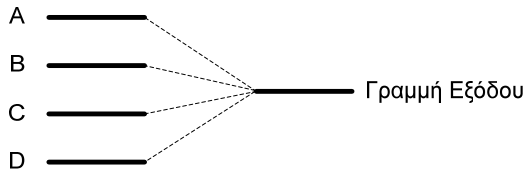
ΑΣΚΗΣΗ 5

Συνοδευτικά αρχεία : 5_1.v, 5_2.v, 5_3.v

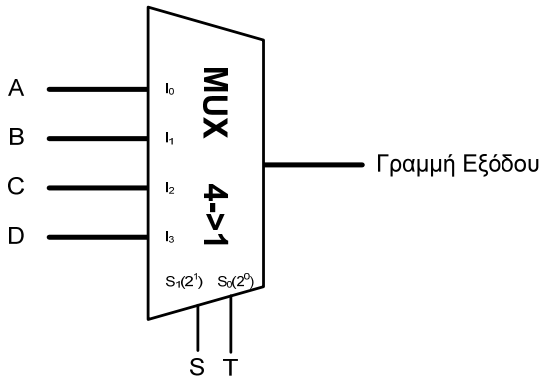
Μέρος 1 : Εισαγωγή

Σκοπός της 5ης άσκησης είναι να καταλάβετε τις έννοιες του ιεραρχικού σχεδιασμού και των επιπέδων ιεραρχίας. Ενα επίπεδο ιεραρχίας σε ένα σχεδιασμό, τον περιγράφει χρησιμοποιώντας άλλα πιο κάτω επίπεδα για τη περαιτέρω ανάλυσή του. Το ίδιο αυτό επίπεδο μπορεί να χρησιμοποιηθεί ως υποσχεδιασμός ενός μεγαλύτερου σχεδιασμού. Καλύτερα ας τα δούμε αυτά στη πράξη.

Ας υποθέσουμε ότι αρχικός μας στόχος είναι να περιγράψουμε και να εξομοιώσουμε ένα πολυπλέκτη 4 σε 1. Ο πολυπλέκτης 4 σε 1 χρησιμοποιείται για την επιλογή μιας από 4 πιθανές πηγές δεδομένων. Στο παρακάτω σχήμα φαίνονται οι πηγές δεδομένων A, B, C και D και μία γραμμή την οποία αυτές οι πηγές θέλουν να οδηγήσουν.



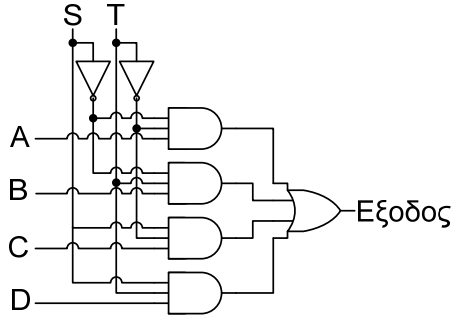
Προφανώς δε θα πρέπει να επιτρέψουμε σε 2 από αυτές τις πηγές ταυτόχρονα να οδηγούν τη γραμμή εξόδου, συνεπώς χρειαζόμαστε ένα κύκλωμα επιλογής βάσει κάποιας συνθήκης. Αυτό το κύκλωμα είναι ο πολυπλέκτης.



Ο πολυπλέκτης επιλέγει ποια από τις γραμμές εισόδου του θα συνδεθεί στη γραμμή εξόδου, χρησιμοποιώντας ως συνθήκη τη τιμή των σημάτων επιλογής. Προσέξτε ότι οι δύο γραμμές των σημάτων επιλογής δεν είναι ισοδύναμες. Η S_1 στο παραπάνω σχήμα έχει διαφορετικό βάρος (2^1) από τη S_0 (2^0). Οταν η S_1 είναι ενεργοποιημένη, στο 1 δηλαδή, επιβάλλει ως πιθανές εξόδους τις εισόδους I_2 ή I_3. Οταν είναι ενεργοποιημένη η S_0 επιβάλλει ως πιθανές εξόδους τις I_1 ή I_3. Αν λοιπόν συνδέσουμε τις πηγές μας όπως στο παραπάνω σχήμα στις εισόδους του πολυπλέκτη και τις μεταβλητές S και T στις εισόδους επιλογής, τότε μπορούμε να πούμε ότι στην έξοδο θα συνδεθεί η πηγή που είναι συνδεδεμένη στην είσοδο $I_{2 \times S + T}$. Μπορούμε εναλλακτικά να πούμε ότι η γραμμή C συνδέεται στην έξοδο μόνο όταν S=1 και T=0, δηλαδή όταν ισχύει η λογική συνάρτηση S(~T) και φυσικά η έξοδος θα έχει τιμή S(~T)C. Κάνοντας το ίδιο για όλες τις εισόδους παίρνουμε ότι η έξοδος αφού μπορεί να οδηγείται από κάποια από αυτές τις εισόδους θα υλοποιεί τη συνάρτηση :

$$\text{Εξοδος} = (\sim S)(\sim T)A + (\sim S)TB + S(\sim T)C + STD$$

Με βάση τη παραπάνω εξίσωση μπορούμε να φτιάξουμε το ακόλουθο σχήμα για το πολυπλέκτη 4 σε 1 :



Οι παρακάτω κώδικες περιγράφουν τον πολυπλέκτη αφενώς με δομικό (structural) τρόπο και αφετέρου βάσει των λογικών εξισώσεών του ή της συμπεριφοράς του.

