Σχεδίαση Ψηφιακών κυκλωμάτων Project

Αραβαντινός Λεωνίδης Χρήστος icsd12012

Κοντοπός Παναγιώτης icsd12086

Πίνακας περιεχομένων

Λίγα Λόγια για το Project μας

- ✓ Το θέμα της εργασίας μας ήταν η δημιουργία ενός calculator , σε κώδικα ***Verilog*** , για την πρόσθεση και αφαίρεση μονοψήφιων αριθμών [0-9] με αποτελέσματα στο εύρος [-9,18].

- ✓ Οι επιθυμητοί τελεστέοι, σύμβολα(πρόσθεσης , αφαίρεσης , ισότητας) πληκτρολογούνται από το keyboard και τα αποτελέσματα εμφανίζονται σε 6 ***7-segment displays*** του ***FPGA Altera Cyclone IV*** που θα τρέξει το ολοκληρωμένο μας κύκλωμα.

- ✓ Παρακάτω περιγράφονται , αναλυτικά και περιγραφικά , τα διάφορα modules που δημιουργήθηκαν , ο τρόπος υλοποίησης του ***state machine*** και περιέχουμε και ένα παράδειγμα εξομοίωσης στο ***Modelsim*** , το οποίο ήταν καθοριστικό για την σωστή δημιουργία του κώδικα πριν περάσει από σύνθεση στο εργαλείο ***Quartus II*** .

- ✓ Επίσης περιλαμβάνουμε και ένα link στο οποίο παρουσιάζουμε , με βίντεο , περιεκτικά την λειτουργία του δικού μας calculator στο ***FPGA*** .

Παρουσίαση κώδικα και περιγραφή των modules

Module πρόσθεσης και αφαίρεσης των αριθμών

a. Στο ***Module*** αυτό , αν γίνεται πρόσθεση , αθροίζουμε τις τιμές των δύο δυαδικών καταχωρητών ,που είναι και οι είσοδοι στο module μας , όπως είσοδος είναι και το σύμβολο της πράξης .

b. Ειδάλλως ο αριθμός που θα εισαχθεί από το πληκτρολόγιο , μετά από το πάτημα του συμβόλου της αφαίρεσης(-) , θα μετατραπεί στο συμπλήρωμά του ως προς 1 και θα του προστεθεί 1 bit.

c. Με αυτό τον τρόπο θα τον μετατρέψουμε στον αρνητικό του , ο οποίος είναι ουσιαστικά το συμπλήρωμά του ως προς 2 .

d. Η πράξη καθορίζεται από την τιμή του καταχωρητή ***symbol*** ο οποίος θα δεχτεί την τιμή του με βάση το ***State Machine*** του ***Top-level Module*** μας.

e. Η έξοδος του module είναι το αποτέλεσμα της πράξης .

```
module Add_Subtract(
output [5:0] result ,
input        symbol ,
input [5:0]  A , B     );

assign result = (symbol) ? ( A+(~B+1) ) : (A+B) ;

endmodule
```

Module αποκωδικοποιητή πρώτου επιπέδου

a. Ο αποκωδικοποιητής πρώτου επιπέδου δέχεται ως είσοδο τα 8-bit από τα 11-bit που δειγματοληπτούμε από το ***keyb_data***.

b. Τα 8-bits είναι η αντίστοιχη κωδικοποίηση ενός πλήκτρου του σε ***hexadecimal***.

c. Σκοπός του αποκωδικοποιητή είναι να παράγει την 6-bit δυαδική αναπαράσταση του συμβόλου του οποίου το πλήκτρο πατήθηκε.

d. Αν δεν πατηθεί κάποιο νούμερο ο αποκωδικοποιητής θα παράγει μία τιμή η οποία θα υποδεικνύει σφάλμα ,στην ***default*** κατάστασή του .

e. Τέλος η ***always*** περιέχει τον scan κωδικό(***scan_code***) διότι είναι η είσοδός μας και ο αποκωδικοποιητής μας είναι συνδυαστικό κύκλωμα.

```
module Dec8_to_6_bit(
output reg [5:0] binary     ,
input      [7:0] scan_code   );

      always @(scan_code)
                case(scan_code)
                        8'h45 : binary<=6'b000000; //Binary == 0

                        8'h70 : binary<=6'b000000; //Binary == 0

                        8'h16 : binary<=6'b000001; //Binary == 1

                        8'h69 : binary<=6'b000001; //Binary == 1

                        8'h1E : binary<=6'b000010; //Binary == 2

                        8'h72 : binary<=6'b000010; //Binary == 2

                        8'h26 : binary<=6'b000011; //Binary == 3

                        8'h7A : binary<=6'b000011; //Binary == 3

                        8'h25 : binary<=6'b000100; //Binary == 4

                        8'h6B : binary<=6'b000100; //Binary == 4

                        8'h2E : binary<=6'b000101; //Binary == 5

                        8'h73 : binary<=6'b000101; //Binary == 5

                        8'h36 : binary<=6'b000110; //Binary == 6

                        8'h74 : binary<=6'b000110; //Binary == 6

                        8'h3D : binary<=6'b000111; //Binary == 7

                        8'h6C : binary<=6'b000111; //Binary == 7

                        8'h3E : binary<=6'b001000; //Binary == 8

                        8'h75 : binary<=6'b001000; //Binary == 8

                        8'h46 : binary<=6'b001001; //Binary == 9

                        8'h7D : binary<=6'b001001; //Binary == 9

                        default: binary<=6'b100000; //Binary is not number
                endcase
endmodule
```

Module πολυπλέκτη 2-σε-1

Ο πολυπλέκτης μας έχει ως εισόδους :

i. Την έξοδο του module του αποκωδικοποιητή πρώτου επιπέδου .

ii. Την έξοδο του module της πρόσθεσης/αφαίρεσης .

iii. Ένα καταχωρητή που καθορίζει ποια από τις παραπάνω τιμές θα οδηγηθεί στην έξοδό του .

iv. Η τιμή του καταχωρητή ***state*** εξαρτάται από το ***Finite State Machine*** του ***Top-level Module*** μας .

```
module Mux_2to_1(
output [5:0] out                    ,
input        state                  ,
input  [5:0] from_dec , from_adder  );

assign out = (state) ? from_dec : from_adder;

endmodule
```

Module αποκωδικοποιητή δεύτερου επιπέδου

a. Ο αποκωδικοποιητής δεύτερου επιπέδου έχει ως είσοδο την έξοδο του πολυπλέκτη 2-σε-1.

b. Η έξοδός , που παράγει , οδηγείται σε κάποιο ***7-Segment*** και είναι καθαρά θέμα του ***Finite State Machine*** κάθε φορά ποιο θα είναι .

