Γλώσσες Περιγραφής Υλικού Ι

Θέματα πρακτικής εξάσκησης

Νικόλαος Καββαδίας
nkavv@uop.gr

08 Ιουνίου 2011

Σκιαγράφηση της διάλεξης

- Ανασκόπηση θεμάτων παλαιών εξετάσεων του μαθήματος
  - Εξεταστική περίοδος Ιουνίου-Ιουλίου 2010
  - Εξεταστική περίοδος Σεπτεμβρίου 2010
- Περιεχόμενο εξετάσεων
  - Θεωρητικά θέματα
    - Τι γνωρίζετε για ROM, RAM, FSM, FSMD
  - Ερωτήσεις πολλαπλής επιλογής (multiple-choice)
  - Συνδυαστικά κυκλώματα
  - Μνήμες
  - Ακολουθιακά κυκλώματα
  - Μη-προγραμματιζόμενοι επεξεργαστές (FSMD)

Ερωτήσεις πολλαπλής επιλογής (1)

1. Τι είναι ένα αρχείο testbench;
   A) Το top-level αρχείο του κυκλώματος.
   B) Αρχείο για τον έλεγχο του κυκλώματος.
   Γ) Ένα πακέτο με δηλώσεις του χρήστη.
   Δ) Εναλλακτική περιγραφή του κυκλώματος.
2. Δίνονται τα διανύσματα `a[6:10]`, `b[6:2]`, `c[1:4]`. Ποιο το εύρος του καθενός, αντίστοιχα;
   A) 6, 4, 4. B) 5, 3, 5. Γ) 5, 5, 4. Δ) 6, 4, 5.
3. Για τα παραπάνω διανύσματα `a`, `b`, `c`, ποια από τις παρακάτω αναθέσεις δεν είναι ορθή;
   A) `a[6:10] <= {c, 1'b1};` B) `a[5:4] <= b[4:5];`
   Γ) `c[1:3] = a[6:8];` Δ) `b[2:2] = 1;`
4. Ποια από τις παρακάτω δεν αποτελεί λέξη-κλειδί της Verilog·
   A) `in`. B) `output`. Γ) `wire`. Δ) `reg`.
5. Για το ακόλουθο τμήμα κώδικα Verilog συμπληρώστε τη σωστή λίστα ευαισθησίας.

```
always @(...) begin
  a = b + 1'b1; c = {1'b0, b[5:0]}; d = a ^ c ^ e;
end
```

A) a, b, c, e. B) a, b, c, d, e. Γ) a, c, d. Δ) a, b, d, e.

Ερωτήσεις πολλαπλής επιλογής (2)

6. Σε ποια κωδικοποίηση, δύο διαδοχικές τιμές διαφέρουν πάντα κατά ένα bit;
   A) Gray. B) One-hot. Γ) Two-hot. Δ) Binary.
7. Ποιο το ελάχιστο εύρος διανύσματος διεύθυνσης, για τη διευθυνσιοδότηση μνήμης RAM με 57 θέσεις;
   A) 5. B) 6. Γ) 7. Δ) 8.
8. Τι είδους κύκλωμα υλοποιεί ο ακόλουθος κώδικας;

```
always @(posedge clk)
  q <= d;
```

   A) Έναν πολυπλέκτη 2-σε-1.
   B) Ένα flip-flop τύπου D.
   Γ) Έναν τρισταθή απομονωτή.
   Δ) Ένα μανδαλωτή.

9. Ποιο το αποτέλεσμα της `a = b ^ c;`, αν `b = 4'b1001, c = 4'b1101`:
   A) `4'b0100`. B) `4'b1000`. Γ) `4'b1011`. Δ) `4'b0111`.
10. Έστω `wire a[3:0]` το οποίο οδηγείται ταυτόχρονα από δύο διαφορετικά σήματα, `b = 4'b1111` και `c = 4'b0000`. Ποια είναι η τιμή του;
    A) `4'b0000`. B) `4'b1111`. Γ) `4'bZZZZ`. Δ) `4'bXXXX`.

