Σχεδιασμός Συστημάτων VLSI

Εργαστήριο 3

Λουδάρος Ιωάννης (1067400)

Μπορείτε να δείτε την τελευταία έκδοση του Project εδώ ή σκανάροντας τον κωδικό QR που βρίσκεται στην επικεφαλίδα.

Περιγραφή Αναφοράς

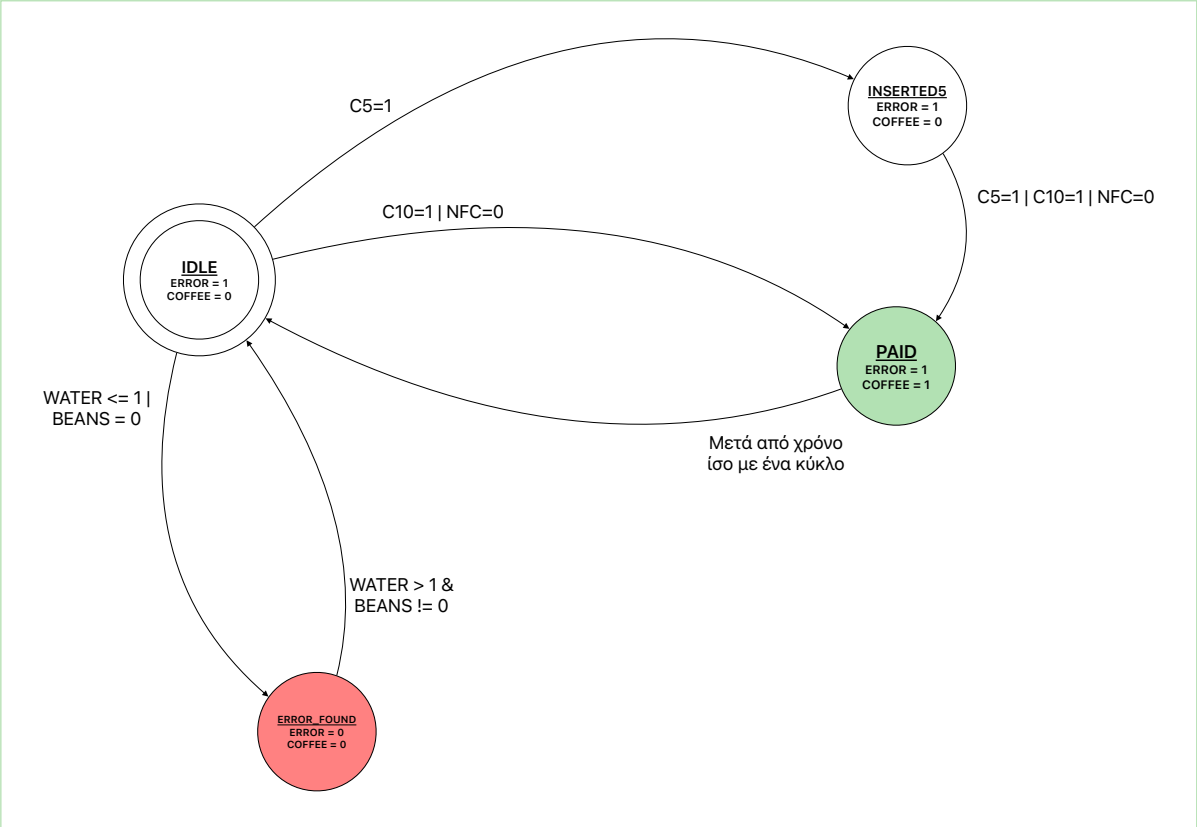
Παρακάτω παραθέτω τις απαντήσεις μου στην "3η Εργαστηριακή Άσκηση" του μαθήματος "Σχεδιασμός Συστημάτων VLSI" καθώς και σχόλια τα οποία προέκυψαν κατά την εκπόνηση του.

