

Γλώσσες Περιγραφής Υλικού I

Σύνταξη παραμετρικών περιγραφών και σχεδίαση μνημών

Νικόλαος Καββαδίας
nkavn@uop.gr

03 Απριλίου 2012

- Σύνταξη παραμετρικών περιγραφών
 - Δήλωση του προεπεξεργαστή της Verilog (``define`, ``ifdef`, ``include`)
 - Δήλωση θυρών κατά το πρότυπο της ANSI C (Verilog 2001)
 - Δήλωση παραμέτρων (`parameter`)
 - Γέννηση κανονικών δομών με τη δήλωση `generate` (Verilog 2001)
 - Υπονοούμενες λίστες ευαισθησίας (Verilog 2001)
 - Βασικά κυκλώματα για χειριστές δεδομένων
- Περιγραφή μνημών ROM και RAM
 - Διεπαφή, οργάνωση και χαρακτηριστικά μνημών
 - Μνήμη μόνο ανάγνωσης (ROM)
 - Μνήμη τυχαίας προσπέλασης (RAM)

Η δήλωση ``define` του προεπεξεργαστή

- Δήλωση ``define` για τον ορισμό μακροαντικαταστάσεων
- Σύνταξη (ορισμός και χρήση):

```
`define Name[(Argument,...)] Text
```

```
...
```

```
`Name [(Expression,...)]
```
- Παρόμοια με την `#define` της ANSI C
- Παράδειγμα 1

```
`define add(a,b) a + b
```

```
...
```

```
f = `add(1,2); // f = 1 + 2;
```

- Παράδειγμα 2

```
`define WORD_WIDTH 32
```

```
`define HIGH 1'b1
```

```
`define LOW 1'b0
```

```
...
```

```
`define WORD ['WORD_WIDTH-1:0]
```

```
`define READ 'LOW
```

```
`define WRITE 'HIGH
```

Η δήλωση ``ifdef...`else...`endif`

- Η δήλωση ``ifdef` χρησιμοποιείται για την υπό συνθήκη μεταγλώττιση τμημάτων κώδικα
- Περιγραφή μοντέλων με εναλλακτικές υλοποιήσεις
- Σύνταξη:

```
`ifdef MacroName
```

```
VerilogCode...
```

```
[`else
```

```
VerilogCode...]
```

```
`endif
```
- Παρόμοια με την `#ifdef...#else...#endif` της ANSI C
- Παράδειγμα

```
`define primitiveModel
```

```
module Test;
```

```
...
```

```
`ifdef primitiveModel
```

```
MyDesign_primitives UUT (...);
```

```
`else
```

```
MyDesign_RTL UUT (...);
```

```
`endif
```

```
endmodule
```

Οι δηλώσεις `undef και `include

- Η δήλωση `undef χρησιμοποιείται για την απενεργοποίηση ενός ορισμού που δόθηκε από μία `define

■ Παράδειγμα

```
'define WORD_WIDTH      32
'undef WORD_WIDTH
'define WORD_WIDTH      16
'define WORD             ['WORD_WIDTH-1:0] // [15:0]
```

- Η δήλωση `include χρησιμοποιείται για τη συμπερίληψη εξωτερικού αρχείου πηγαίου κώδικα Verilog

■ Παράδειγμα

```
// Contents of definitions.v
'define WORD_WIDTH      32
'define WORD             ['WORD_WIDTH-1:0] // [15:0]
...
// Contents of test.v
#include "definitions.v" // WORD_WIDTH and WORD are visible in test.v
```

Η δήλωση `timescale

- Η δήλωση `timescale καθορίζει τη στοιχειώδη μονάδα χρόνου και την ακρίβεια (αναλυτικότητα) της προσομοίωσης (ελάχιστη χρονική ποσότητα)

■ Σύνταξη: `timescale TimeUnit / PrecisionUnit

- όπου: TimeUnit και PrecisionUnit είναι της μορφής <Time> <Unit> με Time = 1, 10 ή 100 και Unit = s, ms, us, ns, ps ή fs

- Καλό είναι να δηλώνεται μία `timescale πριν από την αρχή κάθε module ή έστω μόνο του testbench

