

# Γλώσσες Περιγραφής Υλικού I

## Μοντελοποίηση ακολουθιακών κυκλωμάτων

Νικόλαος Καββαδίας  
nkavn@uop.gr

13 Μαρτίου 2012

# Σκιαγράφηση της διάλεξης

- Στοιχεία ακολουθιακής σχεδίασης με Verilog HDL
  - Λίστα ευαισθησίας και ακμοπυροδότηση
  - Flip-flop και μανδαλωτές
  - Blocking και nonblocking αναθέσεις σε μπλοκ always
- Παραδείγματα σχεδίασης ακολουθιακών κυκλωμάτων
  - Το flip-flop τύπου D
  - Ο D μανδαλωτής
  - flip-flop τύπου JK και T
  - Καταχωρητής
  - Απαριθμητής πάνω/κάτω (up-down counter)
  - Καταχωρητής ολίσθησης
  - Απαριθμητής για το σύστημα BCD (Binary-Coded Decimal)

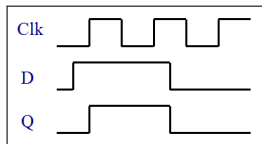
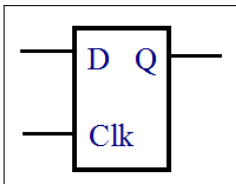
# Διαφορές μεταξύ των περιγραφών συνδυαστικών και ακολουθιακών κυκλωμάτων

- Συνδυαστικά κυκλώματα
  - Δεν ενεργοποιούνται από ακμοπυροδότηση εξωτερικού σήματος αλλά μόνο από τις αλλαγές στη στάθμη των σημάτων εισόδου
  - Η λειτουργία τους δεν επηρεάζεται από το σήμα ρολογιού
- Ακολουθιακά κυκλώματα
  - Ενεργοποιούνται κατά τη θετική ή αρνητική ακμοπυροδότηση του ρολογιού
  - Στη λίστα ευαισθησίας εμφανίζεται μόνο το σήμα ρολογιού και πιθανώς και ένα σήμα επανατοποθέτησης (reset)
  - Μπορεί να περιλαμβάνουν συνδυαστική λογική η οποία παράγει αποτελέσματα για εγγραφή σε καταχωρητές (flip-flop)

# Βασικά στοιχεία ακολουθιακής λογικής: Μανδαλωτές και καταχωρητές (flip-flop)

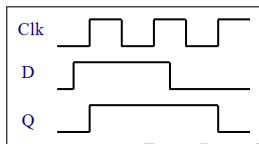
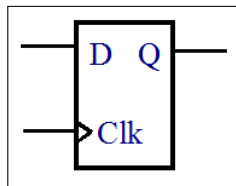
Μανδαλωτής (latch): απόκριση με ευαισθησία επιπέδου (level-sensitive)

Αποθηκεύει δεδομένα π.χ. όταν το σήμα *clk* είναι σε υψηλή στάθμη



Καταχωρητής (register): ακμοπυροδότητο στοιχείο (edge-triggered)


Αποθηκεύει δεδομένα π.χ. κατά την ανερχόμενη ακμή του ρολογιού



# Σήματα ρολογιού και flip-flop

- Τα σήματα ρολογιού δεν είναι τυπικά σήματα εισόδου σε ένα υποκύκλωμα
- Διαδίδονται μέσα από το δίκτυο διάδοσης ρολογιού (clock distribution network)
- Χρειάζονται ειδική καλωδίωση για τη διάδοσή τους σε απομακρυσμένα σημεία του ολοκληρωμένου, διορθώνοντας τυχόν ανεπιθύμητες χρονικές καθυστερήσεις
- ❗ Δεν χρησιμοποιούμε τα σήματα ρολογιού σε λογικές εκφράσεις
- Αντί αυτού προσθέτουμε μία είσοδο ενεργοποίησης ρολογιού (clock enable) στο flip-flop
- Από τους γνωστούς τύπους flip-flop (RS, JK, T, D) στις περισσότερες σχεδιάσεις χρησιμοποιείται ο τύπος D
- Σε ορισμένες σχεδιάσεις εξειδικευμένων μετρητών χρησιμοποιούνται τα flip-flop JK και T