Συνδυαστική άσκηση 1

- Πολυπλέκτης 3-σε-1 με εισόδους δεδομένων των 4-bit.
- Επίλυση της άσκησης

```
module mux3to1_4b(f, a, b, c, sel);
  output [3:0] f;
  input [3:0] a, b, c;
  input [1:0] sel;
  reg [3:0] f;

  always @(a or b or c or sel)
  begin
    case (sel)
      2'b00 : f = a;
      2'b01 : f = b;
      2'b10 : f = c;
      default : f = 4'hZ;
    endcase
  end
endmodule
```


Συνδυαστική άσκηση 2

- Κύκλωμα πλειοψηφίας (majority voter) τεσσάρων εισόδων του 1-bit. Η έξοδος του είναι `1'b1` όταν τρεις ή παραπάνω από τις εισόδους του έχουν την τιμή `1'b1` αλλιώς είναι `1'b0`.
- Επίλυση της άσκησης

```
module majority_4b (y, a, b, c, d);
  input a, b, c, d;
  output y;
  reg [3:0] abcd;
  reg y;

  always @(a or b or c or d)
  begin
    abcd = {a, b, c, d};
    case (abcd)
      7, 11, 13, 14, 15: y = 1'b1;
      default: y = 1'b0;
    endcase
  end
endmodule
```


Συνδυαστική άσκηση 3

- Αποκωδικοποιητής 2-σε-4.
- Επίλυση της άσκησης

```
module ex2_2 (sel, res);
  input [1:0] sel;
  output [3:0] res;
  reg [3:0] res;

  always @(sel)
  begin
    case (sel)
      2'b00 : res = 4'b0001;
      2'b01 : res = 4'b0010;
      2'b10 : res = 4'b0100;
      2'b11 : res = 4'b1000;
      default : res = 4'b0000;
    endcase
  end
endmodule
```


Αριθμητική μονάδα (1)

- Αριθμητική μονάδα με εισόδους δεδομένων a, b των 8-bit (απρόσημοι), είσοδο επιλογής *sel* των 3-bit, και έξοδο δεδομένων y των 8-bit, η οποία να εκτελεί τις λειτουργίες:
  α) ADD: πρόσθεση των a, b,
  β) SUB: αφαίρεση των a, b,
  γ) XOR: αποκλειστικό-Ή των a, b,
  δ) MOVB: μεταφορά του b στην έξοδο,
  ε) AVG: εξαγωγή του μέσου $((a + b)/2)$ των a, b χωρίς τη χρήση διαίρεσης,
  στ) CMP: σύγκριση των a, b. Η έξοδος y είναι `8'h01` όταν $a > b$, αλλιώς είναι `8'h00`.
  Να γράψετε τις σωστές εξόδους κάθε λειτουργίας για `a = 8'h53, b = 8'hBC`.

Αριθμητική μονάδα (2)

```
module ex3_1(a, b, sel, y);
  input [7:0] a, b;
  input [2:0] sel;
  output [7:0] y;
  reg [7:0] y;
  reg [8:0] p;

  always @(a or b or sel) begin
    case (sel)
      // ADD
      3'b000 : y = a + b;
      // SUB
      3'b001 : y = a + (~b) + 1;
      // XOR
      3'b010 : y = a ^ b;
      // MOVB
      3'b011 : y = b;
      // AVG
      3'b100 : begin
        p = {1'b0, a} + {1'b0, b};
        y = p[8:1];
      end
      // CMP
      3'b101 : y = (a > b) ? 1'b1 : 1'b0;
      default : y = 0;
    endcase
  end
endmodule
```


