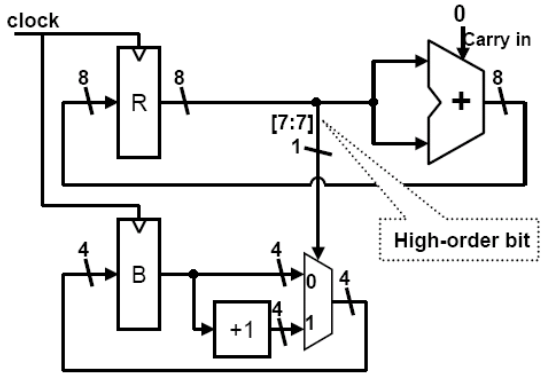
Θέμα 1 **(2.5 μονάδες)**



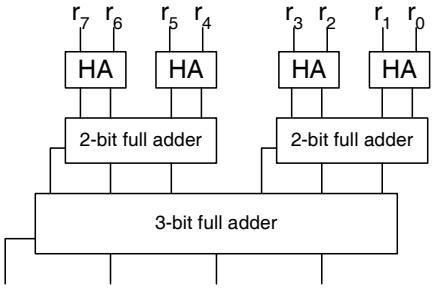
Το ακολουθιακό κύκλωμα του παραπάνω σχήματος εκτελεί ένα συγκεκριμένο υπολογισμό. Οι R και B είναι καταχωρητές των 8 και 4 δυαδικών ψηφίων αντίστοιχα. Για το ζητούμενο υπολογισμό, ο καταχωρητής αρχικά B μηδενίζεται ενώ στον R φορτώνεται μια αρχική τιμή. Το κομμάτι του σχεδιασμού που σημειώνεται σαν "High-order bit" υποδεικνύει ότι το κάθετο καλώδιο συνδέεται μόνο στο πιο σημαντικό ψηφίο της αρτηρίας εξόδου του καταχωρητή R. Το κύκλωμα επίσης απαρτίζεται από έναν προσθετή των 8 δυαδικών ψηφίων με κρατούμενο εισόδου πάντοτε 0, έναν τετραπλό πολυπλέκτη 2 σε ένα και έναν αυξητή (προσθετή δηλαδή της μονάδας) των 4 δυαδικών ψηφίων.

- Με μία πρόταση καθορίστε ποια είναι η σχέση μεταξύ της αρχικής τιμής του R και του περιεχομένου του B μετά από 8 κύκλους ρολογιού ? (0.5)

Εστω ότι αρχικά ο R περιέχει το α. Ο αθροιστής εκτελεί τη πρόσθεση α+α με κρατούμενο εισόδου 0, άρα στον επόμενο χρονικό κύκλο ο R θα περιέχει το 2α (τυχόν κρατούμενο εξόδου αγνοείται), δηλαδή το α αριστερά ολισθημένο. Αρα κατά τους 8 παλμούς, θα εμφανιστούν ένα – ένα όλα τα δυαδικά ψηφία του α, στη γραμμή επιλογής του πολυπλέκτη. Σε κάθε χρονικό κύκλο που το ψηφίο επιλογής του πολυπλέκτη είναι 1, το περιεχόμενο του B αυξάνεται κατά 1. Συνεπώς στο τέλος των 8 κύκλων στον B θα υπάρχει ο αριθμός των "1" της δυαδικής τιμής του α. Στον R θα υπάρχει 0.

- Ποιο συνδυαστικό κύκλωμα θα μπορούσατε εναλλακτικά να χρησιμοποιήσετε στην έξοδο του R για τον ίδιο σκοπό ? (προφανώς αυτό βγάζει το ίδιο αποτέλεσμα σε ένα και όχι σε 8 κύκλους) (0.5)

Απαιτείται ένα δέντρο από ημιαθροιστές / πλήρεις αθροιστές όπως αυτό του παρακάτω σχήματος :



- Τι μεγέθους EPROM θα χρειαζόσασταν αντί του συνδυαστικού κυκλώματος του προηγουμένου ερωτήματος ? Ποια θα ήταν τα περιεχόμενα της EPROM στη θέση με διεύθυνση 00101111_2 ? (0.5)

Μέγεθος = 2^8 θέσεις x 4 bits / θέση = 1Kbit EPROM.
Στη θέση με 00101111_2 θα πρέπει να είναι αποθηκευμένο το 0101_2.