# Λίστα ευαισθησίας (sensitivity list)

- Η λίστα ευαισθησίας αποτελεί κατάλογο εισόδων και εσωτερικών σημάτων για μεταβολές των οποίων ένα μπλοκ `always` υποχρεούται να αναμένει
- Αν έχουμε περισσότερα από ένα σήματα στη λίστα, τα διαχωρίζουμε με τη λέξη-κλειδί `or`
-  Αν ένα σήμα στη λίστα χρησιμοποιεί ακμοπυροδότηση, τότε όλα τα υπόλοιπα πρέπει να δηλωθούν και αυτά με ακμοπυροδότηση
- Παράδειγμα 1:

```
always @(a or b)
begin
  if (a != b)
    cond = 1'b1;
  else
    cond = 1'b0;
end
```

- Παράδειγμα 2: βρείτε το λάθος

```
always @(a)
if (a == 1'b1)
  temp = ~temp;
```

# Ευαισθησία επιπέδου (level sensitivity) και ακμοπυροδότηση (edge triggering)

- Οι μεταβολές των σημάτων που δηλώνονται σε μία λίστα ευαισθησίας και οι οποίες ενεργοποιούν τον υπολογισμό μεταβλητών και σημάτων σε μία `always` είναι δύο τύπων:
  - Μεταβολή επιπέδου (για σήματα επίτρεψης/ενεργοποίησης και δεδομένα)
  - Ανερχόμενη ή κατερχόμενη ακμή (για σήματα ρολογιού)

## ■ Μανδαλωτής (μεταβολή επιπέδου)

```
always @(en or a)
begin
  if (en == 1'b1)
    temp = a;
end
```

## ■ Συγχρονισμός ως προς ανερχόμενη ακμή

```
always @(posedge clk)
begin
  temp = a;
end
// or
always @(posedge clk)
temp = a;
```

- ☞ Οι εκφράσεις `posedge` και `negedge` είναι ΑΛΗΘΕΙΕΣ όταν έχει συμβεί μεταβολή ( $0 \rightarrow 1$  ή  $1 \rightarrow 0$ , αντίστοιχα) στο σήμα `clk`
- ℹ Για τα περισσότερα εργαλεία σύνθεσης, τα σήματα εκτός του `clk` και του `reset` (αν υπάρχει) συνηθίζεται να παραλείπονται από μια λίστα ευαισθησίας (για ακολουθιακές `always`)

# Εντολή WAIT

- Χρησιμοποιείται για τη δήλωση ευαισθησίας επιπέδου
- Όταν χρησιμοποιείται η WAIT, δεν γίνεται χρήση λίστας ευαισθησίας
- Σύνταξη της WAIT:

```
wait (<signal_name>) statement;
```

- Παράδειγμα

```
always  
wait (count_enable) #20 count = count + 1;
```

- Όταν το count\_enable γίνει 1, η δήλωση count = count + 1; εκτελείται μετά από 20 time units
- Καθόσον η count\_enable παραμένει στο 1, η count αυξάνεται κατά 1 κάθε 20 t.u.



# Διαφορά μεταξύ blocking και nonblocking αναθέσεων σε μία always (1)

- Οι αναθέσεις με τον τελεστή = ονομάζονται blocking και εκτελούνται ακολουθιακά
- Εάν είναι επιθυμητό εντός μίας always να επιτρέψουμε την παράλληλη εκτέλεση κάποιων δηλώσεων, χρησιμοποιούμε τον τελεστή <= ο οποίος δηλώνει μία nonblocking ανάθεση
- Παράδειγμα: Τρεις παράλληλες αναθέσεις κατά τη θετική ακμή του ρολογιού

```
always @(posedge clock)
begin
    reg1 <= in1;
    reg2 <= in2 ^ in3;
    reg3 <= reg1;      // Old value of reg1
end
```

