

Γλώσσες Περιγραφής Υλικού I

Προχωρημένα στοιχεία της Verilog HDL

Νικόλαος Καββαδίας
nkavn@uop.gr

27 Μαρτίου 2012

Σκιαγράφηση της διάλεξης

- Προχωρημένα στοιχεία της Verilog HDL
 - Χρήση τελεστών στη σύνταξη εκφράσεων και αναθέσεων στη Verilog HDL
 - Αριθμητικές αναπαραστάσεις
 - Στοιχειώδεις υπομονάδες καθοριζόμενες από το χρήστη (UDP: User-Defined Primitives)
 - Υποπρογράμματα
 - Διεργασίες ή διαδικασίες (tasks)
 - Συναρτήσεις (functions)
 - Ιεραρχική σχεδίαση: Παράδειγμα σχεδιασμού αριθμητικής-λογικής μονάδας (ALU) με υπομονάδες

Σύνταξη κώδικα στη Verilog HDL: Βασικές συμβάσεις

- Η Verilog HDL όπως και η ANSI C έχει ευαισθησία πεζών-κεφαλαίων (case sensitivity). Όλες οι λέξεις-κλειδιά στη Verilog γράφονται με πεζούς χαρακτήρες
- Οι ειδικοί χαρακτήρες \b (κενό), \t (στηλογνώμονας) και \n (νέα γραμμή) δεν είναι εκτυπώσιμοι (κενό διάστημα)
- Στον κώδικα συχνά εισάγουμε σχόλια προκειμένου να βελτιώσουμε την αναγνωσιμότητά του αλλά και ως ένα είδος άμεσης τεκμηρίωσης
- Τα σχόλια στη Verilog είναι μίας γραμμής ή πολλαπλών γραμμών και σημειώνονται όπως στη C

```
a = b && c; // This is a one-line comment

/* This is
a
multi-line comment. */

/* This is /* an ILLEGAL!!! */ comment */
```

Χρήση τελεστών σε αναθέσεις

- Στη Verilog HDL υπάρχουν μοναδιαίοι (unary), δυαδικοί (binary) και τριαδικοί (ternary) τελεστές

```
// ~ is a unary operator  
a = ~ b;  
  
// && is a binary operator. b and c are operands.  
// The result of this operation (logical AND) is  
// written to a  
a = b && c;  
  
// ?: is a ternary operator. b, c and d are operands  
// If b is not equal to ZERO then a is assigned c, else  
// it is assigned d  
a = b ? c : d;
```

Αριθμητικές ποσότητες σε εκφράσεις

- Η δήλωση αριθμών στη Verilog γίνεται με προαιρετική αναφορά του εύρους τους σε bit. Η δήλωση του εύρους αποτελεί καλή πρακτική για αποφυγή λαθών
- Διαμόρφωση αριθμού:
<size> ' <base_format> <number>
- Αριθμητικές βάσεις
 - Δυαδικό: 'b ή 'B
 - Οκταδικό: 'o ή 'O
 - Δεκαδικό: 'd ή 'D
 - Δεκαεξαδικό: 'h ή 'H
- Παραδείγματα

```
4'b1111 // 4-bit binary number  
12'habc // 12-bit hexadecimal number  
16'd255 // 16-bit decimal number
```

- Προσημασμένοι αρνητικοί αριθμοί

```
-6'd3 // negative number stored as 2's complement of 3  
4'd-2 // ILLEGAL!
```

Αναπαράσταση ακεραίων με διανύσματα

Απρόσημοι ακεραίοι σε δυαδικό
και δεκαεξαδικό

| Ακέραιος | Δυαδικό | hex |
|----------|-----------|-------|
| 0 | 5'b000000 | 5'h0 |
| 1 | 5'b000001 | 5'h1 |
| 2 | 5'b000010 | 5'h2 |
| 3 | 5'b000011 | 5'h3 |
| 4 | 5'b000100 | 5'h4 |
| 5 | 5'b000101 | 5'h5 |
| 6 | 5'b000110 | 5'h6 |
| 7 | 5'b000111 | 5'h7 |
| 8 | 5'b010000 | 5'h8 |
| 9 | 5'b010001 | 5'h9 |
| 10 | 5'b010010 | 5'hA |
| 11 | 5'b010011 | 5'hB |
| 12 | 5'b010100 | 5'hC |
| 13 | 5'b010101 | 5'hD |
| 14 | 5'b010110 | 5'hE |
| 15 | 5'b010111 | 5'hF |
| 16 | 5'b100000 | 5'h10 |
| 17 | 5'b100001 | 5'h11 |

