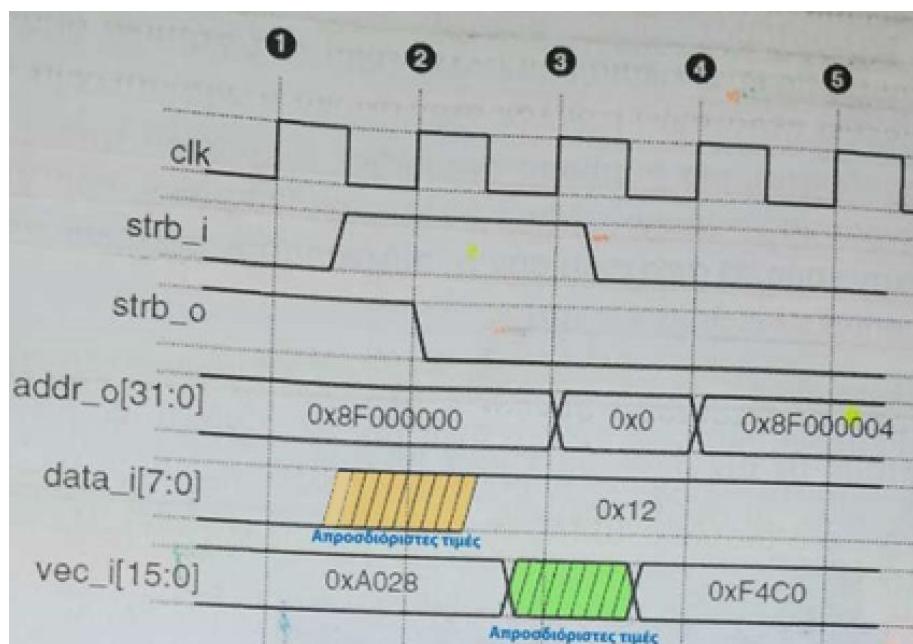
Ένα **όνομα ακροδέκτη** εξαρτήματος θα προβάλει τη λειτουργικότητα αυτού του ακροδέκτη. Για παράδειγμα, ο δίαυλος IBM/Coreconnect χρησιμοποιεί το ABUS για το δίαυλο διεύθυνσης, ο AMBA χρησιμοποιεί τα PADDR ή HADDR, ο Avalon χρησιμοποιεί το address και ο Wishbone χρησιμοποιεί το AD.

Σημείωση: Συμβάσεις διαύλων

Τα σήματα διαύλου δημιουργούνται διασυνδέοντας ακροδέκτες εξαρτημάτων. Τα σήματα διαύλου ακολουθούν επίσης μια σύμβαση. Το βασικό ζήτημα είναι να αποφευχθεί η σύγχυση μεταξύ παρόμοιων σημάτων. Για παράδειγμα, στην Εικόνα 94, υπάρχουν δύο εξαρτήματα αφέντη, το καθένα με ένα σήμα wdata. Για να ξεχωρίζει κανείς αυτά τα σήματα, το όνομα του εξαρτήματος περιλαμβάνεται ως πρόθεμα στο όνομα σήματος του διαύλου, όπως το m2_wdata. Άλλα συστήματα διαύλου χρησιμοποιούν επίσης αυτή την τεχνική. Σε ορισμένες περιπτώσεις, τόσο το όνομα του στιγμιότυπου αφέντη όσο και το όνομα του στιγμιότυπου σκλάβου θα συμπεριληφθούν ως μέρος της ονομασίας του σήματος διαύλου.

Σημείωση: Πρόσθετες Συμβάσεις διαύλων

Τα συστήματα διαύλων μπορούν να χρησιμοποιήσουν πρόσθετες συμβάσεις ονομασίας για να βοηθήσουν έναν σχεδιαστή. Για παράδειγμα, ο AMBA βάζει ως πρόθεμα σε όλα τα σήματα στο δίαυλο υψηλής ταχύτητας (AHB) το γράμμα H και σε όλα τα σήματα στον περιφερειακό δίαυλο (APB) το γράμμα P. Ο Wishbone επισυνάπτει ως κατάληξη σε όλους τους ακροδέκτες εισόδου το γράμμα I και σε όλους τους ακροδέκτες εξόδου το γράμμα O.



Εικόνα 95 - Συμβολισμός διαγράμματος χρονισμού διαύλου



10.1.6 Διάγραμμα Χρονισμού Διαύλου

Σημείωση: Περιγραφή διαγράμματος Εικόνας 95

Επειδή ένα σύστημα διαύλου αντανακλά μια πολύπλοκη, εξαιρετικά παράλληλη οντότητα, τα διαγράμματα χρονισμού χρησιμοποιούνται εκτεταμένα για να περιγράψουν τις **σχέσεις χρονισμού** ενός σήματος με το άλλο. Η Εικόνα 95 απεικονίζει ένα διάγραμμα χρονισμού των δραστηριοτήτων σε έναν γενικό δίαυλο για πέντε κύκλους ρολογιού. Το σήμα ρολογιού εμφανίζεται στην κορυφή και όλα τα σήματα αναφέρονται στην ακμή ανόδου του ρολογιού. Οι διακεκομμένες κάθετες γραμμές υποδεικνύουν την αναφορά χρονισμού.

Οι δίαυλοι σημάτων πολλών καλωδίων μπορούν να συρρικνωθούν σε ένα μόνο ίχνος στο διάγραμμα χρονισμού. Παραδείγματα στην Εικόνα 95 είναι τα addr_o, data_i και vec_i. Η ετικέτα δείχνει πότε αλλάζει τιμή ο δίαυλος. Για παράδειγμα, το addr_o αλλάζει από 0x8F000000 σε 0x00000000 στην τρίτη ακμή ρολογιού και αλλάζει ξανά σε 0x8F000004 έναν κύκλο ρολογιού αργότερα. Υπάρχουν διάφορα σχήματα για να υποδηλώσουμε ότι ένα σήμα ή ένας δίαυλος έχει άγνωστη τιμή ή τιμή αδιάφορου όρου. Η τιμή του data_i, στη δεύτερη ακμή ρολογιού και η τιμή του vec_i στην τρίτη ακμή ρολογιού είναι άγνωστες.

Σημείωση: Διάκριση ακμών και κύκλων ρολογιού

Όταν συζητάμε διαγράμματα χρονισμού, πρέπει να κάνουμε μια διάκριση ανάμεσα στις **ακμές ρολογιού** και στους **κύκλους ρολογιού**. Η διαφορά μεταξύ τους είναι λεπτή, αλλά συχνά προκαλεί σύγχυση. Ο όρος κύκλος ρολογιού (clock cycle) είναι διφορούμενος, διότι δεν υποδεικνύει ένα μοναδικό σημείο στο χρόνο: ένας κύκλος ρολογιού έχει μια αρχή και ένα τέλος.

Σημείωση: Διαφορά ακμής ρολογιού – κ.ρ.

Μια ακμή ρολογιού (clock edge), από την άλλη, είναι ένα ατομικό συμβάν που δεν μπορεί να διαιρεθεί περαιτέρω (τουλάχιστον όχι για το παράδειγμα του σύγχρονου μονού-ρολογιού). Μια συνέπεια του διφορούμενου όρου κύκλος ρολογιού είναι ότι η έννοια του όρου αλλάζει με την κατεύθυνση των σημάτων.

Σημείωση: Σήματα εισόδου

Όταν συζητάμε για την τιμή των σημάτων εισόδου, οι σχεδιαστές συνήθως θέλουν να πουν ότι αυτά τα σήματα πρέπει να είναι σταθερά κατά την έναρξη του κύκλου ρολογιού, ακριβώς **πριν** από την ακμή του ρολογιού.

Σημείωση: Σήματα εξόδου

Όταν συζητάμε για την τιμή των σημάτων εξόδου, οι σχεδιαστές συνήθως μιλούν για σήματα που είναι σταθερά στο τέλος του κύκλου ρολογιού, επομένως **μετά** από μια ακμή ρολογιού.

Σημείωση: Περιγραφή Εικόνας 95

Θεωρούμε για παράδειγμα το σήμα strb_o στην Εικόνα 95. Το σήμα κατέρχεται αμέσως μετά την ακμή ρολογιού με ετικέτα 2. Δεδομένου ότι το strb_o είναι ένα σήμα εξόδου, ένας σχεδιαστής θα έλεγε ότι το σήμα είναι χαμηλό στον κύκλο ρολογιού 2: η έξοδος, θα πρέπει να φτάσει σε μια σταθερή τιμή **μετά** την ακμή ρολογιού 2. Αντίθετα, εξετάζουμε το σήμα strb_i. Αυτό το σήμα εισόδου είναι υψηλό στην ακμή ρολογιού με ετικέτα 2. Επομένως, ένας σχεδιαστής θα έλεγε ότι αυτή η είσοδος είναι υψηλή στον κύκλο ρολογιού 2. Αυτό σημαίνει ότι

το σήμα θα πρέπει να φτάσει σε μια σταθερή **τιμή** πριν από την ακμή ρολογιού 2. Για να αποφύγουμε αυτήν την ασάφεια, θα συζητήσουμε τα διαγράμματα χρονισμού σε όρους ακμών και όχι κύκλων ρολογιού.

10.1.7 Ορισμός του Γενικού Διαύλου

Τα σήματα που απαρτίζουν τον γενικό δίαυλο αναφέρονται στον Πίνακα 10.1

Μεταφορές Διαύλων

Πίνακας 10.1 Σήματα στο γενικό δίαυλο

Όνομα Σήματος	Σημασία
clk	Σήμα ρολογιού. Όλα τα άλλα σήματα διαύλου είναι αναφορές στην ανερχόμενη ακμή ρολογιού
m_addr	Δίαυλος διευθύνσεων αφέντη (master address)
m_data	Δίαυλος δεδομένων από αφέντη σε σκλάβο (λειτουργία εγγραφής) – master address
s_data	Δίαυλος δεδομένων από σκλάβο σε αφέντη (λειτουργία ανάγνωσης) – slave address
m_rnw	Ανάγνωση–Όχι–Εγγραφή. Γραμμή ελέγχου για τη διάκριση μεταξύ λειτουργιών ανάγνωσης και εγγραφής
m_sel	Σήμα επιλογής αφέντη, δηλώνει ότι αυτός ο αφέντης αναλαμβάνει τον έλεγχο του διαύλου
s_ack	Σήμα επιβεβαίωσης σκλάβου, δηλώνει την ολοκλήρωση της μεταφοράς
m_addr_valid	Χρησιμοποιείται στη θέση του m_sel σε χωριστές-μεταφορές (split-transfers)
s_addr_ack	Χρησιμοποιείται για τη διεύθυνση στη θέση του s_ack σε χωριστές-μεταφορές
s_wr_ack	Χρησιμοποιείται για τη write-data στη θέση του s_ack σε χωριστές-μεταφορές
s_rd_ack	Χρησιμοποιείται για τη read-data στη θέση του s_ack σε χωριστές-μεταφορές
m_burst	Υποδεικνύει τον τύπο ριπής (burst) της τρέχουσας μεταφοράς
m_lock	Υποδεικνύει ότι ο δίαυλος είναι κλειδωμένος για την τρέχουσα μεταφορά
m_req	Ζητά προσπέλαση στο δίαυλο από τον διαιτητή διαύλου – master request
m_grant	Υποδεικνύει ότι η πρόσβαση στο δίαυλο έχει παραχωρηθεί

10.1.8 Απλές Μεταφορές Ανάγνωσης και Εγγραφής

Σημασία: Περιγραφή Εικόνας 96

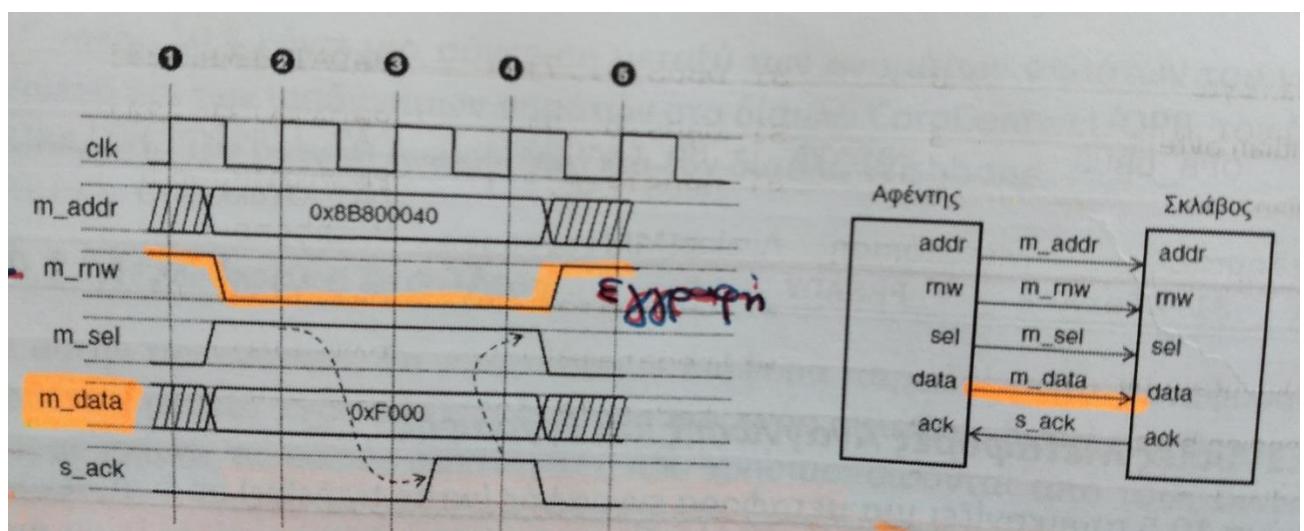
Η Εικόνα 96 απεικονίζει μια μεταφορά εγγραφής (write transfer) σε ένα γενικό περιφερειακό δίαυλο. Ένας αφέντης διαύλου θα γράψει την τιμή 0xF000 στη διεύθυνση 0x8B800040. Υποθέτουμε ότι αυτός ο δίαυλος έχει μόνο ένα αφέντη διαύλου και ότι δεν χρειάζεται διαιτησία. Στην ακμή ρολογιού 2, ο αφέντης παίρνει τον έλεγχο του διαύλου, οδηγώντας την γραμμή επιλογής αφέντη **m_sel** σε υψηλή στάθμη. Αυτό υποδεικνύει στον σκλάβο διαύλου ότι ξεκίνησε μια συναλλαγή διαύλου. Περαιτέρω λεπτομέρειες σχετικά με τη φύση της συναλλαγής διαύλου φαίνονται στην κατάσταση της διεύθυνσης διαύλου **m_addr** και του σήματος ελέγχου ανάγνωσης/εγγραφής διαύλου **m_rnw**. Σε αυτή την περίπτωση, η μεταφορά είναι μια εγγραφή, έτσι το σήμα ανάγνωσης-όχι-εγγραφής (read-not-write δηλ. το **m_rnw**) πηγαίνει σε χαμηλή στάθμη.

Σημασία: Πλεονέκτημα-Μειονέκτημα των wait-states

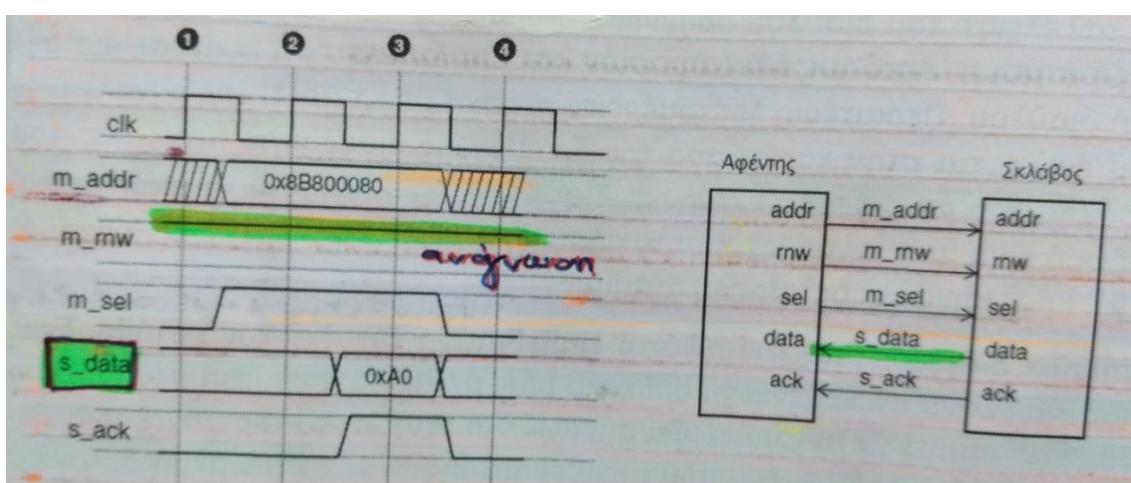
Τα αιτήματα διαύλου από τον αφέντη επιβεβαιώνονται από τον σκλάβο. Ένας σκλάβος μπορεί να παρατείνει τη διάρκεια μιας μεταφοράς σε περίπτωση που δεν μπορεί να ανταποκριθεί αμέσως στο αίτημα του αφέντη. Στην Εικόνα 96, ο σκλάβος εκδίδει ένα σήμα επιβεβαίωσης *s_ack* στην ακμή ρολογιού 4. Αυτό είναι ένα κύκλο ρολογιού αργότερα από την νωρίτερη δυνατή ακμή ρολογιού 3. Ένας τέτοιος κύκλος καθυστέρησης ονομάζεται **κατάσταση αναμονής (wait state)**: η συναλλαγή του διαύλου επεκτείνεται για ένα κύκλο ρολογιού. Οι καταστάσεις αναμονής επιτρέπουν την επικοινωνία μεταξύ εξαρτημάτων διαύλου με πολύ διαφορετική ταχύτητα. Ωστόσο, οι καταστάσεις αναμονής αποτελούν επίσης μειονέκτημα. Κατά τη διάρκεια μιας κατάστασης αναμονής, ο δίαυλος είναι δεσμευμένος και δεν είναι προσβάσιμος σε άλλους αφέντες. Σε ένα σύστημα με πολλούς αργούς σκλάβους, αυτό θα επηρεάσει σημαντικά τη συνολική απόδοση του συστήματος.

Σημασία: Συνθήκη λήξης χρόνου

Ένα χρονικό όριο διαύλου (bus timeout) μπορεί να χρησιμοποιηθεί, για να αποφευχθεί ότι ένας σκλάβος δεσμεύει πλήρως ένα δίαυλο. Εάν, μετά από ένα δεδομένο αριθμό κύκλων ρολογιού, δεν λαμβάνεται απόκριση από το σκλάβο διαύλου, ο διαιτητής του διαύλου μπορεί να δηλώσει μια συνθήκη λήξης χρόνου (timeout). Η συνθήκη λήξης χρόνου θα ειδοποιήσει τον αφέντη διαύλου για να εγκαταλείψει το δίαυλο και να διακόψει τη μεταφορά.



Εικόνα 96 - Μεταφορά εγγραφής με μία κατάσταση αναμονής σε γενικό περιφερειακό δίαυλο



Εικόνα 97 - Μεταφορά ανάγνωσης χωρίς κατάσταση αναμονής σε γενικό περιφερειακό δίαυλο

Σημασία: Περιγραφή Εικόνας 97

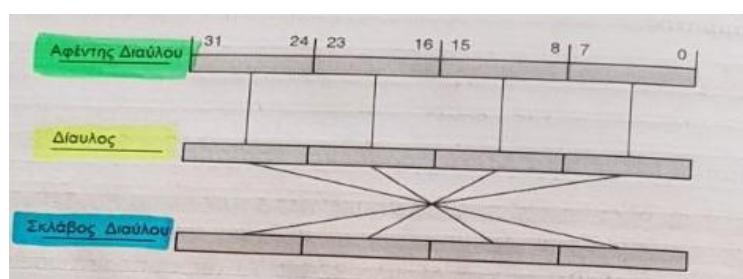
Η Εικόνα 97 δείχνει μια μεταφορά ανάγνωσης χωρίς καταστάσεις αναμονής. Το πρωτόκολλο είναι σχεδόν ταυτόσημο με τη μεταφορά εγγραφής. Μόνο η κατεύθυνση των δεδομένων αντιστρέφεται (από το σκλάβο στον αφέντη) και η γραμμή ελέγχου m_rnw παραμένει σε υψηλή στάθμη για να δείξει μια μεταφορά ανάγνωσης. Τα πρωτόκολλα διαύλου για ανάγνωση και εγγραφή που περιγράφονται εδώ είναι τυπικά για περιφερειακούς διαύλους.

10.1.9 Ορισμοί Μεγέθους Μεταφορών και Endianess

Από προεπιλογή, όλοι οι αφέντες και οι σκλάβοι σε ένα δίαυλο εντός ολοκληρωμένου θα χρησιμοποιήσουν **ομοιόμορφο μήκος λέξης (θ.μ.)** και ομοιόμορφη οργάνωση μνήμης. Για παράδειγμα, οι αφέντες, οι σκλάβοι και ο δίαυλος θα μπορούσαν να χρησιμοποιούν λέξεις little-endian 32 bit. Αυτό θα σημαίνει ότι κάθε μεταφορά δεδομένων διακινεί 32 bits και ότι το λιγότερο σημαντικό byte θα βρεθεί στο χαμηλότερο byte της 32-bit λέξης. Εφόσον ο αφέντης, ο δίαυλος και ο σκλάβος κάνουν πανομοιότυπες υποθέσεις για το μορφότυπο δεδομένων, ένα μόνο αίτημα και ένα μόνο σήμα επιβεβαίωσης θα επαρκεί για τον έλεγχο της μεταφοράς δεδομένων.

Σημείωση: Transfer sizing

Ένα σύστημα διαύλου θα χρειαστεί επίσης να παρέχει έναν μηχανισμό για τον ορισμό μεγέθους μεταφοράς (transfer sizing): επιλέγοντας ποιο μέρος μιας δεδομένης λέξης ανήκει στην πραγματική μεταφορά δεδομένων. Σε ένα δίαυλο δεδομένων 32-bit, για παράδειγμα, είναι χρήσιμο να μπορούμε να μεταφέρουμε ένα μεμονωμένο byte ή μισή λέξη (16 bit). Ο ορισμός **μεγέθους μεταφοράς** εκφράζεται με σήματα επίτρεψης-byte (byte-enable) ή με απευθείας κωδικοποίηση του μεγέθους της μεταφοράς ως μέρος των σημάτων ελέγχου διαύλου. Η προηγούμενη μέθοδος, χρησιμοποιώντας σήματα επίτρεψης-byte, είναι λίγο πιο γενική από την τελευταία, επειδή επιτρέπει σε κάποιον να αντιμετωπίσει τις μη ευθυγραμμισμένες μεταφορές.

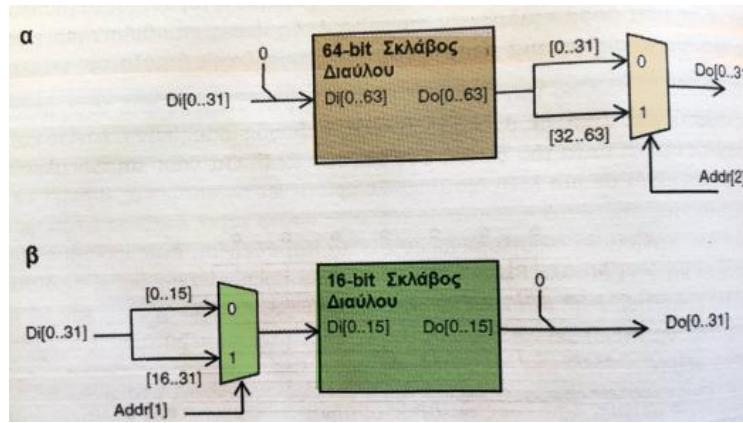


Εικόνα 98 - Σύνδεση ενός big-endian σκλάβου με little-endian αφέντη

Σημείωση: Αντιμετώπιση ποικίλων απαιτήσεων σχετικά με το μήκος λέξης εκ' μέρους των διαύλων

Η Εικόνα 99 δείχνει στο (α) μέρος της το πως μπορεί να συνδεθεί ένας σκλάβος διαύλου 64 – bit και στο (β) μέρος της το πως μπορεί να συνδεθεί ένας σκλάβος διαύλου 16 – bit σε ένα δίαυλο 32 – bit. Στην πρώτη περίπτωση μια εγγραφή δεδομένων θα μεταφέρει κάθε φορά μόνο 32 bits. Τα επάνω (πιο σημαντικά) 32 bits είναι καλωδιωμένα στο μηδέν. Στην περίπτωση της ανάγνωσης – δεδομένων, μια από τους γραμμές διευθύνσεων και πιο συγκεκριμένα η γραμμή Addr(2) θα χρησιμοποιηθεί ως γραμμή επιλογής του MUX 2-1 για να επιλέξει τα 32 bits χαμηλής τάξης και στη συνέχεια θα επιλέξει και επιλέξει τα 32 bits υψηλής τάξης. Στην περίπτωση του

σκλάβου των 16 – bits το σύστημα διαύλου των 32 – bit μπορεί να παραδώσει περισσότερα δεδομένα από όσα μπορεί να χειριστεί ο σκλάβου διαύλου και τότε ένα επιπρόσθετο bit, το Addr[1] χρησιμοποιείται για να προσδιορίσει ποιο μέρος του διαύλου των 32 – bit θα μεταφερθεί στο σκλάβο διαύλου των 16 – bit. Συνοπτικά θα πρέπει να αναφερθεί ότι οι δίαυλοι είναι σε θέση να αντιμετωπίσουν ποικίλες απαιτήσεις σχετικά με το μήκος λέξης, με την εισαγωγή επιπρόσθετων σημάτων ελέγχου και με την προσθήκη επιπλέον υλικού πολυπλεξίας γύρω από σκλάβους/αφέντες διαύλου.



10.1.10 Βελτιωμένες Μεταφορές Διαύλων

Σημείωση: Δραστηριότητες (φάσεις) μεταφορών διαύλου

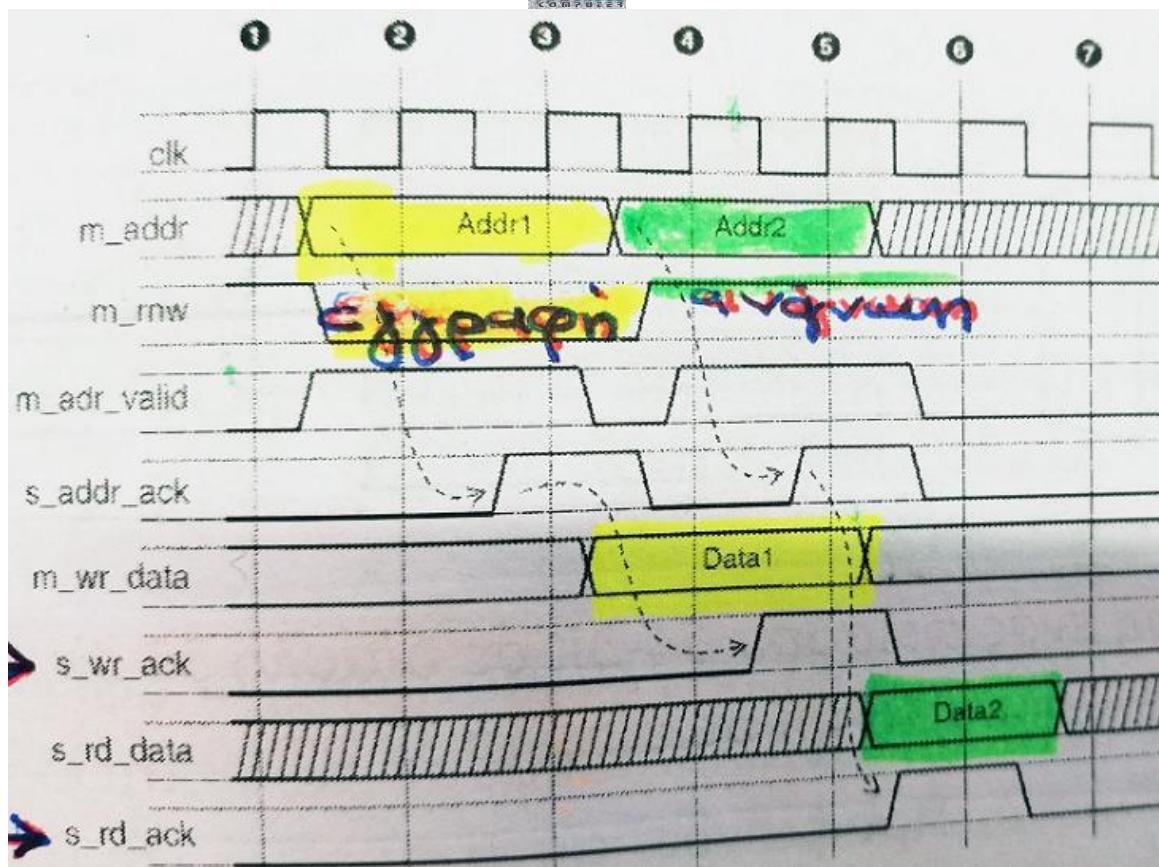
Κάθε μεταφορά δεδομένων διαύλου περιλαμβάνει πολλαπλές φάσεις. Πρώτον, ο αφέντης διαύλου πρέπει να διαπραγματευτεί μια πρόσβαση διαύλου με το διαιτητή διαύλου. Στη συνέχεια, ο αφέντης διαύλου πρέπει να εκδώσει μια διεύθυνση διαύλου και μια εντολή διαύλου. Τρίτον, ο σκλάβος διαύλου πρέπει να επιβεβαιώσει τη μεταφορά δεδομένων. Τέλος, ο αφέντης διαύλους πρέπει να τερματίσει τη μεταφορά του διαύλου και να απελευθερώσει τον έλεγχο πάνω στον δίαυλο. Κάθε μία από αυτές τις δραστηριότητες απαιτεί ένα πεπερασμένο χρονικό διάστημα για να ολοκληρωθεί.

Σημείωση: Μηχανισμοί επιτάχυνσης μεταφορών διαύλων

Οι δίαυλοι εντός ολοκληρωμένου χρησιμοποιούν τρεις μηχανισμούς για να επιταχύνουν αυτές τις μεταφορές. Ο πρώτος μηχανισμός, **ο διαχωρισμός συναλλαγών (transaction splitting)**, που διαχωρίζει κάθε συναλλαγή διαύλου σε πολλαπλές φάσεις και επιτρέπει σε κάθε φάση να ολοκληρωθεί ξεχωριστά. Αυτό εμποδίζει τη δεσμευση του διαύλου για μεγάλο χρονικό διάστημα. Ο δεύτερος μηχανισμός, **η διοχέτευση (pipelining)** εισάγει επικάλυψη στην εκτέλεση των φάσεων μεταφοράς διαύλου. Ο τρίτος μηχανισμός, **η λειτουργία ριπής (burstmode operation)**, επιτρέπει τη μεταφορά πολλαπλών στοιχείων δεδομένων, που βρίσκονται σε κοντινές διευθύνσεις, κατά τη διάρκεια μίας μόνο συναλλαγής διαύλου.

Σημείωση: Ταυτόχρονη λειτουργία διαχωρισμού συναλλαγών και διοχέτευσης

Ένας δίαυλος μπορεί να χρησιμοποιεί έναν ή περισσότερους από τους μηχανισμούς αυτούς ταυτόχρονα. Δύο από αυτούς, ο διαχωρισμός συναλλαγών και οι διοχετευόμενες μεταφορές, συχνά συμβαίνουν μαζί. Θα τους παρουσιάσουμε ταυτόχρονα.



Εικόνα 100 - Παράδειγμα διοχετευμένης ανάγνωσης/εγγραφής σε γενικό δίαυλο

10.1.10.1 Διαχωρισμός Συναλλαγών και Διοχετευμένες Μεταφορές

Σημείωση: Πως λειτουργεί ο διαχωρισμός συναλλαγών

Στο διαχωρισμό συναλλαγών (transaction splitting) μια ενιαία μεταφορά διαύλου διασπάται σε ξεχωριστές φάσεις. Κάθε μία από αυτές τις μεταφορές (φάσεις) έχει διαφορετικό σήμα επιβεβαίωσης. Το σκεπτικό είναι ότι ένας σκλάβος διαύλου θα χρειαστεί λίγο χρόνο μετά τη λήψη μιας διεύθυνσης για να προετοιμαστεί για τη μεταφορά δεδομένων. Συνεπώς, μετά τη μεταφορά μιας διεύθυνσης, ο αφέντης διαύλου θα πρέπει να απελευθερώθει από την αναμονή για το σκλάβο διαύλου και να συνεχίσει.