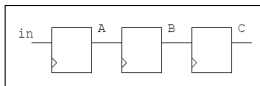
- ☞ Οι τελικές τιμές των reg1, reg2, reg3 δεν εξαρτώνται από τη σχετική σειρά των αναθέσεων

# Διαφορά μεταξύ blocking και nonblocking αναθέσεων σε μία always (2)

## ■ Nonblocking assignment

```
always @(posedge clk) begin
  a <= in;
  b <= a;
  c <= b;
end
```

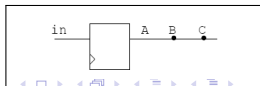
- Όλες οι αναθέσεις εκτελούνται ακριβώς την ίδια χρονική στιγμή
- Σε κάθε ανερχόμενη ακμή του ρολογιού, το a παίρνει την ΠΑΛΙΑ τιμή του in, το b αυτήν του a και το c αυτήν του b
- Θεωρούμε ότι τοποθετείται ένας καταχωρητής μετά από κάθε δήλωση





## ■ Blocking assignment

```
always @(posedge clk) begin
  a = in;
  b = a;
  c = b;
end
```

- Άμεσος υπολογισμός και ανάθεση των δεξιών μελών
- Όλες οι αναθέσεις γίνονται διαδοχικά στην ίδια περίοδο ρολογιού
- Θεωρούμε ότι αντιστοιχεί σε έναν καταχωρητή με πολλές ίδιες εξόδους ή σε πολλούς παράλληλους καταχωρητές



# Διαφορά μεταξύ blocking και nonblocking αναθέσεων σε μία always (3)

- Οι blocking αναθέσεις μπορεί να δημιουργήσουν προβλήματα εάν ο κώδικας περιλαμβάνει πολλαπλά μπλοκ λογικής always
- Με τις nonblocking αναθέσεις υποδηλώνονται εσωτερικά στάδια διοχέτευσης σε μία always και έτσι είναι χρήσιμες π.χ. για την περιγραφή καταχωρητών ολίσθησης
-  Οι blocking αναθέσεις χρησιμοποιούνται για τη μοντελοποίηση συνδυαστικών μπλοκ always
-  Οι nonblocking αναθέσεις χρησιμοποιούνται για τη μοντελοποίηση ακολουθιακών μπλοκ always

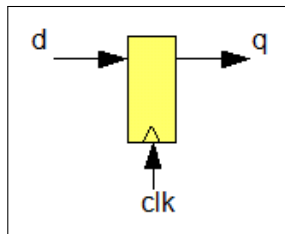
# Απλά ακολουθιακά κυκλώματα: Flip-flop τύπου D

■ Χαρακτηριστική εξίσωση:  $Q(t + 1) = D$

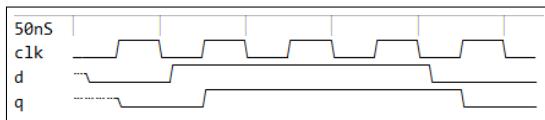
■ Περιγραφή D-type flip-flop του 1-bit

```
module dff(clk, d, q);  
  input clk;  
  input d;  
  output q;  
  reg q;  
  
  always @(posedge clk)  
  begin  
    q <= d;  
  end  
endmodule
```

■ Σχηματικό διάγραμμα



■ Διάγραμμα χρονισμού του κυκλώματος



# Αρχείο ελέγχου/επαλήθευσης (testbench) για το D flip-flop

```
'timescale 1 ns / 1 ns
module main;
  reg clk, d;
  wire q;

  dff test(.clk(clk), .d(d), .q(q));

  // Test clock generation.
  //
  // Clock pulse starts from 0.
  initial
    clk = 1'b0;

  // Free-running clock
  always
    #25 clk = ~clk;

  initial
  begin
    #10 d = 1'b0;
    #50 d = 1'b1;
    #150 d = 1'b0;
    #50
    $stop;
  end

  initial
    $monitor("%t: d=%b, q=%b", $time, d, q);
endmodule
```