c. Ο αποκωδικοποιητής δευτέρου επιπέδου αποκωδικοποιεί τις δυαδικές τιμές που αναπαριστούν τους δεκαδικούς αριθμούς [-9,18] και η έξοδός του είναι η δυαδική τιμή που αναπαριστά κάποιο δεκαδικό νούμερο στο εύρος [0-9] και οδηγείται σε ένα **7-Segment** .

d. Αν γίνει λάθος , το ***7-Segment*** θα πάρει μία τιμή από το default κομμάτι της case που θα δείξει ένα E ώστε να καταλάβουμε ότι είναι λάθος πλήκτρο.

e. Το 7-Segment **θα δείξει** μία τιμή από 0-9 , μόνο όσον αφορά το κομμάτι των τελεστέων . Η τιμή που θα του δοθεί θα είναι μία 7-bit ποσότητα που θα υποδεικνύει ποιά κομμάτια του ***7-Segment*** θα ανάψουν.

```
module Dec_to_7Seg(
output reg [6:0] display2 ,
input      [5:0] num                );

  always @(num)

      case(num)
              6'b000000:  display2 <= 7'b1000000;  //Result == 0

              6'b000001:  display2 <= 7'b1111001;   //Result == 1

              6'b000010:  display2 <= 7'b0100100;   //Result == 2

              6'b000011:  display2 <= 7'b0110000;   //Result == 3

              6'b000100:  display2 <= 7'b0011001;   //Result == 4

              6'b000101:  display2 <= 7'b0010010;   //Result == 5

              6'b000110:  display2 <= 7'b0000010;   //Result == 6

              6'b000111:  display2 <= 7'b1011000;   //Result == 7

              6'b001000:  display2 <= 7'b0000000;   //Result == 8

              6'b001001:  display2 <= 7'b0010000;   //Result == 9

              6'b001010:  display2 <= 7'b1000000;   //Result == 10

              6'b001011:  display2 <= 7'b1111001;   //Result == 11

              6'b001100:  display2 <= 7'b0100100;   //Result == 12

              6'b001101:  display2 <= 7'b0110000;   //Result == 13

              6'b001110:  display2 <= 7'b0011001;   //Result == 14

              6'b001111:  display2 <= 7'b0010010;   //Result == 15

              6'b010000:  display2 <= 7'b0000010;   //Result == 16

              6'b010001:  display2 <= 7'b1011000;   //Result == 17

              6'b010010:  display2 <= 7'b0000000;   //Result == 18

              6'b111111:  display2 <= 7'b1111001;   //Result == -1

              6'b111110:  display2 <= 7'b0100100;   //Result == -2

              6'b111101:  display2 <= 7'b0110000;   //Result == -3

              6'b111100:  display2 <= 7'b0011001;   //Result == -4

              6'b111011:  display2 <= 7'b0010010;   //Result == -5

              6'b111010:  display2 <= 7'b0000010;   //Result == -6

              6'b111001:  display2 <= 7'b1011000;   //Result == -7

              6'b111000:  display2 <= 7'b0000000;   //Result == -8

              6'b110111:  display2 <= 7'b0010000;   //Result == -9

              default:    display2 <= 7'b000110;    //7-Segment error

       endcase

endmodule
```

Κώδικας clock για αρχείο .sdc

Στο αρχείο ***.sdc*** αντικαθιστούμε το όνομα του ρολογιού με το όνομα της μεταβλητής μας που θέλουμε να είναι το εσωτερικό μας ρολόι , σε αυτή την περίπτωση είναι ***clk***.

```
create_clock -period 20.000 -name clk
derive_clocks -period 20.000
derive_clock_uncertainty
```

Κώδικας Calculator για σύνθεση

Παράμετροι του Calculator

Το ***Top-level Module*** μας έχει ως εισόδους :

a. Το εσωτερικό ρολόι του FPGA , ***clk*** .

b. Την ασύγχρονη αρχικοποίηση των μεταβλητών του συστήματός μας , ανάλογα με την τιμή του , ***reset*** .

c. Το ρολόι του πληκτρολογίου μας , το οποίο είναι περίπου 2.500 φορές πιο αργό από το εσωτερικό μας ρολόι , ***keyb_clk*** .

d. Η γραμμή μεταφοράς των δεδομένων από το πληκτρολόγιο , ***keyb_data*** .

Η έξοδος του κυκλώματός μας είναι τα 6 ***7-Segments*** τα οποία θα δείχνουν τα σύμβολα , τους αριθμητικούς τελεστές και τα αποτελέσματα .

```
module Calculator_icsd12012(
input clk , reset  ,
inout keyb_clk     ,
input keyb_data    ,
output reg [6:0]  LED0, LED1 ,LED2 , LED3  , LED4 , LED5  );
```

Δήλωση μεταβλητών ,Module Instantiations και State Machine states

```
parameter WAIT_OPERAND_1=3'b000 ,WAIT_OPERATOR=3'b001,WAIT_OPERAND_2=3'b010 ,WAIT_EQ=3'b011 ,WAIT_F0=3'b101, AFTER_F0=3'b110;

//Kataxwrites kai wires

reg         math_symbol  ,  is_error , mux_state ;
reg  [2:0]  current_state,  state    ;
reg  [5:0]  reg_A        ,  reg_B    ,kbclk_shift;
reg  [7:0]  scan_code;
reg  [10:0] kbdat_shift;

wire [5:0]  result_mux   ,result_math;
wire [5:0]  num , A , B ;
wire [6:0]  result2;

//module Instantiations

Dec8_to_6_bit Dec1( num , scan_code );

Add_Subtract  A0( result_math , math_symbol , A, B);

Mux_2to_1     M0( result_mux , mux_state , num , result_math);

Dec_to_7Seg   Dec2( result2 , result_mux);
```

Περιγραφή λειτουργίας shift register δειγματοληψίας του ***keyb_clk***

Ο καταχωρητής δειγματοληψίας του ρολογιού του πληκτρολογίου στο ασύγχρονο ***reset*** αρχικοποιείται , ειδάλλως δέχεται με δεξιά ολίσθηση τις τιμές του ρολογιού .

```
if(reset)
       kbclk_shift  <= 6'b000000; //Arxikopoihsh kataxwriti deigmatolipsias tou rologiou tou keyboard
 else
       kbclk_shift  <= {keyb_clk , kbclk_shift [5:1] }; //shifting pros ta deksia tis neas timis toy rologiou ston kataxwriti
                                                        //deigmatolipsias tou
```

Περιγραφή λειτουργίας shift register δειγματοληψίας του ***keyb_data***

Ο καταχωρητής δειγματοληψίας του ***keyb_data*** στο ασύγχρονο ***reset*** αρχικοποιείται .