Σημείωση: Παράδειγμα διαχωρισμού / διοχέτευσης συναλλαγών

Η διοχέτευση συναλλαγών (transaction pipelining) συμβαίνει όταν πολλαπλές συναλλαγές μπορούν να προχωρήσουν ταυτόχρονα, καθεμία σε διαφορετικό επίπεδο προόδου. Οι διαχωρισμένες συναλλαγές είναι καλές υποψήφιες για αλληλεπικαλυπτόμενη εκτέλεση. Η Εικόνα 100 παρουσιάζει ένα παράδειγμα **αλληλεπικαλυπτόμενων** μεταφορών ανάγνωσης/εγγραφής για ένα γενικό δίαυλο. Δύο μεταφορές εμφανίζονται στην εικόνα: μια εγγραφή ακολουθούμενη από μια ανάγνωση. Ο δίαυλος που χρησιμοποιείται σε αυτήν την εικόνα είναι ελαφρώς διαφορετικός από αυτόν που χρησιμοποιείται στις Εικόνες 96 και 97. Η διαφορά είναι ότι υπάρχουν δύο σήματα επιβεβαίωσης (acknowledge) και όχι ένα μόνο. Στην ακμή ρολογιού 2, ο αφέντης διαύλου δηλώνει μια εγγραφή στη διεύθυνση Addr1. Ο σκλάβος διαύλου επιβεβαιώνει αυτήν τη διεύθυνση στην ακμή ρολογιού 3. Ωστόσο, αυτήν τη στιγμή η μεταφορά δεδομένων δεν έχει ολοκληρωθεί ακόμα. Επιβεβαιώνοντας τη διεύθυνση, ο σκλάβος απλά δηλώνει ότι είναι έτοιμος να δεχτεί δεδομένα. Από την ακμή ρολογιού 4, ο αφέντης διαύλου εκτελεί

δύο δραστηριότητες. Κατ' αρχάς, στέλνει τα δεδομένα που πρόκειται να γραφτούν, Data1, στον σκλάβο διαύλου. Στη συνέχεια, ξεκινάει την επόμενη μεταφορά οδηγώντας μια νέα διεύθυνση Addr2 στο δίαυλο. Στην ακμή ρολογιού 5 δύο γεγονότα λαμβάνουν χώρα: ο σκλάβος διαύλου αποδέχεται το Data1 και επίσης επιβεβαιώνει τη διεύθυνση ανάγνωσης Addr2. Τελικά, στην ακμή ρολογιού 6, ο σκλάβος διαύλου επιστρέφει τα δεδομένα που προκύπτουν από αυτήν τη λειτουργία ανάγνωσης, Data2.

10.1.10.2 Μεταφορές Ριπής

Η τρίτη τεχνική για τη βελτίωση της απόδοσης συναλλαγών (μεταφορών) διαύλου είναι η χρήση μεταφορών ριπής (*burstmode transfers*). Αυτή θα μεταφέρει πολλαπλά στοιχεία δεδομένων από κοντινές διευθύνσεις **σε μια συναλλαγή διαύλου, με άλλα λόγια με μια μεταφορά διαύλου**.

Σημείωση: Χρησιμότητα μεταφορών ριπής

Οι μεταφορές ριπής είναι χρήσιμες όταν απαιτούνται μεταφορές γειτονικών στοιχείων δεδομένων. Οι προσπελάσεις κύριας μνήμης που γίνονται από έναν επεξεργαστή με κρυφή μνήμη είναι ένα παράδειγμα. Όταν υπάρχει αστοχία κρυφής μνήμης στον επεξεργαστή, πρέπει να αντικατασταθεί μια ολόκληρη γραμμή κρυφής μνήμης. Εάν υποθέσουμε ότι θα υπήρχαν 32 bytes σε μια γραμμή κρυφής μνήμης, τότε μια αστοχία κρυφής μνήμης συνεπάγεται την ανάγνωση 32 διαδοχικών bytes από τη μνήμη.

Σημείωση: Κατηγορίες μεταφορών ριπής

Οι μεταφορές σε μορφή – ριπής μπορούν να έχουν σταθερό ή μεταβλητό μήκος. Σε ένα σχήμα ριπής σταθερού μήκους, ο αφέντης διαύλου θα διαπραγματεύεται τις ιδιότητες της ριπής στην αρχή της μεταφοράς ριπής και στη συνέχεια θα πραγματοποιήσει κάθε μεταφορά μέσα στην ριπή. Σε ένα σχήμα ριπής μεταβλητού μήκους, ο αφέντης διαύλου (ή ο σκλάβος διαύλου) έχει την επιλογή να τερματίσει τη ριπή μετά από κάθε μεταφορά. Οι διεύθυνσεις μέσα σε μια ριπή είναι συνήθως σταδιακά αυξανόμενες, αν και υπάρχουν επίσης εφαρμογές όπου η διεύθυνση πρέπει να παραμείνει σταθερή ή όπου η διεύθυνση αυξάνεται με μια λειτουργία modulo. Έτσι, ως τμήμα της προδιαγραφής ριπής, ένας δίαυλος μπορεί να επιτρέψει στο χρήστη να καθορίσει τη φύση της ακολουθίας διεύθυνσεων ριπής.

Τέλος, το βήμα διεύθυνσης θα εξαρτηθεί από το μέγεθος των δεδομένων εντός της ριπής: τα bytes, οι μισές λέξεις (halfwords) και οι λέξεις (words) θα αυξάνουν τις διεύθυνσεις κατά 1, 2 και 4 αντίστοιχα. Προφανώς, όλες αυτές οι επιλογές συνεπάγονται την προσθήκη επιπλέον σημάτων ελέγχου στο δίαυλο, στη μεριά του αφέντη καθώς επίσης και του σκλάβου. Ο Πίνακας 10.2 παρουσιάζει τα κύρια χαρακτηριστικά για την υποστήριξη ριπής στα συστήματα διαύλων Coreconnect, AMBA, Avalon και Wishbone.

Πίνακας 10.2 Σχήματα μεταφοράς ριπής

Ιδιότητα Ριπής	CoreConnect/OPB	AMBA/APB
Μήκος Ριπής	Σταθερό (2 ..16) ή μεταβλητό	Σταθερό (4, 8, 1) ή μεταβλητό
Ακολουθία Διεύθυνσεων	Incr	Incr/mod/const

Μέγεθος μεταφοράς	Καθορίζεται από δίαυλο	Byte/halfword/word
Ιδιότητα Ριπής Avalon-MM	Wishbone	
Μήκος Ριπής	Σταθερό ($1 \dots 2^{11}$)	Μεταβλητό
Ακολουθία Διευθύνσεων	Incr/mod/const	Incr/const
Μέγεθος μεταφοράς	Byte/halfword/word	Byte/halfword/word

Σημείωση: Παράδειγμα μεταφοράς ριπής

Ένα παράδειγμα μιας μεταφοράς ριπής εμφανίζεται στην Εικόνα 101. Αυτή η μεταφορά απεικονίζει μια μεταφορά ριπής τεσσάρων λέξεων σε συνεχόμενες θέσεις διευθύνσεων. Εκτός από τις εντολές που συζητήθηκαν προηγουμένως (`m_addr`, `m_rnw`, `m_adr_valid`), μια εντολή **m_burst** χρησιμοποιείται για να υποδείξει τον τύπο μεταφοράς ριπής που εκτελείται από τον αφέντη. Σε αυτό το γενικό δίαυλο, υποθέτουμε ότι ένας από τους τύπους ριπής κωδικοποιείται ως `increment_4`, που ερμηνεύεται ως μια ριπή τεσσάρων διαδοχικών μεταφορών με αυξανόμενη διεύθυνση. Στην ακμή ρολογιού 3, ο σκλάβος αποδέχεται αυτή τη μεταφορά με το σήμα `s_addr_ack` και μετά από αυτό ο αφέντης θα παρέχει τέσσερις λέξεις δεδομένων στη σειρά.

Οι διευθύνσεις υποδηλώνονται από τον τύπο της ριπής (`increment_4`) και τη διεύθυνση της πρώτης μεταφοράς. Η Εικόνα 101 υποθέτει ότι το μήκος λέξης κάθε μεταφοράς είναι 4 bytes (μία λέξη). Αξίζει να σημειωθεί ότι πρώτα ενεργοποιείται από τη μεριά του σκλάβου το σήμα `s_addr_ack` για αποδοχή της διεύθυνσης μεταφοράς της ριπής και στη συνέχεια ενεργοποιείται το σήμα `s_wr_ack` για αποδοχή της εγγραφής της ριπής.



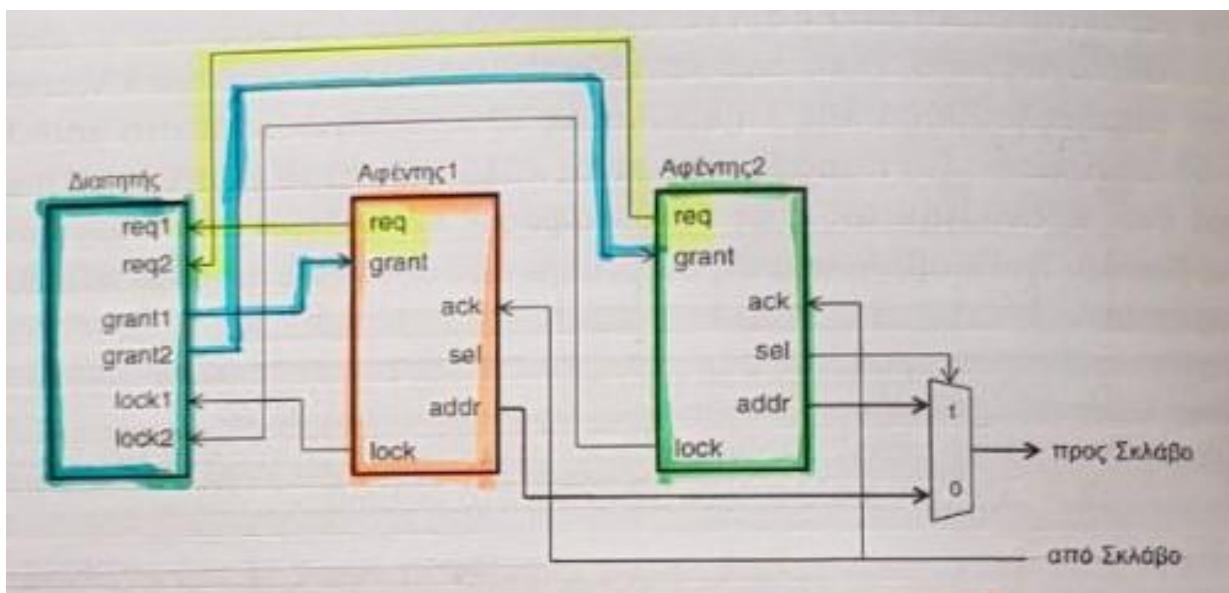
Εικόνα 101 - Μια ριπή εγγραφής τεσσάρων-παλμών (four-beat)

Σημείωση: Πότε απαιτείται διαπραγμάτευση

Όταν υπάρχουν περισσότεροι από ένας αφέντες σε ένα δίαυλο, κάθε μεταφορά διαύλου απαιτεί διαπραγμάτευση. Ένας διαιτητής διαύλου (bus arbiter) θα ελέγχει αυτήν τη διαδικασία διαπραγμάτευσης και θα κατανέμει κάθε θυρίδα μεταφοράς (transfer slot) σε έναν αφέντη διαύλου.

Σημείωση: Σημασία σημάτων request-grant-lock

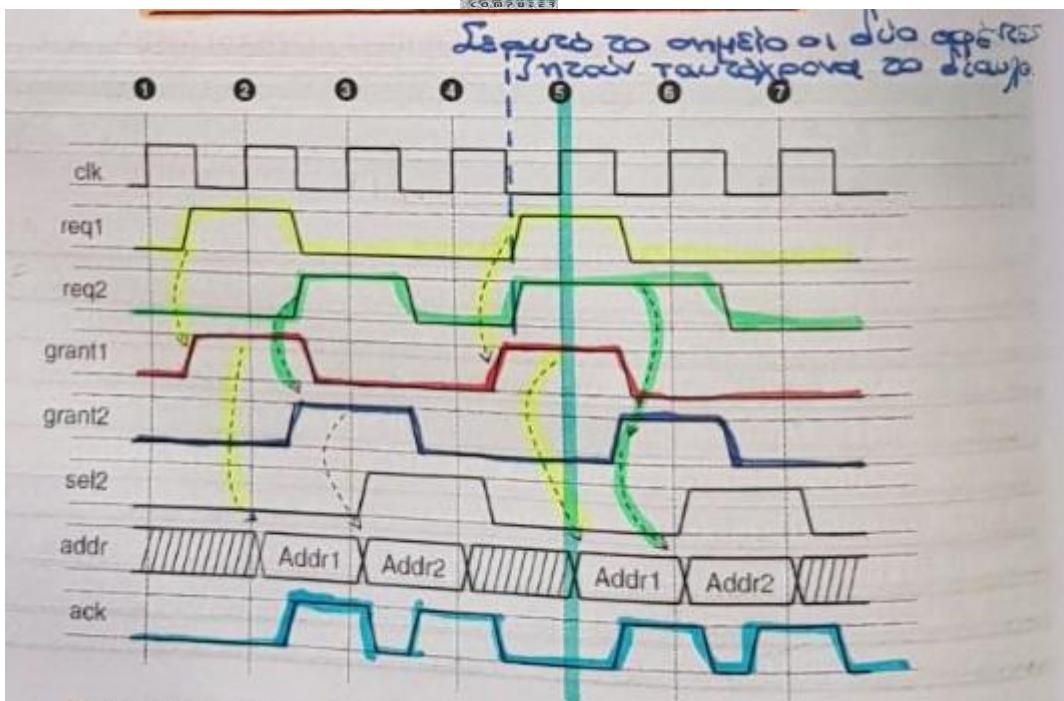
Η Εικόνα 102 παρουσιάζει την τοπολογία ενός (γενικού) διαύλου πολλαπλών αφεντών (multi-master) με δύο αφέντες και ένα κύκλωμα διαιτητή. Οι σκλάβοι δεν φαίνονται στην εικόνα. Από τα συνήθη χαρακτηριστικά διαύλου, ορατά είναι μόνο ο δίαυλος διευθύνσεων και ένα σήμα επιβεβαίωσης-μεταφοράς. Κάθε αφέντης μπορεί να ζητήσει πρόσβαση στο δίαυλο μέσω της σύνδεσης **request**. Ο διαιτητής χρησιμοποιεί το σήμα **grant** για να δείξει στον αφέντη ότι μπορεί να προσπελάσει τον δίαυλο. Μόλις ο αφέντης αποκτήσει τον έλεγχο του διαύλου, θα προχωρήσει σε ένα από τα συνηθισμένα σχήματα μεταφοράς διαύλου. Τα σήματα **lock** χρησιμοποιούνται από τον αφέντη για να αποκτήσει αποκλειστικό έλεγχο πάνω στον δίαυλο και θα αποσαφηνιστούν αργότερα.



Εικόνα 102 - Διαιτησία πολλαπλών-αφεντών

Σημείωση: Περιγραφή εικόνας 103

Η Εικόνα 103 δείχνει πως δύο αφέντες ανταγωνίζονται για το δίαυλο για αρκετούς κύκλους ρολογιού. Στην ακμή ρολογιού 2, ο αφέντης 1 αιτείται για το δίαυλο μέσω του σήματος reg1. Από τη στιγμή που ο αφέντης 2 δεν χρειάζεται τον δίαυλο εκείνη την στιγμή, ο διαιτητής θα παραχωρήσει το δίαυλο στον αφέντη 1. Θα πρέπει να σημειωθεί ότι το σήμα grant έρχεται ως άμεση απάντηση στο σήμα request και πιο συγκεκριμένα το σήμα grant1. Αυτό σημαίνει ότι η διαδικασία διαπραγμάτευσης διαύλου μπορεί να ολοκληρωθεί μέσα σε ένα μόνο κύκλο ρολογιού. Επιπλέον, συνεπάγεται ότι ο διαιτητής θα χρειαστεί να χρησιμοποιήσει συνδυαστική λογική για να παράγει το σήμα grant βάσει του σήματος request.



Εικόνα 103 - Χρονισμός διαιτησίας πολλαπλών-αφεντών

Σημείωση: Επικαλυπτόμενος κύκλος διαιτησίας

Μετά την ακμή ρολογιού 2, ο αφέντης 1 οδηγεί μια διεύθυνση στο δίαυλο διεύθυνσης και ολοκληρώνει μια κανονική μεταφορά διαύλου. Υποθέτουμε ότι ο σκλάβος επιβεβαιώνει την ολοκλήρωση αυτής της μεταφοράς στην ακμή ρολογιού 3, θέτοντας το σήμα ack σε υψηλή στάθμη. Το νωρίτερο όπου μπορεί να λάβει χώρα, η επόμενη διαιτησία για μεταφορά διαύλου, είναι η ακμή ρολογιού 3. Αυτό ονομάζεται επικαλυπτόμενος (*overlapping*) κύκλος διαιτησίας, διότι η διαιτησία της επόμενης μεταφοράς πραγματοποιείται την ίδια στιγμή με την ολοκλήρωση της τρέχουσας μεταφοράς. Η δεύτερη μεταφορά παραχωρείται στον αφέντη 2 με το σήμα grant2 και ολοκληρώνεται στην ακμή ρολογιού 4, όπου το σήμα ack γίνεται και πάλι ενεργό.

Σημείωση: Επίλυση προτεραιότητας

Μεταξύ των ακμών ρολογιού 4 και 5, ο δίαυλος βρίσκεται σε αδράνεια για έναν κύκλο, επειδή κανένας αφέντης δεν ζήτησε πρόσβαση στον δίαυλο. Στην ακμή ρολογιού 5, και οι δύο αφέντες 1 και 2 ζητούν πρόσβαση στο δίαυλο. Μόνον ένας αφέντης επιτρέπεται να προχωρήσει, και αυτό σημαίνει ότι υπάρχει μια επίλυση προτεραιότητας (priority resolution) που υλοποιείται μεταξύ των αφεντών. Σε αυτήν την περίπτωση, ο αφέντης 1 έχει σταθερή προτεραιότητα έναντι του αφέντη 2, πράγμα που σημαίνει ότι ο αφέντης 1 θα αποκτά πάντοτε πρόσβαση στο δίαυλο και ο αφέντης 2 θα έχει πρόσβαση στον δίαυλο μόνο όταν ο αφέντης 1 δεν τον χρειάζεται. Η μεταφορά του αφέντη 1 ολοκληρώνεται στην ακμή ρολογιού 6. Δεδομένου ότι ο αφέντης 2 εξακολουθεί να περιμένει αποδοχή της πρόσβασης, μπορεί να προχωρήσει στην ακμή του ρολογιού 6, επειδή ο αφέντης 1 δεν χρειάζεται πλέον το δίαυλο. Η τέταρτη και η τελική μεταφορά τότε ολοκληρώνεται στην ακμή ρολογιού 7.

10.1.11 Προτεραιότητα Διαύλου

Σημείωση: Σχήμα σταθερής προτεραιότητας

Το διάγραμμα χρονισμού στην Εικόνα 102 αποκαλύπτει την ενδιαφέρουσα έννοια της προτεραιότητας. Όταν πολλαπλοί αφέντες επιχειρούν να αποκτήσουν πρόσβαση στον δίαυλο συγχρόνως, επιτρέπεται να προχωρήσει μόνο ένας αφέντης με βάση την ανάλυση προτεραιότητας.

Σημείωση: Σχήμα σταθερής προτεραιότητας

Το πιο απλό σχήμα προτεραιότητας είναι να κατανείμουμε μια σταθερή προτεραιότητα, αυστηρά αυξανόμενη, σε κάθε αφέντη.

Σημείωση: Μειονέκτημα σταθερής προτεραιότητας

Ενώ υλοποιείται εύκολα, δεν είναι απαραιτήτως η καλύτερη λύση. Όταν ένας αφέντης υψηλής προτεραιότητας συνεχώς προσπελαύνει τον δίαυλο, άλλοι χαμηλής προτεραιότητας αφέντες, μπορεί να αδυνατούν να αποκτήσουν πρόσβαση στο δίαυλο, για παρατεταμένα χρονικά διαστήματα.

Σημείωση: Σήματα ίσης προτεραιότητας → εκ' περιτροπής και LRU

Συχνά, πολλαπλοί αφέντες σε ένα δίαυλο πρέπει να έχουν ίση προτεραιότητα. Για παράδειγμα, όταν οι επεξεργαστές σε συμμετρικούς πολυεπεξεργαστές σε ένα μόνο δίαυλο έχουν πρόσβαση στην ίδια μνήμη, κανένας επεξεργαστής δεν θα πρέπει να έχει προτεραιότητα έναντι του άλλου. Σε αυτή την περίπτωση, η ανάλυση προτεραιότητας υλοποιείται με τη χρήση εκ περιτροπής (round – robin) σχήματος. Κάθε αφέντης παίρνει σειρά για να αποκτήσει πρόσβαση στο δίαυλο. Όταν δύο αφέντες ζητούν συνεχώς το δίαυλο, τότε οι μεταφορές διαύλου του αφέντη 1 και του αφέντη 2 θα εναλλάσσονται. Μια άλλη πιθανή λύση είναι το σχήμα λιγότερο πρόσφατα χρησιμοποιούμενο (least-recently-used), στο οποίο ο αφέντης που περίμενε για το δίαυλο για το μεγαλύτερο χρονικό διάστημα θα έχει πρόσβαση πρώτα.

Σημείωση: Μειονεκτήματα σχημάτων ίσης προτεραιότητας → Τι γίνεται στην πράξη

Σχήματα ίσης προτεραιότητας, όπως το εκ περιτροπής ή το λιγότερο πρόσφατα χρησιμοποιούμενο, αποφεύγουν την λιμοκτονία (starvation) των αφεντών του διαύλου, αλλά καθιστούν επίσης απρόβλεπτη την απόδοση του διαύλου. Όταν εργαζόμαστε με εφαρμογές κρίσιμου λανθάνοντας χρόνου, αυτό μπορεί να είναι ένα πρόβλημα. Για να αντιμετωπιστεί αυτό, οι σχεδιαστές μπορούν να χρησιμοποιήσουν ένα μικτό σχήμα που συνδυάζει πολλαπλά επίπεδα προτεραιότητας με ένα σχήμα ίσων προτεραιοτήτων για να επιτρέψει σε αρκετούς αφέντες να μοιραστούν το ίδιο επίπεδο προτεραιότητας. Ο αλγόριθμος προτεραιότητας που χρησιμοποιείται από τον διατητή διαύλου δεν αποτελεί μέρος του ορισμού του πρωτοκόλλου μεταφοράς διαύλου.

10.1.12 [Κλείδωμα Διαύλου](#)

Σημείωση: Για ποιους λόγους χρειάζεται το κλείδωμα διαύλου. Τί είναι το bus locking;

Η τελευταία ιδέα σε σχήματα διαύλων πολλαπλών-αφεντών είναι το κλείδωμα διαύλου (bus locking): η αποκλειστική εκχώρηση ενός διαύλου σε ένα μόνο αφέντη για τη διάρκεια πολλαπλών μεταφορών.

Σημείωση: Πότε χρειάζεται κλείδωμα διαύλου

Υπάρχουν διάφοροι λόγοι για τους οποίους μπορεί να χρειαστεί το κλείδωμα διαύλου. Πρώτον, όταν πρέπει να μεταφερθούν μεγάλα τμήματα δεδομένων με αυστηρές απαιτήσεις λανθάνοντα χρόνου, μπορεί να απαιτείται



αποκλειστική πρόσβαση στο δίαυλο. Ενώ οι μεταφορές ριπής μπορούν να βοηθήσουν έναν αφέντη να ολοκληρώσει τις μεταφορές γρήγορα, οι μεταφορές αυτές είναι δυνατό να διακοπούν από άλλο αφέντη με υψηλότερη προτεραιότητα. Με το κλείδωμα διαύλου, ο αφέντης μπορεί να εξασφαλίσει ότι κάτι τέτοιο δε θα συμβεί.

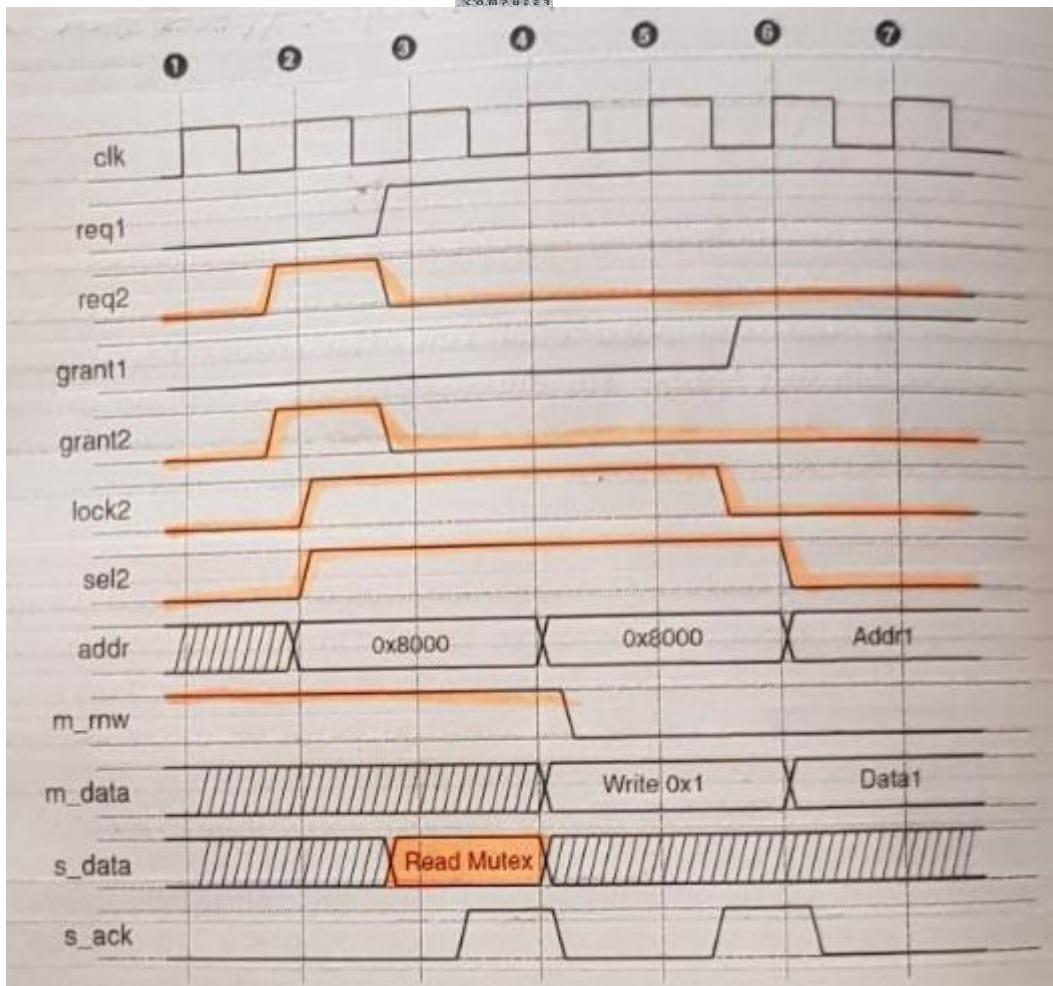
Η δεύτερη ανάγκη για κλείδωμα διαύλου είναι όταν ο αφέντης πρέπει να έχει εγγυημένη, αποκλειστική πρόσβαση σε διαδοχικές μεταφορές, συνήθως μια μεταφορά ανάγνωσης ακολουθούμενη από μια μεταφορά εγγραφής. Αυτό χρειάζεται, για παράδειγμα, όταν υλοποιείται μια εντολή test-and-set (έλεγχε-και-γράψε). Η εντολή test-and-set χρησιμοποιείται για τη δημιουργία ενός mutex, ενός πολύ γνωστού πρωταρχικού στοιχείου λογισμικού για την υλοποίηση αμοιβαίου αποκλεισμού. Ένα mutex είναι παρόμοιο με ένα σηματοφόρο αλλά δεν είναι ταυτόσημο με αυτό.

Σημείωση: Σημασία και χρήση mutex

Ένα mutex υποδηλώνει ιδιοκτησία: μόλις κλειδωθεί ένα mutex, μπορεί να ξεκλειδωθεί μόνο από την ίδια οντότητα που το κλείδωσε. Ένα παράδειγμα όπου μπορεί να χρησιμοποιηθεί ένα mutex είναι ο έλεγχος της πρόσβασης δύο αφεντών σε μια κοινόχρηστη περιοχή μνήμης. Οι αφέντες διαύλου θα ελέγχουν την πρόσβαση χρησιμοποιώντας ένα mutex, υλοποιούμενο μέσω μιας εντολής test-and-set όπως περιγράφεται στη συνέχεια.

Ένα παράδειγμα υλοποίησης της test-and-set φαίνεται παρακάτω. Αυτό το πρόγραμμα C εκτελείται σε κάθε έναν από τους δύο επεξεργαστές (αφέντες διαύλου) που είναι συνδεδεμένοι στον ίδιο δίαυλο. Μοιράζονται μια θέση μνήμης στη διεύθυνση 0x8000. Καλώντας την testandset, ένας επεξεργαστής θα προσπαθήσει να διαβάσει αυτή τη θέση μνήμης και να γράψει σε αυτήν κατά τη διάρκεια μίας μόνο λειτουργίας κλειδωμένου διαύλου. Αυτό σημαίνει ότι δεν μπορεί να διακοπεί η συνάρτηση test_and_set () : μόνο ένας επεξεργαστής θα μπορεί να διαβάσει την τιμή του mutex όταν η τιμή του είναι χαμηλή. Οι δύο επεξεργαστές χρησιμοποιούν αυτή τη συνάρτηση ως εξής. Πριν προσπελάσουν τον κοινόχρηστο πόρο οι επεξεργαστές θα καλούν την enter (), ενώ θα καλέσουν την leave (). Ο κοινόχρηστος πόρος μπορεί να είναι οτιδήποτε χρειάζεται αποκλειστική πρόσβαση από έναν από τους επεξεργαστές.

```
int *mutex = (int *) 0x8000; // θέση του mutex
```



Εικόνα 104 - Λειτουργία test-and-set με κλείδωμα διαύλου

Σημείωση: Παράδειγμα mutex

Η Εικόνα 103 δείχνει ένα παράδειγμα test-and-set με κλείδωμα διαύλου. Στην ακμή ρολογιού 2, ο αφέντης 2 ζητά πρόσβαση στο δίαυλο χρησιμοποιώντας το req2. Η πρόσβαση αυτή παραχωρείται από τον διαιτητή μέσω του grant2. Μετά την ακμή ρολογιού 2, ο αφέντης αυτός αρπάζει το δίαυλο χρησιμοποιώντας το sel2 και τον κλειδώνει χρησιμοποιώντας το lock2. Ο αφέντης 2 θα εκτελέσει τώρα μια λειτουργία test-and-set, η οποία περιλαμβάνει ανάγνωση μιας διεύθυνσης μνήμης ακολουθούμενη αμέσως μετά από μια εγγραφή στην ίδια διεύθυνση μνήμης. Η λειτουργία ανάγνωσης ξεκινάει από την ακμή ρολογιού 3 και ολοκληρώνεται στην ακμή ρολογιού 4. Στην ακμή ρολογιού 3, ο αφέντης οδηγεί μια διεύθυνση στο δίαυλο και σηματοδοτεί μια λειτουργία ανάγνωσης (m_rnw). Στην ακμή ρολογιού 4, ο σκλάβος παραδίδει τα δεδομένα που είναι αποθηκευμένα στη διεύθυνση αυτή και ολοκληρώνει τη μεταφορά χρησιμοποιώντας το s_ack.

Σημείωση: Περιγραφή Εικόνας 103

Εν τω μεταξύ, ένας άλλος αφέντης ζήτησε πρόσβαση στο δίαυλο ξεκινώντας από την ακμή ρολογιού 3 (χρησιμοποιώντας το req1). Ωστόσο, επειδή ο αφέντης 2 έχει κλειδώσει το δίαυλο, ο διαιτητής θα αγνοήσει αυτό το αίτημα. Στον αφέντη 2 θα παραχωρηθεί περαιτέρω χρήση του διαύλου μέχρι να απελευθερώσει το κλείδωμα. Αυτή η πρόσβαση είναι εγγυημένη ακόμα και αν ο αφέντης 2 έχει χαμηλότερη προτεραιότητα από τους άλλους αφέντες που ζητούν το δίαυλο.

Αφού εκτελέσει το τμήμα ανάγνωσης της εντολής test-and-set, ο αφέντης 2 θα γράψει τώρα ένα "1" στην ίδια θέση. Στην ακμή ρολογιού 5, ο αφέντης 2 βάζει τη διεύθυνση και τα δεδομένα στο δίαυλο και στην ακμή ρολογιού 6 ο σκλάβος δέχεται τα δεδομένα. Το κλείδωμα μπορεί να απελευθερωθεί μετά την ακμή ρολογιού 5. Σημειώστε ότι, σε περίπτωση αποτυχίας της λειτουργίας εγγραφής στο σκλάβο, η πλήρης εντολή test-and-set έχει αποτύχει. Υποθέτουμε, ωστόσο, ότι η λειτουργία εγγραφής ολοκληρώνεται σωστά. Μόλις ο αφέντης 2 απελευθερώσει το lock2, ο έλεγχος θα περάσει στον αφέντη 1, λόγω του αιτήματος σε εκκρεμότητα στο req1. Ως αποτέλεσμα, ξεκινώντας με την ακμή ρολογιού 6, μια νέα μεταφορά διαύλου μπορεί να ξεκινήσει, η οποία κατανέμεται στον αφέντη 1.