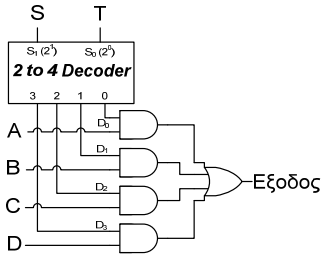
```
module mux4_2_1_struct (A, B, C, D, S, T, O);
  input A, B, C, D, S, T;
  output O;

  not i0 (Sn, S);
  not i1 (Tn, T);
  and i2 (P1, Sn, Tn, A);
  and i3 (P2, Sn,  T, B);
  and i4 (P3,  S, Tn, C);
  and i5 (P4,  S,  T, D);
  or  i6 (O, P1, P2, P3, P4);
endmodule


module mux4_2_1_equations (A, B, C, D, S, T, O);
  input A, B, C, D, S, T;
  output O;

  assign O = A&(~S)&(~T) | B&(~S)&T | C&S&(~T) | D&S&T;
endmodule


module mux4_2_1_behavioral (A, B, C, D, S, T, O);
  input A, B, C, D, S, T;
  output O;

  assign O = (~S) ? (~T) ? A : B : (~T) ? C : D ;
endmodule
```

Ας προσπαθήσουμε να εξομοιώσουμε όλους αυτούς τους κώδικες μαζί για να βεβαιωθούμε ότι λειτουργούν με τον ίδιο τρόπο. Ας βάλουμε στην είσοδο A ένα ρολόι, στη B ένα με διπλάσια περίοδο, στη C ένα με τετραπλάσια και στη D ένα με οκταπλάσια. Στο testbench μας θα επιλέγουμε κάθε 2000 χρονικές στιγμές και μια νέα πηγή.

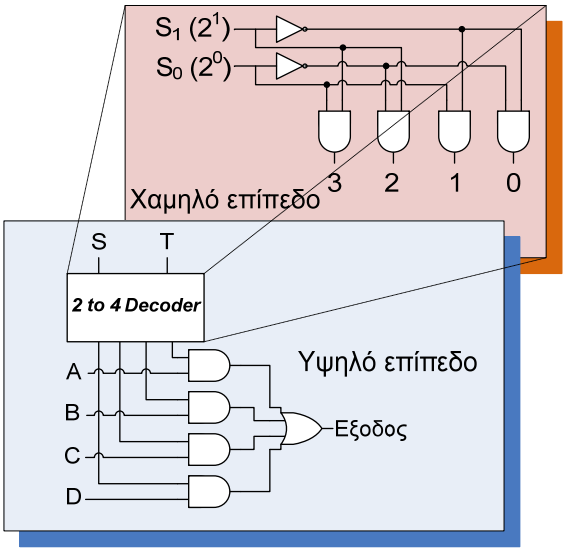
```
module testall ();
  reg A, B, C, D;
  reg S, T;
  wire O1, O2, O3;

  mux4_2_1_struct     i0 (A, B, C, D, S, T, O1);
  mux4_2_1_equations  i1 (A, B, C, D, S, T, O2);
  mux4_2_1_behavioral i2 (A, B, C, D, S, T, O3);

  initial begin A=0; B=0; C=0; D=0; S=0; T=0; end

  always #  20 A=~A;
  always #  40 B=~B;
  always #  80 C=~C;
  always # 160 D=~D;

  always #1000 T=~T;
  always #2000 S=~S;
endmodule
```

ΠΑΡΑΔΟΤΕΟ 1 : (χρησιμοποιείστε το αρχείο 5_1.v) Αφού εκτελέσετε εξομοίωση του παραπάνω κώδικα για 4500 χρονικές στιγμές, δώστε ένα screenshot των κυματομορφών εξομοίωσης. Θα πρέπει να συμπεριλάβετε όλα τα σήματα του module testall.