Περιεχόμενα

Απαντήσεις

1. Vending Machine

1.1. Το FSM του συστήματος



Παραλείπονται τα σήματα που δεν προκαλούν μεταβάσεις κατάστασης. Στο ίδιο πνεύμα, βλέπουμε ότι σε κάθε κύκλο στον κώδικα μας, η επόμενη κατάσταση από προεπιλογή είναι η τωρινή (γραμμή 19). Μόνο όταν έρχεται σήμα που οδηγεί σε μετάβαση, η κατάσταση αλλάζει.

1.2. Το κύκλωμα του vmcoffee και το αντίστοιχο testbench

```
module vmcoffee (
    input C5, C10, NFC, BEANS, rstn, clk,
    input [4:0] WATER,
    output reg COFFEE, ERROR
);

parameter [1:0] IDLE = 2'b00, INSERTED5 = 2'b01, PAID = 2'b10, ERROR_FOUND =
2'b11;
reg [1:0] state, next;

always @(posedge clk or negedge rstn) begin
    if (!rstn) state <= IDLE;
    else       state <= next;
end

always @(state or C5 or C10 or NFC or BEANS or WATER) begin
    next = state; COFFEE = 0 ; ERROR = 1;
    case (state)
        IDLE: begin
            if (C5) next = INSERTED5;
            else if (C10) next = PAID;
            else if (!NFC) next = PAID;
            else if (WATER<2 || !BEANS) next = ERROR_FOUND;
        end

        INSERTED5: begin
            if (C5) next = PAID;
            else if (C10) next = PAID;
            else if (!NFC) next = PAID;
        end

        PAID:
        begin
            next = IDLE;
            COFFEE = 1;
        end

        ERROR_FOUND: begin
            if (WATER>1 && BEANS) next=IDLE;
            ERROR = 0;
        end
    endcase

end

endmodule
```

vmcoffee.v

```
module vmcoffee_tb;

reg C5, C10, NFC, BEANS, clk, rstn ;
wire ERROR, COFFEE;
reg [4:0] WATER ;

vmcoffee dut
(.C5(C5), .C10(C10), .NFC(NFC), .BEANS(BEANS), .rstn(rstn), .clk(clk), .ERROR(
ERROR), .COFFEE(COFFEE), .WATER(WATER));

always #5 clk = ~clk;

initial begin
    $dumpfile("vmcoffee.vcd");
    $dumpvars;

    WATER = 2;
    BEANS = 1;
    C5 = 0 ;
    C10 = 0;
    NFC = 1;



    clk = 0;
    rstn = 0;
    #1 rstn = 1;
    if(dut.state==0) $display("✅ Reset Signal Working.");
    else $display("❌ Reset Signal Working.");

    #1 NFC = 0;
    #4 NFC = 1;

    if ( COFFEE == 1) $display("✅ NFC Payment Working.");
    else $display("❌ NFC Payment Working.");

    WATER = 1;

    #20 if(ERROR == 0) $display("✅ Water Sensor Working : No water.");
    else $display("❌ Water Sensor Working : No water.");

    #1 WATER = 30;

    #10 if(ERROR == 1) $display("✅ Water Sensor Working : Water Full.");
    else $display("❌ Water Sensor Working : Water Full.");

    #1 C5 = 1;
    #10 C5 = 0;

    #5 C5 = 1;
    #5 C5 = 0;

    #1 if(COFFEE == 1) $display("✅ Coin Payment Working.");
    else $display("❌ Coin Payment Working.");

    #10 BEANS = 0;
. . .
```

```
. . .

    #20 if(ERROR == 0) $display("✅ Bean Sensor Working : Out of beans.");
    else $display("❌ Bean Sensor Working : Out of beans.");

    #100 BEANS = 1;
    #10 if(ERROR == 1) $display("✅ Bean Sensor Working : Beans Full.");
    else $display("❌ Bean Sensor Working : Beans Full.");


    #150 $finish;
end



endmodule
```

vmcoffee.v (συνέχεια)