Σημείωση: Συμπέρασμα

Συνοψίζοντας, όταν συνδέονται πολλαπλοί αφέντες σε ένα μόνο δίαυλο, πρέπει να γίνεται διαιτησία των μεμονωμένων μεταφορών διαύλου. Επιπλέον, μπορεί να χρησιμοποιηθεί ένα σχήμα προτεραιότητας μεταξύ των αφεντών, για να εξασφαλιστούν οι απαιτήσεις λανθάνοντα χρόνου για συγκεκριμένους αφέντες. Τέλος το κλείδωμα διαύλου μπορεί να χρησιμοποιηθεί για την υλοποίηση εγγυημένης πρόσβασης για κάποιο παρατεταμένο χρονικό διάστημα. Δεδομένου ότι όλες αυτές οι τεχνικές υλοποιούνται στο υλικό, στο επίπεδο μεταφοράς διαύλου, είναι πολύ γρήγορες και αποδοτικές. Επομένως, τα συστήματα διαύλου διαδραματίζουν σημαντικό ρόλο στην οικοδόμηση αποδοτικής επικοινωνίας υλικού και λογισμικού.

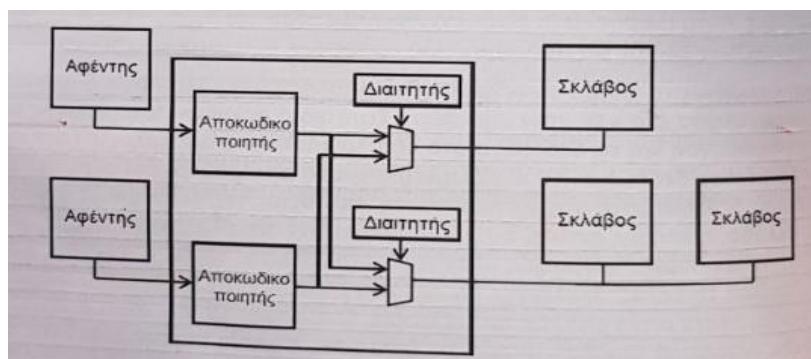
Τοπολογίες Διαύλου

Σημείωση: Προβλήματα τηματοποιημένων διαύλων

Η τοπολογία διαύλου είναι η λογική και φυσική οργάνωση των εξαρτημάτων του διαύλου σε ένα δίκτυο. Μέχρι τώρα έχουμε χρησιμοποιήσει γραμμικές τοπολογίες διαύλου: διαύλους που αποτελούνται από τμήματα διαύλου τα οποία διασυνδέονται με γέφυρες διαύλου.

Σημείωση: Σημασία τοπολογίας διαύλου

Η τοπολογία διαύλου έχει σημαντικό αντίκτυπο στην απόδοση του συστήματος. Ας εξετάσουμε πρώτα ένα ενιαίο τμήμα διαύλου. Όλες οι μονάδες που είναι συνδεδεμένες στο ίδιο τμήμα διαύλου διαμοιράζονται τον ίδιο πόρο επικοινωνίας. Αυτό σημαίνει ότι η επικοινωνία μεταξύ αυτών των μονάδων πρέπει να γίνει ακολουθιακή (sequentialized). Δύο αφέντες διαύλου στο ίδιο τμήμα διαύλου δεν μπορούν να εκκινήσουν παράλληλες μεταφορές διαύλου. Χρειάζονται διαιτησία διαύλου για να γίνει ακολουθιακή η πρόσβαση τους.



Εικόνα 105 - Δίαυλος πολλαπλών επιπέδων

Σημείωση: Μειονεκτήματα γεφυρών

Οι γέφυρες διαύλου μπορούν να χωρίσουν τους διαύλους σε πολλαπλά τμήματα και χρησιμοποιούνται για την ομαδοποίηση εξαρτημάτων διαύλου παρόμοιων επιδόσεων. Μπορείτε να τις σκεφτείτε όπως το διαχωρισμό ενός δρόμου σε πολλαπλές λωρίδες, για να υποστηρίζεται η ταχεία αλλά και η αργή κυκλοφορία. Ωστόσο, ο γέφυρες διαύλων δεν επιλύουν το ζήτημα των παράλληλων μεταφορών διαύλου. Οι γέφυρες διαύλου εισάγουν μια σιωπηρή ιεραρχία μεταξύ των τμημάτων του διαύλου: μια γέφυρα διαύλου είναι αφέντης από τη μία πλευρά και σκλάβος από την άλλη. Σε πολλές περιπτώσεις, για παράδειγμα σε αρχιτεκτονικές πολυεπεξεργαστών, μια ιεραρχία μεταξύ των επεξεργαστών δεν είναι επιθυμητή ή είναι ακόμα και αντιπαραγωγική.

Σημείωση: Τεχνολογικά ζητήματα τμημάτων διαύλου

Εκτός από τους λογικούς περιορισμούς των τμημάτων διαύλου, υπάρχουν επίσης σημαντικά τεχνολογικά ζητήματα. Η υλοποίηση πολύ μακριών καλωδίων σε ένα ολοκληρωμένο είναι δύσκολη και η διανομή σημάτων και χρονισμών υψηλής συχνότητας με τη χρήση τέτοιων καλωδίων είναι ακόμη πιο δύσκολη. Η κατανάλωση ισχύος ενός καλωδίου είναι ανάλογη προς το μήκος του αγωγού και τη συχνότητα μεταγωγής των σημάτων στον συγκεκριμένο αγωγό. Επομένως, τα μεγάλα καλώδια καταναλώνουν σημαντικά περισσότερη ισχύ από τα μικρά, τοπικά καλώδια. Τα ολοκληρωμένα που οργανώνονται με τη χρήση μιας γραμμικής τοπολογίας διαύλου καταναλώνουν περισσότερη ενέργεια για το ίδιο έργο από τα ολοκληρωμένα που οργανώνονται χρησιμοποιώντας μια κατανεμημένη τοπολογία.

Σημείωση: Γραμμική τοπολογία διαύλων έναντι κατανεμημένης τοπολογίας

Σημείωση: Για ποιο λόγο υπάρχει εξατομίκευση

Είναι σαφές ότι η κατασκευή επικοινωνιών διαύλου εντός-ολοκληρωμένου με τμηματοποιημένους διαύλους έχει τους περιορισμούς της. Η εξατομίκευση της τοπολογίας διαύλου και η αντιστοίχισή της με την εφαρμογή, αποτελούν μια λογική βελτίωση σε αυτό.

10.1.13 Μεταγωγείς Διαύλου

Σημείωση: Πρόβλημα στατικής συσχέτισης μεταξύ αφεντών διαύλου

Ένα τμήμα διαύλου δημιουργεί μια στατική συσχέτιση μεταξύ των αφεντών διαύλου και των σκλάβων διαύλου που είναι προσαρτημένοι σε αυτό το τμήμα. Ας δούμε γιατί μια στατική ανάθεση των αφεντών διαύλου σε τμήματα διαύλου μπορεί να είναι πρόβλημα. Ας υποθέσουμε ότι ένας αφέντης συνδέεται με το τμήμα A του διαύλου. Όλοι οι σκλάβοι που πρέπει να επικοινωνήσουν με αυτόν τον αφέντη θα χρειαστεί να συνδεθούν και στο A ή εναλλακτικά σε ένα τμήμα που είναι άμεσα γεφυρωμένο από το A. Επιπλέον, όλοι οι αφέντες που πρέπει να μιλήσουν σε οποιοσδήποτε από τους σκλάβους που συνδέονται με το A ή οποιοδήποτε τμήμα γεφυρωμένο σε αυτό, πρέπει επίσης να συνδεθούν με το τμήμα διαύλου A. Επομένως, όλοι οι αφέντες και όλοι οι σκλάβοι στο σύστημα θα τείνουν να συσσωρεύονται στο ίδιο τμήμα διαύλου A. Σε απλά συστήματα ενός αφέντη, αυτό

δεν αποτελεί πρόβλημα. Σε ετερογενείς αρχιτεκτονικές ή αρχιτεκτονικές με πολλαπλούς πυρήνες, ωστόσο, ο μονός τμηματοποιημένος δίαυλος γίνεται γρήγορα σημείο συμφόρησης.

Σημείωση: Χρήση μεταγωγής / διαστρωμάτωσης διαύλου

Η λύση αυτού του προβλήματος είναι η χρήση μεταγωγής διαύλου (bus switching) ή διαστρωμάτωσης διαύλου (bus layering), η οποία καθιστά ευέλικτη τη συσχέτιση μεταξύ αφεντών και σκλάβων. Η Εικόνα 104 είναι ένα παράδειγμα ενός δίαυλου δύο επιπέδων (layer). Οι δύο αφέντες διαύλου συνδέονται ο καθένας με ένα μεταγωγέα διαύλου (bus switch) με δύο τμήματα εξόδου. Υπάρχουν τρεις συνδέσεις σκλάβων στα τμήματα εξόδου. Οι αφέντες μπορούν να συνδεθούν με οποιοδήποτε από τους σκλάβους στα τμήματα εξόδου. Μια μεταφορά αφέντη αρχικά αποκωδικοποιείται, για να αποφασιστεί ποιο τμήμα εξόδου θα πρέπει να την λάβει. Στην συνέχεια, όλα τα αιτήματα για το ίδιο τμήμα εξόδου συγχωνεύονται και υποβάλλονται σε διαιτησία. Οι ταυτόχρονες μεταφορές στο ίδιο τμήμα εξόδου θα γίνουν ακολουθιακές (sequentialized): ο αφέντης στο τμήμα εισόδου δεν θα μπορέσει να ολοκληρώσει τη μεταφορά μέχρι ο αφέντης στο άλλο τμήμα εισόδου να έχει τελειώσει.

Οι πολλαπλοί αφέντες εξακολουθούν να μοιράζονται το ίδιο τμήμα εισόδου σε ένα μεταγωγέα διαύλου, ο οποίος καθιστά τη μεταγωγή (switching) διαύλου συμβατή με τη διαιτησία διαύλου πολλαπλών-αφεντών. Συστήματα διαύλου που έχουν υλοποιήσει μεταγωγή διαύλου είναι για παράδειγμα τα AMBA/AHB και μια βελτιωμένη έκδοση των Avalon, Merlin.

Σημείωση: Υλοποίηση έμμεσου πλέγματος

Η πιο εξελιγμένη μορφή του μεταγωγέα διαύλου είναι μια υλοποίηση στην οποία κάθε αφέντης έχει το δικό του τμήμα εισόδου στον μεταγωγέα και κάθε σκλάβος έχει το δικό του τμήμα εξόδου. Μια τέτοια υλοποίηση ονομάζεται έμμεσου –πλέγματος (cross-bar). Το έμμεσο-πλέγμα είναι μια εξαιρετικά παράλληλη, αλλά πολύ δαπανηρή υλοποίηση διασύνδεσης εντός-ολοκληρωμένου. Και, ενώ αντιμετωπίζει τα λογικά όρια των τμημάτων διαύλου, δεν αντιμετωπίζει τα ηλεκτρικά ζητήματά τους. Ένα έμμεσο-πλέγμα είναι ένα καθολικό σύστημα διασύνδεσης και δεν έχει δυνατότητα κλιμάκωσης.

10.1.14 Δίκτυο σε Ολοκληρωμένο

Σημείωση: Μειονέκτημα μεταγωγέων διαύλου

Οι μεταγωγείς διαύλου υποστηρίζουν μια δυναμική συσχέτιση αφεντών με σκλάβους, αλλά έχουν περιορισμένη δυνατότητα κλιμάκωσης (scalability). Το θεμελιώδες ζήτημα με τους μεταγωγείς διαύλου είναι ότι διατηρούν μια στενή συσχέτιση μεταξύ ενός αφέντη και ενός σκλάβου. Πράγματι, κάθε συναλλαγή διαύλου γίνεται απευθείας μεταξύ ενός αφέντη και ενός σκλάβου, και η υλοποίηση μιας συναλλαγής διαύλου απαιτεί από το δίκτυο διασύνδεσης να δημιουργήσει μια διαδρομή για να ολοκληρωθεί. Τα πρώτα τηλεφωνικά συστήματα χρησιμοποιούσαν την ίδια ιδέα: για να συνδέσουν έναν καλούντα με έναν καλούμενο, δημιουργούνταν μόνιμο κύκλωμα μεταξύ των δύο μερών για τη διάρκεια της τηλεφωνικής κλήσης τους.

Σημείωση: Μειονέκτημα Εικόνας 104

Σημείωση: Δίκτυο σε ολοκληρωμένο

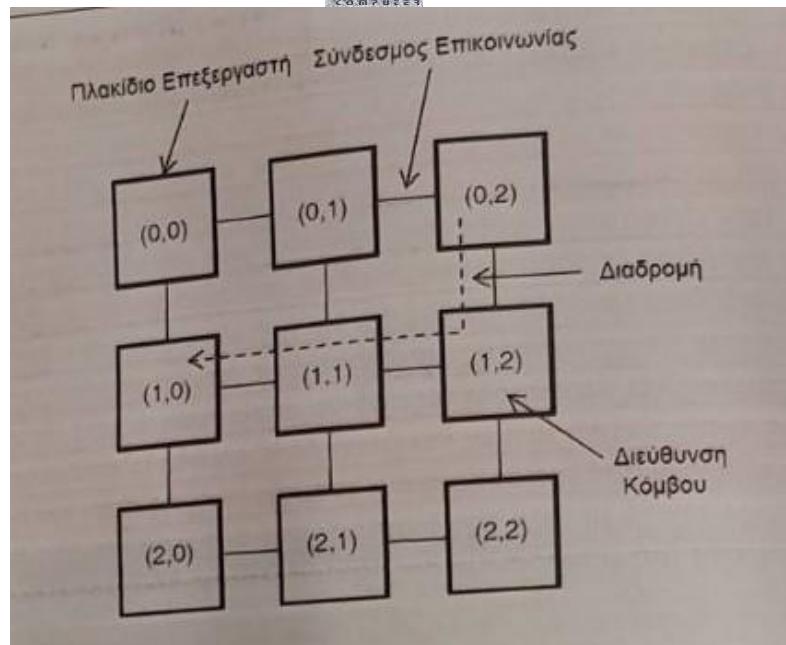
Ωστόσο, η έννοια μιας σταθερής και μόνιμης διαδρομής δεν κλιμακώνεται. Αντίθετα, η ίδια η μεταφορά διαύλου πρέπει να υλοποιείται δυναμικά, σε διάφορα στάδια. Πρώτον, ένας αφέντης συσκευάζει ένα αίτημα σε ένα πακέτο. Ο αφέντης παραδίδει το πακέτο στη διασύνδεση και η διασύνδεση βρίσκει μια διαδρομή από τον αφέντη στον σκλάβο. Τέλος, ο σκλάβος αποδέχεται το πακέτο από τη διασύνδεση. Όταν ο σκλάβος είναι έτοιμος να εκδώσει μια απόκριση, το αντίθετο θα συμβεί: ο σκλάβος παραδίδει ένα πακέτο στη διασύνδεση, η διασύνδεση επιστρέφει το πακέτο στον αφέντη και ο αφέντης δέχεται το πακέτο απόκρισης από τη διασύνδεση. Παρατηρήστε την αλλαγή της ορολογίας από τη συναλλαγή σε αίτημα/απόκριση και πακέτο. Το παράδειγμα επικοινωνίας έχει αλλάξει από συναλλαγές σε αποστολή και λήψη πακέτων. Αυτό το νέο παράδειγμα επικοινωνίας ονομάζεται Δίκτυο σε Ολοκληρωμένο (Network-on-Chip).

Σημείωση: Περιγραφή Εικόνας 105 → Χρήση διαδρομής

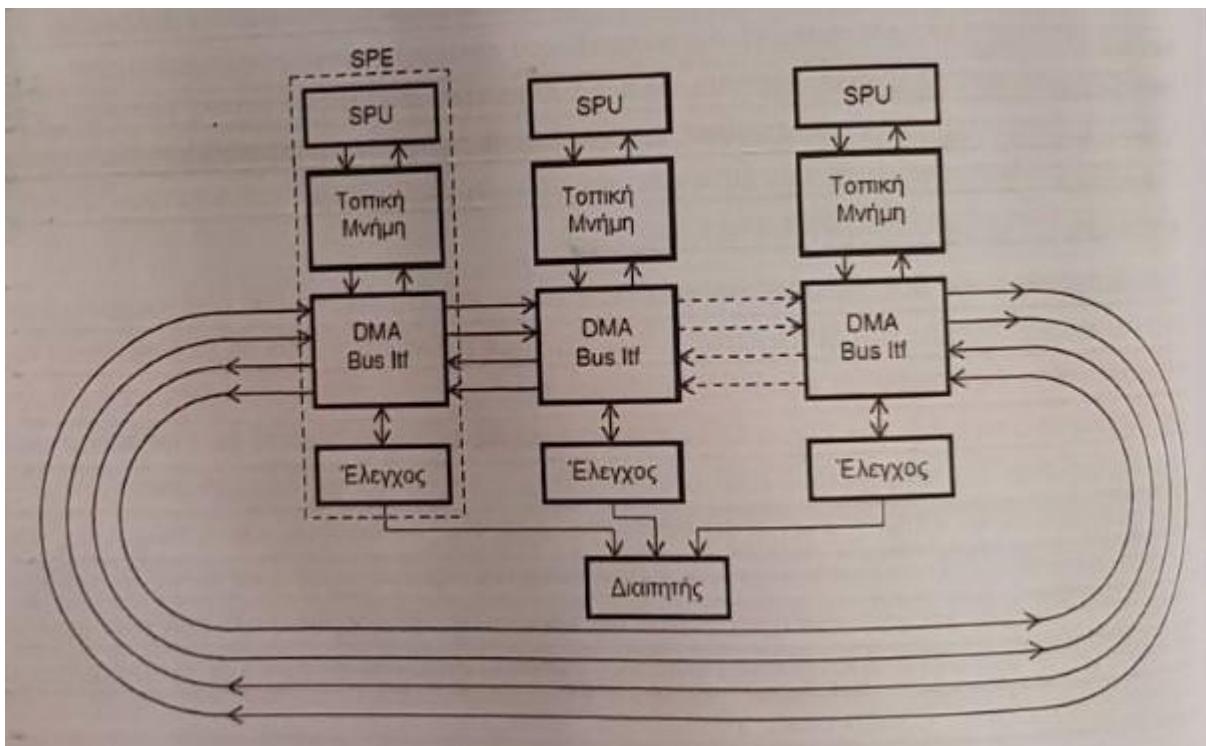
Η Εικόνα 105 παρουσιάζει την έννοια του Δικτύου σε Ολοκληρωμένο. Τα υπολογιστικά στοιχεία του ολοκληρωμένου και οι διασυνδέσεις τους είναι οργανωμένα με κάποιο γεωμετρικό μοτίβο, όπως ο πίνακας ή ο δακτύλιος. Οι διασυνδέσεις γίνονται μεταξύ των πλακιδίων (tiles) του δικτύου σε ολοκληρωμένο. Στην Εικόνα 105, κάθε πλακίδιο μπορεί να επικοινωνήσει απευθείας με τα γειτονικά πλακίδια. Επιπλέον, κάθε πλακίδιο έχει μια διεύθυνση, που δεικτοδοτείται συμβολικά από τους δείκτες πίνακα. Αυτό επιτρέπει σε οποιοδήποτε πλακίδιο να επικοινωνεί με οποιοδήποτε άλλο πλακίδιο. Επιλέγεται μια διαδρομή (route) για την επικοινωνία, και ένα πακέτο δεδομένων μετακινείται μέσω ενός αριθμού αλμάτων (hops), από ένα πλακίδιο αφετηρίας σε ένα πλακίδιο προορισμού πάνω από έναν αριθμό ενδιάμεσων πλακιδίων. Η Εικόνα 105 απεικονίζει μια διαδρομή από το πλακίδιο (0,2) στο πλακίδιο (1,0).

Σημείωση: Σημασία αλγόριθμου δρομολόγησης → στην Εικόνα 105

Ο σχεδιασμός ενός δικτύου σε ολοκληρωμένου και η λειτουργία του, εισάγει ένα σύνολο απαιτητικών προβλημάτων. Στο βασικό επίπεδο, η επικοινωνία και η αναπαράσταση δεδομένων είναι πολύ διαφορετική από την προσέγγιση που χρησιμοποιείται στους διαύλους εντός-ολοκληρωμένου. Σε ένα δίκτυο-σε-ολοκληρωμένο, τα στοιχεία δεδομένων ενθυλακώνονται σε ένα πακέτο (packet) πριν μεταδοθούν. Μόλις ένα πακέτο εγκαταλείψει ένα πλακίδιο αφετηρίας και ταξιδέψει σε ένα πλακίδιο προορισμού, πρέπει να βρει μια διαδρομή. Όπως φαίνεται στην Εικόνα 105, μια διαδρομή δεν είναι μοναδική. Μεταξύ ενός πλακιδίου αφετηρίας και ενός πλακιδίου προορισμού υπάρχουν πολλές δυνατές διαφορετικές διαδρομές. Επομένως, χρειάζεται ένας αλγόριθμος δρομολόγησης ο οποίος θα είναι σε θέση να επιλέξει τμήματα διασύνδεσης έτσι ώστε το συνολικό επίπεδο συμφόρησης να παραμείνει μικρό. Ο σχεδιασμός του Δικτύου σε Ολοκληρωμένο υπήρξε τομέας εντατικής έρευνας κατά την τελευταία δεκαετία.



Εικόνα 106 - Ένα γενικό δίκτυο σε ολοκληρωμένο



Εικόνα 107 - Δίκτυο σε ολοκληρωμένο στον επεξεργαστή CELL

Σημείωση: Χρήση Επεξεργαστή CELL

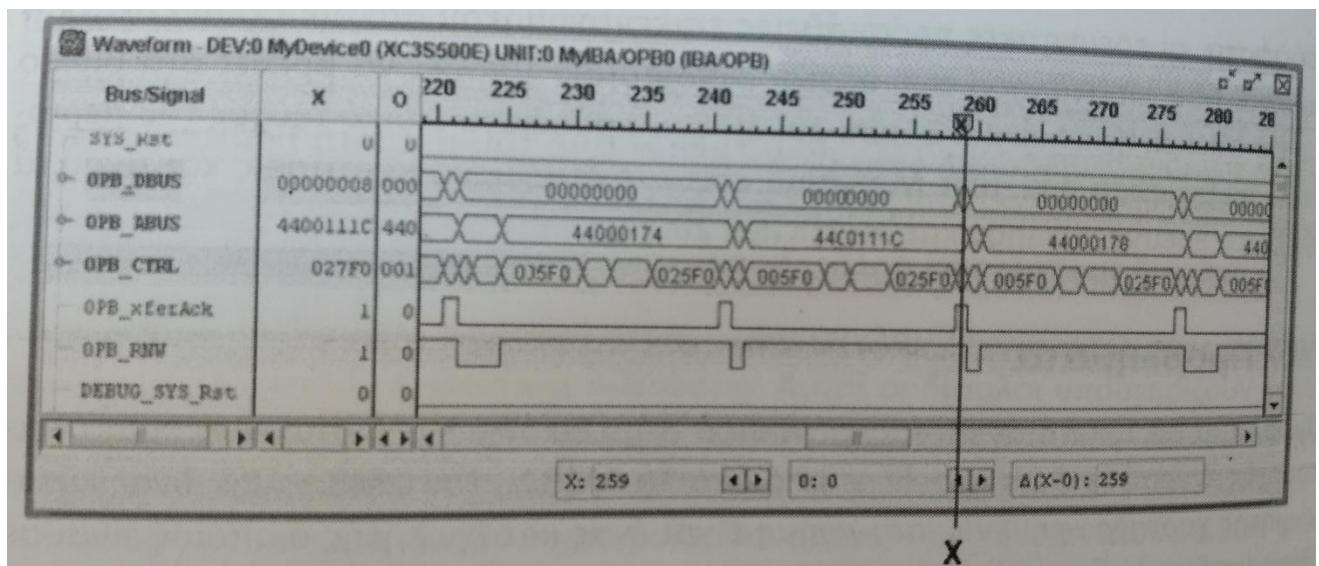
Σημείωση: Περιγραφή Εικόνας 106

Ο επεξεργαστής CELL είναι μια πολύ γνωστή συσκευή πολυεπεξεργαστή που βασίζεται στην τεχνολογία δικτύου σε ολοκληρωμένο για την υλοποίηση επικοινωνιών σε-ολοκληρωμένο. Ο CELL συνδυάζει οκτώ κανονικά εξαρτήματα επεξεργασίας που ονομάζονται SPE (Synergistic Processing Element). Επιπλέον, υπάρχει ένας επεξεργαστής ελέγχου που ονομάζεται PPE (Power Processor Element) και μια μονάδα διασύνδεσης εκτός ολοκληρωμένου. Όλα αυτά τα εξαρτήματα συνδέονται στο ίδιο δίκτυο στο ολοκληρωμένο, που ονομάζεται EIB (Element Interconnect Bus). Όπως απεικονίζεται στην Εικόνα 106, το EIB αποτελείται από τέσσερις δομές δακτυλίου, καθεμία με εύρος 16 byte. Το μοντέλο επικοινωνίας των επεξεργαστών CELL προϋποθέτει ότι οι

επεξεργαστές θα λειτουργήσουν χρησιμοποιώντας την τοπική μνήμη και ότι η επικοινωνία υλοποιείται μετακινώντας μπλοκ δεδομένων από μια τοποθεσία τοπικής μνήμης στην άλλη. Για τη διαχείριση της επικοινωνίας μεταξύ της διασύνδεσης διαύλου και της τοπικής μνήμη χρησιμοποιείται μια μονάδα Αμεσης Προσπέλασης Μνήμης (Direct Memory Access – DMA).

Οι τέσσερις δακτύλιοι λειτουργούν σε αντίθετες κατευθύνσεις, έτσι ώστε κάθε SPE να μπορεί να μιλήσει απευθείας με τους γείτονές του. Η επικοινωνία με άλλα SPE είναι δυνατή κάνοντας αρκετά άλματα πάνω από το δίαυλο επικοινωνίας. Κάθε φορά που ένα SPE θέλει να διαβιβάσει ένα μπλοκ δεδομένων μέσω EIB, θα στείλει ένα κατάλληλο αίτημα στον κεντρικό διαιτητή εντός-ολοκληρωμένου. Ο διαιτητής θα χρονοδρομολογήσει όλα τα εκκρεμή αιτήματα στους τέσσερις δακτυλίους. Έως και τρεις μεταφορές μπορούν να χρονοδρομολογηθούν ταυτόχρονα σε κάθε δακτύλιο, υπό τον όρο ότι οι μεταφορές αυτές χρησιμοποιούν διαφορετικά τμήματα του δακτυλίου.

Προβλήματα



Εικόνα 108 - Διάγραμμα χρονισμού για το Πρόβλημα 10.1

Λίστα 10.1 Πρόγραμμα για το Πρόβλημα 10.1

```
#include <stdio.h>
void main () {
    int i, a[0x40];
    for (i = 0; i< 0x40; i++)
        if (i > 0x23)
            a[i] = a[i-1] + 1;
        else
            a[i] = 0x5;
}
```

Πρόβλημα 10.1 Ενώ κάνετε αποσφαλμάτωση ενός προγράμματος C σε έναν μικροεπεξεργαστή 32-bit (που εμφανίζεται στη Λίστα 10.1), αποτυπώνετε τη μεταφορά διαύλου που φαίνεται στην Εικόνα 107. Ο μικροεπεξεργαστής είναι συνδεδεμένος σε μια μνήμη εκτός ολοκληρωμένου που κρατά το πρόγραμμα και τα δεδομένα. Ο κώδικας και το τμήμα δεδομένων αποθηκεύονται σε μνήμη εκτός ολοκληρωμένου ξεκινώντας από τη

διεύθυνση 0x44000000. Ο πίνακας α [] ξεκινά στη διεύθυνση 0x44001084. Οι εντολές από το σώμα του βρόχου ξεκινούν από τη διεύθυνση 0x44000170. Παρατηρήστε προσεκτικά το διάγραμμα χρονισμού στην Εικόνα 10.18 και απαντήστε στις παρακάτω ερωτήσεις.

(α) Ο δρομέας X στην Εικόνα 107 τοποθετείται σε ένα σημείο για το οποίο ο δίαυλος διευθύνσεων περιέχει 0x4400111C και ο δίαυλος δεδομένων περιέχει 0x8. Είναι αυτό μια ανάγνωση ή μια εγγραφή μνήμης;

(β) Για την ίδια θέση δρομέα 'X', είναι προσπέλαση μνήμης που αφορά ανάκτηση-εντολής ή ανάγνωση μνήμης-δεδομένων;

(γ) Για την ίδια θέση δρομέα 'X', ποια είναι η τιμή του μετρητή βρόχου i από το πρόγραμμα C;

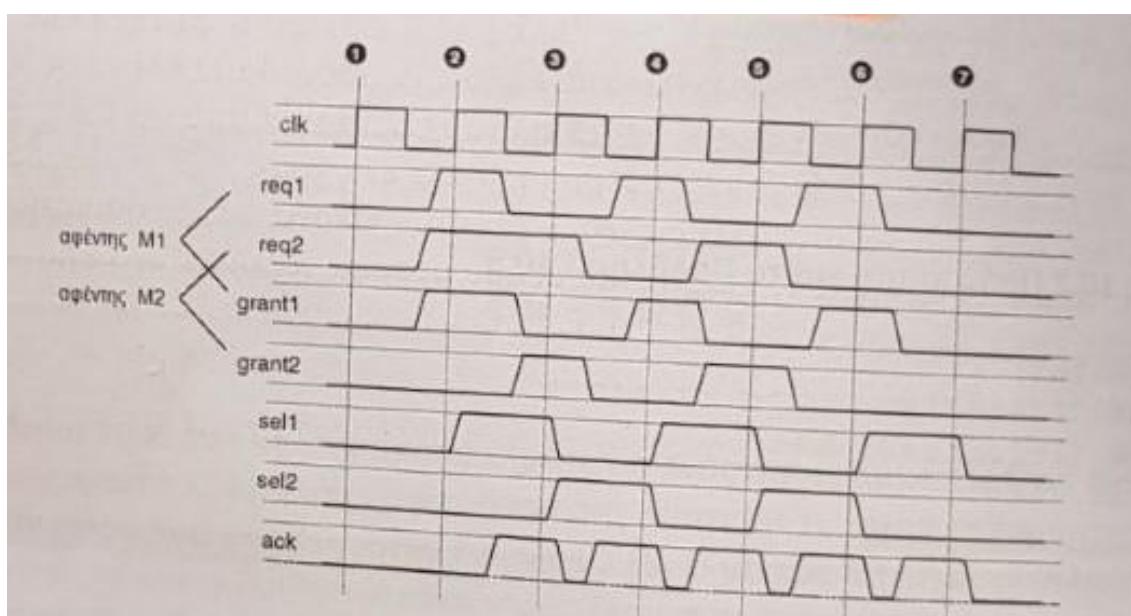
Πρόβλημα 10.2 Το διάγραμμα χρονισμού στην Εικόνα 108 δείχνει τη διαδικασία διαιτησίας δύο αφεντών, M1 και M2, που ζητούν πρόσβαση σε ένα κοινόχρηστο δίαυλο. Απαντήστε στις παρακάτω ερωτήσεις χρησιμοποιώντας τις πληροφορίες που παρέχονται στο διάγραμμα χρονισμού.

