ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
Τμήμα Ηλεκτρονικής & Μηχανικών Υπολογιστών
Εργαστήριο Μικροεπεξεργαστών & Υλικού
Πολυτεχνειούπολη Χανίων Τ.Κ. 73100 Κρήτη
http://www.mhl.tuc.gr/
Τηλ. : 0821-37262 Fax : 0821-37202

ΕΙΣΑΓΩΓΗ ΣΤΗ VHDL

Διονύσης Ευσταθίου

Διδάσκων: Καθ. Απόστολος Δόλλας

Εργαστήριο Μικροεπεξεργαστών & Υλικού
Διευθυντής Εργαστηρίου: Καθ. Απόστολος Δόλλας
Προετοιμασία: Ευριπίδης Σωτηριάδης, Μάρκος Κιμιωνής.
ΕΔΤΠ : Μάρκος Κιμιωνής

Χανιά 2001

ΠΕΡΙΕΧΟΜΕΝΑ

1.Εισαγωγή 3
Διαφορές VHDL με /C++ 3
2.Βασικές αρχές της VHDL..4
2.1 Entity Declaration 5
2.2 Architecture Body 5
2.2.1 Structural Style of Modeling (Δομικός τρόπος σχεδίασης)...........................6
2.2.2 Behavioral Style of Modeling (Συμπεριφερικός τρόπος σχεδίασης).............7
2.2.3 Dataflow Style of Modeling (Σχεδίαση Διαγράμματος ροής).......................9
2.2.4 Mixed Style of Modeling(Συνδυασμός των παραπάνω).............................10
2.3 Οργάνωση της σχεδίασης (Design Organization).. 12
2.3.1 Βιβλιοθήκες...12
2.3.2 Packages (Πακέτα) 12
2.3.3 Πρόσβαση σε βιβλιοθήκες και πακέτα...13
3.Βασικά χαρακτηριστικά της γλώσσας VHDL...13
3.1 Συνδυαστική Λογική Σχεδίαση (Combinational Logic Design) 14
3.1.2 Logical operators (Λογικές πράξεις) 14
3.1.3 Relational operators (Σχεσιακές πράξεις)...15
3.1.4 Multiplexer 4-to-1 (Πολυπλέκτης 4-σε-1)...16
3.1.4.1 Λειτουργία του πολυπλέκτη 4-σε-1 16
3.1.4.2 Σχεδίαση του πολυπλέκτη 4-σε-1 σε VHDL 17

Επιλογή βιβλιοθηκών...17
Ορισμός του entity 17
Architecture body..17

Dataflow design Style (Σχεδίαση με διάγραμμα ροής) 17
Behavioral design Style(Συμπεριφερικός τρόπος σχεδίασης).........................18
Structural design Style (Δομικός τρόπος σχεδίασης) 20
3.1.4.3 Συμπεράσματα από την σχεδίαση του MUX 4 to 1 σε VHDL.................22
3.2 Ακολουθιακή Λογική Σχεδίαση (Sequential Logic Design) 22

3.2.1 Προτερήματα του structural τρόπου σχεδίασης....22
3.2.2 4-bit Shift Register (4-bit Καταχωρητής Ολίσθησης) 22
3.2.2.1 Λειτουργία του 4-bit Shift Register 23
3.2.2.2 Σχεδίαση του 4-bit Shift Register 24

Επιλογή βιβλιοθηκών 24
Ορισμός του entity 24
Architecture body....25

Behavioral design Style(Συμπεριφερικός τρόπος σχεδίασης) 25
Structural design Style (Δομικός τρόπος σχεδίασης) 26
4.Βιβλιογραφία 27

1. ΕΙΣΑΓΩΓΗ

Η VHDL είναι μια γλώσσα περιγραφής υλικού για την ανάπτυξη ψηφιακών ηλεκτρονικών συστημάτων. Ως λέξη αποτελεί συντόμευση των λέξεων: VHSIC Hardware Description Language. Τα δε αρχικά VHSIC είναι με τη σειρά τους συντόμευση για Very High-Speed Integrated Circuit (Ολοκληρωμένα Κυκλώματα Υψηλής Ταχύτητας).
Η VHDL ως γλώσσα προγραμματισμού μπορεί να χρησιμοποιηθεί για την περιγραφή της συμπεριφοράς , της δομής αλλά και της εφαρμογής ψηφιακών συστημάτων. Με βάση αυτά τα χαρακτηριστικά η VHDL χαρακτηρίζεται σαν ένα εργαλείο ECAD (Electronic Computer Aided Design).
Γενικά, σήμερα, η χρήση εργαλείων CAD έχει επεκταθεί καθώς η τεράστια ανάπτυξη της τεχνολογίας ημιαγωγών στην κατασκευή ολοκληρωμένων κυκλωμάτων έχει μετατοπίσει το κέντρο βάρους των μηχανικών από την λεπτομερειακή υλοποίηση κυκλωμάτων στην διαχείριση της αυξανόμενης πολυπλοκότητας. Πιο συγκεκριμένα τη σημερινή εποχή ο μηχανικός-σχεδιαστής, περιορίζεται περισσότερο από την δυνατότητά του να αντεπεξέλθει την πολυπλοκότητα της σχεδίασης του παρά από την ικανότητα της τεχνολογίας να την υποστηρίξει. Αυτό το χάσμα έρχεται να γεφυρώσει η VHDL επιτρέποντας μια υψηλού επιπέδου περιγραφή (Abstract) της σχεδίασης και κατόπιν με την χρήση εργαλείων σύνθεσης (Logic Synthesis Tools) την αυτόματη αποτύπωση αυτής της σχεδίασης σε ολοκληρωμένη μορφή η οποία να είναι εντός των προδιαγραφών που θέτει ο μηχανικός.

Διαφορές VHDL με C/C++

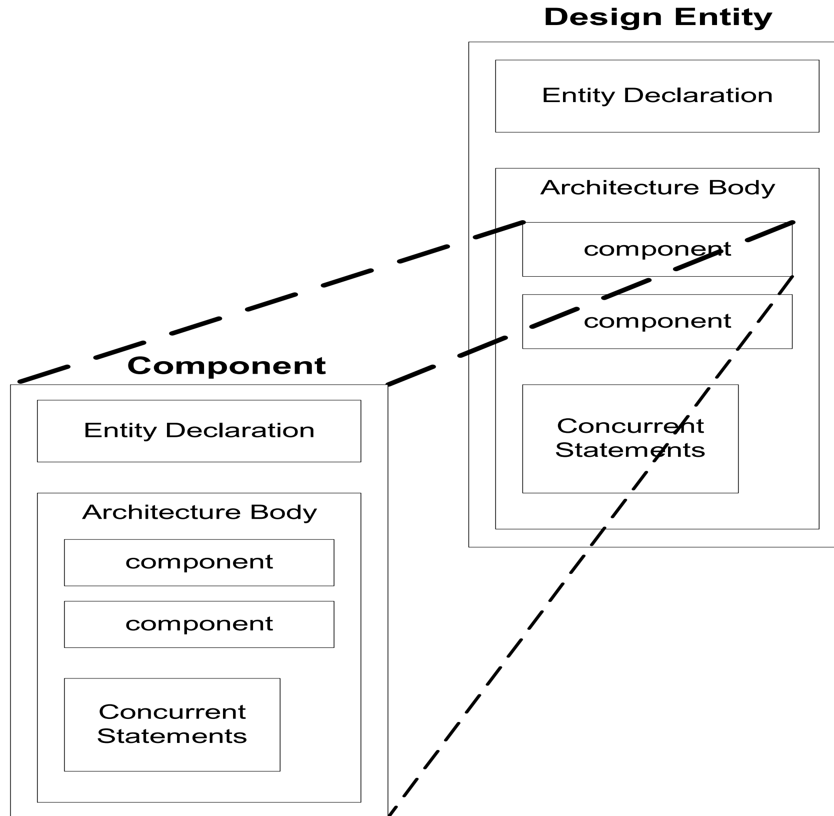
Η VHDL είναι όπως αναφέραμε μια γλώσσα περιγραφής υλικού (HDL). Σε σχέση με γνωστές υψηλού επιπέδου γλώσσες προγραμματισμού (C/C++) υπάρχουν μερικές βασικές διαφορές :

- Στη VHDL δεν υπάρχει κάποια συγκεκριμένη ροή προγράμματος όπως στην C. Κάθε έκφραση στην VHDL εκτελείται ‘‘παράλληλα’’ έκτος αν δηλωθεί το αντίθετο. Βέβαια, υπάρχουν constructs στην VHDL για κατ’ εξαίρεση σειριακή ροή αλλά και αυτό από μόνο του δηλώνει ότι η VHDL είναι μια ‘‘παράλληλη’’ γλώσσα, ότι δηλαδή αυτό γίνεται κατ’ εξαίρεση.
- Η VHDL επιτρέπει την χρήση χρονικών καθυστερήσεων. Μπορούμε δηλ. να πούμε ότι μια δήλωση εκτελείται μετά την πάροδο ενός χρονικού ορίου. Ναι μεν σε μερικές γλώσσες υποστηρίζεται η χρήση χρονικών καθυστερήσεων αλλά αυτό γίνεται μέσω διαφόρων πρόσθετων βιβλιοθηκών και δεν αποτελούν αναπόσπαστο κομμάτι της γλώσσας.

Όμως η VHDL παρουσιάζει και ομοιότητες με μερικές άλλες γλώσσες (π.χ. ADA, C++) κύρια στον τομέα του αντικειμενοστραφή (object-oriented) προγραμματισμού.