■ Παράδειγμα

```
'timescale 10ns / 1ps
```

Δήλωση θυρών κατά το πρότυπο της ANSI C

- Στη Verilog 1995, η δήλωση μιας θύρας απαιτεί μέχρι και τρεις ξεχωριστές δηλώσεις, όπως φαίνεται και στο παρακάτω παράδειγμα (θύρα myport1)

```
module foo(myport1, myport2);
  output [7:0] myport1;
  reg [7:0] myport1;
  input [3:0] myport2;
  ...
endmodule
```

- Στη Verilog 2001, οι δηλώσεις θυρών είναι συντομευμένες και ακολουθούν τον τρόπο με τον οποίο δηλώνονται τα ορίσματα συναρτήσεων στην ANSI C

```
module foo(output reg [7:0] myport1, input [3:0] myport2);
  ...
endmodule
```

ΠΑΡΑΜΕΤΡΟΙ (parameters) στη Verilog

- Συμβολικές σταθερές μέσα σε ένα module
- Χρησιμοποιούνται για τη μοντελοποίηση παραμετρικών (γενικών) μονάδων υλικού
- Π.χ. *N*-bit αθροιστής, *M*-bit καταχωρητής, μοντέλο μνήμης με *NR* καταχωρητές εύρους *DW* bits
- Παράδειγμα: Γενικό μοντέλο LFSR

```
module lfsr_generic(clock, reset, y);
  parameter WIDTH = 8; // Shift register width, also number of taps
  input clock, reset;
  output [WIDTH-1:0] y;
  ...
endmodule
```

- Η προκαθορισμένη τιμή της παραμέτρου WIDTH είναι 8. Η τιμή αυτή μπορεί να μεταβληθεί κατά τη δήλωση αντιτύπου

```
module LFSR_5 (clock, reset, y1);
  input clock, reset;
  output [4:0] y1;
  lfsr_generic #(5) lfsr_5(clock, reset, y1);
endmodule
```

Παράδειγμα χρήσης δήλωσης generate

- Η Verilog 1995 δεν διαθέτει κάποιο τρόπο για την περιγραφή κανονικών δομών υλικού με αλγοριθμικό τρόπο
- Στη Verilog 2001 προστέθηκε η δήλωση generate κατά το πρότυπο της αντίστοιχης δήλωσης που υποστηρίζει η γλώσσα περιγραφής υλικού VHDL
- Παράδειγμα: Μετατροπή από κώδικα Gray σε binary

```
module gray2bin (bin, gray);
  parameter SIZE = 8;
  output [SIZE-1:0] bin;
  input [SIZE-1:0] gray;


  genvar i; // Compile time only

  generate for (i = 0; i < SIZE; i = i + 1)
    begin: bit // IMPORTANT: Use of colon (":") is required for begin.
      assign bin[i] = ^gray[SIZE-1:i];
    end
  endgenerate
endmodule
```

Παραδείγματα κυκλωμάτων που χρησιμοποιούνται σε χειριστές δεδομένων

- Παραμετρικός αθροιστής
- Παραμετρικός καταχωρητής
- Ολισθητής με λειτουργία αριστερής/δεξιάς ολίσθησης
- Υλοποίηση παραμετρικού αθροιστή με generate σε επίπεδο structural

Υπονοούμενες λίστες ευαισθησίας (implicit sensitivity lists)

- Συχνό σφάλμα στη Verilog αποτελεί η δήλωση μίας ελλιπούς λίστας ευαισθησίας
-  Λόγω της μη ευαισθησίας στο σήμα d, ο παρακάτω κώδικας δεν εμφανίζει την αναμενόμενη συμπεριφορά κατά την προσομοίωση

```
always @(a or b or c) // forgot to include d!
  f = a & b | c & d;
```

- Στη Verilog 2001 έχει προστεθεί η δήλωση υπονοούμενης λίστας ευαισθησίας
- Με την παρακάτω δήλωση, απλά ζητείται να ενταχθούν στη λίστα ευαισθησίας, όλα τα σήματα τα οποία εμφανίζονται στο δεξιό μέλος των αναθέσεων σε μία always

```
always @*
  f = a & b | c & d;
```