Στο παρακάτω σχήμα φαίνεται μια εναλλακτική υλοποίηση για τις εξισώσεις του πολυπλέκτη 4 σε 1. Για να δημιουργήσουμε τα σήματα (~S) & (~T), (~S) & T, S & (~T) και S & T, μπορούμε να χρησιμοποιήσουμε έναν αποκωδικοποιητή 2 σε 4 (δείτε και τη προηγούμενη άσκηση περί αποκωδικοποιητών). Μία μόνο έξοδος του αποκωδικοποιητή είναι ανάλογα με τη τιμή των S και T στο λογικό 1, οπότε μόνο μία από τις πιθανές τιμές εισόδους θα περάσει στην έξοδο.



Προσέξτε ότι στο παραπάνω σχήμα, δημιουργείται μια σχεδιαστική ιεραρχία. Οπως δηλαδή χρησιμοποιούμε τις AND και τις OR σα δομικά στοιχεία, χρησιμοποιήσαμε και τον αποκωδικοποιητή. Και ναι μεν αυτό είναι απόλυτα θεμιτό, αλλά το ερώτημα που προκύπτει είναι αν ο αποκωδικοποιητής υπάρχει ως βασικό δομικό στοιχείο στη Verilog. Δυστυχώς μια γλώσσα δε θα μπορούσε να συμπεριλάβει τα πάντα ως δομικά στοιχεία. Ακόμα κι αν το προσπαθούσε για τα ευρέως χρησιμοποιούμενα στοιχεία, πάντα ο κάθε σχεδιαστής θα ήθελε να μπορούσε να ορίσει δικά του δομικά στοιχεία τα οποία θα χρησιμοποιούσε στη συνέχεια για τη περιγραφή μεγαλύτερων σχεδιασμών. Αυτή η διαδικασία μοιάζει αρκετά με τη συγγραφή λογισμικού. Οι βιβλιοθήκες στη C όσο πλούσιες κι αν είναι δε μπορούν να συμπεριλάβουν όλες τις πιθανές συναρτήσεις που τυχόν θα ήθελε ο προγραμματιστής. Ετσι δίνεται η δυνατότητα στο προγραμματιστή να ορίσει τις δικές του συναρτήσεις που μπορεί να τις χρησιμοποιεί για να φτιάξει μεγαλύτερες ή να τις βάζει σε μια δικιά του βιβλιοθήκη. Επιστρέφοντας στο σχήμα, συνεπώς χρειάζεται και ένα δεύτερο επίπεδο ιεραρχίας στο παραπάνω σχήμα. Ενα επίπεδο που θα εξηγεί πως είναι φτιαγμένος ο decoder. Αυτό είναι ένα πιο χαμηλό επίπεδο της ιεραρχίας του σχεδιασμού μας. Το παρακάτω σχήμα απεικονίζει αυτή τη διεπίπεδη ιεραρχία.



Ας προσπαθήσουμε να περιγράψουμε το πολυπλέκτη με τη παραπάνω ιεραρχική προσέγγιση. Ας ξεκινήσουμε πρώτα από τον αποκωδικοποιητή, περιγράφοντάς τον βάσει των λογικών εξισώσεών του.

```
module decoder2_2_4 (S1, S0, O0, O1, O2, O3);
  input S1, S0;
  output O0, O1, O2, O3;

  assign O0 = ~S1 & ~S0;
  assign O1 = ~S1 & S0;
  assign O2 = S1 & ~S0;
  assign O3 = S1 & S0;
endmodule
```

Τώρα που έχουμε τη περιγραφή του αποκωδικοποιητή μπορούμε με δομικό τρόπο να περιγράψουμε το πολυπλέκτη.

```
module mux4_2_1_decoder_based (A, B, C, D, S, T, O);
  input A, B, C, D, S, T;
  output O;
  wire D0, D1, D2, D3;

  decoder2_2_4 i0 (S, T, D0, D1, D2, D3);
  and          i1 (P1, D0, A);
  and          i2 (P2, D1, B);
  and          i3 (P3, D2, C);
  and          i4 (P4, D3, D);
  or           i5 (O, P1, P2, P3, P4);
endmodule
```

Οι διαφοροποιήσεις του παραπάνω κώδικα σε σχέση με τους δομικούς τρόπους που έχετε δει έγκεινται στην εντολή `decoder2_2_4 i0 (S, T, D0, D1, D2, D3);` Με την εντολή αυτή χρησιμοποιούμε ένα αντίγραφο (instance) της οντότητας αποκωδικοποιητή. Το αντίγραφο θα έχει το όνομα `i0`. Προσέξτε ότι κατά τη δημιουργία του αντιγράφου η σειρά αναγραφής των σημάτων διασύνδεσής του έχει φοβερή σημασία. Το σήμα S δηλαδή που αναγράφεται πρώτο, θα συνδεθεί στην είσοδο S_1 του αποκωδικοποιητή καθώς αυτή είναι το πρώτο port (είσοδος / έξοδος) κατά τη δήλωση του αποκωδικοποιητή (γραμμή κώδικα `module decoder2_2_4 (S1, S0, O0, O1, O2, O3);`). Για τον ίδιο λόγο το T θα

