Important Notice

This e-document is digitally certified by the author, through the **GeoTrust** certificate authority. Root certificates in the certificate chain are available at the web address http://www.geotrust.com/resources/roots/. The document requires at least version 6.0 of Adobe Acrobat Reader.

Digital Signature Placeholder (click to validate):

Digitally signed by
Georgios Asimenos
Location: Athens,
Greece

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Μεταγλωττιστής Υλικού (Hardware Compiler)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΓΕΩΡΓΙΟΥ Κ. ΑΣΗΜΕΝΟΥ

Υπεύθυνος : Γ. Παπακωνσταντίνου
Καθηγητής ΕΜΠ

Αθήνα, Ιούλιος 2003

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Μεταγλωττιστής Υλικού (Hardware Compiler)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΓΕΩΡΓΙΟΥ Κ. ΑΣΗΜΕΝΟΥ

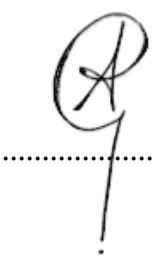
Υπεύθυνος : Γ. Παπακωνσταντίνου
Καθηγητής ΕΜΠ

Ενεκρίθη από την τριμελή εξεταστική επιτροπή την 17η Ιουλίου 2003

.....	.....	.....
Γ. Παπακωνσταντίνου	Π. Τσανάκας	Ν. Κοζύρης
Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Επ. Καθηγητής ΕΜΠ

Αθήνα, Ιούλιος 2003

“Ευχαριστώ πολύ όλους όσους με ανέχτηκαν σε αυτή μου την προσπάθεια. Αισθάνομαι ιδιαίτερα την ανάγκη να ευχαριστήσω τον υποψήφιο διδάκτορα Στέργιο Στεργίου (αυτός τα ξεκίνησε όλα· εγώ δε φταίω σε τίποτε!) και να του ευχηθώ τα καλύτερα για το μέλλον. Του είμαι ευγνώμον που βρέθηκε στο δρόμο μου. Επίσης ευχαριστώ όλους ανεξαιρέτως τους καθηγητές μου, και υπόσχομαι σε όσους με βοήθησαν να φανώ αντάξιος των προσδοκιών τους. Τέλος, ένα μεγάλο ευχαριστώ σε εκείνους του συμφοιτητές μου, που κατάφεραν να είναι οι μοναδικοί πλέον φίλοι μου.”



.....

ΓΕΩΡΓΙΟΣ Κ. ΑΣΗΜΕΝΟΣ

Πίνακας Περιεχομένων

Πίνακας Εικόνων

1

Εισαγωγή

ΖΗΝΩΝΟΣ. Ὁ Ζήνων εἶπε σὲ κάποιον ποὺ ἤθελε πιὸ πολὺ νὰ μιλάη παρὰ ν' ἀκούη: «Νεαρέ μου», τοῦ εἶπε, «ἡ φύση μᾶς ἔδωσε μιὰ γλῶσσα καὶ δυὸ αὐτιὰ, γιὰ ν' ἀκοῦμε διπλάσια ἀπ' ὅσα λέμε».

Εισαγωγή

Η παραδοσιακή προσέγγιση για την αντιμετώπιση ενός υπολογιστικού προβλήματος ή γενικότερα για την υλοποίηση ενός αλγορίθμου αποτελείται συνήθως από τα εξής βήματα:

- Προγραμματισμός σε γλώσσα υψηλού επιπέδου.
- Μεταγλώττιση και μετατροπή σε γλώσσα μηχανής (assembly).
- Ακολουθιακή εκτέλεση σε κάποιον επεξεργαστή (ή σύστημα).

Η διαδικασία αυτή όμως εισάγει κάποιους περιορισμούς για την απόδοση, όπως οι κάτωθι:

- Ο επεξεργαστής είναι συγκεκριμένος, με προκαθορισμένο ρεπερτόριο εντολών και δεν προσαρμόζεται στις ανάγκες του προβλήματος.
- Η εκτέλεση είναι ακολουθιακή με μικρές ίσως βελτιστοποιήσεις παραλληλίας από τον επεξεργαστή κατά τη διάρκεια της εκτέλεσης (runtime).

Μια ιδανική λύση για αποδοτικότερη υλοποίηση θα είχε τα εξής χαρακτηριστικά:

- Προγραμματισμός σε γλώσσα υψηλού επιπέδου. (Δηλαδή ούτως ή άλλως ο χρήστης περιγράφει τον τρόπο λύσης σε μία γλώσσα υψηλού επιπέδου που γνωρίζει καλά).
- Μεταγλώττιση του προγράμματος και σύνθεση ενός ειδικευμένου κυκλώματος—παραγωγή ενός τελικού κομματιού υλικού (hardware).
- Εκτέλεση πάνω στο hardware.

Το εγχείρημα αυτό βασίζεται σε έναν μεταγλωττιστή υλικού (hardware compiler), ένα πρόγραμμα δηλαδή που θα φροντίζει για τη μετατροπή του πηγαίου κώδικα (γραμμένου σε γλώσσα υψηλού επιπέδου) σε κυκλώματα. Το απλούστερο κύκλωμα που μπορεί να παραχθεί είναι ένα λογικό κύκλωμα που υλοποιεί Boolean πράξεις, με ακολουθιακά μέρη (flip-flops) για την αποθήκευση δεδομένων. Πιο συγκεκριμένα:

- Το κύκλωμα φτιάχνεται σύμφωνα με τις ανάγκες του προβλήματος. Το πρόγραμμα σε γλώσσα υψηλού επιπέδου ορίζει τον τρόπο λύσης ο οποίος υλοποιείται κατ' αντιστοιχία στο κύκλωμα με συγκεκριμένες δομές. Για παράδειγμα, μία άθροιση στο πρόγραμμα αντιστοιχεί σε ένα κύκλωμα αθροιστή, ενώ μια δομή while εισάγει ένα ειδικά σχεδιασμένο κύκλωμα βρόχου.
- Η εκτέλεση δεν είναι πλέον αυστηρά ακολουθιακή: Ο προγραμματιστής μπορεί να ορίσει κομμάτια κώδικα τα οποία θέλει να εκτελεστούν παράλληλα, ώστε ο μεταγλωττιστής να κατασκευάσει τα αντίστοιχα κυκλώματα που λειτουργούν ταυτόχρονα, στους ίδιους κύκλους ρολογιού.

Η ιδέα της σύνθεσης κυκλωμάτων από αλγοριθμικές περιγραφές δεν είναι κάτι το καινούριο. Οι περισσότερες γλώσσες περιγραφής υλικού (hardware description languages) προσφέρουν τη δυνατότητα της περιγραφής ενός κυκλώματος με βάση τη συμπεριφορά του (behavior), με χρήση δηλαδή εντολών όπως αυτές που περιλαμβάνουν οι γνωστές γλώσσες υψηλού επιπέδου. Έτσι ο χρήστης μπορεί να περιγράψει πως επιθυμεί να συμπεριφέρεται ένα κύκλωμα, αντί να περιγράψει την εσωτερική δομή του, και να αφήσει το λογισμικό σύνθεσης να κατασκευάσει τα τελικά κυκλώματα. Αυτή η διαδικασία της *σύνθεσης από συμπεριφορά* (behavioral synthesis) έχει αποτελέσει γενικότερο αντικείμενο μελέτης και έρευνας. Ωστόσο, οι γλώσσες περιγραφής υλικού κρατούν ως κοινό παρανομαστή το γεγονός ότι περιγράφονται κυκλώματα, και επεκτείνουν τον τρόπο περιγραφής από δομικό (structural) σε περιγραφή συμπεριφοράς (behavioral). Αντίθετα, στην παρούσα εργασία αυτό που επιχειρείται είναι, με κοινό παρονομαστή μια γνωστή γλώσσα προγραμματισμού υψηλού επιπέδου, η μεταβολή της πλατφόρμας προορισμού (target platform) από γλώσσα μηχανής και επεξεργαστή με μνήμη, σε κυκλώματα. Το βάρος δίδεται περισσότερο στη γλώσσα προγραμματισμού και στην εύρεση τρόπων ώστε ο μεταγλωττιστής να αντιστοιχίσει τον κώδικα της σε κύκλωμα—δεν επιχειρείται δηλαδή η κατασκευή ακόμα μιας γλώσσας περιγραφής υλικού που θα εξυπηρετεί τις ανάγκες της σύνθεσης.

Η προσέγγιση της μεταγλώττισης του πηγαίου κώδικα σε κυκλώματα εισάγει βέβαια κάποια προβλήματα, τα οποία γίνονται αντιληπτά αν αναλογιστεί κανείς τη διαφορετική φιλοσοφία που διέπει ένα ειδικά κατασκευασμένο σταθερό κύκλωμα (fixed hardware) από ένα συμβατικό σύστημα με επεξεργαστή και μνήμη:

- Σε ένα σταθερό κύκλωμα (fixed hardware) κατά την εκτέλεση δεν υπάρχει στοίβα και δυναμική μνήμη.

- Οι μεταβλητές έχουν συγκεκριμένη θέση σε καταχωρητές και δεν έχει νόημα η δεικτοδότηση της μνήμης.

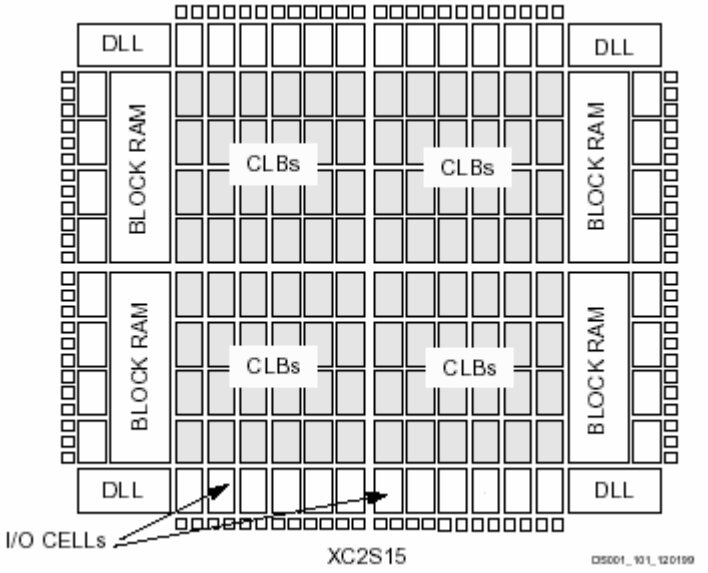
- Η κλήση αναδρομικών συναρτήσεων μπορεί να υλοποιηθεί αποδοτικά μόνο με επιπλέον hardware για κάθε στιγμιότυπο της αναδρομής, εφόσον κάθε συνάρτηση έχει ξεχωριστά κυκλώματα.

- Ο καθορισμός του τρόπου διασύνδεσης επικοινωνίας ενός κυκλώματος με το υπόλοιπο περιβάλλον απαιτεί ιδιαίτερο φορμαλισμό που δεν είναι πάντοτε σαφής σε μία γλώσσα προγραμματισμού.

Τα προβλήματα αυτά μπορούν να ξεπεραστούν κυρίως με τον επανορισμό της γλώσσας προγραμματισμού: ορισμένα χαρακτηριστικά αφαιρούνται τελείως από το συντακτικό της γλώσσας, κάποια άλλα παραμένουν αλλά δεν υποστηρίζονται με τον ίδιο τρόπο, ενώ μερικά νέα εισάγονται για την εκμετάλλευση καινούριων δυνατοτήτων. Γενικότερα, ο προγραμματισμός με στόχο την παραγωγή κυκλωμάτων και όχι κλασικού λογισμικού απαιτεί μια διαφορετική νοοτροπία από μέρους του προγραμματιστή, όσο κι αν η ύπαρξη μιας κοινής γλώσσας προγραμματισμού προσπαθεί να γεφυρώσει τα δύο μέρη.

Ένα άλλο θέμα που τίθεται είναι η πρακτικότητα του μεταγλωττιστή υλικού. Σε έναν επεξεργαστή, κάθε φορά που παράγεται νέος κώδικας μηχανής από τη μεταγλώττιση ενός προγράμματος μπορεί να εκτελεστεί άμεσα. Όταν όμως ένας μεταγλωττιστής

παράγει εκ νέου το σχηματικό ενός κυκλώματος, δεν είναι τόσο εύκολο να αναπαραχθεί ένα νέο κομμάτι υλικού. Σε πρώτη φάση μπορεί να πραγματοποιηθεί προσομοίωση (emulation) και να ελεγχθεί η συμπεριφορά του κυκλώματος, αλλά ακόμα και η διαδικασία της αναπαραγωγής του υλικού είναι δυνατό να διευκολυνθεί, αν χρησιμοποιηθούν ως τελικά hardware chips οι FPGAs. Τα αρχικά 'FPGA' σημαίνουν "Field-Programmable Gate Array". Πρόκειται για κυκλώματα που έχουν τη δυνατότητα να επαναπρογραμματίζονται ηλεκτρονικά σχετικά με τη λειτουργία τους. Στην τυπική περίπτωση, μια FPGA είναι ένα chip που αποτελείται από μερικές χιλιάδες κελιά, συνήθως τοποθετημένα σε διάταξη σχάρας, συνδεδεμένα μεταξύ τους, ενώ τα εξωτερικά κελιά έχουν τη δυνατότητα να επικοινωνούν για είσοδο / έξοδο μέσω των pins του chip.



Εικόνα 1 – Η αρχιτεκτονική μιας Xilinx Spartan II FPGA

Κάθε κελί περιέχει ένα ή περισσότερα LUTs (Lookup Tables) που μπορούν να προγραμματιστούν ώστε να παριστάνουν μια οποιαδήποτε λογική συνάρτηση Bool, καθώς και flip-flops. Έτσι, μια FPGA δεν είναι παρά μια τεράστια, περίεργα κατανεμημένη μνήμη. Κατά τον προγραμματισμό της, καθορίζονται τα περιεχόμενα των LUTs καθώς και οι διασυνδέσεις μεταξύ των κελιών, ώστε το chip να επιτελεί όσα ορίζει το σχηματικό διάγραμμα που εξάγει ο μεταγλωττιστής. Στην πράξη, ο μεταγλωττιστής παράγει το σχηματικό σε μια τελική γλώσσα περιγραφής υλικού, όπως η VHDL (*V*ery high speed integrated circuits *H*ardware *D*escription *L*anguage) ή η Verilog (*Veri*fication *Log*ic). Ο παραγόμενος κώδικας που περιγράφει το κύκλωμα περνά μετά από ένα πρόγραμμα σύνθεσης στο οποίο ο κατασκευαστής κάθε chip δίνει τα ποσοτικά μεγέθη και τις αρχιτεκτονικές προδιαγραφές για τον τρόπο απεικόνισης (mapping) στο χαμηλό επίπεδο του υλικού (low-level hardware), και παράγεται ένα bit stream το οποίο όταν σταλεί στα pins της FPGA τη διαμορφώνει κατάλληλα.

Στην εργασία αυτή λοιπόν σχεδιάζεται ένας μεταγλωττιστής υλικού (hardware compiler) για την παραγωγή κυκλωμάτων από κώδικα γραμμένο σε γλώσσα υψηλού επιπέδου, και συγκεκριμένα στη γλώσσα C. Ένα μεγάλο κομμάτι της εργασίας είναι αφιερωμένο στη μελέτη της διαδικασίας της σύνθεσης και του τρόπου κυκλωματικής αναπαράστασης των στοιχείων της γλώσσας. Η γλώσσα εξετάζεται από όλες τις πτυχές, και τα υποστηριζόμενα χαρακτηριστικά της (που προσαρμόζονται στις ανάγκες

της κυκλωματικής σύνθεσης και γενικά μπορεί να διαφέρουν από την κανονική C) αποτυπώνονται σε κύκλωμα με τρόπο που αναλύεται και περιγράφεται διεξοδικά, ακόμα και με τη βοήθεια εικόνων. Επίσης, παρουσιάζεται η υλοποίηση του μεταγλωττιστή, με γλώσσα εξόδου τη Verilog. Για τη συντακτική ανάλυση χρησιμοποιείται ο RDP, ένας γεννήτορας αναλυτών για LL(1) γλώσσες από περιγραφές BNF. Στις επόμενες ενότητες γίνεται μια μικρή εισαγωγή στο λογισμικό RDP και τη γλώσσα Verilog, ενώ ακολουθεί η κυρίως ανάλυση του μεταγλωττιστή. Τέλος, δίνονται περισσότερες λεπτομέρειες για την υλοποίηση, και εξετάζονται παραδείγματα εκτέλεσης του μεταγλωττιστή.

2

Το λογισμικό RDP

ΔΙΟΓΕΝΟΥΣ. Όταν κάποτε τον ρώτησε ο Ξενιάδης πῶς ἤθελε νὰ τὸν θάψουν, «Μὲ τὸ πρόσωπο πρὸς τὰ κάτω», εἶπε. «Γιατί;», τὸν ρώτησε. «Γιατί σὲ λίγο», εἶπε, « πρόκειται νὰ γυρίσουν τὰ κάτω ἀπάνω».

Γενικά.

Ο RDP (Recursive Descend Parser – Συντακτικός Αναλυτής Αναδρομικής Κατάβασης) είναι ένας μεταγλωττιστής, προϊόν έρευνας του Πανεπιστημίου του Λονδίνου, ο οποίος λαμβάνει ως είσοδο την περιγραφή μιας LL(1) γραμματικής σε EBNF μορφή, και κατασκευάζει έναν αντίστοιχο συντακτικό αναλυτή. Ο αναλυτής αυτός που παράγεται από τον RDP μπορεί να λάβει ως είσοδο αρχεία που υπακούουν στην LL(1) γραμματική και να δημιουργήσει στη μνήμη το αφηρημένο συντακτικό δέντρο (Abstract Syntax Tree) που αντιστοιχεί στη συντακτική ανάλυση του αρχείου εισόδου. Το δέντρο αυτό χρησιμοποιείται ύστερα με ευθύνη του προγραμματιστή για περαιτέρω επεξεργασία και εξαγωγή κάποιας άλλης τελικής μορφής, δηλαδή για τη διαδικασία της μεταγλώττισης. Έτσι, ο παραγόμενος αναλυτής δρα στην πράξη ως το πρόσθιο μέρος (front-end) ενός μεταγλωττιστή για την LL(1) γλώσσα. Υπό αυτή την έννοια ο RDP μπορεί να θεωρηθεί ένας *meta-compiler* (μέτα-μεταγλωττιστής), δηλαδή ένας compiler' compiler (μεταγλωττιστής μεταγλωττιστών).

Οι LL(1) γραμματικές. Οι μορφές BNF.

Ως LL(1) ονομάζεται εκείνη η γραμματική που διαβάζεται από αριστερά προς τα δεξιά, και για την οποία απαιτείται η ανάγνωση ενός μόνο συμβόλου κάθε φορά για να καθοριστεί ο επόμενος κανόνας που θα χρησιμοποιηθεί. Οι γραμματικές αυτές είναι γνωστές στη βιβλιογραφία από τη θεωρία γλωσσών και αυτομάτων, και είναι αυτές για τις οποίες είναι ευκολότερη η κατασκευή ενός συντακτικού αναλυτή.

Μία LL(1) γραμματική μπορεί να δοθεί σε διάφορες μορφές. Μια αρκετά συνήθης μορφή είναι η BNF (Backus-Naur Form), αλλά και η εκτεταμένη έκδοσή της, γνωστή ως EBNF (Extended BNF). Ο RDP χρησιμοποιεί μια παραλλαγή της EBNF που ονομάζει IBNF (Iterated BNF). Τα σημαντικότερα χαρακτηριστικά αυτής της μορφής είναι τα εξής:

- Τα μη τερματικά (non-terminal) σύμβολα γράφονται απευθείας, με το αναγνωριστικό τους όνομα.

- Τα τερματικά (terminal) σύμβολα γράφονται μέσα σε 'μονά εισαγωγικά'. Τα ειδικά σύμβολα ID, INTEGER και REAL (που γράφονται χωρίς μονά εισαγωγικά) αντιπροσωπεύουν τερματικά αναγνωριστικά, ακεραίους αριθμούς και πραγματικούς αριθμούς αντίστοιχα.

- Οι κανόνες γράφονται με τη γνωστή BNF μορφή:

  ```
  non_terminal ::= expression .
  ```

- Δύο ή περισσότερες εναλλακτικές παραστάσεις (expressions) ενός κανόνα χωρίζονται με την κατακόρυφη μπάρα '|'. Οι παρενθέσεις '(' και ')' μπορούν να χρησιμοποιηθούν για την απόδοση συγκεκριμένης προτεραιότητας στον τελεστή '|', όπως και σε όλους τους άλλους τελεστές.

– Οι αγκύλες ‘[’ και ‘]’ χρησιμοποιούνται για τον προσδιορισμό προαιρετικών παραστάσεων, δηλαδή χαρακτηρίζουν τα προαιρετικά μέρη της γραμματικής.

– Τα άγκιστρα ‘{’ και ‘}’ χρησιμοποιούνται για τον προσδιορισμό παραστάσεων που μπορούν να συμμετέχουν καμία, μία, ή περισσότερες φορές στο συγκεκριμένο σημείο της γραμματικής.

– Οι γωνίες ‘<’ και ‘>’ χρησιμοποιούνται για τον προσδιορισμό παραστάσεων που συμμετέχουν μία ή περισσότερες φορές.

– Γενικότερα, για τον προσδιορισμό μιας παράστασης που μπορεί να συμμετέχει από low ως high φορές (όπου low και high ακέραιοι, η απουσία των οποίων δηλώνει 0 και άπειρο αντίστοιχα), και της οποίας οι εμφανίσεις πρέπει να χωρίζονται με ένα συγκεκριμένο τερματικό σύμβολο, χρησιμοποιείται η ακόλουθη σύνταξη :

```
( expression ) low @ high 'terminal'
```

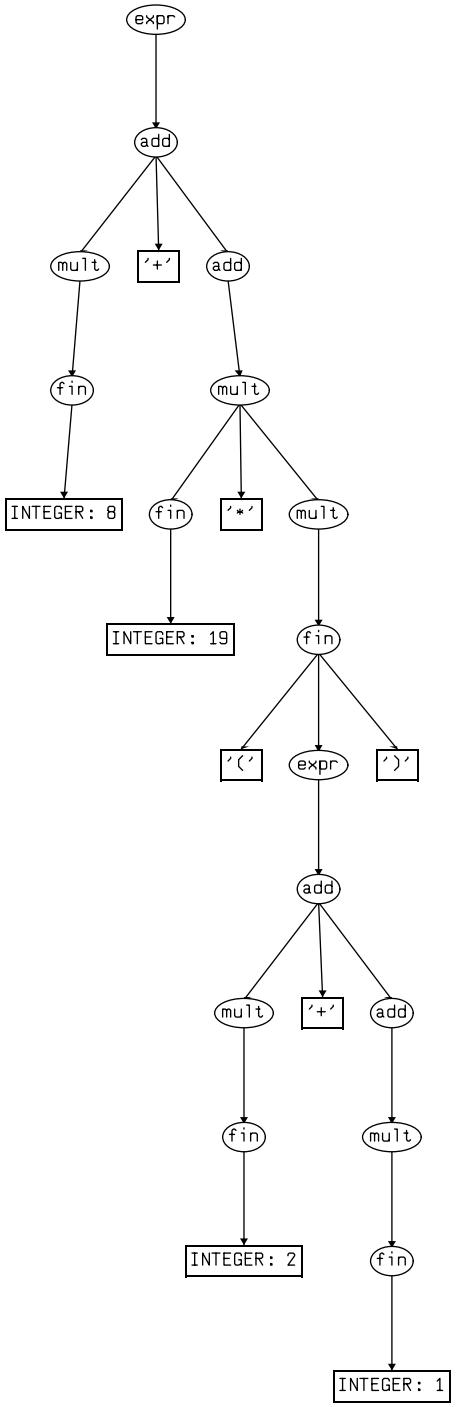
Ένα απλό παράδειγμα IBNF για την περιγραφή μιας γλώσσας που πραγματεύεται μαθηματικές παραστάσεις με άθροισμα και πολλαπλασιασμό φαίνεται παρακάτω:

```
expr ::= add .
add  ::= mult [ '+' add ] .
mult ::= fin [ '*' mult ] .
fin  ::= INTEGER | '(' expr ')' .
```

Ο RDP λαμβάνει ως είσοδο ένα αρχείο IBNF και παράγει έναν μεταγλωττιστή που λειτουργεί με αναδρομική κατάβαση. Η μέθοδος αυτή προσφέρει γρήγορη (γραμμική από άποψη πολυπλοκότητας) συντακτική ανάλυση. Στην περίπτωση του παραδείγματος, ο παραγόμενος μεταγλωττιστής είναι σε θέση να αναγνωρίσει παραστάσεις όπως π.χ. η ‘8 + 19 * (2 + 1)’ και να παράγει στη μνήμη μια δομή δέντρου που αντιστοιχεί στη συντακτική ανάλυση. Ύστερα, ο προγραμματιστής μπορεί να εκτελέσει μια δική του ρουτίνα για την επεξεργασία του δέντρου και την περαιτέρω αποτίμησή του. Αν και ο RDP υποστηρίζει την εισαγωγή σημασιολογικού κώδικα στους κανόνες, είναι προτιμότερο να μη χρησιμοποιείται αυτή η δυνατότητα, παρά να παράγεται συνολικά το δέντρο και μετά να ξεκινά από τον προγραμματιστή η σημασιολογική ανάλυση, με ρουτίνες που το διασχίζουν. Με άλλα λόγια, το δέντρο αποτελεί μια ενδιάμεση μορφή του κώδικα που ο παραγόμενος μεταγλωττιστής λαμβάνει στην είσοδο του κατά την εκτέλεση, και στη συνέχεια από την ενδιάμεση αυτή μορφή προκύπτει το τελικό αποτέλεσμα.

Το δέντρο της συντακτικής ανάλυσης.

Η μορφή του δέντρου είναι ένα σημαντικός παράγοντας για τη σημασιολογική επεξεργασία του. Οι κανόνες μιας γλώσσας πρέπει να δίδονται με τέτοιο τρόπο ώστε να αντικατοπτρίζεται και η σημασιολογία. Πράγματι, στο προηγούμενο παράδειγμα ο τρόπος γραφής των κανόνων συνεπάγεται ότι ο πολλαπλασιασμός έχει μεγαλύτερη προτεραιότητα από την πρόσθεση. Το αντίστοιχο δέντρο που παράγεται από τη συντακτική ανάλυση για την παράσταση ‘8 + 19 * (2 + 1)’ (σύμφωνα με τη γραμματική που δόθηκε ως παράδειγμα) φαίνεται στην εικόνα 2.



Εικόνα 2 – Το δέντρο της παράστασης '8+19*(2+1)' για την αρχική γραμματική

Όπως είναι φανερό, κάθε μετάβαση από το ένα επίπεδο στο επόμενο συμβολίζει και την εφαρμογή κάποιου κανόνα. Στα φύλλα του δέντρου εμφανίζονται τα τερματικά σύμβολα, καθώς αυτά δεν αναλύονται περαιτέρω. Το δέντρο της εικόνας 2, αν και αποτυπώνει με σωστό τρόπο την προτεραιότητα των τελεστών, περιέχει πολλούς κόμβους που δεν είναι απαραίτητοι. Θα ήταν καλύτερο να παραμένουν στο δέντρο μόνο τα στοιχεία εκείνα που απαιτούνται πραγματικά για μια μεταγλώττιση, δηλαδή να εξαλειφθούν τυχόν άχρηστοι ενδιάμεσοι κόμβοι καθώς και όσα από τα τερματικά σύμβολα αποτελούν τη λεγόμενη *συντακτική ζάχαρη* (syntactic sugar). Για το σκοπό αυτό ο RDP επιτρέπει τον χειρισμό του δέντρου (tree manipulation) μέσω ειδικών τε-

λεστών προαγωγής (promotion operators). Οι τελεστές αυτοί εισάγονται στην περιγραφή IBNF δεξιά από τους κόμβους των οποίων η τοποθέτηση στο δέντρο μεταβάλλεται. Οι τελεστές είναι οι εξής:

- Τελεστής '^': Ο κόμβος δεν εισάγεται στο δέντρο, και τα παιδιά που ενδεχομένως έχει ενώνονται απευθείας με τον γονικό κόμβο.

- Τελεστής '^^': Ο κόμβος αυτός αντικαθιστά τον γονικό κόμβο, και γίνεται ο νέος γονέας.

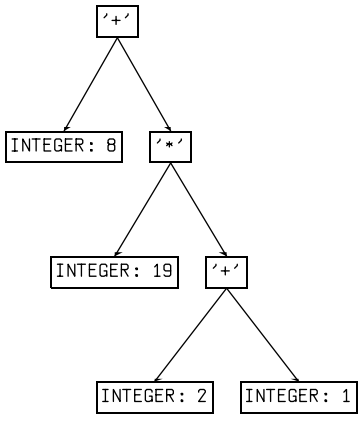
- Τελεστής '^^^': Ο κόμβος εισάγεται στο δέντρο πάνω από τον γονικό κόμβο, δηλαδή γίνεται ο νέος γονέας ενώ ο παλιός γονέας διατηρείται ως παιδί του.

Έτσι, η γραμματική μπορεί να ξαναγραφεί χρησιμοποιώντας τους παραπάνω τελεστές για την παραγωγή ενός πιο συμπαγούς δέντρου:

```
expr ::= add^^ .
add  ::= mult [ '+'^^ add ]:^^ .
mult ::= fin [ '*'^^ mult ]:^^ .
fin  ::= INTEGER^^ | '('^ expr^^ ')'^ .
```

Ας σημειωθεί ότι στο δεύτερο και τον τρίτο κανόνα, η σύνταξη ':^^' μετά από την προαιρετική παράσταση, σημαίνει ότι ο τελεστής '^^' θα εφαρμοστεί στους κόμβους mult και fin που βρίσκονται αριστερά από την προαιρετική παράσταση, σε περίπτωση που η παράσταση αυτή απουσιάζει.

Σύμφωνα με τους κανόνες αυτούς, η παράσταση '8 + 19 * (2 + 1)' αναλύεται όπως δείχνει η εικόνα 3.



Εικόνα 3 – Το δέντρο της παράστασης '8+19*(2+1)' για τη νέα γραμματική

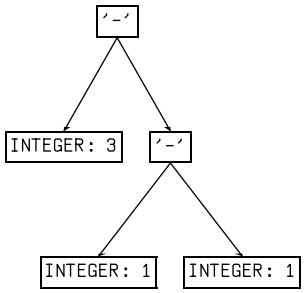
Το νέο δέντρο είναι κατά πολύ μικρότερο αφού έχουν πλέον αφαιρεθεί οι άχρηστοι κόμβοι, ενώ έχουν παραμείνει τα απαραίτητα στοιχεία για τη σημασιολογική επεξεργασία. Για παράδειγμα, έχουν αφαιρεθεί οι παρενθέσεις άλλα έχει διατηρηθεί η προτεραιότητα που αυτές εισάγουν στη μαθηματική παράσταση.

Είναι πρόδηλο ότι εξ' αιτίας του τρόπου δόμησης μιας LL(1) γλώσσας, στην οποία απαγορεύεται η αριστερή αναδρομή, είναι εύκολο να περιγραφούν δεξιά-προσεταιρι-

στικές δομές. Έστω για παράδειγμα το παρακάτω κομμάτι γραμματικής που περιγράφει τον τελεστή αφαίρεσης μιας γλώσσας:

```
sub ::= fin [ '-'^^ sub ]:^^ .
fin ::= INTEGER^^ .
```

Η γραμματική αυτή ορίζει με τέτοιο τρόπο τη γλώσσα έτσι ώστε το παραγόμενο δέντρο να υπονοεί δεξιά προσεταιριστικότητα για τον τελεστή. Για παράδειγμα, η παράσταση '3 – 1 – 1' θα αναλυθεί στο δέντρο της εικόνας 4.

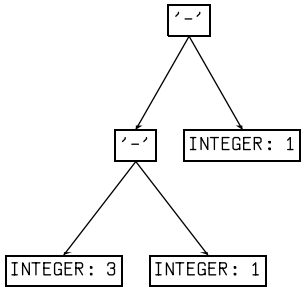


Εικόνα 4 – Το δέντρο της παράστασης '3 - 1 - 1' με δεξιό προσεταιρισμό

Είναι λοιπόν προφανές ότι ενώ η σύνταξη της γλώσσας είναι η σωστή, το παραγόμενο δέντρο δεν βοηθά στη σωστή σημασιολογική επεξεργασία. Θα πρέπει λοιπόν να αλλαχθεί η γραμματική ώστε να δοθούν οι κανόνες με τέτοιο τρόπο που ενώ θα διατηρείται η ίδια συνολικά σύνταξη, θα παράγεται ένα πιο κατάλληλο δέντρο. Η αριστερή προσεταιριστικότητα του τελεστή απαιτεί ένα είδος αριστερής αναδρομής που δεν είναι επιτρεπτό στις LL(1) γραμματικές. Ωστόσο, οι διάφοροι τελεστές προαγωγής του RDP μπορούν να βοηθήσουν στην επίλυση του προβλήματος. Έτσι, η γραμματική μπορεί να γραφεί με τους παρακάτω κανόνες:

```
sub ::= fin^^ { '-'^^^ fin } .
fin ::= INTEGER^^ .
```

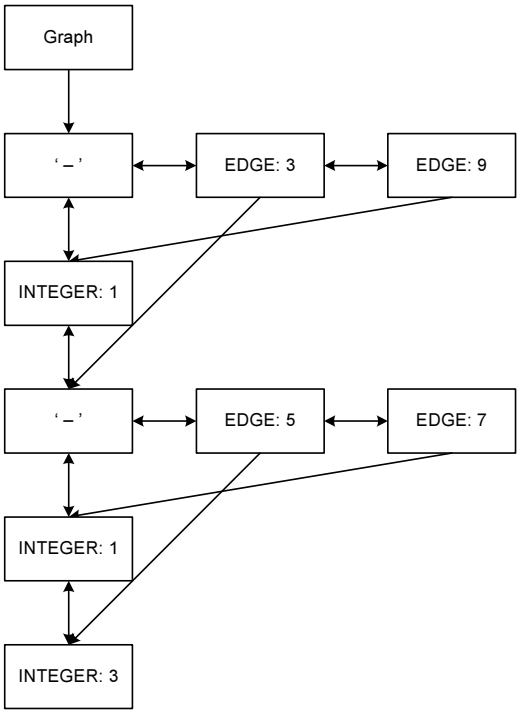
Το παραγόμενο δέντρο φαίνεται στην εικόνα 5. Ένας μεταγλωττιστής που θα επεξεργαζόταν αναδρομικά αυτό το δέντρο θα παρήγαγε το σωστό αποτέλεσμα χωρίς να ανησυχεί περαιτέρω για την προτεραιότητα και την προσεταιριστικότητα των τελεστών.



Εικόνα 5 – Το δέντρο της παράστασης '3 - 1 - 1' με αριστερό προσεταιρισμό

Ο RDP είναι γραμμένος σε γλώσσα C. Οι παραγόμενοι συντακτικοί αναλυτές κατασκευάζονται επίσης σε γλώσσα C. Το δέντρο αποθηκεύεται ως γράφος, σε μια ιδιαί-

τερη δομή η οποία κατασκευάζεται με τη βοήθεια μιας βιβλιοθήκης χειρισμού γράφων που περιλαμβάνεται στο λογισμικό πακέτο. Τη βιβλιοθήκη αυτή μπορεί κατόπιν να χρησιμοποιήσει ο προγραμματιστής για τη διάσχιση του δέντρου. Ο γράφος που παράγεται από τον RDP με τη βοήθεια της βιβλιοθήκης αποτελείται από μια διπλά συνδεδεμένη λίστα με τους κόμβους του γράφου. Για κάθε κόμβο υπάρχει μια διπλά συνδεδεμένη λίστα με τις ακμές που ξεκινούν από τον κόμβο αυτό. Η αναπαράσταση αυτή, για το παράδειγμα της εικόνας 5, φαίνεται στην εικόνα 6. Ο παραγόμενος γράφος είναι γενικά κατευθυνόμενος.



Εικόνα 6 – Η εσωτερική αναπαράσταση του γράφου της εικόνας 5

Στην παρούσα εργασία, ο RDP χρησιμοποιείται για τη συντακτική ανάλυση των αρχείων εισόδου, ώστε να παράγει το δέντρο που χρησιμοποιείται ως ενδιάμεση μορφή για την υπόλοιπη διαδικασία της μεταγλώττισης. Για το σκοπό αυτό εφοδιάζεται με το συντακτικό (σε IBNF) της γλώσσας του μεταγλωττιστή υλικού, όπως αυτή περιγράφεται σε επόμενη ενότητα. Περισσότερες λεπτομέρειες πάνω στην υλοποίηση του μεταγλωττιστή με τη χρήση του RDP δίνονται στο αντίστοιχο κεφάλαιο.

3

Η γλώσσα Verilog

ΑΓΙΔΟΣ ΤΟΥ ΑΡΧΙΔΑΜΟΥ. Ὅταν ἐστάλη, μόνος αὐτὸς, πρεσβευτὴς στὸ Φίλιππο καὶ κεῖνος τὸν ἠρώτησε πῶς ἡ Σπάρτη ἀπέστειλε ἕνα μόνο, ἀπήντησε: «Γιατὶ καὶ σύ, πρὸς τὸν ὁποῖο μ' ἔστειλε, ἕνας εἶσαι».

Εισαγωγή.

Η Verilog είναι μια γλώσσα περιγραφής υλικού (hardware description language), εφοδιασμένη και με αρκετά στοιχεία που διευκολύνουν τη λειτουργία της προσομοίωσης, εκτός από τη διαδικασία της σύνθεσης. Στην εργασία αυτή βέβαια χρησιμοποιείται ως τελική γλώσσα εξόδου, στην οποία περιγράφονται δομικά τα κυκλώματα που κατασκευάζονται από το μεταγλωττιστή.

Στην ενότητα αυτή δεν επιχειρείται μια αναλυτική παρουσίαση της γλώσσας από όλες τις πλευρές της. Αντίθετα, γίνεται μια απόπειρα για μια συνοπτική παρουσίαση μέρους των στοιχείων της γλώσσας, ώστε να μπορεί κανείς να παρακολουθήσει εύκολα το κεφάλαιο της υλοποίησης στο οποίο δίδεται ο κώδικας Verilog για τα διάφορα κυκλωματικά μέρη. Για μια εκτενέστερη παρουσίαση της γλώσσας μπορεί κανείς να ανατρέξει σε βιβλία που έχουν γραφεί για το σκοπό αυτό.

Η σύνταξη της γλώσσας δεν αναφέρεται με αυστηρό τρόπο. Σε γενικές γραμμές, κατά την περιγραφή μιας σύνταξης, τα σύμβολα που γράφονται με **έντονη γραφή** αναπαριστούν τερματικά σύμβολα που πρέπει να εμφανίζονται ως έχουν στη σύνταξη, ενώ τα υπόλοιπα (κανονικής γραφής) σύμβολα αναπαριστούν μη τερματικά στοιχεία που εξηγούνται περαιτέρω.

Γενικά. Τα αναγνωριστικά και οι σταθερές.

Η γλώσσα Verilog θυμίζει κάπως τη γλώσσα C: πολλοί τελεστές έχουν ακριβώς τον ίδιο συμβολισμό με αυτόν της C, τα σχόλια εισάγονται με τον ίδιο τρόπο όπως και στη C, και ως γλώσσα είναι γενικά case-sensitive (δηλαδή διαχωρίζει τα κεφαλαία από τα μικρά). Γενικά η γνώση της C βοηθά σε κάποιο βαθμό την εκμάθηση της Verilog.

Τα *αναγνωριστικά* (identifiers), δηλαδή τα ονόματα που εμφανίζονται στις δηλώσεις της γλώσσας, γράφονται με τους γνωστούς χαρακτήρες a-z, A-Z, 0-9, την κάτω παύλα ‘_’ καθώς και το δολάριο ‘$’, ενώ πρέπει να ξεκινούν με γράμμα ή την κάτω παύλα. Η γλώσσα επιτρέπει επίσης την εισαγωγή οποιουδήποτε εκτυπώσιμου ASCII χαρακτήρα σε ένα αναγνωριστικό, αν το αναγνωριστικό ξεκινά με τον ειδικό χαρακτήρα ‘\’ και τελειώνει με κάποιο χαρακτήρα κενού (όπως το enter, το tab ή το space).

Οι ακέραιες αριθμητικές σταθερές που υποστηρίζονται από τη Verilog γράφονται με τη εξής σύνταξη:

```
size base number
```

Το μέγεθος (size) είναι προαιρετικό, γράφεται ως ένας δεκαδικός ακέραιος και δηλώνει το μήκος (σε bits) της σταθεράς. Η βάση (base), που είναι κι αυτή προαιρετική, δηλώνει τη βάση του αριθμητικού συστήματος στο οποίο είναι γραμμένη η σταθερά. Οι δυνατές βάσεις είναι 'b ή 'B για το δυαδικό, 'o ή 'O για το οκταδικό, 'd ή 'D για το δεκαδικό και 'h ή 'H για το δεκαεξαδικό. Η απουσία βάσης ορίζει τη δεκαδική βάση και συνεπάγεται την απουσία μεγέθους, δηλαδή δε μπορεί να δοθεί το μέγεθος μιας σταθεράς χωρίς να δοθεί και η βάση. Τέλος, ο αριθμός (number) που αναπαριστά τη

σταθερά δεν είναι παρά μια ακολουθία ψηφίων της εκάστοτε βάσης, ενώ επιτρέπεται και η κάτω παύλα για λόγους αναγνωσιμότητας. Επίσης, επιτρέπονται επιπλέον και τα ψηφία X και Z που σημαίνουν ακαθόριστο και υψηλής αντίστασης (high-impedance) αντίστοιχα, αλλά αυτά δεν θα χρησιμοποιηθούν στην παρούσα εργασία, ελλείψει τρισταθών κυκλωμάτων. Ας σημειωθεί ακόμα ότι αν για μια σταθερά γραφούν περισσότερα ψηφία απ' ότι υποστηρίζει το μέγεθός της, τότε τα παραπανίσια bits (από το περισσότερο σημαντικό μέρος) αποκόπτονται. Αντίστοιχα, αν δοθούν λιγότερα bits τότε το περισσότερο σημαντικό μέρος της σταθεράς επεκτείνεται με μηδενικά. Μερικά παραδείγματα σταθερών δίνονται στη συνέχεια:

```
20
'hF1FA
32'b0
7'o733
```

Ο ορισμός των modules.

Μια κυκλωματική μονάδα περιγράφεται στη Verilog σε ένα module. Κατά τον ορισμό ενός module δίδεται το αναγνωριστικό (identifier) αυτού, καθώς και τα ονόματα των *θυρών* (ports) με τις οποίες το κύκλωμα του module θα επικοινωνεί με το υπόλοιπο περιβάλλον. Οι θύρες δεν είναι τίποτε άλλο από σήματα εισόδου ή εξόδου, τα οποία είναι και τα μόνα που φαίνονται από το εξωτερικό περιβάλλον του κυκλώματος—αποτελούν δηλαδή τη *διαπροσωπεία* (interface) αυτού. Ο ορισμός ενός module δεν προκαλεί την κατασκευή του κυκλώματος του· είναι μόνο μια περιγραφή. Η παραγωγή του κυκλώματος γίνεται με την παραγωγή στιγμιοτύπου (instantiation) του module αυτού σε κάποιο άλλο σημείο του κώδικα.

Η σύνταξη ενός module είναι η εξής:

```
module identifier (port_id1, port_id2, ..., port_idN);

   /* ... Κώδικας module ... */

endmodule
```

Για παράδειγμα, για τον ορισμό ενός απλού πολυπλέκτη ο οποίος διαλέγει κάποιο από τα δύο σήματα A και B με βάση ένα σήμα sel, και εξάγει το αποτέλεσμα στο σήμα Y, θα μπορούσε να γραφεί το εξής:

```
module mux (Y, A, B, sel);

   /* ... */

endmodule
```

Φυσικά, ακόμα δεν έχει οριστεί το είδος των σημάτων Y, A, B και sel, δηλαδή δεν έχει καθοριστεί ποιά από αυτά είναι σήματα εισόδου και ποιά εξόδου, ούτε έχει γνωστοποιηθεί ακόμη το μήκος τους σε bits. Για το σκοπό αυτό ο κώδικας ενός module ξεκινά συνήθως με τις δηλώσεις των θυρών (ports declaration). Εκεί, για κάθε θύρα (ή ομάδα θυρών με κοινά χαρακτηριστικά) δηλώνονται τόσο το είδος όσο και το μέγεθος, ως εξής:

```
direction [MSB : LSB] port_id, ..., port_id;
```

Το είδος (direction) μιας θύρας μπορεί να είναι input (για θύρες εισόδου), output (για θύρες εξόδου) ή inout (για θύρες εισόδου–εξόδου). Το μέγεθος μιας θύρας δηλώνεται προαιρετικά με τη σύνταξη [MSB:LSB], όπου MSB και LSB δύο ακέραιες παραστάσεις που καθορίζουν το περισσότερο σημαντικό bit (MSB) και το λιγότερο σημαντικό bit (LSB) της θύρας. Για παράδειγμα, η σύνταξη [7:0] δηλώνει θύρα μεγέθους 8 bits, με το MSB να έχει το μεγαλύτερο δείκτη και το LSB το μικρότερο (λογική 'little-endian'), ενώ η σύνταξη [1:3] δηλώνει θύρα μεγέθους 3 bits, με το MSB να έχει το δείκτη 1 και το LSB να έχει το δείκτη 3 (λογική 'big-endian'). Όταν το μέγεθος παραλείπεται, εννοείται θύρα του ενός bit. Μερικά ολοκληρωμένα παραδείγματα φαίνονται παρακάτω:

```
module mux (Y, A, B, sel);
output Y;
input A, B, sel;
  /* ... */
endmodule

module adder (sum, A, B);
output [7:0] sum;
input [7:0] A, B;
  /* ... */
endmodule
```

Στο δεύτερο από τα προαναφερθέντα παραδείγματα δίδεται ο ορισμός ενός αθροιστή δυο διανυσμάτων των 8 bits. Ωστόσο, θα ήταν επιθυμητό να μπορεί να περιγραφεί πιο γενικά ένας αθροιστής, για ορίσματα οποιουδήποτε μεγέθους. Για το σκοπό αυτό μπορεί να χρησιμοποιηθεί η σύνταξη μιας παραμέτρου (parameter). Οι παράμετροι χρησιμοποιούνται εντός ενός module ως σταθερές (constants), δηλαδή συμμετέχουν οπουδήποτε θα μπορούσε να μετέχει μια σταθερά. Η τιμή μιας παραμέτρου αναφέρεται κατά τη δήλωσή της, αλλά μια παράμετρος μπορεί γενικά να αρχικοποιηθεί σε διαφορετική τιμή, με μια ειδική σύνταξη, κατά την εισαγωγή στιγμιοτύπου (instantiation) του module. Έτσι, το αντίστοιχο παράδειγμα του ορισμού του αθροιστή, εμπλουτισμένο με μια παράμετρο wid που δηλώνει το μήκος του, γράφεται ως εξής:

```
module adder (sum, A, B);
parameter wid = 8;
output [wid-1:0] sum;
input [wid-1:0] A, B;
  /* ... */
endmodule
```

Όταν σε κάποιο άλλο σημείο του κώδικα απαιτηθεί η εισαγωγή του κυκλώματος ενός αθροιστή, θα πρέπει εκεί να παραχθεί ένα στιγμιότυπο (instance) του παραπάνω module. Στη σύνταξη που χρησιμοποιείται για την εισαγωγή ενός στιγμιοτύπου μπορεί να επανακαθοριστεί μια τιμή για την παράμετρο wid, που θα ισχύει για το συγκεκριμένο στιγμιότυπο. Η σύνταξη αυτή θα αναφερθεί αργότερα.

Ο ορισμός των δεδομένων.

Τα περιεχόμενα ενός module συνεχίζουν με τον ορισμό δεδομένων. Τα δεδομένα θυμίζουν τις μεταβλητές ενός κλασικού προγράμματος, αλλά στην περίπτωση της Verilog που είναι μια γλώσσα περιγραφής υλικού αυτά αφορούν κυρίως δύο κατηγορίες:

τους καταχωρητές (registers), που έχουν την ικανότητα να αποθηκεύουν τιμές, και τα απλά σύρματα διασύνδεσης (wires) τα οποία δεν έχουν αποθηκευτική ικανότητα.

Οι καταχωρητές δηλώνονται με την παρακάτω σύνταξη:

```
reg [MSB : LSB] reg_id, ..., reg_id;
```

Το μέγεθος [MSB:LSB], που συντάσσεται όπως και στις δηλώσεις των θυρών, είναι προαιρετικό και η απουσία του δηλώνει καταχωρητές του ενός bit. Ακολουθούν τα αναγνωριστικά ονόματα των καταχωρητών που υπακούουν τους γενικούς κανόνες περί αναγνωριστικών. Αν ένα αναγνωριστικό συμπίπτει με αυτό μιας θύρας εξόδου, τότε η αντίστοιχη θύρα εξόδου συνδέεται με την έξοδο του καταχωρητή, δηλαδή η συγκεκριμένη θύρα θεωρείται καταχωρητής.

Τα σύρματα διασύνδεσης (wires) αποτελούν οντότητες σημάτων που υπάρχουν στο φυσικό επίπεδο, και όπως αναφέρθηκε, δεν έχουν αποθηκευτική ικανότητα, παρά μεταδίδουν μόνιμα μια τιμή. Η τιμή αυτή μπορεί να προέρχεται είτε από την έξοδο κάποιας άλλης κυκλωματικής μονάδας, είτε γενικά από το αποτέλεσμα κάποιας παράστασης που δίδεται άπαξ στο σύρμα με μία συνεχή ανάθεση (continuous assignment).

Τα σύρματα δηλώνονται όπως και οι καταχωρητές, ως εξής:

```
wire [MSB : LSB] wire_id, ..., wire_id;
```

Όλες οι θύρες εισόδου θεωρούνται εξ' ορισμού σύρματα. Το ίδιο συμβαίνει και για τις θύρες εξόδου, με εξαίρεση αυτές που δηλώνονται ρητά ως καταχωρητές, όπως αναφέρθηκε προηγουμένως.

Υπάρχουν και άλλα είδη συρμάτων εκτός από τα απλά σύρματα διασύνδεσης 'wire'. Πρόκειται για τα τρισταθή σύρματα 'tri', τα σύρματα καλωδιωμένης λογικής OR 'wor' και καλωδιωμένης λογικής AND 'wand' (και τα τρισταθή τους ισοδύναμα 'trior' και 'triand'), και διάφορα άλλα που δεν απασχολούν αυτή την εργασία.

Τα πρωτογενή στιγμιότυπα.

Τα πρωτογενή στιγμιότυπα (primitive instances) χρησιμοποιούνται για την εισαγωγή πρωτογενών πυλών στο κύκλωμα, δηλαδή πυλών AND, OR, XOR, NAND, NOR, NXOR και NOT. Ο κώδικας για τη δημιουργία των αντίστοιχων στιγμιοτύπων συντάσσεται ως εξής:

```
and(out, in1, ..., inN);
or(out, in1, ..., inN);
xor(out, in1, ..., inN);
nand(out, in1, ..., inN);
nor(out, in1, ..., inN);
nxor(out, in1, ..., inN);
not(out1, ..., outN, in);
```

Για να εισαχθεί λοιπόν ένα στιγμιότυπο πρωτογενής πύλης, γράφεται το όνομα της και εντός παρενθέσεων τα σήματα εξόδου, ακολουθούμενα από τα σήματα εισόδου. Στην πραγματικότητα ένα σήμα μπορεί να είναι μια ολόκληρη παράσταση, αλλά στην πράξη χρησιμοποιούνται σύρματα (wires). Υπάρχει επίσης η δυνατότητα να καθορι-

στεί η καθυστέρηση (delay) της κάθε πύλης κατά την παραγωγή στιγμιοτύπου, κάτι που χρησιμεύει μόνο για τη λειτουργία της εξομοίωσης. Γενικότερα, στα στοιχεία της γλώσσας που αφορούν την εξομοίωση δεν θα γίνει ιδιαίτερη αναφορά.

Οι πρωτογενείς πύλες είναι καμιά φορά αρκετές για να περιγράψουν ένα απλό κύκλωμα. Έτσι, το παράδειγμα του πολυπλέκτη που είχε δοθεί νωρίτερα μπορεί πλέον να γραφεί ολοκληρωμένο με τη χρήση πυλών, ως εξής:

```
module mux (Y, A, B, sel);
output Y;
input A, B, sel;

wire A_and_sel, not_sel, B_and_not_sel;

  and(A_and_sel, A, sel);
  not(not_sel, sel);
  and(B_and_not_sel, B, not_sel);
  or(Y, A_and_sel, B_and_not_sel);

endmodule
```

Ο κώδικας αυτού του παραδείγματος ουσιαστικά υλοποιεί με πρωτογενείς πύλες τη λογική συνάρτηση Y = A·sel + B·sel´. Για το σκοπό αυτό δηλώνει τα επιπλέον σύρματα A_and_sel, not_sel και B_and_not_sel, τα οποία συμμετέχουν στις διασυνδέσεις μεταξύ των πυλών.

Τα στιγμιότυπα των modules.

Εκτός από τις πρωτογενείς πύλες, στο κύκλωμα ενός module μπορούν να εισαχθούν τα στιγμιότυπα άλλων modules. Η σύνταξη είναι η εξής:

```
module_id #(param_exp) instance_id (exp1, ..., expN);
```

Αρχικά γράφεται το αναγνωριστικό ‘module_id’ του module του οποίου το κύκλωμα θα εισαχθεί. Στη συνέχεια εισάγεται προαιρετικά μια τιμή, εντός των συμβόλων ‘#(‘ και ‘)’, για την παράμετρο του module, αν βέβαια το module φέρει κάποια παράμετρο. Αν στο module έχουν δηλωθεί παραπάνω από μία παράμετροι, στο σημείο αυτό μπορεί να γραφεί μια λίστα παραστάσεων χωρισμένες με κόμμα. Σε κάθε περίπτωση, η σύνταξη συνεχίζεται με ένα αναγνωριστικό ‘instance_id’ που χαρακτηρίζει το στιγμιότυπο, δηλαδή εισάγεται ένα όνομα για το συγκεκριμένο στιγμιότυπο. Τέλος, καθορίζονται τα σήματα που θα συνδεθούν με τις θύρες του στιγμιοτύπου.

Για παράδειγμα, ένας πολυπλέκτης 2×4 μπορεί να περιγραφεί με τη βοήθεια των πολυπλεκτών 1×2 που περιγράφηκαν νωρίτερα (στο module ‘mux’) ως εξής:

```
module mux4 (Y, A, B, C, D, sel1, sel2);
output Y;
input A, B, C, D, sel1, sel2;
wire Y1, Y2;
  mux m1 (Y1, A, B, sel2);
  mux m2 (Y2, C, D, sel2);
  mux m3 (Y, Y1, Y2, sel1);
endmodule
```

Επίσης, ένας αθροιστής ‘adder3’ τριών αριθμών μπορεί να περιγραφεί με τη βοήθεια του module ‘adder’ δυο αριθμών (που περιγράφηκε νωρίτερα ως παράδειγμα για τις παραμέτρους) ως εξής:

```
module adder3 (sum, A, B, C);
parameter N = 8;
output [N-1:0] sum;
input [N-1:0] A, B, C;
wire [N-1:0] temp;
  adder #(N) a1 (temp, A, B);
  adder #(N) a2 (sum, temp, C);
endmodule
```

Η σύνταξη ‘#(N)’ στο παράδειγμα αυτό περνά τη σταθερά N ως τιμή της παραμέτρου wid στον απλό αθροιστή.

Η συνεχής ανάθεση. Οι παραστάσεις.