Παραμετρικός αθροιστής των DW-bit

```
'define DW 8

module add (a, b, sum);
  input ['DW-1:0] a;
  input ['DW-1:0] b;
  output ['DW-1:0] sum;
  wire ['DW:0] temp_sum;

  // Calculate sign-extended result (DW+1 bits)
  assign temp_sum = ({a['DW-1], a}) + ({b['DW-1], b});
  // Copy to sum, neglecting the output carry bit
  assign sum = temp_sum['DW-1:0];
endmodule
```

Παραμετρικός καταχωρητής των DW-bit

```
// Filename: reg_dw.vhd
// Description: DW-bit D-type register with synchronous reset
`timescale 1 ns / 1 ps
`define DW      8

module reg_dw (clk, reset, load, d, q);
    input clk, reset, load;
    input ['DW-1:0] d;
    output ['DW-1:0] q;
    reg ['DW-1:0] q;

    always @(posedge clk)
    begin
        if (reset) begin
            q <= {'DW{1'b0}};
        end
        else if (load) begin
            q <= d;
        end
    end
endmodule
```

Ολισθητής με λειτουργία αριστερής/δεξιάς ολίσθησης (left/right shifter)

```
// Left/right shifter
`define LEFT_SHIFT 1'b0
`define RIGHT_SHIFT 1'b1

// Define a module that contains the function shifter
module lrshifter(outp, inp, control);
    input [31:0] inp;
    input control;
    output [31:0] outp;
    reg [31:0] outp;

    // Compute the right- and left-shifted values whenever
    // a new input value appears
    always @(inp or control)
    begin
        outp = shift(inp, control);
    end

    // Define shift function. The output is a 32-bit value
    function [31:0] shift;
        input [31:0] val;
        input control;
    begin
        shift = (control == 'LEFT_SHIFT) ? (val << 1) : (val >> 1);
    end
    endfunction
endmodule
```

Υλοποίηση παραμετρικού αθροιστή με generate σε επίπεδο structural (Verilog 2001)

```
// Parameterized adder with condition code generation.
module addercc
#(parameter WIDTH = 8) // "Generic" parameter in Verilog-2001.
(
    output reg [WIDTH-1:0] sum, output reg cout, neg, overflow,
    input [WIDTH-1:0] a, b, input cin
);
    reg [WIDTH:0] c;

    generate
    genvar i;
    for (i = 0; i <= WIDTH-1; i = i + 1)
    begin: stage
        always @(*)
        begin
            sum[i] = a[i] ^ b[i] ^ c[i];
            c[i+1] = (a[i] & b[i]) | (b[i] & c[i]) | (a[i] & c[i]);
        end
    end
    endgenerate

    always @(*)
    begin
        c[0] = cin;
        neg = sum[WIDTH-1];
        overflow = c[WIDTH] ^ c[WIDTH-1];
        cout = c[WIDTH];
    end
endmodule
```

Σχεδίαση μνήμης

- Τα κυκλώματα μνήμης προσφέρουν τη δυνατότητα σε ένα σύστημα να αποθηκεύει δεδομένα και αποτελέσματα ώστε να μπορεί να τα επαναχρησιμοποιήσει σε νέους υπολογισμούς
- Βασικές δομές μνήμης
 - Μνήμη μόνο ανάγνωσης (ROM)
 - Μνήμη τυχαίας προσπέλασης (RAM) για ανάγνωση και εγγραφή
- Δομές μνήμης που ανάγονται στις βασικές
 - Πίνακας αναζήτησης (LUT: Look-Up Table): συνήθως αναφέρεται σε μνήμη ROM ή RAM της οποίας τα περιεχόμενα δεν μεταβάλλονται, και η οποία είναι ασύγχρονης ανάγνωσης
 - Αρχείο καταχωρητών (register file): μνήμη RAM με πολλαπλές θύρες εισόδου (εγγραφής) ή/και εξόδου (ανάγνωσης)