- Δώστε τις περιγραφές σε Verilog για το κύκλωμα του σχήματος και του εναλλακτικού κυκλώματος του 2ου υποερωτήματος, συμπεριλαμβάνοντας και στις δύο περιπτώσεις τα σήματα αρχικοποίησης των καταχωρητών, το σήμα παράλληλης φόρτωσης και τις εισόδους παράλληλων δεδομένων. (1.0)

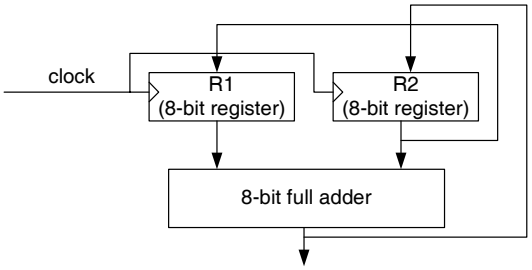
```
module sequential (clk, load, data);
  input clk, load;
  input [7:0] data;
  reg   [7:0] r;
  reg   [3:0] b;
  wire  [3:0] muxout;

  assign muxout = r[7]? b+1 : b;
  always @(posedge clk)
    if (load) begin r<=data; b<=4'h0; end
      else begin r<=r+r; b<=muxout; end
endmodule
```

```
module combinational (clk, load, data, onescount);
  input        clk, load;
  input  [7:0] data;
  reg    [7:0] r;
  output [3:0] onescount;

  always @(posedge clk)
    if (load) begin r<=data; end
  assign onescount = r[7] + r[6] + r[5] + r[4] +
                     r[3] + r[2] + r[1] + r[0];
endmodule
```

Θέμα 2 **(2.5 μονάδες)**

Η ακολουθία Fibonacci είναι μια αριθμητική σειρά, στην οποία κάθε όρος προκύπτει από το άθροισμα των δύο προηγούμενων ενώ οι δύο πρώτοι όροι είναι 0 και 1. Συνεπώς οι πρώτοι όροι της ακολουθίας είναι 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...

- Δώστε το διάγραμμα ενός ακολουθιακού κυκλώματος το οποίο μετά την αρχικοποίησή του θα μας δίνει στην έξοδό του με κάθε κύκλο ρολογιού τον επόμενο όρο της ακολουθίας Fibonacci ξεκινώντας από τον 3ο. Ο τελευταίος όρος που θα χρειάζεται να παράγεται είναι ο 14ος, δηλαδή το 233. (*Υπόδειξη : ΜΗΝ προσπαθήσετε να φτιάξετε ένα FSM*).



Αφού κάθε όρος προκύπτει σαν το άθροισμα των δύο προηγούμενων, συμπεραίνουμε ότι θα χρειαστούμε 2 καταχωρητές έστω R1, R2 οι οποίοι θα αποθηκεύουν τους 2 πιο πρόσφατους όρους της ακολουθίας. Οι καταχωρητές θα πρέπει να μπορούν να αποθηκεύουν έως και το 233, άρα θα πρέπει να είναι των 8 bit. Σε κάθε χρονικό κύκλο, τα περιεχόμενά τους αθροίζονται. Το άθροισμα καθώς και ο προηγούμενος μέγιστος όρος της ακολουθίας θα πρέπει να αποθηκευτούν και πάλι στους δύο καταχωρητές. Αναθέτουμε συνεπώς στον R2 να διατηρεί το αποτέλεσμα και καθώς πριν το παλμό ρολογιού διατηρούσε το προηγούμενο μέγιστο όρο (το άθροισμα που προέκυψε ένα κύκλο πριν) αντιγράφουμε τα περιεχόμενά του στον R1. Κατά την αρχικοποίηση προφανώς ο R1 θα αρχικοποιηθεί με 0 και ο R2 με 1.