Η συνεχής ανάθεση χρησιμοποιείται για την απόδοση τιμής σε ένα σύρμα. Στα παραδείγματα που αναφέρθηκαν μέχρι τώρα, τα σύρματα λάμβαναν τιμές έμμεσα από την έξοδο κάποιων στιγμιοτύπων. Ωστόσο, σε ένα σύρμα μπορεί να ανατεθεί τιμή και με μια εντολή ανάθεσης. Η τιμή αυτή είναι και η μοναδική τιμή που θα φέρει συνεχώς το σύρμα, γι’ αυτό και η διαδικασία ονομάζεται συνεχής ανάθεση. Η σύνταξη είναι η κάτωθι:

```
assign id = expression ;
```

Η παράσταση ‘expression’ παράγει μια τιμή που αποτελεί το σήμα του σύρματος που βρίσκεται στο αριστερό μέλος. Η συνεχής ανάθεση μπορεί να συνδυαστεί και με τη δήλωση ενός σύρματος, με την παρακάτω σύνταξη:

```
wire [MSB : LSB] id = expression ;
```

Οι παραστάσεις συμμετέχουν γενικότερα σε διάφορα σημεία του κώδικα της Verilog, όπως π.χ. στα ορίσματα των στιγμιοτύπων. Μπορούν να περιέχουν ονόματα θυρών, καταχωρητών και συρμάτων, σταθερές, τελεστές, κ.α. Οι τελεστές είναι ιδιαίτερα σημαντικοί καθώς επιτρέπουν την περιγραφή συνδυαστικών κυκλωμάτων πράξεων με γρήγορο και εύκολο τρόπο.

Οι μοναδιαίοι τελεστές.

Στη συνέχεια αναφέρονται συνοπτικά οι σημαντικότεροι μοναδιαίοι τελεστές.

```
- expression
```

Ο μοναδιαίος τελεστής ‘–’ χρησιμοποιείται για την παραγωγή του συμπληρώματος ως προς 2. Το αποτέλεσμα έχει το ίδιο μήκος με το όρισμα.

```
~ expression
```

Ο μοναδιαίος τελεστής ‘~’ χρησιμοποιείται για την αντιστροφή καθενός bit του ορίσματος. Και εδώ το αποτέλεσμα έχει το ίδιο μέγεθος με το όρισμα.

```
& expression
| expression
^ expression
```

Οι μοναδιαίοι τελεστές '&', '|' και '^' χρησιμοποιούνται για την εφαρμογή των λειτουργιών AND, OR και XOR αντίστοιχα σε όλα μαζί τα bits του ορίσματος. Το αποτέλεσμα που προκύπτει είναι μεγέθους ενός bit.

```
! expression
```

Ο μοναδιαίος τελεστής '!' χρησιμοποιείται για την άρνηση μιας λογικής συνθήκης, και γενικότερα παράγει 0 αν το όρισμα είναι μη μηδενικό και 1 αν το όρισμα είναι μηδενικό. Έτσι η δράση του θυμίζει ένα συνδυασμό των τελεστών '~' και '|'.

Οι δυαδικοί τελεστές.

Οι βασικότεροι δυαδικοί τελεστές παρουσιάζονται στη συνέχεια. Όποτε ένας τελεστής απαιτεί δύο ομοειδή ορίσματα του ίδιου μεγέθους, το όρισμα με το μικρότερο μέγεθος επεκτείνεται με μηδενικά. Γενικότερα, για να διατηρηθούν τα περισσότερο σημαντικά bits των ορισμάτων, σε μία παράσταση επεκτείνονται συνήθως όλα τα ορίσματα ώστε να φτάσουν το μέγεθος του μεγαλύτερου ορίσματος που εμφανίζεται στην παράσταση, σύμφωνα με κάποιους μάλλον περίπλοκους κανόνες της Verilog. Έτσι, για ορισμένους τελεστές υπάρχει εξάρτηση από τα συμφραζόμενα μιας παράστασης προκειμένου να καθοριστεί το τελικό μέγεθος των ορισμάτων τους.

```
expressionA + expressionB
expressionA - expressionB
expressionA * expressionB
expressionA / expressionB
expressionA % expressionB
```

Οι δυαδικοί τελεστές '+', '–', '*' και '/' χρησιμοποιούνται για τις βασικές πράξεις της πρόσθεσης, αφαίρεσης, πολλαπλασιασμού και διαίρεσης των ορισμάτων τους αντίστοιχα. Ο τελεστής '%' χρησιμοποιείται για τον υπολογισμό του υπολοίπου της διαίρεσης των δύο ορισμάτων, όπως δηλαδή και στη C. Οι περισσότερες υλοποιήσεις της Verilog θέτουν κάποιους περιορισμούς για τη διαίρεση—συνήθως επιτρέπουν μόνο δυνάμεις του 2 στο δεξιό όρισμα (του διαιρέτη) ή απαιτούν σταθερά ορίσματα.

```
expressionA & expressionB
expressionA | expressionB
expressionA ^ expressionB
```

Οι δυαδικοί τελεστές '&', '|' και '^' χρησιμοποιούνται για τις bitwise πράξεις AND, OR και XOR αντίστοιχα. Οι πράξεις αυτές μεταξύ των δύο ορισμάτων γίνονται bit-προς-bit, δηλαδή παράγεται αποτέλεσμα του ίδιου μεγέθους με τα ορίσματα (τα οποία όπως αναφέρθηκε επεκτείνονται με μηδενικά, όταν αυτό είναι αναγκαίο).

```
expressionA == expressionB
expressionA != expressionB
expressionA < expressionB
expressionA > expressionB
expressionA <= expressionB
```

```
expressionA >= expressionB
```

Οι τελεστές σύγκρισης παράγουν αποτέλεσμα του ενός bit, που έχει την τιμή 1 αν η σύγκριση είναι αληθής, και 0 σε αντίθετη περίπτωση. Τα ορίσματα των τελεστών σύγκρισης θεωρούνται απρόσημοι αριθμοί, δηλαδή δεν υποστηρίζεται άμεσα η σύγκριση αριθμών παράστασης συμπληρώματος ως προς 2.

```
expressionA && expressionB
expressionA || expressionB
```

Οι λογικοί τελεστές '&&' και '||' χρησιμοποιούνται για τη σύζευξη και τη διάζευξη λογικών συνθηκών. Το αποτέλεσμά τους έχει μέγεθος ενός bit. Ακριβέστερα, ο τελεστής '&&' παράγει τιμή 1 αν αμφότερα τα ορίσματά του είναι μη μηδενικά (αληθή), ενώ ο τελεστής '||' παράγει τιμή 1 αν τουλάχιστον ένα από τα ορίσματά του είναι μη μηδενικό (αληθές).

```
expressionA >> expressionB
expressionA << expressionB
```

Οι τελεστές ολίσθησης '<<' και '>>' πραγματοποιούν ολίσθηση του αριστερού ορίσματος κατά τόσες θέσεις όσες ορίζει η τιμή του δεξιού ορίσματος. Κατά την ολίσθηση, οι θέσεις που μένουν κενές γεμίζουν με μηδενικά. Η ολίσθηση επιτρέπεται τόσο με σταθερό όσο και με μεταβλητό αριθμό θέσεων—στην πρώτη περίπτωση γίνεται μια απλή ανακατάταξη των bits του αριστερού ορίσματος ενώ στη δεύτερη περίπτωση παράγεται ένας γενικός ολισθητής.

Άλλοι τελεστές.

Υπάρχουν ορισμένοι ακόμα ενδιαφέροντες τελεστές, που αναφέρονται στη συνέχεια.

```
{ expression1, expression2, ..., expressionN }
```

Ο παραπάνω τελεστής ονομάζεται τελεστής παράθεσης (concatenation) και χρησιμοποιείται για τη δημιουργία μιας τιμής που προκύπτει από την παράθεση των τιμών των παραστάσεων. Το κάθε όρισμα του τελεστή αποτιμάται, και τα προκύπτοντα bit vectors συνενώνονται ώστε να προκύψει ένα μεγάλο διάνυσμα. Μια παραλλαγή του τελεστή είναι η επαναληπτική παράθεση που συντάσσεται ως εξής:

```
{ constant { expression } }
```

Σε μια επαναληπτική παράθεση, η τιμή της παράστασης 'expression' παρατίθεται τόσες φορές όσες ορίζει η σταθερά 'constant'. Για παράδειγμα, η σύνταξη {3{'b10}} ισοδυναμεί με την παράσταση 'b101010.

```
condition ? expressionA : expressionB
```

Ο τελεστής '?:' αποτιμά τη συνθήκη 'condition', και αν είναι αληθής επιλέγει την πρώτη παράσταση 'expressionA', διαφορετικά επιλέγει την παράσταση 'expressionB'.

```
expression [ idx ]
expression [ constant1 : constant2 ]
```

Οι τελεστές επιλογής bit (bit-select) και μέρους (part-select) χρησιμοποιούνται για την επιλογή ενός bit, ή ενός μέρους συνεχόμενων bits, από μια παράσταση ‘expression’. Η παράσταση ‘idx’ του τελεστή επιλογής bit πρέπει να αποτιμάται εντός των ορίων του μεγέθους του αριστερού ορίσματος, ώστε να επιλέξει το αντίστοιχο bit. Ο τελεστής επιλογής μέρους συντάσσεται αποκλειστικά με σταθερές παραστάσεις, που αποτελούν τα δύο όρια του μέρους που επιλέγεται από την παράσταση.

Παραδείγματα.

Έχοντας εξετάσει τους κυριότερους τελεστές, το παράδειγμα του απλού πολυπλέκτη που είχε δοθεί νωρίτερα με πρωτογενείς πύλες μπορεί πλέον να ξαναγραφεί κάνοντας χρήση παραστάσεων.

```
module mux (Y, A, B, sel);
output Y;
input A, B, sel;
  assign Y = (A & sel) | (B & (~sel)) ;
endmodule
```

Αντίστοιχα, το κύκλωμα του αθροιστή μεταβλητού μήκους μπορεί να γραφεί ως εξής:

```
module adder (sum, A, B);
parameter wid = 8;
output [wid-1:0] sum;
input [wid-1:0] A, B;
  assign sum = A + B;
endmodule
```

Τέλος, το κύκλωμα ενός module που εκτελεί επέκταση προσήμου για έναν αριθμό μπορεί να γραφεί ως εξής:

```
module extend (qout, qin);
parameter out_size = 32;
parameter in_size = 8;
output [out_size-1:0] qout;
input [in_size-1:0] qin;
  assign qout = { {out_size-in_size{qin[in_size-1]}}, qin};
endmodule
```

Στο τελευταίο αυτό παράδειγμα, το πρόσημο του αριθμού qin, που είναι το bit που επιλέγεται ως qin[in_size – 1], παρατίθεται out_size – in_size φορές, όσες δηλαδή απαιτούνται για την επέκταση.

Η χρήση προστακτικών μπλοκ.

Η Verilog επιτρέπει τη χρήση προστακτικών μπλοκ, με τα οποία είθισται να γίνεται ο χειρισμός των ακολουθιακών μερών του κυκλώματος. Ένα προστακτικό μπλοκ αποτελείται από εντολές όπως αναθέσεις σε καταχωρητές, εντολές διακλάδωσης (if), εντολές επανάληψης (for, while, repeat), κ.α. Με αυτό τον τρόπο, τα προστακτικά μπλοκ επιτρέπουν τον διαδικαστικό (procedural) προγραμματισμό και μπορούν να χρησιμοποιηθούν για σύνθεση με βάση τη συμπεριφορά (behavioral synthesis). Στην εργασία αυτή, που τα παραγόμενα κυκλώματα περιγράφονται δομικά και όχι από ά-

ποψη συμπεριφοράς (αφού η συμπεριφορά έχει δοθεί σε C και ο σκοπός του μεταγλωττιστή υλικού είναι ακριβώς η μετατροπή σε κυκλώματα), η χρήση των προστατικών μπλοκ περιορίζεται στην ανάθεση των καταχωρητών. Οι καταχωρητές, ως ακολουθιακά κυκλώματα, απαιτούν τον καθορισμό του χρονισμού τους, διότι σε αντίθεση με τα σύρματα που λαμβάνουν συνεχώς μια τιμή, αυτοί έχουν την ικανότητα να αλλάζουν την καταχώρησή τους με κάποιο συμβάν, όπως η έλευση του παλμού ρολογιού.

Ένα προστακτικό μπλοκ ξεκινά με τη λέξη initial ή always, ανάλογα με το αν οι εντολές που περιέχει θα εκτελεστούν άπαξ, μια συνολικά φορά, ή θα εκτελούνται ξανά και ξανά. Στη συνέχεια δηλώνεται μια λίστα από σήματα, η αλλαγή των οποίων θα πυροδοτήσει την έναρξη της εκτέλεσης των εντολών του μπλοκ. Για κάθε σήμα μπορεί να δοθεί η λέξη posedge, ή η λέξη negedge, για να περιορίσει την πυροδότηση στη θετική ή την αρνητική ακμή μιας αλλαγής αντίστοιχα. Τα σήματα χωρίζονται μεταξύ τους με τη λέξη or, ενώ ολόκληρη η λίστα περιβάλλεται μεταξύ '@(' και ')'. Στη συνέχεια, ακολουθεί ένα μπλοκ εντολών ανάμεσα στις λέξεις begin και end, ή τις λέξεις fork και join, ανάλογα με το αν η σημασιολογία της εκτέλεσης είναι σειριακή ή παράλληλη αντίστοιχα, δηλαδή ανάλογα με το αν οι εντολές θα εκτελεστούν με τη σειρά, η μία μετά την άλλη, ή όλες παράλληλα. Αν το μπλοκ περιέχει μία μόνο εντολή, τότε μπορούν να παραληφθούν εντελώς οι λέξεις begin και end αφού ούτως ή άλλως δεν έχει νόημα η σειριακή ή παράλληλη εκτέλεση για μία μόνο εντολή.

Αν και υπάρχουν εντολές διαφόρων ειδών, η σημαντικότερη που αφορά την παρούσα εργασία είναι, όπως αναφέρθηκε, η ανάθεση τιμής σε καταχωρητή. Η σύνταξή της είναι η εξής:

```
id = expression ;
```

Η ανάθεση αυτή εμποδίζει το χρόνο να προχωρήσει μέχρι να πραγματοποιηθεί η αποτίμηση των ορισμάτων και η εκχώρηση της τιμής στον καταχωρητή. Με άλλα λόγια, όταν υπάρχει μια τέτοια ανάθεση μέσα σε ένα σειριακό μπλοκ begin-end, η επόμενη εντολή δεν εκτελείται παρά μόνο όταν ολοκληρωθεί η ανάθεση. Αντίθετα, η παρακάτω σύνταξη αποτελεί μια μη παρεμποδίζουσα (non-blocking) εκδοχή της ανάθεσης:

```
id <= expression ;
```

Όταν εμφανιστεί μια τέτοια εντολή ανάθεσης σε ένα σειριακό μπλοκ, εκτελείται η επόμενη εντολή, και η καταχώρηση της τιμής αναβάλλεται μέχρι το τέλος του συγκεκριμένου χρονικού κύκλου. Η εντολή αυτή είναι χρήσιμη όταν η τιμή του καταχωρητή δεν θα χρησιμοποιηθεί από τις εντολές που ακολουθούν, οπότε μπορεί να δοθεί με αυτό τον τρόπο για να μη παρεμποδίσει τις επόμενες εντολές από το να εκτελεστούν αμέσως. Φυσικά, όταν είναι η μόνη εντολή που αποτελεί ένα μπλοκ τότε έχει ακριβώς την ίδια συμπεριφορά με την κανονική ανάθεση.

Εφαρμόζοντας όλα τα παραπάνω, μπορεί να γραφεί ένα απλό παράδειγμα για την περιγραφή του module ενός DFF:

```
module dff (qout, qin, clk);
output qout;
input qin, clk;
reg qout;
```

```
  always @(posedge clk)
    qout <= qin;

endmodule
```

Στο παράδειγμα αυτό, η θετική ακμή του σήματος clk (που αντιπροσωπεύει το ρολόι) πυροδοτεί την εκχώρηση της τιμής qin στον καταχωρητή qout, του οποίου το περιεχόμενο εξάγεται στην αντίστοιχη θύρα qout, εφόσον φέρει το ίδιο όνομα.

Ένα τελευταίο παράδειγμα ενός ασύγχρονου καταχωρητή των N bits ο οποίος φέρει ακμοπυροδότητες εισόδους write και reset, για την ενεργοποίηση της εγγραφής και το μηδενισμό αντίστοιχα, δίνεται παρακάτω.

```
module asyncreg (qout, qin, write, reset);
parameter N = 8;
output [N-1:0] qout;
input [N-1:0] qin;
input write, reset;
reg [N-1:0] r;

  assign qout = r;
  always @(posedge write or posedge reset)
    r <= reset ? 0 : qin;
endmodule
```

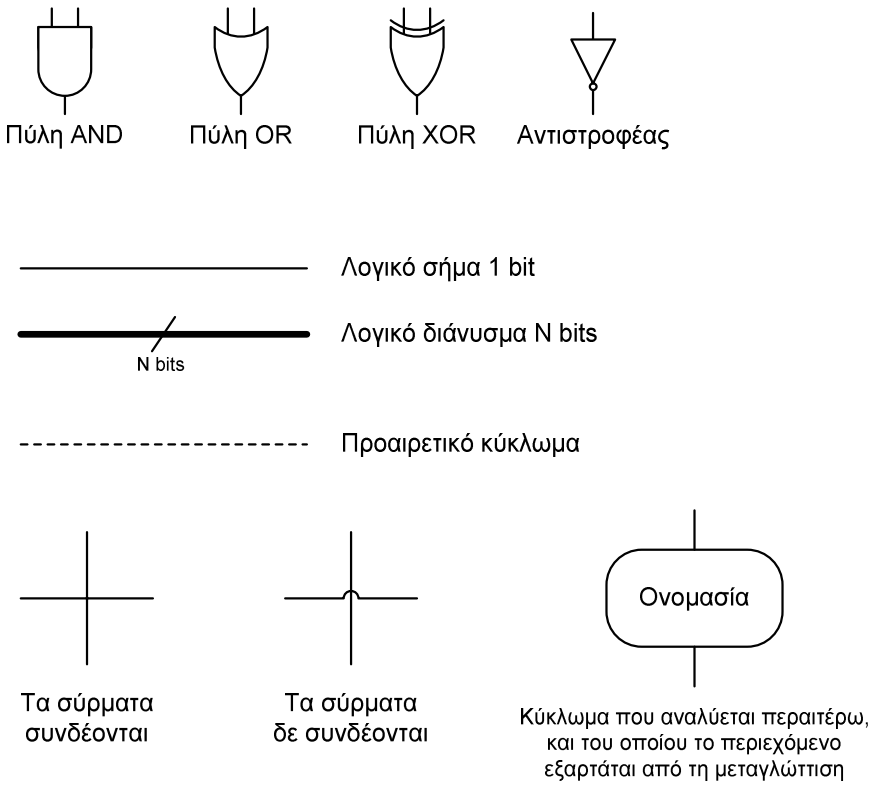
4

Η κυκλωματική σύνθεση

ΦΙΛΙΠΠΟΥ. Ὁ Φίλιππος ἔλεγε, ὅτι εἶναι ἰσχυρότερος
ἕνας στρατὸς ἀπὸ ἐλάφια μὲ λιοντάρι στρατηγὸ,
παρὰ στρατὸς λιονταριῶν μὲ στρατηγὸ ἐλάφι.

Κυκλωματικές συμβάσεις

Τα κυκλώματα που απεικονίζονται σε αυτή την εργασία ακολουθούν κλασικές συμβάσεις, οι κυριότερες από τις οποίες φαίνονται παρακάτω:



Εικόνα 7 – Οι κυριότερες κυκλωματικές συμβάσεις

Η γλώσσα C ως γλώσσα για τη σύνθεση κυκλωμάτων. Οι αλλαγές από το πρότυπο. Οι δυνατότητες της νέας γλώσσας.

Ως αρχική γλώσσα επελέγη η γλώσσα C. Η επιλογή αυτή έγινε με βάση διάφορα κριτήρια:

- Η γλώσσα C είναι πολύ δημοφιλής και κατέχεται από την πλειοψηφία αυτών που ασχολούνται με προγραμματισμό.

- Υπάρχουν ήδη γραμμένες στη C ρουτίνες για τους πιο γνωστούς αλγόριθμους.

Ωστόσο, η γλώσσα C δεν έχει σχεδιαστεί για τη σύνθεση κυκλωμάτων, και έτσι παρουσιάζει κάποιες εγγενείς αδυναμίες στην ικανότητα περιγραφής κυκλωματικών δομών. Επιπλέον, καθώς η C χρησιμοποιείται για την παραγωγή λογισμικού και όχι υλικού, περιλαμβάνει ορισμένα χαρακτηριστικά (όπως οι δείκτες) των οποίων η μετάφραση στο επίπεδο του υλικού είναι μια προβληματική διαδικασία.

Με δεδομένο ότι το σκεπτικό του όλου εγχειρήματος είναι η προσαρμογή μιας υπάρχουσας γλώσσας προγραμματισμού για τη σύνθεση κυκλωμάτων, με όσο το δυνατό πιο διαφανή τρόπο για τον τελικό χρήστη, και όχι η δημιουργία μιας νέας *γλώσσας περιγραφής υλικού* (hardware description language - HDL), έγιναν τα εξής ήδη αλλαγών στη γλώσσα C:

- Ορισμένα χαρακτηριστικά της γλώσσας C τα οποία δεν υποστηρίζονται αφαιρέθηκαν τελείως. Αυτά περιλαμβάνουν τις σύνθετες δομές (struct) και τους ορισμούς τύπων με τη χρήση typedef, τον τελεστή sizeof, τους ορισμούς πινάκων, κ.α.

- Μερικά χαρακτηριστικά της γλώσσας για λόγους συμβατότητας παρέμειναν στο συντακτικό, αλλά δεν υποστηρίζονται σημασιολογικά· ουσιαστικά αγνοούνται από το μεταγλωττιστή. Τέτοια είναι οι δηλώσεις volatile, const, κ.α.

- Έγιναν λιγοστές απαραίτητες επεκτάσεις στη σύνταξη της γλώσσας (ώστε να μη διαφέρει υπερβολικά από το πρότυπο), προκειμένου να προστεθούν νέες δυνατότητες.

- Η σημασιολογία των περισσότερων στοιχείων της γλώσσας προσαρμόστηκε στις απαιτήσεις της σύνθεσης κυκλωμάτων. Η νέα σημασιολογία είναι πιο περιοριστική, μιας και πλέον επιτρέπονται ακόμη λιγότερα από αυτά που επιτρέπονται στην αρχική γλώσσα. Αυτό συμβαίνει μερικές φορές επειδή κάτι δεν συντίθεται εύκολα κυκλωματικά, ενώ άλλες φορές αποτελεί σκόπιμη σχεδιαστική επιλογή.

Αν και γενικά η μεθοδολογία που ακολουθήθηκε προσπαθεί να διατηρήσει μια συμβατότητα, υπό την έννοια ότι κάθε τι που είναι δυνατό στην κλασική C αναζητά ένα τρόπο να αντιστοιχηθεί στο κυκλωματικό υλικό, τελικά οι μεγάλες διαφορές μεταξύ υλικού και λογισμικού απαιτούν έναν επανορισμό των πραγμάτων και οδηγούν σε

μια διαφορετική φιλοσοφία από αυτή που έχει στο νου του ο κλασικός προγραμματιστής. Έτσι είναι μάλλον ανέφικτο να αξιοποιήσει κανείς τα στοιχεία της γλώσσας χωρίς να έχει υπ' όψιν του τις υποκείμενες τεχνικές που χρησιμοποιούνται για την αντιστοίχηση του κώδικα σε κυκλώματα.

Παρακάτω αναφέρονται πολύ συνοπτικά οι κυριότερες αλλαγές από το πρότυπο της γλώσσας:

- Ως προς το **σύστημα τύπων**, έχει εισαχθεί ο θεμελιώδης τύπος bit (και bit<N> για δήλωση διανύσματος των N bits), ενώ υποστηρίζονται οι γνωστοί τύποι char και int (καθώς και οι παραλλαγές short, long, signed, unsigned). Οποιοιδήποτε άλλοι τύποι, συμπεριλαμβανομένων των τύπων float και double, των *πινάκων* (arrays), και των *σύνθετων δομών* (struct), δεν υποστηρίζονται. Επίσης, δεν υποστηρίζονται οι ορισμοί νέων τύπων με typedef, ενώ οι τύποι *δεικτών* (pointers) υποστηρίζονται μερικώς, μόνο για ένα επίπεδο δεικτοδότησης, και μόνο κατά τον ορισμό καθολικών μεταβλητών ή παραμέτρων *κατ' αναφορά* (by reference). Τέλος, ο τύπος void επιτρέπεται μόνο ως τύπος επιστροφής συνάρτησης.

- Ως προς τις **εντολές**, δεν υποστηρίζεται η εντολή goto, ενώ έχει εισαχθεί η λέξη-κλειδί par για την παράλληλη εκτέλεση εντολών ενός μπλοκ.

- Ως προς τους **τελεστές**, δεν υποστηρίζονται οι τελεστές "." (επιλογής μέλους δομής), "->" (επιλογής μέλους δομής από δείκτη), "," (κόμμα), "*" (πολλαπλασιασμού), "/" (διαίρεσης), "%" (υπολοίπου) και "sizeof" (μεγέθους).

- Ως προς τις **δηλώσεις**, δεν υποστηρίζονται συναρτήσεις μεταβλητού πλήθους παραμέτρων, ενώ έχει εισαχθεί η δυνατότητα δήλωσης πολλαπλών αντιτύπων μιας συνάρτησης, για την υποστήριξη αναδρομής συγκεκριμένου βάθους. Τέλος, οι δηλώσεις τοπικών μεταβλητών επιτρέπονται μόνο στην αρχή μιας συνάρτησης και όχι στην αρχή οποιουδήποτε μπλοκ. Οι προσδιοριστές auto, extern, const, register και volatile αγνοούνται.

- Ως προς τις **σταθερές**, δεν υποστηρίζονται σταθερές πραγματικών αριθμών, ενώ οι σταθερές *αλφαριθμητικών* (strings) χρησιμοποιούνται ως τρόπος γραφής δεκαεξαδικών αριθμητικών σταθερών.

Αν και από την αρχική γλώσσα έχουν αφαιρεθεί αρκετά στοιχεία, ωστόσο η προκύπτουσα γλώσσα παραμένει ισχυρή, με πλήθος χρήσιμων χαρακτηριστικών. Οι κυριότερες δυνατότητες που έχουν μεγάλη σημασία ακριβώς λόγω του ότι η γλώσσα χρησιμοποιείται για την παραγωγή κυκλωματικού υλικού, φαίνονται παρακάτω:

- **Υποστήριξη παράλληλης σημασιολογίας (parallel semantics)**: Ο προγραμματιστής μπορεί να καθορίσει οποιουσδήποτε συνδυασμούς απλών ή σύνθετων εντολών σειριακής ή παράλληλης εκτέλεσης. Αυτή η δυνατότητα, σε συνδυασμό με τη χρήση των *καθολικών μεταβλητών* (global variables), οι οποίες είναι *μοιραζόμενες* (shared) από όλα τα σημεία του κώδικα, επιτρέπει την περιγραφή οποιουδήποτε προγραμματιστικού μοντέλου: *πολυεπεξεργασία* (multitasking) με δυνατότητα συγχρονισμού, *σωλήνωση* (pipelining), τεχνικές παραλληλίας τύπου "fork / join", κ.α.

- **Παραγωγή σταθερού υλικού (fixed hardware)**: Το κύκλωμα που παράγεται από τον πηγαίο κώδικα δεν απαιτεί τη χρήση στοίβας ή άλλης δυναμικής μνήμης για την εκτέλεση του. Μια συνάρτηση μπορεί να καλείται από πολλά διαφορετικά σημεία του κώδικα, και να καλεί με τη σειρά της άλλες συναρτήσεις—κάθε κλήση θα επιστρέψει στο σωστό σημείο (αρκεί βέβαια να μην έχει δημιουργηθεί κύκλος) χάρη στα ειδικά ακολουθιακά κυκλώματα συγχρονιστών.

- **Υποστήριξη αναδρομής (recursion)**: Η αναδρομική κλήση συναρτήσεων είναι ως γνωστόν μια δυνατότητα που δεν υλοποιείται εύκολα όταν το κύκλωμα είναι σταθερό και δεν υπάρχει στοίβα κλήσεων. Πάραυτα, υποστηρίζεται αναδρομή συγκεκριμένου βάθους, το οποίο δίδεται από τον προγραμματιστή με ειδικό φορμαλισμό που έχει εισαχθεί για το σκοπό αυτό στη γλώσσα. Η αναδρομή πετυχαίνεται με την παραγωγή πολλαπλών κυκλωματικών αντιτύπων μιας συνάρτησης, ώστε κάθε συνάρτηση που καλεί τον εαυτό της να ενεργοποιεί το επόμενο κατά σειρά αντίτυπό της.

- **Κλήση κατ' αναφορά (call by reference)**: Η δυνατότητα της κλήσης συναρτήσεων κατ' αναφορά, μέσω δεικτοδότησης των παραμέτρων μιας συνάρτησης, έχει παραμείνει και υποστηρίζεται αυτούσια στο υλικό επίπεδο. Σε μία κλήση συνάρτησης με παράμετρο κατ' αναφορά, η συνάρτηση συνδέεται κυκλωματικά με τη μεταβλητή στην οποία αναφέρεται. Έτσι, αν διαφορετικές συναρτήσεις που εκτελούνται παράλληλα αναφέρονται στην ίδια μεταβλητή, παρακολουθούν ζωντανά τις αλλαγές που καθεμία πραγματοποιεί πάνω στη μεταβλητή αυτή. Με αυτό τον τρόπο διατηρείται η πραγματική σημασιολογία των δεικτών της C και δεν αποτελεί μόνον έναν απλό φορμαλισμό για να δηλωθεί μια παράμετρος ως παράμετρος εισόδου-εξόδου.

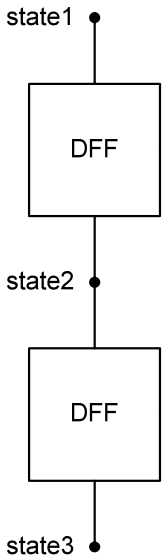
Το είδος των κυκλωμάτων που συντίθενται. Η μορφή του μονοπατιού ελέγχου. Οι καταστάσεις ελέγχου και η σχέση τους με τις εντολές.

Στην εργασία αυτή, ως τελικό αποτέλεσμα συντίθενται ψηφιακά σύγχρονα κυκλώματα των οποίων η περιγραφή δίδεται σε υψηλό επίπεδο RTL. Οι λεπτομέρειες της εσωτερικής κατασκευής τυπικών μονάδων όπως οι αθροιστές, οι συγκριτές ή οι καταχωρητές δεν αποτελούν αντικείμενο μελέτης. Ωστόσο, κάποια ιδιαίτερα κυκλώματα που χρησιμοποιούνται όπως ο *επιλογέας* (Selector) ή ο *συγχρονιστής* (Synchronizer) περιγράφονται αναλυτικά, τουλάχιστον ως προς τη συμπεριφορά τους.

Όπως αναφέρθηκε ήδη, τα παραγόμενα κυκλώματα είναι σύγχρονα. Για το χρονισμό χρησιμοποιείται ένα καθολικό ρολόι το οποίο μεταδίδεται ταυτόχρονα σε όλα τα ακολουθιακά μέρη, και γι' αυτό δεν απασχολεί ιδιαίτερα τον σχεδιασμό. Έτσι, στις περισσότερες κυκλωματικές αναφορές δεν είναι ορατό, αλλά πρέπει να θεωρείται νοητά παρόν.

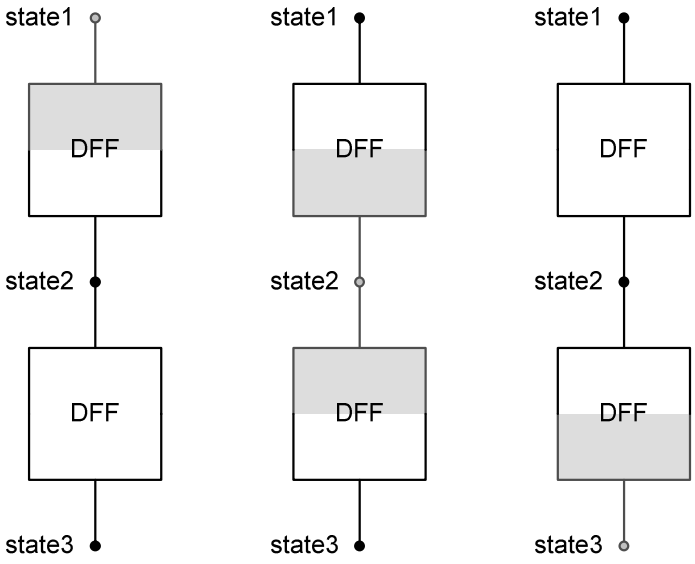
Το αποτέλεσμα της σύνθεσης μπορεί να θεωρηθεί ως μια μεγάλη *μηχανή πεπερασμένων καταστάσεων* (finite state machine – FSM). Πρόκειται δηλαδή για ένα αυτόματο το οποίο αλλάζει *καταστάσεις* (states) σε κάθε παλμό ρολογιού, και το οποίο εκτελεί την επεξεργασία των δεδομένων του προγράμματος. Πιο αναλυτικά, η προσέγγιση που ακολουθείται είναι αυτή της 1-hot-encoding FSM, δηλαδή οι καταστάσεις δεν είναι πολυπλεγμένες, παρά σε κάθε μία αντιστοιχεί και ένα σήμα/bit το οποίο έχει τιμή

1 όταν η συγκεκριμένη κατάσταση είναι ενεργή. Οι καταστάσεις παράγονται δυναμικά με βάση τον πηγαίο κώδικα του προγράμματος—χρησιμοποιούνται για τον *έλεγχο* (control), και γι' αυτό λέγονται και *καταστάσεις ελέγχου* (control states). Μαζί με τα κυκλώματα που χρησιμοποιούνται για τις μεταβάσεις από μία κατάσταση σε κάποια άλλη αποτελούν το λεγόμενο *μονοπάτι ελέγχου* (control path). Η απλούστερη μετάβαση μεταξύ δυο διαδοχικών καταστάσεων γίνεται με D Flip-Flop (DFF), όπως στο παρακάτω παράδειγμα.



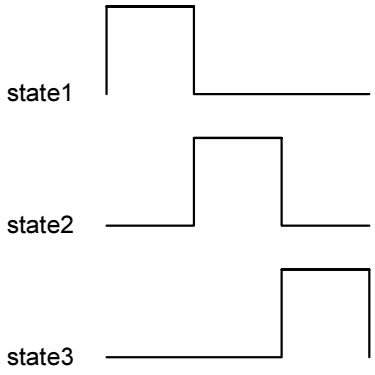
Εικόνα 8 – Ένα απλό κύκλωμα με τρεις καταστάσεις σε σειρά

Στο κύκλωμα αυτό υπάρχουν τρεις καταστάσεις (state1, state2 και state3) σε σειρά. Όταν σε κάποιο κύκλο ρολογιού το κύκλωμα βρεθεί στιγμιαία στην κατάσταση state1, δηλαδή το σήμα state1 έχει τιμή 1 μόνο για το συγκεκριμένο κύκλο ρολογιού, τότε στους επόμενους δύο κύκλους ρολογιού θα είναι ενεργές οι καταστάσεις state2 και state3 κατά σειρά, όπως δείχνουν τα γραμμοσκιασμένα τμήματα του παρακάτω σχήματος.



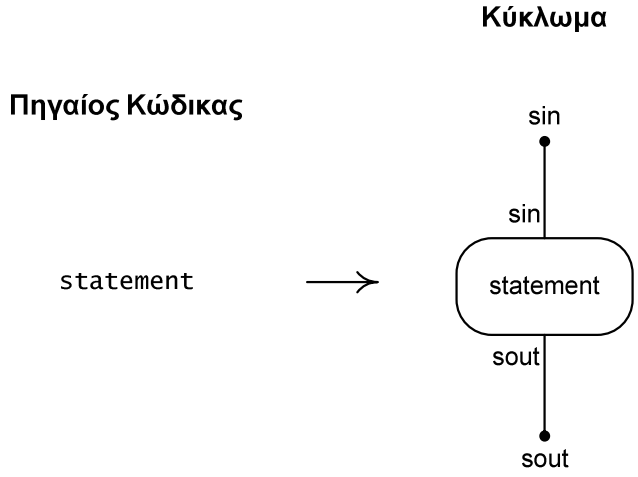
Εικόνα 9 – Τρία διαδοχικά στιγμιότυπα του κυκλώματος κατά την εκτέλεση του

Τα σήματα state1, state2 και state3, που συμβολίζουν τις τρεις αντίστοιχες καταστάσεις, μπορούν εν τω μεταξύ να χρησιμοποιούνται ως είσοδοι σε άλλα μέρη του κυκλώματος για να ενεργοποιήσουν τις λειτουργίες εκείνες που πρέπει να λάβουν χώρα σε κάθε κατάσταση. Οι τιμές τους, με την πάροδο του χρόνου, φαίνονται στο επόμενο σχήμα:



Εικόνα 10 – Οι τιμές των τριών σημάτων με την πάροδο του χρόνου

Βέβαια, στη γενικότερη περίπτωση οι μεταβάσεις μεταξύ των καταστάσεων δεν έχουν πάντοτε αυτή την απλή μορφή του ενός DFF, αλλά εξαρτώνται από τον κώδικα του προγράμματος C που μεταφράζεται σε κύκλωμα. Πιο αναλυτικά, κάθε εντολή του πηγαίου κώδικα C αντιστοιχίζεται σε ένα κύκλωμα το οποίο λαμβάνει μια *κατάσταση εισόδου* (input state – sin) και εξάγει μια *κατάσταση εξόδου* (output state – sout). Η σημασία των καταστάσεων αυτών είναι η αυτονόητη: το κύκλωμα της εντολής ενεργοποιείται μόλις ενεργοποιηθεί η κατάσταση εισόδου του, και σηματοδοτεί το τέλος της εκτέλεσής του ενεργοποιώντας την κατάσταση εξόδου του. Τη μετάβαση μεταξύ των καταστάσεων sin και sout τη χειρίζεται το ίδιο το κύκλωμα της εντολής, όπως φαίνεται στην επόμενη εικόνα.



Εικόνα 11 – Η αντιστοιχία μεταξύ μιας εντολής πηγαίου κώδικα και κυκλώματος

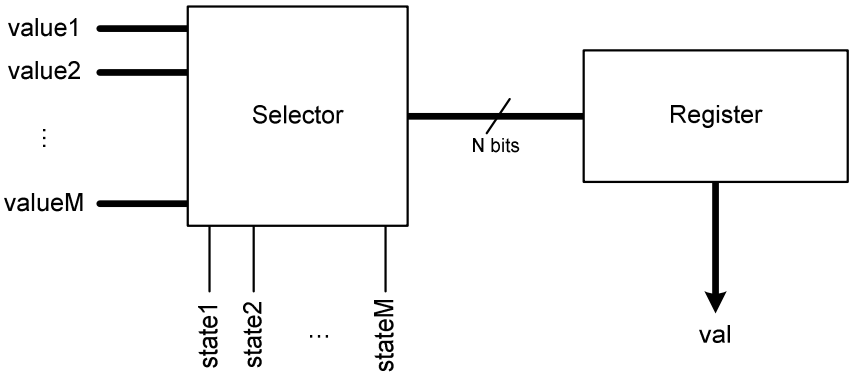
Το περιεχόμενο του κυκλώματος μεταξύ των δύο καταστάσεων αποτελεί ένα σημαντικό αντικείμενο μελέτης, και για κάθε εντολή πρέπει να δοθεί συγκεκριμένη *αντιστοιχία* (mapping) σε κύκλωμα. Όπως θα γίνει φανερό αργότερα, όπου και θα αναλυθεί κάθε εντολή της C ξεχωριστά, είναι δυνατό σε ορισμένες εντολές η μετάβαση με-

ταξύ των δύο καταστάσεων να μην απαιτεί επιπλέον παλμό ρολογιού αλλά να γίνεται στον ίδιο χρόνο, δηλαδή να υπάρχει συνδυαστικό μονοπάτι μεταξύ των σημάτων sin και sout (πράγμα που σημαίνει ότι η εντολή κατά την εκτέλεσή της δεν είχε να επιτελέσει καμία λειτουργία). Κάποιες άλλες εντολές μπορεί να χρειαστούν παραπάνω από έναν κύκλο ρολογιού, ενώ άλλες μπορεί να μην ενεργοποιήσουν ποτέ την κατάσταση εξόδου τους. Γενικότερα, κατά το σχεδιασμό αποφασίστηκε κάθε εντολή να απαιτεί όσο το δυνατό λιγότερους κύκλους ρολογιού για την εκτέλεσή της, και αυτή η φιλοσοφία ακολουθήθηκε παντού (με εξαίρεση ίσως την κενή εντολή της C). Η απόφαση αυτή οδηγεί σε κάποια σημαντικά συμπεράσματα, τα οποία σχολιάζονται σε επόμενη ενότητα.

Όπως περιγράφηκε συνολικά, η μορφή του μονοπατιού ελέγχου, που είναι κατανεμημένη (αφού δεν υπάρχει κάποιο κεντρικό κύκλωμα για το χειρισμό του ελέγχου και των καταστάσεων, παρά κάθε εντολή εισάγει επιτόπου τα δικά της συνδυαστικά και ακολουθιακά μέρη στη νοητή αυτή FSM) αποτελεί μια καλή επιλογή καθώς το παραγόμενο κύκλωμα προορίζεται κυρίως για διαμόρφωση πάνω σε FPGA, οι οποίες διαθέτουν ως γνωστόν flip-flops σε κάθε κελί τους, τα οποία θα έμεναν διαφορετικά αναξιοποίητα.

Η μορφή του μονοπατιού δεδομένων. Η κυκλωματική αναπαράσταση μιας μεταβλητής. Ο επιλογέας.

Εκτός από τον έλεγχο, ένα άλλο σημαντικό κομμάτι της σύνθεσης αποτελούν και τα *δεδομένα* (data). Επειδή το παραγόμενο κύκλωμα δεν χρησιμοποιεί κάποια δυναμική στοίβα ή μνήμη, για την αποθήκευσή των δεδομένων γίνεται χρήση καταχωρητών. Η ροή των δεδομένων γίνεται μέσα από το *μονοπάτι δεδομένων* (data path) που περιλαμβάνει όλες εκείνες τις μονάδες επεξεργασίας (αθροιστές, κλπ) που επενεργούν στα δεδομένα. Στην εργασία αυτή αποφασίστηκε να μη γίνεται κανενός είδους *διαμοίραση πόρων* (resource sharing), ούτε *χρονοδρομολόγηση* (scheduling): αντίθετα, κάθε φορά που εμφανίζεται για παράδειγμα μια πρόσθεση με τον τελεστή '+' στον πηγαίο κώδικα, παράγεται και ένας νέος αθροιστής ο οποίος εκτελεί αποκλειστικά την άθροιση των συγκεκριμένων ορισμάτων. Μελλοντική μελέτη θα μπορούσε να κάνει κάποια βήματα προς την εξοικονόμηση υλικού με την επαναχρησιμοποίηση κυκλωμάτων (αν και ακόμα και με την παρούσα κατάσταση μπορεί ο προγραμματιστής να περιγράψει μια τέτοια συμπεριφορά ορίζοντας μια πράξη μέσα σε μία συνάρτηση, και καλώντας το συγκεκριμένο κύκλωμα της συνάρτησης κάθε φορά που θέλει να εκτελέσει τη συγκεκριμένη πράξη).

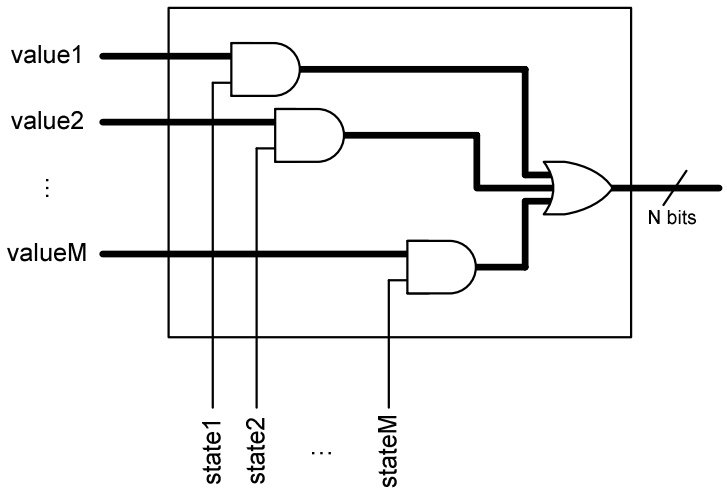


Εικόνα 12 – Το κύκλωμα που αντιστοιχεί σε μία μεταβλητή των N bits

Το τυπικό κύκλωμα που σχετίζεται με μία μεταβλητή των N bits φαίνεται στην εικόνα 12.

Το κύκλωμα αυτό αποτελείται από έναν καταχωρητή των N bits, ο οποίος έχει αποθηκευμένη την τρέχουσα τιμή της μεταβλητής. Η τιμή αυτή εξάγεται στο *σύρμα δεδομένων* (data wire) που στο σχήμα έχει ονομαστεί val, και το οποίο χρησιμοποιείται από το υπόλοιπο κύκλωμα για την ανάγνωση της τιμής της μεταβλητής ανά πάσα στιγμή χρειαστεί. Για την εγγραφή μιας τιμής πίσω στη μεταβλητή, ο καταχωρητής τροφοδοτείται από έναν *επιλογέα* (Selector). Ο επιλογέας εφοδιάζεται με μια σειρά από *ζεύγη τιμών-καταστάσεων* (value-state pairs). Κάθε τέτοιο ζεύγος περιλαμβάνει μια τιμή προς εγγραφή στην μεταβλητή, και την αντίστοιχη κατάσταση στην οποία πρέπει να γίνει η εγγραφή. Έτσι, αν μια μεταβλητή λαμβάνει συνολικά M τιμές σε M διαφορετικές καταστάσεις μέσα σε ένα πρόγραμμα, ο επιλογέας λαμβάνει ως είσοδο τις τιμές και τις καταστάσεις αυτές και εξάγει κάθε φορά την σωστή τιμή προς εγγραφή, ανάλογα με την κατάσταση εγγραφής που είναι ενεργή στην είσοδό του. Η εγγραφή καθαυτή πραγματοποιείται με την έλευση του παλμού ρολογιού στον καταχωρητή, και επομένως γίνεται ορατή στην έξοδο val αφού παρέλθει ένας κύκλος ρολογιού. Αν καμία από τις καταστάσεις του επιλογέα δεν είναι ενεργή, τότε στο συγκεκριμένο κύκλο ρολογιού δεν πραγματοποιείται εγγραφή στη μεταβλητή, και έτσι η μεταβλητή πρέπει να διατηρήσει την υπάρχουσα τιμή της. Αυτή η τελευταία απαίτηση μπορεί να ικανοποιηθεί είτε εισάγοντας ένα επιπλέον ζεύγος με τιμή (value) την ανατροφοδότηση από την έξοδο val του καταχωρητή, και κατάσταση (state) το NOR όλων των άλλων καταστάσεων (ώστε σε κάθε κύκλο να επανεγγράφεται η παλιά τιμή της μεταβλητής, αν δεν συμβαίνει νέα εγγραφή), είτε περνώντας το ρολόι του καταχωρητή μέσα από μια πύλη AND με το OR όλων των καταστάσεων, ώστε να εμποδίζεται το γράψιμο.

Μια απλοποιημένη εκδοχή για το εσωτερικό κύκλωμα ενός επιλογέα φαίνεται παρακάτω:



Εικόνα 13 – Απλοποιημένη εκδοχή του εσωτερικού κυκλώματος ενός επιλογέα

Στην πραγματικότητα ένας επιλογέας είναι πιο σύνθετος, προκειμένου να χειριστεί την προβληματική περίπτωση όπου σε κάποιον κύκλο ρολογιού εμφανίζονται περισσότερες από μία ενεργές καταστάσεις στην είσοδό του. Καθώς η γλώσσα επιτρέπει την παράλληλη εκτέλεση εντολών, είναι δυνατό (εξαιτίας κάποιου προγραμματιστι-

κού σφάλματος) σε κάποια χρονική στιγμή να ανατίθενται δύο διαφορετικές τιμές στην ίδια μεταβλητή! Σε μια τέτοια περίπτωση, θα ήταν συνετότερο ο επιλογέας αντί να διαλέξει το λογικό OR των δύο τιμών (όπως θα έκανε το κύκλωμα του παραπάνω σχήματος), να διαλέξει μία εκ των ανατιθέμενων τιμών εφαρμόζοντας κάποιον αυθαίρετο κανόνα προτεραιότητας.

Ορισμένες μεταβλητές ενδέχεται να λαμβάνουν άπαξ μια αρχική τιμή κατά την αρχικοποίηση του κυκλώματος, καθώς κάτι τέτοιο προβλέπεται από τη C όταν π.χ. δηλώνεται μια καθολική μεταβλητή με αρχική τιμή. Η τιμή αυτή εισάγεται τότε στον καταχωρητή με την ενεργοποίηση του σήματος *μηδενισμού* (reset) του κυκλώματος, και για λόγους απλοποίησης παραλείπεται από τα σχήματα.

Το κύκλωμα της εικόνας 12 αποτελεί ένα πολύ βασικό κύκλωμα μιας και χρησιμοποιείται για την αποθήκευση των παρακάτω οντοτήτων ενός προγράμματος:

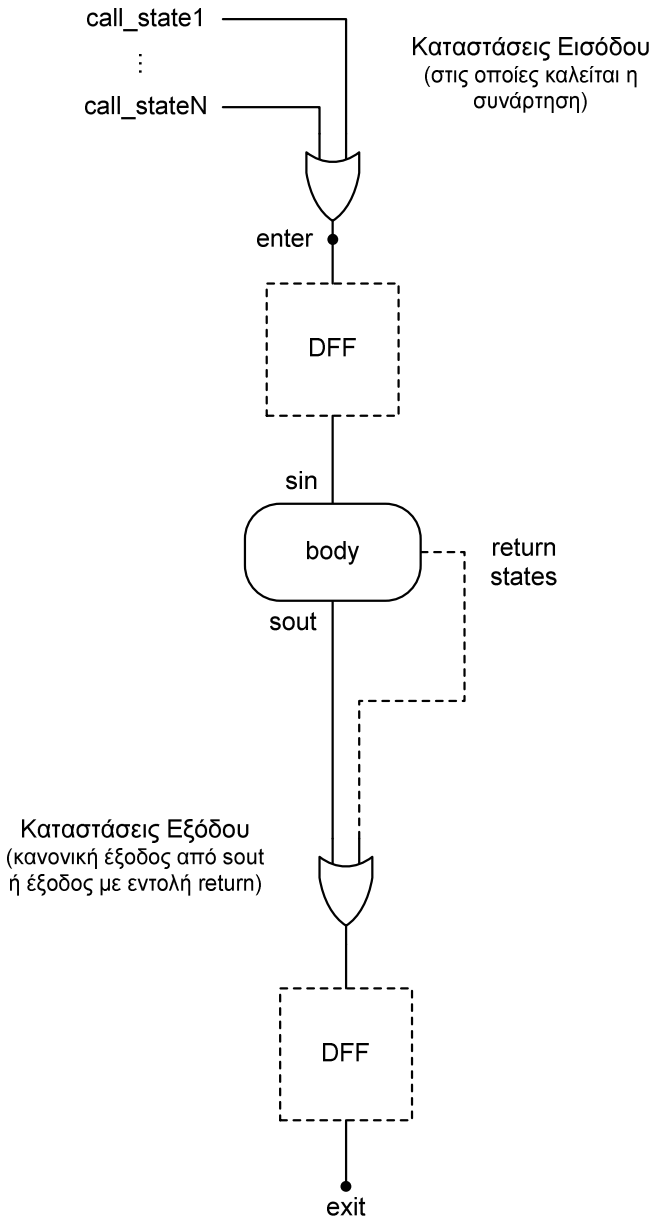
- *Καθολικές μεταβλητές* (global variables) του προγράμματος, που δεν είναι τύπου δείκτη (pointer)
- *Τοπικές μεταβλητές* (local variables) μιας συνάρτησης
- *Παράμετροι κατ' αξία* (parameters by value)
- *Τιμή επιστροφής* (return value) μιας συνάρτησης

Τα ζεύγη των τιμών και των καταστάσεων καθεμίας τέτοιας οντότητας συμπληρώνονται κατά τη μεταγλώττιση από τις εντολές που με τον ένα ή άλλο τρόπο πραγματοποιούν ανάθεση μιας τιμής. Για παράδειγμα, μια εντολή ανάθεσης με χρήση κάποιου τελεστή ανάθεσης της C δημιουργεί ένα νέο ζεύγος τιμής-κατάστασης που προσαρτάται στον επιλογέα μιας μεταβλητής, ενώ η εντολή return που επιστρέφει μια τιμή προσαρτά ένα αντίστοιχο ζεύγος στον επιλογέα της τιμής επιστροφής. Περισσότερα αναφέρονται στις ενότητες που περιγράφουν τις αντίστοιχες εντολές.

Οι *παράμετροι κατ' αναφορά* (parameters by reference), καθώς και οι καθολικές μεταβλητές τύπου δείκτη, δεν απαιτούν δικό τους αποθηκευτικό χώρο αφού αναφέρονται σε κάποια άλλη μεταβλητή. Η απεικόνισή τους σε κύκλωμα είναι πολυπλοκότερη, και περιγράφεται ξεχωριστά σε επόμενη ενότητα.

Ο σκελετός μιας συνάρτησης.

Η βασική μονάδα κώδικα σε ένα πρόγραμμα C είναι η συνάρτηση. Το κύκλωμα ελέγχου που αντιστοιχεί στο σκελετό μιας συνάρτησης φαίνεται στην εικόνα 14.



Εικόνα 14 – Το κύκλωμα ελέγχου που αντιστοιχεί στον σκελετό μιας συνάρτησης

Για την ενεργοποίηση της συνάρτησης συλλέγεται μια σειρά από καταστάσεις εισόδου, δηλαδή καταστάσεις από τις οποίες καλείται η συνάρτηση. Με άλλα λόγια, κάθε εντολή η οποία πραγματοποιεί κλήση στη συνάρτηση προσαρτά στη λίστα αυτή των καταστάσεων εισόδου την κατάσταση από την οποία γίνεται η κλήση. Το λογικό OR όλων αυτών των καταστάσεων αποτελεί την κατάσταση enter που τροφοδοτεί το σήμα εισόδου sin στο σώμα της συνάρτησης, δηλαδή το σήμα εκείνο που θα ενεργοποιήσει την εκτέλεση των εντολών της συνάρτησης. Το σήμα enter περνά πρώτα από ένα προαιρετικό DFF για την καθυστέρηση ενός κύκλου ρολογιού. Η καθυστέρηση αυτή στην είσοδο πρέπει να εισαχθεί στις παρακάτω περιπτώσεις:

– Όταν η συνάρτηση έχει παραμέτρους κατ' αξία. Αυτού του είδους οι παράμετροι, που όπως αναφέρθηκε χρησιμοποιούν το κύκλωμα της εικόνας 12, πρέπει να καταχωρήσουν την τιμή του αντίστοιχου ορίσματος με το οποίο κλήθηκαν πριν ξεκινήσει η εκτέλεση της συνάρτησης. Έτσι, όταν καλείται μια συ-

νάρτηση η οποία περιλαμβάνει παραμέτρους κατ' αξία, οι τιμές των αντίστοιχων ορισμάτων εισάγονται στον επιλογέα της κάθε παραμέτρου, με κατάσταση εγγραφής την κατάσταση στην οποία πραγματοποιείται η κλήση. Η εγγραφή πραγματοποιείται μόλις αλλάξει το ρολόι, οπότε το εισαγωγικό DFF είναι απαραίτητο προκειμένου η εκτέλεση του σώματος να ξεκινήσει στον επόμενο παλμό ρολογιού οπότε και θα είναι καταχωρημένες οι τιμές των ορισμάτων.