Αν είναι γεμάτος , αφού γεμίζει με δεξιά ολίσθηση , η πρώτη του θέση [0] θα πάρει τιμή τελευταία , τότε θα επαναρχικοποιηθεί και την ίδια χρονική στιγμή θα πέσει το ρολόι του πληκτρολογίου ***keyb_clk*** στην τιμή 0 που θα το οδηγούμε εμείς.

Ειδάλλως , αν ο ***shift register*** δειγματοληψίας του πληκτρολογίου έχει την τιμή που μας δείχνει πως μπορούμε να κάνουμε δειγματοληψία , με δεξιά ολίσθηση προσθέτουμε την νέα τιμή του ***keyb_data*** στον ***shift register.***

```
if(reset)
        kbdat_shift  <= 11'b11111111111;//Arxikopoihsh kataxwriti deigmatolipsias tis grammis metaforas dedomenwn
                                        //keyb_data tou keyboard
  else if(!kbdat_shift[0] )
        kbdat_shift  <= 11'b11111111111;//Tin xroniki stigmi poy o kataxwritis tha einai gematos kai to LSB
                                        //tha einai 0 tha epana-arikopoieitai
  else if(kbclk_shift==6'b000111)
        kbdat_shift  <= {keyb_data,kbdat_shift [10:1]};//Shifting pros ta deksia tis neas timis tou keyb_data ston kataxwriti
                                                       // digmatolipsias tou
```

Περιγραφή λειτουργίας του inout ***keyb_clk***

Το keyb_clk μας δίνει την τιμή του ρολογιού του πληκτρολογίου, άρα **λειτουργεί σαν έξοδος** διότι αυτή η τιμή οδηγείται στο κύκλωμά μας. Όταν δειγματοληπτήσουμε 11 bits θέλουμε να σταματήσουμε να δεχόμαστε τιμές μέχρι να είμαστε έτοιμοι να δεχτούμε την επόμενη τιμή .

Για να σταματήσουμε λοιπόν να δεχόμαστε τιμές πρέπει να ρίξουμε το ***keyb_clk*** στο 0 . Η τιμή που δειγματοληπτούμε από 000111 θα γίνει 000000 , ουσιαστικά την αρχικοποιούμε .

Εφόσον το οδηγούμε στο 0 ***λειτουργεί σαν είσοδος*** .

```
assign keyb_clk = (!kbdat_shift[0]) ? 1'b0 : 1'bz;//To keyb_clk einai inout
                                                  //Epomenws otan exoume deigmatoliptisei mia 11da tha odigeitai sto 0 (eisodos)
                                                  //Sto hiz  odigei to kbclk_shift
```

Περιγραφή Λειτουργίας του Finite State Machine

Ασύγχρονο Reset

Αρχικά περιγράφουμε την λειτουργία του ***reset*** , κατά την οποία αρχικοποιούμε τους καταχωρητές μας και θέτουμε τα 6 **7-Segment** σβηστά .

Η πρώτη μας κατάσταση είναι η ***WAIT_OPERAND_1*** και την θέτουμε στους καταχωρητές της παρούσας(***current_state***) και της επόμενης (***state***) κατάστασης.

Ο καταχωρητής mux_state καθορίζει αν ο πολυπλέκτης μας θα οδηγήσει στην έξοδό του την έξοδο του αποκωδικοποιητή πρώτου επιπέδου (***mux_state==1'b1***) ή το αποτέλεσμα της πράξης των 2 αριθμών (***mux_state==1'b0***).

Τέλος θέτουμε hiZ (ανοιχτοκύκλωμα) τους καταχωρητές αποθήκευσης της δυαδικής τιμής των 2 προσθετέων όρων, τον καταχωρητή αποθήκευσης των 8-bit της 16δικής τιμής που μας δείχνει ποιο πλήκτρο πατήθηκε (***scan_code***) και ο καταχωρητής ελέγχου λάθους (***is_error***).

```
if(reset)begin//Arxikopoihsh kataxwritwn

       state <= WAIT_OPERAND_1;//Ousiastika einai h epomenh katastash tou FSM  kathe fora
       current_state <= state; //Parousa katastash gia elegxo se WAIT_f0

       mux_state <= 1'b1;//Me 1 pernaei apotelesma Dec8_to_6  , me 0 to teliko apotelesma tis praksis ,ston Dec_to_7Seg

       LED0 <= 7'b1111111;//Ola ta LED  arxika einai svista
       LED1 <= 7'b1111111;
       LED2 <= 7'b1111111;
       LED3 <= 7'b1111111;
       LED4 <= 7'b1111111;
       LED5 <= 7'b1111111;

       reg_A<=6'bz; //Oi kataxwrites mas arxika leitourgoun , ws anoixtokiklwma , se high-z state
       reg_B<=6'bz;

       scan_code<=8'bz;
       is_error<=1'bz;
 end
```

Κατάσταση WAIT_OPERAND_1

- Κάθε φορά που θα δειγματοληπτούμε ένα καινούργιο 11-bit κωδικό , και ανάλογα την τιμή της επόμενης κατάστασης(***state***) θα τρέχει μία διαφορετική κατάσταση του ***FSM***.

Αν το ***state==WAIT_OPERAND_1*** τότε :

Αποθηκεύουμε τον 8bit hexadecimal κωδικό του πλήκτρου που πατήθηκε και ορίζουμε ως επόμενη κατάσταση την ***WAIT_F0*** και παρούσα την ***WAIT_OPERAND_1***.

```
else if (!kbdat_shift[0] )//Kathe fora poy tha exoume deigmatoliptisei ena neo 11-bit kwdiko apo to pliktrologio

    case(state)//Elegxoume thn epomenh katastash (state) pou tha exoume kathe fora

         WAIT_OPERAND_1: begin//Apothikeuoume ta 8-bit tou hexademical kwdikou sto scan_code
                             //Orizoume tin epomeni kai tin parousa katastasi

              scan_code <= kbdat_shift[8:1];

              current_state <= WAIT_OPERAND_1;
              state <= WAIT_F0;
         end
```

Κατάσταση WAIT_OPERATOR

Αποθηκεύουμε τον 8bit hexadecimal κωδικό του πλήκτρου που πατήθηκε και ορίζουμε ως επόμενη κατάσταση την ***WAIT_F0*** και παρούσα την ***WAIT_OPERATOR***.

```
WAIT_OPERATOR: begin//Paromoiws opws kai stis WAIT_OPERAND_1,WAIT_OPERAND_2,WAIT_EQ

     scan_code <= kbdat_shift[8:1];

     current_state <= WAIT_OPERATOR;
     state<= WAIT_F0;
 end
```