Έλεγχος ορθής λειτουργίας της αριθμητικής μονάδας

Εκτύπωση διαγνωστικής εξόδου στη στάνταρ έξοδο (κονσόλα)

```
10000: a=53, b=bc, sel=000, y=0f
20000: a=53, b=bc, sel=001, y=97
30000: a=53, b=bc, sel=010, y=ef
40000: a=53, b=bc, sel=011, y=bc
50000: a=53, b=bc, sel=100, y=87
60000: a=53, b=bc, sel=101, y=00
70000: a=53, b=bc, sel=110, y=00
80000: a=53, b=bc, sel=111, y=00
```


Σύγχρονη μνήμη ROM

- Σύγχρονη μνήμη ROM (είσοδοι *clk*, *address*, έξοδος *data* των 8-bit) για τους 8 διαδοχικούς πρώτους αριθμούς, ξεκινώντας από τον 2. Πρώτος αριθμός είναι αυτός που διαιρείται ακριβώς μόνο με το 1 και τον εαυτό του. Χρησιμοποιείστε κατάλληλο εύρος διεύθυνσης.
- Επίλυση της άσκησης

```
module rom_8_8 (clk, address, data);
  input  clk;
  input  [2:0] address;
  output [7:0] data;
  reg [7:0] ROM [0:7];
  reg [7:0] data;

  initial begin
    ROM[ 0] = 8'h02; ROM[ 1] = 8'h03;
    ROM[ 2] = 8'h05; ROM[ 3] = 8'h07;
    ROM[ 4] = 8'h0B; ROM[ 5] = 8'h0D;
    ROM[ 6] = 8'h11; ROM[ 7] = 8'h13;
  end

  always @(posedge clk)
    data <= ROM[address];
endmodule
```


Μνήμη RAM σύγχρονης ανάγνωσης

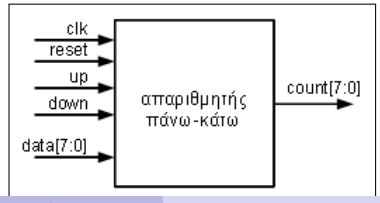
- Μνήμη RAM των 32 θέσεων με σύγχρονη ανάγνωση και εισόδους *clk*, *we* του 1-bit, *address* κατάλληλου εύρους, *din* των 8-bit, και έξοδο *dout* των 8-bit.
- Επίλυση της άσκησης

```
module ex2_1 (clk, we, address, din, dout);
  input  clk, we;
  input  [4:0] address;
  input  [7:0] din;
  output [7:0] dout;
  reg    [7:0] RAM [31:0];
  reg    [7:0] dout;

  always @(posedge clk) begin
    if (we) begin
      RAM[address] <= din;
    end
    dout <= RAM[address];
  end
endmodule
```

Ακολουθιακό κύκλωμα (1)

- Σύγχρονος απαριθμητής των 8-bit με δυνατότητα απαρίθμησης προς τα πάνω και προς τα κάτω (up-down counter). Ο απαριθμητής διαθέτει τις εισόδους `clk, reset, up, down` που είναι είσοδοι ελέγχου, την είσοδο δεδομένων `data` που φορτώνεται κατά το `reset`, και την έξοδο δεδομένων `count`. Όταν μόνο η είσοδος `up` είναι 1 απαριθμεί προς τα πάνω, όταν μόνο η `down` είναι 1 απαριθμεί προς τα κάτω και όταν και οι δύο είσοδοι ελέγχου είναι 1 δεν εκτελείται καμία από τις δύο λειτουργίες.