2. ΒΑΣΙΚΕΣ ΑΡΧΕΣ ΤΗΣ VHDL

Η VHDL ως γλώσσα περιγραφής υλικού μπορεί να χρησιμοποιηθεί για την περιγραφή ενός ψηφιακού κυκλώματος όπως προαναφέραμε. Αυτό το κύκλωμα μπορεί να είναι από μία απλή λογική πύλη έως ένα ολοκληρωμένο ψηφιακό σύστημα. Στη VHDL το ψηφιακό κύκλωμα που σχεδιάζεται, αναφέρεται ως **entity** (οντότητα). Όμως ένα entity X όταν εμπεριέχεται μέσα σ’ ένα entity Y τότε αυτό ονομάζεται **component** (στοιχείο).

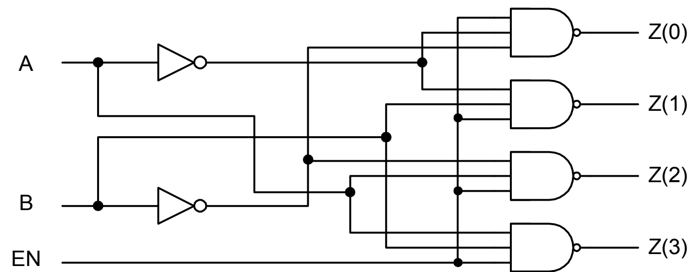


Για την περιγραφή ενός entity, η VHDL προσφέρει πέντε διαφορετικά constructs τα οποία και ονομάζονται μονάδες σχεδίασης (design units):

- Entity declaration
- Architecture body
- Configuration declaration
- Packages

2.1 **Entity Declaration**

Είναι το κομμάτι του κώδικα το οποίο δηλώνει το όνομα του συγκεκριμένου κυκλώματος και επίσης παραθέτει και μια λίστα με τα **Ports** του κυκλώματος. Με τον όρο ports, εννοούμε τα σήματα μέσω των οποίων το συγκεκριμένο entity επικοινωνεί με το εξωτερικό περιβάλλον (είσοδοι, έξοδοι, κ.ά.)



Σχ. 2.1 Κύκλωμα 2 σε 4 decoder

Η δήλωση για το κύκλωμα του σχήματος 2.1 γίνεται ως εξής:

```
entity DECODER2x4 is
    port (A, B, ENABLE: in bit; Z :out BIT_VECTOR(0 to 3));
end DECODER2x4;
```

Το παραπάνω entity έχει τρία ports εισόδου (A, B, ENABLE) και τέσσερα ports εξόδου (Z(0), Z(1), Z(2), Z(3)). Το port Z είναι τύπου bit_vector, δηλαδή ένας μονοδιάστατος πίνακας από bits (bus) του οποίου το μέγεθος δηλώνεται από τον χρήστη. Στην προκειμένη περίπτωση το μέγεθός του είναι 4 (0 **to** 3) ή αλλιώς (3 **downto** 0).

2.2 **Architecture Body**

Από το παραπάνω παράδειγμα είναι φανερό ότι το κομμάτι entity declaration δεν αναφέρει τίποτα για το εσωτερικό του entity. Απλά δηλώνει το όνομα αλλά και τα interface ports. Οι εσωτερικές λεπτομέρειες του entity (δηλ. του κυκλώματος) δηλώνονται στον κύριο κορμό της σχεδίασης που είναι το Architecture body με έναν από τους κάτωθι τρόπους σχεδίασης:

- Ως ένας συνδυασμός αλληλοσυνδεόμενων στοιχείων (components). Structural Style of Modeling (Δομικός τρόπος σχεδίασης).
- Ως ένας συνδυασμός «παράλληλων» δηλώσεων και αναθέσεων. Dataflow Style of Modeling (Σχεδίαση Διαγράμματος ροής).
- Ως ένας συνδυασμός «σειριακών» δηλώσεων και αναθέσεων. Behavioral Style of Modeling (Συμπεριφερικός τρόπος σχεδίασης).
- Ως ένας συνδυασμός των παραπάνω.

2.2.1 **Structural Style of Modeling (Δομικός τρόπος σχεδίασης).**

Ο δομικός τρόπος σχεδίασης περιγράφει το κύκλωμα κυρίως με βάση το στοιχεία (components) που το απαρτίζουν. Αυτά μπορεί να είναι στοιχεία από διάφορες βιβλιοθήκες ή και από τον ίδιο το χρήστη από μία άλλη σχεδίασή του. Το αν ο σχεδιαστής θα χρησιμοποιήσει στοιχεία από «έτοιμες» βιβλιοθήκες ή θα προτιμήσει δικής του κατασκευής στοιχεία (generic/user-defined components) είναι ένα σημαντικό θέμα το οποίο επιλύεται με βάση την εμπειρία του αλλά και με βάση τις απαιτήσεις της σχεδίασής του. Το Architecture body, στο δομικό τρόπο σχεδίασης, ορίζει ποια components εμπεριέχονται στη συγκεκριμένη σχεδίαση καθώς και το πώς ενώνονται μεταξύ τους. Τα κύρια στοιχεία της VHDL που χρησιμοποιούνται σε αυτό τον τρόπο σχεδίασης είναι :

- Component declaration and instantiation (Δήλωση στοιχείων)
- Port mapping and signal interface lists (Αναφορά στον τρόπο σύνδεσης του κάθε στοιχείου της σχεδίασης με τα υπόλοιπα. Δηλαδή η έξοδος του στοιχείου σε ποια είσοδο άλλου στοιχείου πηγαίνει και με τη χρήση ποιου σήματος)
- Libraries and packages (βιβλιοθήκες και πακέτα)
- Signals (for interconnections)

Λεπτομερής ανάλυση των όσων προαναφέρονται θα ακολουθήσει αργότερα.
Με βάση το παράδειγμα του 2-σε-4 decoder του σχήματος 2.1 ας δούμε μία εφαρμογή του architecture body (structural) του entity **DECODER2x4**.

```
architecture DEC_STRUCT of DECODER2x4 is
    component INV
            port(PIN: in BIT; POUT: out BIT);
    end component;
    component NAND3
            port(D0, D1, D2: in BIT; DZ: out BIT);
    end component;
    signal ABAR,BBAR: BIT;
begin
    V0: INV port map (A, ABAR);
    V1: INV port map (A, BBAR);
    N0: NAND3 port map (ENABLE, ABAR, BBAR, Z(0));
    N1: NAND3 port map (ABAR, B, ENABLE, Z(1));
    N2: NAND3 port map (A, BBAR, ENABLE, Z(2));
    N3: NAND3 port map (A, B, ENABLE, Z(3));
end DEC_STRUCT;
```

Σε αυτό το παράδειγμα το όνομα του architecture body είναι DEC_STRUCT. Το entity DECODER2x4 (που περιγράψαμε στην ενότητα 2.1) ορίζει τις εισόδους και τις εξόδους γι' αυτό το architecture body. Το architecture body χωρίζεται σε δύο μέρη: το κομμάτι των ορισμών (**declarative** part) πριν τη λέξη-κλειδί **begin** και το κομμάτι των δηλώσεων (**statement** part) μετά τη λέξη-κλειδί **begin.** Στο κομμάτι των

ορισμών υπάρχει ο ορισμός δύο στοιχείων (components), της πύλης not (INV) και της πύλης nand 3-εισόδων (NAND3). Αυτοί οι ορισμοί δείχνουν ουσιαστικά τις εισόδους και τις εξόδους κάθε στοιχείου. Τα στοιχεία αυτά μπορεί να είναι στοιχεία βιβλιοθηκών ή στοιχεία δημιουργημένα από τον σχεδιαστή. Επίσης στο κομμάτι των ορισμών υπάρχει και ο ορισμός δύο σημάτων (signals) του ABAR και του BBAR. Αυτά τα δύο σήματα είναι κατ' ουσία δύο σύρματα τα οποία χρησιμοποιούνται για την ένωση των στοιχείων για τη δημιουργία του 2-σε-4 decoder. Η εμβέλεια αυτών των σημάτων βρίσκεται μόνο μέσα στο architecture body και κατά συνέπεια δεν είναι ορατά έξω απ' αυτό. Σε αντίθεση με τα ports του entity τα οποία είναι ορατά σε όλα τα μέρη του κώδικα.

Τα ορισμένα στοιχεία κατόπιν «αποτυπώνονται» στο κομμάτι των δηλώσεων (instantiated). Οι ταυτότητες (labels) αυτών των αποτυπώσεων στην συγκεκριμένη περίπτωση είναι οι V0, V1, N0, N1, N2, N3. Η ταυτότητα V0 (μιας πύλης NOT) αν κοιτάξει κανείς τον κώδικα, δείχνει ότι δέχεται, το συγκεκριμένο στοιχείο, ως είσοδο μια είσοδο της σχεδίασης (την A) και έχει ως έξοδο το σήμα ABAR. Ένα σύρμα δηλαδή που αργότερα πηγαίνει ως είσοδο σε δύο πύλες NAND3 και ειδικότερα στα στοιχεία N2 και N3. Το ίδιο συμβαίνει και με τα υπόλοιπα στοιχεία. Η αποτύπωση (instantiation)γενικά είναι μια παράλληλη δήλωση. Γι' αυτό το λόγο δεν παίζει ρόλο η σειρά των δηλώσεων. Ο δομικός τρόπος σχεδίασης γενικά δείχνει απλά το πώς συνδέονται το διάφορα στοιχεία μεταξύ τους. Τα στοιχεία με αυτόν τον τρόπο σχεδίασης αποτελούν «μαύρα κουτιά», δηλ. δεν γίνεται καμία αναφορά για τη συμπεριφορά ή τη λειτουργία τους. Γι' αυτό το λόγο αλλά και για άλλους, ο δομικός τρόπος σχεδίασης χρησιμοποιείται συχνά στο υψηλότερο επίπεδο σχεδίασης έτσι ώστε να γίνεται ένας διαυγής καθορισμός των επιμέρους στοιχείων της τελικής σχεδίασης αλλά και για να έχει η σχεδίαση μια καθαρά ιεραρχική δομή.