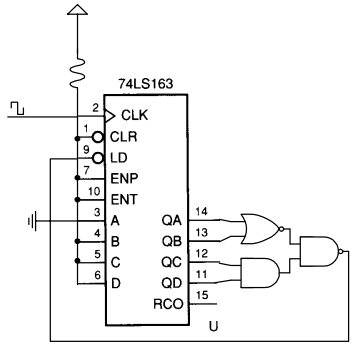
♦ Περιγράψτε σε Verilog το κύκλωμα που προτείνετε. (1.0 μονάδα)

```
module Fib_generator (start, fibterm, clk);
  input        start, clk;
  output [7:0] fibterm;
  reg    [7:0] R1, R2;
  assign fibterm=R1+R2;
  always @(posedge clk)
    if (start) begin R1<=8'h0; R2<=8'h1; end
      else begin R1<=R2; R2<=fibterm; end
endmodule
```

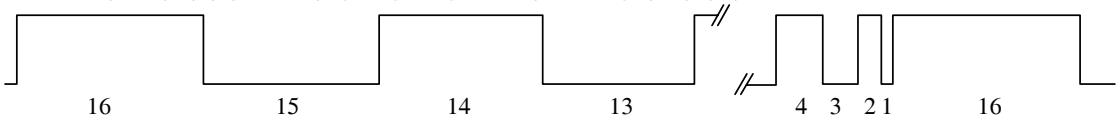
Θέμα 3 **(2.0 μονάδες)**

♦ Ενας "προληπτικός" μετρητής είναι ένας μετρητής ο οποίος υπερπηδά την τιμή 13. Δώστε το λογικό διάγραμμα ενός προληπτικού μετρητή, χρησιμοποιώντας τον 74LS163 τετράμπιτο μετρητή της παρακάτω εικόνας και όσες πύλες *δύο* εισόδων κρίνετε απαραίτητες. (0.7)

Σύμφωνα με τις διαφάνειες του μαθήματος ο 163 είναι ένας 4-bit μετρητής με σύγχρονο load και clear (active low σήματα) με προτεραιότητα στο clear. Οταν και τα δύο αυτά σήματα είναι ανενεργά, και ENP = ENT =1, ο μετρητής μετράει προς τα πάνω. Για να υπερπηδήσουμε τη κατάσταση 13, θα πρέπει όταν δούμε το 12 στην έξοδό του να ενεργοποιήσουμε τη γραμμή φόρτωσης και να θέσουμε τα δεδομένα εισόδου στο 14, ώστε στον επόμενο κύκλο ο μετρητής να φορτωθεί με το 14. Αρα το ζητούμενο σχήμα είναι :



♦ Η παρακάτω κυματομορφή είναι η έξοδος ενός "αποσβένοντος" μετρητή.

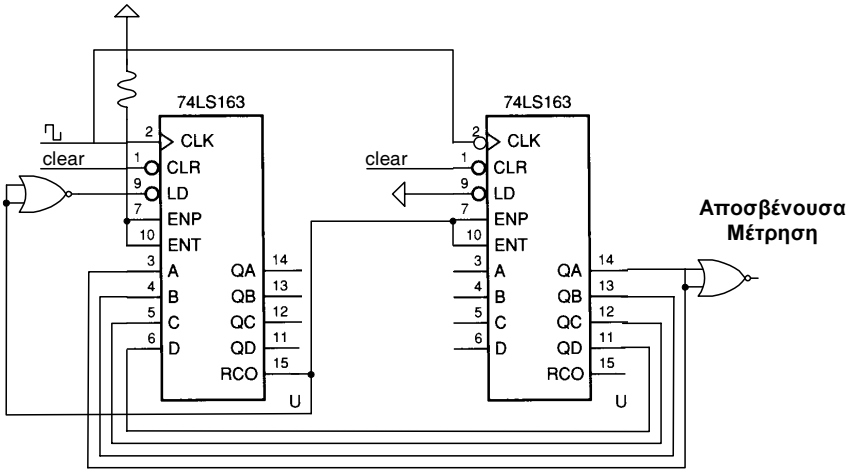


δηλαδή, η έξοδος είναι στο 1 για 16 κύκλους, στο 0 για 15 κύκλους, στο 1 για 14 κύκλους κοκ, μέχρι που είναι στο 0 για 1 κύκλο. Η ακολουθία αμέσως μετά επαναλαμβάνεται. Η περίοδος δηλαδή του μετρητή είναι : 16 + 15 + ... + 2 + 1 = 136 κύκλοι ρολογιού.