Βασικά στοιχεία στη σχεδίαση κυκλωμάτων μνήμης

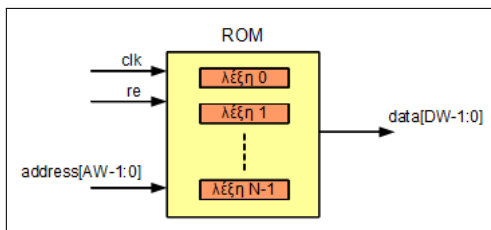
- Τρόποι ανάγνωσης
 - Ασύγχρονη ανάγνωση: αποτελέσματα διαθέσιμα στον ίδιο κύκλο στον οποίο διευθυνσιοδοτήθηκαν με κάποια συνδυαστική χρονική καθυστέρηση
 - Σύγχρονη ανάγνωση: αποτελέσματα διαθέσιμα στον επόμενο κύκλο ρολογιού
- Σήματα επίτρησης
 - RAM Επίτρηση ανάγνωσης (read enable ή *re*)
 - RAM Επίτρηση εγγραφής (write enable ή *we*)
 - RAM,ROM Επίτρηση εξόδου (output enable ή *oe*)
- Παράμετροι
 - Αριθμός θέσεων (καταχωρητών): *N* ή *NR*
 - Εύρος λέξης διεύθυνσης (address width): *AW*
 - Εύρος λέξης δεδομένων (data width): *DW*
 - Η παράμετρος *AW* ορισμένες φορές υπολογίζεται από την *NR* μέσω της έκφρασης: $AW = \lceil \log_2(NR) \rceil$
 - Αριθμός θυρών εισόδου (*NWP*) και εξόδου (*NRP*)

Μνήμη μόνο ανάγνωσης (ROM) (1)

- Μία ROM διαθέτει τουλάχιστον μία είσοδο για τη διευθυνσιοδότηση (*address*) και μία έξοδο για την ανάγνωση των δεδομένων από τη συγκεκριμένη θέση στη μνήμη (*data*)
- Αν είναι σύγχρονη θα διαθέτει είσοδο ρολογιού (*clk*)
- Μπορεί να διαθέτει επίτρηση ανάγνωσης (*re*)
- Τα περιεχόμενα της ROM υλοποιούνται είτε ως τιμές δηλωμένες σε μπλοκ *initial* είτε με αποκωδικοποίηση του σήματος *address*
- Όταν *re = '0'*, τα δεδομένα στην έξοδο συνήθως επιλέγουμε είτε να παραμένουν αμετάβλητα είτε να μην οδηγούνται (υψηλή αντίσταση)
- Η πολλαπλή ανάγνωση από μία θέση στη μνήμη ταυτόχρονα δεν δημιουργεί πρόβλημα διαμάχης (*conflict*)

Μνήμη μόνο ανάγνωσης (ROM) (2)

Διεπαφή της ROM



Σύγχρονη μνήμη ROM των 8-bit με 16 θέσεις και αρχικοποίηση τιμών σε μπλοκ *initial*

```
module rom_16_8 (clk, re, addr, data);
  input clk;
  input re;
  input [3:0] addr;
  output [7:0] data;
  reg [7:0] ROM [15:0];
  reg [7:0] data;

  initial
  begin
    ROM[ 0] = 8'h01; ROM[ 1] = 8'h02;
    ROM[ 2] = 8'h04; ROM[ 3] = 8'h08;
    ROM[ 4] = 8'h10; ROM[ 5] = 8'h20;
    ROM[ 6] = 8'h40; ROM[ 7] = 8'h80;
    ROM[ 8] = 8'h01; ROM[ 9] = 8'h03;
    ROM[10] = 8'h07; ROM[11] = 8'h0F;
    ROM[12] = 8'h1F; ROM[13] = 8'h3F;
    ROM[14] = 8'h7F; ROM[15] = 8'hFF;
  end

  always @(posedge clk)
  begin
    if (re)
      data <= ROM[addr];
  end
endmodule
```

Αρχείο testbench για τη σύγχρονη μνήμη ROM

```
'timescale 1 ns / 10 ps
module main;
  reg clk, re;
  reg [3:0] addr;
  wire [7:0] data;
  integer idx;

  rom_16_8 uut(.clk(clk), .re(re), .addr(addr), .data(data));

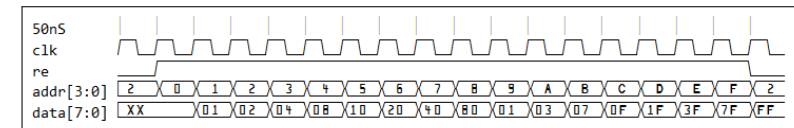
  // Test clock generation.
  initial
    clk = 1'b1;
  always
    #25 clk = ~clk;

  // Data stimulus
  initial begin
    #1 re = 1'b0; addr = 4'h2;
    for (idx = 0; idx <= 15; idx = idx + 1) begin
      #50 addr = idx; re = 1'b1;
    end
    #50
    $finish;
  end

  initial
    $monitor("%t: clk=%b, re=%b, addr=%h, data=%h",
      $time, clk, re, addr, data);
endmodule
```