(α) Με βάση το διάγραμμα χρονισμού, ποιος αφέντης έχει την υψηλότερη προτεραιότητα για μεταφορές διαύλου: ο M1, ο M2 ή είναι αδύνατο να πούμε;

(β) Ποιος αφέντης έχει τον έλεγχο του διαύλου διευθύνσεων μεταξύ της ακμής ρολογιού 3 και της ακμής ρολογιού 4: ο M1, ο M2 ή είναι αδύνατο να πούμε;

(γ) Ποιος τύπος εξαρτήματος καθορίζει την τιμή των σημάτων qrantx: ένας αφέντης διαύλου, ένας διαιτητής διαύλου ή ένας σκλάβος διαύλου;

(δ) Ποιος τύπος εξαρτήματος καθορίζει την τιμή των σημάτων ack: ένας αφέντης διαύλου, ένας διαιτητής διαύλου ή ένας σκλάβος διαύλου;

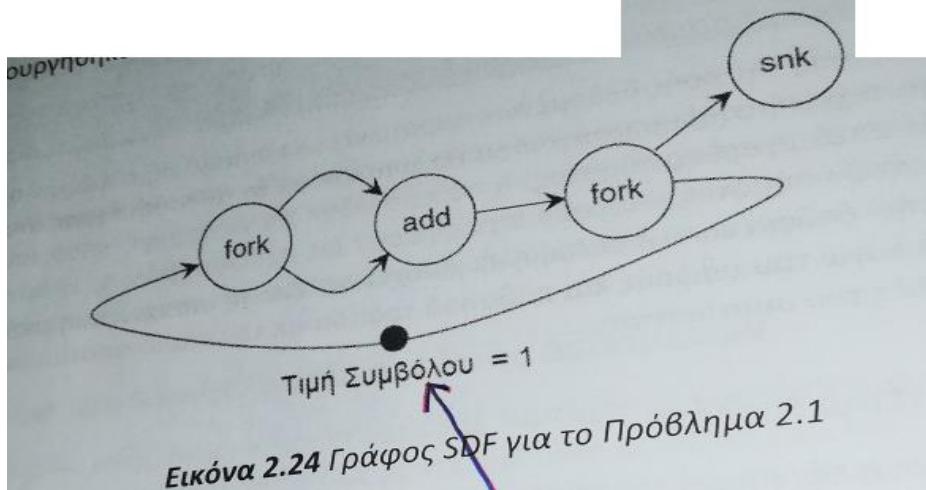


Εικόνα 109 - Διάγραμμα χρονισμού για το Πρόβλημα 10.2

11 Άσκηση 2.1

2.8 Προβλήματα

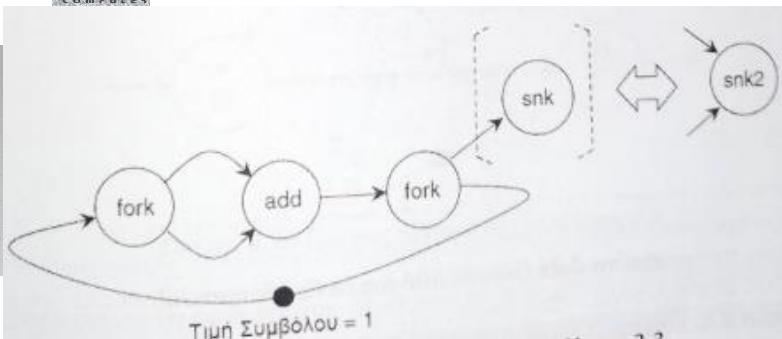
Πρόβλημα 2.1. Εξετάστε τον μονού ρυθμού γράφο SDF στην Εικόνα 2.24. Ο υπόφος περιέχει τρεις τύπους δρώντων. Ο δρων **fork** διαβάζει ένα σύμβολο και παράγει δύο αντίγραφα του συμβόλου εισόδου, ένα σε κάθε έξοδο. Ο δρων **add** προσθέτει δύο σύμβολα, παράγοντας ένα μοναδικό σύμβολο που φέρει το άθροισμα των συμβόλων εισόδου. Ο δρων **snk** είναι ένας αποδέκτης-συμβόλων (signal-sink) που καταγράφει την ακολουθία των συμβόλων που εμφανίζονται στην είσοδό του. Ένα μοναδικό αρχικό σύμβολο, με τιμή 1, τοποθετείται σε αυτόν το γράφο. Βρείτε την τιμή των συμβόλων που παράγονται στον δρώντα **snk**. Βρείτε έναν συμβολισμό συντομογραφίας για αυτήν την ακολουθία αριθμών.



Λύση

Παρατηρούμε από την Εικόνα 2.24 ότι εισέρχεται ένα σύμβολο (token) στο δρώντα **fork** και αυτό διασπάται σε δύο σύμβολα, ένα για την κάθε ουρά που φεύγει από το δρώντα **fork**. Στη συνέχεια, αυτά τα δύο σύμβολα αθροίζονται στο δρώντα **add**. Με δεδομένο ότι το αρχικό σύμβολο έχει τιμή «1», ο δρώντας **fork** θα το διασπάσει σε δύο σύμβολα τιμής «1» το καθένα και ο δρώντας **add** θα τα αθροίσει στη συνέχεια σε τιμή «2», η οποία και θα εμφανιστεί στο δρώντα **snk**, ενώ παράλληλα θα επιστρέψει πίσω. Τη δεύτερη φορά, ο δρώντας **fork**, θα διασπάσει το σύμβολο με τιμή «2» σε δύο σύμβολα με τιμή «2» και ο δρώντας **add** θα τα αθροίσει σε τιμή «4» κ.ο.κ. Επομένως η ακολουθία αριθμών που παράγεται είναι: $2^0, 2^1, 2^2, 2^3, \dots, 2^n$

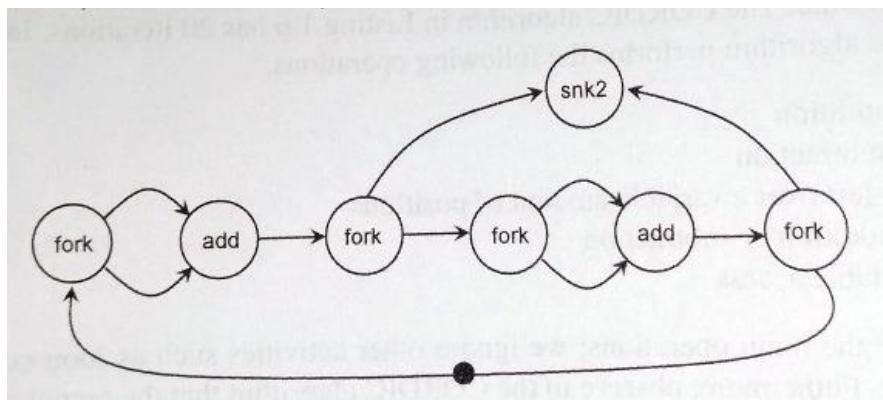
Πρόβλημα 2.3. Εξετάστε το γράφο SDF στην Εικόνα 2.25. Μετατρέψτε αυτόν το γράφο έτσι ώστε να παράγει την ίδια ακολουθία συμβόλων ως πλειάδες αντί ως δρώντα snk με τον snk2, έτσι ώστε να παράγετε αυτή την ακολουθία διπλού-ρυθμού στον snk2.



12 Άσκηση 2.3

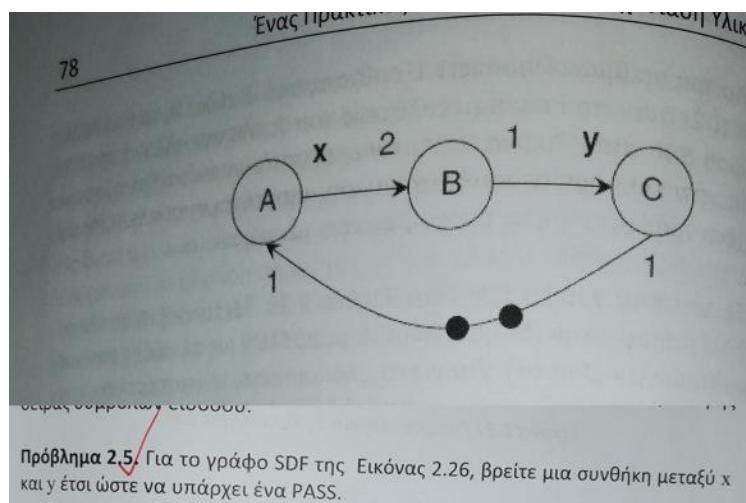
Λύση

Και πάλι εξετάζουμε τον προηγούμενο γράφο. Θέλουμε να τον μετατρέψουμε έτσι ώστε να παράγει την ίδια ακολουθία συμβόλων ως πλειάδες και όχι ως ακολουθία μονών τιμών. Κάθε δρώντας που έχει το όνομα "fork", παίρνει μία είσοδο και τη διασπά σε δύο εξόδους.



Μια επανάληψη δύο αντιγράφων του αρχικού συστήματος ροής δεδομένων θα παράγει τα δύο σύμβολα (tokens) που μπορούν να τροφοδοτηθούν στο δρώντα (actor) με όνομα sink2. Το αρχικό σύμβολο από τον αρχικό γράφο **δεν θα πρέπει να διπλασιαστεί** (αντιγραφεί), διότι κάτι τέτοιο θα τροποποιούσε την ακολουθία τιμών που παρατηρήθηκαν στο snk2. Ο τελευταίος δρώντας απαιτεί **δύο σύμβολα σε δύο διαφορετικές εισόδους του, για να πυροδοτήσει**.

13 Άσκηση 2.5



Λύση

Πάντα ένας ρυθμός παραγωγής είναι θετικός και ένας ρυθμός κατανάλωσης είναι αρνητικός. Για να υπάρχει ένα PASS πρέπει η τάξη του πίνακα να είναι πάντα κατά «1» μικρότερη από το πλήθος των κόμβων (δρώντων). Σχηματίζουμε από το δοθέν SDF τον πίνακα τοπολογίας G που φαίνεται στη συνέχεια. Έχει τόσες γραμμές όσες είναι οι ακμές του γράφου (ουρές) και τόσες στήλες όσοι είναι οι κόμβοι του γράφου (δρώντες).

$$G = \begin{bmatrix} X & -2 & 0 \\ 0 & 1 & -Y \\ -1 & 0 & 1 \end{bmatrix} \rightarrow \begin{array}{l} \text{ακμή AB} \\ \text{ακμή BC} \\ \text{ακμή AC} \end{array}$$

Μεθοδολογία: Πρέπει η στήλη με τους σταθερούς όρους να παραμένει αμετάβλητη και να καταλήγουμε σε αυτή από γρ. συνδυασμούς των υπολοίπων στηλών. Στην προκειμένη περίπτωση για $X = -2$ και $Y = -1$, η δεύτερη στήλη προκύπτει από άθροισμα πρώτης και τρίτης στήλης. Άρα η τάξη του πίνακα = $2 < 3$ (πλήθος κόμβων).

$$\begin{bmatrix} X & -2 & 0 \\ 0 & 1 & -Y \\ -1 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} -2 & -2 & 0 \\ 0 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Αν στη συνέχεια ζητηθεί ένα διάνυσμα πυροδότησης q_{pass} , αυτό θα μπορούσε να είναι το εξής:

$$\begin{bmatrix} -2 & -2 & 0 \\ 0 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

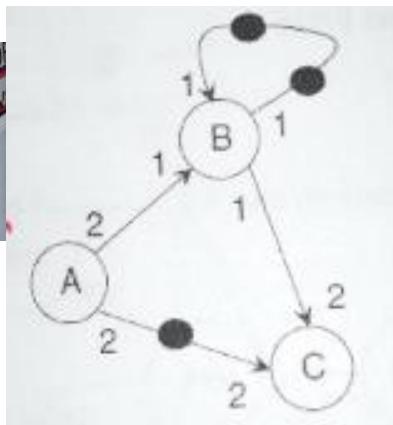
Παρατήρηση: επειδή φυσικά, δεν υπάρχει αρνητική πυροδότηση ενός δρώντα, για τις συγκεκριμένες X, Y

$$\text{θα θέσουμε ως } q_{\text{pass}} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

14 Άσκηση 2.7

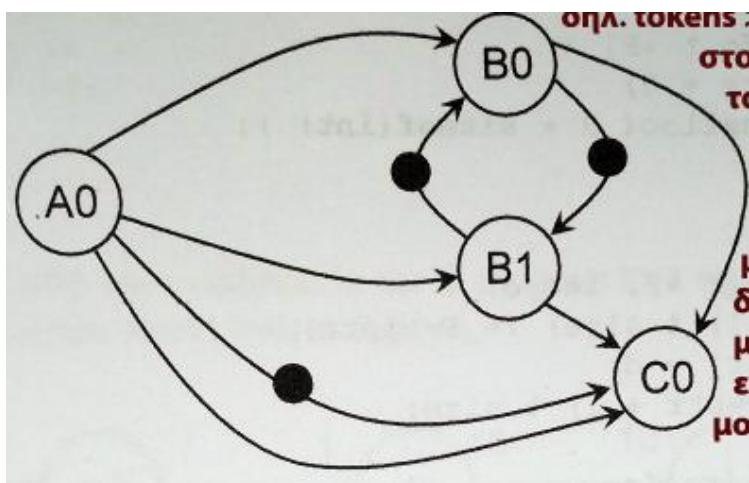
Πρόβλημα 2.7. Σχεδιάστε την επέκταση πολλαπλών ρυθμών για το SDF πλών ρυθμών που δίνεται στην Εικόνα 2.28. Μην ξεχάσετε να αναδιανομέτων αρχικά σύμβολα στο αποτέλεσμα της πολλαπλού-ρυθμού επέκτασης.

Λύση



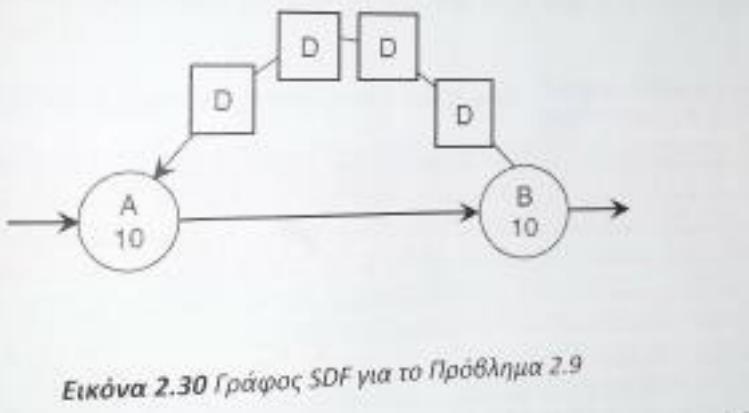
Ακολουθούμε τα βήματα της θεωρίας:

1. Διευκρινίζεται ότι ο πολλαπλός ρυθμός δεν έχει να κάνει με το ρυθμό παραγωγής ή κατανάλωσης. Προσδιορίζουμε τους ρυθμούς πυροδότησης PASS, οι οποίοι δίνονται κάτω από κάθε δρώντα. Στην προκειμένη περίπτωση, επειδή **δεν** δίνονται οι ρυθμοί πυροδότησης PASS, όπως δινόντουσαν στην Εικόνα 2.19, θα θεωρήσουμε ότι ο μόνος δρώντας που έχει **πολλαπλό ρυθμό είναι ο B**, ο οποίος έχει δύο σύμβολα πάνω στην ουρά που ενώνει την έξοδό του με την είσοδό του.
2. Αντιγράφουμε κάθε δρώντα τόσες φορές, όσες υποδεικνύονται από τον αριθμό πυροδότησής του PASS. Στην προκειμένη περίπτωση, αντιγράφουμε μόνο τον B δύο φορές, σε B0 και B1.
3. **Μετατρέπουμε κάθε είσοδο/έξοδο μόνο του δρώντα πολλαπλών ρυθμών (B) σε πολλαπλές εισόδους/εξόδους μονού ρυθμού.** Επειδή η είσοδος του δρώντα B έχει τιμή «1» (ρυθμός κατανάλωσης), συνδέουμε το δρώντα A, με κάθε αντίγραφο B0, B1 με ένα βέλος. Η έξοδος του δρώντα B, συνδέεται μόνο με το δρώντα C με ρυθμό παραγωγής «1». Για το λόγο αυτό, συνδέουμε κάθε αντίγραφο B0, B1 με ένα βέλος (μία ουρά) με το δρώντα C.
4. Εισάγουμε ξανά τις ουρές (βέλη) στο SDF και τις συνδέουμε με τους δρώντες.
5. Εισάγουμε ξανά τα αρχικά σύμβολα στο σύστημα (•) και τα κατανέμουμε ακολουθιακά πάνω στις ουρές μονού – ρυθμού. Υπάρχουν δύο σύμβολα που κινούνται μεταξύ B0 και B1 και ένα σύμβολο μεταξύ A και C.



15 Άσκηση 2.9

περισσότερο, το όριο επανάληψης.
Πρόβλημα 2.9 Να απλώσετε το γράφο της Εικόνας 2.30, τρεις φορές. Προσδιορίστε το όριο επανάληψης, πριν και μετά τη διαδικασία απλώματος.

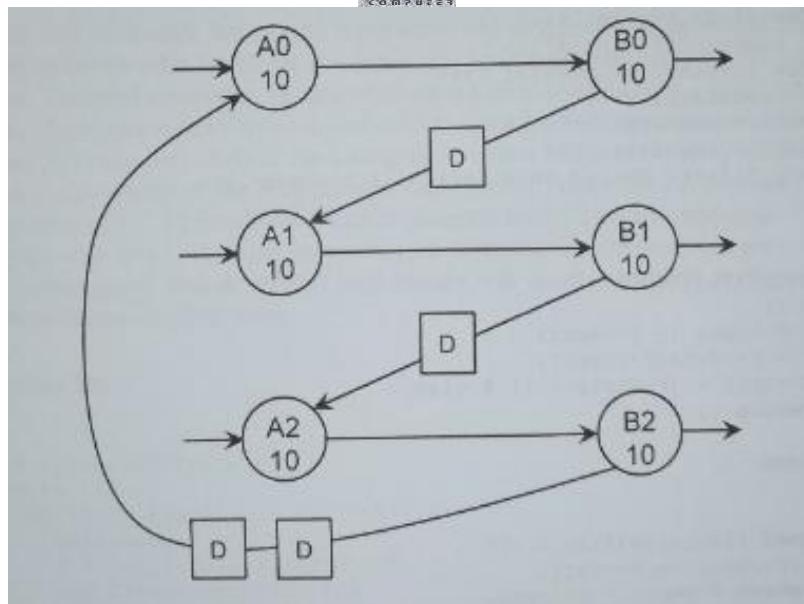


Εικόνα 2.30 Γράφος SDF για το Προβλήμα 2.9

Λύση

Το n – άπλωμα ενός γράφου SDF, δημιουργεί αντίστοιχα n – εισόδους και n – εξόδους. **Στην προκειμένη περίπτωση, επειδή η εκφόνηση αναφέρει τρία απλώματα, σημαίνει ότι η παράμετρος $n = 3$, οπότε κάθε δρώντας του SDF θα επαναλαμβάνεται τόσες φορές, όσες υποδεικνύεται από το άπλωμα, επομένως τρεις φορές στην προκειμένη περίπτωση.** Για το λόγο αυτό ο αρχικός δρώντας A υλοποιείται σε τρία αντίγραφα, A0, A1 και A2 και κάτι ανάλογο ισχύει και για το δρώντα B. Στη συνέχεια, θα πρέπει οι καθυστερήσεις D να ανακατανεμηθούν πάνω στις συνδέσεις, με τέτοιο τρόπο ώστε να υπάρχει παραλληλία για αύξηση της απόδοσης. Αυτό επιτυγχάνεται στην προκειμένη περίπτωση από την αύξηση του ορίου επανάληψης από 5 στο αρχικό SDF σε 15 στο τελικό SDF. Το όριο επανάληψης πριν το άπλωμα, δίνεται από τον τύπο: συνολικό άθροισμα καθυστερήσεων δρώντων (λανθανόντων χρόνων δρώντων) = $\frac{(10+10)}{4} = 5$. Μετά το άπλωμα, το όριο επανάληψης γίνεται: $\frac{10+10+10+10+10}{4} = 60/4 = 15$.

Έτσι, μετά το άπλωμα οι δρώντες θα πυροδοτούνται κάθε 15 χρ. μονάδες, ενώ πριν το άπλωμα ενεργοποιούνται κάθε 5 χρονικές μονάδες. Αυτό σημαίνει ότι μετά το άπλωμα υπάρχει μεγαλύτερη επάρκεια στα δεδομένα εισόδου κάθε δρώντα. Επομένως, ο γράφος SDF έχει μεγαλύτερο χρονικό διάστημα αυτόνομης λειτουργίας, αφού δεν χρειάζεται να επαναλαμβάνει τη συμπεριφορά του κάθε 5, αλλά πλέον κάθε 15 χρονικές μονάδες (nsec). Το άπλωμα των SDF στοχεύει στο να επεξεργαστεί ροές δεδομένων με πολύ υψηλούς ρυθμούς δειγματοληψίας.



Έχουμε $n = 4$ καθυστερήσεις και $v = 3$ απλώματα. Το $i = 0, 1, 2, \dots, v - 1 = 0, 1, 2$

Για $i = 0 \rightarrow \kappa = (i + n) \% v = (0 + 4) \% 3 = 1$

Για $i = 1 \rightarrow \kappa = (i + n) \% v = (1 + 4) \% 3 = 2$

Για $i = 2 \rightarrow \kappa = (i + n) \% v = (2 + 4) \% 3 = 0$

Με βάση τις παραπάνω σχέσεις, υπάρχουν οι ακμές $B_0 - A_1$, $B_1 - A_2$, $B_2 - A_0$. Οι τιμές της μεταβλητής i προσδιορίζουν ακμές των δρώντων B_i και οι τιμές κ προσδιορίζουν ακμές των δρώντων A_κ .

Αναφορικά με τις καθυστερήσεις D που μεταφέρει η ακμή AB_i , αυτές προκύπτουν από τον τύπο $\left\lfloor \frac{i+n}{v} \right\rfloor$ και

για την ακμή $B_0 - A_1$ είναι $\left\lfloor \frac{0+4}{3} \right\rfloor = \left\lfloor \frac{4}{3} \right\rfloor = \left\lfloor 1.333 \right\rfloor = 1$, ενώ για την ακμή $B_1 - A_2$ είναι $\left\lfloor \frac{1+4}{3} \right\rfloor = \left\lfloor \frac{5}{3} \right\rfloor = \left\lfloor 1.666 \right\rfloor = 1$ και για την ακμή $B_2 - A_0$ είναι $\left\lfloor \frac{2+4}{3} \right\rfloor = \left\lfloor \frac{6}{3} \right\rfloor = \left\lfloor 2 \right\rfloor = 2$.

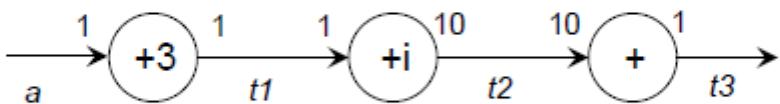
16 Άσκηση 3.3

Λύση

Let's first write a small C program and construct an equivalent SDF graph. Here is a C program with a single input and a single output and a loop.

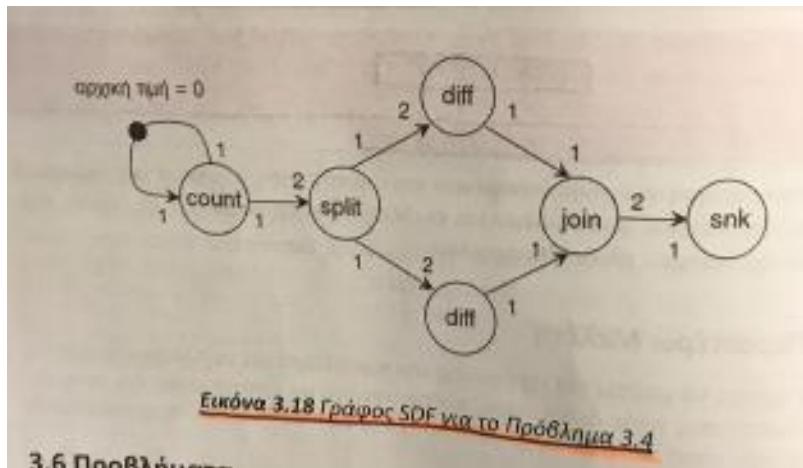
```
unsigned myfunc(unsigned a) {
    unsigned i, j;
    unsigned t1, t2, t3 = 0;
    t1 = a + 3;
    for (i=0; i<10; i++) {
        t2 = t1 + i;
        t3 = t3 + t2;
    }
    return t3;
}
```

Assuming we map each addition in the C program to a separate actor, the SDF version of the program would look as follows,

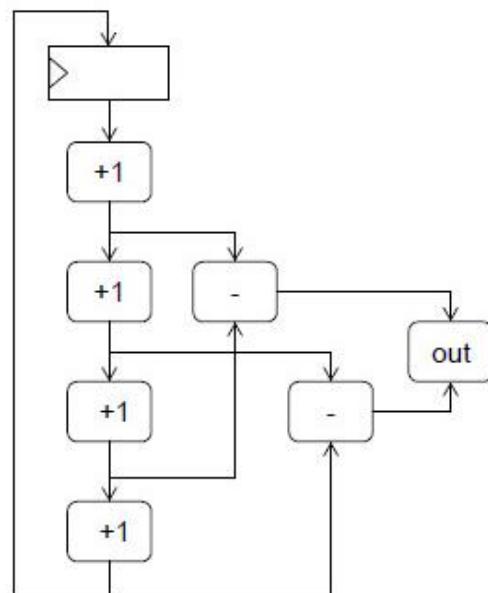
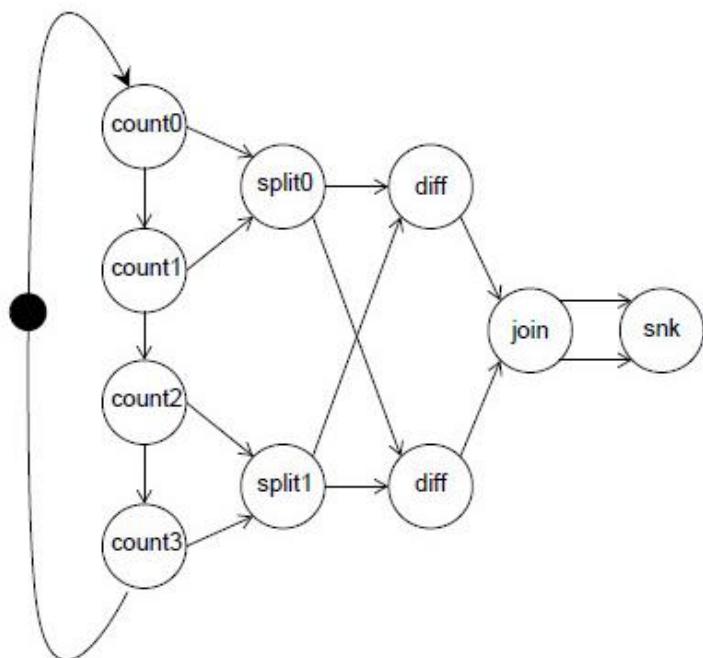


The first actor adds three to the input token and produces a single output token; the second actor adds the numbers 0 to 9 to the input token and produces 10 output tokens. The third actor takes the sum of 10 input tokens and produces a single output token. The loop bound (10) thus translates into actor production/consumption rates. Indeed, the inner loop body in the C program executes 10 times for every execution of the code outside of the loop. Hence, there are 10 values for t_2 produced for every value of t_1 . The multi-rate dataflow diagram thus is a natural rendering of a C program with loops. The loop bounds appear as production/consumption rates, and as a consequence, data-dependent loop bounds will result in variable or unknown production/consumption rates.

17 Άσκηση 3.7



Λύση



Για την επέκταση πολλαπλών ρυθμών, λαμβάνουμε υπόψη τους ρυθμούς κατανάλωσης στην είσοδο των δρώντων εκείνων που έχουν ρυθμό κατανάλωσης > 1 .

This circuit does not have a particular purpose or function; its only use is to demonstrate the conversion of a multi-rate data-flow graph into hardware.

Στην απεικόνιση του γράφου SDF σε υλικό (αφού πρώτα εκτελέσουμε μια επέκταση πολλαπλών ρυθμών), απεικονίζουμε κάθε ουρά επικοινωνίας σε ένα καλώδιο και συγκεκριμένα κάθε ουρά που περιέχει ένα σύμβολο (token) σε καταχωρητή. Επειδή έχουμε μόνο ένα σύμβολο στο αρχικό SDF γράφημα, υπάρχει ένας μόνο καταχωρητής στην υλοποίηση του SDF σε υλικό.

18 Άσκηση 4.1

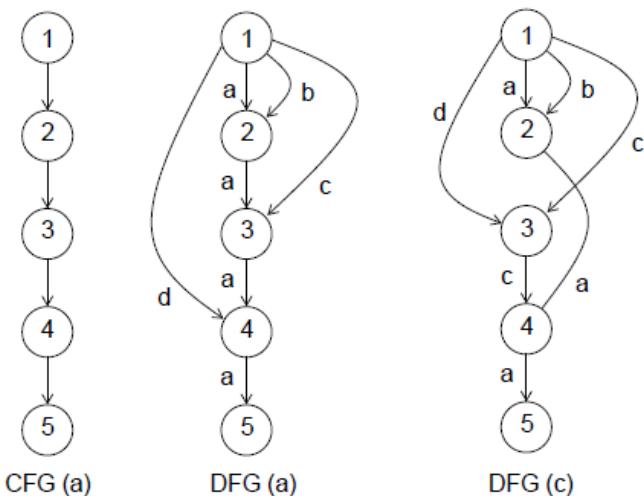
Λίστα 4.1
 1 int addall(int a, int b, int c, int d) {
 2 a = a + b;
 3 a = a + c;
 4 a = a + d;
 5 return a;
 }

$$a = a + b + c + d$$

- (a) Εξάγετε και σχεδιάστε τους CFG και DFG.
 (b) Το μήκος ενός μονοπατιού σε ένα γράφο ορίζεται ως ο αριθμός των ακμών σε αυτό το μονοπάτι. Βρείτε το μακρύτερο μονοπάτι στο DFG.
 (c) Ξαναγράψετε το πρόγραμμα στη Λίστα 4.1 έτσι ώστε να μειώνεται το μέγιστρο μήκος μονοπατιού στο DFG. Υποθέστε ότι μπορείτε να εκτελέσετε μόνο μία αριθμητική λειτουργία ανά εντολή C. Σχεδιάστε το DFG που προκύπτει.

Λύση

Στο σχεδιασμό του CFG έχει σημασία η σειρά ακολουθίας εντολών και στο σχεδιασμό του DFG έχει σημασία η ανταλλαγή δεδομένων μεταξύ των κόμβων (λειτουργιών). Παρόλα αυτά η αρίθμηση των κόμβων είναι κοινή και για τους δύο γράφους ροής δεδομένων. Για το λόγο αυτό η αρχική αρίθμηση των λειτουργιών στον κώδικα, μεταφέρεται και στο CFG και στο DFG. Στο δεύτερο γράφημα από αυτά που ακολουθούν, η σύνδεση μεταξύ των λειτουργιών «2» και «3» γίνεται μόνο με την ενημερωμένη μεταβλητά «a», όπως επίσης και μεταξύ των λειτουργιών «3» και «4».



- (a) Refer to the Figure.
 (b) The longest path is 4.
 (c) The function can be rewritten as shown below. The resulting DFG is shown in the Figure. The longest path in the optimized DFG is 3.

κ) Για τη βελτίωση (μείωση) του μήκους του μεγαλύτερου μονοπατιού, σκεφτόμαστε ως εξής: Η τελική τιμή που επιστρέφεται από τη συνάρτηση είναι:

$$\begin{aligned} a &= a + b \\ &\swarrow \quad \searrow \\ a &= a + c = a + b + c \end{aligned}$$



$$a = \mathbf{a} + d = \mathbf{a} + \mathbf{b} + \mathbf{c} + d$$

Προσπαθούμε να πετύχουμε το ίδιο αποτέλεσμα, αλλάζοντας τη σειρά των αναθέσεων. Παρατηρώντας το βελτιωμένο DFG, διαπιστώνουμε τα εξής:

Στη λειτουργία (κόμβο) 2 εκτελείται η πράξη $\mathbf{a} = \mathbf{a} + \mathbf{b}$, στη λειτουργία 3 η πράξη $\mathbf{c} = \mathbf{c} + \mathbf{d}$ και στη λειτουργία

4 η πράξη $\mathbf{a} + \mathbf{c} = \mathbf{a} + \mathbf{b} + \mathbf{c} + \mathbf{d}$. Αυτή η τροποποίηση φαίνεται με τον κώδικα που ακολουθεί:

```
1. int addall (int a, int b, int c, int d) {  
2.     a = a + b;  
3.     c = c + d;  
4.     a = a + c;  
5.     return a  
}
```

Με τον τρόπο αυτό πετυχαίνεται η μείωση του μέγιστου μήκους του μονοπατιού του DFG από 4 σε μήκος 3.

Παρατήρηση: Θα μπορούσαμε θεωρητικά να εκτελέσουμε όλες τις πράξεις σε μια εντολή, δηλ. $a = a + b + c + d$ για να πετύχουμε στο DFG το μικρότερο μονοπάτι. Αυτό όμως δεν μπορεί να γίνει, διότι στην εκφώνηση αναφέρεται ότι είναι δυνατή η εκτέλεση μόνο μιας αριθμητικής λειτουργίας ανά εντολή C.

19 Άσκηση 4.2

Λύση

Πρόβλημα 4.2 ✓ Σχεδιάστε τους CFG και DFG του προγράμματος στη Λίστα 4.2.
Συμπεριλάβετε όλες τις εξαρτήσεις ελέγχου στο CFG. Συμπεριλάβετε τις εξαρτήσεις δεδομένων για τις μεταβλητές a και b του DFG.