Δώστε το λογικό διάγραμμα ενός αποσβένοντος μετρητή χρησιμοποιώντας τους δύο 74LS163 μετρητές και όση επιπλέον λογική κρίνετε απαραίτητη. Το διάγραμμά σας θα πρέπει να δείχνει καθαρά τις συνδέσεις μεταξύ των μετρητών και της υπόλοιπης λογικής. (1.3)

Ενας μετρητής των 4 bit, προφανώς δε μπορεί να μας δώσει κανένα σήμα σταθερό για 16 ωρολογιακούς παλμούς. Συνεπώς χρειάζεται να συνδέσουμε τους δύο μετρητές ώστε να φτιάξουμε έναν μετρητή των 8 bit. Αυτό μπορεί να γίνει χρησιμοποιώντας ως επίτρεψη για το $2^{ο}$ το RCO του πρώτου (βλέπε διαφάνειες). Κατ' αυτόν το τρόπο ο δεύτερος μετρητής (high order nibble) μετράει κάθε φορά που ο πρώτος έχοντας φτάσει στο 15, επανέρχεται στο 0. Το $5^{ο}$ bit του 8-bit μετρητή έχει ημιπερίοδο 16 παλμών, δηλαδή είναι 16 παλμούς στο 0, 16 στο 1 κ.ο.κ. Αρα αντεστραμμένο, μας παρέχει το πρώτο κομμάτι της αποσβένουσας μέτρησης με τη ζητούμενη πολικότητα. Αφού θέλουμε η διάρκεια των παλμών να μειώνεται, θα πρέπει να ενεργοποιούμε το $2^{ο}$ μετρητή όλο και πιο συχνά ή ισοδύναμα, να φορτώνουμε σε αυτόν όλο και μεγαλύτερες τιμές. Αφού ο $2^{ος}$ αυξάνεται κάθε φορά που ο πρώτος φτάνεις στο 15, συμπεραίνουμε ότι μπορούμε να φορτώνουμε τον πρώτο με τη τιμή του $2^{ου}$ χρησιμοποιώντας το σήμα RCO αντεστραμμένο. Ετσι ο πρώτος μετρητής θα μετράει από 0 έως 15, από 1 έως 15, από 2 έως 15 κ.ο.κ. παρέχοντάς μας το σήμα ενεργοποίησης για το δεύτερο με μικρότερη περίοδο κάθε φορά. Τέλος, αν και οι δύο μετρητές έχουν το ίδιο ρολόι η επιθυμητή φόρτωση του

πρώτου γίνεται ένα κύκλο μετά, άρα θα πάρουμε την ακολουθία 17 κύκλοι στο 1, 16 στο 0 κ.ο.κ. Αποφυγή αυτού μπορεί να γίνει χρησιμοποιώντας συμπληρωματικά ρολόγια. Αρα το ζητούμενο σχήμα είναι :

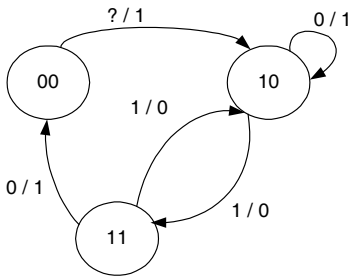


Μπορείτε να επιβεβαιώσετε την ορθότητα του παραπάνω σχήματος, χρησιμοποιώντας τον ακόλουθο κώδικα :

```
module c74163like (clk, clear_n, load_n, enp, ent, data_in, count, rco);
  input        clk, load_n, clear_n, enp, ent;
  input  [3:0] data_in;
  output [3:0] count;
  reg    [3:0] count;
  output       rco;

  always @(posedge clk)
    if (~clear_n) count <= 4'h0;
      else if (~load_n) count <= data_in;
             else if (enp & ent) count <= count + 1 ;
  assign rco = (&count &  ent)? 1'b1 : 1'b0;
endmodule

module chirp (clk, clr, chir);
  input      clk, clr;
  output     chir;
  wire [3:0] val;
  wire       temp;
  assign chir = ~val[0];
  c74163like c0 ( clk, clr, ~temp,  1'b1,  1'b1,  val,    , temp);
  c74163like c1 (~clk, clr,  1'b1,  temp,  temp,     , val,     );
endmodule

module testchirp();
  reg  clk, clr;
  wire chir;
  chirp CUT (clk, clr, chir);
  initial
    begin
          clk = 0;
          clr = 1;
      #25 clr = 0;
      #40 clr = 1;
    end
  always #15 clk = ~clk;
endmodule
```

Θέμα 4 **(3.0 μονάδες)**

Στο παρακάτω σχήμα φαίνεται το διάγραμμα μετάβασης καταστάσεων ενός ακολουθιακού κυκλώματος με 2 *αρνητικά* ακμοπυροδότητα flip – flops, έστω A, B. Κάποιος σχεδιαστής, αποφάσισε να υλοποιήσει το κύκλωμα χρησιμοποιώντας D και JK τύπου φλιπ – φλοπς για τα A και B αντίστοιχα.