– Όταν η συνάρτηση έχει τοπικές μεταβλητές που έχουν δηλωθεί με αρχική τιμή. Ως γνωστόν στη γλώσσα C δίνεται η δυνατότητα για αρχικοποίηση των μεταβλητών κατά τη δήλωσή τους (η αρχικοποίηση αυτή στην παρούσα εργασία επιτρέπεται μόνο με *σταθερές* (constants) των οποίων η δυαδική αναπαράσταση σε bits υπολογίζεται κατά τη μεταγλώττιση). Όταν καλείται μια συνάρτηση που περιλαμβάνει τοπικές μεταβλητές οι οποίες αρχικοποιούνται με τη δήλωσή τους, πρέπει να εγγραφούν στις μεταβλητές αυτές οι αντίστοιχες αρχικές τιμές πριν ξεκινήσει να εκτελείται το σώμα της συνάρτησης. Έτσι, ο επιλογέας της κάθε τέτοιας μεταβλητής εφοδιάζεται με αντίστοιχο ζεύγος τιμής-κατάστασης, και το DFF είναι απαραίτητο για την καθυστέρηση μέχρι να καταχωρηθούν οι αλλαγές, όπως και στην προηγούμενη περίπτωση.

Το *σώμα* (body) της συνάρτησης δεν είναι παρά ένα μπλοκ εντολών (σειριακό ή παράλληλο), και επομένως αντιστοιχεί σε κύκλωμα εντολής (όπως φαίνεται γενικά στην εικόνα 11). Η κατάσταση εξόδου sout του σώματος της συνάρτησης είναι στην πιο απλή περίπτωση η κατάσταση εξόδου της τελευταίας εντολής. Ωστόσο, η έξοδος από τη συνάρτηση δεν γίνεται μόνο όταν η ροή του ελέγχου φτάσει και εκτελέσει την τελευταία εντολή της συνάρτησης—η έξοδος θα πρέπει να πραγματοποιείται και σε κάθε εμφάνιση της εντολής return. Πράγματι, μια συνάρτηση μπορεί να περιέχει ένα ή περισσότερα εναλλακτικά σημεία εξόδου μέσω αντίστοιχων εντολών return. Κάθε φορά που απαντάται η εντολή return μέσα στο σώμα της συνάρτησης, η συγκεκριμένη κατάσταση στην οποία εμφανίζεται η εντολή εισάγεται στη λίστα των *καταστάσεων επιστροφής* (return states) της συνάρτησης. Οι καταστάσεις αυτές, μαζί με την κατάσταση sout του σώματος της συνάρτησης, αποτελούν τις καταστάσεις εξόδου της συνάρτησης και περνούν από πύλη OR ώστε να προκύψει το τελικό σήμα εξόδου exit. Η κατάσταση exit σηματοδοτεί το τέλος της εκτέλεσης της συνάρτησης. Ωστόσο, υπάρχει κι εδώ ένα προαιρετικό DFF μέσα από το οποίο περνά το σήμα εξόδου. Το DFF αυτό είναι παρόν στις συναρτήσεις εκείνες οι οποίες επιστρέφουν κάποια τιμή (δηλαδή στις συναρτήσεις των οποίων ο τύπος επιστροφής δεν είναι void). Μία τέτοια συνάρτηση (που όπως έχει αναφερθεί συμπεριλαμβάνει ένα κύκλωμα επιλογέα-καταχωρητή σαν αυτό της εικόνας 12 για την τιμή επιστροφής) περιλαμβάνει εντολές return που συνοδεύονται από μια τιμή προς επιστροφή. Η εμφάνιση της εντολής return, εκτός από την εισαγωγή της κατάστασής της στις καταστάσεις εξόδου της συνάρτησης, απαιτεί και την εισαγωγή ενός αντίστοιχου ζεύγους τιμής-κατάστασης στον επιλογέα που έχει κατασκευαστεί για την τιμή επιστροφής της συνάρτησης. Για την ολοκλήρωση της καταχώρησης της τιμής επιστροφής είναι απαραίτητο το DFF στην έξοδο ώστε όταν ενεργοποιηθεί η κατάσταση εξόδου exit, να είναι ήδη διαθέσιμη η τιμή επιστροφής στον αντίστοιχο καταχωρητή.

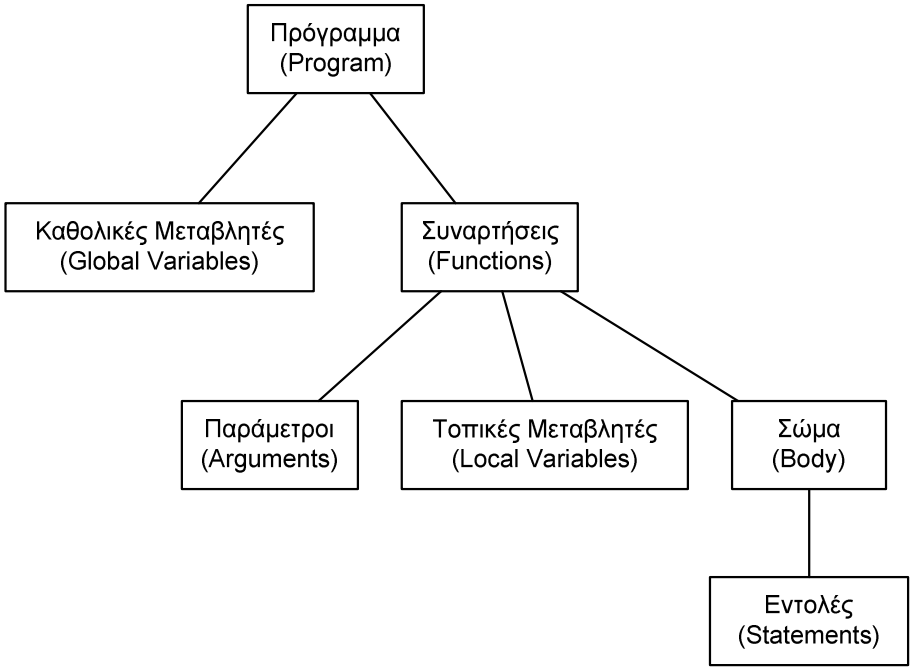
Συμβάσεις σύνταξης.

Για την περιγραφή της σύνταξης της γλώσσας χρησιμοποιούνται οι εξής συμβάσεις:

- Με **έντονη γραφή** αναπαρίστανται σύμβολα που πρέπει να εμφανίζονται αυτούσια στο πρόγραμμα και τα οποία αποτελούν τερματικά σύμβολα της γλώσσας. Σε αυτά συμπεριλαμβάνονται και οι αγκύλες, όταν είναι έντονες.
- Με περίκλειση σε <γωνίες> αναπαρίστανται σύμβολα που χρειάζονται περαιτέρω αντικατάσταση και τα οποία αποτελούν ενδιάμεσα σύμβολα της γλώσσας. Η ύπαρξη ενός αστερίσκου (*) αμέσως μετά τη γωνία κλεισίματος (>) δηλώνει ότι το σύμβολο μπορεί να συμμετέχει καμία, μία ή περισσότερες φορές στη σύνταξη της εντολής.
- Με περίκλειση σε [αγκύλες] αναπαρίστανται τα προαιρετικά στοιχεία, δηλαδή αυτά που μπορούν να παραληφθούν κατά τη σύνταξη.

Τα μέρη ενός προγράμματος.

Τα συστατικά μέρη ενός προγράμματος φαίνονται στην επόμενη εικόνα.



Εικόνα 15 – Τα συστατικά μέρη του προγράμματος

Στην κορυφή του προγράμματος απαντώνται οι καθολικές μεταβλητές και οι συναρτήσεις. Οι οντότητες αυτές βρίσκονται στο ίδιο επίπεδο από άποψη εμβέλειας, και έτσι δεν γίνεται να έχουν κοινά ονόματα.

Οι καθολικές μεταβλητές. Οι υποστηριζόμενοι τύποι της γλώσσας. Οι σταθερές.

Η δήλωση μίας καθολικής μεταβλητής γίνεται με την παρακάτω σύνταξη:

```
<type> [ * ] <identifier> [ = <constant> ] ;
```

Το *αναγνωριστικό* (identifier) της μεταβλητής, δηλαδή το όνομά της, ακολουθεί τους κλασικούς κανόνες αναγνωριστικών της C: πρέπει να ξεκινά με γράμμα, και επιτρέπεται να περιέχει γράμματα, αριθμητικά ψηφία, καθώς και τον χαρακτήρα '_' της *κάτω παύλας* (underscore). Ο προαιρετικός αστερίσκος πριν το αναγνωριστικό χρησιμοποιείται για τη δήλωση μιας καθολικής μεταβλητής τύπου δείκτη. Μια τέτοια μεταβλητή δε διαθέτει αποθηκευτικό χώρο στο κύκλωμα και χρησιμοποιείται ως φορμαλισμός για τον καθορισμό αμφίδρομης επικοινωνίας του προγράμματος με το εξωτερικό κύκλωμα. Επειδή η καθολική μεταβλητή τύπου δείκτη είναι πολυπλοκότερη ως προς την εξήγησή της, προς το παρόν θα δοθεί περισσότερη έμφαση στις απλές καθολικές μεταβλητές, ενώ η καθολική μεταβλητή τύπου δείκτη θα εξεταστεί αργότερα, μαζί με τις παραμέτρους κατ' αναφορά (που ως γνωστόν είναι επίσης δείκτες).

Ο *τύπος* (type) της μεταβλητής αντιπροσωπεύει τις απαραίτητες πληροφορίες για την αποθήκευση της μεταβλητής και τον χειρισμό της μέσα σε παραστάσεις. Για απλότητα η γλώσσα υποστηρίζει στην ουσία μόνο έναν τύπο: το διάνυσμα από bits. Ο θεμελιώδης τύπος 'bit' συμβολίζει ένα ακριβώς bit, ενώ για την δήλωση διανύσματος από N bits χρησιμοποιείται η σύνταξη 'bit <N>', δηλαδή η λέξη-κλειδί bit ακολουθούμενη από τον αριθμό των bits μέσα σε <γωνίες>. Η σύνταξη αυτή επελέγη επειδή θυμίζει την αντίστοιχη σύνταξη των *προτύπων* (templates) της γλώσσας C++. Οι θεμελιώδεις τύποι char και int έχουν παραμείνει ως συντομογραφίες των 'bit <8>' και 'bit <32>' αντίστοιχα, δηλαδή συμβολίζουν ένα διάνυσμα των 8 ή 32 bits. Οποιοιδήποτε άλλοι τύποι, όπως σύνθετες δομές (structs) ή πίνακες (arrays) δεν υποστηρίζονται.

Οι τύποι μπορούν να συνοδεύονται από έναν ή περισσότερους *προσδιοριστές* (qualifiers). Οι προσδιοριστές 'signed' και 'unsigned' χρησιμοποιούνται για να καθορίσουν αν ο τύπος είναι *προσημασμένος* ή *απρόσημος* αντίστοιχα. Μία προσημασμένη μεταβλητή των N bits χρησιμοποιεί την παράσταση συμπληρώματος ως προς 2 και μπορεί να παραστήσει τους ακέραιους αριθμούς από -2^{N-1} ως $2^{N-1} - 1$, ενώ μια απρόσημη μεταβλητή μπορεί να παραστήσει τους ακέραιους αριθμούς από 0 ως $2^N - 1$. Ο χαρακτηρισμός μιας μεταβλητής ως προσημασμένης ή απρόσημης είναι πολύ σημαντικός διότι καθορίζει τη συμπεριφορά πολλών τελεστών της γλώσσας όταν δρουν σε αυτή. Η απουσία προσδιοριστή προσήμου σε έναν τύπο χαρακτηρίζει αυτομάτως τον τύπο ως προσημασμένο (signed), όπως είθισται και στις περισσότερες υλοποιήσεις της C.

Οι προσδιοριστές 'short' και 'long', που εφαρμόζονται μόνο στον τύπο int, χρησιμοποιούνται για την αλλαγή του μεγέθους του τύπου. Συγκεκριμένα, ο προσδιοριστής 'short' μειώνει το μέγεθος του διανύσματος από τα 32 στα 16 bits, ενώ ο προσδιοριστής 'long' δεν επιφέρει αλλαγή (δηλαδή οι τύποι 'int' και 'long int' έχουν το ίδιο μέγεθος των 32 bits). Επίσης, αν στη σύνταξη του τύπου εμφανίζονται κάποιοι από τους προσδιοριστές 'short', 'long', 'unsigned' ή 'signed', αλλά παραλείπεται ο ίδιος ο τύπος (bit, char ή int), τότε εννοείται ο τύπος int.

Οι προσδιοριστές ‘auto’, ‘extern’, ‘register’, ‘const’ και ‘volatile’ αγνοούνται. Έχουν παραμείνει στο συντακτικό της γλώσσας μόνο για λόγους συμβατότητας. Ο προσδιοριστής ‘static’ χρησιμοποιείται για την απόκρυψη της μεταβλητής από τυχόν εξωτερικά κυκλώματα, διότι εξ’ ορισμού τα σήματα των τιμών των καθολικών μεταβλητών εξάγονται, δηλαδή είναι διαθέσιμα προς ανάγνωση από εξωτερικά κυκλώματα που μπορούν να συνδεθούν με το κύκλωμα του προγράμματος.

Η *σταθερά* (constant) μπορεί να δοθεί προαιρετικά για την αρχικοποίηση της μεταβλητής (εφόσον βέβαια δεν πρόκειται για καθολική μεταβλητή τύπου δείκτη). Οι σταθερές που υποστηρίζονται είναι τριών ειδών, αλλά όλες ανάγονται σε μία σειρά από bits, και περιγράφονται παρακάτω:

- **Σταθερές ASCII**: Οι σταθερές ASCII έχουν μήκος 8 bits, θεωρούνται απρόσημες παραστάσεις, και αντανακλούν τη δυαδική αναπαράσταση ενός ASCII χαρακτήρα, ο οποίος γράφεται μέσα σε μονά εισαγωγικά. Για παράδειγμα, η σταθερά ΄b΄ αντιστοιχεί στο δυαδικό διάνυσμα 01100010. Ο μεταγλωττιστής υποστηρίζει τις γνωστές *ακολουθίες διαφυγής* (escape sequences) όπως ΄\n΄, ΄\t΄, κ.α.

- **Δεκαδικές σταθερές**: Οι δεκαδικές σταθερές γράφονται με έναν δεκαδικό αριθμό, ο οποίος μπορεί να ξεκινά με το αρνητικό πρόσημο ‘–’ (για τη δήλωση αρνητικών αριθμών). Ένας θετικός αριθμός καταλαμβάνει ακριβώς τόσα bits όσα απαιτούνται για την παράστασή του. Έτσι, οι σταθερές 0, 1, 2 και 17 αντιστοιχούν στα δυαδικά διανύσματα 0, 1, 10 και 10001, και θεωρούνται απρόσημες. Οι αρνητικοί αριθμοί χρησιμοποιούν τον τρόπο παράστασης συμπληρώματος ως προς 2, ενώ ισχύει και για αυτούς ο κανόνας ότι παριστάνονται με τα λιγότερα δυνατά bits. Για παράδειγμα, οι σταθερές -1, -3 και -17 αντιστοιχούν στα δυαδικά διανύσματα 1, 101, και 101111. Οι αρνητικές δεκαδικές σταθερές θεωρούνται προσημασμένες παραστάσεις, ώστε να υφίστανται *επέκταση προσήμου* (sign extension) όποτε απαιτείται να προσαρμοστούν σε ένα μεγαλύτερο μέγεθος. Τέλος, ας σημειωθεί ότι δε μπορεί να εισαχθεί μια οσοδήποτε μεγάλη δεκαδική σταθερά, διότι πρέπει να βρίσκεται μεταξύ γνωστών ορίων της εκάστοτε υλοποίησης.

- **Δεκαεξαδικές σταθερές**: Οι δεκαεξαδικές σταθερές αποτελούν μια επέκταση–αλλαγή από τη γλώσσα C ώστε να μπορεί κανείς να εισάγει ένα δυαδικό διάνυσμα μεγάλου μεγέθους. Οι σταθερές αυτές εισάγονται ως *αλφαριθμητικά* (strings) της γλώσσας C, δηλαδή περικλείονται σε διπλά εισαγωγικά, εντός των οποίων περιέχονται μόνο δεκαεξαδικά ψηφία (κεφαλαία ή πεζά). Η επιλογή αυτή έγινε διότι τα κανονικά ASCII strings της C δεν έβρισκαν και τόσο μεγάλη χρησιμότητα, ενώ στη θέση τους μπορεί να περιγραφεί ένα οσοδήποτε μεγάλο δυαδικό διάνυσμα και μάλιστα με έναν αρκετά συμπαγή τρόπο γραφής. Οι δεκαεξαδικές σταθερές είναι απρόσημες και χρησιμοποιούν ακριβώς όσα bits απαιτούνται για την παράστασή τους. Για παράδειγμα, οι σταθερές "ff", "101" και "13CC" αντιστοιχούν στα δυαδικά διανύσματα 11111111, 100000001 και 1001111001100.

Αν μια μεταβλητή μεγέθους N αρχικοποιείται με σταθερά μεγαλύτερου μεγέθους, τότε η σταθερά αποκόπτεται και κρατούνται τα N δεξιότερα (λιγότερα σημαντικά) bits αυτής. Στην αντίθετη περίπτωση, όταν η σταθερά έχει μικρότερο μέγεθος από τη με-

ταβλητή, τότε επεκτείνεται προς τα αριστερά μέχρι τα N bits. Αν η σταθερά είναι απρόσημη, η επέκταση πραγματοποιείται προσαρτώντας μηδενικά bits, ενώ αν είναι προσημασμένη συμβαίνει επέκταση προσήμου, δηλαδή αντιγράφεται το αριστερότερο (περισσότερο σημαντικό) bit της σταθεράς ώστε να διατηρηθεί το πρόσημο.

Η δήλωση μιας καθολικής μεταβλητής που δεν είναι τύπου δείκτη εισάγει το κύκλωμα της εικόνας 12, δηλαδή έναν καταχωρητή αντίστοιχου μήκους και έναν επιλογέα στον οποίο θα προσαρτηθούν τα ζεύγη τιμών-καταστάσεων από τις αντίστοιχες εντολές που αναθέτουν τιμή στη μεταβλητή. Όλες οι καθολικές μεταβλητές περιλαμβάνουν αρχική τιμή – αν μια καθολική μεταβλητή δε λαμβάνει συγκεκριμένη αρχική τιμή κατά τη δήλωσή της, θα αρχικοποιηθεί με μηδενική τιμή. Η αρχική τιμή εισάγεται άπαξ στον καταχωρητή όταν ενεργοποιείται το κύκλωμα, συνήθως μέσω των σημάτων *μηδενισμού* (reset) των flip-flops του καταχωρητή. Η ακριβής μέθοδος για την αρχικοποίηση ενός καταχωρητή σε συγκεκριμένη τιμή όταν γίνεται reset το κύκλωμα αποτελεί λεπτομέρεια της υλοποίησης και δεν εξετάζεται εδώ περαιτέρω.

Για λόγους απλότητας, περιγράφηκε η σύνταξη για τον ορισμό ακριβώς μίας καθολικής μεταβλητής σε μία δήλωση. Ωστόσο, στην ίδια δήλωση μπορούν να ορίζονται περισσότερες καθολικές μεταβλητές του ίδιου τύπου, με σύνταξη σαν την ακόλουθη:

```
<type> [ * ] <id> [ = <const> ] ,  …  , [ * ] <id> [ = <const> ] ;
```

Παραδείγματα ορισμών καθολικών μεταβλητών:

```
bit Stall = 1;
unsigned short int Counter, MaxCount = "ffff";
signed bit<64> regA, regB;
```

Οι συναρτήσεις. Η δήλωση των παραμέτρων και του τύπου επιστροφής. Η παραγωγή πολλαπλών αντιτύπων για αναδρομή. Το παράλληλο σώμα εντολών.

Στο ίδιο επίπεδο εμβέλειας με τις καθολικές μεταβλητές εμφανίζονται και οι συναρτήσεις. Για λόγους απλοποίησης, δεν υποστηρίζονται από το μεταγλωττιστή οι δηλώσεις πρωτοτύπων μιας συνάρτησης, διότι δεν είναι και αναγκαίες. Όπως είναι γνωστό, στη C μπορεί κανείς να δηλώσει το πρωτότυπο μιας συνάρτησης πριν ορίσει τη συνάρτηση, σε περίπτωση που θέλει να τη χρησιμοποιήσει μέσα σε άλλες συναρτήσεις των οποίων οι ορισμοί προηγούνται. Η δήλωση αυτή πλέον δεν είναι απαραίτητη, καθώς ο μεταγλωττιστής εκτελεί πρώτα ένα πέρασμα εισάγοντας όλα τα ονόματα των οντοτήτων του πρώτου επιπέδου (καθολικών μεταβλητών και συναρτήσεων) στον *πίνακα συμβόλων* (symbol table) που διαθέτει, ώστε στο δεύτερο κανονικό πέρασμα να μπορεί να χειρίζεται περιπτώσεις στις οποίες μια συνάρτηση καλεί μία άλλη που ορίζεται αργότερα. Έτσι, οι δηλώσεις των συναρτήσεων πρέπει να συμπεριλαμβάνουν και τον ορισμό του σώματος τους, σύμφωνα με την παρακάτω σύνταξη:

```
[ <return_type> ] <identifier> ( [<parameters>] ) [ [ <depth> ] ]
[ par ]
{
  <local_variable>*

  <statement>*
}
```

Το *αναγνωριστικό* (identifier) της συνάρτησης ακολουθεί τους κανόνες που περιγράφηκαν πρωτύτερα για το αναγνωριστικό των καθολικών μεταβλητών. Ο *τύπος επιστροφής* (return type) αναφέρεται στον τύπο των δεδομένων που επιστρέφει η συνάρτηση. Ο τύπος αυτός περιλαμβάνει όλους τους τύπους που έχουν ήδη αναφερθεί για τις καθολικές μεταβλητές, καθώς και τον ειδικό τύπο 'void' με τον οποίο συμβολίζονται οι συναρτήσεις που δεν επιστρέφουν καμία τιμή. Όταν ο τύπος επιστροφής παραλείπεται, εννοείται ο τύπος 'int'.

Η συνάρτηση μπορεί να περιλαμβάνει προαιρετικά μία ή περισσότερες *παραμέτρους* (parameters) που δηλώνονται εντός των παρενθέσεων. Η σύνταξη που χρησιμοποιείται για τη δήλωσή τους είναι η ακόλουθη:

```
<type> [ * ] <identifier> ,  …  , <type> [ * ] <identifier>
```

Για τον τύπο και το αναγνωριστικό της κάθε παραμέτρου ισχύουν οι γνωστοί πλέον κανόνες. Ο αστερίσκος '*' εισάγεται προαιρετικά πριν από κάποιο αναγνωριστικό για τη δήλωση ενός *δείκτη* (pointer), ορίζοντας έτσι μια παράμετρο κατ' αναφορά (ενώ η απουσία του δηλώνει παράμετρο κατ' αξία). Περισσότερα για τις παραμέτρους και τη σχέση τους με τη διαδικασία της σύνθεσης θα αναφερθούν αργότερα.

Μετά από τις παρενθέσεις γράφεται προαιρετικά ένας ακέραιος θετικός αριθμός μέσα σε αγκύλες, για να δηλώσει το πλήθος των αντιτύπων της συνάρτησης. Η απουσία του σημαίνει ότι η συνάρτηση θα παραχθεί ακριβώς μία φορά σε κύκλωμα, διαφορετικά θα αναπαραχθούν τόσα κυκλωματικά αντίτυπα όσα ορίζονται. Πρόκειται για μια επέκταση της σύνταξης της C που έγινε με κύριο στόχο την υποστήριξη της αναδρομής. Πράγματι, όπως έχει αναφερθεί, η δυνατότητα της αναδρομικής κλήσης μιας συνάρτησης δεν είναι εύκολα εφικτή στο κυκλωματικό επίπεδο. Αν η αναδρομική κλήση επαναχρησιμοποιούσε το ίδιο κύκλωμα της συνάρτησης, θα έπρεπε να αποθηκευθούν τα προηγούμενα τοπικά δεδομένα σε μια δυναμική δομή, όπως γίνεται με τη στοίβα εκτέλεσης στους κοινούς υπολογιστές. Επειδή ο σχεδιασμός στοχεύει στην παραγωγή σταθερού υλικού, η λύση που δίδεται για την αναδρομή είναι η παραγωγή πολλαπλών αντιτύπων της συνάρτησης. Μια αναδρομική κλήση της συνάρτησης καλεί στην πράξη το επόμενο κατά σειρά αντίτυπο της, δίνοντας στον προγραμματιστή την αίσθηση της αναδρομής, με τίμημα το επιπλέον υλικό που απαιτείται για κάθε αντίτυπο. Βέβαια, για να λειτουργήσει σωστά αυτό το μοντέλο, θα πρέπει τα αντίτυπα της συνάρτησης να είναι τουλάχιστον τόσα όσο και το μέγιστο βάθος της αναδρομής, γι' αυτό και το πλήθος των αντιτύπων έχει ονομαστεί στη σύνταξη ως *βάθος* (depth).

Μερικά παραδείγματα για το μέρος της σύνταξης που έχει μέχρι στιγμής αναλυθεί:

```
void main()
int power(int base, unsigned exp)
void mult16(short *a, short *b, short *result)
void fibonacci(int *n)[10]
```

Η λέξη-κλειδί 'par' που μπορεί να εισαχθεί προαιρετικά πριν από το σώμα της συνάρτησης δηλώνει ότι ακολουθεί ένα παράλληλο σώμα εντολών, ενώ η απουσία της εννοεί ένα σειριακό σώμα εντολών. Σε ένα σειριακό σώμα εντολών, οι *εντολές* (statements) εκτελούνται ακολουθιακά, καθεμία μετά το πέρας της προηγούμενης, με τη σειρά που είναι γραμμένες, όπως στην κλασική C. Αυτή η σημασιολογία της C όμως οφείλει να επεκταθεί για τις ανάγκες της σύνθεσης κυκλωμάτων, προκειμένου

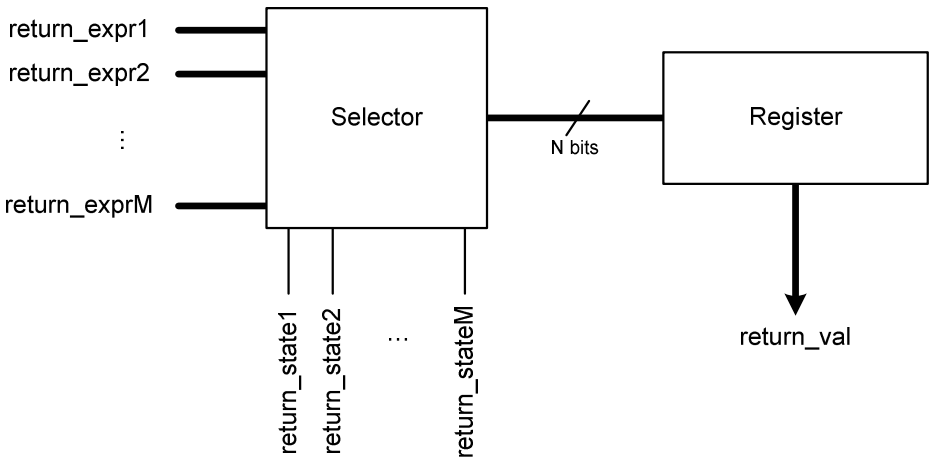
να εκμεταλλευθεί κανείς τη δυνατότητα παράλληλης εκτέλεσης που παρέχεται εγγενώς σε ένα κύκλωμα. Έτσι έχει εισαχθεί στη γλώσσα η λέξη-κλειδί ‘par’ για να προσδίδει σημασιολογία παράλληλης εκτέλεσης στο σώμα των εντολών. Σε ένα παράλληλο σώμα εντολών, οι εντολές εκτελούνται παράλληλα, δηλαδή ξεκινούν όλες ταυτόχρονα, και δεν έχει σημασία η σειρά με την οποία είναι γραμμένες. Η λέξη-κλειδί ‘par’ εφαρμόζεται σε οποιοδήποτε μπλοκ εντολών της C και όχι μόνο στο σώμα εντολών μιας συνάρτησης. Περισσότερα αναφέρονται στην ενότητα περιγραφής των εντολών της γλώσσας.

Όπως αναφέρθηκε, το σώμα της συνάρτησης περιλαμβάνει τις *εντολές* (statements) προς εκτέλεση. Των εντολών αυτών όμως προηγείται η δήλωση των *τοπικών μεταβλητών* (local variables) της συνάρτησης. Οι τοπικές μεταβλητές μιας συνάρτησης δηλώνονται με σύνταξη ίδια με αυτήν των καθολικών μεταβλητών (δεν επιτρέπονται όμως δείκτες). Για όσες από αυτές δηλώνονται με αρχική τιμή, η τιμή αυτή δεν είναι τιμή που λαμβάνουν άπαξ κατά την έναρξη της λειτουργίας του κυκλώματος, αλλά τιμή που οφείλουν να λαμβάνουν κάθε φορά που καλείται η συνάρτηση και ξεκινά την εκτέλεσή της. Η λεπτομέρεια αυτή έχει αντίκτυπο στο παραγόμενο κύκλωμα και εξετάζεται στη συνέχεια, μαζί με τα υπόλοιπα κυκλωματικά μέρη της συνάρτησης.

Η αναπαράσταση της συνάρτησης. Το κύκλωμα της τιμής επιστροφής.

Η ολική σύνθεση μιας συνάρτησης σε κύκλωμα πραγματοποιείται μεταφράζοντας τα διάφορα μέρη της (σώμα εντολών, τιμή επιστροφής, παράμετροι, κλπ). Ο σκελετός της συνάρτησης, που φαίνεται στην εικόνα 14, αποτελεί το βασικό κύκλωμα ελέγχου της συνάρτησης. Οι καταστάσεις εισόδου της συνάρτησης συμπληρώνονται σταδιακά κατά τη διάρκεια της σύνθεσης με όλα εκείνα τα σημεία στα οποία πραγματοποιείται κλήση προς τη συγκεκριμένη συνάρτηση, ενώ οι καταστάσεις εξόδου (που αρχικά περιλαμβάνουν μόνο την κατάσταση εξόδου του σώματος εντολών) συμπληρώνονται με τα σήματα ‘return states’ από τις εντολές return που περιέχονται στο σώμα της συνάρτησης.

Αν η συνάρτηση περιλαμβάνει τιμή επιστροφής, τότε συντίθεται ένα κλασικό κύκλωμα μεταβλητής (που δόθηκε στην εικόνα 12) για την αποθήκευση της τιμής επιστροφής. Για ευκολία, το κύκλωμα αυτό απεικονίζεται παρακάτω με συγκεκριμένα ονόματα σημάτων:



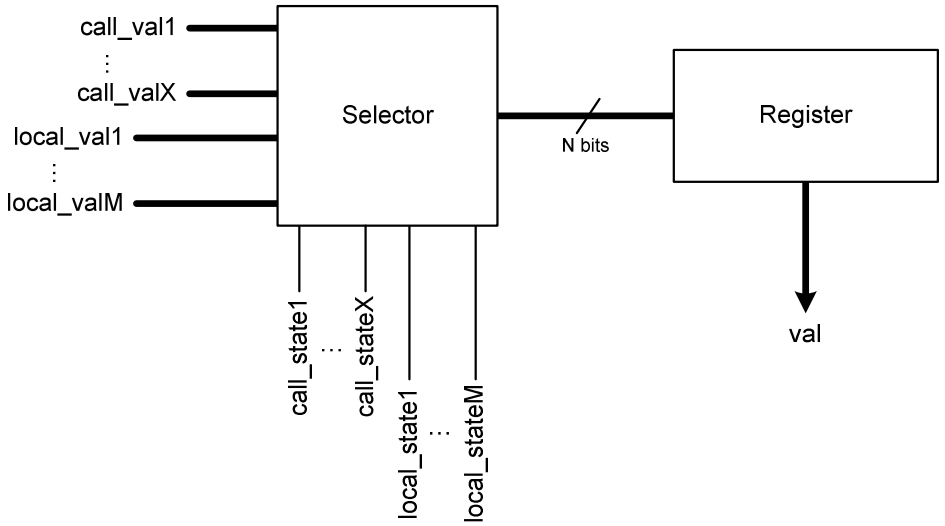
Εικόνα 16 – Το κύκλωμα για την τιμή επιστροφής

Οι καταστάσεις του επιλογέα δεν είναι άλλες από τις καταστάσεις επιστροφής (return states) που προέρχονται από τις εντολές return. Οι αντίστοιχες τιμές που λαμβάνει ο επιλογέας είναι οι τιμές των παραστάσεων που ακολουθούν την εκάστοτε εντολή return. Με άλλα λόγια, όποτε συναντάται μία εντολή return με μία παράσταση προς επιστροφή, αρχικά αποτιμάται η παράσταση και παράγεται μια τιμή return_expr, και μαζί με την αντίστοιχη κατάσταση return_state στην οποία βρέθηκε η εντολή return εισάγονται στον επιλογέα. Όταν η συνάρτηση επιστρέψει στο σημείο από όπου εκλήθη (μετά από ένα παλμό ρολογιού χάρη στο DFF εξόδου που υπάρχει για αυτό το λόγο στο κύκλωμα ελέγχου, ώστε να εγγραφεί η τιμή επιστροφής στον καταχωρητή), το σήμα return_val που εξάγεται από τον καταχωρητή θα περιέχει το αποτέλεσμα της συνάρτησης και θα είναι έτοιμο για χρήση από το υπόλοιπο πρόγραμμα που κάλεσε τη συνάρτηση.

Το κύκλωμα των παραμέτρων κατ' αξία.

Αν η συνάρτηση περιέχει παραμέτρους, τότε για καθεμία εξ' αυτών συντίθεται και ένα αντίστοιχο κύκλωμα. Ως γνωστόν, υπάρχουν δύο ειδών παράμετροι, που αντιστοιχούν και σε διαφορετικά κυκλώματα.

Οι *παράμετροι κατ' αξία* (parameters by value) διαθέτουν τοπικό αποθηκευτικό χώρο στη συνάρτηση. Όταν καλείται η συνάρτηση, τότε για μια τέτοια παράμετρο μεταβιβάζεται μόνο η *αξία* (value) του ορίσματος, δηλαδή αποτιμάται το αντίστοιχο όρισμα με το οποίο πραγματοποιείται η κλήση (που γενικά μπορεί να είναι μια οποιαδήποτε παράσταση), και στέλνεται μόνο η τιμή του στη συνάρτηση. Στη συνέχεια, όποιες αλλαγές πραγματοποιήσει η συνάρτηση κατά την εκτέλεση της στην παράμετρο αυτή θα είναι τοπικές, δηλαδή χειρίζεται την παράμετρο σαν τοπική μεταβλητή. Αυτή είναι γενικά η σημασιολογία της κλήσης κατ' αξία, η οποία διατηρείται και στην κυκλωματική αναπαράσταση χωρίς ιδιαίτερα προβλήματα. Το κύκλωμα που συντίθεται για μια παράμετρο κατ' αξία δεν είναι άλλο από το γνωστό αποθηκευτικό κύκλωμα του επιλογέα-καταχωρητή, που απεικονίζεται εκ νέου παρακάτω:



Εικόνα 17 – Το κύκλωμα που αντιστοιχεί σε μια παράμετρο κατ' αξία

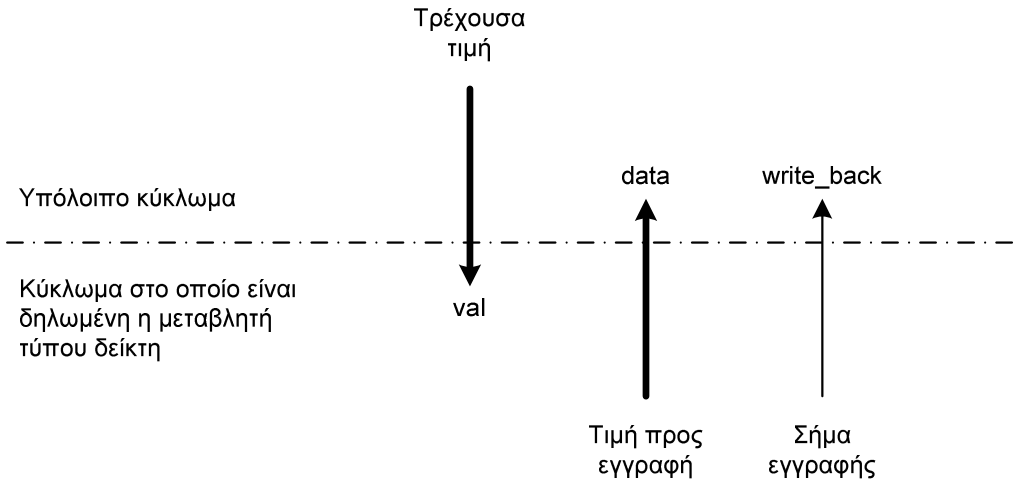
Όπως φαίνεται, τα ζεύγη τιμών-καταστάσεων του επιλογέα προέρχονται από δύο διαφορετικές πηγές. Τα πρώτα ζεύγη, τα μέλη των οποίων έχουν ονομαστεί call_val

και call_state στο σχήμα, προέρχονται από τις κλήσεις που πραγματοποιούνται προς τη συνάρτηση. Κάθε φορά που καλείται η συνάρτηση, η τιμή call_val του αντίστοιχου ορίσματος με την οποία γίνεται η κλήση μεταδίδεται στον επιλογέα μέσω της αντίστοιχης κατάστασης call_state. Η κατάσταση call_state είναι αυτή στην οποία συμβαίνει η συγκεκριμένη κλήση, η ίδια δηλαδή κατάσταση που συμμετέχει και στις καταστάσεις εισόδου της συνάρτησης, όπως δείχνει και η εικόνα 14. Το DFF εισόδου, που φαίνεται στην ίδια εικόνα, εισάγει μια καθυστέρηση ενός παλμού ρολογιού ώστε να πραγματοποιηθεί η εγγραφή της καλούμενης τιμής στον καταχωρητή της παραμέτρου. Έτσι, όταν ξεκινήσει η εκτέλεση του σώματος της συνάρτησης, η τιμή θα έχει καταχωρηθεί και θα είναι διαθέσιμη μέσω του σήματος εξόδου val του καταχωρητή. Στο εσωτερικό της συνάρτησης είναι δυνατό να ανατίθεται πάλι κάποια τιμή στην παράμετρο, μιας και η παράμετρος συμπεριφέρεται γενικά σαν τοπική μεταβλητή. Έτσι, σε κάθε σημείο που ο κώδικας της συνάρτησης αλλάζει την τιμή της, το αντίστοιχο ζεύγος local_val και local_state με την τιμή και την κατάσταση εγγραφής εισάγεται στον ίδιο επιλογέα.

Οι δείκτες. Το κύκλωμα των καθολικών μεταβλητών τύπου δείκτη. Το κύκλωμα των παραμέτρων κατ' αναφορά. Ο κολλώδης ανανεωτής.

Στη συνέχεια θα εξεταστούν οι *παράμετροι κατ' αναφορά* (parameters by reference), καθώς και οι καθολικές μεταβλητές τύπου δείκτη, που αμφότερες δηλώνονται ως δείκτες της C. Τα δεδομένα τύπου δείκτη δε διαθέτουν αποθηκευτικό χώρο μέσα στο κύκλωμα στο οποίο ορίζονται. Στην κλασική C, οι παράμετροι κατ' αναφορά μεταβιβάζουν ένα δείκτη προς κάποιο δεδομένο, δηλαδή μια διεύθυνση μνήμης μέσω της οποίας μπορεί η συνάρτηση να προσπελάσει και να μεταβάλει το δεδομένο. Η φιλοσοφία των δεικτών δεν είναι εύκολο να ακολουθηθεί στην περίπτωση της σύνθεσης σταθερών κυκλωμάτων, μιας και δε χρησιμοποιείται μνήμη αλλά σταθεροί καταχωρητές. Έτσι, η υποστήριξη των δεικτών στη νέα γλώσσα έχει παραμείνει μόνο για δύο περιπτώσεις: για τις ανάγκες του χειρισμού μιας παραμέτρου κατ' αναφορά, και για τις καθολικές μεταβλητές τύπου δείκτη, με τις τελευταίες να εισάγονται για λόγους επικοινωνίας του προγράμματος με το εξωτερικό περιβάλλον.

Στην πρώτη περίπτωση, η σημασιολογία της κλήσης κατ' αναφορά μεταφράζεται πλέον ως εξής: όταν μια συνάρτηση περιέχει ένα τέτοιο όρισμα, τότε κατά την κλήση της δε δίδεται μια τιμή αλλά μια μεταβλητή με την οποία θα συνδέεται η συνάρτηση όσο εκτελείται, ώστε να μπορεί να τη διαβάσει και να αλλάξει την τιμή της (ακριβώς όπως θα συνέβαινε με έναν δείκτη). Ο μηχανισμός αυτός βέβαια παρουσιάζει μια δυσκολία ως προς την υλοποίησή του, διότι πρέπει σε κάθε διαφορετική κλήση να συνδεθεί η συνάρτηση και με μια διαφορετική μεταβλητή. Αντίστοιχα στην περίπτωση ορισμού μιας καθολικής μεταβλητής τύπου δείκτη, δε δημιουργείται καταχωρητής για τη μεταβλητή, παρά ορίζονται κάποια σήματα με τα οποία το πρόγραμμα επικοινωνεί με το εξωτερικό περιβάλλον, για την εγγραφή και την ανάγνωση της μεταβλητής. Τυχόν εξωτερικό κύκλωμα που συνδέεται με το κύκλωμα του προγράμματος που παράγεται από το μεταγλωττιστή μπορεί να χρησιμοποιηθεί για να διαχειριστεί αυτά τα σήματα, και να διαδραματίσει για παράδειγμα το ρόλο μιας αποθηκευτικής μονάδας στην οποία αναφέρεται η καθολική μεταβλητή τύπου δείκτη. Σε κάθε περίπτωση, μια μεταβλητή τύπου δείκτη (είτε πρόκειται για παράμετρο κατ' αναφορά μιας συνάρτησης, είτε για καθολική μεταβλητή του συνολικού κυκλώματος) σχετίζεται κυκλωματικά με τρία βασικά σήματα τα οποία χρησιμοποιούνται για τις απαιτούμενες λειτουργίες. Τα σήματα αυτά φαίνονται στην επόμενη εικόνα:

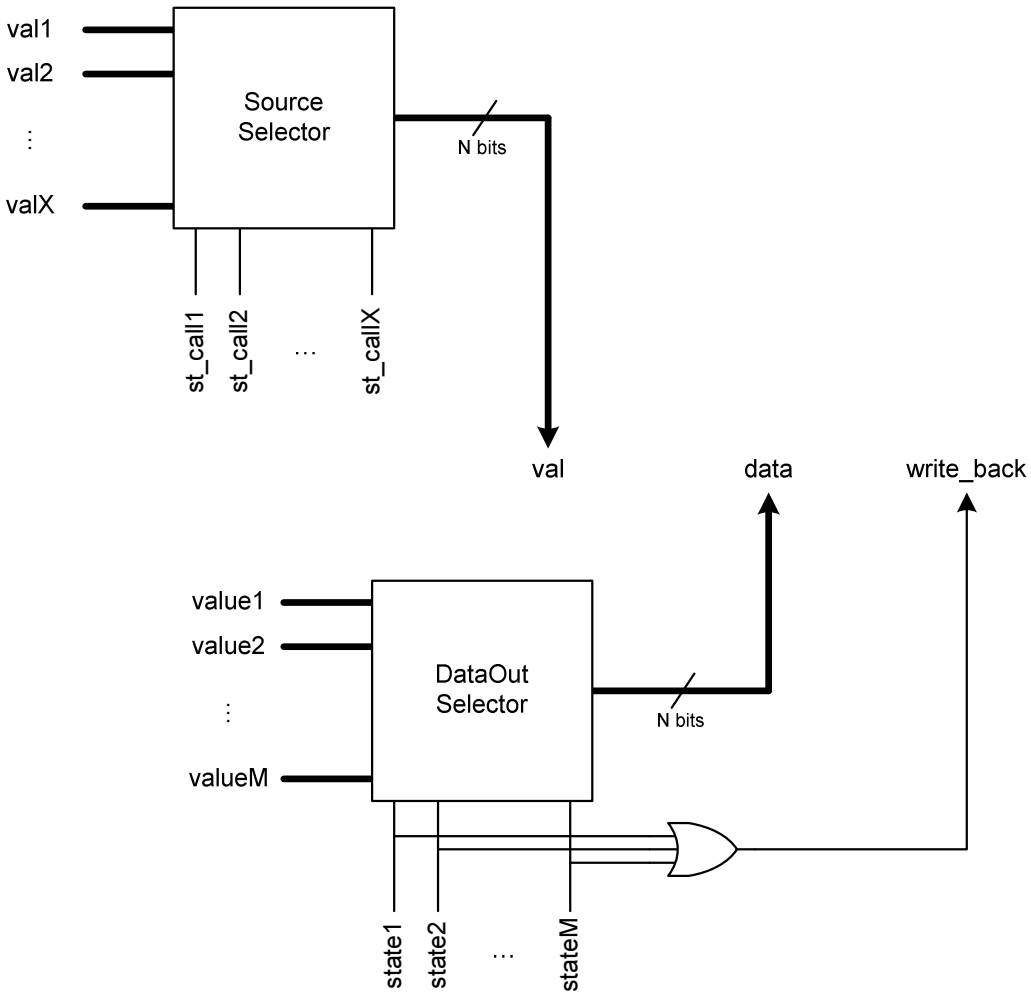


Εικόνα 18 – Τα σήματα που σχετίζονται με μια μεταβλητή τύπου δείκτη

Στην περίπτωση μιας παραμέτρου κατ' αναφορά μιας συνάρτησης, η συνάρτηση λαμβάνει ένα εισερχόμενο σήμα 'val' με την τρέχουσα τιμή της εκάστοτε μεταβλητής που έχει περαστεί ως παράμετρος. Το σήμα αυτό οφείλει να αντικατοπτρίζει άμεσα οποιεσδήποτε αλλαγές συμβούν είτε από την ίδια τη συνάρτηση είτε από άλλα κομμάτια κώδικα που τρέχουν παράλληλα και πειράζουν την ίδια μεταβλητή, διαφορετικά καταστρατηγείται η έννοια της κλήσης κατ' αναφορά. Άρα, το σήμα 'val' πρέπει να είναι το σήμα εξόδου του καταχωρητή της εκάστοτε μεταβλητής (αλλά όπως θα αναφερθεί παρακάτω μπορεί να προέρχεται και από το σήμα val της παραμέτρου κατ' αναφορά μιας άλλης συνάρτησης). Επίσης, η συνάρτηση εξάγει τα σήματα 'data' και 'write_back'. Αυτά υφίστανται για την εγγραφή μιας τιμής πίσω στη μεταβλητή: η τιμή προς εγγραφή εξάγεται στο σήμα 'data', ενώ το σήμα 'write_back' δηλώνει την κατάσταση εκείνη στην οποία θα συμβεί η εγγραφή, δηλαδή ενεργοποιείται κάθε φορά που η συνάρτηση ζητά να γράψει κάποια τιμή πίσω στη μεταβλητή.

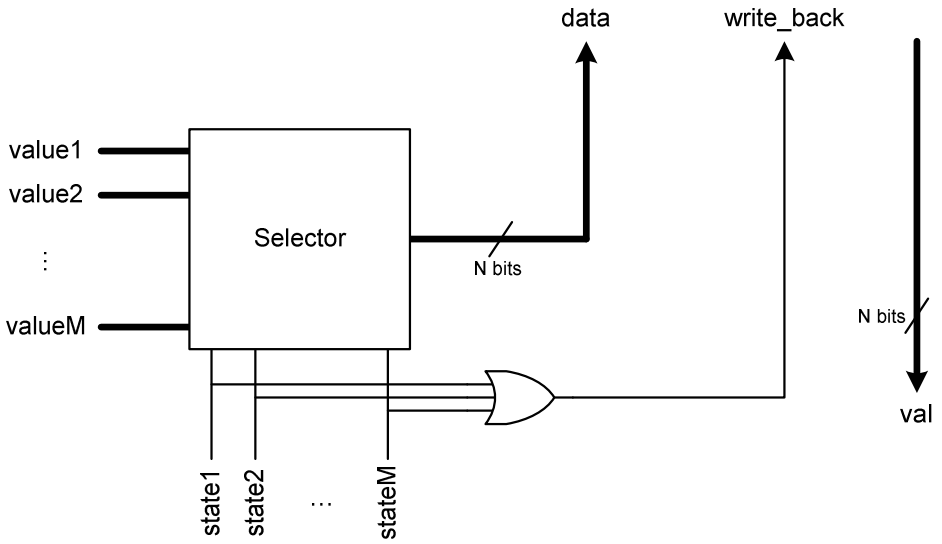
Στην περίπτωση μιας καθολικής μεταβλητής τύπου δείκτη, το σήμα val αποτελεί ένα εξωτερικό σήμα το οποίο λαμβάνεται από κάποιο άλλο κύκλωμα, εκτός του προγράμματος. Η προέλευσή του είναι άγνωστη και δεν αφορά το μεταγλωττιστή. Το σήμα αυτό περιέχει ανά πάσα στιγμή την τρέχουσα τιμή της μεταβλητής, δηλαδή όποτε μέσα στο πρόγραμμα χρησιμοποιείται η τιμή του δείκτη, αρκεί να γίνει χρήση της τιμής val. Το εξωτερικά συνδεόμενο κύκλωμα οφείλει να τροφοδοτεί συνεχώς αυτό το σήμα με τη σωστή τιμή. Όμοια, όποτε μέσα στο πρόγραμμα εμφανίζεται μια ανάθεση τιμής στην καθολική μεταβλητή τύπου δείκτη, η αντίστοιχη τιμή προς εγγραφή εξάγεται στο σήμα 'data', και το σήμα 'write_back' ενεργοποιείται για να δηλώσει ότι τώρα απαιτείται εγγραφή. Το εξωτερικό κύκλωμα είναι υπεύθυνο να παρακολουθεί τα σήματα αυτά και να επιτελεί ό,τι λειτουργίες είναι αναγκαίες για να χρησιμοποιήσει τις νέες τιμές και να ενημερώσει ύστερα το σήμα val. Γενικότερα η καθολική μεταβλητή τύπου δείκτη χρησιμοποιείται απλά για τη διασύνδεση του προγράμματος με εξωτερικά κυκλώματα, δηλαδή προσφέρει ένα φορμαλισμό για την πραγματοποίηση επικοινωνίας με το εξωτερικό περιβάλλον, και η πολυπλοκότητα της σταματά ακριβώς εκεί, μιας και το εξωτερικό κύκλωμα δεν αφορά το μεταγλωττιστή. Αντίθετα, μια παράμετρος κατ' αναφορά χρησιμοποιείται για να μπορούν να περαστούν σε μία συνάρτηση μεταβλητές του προγράμματος ως παράμετροι και να μπορεί να αλλάξει η τιμή τους (δηλαδή για εσωτερική διακίνηση δεδομένων), και απαιτεί επιπρόσθετα σύνθετα κυκλώματα.

Οι εικόνες 19 και 20 δείχνουν τα κυκλωμάτων που χρησιμοποιούνται για τα σήματα val, data και write_back, για τις περιπτώσεις της παραμέτρου κατ' αναφορά και της καθολικής μεταβλητής τύπου δείκτη αντίστοιχα.



Εικόνα 19 – Το κύκλωμα που αντιστοιχεί σε μία παράμετρο κατ' αναφορά

Το κύκλωμα παραγωγής των σημάτων data και write_back είναι αρκετά απλό. Για το σήμα data, χρησιμοποιείται ένας επιλογέας στον οποίο μπαίνουν τα ζεύγη τιμών-καταστάσεων τα οποία προορίζονται για εγγραφή, σαν να ήταν δηλαδή ο δείκτης μια κανονική μεταβλητή. Ώστε λοιπόν, όλα τα σημεία του κώδικα τα οποία αναθέτουν τιμή στην παράμετρο κατ' αναφορά ή στην καθολική μεταβλητή τύπου δείκτη εισάγουν απλά την τιμή και την αντίστοιχη κατάσταση στον επιλογέα αυτό, ο οποίος στην πρώτη περίπτωση έχει ονομαστεί *επιλογέας δεδομένων εξόδου* (DataOut Selector) για να ξεχωρίζει από ένα δεύτερο επιλογέα που υπάρχει στο σχήμα της παραμέτρου. Το σήμα εξόδου write_back προκύπτει από το λογικό OR των καταστάσεων του επιλογέα, ώστε να ενεργοποιείται όταν είναι ενεργή η αντίστοιχη κατάσταση εγγραφής μιας τιμής. Για την καθολική μεταβλητή τύπου δείκτη, αυτά είναι και τα μόνα κυκλώματα που απαιτούνται, αφού η τιμή val προέρχεται όπως αναφέρθηκε από το εξωτερικό περιβάλλον χωρίς άλλους χειρισμούς. Έτσι τελικά αυτό που διαχωρίζει την καθολική μεταβλητή τύπου δείκτη (εικόνα 20) από μια κανονική μεταβλητή (εικόνα 12) είναι ότι αντί να μεσολαβεί ένας καταχωρητής, μεσολαβεί ένα άγνωστο εξωτερικό κύκλωμα που είναι υπεύθυνο για την αποθήκευση της μεταβλητής.