Προσημασμένοι ακεραίοι σε
συμπλήρωμα ως προς 2

| Ακέραιος | Δυαδικό | hex |
|----------|---------|------|
| -8 | 4'b1000 | 4'h8 |
| -7 | 4'b1001 | 4'h9 |
| -6 | 4'b1010 | 4'hA |
| -5 | 4'b1011 | 4'hB |
| -4 | 4'b1100 | 4'hC |
| -3 | 4'b1101 | 4'hD |
| -2 | 4'b1110 | 4'hE |
| -1 | 4'b1111 | 4'hF |
| 0 | 4'b0000 | 4'h0 |
| 1 | 4'b0001 | 4'h1 |
| 2 | 4'b0010 | 4'h2 |
| 3 | 4'b0011 | 4'h3 |
| 4 | 4'b0100 | 4'h4 |
| 5 | 4'b0101 | 4'h5 |
| 6 | 4'b0110 | 4'h6 |
| 7 | 4'b0111 | 4'h7 |

Παραδείγματα χρήσης αριθμητικών τελεστών


■ Δυαδικοί τελεστές

```
A = 4'b0011; // A is a 4-bit register
B = 4'b0100; // so is B
D = 6;      // D, E are integers
E = 4;

A * B // Multiply A and B = 4'b1100
D / E // Dividing D by E equals to 1 (truncates fractional part)
A + B // Equals to 4'b0111
B - A // Equals to 4'b0001
```

■ Τελεστής ακέραίου υπολοίπου

```
13 % 3 // = 1
16 % 4 // = 0
-7 % 2 // -1, taking sign of the first operand (negative)
7 % -2 // 1, taking sign of the first operand (positive)
```

 Αν κάποιος από τα bit είναι **X** τότε οποιαδήποτε υπολογιζόμενη έκφραση ισούται με **X**

Παραδείγματα χρήσης μοναδιαίων και λογικών τελεστών

■ Μοναδιαίοι τελεστές

```
-4 // Negative number with absolute value of 4  
+5 // Positive number with absolute value of 5
```

■ Υπολογισμοί με αρνητικούς αριθμούς

```
-10 / 5 // = -2
```

■ Λογικοί τελεστές

```
A = 3;  
B = 0;  
  
A && B // evaluates to 0  
A || B // evaluates to 1  
!A // evaluates to 0  
!B // evaluates to 1  
  
A = 2'b0x;  
B = 2'b10;  
A && B // evaluates to x, equivalent to (x && TRUE)  
  
(a == 2) && (b == 3) // evaluates to 1 if both a == 2 and b == 3 are TRUE,  
// evaluates to FALSE if either one is false.
```


Παραδείγματα χρήσης συσχετιστικών και bitwise τελεστών

■ Συσχετιστικοί τελεστές (για συγκρίσεις)

```
A = 4;  
B = 3;  
X = 4'b1010;  
Y = 4'b1101;  
Z = 4'b1xxx;  
  
A <= B    // evaluates to logical 0 (FALSE)  
A > B     // evaluates to TRUE  
Y >= X    // evaluates to TRUE  
Y < Z     // evaluates to an X (neither FALSE or TRUE)  
A == B    // evaluates to FALSE  
X != Y    // evaluates to TRUE  
X == Z    // evaluates to an X
```

■ bitwise τελεστές

```
X = 4'b1010; Y = 4'b1101; Z = 4'b10x1;  
  
~X        // negation. Result is 4'b0101  
X & Y     // bitwise and. Result is 4'b1000  
X | Y     // bitwise or. Result is 4'b1111  
X ^ Y     // bitwise xor. Result is 4'b0111  
X ~^ Y    // bitwise xnor. Result is 4'b1000  
X & Z     // result is 4'b10x0
```

Παραδείγματα χρήσης τελεστών μείωσης και ολίσθησης

- Τελεστές μείωσης: δέχονται ένα μόνο όρισμα το οποίο είναι ένα διάνυσμα και παράγουν ως αποτέλεσμα μία ποσότητα του 1 bit
- Εφαρμόζουν τον ίδιο bitwise τελεστή διαδοχικά, σε όλα τα ψηφία του διανύσματος

```
X = 4'b1010;  
  
&X // equivalent to 1 & 0 & 1 & 0. Results in 1'b0  
|X // equivalent to 1 | 0 | 1 | 0. Results in 1'b1  
^X // equivalent to 1 ^ 0 ^ 1 ^ 0. Results in 1'b0
```