Λίστα 4.2 Πρόγραμμα για το Πρόβλημα 4.2

```

L  int count(int a, int b) {
  2   while (a < b)
      a = a * 2;
  3   return a + b;
}
  
```

CFG

```

graph TD
    1((1)) --> 2((2))
    2 --> 3((3))
    3 --> 2
    3 --> 4((4))
    4 --> 1
  
```

DFG

```

graph TD
    1((1)) -- "a,b" --> 2((2))
    2 -- "a,b" --> 3((3))
    3 -- "a" --> 2
    3 -- "a" --> 4((4))
    4 -- "a,b" --> 1
  
```

Θα πρέπει να αριθμήσουμε τις λειτουργίες του κώδικα, όπως φαίνεται. Στο CFG δίνουμε έμφαση στη σειρά (ακολουθία) εκτέλεσης των εντολών, ενώ στο DFG δίνουμε έμφαση στα δεδομένα που ανταλλάσσονται μεταξύ των εντολών.

20 Άσκηση 4.3

Πρόβλημα 4.3. ✓ Σχεδιάστε μια διαδρομή δεδομένων στο υλικό για το πρόγραμμα που εμφανίζεται στη Λίστα 4.3. Εκχωρήστε καταχωρητές και τελεστές. Υποδείξτε τις εισόδους ελέγχου που απαιτούνται από τη διαδρομή δεδομένων και τις σημαίες συγθηκών που παράγονται από τη διαδρομή δεδομένων.

```
unsigned char mysqrt (unsigned int N) {  
    unsigned int x, j;  
    x = 0;  
    for (j = 1<<7; j!=0; j>>=1) {  
        x = x + j  
        if (x * x > N)  
            x = x - j;  
    }  
    return x;  
}
```

Σημείωση: Η εκτέλεση του for θα γίνει για τιμές j = 128, 64, 32, 16, 8, 4, 2, 1

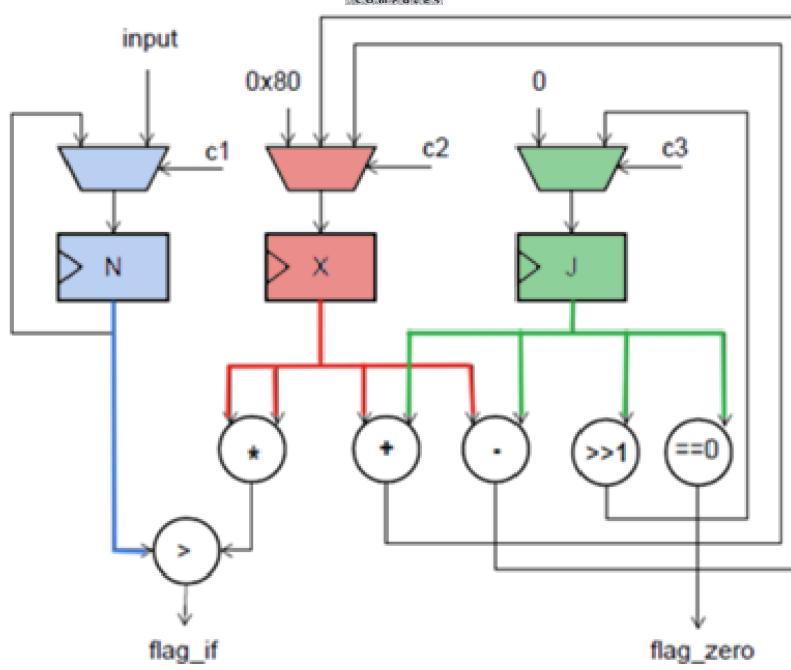
Λύση

Γνωρίζουμε ότι η υλοποίηση ενός αλγόριθμου (σχεδίαση μιας διαδρομής δεδομένων) σε υλικό πρέπει να πληροί τις εξής προϋποθέσεις:

- **Κάθε μεταβλητή στον κώδικα C → πολυπλέκτη και καταχωρητή.** Στην προκειμένη περίπτωση, οι τρείς μεταβλητές N, x, j που εμφανίζονται μέσα στο πρόγραμμα πρέπει να υλοποιηθούν με καταχωρητή και αντίστοιχο πολυπλέκτη η κάθε μια. Για τη μεταβλητή N, εμφανίζεται στο κύκλωμα μια είσοδος input, η οποία είναι η παράμετρος που δίνεται ως είσοδος στη function. Στη μεταβλητή j, επειδή γίνεται σύγκριση με το «0», υπάρχει στον πολυπλέκτη η είσοδος «0», και στον πολυπλέκτη της μεταβλητής x έχει εισαχθεί ως αρχική τιμή το $0x80 = 1000\ 0000 = 2^7 = 128$ (αρχική τιμή της επανάληψης).

- **Κάθε πράξη στον κώδικα C → συνδυαστικό κύκλωμα υλοποίησης αυτής της πράξης** (στην προκειμένη περίπτωση αφαιρέτης και αθροιστής και πολλαπλασιαστής και ολισθητής). Κάθε πολυπλέκτης, από τους τρεις συνολικά, έχει μια γραμμή επιλογής c1, c2, c3.

- **Κάθε λειτουργία καταχώρησης → με ακμή δεδομένων που συνδέει καταχωρητή με το συνδυαστικό κύκλωμα.** Στην προκειμένη περίπτωση **κάθε ακμή δεδομένων αναπαρίσταται με ένα αντίστοιχο χρώμα.** Η μεταβλητή N συμμετέχει μόνο στη σύγκριση $x * x > N$, και για το λόγο αυτό εμφανίζεται μόνο στο συγκριτή. Με ανάλογο τρόπο και οι υπόλοιπες γραμμές συνδέονται με εκείνα τα συνδυαστικά κυκλώματα στα οποία συμμετέχουν, π.χ. η μεταβλητή X εμφανίζεται στις πράξεις $x = x + j$, $x * x > N$, $x = x - j$ κ.ο.κ. Μάλιστα στον πολλαπλασιαστή πηγαίνει δύο φορές, για να δημιουργηθεί το γινόμενο $x * x$ στη σύγκριση $x * x > N$.

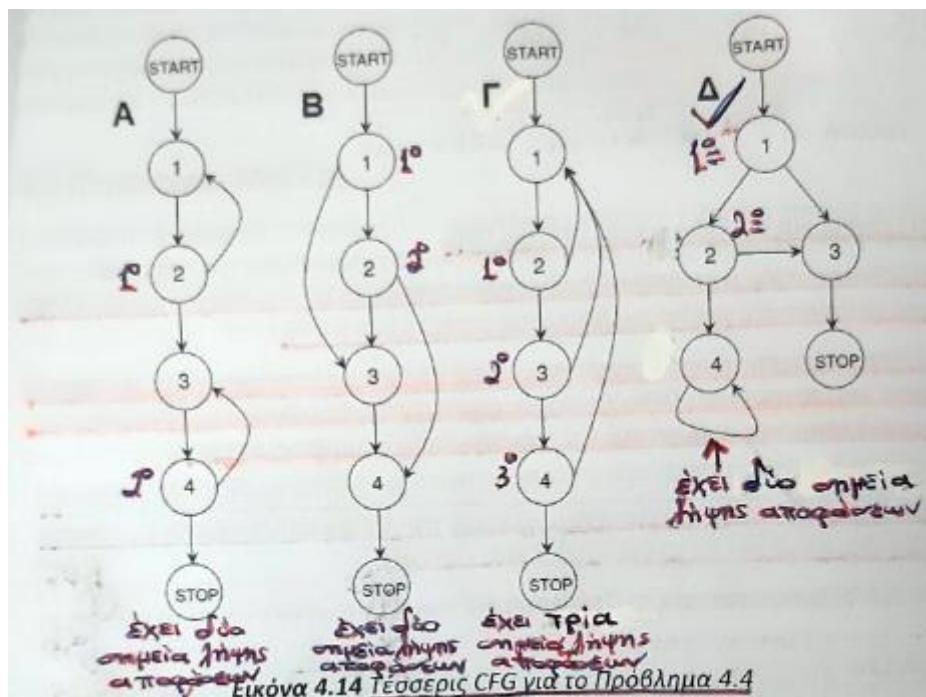


21 Άσκηση 4.4

Πρόβλημα 4.4. Ένα καλά δομημένο πρόγραμμα C είναι ένα πρόγραμμα που περιέχει μόνο τις ακόλουθες εντολές ελέγχου: if-then-else, while, do-while, και for. Εξετάστε τους τέσσερις CFG στην Εικόνα 4.14. Ποιος από αυτούς τους CFG αντιστοιχεί σε ένα καλά δομημένο πρόγραμμα C; Σημειώστε ότι ένας μεμονωμένος κόλυβος στο CFG μπορεί να περιέχει περισσότερες από μία εντολές, αλλά δεν θα μπορεί ποτέ να περιέχει περισσότερα από ένα σημεία λήψης αποφάσεων.

Λύση

Κάθε ένα CFG από αυτά τα τέσσερα, αντιστοιχεί σε κάποιο δομημένο κώδικα C. Το πρώτο CFG περιέχει 2 σημεία λήψης αποφάσεων, που σημειώνονται με τις τιμές «1^o» και «2^o», το δεύτερο CFG περιέχει επίσης 2 σημεία λήψης αποφάσεων, που σημειώνονται με τις τιμές «1^o» και «2^o», το τρίτο CFG περιέχει 3 σημεία λήψης αποφάσεων, που σημειώνονται με τις τιμές «1^o», «2^o» και «3^o» αντίστοιχα, ενώ το τέταρτο CFG περιέχει 2 σημεία λήψης αποφάσεων, που σημειώνονται με τις τιμές «1^o» και «2^o». Από αυτά τα τέσσερα CFG αυτό που αντιστοιχεί σε ένα καλά δομημένο πρόγραμμα C είναι το τρίτο, διότι περιέχει περισσότερα σημεία λήψης αποφάσεων.



22 Άσκηση 4.5

Πρόβλημα 4.5. Σχεδιάστε τον DFG για το πρόγραμμα στη Λίστα 4.4. Υποθέστε ότι όλα τα στοιχεία του πίνακα $a[]$ θα αποθηκευτούν σε έναν μοναδικό πόρο.

Λίστα 4.4 Πρόγραμμα για το Πρόβλημα 4.5

```

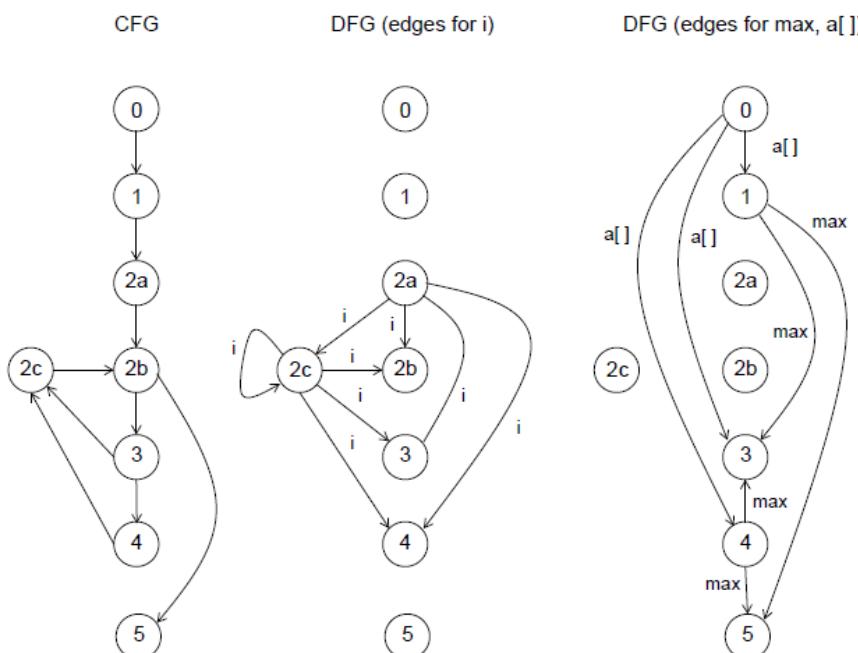
0 int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int findmax() {
    int max, i;
    1 max = a[0];
    2 for (i=1; i<10; i++)
        2a 2b 2c
            3 if (max < a[i])
                4 max = a[i];
    5 return max;
}

```

Λύση

Γνωρίζουμε ότι όταν αριθμούμε τις εντολές ενός κώδικα σε C προκειμένου να σχεδιάσουμε στη συνέχεια τους γράφους CFG και DFG, **δεν αριθμούμε τις δηλώσεις των μεταβλητών ως λειτουργίες, όταν στις μεταβλητές αυτές δεν εκχωρείται κάποιο αρχικό περιεχόμενο**.

Αυτό σημαίνει ότι στον κώδικα που δίνεται, δεν αριθμούμε τις εντολές δήλωσης των μεταβλητών max και i καθώς και τη δήλωση της συνάρτησης int findmax(). Πάνω στον κώδικα αναγράφονται οι εντολές όπως τις έχουμε αριθμήσει. Παρότι η εκφώνηση δεν ζητά CFG, απαραίτητη προϋπόθεση για να σχεδιαστεί το DFG είναι η προγενέστερη σχεδίαση του CFG. Στο μεσαίο DFG δεν υπάρχει κάποια ακμή μεταξύ των λειτουργιών «0» και «1», επειδή στη λειτουργία «0» που είναι η δήλωση της συνάρτησης δεν υπάρχουν καθόλου μεταβλητές, με αποτέλεσμα να μην δίνεται κανένα δεδομένο στη λειτουργία «1» από τη λειτουργία «0». Ομοίως δεν υπάρχει ανταλλαγή δεδομένων μεταξύ των λειτουργιών «1» και «2». Υπάρχουν δύο γράφοι DFG: ο πρώτος περιγράφει την ανταλλαγή της μεταβλητής i μεταξύ των λειτουργιών και ο δεύτερος περιγράφει την ανταλλαγή των μεταβλητών max και a[] μεταξύ των λειτουργιών.





Η δημιουργία του DFG έχει γίνει με το δεύτερο τρόπο όπως αυτός περιγράφεται στη σελίδα 128 της θεωρίας. Πιο συγκεκριμένα, διαχωρίζεται το DFG σε δύο επιμέρους, όπου στο πρώτο DFG αναφέρονται μόνο οι ακμές που περιέχουν το δείκτη i , ενώ στο δεύτερο αναφέρονται μόνο οι ακμές που αφορούν τον πίνακα και το μέγιστο στοιχείο του max .



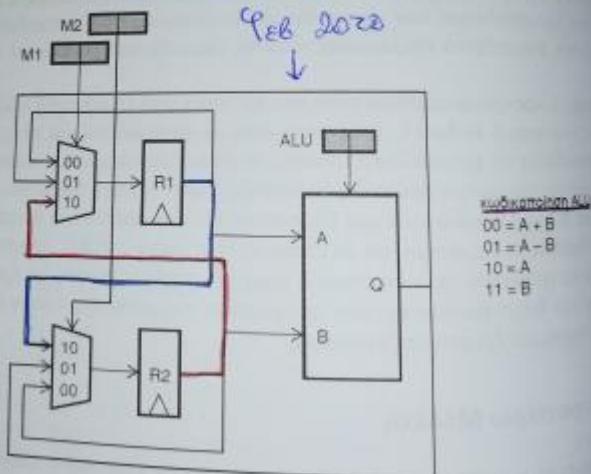
23 Άσκηση 4.7

Λύση

```
unsigned char mysqrt(unsigned int N) {
    unsigned int x1, x2, x3, j1, j2;
    x1 = 0;
    for (j1 = 1<<7; merge(j1, j2) != 0; j2 = merge(j1, j2) >> 1) {
        x2 = merge(x1, x2, x3) + merge(j1, j2);
        if (x2 * x2 > N)
            x3 = x2 - merge(j1, j2);
    }
    return merge(x2, x3);
}
```

24 Άσκηση 6.1

Πρόβλημα 6.1 / Η Εικόνα 6.15 δείχνει μια μικροπρογραμματιζόμενη διάδοση δεδομένων. Υπάρχουν έξι bit ελέγχου για τη διαδρομή δεδομένων: 2 bits για την παραλλαγή από τους πολυπλέκτες M1 και M2 και 2 bits για την ALU. Η κωδικοποίηση των bit ελέγχου υποδεικνύεται στην εικόνα.



Πίνακας 6.5 Μικρο-εντολές για το Πρόβλημα 6.1

Αντιτιθέστε το περιεχόμενο των R1 και R2.

Προσθέστε τα προιεγόμενα των R1 και R2 και αποθηκεύστε τα περιεχόμενα στον Rx, που είναι ίσος με R1 ή R2. Υπάρχουν δύο παραλλαγές της εντολής εξαρτώμενες από τον Rx.

Αντιγράψτε τα περιεχόμενα του Ry στον Rx. Το (Rx, Ry) είναι ίσος (R1, R2).

Μην κάνεις τίποτα.

- (a) Αναπτύξτε μια οριζόντια κωδικοποίηση μικρο-εντολής για τη λίστα των μικροεντολών που δίνονται στον Πίνακα 6.5.
- (β) Αναπτύξτε μιας κατακόρυφη κωδικοποίηση μικρο-εντολών για την ίδια λίστα εντολών. Χρησιμοποιήστε μια ικανοποιητική κωδικοποίηση που καταλλύει σε ένα συμπαγή και αποδοτικό αποκωδικοποιητή για τη διαδρομή δεδομένων.

Λύση

Στις εντολές SWAP, COPY R1, COPY R2, NOP δεν χρησιμοποιείται η ALU και γιατρι βάζουμε αδιάφορους όρους στα bits της ALU και επειδή δεν επιτρέπονται X, βάζουμε 0

17 Answers to Selected Exercises

Instruction	M1 M1 [1:0]	M2 M2 [1:0]	ALU ALU[1:0]	Horizontal H[5:0]	Vertical V[2:0]
SWAP	10	10	00	101000	001
ADD R1	01	00	00	010000	010
ADD R2	00	01	00	000100	011
COPY R1	00	10	00	001000	100
COPY R2	10	00	00	100000	101
NOP	00	00	00	000000	000

- The encoding of the horizontal micro-instruction corresponds to M1 M2 ALU.
- The encoding of the vertical micro-instruction implies an additional decoder that transforms the vertically encoded micro-operation into a horizontally encoded micro-operation. For example, the msbit of M1 can be obtained as $M1[1] = (\tilde{V}[2] \& \tilde{V}[1] \& V[0]) | (V[2] \& \tilde{V}[1] \& V[0])$.

Αντιγραφή του R1

Αντιγραφή του R2

A) Η **οριζόντια κωδικοποίηση** περιγράφεται στη στήλη "Horizontal"

B) Η **κατακόρυφη κωδικοποίηση** περιγράφεται στη στήλη "Vertical". Επειδή έχουμε συνολικά 6 εντολές (μαζί με τις παραλλαγές τους), αυτό σημαίνει ότι για την κατακόρυφη κωδικοποίηση χρειαζόμαστε x bits, όπου $2^x \geq 6 \Rightarrow x = 3$ bits. Μπορούμε να χρησιμοποιήσουμε μια αύξουσα αρίθμηση για την κατακόρυφη κωδικοποίηση επιλέγοντας την τιμή "000" για την εντολή NOP.

Σημείωση: Όσα bits δεν χρησιμοποιούνται στην οριζόντια κωδικοποίηση, επειδή δεν μπορούν να πάρουν την τιμή «0», παίρνουν την τιμή «X» (αδιάφορος όρος).



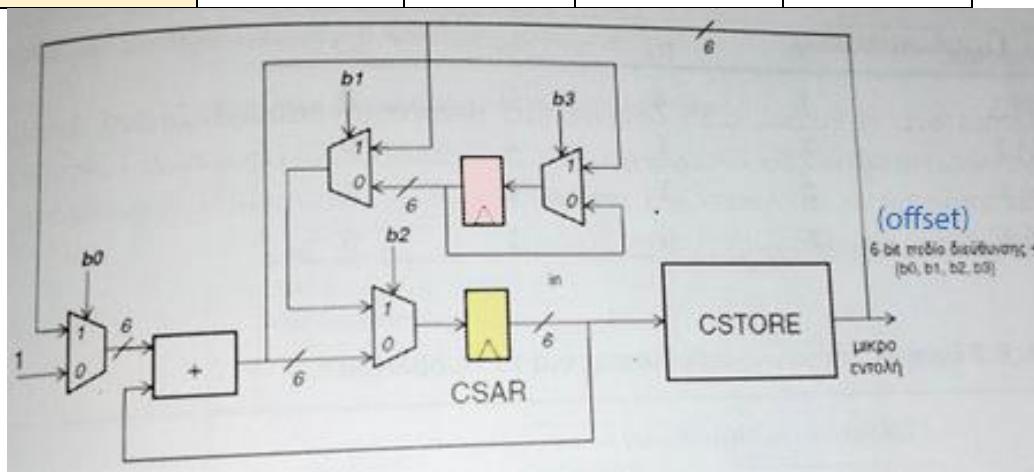
Μια ικανοποιητική κωδικοποίηση που να καταλήγει σε ένα συμπαγή και αποδοτικό αποκωδικοποιητή για τη διαδρομή δεδομένων είναι η **χρήση ενός επιπρόσθετου αποκωδικοποιητή** που να μετασχηματίζει την **κατακόρυφα κωδικοποιημένη μικρολειτουργία** σε μία οριζόντια κωδικοποιημένη μικρολειτουργία. Για παράδειγμα, το msbit του M1, δηλ. το M1[1], μπορεί να ληφθεί από τη σχέση **($\tilde{V}[2] \& \tilde{V}[1] \& V[0]$) | ($V[2] \& \tilde{V}[1] \& V[0]$)**, δηλ. $M1[1] = (\tilde{V}[2] \& \tilde{V}[1] \& V[0]) | (V[2] \& \tilde{V}[1] \& V[0])$. Αυτοί οι δύο συνδυασμοί παρέχουν τους «1» της στήλης M1[1] και πιο συγκεκριμένα, το πρώτο ζεύγος () παρέχει τον πρώτο «1» αυτής της στήλης, ενώ το δεύτερο ζεύγος () παρέχει τον δεύτερο «1» αυτής της στήλης. Με ανάλογο τρόπο μπορούμε να υπολογίσουμε το $M2[1] = (\tilde{V}[2] \& \tilde{V}[1] \& V[0]) | (V[2] \& \tilde{V}[1] \& \tilde{V}[0])$.

Σημείωση: Για χρήση decoder θα πρέπει η λογική συνάρτηση να εκφραστεί ως άθροισμα γινομένων.

25 Άσκηση 6.3

Η επόμενη εικόνα δείχνει την υλοποίηση ενός αποκωδικοποιητή επόμενης – διεύθυνσης. Συνολικά 10 bit από τη μικροεντολή χρησιμοποιούνται για τον έλεγχο της λογικής επόμενης – διεύθυνσης: ένα πεδίο διεύθυνση 6 – bit και τέσσερα bits ελέγχου: b0, b1, b2 και b3. Για καθένα από τους συνδυασμούς bit ελέγχου που φαίνονται στον παρακάτω πίνακα, βρείτε μια καλή περιγραφή της εντολής που αντιστοιχεί των δυαδικών ψηφίων ελέγχου που φαίνονται. Μην γράφετε γενικές περιγραφές (όπως ο καταχωρητής CSAR αυξάνεται κατά 1), αλλά δώστε μια υψηλού επιπέδου περιγραφή της εντολής που υλοποιείται. Χρησιμοποιήστε όρους που μπορεί να καταλάβει ένας προγραμματιστής λογισμικού.

Συνδυασμός	b0	b1	b2	b3
Εντολή 1	1	X	0	0
Εντολή 2	X	1	1	0
Εντολή 3	0	1	1	1
Εντολή 4	X	0	1	0



Από τους δύο καταχωρητές του σχήματος, ο κάτω είναι ο CSAR και ο πάνω είναι ο Return, ο οποίος, όταν χρησιμοποιείται, αποθηκεύει τη διεύθυνση επιστροφής από μια υπορουτίνα.

Λύση

Συνδυασμός	Bits b0, b1, b2, b3	Λειτουργία
Εντολή 1	1 x 0 0	Jump Relative (CSAR = CSAR + offset) Σχετικό άλμα (η τιμή του CSAR δεν αυξάνεται κατά «1», αλλά αυξάνεται κατά μια ποσότητα offset, και λαμβάνεται υπόψη η προηγούμενη τιμή του CSAR).
Εντολή 2	x 1 1 0	Jump Absolute (CSAR = offset) Απόλυτο άλμα (δεν λαμβάνεται υπόψη η προηγούμενη τιμή του CSAR, ο οποίος κάνει άλμα απευθείας στη διεύθυνση που υποδεικνύεται από το offset)
Εντολή 3	0 1 1 1	Call Subroutine (CSAR = offset, Return = CSAR + 1) Κλήση μιας υπορουτίνας, όπου στον CSAR, μεταβιβάζουμε τη μετατόπιση (offset) που πρέπει να κάνει για να μπει στην υπορουτίνα. Ταυτόχρονα αποθηκεύσουμε την τρέχουσα τιμή του αυξημένη κατά «1» στον πάνω καταχωρητή του κυκλώματος, ως διεύθυνση επιστροφής από την υπορουτίνα. Πάντα, όταν επιστρέφουμε από κάποια υπορουτίνα στο κύριο

πρόγραμμα, η επιστροφή αυτή γίνεται στην αμέσως επόμενη εντολή, από αυτή που κλήθηκε η υπορουτίνα.

Εντολή 4

x 0 1 0

Return from Subroutine (CSAR = return)

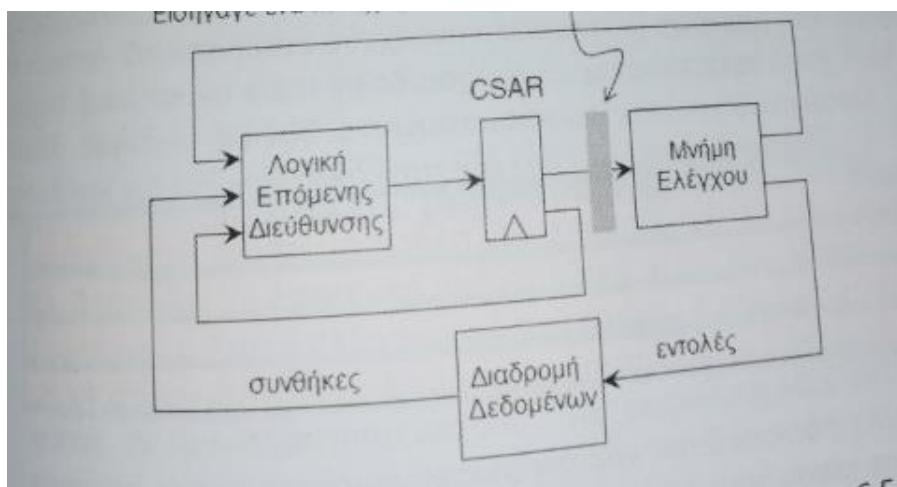
Επαναφέρουμε από τον πάνω καταχωρητή του κυκλωματος τη διεύθυνση επιστροφής που είχαμε νωρίτερα αποθηκεύσει και την αποθηκεύσουμε στον CSAR, προκειμένου να πραγματοποιηθεί επιστροφή από την υπορουτίνα στο κύριο πρόγραμμα.

26 Άσκηση 6.5

Ο συνάδελφός σας, σας ζητά να υπολογίσετε μια βελτίωση για μια μικροπρογραμματιζόμενη αρχιτεκτονική, όπως φαίνεται στην επόμενη Εικόνα. Η βελτίωση είναι να εισαγάγετε έναν καταχωρητή διοχέτευσης ακριβώς μπροστά από τη μνήμη ελέγχου.

α) Μειώνει αυτός ο πρόσθετος καταχωρητής το κρίσιμο μονοπάτι της συνολικής αρχιτεκτονικής;

Β) Ο συνάδελφός σας το αποκαλεί αρχιτεκτονική διπλού – νήματος και ισχυρίζεται ότι αυτή η βελτίωση επιτρέπει στη μηχανή μικρο – ελέγχου να εκτελεί δύο εντελώς ανεξάρτητα προγράμματα με εναλλασσόμενο τρόπο. Συμφωνείτε με αυτό ή όχι;



Λύση

Α) Η σημασία ενός καταχωρητή διοχέτευσης είναι να **προκαλεί επικάλυψη εντολών**, προκειμένου να αυξηθεί η απόδοση. Ο πρόσθετος καταχωρητής που τοποθετείται πριν από τη μνήμη ελέγχου δεν μειώνει (βελτιώνει) το κρίσιμο μονοπάτι, διότι με αυτήν την τοποθέτηση δεν επιτρέπει την επικάλυψη υπολογισμού διαδρομής δεδομένων, υπολογισμού επομένης διεύθυνσης και ανάκλησης μικροεντολής. Ούτως ή άλλως από τον CSAR προς τη μνήμη ελέγχου, μόνο μια λειτουργία γίνεται: ο εντοπισμός μιας εντολής. Αυτή η μοναδική λειτουργία δεν μπορεί να επηρεαστεί από την τοποθέτηση του καταχωρητή διοχέτευσης.

Β) Συμφωνούμε με την προτεινόμενη βελτίωση, αν υποθέσουμε ότι έχουμε προσεκτική αρχικοποίηση του CSAR (καταχωρητής μακροεντολών) και επίσης, αν υποθέσουμε ότι η κατάσταση της διαδρομής δεδομένων (data path) μπορεί να διαμοιραστεί με ασφάλεια μεταξύ αυτών των δύο διαφορετικών νημάτων (ανεξάρτητων προγραμμάτων).

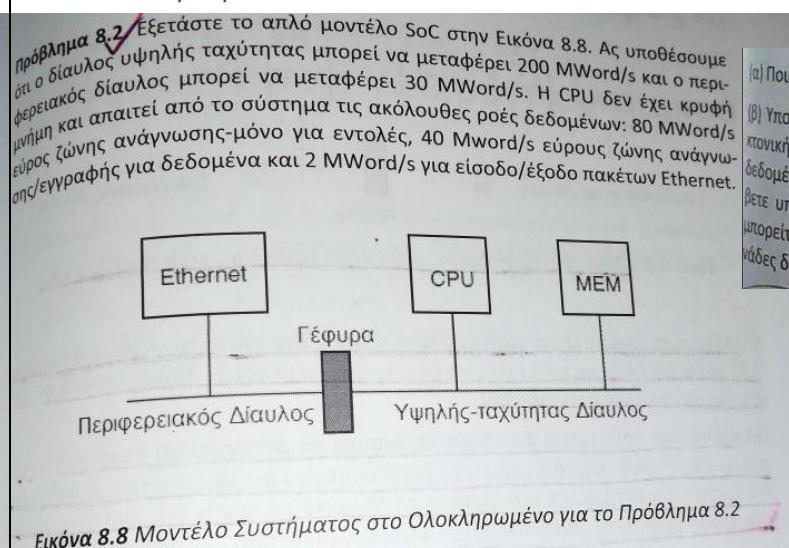
27 Άσκηση 8.1

- α) Εξηγήστε γιατί η περιοχή διευθύνσεων που καταλαμβάνεται από το περιφερειακό UART του SoC, δεν μπορεί να αποθηκεύεται σε κρυφή μνήμη από τον επεξεργαστή RISC.
- β) Υποθέστε ότι ο δίαυλος υψηλής ταχύτητας θα περιλαμβάνει ένα **δεύτερο πυρήνα RISC**, ο οποίος διαθέτει επίσης μια κρυφή μνήμη εντολών και μια κρυφή μνήμη δεδομένων. Εξηγήστε, γιατί χωρίς ειδικά μέτρα, η χρήση κρυφής μνήμης μπορεί να προκαλέσει προβλήματα στη σταθερή λειτουργία του συστήματος.
- γ) Μια γρήγορη λύση του προβλήματος που περιγράφεται στο (β) θα μπορούσε να επιτευχθεί με την απομάκρυνση μιας από τις κρυφές μνήμες σε κάθε επεξεργαστή. Σε μια τέτοια περίπτωση, ποια κρυφή μνήμη πρέπει να αφαιρεθεί: η κρυφή μνήμη εντολών ή η κρυφή μνήμη δεδομένων;

Λύση

- α) Κάθε προσπέλαση στο UART **χρειάζεται να γίνει απευθείας στο περιφερειακό**. Η αποθήκευση στην κρυφή μνήμη αυτών των μεταβλητών που σχετίζονται με το UART δεν έχει νόημα. **Οι κρυφές μνήμες εντολών I\$ και δεδομένων D\$ είναι χρήσιμες μόνο για το λογισμικό που εκτελείται στο μικροεπεξεργαστή**. Αυτό που θα πρέπει να ειπωθεί είναι ότι περιοχή διευθύνσεων που καταλαμβάνεται από το περιφερειακό UART του SoC **πρέπει να βρίσκεται έξω από τον έλεγχο του επεξεργαστή**. Ένα παράδειγμα ενός τέτοιου καταχωρητή είναι ένας καταχωρητής λήψης δεδομένων (data - receive register) του UART.
- β) Πολλαπλοί επεξεργαστές μπορούν να ενημερώσουν θέσεις μνήμης **ανεξάρτητα** ο ένας από τον άλλο. Κάθε φορά που μια δεδομένη θέση μνήμης ενημερώνεται (εγγράφεται), όλα τα αντίγραφα της κρυφής μνήμης (cached copies) αυτής της τοποθεσίας μνήμης θα πρέπει να ενημερωθούν επίσης. Αυτό το πρόβλημα είναι γνωστό ως πρόβλημα **συνοχής της cache (cache - coherency problem)**: πολλαπλές κρυφές μνήμες σε ένα δίαυλο πρέπει να διατηρούν συνεπή προβολή της κύριας μνήμης.
- Γ) **Την κρυφή μνήμη δεδομένων**, από τη στιγμή που η μνήμη εντολών είναι μόνο ανάγνωσης.