Ακολουθιακό κύκλωμα (2)

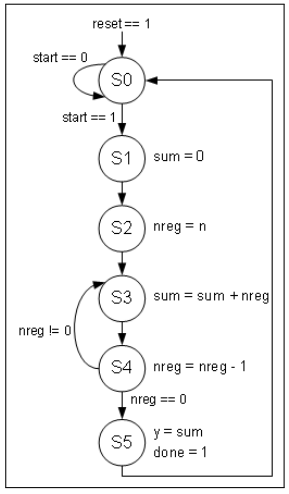
Επίλυση της άσκησης

```
module up_down_counter8 (clk, reset, up, down, data, count);
  input       clk, reset, up, down;
  input[7:0]  data;
  output[7:0] count;
  reg[7:0]    count;

  always @(posedge clk or posedge reset) begin
    if (reset)
      count <= data;
    else if ((up) && (~down))
      count <= count + 1;
    else if ((~up) && (down))
      count <= count - 1;
    else
      count <= count;
  end
endmodule
```


FSMD 1: Κύκλωμα άθροισης όρων (summing circuit)

Το διπλανό σχήμα περιγράφει το διάγραμμα καταστάσεων του FSMD για την άθροιση n διαδοχικών όρων (summing circuit). Το κύκλωμα διαθέτει είσοδο `n` με εύρος W-bit, είσοδο ρολογιού `clk`, επανατοποθέτησης `reset`, έξοδο δεδομένων `y` των W-bit, και έξοδο κατάστασης `done` που γίνεται 1 με την ολοκλήρωση των υπολογισμών. Ακόμη, υπάρχει καταχωρητής `nreg` που προσφέρει προσωρινή αποθήκευση για τα δεδομένα n και καταχωρητής `sum` των W-bit για το συσσωρευόμενο αποτέλεσμα.



Περιγραφή της υλοποίησης FSMD του summing circuit

```
module summing (clk, reset, start, n, y, done);
  parameter WIDTH = 8;
  parameter s0=3'b000, s1=3'b001, s2=3'b010,
            s3=3'b011, s4=3'b100, s5=3'b101;
  input clk, reset, start;
  input [WIDTH-1:0] n;
  output [WIDTH-1:0] y;
  output done;
  reg [WIDTH-1:0] y, sum, n_reg;
  reg done;
  reg[2:0] state;

  always @(posedge clk or posedge reset) begin
    done = 1'b0;
    if (reset) begin
      state = s0; done = 1'b0; y = 0;
    end
    else begin
      case (state)
        s0: begin
          if (start)
            state = s1;
          else
            state = s0;
        end
        s1: begin
          sum = 0;
          state = s2;
        end
        s2: begin
          n_reg = n;
          state = s3;
        end
        s3: begin
          sum = sum + n_reg;
          state = s4;
        end
        s4: begin
          n_reg = n_reg - 1;
          if (n_reg == 0)
            state = s5;
          else
            state = s3;
        end
        s5: begin
          y = sum;
          done = 1'b1;
          state = s0;
        end
        default:
          state = s0;
      endcase
    end
  end
endmodule
```

Έλεγχος ορθής λειτουργίας του FSMD 1 με testbench (1)

- Κλήση της διεργασίας `$readmemh` για την ανάγνωση δεδομένων εισόδου από αρχείο
- Κλήση της διεργασίας `$fopen` και εκτύπωση διαγνωστικής εξόδου σε αρχείο
- Τα περιεχόμενα του αρχείου κειμένου `"summing_test_data_hex.txt"` (n, y)

```
01 01
02 03
03 06
04 0a
05 0f
06 15
07 1c
08 24
09 2d
0a 37
0b 42
0c 4e
0d 5b
0e 69
0f 78
10 88
```

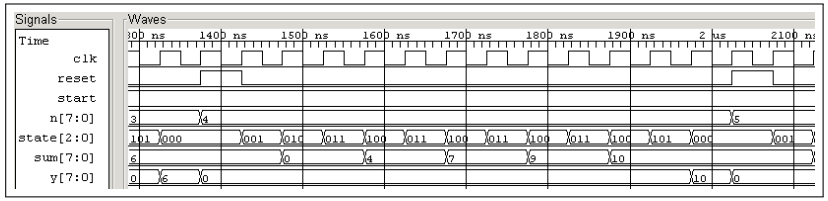

Έλεγχος ορθής λειτουργίας του FSMD 1 με testbench (2)