2.2.2 **Behavioral Style of Modeling (Συμπεριφερικός τρόπος σχεδίασης).**

Ο συμπεριφερικός τρόπος σχεδίασης εμπεριέχει «παράλληλες» δηλώσεις με κομμάτια σειριακών δηλώσεων οι οποίες περιγράφουν τις εξόδους του κυκλώματος σε συγκεκριμένες χρονικές στιγμές και με συγκεκριμένες εισόδους. Από όλους τους τρόπους σχεδίασης μόνο ο συμπεριφερικός τρόπος χρησιμοποιεί άμεσα τις έννοιες του χρόνου και του ελέγχου. Αυτός ο τρόπος γενικά, περιγράφει τις λειτουργίες του κυκλώματος σε αλγοριθμικό επίπεδο. Τα κύρια στοιχεία της VHDL που χρησιμοποιούνται σε αυτό τον τρόπο σχεδίασης είναι :

- Process statements and sensitivity list
- Sequential statements (σειριακές δηλώσεις)
- Variables (μεταβλητές)

Επανερχόμενοι στο θέμα των δηλώσεων, ο συμπεριφερικός τρόπος σχεδίασης περιγράφει τη συμπεριφορά ενός entity με ένα αριθμό δηλώσεων οι οποίες εκτελούνται σειριακά, δηλ. σύμφωνα με τη σειρά που αυτές έχουν. Αυτό το σετ δηλώσεων οι οποίες βρίσκονται μέσα σε ένα process (διαδικασία) δεν αναφέρονται στη δομή του κυκλώματος-entity αλλά μόνο στη λειτουργικότητά του. Από την άλλη, ένα process εξωτερικά είναι μια παράλληλη δήλωση και εκτελείται παράλληλα (δηλ. άσχετα με τη θέση του σε σχέση με άλλες παράλληλες δηλώσεις ή άλλα process).
Με βάση το παράδειγμα του 2-σε-4 decoder του σχήματος 2.1 ας δούμε μία εφαρμογή του architecture body (behavioral) του entity **DECODER2x4**.

```
architecture DEC_SEQUENTIAL of DECODER2x4 is
begin
    process (A,B, ENABLE)
            variable ABAR, BBAR: BIT;
    begin
            ABAR:= not A;                              -- statement 1
            BBAR:= not B;                              -- statement 2
            if ENABLE = '1' then                       -- statement 3
                    Z(3) <= not (A and B);             -- statement 4
                    Z(0) <= not (ABAR and BBAR);       -- statement 5
                    Z(2) <= not (A and BBAR);  -- statement 6
                    Z(1) <= not (ABAR and B);  -- statement 7
            else
                    Z <= "1111";                       -- statement 8
            end if;
    end process;
end DEC_ SEQUENTIAL;
```

Όπως και στο δομικό τρόπο σχεδίασης, ένα process έχει και αυτό ένα κομμάτι ορισμών (declarative part) και ένα κομμάτι δηλώσεων (statement part) πριν τη λέξη-κλειδί **begin** και μεταξύ των λέξεων-κλειδιά **begin** και **end process**, αντίστοιχα. Οι δηλώσεις που υπάρχουν στο statement part εκτελούνται σειριακά. Δίπλα στη λέξη-κλειδί process και σε παρένθεση υπάρχει μία λίστα σημάτων (στην περίπτωσή μας οι είσοδοι και οι έξοδοι του κυκλώματος). Αυτά αποτελούν την sensitivity list (λίστα ευαισθησίας) του process. Το process ενεργοποιείται όταν υπάρξει αλλαγή κατάστασης σ' ενα από αυτά τα σήματα της λίστας. Τότε και μόνο τότε εκτελείται το process. Αυτή η λίστα είναι προαιρετική.

Οι μεταβλητές του process στο συγκεκριμένο παράδειγμα ορίζονται οι ABAR και BBAR. Οι διαφορές των μεταβλητών με τα σήματα (variables vs signals) είναι ότι οι μεν μεταβλητές αποκτούν αμέσως τιμή ενώ τα σήματα μετά από ένα χρονικό διάστημα (ορισμένο από τον χρήστη ή το delta delay). Άλλη διαφορά είναι ότι ενώ στις μεταβλητές τους γίνεται ανάθεση τιμής μέσω του συμβόλου **:=** στα σήματα αυτή η ανάθεση γίνεται μέσω του συμβόλου **<=**. Οι μεταβλητές που δηλώνονται μέσα σε ένα process έχουν εμβέλεια μόνο το process αυτό (local variables) ενώ οι μεταβλητές που δηλώνονται εξωτερικά (shared variables) έχουν εμβέλεια σε όλα τα process.
Τα σήματα, επίσης, δεν μπορούν να δηλωθούν μέσα σ' ένα process και να λειτουργούν «παράλληλα».

Επανερχόμαστε τώρα στο παραπάνω παράδειγμα. Αν στα σήματα A, B, ENABLE υπάρξει αλλαγή καταστάσεως, τότε αρχικά θα εκτελεστεί η 1η δήλωση (statement 1) ύστερα η δεύτερη. Κατά την εκτέλεση της τρίτης δήλωσης (**if** statement),αν για παράδειγμα η είσοδος ENABLE έχει την τιμή '1' την στιγμή που παρατηρήθηκε η αλλαγή καταστάσεων των σημάτων (δηλ. αρχικά όταν ενεργοποιηθεί το process) τότε θα εκτελεστούν σειριακά οι δηλώσεις 4 έως 7. Αυτό θα συμβεί άσχετα με το αν τα σήματα ή οι μεταβλητές A, B, ABAR, BBAR έχουν, στο ενδιάμεσο αλλάξει τιμές. Όταν αργότερα η ροή του προγράμματος έχει φτάσει

στο τέλος του process, τότε αυτό παραμένει ανενεργό μέχρι να ξαναπαρατηρηθεί αλλαγή κατάστασης σε κάποιο από τα σήματα A, B, ENABLE. Άρα είναι φανερό ότι ένα process, σαν οντότητα είναι μια **παράλληλη** δήλωση καθώς για να εκτελεστεί περιμένει μια αλλαγή κατάστασης στα σήματα με τα οποία συνδέεται.
Τέλος αξίζει να αναφερθεί ότι ο συμπεριφερικός τρόπος σχεδίασης μπορεί να περιγράψει και συνδυαστικά αλλά και ακολουθιακά κυκλώματα. Άρα είναι ιδανικό εργαλείο για την υλοποίηση **μηχανών πεπερασμένων καταστάσεων (FSMs)**.

2.2.3 **Dataflow Style of Modeling (Σχεδίαση Διαγράμματος ροής).**

Με τη χρήση σχεδίασης διαγράμματος ροής για την υλοποίηση ενός κυκλώματος, περιγράφουμε απλά τη «ροή» των δεδομένων μεταξύ στοιχείων συνδυαστικής λογικής, όπως απλές λογικές πύλες, αθροιστές, αποκωδικοποιητές. Με άλλα λόγια περιγράφεται το **RTL** (Register-Transfer-Level) του κυκλώματος. Τα κύρια στοιχεία της VHDL που χρησιμοποιούνται σε αυτό τον τρόπο σχεδίασης είναι :

- Operators – (πράξεις : λογικές, σχεσιακές, αριθμητικές)
- Παράλληλες δηλώσεις

Με αυτόν τον τρόπο σχεδίασης η δομή του κυκλώματος δεν ορίζεται άμεσα, δηλαδή με την «φανερή» χρήση στοιχείων (components) όπως με την structural δομή, αλλά έμμεσα.
Με βάση το κύκλωμα του σχήματος 2.1 (2 to 4 decoder), το architecture body του entity **DECODER2x4** με τη χρήση της dataflow σχεδίασης είναι:

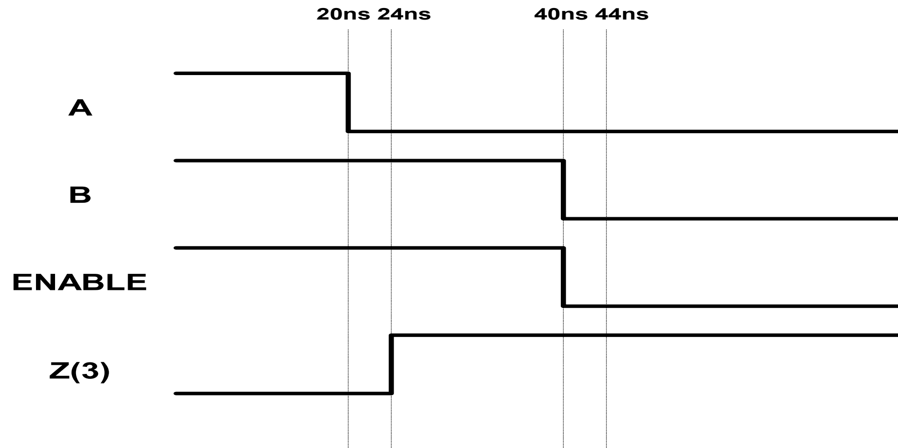
```
architecture DEC_DATAFLOW of DECODER2x4 is
    signal ABAR, BBAR: BIT;
begin
        Z(3)<= not (A and B and ENABLE);            -- statement 1
        Z(0)<= not (ABAR and BBAR and ENABLE);      -- statement 2
        BBAR<=not B;                                -- statement 3
        Z(2) <= not (A and BBAR and ENABLE);  -- statement 4
        ABAR<= not A;                               -- statement 5
        Z(1) <= not (ABAR and B and ENABLE);  -- statement 6
end DEC_DATAFLOW;
```