# Καταχωρητής με ασύγχρονη και σύγχρονη επαναφορά (reset)

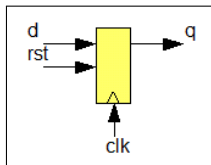
## Ασύγχρονη επαναφορά

```
module dff(clk, rst, d, q);  
  input clk;  
  input rst;  
  input d;  
  output q;  
  reg q;  
  
  always @(posedge clk or negedge rst)  
  begin  
    if (rst == 1'b1)  
      q <= 1'b0;  
    else  
      q <= d;  
    end  
endmodule
```

## Σύγχρονη επαναφορά

```
module dff(clk, rst, d, q);  
  input clk;  
  input rst;  
  input d;  
  output q;  
  reg temp;  
  
  always @(posedge clk)  
  begin  
    if (rst == 1'b1)  
      temp <= 1'b0;  
    else  
      temp <= d;  
    end  
  assign q = temp;  
endmodule
```

## Σχηματικό διάγραμμα

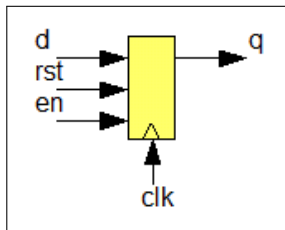


# Καταχωρητής με επίτρεψη φόρτωσης (load enable)

Με επίτρεψη φόρτωσης

```
module dff(clk, rst, en, d, q);
  input clk, rst, en;
  output q;
  reg temp;
  always @(posedge clk)
  begin
    if (rst == 1'b1)
      temp <= 1'b0;
    else
      begin
        if (en == 1'b1)
          temp <= d;
      end
    end
  assign q = temp;
endmodule
```

Σχηματικό διάγραμμα



## Μανδαλωτής (latch) τύπου D

- Πολύ συχνά σε μία ψηφιακή σχεδίαση εισάγονται ανεπιθύμητοι μανδαλωτές από κάποιο σφάλμα του σχεδιαστή (π.χ. μη χρήση ακμοπυροδότησης σε ένα D flip-flop ή λόγω υπονοούμενης αποθήκευσης)
- Υπονοούμενη αποθήκευση δημιουργείται όταν δεν δηλώνουμε όλες τις περιπτώσεις μίας IF ή CASE σε συνδυαστικό κώδικα
- Περιγραφή D-type latch του 1-bit

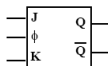
```
module dlatch(guard, d, q);  
  input guard;  
  input d;  
  output q;  
  reg q;  
  always @(guard or d)  
  begin  
    if (guard == 1)  
      q <= d;  
  end  
endmodule
```



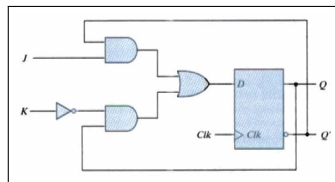
# Flip-flop τύπου JK

- Με τη βοήθεια δύο ξεχωριστών εισόδων (J, K), το JK flip-flop υλοποιεί τρεις βασικές λειτουργίες:
  - Τοποθέτηση της κατάστασης στο 1
  - Επανατοποθέτηση στο 0
  - Διατήρηση της τρέχουσας κατάστασης
  - Αντιστροφή της τρέχουσας κατάστασης
- Χαρακτηριστική εξίσωση:  $Q(t + 1) = JQ' + K'Q$

## ■ Πίνακας διέγερσης του JK-ff

	<table border="1"><thead><tr><th><math>J_n</math></th><th><math>K_n</math></th><th><math>Q_{n+1}</math></th></tr></thead><tbody><tr><td>0</td><td>0</td><td><math>Q_n</math></td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td><math>\overline{Q}_n</math></td></tr></tbody></table>	$J_n$	$K_n$	$Q_{n+1}$	0	0	$Q_n$	0	1	0	1	0	1	1	1	$\overline{Q}_n$
$J_n$	$K_n$	$Q_{n+1}$														
0	0	$Q_n$														
0	1	0														
1	0	1														
1	1	$\overline{Q}_n$														