28 Άσκηση 8.2



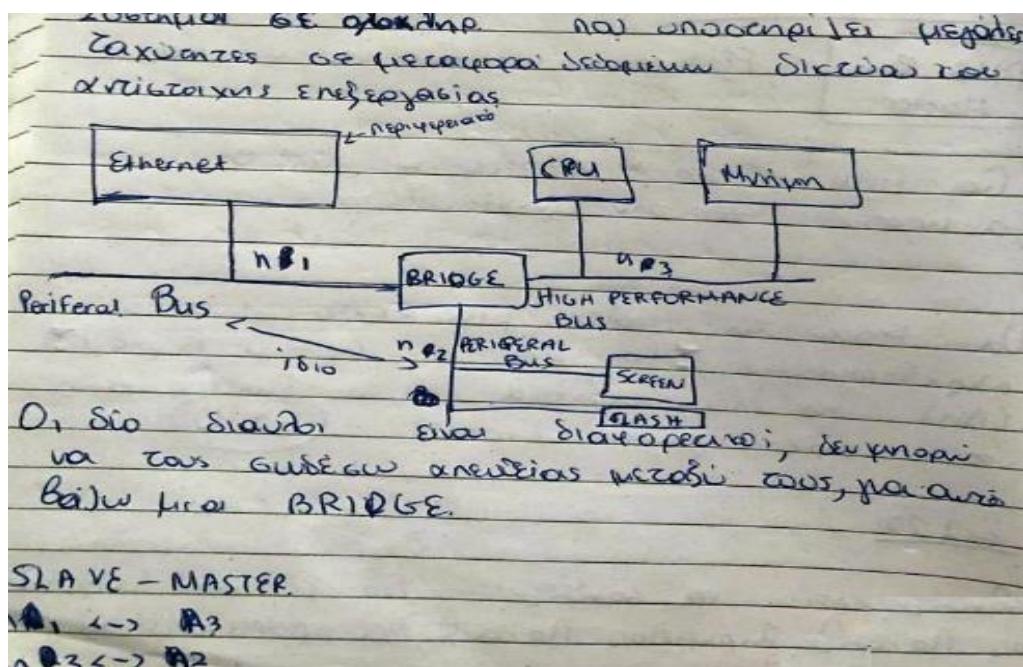
- (a) Ποιο είναι το εύρος ζώνης δεδομένων μέσω της γέφυρας διαύλου;
- (b) Υποθέστε ότι πρέπει να μετατρέψετε αυτή την αρχιτεκτονική σε μια αρχιτεκτονική διπλού πυρήνα, όπου ο δεύτερος πυρήνας έχει τις ίδιες απαιτήσεις ροής δεδομένων με τον πρώτο πυρήνα. Συζητήστε πώς θα τροποποιήσετε το SoC. Λάβετε υπόψη ότι μπορείτε να προσθέσετε εξαρτήματα και διαύλους, αλλά δεν μπορείτε να αλλάξετε τις προδιαγραφές τους. Μην ξεχάσετε να προσθέσετε μονάδες διαιτησίας διαύλου, εάν τις χρειάζεστε.

Λύση

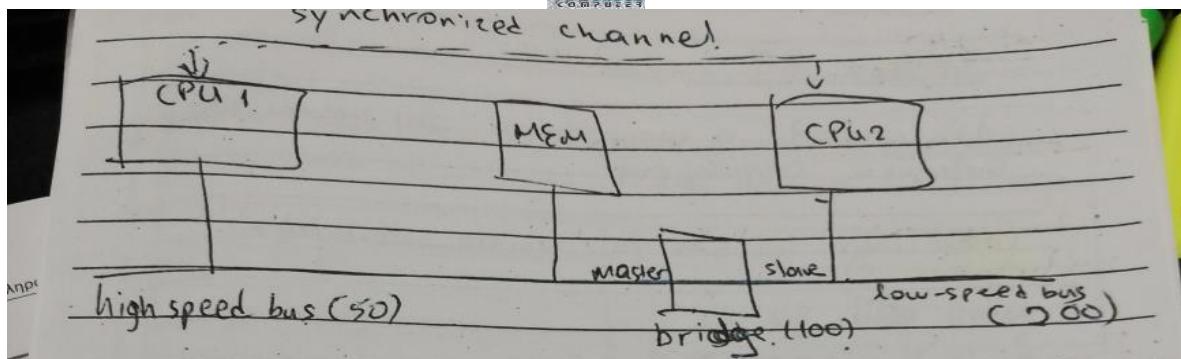
α) Μια γέφυρα διαύλου μπορεί να λειτουργεί είτε ως αφέντης είτε ως σκλάβος, **ανάλογα με την κατεύθυνση των μεταφορών**. Όταν πηγαίνουμε από το δίαυλο υψηλής ταχύτητας προς τον περιφερειακό δίαυλο, η γέφυρα λειτουργεί ως σκλάβος - διαύλου στο δίαυλο υψηλής ταχύτητας και ως αφέντης - διαύλου στον περιφερειακό δίαυλο. Το εύρος ζώνης θα είναι 200 MWord/s

β) Τροποποίηση SOC. Συνδέουμε και άλλες περιφερειακές συσκευές, όπως μια οθόνη (screen) και μια μνήμη (FLASH).

Η σύνδεση αυτή πραγματοποιείται μέσω της γέφυρας που έχουμε. Έχουμε διαύλους (αρτηρίες) διαφορετικών ταχυτήτων, π.χ. **high – speed bus** για σύνδεση γρήγορων συσκευών, όπως είναι η CPU και η κύρια μνήμη και **peripheral bus** για σύνδεση αργών περιφερειακών συσκευών. Η σύνδεση επιπλέον συσκευών, όπως π.χ. οθόνης και μνήμης, απαιτεί τη σύνδεσή τους με την υπάρχουσα γέφυρα.



Άσκηση από το μάθημα



CPU synchronization requires the connection.

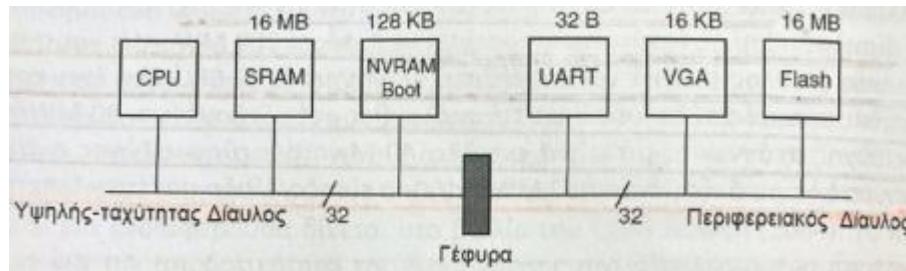
low-speed : 200 ns, high-speed : 50ns
 data transfer (bridge) : 100ns
 read-write (memory) : 80ns

$(\text{CPU1} \rightarrow \text{CPU2} : \text{read-write} : 50 + 80 + 100 = 230\text{ns})$
 (gia va dia ban n 2 vao tuu 1)

$(\text{CPU2} \rightarrow \text{CPU1} : 200 + 100 + 80 = 380\text{ns})$
 (gia va dia ban n 1 vao tuu 2)

29 Άσκηση 8.3

Θα πρέπει να σχεδιάσετε ένα χάρτη μνήμης για το SoC που φαίνεται στην επόμενη Εικόνα. Το σύστημα περιέχει ένα δίαυλο υψηλής ταχύτητας και ένα περιφερειακό δίαυλο, και οι δύο με χώρο διεύθυνσης 32 bit μεταφέρουν λέξεις (32 bit). Τα εξαρτήματα του συστήματος περιλαμβάνουν έναν RISC, 16 MB RAM, 128 KB μη πιητική μνήμη προγράμματος, 16 MB μνήμης Flash. Επιπλέον, υπάρχει μια περιφερειακή VGA και ένα περιφερειακό UART. Η VGA διαθέτει μνήμη προσωρινής αποθήκευσης βίντεο 16 KB και ο UART περιέχει 32 byte καταχωρητών μετάδοσης/λήψης.



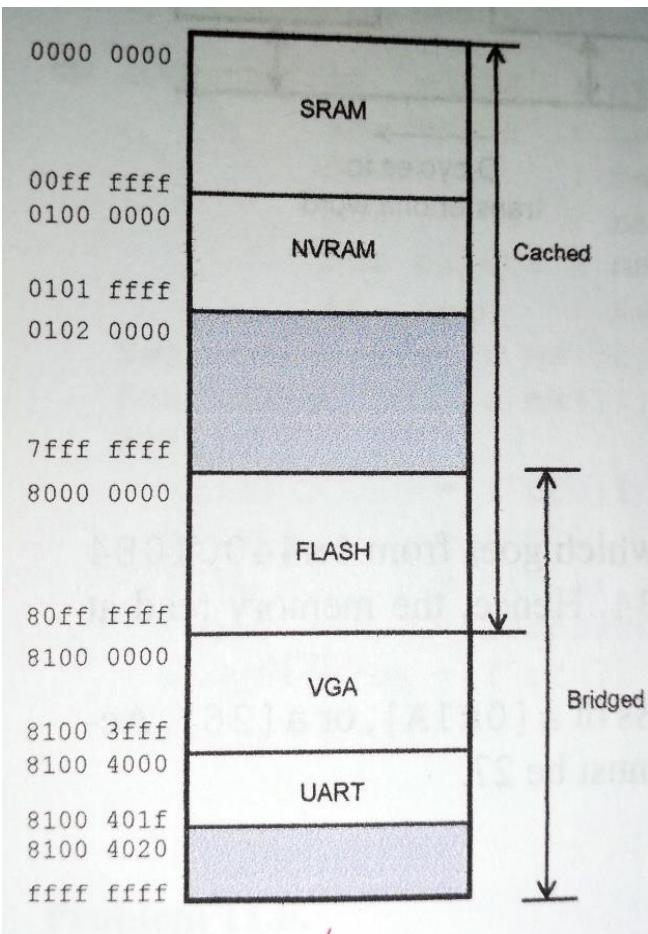
(α) Σχεδιάστε έναν πιθανό χάρτη μνήμης για τον επεξεργαστή. Λάβετε υπόψη σας ότι η Γέφυρα Διαύλου μπορεί μόνο να μετατρέπει μεταφορές διαύλων μέσα σε ένα ενιαίο, συνεχή χώρο διεύθυνσης.

(β) Καθορίστε ποιο εύρος διευθύνσεων μπορεί να αποθηκευτεί σε κρυφή μνήμη από τον επεξεργαστή. Μια «περιοχή διευθύνσεων κρυφής μνήμης» σημαίνει ότι μια ανάγνωση μνήμης σε μια διεύθυνση σε αυτό το εύρος θα οδηγήσει σε ένα αντίγραφο ασφαλείας αποθηκευμένο στην κρυφή μνήμη.

Λύση

Μέσα στο SoC υπάρχουν διαφορετικά είδη διαύλων, όπως για παράδειγμα περιφερειακοί δίαυλοι (Peripheral Bus) και θώρακες δίαυλοι υψηλής ταχύτητας (High – speed Bus). Μεταξύ διαύλων διαφορετικών ταχυτήτων μεσολαβούν προσαρμοστικά κυκλώματα που ονομάζονται γέφυρες.

α) Για να σχεδιάσουμε ένα χάρτη μνήμης (**memory map**), θα πρέπει να είναι διαθέσιμες οι χωρητικότητες των στοιχείων μνήμης του κυκλώματος. Παρατηρούμε ότι αυτές είναι διαθέσιμες από το κύκλωμα που δίνεται. Στη συνέχεια παρατίθεται ο ζητούμενος χάρτης μνήμης. Όταν υπάρχουν διαφορετικά είδη μνήμης όπως στην προκειμένη περίπτωση (π.χ. SRAM, NVRAM, FLASH κ.λ.π.), είναι **τυποποιημένος** ο τρόπος τοποθέτησής τους για να σχηματίσουν **ένα ενιαίο σύστημα μνήμης (αρχιτεκτονική Von Neumann)**. Πιο συγκεκριμένα, τα διαφορετικά είδη μνήμης τοποθετούνται με τη σειρά που φαίνονται δίπλα, από πάνω προς τα κάτω (+).



- **SRAM: 16 MB** = $2^4 \times 2^{20}$ bytes = 2^{24} bytes = 2^{24} θ.μ. = 10000

0000 0000 0000 0000 **0000** = 1000000H – 1 = OFFFFFFH

- **NVRAM: 128 KB** = $2^7 \times 2^{10}$ bytes = 2^{17} bytes = 2^{17} θ.μ. = 10

0000 0000 0000 **0000** = 20000, από 0100 0000 + 0002 0000

= **01020000 -1 → 0101FFFFH.**

- **FLASH: 16MB:** = 2^{24} θ.μ. = 1000000H, από 8000 0000 +

1000000H = **8100 0000H – 1 → 80FFFFFFH.**

- **VGA: 16KB** = 2^{14} θ.μ. = 100 0000 0000 0000 = 4000H, από

8100 0000 + 4000H = **8100 4000H - 1 → 8100 3FFFH.**

- **UART (περιφερειακή συσκευή) 32 bytes** = 100000 = 20H,

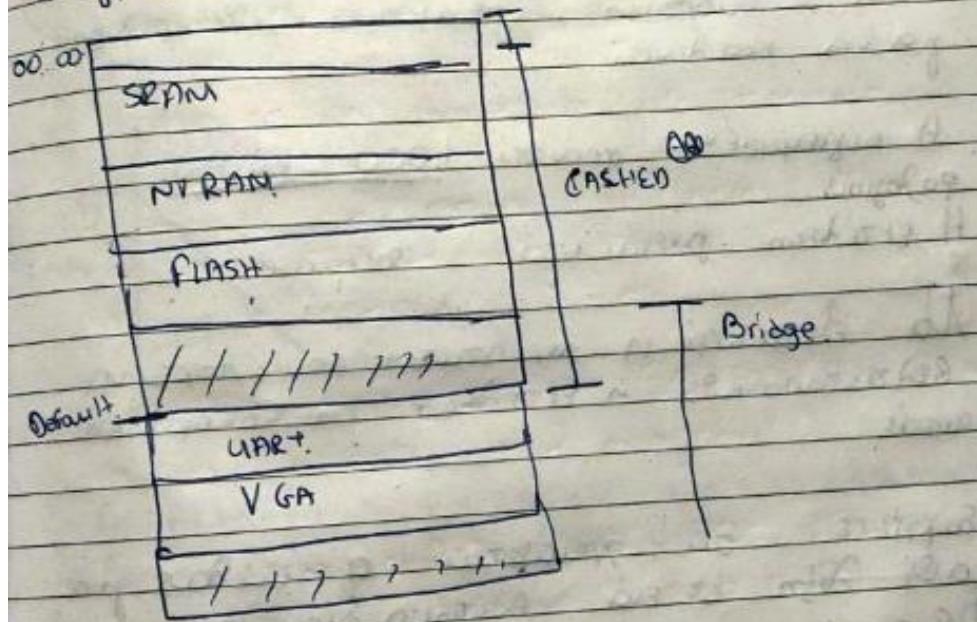
από **8100 4000 + 20 H = 8100 4020 – 1 → 8100401F.**

Παρατήρηση: Επειδή το address bus = 32 bit, σημαίνει ότι υπάρχουν 2^{32} θ.μ. = 1 0000 0000 0000 0000 0000 0000 0000 0000 = 1 0000 0000H, αυτό το μέγεθος ανήκει στην περιοχή διευθύνσεων από **0000 0000 – FFFF FFFF**

β) Στον παραπάνω χάρτη μνήμης τοποθετούνται με αύξουσα σειρά προς τα κάτω, **πρώτα** οι μνήμες που βρίσκονται συνδεδεμένες στο δίαυλο υψηλής ταχύτητας (high – speed bus) του SoC και στη συνέχεια οι μνήμες που βρίσκονται συνδεδεμένες στο δίαυλο χαμηλής ταχύτητας (peripheral bus) του SoC. Ο χώρος μνήμης που υπάρχει στο χάρτη μνήμης στην περιοχή **0000 0000 – 80FF FFFF** θα μπορούσε να χρησιμοποιηθεί ως περιοχή διευθύνσεων κρυφής μνήμης επεξεργαστή (cached). Παρατηρούμε ότι λόγω του κενού χώρου που υπάρχει στο χάρτη μνήμης, οι διευθύνσεις των μνημών που είναι συνδεδεμένες στο peripheral bus, ξεκινούν από «1» (MSB).

Παρατήρηση: Αν ζητήσει βελτιστοποίηση του προηγούμενου σχήματος και του χάρτη μνήμης, τότε θα πρέπει να χωρίσουμε τα περιφερειακά σε γρήγορα και αργά και έχω δύο γέφυρες.

Bentuk koningen → Na Xuh Gau kei
Ge jeppeva kei opje kei va Ewu Sub Bridge.



Mengabarkan tipe tns Default (Toko
nasi nio xaw)

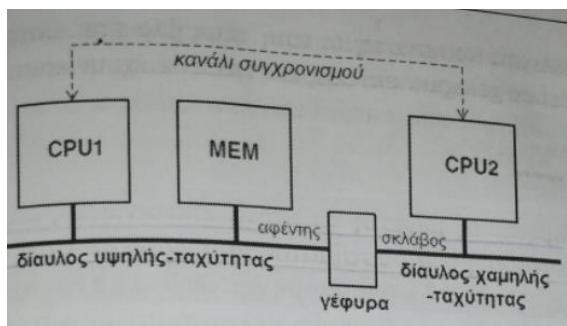
Bajw in flash wa high speed va
xive cached.

30 Πρόβλημα 9.1

7 Πρόβλημα

πρόβλημα 9.1. Βρείτε τη μέγιστη ταχύτητα επικοινωνίας από την CPU1 στην CPU2 στην αρχιτεκτονική του συστήματος που φαίνεται στην Εικόνα 9.11. Υπάρχεται ένα κανάλι συγχρονισμού που διατίθεται στην διάθεσή τους. Τα δύο CPU έχουν στην διάθεσή τους ένα αφοσιωμένο κανάλι συγχρονισμού που χρησιμοποιείται ώστε να μπορούν να επιλέξουν τη βέλτιστη στιγμή για να εκτελέσουν τις απαιτήσεις των υπηρεσιών. Συναλλαγή ανάγνωσης ή εγγραφής. Χρησιμοποιήστε τις ακόλουθες σταθερές για την ψηφιακή σχεδίαση.

- Κάθε συναλλαγή διαύλου στον δίαυλο υψηλής ταχύτητας διαρκεί 50 ns.
- Κάθε συναλλαγή διαύλου στο δίαυλο χαμηλής ταχύτητας διαρκεί 200 ns.
- Κάθε προσπέλαση μνήμης (ανάγνωση ή εγγραφή) διαρκεί 80 ns.
- Κάθε μεταφορά γέφυρας διαρκεί 100 ns.
- Οι CPU είναι πολύ ταχύτερες από το δίαυλο συστήματος και μπορούν να διαβάζουν/γράφουν δεδομένα στο δίαυλο σε οποιοδήποτε επιλεγμένο ρυθμό δεδομένων.



Λύση

To communicate from CPU1 to CPU2, at least the following bus transactions will be needed:

- A high-speed bus transaction (50ns)
- A bridge transfer (100 ns)
- A low-speed bus transaction (200ns)

The communication latency is therefore $50 + 100 + 200\text{ns}$, or 350ns , or $2.87\text{M}\text{transactions/sec}$. This is an optimistic upperbound; in practice additional memory accesses will slow down the system operation.

Επειδή μας ενδιαφέρει η επικοινωνία **CPU1 – CPU2**, δεν θα λάβουμε υπόψη από τα δεδομένα της εκφώνησης το χρόνο προσπέλασης της μνήμης (80 nsec). Θα λάβουμε υπόψη **μόνο** εκείνα τα στοιχεία που είναι απαραίτητα για την επικοινωνία, δηλαδή: **δίαυλος υψηλής - ταχύτητας – γέφυρα – δίαυλος χαμηλής – ταχύτητας** → $50\text{nsec} - 100\text{nsec} - 200\text{nsec} = 350\text{nsec}$, οπότε η μέγιστη ταχύτητα επικοινωνίας είναι: $\frac{1}{350\text{nsec}} = 0,002857 \times 10^9 \text{Hz} = 2.857 \times 10^6 \text{Hz} = 2.857 \text{MHz}$ ή αλλιώς **2.857 Mtransactions/sec**. Αυτό είναι θεωρητικό άνω φράγμα, στην πραγματικότητα επιπρόσθετες προσπελάσεις μνήμης θα επιβραδύνουν τη λειτουργία του συστήματος.

Παρατήρηση: αν η εκφώνηση ζητούσε μέγιστη ταχύτητα επικοινωνίας μεταξύ **CPU1 – μνήμης**, τότε ο συνολικός χρόνος θα ήταν δίαυλος υψηλής ταχύτητας – **MEM** → $50\text{nsec} - 80\text{nsec} = 130\text{nsec}$, οπότε η μέγιστη ταχύτητα επικοινωνίας είναι: $\frac{1}{130\text{nsec}} = 0,007692 \times 10^9 \text{Hz} = 7.692 \times 10^6 \text{Hz} = 7.692 \text{MHz}$ ή αλλιώς **7.692 Mtransactions/sec**.

Mtransactions. Αυτό είναι θεωρητικό άνω φράγμα, στην πραγματικότητα επιπρόσθετες προσπελάσεις μνήμης θα επιβραδύνουν τη λειτουργία του συστήματος.

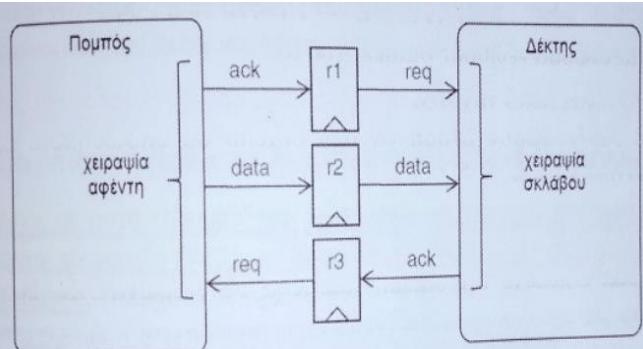
31 Πρόβλημα 9.2

Πρόβλημα 9.2. Εξετάστε την διμερή χειραψία στην Εικόνα 9.12. Ένας πομπός συγχρονίζεται με ένα δέκτη και μεταδίδει μια ακολουθία συμβόλων δεδομένων μέσω ενός καταχωρητή δεδομένων.

(α) Περιγράψτε κάτω από ποιες συνθήκες ο καταχωρητής $r1$ μπορεί να αφαιρεί χωρίς να ζημιώθει η ακεραιότητα της επικοινωνίας. Υποθέστε ότι, αφού απομακρυνθεί ο $r1$, η είσοδος req του δέκτη συνδέεται με λογικό-1.

(β) Περιγράψτε κάτω από ποιες συνθήκες ο καταχωρητής $r3$ μπορεί να αφαιρεί χωρίς να ζημιώθει η ακεραιότητα της επικοινωνίας. Υποθέστε ότι, αφού απομακρυνθεί ο $r3$, η είσοδος req του πομπού συνδέεται με λογικό-1.

(γ) Υποθέστε ότι πρόκειται να αντικαταστήσετε τον καταχωρητή $r1$ με δύο καταχωρητές στη σειρά, έτσι ώστε ολόκληρη η μετάβαση από το πομπό- ack στο δέκτη- req να διαρκεί πλέον δύο κύκλους ρολογιού αντί για έναν. Περιγράψτε την επίπτωση αυτής της αλλαγής στο ρυθμό μετάδοσης της επικοινωνίας και περιγράψτε την επίπτωση αυτής της αλλαγής στο λογισμικό χρόνο της επικοινωνίας.



Λύση

α) Η μεταφορά δεδομένων από πομπό προς δέκτη, μπορεί να γίνει μέσω της γραμμής $data$. Η τιμή του $r1$ μπορεί να αλλάξει (χωρίς να αλλάξει η λειτουργία του συστήματος) στον **επόμενο** κύκλο ρολογιού (αφού πρώτα έχουμε ορίσει για κύκλο ρολογιού μια συγκεκριμένη χρονική διάρκεια).

β) Η τιμή του $r3$ μπορεί να αλλάξει (χωρίς να αλλάξει η λειτουργία του συστήματος) στον επόμενο κύκλο ρολογιού (αφού πρώτα έχουμε ορίσει για κύκλο ρολογιού μια συγκεκριμένη χρονική διάρκεια).

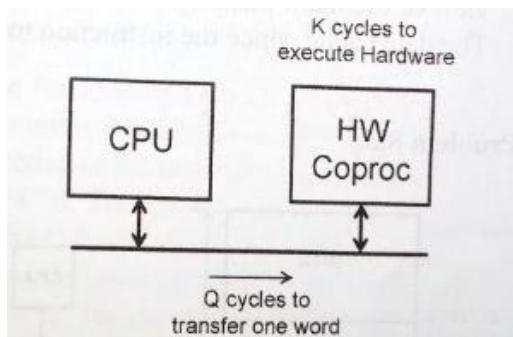
γ) Αρχικά **το latency είναι 3 κ.ρ.** (1 η πρώτη γραμμή, 1 η δεύτερη και 1 η τρίτη). Μετά το latency γίνεται 4 κ.ρ. (2 κ.ρ. η πρώτη γραμμή, 1 η δεύτερη και 1 η τρίτη). Υποθέτουμε ότι οι άλλες δύο «γραμμές» γίνονται (λειτουργούν) σε 1 κ.ρ. Και πριν τελειώσει η επικοινωνία, ο πομπός δεν μπορεί να ξεκινήσει καινούρια μετάδοση. Επειδή είναι διμερής, θα πρέπει να περιμένει. Δηλαδή, όσον αφορά την επίπτωση αυτής της αλλαγής στο **λανθάνοντα χρόνο** (latency), θα πρέπει να αναφερθεί ότι αυτός αυξάνεται από 3 κ.ρ. σε 4 κ.ρ. Όσον αφορά το **ρυθμό μετάδοσης**, αυτός δεν επηρεάζεται από την αλλαγή, αφού η μετάδοση των δεδομένων ολοκληρώνεται πάλι σε 1 κ.ρ.

32 Πρόβλημα 9.3

Μια συνάρτηση C έχει δέκα εισόδους και δέκα εξόδους, όλες ακέραιοι. Η συνάρτηση απαιτεί 1000 κύκλους για εκτέλεση στο λογισμικό. Πρέπει να αξιολογήσετε αν είναι λογικό να κατασκευάσετε ένα συνεπεξεργαστή για αυτή τη συνάρτηση. Υποθέστε ότι η συνάρτηση απαιτεί K κύκλους για εκτέλεση σε υλικό και ότι χρειάζεστε Q κύκλους για να μεταφέρετε μια λέξη μεταξύ του λογισμικού και του συνεπεξεργαστή μέσω ενός διαύλου συστήματος. Σχεδιάστε ένα γράφημα που απεικονίζει το Q συναρτήσει του K, και υποδείξτε ποιες περιοχές σε αυτό το γράφημα δικαιολογούν έναν συνεπεξεργαστή.

Λύση

Η επικοινωνία μεταξύ ΚΜΕ (CPU) και συνεπεξεργαστή (coprocessor) περιγράφεται με το επόμενο σχήμα:

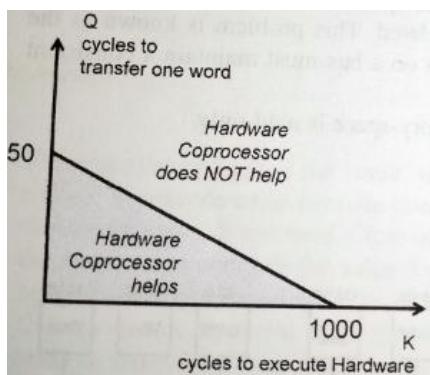


Οι **1000 κ.ρ.** αφορούν το συνολικό χρόνο εκτέλεσης για μια λειτουργία στο λογισμικό, ενώ το **άθροισμα K** (κύκλοι για εκτέλεση σε υλικό) + $20 * Q$ (κύκλοι μεταφοράς μεταξύ λογισμικού και συνεπεξεργαστή) αφορά το συνολικό χρόνο εκτέλεσης για μια λειτουργία στο υλικό.

Ο συνεπεξεργαστής απαιτεί K κ.ρ. για τη λειτουργία του και επίσης απαιτούνται και Q κ.ρ. για τη μεταφορά μιας λέξης (word) από τον επεξεργαστή στο συνεπεξεργαστή. Αρχικά η εξίσωση της ευθείας που περιγράφει τη συγκεκριμένη C συνάρτηση, που έχει 10 εισόδους και 10 εξόδους (συνολικά 20 λέξεις) είναι η εξής (από τη στιγμή που μεταφέρονται στο συνεπεξεργαστή 20 λέξεις και απαιτούνται Q κύκλοι για τη μεταφορά μιας λέξης, συνολικά θα χρειαστούν $20 * Q$ κύκλοι):

$$1000 = K \text{ (κύκλοι για εκτέλεση σε υλικό)} + 20 * Q \text{ (κύκλοι μεταφοράς μεταξύ λογισμικού και συνεπεξεργαστή)}$$

Στη συνέχεια φαίνεται η γραφική παράσταση της εν' λόγω εξίσωσης, η οποία διαιρεί το επίπεδο (K, Q) σε δύο περιοχές:

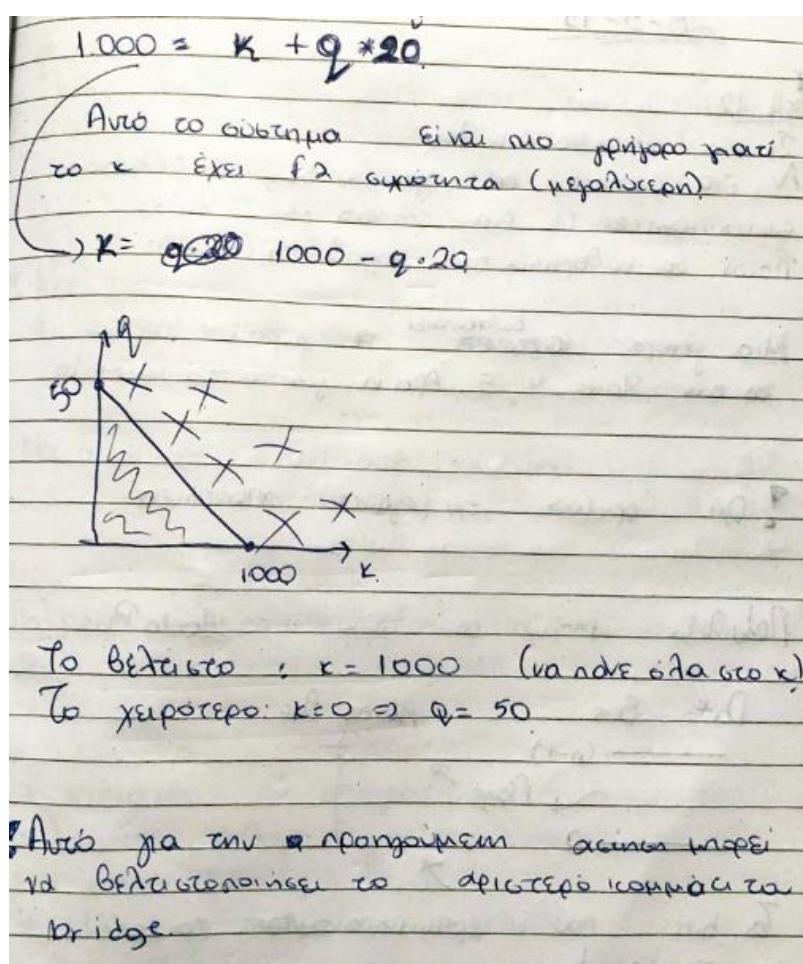


Στην περιοχή που βρίσκεται κάτω από τη γραμμή (κάτω περιοχή) ισχύει ότι $1000 > K + 20 * Q$. Αυτό σημαίνει ότι ο συνολικός χρόνος εκτέλεσης που καταναλώνεται από το υλικό¹ είναι **μικρότερος** από το χρόνο

¹ Ο όρος υλικό αναφέρεται στο εξειδικευμένο υλικό που περιέχεται στο συνεπεξεργαστή.