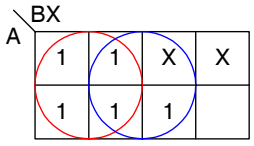
♦ Θεωρώντας ότι ο σχεδιαστής ακολούθησε στρατηγική ελάχιστου κόστους δείξτε τη διαδικασία σχεδιασμού που ακολούθησε και το λογικό κύκλωμα στο οποίο κατέληξε. Εξηγείστε αν το κύκλωμα έχει την ικανότητα "αυτόματης εκκίνησης" (αυτοδιόρθωσης).

(2.0)

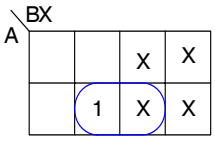
Από το δριάγραμμα καταστάσεων προκύπτει ότι έχουμε 3 καταστάσεις, άρα θα χρειαστούμε 2 ff, έστω A και B, με το A τύπου D και το B τύπου JK. Επίσης έχουμε 1 είσοδο και μία έξοδο. Καταστρώνουμε το πίνακα μετάβασης καταστάσεων και εξόδου καθώς και επόμενης εισόδου των ff. Η εναπομένουσα κατάσταση 01, για στρατηγική ελάχιστου κόστους, είναι αδιάφορη.

Τρέχουσα Κατάσταση A(t) B(t)	Είσοδος X	Επόμενη Κατάσταση A(t+1) B(t+1)	DA(t+1)	JB(t+1) KB(t+1)	Εξοδος Y
0 0	0	1 0	1	0 X	1
0 0	1	1 0	1	0 X	1
0 1	0	X X	X	X X	X
0 1	1	X X	X	X X	X
1 0	0	1 0	1	0 X	1
1 0	1	1 1	1	1 X	0
1 1	0	0 0	0	X 1	1
1 1	1	1 0	1	X 1	0

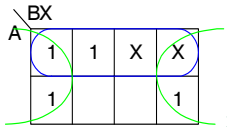
Προφανώς είναι KB(t+1) = 1, ενώ για τις υπόλοιπες συναρτήσεις έχουμε τους εξής πίνακες αληθείας :



και DA(t+1) = ~B|X,

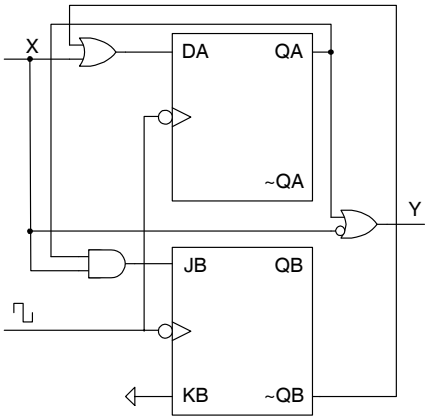


και JB(t+1) =A&X,



και Y=A|~X.

Το κύκλωμα στο οποίο κατέληξε ο σχεδιαστής είναι συνεπώς το :



Αφού KB = 1, ανάλογα με τη τιμή του JB, το B ff είτε καθαρίζεται είτε αντιστρέφεται. Αν το κύκλωμα βρεθεί στην μη αναμενόμενη κατάσταση AB=01, η επόμενη κατάσταση θα είναι συνεπώς AB=X0. Ομως και οι δύο καταστάσεις AB=00 και AB=10 είναι στο σύνολο των μοντελοποιημένων καταστάσεων. Αρα το κύκλωμα έχει αυτόματη εκκίνηση.

♦ Περιγράψτε σε Verilog το δεδομένο Mealy FSM. (1.0)

```
module Mealy_fsm (clk, X, Y);
  input clk, X;
  output Y;
  reg Y;
  reg [1:0] current, next;

  parameter S0=0, S2 = 2, S3 = 3;

  always @(negedge clk) current<= next;
  always @(current or X)
    case (current)
      S0      : begin next = S2; Y=1; end
      S2      : begin next = (X) ? S3 : S2; Y = (X)? 0: 1; end
      S3      : begin next = (X) ? S2 : S0; Y = (X)? 0: 1; end
      default : begin next=2'bX; Y=1'bX; end
    endcase
endmodule
```