Το παραπάνω architecture body, αποτελείται από το κομμάτι ορισμών (declarative part) όπου απλά δηλώνονται δύο σήματα (ABAR, BBAR) και από το κομμάτι των δηλώσεων όπου και βρίσκονται 6 δηλώσεις (παράλληλες).
Αξίζει σε αυτό το σημείο να τονίσουμε κάτι που αναφέρθηκε αρχικά. Είχαμε πει ότι στην VHDL μπορούμε να κάνουμε χρήση χρονικών καθυστερήσεων. Έτσι στην προκειμένη περίπτωση θα μπορούσαμε για παράδειγμα να γράψουμε για την 1η δήλωση:

```
Z(3)<= not (A and B and ENABLE) after 4ns;            -- statement 1
```

Σε αυτήν την περίπτωση αν υποθέσουμε ότι ένα από τα τρία σήματα, στο δεξί μέρος της δήλωσης, αλλάξει κατάσταση την χρονική στιγμή **T**, τότε η νέα τιμή του Z(3) θα εμφανιστεί την χρονική τιμή **T+4ns (**βλέπε Σχ.2.2). Και για να το αποσαφηνίσουμε ακόμα περισσότερο, αν τη χρονική τιμή **T΄** όπου

T< T΄< T+4ns υπάρξει νέα αλλαγή καταστάσεων πάλι σε ένα από τα τρία σήματα, αυτή θα φανεί στο σήμα Z(3) μετά από χρόνο **T΄+4ns.**



Σχ. 2.2 Κυμματομορφή Z(3) σε σχέση με τα σήματα εισόδου

Επανερχόμενοι στον αρχικό κώδικα όπου δεν ήταν δηλωμένη καμία χρονική καθυστέρηση, υποθέτουμε ότι αυτή είναι μηδενική (0ns) ή απείρως μικρή. Αυτή η καθυστέρηση ονομάζεται **delta delay (Δ)**.

Για να κατανοήσουμε γενικά τον κώδικα dataflow, ας υποθέσουμε ότι η είσοδος B αλλάζει κατάσταση την χρονική στιγμή T. Αυτό θα έχει ως αποτέλεσμα να «ενεργοποιηθούν» οι δηλώσεις 1,3,6. Οι νέες τιμές των Z(3),BBAR και Z(1) θα εμφανιστούν μετά από χρόνο T+Δ. Αφού όμως αλλάζει η τιμή του σήματος BBAR την χρονική στιγμή T+Δ τότε θα «ενεργοποιηθεί» και η 2η και η 4η δήλωση και οι νέες τιμές των Z(0) και Z(2) θα εμφανιστούν την χρονική στιγμή T+2Δ.
Συνοψίζοντας για την σχεδίαση dataflow, μπορούμε να πούμε ότι αυτή δεν είναι ιδανική για την περιγραφή ακολουθιακών κυκλωμάτων (δηλ. κυκλωμάτων που έχουν clock). Αντίθετα όπως υποδηλώνει και το όνομά του, αυτός ο τρόπος σχεδίασης είναι κατάλληλος για περιγραφή κυκλωμάτων που υπάρχει «ροή δεδομένων», όπως ALUs.

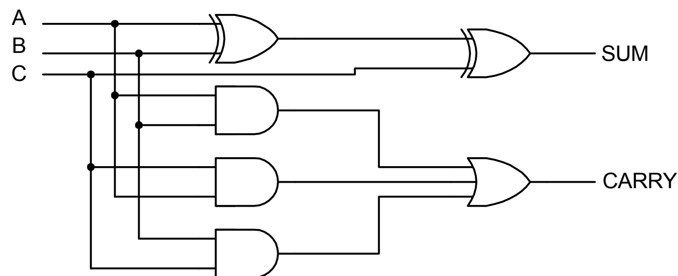
2.2.4 **Mixed Style of Modeling(Συνδυασμός των παραπάνω).**

Είναι δυνατό μία σχεδίαση να περιέχει και τους τρεις προαναφερθέντες τρόπους σχεδίασης. Δηλαδή να γίνεται χρήση components (Structural Modeling), παράλληλες δηλώσεις (Dataflow Modeling) καθώς και processes (Behavioral Modeling). Το πιο συνηθισμένο όμως είναι να συνυπάρχει ο dataflow τρόπος σχεδίασης με έναν από τους άλλους δύο.
Ακολουθεί περιγραφή ενός αθροιστή (1-bit) και με τους τρεις τρόπους σχεδίασης:

```
entity FULL_ADDER is
   port (
         A     : in bit;
         B     : in bit;
         CIN   : in bit;
         SUM   : out bit;
```

```
        COUT : out bit
        );
end FULL_ADDER;


architecture FA_MIXED of FULL_ADDER is
    component XOR2
        port ( P1, P2 : in bit; PZ: out bit);
    end component;
    signal S1 : bit;
begin
    X1: XOR2 port map ( A, B, S1);                 --structure
    process ( A, B, CIN)                           --behavior
        variable T1, T2, T3 : bit;
    begin
        T1:=A and B;
        T2:=B and CIN;
        T3:=A and CIN;
        COUT<= T1 or T2 or T3;
    end process;
    SUM <= S1 xor CIN;                             --dataflow
end FA_MIXED;
```

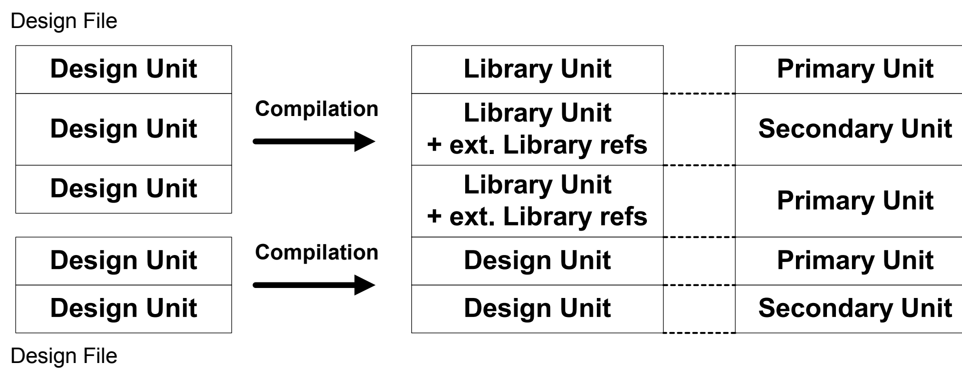


Σχ. 2.3 Πλήρης αθροιστής 1-bit

2.3 **Οργάνωση της σχεδίασης (Design Organization)**

2.3.1 **Βιβλιοθήκες**

Σχεδόν καμία σχεδίαση που κάνουμε δεν είναι απόλυτα αυτόνομη. Πάντα θα γίνεται αναφορά σε κάποιο block το οποίο αναπαριστά ένα component ή έναν ορισμό. Αυτά μπορεί να είναι αποθηκευμένα στην ίδια βιβλιοθήκη με το παρόν σχέδιο ή σε κάποια ευρύτερη (global) βιβλιοθήκη. Επίσης κάθε βιβλιοθήκη χωρίζεται σε επιμέρους τμήματα, ανάλογα με τα είδη αναφορών που έχει.
Η δομή μίας βιβλιοθήκης φαίνεται στο παρακάτω σχήμα:



Σχ. 2.4 Δομή Βιβλιοθήκης

Ένα design file περιέχει τον VHDL κώδικα ενός στοιχείου (element) του σχεδίου. Πολλά design files μπορεί να χρησιμοποιηθούν για την εφαρμογή πολλών elements του ιδίου σχεδίου. Κάθε τέτοιο αρχείο μπορεί και να περιέχει κομμάτι(α) της δομής ενός στοιχείου κυκλώματος. Αυτά τα κομμάτια (units) χωρίζονται σε:

- Πρωτεύοντα (primary units). Περιέχουν τον ορισμό ενός entity ή ένα πακέτο (package).
- Δευτερεύοντα (secondary units). Περιέχουν τον ορισμό ενός architecture body ή ένα πακέτο (package).

Τα primary units πρέπει να «περάσουν» το compilation και να αποθηκευτούν σε μια βιβλιοθήκη πριν ακολουθήσουν τα δευτερεύοντα με τα οποία συνδέονται. Το σύνολο αυτών των μονάδων απαρτίζουν τα library units.

2.3.2 **Packages (Πακέτα).**

Ο ορισμός πακέτων χρησιμοποιείται για την αποθήκευση ενός συνόλου κοινών στοιχείων (components), διαδικασιών (procedures), συναρτήσεων (functions), types, subtypes κ.ά. τα οποία χρησιμοποιούνται από τον σχεδιαστή συχνά. Δήλωση πακέτων (package declaration) καθώς και τα πακέτα τα ίδια (package body) είναι σχεδιαστικές μονάδες(design units), κατά συνέπεια μπορούν να κάνουν χρήση των δεδομένων άλλων πακέτων.

Ένα παράδειγμα πακέτου προσβάσιμο σε όλους τους χρήστες είναι το STANDARD. Αυτό δεν μπορεί να τροποποιηθεί από τον χρήστη και μαζί με το πακέτο TEXTIO, βρίσκονται μέσα στη βιβλιοθήκη STD. Η ειδική χρησιμότητα των πακέτων δεν φαίνεται εκ πρώτης όψεως. Όμως αξίζει να αναφερθεί ότι μερικά εξειδικευμένα πακέτα που προσφέρουν οι εταιρίες κατασκευής ASICs (Application Specific ICs) και FPGAs (Field Programmable Gate Arrays), μπορούν να βοηθήσουν τον σχεδιαστή να υλοποιήσει ένα σύστημα βασισμένος σε μια συγκεκριμένη τεχνολογία.