Αποτελέσματα προσομοίωσης για τη σύγχρονη μνήμη ROM των 8-bit με 16 θέσεις

Διάγραμμα χρονισμού



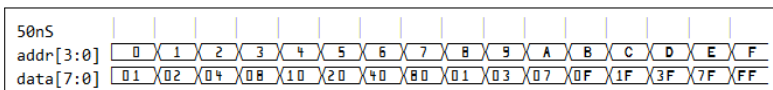
Ασύγχρονη μνήμη ROM των 8-bit με 16 θέσεις και αποκωδικοποίηση διεύθυνσης

```
module rom_16_8 (addr, data);
  input [3:0] addr;
  output [7:0] data;
  reg [7:0] data;

  always @(addr)
  begin
    case (addr)
      4'b0000: data <= 8'h01;
      4'b0001: data <= 8'h02;
      4'b0010: data <= 8'h04;
      4'b0011: data <= 8'h08;
    endcase
  end
endmodule
```

```
4'b0100: data <= 8'h10;
4'b0101: data <= 8'h20;
4'b0110: data <= 8'h40;
4'b0111: data <= 8'h80;
4'b1000: data <= 8'h01;
4'b1001: data <= 8'h03;
4'b1010: data <= 8'h07;
4'b1011: data <= 8'h0F;
4'b1100: data <= 8'h1F;
4'b1101: data <= 8'h3F;
4'b1110: data <= 8'h7F;
4'b1111: data <= 8'hFF;
endcase
end
endmodule
```

Διάγραμμα χρονισμού



Μνήμη τυχαίας προσπέλασης (RAM)

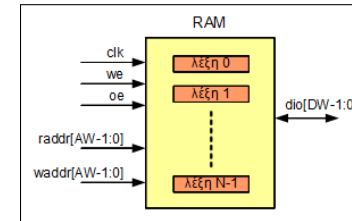
- Μία RAM διαθέτει τουλάχιστον μία είσοδο για τη διευθυνσιοδότηση (address) και τουλάχιστον μία θύρα για την ανάγνωση ή/και εγγραφή δεδομένων από και προς συγκεκριμένη θέση στη μνήμη
- Υποχρεωτικά διαθέτει είσοδο ρολογιού (clk) και επίτρεψη εγγραφής (we) για κάθε θύρα εγγραφής
- Μπορεί να διαθέτει reset για τον καθαρισμό των περιεχομένων όλων των θέσεων
- Τα περιεχόμενα της RAM υλοποιούνται ως πίνακας διανυσμάτων (array of vectors)
- Μπορεί να οριστεί και επίτρεψη ανάγνωσης θύρας εξόδου
- Οι πολλαπλές αιτήσεις για εγγραφή στην ίδια θέση δημιουργούν πρόβλημα διαμάχης και επιλύονται με κατάλληλη λογική ελέγχου (προτεραιότητα)

Διεπαφή και οργάνωση μιας RAM (1)

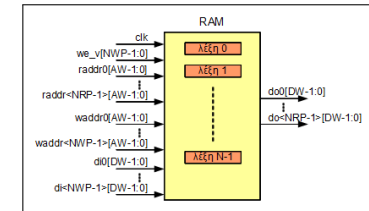
- Μία RAM μπορεί να χρησιμοποιεί ξεχωριστές θύρες για την είσοδο και έξοδο δεδομένων ή αμφίδρομη θύρα η οποία χρησιμοποιείται τόσο για την είσοδο όσο και για την έξοδο δεδομένων (θύρα τύπου `inout`)
- Διαθέτει τουλάχιστον μία είσοδο για τη διευθυνσιοδότηση
- Πολλαπλά σήματα επίτρεψης (π.χ. `we_1`, `we_2`, ... `we_nwp`) μπορούν να αντικατασταθούν από ένα διάνυσμα επίτρεψης (`we_v[`NWP-1:0]`)
- Για την επίλυση της εγγραφής και ανάγνωσης προς/από την ίδια θέση στον ίδιο κύκλο σε RAM σύγχρονης ανάγνωσης θεωρούμε
 - α) τα παλαιά περιεχόμενα της θέσης εμφανίζονται στην έξοδο
 - β) τα νέα δεδομένα γράφονται στην ίδια θέση (ανάγνωση πριν την εγγραφή)
- Σε μοντέρνες τεχνολογίες FPGA, τα ολοκληρωμένα έχουν διαθεσιμότητα ενσωματωμένων μπλοκ μνήμης (block RAM)