2. Digital Lock

2.1. Το κύκλωμα lockcomp

```
module lockcomp #(
    parameter size = 16,
    parameter password = 16'b0
) (
    input reset, preset,
    input [size-1:0] in,
    output reg equal
);

reg [size-1:0] stored_number = password;


always @(in or stored_number) begin
    if (in == stored_number) equal = 1;
    else equal = 0;
end

always @(posedge preset) stored_number = in;
always @(posedge reset) stored_number = password;
endmodule
```

vmcoffee.v

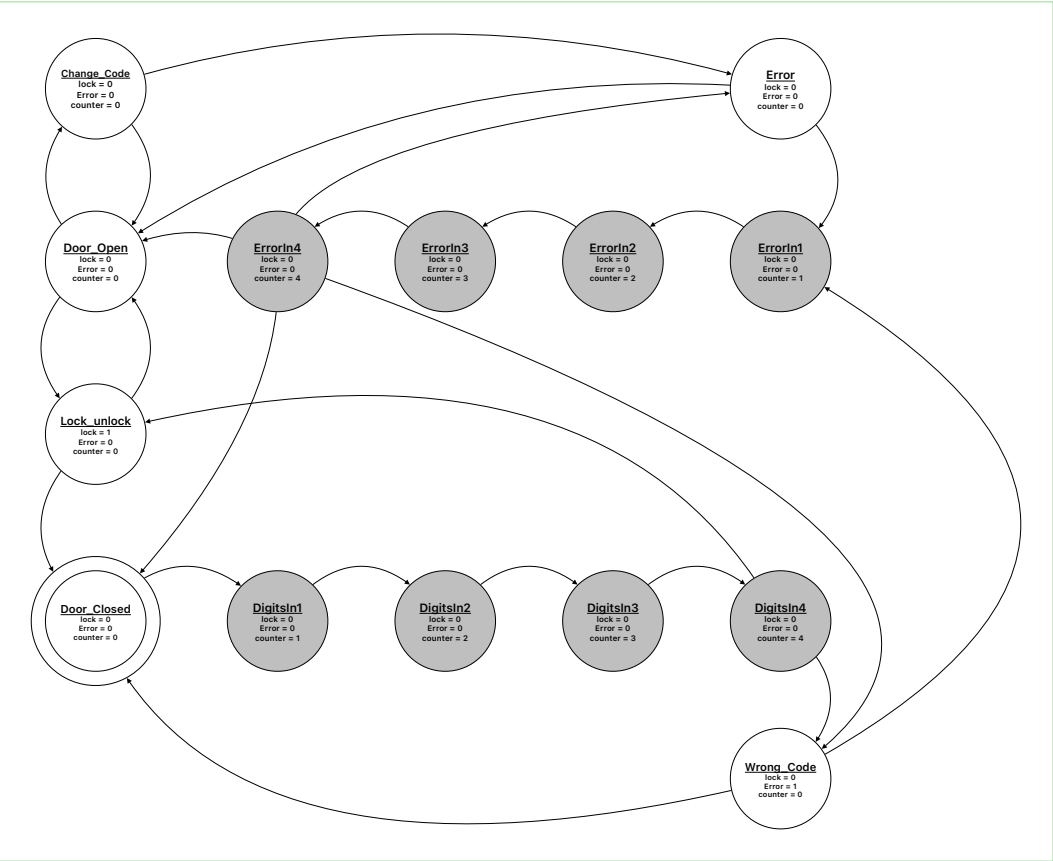
```
module lockcomp_tb;

parameter password = 16'b0;
parameter size = 16;

reg reset, preset;
reg [size-1:0] input_number;
wire equal;



reg [size-1:0] temp;

lockcomp #(.password(password)) dut
(.reset(reset), .preset(preset), .in(input_number), .equal(equal));


initial begin

    $dumpfile("lockcomp.vcd");
    $dumpvars;

    preset = 0;
    reset = 0;

    input_number = 0;
    #1
    if (equal==1) $display("✅ Default password working.");
    else $display("❌ Default password working.”); . . .
```