2.3.3 **Πρόσβαση σε βιβλιοθήκες και πακέτα**

Πρώτα απ' όλα να σημειωθεί ότι υπάρχουν design units μέσα σε βιβλιοθήκες που είναι «ορατά» σε κάθε σχεδίαση. Αυτά βρίσκονται στο STANDARD πακέτο στην STD βιβλιοθήκη καθώς και σε όλα τα πακέτα της WORK βιβλιοθήκης. Άμα τυχόν αυτά δεν ήταν ορατά στον χρήστη τότε κάθε σχεδίασή του έπρεπε να **προαναφέρει** τις ακόλουθες δηλώσεις:

```
library STD,WORK; --δήλωση βιβλιοθηκών
use STD.STANDARD.all      --δήλωση όλων των στοιχείων ενός πακέτου
```

Η πρώτη δήλωση ονομάζεται library clause και κάθε βιβλιοθήκη μπορεί να γίνει «ορατή» στη σχεδίαση του χρήστη με αυτόν τον τρόπο. Αντίστοιχα η δεύτερη δήλωση ονομάζεται use clause και καλεί συγκεκριμένα πακέτα ή στοιχεία πακέτων από μία βιβλιοθήκη. Όμως αν ο χρήστης θέλει να έχει πρόσβαση σε ένα αντικείμενο από ένα πακέτο, δεν είναι αναγκαίο να «φορτώσει» όλο το πακέτο αλλά μόνο το επίμαχο αντικείμενο. Ας υποθέσουμε ότι το αντικείμενο είναι μία συνάρτηση (procedure) ή ένα στοιχείο (component) με το όνομα DECODE και βρίσκεται στο πακέτο MY_PROCS μέσα στη βιβλιοθήκη MY_LIB. Τότε η πρόσβαση γίνεται ως εξής:

```
MY_LIB.MY_PROCS.DECODE;
```

3. Βασικά χαρακτηριστικά της γλώσσας VHDL

Η ανάλυση των βασικών χαρακτηριστικών της γλώσσας VHDL θα προκύψει μέσα από τον διαχωρισμό των λογικών ψηφιακών κυκλωμάτων σε δύο κύριες κατηγορίες. Στα συνδυαστικά λογικά κυκλώματα και στα ακολουθιακά λογικά κυκλώματα. Θα ακολουθήσουν και διάφορα παραδείγματα, ενδεικτικά για κάθε κατηγορία τα οποία θα βοηθήσουν στην καλύτερη κατανόηση της VHDL ως γλώσσας προγραμματισμού.

3.1 **Συνδυαστική Λογική Σχεδίαση (Combinational Logic Design)**

Τα κυκλώματα συνδυαστικής λογικής, αποτελούνται από απλές λογικές πύλες και οι έξοδοι αυτών των κυκλωμάτων εξαρτώνται πάντα από τις εισόδους τους με σχέσεις που καθορίζονται επακριβώς από κάποιες συναρτήσεις Boole. Ο τρόπος σχεδίασης κυκλωμάτων όπως 2bit πολυπλέκτες και 16x16 πολλαπλασιαστές θα καθορίζεται με βάση διάφορους παράγοντες. Αυτοί είναι:

- **Η πολυπλοκότητα της σχεδίασης.** Παράγοντες όπως το μέγεθος και ο αριθμός των λογικών πυλών του κυκλώματος μπορεί να αναγκάσουν τον σχεδιαστή να υιοθετήσει ένα συγκεκριμένο τρόπο σχεδίασης (Structural, Behavioral, Dataflow).
- **Η ευελιξία της σχεδίασης.** Αν το κύκλωμα μπορεί να περιγραφεί αλγοριθμικά ή με χρήση συναρτήσεων Boole κ.τ.λ., τότε το architecture body μπορεί να περιέχει περισσότερους από έναν τρόπο σχεδίασης.
- **Η τεχνολογία σχεδίασης.** Αν η σχεδίαση προορίζεται να υλοποιηθεί πάνω σε ένα, συγκεκριμένης τεχνολογίας, υλικό, τότε ο χρήστης θα πρέπει να κάνει χρήση προκατασκευασμένων στοιχείων (components) από βιβλιοθήκες.
- **Τα χαρακτηριστικά του κυκλώματος.** Αν το κύκλωμα πρέπει να απαρτίζεται από συγκεκριμένα στοιχεία, τότε είναι απαραίτητο ο σχεδιαστής να καταλήξει σε ένα δομικό (structural) τρόπο σχεδίασης για την υλοποίησή του.

Παρ’ όλο που είναι σημαντικό να λαμβάνουμε υπ’ όψιν όλους τους παραπάνω παράγοντες, πρέπει να σημειωθεί ότι τα συνδυαστικά κυκλώματα υλοποιούνται κατά κύριο λόγο με τον Dataflow τρόπο σχεδίασης. Σε επόμενο στάδιο, αυτά τα κυκλώματα ενώνονται μεταξύ τους, σε υψηλότερο επίπεδο, δομικά.

Τα παραδείγματα που θα ακολουθήσουν κάνουν χρήση όλων των τρόπων αρχιτεκτονικής σχεδίασης.

3.1.2 **Logical operators (Λογικές πράξεις)**

Η VHDL υποστηρίζει λογικές πράξεις για τους βασικούς της τύπους. **BIT, BOOLEAN** αλλά και για τους μονοδιάστατους πίνακές της **BIT_VECTOR**. Τα αποτελέσματα των λογικών αυτών πράξεων είναι του ιδίου τύπου και μεγέθους με τους τελεστέους. Πράξεις μεταξύ τύπων διαφορετικού μεγέθους απαγορεύονται.

Operator	***Function***
Not	Αντιστροφή
And	Και
Nand	Όχι-και
Or	Ή
Nor	Ούτε
Xor	Αποκλειστικό – Ή

Xnor	Αποκλειστικό – Ούτε

Πίνακας 3.1 Λογικές πράξεις στη VHDL

Η πράξη **not** έχει την μεγαλύτερη προτεραιότητα ενώ όλες οι άλλες λογικές πράξεις, έχουν την ίδια προτεραιότητα. Γι' αυτό το λόγο όταν σε μία δήλωση υπάρχουν πολλές λογικές πράξεις, είναι καλό να χρησιμοποιούμε παρενθέσεις για να δηλώνουμε την προτεραιότητα της κάθε μίας. Η πράξη not όμως δεν χρειάζεται παρένθεση. Ας δούμε μερικά παραδείγματα πράξεων με τελεστέους **BIT** ή **BOOLEAN**:

C<= A **nand** B; --**Νόμιμη** πράξη, δεν χρειάζεται παρένθεση.
C<= A **and not** B;--**Νόμιμη**,το **not** B έχει την υψηλότερη προτεραιότητα.
D<= (A **nand** B) **xor** C; --**Νόμιμη** πράξη, με παρένθεση.
D<= A **and** B **and** C; --**Νόμιμη**, όχι παρένθεση διότι είναι ιδίου τύπου.
D<= A **and** B **and** C; --**Νόμιμη**, όχι παρένθεση διότι είναι ιδίου (**non-inverting**)τύπου.
D<= A **nand** B **nand** C; --**Παράνομη**, διότι ιδίου (**inverting**)τύπου θέλει παρένθεση.
D<= A **and** B **or** C; --**Παράνομη**, διότι (**mixed**)τύπου θέλει παρένθεση.

Όσον αφορά τώρα πράξεις μεταξύ τελεστέων ιδίου ή διαφορετικού μεγέθους:

signal W,X : BIT_VECTOR(3 **downto** 0);
signal Y : BIT;
signal Z : BIT_VECTOR(4 **downto** 0)

W <= X **or** Y ; -- **Παράνομη** πράξη διότι το X έχει μέγεθος πίνακα 4 ενώ το Y έχει μέγεθος 1 (BIT).
Z <= W **and** X; -- **Παράνομη** πράξη όχι λόγω μεγέθους τελεστέων αλλά μεγέθους αποτελέσματος (W,X =4 & Z=5 bits)
Z(3)<=Y **xor** W(1); --**Νόμιμη**, διότι όλοι οι τελεστέοι αλλά και το αποτέλεσμα, είναι μεγέθους bit

3.1.3 **Relational operators (Σχεσιακές πράξεις).**

Οι σχεσιακές πράξεις ελέγχουν για ισότητα, ανισότητα και γενικά κάνουν συγκρίσεις μεταξύ τελεστέων. Όλες οι σχεσιακές πράξεις έχουν την ίδια προτεραιότητα και μεγαλύτερη από τις λογικές πράξεις. Το δε αποτέλεσμα είναι πάντα τύπου Boolean (True/False).Για παράδειγμα:

A<= B>C; -- Το A είναι True όταν το B είναι μεγαλύτερο του C.

F <= D=E **nand** (B=C); -- Το F είναι False αν το D = E και το B = C.

Operator	***Function***
=	Ισότητα
/=	Ανισότητα
<	Μικρότερο από…
<=	Μικρότερο ή ίσο από…
>	Μεγαλύτερο από…
>=	Μεγαλύτερο ή ίσο από...

Πίνακας 3.2 Σχεσιακές πράξεις στην VHDL.