Εικόνα 20 – Το κύκλωμα που αντιστοιχεί σε μία καθολική μεταβλητή τύπου δείκτη

Ωστόσο, στην περίπτωση μιας παραμέτρου κατ' αναφορά (εικόνα 19), εμπλέκονται περισσότερα κυκλώματα, διότι η τιμή της παραμέτρου προέρχεται από κάποια άλλη μεταβλητή που μεταβιβάζεται κατά την κλήση της συνάρτησης. Τα κυκλώματα αυτά εξετάζονται στη συνέχεια. Έτσι, για το σήμα εισόδου val υπάρχει εξίσου ένας επιλογέας, που έχει ονομαστεί *επιλογέας πηγής* (Source Selector). Χρησιμεύει στην επιλογή της πηγής από την οποία θα λαμβάνεται η τιμή της παραμέτρου κατ' αναφορά όταν είναι να χρησιμοποιηθεί από τη συνάρτηση, δηλαδή επιλέγει από όλες τις πηγές με τις οποίες καλείται η συγκεκριμένη παράμετρος. Ακριβέστερα, αν σε όλο το πρόγραμμα εμφανίζονται X διαφορετικές κλήσεις προς τη συνάρτηση που περιέχει την παράμετρο κατ' αναφορά, τότε τα σήματα val1 ως valX στην είσοδο δεδομένων του επιλογέα αποτελούν αντίστοιχες εξόδους καταχωρητών (για τις κλήσεις που πέρασαν μια μεταβλητή στη θέση της παραμέτρου κατ' αναφορά) ή αντίστοιχες εξόδους άλλων επιλογέων πηγής (για τις κλήσεις που πέρασαν μια παράμετρο κατ' αναφορά από μια άλλη συνάρτηση, δηλαδή την περίπτωση εκείνη όπου μια συνάρτηση που λαμβάνει μια παράμετρο κατ' αναφορά την περνάει με τη σειρά της σε μια άλλη συνάρτηση σε μετέπειτα κλήση). Τα σήματα st_call1 ως st_callX αποτελούν κάποια σήματα επιλογής, ώστε ανά πάσα στιγμή να επιλέγεται η σωστή πηγή για την τροφοδοσία του σήματος val. Προφανώς πρέπει κάθε φορά να είναι ενεργοποιημένο εκείνο που αντικατοπτρίζει την αντίστοιχη κλήση που έχει γίνει προς τη συνάρτηση, δηλαδή κάθε σήμα st_call διακρίνει σε ποια από τις X διαφορετικές κλήσεις βρίσκεται το κύκλωμα. Αν και είναι γνωστό ότι κάθε κλήση προς τη συνάρτηση εισάγει την αντίστοιχη κατάσταση κλήσης 'call_state' στη λίστα των καταστάσεων εισόδου της συνάρτησης (που φαίνεται στην εικόνα 14), η κατάσταση αυτή δεν είναι αρκετή για να διακρίνει την κλήση διότι είναι ενεργή μόνο για έναν παλμό ρολογιού, αφού μετά διαδίδεται στο σώμα της συνάρτησης μέσω του σκελετού ελέγχου. Πράγματι, το απαιτούμενο σήμα st_call θα πρέπει να παραμένει ενεργό καθ' όλη τη διάρκεια της κλήσης. Για την παραγωγή των τιμών st_call από τις αντίστοιχες καταστάσεις κλήσεων, χρησιμοποιείται ένα ειδικό ακολουθιακό κύκλωμα, το οποίο συντίθεται μόνο για τις συναρτήσεις που περιέχουν παραμέτρους κατ' αναφορά. Το κύκλωμα αυτό, που βαφτίστηκε με τη μάλλον αστεία ονομασία *κολλώδης ανανεωτής* (sticky updater), μετατρέπει τις καταστάσεις call_state στα αντίστοιχα *κολλώδη* (sticky) σήματα st_call. Η αναπαράστασή του φαίνεται στην επόμενη εικόνα.

Εικόνα 21 – Ο κολλώδης ανανεωτής

Η ακριβής λειτουργία του κολλώδη ανανεωτή είναι η εξής: Με την αρχικοποίηση του κυκλώματος, τα σήματα στην έξοδο του ανανεωτή είναι όλα μηδενικά, καθώς δεν έχει πραγματοποιηθεί καμία κλήση. Μόλις ενεργοποιηθεί μια είσοδος call_state, ενεργοποιείται η αντίστοιχη έξοδος st_call και οι υπόλοιπες έξοδοι μηδενίζονται. Η κατάσταση αυτή διατηρείται συνεχώς, ακόμα και όταν απενεργοποιηθεί η είσοδος (η οποία, μιας και προέρχεται από την κατάσταση κλήσης μιας συνάρτησης, θα είναι ενεργή μόνο για έναν κύκλο ρολογιού), δηλαδή το κύκλωμα θυμάται για τους επόμενους κύκλους ρολογιού από που προήλθε η τελευταία κλήση. Όταν ξανακληθεί η συνάρτηση και ενεργοποιηθεί κάποια άλλη είσοδος call_state, τότε θα ενεργοποιήσει την νέα αντίστοιχη έξοδο st_call και θα μηδενίσει εκ νέου τις υπόλοιπες εξόδους, έτσι ώστε κάθε φορά μόνο μία έξοδος του κολλώδη ανανεωτή να είναι ενεργή. Στην προβληματική περίπτωση όπου εμφανιστούν ταυτόχρονα δύο ή περισσότερες ενεργές είσοδοι, δηλαδή η συνάρτηση καλείται ταυτόχρονα από δύο ή περισσότερα διαφορετικά σημεία, επιλέγεται ακριβώς μία κατάσταση για να ακολουθηθεί στην έξοδο, με κάποιο αυθαίρετο κανόνα προτεραιότητας. Αν και ο κολλώδης ανανεωτής είναι ένα αμιγώς ακολουθιακό κύκλωμα, η σύνθεση απαιτεί να δίνει το σωστό αποτέλεσμα άμεσα, χωρίς να απαιτείται να παρέλθει ένας παλμός ρολογιού, ώστε να μπορεί να επιλύεται η πηγή μιας παραμέτρου κατ' αναφορά δίχως καθυστέρηση. Έτσι, στην υλοποίηση του εσωτερικού ενός κολλώδη ανανεωτή προστίθεται και ένα συνδυαστικό κύκλωμα που προωθεί το σωστό αποτέλεσμα στην έξοδο. Μια υλοποίηση του κολλώδη ανανεωτή σε Verilog δίνεται σε επόμενη ενότητα.

Στην απλή ιδεατή περίπτωση όπου η συνάρτηση που περιέχει παράμετρο κατ' αναφορά καλείται μόνο μία συνολικά φορά από κάποιο άλλο σημείο του προγράμματος, με παράμετρο μια μεταβλητή, το σήμα val θα προερχόταν από τον καταχωρητή της μεταβλητής, ενώ τα σήματα data και write_back θα δημιουργούσαν ένα ζεύγος τιμής-κατάστασης στον επιλογέα της μεταβλητής. Στην πράξη όμως μια συνάρτηση ενδεχομένως να καλείται από πολλά διαφορετικά σημεία με διαφορετικές παραμέτρους, ώστε κάθε φορά τα σήματα val, data και write_back να πρέπει να δρομολογούνται με διαφορετικό τρόπο. Για το σήμα val αναφέρθηκε ήδη ότι η επιλογή της πηγής γίνεται με χρήση του επιλογέα πηγής, ο οποίος βρίσκει τη σωστή προέλευση χάρη στα αντίστοιχα σήματα st_call που διακρίνουν την τρέχουσα κλήση. Απομένουν όμως τα σήματα data και write_back που πρέπει εξίσου να βρουν το δρόμο τους προς τη σωστή κατεύθυνση. Για κάθε κλήση η οποία περνά ως παράμετρο κατ' αναφορά στη συνάρτηση μια μεταβλητή, τα σήματα data και write_back θα μπορούσαν να δημιουργήσουν ένα ζεύγος τιμής-κατάστασης για τον επιλογέα που τροφοδοτεί τον καταχωρητή της μεταβλητής. Δεν είναι όμως σωστό να εισαχθούν αυτούσια τα σήματα data και write_back σε όλους τους επιλογείς των μεταβλητών από τις οποίες προέρχεται η παράμετρος, διότι έτσι σε κάθε εγγραφή θα μεταβάλλονταν όλες οι μεταβλητές, ενώ το ορθό είναι να μεταβληθεί μόνο εκείνη η μεταβλητή με την οποία έγινε η συγκεκριμένη κλήση. Για τη διάκριση αυτή, η λύση είναι το σήμα write_back να υποστεί λογικό

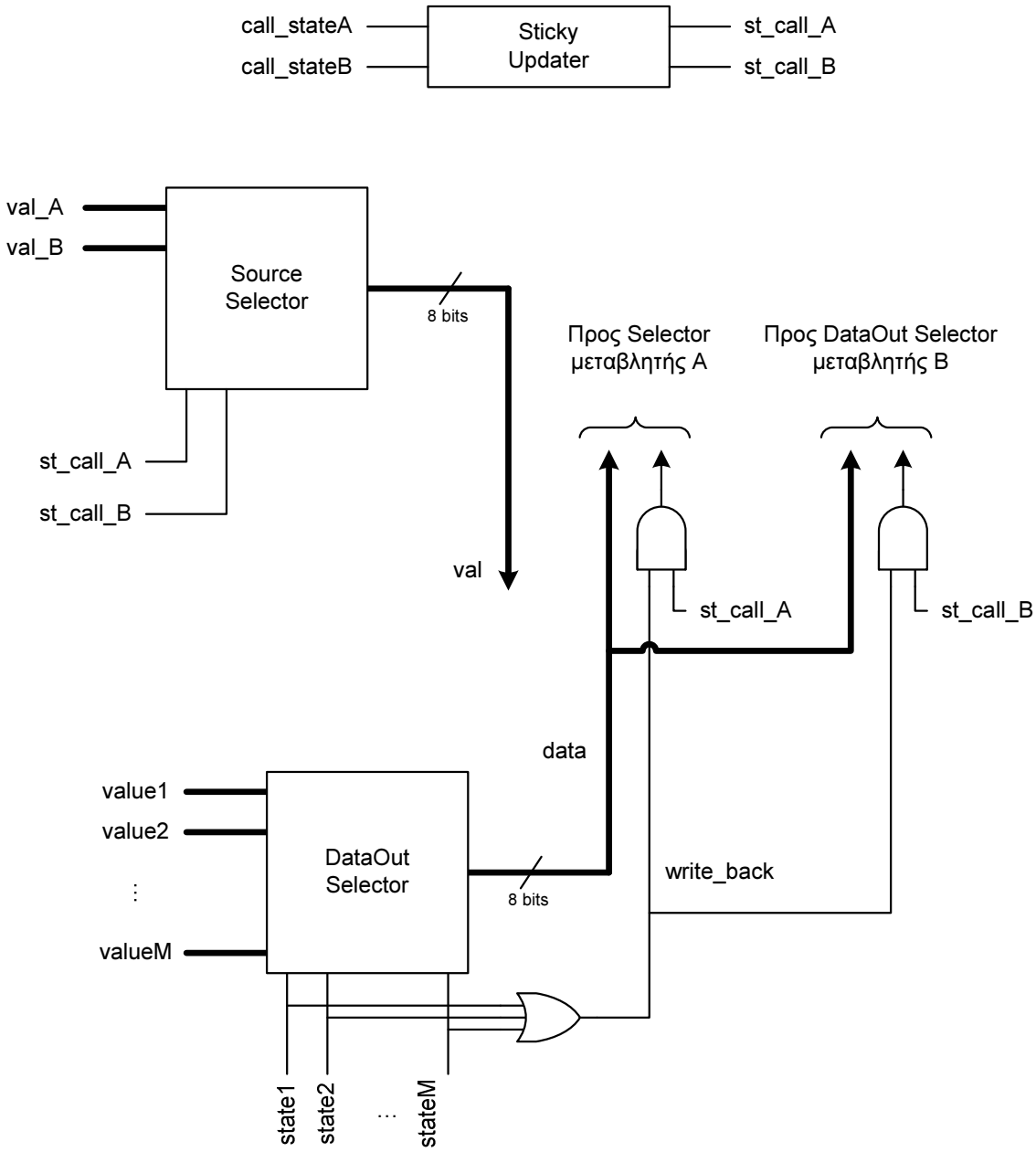
AND με καθένα από τα σήματα st_call, ώστε να παραχθούν X διαφορετικά καινούρια σήματα εγγραφής. Έτσι, κάθε φορά που εγγράφεται μια τιμή πίσω στην παράμετρο, το σήμα write_back θα διαδίδεται μόνο σε ένα σημείο και τα υπόλοιπα θα αποκόπτονται, ώστε τελικά να φθάνει η ενεργή κατάσταση εγγραφής μόνο στη μεταβλητή που χρησιμοποιήθηκε ως παράμετρος στη συγκεκριμένη κλήση. Βέβαια καθώς έχει ήδη αναφερθεί, ως παράμετρος κατ' αναφορά μπορεί να περαστεί σε μία κλήση και μια άλλη παράμετρος κατ' αναφορά μιας άλλης συνάρτησης. Σε αυτή την περίπτωση, το σήμα data και το τροποποιημένο σήμα write_back εισάγονται στον επιλογέα δεδομένων εξόδου της άλλης παραμέτρου.

Για να γίνει πιο εύκολα κατανοητός ο μηχανισμός της κλήσης κατ' αναφορά, θα δοθεί ένα παράδειγμα που περιλαμβάνει δύο περιπτώσεις κλήσης. Έστω λοιπόν ότι σε ένα πρόγραμμα περιλαμβάνονται δύο συναρτήσεις funcA και funcB, οι οποίες καλούν μια τρίτη συνάρτηση testfunc, που λαμβάνει μια παράμετρο κατ' αναφορά. Η συνάρτηση funcA καλεί την testfunc με παράμετρο μια μεταβλητή a, ενώ η συνάρτηση funcB καλεί την testfunc με παράμετρο μια δική της παράμετρο κατ' αναφορά b. Ο κώδικας C που αντιστοιχεί σε αυτό το μοντέλο φαίνεται παρακάτω:

```
char a;

testfunc(bit *Q)
{
  /* ... Εντολές ... */
}

funcA()
{
  /* ... */

  testfunc(&a);

  /* ... */
}

funcB(char *b)
{
  /* ... */

  testfunc(b);

  /* ... */
}
```

Στο πρόγραμμα χρησιμοποιούνται στοιχεία που δεν έχουν περιγραφεί ακόμη, όπως ο τελεστής διεύθυνσης (&) ο οποίος χρησιμοποιείται για τη μετατροπής μιας άλφατιμής (l-value) σε δείκτη, ώστε να μπορεί να μεταβιβαστεί ως παράμετρος κατ' αναφορά. Η προσοχή θα πρέπει να εστιαστεί στο κύκλωμα της παραμέτρου κατ' αναφορά Q της συνάρτησης testfunc, για το οποίο έχει κατασκευαστεί αυτό το παράδειγμα. Η μεταβλητή a είναι μια καθολική μεταβλητή και άρα αποτελείται από ένα κύκλωμα σαν αυτό της εικόνας 12, του οποίου η έξοδος του καταχωρητή ας ονομαστεί val_A. Η μεταβλητή b (που καταχρηστικά ονομάζεται μεταβλητή) είναι μια παράμετρος κατ' αναφορά της συνάρτησης funcB και επομένως αποτελείται από ένα κύκλωμα σαν αυτό της εικόνας 19. Ας ονομαστεί val_B η έξοδος του επιλογέα πηγής του κυκλώματος της μεταβλητής b, δηλαδή η τρέχουσα τιμή της παραμέτρου b. Η παράμετρος Q της

συνάρτησης testfunc απεικονίζεται εξίσου σε ένα κύκλωμα σαν αυτό της εικόνας 19, του οποίου η ειδικότερη μορφή φαίνεται παρακάτω:



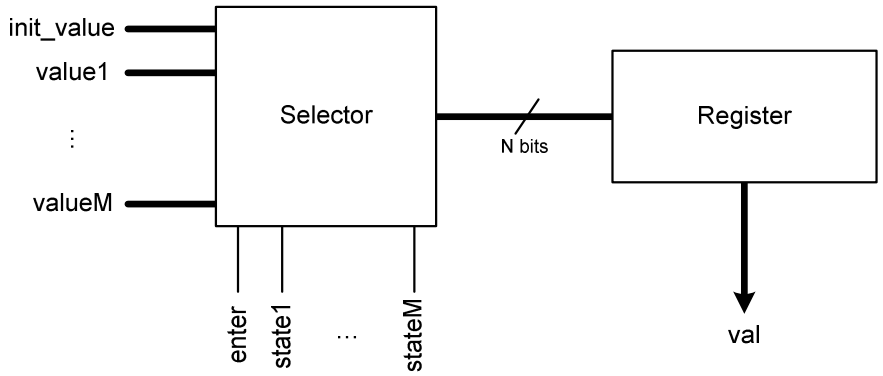
Εικόνα 22 – Τα κυκλώματα που αντιστοιχούν στην παράμετρο Q

Το κύκλωμα περιλαμβάνει αρχικά τον κολλώδη ανανεωτή (που αποτελεί γενικά μέρος της συνάρτησης testfunc και όχι της παραμέτρου Q, δηλαδή αντιστοιχεί ακριβώς ένας για κάθε συνάρτηση που περιέχει παραμέτρους κατ' αναφορά), για την παραγωγή των σημάτων st_call. Έστω call_state_A και call_state_B οι δύο καταστάσεις στις οποίες εμφανίζονται οι κλήσεις προς τη συνάρτηση testfunc. Ο κολλώδης ανανεωτής παράγει τα αντίστοιχα σήματα st_call_A και st_call_B που δείχνουν σε ποια από τις δύο κλήσεις βρίσκεται το κύκλωμα. Για καθαρότερη παρουσίαση, τα σήματα αυτά χρησιμοποιούνται στη συνέχεια χωρίς να φαίνεται η σύνδεσή τους με το κύκλωμα του ανανεωτή, αλλά αυτή πρέπει να εννοείται. Ο επιλογέας πηγής της παραμέτρου Q εφοδιάζεται με τις τιμές val_A και val_B των μεταβλητών a και b αντίστοιχα, και ε-

πιλέγει τη σωστή από τις δύο στην έξοδο του val. Ο επιλογέας δεδομένων εξόδου έχει όλα τα ζεύγη τιμών-καταστάσεων που εγγράφονται στην παράμετρο Q, π.χ. από εντολές ανάθεσης που βρίσκονται μέσα στο σώμα της testfunc, κ.α. Τα σήματα data και write_back δρομολογούνται αφενός προς τη μεταβλητή a, και αφετέρου προς τη μεταβλητή b. Για την πρώτη μεταβλητή, το σήμα write_back φιλτράρεται από το σήμα st_call_A ώστε να πραγματοποιείται η εγγραφή της μεταβλητής a μόνο εφόσον η κλήση έχει γίνει από τη συνάρτηση funcA, και μαζί με το σήμα data εισάγονται στον επιλογέα της καθολικής μεταβλητής a. Με όμοιο τρόπο παράγονται τα αντίστοιχα σήματα για τη μεταβλητή b, και εισάγονται στον επιλογέα δεδομένων εξόδου της παραμέτρου b, αφού πρόκειται για παράμετρο κατ' αναφορά και όχι καθολική ή τοπική μεταβλητή. Ας αναφερθεί επίσης ότι εάν επιπλέον η συνάρτηση testfunc καλούσε τη funcB με όρισμα Q στη θέση της παραμέτρου b, τότε ο επιλογέας δεδομένων εξόδου της παραμέτρου Q που φαίνεται στο παραπάνω σχήμα θα περιλάμβανε και ένα ζεύγος τιμής-κατάστασης που θα προερχόταν από τα αντίστοιχα σήματα εξόδου του κύκλωμα της παραμέτρου b (το οποίο θα ήταν ένα κύκλωμα αντίστοιχο με αυτό τις παραπάνω εικόνας).

Το κύκλωμα των τοπικών μεταβλητών.

Οι τοπικές μεταβλητές, που δηλώνονται στην αρχή του σώματος της συνάρτησης, αντιστοιχούν σε κυκλώματα ανάλογα με αυτά των καθολικών μεταβλητών, δηλαδή το γνωστό κύκλωμα επιλογέα-καταχωρητή της εικόνας 12. Κάθε εντολή της συνάρτησης που με τον ένα ή άλλο τρόπο αλλάζει τα δεδομένα μιας τοπικής μεταβλητής, εισάγει το αντίστοιχο ζεύγος τιμής-κατάστασης στον επιλογέα. Ωστόσο, όπως αναφέρθηκε σε προηγούμενη παράγραφο, υπάρχει μια μικρή διαφοροποίηση για τις τοπικές μεταβλητές που αρχικοποιούνται στη δήλωσή τους. Μία τοπική μεταβλητή που δηλώνεται με αρχική τιμή οφείλει να λαμβάνει την τιμή της αυτή κάθε φορά που ξεκινά να εκτελείται η συνάρτηση, δηλαδή σε κάθε κλήση της συνάρτησης (και όχι άπαξ κατά την αρχικοποίηση των κυκλωμάτων, όπως συμβαίνει με τις καθολικές μεταβλητές). Έτσι, στον αντίστοιχο επιλογέα που δημιουργείται για την τοπική μεταβλητή εισάγεται ένα επιπλέον ζεύγος τιμής-κατάστασης, όπως δείχνει το ακόλουθο σχήμα.



Εικόνα 23 – Το κύκλωμα που αντιστοιχεί σε μία τοπική μεταβλητή που δηλώνεται με αρχική τιμή

Η τιμή του ζεύγους είναι η αρχική τιμή init_value με την οποία δηλώνεται η μεταβλητή, ενώ η κατάσταση δεν είναι άλλη από την κατάσταση enter του κυκλώματος ελέγχου της συνάρτησης (εικόνα 14). Το εισαγωγικό DFF του κυκλώματος ελέγχου φροντίζει ώστε η αρχική τιμή να προλάβει να εγγραφεί και μετά να ξεκινήσει η εκτέλεση του σώματος της συνάρτησης.

Ανακεφαλαιώνοντας, τα διάφορα μέρη από τα οποία αποτελείται το συνολικό κύκλωμα μιας συνάρτησης είναι τα εξής:

- Το κύκλωμα ελέγχου της συνάρτησης (εικόνα 14).
- Το κύκλωμα της τιμής επιστροφής (εικόνα 16).
- Το κύκλωμα καθεμίας παραμέτρου κατ' αξία (εικόνα 17).
- Το κύκλωμα καθεμίας παραμέτρου κατ' αναφορά (εικόνα 19).
- Το κύκλωμα του κολλώδη ανανεωτή (εικόνα 21).
- Το κύκλωμα καθεμίας τοπικής μεταβλητής (εικόνα 23).

Το κύκλωμα ελέγχου της συνάρτησης περιλαμβάνει και τα κυκλώματα των εντολών του σώματος της συνάρτησης, που είναι τα μόνα που δεν έχουν αναλυθεί μέχρι στιγμής. Η επόμενη ενότητα ασχολείται με τη σύνταξη και την κυκλωματική αναπαράσταση των εντολών, των παραστάσεων, και των άλλων στοιχείων της γλώσσας που δεν έχουν καλυφθεί.

Η κυκλωματική σύνθεση του σώματος εντολών μιας συνάρτησης αποτελεί το κυριότερο κομμάτι της σύνθεσης. Οι εντολές, καθώς απαντώνται στον πηγαίο κώδικα, μεταφράζονται σε αντίστοιχα κυκλώματα, η γενική μορφή των οποίων παρουσιάστηκε αρχύτερα στην εικόνα 11. Στην ενότητα αυτή παρουσιάζεται αναλυτικά το κύκλωμα για καθεμία εντολή της C. Πολλές εντολές της C συντάσσονται με *παραστάσεις* (expressions). Οι παραστάσεις, που γενικά θα μπορούσαν να χαρακτηριστούν ως η εφαρμογή τελεστών της C πάνω σε δεδομένα (αν και στην πράξη είναι πιο πολύπλοκες), εξετάζονται εξίσου με τις εντολές, τόσο ως προς τη σύνταξη όσο και ως προς την κυκλωματική αναπαράσταση.

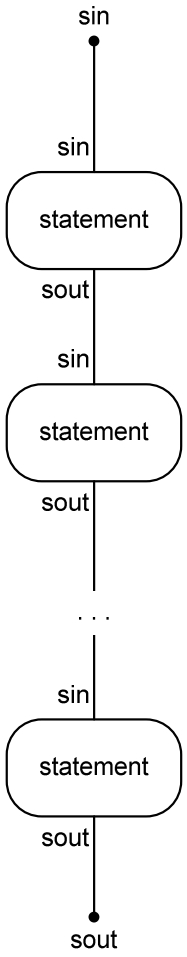
Από άποψη σύνθεσης, το σώμα εντολών μιας συνάρτησης δεν είναι παρά ένα σειριακό ή παράλληλο μπλοκ εντολών. Τα μπλοκ εντολών, που στη C γράφονται με *άγκιστρα* (curly brackets), μπορούν να εισαχθούν στη θέση αντί οποιασδήποτε εντολής, μιας και θεωρούνται κι αυτά εντολές. Αν και στην αρχή του σώματος μιας συνάρτησης μπορούν να δηλωθούν τοπικές μεταβλητές, αυτή η δυνατότητα δεν διατίθεται για οποιοδήποτε μπλοκ εντολών, ώστε να μειωθεί η πολυπλοκότητα του μεταγλωττιστή. Παρακάτω εξετάζονται τα δύο είδη μπλοκ εντολών.

Το σειριακό μπλοκ εντολών.

Το σειριακό μπλοκ εντολών συντάσσεται ως εξής:

```
{
  <statement>*
}
```

Το σειριακό μπλοκ εντολών χρησιμοποιείται για τον προσδιορισμό μιας σειράς εντολών που θα εκτελεστούν ακολουθιακά, ακριβώς με τη σειρά που είναι γραμμένες. Πρόκειται για το κλασικό μπλοκ εντολών της C, του οποίου η σημασιολογία έχει διατηρηθεί. Η κυκλωματική αναπαράσταση φαίνεται στην εικόνα 24. Το μπλοκ αντιστοιχεί στη σειριακή σύνδεση των καταστάσεων ελέγχουν των εντολών ώστε να προκύψει μια εντολή με κατάσταση εισόδου αυτή της πρώτης εντολής, και κατάσταση εξόδου αυτή της τελευταίας. Μόλις μια εντολή του μπλοκ ολοκληρώσει την εκτέλεσή της ενεργοποιεί το σήμα εξόδου της sout, ξεκινώντας ταυτόχρονα την επόμενη κατά σειρά εντολή, μιας και ενεργοποιείται το αντίστοιχο σήμα sin. Όπως φαίνεται και στην εικόνα, οι εντολές συνδέονται άμεσα χωρίς την εισαγωγή DFF καθυστέρησης. Δεν υπάρχει λόγος να εισαχθούν τέτοια DFF, καθώς όπου απαιτούνται τέτοιες καθυστερήσεις θα έχουν εισαχθεί από τις ίδιες τις εντολές στα κυκλώματά τους.



Εικόνα 24 – Το κύκλωμα του σειριακού μπλοκ εντολών

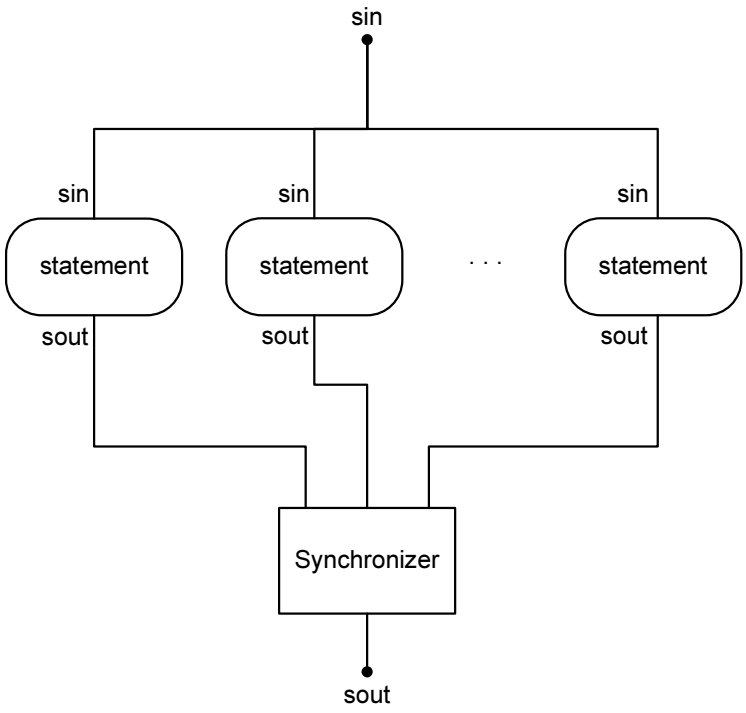
Το παράλληλο μπλοκ εντολών (par). Ο συγχρονιστής.

Το παράλληλο μπλοκ εντολών συντάσσεται ως εξής:

```
par {
  <statement>*
}
```

Το παράλληλο μπλοκ εντολών χρησιμοποιείται για τον προσδιορισμό μιας ομάδας εντολών που θα εκτελεστούν παράλληλα. Αποτελεί μια επέκταση της σύνταξης της C, με την τοποθέτηση της λέξης-κλειδί ‘par’ πριν από το μπλοκ, ώστε να προσδίδεται παράλληλη σημασιολογία. Η παράλληλη εκτέλεση σημαίνει ότι οι εντολές θα ξεκινήσουν να εκτελούνται ταυτόχρονα, και έτσι δεν έχει σημασία η σειρά με την οποία είναι γραμμένες μέσα στο μπλοκ. Ο προγραμματιστής φέρει την ευθύνη ώστε οι εντολές που εμφανίζονται μέσα σε ένα παράλληλο μπλοκ να μπορούν όντως να εκτελεστούν ταυτόχρονα, αν και προβληματικές περιπτώσεις όπως π.χ. η προσπάθεια ανάθεσης δύο διαφορετικών τιμών στην ίδια μεταβλητή στον ίδιο παλμό ρολογιού εί-

ναι δυνατό να αντιμετωπιστούν χάρη στις υλοποιήσεις συγκεκριμένων κυκλωματικών κομματιών (όπως ο επιλογέας) που εφαρμόζουν κάποιους κανόνες προτεραιότητας.



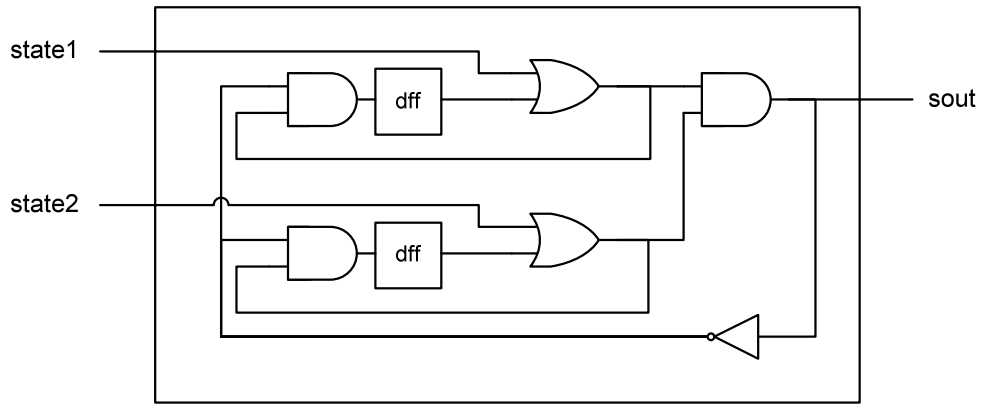
Εικόνα 25 – Το κύκλωμα του παράλληλου μπλοκ εντολών

Το κύκλωμα στο οποίο αντιστοιχεί ένα παράλληλο μπλοκ εντολών φαίνεται στην εικόνα 25. Όπως θα περίμενε κανείς, αρκεί η μετάδοση της κατάστασης εισόδου του παράλληλου μπλοκ σε όλες τις περιεχόμενες εντολές, ώστε αυτές να ξεκινήσουν ταυτόχρονα. Η κατάσταση εξόδου του παράλληλου μπλοκ εντολών είναι όμως ένα ζήτημα προς εξέταση: θα πρέπει να ενεργοποιηθεί μόλις τελειώσουν όλες οι εντολές την εκτέλεσή τους, χωρίς επιπλέον καθυστέρηση. Επειδή στη γενική περίπτωση δεν αναμένεται να τελειώσουν όλες οι εντολές στον ίδιο κύκλο ρολογιού, απαιτείται η εισαγωγή ενός ειδικού κυκλώματος για την παραγωγή της κατάστασης εξόδου. Το κύκλωμα αυτό είναι ο συγχρονιστής.

Ο *συγχρονιστής* (Synchronizer) αποτελεί ένα σημαντικό κομμάτι της σύνθεσης διότι χρησιμοποιείται κάθε φορά που πρέπει να συγχρονιστούν κάποιες καταστάσεις. Είναι ένα ακολουθιακό κύκλωμα το οποίο λαμβάνει δύο ή περισσότερες καταστάσεις ως είσοδο, και παράγει μια κατάσταση εξόδου. Η λειτουργία του είναι η εξής: Στο εσωτερικό του συγχρονιστή υπάρχουν τόσα flip-flops όσες και οι είσοδοί του. Αρχικά αυτά είναι μηδενισμένα, και η έξοδος του συγχρονιστή είναι ανενεργή. Μόλις ενεργοποιηθούν κάποιες από τις καταστάσεις στην είσοδο του συγχρονιστή, σημειώνονται στο εσωτερικό του κύκλωμα θέτοντας τα αντίστοιχα flip-flop. Τα flip-flop του συγχρονιστή διατηρούν την τιμή τους, ώστε αυτός να θυμάται ποιες είσοδοι έχουν μέχρι στιγμής ενεργοποιηθεί. Οι είσοδοι ενός συγχρονιστή προέρχονται συνήθως από τις καταστάσεις εξόδου κάποιων εντολών, και επομένως μια είσοδος θα είναι ενεργή μόνο για έναν παλμό ρολογιού. Έτσι, αυτό που καταφέρνει ο συγχρονιστής είναι να σημειώνει στο εσωτερικό του ποιες εντολές έχουν μέχρι στιγμής τελειώσει, δηλαδή ποια σήματα εισόδου έχουν μέχρι στιγμής υπάρξει ενεργά σε κάποιο παλμό ρολογιού. Η διαδικασία αυτή συνεχίζεται, μέχρι τη στιγμή που ενεργοποιούνται και οι εναπομεί-

νασες είσοδοι του συγχρονιστή—τότε το κύκλωμα ενεργοποιεί την κατάσταση εξόδου για ένα παλμό ρολογιού ενώ ταυτόχρονα μηδενίζεται η εσωτερική του κατάσταση ώστε να είναι έτοιμο να επαναλάβει το συγχρονισμό όταν ξαναχρειαστεί.

Μία απλοποιημένη υλοποίηση ενός συγχρονιστή δύο καταστάσεων φαίνεται στο σχήμα 26. Επειδή η σχεδίαση απαιτεί το αποτέλεσμα να είναι διαθέσιμο όσο το δυνατό νωρίτερα, υπάρχει ένα συνδυαστικό κομμάτι τέτοιο ώστε η ενεργοποίηση της εξόδου να γίνει αμέσως μόλις ενεργοποιηθεί και η τελευταία είσοδος. Έτσι, στην περίπτωση του συγχρονισμού των εντολών ενός παράλληλου μπλοκ, ο συμπεριφορά του συγχρονιστή φαίνεται εντελώς διάφανη αφού μπορεί να θεωρηθεί ότι η κατάσταση εξόδου του παράλληλου μπλοκ είναι αυτή της πιο αργής εντολής.



Εικόνα 26 – Το εσωτερικό κύκλωμα ενός συγχρονιστή δύο καταστάσεων

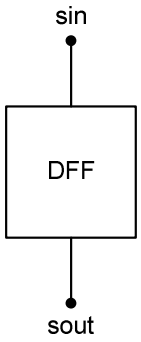
Αξίζει να σημειωθεί ότι μέσα σε ένα παράλληλο μπλοκ απαγορεύεται η χρήση των εντολών return, break και continue διότι δεν έχουν νόημα υπό το πρίσμα της παράλληλης εκτέλεσης.

Η κενή εντολή.

Μετά την εξέταση των μπλοκ εντολών, σειρά έχουν οι υπόλοιπες εντολές της C. Η πιο απλή από αυτές είναι η κενή εντολή, που συντάσσεται με ένα ελληνικό ερωτηματικό:

```
;
```

Η εντολή αυτή χρειάζεται ακριβώς έναν κύκλο ρολογιού για να εκτελεστεί και μπορεί να χρησιμοποιηθεί για να εισαχθούν σκόπιμα καθυστερήσεις σε κάποιο σημείο του προγράμματος. Το κύκλωμά της φαίνεται στην εικόνα 27: αντιστοιχεί σε ένα DFF μεταξύ των καταστάσεων εισόδου και εξόδου, προκαλώντας καθυστέρηση τους ενός κύκλου στο μονοπάτι ελέγχου.



Εικόνα 27 – Το κύκλωμα της κενής εντολής

Το παράδειγμα που είχε αναφερθεί σε προηγούμενη ενότητα για τη μορφή του μονοπατιού ελέγχου και το οποίο είχε παρουσιαστεί στην εικόνα 8, μπορεί πλέον να θεωρηθεί ως το κύκλωμα που αντιστοιχεί σε ένα σειριακό μπλοκ που περιέχει δύο κενές εντολές, δηλαδή το κύκλωμα του παρακάτω κώδικα:

```
{
   ;
   ;
}
```

Αν και δεν είναι αμέσως αντιληπτό, η κενή εντολή παρουσιάζει ενδιαφέρουσα χρησιμότητα στην πράξη. Πράγματι, καθώς έχει ήδη αναφερθεί και σύμφωνα με τις απαιτήσεις της σχεδίασης, αρκετές εντολές περιέχουν συνδυαστικό μονοπάτι μεταξύ των καταστάσεων εισόδου και εξόδου, προκειμένου να απαιτούν το λιγότερο δυνατό χρόνο εκτέλεσης (οποτεδήποτε βέβαια είναι αυτό εφικτό). Για παράδειγμα, όπως θα φανεί αργότερα στην παρουσίαση της εντολής if, μια εντολή if χωρίς else δεν απαιτεί καθόλου χρόνο εκτέλεσης αν η συνθήκη που ελέγχεται είναι ψευδής—σε μία τέτοια περίπτωση η κατάσταση εξόδου ενεργοποιείται στον ίδιο παλμό που ενεργοποιείται η κατάσταση εισόδου, χάρη σε αντίστοιχο συνδυαστικό κύκλωμα που περιέχει η εντολή if. Επειδή αυτή η φιλοσοφία ακολουθείται γενικότερα στις περισσότερες εντολές, είναι δυνατό στην κυκλωματική μετάφραση μιας ολόκληρης σειράς εντολών να υπάρχει ένα μεγάλο συνδυαστικό μονοπάτι. Το φαινόμενο αυτό, ενώ οδηγεί σε λιγότερους κύκλους ρολογιού εκτέλεσης, αυξάνει το λεγόμενο *κρίσιμο μονοπάτι* (critical path) του κυκλώματος, δηλαδή αυξάνει την ελάχιστη δυνατή περίοδο του παλμού ρολογιού, αφού μεταξύ δυο ακμών του ρολογιού υπάρχει ένα μεγάλο συνδυαστικό μονοπάτι για να διανυθεί. Έτσι παρότι το *throughput* του κυκλώματος είναι υψηλό, το ρολόι γίνεται πιο αργό, αφού πρέπει να κρατηθεί σε μικρότερους ρυθμούς για να υποστηρίξει το κύκλωμα. Για το σκοπό αυτό μπορεί λοιπόν να χρησιμοποιηθεί από τον προγραμματιστή η κενή εντολή ώστε να σπάσει το συνδυαστικό μονοπάτι, με το τίμημα των επιπλέον κύκλων ρολογιού.

Τα είδη των παραστάσεων.

Πριν εξεταστούν οι υπόλοιπες εντολές θα πρέπει να αναφερθούν κάποια στοιχεία σχετικά με τις παραστάσεις. Οι *παραστάσεις* (expressions) της C χρησιμοποιούνται σε πολλά διαφορετικά σημεία μέσα σε ένα πρόγραμμα. Το πρότυπο της C ορίζει ένα ευρύ φάσμα παραστάσεων και ουσιαστικά χειρίζεται τα περισσότερα πράγματα ως παραστάσεις, επιτρέποντας κάποιες καταχρήσεις σαν την κάτωθι:

```
int test(int y)
{
  int x;

  (x = func(y)) || (x = -1);
  return x;
}
```

Οι δυνατότητες αυτού του είδους δεν έχουν παραμείνει—αντιθέτως, οι παραστάσεις έχουν οριστεί πιο αυστηρά και η χρήση των τελεστών είναι πιο περιοριστική. Ορισμένοι τελεστές (όπως οι τελεστές του πολλαπλασιασμού και της διαίρεσης) δεν υποστηρίζονται, αλλά ένας προγραμματιστής μπορεί ούτως ή άλλως να κατασκευάσει συναρτήσεις εκτέλεσης πράξεων που να καλούνται αντί των τελεστών.

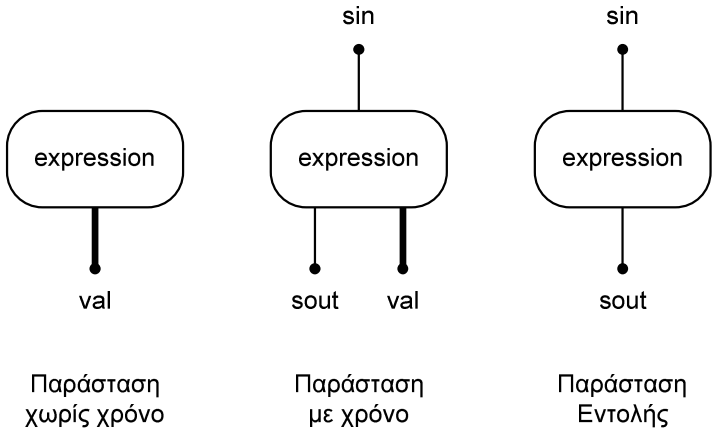
Για τις ανάγκες της σύνθεσης οι παραστάσεις που εμφανίζονται σε ένα πρόγραμμα εντάσσονται στα παρακάτω κύρια είδη, ο κυκλωματικός συμβολισμός των οποίων φαίνεται στην εικόνα 28:

- **Παραστάσεις χωρίς χρόνο (no-time expressions).** Οι παραστάσεις αυτού του είδους είναι παραστάσεις που αποτιμώνται άμεσα, χωρίς να χρειάζεται να παρέλθει έστω και ένας κύκλος ρολογιού. Η έννοια του χρόνου και των καταστάσεων ελέγχου δεν εφαρμόζεται σε αυτή την περίπτωση, και έτσι αυτές οι παραστάσεις απλώς εξάγουν ένα σήμα val που περιέχει το αποτέλεσμά τους. Οι περισσότερες πράξεις μπορούν να ενταχθούν σε αυτή την κατηγορία, διότι τα κυκλώματα που απαιτούνται για την υλοποίησή τους δεν χρειάζονται επιπλέον κύκλους ρολογιού, και δεν προκαλούν τις λεγόμενες *συνέπειες* ή *παρενέργειες* (side effects). Για παράδειγμα, η παράσταση ‘x + y’, όπου x και y δυο μεταβλητές, μπορεί εύκολα να ενταχθεί σε αυτή την κατηγορία: οι τιμές των μεταβλητών x και y (που στη γενική περίπτωση λαμβάνονται από τους αντίστοιχους καταχωρητές τους) εισάγονται στην είσοδο ενός αθροιστή, ο οποίος εξάγει το αποτέλεσμα. Έτσι μια παράσταση χωρίς χρόνο συμβολίζει στην πράξη ένα συνδυαστικό κύκλωμα του οποίου η έξοδος περιέχει ανά πάσα στιγμή την τιμή της παράστασης. Οι παραστάσεις χωρίς χρόνο χρησιμοποιούνται σε εντολές όπως οι if, while, do, κλπ.

- **Παραστάσεις με χρόνο (expressions with time).** Οι παραστάσεις αυτού του είδους ενδέχεται να απαιτούν χρόνο για τον υπολογισμό τους, υπό την έννοια ότι το αποτέλεσμά τους μπορεί να μην είναι διαθέσιμο στον ίδιο κύκλο ρολογιού στον οποίο ξεκινά η αποτίμησή τους. Έτσι, λαμβάνουν ως είσοδο την κατάσταση ελέγχου sin στην οποία ξεκινά η αποτίμηση τους, και εξάγουν την κατάσταση sout και το αποτέλεσμα val. Για παράδειγμα, όταν σε μία παράσταση με χρόνο εμφανίζεται η κλήση προς μια συνάρτηση, τότε η κατάσταση sin είναι η κατάσταση στην οποία θα πραγματοποιηθεί η κλήση προς τη συνάρτηση, ενώ η κατάσταση ελέγχου sout θα ενεργοποιηθεί μόλις επιστρέψει η συνάρτηση εξάγοντας το αποτέλεσμα val. Επειδή οι περισσότεροι τελεστές υλοποιούνται με απλά συνδυαστικά κυκλώματα, στην πράξη μόνο η ύπαρξη κλήσεων συναρτήσεων εντός μιας παράστασης με χρόνο μπορεί να μεταβάλλει την κατάσταση ελέγχου—οι υπόλοιπες πράξεις (όπως π.χ. ο τελεστής του μοναδιαίου ‘-’) μπορούν να θεωρηθούν ως παραστάσεις με χρόνο των οποίων η έξοδος sout ακολουθεί την είσοδο sin, αφού το αποτέλεσμα είναι διαθέσιμο

στιγμιαία. Οι παραστάσεις με χρόνο χρησιμοποιούνται στο δεξί σκέλος μιας εντολής ανάθεσης, στις παραστάσεις που αποτελούν ορίσματα κλήσης μιας συνάρτησης, κλπ.

– **Παραστάσεις εντολής (expression statements).** Ορισμένες παραστάσεις, όπως η κλήση συνάρτησης, οι διάφοροι τελεστές ανάθεσης, οι τελεστές αύξησης και μείωσης, κλπ., μπορούν να χρησιμοποιηθούν στη θέση εντολών. Οι παραστάσεις αυτές όταν χρησιμοποιούνται ως εντολές απεικονίζονται με το συμβολισμό των εντολών, δηλαδή λαμβάνουν μια κατάσταση εισόδου sin και εξάγουν μια κατάσταση sout για να σηματοδοτήσουν την ολοκλήρωση της εκτέλεσής τους. Η τελική τιμή μιας παράστασης εντολής δεν έχει κάποια χρησιμότητα ακριβώς διότι η παράσταση χρησιμοποιείται με σημασιολογία εντολής—έτσι π.χ. όταν στη θέση μιας εντολής εμφανίζεται η κλήση μιας συνάρτησης, η τιμή επιστροφής της συνάρτησης αγνοείται.



Εικόνα 28 – Τα κύρια είδη των παραστάσεων

Η παραπάνω κατηγοριοποίηση γίνεται σύμφωνα με τις απαιτήσεις της σύνθεσης, δηλαδή κάθε φορά που ζητείται η συμμετοχή μιας παράστασης πρέπει να εμπίπτει σε κάποια τέτοια κατηγορία, ανάλογα με το σημείο του προγράμματος στο οποίο εμφανίζεται. Οι κατηγορίες αυτές δεν είναι αμοιβαίως αποκλειόμενες: η παράσταση ‘x+y’ μπορεί να συμμετέχει και ως παράσταση με χρόνο και ως παράσταση χωρίς χρόνο, ενώ μια παράσταση κλήσης συνάρτησης ‘myfunc()’ μπορεί να συμμετέχει και ως παράσταση με χρόνο και ως παράσταση εντολής.

Το ζήτημα των παραστάσεων θα διαλευκανθεί περισσότερο σε επόμενη παράγραφο όπου και θα αναλυθούν οι διάφοροι τελεστές, ο τρόπος αποτίμησης, κλπ. Προς το παρόν αξίζει να συγκρατήσει κανείς τους συμβολισμούς της εικόνας 28 και τις σημασίες που αντιπροσωπεύουν.

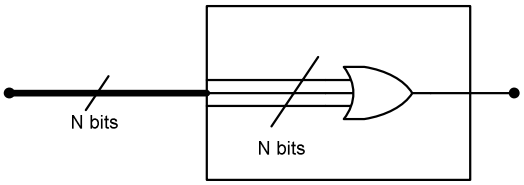
Η εντολή if. Η μετατροπή σε τιμή αληθείας.

Η εντολή if συντάσσεται ως εξής:

```
if ( <expression> ) <statement> [ else <statement> ]
```

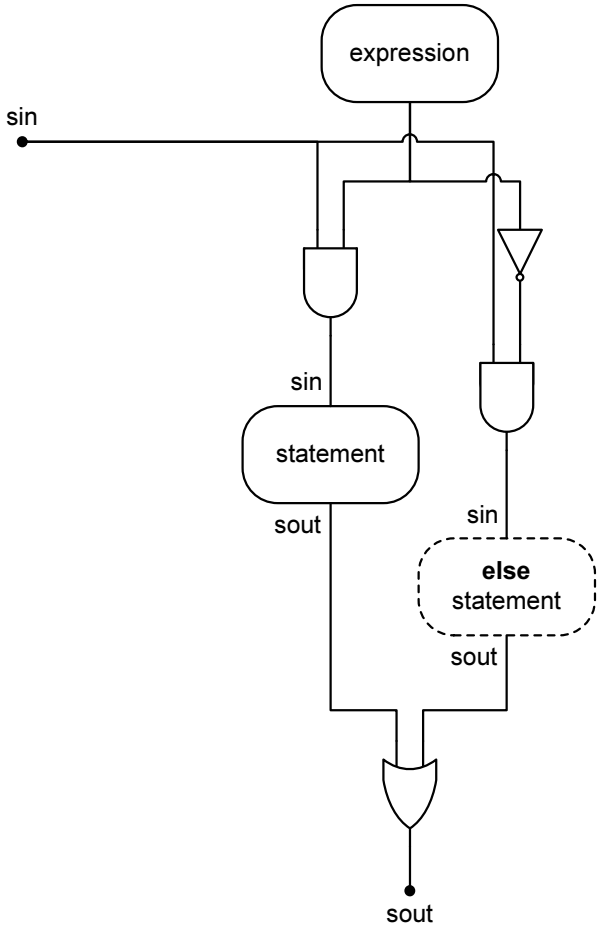
Η σημασιολογία της εντολής if είναι ως γνωστόν η εξής: αν η τιμή της λογικής παράστασης ‘expression’ είναι αληθής τότε εκτελείται η πρώτη εντολή ‘statement’, αλλιώς εκτελείται η εντολή ‘statement’ που βρίσκεται μετά από το ‘else’, αν υπάρχει.

Η παράσταση με την οποία συντάσσεται η if είναι μια παράσταση χωρίς χρόνο, δηλαδή θα πρέπει να αποτιμάται άμεσα (χωρίς να χρειαστούν επιπλέον κύκλοι ρολογιού). Επειδή η παράσταση αντιπροσωπεύει μια λογική συνθήκη, η αποτίμησή της πρέπει να οδηγεί σε αποτέλεσμα του ενός bit, το οποίο περιέχει την τιμή αληθείας. Στη γενική περίπτωση που η αποτίμηση της παράστασης οδηγεί σε ένα διάνυσμα των N bits, διατηρείται η γνωστή σημασιολογία σύμφωνα με την οποία η τιμή θεωρείται αληθής αν είναι μη μηδενική. Για το σκοπό αυτό πραγματοποιείται η *μετατροπή σε τιμή αληθείας* (truth value conversion) με την εισαγωγή μιας πύλης OR των N εισόδων, όπως φαίνεται και στην εικόνα 29.



Εικόνα 29 – Το κύκλωμα μετατροπής σε τιμή αληθείας

Το κύκλωμα το οποίο συντίθεται για μια εντολή if φαίνεται στην εικόνα 30.



Εικόνα 30 – Το κύκλωμα της εντολής if

Η αντιστοίχηση σε κύκλωμα γίνεται με απλό τρόπο: στη γενική περίπτωση, όπου υφίσταται και η πρόταση ‘else’, η κατάσταση εισόδου sin της εντολής if μεταδίδεται σε μία από τις δύο εντολές, η επιλογή της οποίας γίνεται με βάση το σήμα εξόδου της αποτιμημένης παράστασης (το οποίο έχει υποστεί την ενδεχόμενη μετατροπή σε τιμή αληθείας, που για λόγους απλότητας παραλείπεται από το σχήμα). Η τελική κατάσταση εξόδου sout προκύπτει από το λογικό OR των αντίστοιχων καταστάσεων εξόδου των δύο εντολών.

Σε περίπτωση απουσίας της πρότασης ‘else’, απλά παραλείπεται η δεύτερη εντολή που συμβολίζεται με διακεκομμένη γραμμή. Έτσι, δημιουργείται σίγουρα ένα συνδυαστικό μονοπάτι μεταξύ των καταστάσεων εισόδου sin και sout, που σημαίνει ότι όταν η λογική συνθήκη είναι ψευδής και ενεργοποιηθεί το σήμα sin τότε η εντολή θα εκτελεστεί αμέσως ενεργοποιώντας το σήμα εξόδου sout, ώστε να ξεκινήσει η εκτέλεση της επόμενης κατά σειρά εντολής μέσα στον ίδιο κύκλο ρολογιού. Κατ’ αυτόν τον τρόπο, αλλεπάλληλες εντολές if (είτε είναι φωλιασμένες είτε όχι) μπορούν να πραγματοποιήσουν γρήγορο έλεγχο των συνθηκών τους, ώστε όσες είναι ψευδείς να μην καθυστερήσουν με επιπλέον κύκλο ρολογιού το κύκλωμα. Βέβαια, όπως αναφέρθηκε και νωρίτερα, κάτι τέτοιο αυξάνει το κρίσιμο μονοπάτι του κυκλώματος, αλλά ο προγραμματιστής μπορεί να εισάγει π.χ. την κενή εντολή στην πρόταση else, για να επιβάλει την πάροδο του παλμού ρολογιού και να κατακερματίσει τα μεγάλα συνδυαστικά μονοπάτια.

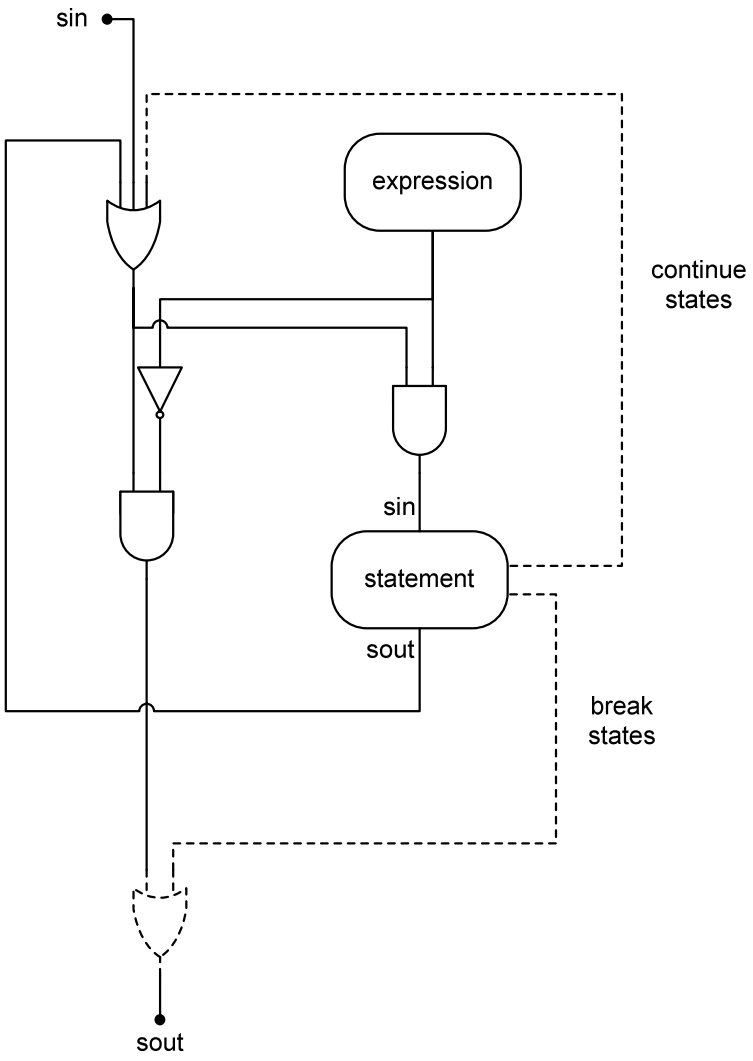
Η εντολή while.

Η εντολή while συντάσσεται ως εξής:

```
while ( <expression> ) <statement>
```

Η εντολή while ορίζει μια δομή επανάληψης (βρόχου): όσο η λογική παράσταση με την οποία συντάσσεται παραμένει αληθής, εκτελείται η αντίστοιχη εντολή που αποτελεί το σώμα του βρόχου. Ο έλεγχος της συνθήκης γίνεται μία φορά στην αρχή και από εκεί και πέρα ξανά στο τέλος κάθε εκτέλεσης της εντολής του βρόχου. Από σχεδιαστική άποψη η λογική παράσταση απαιτείται να είναι μια παράσταση χωρίς χρόνο, όπως δηλαδή και στην εντολή if, προκειμένου η αποτίμησή της να γίνεται άμεσα. Επίσης, υφίσταται κι εδώ η μετατροπή σε τιμή αληθείας, αν είναι αναγκαία.