συνδεθεί στο S_0 του αποκωδικοποιητή και οι έξοδές του O_0-O_3 στις γραμμές D_0-D_3 του υψηλότερου ιεραρχικά σχεδιασμού. Αντίθετα δηλαδή με τις πρωταρχικά δομικά στοιχεία της Verilog που πάντα έχουν την έξοδο πρώτα και μετά τις εισόδους (για παράδειγμα στη γραμμή `and i0 (x, a, b);` γνωρίζουμε ότι το `x` είναι η έξοδος και οι `a, b` είσοδοι) στις δικές μας συναρτήσεις η σειρά αναγραφής των ports πρέπει να είναι αντίστοιχη με αυτή της δήλωσης του module. Για να εξομοιώσουμε το πολυπλέκτη που αποτελείται από 2 ιεραρχικά επίπεδα μπορούμε να χρησιμοποιήσουμε τον παρακάτω κώδικα :

```
module testhier ();
  reg A, B, C, D;
  reg S, T;
  wire O1;

  mux4_2_1_decoder_based i0 (A, B, C, D, S, T, O1);

  initial begin A=0; B=0; C=0; D=0; S=0; T=0; end

  always #  20 A=~A;
  always #  40 B=~B;
  always #  80 C=~C;
  always # 160 D=~D;

  always #1000 T=~T;
  always #2000 S=~S;
endmodule
```

Αξίζει να παρατηρήσετε ότι κατά τη διαδικασία εκκίνησης της εξομοίωσης ο εξομοιωτής αντιλαμβανόμενος την ιεραρχία του σχεδιασμού θα προσπαθήσει να βρει τα object αρχεία όλων των υποσχεδιασμών και θα σας δώσει τα αντίστοιχα μηνύματα :

```
# vsim work.testhier
# Loading work.testhier
# Loading work.mux4_2_1_decoder_based
# Loading work.decoder2_2_4
```

Αυτή η διαδικασία μοιάζει πάρα πολύ με τη διαδικασία του linking του compiler κατά τη συμβολομετάφραση πηγαίου κώδικα.

ΠΑΡΑΔΟΤΕΟ 2 : (Χρησιμοποιείστε το αρχείο 5_2.v) Αφού εκτελέσετε εξομοίωση του παραπάνω κώδικα για 4500 χρονικές στιγμές, δώστε ένα screenshot των κυματομορφών εξομοίωσης. Θα πρέπει να συμπεριλάβετε όλα τα σήματα του module testhier.

Μέρος 2 : Ζητούμενα

Ζητείται να σχεδιαστεί ένας πολυπλέκτης από 8 εισόδους (I_7-I_0) σε μία έξοδο (O), χρησιμοποιώντας ως δομικά στοιχεία πολυπλέκτες 2 σε 1 ή / και πολυπλέκτες 4 σε 1. Στη περίπτωση των πολυπλεκτών 4 σε 1 εξετάστε και ιεραρχικές υλοποιήσεις τους με δομικά στοιχεία αποκωδικοποιητές ή μικρότερους πολυπλέκτες.

Μέρος 3 : Ενδεικτική Λύση

Ένας πολυπλέκτης από $2^κ$ πηγές σε 1 γραμμή εξόδου χρησιμοποιεί κ σήματα επιλογής. Στο παράδειγμά μας είναι κ=3. Άρα το ζητούμενο κύκλωμα θα έχει τις γραμμές επιλογής S_2, S_1 και S_0 με βάρη 2^2, 2^1 και 2^0 αντίστοιχα. Θα εκτελεί δε τη λογική συνάρτηση (δες μέρος 1 για το πως προκύπτει) :

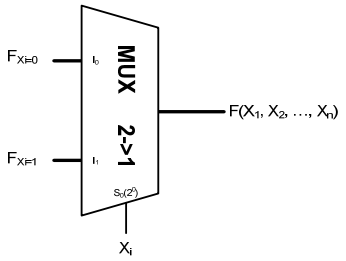
$$O = S_2S_1S_0I_7 + S_2S_1(\sim S_0)I_6 + S_2(\sim S_1)S_0I_5 + S_2(\sim S_1)(\sim S_0)I_4 + (\sim S_2)S_1S_0I_3 + (\sim S_2)S_1(\sim S_0)I_2 + (\sim S_2)(\sim S_1)S_0I_1 + (\sim S_2)(\sim S_1)(\sim S_0)I_0$$