3.1.4 **Multiplexer 4-to-1 (Πολυπλέκτης 4-σε-1).**

Με αυτό το παράδειγμα θα παρουσιαστούν τα βασικότερα στοιχεία (constructs) της VHDL που χρησιμοποιούνται στη σχεδίαση κυκλωμάτων συνδυαστικής λογικής. Ο 4 σε 1 πολυπλέκτης θα σχεδιαστεί και με τους τρεις τρόπους αρχιτεκτονικής σχεδίασης (behavioral, structural, dataflow), αποδεικνύοντας την ευελιξία της γλώσσας. Τα παρακάτω βασικά σημεία της VHDL θα γίνουν ορατά με αυτό το παράδειγμα:

- Ονόματα και identifiers
- Παράλληλες δηλώσεις (Concurrent statements)
  - Conditional signal assignments
  - Selected signal assignments
  - Component instantiations (Δηλώσεις στοιχείων)
- Σειριακές δηλώσεις (Sequential statements)
  - Case
  - If

3.1.4.1 **Λειτουργία του πολυπλέκτη 4-σε-1.**

Ο 4 σε 1 πολυπλέκτης επιλέγει ως έξοδο Y μία από τις εισόδους A, B, C, D βασισμένος στην κατάσταση των σημάτων ελέγχου (control lines) S0, S1. Οι λογικές συναρτήσεις του κυκλώματος παρουσιάζονται στον πίνακα 3.3. και το σχήμα 3.1 δείχνει τον πολυπλέκτη.

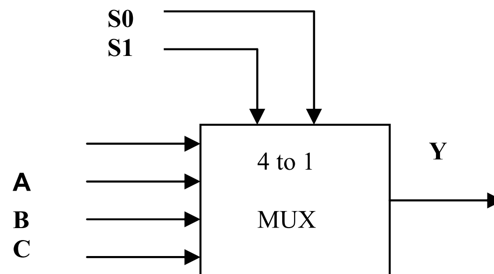
S1	***S0***	***Output***
0	*0*	***A***
0	1	***B***
1	0	***C***

1	1	***D***

Πίνακας 3.3 Συνάρτηση του 4 σε 1 πολυπλέκτη

Η λογική συνάρτηση του κυκλώματος και η οποία βγαίνει από τον πίνακα 3.3 είναι:

Y= A*S1΄*S0΄ + B*S1΄*S0 + C*S1*S0΄ + D*S1*S0



3.1.4.2 **Σχεδίαση του πολυπλέκτη 4-σε-1 σε VHDL.**

Επιλογή βιβλιοθηκών

Το πρώτο στάδιο είναι η επιλογή των κατάλληλων βιβλιοθηκών για την συγκεκριμένη σχεδίαση. Σε αυτό το παράδειγμα θα χρησιμοποιήσουμε objects τύπου bit και bit_vector τα οποία βρίσκονται στη βιβλιοθήκη STD και στο πακέτο STANDARD. Όμως αυτά τα objects καθώς και οι λογικές πράξεις που θα χρησιμοποιηθούν είναι ορατά σε κάθε σχέδιο, κατά συνέπεια δεν είναι απαραίτητη η χρήση **library clause** και **use clause**.

Ορισμός του entity

Το entity, όπως προαναφέραμε, το «κέλυφος» της σχεδίασης και λειτουργεί ως το I/O interface μεταξύ του κυκλώματος και του εξωτερικού περιβάλλοντος. Για το συγκεκριμένο παράδειγμα το entity είναι:

```
entity MUX4TO1 is
    port (
            A, B, C, D, S0, S1 : in bit;
            Y : out bit
            );
end MUX4TO1;
```

Architecture body

Το κύριο και μεγαλύτερο μέρος μιας σχεδίασης είναι το Architecture body του. Εδώ περιγράφει την λειτουργία του πολυπλέκτη με δομικό, συμπεριφερικό ή διαγράμματος ροής τρόπο (behavioral, structural, dataflow).

Dataflow design Style (Σχεδίαση με διάγραμμα ροής).
Για τον συγκεκριμένο τρόπο σχεδίασης και με βάση πάντα το entity MUX4TO1, θα παραθέσουμε δύο παραδείγματα, ενδεικτικά.

-- Concurrent signal assignment

```
architecture DATAFLOW1 of MUX4TO1 is
begin
    -- οι παρενθέσεις χρειάζονται όταν αναμιγνύονται λογικές πράξεις ίδιας
    --προτεραιότητας.
    Y<= ( A and not S1 and not S0) or (B and not S1 and S0) or (C and S1
          and not S0) or (D and S1 and S0);
end DATAFLOW1;
```

-- Conditional signal assignment

```
architecture DATAFLOW2 of MUX4TO1 is
begin
    Y <=  A when (S1='0' and S0='0') else
          B when (S1='0' and S0='1') else
          C when (S1='1' and S0='0') else
          D;
end DATAFLOW2;
```

Η πρώτη αρχιτεκτονική (dataflow1) χρησιμοποιεί δηλώσεις που παραπέμπουν στην συνάρτηση Boole του κυκλώματος. Αυτές οι δηλώσεις ονομάζονται **unconditional signal assignments**. Τέτοιες δηλώσεις είναι παράλληλες.

Αντίθετα στη δεύτερη αρχιτεκτονική (dataflow2) χρησιμοποιούνται άλλου είδους δηλώσεις. Τα **conditional signal assignments**. Εδώ γίνεται ανάθεση N τιμών σε ένα σήμα, το Y, ανάλογα με τις συνθήκες που ικανοποιούνται. Είναι επίσης δυνατό να ορίσουμε συγκεκριμένα στα conditional signal assignments ότι το σήμα (π.χ. το Y) παραμένει αμετάβλητο όταν μία συνθήκη γίνει αληθής (True). Αυτό γίνεται με τη χρήση της λέξης-κλειδί **unaffected**. Στο παρακάτω παράδειγμα οι δύο κλάδοι του conditional signal assignment θα αναθέσουν στην έξοδο της δήλωσης την ίδια αρχική τιμή του, A:

```
A <= A when B=3 else
       unaffected;
```

Η δήλωση αυτή δεν κάνει τίποτα παραπάνω από το να αναθέτει στο A το A. Κατά συνέπεια λειτουργεί σαν τη δήλωση **null** στα process. Επίσης είναι καλό να αναφερθεί ότι τα conditional signal assignments αντιστοιχούν στα Case μέσα στα process.

Behavioral design Style (Συμπεριφερικός τρόπος σχεδίασης).

Οι παρακάτω κώδικες δείχνουν δύο διαφορετικούς τρόπους κατασκευής ενός πολυπλέκτη 4 σε 1, με τον behavioral τρόπο σχεδίασης. Το δε entity MUX4TO1, παραμένει το ίδιο με μόνη αλλαγή τη χρήση του bit_vector S αντί των σημάτων S0, S1 τύπου bit.

```
 entity MUX4TO1 is
     port (  A, B, C, D       : in bit;
             S                : in bit_vector (1 downto 0);
     -- θα μπορούσαμε να γράψουμε bit_vector (0 to 1)
             Y : out bit
             );
 end MUX4TO1;

-- Η δήλωση CASE
architecture CASE1 of MUX4TO1 is
begin
     process (A, B, C, D, S(1 downto 0))
     begin
             case S is
                     when "00" => Y <= A;
                     when "01" => Y <= B;
                     when "10" => Y <= C;
                     when "11" => Y <= D;
             end case;
     end process;
end CASE1;

-- Η δήλωση IF - ELSE
architecture IF_ELSE of MUX4TO1 is
begin
     process (A, B, C, D, S(1 downto 0))
     begin
             if (S(1)='0' and S(0)='0') then
                     Y<=A;
             elsif (S(1)='0' and S(0)='1') then
                     Y<=B;
             elsif (S(1)='1' and S(0)='0') then
                     Y<=C;
             else
                     Y<=D;
             end if;
     end process;
```

```
end IF_ELSE;
```

Ο συμπεριφερικός (behavioral) τρόπος σχεδίασης γίνεται αντιληπτός από την πρώτη στιγμή, με την ύπαρξη της λέξης-κλειδί **process**. Όπως προαναφέραμε, μπορεί ένα process να είναι, στο σύνολό του, μια παράλληλη δήλωση, αλλά όλες οι δηλώσεις που αυτό έχει εσωτερικά εκτελούνται σειριακά. Συνήθως χρησιμοποιείται όταν η πολυπλοκότητα ενός κυκλώματος μας αναγκάζει να το προσεγγίσουμε από την πλευρά της συμπεριφοράς του. Δηλαδή το κύκλωμα το βλέπουμε ως ένα «μαύρο κουτί», από δομικής άποψης, και απλά περιγράφουμε την συμπεριφορά των εξόδων του σε σχέση με τις εισόδους του.

Επανερχόμενοι τώρα στα δύο παραδείγματά μας (για τον πολυπλέκτη 4σε1), το πρώτο υλοποιεί τον πολυπλέκτη με τη χρήση του **Case** και το δεύτερο με τη χρήση του **If – Else**. Η χρήση του Case δείχνει με απλό και καθαρό τρόπο την ευελιξία του καθώς κάθε συνθήκη μπορεί να διαχειρίζεται περισσότερες από μία δήλωση. (signal assignment statement). Σ' αυτό το παράδειγμα η Case ελέγχει και τις 4 περιπτώσεις του S. Όμως μερικές φορές δεν οδηγούν όλες οι περιπτώσεις (του ή των σημάτων που ελέγχει η Case) σε εκτέλεση δηλώσεων. Τότε χρησιμοποιείται το **when others => null;**. Η δεύτερη αρχιτεκτονική (If – Else) δίνει μία εναλλακτική λύση στη χρήση της Case, την If – Else δήλωση. Το If – Else ψάχνει όλες τις περιπτώσεις να δει ποια αληθεύει και μετά εκτελεί σειριακά τις δηλώσεις (μια ή περισσότερες) που αυτή η αληθής περίπτωση, περιέχει. Όπως η Case, έτσι και η If – Else μπορούν να εκτελέσουν σε κάθε κλάδο τους περισσότερες από μια δηλώσεις, σε αντίθεση με τα conditional signal assignments (τα παράλληλα αντίστοιχά τους).