- Εκτύπωση διαγνωστικής εξόδου στο αρχείο `"summing_alg_test_results.txt"`

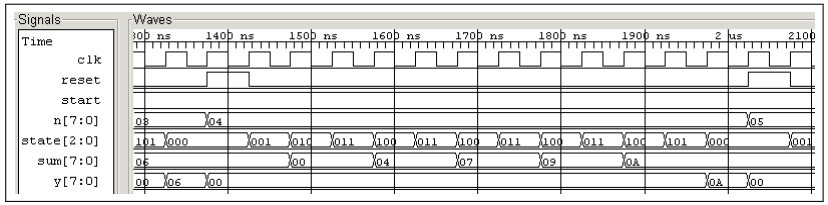
```
SUMMING OK: Number of cycles =           6
SUMMING OK: Number of cycles =           8
SUMMING OK: Number of cycles =          10
SUMMING OK: Number of cycles =          12
SUMMING OK: Number of cycles =          14
SUMMING OK: Number of cycles =          16
SUMMING OK: Number of cycles =          18
SUMMING OK: Number of cycles =          20
SUMMING OK: Number of cycles =          22
SUMMING OK: Number of cycles =          24
SUMMING OK: Number of cycles =          26
SUMMING OK: Number of cycles =          28
SUMMING OK: Number of cycles =          30
SUMMING OK: Number of cycles =          32
SUMMING OK: Number of cycles =          34
SUMMING OK: Number of cycles =          36
SUMMING algorithm test has passed
```


Προσομοίωση του επεξεργαστή summing

Χρονικό διάγραμμα (τιμές στο δεκαδικό)



Χρονικό διάγραμμα (τιμές στο δεκαεξαδικό)



FSMD 2: Κύκλωμα πολλαπλασιασμού αλά ρωσικά (1)

- Ο επόμενος πίνακας περιγράφει σε φυσική γλώσσα κατά βήματα τον αλγόριθμο του πολλαπλασιασμού αλά ρωσικά (Russian peasant multiplication). Ζητείται ο σχεδιασμός του αντίστοιχου κυκλώματος FSMD σε Verilog. Το κύκλωμα διαθέτει θετικές εισόδους $m1, m2$ με εύρος 16-bit, είσοδο ρολογιού *clk*, επανατοποθέτησης *reset*, ενεργοποίησης *start* καθώς και έξοδο δεδομένων y των 16-bit, και έξοδο κατάστασης *done* που γίνεται 1 με την ολοκλήρωση των υπολογισμών. Ακόμη, υπάρχει καταχωρητής p που προσφέρει αποθήκευση του προσωρινού αποτελέσματος και βοηθητικοί καταχωρητές $t1, t2$ των 16-bit.

FSMD 2: Κύκλωμα πολλαπλασιασμού αλά ρωσικά (2)

- Περιγραφή λειτουργίας του πολλαπλασιαστή

Κατάσταση	Λειτουργία
S1	`p = 0; t1 = m1; t2 = m2;`
S2	Αν `t2 > 0`, τότε επόμενη κατάσταση είναι η S3, αλλιώς η S7
S3	Αν ο `t2` είναι περιττός, τότε `p = p + t1;`
S4	Διπλασιασμός του `t1`
S5	Υποδιπλασιασμός του `t2`
S6	Επόμενη κατάσταση η S2
S7	`y = p;`