Κατάσταση WAIT_OPERAND_2

Αποθηκεύουμε τον 8bit hexadecimal κωδικό του πλήκτρου που πατήθηκε και ορίζουμε ως επόμενη κατάσταση την ***WAIT_F0*** και παρούσα την ***WAIT_OPERAND_2***.

```
WAIT_OPERAND_2: begin//Paromoiws opws kai stis WAIT_OPERAND_1,WAIT_OPERATOR,WAIT_EQ
        scan_code  <= kbdat_shift[8:1];

        current_state <= WAIT_OPERAND_2;
        state <= WAIT_F0;
end
```

Κατάσταση WAIT_EQUAL

Αποθηκεύουμε τον 8bit hexadecimal κωδικό του πλήκτρου που πατήθηκε , ορίζουμε ως επόμενη κατάσταση την ***WAIT_F0*** και παρούσα την ***WAIT_EQUAL*** και ορίζουμε ***mux_state=1'b0*** ώστε ο πολυπλέκτης να οδηγήσει στην έξοδό του το αποτέλεσμα της πράξης .

```
WAIT_EQ: begin//Paromoiws opws kai stis WAIT_OPERAND_1,WAIT_OPERATOR,WAIT_OPERAND_2
             //To mux_state allazei wste ,en telei, na bgei to apotelesma tis praksis apo ton poliplekti
             //kai na perastei sto DEC_to_7Seg

        mux_state <= 1'b0;
        scan_code <= kbdat_shift[8:1];

        current_state <= WAIT_EQ;
        state <= WAIT_F0;
end
```

Κατάσταση WAIT_F0

- Η κατάσταση αυτή είναι πολύ σημαντική διότι εδώ μπορούμε να κάνουμε τους ελέγχους και τους ορισμούς τιμών που θέλουμε με βάση το ***scan_code*** που έχουμε αποθηκεύσει στις καταστάσεις ***WAIT_OPERAND_1*** , ***WAIT_OPERATOR*** , ***WAIT_OPERAND_2*** και ***WAIT_EQUAL***.

- Ταυτόχρονα περιμένουμε μέχρι να δεχτούμε τον κωδικό F0 ώστε να μεταβούμε στην ***AFTER_F0***.

- Σε αυτή την κατάσταση *δεν αποθηκεύουμε καινούργια τιμή* στο **scan_code** όμως περιμένουμε το F0.

- Συνεπώς σε μία συνθήκη ***case*** με παράμετρο την παρούσα κατάσταση κάνουμε διαφορετικές ενέργειες κάθε φορά :

Current_State == WAIT_OPERAND_1

  - Θέτουμε τον καταχωρητή ***reg_A*** με την έξοδο του αποκωδικοποιητή πρώτου επιπέδου.

  - Το ***7-Segment*** απεικόνισης του πρώτου αριθμού θα πάρει το αποτέλεσμα του αποκωδικοποιητή δεύτερου επιπέδου .

  - Αν δεν πατήθηκε αριθμός τότε το ***num*** θα έχει μία τιμή που θα δείχνει λάθος και τελικά θα φανεί στο ***7-Segment*** μας και ο καταχωρητής λάθους θα πάρει την ανάλογη τιμή.

  - Η τελευταία if είναι απαραίτητη στην περιγραφή του ***Calculator*** διότι όσο είναι πατημένο το πλήκτρο τότε θα

στέλνεται συνεχώς ο ίδιος 11-bit κωδικός(που πρωτοπήραμε σε μία από τις καταστάσεις που μετά μεταβαίνουν στο ***WAIT_F0***) και αυτό θα συμβεί στην κατάσταση ***Wait_F0*** .

- Όταν όμως αφεθεί το πλήκτρο θα σταλεί ο κωδικός F0 άρα τότε πρέπει να έχουμε εμφανίσει το αποτέλεσμα του ***7-Segment*** και ουσιαστικά είναι η στιγμή που ορίζουμε την επόμενη κατάσταση ***state==AFTER_F0***.

```
WAIT_F0: begin//Analoga me tin parousa katastasi kanoume diaforetikes energeies

    case(current_state)

            WAIT_OPERAND_1: begin//Apothikeuoume ton arithmo se ena kataxwriti kai to apotelesma tis telikis
                                 //apokwdikopoihshs sto LED.
                reg_A<=num;
                LED5<=result2;

                if(num[5])is_error<=1'b1;//An to apotelesma einai 100000 tote den einai kati pou theloume ara lathos

                if(kbdat_shift[8:1]==8'hF0)state<=AFTER_F0;//Otan to koumbi tou keyboard tha afethei , tha epistrafei F0
                                                           //Mexri tote omws theloume tin idia katastasi

             end
```

Current_State == WAIT_OPERATOR

Σε αυτό το σημείο δημιουργούμε ένα μικρό αποκωδικοποιητή και ελέγχουμε αν ο ***scan_code*** είναι το σύμβολο (+ ή -) και ορίζουμε το ***7-Segment*** που το εμφανίζει με την κατάλληλη τιμή όπως και τον καταχωρητή που η τιμή του εξαρτάται από το σύμβολο.

Σε περίπτωση λάθους οδηγούμε τον καταχωρητή ***is_error*** στην κατάλληλη τιμή , το ***7-Segment*** θα πάρει την ανάλογη τιμή για εμφάνιση λάθους και ο καταχωρητής συμβόλου θα είναι ανοιχτοκύκλωμα.

Τέλος έχουμε την ίδια if που περιγράψαμε στο Current_State == WAIT_OPERAND_1.

```
WAIT_OPERATOR: begin//Elegxoume se ksexwristo apokwdikopoihth to simbolo tis praksis

      case(scan_code)
            8'h7B : begin
                  math_symbol <= 1'b1; //Symbol is -
                  LED4<=7'b0111111;

            end
            8'h79 : begin
                  math_symbol <= 1'b0; //Symbol is +
                  LED4<=7'b0001111;

            end
            default: begin
                  math_symbol <= 1'bz; //Symbol Unknown
                  LED4<=7'b000110;
                  is_error<=1'b1;
            end
         endcase

       if(kbdat_shift[8:1]==8'hF0)state<=AFTER_F0;//Otan to koumbi tou keyboard tha afethei , tha epistrafei F0
                                                  //Mexri tote omws theloume tin idia katastasi
end
```

Current_State == WAIT_OPERAND_2

Η υλοποίηση του κώδικα είναι παρόμοιας λογικής με την Current_State ==WAIT_OPERAND_1.

```
WAIT_OPERAND_2: begin//Apothikeuoume ton arithmo se ena kataxwriti kai to apotelesma tis telikis
                     //apokwdikopoihshs sto LED.

    reg_B<=num;
    LED3 <= result2;

    if(num[5])is_error<=1'b1;//An to apotelesma einai 100000 tote einai lathos

    if(kbdat_shift[8:1]==8'hF0)state<=AFTER_F0;//Otan to koumbi tou keyboard tha afethei , tha epistrafei
                                               //Mexri tote omws theloume tin idia katastasi
end
```

Current_State == WAIT_EQUAL

Αρχικά ελέγχουμε αν ο κωδικός που δειγματοληπτήσαμε αντιστοιχεί στο πλήκτρο (=).