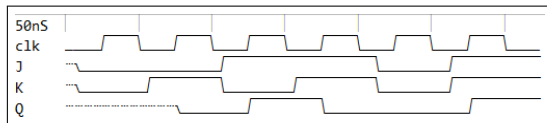
## ■ Υλοποίηση του JK-ff με βάση το D-ff



# Μια γενική περιγραφή του JK flip-flop

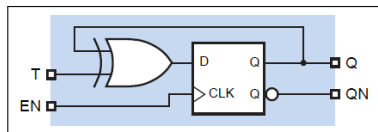
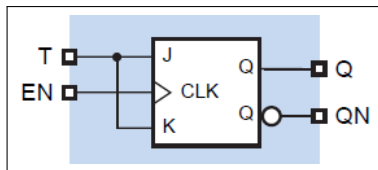
```
module jkff(J, K, clk, Q, Q_b);  
  input J, K, clk;  
  output Q, Q_b;  
  reg Q;  
  
  always @(posedge clk)  
    case ({J, K})  
      2'b00: Q <= Q;  
      2'b01: Q <= 1'b0;  
      2'b10: Q <= 1'b1;  
      2'b11: Q <= ~Q;  
    endcase  
  
  assign Q_b = ~Q;  
  
endmodule
```

Διάγραμμα χρονισμού του κυκλώματος



# Flip-flop τύπου T

- Το T flip-flop μεταβάλλει την κατάσταση του κάθε φορά που αλλάζει το σήμα στην είσοδο T
- Χαρακτηριστική εξίσωση:  $Q(t + 1) = Q \oplus T$
- Υλοποίηση με βάση το JK flip-flop
- Υλοποίηση με βάση το D flip-flop



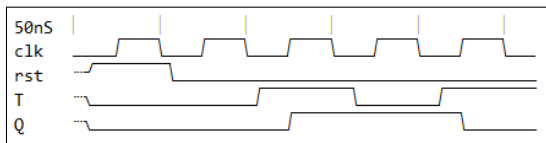
- Τα JK και T flip-flop χρησιμοποιούνται ορισμένες φορές σε σχεδιάσεις μετρητών για μη-δυναδικές ακολουθίες

# Μία γενική περιγραφή του T flip-flop

## ■ Περιγραφή T flip-flop

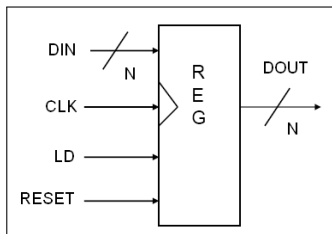
```
module tff (Q, T, clk, rst);  
    output Q;  
    input T, clk, rst;  
    reg Q;  
  
    always @(posedge clk)  
    begin  
        if (rst == 1'b1)  
            Q <= 1'b0;  
        else if (T == 1'b1)  
            Q <= ~Q;  
    end  
endmodule
```

## ■ Διάγραμμα χρονισμού του κυκλώματος



# Καταχωρητής (register)

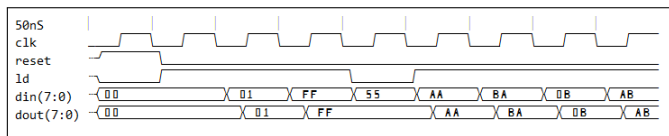
- Ο καταχωρητής αποτελεί το πιο συχνά χρησιμοποιούμενο δομικό στοιχείο σε μία ψηφιακή σχεδίαση στο επίπεδο RTL
- Ως εσωτερικό στοιχείο αποθήκευσης, συνήθως χρησιμοποιείται το D flip-flop
- Η είσοδος DIN προσφέρει είσοδο για παράλληλη αποθήκευση νέων δεδομένων στον καταχωρητή
- Τα περιεχόμενα του καταχωρητή αλλάζουν μόνο στην περίπτωση που  $LD = 1$  σε ανερχόμενη ακμή του ρολογιού