Structural design Style (Δομικός τρόπος σχεδίασης).

Όπως ένα process υποδηλώνει μια behavioral σχεδίαση, έτσι και η ύπαρξη στοιχείων (component declaration και instantiation) υποδηλώνει την ύπαρξη δομικής (structural) σχεδίασης. Αντίθετα με τη συμπεριφερική σχεδίαση, η οποία υποδεικνύει το τι πρέπει να κάνει το κύκλωμα, η δομική σχεδίαση περιγράφει την λειτουργία του κυκλώματος δείχνοντας τον τρόπο με τον οποίο είναι ενωμένα τα εσωτερικά του στοιχεία.

Ο αριθμός των αρχιτεκτονικών δομών (architectural bodies) που μπορούν να δημιουργηθούν για ένα κύκλωμα είναι πολύ μεγάλος. Για την συγκεκριμένη σχεδίαση (του πολυπλέκτη 4σε1) κατασκευάστηκε ένα architectural body
μόνο με λογικές πύλες nand N εισόδων.

```
entity MUX4TO1 is
    port (  A, B, C, D      : in bit;
            S               : in bit_vector (1 downto 0);
    -- θα μπορούσαμε να γράψουμε bit_vector (0 to 1)
            Y : out bit
            );
end MUX4TO1;
```

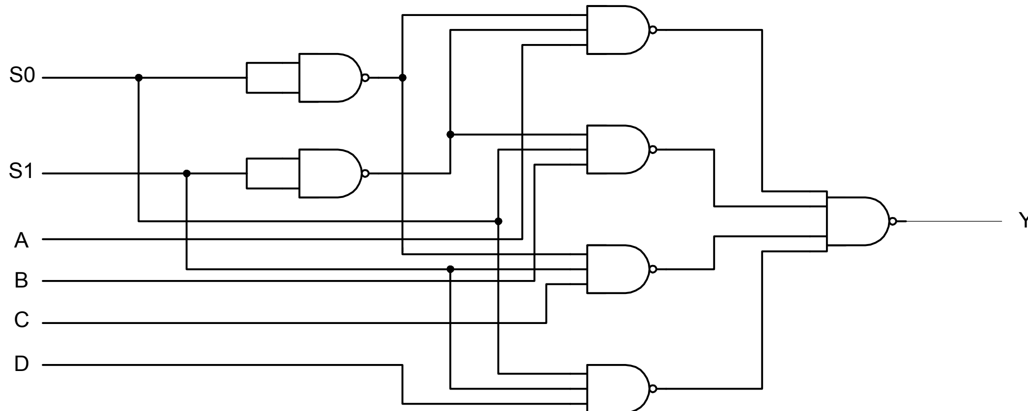
--Πολυπλέκτης 4 σε 1 με NAND πύλες

```
architecture STRUCTURAL1 of MUX4TO1 is
--Δήλωση component στο κομμάτι ορισμού (declarative part)
    component NAND_2
            port ( A,B : in bit; C: out bit);
    end component;
    component NAND_3
            port ( A,B,C : in bit; D: out bit);
    end component;
    component NAND_4
            port ( A,B,C,D : in bit; E: out bit);
    end component;
    signal 10,11,12,13,14,15 : bit;
begin
    -- Κατασκευή των αντίστροφων των σημάτων ελέγχου S(0) και S(1)
    INVERT0:NAND_2 port map (S(0),S(0),10);
    INVERT1:NAND_2 port map (S(1),S(1),11);
    -- Κατασκευή των ενδιάμεσων σημάτων
    PROD0:NAND_3 port map (11,10,A,12);
    PROD1:NAND_3 port map (11,S(0),B,13);
    PROD2:NAND_3 port map (S(1),10,C,14);
    PROD3:NAND_3 port map (S(1),S(0),D,15);
    -- Κατασκευή του τελικού σήματος
    SUM:NAND_4 port map (12,13,14,15,Y);
end STRUCTURAL1;
```

Κατά τη δομική σχεδίαση ενός κυκλώματος πρέπει πάντα να δηλώνουμε τα στοιχεία (components) τα οποία χρησιμοποιούμε. Το component declaration, πληροφορεί την αρχιτεκτονική μας ότι ένα συγκεκριμένο στοιχείο θα χρησιμοποιηθεί και παραθέτει τις εισόδους και τις εξόδους του. Το component instantiation, «καλεί» το στοιχείο που έχει δηλωθεί παραπάνω. Αυτή η αναφορά στο στοιχείο πρέπει να συνοδεύεται από ένα όνομα (μοναδικό για κάθε instantiation), μία λίστα με το I/O αυτού και πώς αυτή η αναφορά το χρησιμοποιεί (port mapping). Στην προηγούμενη αρχιτεκτονική, όπου χρησιμοποιήθηκαν μόνο πύλες nand η Boolean λογική της είναι:

Y= ((A * (S1)’ * (S0)’)’ * (B * (S1)’ * S0)’ * (C * S1 * (S0)’)’ * (D * S1 * S0)’)’

Τα σήματα 10,11,12,13,14,15 της σχεδίασης αποτελούν τα «σύρματα» που ενώνουν τις πύλες μεταξύ τους. Αυτό φαίνεται καθαρά και στο σχήμα 3.4



Σχ. 3.4 Πολυπλέκτης 4 σε 1 με τη χρήση λογικών πυλών NAND

3.1.4.3 **Συμπεράσματα από την σχεδίαση του MUX 4 to 1 σε VHDL.**

Στην προηγούμενη παράγραφο παρουσιάστηκαν διάφοροι τρόποι σχεδίασης ενός απλού πολυπλέκτη 4 σε 1, σε VHDL. Δεν συνιστώνται όλες αυτές οι προσεγγίσεις σε ένα αντίστοιχο κύκλωμα. Ο λόγος για τον οποίο έγινε αυτό, ήταν η αναφορά στα βασικότερα χαρακτηριστικά της VHDL πάνω στη σχεδίαση συνδυαστικών λογικών κυκλωμάτων. Αυτό από μόνο του υποδηλώνει ότι το θέμα δεν εξαντλείται εδώ. Υπάρχουν πολλοί συνδυασμοί των παραπάνω οι οποίοι θα μπορούσαν να είχαν υλοποιηθεί.

3.2 **Ακολουθιακή Λογική Σχεδίαση (Sequential Logic Design).**

Είναι γνωστό ότι στα ακολουθιακά κυκλώματα, η έξοδος δεν εξαρτάται μόνο από την παρούσα κατάσταση των εισόδων αλλά και από την προηγούμενή κατάσταση της εξόδου. Άρα τα ακολουθιακά κυκλώματα χαρακτηρίζονται από την χρονική τους εξάρτηση. Κάθε κύκλωμα με ρολόι (clock) είναι ακολουθιακό. Στην VHDL μπορούμε να περιγράψουμε σύγχρονα και ασύγχρονα ακολουθιακά κυκλώματα αλλά όχι με όλους τους τρόπους σχεδίασης. Ο dataflow τρόπος σχεδίασης περιγράφει, όπως έχουμε αναφέρει, την ροή δεδομένων σε συνεχή χρόνο και άρα δεν μπορεί να περιγράψει κυκλώματα εξαρτώμενα από τον χρόνο (ακολουθιακά). Ο δομικός (structural) τρόπος σχεδίασης μπορεί να χρησιμοποιηθεί σε ακολουθιακά κυκλώματα, αλλά τα στοιχεία (components) που θα καλέσει πρέπει να προέρχονται από «έτοιμες» βιβλιοθήκες. Αυτό οφείλεται στο ότι οι κύριες μονάδες-στοιχεία των ακολουθιακών κυκλωμάτων τα flip flops και τα latches είναι δύσκολο να κατασκευαστούν δομικά, κατά συνέπεια ο σχεδιαστής θα τα πάρει από διάφορες βιβλιοθήκες (ή θα τα κατασκευάσει σε behavioral). Συνεπώς με τον δομικό τρόπο σχεδίασης ακολουθιακών κυκλωμάτων περιορίζεται η ευελιξία της υλοποίησης.
Αντίθετα με τον behavioral τρόπο σχεδίασης είναι δυνατή η χρήση ειδικών constructs που έχει η VHDL. Αυτές οι ειδικές δηλώσεις τις VHDL έχουν και το τίμημά τους. Αν

δεν γίνει απόλυτη περιγραφή (με αυτά τα constructs) των σημάτων του κυκλώματος, ο compiler-synthesizer θα υλοποιήσει μεν το κύκλωμα αλλά όπως αυτός νομίζει καλλίτερα, περιορίζοντας πάλι τις δυνατότητες του σχεδιαστή.

3.2.1 **Προτερήματα του structural τρόπου σχεδίασης.**

Με τη χρήση του δομικού τρόπου σχεδίασης, ο χρήστης επιλέγει τα στοιχεία που αυτός νομίζει ότι ανταποκρίνονται καλλίτερα στη σχεδίασή του με τα ακόλουθα προτερήματα:

- Οι λειτουργίες των στοιχείων που θα χρησιμοποιηθούν και κυρίως των αποθηκευτικών στοιχείων είναι γνωστά στον χρήστη και δεν αφήνονται στην «διακριτική ευχέρεια» του synthesizer.
- Είναι γνωστός επίσης ο αριθμός των αποθηκευτικών στοιχείων που θα χρησιμοποιηθούν, ενώ με τον behavioral τρόπο σχεδίασης ο synthesizer είναι αυτός που τον καθορίζει.
- Τέλος ο χρήστης μπορεί από την αρχή να επιλέξει το «τεχνολογικό» πλαίσιο μέσα στο οποίο θα σχεδιάσει το κύκλωμά του.