Αν όντως αντιστοιχεί , τότε με ένα μικρό αποκωδικοποιητή και με παράμετρο στην ***casex*** το αποτέλεσμα της πράξης των δύο αριθμών , οδηγούμε στο ***7-Segment*** που θα πάρει είτε 1 (π.χ. 13) , - (π.χ. -5) ή τίποτα (π.χ. 8) την ανάλογη 7-bit τιμή .

Ένα ***7-Segment*** θα δείχνει το σύμβολο της ισότητας (=) και ένα άλλο το αποτέλεσμα του αποκωδικοποιητή δεύτερου επιπέδου , το οποίο θα είναι τιμή [0-9].

Αν όμως δεν αντιστοιχεί τότε θα εμφανίσουμε στο ανάλογο ***7-Segment*** μήνυμα λάθους και θα οδηγήσουμε τον καταχωρητή λάθους στην κατάλληλη τιμή.

Τέλος έχουμε την ίδια ***if*** που περιγράψαμε στο Current_State == WAIT_OPERAND_1.

```
WAIT_EQ: begin//Elegxoume scan_code gia na emfanisoume = kai apotelesma  ,alliws minima lathous

    if(scan_code == 8'h55 ) begin

        casex(result_math)//Emfanisi LED1 me vasi apotelesmatos praksis

             6'b0011xx:LED1<=7'b1111001;   //Symbol is 1

             6'b00101x:LED1<=7'b1111001;   //Symbol is 1

             6'b01xxxx:LED1<=7'b1111001;   //Symbol is 1

             6'b1xxxxx:LED1<=7'b0111111;   //Symbol is -

             default:  LED1<=7'b1111111;   //LED is Deactivated

         endcase

         LED2 <= 7'b0110111;//Emfanisi =
         LED0 <= result2;//Emfanisi 0-9
    end
    else begin//An to scan_code den einai to =

         LED2<=7'b000110;//Minima lathous
         is_error<=1'b1; //Kataxwritis tha exei tin timi pou ipodikniei Error
    end

    if(kbdat_shift[8:1]==8'hF0)state<=AFTER_F0;//Otan to koumbi tou keyboard tha afethei , tha epistrafei
                                               //Mexri tote omws theloume tin idia katastasi

end
```

Κατάσταση AFTER_F0

Σε αυτό το σημείο της περιγραφής του ***Calculator*** ενώ έχουμε πατήσει ένα πλήκτρο το αφήνουμε , επομένως :

Αν ο καταχωρητής λάθους έχει τιμή που υποδεικνύει λάθος τότε η επόμενη κατάσταση θα είναι η παρούσα(***current_state***) που είχε οριστεί πριν την μετάβαση στο **WAIT_F0**.

Αν δεν υπάρχει λάθος τότε η επόμενη κατάσταση θα είναι η επόμενη της παρούσας , που είχε οριστεί πριν την μετάβαση στο **WAIT_F0**.Ταυτόχρονα αρχικοποιούμε ή **ουσιαστικά** απενεργοποιούμε τον καταχωρητή λάθους θέτοντάς τον hiZ (ανοιχτοκύκλωμα).

```
AFTER_F0: begin//Efoson afisame to koumbi toy keyboard tha erthoume se auti thn katastasi

    if(is_error)begin//An exoume sfalma tote epistrefoume stin parousa katastasi i opoia tha einai
                    //Eite WAIT_OPERAND_1,WAIT_OPERATOR,WAIT_OPERAND_2,WAIT_EQ

       state<=current_state;
       is_error<=1'bz;
    end
    else begin//Eidallws tha pame stin epomeni apo tin parousa katastasi

       state<=current_state + 1'b1;
       is_error<=1'bz;
    end
end
```

Κατάσταση Default

- Στην κατάσταση ***Default*** , ενώ έχουμε εμφανίσει τελεστές , σύμβολά και αποτέλεσμα , θα μπούμε σε αυτή την κατάσταση όταν πατηθεί ένα πλήκτρο για εισαγωγή νέου αριθμού σε καινούργια πράξη , σαν να είμασταν στην ***WAIT_OPERAND_1*****,** με μερικές διαφορές .

  Αρχικοποιούμε όλα τα ***7-Segments*** , εκτός αυτού που εμφανίζει το νούμερο του πρώτου τελεστέου , την κατάσταση του πολυπλέκτη και τους καταχωρητές αποθήκευσης των τελεστέων .

  Οι 2 καταχωρητές της παρούσας(***current_state***) και της επόμενης κατάστασης(***state***) του ***State Machine*** θα πάρουν τις ίδιες τιμές που είχαν πάρει και την κατάσταση ***WAIT_OPERAND_1***.

  Με τον καταχωρητή ***scan_code*** δειγματοληπτούμε τα 8-bit που μας δείχνει ποιο πλήκτρο πατήθηκε μέσα από την αποκωδικοποίηση του .

```
default: begin //Arxikopoioume opws reset ektos apo LED5 kai diavazoume  neo scan_code

     LED0 <= 7'b1111111;//Svinoun ksana ola ta LED
     LED1 <= 7'b1111111;
     LED2 <= 7'b1111111;
     LED3 <= 7'b1111111;
     LED4 <= 7'b1111111;

    state <= WAIT_F0;//H epomenh katastasi tha einai i Wait_F0 afou tha diabasoume scan_code

    current_state <= WAIT_OPERAND_1;//Ousiastika eimaste stin Wait_Operand_1

    scan_code <= kbdat_shift[8:1];

    mux_state <= 1'b1;//Dinoume tis arxikes times ksana stous kataxwrites mas
    reg_A<=6'bz;
    reg_B<=6'bz;
    is_error<=1'bz;

end
```

Κώδικας Calculator test bench για εξομοίωση

- Στο ***test_bench*** έχουμε φτιάξει σενάρια στα οποία εξομοιώνουμε την λειτουργία του πληκτρολογίου και δίνουμε τιμές στο ***keyb_data*** .