# Περιγραφή ενός 8-bit καταχωρητή στη Verilog (ασύγχρονο reset)

```
module register8 (clk, reset, ld, din, dout);  
  input      clk, reset, ld;  
  input [7:0] din;  
  output [7:0] dout;  
  reg [7:0]  dout;  
  
  // Use of asynchronous reset  
  always @(posedge clk or posedge reset)  
  begin  
    if (reset == 1'b1)  
      dout <= 0;  
    // Change register state on rising edge and assertion of load line  
    else if (ld == 1'b1)  
      dout <= din;  
  end  
  
endmodule
```

Διάγραμμα χρονισμού



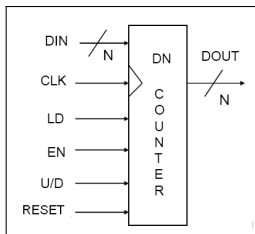
## Περιγραφή ενός 8-bit καταχωρητή στη Verilog (σύγχρονο reset)

```
module register8 (clk, reset, ld, din, dout);
    input        clk, reset, ld;
    input [7:0]  din;
    output [7:0] dout;
    reg [7:0]    dout;

    // Use of synchronous reset
    always @(posedge clk)
    begin
        if (reset == 1'b1)
            dout <= 0;
        // Change register state on rising edge and
        // assertion of load line
        else if (ld == 1'b1)
            dout <= din;
    end
endmodule
```

# Απαριθμητής πάνω/κάτω (up-down counter)

- Η σχεδίαση ενός απαριθμητή βασίζεται στο μοντέλο του καταχωρητή με την προσθήκη ενός αθροιστή/αφαιρέτη
- Η είσοδος U\_D καθορίζει τον τρόπο μεταβολής της κατάστασης του απαριθμητή
- Η λειτουργία του απαριθμητή είναι ενεργή μόνο για  $EN = 1$
- Η είσοδος DIN προσφέρει είσοδο για παράλληλη αποθήκευση νέων δεδομένων στον εσωτερικό καταχωρητή
- Η επίτρεψη αποθήκευσης (LD) έχει προτεραιότητα έναντι της EN

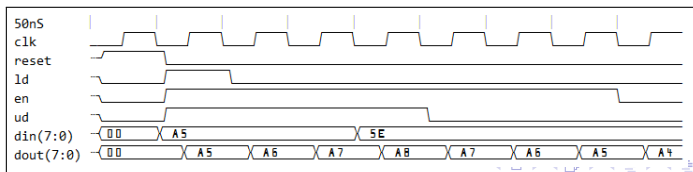




# Περιγραφή ενός up-down counter

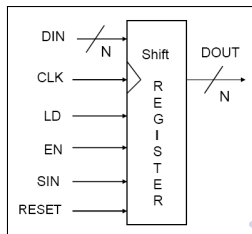
```
module up_down_counter8 (clk, reset, ld, en, ud, din, dout);  
    input      clk, reset, ld, en, ud;  
    input [7:0] din;  
    output [7:0] dout;  
    reg [7:0]  dout;  
  
    always @(posedge clk or posedge reset)  
        if (reset)  
            dout <= 0;  
        else if (ld)  
            dout <= din;  
        else if ((en) && (ud))  
            dout <= dout + 1;  
        else if ((en) && (~ud))  
            dout <= dout - 1;  
        else  
            dout <= dout;  
endmodule
```

Διάγραμμα χρονισμού



# Καταχωρητής ολίσθησης (shift register)

- Ο καταχωρητής ολίσθησης διαθέτει μία είσοδο **EN**, η οποία όταν είναι ενεργή πραγματοποιείται αριστερή (δεξιά) ολίσθηση των περιεχομένων του καταχωρητή κατά μία θέση
- Η είσοδος **SIN** εισάγεται είτε στο **LSB** ενός αριστερού ολισθητή είτε στο **MSB** ενός δεξιού ολισθητή
- Η είσοδος **DIN** προσφέρει είσοδο για παράλληλη αποθήκευση νέων δεδομένων στον εσωτερικό καταχωρητή
- Η επιτρεψη αποθήκευσης (**LD**) έχει προτεραιότητα έναντι της **EN**