- Οι τελεστές ολίσθησης προσδιορίζουν την αριστερή και τη δεξιά ολίσθηση κατά ένα αριθμό θέσεων

```
X = 4'b1100;  
  
Y = X >> 1; // Y is 4'b0110 (shifted right by 1, filled MSB with 0)  
Y = X << 1; // Y is 4'b1000 (shifted left by 1, filled LSB with 0)  
Y = X << 2; // Y is 4'b0000 (shifted left by 2 bits)
```

Παραδείγματα χρήσης τελεστών συνένωσης και επανάληψης

- Ο τελεστής συνένωσης προσφέρει ένα μηχανισμό για τη σύνθεση ενός ορίσματος από πολλά μικρότερου εύρους

```
reg A;  
reg [1:0] B, C;  
reg [2:0] D;  
A = 1'b1; B = 2'b00; C = 2'b10; D = 3'b110;  
  
Y = {B, C}; // Result Y is 4'b0010  
Y = {A, B, C, D, 3'b001}; // Result Y is 11'b10010110001  
Y = {A, B[0], C[1]}; // Result Y is 3'b101
```

- Τελεστής επανάληψης (κλωνοποίησης): πραγματοποιεί επαναληπτική συνένωση ενός δοθέντος αριθμού όσες φορές δηλώνει μία σταθερά επανάληψης

```
A = 1'b1; B = 2'b00; C = 2'b10; D = 3'b110;  
  
Y = { 4{A} }; // Result Y is 4'b1111  
Y = { 4{A}, 2{B} }; // Result Y is 8'b11110000  
Y = { 4{A}, 2{B}, C }; // Result Y is 10'b1111000010
```

Στοιχειώδεις μονάδες καθοριζόμενες από το χρήστη (UDP: User-Defined Primitives)

- Η χρήση UDP προσφέρει μία μέθοδο για τον ορισμό απλών ή σύνθετων πυλών και απλών ακολουθιακών κυκλωμάτων με τη βοήθεια ενός πίνακα αληθείας (truth table)
- Στα πλεονεκτήματα της χρήσης UDP είναι η ταχύτερη προσομοίωση σε επίπεδο structural σχεδίασης
- Δίνει πιο άμεσο έλεγχο στον προσδιορισμό της συμπεριφοράς ενός κυκλώματος όταν κάποια/κάποιες είσοδοι είναι X
- ☠ Τα UDP δεν είναι γενικώς συνθέσιμα καθώς δεν έχουν χαρακτηριστεί εκ των προτέρων από το εργαλείο λογικής σύνθεσης
- Χρησιμοποιούνται συνήθως για τη σχεδίαση βιβλιοθηκών κυκλωματικών στοιχείων για τεχνολογίες ASIC

Υλοποίηση πύλης AND των 2 εισόδων ως UDP

```
primitive udp_and2(out, a, b);  
  output out;  
  input a, b;  
  
  table  
  // a b : out;  
    0 0 : 0;  
    0 1 : 0;  
    1 0 : 0;  
    1 1 : 1;  
  endtable  
  
endprimitive
```

Υλοποίηση πολυπλέκτη 2-σε-1 ως UDP

```
primitive mux2to1(f, a, b, sel);
  output f;
  input a, b, sel;

  table
    // Behavior defined using a truth table that includes
    // "don't-cares"
    1?0 : 1;
    0?0 : 0;
    ?11 : 1;
    ?01 : 0;
    11? : 1;
    00? : 0;
  endtable
endprimitive
```

Πλήρης αθροιστής με UDP

```
module fulladd(sum, cout, a, b, cin);
  output sum, cout;
  input a, b, cin;
  wire s1, c1, c2;

  xor      g1(s1, a, b);
  udp_and2 g2(c1, a, b);

  xor      g3(sum, s1, cin);
  udp_and2 g4(c2, s1, cin);

  or       g5(cout, c2, c1);

endmodule
```