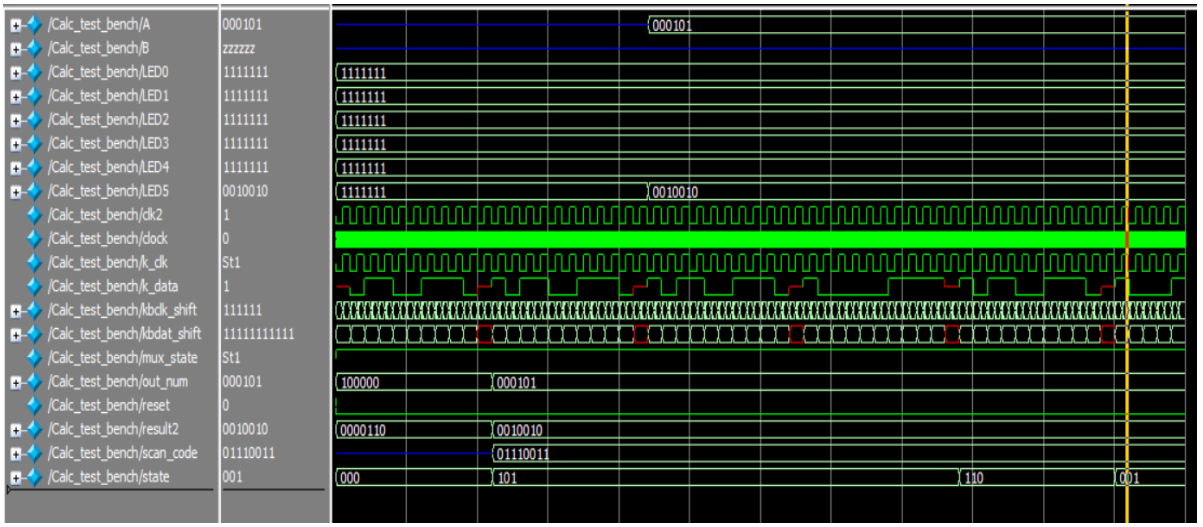
- Με βάση αυτά τα τεστ παραμετροποιήσαμε τον κώδικά μας ώστε να λειτουργεί τελικά σωστά στη σύνθεση στο ***Altera Cyclone IV***.

- Επειδή η παρουσίαση όλων των εξομοιώσεων θα ήταν πρακτικά πολύ δύσκολη , θεωρώντας ότι πρέπει ταυτόχρονα να είναι και κατανοητή , θα παρουσιάσουμε κάποια κομμάτια(σενάρια) τα οποία θα δείξουμε ότι έχουν να κάνουν άμεσα με την παραμετροποίηση του κώδικά μας .

Σενάριο ελέγχου σωστής δειγματοληψίας και ορισμού τιμής για 7-Segments

- Ξεκινώντας την εξομοίωσή μας θα δώσουμε 3 φορές την ίδια 11-δα bits πριν τον κωδικό F0 που αντιστοιχεί στο πλήκτρο με αριθμό 5. Μετά το F0 θα ξαναστείλουμε τον κωδικό όπως γίνεται στην πραγματικότητα από το πληκτρολόγιο.

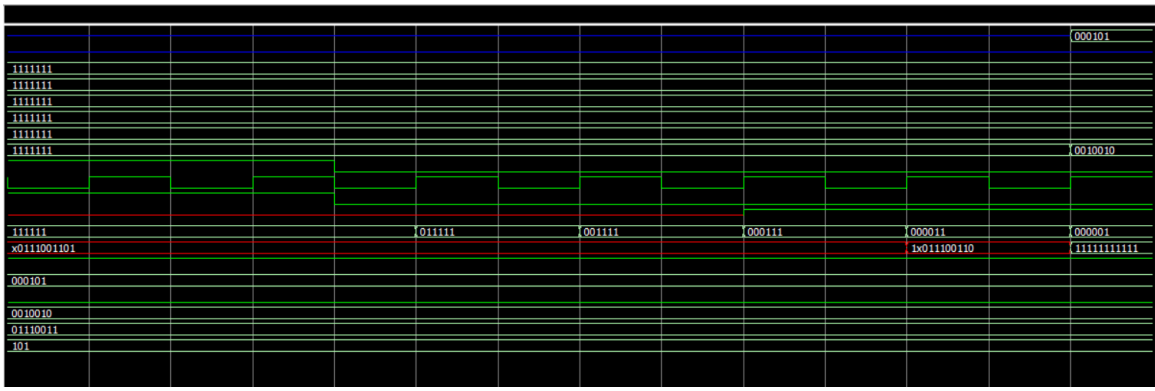
- Πρακτικά αυτό σημαίνει ότι πατάμε το πλήκτρο και το κρατάμε για ένα διάστημα πριν το αφήσουμε .Δεν θα αναλύσουμε το σενάριο αυτό όμως .Θα δείξουμε ότι παίρνει την τιμή και ορίζει ανάλογα το σωστό ***7-Segment*** την κατάλληλη στιγμή.



1. Αρχικά με το reset αρχικοποιούμε τους καταχωρητές μας και τα ***7-Segments*** μας .Αμέσως μετά δειγματοληπτούμε το πληκτρολόγιο με βάση την εξομοίωσή μας .

2. Επειδή το parity bit το έχουμε ορίσει 1'bx όταν θα το δειγματοληπτήσουμε θα κοκκινίζει το τετράγωνο με το ***kbdat_shift*** .

3. Υπάρχει και ένα τετράγωνο , όπου έχουμε δειγματοληπτήσει μία 11-δα bits , πολύ μικρή σε χρόνο και είναι αμέσως μετά .



4. Με βάση τον κώδικα στην κατάσταση ***WAIT_OPERAND_1***(000) δειγματοληπτούμε το ***scan_code*** και στην ***WAIT_F0***(101) μπορούμε να την χρησιμοποιήσουμε.

5. Στο τέλος της πρώτης ***WAIT_F0*** έχουμε και την εμφάνιση της τιμής στο ***7-Segment***.