Η εντολή while επιτρέπει την έξοδο από το βρόχο με τη χρήση της εντολής break, καθώς και την επανάληψη του βρόχου με τη χρήση της εντολής continue. Συνεπώς, το κύκλωμα της εντολής ‘statement’ ενδέχεται να εξάγει μια σειρά από *καταστάσεις επανάληψης* (continue states) και *καταστάσεις διαφυγής* (break states), δηλαδή καταστάσεις ελέγχου που δημιουργούνται από τις εντολές continue και break αντίστοιχα. (περισσότερες πληροφορίες στις αντίστοιχες περιγραφές των εντολών αυτών). Οι καταστάσεις αυτές συμβολίζονται με διακεκομμένες γραμμές που πηγάζουν από το κύκλωμα της εντολής ‘statement’, όπως δείχνει και η εικόνα 31 στην οποία αναπαρίσταται το κυκλωματικό ισοδύναμο της while.



Εικόνα 31 – Το κύκλωμα της εντολής while

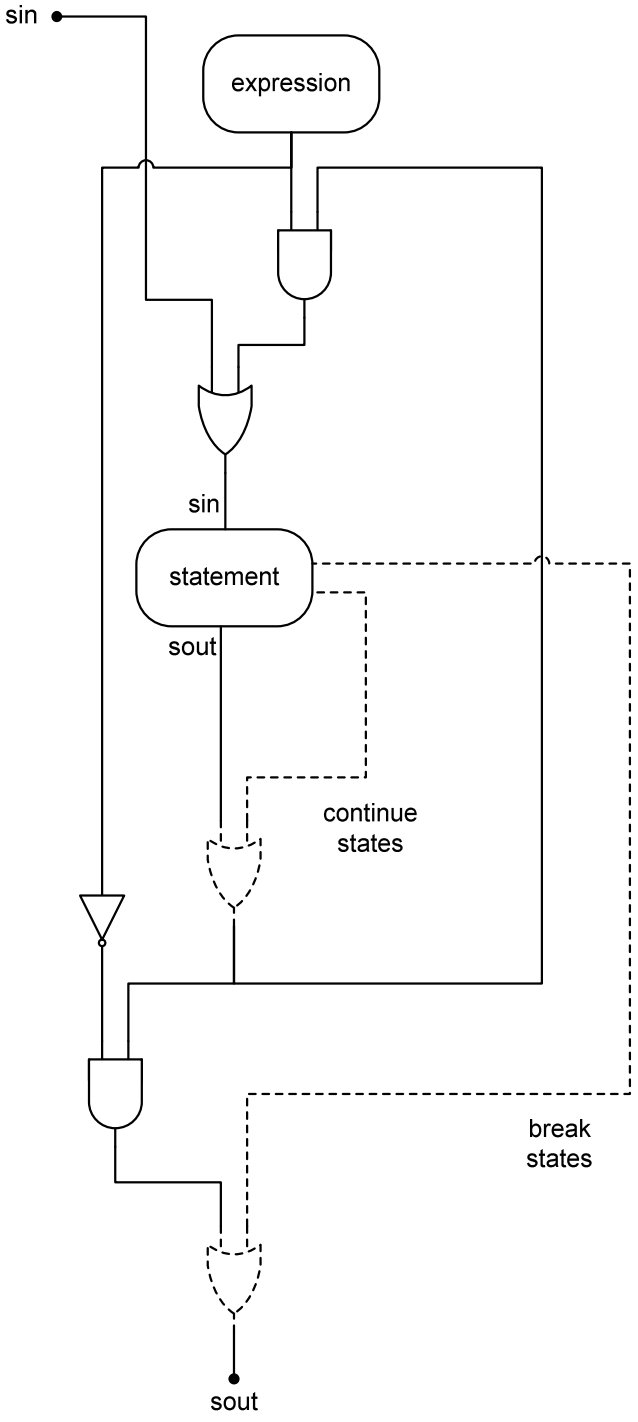
Το κύκλωμα που αντιστοιχεί μοιάζει με αυτό της εντολής if, με τη διαφορά ότι η κατάσταση εξόδου της εντολής ‘statement’ οδηγεί πίσω στην είσοδο. Η τελική πύλη διάζευξης υφίσταται μόνο αν υπάρχει τουλάχιστον μια κατάσταση διαφυγής (break state). Επίσης, αν η λογική συνθήκη είναι εξ’ αρχής ψευδής, τότε η εντολή ολοκληρώνεται άμεσα χωρίς να παρέλθει κύκλος ρολογιού. Τέλος, όπως είναι φανερό και από το σχήμα, δεν εισάγεται καμία ακολουθιακή καθυστέρηση μετά από την κατάσταση εξόδου sout της εντολής ‘statement’ του βρόχου. Έτσι, σε κάθε επανάληψη της εντολής του βρόχου, το ενεργό σήμα εξόδου sout της εντολής ‘statement’ ανατροφοδοτεί την ίδια την είσοδο της. Αν η εντολή περιέχει στο εσωτερικό της συνδυαστικό μονοπάτι μεταξύ των καταστάσεων εισόδου και εξόδου, τότε δημιουργείται ένα συνδυαστικό loop το οποίο δεν μπορεί να συντεθεί σε πραγματικό υλικό, μιας και η περίοδος του ρολογιού απειρίζεται. Ο προγραμματιστής θα πρέπει να είναι προσεκτικός ώστε να μη δημιουργούνται συνδυαστικά loops, που στην πιο απλή περίπτωση μπορούν να αποφευχθούν με την κατάλληλη παρεμβολή της κενής εντολής.

Η εντολή do.

Η εντολή do συντάσσεται ως εξής:

```
do <statement> while ( <expression> )
```

Η εντολή do ορίζει άλλη μια δομή επανάληψης, όπως και η while, με τη διαφορά ότι πρώτα εκτελείται η εντολή ‘statement’ του βρόχου και ύστερα ελέγχεται η τιμή της λογικής παράστασης ‘expression’. Συνεπώς, είναι σίγουρο ότι η περιεχόμενη εντολή θα εκτελεστεί τουλάχιστον μία φορά. Κατά τα γνωστά, η λογική συνθήκη είναι παράσταση χωρίς χρόνο που αποτιμάται σε ένα bit, με ενδεχόμενη μετατροπή σε τιμή αληθείας, ενώ υπάρχουν εξίσου οι καταστάσεις επανάληψης (continue states) και διαφυγής (break states). Το κύκλωμα που συντίθεται από μία εντολή do φαίνεται στην εικόνα 32.



Εικόνα 32 – Το κύκλωμα της εντολής do

Το κύκλωμα που αντιστοιχεί στην εντολή do είναι λίγο διαφορετικό από αυτό της εντολής while καθώς ο έλεγχος της συνθήκης γίνεται στο τέλος κάθε επανάληψης και όχι στην αρχή. Οι πύλες OR που εμφανίζονται με διακεκομμένες γραμμές συμμετέχουν μόνο εφόσον υφίστανται αντίστοιχα σήματα break ή continue. Κατά τα άλλα, ισχύουν όσα αναφέρθηκαν και στην εντολή while περί συνδυαστικών loops.

Η εντολή for.

Η εντολή for συντάσσεται ως εξής:

```
for ( [ <init_expr> ] ; [ <cond_expr> ] ; [ <trans_expr> ] )
<statement>
```

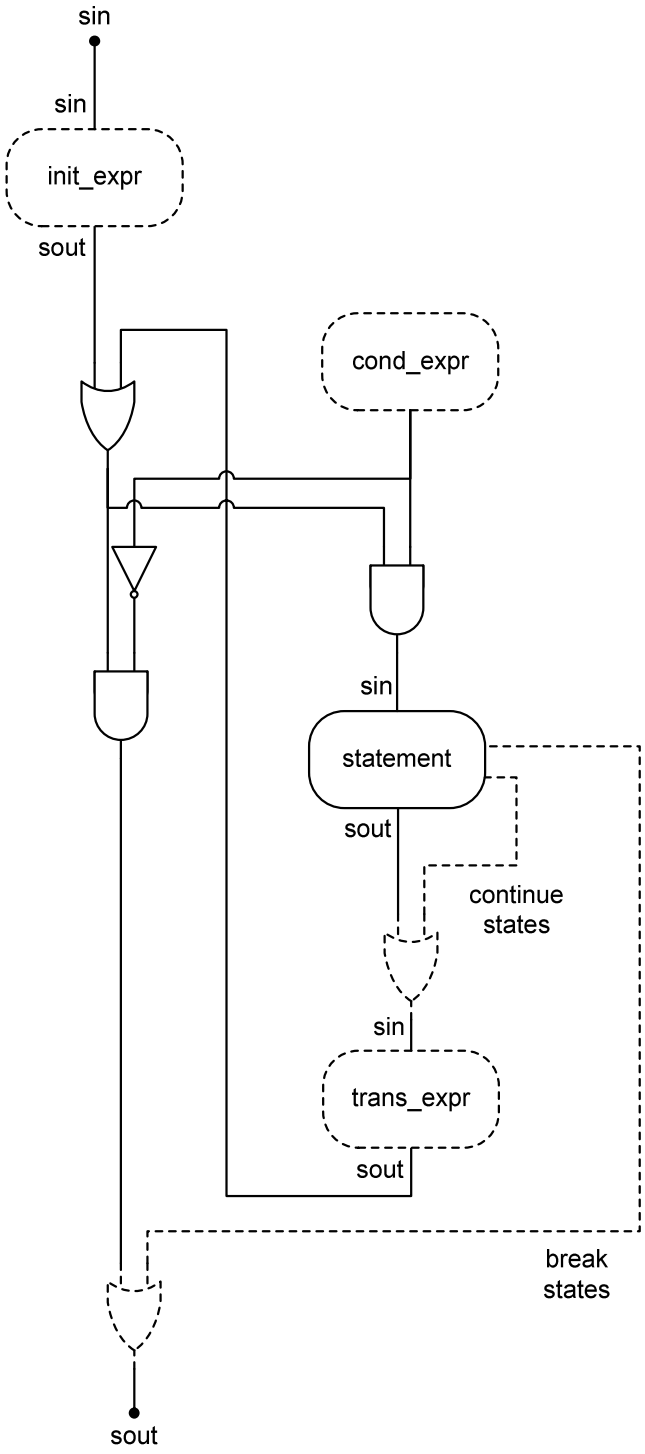
Η εντολή for ορίζει εξίσου μια δομή επανάληψης. Χρησιμοποιούνται τρεις διαφορετικές (και μάλιστα προαιρετικές) παραστάσεις:

- Η *παράσταση αρχικοποίησης* (initialization) είναι μια *παράσταση εντολής* που χρησιμοποιείται για μια συνολική αρχικοποίηση. Εκτελείται μόνο μια φορά κατά την έναρξη της εκτέλεσης της εντολής for.
- Η *παράσταση συνθήκης* (condition) είναι μια *παράσταση χωρίς χρόνο*, η τιμή αληθείας της οποίας (όπως και στην εντολή while) καθορίζει την έξοδο από το βρόχο. Ο έλεγχος της παράστασης συνθήκης γίνεται μία φορά στην αρχή (μετά από την παράσταση αρχικοποίησης) αλλά και μετά από κάθε επανάληψη (μετά από την παράσταση μετάβασης).
- Η *παράσταση μετάβασης* (transition) είναι μια *παράσταση εντολής* που εκτελείται μετά από κάθε ολοκλήρωση της εκτέλεσης της εντολής ‘statement’ του βρόχου.

Για την παράσταση συνθήκης ισχύουν όσα έχουν αναφερθεί και για τη λογική παράσταση των εντολών while και do: αποτιμάται άμεσα, και αν το αποτέλεσμα είναι δύο ή περισσοτέρων bits, τότε υφίσταται μετατροπή σε τιμή αληθείας. Οι καταστάσεις τύπου break και continue είναι και εδώ παρούσες—μάλιστα μια εντολή continue παρακάμπτει το υπόλοιπο του βρόχου αλλά όχι και την παράσταση μετάβασης, η οποία προηγείται της επανάληψης.

Το κύκλωμα που αντιστοιχεί στην εντολή for φαίνεται στην εικόνα 33. Η παράληψη της παράστασης αρχικοποίησης ή της παράστασης μετάβασης απλά αφαιρεί από το κύκλωμα το αντίστοιχο διακεκομμένο κομμάτι. Η παράληψη της παράστασης συνθήκης εισάγει στη θέση της ένα σήμα που είναι πάντοτε αληθές, δηλαδή το λογικό 1. Βέβαια σε αυτή την τελευταία περίπτωση, ένας μεταγλωττιστής μπορεί να επιτελέσει και άλλες βελτιστοποιήσεις, αφαιρώντας τον αντιστροφέα και ορισμένες πύλες AND. Μάλιστα η σύνταξη ‘for(;;)’ δεν εισάγει παρά ένα σύρμα ανάδρασης μεταξύ της εξόδου και της εισόδου της εντολής ‘statement’, προκειμένου ο βρόχος να εκτελείται συνέχεια.

Και η εντολή for είναι δυνατό να οδηγήσει σε ένα συνδυαστικό loop, αν η παράσταση μετάβασης παραλείπεται και ταυτόχρονα η εντολή ‘statement’ περιέχει συνδυαστικό μονοπάτι.



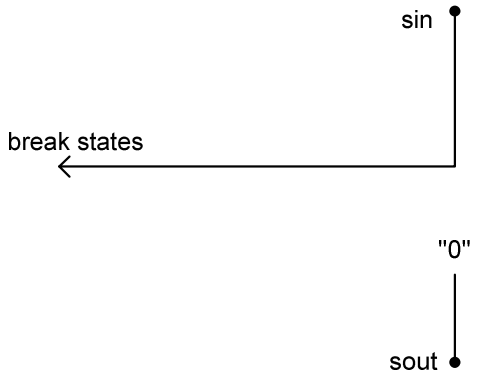
Εικόνα 33 – Το κύκλωμα της εντολής for

Η εντολή break.

Η εντολή break συντάσσεται ως εξής:

```
break ;
```

Η εντολή break χρησιμοποιείται μέσα σε ένα βρόχο while, do ή for για τη διαφυγή από το βρόχο, δηλαδή την έξοδο από τη δομή επανάληψης. Η εμφάνισή της προσαρτά στη λίστα των καταστάσεων διαφυγής την κατάσταση εισόδου στην οποία εμφανίζεται. Ως κατάσταση εξόδου παράγεται ένα σήμα το οποίο είναι μόνιμα συνδεδεμένο στο λογικό 0, ώστε να μην εκτελεστεί η εντολή που ακολουθεί.



Εικόνα 34 – Το κύκλωμα της εντολής break

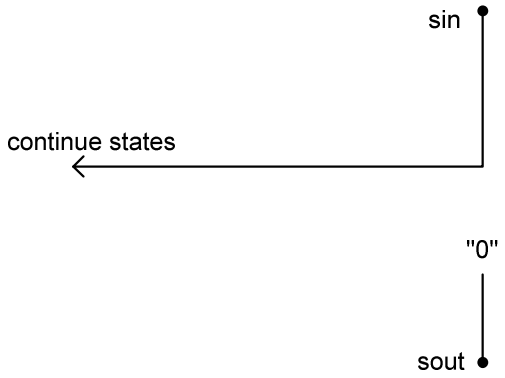
Η εμφάνιση της εντολής break δεν επιτρέπεται μέσα σε παράλληλα μπλοκ εντολών, ούτε σε περιπτώσεις όπου δεν υπάρχει πλησιέστερη εντολή βρόχου (while, do ή for) στην οποία να αναφέρεται η διαφυγή. Επίσης η εντολή break δύναται να εμφανίζεται στο σώμα μιας εντολής switch. Ο χειρισμός αυτής της περίπτωσης γίνεται με διαφορετικό τρόπο και αναλύεται στην περιγραφή της εντολής switch.

Η εντολή continue

Η εντολή continue συντάσσεται ως εξής:

```
continue ;
```

Η εντολή continue δρα κατά τρόπο ανάλογο με αυτόν της εντολής break.



Εικόνα 35 – Το κύκλωμα της εντολής continue

Η εντολή return. Η μετατροπή τύπου.

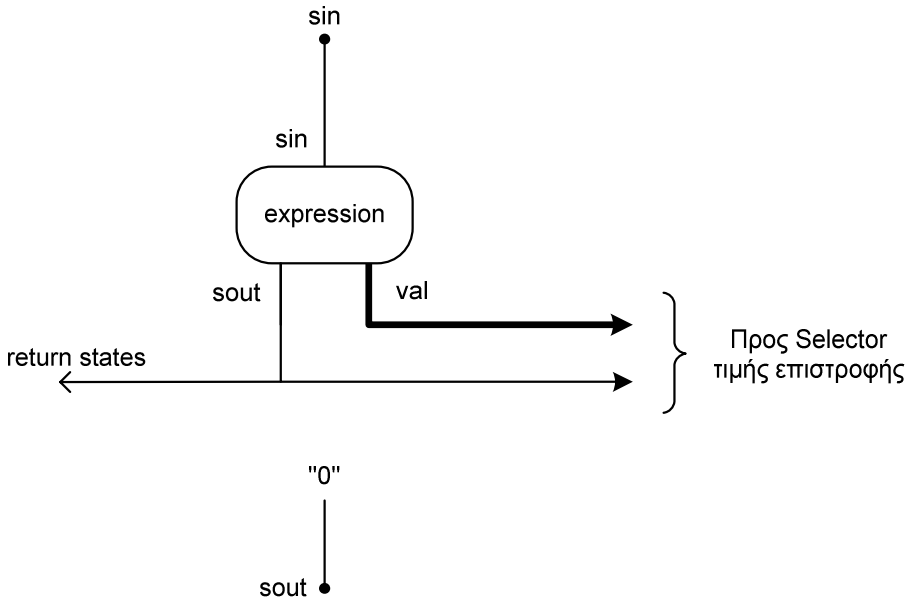
Η εντολή return συντάσσεται ως εξής:

```
return [ <expression> ] ;
```

Η εντολή return χρησιμοποιείται από μία συνάρτηση για την επιστροφή πίσω στο καλούν πρόγραμμα. Αν η συνάρτηση επιστρέφει κάποια τιμή, δηλαδή δεν έχει τύπο επιστροφής 'void', τότε η εντολή πρέπει να συνοδεύεται από μία παράσταση, η τιμή της οποίας θα επιστραφεί. Η παράσταση 'expression' που θα αποτιμηθεί είναι μια *παράσταση με χρόνο*. Σύμφωνα με όσα έχουν ήδη αναφερθεί, μια τέτοια παράσταση ενδέχεται να μεταβάλλει την κατάσταση ελέγχου sin που δέχεται (στην περίπτωση που περιέχει π.χ. κλήσεις προς άλλες συναρτήσεις, οπότε και πρέπει να αναμένει για την ολοκλήρωση τους). Το σήμα sout της παράστασης ενεργοποιείται όταν ολοκληρωθεί η αποτίμηση, οπότε και θα είναι έγκυρη η τιμή val. Η τιμή αυτή θα πρέπει να μετατραπεί στον τύπο επιστροφής της συνάρτησης—η διαδικασία αυτή γίνεται με τον ίδιο τρόπο με αυτόν που αναφέρθηκε σε προηγούμενη παράγραφο για τις σταθερές. Γενικότερα, η διαδικασία της *μετατροπής τύπου* (type conversion) μιας τιμής από έναν παλιό τύπο σε ένα νέο γίνεται με τον εξής τρόπο:

- Αν οι δύο τύποι έχουν το ίδιο πλήθος από bits, τότε η τιμή δε χρειάζεται να υποστεί περαιτέρω αλλαγές.
- Αν ο νέος τύπος έχει μικρότερο μέγεθος από τον παλιό, τότε απορρίπτονται bits από το αριστερό (περισσότερο σημαντικό) κομμάτι της τιμής ώστε να παραμείνουν τόσα bits όσα ορίζει ο νέος τύπος. Με άλλα λόγια, παραμένουν τελικά τόσα από τα δεξιότερα (λιγότερο σημαντικά) bits της τιμής όσα απαιτεί ο νέος τύπος.
- Αν ο νέος τύπος έχει μεγαλύτερο μέγεθος από τον παλιό, τότε η τιμή πρέπει να επεκταθεί μέχρι του πλήθους των bits του νέου τύπου, δηλαδή πρέπει να εισαχθούν στο αριστερότερο (περισσότερο σημαντικό) μέρος κάποια επιπλέον ψηφία. Η επέκταση γίνεται με βάση τον παλιό τύπο: αν είναι απρόσημος (unsigned) τότε προσαρτώνται μηδενικά, ενώ αν είναι προσημασμένος (signed) τότε γίνεται επέκταση προσήμου, δηλαδή επαναλαμβάνεται το αριστερότερο bit της τιμής, διατηρώντας το πρόσημο.

Η κυκλωματική αντιστοιχία της εντολής return, στη γενική περίπτωση όπου υφίσταται τιμή επιστροφής, φαίνεται στην εικόνα 36. Μόλις υπολογιστεί η παράσταση προς επιστροφή, η κατάσταση εξόδου sout στην οποία είναι διαθέσιμη η παράσταση εισάγεται μαζί με την αντίστοιχη τιμή val (της οποίας η μετατροπή τύπου έχει παραληφθεί από το σχήμα για λόγους απλότητας αλλά θα πρέπει να εννοείται) στον επιλογέα της τιμής επιστροφής της συνάρτησης (που παρουσιάστηκε στην εικόνα 16). Συνάμα, η ίδια κατάσταση sout της παράστασης εξάγεται στη λίστα των καταστάσεων επιστροφής (return states) από τις οποίες η συνάρτηση πραγματοποιεί την έξοδό της. Το DFF εξόδου που βρίσκεται στον σκελετό της συνάρτησης (εικόνα 14) φροντίζει ώστε να παρέλθει ένας κύκλος ρολογιού και να γραφτούν τα δεδομένα στον καταχωρητή της τιμής επιστροφής πριν επιστρέψει η συνάρτηση. Ας τονιστεί εδώ ότι αν η εντολή return συντάσσεται χωρίς παράσταση, τότε το σήμα εισόδου sin είναι αυτό που εξάγεται στη λίστα των καταστάσεων επιστροφής. Σε κάθε περίπτωση, η εντολή οφείλει να εξάγει και μια συνολική κατάσταση εξόδου sout, η οποία είναι μονίμως ανενεργή (συνδέεται με το λογικό 0) ώστε να μην εκτελεστεί ποτέ η επόμενη κατά σειρά εντολή.



Εικόνα 36 – Το κύκλωμα της εντολής return

Η εντολή switch. Οι ετικέτες case και default.

Η εντολή switch χρησιμοποιείται για τη διάκριση μεταξύ διαφορετικών τιμών μιας παράστασης, ώστε να εκτελεστεί και ο ανάλογος κώδικας. Αν και η συμπεριφορά της μοιάζει με αυτή αλλεπάλληλων διατάξεων if-then-else, στην πραγματικότητα είναι πιο σύνθετη. Μια όχι και τόσο αυστηρή σύνταξη δίνεται παρακάτω:

```
switch ( <expression> )
{
  case <constant> :
    <statement>*
    [ break ; ]

  case <constant> :
    <statement>*
    [ break ; ]

  …

  [
  default :
    <statement>*
    [ break ; ]
  ]
}
```

Όπως φαίνεται, η εντολή switch λαμβάνει μια παράσταση εντός παρενθέσεων, ακολουθούμενη υποχρεωτικά από ένα ειδικό μπλοκ εντολών. Μέσα σε αυτό το μπλοκ εντολών υπάρχουν *ετικέτες* case (case labels) οι οποίες διακρίνουν μεταξύ διαφόρων σταθερών. Το πρόγραμμα ελέγχει αν η τιμή της παράστασης είναι ίση με καθεμία από τις σταθερές που περιλαμβάνονται σε ετικέτες case, ώστε να μεταφερθεί ο έλεγχος στο αντίστοιχο σημείο του μπλοκ, και να ξεκινήσει η εκτέλεση των εντολών που βρίσκονται μετά από την ετικέτα της οποίας η σταθερά είναι ίση με την παράσταση. Η εκτέλεση των εντολών του μπλοκ ακολουθεί τη σειριακή σημασιολογία, δηλαδή οι

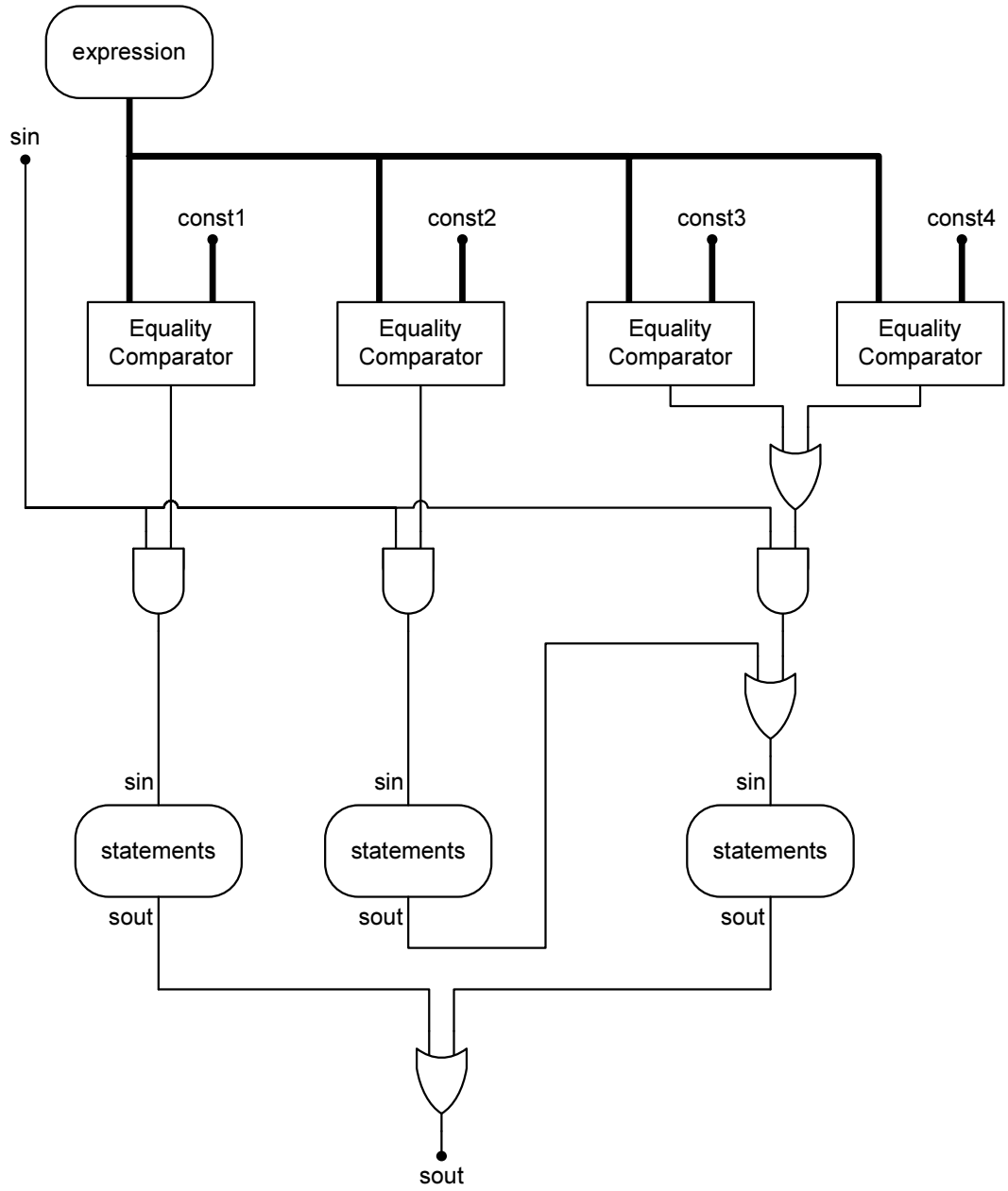
εντολές εκτελούνται με τη σειρά που είναι γραμμένες στο μπλοκ, μέχρι η εκτέλεση να συναντήσει μια εντολή break, οπότε και ολοκληρώνεται η εντολή switch. Βέβαια, η εντολή break είναι προαιρετική: η απουσία της σημαίνει ότι θα συνεχίσουν να εκτελούνται οι εντολές που βρίσκονται μετά από την επόμενη κατά σειρά ετικέτα, κ.ο.κ. Επίσης, οι ετικέτες case ενδέχεται να είναι ομαδοποιημένες. Πράγματι, αν μια ετικέτα case δεν περιλαμβάνει εντολές αλλά ακολουθείται από άλλη ετικέτα, τότε στην ουσία έχουν δηλωθεί περισσότερες σταθερές προς έλεγχο, οι οποίες θα εκτελέσουν τις ίδιες εντολές: αυτές που ακολουθούν ύστερα από την ομάδα των ετικετών case. Τέλος, υπάρχει και η ετικέτα default η οποία είναι προαιρετική και δηλώνει το σημείο στο οποίο θα μεταφερθεί ο η εκτέλεση αν τελικά καμία σταθερά δεν είναι ίση με την παράσταση. Η σημασιολογία της εντολής switch δε θα εξηγηθεί περαιτέρω, καθώς είναι ήδη γνωστή στους περισσότερους από την προγραμματιστική τους εμπειρία.

Η σύνθεση της εντολής switch γίνεται βάση ενός αλγόριθμου παραγωγής αντίστοιχων κυκλωμάτων ο οποίος θα περιγραφεί στη γενική του μορφή, αλλά ταυτόχρονα θα εξηγείται και μέσα από ένα παράδειγμα, για ευκολότερη κατανόηση. Για τις ανάγκες αυτού του παραδείγματος, ας θεωρηθεί η παρακάτω περίπτωση σύνταξης:

```
switch ( <expression> )
{
  case <const1> :
    <statement>*
    break ;

  case <const2> :
    <statement>*

  case <const3> :
  case <const4> :
    <statement>*
    break ;
}
```

Το τελικό αποτέλεσμα του αλγορίθμου σύνθεσης της εντολής switch, για την παραπάνω σύνταξη, φαίνεται στην εικόνα 37.

Ο αλγόριθμος λειτουργεί ως εξής: Αρχικά, πρέπει να υπολογιστεί η τιμή της παράστασης της εντολής switch. Η παράσταση αυτή απαιτείται να είναι μια *παράσταση χωρίς χρόνο*. Έτσι, κατά τα γνωστά, η παράσταση ‘expression’ αποτιμάται σε ένα διάνυσμα από bits, διαθέσιμο άμεσα κατά την έναρξη της εκτέλεσης της εντολής. Η τιμή αυτή της παράστασης πρέπει να συγκριθεί με όλες τις σταθερές τιμές που έχουν δοθεί στις ετικέτες case. Έτσι, για κάθε ετικέτα case που υπάρχει εντός του μπλοκ, η αντίστοιχη σταθερά εισάγεται σε ένα *συγκριτή ισότητας* (equality comparator) μαζί με την τιμή της παράστασης. Η σταθερά είναι προϋπολογισμένη σε ένα δυαδικό διάνυσμα (του ίδιου μεγέθους με αυτό του τύπου της παράστασης) κατά τη φάση της μεταγλώττισης, σύμφωνα με όσα έχουν αναφερθεί αναλυτικά για τις σταθερές σε προηγούμενη παράγραφο. Οι συγκριτές, που είναι απλά συνδυαστικά κυκλώματα, ελέγχουν αν τα δύο ορίσματά τους είναι ίσα bit-προς-bit, και εξάγουν ένα λογικό σήμα ισότητας. Έτσι, ο συγκριτής του οποίου η σταθερά και η τιμή της παράστασης είναι ίσες θα ενεργοποιήσει την έξοδό του. Στο παράδειγμα, εισάγονται τέσσερεις συγκριτές για τις αντίστοιχες σταθερές ‘const1’ ως ‘const4’.



Εικόνα 37 – Το κύκλωμα μιας παραδειγματικής εντολής switch

Στη συνέχεια οι ετικέτες που είναι ομαδοποιημένες στον κώδικα ορίζουν αντίστοιχες ομαδοποιήσεις και στο κύκλωμα: τα σήματα των συγκριτών που αντιστοιχούν σε διαδοχικές ετικέτες case υφίστανται λογικό OR ώστε για κάθε ομάδα να προκύψει ένα ενιαίο σήμα. Στο παράδειγμα, τα σήματα που ομαδοποιούνται είναι αυτά των συγκριτών που αντιστοιχούν στις σταθερές ‘const3’ και ‘const4’, αφού στον κώδικα του παραδείγματος οι αντίστοιχες ετικέτες είναι διαδοχικά γραμμένες.

Κάθε σήμα που έχει δημιουργηθεί (είτε από μεμονωμένες είτε από ομαδοποιημένες ετικέτες) εισάγεται σε μια ξεχωριστή πύλη AND μαζί με το σήμα sin της κατάστασης εισόδου. Έτσι, στις εξόδους των πυλών AND προκύπτουν νέα σήματα που υποδηλώνουν, όταν θα είναι ενεργά, ότι έφτασε η ώρα της εκτέλεσης της εντολής switch και ότι η εκτέλεση θα μεταφερθεί στο συγκεκριμένο σημείο που βρίσκεται το κάθε σήμα, επειδή η σύγκριση κάποιας ετικέτας ήταν επιτυχής. Από εκεί και πέρα ο αλγόριθμος

συνεχίζει ως εξής: κάθε σήμα από την έξοδο των πυλών AND τροφοδοτεί, ως κατάσταση εισόδου, το κύκλωμα της πρώτη εντολής που ακολουθεί την αντίστοιχη ετικέτα. Έτσι, στο παράδειγμα τροφοδοτούνται οι τρεις ομάδες εντολών, που για ευκολία σημειώνονται στο σχήμα ως ‘statements’, αλλά πρέπει να εννοείται ότι κάθε τέτοιο κουτί ‘statements’ είναι μια σειριακή αλυσίδα εντολών (σαν αυτή της εικόνας 24). Αν στο τέλος μιας ομάδας εντολών εμφανίζεται η εντολή break, τότε η κατάσταση εξόδου της τελευταίας εντολής εξάγεται προς την τελική κατάσταση sout της εντολής switch. Ακριβέστερα, οι καταστάσεις από τις τελευταίες εντολές εισάγονται σε μία πύλη OR ώστε να προκύψει η συνολική κατάσταση εξόδου sout. Στο παράδειγμα, αυτό συμβαίνει με τις τελικές εντολές όλων των ετικετών εκτός από αυτές της ετικέτας ‘const2’. Στην περίπτωση που η εντολή break παραλείπεται από τις εντολές μιας ετικέτας (όπως π.χ. από την ετικέτα της σταθεράς ‘const2’ του παραδείγματος), τότε ο έλεγχος συνεχίζει στην επόμενη εντολή. Έτσι, η πρώτη εντολή της επόμενης ετικέτας πρέπει να λάβει και άλλο σήμα έναρξης: το σήμα εξόδου της προηγούμενής της ετικέτας. Συνεπώς, στο παράδειγμα ο έλεγχος μετά από τις εντολές της δεύτερης ετικέτας ‘const2’ μεταβιβάζεται μέσω μιας επιπλέον πύλης OR στις εντολές της τρίτης ομάδας. Άρα κάθε ομάδα δεν λαμβάνει ως κατάσταση εισόδου μόνο το σήμα της πύλης AND, αλλά υπάρχει ενδεχομένως και το τελικό σήμα της εντολής της προηγούμενης ετικέτας (αν αυτή δεν περιέχει την εντολή break). Ας σημειωθεί επίσης ότι αν η εντολή break εμφανίζεται νωρίτερα σε κάποιο σημείο, δηλαδή δεν είναι η τελευταία εντολή μιας ομάδας αλλά ακολουθείται και από άλλες εντολές πριν την επόμενη ετικέτα, τότε οι εντολές αυτές αγνοούνται καθώς είναι αδύνατο να φθάσει σε αυτές ποτέ ο έλεγχος. Αυτό γίνεται και για τυχόν εντολές που υπάρχουν στην αρχή του μπλοκ της switch, πριν από την πρώτη ετικέτα.

Για τον χειρισμό της ετικέτας default γίνεται το εξής: Αφού παραχθούν όλοι οι συγκριτές, τα σήματα εξόδου αυτών εισάγονται σε μία πύλη NOR, και το προκύπτον σήμα θεωρείται το σήμα σύγκρισης για την ετικέτα default,. Ύστερα, ο χειρισμός του σήματος αυτού δεν διαφέρει από τα υπόλοιπα, δηλαδή θεωρείται από τον αλγόριθμο σα να είχε προκύψει από έναν καινούριο συγκριτή. Ενδέχεται μάλιστα να συμμετέχει και σε ομαδοποιήσεις, αν η ετικέτα default βρίσκεται διαδοχικά μαζί με μία ετικέτα case.

Όπως αναφέρθηκε, η εντολή break έχει ιδιαίτερη σημασία μέσα σε ένα switch μπλοκ. Η ύπαρξή της οριοθετεί τις εντολές που εκτελούνται για κάθε ετικέτα, και εμπλέκεται κυκλωματικά με τρόπο διαφορετικό από αυτόν που είχε περιγραφεί στην εικόνα 34. Για ευκολία, ο αλγόριθμος ελέγχει την ύπαρξη της εντολής break στο πρώτο επίπεδο εντολών που βρίσκονται στο switch μπλοκ, και όχι μέσα σε υπο-μπλόκ εντολών. Έτσι, οι εντολές break για τη διαφυγή από ένα switch μπλοκ θα πρέπει να μη γράφονται σε φωλιασμένα μπλοκ άλλων εντολών (π.χ. στο μπλοκ εντολών μιας εντολής if).

Οι παραστάσεις εντολής.

Το μόνο είδος εντολής που απομένει προς εξέταση είναι η *παράσταση εντολής* (expression statement). Η σύνταξη μιας εντολής τέτοιου είδους είναι προφανής:

```
<expression> ;
```

Οι παραστάσεις εντολής είναι παραστάσεις που μπορούν να εμφανιστούν στη θέση μιας εντολής (αν και συμμετέχουν και σε άλλα σημεία του προγράμματος, και συγκε-

κριμένα στην εντολή for). Μια παράσταση εντολής μπορεί να είναι μόνο κάτι από τα παρακάτω:

- Μια ανάθεση (assignment), με χρήση κάποιου τελεστή ανάθεσης.
- Μια μοναδιαία αύξηση (ή μια μοναδιαία μείωση), με χρήση των προθεματικών ή μεταθεματικών τελεστών αύξησης (ή μείωσης).
- Μία κλήση συνάρτησης (function call).

Η κυκλωματική αντιστοιχία της παράστασης εντολής εξαρτάται από το είδος της παράστασης, αλλά μακροσκοπικά δε διαφέρει από μια κανονική εντολή, όπως δείχνει και η εικόνα 28. Περισσότερα αναφέρονται παρακάτω, στις ενότητες των εκάστοτε τελεστών.

Το υπόλοιπο μέρος αυτού του κεφαλαίου ασχολείται με την εξέταση των τελεστών και γενικά των στοιχείων που συμμετέχουν στις παραστάσεις της γλώσσας, ώστε να αποσαφηνιστεί πλήρως η σύνταξη και ο τρόπος αποτίμησης των παραστάσεων που μπορούν να εμφανίζονται μέσα σε ένα πρόγραμμα.

Ο βασικός τελεστής ανάθεσης. Οι άλφα-τιμές. Ο τελεστής αποαναφοράς.

Ο βασικός τελεστής ανάθεσης συντάσσεται ως εξής:

```
<l-value> = <expression>
```

Στα αριστερά του τελεστή ανάθεσης βρίσκεται μια *α-τιμή* (l-value). Ο όρος αυτός χρησιμοποιείται για να περιγράψει εκείνα τα πράγματα που μπορούν να εμφανίζονται στο αριστερό κομμάτι μιας ανάθεσης, δηλαδή εκείνες τις οντότητες στις οποίες μπορεί να ανατεθεί μια τιμή. Στα δεξιά του τελεστή ανάθεσης βρίσκεται μια παράσταση με χρόνο. Η σημασιολογία του τελεστή είναι η εξής: υπολογίζεται το αποτέλεσμα της παράστασης που βρίσκεται στα δεξιά, και ύστερα ανατίθεται στην α-τιμή που βρίσκεται στα αριστερά.

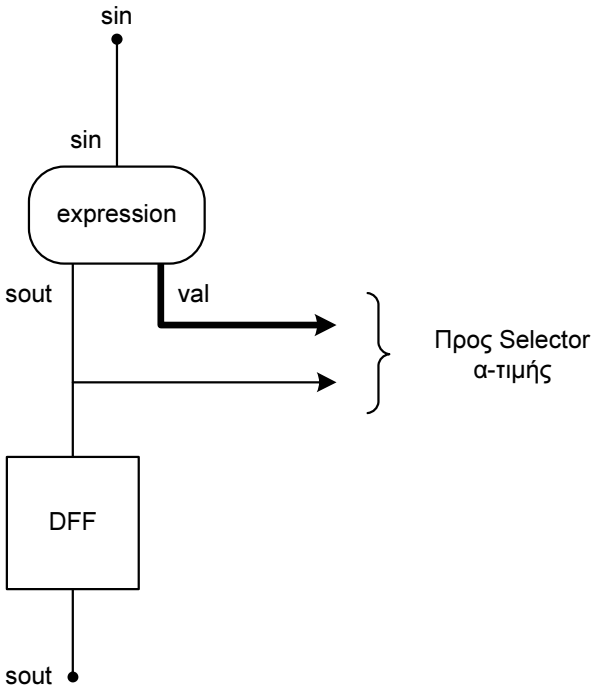
Οι α-τιμές που υποστηρίζονται από τη γλώσσα είναι όλες εκείνες οι αποθηκευτικές μονάδες που έχουν περιγραφεί. Συγκεκριμένα, μια α-τιμή μπορεί να είναι μόνο κάτι από τα παρακάτω:

- Μία απλή καθολική ή μία τοπική μεταβλητή. Στην περίπτωση αυτή, στη θέση της α-τιμής γράφεται απλά το αναγνωριστικό όνομα της μεταβλητής.
- Μία παράμετρος κατ' αξία. Και σε αυτή την περίπτωση η α-τιμή γράφεται με το αναγνωριστικό όνομα της παραμέτρου.
- Μία παράμετρος κατ' αναφορά ή μια καθολική μεταβλητή τύπου δείκτη. Σε αυτή την περίπτωση, επειδή το αναγνωριστικό όνομα θεωρείται σημασιολογικά ως δείκτης, θα πρέπει να γίνει *αποαναφορά* (dereferencing)—γνωστή και ως *έμμεση αναφορά* (indirection). Η πράξη αυτή γίνεται με τη βοήθεια του τελεστή αποαναφοράς '*'. Η α-τιμή προκύπτει με την εφαρμογή του μοναδιαίου αυτού τελεστή πάνω στο όνομα της παραμέτρου, δηλαδή πριν από το αναγνωριστικό (identifier) της μεταβλητής γράφεται ο τελεστής '*'.

Ο τελεστής αποαναφοράς δεν συντίθεται σε κάποιο κύκλωμα, και επομένως δεν απαιτείται να εξεταστεί περαιτέρω με ιδιαίτερο τρόπο. Γενικά, ο τελεστής αυτός εφαρ-

μόζεται σε ορίσματα τύπου δείκτη, και η δράση του περιορίζεται στη μετατροπή του τύπου του ορίσματος, από δείκτη, σε πραγματικό δεδομένο. Στην πράξη, επειδή σε ένα πρόγραμμα οι μόνοι δείκτες που μπορεί να εμφανιστούν είναι οι παράμετροι κατ' αναφορά ή οι καθολικές μεταβλητές τύπου δείκτη, ο τελεστής αποαναφοράς χρησιμοποιείται κυρίως για τη χρήση της τιμής μιας τέτοιας οντότητας (είτε ως κανονική τιμή είτε ως α-τιμή) μέσα σε ένα πρόγραμμα.

Ο βασικός τελεστής ανάθεσης, όπως και οι υπόλοιποι τελεστές ανάθεσης, θεωρείται αποκλειστικά ως παράσταση εντολής. Αυτή η παρατήρηση είναι πολύ σημαντική, διότι απαγορεύει τη συμμετοχή του τελεστή ανάθεσης σε μία παράσταση με ή χωρίς χρόνο. Σε μία παράσταση χωρίς χρόνο θα ήταν ούτως ή άλλως αδύνατο να συμμετέχει μια ανάθεση, αφού για την πραγματοποίηση της ανάθεσης απαιτείται ένας κύκλος ρολογιού. Όμως, όπως θα γίνει αργότερα φανερό, ούτε σε μία παράσταση με χρόνο θα ήταν δυνατό να συμμετέχει, διότι ο χρονισμός είναι τέτοιος που δεν μπορεί να διατηρηθεί η κλασική σημασιολογία του τελεστή.



Εικόνα 38 – Το κύκλωμα του βασικού τελεστή ανάθεσης

Έτσι λοιπόν ο τελεστής φαίνεται εξωτερικά ως μία εντολή. Το κύκλωμα που αντιστοιχεί φαίνεται στην εικόνα 38. Η παράσταση που βρίσκεται στο δεξί μέλος μιας ανάθεσης είναι μια παράσταση με χρόνο. Η τιμή της υπολογίζεται ξεκινώντας με κατάσταση sin την κατάσταση εισόδου της εντολής, και εξάγεται η κατάσταση sout και η τιμή val. Η τιμή αυτή πρέπει να εγγραφεί στην α-τιμή. Για το σκοπό αυτό, το σήμα val υφίσταται μετατροπή τύπου (όπως περιγράφηκε αναλυτικά σε προηγούμενη παράγραφο), ώστε να είναι συμβατό προς ανάθεση, και κατόπιν το ζεύγος των σημάτων sout και val εισάγεται στον επιλογέα της α-τιμής. Η μετατροπή τύπου (που ουσιαστικά είναι μια αποκοπή ή εισαγωγή bits και δεν απαιτεί ειδικά κυκλώματα) παραλείπεται από την εικόνα 38. Η α-τιμή, που είναι είτε μια καθολική ή τοπική μεταβλητή, είτε μια παράμετρος της τρέχουσας συνάρτησης, θα περιέχει κατά τα γνωστά έναν επιλογέα στον οποίο θα προσαρτηθούν τα νέα σήματα. Στην περίπτωση μιας καθολικής

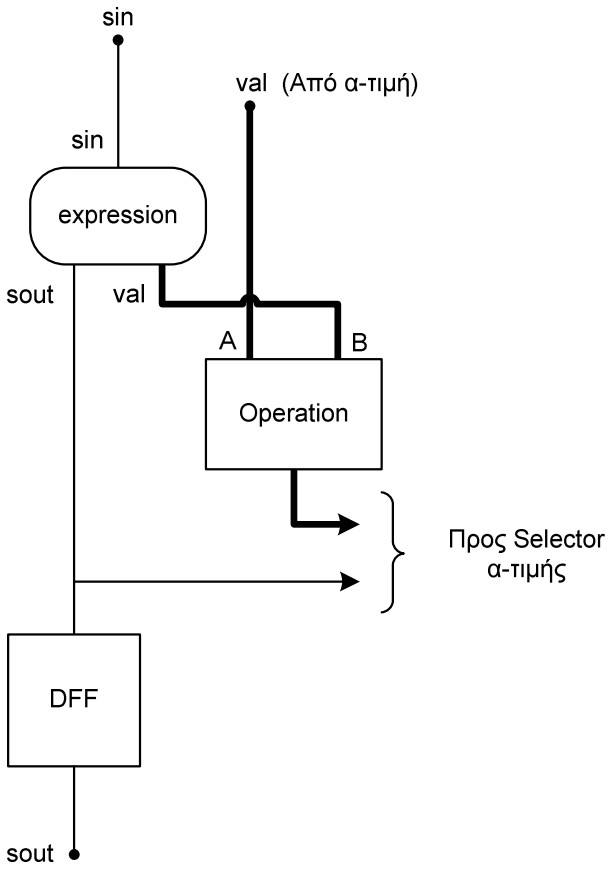
μεταβλητής (εικόνα 12 ή εικόνα 20), μιας τοπικής μεταβλητής (εικόνα 23) ή μιας παραμέτρου κατ' αξία (εικόνα 17), ο επιλογέας αυτός είναι μοναδικός. Στην περίπτωση μιας παραμέτρου κατ' αναφορά (εικόνα 19), που υπάρχουν δύο επιλογείς, ο αντίστοιχος επιλογέας είναι αυτός των δεδομένων εξόδου (DataOut Selector). Επειδή ο τελεστής ανάθεσης είναι μια εντολή, οφείλει να εξάγει μια τελική κατάσταση sout στην οποία θεωρείται ότι έχει τελειώσει η ανάθεση. Η τελική αυτή κατάσταση προκύπτει από την κατάσταση sout στην οποία είναι διαθέσιμη η παράσταση, με την εισαγωγή ενός DFF για την καθυστέρηση ενός κύκλου ρολογιού. Έτσι, αφού αποτιμηθεί η παράσταση, παρέρχεται ένας κύκλος ρολογιού στον οποίο συμβαίνει καθαυτή η εγγραφή των δεδομένων μέσω του επιλογέα πίσω στην α-τιμή, και ενεργοποιείται η τελική έξοδος sout σηματοδοτώντας την ολοκλήρωση της ανάθεσης.

Οι υπόλοιποι τελεστές ανάθεσης. Η μετατροπή κοινού τύπου.

Εκτός από τον βασικό τελεστή ανάθεσης, υποστηρίζονται και οι εξής γνωστοί τελεστές ανάθεσης της C:

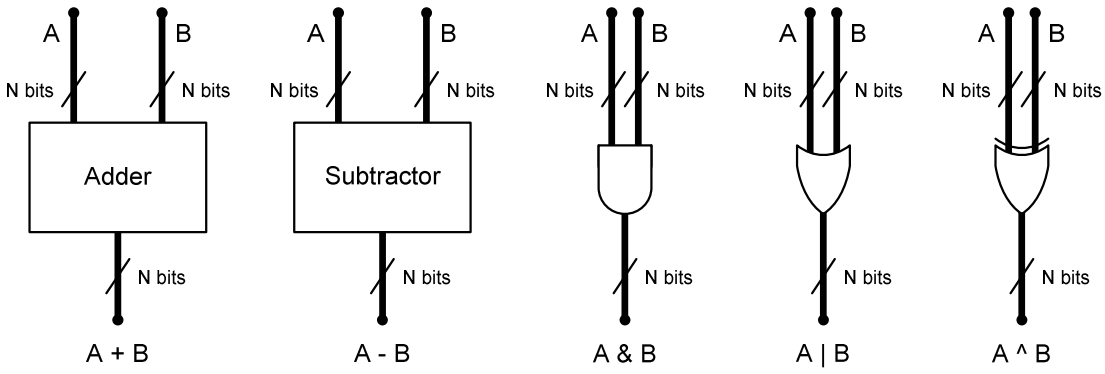
- Ο τελεστής ανάθεσης με πρόσθεση '+='.
- Ο τελεστής ανάθεσης με αφαίρεση '–='.
- Ο τελεστής ανάθεσης με bitwise AND '&='.
- Ο τελεστής ανάθεσης με bitwise OR '|='.
- Ο τελεστής ανάθεσης με bitwise XOR '^='.
- Ο τελεστής ανάθεσης με αριστερή ολίσθηση '<<='.
- Ο τελεστής ανάθεσης με δεξιά ολίσθηση '<<='.

Οι τελεστές αυτοί διατηρούν τη συμπεριφορά του βασικού τελεστή ανάθεσης, δηλαδή συντάσσονται με μία α-τιμή στο αριστερό μέλος και με μία παράσταση με χρόνο στο δεξί, ενώ απαντώνται σε ένα πρόγραμμα μόνο ως παραστάσεις εντολής. Το κύκλωμα τους όμως είναι διαφορετικό, καθώς πλέον η ανάθεση περιλαμβάνει και μια πράξη που γίνεται πάνω στην α-τιμή. Για το σκοπό αυτό το κύκλωμα ενός τελεστή ανάθεσης αυτού του είδους χρειάζεται και την τρέχουσα τιμή των δεδομένων που περιέχει η α-τιμή προκειμένου να υπολογίσει τη νέα τιμή προς ανάθεση. Στην περίπτωση που η α-τιμή είναι μια απλή καθολική μεταβλητή, μια τοπική μεταβλητή ή μια παράμετρος κατ' αναφορά, η τρέχουσα τιμή λαμβάνεται από την έξοδο val του αντίστοιχου καταχωρητή. Αν η α-τιμή είναι μια παράμετρος κατ' αναφορά, η τιμή λαμβάνεται από την έξοδο val του επιλογέα πηγής (Source Selector) της παραμέτρου, ενώ αν είναι καθολική μεταβλητή τύπου δείκτη λαμβάνεται από το εξωτερικό σήμα val. Γίνεται λοιπόν φανερό πόσο εύκολη κάνει ο τρόπος σχεδίασης των δεικτών την ενιαία αντιμετώπισή τους σε σχέση με τα άλλου είδους δεδομένα: όποτε πρέπει να γραφεί η τιμή μιας παραμέτρου κατ' αναφορά, υπάρχει ο επιλογέας δεδομένων εξόδου ο οποίος λειτουργεί σαν τους επιλογείς των κανονικών μεταβλητών, ενώ όποτε πρέπει να γίνει ανάγνωση της τιμής της, υπάρχει το σήμα val του επιλογέα τιμής που λειτουργεί σαν το σήμα val των καταχωρητών των κανονικών μεταβλητών. Ανάλογα λειτουργούν και οι καθολικοί δείκτες. Σε κάθε περίπτωση, ό,τι είδους και να είναι η α-τιμή, η έξοδος val αυτής είναι το πρώτο νοητό όρισμα του τελεστή πράξης της ανάθεσης, ενώ το δεύτερο όρισμα είναι η παράσταση με την οποία συντάσσεται. Τα δύο αυτά ορίσματα θα πρέπει να υποστούν την αντίστοιχη πράξη που ορίζει ο τελεστής, και το προκύπτον αποτέλεσμα να εκχωρηθεί πίσω στην α-τιμή. Ένα αφαιρετικό σχήμα με τη γενική μορφή του κυκλώματος που αντιστοιχεί φαίνεται στην εικόνα 39.



Εικόνα 39 – Το κύκλωμα των υπόλοιπων τελεστών ανάθεσης

Όπως είναι φανερό, όπως και στο βασικό τελεστή ανάθεσης, γίνεται αρχικά η αποτίμηση της παράστασης με χρόνο που βρίσκεται στο δεξιό μέλος. Η προκύπτουσα τιμή B, μαζί με την τιμή val A που προέρχεται από την α-τιμή, υφίστανται την εκάστοτε πράξη, και το αποτέλεσμα οδηγείται πίσω στον επιλογέα της α-τιμής. Το τελικό DFF στην έξοδο φροντίζει ώστε να ενεργοποιηθεί το σήμα εξόδου αφού εγγραφεί το αποτέλεσμα στην α-τιμή. Το σχήμα της εικόνας 39 περιλαμβάνει ένα κύκλωμα πράξης, το οποίο έχει συμβολιστεί ως 'Operation'. Ο κάθε τελεστής ανάθεσης απαιτεί και ένα διαφορετικό κύκλωμα για την πραγματοποίηση της αντίστοιχής του πράξης. Τα κυκλώματα πέντε βασικών πράξεων φαίνονται στην εικόνα 40.