Στοιχεία της Verilog HDL που δεν χρησιμοποιούνται συχνά

- Μοντελοποίηση κυκλώματος σε επίπεδο διακοπών (switch-level modeling)
 - Η προσομοίωση στο επίπεδο αυτό είναι πολύ πιο αργή από τα επίπεδα πυλών και RTL
 - Δεν αποκαλύπτει λεπτομέρειες για τα περισσότερα προβλήματα σχεδίασης σε επίπεδο τρανζίστορ
 - Η προσομοίωση σε τόσο χαμηλό επίπεδο γίνεται συνήθως με προσομοίωση συστημάτων διαφορικών εξισώσεων (εργαλεία τύπου SPICE)
- Χρονικές καθυστερήσεις σε κυκλωματικά μοντέλα
 - Η προσομοίωση παρουσία καθυστερήσεων δεν αυξάνει τη γνώση μας για την αξιοπιστία ενός κυκλώματος
 - Δεν πληροφορεί για τη χειρότερη περίπτωση λειτουργίας του κυκλώματος (worst case)
 - Έχει πλέον αντικατασταθεί από τη στατική χρονική ανάλυση (static timing analysis) στη σχεδίαση βιβλιοθηκών ASIC

Υποπρογράμματα

- Τα υποπρογράμματα στη Verilog χρησιμοποιούνται με παρόμοιο τρόπο σε σχέση με τις συμβατικές γλώσσες διαδικαστικού προγραμματισμού
- Επιτρέπουν τη χρήση επαναλαμβανόμενων τμημάτων κώδικα χωρίς να χρειάζεται αυτά να ξαναγραφτούν
- Καταμερισμός κώδικα σε μικρότερα τμήματα για ευκολότερη διαχείριση (δομημένος προγραμματισμός)
- Τα υποπρογράμματα είναι χρήσιμα για την επαναχρησιμοποίηση τμημάτων κώδικα που δεν είναι απαραίτητα συνθέσιμα (π.χ. σε testbenches)
- Η Verilog παρέχει συναρτήσεις (functions) και διαδικασίες (tasks) οι οποίες είναι τοπικές μέσα στο module στο οποίο ορίζονται
- Μπορούν να κληθούν από άλλο module με το ιεραρχικό τους όνομα

Διαφορές μεταξύ διαδικασιών (TASK) και συναρτήσεων (FUNCTION)

FUNCTIONS

- Μία `function` μπορεί να ενεργοποιήσει μία άλλη `function` αλλά όχι μία `task`
- Οι συναρτήσεις εκτελούνται σε μηδενικό χρόνο προσομοίωσης
- Οι συναρτήσεις δεν περιλαμβάνουν δηλώσεις χρονισμού ή καθυστέρησης
- Πρέπει να έχουν τουλάχιστον ένα όρισμα εισόδου (`input`)
- Πάντα επιστρέφουν ακριβώς μία τιμή, ενώ δεν έχουν ορίσματα τύπου `output` ή `inout`

TASKS

- Μία `task` μπορεί να ενεργοποιήσει άλλες `task` και `function`
- Οι διαδικασίες μπορούν να εκτελούνται σε μετρήσιμο χρόνο προσομοίωσης
- Οι διαδικασίες μπορούν να περιλαμβάνουν δηλώσεις χρονισμού ή καθυστέρησης
- Μπορούν να έχουν μηδέν ή περισσότερα ορίσματα του τύπου `input`, `output`, ή `inout`
- Δεν επιστρέφουν τιμή, αλλά περνούν πολλαπλές τιμές μέσω ορισμάτων `output` ή `inout`

ΣΥΝΑΡΤΗΣΗ (FUNCTION)

- Οι συναρτήσεις αποτελούν είδος υποπρογράμματος το οποίο επιστρέφει μία μοναδική τιμή
- Οι συναρτήσεις καλούνται από εκφράσεις
- Σύνταξη:

```
function [RangeOrType] FunctionName;  
Declarations...  
Statement  
endfunction
```

- Ως RangeOrType μπορεί να χρησιμοποιηθεί integer, real ή εύρος bit
- Κλήση συνάρτησης
<Function name> (<expression><, <expression> >*);

Παραδείγματα συναρτήσεων (1)

- Παράδειγμα: Συνάρτηση υπολογισμού ομοτιμίας

```
module parity;
  reg [31:0] addr;
  reg parity;

  always @(addr)
  begin
    // First invocation of parity_gen
    parity = parity_gen(addr);
    $display("Parity calculated = %b", parity_gen(addr));
  end

  function parity_gen;
    input [31:0] address;
    begin
      parity_gen = ^address;
    end
  endfunction
endmodule
```

Παραδείγματα συναρτήσεων (2)