εκτέλεσης για την ίδια λειτουργία στο λογισμικό. Άρα, **ο συνεπεξεργαστής (coprocessor) παρέχει ένα συνολικό κέρδος, από την οπτική της επιτάχυνσης.**

Στην περιοχή που βρίσκεται πάνω από τη γραμμή (άνω περιοχή) ισχύει ότι $1000 < K + 20 * Q$. Αυτό σημαίνει ότι ο συνολικός χρόνος εκτέλεσης που καταναλώνεται από το υλικό, είναι μεγαλύτερος από το χρόνο εκτέλεσης για την ίδια λειτουργία στο λογισμικό. Άρα, **ο συνεπεξεργαστής (coprocessor) δεν παρέχει στην περίπτωση αυτή κάποιο όφελος (επιτάχυνση) έναντι του λογισμικού, άρα στην περίπτωση αυτή δεν δικαιολογείται η χρήση του.**



33 Άσκηση 10.2

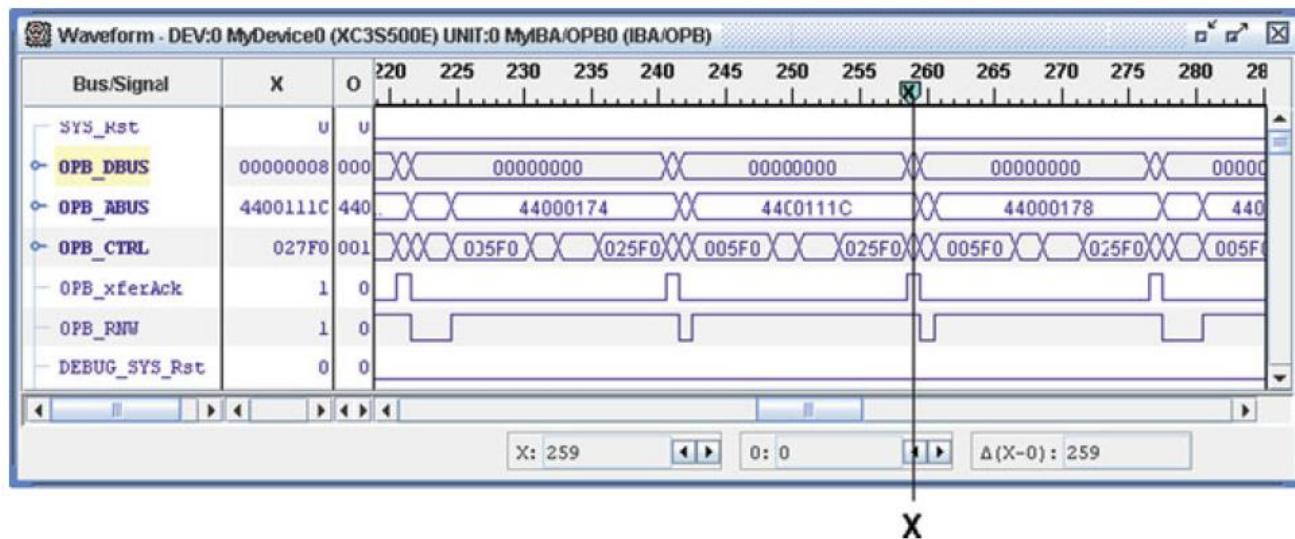
Ενώ κάνετε αποσφαλμάτωση ενός προγράμματος C σε ένα μικροεπεξεργαστή 32 – bit, που εμφανίζεται στη συνέχεια, αποτυπώνετε τη μεταβλητή διαύλου που φαίνεται που φαίνεται στο παρακάτω γράφημα.

```
#include <stdio.h>
void main() {
int i, a[0x40];
for (i=0; i< 0x40; i++)
if (i > 0x23)
a[i] = a[i-1] + 1;
else
a[i] = 0x5;
}
```

Ο μικροεπεξεργαστής είναι συνδεδεμένος σε μια μνήμη εντός ολοκληρωμένου που κρατά το πρόγραμμα και τα δεδομένα. Ο κώδικας και το τμήμα δεδομένων αποθηκεύονται σε μνήμη εκτός ολοκληρωμένου ξεκινώντας από τη διεύθυνση 0x44001084. Οι εντολές από το σώμα του βρόχου ξεκινούν από τη διεύθυνση 0x44000170.

Παρατηρήστε προσεκτικά το διάγραμμα χρονισμού που ακολουθεί και απαντήστε στις ακόλουθες ερωτήσεις:

- (α) Ο δρομέας X στην επόμενη εικόνα τοποθετείται σε ένα σημείο για το οποίο ο δίαυλος διευθύνσεων περιέχει την τιμή 0x4400111C και ο δίαυλος δεδομένων περιέχει 0x8. Είναι αυτό μια ανάγνωση ή μια εγγραφή μνήμης;
- (β) Για την ίδια θέση δρομέα X, είναι προσπέλαση μνήμης που αφορά ανάκτηση – εντολής ή ανάγνωση μνήμης – δεδομένων;
- (γ) Για την ίδια θέση δρομέα X, ποια είναι η τιμή του μετρητή βρόχου i από το πρόγραμμα C;



Λύση

- (α) Πρόκειται για **διαδικασία ανάγνωσης** και αυτό το καταλαβαίνουμε όχι από το address bus ή το data bus, αλλά από το σήμα **OPB_RNW (Read not Write)** που υπάρχει και το οποίο έχει ενεργοποιηθεί (δηλ. είναι σε υψηλό δυναμικό). Όταν αυτό το σήμα αυτό είναι σε υψηλό δυναμικό -όπως εδώ- έχουμε ανάγνωση.

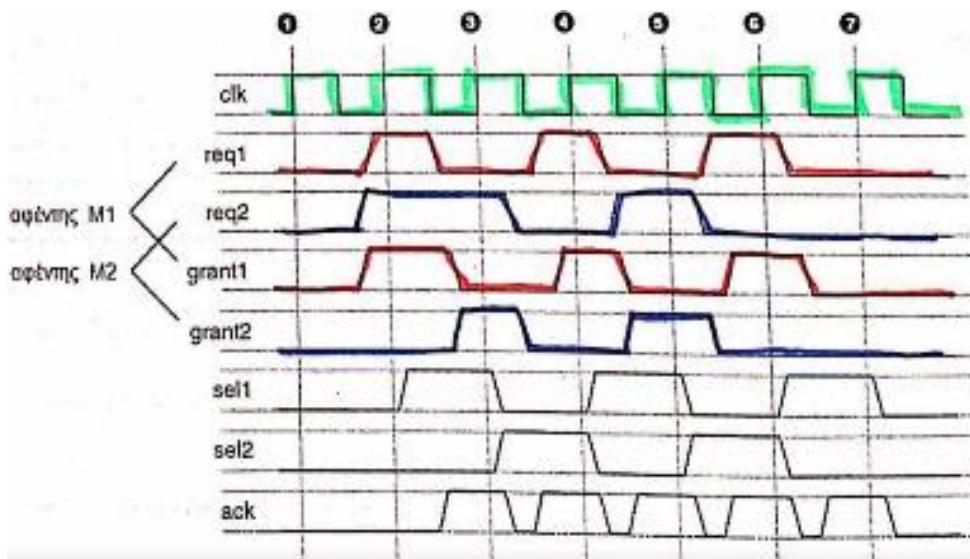
(β) Αν παρατηρήσουμε τη θέση X θα δούμε ότι αφορά τη διεύθυνση 0x4400111C, η οποία είναι μετά τη διεύθυνση έναρξης 0x44001084 του πίνακα a[], αφού **0x4400111C > 0x44001084** και μάλιστα εμπίπτει εντός των ορίων 0x44001084 και $0x44001084 + 0x40 * 4 = 0x440011E4$. Αν παρατηρήσουμε το data bus, με το σήμα OPB_DBUS θα διαπιστώσουμε ότι αποτελείται από 8 δεκεξαεδικά ψηφία = $8 \times 4 = 32$ bits = 4 bytes. Ο πίνακας a[] περιέχει δεδομένα, έχει μέγεθος $0x40 * 4 = 0x160$ bytes.

(γ) Η διεύθυνση 0x4400111C αντιστοιχεί στη διεύθυνση του a[0x1A], ή αλλιώς a[26]. Σύμφωνα με τη λίστα 10.1 το a[26] διαβάζεται ότι το i είναι 27. Αυτό προκύπτει διότι βρίσκεται στην ανάγνωση και μετα το i = 23, το $a[i - 1] + 1 = 0x23 - 0x9 = 0x1A$ θέσεις. Το 9 προκύπτει λόγω του διαύλου δεδομένων που έχει τιμή 0x08.

34 Άσκηση 10.4

Το διάγραμμα χρονισμού στη Εικόνα που ακολουθεί δείχνει τη διαδικασία διαιτησίας δύο αφεντών M₁ και M₂, που ζητούν πρόσβαση σε ένα κοινόχρηστο δίαυλο. Απαντήστε στις παρακάτω ερωτήσεις, χρησιμοποιώντας τις πληροφορίες που παρέχονται στο διάγραμμα χρονισμού.

- (α) Με βάση το διάγραμμα χρονισμού, ποιος αφέντης έχει την υψηλότερη προτεραιότητα για μεταφορές διαύλου: ο M₁, ο M₂ ή είναι αδύνατο να πούμε;
- (β) Ποιος αφέντης έχει τον έλεγχο του διαύλου διευθύνσεων μεταξύ της ακμής ρολογιού 3 και της ακμής ρολογιού 4: ο M₁, ο M₂ ή είναι αδύνατο να πούμε;
- (γ) Ποιος τύπος εξαρτήματος καθορίζει την τιμή των σημάτων grantx: ένας αφέντης διαύλου, ένας διαιτητής διαύλου ή ένας σκλάβος διαύλου;
- (δ) Ποιος τύπος εξαρτήματος καθορίζει την τιμή του σήματος ack: ένας αφέντης διαύλου, ένας διαιτητής διαύλου ή ένας σκλάβος διαύλου;





(a) Τόσο ο αφέντης M1 όσο και ο αφέντης M2 ζητούν ταυτόχρονα το δίαυλο, και αυτό φαίνεται από τα σήματα req1 και req2 που ενεργοποιούνται ταυτόχρονα. **Ο αφέντης M1 έχει μεγαλύτερη προτεραιότητα**, διότι στη 2^η ακμή ρολογιού έχει ενεργοποιηθεί **πρώτο το σήμα grant1**, το οποίο επιστρέφεται από το διαιτητή διαύλου (bus arbiter). **Ένα σήμα grant έρχεται πάντα από το διαιτητή διαύλου στον αφέντη**.

(b) Στην ακμή ρολογιού 3, **ο αφέντης (master) 2 έχει τον έλεγχο του διαύλου**. Αυτό φαίνεται από το ότι μεταξύ των ακμών 3 και 4, **τέμνεται το σήμα grant2**, το οποίο επιστρέφεται από το διαιτητή διαύλου (bus arbiter).

(c) Αυτό το κάνει πάντα **ο διαιτητής διαύλου (bus arbiter)**, ο οποίος παράγει τα σήματα grant, τα οποία έρχονται ως άμεση απάντηση στα σήματα request. Επομένως ο διαιτητής διαύλου είναι η συσκευή που προσδιορίζει ποια μονάδα θα πάρει το δίαυλο υπό τον έλεγχό της.

(d) Αυτό το κάνει πάντα **ο σκλάβος διαύλου**, διότι είναι αυτός που **αποδέχεται** τη μεταφορά από τον αφέντη και η μεταφορά αυτή ξεκινά μόνο κατόπιν της ενεργοποίησης του σήματος ack εκ' μέρους του σκλάβου.

Κανόνας:

- Σήματα με ένδειξη req_i (request) προέρχονται από κάποιο master
- Σήματα με ένδειξη grant_i (grant) προέρχονται από κάποιο διαιτητή διαύλου
- Σήματα με ένδειξη ack_i (acknowledge) προέρχονται από κάποιο slave.

35 Επίλυση Θεμάτων Ιουνίου 2020

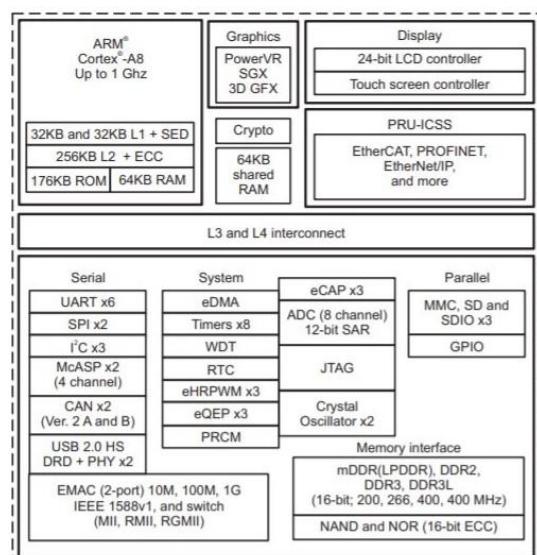
Ερώτηση 1

Να αναπτύξετε και να περιγράψτε λεπτομερώς ένα **σύστημα προηγμένου επεξεργαστή**: α) Περιγραφή εφαρμογής και λειτουργίας: τεχνικές προδιαγραφές. β) Μοντελοποίηση. γ) Μετασχηματισμός ροής δεδομένων. δ) Μηχανή πεπερασμένων καταστάσεων. ε) Αρχιτεκτονική

(Γενικές απαντήσεις, γενικές περιγραφές και απαντήσεις εκτός θέματος δεν αξιολογούνται. Να είστε όσο πιο ακριβείς συγκριτικά με την ύλη του μαθήματος μπορείτε. Τα θέματα είναι ισοδύναμα)

Λύση

α) Ένα **σύστημα προηγμένου μικροεπεξεργαστή** είναι ένα SoC (System on Chip) το οποίο συνδυάζει διάφορα εξαρτήματα σε ένα σύστημα διαύλου. Ένα SoC περιλαμβάνει όλες εκείνες τις βασικές δυνατότητες ενός επεξεργαστή. Τα SoC καταλαμβάνουν μικρό χώρο, είναι αυτόνομα, κάνουν εξοικονόμηση ενέργειας και παράγουν χαμηλή θερμότητα στην έξοδό τους. Μπορούν να ενσωματώνουν αναλογικές, 3G και WiFi δυνατότητες. Καθίστανται ολοένα και πιο δημοφιλή με την ανάπτυξη του Διαδικτύου των Πραγμάτων (IoT) και των φορητών υπολογιστών. Ένα παράδειγμα τεχνολογίας που χρησιμοποιεί ένα SoC είναι οι κονσόλες βιντεοπαιχνιδιών όπως για παράδειγμα το **Nvidia Tegra X1** που χρησιμοποιείται στο Nintendo. Αναφορικά με τις τεχνικές προδιαγραφές, υπάρχει ένας μικροεπεξεργαστής (τυπικά ένας επεξεργαστής γενικού σκοπού - RISC), που λειτουργεί ως κεντρικός ελεγκτής στο SoC. Επιπλέον, άλλα εξαρτήματα περιλαμβάνουν την μνήμη σε ολοκληρωμένο, τις εκτός ολοκληρωμένου επαφές μνήμης, τα αφοσιωμένα περιφερειακά, τους συνεπεξεργαστές υλικού και τους συνδέσμους επικοινωνίας εξαρτήματος προς εξάρτημα. Ένα άλλο παράδειγμα SoC είναι ο επεξεργαστής Cortex A8 της εταιρείας ARM, ο οποίος περιγράφεται από την επόμενη Εικόνα (περιγραφή της εικόνας): Όπως μπορεί να διαπιστωθεί, περιέχει πέρα από τον επεξεργαστή Cortex – A8, ενσωματωμένα περιφερειακά (όπως π.χ. UART, SPI, USB κ.λ.π.), κρυφή και κύρια μνήμη, ελεγκτή οθόνης και δίαυλο που διασυνδέει όλα τα προαναφερόμενα. Ένα SoC χρησιμοποιείται στους μικροελεγκτές και σχεδιάζεται για μια ειδική εφαρμογή, με αποτέλεσμα η χρήση του να είναι πολύ εξειδικευμένη. Ιδιαίτερα χαρακτηριστικά του αποτελούν η χαμηλή κατανάλωση ενέργειας, το μικρό μέγεθος καθώς επίσης και η υψηλή απόδοση.



β) Ένα σύστημα προηγμένου επεξεργαστή, μπορεί να μοντελοποιηθεί με την τεχνική που ονομάζεται Ροή Δεδομένων (Data Flow), η οποία επιτυγχάνει το στόχο ενός παράλληλου μοντέλου. Γενικά η κατασκευή μιας ακολουθιακής υλοποίησης για ένα παράλληλο μοντέλο είναι πολύ ευκολότερη από το αντίθετο. Ένα ισχυρό χαρακτηριστικό του μοντέλου ροής δεδομένων είναι ότι είναι ένα ταυτόχρονο μοντέλο. Ένα ταυτόχρονο μοντέλο σημαίνει ότι μπορεί να απεικονιστεί μία παράλληλη ή μια ακολουθιακή υλοποίηση και έτσι μπορούν να στοχεύσουν τόσο σε υλικό όσο και σε λογισμικό. Τα μοντέλα ροής δεδομένων είναι κατανεμημένα και δεν υπάρχει ανάγκη για κεντρικό ελεγκτή στο σύστημα, προκειμένου να κρατάει τα επιμέρους εξαρτήματα σε συγχρονισμό. Επιπλέον, τα μοντέλα ροής δεδομένων είναι αρθρωτά και είναι δυνατόν να αναλυθούν. Η ροή δεδομένων παραμένει ιδιαίτερα δημοφιλής για την περιγραφή συστημάτων επεξεργασίας σημάτων, όπως π.χ. είναι το εμπορικό εργαλείο Simulink, που χρησιμοποιεί την ιδέα της ροής δεδομένων.

γ) Η ροή δεδομένων μπορεί να υλοποιηθεί τόσο σε υλικό όσο και σε λογισμικό. Ο απλούστερος τρόπος είναι η απεικόνιση από ένα γράφο SDF μονού ρυθμού στο υλικό. Στο μετασχηματισμό αυτό, κάθε ουρά επικοινωνίας απεικονίζεται με ένα καλώδιο, και κάθε ουρά που περιέχει ένα σύμβολο απεικονίζεται σε έναν καταχωρητή. Επιπλέον κάθε δρώντας απεικονίζεται σε ένα συνδυαστικό κύκλωμα. Αξίζει να σημειωθεί ότι οι τεχνικές υλοποίησης που χρησιμοποιούνται για την απεικόνιση γράφων ροής δεδομένων σε υλικό ή λογισμικό, μπορούν να συνδυαστούν έτσι ώστε ένα σύστημα ροής δεδομένων με πολλούς δρώντες, να μπορεί να υλοποιηθεί με τέτοιο τρόπο έτσι ώστε τμήμα των δρώντων να υλοποιείται σε υλικό, ενώ το άλλο μισό να υλοποιείται σε λογισμικό. Στο μετασχηματισμό ενός γράφου ροής δεδομένων σε υλικό, μπορούμε να χρησιμοποιήσουμε κάποιες τεχνικές (μετασχηματισμούς), προκειμένου να αυξήσουμε την απόδοση, όπως είναι για παράδειγμα η επέκταση πολλαπλών ρυθμών, ο επαναχρονισμός, η διοχέτευση και το άπλωμα.

δ) Ένα σύνηθες μοντέλο ελέγχου για την περιγραφή του υλικού είναι η μηχανή πεπερασμένων καταστάσεων (**FSM – Finite State Machine**) που μπορεί να χρησιμοποιηθεί για την περιγραφή της παραγωγής αλληλουχιών (Sequencing) και της λήψης αποφάσεων. Μια FSM χαρακτηρίζεται από σύνολο καταστάσεων, ένα σύνολο εισόδων και εξόδων και συνάρτηση μετάβασης καταστάσεων ή συνάρτηση εξόδου. Με τον όρο μηχανή πεπερασμένων καταστάσεων με διαδρομή δεδομένων (**FSMD**) εννοούμε τον παραγόμενο **συνδυασμό** της διαδρομής δεδομένων και της γραμμής πεπερασμένων καταστάσεων. Σε αντίθεση με τις διαδρομές δεδομένων που χρησιμοποιούν μόνο μπλοκ always, οι διαδρομές δεδομένων FSMD ορίζουν επίσης μια ή περισσότερες εντολές. Ο ελεγκτής FSM αποφασίζει το χρονοδιάγραμμα των εντολών που εκτελούνται στη διαδρομή δεδομένων, για παράδειγμα στην περίπτωση ενός αμφίδρομου μετρητή, κάθε εντολή διαδρομής δεδομένων περιέχει μόνο μια έκφραση και η FSM επιλέγει μια εντολή διαδρομής δεδομένων για εκτέλεση κατά τη διάρκεια κάθε κύκλου ρολογιού. Οι FSMD είναι χρήσιμες επειδή καταγράφουν την ροή ελέγχου και την ροή δεδομένων στο υλικό και επιπλέον μια FSMD που αποτυπώνεται ως μια ενιαία γραμμή δεδομένων είναι καλή για σχεδιασμούς με απλό ή καθόλου έλεγχο χρονοδρομολόγησης και μπορεί να αποτυπωθεί για οποιαδήποτε κατάλληλη γλώσσα ροής υλικού.

ε) Περιληπτικά, η αρχιτεκτονική ενός SoC περιέχει 4 διαστάσεις που είναι έλεγχος, επικοινωνία, υπολογισμός και αποδήμευση. Ένα πρώτο προεξέχων χαρακτηριστικό μιας αρχιτεκτονικής SoC, είναι η ετερογενής και κατανεμημένη επεξεργασία δεδομένων. Πιο συγκεκριμένα, ένα SoC, μπορεί να περιέχει πολλαπλές, ανεξάρτητες και κατανεμημένες υπολογιστικές μονάδες. Επιπλέον, αυτές οι μονάδες μπορεί να είναι ετερογενείς, καθώς μπορούν να περιλαμβάνουν FSMD, μικροπρογραμματιζόμενες μηχανές ή μικροεπεξεργαστές. Οι ετερογενείς και κατανεμημένες επικοινωνίες SoC, δίνουν τη

δυνατότητα σε έναν σχεδιαστή να εκμεταλλευτεί το εύρος ζώνης επικοινωνίας στο ολοκληρωμένο. Στη σύγχρονη τεχνολογία αυτό το εύρος ζώνης είναι εξαιρετικά υψηλό. Αντί για μία μονή κεντρική μνήμη, ένα SoC θα χρησιμοποιήσει μια συλλογή από αφοσιωμένες μνήμες. Οι επεξεργαστές και οι μικρο-κωδικοποιημένες μηχανές μπορούν να περιέχουν τοπικές μνήμες εντολών. Οι επεξεργαστές μπορούν επίσης να χρησιμοποιούν κρυφές μνήμες για να διατηρούν τα τοπικά αντίγραφα των δεδομένων και των εντολών. Τα εξαρτήματα ενός SoC λαμβάνουν εντολές από ένα κεντρικό σημείο ελέγχου και μπορούν να λειτουργούν σχεδόν ανεξάρτητα μεταξύ τους, αλλά σε κάποιο σημείο πρέπει να συγχρονίζονται και να δίνουν αναφορές στο κεντρικό σημείο ελέγχου. Ο συγχρονισμός μπορεί να επιτευχθεί είτε με σηματοφόρους, είτε με χειραψία (μονόδρομη – αμφίδρομη).

Ερώτηση 2 (Θέμα Β)

Για το σύστημα του προηγμένου επεξεργαστή του θέματος Α που απαντήσατε, να αιτιολογήσετε τα κριτήρια των αποφάσεων σας, σχετικά με τα ζητήματα: α) Περιγραφή εφαρμογής και λειτουργίας: τεχνικές προδιαγραφές. β) Μοντελοποίηση και Μετασχηματισμός ροής δεδομένων. γ) Μηχανή πεπερασμένων καταστάσεων. δ) Αρχιτεκτονική υλικού. ε) Αρχιτεκτονική λογισμικού

(Γενικές απαντήσεις, γενικές περιγραφές και απαντήσεις εκτός θέματος δεν αξιολογούνται. Να είστε όσο πιο ακριβείς συγκριτικά με την ύλη του μαθήματος μπορείτε. Τα θέματα είναι ισοδύναμα)

Λύση

α) Η επιλογή της σωστής προσέγγισης των τεχνικών προδιαγραφών περιλαμβάνει την αντιστάθμιση πολλών παραγόντων συμπεριλαμβανομένου του απαιτούμενου εύρους επικοινωνίας, της πολυπλοκότητας συνδυασμού, της εξατομικευμένης διασύνδεσης υλικού, του λογισμικού, του διαθέσιμου χρόνου σχεδιασμού και του συνολικού προϋπολογισμού κόστους.

β) Η μοντελοποίηση ροής δεδομένων προκαλεί την εύκολη κατανόηση της τεχνικής σχεδιασμού και μοντελοποίησης και είναι πολύ δημοφιλής για εφαρμογές επεξεργασίας σήματος ή σε οποιαδήποτε εφαρμογή, όπου άπειρες ουρές δειγμάτων σήματος μπορούν να ληφθούν ως ουρές συμβόλων. Η μοντελοποίηση ροής δεδομένων έχει υψηλή συσχέτιση με τη συσχεδίαση λογισμικού/υλικού λόγω του σαφούς και καθαρού τρόπου με τον οποίο αποτυπώνει τις προδιαγραφές του συστήματος.

γ) Με τη βοήθεια μιας FSMD ο σχεδιαστής καθορίζει την ποσότητα της εργασίας που γίνεται σε έναν κύκλο ρολογιού και αυτή είναι το σύνολο όλων των εντολών διαδρομής δεδομένων που θα εκτελούνται ταυτόχρονα σε έναν κύκλο ρολογιού. Ως εκ τούτου προκειμένου να επιτευχθεί το καλύτερο δυνατόν μοίρασμα, πρέπει να διανείμει παρόμοιες λειτουργίες σε πολλαπλούς τύπους ρολογιού. Οι μηχανές πεπερασμένων καταστάσεων είναι κατάλληλες για την αποτύπωση της ροής ελέγχου και των αλγορίθμων λήψης απόφασης. Τα διαγράμματα μετάβασης καταστάσεων των FSM μοιάζουν με τους γράφους εξαρτήσεων ελέγχου (CDG – Control Dependency Graph).

δ) Ο στόχος της αρχιτεκτονικής υλικού είναι διπλός: αφενός εξειδίκευση της πλατφόρμας που εξασφαλίζει την υψηλότερη αποδοτικότητα της επεξεργασίας (συνεπώς χαμηλότερης κατανάλωσης ενέργειας) και αφετέρου ευελιξία της πλατφόρμας που εξασφαλίζει μια επαναχρησιμοποιήσιμη λύση που λειτουργεί σε πολλαπλές εφαρμογές.

ε) Τα κριτήρια για την επιλογή της κατάλληλης αρχιτεκτονικής λογισμικού είναι:

- Η σχεδιαστική πολυπλοκότητα :μια συνήθης προσέγγιση είναι να διατηρείται η υλοποίηση όσο το δυνατόν πιο ευέλικτη χρησιμοποιώντας προγραμματιζόμενους επεξεργαστές που εκτελούν λογισμικό .
- **Κόστος σχεδιασμού:** Οι σχεδιαστές υλικού κάνουν τα ολοκληρωμένα προγραμματιζόμενα να επαναχρησιμοποιηθούν για πολλαπλά προϊόντα ή γενιές προϊόντων.
- **Συρρίκνωση χρονοδιαγράμματος σχεδιασμού:** απαιτεί οι ομάδες μηχανικών να δουλεύουν σε πολλαπλές εργασίες ταυτόχρονα. Το υλικό και το λογισμικό αναπτύσσονται ταυτόχρονα.

Επίλυση θεμάτων Ιουνίου 2020 (μέσα από το βιβλίο του μαθήματος)

Ερώτηση 1 (Ελεύθερου Κειμένου — 5 βαθμοί)

Να αναπτύξετε και να περιγράψετε λεπτομερώς ένα σύστημα προηγμένου επεξεργαστή: α) Περιγραφή εφαρμογής και λειτουργίας: τεχνικές προδιαγραφές. β) Μοντελοποίηση. γ) Μετασχηματισμός ροής δεδομένων. δ) Μηχανή πεπερασμένων καταστάσεων. ε) Αρχιτεκτονική.

(Γενικές απαντήσεις, γενικές περιγραφές και απαντήσεις εκτός θέματος, δεν αξιολογούνται. Να είστε όσο πιο ακριβείς, συγκριτικά με την ύλη του μαθήματος μπορείτε. Τα θέματα είναι ισοδύναμα.)

Λύση

α) Ένα σύστημα προηγμένου μικροεπεξεργαστή είναι ένα SoC (System on Chip) που συνδυάζει διάφορα εξαρτήματα σε ένα σύστημα διαύλου. Στη συνέχεια, **αναφορικά με τις τεχνικές προδιαγραφές**, να γραφεί η πρώτη παράγραφος του υποκεφαλαίου 8.1, που ξεκινά από τη δεύτερη γραμμή με τη φράση: Συνδυάζει διάφορα εξαρτήματα.....εξαρτήματος – προς – εξάρτημα. Στη συνέχεια, **αναφορικά με τις προδιαγραφές**, να γραφεί η δεύτερη παράγραφος του υποκεφαλαίου 8.1, που ξεκινά με τη φράση: *Η περιοχή εφαρμογής επηρεάζει*μετασχηματισμούς χρωμάτων εικόνας (να γραφούν τα σημαντικότερα σημεία αυτής της παραγράφου).

β) **Σελ. 74 βιβλίου** η πρώτη παράγραφος του υποκεφαλαίου 2.6 (*Τα μοντέλα ροής δεδομένων εκφράζουν ανά επανάληψη (ή κλήση)*). Επίσης, **σελ. 64 βιβλίου:** συνοπτικά οι δύο παράγραφοι με κουκκίδες του υποκεφαλαίου 2.4.2.

γ) **Σελ. 125 βιβλίου:** η πρώτη παράγραφος του υποκεφαλαίου 4.4. Επιπλέον, **Σελ. 131 βιβλίου: από το 1. να γραφούν οι τρεις πρώτες γραμμές** (*έωςνα ενημερώσουν τον καταχωρητή*). Από το 2. να γραφούν οι δύο πρώτες γραμμές και από το 3. να γραφούν οι τέσσερις πρώτες γραμμές (*έως ενός συνδυαστικού κυκλώματος*). Επίσης, σελ. 139 (*τελευταία γραμμή*) η φράση: «*Η ροή δεδομένων και η ροή ελέγχου μπορεί να αλλάξει δραστικά*» (σελ. 140, πάνω μέρος).

δ) **Σελ. 134 βιβλίου:** Από την **τρίτη** γραμμή της πρώτης παραγράφου αυτής της σελίδας έως την **έκτη** γραμμή της ίδιας παραγράφου (*Ο παραγόμενος συνδυασμός- FSMD*). Επίσης, να γραφεί και η τελευταία παράγραφος της σελ. 163 (*Μια μηχανή πεπερασμένων καταστάσεωνεκτελούμενο μπλοκ always*). Τέλος, να γραφεί και η φράση: «*το μοντέλο FSMD είναι γενικό και μπορεί να αποτυπωθεί σε οποιαδήποτε κατάλληλη γλώσσα προγραμματίσου.*»



ε) Σελ. 308 - 309 βιβλίου: Περιληπτικά η αρχιτεκτονική ενός SoC με τις τέσσερις διαστάσεις της: έλεγχος, επικοινωνία, υπολογισμός και αποθήκευση έωςνα έχει ή να μην έχει τοπική μνήμη εντολών. Επίσης, η πρώτη παράγραφος του υποκεφαλαίου 8.2.1 (*Ένα πρώτο προεξέχον χαρακτηριστικό μηχανές ή μικροεπεξεργαστές*), οι τρεις πρώτες γραμμές της δεύτερης παραγράφου του υποκεφαλαίου 8.2.2 (*Οι ετερογενείς και κατανεμημένες εξαιρετικά υψηλό*), οι πρώτες οκτώ γραμμές της πρώτης παραγράφου του υποκεφαλαίου 8.2.3 (*Ένα τρίτο χαρακτηριστικό χρησιμοποιούν τοπικά αρχεία καταχωρητών*) και τέλος να γραφούν οι τέσσερις πρώτες γραμμές της πρώτης παραγράφου του υποκεφαλαίου 8.2.4 (*Η τελική ιδέα στην αρχιτεκτονική ενός SoC κεντρικό σημείο ελέγχου*).