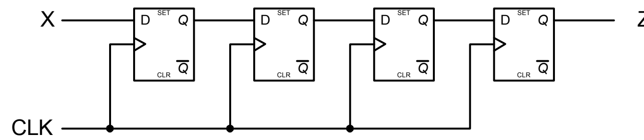
3.2.2 **4-bit Shift Register (4-bit Καταχωρητής Ολίσθησης).**

Ο καταχωρητής ολίσθησης είναι ένα απλό παράδειγμα ακολουθιακού κυκλώματος το οποίο θα παρουσιάσει τα βασικά στοιχεία (constructs) της VHDL πάνω στην υλοποίηση ακολουθιακών κυκλωμάτων. Τα παρακάτω θέματα θα παρουσιαστούν:

- Ονόματα πινάκων
- Πράξεις πινάκων
- Παράλληλες δηλώσεις
    - Process
    - For – Generate
- Σειριακές δηλώσεις
    - If
    - Wait

3.2.2.1 **Λειτουργία του 4-bit Shift Register.**

Ένας καταχωρητής ο οποίος μπορεί να «ολισθαίνει» τις πληροφορίες που περιέχει προς τη μία ή προς την άλλη κατεύθυνση λέγεται καταχωρητής ολίσθησης. Ένας τέτοιος καταχωρητής αποτελείται από μία αλυσίδα από flip flops συνδεδεμένα στη σειρά με την έξοδο του ενός να τροφοδοτεί την είσοδο του γειτονικού του. Όλα τα flip flops συνδέονται με κοινό ρολόι. Στην προκειμένη περίπτωση έχουμε 4 D flip flops στη σειρά. Τα flip flops αυτά είναι ακμοπυροδοτούμενα, δηλ. οι έξοδοί τους ενεργοποιούνται στην θετική ακμή του ρολογιού.



Σχ. 3.5 Shift Register 4-bit (Καταχωρητής ολισθήσης 4-bit)

3.2.2.2 **Σχεδίαση του 4-bit Shift Register.**

Επιλογή βιβλιοθηκών

Και σε αυτήν την περίπτωση οι βασικές ανάγκες τις σχεδίασης καλύπτονται από την STD βιβλιοθήκη και το STANDARD πακέτο. Όπως και στο παράδειγμα του πολυπλέκτη, οι μεταβλητές και τα σήματα που θα μας χρειαστούν μπορούν να ανήκουν στους τύπους bit και bit_vector. Εμείς θα χρησιμοποιήσουμε bit_vector πίνακα μεγέθους 4 (4 bit). Παρ' όλα αυτά γενικά είναι προτιμότερο να χρησιμοποιήσουμε **predefined** τύπους, δηλ. τύπους που εμείς θα έχουμε κατασκευάσει διότι με αυτόν τον τρόπο δεν θα χρειαστεί μεγάλος κόπος για να αλλάξουμε το μέγεθος του κυκλώματός μας (π.χ. 6-bit καταχωρητής ολίσθησης). Από κάτω δίνεται ένα πακέτο (user-defined) το οποίο έχει ως σκοπό την εξοικείωση με τη δημιουργία και την χρήση πακέτων:

```
package NEW_TYPE_DEFS is
    Subtype HALF_REG is BIT_VECTOR(1 to 4);
end NEW_TYPE_DEFS;
```

Το πακέτο αυτό (NEW_TYPE_DEFS) βρίσκεται στην βιβλιοθήκη USER_UTILS. Για να γίνει ορατό σε μία σχεδίαση πρέπει πριν την σχεδίαση να χρησιμοποιηθούν **library** και **use clauses**.

Ορισμός του entity

Γενικά κάθε D flip flop πρέπει να έχει preset και/ή clear σήματα εισόδου για να μπορεί το κύκλωμα να επανέρχεται πάντα σε μία γνωστή κατάσταση. Και στην behavioral αλλά και στην structural σχεδίαση, αυτό μπορεί να γίνει με την ύπαρξη ενός σήματος (π.χ. INIT). Το entity FOUR_SHIFT που ακολουθεί κάνει και χρήση του πακέτου NEW_TYPE_DEFS:

```
library USER_UTILS;
use USER_UTILS.NEW_TYPE_DEFS.all;
entity FOUR_SHIFT is
    port (
            X               :in bit;
            CLK,INIT        :in bit;
```

```
        Z            :out bit );
end FOUR_SHIFT;
```

Architecture body

Behavioral design Style (Συμπεριφερικός τρόπος σχεδίασης).

Στον κώδικα του architecture body, ορίζεται ένα σήμα (signal) το REG τύπου HALF_REG στο κομμάτι signal declaration. Η είσοδος X αποθηκεύεται στο REG(1) ενώ η έξοδος Z θα προέλθει από το σήμα REG(4). Το architecture body περιέχει δύο παράλληλες δηλώσεις, το process και μια (unconditional) signal assignment. Το process είναι το γνήσιο ακολουθιακό κύκλωμα καθώς αυτό είναι που θα χρειαστεί το ρολόι και η έξοδος του process,ανεστραμένη, μέσω του signal assignment θα δώσει την έξοδο του κυκλώματος.

Το δε entity FOUR_SHIFT παρουσιάστηκε παραπάνω.

```
architecture SIMPLE of FOUR_SHIFT is;
signal REG:HALF_REG;
begin
   process(X,CLK,INIT)
   begin
         --Δήλωση ότι το ρολόι (clk) είναι ακμοπυροδοτούμενο
         if (clk’event and clk=’1’) then
               if INIT=’1’ then
                     REG<= X & REG(1 to 3);    --concatenation
               else
                     REG<= “1000”;
               end if;
         end if;
   end process;
   Z<= not REG(4);
end SIMPLE;
```

Αναλύοντας τώρα το εσωτερικό του process και κυρίως το εξωτερικού if-branch,παρατηρούμε ότι αυτό ελέγχει πότε το ρολόι πηγαίνει στην κατάσταση ‘1’. Μόλις αυτό συμβεί και αν το σήμα INIT=’1’, τότε το σήμα REG (4μπιτο) αποκτά νέα τιμή : την είσοδο X μαζί με τα τρία τελευταία του bit (concatenation). Αν το INIT είναι ‘0’, τότε εμείς δηλώνουμε το πώς πρέπει να συμπεριφερθεί το κύκλωμά μας (να αποκτήσει ο REG την τιμή ”1000”), δηλ. το κύκλωμα να πάει σε κατάσταση RESET, όπως εμείς την ορίζουμε. Με αυτόν τον τρόπο το σήμα REG μετατρέπεται σ’ ένα καταχωρητή ολίσθησης ο οποίος αλλάζει τιμή σε κάθε θετική ακμή του ρολογιού (clk’event and clk=’1’).

Η δήλωση **if (clk’event and clk=’1’) then…** μπορεί να αντικατασταθεί από την **wait until (clk’event and clk=’1’)**.

Structural design Style (Δομικός τρόπος σχεδίασης).

Μια δομική περιγραφή του 4μπιτου καταχωρητή ολίσθησης μπορεί να γίνει με δύο τρόπους. Ο πιο «ευθύς» τρόπος είναι να προμηθευτούμε τα απαραίτητα στοιχεία αποθήκευσης από βιβλιοθήκες. Αν αυτά δεν υπάρχουν τότε ο χρήστης θα αναγκαστεί να τα δημιουργήσει (με behavioral τρόπο). Αυτός είναι ο δεύτερος τρόπος. Ακολουθώντας τώρα τον πρώτο τρόπο, ο κώδικας είναι:

```
entity FOUR_SHIFT is
    port (
            X             :in bit;
            CLOCK,INIT    :in bit;
            Z             :out bit );
end FOUR_SHIFT;

architecture STRUCTURAL of FOUR_SHIFT is;
component DFF
    port (D, CLK, PC : in bit; Q, QBAR :out bit);
end component;
signal Z, ZBAR : bit_vector(1 to 3);
begin
    DFF1 : DFF port map (X, CLOCK, INIT, Z(1), ZBAR(1));
    DFF2 : DFF port map (Z(1), CLOCK, INIT, Z(2), ZBAR(2));
    DFF3 : DFF port map (D=>Z(2), CLK=>CLOCK, Q=>Z(3),
    QBAR=>ZBAR(3) , PC=>INIT);
    DFF4 : DFF port map (Z(3), CLK=>CLOCK, INIT, Q=>open,Y);
end STRUCTURAL;
```

Είναι σημαντικό να παρατηρήσουμε ότι το port mapping των στοιχείων που χρησιμοποιούμε, μπορεί να γίνει με διάφορους τρόπους. Είτε με απλή παράθεση των σημάτων και των εισόδων/εξόδων αλλά πάντα με την σωστή σειρά όπως αυτή υποδηλώνεται από τον αρχικό ορισμό του DFF (βλέπε DFF1, DFF2), ή με ανάθεση των σημάτων και των I/O του σχεδίου στις εισόδους εξόδους του component (βλέπε DFF4). Σε αυτήν την περίπτωση δεν έχει σημασία η σειρά τους στο port mapping. Κλείνοντας πρέπει να αναφέρουμε ότι αν δε θέλουμε να χρησιμοποιήσουμε μια είσοδο ή έξοδο ενός στοιχείου τότε κατά το port mapping μπορούμε να βάλουμε στη θέση του τη λέξη **open**.

4. Βιβλιογραφία

i. " A VHDL PRIMER " Revised Edition
J. Bhasker

ii. " VHDL " Second Edition
Douglas L. Perry

iii. " A DESIGNER'S GUIDE TO VHDL "
Peter J. Ashenden

iv. " THE VHDL COOKBOOK" First Edition
Peter J. Ashenden