Ας προσπαθήσουμε να αλλάξουμε λίγο τη μορφή αυτής της συνάρτησης ώστε να μπορεί να υλοποιηθεί χρησιμοποιώντας πολυπλέκτες μικρότερου μεγέθους. Μπορούμε να ξαναγράψουμε τη παραπάνω συνάρτηση ως :

$$O = S_2 [S_1S_0I_7 + S_1(\sim S_0)I_6 + (\sim S_1)S_0I_5 + (\sim S_1)(\sim S_0)I_4] + (\sim S_2) [S_1S_0I_3 + S_1(\sim S_0)I_2 + (\sim S_1)S_0I_1 + (\sim S_1)(\sim S_0)I_0] = S_2F_{S2=1} + (\sim S_2)F_{S2=0.}$$

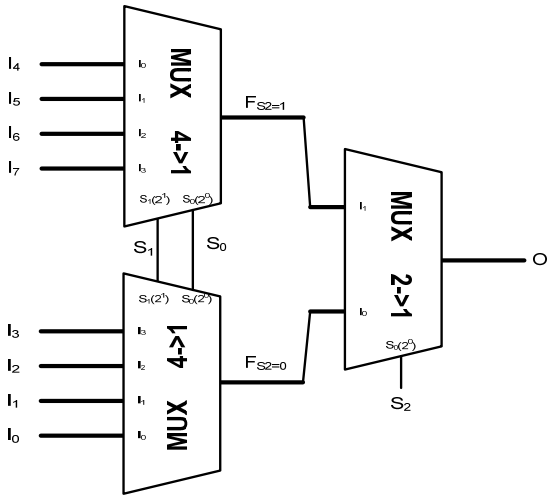
(<u>Απαραίτητη διευκρίνιση :</u>

Παρότι στην εξίσωση των πολυπλεκτών το παραπάνω βήμα φαίνεται να προέκυψε με την ανάστροφη εφαρμογή της επιμεριστικής ιδιότητας του AND ως προς το OR, είναι απλά μια ειδική περίπτωση του θεωρήματος του Shannon, το οποίο μας λέει ότι δοθείσης μιας συνάρτησης F των μεταβλητών X_1, X_2, …, X_n, αυτή μπορεί να γραφεί ως σύνθετη συνάρτηση 2 υποσυναρτήσεων των n-1 μεταβλητών ως $F(X_1, X_2, …, X_n) = (\sim X_i)F_{Xi=0} + X_i F_{Xi=1}$, όπου $F_{Xi=0}$ και $F_{Xi=1}$ είναι οι υποσυναρτήσεις που προκύπτουν θέτοντας ως τιμή της μεταβλητής X_i 0 και 1 αντίστοιχα. Επειδή η μορφή της συνάρτησης που προκύπτει από το θεώρημα Shannon αντιστοιχεί σε υλοποίηση με έναν πολυπλέκτη 2 σε 1 :



και μπορεί να εφαρμοστεί επαναληπτικά πάνω στις προκύπτουσες υποσυναρτήσεις ($F_{Xi=0}$ και $F_{Xi=1}$), στην ουσία έχουμε μια διαισθητική απόδειξη του ότι μπορούμε να υλοποιήσουμε το όποιο συνδυαστικό κύκλωμα αποκλειστικά με πολυπλέκτες).

Ας δούμε λίγο πιο προσεκτικά τη συνάρτηση $F_{S2=1} = S_1S_0I_7 + S_1(\sim S_0)I_6 + (\sim S_1)S_0I_5 + (\sim S_1)(\sim S_0)I_4$. Παρατηρούμε ότι πρόκειται για τη συνάρτηση εξόδου ενός πολυπλέκτη 4 σε 1, με πηγές δεδομένων I_7-I_4 και σήματα επιλογής S_1 και S_0. Το ίδιο ισχύει για την $F_{S2=0}$ που αντιστοιχεί στη συνάρτηση εξόδου ενός πολυπλέκτη 4 σε 1, με πηγές δεδομένων I_3-I_0 και σήματα επιλογής S_1 και S_0. Τέλος η εξίσωση εξόδου του πολυπλέκτη 8 σε 1 αντιστοιχεί σε έναν πολυπλέκτη 2 σε 1 με σήμα επιλογής το S_2 και πηγές δεομένων τις $F_{S2=1}$ και $F_{S2=0}$. Δηλαδή έχουμε το παρακάτω σχήμα :