vmcoffee.v

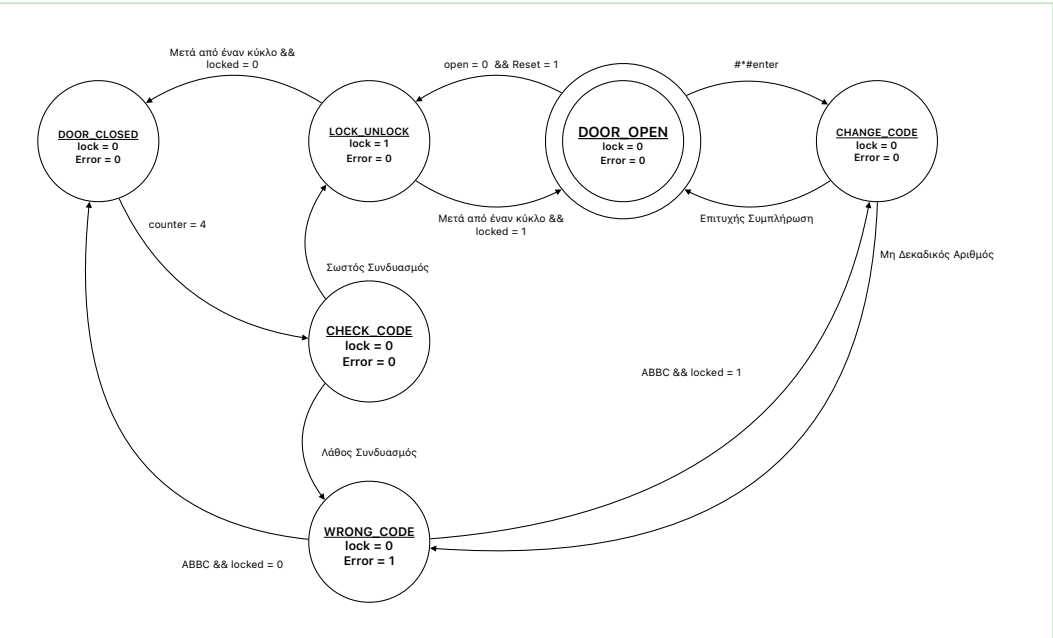
```
    #5
    temp = $random;
    input_number = temp;
    preset = 1;
    #1
    preset = 0;
    if (dut.stored_number==temp) $display("✅ Preset Working. (Number
Stored)");
    else $display("❌ Preset Working. (Number Stored)");

    #5
    input_number = $random;
    #1
    if (equal==0) $display("✅ Wrong comparison Working.");
    else $display("❌ Wrong comparison Working.");

    #5
    input_number = temp;
    #1
    if (equal==1) $display("✅ Preset Working. (Number Used)");
    else $display("❌ Preset Working. (Number Used)");

    reset=1;
    #1
    reset=0;
    #1
    input_number=16'b0;
    #1
    if (equal==1) $display("✅ Reset Working. (Correct Comparison)");
    else $display("❌ Reset Working.");

    #5
    input_number = $random;
    #10
    if (equal==0) $display("✅ Reset Working. (Wrong Comparison)");
    else $display("❌ Reset Working. (Wrong Comparison)");
    #10


    $finish;
end

endmodule
```

2.2.To FSM του clock

Οι καταστάσεις που σημειώνονται με γκρι, είναι κρυφές, υπό την έννοια ότι αποθηκεύονται στον counter και όχι στον καταχωρητή κατάστασης, όπως οι υπόλοιπες καταστάσεις.



Έτσι τις παραλείπω στο σχήμα, και αναλύω τις υπόλοιπες μεταβάσεις. Σε μια προσεκτική εξέταση, θα μπορούσε κανείς να θεωρήσει τον καταχωρητή κατάστασης και τον counter ως έναν ενιαίο καταχωρητή κατάστασης, αφού και οι δύο χρησιμεύουν (όπως φαίνεται αργότερα) ώστε το σύστημα να αποφασίσει σε ποια κατάσταση βρίσκεται.