Ερώτηση 1 (Ελεύθερος Κειμένου — 5 βαθμοί)

Για το σύστημα του προηγμένου επεξεργαστή, του Θέματος Α, που απαντήσατε, για αιτιολογήστε τα κριτήρια των αποφάσεων σας, σχετικά με τα ζητήματα: α) Περιγραφή εφαρμογής και λειτουργίας: τεχνικές προδιαγραφές. β) Μοντελοποίηση και Μετασχηματισμός ροής δεδομένων. γ) Μηχανή πεπερασμένων καταστάσεων. δ) Αρχιτεκτονική Υλικού. ε) Αρχιτεκτονική Λογισμικού. Καλή επιτυχία και Καλό καλοκαίρι !

(Γενικές απαντήσεις, γενικές περιγραφές και απαντήσεις εκτός θέματος, δεν αξιολογούνται. Να είστε όσο πιο ακριβείς, συγκριτικά με την ύλη του μαθήματος μπορείτε. Τα θέματα είναι ισοδύναμα.)

Τελευταίο Θέμα.

Λύση

α) **Σελ. 311 βιβλίου**, η τελευταία παράγραφος (*δεν υπάρχει μόνο ένας καλός τρόπος και του συνολικού προϋπολογισμού κόστους*).

β) **Σελ. 75 βιβλίου: η τρίτη παράγραφος** (*Η μοντελοποίηση της ροής δεδομένων αποτυπώνει τις προδιαγραφές του συστήματος*).

γ) Ο ελεγκτής FSM αποφασίζει για το χρονοδιάγραμμα των εντολών στη διαδρομή δεδομένων και μια μηχανή πεπερασμένων καταστάσεων με διαδρομή ελέγχου (FSMD) είναι ιδιαίτερα χρήσιμη, καθότι τόσο τη ροή ελέγχου όσο και τη ροή δεδομένων στο υλικό. Μια FSMD που αποτυπώνεται ως μια ενιαία διαδρομή δεδομένων είναι καλή για σχεδιασμούς με απλό ή καθόλου έλεγχο χρονοδρομολόγησης, όπως οι σχεδιασμοί με απαιτήσεις υψηλής απόδοσης.

δ) **Σελ. 307 - 308 βιβλίου:** περιληπτικά τα χαρακτηριστικά ενός SoC που είναι η **εξειδίκευση** και η **ευελιξία** της πλατφόρμας, που βρίσκονται στο τέλος της **σελίδας 307 του βιβλίου και στην αρχή της σελίδας 308**. **Επιπλέον**, για το ίδιο θέμα, να γραφούν **συνοπτικά** οι τρείς παράγραφοι με κουκίδες, που υπάρχουν στη σελ. 19 του βιβλίου, που αναφέρονται στα κριτήρια (απόδοση, ενεργειακή αποδοτικότητα και πυκνότητα ισχύος) που συνηγορούν και επηρεάζουν την αρχιτεκτονική υλικού.

ε) **Σελ. 112 - 113 βιβλίου:** η τελευταία παράγραφος, που συνεχίζει και στη σελίδα 113. **Επιπλέον**, για το ίδιο θέμα, να γραφεί και η φράση: «*Η ροή δεδομένων υπερέχει στην περιγραφή της επεξεργασίας συνεχούς ροής και συνεπώς παραμένει πολύ δημοφιλής στις εφαρμογές επεξεργασίας σήματος*». **Επιπλέον**, να γραφούν



συνοπτικά οι τρείς παράγραφοι με κουκκίδες, που υπάρχουν στη **σελ. 20** του βιβλίου, που αναφέρονται στα κριτήρια (σχεδιαστική πολυπλοκότητα, κόστος σχεδιασμού και συρρίκνωση χρονοδιαγράμματος σχεδιασμού) που συνηγορούν και επηρεάζουν την αρχιτεκτονική λογισμικού.

Ισχυρή σύσταση: επειδή αυτά θα τα έχουν πολλοί φοιτητές, **ΝΑ ΤΑ ΑΛΛΑΖΕΤΕ ΟΣΟ ΜΠΟΡΕΙΤΕ.**



36 Επίλυση Θεμάτων Σεπτεμβρίου 2020 – Φεβρουάριος 2021

Θέμα Α

Θεωρείστε τη μικροεντολή $\alpha = 2 * \alpha$. Για τα δύο τελευταία ψηφία του αριθμού μητρώου σας K1Κ0, να υλοποιήσετε τον κατακόρυφο μικροπρογραμματισμό ΚΟΚ1, για την εντολή και τον αντίστοιχο οριζόντιο.

Λύση

Έστω ότι το **ΑΜ = 1055476**. Θέλουμε για τα δύο τελευταία ψηφία του ΑΜ να υλοποιήσουμε τον κατακόρυφο μικροπρογραμματισμό 67. Θα θεωρήσουμε ότι αυτό είναι το περιεχόμενο του καταχωρητή "α" Για τον **κατακόρυφο μικροκώδικα** θα χρησιμοποιήσουμε το ζεύγος των δυαδικών ψηφίων **00** και αυτά τα δύο bits θα τα στείλουμε σε κάποιον αποκωδικοποιητή, ο οποίος θα παράγει τα κατάλληλα σήματα ελέγχου. Θα έχουμε έναν καταχωρητή, ο οποίος θα αποθηκεύει την τιμή της μεταβλητής «α», η έξοδός του θα πάει σε έναν αθροιστή, του οποίου η δεύτερη είσοδος θα είναι η μεταβλητή «α» προκειμένου να πραγματοποιηθεί το άθροισμα $\alpha + \alpha = 2 * \alpha$. Όσον αφορά τον **οριζόντιο μικροκώδικα** θα χρησιμοποιήσουμε μια ακολουθία από 3 δυαδικά ψηφία με τιμές 000, προκειμένου ο αθροιστής του κυκλώματος να εκτελέσει πρόσθεση (θεωρούμε ότι το «0» αντιστοιχεί στην πρόσθεση και το «1» στην αφαίρεση στον αθροιστή αυτό). Επίσης, θέλουμε να επιλεγεί από τον πολυπλέκτη του κυκλώματος η είσοδος «0», που αντιστοιχεί στην τιμή «α», προκειμένου να φτάσουν στον αθροιστή οι τιμές «α» και «α» για να γίνει το άθροισμα $\alpha + \alpha$. Επίσης θέλουμε από τον 2^ο πολυπλέκτη του κυκλώματος να μην περάσει κάποια εξωτερική είσοδος (IN), αλλά η είσοδος «α», που θα αποθηκευτεί στη συνέχεια στον καταχωρητή.

Θέμα Β

α) Να σχεδιάσετε και να υλοποιήσετε το γράφο της ροής δεδομένων της λύσης που δώσατε στο Θέμα Α (Προσοχή: δεν θα πρέπει να απαντήσετε σε γενικές κατευθύνσεις (οι οποίες δεν αξιολογούνται), αλλά ακριβώς για τη λύση που εσείς δώσατε στο Θέμα Α). β) Με βάση το γράφο ροής δεδομένων που απαντήσατε στο ερώτημα (α) να υλοποιήσετε τη ροή δεδομένων σε υλικό και λογισμικό και γ) Να αναδείξετε τα πλεονεκτήματα και τα μειονεκτήματα της νέας υλοποίησης του ερωτήματος (β)

Λύση

α) Οι ακμές δεδομένων μπορούν να απεικονιστούν σε ένα γράφο, όπου κάθε λειτουργία αντιπροσωπεύει έναν κόμβο και ουσιαστικά εκφράζουν μια σχέση μεταξύ δύο κόμβων για μια συγκεκριμένη μεταβλητή. Μια ακμή δεδομένων αντανακλά μια απαίτηση για τη ροή των πληροφοριών. Ο γράφος της ροής δεδομένων για τη συγκεκριμένη μικροεντολή αποτελείται από την λειτουργία 1, (που είναι διάβασμα της μεταβλητής «α» στο μικροπρόγραμμα), τη λειτουργία 2 (που είναι το άθροισμα $2 * \alpha$) και την λειτουργία 3 (που είναι η επιστροφή του αποτελέσματος). Οι ακμές δεδομένων ορίζονται μεταξύ των αντίστοιχων λειτουργιών παραγωγής/κατανάλωσης. Για παράδειγμα, η λειτουργία (1) ορίζει την τιμή της μεταβλητής «α». Η τιμή του «α» χρησιμοποιείται από τη λειτουργία (2), επομένως θα υπάρχει μια ακμή δεδομένων από τη λειτουργία (1) στη λειτουργία (2). Επίσης, θα υπάρχει μια ακμή δεδομένων από τη λειτουργία (2) στη λειτουργία (3). β) Μια ακμή δεδομένων πρέπει πάντα να υλοποιείται, ανεξάρτητα από την υποκείμενη αρχιτεκτονική. Για την υλοποίηση της προηγούμενης ροής δεδομένων σε λογισμικό, θα χρησιμοποιήσουμε το γράφημα DFG με ακμές δεδομένων. Το κύκλωμα της ροής δεδομένων και η μεταβλητή καταχωρητή συνδέονται με βάση τις ακμές δεδομένων του DFG. Κάθε

ακμή δεδομένων συνδέει έναν καταχωρητή με την είσοδο ενός συνδυαστικού κυκλώματος. Τέλος, συνδέουμε τις εισόδους και τις εξόδους του συστήματος σε εισόδους κυκλωμάτων της διαδρομής δεδομένων και τις εξόδους καταχωρητή αντίστοιχα. Για να βρούμε τις ακμές δεδομένων, θα εξετάσουμε τα μοντέλα παραγωγής/κατανάλωσης κάθε λειτουργίας που προαναφέρθηκε. Για τη λειτουργία 1 έχουμε παραγωγή της μεταβλητής α , για τη λειτουργία 2 έχουμε κατανάλωση της μεταβλητής α και για τη λειτουργία 3, που είναι η επιστροφή του αποτελέσματος, έχουμε κατανάλωση της μεταβλητής α . Για την υλοποίηση της προηγούμενης ροής δεδομένων σε **υλικό**, θα χρησιμοποιήσουμε έναν **καταχωρητή** για την αποθήκευση της μεταβλητής « α » και έναν αθροιστή για την πραγματοποίηση του αθροίσματος $\alpha + \alpha = 2 * \alpha$ (συνδυαστικό κύκλωμα). Η ολοκλήρωση της άθροισης θα γίνεται σε έναν κύκλο ρολογιού.

γ) Αν χρησιμοποιήσουμε ένα πρόγραμμα **μοναδικής ανάθεσης** (*single – assignment program*) θα μπορούσαμε να εκτελέσουμε όλες τις λειτουργίες της μικροεντολής $\alpha = 2 * \alpha$ σε έναν μόνο κύκλο ρολογιού. Το **πλεονέκτημα** είναι ότι η ροή δεδομένων διατηρείται σε διαφορετικές υλοποιήσεις στο υλικό και το λογισμικό. Όσον αφορά το **μειονέκτημα**, στη σύνθεση υψηλού επιπέδου, υπάρχουν τα προβλήματα σχεδιασμού που σχετίζονται με την υλοποίηση και τη διαχείριση των στοιχείων μνήμης (memory management).

Θέμα Α (Άλλη ομάδα)

Θεωρείστε τη μικροεντολή **$\alpha = \alpha + \beta$** . Για τα δύο τελευταία ψηφία του αριθμού μητρώου σας K1K0, να υλοποιήσετε το κατακόρυφο μικροπρογραμματισμό K0K1, για την εντολή και τον αντίστοιχο οριζόντιο

Λύση

Έστω ότι το **AM = 1055476**. Θέλουμε για τα δύο τελευταία ψηφία του AM να υλοποιήσουμε τον κατακόρυφο μικροπρογραμματισμό 67. Για τον **κατακόρυφο μικροκώδικα** θα χρησιμοποιήσουμε το ζεύγος των δυαδικών ψηφίων **00** και αυτά τα δύο bits θα τα στείλουμε σε κάποιον αποκωδικοποιητή, ο οποίος θα παράγει τα κατάλληλα σήματα ελέγχου. Θα έχουμε έναν καταχωρητή, ο οποίος θα αποθηκεύει την τιμή της μεταβλητής « α », και έναν δεύτερο καταχωρητή που θα αποθηκεύει τη μεταβλητή « β ». Η έξοδος « α » του πρώτου καταχωρητή θα εισαχθεί στη συνέχεια σε έναν αθροιστή, του οποίου η δεύτερη είσοδος θα είναι η μεταβλητή « β » από ένα δεύτερο καταχωρητή, προκειμένου να πραγματοποιηθεί το άθροισμα $\alpha + \beta$. Όσον αφορά τον **οριζόντιο μικροκώδικα** θα χρησιμοποιήσουμε μια ακολουθία από 3 δυαδικά ψηφία με τιμές 000, προκειμένου ο αθροιστής του κυκλώματος να εκτελέσει πρόσθεση (θεωρούμε ότι το «0» αντιστοιχεί στην πρόσθεση και το «1» στην αφαίρεση στον αθροιστή αυτό). Επίσης, θέλουμε να επιλεγεί από τον έναν πολυπλέκτη του κυκλώματος η είσοδος «0», που αντιστοιχεί στην τιμή « α », και από το δεύτερο πολυπλέκτη η είσοδος «0», που αντιστοιχεί στην τιμή « β », προκειμένου να φτάσουν στον αθροιστή οι τιμές « α » και « β » για να γίνει το άθροισμα $\alpha + \beta$. Επίσης, θέλουμε από τον κάθε πολυπλέκτη του κυκλώματος να μην περάσει κάποια εξωτερική είσοδος (IN), αλλά η είσοδος « α » και « β » αντίστοιχα (θεωρούμε ότι αν θέσουμε τιμή «0» στη γραμμή επιλογής του κάθε πολυπλέκτη, θα επιλεγεί η είσοδος « α » και « β » αντίστοιχα και όχι κάποια εξωτερική είσοδος IN), που θα αποθηκευτούν στη συνέχεια σε αντίστοιχους καταχωρητές, προ σταλθούν στον αθροιστή.

Θέμα

α) Να σχεδιάσετε και να υλοποιήσετε το γράφο της ροής δεδομένων της λύσης που δώσατε στο Θέμα Α (Προσοχή: δεν θα πρέπει να απαντήσετε σε γενικές κατευθύνσεις (οι οποίες δεν αξιολογούνται), αλλά ακριβώς για τη λύση που εσείς

δώσατε στο Θέμα A). β) Με βάση το γράφο ροής δεδομένων που απαντήσατε στο ερώτημα (α) να υλοποιήσετε τη ροπή δεδομένων σε υλικό και λογισμικό και γ) Να αναδείξετε τα πλεονεκτήματα και τα μειονεκτήματα της νέας υλοποίησης του ερωτήματος (β)

Λύση

α) Οι ακμές δεδομένων μπορούν να απεικονιστούν σε ένα γράφο, όπου κάθε λειτουργία αντιπροσωπεύει έναν κόμβο και ουσιαστικά εκφράζουν μια σχέση μεταξύ δύο κόμβων για μια συγκεκριμένη μεταβλητή. Μια ακμή δεδομένων αντανακλά μια απαίτηση για τη ροή των πληροφοριών. Ο γράφος της ροής δεδομένων για τη συγκεκριμένη μικροεντολή αποτελείται από την λειτουργία 1, (που είναι διάβασμα της μεταβλητής «α» και της μεταβλητής «β» στο μικροπρόγραμμα), τη λειτουργία 2 (που είναι το άθροισμα $\alpha + \beta$) και την λειτουργία 3 (που είναι η επιστροφή του αποτελέσματος). Οι ακμές δεδομένων ορίζονται μεταξύ των αντίστοιχων λειτουργιών παραγωγής/κατανάλωσης. Για παράδειγμα, η λειτουργία (1) ορίζει την τιμή της μεταβλητής «α» και της μεταβλητής «β». Η τιμή του «α» και του «β» χρησιμοποιείται από τη λειτουργία (2), επομένως θα υπάρχει μια ακμή δεδομένων από τη λειτουργία (1) στη λειτουργία (2). Επίσης, θα υπάρχει μια ακμή δεδομένων από τη λειτουργία (2) στη λειτουργία (3).

β) Μια ακμή δεδομένων πρέπει πάντα να υλοποιείται, ανεξάρτητα από την υποκείμενη αρχιτεκτονική. Για την υλοποίηση της προηγούμενης ροής δεδομένων σε **λογισμικό**, θα χρησιμοποιήσουμε το γράφημα DFG με ακμές δεδομένων. Το κύκλωμα της ροής δεδομένων και η μεταβλητή καταχωρητή συνδέονται με βάση τις ακμές δεδομένων του DFG. Κάθε ακμή δεδομένων συνδέει έναν καταχωρητή με την είσοδο ενός συνδυαστικού κυκλώματος. Τέλος, συνδέουμε τις επίσης τις εισόδους και τις εξόδους συστήματος σε εισόδους κυκλωμάτων της διαδρομής δεδομένων και τις εξόδους καταχωρητή αντίστοιχα. Για να βρούμε τις ακμές δεδομένων, θα εξετάσουμε τα μοντέλα παραγωγής/κατανάλωσης κάθε λειτουργίας που προαναφέρθηκε. Για τη λειτουργία 1 έχουμε παραγωγή της μεταβλητής «α» και της μεταβλητής «β», για τη λειτουργία 2 έχουμε κατανάλωση της μεταβλητής «α» και της «β» και για τη λειτουργία 3, που είναι η επιστροφή του αποτελέσματος (αθροίσματος), έχουμε κατανάλωση της μεταβλητής «α» και της «β». Για την υλοποίηση της προηγούμενης ροής δεδομένων σε **υλικό** θα χρησιμοποιήσουμε έναν καταχωρητή για την αποθήκευση της μεταβλητής «α» και έναν καταχωρητή για την αποθήκευση της μεταβλητής «β» καθώς και έναν αθροιστή για την πραγματοποίηση του αθροίσματος $\alpha + \beta$ (συνδυαστικό κύκλωμα).. Η ολοκλήρωση της άθροισης θα γίνεται σε έναν κύκλο ρολογιού.

γ) Αν χρησιμοποιήσουμε ένα πρόγραμμα μοναδικής ανάθεσης (single – assignment program) θα μπορούσαμε να εκτελέσουμε όλες τις λειτουργίες της μικροεντολής $\alpha + \beta$ σε έναν μόνο κύκλο ρολογιού. Το πλεονέκτημα είναι ότι η ροή δεδομένων διατηρείται σε διαφορετικές υλοποιήσεις στο υλικό και το λογισμικό. Όσον αφορά το μειονέκτημα, στη σύνθεση υψηλού επιπέδου, υπάρχουν τα προβλήματα σχεδιασμού που σχετίζονται με την υλοποίηση των στοιχείων μνήμης (memory management).

Θέμα Γ

Θεωρείστε ένα σύστημα προηγμένου μικροεπεξεργαστή, που εκτελεί δύο διεργασίες: **διαίρεση και λογικό XOR**. α) Να γράψετε τον αντίστοιχο ψευδοκώδικα για κάθε διεργασία. β) Να προτείνετε τη συσχεδίαση υλικού και λογισμικού για την καλύτερη δυνατή υλοποίηση του προβλήματος και να αιτιολογήσετε αναλυτικά, την πρότασή σας.

Λύση

α) Έστω ότι γράφουμε για τη διαίρεση μια συνάρτηση σε ψευδοκώδικα, η οποία δέχεται δύο ακεραίους a, b και επιστρέφει το πιλίκο τους a/b.

συνάρτηση division(a, b)	//Λειτουργία 1 – είσοδος στη συνάρτηση
μεταβλητές	
ακέραιες: r	
αν (b!=0) τότε	//Λειτουργία 2 – αν - τότε
r = a/b	//Λειτουργία 3 – καταχώρηση αποτελέσματος
τέλος_αν	
επέστρεψε r	//Λειτουργία 4 – επιστροφή αποτελέσματος

Έστω ότι στη συνέχεια γράφουμε για το λογικό XOR μια συνάρτηση ένα ψευδοκώδικα, η οποία δέχεται δύο ακεραίους a, b και επιστρέφει τη λογική πράξη XOR ανάμεσά τους.

συνάρτηση xor(a, b)	//Λειτουργία 1 – είσοδος στη συνάρτηση
ακέραιες: r	
r = a xor b	//Λειτουργία 2 – καταχώρηση αποτελέσματος
επέστρεψε r	//Λειτουργία 3 – επιστροφή αποτελέσματος

β) Μια συσχεδίαση υλικού και λογισμικού για τον κάθε ψευδοκώδικα θα μπορούσε να είναι η εξής: Μια υλοποίηση ακμών δεδομένων (DFG) και ελέγχου (CFG), όπου μια ακμή δεδομένων πρέπει πάντα να υλοποιείται ανεξάρτητα από την υποκείμενη αρχιτεκτονική, ενώ μια ακμή ελέγχου μπορεί να απομακρυνθεί αν η υποκείμενη αρχιτεκτονική μπορεί να διαχειριστεί τον παραγόμενο ταυτοχρονισμό. Πιο συγκεκριμένα, για τον πρώτο ψευδοκώδικα έχουμε τη λειτουργία 1 (είσοδος στη συνάρτηση), τη λειτουργία 2 (αν – τότε), τη λειτουργία 3 (καταχώρηση αποτελέσματος) και τη λειτουργία 4 (επιστροφή αποτελέσματος). Πρώτα θα κατασκευάσουμε το γράφο CFG όπου η ακολουθία ακμών είναι $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$, στην οποία ο κόμβος 1 είναι ο κόμβος - ανάγνωση των μεταβλητών a, b, που θα διαιρεθούν, ο κόμβος 2 εκτελεί τη σύγκριση της μεταβλητής b με το «0» και ο κόμβος 3 είναι ο κόμβος – εγγραφή του αποτελέσματος. Στη συνέχεια θα κατασκευάσουμε το γράφο DFG στον οποίο θα πρέπει να γράψουμε ακμές δεδομένων για κάθε κόμβο που τροποποιούν τη μεταβλητή r. Για να γίνει αυτό πρέπει να εντοπίσουμε όλες τις εκφράσεις συνθηκών που επηρεάζουν το αποτέλεσμα του κόμβου που περιέχει τη μεταβλητή r. Στη συνέχεια θα πρέπει να μεταφράσουμε τη ροή δεδομένων και τη ροή ελέγχου που γράψαμε σε υλικό. Κάθε μεταβλητή που εμφανίζεται μέσα στον ψευδοκώδικα μεταφράζεται σε έναν καταχωρητή με έναν πολυπλέκτη μπροστά του. Ο πολυπλέκτης είναι αναγκαίος όταν πολλές πηγές μπορούν να ενημερώσουν τον καταχωρητή. Για κάθε έκφραση του ψευδοκώδικα ενσωματωμένη σε έναν κόμβο του CFG θα δημιουργήσουμε ένα ισοδύναμο συνδυαστικό κύκλωμα για να υλοποιήσουμε αυτή την έκφραση. Το συνδυαστικό κύκλωμα που απαιτείται στον πρώτο ψευδοκώδικα είναι ένας διαιρέτης. Συνολικά χρειαζόμαστε τρεις καταχωρητές για κάθε μία από τις μεταβλητές a, b και r. Η έκφραση συνθήκης για την εντολή «αν» απαιτεί ένα συγκριτής μεγέθους. Επομένως συνολικά το υλικό περιέχει 3 καταχωρητές με τους αντίστοιχους πολυπλέκτες, ένα συγκριτής και ένα κύκλωμα που εκτελεί την διαίρεση μεταξύ των μεταβλητών a και b.

Για το δεύτερο ψευδοκώδικα έχουμε τη λειτουργία 1 (είσοδος στη συνάρτηση), τη λειτουργία 2 (καταχώρηση αποτελέσματος) και την λειτουργία 3 (επιστροφή αποτελέσματος). Θα κατασκευάσουμε και πάλι πρώτα τον γράφο CFG όπου η ακολουθία ακμών είναι $1 \rightarrow 2 \rightarrow 3$, στην οποία ο κόμβος 1 είναι ο κόμβος - ανάγνωση των μεταβλητών a, b, μεταξύ των οποίων θα λάβει χώρα η πράξη XOR, ο κόμβος 2 στον οποίο γίνεται η εγγραφή του αποτελέσματος και ο κόμβος 3 που είναι ο κόμβος για την επιστροφή του αποτελέσματος. Σε αυτήν την περίπτωση αυτή θα χρειαστούμε και πάλι 3



καταχωρητές για κάθε μια από τις μεταβλητές a , b , r που εμφανίζονται στον κώδικα με τους αντίστοιχους πολυπλέκτες μπροστά τους. Το συνδυαστικό κύκλωμα που απαιτείται για την υλοποίηση της έκφρασης $r = a \text{ xor } b$ είναι αυτό που υλοποιεί την συγκεκριμένη λογική πράξη. Θα μπορούσαμε να σχεδιάσουμε έναν ελεγκτή ο οποίος θα περιέχει έναν συνδυασμό της κάθε μίας διαδρομής δεδομένων (για κάθε έναν από τους προηγούμενους ψευδοκώδικες) και μια μηχανής πεπερασμένων καταστάσεων, με λίγα λόγια να υλοποιήσουμε τις 2 διεργασίες που αναφέρονται στην εκφώνηση με έναν FSMD. Για βελτίωση του υλικού θα μπορούσε να γίνει χρήση προγραμμάτων μοναδικής-ανάθεσης, οπού σε αυτή την προσέγγιση κάθε εντολή του ψευδοκώδικα χρειάζεται έναν μόνο κύκλο ρολογιού για να εκτελεστεί, επειδή κάθε μεταβλητή έχει απεικονιστεί σε έναν καταχωρητή. Επιπλέον θα γίνει χρήση της συνάρτησης `merge` που μπορεί να συγχωνεύσει πολλαπλές ακμές δεδομένων σε μία.

Θέμα Γ (άλλη ομάδα)

Θεωρείστε ένα σύστημα προηγμένου μικροεπεξεργαστή, που εκτελεί δύο διεργασίες: **πολλαπλασιασμό και λογικό NOT**.

α) Να γράψετε τον αντίστοιχο ψευδοκώδικα για κάθε διεργασία. **β)** Να προτείνετε τη συσχεδίαση υλικού και λογισμικού για την καλύτερη δυνατή υλοποίηση του προβλήματος και να αιτιολογήσετε αναλυτικά, την πρότασή σας.

Λύση

α) Έστω ότι γράφουμε για τη διαίρεση μια συνάρτηση σε ψευδοκώδικα, η οποία δέχεται δύο ακεραίους a , b και επιστρέφει το πηλίκο τους a/b .

συνάρτηση multiplication(a , b) //Λειτουργία 1 – είσοδος στη συνάρτηση

μεταβλητές

ακέραιες: r

$r = a * b$

επέστρεψε r //Λειτουργία 2 – καταχώρηση αποτελέσματος

//Λειτουργία 3 – επιστροφή αποτελέσματος

Έστω ότι στη συνέχεια γράφουμε για το λογικό NOT μια συνάρτηση ένα ψευδοκώδικα, η οποία δέχεται έναν ακέραιο a και επιστρέφει το συμπλήρωμα του a .

συνάρτηση not(a) //Λειτουργία 1 – είσοδος στη συνάρτηση

ακέραιες: r

$r = \text{not } a$ //Λειτουργία 2 – καταχώρηση αποτελέσματος

επέστρεψε r //Λειτουργία 3 – επιστροφή αποτελέσματος

β) Μια συσχεδίαση υλικού και λογισμικού για τον κάθε ψευδοκώδικα θα μπορούσε να είναι η εξής: Μια υλοποίηση ακμών δεδομένων (DFG) και ελέγχου (CFG), όπου μια ακμή δεδομένων πρέπει πάντα να υλοποιείται ανεξάρτητα από την υποκείμενη αρχιτεκτονική, ενώ μια ακμή ελέγχου μπορεί να απομακρυνθεί αν η υποκείμενη αρχιτεκτονική μπορεί να διαχειριστεί τον παραγόμενο ταυτοχρονισμό. Πιο συγκεκριμένα, για τον πρώτο ψευδοκώδικα έχουμε τη λειτουργία 1 (είσοδος στη συνάρτηση), τη λειτουργία 2 (καταχώρηση αποτελέσματος) και τη λειτουργία 3 (επιστροφή αποτελέσματος). Πρώτα θα κατασκευάσουμε το γράφο CFG όπου η ακολουθία ακμών είναι $1 \rightarrow 2 \rightarrow 3$, στην οποία ο κόμβος 1 είναι ο κόμβος - ανάγνωση των μεταβλητών a , b , που θα πολλαπλασιαστούν, ο κόμβος 2 είναι ο κόμβος – εγγραφή του αποτελέσματος και ο κόμβος 3 είναι για την επιστροφή του αποτελέσματος. Στη συνέχεια θα κατασκευάσουμε το γράφο DFG στον οποίο θα πρέπει να γράψουμε ακμές δεδομένων για κάθε κόμβο που τροποποιούν τη μεταβλητή r . Για να γίνει

αυτό πρέπει να εντοπίσουμε όλες τις εκφράσεις συνθηκών που επηρεάζουν το αποτέλεσμα του κόμβου που περιέχει τη μεταβλητή r. Στη συνέχεια θα πρέπει να μεταφράσουμε τη ροή δεδομένων και τη ροή ελέγχου που γράψαμε σε υλικό. Κάθε μεταβλητή που εμφανίζεται μέσα στον ψευδοκώδικα μεταφράζεται σε έναν καταχωρητή με έναν πολυπλέκτη μπροστά του. Ο πολυπλέκτης είναι αναγκαίος όταν πολλές πηγές μπορούν να ενημερώσουν τον καταχωρητή. Για κάθε έκφραση του ψευδοκώδικα ενσωματωμένη σε έναν κόμβο του CFG θα δημιουργήσουμε ένα ισοδύναμο συνδυαστικό κύκλωμα για να υλοποιήσουμε αυτή την έκφραση. Το συνδυαστικό κύκλωμα που απαιτείται στον πρώτο ψευδοκώδικα είναι ένας διαιρέτης. Συνολικά χρειαζόμαστε τρεις καταχωρητές με τους αντίστοιχους πολυπλέκτες μπροστά τους για κάθε μία από τις μεταβλητές a, b και r και ένα κύκλωμα που εκτελεί τον πολλαπλασιασμό μεταξύ των a και b.

Για το δεύτερο ψευδοκώδικα έχουμε τη λειτουργία 1 (είσοδος στη συνάρτηση), τη λειτουργία 2(καταχώρηση αποτελέσματος) και την λειτουργία 3 (επιστροφή αποτελέσματος). Θα κατασκευάσουμε και πάλι πρώτα τον γράφο CFG όπου η ακολουθία ακμών είναι $1 \rightarrow 2 \rightarrow 3$, στην οποία ο κόμβος 1 είναι ο κόμβος - ανάγνωση των μεταβλητών a, b, μεταξύ των οποίων θα λάβει χώρα η πράξη NOT, ο κόμβος 2 στον οποίο γίνεται η εγγραφή του αποτελέσματος και ο κόμβος 3 που είναι ο κόμβος για την επιστροφή του αποτελέσματος. Σε αυτήν την περίπτωση αυτή θα χρειαστούμε και πάλι 2 καταχωρητές για κάθε μια από τις μεταβλητές a, r που εμφανίζονται στον κώδικα με τους αντίστοιχους πολυπλέκτες μπροστά τους. Το συνδυαστικό κύκλωμα που απαιτείται για την υλοποίηση της έκφρασης $r = \text{not } a$ είναι αυτό που υλοποιεί την συγκεκριμένη λογική πράξη. Θα μπορούσαμε να σχεδιάσουμε έναν ελεγκτή ο οποίος θα περιέχει έναν συνδυασμό της κάθε μίας διαδρομής δεδομένων (για κάθε έναν από τους προηγούμενους ψευδοκώδικες) και μια μηχανής πεπερασμένων καταστάσεων, με λίγα λόγια να υλοποιήσουμε τις 2 διεργασίες που αναφέρονται στην εκφώνηση με έναν FSMD. Για βελτίωση του υλικού θα μπορούσε να γίνει χρήση προγραμμάτων μοναδικής-ανάθεσης, οπού σε αυτή την προσέγγιση κάθε εντολή του ψευδοκώδικα χρειάζεται έναν μόνο κύκλο ρολογιού για να εκτελεστεί, επειδή κάθε μεταβλητή έχει απεικονιστεί σε έναν καταχωρητή. Επιπλέον θα γίνει χρήση της συνάρτησης merge που μπορεί να συγχωνεύσει πολλαπλές ακμές δεδομένων σε μία.

Θέμα A (Άλλη ομάδα)

Θεωρείστε τη μικροεντολή $\alpha = \alpha + 3$. Για τα δύο τελευταία ψηφία του αριθμού μητρώου σας K1K0, να υλοποιήσετε το κατακόρυφο μικροπρογραμματισμό K0K1, για την εντολή και τον αντίστοιχο οριζόντιο

Λύση