- Παράδειγμα: Συνάρτηση $\lceil \log_2 \rceil$
- Μια συνάρτηση λογαρίθμου ως προς 2 τύπου ceiling (στρογγυλοποίηση αποτελέσματος στον πλησιέστερο μεγαλύτερο ακέραιο) είναι ιδιαίτερα χρήσιμη για την περιγραφή εύρους bit διευθύνσεων σε σχέση με το συνολικό αριθμό των διευθυνσιοδοτούμενων θέσεων, με μία μόνο παράμετρο
- Παραδείγματα χρήσης: αποκωδικοποιητές, κωδικοποιητές, γενικευμένοι πολυπλέκτες

```
function integer log2c;  
  input inp;  
  integer temp, i;  
  begin  
    log2c = 0;  
    temp = 1;  
    for (i = 0; i <= inp; i++)  
      begin  
        if (temp < inp)  
          begin  
            log2c = log2c + 1;  
            temp = temp * 2;  
          end  
        end  
      end  
    endfunction
```

Παραδείγματα συναρτήσεων (3)

- Υλοποίηση της συνάρτησης ReverseBits

```
function [7:0] ReverseBits;  
  input [7:0] Byte;  
  integer i;  
  begin  
    for (i = 0; i < 8; i = i + 1)  
      ReverseBits[7-i] = Byte[i];  
    end  
  endfunction
```

- Κλήση της συνάρτησης ReverseBits

```
byte = ReverseBits(byte);
```

ΔΙΑΔΙΚΑΣΙΑ (TASK)

- Οι διαδικασίες συντάσσονται με παρόμοιο τρόπο με τις function
- Καλούνται από δηλώσεις (statements)
- Μια διαδικασία μπορεί να χρησιμοποιηθεί ως εντολή από μόνη της
- Οι παράμετροι μπορεί να έχουν κατευθυντικότητα (input, output, inout)
- Σύνταξη:
task TaskName;
[declaration list]
statement list
endtask
- Η λίστα ορισμών (declaration list) μπορεί να περιλαμβάνει αντικείμενα input, output, inout, reg ή parameter
- Κλήση διαδικασίας
<Task name>;
or
<Task name> (<expression><, <expression> >*);

Παραδείγματα διαδικασιών (1)

■ Παράδειγμα: Διαδικασία sort

```
module keep_order;
  integer in1a, in2a;
  integer mina, maxa;

  always @(in1a or in2a)
  begin
    sort(in1a, in2a, mina, maxa);
  end

  task sort;
    input integer in1, in2;
    output integer min, max;
    begin
      if (in1 >= in2)
        begin
          max = in1;
          min = in2;
        end
      else
        begin
          max = in2;
          min = in1;
        end
      end
    endtask
endmodule
```


Παραδείγματα διαδικασιών (2)

- Υλοποίηση της διαδικασίας counter

```
task counter;  
  inout [3:0] count;  
  input reset;  
  
  if (reset)    // Synchronous Reset  
    count = 0; // Must use non-blocking for RTL  
  else  
    count = count + 1;  
  
endtask
```

- Κλήση της διαδικασίας counter

```
always @(posedge clock)  
  counter(cnt, rst);
```

Λίστα των προκαθορισμένων λέξεων-κλειδιών στη Verilog

| | | | |
|--------------|-------------|-----------|----------|
| and | for | output | strong1 |
| always | force | parameter | supply0 |
| assign | forever | pmos | supply1 |
| begin | fork | posedge | table |
| buf | function | primitive | task |
| buff0 | highz0 | pulldown | tran |
| buff1 | highz1 | pullup | tranif0 |
| case | if | pull0 | tranif1 |
| casex | ifnone | pull1 | time |
| casez | initial | rcmos | tri |
| cmos | inout | real | triand |
| deassign | input | realtime | trior |
| default | integer | reg | trireg |
| defparam | join | release | tri0 |
| disable | large | repeat | tri1 |
| edge | macromodule | rnmos | vectored |
| else | medium | rpmos | wait |
| end | module | rtran | wand |
| endcase | nand | rtranif0 | weak0 |
| endfunction | negedge | rtranif1 | weak1 |
| endprimitive | nor | scalared | while |
| endmodule | not | small | wire |
| endspecify | notif0 | specify | wor |
| endtable | notif1 | specparam | xnor |
| endtask | nmos | strength | xor |
| event | or | strong0 | |

Ιεραρχική σχεδίαση σε Verilog: ALU αποτελούμενη από υπομονάδες (1)