6. Μετά από 3 δειγματοληψίες 11-bit κωδικών στην κατάσταση ***Wait_F0*** ,στην τρίτη δειγματοληψία (***F0***) , θα πάμε στην ***AFTER_F0***(110) και η επόμενη κατάσταση είναι η ***WAIT_OPERATOR***(001).

```
//NUMBER 5 ------------OPERAND 1 -->WAIT F0
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #200  k_data<=1'b0;
  #200  k_data<=1'b1;
  #300  k_data<=1'b0;
  #100  k_data<=1'bx;
  #100  k_data<=1'b1;
//NUMBER 5 ------------OPERAND 1 -->WAIT F0
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #200  k_data<=1'b0;
  #200  k_data<=1'b1;
  #300  k_data<=1'b0;
  #100  k_data<=1'bx;
  #100  k_data<=1'b1;
//NUMBER 5 ------------OPERAND 1-->WAIT F0
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #200  k_data<=1'b0;
  #200  k_data<=1'b1;
  #300  k_data<=1'b0;
  #100  k_data<=1'bx;
  #100  k_data<=1'b1;
//F0 ------------WAIT FO --> AFTER F0
  #100 k_data<=1'b0;
  #500 k_data<=1'b1;
  #400 k_data<=1'bx;
  #100 k_data<=1'b1;
//NUMBER 5 (SAME DATA) ---------AFTER F0 -->WAIT_OPERATOR
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #200  k_data<=1'b0;
  #200  k_data<=1'b1;
  #300  k_data<=1'b0;
  #100  k_data<=1'bx;
  #100  k_data<=1'b1;
```

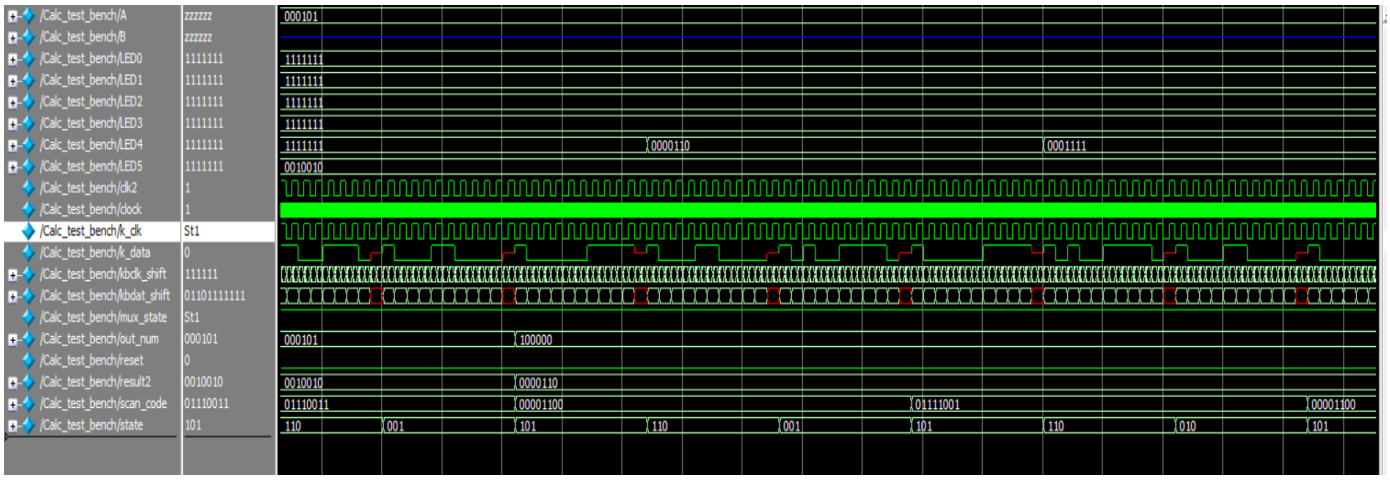
Σενάριο παραμονής στην Wait_F0 όσο κρατάω πατημένο το πλήκτρο

- Η παραμονή στην ίδια κατάσταση είναι σημαντική και απαραίτητη καθώς αν δεν περιγραφεί τότε το ***State Machine*** δεν θα περιμένει να αφήσουμε το πλήκτρο και θα πάει στην επόμενη κατάσταση και θα εμφανίσει μήνυμα σφάλματος και αυτό μπορεί να συμβεί σε οποιαδήποτε κατάσταση του ***FSM***.
- Ο κώδικας της εν λόγω περιγραφής παρουσιάζεται στην κάθε περίπτωση της case της ***Wait_F0***.

- Στην εξομοίωση παραπάνω φαίνεται το εν λόγω σενάριο.

Σενάριο πατήματος λάθους πλήκτρου

- Σε συνέχεια του κώδικα που παρουσιάσαμε στα 2 παραπάνω σενάρια , σε αυτό το σενάριο , θα εξομοιώσουμε το πάτημα ενός πλήκτρου που δεν αντιστοιχεί σε αριθμό ή σύμβολο στην κατάσταση ***WAIT_OPERATOR***.
- Στην ***WAIT_OPERATOR***(001) δειγματοληπτούμε τον ***scan_code*** ο οποίος φαίνεται την στιγμή που είμαστε στην ***WAIT_F0*** το οποίο συμβαίνει σε κάθε μετάβαση σε ***WAIT_F0***.
- Στο τέλος της ***Wait_F0*** θα εμφανίσει το λάθος και αφού θα περάσει η ***AFTER_F0*** θα επαναληφθεί η ***WAIT_OPERATOR*** μέχρι να πατηθεί σωστό πλήκτρο συμβόλου.
- Σε αυτή την περίπτωση το σύμβολο(+).



- Ο κώδικας για αυτό το σενάριο παρουσιάζεται παρακάτω:

```
//Tixaia timi (Den antixstoixei se arithmo 0-9) Skopimo LATHOS WAIT_OPERATOR-->WAIT_F0
  #100  k_data<=1'b0;
  #100  k_data<=1'b0;
  #200  k_data<=1'b1;
  #200  k_data<=1'b0;
  #300  k_data<=1'b0;
  #100  k_data<=1'bx;
  #100  k_data<=1'b1;
//F0 ------------WAIT FO --> AFTER F0
  #100 k_data<=1'b0;
  #500 k_data<=1'b1;
  #400 k_data<=1'bx;
  #100 k_data<=1'b1;
//Tixaia timi (Den antixstoixei se arithmo 0-9) ------------  Skopimo LATHOS AFTER_F0 --> WAIT_OPERATOR (LOGW LATHOUS)
  #100  k_data<=1'b0;
  #100  k_data<=1'b0;
  #200  k_data<=1'b1;
  #200  k_data<=1'b0;
  #300  k_data<=1'b0;
  #100  k_data<=1'bx;
  #100  k_data<=1'b1;

//SYMBOL + ------------OPERATOR
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #100  k_data<=1'b0;
  #200  k_data<=1'b1;
  #400  k_data<=1'b0;
  #100  k_data<=1'bx;
  #100  k_data<=1'b1;
//F0 ------------WAIT FO --> AFTER F0
  #100 k_data<=1'b0;
  #500 k_data<=1'b1;
  #400 k_data<=1'bx;
  #100 k_data<=1'b1;
//SYMBOL +(SAME DATA) ------------AFTER F0 --> OPERAND2
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #100  k_data<=1'b0;
  #200  k_data<=1'b1;
  #400  k_data<=1'b0;
  #100  k_data<=1'bx;
  #100  k_data<=1'b1;
```