Εικόνα 40 – Τα κυκλώματα πέντε βασικών πράξεων

Συνεπώς, οι πρώτοι πέντε τελεστές ανάθεσης (από τη λίστα που δόθηκε αρχικά) υλοποιούνται άμεσα αντικαθιστώντας στο κύκλωμα ‘Operation’ της εικόνας 39 το αντίστοιχο κύκλωμα της εικόνας 40. Η εικόνα 40 παρουσιάζει κάποια συνδυαστικά κυκλώματα για την υλοποίηση των πράξεων, και όχι τα συνολικά κυκλώματα των δυαδικών τελεστών ‘+’, ‘&’, κλπ., οι οποίοι και θα εξεταστούν σε άλλη παράγραφο διότι δεν συντάσσονται ούτε συμπεριφέρονται ακριβώς το ίδιο με τους αντίστοιχους τελεστές ανάθεσης. Ο αθροιστής (adder) και ο αφαιρέτης (subtractor) λειτουργούν με αριθμητική συμπληρώματος ως προς δύο για τους προσημασμένους αριθμούς. Επίσης, οι πράξεις bitwise AND, bitwise OR και bitwise XOR πραγματοποιούνται, όπως λέει και η ονομασία τους, bit-προς-bit. Πρέπει να τονιστεί σε αυτό το σημείο κάτι σημαντικό που δεν έχει αναφερθεί μέχρι στιγμής. Οι πέντε συγκεκριμένες πράξεις έχουν την ιδιότητα να λαμβάνουν ομοειδή ορίσματα (σε αντίθεση με την πράξη της ολίσθησης). Έτσι, τα δύο ορίσματά τους πρέπει να είναι του ιδίου τύπου. Αν αυτό δε συμβαίνει, εφαρμόζεται μια συγκεκριμένη σύμβαση για τη μετατροπή των ορισμάτων σε κοινό τύπο. Αυτή η διαδικασία, που ονομάζεται *μετατροπή κοινού τύπου* (common type conversion), ορίζει έναν καινούριο τύπο και ύστερα μετατρέπει τα δύο ορίσματα στο νέο αυτό κοινό τύπο σύμφωνα με τους κανόνες της απλής *μετατροπής τύπου* (που περιγράφηκε νωρίτερα στα πλαίσια της εντολής return). Ο νέος κοινός τύπος ορίζεται ως εξής:

- Αν αμφότερα τα ορίσματα είναι μη προσημασμένου (unsigned) τύπου, τότε ο νέος κοινός τύπος ορίζεται απρόσημος, με μέγεθος το μεγαλύτερο από τα μεγέθη των τύπων των δύο ορισμάτων.
- Αν έστω και ένα από τα ορίσματα είναι προσημασμένου (signed) τύπου, τότε ο νέος κοινός τύπος ορίζεται κι αυτός προσημασμένος, με μέγεθος πάλι το μεγαλύτερο από τα μεγέθη των τύπων των δύο ορισμάτων.

Στην πράξη, η διαδικασία αυτή στην πρώτη περίπτωση θα πειράξει μόνο το όρισμα με το μικρότερο μέγεθος προσαρμόζοντάς το στο μεγαλύτερο μέγεθος με την προσάρτηση μηδενικών bits στο αριστερότερο (περισσότερο σημαντικό) κομμάτι του. Η δεύτερη περίπτωση είναι πολυπλοκότερη: το μικρότερο όρισμα θα επεκταθεί είτε με μηδενικά είτε με το ψηφίο προσήμου (ανάλογα με το αν το μικρότερο αυτό όρισμα είναι προσημασμένου τύπου) μέχρι του μεγέθους του μεγαλύτερου ορίσματος, και ύστερα τα δύο ορίσματα θα χαρακτηριστούν ως προσημασμένα. Η μετατροπή κοινού τύπου δεν εμφανίζεται στην εικόνα 40 για λόγους απλότητας, και γενικά δεν είναι μια διαδικασία που ακολουθείται σε όλες τις πράξεις, καθώς ορισμένες απαιτούν ειδικότερους χειρισμούς.

Το αποτέλεσμα καθεμίας από τις πέντε πράξεις που φαίνονται στην εικόνα 40 διατηρεί τον κοινό τύπο που έχουν τα εκάστοτε ορίσματά, όπως αυτός προέκυψε από τη διαδικασία μετατροπής κοινού τύπου. Ο τύπος αυτός όμως ενδέχεται να μην είναι συμβατός για ανάθεση με τον τύπο της α-τιμής, ακριβώς διότι η παράσταση του τελεστή ανάθεσης μπορεί να έχει μεγαλύτερο μέγεθος, το οποίο προκαλεί ένα εξίσου μεγάλο μέγεθος για τον κοινό τύπο. Πράγματι, αν σε ένα πρόγραμμα μια μεταβλητή A έχει μικρότερο μέγεθος από μια μεταβλητή B, τότε η εντολή ανάθεσης ‘A &= B’ θα εκτελέσει ενδιάμεσα την πράξη ‘A & B’ στο μέγεθος της μεγαλύτερης μεταβλητής B. Έτσι, το τελικό αποτέλεσμα της πράξης ‘Operation’ της εικόνας 39 θα πρέπει να υποστεί ακόμη μια μετατροπή τύπου, με νέο τύπο αυτόν της α-τιμής, ώστε να είναι συμβατή η ανάθεση.

Οι δύο εναπομείναντες τελεστές ανάθεσης, που εμπλέκουν ολίσθηση, δεν έχουν την ιδιότητα των ομοειδών ορισμάτων και συνεπώς τα ορίσματά των πράξεών τους δεν υφίστανται μετατροπή κοινού τύπου. Το βασικό όρισμα μιας πράξης ολίσθησης, ο τύπος του οποίου διατηρείται και στο αποτέλεσμά, είναι στην περίπτωση της ανάθεσης η α-τιμή (και γενικότερα το αριστερό όρισμα της πράξης ολίσθησης). Μια ολίσθηση είναι πολύ εύκολο να υλοποιηθεί κυκλωματικά εφόσον πρόκειται για σταθερό αριθμό θέσεων ολίσθησης. Συνεπώς, ο μεταγλωττιστής οφείλει να ελέγξει αν το δεύτερο όρισμα μιας πράξης ολίσθησης είναι μια σταθερά, και στην περίπτωση αυτή να εισάγει ένα απλό κύκλωμα ολίσθησης (που απλά συνδέει τα σύρματα της εισόδου στις κατάλληλες ολισθημένες θέσεις στην έξοδο). Στην περίπτωση που το όρισμα είναι μεταβλητό, και η τιμή του δεν είναι γνωστή κατά τη φάση της μεταγλώττισης, προτιμάται ένα συνδυαστικό κύκλωμα που λέγεται *ολισθητής βαρελιού* (barrel shifter). Το κύκλωμα αυτό παράγει κατά κάποιο τρόπο όλα τα δυνατά αποτελέσματα ολίσθησης και επιλέγει το σωστό με βάση το μεταβλητό όρισμα. Η δομή και η λειτουργία του αναφέρονται στη βιβλιογραφία της σύνθεσης κυκλωμάτων και έτσι δεν θα αναλυθεί περαιτέρω. Ας σημειωθεί επίσης ότι η πράξη της αριστερής ολίσθησης γεμίζει τις δεξιές θέσεις με μηδενικά, ενώ η πράξη της δεξιάς ολίσθησης γίνεται με αριθμητικό τρόπο, γεμίζοντας τις αριστερές θέσεις είτε με το πρόσημο των δεδομένων αν αυτά είναι προσημασμένα, είτε με μηδενικά αν είναι απρόσημα (ώστε να ισοδυναμεί πάντοτε με μια διαίρεση με το δύο). Το αποτέλεσμα έχει τόσα bits όσα και η α-τιμή (και γενικότερα το αριστερό όρισμα μιας πράξης ολίσθησης) και έτσι δεν απαιτεί άλλη μετατροπή.

Οι τελεστές αύξησης και μείωσης.

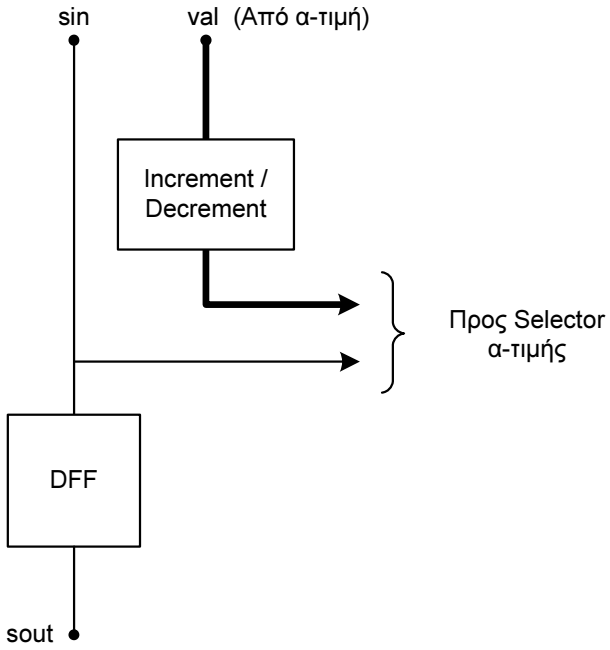
Οι τελεστές μοναδιαίας αύξησης και μείωσης χρησιμοποιούνται για την αύξηση ή μείωση κατά ένα μιας α-τιμής. Η C παρέχει τους τελεστές αυτούς τόσο σε προθεματική όσο και σε μεταθεματική μορφή, σύμφωνα με τις παρακάτω μορφές σύνταξης:

```
<l-value> ++
<l-value> --

++ <l-value>
-- <l-value>
```

Επειδή ακριβώς για λόγους σχεδίασης οι τελεστές αύξησης και μείωσης θεωρούνται αποκλειστικά παραστάσεις εντολής, δεν μπορούν να συμμετέχουν σε παραστάσεις με ή χωρίς χρόνο, και έτσι δεν έχει κάποια ιδιαίτερη σημασία η χρήση της προθεματικής έναντι της μεταθεματικής μορφής. Πράγματι, κάθε φορά που απαντάται ένας τελεστής αύξησης ή μείωσης σε ένα πρόγραμμα, θα πρέπει να είναι μόνος του (μαζί μόνο με μια α-τιμή, χωρίς άλλες πράξεις) σε μία παράσταση εντολής, και επομένως η προτεραιότητα του (που είναι διαφορετική για την προθεματική και τη μεταθεματική μορφή) δε θα παίζει ρόλο. Ο λόγος για τον οποίο οι τελεστές αύξησης και μείωσης δε συμμετέχουν σε άλλου είδους παραστάσεις είναι ο ίδιος για τον οποίο αποκλείονται και οι αναθέσεις από μια παράσταση με χρόνο, και σχετίζεται με την αδυναμία διατήρησης της σημασιολογίας τους. Περισσότερα για αυτό θα αναφερθούν σε επόμενη παράγραφο.

Το κύκλωμα που αντιστοιχεί στους τελεστές αύξησης και μείωσης φαίνεται στην εικόνα 41. Τα τρέχοντα δεδομένα που περιέχει η α-τιμή λαμβάνονται από το σήμα val και υφίστανται μια αύξηση ή μείωση μέσω αντίστοιχου συνδυαστικού κυκλώματος,

που στο σχήμα έχει ονομαστεί ‘Increment / Decrement’. Το κύκλωμα αυτό δεν είναι στην πράξη παρά ένας αθροιστής ή ένας αφαιρέτης, του οποίου το δεύτερο όρισμα είναι η μονάδα. Το αυξημένο ή μειωμένο αποτέλεσμα, μαζί με την αντίστοιχη κατάσταση εισόδου sin (η οποία υπάρχει μιας και ο τελεστής συμπεριφέρεται ως παράσταση εντολής) οδηγούνται στον επιλογέα της α-τιμής, σύμφωνα και με όσα έχουν ήδη αναφερθεί για τις αναθέσεις. Είναι πρόδηλο ότι δεν απαιτείται κάποια επιπλέον μετατροπή τύπου. Το τελικό DFF στην έξοδο εισάγει την απαραίτητη καθυστέρηση στο μονοπάτι ελέγχου.



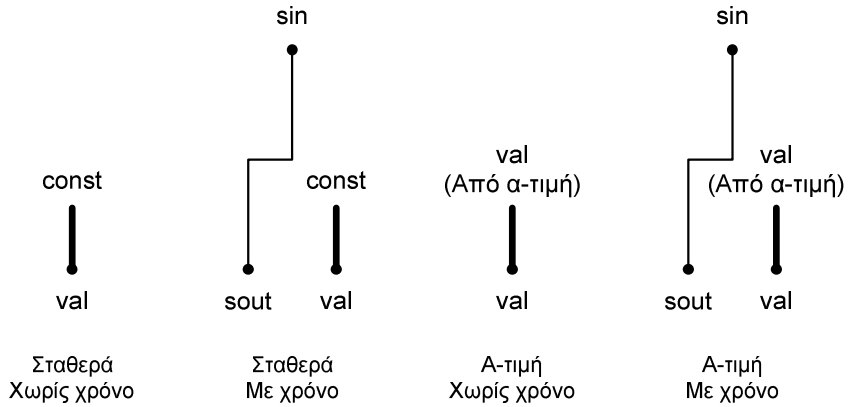
Εικόνα 41 – Το κύκλωμα των τελεστών αύξησης και μείωσης

Οι μόνες παραστάσεις εντολής που δεν έχουν εξεταστεί μέχρι στιγμής είναι η κλήσεις προς συναρτήσεις. Επειδή μια κλήση προς συνάρτηση μπορεί να συμμετέχει και ως παράσταση με χρόνο, εκτός από παράσταση εντολής, η εξέτασή των κλήσεων θα γίνει αργότερα. Στη συνέχεια αναλύονται οι τελεστές και τα υπόλοιπα στοιχεία που απαρτίζουν τις παραστάσεις με ή χωρίς χρόνο.

Οι άλφα-τιμές και οι σταθερές ως παραστάσεις με ή χωρίς χρόνο.

Οι παραστάσεις χωρίς χρόνο, που γενικά κάνουν την εμφάνισή τους σε εντολές όπως η if, η while, η switch, κ.α., καθώς και οι παραστάσεις με χρόνο, που εμφανίζονται μετά από μια εντολή return ή στο δεξί μέλος μιας ανάθεσης, κλπ., πρωτοπαρουσιάστηκαν από μακροσκοπική άποψη στην εικόνα 28. Στη συνέχεια, για ευκολία, θα γίνεται αναφορά και στα δύο είδη με το γενικό όρο *παραστάσεις*. Τα πιο συχνά εμφανιζόμενα στοιχεία μέσα σε μια παράσταση είναι οι σταθερές και οι α-τιμές. Οι σταθερές, που έχουν περιγραφεί αναλυτικά ως προς τη σύνταξη και τον τρόπο αναπαράστασης σε προηγούμενη παράγραφο, εισάγουν ένα αντίστοιχο δυαδικό διάνυσμα val του οποίου τα bits έχουν τις συγκεκριμένες τιμές που καθορίζει η σταθερά. Ομοίως, οι α-τιμές, δηλαδή οι μεταβλητές και οι παράμετροι με τη γνωστή σύνταξη που περιγράφηκε στα πλαίσια του τελεστή ανάθεσης, συμμετέχουν δίνοντας την τιμή τους val (π.χ. από τον καταχωρητή, ή για τις παραμέτρους κατ’ αναφορά από την έξοδο του

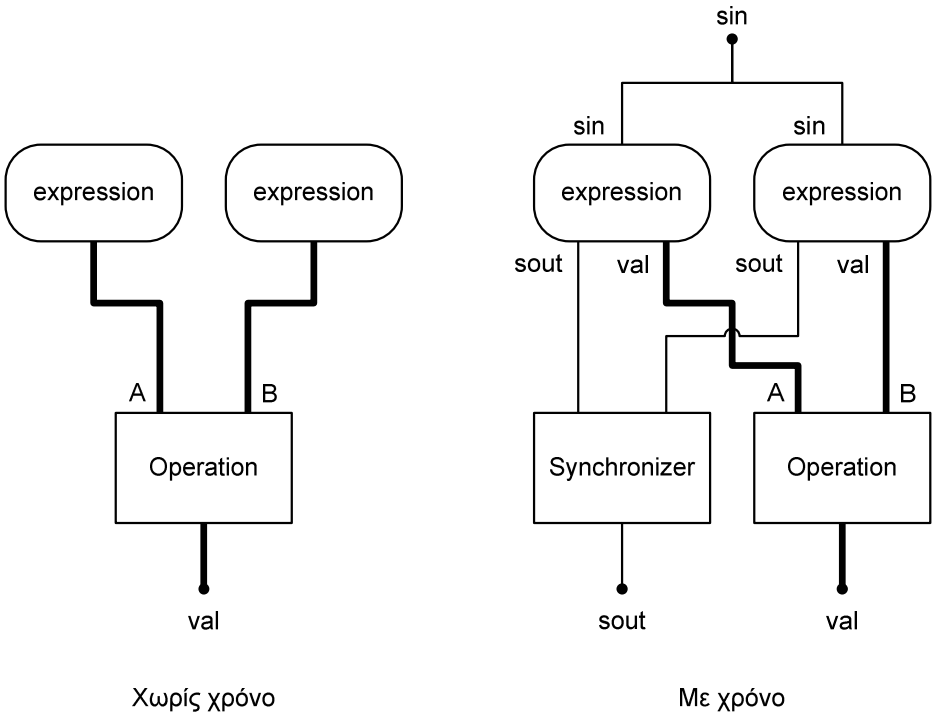
επιλογέα πηγής). Σε καμία περίπτωση δεν απαιτείται μεταβολή του χρόνου για τον υπολογισμό της τιμής εξόδου της παράστασης, και έτσι, στην περίπτωση που η σταθερά ή η α-τιμή συμμετέχουν σε μία παράσταση με χρόνο, η κατάσταση εξόδου sout απλά συνδέεται με την κατάσταση εισόδου sin. Οι διάφορες περιπτώσεις φαίνονται στην εικόνα 42.



Εικόνα 42 – Η συμμετοχή των σταθερών και των α-τιμών στις παραστάσεις

Οι δυαδικοί τελεστές.

Οι δυαδικοί τελεστές της γλώσσας μπορούν να συμμετέχουν εξίσου σε παραστάσεις με ή χωρίς χρόνο. Η τεχνική που χρησιμοποιείται για τη σύνθεσή τους είναι η εισαγωγή αντίστοιχου κυκλώματος που επιτελεί την πράξη τους, και αυτό γίνεται εκ νέου για κάθε εμφάνισή τους μέσα στο πρόγραμμα. Γενικότερα, όπως είχε αναφερθεί σε προηγούμενη παράγραφο σχετικά με τη μορφή του μονοπατιού δεδομένων, δε γίνεται κάποιου είδους διαμοίραση πόρων, παρά κάθε παράσταση εισάγει τα δικά της κυκλώματα για την αποκλειστική επιτέλεση των πράξεών της. Η γενική αναπαράσταση ενός δυαδικού τελεστή φαίνεται στην εικόνα 43.



Εικόνα 43 – Το κύκλωμα της παράστασης ενός δυαδικού τελεστή

Όταν ο τελεστής συμμετέχει σε μια παράσταση χωρίς χρόνο, τότε τα δύο ορίσματά του είναι εξίσου παραστάσεις χωρίς χρόνο. Συνεπώς, δε μένει παρά να εισαχθούν οι δύο τιμές A και B σε ένα συνδυαστικό κύκλωμα 'Operation' που επιτελεί την πράξη του εκάστοτε δυαδικού τελεστή. Οι τυχόν μετατροπές τύπων που μπορεί να συμβούν, ώστε να γίνουν συμβατά τα δύο ορίσματα για την εκτέλεση της πράξης, εξαρτώνται από το είδος του τελεστή. Το αποτέλεσμα val εξάγεται άμεσα από το κύκλωμα της πράξης.

Στην περίπτωση που ο τελεστής συμμετέχει σε μια παράσταση με χρόνο, η διαδικασία αποτίμησης είναι περισσότερο πολύπλοκη. Το δίλημμα που τίθεται είναι ο τρόπος με τον οποίο θα διαδοθεί το σήμα εισόδου sin ώστε να αποτιμηθούν τα δύο ορίσματα του τελεστή, τα οποία είναι εξίσου παραστάσεις με χρόνο. Μια πρώτη σκέψη θα ήταν η κατάσταση εισόδου sin να τροφοδοτήσει την είσοδο sin της παράστασης του αριστερού ορίσματος του τελεστή, η έξοδος sout της οποίας θα μπορούσε να συνδεθεί κατόπιν με την είσοδο του δεξιού ορίσματος του τελεστή. Έτσι τα δύο ορίσματα θα αποτιμούνταν σειριακά, και η τελική κατάσταση εξόδου θα υποδεικνυόταν από την έξοδο sout της παράστασης του δεξιού ορίσματος. Ωστόσο, η τεχνική αυτή δεν υιοθετήθηκε τελικά, διότι θεωρήθηκε ότι είναι προτιμότερο να εφαρμοστεί μια παράλληλη σημασιολογία για την αποτίμηση των ορισμάτων των τελεστών, στο γενικότερο πνεύμα των βελτιώσεων που έχουν γίνει για την εκμετάλλευση των δυνατοτήτων που προσφέρει το υλικό. Επομένως, το σήμα εισόδου sin διαδίδεται ταυτόχρονα στην είσοδο των παραστάσεων των δύο ορισμάτων. Οι έξοδοι sout των παραστάσεων ενεργοποιούνται όταν ολοκληρωθεί η αποτίμηση τους, και άρα το τελικό σήμα sout απαιτεί έναν συγχρονιστή. Ο συγχρονιστής αυτός θα αναμείνει μέχρις ότου αμφότερα τα δύο ορίσματα αποτιμηθούν και ύστερα θα ενεργοποιήσει το τελικό σήμα εξόδου sout. Πρόκειται για το γνωστό κύκλωμα συγχρονισμού που παρουσιάστηκε αναλυτικότερα στην παράγραφο του παράλληλου μπλοκ εντολών (par). Παρόλο που το μονοπάτι ελέγχου παρουσιάζει αυτή την ιδιομορφία, το μονοπάτι των δεδομένων έχει την ίδια μορφή με αυτή μιας παράστασης χωρίς χρόνο: οι τιμές των παραστάσεων των δύο ορισμάτων εισάγονται σε ένα κύκλωμα πράξης 'Operation' που εξαρτάται από τον τελεστή, και παράγεται η τελική τιμή val. Η τιμή αυτή υπολογίζεται αδιάκοπα, καθώς η πράξη δεν είναι παρά ένα συνδυαστικό κύκλωμα, αλλά η εγκυρότητά της υποδεικνύεται από το σήμα sout: όταν ενεργοποιηθεί, αφού δηλαδή ολοκληρωθεί η αποτίμηση των δύο ορισμάτων, τότε θα είναι έγκυρη η τιμή val.

Η παράλληλη σημασιολογία αποτίμησης έχει το πλεονέκτημα ότι σε μία παράσταση της μορφής 'f() + g()', οι συναρτήσεις f και g θα κληθούν ταυτόχρονα και τα αποτελέσματά τους θα αθροιστούν αμέσως μόλις επιστρέψει η πιο αργή από αυτές. Πρέπει επίσης να τονιστεί ότι η μορφή του μονοπατιού ελέγχου δεν εισάγει ενδιάμεσες αποθηκεύσεις για τα ορίσματα των δυαδικών τελεστών μιας παράστασης με χρόνο. Αυτό εισάγει μια αδυναμία στη διατήρηση της κλασικής σημασιολογίας των τελεστών της ανάθεσης (όπως και των τελεστών αύξησης και μείωσης), οι οποίοι και δε συμμετέχουν σε παραστάσεις με χρόνο. Πράγματι, έστω το παρακάτω κομμάτι κώδικα:

```
i = 2;
j = i + (i = 10);
```

Αυτό το κομμάτι κώδικα δεν είναι έγκυρο υπό τους κανόνες της νέας γλώσσας, καθώς η ανάθεση 'i = 10' δεν επιτρέπεται μέσα σε μία παράσταση. Κάτι τέτοιο απαγορεύεται διότι δεν θα ήταν δυνατό να λάβει η μεταβλητή j τη σωστή τιμή. Πράγματι, η

μεταβλητή j θα έπρεπε κανονικά να λάβει την τιμή 12, δηλαδή 2 από την παλιά τιμή της μεταβλητής i, συν 10 από τη νέα τιμή της μεταβλητής i, της οποίας η ανάθεση συμβαίνει αφού έχει υπολογιστεί ο αριθμός 2 για το πρώτο όρισμα. Ωστόσο, στην πράξη κάτι τέτοιο δεν γίνεται χωρίς αποθήκευση του ενδιάμεσου αποτελέσματος: αν επιτρεπόταν να εμφανιστεί μια τέτοια παράσταση σε ένα πρόγραμμα, ο αθροιστής θα λάμβανε ως είσοδο για το αριστερό όρισμα την έξοδο val της μεταβλητής i, η οποία σε κάθε περίπτωση (είτε με παράλληλη είτε με σειριακή σημασιολογία εκτέλεσης) θα είχε ήδη λάβει την τιμή 10 (αφού πρέπει να αποτιμηθεί και το δεξί όρισμα) κατά την άθροιση, και έτσι θα προέκυπτε ως αποτέλεσμα ο αριθμός 20. Άρα, εφόσον σχεδιαστικά έχει γίνει η επιλογή να μην εισάγονται ενδιάμεσες θέσεις αποθήκευσης (που στην προκειμένη περίπτωση θα μπορούσαν να αποθηκεύσουν την τιμή 2 ώστε ακόμα και με την αλλαγή της μεταβλητής i να υπολογιστεί το σωστό αποτέλεσμα), οι τελεστές ανάθεσης (καθώς και αύξησης ή μείωσης) αποκλείονται από τις παραστάσεις, και επιτρέπονται μόνο σε παραστάσεις εντολής.

Οι πέντε βασικοί δυαδικοί τελεστές (πρόσθεσης, αφαίρεσης, bitwise OR, bitwise AND και bitwise XOR).

Οι πέντε βασικοί δυαδικοί τελεστές (πρόσθεσης, αφαίρεσης, bitwise OR, bitwise AND και bitwise XOR), των οποίων οι πράξεις έχουν ήδη παρουσιαστεί στους αντίστοιχους τελεστές ανάθεσης, έχουν την παρακάτω σύνταξη:

```
<expressionA> + <expressionB>
<expressionA> - <expressionB>
<expressionA> | <expressionB>
<expressionA> & <expressionB>
<expressionA> ^ <expressionB>
```

Οι τελεστές αυτοί εισάγουν το κύκλωμα της εικόνας 43, ενώ η πράξη ‘Operation’ περιλαμβάνει τα κυκλώματα που παρουσιάστηκαν στην εικόνα 40. Επειδή δέχονται ομοειδή ορίσματα, θα πρέπει πρώτα να πραγματοποιήσουν τη γνωστή διαδικασία της μετατροπής κοινού τύπου. Κατά τα άλλα, οι πράξεις γίνονται με το γνωστό τρόπο: επειδή χρησιμοποιείται παράσταση συμπληρώματος ως προς δύο για τους προσημασμένους αριθμούς, ο τρόπος λειτουργίας του αθροιστή (όπως και του αφαιρέτη) είναι κοινός για όλους τους δυνατούς τύπους δεδομένων εισόδου. Οι τελεστές bitwise OR, bitwise AND και bitwise XOR επενεργούν bit-προς-bit στα ορίσματά τους, και δεν πρέπει να λησμονείται το γεγονός ότι η μετατροπή κοινού τύπου καθορίζει και τον τύπο εξόδου της πράξης. Έτσι, παρότι οι τελεστές bitwise OR, bitwise AND και bitwise XOR δεν είναι αμιγώς αριθμητικοί, εν τούτοις οι μετατροπή κοινού τύπου προσδίδει μια αριθμητική σημασία. Για παράδειγμα, έστω ο παρακάτω κώδικας:

```
char c = "FF";
unsigned int d;

d = c ^ 1;
```

Στον κώδικα αυτό, η δήλωση της μεταβλητής c (όπως έχει αναφερθεί στην αντίστοιχη παράγραφο που σχολιάζονται οι τύποι δεδομένων της γλώσσας) γίνεται με τον προσημασμένο τύπο signed bit<8>. Ο δυαδικός τελεστής του bitwise XOR παράγει τον κοινό τύπο signed bit<8> διότι το δεύτερο όρισμα είναι της μορφής unsigned bit<1>. Έτσι το αποτέλεσμα της πράξης ‘c ^ 1’, είναι το δυαδικό διάνυσμα 11111110, και έ-

χει τον τύπο signed bit<8>. Το αποτέλεσμα αυτό, επειδή είναι προσημασμένο, υφίσταται επέκταση προσήμου πριν εκχωρηθεί στην μεταβλητή d (η οποία είναι του τύπου unsigned bit<32>). Έτσι, η μεταβλητή d θα αποκτήσει τελικά τη δεκαεξαδική τιμή 'FFFFFFFE' (την ίδια συμπεριφορά εξάλλου θα παρουσίαζε ο κώδικας αν είχε μεταγλωττιστεί π.χ. με τον gcc).

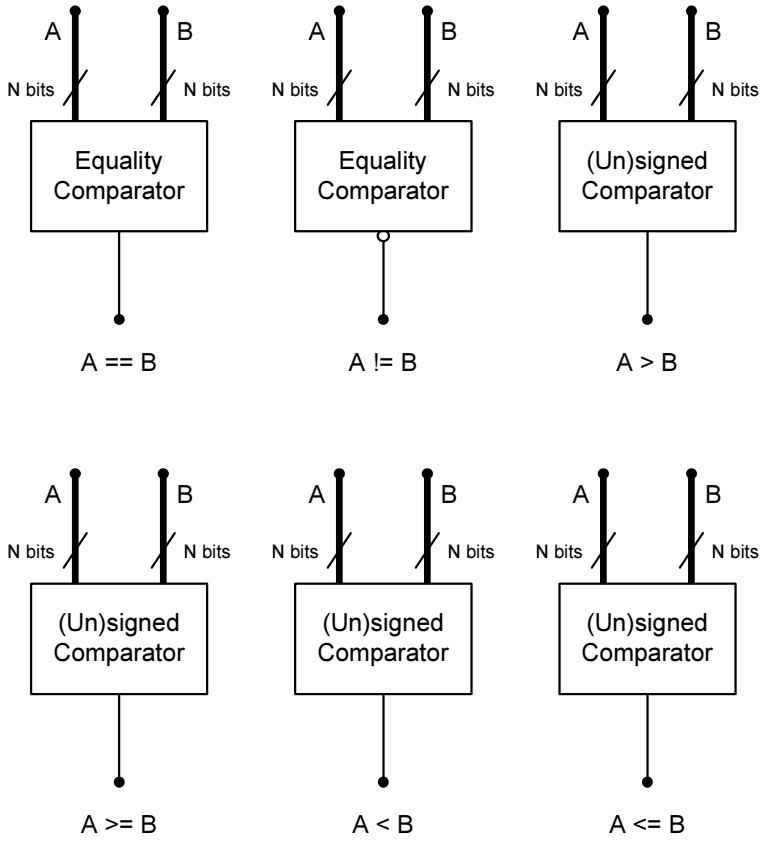
Οι δυαδικοί τελεστές σύγκρισης.

Οι γνωστοί δυαδικοί τελεστές σύγκρισης της C δεν θα μπορούσαν φυσικά να λείπουν. Η σύνταξή τους φαίνεται παρακάτω:

```
<expressionA> == <expressionB>
<expressionA> != <expressionB>
<expressionA> > <expressionB>
<expressionA> >= <expressionB>
<expressionA> < <expressionB>
<expressionA> <= <expressionB>
```

Οι πρώτοι δύο τελεστές ελέγχουν αν τα ορίσματά τους είναι ίσα ή άνισα αντιστοίχως. Ο έλεγχος αυτός των ορισμάτων γίνεται bit-προς-bit, αφού όμως πρώτα έχουν υποστεί μετατροπή κοινού τύπου, ώστε να συγκριθούν δύο διανύσματα του ιδίου τύπου. Για τους υπόλοιπους τέσσερεις τελεστές, που πρέπει να αποφανθούν συν τοις άλλοις αν κάποιο όρισμα είναι μεγαλύτερο ή μικρότερο σε σύγκριση με το άλλο, η διαδικασία της μετατροπής κοινού τύπου έχει ουσιαστικότερη σημασία καθώς καθορίζει αν η σύγκριση θα είναι προσημασμένη ή όχι. Σε μία προσημασμένη σύγκριση τα ορίσματα συγκρίνονται με βάση την τιμή που έχουν θεωρούμενα στην παράσταση συμπληρώματος ως προς 2, και άρα μπορούν γενικά να προκύψουν διαφορετικά αποτελέσματα από μια απρόσημη σύγκριση.

Τα κυκλώματα που παράγονται είναι τα κυκλώματα δυαδικών τελεστών της εικόνας 43, με την πράξη 'Operation' της σύγκρισης να επιτελείται από συγκριτές ισότητας ή ανισότητας, προσημασμένους ή απρόσημους, ανάλογα με τον κοινό τύπο των παραστάσεων των ορισμάτων. Οι συγκριτές αυτοί φαίνονται για τυπικούς περισσότερο λόγους στην εικόνα 44. Μία πολύ σημαντική παρατήρηση που πρέπει να αναφερθεί είναι ότι το αποτέλεσμα των τελεστών σύγκρισης είναι του τύπου signed bit, δηλαδή ένα bit το οποίο είναι ενεργό όταν η σύγκριση είναι αληθής, και το οποίο θεωρείται προσημασμένο ώστε να υφίσταται επέκταση προσήμου. Ο τύπος αυτός χρησιμοποιείται γενικότερα ως τύπος εξόδου κάθε λογικής παράστασης (boolean expression), πράγμα που θα γίνει περισσότερο φανερό στους λογικούς τελεστές.



Εικόνα 44 – Τα κυκλώματα των πράξεων σύγκρισης

Οι τελεστές boolean AND και boolean OR.

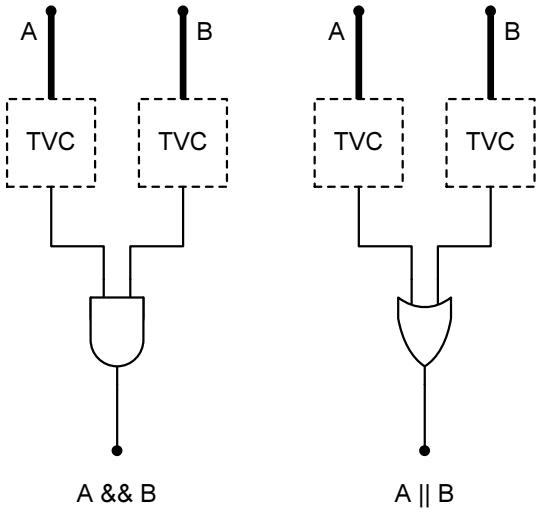
Οι λογικοί τελεστές boolean AND και boolean OR χρησιμοποιούνται για τη σύζευξη ή διάζευξη λογικών παραστάσεων. Η σύνταξή τους είναι η εξής:

```
<expressionA> && <expressionB>
<expressionA> || <expressionB>
```

Οι τελεστές αυτοί επιδρούν πάνω σε λογικές παραστάσεις, δηλαδή τιμές αληθείας που, όπως αναφέρθηκε πρωτύτερα, έχουν τον τύπο signed bit και είναι ενεργές όταν η αντίστοιχη παράσταση είναι αληθής. Σε περίπτωση που η παράσταση κάποιου ορίσματος δεν είναι ήδη σε αυτή τη μορφή, δηλαδή έχει μέγεθος μεγαλύτερο από 1 bit, τότε πρέπει να προηγηθεί η διαδικασία της μετατροπής σε τιμή αληθείας (truth value conversion – TVC) που παρουσιάστηκε στην εικόνα 29. Η έξοδος των τελεστών είναι ομοίως μια λογική τιμή του τύπου signed bit.

Οι λογικοί τελεστές συμπεριφέρονται όπως όλοι οι δυαδικοί τελεστές, δηλαδή χρησιμοποιούν το κύκλωμα της εικόνας 43. Πρέπει να τονιστεί ότι εξαιτίας του τρόπου αποτίμησης των ορισμάτων, οι λογικοί τελεστές δεν χρησιμοποιούν την τεχνική του βραχυκυκλώματος (short-circuit). Αυτή είναι μια σημαντική διαφορά από την κλασική σημασιολογία της C, αφού μια παράσταση π.χ. ‘f() && g()’ θα καλέσει αμφότερες τις συναρτήσεις f και g (οι οποίες μάλιστα θα ξεκινήσουν ταυτόχρονα), δηλαδή η g θα κληθεί όποιο και να είναι το αποτέλεσμα της f.

Το κύκλωμα ‘Operation’ των αντίστοιχων λογικών πράξεων φαίνεται στην εικόνα 45.



Εικόνα 45 – Τα κυκλώματα των πράξεων boolean AND και boolean OR.

Οι δυαδικοί τελεστές ολίσθησης.

Οι δυαδικοί τελεστές ολίσθησης συντάσσονται ως εξής:

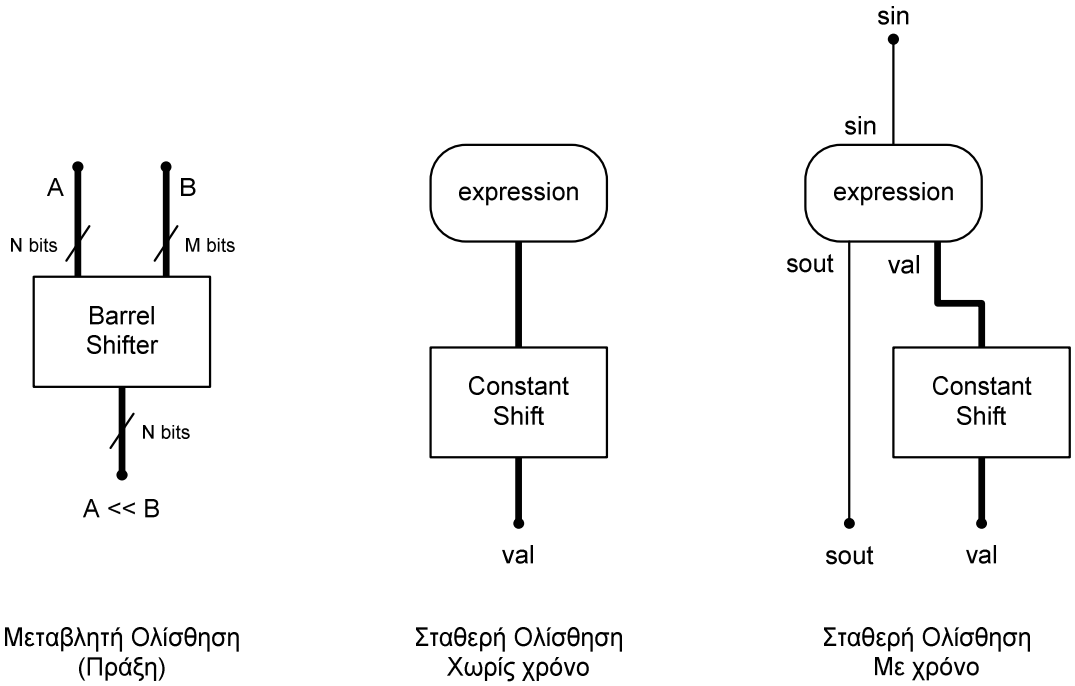
```
<expressionA> << <expressionB>
<expressionA> >> <expressionB>
```

Η ολίσθηση είναι μια πράξη της οποίας τα ορίσματα δεν είναι ομοειδή. Η έξοδος ενός τελεστή ολίσθησης έχει τον ίδιο τύπο με το αριστερό όρισμα, το οποίο και υφίσταται την ολίσθηση. Το πλήθος των θέσεων που θα ολισθήσει καθορίζεται από το δεύτερο όρισμα, που μπορεί γενικά να είναι σταθερό ή μεταβλητό. Όπως αναφέρθηκε και στους αντίστοιχους τελεστές ανάθεσης, μια σταθερή ολίσθηση γίνεται με ευκολότερο τρόπο από μια μεταβλητή, και έτσι ο μεταγλωττιστής πρέπει να είναι σε θέση να αναγνωρίσει την περίπτωση και να παράγει διαφορετικά κυκλώματα.

Η περίπτωση της μεταβλητής ολίσθησης ακολουθεί τις γνωστές τεχνικές των δυαδικών τελεστών. Τα ορίσματα δεν υφίστανται καμία μετατροπή τύπου, και μπορούν γενικά να έχουν διαφορετικά μεγέθη. Το κύκλωμα της εικόνας 43 που εισάγεται για τις ανάγκες του τελεστή, συμπληρώνεται από άποψη πράξης ‘Operation’ με ένα κύκλωμα ολισθητή βαρελιού (barrel shifter), όπως δείχνει και το πρώτο από τα τρία σχήματα της εικόνας 46.

Στην περίπτωση της σταθερής ολίσθησης, που το δεύτερο όρισμα είναι μια σταθερά γνωστή κατά τη διάρκεια της μεταγλώττισης, το κύκλωμα απλοποιείται αρκετά. Η πράξη της ολίσθησης υλοποιείται εύκολα με απλή σύνδεση των bits της εισόδου στις κατάλληλες ολισθημένες θέσεις στην έξοδο, αλλά και γενικότερα το κύκλωμα του τελεστή δεν απαιτεί όλα τα κυκλώματα της εικόνας 43. Έτσι, στην εικόνα 46 έχουν ξανασχεδιαστεί τα κυκλώματα του τελεστή για την περίπτωση της σταθερής ολίσθησης, για τις παραστάσεις με χρόνο και χωρίς χρόνο. Το δεύτερο όρισμα που καθορίζει τις θέσεις της ολίσθησης περιλαμβάνεται ως παράμετρος στο σταθερό ολισθητή ‘Constant Shift’ του σχήματος.

Αν και η αριστερή ολίσθηση προσθέτει μηδενικά, η δεξιά ολίσθηση λειτουργεί με αριθμητικό τρόπο, διατηρώντας το πρόσημο του ορίσματος.



Εικόνα 46 – Τα κυκλώματα της σταθερής και της μεταβλητής ολίσθησης

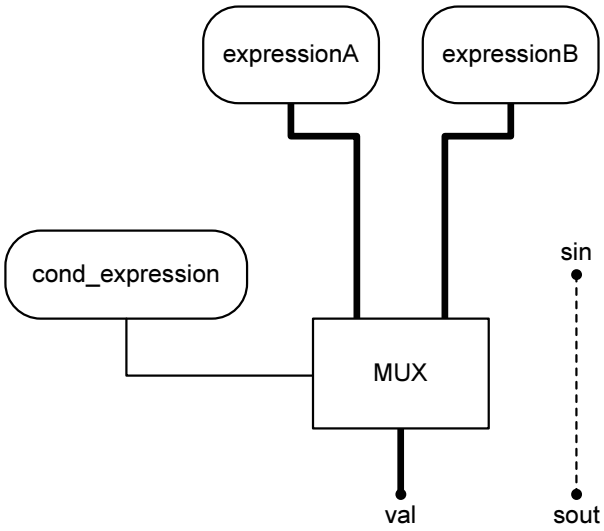
Ο τελεστής παράστασης υπό συνθήκη.

Ο τριαδικός τελεστής '?:', γνωστός και ως τελεστής παράστασης υπό συνθήκη (conditional expression operator), χρησιμοποιείται για την επιλογή μεταξύ των τιμών δυο παραστάσεων, με βάση την τιμή μιας λογικής συνθήκης. Η σύνταξη είναι η κάτωθι:

```
<cond_expression> ? <expressionA> : <expressionB>
```

Η λογική παράσταση 'cond_expression' αποτιμάται, και αν είναι αληθής επιλέγεται η τιμή της παράστασης 'expressionA', διαφορετικά επιλέγεται αυτή της 'expressionB'. Η κανονική σημασιολογία του τελεστή ορίζει ότι από τις δύο παραστάσεις θα αποτιμηθεί μόνο εκείνη που πρέπει (με βάση την τιμή της λογικής συνθήκης). Για λόγους απλότητας αποφασίστηκε λοιπόν οι τρεις παραστάσεις που εμφανίζονται στη σύνταξη του τελεστή να είναι παραστάσεις χωρίς χρόνο, παρότι ο ίδιος ο τελεστής μπορεί να συμμετέχει και σε μια παράσταση με χρόνο. Εφόσον τα ορίσματα είναι παραστάσεις χωρίς χρόνο, μπορούν να αποτιμηθούν ταυτόχρονα και να επιλεγεί εύκολα η εκάστοτε τιμή με βάση τη λογική παράσταση. Εάν ο τελεστής εμφανίζεται σε μία παράσταση με χρόνο, τότε απλά μεταδίδει την είσοδο sin στην έξοδο sout, αφού κατά τα άλλα η αποτίμησή του δεν απαιτεί συνολικά χρόνο.

Το κύκλωμα που αντιστοιχεί φαίνεται στην εικόνα 47. Η τιμή αληθείας της λογικής συνθήκης 'cond_expression', που έχει προκύψει και από ενδεχόμενη μετατροπή σε τιμή αληθείας (που παραλείπεται από το σχήμα), εισάγεται ως σήμα επιλογής σε έναν πολυπλέκτη (MUX). Ο πολυπλέκτης επιλέγει μεταξύ των τιμών των δύο παραστάσεων και εξάγει τη σωστή έξοδο val. Η σχεδίαση αυτή του τελεστή επιτρέπει στον προγραμματιστή να εισάγει εύκολα ένα συνδυαστικό κύκλωμα επιλογής σαν να συνέταζε μια αντίστοιχη εντολή if.



Εικόνα 47 – Το κύκλωμα του τελεστή παράστασης υπό συνθήκη

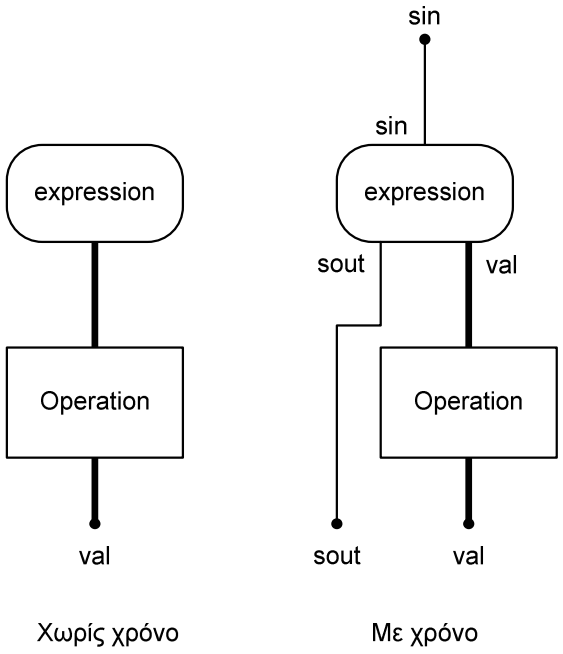
Ένα θέμα που τίθεται στην υλοποίηση του τελεστή είναι ο τρόπος που μετατρέπονται τα ορίσματα σε περίπτωση που είναι διαφορετικού τύπου. Αν αμφότερα τα ορίσματα είναι προσημασμένα ή απρόσημα, αρκεί η γνωστή διαδικασία για την μετατροπή κοινού τύπου. Αν όμως το όρισμα που έχει το μεγαλύτερο μέγεθος είναι απρόσημο και το άλλο, μικρότερου ή ίσου μεγέθους, είναι προσημασμένο, τότε αξίζει να γίνει μετατροπή κοινού τύπου αυξάνοντας το συνολικό αριθμό bit κατά ένα, ώστε το τελικό αποτέλεσμα ναι μεν να είναι προσημασμένο, αλλά να παριστάνει τον απρόσημο αριθμό με το σωστό τρόπο. Για να γίνει αυτό περισσότερο κατανοητό, έστω το παρακάτω παράδειγμα:

```
unsigned char u = 255;
signed char s = -1;
int i, c;

i = (c == 2) ? u : s;
```

Στον κώδικα αυτό, ορίζονται δύο μεταβλητές μήκους 8 bits, που αμφότερες έχουν τη δυαδική τιμή 11111111 (ή FF, στο δεκαεξαδικό): η απρόσημη μεταβλητή u αρχικοποιείται με τη θετική δεκαδική σταθερά 255, και η προσημασμένη μεταβλητή s με την αρνητική σταθερά -1, που επεκτεινόμενη δίνει την ίδια αναπαράσταση. Στη συνέχεια, η ακέραια μεταβλητή i, μεγέθους 32 bits, λαμβάνει μία εκ των δύο τιμών: αν η μεταβλητή c είναι ίση με 2, λαμβάνει την τιμή 255, δηλαδή πρέπει να γεμίσει τα bits της με τη δεκαεξαδική τιμή 000000FF, αλλιώς λαμβάνει την τιμή -1, δηλαδή πρέπει τελικά να αποκτήσει τη δεκαεξαδική τιμή FFFFFFFF. Ωστόσο, με την κανονική μετατροπή κοινού τύπου ο τελεστής της παράστασης υπό συνθήκη θα μετέτρεπε το τελικό αποτέλεσμά του σε signed bit<8>, και έτσι ακόμα και η απρόσημη τιμή της μεταβλητής u θα θεωρούταν προσημασμένη, μετατρεπόμενη με επέκταση προσήμου στην τελική δεκαεξαδική τιμή FFFFFFFF και όχι 000000FF. Η λύση είναι ο τελεστής να παράγει ένα αποτέλεσμα της μορφής signed bit<9>, μετατρέποντας τα ορίσματα στα δυαδικά διανύσματα 011111111 και 111111111 αντίστοιχα. Έτσι, οι παραστάσεις αυτές θεωρούμενες ως προσημασμένες διατηρούν πάραυτα την αρχική τους τιμή, και η επέκταση προσήμου που γίνεται τελικά κατά την ανάθεση δεν οδηγεί σε εσφαλμένο αποτέλεσμα.

Οι μοναδιαίοι τελεστές.

Οι μοναδιαίοι τελεστές που συμμετέχουν σε παραστάσεις με ή χωρίς χρόνο συντάσσονται με ένα μόνο όρισμα και το κύκλωμά τους δεν είναι τόσο σύνθετο όσο αυτό των δυαδικών τελεστών. Η γενική μορφή του κυκλώματος που παράγεται για ένα μοναδιαίο τελεστή φαίνεται στην εικόνα 48.



Εικόνα 48 – Το κύκλωμα της παράστασης ενός μοναδιαίου τελεστή

Όπως δείχνει και η εικόνα, ό,τι είδους είναι η παράσταση στην οποία συμμετέχει ο τελεστής (με ή χωρίς χρόνο), του ίδιου είδους είναι και η παράσταση με την οποία συντάσσεται. Η πράξη του κάθε τελεστή υλοποιείται με το αντίστοιχο συνδυαστικό κύκλωμα 'Operation'. Δε μένει λοιπόν παρά να εξεταστεί το κύκλωμα αυτό της πράξης για κάθε τελεστή ξεχωριστά.

Οι πράξεις τεσσάρων βασικών μοναδιαίων τελεστών της C φαίνονται στην εικόνα 49. Οι τελεστές αυτοί είναι οι εξής:

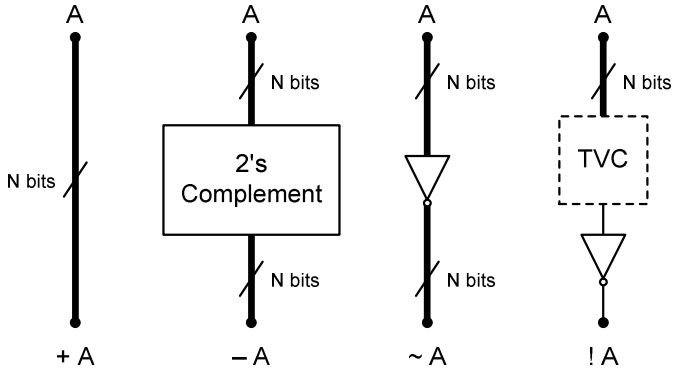
- Τελεστής μοναδιαίου συν: '+'
- Τελεστής μοναδιαίου πλην: '–'
- Τελεστής bitwise NOT: '~'
- Τελεστής boolean NOT: '!'

Η σύνταξή τους είναι η εξής:

```
+ <expression>
- <expression>
~ <expression>
! <expression>
```

Ο τελεστής του μοναδιαίου συν, που εισήχθη για πρώτη φορά στο πρότυπο της ANSI C για λόγους συμμετρίας, δεν απαιτεί κανένα ιδιαίτερο κύκλωμα για την πράξη του: απλά μεταδίδει την είσοδο στην έξοδο, διατηρώντας και τον ίδιο τύπο. Αντίθετα, ο

τελεστής του μοναδιαίου πλην απαιτεί ένα κύκλωμα για την παραγωγή του συμπληρώματος ως προς 2 της εισόδου. Το αποτέλεσμα έχει το ίδιο μέγεθος με το όρισμα, αλλά ο τύπος της εξόδου είναι πάντοτε προσημασμένος. Ο τελεστής bitwise NOT, γνωστός και ως τελεστής συμπληρώματος ως προς 1, χρειάζεται απλούς αντιστροφείς για τη μετατροπή καθενός bit της εισόδου στο αντίστροφό του. Σε αντίθεση με τον τελεστή μοναδιαίου πλην, ο τελεστής bitwise NOT διατηρεί τον ίδιο τύπο (προσημασμένο ή απρόσημο) στην έξοδο. Τέλος, ο τελεστής boolean NOT πραγματοποιεί το λογικό NOT με τη βοήθεια ενός και μόνου αντιστροφέα, αφού παράγει την τιμή αληθείας με τη βοήθεια του κυκλώματος μετατροπής σε τιμή αληθείας (truth value conversion – TVC) το οποίο παρουσιάστηκε στην εικόνα 29. Το κύκλωμα αυτό είναι απόν σε περίπτωση που η είσοδος έχει ήδη μέγεθος ενός bit. Σε κάθε περίπτωση, η έξοδος του τελεστή είναι του τύπου signed bit, όπως γίνεται με όλες εξάλλου τις λογικές παραστάσεις.



Εικόνα 49 – Τα κυκλώματα των πράξεων τεσσάρων βασικών μοναδιαίων τελεστών

Ο τελεστής επιλογής bit.

Αν και από τη γλώσσα έχουν αφαιρεθεί γενικά οι πίνακες (μιας και γίνεται πάντοτε χειρισμός διανυσμάτων από bits), η παρακάτω σύνταξη (στην οποία γράφεται μια σταθερά μέσα σε αγκύλες μετά από μια παράσταση) έχει παραμείνει ως ένας τρόπος επιλογής ενός bit από ένα δεδομένο:

```
<expression> [ <integer> ]
```

Η ακέραια σταθερά ‘integer’ με την οποία συντάσσεται ο τελεστής δεικνύει τη θέση του bit που θα επιλεγεί από το διάνυσμα του αποτελέσματος της παράστασης. Τα bits αριθμούνται ξεκινώντας από το μηδέν, που είναι το δεξιότερο (λιγότερο σημαντικό) bit, και η θέση τους αυξάνεται προς τα αριστερά. Φυσικά, η σταθερά θα πρέπει να επιλέγει ένα bit εντός των ορίων που καθορίζει το μέγεθος της παράστασης. Ο χαρακτηρισμός του bit εξόδου ως προσημασμένο ή απρόσημο, από άποψη τύπου, γίνεται με βάση τον αντίστοιχο χαρακτηρισμό που φέρει η παράσταση του ορίσματος. Ο τελεστής επιλογής bit μπορεί να συμμετέχει τόσο σε παραστάσεις με χρόνο όσο και σε παραστάσεις χωρίς χρόνο, θεωρούμενος σαν ένας μοναδιαίος τελεστής, με το κύκλωμα της εικόνας 48. Η πράξη ‘Operation’ του τελεστή δεν είναι παρά μια απομόνωση ενός από τα N bits της τιμής val, η θέση του οποίου είναι γνωστή κατά τη διάρκεια της σύνθεσης. Ας σημειωθεί επίσης ότι η επιλογή ενός bit δεν αποτελεί σύνταξη α-τιμής, δηλαδή η παραπάνω σύνταξη δε μπορεί να χρησιμοποιηθεί για την απόδοση τιμής σε ένα συγκεκριμένο bit μιας μεταβλητής.