- Αξίζει να σταματήσετε λίγο και να κάνετε προσεκτική παρατήρηση της συνδεσμολογίας στο παραπάνω σχήμα. Το I_7 και το I_3 οδηγούνται στις εισόδους I_3 των 4 σε 1 πολυπλεκτών γιατί πρέπει να εμφανιστούν στις αντίστοιχες εξόδους μόνο όταν $S_1 = S_0 = 1$.
- Για την υλοποίηση στο φυσικό κόσμο θα χρειαστούν 2 αντίγραφα πολυπλέκτη 4 σε 1. Η περιγραφή σε Verilog όμως δε χρειάζεται να έχει 2 φορές τον ίδιο κώδικα. Αρκεί να γράψετε το κώδικα για τον πολυπλέκτη 4->1 μία φορά και οι δύο πολυπλέκτες του πιο πάνω σχήματος είναι απλά αντίγραφα (με διαφορετικές παραμέτρους καθένα) του ίδιου κώδικα. Ακριβώς το ίδιο κάνατε μέχρι ώρας όταν χρησιμοποιούσατε τις πρωταρχικές συναρτήσεις (primitives) της γλώσσας, απλά δεν είχατε γράψει το κώδικα γι' αυτές. Αυτό είναι και ένα από τα πιο σημαντικά πλεονεκτήματα της χρήσης s/w για το σχεδιασμό υλικού. Ότι γράψετε είναι επαναχρησιμοποιήσιμο (reusable).
- Χωρίς να το καταλάβετε έχετε δημιουργήσει ένα επιπλέον ιεραρχικό επίπεδο στο σχεδιασμό σας. Ας υποθέσουμε ότι επιλέγετε τον διεπίπεδο ιεραρχικό σχεδιασμό για τον πολυπλέκτη 4 σε 1. Ο σχεδιασμός σας αποτελείται από 3 επίπεδα :
    - Στο πιο χαμηλό υπάρχουν οι περιγραφές του αποκωδικοποιητή 2 σε 4 και του πολυπλέκτη 2 σε 1.
    - Στο 2^o επίπεδο, η περιγραφή του πολυπλέκτη 4 σε 1 που χρησιμοποιεί τον αποκωδικοποιητή.
    - Στο υψηλότερο, περιγράφεται ο πολυπλέκτης 8 σε 1 με διασύνδεση των πολυπλεκτών 4 σε 1 και 2 σε 1.

  Φυσικά θα μπορούσατε να επιλέξετε οποιαδήποτε άλλη περιγραφή για τον πολυπλέκτη 4 σε 1 ιεραρχική ή μη. Κάθε επιλογή δημιουργεί και μια διαφορετική ιεραρχία και συσχέτιση ιεραρχικών επιπέδων.
- Τέλος, θα πρέπει να μπορείτε να διακρίνετε τις έννοιες του τρόπου περιγραφής και της ιεραρχίας. Δηλαδή αν το κάθε επίπεδο περιγράφεται με δομικό ή behavioural τρόπο δεν επηρεάζει σε τίποτε την επαναχρησιμοποίησή του.

Παρακάτω δίνεται ο απαιτούμενος κώδικας για διάφορες λύσεις.

Λύση 1 : Behavioural μη ιεραρχικός κώδικας (δεν απαντά στα ζητούμενα της άσκησης)

```
module mux8_2_1_beh(I, S, O);
  input [7:0] I; // οι πηγές μας σχηματίζουν μια αρτηρία 8 σημάτων
  input [2:0] S; // τα 3 σήματα επιλογής της πηγής
  output      O;

  always @(I or S)
    case (S)
      0 : O = I[0];
      1 : O = I[1];
      2 : O = I[2];
      3 : O = I[3];
      4 : O = I[4];
      5 : O = I[5];
      6 : O = I[6];
      7 : O = I[7];
    endcase
endmodule
```

Λύση 2 : Διεπίπεδος ιεραρχικός κώδικας. Πολυπλέκτης 4 σε 1 behavioural και ο 8 σε 1 behavioural & δομικά

```
module mux4_2_1_behavioral (I0, I1, I2, I3, S1, S0, F);
  input I0, I1, I2, I3, S1, S0;
  output F;

  assign F = (~S1) ? (~S0) ? I0 : I1 : (~S0) ? I2 : I3 ;
endmodule

module mux8_2_1_mixed (I, S, O);
  input [7:0] I; // οι πηγές μας σχηματίζουν μια αρτηρία 8 σημάτων
  input [2:0] S; // τα 3 σήματα επιλογής της πηγής
  output      O;
  wire        F_S2_0, F_S2_1;

  mux4_2_1_behavioral cp0 (I[0], I[1], I[2], I[3], S[1], S[0], F_S2_0);
  mux4_2_1_behavioral cp1 (I[4], I[5], I[6], I[7], S[1], S[0], F_S2_1);
  assign O = S[2] ? F_S2_1 : F_S2_0;
endmodule
```