Διεπαφή και οργάνωση μιας RAM (2)

- Διεπαφή RAM με μία αμφίδρομη (δικατευθυντική) θύρα



- Διεπαφή RAM με πολλαπλές θύρες ανάγνωσης και εγγραφής



RAM με ασύγχρονη ανάγνωση (1)

■ Στα FPGA αυτή η μνήμη υλοποιείται σε λογικά κελιά (distributed RAM)

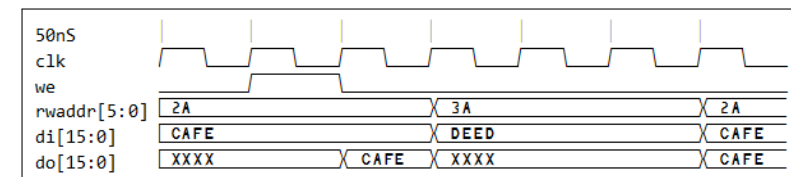
```
module ram_async (clk, we, rwaddr, di, do);
    input  clk;
    input  we;
    input  [5:0] rwaddr;
    input  [15:0] di;
    output [15:0] do;
    reg    [15:0] RAM [63:0];

    always @(posedge clk)
        begin
            if (we)
                RAM[rwaddr] <= di;
        end

    assign do = RAM[rwaddr];
endmodule
```

RAM με ασύγχρονη ανάγνωση (2)

Διάγραμμα χρονισμού



RAM με σύγχρονη ανάγνωση (1)

```
module ram_rf (clk, we, rwaddr, di, do);
  input clk;
  input we;
  input [5:0] rwaddr;
  input [15:0] di;
  output [15:0] do;
  reg [15:0] RAM [63:0];
  reg [15:0] do;

  always @(posedge clk) begin
    if (we)
      begin
        RAM[rwaddr] <= di;
      end
    do <= RAM[rwaddr];
  end
endmodule
```

RAM με σύγχρονη ανάγνωση (2)

Αρχείο testbench (1)

```
'timescale 1 ns / 10 ps
module main;
  reg clk, we;
  reg [5:0] rwaddr;
  reg [15:0] di;
  wire [15:0] do;

  ram_rf uut (
    .clk(clk),
    .we(we),
    .rwaddr(rwaddr),
    .di(di),
    .do(do)
  );

  // Test clock generation.
  initial
    clk = 1'b1;
  always
    #25 clk = ~clk;
endmodule
```

RAM με σύγχρονη ανάγνωση (3)

Αρχείο testbench (2)

```
// Data stimulus
initial
  begin
    #1 we = 1'b0; rwaddr = 6'h2A; di = 16'hCAFE;
    #50 we = 1'b1; rwaddr = 6'h2A; di = 16'hCAFE;
    #50 we = 1'b0; rwaddr = 6'h2A; di = 16'hCAFE;
    #50 we = 1'b0; rwaddr = 6'h3A; di = 16'hDEED;
    #50 we = 1'b0; rwaddr = 6'h3A; di = 16'hDEED;
    #50 we = 1'b0; rwaddr = 6'h3A; di = 16'hDEED;
    #50 we = 1'b0; rwaddr = 6'h2A; di = 16'hCAFE;
    #50 we = 1'b1; rwaddr = 6'h2A; di = 16'hCAFE;
    #50
    $finish;
  end

initial
  $monitor("%t: clk=%b, we=%b, rwaddr=%h, di=%h, do=%h",
    $time, clk, we, rwaddr, di, do);
endmodule
```

RAM με σύγχρονη ανάγνωση (4)

Διάγραμμα χρονισμού