## Περιγραφή ενός shift register με ολίσθηση προς τα αριστερά

```
module left_shift_register8(clk, reset, ld, en, sin, din, dout);
  input      clk, reset, ld, en, sin;
  input [7:0] din;
  output [7:0] dout;
  reg [7:0]   dout;

  always @(posedge clk or posedge reset)
    if (reset)
      dout <= 0;
    else if (ld)
      dout <= din;
    else if (en)
      dout <= {dout[6:0], sin};
    else
      dout <= dout;

endmodule
```

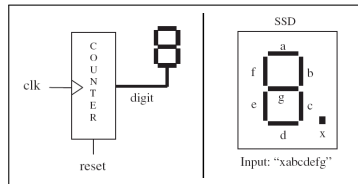
# Αποτελέσματα από την προσομοίωση του left shift register

```
0: reset=x, ld=x, en=x, sin=x, din=xxxxxxxx, dout=xxxxxxxx
10: reset=1, ld=0, en=0, sin=0, din=00000000, dout=00000000
60: reset=0, ld=1, en=1, sin=1, din=10100101, dout=00000000
75: reset=0, ld=1, en=1, sin=1, din=10100101, dout=10100101
110: reset=0, ld=0, en=1, sin=0, din=10100101, dout=10100101
125: reset=0, ld=0, en=1, sin=0, din=10100101, dout=01001010
160: reset=0, ld=0, en=1, sin=1, din=10100101, dout=01001010
175: reset=0, ld=0, en=1, sin=1, din=10100101, dout=10010101
225: reset=0, ld=0, en=1, sin=1, din=10100101, dout=00101011
275: reset=0, ld=0, en=1, sin=1, din=10100101, dout=01010111
325: reset=0, ld=0, en=1, sin=1, din=10100101, dout=10101111
375: reset=0, ld=0, en=1, sin=1, din=10100101, dout=01011111
```

# Παράδειγμα περιγραφής κυκλώματος: Απαριθμητής ψηφίου με έξοδο σε οθόνη επτά τμημάτων (1)

- Δεκαδικός απαριθμητής ενός ψηφίου (μετρά 0 → 9) και μετατρέπει τους δυαδικά κωδικοποιημένους δεκαδικούς (BCD: Binary Coded Decimal) σε μορφή κατάλληλη για απεικόνιση σε οθόνη επτά τμημάτων (SSD: Seven Segment Display)
- Το κύκλωμα διασυνδέεται με την οθόνη ως εξής: abcdefg, με το πιο σημαντικό bit (MSB) να τροφοδοτεί το τμήμα a και το LSB το τμήμα g. Η υποδιαστολή x δεν χρησιμοποιείται

```
module bcdcounter (clk, reset, digit);  
  input  clk, reset;  
  output [6:0] digit;  
  reg [6:0] digit;  
  reg [3:0] temp;
```



## Παράδειγμα περιγραφής κυκλώματος: Απαριθμητής ψηφίου με έξοδο σε οθόνη επτά τμημάτων (2)

```
always @(posedge clk or posedge reset)
begin
  if (reset == 1'b1)
    temp = 4'h0;
  else
  begin
    temp = temp + 1;
    if (temp == 4'hA)
      temp = 4'h0;
  end
  case (temp)
    4'h0: digit <= 7'b1111110;
    4'h1: digit <= 7'b0110000;
    4'h2: digit <= 7'b1101101;
    4'h3: digit <= 7'b1111001;
    4'h4: digit <= 7'b0110011;
    4'h5: digit <= 7'b1011011;
    4'h6: digit <= 7'b1011111;
    4'h7: digit <= 7'b1110000;
    4'h8: digit <= 7'b1111111;
    4'h9: digit <= 7'b1111011;
    default: $display("Invalid BCD code.");
  endcase
end
endmodule
```

# Παράδειγμα περιγραφής κυκλώματος: Απαριθμητής ψηφίου με έξοδο σε οθόνη επτά τμημάτων (3)

Διάγραμμα χρονισμού για το κύκλωμα του απαριθμητή ψηφίου