Λύση 3 : Διεπίπεδος ιεραρχικός κώδικας. Πολυπλέκτης 8 σε 1 δομικά με στοιχεία πολυπλέκτες 2 σε 1 μόνο.
(Για περισσότερη κατανόηση πως προέκυψε ο κώδικας χρειάζεται να προσπαθήσετε να υλοποιήσετε και το πολυπλέκτη 4 σε 1 από πολυπλέκτες 2 σε 1 εφαρμόζοντας το θεώρημα του Shannon)

```
module mux2_2_1_behavioral (I0, I1, S0, F);
  input I0, I1, S0;
  output F;

  assign F = (~S0) ? I0 : I1;
endmodule

module mux8_2_1_struct (I, S, O);
  input [7:0] I; // οι πηγές μας σχηματίζουν μια αρτηρία 8 σημάτων
  input [2:0] S; // τα 3 σήματα επιλογής της πηγής
  output      O;

// Οι παρακάτω 4 πολυπλέκτες ανάλογα με τη τιμή του S[0] αποφασίζουν αν θα επιλεγεί μια πηγή
// δεδομένων με άρτια ή περιττή τιμή
  mux2_2_1_behavioral cp0 (I[0], I[1], S[0], even_odd_0);
  mux2_2_1_behavioral cp1 (I[2], I[3], S[0], even_odd_1);
  mux2_2_1_behavioral cp2 (I[4], I[5], S[0], even_odd_2);
  mux2_2_1_behavioral cp3 (I[6], I[7], S[0], even_odd_3);

// Οι παρακάτω 2 πολυπλέκτες ανάλογα με τη τιμή του S[1] αποφασίζουν ποια από τις πηγές που
// προκρίθηκαν από το προηγούμενο στάδιο θα συνεχίσουν να είναι υποψήφιες. Όταν S[1] = 0
// θέλουμε κάποια από τις (0,1) και (4,5) ενώ όταν S[1] =1 κάποια από τις (2,3) ή (6,7). Εντός
// της δυάδας η επιλογή έχει γίνει στο προηγούμενο στάδιο.
  mux2_2_1_behavioral cp4 (even_odd_0, even_odd_1, S[1], can_0);
  mux2_2_1_behavioral cp5 (even_odd_2, even_odd_3, S[1], can_1);

// Η τελική επιλογή βάσει του S[2]
  mux2_2_1_behavioral cp6 (can_0, can_1, S[2], O);
endmodule
```

Λύση 4 : Τριεπίπεδος ιεραρχικός κώδικας. Πολυπλέκτης 8 σε 1 με δομικά στοιχεία πολυπλέκτες 4 σε 1 και 2 σε 1. Ο πολυπλέκτης 4 σε 1 φτιάχνεται από αποκωδικοποιητή και πύλες.

```
module decoder2_2_4 (S1, S0, O0, O1, O2, O3);
  input S1, S0;
  output O0, O1, O2, O3;

  assign O0 = ~S1 & ~S0;
  assign O1 = ~S1 & S0;
  assign O2 = S1 & ~S0;
  assign O3 = S1 & S0;
endmodule

module mux4_2_1_decoder_based (A, B, C, D, S, T, O);
  input A, B, C, D, S, T;
  output O;
  wire D0, D1, D2, D3;

  decoder2_2_4 i0 (S, T, D0, D1, D2, D3);
  and          i1 (P1, D0, A);
  and          i2 (P2, D1, B);
  and          i3 (P3, D2, C);
  and          i4 (P4, D3, D);
  or           i5 (O, P1, P2, P3, P4);
endmodule

module mux2_2_1_behavioral (I0, I1, S0, F);
  input I0, I1, S0;
  output F;

  assign F = (~S0) ? I0 : I1;
endmodule

module mux8_2_1_too_many_levels (I, S, O);
  input [7:0] I; // οι πηγές μας σχηματίζουν μια αρτηρία 8 σημάτων
  input [2:0] S; // τα 3 σήματα επιλογής της πηγής
  output      O;
  wire        F_S2_0, F_S2_1;

  mux4_2_1_decoder_based cp0 (I[0], I[1], I[2], I[3], S[1], S[0], F_S2_0);
  mux4_2_1_decoder_based cp1 (I[4], I[5], I[6], I[7], S[1], S[0], F_S2_1);

  mux2_2_1_behavioral cp2 (F_S2_0, F_S2_1, S[2], O);
endmodule
```

Πριν προχωρήσουμε στην εξομοίωση των παραπάνω κωδίκων, αξίζει να σταθούμε λίγο στην αποδοτικότητα της ύπαρξης ιεραρχίας σχεδιασμού. Γιατί δηλαδή να έχουμε ιεραρχικούς σχεδιασμούς και όχι μονοεπίπεδους ? Οι λόγοι για κάτι τέτοιο είναι πολλοί :