Ο τελεστής προσαρμογής τύπου (cast).

Ο τελεστής προσαρμογής τύπου (cast) χρησιμοποιείται για την αλλαγή του τύπου μιας παράστασης. Ο τελεστής αυτός συντάσσεται ως εξής:

```
( <type> ) <expression>
```

Ο τύπος, ο οποίος γράφεται εντός παρενθέσεων στα αριστερά μιας παράστασης, ακολουθεί το γνωστό τρόπο δήλωσης που περιγράφηκε στην ενότητα των καθολικών μεταβλητών. Το κύκλωμα της εικόνας 48 που εισάγεται για τη σύνθεση του τελεστή συμπληρώνεται από ένα κύκλωμα μετατροπής τύπου (type conversion) στη θέση της πράξης 'Operation'. Η διαδικασία της μετατροπής τύπου (που περιγράφηκε αναλυτικότερα στην ενότητα της εντολής return) είτε αποκόπτει κάποια bits, αν ο νέος τύπος έχει μικρότερο μέγεθος, είτε επεκτείνει τα bits της παλιάς τιμής με προκαθορισμένο τρόπο, αν ο νέος τύπος έχει μεγαλύτερο μέγεθος.

Η χρήση παρενθέσεων σε παραστάσεις.

Όποτε απαιτείται στη σύνταξη της γλώσσας μια οποιουδήποτε είδους παράσταση (είτε πρόκειται για παράσταση με χρόνο, παράσταση χωρίς χρόνο, παράσταση εντολής ή α-τιμή), αυτή μπορεί να γραφεί με τη χρήση παρενθέσεων:

```
( <expression> )
```

Οι παρενθέσεις χρησιμοποιούνται για να επιβάλουν μια συγκεκριμένη προτεραιότητα στον τρόπο αποτίμησης και δεν μεταφράζονται σε κύκλωμα.

Η κλήση συνάρτησης. Ο τελεστής διεύθυνσης.

Η κλήση συναρτήσεων είναι ένα σημαντικό κομμάτι της εργασίας. Η δυνατότητα δυναμικών κλήσεων μιας συνάρτησης από πολλά σημεία ενός προγράμματος είναι μια πρόκληση για τη διαδικασία της σύνθεσης, και στην παρούσα εργασία δίδεται μια λύση χωρίς τη χρήση στοίβας ή άλλων δυναμικών δομών. Ορισμένα πολύ σημαντικά επιμέρους στοιχεία μιας κλήσης (όπως οι παράμετροι κατ' αξία και κατ' αναφορά, η τιμή επιστροφής, ο κολλώδης ανανεωτής, κ.α.) έχουν ήδη αναλυθεί σε προηγούμενες ενότητες και έτσι αυτή η παράγραφος θα επικεντρωθεί περισσότερο σε όσα δεν έχουν αναφερθεί, συνοψίζοντας και ανακεφαλαιώνοντας τη διαδικασία μιας κλήσης.

Μία κλήση μπορεί να εμφανίζεται είτε σε παράσταση με χρόνο, είτε σε παράσταση εντολής. Στην πρώτη περίπτωση θα πρέπει να επιστρέψει κάποια τιμή, και άρα δεν μπορεί να χρησιμοποιηθεί μια συνάρτηση με void τύπο επιστροφής. Στη δεύτερη περίπτωση, η τιμή επιστροφής γενικά αγνοείται, αφού η κλήση λειτουργεί σαν εντολή.

Η σύνταξη είναι η εξής:

```
<identifier> [ [ <integer> ] ] ( <expression> , … , <expression> )
```

Το *αναγνωριστικό* (identifier) καθορίζει το όνομα της συνάρτησης που θα κληθεί. Δίπλα από το αναγνωριστικό εισάγεται προαιρετικά ένας ακέραιος αριθμός μέσα σε αγκύλες, για τον καθορισμό του αντιτύπου της συνάρτησης που θα κληθεί. Όπως έχει

αναφερθεί, κατά τη δήλωση μιας συνάρτησης είναι δυνατό να δοθεί (με μια σύνταξη που θυμίζει δήλωση πινάκων) ένας αριθμός που καθορίζει το πλήθος των αντιτύπων της συνάρτησης. Εφόσον μια συνάρτηση παράγεται λοιπόν σε περισσότερα του ενός κυκλωματικά αντίτυπα, μπορεί κατά την κλήση της να δοθεί προαιρετικά ένας αριθμός για να επιλεγεί ένα συγκεκριμένο αντίτυπο. Εάν στη σύνταξη μιας κλήσης δεν δοθεί ο αριθμός του αντιτύπου κλήσης, επιλέγεται αυτόματα το πρώτο αντίτυπο της συνάρτησης. Εξαίρεση αποτελεί η περίπτωση που μια συνάρτηση αποπειράται να καλέσει τον εαυτό της: τότε, αν δεν καθοριστεί ο αριθμός του αντιτύπου, πρόκειται για μια ψευδο-αναδρομή που πραγματοποιείται με την κλήση του επόμενου κατά σειρά αντιτύπου. Δηλαδή, αν στον κώδικα μιας συνάρτησης εμφανίζεται μια κλήση προς τον εαυτό της, τότε σε εκείνο το σημείο το κύκλωμα του πρώτου αντιτύπου θα καλεί το δεύτερο, ενώ το κύκλωμα του δεύτερου αντιτύπου θα καλεί το τρίτο, κ.ο.κ. Για λόγους συμμετρίας, το τελευταίο αντίτυπο καλεί ξανά το πρώτο.

Τα *ορίσματα* (arguments) της κλήσης γράφονται εντός παρενθέσεων. Οι παρενθέσεις είναι παρούσες ακόμα και αν η συνάρτηση δε δέχεται ορίσματα. Τα ορίσματα αντιστοιχούν ένα προς ένα στις παραμέτρους με τις οποίες έχει δηλωθεί η συνάρτηση. Ο χειρισμός των ορισμάτων που αντιστοιχούν σε παραμέτρους κατ' αξία γίνεται με διαφορετικό τρόπο σε σχέση με τα ορίσματα των παραμέτρων κατ' αναφορά.

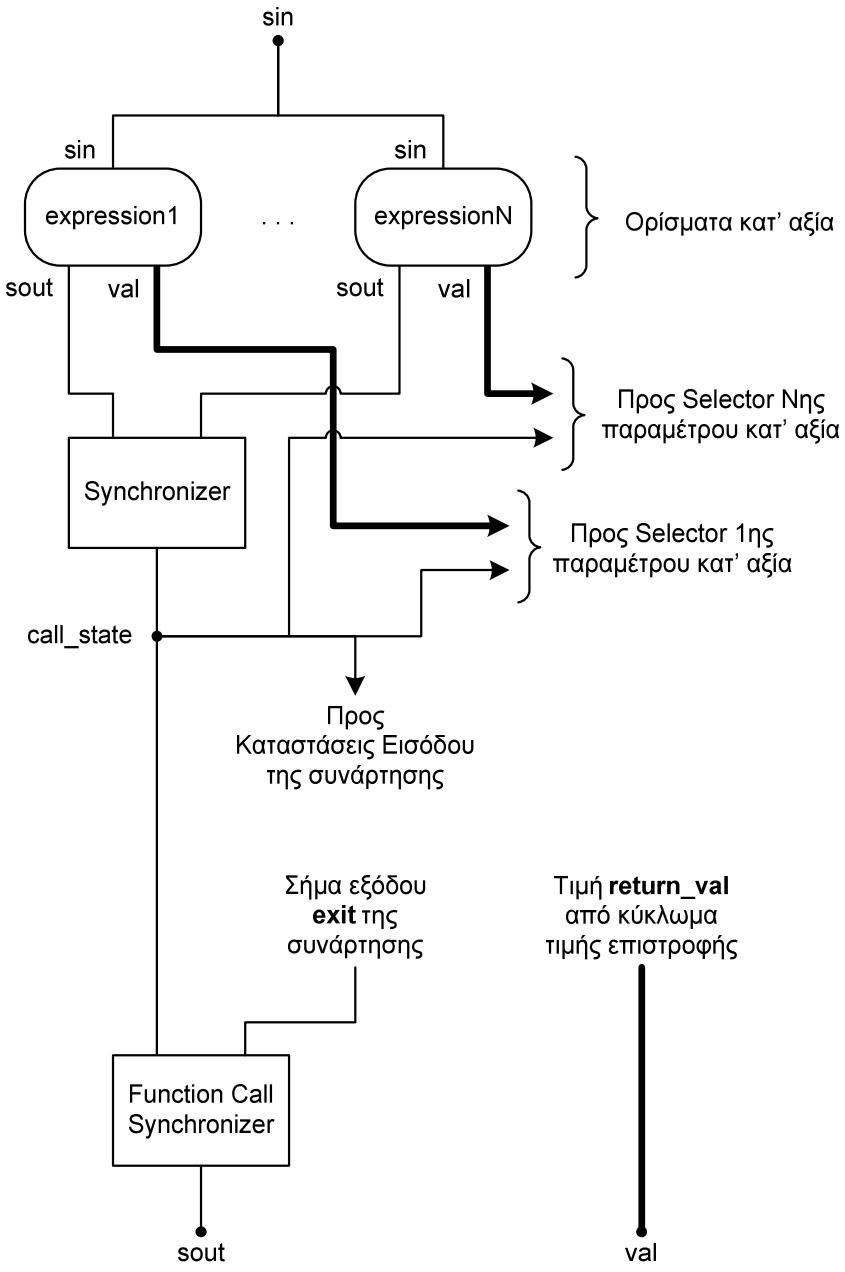
Από κυκλωματική άποψη, η κλήση συνάρτησης, είτε ως παράσταση με χρόνο είτε ως παράσταση εντολής, λαμβάνει μια κατάσταση εισόδου sin, και όταν ολοκληρωθεί ενεργοποιεί την κατάσταση εξόδου sout και (στην περίπτωση της παράστασης με χρόνο) εξάγει μια τιμή val. Για την πραγματοποίηση μιας κλήσης πρέπει να αποφασιστεί η τεχνική αποτίμησης των ορισμάτων, και αφού αποτιμηθούν τα ορίσματα και μεταδοθεί η τιμή τους στη συνάρτηση, να ενεργοποιηθεί το κύκλωμα της καλούμενης συνάρτησης. Όταν η συνάρτηση επιστρέψει, το κύκλωμα της κλήσης πρέπει να λάβει την επιστρεφόμενη τιμή (στις περιπτώσεις που αυτό απαιτείται) και να ολοκληρώσει την εκτέλεσή του ενεργοποιώντας το τελικό σήμα εξόδου sout. Όλα αυτά τα επιμέρους κομμάτια (ο τρόπος αποτίμησης και μετάδοσης των ορισμάτων, η ενεργοποίηση της συνάρτησης, ο τρόπος αναμονής για την επιστροφή, κ.α.) που συνθέτουν μια κλήση αναλύονται στη συνέχεια.

Τα ορίσματα που αντιστοιχούν σε παραμέτρους κατ' αξία συντάσσονται με παραστάσεις με χρόνο. Οι παραστάσεις αυτές καθορίζουν και το χρονισμό ολόκληρης της κλήσης, διότι αν δεν αποτιμηθούν δεν είναι δυνατό να ξεκινήσει η κλήση. Αντίθετα, τα ορίσματα που αντιστοιχούν σε παραμέτρους κατ' αναφορά δεν αποτιμώνται, αφού για αυτά δε στέλνεται μια τιμή, παρά πραγματοποιείται μια σύνδεση σημάτων ελέγχου και δεδομένων με τη συνάρτηση, όπως εξηγήθηκε και στην παράγραφο που αφορά την αναπαράσταση των παραμέτρων κατ' αναφορά. Έτσι, στη θέση ενός ορίσματος κατ' αναφορά δε γράφεται μια παράσταση με χρόνο, αλλά μια *παράσταση δείκτη* (pointer expression). Η παράσταση αυτή είναι μια παράμετρος κατ' αναφορά της καλούσης συνάρτησης (στην περίπτωση που μια συνάρτηση θέλει να περάσει αυτούσια μια παράμετρο κατ' αναφορά σε μια άλλη συνάρτηση), ή μια καθολική μεταβλητή τύπου δείκτη, ή ένας δείκτης σε κάποια μεταβλητή που εισάγεται με τη χρήση του τελεστή διεύθυνσης. Ο *τελεστής διεύθυνσης* χρησιμοποιείται για τη μετατροπή μιας α-τιμής σε δείκτη, και συντάσσεται ως εξής:

```
& <l-value>
```

Σε κάθε περίπτωση, ένα όρισμα που αντιστοιχεί σε παράμετρο κατ' αναφορά καταλήγει να είναι μια αναφορά σε κάποιά αποθηκευτική μονάδα, η οποία έχει τη δυνατότητα να εξάγει μια τιμή val και να δεχτεί ένα ζεύγος τιμής-κατάστασης προς εγγραφή.

Επειδή, όπως αναφέρθηκε, για την προετοιμασία της κλήσης απαιτούν χρόνο μόνο τα ορίσματα κατ' αξία, αυτά είναι που λαμβάνουν την κατάσταση εισόδου sin της κλήσης. Η αποτίμησή τους γίνεται παράλληλα, δηλαδή το σήμα sin μεταδίδεται ταυτόχρονα σε όλα τα κυκλώματα των αντίστοιχων παραστάσεων με χρόνο. Εξάλλου, το πρότυπο της ANSI C αναφέρει ότι σε μία κλήση "η σειρά υπολογισμού των ορισμάτων είναι ακαθόριστη". Η παράλληλη μέθοδος αποτίμησης σημαίνει ότι σε μία κλήση όπως η f(g(), h()), θα κληθούν πρώτα ταυτόχρονα οι συναρτήσεις g και h. Επειδή γενικά δεν ολοκληρώνουν όλα τα ορίσματα την αποτίμησή τους στον ίδιο χρόνο, απαιτείται ένας συγχρονιστής, όπως φαίνεται και στην εικόνα 50.

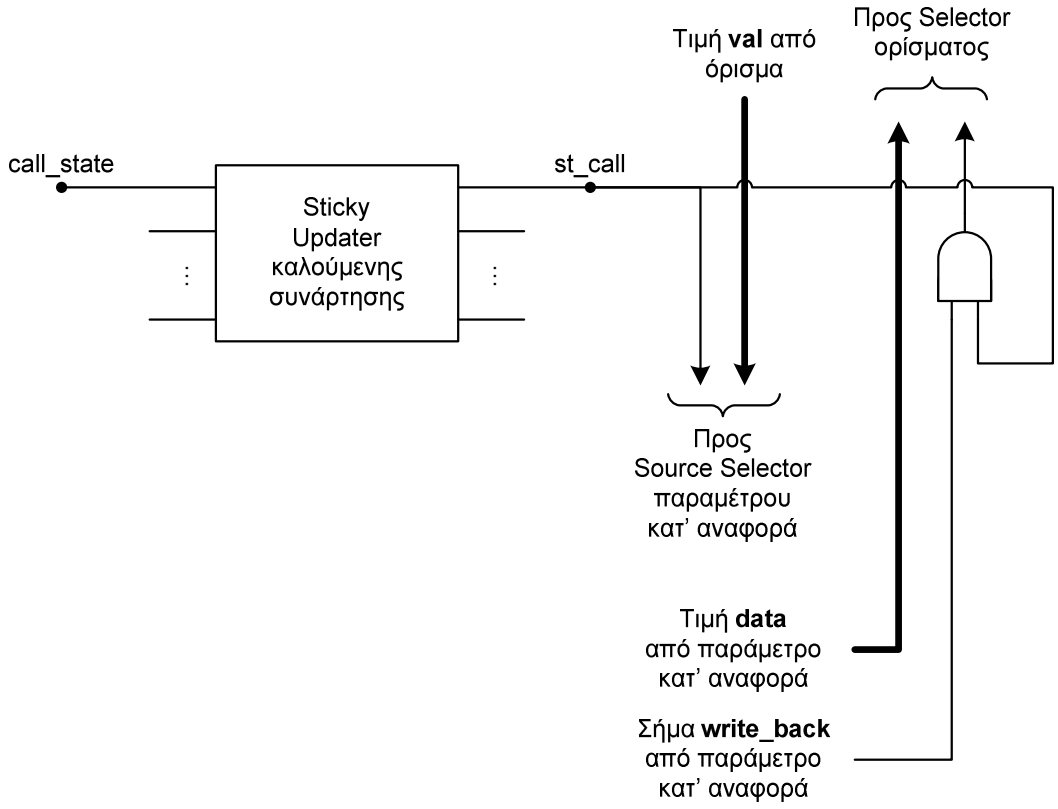


Εικόνα 50 – Το βασικό μέρος του κυκλώματος κλήσης

Ο συγχρονιστής αυτός λαμβάνει τις καταστάσεις εξόδου των παραστάσεων με χρόνο των ορισμάτων κατ' αξία και παράγει μια συνολική κατάσταση εξόδου call_state. Η κατάσταση αυτή είναι αρκετά σημαντική διότι είναι εκείνη στην οποία θα ξεκινήσει η κλήση. Στην περίπτωση που η συνάρτηση δεν περιέχει παραμέτρους κατ' αξία, η κατάσταση call_state ταυτίζεται με την κατάσταση εισόδου sin.

Η μετάδοση των ορισμάτων κατ' αξία πραγματοποιείται με την εισαγωγή των αντίστοιχων ζευγών τιμών-καταστάσεων στους επιλογείς των παραμέτρων της καλούμενης συνάρτησης. Για κάθε παράμετρο κατ' αξία, η κατάσταση call_state μαζί με την τιμή val της αντίστοιχης παράστασης εισάγονται στον επιλογέα της παραμέτρου. Ο επιλογέας της παραμέτρου κατ' αξία, που παρουσιάστηκε στην εικόνα 17, αποτελεί μέρος της καλούμενης συνάρτησης, και έχει ήδη δημιουργηθεί κατά τη δήλωση της. Ας σημειωθεί ακόμα ότι η τιμή val κάθε παράστασης θα πρέπει πρώτα να μετατραπεί στον τύπο της αντίστοιχης παραμέτρου, με τη γνωστή διαδικασία μετατροπής τύπου.

Για τα ορίσματα που αντιστοιχούν σε παραμέτρους κατ' αναφορά η διαδικασία μετάδοσης έχει ήδη περιγραφεί σε προηγούμενη ενότητα που παρουσιάστηκαν τα κυκλώματα των παραμέτρων κατ' αναφορά, και μέσα από παράδειγμα (εικόνα 22). Η διαδικασία αυτή φαίνεται γενικότερα στην εικόνα 51.

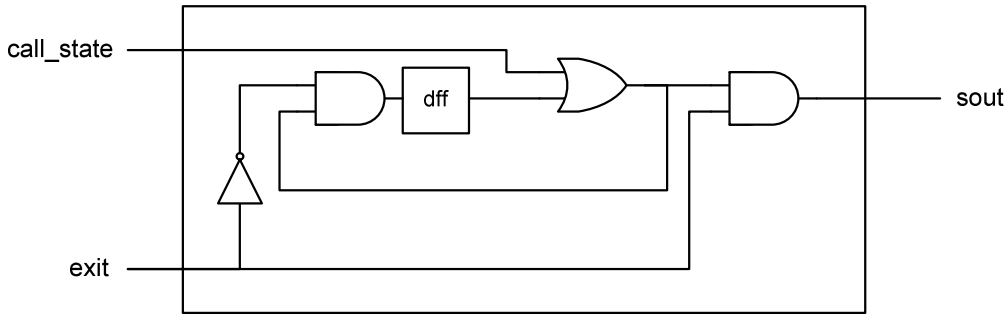


Εικόνα 51 – Το μέρος του κυκλώματος κλήσης που αφορά τις παραμέτρους κατ' αναφορά

Όπως είναι φανερό, το σήμα κλήσης call_state εισάγεται στον κολλώδη ανανεωτή (Sticky Updater) της καλούμενης συνάρτησης ώστε να προκύψει το αντίστοιχο κολλώδες (sticky) σήμα st_call. Από εκεί και πέρα, για κάθε παράμετρο κατ' αναφορά πραγματοποιείται η σύνδεση των σημάτων του ορίσματος με την παράμετρο, όπως δείχνει και το σχήμα: η τιμή val του ορίσματος εισάγεται στον επιλογέα πηγής της

παραμέτρου, ενώ η τιμή data της παραμέτρου μαζί με το (φιλτραρισμένο από το st_call) σήμα εγγραφής write_back εισάγονται πίσω στον επιλογέα του ορίσματος. Αν το όρισμα δεν έχει ακριβώς το ίδιο μέγεθος με την παράμετρο, τότε η σύνδεσή τους θεωρείται μη επιτρεπτή.

Επιστρέφοντας στην εικόνα 50, το σήμα call_state εισάγεται στη λίστα των καταστάσεων εισόδου της καλούμενης συνάρτησης, προκειμένου να ξεκινήσει η εκτέλεσή της. Οι καταστάσεις εισόδου της καλούμενης συνάρτησης ενεργοποιούν το κύκλωμα ελέγχου της εικόνας 14, και αφού γραφούν τα δεδομένα των παραμέτρων κατ' αξία εκκινείται το σώμα της συνάρτησης. Κατά την εκτέλεσή της, η καλούμενη συνάρτηση δεν πρέπει να ξανακληθεί—εξάλλου αν κάτι τέτοιο είναι απόλυτα απαραίτητο, γι' αυτό έχει προβλεφθεί ο μηχανισμός δημιουργίας αντιτύπων. Δε μένει παρά να εξεταστεί ο τρόπος που η συνάρτηση επιστρέφει στο σωστό σημείο από το οποίο εκλήθη. Το κύκλωμα της συνάρτησης (εικόνα 14) εξάγει ένα σήμα exit που ενεργοποιείται όταν αυτή ολοκληρώσει την εκτέλεσή της και επιστρέψει. Το σήμα αυτό εισάγεται σε έναν ειδικό συγχρονιστή (εικόνα 50) μαζί με το σήμα call_state που ήταν υπεύθυνο για την κλήση, και το προκύπτον σήμα sout αποτελεί την τελική κατάσταση εξόδου της κλήσης. Ο ειδικός αυτός συγχρονιστής ονομάζεται *συγχρονιστής κλήσης συνάρτησης* (function call synchronizer) και λειτουργεί ως εξής: μόλις ξεκινήσει η κλήση, το σήμα call_state ενεργοποιείται και θέτει σε αναμονή το συγχρονιστή. Ο συγχρονιστής, όσο βρίσκεται σε αναμονή, περιμένει για την έλευση του σήματος exit της συνάρτησης. Όταν ενεργοποιηθεί αυτό το σήμα εξόδου της συνάρτησης, ο συγχρονιστής ενεργοποιεί την έξοδο του sout και ταυτόχρονα μηδενίζεται ώστε να μπορεί να ξαναχρησιμοποιηθεί αν χρειαστεί. Μια υλοποίηση αυτής της συμπεριφοράς που περιγράφηκε φαίνεται στην εικόνα 52. Όπως και με τα περισσότερα κυκλώματα, ο συγχρονιστής κλήσης συνάρτησης ενεργοποιεί την έξοδο sout αμέσως μόλις επιστρέψει η συνάρτηση, χωρίς να απαιτείται επιπλέον κύκλος ρολογιού.



Εικόνα 52 – Μια εκδοχή του εσωτερικού κυκλώματος ενός συγχρονιστή κλήσης συνάρτησης

Η μεγάλη διαφορά του κυκλώματος αυτού από τους απλούς συγχρονιστές (που χρησιμοποιούνται π.χ. στο παράλληλο μπλοκ εντολών) είναι η ασυμμετρία των εισόδων: δεν πρόκειται για έναν απλό συγχρονισμό των σημάτων call_state και exit, παρά απαιτείται πρώτα να ενεργοποιηθεί η κλήση και μετά η επιστροφή. Αντίθετα, ένας απλός συγχρονιστής θα περίμενε για την έλευση των σημάτων με οποιαδήποτε σειρά, και έτσι δεν θα λειτουργούσε σωστά. Πράγματι, αν μια συνάρτηση καλούταν δύο φορές από δύο διαφορετικά σημεία του προγράμματος (και χρησιμοποιούνταν απλοί συγχρονιστές), τότε όταν θα επέστρεφε η πρώτη κλήση, το σήμα exit (που μεταδίδεται σε όλα τα σημεία από τα οποία πραγματοποιείται κλήση) θα ενεργοποιούσε και το συγχρονιστή του δεύτερου κυκλώματος κλήσης. Έτσι, όταν θα γινόταν η δεύτερη κλήση, θα έδινε την εντύπωση ότι επέστρεφε κατευθείαν αφού ο απλός συγχρονιστής

θα είχε λάβει ενεργοποίηση του σήματος exit στο παρελθόν, και θα περίμενε μόνο για την ενεργοποίηση του σήματος call_state. Η εισαγωγή του συγχρονιστή κλήσης συνάρτησης στη διαδικασία της σύνθεσης εξασφαλίζει ότι μια συνάρτηση που δύναται να κληθεί από πολλά διαφορετικά σημεία του προγράμματος θα επιστρέφει κάθε φορά στο σωστό σημείο, αφού μόνο ένας τέτοιος συγχρονιστής θα είναι σε αναμονή (οι άλλοι θα λαμβάνουν μεν το σήμα exit αλλά θα το αγνοούν, γιατί δεν θα έχουν πρωτύτερα ενεργοποιηθεί με το δικό τους σήμα call_state). Η λειτουργία αυτή κάνει φανερό και το γεγονός ότι δεν πρέπει μια συνάρτηση να κληθεί από δύο διαφορετικά σημεία ταυτόχρονα, δηλαδή απαγορεύεται να ξανασυμβεί μια κλήση προς τη συνάρτηση πριν η προηγούμενη κλήση επιστρέψει, διότι τότε θα υπάρξουν προς στιγμήν δύο ενεργοποιημένοι συγχρονιστές, που θα θεωρήσουν την επόμενη κατά σειρά επιστροφή της συνάρτησης ως επιστροφή για αμφότερους.

Τέλος, αφού η συνάρτηση επιστρέψει, η τιμή return_val του καταχωρητή που συμμετέχει στο κύκλωμα της τιμή επιστροφής (εικόνα 16) εξάγεται στο σήμα val, εφόσον βέβαια η κλήση αποτελεί παράσταση με χρόνο και όχι παράσταση εντολής.

Η διασύνδεση με το εξωτερικό περιβάλλον

Η διασύνδεση του κυκλώματος με το εξωτερικό περιβάλλον είναι ένα πολύ σημαντικό ζήτημα. Κατά κάποιο τρόπο, το πρόγραμμα αποτελεί ένα κλειστό σύστημα υπό την έννοια ότι απαρτίζεται από συναρτήσεις που καλούνται μεταξύ τους, ανταλλάσσοντας δεδομένα. Για την επικοινωνία με το εξωτερικό περιβάλλον έχουν προβλεφθεί μηχανισμοί που να είναι όσο το δυνατό περισσότερο διαφανείς και να μην απαιτούν ιδιαίτερη σύνταξη ή αλλαγές στη γλώσσα C.

Έτσι, όπως έχει ήδη αναφερθεί, το πρόγραμμα μπορεί να μεταδώσει και να λάβει δεδομένα από το εξωτερικό περιβάλλον χρησιμοποιώντας τις καθολικές μεταβλητές. Πιο συγκεκριμένα, οι απλές καθολικές μεταβλητές (εικόνα 12) που δεν έχουν δηλωθεί ως static εξάγονται κυκλωματικά, δηλαδή το σήμα val που παριστάνει την τιμή τους αποτελεί έξοδο του κυκλώματος. Το σήμα αυτό μπορεί να χρησιμοποιηθεί από τυχόν άλλα εξωτερικά κυκλώματα που χρειάζεται να αναγνώσουν την τιμή της εκάστοτε μεταβλητής. Επίσης, οι καθολικές μεταβλητές τύπου δείκτη (εικόνα 20) χρησιμοποιούν μια διαπροσωπεία τριών σημάτων για την επικοινωνία με το περιβάλλον. Το σήμα val εισέρχεται προς το κύκλωμα του προγράμματος και περιέχει την τρέχουσα τιμή του δεδομένου που παριστάνει ο δείκτης, ενώ το σήμα data εξάγει μια τιμή προς εγγραφή, μαζί με την αντίστοιχη κατάσταση write_back.

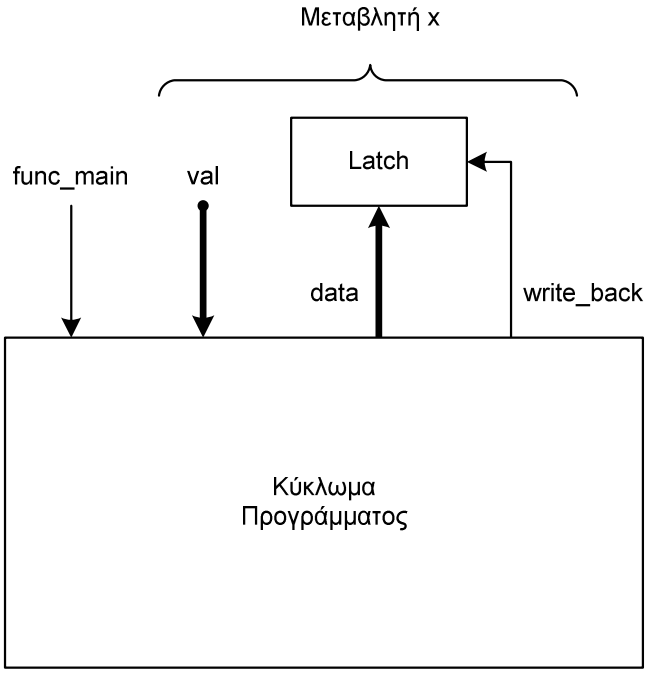
Εκτός όμως από τα δεδομένα, το πρόγραμμα απαιτείται να επικοινωνεί με το εξωτερικό περιβάλλον και για τους σκοπούς του ελέγχου, δηλαδή απαιτούνται σήματα προκειμένου να πυροδοτηθεί το πρόγραμμα και να ξεκινήσει η εκτέλεση του κώδικά του. Στην κλασική C, το σημείο εκκίνησης ενός εκτελέσιμου είναι η συνάρτηση main αλλά για τις ανάγκες της παρούσας εργασίες δεν είναι κάτι τέτοιο αναγκαίο, αφού είναι αρκετά εύκολο κάθε συνάρτηση να συμπεριλάβει άλλο ένα σήμα κατάστασης κλήσης (call state) στην είσοδό της, το οποίο να προέρχεται από το εξωτερικό περιβάλλον. Έτσι, ο χρήστης του κυκλώματος μπορεί να εκκινήσει όποια συνάρτηση θέλει, σε όποια χρονική στιγμή επιθυμεί, αλλά συνήθως θα θέλει να το πράξει μόνο για κάποια συνάρτηση αρχικοποίησης (αντίστοιχη της κλασικής main), η οποία θα περιέχει κώδικα για την κλήση των υπόλοιπων συναρτήσεων του προγράμματος.

Οι παραπάνω μηχανισμοί που περιγράφηκαν είναι αρκετοί για να δημιουργηθούν πολυάριθμα σενάρια χρήσης ενός προγράμματος από το εξωτερικό περιβάλλον. Στην πιο απλή περίπτωση επεξεργασίας ενός δεδομένου, ένα παράδειγμα χρήσης φαίνεται παρακάτω:

```
int *x;

void main()
{
  *x = f(*x);
}
```

Η συνάρτηση f() λαμβάνει ένα όρισμα κατ' αξία και πραγματοποιεί κάποια επεξεργασία. Η καθολική μεταβλητή x τύπου δείκτη συνδέεται με το εξωτερικό περιβάλλον το οποίο τροφοδοτεί την τιμή προς επεξεργασία στο αντίστοιχο σήμα val. Όταν ξεκινά την εκτέλεσή της η main, η συνάρτηση f λαμβάνει το αντίστοιχο όρισμα και πραγματοποιεί την επεξεργασία του, επιστρέφοντας μια τελική τιμή. Η τιμή αυτή αποθηκεύεται πίσω στη μεταβλητή x, στο σήμα data αυτής, ενεργοποιώντας ταυτόχρονα

την κατάσταση write_back. Έτσι, το εξωτερικό κύκλωμα δεν έχει παρά να αναμένει την έλευση του παλμού στο σήμα write_back, οπότε και λαμβάνει τα αποτελέσματα στο σήμα data. Το τελικό αποτέλεσμα μπορεί να αποθηκευτεί σε ένα latch το οποίο να κλειδώσει την τιμή που δέχεται χρησιμοποιώντας το σήμα write_back. Αυτός ο απλός τρόπος επικοινωνίας παριστάνεται στην εικόνα 53. Το σήμα func_main χρησιμοποιείται για να πυροδοτήσει τη main.

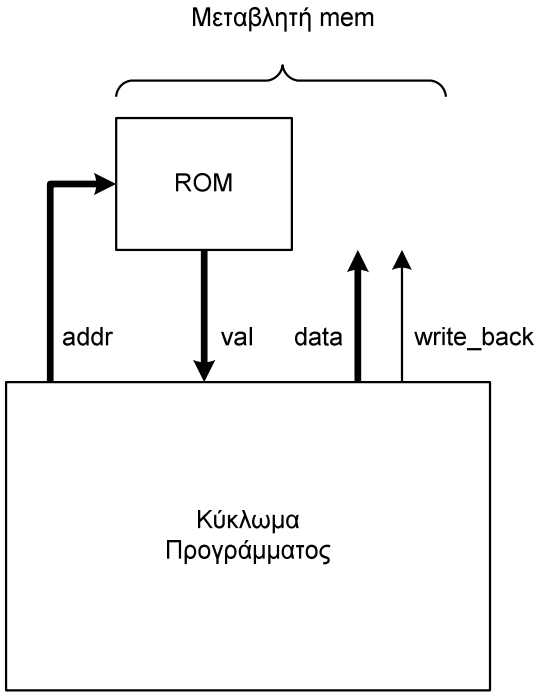


Εικόνα 53 – Ένα απλό παράδειγμα διασύνδεσης

Μια πιο απαιτητική εφαρμογή μπορεί να χρειάζεται σύνδεση με κάποιο είδος μνήμης. Στην πιο απλή περίπτωση υπάρχει η μνήμη ανάγνωσης (ROM – Read Only Memory). Η μνήμη αυτή λαμβάνει από μία θύρα μια διεύθυνση, και εξάγει το περιεχόμενο αυτής της διεύθυνσης μνήμης σε μία άλλη θύρα δεδομένων. Δεν απαιτούνται άλλα σήματα που να ενεργοποιούν την ανάγνωση, διότι η αντιστοίχηση είναι συνεχής: το κύκλωμα της μνήμης δρα συνέχεια, παρέχοντας το περιεχόμενο της διεύθυνσης εισόδου στην έξοδο. Για τη σύνδεση με μια τέτοια μνήμη, αρκούν από τη μεριά του προγράμματος μια μεταβλητή π.χ. 'addr', για την εξαγωγή της διεύθυνσης, και μια μεταβλητή δείκτη π.χ. 'mem', για την εισαγωγή των δεδομένων. Μέσα στον κώδικα, αρκεί να τεθεί μια τιμή στη μεταβλητή addr, και στη συνέχεια να διαβαστεί η τιμή της mem:

```
int *mem;
int addr;

void main()
{
  int m0, m1, m5;

  /* Παράδειγμα ανάγνωσης Mem[0], Mem[1], Mem[5] */
  addr = 0;
  par { m0 = *mem; addr++; }
  par { m1 = *mem; addr = 5; }
  m5 = *mem;
}
```

Στο προηγούμενο πρόγραμμα, διαβάζονται οι θέσεις 0, 1 και 5 της μνήμης στις αντίστοιχες μεταβλητές m0, m1 και m5. Για κάθε ανάγνωση γράφεται πρώτα η μεταβλητή addr, και ύστερα διαβάζεται η μεταβλητή mem. Χάρη στο χρονισμό της ανάθεσης, μπορεί ταυτόχρονα να προετοιμαστεί η επόμενη διεύθυνση addr στον ίδιο κύκλο που διαβάζεται η προηγούμενη διεύθυνση. Ο τρόπος σύνδεσης της μνήμης με το κύκλωμα του προγράμματος φαίνεται στο σχήμα 54.



Εικόνα 54 – Ένα παράδειγμα διασύνδεσης με μνήμη ROM

Είναι φανερό ότι τα σήματα data και write_back της μεταβλητής mem δε χρησιμοποιούνται, αφού η μνήμη δεν είναι εγγράψιμη (και έτσι δεν πραγματοποιούνται και αναθέσεις της μεταβλητής mem στο πρόγραμμα).

Στην περίπτωση σύνδεσης με εγγράψιμη μνήμη RAM, ο προγραμματιστής μπορεί να χρησιμοποιήσει διάφορες τεχνικές. Συνήθως αρκεί πάλι μια καθολική μεταβλητή addr, και ένας καθολικός δείκτης mem. Η μεταβλητή addr συμβολίζει την τρέχουσα διεύθυνση, της οποίας τα περιεχόμενα εξάγωνται συνεχώς από τη μνήμη πίσω στο πρόγραμμα μέσω του σήματος val του δείκτη mem, όπως δηλαδή και στο προηγούμενο παράδειγμα. Όταν το πρόγραμμα θέλει να γράψει μια τιμή στην τρέχουσα διεύθυνση, δεν έχει παρά να πραγματοποιήσει ανάθεση στη μεταβλητή mem. Με άλλα λόγια, τελικά αρκεί να γίνει μια επέκταση της προηγούμενης περίπτωσης, με τα σήματα data και write_back να μεταδίδονται στη μνήμη. Όταν το σήμα write_back ενεργοποιηθεί, η μνήμη θα γράψει στην τρέχουσα διεύθυνση τα δεδομένα που φέρει το σήμα data. Για να διευκολυνθεί κάτι τέτοιο θα πρέπει η ψηφίδα της μνήμης να διαθέτει ξεχωριστή θύρα για τα δεδομένα εγγραφής από τα δεδομένα ανάγνωσης, αν και σε κάθε περίπτωση μπορούν να βρεθούν εύκολοι τρόποι για τη διασύνδεση περιφερειακών και την άνετη χρήση τους μέσα από το πρόγραμμα.

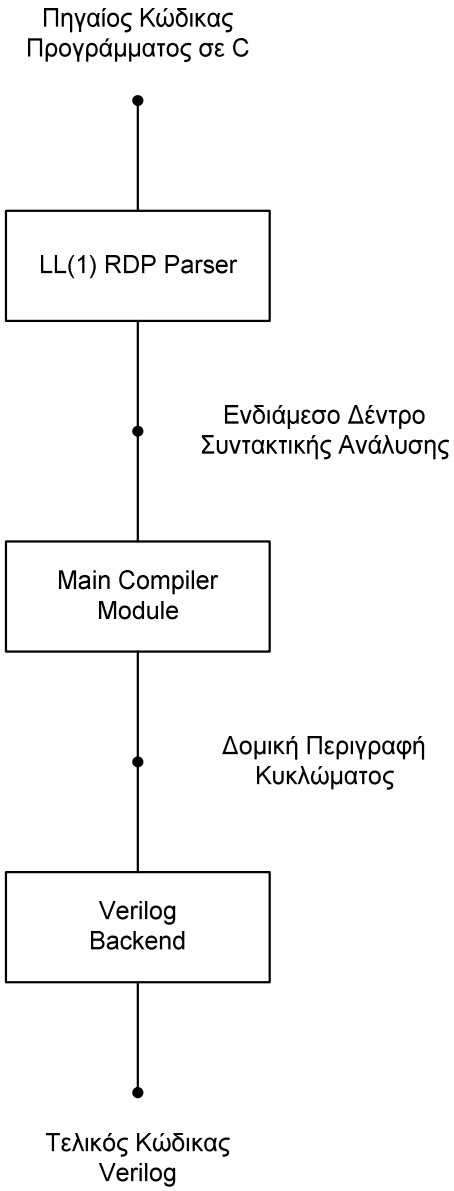
5

Η υλοποίηση

Κάποιος ἠρώτησε τὸν Ἀνάξανδρο τοῦ Εὐρυκράτους, γιατὶ οἱ Σπαρτιᾶτες δὲν συγκεντρώνουν χρήματα σὲ δημόσιο ταμεῖο, καὶ ἀπήντησε: «Γιὰ νὰ μὴ διαφθείρωνται οἱ φύλακές του».

Γενικά. Η δομή του μεταγλωττιστή.

Η ενότητα αυτή ασχολείται με την υλοποίηση του μεταγλωττιστή. Ένα μπλοκ διάγραμμα σχετικά με τη δομή και τον τρόπο λειτουργίας του μεταγλωττιστή παρουσιάζεται στην εικόνα 55.



Εικόνα 55 – Η δομή και ο τρόπος λειτουργίας του μεταγλωττιστή

Η υλοποίηση του μεταγλωττιστή χρησιμοποιεί ακριβώς αυτή τη δομή, και όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, είναι γραμμένη σε C. Το συνολικό μέγεθος του πηγαίου κώδικα που γράφτηκε για την υλοποίηση ξεπερνά τα 100kb (μη συμπεριλαμβανομένου του κώδικα που παράγεται από τον RDP). Είναι πρόδηλο ότι ένα τόσο μεγάλο και σύνθετο πρόγραμμα δε θα ήταν πρακτικό να περιγραφεί με μεγάλη σχολαστικότητα, καθώς αυτό που έχει μεγαλύτερη σημασία είναι μια συνολική παρουσί-

αση μέσα από τα κυριότερα σημεία, με έμφαση σε ουσιώδεις λεπτομέρειες. Στη συνέχεια εξετάζονται τα διάφορα μέρη του μεταγλωττιστή, όπως παρουσιάζονται στο παραπάνω σχήμα.

Το πρόσθιο μέρος (front-end).

Όπως δείχνει το διάγραμμα, ο μεταγλωττιστής ξεκινά λαμβάνοντας το αρχείο εισόδου με τον πηγαίο κώδικα του προγράμματος C το οποίο προορίζεται για μεταγλώττιση. Το πρόσθιο κομμάτι (front-end) του μεταγλωττιστή είναι υπεύθυνο για την ανάγνωση της εισόδου και την πραγματοποίηση της συντακτικής ανάλυσης. Το κομμάτι αυτό κατασκευάστηκε με χρήση του λογισμικού 'RDP' που περιγράφηκε σε προηγούμενη ενότητα. Ο RDP είναι ένας γεννήτορας συντακτικών αναλυτών για LL(1) γλώσσες. Λαμβάνει ως είσοδο τη BNF μορφή της LL(1) γλώσσας που υποστηρίζει ο μεταγλωττιστής (δηλαδή της τροποποιημένης γλώσσας C) και παράγει άπαξ ένα συντακτικό αναλυτή, ειδικά κατασκευασμένο για την ανάλυση αρχείων γραμμένα στη συγκεκριμένη γλώσσα. Το μέρος της εικόνας 55 που φέρει τον τίτλο 'LL(1) RDP Parser' είναι ακριβώς αυτός ο συντακτικός αναλυτής που δημιουργήθηκε με τη βοήθεια του RDP. Ο παραγόμενος κώδικας περιλαμβάνει ένα ολοκληρωμένο front-end για το μεταγλωττιστή, δηλαδή ο RDP κατασκευάζει κώδικα για την ανάγνωση του αρχείου εισόδου, για τη δυνατότητα εισαγωγής διακοπτών στη γραμμή εντολών (command-line switches), για την αναγνώριση των λεκτικών, και φυσικά για τη συντακτική ανάλυση. Έτσι, ήδη ο κώδικας του πρόσθιου μέρους είναι αρκετός για ένα αύταρκες εργαλείο που μπορεί να διαβάζει C αρχεία και να δημιουργεί το αντίστοιχο δέντρο συντακτικής ανάλυσης στη μνήμη.

Ένα παραδειγματικό απόσπασμα από το αρχείο BNF φαίνεται παρακάτω. Το κομμάτι αυτό αφορά τη σύνταξη μιας εντολής (statement) της γλώσσας C, όπως αυτή έχει περιγραφεί σε προηγούμενο κεφάλαιο.

```
stat ::= 'case'^^ initializer ':'^
       | 'default'^^ ':'^
       | seq^^
       | par^^
       | 'if'^^ '('^ exp ')'^ stat [ 'else'^ stat ]
       | 'switch'^^ '('^ exp ')'^ seq
       | 'while'^^ '('^ exp ')'^ stat
       | 'do'^^ stat 'while'^ '('^ exp ')'^ ';'^
       | 'for'^^ '('^ [f_init] ';'^ [f_cond] ';'^ [f_trans] ')'^ stat
       | 'continue'^^ ';'^
       | 'break'^^ ';'^
       | 'return'^^ [ exp ] ';'^
       | ';'^^
       | stat_exp^^ ';'^ .

seq ::= '{'^ { stat } '}'^ .
par ::= 'par'^^ '{'^ { stat } '}'^ .

f_init ::= exp .
f_cond ::= exp .
f_trans ::= exp .

stat_exp ::= exp .
```

Στο απόσπασμα αυτό, φαίνονται οι εντολές if, switch, while, do, for, continue, break και return, η κενή εντολή, οι ετικέτες case και default, καθώς και τα σειριακά (seq) και παράλληλα (par) μπλοκ εντολών. Τα σύμβολα 'exp' και 'initializer' αφορούν παραστάσεις και σταθερές αντίστοιχα και αναλύονται σε άλλο σημείο του αρχείου BNF.

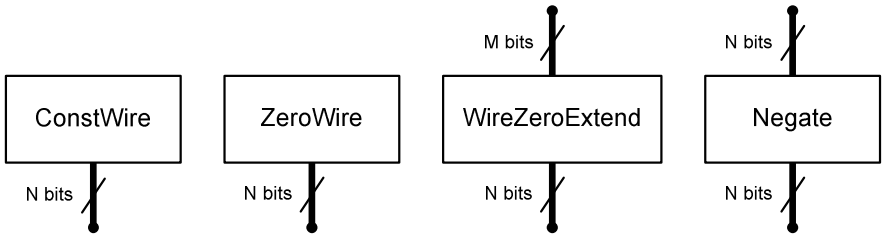
Το κύριο μέρος της μεταγλώττισης. Η δομική περιγραφή με αντικείμενα.

Το αποτέλεσμα της συντακτικής ανάλυσης, όπως αναφέρθηκε, είναι ένα δέντρο. Κατά την ολοκλήρωση της ανάγνωσης του αρχείου εισόδου, αφού κατασκευαστεί ολόκληρο το συντακτικό δέντρο που αποτελεί και την ενδιάμεση μορφή του μεταγλωττιστή, καλείται μια ρουτίνα για την περαιτέρω μεταγλώττιση. Η διαδικασία αυτή, που στην εικόνα 55 σημειώνεται ως 'Main Compiler Module' αποτελεί το κύριο μέρος του μεταγλωττιστή. Στο μέρος αυτό αναλύεται το δέντρο με αναδρομικό τρόπο, και πραγματοποιείται καθαυτή η μεταγλώττιση και η μετατροπή σε κυκλώματα. Πιο συγκεκριμένα, αρχικά γίνεται ένα πρώτο πέρασμα του δέντρου και σημειώνονται τα ονόματα των καθολικών μεταβλητών καθώς και των συναρτήσεων, ώστε να κατασκευαστεί ο *πίνακας συμβόλων* (symbol table), δηλαδή μια δομή στην οποία αποθηκεύονται πληροφορίες για τα αναγνωριστικά (identifiers) που ορίζονται στο πρόγραμμα. Στη συνέχεια πραγματοποιείται ένα δεύτερο πέρασμα στο οποίο γίνεται η επεξεργασία του σώματος (body) των συναρτήσεων, ώστε να συντεθούν τα αντίστοιχα κυκλώματα. Κατά την ανάλυση του σώματος των εντολών, λαμβάνουν χώρα όλες εκείνες οι διαδικασίες που έχει κατά τα άλλα να επιτελέσει ένας μεταγλωττιστής, πέρα από την τελική σύνθεση. Αυτές είναι για παράδειγμα η αναζήτηση και επίλυση των αναγνωριστικών στον πίνακα συμβόλων, ο έλεγχος ορθότητας τύπων, ο έλεγχος συμβατότητας των τελεστών με τα διάφορα είδη των παραστάσεων, ο έλεγχος για επιτρεπτούς συνδυασμούς εντολών (καθώς απαγορεύεται π.χ. η εντολή return σε ένα par μπλοκ, ή η εντολή continue εκτός ενός for/while/do loop, κ.α.), και γενικά οι διαδικασίες που συντελούν στη σημασιολογική επαλήθευση του προγράμματος.

Η διαδικασία της σύνθεσης απαιτείται να γίνεται με τέτοιο τρόπο ώστε να υπάρχει ανεξαρτησία από την τελική γλώσσα περιγραφής υλικού. Έτσι, το αποτέλεσμα αυτού του σταδίου της μεταγλώττισης είναι μια δομική περιγραφή του κυκλώματος, με τη μορφή 'αντικειμένων'. Ακριβέστερα, η διαδικασία της σύνθεσης του κυκλώματος μιας συνάρτησης παράγει μια λίστα από αντικείμενα για την εν λόγω συνάρτηση. Τα αντικείμενα αυτά είναι λειτουργικές οντότητες που θα απαρτίζουν το τελικό κύκλωμα, και χρησιμοποιούνται με αυτό τον τρόπο για την περιγραφή της δομής του κυκλώματος σε υψηλό επίπεδο. Δεν πρόκειται για προϊόντα αντικειμενοστραφούς προγραμματισμού (εξάλλου η γλώσσα που χρησιμοποιείται για την υλοποίηση του μεταγλωττιστή είναι η απλή C), αλλά για εγγραφές που περιλαμβάνουν έναν τύπο (που συμβολίζει τον τύπο του αντικειμένου) και μια σειρά από σήματα εισόδου / εξόδου. Για οποιαδήποτε λειτουργία απαιτείται να περιγραφεί στην τελική γλώσσα, εισάγεται και ένα αντικείμενο αντίστοιχου τύπου στη λίστα των αντικειμένων της συνάρτησης, με τις ανάλογες εισόδους και εξόδους.

Υπάρχουν πολλά είδη αντικειμένων, ώστε να μπορούν να περιγραφούν όλες οι λειτουργίες που μπορούν να απαιτηθούν από την τελική γλώσσα περιγραφής. Για παράδειγμα, το αντικείμενο ConstWire (που έχει μόνο σήμα εξόδου) χρησιμοποιείται για την εξαγωγή ενός σήματος που έχει σταθερή τιμή, η οποία δίνεται ως παράμετρος (πεδίο) στην εγγραφή του αντικειμένου στη λίστα. Επίσης, το αντικείμενο ZeroWire χρησιμοποιείται για την εξαγωγή ενός μηδενικού σήματος, ενώ το αντικείμενο

WireZeroExtend (που λαμβάνει ένα σήμα και εξάγει ένα άλλο) αντιπροσωπεύει την επέκταση του σήματος εισόδου με μηδενικά. Ακόμη, το αντικείμενο Negate συμβολίζει τον υπολογισμό του συμπληρώματος ως προς 2 ενός σήματος. Αυτά τα ενδεικτικά αντικείμενα φαίνονται στην εικόνα 56. Η πλήρης λίστα των αντικειμένων υπάρχει στον πηγαίο κώδικα της υλοποίησης.



Εικόνα 56 – Μερικά ενδεικτικά αντικείμενα

Τα σήματα εισόδου και εξόδου των αντικειμένων προσαρτώνται σε μια συνδεδεμένη λίστα και περιλαμβάνουν πλήθος πληροφοριών για τον τύπο τους, το μέγεθός τους και το όνομά τους, που στη συγκεκριμένη υλοποίηση κωδικοποιείται με έναν αύξοντα αριθμό. Έτσι, το πρώτο σήμα που χρησιμοποιείται σε ένα κύκλωμα είναι το #1, το επόμενο σήμα που θα χρειαστεί ονομάζεται #2, κ.ο.κ. Τα κατασκευαζόμενα αντικείμενα μπορούν να θεωρηθούν ως συνδετικοί κόμβοι των σημάτων, που επιτελούν και διάφορες λειτουργίες.

Το οπίσθιο μέρος.

Στο επόμενο και τελικό στάδιο, τα αντικείμενα που έχουν κατασκευαστεί για το σώμα κάθε συνάρτησης μεταφράζονται με κάποιο συγκεκριμένο τρόπο σε κώδικα της τελικής γλώσσας περιγραφής υλικού, ανάλογα με την εκάστοτε γλώσσα. Έτσι ο μεταγλωττιστής μπορεί να χρησιμοποιηθεί με οποιαδήποτε γλώσσα περιγραφής υλικού, και να παραχθεί η αντίστοιχη έξοδος, αρκεί να γραφεί μια βιβλιοθήκη που θα μετατρέπει τα υψηλού επιπέδου αντικείμενα σε αντίστοιχο κώδικα, και να συνδεθεί ως οπίσθιο μέρος (back-end) με το κύριο module του μεταγλωττιστή. Στη συγκεκριμένη υλοποίηση του μεταγλωττιστή χρησιμοποιήθηκε η γλώσσα Verilog, δηλαδή γράφτηκαν οι αντιστοιχίες των αντικειμένων σε αυτή τη γλώσσα περιγραφής υλικού. Για κάθε σήμα δημιουργείται το αντίστοιχο wire της Verilog, με όνομα W_id, όπου id ο αύξων αριθμός του σήματος. Όμοια, για κάθε αντικείμενο εισάγεται ο κώδικας Verilog που αντιστοιχεί στο αντικείμενο. Για παράδειγμα, έστω ότι κάποιο από τα τέσσερα αντικείμενα της εικόνας 56 εμφανίζεται με είσοδο το σήμα υπ' αριθμ. #7 μεγέθους 2 bits (αν πρόκειται για αντικείμενο που λαμβάνει είσοδο), και έξοδο το σήμα υπ' αριθμ. #8 με μέγεθος 4 bits. Ο κώδικας που θα γραφτεί στην έξοδο θα είναι κάποια από τις παρακάτω αντίστοιχες γραμμές (υποθέτοντας ότι η σταθερά του ConstWire είναι π.χ. η δυαδική σταθερά '1011'):

```
assign W_8 = 'b1011 ;
assign W_8 = 'b0 ;
assign W_8 = {2'b0, W_7} ;
assign W_8 = -W_7 ;
```

Με άλλα λόγια, η εμφάνιση της δυαδικής σταθεράς '1011' μέσα σε ένα πρόγραμμα εισάγει ένα αντικείμενο ConstWire το οποίο γράφεται ως ανάθεση ενός σήματος W_n

στον τελικό κώδικα. Το σήμα αυτό θα χρησιμοποιηθεί ως η τιμή της σταθεράς, όπου απαιτηθεί. Αντίστοιχα, όποτε ένα σήμα 2 bits χρειάζεται να επεκταθεί με μηδενικά ως στα 4 bits (λόγω π.χ. της διαδικασίας προσαρμογής τύπου) κατασκευάζεται ένα ανάλογο αντικείμενο WireZeroExtend το οποίο μεταφράζεται στον κώδικα της τρίτης γραμμής του παραπάνω παραδείγματος, δηλαδή γίνεται χρήση του τελεστή παράθεσης της Verilog.