2.3.Το κύκλωμα dlock

```
module dlock (
    input [3:0] SW16, // keyboard
    input SW1, //reset
    input SW2, //enter
    input SW3, //open check
    input rstn, //password reset
    input clk,
    output reg lock,
    output reg Error,
    output reg  [2:0] counter
);

reg comp_reset, comp_preset, locked;
wire equal;
reg [15:0] log;

reg [2:0] state, next;

parameter [2:0]
DOOR_CLOSED = 3'b000,
LOCK_UNLOCK = 3'b001,
DOOR_OPEN = 3'b010,
CHANGE_CODE = 3'b011,
WRONG_CODE = 3'b100,
CHECK_PASSWORD = 3'b101;




lockcomp #(.password({4'b0011,4'b0011,4'b0011,4'b0011})) password_holder (
    .reset(comp_reset),
    .preset(comp_preset),
    .equal(equal),
    .in(log)
    );

always @(counter) if (counter>4) counter = 0;



always @(posedge clk or negedge rstn) begin

    if (!rstn) begin
        state <= DOOR_OPEN;
        comp_reset <= 1;
        counter <= 0;
        locked <= 0;
    end

    else       state <= next;
end


always @(state, SW16, posedge SW1, posedge SW2, posedge SW3) begin
    next = state; lock = 0; Error = 0; comp_preset = 0; comp_reset = 0;

    case (state)

    DOOR_CLOSED: begin

        if (SW2) begin . . .
```

```
. . .
         log[counter*4 +:4] = SW16;
         counter = counter + 1;
        end

        if (counter == 4) begin
            next = CHECK_PASSWORD;
        end
    end

    CHECK_PASSWORD:begin
        counter = 0;
        if (equal == 1) next = LOCK_UNLOCK;
        else next = WRONG_CODE;
    end



    LOCK_UNLOCK: begin
        counter = 0;
        lock = 1;
        log = 16'bx;
        locked = ~locked;
        if(!SW3 && locked) next = DOOR_CLOSED;
        else if (!locked) next = DOOR_OPEN;
    end



    DOOR_OPEN: begin
        if (SW1 && !SW3) next = LOCK_UNLOCK;

        if (SW2) begin
            if(log[0 +: 12] == 12'b111100011111) begin
            counter = 0;
            next = CHANGE_CODE;
            end
        end

        if(!SW2 && (SW16 !== 4'bz))begin
         log[counter*4 +:4] = SW16;
         counter = counter + 1;
        end
    end



    CHANGE_CODE: begin

        if (SW2) begin
            if (
                SW16 == 4'b0001 ||  //*
                SW16 == 4'b0010 ||  //A
                SW16 == 4'b0110 ||  //B
                SW16 == 4'b0101 ||  //C
                SW16 == 4'b0100 ||  //D
                SW16 == 4'b1111 )   //#

                begin
                    next = WRONG_CODE;
                    counter = 0;
                end

                else begin
                    if(SW16 !== 4'bz) begin . . .
```

```
                        log[counter*4 +:4] = SW16;
                        counter = counter + 1;
                    end
                end
        end

        if (counter == 4) begin
                        comp_preset = 1;
                        next = DOOR_OPEN;
                    end
    end



    WRONG_CODE: begin
        Error = 1;
        if(SW16 !== 4'bz) begin
            log[counter*4 +:4] = SW16;
            counter = counter + 1;
        end

        if(log == 16'b0101011001100010) begin
            counter = 0;
            next = locked?DOOR_CLOSED:CHANGE_CODE;
        end
    end
    endcase
end




endmodule
```

vmcoffee.v

Για οικονομία χώρου, παραθέτω το testbench μέσω συνδέσμου στο GitHub.

Testbench