- Ο ιεραρχικός τρόπος συνάδει με τη κατάτμηση της πολυπλοκότητας των συγχρόνων συστημάτων. Όταν έχουμε να σχεδιάσουμε κάτι πολύ πολύπλοκο προσπαθούμε να το σπάσουμε σε μικρότερα κομμάτια μέχρις ότου η πολυπλοκότητα ενός κομματιού γίνει διαχειρίσιμη.
- Προσφέρει πολύ πιο απλές διαδικασίες εύρεσης και διόρθωσης λαθών. Αν δεν είχαμε ιεραρχικό σχεδιασμό θα αναγκαζόμασταν να επαναλάβουμε κομμάτια κώδικα. Λάθος σε αυτό το κομμάτι σημαίνει ότι θα πρέπει να ανιχνεύουμε πολλά πιθανά σημεία λαθών και να διορθώνουμε το κώδικα σε καθένα. Αντίθετα αρκεί να διορθώσουμε το κώδικα δήλωσης ενός module σε ένα ιεραρχικό σχεδιασμό και η διόρθωση αυτή θα ισχύσει για όλα τα αντίτυπά του που χρησιμοποιούμε.

Για την εξομοίωση της ορθότητας των λύσεων 1 έως 4, ας ακολουθήσουμε μια μεθοδολογία σύγκρισης των αποκρίσεων αυτών των κυκλωμάτων. Οσο οι αποκρίσεις συμφωνούν μεταξύ τους θεωρούμε ότι όλα τα κυκλώματα λειτουργούν σωστά. Σταματάμε την εξομοίωση όταν διαπιστωθεί ασυμφωνία. Στις πηγές δεδομένων εφαρμόζουμε κυματομορφές με διπλάσια, τετραπλάσια, οκταπλάσια, … περίοδο μιας βασικής. Κάθε 2000 χρονικές στιγμές διαλέγουμε και μια νέα πηγή για την έξοδό μας.

```
module test_them_all ();
  reg [7:0] DataSource;
  reg [2:0] Select;

  mux8_2_1_beh             CUT1 (DataSource, Select, O1);
  mux8_2_1_mixed           CUT2 (DataSource, Select, O2);
  mux8_2_1_struct          CUT3 (DataSource, Select, O3);
```

```
  mux8_2_1_too_many_levels CUT4 (DataSource, Select, O4);

  initial begin DataSource=0; Select=0; end
  always   #10 DataSource = DataSource + 1 ;
  always #2000 Select = Select + 1;
  always     #5 if ((O1!= O2) | (O1!= O3) | (O1!= O4) | (O2!= O3) | (O2!= O4) | (O3!= O4)) $stop();
endmodule
```

Όταν θα προσπαθήσετε να τρέξετε εξομοίωση (χρησιμοποιώντας το αρχείο 5_3.v), θα διαπιστώσετε ότι ο εξομοιωτής θα σταματήσει για πρώτη φορά τη χρονική στιγμή 2015, γιατί η έξοδος O4 διαφέρει από όλες τις υπόλοιπες. Παρότι στο παρόν κείμενο ο κώδικας που παραθέτουμε είναι απολύτως σωστός, στο συνοδευτικό αρχείο έχουμε εισάγει κάποιο λάθος. Σκοπός είναι όχι να συγκρίνετε αυτούς τους 2 κώδικες, αλλά χρησιμοποιώντας τον εξομοιωτή να βρείτε το λάθος, να το διορθώσετε και να βεβαιώσετε ότι πλέον όλα τα κυκλώματα λειτουργούν με τον ίδιο τρόπο. Παρακάτω δίνουμε μερικά βήματα που θα σας βοηθήσουν :

1. Αφού το πρόβλημα είναι στην O4 που παράγεται από το αντίγραφο CUT4, στο παράθυρο structure χτυπήστε το CUT4. Ετσι στο παράθυρο signals θα εμφανιστούν τα σήματα του CUT4. Προσθέστε τα στις κυματομορφές και ξανατρέξτε εξομοίωση μέχρι τη χρονική στιγμή 2015.
2. Παρατηρούμε ότι το CUT4 παίρνει σωστές τιμές στις εισόδους I και S. Ο τελευταίος πολυπλέκτης του (2 σε 1) λειτουργεί σωστά, δηλαδή αφού το S[2] είναι 0, καλώς κάνει και περνάει το F_S2_0 στην έξοδο. Αρα το πρόβλημα έγκειται στον υπολογισμό των F_S2_0 ή / και F_S2_1.
3. Αυτά παράγονται από τους 4 σε 1 πολυπλέκτες. Αρα στα επόμενα βήματα επικεντρωθείτε σε αυτούς, δείτε τι εισόδους παίρνουν, εντοπίστε τυχόν λάθη και διορθώστε τα.

ΠΑΡΑΔΟΤΕΟ 3 : ***(Χρησιμοποιείστε το αρχείο 5_3.v)***
α) Μόνο οι γραμμές κώδικα που αλλάξατε.
β) Οι κυματομορφές εξομοίωσης του διορθωμένου κώδικα κατά τις χρονικές στιγμές 2000 - 7000.