Σενάριο εμφάνισης σωστών αποτελεσμάτων και αρχικοποίησης

- Σε συνέχεια του σεναρίου που έχουμε ξεκινήσει θα εξομοιώσουμε το πάτημα του 5 ως δεύτερου τελεστή και μετά θα πατήσουμε το πλήκτρο(=) για να δούμε τα αποτελέσματά μας .

```
//NUMBER 5 ------------OPERAND 2 -->WAIT F0
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #200  k_data<=1'b0;
  #200  k_data<=1'b1;
  #300  k_data<=1'b0;
  #100  k_data<=1'bx;
  #100  k_data<=1'b1;
//F0 ------------WAIT FO --> AFTER F0
  #100 k_data<=1'b0;
  #500 k_data<=1'b1;
  #400 k_data<=1'bx;
  #100 k_data<=1'b1;
//NUMBER 5 (SAME DATA) ---------AFTER F0 -->WAIT_EQ
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #200  k_data<=1'b0;
  #200  k_data<=1'b1;
  #300  k_data<=1'b0;
  #100  k_data<=1'bx;
  #100  k_data<=1'b1;
//SYMBOL =-------WAIT EQ -----WAIT_FO
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #100  k_data<=1'b0;
  #100  k_data<=1'bx;
  #100  k_data<=1'b1;
//F0 ------------WAIT FO --> AFTER F0
  #100 k_data<=1'b0;
  #500 k_data<=1'b1;
  #400 k_data<=1'bx;
  #100 k_data<=1'b1;
//SYMBOL =------- AFTER_F0 --> DEFAULT
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
```

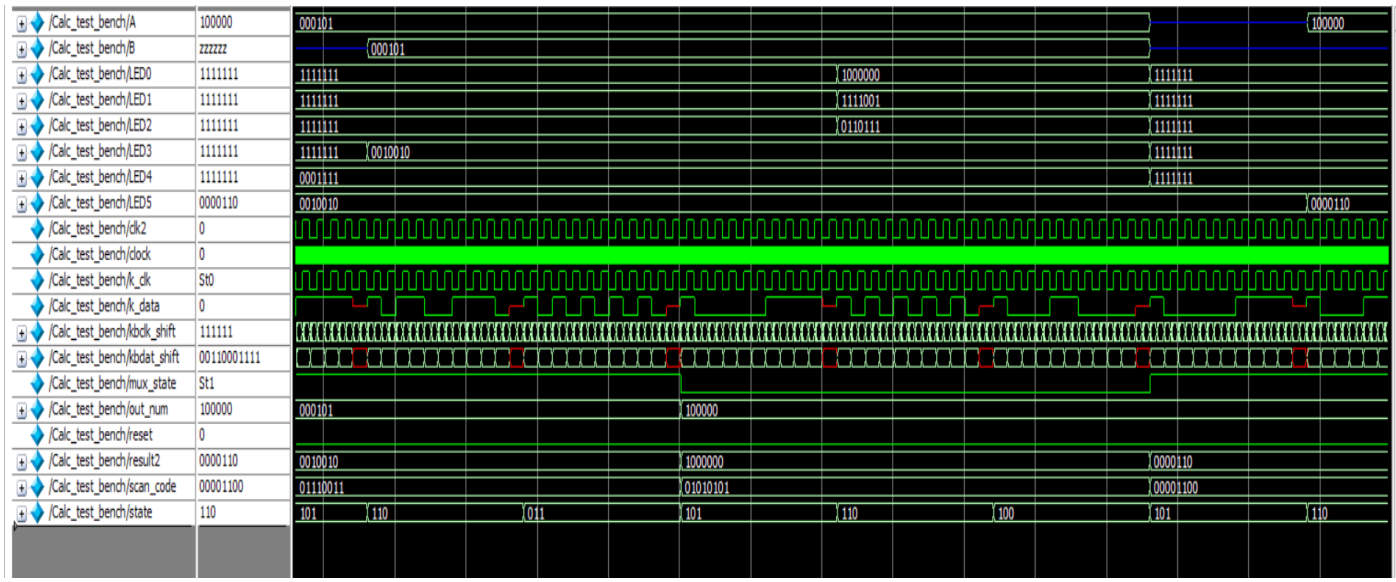
- Ο κώδικας ήταν πολύ μεγάλος για την φωτογραφία που τραβήξαμε όμως πριν το F0 και μετά έχουμε την ίδια 11δα από bits που στέλνουμε.

- Καθώς πατάμε το πλήκτρο 5 ,όπως αναλύσαμε σε σενάριό πιο πάνω , οδηγούμε την τιμή μας στον καταχωρητή B και το ***7-Segment*** τελικά θα δείξει την απεικόνιση του αριθμού .
- Όταν θα περάσουμε από την ***WAIT_EQ*** που θα δειγματοληπτήσουμε το ***scan_code*** , θα αλλάξει και το ***mux_state***.
- Στην ***WAIT_F0*** θα εμφανίσουμε το αποτέλεσμα της πράξης με βάση τους 2 αποκωδικοποιητές , ο αποκωδικοποιητής δευτέρου επιπέδου και ένας δικός μας, μαζί με το σύμβολο (=) .

- Η εξομοίωσή μας , αν δεν στέλναμε άλλη 11δα θα τελείωνε εδώ . Έστω όμως ότι θέλω να στείλω μία 11δα που υποδεικνύει το πάτημα του πλήκτρου 4.

- Όλα , πλην του πρώτου ***7-Segment*** θα σβήσουν , θα αρχικοποιηθούν οι καταχωρητές μας και όταν φτάσουμε στην ***DEFAULT***→***WAIT_F0*** ,ανάλογα το ***scan_code*** θα έχουμε και ανάλογη απεικόνιση.

```
//NUMBER 4 ------------OPERATOR -->WAIT F0
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #200  k_data<=1'b0;
  #100  k_data<=1'b1;
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #200  k_data<=1'b0;
  #100  k_data<=1'bx;
  #100  k_data<=1'b1;
//F0 ------------WAIT FO --> AFTER F0
  #100 k_data<=1'b0;
  #500 k_data<=1'b1;
  #400 k_data<=1'bx;
  #100 k_data<=1'b1;
//NUMBER 4 (SAME DATA) ---------AFTER F0 -->WAIT_OPERATOR
   #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #200  k_data<=1'b0;
  #100  k_data<=1'b1;
  #100  k_data<=1'b0;
  #100  k_data<=1'b1;
  #200  k_data<=1'b0;
  #100  k_data<=1'bx;
  #100  k_data<=1'b1;
```



Link για προβολή Βίντεο λειτουργίας και περιληπτικό RTL Design

https://www.youtube.com/watch?v=8hWS5wE1efs