Εκτός από τα αντικείμενα της κάθε συνάρτησης, τα οποία αναφέρονται στις εντολές του σώματός (body) αυτής, στην τελική γλώσσα περιγραφής υλικού χρειάζεται να παραχθούν και τα υπόλοιπα κομμάτια του κυκλώματος, δηλαδή οι επιλογείς και οι καταχωρητές για τις μεταβλητές, τα κυκλώματα ελέγχου των σκελετών των συναρτήσεων, και φυσικά η διασύνδεση του τελικού κυκλώματος με το εξωτερικό περιβάλλον. Για όλα αυτά το οπίσθιο μέρος γνωρίζει τις πληροφορίες που έχουν αποθηκευτεί από το κύριο μέρος της μεταγλώττισης σε ειδικές δομές που συνοδεύουν τον πίνακα συμβόλων. Για παράδειγμα, κάθε αναγνωριστικό συνάρτησης που είναι αποθηκευμένο στον πίνακα συμβόλων περιλαμβάνει και μια συνδεδεμένη λίστα με τις καταστάσεις κλήσης (που παρουσιάστηκαν στην εικόνα 14), όπως και μια λίστα με τις καταστάσεις εισόδου και εξόδου του κολλώδη ανανεωτή (αν αυτός υπάρχει), κλπ. Αντίστοιχα, κάθε μεταβλητή περιλαμβάνει στοιχεία όπως τα ζεύγη τιμών-καταστάσεων που απαιτεί ο επιλογέας της, το μέγεθος του καταχωρητή της (ή ανάλογες πληροφορίες για την περίπτωση που είναι παράμετρος κατ' αναφορά ή γενικά μεταβλητή τύπου δείκτη, κλπ). Αυτές οι πληροφορίες δεν σχετίζονται με τα αντικείμενα που κατασκευάζονται για κάθε συνάρτηση: η λίστα των αντικειμένων αφορά μόνο τις εντολές του σώματος της κάθε συνάρτησης, και όχι τα υπόλοιπα αποθηκευτικά κυκλώματα ή τα κυκλώματα ελέγχου που περιβάλλουν το σώμα της συνάρτησης. Το οπίσθιο μέρος είναι ελεύθερο να παράγει τον τελικό κώδικα που αφορά τα υπόλοιπα αυτά κυκλώματα (εκτός δηλαδή από τα αντικείμενο), με τον δικό του τρόπο, ανάλογα με την εκάστοτε γλώσσα περιγραφής που χρησιμοποιείται. Για την παρούσα υλοποίηση, ο συνολικός τρόπος περιγραφής σε Verilog αναφέρεται αργότερα.

Η βελτιστοποίηση (optimization).

Το κύριο στάδιο της μεταγλώττισης, που όπως αναφέρθηκε νωρίτερα συνθέτει τη δομική περιγραφή του κυκλώματος από το συντακτικό δέντρο, είναι και αυτό στο οποίο μπορεί να υπεισέλθει η διαδικασία της *βελτιστοποίησης* (optimization). Η βελτιστοποίηση μπορεί να δράσει σε διάφορα επίπεδα, παράγοντας ένα μικρότερο κύκλωμα στα σημεία που αυτό είναι δυνατό. Για παράδειγμα, μπορεί να διαγνώσει εντολές που δεν πρόκειται ποτέ να εκτελεστούν και άρα να παραλείψει τη μετάφρασή τους, ή να διακρίνει παραστάσεις των οποίων κάποιες πράξεις μπορούν να αναδιαταχθούν ή να εκτελεστούν με γρηγορότερο και ευκολότερο τρόπο, αν το αποτέλεσμα τους έχει συγκεκριμένη μορφή λόγω κάποιων ιδιοτήτων. Μία τέτοια βελτιστοποίηση πραγματοποιείται και στην παρούσα υλοποίηση. Πιο συγκεκριμένα, ο αναλυτής των παραστάσεων (expression parser), δηλαδή το κομμάτι του μεταγλωττιστή που αναλύει τις συντακτικό δέντρο των εκφράσεων και παράγει τα αντίστοιχα αντικείμενα, απλοποιεί κάποιες πράξεις με χρήση μιας ιδιαίτερης πληροφορίας για τα ορίσματα τους. Η πληροφορία αυτή αφορά το πλήθος των λιγότερο σημαντικών (δεξιότερων) bits τα οποία είναι μηδενικά, και συμβολίζεται με 'rzb' (right zero bits). Για παράδειγμα, ο αριθμός rzb είναι γνωστός για όλες τις σταθερές· π.χ. η σταθερά 8 έχει rzb=3, ενώ η σταθερά "F9A0" έχει rzb=5. Αλλά και για το αποτέλεσμα πράξεων από τελεστές ο αριθμός αυτός μπορεί να υπολογιστεί· π.χ. σε ένα άθροισμα δύο παραστάσεων A + B, το απο-

τέλεσμα έχει rzb ίσο με το μικρότερο των rzb των δύο ορισμάτων, αφού με την άθροιση παραμένουν τα μηδενικά στο δεξιό άκρο του αποτελέσματος. Έτσι λοιπόν κατά την ανάλυση μιας παράστασης μπορούν να υπολογιστούν οι ενδιάμεσοι αριθμοί rzb (όποτε βέβαια αυτό είναι δυνατό, αφού π.χ. για τις μεταβλητές οι αριθμοί αυτοί δεν είναι γνωστοί κατά τη μεταγλώττιση) και να χρησιμοποιηθούν ώστε να απλοποιηθούν κάποιες πράξεις. Η απλοποίηση αυτή μπορεί να γίνει σε πολλές περιπτώσεις. Σε μια άθροιση απρόσημων ποσοτήτων, αν το ένα όρισμα έχει αριθμό rzb μεγαλύτερο από το συνολικό μέγεθος του άλλου ορίσματος, τότε δε χρειάζεται να γίνει άθροιση, παρά αρκεί μια αντικατάσταση, όπως στο παρακάτω παράδειγμα:

```
unsigned bit<8> a, bit<4> b;

a = b + 80;
```

Στο παράδειγμα αυτό, δεν είναι αναγκαίο να παραχθεί ένα αντικείμενο αθροιστή για την πράξη 'b + 80', παρά αρκεί η αντικατάσταση των τεσσάρων δεξιότερων bits της δεκαδικής σταθεράς '80' με το περιεχόμενο της μεταβλητής b.

Μια άλλη πιο ενδιαφέρουσα περίπτωση βελτιστοποίησης με χρήση του αριθμού rzb είναι η ολίσθηση. Κατά την αριστερή ολίσθηση, όποιο κι αν είναι το όρισμα που ολισθαίνει, είναι γνωστό ότι το αποτέλεσμα αποκτά μηδενικά δεξιά bits και έτσι μπορεί να υπολογιστεί ο αριθμός rzb. Με αυτό τον τρόπο είναι εύκολο να απλοποιηθούν κάποιες πράξεις όταν έχει προηγηθεί ολίσθηση σε μια παράσταση, και έτσι να πραγματοποιηθούν κάποιες συνήθεις διεργασίες με βέλτιστο τρόπο. Για παράδειγμα, συχνά μπορεί να απαιτηθεί σε ένα πρόγραμμα να συνενωθούν δύο παραστάσεις, παράγοντας μια τρίτη που είναι η παράθεσή τους. Αυτό μπορεί να γίνει εύκολα ως εξής:

```
bit<8> a, b;

s = ( (bit<16>)a << 8 ) | b;
```

Στο παράδειγμα αυτό, η εκτεταμένη σε 16 bits μεταβλητή a ολισθαίνει 8 θέσεις και υφίσταται bitwise OR με τη μεταβλητή b. Η τελευταία αυτή πράξη δεν απαιτεί πραγματικές πύλες OR αφού είναι γνωστό ότι μετά την ολίσθηση υπάρχουν 8 μηδενικά δεξιά bits. Έτσι, αρκεί η μεταβλητή b να αντικαταστήσει το δεξιό μέρος, και να παρατεθεί δίπλα στη μεταβλητή a. Για το σκοπό αυτό έχουν εισαχθεί ειδικά αντικείμενα στη συλλογή των δυνατών δομικών αντικειμένων που μπορεί να παράγει ο μεταγλωττιστής, όπως το αντικείμενο RightReplace που αντιπροσωπεύει τη δεξιά αντικατάσταση ενός μέρους μιας παράστασης με κάποια άλλη.

Ένα μικρό παράδειγμα μεταγλώττισης μιας εντολής.

Για να γίνει περισσότερο κατανοητή η συνολική διαδικασία της μεταγλώττισης, θα δοθεί ένα μικρό παράδειγμα μεταγλώττισης μιας εντολής, το οποίο θα αναλυθεί για όλα τα στάδια. Έστω λοιπόν η παρακάτω εντολή, που εμφανίζεται σε κάποιο σημείο ενός προγράμματος:

```
x = -y ^ 9;
```

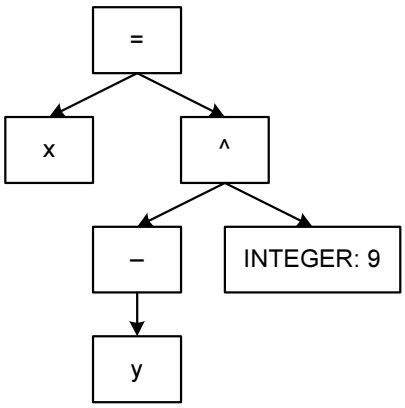
Ας υποτεθεί ότι οι μεταβλητές x και y είναι προσημασμένες, μεγέθους οκτώ και τεσσάρων bits αντίστοιχα. Ακριβέστερα, μπορεί να είναι απλές καθολικές ή τοπικές με-

ταβλητές, ή παράμετροι κατ' αξία της συνάρτησης, αλλά η προέλευσή τους δεν παίζει σημαντικό ρόλο για τις ανάγκες του παραδείγματος. Το πρόσθιο μέρος του μεταγλωττιστή διαβάζει το αρχείο εισόδου και συναντά τον κώδικα του παραδείγματος που για πληρότητα ξαναγράφεται στην εικόνα 57.

```
x = -y ^ 9;
```

Εικόνα 57 – Ο πηγαίος κώδικας του παραδείγματος

Ο κώδικας αυτός αναλύεται, σύμφωνα με τη σύνταξη BNF της γλώσσας C, στο συντακτικό υπόδεντρο της εικόνας 58.

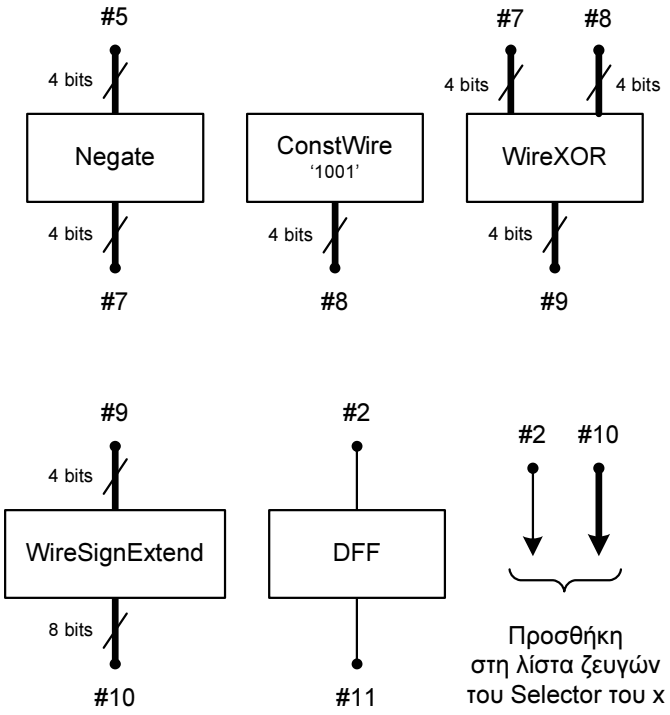


Εικόνα 58 – Το δέντρο συντακτικής ανάλυσης του παραδείγματος

Στη συνέχεια εκτελείται το κύριο μέρος του μεταγλωττιστή. Πριν ο μεταγλωττιστής φθάσει το σημείο εκείνο του δέντρου που εμφανίζεται η εντολή, θα έχει ήδη περάσει από τα σημεία των δηλώσεων των μεταβλητών x και y, όπου και θα έχει κατασκευάσει καταχωρήσεις στον πίνακα συμβόλων με πληροφορίες για το είδος των μεταβλητών, το μέγεθός τους, τα μέχρι στιγμής ζεύγη τιμών-καταστάσεων του επιλογέα τους, τον αύξοντα αριθμό του σήματος τιμής που εξάγεται από τους καταχωρητές τους, κλπ. Επίσης, όταν ο μεταγλωττιστής φτάσει την εντολή του παραδείγματος, θα βρίσκεται σε κάποια κατάσταση εισόδου για την εντολή αυτή.

Έστω λοιπόν ότι η μεταβλητή y εξάγεται στο σήμα #5, ότι η κατάσταση ελέγχου στην οποία βρίσκεται ο μεταγλωττιστής έχει αύξοντα αριθμό #2, και ότι ο επόμενος μη χρησιμοποιούμενος μέχρι στιγμής αριθμός σήματος είναι ο #7. Ο μεταγλωττιστής θα συνθέσει τα αντικείμενα που φαίνονται στην εικόνα 59. Συγκεκριμένα, αρχικά θα αναλύσει την παράσταση με χρόνο που βρίσκεται στο δεξί μέρος της ανάθεσης. Η παράσταση αυτή δεν έχει κλήση συνάρτησης και έτσι δε μεταβάλλει το χρόνο, ενώ τα αντικείμενα της είναι τα Negate, ConstWire, WireXOR και WireSignExtend, με αυτό το τελευταίο να χρησιμοποιείται για την επέκταση του αποτελέσματος στο μέγεθος της μεταβλητής x που υφίσταται την ανάθεση. Το τελικό αποτέλεσμα της παράστασης είναι η τιμή του σήματος με αύξοντα αριθμό #10, και η κατάσταση #2 που έχει παραμείνει η ίδια και η οποία δεν απαιτεί κάποιο ιδιαίτερο κύκλωμα. Σύμφωνα λοιπόν με το κύκλωμα του βασικού τελεστή ανάθεσης (που παρουσιάστηκε στην εικόνα 38), το ζεύγος της τιμής #10 και της κατάσταση #2 θα εισαχθεί στη συνδεδεμένη λί-

στα που κρατά εσωτερικά ο μεταγλωττιστής για τον επιλογέα της μεταβλητής x (και το οποίο έχει καταχρηστικά σημειωθεί ως αντικείμενο στην εικόνα 59). Επίσης, η κατάσταση #2 θα καθυστερηθεί με ένα DFF, παράγοντας την τελική κατάσταση εξόδου #11.



Εικόνα 59 – Τα αντικείμενα του παραδείγματος

Τα αντικείμενα της εικόνας 59 περνούν κατόπιν στο τελικό στάδιο της μεταγλώττισης που αφορά την παραγωγή του τελικού κώδικα. Εκτός από τη συλλογή των αντικειμένων κάθε συνάρτησης, στο οπίσθιο μέρος (back-end) δίνονται και όλες οι πληροφορίες για τις μεταβλητές του προγράμματος, τους επιλογείς τους με τα αντίστοιχα ζεύγη, κλπ. Δηλαδή, το τελικό μέρος είναι εφοδιασμένο με τον πίνακα συμβόλων που περιέχει όλες τις πληροφορίες που απαιτούνται για την κατασκευή του κυκλώματος, εκτός από τη δομική περιγραφή του κυκλώματος κάθε συνάρτησης που πραγματοποιείται μέσω των αντικειμένων. Έτσι, για τις μεταβλητές x και y, περιγράφονται σε γλώσσα Verilog τα κυκλώματα του επιλογέα και του καταχωρητή, καθώς και τα σήματα που συνδέονται με αυτούς. Ακόμα δεν έχει αναφερθεί συστηματικά ο τρόπος που γράφονται τα κυκλώματα αυτά στη Verilog, αλλά θα δοθεί εδώ ένα πρώτο παράδειγμα για το κύκλωμα της μεταβλητής x. Αν υποτεθεί λοιπόν ότι η μεταβλητή x δεν υφίσταται άλλη ανάθεση στο πρόγραμμα, τότε ο επιλογέας της θα περιέχει μόνο ένα ζεύγος, και το συνολικό κύκλωμα της εικόνας 12 θα εισαχθεί με τον ακόλουθο κώδικα παραγωγής αντιτύπου (instantiation) της Verilog:

```
// unsigned bit<8> x
regselector1 #(8) regselector1_1 (W_6, clk, reset, 8'b0, W_10, W_2);
```

Το module 'regselector1' συμβολίζει ένα κύκλωμα επιλογέα μιας εισόδου, με καταχωρητή μεγέθους N το οποίο μεταδίδεται ως παράμετρος (εδώ είναι 8 bits). Τα ορίσματα που δίδονται κατά την εισαγωγή του αντιτύπου είναι το σήμα εξόδου, το σήμα του ρολογιού και της αρχικοποίησης (reset), μια τιμή αρχικοποίησης, και τα ζεύγη

των τιμών-καταστάσεων, που όπως αναφέρθηκε περιορίζονται στο ζεύγος των σημάτων με αύξοντες αριθμούς #10 και #2.

Κατά τα άλλα, όταν το τελικό στάδιο φθάσει στα αντικείμενα της εντολής του παραδείγματος, δηλαδή όταν έρθει η στιγμή να μεταφραστούν τα αντικείμενα της εικόνας 59 που εμπλέκουν καθαυτή την εκτέλεση της ανάθεσης, θα παραχθεί ο κώδικας της εικόνας 60.

```
assign W_7 = -W_5;
assign W_8 = 'b1001;
assign W_9 = W_7 ^ W_8;
assign W_10 = {{4{W_9[3]}}, W_9};
dff dff_3 (w_11, clk, reset, w_2);
```

Εικόνα 60 – Ο τελικός κώδικας Verilog του παραδείγματος

Στον κώδικα αυτό, τα σήματα #7, #8, #9 και #10 που παράγονται από τα αντικείμενα Negate, ConstWire, WireXOR και WireSignExtend (δηλαδή απλά συνδυαστικά κυκλώματα) περιγράφονται απευθείας με ανάλογο τρόπο με τους συνδυαστικούς τελεστές της Verilog, ενώ το DFF εισάγεται ως instance του αντίστοιχου module.

Η υλοποίηση ορισμένων κυκλωμάτων σε Verilog.

Στην ίδια κατηγορία με το DFF, δηλαδή στην κατηγορία των κυκλωμάτων εκείνων που περιγράφονται ως ξεχωριστά modules και στον τελικό κώδικα εισάγονται τα αντίτυπά τους στα αντίστοιχα σημεία, ανήκουν και κάποια άλλα κυκλώματα. Τα κυκλώματα αυτά συνολικά είναι τα εξής:

- Το DFF (dff).
- Ο απλός συγχρονιστής (sync), παράδειγμα του οποίου δόθηκε στην εικόνα 26.
- Ο συγχρονιστής κλήσης συνάρτησης (funcsync), που περιγράφηκε παραδειγματικά στην εικόνα 52.
- Ο επιλογέας των M ζευγών με ενσωματωμένο καταχωρητή (regselectorM), ο οποίος αποτελεί το κύκλωμα των αποθηκευτικών μονάδων, όπως δείχνουν και οι εικόνες 12, 16, 17 και 23.
- Ο σκέτος επιλογέας των M ζευγών (selectorM), ο οποίος χρησιμοποιείται στα κυκλώματα των δεικτών, όπως δείχνουν και οι εικόνες 19 και 20.
- Ο κολλώδης ανανεωτής M καταστάσεων (stickyupdateM), που παρουσιάστηκε στην εικόνα 21.

Τα τρία τελευταία κυκλώματα έχουν μια συγκεκριμένη ιδιαιτερότητα: δεν μπορούν να περιγραφούν θέτοντας το M ως παράμετρο του module της Verilog, διότι από την παράμετρο αυτή εξαρτάται το πλήθος κάποιων εσωτερικών στοιχείων τους. Επειδή

λοιπόν η Verilog δεν υποστηρίζει τέτοιου είδους παραμετρική σύνθεση, πρέπει για κάθε διαφορετικό M να γραφεί και ο αντίστοιχος κώδικας του module. Για παράδειγμα, αν σε ένα πρόγραμμα υπάρχουν μεταβλητές κατ' αναφορά (των οποίων το κύκλωμα ως γνωστόν εμπλέκει σκέτους επιλογείς χωρίς καταχωρητή), απαιτούνται κάποια κυκλώματα τύπου selectorM, όπου M συμβολίζει το πλήθος των M διαφορετικών τιμών του επιλογέα. Αν όλες οι παράμετροι κατ' αναφορά του προγράμματος έχουν ακριβώς δυο τιμές επιλογής (διότι π.χ. καλούνται με δύο διαφορετικές τιμές από δύο σημεία του προγράμματος), τότε χρειάζεται μόνο ο επιλογέας τύπου selector2. Ωστόσο, αν κάποια μεταβλητή κατ' αναφορά λαμβάνει για παράδειγμα 3 διαφορετικές τιμές, τότε χρειάζεται να περιγραφούν τόσο ο επιλογέας selector2 όσο και ο selector3. Γενικά, κατά τη μεταγλώττιση δημιουργείται μια λίστα με τα απαραίτητα προς περιγραφή κυκλώματα, των οποίων τα αντίτυπα έχουν εισαχθεί στο κύριο κύκλωμα, και στο τέλος του αρχείου εξόδου παράγονται κατ' απαίτηση οι αντίστοιχες περιγραφές. Αυτό απαιτεί ο κώδικας του οπίσθιου μέρους του μεταγλωττιστή να γνωρίζει πως να παράγει παραμετρικά τον κώδικα περιγραφής ενός επιλογέα για οποιαδήποτε τιμή της παραμέτρου M.

Οι υλοποιήσεις που χρησιμοποιούνται για τον συγχρονιστή (sync), το DFF (dff) και το συγχρονιστή κλήσης συνάρτησης (funcsync) φαίνονται παρακάτω:

```
module sync(sout, clk, reset, sin);
  parameter N = 8;
  output sout;
  input clk, reset;
  input [N-1:0] sin;
  reg [N-1:0] q;
  wire [N-1:0] new_q = q | sin;

  assign sout = &new_q;
  always @(posedge clk)
    q <= new_q & {N{~(sout | reset)}};
endmodule

module dff(sout, clk, reset, sin);
  output sout;
  input clk, reset, sin;
  reg q;

  assign sout = q;
  always @(posedge clk)
    q <= sin & ~reset;
endmodule

module funcsync(sout, clk, reset, sgo, sfunc);
  output sout;
  input clk, reset, sgo, sfunc;
  reg go;
  wire trig = go | sgo;

  assign sout = trig & sfunc;
  always @(posedge clk)
    go <= trig & (~(reset | sfunc));
endmodule
```

Όπως είναι φανερό, ο απλός συγχρονιστής λαμβάνει μια παράμετρο N που συμβολίζει το πλήθος των σημάτων προς συγχρονισμό. Τα σήματα αυτά δίδονται ως το όρι-

σμα sin, ενωμένα σε έναν διάνυσμα από bits. Η έξοδος λαμβάνεται από το σήμα sout, ενώ συμμετέχουν και τα σήματα clk και reset που συμβολίζουν το ρολόι και το μηδενισμό (αρχικοποίηση) αντίστοιχα. Το DFF είναι πιο απλό στη χρήση, και περιλαμβάνει εκτός από τα σήματα clk και reset, τα σήματα sin και sout που αποτελούν την είσοδο και την έξοδο του flip-flop αντίστοιχα. Ο συγχρονιστής κλήσης συνάρτησης είναι εξίσου απλός καθώς έχει συγκεκριμένο αριθμό εισόδων, χωρίς να απαιτεί παραμετροποίηση. Τα σήματα sgo και sfunc συμβολίζουν τα αντίστοιχα σήματα call_state και exit της εικόνας 52.

Μερικά παραδείγματα της χρήσης των παραπάνω κυκλωμάτων μέσα στον τελικό κώδικα παρουσιάζονται παρακάτω:

```
// #14 and #15 are synced to #16
sync #(2) sync_7 (W_16, clk, reset, {W_15, W_14});

// Function call #10 and return #4 are func-synced to #34
funcsync funcsync_12 (W_34, clk, reset, W_10, W_4);
```

Οι επιλογείς με καταχωρητή (regselectorM), που συμμετείχαν και στο μικρό παράδειγμα που δόθηκε για την εντολή ανάθεσης της μεταβλητής x, είναι κυκλώματα των οποίων η περιγραφή, όπως αναφέρθηκε, κατασκευάζεται κατ' απαίτηση για τις διαφορετικές τιμές της παραμέτρου M. Περιλαμβάνουν και μια παράμετρο N (η οποία είναι κανονική παράμετρος της Verilog) που συμβολίζει το μέγεθος σε bits των τιμών του επιλογέα, και άρα το μήκος του αντίστοιχου καταχωρητή. Οι θύρες του module περιλαμβάνουν το σήμα εξόδου wout του καταχωρητή, το ρολόι clk, το σήμα αρχικοποίησης reset, το διάνυσμα αρχικοποίησης iv, και τα ζεύγη των τιμών wrX και καταστάσεων stX, εναλλάξ. Οι υλοποιήσεις για M = 0 (σκέτος καταχωρητής), M = 1 και M = 3 δίνονται στις επόμενες γραμμές.

```
module regselector0(wout, clk, reset, iv);
  parameter N = 8;
  output [N-1:0] wout;
  input clk, reset;
  input [N-1:0] iv;
  reg [N-1:0] q;

  assign wout = q;
  always @(posedge clk)
    q <= reset ? iv : q;
endmodule

module regselector1(wout, clk, reset, iv, wr1, st1);
  parameter N = 8;
  output [N-1:0] wout;
  input clk, reset;
  input [N-1:0] iv, wr1;
  input st1;
  wire wen = ss1;
  wire [N-1:0] val = wr1;
  reg [N-1:0] q;

  assign wout = q;
  always @(posedge clk)
    q <= reset ? iv : (wen ? val : q);
endmodule
```

```
module regselector3(wout, clk, reset, iv, wr1, st1, wr2, st2, wr3, st3);
  parameter N = 8;
  output [N-1:0] wout;
  input clk, reset;
  input [N-1:0] iv, wr1, wr2, wr3;
  input st1, st2, st3;
  wire ss1 = st1;
  wire ss2 = st2 & ~(st1);
  wire ss3 = st3 & ~(st2 | st1);
  wire wen = st1 | st2 | st3;
  wire [N-1:0] val = (wr1 & {N{ss1}})|(wr2 & {N{ss2}})|(wr3 & {N{ss3}});
  reg [N-1:0] q;

  assign wout = q;
  always @(posedge clk)
    q <= reset ? iv : (wen ? val : q);
endmodule
```

Με αντίστοιχο τρόπο κατασκευάζεται και ο απλός επιλογέας (selectorM). Είναι και εδώ παρούσα η παράμετρος N του μεγέθους των τιμών, ενώ οι θύρες του module περιορίζονται στην έξοδο wout και στα ζεύγη των τιμών wrX και των καταστάσεων stX που γράφονται εναλλάξ. Ο μικρότερος δυνατός επιλογέας είναι προφανώς ο selector2, του οποίου η υλοποίηση είναι η εξής:

```
module selector2(wout, wr1, st1, wr2, st2);
  parameter N = 8;
  output [N-1:0] wout;
  input [N-1:0] wr1, wr2;
  input st1, st2;
  wire ss1 = st1;
  wire ss2 = st2 & ~(st1);

  assign wout = (wr1 & {N{ss1}}) | (wr2 & {N{ss2}});
endmodule
```

Τέλος, υπάρχει και το κύκλωμα του κολλώδη ανανεωτή, το οποίο παράγεται παραμετρικά ανάλογα με το πλήθος M των καταστάσεων που απαιτούνται. Εκτός από το ρολόι clk και το σήμα αρχικοποίησης reset, το module δέχεται εναλλάξ τις καταστάσεις εισόδου siX και εξόδου soX. Για δύο καταστάσεις, ο κώδικας είναι ο εξής:

```
module stickyupdate2(clk, reset, si1, so1, si2, so2);
  input clk, reset, si1, si2;
  output so1, so2;
  reg [1:0] q;
  wire ss1 = si1;
  wire ss2 = si2 & ~(si1);
  wire wen = si1 | si2;

  assign so1 = wen ? ss1 : q[0];
  assign so2 = wen ? ss2 : q[1];
  always @(posedge clk)
    q <= reset ? 8'b0 : (wen ? {ss2, ss1} : q);
endmodule
```

Το τελικό αποτέλεσμα.

Στη συνέχεια θα περιγραφεί ο συστηματικός τρόπος με τον οποίο παράγεται η τελική έξοδος στη γλώσσα Verilog. Το αρχείο εξόδου έχει γενικά την παρακάτω δομή:

- Κώδικας περιγραφής των θυρών του module.
- Κώδικας δηλώσεων τοπικών wires.
- Κώδικας που αντιστοιχεί σε κάθε καθολική μεταβλητή.
- Κώδικας που αντιστοιχεί σε κάθε συνάρτηση.
- Κώδικας περιγραφής των δυναμικά κατασκευασμένων δομών (regselectorM, selectorM και stickyupdateM).

Ο κώδικας περιγραφής των θυρών του module δεν είναι τίποτε άλλο από την επικεφαλίδα του module, και τις δηλώσεις input / output για τις θύρες. Οι θύρες ενός module είναι τα σήματα με τα οποία επικοινωνεί με το εξωτερικό περιβάλλον. Εκτός από τα πατροπαράδοτα clk και reset, για το ρολόι και την αρχικοποίηση αντίστοιχα, υπάρχουν και τα εξής σήματα που μετέχουν ως θύρες:

- Για κάθε συνάρτηση X, υπάρχει το σήμα εισόδου func_X που συμβολίζει μια εξωτερικά προερχόμενη κατάσταση κλήσης (call state) για τη συνάρτηση, και το οποίο μπορεί να χρησιμοποιηθεί για να εκκινήσει την εκτέλεση της συνάρτησης.
- Για κάθε απλή (όχι δείκτη) καθολική μεταβλητή X που δεν έχει δηλωθεί ως static, υπάρχει το σήμα εξόδου val_X που εξάγει συνεχώς την τρέχουσα τιμή της μεταβλητής.
- Για κάθε καθολική μεταβλητή X τύπου δείκτη, υπάρχει το σήμα εισόδου din_X και τα σήματα εξόδου dout_X και wout_X, που αντιπροσωπεύουν τα σήματα val, data και write_back αντίστοιχα (εικόνα 20).

Ο κώδικας δηλώσεων τοπικών wires αποτελείται από γραμμές κώδικα στις οποίες δηλώνονται κατά σειρά όλα τα ενδιάμεσα σήματα W_# που χρησιμοποιήθηκαν από το μεταγλωττιστή, με το αντίστοιχο μέγεθός τους. Έτσι, σε αυτό το σημείο ο μεταγλωττιστής σαρώνει τη λίστα ενδιάμεσων σημάτων που έχει δημιουργήσει, και για κάθε αύξοντα αριθμό κατασκευάζει μια αντίστοιχη γραμμή δήλωσης. Έτσι δηλώνονται σταδιακά τα σήματα W_1, W_2, W_3 κ.ο.κ. που χρησιμοποιούνται στο υπόλοιπο του προγράμματος.

Ο κώδικας που αντιστοιχεί σε κάθε καθολική μεταβλητή εξαρτάται από τον τύπο της μεταβλητής αυτής. Έτσι, για τις απλές καθολικές μεταβλητές είναι η εισαγωγή ενός στιγμιοτύπου regselectorM. Επίσης, για τις μεταβλητές αυτές που δεν έχουν δηλωθεί ως static, και άρα απαιτούν σύνδεση με το εξωτερικό περιβάλλον, το σήμα val_X γίνεται assign με το ανάλογο σήμα W_# που εξάγει ο regselector. Για τις καθολικές μεταβλητές τύπου δείκτη ο κώδικας που δημιουργείται είναι η εισαγωγή ενός στιγμιοτύπου selectorM και μιας πύλης OR για τις καταστάσεις (σύμφωνα δηλαδή με την εικόνα 20). Αντίστοιχα, το σήμα dout_X γίνεται assign με το ανάλογο σήμα W_# της εξόδου του selector, και το σήμα wout_X με το σήμα εξόδου της πύλης OR.

Ο κώδικας που αντιστοιχεί σε κάθε συνάρτηση αποτελείται από διάφορα μικρότερα μέρη τα οποία είναι τα εξής:

- Κώδικας εισαγωγής του κολλώδη ανανεωτή.
- Κώδικας που αντιστοιχεί σε κάθε παράμετρο.
- Κώδικας που αντιστοιχεί σε κάθε τοπική μεταβλητή.
- Κώδικας εισαγωγής του κυκλώματος τιμής επιστροφής.
- Κώδικας περιγραφής του σκελετού της συνάρτησης.
- Κώδικας περιγραφής των αντικειμένων του σώματος της συνάρτησης.

Ο κώδικας εισαγωγής του κολλώδη ανανεωτή είναι η εισαγωγή ενός στιγμιοτύπου του module stickyupdateM, ενώ ο κώδικας που αντιστοιχεί σε κάθε παράμετρο είναι αντίστοιχος με αυτόν που παράγεται και για τις καθολικές μεταβλητές. Βέβαια, οι παράμετροι δε συνδέονται με το εξωτερικό περιβάλλον και έτσι ο κώδικας περιλαμβάνει μόνο εισαγωγές στιγμιοτύπων τύπου regselectorM, για τις παραμέτρους κατ' αξία, και στιγμιοτύπων τύπου selectorM (μαζί με πύλες OR) για τις παραμέτρους κατ' αναφορά (ώστε να περιγραφεί το κύκλωμα της εικόνας 19). Εξίσου εισαγωγή στιγμιοτύπων του module regselectorM αποτελεί και ο κώδικας που αντιστοιχεί σε κάθε τοπική μεταβλητή, όπως και στο κύκλωμα της τιμής επιστροφής (αν αυτό βέβαια είναι παρόν, δηλαδή αν η συνάρτηση δεν έχει τύπο επιστροφής void). Για το σκελετό της συνάρτησης περιγράφεται το κύκλωμα της εικόνας 14, δηλαδή εισάγεται μια πύλη OR (ως πρωτογενές στιγμιότυπο της Verilog) με τα σήματα κλήσης της συνάρτησης καθώς και με το εξωτερικό σήμα func_X που αναφέρθηκε πρωτύτερα, και προαιρετικά ένα DFF εισόδου (με το module dff). Για την έξοδο από τη συνάρτηση εισάγεται όμοια μια πύλη OR με τις καταστάσεις εξόδου, και ένα προαιρετικό DFF εξόδου. Το κύριο μέρος της συνάρτησης, που αποτελείται από τα αντικείμενα του σώματος (body), εισάγει αντίστοιχο κώδικα μεταφράζοντας ένα προς ένα τα αντικείμενα που έχουν κατασκευαστεί από το προηγούμενο στάδιο της μεταγλώττισης, όπως αναφέρθηκε και σε προηγούμενο παράδειγμα.

Η έξοδος ολοκληρώνεται με τον κώδικα περιγραφής των δυναμικά κατασκευασμένων δομών (regselectorM, selectorM και stickyupdateM). Τα κυκλώματα αυτά πρέπει να περιγραφούν ξεχωριστά για κάθε τιμή M, και έτσι σε κάθε μεταγλώττιση καταγράφεται ποιά κυκλώματα χρησιμοποιήθηκαν (και με ποιες τιμές M) και γεννάται η αντίστοιχη περιγραφή τους. Για τα υπόλοιπα σύνθετα κυκλώματα (dff, sync και funcsync) των οποίων η περιγραφή είναι ως γνωστόν σταθερή, εισάγεται στο τέλος του προγράμματος μια εντολή `include ώστε να περιληφθεί ένα εξωτερικό αρχείο Verilog στο οποίο είναι γραμμένες αυτές οι περιγραφές. Το αρχείο αυτό περιέχει τον κώδικα των modules dff, sync και funcsync που δόθηκε στην προηγούμενη παράγραφο.

Μερικά συνολικά παραδείγματα εκτέλεσης του μεταγλωττιστή αναφέρονται στο επόμενο κεφάλαιο.

6

Παραδείγματα εκτέλεσης

ΔΙΟΓΕΝΟΥΣ. Ἐρωτηθεὶς ἂν ἔχῃ ὑπηρέτρια ἢ ὑπηρέτη, ἀπάντησε «Ὄχι». «Καὶ ποιός θὰ σε θάψῃ ἅμα πεθάνῃς;», τοῦ εἶπαν. «Ἐκεῖνος ποὺ θὰ χρειάζεται τὸ σπίτι», ἀπάντησε.

Στο κεφάλαιο αυτό παρουσιάζονται μερικά παραδείγματα εκτέλεσης του μεταγλωττιστή. Η υλοποίηση του μεταγλωττιστή, με την οποία μεταγλωττίστηκαν τα παραδείγματα αυτού του κεφαλαίου, διατίθεται από τον γράφοντα κατ' αίτηση στην ηλεκτρονική διεύθυνση **asimenos@ieee.org**.

Υπολογισμός αριθμών Fibonacci.

Το πρώτο παράδειγμα που θα εξεταστεί είναι ένα πρόγραμμα υπολογισμού αριθμών της ακολουθίας Fibonacci. Η ακολουθία αυτή μπορεί να οριστεί μαθηματικά με την παρακάτω αναδρομική σχέση:

```
fib(0) = 1
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2), n>1
```

Βέβαια, είναι πάρα πολύ απλό να υπολογιστεί ένας αριθμός Fibonacci χωρίς τη χρήση αναδρομής: αρκεί να διατηρούνται κάθε φορά οι δύο προηγούμενες τιμές και να υπολογίζεται επαναληπτικά η επόμενη. Το επόμενο πρόγραμμα υπολογίζει έναν αριθμό της ακολουθίας Fibonacci, κάνοντας χρήση της δυνατότητας παράλληλης εκτέλεσης εντολών.

```
unsigned int *num, result;

void fib(unsigned int N, unsigned int *f)
{
  unsigned int fold = 0;

  *f = 1;

  while (N) par{
    *f += fold;
    fold = *f;
    N--;
  };
}

void main()
{
  fib(*num, &result);
}
```

Όπως φαίνεται, απαιτείται ακριβώς ένας κύκλος ρολογιού για κάθε επανάληψη του βρόχου while της συνάρτησης fib, επειδή οι περιεχόμενες εντολές εκτελούνται παράλληλα. Επίσης, στο βρόχο αυτό χρησιμοποιούνται δύο συνολικά μεταβλητές (f και fold), και όχι τρεις, αφού ο χρονισμός της ανάθεσης επιτρέπει την ταυτόχρονη εκχώρηση τιμών, χωρίς να απαιτείται επιπλέον ενδιάμεση μεταβλητή. Για τη διασύνδεση με το εξωτερικό περιβάλλον έχουν δηλωθεί δύο καθολικές μεταβλητές, για το όρισμα (num) και το αποτέλεσμα (result) αντίστοιχα.

Το αποτέλεσμα της μεταγλώττισης είναι το ακόλουθο κύκλωμα:

```
module myc_program(clk, reset, func_fib, func_main, din_num, dout_num,
                   wout_num, val_result);
input clk, reset, func_fib, func_main;
input [31:0] din_num;
output [31:0] dout_num;
output wout_num;
output [31:0] val_result;

/* Wire Declaration */
wire [31:0] W_1;
wire W_2;
wire [31:0] W_3;
wire [31:0] W_4;
wire W_5;
wire W_6;
wire W_7;
wire W_8;
wire [31:0] W_9;
wire [31:0] W_10;
wire W_11;
wire [31:0] W_12;
wire W_13;
wire W_14;
wire W_15;
wire W_16;
wire [31:0] W_17;
wire [31:0] W_18;
wire W_19;
wire [31:0] W_20;
wire W_21;
wire W_22;
wire W_23;
wire W_24;
wire W_25;
wire W_26;
wire [31:0] W_27;
wire W_28;
wire W_29;
wire [31:0] W_30;
wire W_31;
wire W_32;
wire W_33;
wire W_34;
wire W_35;
wire W_36;


/* Global variables */
// unsigned bit<32> *num
assign W_3 = din_num;
assign dout_num = W_1;
assign wout_num = W_2;
assign W_1 = 32'b0; /* Nothing to select from! */
assign W_2 = 0; /* Never write back */
// unsigned bit<32> result
regselector1 #(32) regselector1_1 (W_4, clk, reset, 32'b0, W_10, W_35);
assign val_result = W_4;


/* Functions */

/* ------------------------------------------- */
// void fib(unsigned bit<32> N, unsigned bit<32> *f)
/* ------------------------------------------- */

// Sticky Updater:
stickyupdate1 stickyupdate1_2 (clk, reset, W_14, W_34);
```

```
/* Function arguments */

// unsigned bit<32> N
regselector2 #(32) regselector2_3 (W_9, clk, reset, 32'b0, W_3, W_14, W_30,
                                   W_24);

// unsigned bit<32> *f
// Argument Call ByReference: DataOut = W_10, WriteOut = W_11, DataIn = W_12
selector2 #(32) selector2_4 (W_10, W_27, W_24, W_20, W_6);
or(W_11, W_24, W_6);
assign W_12 = W_4;

/* Function local variables */
// unsigned bit<32> fold = 32'b0
regselector2 #(32) regselector2_5 (W_17, clk, reset, 32'b0, W_12, W_24,
                                   W_18, W_5);

/* Entry states */
or(W_5, func_fib, W_14);
dff dff_6 (W_6, clk, reset, W_5);

/* Objects for this function: */
assign W_18 = 'b00000000000000000000000000000000;
assign W_19 = 'b1;
assign W_20 = {31'b0, W_19};
dff dff_7 (W_21, clk, reset, W_6);
assign W_22 = |W_9;
and(W_24, W_22, W_23);
not(W_25, W_22);
assign W_27 = W_12 + W_17;
dff dff_8 (W_28, clk, reset, W_24);
dff dff_9 (W_29, clk, reset, W_24);
assign W_30 = W_9 - 32'b1;
dff dff_10 (W_31, clk, reset, W_24);
sync #(3) sync_11 (W_32, clk, reset, {W_31, W_29, W_28});
or(W_23, W_32, W_21);
and(W_26, W_25, W_23);
dff dff_12 (W_33, clk, reset, W_26);

/* Exit states */
assign W_7 = W_33;
assign W_8 = W_7; /* No DFF required */


/* ------------------------------------------- */
// void main()
/* ------------------------------------------- */

/* Entry states */
assign W_13 = func_main;
assign W_14 = W_13; /* No DFF required */

/* Objects for this function: */
and(W_35, W_11, W_34);
funcsync funcsync_13 (W_36, clk, reset, W_14, W_8);

/* Exit states */
assign W_15 = W_36;
assign W_16 = W_15; /* No DFF required */

endmodule

/* Custom module generation */

module selector2(wout, wr1, st1, wr2, st2);
  parameter N = 8;
  output [N-1:0] wout;
  input [N-1:0] wr1, wr2;
```

```
  input st1, st2;
  wire ss1 = st1;
  wire ss2 = st2 & ~(st1);

  assign wout = (wr1 & {N{ss1}}) | (wr2 & {N{ss2}});
endmodule

module regselector1(wout, clk, reset, iv, wr1, st1);
  parameter N = 8;
  output [N-1:0] wout;
  input clk, reset;
  input [N-1:0] iv, wr1;
  input st1;
  wire wen = ss1;
  wire [N-1:0] val = wr1;
  reg [N-1:0] q;

  assign wout = q;
  always @(posedge clk)
    q <= reset ? iv : (wen ? val : q);
endmodule

module regselector2(wout, clk, reset, iv, wr1, st1, wr2, st2);
  parameter N = 8;
  output [N-1:0] wout;
  input clk, reset;
  input [N-1:0] iv, wr1, wr2;
  input st1, st2;
  wire ss1 = st1;
  wire ss2 = st2 & ~(st1);
  wire wen = st1 | st2;
  wire [N-1:0] val = (wr1 & {N{ss1}}) | (wr2 & {N{ss2}});
  reg [N-1:0] q;

  assign wout = q;
  always @(posedge clk)
    q <= reset ? iv : (wen ? val : q);
endmodule

module stickyupdate1(clk, reset, si1, so1);
  input clk, reset, si1;
  output so1;
  reg [0:0] q;
  wire ss1 = si1;
  wire wen = si1;
  assign so1 = wen ? ss1 : q[0];
  always @(posedge clk)
    q <= reset ? 8'b0 : (wen ? {ss1} : q);
endmodule

`include "verilib.v"
```

Το αρχείο αυτό ακολουθεί τη δομή που περιγράφηκε στο προηγούμενο κεφάλαιο σχετικά με την υλοποίηση. Το συμπεριλαμβανόμενο αρχείο ‘verilib.v’ περιέχει τις υλοποιήσεις των modules ‘dff’, ‘sync’ και ‘funcsync’.

Ο επεξεργαστής OISC.

Ο επεξεργαστής OISC (One Instruction Set Computer), του οποίου το όνομα αποτελεί μάλλον μια παρωδία έναντι στους επεξεργαστές RISC (Reduced Instruction Set Computer), είναι ένας επεξεργαστής ο οποίος υποστηρίζει μία και μόνο εντολή. Η εντολή αυτή (που στη βιβλιογραφία απαντάται με διάφορα ονόματα, και σε κάποιες

παραλλαγές) λαμβάνει τρία ορίσματα, έστω A, B και C. Και τα τρία αυτά ορίσματα αντιπροσωπεύουν διευθύνσεις μνήμης. Η λειτουργία της εντολής είναι η εξής:

- Αρχικά, από το περιεχόμενο της μνήμης στη διεύθυνση A αφαιρείται το περιεχόμενο της μνήμης στη διεύθυνση B.
- Αν το αποτέλεσμα είναι αρνητικό, ο έλεγχος μεταφέρεται στην εντολή της διεύθυνσης C, διαφορετικά στην επόμενη εντολή.

Σε ψευδοκώδικα, η λειτουργία της εντολής είναι η εξής:

```
Memory(A) := Memory(A) - Memory(B);
if (Memory(A) < 0) then JUMP C;
```

Ας υποτεθεί ότι τα δεδομένα χρησιμοποιούν παράσταση συμπληρώματος ως προς 2. Ο επεξεργαστής δε διαθέτει καταχωρητές, παρά δρα αποκλειστικά στα δεδομένα της μνήμης. Η μνήμη δεδομένων είναι η ίδια με τη μνήμη εντολών, ενώ η κωδικοποίηση των εντολών στη μνήμη γίνεται απλά, γράφοντας τα ορίσματα σε τρεις διαδοχικές θέσεις μνήμης. Έτσι, στην πράξη οποιαδήποτε θέση μνήμης μπορεί να θεωρηθεί ως η θέση έναρξης μιας εντολής, ενώ μια οποιαδήποτε συλλογή δεδομένων αποτελεί ταυτόχρονα και εκτελέσιμο πρόγραμμα!

Αποδεικνύεται ότι αυτή η μία και μοναδική εντολή του OISC είναι αρκετή για να περιγράψει ακόμα και τις πιο σύνθετες λειτουργίες που πραγματοποιούν οι μεγαλύτεροι επεξεργαστές RISC. Πράγματι, κάθε τέτοια λειτουργία μπορεί να μοντελοποιηθεί από μια ακολουθία εντολών του OISC, όσο ακατόρθωτο και αν αυτό φαίνεται εκ πρώτης όψεως!

Μια υλοποίηση του επεξεργαστή OISC δίδεται στον παρακάτω κώδικα:

```
int addr;
int *mem;

void oisc()
{
  int a, datum, c;
  int pc;

  for(;;)
  {
    par { a=*mem; addr=*mem; pc=addr+1; }
    par { datum=*mem; addr=pc; }
    par { addr=*mem; pc++; }
    par { datum-=*mem; addr=pc; }
    par { c=*mem; addr=a; }
    par
    {
      *mem=datum;
      if(datum<0)
        addr=c;
      else
        addr=pc+1;
    }
  }
}
```

Η υλοποίηση αυτή χρησιμοποιεί τη διασύνδεση με μνήμη που περιγράφηκε στο προηγούμενο κεφάλαιο. Ο κώδικας λειτουργεί επ' άπειρον σε ένα επαναληπτικό βρόχο, ο οποίος εκτελεί την επόμενη κατά σειρά εντολή του OISC. Η μεταβλητή datum λαμβάνει αρχικά το δεδομένο της μνήμης που βρίσκεται στη διεύθυνση A, από το οποίο ύστερα αφαιρείται το περιεχόμενο της μνήμης που βρίσκεται στη διεύθυνση B. Τελικά διαβάζεται και το δεδομένο C, και η νέα διεύθυνση ανάγνωσης τίθεται κατάλληλα είτε στη διεύθυνση C είτε στην επόμενη εντολή (με τη βοήθεια της μεταβλητής pc που αναπαριστά τον Program Counter). Η παράλληλη εκτέλεση εντολών επιτρέπει την πλήρη απασχόληση της μνήμης σε κάθε κύκλο ρολογιού, ώστε αφού διαβαστεί ένα δεδομένο να προετοιμαστεί ταυτόχρονα η διεύθυνση του επόμενου.

Ο κώδικας Verilog που παράγεται από τη μεταγλώττιση είναι ο εξής:

```
module myc_program(clk, reset, func_oisc, val_addr, din_mem, dout_mem,
                   wout_mem);
input clk, reset, func_oisc;
output [31:0] val_addr;
input [31:0] din_mem;
output [31:0] dout_mem;
output wout_mem;

/* Wire Declaration */
wire [31:0] W_1;
wire [31:0] W_2;
wire W_3;
wire [31:0] W_4;
wire W_5;
wire W_6;
wire W_7;
wire W_8;
wire [31:0] W_9;
wire [31:0] W_10;
wire [31:0] W_11;
wire [31:0] W_12;
wire W_13;
wire W_14;
wire W_15;
wire W_16;
wire [31:0] W_17;
wire [31:0] W_18;
wire W_19;
wire W_20;
wire W_21;
wire W_22;
wire W_23;
wire W_24;
wire [31:0] W_25;
wire W_26;
wire W_27;
wire [31:0] W_28;
wire W_29;
wire W_30;
wire W_31;
wire W_32;
wire W_33;
wire W_34;
wire W_35;
wire W_36;
wire [31:0] W_37;
wire W_38;
wire W_39;
wire W_40;
wire W_41;
```

```
wire W_42;
wire W_43;
wire [31:0] W_44;
wire [31:0] W_45;
wire W_46;
wire W_47;
wire W_48;
wire W_49;


/* Global variables */
// signed bit<32> addr
regselector7 #(32) regselector7_1 (W_1, clk, reset, 32'b0, W_45, W_41, W_11,
                                   W_39, W_9, W_31, W_12, W_27, W_4, W_23,
                                   W_12, W_20, W_4, W_13);
assign val_addr = W_1;
// signed bit<32> *mem
assign W_4 = din_mem;
assign dout_mem = W_2;
assign wout_mem = W_3;
assign W_2 = W_10;
assign W_3 = W_34;


/* Functions */

/* ----------------------------------------- */
// void oisc()
/* ----------------------------------------- */

/* Function local variables */
// signed bit<32> pc
regselector2 #(32) regselector2_2 (W_12, clk, reset, 32'b0, W_25, W_23, W_18,
                                   W_13);
// signed bit<32> c
regselector1 #(32) regselector1_3 (W_11, clk, reset, 32'b0, W_4, W_31);
// signed bit<32> datum
regselector2 #(32) regselector2_4 (W_10, clk, reset, 32'b0, W_28, W_27, W_4,
                                   W_20);
// signed bit<32> a
regselector1 #(32) regselector1_5 (W_9, clk, reset, 32'b0, W_4, W_13);

/* Entry states */
assign W_5 = func_oisc;
assign W_6 = W_5; /* No DFF required */

/* Objects for this function: */
dff dff_6 (W_14, clk, reset, W_13);
dff dff_7 (W_15, clk, reset, W_13);
assign W_16 = 'b1;
assign W_17 = {31'b0, W_16};
assign W_18 = W_1 + W_17;
dff dff_8 (W_19, clk, reset, W_13);
sync #(3) sync_9 (W_20, clk, reset, {W_19, W_15, W_14});
dff dff_10 (W_21, clk, reset, W_20);
dff dff_11 (W_22, clk, reset, W_20);
sync #(2) sync_12 (W_23, clk, reset, {W_22, W_21});
dff dff_13 (W_24, clk, reset, W_23);
assign W_25 = W_12 + 32'b1;
dff dff_14 (W_26, clk, reset, W_23);
sync #(2) sync_15 (W_27, clk, reset, {W_26, W_24});
assign W_28 = W_10 - W_4;
dff dff_16 (W_29, clk, reset, W_27);
dff dff_17 (W_30, clk, reset, W_27);
sync #(2) sync_18 (W_31, clk, reset, {W_30, W_29});
dff dff_19 (W_32, clk, reset, W_31);
dff dff_20 (W_33, clk, reset, W_31);
sync #(2) sync_21 (W_34, clk, reset, {W_33, W_32});
```

```
dff dff_22 (W_35, clk, reset, W_34);
assign W_36 = 'b0;
assign W_37 = {31'b0, W_36};
assign W_38 = (W_10 < W_37);
and(W_39, W_38, W_34);
not(W_40, W_38);
and(W_41, W_40, W_34);
dff dff_23 (W_42, clk, reset, W_39);
assign W_43 = 'b1;
assign W_44 = {31'b0, W_43};
assign W_45 = W_12 + W_44;
dff dff_24 (W_46, clk, reset, W_41);
or(W_47, W_46, W_42);
sync #(2) sync_25 (W_48, clk, reset, {W_47, W_35});
or(W_13, W_6, W_48);
assign W_49 = 1'b0;

/* Exit states */
assign W_7 = W_49;
assign W_8 = W_7; /* No DFF required */

endmodule

/* Custom module generation */

module regselector1(wout, clk, reset, iv, wr1, st1);
  parameter N = 8;
  output [N-1:0] wout;
  input clk, reset;
  input [N-1:0] iv, wr1;
  input st1;
  wire wen = ss1;
  wire [N-1:0] val = wr1;
  reg [N-1:0] q;

  assign wout = q;
  always @(posedge clk)
    q <= reset ? iv : (wen ? val : q);
endmodule

module regselector2(wout, clk, reset, iv, wr1, st1, wr2, st2);
  parameter N = 8;
  output [N-1:0] wout;
  input clk, reset;
  input [N-1:0] iv, wr1, wr2;
  input st1, st2;
  wire ss1 = st1;
  wire ss2 = st2 & ~(st1);
  wire wen = st1 | st2;
  wire [N-1:0] val = (wr1 & {N{ss1}}) | (wr2 & {N{ss2}});
  reg [N-1:0] q;

  assign wout = q;
  always @(posedge clk)
    q <= reset ? iv : (wen ? val : q);
endmodule

module regselector7(wout, clk, reset, iv, wr1, st1, wr2, st2, wr3, st3, wr4,
                    st4, wr5, st5, wr6, st6, wr7, st7);
  parameter N = 8;
  output [N-1:0] wout;
  input clk, reset;
  input [N-1:0] iv, wr1, wr2, wr3, wr4, wr5, wr6, wr7;
  input st1, st2, st3, st4, st5, st6, st7;
  wire ss1 = st1;
  wire ss2 = st2 & ~(st1);
  wire ss3 = st3 & ~(st2 | st1);
  wire ss4 = st4 & ~(st3 | st2 | st1);
```

```
  wire ss5 = st5 & ~(st4 | st3 | st2 | st1);
  wire ss6 = st6 & ~(st5 | st4 | st3 | st2 | st1);
  wire ss7 = st7 & ~(st6 | st5 | st4 | st3 | st2 | st1);
  wire wen = st1 | st2 | st3 | st4 | st5 | st6 | st7;
  wire [N-1:0] val = (wr1 & {N{ss1}}) | (wr2 & {N{ss2}}) | (wr3 & {N{ss3}}) |
                     (wr4 & {N{ss4}}) | (wr5 & {N{ss5}}) | (wr6 & {N{ss6}}) |
                     (wr7 & {N{ss7}});
  reg [N-1:0] q;

  assign wout = q;
  always @(posedge clk)
    q <= reset ? iv : (wen ? val : q);
endmodule

`include "verilib.v"
```