Περιγραφή της υλοποίησης FSMD του Russian peasant multiplication (1)

```
module ex4 (clk, reset, start, m1, m2, y, done);
  parameter WIDTH = 16;
  parameter s1=3'b000, s2=3'b001, s3=3'b010,
            s4=3'b011, s5=3'b100, s6=3'b101,
            s7=3'b110;
  input clk, reset, start;
  input [WIDTH-1:0] m1;
  input [WIDTH-1:0] m2;
  output [WIDTH-1:0] y;
  output done;
  reg [WIDTH-1:0] y, p;
  reg [WIDTH-1:0] t1, t2;
  reg done;
  reg[2:0] state;

  always @(posedge clk or posedge reset)
  begin
    done = 1'b0;
    if (reset) begin
      state = s1;
      p = 0;
      t1 = 0;
      t2 = 0;
    end
    else begin
```

```
      case (state)
        s1: begin
          if (start) begin
            p = 0;
            t1 = m1;
            t2 = m2;
            state = s2;
          end
          else
            state = s1;
        end
        s2: begin
          if (t2 > 0)
            state = s3;
          else
            state = s7;
        end
        s3: begin
          if (t2[0] == 1'b1) begin
            p = p + t1;
          end
          state = s4;
        end
```


Περιγραφή της υλοποίησης FSMD του Russian peasant multiplication (2)

```
        s4: begin
          t1 <= {t1[WIDTH-2:0], 1'b0};
          state = s5;
        end
        s5: begin
          t2 <= {1'b0, t2[WIDTH-1:1]};
          state = s6;
        end
        s6: begin
          state = s2;
        end
        s7: begin
          y = p;
          done = 1'b1;
          state = s1;
        end
        default:
          state = s1;
      endcase
    end
  end

endmodule
```


Έλεγχος ορθής λειτουργίας του FSMD 2 με testbench (1)

Τα περιεχόμενα του αρχείου κειμένου `"ex4_test_data.txt"` ($m1$, $m2$, y)

```
...
06 07 002a
06 08 0030
06 09 0036
06 0a 003c
06 0b 0042
06 0c 0048
06 0d 004e
06 0e 0054
06 0f 005a
07 00 0000
07 01 0007
07 02 000e
07 03 0015
07 04 001c
07 05 0023
07 06 002a
07 07 0031
07 08 0038
07 09 003f
07 0a 0046
...
```

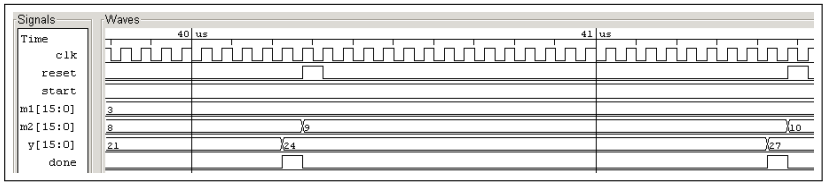
Έλεγχος ορθής λειτουργίας του FSMD 2 με testbench (2)

Εκτύπωση διαγνωστικής εξόδου στο αρχείο "ex4_alg_test_results.txt"

```
...
RPMULT OK: Number of cycles =          23
RPMULT OK: Number of cycles =          23
RPMULT OK: Number of cycles =          23
RPMULT OK: Number of cycles =          23
RPMULT OK: Number of cycles =          23
RPMULT OK: Number of cycles =          23
RPMULT OK: Number of cycles =          23
RPMULT OK: Number of cycles =          23
RPMULT OK: Number of cycles =          28
RPMULT OK: Number of cycles =           8
RPMULT OK: Number of cycles =          13
RPMULT OK: Number of cycles =          13
RPMULT OK: Number of cycles =          18
RPMULT OK: Number of cycles =          18
RPMULT OK: Number of cycles =          18
RPMULT OK: Number of cycles =          18
RPMULT OK: Number of cycles =          23
RPMULT OK: Number of cycles =          23
RPMULT OK: Number of cycles =          23
RPMULT OK: Number of cycles =          23
...
RPMULT algorithm test has passed
```


Προσομοίωση του επεξεργαστή rpmult

Χρονικό διάγραμμα (τιμές στο δεκαδικό)



Χρονικό διάγραμμα (τιμές στο δεκαεξαδικό)