- Περιγραφή της αριθμητικής-λογικής μονάδας (ALU) της 2ης διάλεξης με χρήση υπομονάδων
- Δημιουργία ξεχωριστών περιγραφών για τις επιμέρους υπομονάδες: λογική μονάδα (`logic.v`), αριθμητική μονάδα (`arith.v`) και πολυπλέκτης διανυσμάτων (`vmux.v`)
- Διασύνδεση των επιμέρους υπομονάδων για την περιγραφή του συνολικού κυκλώματος στο αρχείο `alu.v`

ALU αποτελούμενη από υπομονάδες (2)

- Περιγραφή του πολυπλέκτη διανυσμάτων

```
// vmux.v
module vmux(a, b, sel, y);
  input [7:0] a, b;
  input sel;
  output [7:0] y;
  reg [7:0] y;

  // Multiplexer
  always @(a or b or sel) begin
    case (sel)
      1'b0 : y = a;
      default : y = b;
    endcase
  end

endmodule
```

ALU αποτελούμενη από υπομονάδες (3)

■ Περιγραφή της αριθμητικής μονάδας

```
// arith.v
module arith_unit(a, b, cin, sel, y);
  input [7:0] a, b;
  input cin;
  input [2:0] sel;
  output [7:0] y;
  reg [7:0] y;

  // Arithmetic unit
  always @(a or b or cin or sel) begin
    case (sel)
      3'b000 : y = a;
      3'b001 : y = a + 1;
      3'b010 : y = a - 1;
      3'b011 : y = b;
      3'b100 : y = b + 1;
      3'b101 : y = b - 1;
      3'b110 : y = a + b;
      default : y = a + b + cin;
    endcase
  end
endmodule
```

ALU αποτελούμενη από υπομονάδες (4)

■ Περιγραφή της λογικής μονάδας

```
// logic.v
module logic_unit(a, b, sel, y);
    input [7:0] a, b;
    input [2:0] sel;
    output [7:0] y;
    reg [7:0] y;

    // Logic unit
    always @(a or b or sel) begin
        case (sel)
            3'b000 : y = ~a;
            3'b001 : y = ~b;
            3'b010 : y = a & b;
            3'b011 : y = a | b;
            3'b100 : y = ~(a & b);
            3'b101 : y = ~(a | b);
            3'b110 : y = a ^ b;
            default : y = ~(a ^ b);
        endcase
    end
endmodule
```

ALU αποτελούμενη από υπομονάδες (5)

Η περιγραφή της ALU στο υψηλότερο ιεραρχικά επίπεδο, χρησιμοποιεί τις υπομονάδες για την υλοποίηση του συνολικού κυκλώματος

```
// alu.v
module alu(a, b, cin, sel, y);
  input [7:0] a, b;
  input cin;
  input [3:0] sel;
  output [7:0] y;
  wire [7:0] y;
  wire [7:0] arith_res;
  wire [7:0] logic_res;

  // Arithmetic unit
  arith_unit t1(.a(a), .b(b), .cin(cin), .sel(sel[2:0]), .y(arith_res));

  // Logic unit
  logic_unit t2(.a(a), .b(b), .sel(sel[2:0]), .y(logic_res));

  // 2-to-1 8-bit multiplexer
  vmux      t3(.a(arith_res), .b(logic_res), .sel(sel[3]), .y(y));
endmodule
```

Αποτελέσματα από την προσομοίωση της ALU

```
10: a=93, b=a7, cin=0, sel=0, y=93
20: a=93, b=a7, cin=0, sel=1, y=94
30: a=93, b=a7, cin=0, sel=2, y=92
40: a=93, b=a7, cin=0, sel=3, y=a7
50: a=93, b=a7, cin=0, sel=4, y=a8
60: a=93, b=a7, cin=1, sel=5, y=a6
70: a=93, b=a7, cin=0, sel=6, y=3a
80: a=93, b=a7, cin=0, sel=7, y=3a
90: a=93, b=a7, cin=0, sel=8, y=6c
100: a=93, b=a7, cin=0, sel=9, y=58
110: a=93, b=a7, cin=0, sel=a, y=83
120: a=93, b=a7, cin=0, sel=b, y=b7
130: a=93, b=a7, cin=0, sel=c, y=7c
140: a=93, b=a7, cin=0, sel=d, y=48
150: a=93, b=a7, cin=0, sel=e, y=34
160: a=93, b=a7, cin=0, sel=f, y=cb
```